

Duncan Mackenzie
Kent Sharkey

Aprenda

informações essenciais
sobre tudo o que é novo
no Visual Basic .NET

Aplique

seus conhecimentos
ao mundo real

Aprenda

Visual Basic .NET



MAKRON
Books

Pearson
Education

em **21** dias

PÁGINA EM BRANCO

Aprenda Visual Basic .NET

em 21 Dias

Duncan Mackenzie

Kent Sharkey

Tradução:

Aldir José Coelho Correa da Silva

Revisão Técnica:

Marcos Jorge

Consultor Especialista na Plataforma .NET
Analista de Sistemas



São Paulo

Brasil Argentina Colômbia Costa Rica Chile Espanha
Guatemala México Peru Porto Rico Venezuela

©2003 Pearson Education do Brasil
Título Original: Sams, Teach Yourself Visual Basic .NET in 21 Days
©2002 Sams Publishing
1ª Edição em inglês Sams Teach Yourself Visual Basic .NET
in 21 Days publicada pela Pearson Education Inc.,
sob o selo Sams Publishing
Todos os direitos reservados
Diretor Editorial: José Martins Braga
Editora: Gisélia do Carmo Costa
Produtora Editorial: Marileide Gomes
Designer de Capa: Marcelo da Silva Françoze
(sobre o projeto original de Aren Howell)
Editoração Eletrônica: ERJ Composição Editorial e Artes Gráficas Ltda
Impressão: São Paulo – SP

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Mackenzie, Duncan

Aprenda Visual Basic .NET em 21 Dias / Duncan Mackenzie, Kent Sharkey; tradução: Aldir José Coelho Correa da Silva; revisão técnica: Marcos Jorge. -- São Paulo: Pearson Education do Brasil, 2003.

Título original: Sams teach yourself Visual Basic .NET in 21 days.
ISBN: 85.346.1450-4

1. BASIC (Linguagem de programação para computadores) 2. Microsoft Visual Basic 3. Microsoft Windows (Programa de computador)
I. Sharkey, Kent. II. Título.

02-4239

CDD-005-133

Índices para catálogo sistemático

1. Visual Basic.NET: Linguagem de programação:
Computadores: Processamento de dados 005.133

2003

Proibida a reprodução total ou parcial.
Os infratores serão punidos na forma da lei.
Direitos exclusivos para a língua portuguesa cedidos à
Pearson Education do Brasil,
uma empresa do grupo Pearson Education
Av. Ermano Marchetti, 1435
CEP 05038-001 – Lapa – São Paulo – SP
Tel. (11) 3613-1222 Fax: (11) 3611-0444
e-mail: vendas@pearsoned.com

Dedicatória

De Duncan Mackenzie:

Enquanto trabalhava neste livro, quase tudo em minha vida se modificou, mas em grande parte, isso não estava relacionado com ele. Agora tenho um filho, Connor, me mudei para outra região e estou com um novo trabalho na Microsoft. Gostaria de dedicar este livro a minha esposa, Laura, que sempre apoiou a iniciativa de escrevê-lo, independentemente de quantos meses ultrapassamos o prazo inicial, e a meu filho, que não tinha a mínima idéia de que este livro estava sendo escrito e mesmo assim estava muito feliz e solidário. Para concluir, tenho de agradecer a Kent, porque ele escreveu metade do livro, e não acho que eu teria sobrevivido se tivesse que fazê-lo sozinho.

De Kent Sharkey:

Em princípio, pretendia parecer inteligente e escrever um poema aqui, mas esse lado de meu cérebro está com problemas nestes últimos dias. Portanto, em vez disso, essa será uma lista bastante típica de todas as pessoas que me ajudaram neste livro. Gostaria de agradecer ao meu co-autor, Duncan. Obrigado pela oportunidade e por nunca aceitar um “não” ou “Desisto” como resposta. Obrigado a todos os editores que trabalharam no livro, principalmente Sondra e Todd. Obrigado a Squirrel por me manter são e salvo e a Cica por deixar meu laptop aquecido e, às vezes, sem atividade. Por último, mas não menos importante, obrigado a Margaret por me apoiar enquanto trabalhava nesta atividade, um novo trabalho em um novo local. Agora teremos novamente essa coisa que chamam de fim de semana!

Sobre os Autores

Duncan Mackenzie é profissional certificado MCSD, MCSE e MCT que trabalha para o grupo MSDN (<http://msdn.microsoft.com>) da Microsoft em Redmond, Washington. Ele é um programador dinâmico do Visual Basic .NET que possui o aborrecido hábito de escrever artigos ocasionais. Duncan também atua como instrutor de Visual Basic e ministra muitos cursos, que abrangem da programação intermediária à avançada do VB. Escreveu vários livros e colaborou em tantos outros sobre as tecnologias da Microsoft. É ainda orador em muitas conferências que enfocam o desenvolvimento da Microsoft.

Kent Sharkey é MCSD, MCSE, MCT e MCP+SB. No momento trabalha na Microsoft como Instrutor Técnico no Grupo de Soluções .NET, em que sua ocupação atual é o .NET Framework e o Visual Studio .NET. Antes de entrar para a Microsoft, Kent já possuía anos de experiência como instrutor e consultor, concentrando-se em projetos e desenvolvimento de aplicativos com o uso do Visual Basic. Ele escreveu e colaborou em diversos livros, incluindo sobre MCSD e Visual Basic. Faz exposições regularmente em conferências de desenvolvedores dando ênfase ao desenvolvimento da Microsoft.

Diga-nos o Que Você Acha!

Como leitor deste livro, você é nosso crítico e colaborador mais importante. Valorizamos sua opinião e queremos saber o que estamos fazendo corretamente, o que poderíamos fazer melhor, sobre quais áreas você gostaria de nos ver publicando e qualquer outra sugestão importante que deseje passar para nós.

Receberemos com satisfação seus comentários. Você pode enviar um fax, e-mail ou escrever diretamente, para que possamos saber do que gostou ou não neste livro — assim como o que podemos fazer para tornar nossos livros melhores.

Por favor, entenda que não podemos ajudá-lo com problemas técnicos relacionados ao tópico deste livro e que, devido ao grande volume de correspondência que recebemos, talvez não possamos responder a todas as mensagens.

Quando você escrever, certifique-se de incluir o título e o autor deste livro, assim como seu nome e número de telefone ou fax. Examinaremos seus comentários cuidadosamente e os compartilharemos com os autores e editores que trabalharam no livro.

Fax: (11) 3611 9686

Fone: (11) 3613 1213

Endereço Eletrônico: clientes@makron.com.br e livros_informatica@pearsoned.com.br

Endereço Postal: Pearson Education do Brasil
Rua Emílio Goeldi, 747 - Lapa
São Paulo – SP/ 05065-110

Aprenda Visual Basic.NET em 21 Dias

DIA 1 Introdução ao Microsoft .NET. Conheça a evolução do Visual Basic .NET até o que ele é hoje.	DIA 2 Aprenda sobre o ambiente de desenvolvimento do Visual Studio .NET e como você pode usá-lo para criar aplicativos do Visual Basic.	DIA 3 Aprenda a criar e usar variáveis e procedimentos em seus programas do Visual Basic .NET.	DIA 4 Aprenda a usar as funções de controle e iteração no Visual Basic .NET. Teste laços, instruções If e muito mais.	DIA 5 Aprenda sobre as opções de arquitetura disponíveis na plataforma .NET e como você pode escolher uma delas para seus sistemas.	DIA 6 Aprenda a proteger seu código de erros e como encontrá-los antes dos usuários.	DIA 7 Os objetos estão em todos os locais da plataforma .NET. Aprenda a trabalhar com eles para desenvolver sistemas por meio do Visual Basic .NET.
DIA 8 Aprenda sobre o .NET Framework, a base da plataforma .NET e do Visual Basic .NET.	DIA 9 Aprenda sobre os formulários Windows, a tecnologia da plataforma .NET que permitirá a criação de interfaces gráficas poderosas em seus aplicativos.	DIA 10 Aprenda a criar aplicativos para a World Wide Web usando o Visual Basic .NET e os formulários da Web.	DIA 11 Uma introdução aos bancos de dados e a atribuição deles em seu aplicativo. Aprenda sobre o espaço de nome System.Data da plataforma .NET.	DIA 12 Use o espaço de nome System.Data para se conectar, consultar e atualizar bancos de dados pelo seu código do Visual Basic .NET.	DIA 13 Veja como o Server Explorer torna fácil se beneficiar dos recursos avançados do Windows.	DIA 14 Dando continuidade ao tema abordado no Dia 7, aprenda como os objetos podem alterar a organização e o projeto básico de seus sistemas por meio da programação orientada a objetos.
DIA 15 Crie seus objetos completando-os com propriedades, métodos e eventos. Aprenda a gerar e usar montagens da biblioteca de classes.	DIA 16 Aprofunde-se mais na criação de aplicativos com os formulários Windows. Aprenda a gerar menus e trabalhe com controles avançados.	DIA 17 Aprenda a acessar arquivos e figuras por meio das classes do Visual Basic .NET e do .NET Framework.	DIA 18 Quando a codificação estiver concluída, o trabalho ainda não terá acabado. Aprenda a documentar seu sistema e outras etapas de finalização no desenvolvimento de aplicativos.	DIA 19 O aplicativo não estará concluído até que os usuários o tenham instalado, e você precisa desenvolver o programa de instalação. Aprenda a criar esses programas por meio da tecnologia do Windows Installer.	DIA 20 Aprenda sobre a XML e como você pode usá-la em seus programas.	DIA 21 Aprenda sobre o SOAP e os serviços da Web. Veja como eles possibilitam o funcionamento de seu aplicativo na rede ou na Internet.

Sumário

Introdução	XXI
SEMANA 1 Visão Geral	1
Dia 1 Bem-Vindo ao Visual Basic .NET	3
Compreendendo a Programação de Computadores	3
A Função dos Sistemas Operacionais	4
O Papel das Linguagens de Programação	5
Por Que Escrever Programas de Computador?	8
Um Breve Histórico sobre o Visual Basic	10
O Que É .NET?	13
Servidores .NET	14
.NET Framework	14
Serviços .NET	15
Dispositivos .NET	16
Desenvolvendo seu Primeiro Aplicativo no Visual Basic .NET	17
Preparando-se para Codificar	17
Onde Está meu IDE?	21
Uma Tarefa Simples	21
Escrevendo o Código	22
Resumo	27
P&R	27
Workshop	28
Teste	28
Exercícios	28
Dia 2 Trabalhando com o Visual Basic .NET	29
O IDE do Visual Studio	29
Iniciando	30
A Janela Principal do IDE do Visual Studio	32
Soluções e Projetos	50
Arquivos	52
Criando Nosso Primeiro Aplicativo Windows	53
Crie o Projeto	54
Desenvolva a Interface com o Usuário	54
Executando o Projeto	55
Construindo o Projeto	56
Adicionando Seu Próprio Código	58
Resumo	60

P&R	60
Workshop	61
Teste	61
Exercícios	61
Dia 3 Introdução à Programação com o Visual Basic .NET	63
Variáveis e Atribuição	64
O Que É uma Variável?	64
Tipos de Variáveis Disponíveis	64
Variáveis Simples	65
Declarando Variáveis	69
Arrays	70
Atribuição	72
Constantes	72
Algumas Sugestões para os Padrões de Nomeação	73
Cálculos Simples	74
Usando Operadores	75
Funções Internas	75
Escrevendo Suas Próprias Rotinas	80
Sub-Rotinas	80
Funções	81
Escopo	81
Exemplo de Aplicativo: Calculando um Valor Futuro	83
Resumo	88
P&R	89
Workshop	89
Teste	90
Exercícios	90
Dia 4 Controlando o Fluxo dos Programas	91
Tomando Decisões com as Instruções de Controle	91
A Instrução If	92
Estendendo a Instrução If	95
Instruções If em Sequência na Mesma Linha	99
Expressões e Lógica Booleana	100
Operadores de Comparação	100
Operadores Lógicos	101
Avaliação Abreviada	103
Lidando com Múltiplas Possibilidades: A Instrução Select Case	104
Laços	105
For...Next	106
A Variável do Contador	106
Especificando o Valor do Incremento com o Uso de Step	108
While...End While	109

Laço Do	111
Condições de Saída	113
Laços Infinitos	114
Implicações sobre o Desempenho	115
Aplicativos Que Farão Uso de Seu Conhecimento Recém-Adquirido	116
Leitura de um Arquivo	117
Um Jogo Simples	119
Evitando Laços Complexos por Meio da Recursão	121
Resumo	123
P&R	123
Workshop	124
Teste	124
Exercícios	125
Dia 5 Arquitetura dos Aplicativos na Plataforma .NET	127
O Que É a Arquitetura do Aplicativo?	127
Função do Arquiteto de Softwares	128
Que Partes de um Sistema São Consideradas Arquitetura do Aplicativo?	129
Arquiteturas Viáveis na Plataforma .NET	133
Os Três Elementos de Qualquer Aplicativo	133
Quantas Camadas?	134
Windows DNA	134
Onde a Plataforma .NET Se Encaixa?	135
Escolhendo uma Tecnologia de Cliente	136
Decidindo Que Arquitetura Usar	138
Fatores Essenciais Que Influenciarão Sua Decisão	139
Exemplos de Cenários	141
Resumo	145
P&R	145
Workshop	146
Teste	146
Dia 6 O Que Fazer Quando Programas Bons Apresentam Problemas e para Se Certificar de Que Isso Não Aconteça	147
Tratamento de Exceções Estruturadas	148
O Que É o Tratamento de Exceções Estruturadas?	148
Erros e Exceções	148
O Bloco Try	149
A Seção Catch	149
Aninhando Blocos Try...End Try	155
A Seção Finally	156
Lançando Exceções	157
Depurando	157
A Fonte dos Erros	158

Aprendendo a Depurar com a Prática	159
Os Modos na Vida de um Programa	162
Percorrendo Seu Código	165
Examinando as Variáveis	168
Outras Ferramentas para Depuração	172
Resumo	173
P&R	173
Workshop	174
Teste	174
Exercícios	174
Dia 7 Trabalhando com Objetos	177
Para Começar: O Que É um Objeto?	177
Classes e Instâncias	178
Referências	178
Passando o Conceito para o Código	179
Propriedades	180
Propriedades ReadOnly e WriteOnly	181
Criando a Instância de um Objeto	182
Encapsulando Códigos em Suas Classes	183
Tópicos Avançados	187
Sobreposição	187
Herança	189
A Base de Todas as Classes Básicas	194
Construtores	195
Espaços de Nome	197
Membros e Objetos Compartilhados	199
Resumo	199
P&R	200
Workshop	200
Teste	201
Exercícios	201
SEMANA 1 Revisão	202
SEMANA 2 Visão Geral	203
Dia 8 Introdução ao .NET Framework	205
O Que É o .NET Framework?	205
Classes Importantes do .NET Framework	206
Console	206
Resultados Mais Simples	207
Environment (Ambiente)	212
Random	213

Math	214
Classes de Conjuntos no .NET Framework	215
Encontrando o Que Precisa no .NET Framework	219
As Regras da Busca	219
A Saga pela Classe Perfeita	220
Resumo	223
P&R	224
Workshop	224
Teste	224
Exercícios	224
Dia 9 Desenvolvendo uma Interface com o Usuário com os Formulários Windows	225
Visão Geral dos Formulários Windows	225
Criando um Aplicativo com Formulários Windows	227
Configurando o Projeto	227
Adicionando Controles ao Formulário	228
Nomeando Seus Controles	229
Manipulação de Eventos	229
Criando Vários Manipuladores de Eventos para um Único Evento	231
Encontrando Objetos e Eventos por Meio do Editor de Códigos	232
Múltiplos Eventos com um Manipulador	233
Mais Informações sobre os Controles	233
Criando Grupos de Botões de Opção	234
Adicionando uma Caixa de Seleção ao Exemplo de Filer	236
Validação de Entradas	238
Usando a Classe MessageBox	241
Parâmetros	242
Obtendo Resultados	243
Controles Ocultos	246
Timer	247
NotifyIcon	248
ErrorProvider	249
Controles das Caixas de Diálogo	250
Construindo Suas Caixas de Diálogo	254
Criando a Caixa de Diálogo	255
Configurando o Resultado da Caixa de Diálogo	256
Exibindo a Caixa de Diálogo	257
Resumo	259
P&R	259
Workshop	260
Teste	260
Exercícios	260

Dia 10	Construindo a Interface com o Usuário com os Formulários da Web	263
	O Modelo de Programação da Web ·····	263
	ASP.NET ·····	265
	Como a Criação de Programas com Base na Web Difere da de Programas com Base no Windows ·····	266
	Usando os Controles-padrão dos Formulários da Web ·····	268
	Usando os Controles Avançados dos Formulários da Web ·····	277
	Usando os Controles Validator ·····	279
	Resumo ·····	283
	P&R ·····	283
	Workshop ·····	284
	Teste ·····	284
	Exercício ·····	284
Dia 11	Introdução aos Bancos de Dados	285
	Um Banco de Dados É a Solução para Todos os Problemas Cotidianos ·····	285
	A Decisão É Tomada ·····	286
	A Passagem para um Banco de Dados Real ·····	287
	Uma Introdução à SQL ·····	288
	Recuperando Registros com a Instrução SELECT ·····	288
	Adicionando Novos Registros ·····	290
	Alterando Registros ·····	291
	Removendo Registros Indesejados ·····	292
	Para Onde Ir a Partir Daqui Abordando a SQL ·····	292
	Problemas Comuns dos Bancos de Dados e Suas Soluções ·····	293
	Inconsistências de Atualização ·····	293
	Campos Multivalorados ·····	297
	Associações: Consultando Várias Tabelas de uma Só Vez ·····	298
	Relacionamentos Muitos-para-Muitos ·····	300
	Integridade Referencial ·····	300
	Criando Chaves Primárias ·····	301
	Criando o Banco de Dados de Exemplo ·····	304
	Access 2000 ou Access 2002 ·····	304
	MSDE e SQL Server 2000 ·····	305
	Testando a Configuração com System.Data ·····	305
	Resumo ·····	307
	P&R ·····	308
	Workshop ·····	308
	Teste ·····	309
	Exercícios ·····	309
Dia 12	Acessando Dados com a Plataforma .NET	311
	Uma Visão Geral do Acesso aos Dados na Plataforma .NET ·····	311
	O ADO e o OLEDB ·····	312

ADO.NET	312
Tarefas-padrão dos Bancos de Dados	314
Conectando-se ao Banco de Dados	314
Executando uma Instrução SQL	319
Recuperando Dados	321
Trabalhando com Data Sets	323
Inserindo Dados em um DataSet	323
Navegando pelos Dados	326
Editando Dados (Adicionar, Editar e Excluir)	329
Editando Registros	331
Atualizando o Banco de Dados	334
Trabalhando com Múltiplas Tabelas	346
Visualizações	348
Vinculação de Dados	350
Vinculação de Dados com os Formulários Windows	351
Resumo	354
P&R	355
Workshop	355
Teste	355
Exercícios	356
Dia 13 Usando o Server Explorer	357
O Que É o Server Explorer	357
O Que É um Serviço?	359
Examinando os Serviços	359
Caminho a Ser Percorrido para Estabelecer uma Conexão com o Banco de Dados	363
Trabalhando com os Serviços	364
Visualizando os Serviços	365
Conectando-se com Outro Servidor	366
Escrevendo Programas Que Usam os Serviços	366
Escrevendo Códigos de Acesso a Dados com o Server Explorer	366
Resumo	379
P&R	380
Workshop	380
Teste	380
Exercícios	380
Dia 14 Introdução à Programação Orientada a Objetos	381
Visão Geral da Programação Orientada a Objetos	381
Comparando a Programação Orientada a Objetos com a Linear	382
Usando os Objetos na Organização do Código	384
Conceitos Importantes na POO	385
Classes, Objetos e Instâncias	385

Propriedades ·····	387
Métodos ·····	388
Herança ·····	389
Construtores ·····	393
Projetando um Aplicativo com o Uso da POO ·····	396
Identificando os Objetos ·····	397
Determinando as Propriedades e Métodos ·····	398
Modelando Seus Objetos ·····	399
Resumo ·····	400
P&R ·····	400
Workshop ·····	400
Teste ·····	401
Exercícios ·····	401
SEMANA 2 Revisão	403
SEMANA 3 Visão Geral	405
Dia 15 Criando Objetos no Visual Basic .NET	407
Criando Objetos ·····	407
Declarando uma Classe Nova no Visual Basic .NET ·····	408
Herança ·····	408
Adicionando Propriedades ·····	410
Usando Rotinas de Propriedade para Validar Dados ·····	413
Criando Métodos ·····	417
Adicionando Eventos ·····	423
Definindo e Usando Interfaces ·····	425
Usando os Objetos Que Você Criou ·····	434
Espaços de Nome ·····	434
Criando e Usando uma DLL da Biblioteca ·····	436
Resumo ·····	438
P&R ·····	438
Workshop ·····	438
Teste ·····	438
Exercícios ·····	439
Dia 16 Formulários Windows Avançados	441
Menus ·····	441
Adicionando um Menu a um Formulário ·····	441
Os Teclados e os Menus ·····	444
Adicionando Código ·····	445
Algumas Sugestões ·····	448
Programas de Interface de Documentos Múltiplos ·····	449
O Que É uma Interface de Documentos Múltiplos? ·····	449

Adicionando o Formulário-pai	450
A MDI e os Menus	451
Controles Avançados dos Formulários Windows	458
TreeView	459
ListView	462
Controles Splitter	463
Resumo	470
P&R	471
Workshop	471
Teste	471
Exercícios	472
Dia 17 Usando o .NET Framework	473
Fluxos e Arquivos	473
O Que É um Stream?	474
Arquivos e Diretórios	474
Lendo um Arquivo de Texto	476
Gravando em um Arquivo de Texto	478
Desenhando com as Classes de Figuras	492
Examinando as Classes de Figuras	492
Onde Posso Desenhar?	498
Desenhando Formas	503
Salvando Figuras	506
Resumo	509
P&R	509
Workshop	510
Teste	510
Exercícios	510
Dia 18 Retoques Finais	511
Documentando Seu Aplicativo	511
Crie Soluções Mais Simples	512
Evitando Conjecturas	515
Não comente o Óbvio, Só o Que For Confuso	516
Documente o Sistema, e Não Apenas o Seu Código	516
Melhores Práticas e Padrões de Codificação	518
Nomeação de Variáveis, Controles e Objetos	519
Blocos de Comentário	521
Usando o Controle do Código-fonte	522
Extraíndo o Código	522
Armazenando o Código	526
Visualizando e Retornando Suas Alterações	528
Considerações sobre Segurança no Uso do Visual Source Safe	529
Resumo	529

P&R	530
Workshop	530
Teste	530
Dia 19 Implantando Seu Aplicativo	531
Introdução à Implantação	531
Criando uma Instalação Simples	533
Crie o Projeto e a Solução	533
Arquivos de Configuração	540
Implantações de Múltiplos Projetos	542
Resumo	545
P&R	545
Workshop	546
Teste	546
Exercícios	546
Dia 20 Introdução à XML	547
O Que É XML?	547
Elementos	551
Atributos	552
Esquemas	553
Trabalhando com a XML	556
O Document Object Model	556
Objetos de Leitura e de Gravação	559
Lendo XML	561
Gravando XML	566
Resumo	569
P&R	569
Workshop	571
Teste	571
Exercícios	571
Dia 21 Criando Serviços Web com o Visual Basic .NET	573
O Que É um Serviço Web?	573
O Simple Object Access Protocol	575
O Protocolo	575
Web Service Description Language (WSDL)	576
Discovery	579
Criando um Serviço Web Simples	580
Criando o Projeto	581
Adicionando o Código	584
Compilando o Serviço Web	585
Criando um Serviço Web Cliente	587
Criando o Projeto	588
Adicionando o Código	589

Um Serviço Web Mais Complexo	592
Criando o Serviço	592
Testando o Serviço Web	596
Criando o Cliente	597
Adicionando o Código	600
Resumo	604
P&R	605
Workshop	605
Teste	605
Exercícios	605

SEMANA 3 Resumo 606

Apêndice A – Respostas dos Testes/Exercícios 609

Respostas do Dia 1	609
Teste	609
Exercícios	610
Respostas do Dia 2	611
Teste	611
Exercício	611
Respostas do Dia 3	611
Teste	611
Exercícios	612
Respostas do Dia 4	613
Teste	613
Exercício	614
Respostas do Dia 5	615
Teste	615
Respostas do Dia 6	615
Teste	615
Exercícios	615
Respostas do Dia 7	617
Teste	617
Exercícios	617
Respostas do Dia 8	619
Teste	619
Exercícios	620
Respostas do Dia 9	621
Teste	621
Exercícios	622
Respostas do Dia 10	623
Teste	623
Exercícios	623

Respostas do Dia 11	627
Teste	627
Exercícios	628
Respostas do Dia 12	629
Teste	629
Exercícios	630
Respostas do Dia 13	630
Teste	630
Respostas do Dia 14	630
Teste	630
Exercícios	630
Respostas do Dia 15	631
Teste	631
Exercícios	631
Respostas do Dia 16	633
Teste	633
Exercícios	633
Respostas do Dia 17	638
Teste	638
Exercícios	638
Respostas do Dia 18	639
Teste	639
Respostas do Dia 19	640
Teste	640
Exercícios	640
Respostas do Dia 20	641
Teste	641
Exercícios	641
Respostas do Dia 21	644
Teste	644
Índice	645

Introdução

Bem-vindo a uma lição estimulante e informativa de 21 dias sobre o Visual Basic .NET, a mais nova encarnação da linguagem de programação mais popular do mundo. Este livro foi planejado para fornecer a você uma introdução ao .NET Framework e ao Visual Basic .NET, e fazê-lo começar a programar aplicativos reais o mais rápido possível.

Visão Geral

O .NET Framework é o conjunto básico de conceitos e tecnologia subjacente ao mais novo grupo de ferramentas de desenvolvimento da Microsoft e constituirá a base para a nova série de servidores, aplicativos e serviços com base na Web implantados ao redor do mundo. Como já é de se esperar, a plataforma .NET não é algo que possa ser completamente discutido em uma hora ou mesmo em 21 dias, portanto, este livro terá uma abordagem mais específica. Você precisará se tornar produtivo rapidamente e isso é o que conseguirá nos próximos 21 dias enquanto ler este livro e trabalhar com os exercícios dele. Pelo fornecimento dos elementos básicos da linguagem do Visual Basic .NET e uma explicação suficiente do próprio .NET Framework, você estará pronto para já começar a programar e preparado para continuar aprendendo ao progredir.

Desde o início, é importante observar um fato essencial sobre a plataforma .NET: a linguagem que você usa é menos relevante do que foi no passado. O Framework (o conjunto de tecnologia em que toda a plataforma .NET está baseada) é a própria plataforma .NET e também pode ser acessado sem problemas por qualquer linguagem .NET (incluindo o Visual Basic, C#, C++ e outras). Essas são notícias excelentes tanto para os desenvolvedores novos do Visual Basic quanto para os experientes. O Visual Basic não é mais uma linguagem de segunda classe, com certos recursos avançados do sistema operacional restritos apenas aos programadores que usam C++. Agora, qualquer sistema, independentemente do tipo, pode ser criado no Visual Basic.

Essa independência também significa que você (ou sua empresa) está livre para escolher qualquer linguagem com a qual queira trabalhar. Dada essa opção, muitas pessoas, incluindo os autores deste livro, irão adotar o Visual Basic .NET. Sua sintaxe e estilo fáceis de usar o tornaram a linguagem de programação mais popular do mundo, mesmo com as limitações que a versão anterior apresentava. Agora, com a plataforma .NET, nada irá impedi-la de ser usada em qualquer projeto de todos os tipos de empresa.

Se você estiver interessado em aprender mais do que apenas o Visual Basic .NET – talvez a C# possa ser bem interessante –, continuará tendo um bom ponto de partida. Este livro vai fornecer uma abordagem de como usar o .NET Framework, conhecimento que pode ser transferido facilmente para qualquer linguagem .NET.

Este livro foi planejado como uma série de lições, cada uma representando algum conceito importante no desenvolvimento de aplicativos (como o acesso a bancos de dados) ou um trecho es-

sencial com conhecimento para tornar você produtivo (o uso do IDE, por exemplo). Essas lições podem ser examinadas em qualquer ordem, mas se você for novo na atividade de programação em geral, então, será mais vantajoso seguir a partir do início, na primeira vez que o livro for usado. A maioria das lições, mesmo a primeira, inclui algum exemplo de código e exercícios para os quais será necessária a elaboração de mais um pouco de codificação. Para aproveitar ao máximo o tempo gasto no livro, tente realizar todos esses exemplos e exercícios. Nada aumentará sua compreensão desses tópicos mais rapidamente do que colocar a mão na massa e codificar.

Quem Deve Ler Este Livro?

Embora o objetivo principal deste livro sejam as pessoas iniciantes na atividade de programação em geral e no Visual Basic .NET especificamente, ele será útil a uma variedade maior de leitores. Se você já for um programador do Visual Basic, então, procure nos diversos tópicos do livro algumas das ótimas explicações e exemplos que envolvam os novos recursos. Será possível perceber que há muitas diferenças no universo do Visual Basic 6.0, e o nível de abordagem fornecido neste livro deve ajudá-lo a fazer a transição.

Se você já for um programador experiente que não conhece o Visual Basic, poderá ficar tentado a saltar ou ler de modo superficial grande parte das primeiras lições. Independentemente de seu nível de habilidade, será bom examinar com detalhes a introdução (Dia 1) para compreender o conceito geral da plataforma .NET e a abordagem do IDE (Dia 2) para ganhar velocidade ao trabalhar com o Visual Studio .NET. Em seguida, poderá passar pelo resto do livro com seu próprio ritmo, verificando por alto algumas das seções que abordam conceitos gerais de programação e se dedicando aos capítulos que explicam como o Visual Basic .NET realiza tarefas avançadas, como a criação de objetos, o acesso a bancos de dados e a geração de aplicativos na Internet.

O Que Você Aprenderá

Este livro ensinará a você como criar vários tipos diferentes de aplicativos usando o Visual Basic .NET, incluindo tanto os cliente/servidor quanto aqueles com base na Web. Quando passar pelos tópicos sobre o Visual Basic .NET, também aprenderá muito sobre o .NET Framework e conhecerá alguns servidores .NET, inclusive o SQL Server e o Internet Information Services.

No que diz respeito ao projeto e à arquitetura, você aprenderá sobre os recursos orientados a objetos do Visual Basic .NET, como, por exemplo, a criação de classes e o uso de objetos, além dos princípios básicos da herança, sobreposição, anulação e outras funcionalidades avançadas fornecidas por esse tipo de programação.

Na etapa de desenvolvimento que envolve a interface ou saída, você aprenderá a criar aplicativos 'Windows' por meio dos novos recursos de formulários do Visual Basic .NET, a construir interfaces com base em páginas da Web usando formulários da Web e a gerar um serviço simples disponível na Web que poderá ser utilizado por qualquer linguagem ou plataforma de programação capaz de acessar a rede e compreender a XML.

Nosso objetivo é prepará-lo nos próximos 21 dias para que você possa criar muitos de seus próprios aplicativos simples no Visual Basic .NET e trabalhar como parte da equipe que está desenvolvendo um grande aplicativo Windows ou com base na Web.

O Que Não Abordaremos

Apesar da discussão anterior sobre a independência da linguagem e a importância das tecnologias subjacentes, este é um livro que tem como prioridade o Visual Basic .NET. Como tal, não abordaremos a C# ou qualquer outra linguagem .NET que não seja o Visual Basic. Mencionaremos a utilização de bancos de dados, como o SQL Server, mas você não precisará recorrer a livros sobre ele para obter detalhes completos sobre como configurar e gerenciar seu servidor de banco de dados.

Também não pretendemos abordar todo o Visual Basic .NET neste livro. Essa é uma linguagem portentosa, com muitos recursos. Apenas para listar todos eles e a sintaxe usada para empregá-los, precisaríamos facilmente de um livro deste tamanho. Discutiremos o Visual Basic .NET com detalhes suficientes para que você se torne produtivo ao usá-lo e seja fornecido o conhecimento necessário para que comece a projetar aplicativos que se beneficiem dos novos recursos da plataforma .NET.

Requisitos

Já que se trata de um livro sobre o Visual Basic .NET, o requisito mais premente é um sistema que execute o Visual Studio .NET, e que tenha no mínimo as especificações básicas a seguir:

- **Sistema operacional** Windows XP Professional, Windows 2000 (Datacenter Server, Advanced Server, Server ou Professional) ou Windows NT 4.0 Server. O Visual Studio se encarregará da instalação de qualquer pacote de serviços, arquivos atualizados de acesso a dados e da versão 6 do Internet Explorer, todos sendo requisitos para o Visual Studio .NET.
- **Hardware** Pentium II com 450 Mhz ou equivalente, 128 MB de RAM, placa de vídeo com capacidade para resolução de 800 x 600 pixels, 256 cores e no mínimo 1 GB de espaço em disco rígido. Uma unidade de CD-ROM é necessária para instalação, mas você pode colocar seus CDs em outra máquina e instalar por meio de uma conexão LAN.

Além do Visual Studio, os exemplos deste livro requerem a possibilidade de acesso a um servidor Web que esteja na mesma máquina do Visual Studio ou tenha o SDK do .NET Framework instalado nela. O melhor a fazer é se certificar de que sua máquina principal de desenvolvimento tenha um servidor Web em execução nela, como no Windows 2000 ou no Windows NT. Outro requisito básico usado pelos exemplos na última metade deste livro é o acesso a um banco de dados, especificamente o SQL Server 7.0 ou 2000. Se você não tiver o SQL Server, então poderá usar o Microsoft Data Engine (MSDE), que funciona de maneira semelhante ao SQL Server completo. Pode ser usado um banco de dados Access, se for esse o seu, mas será preciso alterar alguns dos exemplos nas lições de acesso a bancos de dados para fazê-los funcionar sem o SQL Server.

Habilidades Que Você Precisa Ter

Este é um livro destinado a programadores iniciantes, mas algumas habilidades básicas são necessárias. Espera-se que você esteja familiarizado com o uso de computadores com base em Windows, incluindo o sistema operacional que estiver instalado para executar a plataforma .NET. Copiar arquivos, imprimir, abrir arquivos no Bloco de notas e habilidades de edição básica de texto (recortar, copiar, colar) são todas necessárias, e não serão explicadas nas lições. Saber como conectar sua máquina à Internet e navegar em sites da Web também são habilidades exigidas.

Além desse conhecimento básico em computadores, você não precisa ser um programador ou saber como construir bancos de dados. Qualquer conhecimento efetivo nessas áreas provavelmente será útil, mas nossos exemplos e explicações foram elaborados para serem claros mesmo para alguém que nunca tenha tentado codificar antes.

O Site da Web

Este livro possui um site da Web associado a ele no endereço <http://www.makron.com.br>. Localize a página do livro no site, digitando seu título na caixa Pesquisa. Ao entrar na página do livro, procure o link MATERIAL COMPLEMENTAR. Você pode fazer o download de todos os códigos do livro e alguns links ou materiais complementares que achamos ser úteis para o leitor compreender o Visual Studio .NET. É claro que é totalmente possível usar este livro e acompanhar todos os exemplos, sem ser preciso visitar o site, mas em alguns dos exemplos mais longos, pode ser melhor economizar algum esforço em digitação fazendo o download dos arquivos que contêm os códigos.

Layout dos Capítulos

Vinte e um dias é um período longo, portanto, além de dividir as lições em dias, organizamos o livro em seções com duração de três semanas (ou partes). Na primeira parte, nos dedicaremos a introduzi-lo nos conceitos gerais de programação e da plataforma .NET, e abordar a sintaxe e técnicas básicas para a criação de programas com o Visual Basic .NET. Na segunda parte nos aprofundaremos no .NET Framework para fornecer uma compreensão mais detalhada dessa base essencial de toda a programação e também discutiremos os fundamentos da criação de programas reais, como a maneira de gerar uma interface com o usuário (com formulários Windows e da Web) e como trabalhar com bancos de dados. A última parte introduzirá alguns dos tópicos mais avançados da programação .NET, incluindo a criação de seus próprios objetos, a implantação de seu aplicativo nos computadores de outras pessoas e o trabalho com a XML e os serviços da Web. Como falamos anteriormente, tente percorrer essas lições em ordem, mas fique à vontade para saltar adiante se houver um tópico sobre o qual queira muito ler mais no momento que desejar.

Retorno

Trabalhamos duro para tornar este livro uma ferramenta útil de aprendizado da plataforma .NET e uma aquisição valiosa para sua biblioteca de desenvolvimento. Se você achar que deveríamos ter gasto mais ou menos tempo em algum tópico específico ou tiver sugestões para melhorar o livro, faça contato conosco (clientes@makron.com.br). Tentaremos incorporá-las em livros futuros e, se for o caso, na próxima revisão deste mesmo volume.

Convenções Usadas Neste Livro

Este livro usa várias convenções para ajudá-lo a priorizar e recorrer às informações que ele contém:



As notas fornecem informações úteis em destaque, que você pode ler imediatamente ou retornar a elas sem perder o fluxo do tópico que estiver examinando.



As dicas realçam as informações que podem tornar sua programação no VB mais eficaz.



Os alertas de cuidado concentram sua atenção em problemas ou efeitos colaterais que podem ocorrer em situações específicas.

Faça

Os quadros Faça/Não Faça enfatizam práticas boas que você deve adotar ...

Não Faça

... e ruins que devem ser evitadas.

NOVO TERMO

Os ícones Novo Termo assinalam locais onde uma nova terminologia foi usada e definida pela primeira vez. Essa terminologia aparece destacada com uma fonte em itálico.

CÓDIGO

Os ícones Código são usados em exemplos de código que o usuário deve inserir.

ANÁLISE

Os ícones Análise apontam para discussões com explicações sobre os exemplos de códigos.

O código é apresentado em fonte monoespaçada.

PÁGINA EM BRANCO

SEMANA 1

Visão Geral

Durante esta parte, você aprenderá vários tópicos essenciais:

- O Visual Basic .NET e o conceito de programação (Dia 1).
- Configuração e uso do Visual Studio .NET para começar a escrever programas (Dia 2).
- A sintaxe e as técnicas de programação do Visual Basic para tipos de dados, procedimentos e variáveis (Dia 3); o controle do fluxo dos programas – laços e instruções condicionais (Dia 4); e o tratamento de erros (Dia 6).
- Organização e projeto de uma solução no Visual Basic .NET (Dia 5).
- Conceitos importantes relacionados ao desenvolvimento orientado a objetos com os quais você trabalhará por todo o livro (Dia 7).

Esta parte é importante. Ela define os fundamentos do conhecimento básico nos quais todos os outros conceitos de programação do Visual Basic .NET se basearão. O Dia 1 preparará você para começar a explorar e aprender o Visual Basic, por meio da introdução dos conceitos fundamentais de programação e da plataforma .NET, junto com um breve histórico do próprio Visual Basic. No Dia 2, aprenderemos a usar o ambiente de desenvolvimento com todos os recursos do Visual Studio para criar projetos.

Nos Dias 3 e 4, você irá programar com muitas informações sobre a sintaxe e conceitos (incluindo variáveis, laços, instruções `if` e mais) que farão parte de todos os programas do Visual Basic escritos desse ponto em diante.

No Dia 5, você conhecerá todos os tipos diferentes de projeto que poderá criar no Visual Basic .NET e aprenderá a adaptá-los à arquitetura geral do sistema. Para concluir, os Dias 6 e 7 retornam ao mundo prático da codificação em que aprenderemos a lidar com erros e a usar objetos em programas.

Esta parte fornecerá as informações de que você precisa para prosseguir com a leitura do livro, provendo todos os detalhes sobre os quais os outros capítulos se desenvolverão.

1

2

3

4

5

6

7

PÁGINA EM BRANCO

SEMANA 1

DIA 1

Bem-Vindo ao Visual Basic .NET

Hoje, o introduzirei no universo da programação do Visual Basic respondendo às perguntas a seguir:

- O que é programação e por que seria interessante aprendê-la?
- Como o Visual Basic se encaixa em tudo isso?
- O que é .NET?

Nos dias de hoje, o computador já é coisa trivial, e muitas pessoas que conheço passam o dia inteiro trabalhando com eles. Mas, mesmo assim, a pergunta mais comum é: “O que um programador de computadores faz?”. Essa pergunta tem sido feita tantas vezes, que na lição de hoje, usarei um pouco do tempo discorrendo sobre o que é programar e por que você gostaria de fazer isso.

Compreendendo a Programação de Computadores

Embora falemos com frequência em computadores no que diz respeito ao hardware (é comum ouvir comentários como “Tenho um Pentium III com 600 MHz e 256 MB de RAM”, por exemplo), só isso não é o bastante. A CPU (Central Processing Unit ou unidade central de processamento do computador), por exemplo, pode realizar muitas tarefas importantes como cálculos matemáticos ou a transferência de dados entre várias partes do sistema. Sozinho, no entanto, não é capaz nem mesmo de ler um arquivo no disco rígido. Um *programa de computador* é um con-

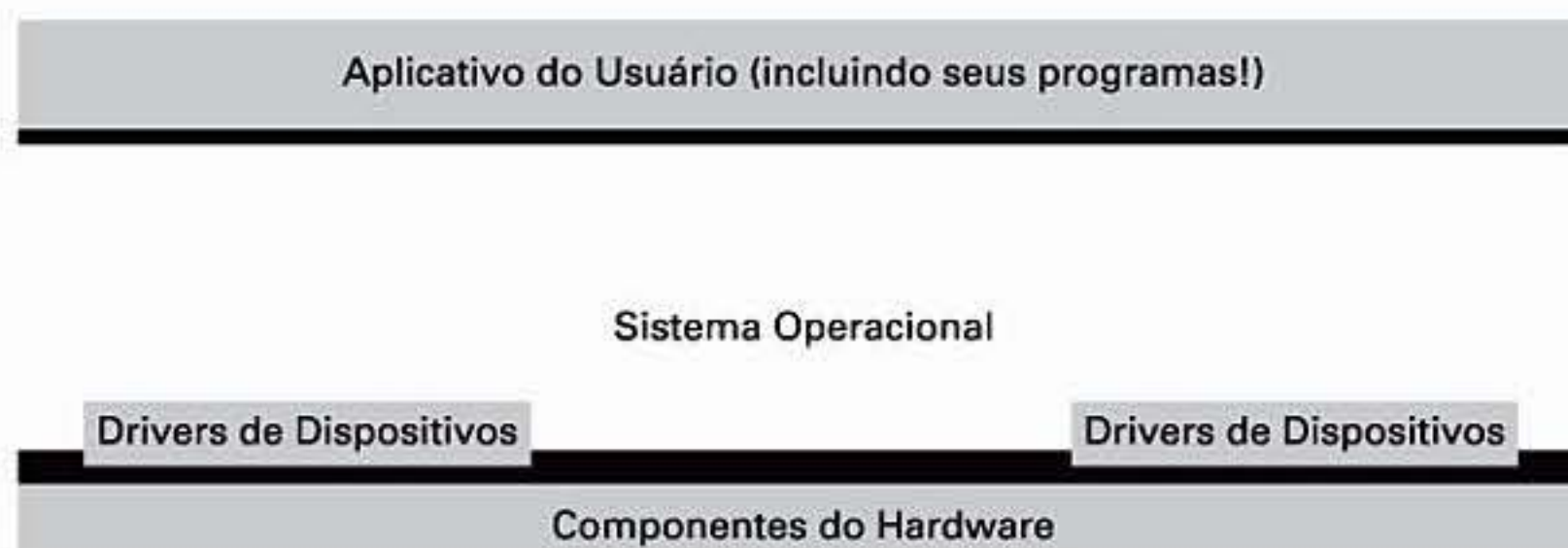
junto de instruções para todos esses elementos de hardware, em geral escrito para execução de alguma tarefa que esse não poderia concluir sozinho. Todas as operações básicas que envolvem o uso de unidades de disco, memória, monitor e impressora são complexas. Um programa que tivesse de interagir com essas operações gastaria a maior parte de seu tempo nelas, e só um pequeno percentual seria empregado em sua finalidade real. Escrito no hardware, um programa para calcular os pagamentos de uma hipoteca provavelmente teria centenas ou milhares de linhas para gerenciar a exibição e outras questões, e apenas algumas linhas para fazer o cálculo. Assim a programação era feita no passado, sem que fosse muito produtivo porque muito pouco tempo podia ser gasto na finalidade real do aplicativo. O que se precisava era de alguma maneira pela qual todos esses detalhes fossem manipulados para que os programas pudessem se dedicar a suas tarefas específicas.

A Função dos Sistemas Operacionais

Para fornecer essa camada básica de funcionalidade nos equipamentos, os sistemas operacionais foram criados. Eles mesmos são programas de computador, mas sua função é manipular todos os detalhes de gerenciamento de memória, entradas/saídas do disco (E/S) e outras tarefas de nível inferior. Quando há um sistema operacional (OS, Operating System) em um computador, outros programas podem ser escritos que não tenham de lidar com todos os detalhes de nível inferior; se os programas precisarem abrir um arquivo ou formatar um disquete, poderão solicitar ao sistema operacional que execute essa função para eles. Examinando isso graficamente (veja a Figura 1.1), você poderá visualizar as relações entre o hardware do computador e o sistema operacional, ou desse com outros programas, como várias camadas de funcionalidade.

FIGURA 1.1

O sistema operacional se torna a interface entre o equipamento do computador e seu programa, permitindo que você evite códigos específicos de hardware.



Com frequência, os relacionamentos não são definidos com muita clareza; um programa pode precisar acessar o hardware diretamente (sem passar pelo sistema operacional) para usar um recurso dele ao qual o sistema operacional não dê suporte ou para tentar melhorar o desempenho. Isso decerto foi o que aconteceu, no início, com um dos primeiros sistemas operacionais de PCs, no qual muitos programas tinham de interagir diretamente com o hardware. Essas limitações significavam mais trabalho para as pessoas que quisessem escrever softwares para PCs, já que cada programa precisava manipular suas próprias impressoras e outros detalhes. Ao mesmo tempo em que versões sucessivas dos sistemas operacionais forneciam um aumento na funcionali-

dade, se tornou mais fácil escrever programas para eles. Por fim, o Windows substituiu esses sistemas. Uma das maiores melhorias oferecidas pelo Windows é que o sistema operacional agora provê recursos de interface de usuário para outros programas. No Windows, se um programa de computador precisar mostrar uma caixa de diálogo na tela (como a da Figura 1.2), ele apenas solicita ao sistema operacional para exibi-la e fornece a mensagem que a acompanhará. Para acessar toda a funcionalidade que o sistema operacional provê, um conjunto de APIs (Application Programming Interfaces) é disponibilizado. Essas APIs representam a exposição de todos os recursos do sistema operacional e podem, portanto, ser usadas em seus programas.

FIGURA 1.2

O Windows manipula a exibição de elementos de GUIs (Graphical User Interfaces, Interfaces gráficas com o usuário) como esta caixa de diálogo, o que faz parte dos serviços que ele fornece aos programas.



O resultado de todas essas melhorias é que cada programa tem de manipular cada vez menos operações genéricas do computador e pode, portanto, se dedicar a sua finalidade real. Outro grande benefício da remoção de códigos específicos de hardware dos aplicativos é que, quando ocorrem alterações no hardware (impressoras e unidades novas de disco rígido, CPUs mais velozes), o sistema operacional pode ser atualizado para manipular o equipamento novo, mas os programas executados nesse computador não devem ser afetados.

O que isso significa para você como programador do Visual Basic? Bem, significa que poderá criar programas de computador para realizar uma tarefa específica sem ter de saber nada sobre como o Windows desenha as figuras no monitor, emite documentos na impressora ou salva arquivos no disco rígido. Você se tornará produtivo rapidamente, com a ajuda deste livro.

O Papel das Linguagens de Programação

Então, agora você sabe o que o sistema operacional fornece, mas e os programas propriamente ditos – como são criados? Antes, neste capítulo, defini um programa de computador como um conjunto de instruções para o hardware. Já que o hardware só pode realizar operações relativamente fáceis, as instruções também devem ser simples. O resultado final do programa é um código que o hardware pode entender, chamado com frequência de *linguagem de máquina* ou *código nativo*. As instruções, depois de carregadas na memória pelo sistema operacional, são compostas de comandos como “transfira a memória de um local para outro” ou “execute uma função matemática com os valores”. Milhares desses comandos formam um programa completo.

É possível escrever programas usando diretamente esse código nativo, com a criação de um arquivo no disco rígido repleto de instruções, mas seria necessário muito trabalho para produzir até mesmo um programa mais simples. Para evitar todo esse esforço e permitir que os programadores se dediquem à finalidade de seus programas, linguagens de nível superior foram criadas. Essas linguagens permitem que você use instruções mais avançadas e complexas que são, então, convertidas para as diversas instruções necessárias que sejam correspondentes às outras na linguagem de máquina. Uma única linha de uma linguagem desse tipo provavelmente poderia se transformar em dez instruções separadas na linguagem de máquina.

NOVO TERMO

O processo de converter uma linguagem de computador de alto nível em código de máquina ou nativo é conhecido como compilação. Os programas que fazem essa conversão são chamados de compiladores.

Muitas dessas linguagens foram criadas com o passar dos anos. FORTRAN, COBOL, APL, Pascal, C e BASIC são apenas alguns exemplos, mas centenas de linguagens diferentes estão disponíveis. Cada linguagem possui seu próprio conjunto de comandos e, com o tempo, novos comandos são criados para simplificar ainda mais o esforço de programação. Assim como os computadores evoluíram progressivamente, o mesmo aconteceu às linguagens usadas para programá-los, sendo adicionados novos recursos às existentes ou com a criação de novas linguagens como a C++ (que, como você deve ter adivinhado, foi baseada na C) e a JAVA. Em geral, os aprimoramentos nas linguagens têm o objetivo de aumentar a produtividade da programação, permitindo que o desenvolvedor crie o programa desejado da maneira mais fácil possível.

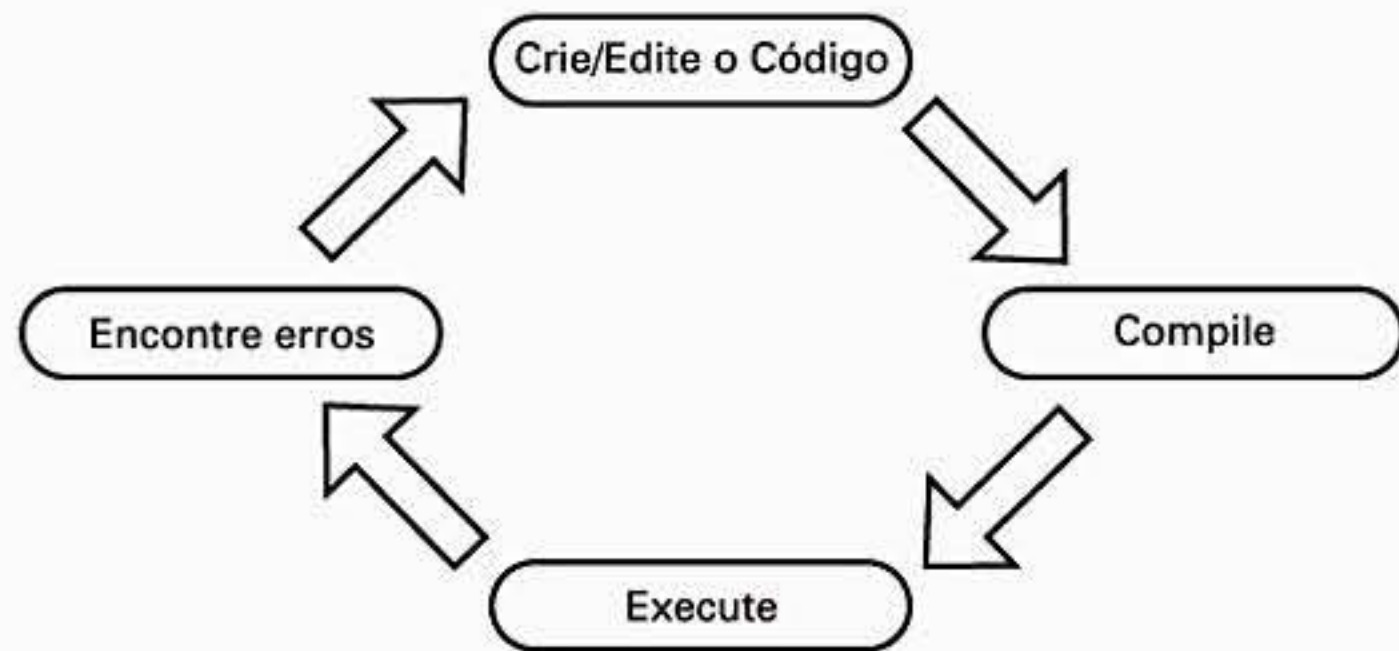
**NOTA**

É simplista declarar que as linguagens de programação evoluem apenas para tornar o desenvolvimento mais rápido. A velocidade no desenvolvimento é apenas uma das motivações por trás das melhorias que ocorreram. Outros objetivos e resultados são a produção de aplicativos mais rápidos ou estáveis (com menos falhas) ou até mesmo a criação de aplicativos mais fáceis de instalar.

Originalmente, a maioria das linguagens de programação era composta de apenas um item, um compilador. O programa era criado com o uso de um editor de texto, como o Bloco de notas, e o compilador era executado, passando o nome do arquivo ou programa-fonte. Em seguida, ele produziria o resultado final, um programa executável, pressupondo a inexistência de erros. O resultado compilado era executado, testado para a verificação de erros e, então, você voltaria a seu editor de texto para fazer alterações no código. Ele seria compilado mais uma vez, e o ciclo se repetiria. Esse processo de desenvolvimento (veja a Figura 1.3) não era de uma linguagem específica; era uma atividade comum para todos os programadores.

FIGURA 1.3

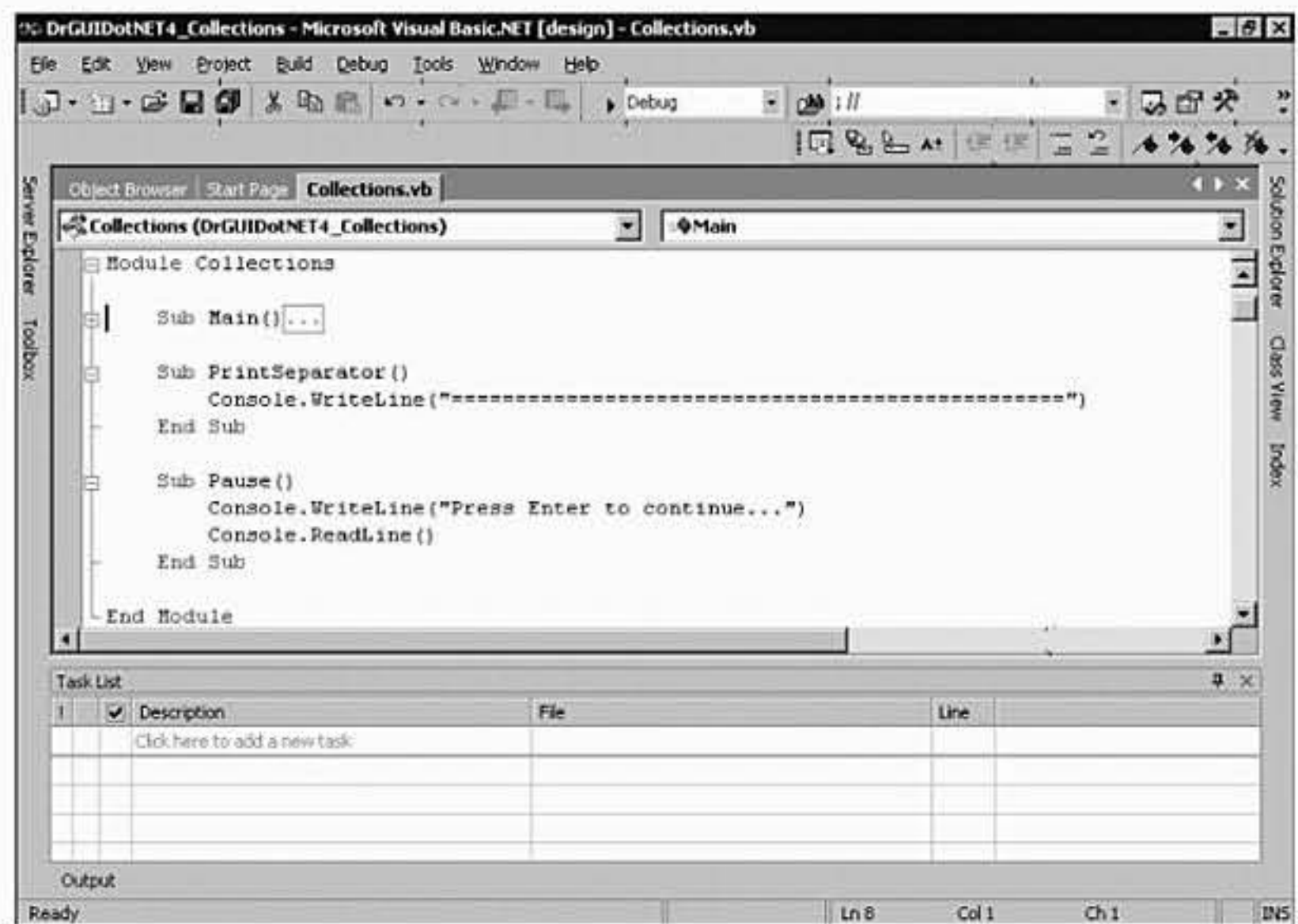
Os compiladores convertem o código-fonte de linguagens de alto nível em instruções que o computador possa compreender.



Enquanto as linguagens de programação evoluíam, esse ciclo foi melhorando também, resultando no desenvolvimento de compiladores mais avançados e no conceito de um Ambiente Integrado de Desenvolvimento (IDE, Integrated Development Environment). A finalidade de um IDE é combinar os componentes de edição, depuração e compilação do desenvolvimento de softwares em uma única interface para o programador (veja a Figura 1.4). Apesar da interface única, a tecnologia real é muito semelhante. Na maioria dos casos, o código ainda é compilado (veja a Nota a seguir para mais informações) e o programador ainda cria arquivos de texto, mas o ambiente de trabalho é muito mais amigável.

FIGURA 1.4

O Visual Studio é um IDE que fornece uma interface única para qualquer tipo de linguagem, inclusive para o Visual Basic.





Além das linguagens compiladas (em que o código-fonte deve ser processado em um compilador antes que possa ser executado), existe um outro tipo chamado *linguagem interpretada*. Nessas linguagens, o código-fonte não é compilado. Em vez disso, um programa especial executa o código lendo o fonte e processando o que for apropriado. Em geral, já que a análise (a leitura do fonte à procura de comandos) tem de ocorrer sempre que o programa é executado, as linguagens interpretadas são processadas mais lentamente do que as compiladas, nas quais a análise só ocorre uma vez, na hora da compilação. O Visual Basic já foi uma linguagem interpretada, mas esse não é mais o caso.

A criação de IDEs gerou alguma confusão entre quais seriam os seus recursos e os da linguagem a ser usada. Isso é realmente verdade no caso do Visual Basic, para o qual o IDE fornece muitos recursos que permitem ao programador criar com facilidade uma funcionalidade avançada. Esses recursos quase sempre geram o código do Visual Basic, e é ele que executa o trabalho. Nesse caso, é o IDE que adiciona a funcionalidade, e não a linguagem, mas ambos são em geral vistos como o mesmo elemento. Hoje, quando for introduzido o conceito do Visual Studio .NET, você verá que um IDE pode dar suporte a muitas linguagens, e talvez seja importante compreender a diferença entre ele e sua linguagem subjacente.

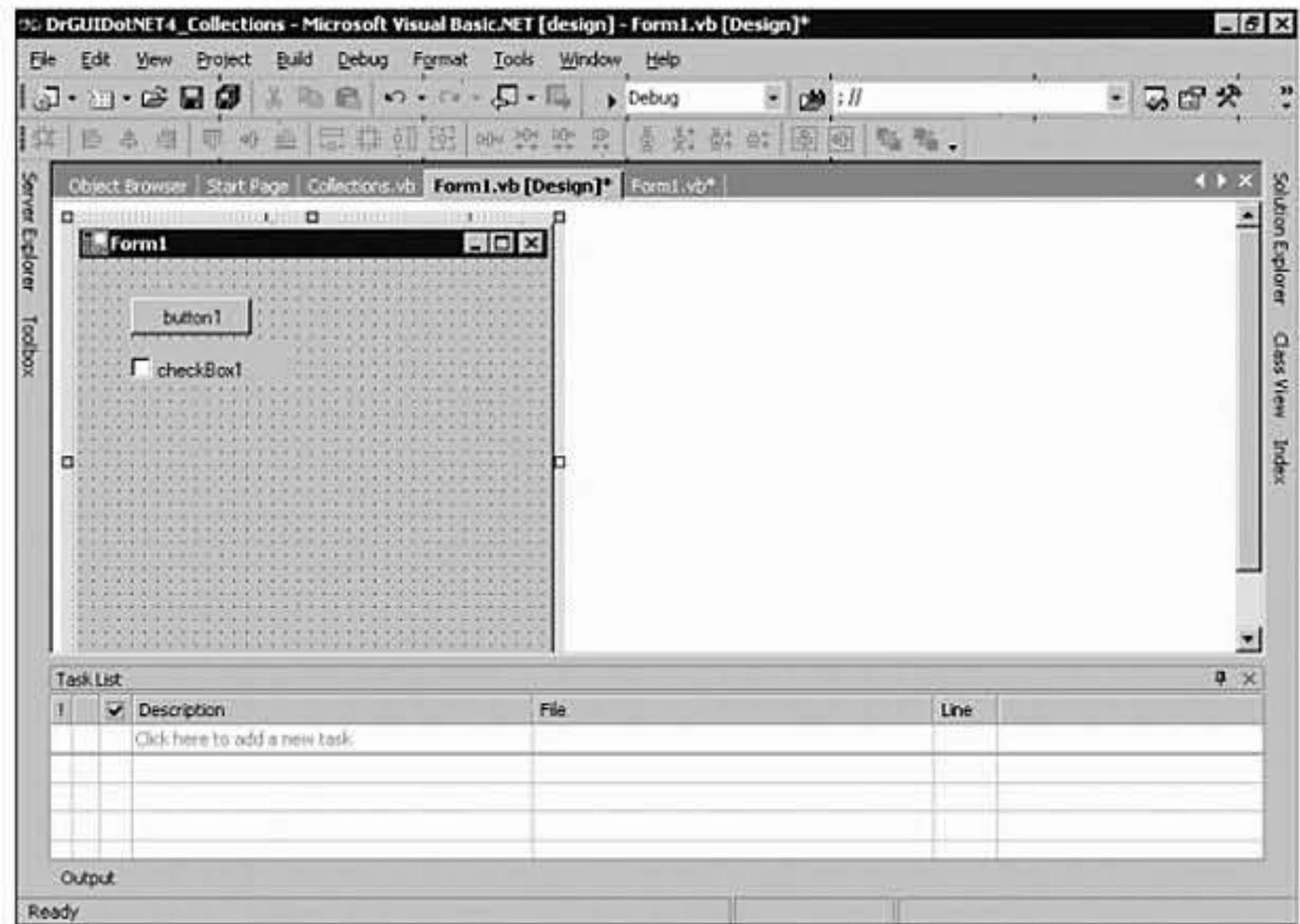
Outro avanço importante na história das linguagens de programação foi a criação das linguagens ‘visuais’, como o Visual Basic. Essas linguagens são chamadas ‘visuais’ porque permitem a geração de programas que usem uma interface gráfica. O recurso mais comum desse tipo de linguagem é a possibilidade de inserir botões, texto e outros itens em uma tela para construir uma interface com o usuário (veja a Figura 1.5). Mais uma vez, na verdade, um código real frequentemente é gerado, mas o programador tem um trabalho muito mais agradável com a criação das partes gráficas de seu aplicativo. Abordarei a posição ocupada pelo Visual Basic como linguagem de programação um pouco mais adiante na lição de hoje.

Por Que Escrever Programas de Computador?

Mencionei antes que a pergunta mais comum que ouço é: “O que faz um programador de computadores?”. Depois de ter explicado o conceito geral do trabalho, quase sempre ouço a segunda pergunta mais comum, “Por que você iria querer fazer isso?”. A resposta, “Porque é divertido escrever códigos”, parece nunca satisfazer as pessoas. Portanto, já que começamos a aprender a escrever programas, vale a pena dedicar um pouco do tempo considerando o que iremos escrever e por quê.

FIGURA 1.5

As ferramentas visuais de desenvolvimento permitem que você crie interfaces graficamente e, em seguida, gere o código necessário.



1

A primeira coisa a compreender sobre programação de computadores é o tipo de programa que será desenvolvido. Quando é solicitada a dar exemplos de aplicativos, a maioria das pessoas responde com programas como o Microsoft Word ou o Excel, algumas podem mencionar jogos de computador e muito poucas incluiriam o próprio Windows na lista. Todos esses são definitivamente aplicativos de computador e alguém tem de programá-los, mas a categoria maior de programas é a que quase nunca é mencionada. Os programas mais comuns são os sistemas que vemos a toda hora. A menos que você esteja interessado em (ou obcecado por) programação, não pensa com frequência nos meses ou anos da vida do programador que são gastos no desenvolvimento. Esses programas, como o sistema de computador na locadora próxima ou o programa que o governo local usa para rastrear licenças de motorista, são desenvolvidos para um único cliente ou um pequeno nicho de mercado. Fora de seu universo de aplicação, não são tão diferentes de softwares prontos como o Microsoft Word. No entanto, sua natureza individualizada significa que há uma taxa muito alta de trabalhos de programação por cliente.

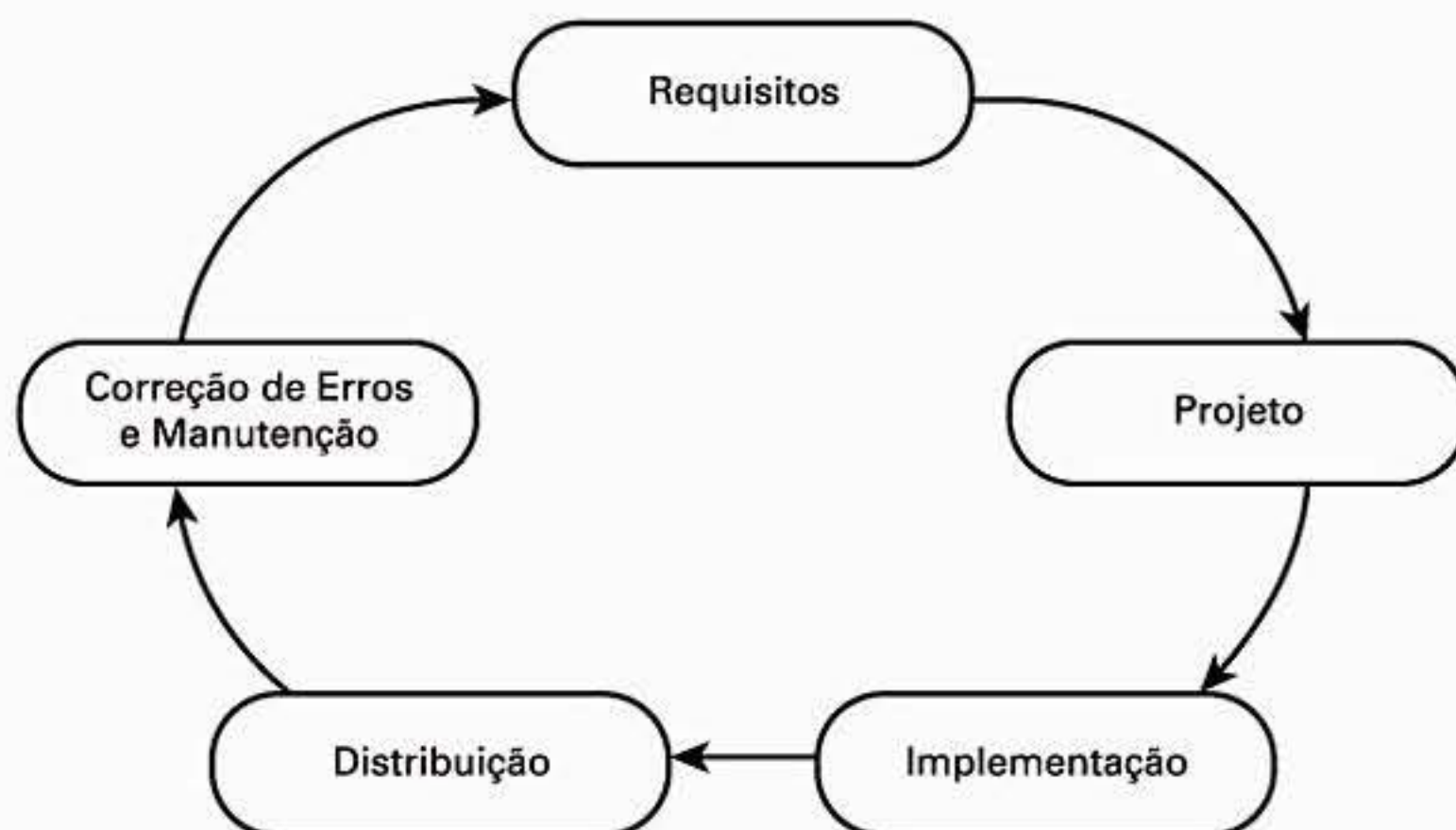
A função de um programador abrange muitos aspectos diferentes no desenvolvimento de softwares, principalmente no mundo do software personalizado. Mas o ciclo do desenvolvimento de softwares (SDLC, Software Development Life Cycle) contém várias outras etapas essenciais (veja a Figura 1.6).

Antes de a codificação começar, deve-se saber à que o novo sistema é destinado, isto é, seus *requisitos*. Essas informações devem se tornar uma planta geral para o aplicativo de computador, ou seja, o *projeto*. Podemos dizer que essa planta é, em linhas gerais, uma interpretação das necessidades do cliente na forma de partes de um sistema de computador, e mais tarde se tornará o guia por meio do qual o programador trabalhará. O resultado desejado é um sistema de computador que atenda às necessidades originalmente especificadas pelo usuário. Esse sistema completo

deve, então, ser distribuído para os usuários, com manutenção e aprimoramentos posteriores. Os erros – problemas no sistema – devem ser corrigidos, e novos recursos adicionados.

FIGURA 1.6

O ciclo de vida de um projeto de desenvolvimento de software passa por várias etapas distintas.



Esse ciclo é a vida e o trabalho do desenvolvedor de softwares. Mas, para a maioria dos programadores, o *motivo* pelo qual programam pode ser definido em termos mais simples: eles escrevem códigos que fazem algo de útil. A finalidade efetiva da programação é fazer o hardware do computador realizar tarefas práticas; é isso que guia a maioria dos desenvolvedores de softwares. Meu co-autor e eu mantivemos esse conceito em mente por todo o livro e nos certificamos de que quase todos os programas que mostramos executasse uma função de utilidade, além de fornecer a experiência para o aprendizado. É esse o objetivo real, criar aplicativos úteis, que conduz muitas pessoas ao Visual Basic, já que ele é considerado por muitas delas como o caminho mais rápido para passar da idéia ao aplicativo.

Um Breve Histórico sobre o Visual Basic

A história do Visual Basic na verdade começa com a invenção do BASIC (Beginner's All-purpose Symbolic Instruction Code) em 1964, uma linguagem que é facilmente assimilada e usada por programadores iniciantes. Essa linguagem se tornou popular, e, durante os 15 anos seguintes, várias pessoas e empresas criaram compiladores e interpretadores para o BASIC.

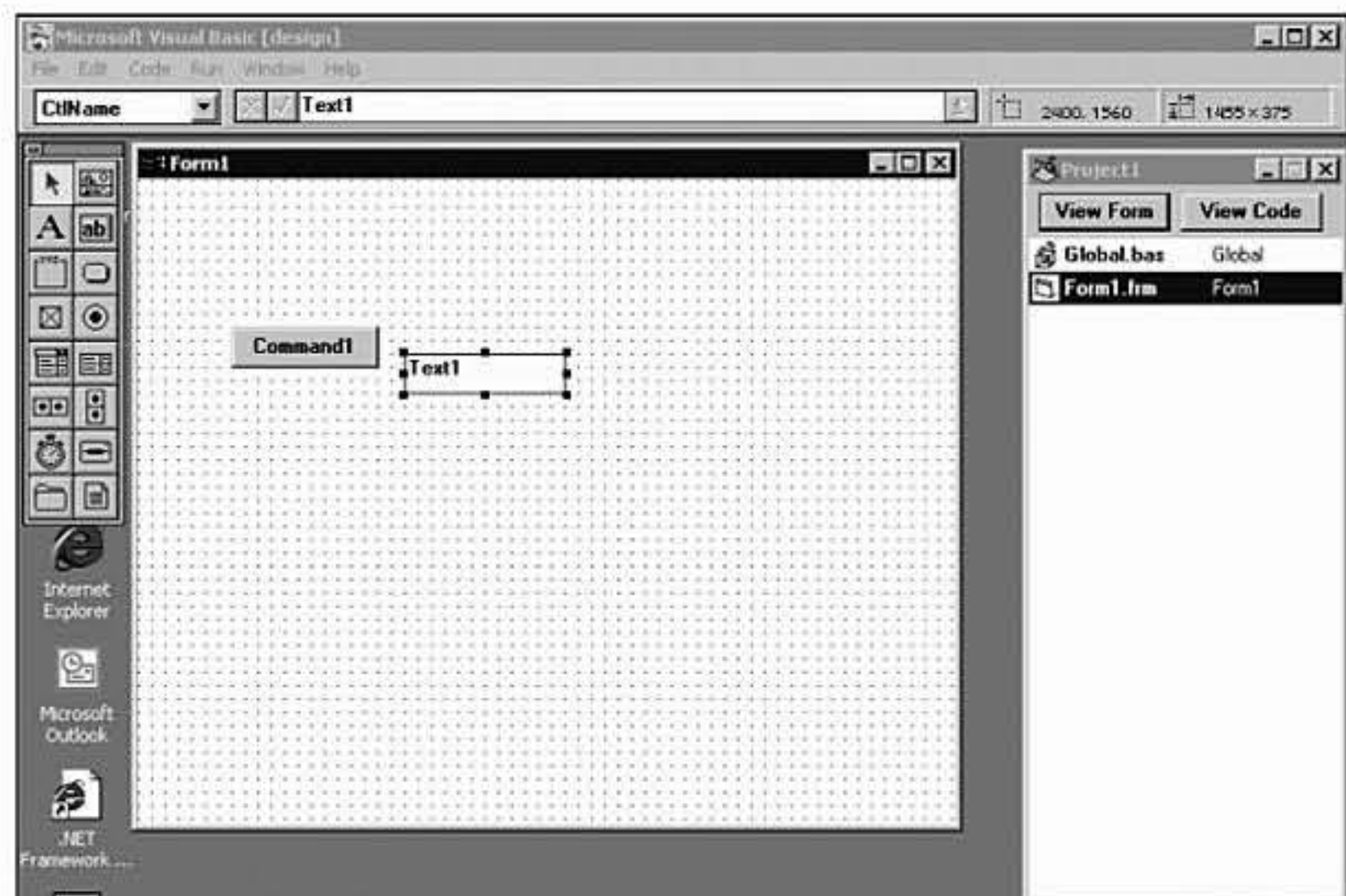
Em 1975, quando a Microsoft era uma empresa nova, uma versão da linguagem BASIC foi um dos primeiros produtos que ela criou, e que se tornou um sucesso. O BASIC da Microsoft e seu sucessor, o Quick BASIC (ou QBASIC, como também é conhecido), se transformaram nas versões do BASIC mais amplamente disponíveis nos PCs. Elas ainda possuem uma quantidade bastante grande de usuários (se você estiver interessado, acesse o endereço <http://www.qbasic.com> para obter recursos e links sobre essa linguagem). O Quick BASIC estava disponível para Windows quando foi lançado, mas era necessário um trabalho considerável para codificar uma inter-

face com o estilo do Windows, de modo que ele não era muito apropriado para codificação nesse novo ambiente.

No entanto, a Microsoft apresentou um novo produto que combinava a linguagem popular e fácil de aprender do BASIC com um ambiente de desenvolvimento que permitia aos programadores criar graficamente a interface com o usuário para um programa. Esse produto, como você já deve ter adivinhado, era o Visual Basic 1.0. No início, ele não foi adotado, mas fornecia realmente um ambiente veloz para o desenvolvimento de interfaces gráficas com o usuário (GUIs). O que pode ser uma surpresa é que o Visual Basic, na verdade, começou com uma versão em DOS, embora tenha sido logo transferido para o Windows (veja a Figura 1.7).

FIGURA 1.7

A primeira versão do Visual Basic para Windows fornecia muitos dos recursos essenciais comuns em várias ferramentas atuais, e o mais importante foi sua forma visual de arrastar e soltar usada no desenvolvimento.



O Visual Basic cresceu em popularidade com o tempo. Um único recurso se mostrou essencial para seu sucesso. Era a possibilidade de a Microsoft, o programador e outros fornecedores criarem componentes de interface personalizados que podiam ser adicionados aos programas. As empresas rapidamente entraram nesse mercado, desenvolvendo componentes que adicionavam vários recursos aos aplicativos do Visual Basic, como a geração de gráficos, edição de imagens, conexões com modems e muitos outros. Esses componentes permitiam que o programador do Visual Basic criasse aplicativos mais avançados combinando vários componentes com seu próprio código. Isso aumentou muito o desenvolvimento rápido possibilitado pelo Visual Basic e ajudou a torná-lo uma das linguagens mais populares de programação.

Toda versão atualizada do Visual Basic adicionava novos recursos, o que o tornava cada vez mais uma ferramenta de desenvolvimento completamente capacitada. Uma alteração importante, em particular, foi na maneira como o Visual Basic era processado no tempo de execução. Até a versão 5 ser lançada em 1997, o Visual Basic era uma linguagem interpretada, o que resultava em um desempenho relativamente fraco se comparado com o do Visual C++, Delphi e outras lin-

guagens compiladas. O Visual Basic 5.0 tornou possível criar versões compiladas ou interpretadas de programas, e o desempenho melhorou de modo considerável.

Outra alteração essencial no Visual Basic foi a possibilidade de criar componentes. Em programação, encontra-se com frequência um trecho de código, como uma rotina de cálculo de hipoteca, que poderia ser usado em muitos locais do programa e potencialmente até mesmo em vários programas diferentes. O compartilhamento desse código dentro de um programa em geral é conseguido escrevendo-o como um procedimento, um bloco de código que é digitado uma vez no programa, mas pode ser chamado de qualquer local do aplicativo. O compartilhamento entre programas pode ser feito apenas copiando-se o código do procedimento em cada novo aplicativo que for desenvolvido, mas isso gera um problema. Sempre que você, ou outra pessoa, fizer uma alteração nesse bloco de código (para corrigir um erro ou melhorar a maneira do código funcionar), precisará copiar o código para todos os aplicativos que o utilizam.

Uma maneira de melhor compartilhar o código é criar uma biblioteca de códigos armazenada em um arquivo separado dos programas que o utilizam e que possa ser alterado de modo independente. Essa biblioteca é conhecida como *componente* e é criada quase sempre na forma de um arquivo .dll. Usar uma biblioteca é o método preferido de compartilhar códigos. No decorrer do desenvolvimento de novas versões do Visual Basic, sua capacidade de criar esses componentes foi avançando regularmente. A versão 4.0 (lançada em 1996) foi a primeira a permitir a criação de componentes, e hoje esse é um recurso extremamente essencial em projetos de desenvolvimento. Discutiremos os componentes com mais detalhes no Dia 14, “Introdução à Programação Orientada a Objetos”.

Muitos recursos complementares foram adicionados ao Visual Basic, mas tudo foi construído em cima da base existente. Isso não é raro. A maioria das ferramentas de desenvolvimento progride dessa maneira, contudo há o efeito colateral de acumular lixo. As versões novas de uma ferramenta tentam manter a compatibilidade com todos os aspectos muito pouco eficazes das anteriores. Reescrever uma linguagem a partir do zero é quase impensável. O trabalho necessário poderia ser enorme, e romper a compatibilidade com o código já existente para o usuário pode não ser bem aceito. O benefício de uma mudança dessas seria uma implementação nova e limpa por completo, que poderia manter o que fosse bom e descartar as partes inválidas da linguagem anterior.

Isso foi exatamente o que a Microsoft fez na passagem do Visual Basic 6.0 para o Visual Basic .NET. Ela reescreveu a linguagem para criar uma versão limpa que eliminasse o lixo que se acumulou por uma década de atualizações sucessivas. Isso significa uma curva de aprendizado severa para as pessoas que tinham experiência na versão anterior da linguagem, mas o resultado final vale o esforço. A alteração radical faz com que este seja um grande momento para ser iniciante no Visual Basic, já que os conceitos ensinados neste livro terão uma vida útil muito mais longa do que aquilo que se aprenderia com um material relacionado à versão anterior.

Há muitos benefícios nessa mudança, todos eles foram estimuladores dessa decisão, mas a motivação mais significativa foi a necessidade de adaptação ao novo ambiente .NET. No decorrer

desta lição, você aprenderá mais sobre a plataforma .NET, o que é e como o Visual Basic se enquadra nela.

O Que É .NET?

À primeira vista, .NET pode parecer apenas um conceito de marketing, uma maneira de evitar mais um número após Visual Basic, mas é muito mais que isso. .NET representa toda uma gama de tecnologias e conceitos que formam uma plataforma na qual você pode desenvolver aplicativos.

1



NOTA

Na verdade, o Visual Basic .NET possui um número real de versão, 7.0 – só que ele não é muito usado. Da mesma maneira como o Windows 2000 é na realidade a versão 5.0 do Windows NT, quanto mais simples ou assimilável é o nome, mais ele é usado. No entanto, não espere ouvir Visual Basic 7.0 com frequência; havia até uma penalidade em dinheiro dentro da Microsoft para quem se referisse ao Windows 2000 como NT 5.0.

No início desta lição, expliquei como o sistema operacional fornece um nível de funcionalidade básica aos aplicativos (como a possibilidade de ler arquivos de um disco rígido ou disquete). A plataforma .NET pode ser explicada de maneira semelhante; ela é uma camada que existe abaixo de seus programas e fornece um conjunto de serviços e funções básicas. Essa camada contém um grupo de aplicativos e sistemas operacionais chamado servidores .NET; um conjunto básico de objetos chamado .NET Framework; e um conjunto de serviços que dá suporte a todas as linguagens .NET chamado Common Language Runtime (CLR). Cada uma dessas partes da plataforma .NET (veja a Figura 1.8) será abordada individualmente.

FIGURA 1.8

.NET é mais do que apenas um item; é um conjunto de softwares e conceitos que funcionam juntos para permitir a criação de soluções para empresas.



Servidores .NET

O principal objetivo do conceito .NET é diminuir o desenvolvimento de sistemas distribuídos, nos quais o trabalho é feito em vários locais diferentes. Em geral, esse tipo de sistema faz seu trabalho no back-end, no nível do servidor. A Microsoft fornece um conjunto de softwares que juntos são conhecidos como .NET Enterprise Servers. Eles foram projetados para substituir os recursos de back-end necessários ao sistema distribuído. Esses produtos incluem:

- Um sistema operacional do servidor, o Microsoft Windows (Server, Advanced Server e Datacenter Server)
- Softwares de agrupamento e balanceamento como o Microsoft App Center e o Microsoft Cluster Server
- Um servidor de banco de dados, o Microsoft SQL Server
- Um sistema colaborativo de correio eletrônico com armazenamento de informações de forma livre, o Microsoft Exchange Server
- Um mecanismo de transformação de dados com base em XML (falarei mais sobre a XML no Dia 20), chamado Microsoft BizTalk Server
- Um servidor para o acesso a sistemas legados, como o AS/400s, chamado Host Integration Server
- E mais...

Juntos, esses servidores fornecerão os serviços básicos a seus aplicativos .NET, formando o alicerce dos sistemas. Este livro se referirá a eles algumas vezes, mostrando onde podem ser usados em seus sistemas.

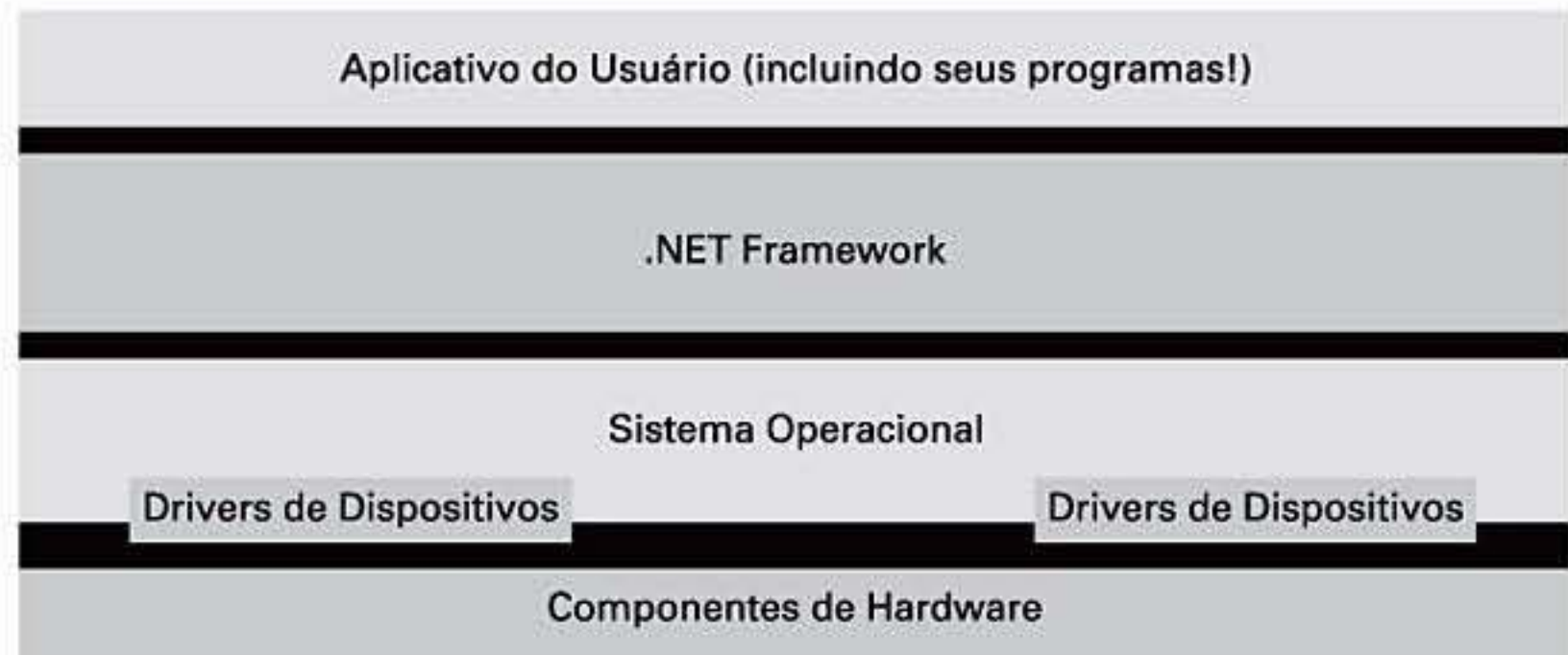
.NET Framework

Na passagem para o Visual Basic .NET muitas coisas foram radicalmente alteradas; uma delas foi o desenvolvimento de uma nova base para todas as ferramentas .NET de desenvolvimento. Essa base, conhecida como .NET Framework, fornece dois itens essenciais: o ambiente principal de tempo de execução e um conjunto de classes básicas. O ambiente de tempo de execução é semelhante ao sistema operacional por proporcionar uma camada entre seu programa e as complexidades do resto do sistema, executando serviços para seu aplicativo e simplificando o acesso aos recursos das camadas inferiores. As classes básicas fornecem um vasto conjunto de recursos, empacotando e definindo essas tecnologias como protocolos da Internet, acesso a sistemas de arquivos, manipulação de XML e mais.

O verdadeiro poder dessa estrutura será explorado com mais detalhes no Dia 8, “Introdução ao .NET Framework”. Por enquanto, só temos de entender que o .NET Framework é em muitos aspectos semelhante ao sistema operacional e que fornece seu próprio conjunto de APIs a fim de facilitar aos programadores o aproveitamento de seus recursos. A Figura 1.9 ilustra o relacionamento do Framework com o seu código e com os serviços subjacentes do sistema operacional.

FIGURA 1.9

O .NET Framework fornece outra camada de abstração sobre o sistema operacional, exatamente como esse faz para o hardware do computador.



1

Para uma linguagem de programação se beneficiar do ambiente de tempo de execução e outros recursos do .NET Framework, o compilador deve produzir um código que adote um certo padrão. A Microsoft fornece esse padrão, o Common Language Specification, como uma maneira de fazer com que qualquer compilador possa ser usado na plataforma .NET. Ela criou os compiladores do Visual Basic, Visual C++ e C# que se destinam ao .NET Framework, mas também disponibilizou o Common Language Specification fora da Microsoft de modo que outras empresas pudessem desenvolver compiladores para outras linguagens. O resultado é que, além das linguagens fornecidas pela Microsoft, muitas outras existentes (como o COBOL, APL, Smalltalk e assim por diante) foram desenvolvidas sobre exatamente a mesma base do Visual Basic. NET.

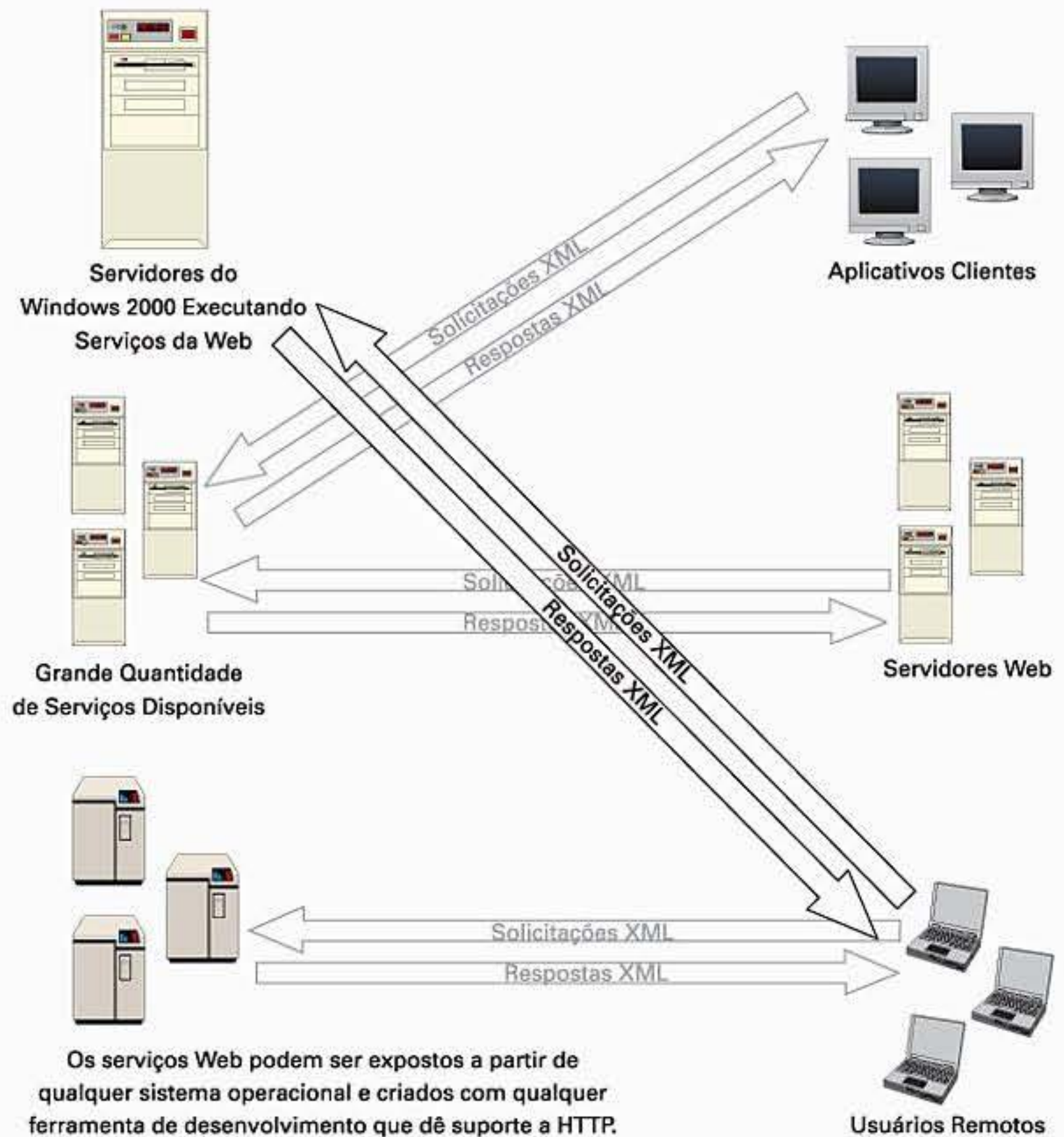
Neste ponto, é importante distinguir os recursos fornecidos pelas linguagens e os proporcionados pelo .NET Framework. O Visual Basic e o COBOL compartilham todos os recursos do Framework, mas, já que são linguagens diferentes, também possuem suas próprias funções e sintaxe exclusivas. Ao dar prosseguimento à leitura, você aprenderá tanto sobre os recursos específicos do Visual Basic quanto os fornecidos pelo Framework.

Serviços .NET

A plataforma .NET inclui certos conceitos que se estendem além dos detalhes de programação para descrever como os sistemas devem ser desenvolvidos e podem interagir. Um desses conceitos essenciais é a idéia de serviços da Web, recurso distribuído de uma maneira consistente de modo que sua execução fique totalmente confinada em seu ambiente. Um exemplo desse tipo de serviço pode ser uma folha de pagamentos, para a qual uma empresa possui servidores e aplicativos em sua própria organização que podem manipulá-la. A empresa fornece esse serviço para outras por meio de um serviço da Web (veja a Figura 1.10). Isso é diferente de apenas disponibilizar um site da Web; é uma interface que outros aplicativos ou sites da Web podem acessar através de código.

FIGURA 1.10

Um serviço da Web permite que as empresas forneçam serviços com base em software pela Internet e que outros aplicativos e/ou sites da Web os utilizem.



Essa tecnologia pode ser usada de várias maneiras, mas o conceito de serviços da Web consiste em que há certos serviços básicos ou fundamentais de que muitos aplicativos precisam, como autenticação, organização de calendário, envio de mensagens (correio eletrônico) e outros. Se esses tipos de recursos forem serviços da Web, qualquer pessoa no mundo poderá se beneficiar deles para reduzir o tempo de desenvolvimento de seu próprio sistema. A Microsoft, como participante da iniciativa .NET, está fornecendo alguns desses serviços para autenticação (conhecido como Microsoft Passport, www.passport.com), envio de mensagens (hotmail, www.hotmail.com) e outros.

Posteriormente, no Dia 21, “Criando Serviços Web com o Visual Basic .NET”, forneceremos uma introdução apropriada aos serviços da Web e nos aprofundaremos em como você pode usá-los em seus aplicativos.

Dispositivos .NET

Atualmente, há uma grande variedade de sistemas que podem ser usados para se conseguir acesso à Internet, à rede de sua empresa ou a suas informações pessoais. Sendo eles PCs completos, terminais da Internet com base em TVs, simples clientes ou Personal Digital Assistants (PDAs); todos são possíveis dispositivos utilizados pelo usuário para acessar um aplicativo .NET. Essa

tendência direcionada a vários dispositivos forçará você, como programador, a esquecer o padrão geral de um único tipo de cliente, em geral um PC, e a considerar, em vez disso, que um cliente poderia estar se conectando por meio de um entre muitos dispositivos possíveis. Esses podem ser classificados como dispositivos .NET – uma combinação de recursos de hardware e software projetados para funcionar com os serviços e aplicativos com base na plataforma .NET. Hoje, a variedade de dispositivos .NET disponíveis inclui computadores que executam o Windows (Windows 9x, Millennium Edition e Windows 2000 com o .NET Framework instalado) e dispositivos que executam o Windows CE. (O .NET Framework compacto está disponível para essa plataforma, permitindo o suporte aos recursos .NET.) Haverá um crescimento contínuo nessa área.

Não se preocupe se ainda houver partes da plataforma .NET que parecem confusas: elas precisam ser! Durante os dias restantes, as lições deste livro explicarão mais detalhes que ajudarão a esclarecer tudo. Por enquanto, lembre-se do relacionamento entre a plataforma .NET e seus programas – uma camada de funcionalidade que já é fornecida e fácil de acessar pelo seu código. Com esse conceito em mente e sabendo que o resultado final é um desenvolvimento mais rápido na passagem da idéia para o programa concluído, você terá aprendido bastante sobre a plataforma .NET nesta lição.

Desenvolvendo seu Primeiro Aplicativo no Visual Basic .NET

É hora de escrever seu primeiro trecho de código do Visual Basic .NET. Antes de continuar com este exemplo, e os do resto do livro, você deve ter o Visual Basic .NET instalado. Nesta seção, explorarei o processo de instalação do Visual Basic ou do Visual Studio (incluindo o Visual Basic) em seu computador. Se já o tiver instalado, vá para a seção “Onde Está meu IDE?”.

Preparando-se para Codificar

Antes de realizar qualquer trabalho com o Visual Basic .NET, será preciso instalá-lo. Você deve ter vários CDs ou um DVD (dependendo de onde conseguiu o produto) pelos quais possa fazer a instalação. Também é possível instalar de um local da rede, mas o processo é basicamente o mesmo nos três casos, sendo a principal diferença a alternância de CDs se você tiver a versão que vem em vários deles.

A primeira tela que você verá quando inserir o disco 1 ou executar a instalação pelo CD ou pela rede, será uma caixa de diálogo que apresentará três etapas distintas (veja a Figura 1.11).

A primeira etapa é elevar os componentes do sistema operacional de sua máquina até o nível que a plataforma .NET exige. Isso é chamado de Windows Component Update (WCU), e se você der um clique na primeira opção da caixa de diálogo da instalação (veja a Figura 1.11), será solicitado a fornecer o disco de WCU, se necessário, e a instalação começará. O que é visto durante a

instalação do WCU depende do sistema, porque apenas os itens que ainda não foram atualizados o serão. Algumas das instalações possíveis são a do Windows 2000 Service Pack 2, do MDAC 2.7 (componentes atualizados de acesso a dados) e do Internet Explorer 6.0. Um item que será instalado na maioria dos sistemas é o próprio .NET Framework, que não inclui o IDE do Visual Studio .NET, mas fornece todas as classes e arquivos de suporte do Framework necessários à execução dos aplicativos .NET. Dependendo de que componentes forem precisos, o programa de instalação pode ter de reinicializar seu computador uma ou mais vezes. Quando essa instalação estiver concluída, você retornará à primeira caixa de diálogo de instalação, como mostra a Figura 1.11.

FIGURA 1.11
*O processo de
instalação do Visual
Studio .NET é dividido
em três etapas.*

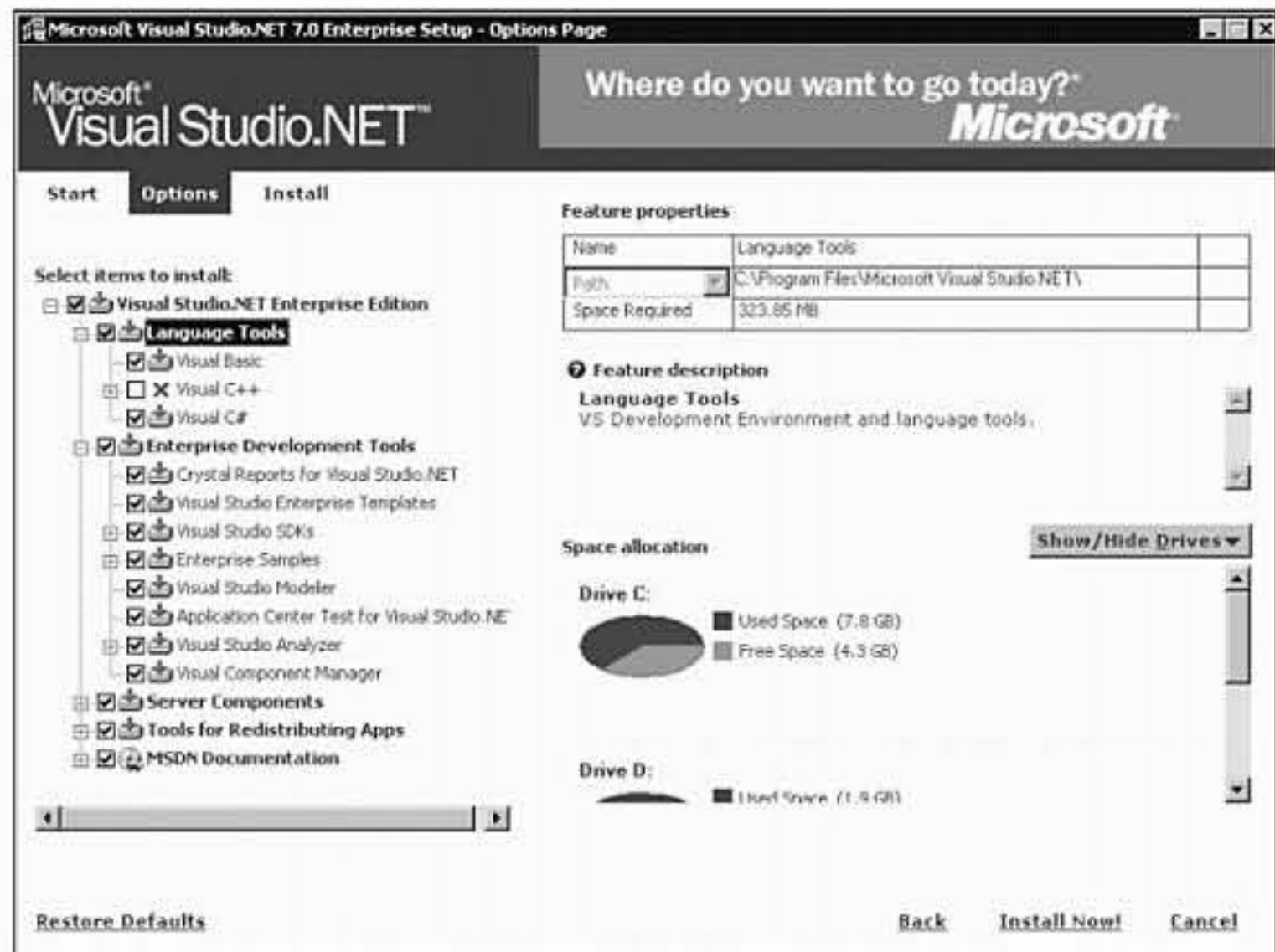


Dê um clique na etapa 2 para iniciar a próxima parte da instalação do Visual Studio .NET. Você será solicitado a inserir a chave do produto e aceitar ou não o Contrato de Licença de Usuário Final (EULA, End User License Agreement) antes que a instalação principal possa ser iniciada. Depois que as informações solicitadas forem fornecidas e o EULA aceito, surgirá uma tela com várias opções em que poderão ser selecionados os elementos do Visual Studio .NET que você deseja instalar (veja a Figura 1.12).

A Figura 1.12 foi baseada no Visual Studio .NET Beta 2 e algumas das opções podem se alterar, mas no geral você deve ver a mesma lista. Se estiver planejando programar grande parte no Visual Basic .NET, sugiro a seleção das mesmas opções que podemos ver na Figura 1.12. A primeira escolha digna de nota foi a seleção do Visual Basic e do Visual C# na opção das linguagens. O Visual C# não é necessário, mas veremos que muitos exemplos de código estarão disponíveis nessa linguagem, e é mais provável que façamos testes com a C# do que com a C++. Também é possível selecionar o Visual C++, mas só é aconselhável fazê-lo se houver realmente a pretensão de usá-lo já que ele ocupa mais espaço em disco do que qualquer das outras linguagens.

FIGURA 1.12

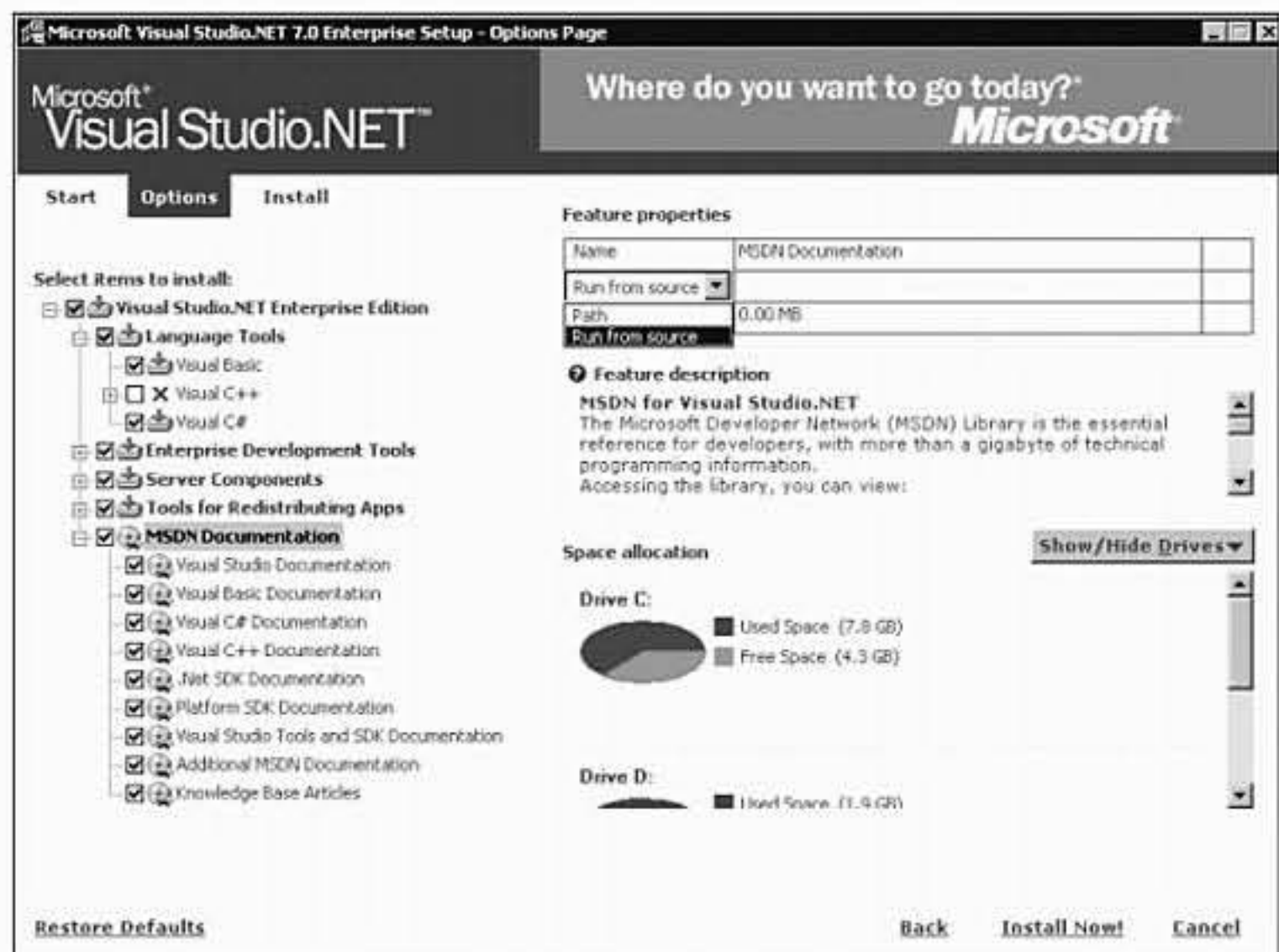
Você pode examinar e selecionar quais componentes do Visual Studio .NET deseja instalar.



A seguir, certifique-se de que marcou todas as opções disponíveis em Enterprise Tools, Server Components e Tools for Redistributing Apps. Não explicarei o que essas opções contêm, mas marcá-las assegurará que você tenha todos os recursos disponíveis em sua versão do Visual Studio (versões distintas do Visual Studio possuem conjuntos diferentes de opções de recursos). Para concluir, selecione MSDN e mantenha a configuração-padrão Run from source, que solicita acesso a seu CD, DVD ou local de instalação na rede, ou se tiver bastante espaço em disco, altere para a instalação em um caminho específico (veja a Figura 1.13). Executar o MSDN do local de instalação é mais lento do que tê-lo em sua máquina local, mas a decisão em geral é baseada no espaço disponível em disco em vez de na velocidade.

FIGURA 1.13

As bibliotecas MSDN, que contêm muitos documentos e artigos úteis, podendo os exemplos ser instalados localmente para que se obtenha mais velocidade.



Depois que você fizer todas as suas escolhas, dê um clique no link **Install Now!** na parte inferior da caixa de diálogo para iniciar a instalação. Vários trechos de texto que descrevem os recursos da plataforma .NET serão exibidos enquanto a instalação progride, encerrando por fim com uma tela de término (veja a Figura 1.14). Se algo der errado durante a instalação, será exibido nessa tela.

FIGURA 1.14

Se algo der errado durante a instalação, você será informado do problema nesta tela.



Por fim, quando você tiver retornado à tela inicial da instalação, a terceira opção estará disponível. Dar um clique nessa opção, **Service Releases**, o conduzirá à outra caixa de diálogo (veja a Figura 1.15), em que poderão ser selecionadas as atualizações da instalação pela Web ou pelo disco. Se houver conectividade com a Internet, recomendo escolher a primeira opção para assegurar a obtenção da lista mais recente de atualizações.

FIGURA 1.15

O programa de instalação do Visual Studio .NET pode fazer o download e instalar atualizações pela Internet.



Agora o Visual Studio .NET está instalado e deve estar pronto para ser executado, portanto, você pode passar para a próxima seção em que começará a escrever alguns códigos.

Onde Está meu IDE?

Agora estudaremos o Visual Studio .NET, um dos mais avançados pacotes de desenvolvimento disponível, e seu primeiro programa vai ser elaborado com o uso do Bloco de notas e do prompt de comando. Sim, é exatamente o que leu, o prompt de comando. No início desta lição, expliquei que muitas linguagens de programação eram compostas apenas de um compilador. Em uma linguagem desse tipo, os programas são escritos como arquivos de texto e, em seguida, o compilador é chamado para converter esse código (pressupondo-se que não tenham sido cometidos erros) em um programa executável. Na maioria das linguagens, isso nunca foi alterado. A finalidade de um IDE é fornecer um ambiente único de trabalho, mas ele ainda estará usando a mesma seqüência de edição/compilação/execução do processo manual. Antes do Visual Basic .NET, uma linha de comando podia ser usada para compilar os programas, mas isso era o mesmo que executar o IDE de modo não visível. O Visual Basic .NET separa as duas funções, compilação e edição, pelo fornecimento do compilador como um programa completamente distinto.

Com o IDE, você nunca precisará examinar a compilação. Ela será manipulada de modo automático, mas ainda assim será executada. A principal diferença entre o compilador do Visual Basic .NET e o de versões anteriores é que aquele é totalmente independente do IDE; não há nada que você possa fazer no IDE que não possa ser feito com o uso apenas do compilador da linha de comando e o Bloco de notas. Já que o IDE é um programa grande e complicado, será mais simples se concentrar na linguagem e trabalhar no nível da linha de comando de tempos em tempos. A lição de amanhã – Dia 2, “Trabalhando com o Visual Basic .NET” – introduzirá o IDE e percorrerá os fundamentos de seu uso para edição, execução e compilação de seus programas.

Uma Tarefa Simples

Para criar seu primeiro programa, você precisará de um objetivo, algo para ele executar. Não é minha intenção produzir programas “Hello World!” neste livro, aqueles que não possuem uma finalidade. Isso é particularmente difícil quando se tenta escrever o primeiro pequeno trecho de código.

NOVO TERMO

A expressão programa “*Hello World!*” ou “*Olá Mundo!*” descreve o exemplo de um programa que não faz nada de útil, ou mais especificamente, nada mais útil do que apenas exibir a mensagem solicitada.

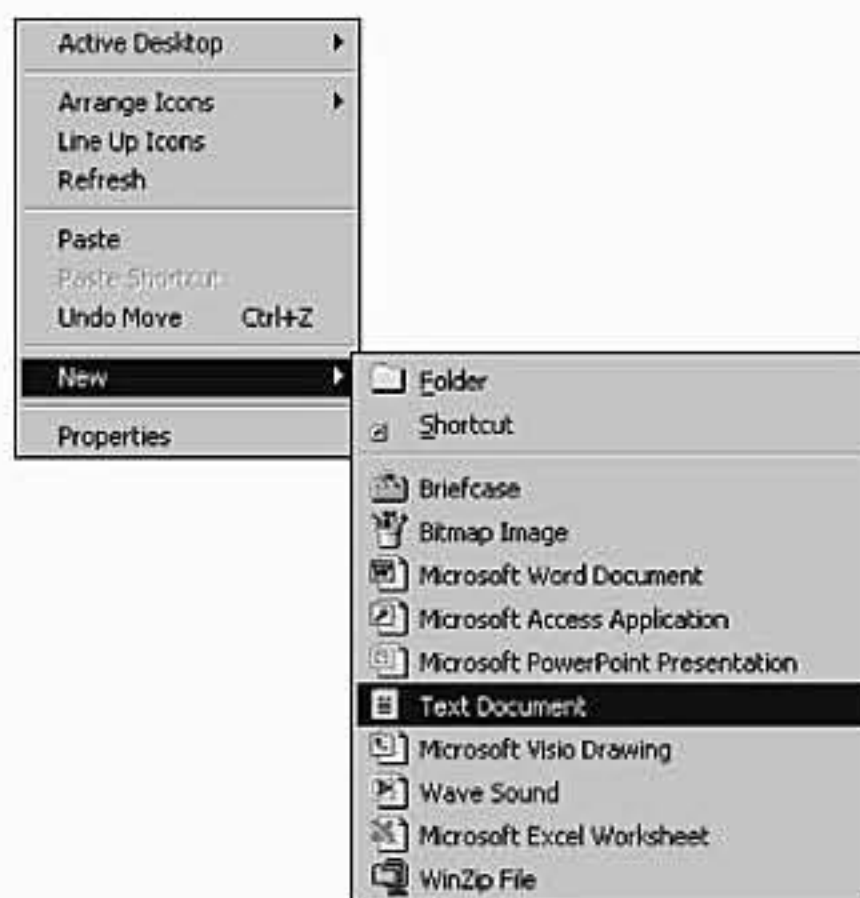
Você vai começar com um programa que poderá executar da linha de comando e que retornará um texto pequeno. Em seguida, mostrarei como esse programa ilustra quase tudo que é preciso saber para desenvolver um utilitário proveitoso que exiba informações sobre a versão de seu sistema operacional (por exemplo, no lançamento do Windows 2000 houve uma versão nº 5.00.2195). Isso pode ou não ser útil como um utilitário de linha de comando, mas demonstra o conceito básico de sua criação.

Escrevendo o Código

Já que você começará em um editor de texto (como o Bloco de notas) em vez de no IDE, nada será feito de modo automático, e o arquivo a princípio estará vazio. Isso significa que será preciso configurar a estrutura básica de seu programa antes de começar a escrever um código que seja compilado corretamente. Para iniciar, crie um arquivo novo de texto em algum local de seu computador; para os exemplos deste livro, crie um diretório em seu computador chamado `C:\TYVB\C1\`. Para gerar o novo arquivo de texto, dê um clique com o botão direito do mouse na pasta apropriada e selecione New (Novo) e Text Document (Documento de texto) (veja a Figura 1.16). Isso criará um arquivo de texto com extensão `txt`. Renomeie esse arquivo para que indique seu conteúdo final alterando para `Step1.vb`.

FIGURA 1.16

O menu de atalho do Explorer fornecerá a você uma maneira direta de criar um arquivo de texto e muitos outros documentos.



Com o arquivo renomeado e no diretório correto, você poderá dar prosseguimento e começar a configurar seu código efetivo. Abra o arquivo no Bloco de notas dando um clique nele com o botão direito do mouse e selecionando Abrir com. Isso abrirá uma caixa de diálogo por meio da qual será possível selecionar o Bloco de notas ou outro editor de texto (o Word Pad, por exemplo). Depois de ter selecionado um programa, dê um clique em OK, e o arquivo será aberto. Não deve haver nada nele neste momento; você terá de adicionar tudo que o fará funcionar.

Execução Passo a Passo

Sempre que você escrever um código, e principalmente quando fizer algo novo, tente resolver o caso em várias etapas. Dessa maneira, ao se deparar com um problema, saberá que partes do programa funcionaram e quais não tiveram êxito.

Na primeira etapa, você se certificará de que pode criar um aplicativo por meio do compilador de linha de comando. Para tanto, precisará de um código, porém ele não precisará executar realmente nada. Apenas terá de apresentar o formato correto. O código a seguir representa a estru-

ra básica de um programa do Visual Basic .NET; digite-o na janela de seu Bloco de notas e, em seguida, salve-o como Step1.vb.

```
Public Class Step1
    Shared Sub Main()
    End Sub
End Class
```



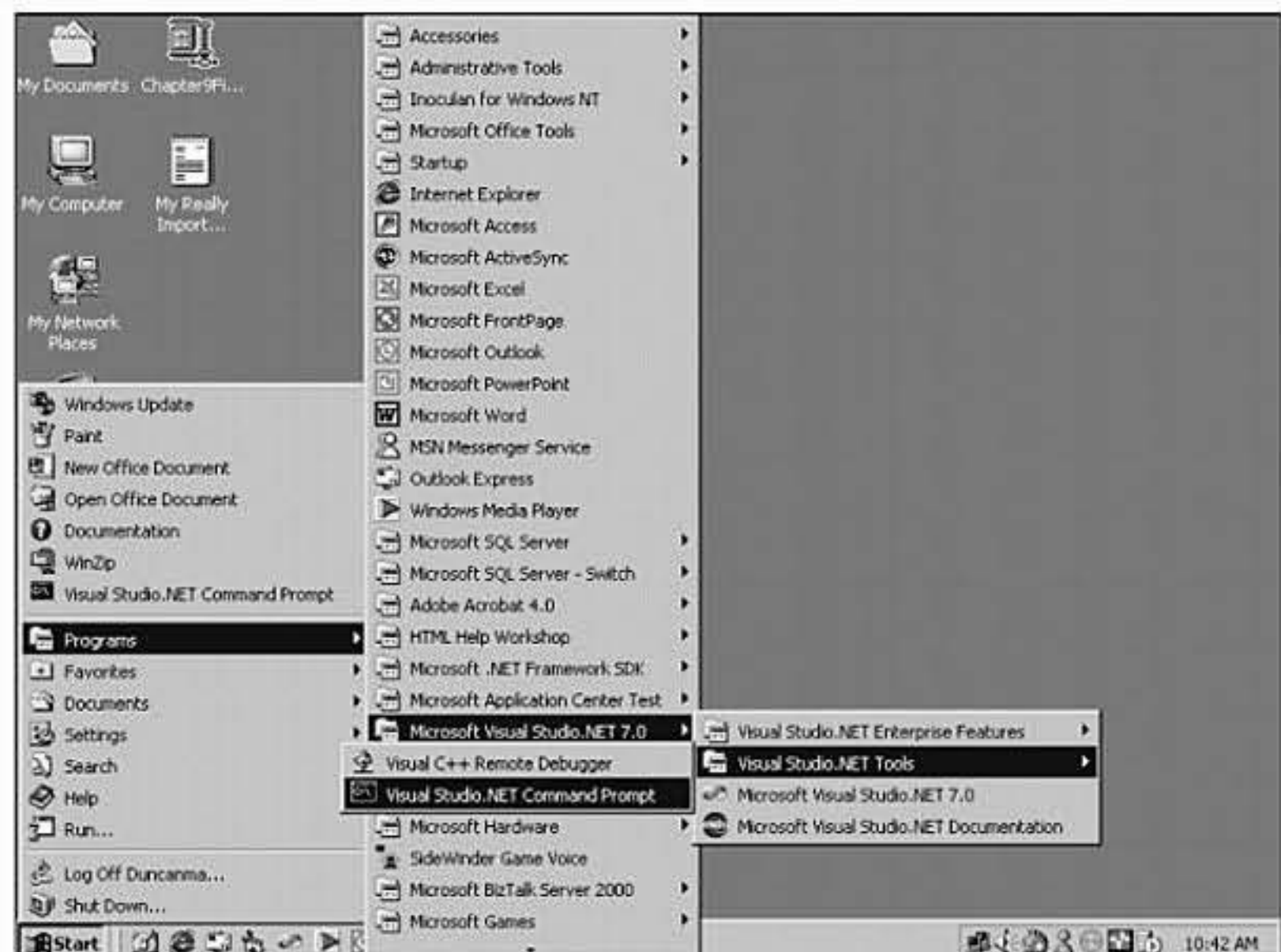
Ao tentar salvar seu código como Step1.vb, você pode acabar com alguns resultados estranhos. Por causa da tendência do Bloco de notas de criar arquivos de texto e dependendo das configurações de seu sistema com relação às extensões, pode terminar com um arquivo chamado step1.vb.txt em seu disco rígido. Para evitar isso, certifique-se de ter desmarcado a opção Ocultar extensões de tipos de arquivo conhecidos (guia Modo de exibição em Opções de pasta no menu Ferramentas do Windows Explorer).

1

A seguir, inicie um prompt de comando do programa Visual Studio .NET Tools, como na Figura 1.17.

FIGURA 1.17

Selecionar o prompt de comando do Visual Studio .NET é a melhor maneira de trabalhar com a plataforma .NET a partir de uma linha de comando.

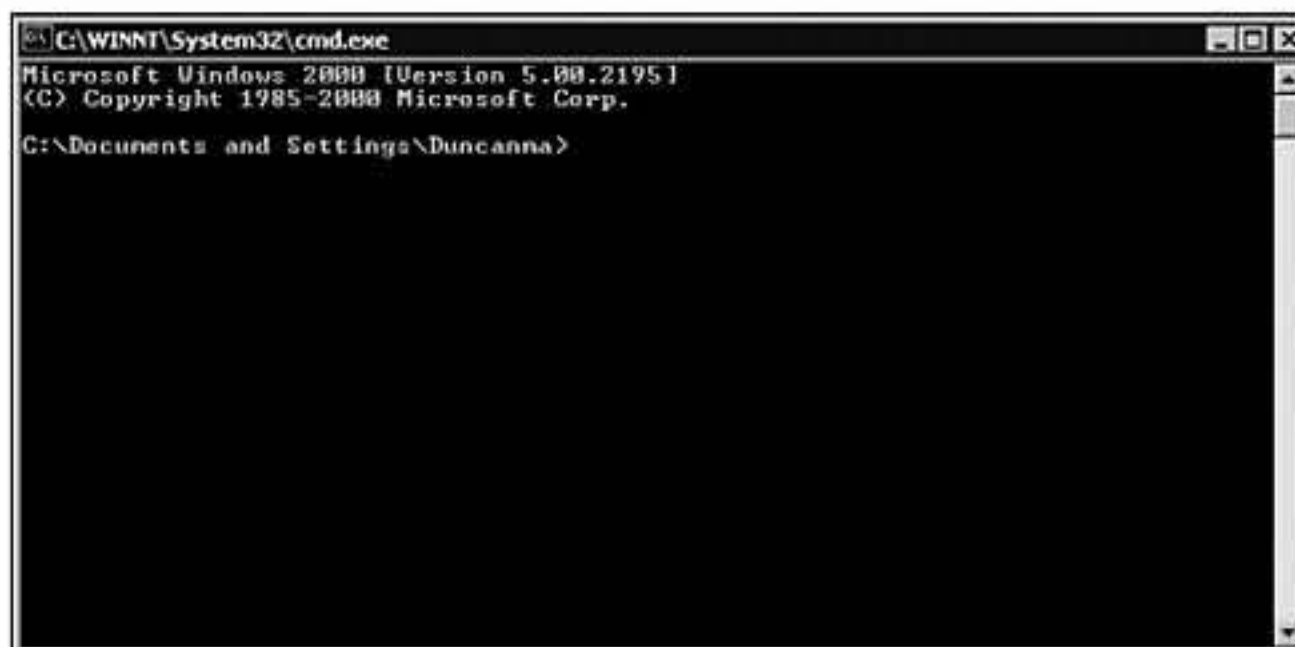


É essencial que você use essa opção para acessar a linha de comando, porque só através dela será possível conseguir que o(s) caminho(s) correto(s) seja(m) configurado(s) para executar ferramentas .NET como o compilador do Visual Basic (vbc.exe). Se quiser empregar uma linha de comando habitual, terá de adicionar a configuração do caminho apropriado (execute uma pesquisa em vbc.exe para encontrar o diretório que precisa estar em seu caminho) à sua máquina. Isso apresentará um console de comando (como mostra a Figura 1.18). Desse local, navegue para o diretório que contém seu arquivo Step1.vb. Agora você pode compilar seu programa,

criando um executável, mas adie um pouco mais adicionando uma etapa complementar antes da compilação.

FIGURA 1.18

O uso do compilador de linha de comando é feito por meio do console (ou do prompt do MS-DOS, como é frequentemente chamado).



No prompt de comando, digite `vbc` e pressione Enter. Você deve ver este resultado

RESULTADO

```
Microsoft (R)Visual Basic.NET Compiler version 7.00.9254
for Microsoft (R).NET CLR version 1.00.2914.16
Copyright Microsoft Corp 2001. All rights reserved.
Visual Basic Compiler Options
```

seguido por uma grande quantidade de informações de ajuda sobre o uso do compilador do Visual Basic. Se isso funcionar, você terá confirmado vários itens:

- Você realmente tem o Visual Basic .NET instalado.
- Você possui as permissões apropriadas para executar o compilador.
- O sistema pôde encontrar o arquivo `vbc.exe`.

Ao testar primeiro o `vbc` automaticamente, você poderá eliminá-lo como fonte de qualquer problema que encontrar ao tentar compilar um programa efetivo. Continuando esse raciocínio, agora podemos compilar seu programa salvo, `Step1.vb`, inserindo um comando do `vbc`, `Step1.vb /t:exe` e, em seguida, pressionando Enter. Se tudo der certo, deve aparecer este resultado:

```
Microsoft (R)Visual Basic.NET Compiler version 7.00.9254
for Microsoft (R).NET CLR version 1.00.2914.16
Copyright Microsoft Corp 2001. All rights reserved.
```

Isso significa que a compilação foi bem-sucedida, e se você estivesse examinando essa pasta agora, por meio do Windows ou emitindo um comando `DIR`, veria que já há um novo arquivo `Step1.exe`. Processar esse executável (digitando `Step 1` no prompt de comando e pressionando Enter) não produzirá nenhum resultado visível, o que é de esperar porque o código de `Step1.vb`, na verdade, não tem uma finalidade. O fato de que foi compilado e executado demonstra que ele até aqui está correto.

A chamada ao vbc pode aceitar uma grande variedade de parâmetros opcionais (comandos complementares adicionados à linha de comando, em geral separados por barras [/]), como você pode ver na execução automática do vbc. No entanto, nesse caso, só foram usados dois. Com o fornecimento do nome do arquivo-fonte, `step1.vb`, foi informado ao vbc que arquivo compilar. A segunda opção para `/t` (que significa ‘destino’ e também pode ser digitado como `/target`) pode ser um programa winexe, `exe`, uma biblioteca ou um módulo. Todas essas opções serão abordadas depois, mas, por enquanto, só as duas primeiras são particularmente relevantes; `winexe` cria um aplicativo do Windows e `exe`, uma linha de comando ou aplicativo do console (que é nosso objetivo). Por padrão, quando vbc for executado, ele usará `/t:exe`.

Vejamos a Exibição do Resultado

Agora, lembrando-se de seu objetivo final de retornar a versão do sistema operacional, tente exibir algum texto. Antes de conseguir um resultado efetivo em um programa do Visual Basic .NET, é preciso informar ao compilador que partes, se houver alguma, do .NET Framework serão usadas. Nesse caso, será empregado apenas o root (raiz) ou o objeto do nível superior do Framework, `System`, portanto, adicione a linha `Imports System` ao início do arquivo de código.

Agora você pode se referir a esse objeto, `System`, e usar seus recursos em seu código. Um dos membros ou partes da classe `System` do Framework é o objeto `Console`. Esse objeto permite que os usuários leiam e gravem através da interface de linha de comando. Nesse exemplo, empregue a instrução `Console.WriteLine` para retornar algum texto à janela do comando. Não importa que texto será retornado, é apenas um teste de exibição de informações. Em seu código, foi utilizado o texto *É aqui que entrarão as informações sobre a versão do sistema operacional!* para deixar claro que esse é só um espaço reservado. Salve esse novo arquivo de código alterado (que deve se parecer com o código a seguir) como `Step2.vb` e, em seguida, volte para a janela do comando a fim de compilá-lo:

```
Imports System

Public Class Step2
    Shared Sub Main()
        System.Console.WriteLine("É aqui que entrarão as informações sobre a
                                   versão do sistema operacional!")
    End Sub
End Class
```

A instrução de compilação será quase a mesma da etapa anterior, exceto por você especificar o nome do novo arquivo-fonte, `vbc Step2.vb /t:exe`. A execução dessa instrução de compilação produzirá um arquivo chamado `Step2.exe`, que poderá, em seguida, ser executado do prompt de comando digitando-se `Step2` e pressionando-se a tecla `Enter`. Se sua compilação funcionou e não houve problemas no código, o resultado a seguir deve aparecer em resposta a seus comandos:

**CÓDIGO/
RESULTADO**

```
C:\TYVB \C1>vbc step2.vb /t:exe
Microsoft (R)Visual Basic.NET Compiler version 7.00.9254
for Microsoft (R).NET CLR version 1.00.2914.16
Copyright Microsoft Corp 2001. All rights reserved.
```

```
C:\TYVB \C1>step2
```

É aqui que entrarão as informações sobre a versão do sistema operacional!

Adicionando a Funcionalidade das Informações sobre a Versão

Você sabe que o código está correto até este ponto (porque funcionou), portanto, ele fornece um bom ponto de partida para a próxima etapa. O que precisamos fazer agora é obter as informações reais sobre o sistema operacional e, em seguida, exibi-las no lugar do trecho genérico de texto. Teremos de usar o .NET Framework novamente para conseguir essas informações, por meio de outro recurso que ele fornece, o acesso aos dados sobre o sistema operacional atual, como o Windows 2000. Essas informações são disponibilizadas pelo mesmo objeto *System*, mas em vez de usar *System.Console*, empregaremos *System.Environment.OSVersion*. Altere o código de *Step2.vb* para que tenha a aparência do descrito abaixo e, então, salve o arquivo como *Step3.vb*.

```
Imports System
```

```
Public Class Step3
```

```
    Shared Sub Main()
```

```
        System.Console.WriteLine(System.Environment.OSVersion.ToString())
```

```
    End Sub
```

```
End Class
```

A compilação do código `vbc step3.vb /t:exe` não será diferente do que foi feito nos dois exemplos anteriores. Essa compilação produzirá um programa executável com o nome `step3.exe`. Executar esse programa exe, digitando `Step3` Enter na janela aberta da linha de comando, gerará o resultado a seguir:

```
C:\T V \C1 step3
```

```
Microsoft Windows NT 5.0.2195.0
```

Observe que o resultado indica um computador Windows 2000 (NT 5.0); outros resultados serão gerados para computadores diferentes. Agora você tem um pequeno programa exe que retornará a versão do sistema operacional, decerto ele será útil pelo menos para administradores de rede. O aplicativo é chamado de `step3.exe`, mas seu nome pode ser facilmente alterado para algo mais próximo de sua finalidade real.

Usando o Compilador de Linha de Comando

Nos exemplos anteriores, você usou um compilador de linha de comando para especificar dois itens, o destino dele (`/t:exe`) e o arquivo-fonte a ser compilado (`step1.vb`, por exemplo). O compilador funcionou muito bem com essas duas opções, produzindo automaticamente programas

executáveis com o mesmo nome do arquivo-fonte, mas há muitas opções complementares disponíveis como parte desse comando. Aqui estão alguns parâmetros de linha de comando que podem ser mais úteis quando começarmos a fazer testes com o Visual Basic .NET:

- `/target:<winexe, exe, library, ou module>` (pode ser chamado como `/t`) especifica que tipo de resultado deve ser criado pelo compilador. A opção `winexe` cria um aplicativo do Windows. A opção `exe`, usada em nossos exemplos, gera um aplicativo do console ou de linha de comando e é o padrão se nenhum parâmetro `/t` for empregado. As duas outras opções, `library` e `module`, são utilizadas na criação de resultados que não possam ser usadas diretamente, mas que estejam destinadas a fazer parte de outro aplicativo. Você aprenderá mais sobre todos esses tipos de saída nos dias posteriores.
- `/out:<nomedoarquivo>` é usado para especificar o nome do arquivo criado. Em seus exemplos, você o omitiu, fazendo com que `vbc` empregasse o nome de seu arquivo-fonte (`Step1.vb`) para gerar o arquivo de saída (`Step1.exe`, por exemplo).
- `/help` (ou `/?`) é equivalente a executar o `vbc` sem opções: em qualquer das duas formas, o resultado será uma listagem detalhada de todos os parâmetros disponíveis.
- `/verbose` faz o compilador produzir resultados mais detalhados durante a compilação e pode ajudar a solucionar problemas.
- `/reference:<nomedoarquivo>` (abreviando-se temos `/r:<nomedoarquivo>`) é usado para informar ao compilador que seu código requer alguns arquivos complementares além do que está incluído como padrão. Por exemplo, se você quisesse trabalhar com as partes de `System.Web` do .NET Framework, precisaria adicionar uma linha `Imports System.Web` no início de seu código e, em seguida, especificar `/r:system.web.dll` quando o compilasse.

Resumo

Nesta lição abordamos um pouco do histórico da programação, o Visual Basic e a plataforma .NET antes de criar um programa pequeno. Agora, você está pronto para passar ao aprendizado da linguagem propriamente dita e escrever programas mais complexos. O Visual Basic é uma das linguagens de programação mais populares do mundo, e já cobrimos um dia na direção que conduz ao seu domínio.

P&R

P Por que o Visual Basic .NET é tão diferente das versões anteriores do Visual Basic?

R O Visual Basic tem evoluído com o tempo. Cada versão tem sido muito semelhante à anterior, mas, em certos pontos, uma revisão maior como a da plataforma .NET era necessária para assegurar que a linguagem acompanhasse outras alterações da indústria, como a Internet e a computação distribuída.

P Ouvi falar sobre outra linguagem .NET, a C#. Devo aprendê-la em vez do Visual Basic .Net ou estudar as duas?

R A C# é uma ótima linguagem nova, uma maneira mais simples e fácil de usar a C++, mas não, você não tem de aprendê-la. Todas as linguagens .NET são iguais em termos de suas capacidades; se você puder criar um certo tipo de aplicativo em C#, poderá fazê-lo no **Visual Basic .NET**. É possível que as pessoas achem que muitos exemplos da Microsoft aparecerão em C#, apenas porque é uma linguagem que tanto os programadores do Visual Basic quanto da C++ acham relativamente fácil de ler.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Qual produto da Microsoft foi o predecessor do Visual Basic?
2. Por que todas as linguagens .NET (Visual Basic, C# e outras) compartilham certos recursos comuns (o .NET Framework, métodos de criação e uso de objetos e mais)?
3. Como se denomina o processo de converter o código-fonte (seus programas) em código nativo de máquina (como um arquivo exe)?
4. Considere o código a seguir:

```
Public Class MyClass
    Shared Sub Main()
        End Sub
End Class
```

Se esse código fosse salvo em um arquivo chamado `MySourceCode.vb` e, em seguida, fosse executado o comando `vbc /t:exe MySourceCode.vb`, que arquivo seria criado?

Exercícios

1. Nos exemplos desta lição, você não especificou um nome para o arquivo de resultado quando compilou, o que fez com que o compilador usasse o nome do arquivo-fonte como padrão. Recompile `step3.vb`, definindo um nome para o arquivo de resultado igual a `WhatOS.exe`.
2. Para obter informações sobre a versão, você empregou o objeto `System.Environment`, que possui várias outras propriedades úteis. Use a documentação de ajuda que está instalada no Visual Basic (veja a seção “Preparando-se para Codificar”) para saber o que mais está disponível e, em seguida, crie outro programa para exibir uma ou mais dessas propriedades complementares.

SEMANA 1

DIA 2

Trabalhando com o Visual Basic .NET

No Dia 1, “Bem-Vindo ao Visual Basic .NET”, você aprendeu algumas informações gerais sobre programação, Visual Basic e plataforma .NET, que lhe forneceram o conhecimento necessário para começar a desenvolver aplicativos. Nesta lição criaremos um aplicativo Windows real por meio do Visual Basic e, no processo, abordaremos os tópicos a seguir:

- O IDE do Visual Studio
- Soluções, projetos e arquivos
- Criação e execução de um exemplo de projeto no Windows

Começarei com uma visão geral do IDE do Visual Studio (VS). Os IDEs foram abordados na lição anterior (Dia 1), mas incluirei uma revisão rápida sobre seu significado e finalidade.

O IDE do Visual Studio

A finalidade do IDE (Integrated Development Environment, ou Ambiente Integrado de Desenvolvimento) é ser um ambiente único de trabalho para os desenvolvedores. Em geral, o código-fonte é composto apenas de texto e poderia ser inserido e editado em qualquer editor de texto (o Bloco de notas, por exemplo), e compiladores podem ser usados na linha de comando sem muito problema, portanto, tecnicamente o IDE é desnecessário. Apesar disso, há poucos programadores que concordariam em trabalhar com uma linguagem que não possuísse algum tipo de

IDE. Mesmo as tarefas simples de edição e compilação de seu código serão muito simplificadas pelo IDE do Visual Studio que fornece muitas possibilidades complementares que apenas não existiriam sem ele.

Iniciando

Antes que você possa usar o IDE do Visual Studio, ele deve ser instalado em sua máquina. O processo usado para fazer isso foi detalhado na lição de ontem, portanto, leia esta seção se precisar concluir a instalação antes de continuar.

Configurações do Perfil

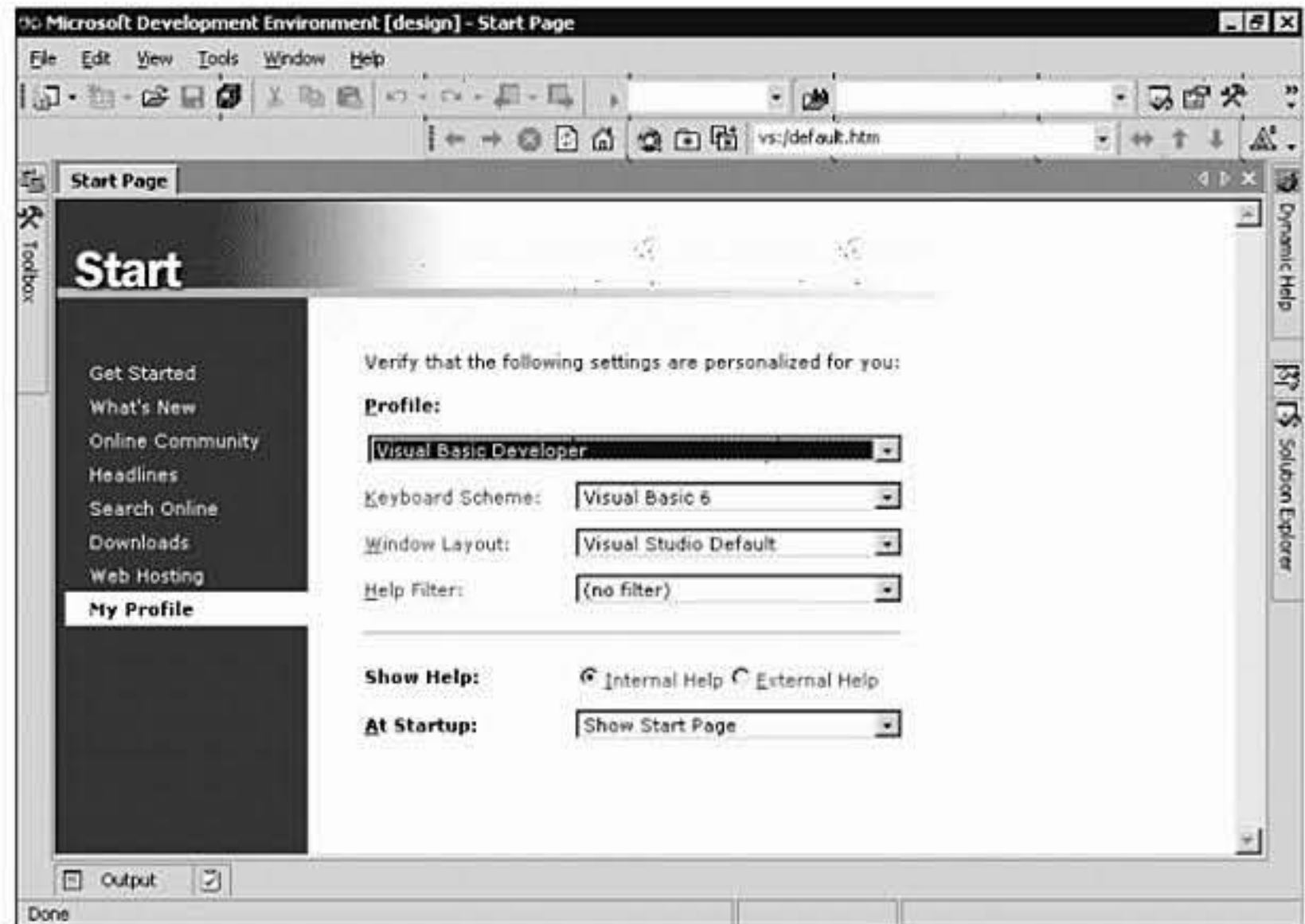
O primeiro item que você verá quando iniciar o Visual Studio .NET será uma interface parecida com uma página da Web (que foi chamada de Visual Studio Home Page) solicitando a confirmação das configurações de seu perfil. O conceito de perfil é novo nesta versão do Visual Studio, uma variação da idéia de preferências do usuário que provê a capacidade de configurar muitas opções diferentes em um único perfil. As opções básicas fornecidas foram projetadas para facilitar sua transição para este ambiente dependendo de sua experiência particular. Essa disponibilidade de padrões me levou a escolher rapidamente as configurações mostradas na Figura 2.1. No entanto, depois de algumas horas de uso, as alterei para algo um pouco diferente de qualquer dos perfis-padrão. É provável que você faça o mesmo.

Por enquanto, selecione o perfil Visual Basic Developer, e depois você poderá voltar à caixa de diálogo para alterar essas configurações se desejar. Dê um clique em Get Started para sair da página de configurações de perfil. O que apareceu agora é considerada a área principal de trabalho do IDE do Visual Studio, uma localização geral que contém conteúdo de vários tipos, como o código editado ou as páginas da Web projetadas. Ela também inclui um navegador da Web interno, que é aberto por padrão; ele é usado para mostrar uma interface para o Visual Studio com jeito de Web.

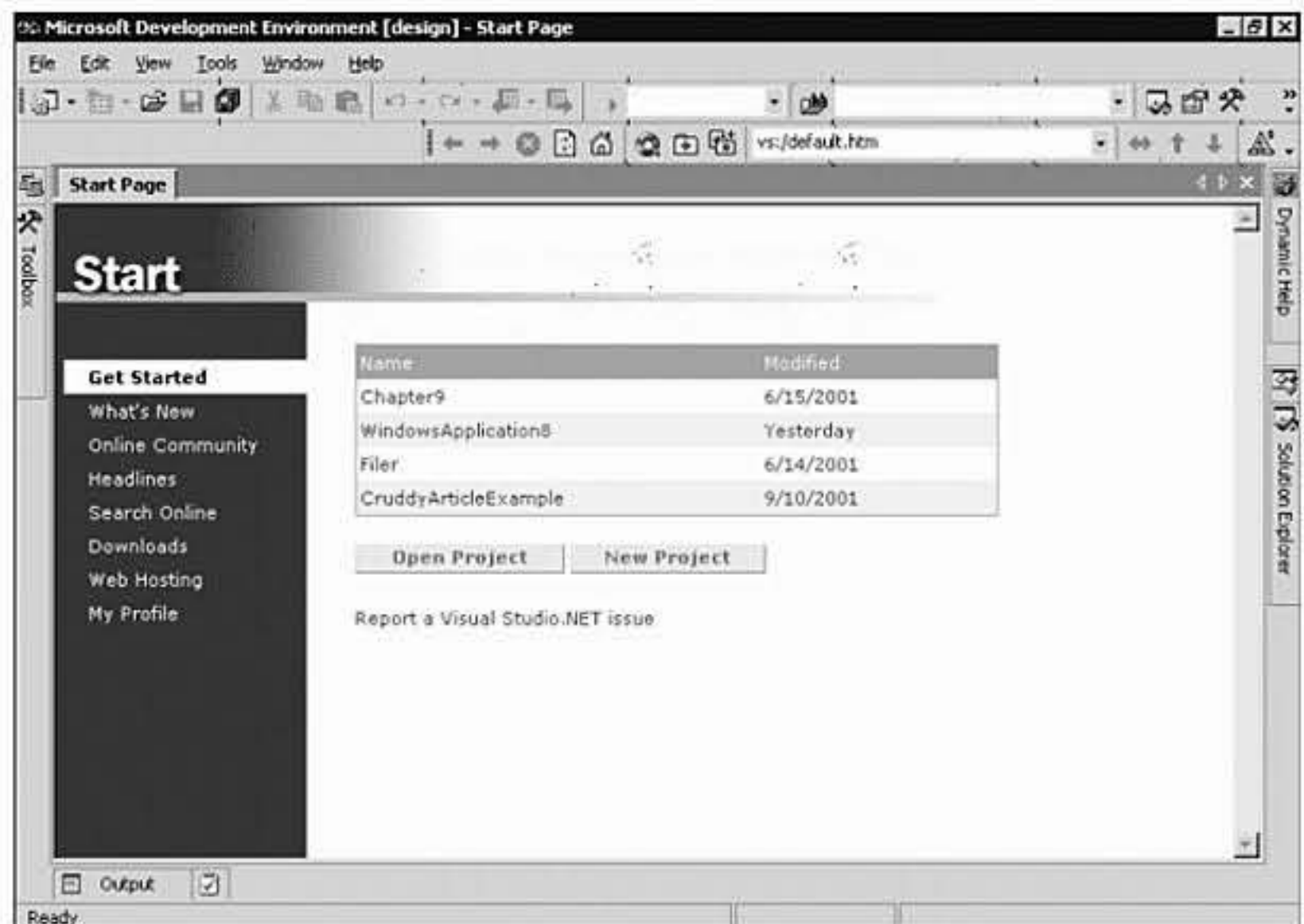
Na Home Page (veja a Figura 2.2), você encontrará várias opções úteis. A primeira, Get Started, é a página que será mostrada sempre que o Visual Studio for aberto. Projetada como uma preparação inicial para seu trabalho, essa página fornece tanto uma lista dos projetos recentemente abertos quanto um link para a criação de um novo, permitindo que se dê prosseguimento com um simples clique em algum local dela. Por hora, selecionaremos uma das opções de navegação que se encontram no lado esquerdo da página.

FIGURA 2.1

O IDE do Visual Studio .NET pode ser personalizado para tornar mais fácil a transição por meio de outras ferramentas de desenvolvimento, incluindo a versão anterior do Visual Basic.

**FIGURA 2.2**

A página inicial fornecida pelo Visual Studio disponibiliza uma visualização funcional dos projetos que você abriu recentemente, além do acesso a várias fontes de informação.



As opções laterais restantes apresentam:

- Detalhes dos novos recursos do Visual Studio .NET. What's New (O Que Há de Novo) está vinculada a vários recursos on-line como newsgroups (Online Community)
- Uma página de notícias ao vivo sobre o Visual Studio e outros tópicos relacionados a desenvolvedores (Headlines, veja a Figura 2.3)
- Um link direto para pesquisas na Web (Search Online)

- Um link de retorno à página de seleção de perfis que é apresentada por padrão na primeira execução do IDE (My Profile)

Todos esses recursos estão disponíveis para o desenvolvedor do Visual Studio, o que torna essa página inicial um bom ponto de partida para muitas pessoas. No entanto, se você quiser adicionar algo a essas opções ou substituir por completo o conjunto de páginas, a fonte integral da página padrão está disponível em `\Program Files\Microsoft Visual Studio.NET\HTML`. Porém, aí vai um aviso – as páginas-padrão não são simples, e seria fácil danificá-las sem que possam ser reparadas. Faça uma cópia do diretório como backup antes de começar a personalizar!

FIGURA 2.3

Informações relevantes para programadores do Visual Basic .NET podem com frequência ser encontradas no site da Web msdn.microsoft.com, que é realçado automaticamente em seções da página inicial do Visual Studio.



A Janela Principal do IDE do Visual Studio

Mesmo sendo tão útil, essa página inicial é apenas uma das várias janelas disponíveis como parte do Visual Studio, e iremos percorrer a maioria delas nesta lição. Com o perfil Visual Basic Developer selecionado, você já terá diversas janelas visíveis: o navegador da Web interno discutido anteriormente, o Solution Explorer, Properties e Toolbox no lado esquerdo. As outras janelas, que abordaremos nesta lição, mas que não se tornam visíveis no perfil atual por padrão, são:

- Object Browser
- Command/Immediate
- Task List
- Class View
- Server Explorer

Há outras janelas que abordarei quando se tornarem importantes, mais adiante no livro.

Recursos Comuns das Janelas

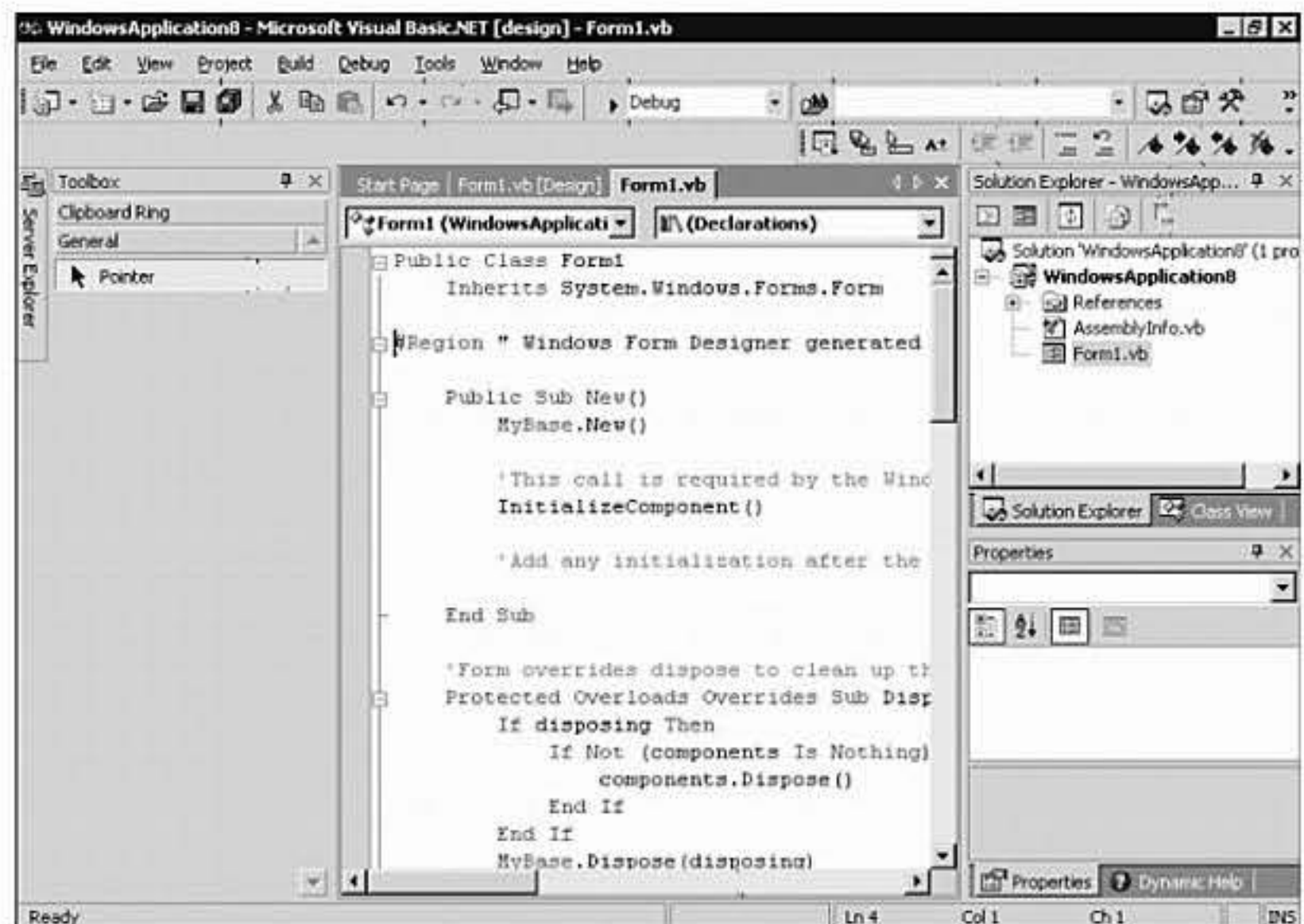
Todos os recursos destas janelas, incluindo o conceito de múltiplas janelas, foram desenvolvidos para proporcionar uma utilização mais eficiente do espaço. Em qualquer IDE, e principalmente no Visual Studio .NET, a quantidade de opções e ferramentas disponíveis é quase infinita, mas a área de seu monitor não é. Uma solução para esse problema é utilizar um monitor de 21 polegadas (ou maior!). Por alguma razão desconhecida, essa medida ainda não foi adotada, portanto, outros métodos foram desenvolvidos. Um deles é dividir as opções disponíveis em janelas diferentes, que é a abordagem usada pelo Visual Studio .NET. Agora, o objetivo é tornar o posicionamento dessas janelas fácil e flexível o suficiente para que os desenvolvedores criem seu ambiente ideal de trabalho. Elas são chamadas individualmente de *janelas de ferramentas*. Todos os membros de uma janela de ferramentas compartilham alguns recursos comuns, como a capacidade de estar encaixados/desencaixados, ocultos, associados em grupos de guias de janelas e assim por diante. Todos eles podem ser redimensionados de várias formas.

Encaixando/Desencaixando

No perfil Visual Basic Developer, tanto o Solution Explorer quanto Properties ficam alinhados no lado direito da tela do Visual Studio, e Toolbox fica no lado esquerdo (veja a Figura 2.4). O posicionamento de uma janela rente à outra área é descrito como *encaixar* essa janela. Quando está encaixada, a janela é bloqueada por essa área em um ou dois lados (dois lados, se estiver encaixada em um canto). Todas as janelas de ferramentas do Visual Studio .NET podem ser encaixadas e não há limite para as margens nas quais é possível alinhá-las.

FIGURA 2.4

Quando o Visual Studio .NET é configurado com o perfil de desenvolvedor do Visual Basic, ele organiza suas janelas de forma aproximada à do IDE do Visual Basic 6.0.

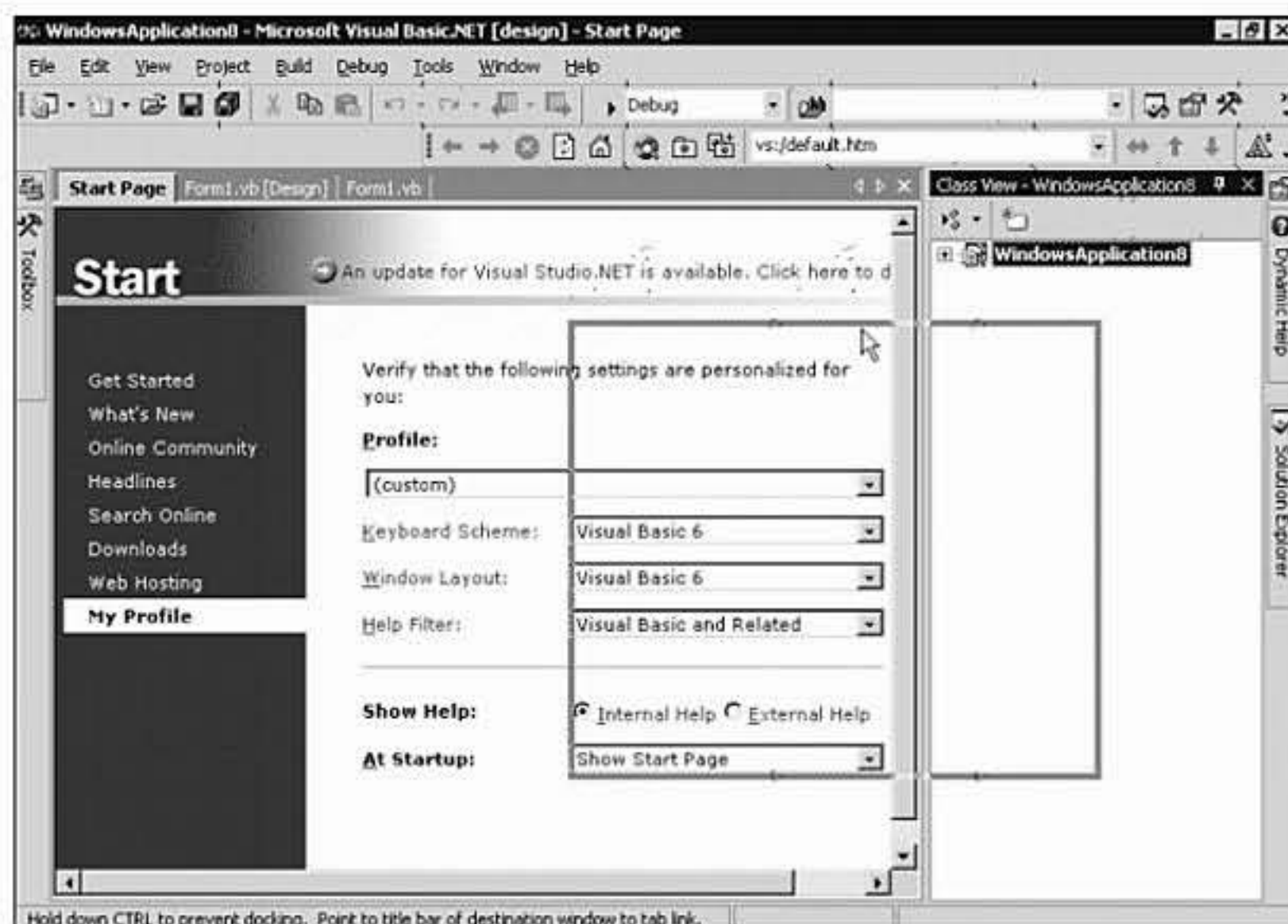


Para mover uma janela encaixada para outra área da tela, deve-se dar um clique e manter pressionado o botão do mouse na barra de título da janela de ferramentas e, em seguida, arrastá-la para a

nova posição. Enquanto ela estiver sendo arrastada, um contorno aparecerá na tela para mostrar onde a janela seria posicionada se você soltasse o botão do mouse. Para encaixar a janela em outro lado do IDE, apenas arraste-a para uma borda dele, mantendo o botão do mouse pressionado e soltando-o quando o contorno da janela mostrar o resultado desejado. No trajeto, pode-se ter a impressão de que o contorno não foi encaixado em nenhuma parte do IDE (veja a Figura 2.5).

FIGURA 2.5

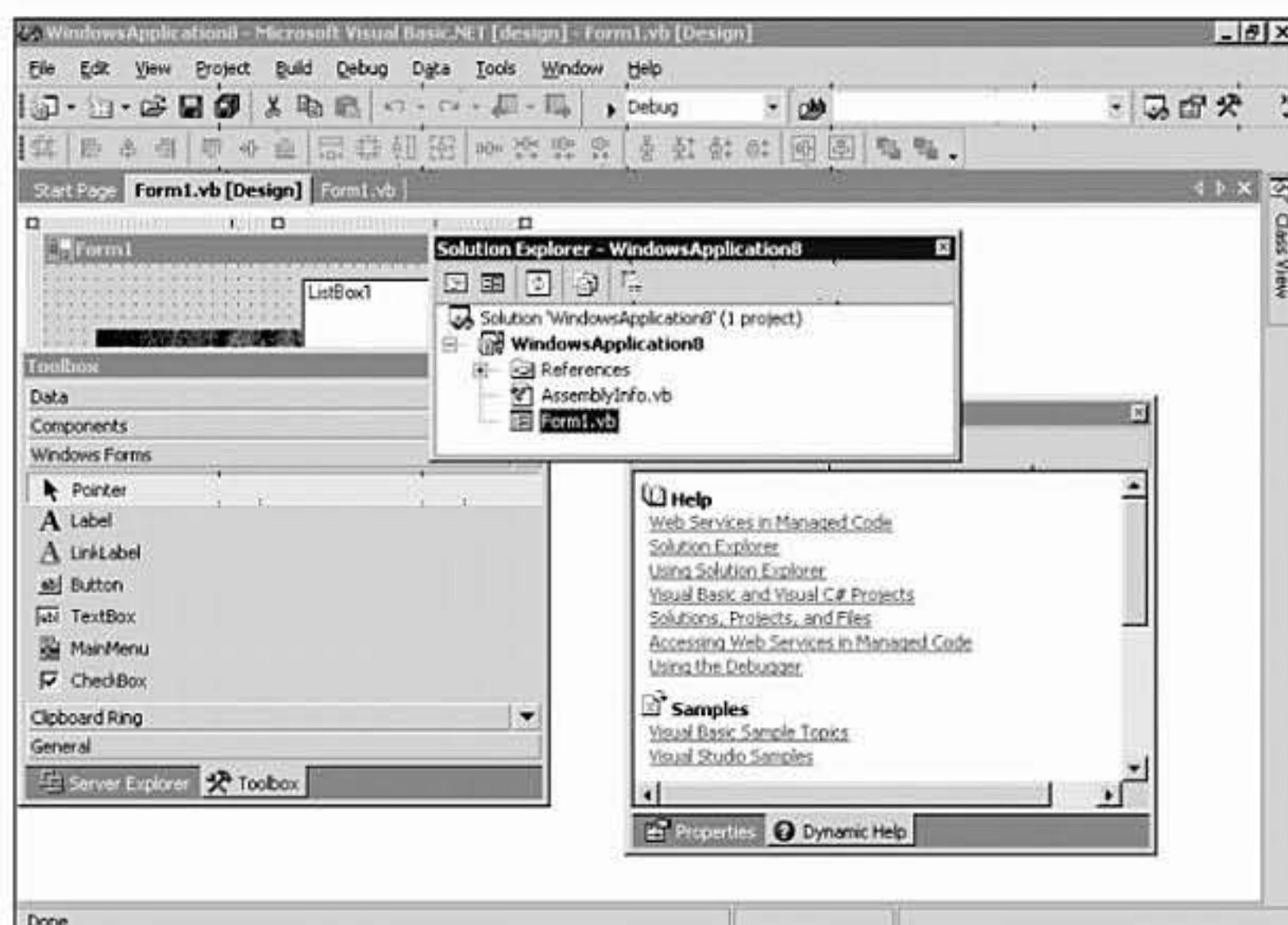
O contorno de uma janela que está sendo arrastada se altera para indicar o que ocorreria se você parasse de arrastá-la em um lugar específico.



Se você deixasse a janela de ferramentas nesse ponto, ela ficaria ‘desencaixada’, o que também é conhecido como janela *flutuante* (veja a Figura 2.6), posição na qual ela não estaria mais fixada a nenhuma borda do IDE.

FIGURA 2.6

Se você tiver bastante espaço na tela, poderá deixar suas janelas desencaixadas.



Quando as janelas estiverem desencaixadas, você poderá reencaixá-las seguindo o mesmo procedimento que acabou de usar.

**NOTA**

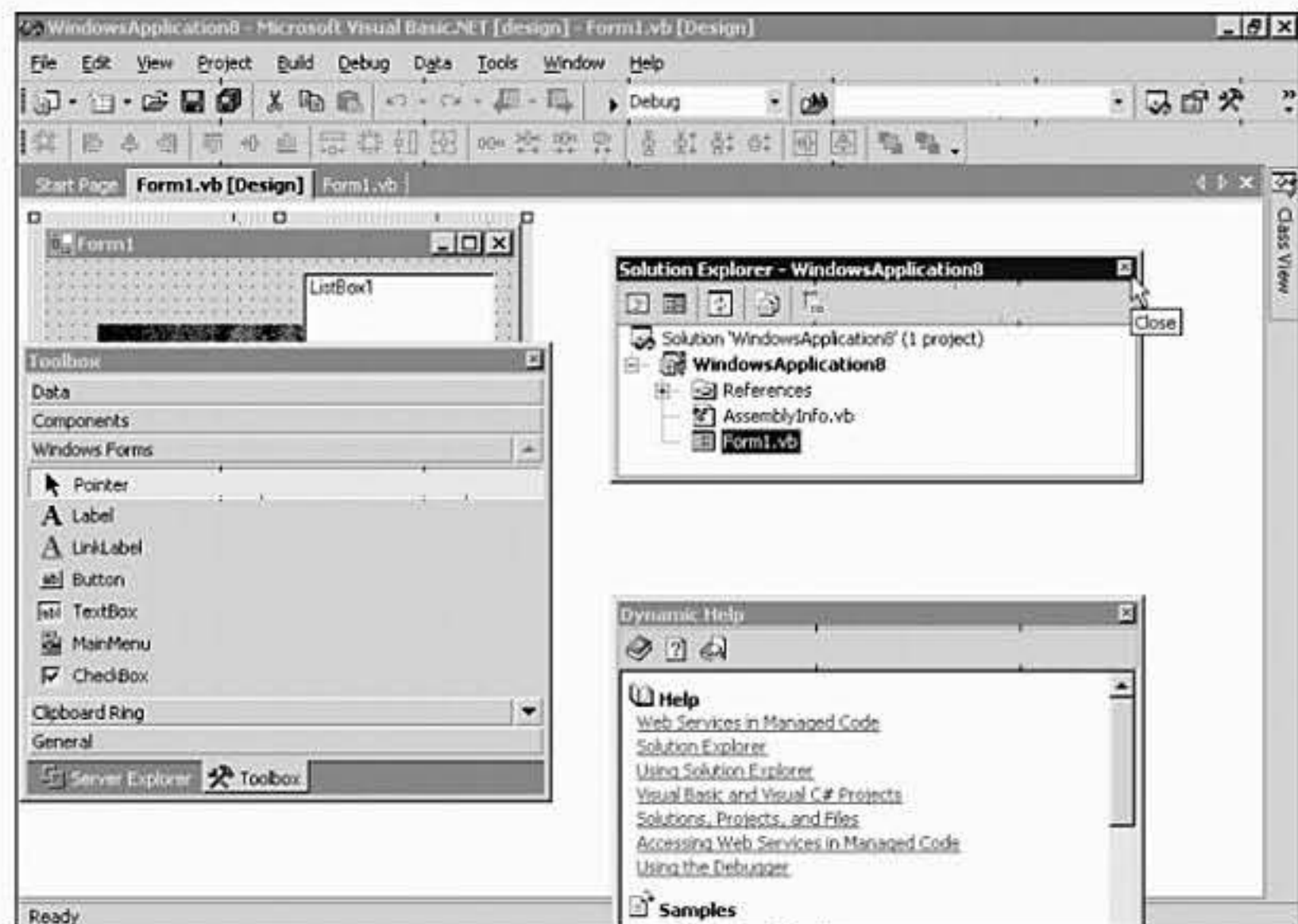
Pode ser complicado levar uma janela encaixada de volta ao local exato do qual a moveu, portanto, aí vai uma pequena dica: em vez de arrastar a janela para encaixar e desencaixá-la, dê um clique duplo na barra de título. Isso desencaixará a janela de ferramentas que estiver encaixada ou levará uma janela flutuante de volta a seu local original.

Ocultação

Encaixadas ou não, as janelas ainda ocupam espaço em sua tela, mas é possível ocultar ou fechar janelas específicas que você não queira que permaneçam visíveis. Toda janela de ferramentas possui um botão X (veja a Figura 2.7), semelhante ao ícone comum de fechamento de outras janelas. Dar um clique nesse botão fechará a janela de ferramentas, removendo-a da tela.

FIGURA 2.7

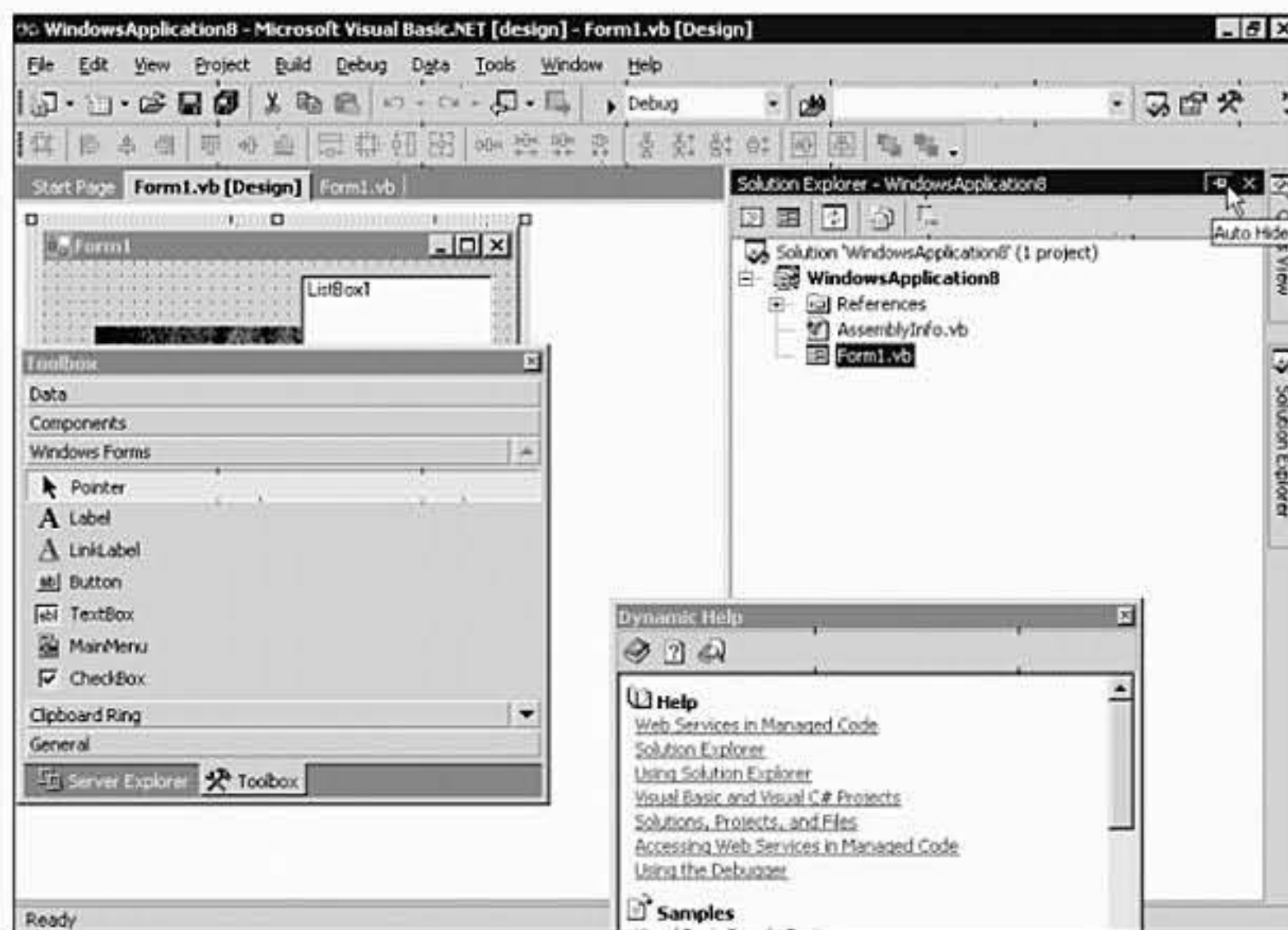
As janelas de ferramentas podem ser fechadas por meio do pequeno botão X, que as ocultará da tela até que sejam necessárias novamente.



Para trazer a janela de volta, você precisará usar uma opção do menu. Por exemplo, se der um clique no botão X do Solution Explorer, esse desaparecerá da tela. Para trazê-lo de volta, é preciso selecionar Solution Explorer no menu View (ou pressionar Ctrl+R). Não é tão difícil, principalmente para uma janela que não é utilizada com frequência, mas e aquelas que podem não estar sendo usadas agora, porém, em geral, o são? O Visual Studio .NET fornece uma maneira de recuperar o espaço da tela utilizado por essas janelas, tornando-as ainda assim fácil de acessar – o recurso Auto-Hide. Toda janela de ferramentas encaixada possui um ícone em sua barra de título que se parece com um pequeno pino (veja a Figura 2.8).

FIGURA 2.8

Todas as janelas de ferramentas possuem um pequeno pino que permite que sejam confinadas em uma posição onde ficam abertas.



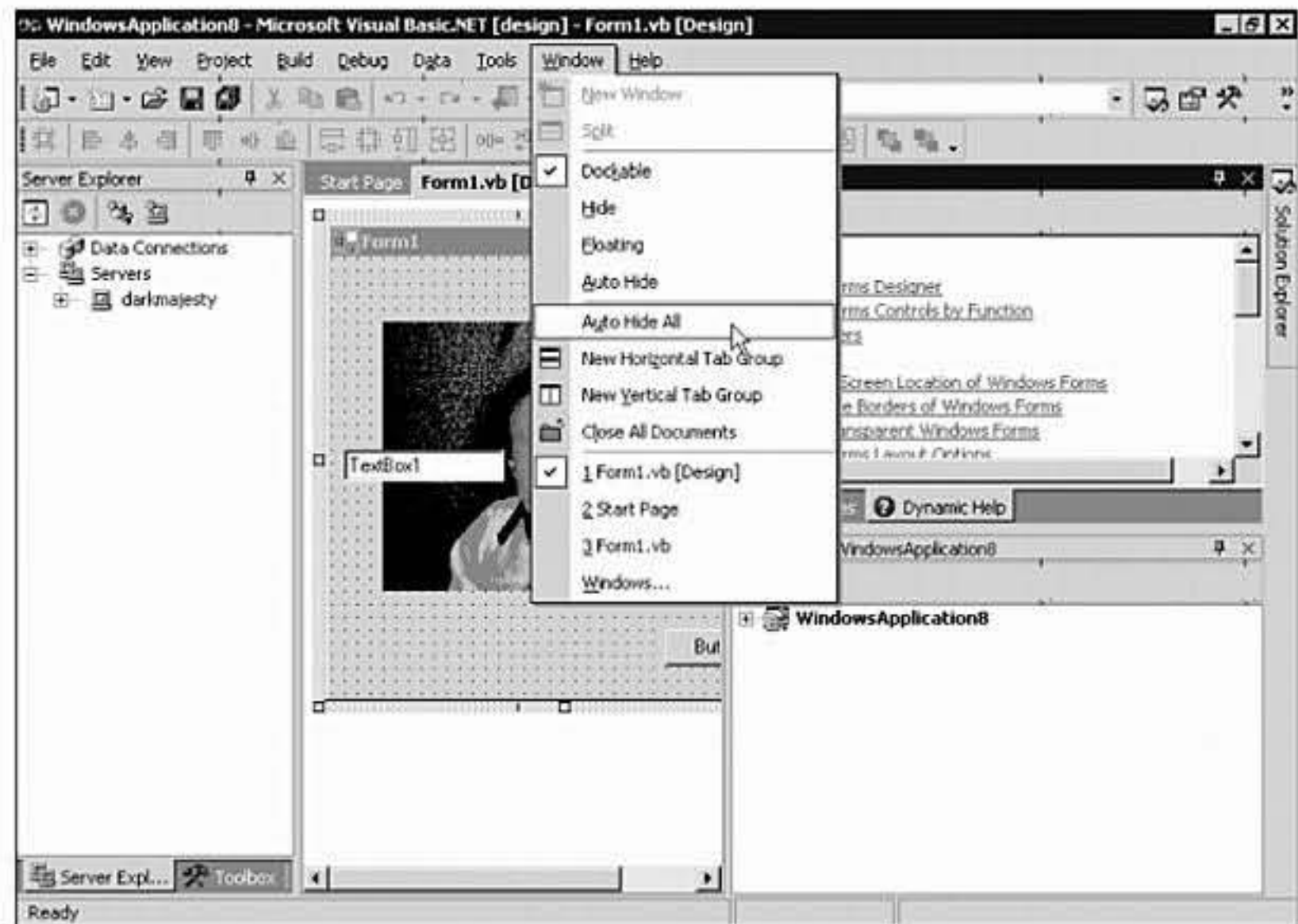
Esse é um botão de dois estados que controla se a janela deve ou não ser automaticamente oculta quando não estiver em uso. Por padrão, a janela do Server Explorer é configurada para empregar esse recurso e, portanto, aparece apenas como uma guia ou botão cinza no lado esquerdo do IDE. Passar o ponteiro do mouse por cima dessa guia fará com que o Server Explorer se torne visível por completo, deslizando para a tela. Qualquer janela de ferramentas com a opção Auto-Hide ativada será minimizada automaticamente em uma guia na lateral do IDE onde estiver encaixada. A janela oculta será trazida à tela sempre que o usuário passar o mouse sobre essa guia, o que significa que só ocupará espaço quando for necessária. Esse recurso, junto da opção do menu Window, Auto Hide All, é utilizado por mim para configurar o Visual Studio com a aparência que desejo (veja a Figura 2.9). Isso deixa o máximo de espaço para a janela principal, onde o código aparecerá quando você tiver aberto um trabalho. Uma interface extrema como essa pode ser desnecessária quando se trabalha em um monitor grande, mas, como usuário de laptop, a acho perfeita.

Guias

Outro recurso para economizar espaço: as janelas de ferramentas múltiplas podem ser agrupadas em uma única área da tela onde automaticamente se transformarão em guias individuais de uma janela com várias guias (veja a Figura 2.10). No perfil Visual Basic Developer, diversas janelas já foram agrupadas (o Solution Explorer compartilha o espaço com a janela Class View, e Properties e Dynamic Help foram configuradas de maneira semelhante), mas qualquer combinação de janelas de ferramentas é possível.

FIGURA 2.9

Você pode usar Auto-Hide para maximizar sua área central de trabalho, que é onde editará tanto códigos quanto objetos visuais.



2

FIGURA 2.10

As guias permitirão que você tenha muitas janelas diferentes de ferramentas abertas enquanto ocupam o mesmo espaço precioso da tela.



Para adicionar uma janela a um grupo de guias ou criar uma janela com guias, arraste uma janela de ferramentas (dando um clique em seu título e arrastando) para outra, soltando o botão do mouse quando o contorno se alterar para mostrar uma janela com guias, exibindo a extensão de uma pequena guia na parte inferior dele. Consegue-se a remoção de uma janela de maneira parecida. Apenas arraste uma das guias para fora do grupo até que o contorno perca o formato de guia e, em seguida, arraste-a para o novo local desejado e solte.

Redimensionando

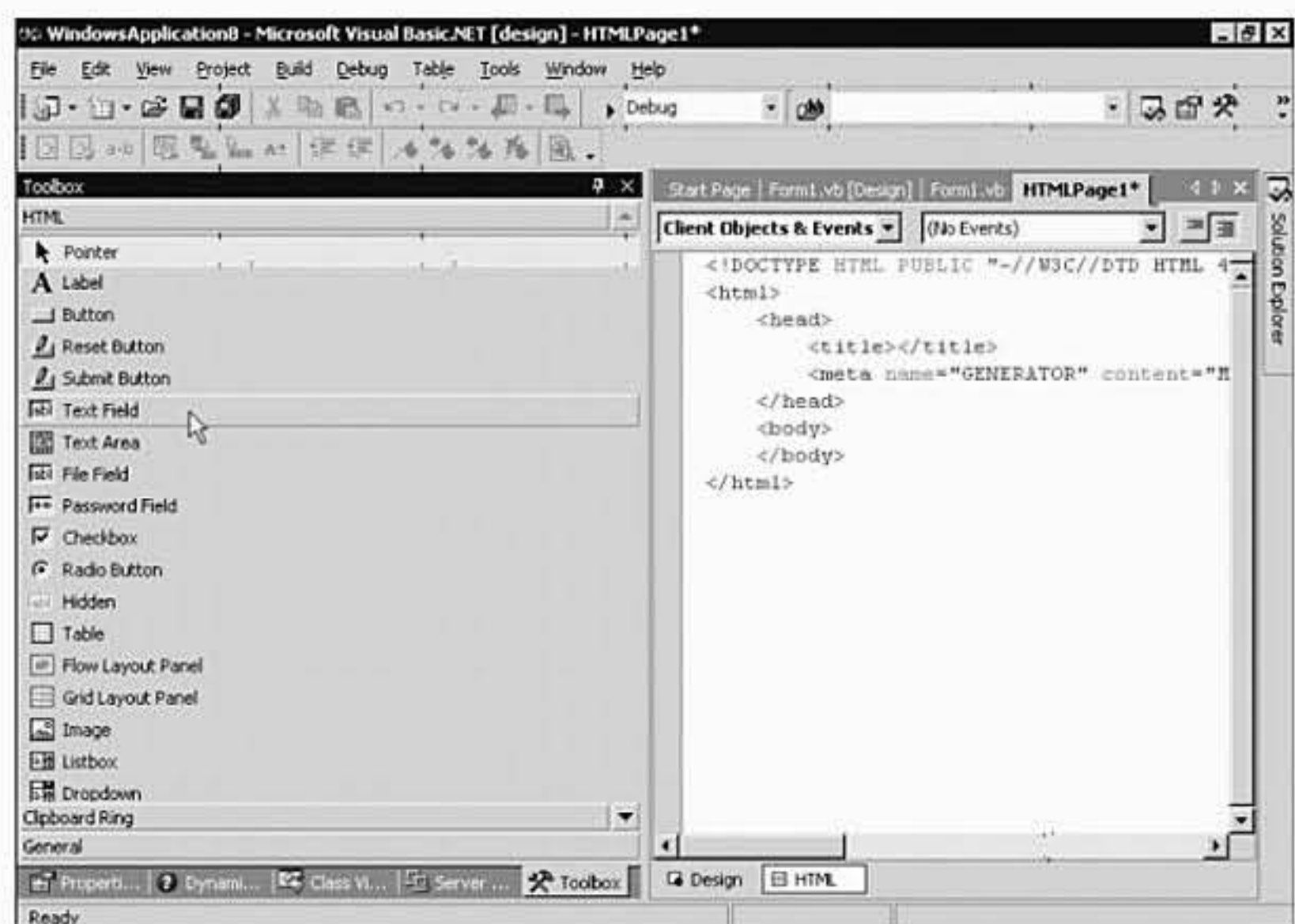
Qualquer janela de ferramentas pode ser redimensionada, mas, se estiver encaixada, só o poderá ser ao longo de suas bordas desencaixadas. Para redimensionar uma janela, mova o ponteiro do mouse sobre a margem dela, até que ele se transforme em um indicador de redimensionamento, mostrando a direção permitida para a alteração. Quando o ponteiro indicar que você está com o mouse na posição certa, dê um clique e arraste para estender ou retrair a borda da janela até o tamanho desejado. Observe que o redimensionamento entre duas janelas encaixadas estará na verdade alterando ambas porque uma deve encolher para permitir que a outra se expanda.

Toolbox

Uma das janelas mais usadas, a Toolbox (caixa de ferramentas) fornece uma listagem de vários trechos de texto, elementos de interface com o usuário e outros itens que são disponibilizados para serem adicionados aos projetos. A seleção de itens mostrados dependerá do que estiver sendo editado no momento na janela principal do IDE. Por exemplo, se nada ou apenas o navegador da Web for selecionado na janela principal, então, o único item disponível na Toolbox será o ícone Pointer. Esse ícone, que está sempre presente, é fornecido como uma maneira de desmarcar qualquer outro item da janela Toolbox. Se você estiver editando algo, como o código HTML da página inicial do Visual Studio (dê um clique com o botão direito do mouse no navegador da Web e selecione Exibir código fonte), guias complementares serão adicionadas à janela Toolbox. No caso da edição do código, uma guia HTML foi adicionada (veja a Figura 2.11) contendo vários itens que representam diferentes tags HTML.

FIGURA 2.11

Na edição de código HTML, todos os elementos de UI típicos da linguagem ficam disponíveis na Toolbox.



Qualquer item, com exceção do especial, Pointer, pode ser usado de uma das duas maneiras:

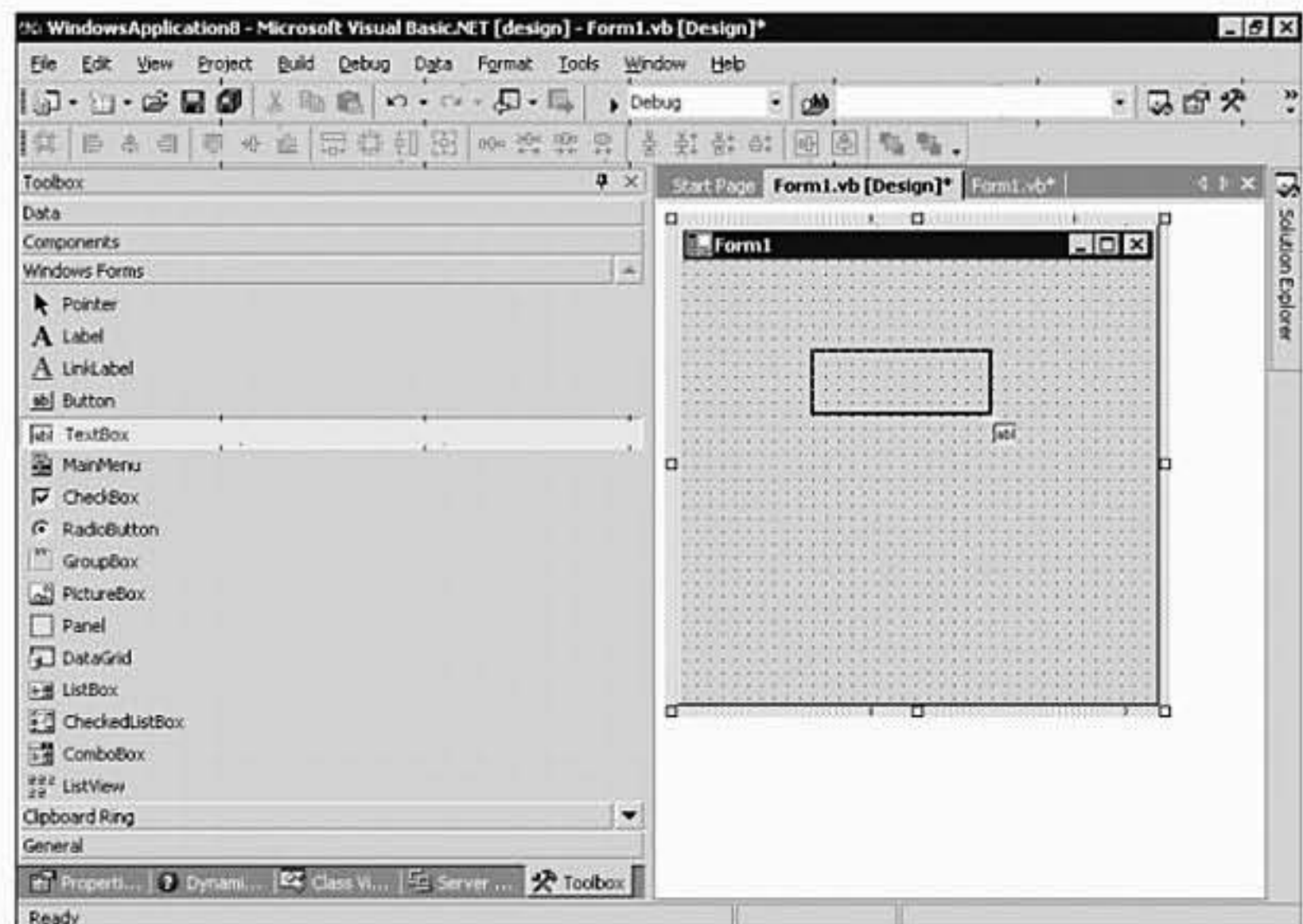
- Dê um clique e arraste o item para a janela de edição (soltando quando você tiver movido o mouse para o local desejado).
- Dê um clique duplo no item, o que fará com que seja adicionado à janela de edição no ponto de inserção selecionado (qualquer que seja o local onde o cursor estiver posicionado na janela de edição quando houver o clique duplo sobre ele).

Esses dois métodos de uso dos itens estão disponíveis para qualquer tipo de documento que estiver sendo editado, mas, se uma interface gráfica com o usuário (algum documento visual sem texto como um formulário Windows) for o documento atual, então, as duas opções anteriores se comportarão de maneira um pouco diferente. Uma terceira opção também está disponível:

- Dar um clique e arrastar o item para o documento visual funciona da mesma forma que para um texto, mas, em vez de aparecer um ponto de inserção do cursor, será mostrado um contorno real do item quando você mover o mouse sobre o documento.
- Dar um clique duplo também funciona, mas, já que um documento visual nem sempre possui um ponto de inserção selecionado naquele momento, o novo item em geral é apenas criado no centro do documento.
- Uma terceira opção, que não está disponível para edição de texto, é selecionar o item dando um clique nele, realçá-lo na Toolbox e, em seguida, dar um clique e arrastá-lo para o documento visual. Isso esboçará o tamanho e o local no qual você deseja que haja a inserção no documento, e o novo item será criado de acordo com o definido (veja a Figura 2.12).

FIGURA 2.12

Você pode desenhar elementos visuais, como caixas de texto, em um formulário depois que os tiver selecionado na Toolbox.





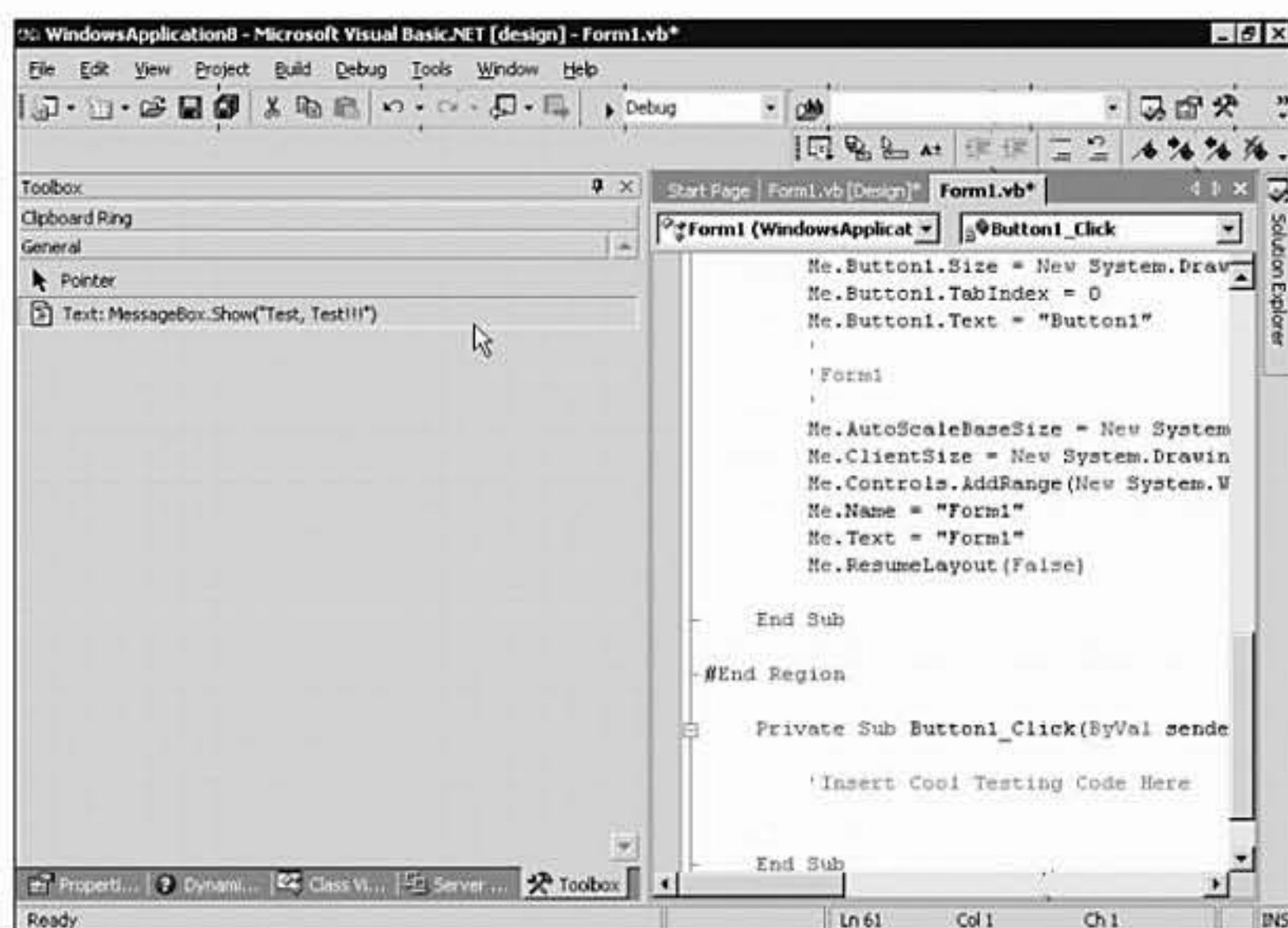
Há um pouco de imprecisão nas descrições anteriores, que podem fazer você pensar que a Toolbox é mais difícil de usar do que parece. Bem, a falta de clareza é resultado da natureza do IDE do Visual Studio, que foi projetado para atender a uma grande variedade de linguagens de programação, nem todas disponíveis hoje. Portanto, seu comportamento exato (ou de qualquer parte dele, como a Toolbox) é difícil de descrever. Mas, pode estar certo de que, no aspecto geral, ele sempre funcionará como o descrito, independentemente da linguagem utilizada.

Apresentarei de modo detalhado um exemplo de uso da Toolbox com um documento visual posteriormente nesta lição, quando você criar um formulário Windows como parte de seu primeiro aplicativo empregando o IDE. A Toolbox pode conter qualquer trecho arbitrário de texto, além de fornecer alguns para a edição de HTML, desenvolvimento de formulários e muitos outros tipos de trabalho, o que significa ser possível gerar seus próprios itens que representem seções do texto. Esse recurso é uma maneira útil de selecionar um fragmento de texto (que pode ser, e provavelmente será, código) que se espera utilizar com frequência e torná-lo facilmente disponível.

Para conseguir esse excelente feito de produtividade, selecione o texto desejado na janela de edição (o que pode incluir primeiro a digitação do texto) e arraste essa opção para a janela da Toolbox. Independentemente da guia que você arrastar, o texto dela determinará onde seu item aparecerá. Como podemos ver na Figura 2.13, o item apresentará um nome inoportuno e bastante sem sentido, como HTML Fragment, mas é possível dar um clique nele com o botão direito do mouse e selecionar *Rename Item* para fornecer uma descrição apropriada. Voilà! Agora temos um novo item personalizado na Toolbox, que pode ser usado sempre que se desejar apenas arrastando-o para a janela de edição.

FIGURA 2.13

Códigos, linguagem HTML ou outros trechos de texto podem ser inseridos na Toolbox e, em seguida, usados (arrastados para a janela de edição de códigos e HTML) exatamente como qualquer outro controle dela.



A Toolbox, assim como o resto do IDE, possui muitas opções complementares que não abordarei, como a capacidade de adicionar mais guias (guias são aquelas seções deslizantes da janela Toolbox), renomeá-las e alterar a visualização da lista de ícones de qualquer seção. Esses outros recursos podem ser acessados por meio do menu suspenso da Toolbox (o menu que é acessado com um clique no botão direito do mouse) e também estão documentados nos arquivos de ajuda do IDE.

Command/Immediate

Você já se surpreendeu executando o console de comando (ou a janela do DOS, como alguns gostam de chamá-lo) para efetuar uma tarefa? Para muitas pessoas, algumas tarefas só podem ser processadas mais rapidamente com o uso do teclado e de uma interface de linha de comando em vez de se empregar o mouse para navegar pelos ícones, menus e caixas de diálogo. Já que a produtividade do programador é o objetivo final, qualquer método que puder ser mais veloz vale a tentativa. O Visual Studio .NET possui uma janela que fornece duas formas com base no console de interagir com o IDE. Uma fez parte do Visual Basic por algum tempo, a janela Immediate e a outra esteve presente no Fox Pro por muitos anos e acabou adicionada ao Visual Studio, a janela Command.

Em termos de utilização, elas são realmente duas janelas, mas foram associadas para tornar a operação um pouco mais confusa. Você pode considerá-las como duas janelas (sei que vou fazer isso), depois que tiver aprendido dois itens essenciais: como alternar o modo da janela (de Command para Immediate e vice-versa) e como informar em que modo a janela está atualmente. Primeiro, o prioritário – tornemos essa janela visível; selecione View, Other Windows e Command Window no menu, exibindo assim essa nova janela.

Essa janela, agora com o nome de Command Window, deve conter apenas uma linha em branco seguida de um prompt `>` (quase exatamente como o console de comando ou o prompt do DOS com o qual está acostumado). Ela já está no modo Command, e você pode inserir o comando que quiser e executá-lo pressionando Return. Para passar essa janela para o modo Immediate, apenas digite o comando `immed` (e pressione Return ou Enter) no prompt fornecido. Agora, a janela terá passado para o modo Immediate, distinto do modo anterior pelo acréscimo de `– Immediate` em sua barra de título e pela remoção do prompt `>` de seu texto efetivo. Para retornar ao modo Command, digite `cmd` e pressione Return (sim, você mesmo precisa incluir o prompt `>`). Agora que sabemos como alternar rapidamente entre esses dois modos, podemos examinar a finalidade e o uso de cada um.

O modo Command dessa janela permite que você controle o IDE por meio de comandos digitados – por exemplo, digitando `File.New Project` para realizar a mesma tarefa que seria executada se selecionasse os itens File, New e Project no menu. Uma interface do console em geral pode ser mais veloz do que uma interface gráfica com o usuário. Essa duplicação de funcionalidade é fornecida como uma maneira potencial de acelerar seu trabalho dentro do IDE. Uma grande quantidade de comandos está disponível, mas o modo mais rápido de encontrar muitos deles é

percorrer os nomes dos menus visíveis. Depois que tiver digitado um nome (como **Edit** ou **File**), adicione um ponto, e será mostrada uma lista suspensa com os comandos disponíveis para esse nome de menu. Aqui está uma lista curta de comandos que é bom conhecer:

```
File.NewProject  
File.SaveAll  
Window.AutoHideAll
```

A janela Immediate fornece a capacidade de avaliar as instruções do código diretamente (de imediato!). Isso permitirá que você insira uma única linha de código e veja os resultados sem ter de criar um protótipo inteiro do projeto. Esse recurso é útil quando se está no modo de interrupção, em que se interrompe a execução de um programa em curso. Iremos desenvolver um exemplo simples de projeto, no qual usaremos um ‘ponto de interrupção’ para provocar o encerramento da execução do programa em uma certa linha de código. (O ponto de interrupção será abordado com mais detalhes no Dia 6, “O Que Fazer Quando Programas Bons Apresentam Problemas e para Se Certificar de Que Isso Não Aconteça”, porque ele é realmente uma ferramenta de depuração).

Você usará a janela Command no início deste exemplo apenas para ter uma noção de como ela poderá ser útil no futuro. Certifique-se de que a janela Command fique visível e no modo Command, selecionando View, Other Windows e Command Window na barra de menu. Agora, a janela Command deve estar visível com um prompt > mostrando que está no modo Command. Digite o comando **File.NewProject** e pressione Enter. Uma caixa de diálogo aparecerá, solicitando a criação de um novo projeto. Selecione a pasta chamada Visual Basic Projects na lista da esquerda e o tipo individual de projeto chamado Windows Application, na caixa à direita. Dê um clique em OK para fechar a caixa de diálogo, gerando um novo projeto em branco.

O projeto criado contém apenas um formulário Windows, e você ainda precisa adicionar seu próprio código. No entanto, o Visual Basic já inseriu de modo automático um pequeno trecho de código nele, um pouco do trabalho que é necessário para gerar e inicializar o novo formulário em branco. Esse código pode ser visto com um clique no botão direito do mouse sobre o novo formulário (na janela central do IDE) e selecionando-se View Code. Esse procedimento irá adicionar e selecionar uma guia nova na janela central, uma janela de código que exibe aquele associado a esse formulário. Já que nada foi adicionado ao formulário, o código é um pouco limitado, mas suficiente para o exemplo.

Selecione a linha `Form1 = Me`, rolando para baixo a fim de encontrá-la, se necessário. Agora, você quer marcar essa linha com um ponto de interrupção, portanto, a execução desse código será encerrada ou ‘interrompida’ quando ele a atingir. Há três maneiras diferentes de marcar a linha. Uma é dar um clique na margem (a área cinza-claro no lado esquerdo da janela do código), outra é dar um clique com o botão direito do mouse e selecionar Insert Breakpoint, e a terceira é usar o atalho do teclado para essa função, pressionando F9. Empregando o método que quiser, adicione o ponto de interrupção, e verá um ponto vermelho aparecer na margem próxima à linha. Ela indica a presença de um ponto de interrupção.

Com esse ponto de interrupção inserido, você poderá executar o projeto, e o processamento será encerrado quando ele atingir essa linha. Como para o ponto de interrupção, há três maneiras principais de iniciar um projeto: uma é usar o botão da barra de ferramentas (que se parece com o botão de reprodução dos CD players ou videocassetes – veja a Figura 2.14), outra é utilizar as opções Debug e Start do menu, e a terceira é empregar o atalho do teclado, F5. É claro que a opção a ser usada é uma preferência pessoal. Muitos programadores acham que, no final das contas, os atalhos do teclado são a forma mais fácil de acessar as funções mais comuns.

Quando você iniciar a execução do projeto, ele será rapidamente encerrado e exibirá a linha de código marcada com um ponto de interrupção. Agora, estamos no modo de interrupção, como indicado por [break] na barra de título do IDE do Visual Studio. A seta amarela que pode ser vista na margem da janela do código indica a linha que está para ser executada (processada). Nesse ponto, pode-se alterar a janela Command para o modo Immediate e testá-lo.

Se sua janela Command estava visível antes da execução do projeto, ainda deve estar presente, embora o layout possa ter sido um pouco alterado quando certas janelas foram abertas automaticamente no momento em que você estava no meio do processamento do projeto. Se a janela Command não estiver visível, abra-a usando as opções View, Other Windows e Command Window do menu.

FIGURA 2.14

A barra de ferramentas fornece botões para iniciar e interromper a execução de programas, usando ícones parecidos com os controles de videocassetes.



Dê um clique na janela para selecioná-la (tornando-a a janela ativa no IDE) e digite **immed** (seguido da tecla Enter) para passá-la para o modo Immediate. Agora você pode inserir qualquer instrução do Visual Basic, e ela será avaliada imediatamente (daí o nome). Tente as instruções a seguir:

```
? Me.Width
Me.Width = 50
? Me.Width
? 3 + 5
? 3 = 5
```




Usar as teclas de seta para cima e seta para baixo enquanto estiver na janela Command/Immediate nem sempre o transferirá para outra linha dela. Em vez disso, se você já tiver começado a inserir algum texto, poderá passar pelos comandos executados. Se você selecionar uma linha que já passou (na janela) e começar a adicionar texto nela, será criada de modo automático uma cópia dessa linha, com suas novas alterações, na parte inferior da janela. Isso torna qualquer texto antes da última linha da janela efetivamente de leitura.

Notou o ponto de interrogação em frente a algumas das instruções anteriores? Isso indica 'exibir' e, sem ele, o Visual Basic não saberá o que fazer com as instruções que retornam um valor. Por exemplo, `3 + 5` produzirá 8, mas, sem a instrução de exibição, 8 não será um comando válido do Visual Basic. Por outro lado, instruções como `Me.Width = Me.Width * 2` são trechos de código válidos do Visual Basic e não precisam do ponto de interrogação em sua frente.

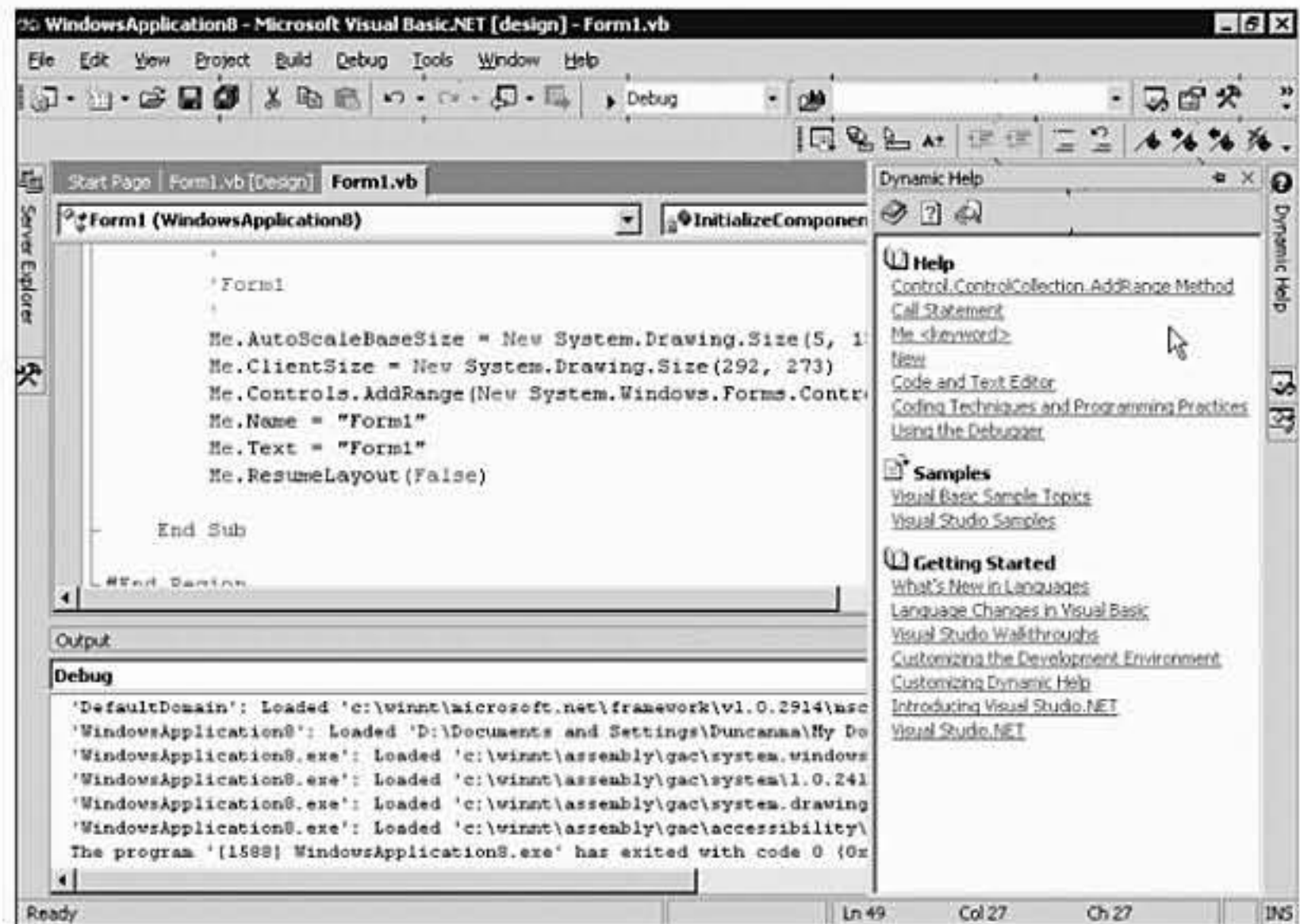
Pressione F5 para fazer com que a execução do código continue após o ponto de interrupção, e o formulário aparecerá na tela maior do que o tamanho original se você processou as instruções do exemplo fornecido anteriormente. Como pode ver, é possível afetar partes de seu programa a partir da janela Immediate, o que a faz uma ótima ferramenta de depuração.

Dynamic Help

Esta janela de ferramentas é configurada como uma guia com a janela Properties (se você estiver usando as configurações do perfil Visual Basic Developer) e fornece referências, com base no contexto, da documentação dos arquivos de ajuda do Visual Studio. Em vez de esperar que você solicite ajuda, esta janela de ferramentas age de modo proativo quando a tecla F1 é pressionada ou algo é selecionado no menu Help. Com base em sua opção ou tarefa atual, ela exibirá uma lista de tópicos relacionados. Na Figura 2.15, a janela de ferramentas Dynamic Help mostra um conjunto de tópicos de ajuda sobre a tag HTML atualmente selecionada na janela de edição de códigos. Além dos tópicos relacionados, essa janela em geral exibe um link para vários tópicos mais genéricos como (nesse caso) a seção Coding Techniques and Programming Practices (Técnicas de Codificação e Práticas de Programação) da documentação. Essa janela de ferramentas também fornece um link direto para as seções de pesquisa, conteúdo e índice da documentação de ajuda por meio de três ícones da barra de ferramentas.

FIGURA 2.15

A janela Dynamic Help tenta mostrar informações úteis mesmo antes que você as solicite.



2

Server Explorer

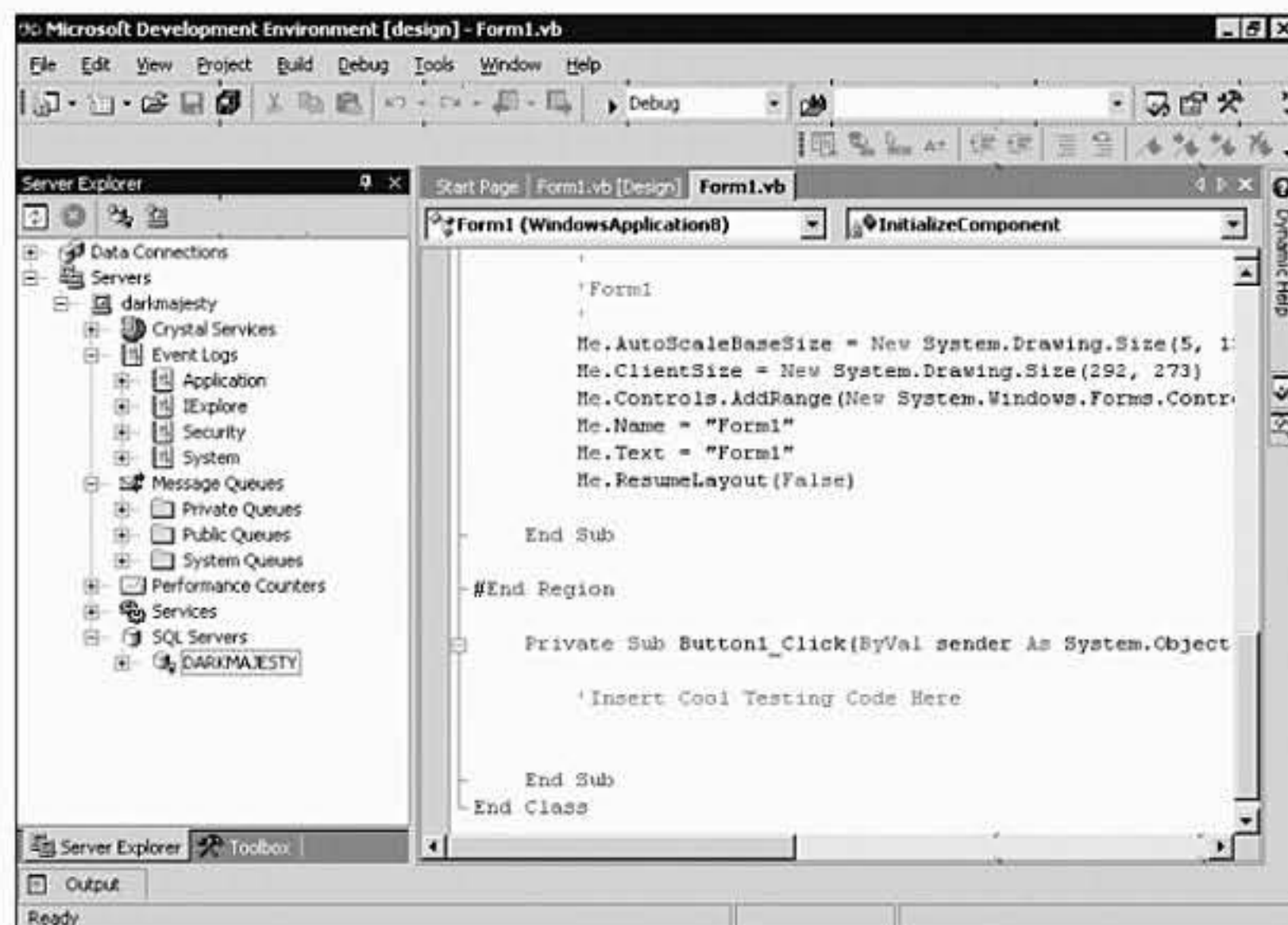
Esta janela de ferramentas (veja a Figura 2.16) fornece uma listagem visual de dois recursos essenciais, bancos de dados e servidores. O primeiro conjunto de recursos representa todas as conexões estabelecidas entre seu projeto e vários servidores de banco de dados, e permite que você explore esses bancos de dados para examinar tabelas, procedimentos armazenados e outras informações úteis.

O segundo conjunto de informações, Servers, representa qualquer máquina com a qual você possa se conectar e que forneça uma interface visual para os recursos que ela possa tornar disponível para seu programa. Esses recursos incluem contadores de desempenho, registros de eventos, filas de mensagens e mais, todos facilmente encontrados por meio desta janela de ferramentas.

O Dia 13, “Usando o Server Explorer”, se aprofunda no uso dessa janela de ferramentas para encontrar e manipular recursos de servidores.

FIGURA 2.16

O Server Explorer fornece uma maneira visual de examinar e usar recursos tanto do servidor local quanto de outras máquinas.



Properties

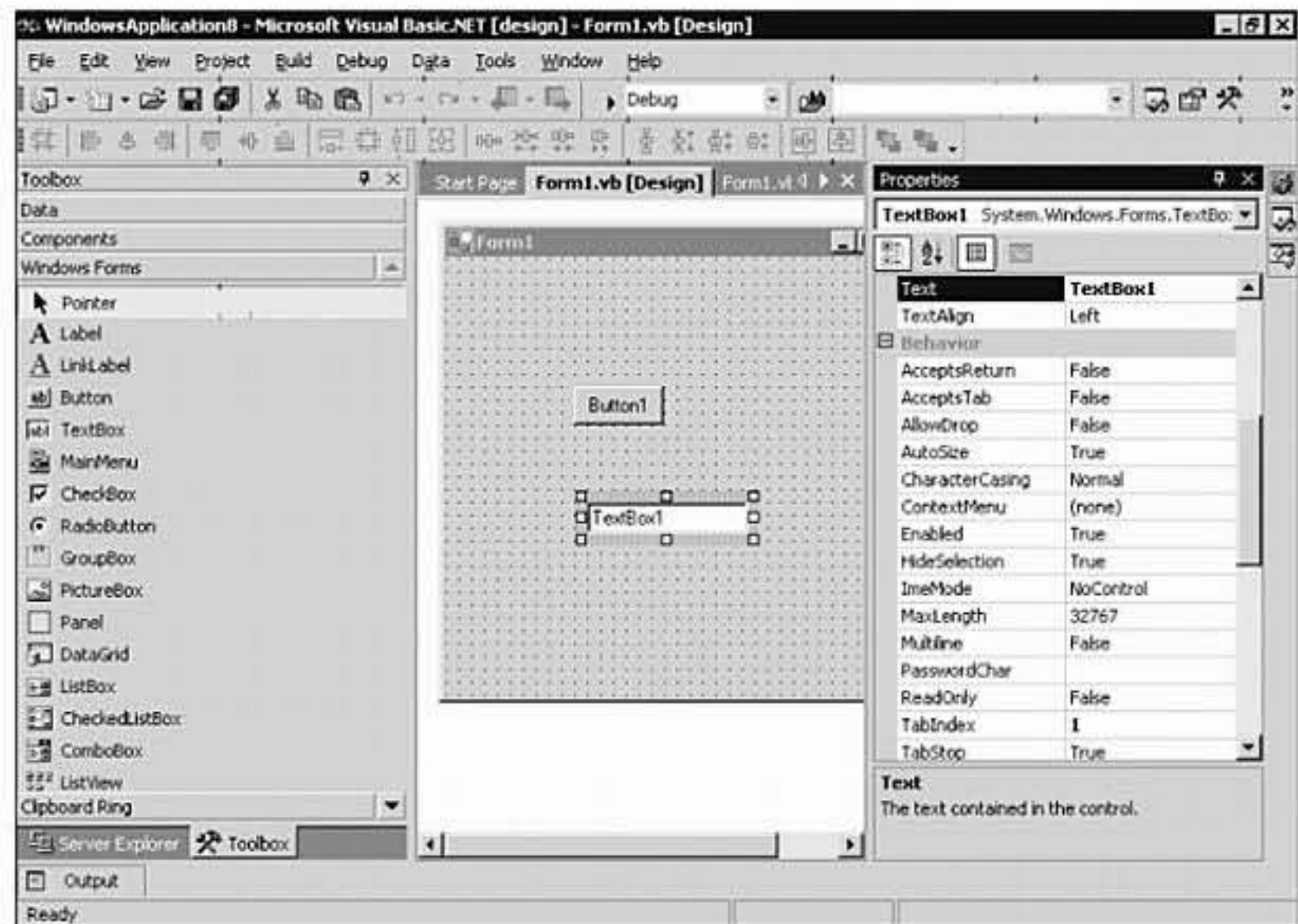
O IDE do Visual Studio permite que você trabalhe com muitos itens, projetos, soluções, formulários e classes diferentes, além de outros recursos, todos possuindo atributos e propriedades. Essas propriedades são informações que descrevem o item, como o nome de um projeto. Embora as propriedades sejam em geral preenchidas automaticamente, com valores-padrão, é preciso uma maneira de alterá-las. A janela Properties fornece essa funcionalidade. Sempre que um item for selecionado no IDE, os atributos desse objeto serão exibidos nesta janela de ferramentas (veja a Figura 2.17). Alguns desses atributos podem ser de leitura (não podem ser alterados), mas, se assim o for, então, será possível dar um clique neles na janela Properties e alterá-los quando necessário.

Solution Explorer

De muitas maneiras, o Solution Explorer é semelhante ao recurso do Windows Explorer usado no Windows. É a interface de gerenciamento de arquivos do Visual Studio .NET. No Visual Studio .NET, o código que você criar poderá ser organizado em camadas diferentes de agrupamento: soluções, projetos e arquivos. A janela Solution Explorer permite que sejam visualizados todos os objetos que estejam abertos no momento, em seus respectivos grupos ou janelas. A janela Solutions contém projetos, isto é, aplicativos e componentes individuais, como um sistema completo, incluindo componentes executados tanto no cliente quanto no servidor. Dentro de cada janela Project estão todos os arquivos efetivos do projeto (classes, formulários e outros elementos).

FIGURA 2.17

Quando um objeto for selecionado na janela de edição (uma caixa de texto em um formulário nesse caso), a janela Properties exibirá todos os seus atributos.



Além de permitir que você visualize o que está aberto no momento, a janela Solution Explorer fornece vários recursos. Por meio dela, podemos

- Adicionar novos arquivos a um projeto (dê um clique com o botão direito do mouse no projeto e selecione Add)
- Remover arquivos (dê um clique com o botão direito do mouse em um arquivo específico e selecione Remove)
- Adicionar e remover projetos inteiros de um grupo Solution (dê um clique com o botão direito do mouse na solução para adicionar um projeto e em um projeto para a opção de removê-lo da solução atual)

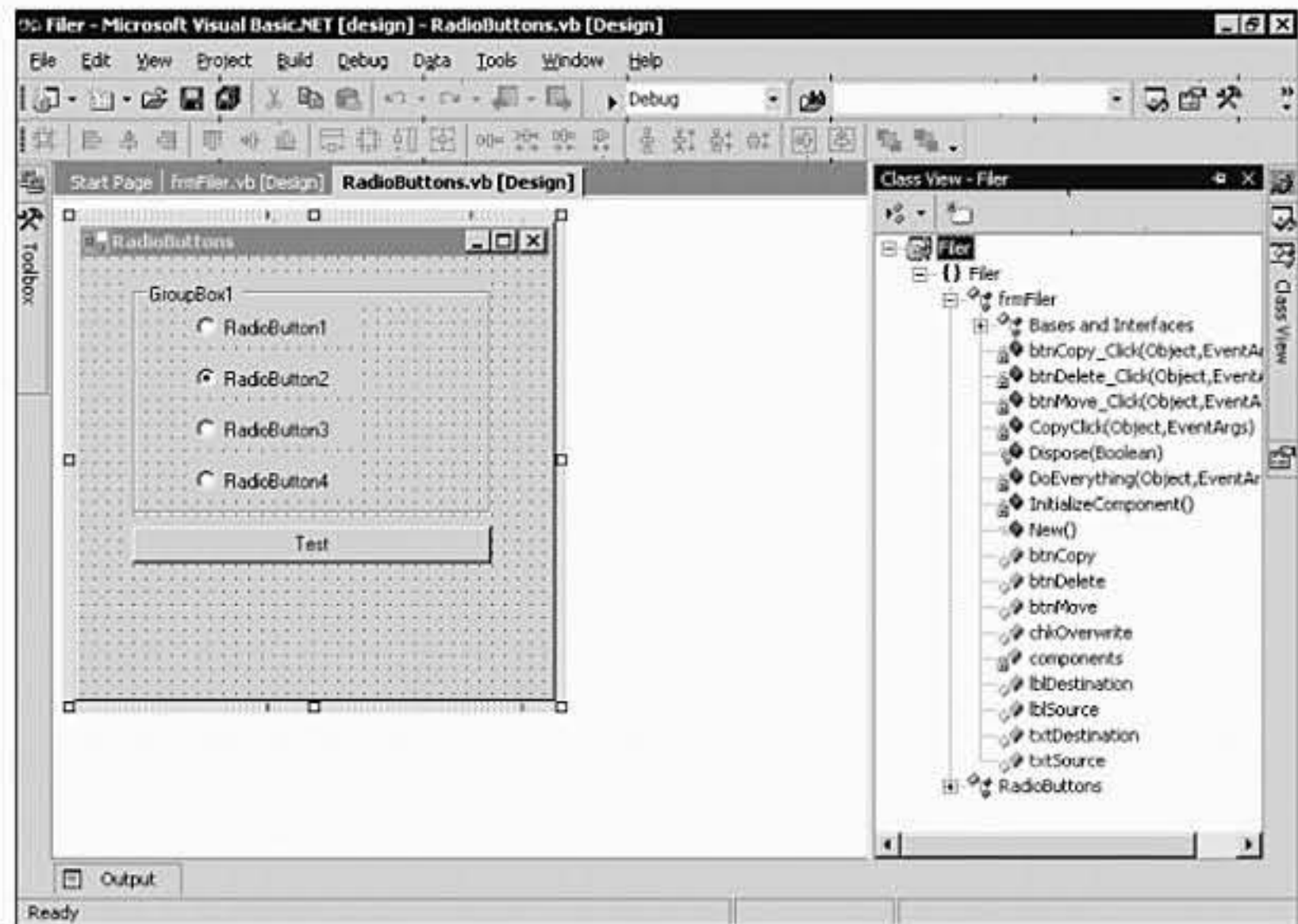
Class View

Como parte da discussão sobre o Solution Explorer, expliquei que poderia haver muitos arquivos diferentes envolvidos em um único projeto ou solução. Esses arquivos em geral correspondem às classes criadas (como uma classe Veículo ou Conta), mas não há um requisito para que a organização do arquivo se pareça com a organização conceitual das classes no projeto. A janela Class View (veja a Figura 2.18) foi projetada para permitir que você visualize a estrutura do objeto de seu projeto e use essa visualização para navegar em seu código.

Dentro desta janela, você pode minimizar e expandir os objetos exibidos, para acessar as diversas propriedades e métodos que eles expõem. Dar um clique duplo em um item específico de Class View o levará para essa classe, método, evento, propriedade ou procedimento. Se o item sobre o qual houve o clique duplo não estiver disponível como parte de seu código, Class View o conduzirá para a definição dessa parte da classe dentro do Object Browser (veja a próxima seção). A janela Class View é útil como uma maneira de examinar seu projeto através das classes definidas nele, ignorando os detalhes físicos dos arquivos reais.

FIGURA 2.18

A janela Class View mostra seu projeto organizado por seus objetos e não por seus arquivos físicos, e fornece acesso direto ao interior desses objetos.



Object Browser

Toda a programação na plataforma .NET é baseada em objetos – os objetos fornecidos como parte do .NET Framework, os que você criar e até os que outros participantes de sua própria equipe desenvolverem. Todos esses objetos possuem propriedades e métodos por meio dos quais é possível interagir com eles, mas como saber o que está disponível? O Object Browser foi projetado para ajudá-lo no trabalho com todos esses objetos, permitindo que você navegue e pesquise em um catálogo de objetos disponíveis. Esse catálogo inclui os objetos (classes) expostos por qualquer biblioteca de classes que for referenciada, além das classes contidas em seu próprio projeto. Semelhante em alguns aspectos à Class View, o Object Browser vai além da funcionalidade dessa outra janela, incluindo objetos externos ao seu projeto. Essa janela é mais útil como uma maneira de documentação ou referência, permitindo que classes sejam encontradas dentro do .NET Framework ou outras bibliotecas de classes e seus detalhes visualizados, como suas propriedades e métodos. Na Figura 2.19, podemos ver o Object Browser sendo usado para pesquisar o conteúdo da biblioteca System.Data, exibindo informações detalhadas que vão até o nível dos parâmetros necessários para uma certa chamada de método.

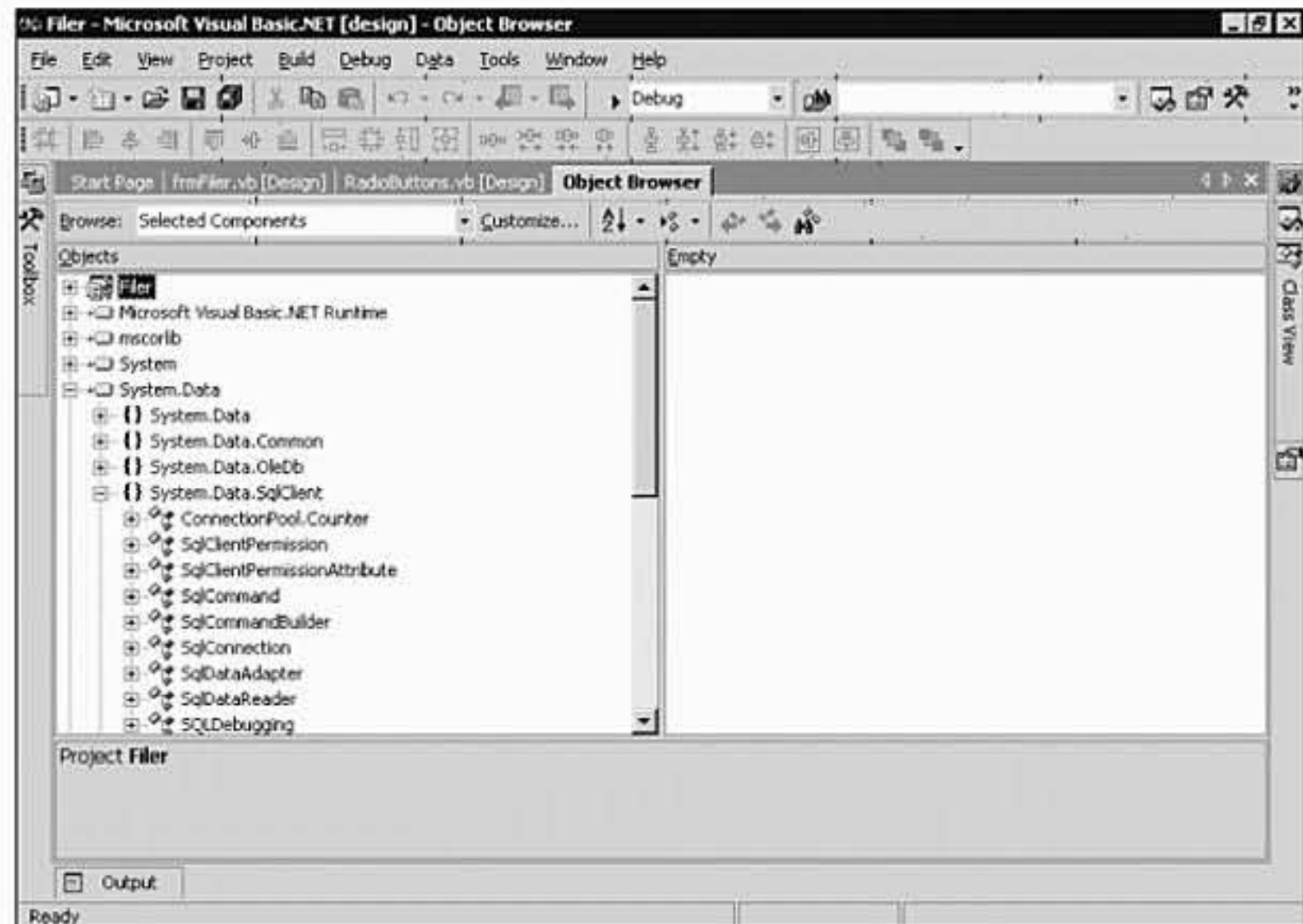
Lista de Tarefas

Em qualquer projeto de desenvolvimento, mesmo os concluídos, há a probabilidade de ficarem várias tarefas pendentes para serem completadas. As seções do programa podem precisar de ajuste no desempenho. Pode haver erros conhecidos ou recursos que necessitem ser remediados. Quando as tarefas pendentes podem ser relacionadas em uma área do código, uma prática comum entre os programadores é marcar essa área com comentários. Quando os programadores incluem consistentemente certas palavras-chave como TODO ou BUG nesses comentários, é mais

fácil varrer o código procurando-as para encontrar os trechos apropriados. O Visual Studio .NET formalizou esse processo fornecendo uma lista real de tarefas que é preenchida de modo automático com referências a qualquer seção de seu código que contenha uma das várias palavras-chave como TODO (mas, você pode especificar a palavra-chave que quiser). Cada comentário encontrado será, em seguida, relacionado em uma lista fácil de usar, detalhando não somente o próprio comentário, mas também o arquivo e a linha onde foi encontrado (veja a Figura 2.20).

FIGURA 2.19

O Object Browser permite que sejam examinadas as classes fornecidas pelo .NET Framework e de qualquer outra biblioteca que você tenha criado ou referenciado.

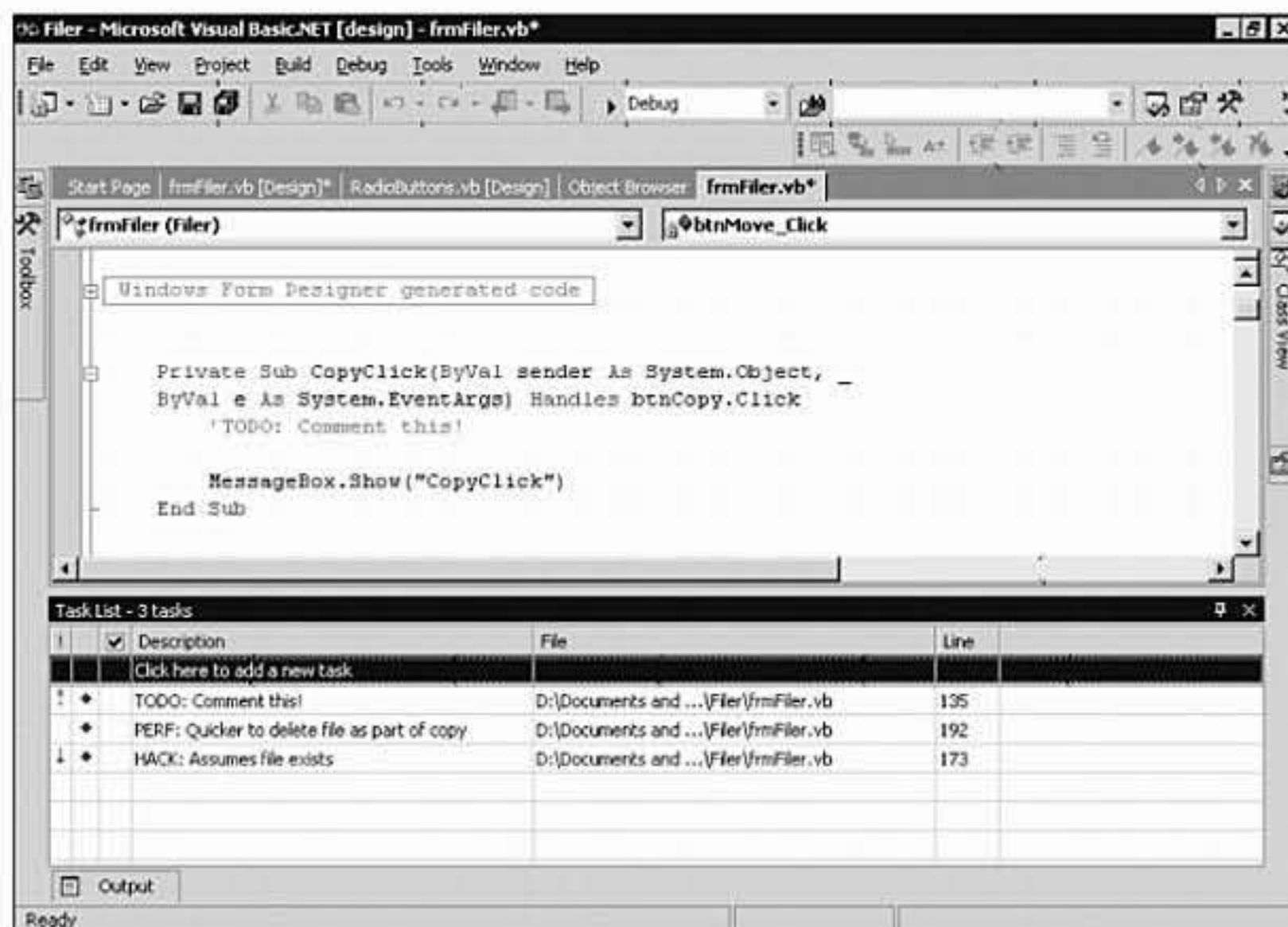
**NOTA**

Você pode adicionar suas próprias palavras-chave à lista de símbolos reconhecidos por meio da caixa de diálogo Options. Na sua seção Environment\Task List, é possível acrescentar novos símbolos e especificar configurações para a tarefa, que serão criadas quando essa palavra-chave for encontrada. As palavras-chave só serão consideradas quando forem encontradas nos comentários de seu código.

Com um rápido clique duplo em uma tarefa pendente, você será levado de imediato para o código, no qual poderá trabalhar no que ela indicar. Além dessa funcionalidade – por si só bastante útil –, a lista de tarefas pode conter várias outras espécies de tarefas. O Visual Studio adiciona outras tarefas automaticamente, como a referência a erros compilados e outros itens perceptíveis. Mas você também pode acrescentar dois tipos de tarefa a essa lista: atalhos de código e tarefas definidas pelo usuário.

FIGURA 2.20

Cada comentário marcado com uma palavra-chave especial é mostrado em uma lista de tarefas bem organizada.



Os atalhos de código são semelhantes às tarefas com base em comentários, mas são uma referência a alguma linha do código. Não precisam de nenhuma palavra-chave especial. Para adicionar um atalho de código à lista de tarefas, dê um clique com o botão direito do mouse na janela de edição de códigos e selecione o item Add Task List Shortcut no menu.

Uma nova tarefa será adicionada a sua lista, com o padrão para a descrição sendo a linha de código selecionada (embora você possa, e provavelmente deva, alterar isso para outra descrição que queira usar). Em seguida, você poderá retornar rapidamente para essa linha apenas com um clique duplo nessa tarefa. Quando um atalho é criado, uma seta azul é posicionada na margem esquerda da janela do código próxima à linha apropriada. É possível remover o atalho dando um clique com o botão direito do mouse na linha do código e selecionando Remove Task List Shortcut ou selecionando o item novo da lista de tarefas e excluindo-o diretamente.

O outro tipo de tarefa que você pode criar é uma tarefa do usuário, que não está associada a nenhum trecho em particular do código, semelhante a uma tarefa-padrão do Outlook. Uma tarefa do usuário é adicionada com um clique na seção Click Here to Add a New Task da lista de tarefas e preenchendo-se os detalhes. Observe que, diferente das outras tarefas, essa não possui campos de arquivo/linha preenchidos e, portanto, só têm dois campos disponíveis, a descrição e a prioridade (baixa, normal ou alta). Se quiser criar uma observação sobre uma área específica do código, você com certeza achará mais útil gerar um atalho de código e alterar a prioridade e a descrição para fornecer mais detalhes sobre a questão real.

Soluções e Projetos

Como discutimos na seção “Solution Explorer”, existem vários níveis nos quais seus códigos podem ser agrupados. O primeiro, a solução, representa o sistema completo que está sendo cria-

do, enquanto os componentes individuais dele são representados por projetos separados. Antes que você possa desenvolver qualquer código no IDE do Visual Studio, precisa configurar a solução e pelo menos um projeto. Nesta seção, percorreremos os princípios básicos da organização do código, criando novos projetos e trabalhando com projetos e arquivos existentes. A seguir apresento um resumo desses tópicos, mas, na seção logo a seguir, passaremos a praticar essas habilidades gerando um exemplo completo de um aplicativo.

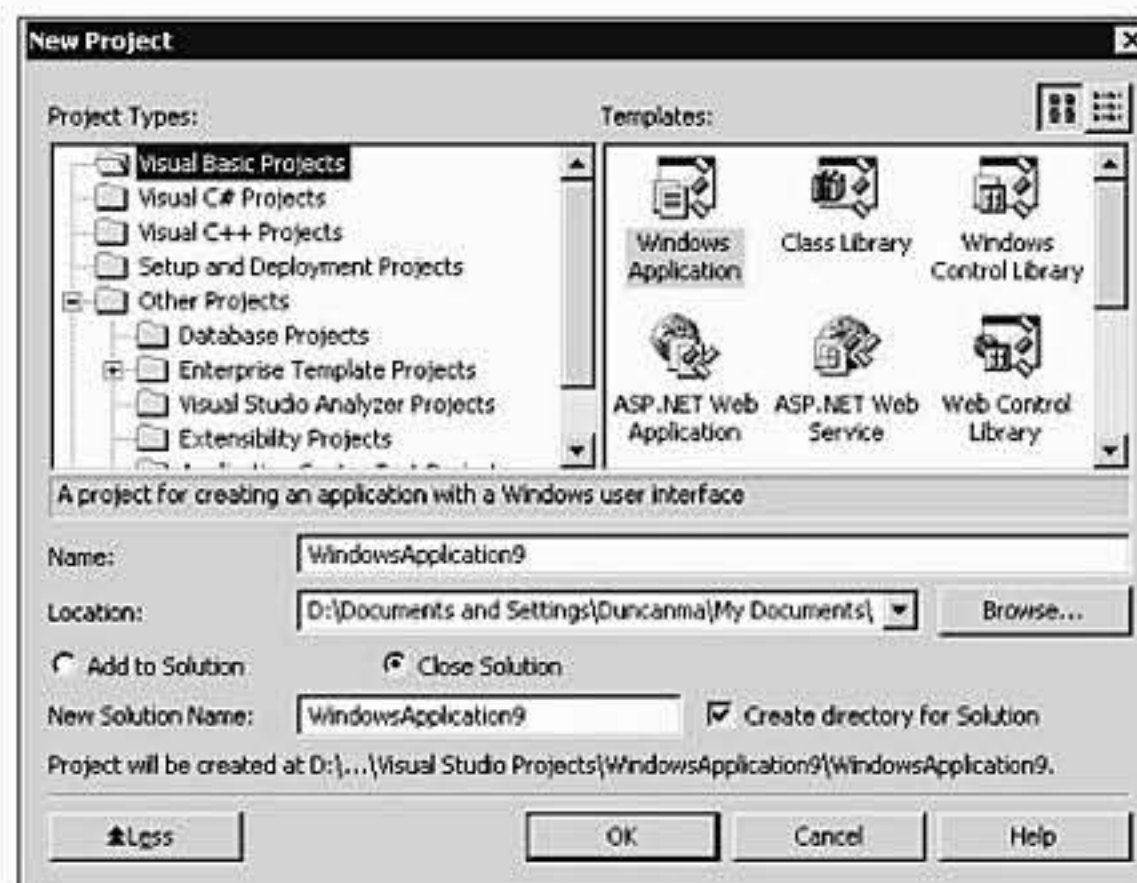
Criando um Novo Projeto

Existem algumas maneiras de criar um novo projeto, mas o método mais comum usa as opções File, New e Project do menu. Essas opções apresentam uma caixa de diálogo que mostra todos os diferentes tipos de projetos que o IDE pode gerar (veja a Figura 2.21). Já que o IDE do Visual Studio trabalha com várias linguagens, a caixa de diálogo mostra as opções com base nas linguagens que você tem instaladas e podem apresentar uma aparência diferente da exibida na Figura 2.21. Por enquanto, desenvolveremos projetos com base nas opções da pasta Visual Basic Projects.

2

FIGURA 2.21

O Visual Studio possui uma caixa de diálogo chamada New Project, que permite que novos projetos sejam acrescentados quando você instala modelos ou linguagens adicionais.



Para criar um aplicativo que seja executado localmente em sua máquina e usar uma interface com o usuário com base no Windows (com caixas de diálogo e outros elementos UI do Windows), selecione Windows Application na lista de tipos de projetos. Para concluir o processo de criação, digite um nome para seu novo aplicativo e, se quiser, altere o caminho sugerido. Dê um clique em OK, e o Visual Studio gerará seu novo projeto. É uma boa idéia dar a seus projetos nomes significativos, mesmo quando você estiver apenas fazendo testes; caso contrário, logo haverá todo um grupo de projetos com os nomes WindowsApplication1 e WindowsApplication2, dificultando a localização de algo em que tenha estado trabalhando.

Abrindo um Projeto Existente

Quando o Visual Studio for encerrado, ele perguntará se você deseja salvar o que esteve usando e fechará tudo de modo automático. Para que se possa voltar a um projeto anterior, esse precisará

ser aberto no IDE. O Visual Studio fornece algumas maneiras fáceis de abrir projetos passados. Um método é usar o menu através de File, Open e Project ou diretamente pela opção Recent Projects próxima à parte inferior do menu File. Outro método é por meio da seção Get Started da página inicial do Visual Studio, uma página HTML que lista os últimos projetos utilizados. Nesse local, é possível dar um clique no projeto específico que se quer abrir ou até criar um novo através de um link adicional. Abrir um projeto novo faz com que qualquer outro projeto que já esteja sendo usado seja fechado, a menos que sejam empregadas as opções File e Add Project do menu, que adicionam um projeto novo ou existente à solução aberta recentemente.

Arquivos

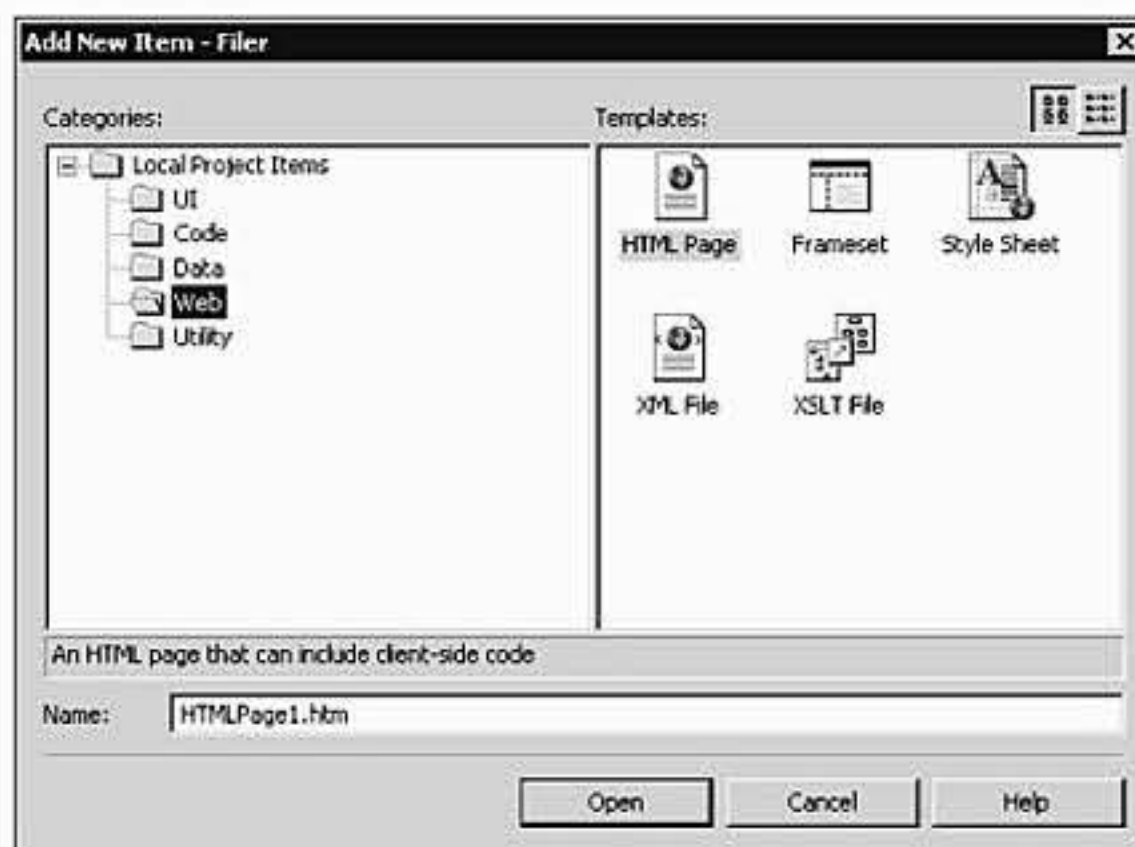
As soluções e projetos existem quase apenas para fins de organização; o código efetivo reside em um ou mais arquivos individuais. Quando você cria um projeto novo, em geral são gerados certos arquivos, como um novo formulário Windows (Form1.vb) ao ser criado um aplicativo Windows e um novo módulo (Module1.vb) para um aplicativo do console. Esses arquivos são gerados em disco e existem independentemente de seu projeto, permitindo que um único arquivo seja compartilhado por vários projetos se isso for desejado.

Adicionando Arquivos a um Projeto

Além dos arquivos que são criados de modo automático como parte de seu novo projeto, você também pode querer adicionar módulos, classes e formulários complementares ou outros tipos de arquivos de código. Por meio do menu Project ou do que surge com um clique no botão direito do mouse sobre o projeto da janela Solution Explorer, é possível optar por acrescentar qualquer arquivo de vários tipos. Independentemente da opção específica do menu que foi escolhida, com exceção de Add Existing Item, todas o conduzirão à caixa de diálogo Add New Item (veja a Figura 2.22). Se, em vez de criar um novo item, você quiser adicionar um arquivo existente ao disco, a opção Add New Existing Item do menu apresentará, para esse fim, uma caixa de diálogo para abertura de um arquivo-padrão.

FIGURA 2.22

Da mesma maneira que a caixa de diálogo New Project, a interface para adicionar itens novos ao projeto é expansível.



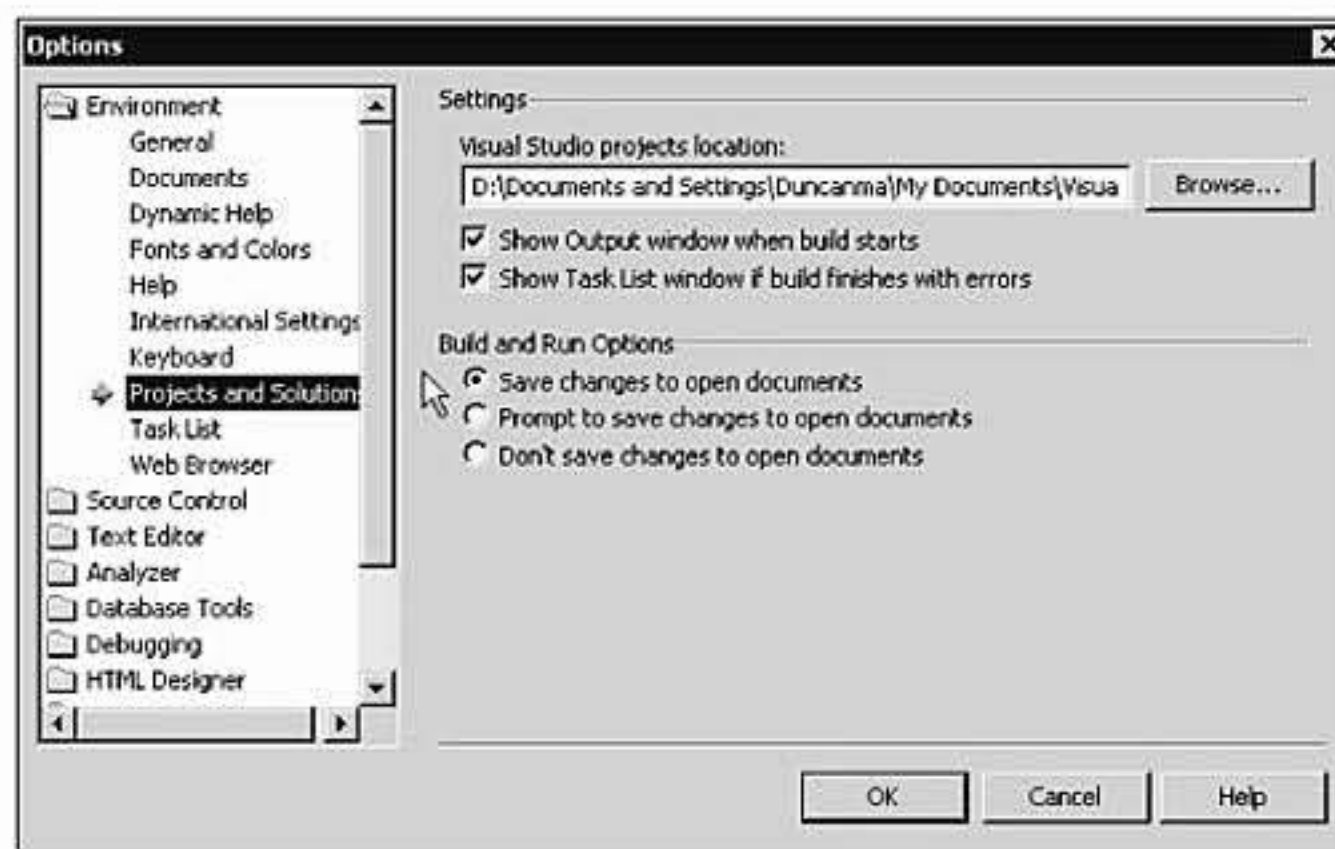
Salvando Tudo

Com todos esses grupos diferentes (soluções, projetos e arquivos), é importante saber como salvar os trabalhos que foram executados, mesmo se estiverem localizados em mais de um arquivo. No IDE do Visual Studio, isso é feito através de dois comandos diferentes: Save e Save All. Esses comandos, situados no menu File e na barra de ferramentas, permitem que seja salvo apenas o arquivo selecionado no momento (selecionado na janela Server Explorer) com o uso do comando Save ou, por meio do comando Save All, todos os arquivos abertos que tenham sido alterados.

Se você tem medo de perder seu trabalho, como eu, ficará especialmente interessado em uma das opções do IDE. Na caixa de diálogo Options (acessada por meio dos itens Tools e Options do menu), é possível expandir o grupo Environment e selecionar o item Projects and Solutions para ver um conjunto de três botões de opção sob o cabeçalho On Build/Run (veja a Figura 2.23). Essas opções controlam se o IDE salvará algum arquivo alterado antes de começar a executar um projeto. Essa é uma configuração importante porque, se o IDE falhar em algum momento, é bem mais provável que isso aconteça quando seu código estiver em execução. Essa opção fornece uma maneira fácil de assegurar que todas as suas alterações sejam sempre salvas antes da execução de seu código.

FIGURA 2.23

Ao usar um computador novo, verifique sempre essas configurações para salvar no momento do desenvolvimento, de modo que evite a perda de algumas horas de codificação.



Criando Nosso Primeiro Aplicativo Windows

Agora que você aprendeu alguns princípios básicos relativos ao uso do IDE do Visual Studio, pode usar essas informações para desenvolver um aplicativo sem precisar codificar muito. Esse exercício se concentrará no uso do próprio IDE, o que significa que iremos gerar um aplicativo relativamente simples. No exemplo, criaremos um aplicativo Windows (um aplicativo que usa a interface com o usuário do Windows e é executado localmente em sua máquina) que permite ao usuário inserir dois números. Em seguida, o aplicativo adicionará os números, exibindo o resultado final.

Crie o Projeto

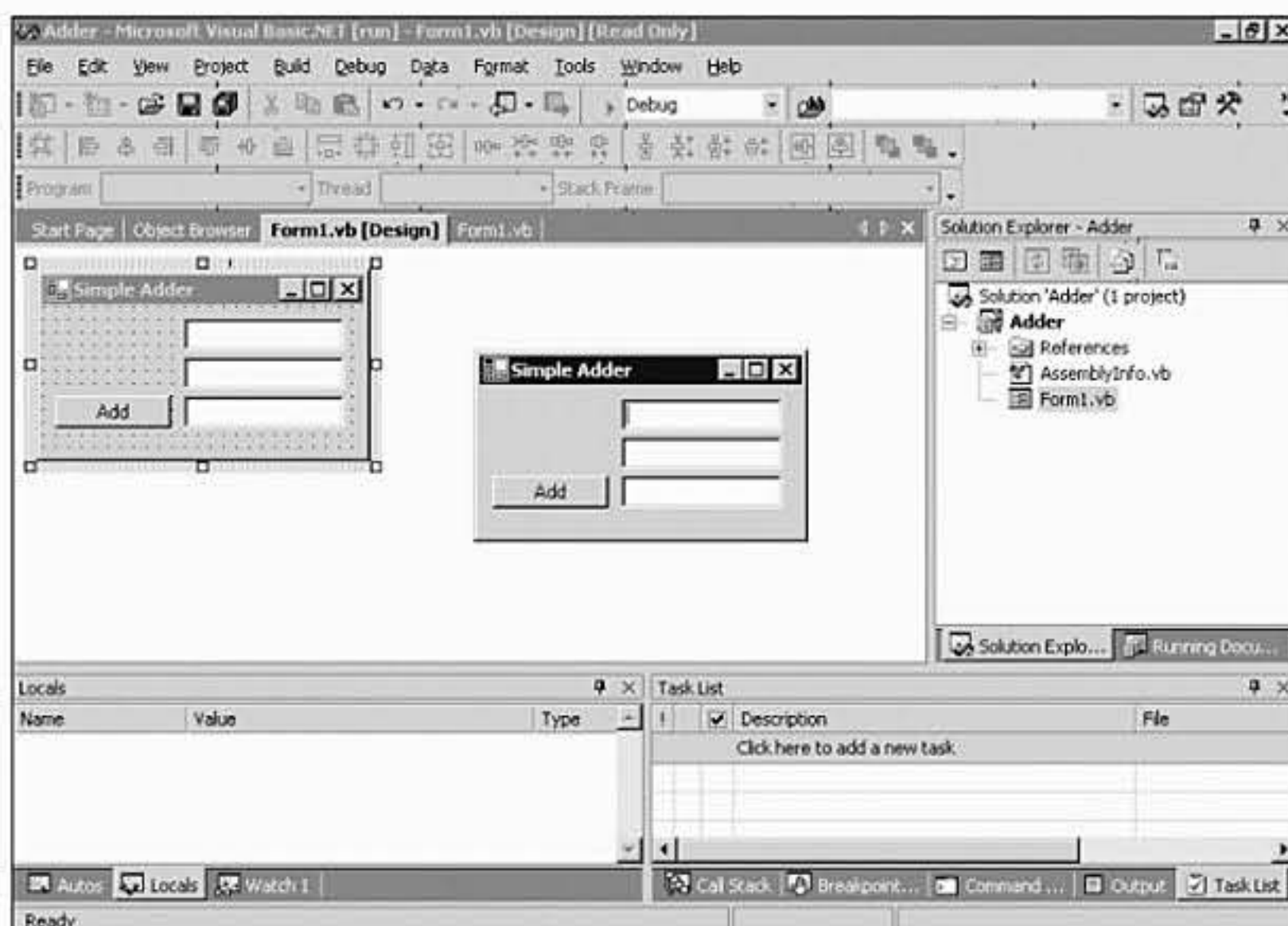
Nos menus, selecione os comandos File, New e Project, apresentando assim a caixa de diálogo New Project. Na categoria Visual Basic, selecione o ícone Windows Application e altere o nome do projeto de WindowsApplication(x) – o nome-padrão numerado – para **Adder** (a menos que você seja importunado por intrometidos, situação na qual poderá nomear o projeto da maneira que quiser). O novo projeto já terá um formulário, que é tudo de que você precisa para começar a trabalhar no aplicativo. O Visual Studio criará automaticamente uma pasta Solution para armazenar seu projeto novo, também dando o nome de Adder a ela. Dê um clique em OK depois que tiver inserido os nomes corretos do projeto e da solução.

Desenvolva a Interface com o Usuário

Você precisa de três caixas de texto e apenas um botão no formulário. A posição não é assim tão importante, mas você pode desejar que sua interface tenha uma aparência semelhante à da Figura 2.24. Dê um clique duplo em Form1 no Server Explorer, trazendo-o para a janela de projeto no centro do IDE. Agora, com o formulário no modo Design, selecione ou abra a janela Toolbox. Essa janela, que mostrará todos os objetos disponíveis que podem ser colocados em seu formulário, contém um controle TextBox e um controle Button. Para inserir um desses controles em seu formulário, dê um clique nele e arraste-o para a posição correta. Quando ele estiver no formulário, selecione o controle e use suas alças para redimensioná-lo até a forma e tamanho desejados. Trabalhe com o redimensionamento e a posição desses controles até que consiga que todas as três caixas de texto e o único botão estejam no formulário com a aparência do exemplo (Figura 2.24). Depois que tudo estiver posicionado, altere algumas propriedades desses controles.

FIGURA 2.24

Organize as três caixas de texto e o único botão em seu formulário para que tenham uma aparência semelhante a esta figura.



Selecione a primeira caixa de texto (a mais próxima da parte superior do formulário) e exiba a janela Properties (pressione F4 ou selecione View e Properties Window no menu, se a janela já não estiver visível). Várias propriedades diferentes serão listadas, mas você só irá alterar duas delas:

- **Text** (no grupo Appearance) Representa o conteúdo da caixa de texto. Exclua o conteúdo desta propriedade para fazer com que a caixa de texto fique vazia quando o programa for iniciado.
- **(Name)** (no grupo Design) No código, você se referirá a essa caixa de texto usando o nome desta propriedade, que tem como padrão um nome relativamente irrelevante, como `Text2` ou `Text1`. Neste exemplo, altere para **`txtFirstValue`**.

Continue com as outras duas caixas de texto, deixando sua propriedade **Text** em branco e alterando seus nomes para **`txtSecondValue`** e **`txtResult`**, respectivamente.

Agora selecione o botão para exibir seus atributos na janela Properties. Você também alterará somente dois valores desse objeto, **(Name)** para **`btnAdd`** e **Text** para **`Add`**.

Para concluir, só porque tem esse recurso, você alterará uma das propriedades do próprio formulário. Selecione o formulário (dê um clique em algum local dele que não seja outro objeto) e role pela lista de propriedades para encontrar a propriedade **Text** no grupo **Appearance**. Para um formulário, essa propriedade representa seu título (o texto exibido na barra de título), o qual poderemos configurar como **`Simple Adder`** no exemplo.

Executando o Projeto

Mesmo se você não tiver inserido um código, o projeto poderá ser executado da maneira como se encontra. O IDE do Visual Studio permite que o programa seja processado dentro dele, sem a existência de um programa executável (um arquivo `.exe`, por exemplo). É possível processar rapidamente o trecho de código necessário durante o desenvolvimento e, o que é mais importante, executar várias tarefas de depuração enquanto o programa estiver sendo processado. Mais informações sobre os aplicativos de depuração serão fornecidas no Dia 6, mas, por enquanto, é essencial observar que há uma diferença entre processar seu código no IDE e gerar um arquivo executável real. A criação de um executável, ou outro tipo de arquivo de resultado de seu projeto, é chamada de construção e será abordada na próxima seção.

Para executar um projeto no IDE, selecione **Start** no menu **Debug**, pressione F5 ou use o botão da barra de ferramentas que se parece com uma seta para a direita (ou com o botão de reprodução de um videocassete). Tente isso agora, com o projeto **Adder** que você já abriu, e verá o formulário sobre o IDE do Visual Studio. Mesmo sem que tenha sido escrita uma única linha de código, o formulário parecerá bem funcional. Ele poderá ser arrastado pela tela e minimizado, tudo por causa do .NET Framework e do IDE subjacentes que permitem a criação visual de uma interface com o usuário e produzem o código necessário para fazê-la funcionar.

Embora você não tenha escrito nenhum código, uma codificação extensa já foi gerada pelo IDE, e é ela que é executada quando o projeto é processado.

Construindo o Projeto

Construir um projeto é a criação de um executável ou outros arquivos de resultado. Para um aplicativo Windows como o do exemplo, isso significa a compilação de seu código em um arquivo .exe que possa ser executado fora do IDE do Visual Studio. Essa é uma etapa essencial se você pretende que um projeto seja usado por outras pessoas que não sejam desenvolvedores, embora possa ser evitada durante o desenvolvimento pela execução dentro do IDE.

Configurações para a Construção de Projetos

Para construir um projeto, selecione Build no menu Build (não são nomes muito criativos para os menus, mas são fáceis de entender), que parecerá ser de muito pouca utilidade. Sem fornecer muito em termos de informação, as configurações-padrão de Build criaram um arquivo Adder.exe, inserindo-o no subdiretório bin da pasta de seu projeto. A menos que você tenha optado por personalizar o caminho exibido quando criou este projeto, ele deve estar localizado em My Documents\Visual Studio Projects\Adder, e o arquivo executável em \bin\Adder.exe dentro desse diretório. Para examinar todas essas configurações-padrão, e, quem sabe, alterá-las, dê um clique com o botão direito do mouse em seu projeto na janela Solution Explorer e selecione Properties no menu suspenso que aparecerá. A caixa de diálogo das propriedades de seu projeto contém várias configurações, mas as que são relevantes para o processo de construção estão descritas a seguir.

Em Common Properties\General:

- **Assembly name** Este valor fornece a primeira parte do nome do arquivo de seu arquivo de resultado. No caso do exemplo, ele é Adder, portanto, Adder.exe foi criado. Altere-o para **MyAdder**, e MyAdder.exe será criado quando você iniciar a construção.
- **Output type** Informa ao IDE que tipo de arquivo será criado na construção deste projeto, com a extensão .exe se Windows Application ou Console Application for selecionado, ou .dll, se Class Library for selecionado.
- **Startup object** Indica a parte do projeto que deve ser executada por padrão quando o aplicativo for processado. No exemplo, ela deve ser Adder.Form1 para indicar que o formulário deve ser executado de maneira automática. Observe que, se você alterar, mesmo temporariamente, Output Type de Windows Application para qualquer outra opção, essa propriedade também será modificada e pode acabar configurada de modo incorreto.

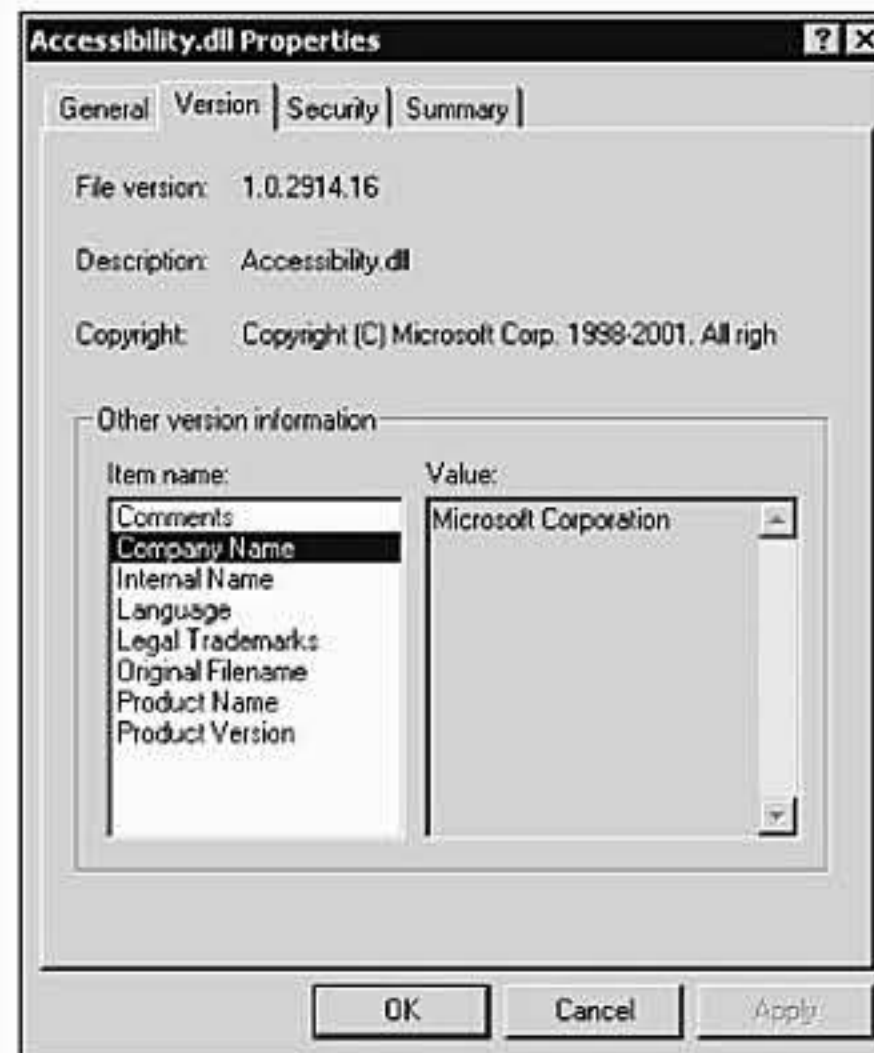
Cancele todo o processamento da caixa de diálogo de propriedades do projeto (dando um clique no botão Cancel na parte inferior do formulário) se você perceber que alterou algo que não sabe como corrigir.

Todas as configurações de Common Properties\Version são armazenadas nos arquivos de saída quando o projeto é construído, e ficam visíveis como parte das informações das propriedades do

arquivo executável final (veja a Figura 2.25). Enquanto você estiver aprendendo o Visual Basic, as informações inseridas nessa caixa de diálogo podem ser irrelevantes, mas, quando criar um aplicativo que será distribuído para os usuários, é recomendável levar em consideração as informações dessa página com muito cuidado. No mínimo, certifique-se de que inseriu os números corretos das versões na página porque eles fornecem um método conclusivo para os usuários verificarem que versão de um aplicativo possuem.

FIGURA 2.25

Quando você visualizar as propriedades de um arquivo executável ou DLL, poderá examinar todas as informações especificadas antes de sua construção.



Em Common Properties\Build, apesar do nome, encontramos apenas uma propriedade que é claramente relevante para o processo de construção. O valor de Application Icon determina a aparência do arquivo final .exe no Windows e permite que você selecione o arquivo de ícones (.ico) que desejar.

Embora não sejam as únicas configurações adicionais que afetam a construção, os últimos itens que mencionarei estão em Configuration Properties\Build. Nesse local você encontrará várias configurações relacionadas com a depuração, assim como a configuração de Output Directory, que determina onde o arquivo executável ou outros que forem criados serão posicionados.

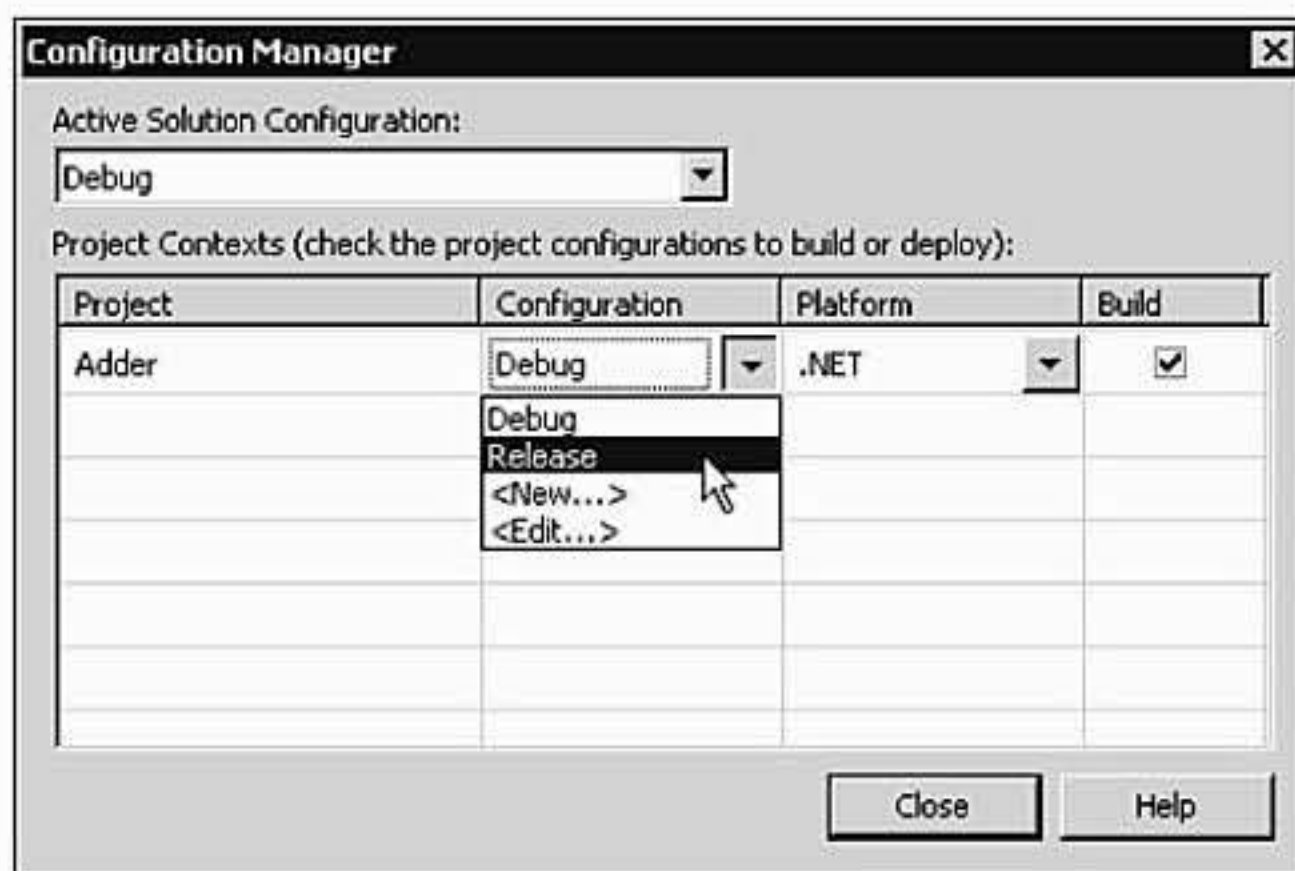
Outras Configurações de Construção

Na parte superior da caixa de diálogo Project se encontra uma lista suspensa chamada Configuration. O Solution Configurations é um recurso útil do IDE do Visual Studio, que permite que você crie mais de um grupo de configurações para o mesmo projeto. Por padrão, duas configurações são fornecidas (Release e Debug), projetadas para indicar se estão sendo construídos resultados para fins de teste (Debug) ou para desenvolvimento real (Release). As definições para essas configurações-padrão são um bom exemplo da finalidade do Solution Configurations, que define o status de vários recursos de depuração e até de um local diferente de resultado para cada um deles.

Com o Configuration Manager (veja a Figura 2.26), você poderá criar quantos grupos diferentes de configuração ou até mesmo remover uma existente. Por enquanto, será preferível deixar as configurações como estão, selecionando a versão Release do resultado quando estiver implantando o projeto, e Debug no teste.

FIGURA 2.26

O Configuration Manager permite que você crie configurações diferentes para finalidades distintas (teste, depuração, aceitação do usuário, liberação e outras), cada uma podendo ter configurações diferentes no menu Build.



Criando um Arquivo Executável

A melhor maneira de entender um recurso do IDE é usando-o, o que significa que é hora de construir o projeto do exemplo e produzir um arquivo executável. Dê um clique na opção Build do menu de mesmo nome, e verá uma informação piscar rapidamente na janela Output (exibida na parte inferior de sua tela, se você não a tiver mudado de lugar). Se tudo correr bem e não houver erros em nenhum código que tenha sido adicionado ou alterado, o processo de construção produzirá um arquivo executável. Esse arquivo, que foi chamado de `Adder.exe`, será criado no subdiretório `bin` da pasta de seu projeto (por padrão, `My Documents\Visual Studio Projects\Adder`). Minimize o IDE do Visual Studio, use o Windows Explorer para encontrar o arquivo e, em seguida, dê um clique duplo nele para executar seu aplicativo Windows recém-criado. O formulário, com suas diversas caixas de texto e botões, aparecerá para mostrar que o programa está sendo processado e continuará a ser até que esse mesmo formulário seja fechado. Esse arquivo executável, junto com o .NET Framework, é tudo que é necessário para fazer a distribuição para a máquina de um usuário de modo que esse possa executar seu programa.

Adicionando Seu Próprio Código

Até aqui, o exemplo do projeto em que você esteve trabalhando continha apenas o código gerado pelo IDE do Visual Studio, que tem sido ótimo para mostrar o formulário, mas não faz mais nada. Como já deve ter sido percebido pelos nomes e layout do formulário, esse aplicativo adicionará valores à primeira e segunda caixas de texto e colocará o resultado em uma terceira e última. Para tanto, será preciso acrescentar um código ao projeto que será executado quando o usuário der um clique no botão Add.

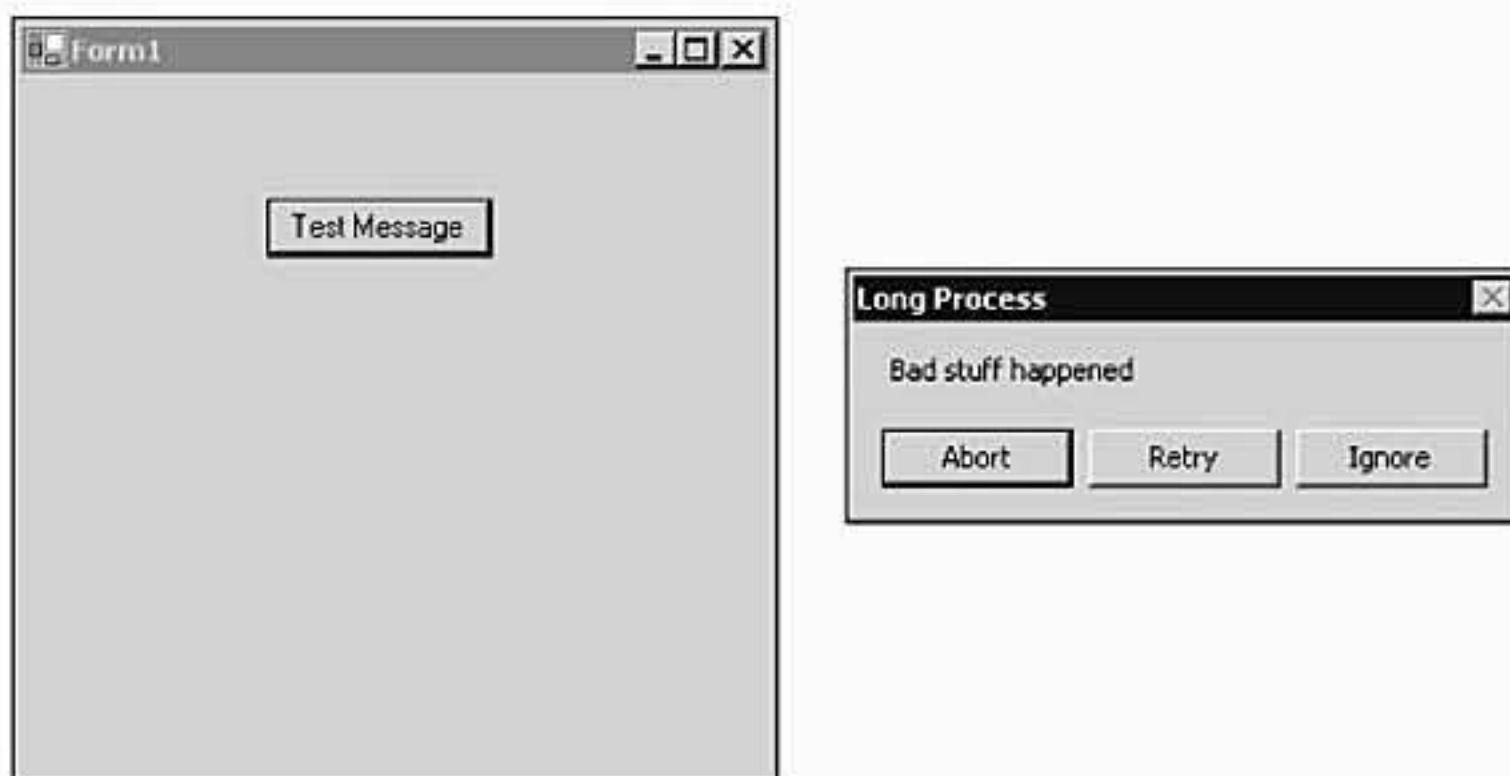
O uso do IDE faz com que esse processo seja muito direto: dê um clique duplo no formulário do Solution Explorer para assegurar que a visualização do projeto esteja ativada e, em seguida, dê um clique duplo no botão Add. Isso o conduzirá para a visualização do código do formulário e a uma sub-rotina que foi adicionada pelo IDE. Um controle, como esse botão, caixas de texto ou outros componentes do formulário, pode ter eventos associados a ele. A maioria dos eventos de um controle representa ações executadas neles como serem clicados, clicados duas vezes, selecionados e desmarcados. É possível criar procedimentos que serão executados quando um desses eventos ocorrer e, por padrão, esses procedimentos são designados pelo seu nome (o nome do controle, `btnAdd`, seguido do evento, `Click`). Podem-se associar procedimentos a eventos, independentemente de seus nomes, mas neste caso, o procedimento `btnAdd_Click` será executado se o usuário der um clique no botão. É fácil testar essa funcionalidade usando um recurso muito útil do .NET Framework que fornece os formulários, botões, caixas de texto e outros elementos de interface, e, portanto, está disponível para qualquer projeto no qual se esteja usando esses objetos. A classe `MessageBox` permite que uma mensagem seja exibida em uma caixa de diálogo com uma única linha de código, como esta:

```
MessageBox.Show("O botão foi clicado")
```

Esse código exibe uma caixa de diálogo como a mostrada na Figura 2.27. A simplicidade do uso dessa classe a torna perfeita para ser empregada como uma ferramenta de teste ou de depuração. Adicione a linha de código anterior à sub-rotina `btnAdd_Click` e, em seguida, execute o projeto pressionando F5. Depois que o formulário aparecer, tente dar um clique no botão. Cada clique deve provocar o aparecimento da caixa de mensagem, mostrando a você que o código de `btnAdd_Click` é executado sempre que o botão é pressionado.

FIGURA 2.27

A classe `MessageBox` fornece uma maneira fácil de exibir informações na tela e é em geral usada nas fases de depuração/teste.



Agora que você já sabe como executar um código que responde ao clique em um botão, poderá criar o código efetivo de seu projeto. Ele tem de somar dois valores para produzir um resultado, o que soa mais fácil do que é na realidade. Os valores que queremos são os conteúdos de suas caixas de texto, disponíveis por meio da propriedade `Text` desses controles, mas antes que possamos usá-los em uma equação matemática (somando-os), temos de convertê-los de strings (texto)

para números. O código a seguir, se colocado no lugar da chamada de `MessageBox`, adicionada anteriormente, fará o que precisamos:

```
txtResult.Text = (CInt(txtFirstValue.Text) _  
    + CInt(txtSecondValue.Text)).ToString
```

Esse código converte o conteúdo das duas caixas de texto em números (inteiros, neste caso), faz a soma e, em seguida, converte o resultado de novo para uma string (texto) de modo que ele possa ser inserido na terceira caixa de texto. São necessárias algumas etapas para fazer algo que aparentemente é muito fácil quando descrito pela primeira vez, e o resultado final pode parecer um pouco confuso. Tudo ficará mais claro ao darmos prosseguimento no Dia 3, “Introdução à Programação com o Visual Basic .NET”.

Resumo

O objetivo do IDE é fornecer um ambiente no qual você possa escrever, editar, executar e depurar seus programas, e o do Visual Studio fornece todos esses recursos e mais. Esta lição abordou os recursos básicos do IDE e a finalidade de cada uma de suas janelas principais. Também o conduziu pela criação de um aplicativo Windows simples. Daqui em diante, estaremos usando o IDE freqüentemente, apesar de nem sempre trabalharmos com elementos visuais (como formulários e controles). Portanto, aprenderemos mais sobre ele e seus recursos o tempo todo. Embora possa parecer um pouco confuso e cansativo agora, o IDE é a ferramenta que você usará para fazer sua programação de modo que, com o tempo, ela se tornará familiar.

P&R

P Preciso usar o IDE ou posso apenas empregar um editor de texto e a linha de comando?

R No Visual Basic .NET, é possível fazer qualquer coisa usando apenas a linha de comando e um editor de texto, mas o IDE fornece uma grande quantidade de recursos para tornar o desenvolvimento um processo muito mais fácil. O IntelliSense, a estrutura das instruções e a edição codificada em cores, tudo faz com que a criação de códigos seja mais agradável, além de ser difícil ficar sem os recursos de depuração.

P Posso adicionar meus próprios recursos ao IDE do Visual Studio?

R Com certeza! O IDE dá suporte à personalização por meio de vários métodos diferentes, incluindo macros e complementos. Não abordarei a personalização do IDE neste livro, mas, por hora, examine os ótimos exemplos de macros nas opções Tools, Macros e Macros IDE do menu.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Se você quisesse ver todos os arquivos que fazem parte de seu projeto, que janela do IDE usaria?
2. Qual é o local-padrão dos novos projetos do Visual Studio?
3. Como você pode escolher um ícone para um aplicativo que esteja desenvolvendo no Visual Studio?
4. Se a janela Command estiver no modo Immediate, como posso passar para o modo Command e depois voltar?

2

Exercícios

Exatamente como você fez nesta lição, use a classe `MessageBox` para adicionar mensagens ao procedimento de outros eventos e verifique quando forem chamados. Tente selecionar `txtResult` na primeira lista suspensa da janela de edição de códigos e, em seguida, `TextChanged`, na segunda lista para trabalhar com esse evento.

PÁGINA EM BRANCO

SEMANA 1

DIA 3

Introdução à Programação com o Visual Basic .NET

Agora que você se familiarizou com o ambiente .NET de desenvolvimento, é hora de começar a escrever códigos. Embora com o Visual Basic .NET seja fácil escrever um programa simples sem o uso de muita codificação, mesmo se for mais simples que uma versão de demonstração, ele precisará manter o registro das informações e executar cálculos elementares e tarefas semelhantes. Para escrever códigos que executem essas tarefas, será necessário um bom conhecimento sobre variáveis. Por meio da compreensão do emprego e dos tipos de variáveis, é criada a base para assimilação do Visual Basic .NET. De maneira parecida, exatamente como quando iniciamos o aprendizado da aritmética simples, precisaremos aprender alguns operadores fundamentais que podem ser utilizados no trabalho com variáveis numéricas e alfanuméricas. Nesta lição, você aprenderá:

- Os tipos de variáveis que pode criar com o Visual Basic .NET.
- Alguns operadores e funções simples disponíveis no Visual Basic .NET.
- Os princípios básicos para escrever códigos no Visual Basic .NET, inclusive como desenvolver procedimentos.

Variáveis e Atribuição

Variáveis e atribuição são a base de toda linguagem de programação. As variáveis permitem que você armazene informações para uso posterior e a atribuição é a maneira de inserir informações nas variáveis.

O Que É uma Variável?

Uma variável é um depósito. É um local para guardar as informações até que sejam necessárias. Você usará variáveis em toda a extensão de seus programas para reter valores temporários durante os cálculos, armazenar entradas do usuário e preparar as informações que serão exibidas posteriormente para eles.

Tipos de Variáveis Disponíveis

Exatamente como acontece com as calças, só um tamanho não atende a todas as variáveis. Embora seja possível criar e usar uma variável que seja capaz de conter o que for, essa nem sempre é a melhor solução. É fácil deduzir que uma variável que contém strings deve executar algo diferente de outra criada para armazenar números. Além disso, mesmo tipos diferentes de números requerem tipos distintos de variáveis. Alguns números, como 1 ou 5280, não possuem casas decimais, enquanto 3,14159265358979 e 16,50 apresentam essa característica. Uma variável gerada para conter um número decimal deve ter uma maneira específica de manter o registro dos valores depois da vírgula. É claro, que isso significa que os números decimais *provavelmente* consumirão mais memória. Sempre que um computador ou programa realiza um trabalho maior, em geral precisa de mais memória. Portanto, é importante lembrar não só do tipo de informação que você precisa armazenar, mas também do espaço da memória que o computador terá de manipular para registrar a variável.

Há três tipos abrangentes de variáveis que você pode criar com o Visual Basic .NET. O primeiro conjunto engloba as variáveis que armazenam valores simples, como números ou strings. Há muitas desse tipo, cada uma foi projetada para conter valores de vários tamanhos. A segunda categoria é a das variáveis complexas, que contêm alguma combinação de variáveis simples e incluem arrays e tipos definidos pelo usuário. Os *arrays* são variáveis que armazenam muitas outras, e os *tipos definidos pelo usuário* permitem que o usuário crie novos tipos de variáveis. A terceira categoria é a das variáveis de objeto.

Os tipos definidos pelo usuário (também conhecidos como *estruturas*) e as variáveis de objeto serão abordados no Dia 7, “Trabalhando com Objetos”. A discussão desta lição se concentrará nas variáveis simples e arrays.

Variáveis Simples

Como descrito anteriormente, os tipos simples de variáveis ‘armazenam’ valores como números e palavras. Portanto, você pode achar que só é necessário existir dois tipos de variáveis: de números e de palavras. Porém, na verdade, há vários tipos diferentes de variáveis simples – cada um criado para armazenar tamanhos ou tipos distintos de números ou strings.

Tente usar o melhor tipo de variável para a situação. Em algumas vezes, só será preciso manter o registro de um número pequeno – por exemplo, se estiver armazenando os meses do ano. Em outras, terá de trabalhar com números grandes – por exemplo, se estiver escrevendo um programa que execute cálculos científicos ou de engenharia.

As variáveis simples podem ser divididas em quatro subgrupos. O primeiro e maior é o dos inteiros, números que não possuem casas decimais. O segundo grupo é usado para números com casas decimais. As strings e os caracteres compõem o terceiro grupo, e o quarto seria melhor descrito como ‘diversos’. Examinemos cada um desses grupos e vejamos quando qual é apropriado usar em que situação.

Variáveis de Inteiros

NOVO TERMO

As *variáveis de inteiros* (integer) armazenam os conhecidos números inteiros (isto é, números sem casas decimais). Essas variáveis são as que você usará com mais frequência nos programas e as mais fáceis para os computadores tratar. Por causa dessa comodidade, elas devem ser seu tipo preferido quando for preciso trabalhar com números. A Tabela 3.1 mostra diversas variáveis de inteiros diferentes, cada uma foi criada para armazenar números de tamanhos distintos e para empregar quantidades diferentes de memória. A quantidade de memória utilizada é medida em bytes. (Um byte contém oito bits, o que é apenas uma maneira técnica de dizer que cada byte possui oito algarismos um ou zero, ou uma combinação de uns e zeros.) Embora não haja problema em usar uma variável projetada para conter valores maiores do que o necessário, ela consumirá mais memória. Além disso, pode fazer com que seu programa seja executado mais lentamente porque teria de manter registros de seções maiores da memória, mesmo que essas nunca fossem utilizadas.

TABELA 3.1 Tipos de Variável de Inteiros

<i>Tipo de Dado</i>	<i>Tamanho (Bytes)</i>	<i>Abrangência</i>	<i>Comentários</i>
Byte	1	0 a 255	Um pouco limitado e, diferente dos outros tipos de dados inteiros; o Byte não dá suporte a números negativos. Isso acontece porque ele representa o valor que o computador realmente armazena em cada byte da memória. Para armazenar números negativos, o computador usa uma parte de cada byte para guardar a porção 'negativa'. Útil quando se mantém registros de números pequenos que nunca são negativos, como os dias de um mês ou os meses de um ano.
Curto (Short)	2	-32.768 a 32.767	Uma variável de inteiros pequena e prática. Você pode usar um tipo curto sempre que não precisar de todo o intervalo de um tipo inteiro, por exemplo, se estivesse escrevendo um programa para contar a quantidade de empregados de uma empresa que só tivesse alguns milhares de funcionários.
Inteiro (Integer)	4	-2.147.483.648 a 2.147.483.647	A variável de inteiros-padrão. Em geral, o inteiro é o tipo mais veloz de variável para se usar, já que ele requer menos trabalho do computador. Um emprego para esse tipo de variável seria registrar a quantidade de ovelhas da Nova Zelândia (aproximadamente 47.394.000 em 1997).
Longo (Long)	8	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	O tipo de variável perfeito para esta época em que estamos trabalhando com números muito, muito grandes (isso significa -9 quintilhões a +9 quintilhões ou aproximadamente 9×10^{18} para a manutenção de registros). Um tipo longo seria útil se você estivesse armazenando a quantidade de estrelas no universo (estimada em cerca de 10^{22}).

Números com Casas Decimais

Uma grande quantidade de processamento numérico é realizada sem casas decimais. No entanto, mais cálculos, principalmente em engenharia, finanças e ciências, requerem que você também possa armazenar valores decimais. A Tabela 3.2 descreve os dois principais tipos de variáveis decimais. Decidir qual usar depende do grau de precisão que for necessário manter, em vez do tamanho dos valores, já que todos eles podem ter números bastante grandes. Caso não se

lembre da notação científica que aprendeu na escola, o número sobrescrito acima do algarismo 10 é a quantidade de vezes que é preciso multiplicar por 10 (se positivo) ou dividir por 10 (se negativo). Portanto, 10^6 é 10 seis vezes ou 1.000.000, e 10^{-6} é igual a 0,000001.

TABELA 3.2 Tipos de Variável Decimal

<i>Tipo de Dado</i>	<i>Tamanho (Bytes)</i>	<i>Abrangência</i>	<i>Comentários</i>
Simples (Single)	4	$-3,402823 \times 10^{38}$ a $-1,401298 \times 10^{-45}$ para números negativos; $1,401298 \times 10^{-45}$ a $3,402823 \times 10^{38}$ para números positivos	<p>Não se preocupe muito com o tamanho desses números do intervalo. O tipo simples pode manter o registro de números muito grandes (ou muito pequenos). O importante para este tipo de dado é a precisão. O nome 'simples' quer dizer que este tipo de variável é para <i>números com ponto flutuante de precisão simples</i>. Esse é o significado do jargão da informática: "Ele só é realmente bom para armazenar sete dígitos importantes". Examine cada um dos números do intervalo.</p> <p>Observe que possuem um número antes da casa decimal e seis a seguir, mais o expoente (o número acima do algarismo 10). Portanto, embora o tipo simples seja bom para armazenar tanto números grandes quanto pequenos, não é tão preciso quanto os outros, e poderia causar erros de arredondamento se você fizesse muitos cálculos com valores realmente altos ou muito baixos. O tipo de variável simples seria útil em um programa em que menos exatidão fosse necessária.</p>
Duplo (Double)	8	$-1,79769313486231 \times 10^{308}$ a $-4,94065645841247 \times 10^{-324}$ para números negativos; $4,94065645841247 \times 10^{-324}$ a $1,79769313486232 \times 10^{308}$ para números positivos	<p>O tipo duplo é uma variável de 'ponto flutuante com precisão dupla', portanto, armazena duas vezes mais dígitos significativos que o tipo simples ou 15 casas decimais. Use um tipo duplo sempre que você fizer cálculos com números grandes ou quando quiser evitar os erros de arredondamento que podem acontecer no tipo simples, como ao efetuar cálculos em aplicativos científicos ou de engenharia.</p>

Strings e Caracteres

Os números são adequados se você precisa armazenar quantidades ou espaços de tempo, mas com frequência também é necessário lidar com palavras na programação. O Visual Basic .NET fornece variáveis para armazenamento de strings: os tipos de dado `Char` e `String`. O tipo `Char` é apropriado para o armazenamento de apenas um caractere (daí o nome), enquanto o tipo `String` pode conter strings de comprimento maior. A Tabela 3.3 descreve os dois tipos de dados com mais detalhes.



TABELA 3.3 Tipos de Variáveis de Strings

<i>Tipo de Dado</i>	<i>Tamanho (Bytes)</i>	<i>Abrangência</i>	<i>Comentários</i>
Char	2	Um caractere	Adequado para armazenar um único caractere.
String	10 + 2 por caractere	Até dois bilhões de caracteres	Use um tipo <code>String</code> para armazenar aquele romance que sempre quis escrever. Se estiver calculando cinco caracteres em média por palavra e 250 palavras por página, saiba que uma única variável alfanumérica pode conter 1,7 milhão de páginas de texto.

Por que cada tipo `Char` e cada caractere de um tipo `String` ocupam dois bytes? Afinal, há apenas 25 caracteres usados no idioma português, além dos números e símbolos – decerto, você não precisa de dois bytes (65.536 valores possíveis). Não precisaria, se todas as linguagens empregassem o mesmo conjunto de caracteres.

NOVO TERMO

O cenário é o que o conjunto de caracteres *ASCII* (ou *ANSI*), popular no passado, definiu. Em *ASCII*, um byte é igual a um caractere, e todos os computadores que usavam essa linguagem sempre dispunham os mesmos caracteres na mesma posição da lista. Portanto, o valor 65 em *ASCII* era sempre a letra ‘A’, e o símbolo ‘@’ possuía valor igual a 64. Se você quisesse fazer o mesmo com todos os outros símbolos que as pessoas usam na escrita, no entanto, precisaria de mais caracteres. Para resolver esse problema, um novo sistema foi desenvolvido, chamado *Unicode*.

No *Unicode*, cada caractere é representado por dois bytes. Isso permite o armazenamento de todos os caracteres da lista *ASCII*, bem como os dos idiomas russo, grego, japonês e tailandês, os dos matemáticos e assim por diante. Em *Unicode*, 65 ainda representa a letra ‘A’, mas 8800 é o caractere . O caractere japonês Hiranaga “” (‘não’) é representado por 12398. O Visual Basic .NET usa os valores *Unicode* para todos os caracteres, portanto o tipo `Char` usa dois bytes, e cada caractere de um tipo `String` acrescenta dois bytes de espaço complementar de armazenamento.

Outros Tipos de Variável Simples

Nunca falha quando se tenta categorizar as coisas. Algo sempre desafia a categorização (apenas imagine o primeiro zoólogo que se deparou com o ornitorrinco). Da mesma maneira, há algumas variáveis que não se encaixam bem nas categorias descritas anteriormente. No Visual Basic .NET, existem dois desses tipos de variáveis ‘diversas’, o tipo booleano (Boolean) e a data (Date). Eles são descritos com mais detalhes na Tabela 3.4.

TABELA 3.4 Tipos Diversos de Variáveis Simples

<i>Tipo de Dado</i>	<i>Tamanho (Bytes)</i>	<i>Abrangência</i>	<i>Comentários</i>
Booleano (Boolean)	2	Verdadeiro (True) ou falso (False)	Se está armazenando apenas verdadeiro ou falso, para que dois bytes? O Visual Basic tem usado tradicionalmente 0 e -1 para falso e verdadeiro. Definir esses dois valores requer dois bytes.
Data (Date)	8	1º de janeiro de 100 a 31 de dezembro de 9999	Esta variável pode conter a maioria das datas com as quais você lidará (a menos que seja um historiador ou geólogo). Ela também segue todas as regras do calendário (como adicionar um dia em anos bissextos), portanto, se acrescentarmos 1 ao valor ‘28 de fevereiro de 2000’ da variável de data, obteremos ‘29 de fevereiro de 2000’, mas se fizermos o mesmo para ‘28 de fevereiro de 2001’, a resposta será ‘1º de março de 2001’. A única limitação efetiva da variável de data é seu tamanho.



NOTA

Você poderia usar outro tipo de dado para armazenar datas como strings ou inteiro representando o período em dias após alguma data específica.

Declarando Variáveis

Agora que você conhece os tipos de variáveis disponíveis, como criá-las em seus programas? A maneira mais simples é com a palavra-chave Dim (abreviatura de ‘Dimensão’), seguida pelo nome da variável, a palavra-chave As e, por fim, pelo tipo da variável. A aparência é a seguinte:

```
Dim iAlgunNumero As Integer
```

Isso criaria a nova variável iAlgunNumero, que possui quatro bytes e pode armazenar um número com um tamanho aproximado a 2 bilhões. Veja algumas declarações de variáveis possíveis:


```
Dim sPrimeiroNome As String
Dim dblProdutoInternoBruto As Double
Dim bAprendido As Boolean
```

Um recurso novo no Visual Basic .NET é a capacidade de fornecer um valor à variável quando ela estiver sendo criada. Faça isso usando a mesma linha da instrução Dim:

```
Dim dtDataDeAssinaturaDaCartaMagna As Date = #June 15, 1215#
Dim lPessoasNaTerra As Long = 6000000000
```

Examinaremos algumas outras maneiras de declarar variáveis posteriormente, quando discutirmos o escopo.

Arrays

A capacidade de uma variável de armazenar qualquer coisa é prática e até mesmo essencial em programação. No entanto, você pode precisar armazenar vários itens relacionados. Por exemplo, se estivesse escrevendo um programa de jogo de xadrez, os quadrados do tabuleiro precisariam ser representados como um conjunto de itens relacionados. Use os *arrays* para criar variáveis que armazenem juntos todos os itens relacionados. No programa do jogo de xadrez, o tabuleiro provavelmente seria armazenado como um array de posições, cada uma contendo o tipo de peça (ou nenhuma) dessa posição. Se não fosse usada uma variável de array, teriam de ser empregadas 64 variáveis separadas. Também pode ser preciso manter o registro de uma lista de strings, por exemplo, dos nomes dos alunos de uma sala. Sempre que for necessário armazenar uma lista de itens, use um array.

Como na declaração das variáveis simples, a de um array é feita com a palavra-chave Dim. No entanto, há algumas diferenças entre declarações de variáveis simples e declarações de arrays porque esses são conjuntos de variáveis. A Listagem 3.1 mostra três maneiras possíveis de declarar arrays.

CÓDIGO

LISTAGEM 3.1 Declarando Arrays

```
1 'Declaração simples
2 Dim iValues(3) As Integer
3 Dim dtDates() As Date
4 Dim I As Integer
5 For I = 1 To 3
6     iValues(I-1) = I
7 Next

8 'Alterando o tamanho de um array existente
9 ReDim dtDates(4)
10 'preencha a lista de datas
11 dtDates(0) = "15/6/1215" 'Assinatura da Carta Magna
12 dtDates(1) = "28/8/1962" 'Martin Luther King Jr. pronuncia "Tenho um sonho"
```


CÓDIGO**LISTAGEM 3.1** Declarando Arrays (*continuação*)

```
13 dtDates(2)="20/7/1969" 'A Apollo 11 aterrissa na Lua
14 dtDates(3)="14/2/1946" 'ENIAC revelado ao público
15 'Declaração com Inicialização
16 Dim sMonths()As String ={"Jan","Fev","Mar","Abr","Maio","Jun", _
17     "Jul","Ago","Set","Out","Nov","Dez"}

18 'Usando arrays
19 Console.WriteLine("")
20 Console.WriteLine("Segundo valor em iValues = {0}",iValues(1))
21 Console.WriteLine("Terceira date em dtDates = {0}",dtDates(2))
22 Console.WriteLine("Décimo primeiro mês do ano = {0}",sMonths(10))
```

ANÁLISE

No primeiro exemplo, `iValues` foi declarada como um array de três membros. Todos os itens do array são inteiros (`Integer`). Tudo faz sentido. A parte potencialmente confusa está na maneira usada para fazer referência a cada membro do array. Isso é mostrado no laço `For...Next` que vai da linha 5 à 7 (abordaremos o laço `For...Next` no próximo capítulo). Observe que os três membros do array estão numerados de 0 a 2. Portanto, o segundo membro do array está na posição 1, e não 2.

3

NOTA

Vale a pena ressaltar que os computadores, diferente das pessoas, sempre começam sua contagem com o zero. É melhor deixar a verdadeira razão disso nos recônditos de suas minúsculas mentes de silicone, mas precisamos estar conscientes desse detalhe, principalmente no que diz respeito a arrays.

O segundo array criado, com os meses do ano, foi declarado e inicializado. Exatamente como você pode inicializar variáveis simples quando as declara, é possível fazer o mesmo com arrays. Nesse caso, no entanto, coloque cada elemento do array em uma lista delimitada por vírgulas, envolvida em chaves, como mostrado na linha 16. Isso criará um array de 12 membros, com cada elemento contendo uma string. Lembre-se, contudo, de que se quiser se referir a cada um, eles estarão numerados de 0 a 11 – portanto, `sMonths(10)` seria "Nov", e não "Out".

A última declaração é de um array dinamicamente dimensionado. Esse array pode depois ser redimensionado para o tamanho correto com a palavra-chave `ReDim` como mostrado na linha 10. Como nos outros tipos, os itens do array dinâmico estão numerados de zero a `I` menos 1. Depois que o array for dimensionado, você poderá usá-lo como qualquer outro. Esse tipo de declaração é útil se o tamanho do array depender do valor de outra variável, portanto, isso não seria conhecido até o tempo de execução.



Tenho repetido o fato de que todos os arrays começam com 0 no Visual Basic .NET porque, anteriormente, o Visual Basic podia não ser operado dessa maneira. Em suas versões anteriores ao Visual Basic .NET, você poderia usar a declaração `Option Base 1` no início de um módulo para assegurar que todos os arrays criados nele comessem em 1. De modo alternativo, ao declarar arrays, seria possível definir os membros inicial e final do array, como será mostrado na declaração a seguir. Nenhuma dessas opções está disponível para os arrays do Visual Basic .NET.

Portanto, a linha de código a seguir não é válida no Visual Basic .NET:

```
Dim sngValues(15 To 51)As Single
```

Atribuição

A atribuição foi simplificada no Visual Basic .NET. Versões mais antigas do Visual Basic (Visual Basic 4.0 a 6.0) possuem duas maneiras diferentes de atribuir um valor a uma variável – uma para variáveis simples (incluindo estruturas e arrays) e outra para variáveis de objeto. Felizmente, os desenvolvedores do Visual Basic .NET decidiram remover o método de atribuição usado para as variáveis de objeto e empregar apenas o utilizado para variáveis simples. Você vai atribuir valores às variáveis (simples ou de objeto) colocando-as à esquerda de um sinal de igualdade, como mostra o código a seguir:

```
iSomeVar =1234  
oObjectVar =New Something()
```



Da versão 4.0 à 6.0 do Visual Basic, as linhas de atribuição mostradas anteriormente teriam aparecido com a seguinte forma:

```
iSomeVar =1234  
Set oObjectVar =New Something
```

Contudo, as regras de quando usar `Set` eram confusas, e a Microsoft removeu a necessidade dessa palavra-chave.

Constantes

As constantes são outra classe de valores que você pode usar em seus programas do Visual Basic .NET. *Constantes* são valores que não se alteram nunca ou durante a vida útil de seu programa. Por exemplo, os meses de um ano, o valor de pi e o servidor de banco de dados a partir do qual seu programa recupera dados, todos são valores constantes. É possível definir um valor como constante quando ele for declarado. Qualquer tentativa de alterar o valor de uma constante será assinalada como um erro enquanto você ainda estiver no IDE (Integrated Development Environment) e antes de tentar executar o aplicativo. As constantes são declaradas usando-se as duas formas mostradas no código a seguir:

CÓDIGO

```
Const PI = 3.1415 As Double
Const DSN As String = "Random"
```

ANÁLISE

Se o tipo da constante não for descrito em uma declaração, o compilador terá de usar o tipo de valor que melhor se encaixar. No entanto, ele nem sempre seleciona o melhor tipo possível. Em geral, quando constantes são declaradas, se o tipo do valor não é incluído, o Visual Basic .NET cria os tipos de variáveis a seguir:

- Longo (Long) Para qualquer número inteiro declarado.
- Duplo (Double) Para qualquer número decimal declarado. (Observação: se o valor for efetivamente muito grande para um tipo duplo, ele será truncado.)
- String Para qualquer valor com caracteres.

**NOTA**

Defina o tipo que quiser quando declarar constantes, exatamente como faz com as variáveis.

Algumas Sugestões para os Padrões de Nomeação

Com todos esses tipos diferentes de variáveis e tantos programadores usando-os, há muitos nomes que podem ser empregados ao declará-las. Isso pode levar a um problema quando as examinarmos posteriormente. A menos que a declaração esteja visível, poderão existir dificuldades para se conhecer o tipo da variável. De maneira semelhante, se o código for herdado de um escritor por outro desenvolvedor, pode ser preciso algum tempo para compreender como ele nomeou suas variáveis. As convenções de nomeação, de preferência compartilhadas, reduzem essas duas espécies de dificuldade pela identificação do tipo da variável.

Uma convenção de nomeação normalmente usada é adicionar um prefixo em minúsculas aos nomes das variáveis. O prefixo identifica o tipo de variável. A Tabela 3.5 mostra sugestões para um conjunto de prefixos.

TABELA 3.5 Convenções de Nomeação Sugeridas

<i>Tipo de Variável</i>	<i>Prefixo</i>	<i>Exemplo</i>
Byte	byt	bytIdade
Curto (Short)	sht	shtContagem
Inteiro (integer)	i ou int	i0velhas
Longo (Long)	l ou lng	lPopulacao
Simples (Single)	sng	sngGolpe

TABELA 3.5 Convenções de Nomeação Sugeridas (*continuação*)

<i>Tipo de Variável</i>	<i>Prefixo</i>	<i>Exemplo</i>
Duplo (double)	d ou dbl	dblJuros
Char	c	cIniciaisIntermediarias
String	s ou str	sNome
Booleano	b	bEstaAberto
Data (Date)	dt	dtDataAluguel
Tipos definidos pelo usuário	Dois ou três caracteres importantes do nome da estrutura	Variáveis criadas, com base em estruturas equivalentes a um ponto ou retângulo poderiam ser chamadas de ptLocal e retTamanho
Constantes	Nenhum prefixo. O nome é todo em maiúsculas, com cada palavra separada pelo caractere sublinhado (_)	PI, TAXA_IMPOSTO
Enumerações	Dois ou três caracteres significativos	disDiaSemana, corCorFundo



Por que essa confusão de um, dois e três caracteres? Tenho de admitir que alterei os prefixos que uso com o passar dos anos. Minha filosofia básica era originalmente empregar só um caractere, para limitar a desordem que os prefixos causam. No entanto, alguns deles só iriam provocar mais confusão. Por exemplo, qual seria o tipo de dado de sValue? Poderia ser o curto (Short), simples (Single) ou String. Para tipos de dados que começam com a mesma letra, estendi o prefixo em alguns caracteres. Devo admitir, contudo, que ainda tendo a usar a letra s para strings porque tem sido empregada com frequência. (Você tem de fazer alguns sacrifícios para evitar a síndrome do esforço repetitivo.)

Quando começar a usar esses prefixos, você pode achar que são um pouco confusos. No entanto, eles logo se tornam familiares, e as informações que fornecem são inestimáveis.

Cálculos Simples

Apenas ter alguns valores definidos não produz um programa. Você precisa fazer algo com eles. É necessário efetuar cálculos matemáticos com os valores ou gerar algo mais substancial. De maneira semelhante, para tornar seus programas mais compreensíveis, com frequência será preciso usar ou criar procedimentos. O Visual Basic .NET fornecerá alguns deles; outros terão de ser escritos. Esses procedimentos vão de operadores, que executam muitos dos cálculos matemáticos comuns, a funções mais complexas que poderiam afetar strings ou números.

Usando Operadores

No Visual Basic .NET, os operadores executam cálculos simples e ‘funções’ semelhantes. A maioria dos operadores deve ser familiar a você como símbolos algébricos comuns. No entanto, alguns deles são exclusivos da programação. A Tabela 3.6 lista os operadores mais usados.

TABELA 3.6 Operadores Comuns do Visual Basic .NET

<i>Operador</i>	<i>Uso</i>	<i>Exemplo</i>
=	Atribui um valor a outro	$x = 6$
+	Soma dois valores	$Y = X + 7$ (Y é igual a 13)
-	Subtrai um valor de outro	$Y = X - 4$ (Y é igual a 2)
*	Multiplica dois valores	$Y = X * 2$ (Y é igual a 12)
/	Divide um valor por outro	$Y = X / 2$ (Y é igual a 3)
\	Divide um valor por outro, mas só retorna um número inteiro	$Y = X \setminus 3$ (Y é igual a 1)
Mod	Abreviatura de módulo; retorna o resto de uma divisão	$Y = X \text{ Mod } 3$ (Y é igual a 2)
&	Associa duas strings	$S = \text{"Olá " \& "Mundo"}$ (S é igual a “Olá Mundo”)
+=	Símbolo para adicionar um valor e atribuir o resultado	$X += 2$ (X é igual a 8)
-=	Símbolo para subtrair um valor e atribuir o resultado	$X -= 3$ (X é igual a 5)
*=	Símbolo para multiplicar um valor e atribuir o resultado	$X *= 6$ (X é igual a 30)
/=	Símbolo para dividir por um valor e atribuir o resultado	$X /= 5$ (X é igual a 6)
&=	Símbolo para acrescentar a strings e atribuição do resultado	$S \&= \text{" , John"}$ (S é igual a “Olá Mundo, John”)
^	Eleva um valor a potência de um expoente	3^4 (3 elevado a 4, retorna 81)

Funções Internas

Além das fornecidas pelo .NET Framework, o Visual Basic .NET possui muitas funções internas. Essas funções geram vários recursos úteis, incluindo a conversão de um tipo de dado em outro, cálculos matemáticos, manipulação de strings e assim por diante. Você deve conhecer algumas delas para poder prosseguir com o estudo do Visual Basic .NET.

Funções de Conversão

Algumas das funções mais importantes disponíveis no Visual Basic .NET são as de conversão. Elas permitem que um tipo de dado seja convertido em outro. As funções de conversão se tornaram ainda mais importantes nessa versão do Visual Basic porque ela é muito mais restrita com relação aos tipos de dados, e não converte automaticamente um tipo em outro como as anteriores faziam.



Se você quiser que o Visual Basic .NET converta de modo automático os tipos de dados, poderá desativar a verificação restrita de tipos adicionando `Option Strict Off` no início de seus arquivos. É bom saber, no entanto, que isso poderia levar a resultados inesperados em seu código (ou seja, erros) se o Visual Basic .NET converter uma variável quando essa operação não for esperada.

Todas as funções de conversão do Visual Basic .NET começam com a letra 'C' (como na palavra conversão) e terminam com uma forma abreviada do novo tipo. Além disso, há uma função genérica, `Ctype`, que pode efetuar conversões para qualquer tipo. A Tabela 3.7 descreve as principais funções de conversão.

TABELA 3.7 Funções de Conversão

<i>Função</i>	<i>Descrição</i>
<code>CBool</code>	Converte para um booleano (<code>Boolean</code>). O que for avaliado como falso (<code>False</code>) ou 0 será configurado como falso (<code>False</code>); caso contrário, será definido como verdadeiro (<code>True</code>).
<code>CByte</code>	Converte para um tipo <code>Byte</code> . Todos os valores maiores do que 255 ou informações fracionárias serão perdidos.
<code>CChar</code>	Converte para um único caractere. Se o valor for maior do que 65,535, ele será perdido. Se você converter uma <code>String</code> , só o primeiro caractere será alterado.
<code>CDate</code>	Converte para uma data (<code>Date</code>). Uma das funções mais avançadas de conversão, <code>CDate</code> pode reconhecer alguns dos formatos mais comuns de inserção de datas.
<code>CDbl</code>	Converte para um tipo duplo (<code>Double</code>).
<code>CInt</code>	Converte para um inteiro (<code>Integer</code>). As frações são arredondadas para o valor mais próximo.
<code>CLng</code>	Converte para um tipo longo (<code>Long</code>). As frações são arredondadas para o valor mais próximo.
<code>CSht</code>	Converte para um tipo curto (<code>Short</code>). As frações são arredondadas para o valor mais próximo.
<code>CStr</code>	Converte para uma <code>String</code> . Se o valor for uma data (<code>Date</code>), apresentará o formato abreviado.

TABELA 3.7 Funções de Conversão (*continuação*)

<i>Função</i>	<i>Descrição</i>
CType	Converte para qualquer tipo. É uma função poderosa que permite a conversão de qualquer tipo de dado em outro. Portanto, a sintaxe dessa função é um pouco diferente das outras.

SINTAXE

A sintaxe de CType é

```
oNovaVariavel = CType(oVariavelAnterior, NovoTipo)
```

onde *oNovaVariavel* e *oVariavelAnterior* são lugares reservados para as variáveis resultante e original da conversão, respectivamente. O tipo que surgirá após a conversão será o inserido em *NovoTipo*. Ele pode ser qualquer variável aceita depois de As em uma declaração, portanto, é possível usar essa função para efetuar conversões para tipos de objeto, enumerações e estruturas, assim como para tipos simples.

Funções de Manipulação de Strings

A maioria das funções anteriores relacionadas a strings no Visual Basic foi substituída no Visual Basic .NET pela funcionalidade interna da classe String (examinaremos a classe String com detalhes no Dia 7). No entanto, você pode ver algumas das funções do código anterior listadas na Tabela 3.8 para se familiarizar com elas.

TABELA 3.8 Funções de Manipulação de Strings

<i>Função</i>	<i>Descrição</i>	<i>Exemplo</i>
Len	Retorna o comprimento de uma string.	<i>iValue</i> = Len("Olá") ('iValue é igual a 3).
Chr	Retorna o caractere com base no valor ASCII ou Unicode inserido.	<i>sValue</i> = Chr(56) ('sValue é igual a letra A).
Asc	Retorna o valor ASCII ou Unicode	<i>iValue</i> = Asc("A") ('iValue é igual a 56).
Left	Retorna caracteres de uma string, começando com o da extrema esquerda. Também precisa da quantidade de caracteres a retornar.	<i>sValue</i> = Left("Olá Mundo", 2) ('sValue é igual a O).
Right	Retorna caracteres de uma string, começando com o da extrema direita (o oposto de Left). Também precisa da quantidade de caracteres a retornar.	<i>sValue</i> = Right("Olá Mundo", 4) ('sValue é igual a undo).

TABELA 3.8 Funções de Manipulação de Strings (*continuação*)

<i>Função</i>	<i>Descrição</i>	<i>Exemplo</i>
Mid	Retorna caracteres que não estão em nenhuma das extremidades de uma string. Mid retorna qualquer número de caracteres. Sua sintaxe é <code>sReturn = Mid(String, Início, Comprimento)</code> onde <i>Início</i> é o caractere a partir do qual retornar, e <i>Comprimento</i> é a quantidade de caracteres (incluindo <i>Início</i>) a retornar. Um recurso interessante é que se você omitir <i>Comprimento</i> , produzirá o retorno de todos os caracteres desde <i>Início</i> .	<p><code>sValue = Mid("Olá Mundo", 3, 5))</code> ('sValue é igual a á Mun).</p> <p><code>sValue = Mid("Olá Mundo", 4)</code> ('sValue é igual a Mundo).</p>
Instr	Encontra uma string dentro de outra. É útil quando se procura alguma string em um arquivo. A sintaxe da função Instr é <code>iReturn = Instr(CharacterInicial, StringPesquisada, ItemProcurado, TipoCorrespondência)</code>	<p><code>iValue = Instr(1, "Olá Mundo", "1")</code> ('iValue é igual a 2). Lembre-se de que a string que você está procurando pode ter vários caracteres, portanto, em uma busca por "Mundo", como a de <code>iValue = Instr(1, "Olá Mundo", "Mundo"))</code> 'iValue é igual a 5.</p> <p><i>CharacterInicial</i> é a posição na <i>StringPesquisada</i>, onde o programa iniciará a busca (a contagem começa em 1). <i>StringPesquisada</i> é onde será executada a busca e <i>ItemProcurado</i> é a string que se quer encontrar.</p> <p><i>TipoCorrespondência</i> determina se a busca irá diferenciar maiúsculas de minúsculas. Se estiver configurado como 0 (correspondência binária), a pesquisa diferenciará maiúsculas de minúsculas. Se for ignorado ou configurado como 1 (correspondência textual), não diferenciará. A variável <i>iReturn</i> seria igual a posição na <i>StringPesquisada</i> onde o <i>ItemProcurado</i> começa. Se a string procurada não for encontrada, <i>iReturn</i> será igual a 0.</p>

TABELA 3.8 Funções de Manipulação de Strings (*continuação*)

<i>Função</i>	<i>Descrição</i>	<i>Exemplo</i>
		<i>InstrRev</i> pesquisa do lado direito da string; no resto, é idêntica a <i>Instr</i> . <i>InstrRev</i> será útil quando você estiver pesquisando uma string que apresente um caminho de diretório e quiser visualizar primeiro os diretórios inferiores.
		<code>iValue = InstrRev("Olá Mundo", "o")</code> 'iValue é igual a 9.
Lcase	Converte uma string em minúsculas.	<code>sValue = LCase("Olá Mundo")</code> 'sValue é igual a olá mundo
Ucase	Converte uma string em maiúsculas.	<code>sValue = UCase("Olá Mundo")</code> 'sValue é igual a OLÁ MUNDO
LTrim	Remove todos os espaços que precedem uma string	<code>Value = Ltrim(" Olá Mundo ")</code> 'sValue é igual a "Olá Mundo "
Rtrim	Remove todos os espaços que sucedem uma string	<code>sValue = Rtrim(" Olá Mundo ")</code> 'sValue é igual a " Olá Mundo"
Trim	Remove todos os espaços que antecedem e sucedem uma string	<code>sValue = Trim(" Olá Mundo ")</code> 'sValue é igual a "Olá Mundo"

Outras Funções Úteis

Finalizaremos com algumas funções geralmente úteis que não se enquadram nas outras categorias. Aqui encontraremos as que permitem a determinação do tipo de uma variável, assim como funções de manipulação de datas. A Tabela 3.9 descreve algumas dessas funções.

TABELA 3.9 Funções Internas Diversas

<i>Função</i>	<i>Descrição</i>
IsArray	Retorna True (verdadeiro) se o parâmetro for um array.
IsDate	Retorna True se o parâmetro for reconhecido como uma data.
IsNumeric	Retorna True se o parâmetro for reconhecido como um número.
IsObject	Retorna True se o parâmetro for algum tipo de objeto.
TypeName	Retorna o nome do tipo de dado do parâmetro, por exemplo, <code>TypeName(sName)</code> retornaria "String".

TABELA 3.9 Funções Internas Diversas (*continuação*)

<i>Função</i>	<i>Descrição</i>
Now	Retorna a data e a hora atual.
Today	Retorna a data atual, com a hora configurada como 0:00:00 a.m. (meia-noite).

Escrevendo Suas Próprias Rotinas

Embora as funções internas sejam bastante úteis, sempre haverá vezes em que você precisará criar suas rotinas. Pode ter de selecionar um conjunto de funções internas que sejam chamadas da mesma maneira ou talvez gerar alguma funcionalidade exclusiva. De qualquer modo, o Visual Basic .NET tornará fácil a criação de suas rotinas.

Há dois tipos de rotinas usadas no Visual Basic .NET. Um deles é a rotina que executa algo, mas não retorna nenhum valor. São as chamadas *sub-rotinas* (ou, na abreviatura, *sub*). O outro tipo de procedimento executa algo, porém retornando um valor. Essas são denominadas *funções*.

Sub-Rotinas

Uma sub-rotina é um bloco de código do Visual Basic .NET que executa alguma tarefa – por exemplo, o método `Console.WriteLine` que você vê em muitos dos exemplos. Ele exibe informações na tela, mas não retorna nenhum valor. Use as sub-rotinas para executar tarefas em seus programas.

Em geral é aconselhável inserir em uma sub-rotina um código que será executado mais de uma vez. De maneira semelhante, se tiver de usar um código em vários aplicativos, é bom que também o coloque em uma sub-rotina. As sub-rotinas permitem que um pequeno trecho de seu programa seja isolado, de modo que, em vez de repetir todo o bloco de código, só seja preciso referenciá-lo pelo nome. Isso não significa que a sub-rotina sempre executará exatamente as mesmas etapas, mas que realizará alguma tarefa. Por exemplo, um receita pode instruir, “Adicione uma porção de vinagre a três de óleo”. Em um momento pode-se misturar uma xícara de vinagre a três de óleo, enquanto em outro, podem ser somente três colheres de sopa de vinagre para nove de óleo. De qualquer modo, foi executada a sub-rotina `CrieVinagrete`. Sim, estou cozinhando o jantar enquanto escrevo isso.

Para criar suas sub-rotinas, use a palavra-chave `Sub`:

SINTAXE

```
Sub NomeSubRotina(Parametro1 As Type,Parametro2 As Type,...
    ➔ ParametroN As Type)
    'Insira o que quiser aqui
End Sub
```

Nessa sintaxe, cada parâmetro define um valor que tem de ser passado para a rotina. A Listagem 3.2 mostra a declaração e o uso de uma sub-rotina.

CÓDIGO**LISTAGEM 3.2** Criando e Usando uma Sub-rotina

```

1 Sub ShowMessage(ByVal Message As String)
2     Console.WriteLine(Message)
3 End Sub

4 ShowMessage("Olá Mundo do Visual Basic .NET")

```

ANÁLISE

Nossa sub-rotina começa com a palavra-chave `Sub`, como vemos na linha 1. A sub-rotina se chama `ShowMessage` e usa um parâmetro quando é chamada. Ela termina com a palavra-chave `End Sub` (linha 3). No interior está o código real executado pela sub-rotina. Nesse caso, ela apenas exibe o conteúdo do parâmetro na janela do console. A linha 4 mostra uma maneira possível de chamar a sub-rotina, passando a string "Olá Mundo do Visual Basic .NET".

Funções

Criar suas funções permitirá que você insira novos recursos em seus aplicativos. Gerar uma função é semelhante a definir novas sub-rotinas, exceto por ter de ser definido o tipo de valor retornado. Dentro do procedimento, identifique o valor a ser retornado, como mostrado abaixo:

SINTAXE

Function NomeFunção(Parâmetro1 As Type, ... ParametroN As Type) As TipoRetornado

'Insira o que quiser aqui

Return ValorRetornado

End Function

Nessa sintaxe, cada parâmetro define um valor que terá de ser passado para a rotina; *TipoRetornado* é o tipo de dado que a função retorna, e *ValorRetornado*, o valor que será retornado pela função. A Listagem 3.3 mostra a declaração e o uso de uma função simples.

CÓDIGO**LISTAGEM 3.3** Criando e Usando uma Função

```

1 Function Volume(ByVal Length As Integer, _
2     ByVal Width As Integer, ByVal Height As Integer) As Integer
3     Return Length * Width * Height
4 End Function
5
6 Console.WriteLine(Volume(3,4,5))

```

Escopo

NOVO TERMO

Escopo é uma dessas palavras adoráveis do jargão da informática que significa “Quem mais pode me ver?”. Formalmente, o *escopo* define a visibilidade das variá-

veis de um programa, isto é, que rotinas poderiam usar uma certa variável. Você pode não querer que todas as rotinas acessem todas as variáveis. Permitir que todas as rotinas conheçam todas as variáveis poderia levar uma rotina a ‘acidentalmente’ alterar o valor de uma variável, introduzindo um erro em seu programa.

Até agora, temos em geral declarado as variáveis por meio da palavra-chave `Dim` dentro dos procedimentos. No entanto, você também pode declarar as variáveis externamente para torná-las disponíveis a vários procedimentos. Se fizer isso, poderá usar duas outras palavras-chave, `Public` e `Private`:

- As variáveis `Public` ficam disponíveis em todo o aplicativo. Elas são *variáveis globais*, que existem *globalmente*, ou, por todo o aplicativo. Devem ser usadas parcimoniosamente, mas serão úteis quando você precisar de algum valor que será empregado em muitos pontos de seu programa, como a conexão a um banco de dados, ou um arquivo.
- As variáveis `Private` ficam disponíveis no módulo ou classe onde são declaradas. Elas são usadas com frequência em aplicativos quando se precisa de uma única variável que possa ser empregada em vários procedimentos. Criando-a com a palavra-chave `Private`, estaremos permitindo que todos os procedimentos de um módulo ou classe acessem a variável. As variáveis `Private` são úteis para compartilhar informações comuns necessárias a uma tarefa, como um valor intermediário que possa ser utilizado por funções diferentes para executar um cálculo.



Quando for criar uma variável, é bom declará-la o mais próximo possível de onde ela for necessária. Se você só for usar uma variável em um procedimento, declare-a dentro dele.

Use as variáveis `Private` e `Public` no nível de módulos com moderação.

Por Que o Escopo É Importante?

O escopo permite que você isole os dados usados pelos procedimentos de seu aplicativo. Muitas versões mais antigas do BASIC não tinham o recurso do escopo, e todas as variáveis podiam ser acessadas e alteradas de qualquer parte do programa. Imagine escrever um programa naquela época – você poderia reutilizar com frequência (propositada ou acidentalmente) uma variável em qualquer local de um programa. Isso poderia levar a uma falha se o valor fosse alterado em algum ponto, só para que se cometesse o erro de ler posteriormente o novo valor, quando o pretendido fosse o original.

O Escopo e os Procedimentos

Exatamente como as variáveis podem ter um escopo, os procedimentos (sub-rotinas e funções) também possuem um. O escopo para os procedimentos significa o mesmo que para as variáveis:

ele descreve em que outro local de seu programa você pode usar o procedimento (ou fora de seu programa, como veremos quando começarmos a criar objetos).

O escopo do procedimento é definido com o uso das mesmas palavras-chave empregadas para o das variáveis. Em geral, ele também possui o mesmo significado.

- **Public** O procedimento pode ser chamado de qualquer parte do aplicativo. Esse é o padrão se você não adicionar nenhuma outra palavra-chave.
- **Private** O procedimento só pode ser chamado a partir de outro que esteja situado dentro do mesmo módulo ou classe onde foram definidos. Isso será útil quando você estiver escrevendo várias rotinas de suporte usadas no decorrer de um cálculo, mas as outras não precisariam usá-lo.

Do mesmo modo que com as variáveis, as palavras-chave adicionais de escopo serão aplicadas quando você estiver criando objetos no Visual Basic .NET. Examinaremos essas palavras-chave posteriormente.

Exemplo de Aplicativo: Calculando um Valor Futuro

3

Agora que você já explorou a criação e uso de variáveis e procedimentos, poderá gerar um aplicativo que execute o cálculo de um investimento. Esse aplicativo permitirá a análise dos benefícios maravilhosos do investimento regular e dos juros compostos. Neste exemplo de aplicativo do console evitaremos a complexidade que uma interface gráfica com o usuário adicionaria. A Listagem 3.4 mostra o código de uma execução do programa.

CÓDIGO/ RESULTADO

LISTAGEM 3.4 0 Programa de Cálculo de Investimentos

```
1 InvestCalc.exe
2 Saldo Inicial: 10000
3 Juros Anuais (por exemplo, para 5%, insira 5):5
4 Depósito Mensal: 200
5 Período do Investimento em Anos: 30
6
7 Se você começar com US$10.000,00
8     e investir US$200,00 por mês
9     durante 30 anos
10    a 5% de juros.
11 Seu saldo final será: $211.129,17
```

Aqui vemos o resultado de se começar com um saldo de US\$10.000 adicionando US\$200 por mês durante 30 anos, com juros fixos de 5 %.

O programa requer que o usuário insira os quatro valores (Saldo Inicial, Juros Anuais, Depósito Mensal e Período do Investimento em Anos). Por sua vez, ele calculará o saldo final. Essa operação é conhecida como cálculo do valor futuro (FV), e o Visual Basic .NET a inclui como uma de suas funções internas. A fórmula do Valor Futuro é

$$FV = \text{DepositoMensal} * ((1 + \text{JurosMensais}) ^ \text{Meses} - 1) / \text{JurosMensais} + \text{SaldoInicial} * (1 + \text{JurosMensais}) ^ \text{Meses}$$

As etapas a seguir descrevem esse procedimento para que você possa compreender melhor como funciona:

1. Comece criando um novo projeto no Visual Basic .NET. Selecione um novo aplicativo do console. O visual Basic .NET criará um projeto com um módulo.
2. Feche a janela do arquivo e o renomeie usando o Solution Explorer. Dê um clique com o botão direito do mouse no nome do arquivo, Module1.vb, no Solution Explorer e selecione Rename. Altere o nome do arquivo para modInvest.vb.
3. Altere o nome de Startup Object também. Dê um clique com o botão direito do mouse no projeto do Solution Explorer e selecione Properties. Na página General, altere o Startup Object para Invest. Ele deve estar na lista suspensa.
4. Você está pronto para começar a codificar. Precisa de ao menos quatro variáveis para armazenar a entrada do usuário. Declare-as como mostra a Listagem 3.5. A maioria desses valores é de números de ponto flutuante, com exceção de Período. Essas declarações devem ocorrer entre as linhas Module e Sub Main() porque as variáveis estarão no nível do módulo.

CÓDIGO

LISTAGEM 3.5 Declarações para o Cálculo do Valor Futuro

```
1 Private dblJurosAnuais As Double = 0
2 Private iPeriodo As Integer = 0
3 Private decSaldoInicial As Double = 0
4 Private decDepositoMensal As Double = 0
```

5. Use a rotina Main para chamar as rotinas de obtenção das entradas do usuário, execução dos cálculos e exibição do resultado, como na Listagem 3.6.

CÓDIGO

LISTAGEM 3.6 Rotina Main

```
1 Shared Sub Main()
2     Dim decResultado As Double
3     'obtencao dos valores das entradas
4     GetInputValues()
5     'calculo
6     decResultado = CalculateFV(dblJurosAnuais, _
```


CÓDIGO**LISTAGEM 3.6** Rotina Main (*continuação*)

```
7      iPeriodo, _  
8      decDepositoMensal, _  
9      decSaldoInicial)  
10     'exibicao do resultado  
11     DisplayResults(decResultado)  
12 End Sub
```

Na Listagem 3.7, cada função principal do aplicativo é uma sub-rotina ou função separada. Isso permitirá que você altere mais rapidamente as técnicas para obter as entradas ou exibir o resultado posteriormente.

6. Adicione o código da Listagem 3.7 para permitir ao usuário inserir informações. O procedimento não usa nenhum parâmetro, nem retorna um valor. Já que esse é um aplicativo do console, você empregará a rotina `Console.Read` para obter valores.

CÓDIGO**LISTAGEM 3.7** A Rotina das Entradas `Console.Read`

```
1 Private Sub GetInputValues()  
2     Console.WriteLine()  
3     decSaldoInicial = CDec(_  
4         GetValue("Saldo Inicial: "))  
5     dblJurosAnuais = CDb1(_  
6         GetValue("Juros Anuais (por exemplo, para 5%, insira 5): "))  
7     decDepositoMensal = CDec(GetValue("Depósito mensal: "))  
8     iPeriodo = CInt(GetValue("Período do investimento em anos: "))  
9     Console.WriteLine()  
10 End Sub  
11  
12 Private Function GetValue(ByVal Prompt As String) As String  
13     Console.Write(Prompt)  
14     Return Console.ReadLine  
15 End Function
```

Observe que a sub-rotina `GetInputValues` chama a função `GetValue`. Esse é um exemplo da criação de rotinas de suporte. Em vez de reescrever o código para solicitar informações ao usuário várias vezes, isole-o e crie um procedimento para executar a tarefa. Dessa maneira, o código resultante para `GetInputValues` será simplificado.

7. Escreva a rotina que exibirá a saída quando for calculada. No final, ela poderá ser exibida em uma janela, mas por enquanto, use o procedimento `Console.WriteLine`, mostrado na Listagem 3.8, para exibir as informações. Esse procedimento deve pegar o valor a ser exibido e não retornar nada.

CÓDIGO**LISTAGEM 3.8** A Rotina de Saída `Console.WriteLine`

```
1 Private Sub DisplayResults(ByVal Resultado As Double)
2     Console.WriteLine()
3     Console.WriteLine("Se você começar com {0:c}, ", decSaldoInicial)
4     Console.WriteLine(" e investir {0:c} por mês", decDepositoMensal)
5     Console.WriteLine(" durante {0} anos", iPeriodo)
6     Console.WriteLine(" a {0}% de juros.", dblJurosAnuais)
7     Console.WriteLine()
8     Console.WriteLine("Seu saldo final será: {0:c}", Resultado)
9 End Sub
```

Essa é uma rotina simples, composta de várias chamadas a `Console.WriteLine` para exibir os valores inseridos e o resultado do cálculo.

8. Execute a cálculo do valor futuro. Essa rotina deve usar os quatros valores como parâmetros e retornar o resultado do cálculo. Como possui um valor de retorno, esse procedimento é uma função. A Listagem 3.9 mostra a função `CalculateFV`.

CÓDIGO**LISTAGEM 3.9** A Função `CalculateFV`

```
1 Private Function CalculateFV(ByVal JurosAnuais As Double, _
2     ByVal Periodo As Integer, _
3     ByVal DepositoMensal As Double, _
4     ByVal SaldoInicial As Double) As Double
5     'divida por 1200 para torná-lo um percentual mensal
6     Dim decJurosMensais As Double = CDec(Juros Anuais /1200)
7     Dim iMeses As Integer = Período * 12
8     Dim decTemp As Double
9     Dim decReturn As Double
10    'precisaremos desse valor em alguns locais
11    decTemp = CDec((1 +decJurosMensais) ^ iMeses)
12    decReturn = CDec(DepositoMensal * _
13        ((decTemp - 1) / decJurosMensais) _
14        + (SaldoInicial * decTemp))
15    Return decReturn
16 End Function
```


Exatamente como em `GetInputValues`, você poderia ter isolado o código que calculou o valor de `decTemp`. No entanto, já que só precisamos desse cálculo nessa rotina e é bem provável que ele não seja mais necessário, foi melhor não fazê-lo.

A Listagem 3.10 mostra o código completo para o exemplo do aplicativo.

CÓDIGO**LISTAGEM 3.10** 0 Programa Completo de Cálculo de Investimentos

```
1 Module Invest
2
3     Private dblJurosAnuais As Double = 0
4     Private iPeriodo As Integer = 0
5     Private decSaldoInicial As Double = 0
6     Private decDepositoMensal As Double = 0
7
8     Sub Main()
9         Dim decResultado As Double
10        'obtenção dos valores das entradas
11        GetInputValues()
12        'cálculo
13        decResultado = CalculateFV(dblJurosAnuais, _
14            iPeriodo, _
15            decDepositoMensal, _
16            decSaldoInicial)
17        'exibição do resultado
18        DisplayResults(decResultado)
19    End Sub
20
21    Private Function CalculateVF(ByVal JurosAnuais As Double, _
22        ByVal Periodo As Integer, _
23        ByVal DepositoMensal As Double, _
24        ByVal SaldoInicial As Double) As Double
25        'divida por 1200 para torná-lo um percentual mensal
26        Dim decJurosMensais As Double = CDec(JurosAnuais / 1200)
27        Dim iMeses As Integer = Periodo * 12
28        Dim decTemp As Double
29        Dim decReturn As Double
30        'precisaremos desse valor em alguns locais
31        decTemp = CDec((1 + decJurosMensais) ^ iMeses)
32        decReturn = CDec(DepositoMensal * ((decTemp - 1) _
33            / decJurosMensais) _
34            + (SaldoInicial * decTemp))
35        Return decReturn
36    End Function
37
38    Private Function GetValue(ByVal Prompt As String) As String
39        Console.Write(Prompt)
```


CÓDIGO**LISTAGEM 3.10** O Programa Completo de Cálculo de Investimentos
(*continuação*)

```
40     Return Console.ReadLine
41 End Function
42
43 Private Sub GetInputValues()
44     Console.WriteLine()
45     decSaldoInicial = CDec(GetValue("Saldo Inicial: "))
46     dblJurosAnuais =
47         CDb1(GetValue("Juros Anuais (para 5%,insira 5): "))
48     decDepositoMensal = CDec(GetValue("Depósito mensal: "))
49     iPeriodo = CInt(GetValue("Período do investimento em anos: "))
50     Console.WriteLine()
51 End Sub
52
53 Private Sub DisplayResults(ByVal Resultado As Double)
54     Console.WriteLine()
55     Console.WriteLine("Se você começar com {0:c}, ", decSaldo Inicial)
56     Console.WriteLine(" e investir {0:c} por mês", decDepositoMensal)
57     Console.WriteLine("durante {0} anos", iPeriodo)
58     Console.WriteLine("a {0}% de juros.", dblJurosAnuais)
59     Console.WriteLine()
60     Console.WriteLine("Seu saldo final será: {0:c}", Resultado)
61 End Sub
62 End Module
```

9. Execute o aplicativo. O resultado deve ser semelhante ao mostrado no início desta seção. Você pode executar o aplicativo a partir do IDE dando um clique no botão Play da barra de ferramentas. No entanto, é provável que a janela exibida desapareça muito rapidamente para que se possa ver a resposta. Em vez disso, execute o programa a partir do prompt de comando processando o executável criado.

Você pode tentar fazer uma experiência surpreendente com esse programa de cálculo. Insira os valores dos depósitos que representam quanto você costuma gastar em um hábito mensal (almoço no trabalho, cigarros, dispositivos eletrônicos). O valor futuro resultante em geral é perturbador.

Resumo

A lição de hoje avançou bastante, examinando os diversos tipos de variáveis que você pode criar no Visual Basic .NET e como empregá-las. Além disso, explorou o conceito de funções e sub-rotinas, tanto as que estão embutidas no Visual Basic .NET quanto as que podem ser criadas. Esses dois tópicos são fundamentais para sua compreensão do Visual Basic .NET e serão usados em todos os tipos de aplicativo.

No Dia 4, “Controlando o Fluxo dos Programas”, continuaremos a explorar a criação de códigos no Visual Basic .NET examinando como você pode gerenciar as decisões em seus programas.

P&R

P Li que o Visual Basic .Net dá suporte à programação ‘sem tipos’. O que é isso?

R Quando você declara o tipo de uma variável ou função, define certas regras: que espécie de informações ela representa, onde pode ser usada e assim por diante. A *programação sem tipos* ocorre quando o tipo não é declarado. Todas as variáveis são, então, objetos que podem conter qualquer tipo de informação. O Visual Basic .NET é o único membro da família Visual Studio que dá suporte a esse tipo de programação.

P Quando quiser criar um procedimento, ele deve ser uma sub-rotina ou uma função?

R Uma resposta breve seria, “Depende”. Você deve criar o tipo de procedimento que forneça a funcionalidade necessária. Alguns são óbvios. Se for preciso gerar um procedimento que execute alguns cálculos ou manipulações e retorne o resultado, é bom usar uma função. Outras rotinas – aquelas que podem ou não ter de retornar um valor – efetivamente levam a uma escolha. Além disso, não há escolha correta, apenas preferências. Se for evidente a inexistência de um valor a ser retornado, selecionar qual tipo de procedimento será criado é uma questão de opinião pessoal e/ou empresarial. Algumas pessoas e empresas sempre usam funções; outras criam sub-rotinas quando necessário.

P Como posso encontrar a lista de todas as funções internas?

R Há duas maneiras pelas quais você pode encontrar informações sobre as funções internas:

- **Ajuda on-line** A ajuda on-line contém descrições e exemplos de código de todas as funções internas. Você pode encontrá-los (on-line) procurando por Visual Basic Language Reference.
- **Object Browser** Se tudo que você precisa é de uma descrição breve de uma função interna, poderá encontrá-la no Object Browser. Abra-o selecionando View, Other Windows e Object Browser. As funções internas se encontram na seção Microsoft.VisualBasic.dll.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Como devo decidir que tipo de variável usar quando estiver trabalhando com números?
2. Qual a maneira correta de chamar esta sub-rotina?

`Function RaiseToPower(ByVal Number As Integer, ByVal Power As Integer) As Long`

- A: `Dim lValue = RaiseToPower(3,4)`
- B: `RaiseToPower 3,4`
- C: `Console.WriteLine(RaiseToPower(3,4))`
- D: `Dim lValue =RaiseToPower 3,4`

3. Se uma variável for declarada com o uso da palavra-chave `Private`, onde poderá ser usada em um programa?

Exercícios

1. Reescreva o exemplo do aplicativo para cálculo do valor futuro de modo que represente o pagamento de um empréstimo em vez de um investimento. Ele deve solicitar a quantia do empréstimo, os juros anuais, o período em meses e calcular o valor de seus pagamentos. A fórmula para esse cálculo é

$$\text{Pagamento} = \frac{\text{QuantiaEmprestimo} * (\text{JurosMensais} * ((1 + \text{JurosMensais}) ^ \text{Meses})}{(((1 + \text{JurosMensais}) ^ \text{Meses}) - 1)}$$

SEMANA 1

DIA 4

Controlando o Fluxo dos Programas

Até agora, este livro abordou a criação de programas de computador, mas para escrever um que seja realmente útil, você precisa de mais alguns elementos essenciais. Até o momento, estivemos desenvolvendo programas compostos de uma série linear de instruções; cada linha era executada em ordem quando o programa era processado. Para que um programa seja realmente interativo, para que use caminhos diferentes no decorrer do código dependendo das entradas que receber, é preciso um novo tipo de instrução, uma instrução de controle. Quase toda linguagem de programação possui instruções de controle, e o Visual Basic .NET não é uma exceção. Nesta lição você aprenderá sobre as duas categorias de instruções de controle:

- Instruções condicionais
- Instruções de laço

Em cada categoria, abordarei diversas variações dos tipos, e você aprenderá que instrução específica é melhor para uma situação em particular.

Tomando Decisões com as Instruções de Controle

As *instruções de controle* são aquelas partes de uma linguagem de programação que existem apenas para determinar que outras partes do programa serão executadas. A determinação ocorre

por causa do valor de algumas variáveis ou outra circunstância, permitindo que o programa aja diferentemente dependendo da situação. A necessidade desse tipo de comportamento é evidente quando você percorre a maioria dos programas que usam pseudocódigo.

NOVO TERMO

Quando descrevemos um problema por meio de uma linguagem que é algo entre nossa maneira habitual de falar e códigos de computador, chamamos essa descrição ‘quase em código’ de *pseudocódigo*. Esta lição fornece vários exemplos do uso desse tipo de linguagem para descrever o que seus programas devem fazer. Muitas pessoas, incluindo os autores deste livro, acham essa maneira útil para planejar o fluxo de seus programas de um modo que leitores também possam compreender. Pode ser muito mais fácil descrever um programa inteiro em pseudocódigo antes de compor o código real (isso se tornará excessivamente importante quando você lidar com programas mais complicados).

Por exemplo, consideremos um código que consistiria apenas em uma pequena parcela de um aplicativo, a saudação na hora da conexão. Usando a linguagem comum para descrever o que esse código faz, você poderia dizer, “Dá as boas-vindas ao usuário que iniciou o programa, usando a saudação apropriada dependendo da hora do dia”. Para um trecho tão pequeno de código, essa provavelmente seria a descrição integral fornecida, e a codificação poderia ser iniciada exatamente nesse ponto. Com relação a essa tarefa específica, o pseudocódigo poderia ter a aparência a seguir:

Quando o programa for iniciado...

Se for antes do meio-dia, exiba "Bom Dia!"

Se for depois do meio-dia, porém antes das 6 da tarde, exiba "Boa Tarde!"

Para qualquer outra hora, exiba "Boa Noite!"

Continue com o restante do programa...

Mesmo esse exemplo simples mostra a necessidade de tomar algumas decisões no código, para permitir que o programa exiba uma mensagem diferente dependendo da hora do dia. Para transformar isso em código, use uma instrução de controle a fim de que possa verificar uma condição (nesse caso, a hora) e, em seguida, escolha que ação tomar.

A Instrução If

A instrução de controle If é a mais simples, e é a mais comum em quase toda linguagem de programação. Ela possui vários formatos diferentes, mas basicamente tem a seguinte aparência:

If <condição> Then

 Código a ser executado se a condição for verdadeira

End If

A <condição> é a parte essencial dessa instrução; ela determina se o bloco de código interno será executado ou não. Essa condição toma a forma de uma expressão, que é uma combinação de valores e operadores avaliada no tempo de execução para gerar um único valor. Já que a instrução

If dá suporte a apenas duas ações possíveis – o código é executado ou não –, a expressão só deve ter dois valores. Isso significa que qualquer expressão usada como uma condição tem de apresentar um resultado definitivo “sim ou não” (Verdadeiro ou Falso), como estes exemplos:

- Estamos em um horário antes do meio-dia.
- Estamos em um horário depois do meio-dia, porém antes das 6 da tarde.
- A quantidade de alunos excede à de cadeiras.

Cada uma dessas expressões é verdadeira ou não. Não há meio termo e, portanto, elas se enquadram como a condição de uma instrução If. Se você tiver alguma dúvida sobre a conformidade de uma expressão em particular, apenas teste-a no pseudocódigo. Por exemplo, considere estas expressões:

- $3 + 5$
- John
- Quarta-feira

Agora, teste uma dessas expressões em seu pseudocódigo: “Se $3 + 5$, então, encerre o programa”. O que isso significa? A expressão “ $3 + 5$ ” não tem um resultado que possa ser avaliado como verdadeiro ou falso e, portanto, a instrução If não faz nenhum sentido. Expressões que retornam verdadeiro ou falso são conhecidas como *booleanas*; abordarei esse assunto com mais detalhes ainda nesta lição.

A seguir, iremos voltar ao exemplo original do pseudocódigo, exibindo uma saudação no início de um programa.

Quando o programa for iniciado...

Se for antes do meio-dia, exiba "Bom Dia!"

Se for depois do meio-dia, porém antes das 6 da tarde, exiba "Boa Tarde!"

Para qualquer outra hora, exiba "Boa Noite!"

Continue com o restante do programa...

Antes de você converter esse pseudocódigo em um programa do Visual Basic .NET, pode valer a pena aproximar essas mesmas linhas (legíveis pelo usuário) um pouco mais do que seria um código. Isso pode ser feito apenas pela reformulação do texto para fornecer mais detalhes:

Quando o programa for iniciado...

Se a hora atual no sistema for menor que meio-dia
então, exiba "Bom Dia!"

Se a hora atual no sistema for igual ou maior que meio-dia
e a hora atual do sistema também for menor que 6 da tarde
então, exiba "Boa Tarde!"

Se a hora atual no sistema for igual ou maior que 6 da tarde

então, exiba "Boa Noite!"

Continue com o restante do programa...

Agora, para escrever esse programa, tudo que você precisa saber é como obter a hora atual do sistema; o resto não deve ser tão difícil. A hora atual do sistema está disponível no objeto `Now()`, que também inclui informações sobre a data atual e exibe várias propriedades úteis como `Hour`, `Minute`, `DayOfWeek` e `Seconds`. Para nossa finalidade, podemos nos dar por satisfeitos usando apenas `Hour`, que retornará a hora atual representada em um período de 24 horas (valores de 0 a 23). Traduzir da maneira mais literal possível do pseudocódigo para o Visual Basic .NET produzirá o programa da Listagem 4.1.

LISTAGEM 4.1 `Greeting.vb`

```
1 Imports System
2
3 Public Class Greeting
4     'Execute o aplicativo
5     Shared Sub Main()
6         Dim dtCurrent As System.DateTime
7         Dim iHour As Integer
8
9         dtCurrent = dtCurrent.Now()
10        iHour = dtCurrent.Hour
11
12        If (iHour < 12)Then
13            Console.WriteLine("Bom Dia!")
14        End If
15        If (iHour >= 12)And (iHour < 18)Then
16            Console.WriteLine("Boa Tarde!")
17        End If
18        If (iHour >= 18)Then
19            Console.WriteLine("Boa Noite!")
20        End If
21    End Sub
22
23 End Class
```

Você pode fazer o download de `Greeting.vb` na página deste livro na Web e testá-lo. Para compilar esse programa, e a maioria dos exemplos desta lição, faça o download desse arquivo ou crie um arquivo de texto, insira o código da Listagem 4.1 nele e, em seguida, salve-o como `Greeting.vb`. Vá para o console de comando (prompt do DOS), use o comando `cd` (change directory, alterar diretório) para se assegurar de que está trabalhando no mesmo diretório do arquivo `Greeting.vb` e compile o programa digitando `vbc r:System.dll t:exe Greeting.vb`. O compilador criará, por padrão, um programa executável com o mesmo nome do arquivo-fonte, portanto, nesse caso,

terminaremos com um arquivo novo, no mesmo diretório, chamado Greeting.exe. A execução do programa executável recém-criado produzirá o resultado apropriado, dependendo da hora do dia em que ele for processado.

Para compreender melhor como essas três condições `If` funcionaram, percorreremos o código e examinaremos o que aconteceu. Você poderia fazer isso dentro do IDE do Visual Studio, mas usaremos o papel para deixar todo esse esforço da máquina para mais tarde.

ANÁLISE

A primeira linha executada foi a 9, que inicializa a variável de data com a hora e data atuais; em seguida, a linha 10 salva a hora atual em outra variável, `iHour`. A primeira instrução `If` é executada na linha 12. Nesse ponto, a expressão `(iHour < 12)` é avaliada. A variável `iHour` é igual ao valor da hora atual, que é 13 (de acordo com o relógio do sistema, para os fins deste exemplo, apenas suponha que a décima terceira hora é 1 da tarde). Agora, a expressão, que foi reduzida a `13 < 12`, será reduzida ainda mais para um simples valor booleano `False` (13 não é menor do que 12, ao menos não pelo que aprendi em matemática). Portanto, um valor falso informa à instrução `If` para não executar seu bloco de código, e o processamento imediatamente passa para a linha 14, `End If`. Essa instrução na verdade é apenas um espaço reservado, portanto, nada acontece até a linha 15, a próxima instrução `If`. Nesse caso, a instrução `If`, `(iHour >= 12) And (iHour < 18)`, primeiro é reduzida para `(13 >= 12) And (13 < 18)` e, em seguida, para `True And True` (Verdadeiro e Verdadeiro). Quando combinamos (`And`) dois valores booleanos, só conseguimos um resultado verdadeiro se ambos o forem, o que parece ser o caso aqui. Assim, a instrução `If` termina com uma condição verdadeira, o bloco de código é executado (linha 16), e a tão estimulante mensagem `Boa Tarde!` é exibida no console. Agora, nosso programa executou todo o seu trabalho; exibimos a mensagem correta para essa hora do dia. O código não encerrou sua execução, no entanto; não há nada que o informe que não há motivos para continuar, portanto, a execução prosseguirá em `End If` e, depois, na última instrução da linha 18. Essa instrução tem a expressão `(iHour >= 18)` como sua condição, que se torna `(13 >= 18)` e termina como falsa. Felizmente a execução passa para a última instrução `End If` na linha 20 e, em seguida, encerra o programa.

4

Estendendo a Instrução If

Embora a Listagem 4.1 seja bem direta e decerto não é o programa mais complicado do mundo, efetivamente levanta uma questão: por que se preocupar com a execução de um código depois que você tiver obtido sua resposta? Esse é um bom ponto. Mesmo que as linhas 18 a 20 da Listagem 4.1 fossem apenas algumas linhas adicionais de execução, poderia haver muitas outras sobrando no final de um programa no mundo real. Nesse programa, a intenção era realmente passar para a próxima instrução `If`, só se a atual fosse falsa. Poderia ter sido mais claro em meu pseudocódigo escrevendo-o da maneira a seguir:

Quando o programa for iniciado...

Se a hora atual no sistema for menor que meio-dia
então, exiba "Bom Dia!"
caso contrário

Se a hora atual no sistema for igual ou maior que meio-dia e a hora atual do sistema também for menor que 6 da tarde então, exiba "Boa Tarde!"

caso contrário

Se a hora atual no sistema for igual ou maior que 6 da tarde então, exiba "Boa Noite!"

Continue com o restante do programa...

Esse exemplo é quase o mesmo que o do código anterior, mas agora está claro que você só passará para a próxima instrução `If` quando o valor da expressão não atender à condição da atual. Como já deve ter sido percebido, a capacidade de aninhar instruções `If`, como aqui, é uma necessidade comum na programação de computadores, portanto, o Visual Basic e a maioria das outras linguagens fornecem uma maneira de expressar exatamente esse conceito com a instrução `If`. Estivemos examinando a forma simples desta instrução:

```
If <condição> Then
    bloco de código
End If
```

Se a avaliação da *condição* apresentar como resultado `True`, então, o bloco de código será executado; se obtiver `False`, o bloco será ignorado, e a execução continuará depois de `End If`. Você pode tornar isso um pouco mais complexo adicionando uma nova cláusula, `Else`. Agora, a sintaxe se parecerá com a descrita a seguir:

```
If <condição> Then
    bloco de código #1
Else
    bloco de código #2
End If
```

Nessa nova estrutura, se a condição encontrar um resultado `True`, o bloco de código #1 será executado, e o programa continuará depois de `End If`. No entanto, se a condição obtiver `False`, então, o bloco de código #2 será executado, e o programa continuará depois de `End If`. Já que a condição deve ser uma expressão booleana e todas elas são verdadeiras ou falsas, um dos dois blocos de código terá de ser processado; essa instrução nunca fará com que os dois ou nenhum deles sejam executados ao mesmo tempo.

Reescrever o código para se beneficiar dessa nova instrução `If` aperfeiçoada produzirá a Listagem 4.3.

LISTAGEM 4.2 Greeting_IFELSE.vb

```
1 Imports System
2
3 Public Class Greeting
4     'Execute o aplicativo
```


LISTAGEM 4.2 Greeting_IFELSE.vb (*continuação*)

```
5      Shared Sub Main()  
6          Dim dtCurrent As System.DateTime  
7          Dim iHour As Integer  
8  
9          dtCurrent =dtCurrent.Now  
10         iHour =dtCurrent.Hour  
11  
12         If (iHour <12)Then  
13             Console.WriteLine("Bom Dia!")  
14         Else  
15             If (iHour >= 12)And (iHour <18)Then  
16                 Console.WriteLine("Boa Tarde!")  
17             Else  
18                 Console.WriteLine("Boa Noite!")  
19             End If  
20         End If  
21     End Sub  
22  
23 End Class
```

Mais simples? Não, nem tanto, mas a Listagem 4.2 é mais eficiente que a 4.1. Nessa nova versão da solução, depois que a saudação correta foi exibida, o programa foi encerrado. Nenhum código desnecessário foi executado.

Nesse programa específico, aninhei outra instrução `If` dentro de cada cláusula `Else`, para testar mais uma condição caso a anterior fosse falsa. Não é só um código desse tipo que pode ser inserido em uma cláusula `Else`, mas como essas instruções `If` aninhadas são freqüentes, o Visual Basic acrescentou mais uma melhoria, a instrução `ElseIf`. Essa instrução combina a funcionalidade de `Else`, com a possibilidade de outra instrução `If` ser *aninhada* ou aparecer imediatamente a seguir. Sua sintaxe é uma condensação do que você teria de escrever se usasse `If` e `Else`. A seguir, vemos a sintaxe de uma instrução `If` em que a cláusula `Else` contém outra instrução `If` aninhada, seguida pela sintaxe que seria usada com a instrução `ElseIf`:

```
If <condição #1> Then  
    bloco de código #1  
Else  
    If <condição #2> then  
        bloco de código #2  
    End If  
End If
```

passa a ser

```
If <condição #1> Then  
    bloco de código #1
```



```
ElseIf <condição #2> Then
    bloco de código #2
End If
```

Observe que falta uma instrução `End If` complementar na segunda sintaxe do exemplo. A instrução `ElseIf` é considerada apenas uma cláusula e, portanto, também faz parte da instrução `If` original. Ainda seria possível inserir uma cláusula `Else` naquela segunda condição ou até mesmo outra cláusula `ElseIf`. Esses dois exemplos são mostrados aqui:

```
If <condição #1> Then
    bloco de código #1
ElseIf <condição #2> Then
    bloco de código #2
Else
    bloco de código #3
End If
```

```
If <condição #1> Then
    bloco de código #1
ElseIf <condição #2> Then
    bloco de código #2
ElseIf <condição #3> Then
    bloco de código #3
End If
```

Mesmo com outra cláusula `ElseIf`, todo o bloco é considerado parte da instrução `If` original, e só uma instrução `End If` é necessária. Voltando ao primeiro exemplo, a exibição da saudação apropriada dependerá da hora do dia, e a cláusula `ElseIf` (veja a Listagem 4.3) permitirá que você simplifique bastante o código.

LISTAGEM 4.3 Greetings_ElseIf.vb

```
1 Imports System
2
3 Public Class Greeting
4     'Execute o aplicativo
5     Shared Sub Main()
6         Dim dtCurrent As System.DateTime
7         Dim iHour As Integer
8
9         dtCurrent = dtCurrent.Now
10        iHour = dtCurrent.Hour
11
12        If (iHour < 12)Then
13            Console.WriteLine("Bom Dia!")
14        ElseIf (iHour >= 12)And (iHour < 18)Then
```


LISTAGEM 4.3 Greetings_ElseIf.vb (*continuação*)

```
15         Console.WriteLine("Boa Tarde!")
16     Else

17         Console.WriteLine("Boa Noite!")
18     End If
19 End Sub
20
21 End Class
```

Instruções If em Seqüência na Mesma Linha

Além da forma em bloco apresentada anteriormente, na qual um conjunto interno de código é inserido entre as instruções If e End If, também é possível expressar uma instrução If em uma única linha. Aqui está um exemplo, em que uma condição é verificada e uma ação tomada, tudo em apenas uma linha:

```
If iHour > 11 Then System.Console.WriteLine("Não é de manhã!")
```

Esse conceito da linha única pode ser ampliado com a introdução de uma cláusula Else:

```
If iHour > 11 Then DoSomething() Else DoSomethingElse()
```

Ele pode até ser usado com mais de uma instrução a ser executada em blocos de código verdadeiros ou falsos (ou em ambos) já que instruções múltiplas podem ser separadas com o uso de dois-pontos, como na linha a seguir:

```
If iHour > 11 Then DoSomething(): DoMore() Else DoSomethingElse()
```

Incluí essa instrução mais para fins de complementaridade do que por alguma necessidade real. Não há nada que você possa fazer com a instrução If em apenas uma linha que não seja possível fazer com a forma habitual em bloco. Colocar todo o código em uma linha não fará com que seja executado mais rápido. Tudo que se consegue, na maioria das vezes, é produzir um código-fonte que ocupa menos espaço em disco e é muito mais difícil de entender. Ocasionalmente, essa versão de linha única pode dar a seu código uma aparência melhor, como mostramos na Listagem 4.4. Nessa listagem, várias instruções If foram necessárias, todas executando um código simples se suas condições forem verdadeiras. Essas situações são muito raras para justificar o uso de outra sintaxe em uma instrução tão simples. Recomendo a adoção da outra forma dessa instrução para que não provoquemos dores de cabeça em nossos colegas programadores.

LISTAGEM 4.4 Instruções If em uma Única Linha Podem Dar ao Código uma Aparência Melhor

```
1 If X=5 Then strChar = "A"
2 If X=23 Then strChar = "B"
```

LISTAGEM 4.4 Instruções If em uma Única Linha Podem Dar ao Código uma Aparência Melhor (*continuação*)

```
3 If X=2 Then strChar = "C"  
4 ...
```

Expressões e Lógica Booleana

Todas as instruções de controle dependem, de alguma maneira, de tomada de decisão, com base no valor de uma variável, constante ou fato relacionado com a situação atual. Independentemente de que valor esteja sendo verificado, o resultado só pode ser verdadeiro ou falso. Como já discutimos, todas as expressões booleanas geram uma entre duas respostas: sim ou não, verdadeiro ou falso. No Dia 3, “Introdução à Programação com o Visual Basic .NET”, você aprendeu sobre as variáveis booleanas, um tipo de variável que só pode conter valores verdadeiros ou falsos. Esses são os únicos tipos de expressão e valores permitidos como parte de uma instrução de controle porque ela precisa tomar uma decisão com base nesse valor. Expressões que não são booleanas apenas não funcionam; não produziram uma resposta positiva/negativa ou verdadeira/falsa, e o programa não saberia o que fazer.

Os dois estados mais simples das expressões booleanas são verdadeiro e falso; todas as expressões booleanas, quando avaliadas, terminam com um desses dois valores. No entanto, não há muitos programas que os usem diretamente. Em vez disso, são criadas expressões mais complicadas que são comparações entre dois valores não booleanos, operações lógicas com valores booleanos ou uma combinação desses dois tipos de expressões.

Operadores de Comparação

O tipo mais comum de expressão usado em programas é uma *comparação*, duas expressões não booleanas com um operador no meio. Os operadores de comparação a seguir estão disponíveis para serem usados em expressões:

- >, maior que
- <, menor que
- =, igual a
- <>, diferente de
- >=, maior ou igual a
- <=, menor ou igual a

Todos esses operadores funcionam tanto com strings quanto com valores numéricos. Um operador de comparação adicional, Like, também está disponível para verificar a correspondência de padrões nas strings. O operador Like permite que você compare uma variável de string com pa-

drões que empreguem caracteres especiais e comuns. Entre os caracteres especiais que podem ser usados com Like estão:

- *, para indicar a quantidade de caracteres adicionais
- ?, para representar um caractere
- #, para representar um dígito (0-9)
- Intervalos ([a-g]), por exemplo) para especificar que qualquer caractere dentro dele deve ser considerado uma correspondência

Desenvolveremos um pequeno programa para testarmos a instrução If e o operador Like. Esse programa (veja a Listagem 4.5) aceitará um valor de teste e um padrão como entradas e, em seguida, verificará se o valor de teste coincide com o padrão.

LISTAGEM 4.5 PatternMatcher.vb

```
1 Public Class PatternMatcher
2     Shared Sub Main()
3         Dim sInput As String
4         Dim sPattern As String
5         Dim sMatch As String
6
7         System.Console.Write("Insira um padrão:")
8         sInput = System.Console.ReadLine()
9         sPattern = sInput
10
11         System.Console.Write("Insira uma string para a comparação:")
12         sInput = System.Console.ReadLine()
13         sMatch = sInput
14
15         If sMatch Like sPattern Then
16             System.Console.WriteLine(sMatch & " Matched with " & sPattern)
17         Else
18             System.Console.WriteLine(sMatch & " não coincidiu com "& sPattern)
19         End If
20     End Sub
21 End Class
```

Depois de inseri-lo em um arquivo de texto (ou fazer seu download) e compilar (vbc /t:exe PatternMatcher.vb), tente executar esse programa com várias entradas. Por exemplo, você poderia usar um padrão como C*T e tentar testar valores como CAT, coat, ct e assim por diante.

Operadores Lógicos

O outro tipo de expressão que pode ser usada como booleana é a que emprega operadores lógicos. Esses operadores trabalham com expressões ou valores booleanos e produzem um resultado

booleano. Já que os valores booleanos são muito semelhantes aos bits (valores binários, 1 ou 0), os operadores lógicos são freqüentemente chamados de *comparações bit a bit*. Os operadores dessa categoria são AND, OR e XOR, que comparam dois valores ou expressões, e NOT, que usa um único valor ou expressão booleana.

Ao usar o operador AND entre dois valores booleanos, você só obtém um resultado igual a verdadeiro se os dois o forem. Com OR, se um dos valores for verdadeiro, então o resultado também será. O operador XOR, também chamado de *exclusive OR*, gera um resultado verdadeiro, se um dos valores for verdadeiro e o outro for falso. NOT é apenas um operador de negação: retorna o oposto de qualquer valor que for usado com ele. A Tabela 4.1 lista todas as combinações possíveis de valores e o que os diversos operadores lógicos produziram em cada caso. Em todos os exemplos, seria possível substituir os valores True (verdadeiro) e False (falso) por expressões que pudessem ser avaliadas até chegar a um valor booleano.

TABELA 4.1 Combinações Booleanas

Expressão	Resultado
TRUE AND TRUE	TRUE
FALSE AND TRUE	FALSE
TRUE AND FALSE	FALSE
FALSE AND FALSE	FALSE
TRUE OR TRUE	TRUE
TRUE OR FALSE	TRUE
FALSE OR TRUE	TRUE
FALSE OR FALSE	FALSE
TRUE XOR TRUE	FALSE
TRUE XOR FALSE	TRUE
FALSE XOR TRUE	TRUE
FALSE XOR FALSE	FALSE
NOT TRUE	FALSE
NOT FALSE	TRUE

A disponibilidade dos operadores lógicos permite que você combine outras expressões e valores para produzir expressões booleanas mais complicadas, como : If X > 3 AND X < 8 Then, ou If (((X+3) * 5) > (Y*3)) AND (SystemIsRunning() OR iHour < 5) Then. Usaremos os dois tipos de expressões, de comparação e lógica, posteriormente em alguns exercícios envolvendo instruções If.

Avaliação Abreviada

De maneira semelhante ao resultado de uma eleição, o de uma expressão booleana em geral é conhecido antes que ela tenha sido integralmente avaliada. Considere esta expressão booleana: $(X > 1) \text{ AND } (X < 10)$. Se X for igual a 1, então, assim que você avaliar o lado esquerdo da expressão (obtendo falso), saberá que o direito é irrelevante. Devido à natureza da instrução AND, não há necessidade de avaliar o outro lado. A expressão inteira será falsa, independentemente de que valor for retornado pela outra extremidade. Chegaríamos a essa conclusão sem precisar pensar muito ao avaliarmos as expressões booleanas, mas nem sempre está tão claro para o computador.

NOVO TERMO

O comportamento que esperamos, não avaliando partes desnecessárias de uma expressão, é chamado *abreviação (short-circuiting)*, mas o Visual Basic .NET, por padrão, não trabalha desse modo. Para fazê-lo abreviar uma expressão booleana, você precisa usar formas alternativas dos operadores AND e OR, ANDALSO e ORELSE. No entanto, não é bom apenas confiar que ele se comportará dessa maneira; um programa simples de teste (veja a Listagem 4.6) pode ser usado para que examinemos exatamente o que acontecerá.

LISTAGEM 4.6 ShortCircuiting.vb

```
1 Public Class ShortCircuiting
2
3     Shared Sub Main()
4         If Test("Esquerda")ANDALSO Test("Direita")Then
5             'execute algo
6         End If
7     End Sub
8
9     Shared Function Test(sInput As String) As Boolean
10         System.Console.WriteLine(sInput)
11         Test = FALSE
12     End Function
13
14 End Class
```

Se a função `Test ()` retornar falso, como fez na Listagem 4.6, então, você saberá o resultado da expressão inteira apenas avaliando o lado esquerdo. A execução do código da Listagem 4.6 produzirá só uma linha de resultado, nesse caso, "Esquerda". Se `Test ()` retornar verdadeiro, os dois lados precisarão ser executados, e o programa exibirá tanto "Esquerda" quanto "Direita". Para testar o comportamento-padrão dos operadores booleanos, tente alterar `ANDALSO` para somente `AND`, e veja que resultado obteve.

Lidando com Múltiplas Possibilidades: A Instrução Select Case

A instrução If pode manipular quase todo tipo de requisito para tomada de decisões, mas na verdade, foi criada para lidar com opções de apenas uma ramificação. Se diversos valores diferentes precisarem ser verificados e uma ação distinta for tomada para cada um, as instruções If poderão se tornar inadequadas. Considere o exemplo de um programa para inserção de dados de imposto de renda com rotinas separadas para manipular cinco categorias diferentes de clientes. A categoria do cliente é baseada na quantidade de pessoas em um endereço domiciliar. Você prefere direcionar os usuários para a rotina correta com base nesse valor. Com o uso das instruções If, o código pode ficar com a aparência da Listagem 4.7.

LISTAGEM 4.7 Usando Muitas Instruções If Aninhadas

```
1 ...
2 If lngQuantidadePessoas = 1 Then
3     Call RetornaImpostoPerCapta ()
4 ElseIf lngQuantidadePessoas = 2 Then
5     Call RetornaImpostoDuasPessoas ()
6 ElseIf lngQuantidadePessoas = 3 OR lngQuantidadePessoas = 4 Then
7     Call RetornaImpostoDomicílioMédio ()
8 ElseIf lngQuantidadePessoas > 4 AND lngQuantidadePessoas < 10 Then
9     Call RetornaImpostoDomicílioGrande()
10 Else
11     Call RetornaImpostoDomicílioMuitoGrande ()
12 End If
13 . . .
```

Quando você começar a testar mais do que algumas opções possíveis, todas as diversas cláusulas If se tornarão excessivamente complexas. Para manipular o teste com múltiplos valores ou vários conjuntos de valores, o Visual Basic incluiu a instrução Select Case, que tem a sintaxe a seguir:

```
Select Case <variável ou expressão sendo comparada>
    Case <valor ou intervalo de valores>
        bloco de código
    Case <valor ou intervalo de valores>
        bloco de código
    Case Else
        bloco de código
End Select
```


O uso da instrução `Select Case` no lugar das instruções `If` da Listagem 4.7 produzirá o código alternativo mostrado na Listagem 4.8.

LISTAGEM 4.8 O Comando `Select Case` Pode Simplificar Muito Seu Código

```

1 Select Case lngQuantidadePessoas
2     Case 1
3     Call RetornaImpostoPerCapta ()
4     Case 2
5     Call RetornaImpostoDuasPessoas ()
6     Case 3,4
7     Call RetornaImpostoDomicílioMédio()
8     Case 5 to 9
9     Call RetornaImpostoDomicílioGrande ()
10    Case Else
11        Call RetornaImpostoDomicílioMuitoGrande ()
12 End Select

```

A cláusula `Case Else` é usada exatamente como a cláusula `Else` em uma instrução `If`, exceto por, nesse exemplo, ser executada se nenhuma das condições for atendida. Observe que, na Listagem 4.8, apenas uma das condições poderia ser verdadeira a cada vez. Não há sobreposição entre as diversas condições `Case`, o que faz muito sentido. Na verdade, as sobreposições não são evitadas de modo algum pelo Visual Basic; é possível ter condições sobrepostas nas quais mais de uma condição `Case` pode coincidir com um valor específico. Se esse for o caso, só a primeira condição que tiver uma correspondência será executada porque o programa sai da instrução `Select Case` depois que uma correspondência foi encontrada e o bloco de código apropriado foi processado. Embora não cause um erro, a sobreposição de condições pode ser confusa para o programador, e é melhor evitá-la apenas por isso.

4

Faça	Não Faça
Aborde sempre toda condição possível incluindo uma cláusula <code>Case Else</code> . Isso capturará toda entrada inesperada para as quais você não usou outra instrução <code>Case</code> . Seu programa ficará mais consistente.	Não use múltiplas condições em sua instrução <code>Select Case</code> se um único valor puder ter correspondência com mais de uma delas. Esse tipo de código não é um erro para o Visual Basic, mas será difícil de entender o que não é muito desejável.

Laços

Até agora, nesta lição, você aprendeu a controlar que código será executado com o uso das instruções `If` e `Select`, mas há outra necessidade muito comum – a de processar o mesmo código

várias vezes. Esse requisito é manipulado por meio de outra espécie de instrução de controle, o laço (loop).

Vários tipos diferentes de laços estão disponíveis no Visual Basic, todos podem executar a maioria das tarefas, mas cada um foi projetado para atender a uma finalidade específica. Começaremos nosso estudo da repetição examinando o mais básico dos laços, For...Next.

For...Next

O objetivo do laço é executar um bloco de código várias vezes, em geral parando quando alguma condição é verdadeira. (Embora um laço não tenha de parar realmente; essa situação é chamada de *laço infinito*.) O laço For executa um bloco de código um número específico de vezes. A sintaxe desse controle é

```
For <variável do contador> = <valor inicial> to <valor final>  
    Código a ser executado  
Next <variável do contador>
```

A inclusão da variável do contador depois da instrução Next final é opcional, mas ela ajuda a indicar à que laço For essa instrução Next pertence e é boa prática de programação.

A Variável do Contador

A variável do contador é incrementada a cada passagem pelo laço, do valor inicial ao final. Quando esse valor final é atingido, o laço encerra sua execução, e o programa continua na linha imediatamente após a instrução Next. Para visualizar esse conceito com alguns valores reais, criaremos uma versão no Visual Basic .NET do primeiro programa que escrevi (veja a Listagem 4.9).

LISTAGEM 4.9 Exibindo Seu Nome Repetidamente

```
1 Dim iCounter As Integer  
2 For iCounter = 1 to 10  
3     System.Console.WriteLine("Duncan Mackenzie")  
4 Next iCounter
```

É claro que em geral uso valores na casa dos milhares...pois adoro ver meu nome rolando na tela!

A variável do contador é real e, exatamente como na Listagem 4.9, deve ser declarada antes de você usá-la como parte de seu laço. Também é importante se certificar de empregar o tipo de dado correto para essa variável. Na Listagem 4.9, a variável iCounter seria utilizada para armazenar valores de 1 a 10, o que torna os tipos de dado inteiro (Integer) e byte os mais adequados. Em outras situações, poderíamos estar lidando com números muito maiores e, portanto, precisar de um inteiro longo (Long Integer). Para obter mais informações sobre os diversos tipos de dados, incluindo que intervalo de valores cada um pode conter, recorra ao Dia 3. Como já mencio-

nado, a variável do contador é incrementada a cada passagem pelo laço, o que com frequência é útil porque ela pode ser usada em seu código.

Faça	Não Faça
<p>Use o tipo de dado mais apropriado para a situação; não empregue o inteiro longo se o inteiro puder atender bem. Sendo o intervalo fechado ou podendo crescer muito, certifique-se de usar o tipo de dado que melhor manipule o maior intervalo possível para o contador.</p>	<p>Não altere o valor da variável do contador dentro do laço. A funcionalidade interna do laço For aumenta o valor da variável do contador sempre que o código do laço é executado, mas você não tem impedimentos para alterar por sua própria conta esse valor. Resista à tentação, ela só resultará em erros desconhecidos e em um código incompreensível.</p>

A Listagem 4.10 mostra como você poderia usar a variável do contador como parte de seu código. Uma pequena função chamada `WeekDayName` produz uma listagem dos dias úteis da semana.

LISTAGEM 4.10 Usando a Variável do Contador

```

1 Public Class DaysOfTheWeek
2
3     Shared Sub Main()
4         Dim sDayName As String
5         Dim iFirstDay As Integer
6         Dim iLastDay As Integer
7         Dim iCurrentDay As Integer
8
9         iFirstDay = 2
10        iLastDay = 6
11        For iCurrentDay = iFirstDay to iLastDay
12            System.Console.WriteLine(WeekdayName(iCurrentDay))
13        Next iCurrentDay
14
15    End Sub
16
17    Shared Function WeekdayName(ByVal iDayNumber As Integer) As String
18        Dim sWeekdayName As String
19
20        Select Case iDayNumber
21            Case 1
22                sWeekdayName = "Sunday "
23            Case 2
24                sWeekdayName = "Monday "
25            Case 3
26                sWeekdayName = "Tuesday "
```


LISTAGEM 4.10 *(continuação)*

```
27         Case 4
28             sWeekdayName = "Wednesday"
29         Case 5
30             sWeekdayName = "Thursday"
31         Case 6
32             sWeekdayName = "Friday"
33         Case 7
34             sWeekdayName = "Saturday"
35
36         Case Else
37             sWeekdayName = "Invalid Day Number"
38     End Select
39     Return sWeekdayName
40 End Function
41 End Class
```

Observe que na Listagem 4.10, Sunday é considerado o primeiro dia, portanto, esse código produziria os resultados a seguir:

Monday
Tuesday
Wednesday
Thursday
Friday

**NOTA**

Esse exemplo ainda não está pronto para produção porque gerará a lista de nomes apenas em inglês, não levando em consideração as configurações da máquina. Há outras maneiras, um pouco mais complicadas, de conseguir essa funcionalidade integral com o suporte a toda as configurações regionais dos usuários. Retornaremos a esse tópico no Dia 8, "Introdução ao .NET Framework".

Especificando o Valor do Incremento com o Uso de Step

Nos exemplos anteriores, a variável do contador era incrementada em 1 cada vez que o laço era percorrido, mas é possível especificar o valor desse aumento. Depois do trecho com o valor final da instrução For, você pode inserir Step <valor do aumento>, e a variável do contador será incrementada com base no valor que for fornecido. Usando o exemplo do código na Listagem 4.11 como ponto de partida, tente valores diferentes para First, Last e Increment e veja os resultados.

LISTAGEM 4.11 ForExample.vb

```
1 Imports System
2 Public Class ForExample
3
4     Shared Sub Main()
5         Dim iCounter As Integer
6         Dim iFirstValue As Integer
7         Dim iLastValue As Integer
8         Dim iIncrement As Integer
9
10        iFirstValue = 0
11        iLastValue = 100
12        iIncrement = 10
13        For iCounter = iFirstValue to iLastValue Step iIncrement
14            System.Console.WriteLine(iCounter)
15        Next iCounter
16
17    End Sub
18 End Class
```

Uma característica interessante e útil de ter a opção `Step` no laço `For` é que ela permite que você percorra um intervalo de valores de maneira invertida. Tente os valores 10, 0 e 1 para as variáveis `First`, `Last` e `Increment`, respectivamente. Nada será exibido, porque `Last` já será menor que `First`, mas se o valor de `Increment` for alterado de 1 para -1, algo interessante acontecerá (não aceite simplesmente o que digo, tente!).

Ah, você conseguiu um laço que é executado o número exato de vezes que quiser, e agora os valores são listados de trás para a frente. É difícil acreditar que possa ficar melhor do que está, mas pode. Espere até chegarmos ao laço `Do`!

while...End while

O laço `For`, embora útil, é limitado. Foi projetado para situações nas quais você sabe quantas vezes quer percorrê-lo, o que não é sempre o caso. Por estar atualizado com relação a esse fato, o Visual Basic possui dois laços mais flexíveis. O primeiro deles, `While...End While`, continua sendo executado enquanto uma expressão booleana específica for verdadeira, como o descrito a seguir:

`While <expressão booleana>`

 Código a ser executado

`End While`

Qualquer expressão booleana válida pode ser usada, exatamente como em uma instrução `If` e, portanto, é possível dar suporte a condições complexas. Por exemplo, um laço `While` pode fornecer com facilidade a mesma funcionalidade do laço `For`, como demonstramos na Listagem 4.12.

LISTAGEM 4.12 `WhileExample.vb`

```
1 Imports System
2 Public Class WhileExample
3     Shared Sub Main()
4         Dim iCounter As Integer
5         Dim iFirstValue As Integer
6         Dim iLastValue As Integer
7         Dim iIncrement As Integer
8         iFirstValue = 0
9         iLastValue = 100
10        iIncrement = 10
11        While iCounter <= iLastValue
12            '<Insira o bloco de código aqui>
13            iCounter = iCounter + iIncrement
14        End While
15    End Sub
16 End Class
```

É claro que reproduzir a funcionalidade do laço `For` não é uma maneira útil de usar seu tempo. Não precisamos selecionar apenas um laço; temos de utilizar todos! É preferível empregar o laço `While` na execução de operações mais complicadas, como varrer um array em busca de um trecho específico de dados. Na Listagem 4.13, você se preparará para essa pesquisa carregando um array com algumas strings. Em seguida, usando o laço `While`, varrerá o array até ultrapassar a extensão dele ou encontrar a correspondência que procura.

LISTAGEM 4.13 `WhileSearch.vb`

```
1 Imports System
2 Public Class WhileExample
3     Shared Sub Main()
4         Dim iCounter As Integer = 0
5         Dim arrList(9) As String
6         Dim iMatch As Integer = -1
7         Dim sMatch As String
8         sMatch = "Winnipeg"
9         arrList(0) = "San Diego"
10        arrList(1) = "Toronto"
11        arrList(2) = "Seattle"
12        arrList(3) = "Londres"
13        arrList(4) = "Nova York"
```


LISTAGEM 4.13 WhileSearch.vb (*continuação*)

```
14      arrList(5) = "Paris"
15      arrList(6) = "Winnipeg"
16      arrList(7) = "Sydney"
17      arrList(8) = "Calgary"
18      arrList(9) = "Orlando"
19      While iCounter <= 9 AND iMatch = -1
20          If arrList(iCounter)Like sMatch Then
21              iMatch = iCounter
22          Else
23              iCounter = iCounter + 1
24          End If
25      End While
26      If iMatch < -1 Then
27          System.Console.WriteLine("Matched" & iMatch)
28      End If
29  End Sub
31 End Class
```

O operador de comparação Like é usado na Listagem 4.13, o que permite que as correspondências sejam examinadas com o uso de curingas. Ao percorrermos esse código, poderemos perceber que o programa passa para outra etapa logo depois que cada uma das condições de saída é avaliada como verdadeira. O laço While é muito útil, mas ainda há outro tipo de laço disponível, Do. Se você for como eu, provavelmente esquecerá tudo que viu sobre a instrução While, depois de ter usado o laço Do.

4

Laço Do

O laço Do, além de ser mais simples, apresenta a estrutura de laço mais flexível disponível no Visual Basic. Sua sintaxe, na forma mais básica, é :

```
Do
    Código a ser executado
Loop
```

A sintaxe, contudo, não especifica nenhuma condição de saída, portanto, o código interno continuará a ser executado de modo infinito. Esse problema é facilmente contornado porque a instrução Do dá suporte a duas maneiras de iniciar as condições de saída. As opções disponíveis são While <condição>, que faz com que o laço seja executado enquanto a condição for verdadeira e Until <condição>, que permite a continuação do processamento do laço enquanto a condição for falsa.

Qual deve ser usada, While ou Until? Tecnicamente, não tem nenhuma importância; você pode tranquilamente empregar qualquer das duas opções apenas utilizando uma negação em sua con-

dição de saída quando apropriado. Esses dois exemplos de código se comportarão da mesma maneira:

```
Do While iMatch = 3
```

```
Loop
```

```
Do Until Not (iMatch = 3)
```

```
'iMatch 3 também funcionaria
```

```
Loop
```

Portanto, `While` ou `Until` não são tão diferentes em seu efeito, mas o laço `Do` oferece outra opção que fornece ainda mais flexibilidade. Você pode colocar a condição de saída (usando a cláusula `Until` ou `While`) no início (com `Do`) ou no final (com `Loop`) do laço. Isso significa que é possível criar laços como o descrito a seguir:

```
Do
```

```
Loop Until bFound or iCounter > iTotal
```

Diferente de apenas escolher entre `While` ou `Until`, a posição de sua condição de saída terá um efeito maior sobre como seu laço será executado. Se você colocar a condição no início do laço, então, ela será verificada antes de cada passagem pelo código, antes até da primeira vez que isso acontecer. Se essa condição não for atendida, o laço não será iniciado, e o código não será executado nem mesmo uma vez. Ao contrário, se a condição for colocada na instrução `Loop`, então, ela será verificada depois de cada passagem pelo código. Independentemente do valor da condição, o código sempre será executado pelo menos uma vez.

Com todas essas opções, há um total de quatro configurações diferentes para a instrução do laço `Do`, tornando-a, de longe, o método mais flexível de executar laços. Você ainda terá de escolher entre essas quatro opções, portanto, examinaremos alguns itens para ajudar nessa decisão:

- Você pode alternar entre `While` e `Until` apenas usando uma negação na expressão booleana.
- Ao escolher entre `While` e `Until`, use a que não precisar de uma negação na instrução condicional. Isso resultará em uma expressão booleana um pouco mais simples, e em termos de codificação, mais simples em geral é melhor.
- O posicionamento da instrução condicional é muito importante. Se você colocá-la no início do laço, então, ele não será executado de forma alguma se essa condição não for atendida. Se for inserida no final, o laço sempre será processado uma vez.
- Escolha entre as duas posições possíveis definindo se realmente não quer que o laço seja executado caso a condição não seja atendida ou se quer que o processamento sempre ocorra pelo menos uma vez.



O laço `Do While` pode ser usado no lugar de `While` porque eles possuem exatamente o mesmo efeito. Não é raro ver os programadores deixarem o laço `While` de lado para empregar essa instrução.

Condições de Saída

NOVO TERMO

A *condição de saída* de qualquer laço é a expressão que será avaliada para determinar quando ele deve terminar. No caso de um laço `While` ou `Do`, elas são claramente definidas e aparecem no início ou no final dele. No laço `For`, a condição de saída é deduzida pela configuração dos limites superior e inferior. Entretanto, para cada um desses tipos de laço há outra maneira de especificar quando sair deles com o uso da instrução `Exit`. Há uma instrução `Exit` correspondente a cada laço (`Exit For`, `Exit Do` e `Exit While`). Quando a instrução apropriada for executada, ele será abandonado de imediato, e a execução do programa continuará na linha seguinte ao fim do laço.

Embora possam ser encontradas muitas ocasiões em que essas instruções pareçam um meio perfeito de fazer o programa se comportar corretamente, elas são um dos muitos exemplos de prática imprópria de programação. Usando a instrução `Exit`, você na verdade só terá especificado uma parte complementar de sua condição de saída, mas de uma maneira mais do que óbvia. A melhor forma de fazer isso seria adicionar essa segunda condição à condição principal de saída de seu laço. Nos exemplos a seguir, veremos algumas maneiras comuns de empregar as instruções `Exit` e o código correspondente que poderia ser usado como alternativa.

4

Exemplo 1: Usando um Laço For para Pesquisar um Array

Você poderia criar um laço `For` para executar uma varredura em um array de tamanho fixo, abortando-o com `Exit For` quando a correspondência fosse encontrada.

```
For i = 1 to 100
    If arrNomes(i) = "Joe" Then
        System.Console.WriteLine("Encontrado em #" & i)
        Exit For
    End If
Next i
```

O problema existente aqui é que o laço `For` não devia de modo algum ter sido usado, mas não sabemos antecipadamente quantos laços teremos de executar. Uma maneira de tornar esse código mais objetivo seria empregando um laço `Do` que controlasse a passagem pelo limite superior e procurasse a correspondência, como descrito a seguir:

```
i = 1
Do Until i > 100 or arrNomes(i) = "Joe"
    i = i + 1
Loop
```


Exemplo 2: Procurando um Valor de Escape

Nos laços em que se quer capturar a entrada do usuário e ainda permitir que eles a cancelem, duas condições de saída e uma instrução `Exit` em geral são usadas em um deles:

```
iCurrentGuess = 0
iTarget = 5
Do Until iCurrentGuess = iTarget
    iCurrentGuess = GetNextGuess()
    If iCurrentGuess = -1 Then
        Exit Do
    End If
Loop
```

Mais uma vez, a condição efetiva de saída é mais complicada do que esse laço a faz parecer. Uma solução apropriada seria:

```
iCurrentGuess = 0
iTarget = 5
Do Until (iCurrentGuess = iTarget) Or (iCurrentGuess = -1)
    iCurrentGuess = GetNextGuess()
Loop
```

Se essas instruções `Exit` não são adequadas, você deve estar querendo entender por que as abor-do. Bem, embora saiba que poderá evitá-las em um código de sua autoria, a maioria dos programadores em geral trabalha com códigos escritos por outra pessoa, e é importante compreender o que pode ser encontrado neles.

Laços Infinitos

Qualquer laço pode ter erros, mas um incômodo em particular é quando ele é executado continuamente, forçando você a interromper o programa para encerrá-lo. O fato de não ser preciso especificar nenhuma condição em seu laço `Do` o torna um pouco mais propenso a esse tipo de erro. Se o programa for executado e parecer não terminar nunca, e, se ele for baseado em DOS, use a combinação de teclas `Ctrl+C` para causar sua interrupção. Se estiver utilizando o IDE do Visual Basic, use a combinação de teclas `Ctrl+Break` para encerrá-lo.

Faça	Não Faça
Certifique-se de ter uma condição de saída em qualquer laço que criar.	Não torne sua condição de saída muito complexa; ela precisa ser avaliada a cada passagem pelo laço.

Algumas causas comuns dos laços infinitos são o esquecimento de que a variável do contador precisa ser aumentada progressivamente (em laços diferentes de `For`), a reinicialização de uma variável que deveria ser crescente e o uso de uma condição de saída que nunca possa ser atingida.

Implicações sobre o Desempenho

Há várias dicas úteis para ajudá-lo a conseguir o melhor desempenho com os laços em seu código. Primeiro, lembre-se sempre de que toda otimização do desempenho em um laço muitas vezes é mais benéfica do que em outro local do programa. Já que o código do laço é executado repetidamente, qualquer melhoria no desempenho desse código é incrementada pela quantidade de iterações. Como exemplo, considere o código da Listagem 4.14.

LISTAGEM 4.14 LoopPerformance.vb

```
1 Public Class LoopPerformance
2
3     Shared Sub Main()
4         Dim i As Integer
5 6         For i = 1 to 1000
7             System.Console.WriteLine(Username())
8         Next i
9     End Sub
10
11     Shared Function Username() As String
12         Dim sName As String
13         sName = System.Environment.UserName
14         Username = sName
15     End Function
16
17 End Class
```

O nome do usuário é fornecido por meio de uma pequena função que usa o .NET Framework para obter suas informações atuais de segurança, mas o mais importante da Listagem 4.14 é que ilustra erros comuns relacionados ao desempenho. A chamada à função `Username` ocorre dentro do laço, o que significa que ela será executada 1.000 vezes, cada uma provavelmente resultando em alguma forma de chamada do sistema operacional para obtenção do nome do usuário atual. Já que não se espera que o usuário atual seja alterado nesse laço, é muito mais eficiente usar um laço de sua autoria em vez disso, como mostra a Listagem 4.15. O valor da função `Username` não é alterado e, portanto, não será incluído nessa segunda listagem.

LISTAGEM 4.15 LoopPerformance_Better.vb

```
1 Public Class LoopPerformance
2
3     Shared Sub Main()
4         Dim i As Integer
5         Dim sName As String
6         sName = Username()
```


LISTAGEM 4.15 LoopPerformance_Better.vb (*continuação*)

```
7
8     For i = 1 to 1000
9         System.Console.WriteLine(sName)
10    Next i
11
12 End Sub
13
14 Shared Function UserName() As String
15     Dim sName As String
16     sName = System.Environment.UserName
17     UserName = sName
18
19 End Function
20
21 End Class
```

Outra dica de desempenho importante, ao usar expressões booleanas, é certificar-se de que as partes mais simples da expressão sejam colocadas em seu lado esquerdo e, em seguida, utilizar as cláusulas de abreviação ANDALSO e ORELSE. Com essas versões dos operadores booleanos, é possível que apenas o lado esquerdo da expressão seja avaliado, portanto é preferível que ele contenha a mais rápida das duas cláusulas.

Aplicativos Que Farão Uso de Seu Conhecimento Recém-Adquirido

Agora que você sabe como usar instruções condicionais, como If e Select Case, e vários tipos de laços, poderá usar essas instruções para criar alguns exemplos interessantes de programas. Para desenvolver esses exemplos, será necessário algo além dos recursos do Visual Basic aprendidos até agora; precisaremos das classes do .NET Framework. Presentes de alguma forma em todos os exemplos anteriores, essas classes são conjuntos de códigos já existentes que foram empacotados e disponibilizados para os programas pelo fato de o Visual Basic ser uma linguagem .NET. Esses conjuntos de código são fornecidos como objetos, o que é em essência uma maneira de representar um conceito ou entidade dentro de códigos. Sempre que um objeto como System.Console ou System.Security.Principal for empregado, estaremos trabalhando com uma parte do .NET Framework. Para fazer isso, em geral é preciso informar ao Visual Basic que se cogita utilizá-lo incluindo uma linha como Imports System.Security.Principal em seu código. Considere esses objetos como parte da plataforma .NET; eles podem trazer para os programas um conjunto enorme de recursos para que os programadores não precisem desenvolver essa funcionalidade individualmente. Esse é um conceito crítico do Visual Basic .NET, mas ainda o abordarei nos próximos capítulos, portanto, não se preocupe se não parecer muito claro neste momento.

Leitura de um Arquivo

Ler um arquivo no disco é uma necessidade comum de muitos programas, portanto é fornecida pelo .NET Framework. Nesta lição, você usará dois objetos diferentes que são parte da seção `System.IO` do Framework, `System.IO.File` e `System.IO.StreamReader`. Eles representam, respectivamente, o arquivo real no disco e sua leitura na memória.

O objeto `StreamReader` é criado com o uso do método `OpenFile` do objeto `File`, sendo necessário especificar o caminho e o nome do arquivo que se deseja abrir. Você pode usar o objeto `StreamReader`, para ler cada linha do arquivo, uma por vez, através de seu método `ReadLine` até que alcance o final do arquivo. Para verificar se esse foi atingido, compare o último valor lido com a constante especial `Nothing`, que é diferente de uma linha vazia e permite distinguir entre uma linha em branco em um arquivo e o fim real de todos os dados. Já que `Nothing` é um tipo especial de valor, utilize o operador `is` para comparar sua string com ele, em vez de um operador comum de igualdade. Na Listagem 4.16, a primeira etapa é inicializar todos os seus objetos, enquanto aponta o objeto `StreamReader` para o arquivo que você deseja ler. É necessário um arquivo de texto para fazer este exercício, mas qualquer de seus outros exemplos de arquivos de programa `.vb` deve servir.

LISTAGEM 4.16 Etapa 1: Configure

```
1 Public Class ReadFromFile
2
3     Shared Sub Main()
4         Dim sFileName As String
5         Dim srFileReader As System.IO.StreamReader
6         Dim sInputLine As String
7
8         sFileName = "MySampleFile.txt"
9         srFileReader = System.IO.File.OpenText(sFileName)
10
11     End Sub
12 End Class
```

Depois de obter seu objeto `StreamReader`, que foi inicializado de modo que apontasse para o arquivo de teste, você poderá usar um laço `Do While` (veja a Listagem 4.17) para ler o arquivo, uma linha por vez. Para que esse programa produza alguma saída, a fim de tornar possível saber qual sua função, também imprimiremos cada uma das linhas ao serem lidas.

LISTAGEM 4.17 Etapa 2: Insira Este Código Acima da Instrução `End Sub` da Listagem 4.16

```
1     sInputLine = "algo"
2     Do Until sInputLine is Nothing
```

LISTAGEM 4.17 Etapa 2: Insira Este Código Acima da Instrução End Sub da Listagem 4.16 (*continuação*)

```
3      sInputLine = srFileReader.ReadLine()  
4      System.Console.WriteLine(sInputLine)  
5  Loop
```

Antes de percorrer o laço pela primeira vez, é melhor certificar-se de que suas condições sejam atendidas, portanto, inicialize sInputLine para assegurar que não seja Nothing. Na Listagem 4.18, você tentará exibir sInputLine mesmo quando for Nothing, mas seria bom adicionar uma instrução If para verificar essa possibilidade.

LISTAGEM 4.18 Adicionado uma Verificação de Nothing

```
1  sInputLine = "algo"  
2  Do Until sInputLine is Nothing  
3      sInputLine = srFileReader.ReadLine()  
4      If Not sInputLine is Nothing Then  
5          System.Console.WriteLine(sInputLine)  
6      End If  
7  Loop
```

Alternativamente (veja a Listagem 4.19), você poderia usar um método de laço um pouco diferente e assegurar que uma saída não seja exibida depois que o final do arquivo tenha sido atingido.

LISTAGEM 4.19 Um Laço Melhor

```
1 Public Class ReadFromFile  
2  
3     Shared Sub Main()  
4         Dim sFileName As String  
5         Dim srFileReader As System.IO.StreamReader  
6         Dim sInputLine As String  
7  
8         sFileName = "MySampleFile.txt"  
9         srFileReader = System.IO.File.OpenText(sFileName)  
10        sInputLine = srFileReader.ReadLine()  
11        Do Until sInputLine is Nothing  
12            System.Console.WriteLine(sInputLine)  
13            sInputLine = srFileReader.ReadLine()  
14        Loop  
15    End Sub  
16 End Class
```

Qualquer um dos métodos funcionará, mas o aplicativo final fornecido na Listagem 4.19 produziu o código mais simples (e, portanto, melhor) dentro do laço. Se você quiser executar esse código, terá de criar um arquivo de texto com um conteúdo de teste no mesmo diretório que seu programa executável compilado.

Um Jogo Simples

Outro uso comum para o laço é consultar repetidamente o usuário solicitando uma resposta, até que se consiga a desejada. Isso pode soar incômodo (de algum modo, parecido com as perguntas de uma criança), mas no tipo certo de programa, pode ser útil. Neste exemplo, você criará um jogo simples de adivinhação de números. Esse jogo em particular foi a maneira que meu pai encontrou para me manter ocupado enquanto esperávamos ser atendidos em restaurantes, embora o jogássemos de uma maneira menos tecnológica. Primeiro, ele escrevia um limite superior e um inferior nas extremidades de cima e de baixo de um guardanapo, 1 e 100, por exemplo, e, em seguida, selecionava um número aleatoriamente (a maneira aleatória de meu pai, e não a dos computadores) e o escrevia atrás do guardanapo. Eu começava a dar palpites de números até que encontrasse o correto, sendo informado a cada vez se meu palpite era muito alto ou muito baixo. Embora duvido que eu tenha sido metódico com relação a isso, essas informações de certo facilitaram bastante. Como parte da luta contínua para informatizar tudo que não precisa ser informatizado, criaremos um programa de computador para jogar “Adivinhe o número!”.

Os aspectos básicos deste programa foram esboçados acima; poderíamos usar minha pequena história como o pseudocódigo para o programa. Seria melhor reformular rapidamente os detalhes em uma definição mais clara do que o programa deve fazer:

1. Solicitar ao usuário um limite numérico superior e inferior.
2. Determinar um número aleatório dentro desse intervalo; ele é o alvo.
3. Solicitar ao usuário um palpite.
4. Se o palpite estiver correto, encerrar o jogo e informar ao usuário quantos palpites foram necessários.
5. Caso contrário, informar o usuário se o palpite é muito alto ou muito baixo e voltar à etapa 3.

Esse pseudocódigo não é complexo, mas há um conceito novo nele que você ainda tem de aprender, a geração de números aleatórios. Felizmente, o .NET Framework fornece uma classe para esse fim, `System.Random`. Com esse objeto, é possível gerar um número aleatório entre um limite superior e um inferior com um código como o descrito a seguir:

```
Dim iTargetNumber,iUpperBound,iLowerBound As Integer
Dim objRandom As System.Random = New System.Random
iUpperBound = 100
iLowerBound = 1
iTargetNumber = objRandom.Next(iLowerBound,iUpperBound + 1)
```


**NOTA**

Esse método de gerar números aleatórios retorna valores que são maiores ou iguais ao limite inferior e menores que o superior ($iLowerBound \leq x < iUpperBound$). Você precisa especificar um valor superior que seja maior do que o mais alto que deseja permitir. O código deste exemplo já faz isso.

Depois que o objeto `Random` tiver sido inicializado, você poderá chamar seu método `Next` sempre que desejar, e obterá um novo número aleatório a cada vez. Agora que sabe como obter um número aleatório, pode gravar o código mostrado na Listagem 4.20 para solicitar os limites ao usuário e, em seguida, usar esses valores para obter seu alvo.

LISTAGEM 4.20 Obtendo as Informações Necessárias com o Usuário

```
1 Public Class NumberGuesser
2     Shared Sub Main()
3         Dim iTargetNumber As Integer
4         Dim iUpperBound As Integer
5         Dim iLowerBound As Integer
6         Dim iCurrentGuess As Integer
7         Dim iGuessCount As Integer
8         Dim sInput As String
9         Dim objRandom As System.Random = New System.Random()
10
11         System.Console.WriteLine("Insira o limite inferior: ")
12         sInput = System.Console.ReadLine()
13         iLowerBound = CInt(sInput)
14
15         System.Console.WriteLine("Insira o limite superior: ")
16         sInput = System.Console.ReadLine()
17         iUpperBound = CInt(sInput)
18
19         'Adivinhação do número
20         iTargetNumber = objRandom.Next(iLowerBound, iUpperBound + 1)
21         System.Console.WriteLine(iTargetNumber)
22
23     End Sub
24 End Class
```

Na Listagem 4.20, você exibiu o número depois que o gerou. Isso é útil para testar seu programa, mas teria de ser removido ou isolado (marcado com um caractere de comentário no início (') para indicar que não deve ser compilado) na versão final. Em seguida, é preciso criar um laço que solicitará repetidamente um novo palpite até que o alvo seja adivinhado. Enquanto o programa estiver executando o laço, também precisaremos de outra variável para manter o registro da quantidade de palpites que foram necessários para chegar ao alvo. Esse código (veja a Listagem 4.21) teria de ser inserido antes da instrução `End Sub` para encerrar o programa.

LISTAGEM 4.21 O Laço das Entradas de seu Jogo

```
1      iCurrentGuess = 0
2      iGuessCount = 0
3      Do While iCurrentGuess < iTargetNumber
4          System.Console.WriteLine("Insira um palpite: ")
5          sInput = System.Console.ReadLine()
6          iGuessCount = iGuessCount + 1
7          iCurrentGuess = CInt(sInput)
8
9          If iCurrentGuess < iTargetNumber Then
10             System.Console.WriteLine("Seu palpite é baixo!")
11         ElseIf iCurrentGuess > iTargetNumber Then
12             System.Console.WriteLine("Seu palpite é alto!")
13         End If
14     Loop
15     System.Console.WriteLine("Você conseguiu em " & iGuessCount & " palpites")
```

Combinadas, as duas listagens de código anteriores produzem o jogo completo que você poderá compilar e tentar ganhar. É bom remover o último comando `WriteLine` da Listagem 4.21 antes de jogar, ou pode ficar muito fácil. Há uma maneira bastante racional e, portanto, chata de jogar, que garante a obtenção da resposta correta com uma certa quantidade de palpites. Você pode apenas fornecer a cada palpite um valor intermediário para o intervalo, por meio do critério alto/baixo, para criar um novo intervalo (com a metade da extensão) entre seu palpite e os limites inferior e superior. No jogo do exemplo, que usa de 1 a 100, esse método assegura a descoberta da solução em sete palpites (ou menos), com base em uma fórmula matemática. Você sabe que fórmula informará a quantidade de palpites necessários para qualquer intervalo de valores? A resposta estará na seção de exercícios desta lição.

4

Evitando Laços Complexos por Meio da Recursão

Em geral, há mais de uma maneira de resolver um certo problema e, embora um laço-padrão possa funcionar, pode existir um modo mais simples. Se um problema for estruturado de modo correto ou você puder reestruturá-lo da maneira certa, então, será possível usar a recursão como alternativa aos laços comuns. Em programação, a *recursão* ocorre quando um programa ou procedimento chama a ele próprio em um esforço para resolver um problema. Os problemas que podem ser solucionados por meio da recursão são aqueles nos quais um subconjunto deles tem exatamente a mesma estrutura que o seu todo. A melhor maneira de explicar esse conceito é com um exemplo, o cálculo de um fatorial. A fórmula para o fatorial de um valor ($n!$) é $n(n-1)(n-2)\dots(n-(n-1))(1)$. Para um valor como 10, a fórmula seria $10 * 9 * 8 * 7 * 6 * 5 * 4 * 3 *$

$2 * 1 = 3628800$. (Observe que o resultado é bem grande com relação ao valor de n .) Essa fórmula pode ser expressa com o uso de um laço For, como vemos na Listagem 4.22.

LISTAGEM 4.22 Factorial.vb

```
1 Public Class Factorial
2     Shared Sub Main()
3         Dim sInput As String
4         Dim iInput As Integer
5         Dim iCounter As Integer
6         Dim iFactorial As Integer
7         System.Console.Write("Insira um número: ")
8         sInput = System.Console.ReadLine()
9         iInput = CInt(sInput)
10        iFactorial = 1
11        For iCounter = 0 to (iInput - 1)
12            iFactorial = (iInput - iCounter) * iFactorial
13        Next iCounter
14        System.Console.WriteLine(iFactorial)
15    End Sub
16 End Class
```

**NOTA**

Quando experimentar esse programa, você precisará manter o valor de seu teste (10 em nosso exemplo) menor ou igual a 12. Qualquer valor maior produzirá um fatorial que excederá o tamanho máximo de uma variável de inteiros (veja o Dia 3). Se tiver de dar suporte a valores mais altos, use outros tipos de variável como o longo.

No entanto, essa não é a única maneira de gerar um fatorial já que essa fórmula também pode ser expressa como $n * (n-1)!$, ou seja, n multiplicado pelo fatorial de $n-1$. Essa expressão define a fórmula recursivamente; a solução de um fatorial inclui outro fatorial. Portanto, essa é uma definição mais clara que a da fórmula $n(n-1)(n-2)...(n-(n-1))(1)$ e, se você o escrever desse modo (veja a Listagem 4.23), produzirá um código que também será mais simples do que a rotina correspondente da Listagem 4.22.

LISTAGEM 4.23 RecursiveFactorial.vb

```
1 Public Class RecursiveFactorial
2     Shared Sub Main()
3         Dim sInput As String
4         Dim iInput As Integer
```


LISTAGEM 4.23 RecursiveFactorial.vb (continuação)

```
5      Dim iCounter As Integer
6      Dim iFactorial As Integer
7          System.Console.WriteLine("Insira um número: ")
8          sInput = System.Console.ReadLine()
9          iInput = CInt(sInput)
10         System.Console.WriteLine(Factorial(iInput))
11     End Sub
12
13     Shared Function Factorial(n as Integer) as Integer
14
15         If n = 1 Then
16             Return 1
17         Else
18             Return n * Factorial(n-1)
19         End If
20
21     End Function
22 End Class
```

O código produzido pode não ser mais curto que o da Listagem 4.22, mas é mais simples e, só por isso, esse é um resultado válido. Observe que na função `Factorial ()`, você procura $n = 1$. Essa verificação assegura que as chamadas recursivas finalmente sejam encerradas, semelhante ao que se consegue com a condição de saída em um laço. Sem ela, como no laço, o programa nunca terá sua execução encerrada. A recursão pode ser usada para resolver muitos problemas, e veremos códigos que a empregam em sistemas de vários tipos (e espero que você escreva o seu).

4

Resumo

Nesta lição, examinamos as instruções de controle, a base de muitos dos programas de computador que você criará no futuro. Com essas instruções, é possível começar a converter processos reais em programas, primeiro passando-os para pseudocódigo e, em seguida, usando esse recurso para gerar o código real necessário.

P&R

P Meu colega me disse que os laços `while` são melhores que os laços `Do` e que nunca devo usar o laço `For`! Que laço é o melhor?

R Embora seja possível usar apenas um laço em todos os seus programas, não se ganha nada fazendo isso. No final, tudo que importa é que seu código seja o mais claro e simples que puder. Empregue o laço que melhor resolver o problema. Em geral, o laço `For`

funciona melhor quando é preciso uma quantidade fixa de iterações; use um laço `Do` ou `While` quando esse não for o caso.

P As instruções `If` de uma linha são mais rápidas que a forma `If...End If`?

R Não. O compilador do Visual Basic converte as duas formas no mesmo resultado, portanto, não há diferença na velocidade de execução desses dois formatos. A diferença principal está na facilidade de manutenção e leitura do código.

P Em versões anteriores do Visual Basic e no Visual Basic for Application (VBA), usei uma forma da instrução `If` chamada `Immediate If` `IIF`. Ela existe no Visual Basic .NET?

R Sim, existe. Mas só como elemento de um conjunto especial de objetos (Microsoft.VisualBasic), projetado para fornecer acesso à instrução `IIF` e muitas outras partes da última versão do Visual Basic que não existem mais no Visual Basic .NET. O fato de essas funções não estarem embutidas na versão mais recente do Visual Basic pode significar que elas realmente não estarão disponíveis em futuras versões. A variante `IIF` é uma forma muito útil da instrução `If`; como função, ela pode ser usada no meio de outra expressão, por exemplo, quando uma string é exibida. No entanto, já que as próximas versões do Visual Basic podem não incluir a instrução `IIF`, você deve evitar seu uso se possível.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Qual dos três métodos de laço disponíveis é mais adequado para um intervalo fixo de valores?
2. Qual dos três métodos de laço é o mais flexível?
3. Por que você deve tentar fazer com que o bloco de código interno em um laço seja o mais simples possível?
4. Suponhamos que você tivesse a expressão booleana a seguir em seu programa: `CalculatedTotalNetWorth(iCustomerID) < 10000 AND dtCurrentDate.Hour > 12`. O que poderia fazer ser feito para assegurar que ela seja o mais eficiente possível?

Exercícios

1. Escreva o inverso do programa NumberGuesser criado nesta lição – um programa com limites superior e inferior que tente determinar que valor foi selecionado pelo usuário. Para cada palpite que o programa de computador gerar, você terá de permitir ao usuário responder com “A” para muito alto, “B” para muito baixo ou = para correto. Se preferir usar o método racional nos palpites de seu programa, então, a quantidade máxima de tentativas necessárias poderá ser encontrada na resolução dessa equação: $(2^N \geq \text{Limite Superior} - \text{Limite Inferior})$ onde N é a quantidade máxima de tentativas. Por exemplo, em um intervalo de 1 a 100, a equação resultaria em 7 porque $2^6 = 64$ e $2^7 = 128$.

PÁGINA EM BRANCO

SEMANA 1

DIA 5

Arquitetura dos Aplicativos na Plataforma .NET

No decorrer deste livro você está aprendendo a usar o Visual Basic .NET para criar aplicativos, mas as questões mais importantes se encontram na fase de projeto do processo de desenvolvimento. Esta lição abordará

- O que é ‘arquitetura do aplicativo’?
- Que opções de arquitetura a plataforma .NET fornece?
- O que está envolvido na escolha de uma arquitetura de aplicativo?

Além da abordagem de alto nível desses tópicos, essa lição também inclui uma discussão de alguns cenários e um estudo da determinação de arquiteturas para cada um deles.

O Que É a Arquitetura do Aplicativo?

Anterior à construção e muito antes de um prédio estar pronto para ser usado, um arquiteto manipulou a fase do projeto. Depois que se obteve um consenso sobre sua concepção básica, são criados esboços que mostram as particularidades do edifício proposto com maior transparência e formam a base do projeto detalhado que deve ser criado antes que qualquer trabalho de construção seja iniciado. No decorrer desse processo, não só no começo, o arquiteto com toda a equipe do projeto, é responsável por produzir o resultado correto.

O conceito de arquiteto, tanto o profissional quanto os aspectos do projeto pelo qual ele é responsável, foi tomado emprestado pela indústria de desenvolvimento de softwares. A semelhança sugerida entre as duas áreas não deve agradar aos verdadeiros arquitetos, e não os culpo por ficarem incomodados. Se os prédios fossem construídos da mesma maneira que a maioria dos sistemas de software, então, passaria o resto de meus dias vivendo a céu aberto. Muito do que está envolvido no desenvolvimento de softwares empresariais é feito sem planejamento suficiente, resultando em sistemas que são instáveis, com manutenção difícil e quase sempre com um orçamento muito alto.

Função do Arquiteto de Softwares

Independentemente da apropriação do nome de uma profissão existente, as semelhanças podem ser traçadas. Quando um edifício está em fase de projeto, os arquitetos usam seu conhecimento avançado tanto da função quanto do projeto para idealizar e planejar sua estrutura e fundação geral com base nos requisitos que forneceram a eles. Com os softwares, o mesmo processo ocorre. Um arquiteto de softwares (ou sistemas) desenvolve um planejamento para construir o sistema com base em requisitos. Nos dois casos, o planejamento no final se torna um projeto detalhado de como deve ser construído, e outras equipes de profissionais se encarregam de manipular a implementação real.

Embora algumas pessoas não consigam ver essa atividade como algo próximo à complexidade do projeto e construção de um imenso arranha-céu, o desenvolvimento de softwares é complicado, e a função de um arquiteto é de extrema necessidade em sistemas de quase todos os tamanhos. Independentemente da dimensão do aplicativo que você estiver construindo ou de quão pequena for sua contribuição individual para o sistema, uma arquitetura terá sido escolhida (talvez de modo informal) e estará sendo usada.



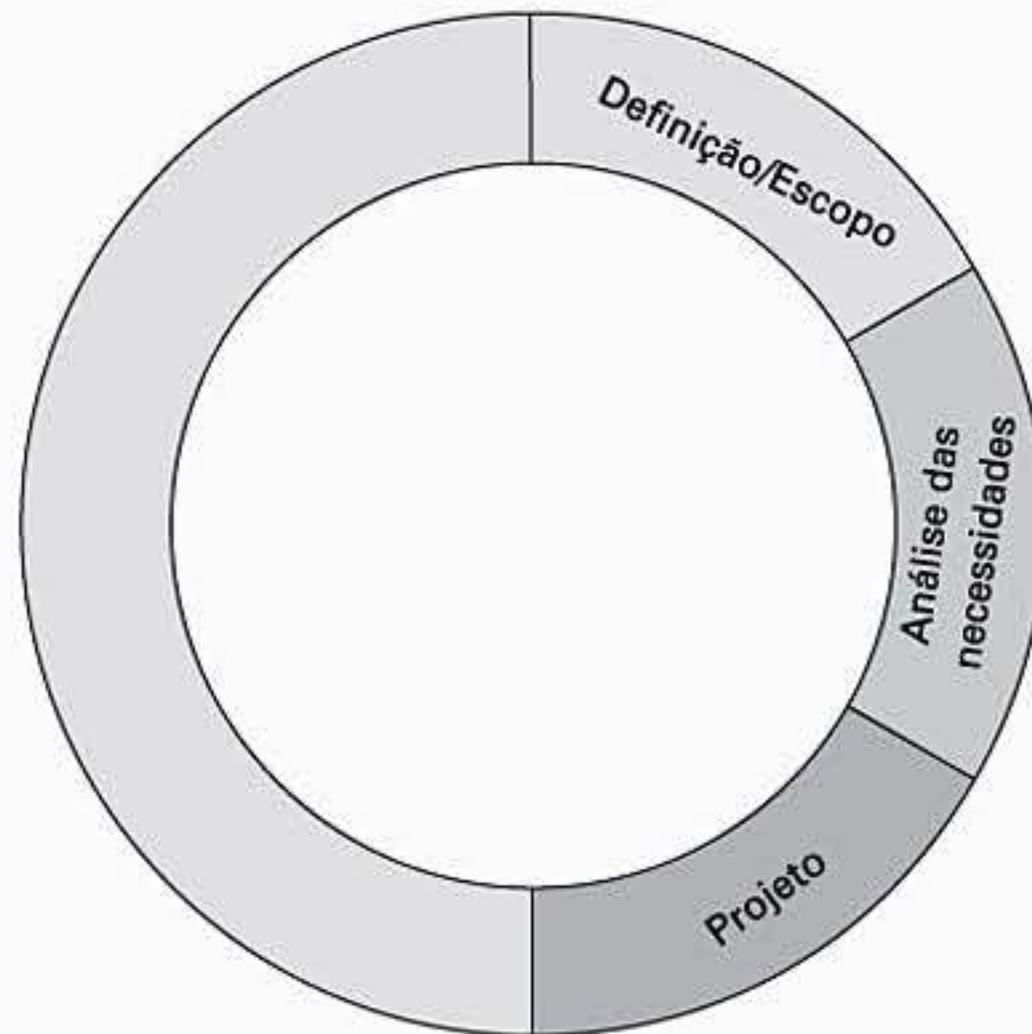
NOTA

Em projetos menores, é possível gerenciar sem um arquiteto, mas o que estará ocorrendo é que você (ou outro membro da equipe de desenvolvimento) terá informalmente assumido essa função. O problema das pessoas que assumem a função de projetista informalmente é que a equipe não tem consciência de quem é o responsável por nenhum dos aspectos específicos do projeto.

Você pode ser um iniciante no Visual Basic ou também no desenvolvimento. Talvez não queira assumir o papel do arquiteto de sistemas no futuro próximo. Mas isso não altera a necessidade de compreender o processo e as decisões sobre a arquitetura que serão tomadas em qualquer projeto com seu envolvimento. No ciclo de evolução do desenvolvimento de softwares, a equipe inteira deve estar envolvida no processo integral, porém o arquiteto é o principal condutor do planejamento nas fases iniciais da Definição/Escopo, Análise das Necessidades e Projeto (veja a Figura 5.1).

FIGURA 5.1

No ciclo de evolução do desenvolvimento de softwares, a arquitetura está envolvida principalmente nas primeiras fases, em que o sistema é projetado.



Depois dessas fases iniciais, o foco passa para a equipe de implementação (desenvolvedores de software) e, em seguida, para os grupos de implantação. O ideal é que a pessoa que assumir a função de arquiteto seja altamente experiente, com bastante conhecimento técnico para avaliar todas as opções disponíveis e habilidade operacional suficiente para interpretar com correção os requisitos do sistema. Em geral, o papel de arquiteto é desempenhado por membros da categoria sênior de um grupo de desenvolvimento.

Que Partes de um Sistema São Consideradas Arquitetura do Aplicativo?

Uma resposta breve para essa pergunta seria ‘todas’, mas ela é muito abrangente para ser útil. Os aspectos que envolvem a arquitetura de um sistema são os ‘grandes detalhes’, como que tipo de interface com o cliente será usada, se serão executados aplicativos Windows ou da Web e como o código da página da Web se comunicará com o banco de dados de back-end. O aspecto principal é que a arquitetura do sistema tem relação com a base e a estrutura do aplicativo, e não com as opções de implementação que serão selecionadas dentro dessa estrutura.

O arquiteto, no entanto, será envolvido na fase de implementação, mesmo se parecer que todos os ‘grandes detalhes’ já foram determinados e só restaram os mais fáceis relacionados à implementação. Ele continua engajado por duas razões principais: para verificar se a arquitetura escolhida provou ser a correta (fazendo os ajustes necessários para lidar com os problemas que aparecerem) e para se certificar de que o projeto esteja sendo implementado como foi planejado. Pode ser difícil de acreditar, mas às vezes os desenvolvedores se desviam do projeto de um sistema, produzindo resultados indesejados.

Para responder a pergunta mais especificamente, em sistemas .NET, a arquitetura deve fornecer uma visão geral de alto nível de cada aspecto do aplicativo. O essencial é o nível de detalhes; a maioria dos itens discutidos na fase da arquitetura será abordada novamente no projeto detalha-

do e na implementação. Por exemplo, parte da arquitetura do sistema pode ser uma discussão sobre segurança, e nesse nível a declaração a seguir poderia ser suficiente para abordar esse tópico: “O aplicativo baseará suas permissões de segurança nos usuários e em grupos do Windows 2000 já em uso na empresa”. Essa declaração é o bastante como ponto de partida; o próximo nível se tornaria mais detalhado e poderia incluir informações sobre que áreas do aplicativo serão protegidas e a quem será concedido acesso para cada área específica.

Na fase de implementação ou desenvolvimento, o programador teria de determinar exatamente como encontrar o logon do grupo e do usuário do Windows, mas esses detalhes decerto não fazem parte da arquitetura do sistema. Embora ela aborde tudo em um nível superior, as áreas-chave a seguir devem ser reconhecidas como parte de qualquer arquitetura de aplicativo:

- Distribuição física e lógica dos códigos (que código será executado onde?)
- Tecnologias usadas na interface com o usuário, bancos de dados e lógica operacional
- Método de comunicação entre componentes diferentes do sistema e entre esse e os outros sistemas da empresa
- Segurança
- Acesso aos dados
- Escalabilidade e disponibilidade

Cada uma dessas áreas é por si só importante, mas a arquitetura é a combinação de todos esses fatores em um projeto que funcione como um todo. A finalidade de cada área será abordada nas seções seguintes.

Distribuição Lógica e Física dos Códigos

No nível da arquitetura, as decisões podem ser tomadas com base em uma organização pretendida, permitindo que os códigos sejam categorizados em grupos como *código de acesso aos dados*, *código de segurança*, *código de interface com o usuário* e outros. Na realidade, qualquer divisão de códigos depende de cada programador e do resto da equipe do projeto. Nada relacionado à plataforma .NET (ou qualquer outra ferramenta de desenvolvimento) o forçará a manter os códigos agrupados de maneira organizada. No entanto, não deixe que a realidade o bloqueie; para as finalidades do projeto, é importante categorizar os códigos mesmo se essa organização nunca existir efetivamente. O que deve ser determinado sobre esses grupos é onde serão executados; serão todos processados como parte de um único aplicativo em apenas uma máquina ou o código de acesso aos dados será executado em um servidor independente?

Mais à frente nesta lição, abordarei o conceito de camadas, em que toda a funcionalidade de um aplicativo é dividida de modo lógico. Essas camadas são, então, tratadas como se fossem conjuntos de código independentes, os quais podem ser movidos à vontade. As decisões a respeito de onde executar cada camada de um aplicativo e de quantos servidores serão usados para cada uma delas constituem esse elemento da arquitetura do sistema.

Tecnologias

Uma das áreas mais objetivas, esse elemento da arquitetura do sistema traça um esboço ‘geral’ das tecnologias que serão usadas. O segredo nessa área é disponibilizar informações suficientes para ser útil, enquanto detalhes que possam não ter sido determinados ou sejam irrelevantes no nível da arquitetura devem ser evitados. Considere esses dois trechos (não se preocupe em compreender os detalhes, só existem para atender à finalidade do exemplo):

Exemplo 1:

A equipe do projeto desenvolverá a interface com o usuário por meio do Visual Basic 6.0 (Service Pack 5) e de um controle de grade ActiveX do Janus. Várias janelas serão criadas, todas com menus, uma grade principal e diversos botões funcionais. Esses formulários empregarão o DCOM para se conectarem com os servidores da camada intermediária, os quais executarão o Windows 2000 Advanced Server (Service Pack 2). Os servidores da camada intermediária hospedarão cinco componentes dentro do ambiente de serviços do componente do COM+ do Windows 2000. Esses componentes operacionais serão DLLs do COM, construídas com o uso do Visual Basic 6.0 (Service Pack 5) e com suas propriedades configuradas como Public. Cada componente será composto de...

Exemplo 2:

A interface com o usuário será baseada em formulários e usará o DCOM para se comunicar com a camada intermediária. Dentro dessa camada, os objetos operacionais do COM terão sido instalados no COM+ e manipularão a comunicação com a camada de dados. O ADO será empregado para conectar a camada intermediária com os dados de back-end, e todas as informações serão retornadas para a camada de apresentação depois de passarem pela comercial.

Embora o Exemplo 1 tenha sido interrompido, na verdade ele não estava nem perto de ser concluído e ainda tinha de transmitir tantas informações úteis quanto o Exemplo 2. O segredo é se lembrar sempre do principal, que é conseguir descrever a arquitetura. Ainda faltam o projeto detalhado e as fases da implementação para manipular a entrada nesse nível de particularidades. Outra regra interessante é nunca passar muito tempo discutindo tecnologias específicas. Tente se concentrar e só discutir a tecnologia em termos do que ela trará para esse projeto.

Comunicação entre Componentes

Se você estiver desenvolvendo um sistema distribuído, para ser executado em várias máquinas, então, alguma forma de comunicação precisa existir entre os diversos componentes. Há muitas opções diferentes, mas no nível da arquitetura, só é preciso ser específico com detalhes que já forem definitivos ou relevantes devido a algum outro aspecto do projeto. Nessa área, pode ser necessário definir o protocolo de comunicação se tivermos de nos conectar com outro sistema ou se os detalhes técnicos forem cruciais para um projeto paralelo como a configuração da

rede/firewall. A plataforma .NET fornece várias opções diferentes para esse tipo de comunicação incluindo o SOAP.

Segurança

Um título indefinido para uma área vaga, a segurança é um tópico que deve ser considerado apenas porque toda discussão acaba por abordá-lo e permanece aí se ele já não tiver sido detalhado. A arquitetura deve detalhar como a segurança será proporcionada (usando o Windows 2000 Active Directory, por exemplo) e conceitualmente como será implementada (“as opções de interface ficarão ocultas” ou “a segurança será verificada a cada página/formulário”). É necessário detalhar exatamente como os programadores implementarão esses recursos.

Acesso aos Dados

Que áreas de código acessarão os dados? Como farão isso? Esta seção detalha como o aplicativo se conectará aos dados (provedor OLEDB usando a segurança do SQL Server, por exemplo). Questões mais complexas, como o projeto e a organização do banco de dados efetivo, são mais bem definidas na fase do projeto detalhado ou como parte da implementação final.

Escalabilidade e disponibilidade

Esses são dois tópicos importantes e complexos e uma das principais razões pela qual a arquitetura dos aplicativos recebe tanta atenção. Eles estão relacionados de muitas maneiras: as técnicas e a arquitetura usada para fornecer um serviço em geral são utilizadas para disponibilizar o outro.

A *escalabilidade* descreve a capacidade de um sistema de manipular cargas maiores, fornecendo o mesmo nível de serviço (tempos de resposta, quantidade de solicitações por segundo e assim por diante) por meio de um incremento no hardware. Lembre-se de que essa não é uma medida de desempenho; um sistema pode ser escalonável e ter um péssimo desempenho. Como assegurar que um aplicativo possa ter uma boa escalabilidade está além do escopo desta lição e deste livro, mas um link para outros recursos será fornecido no final deste capítulo caso você esteja interessado em uma abordagem mais profunda desse tópico.

A *disponibilidade* é uma medida que avalia com que frequência o sistema está em execução e pode processar solicitações. Agora que os aplicativos da Web com interface pública estão sendo desenvolvidos, a necessidade de sistemas que estejam sempre disponíveis está sendo difundida. Em geral esse conceito é descrito como ‘tempo de funcionamento’ e em termos da quantidade de ‘algarismos nove’ existentes na medida de disponibilidade que um sistema tem fornecido. O termo ‘algarismos nove’ se refere ao tempo de funcionamento de 99% (dois algarismos nove), 99,9% (três algarismos nove) ou maior que um sistema possui. Enquanto redigia esta seção, alguns sistemas de uso comercial foram registrados com um tempo de funcionamento igual a cinco algarismos nove (99,999%). Em termos práticos, um site com um tempo de funcionamento de cinco algarismos nove teria estado inativo por somente cinco minutos em um ano inteiro (24 horas por dia, 7 dias por semana). Isso quase parece ridículo, e excede muito meu tempo ativo no

trabalho, mas se você estiver executando um site como o Amazon.com, então, ficar inativo de alguma maneira pelo espaço de tempo que for é inaceitável.

Criar um sistema com esse nível de disponibilidade é uma combinação de projeto e processo. Do ponto de vista do projeto, o sistema deve estar completamente a salvo de estouros de memória ou outras falhas que impediriam o uso contínuo e deve poder ser executado em várias outras máquinas, já que múltiplos grupos de hardware têm de ser usados para fornecer redundância. Do ponto de vista do processo, no entanto, tudo é ainda mais complexo. Por exemplo, atualizações de software terão de ser feitas de maneira seqüencial para que o conjunto de servidores nunca fique integralmente inativo. As discussões sobre o tempo de funcionamento em geral se concentram no sistema operacional ou na tecnologia específica do servidor de banco de dados /servidor de componentes/servidor Web em uso, sugerindo, digamos, que o Windows 2000 fornece uma certa quantidade de algoritmos nove de tempo de funcionamento. Essa não é uma discussão realista. O sistema operacional ou o servidor do banco de dados é só uma parte do sistema, e tudo causa impacto na sua capacidade de manter uma disponibilidade alta.

Arquiteturas Viáveis na Plataforma .NET

Não há uma quantidade fixa definida para as arquiteturas que podem ser desenvolvidas – todas as opções diferentes poderiam ser combinadas para produzir um universo de variações –, mas existem certas arquiteturas generalizadas dentro das quais a maioria dos sistemas .NET falharia.

Os Três Elementos de Qualquer Aplicativo

As diferenças entre todas as arquiteturas possíveis residem em como as três camadas distintas de um sistema de computador são distribuídas. Essas três camadas são listadas aqui, junto a uma descrição breve:

- **Apresentação** Esta camada representa a interface com o usuário e outros sistemas, o aspecto físico público do sistema.
- **Lógica Operacional** Todo os códigos que não estejam envolvidos, em particular, na criação da interface com o usuário ou outros aspectos da camada de apresentação. Esta camada representa o núcleo do aplicativo, o código que realmente faz o trabalho do sistema.
- **Dados** O banco de dados (ou outra fonte de dados como a XML) e o código que o acessa são considerados a camada de dados.

Essas três camadas são a representação lógica de um sistema completo, mas esse pode assumir várias formas: um único programa executável, componentes distribuídos na quantidade de servidores que se desejar, ou um simples site da Web. Independentemente do aplicativo em particular, é útil descrever todos os sistemas com relação a essas três camadas.

Quantas Camadas?

Uma das maneiras mais comuns pelas quais as arquiteturas de aplicativos têm sido descritas é em termos da quantidade de máquinas diferentes que executam partes do sistema. A arquitetura mais simples é apresentada como um sistema de camada única, em que o programa (apresentação, lógica operacional e dados) está todo em uma máquina. Essa é a arquitetura empregada pela maioria dos aplicativos comerciais como o Microsoft Office e muitos softwares empresariais projetados para serem usados por alguns usuários ao mesmo tempo. Só as máquinas que executarem alguma forma de processamento são importantes. Um programa em que os arquivos de dados estejam localizados em um servidor de arquivos também é considerado um aplicativo de camada única porque todo o trabalho efetivo é feito em uma máquina, e o servidor de arquivos apenas fornece um local na rede para armazenar dados.

Por outro lado, quando um servidor de banco de dados (como o SQL Server ou o Oracle) é usado, então, ele é considerado uma segunda camada porque realmente executa um processamento. Os sistemas que possuem um software cliente (em geral na máquina de mais de uma pessoa) que se conecta de modo direto com um banco de dados de back-end são chamados de aplicativos de duas camadas ou cliente/servidor. A arquitetura cliente/servidor é comum nos aplicativos empresariais. Ela permite que muitos usuários trabalhem com o mesmo conjunto de dados enquanto fornece um desempenho muito melhor do que um sistema com base em arquivos, como o Microsoft Access.

Para concluir, a forma mais recente de arquitetura de aplicativo é chamada de três camadas ou *várias camadas* e descreve sistemas em que o código é executado em três ou mais seções distintas. A divisão lógica de um sistema desses também é composta de três camadas, mas o layout físico pode exceder três grupos distintos. Em geral, isso significa que algum tipo de código cliente, como um site interativo da Web ou talvez um aplicativo Windows, que chama um código executado em outro servidor ou conjunto de servidores para manipular a lógica operacional, e um banco de dados de back-end estão sendo usados. Essa arquitetura está se tornando mais popular porque fornece muita flexibilidade para a manipulação de diversos usuários e, portanto, é bem adequada para aplicativos com base na Internet. Há várias maneiras técnicas diferentes de criar aplicativos de três camadas, mas o método recomendado pela Microsoft antes da plataforma .NET se tornar disponível era chamado de Windows DNA.

Windows DNA

O Windows Distributed Network Architecture (ou Distributed interNet Architecture, dependendo de onde você encontrar sua definição), é o conjunto de tecnologias e diretrizes da Microsoft anteriores à plataforma .NET para criação de sistemas de três camadas. Com base na idéia geral de que todo aplicativo pode ser dividido em três camadas, de Apresentação, Lógica Operacional e Dados, o Windows DNA esboçou a ‘melhor’ maneira de desenvolver sistemas distribuídos. Em um aplicativo Windows DNA, o código-cliente era executado como um aplicativo Windows padrão ou como uma interface da Web criada com o Active Server Pages. O código-cliente con-

tinha apenas a lógica relacionada à interface e acessava toda a operacional chamando componentes do COM localizados localmente ou em outro servidor, componentes que, então, se encarregariam de interagir com o banco de dados.

Um princípio essencial do modelo Windows DNA para desenvolvimento de aplicativos é que tudo flui através de três camadas, de modo que a camada de apresentação se comunica apenas com os objetos operacionais, e esses manipulam toda a comunicação com o banco de dados. A camada de apresentação nunca acessa diretamente o banco de dados e, portanto, as três camadas são abstratas, o que proporciona a possibilidade de alternar o banco de dados sem que seja preciso reescrever toda a interface com o usuário, ou criar uma interface totalmente nova sem ser necessário alterar a camada operacional ou o banco de dados. A flexibilidade obtida nesse modelo mais do que compensa qualquer trabalho adicional envolvido para assegurar que as três camadas sejam apropriadamente independentes.

Agora, embora o modelo Windows DNA seja bem detalhado, a implementação real pode ter uma entre várias configurações diferentes. Em um caso, o aplicativo poderia executar tudo em um servidor, com o Active Server Pages (interface da Web) se comunicando com os componentes (provavelmente escritos em Visual Basic 6.0), que, por sua vez, funcionariam com uma instalação local do SQL Server. Apenas uma máquina é usada, mas as três camadas ainda são distintas, e o modelo Windows DNA é mantido. De modo alternativo, o modelo também pode ser expandido para quantas máquinas forem necessárias a fim de manipular a carga do sistema. Um sistema possível poderia empregar um grupo de 20 servidores Web, todos atendendo ao Active Server Pages, que se conectaria a um agrupamento de máquinas com balanceamento de carga executando o mesmo conjunto de componentes do COM, que, por sua vez, acessariam um par de servidores processando o SQL Server em uma configuração de grupo. Apesar da implementação em escala muito maior, o mesmo modelo está sendo adotado, o que demonstra a flexibilidade e escalabilidade do Windows DNA de três camadas que o tornou tão popular.

Onde a Plataforma .NET Se Encaixa?

A plataforma .NET é uma mudança radical no desenvolvimento, mas podem ser aplicados a ela os mesmos conceitos gerais de arquitetura. Muitos sistemas .NET seguem os modelos e exemplos discutidos anteriormente e podem ser classificados como de somente uma camada, cliente/servidor ou até como aplicativos Windows. A tecnologia pode mudar (ASP.NET em vez de ASP, classes .NET em vez de objetos do COM e assim por diante), mas a arquitetura ainda é a mesma. No entanto, algumas outras opções estão disponíveis para aprimorar a arquitetura do Windows DNA quando a plataforma .NET for usada.

O primeiro conceito novo envolve a comunicação entre as camadas. No Windows DNA tradicional, as páginas da Web se comunicavam com os objetos operacionais do COM usando o DCOM (Distributed COM), que é um padrão binário usado para ligar todos os aplicativos configurados para COM através de uma conexão de rede. A abstração entre as camadas é limitada por esse método de comunicação porque só é possível empregar as tecnologias que podem usar o

COM nas camadas de apresentação e operacional. No Windows DNA convencional, você poderia reescrever seus objetos operacionais sem ter de alterar a camada de apresentação ou de dados, mas apenas se estivesse utilizando uma linguagem compatível com o COM (em geral o Visual Basic ou o VC++). A plataforma .NET introduz uma nova maneira de comunicação que pode permitir a abstração real entre as camadas, o SOAP (Simple Object Access Protocol).

A comunicação do SOAP é toda feita com o uso da XML, em vez de um formato binário, e é executada antes do conhecido protocolo HTTP. Para o desenvolvimento de aplicativos distribuídos, isso significa que qualquer linguagem/ferramenta poderia ser usada para criar as camadas do sistema. Seria possível que os componentes já existentes em sua empresa estivessem escritos em Java, que poderia fornecer a camada de apresentação do sistema, enquanto o front-end estaria na plataforma .NET usando o ASP.NET ou um aplicativo Windows. A versão .NET do Windows DNA é até melhor do que antes quando ele empregava o conjunto antigo de tecnologia para desenvolvimento.

Escolhendo uma Tecnologia de Cliente

Uma das escolhas mais polêmicas quando se projeta um sistema é que tipo de cliente será criado, que em geral é uma discussão sobre a possibilidade de do tipo 'thick' ou do tipo 'thin'. As principais opções na plataforma .NET são as mesmas que já estão disponíveis por algum tempo, ASP.NET para criar um aplicativo da Web ou Windows Forms (Formulários ou Aplicativos Windows) para gerar um aplicativo Windows. O segredo é determinar qual é mais adequada para um sistema ou projeto específico. As seções a seguir descrevem as duas tecnologias, abordando seus benefícios e problemas. Posso fornecer a você uma regra simples para informar que tecnologia usar em qual situação? Não, mas ainda nesta lição, disponibilizarei uma lista geral de perguntas que pode ser usada para ajudar a tomar várias decisões sobre a arquitetura, incluindo que tipo de cliente usar.

Sistemas de cliente do Tipo 'Thin' (ASP.NET)

As Active Server Pages são desenvolvidas com o uso das linguagens .NET padrão (como o Visual Basic.NET) e são projetadas para processar solicitações recebidas de usuários e retornar o resultado apropriada em seus navegadores. O código que você escrever para uma página ASP.NET será executado no servidor Web, e não no cliente, que é o ponto mais importante quando se compara esse tipo de interface com qualquer outra. O ASP.NET não possui nenhum requisito de cliente além do acesso à rede e alguma forma de navegador. Podem ser desenvolvidas páginas ASP.NET que exibam um conteúdo HTML mais sofisticado e que termine com uma saída que precise da maior e mais recente versão do Internet Explorer, mas essa é uma opção exclusivamente sua. Também é possível escrever as páginas da Web de modo que elas retornem o conteúdo HTML mais simples já visto, permitindo que até o primitivo navegador Mosaic consiga visualizá-las. O sistema operacional do cliente também não é um problema. Exatamente como na Web, suas páginas podem ser visualizadas de qualquer plataforma em que a Internet esteja disponível.

Continuando nessa mesma linha de raciocínio, a falta de requisitos de cliente significa que para seu aplicativo ser executado, nada (além do requisito básico de um navegador da Web de algum tipo) precisa ser instalado no computador do usuário. Quando você atualizar seu site, alterando as páginas armazenadas em seu servidor, o próximo usuário a acessá-lo verá a nova versão, e nenhum outro problema relacionado à atualização ou distribuição. Todos esses benefícios que resultam da falta de requisitos de cliente tornam a escolha de uma interface da Web relativamente fácil, exceto por dois pontos:

- A experiência que o usuário tem com um navegador da Web ainda não é tão boa quanto com um aplicativo Windows. Considere alguns dos aplicativos Windows que você tem utilizado, como o Microsoft Office, o Visual Studio .NET, o Adobe Photoshop e outros, e perceberá que quase nada na Web chega próximo do nível da interface com o usuário. É possível produzir uma interface na Web que seja semelhante, mas o trabalho necessário será muito maior se compararmos com um aplicativo Windows.
- Um aplicativo ASP.NET em geral só funciona quando o usuário está conectado e seus recursos são afetados pela velocidade dessa conexão.

Alguns itens apenas não funcionam em um sistema com base na Web, como uma experiência off-line igualmente funcional e satisfatória. Por exemplo, o Outlook é capaz de funcionar on-line ou off-line, mas sua contrapartida na Web, o Outlook Web Access, não possui nenhuma funcionalidade off-line. Um programa projetado para trabalhar com arquivos ou outros aspectos ‘locais’ de seu sistema não terá um desempenho tão bom se transferido para uma interface com o usuário com base na Web.

Concluindo, é difícil encontrar razões contra o uso de uma interface da Web, e há muitas vantagens em desenvolver um sistema dessa maneira.

Aplicativos Windows (Windows Forms)

Em geral, é difícil explicar o que quero dizer com ‘aplicativo Windows’ porque esse é o único tipo de aplicativo que a maioria das pessoas já viu. Basicamente, um aplicativo Windows é algo que você pode criar usando uma combinação de janelas e caixas de diálogo para ser executado na plataforma Windows. Quase todos os aplicativos que estamos acostumados a utilizar – o Microsoft Office, o Quicken e até o Solitaire – são exemplos do que pode ser desenvolvido com o uso dos recursos do Windows Forms da plataforma .NET.

Depois de ler a seção anterior sobre o ASP.NET, você pode estar se perguntando por que não usar sempre uma interface da Web, e não é o único. Um aplicativo do Windows Forms possui alguns requisitos para a máquina cliente:

- O .NET Framework deve ser instalado no cliente, enquanto no caso do ASP.NET, só é necessário no servidor Web.
- O Framework só é executado em plataformas Windows.
- Seu aplicativo deve ser instalado.

- Atualizar seu aplicativo em geral significa lidar com todos os microcomputadores de alguma maneira (ferramentas automatizadas ou opções de download de rotinas ou drivers específicos podem eliminar um pouco desse esforço).

A dependência da máquina-cliente possui um efeito colateral pernicioso quando se está lidando com aplicativos públicos; o desempenho de seu aplicativo dependerá de uma máquina sobre a qual você não tem nenhum controle.

Todos esses requisitos fazem parecer que a opção entre cliente do tipo 'thick' (mais robusto) e do tipo 'thin' (mais simples) não é absolutamente uma tomada de decisão. Na verdade, contudo, ainda há várias vantagens interessantes nos aplicativos do Windows Forms que valem a pena mencionar. A primeira grande vantagem é que o desenvolvedor que usar o Windows Forms poderá criar interfaces com o usuário completas e funcionais e será possível fazê-lo muito mais rápido do que com o ASP.NET. Apesar de todos os avanços na tecnologia da Web, ainda é mais difícil gerar uma interface complexa nesse local do que no Windows.

**NOTA**

Muitos desenvolvedores da Web não concordariam com meus comentários de que o ASP.NET é mais difícil de usar no desenvolvimento do que o Windows Forms, mas apesar das diversas virtudes que apresenta, é aí que ele se torna complicado.

Uma segunda vantagem está no desempenho da interface com o usuário, uma interface do Windows em geral responde melhor (responde mais rápido aos cliques e outras ações do usuário). Para concluir, a última vantagem em usar o Windows Forms é que há aqueles itens (uso off-line, processamento mais rápido de objetos locais, como por exemplo, arquivos) que simplesmente não podem ser realizados na Web.

**NOTA**

Com o uso do Dynamic HTML, controles ActiveX e outras tecnologias mais complexas da Web é possível melhorar algumas das questões descritas anteriormente. Ao fazer isso, no entanto, algumas vantagens do ASP.NET serão perdidas já que você estará criando um aplicativo com suporte restrito às plataformas (os controles ActiveX só são executados em certos sistemas operacionais e processadores, e o Dynamic HTML não tem suporte em qualquer navegador) e que pode até ter alguns requisitos de implantação.

Decidindo Que Arquitetura Usar

Tomar uma decisão efetiva quanto à arquitetura, mesmo antes das alterações específicas do aplicativo serem feitas, decerto valerá a pena independentemente de quanto você precisar pagar para um desenvolvedor de sistemas experiente. Se tomar a decisão errada nesse momento, todo o pro-

jeto pode estar fadado à falha. É claro que os erros sempre poderão ser corrigidos, mas nesse caso, muito trabalho poderia ser eliminado se uma nova arquitetura precisasse ser selecionada.

Fatores Essenciais Que Influenciarão Sua Decisão

O fator mais importante na escolha da melhor arquitetura são os requisitos do sistema. Esses detalhes, obtidos com os líderes operacionais do projeto e seus futuros usuários, devem especificar minuciosamente o que o sistema precisa fazer. Após conhecê-los, o segredo será projetar um sistema que possa manipular essas necessidades. Você pode analisar um resumo dos requisitos de sistemas na seção “Exemplos de Cenários” para ter uma percepção dessa tomada de decisões, mas primeiro fornecerei um conjunto de perguntas para serem formuladas quando for interpretar os requisitos ou examinar um grupo já existente deles.



NOTA

Com base em minha experiência, e por ter observado a de outras pessoas que não foi muito agradável, recomendo nunca desenvolver um sistema (ou projetar estimativas ao desenvolver um) com base nos requisitos interpretados pelos outros. Por outros quero dizer alguém que não seja você ou um membro de sua equipe em cujo trabalho confie. Requisitos incorretos em geral resultam em um sistema inadequado e terminarão definitivamente em uma estimativa inválida. Depois de ocorrido, culpar os requisitos fornecidos não irá ajudar, portanto, certifique-se de que você ou um membro de sua equipe esteja envolvido na documentação deles. Se isso não for possível, então, estabeleça um tempo maior a seu projeto para revisar a análise dos requisitos existentes e para examinar qualquer parte obscura pesquisando diretamente a fonte (o pessoal operacional por trás do projeto, os usuários ou mesmo um sistema anterior que executasse as mesmas ações).

Plataforma Cliente

Como parte de seu planejamento, você precisa saber alguns detalhes sobre a plataforma em que seu sistema será executado:

- Que sistemas operacionais estão instalados nos sistemas de destino?
- Quais são as especificações de hardware das máquinas de destino (CPU/RAM/espço em disco)?
- Qual o nível de controle que a empresa tem sobre as máquinas de destino? Ela controla a configuração, a instalação do software e a proteção contra vírus? Soluções extremas para essas questões seriam um servidor só para terminais burros em uma extremidade e computadores públicos de propriedade dos usuários na outra.
- Algum dos computadores clientes é um laptop? O acesso off-line ou remoto é necessário? Serão precisos serviços de discagem ou eles já existem?

Rede

Muitas opções relacionadas à arquitetura, como o tipo de cliente, a replicação de bancos de dados e o protocolo usado dependem da rede em que o sistema será executado:

- Qual a velocidade de comunicação disponível entre os usuários e os servidores de back-end (ou o que estará disponível, se os servidores de back-end ainda não existirem)?
- Os serviços de acesso remoto serão necessários e/ou fornecidos? VPN? Dial-up?
- Os computadores-cliente têm acesso à Internet?
- Há algum software/hardware de firewall entre os usuários e os servidores de back-end?
- Que tipo (TCP/IP, IPX, NetBEUI e outros) de rede está em uso?

Segurança

Todo sistema precisa ter pelo menos algumas especificações de segurança; mesmo um aplicativo não protegido deve possuir uma declaração de que foi projetado dessa maneira. Uma informação essencial que precisa ser determinada, mas que não é fácil de converter em uma simples pergunta, é qual o nível de importância que será dado à segurança do aplicativo e seus dados? Tanto um sistema financeiro quanto um que registre a pontuação em jogos de golfe consideram a segurança, mas o grau de preocupação provavelmente será um pouco mais alto em um deles. Além dessa parcela da informação, as perguntas a seguir serão pontos de partida úteis na determinação das necessidades de segurança do aplicativo:

- Como os usuários serão autenticados na rede existente?
- As máquinas clientes pertencem a um domínio do Windows 2000/NT?
- Se público, o acesso será anônimo ou os usuários terão de se identificar?
- Esse aplicativo será responsável por alterações na senha ou pelo gerenciamento de outros recursos de segurança?
- Os usuários terão diferentes níveis de acesso, possivelmente com restrições para os recursos do aplicativo que poderão usar?

Outras Considerações

Outros fatores essenciais no desenvolvimento de uma arquitetura serão listados aqui, porém, é nessas áreas que o nível de experiência do desenvolvedor tem a maior importância. Com relação a esses pontos, não há respostas definitivas. É mais uma questão de conseguir informações suficientes a fim de incluí-las no planejamento da arquitetura.

- Velocidade das alterações na lógica operacional – Se a lógica operacional que conduz o sistema se alterar rapidamente, isso afetará a frequência com que o cliente em um sistema cliente/servidor teria de ser atualizado. Alterações maiores na lógica operacional decerto precisariam até mesmo de uma atualização também do cliente.
- Conjunto de habilidades da equipe de desenvolvimento – Algumas formas de desenvolvimento são mais complexas e, portanto, mais difíceis do que outras. Na maioria dos casos,

um sistema cliente/servidor é mais simples de escrever do que um completo com três camadas. Quando escolher uma arquitetura, você poderá tentar desenvolvê-la de acordo com o nível de habilidade da equipe ou basear a aptidão necessária na arquitetura selecionada.

- **Requisitos/planos futuros** – Ao desenvolver a arquitetura para um sistema, você terá de enxergar além dos requisitos atuais e considerar os de amanhã. No mínimo, mesmo se não tiver informações sobre quais serão os planos futuros, desenvolva um sistema que possa ser aprimorado. Se já souber os detalhes específicos, documente-os em seus requisitos e projete de modo apropriado. Como exemplo desse tipo de informações, imagine um sistema planejado para ser usado por duas pessoas, mas que será aberto para a comunidade da Web em alguns meses.

Exemplos de Cenários

As discussões anteriores sobre os fatores essenciais se concentraram no que você precisa apreender durante a fase de análise dos requisitos, mas isso na verdade envolve examinar várias descrições com estilo de narrativa e tentar decifrar as informações que elas contêm. Nesta seção, mostrarei alguns exemplos breves de requisitos de sistema e as informações sobre a arquitetura que podem ser obtidas por meio deles.

Sistema de uma Videolocadora

Narrativa dos Requisitos:

“Uma cadeia de videolocadoras precisa de um novo sistema para rastrear as fitas, os clientes e as locações. Todos os locais atualmente estão conectados ao escritório central por meio de modems. Cada loja possui um ou mais computadores agindo como terminais de ponto-de-venda. Os computadores são novos, com placas Pentium II. Os dados sobre as Vendas/Clientes/Vídeos de cada loja não precisam estar disponíveis no escritório central em tempo real; o sistema atual executa à noite alguns comandos em lote que transmitem os dados para o escritório central, que em geral são suficientes para os propósitos da loja.”

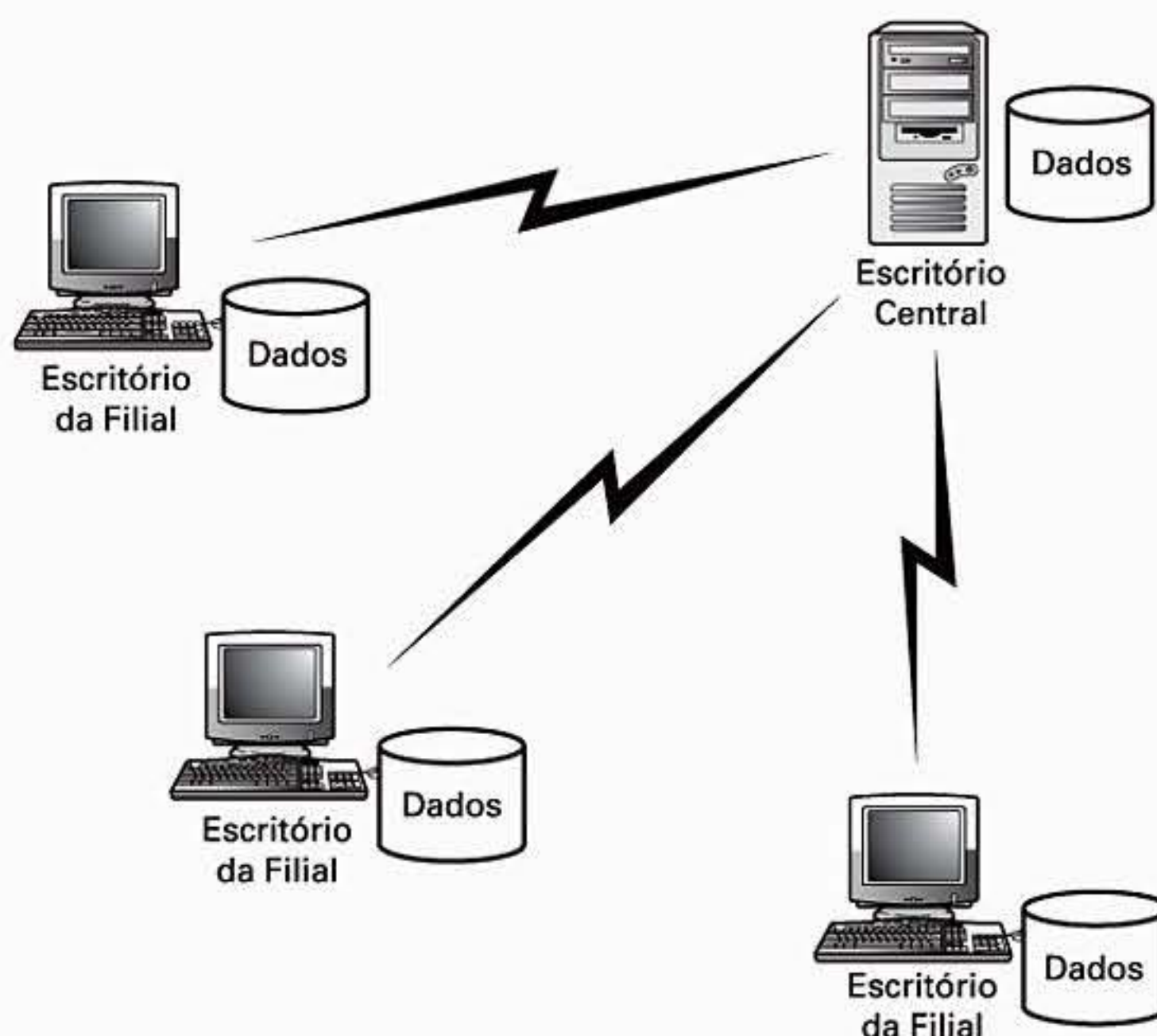
Discussão

Esse é um cenário bem comum – múltiplos locais ou filiais executando a entrada de dados, e um escritório central que deseja ser atualizado regularmente com os dados desses locais (veja a Figura 5.2). As máquinas no local cliente parecem (de acordo com essa breve discussão) ser de um padrão comum e bastante novas de modo que um aplicativo do Windows Forms é uma possibilidade, e um do ASP.NET sempre será uma opção. Nesse caso, o fator decisivo terminará sendo a declaração que diz que a atualização noturna dos dados atenderá. Com esse comentário e uma conexão relativamente lenta retornando para o escritório central, eu consideraria um sistema em que os dados ficariam armazenados nesse escritório e seriam replicados para cada local durante períodos lentos ou à noite, depois que cada loja fechasse.

Os sistemas para cada loja seriam aplicativos do Windows Forms, eliminando, assim, a necessidade de um servidor Web. Um banco de dados local de algum tipo (MSDE ou Access) seria instalado em cada loja, e uma carga programada desses dados seria enviada para o escritório central toda noite. Se, em vez disso, fosse usado um banco de dados central, com todos os escritórios executando o seu trabalho através de uma conexão dial-up, proporcionaríamos a vantagem de ter os dados de todas as lojas disponíveis e atualizados no escritório central, mas o desempenho de uma conexão via modem provavelmente seria um problema. Um front-end com base na Web que desse suporte a funções que uma videolocadora em geral precisa (seria necessário pesquisar-mos mais para determinar isso), como a impressão de recibos e talvez cartões de associados, seria difícil de criar.

FIGURA 5.2

A videolocadora possui uma arquitetura comum, com muitas filiais se conectando a um escritório central. A decisão principal é se o cliente deve se conectar diretamente ao escritório central ou usar algum tipo de depósito de dados local.



Acesso Público

Considere o que seria diferente se a linha a seguir fosse incluída nos requisitos da videolocadora: “Por fim, esperamos criar um site da Web público para nossos clientes, no qual eles poderão procurar filmes e verificar se um certo filme existe em estoque em uma loja específica.”

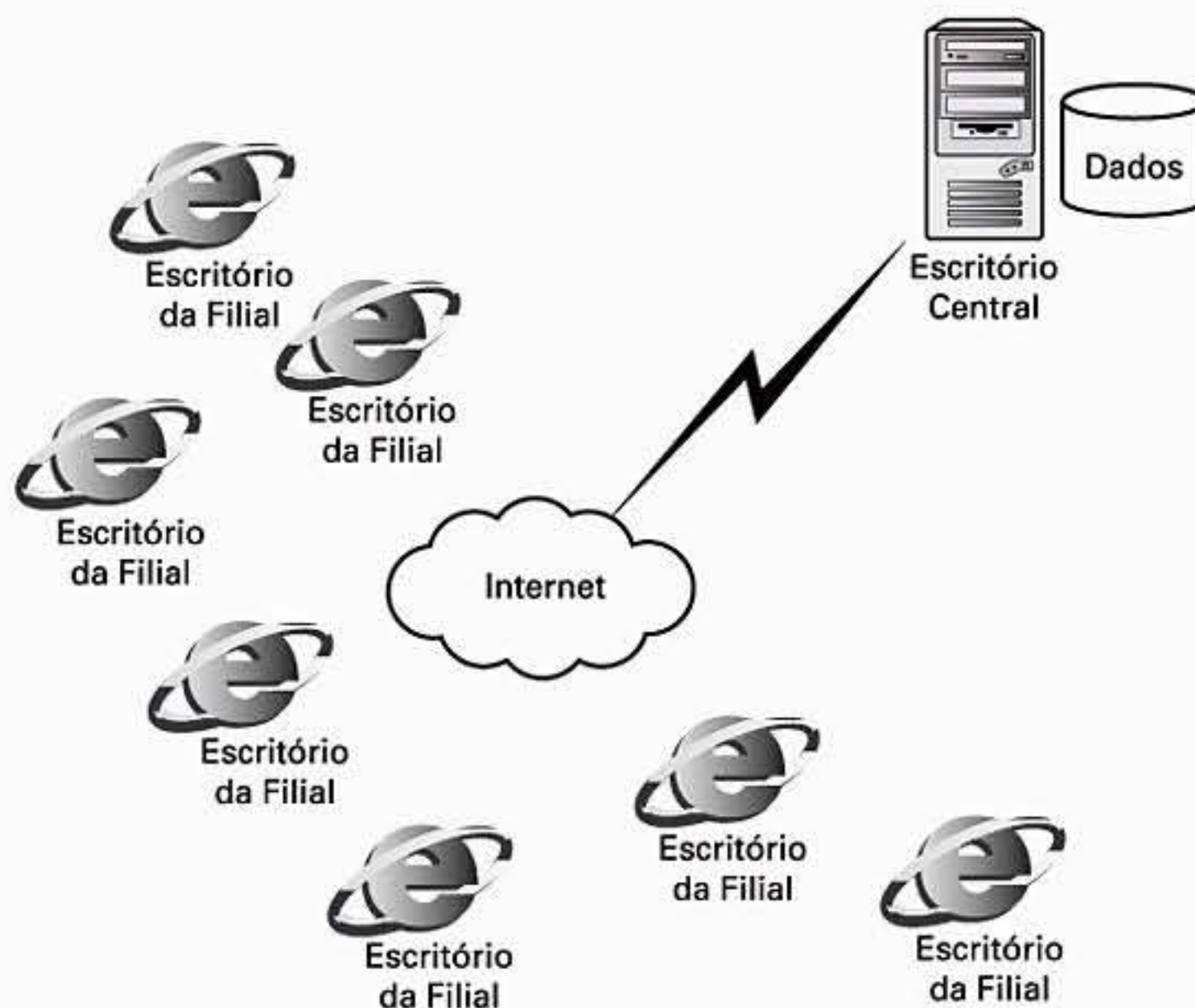
Essa declaração provoca mais alterações do que você pode imaginar. Como é um ‘requisito futuro’, poderíamos apenas ignorá-lo e projetar o sistema com base no resto da informação. Isso produziria um sistema que teria de ser redefinido logo que a empresa passasse para a próxima etapa de seus planos. Em vez disso, é preferível tentar desenvolver uma arquitetura que aborde os planos atuais e possa manipular os requisitos futuros.

Sem entrarmos em muitos detalhes, porque você com certeza iria precisar coletar mais informações sobre os planos futuros a fim de criar uma arquitetura adequada para o site público da Web, considere as alterações básicas que seriam necessárias (veja a Figura 5.3). O comentário sobre “alguns comandos em lote transmitem os dados para o escritório central toda noite” de certo já é incorreto. Se o site da Web disponibilizasse informações sobre os filmes, mas estivesse atrasado em um dia, não seria tão útil! Não é necessário obter as informações em tempo real, mas um atraso aceitável teria de ser determinado (talvez 15 minutos).

Se o tempo de espera planejado não for tão curto, então, talvez o sistema também possa ser projetado com um armazenamento de dados local e atualizações em lote. Para um sistema de computador, 15 minutos é muito tempo, e seria preciso uma largura de banda muito menor do que em uma conexão contínua. Com esse novo requisito, um aplicativo da Web parece mais adequado porque todos os dados seriam armazenados no escritório central e alguns códigos poderiam ser compartilhados entre os dois sistemas. Mas um site da Web não teria um nível de resposta tão bom quanto um aplicativo do Windows Forms com o armazenamento de dados no local da loja.

FIGURA 5.3

A decisão de ter um site público da Web alterou a arquitetura da videolocadora para assegurar que os dados do escritório central fiquem atualizados.



Serviço de Listagens Imobiliárias

Narrativa dos Requisitos:

“Uma grande empresa imobiliária, de abrangência nacional, deseja informatizar suas listagens de imóveis. Os corretores adicionariam as listagens diretamente a um sistema de computador, e um site público da Web seria disponibilizado para os clientes pesquisarem as que estivessem disponíveis e enviarem mensagens para os corretores apropriados. Esses precisariam do serviço de listagens, assim como de um recurso para inserir listagens novas, e acessariam os dois de seus

laptops. Já que esses corretores em geral estão viajando durante todo o dia, eles usam seus laptops remotamente a maior parte do tempo.”

Discussão:

Dada a descrição anterior, você poderá extrair alguns elementos essenciais logo de imediato. Haverá um site da Web, que já está definido, pelo menos com relação à interface com o público. A questão é “como manipular a interação com os corretores”. Eles têm de poder acessar e ainda adicionar, atualizar e excluir as listagens que inseriram no serviço, tudo enquanto viajam o dia inteiro. Poderíamos sincronizá-los com o sistema quando estivessem no escritório, fazer o download das listagens em suas máquinas e carregar qualquer listagem nova que tivessem acrescentado. É provável, no entanto, que eles só passem pelo escritório ocasionalmente e, portanto, uma nova listagem não chegaria a eles ou suas listagens novas não seriam carregadas por cerca de um dia. Isso é aceitável? Talvez, precisamos de mais informações.

De modo alternativo, os corretores poderiam usar modems sem fio (com a tecnologia dos celulares, como o CDMA), que poderiam conectá-los ao escritório central ou à Internet de qualquer local. Por meio dessa opção, você agora teria duas escolhas: continuar utilizando o aplicativo off-line, porém executando sua sincronização através de uma conexão sem fio (resultando em atualizações mais frequentes) ou desenvolver o recurso adicionar/editar/excluir em uma parte segura do site público da Web da agência imobiliária, <http://www.nomedosite.com.br/agentes>, que os agentes acessariam com um navegador comum pelo modem celular. Devido à baixa velocidade (atual) desses modems celulares, gostaria de considerar as duas opções e executar alguns testes, mas o site público poderia ser iniciado sem espera.

Como você pode ver, a arquitetura do aplicativo não é um conceito definitivo, principalmente quando se lida só com uma quantidade limitada de informações sobre a situação. Em geral, é preciso solicitar todos os requisitos possíveis, analisá-los e, em seguida, voltar aos diversos entrevistados e pedir dados adicionais. Esse processo pode ser apenas para esclarecer as perguntas originais, ou talvez novas perguntas surjam das respostas fornecidas.

Biblioteca Pessoal com Informações sobre CDs

Narrativa dos Requisitos

“Como projeto prático, você irá desenvolver uma biblioteca pessoal com informações sobre CDs que os usuários poderão instalar em suas máquinas para registrar suas coleções. Ela dará suporte à pesquisa por nomes de CDs e ao rastreamento de títulos na Internet. Todos os detalhes sobre a coleção de CDs do usuário serão armazenados, e o sistema permitirá a geração de alguns relatórios simples que usem esses dados.”

Discussão:

Esse sistema é bem simples, e a arquitetura para desenvolvê-lo também deve ser. O único item realmente desconhecido nele é a pesquisa de detalhes sobre os CDs na Internet, mas essa não é

uma questão relacionada à arquitetura. Um aplicativo do Windows Forms seria o mais adequado, porém você deve escolher um banco de dados que não seja o SQL Server porque é preferível fazer a distribuição sem precisar de nenhum software adicional nas máquinas dos usuários. Um arquivo do Access (.mdb) ou um banco de dados MSDE atenderiam satisfatoriamente a esse projeto. Como alternativa, todas as informações poderiam ser armazenadas apenas como XML em um conjunto de dados (DataSet).

Resumo

O universo da plataforma .NET é novo e poderoso, mas tudo precisa de uma base. O projeto e a arquitetura de seus sistemas são críticos para a obtenção de êxito. Apesar dos novos recursos e alterações radicais na tecnologia subjacente, a arquitetura de um sistema .NET não é tão diferente do que era para um sistema do Visual Basic 6.0; muitas opções ainda são as mesmas. Certifique-se sempre de basear suas escolhas nos requisitos que obtiver e não esqueça de planejar para o futuro!

Como foi mencionado, muitos aspectos da arquitetura dos sistemas estão além do escopo desta lição, mas o MSDN dedicou uma seção de seu site da Web para a arquitetura dos aplicativos .NET. Acesse <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daag.asp> para encontrar muitos recursos úteis.

P&R

P Ouvi dizer que o Windows DNA foi abandonado e não é mais aplicado agora que a plataforma .NET existe. Você afirmou que ele ainda é empregado, mas, na verdade, não é um recurso ultrapassado?

R O Windows DNA representa tanto alguns conceitos quanto tecnologias. Algumas das tecnologias a ele incorporadas, como o Visual Studio 6.0, foram substituídas pelas equivalentes da plataforma .NET, mas o conceito do Windows DNA (computação distribuída em várias camadas, separando as de apresentação, operacional e de dados) ainda é válido. Os sistemas que já foram desenvolvidos com o modelo do Windows DNA serão fáceis de transferir para o universo da plataforma .NET.

P No universo da plataforma .NET, todos os meus componentes se comunicarão uns com os outros por meio do SOAP (XML acima do HTTP), em vez de com o velho padrão binário do DCOM?

R Você está correto com relação ao fato de que o DCOM não é mais o método preferido, mas a plataforma .NET fornece mais do que apenas o SOAP como um meio para dois aplicativos se comunicarem. O SOAP é excelente para a comunicação com a plataforma .NET ou sistemas que não sejam .NET e é especialmente bom quando se trabalha entre

duas redes porque ele passa com facilidade através de firewalls. O SOAP é uma opção, mas há várias outras formas de comunicação incluindo os formatos binários.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Nomeie uma vantagem relevante em usar uma interface com usuário com base na Web em vez de um aplicativo Windows.
2. Se um aplicativo usar um arquivo do banco de dados Access localizado em um servidor, ele será considerado um sistema cliente/servidor ou de uma única camada?
3. O que tem de ser instalado para que um computador execute um aplicativo criado com a plataforma .NET (não é só procurar um site da Web criado com ela)?

SEMANA 1

DIA 6

O Que Fazer Quando Programas Bons Apresentam Problemas e para Se Certificar de Que Isso Não Aconteça

Pode acontecer um erro quando seu programa estiver sendo executado (normalmente causando a interrupção dele). Se ocorrer no momento em que um cliente estiver processando seu programa, algo bem pior poderá acontecer (você pode não ser pago). Portanto, seria melhor se pudéssemos evitar esses problemas antes que acontecessem.

Há duas técnicas importantes que o ajudarão a evitar erros. Primeiro, você deve detectar quando as falhas ocorrerem em seus programas e lidar com elas em vez de permitir que a mensagem de erro padrão surja como resposta. Em segundo lugar, é bom se certificar de que seu aplicativo tenha a menor quantidade de erros possível.

Hoje, você aprenderá as duas principais maneiras de evitar problemas em seu código:

- Tratamento de exceções estruturadas
- Depuração

Tratamento de Exceções Estruturadas

As versões mais antigas do Visual Basic usavam o tratamento de erros para proteger um aplicativo. O Visual Basic .NET é a primeira versão a dar suporte a um conceito melhor em linguagens de programação – o tratamento de exceções estruturadas.

O Que É o Tratamento de Exceções Estruturadas?

NOVO TERMO

Antes que você possa examinar o que é Structured Exception Handling (SEH, tratamento de exceções estruturadas), deve aprender apenas o que significa uma *exceção estruturada* – ou até uma *exceção não estruturada*. Caso contrário, não saberá o que está tratando. Uma *exceção* é algo que sai da rotina (e em geral de maneira inesperada) em seus aplicativos. Há dois tipos de exceções – exceções de hardware e de software. É claro que uma exceção de hardware é aquela causada pelo hardware – por exemplo, quando seu programa tenta acessar uma parte da memória que não deveria ou um evento semelhante que ocorra com base no hardware. Uma exceção de software ocorre quando se tenta atribuir um valor incompatível a uma variável ou um erro parecido acontece.

O *tratamento de exceções estruturadas* é uma estratégia para lidar com erros de hardware ou software. Tratando os dois tipos de exceção da mesma maneira, o código necessário para processá-las será mais fácil de escrever e mais consistente. Execute o tratamento de exceções estruturadas para ‘proteger’ as seções de código que estiverem propensas a serem alvos de exceções. Por exemplo, quando estiver para efetuar uma divisão, poderá proteger o código que a calcula. Se – por causa de um erro que você cometeu no programa ou um valor que o usuário inseriu – o programa tentar fazer a divisão por zero, o código protegido poderia lidar com a exceção resultante. Outras vezes em que se deve usar o SEH são na leitura em bancos de dados, abertura, leitura ou gravação em arquivos, ou qualquer outro momento em que você ache que um erro possa ocorrer.



NOTA

Sou daqueles que crê ser difícil usar em demasia o SEH. Os benefícios que pode obter protegendo a si mesmo, seus programas e usuários de erros compensam em muito uma pequena redução no desempenho que resulte da utilização do SEH. Por outro lado, seja sensato. Se não houver nenhuma possibilidade (e quero dizer, *realmente* não existir a possibilidade) de um erro ocorrer, deixe de lado a digitação adicional.

Erros e Exceções

Bem, em que um erro difere de uma exceção? Se você estiver familiarizado com outras versões do Visual Basic ou mesmo com a maioria das linguagens de programação, decerto se deparou com o conceito de erro, mas não de exceção. Qual é a diferença? Por que precisamos de uma palavra nova e de algo que soa tão extravagante como o tratamento de exceções estruturadas?

Primeiro, uma exceção na verdade é um erro. Parte do problema, no entanto, é que um erro significa muito mais. Um exemplo de erro poderia ser uma função que retornasse um resultado que não fosse esperado ou se o programador usasse o algoritmo errado. Uma exceção é mais formal. A exceção ocorre, para repetir a definição anterior, quando algo acontece em seu código fora do fluxo normal de controle. O SEH define uma maneira particular de lidar com exceções, assegurando que tudo seja normalizado depois do fato. Além disso, um outro recurso adequado das exceções é que hoje elas são compartilhadas entre todas as linguagens que tenham suporte da plataforma .NET. Isso significa que seria possível ter um código que causasse uma exceção em uma linguagem, mas a tratasse em outra.

A maioria dos erros no Visual Basic tem sido tradicionalmente tratados com o uso da instrução `On Error`. Essa instrução *não é estruturada* e pode resultar em um código confuso que salte de um local para outro de um procedimento. As exceções são tratadas com a utilização de uma abordagem mais estruturada, ou organizada, como você verá.

O Bloco Try

Você deve proteger a seção dos códigos com o bloco `Try...End Try` a fim de tratar as exceções sempre que ocorrerem. Inicie o código que deseja proteger com `Try` e finalize a seção com `End Try`.

A Sintaxe de Try...End Try

SINTAXE

O código a seguir mostra a sintaxe do bloco `Try...End Try`:

```
Try
.
Catch Ex As Exception
.
End Try
```

onde `Try` é o início da seção do código a ser protegido e `End Try` finaliza o bloco de código. A linha `Catch` marca a seção do bloco que realmente trata as exceções que possam ocorrer.

A Seção Catch

Marcar apenas uma seção de código com `Try...End Try` na verdade não produz nenhum resultado. Se uma exceção ocorrer no código protegido, ela deve ser ‘capturada’ e tratada. Capture a exceção com a palavra-chave `Catch`.

Cada bloco `Try...End Try` pode ter uma ou mais seções `Catch`. Cada seção `Catch` em geral se destina a um ou mais tipos de exceção. Se decidir ter várias seções `Catch` em seus blocos `Try`, cada seção deve capturar tipos diferentes de exceção. Você pode querer fazer isso para permitir que cada tipo de exceção seja tratado de modo diferente. Se resolver inserir todas essas seções `Catch`, também é recomendável ter uma seção `Catch` genérica que capturará todas as exceções

que possam passar pelas outras seções. Essa forma do bloco Try...End Try teria uma aparência semelhante à mostrada na Listagem 6.1.

CÓDIGO**LISTAGEM 6.1** Capturando Várias Exceções

```
1 Try
2 '
3 Catch eNoFile As FileNotFoundException
4 ' trate aqui a exceção causada por arquivos não encontrados
5 Catch eIO As IOException
6 ' trate aqui a exceção causada por uma entrada/resultados
7 Catch Ex As Exception
8 ' trate aqui as outras exceções
9 End Try
```

ANÁLISE

Esse bloco Try foi projetado com a finalidade de capturar erros em um procedimento que processa arquivos. Há seções separadas para tratar exceções por arquivos não encontrados (FileNotFoundExceptions), de E/S (IOExceptions) e outras. Se existisse apenas uma seção Catch, você precisaria de algum código que determinasse a exceção específica. Dessa maneira, fica mais evidente que código é usado para cada tipo de exceção.

Criaremos um programa com uma exceção intencional, de modo que você possa ver como ela funciona. Examinaremos seus resultados e o que ela faz ao programa. Em seguida, adicionaremos o tratamento da exceção, e saberemos como ele pode proteger o código.

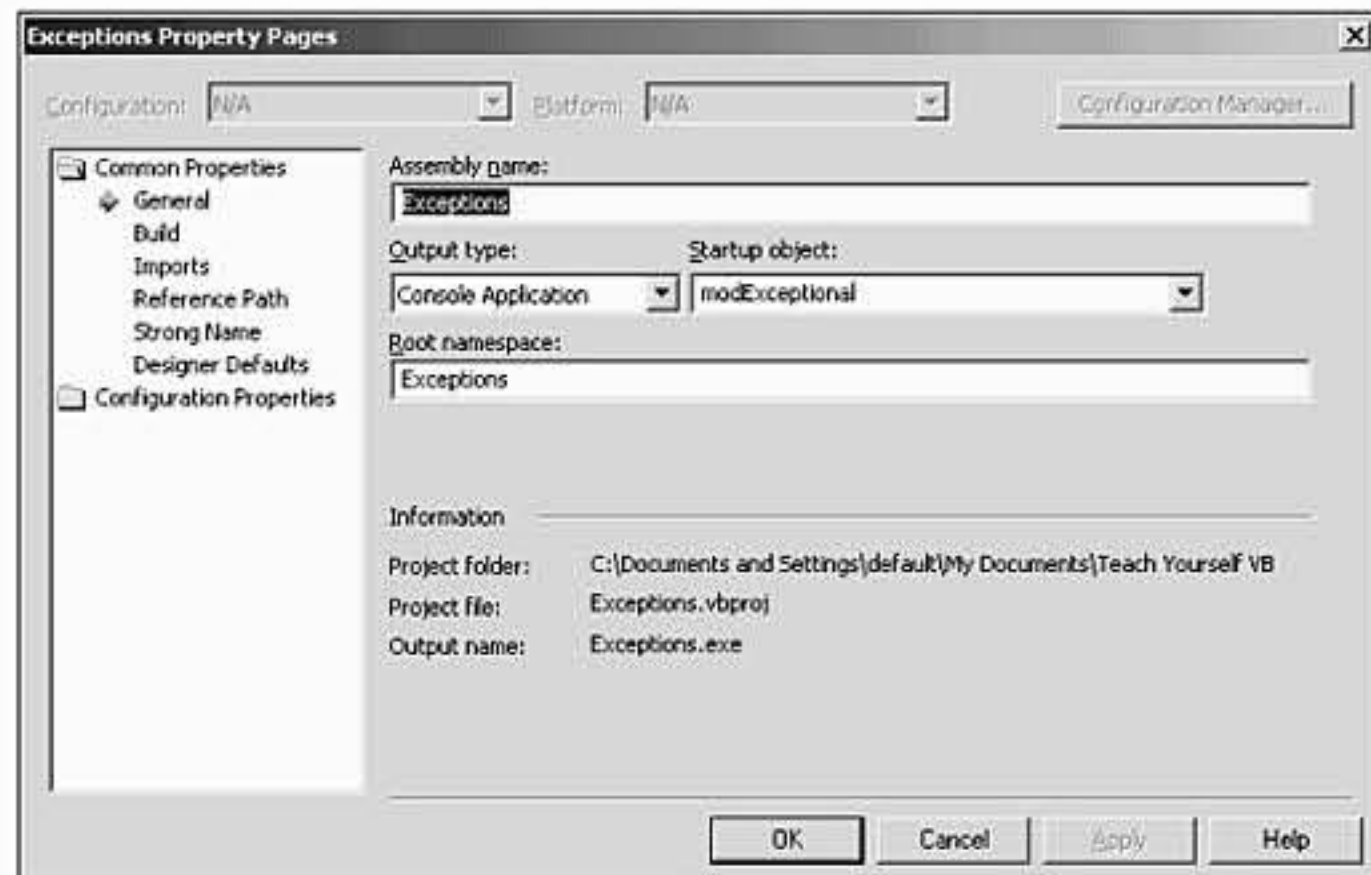
1. Crie um novo aplicativo do console. Chame-o de Exceptions.
2. Altere o nome do módulo criado para **modExceptional**.
3. Acesse a caixa de diálogo Project Properties dando um clique com o botão direito do mouse no nome do projeto do Solution Explorer e selecionado Properties.
4. Configure o Startup Object como **modExceptional**. As configurações finais devem ficar como as que são mostradas na Figura 6.1.

Para atender a finalidade deste exemplo, você usará uma exceção simples e surpreendentemente comum, Division By Zero, que ocorre quando há uma divisão por zero. Esse erro pode ser demonstrado com facilidade dividindo-se um número por uma variável não inicializada. (É uma das causas mais comuns dessa exceção).

A Listagem 6.2 mostra que aparência o código final deverá ter:

FIGURA 6.1

*A caixa de diálogo
Project Properties
para um programa
com uma exceção.*



LISTAGEM 6.2 Um Programa com Exceção

```

1 Module modExceptional
2
3     Sub Main()
4         '***Este código contém uma exceção,
5         'e portanto não será executado até o final
6         Dim dDividend As Decimal = 5
7         Dim dDivisor As Decimal = 0
8         Dim dResult As Decimal = 0
9         'Essa linha é a que causará
10        ' uma Exceção DivisionByZero
11        dResult = dDividend / dDivisor
12        System.Console.ReadLine()
13    End Sub
14
15 End Module

```

ANÁLISE

As linhas 6, 7 e 8 declaram suas variáveis. Você usou decimais aqui, mas também poderia empregar outros tipos de variável numérica. Os valores também foram inicializados.

Antes da divisão na linha 11, o valor com o qual o divisor foi inicializado, 0, não foi alterado. Portanto, quando o computador tentar efetuá-la, uma exceção ocorrerá. Se você executar esse programa no IDE, verá um erro como o mostrado na Figura 6.2.

Se você selecionar Break, o programa ainda será executado, mas no modo Debug (depuração). Examinaremos melhor o modo Debug ainda nesta lição, portanto selecione Continue neste momento. Isso fará com que o programa continue a ser executado. Neste caso, ele será finalizado.

FIGURA 6.2

Mensagem-padrão de exceção.



Se você reproduzir o exemplo e tentar executá-lo em uma janela do prompt de comando, o resultado final será semelhante, porém sua correção será mais difícil. Primeiro, em um computador de desenvolvimento (que tenha o Visual Basic .NET instalado), deve aparecer a caixa de diálogo Just-In-Time Debug. Selecione No; seu programa continuará, e este resultado aparecerá:

RESULTADO

```
1 Unhandled Exception: System.DivideByZeroException:
2 An exception of type System.DivideByZeroException was thrown.
3   at System.Decimal.Divide(Decimal d1, Decimal d2)
4   at Exceptions.modExceptional.Main() in
5 C:\Code\Day06\Exceptions\modExceptional.vb:line 11
```

ANÁLISE

Essa é uma mensagem-padrão que você verá quando uma exceção ocorrer. Várias mensagens de erro relacionadas a exceções são muito mais longas, no entanto, todas contêm informações semelhantes. A primeira linha sempre identifica o tipo de exceção que acabou de ocorrer. Neste caso, descobriremos que foi uma do tipo `System.DivideByZeroException`. O tipo de exceção em geral é uma grande indicação da natureza do erro e pode até proporcionar idéias de como corrigi-la. Aqui, o tipo de exceção `System.DivideByZeroException` informa principalmente dois detalhes:

- A exceção que acabou de ocorrer é uma das ‘comuns’, isto é, que faz parte do espaço de nome `System` (e não de outro espaço de nome, como `System.IO.FileNotFoundException`).
- A exceção que acabou de ocorrer envolve uma operação de divisão. Portanto, você deve procurar uma possível causa em seu código, onde estiver efetuando uma divisão.

De maneira semelhante, o restante da exceção disponibiliza muitas informações. As outras linhas são uma lista invertida dos procedimentos que estavam efetivamente em execução quando a exceção ocorreu. Na saída do exemplo, vemos que na verdade ela surgiu no procedimento `System.Decimal.Divide`. Esse procedimento foi chamado por `Exceptions.modExceptional.Main`.

A principal função de todas essas informações é localizar candidatos ao erro. Ao encontrar uma exceção, você deve primeiro examinar o código de cada um desses procedimentos. Em geral (mas nem sempre), o erro se encontrará em um deles. Comece com o primeiro item da lista (nesse caso, `modExceptional.Main`) e percorra-a até o final. Mais adiante, nesta lição, examinaremos algumas ferramentas que tornarão esse trabalho mais fácil.

Algumas outras exceções comuns estão descritas na Tabela 6.1.



Esta não é uma lista completa de todas as exceções possíveis. Há muitas outras; por favor consulte a ajuda do Visual Basic .NET para obter mais detalhes. Além disso, você pode criar suas próprias exceções para adicionar a essa lista se precisar, como veremos ainda nesta lição.

TABELA 6.1 Exceções Comuns

Tipo de Exceção	Quando Ocorre?
ArgumentException	Categoria geral para erros que ocorram quando o tipo (ou valor) errado for passado para um método. Inclui ArgumentNullException e ArgumentOutOfRangeException. Pode surgir por causa de um erro do programador ou nas entradas de dados do usuário.
ArgumentNullException	Ocorre quando você passa um valor nulo para um método, e ele não o aceita.
ArgumentOutOfRangeException	Ocorre quando é passada uma variável que é grande ou pequena demais para o método, por exemplo, se você passar o número -1 a um método que foi criado para conter um valor relativo ao mês (isto é, entre 1 e 12).
DivideByZeroException	Ocorre quando se tenta efetuar uma divisão por uma variável não inicializada ou que contenha o valor zero. Em geral só acontece quando há erro do programador.
IndexOutOfRangeException	Ocorre quando se tenta acessar o membro de um array que não existe. Em geral acontece por causa de erro do programador, mas também poderia ser causada por entradas inválidas do usuário.
NotImplementedException	Geralmente usada como um espaço reservado quando o desenvolvedor está trabalhando pela primeira vez no programa. Você pode criar o shell para seu aplicativo e, em seguida, lançar essa exceção de qualquer método. Enquanto continuar no aplicativo, substitua a exceção pelo código efetivo. Isso assegurará que conclua todos os seus procedimentos.
OutOfMemoryException	Ocorre quando seu programa não tem memória suficiente. Isso poderá acontecer ao preencher arrays extensos ou ao executar um laço.
OverflowException	Uma exceção bem comum que ocorre quando você tenta inserir um valor que é muito grande em uma variável, como, por exemplo, na atribuição a seguir: <code>Dim iSmallishNumber As Short = 50000</code>
FileNotFoundException	O exemplo de um erro que não é definido no espaço de nome System. Nesse caso, a exceção é definida no espaço de nome System.IO (veja o Dia 8, "Introdução ao .NET Framework", para obter mais informações sobre os espaços de nome). Esta exceção ocorre quando você tenta acessar um arquivo que não existe. Isso pode acontecer porque talvez ele não tenha sido criado ou o caminho está incorreto.

Agora que você já viu a exceção e tem (espero) uma idéia melhor do que seja, deve adicionar o SEH a seu programa para fornecer ao usuário mais informações quando um erro ocorrer.

Abra o projeto Exceptions no Visual Basic .NET. Você copiará o projeto anterior e fará alterações para capturar a exceção, como mostram as etapas a seguir:

1. Altere o nome do projeto para **Exceptions2**. Faça isso selecionando o projeto no Solution Explorer e, em seguida, selecione File, Save Exception As no menu.
2. Altere o nome do arquivo que contém o código para **Exceptions2.vb**. Faça isso selecionando o arquivo no Solution Explorer e, em seguida, selecione File, Save Exception.vb As.
3. Adicione um bloco Try...End Try ao código que efetua o cálculo. Use uma instrução Catch genérica para capturar a exceção Division By Zero.
4. Adicione um código para exibir uma mensagem mais amigável quando a exceção ocorrer.

O novo código deve ficar com uma aparência semelhante ao da Listagem 6.3.

CÓDIGO**LISTAGEM 6.3** Adicionando o Bloco Try...End Try

```
1 Module modExceptional
2
3     Sub Main()
4         '***Este código contém uma exceção,
5         'e portanto não será executado até o final
6         Dim dDividend As Decimal = 5
7         Dim dDivisor As Decimal = 0
8         Dim dResult As Decimal = 0
9         'Esta linha é a que causará
10        'uma exceção DivisionByZero
11        Try
12            dResult = dDividend / dDivisor
13        Catch Ex As Exception
14            System.Console.WriteLine("Ocorreu um erro de divisão por zero.")
15            System.Console.WriteLine("Verifique o divisor.")
16        End Try
17
18        System.Console.ReadLine()
19    End Sub
20
21 End Module
```

ANÁLISE

A linha 11 inicia o bloco Try, portanto, o código desse local até o primeiro bloco Catch da linha 13 está protegido. Nesse caso, a única linha executável é a que causa a exceção da divisão por zero na linha 12. Quando essa exceção ocorrer, o programa será interrompido e procurará algo para tratar dela. Já que a seção Catch da linha 13 é genérica, capturará qualquer tipo de exceção. A variável Ex será usada para armazenar a exceção criada, e você poderá empregar as propriedades dessa variável para visualizar mais informações sobre a exceção. Em

seguida, será exibida uma mensagem de erro, e o código continuará até o final, começando na linha posterior a instrução `End Try`.

Aninhando Blocos `Try...End Try`

Você pode encontrar uma situação na qual queira proteger duas seções de código com blocos `Try...End Try`, mas deseja lidar com as exceções de maneira diferente. Se os dois forem blocos separados de código, não haverá problema. No entanto, se um dos blocos estiver contido no outro (veja a Listagem 6.4), não há opção. Nesse caso, é preciso aninhar os blocos `Try...End Try`, como mostra a Listagem 6.4.

CÓDIGO

LISTAGEM 6.4 Aninhando Blocos `Try...End Try`

```
1 Sub WriteToFile(ByVal FileName As String)
2     Dim fsOut As System.IO.FileStream
3     Dim strOut As System.IO.StreamWriter
4     Try
5         'Abra o arquivo
6         fsOut = _
7             New System.IO.FileStream(FileName, _
8                 System.IO.FileMode.OpenOrCreate, _
9                 System.IO.FileAccess.Write)
10    Try
11        'Grave no arquivo
12        strOut = _
13            New System.IO.StreamWriter(fsOut)
14        strOut.Write(DateTime.Today.ToString())
15    Catch eIO As Exception
16        Console.WriteLine("Não foi possível gravar no arquivo: {0}.",
17            FileName)
18    End Try
19    Catch eFile As Exception
20        Console.WriteLine("Não foi possível abrir o arquivo: {0}.", FileName)
21    End Try
22 End Sub
```

ANÁLISE

A Listagem 6.4 é um exemplo de gravação em um arquivo. Você examinará esse assunto com mais detalhes no Dia 8. Porém, há dois blocos `Try...End Try` neste exemplo. Um dos blocos `Try` (o que começa na linha 10) está todo contido dentro do outro bloco `Try` (o que começa na linha 4). Portanto, o bloco `Try` interno está *aninhado* no outro bloco. Se uma exceção ocorrer na gravação que acontece na linha 14, a instrução `Catch` da linha 15 a capturará, e o usuário verá "Não foi possível gravar no arquivo SomeFile.out". Além disso, se o arquivo não puder ser aberto, o bloco `Catch` da linha 18 capturará a exceção, exibindo "Não foi possível abrir o arquivo: SomeFile.out".

Exatamente como com `If...End If` e outros blocos, não há limite para a maneira de aninhá-los. Às vezes, aninhar blocos permite que a escrita do código seja feita de uma maneira mais organizada do que se eles não fossem aninhados.

A Seção Finally

Quando você escrever blocos `Try`, às vezes se deparará com situações nas quais, mesmo se uma exceção ocorrer, *terá de* executar algo. Por exemplo, se você escrever um código que grava informações em um arquivo, deve fechá-lo, ocorrendo ou não um erro durante a gravação. Adicione essa funcionalidade com a seção `Finally`. Essa seção surge depois de todas as seções `Catch`, e deve conter apenas o código que sempre terá de ser executado. Esse é um bom local para fechar arquivos, configurar variáveis com `Nothing` ou excluí-las quando desejar. A Listagem 6.5 mostra a inserção de uma seção `Finally` no código da Listagem 6.4.

CÓDIGO

LISTAGEM 6.5 Usando a Seção Finally

```
1 Sub WriteToFile(ByVal FileName As String)
2     Dim fsOut As System.IO.FileStream
3     Dim strOut As System.IO.StreamWriter
4     Try
5         'Abra o arquivo
6         fsOut = _
7             New System.IO.FileStream(FileName, _
8                 System.IO.FileMode.OpenOrCreate, _
9                 System.IO.FileAccess.Write)
10    Try
11        'Grave no arquivo
12        strOut = _
13            New System.IO.StreamWriter(fsOut)
14        strOut.Write(DateTime.Today.ToString())
15    Catch eIO As Exception
16        Console.WriteLine("Não foi possível gravar no arquivo:{0}.",
17            FileName)
18    Finally
19        strOut.Close()
20    End Try
21 Catch eFile As Exception
22    Console.WriteLine("Não foi possível abrir o arquivo:{0}.", FileName)
23 Finally
24    fsOut.Close()
25 End Try
26 End Sub
```


ANÁLISE

Aqui, tanto StreamWriter quanto File serão fechados, mesmo se erros ocorrerem. Embora isso não seja necessário, é boa prática fazer uma limpeza sempre que não precisar mais de uma variável.

Lançando Exceções

Ocasionalmente, pode-se querer notificar um usuário de que algo de muito errado aconteceu. Pode ser algo relacionado ao seu aplicativo ou uma exceção ‘normal’. Por exemplo, você pode ter criado dois objetos nesse aplicativo, Employee (funcionário) e Customer (cliente). Talvez queira gerar uma exceção se o programa tentar atribuir uma instância de Employee a uma variável criada para conter objetos Customer. Em vez de criar um novo tipo de exceção, é possível reutilizar InvalidCastException. Assim, uma nova exceção InvalidCastException poderá ser gerada e usada como notificação do aplicativo através da instrução Throw. Isso é mostrado na Listagem 6.6.

CÓDIGO**LISTAGEM 6.6** A Instrução Throw

```
1 Dim oCust As Customer = New Customer("Bob","Sjerunkl")
2 Dim oEmp As Employee = New Employee("Phil","Barr")
3 Dim oSomething As Object
4 oSomething = oEmp
5 If TypeOf oSomething Is Customer Then
6     oCust = oSomething
7 Else
8     Throw New InvalidCastException("Não é possível atribuir Employee a Customer.")
9 End If
```

ANÁLISE

A linha importante do exemplo é a oitava. Ali, você criou uma nova instância do objeto InvalidCastException e a ‘lançou’. Isso gerará uma exceção adicional que deve ser capturada por uma instrução Catch ou tratada pela rotina-padrão, causando o surgimento do depurador (se houver um ambiente de desenvolvimento instalado) ou de uma mensagem de erro.

O tratamento de exceções fornece uma maneira estruturada e limpa de proteger seus programas de erros devido a problemas no hardware, entradas inválidas do usuário ou a seus próprios enganos. Qualquer programa se tornará mais robusto se você adicionar o tratamento de exceção a um código que poderia gerá-la.

Depurando

NOVO TERMO

Tão importante quanto a tratar apropriadamente os erros e exceções é a *depuração* – o ato (e arte) de tentar encontrar e corrigir os erros do código. É claro que isso leva com frequência à pergunta, “Em primeiro lugar, por que há erros no código? Os desenvolvedores

não têm de ser astutos?”. Mesmo sendo verdade que todos os desenvolvedores são talentosos (você está lendo este livro, não?), às vezes enganos acontecem. Antes que você possa remover os erros de seus programas, passemos mais algum tempo examinando como eles podem surgir.

A Fonte dos Erros

Um erro (em um programa de computador) pode aparecer de várias maneiras. É possível que surja como

- Um engano na estratégia usada para executar alguma tarefa. Essas falhas são em geral chamadas de *erros lógicos* que às vezes são os mais difíceis de corrigir. Eles ocorrem quando se escreve o código de maneira errada. Você poderia estar ordenando uma lista de nomes, mas o modo usado para fazê-lo não levou em conta todas as possibilidades. Por exemplo, alguém com o sobrenome St. Jean deve vir antes ou depois de uma pessoa que tenha como primeiro nome Santana?
- Um engano ao inserir o código, um erro de digitação. Esses podem ser fáceis ou difíceis de resolver. De certa maneira, esses erros devem desaparecer quase inteiramente, já que o Visual Basic .NET verificará a digitação das palavras-chave. Porém, os erros de digitação podem se tornar um problema, se você não configurar `Option Explicit` em seus módulos. Se não o fizer, poderá cometer um erro acidental ao digitar um nome de variável posteriormente, fazendo com que o Visual Basic .NET crie de modo automático uma nova variável. Essa variável nova seria inicializada com o valor 0 (em variáveis numéricas), o que pode não ser o desejado. Por exemplo, sem `Option Explicit`, o código a seguir seria compilado, mas nunca retornaria nada.

```
1 Imports System
2 Module Typo
3
4     Sub Main
5         Dim sName As String
6
7         Console.WriteLine("Digite o nome: ")
8         Name = Console.ReadLine()
9         Console.WriteLine("O nome digitado foi {0}", sName)
10    End Sub
11 End Module
```

Como já foi dito, `Option Explicit` faria com que o Visual Basic .NET capturasse esse erro. Sem ele, você também poderia capturá-lo sem grandes dificuldades. No entanto, há um subconjunto desses tipos de erros que, por razões que desconheço, o cérebro sempre lê corretamente. Examinei durante alguns minutos uma seção de código na qual sabia que havia um erro de digitação, mas não vi nada. Toda vez que leio a palavra escrita incorretamente, a enxergo da maneira ‘correta’. Nessas situações, chamamos sempre outra pessoa,

que olha por cima de nossos ombros e imediatamente percebe o erro, causando algum embaraço. Depois de alguns momentos de brincadeiras, os dois voltam ao trabalho.

- Um erro causado por entradas do usuário ou pelos dados. Esses são erros difíceis de corrigir depois que ocorrem. A melhor solução para a entrada ou dados incorretos é reduzir a chance do usuário inserir informações inválidas. Você examinará alguns dos controles que podem ser usados para isso posteriormente neste livro. Além disso, ser um ‘programador defensivo’ também ajudará. Pressuponha que os dados sejam inválidos – você deve verificar os valores, para ter certeza de que são apropriados e proteger as seções de código que possam gerar exceções usando o tratamento de exceções estruturadas.

Faça	Não Faça
Use Option Explicit em todos os módulos. Isso ajudará o Visual Basic .NET a detectar muitos erros que você possa por acidente adicionar a seu código digitando incorretamente nomes de variáveis.	Não esqueça também de ativar Option Strict. Essa opção assegurará que você sempre esteja a par quando um valor for passado a uma variável de um tipo diferente. A conversão automática pode levar a erros traiçoeiros. Consciente de sua necessidade, você poderá executar a tarefa de modo mais apropriado.

Bem, pode haver muitas fontes de erros em seus programas. Algumas delas estarão sob controle, enquanto outras não. De qualquer modo, o Visual basic .NET fornece a você muitas ferramentas para ajudá-lo a corrigir, ou depurar, seus programas.

Aprendendo a Depurar com a Prática

Como já foi dito, a depuração é quase tanto uma arte quanto uma ciência. Às vezes, ela requer uma certa destreza para que se descubra o que pode estar causando um erro. Portanto, tenho a impressão de que umas das melhores maneiras de aprender a depurar é utilizar esta técnica em um programa real, e não apenas examinar suas ferramentas (o que pode ser um pouco subjetivo). Por isso, você depurará um programa que calcula uma tabela de valores de hipoteca. Ele mostra os diversos pagamentos mensais que seriam necessários a empréstimos de vários períodos e taxas de juros. A Listagem 6.7 contém a saída procurada envolvendo um empréstimo de 100 mil dólares.

RESULTADO **LISTAGEM 6.7** Resultado de um Programa Que Calcula a Tabela de Pagamentos de uma Hipoteca

Digite o valor do empréstimo:
100000

Anos	10	15	20	25	30
Juros					
5.00	1060.66	790.79	659.96	584.59	536.82
5.25	1072.92	803.88	673.84	599.25	552.20

LISTAGEM 6.7 Resultado de um Programa Que Calcula a Tabela de Pagamentos de uma Hipoteca (*continuação*)

5.50	1085.26	817.08	687.89	614.09	567.79
5.75	1097.69	830.41	702.08	629.11	583.57
6.00	1110.21	843.86	716.43	644.30	599.55
6.25	1122.80	857.42	730.93	659.67	615.72
6.50	1135.48	871.11	745.57	675.21	632.07
6.75	1148.24	884.91	760.36	690.91	648.60
7.00	1161.08	898.83	775.30	706.78	665.30
7.25	1174.01	912.86	790.38	722.81	682.18
7.50	1187.02	927.01	805.59	738.99	699.21
7.75	1200.11	941.28	820.95	755.33	716.41
8.00	1213.28	955.65	836.44	771.82	733.76
8.25	1226.53	970.14	852.07	788.45	751.27
8.50	1239.86	984.74	867.82	805.23	768.91
8.75	1253.27	999.45	883.71	822.14	786.70
9.00	1266.76	1014.27	899.73	839.20	804.62
9.25	1280.33	1029.19	915.87	856.38	822.68
9.50	1293.98	1044.22	932.13	873.70	840.85
9.75	1307.70	1059.36	948.52	891.14	859.15
10.00	1321.51	1074.61	965.02	908.70	877.57

A Listagem 6.8 mostra a tentativa inicial de criar esse programa e seu resultado, confirmando que você precisa executar uma depuração.

**CÓDIGO/
RESULTADO**
LISTAGEM 6.8 Um Programa Que Precisa de Depuração

```

1 Option Explicit On
2 Imports System
3 Imports Microsoft.VisualBasic.ControlChars
4
5 Module modMain
6
7     Sub Main()
8
9         Dim sInput As String
10        Dim dblReturn As Double
11
12        'Armazena a entrada do usuário
13        Dim dblPayment As Double
14        Console.WriteLine("Digite o valor do empréstimo:")
15        sInput = Console.ReadLine()
16        dblReturn = CDb1(sInput)
17

```


**CÓDIGO/
RESULTADO****LISTAGEM 6.8** Um Programa Que Precisa de Depuração (*continuação*)

```

18      'Crie a tabela
19      OutputMortgageTable(dblReturn)
20      Console.WriteLine("Pressione ENTER para continuar")
21      Console.ReadLine()
22  End Sub
23
24  Private Sub OutputMortgageTable(ByVal Principal As Double)
25
26      Dim iYears As Integer
27      Dim iRate As Integer
28      Dim dblAnnualInterest As Double
29
30      Console.WriteLine(Tab &"Years" &Tab &"10" &Tab &"15" &_
31          Tab &"20" &Tab &"25" &Tab &"30")
32      Console.WriteLine("Juros")
33
34      For iRate = 500 To 1000 Step 25
35          dblAnnualInterest = iRate / 100
36
37          For iYears = 10 To 30 Step 5
38              Console.Write(Format(dblAnnualInterest,"0.##0")& Tab & Tab)
39              'iYears * 12 para obter o número de meses (pressupondo
39              ➡pagamentos mensais)
40              Console.Write(Format(Payment(Principal, _
41                  dblAnnualInterest,iYears * 12), "0.00") & Tab)
42          Next
43          Console.WriteLine()
44      Next
45
46  End Sub
47
48  Public Function Payment(ByVal Principal As Double, _
49      ByVal AnnualInterest As Double, _
50      ByVal Periods As Integer) As Double
51      Dim dblMonthlyInterest As Double = AnnualInterest / 1200
52      Return Principal * dblMonthlyInterest * 1 + _
53          dblMonthlyInterest ^ Periods /1 + _
54          dblMonthlyInterest ^ Periods -1
55  End Function
56
57 End Module

```

Digite o valor do empréstimo: **100000**

CÓDIGO/ RESULTADO		LISTAGEM 6.8 Um Programa Que Precisa de Depuração (continuação)						
Anos		10	15	20	25	30		
Juros								
5.00		415.67	5.00		415.67	5.00	415.67	5.00
	415.67	5.00		415.67				
5.25		436.50	5.25		436.50	5.25	436.50	5.25
	436.50	5.25		436.50				
5.50		457.33	5.50		457.33	5.50	457.33	5.50
	457.33	5.50		457.33				
5.75		478.17	5.75		478.17	5.75	478.17	5.75
	478.17	5.75		478.17				
6.00		499.00	6.00		499.00	6.00	499.00	6.00
	499.00	6.00		499.00				
6.25		519.83	6.25		519.83	6.25	519.83	6.25
	519.83	6.25		519.83				
6.50		540.67	6.50		540.67	6.50	540.67	6.50
	540.67	6.50		540.67				
6.75		561.50	6.75		561.50	6.75	561.50	6.75
	561.50	6.75		561.50				
7.00		582.33	7.00		582.33	7.00	582.33	7.00
	582.33	7.00		582.33				
7.25		603.17	7.25		603.17	7.25	603.17	7.25
	603.17	7.25		603.17				
7.50		624.00	7.50		624.00	7.50	624.00	7.50
	624.00	7.50		624.00				
7.75		644.83	7.75		644.83	7.75	644.83	7.75
	644.83	7.75		644.83				
8.00		665.67	8.00		665.67	8.00	665.67	8.00
	665.67	8.00		665.67				
8.25		686.50	8.25		686.50	8.25	686.50	8.25
	686.50	8.25		686.50				
8.50		707.33	8.50		707.33	8.50	707.33	8.50
	707.33	8.50		707.33				
8.75		728.17	8.75		728.17	8.75	728.17	8.75
	728.17	8.75		728.17				
9.00		749.00	9.00		749.00	9.00	749.00	9.00
	749.00	9.00		749.00				
9.25		769.83	9.25		769.83	9.25	769.83	9.25
	769.83	9.25		769.83				
9.50		790.67	9.50		790.67	9.50	790.67	9.50
	790.67	9.50		790.67				
9.75		811.50	9.75		811.50	9.75	811.50	9.75
	811.50	9.75		811.50				
10.00		832.33	10.00		832.33	10.00	832.33	10.00
	832.33	10.00		832.33				

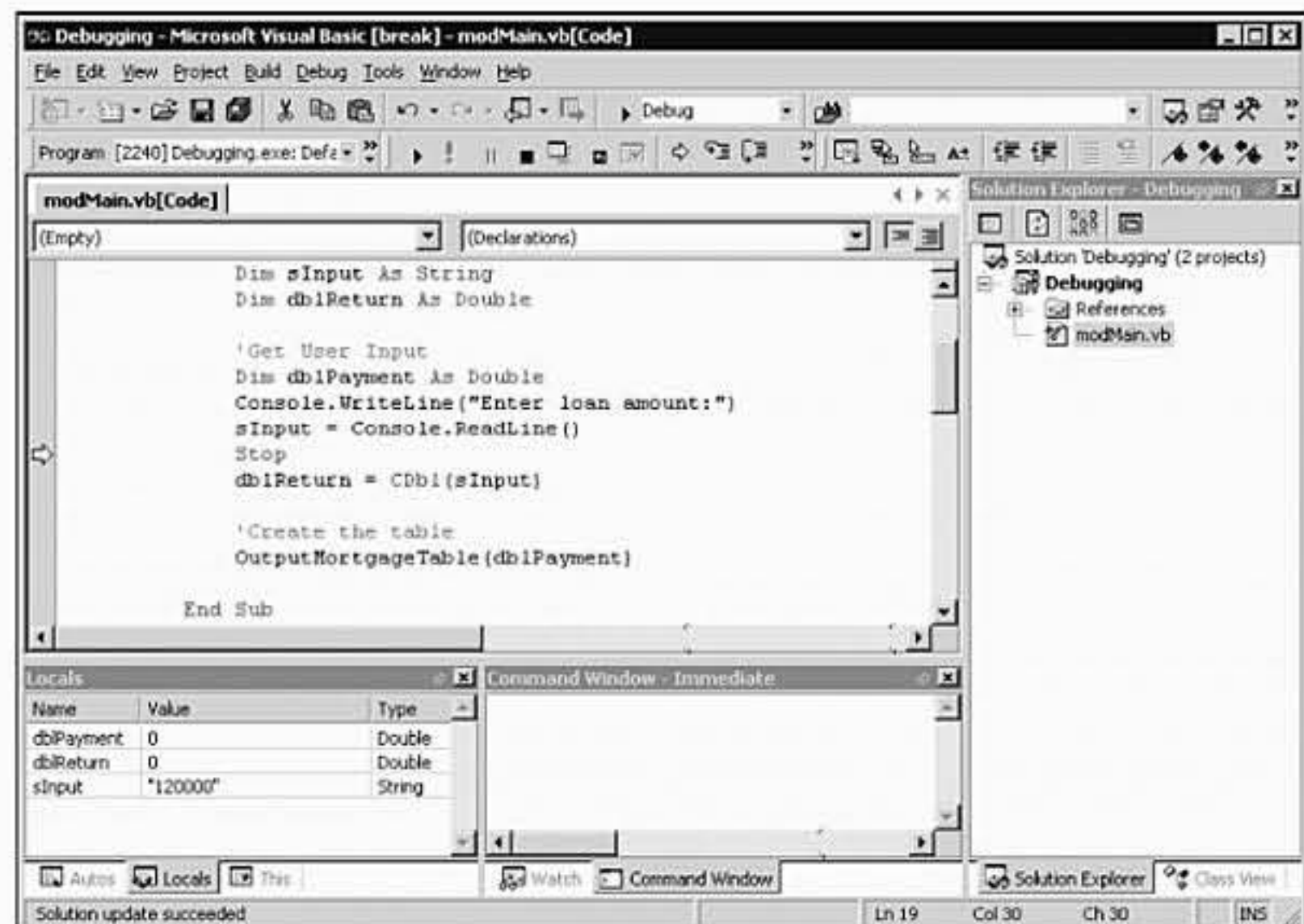
Os Modos na Vida de um Programa

Durante a criação dos programas no Visual Basic .NET, eles passarão por três modos distintos. Quando você iniciar pela primeira vez o VB .NET e estiver trabalhando em um programa, esse

será o modo Design. Para confirmação sobre qual modo está em uso, examine a barra de título da janela. Depois das palavras 'Microsoft Visual Basic', aparecerá [design]. De maneira semelhante, quando um programa for executado no IDE, veremos [run], significando que esse é o modo Run. O último modo é Break, ou Debug. Esse modo surgirá se o programa que estiver em execução for interrompido de alguma forma. Essa interrupção pode se dar devido à ocorrência de uma exceção, ou o modo Break pode ser introduzido no IDE. Veja a Figura 6.3 para ver um exemplo do IDE no modo Break. Observe que a barra de título inclui o marcador [break].

FIGURA 6.3

O Visual Basic .NET no modo Break.



Use o modo Break para auxiliar na depuração de seus programas. Nesse modo, são disponibilizadas várias ferramentas por meio das quais você poderá visualizar o conteúdo de variáveis, monitorar ou alterar o fluxo do programa, ou testar blocos de código. Essas ferramentas terão um valor inestimável na revelação e correção de erros em seus programas. Se quiser introduzir o modo Break, empregue a palavra-chave Stop, como mostra a Figura 6.3.

Como alternativa, você pode inserir um ponto de interrupção na linha onde gostaria que o programa fosse interrompido. Ele será executado normalmente até que atinja a linha e, em seguida, passará para o modo Break e retornará ao IDE. Seu programa ainda estará em processamento em segundo plano, mas se encontrará em uma pausa. Um ponto de interrupção pode ser inserido em seu código de uma entre três maneiras:

- Dê um clique na margem colorida esquerda da janela do código, próximo à linha na qual gostaria que estivesse o ponto de interrupção. Um círculo vermelho deve aparecer na margem.
- Dê um clique com o botão direito do mouse na linha do código e selecione Insert Breakpoint. Um círculo vermelho deve aparecer na margem.

- Selecione Debug, New Breakpoint no menu. Isso abrirá a caixa de diálogo New Breakpoint (veja a Figura 6.4). Essa caixa é o meio mais flexível de criar um ponto de interrupção. Ela não só permite que você defina um ponto de interrupção para encerrar o programa quando uma linha for atingida, como também que os crie para situações em que uma variável for alterada ou alcançar um valor específico, ou simplesmente em que uma certa condição for verdadeira.

FIGURA 6.4
*Caixa de diálogo
New Breakpoint.*



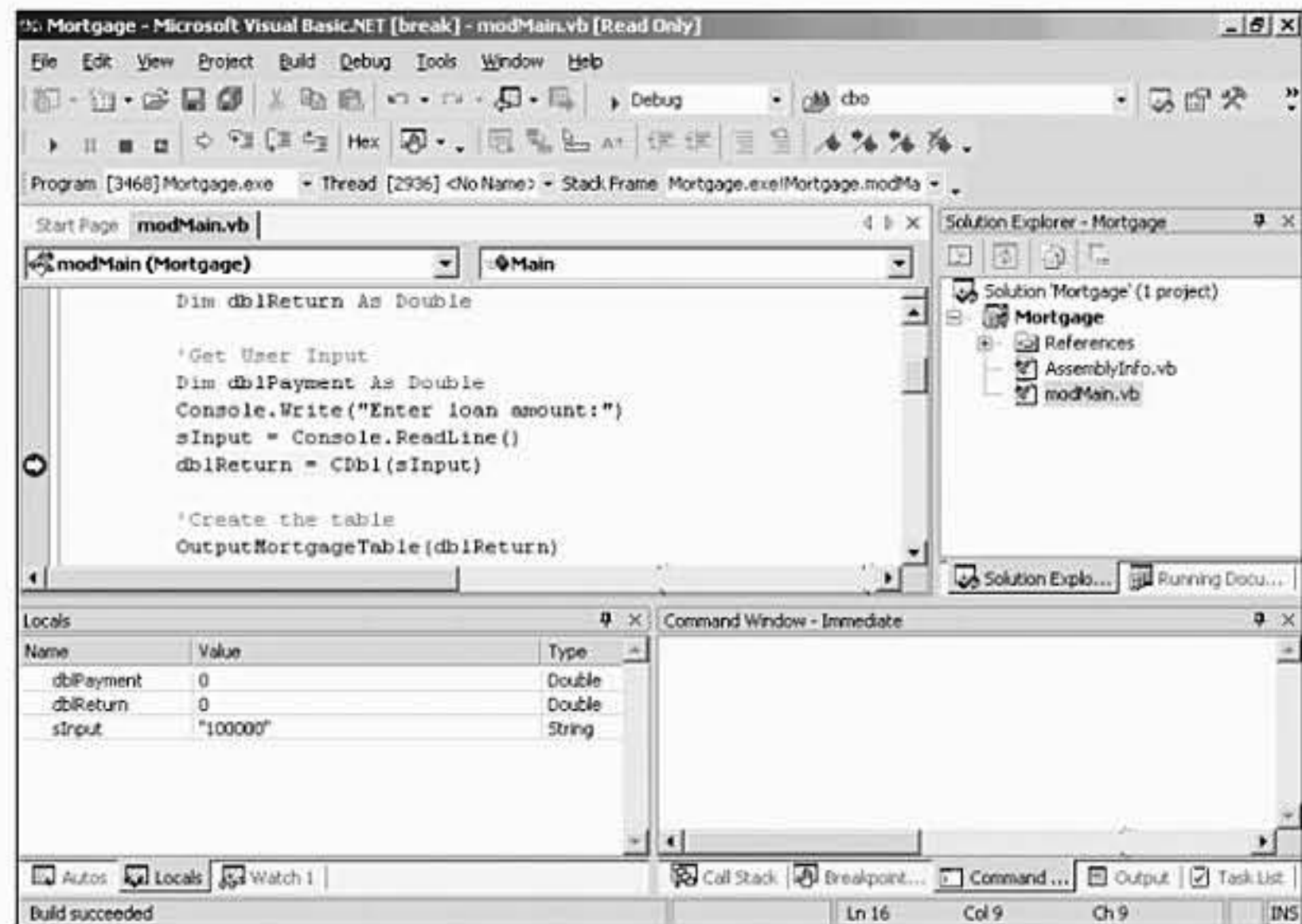
Usando Pontos de Interrupção para Examinar o Programa de Exemplo

Carregue a versão inicial do programa no IDE (do diretório OriginalMortgage), de modo que você possa testar os pontos de interrupção. Com o módulo `modMain.vb` aberto no editor, dê um clique na margem colorida, próximo à linha

```
dblReturn = Cdbl(sInput)
```

Deverá aparecer um círculo vermelho perto da linha. Se isso acontecer, inicie o programa no IDE. A janela do console deve surgir, permitindo que seja digitado um valor para o empréstimo. Insira **10000** e pressione Enter. Você deverá ser enviado novamente ao IDE onde uma seta amarela será adicionada ao círculo vermelho, como vemos na Figura 6.5. A seta amarela sempre mostra a próxima linha a ser executada.

FIGURA 6.5
*Executando um
programa com um
ponto de
interrupção.*



Percorrendo Seu Código

Um dos recursos mais úteis disponíveis no modo Break é a capacidade de percorrer o código. Ele permite que o usuário visualize o fluxo ou lógica de um programa, possivelmente encontrando erros em laços ou durante chamadas de procedimentos. Depois que você tiver entrado no modo Break, poderá começar a percorrer seu código linha a linha. Isso permitirá que examine o efeito de cada uma delas sobre as variáveis ou que se certifique de que os programas executem realmente as etapas que planejou. Há quatro funções principais relativas às etapas, disponíveis no menu Debug ou na barra de ferramentas de depuração. Elas são:

- **Step Into** Esse é o comando de etapas mais usado. Toda vez que você selecioná-lo, a próxima ação disponível será executada. Selecione-o de dentro do procedimento `OutputMortgageTable`, e você deve ver a seta amarela se mover para a linha `OutputMortgageTable(dblReturn)`. Selecione-o novamente, e você deverá passar para o procedimento `OutputMortgageTable`. Selecione-o mais uma vez, e estará na linha `Console.WriteLine(Tab & "Years" & Tab & "10" & Tab & "15" & Tab & "20" & Tab & "25" & Tab & "30")`. Observe que a linha amarela não se detém nas instruções de declaração de variáveis. Isso acontece porque elas não são consideradas código executável. A seta amarela só parará em linhas que executem algo, e não nas que declaram variáveis novas.
- **Step Out** Ocasionalmente este comando será útil se você passar por um procedimento no qual sabe que não há erros. Ele faz com que o resto do procedimento seja executado e move a seta amarela para a primeira linha depois da chamada original. Selecione-o de

dentro do procedimento `OutputMortgageTable`, por exemplo, e deverá ser enviado de volta ao procedimento `Sub Main`.

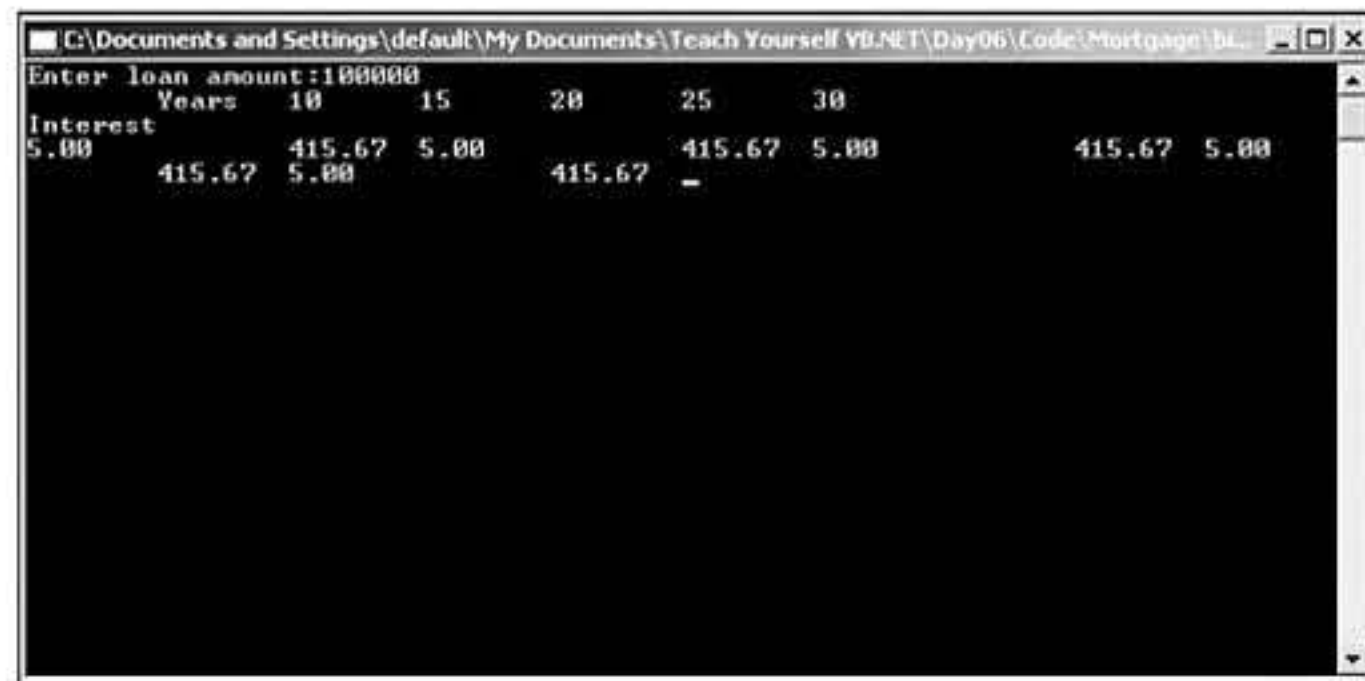
- **Step Over** Também é útil quando há uma depuração nos procedimentos em que sabemos que não existem erros. O comando `Step Over` tratará o procedimento como se fosse apenas uma linha de código. Por exemplo, se a seta amarela estivesse na linha `OutputMortgageTable` de `Sub Main`, e `Step Over` fosse selecionado, todo o código de `OutputMortgageTable` seria executado, e você seria transferido para a próxima linha de `Sub Main`.
- **Run To Cursor** Outro comando útil quando você sabe que uma seção de código não possui erros. O comando `Run To Cursor` permitirá a execução de todo o código até a linha selecionada. Para usar este modo, selecione a linha até onde deseja executar o código e a opção `Run To Cursor` no menu `Debug` ou no de atalho.

Usando o Recurso de Percorrer o Código para Examinar o Programa de Exemplo

Percorrer um programa às vezes pode revelar erros causados pelo desenvolvedor. Esses podem ser laços que terminam precocemente ou que não são executados o suficiente, ou testes lógicos que não produzem os resultados esperados.

Para percorrer o programa de exemplo, faça o seguinte:

1. Encerre o programa selecionando o botão `Stop Debugging` na barra de ferramentas ou selecione `Stop Debugging` no menu `Debug`.
2. Remova o ponto de interrupção que você criou anteriormente dando um clique no círculo vermelho da margem e gere um novo no início do primeiro laço `For` dentro de `OutputMortgageTable`.
3. Inicie o programa. Quando as informações forem solicitadas, digite **100000** como quantia do empréstimo. O programa deve entrar no modo `Break` na linha
`For iRate = 500 To 1000 Step 25`
4. Percorra o código (ou selecione `Run To Cursor`) até que alcance a linha
`Console.WriteLine()`
5. Examine a janela do console (veja a Figura 6.6). Observe que na verdade há dez itens na linha, em vez dos cinco que você esperava (um para cada coluna de 10 a 30). O valor da taxa de juros está sendo repetido a cada passagem pelo laço quando só deveria ser executado uma vez antes do laço.

FIGURA 6.6*Exibindo o cabeçalho.*

6. Após interromper o programa, transfira a linha de código que exibe a taxa de juros em cada linha da tabela para que fique antes do início do segundo laço For. O resultado deve ter a seguinte aparência:

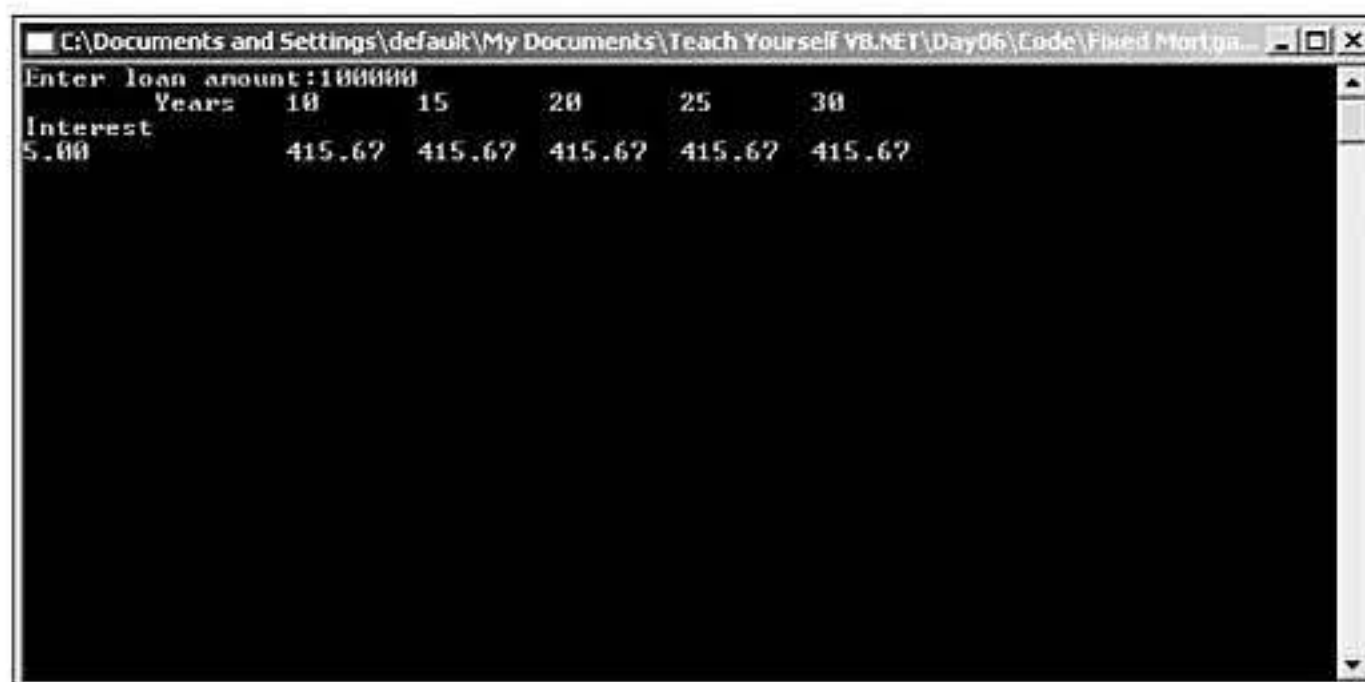
Antes:

```
For iYears = 10 To 30 Step 5
Console.Write(Format(dblAnnualInterest, "0.##0") & Tab & Tab)
```

Depois:

```
Console.Write(Format(dblAnnualInterest, "0.##0") & Tab & Tab)
For iYears = 10 To 30 Step 5
```

7. Pressione Step Into para continuar a execução do programa. Observe que a seta amarela está novamente no início do procedimento OutputMortgageTable. O Visual Basic .NET recompilou e recarregou o procedimento com as alterações.
8. Agora se você percorrer o código até a linha da etapa 4, deverá ver apenas cinco valores exibidos nessa linha, como na Figura 6.7. Eles ainda serão valores incorretos, mas pelo menos um dos erros foi corrigido.

FIGURA 6.7*Primeira correção.*

9. Execute o programa até o final usando antes Step Out para retornar a Sub Main. Examine a janela do console para ver os valores errados (eles são os mesmos para todas as linhas). Em seguida, retorne ao IDE e selecione Continue para finalizar o programa.

Examinando as Variáveis

Além de investigar o fluxo da lógica de seu aplicativo, o Visual Basic .NET fornece ferramentas que permitem a visualização do conteúdo das variáveis. Isso permite determinar se seus conteúdos representam o que você definiu e se o cálculo está correto.

Exatamente como quando percorreu o código, há várias ferramentas que você pode usar para monitorar as variáveis. Algumas das mais úteis são mostradas na Figura 6.8 e descritas a seguir:

- **Pesquisa em janela suspensa** Se você mantiver o cursor do mouse posicionado sobre uma variável do procedimento atual, uma pequena janela aparecerá mostrando seu valor no momento. É possível ver isso na Figura 6.8, em que a variável `dblReturn` atualmente armazena o valor 100000. Essa é uma ferramenta útil e prática se só quisermos fazer uma verificação breve de um valor ocasionalmente. Outro benefício dessa ferramenta é que se pode selecionar a parcela de um cálculo para saber seu valor, sem precisar que todo ele seja executado.
- **Janela Locals** Esta janela em geral aparece na parte inferior da tela quando se está no modo Break. Caso contrário, selecione-a na opção Windows do menu Debug. A janela Locals mostra todas as variáveis do procedimento atual, assim como seu tipo e valor no momento. Isso é prático quando se está percorrendo um código, porque os valores ficarão com a cor vermelha quando forem alterados. A janela Locals será útil sempre que você depurar um procedimento que altere os valores das variáveis.
- **Janela Watch** Esta janela em geral aparece na parte inferior da tela quando se está no modo Break. Caso contrário, selecione-a na opção Windows do menu Debug. A janela Watch mostrará as variáveis as quais você estiver interessado. Dê um clique com o botão direito do mouse nas variáveis e selecione Add Watch para adicioná-las à janela. Se elas estiverem no escopo, o valor atual será exibido. Senão, ela o informará que, “O nome [*nome de sua variável*] não foi declarado”. Esta janela é útil para monitorar um grupo de variáveis. Em vez de só examiná-las quando estiverem na janela Locals, será possível monitorá-las durante toda a sessão de depuração.

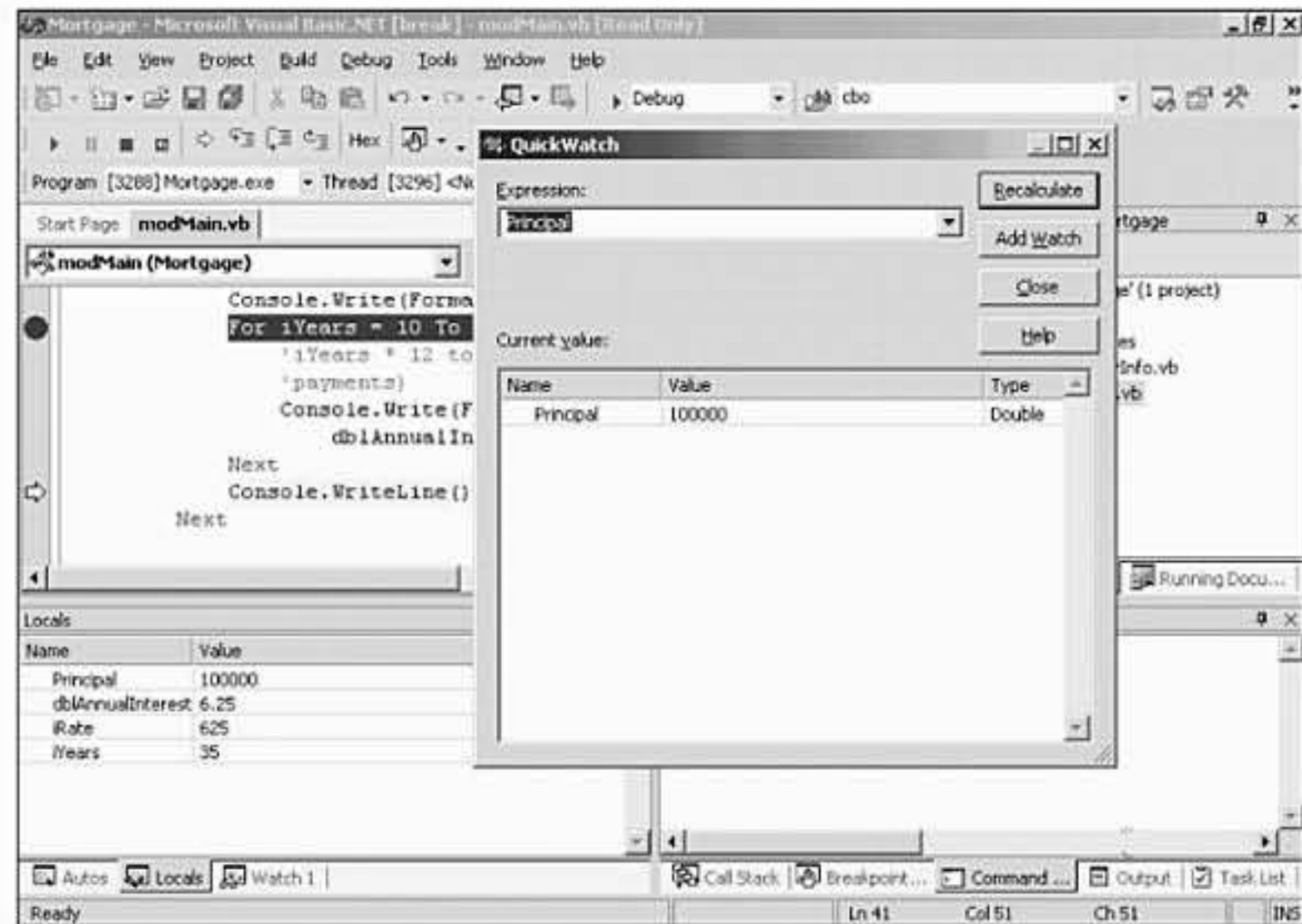
Examinando Variáveis para Depurar o Programa de Exemplo

Já que você deu uma aparência melhor para o layout da tabela da hipoteca, o único erro que ainda poderia existir estaria no cálculo de cada variável. É recomendável depurar a função `Payment` para se certificar de que apenas os valores corretos estarão em uso. Siga estas etapas:

1. Encerre o programa se ele estiver em execução agora. Remova todos os pontos de interrupção existentes e adicione um à primeira linha da função `Payment`. A linha deve ser esta
`Dim dblMonthlyInterest As Double = AnnualInterest / 1200`
2. Adicione um segundo ponto de interrupção à linha `End Sub` de `Sub Main` para permitir que você visualize a tabela antes que o programa se encerre.

FIGURA 6.8

Examinando os valores das variáveis.

**NOTA**

Mesmo começando com uma declaração, essa linha é executável por causa da atribuição.

3. Execute o programa até que alcance o ponto de interrupção. Insira **100000** para o valor do empréstimo.
4. Quando o programa entrar no modo Break, examine a janela Locals. Observe que ela inclui todos os valores de Principal, AnnualInterest e Periods. Do mesmo modo, se você mantiver o cursor do mouse sobre uma variável (por exemplo, AnnualInterest) no IDE, deverá ver uma janela suspensa mostrando o valor atual.
5. Avance uma etapa para processar essa linha. Podemos notar que a variável `dblMonthlyInterest` se alterou e ficou vermelha na janela Windows.
6. Selecione o código


```
1 + dblMonthlyInterest ^ Periods
```

Em seguida, mantenha o cursor do mouse sobre o texto selecionado. Uma janela suspensa aparecerá, informando que o valor é 1. Um cálculo rápido em algum local confirmará que isso estará correto se você executar primeiro a operação exponencial e, em seguida, a adição. No entanto, o valor certo deveria ser aproximadamente 1.647 porque a adição deve ser efetuada antes, e só então elevaríamos o resultado à quantidade referente aos períodos. Esse erro será repetido na segunda metade da função. Acrescente parênteses ao cálculo de modo que as adições sejam executadas antes e, então, teste o valor novamente.

7. De maneira semelhante, o cálculo como um todo sofre de uma grande carência de parênteses. Interrompa o programa e altere a linha de cálculo para que fique como a descrita a seguir:


```
Return Principal * (dblMonthlyInterest *
((1 + dblMonthlyInterest) ^ Periods) /
(((1 + dblMonthlyInterest) ^ Periods) - 1))
```

DICA

Os parênteses não custam nada. Quando você estiver escrevendo uma função matemática, não economize em seu uso – empregue quantos precisar para se certificar de que as operações sejam efetuadas na ordem correta. Como alternativa, é possível colocar cada etapa em uma linha, embora isso exija mais variáveis temporárias no cálculo.

8. Encerre o aplicativo, remova o ponto de interrupção na linha da função `Payment` e reexecute o programa. Você deve ver a tabela correta da hipoteca, como mostra a Figura 6.9.

FIGURA 6.9

Tabela correta da hipoteca.

Years	10	15	20	25	30
Interest 5.00	1060.66	790.79	659.96	584.59	536.82
5.25	1072.92	803.88	673.84	599.25	552.20
5.50	1085.26	817.08	687.89	614.09	567.79
5.75	1097.69	830.41	702.08	629.11	583.57
6.00	1110.21	843.86	716.43	644.30	599.55
6.25	1122.80	857.42	730.93	659.67	615.72
6.50	1135.48	871.11	745.57	675.21	632.07
6.75	1148.24	884.91	760.36	690.91	648.60
7.00	1161.08	898.83	775.30	706.78	665.30
7.25	1174.01	912.86	790.38	722.81	682.18
7.50	1187.02	927.01	805.59	738.99	699.21
7.75	1200.11	941.28	820.95	755.33	716.41
8.00	1213.28	955.65	836.44	771.82	733.76
8.25	1226.53	970.14	852.07	788.45	751.27
8.50	1239.86	984.74	867.82	805.23	768.91
8.75	1253.27	999.45	883.71	822.14	786.70
9.00	1266.76	1014.27	899.73	839.20	804.62
9.25	1280.33	1029.19	915.87	856.38	822.68
9.50	1293.98	1044.22	932.13	873.70	840.85
9.75	1307.70	1059.36	948.52	891.14	859.15
10.00	1321.51	1074.61	965.02	908.70	877.57

Retoques Finais no Programa de Exemplo

Há algumas outras alterações menores que você pode fazer para organizar melhor este programa. Por exemplo, o código que é usado para formular uma pergunta ao usuário poderia ser inserido em uma função. Isso permitiria sua posterior substituição por um código que processasse uma função semelhante, porém com formulários Windows, da Web, ou em alguma outra tela.

DICA

Uma boa idéia é isolar seções de código que sejam específicas do sistema operacional ou da interface com o usuário nos procedimentos. Assim, quando você precisar alterar a interface com o usuário, só terá de substituir o procedimento, e não o código que o utiliza.

Outra alteração semelhante é simplificar o cálculo da função de pagamento da hipoteca. Já que a expressão $(1 + \text{dblMonthlyInterest})^{\text{Periods}}$ aparece duas vezes no cálculo, você pode criá-la como uma operação isolada. Declare uma variável temporária do tipo duplo para armazenar o conteúdo e substitua o cálculo por essa variável.

Depois que você tiver feito todas as alterações no programa do exemplo, ele deve ficar semelhante ao mostrado na Listagem 6.9.

CÓDIGO**LISTAGEM 6.9** Código Depurado

```
1 Option Explicit On
2 Imports System
3 Imports Microsoft.VisualBasic.ControlChars
4
5 Module modMain
6
7     Sub Main()
8
9         Dim sInput As String
10        Dim dblReturn As Double
11
12        'Solicite a entrada do usuário
13        Dim dblPayment As Double
14        Console.WriteLine("Digite a quantia do empréstimo:")
15        sInput = Console.ReadLine()
16        dblReturn = CDb1(sInput)
17
18        'Crie a tabela
19        OutputMortgageTable(dblReturn)
20        Console.WriteLine("Pressione ENTER para continuar")
21        Console.ReadLine()
22    End Sub
23
24    Private Sub OutputMortgageTable(ByVal Principal As Double)
25
26        Dim iYears As Integer
27        Dim iRate As Integer
28        Dim dblAnnualInterest As Double
29
30        Console.WriteLine(Tab & "Years" & Tab & "10" & Tab & "15" & _
31            Tab & "20" & Tab & "25" & Tab & "30")
32        Console.WriteLine("Juros")
33
34        For iRate = 500 To 1000 Step 25
35            dblAnnualInterest = iRate / 100
36
37            Console.WriteLine(Format(dblAnnualInterest,"0.##") & Tab & Tab)
38            For iYears = 10 To 30 Step 5
39                'iYears * 12 para obter o número de meses (pressuponha pagamentos
40                'mensais)
41                Console.WriteLine(Format(Payment(Principal,
42                    dblAnnualInterest,iYears * 12),"0.00") & Tab)
43            Next
44        Console.WriteLine()
```


CÓDIGO**LISTAGEM 6.9** Código Depurado (*continuação*)

```

44      Next
45
46      End Sub
47
48      Public Function Payment(ByVal Principal As Double, _
49          ByVal AnnualInterest As Double, _
50          ByVal Periods As Integer)As Double
51          Dim dblMonthlyInterest As Double = AnnualInterest / 1200
52          Return Principal * (dblMonthlyInterest *
53              ((1 +dblMonthlyInterest) ^ Periods) / _
54              (((1 +dblMonthlyInterest) ^ Periods) - 1))
55      End Function
56
57 End Module

```

Outras Ferramentas para Depuração

Muitas outras ferramentas estão disponíveis para ajudá-lo na depuração. Muitas delas são avançadas e não fazem parte do escopo deste livro, mas você deve ao menos saber que existem. Entre elas destacamos

- Immediate Window** Está sempre disponível no menu Debug, Windows. Abre uma janela que permite inserir uma linha de código a ser executada imediatamente. Isso possibilita o teste de pequenos trechos de código. Já que está disponível havendo ou não a depuração, pode-se usar a janela Immediate para testar pequenos cálculos enquanto os programas são escritos. Um emprego comum para esta janela é exibir o conteúdo das variáveis ‘disponibilizando-o’ nesse local, por exemplo:


```
?dblMonthlyInterest
```

4.16666666666667E-03
- Quickwatch Window** Disponível no menu Debug quando se está no modo Break. Abre uma janela que mostra o valor das variáveis. Esse recurso ficou um pouco ultrapassado por causa da janela suspensa que já exibe o valor das variáveis. No entanto, a janela Quickwatch também é útil para testar as expressões tanto quanto as variáveis.
- Call Stack** Disponível no menu Debug, quando se está no modo Break. Abre uma janela que mostra a lista de procedimentos ativos no momento. Por exemplo, se seu Sub Main chamasse OutputMortgageTable, a pilha de chamadas exibiria os dois procedimentos (em ordem inversa). Você pode usar esse recurso para navegar nos dois procedimentos; isso permitiria a visualização da linha de Sub Main que chama o segundo procedimento. Esta janela pode ser empregada para depurar problemas em que a decisão errada seja selecionada em uma instrução If.

- **Disassembly** Disponível no menu Debug, quando se está no modo Break. Esta é uma ferramenta avançada que mostra a linguagem de máquina efetiva criada para o programa e só será realmente útil se você conhecer a montagem.
- **Threads** Disponível no menu Debug, quando se está no modo Break. Esta é uma ferramenta avançada que mostra as linhas de execução ativas em um programa. Útil quando se executa uma programação com múltiplas linhas de execução no Visual Basic .NET.

O Visual Basic .NET fornece muitas ferramentas que o ajudarão a disponibilizar programas sem erros. A melhor maneira de conhecer as ferramentas e saber quais delas o auxiliarão mais, é testá-las. Em vez de tentar eliminar os erros com a leitura do código, percorra-o minuciosamente, investigue os valores das variáveis utilizadas e isole as falhas empregando essas ferramentas.

Resumo

Todos os programas têm erros. Algumas vezes, o erro está na pessoa que executa o programa. Em outras, há realmente uma falha ou erro no código. Isso pode acontecer porque o autor do programa não testou todas as possibilidades de entradas ou não protegeu o programa da falta de arquivos ou bancos de dados. Em qualquer dos casos, é sua responsabilidade, como desenvolvedor, tentar assegurar que nenhum desses erros faça o usuário perder informações. Você deve se esforçar para tornar seus programas tão livres de erros quanto possível. As ferramentas de depuração do Visual Basic .NET o ajudarão nessa tarefa. Certifique-se de usá-las para percorrer seu código e de que o fluxo se encontre da forma que foi planejado. Assegure que as variáveis armazenem os valores corretos e que os cálculos sejam exatos.

De maneira semelhante, você pode proteger o programa de erros adicionando o tratamento de exceções estruturadas às seções críticas de código. Pontos propensos à falhas – como onde o código abre arquivos, lê ou grava informações, ou faz cálculos – devem ser inseridos em blocos Try...Catch...End Try. Esses blocos tratam os erros de modo sofisticado, permitindo perfeitamente que o usuário continue a usar o programa.

Na próxima lição, você trabalhará com objetos. Como poderá ver, já estamos fazendo isso; no entanto, estudaremos o assunto com mais detalhes no Dia 7, “Trabalhando com Objetos”.

P&R

P O que acontecerá se eu tiver algum código antigo que use `On Error Resume Next` e preciso reescrevê-lo para empregar o tratamento de exceções estruturadas?

R Não. Se incluir a referência `Microsoft.VisualBasic`, poderá continuar a usar `On Error Goto` e `On Error Resume Next`. Além disso, o objeto `Err` está disponível para você.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Dê o nome de três ferramentas que podem ser usadas para visualizar o conteúdo de uma variável enquanto você estiver no modo Break.

Exercícios

1. Você acabou de herdar o bloco de código a seguir. Ele solicita ao usuário dois valores. Em seguida, calcula e exibe uma tabela com a multiplicação de todos os valores entre os dois números inseridos. Por exemplo, se o usuário inserir 5 e 9, o resultado deve ser semelhante a

RESULTADO

Tabela de Multiplicação (de 5 a 9)

	5	6	7	8	9
5	25	30	35	40	45
6	30	36	42	48	54
7	35	42	49	56	63
8	40	48	56	64	72
9	45	56	63	72	81

Adicione o tratamento de exceções e depure o código da Listagem 6.10 para assegurar que ele gere o resultado desejado.

Código

LISTAGEM 6.10 Código da Tabela de Multiplicação

```

1 Imports System
2 Imports Microsoft.VisualBasic.ControlChars
3
4 Module modTable
5
6     Sub Main()
7         Dim iLow,iHigh As Integer
8         Dim sInput As String
9
10        'Permita várias execuções da geração da tabela
11        Do
12            'solicite valores
13            Console.Write("Valor mínimo (máximo igual a 20, 0 para
                finalizar): ")

```


Código**LISTAGEM 6.10** Código da Tabela de Multiplicação (*continuação*)

```

14      sInput = Console.ReadLine()
15      iLow = CInt(sInput)
16
17      If iLow <> 0 Then
18          Console.WriteLine("Valor máximo (máximo igual a 20): ")
19          sInput = Console.ReadLine()
20          iHigh = CInt(sInput)
21
22          OutputTable(iLow, iHigh)
23      End If
24      Loop Until iLow = 0
25      Console.WriteLine("Pressione ENTER para continuar")
26      Console.ReadLine()
27  End Sub
28
29  Private Sub OutputTable(ByVal MinValue As Integer, _
30      ByVal MaxValue As Integer)
31      Dim iCount, iCount2 As Integer
32
33      Console.WriteLine()
34      Console.WriteLine("Tabela de multiplicação ({0} a {1}", _
35          MinValue, MaxValue)
36
37      'exiba o cabeçalho
38      For iCount = MinValue To MaxValue
39          Console.WriteLine(Tab & CStr(iCount))
40      Next
41      Console.WriteLine()
42
43      'Exiba cada uma das linhas da tabela
44      For iCount = MinValue To MaxValue
45          For iCount2 = MinValue To MaxValue
46              Console.Write(CStr(iCount) & Tab & CStr(iCount * iCount2))
47          Next
48          Console.WriteLine()
49      Next
50  End Sub
51
52  End Module

```

- Conforme avançar no livro e tiver problemas com seu código, tente usar as ferramentas de depuração para corrigir o programa. Empregue pontos de interrupção para isolar seções que você achar que possam conter erros, percorra o código e utilize as janelas Locals e Watch para monitorar as variáveis.

3. Se você encontrar exceções em seus programas, adicione o tratamento de exceções. Procure na ajuda on-line a palavra-chave 'Exception' para examinar os diversos tipos de exceções. Como alternativa, pesquise a lista selecionando Exceptions no menu Debug, do Windows.
4. Use o aplicativo do exemplo para testar os tipos diferentes de pontos de interrupção. Tente configurar um quando o pagamento estiver abaixo de um certo valor.

SEMANA 1

DIA 7

Trabalhando com Objetos

Tanto o uso quanto a criação de objetos são essenciais para o desenvolvimento no Visual Basic .NET. Embora você já tenha trabalhado com eles no decorrer deste livro, esta lição será dedicada a esse tópico e abordará:

- O que são objetos.
- Como funcionam.
- Onde os objetos serão inseridos em seus programas.

Começaremos com a definição básica de objeto e todos os termos e conceitos que a acompanham.

Para Começar: O Que É um Objeto?

Para definir o termo ‘objeto’, poderíamos retornar no tempo até o latim medieval, ‘obiectum’ ou ‘coisa colocada na frente da mente’, mas para usarmos um termo mais simples começaremos apenas com a palavra ‘coisa’. Um objeto é uma descrição genérica de qualquer coisa que se pode querer discutir, ou usar no trabalho. Em conversas corriqueiras, o termo objeto em geral se destina a descrever apenas coisas materiais, mas em programação, em que muito pouco é realmente físico, essa definição é ampliada para incluir qualquer entidade. Podemos nos referir a um carro, pessoa ou prédio como um objeto, mas também é aceitável usar esse termo para descrever algo menos tangível como uma taxa de juros ou uma regra que será aplicada à correspondência eletrônica recebida.

O uso de objetos permitirá que seus programas sejam dedicados às entidades com as quais você estiver trabalhando, cujo objetivo final será produzir sistemas que sejam fáceis de compreender e aprimorar. Em vez de um programa em que todas as informações e códigos relacionados a apenas uma entidade estejam espalhados por todo o aplicativo, uma abordagem com base no objeto consolidará essas informações trazendo-as para a definição do objeto.

Classes e Instâncias

Para compreender os objetos você terá de passar rapidamente para o conceito de classe. As classes, como nas classificações, descrevem um grupo ou tipo de entidade, e todos os objetos são membros de alguma classe. Elas são a descrição de um objeto, fornecendo detalhes que definem seu comportamento e relatando que tipos de informação estão disponíveis sobre ele. Poderíamos, por exemplo, ter uma classe Car (carro). Ela nos relataria que as informações a seguir estariam disponíveis sobre um carro: sua cor, velocidade, peso, marca, modelo e ano. Todos esses itens são atributos do objeto, parcelas descritivas de informação às quais nos referimos como propriedades. Além dessas propriedades, a classe também descreve o que o objeto pode fazer e como. Esses comportamentos em geral são chamados de métodos do objeto, e um objeto carro poderia ter métodos como “VirarEsquerda”, “Avançar”, “DarRé” e assim por diante. É com o uso dessa classe, que fornece informações sobre as propriedades e métodos do objeto, junto a alguns outros detalhes, como um modelo, que os objetos são criados.

Retornando ao exemplo do carro, a classe seria a sua especificação, o projeto que descreve como ele funciona e se parece. Essa classe é então usada para criar muitos carros, cada um com uma existência própria, mas todos eles baseados na mesma especificação. Todos possuem as mesmas propriedades porque elas provêm da especificação, como a cor, mas cada um pode ter um valor diferente para essas propriedades (carro azul, carro vermelho, carro amarelo e assim por diante). Todos os carros também compartilhariam os mesmos comportamentos ou ações (os métodos da classe), como “Avançar” e os que fossem construídos a partir da mesma especificação executariam essa ação da mesma maneira.

Criar um carro com base na especificação é equivalente a gerarmos um objeto com base em uma classe. Portanto, enquanto o Ford Thunderbird seria uma classe, o T-Bird azul de Bob seria um objeto. Cada um desses objetos é, individualmente, uma *instância* da classe e não há limite para quantas podem ser criadas. Todas as instâncias compartilham o modelo, ou descrição, fornecido por sua classe. Isso significa que todas as instâncias da classe Car terão as mesmas propriedades e métodos e se comportarão da mesma maneira. No entanto, cada instância terá valores próprios para suas propriedades; todos os carros têm uma propriedade em comum que é a cor, mas cada um deles possui uma cor diferente.

Referências

Em programação, um conceito adicional é introduzido, o da *referência* a um objeto. Uma variável de objeto, qualquer variável declarada como um objeto de algum tipo (Dim myCar As Car),

não contém o objeto propriamente dito, mas apenas uma referência a ele. É diferente dos tipos comuns de variáveis, como os inteiros ou strings, nos quais elas armazenam diretamente o valor. Isso significa que mais de uma variável de objeto pode se referir, ou apontar, ao mesmo objeto. Em geral, esse não é o caso. O objeto é criado e usado em uma variável, mas é importante compreender a diferença entre os objetos e outros tipos de variáveis a esse respeito.

Passando o Conceito para o Código

Passemos alguns desses conceitos para código. Primeiro, para criar objetos, você deve ter uma classe. Existem muitas classes disponíveis que foram criadas por outras pessoas; na verdade, o .NET Framework inteiro é uma ‘biblioteca de classes’, um conjunto de classes pré-existentes que podem ser usadas em seus programas. Em nosso exemplo, no entanto, criaremos nossa própria classe, Car, porque é algo simples de fazer.



A finalidade desta lição não é ensinar tudo a respeito de classes, e sim o suficiente sobre a construção de classes para que você compreenda as que já foram criadas. Esse conhecimento será útil para se aprofundar no .NET Framework no Dia 8, “Introdução ao .NET Framework”, e ele fornecerá uma visão inicial para o desenvolvimento de seus próprios objetos no Dia 15, “Criando Objetos no Visual Basic .NET”.

Abra o Visual Studio .NET e crie um novo projeto (Empty Project) na pasta Visual Basic Projects (veja a Figura 7.1).

Esse projeto não contém arquivo de nenhum tipo, portanto precisamos adicionar um a ele; selecione Project, Add Class no menu. Isso adicionará uma classe vazia ao projeto (chame-a clsCar.vb), um ótimo ponto de partida para esse exercício. Agora, foi criado um shell de uma classe nesse arquivo, fornecendo o código a seguir:

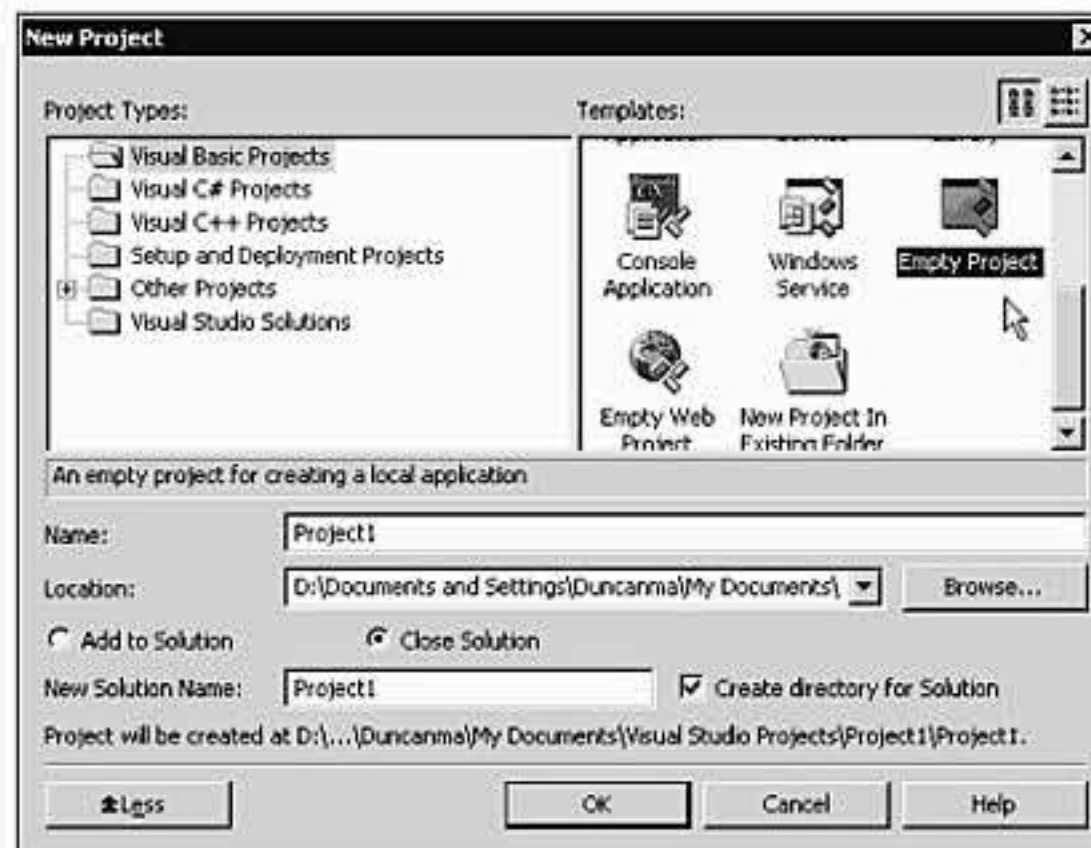
```
Public Class clsCar
```

```
End Class
```

Neste ponto, uma classe foi criada, mas está completamente vazia. Agora, dentro desse shell, podemos começar a descrever as propriedades e métodos dos objetos de nossa classe Car. Qualquer item que inserirmos nessa classe fará parte de todas as instâncias do objeto criadas a partir dele. Por enquanto, adicionaremos as propriedades Make, Model e Color.

FIGURA 7.1

Um projeto vazio é iniciado sem conter nenhum arquivo.



Propriedades

As propriedades são atributos de um projeto que você pode recuperar e configurar, e que podem ser adicionados a uma classe de uma entre duas maneiras. O primeiro método, e também o mais simples, é declarar uma variável como `Public`. Como discutimos no Dia 3, “Introdução à Programação com o Visual Basic .NET”, `Public` descreve o *escopo* da variável. No caso de uma classe, qualquer item que for declarado como `Public` estará disponível para todas as pessoas que usarem essa classe. Em nossa classe, apenas incluir a linha `Dim Year As Integer` não criaria uma propriedade exposta porque `Year` só estaria disponível internamente. Se fôssemos declarar `Year` como `Public Year As Integer`, então, de repente ela seria exposta em todas as instâncias desse objeto.

Instruções de Propriedade

Como alternativa, as propriedades podem ser declaradas com o uso de uma sintaxe `Property` especial:

```
Dim myVar as <Tipo de dado da propriedade>
Public Property <Nome da propriedade>() As <Tipo de dado da propriedade>
    Get
        Return myVar
    End Get
    Set(ByVal Value As <Tipo de dado da propriedade>)
        myVar = value
    End Set
End Property
```

As duas partes da definição da propriedade, `Get` e `Set`, representam a recuperação do valor dela e a configuração desse valor. Em geral, o código de `Get` só retorna o valor de uma variável interna (uma variável no nível da classe que representa a propriedade, normalmente com o prefixo `m` para indicar um valor membro), e o de `Set` insere um valor (que é fornecido por meio da palavra-chave especial `Value`) na mesma variável interna. Para implementar a propriedade da cor em `clsCar`, poderíamos usar um código como o da Listagem 7.1.

CÓDIGO**LISTAGEM 7.1** Criando Propriedades em Classes

```
1 Public Class clsCar
2     Dim m_sColor As String
3     Public Property Color() As String
4         Get
5             Return m_sColor
6         End Get
7         Set(ByVal Value As String)
8             m_sColor = value
9         End Set
10    End Property
11 End Class
```

Agora, adicione o código da propriedade aos de `Make` e `Model`, lembrando de também acrescentar duas variáveis internas complementares (essas duas propriedades devem usar strings como seu tipo de dado). No aspecto geral, essa segunda maneira de declarar propriedades pode produzir resultados semelhantes, mas é muito mais flexível porque permite que você defina qual o código necessário para controlar apropriadamente a configuração e a recuperação de um valor da propriedade.

Propriedades `ReadOnly` e `WriteOnly`

Não será raro surgirem propriedades que você achará melhor serem de leitura, como a de uma versão ou talvez uma data de criação. É muito menos comum, se não impossível, haver uma propriedade que possa ser alterada, porém não lida, como um campo de senha. Em versões anteriores do Visual Basic, você poderia criar essas propriedades de leitura e de gravação apenas optando por não implementar o trecho `Set` ou `Get` de suas definições. Isso é parcialmente o que se faz no Visual Basic .NET. Se uma propriedade de leitura estiver sendo criada, o trecho `Set` de sua definição não deve ser incluído, porém `ReadOnly` deverá ser especificada como palavra-chave adicional. É o que mostra o código a seguir com a propriedade `Description`, que é um valor calculado e, portanto, não faria muito sentido gravá-lo. Não seria interessante configurar uma propriedade como `Description` porque, na verdade, ela é apenas o resultado de um cálculo.

```
1 Public ReadOnly Property Description() As String
2     Get
3         Return m_sColor & " " & m_sMake & " " & m_sModel
4     End Get
5 End Property
```

A palavra-chave usada para uma propriedade de gravação é `WriteOnly` e, nesse caso, só é inserido o trecho `Set` em sua definição:

```
1 Dim m_sPassword As String
2 Public WriteOnly Property Password() As String
```



```
3 Set(ByVal Value As String)
4     m_Password = Value
5 End Set
6 End Property
```

Criando a Instância de um Objeto

Depois que você tiver essa classe, poderemos gerar uma instância dela em outra parte de nosso projeto. Adicione um módulo a ele (selecione Project, Add Module no menu) chamado `Main.vb` e crie uma sub-rotina `Sub Main()` em seu interior. Nesse ponto, seu módulo deve ser semelhante ao código da Listagem 7.2.

CÓDIGO

LISTAGEM 7.2 Criando um Módulo Novo para Teste

```
1 Module Main
2     Sub Main()

3     End Sub
4 End Module
```

Esse novo procedimento `Main()` é o ponto de partida de nossa solução. Seu código será executado quando processarmos esse aplicativo, e é aí que escreveremos as instruções que trabalharão com nossa nova classe. Para começar a usá-la, primeiro temos de criar uma variável do tipo apropriado:

```
Dim objCar As clsCar 'ou Chapter7.clsCar, examinaremos melhor esse ponto posteriormente
```

Essa linha parece estar declarando uma variável comum, como uma string ou um inteiro, mas ela é muito diferente. Aqui, temos uma variável que poderia conter a referência a um objeto do tipo `clsCar`, mas que na verdade não armazena nada. Quando declaramos uma variável alfanumérica, uma string é criada. Embora possa não ter nenhum dado, ela realmente existe. Nesse caso, não temos nada em `objCar`. Portanto, a próxima etapa é criar uma instância de nossa classe, o que podemos fazer com a palavra-chave `New`:

```
objCar = New clsCar()
```

Agora sim criamos uma instância de `clsCar` e atribuímos à variável `objCar` uma referência a esse novo objeto. Já podemos, por meio de `objCar`, acessar as propriedades desse objeto:

```
1 objCar.Color = "Red"
2 objCar.Make = "Ford"
3 objCar.Model = "Escort"
```

Essas propriedades podem ser recuperadas com muita facilidade:

```
Dim sCarDesc As String
```



```
sCarDesc = objCar.Color & " " & objCar.Make & " " & objCar.Model
```

Já que os objetos funcionam por referência, podemos criar variáveis adicionais, todas apontando para o mesmo local:

```
1 Dim objCar As clsCar
2 Dim objCar2 As clsCar
3 Dim sMake As String
4 objCar = New clsCar()
5 objCar2 = objCar
6 objCar.Make = "Ford"
7 objCar2.Make = "Chevy"
8 'objCar2.Make é igual a objCar.Make
```

Compare isso com uma variável que não seja de objeto, como uma string, em que o valor real é transferido entre os locais:

```
1 Dim sFirstValue As String
2 Dim sSecondValue As String
3
4 sFirstValue = "Dog"
5 sSecondValue = sFirstValue
6 sSecondValue = "Cat"
7 ' sFirstValue <> sSecondValue
```

Normalmente, quando lidamos com variáveis, logo que elas saem de escopo (veja o Dia 3 para obter mais informações sobre as variáveis e o escopo), deixam de existir. Já que múltiplas variáveis podem apontar para um único objeto, as regras que controlam a eliminação dele são um pouco diferentes. Quando todas as variáveis que fazem referência ao objeto não existirem mais, ele se tornará inacessível e acabará sendo eliminado pelos serviços em segundo plano da plataforma .NET. Esse processo, que é chamado de *coleta de lixo*, permite que o programa crie e libere livremente os objetos sabendo que o sistema o acompanha, limpando tudo que é desnecessário. Dessa maneira, a plataforma .NET faz a limpeza para seu programa, fornecendo outro serviço subjacente para que nenhum dos códigos precise se encarregar desse tipo de operação.

Encapsulando Códigos em Suas Classes

Agora você já viu, no código, a criação de uma classe, a instanciação de objetos com base nessa classe e a manipulação das propriedades desse objeto. Dando continuidade, considere a idéia de uma classe, diferente de um UDT, (novamente, veja o Dia 3 para obter mais informações sobre as estruturas ou tipos definidos pelo usuário), que descreva mais do que apenas um conjunto de valores, podendo incluir também o comportamento. Para fornecer essa implementação do comportamento, a classe apresentará mais do que apenas simples códigos de configuração e recuperação de valores; ela também poderá possuir um código para executar a validação da propriedade e outras ações. Em nossa classe `clsCar`, podemos demonstrar esse recurso adicionando um código de validação a nossas propriedades. Em sua estrutura atual, você poderia con-

figurar a propriedade Color com qualquer valor alfanumérico, mesmo se ele nem fosse uma cor (objCar.Color = "John"). Para tornar nossa representação do objeto Car um pouco mais sofisticada, podemos adicionar um trecho de código que verifique qualquer valor informado em uma lista de cores e rejeite o que não tiver uma correspondência. Isso envolve reescrever a rotina da propriedade Color como mostra a Listagem 7.3.

CÓDIGO**LISTAGEM 7.3** Adicionado Validação à Propriedade Color

```
1 Public Class clsCar
2     Dim m_sColor As String
3
4     Public Property Color() As String
5
6         Get
7             Return m_sColor
8         End Get
9
10        Set(ByVal Value As String)
11
12            Select Case Value.ToUpper()
13                Case "RED"
14                    m_sColor = Value
15                Case "YELLOW"
16                    m_sColor = Value
17                Case "BLUE"
18                    m_sColor = Value
19                Case Else
20                    Dim objException As System.Exception
21                    objException = New System.ArgumentOutOfRangeException()
22                    Throw objException
23            End Select
24        End Set
25    End Property
26
27 End Class
```

Agora, uma tentativa de configurar a propriedade com uma cor inválida (inválida na lista interna de nosso código, que considera a cor popular ‘lilás’, por exemplo, como inadequada) resultará no lançamento de uma exceção. Para obter mais informações sobre as exceções e o tratamento de erros com base nelas, veja o Dia 6, “O Que Fazer quando Programas Bons Apresentam Problemas e para Se Certificar de Que Isso Não Aconteça”. Como descrito nessa lição, podemos tratar corretamente essa exceção reescrevendo nosso código de teste (contido em Sub Main()) para que inclua uma estrutura Try...Catch. Esse código alterado é mostrado na Listagem 7.4.

CÓDIGO**LISTAGEM 7.4** Incluído Tratamento de Erros em Nosso Código de Teste

```
1 Sub Main()  
2     Dim objCar As clsCar  
3     Dim sColor As String  
4  
5     objCar = New clsCar()  
6     objCar.Year = 1999  
7  
8     Try  
9         objCar.Color = "Green"  
10    Catch objException As System.ArgumentOutOfRangeException  
11        'Opa! Trate o erro!  
12        System.Console.WriteLine("Opa!")  
13    End Try  
14    sColor = objCar.Color  
15  
16    objCar.Make = "Ford"  
17    objCar.Model = "Escort"  
18    System.Console.WriteLine(objCar.Description)  
19 End Sub
```

Além da validação da propriedade, que por si só já é poderosa, uma classe pode conter uma função ou sub-rotina que não faça parte de nenhum código de recuperação ou configuração da propriedade, geralmente chamada *método*. Os métodos são criados para fornecer uma funcionalidade relacionada a um objeto e em geral agem com base nas informações da propriedade (porque estão disponíveis). Para nossa classe `clsCar`, um método útil pode ser gerar o tempo de existência do carro comparando a data e a hora atuais com uma propriedade desse que represente sua data de produção. Criar a propriedade da data de produção é relativamente simples. Será do mesmo modo como a propriedade anterior exceto que é uma data, e adicionar esse método é mesmo tão fácil quanto criar uma função pública na definição de nossa classe.

Primeiro, a nova propriedade:

```
1 Dim m_dtManufactured As Date  
2  
3 Public Property Manufactured() As Date  
4     Get  
5         Return m_dtManufactured  
6     End Get  
7     Set(ByVal Value As Date)  
8         m_dtManufactured = value  
9     End Set  
10 End Property
```


**NOTA**

Se criássemos essa função como privada, ela poderia ser utilizada a partir dessa classe, mas não estaria disponível para nenhum outro código. Como alternativa, também podemos declarar elementos de nossa classe (propriedades, funções, sub-rotinas) como Friend. Essa declaração assegura que códigos dentro da mesma montagem possam acessar essas partes da classe como se fossem públicas, mas que elas estejam ocultas (privadas) para qualquer código externo à montagem da classe. (As montagens serão abordadas com detalhes como parte da implantação no Dia 19, "Implantando Seu Aplicativo"; por enquanto considere-as como as várias partes de uma mesma solução.)

A Listagem 7.5 mostra o novo método.

CÓDIGO**LISTAGEM 7.5** O Método GetAge

```
1 Public Function GetAge() As Long
2     Dim lngDays As Long
3     Dim dtCurrent As Date
4     dtCurrent = System.DateTime.Today
5     lngDays = dtCurrent.Subtract(m_dtManufactured).Days
6     Return lngDays
7 End Function
```

Depois que tivermos adicionado esse código a nossa classe, poderemos chamá-lo por meio de qualquer instância (objCar.GetAge()), como na Listagem 7.6.

CÓDIGO**LISTAGEM 7.6** Usando Nosso Método Novo

```
1 Dim objCar As clsCar
2 objcar = New clsCar()
3 objCar.Manufactured = #30/1/2000#
4 System.Console.WriteLine(objCar.GetAge())
```

Em algumas situações, o novo método seria mais adequado como uma propriedade (que, então, renomearíamos para Age, já que seria um atributo, e não uma ação) porque na verdade não executa nenhuma ação e só retorna um valor. Para nossa classe Car, um exemplo melhor de método poderia ser algo relacionado à ação, como StartEngine, cuja implementação é fornecida na Listagem 7.7.

CÓDIGO**LISTAGEM 7.7** Nosso Novo Método Orientado à Ação

```
1 Public Sub StartEngine()
2     System.Console.WriteLine("Vroom,Vroom...!!!")
3 End Sub
```


Com esse código adicionado a `clsCar`, teríamos disponível um método mais orientado à ação. Por meio de uma combinação de propriedades (algumas com código e outras sem) e métodos, é possível criar objetos complexos. Lembre-se de que todos os objetos que compõem o .NET Framework (como `System.Console` e outros) foram construídos de acordo com essas regras, e os que você criar terão as mesmas características deles. A única diferença existente entre os objetos do .NET Framework e os seus é que esses não precisam ser escritos no Framework!

Tópicos Avançados

Embora as propriedades e métodos permitam que você crie objetos complexos e poderosos, o suporte ao objeto na plataforma .NET possui muitos outros recursos além desses básicos. Os recursos avançados facilitarão a representação de conceitos e entidades em nosso código, produzindo um sistema mais simples em termos de utilização, manutenção e expansão. Apesar de esse suporte ser amplo, forneceremos uma visão geral das cinco áreas principais: sobreposição, herança, construtores, espaços de nome e membros estáticos da classe.

Sobreposição

Esta seção aborda os aspectos básicos da sobreposição; detalhes adicionais serão discutidos no Dia 15, quando criarmos nossos próprios objetos. A sobreposição permite que uma única função ou sub-rotina com vários parâmetros diferentes seja chamada. Isso possibilita que um só método, por exemplo, aceite os parâmetros em combinações distintas ou usando tipos de dados diversos. Portanto, voltando ao nosso exemplo de `clsCar`, seria possível projetar o método `GetAge` de modo que ele pudesse funcionar de uma entre várias maneiras. A implementação existente não usa parâmetros e retorna a diferença em dias entre a data atual e a de produção, mas talvez também quiséssemos permitir que o usuário de nosso objeto solicitasse a diferença entre a data da produção e qualquer outra aleatória e ainda especificasse a unidade de tempo a ser usada. Para fazer isso sem esse conceito de sobreposição, teríamos de criar uma função diferente para cada chamada possível, como na Listagem 7.8.

CÓDIGO

LISTAGEM 7.8 Criando Várias Opções em um Único Método

```
1 Public Function GetAge() As Long
2     Dim lngDays As Long
3     Dim dtCurrent As Date
4     dtCurrent = System.DateTime.Today
5     lngDays = dtCurrent.Subtract(m_dtManufactured).Days
6     Return lngDays
7 End Function
8
9 Public Function GetAgeAtDate(ByVal dtPointInTime As Date) As Long
10    Dim lngDays As Long
11    lngDays = dtPointInTime.Subtract(m_dtManufactured).Days
```


CÓDIGO**LISTAGEM 7.8** Criando Várias Opções em um Único Método (*continuação*)

```
12 Return lngDays
13 End Function
14
15 Public Function GetAgeInUnits(ByVal sUnit As String) As Long
16     Dim lngUnits As Long
17     Dim dtCurrent As Date
18     Dim tsDifference As System.TimeSpan
19     dtCurrent = System.DateTime.Today
20     tsDifference = dtCurrent.Subtract(m_dtManufactured)
21     Select Case sUnit
22         Case "Hours"
23             lngUnits = tsDifference.Hours
24
25         Case "Days"
26             lngUnits = tsDifference.Days
27
28         Case "Minutes"
29             lngUnits = tsDifference.Minutes
30
31         Case "Years"
32             lngUnits = tsDifference.Days \ 365
33     End Select
34     Return lngUnits
35 End Function
```

Todas essas funções na verdade são apenas variações de `GetAge`, mas cada lista de parâmetros diferente e seu código correspondente precisa de seu próprio nome de função. Com a sobreposição, eliminamos essa restrição e podemos criar todas essas funções usando o mesmo nome, `GetAge`. Para usar esse recurso, tudo que precisamos fazer é adicionar a palavra-chave `Overloads` à frente de cada (incluindo a original) declaração de função (antes de `Public`) e alterar todos os nomes das funções para que usem apenas um:

```
Public Overloads Function GetAge() As Long
End Function
```

```
Public Overloads Function GetAge(ByVal dtPointInTime As Date) As Long
End Function
```

```
Public Overloads Function GetAge(ByVal sUnit As String) As Long
End Function
```

No código de nosso exemplo que usa essa função, já podemos escolher qualquer uma das três declarações que ela pode utilizar (veja a Figura 7.2).

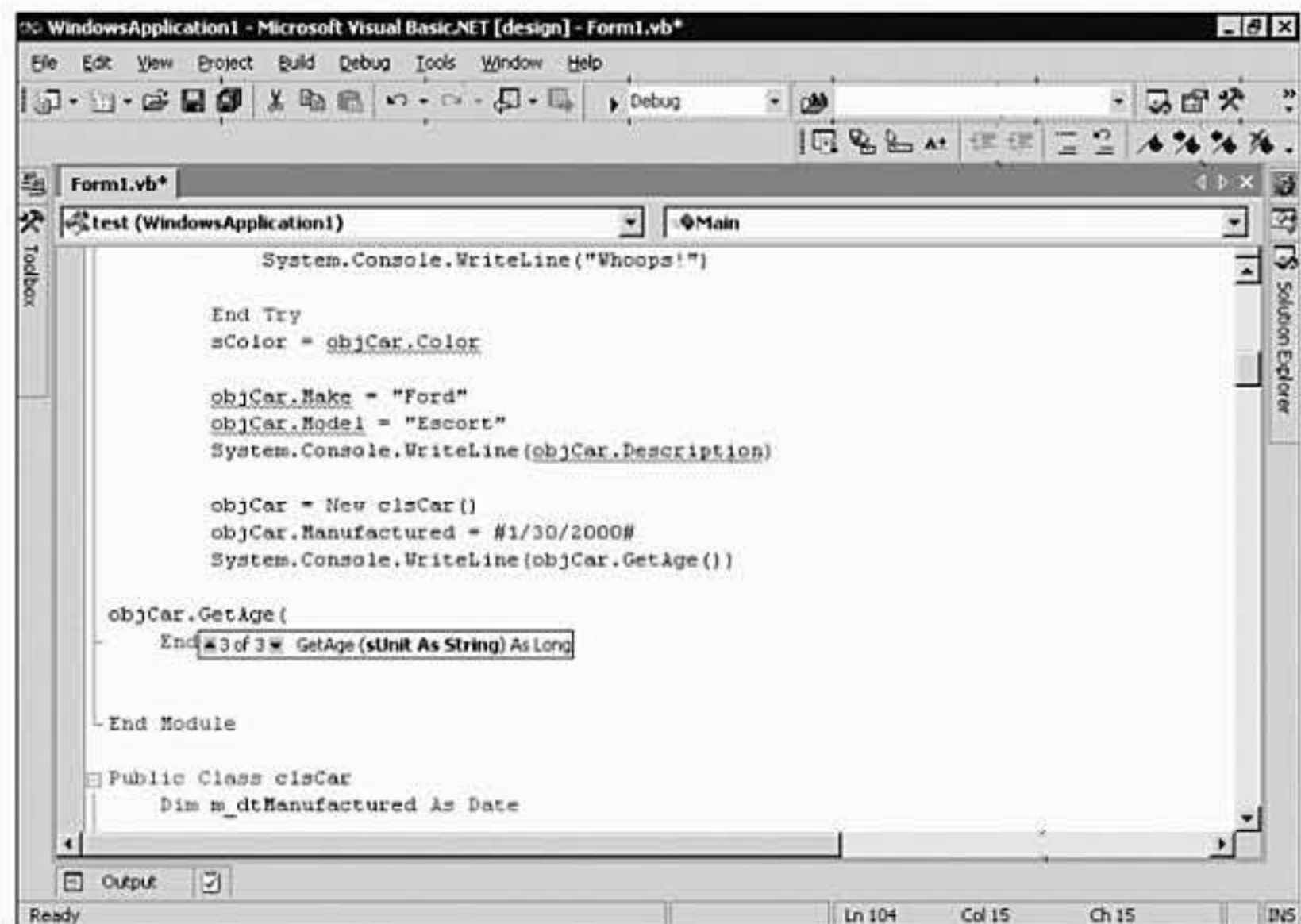


Cada declaração de função deve ser diferente de alguma maneira – quantidade de parâmetros, tipo de dados do parâmetro ou do valor de retorno –, ou não poderá ser usada.

A sobreposição representa o conceito de que a mesma ação ou solicitação pode ser usada de várias maneiras e permite que você use esse recurso na modelagem de seu objeto sem ter de recorrer à criação de diversos métodos diferentes (GetAge, GetAgeFromDate e outros). Essa técnica é usada em todo o .NET Framework para permitir a chamada de funções com vários conjuntos de parâmetros diferentes. Considere, por exemplo, o método `System.Console.WriteLine`, que pode ser chamado usando qualquer uma das 18 listas diferentes de parâmetros. Como o Framework demonstra, essa é uma maneira útil de simplificar os objetos e fornecer mais opções aos programas que os usam.

FIGURA 7.2

Todas as versões disponíveis de uma função serão mostradas por meio do IntelliSense quando você usar o Visual Studio .NET para criar um cliente.



Herança

Para algumas pessoas, este é um dos recursos mais estimulantes do Visual Basic .NET – um recurso considerado fundamental para a criação de sistemas com base em objetos e que faltava no Visual Basic até esta versão. Não vou questionar essa opinião, mas de alguma forma consegui desenvolver sistemas por muitos anos, sem a herança, antes que a plataforma .NET chegasse. Independentemente disso, a inclusão da herança na linguagem do Visual Basic é um recurso importante e vale uma pequena discussão.

Como já abordei nesta lição, os objetos são maneiras de você representar conceitos ou entidades no código, e todos os recursos de objetos no Visual Basic foram projetados para ajudá-lo a tornar a representação o mais útil possível. Em muitas situações, uma entidade ou conceito na verdade

é o que chamaríamos de um subobjeto de uma entidade ou conceito mais básico. Vários exemplos disso são usados em livros de programação e infelizmente não apresentarei nada tão revolucionário. Considere nossa classe de objetos criada para representar carros, como um Ford Mustang, um Toyota Celica ou um Chevy Cavalier. A classe teria várias propriedades, como Doors (quantidade de portas que o carro possui), MaxSpeed, Color e outras.

A classe Car geral, na verdade, contém várias subclasses de objetos, como Hatchbacks e Convertibles. É claro que essas classes teriam todas as propriedades de sua classe-pai, Car, como Doors, MaxSpeed e Color, mas também poderiam ter propriedades exclusivamente suas. Um carro de dois módulos poderia ter propriedades que descrevessem o tamanho e o comportamento de sua porta traseira. Esse relacionamento, entre Car e suas subclasses, Hatchback e Convertible, seria considerado um relacionamento pai-filho, e o método para representá-lo em nossos sistemas é chamado *herança*. Diz-se da classe Hatchback que ela é herdeira de sua classe básica, Car. Esse relacionamento significa que, além de qualquer método e propriedade criados na classe-filha, ela também possuirá todas as propriedades e métodos herdados do seu pai.

O exemplo anterior foi iniciado com nossa classe Car e seguiu daí em diante. Usemos o mesmo ponto de partida, Car, e sigamos em direção a um exemplo mais detalhado de herança. Para começar, poderíamos ter uma classe básica Vehicle para representar qualquer tipo de veículo (barco, carro, caminhão, avião) e que possuísse as propriedades MaxSpeed, NumberOfPassengers, Color e Description. Essa classe seria facilmente representada em um código do Visual Basic, como mostra a Listagem 7.9.

CÓDIGO**LISTAGEM 7.9** Nossa Classe de Veículos

```
1 Public Class Vehicle
2
3     Public Property MaxSpeed() As Long
4         Get
5         End Get
6         Set(ByVal Value As Long)
7         End Set
8     End Property
9
10    Public Property NumberOfPassengers() As Long
11        Get
12
13        End Get
14        Set(ByVal Value As Long)
15        End Set
16    End Property
17
18    Public Property Color() As String
19        Get
20
```


CÓDIGO**LISTAGEM 7.9** Nossa Classe de Veículos (*continuação*)

```
21      End Get
22      Set(ByVal Value As String)
23      End Set
24  End Property
25
26  Public Function Description() As String
27      End Function
28 End Class
```

O código que é usado em vários procedimentos dessa classe não é relevante para nosso exemplo, portanto o deixaremos de fora por enquanto. Se passássemos para algum outro código, tentando usar nosso objeto (que poderia ser o da sub-rotina Sub Main () de nosso projeto, como em exemplos anteriores), perceberíamos que é possível criar objetos do tipo Vehicle e trabalhar com suas propriedades, como na Listagem 7.10.

CÓDIGO**LISTAGEM 7.10** Trabalhando com Nossa Classe de Veículos

```
1 Module UseVehicle
2     Sub Main()
3         Dim objVehicle As Vehicle
4         objVehicle = New Vehicle()
5
6         objVehicle.Color = "Red"
7         objVehicle.MaxSpeed = 100
8     End Sub
9 End Module
```

Agora, adicionando uma classe complementar a nosso projeto (veja a Listagem 7.11), podemos criar uma classe (Car) que herde características de Vehicle, exatamente como a classe real de objetos Car é uma subclasse ou filha da classe Vehicle. Já que estamos gerando uma classe projetada para tratar apenas de carros, podemos adicionar duas propriedades (NumberOfDoors e NumberOfTires) específicas dessa subclasse de Vehicle.

CÓDIGO**LISTAGEM 7.11** Criando uma Classe-Filha

```
1 Public Class Car
2     Inherits Vehicle
3
4     Public Property NumberOfTires() As Integer
5         Get
6
7         End Get
```


CÓDIGO**LISTAGEM 7.11** Criando uma Classe-Filha (*continuação*)

```
8      Set(ByVal Value As Integer)
9
10     End Set
11 End Property
12
13 Public Property NumberOfDoors() As Integer
14     Get
15
16     End Get
17     Set(ByVal Value As Integer)
18
19     End Set
20 End Property
21
22 End Class
```

A parte essencial desse código é a linha `Inherits Vehicle`, que informa ao Visual Basic que a classe é filha de `Vehicle` e, portanto, deve herdar todas as propriedades e métodos dessa classe. Novamente, não há nenhum código inserido na definição de qualquer dessas propriedades porque isso não é relevante no momento. Depois que esse código estiver posicionado, sem nenhum esforço adicional, poderemos ver o efeito da instrução `Inherits`.

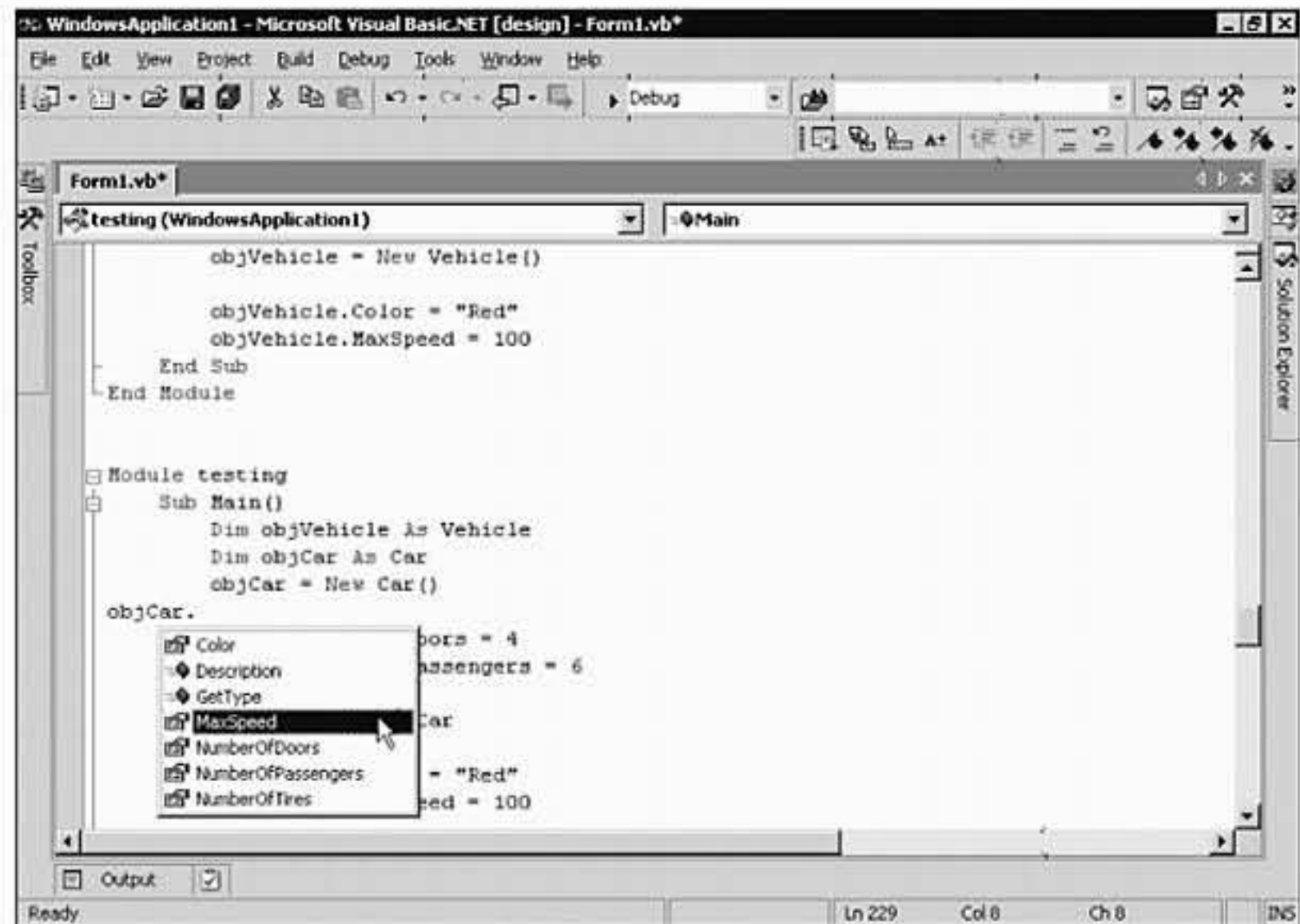
Voltando a nosso procedimento `Main()`, podemos criar um objeto do tipo `Car`, e logo veremos ele expor tanto suas propriedades quanto as da classe-pai (veja a Figura 7.3).

Quando uma classe herdada adiciona novos métodos ou propriedades, diz-se que a classe básica *está-se estendendo*. Além da extensão, também é possível que uma classe-filha *sobreponha* alguma ou toda a funcionalidade da classe básica. Isso acontece quando a classe-filha implementa um método ou propriedade que também é definido na classe-pai ou básica. Nesse caso, o código da classe-filha será executado em vez do código do pai, permitindo que você crie versões especializadas da propriedade ou método básico.

Para uma classe-filha sobrepor alguma parte da classe básica, essa parcela deve ser marcada com `Overridable` na definição da classe básica. Por exemplo, na versão de `Vehicle` listada anteriormente, nenhuma de suas propriedades possuía a palavra-chave `Overridable` e, portanto, as classes-filhas não poderiam fornecer suas próprias implementações. Em uma demonstração de como a sobreposição é configurada nas classes básica e filha, o código da Listagem 7.12 marcará a função `Description()` como podendo ser sobreposta e, em seguida, a sobreporá na classe-filha `Car`. Observe que as partes não relevantes das duas classes foram removidas para dar maior clareza.

FIGURA 7.3

As classes expõem todas as suas propriedades e métodos públicos, além dos da classe em que estão baseadas.

**CÓDIGO****LISTAGEM 7.12** Usando as Palavras-Chave Overridable e Overrides

```

1 Public Class Vehicle
2
3     'Código removido para fins de simplificação....
4
5     Public Overridable Function Description() As String
6         Return "Essa é minha descrição genérica de veículo!"
7     End Function
8 End Class
9
10 Public Class Car
11     Inherits Vehicle
12
13     'Código removido para fins de simplificação....
14
15     Public Overrides Function Description() As String
16         Return "Essa é a descrição do meu carro"
17     End Function
18
19 End Class

```

Quando sobrepuser um método ou propriedade, como fizemos na Listagem 7.12, você poderá se referir novamente ao membro original da classe básica usando o objeto interno MyBase. Por exemplo, para se referir ao método Description() existente na classe Vehicle, poderíamos chamar MyBase.Description() de dentro do método Description() de Car. Esse recurso permite o

fornecimento de uma funcionalidade adicional podendo-se empregar a sobreposição sem que depois seja necessário recompor todo o código original.

Além de marcar o código como `Overridable`, também é possível marcar um método ou propriedade como `MustOverride` e uma classe como `MustInherit`. A palavra-chave `MustOverride` indica que toda filha dessa classe deve fornecer sua própria versão dessa propriedade ou método, e a palavra-chave `MustInherit` significa que essa classe não pode ser usada sozinha (você deve basear outras classes nela). É importante observar que se uma classe tiver um método marcado com `MustOverride`, então, ela própria deve ser marcada com `MustInherit`.

A herança é um tópico extenso e não o abordamos completamente, mas com as informações fornecidas nesta lição, você estará pronto para começar a projetar alguns aplicativos que se beneficiem desse recurso dos objetos.

A Base de Todas as Classes Básicas

Se você examinar a lista do que é exposto por essa nova instância da classe `Car`, verá mais do que as propriedades de `Vehicle` e de `Car`. Os métodos `ToString()` e `GetType()` são expostos por esse objeto, mas não fazem parte dessa classe ou de sua classe-pai. Esses métodos são na verdade outro resultado da herança. Enquanto `Car` herda características de `Vehicle`, tanto `Vehicle` quanto `Car` (e todas as outras classes da plataforma .NET) herdam características da classe básica `System.Object`. Essa classe básica definitiva fornece alguns métodos que automaticamente farão parte de toda classe que criarmos.

Uma consequência adicional da herança, que decerto vale a pena mencionar, se percebe na área dos tipos de dados. Como discutimos no Dia 3, toda variável se enquadra em algum tipo de dado, e os objetos não são exceções. Quando declaramos uma variável como do tipo `Car`, isso é tão restrito quanto a tipificação de dados como inteiros e strings. Se criarmos o parâmetro de uma função com esse tipo, então, apenas esse tipo de objeto poderá ser passado para ela. Em uma situação de herança, a classe-filha atua como se fosse uma instância da classe-pai. Isso significa, em nosso exemplo, que podemos inserir nossos objetos `Car` em argumentos de procedimentos e variáveis que sejam do tipo `Vehicle`. A Listagem 7.13 mostra um exemplo.

CÓDIGO

LISTAGEM 7.13 Uma Classe-Filha Agindo como uma Instância da Classe-Pai

```
1 Sub Main()  
2     Dim objVehicle As Vehicle  
3     Dim objCar As Car  
4     objCar = New Car()  
5  
6     objCar.NumberOfDoors = 4  
7     objCar.NumberOfPassengers = 6  
8  
9     objVehicle = objCar
```


CÓDIGO**LISTAGEM 7.13** Uma Classe-Filha Agindo como uma Instância da Classe-Pai (*continuação*)

```
10
11  objVehicle.Color = "Red"
12  objVehicle.MaxSpeed = 100
13 End Sub
```

A instância de Car representada por objCar foi facilmente inserida na variável objVehicle, e desse ponto em diante poderia ser tratada como um objeto Vehicle. Esse fato, de que um objeto-filho pode ser usado como se fosse uma instância da classe-pai, permite a criação de um código genérico que funcionará com uma classe ou qualquer uma de suas subclasses. Essa é uma das maneiras como a herança pode ser empregada para gerar soluções melhores, mas há muitas outras. Apresentei uma visão geral do assunto herança, e ele merece uma discussão adicional tanto do ponto de vista do projeto quanto da implementação. Por essa razão, esse tópico e outras abordagens relacionadas a objetos serão examinados com mais detalhes no restante do livro.

Construtores

Quando você deseja usar um objeto, tem de trabalhar com uma instância existente ou criar a sua. A instância de um objeto é gerada com a palavra-chave New, que emprega a classe especificada e estabelece uma área na memória para a instância dessa classe. Os *construtores* são uma maneira de fornecer informações para essa classe no momento em que é gerada para permitir que ela mesma se inicialize ou execute outras tarefas de configuração nessa hora. Se uma classe for um construtor, e muitas classes do .NET Framework são, então, em geral é possível fornecer parâmetros no momento da criação, como parte da chamada a New. O código a seguir mostra esse trabalho na criação de um novo objeto de exceção (veja o Dia 6 para obter mais informações sobre as exceções e outros tópicos de tratamento de erros), fornecendo uma mensagem de erro como parâmetro para seu construtor. Essa mensagem de erro será usada de modo automático pelo novo objeto para preencher uma das propriedades.

```
Dim exError As System.Exception
Dim sMessage As String
sMessage = "Essa será a mensagem de erro."
exError = New System.Exception(sMessage)
```

A criação de um construtor para nossa classe Vehicle é relativamente fácil. Primeiro, você precisa gerar um método chamado New que seja público e não tenha parâmetros.

```
Public Sub New()
End Sub
```

Após concluir a criação desse construtor (isso ainda não produz nenhum resultado), você verá pouca diferença em seu código, e até que adicionemos alguma funcionalidade a essa sub-rotina New(), nada diferente acontecerá. O construtor, mesmo sem parâmetros, pode ser usado como

um local para inicializar variáveis internas, como a data de produção de nossa classe `Vehicle` (veja a Listagem 7.14).

CÓDIGO**LISTAGEM 7.14** Usando um Construtor para Inicializar os Membros de uma Classe

```
1 Public Sub New()  
2     m_dtManufactured = System.Date.Now()  
3  
4 End Sub
```

Exatamente como com qualquer outro método de um objeto, é possível sobrepor esse e fornecer mais de uma maneira de chamá-lo. Ao contrário dos outros métodos, no entanto, sobrepor o construtor não requer o uso da palavra-chave `Overloads`. Você pode apenas criar múltiplas versões do procedimento `New`, e cada uma será tratada como uma versão disponível do construtor. Em nosso caso, poderíamos rapidamente gerar alguns construtores úteis (mostrados na Listagem 7.15) considerando as diferentes maneiras nas quais alguém pode querer inicializar nosso objeto.

CÓDIGO**LISTAGEM 7.15** Sobrepondo o Construtor de uma Classe para Fornecer Maneiras de Inicializar Objetos

```
1 Public Sub New()  
2     m_dtManufactured = System.Date.Now()  
3  
4 End Sub  
5  
6 Public Sub New(ByVal sColor As String)  
7     m_dtManufactured = System.Date.Now  
8     m_sColor = sColor  
9  
10 End Sub  
11  
12 Public Sub New(ByVal dtManufactured As Date, _  
13     ByVal sColor As String)  
14     m_dtManufactured = dtManufactured  
15     m_sColor = sColor  
16  
17 End Sub
```

No caso de uma classe-filha, aquela que herda características de outra classe, você pode querer chamar o construtor da classe básica de seu procedimento `New`. Isso pode ser feito facilmente com o uso do objeto especial `MyBase`, empregando um código como `MyBase.New()`.

Espaços de Nome

O *espaço de nome* é um conceito abstrato usado para agrupar vários módulos ou classes, permitindo que você categorize logicamente todos esses objetos dentro de um único objeto de nível superior. Portanto, tendo `Namespace Chapter7` no início de nossas classes e `End Namespace` no final, criamos de modo efetivo um agrupamento chamado `Chapter7`, que contém todas as nossas classes dentro dele. Depois que esse espaço de nome existir, será usado como padrão para referências a objetos feitas em um código dentro do mesmo espaço de nome, mas também poderá ser declarado explicitamente (`Dim objCar as Chapter7.Car`).

Há muitas razões pelas quais você pode querer criar e usar espaços de nome, e não são poucas delas a base do ‘espaço de nome’ identificador, como uma maneira de gerar uma área privada para assegurar que os nomes de sua classe sejam exclusivos. Pela definição de um espaço de nome, nossa classe `Car` se torna `Chapter7.Car`, e, portanto, não entrará mais em conflito com nenhuma outra classe criada com o nome `Car`.

Outra razão mais sutil para usar os espaços de nome é porque produzem um código em que a manutenção é mais fácil. O agrupamento de classes sob espaços de nome de nível superior resulta em um código que é definido claramente por algum esquema de categorização sendo, portanto, mais legível e sua manutenção mais simples.

Em nossos exemplos no decorrer deste livro, poderíamos ter criado espaços de nome com base no título da lição (por exemplo, o código integral desta lição seria inserido sob o espaço de nome `Dia7`), podendo todos eles, em seguida, ser posicionados sob um espaço de nome mais abrangente com relação ao livro chamado `AprendaVB`. Nossas classes poderiam ser criadas apenas como `Dia7`, mas para assegurar que não houvesse ambigüidades, também poderíamos nos referir a elas com seu nome totalmente qualificado (`AprendaVB.Dia7.Car`, por exemplo).

Esse método de agrupar classes é semelhante ao conceito de escopo que foi discutido no `Dia 3`; uma classe só tem de ser exclusiva dentro de seu espaço de nome específico. Se por acaso você criar uma classe que compartilhe seu nome com outra que exista em um espaço de nome diferente, então, precisará se certificar de especificar o nome completo da classe sempre que fizer referência a ela de fora de seu espaço de nome.

Os espaços de nome são hierárquicos, o que permite que você crie um esquema com vários níveis para agrupar suas classes e objetos, exatamente como dentro do próprio .NET Framework. Há duas maneiras de criar um espaço de nome de nível inferior. Defina-o usando o nome totalmente qualificado (veja a Listagem 7.16) ou aninhe as definições dos espaços de nome (veja a Listagem 7.17).

CÓDIGO

LISTAGEM 7.16 Declarando um Espaço de Nome com Várias Partes

```
1 Imports System
2 Namespace MyApp.Console
3     Module Main
4         Sub Main()
```


CÓDIGO**LISTAGEM 7.16** Declarando um Espaço de Nome com Várias Partes

```
5         Dim objHW As New MyApp.Console.Utilities()
6         objHW.PrintOut()
7     End Sub
8 End Module
9
10    Public Class Utilities
11        'Run the application
12        Public Sub PrintOut()
13            Console.WriteLine(Environment.MachineName)
14            Console.WriteLine(Environment.SystemDirectory)
15            Console.WriteLine(Environment.GetLogicalDrives())
16            Console.WriteLine(Environment.Version.ToString())
17        End Sub
18    End Class
19 End Namespace
```

CÓDIGO**LISTAGEM 7.17** Usando Espaços de Nome Aninhados para Criar Hierarquias de Objetos

```
1 Imports System
2 Namespace MyApp
3     Namespace Console
4         Module Main
5             Sub Main()
6                 Dim objHW As New MyApp.Console.Utilities()
7                 objHW.PrintOut()
8             End Sub
9         End Module
10
11        Public Class Utilities
12            'Run the application
13            Public Sub PrintOut()
14                Console.WriteLine(Environment.MachineName)
15                Console.WriteLine(Environment.SystemDirectory)
16                Console.WriteLine(Environment.GetLogicalDrives())
17                Console.WriteLine(Environment.Version.ToString())
18            End Sub
19        End Class
20    End Namespace
21 End Namespace
```


Em seus aplicativos, você pode usar os espaços de nome como uma maneira de agrupar códigos conceitualmente relacionados, mas além de seu efeito no escopo das classes, eles não são realmente necessários na construção de um sistema.

Membros e Objetos Compartilhados

No começo desta lição, quando descrevemos o relacionamento entre as classes e os objetos, você aprendeu que para usar qualquer propriedade ou método especificado em uma classe teria de empregar ou criar uma instância dessa classe. Em geral, é isso que acontece; não é possível trabalhar diretamente com uma classe, apenas com as instâncias dela que forem criadas. Há uma maneira de expor uma certa funcionalidade por meio da própria classe, no entanto, independentemente de qualquer instância particular dela, com o uso da palavra-chave `Shared`. Essa palavra-chave, como os outros descritores que foram vistos (como `Public`, `Private` e `Friend`), significa que parte de uma classe está disponível sem a criação de uma instância. O .NET Framework usa esse recurso em suas classes, como a propriedade exposta `Today` da classe `System.DateTime` (cujo exemplo é mostrado na Listagem 7.18).

CÓDIGO

LISTAGEM 7.18 Funções Internas do Visual Basic 6.0 Foram Substituídas pelos Métodos Compartilhados no Visual Basic .NET

```
1 Imports System
2 Module Main
3     Sub Main()
4         Dim dtToday As Date
5         dtToday = DateTime.Today()
6     End Sub
7 End Module
```

Esses membros compartilhados podem ser criados em seu próprio código quando você quiser que um método ou propriedade em particular esteja sempre acessível, sem a sobrecarga de gerar um objeto. Em geral, no entanto, evite usar esse recurso a menos que seja realmente necessário porque ter muitos membros compartilhados é quase o mesmo que apenas criar um módulo cheio de procedimentos e reduz o significado de seus objetos. Lembre-se de que `Modules` é um tipo especial de `Class`, todos os membros de um módulo são considerados compartilhados (`Shared`) por padrão.

Resumo

O .NET Framework é uma biblioteca de classes que você pode usar em seus aplicativos, todas elas construídas a partir dos princípios básicos da programação orientada a objetos, comuns às linguagens .NET. Esses princípios (o conceito de classes, instâncias, propriedades, métodos, herança e assim por diante) também serão a base da criação das classes de sua autoria dentro do Vi-

sual Basic .NET. Para desenvolver qualquer aplicativo que vá além do exemplo mais simples, será necessário empregar o .NET Framework e, com frequência, também será preciso gerar suas próprias classes, tornando a utilização de objetos uma habilidade essencial para todos os programadores da plataforma .NET. Nesta lição abordamos, por meio da criação de nossas classes como exemplo, conceitos básicos orientados a objetos e mostramos como implementá-los em um código de sua autoria. Na próxima lição, Dia 8, introduziremos as classes disponíveis no .NET Framework e ilustraremos como empregá-las para desenvolver poderosos aplicativos .NET.

P&R

- P Nunca trabalhei com uma linguagem orientada a objetos antes; posso ficar apenas nos procedimentos e módulos?**
- R** Você não precisa criar objetos no Visual Basic .NET, mas achará difícil usar essa linguagem sem empregar pelo menos alguns deles, como o .NET Framework. Por essa razão, o enfoque inicial para um desenvolvedor que não conheça objetos é se familiarizar com sua utilização antes de passar para sua criação.
- P Li que, no Visual Basic .NET, não é possível dizer exatamente quando um objeto será eliminado e que esse é um fator limitante por várias razões. O que significam esses comentários, e eles são verdadeiros?**
- R** Quando se trabalha com objetos, sua eliminação nem sempre ocorre em um ponto definido do código, portanto, será difícil fazer com que um certo trecho desse seja executado sempre que um objeto for destruído. Em algumas linguagens, incluindo as versões anteriores do Visual Basic, havia um mecanismo disponível para adicionar um código ao procedimento de um ‘evento de eliminação’. O código inserido em um procedimento desse tipo teria sua execução garantida sempre que o objeto fosse eliminado. No Visual Basic .NET, o subsistema de coleta de lixo eliminará os objetos quando necessário, mas não é possível criar um código que seja processado na hora da destruição. Em geral, essa ‘limitação’ não causa muitos problemas, mas às vezes (principalmente quando se transfere um código antigo do Visual Basic) há a expectativa da execução de um código no final do ciclo de vida do objeto, e isso não é facilmente transferido para uma implementação do Visual Basic .NET.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Se uma classe tiver um método marcado com `MustOverride`, o que mais acontecerá?
2. Qual a palavra-chave que você pode usar em um construtor para chamar o de uma classe básica?
3. O que está errado nos construtores sobrepostos desse código (veja a Listagem 7.19)?

CÓDIGO**LISTAGEM 7.19** Versões Diferentes de Construtores Disponíveis para a Classe `Vehicle`

```
1 Public Class Vehicle
2     Dim m_dtManufactured As Date
3     Dim m_lngNumberOfPassengers As Long
4     Dim m_sColor As String
5
6     Public Sub New()
7         m_dtManufactured = System.DateTime.Now()
8     End Sub
9
10    Public Sub New(ByVal sColor As String)
11        m_dtManufactured = System.DateTime.Now()
12        m_sColor = sColor
13    End Sub
14
15    Public Sub New(ByVal sName As String)
16    End Sub
17
18    Public Sub New(ByVal dtManufactured As Date, _
19                  ByVal sColor As String)
20        m_dtManufactured = dtManufactured
21        m_sColor = sColor
22    End Sub
23
24 End Class
```

Exercícios

Por meio da herança, das classes, das propriedades e dos métodos, crie um exemplo de hierarquia de classes começando em `Animal` e descendo alguns níveis com algumas classes ilustrativas.

SEMANA 1

Revisão

Agora que você terminou sua trajetória pela primeira semana de lições, já deve ter uma boa compreensão da programação, do Visual Basic .NET e de como tudo se encaixa. (Dias 1 e 2). Também criou alguns exemplos de programas na plataforma .NET, usando tanto o IDE quanto a linha de comando para compilá-los. Com essa experiência, já pode testar todos os exemplos deste livro e até criar os seus.

A lição sobre os tipos de dados e as técnicas básicas de programação usadas na criação de aplicativos do Visual Basic .NET (Dia 3) forneceu-lhe uma boa introdução à programação simples. O Dia 4 apresentou as instruções condicionais (inclusive as instruções If) e laços, dois itens que serão empregados em quase todos os programas que vier a escrever. Pelo fato de o Visual Basic .NET permitir a geração de tantos tipos diferentes de programas, por meio de uma grande variedade de projetos e diversos arquivos-fonte, o Dia 4 abordou as opções de arquitetura para desenvolver todas as espécies distintas de sistemas que a plataforma .NET pode ser usada para criar.

Independentemente da sua habilidade em programação, erros sempre ocorrem em programas. Não importando se eles poderiam ser evitados, as técnicas de tratamento de erros abordadas no Dia 6 permitem que você lide com o problema de maneira sofisticada. Para concluir, já que grande parte do .NET Framework é baseada em objetos, o enfoque da lição do Dia 7 foi sobre os termos e conceitos necessários para possibilitar o uso de objetos em seus próprios programas.

Todas essas informações, bem como a prática da codificação que as acompanhou, devem tê-lo preparado para encarar o primeiro projeto como bonificação (The Game of Life – O Jogo da Vida). Esse projeto, que você pode encontrar acessando a página Web deste livro no site www.makron.com.br (veja a Introdução para maiores detalhes), além de conduzi-lo na criação de um programa real completo, ilustra o uso de instruções condicionais, arrays, laços, variáveis e até técnicas de tratamento de erros mostradas no decorrer das lições que acabamos de concluir. Leia o material, desenvolva o projeto e faça testes com os resultados durante algum tempo. Quando estiver pronto para seguir em frente, tente modificar o código para alterar como o programa funciona, criando a sua própria variação dele.

SEMANA 2

8

Visão Geral

Durante esta semana, você se aprofundará realmente no estimulante mundo do .NET Framework. Ela começa com uma lição no próprio Framework (Dia 8), incluindo detalhes sobre o uso das classes `Console.Math.Random` e `Environment`, e informações sobre toda a excelente funcionalidade das listas e arrays fornecida pelo Framework. Além de todas essas particularidades, essa lição também inclui uma introdução a como encontrar o recurso necessário no .NET Framework por sua própria conta – informações valiosas quando você iniciar projetos de sua autoria usando a plataforma .NET.

As próximas duas lições, Dias 9 e 10, enfocam a criação de uma interface com o usuário em seus aplicativos, para Windows (usando os formulários Windows, Dia 9) ou para a Web (usando a tecnologia dos formulários da Web, abordada no Dia 10). Poucos programas são executados sem alguma interface com o usuário, o que torna essas duas lições absolutamente essenciais.

Quase todo aplicativo empresarial existente usa algum tipo de banco de dados, portanto, você passará duas lições examinando esse tópico. O Dia 11 aborda os conceitos subjacentes aos bancos de dados e o conduz pela criação de um deles para um aplicativo de registro de CDs. O Dia 12 dá prosseguimento à lição sobre dados mostrando como se conectar ao banco de dados de seu exemplo e desenvolver um aplicativo que permita adicionar, editar e excluir CDs de sua coleção.

O trabalho com os bancos de dados introduzirá você no Server Explorer, mas o Dia 13 mostrará como usar seus vários recursos. Nessa lição, trabalharemos com contadores de desempenho e registros de eventos, e aprenderemos uma maneira mais fácil de configurar uma conexão de banco de dados.

9

10

11

12

13

14

Para concluir, no Dia 14, você aprenderá alguns dos mais avançados recursos orientados a objetos do Visual Basic .NET, incluindo a herança e a sobreposição. Essa lição realmente tornará mais claro o poder dessa versão do Visual Basic e mostrará como é possível usar esses recursos para projetar sistemas que sejam mais fáceis de estender e tenham uma manutenção mais simples.

No final da Semana 2, você terá aprendido tantos recursos do Visual basic .NET que estará pronto para criar aplicativos complexos. O projeto de bonificação número 2 (Parte 1 do Projeto de Bônus 2) fornecerá a prática por meio de algumas dessas técnicas novas.

Semana 2

DIA 8

Introdução ao .NET Framework

O .NET Framework não é especificamente um novo recurso do Visual Basic .NET porque ele é compartilhado com muitas linguagens (mais de 20 no momento em que esse texto foi escrito). O Framework fornece um poderoso conjunto de componentes para você usar em seus programas. Ele contém uma quantidade imensa de classes que executam várias funções, da manipulação de strings ou números à criptografia e o acesso à rede. Assimilar o .NET Framework é um processo contínuo, mas conhecer os termos-chave e a maneira como o Framework é organizado são conhecimentos valiosos que o ajudarão no desenvolvimento de mais aplicativos com o Visual Basic .NET.

Hoje, você aprenderá:

- O que é o .NET Framework.
- Algumas classes importantes do .NET Framework.
- Com o encontrar o que precisa no .NET Framework.

O Que É o .NET Framework?

O termo *.NET Framework* foi o nome dado a vários componentes e serviços que foram combinados para criar um ambiente poderoso de desenvolvimento. Ele inclui uma grande quantidade de classes (mais de 6 mil) que fornecem grande parte da funcionalidade anteriormente existente no Visual Basic ou na API do Windows. Isso permitirá que você escreva aplicativos com base no Windows e na Web, acesse a rede, crie elementos gráficos e muito mais.

Além disso, o .NET Framework possui o *Common Language Runtime (CLR)*, que é responsável por executar seu código. O CLR oferece várias inovações importantes que afetam os desenvolvedores de aplicativos .NET no Visual Basic .NET ou em outras linguagens que tenham suporte. A principal alteração é que agora todas essas linguagens são compiladas para a Microsoft Intermediate Language (MSIL). Em seguida, ela é convertida pelo CLR em código nativo quando é executada pela primeira vez. O resultado final é que você se beneficiará do desempenho de um código totalmente compilado, e não de um que seja interpretado no tempo de execução.

Ainda há mais, todas as linguagens que têm suporte do CLR usam o mesmo tipo de dados. Isso significa que é muito mais fácil para duas (ou mais) linguagens interoperarem. Anteriormente, se você precisasse passar informações de uma linguagem (como a C++) para outra (digamos o Visual Basic), poderia ter de executar algumas conversões para interpretar de modo apropriado a variável. Agora, já que as duas linguagens empregarão o mesmo tipo de dados, isso é simples. O CLR permite que os desenvolvedores utilizem a linguagem com a qual se sentem mais confortáveis, podendo ainda se comunicarem com outras.

Embora o CLR seja importante, ele em geral funciona em segundo plano. A parte essencial e visível do .NET Framework que você deve examinar são algumas das classes que o compõem.

Classes Importantes do .NET Framework

Você usará muitas classes do .NET Framework quando desenvolver aplicativos no Visual Basic .NET. No entanto, empregará algumas delas com mais frequência do que outras. Entre as classes mais úteis estão

- **Console** Permite a leitura e a exibição na linha de comando.
- **Environment** Permite a leitura e a gravação nas variáveis de ambiente do sistema.
- **Random** Permite a geração de números aleatórios.
- **Math** Inclui vários cálculos matemáticos.
- **Collections** Fornece várias classes para diferentes estratégias de armazenamento de conjuntos de itens.

Console

Você já viu algumas das classes **Console** quando desenvolveu aplicativos anteriormente. Os métodos **WriteLine** e **ReadLine** dessa classe têm sido usados com frequência para exibir resultados e ler códigos de vários dos aplicativos do console, ou de linha de comandos, criados no livro até agora. Essa é a utilização mais importante da classe **Console**. No entanto, há alguns recursos menos empregados dela que vale a pena examinar. Alguns dos métodos essenciais desta classe são mostrados na Tabela 8.1.

TABELA 8.1 Métodos da Classe Console

<i>Método</i>	<i>Descrição</i>
Read	Lê informações da linha de comandos (ou outro código). Não precisa que a linha termine com Enter.
ReadLine	Lê informações da linha de comandos. Lê todos os caracteres até o Enter, sem incluí-lo.
SetError	Altera o destino das mensagens de erro a serem exibidas enquanto seu programa estiver em execução. Pode ser usado na criação de um mecanismo simples de registro de erros em seu aplicativo.
SetIn	Altera a origem da entrada de Read e ReadLine. Pode ser usado na alteração de um aplicativo de linha de comandos para que leia um arquivo ou local de rede. Veja a seção “Redirecionamento”, mais à frente nesta lição.
SetOut	Altera o destino dos métodos Write e WriteLine. Pode ser usado na alteração do destino da saída para registro ou outra finalidade. Veja a seção “Redirecionamento”, mais adiante nesta lição.
Write	Exibe informações na linha de comandos (ou outra saída). Não termina com uma nova linha.
WriteLine	Exibe informações na linha de comandos (ou outra saída). Termina a saída com uma linha nova.

Resultados Mais Simples

Já que a classe Console com frequência é usada para saídas, seria bom se tornasse sua vida mais fácil quando você executasse essa tarefa. Felizmente, ela faz isso. Quando empregamos os métodos Write ou WriteLine para exibir informações, nos beneficiamos de sua capacidade de utilizar espaços reservados para as variáveis. Em geral, quando se quer exibir uma string que contém informações armazenadas em variáveis, uma técnica conhecida como concatenação é usada.

NOVO TERMO

Concatenação é mais uma palavra sofisticada que quer dizer ‘combinação de strings’. Em vez de somar as strings com o símbolo +, como fazemos com os números, é usado o símbolo &, como em

```
Console.WriteLine("Insira" & ITEM_COUNT & _  
  
"itens. Pressione ENTER entre os itens.")
```

Se você tiver uma string complicada, pode ser difícil lidar com todas as aspas e E comerciais. Em vez de construir uma string complexa usando a concatenação, é possível utilizar esses espaços reservados nos locais certos e incluir posteriormente as variáveis que serão empregadas em seu preenchimento, como descrito a seguir:

```
Console.WriteLine("Insira {0} itens. Pressione ENTER entre os itens.",  
ITEM_COUNT)
```

O item {0} é um espaço reservado para a variável que você incluiu no final da string. Ela é a variável ITEM_COUNT das instruções anteriores. É possível inserir vários espaços reservados (ou até

mesmo usar o mesmo diversas vezes) na string. Assim, as variáveis que forem incluídas depois da string serão inseridas nela em ordem, começando com o item “0” (primeiro item) e dando continuidade, percorrendo as variáveis e espaços reservados em sequência ({0}, {1}, {2} e assim por diante). A Listagem 8.1 demonstra esse recurso.

CÓDIGO**LISTAGEM 8.1** Usando a Classe Console para Entradas e Resultados

```
1 Imports System
2 Public Class ConsoleTest
3     Private Const ITEM_COUNT As Integer = 10
4     Shared Sub Main()
5         Dim I As Integer
6         Dim sItems(ITEM_COUNT) As String
7         Console.WriteLine("Insira {0} itens." & _
8             Pressione ENTER entre os itens.", ITEM_COUNT)
9         For I = 0 To ITEM_COUNT-1
10             sItems(I) = Console.ReadLine
11         Next
12         Console.WriteLine()
13         Console.WriteLine("Itens em ordem inversa:")
14         For I = ITEM_COUNT - 1 To 0 Step -1
15             Console.WriteLine(sItems(I))
16         Next
17     End Sub
18 End Class
```

ANÁLISE

O código da Listagem 8.1 é bem simples e pretende demonstrar apenas algumas das maneiras como você tem usado a classe Console nessas sete lições.

A linha 1 importa o espaço de nome System. É nesse local que a classe Console é definida, assim como muitas outras classes importantes. Portanto, você deve sempre importar esse espaço de nome na maioria dos aplicativos. O IDE do Visual Basic .NET fará isso automaticamente para aplicativos que forem criados quando ele estiver em uso.

Dentro da única classe definida em seu arquivo, você declara apenas uma constante, ITEM_COUNT (linha 3), e uma sub-rotina Main (linha 4). A constante é usada para substituir a inserção do número 10 em todo o código, prevendo o caso de se querer alterar o valor posteriormente. É muito mais fácil alterar só o valor da constante, em vez de ter de procurar todas as ocorrências desse valor no aplicativo inteiro. O uso de uma constante também mostra que o desenvolvedor sabe que os valores estão relacionados. Se tivéssemos repetido o número 10 em todos os locais e examinássemos esse código seis meses depois, poderíamos não lembrar se todos indicariam o mesmo valor ou se seria somente uma coincidência.

O procedimento `Shared Sub Main` é, como você já viu muitas vezes, o primeiro método executado em um aplicativo. Dentro do procedimento, você declara um array de strings (linha 6) e o preenche com o método `Console.ReadLine`.

A linha 7 é digna de nota. Dentro do item exibido com `Console.WriteLine` (ou `Console.Write`), você pode inserir vários *caracteres*, como o item `{0}` no meio da string da linha 7. Posteriormente, cada uma dessas linhas será preenchida com variáveis. Essas são incluídas depois da string na chamada de `Console.WriteLine`, como fizemos com a constante `ITEM_COUNT`.



NOTA

Usar a abordagem de caracteres na construção de uma string pode ser útil, mas, se fará ou não sentido dependerá um pouco de critérios pessoais. Nessa técnica, cada valor substituirá um caractere na ordem que eles aparecerem, começando com 0 que é o primeiro. As variáveis podem ser numéricas, alfanuméricas ou qualquer outro tipo que possa ser convertido em uma string. Por exemplo, se você quisesse usar essa técnica para escrever uma string com quatro variáveis incluídas, ela teria a seguinte aparência:

```
Console.WriteLine("{0}+{1} = {2} : {3}", 1, 1, 2, _
    "Bertrand Russell, 1912")
```

Os desenvolvedores experientes do Visual Basic podem estar um pouco mais familiarizados com uma abordagem alternativa. Nela você constrói a string concatenando strings e variáveis. A string anterior poderia ser construída e exibida com o código:

```
Console.WriteLine(value1 & "+" & value2 & "=" & result & ":" &
    sSource)
```

Embora isso possa parecer mais 'natural', você pode esquecer de incluir um caractere de concatenação ("&") quando adicionar as strings. Seu programa falharia na compilação ou na execução.

Cada linha da entrada é lida (linha 10) e exibida em um dos elementos do array. Observe que esses elementos são numerados de 0 a 9 (linha 9).

Para concluir, os elementos são exibidos na tela em ordem inversa. Isso é feito com o uso da cláusula `Step` da instrução `For...Next` (linha 14). Eles são exibidos um por vez na saída atual, como vemos na linha 15. A Listagem 8.2 demonstra esse tipo de saída.

CÓDIGO/ RESULTADO

LISTAGEM 8.2 Executando o Aplicativo do Console

- 1 [c:\code]Console1.exe
- 2 Insira 10 itens. Pressione ENTER entre os itens.
- 3 Porco-da-terra
- 4 Rato gigante da Índia
- 5 Casuar
- 6 Dugão
- 7 Équidna
- 8 Tentilhão

**CÓDIGO/
RESULTADO****LISTAGEM 8.2** Executando o Aplicativo do Console (*continuação*)

```
9 Girafa
10 Hipopótamo
11 Iguana
12 Chacal

13 Itens em ordem inversa:
14 Chacal
15 Iguana
16 Hipopótamo
17 Girafa
18 Tentilhão
19 Équidna
20 Dugão
21 Casuar
22 Rato gigante da Índia
23 Porco-da-terra
```

Redirecionamento

Redirecionamento significa apenas ‘enviar algo para um local diferente’. Quando lidamos com a classe `Console`, isso quer dizer que você pode alterar o local onde fará a leitura ou exibição. Por exemplo, ela poderia ser usada para exibir o resultado de um aplicativo que recebesse entradas de linhas de comandos ou de um arquivo. De maneira semelhante, você poderia empregar `Console.WriteLine` quando depurasse o aplicativo em testes e, em seguida, o redirecionasse a um arquivo a fim de criar uma ferramenta simples de registro para ele.

Você pode redirecionar a entrada, o resultado ou as informações de erro de seu programa. Essas informações podem ser redirecionadas para qualquer destino onde houver `TextReader` (para entradas) ou `TextWriter` (para saídas e erros). Observe que não é possível criar nenhuma dessas duas classes diretamente, já que são abstratas. Em vez disso, é preciso gerar uma das classes que implementará ou herdará características de `TextReader` ou `TextWriter` (veja o Dia 7, “Trabalhando com Objetos” e o Dia 14, “Introdução à Programação Orientada a Objetos”, para obter mais detalhes sobre a abstração e a herança). Algumas das implementações dessas duas classes permitem ler e gravar em arquivos, locais na rede e assim por diante. A Listagem 8.3 mostra como se pode alterar o exemplo mostrado na Listagem 8.2 para que envie o resultado para um arquivo em vez de para a janela das linhas de comando. Observe que os comandos de `Console.WriteLine` não foram alterados; somente o destino foi.

CÓDIGO**LISTAGEM 8.3** Redirecionando a Saída do Console

```
1 Imports System
2 Imports System.IO
```


CÓDIGO**LISTAGEM 8.3** Redirecionando a Saída do Console (*continuação*)

```
3 Public Class ConsoleTest
4     Private Const ITEM_COUNT As Integer = 10

5     Shared Sub Main()
6         Dim I As Integer
7         Dim sItems(ITEM_COUNT)As String
8         Dim oFile As TextWriter = File.CreateText("Output.txt")
9         Dim oOut As TextWriter = Console.Out

10        Console.WriteLine("Insira {0} itens. Pressione ENTER entre os
        ➡itens.", ITEM_COUNT)
11        For I = 0 To ITEM_COUNT-1
12            sItems(I) = Console.ReadLine
13        Next

14        Console.WriteLine()
15        Console.SetOut(oFile)
16        Console.WriteLine("Itens em ordem inversa:")
17        For I = ITEM_COUNT - 1 To 0 Step -1
18            Console.WriteLine(sItems(I))
19        Next

20        oFile.Close()

21        Console.SetOut(oOut)
22        Console.WriteLine("Concluído")
23        Console.ReadLine()

24    End Sub
25 End Class
```

Executando o aplicativo:

```
1 [c:\code]Console2.exe
2 Insira 10 itens. Pressione ENTER entre os itens.
3 Porco-da-terra
4 Rato gigante da Índia
5 Casuar
6 Dugão
7 Équidna
8 Tentilhão
9 Girafa
10 Hipopótamo
11 Iguana
12 Chacal
```


13

14 Concluído

ANÁLISE

A única alteração entre as Listagem 8.1 e 8.2 foi o acréscimo das linhas 2, 8, 9, 15, 20, 21 e 22. Essas são as linhas que criam o arquivo, e provocam e encerram o redirecionamento da saída. A linha 2 importa o espaço de nome `System.IO`. Ele será necessário já que contém a definição para as classes `TextWriter` e `File` que você usará depois. A linha 8 gera o arquivo `Output.txt` empregando o método `CreateText` da classe `File`. Observe que na verdade não é preciso criar uma instância da classe `File`, já que o método `CreateText` é compartilhado. Se já houver um arquivo com esse nome, ele será excluído. Se em vez disso quisermos continuar a preencher um arquivo existente, devemos utilizar `AppendText` no lugar de `CreateText`. A linha 9 é usada para salvar o destino original da saída, portanto ele poderá ser configurado mais uma vez na linha 21. A linha 15 provoca o redirecionamento para seu arquivo. Todas as saídas que empreguem `Console.Write` e `Console.WriteLine` após esse redirecionamento serão enviadas para seu arquivo em vez de para a janela das linhas de comandos. A última linha importante é a 20. Se não conseguirmos fechar o arquivo, o conteúdo não poderá ser gravado e terminaremos com um arquivo sem bytes (vazio).

O código da Listagem 8.3 deve produzir um arquivo `Output.txt` com o conteúdo a seguir:

RESULTADO

```
1 Itens em ordem inversa:
2 Chacal
3 Iguana
4 Hipopótamo
5 Girafa
6 Tentilhão
7 Équidna
8 Dugão
9 Casuar
10 Rato gigante da Índia
11 Porco-da-terra
```

Environment (Ambiente)

Saber seu lugar em geral produz a diferença entre sucesso e constrangimento. E essa percepção vem frequentemente da vizinhança, ou do ambiente. De maneira semelhante, em programação, você é envolvido por um ambiente específico. O ambiente em que seus programas são executados é composto de informações sobre o sistema operacional, assim como muitas variáveis e outras configurações que afetam seu computador. Esse ambiente permite que você consulte configurações para o usuário, como o local do diretório de seus arquivos temporários, o conteúdo de seu caminho de pesquisa ou até outros itens sobre a linha de comandos. Algumas das propriedades e métodos mais importantes da classe `Environment` são listados na Tabela 8.2. Observe que todas essas propriedades e métodos são compartilhados.

TABELA 8.2 Métodos e Propriedades da classe Environment

<i>Membro</i>	<i>Descrição</i>
Propriedades	
CommandLine	Representa a linha de comandos completa que iniciou o aplicativo.
CurrentDirectory	Retorna o caminho do diretório atual.
OSVersion	Retorna informações sobre o sistema operacional atual, por exemplo, se é o Windows 9x, o NT ou o 2000.
SystemDirectory	Retorna o caminho do diretório do sistema (\winnt\system32 no Windows NT ou no 2000).
Version	Retorna informações sobre a versão de uma montagem.
Métodos	
Exit	Encerra um aplicativo, opcionalmente retornando um código de erro que pode ser usado pelo sistema operacional.
GetCommandLineArgs	Retorna todos os itens listados na linha de comandos quando o aplicativo foi iniciado. Esse retorno é feito na forma de um array de strings. O próprio executável é o elemento zero (primeiro) do array.
GetEnvironmentVariable	Retorna o valor de uma variável de ambiente solicitada. Traz informações armazenadas no ambiente com o comando Set, como o caminho de pesquisa (path), o diretório para arquivos temporários (temp) ou outras configurações.
GetLogicalDrives	Retorna a lista de unidades disponíveis. Esse retorno é feito na forma de um array de strings. O primeiro (item 0) elemento geralmente é A:\.

A classe Environment é útil quando você precisa de informações sobre o sistema que está ao redor da execução de seu programa, como quando tem de saber onde gravar seus arquivos temporários ou em que sistema operacional está sendo executado.

Random

Random é uma classe simples, destinada a criar números aleatórios, em geral um inteiro (Integer) ou um duplo (Double). Os métodos importantes dessa classe estão listados na Tabela 8.3.

TABELA 8.3 Métodos da Classe Random

<i>Método</i>	<i>Descrição</i>
Next	Retorna um inteiro entre 0 e o valor máximo possível para um inteiro (aproximadamente 2 bilhões).
Next(MaxValue)	Retorna um inteiro entre 0 e o valor de MaxValue (algum inteiro).

TABELA 8.3 Métodos da Classe Random (*continuação*)

<i>Método</i>	<i>Descrição</i>
Next(MinValue, MaxValue)	Retorna um inteiro entre os valores mínimo e máximo. Essa é a variação mais usada quando se precisa de 1 como valor mínimo.
NextDouble	Retorna um tipo duplo entre 0 e 1.

Por exemplo, se você quisesse gerar um inteiro aleatório entre 1 e 100 (inclusive), escreveria

```
Dim oRand As New Random
Dim iValue As Integer = oRand.Next(1,100)
```

Math

A classe Math contém muitas das constantes e funções matemáticas importantes. A maioria delas são métodos compartilhados e, portanto, podem ser usados sem a criação de uma instância da classe Math. Os métodos mais relevantes são descritos na Tabela 8.4. Se você examinar a classe Math na ajuda (Help), verá que há muitos outros métodos disponíveis.

TABELA 8.4 Métodos da Classe Math

<i>Método</i>	<i>Descrição</i>
Abs	Retorna o valor absoluto de um número (se for negativo, ele retornará o valor positivo do número).
Cós	Retorna o cosseno de um ângulo (medido em radianos; veja a Nota a seguir).
E	Retorna um tipo duplo representando o valor de e (2.7182818284590451).
Max	Retorna o valor máximo entre dois valores.
Min	Retorna o valor mínimo entre dois valores.
Pi	Retorna um tipo duplo representando o valor de pi (3.1415926535897931).
Round	Arredonda um número para o número inteiro mais próximo.



NOTA

Para aqueles entre nós que dormiram durante as aulas de geometria, um radiano é aproximadamente igual a 57.296 graus. Por que um número tão redondo? Os radianos estão relacionados à circunferência de um círculo – a medida exata é

$$\# \text{ Radianos} = (\# \text{ graus} * \pi) / 180$$

Os radianos são usados em computação devido aos cálculos e, portanto, estão além do escopo deste livro. Simplesmente lembre-se de converter seus ângulos de graus para radianos antes de empregar qualquer dos métodos de Math relacionados aos ângulos.

A classe `Math` em geral é útil para calcular valores. Mesmo se você não lembrar (ou nunca tiver aprendido) o que é uma tangente hiperbólica, é na classe `Math` que a encontrará.

Classes de Conjuntos no .NET Framework

Você já conheceu o tipo de dados `Array` no Dia 3, “Introdução à Programação com o Visual Basic .NET”. O .NET Framework possui várias outras classes de conjuntos que adicionam mais recursos a `Array`. Essas classes permitem o armazenamento de uma lista de informações exatamente como nos arrays, mas, além disso, elas têm recursos como a ordenação de listas e a possibilidade de incluir e recuperar com maior facilidade na lista. Várias classes de conjuntos (a maioria no espaço de nome `System.Collections`) estão disponíveis. Algumas serão descritas ainda nesta lição. Se precisar de outros tipos de coleções, verifique nesse espaço de nome antes de procurar em outro local.

ArrayList

`ArrayList` é o conjunto que mais se parece com um `Array`. A principal diferença é que foi projetado para permitir um crescimento fácil conforme forem sendo adicionados mais elementos. Como as outras classes do espaço de nome `System.Collections`, `ArrayList` foi criado para armazenar um conjunto de variáveis `Object`. Portanto, pode ser usado para conter qualquer tipo de dado. O uso da classe `ArrayList` será adequado se você tiver um conjunto muito dinâmico que possa aumentar ou diminuir com o passar do tempo e se não precisar dos recursos dos outros conjuntos.

Você pode criar uma nova instância da classe `ArrayList` usando um dos construtores disponíveis. Uma versão permite a definição da capacidade inicial do conjunto, e outra configura o tamanho inicial como 16:

```
Dim arrList As New ArrayList
' cria um novo ArrayList, com 16 membros inicialmente
Dim arrList2 As New ArrayList(20)
' cria um novo ArrayList, com 20 membros inicialmente
Dim arrList3 As ArrayList
Set arrList3 = New ArrayList(52)
' cria um novo ArrayList, com 52 membros inicialmente
```

As outras propriedades e métodos importantes da classe `ArrayList` são usados para adicionar, recuperar ou excluir membros do conjunto. A Tabela 8.5 resume alguns dos principais membros.

TABELA 8.5 Métodos e Propriedades da Classe ArrayList

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Capacity	O tamanho atual da classe ArrayList. Configurado pela primeira vez quando ela é criada (16 por padrão), podendo aumentar conforme mais elementos forem adicionados.
Count	A quantidade efetiva de itens na classe ArrayList.
Item	Retorna um elemento específico da classe ArrayList.
Métodos	
Add	Adiciona um novo item à ArrayList. Se isso fizer com que Count ultrapasse Capacity, essa propriedade será aumentada (com sua quantidade inicial – 16 por padrão).
Clear	Remove todos os itens de ArrayList. Count é configurada com 0, mas Capacity não é alterada.
IndexOf	Retorna a posição de um certo objeto na classe ArrayList. Esse método é útil quando usado depois de uma ordenação.
Insert	Adiciona um novo elemento à ArrayList em uma posição solicitada.
Remove	Remove o objeto solicitado de ArrayList.
RemoveAt	Remove o elemento da posição solicitada.
Sort	Ordena os membros de ArrayList.
ToArray	Copia toda ou parte de uma classe ArrayList em um array.
TrimToSize	Diminui ArrayList de modo que tanto Count quanto Capacity sejam iguais à quantidade atual de elementos na classe.

O uso da classe ArrayList é mais adequado na substituição de arrays quando sabemos que o tamanho aumentará, mas não tivermos noção de até onde o conjunto avançará. Seu crescimento ilimitado permite que os itens sejam adicionados quando necessário.

Queue e Stack

Os conjuntos Queue e Stack são semelhantes. São dois tipos ‘clássicos’ de conjuntos, sendo úteis em muitos esforços de programação. As duas classes são conjuntos que permitirão que você adicione com facilidade novos itens. Em geral removem o item quando é visualizado porque foram projetadas para armazenar o conjunto temporariamente. Elas diferem no modo como os itens são adicionados em cada conjunto e, principalmente, em como são removidos dos conjuntos.

A classe Queue é um conjunto ‘primeiro a entrar, primeiro a sair’ (FIFO, first-in, first-out). Isso significa que os itens são removidos da fila na mesma ordem que foram adicionados. Essa característica é semelhante à da maioria das filas (ou formações em sequência) que você possa ter visto. Em geral, em um ponto de ônibus, na fila do almoço ou na passagem pela alfândega, as pessoas que chegam primeiro são atendidas antes. As filas são frequentemente usadas em pro-

gramação quando esse comportamento é desejado. Por exemplo, quando dois programas (ou objetos) estão se comunicando, as mensagens são colocadas em uma fila para o objeto receptor. Em seguida, ele poderá processá-las na ordem que foram recebidas. A Tabela 8.6 descreve alguns dos métodos e propriedades importantes da classe Queue.

TABELA 8.6 Métodos e Propriedades da Classe Queue

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Count	A quantidade de itens na fila.
Métodos	
Clear	Remove todos os itens da fila.
Dequeue	Extraí e retorna um objeto para a fila.
Enqueue	Insere um novo objeto na fila.
Peek	Permite que você examine o próximo item sem removê-lo da fila. Isso é útil porque removê-lo poderia impedir outro método de lidar com a fila adequadamente.



NOTA

Por que os métodos são chamados de Dequeue e Enqueue? Por que não Add e Remove? É simplesmente uma questão de manter os nomes tradicionais. Os programadores sempre se referiram ao ato de adicionar um item a uma fila como *enfileirar* o item, portanto, esse termo, assim como *desenfileirar*, foram adotados.

A classe Stack é um conjunto ‘primeiro a entrar, último a sair’ (FILO, first-in, last-out). Isso significa que os itens são removidos da pilha na ordem inversa à qual foram adicionados. A ordem FILO é análoga a uma pilha de pratos: o prato adicionado por último sempre é removido primeiro. As pilhas são a solução clássica para lidar com problemas que requerem a inversão de uma ordem de operações. Muitos cálculos são processados internamente com a ajuda das pilhas. A Tabela 8.7 descreve alguns dos métodos e propriedades da classe Stack.

TABELA 8.7 Métodos e Propriedades da Classe Stack

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Count	A quantidade de itens na pilha.
Métodos	
Clear	Remove todos os itens da pilha.
Pop	Remove e retorna o próximo item para a pilha. Exatamente como a fila possui Enqueue e Dequeue, os dois métodos usados tradicionalmente para lidar com pilhas são Pop e Push.

TABELA 8.7 Métodos e Propriedades da Classe Stack (*continuação*)

<i>Membro</i>	<i>Descrição</i>
Push	Insere um novo objeto na pilha.
Peek	Permite que você examine o próximo item da pilha sem removê-lo. Queue e Stack são conjuntos interessantes, em parte por uma perspectiva histórica, mas também porque resolvem problemas específicos de programação. Quando for necessário um conjunto com o comportamento desses dois objetos, é recomendável lembrar deles.

SortedList

A classe `SortedList` possui características tanto de `ArrayList` quanto de `NameValueCollection`, assim como alguns recursos úteis adicionais específicos. Semelhante à `ArrayList`, a classe `SortedList` pode aumentar, e apresenta as mesmas propriedades `Count` e `Capacity` daquele conjunto. Como em `NameValueCollection`, cada item é identificado por um nome. Além disso, os valores de `SortedList` são ordenados com base no nome fornecido a cada item. Assim, ela será útil sempre que você precisar de uma lista de itens organizados que possa ‘crescer’, como os participantes de uma conferência ou os resultados de uma bateria de provas de uma turma escolar. A Tabela 8.8 descreve alguns dos métodos e propriedades significativos da classe `SortedList`.

TABELA 8.8 Métodos e Propriedades da Classe `SortedList`

<i>Membro</i>	<i>Tipo</i>	<i>Descrição</i>
Propriedades		
Capacity		O tamanho atual da classe <code>SortedList</code> . Configurado pela primeira vez quando ela é criada (16 por padrão), podendo aumentar conforme mais elementos forem adicionados.
Count		A quantidade efetiva de itens em <code>SortedList</code> .
Métodos		
Add		Adiciona um novo item à <code>SortedList</code> . Se isso fizer com que <code>Count</code> ultrapasse <code>Capacity</code> , essa propriedade será aumentada com sua quantidade inicial – 16 por padrão.
Propriedades		
Clear		Remove todos os itens de <code>SortedList</code> . <code>Count</code> é configurada como 0, mas <code>Capacity</code> não é alterada.
IndexOfKey		Retorna a posição de um certo objeto em <code>SortedList</code> . Isso é muito útil após a execução de <code>Sort</code> .
Item	Propriedade	Retorna um elemento específico de <code>SortedList</code> .
Keys	Propriedade	Retorna todas as chaves armazenadas em <code>SortedList</code> .
Remove	Método	Remove o objeto solicitado de <code>SortedList</code> .

TABELA 8.8 Métodos e Propriedades da Classe SortedList (*continuação*)

<i>Membro</i>	<i>Tipo</i>	<i>Descrição</i>
RemoveAt	Método	Remove o elemento da posição solicitada.
TrimToSize	Método	Diminui SortedList de modo que tanto Count quanto Capacity sejam iguais à quantidade atual de elementos na classe.
Values	Propriedade	Retorna todos os valores armazenados em SortedList, em ordem de suas chaves.

Em geral quando você trabalhar com listas, precisará mantê-las em ordem porque terá de exibi-las para o usuário em uma sequência organizada. Nessas situações, a classe SortedList pode ser útil.

Encontrando o Que Precisa no .NET Framework

Com mais de 6 mil classes no .NET Framework, encontrar o que você precisa às vezes será a parte mais difícil de seu uso. No entanto, exatamente como achar um livro em sua livraria favorita ou um site na Internet, conhecer algumas regras básicas o ajudará a encontrar a classe de que necessita para terminar seu aplicativo. Esta seção o auxiliará a descobrir várias classes diferentes no Framework, demonstrando como se faz para encontrá-las. Espero que isso seja útil em suas próprias buscas.

As Regras da Busca

Na maioria das bibliotecas, os livros se encontram organizados. Elas podem usar o sistema decimal de Dewey, algum outro método de catalogação ou apenas a ordem alfabética (por título ou sobrenome do autor). Se uma biblioteca não fosse organizada, imagine tentar encontrar um livro nela. Você teria de vagar de modo aleatório até acidentalmente se deparar com o livro desejado. Muitas pessoas tentam encontrar classes no .NET Framework de maneira semelhante.

O .NET Framework é organizado de maneira hierárquica. A denominação para cada espaço de nome é composta de uma série de trechos, separados por um ponto. Espaços de nome inferiores (os que possuem mais trechos) não estão contidos dentro dos superiores, mas relacionados a eles. Por exemplo, o espaço de nome System.Data.SqlClient está relacionado a System.Data, mas não está contido nele porque se encontra em um nível hierárquico inferior a System.Data.

No .NET Framework, há dois nomes de nível mais elevado, System e Microsoft. Os espaços de nome System são aqueles que fazem parte do .NET Framework e estão disponíveis para os usuários do Visual Basic .NET, assim como para os de outras linguagens que empreguem o Framework. As classes Microsoft são em geral específicas para o Visual Studio e destinadas a um ou

mais ambientes. Por exemplo, existe um espaço de nome `Microsoft.VisualBasic` que contém muitos dos recursos que eram encontrados no Visual Basic antes dessa versão.

A Saga pela Classe Perfeita

A fim de demonstrar algumas das técnicas que você pode usar para encontrar recursos dentro do .NET Framework, apresentarei o processo de busca de três procedimentos. Tentarei descrever alguns dos erros que cometi ao tentar encontrar o recurso e o resultado obtido. Pode haver outras maneiras de rastrear a funcionalidade procurada, ou outras respostas dentro do .NET Framework, mas essas são as que encontrei.

Que Cor Tem Aquele Pincel na Janela? Nomeando Cores

Essa pesquisa foi iniciada por uma necessidade simples, a de descrever uma cor. Os controles podem ser coloridos e a cor pode ser alterada no tempo de execução. Alterar a cor no tempo de projeto é fácil; apenas selecione a cor na lista suspensa apropriada (por exemplo, `BackColor` ou `ForeColor`). No entanto, para alterar uma cor no tempo de execução, você precisará saber como identificá-la.

Todas as cores do Visual Basic .NET podem ser descritas com base nos percentuais de vermelho, verde e azul que as compõem. Nessa linguagem cada uma das três cores citadas varia em valor dentro de um intervalo entre 0 e 255, elevando a quantidade total de cores disponíveis para 16.581.375. Portanto, você pode identificar qualquer cor através de suas proporções de vermelho, verde e azul. Uma cor composta com cada uma dessas outras três com 50% de saturação, ou com um valor de 128, teria uma tonalidade cinza-clara.

Em outras situações, pode-se querer identificar uma cor pelo nome – isto é, quando ‘vermelho’ for mais relevante do que saber que o valor da cor é -65536.

Você poderia considerar que, por serem as cores tão importantes, suas denominações apareceriam no espaço de nome `System`. No entanto, se pensasse assim (e foi o que aconteceu comigo, na primeira busca), estaria errado. As cores com denominações conhecidas e a capacidade de criar outras novas são encontradas no espaço de nome `System.Drawing`.

Dentro do espaço de nome `System.Drawing`, há dois conjuntos de cores: a estrutura `Color` e a classe `SystemColors`.

A estrutura `Color` tem duas finalidades principais. A primeira é seu aspecto ‘funcional’. Dois métodos, descritos na Tabela 8.9, permitem que você crie uma cor e descubra seu valor numérico.

TABELA 8.9 Métodos da Estrutura Color

<i>Método</i>	<i>Descrição</i>
FromARGB	Cria uma cor nova com base nos valores de vermelho, verde e azul (cada um entre 0 e 255). Também há outras versões dessa função. Uma variação usa um valor Alpha além dos de vermelho, verde e azul, enquanto outra é empregada para adicionar o valor Alpha a uma cor existente. Esse valor representa a transparência da cor. Como os outros valores, Alpha varia de 0 a 255, sendo esse último valor o correspondente a completamente opaco. A última variação de FromARGB utiliza um valor inteiro e retorna a cor.
ToARGB	Retorna o inteiro que representa uma composição da cor que usa Alpha, vermelho, verde e azul.

Além da capacidade de criar e converter cores, a estrutura Color contém várias cores identificadas. Elas constituem uma ampla variedade de cores diferentes, muitas com nomes representativos e evocativos, como PapayaWhip, Gainsboro e BurlyWood. Esses são atalhos úteis se você quiser colorir algo, mas não estiver certo do valor numérico exato que deseja. Mais vantajoso ainda do que os nomes é que você não tem de criar uma instância nova da estrutura Color antes de usar essas cores. Portanto, poderá empregá-las em um programa utilizando um código semelhante a

```
frmMain.BackColor = Color.SeaShell
```

Um problema na definição de suas próprias cores, com os valores RGB (vermelho, verde e azul) ou selecionando uma cor identificada, é que sua percepção delas pode ser diferente da de seus usuários. Tenho encontrado pessoas que configuram seus microcomputadores com as cores mais extravagantes, e outras que nunca alteram os padrões. De maneira semelhante, estive com programadores que tinham criado aplicativos estranhos, multicoloridos, e outros que embutiram todas as cores em códigos de sombreados cinzas. Se você embutir as cores em códigos, seu programa deverá se impor nos microcomputadores dos usuários, podendo se tornar irritante. Para evitar isso, programadores educados configuram a cor dos elementos da tela usando as do sistema (aquelas definidas no painel de controle). As cores do sistema estão disponíveis na classe SystemColors, que possui várias propriedades representando os diferentes elementos que aparecem no Windows (como o WindowsText, o Control ou o Menu). Colorir as partes de seu aplicativo usando essas cores significa que elas assumirão qualquer cor que o usuário selecionar para esse item.

Usar a classe SystemColor é semelhante a empregar a estrutura Color. Não é necessário criar uma nova instância da classe antes de utilizar os valores. Portanto, você poderia colorir o plano de fundo de um formulário com a cor já definida Desktop por meio do código a seguir:

```
frmMain.BackColor = SystemColors.Desktop
```

As cores estão em todos os locais no Windows, portanto, saber a maneira de criá-las é importante.

Quem Sou Eu? Encontrando o Nome de Seu Computador

Muitos programas precisam conhecer o nome do computador em que estão sendo executados. Por exemplo, você pode estar tentando acessar alguns recursos ou serviços do sistema, como

fará no Dia 13, “Usando o Server Explorer”. Ou pode querer registrar ou armazenar o nome do computador para acessar o banco de dados. Essa busca de uma solução foi iniciada por mim por uma necessidade desse tipo. Infelizmente, também foi a mais demorada. Sorte sua que encontrei muitas maneiras que podem ser usadas para determinar o nome de seu computador. Dependendo de que espaços de nome já tenha importado ou do tipo de aplicativo que estiver executando, pelo menos uma delas (e provavelmente todas) devem estar disponíveis para você.

Você já examinou a primeira das três maneiras fáceis de obter o nome de um computador em que um programa está sendo executado: `System.Environment`. No Windows 2000 e superiores, o nome do computador é armazenado como uma variável de ambiente chamada `COMPUTERNAME`. Se esse for o caso, o nome do computador local poderá ser recuperado com

```
Console.WriteLine("Using Environment: {0}", _  
System.Environment.GetEnvironmentVariable("COMPUTERNAME"))
```

No entanto, depender de uma variável de ambiente, mesmo que configurada pelo sistema operacional, me parece confiar um pouco demais. Assim, fui mais fundo.

A rede em geral precisa do nome do computador, portanto `System.Net` foi minha próxima parada. Depois de algumas tentativas sem êxito, encontrei `GetHostName` na classe `DNS`. Isso basicamente faz sentido – o DNS (Domain Naming Service) é um serviço usado para registrar nomes de computadores na Internet (ou em redes locais). Você pode usar a classe `DNS` para recuperar o nome de um computador incluindo o espaço de nome `System.Net` em seu projeto e acessando-o com um código semelhante a

```
Console.WriteLine("Using Net.DNS: {0}", DNS.GetHostName)
```

Essa solução pode não funcionar, no entanto, se você utilizá-la em um computador que não tenha o TCP/IP instalado (o que, com a popularidade da Internet, poderia significar cerca de dois computadores). E assim, sua pesquisa continuaria.

Bem no âmago de `System.Windows.Forms` se encontra uma classe que é um tesouro: `SystemInformation`. É muito estranho que `SystemInformation` possua informações sobre o programa atual e o sistema operacional no qual ele está sendo executado. Se você examinar essa classe, verá que ela tem propriedades que representam todas as definições essenciais para os aplicativos, principalmente aqueles com base em formulários Windows: `FrameBorderSize`, `CaptionHeight`, `MenuHeight` e `MousePresent`. O mais importante para a sua pesquisa é que a classe `SystemInformation` possui uma propriedade `ComputerName`, que pode ser usada para recuperar o nome do computador se o espaço de nome `System.Windows.Forms` for incluído em seu aplicativo, como descrevo a seguir:

```
Console.WriteLine("Using SystemInformation: {0}", _  
SystemInformation.ComputerName)
```

Não deixe o fato de essa classe estar oculta dentro do espaço de nome `System.Windows.Forms` desanimá-lo. Ela pode ser usada em qualquer tipo de aplicativo se o espaço de nome for carregado.

Para que ter três maneiras de obter o nome do computador em que um programa está sendo executado em vez de apenas uma? Bem, apesar de não ser uma das pessoas que decide esse tipo de coisa, sugeriria algumas razões:

- Grupos diferentes trabalhando em seções distintas criaram maneiras diferentes de resolver o mesmo problema.
- Cada solução só é apropriada em certas situações (como, por exemplo, se a variável de ambiente for configurada, se você executar uma rede TCP/IP, se usar formulários Windows e assim por diante).

Na pesquisa através do .NET Framework, em geral encontramos múltiplas soluções. Use a que atender melhor suas necessidades, até encontrar uma ainda mais adequada.

Um Certo Dia na História

A busca dessa solução foi iniciada em uma função que existia no Visual Basic 6, `WeekDayName`. Essa função retorna o nome do dia da semana. Você pode descobrir esse dia em qualquer data usando

```
sDayName = WeekDayName(WeekDay(CDate("May 6, 1937")))
```

Só para registro, o desastre com o Hindenburg foi em uma terça-feira.

Essa função, como muitas outras que existiam em versões mais antigas do Visual Basic, pode ser encontrada de maneira bastante fácil, no espaço de nome `Microsoft.VisualBasic`. Esse espaço de nome é carregado de modo automático quando o IDE é utilizado no desenvolvimento, permitindo que códigos escritos para versões anteriores do Visual Basic sejam executados. Portanto, se tudo que você deseja é uma função que ‘costumava existir’, provavelmente ela estará nesse espaço de nome.

Resumo

Em geral é difícil compreender onde o .NET Framework termina e o Visual Basic .NET começa. A maioria dos recursos do Visual Basic .NET na verdade vem do Framework, incluindo os que anteriormente faziam parte da API do Windows ou que não estavam disponíveis para os programadores do Visual Basic antes dessa versão. É incompleto descrever o .NET Framework como algo importante para os desenvolvedores do Visual Basic .NET. No Dia 17, “Usando o .NET Framework”, você passará mais tempo com o conjunto de classes do .NET Framework. Além disso, é possível obter mais informações sobre o .NET Framework na seção do MSDN (Microsoft Developer Network) sobre a plataforma .NET no endereço <http://microsoft.com/net>.

No Dia 9, “Desenvolvendo uma Interface com o Usuário com os Formulários Windows”, examinaremos outra família de classes do .NET Framework, as que permitem a construção de interfaces com o usuário do Windows. Essas classes serão usadas com frequência quando você criar soluções que empreguem as sofisticadas interfaces com o usuário dos aplicativos-padrão.

P&R

P Devo usar o .NET Framework ou uma função antiga sobre a qual li que o Visual Basic dá suporte?

R Se você, no passado, programou no Visual Basic, ou ainda, se leu livros do tipo ‘Como Fazer Isso’ ou mesmo artigos de revista que abordavam versões anteriores dessa linguagem, terá visto várias das maneiras ‘antigas’ de executar tarefas. Muitas dessas funções ainda existem, dentro do espaço de nome `Microsoft.VisualBasic`. É possível importar esse espaço de nome para fazer uso de algumas das funções ou constantes mais antigas, como as funções matemáticas ou a constante `vbCrLf` (uma constante que significa ‘Adicione uma nova linha aqui’). No entanto, se possível, seria melhor empregar os recursos mais recentes do .NET Framework.

P Por que você não abordou o espaço de nome "Fill-in-My-Favorite"? Preciso muito saber como usá-lo em meu programa.

R Com 6 mil classes disponíveis dentro do .NET Framework, uma lição como esta mal consegue abordar mesmo uma pequena porção delas com os detalhes que precisamos. Você se aprofundará mais no .NET Framework no Dia 17. Além disso, também iremos usá-lo em outros capítulos que compõem este livro.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Como você poderia fazer uso da classe `Console` se escrever um aplicativo com base no Windows ou na Web?
2. Por que seria melhor escolher o conjunto `SortedList` em vez de `ArrayList` se esse possui um método de ordenação?
3. Qual seria o resultado da execução do código a seguir?

```
Dim oRand As New System.Random
Dim iValue As Integer = oRand.Next(1, 6) + oRand.Next(1, 6)
Console.WriteLine(iValue)
```

Exercícios

1. Escreva um pequeno programa do console que aceite várias palavras na linha de comandos e as exiba na tela em ordem alfabética.

SEMANA 2

DIA 9

Desenvolvendo uma Interface com o Usuário com os Formulários Windows

No Dia 5, organizando programas, você aprendeu diferentes maneiras pelas quais os aplicativos .NET poderiam ser projetados. Uma decisão essencial era que tipo de interface com o usuário criar. A escolha em geral termina com um aplicativo no estilo Windows ou da Web. Na plataforma .NET, o modo existente para gerar um aplicativo Windows é por meio da tecnologia Windows Forms (formulários Windows). Na lição de hoje examinaremos:

- A adição e manipulação de controles em um formulário.
- A manipulação em código de eventos da interface com o usuário.
- A aceitação e validação de entradas do usuário.
- O uso de caixas de diálogo.

Além desses tópicos, serão abordados vários controles especiais visíveis e ocultos usados nos formulários Windows.

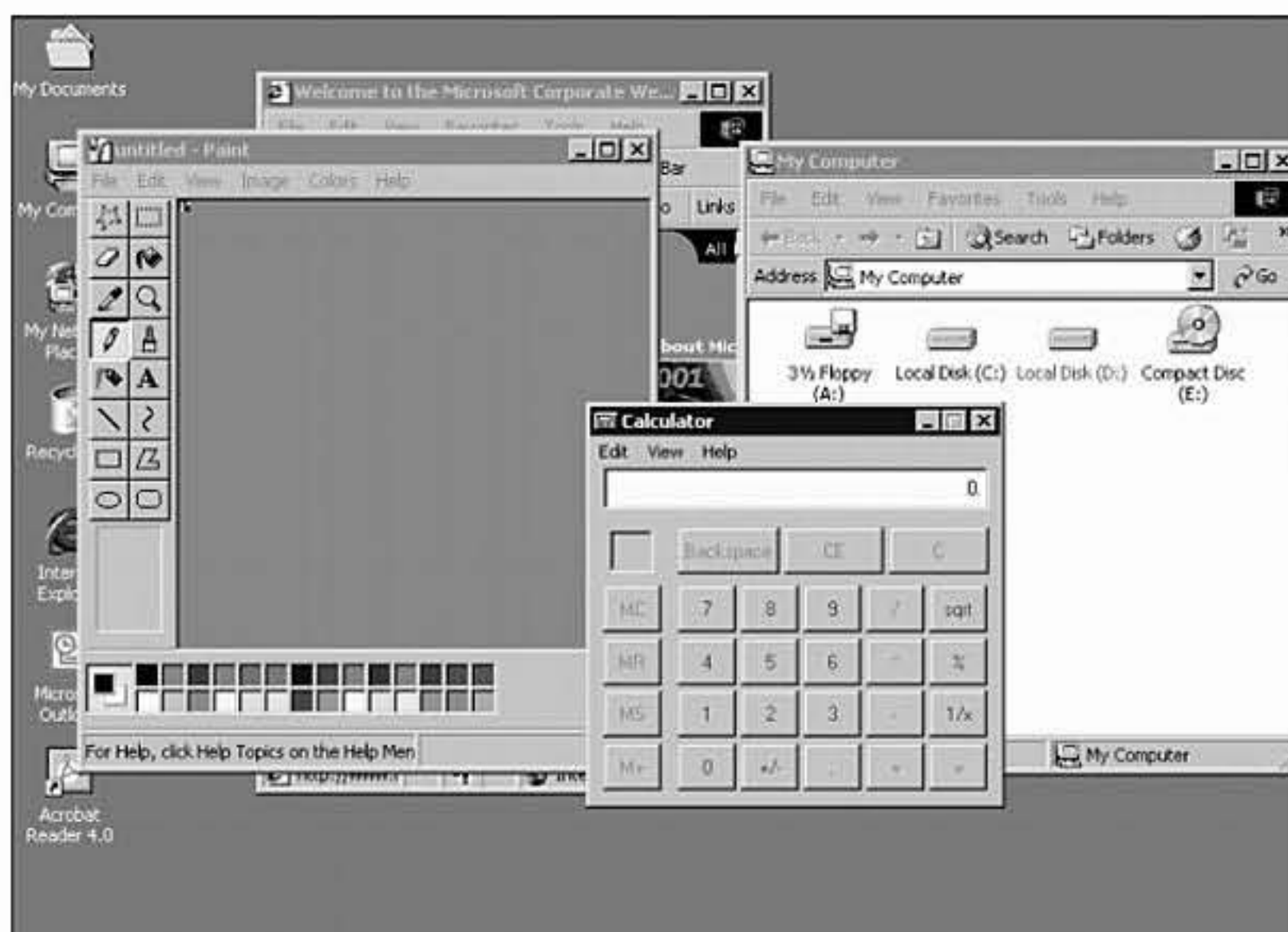
Visão Geral dos Formulários Windows

Os formulários Windows (Windows Forms) são utilizados na criação de aplicativos que fazem uso de interfaces com o usuário comuns (com pequenas variações na aparência final) em todos

os sistemas operacionais Windows (veja a Figura 9.1). Empregando esse conjunto de classes e funções, você poderá adicionar esse estilo de interface com o usuário, incluindo janelas e caixas de diálogo, em seus aplicativos.

FIGURA 9.1

A maioria dos aplicativos Windows compartilha o mesmo estilo de interface porque o sistema operacional manipula o desenho das janelas e outros recursos dela.



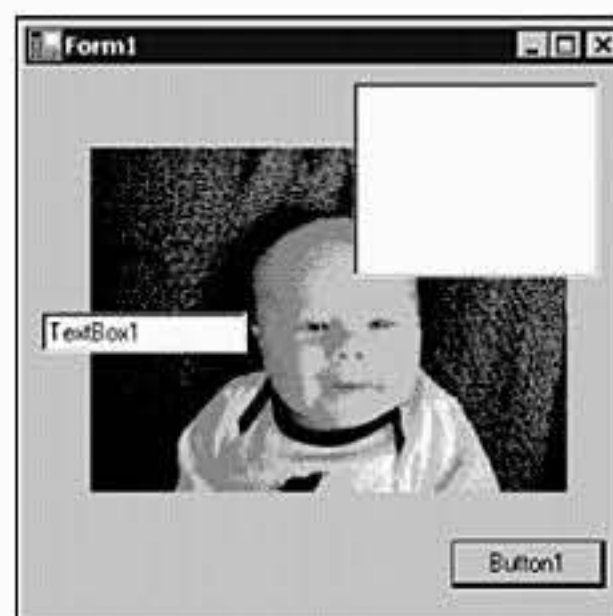
Antes que você se aprofunde na criação e configuração de um aplicativo Windows, faremos uma recapitulação dos principais termos relacionados aos formulários Windows. O primeiro e mais difícil de definir é 'janela'. Uma *janela*, na perspectiva dos computadores, é uma área da tela usada por um aplicativo para exibir informações. Um único aplicativo pode ter mais de uma janela, mas cada janela pertence a apenas um aplicativo. Elas podem ser movidas e redimensionadas (incluindo a maximização e minimização), mas essas opções podem ser desativadas pelo aplicativo que possui a janela, permitindo que ela tenha um tamanho ou uma posição fixa. Os subelementos da janela, como os botões, caixas de texto, listas ou figuras (veja a Figura 9.2) compõem a interface de um aplicativo e são chamados de *controles*.

Esses controles produzem a interface do aplicativo porque podem ser manipulados (clicados, digitados, ou o que mais for apropriado) para fornecer entradas ao programa. Eles também exibem informações para o usuário. As janelas podem estar em um entre dois estados – modal ou não-modal. As janelas do tipo modal devem ser manipuladas e fechadas antes que o usuário possa trabalhar com qualquer outro aspecto desse aplicativo – por exemplo, uma janela que pergunte se você deseja salvar as alterações de um documento. As janelas do tipo não-modal permitem que o usuário interaja de uma maneira menos estruturada, movendo-se entre todas elas em um aplicativo quando quiser, se assemelhando mais a janela de um documento do Microsoft Word ou a Toolbox do Visual Studio .NET. A janela modal também é conhecida como *caixa de diálogo* e na plataforma .NET possui recursos específicos que a torna fácil de usar para obter algum

tipo de informação de um usuário. As caixas de diálogo serão abordadas com mais detalhes ainda nesta lição.

FIGURA 9.2

Todos os elementos de uma janela são chamados de controles.



Criando um Aplicativo com Formulários Windows

Como exemplo, você pode desenvolver um aplicativo simples com formulários Windows que aceite entradas do usuário e demonstre como esses objetos da interface funcionam. Associado ao aprendizado sobre os formulários Windows, este exemplo também mostrará como executar algumas manipulações simples de arquivos usando as classes `System.IO` do .NET Framework. Primeiro, o conduzirei pela criação deste exemplo simples e, em seguida, abordarei diretamente tópicos individuais sobre procedimentos de eventos e controles.

Configurando o Projeto

A primeira etapa no desenvolvimento do projeto do Visual Basic .NET que usa os formulários Windows é criar um projeto novo, selecionando o tipo de projeto aplicativo Windows (Windows application). Um projeto novo será gerado contendo um único objeto formulário, que estará pronto para você começar a trabalhar. O novo formulário será chamado de `Form1` por padrão, mas não pretendemos usar esse nome porque ele não possui um significado em seu aplicativo.

Já que você está desenvolvendo um programa que trabalhará com arquivos, renomeie o formulário como `frmFiler`, o que deve ser feito em duas etapas:

1. Renomeie o arquivo dando um clique com o botão direito do mouse em `Form1.vb` na janela do Solution Explorer, selecionando `Rename` e, em seguida, digitando o novo valor `frmFiler.vb`.
2. Dê um clique com o botão direito do mouse no novo formulário em branco da janela de projeto que o estiver exibindo e selecione `Properties`. Isso deve exibir a janela `Properties Toolbox`, por meio da qual você poderá encontrar e alterar a propriedade `Name` (que pode ser visualizada na seção `Design` das propriedades) de `Form1` para `frmFiler`. É melhor fa-

zer isso antes que qualquer codificação seja iniciada, evitando situações em que já tenha escrito um código que se refira a Form1.

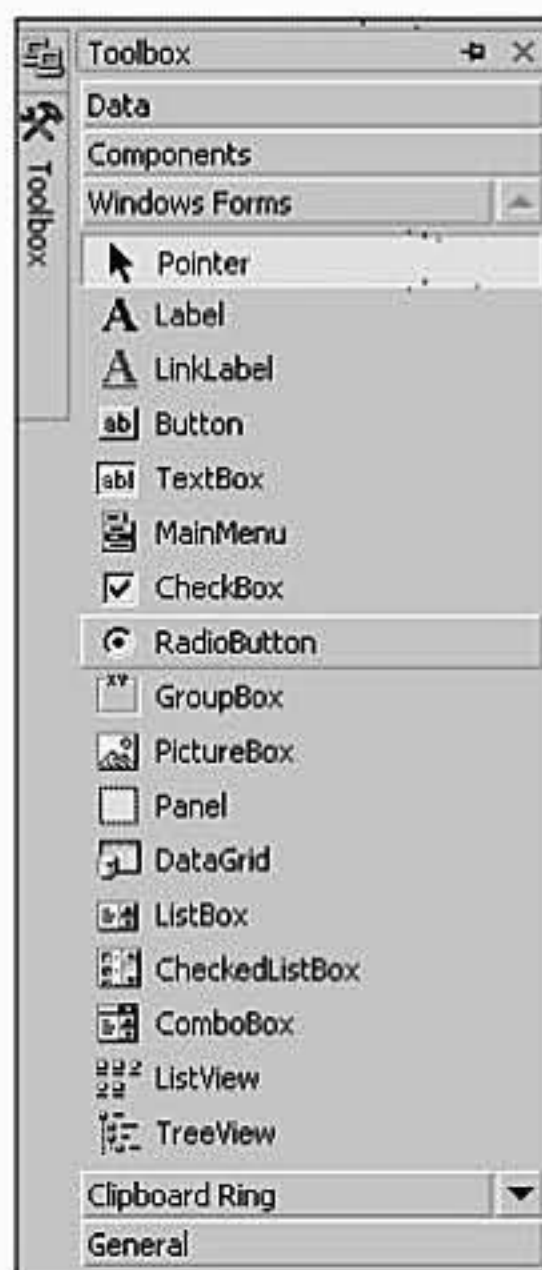
Neste ponto, observe que o título do formulário ainda será Form1, e não frmFiler como você esperava. Isso acontece porque ele não está associado ao nome do formulário, mas o IDE usa esse título como padrão na criação. Acesse a janela de propriedades do formulário novamente e encontre a propriedade Text, que representa o título do formulário. Altere-o para o que desejar – Filer ficaria bom –, e o verá ser exibido conforme o pretendido.

Adicionando Controles ao Formulário

Antes que um formulário possa ter alguma utilidade, você precisa inserir controles nele. Todos os controles embutidos nos formulários Windows estão disponíveis por meio da janela Toolbox. Exiba-a na tela e localize a seção Windows Forms (veja a Figura 9.3).

FIGURA 9.3

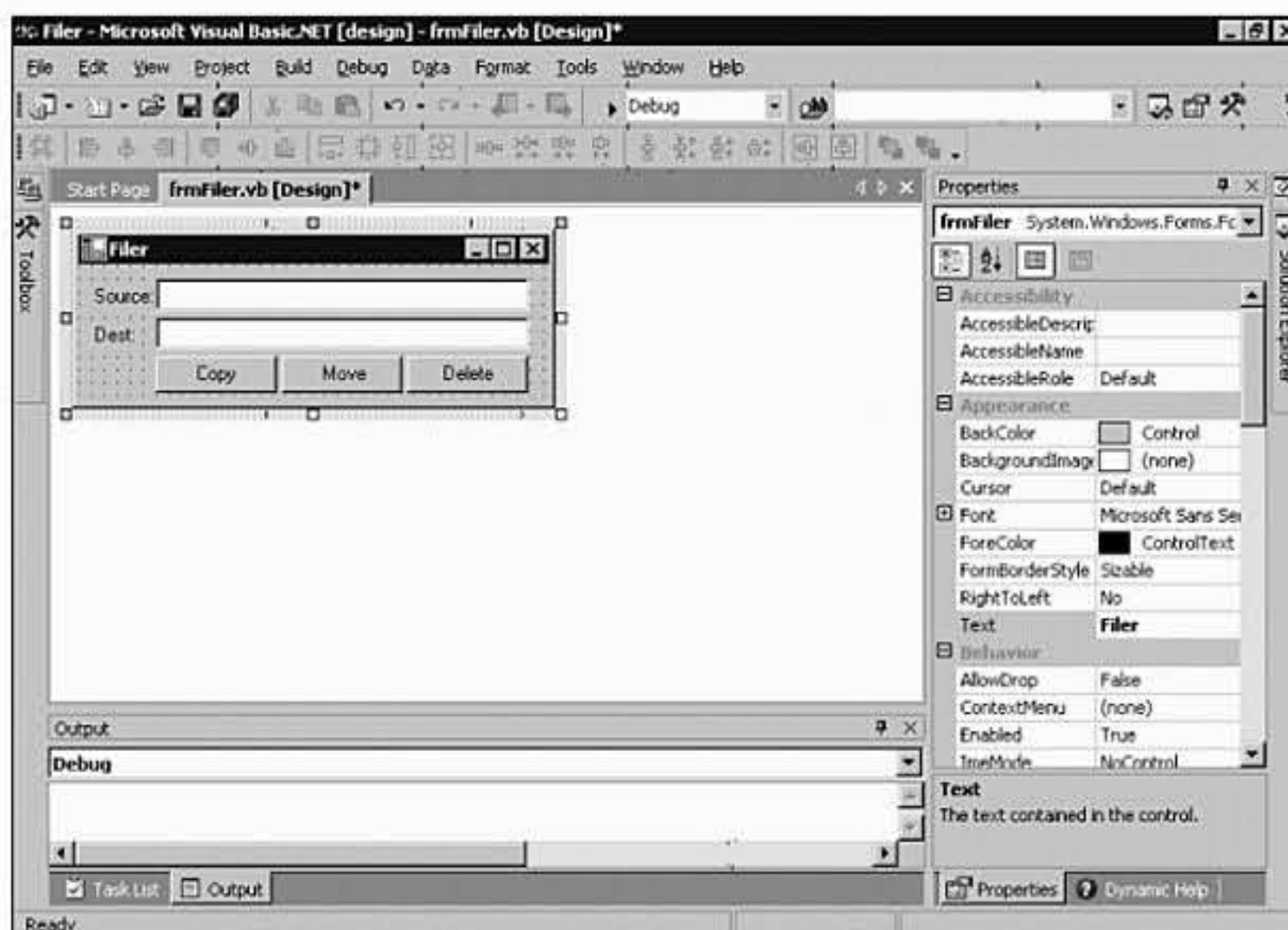
A seção Windows Forms de Toolbox fornece uma grande variedade de controles para seus formulários.



Nessa seção de Toolbox, você encontrará vários controles, mas por enquanto só precisaremos dos botões e caixas de texto. Adicione um controle ao formulário dando um clique duplo nele (o que adiciona o controle ao canto superior esquerdo do formulário), dando um clique e arrastando-o para o formulário, ou selecionando o controle (dê um clique e solte) e, em seguida, dando um clique e arrastando o mouse no formulário para demarcar o local desejado. Qualquer desses métodos produzirá o mesmo resultado: o controle será inserido no formulário. Empregando um dos métodos que acabaram de ser descritos, adicione três botões, dois títulos e duas caixas de texto ao formulário. Organize os controles conforme o layout mostrado na Figura 9.4, empregando a propriedade Text dos botões e títulos para tornar a interface do formulário igual à da figura.

FIGURA 9.4

Uma maneira possível de organizar a interface no aplicativo de exemplo.



Nomeando Seus Controles

Antes de criar um manipulador de eventos, renomeie todos os nomes-padrão de seus controles (Button1, Button2, TextBox1 e assim por diante). Você terá de usar o nome para acessar o controle e seus atributos, portanto, recomendo que utilize alguma forma de convenção de nomeação. Veja a seguir minhas sugestões para esses controles, com base na convenção de nomeação que tenho usado (e muitas outras pessoas também) no Visual Basic há anos:

- txtSource e txtDestination para as duas caixas de texto, ordenadas de cima para baixo.
- btnCopy, btnMove e btnDelete para os botões. No Visual Basic 6.0 e em versões anteriores, os botões eram chamados de *botões de comando* (em vez de apenas 'botão' como no Visual Basic .NET), portanto, muitas pessoas usavam o prefixo cmd quando os nomeavam.
- lblSource e lblDestination para os títulos.

Renomear um controle é igual a fazê-lo com um formulário: selecione o controle, acesse suas propriedades e, em seguida, altere a propriedade (Name). Renomeie todos os controles antes de passar para a próxima parte do exemplo.

Manipulação de Eventos

Para programar uma interface com o usuário, você precisa entender o conceito de eventos e programação dirigida a eventos. Sempre que o usuário executa uma ação, como dar um clique em um botão ou digitar um valor na caixa de texto, ela é chamada de *evento*. Se quisermos ter um código que seja processado quando um evento ocorrer, precisaremos criar um manipulador de eventos para aquele no qual estejamos interessados.

Por meio do IDE do Visual Studio .NET, será fácil criar um manipulador de eventos para um de seus controles. Um evento comum para o qual pretendemos ter um código é, por exemplo, o clique em um botão de seu formulário. Essa ação é chamada evento `Click` de um botão, e pode-se adicionar ou editar o código associado a esse evento dando um clique duplo no botão. Você será enviado para a janela de edição de códigos e a uma nova sub-rotina chamada *<nome do botão>_Click* (veja a Listagem 9.1). O nome da rotina não é muito importante, mas é o formato-padrão que o IDE utiliza. Dê um clique duplo no botão `Copy` para passar para o código apropriado. Observe que se não tivesse renomeado o botão, seu novo evento `Click` teria um nome parecido com `Button1_Click`, em vez de `btnCopy_Click`.

LISTAGEM 9.1 Listando um Manipulador de Eventos Vazio para o Evento `Copy`

```
1 Private Sub btnCopy_Click(ByVal sender As System.Object, _  
2   ByVal e As System.EventArgs) Handles btnCopy.Click  
3  
4 End Sub
```

Agora, qualquer código que você inserir nessa rotina será executado quando o usuário der um clique no botão. Neste exemplo, adicionaremos um código a todos os botões (`btnCopy`, `btnMove` e `btnDelete`), mas teremos de começar com um, portanto, dê um clique duplo em `btnCopy` e acesse o código associado a ele. Nesse manipulador de eventos, queremos adicionar um código que copie o arquivo especificado em `txtSource` para o existente em `txtDestination`. Para tanto, teremos de trabalhar com o espaço de nome `System.IO`. Esse espaço de nome já foi referenciado em seu projeto porque faz parte de `System.dll`, mas podemos tornar seu código muito mais simples de ler e digitar se adicionarmos uma instrução `Imports System.IO` no início do código de seu formulário.

Depois que a instrução for inserida, você poderá referenciar os objetos desse espaço de nome sem ter de incluir a referência completa. Para obter uma cópia do arquivo, podemos usar o método estático `Copy` da classe `System.IO.File`, mas antes precisamos dos nomes dos arquivos de origem e de destino. Para acessar o conteúdo de uma caixa de texto, use sua propriedade `Text`, referindo-se ao objeto específico através de seu nome.

LISTAGEM 9.2 Copiando Arquivos por Meio dos Métodos Estáticos do Objeto `File`

```
1 Private Sub btnCopy_Click(ByVal sender As System.Object, _  
   ByVal e As System.EventArgs) Handles btnCopy.Click  
2     Dim sSource As String  
3     Dim sDestination As String  
4     sSource = txtSource.Text()  
5     sDestination = txtDestination.Text()  
6     File.Copy(sSource, sDestination)  
7 End Sub
```

ANÁLISE

Com os valores que você precisa armazenados em variáveis alfanuméricas, a cópia efetiva do arquivo pode ocorrer. Já que `Copy` é um método estático da classe `File`, não é necessário criar uma instância; basta chamar `File.Copy`. No entanto, só essa linha (linha 6 da Listagem 9.2) não representa todo o processo, porque um tratamento apropriado de erros deve ser adicionado. No caso de uma cópia de arquivo, muitos erros diferentes poderiam ocorrer, desde ‘o arquivo de destino não existe’ até ‘não há espaço suficiente em disco’. Sempre que um erro acontecer, poderemos apenas comunicar ao usuário usando o método estático `Show` da classe `MessageBox`. Essa classe, que já foi empregada neste livro, faz parte do espaço de nome `System.Windows.Forms` e é uma maneira rápida e fácil de inserir uma caixa de diálogo. A Listagem 9.3 fornece um exemplo da utilização do método `MessageBox.Show` em resposta a uma exceção.

LISTAGEM 9.3 Usando a Classe `MessageBox` para Exibir um Erro

```

1 Private Sub btnCopy_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles btnCopy.Click
3   Dim sSource As String
4   Dim sDestination As String
5   sSource = txtSource.Text()
6   sDestination = txtDestination.Text()
7   Try
8     File.Copy(sSource, sDestination)
9   Catch objException As Exception
10    MessageBox.Show(objException.Message)
11  End Try
12 End Sub

```

Criando Vários Manipuladores de Eventos para um Único Evento

A parte importante da rotina do manipulador de eventos não é seu nome, `btnCopy_Click`, mas `Handles btnCopy_Click` que foi adicionada no final da declaração. Essa instrução informa ao Visual Basic .NET que essa rotina é o manipulador desse evento. Diferente das versões anteriores do Visual Basic, que usavam apenas o nome da rotina para associar o código a eventos específicos, no Visual Basic .NET é possível ter um único procedimento manipulando múltiplos eventos ou só um evento com vários manipuladores. Copie toda a rotina `btnCopy_Click` e a renomeie como `CopyClick`, deixando a mesma lista de parâmetros e ainda usando `Handles btnCopy.Click`.

A identidade do procedimento (que é composta de sua lista de parâmetros e do valor retornado, se houver algum) deve ser exatamente a mesma vista no procedimento original para que atue como um manipulador de eventos, mas o código pode ser qualquer outro. Agora, altere o código desse segundo procedimento para que apenas exiba uma mensagem em vez de realizar todo o trabalho de cópia do arquivo (veja a Listagem 9.4). Tente executar seu projeto e dê um clique no

botão Copy. Os dois manipuladores de eventos serão chamados, fazendo a cópia do arquivo e exibindo a caixa de diálogo com a mensagem.

LISTAGEM 9.4 A Instrução Handles Vincula uma Rotina a um Evento Específico

```
1 Private Sub CopyClick(ByVal sender As System.Object, _  
2   ByVal e As System.EventArgs) Handles btnCopy.Click  
3   MessageBox.Show("CopyClick")  
4 End Sub
```

Remova a rotina CopyClick selecionando e excluindo o texto, e passe para as rotinas dos eventos dos dois outros botões.

Encontrando Objetos e Eventos por Meio do Editor de Códigos

Em vez de voltar à janela do projeto e dar um clique duplo em cada um dos outros dois botões, você pode passar para a rotina de manipulação do evento Click diretamente na janela do código. Selecione o nome do objeto (btnMove, por exemplo) na primeira lista suspensa (lado esquerdo) acima do editor de códigos e, em seguida, selecione o evento desejado (Click) na segunda lista suspensa. Isso terá o mesmo efeito de dar um clique duplo no objeto do projeto, portanto, os dois métodos podem ser usados. O código dos botões Move e Delete empregarão os métodos estáticos correspondentes do objeto File. A Listagem 9.5 mostra uma maneira com a qual esses dois botões poderiam ser codificados.

LISTAGEM 9.5 Transferindo e Excluindo Arquivos por Meio dos Métodos Estáticos do Objeto File

```
1 Private Sub btnMove_Click(ByVal sender As System.Object, _  
2   ByVal e As System.EventArgs) Handles btnMove.Click  
3   Dim sSource As String  
4   Dim sDestination As String  
5   sSource = txtSource.Text()  
6   sDestination = txtDestination.Text()  
7   File.Move(sSource, sDestination)  
8 End Sub  
9  
10 Private Sub btnDelete_Click(ByVal sender As Object, _  
11   ByVal e As System.EventArgs) Handles btnDelete.Click  
12   Dim sSource As String  
13   sSource = txtSource.Text()  
14   File.Delete(sSource)  
15 End Sub
```

Múltiplos Eventos com um Manipulador

Como alternativa, você pode manipular os três eventos `Click` por meio de um procedimento, alterando a instrução `Handles` para que inclua todos eles. Apenas a declaração dessa rotina de manipulação de eventos é mostrada na Listagem 9.6; os detalhes foram deixados para os exercícios desta lição.

LISTAGEM 9.6 Usando a Palavra-Chave `Handles` para Vincular Vários Eventos a um Único Procedimento ou Vários Procedimentos a um Único Evento

```
1 Private Sub DoEverything(ByVal sender As Object, _  
2   ByVal e As System.EventArgs) _  
3   Handles btnCopy.Click, btnMove.Click, btnDelete.Click
```

A única restrição à quantidade de eventos que você pode manipular com apenas uma rotina é que todos têm de empregar o mesmo conjunto de parâmetros. O evento `Click` de um botão, por exemplo, recebe dois parâmetros (`System.Object` e `System.EventArgs`), enquanto o evento `ChangeUICues` do mesmo botão recebe parâmetros diferentes (`System.Object` e `System.Windows.Forms.UICuesEventArgs`). Portanto, os dois não poderiam ser manipulados por uma única rotina. A finalidade desses parâmetros é fornecer informações sobre o evento a seu manipulador, porém alguns eventos passam informações de tipos diferentes das de outros.

Mais Informações sobre os Controles

O exemplo no qual você acabou de trabalhar fez com que adicionasse caixas de texto, títulos e botões a um formulário novo, mas muitos outros controles estão disponíveis.

A seguir encontramos uma lista com vários dos controles embutidos mais freqüentemente usados, e uma descrição breve:

- **Label/Link Label** Estes dois controles proporcionam uma maneira de inserir textos estáticos (não editáveis) em um formulário. O controle `Link Label` adiciona alguns recursos complementares, que podem fazer com que uma parte ou todo o seu texto se pareça com um link HTML e possua um evento extra (`LinkClicked`) para indicar quando um usuário deu um clique no trecho do hiperlink contido no nome.
- **Button** O botão-padrão, com bem mais recursos do que possuía em versões anteriores do Visual Basic. Com as propriedades do botão, você pode adicionar uma figura, alterar o layout do texto e da figura, configurar o título e muito mais.
- **Text Box** Fornece uma área para inserção de texto em formulários e dá suporte a uma edição básica. Pode ter várias linhas se a propriedade `MultiLine` for configurada com `True` e disponibiliza um menu de atalho-padrão (com um clique no botão direito do mouse) que dá ao usuário o recurso de recortar/copiar/colar.

- **Main Menu e Context Menu** Embora não estejam juntos na Toolbox, esses dois controles são usados na criação de sistemas com menus para sua janela. O Dia 16, “Formulários Windows Avançados” detalhará como construir menus para seus sistemas usando esses dois controles.
- **Check Box/Radio Button** Estes dois controles são usados para exibir e inserir valores simples sim/não ou verdadeiro/falso. Em geral são empregados para mostrar uma lista de opções que devem ser ativadas ou desativadas, mas a diferença entre eles está em como manipulam a seleção de outro controle do mesmo tipo dentro de seu grupo. Em um grupo de botões de opção, a intenção é que apenas um deles seja ativado por vez. Por outro lado, várias caixas de seleção podem ser selecionadas, o que significa que você pode marcar quantas desejar.
- **Picture Box** Disponibiliza um local para você exibir uma figura interessante ou desenhar figuras usando a biblioteca de elementos gráficos.

Há muitos outros, mas todos funcionam em essência da mesma maneira. Você os insere em seu formulário, manipula suas propriedades por meio de código ou da janela Properties e escreve um código para manipular seus eventos. Embora não tenha a intenção de abordar cada controle individualmente, detalharei o uso de dois deles a título de demonstração.

Criando Grupos de Botões de Opção

Como já mencionei, os botões de opção e as caixas de seleção são usados para representar as configurações que podem ser ativadas ou desativadas, mas os dois tipos de controle apresentam uma diferença importante. Os botões de opção foram projetados para ser usados apenas em grupos de dois ou mais, e apenas um botão do grupo pode ser selecionado. Dar um clique em um botão de opção faz com que ele seja selecionado e, automaticamente, desativa o botão ativado antes. As caixas de seleção, por outro lado, também são ativadas e desativadas por meio de um clique, mas uma quantidade aleatória delas pode ser selecionada ao mesmo tempo em um grupo. Considere esses dois tipos de controles em termos de um exame de múltipla escolha: as caixas de seleção seriam adequadas para uma pergunta em que a resposta fosse elaborada como “Selecione todas as respostas corretas”, enquanto os botões de opção só serviriam em perguntas do tipo “Selecione a resposta mais apropriada”.

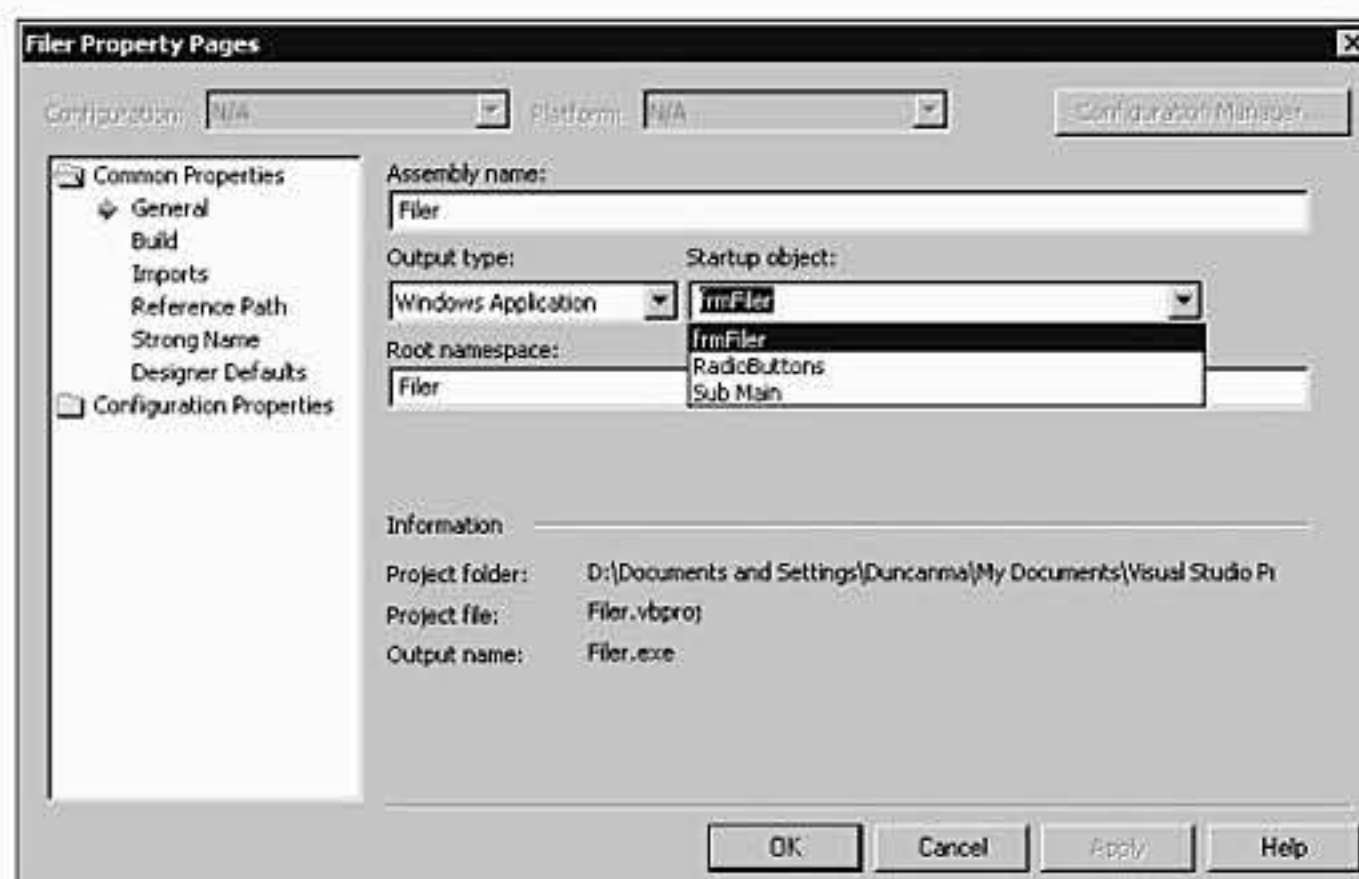
Para criar um grupo de botões de opção, primeiro é preciso gerar um novo formulário, porque o anterior, Filer, já está bem cheio. Com o projeto que contém Filer aberto na janela Solution Explorer, dê um clique nele com o botão direito do mouse, selecione as opções Add e Add Windows Form no menu que aparecerá, e digite um nome para seu novo formulário na caixa de diálogo resultante (você poderia digitar **Radio uttons**, por exemplo). Dê um clique em OK para adicionar esse formulário novo ao projeto atual. Agora, no formulário recém-criado, use a Toolbox para adicionar um controle Group Box. Esse controle, que possui uma aparência retangular com um título e bordas de aparência bem definida, é usado na criação de grupos e contém outros controles. Mova e redimensione a caixa de modo que ocupe a tela quase toda. Para criar um conjunto de botões de opção, só temos de começar a adicioná-los a essa caixa de grupo. Insira quatro

desses controles na caixa e posicione-os em um layout apropriado. Não é preciso fazer nada mais com eles para que ajam como um grupo; isso é determinado pelo fato de que foram inseridos na mesma caixa de grupo.

Se você executasse seu projeto neste momento, seu novo formulário não apareceria porque, ao ser criado, o projeto é configurado para exibir o primeiro formulário. Para alterar isso a fim de que seu novo formulário seja executado em vez do primeiro, dê um clique com o botão direito do mouse no projeto, que se encontra na janela Solution Explorer (do mesmo modo como quando adicionou um arquivo anteriormente), e selecione Properties no menu de atalho. Isso abrirá um caixa de diálogo grande contendo várias opções e configurações (veja a Figura 9.5). O valor que precisamos alterar está na primeira página (Common Properties, General) e se chama Startup Object. A configuração de Startup Object determina qual dos formulários, ou outros códigos, será executado quando o projeto for processado. No momento, ele deve ter o valor frmFiler (o nome do primeiro formulário deste projeto). Altere-o para RadioButtons ou para o nome que deu ao formulário que adicionou por último.

FIGURA 9.5

Usando a caixa de diálogo das propriedades do projeto, você pode controlar várias configurações, incluindo o Startup Object.



Agora, se você executar o projeto, seu novo formulário (com seus quatro botões de opção) será exibido (veja a Figura 9.6).

FIGURA 9.6

Um grupo de botões de opção é uma maneira excelente de permitir que o usuário escolha uma opção entre várias.



Depois que estiverem ativos e funcionando, tente dar um clique nos botões de opção e observe que apenas um pode ser selecionado. É possível empregar a propriedade `Checked` para saber se um botão de opção foi selecionado usando um código. Para tentar acessar esses controles por meio de código, você pode adicionar um script de teste a seu novo formulário. Feche-o (utilizando o botão X do canto superior direito) e volte ao seu modo estrutura. Adicione um botão qualquer da Toolbox, altere seu nome para `btnTest`, o título para "Test" e, em seguida, dê um clique duplo nele para passar à visualização do código de seu evento-padrão (`Click`). O código da Listagem 9.7 demonstra como poderíamos verificar que conjunto de botões de opção foi selecionado e exibir o valor apropriado usando `MessageBox`. Insira esse código no evento `Click` de `btnTest` e execute seu projeto para testá-lo.

LISTAGEM 9.7 Usando Botões de Opção quando apenas uma Opção Puder Ser Selecionada

```
1 Private Sub btnTest_Click(ByVal sender As System.Object, _  
2   ByVal e As System.EventArgs) Handles btnTest.Click  
3     Dim sSelected As String  
4  
5     If RadioButton1.Checked Then  
6       sSelected = "RadioButton1"  
7     ElseIf RadioButton2.Checked Then  
8       sSelected = "RadioButton2"  
9     ElseIf RadioButton3.Checked Then  
10      sSelected = "RadioButton3"  
11     ElseIf RadioButton4.Checked Then  
12      sSelected = "RadioButton4"  
13     End If  
14  
15     MessageBox.Show(sSelected, "Selected Radio Button")  
16 End Sub  
17 End Class
```

Adicionando uma Caixa de Seleção ao Exemplo de Filer

Se você substituísse os botões de opção do exemplo anterior por caixas de seleção, poderia selecionar quantas delas desejasse ao mesmo tempo (ou mesmo nenhuma). No entanto, o código de teste ainda funcionaria, (presumindo que você desse às caixas de seleção o mesmo nome dos botões de opção ou alterasse o código incluindo nomes novos) porque as caixas de seleção também fornecem a propriedade `Checked`.

Um exemplo mais útil de uma caixa de seleção pode ser criado pela alteração do primeiro formulário que você gerou (`Filer`). Nesse formulário, se o usuário inserisse o nome de um arquivo de destino que já existisse e, em seguida, desse um clique nos botões `Copy` ou `Move`, um erro ocorreria. Como alternativa, o código poderia apenas substituir o arquivo existente sempre que al-

gum fosse encontrado, sem exibir nenhuma mensagem de erro. Você pode adicionar uma caixa de seleção a esse formulário que dê suporte à ativação e desativação desse recurso de substituição automática.

Para começar, acesse o primeiro formulário (Filer) no modo estrutura e, em seguida, adicione um novo controle Check Box da caixa de ferramentas (Toolbox). Mude o título para 'Overwrite Existing' alterando a propriedade Text na janela Properties. Neste ponto, você também pode configurar o padrão ou o valor inicial da caixa de seleção definindo a propriedade Checked na janela Properties. Mova o novo controle para a posição abaixo do campo para o destino e, então, dê um clique duplo no botão Copy para passar à edição do código. Será preciso fazer uma pequena alteração tanto em btnCopy_Click quanto em btnMove_Click antes que a caixa de seleção possa ter algum efeito real. No momento, a chamada ao comando Copy e a chamada ao comando Move usam apenas dois parâmetros (origem e destino), mas Copy dá suporte a uma opção adicional (substituir) que determina se ele deve excluir um arquivo existente caso seja encontrado. Configurando essa nova propriedade da mesma maneira que a propriedade Checked da caixa de seleção, poderemos controlar como um arquivo existente será manipulado. A Listagem 9.8 mostra a alteração em btn_Click, que manipula a nova caixa de seleção.

LISTAGEM 9.8 Adicionando uma Caixa de Seleção

```
1 Private Sub btnCopy_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles btnCopy.Click
3     Dim sSource As String
4     Dim sDestination As String
5     sSource = txtSource.Text()
6     sDestination = txtDestination.Text()
7     Try
8       File.Copy(sSource, sDestination, chkOverwrite.Checked)
9     Catch objException As Exception
10      MessageBox.Show(objException.Message)
11    End Try
12 End Sub
```

Manipular essa mesma situação no caso do comando Move é um pouco mais complicado. Em vez de apenas alterar um parâmetro, você terá de gerar uma resposta ao erro de arquivo existente e abortar a tentativa de transferência ou excluir o arquivo inadequado. A Listagem 9.9 mostra uma maneira de manipular o evento btnMove_Click.

LISTAGEM 9.9 Manipulando o Evento btnMove_Click

```
1 Private Sub btnMove_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles btnMove.Click
3     Dim sSource As String
```


LISTAGEM 9.9 Manipulando o Evento btnMove_Click (*continuação*)

```
4 Dim sDestination As String
5 sSource = txtSource.Text()
6 sDestination = txtDestination.Text()
7 If File.Exists(sSource)Then
8     If File.Exists(sDestination)Then
9         If chkOverwrite.Checked Then
10             File.Delete(sDestination)
11         Else
12             MessageBox.Show("Transferência abortada, o arquivo de destino já
13                 existe")
14             Return
15         End If
16     End If
17     File.Move(sSource, sDestination)
18 End If
19 End Sub
```

Validação de Entradas

Como o nome ‘formulário’ sugere, as janelas e caixas de diálogo são em geral usadas para permitir a entrada de dados em seu aplicativo. Com frequência, quando as informações forem inseridas no sistema, é recomendável que elas passem por alguma forma de validação (verificação de datas válidas, quantidade correta de dígitos nos números telefônicos e assim por diante). O .NET Framework fornece um modelo para codificação dessa validação.

Dado um formulário com vários campos de entrada (caixas de seleção, por exemplo), é provável que também haja um botão ou dois. Pelo menos um desses botões indicará que o usuário já considerou concluída a digitação de dados. Portanto, quando ele der um clique em OK, você desejará que o código de validação seja executado, verificando se está tudo correto nos campos. Quando o usuário der um clique no botão Cancel, não haverá motivo para se preocupar com a validação dos dados porque ele desejará cancelar o formulário.

Verificar a validade dos dados apenas quando o usuário dá um clique em OK (ou Save, ou o que for apropriado em seu aplicativo) evitará o surgimento de problemas em locais onde ele tiver de inserir dados válidos antes que você permita que encerre seu aplicativo. Trabalhei com muitos aplicativos que me forçavam a digitar um número telefônico adequado antes que me deixassem sair de uma caixa de diálogo indesejável, mesmo quando esse número era descartado por eu ter dado um clique no botão Cancel!

Vários aspectos das classes dos formulários Windows funcionam em conjunto para fornecer essa validação: a propriedade `CausesValidation` e os eventos `Validating/Validated`, que existem em todos os controles. O processo geral para o uso dessas propriedades e eventos é configu-

rar `CausesValidation` como `True` (ativado) em todos os controles de entrada efetiva de dados (caixas de texto, botões de opção, caixas de combinação e outros) e também em qualquer botão que faça com que os dados sejam usados ou salvos (OK, Save, Continue, Next e outros). Em botões como Help, Cancel, Previous ou outros que, ao serem clicados, você não precise saber se os dados são válidos, configure a propriedade `CausesValidation` como `False` (desativada). A seguir, insira um código no evento `Validating` para verificar se um campo é válido. Quando o usuário tentar alternar seu foco para um controle em que `CausesValidation = True`, o evento `Validating` será lançado em todos os controles de edição que tiverem recebido o foco desde a última vez em que um controle com `CausesValidation` igual a `True` foi acessado. Isso é um pouco complexo, portanto, percorreremos as etapas do processo:

Você possui um formulário para inserir/editar endereços (veja a Figura 9.7), com um botão OK e um Cancel, e várias caixas de texto para entrada de dados. Você configurou a propriedade `CausesValidation` de todas as caixas de texto como `True`. Também o fez no botão OK, mas não em Cancel.

FIGURA 9.7

Esta caixa de diálogo, projetada para entrada e edição de endereços, é um exemplo de um formulário de entrada de dados e está disponível para download no site da Web deste livro.

O foco no momento está na caixa de texto `txtStreet`, e o usuário pressiona Tab a fim de passar para a caixa de texto `txtCity`. Neste ponto, por causa da propriedade `CausesValidation` de `txtCity`, o evento `Validating` de `txtStreet` é acionado. Agora, o evento `Validating` deve verificar o conteúdo de `txtStreet` e se certificar de que seja válido. Ele pode cancelar a validação se os dados não forem corretos.

Se os dados estiverem corretos (e o evento não for cancelado), então, o evento `Validated` de `txtStreet` será chamado indicando que a validação foi bem-sucedida. Se os dados não estiverem corretos e o evento for cancelado, a tentativa de mover o foco também será anulada, deixando-o na caixa de texto `txtStreet`.

Portanto, sempre que o foco estiver em um controle em que `CausesValidation = True`, o evento `Validating` será lançado em todos os controles que tenham sido visitados desde a última vez que você acessou um controle em que a propriedade `CausesValidation` estivesse ativada. Isso não depende da direção, assim, se o usuário tivesse pressionado Tab ou dado um clique novamente em `txtStreet` depois de passar para `txtCity`, o evento `Validating` ainda seria chamado em `txtCity`.

Ainda está confuso? A Listagem 9.10 contém o código de vários eventos de validação para o formulário de endereços (Address Form), o que pode ajudar um pouco. O formulário completo também pode ser descarregado do site deste livro na Web, para que você não perca tempo reescrevendo-o.

LISTAGEM 9.10 Eventos de Validação do Formulário de Endereços

```
1 Private Sub txtZip_Validating(ByVal sender As Object, _
2   ByVal e As System.ComponentModel.CancelEventArgs) _
3   Handles txtZip.Validating
4   'converta o parâmetro sender em um controle da classe System.Windows.Forms.Control
5   Dim ctlSender As Control
6   ctlSender = CType(sender, Control)
7   Dim bValidPostalCode As Boolean = False
8   'Aceite como válido somente o formato de código postal dos EUA/Canadá.
9   'Verifique se txtCountry = Canada, caso contrário pressuponha EUA
10  'Manipule a possibilidade de três formatos,
11  'EUA curto #####, EUA longo #####-####, Cdn A#A#A#
12
13  Dim sPostalCode As String
14  Dim sPattern As String
15  Dim objRegex As Regex
16
17  sPostalCode = ctlSender.Text.Trim.Replace(" ", " ")
18  If txtCountry.Text.Trim.ToUpper = "CANADA" Then
19    If sPostalCode.Length = 6 Then
20      sPattern = "[ABCEGHJKLMNPRSTVXY]\d[A-Z]\d[A-Z]\d"
21    End If
22  Else
23    If sPostalCode.Length = 10 Then
24      sPattern = "\d\d\d\d\d\d-\d\d\d\d"
25    ElseIf sPostalCode.Length = 5 Then
26      sPattern = "\d\d\d\d\d"
27    End If
28    objRegex.IsMatch(sPostalCode, " ")
29  End If
30
31  If sPattern <> " " Then
32    If objRegex.IsMatch(sPostalCode, sPattern) Then
33      bValidPostalCode = True
34    End If
35  End If
36  If bValidPostalCode = False Then
37    e.Cancel = True
38    errAddress.SetError(ctlSender, "Código Postal Inválido")
39  End If
40 End Sub
41
42 Private Sub GenericValidated(ByVal sender As Object, _
43   ByVal e As System.EventArgs) _
44   Handles txtStreet.Validated, txtCity.Validated, _
```


LISTAGEM 9.10 Eventos de Validação do Formulário de Endereços (*continuação*)

```

45 txtCountry.Validated, txtZip.Validated, _
46 txtState.Validated
47     'converta o parâmetro sender em um controle da classe
      System.Windows.Forms.Control
48     Dim ctlSender As Control
49     ctlSender = CType(sender, Control)
50     'Elimine o erro, se existir algum
51     errAddress.SetError(ctlSender, " ")
52 End Sub
53
54 Private Sub GenericNotEmpty(ByVal sender As Object, _
55 ByVal e As System.ComponentModel.CancelEventArgs) _
56 Handles txtStreet.Validating, txtCity.Validating, _
57     txtState.Validating
58
59     'converta o parâmetro sender em um controle da classe
      System.Windows.Forms.Control
60     Dim ctlSender As Control
61     ctlSender = CType(sender, Control)
62     If ctlSender.Text.Trim = " " Then
63         e.Cancel = True
64         errAddress.SetError(ctlSender, "Preenchimento obrigatório")
65     End If
66 End Sub

```

Quando combinados com o controle `ErrorProvider`, que será discutido ainda nesta lição, os recursos de validação da plataforma .NET tornam fácil a criação de formulários para entrada de dados.

Usando a Classe `Message` ox

Uma caixa de diálogo é um tipo especial de formulário exibido de maneira restritiva, o que significa que o usuário precisa lidar com ela antes de poder interagir com qualquer outra parte do aplicativo. Em geral são usadas para informar algo ao usuário (como um erro) ou obter alguma informação. Elas não são empregadas na exibição de informações de status ou progresso, devido a sua natureza restritiva; devem ser utilizadas apenas quando você tiver de se comunicar de imediato com o usuário, antes de dar continuidade a algum tipo de execução.

Você mesmo pode criar caixas de diálogo como se fossem qualquer outro formulário Windows, e mostrarei como fazer isso da maneira correta mais tarde nesta lição. Em geral, no entanto, não é preciso algo complexo; só queremos formular uma pergunta simples (cujas respostas serão sim/não ou OK/cancele) ou exibir uma mensagem para o usuário. É aí que a classe `MessageBox`

entra em cena. Essa classe é usada para exibir textos em uma caixa de diálogo simples e um conjunto de botões entre os quais o usuário poderá escolher sua resposta (veja a Figura 9.). Não é possível criar uma instância dessa classe, mas ela expõe apenas um método estático, `Show`, através do qual pode-se configurar a caixa de diálogo da maneira desejada e exibi-la, tudo em uma só chamada.

FIGURA 9.8

A classe `MessageBox` é uma ferramenta útil que pode exibir uma mensagem junto a várias combinações de botões.



Parâmetros

O método `Show` aceita sete parâmetros diferentes, mas você pode fornecer apenas os valores que tiver. Esse método possui 12 tipos de sobreposições para dar suporte a várias combinações de parâmetros. Todos os parâmetros que podem ser usados são listados a seguir, com uma breve descrição de sua finalidade:

- **Text** Representando a mensagem exibida por `MessageBox`, este parâmetro não é opcional; todas as sobreposições de `Show` o incluem.
- **Caption** Outra string, como `Text`, este parâmetro determina a barra de título que será mostrada por `MessageBox`.
- **Buttons** Aceitando um dos valores possíveis enumerados, este parâmetro controla os botões mostrados na caixa de mensagens.
`MessageBoxButtons.AbortRetryIgnore` fará a caixa de diálogo exibir os botões `Abort`, `Retry` e `Ignore`; `MessageBoxButtons.YesNo` exibe os botões `Yes` e `No` e assim por diante.
- **Icon** Controla qual figura, se houver alguma, será exibida junto à mensagem. Pode ser qualquer um dos nove valores, mas as versões atuais dos sistemas operacionais só fornecem figuras para quatro deles e, portanto, todas as nove opções são convertidas em uma das quatro figuras.
- **DefaultButton** Quando você tiver mais de um botão na caixa de diálogo, só um deles poderá ser o padrão. Se o usuário pressionar `Return` ou `Enter` quando a caixa de mensagem (`MessageBox`) for aberta, ela será tratada como se ele tivesse dado um clique no botão-padrão. Este parâmetro pode ser configurado como `Button1`, `Button2` ou `Button3`, o que conduzirá a um dos botões apropriados.
- **Options** Estas opções controlam a aparência da caixa de mensagem e serão especialmente úteis quando seu aplicativo estiver localizado em outro país/região. As opções incluem tornar o texto justificado à direita, legível da direita para a esquerda e certificar que a caixa de mensagem apareça apenas na área de trabalho principal do sistema operacional.

- **OwnerWindow** Este parâmetro especifica uma janela dentro de seu aplicativo na qual a caixa de mensagem (MessageBox) deve aparecer. Em geral, esta funcionalidade não é necessária, mas está disponível.

As Figuras 9.9 e 9.10 mostram alguns exemplos de chamadas, a `MessageBox.Show()` e as caixas de diálogo resultantes.

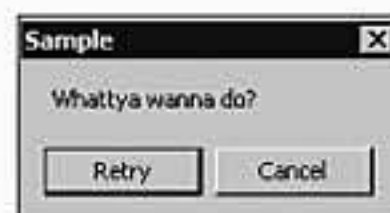
FIGURA 9.9

A caixa de mensagem-padrão, chamada apenas com o texto da mensagem (o mínimo).



FIGURA 9.10

Uma caixa de mensagem mais personalizada, especificando o texto, o título e os parâmetros dos botões.



Obtendo Resultados

Depois que você tiver escolhido os botões que quer exibir, desejará saber em qual o usuário deu um clique (pressupondo que você tenha exibido algo mais além de um botão OK). Essa informação é retornada pelo método `Show` como um valor `DialogResult`, que pode ser armazenado em uma variável, ou a chamada do método pode ser utilizada diretamente em uma expressão ou instrução condicional. A Listagem 9.11 mostra como formular uma pergunta ao usuário empregando a classe `MessageBox` e duas maneiras de manipular o resultado.

LISTAGEM 9.11 Usando a Classe `MessageBox` para Formular Perguntas Simples e, em Seguida, Atuar sobre a Resposta

```

1 Private Sub btnTest_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles btnTest.Click
3     Dim drResult As DialogResult
4     drResult = MessageBox.Show("O que deseja fazer?", _
5       "Exemplo", MessageBoxButtons.RetryCancel)
6
7     If drResult = DialogResult.Retry Then
8       'repita
9     Else
10      'cancele

```

LISTAGEM 9.11 Usando a Classe MessageBox para Formular Perguntas Simples e, em Seguida, Atuar sobre a Resposta (*continuação*)

```
11 End If
12
13 Select Case MessageBox.Show("Algo inválido ocorreu", _
14 "Processo Longo", MessageBoxButtons.AbortRetryIgnore)
15     Case DialogResult.Abort
16         'aborte
17     Case DialogResult.Retry
18         'repita
19     Case DialogResult.Cancel
20         'cancele
21     Case Else
22         'hmm...como cheguei aqui?
23 End Select
24 End Sub
```

Como demonstração do uso de MessageBox em um sistema real, você pode adicionar uma funcionalidade complementar ao formulário Filer original. Esse formulário, que permite copiar, transferir e excluir arquivos, não apresenta nenhuma mensagem de confirmação ('Está certo de que deseja excluir c:\test.txt?'), mesmo quando é solicitada a exclusão do arquivo completo. A fim de adicionar esse recurso ao formulário Filer, poderíamos usar uma caixa de mensagem para exibir a confirmação e botões Yes/No, executando ou cancelando a ação conforme a resposta. A Listagem 9.12 mostra as rotinas dos três eventos, alteradas para incluir a etapa de confirmação.

LISTAGEM 9.12 Usando o Método MessageBox.Show para Solicitar Confirmação

```
1 Private Sub btnCopy_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles btnCopy.Click
3     Dim sSource As String
4     Dim sDestination As String
5     sSource = txtSource.Text()
6     sDestination = txtDestination.Text()
7
8     If File.Exists(sSource) Then
9
10        Dim sConfirm As String
11        sConfirm = _
12        String.Format("Está certo de que deseja copiar {0} em {1}?", _
13            sSource, sDestination)
14        If MessageBox.Show(sConfirm, _
15            "Confirm Copy", _
16            MessageBoxButtons.YesNo, _
```


LISTAGEM 9.12 Usando o Método `MessageBox.Show` para Solicitar Confirmação
(*continuação*)

```
17     MessageBoxIcon.Question, _  
18     MessageBoxDefaultButton.Button2) = DialogResult.Yes Then  
19  
20     Try  
21         File.Copy(sSource, sDestination, chkOverwrite.Checked)  
22     Catch objException As Exception  
23         MessageBox.Show(objException.Message)  
24     End Try  
25 End If  
26 End If  
27 End Sub  
28  
29 Private Sub btnMove_Click(ByVal sender As System.Object, _  
30     ByVal e As System.EventArgs) Handles btnMove.Click  
31     Dim sSource As String  
32     Dim sDestination As String  
33     sSource = txtSource.Text()  
34     sDestination = txtDestination.Text()  
35     If File.Exists(sSource) Then  
36         Dim sConfirm As String  
37         sConfirm = String.Format(_  
38             "Está certo de que deseja transferir {0} para {1}?", _  
39             sSource, sDestination)  
40  
41         If MessageBox.Show(sConfirm, _  
42             "Confirm Move", MessageBoxButtons.YesNo, _  
43             MessageBoxIcon.Question, _  
44             MessageBoxDefaultButton.Button2) = DialogResult.Yes Then  
45  
46             If File.Exists(sDestination) Then  
47                 If chkOverwrite.Checked Then  
48                     File.Delete(sDestination)  
49                 Else  
50                     MessageBox.Show("Transferência abortada, arquivo de destino já  
51                         existe")  
52                     Return  
53                 End If  
54             End If  
55             File.Move(sSource, sDestination)  
56         End If  
57     End If  
58 End Sub  
59
```

LISTAGEM 9.12 Usando o Método `MessageBox.Show` para Solicitar Confirmação
(*continuação*)

```
60 Private Sub btnDelete_Click(ByVal sender As Object, _  
61     ByVal e As System.EventArgs) Handles btnDelete.Click  
62     Dim sSource As String  
63     sSource = txtSource.Text()  
64  
65     If File.Exists(sSource) Then  
66         Dim sConfirm As String  
67         sConfirm = String.Format(_  
68             "Está certo de que deseja excluir {0}?", _  
69             sSource)  
70  
71         If MessageBox.Show(sConfirm, _  
72             "Confirm Delete", MessageBoxButtons.YesNo, _  
73             MessageBoxIcon.Question, _  
74             MessageBoxDefaultButton.Button2) = DialogResult.Yes Then  
75  
76             File.Delete(sSource)  
77         End If  
78     End If  
79 End Sub
```

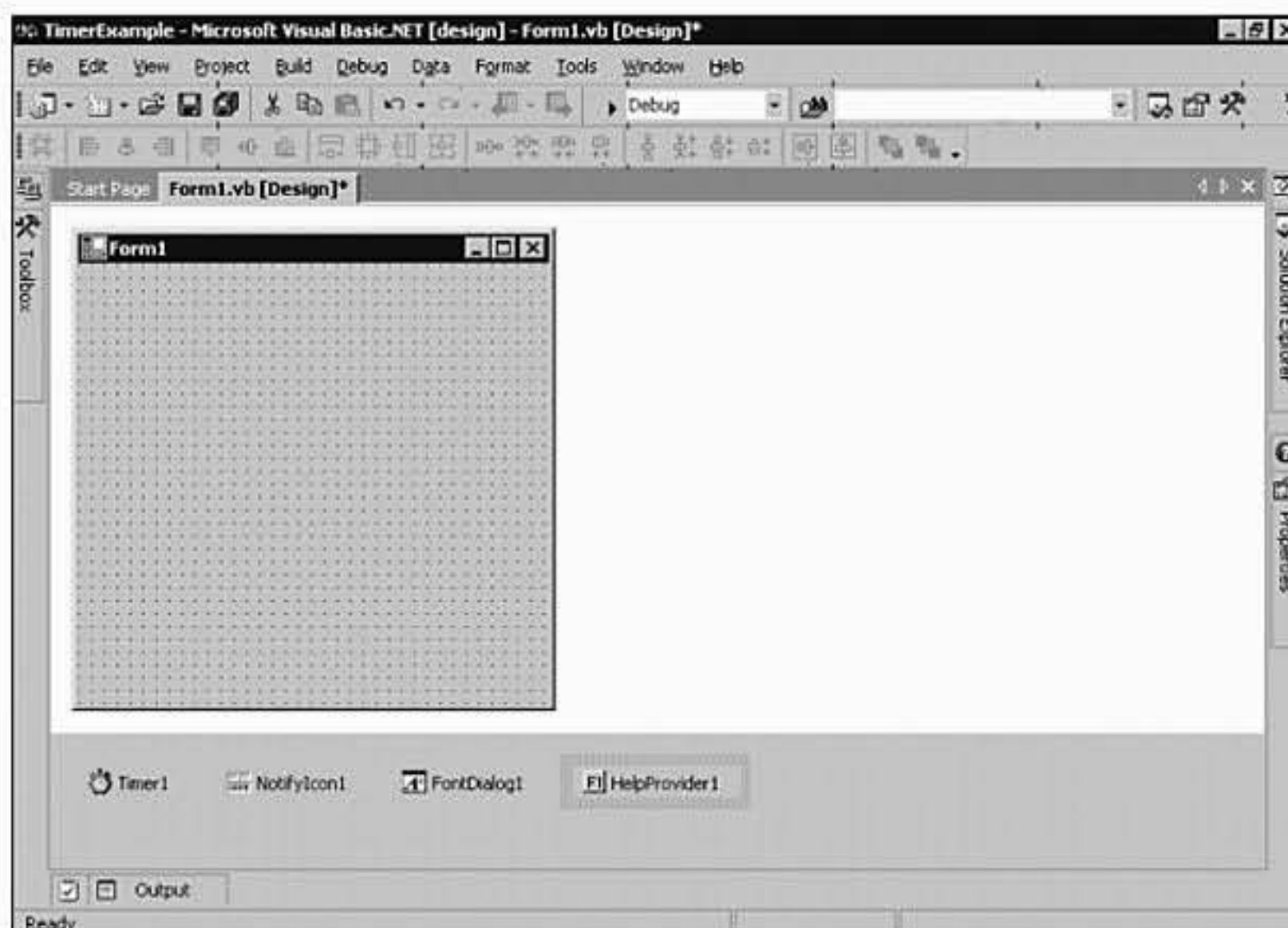
Controles Ocultos

Todos os controles usados nesta lição até agora estavam visíveis quando seu projeto foi executado. Esse é o tipo padrão de controle com o qual você trabalhará. No entanto, a plataforma .NET também apresenta controles que são iguais aos habituais em quase tudo, exceto por não possuírem uma interface visível no tempo de execução. Esses controles também têm propriedades, métodos e eventos e foram projetados como uma maneira modular fácil de adicionar recursos específicos a um formulário.

Antes do Visual Basic .NET, os controles ocultos também podiam ser inseridos nos formulários, complicando a interface do tempo de projeto, mas ficando totalmente invisíveis no tempo de execução. No Visual Studio .NET encontramos uma maneira muito melhor de manipular esse tipo de controle, posicionando-os em uma área separada abaixo da do projeto do formulário (veja a Figura 9.11). Você ainda poderá arrastar, selecionar, clicar e excluí-los, porém sem que interfiram na visualização do projeto de seu formulário.

FIGURA 9.11

No Visual Studio .NET, os controles ocultos estão posicionados em uma área especial do projeto, para evitar serem confundidos com elementos da interface visível.



Há vários desses controles de tempo de projeto, mas abordarei nesta seção alguns dos mais usados: Timer, NotifyIcon, ErrorProvider e os controles de caixas de diálogo.

Timer

O controle Timer foi projetado para permitir que os códigos sejam executados a certos intervalos com base em períodos de tempo. Se o controle estiver ativado, ele acionará automaticamente seu evento Tick a intervalos regulares. Pela inserção de um código no manipulador do evento Tick, você poderá executar a tarefa que desejar de maneira programada.

O uso do controle Timer envolve apenas algumas etapas:

1. Depois que você adicionar esse controle a seu formulário, terá de configurar suas propriedades; configure Enabled como True para indicar que o deseja ativo, e a propriedade Interval com o período de tempo (em milissegundos – 1000 ms são iguais a 1s) que quer entre os eventos.
2. Adicione seu código ao evento Tick (dê um clique duplo no controle Timer para acessar rapidamente o manipulador dele).

Isso é tudo o que se tem a fazer para que o código inserido no manipulador do evento Tick seja executado uma vez a cada valor de Interval em milissegundos.

Você pode testar esse controle em um projeto de sua autoria seguindo estas etapas:

1. Crie um novo projeto vazio – um aplicativo Windows. Um formulário em branco será adicionado ao projeto, Form1. Certifique-se de estar visualizando o modo estrutura desse formulário vazio.

2. Arraste um controle Timer de Toolbox (no Windows Forms) para o formulário. Ele será adicionado como um controle à área de controles ocultos (que agora aparecerão) com o nome Timer1.
3. Dê um clique em Timer1 para selecioná-lo e, em seguida, visualize a janela de propriedades, que deve mostrar a você os valores de Enabled e Interval que determinam o comportamento do período de tempo.
4. Configure Enabled como True, e Interval igual a 1000 (1 segundo).
5. Dê um clique duplo no controle Timer para passar à visualização do código, e acesse a rotina de manipulação do evento Tick.

Adicione o código mostrado na Listagem 9.13 ao manipulador do evento Tick.

LISTAGEM 9.13 Código Que Fará com Que o Título do Formulário Aja como um Relógio

```
1 Private Sub Timer1_Tick(ByVal sender As System.Object, _  
2   ByVal e As System.EventArgs) Handles Timer1.Tick  
3  
4     Me.Text = DateTime.Now.ToString()   
5  
6 End Sub
```

Execute o projeto (pressionando F5 ou selecionando Start no menu Debug), certificando-se de que ele será o iniciado (dê um clique com o botão direito do mouse sobre o projeto no Solution Explorer e selecione Set em StartUp Project) caso tenha mais de um projeto aberto.

Quando o projeto for executado, o formulário deverá aparecer, e seu título refletirá a hora atual até os segundos. Faça o teste com valores diferentes em Interval para ver como essa propriedade afeta o acionamento do evento Tick. Para que fique um pouco mais divertido, adicione um botão ao formulário e insira essa linha de código em seu evento Click:

```
Timer1.Enabled = Not Timer1.Enabled
```

Tente executar o projeto novamente e dê um clique no botão algumas vezes para ver o que acontece.

NotifyIcon

Quando um programa precisa ser executado continuamente e é necessário algum tipo de notificação visual, um método comum é inserir um ícone na bandeja do sistema ou área da notificação à extrema direita da barra de tarefas. Esse pode não ser o melhor local para posicionar algum aplicativo, é uma área que já está muito cheia, mas é útil para certos utilitários com finalidades especiais.

Antes da existência da plataforma .NET, adicionar um ícone à bandeja do sistema envolvia o uso de várias chamadas à API do Win 32 e uma técnica denominada subclassificação que poderia tornar instável seu aplicativo do Visual Basic. Na plataforma .NET, tudo que você precisa fazer é adicionar um controle `NotifyIcon` a seu formulário e configurar suas propriedades. A propriedade mais importante é `Icon`, que pode ser configurada com qualquer arquivo de ícone para controlar o que irá aparecer na área de notificação. Se quisermos que um menu surja quando o usuário der um clique com o botão direito do mouse em seu ícone, podemos adicionar um controle `ContextMenu` ao formulário e configurar a propriedade `ContextMenu` de `NotifyIcon`. Seja cuidadoso para só usar esse controle quando for apropriado; há tantos aplicativos colocando ícones nessa área que foi adicionado ao Windows XP um recurso de ocultação automática!

ErrorProvider

Outro grande recurso para o projeto de formulários de entrada de dados, o controle `ErrorProvider` permite que você indique visualmente (veja a Figura 9.12) que controles de um formulário possuem erros associados a eles.

Para usar este controle, apenas insira-o em seu formulário e configure as propriedades. Na verdade, você pode fazer isso com ainda menos trabalho porque os valores-padrão costumam ser adequados para grande parte dos objetivos, a menos que queira empregar um ícone diferente. Então, sempre que quiser indicar que um controle possui um erro associado a ele, chame o método `SetError` do controle `ErrorProvider`:

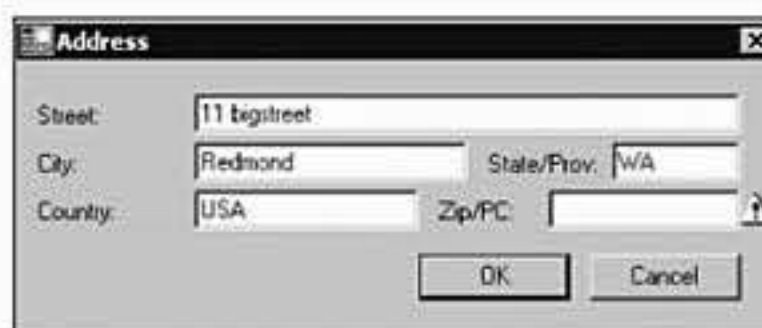
```
errorProvider1.SetError(txtStreet, "Endereço da rua inválido")
```

Quando o erro tiver sido corrigido, você poderá eliminá-lo de `ErrorProvider` configurando-o com uma string vazia:

```
errorProvider1.SetError(txtStreet, " ")
```

FIGURA 9.12

Com um controle `ErrorProvider` em seu formulário, será fácil mostrar aos usuários os erros de entrada de dados.



A caixa de diálogo do exemplo (de entrada/edição de endereços) abordado rapidamente nesta lição (e que pode ser descarregado do site da Web deste livro) usa esse controle oculto em seu código e é uma boa demonstração a ser examinada.

Controles das Caixas de Diálogo

O último tipo de controle oculto abordado nesta lição na verdade se refere a todo um grupo de controles que vão de Open e Save às caixas de diálogo de seleção de fontes. Esses controles permitem que sejam usadas várias caixas de diálogo-padrão do Windows em seu programa (veja a Figura 9.13), o que é muito mais adequado do que ter de você mesmo criá-las.

FIGURA 9.13

A caixa de diálogo para abertura de arquivos manipula toda a comunicação com o sistema operacional e o sistema de arquivos, sem que você tenha de escrever nenhum código.



Como com os outros controles ocultos, é possível arrastar qualquer caixa de diálogo para que seja usada em seu formulário ou em uma área de controles ocultos. Depois que tiver uma instância do controle em seu formulário, comece a trabalhar com as propriedades dele a fim de configurá-lo para seu aplicativo. Conduzirei você através do exemplo de um programa que emprega as caixas de diálogo para abrir e salvar arquivos, e também poderei utilizar as de fonte e cor, apenas para ser conclusivo.

O próprio formulário contém vários controles incluindo uma caixa Rich Text, quatro botões e um controle de painel, que é usado para armazenar os botões. Empregar um controle de painel dessa maneira, e configurá-lo para que fique encaixado à direita e que a caixa Rich Text ocupe todo o formulário, fornecerá uma interface que será alterada de modo correto quando o formulário for redimensionado ou maximizado. Não abordarei os detalhes da configuração desses controles, mas o formulário completo pode ser descarregado a partir do site deste livro na Web.

Além dos controles visíveis, também inseri os quatro controles das caixas de diálogo (Open, Save, Font e Color) no formulário. Cada um dos quatro botões usa uma das caixas de diálogo para abrir arquivos, salvar, alterar a fonte ou mudar a cor dela, e seu código está listado nas seções a seguir. Observe que configurei os controles das caixas de diálogo dentro do procedimento de dar um clique no botão, muito embora pudesse ter configurado muitas dessas propriedades empregando a janela de propriedades do controle de caixa de diálogo. Só agi assim porque esse é um exemplo; qualquer propriedade de controle que seja a mesma durante toda a existência do formulário deve ser configurada apenas uma vez, que é o que ocorre quando se manipulam as

propriedades por meio da janela referente a elas. Agora percorrerei o código que representa cada um dos quatro botões e discutirei como o controle de caixa de diálogo foi utilizado em cada caso.

Caixa de Diálogo Open

O manipulador de eventos do botão Open (veja a Listagem 9.14) permite que o usuário selecione um nome de arquivo usando a caixa de diálogo Open e, em seguida, exibe esse arquivo no controle da caixa Rich Text.

LISTAGEM 9.14 Manipulador de Eventos do Botão Open

```

1 Private Sub btnOpen_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles btnOpen.Click
3     Dim sNewFile As String
4     Dim trNewFile As System.IO.TextReader
5     If MessageBox.Show("Sobrepor o conteúdo atual?", _
6       "Open New File", MessageBoxButtons.YesNo, MessageBoxIcon.Question, _
7       MessageBoxDefaultButton.Button2) = DialogResult.Yes Then
8       With dlgOpen
9         .Filter = "Text files (*.txt)|*.txt"
10        .AddExtension = True
11        .CheckFileExists = True
12        .CheckPathExists = True
13        .InitialDirectory = IO.Path.GetDirectoryName(sCurrentFile)
14        .Multiselect = False
15
16        If .ShowDialog()=DialogResult.OK Then
17          sNewFile = .FileName
18          If IO.File.Exists(sNewFile)Then
19            trNewFile = New IO.StreamReader(sNewFile)
20            rtfContents.Text = trNewFile.ReadToEnd()
21            sCurrentFile = sNewFile
22          End If
23        End If
24      End With
25    End If
26  End Sub

```

ANÁLISE

As linha 2 à 7, depois de declararem as variáveis necessárias, confirmam se o usuário quer carregar um novo arquivo, sobrepondo, portanto, o conteúdo atual da caixa de texto. Uma caixa de mensagem é usada para formular essa questão porque ela resulta em uma resposta simples Yes/No. Se o usuário responder Yes, siga em frente e sobreponha o conteúdo atual. Em seguida, a caixa de diálogo Open é elaborada das linhas 8 à 14. Um filtro é configurado para restringir os tipos de arquivo aos somente de texto. É informado a caixa de diálogo para que retor-

ne a extensão junto ao nome do arquivo, a fim de que seja verificado se o caminho e o arquivo existem e para que ele seja iniciado na mesma pasta do último arquivo carregado. A propriedade `MultiSelect` também está configurada como `False`, indicando que você deseja que o usuário só possa selecionar um arquivo por vez.

A linha 16 exibe a caixa de diálogo e verifica se o resultado está correto. Se estiver, as linhas 18 à 20 se certificam novamente da existência do arquivo e, em seguida, o carregam em uma única leitura na caixa Rich Text. Deveriam ser adicionados mais recursos de manipulação de erros aqui se esse fosse um sistema de produção, porque a abertura de arquivos é uma situação propensa a falhas devido a privilégios de segurança e outras razões.

Caixa de Diálogo Save

Quando você precisar que o usuário forneça o nome de um arquivo de destino, a caixa de diálogo Save será necessária. Ela faz parte do sistema operacional, portanto, tem conhecimento das unidades, pastas, compartilhamentos de arquivos, links de atalho e o que o mais o Windows possa lançar nela. Escrever uma caixa de diálogo desse tipo seria difícil e precisaríamos de uma manutenção contínua, já que os recursos do sistema de arquivos do sistema operacional são alterados com o tempo. O uso das caixas de diálogo internas torna seu sistema melhor integrado ao sistema operacional (veja a Listagem 9.15).

LISTAGEM 9.15 Usando as Caixas de Diálogo Internas

```
1 Private Sub btnSave_Click(ByVal sender As System.Object, _  
2   ByVal e As System.EventArgs) Handles btnSave.Click  
3  
4     Dim sNewFileName As String  
5     Dim swOutput As IO.StreamWriter  
6  
7     sNewFileName = sCurrentFile  
8  
9     If MessageBox.Show("Salvar com um nome diferente de arquivo?", _  
10    "Save File", MessageBoxButtons.YesNo, MessageBoxIcon.Question, _  
11    MessageBoxDefaultButton.Button2) = DialogResult.Yes Then  
12         With dlgSave  
13             .FileName = sCurrentFile  
14             .CheckFileExists = False  
15             .CheckPathExists = True  
16             .DefaultExt = ".txt"  
17             .Filter = "Text files (*.txt)|*.txt"  
18             .AddExtension = True  
19             .InitialDirectory = IO.Path.GetDirectoryName(sNewFileName)  
20             .OverwritePrompt = True  
21             .CreatePrompt = False  
22
```


LISTAGEM 9.15 Usando as Caixas de Diálogo Internas (*continuação*)

```

23         If .ShowDialog()= DialogResult.OK Then
24             sNewFileName = .FileName
25             swOutput = New IO.StreamWriter(sNewFileName)
26
27             swOutput.Write(rtfContents.Text)
28             swOutput.Close()
29             sCurrentFile = sNewFileName
30         End If
31     End With
32 End If
33 End Sub

```

ANÁLISE

O uso da caixa de diálogo Save é semelhante ao de Open. Depois que o usuário confirma se deseja salvar o arquivo (linha 9), as propriedades da caixa de diálogo são configuradas para criar a imagem adequada. O nome do arquivo é configurado como padrão com o último utilizado. A caixa de diálogo é informada de que não é necessário ser um arquivo já existente (`CheckFileExists = False`), mas a pasta sim (`CheckPathExists = True`). Por padrão, o arquivo será salvo com a extensão `.txt` (linha 16), mas a caixa de diálogo não impedirá o usuário de salvá-lo como desejar. Duas opções interessantes, `OverwritePrompt` e `CreatePrompt`, não são aplicadas à caixa de diálogo Open, mas são importantes ao salvar. `OverwritePrompt` controla se a caixa de diálogo avisará o usuário antes de permitir que ele salve utilizando o caminho e o nome de um arquivo existente. `CreatePrompt` determina se o usuário deve ser avisado caso tente fazer exatamente o contrário (usar um caminho e um nome que ainda não existam).

Depois de pronta, a caixa de diálogo é exibida (linha 23), e se o usuário sair dela dando um clique em OK, então, o conteúdo da caixa Rich Text será gravado no arquivo (linhas 24 à 28).

Caixa de Diálogo Font

Os nomes, tamanhos disponíveis e estilos das fontes da máquina de um usuário podem ser um conjunto razoável de informações, e essa caixa de diálogo manipula tudo isso para você enquanto fornece uma interface que provavelmente aparece em muitos dos programas com as quais o usuário trabalha. A Listagem 9.16 mostra como a caixa de diálogo Font pode ser usada para permitir que o usuário selecione a fonte e o estilo de uma fonte.

LISTAGEM 9.16 Usando a Caixa de Diálogo Font

```

1 Private Sub btnFont_Click(ByVal sender As System.Object, _
2     ByVal e As System.EventArgs) Handles btnFont.Click
3     dlgFont.Font = rtfContents.Font
4
5     If dlgFont.ShowDialog()= DialogResult.OK Then

```


LISTAGEM 9.16 Usando a Caixa de Diálogo Font (*continuação*)

```
6         rtfContents.Font = dlgFont.Font
7     End If

8 End Sub
```

ANÁLISE

A caixa de diálogo Font é simples de usar: apenas carregue as configurações de fonte atuais nela (linha 3) e, em seguida, exiba a caixa de diálogo (linha 5). Se o usuário der um clique em OK, então, pegue as novas configurações de fonte e leve-as de volta ao destino (nesse caso a caixa Rich Text, linha 6). Isso é tudo; funciona perfeitamente.

Caixa de Diálogo Color

A menos que precise de alguma funcionalidade avançada, a caixa de diálogo Color é exatamente do que você necessita para permitir que o usuário escolha uma cor (veja a Listagem 9.17). Ele pode selecionar uma cor-padrão predefinida ou criar sua própria combinação de cores, tudo por meio dessa caixa de diálogo.

LISTAGEM 9.17 Adicionando uma Seleção Gráfica de Cores a um Aplicativo

```
1 Private Sub btnColor_Click(ByVal sender As System.Object, _
2     ByVal e As System.EventArgs) Handles btnColor.Click
3     dlgColor.Color = rtfContents.ForeColor
4
5     If dlgColor.ShowDialog() = DialogResult.OK Then
6         rtfContents.ForeColor = dlgColor.Color
7     End If
8
9 End Sub
```

ANÁLISE

A caixa de diálogo Color funciona exatamente da mesma maneira que Font; simplesmente carregue os valores das cores atuais (linha 3), exiba a caixa de diálogo (linha 5) e configure o destino que usará os valores selecionados nela – contanto que o usuário dê um clique em OK (linha 6).

Construindo Suas Caixas de Diálogo

A classe MessageBox fornece uma maneira de exibir caixas de diálogo simples, mostrando uma mensagem e/ou solicitando ao usuário para que faça uma escolha em um conjunto fixo de opções (Yes/No, OK/Cancel, Abort/Retry/Ignore). Mas haverá vezes em que você precisará de uma caixa de diálogo com recursos mais complexos. É possível transformar qualquer formulário Win-

dows em uma caixa de diálogo, dando a ele a aparência e o comportamento de uma, e usá-la em seu programa. Existem vários estágios diferentes, e os percorrerei enquanto mostro como construir uma caixa de diálogo a ser empregada como a tela de login de um aplicativo. Para acompanhar este exemplo ao avançar nesta seção, crie um novo projeto (do tipo aplicativo Windows).

Criando a Caixa de Diálogo

Uma caixa de diálogo não só se comporta de modo diferente de um formulário Windows comum, mas ela também possui outra aparência. Adicione um novo formulário a seu projeto, chamado `LogonDialog`, e acesse-o no modo estrutura. Dê um clique no formulário e acesse a janela de propriedades. Para fazer com que ele se pareça com uma caixa de diálogo, configure as propriedades a seguir:

- `FormBorderStyle = FixedDialog` Isto impedirá que a caixa de diálogo seja redimensionada.
- `Text = "Logon"` Este é o título, e `Logon` é mais apropriado do que `LogonDialog`.
- `MaximizeBox` e `MinimizeBox` configuradas como `False` Não é necessário minimizar ou maximizar uma caixa de diálogo.

Agora adicione os controles necessários para criar uma caixa de diálogo de logon, duas caixas de texto (`UserID` e `Password`), dois títulos (um para cada caixa de texto) e dois botões (`OK` e `Cancel`). A maneira como irá organizá-los não é tão importante para os fins de um exemplo, mas a Figura 9.14 mostra como fiz.

FIGURA 9.14

Este formulário foi redimensionado para ter uma aparência comum somente com alguns controles nele.



Nomeie os controles usando as convenções de nomeação que viu no decorrer desta lição, gerando `lblUserid`, `lblPassword`, `txtUserid`, `txtPassword`, `btnOK` e `btnCancel`. Para concluir, como última parte da configuração, visualize as propriedades de `txtPassword` e configure `PasswordChar = "*"` . Com `PasswordChar` configurada, qualquer texto que for inserido nesse campo será mostrado como uma string desse caractere.

Já que os dois botões foram adicionados ao formulário, você poderá voltar às propriedades do formulário e configurar duas que não poderiam ser definidas sem algum botão disponível. Configure `AcceptButton` e `CancelButton`, respectivamente como `btnOK` e `btnCancel`. O resultado da configuração dessas duas propriedades é que se o usuário pressionar `Enter` nessa caixa de diálogo, o efeito produzido será o mesmo de dar um clique no botão `OK`. Se o usuário pressionar a tecla `Escape`, isso terá o mesmo resultado de dar um clique no botão `Cancel`.

Configurando o Resultado da Caixa de Diálogo

Quando a caixa de diálogo for exibida e o usuário a fechar dando um clique em OK ou em Cancel, seu programa terá de determinar dois itens: primeiro, se o usuário deu um clique em OK ou em Cancel e, em segundo lugar, o que o usuário inseriu nas caixas UserID e Password. A primeira informação, que botão foi pressionado, é semelhante ao que é retornado de uma chamada a `MessageBox.Show()`, e sua caixa de diálogo a manipulará da mesma maneira. Há uma propriedade `DialogResult` no formulário, e qualquer valor configurado nela será retornado para o programa que exibiu a caixa de mensagem. Você pode configurar essa propriedade usando apenas uma linha de código como a que aparece a seguir:

```
Me.DialogResult = DialogResult.Cancel
```

Há outra maneira de configurar esse resultado, que é definir a propriedade `DialogResult` de `btnOK` e `btnCancel` (como OK e Cancel, respectivamente). Se esses valores forem configurados, então, a caixa de diálogo será fechada automaticamente quando o usuário der um clique em um dos botões, e seu resultado será definido como o valor da caixa de diálogo do botão. É tudo uma questão de controle e de onde você planejou executar a validação.

Se a propriedade `CausesValidation` for configurada como `True` em `txtUserId`, `txtPassword`, e `btnOK` (e como `False` em `btnCancel`), então, poderemos usar o evento `Validating` em `txtUserId` e `txtPassword` para verificar a entrada do usuário. Como alternativa, você pode querer executar toda a verificação no evento `Click` de `btnOK`, situação na qual seria desejável que `btnOK.DialogResult` fosse configurada como `None`, para que a caixa de diálogo não fosse fechada automaticamente quando dessem um clique no botão. Em código, é sempre possível fecharmos nós mesmos o formulário configurando a propriedade `DialogResult` diretamente. Em qualquer um dos métodos, podemos deixar a propriedade `DialogResult` de `btnCancel` configurada como `Cancel`, porque nenhuma validação deve ocorrer se o usuário der um clique nesse botão.

Qual dessas duas maneiras é a melhor? Em um formulário com apenas dois campos de texto, provavelmente não fará diferença, mas se você tivesse uma caixa de diálogo grande com um nível alto de validação, seria recomendável usar os eventos de validação e a propriedade `CausesValidation`.

A listagem 9.18 mostra o código para o evento `Click` de `btnOK`, validando os dados inseridos nas duas caixas de diálogo e configurando a propriedade do resultado da caixa de diálogo do formulário. Esse código pressupõe `CausesValidation` igual a `False` para tudo que existir no formulário, e a propriedade `DialogResult` de `btnOK` igual a `None`.

LISTAGEM 9.18 Usando as Propriedades de Validação

```
1 Private Sub btnOK_Click(ByVal sender As System.Object, _  
2     ByVal e As System.EventArgs) Handles btnOK.Click  
3     Dim sUserID As String  
4     Dim sPassword As String
```


LISTAGEM 9.18 Usando as Propriedades de Validação (*continuação*)

```

5
6     sUserID = txtUserid.Text
7     sPassword = txtPassword.Text
8
9     If sUserID.Trim()= " " Then
10        MessageBox.Show("Identificação do usuário é obrigatória,"
11        & " insira uma identificação de usuário adequada.", "Error", _
12        MessageBoxButtons.OK, MessageBoxIcon.Error)
13        txtUserid.Select()
14    Else
15        If sPassword.Trim() = " " Then
16            MessageBox.Show("A senha é obrigatória, " & _
17            "insira uma senha apropriada.", "Error", _
18            MessageBoxButtons.OK, MessageBoxIcon.Error)
19            txtPassword.Select()
20        Else
21            Me.DialogResult = DialogResult.OK
22        End If
23    End If
24 End Sub

```

ANÁLISE

A rotina `Click` não executa nenhuma validação complexa; apenas verifica se o usuário inseriu uma identificação e uma senha (linhas 9 e 15) e configura o resultado da caixa de diálogo como OK (linha 21) se isso tiver ocorrido. A configuração da propriedade `DialogResult` do formulário na linha 21 fecha a caixa de diálogo para ocultar e retornar esse resultado ao programa que o chamou. Observe que se `btnOK.DialogResult` tivesse sido configurada como OK (por meio da janela de propriedades), então, a caixa de diálogo seria fechada ao término de seu evento `Click` independentemente de ter sido inserida a identificação/senha do usuário.

Configure a propriedade `btnCancel.DialogResult` como `Cancel` (de modo que sempre execute o cancelamento, e nenhuma manipulação de evento seja necessária) por meio da janela de propriedades, e essa caixa de diálogo de logon estará pronta.

Exibindo a Caixa de Diálogo

Para usar sua nova caixa `LogonDialog`, você precisa que ela seja exibida. Configurar esse formulário como o objeto a ser iniciado em seu projeto não seria útil porque ele executaria a caixa de diálogo, mas o aplicativo seria encerrado quando o usuário inserisse uma identificação e uma senha. É preciso chamar `LogonDialog` antes de exibir `Form1`, que é o objeto de inicialização desse projeto. Exatamente como os botões, os formulários também possuem eventos, e um em particular, `Load`, é perfeito para esse propósito porque é chamado antes de o formulário ser exibido. Visualize o código de `Form1` e selecione `Base Class Events` no menu suspenso da esquerda, na parte

superior da janela de edição de códigos. Selecionar essa opção preencherá a lista suspensa à direita com a relação de eventos aos quais seu formulário dá suporte. Selecione Load na lista suspensa, e poderá editar o procedimento do evento Form1_Load.

Nesse procedimento, você precisará criar uma instância de seu formulário LogonDialog, exibí-lo como uma caixa de diálogo e, em seguida, verificar a identificação/senha que o usuário inseriu. A Listagem 9.19 fornece um exemplo de como esse procedimento de evento poderia ser escrito.

LISTAGEM 9.19 Exibindo um Formulário como uma Caixa de Diálogo

```
1 Private Sub Form1_Load(ByVal sender As Object, _  
2     ByVal e As System.EventArgs) Handles MyBase.Load  
3     Dim frmLogon As New LogonDialog()  
4     Dim bLogonSuccessful As Boolean = False  
5     Dim sFailureMessage As String  
6  
7     If frmLogon.ShowDialog() = DialogResult.OK Then  
8  
9         If frmLogon.txtUserid.Text = "Duncan" Then  
10            If frmLogon.txtPassword.Text = "password" Then  
11                bLogonSuccessful = True  
12            Else  
13                sFailureMessage = "Senha Inválida!"  
14            End If  
15        Else  
16            sFailureMessage = "Identificação do Usuário Inválida!"  
17        End If  
18    Else  
19        sFailureMessage = "Tentativa de Logon Cancelada"  
20    End If  
21  
22    If Not bLogonSuccessful Then  
23        MessageBox.Show(sFailureMessage, "Não foi possível efetuar logon", _  
24            MessageBoxButtons.OK, MessageBoxIcon.Error)  
25        Me.Close()  
26    End If  
27 End Sub
```

ANÁLISE

Uma nova instância de LogonDialog é criada (linha 3), o que é necessário antes que você possa exibir esse formulário. Duas variáveis são usadas (linhas 4 e 5) para registrar o sucesso/falha da tentativa de conexão. A linha 7 emprega o método ShowDialog do formulário para exibi-lo de maneira restrita. Devido a ShowDialog, a próxima linha de código desse procedimento não será executada até que o formulário esteja oculto ou fechado, mas usar apenas Show teria exibido o formulário de maneira não restrita, e o processamento do código teria continuado antes que o usuário concluísse o logon.

O método `ShowDialog`, exatamente como `Show` na classe `MessageBox`, retorna um valor de `DialogResult`, como discutimos anteriormente enquanto você criava o formulário `LogonDialog`. A linha 7 verifica o valor retornado, porque se o usuário cancelar a caixa de diálogo, sua identificação e senha não devem ser processadas. Em seguida, os controles de caixa de texto de `LogonDialog` são acessados e seus valores verificados em relação à identificação e senha do usuário embutidas no código (linhas 9 e 10). Em seus programas reais, você provavelmente verificará esses valores confrontando-os com algum tipo de banco de dados ou lista de segurança. Para concluir, se tudo estiver correto, `bLogonSuccessful` será configurado como `True` (linha 11), ou a mensagem de erro apropriada será utilizada por `sFailureMessage` (linha 11 à 19). Se a conexão não tiver êxito, a mensagem de erro será exibida em uma caixa de mensagens (linha 23), e `Form1` será fechado (linha 25) para bloquear o acesso ao programa.

Resumo

Os formulários Windows são usados para o desenvolvimento de aplicativos .NET com interfaces do Windows, que é um dos dois tipos de aplicativo que você criará (o outro seria os sistemas com base na Web). Contendo vários grandes avanços não existentes em versões anteriores do Visual Basic e fornecendo um sistema comum para o desenvolvimento de formulários por toda a plataforma .NET, o sistema Windows Forms permitirá que você construa a interface que seu aplicativo precisar. O Dia 16 dará continuidade a esse tópico e fornecerá mais informações sobre o uso desses formulários em seus programas.

P&R

P Posso ter mais de um formulário Windows em um projeto/aplicativo?

R Sim, você pode ter quantos desejar. No entanto, apenas um formulário pode ser definido para a inicialização, de modo que será preciso criar instâncias dos outros dentro de seu código e, então, chamar seu método `Show` quando quiser exibi-los.

P Quero usar uma caixa de mensagem (Message box) em um aplicativo do console que criei, mas não consigo fazê-lo funcionar. Ele parece não reconhecer o nome da classe.

R A classe `MessageBox` faz parte do espaço de nome `System.Windows.Forms`, portanto, você deve tê-lo disponível antes de usá-la. Quando criar um aplicativo Windows empregando o Visual Studio, ele automaticamente adicionará a referência necessária, mas não fará isso para um aplicativo do console. Utilizando o Solution Explorer, dê um clique com o botão direito do mouse na pasta `References` dentro de seu projeto e selecione `Add Reference`. Na caixa de diálogo que aparecerá, selecione `System.Windows.Forms.dll` na lista. Outra boa idéia é adicionar uma linha `Imports System.Windows.Forms` no início de seu arquivo de classes ou módulo de modo que possa se referir à `MessageBox` sem ter de sempre usar o prefixo `System.Windows.Forms` nessa classe.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Qual é a diferença entre um formulário modal e um não-modal?
2. Por que não podemos configurar a propriedade `CausesValidation` de um botão `Cancel` com o valor `True`?
3. Que instrução é adicionada no final da declaração de um procedimento para indicar que ele é um manipulador de eventos?

Exercícios

1. Dado que um único procedimento pode manipular vários eventos (`Sub myhandler() Handles btnOpen.Click, btnClose.Click, btnSave.Click`), como você poderia reescrever esses três procedimentos de eventos em um único procedimento?

Dica: Você precisa converter (usando `CType`) o parâmetro `sender` em um controle de `System.Windows.Forms.Control` ou `System.Windows.Forms.Button` antes de poder acessar o conjunto comum de propriedades de controle.

LISTAGEM 9.20 Usando a Palavra-Chave `Handles` para criar um único procedimento chamado por vários eventos

```
1 Private Sub btnCopy_Click(ByVal sender As System.Object, _  
2     ByVal e As System.EventArgs) Handles btnCopy.Click  
3     Dim sSource As String  
4     Dim sDestination As String  
5     sSource = txtSource.Text()  
6     sDestination = txtDestination.Text()  
7     File.Copy(sSource, sDestination)  
8 End Sub  
9  
10 Private Sub btnMove_Click(ByVal sender As System.Object, _  
11     ByVal e As System.EventArgs) Handles btnMove.Click  
12     Dim sSource As String  
13     Dim sDestination As String  
14     sSource = txtSource.Text()  
15     sDestination = txtDestination.Text()  
16     File.Move(sSource, sDestination)  
17 End Sub  
18
```


LISTAGEM 9.20 Usando a Palavra-Chave `Handles` para criar um único procedimento chamado por vários eventos (*continuação*)

```
19 Private Sub btnDelete_Click(ByVal sender As Object, _  
20     ByVal e As System.EventArgs) Handles btnDelete.Click  
21     Dim sSource As String  
22     sSource = txtSource.Text()  
23     File.Delete(sSource)  
24 End Sub
```


PÁGINA EM BRANCO

SEMANA 2

DIA 10

Construindo a Interface com o Usuário com os Formulários da Web

Hoje cada vez mais aplicativos são escritos com base em navegadores. Antes do Visual Basic .NET, era difícil criar esses aplicativos com base em navegador usando o Visual Basic. Com o Visual Basic .NET, seu desenvolvimento se tornou tão fácil quanto o de aplicativos com base no Windows. Nesta lição, você examinará como pode gerar interfaces com o usuário com base em navegador. As ferramentas do Visual Basic .NET ajudam o desenvolvedor na criação de páginas da Web que fornecem interfaces com o usuário bem sofisticadas para qualquer tipo de navegador. Em particular, esta lição enfocará:

- Como o modelo de programação da Web difere do tradicional com base no Windows.
- O uso dos controles-padrão dos formulários da Web.
- O uso dos controles avançados dos formulários da Web.
- Os uso dos controles Validator.

O Modelo de Programação da Web

Às vezes, parece que todas as pessoas no mundo têm acesso à Internet, principalmente quando a velocidade da minha conexão fica bastante lenta porque todos estão navegando, conversando e enviando correspondências eletrônicas (porém, nada relacionado a trabalho). É claro que um dos

mais importantes, se não o mais popular, aspectos da Internet é a World Wide Web (WWW ou apenas Web). No entanto, é freqüente a falta de ferramentas de programação realmente boas para a criação de ‘programas’ na Web, em vez de simples páginas da Web. Isso acontece em parte porque o desenvolvimento de aplicativos para a Web é diferente da geração dos de microcomputadores, com os quais temos mais controle. Além disso, os aplicativos da Web precisam lidar com a rede com mais periodicidade.

Assim, o que é o ‘modelo de programação na Web’? É apenas um termo usado para descrever como você pode projetar, ou desenvolver, um programa que empregue páginas da Web em um navegador para permitir que o usuário insira informações. Essas páginas da Web são projetadas com a utilização de HTML (HyperText Markup Language). Este livro não abordará HTML, mas há muitos outros no mercado que o fazem.

O *navegador* é um aplicativo que sabe como ler HTML e exibi-la na tela. Em geral (mas nem sempre), grande parte do trabalho de um aplicativo é executado no servidor Web. O servidor Web é outro programa processado em um computador que sabe como retornar HTML quando solicitado. No Windows NT ou no 2000, esse programa é chamado de Internet Information Service (IIS). As informações são transferidas entre o navegador e o servidor por meio de um protocolo, ou linguagem, chamado HTTP (Hypertext Transfer Protocol).

**NOTA**

O nome efetivo do IIS foi alterado com as diferentes versões. No Windows NT 4.0 Server, ele é chamado Internet Information Server. No Windows 2000, foi denominado de Internet Information Services, enquanto no Windows NT 4.0 Professional, seu nome é Personal Web Server.

Nos ‘primórdios’ da World Wide Web, as páginas da Web eram estáticas. Isto é, nunca mudavam realmente. Tudo ficou mais interessante quando as pessoas começaram a criar páginas da Web dinâmicas ou que podiam ser alteradas. Essa foi a origem dos programas da Web. Com eles, em vez de apenas retornar sempre o mesmo resultado HTML, o servidor Web pode executar algumas tarefas e retornar o resultado HTML apropriado. Por exemplo, o usuário pode solicitar informações sobre as vendas de um período em particular. Essas informações serão passadas para o servidor. Por sua vez, o servidor poderá procurá-las em um banco de dados e, em seguida, convertê-las em HTML exibindo-as para o usuário. O processo completo é semelhante ao da Figura 10.1.

Como alternativa, o servidor (ou o projetista da página da Web) pode adicionar informações de programação à própria página, criando uma que seja, ela mesma, um programa. Em geral, chamamos esse recurso de Dynamic HTML (ou DHTML). Por meio da DHTML, é incluído na página algum código JavaScript (uma linguagem de programação, como o Visual Basic .NET, que é executada em páginas da Web) ou de outra linguagem. O código pode ser processado no navegador sem precisar retornar nenhuma informação para o servidor. A Figura 10.2 mostra esse modelo em ação.

FIGURA 10.1

Modelo de programação na Web.

**FIGURA 10.2**

O modelo de programação da Dynamic HTML.



Há várias técnicas que podem ser empregadas na criação de um programa da Web. Algumas das mais usadas no passado foram o ASP (Active Server Pages), a Perl (outra linguagem de programação) ou o JSP (Java Server Pages). A técnica utilizada pelos formulários da Web é um aperfeiçoamento do ASP, o ASP.NET.

ASP.NET

ASP.NET é o nome que a Microsoft deu para sua versão aprimorada do ASP. Embora o ASP fosse um método fácil de construir páginas dinâmicas na Web, apresentava alguns problemas que o ASP.NET resolveu:

- O ASP quase sempre exigia muita codificação para que algo fosse executado. O ASP.NET requer menos código (em geral, muito menos) para tarefas comuns de programação.
- O ASP também padecia da quantidade limitada de controles que a HTML apresenta. O ASP.NET adicionou o conceito dos 'controles no lado do servidor' que podem gerar o resultado HTML apropriado para o navegador que o solicitar. Embora esses controles no navegador sejam apenas HTML, podem representar muito da HTML e da codificação que você será poupado de ter de escrever.
- O ASP só aceitava programas em uma linguagem como o VBScript. O VBScript é interpretado no tempo de execução, não é compilado como o Visual Basic. O ASP.NET permite que você escreva páginas da Web em um código totalmente compilado do Visual Basic .NET.

O ASP.NET também foi projetado para resolver outros problemas do ASP que não são inerentes a essa discussão sobre a construção de interfaces com o usuário, como melhorias no redimensionamento e na utilização da memória.

Como a Criação de Programas com Base na Web Difere da de Programas com Base no Windows

Ao projetar um programa por meio dos formulários da Web, há várias diferenças que você deve ter em mente. Algumas delas são:

- Os aplicativos com base na Web tendem a ter mais código no servidor, em vez de no cliente. Isto é, a aparência de seu programa virá do navegador, mas os recursos avançados estarão no servidor.
- Os aplicativos com base na Web dependem dos recursos do navegador usado para visualizá-los. Infelizmente, os navegadores possuem recursos diferentes uns dos outros, e muitos desenvolvedores da Web têm-se deparado com essas diferenças no momento em que projetam seus programas.
- Quando você acessar uma página da Web, provavelmente ela será estática. Embora haja maneiras de poder atualizá-la sem retornar ao servidor (isto é, torná-la dinâmica), esses métodos fazem com que sua criação seja mais complexa. Portanto, produzir formulários animados (ou qualquer tipo de resposta para o usuário) é mais difícil com os aplicativos com base na Web.
- Muitas operações de aplicativos com base na Web requerem um ‘percurso de ida e volta na rede’. Isso acontece por causa da separação entre código e projeto. Para fazermos um botão ou outro controle executar algo, em geral é necessário enviar informações ao servidor. Assim, ele responderá de modo apropriado. Você deve se lembrar disso quando criar aplicativos da Web. Esse percurso de envio e retorno na comunicação pode levar algum tempo, portanto, ele só deve existir quando necessário.

Os aplicativos com base na Web são restritos, tanto pelas limitações do próprio navegador quanto pela quantidade de navegadores disponíveis no mercado. Os navegadores são limitados nos tipos de controles que podem ser usados, assim como por seus recursos insuficientes de desenho — isto é, em geral é impossível desenhar na página da Web. Além disso, se o usuário tiver uma versão mais antiga de um navegador instalada ou tiver desativado certos recursos, a página da Web poderá reagir de várias maneiras. Essa é uma das principais razões pela qual os aplicativos com base na Web tendem a apresentar grande parte da codificação no servidor. Isso também significa que esses aplicativos tradicionalmente precisam que muita codificação seja adicionada para que a aparência da página possa ser alterada conforme o navegador que a visualizar.

Felizmente, os controles dos formulários da Web ocultam a maioria desses detalhes. Eles foram criados para produzir uma saída dinâmica (isto é, a página pode ser alterada sem que seja necessário retornar informações ao servidor) se os controles detectarem que o navegador pode usá-la. Se eles não reconhecerem o navegador que estiver em uso ou se esse não der suporte a atualização dinâmica, só a HTML sem formatação será retornada ao navegador cliente. Isso assegura ao desenvolvedor que o navegador cliente receba a página da Web como foi projetada, respeitando as limitações do navegador.

Além das restrições em decorrência do navegador, os aplicativos com base na Web também tornam necessário que o desenvolvedor considere o fato de que o cliente e o servidor estão separados, possivelmente por grandes distâncias através de uma rede. Isso significa que as operações que poderiam levar alguns segundos se o cliente e o servidor estivessem próximos (ou até na mesma máquina) poderão demorar muito tempo. Portanto, operações como as animações podem ficar desfiguradas ou não serem exibidas de maneira alguma até que o download seja concluído. E ainda há a velocidade da conexão. Se você estiver acessando a página da Web por meio de um modem mais lento, essa diferença se tornará ainda mais relevante.

Com todas essas questões para lembrar, você deve estar pensando “Por que me preocupar em criar aplicativos da Web?”. Embora haja desvantagens em desenvolver aplicativos da Web, existem muitos benefícios:

- **Instalação** Para tornar seu aplicativo disponível, tudo que você precisa fazer é inserir algum endereço no URL. O aplicativo ficará imediatamente disponível para uso do cliente. Isso evitará que você tenha de ir até cada uma das máquinas clientes ou fazer com que todos os seus usuários instalem o aplicativo.
- **Novas versões e/ou correção de erros** Quando você quiser atualizar uma parte de seu aplicativo com uma versão mais nova, só terá de instalar as atualizações no servidor, e não em cada cliente.
- **Desempenho** Melhorar o desempenho de aplicativos da Web é muito mais fácil que fazê-lo com aplicativos *comuns*. Você pode aperfeiçoar seu desempenho adicionando mais servidores e distribuindo as solicitações por todos eles.
- **Conhecimento** Se você já conhece um pouco de HTML, poderá ser muito mais simples criar aplicativos da Web do que do Windows. Eles também são mais fáceis de assimilar, se não houver conhecimento de HTML ou de Windows.

Então, quando projetar um aplicativo, você deve criar um programa com base no Windows ou na Web? A resposta mais fácil (porém insatisfatória) é “Depende”. Muitos se encaixam em qualquer tipo, mas as pessoas estão começando a criar mais aplicativos com base na Web. A capacidade de fazer atualizações facilmente e as correções disponíveis são atraentes, portanto, pode ser preferível pelo menos considerar primeiro a criação de programas como aplicativos da Web.

No entanto, alguns programas não são candidatos a aplicativos da Web. Qualquer programa que precise de um vínculo contínuo entre o cliente e o servidor não é apropriado, nem os que requeiram muitas figuras (como os jogos). Para concluir, se só tiver um computador envolvido (isto é, não for um aplicativo cliente/servidor como um programa de banco de dados) ou se o aplicativo só for usado por você, pode ser mais sensato criar um aplicativo com base no Windows.

Usando os Controles-padrão dos Formulários da Web

Projetar uma página da Web usando o Visual Basic .NET é semelhante a criar um aplicativo comum do Visual Basic .NET. A única diferença é o que acontece no nível interno. Em vez de se adicionar um código para criar os controles e configurar suas propriedades, as tags HTML são inseridas na página ASP.NET, e o código é armazenado em um arquivo do Visual Basic .NET que funciona internamente.

Os controles que estarão disponíveis na criação de um aplicativo da Web são semelhantes àqueles que podem ser usados nos aplicativos Windows. Aí estão incluídos todos os controles comuns que você está acostumado a empregar (veja a Figura 10.3). A Tabela 10.1 apresenta uma descrição resumida desses controles.

FIGURA 10.3

Controles-padrão para os formulários da Web.

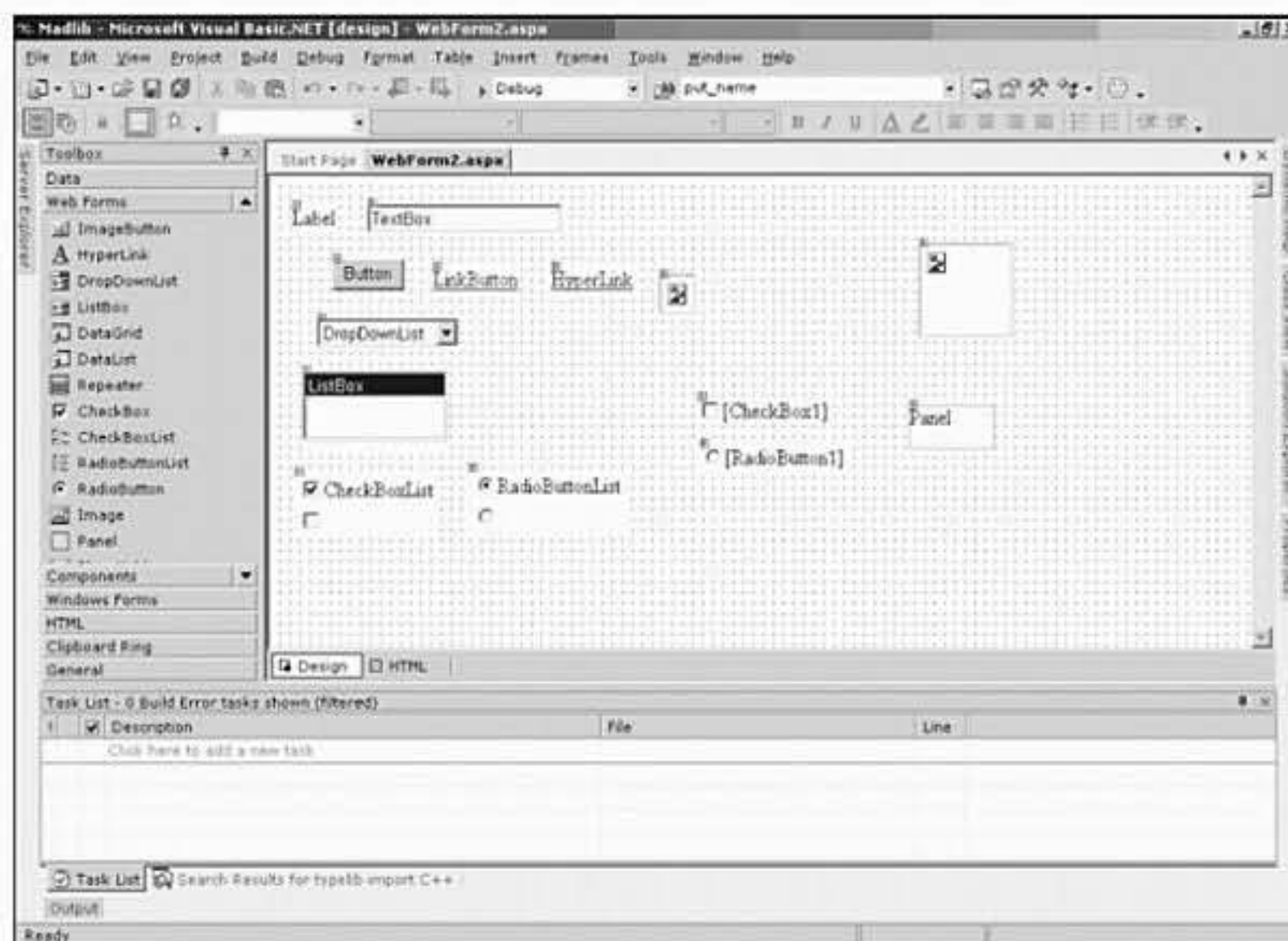


TABELA 10.1 Controles-padrão dos Formulários da Web

Controle	Descrição
Label	Use para inserir texto no formulário da Web. Como alternativa, você pode apenas dar um clique no formulário e digitar. O controle Label fornece uma administração melhor da formatação e permite a inserção do texto onde for desejado. Para concluir, este controle também possibilita a alteração dinâmica do conteúdo de seu aplicativo, o que não pode ser feito com o texto adicionado ao formulário.
TextBox	Use para fornecer ao usuário um campo onde inserir informações. Este em geral é o controle mais comum adicionado a um aplicativo da Web.

TABELA 10.1 Controles-padrão dos Formulários da Web (*continuação*)

<i>Controle</i>	<i>Descrição</i>
Button	Use para fornecer ao usuário algo para dar um clique a fim de executar alguma ação.
LinkButton	Semelhante no resultado ao controle comum Button, LinkButton é um recurso no qual o usuário de seu aplicativo da Web pode dar um clique. A diferença é que Button se parece com um botão, enquanto LinkButton é um hyperlink. (Ou seja, o usuário vê um bonito ponteiro azul, sublinhado em algum local.)
ImageButton	Semelhante no resultado ao controle comum Button, ImageButton é um recurso no qual seus usuários podem dar um clique a fim de executar alguma ação. A diferença é que ImageButton é uma figura.
Hyperlink	Semelhante a LinkButton, exceto por não possuir um evento Click. Isso significa que você só pode escrever códigos que lidem com a ação de clicar em LinkButton, ao passo que Hyperlink pode ser usado apenas para enviar o usuário para outro local.
DropDownList	Os controles DropDownList são comuns nos formulários da Web. Trata-se de uma lista que inicialmente só ocupa uma linha. Você pode dar um clique na seta suspensa para abrir e ver a lista completa. Depois de um item ser selecionado ela será fechada, e só uma linha será mostrada, contendo sua opção. Esses controles poderão ser usados nas situações em que seu usuário tiver de selecionar apenas um item em uma lista e quando se quiser economizar espaço na tela – por exemplo, para selecionar o código de um Estado ou país.
ListBox	Os controles ListBox permitem que o usuário selecione um ou mais itens em uma lista de opções. Eles diferem de DropDownList pelo fato de a lista ficar sempre visível. Outra diferença é que é possível selecionar vários itens em ListBox. Use este controle quando você precisar do recurso de seleção múltipla (porém, examine CheckBoxList), quando quiser que o usuário possa ver todas as opções ou quando houver bastante espaço na tela.
CheckBox	O controle CheckBox representa a resposta afirmativa ou negativa a uma pergunta. É marcado ou desmarcado e, portanto, usado quando se quer que o usuário selecione ou não uma opção. Ele difere de RadioButton, no fato de que CheckBox não depende de outros controles iguais a ele, enquanto RadioButton em geral é uma opção entre muitas.
CheckBoxList	O controle CheckBoxList é composto de vários controles CheckBox. Mesmo sendo todos independentes, o controle CheckBoxList é uma maneira prática de adicionar vários controles CheckBox a uma página. Este controle é especialmente útil quando se tem um conjunto de itens (que podem ter sido recuperados de um banco de dados) entre os quais o usuário deve selecionar. CheckBoxList também é um substituto adequado para ListBox quando se pretende que o usuário selecione vários itens. No entanto, é recomendável usar ListBox se houver mais de seis itens.

TABELA 10.1 Controles-padrão dos Formulários da Web (*continuação*)

<i>Controle</i>	<i>Descrição</i>
RadioButton	O controle RadioButton é semelhante a CheckBox no fato de seu valor só poder ser True ou False. A diferença entre os dois é que os controles RadioButton tendem a 'viajar em pacotes'. Embora cada CheckBox de um formulário possa ser configurado independentemente como True ou False, apenas um RadioButton de um conjunto pode ser True. Portanto, você pode considerar CheckBox como um controle que produz uma resposta afirmativa ou negativa, enquanto RadioButton (ou melhor, um grupo de controles RadioButton) se assemelha mais a uma pergunta de múltipla escolha para a qual só uma resposta é correta.
RadioButtonList	O controle RadioButtonList nada mais é que um grupo de controles RadioButton. Ele facilita a criação desse grupo, se você já tiver uma lista proveniente de algum outro local (como um banco de dados).
Image	O controle Image permite que você insira uma figura na página.
Panel	O controle Panel é semelhante ao Label no fato de ser apenas um espaço reservado para texto. A diferença é que Panel pode conter outros controles. Portanto, é um ótimo recurso a ser usado quando você precisa separar ou realçar informações ou controles. Controles semelhantes ou relacionados podem ser agrupados em Panel para que se destaquem.

Exatamente como deve ser feito com os controles Windows, para usar os da Web, dê um clique duplo sobre eles na caixa de ferramentas ou arraste-os para seu formulário. Nesse caso, porém, o formulário será uma página da Web.

Iremos criar um aplicativo simples com um formulário da Web para ver como esses controles nos ajudam a escrever programas na rede.

Inicie o ambiente de desenvolvimento se ele não estiver em execução e crie um aplicativo da Web. Chame-o de **Madlib**. Ele será usado como exemplo do desenvolvimento de um aplicativo simples da Web para que possamos ver muitos dos controles-padrão em ação. Selecione **File, New e Project** para abrir a caixa de diálogo **New Project**. Abra a pasta **Visual Basic Projects** e selecione o modelo de projeto **Web Application**. Altere o nome do projeto para **Madlib** e dê um clique em **OK** para poder construí-lo.

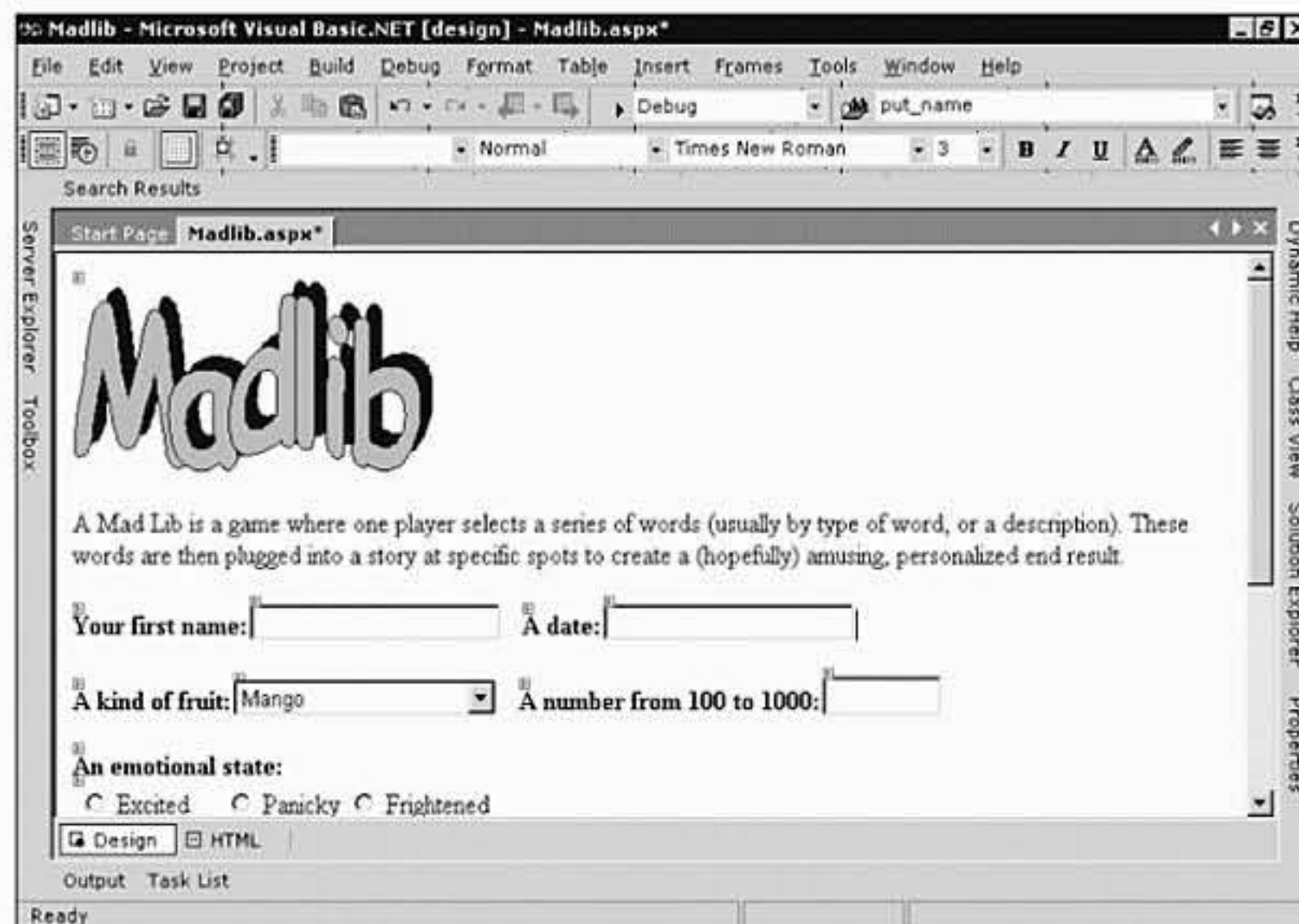
**NOTA**

Antes de criar um aplicativo da Web, você deve instalar ou poder acessar o Internet Information Services (ou o Internet Information Server).

Exatamente como nos aplicativos com base no Windows, sua primeira etapa será dispor os controles que usará em seu aplicativo (veja a Figura 10.4 para o resultado final). Comece adicionando uma figura a sua página.

O programa a seguir é conhecido como MadLib. Na verdade, é um jogo no qual um participante seleciona uma série de palavras (em geral pelo tipo de palavra ou uma descrição). Em seguida, essas palavras são inseridas em locais específicos de uma história para gerar um resultado divertido e personalizado (é o que se espera).

FIGURA 10.4
Formulário Madlib.



10

Arraste um controle Image para o formulário. No início, ele deve ter a aparência de um quadrado vazio (ou talvez de uma figura incompleta) porque você precisa configurar o caminho para a figura. Acesse a janela de propriedades e localize a propriedade ImageUrl. Quando der um clique na janela Property para acessar a propriedade ImageUrl, verá um botão com três pontos. Exatamente como nos aplicativos Windows, isso significa que uma caixa de diálogo o ajudará a configurar essa propriedade. Dê um clique no botão e procure uma figura adequada. (Criei uma que exibe Madlib no programa Paint que vem com o Windows 2000.) Dê um clique em OK, e a figura já deverá estar no formulário.

Fornecer uma explicação de seu programa em geral é uma boa prática. Adicione um controle Label e uma explicação simples na propriedade Text. O texto que inseri você encontra no primeiro parágrafo após a imagem.

A seguir, você adicionará à página os controles para os vários itens que inserirá. A Tabela 10.2 descreve as configurações das propriedades e controles usados.

TABELA 10.2 Controles Usados no Formulário Madlib da Web

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Label	(ID)	lblName
	Text	Your first name:
	Font Bold	True
TextBox	(ID)	TxtName
Label	(ID)	lblDate
	Text	A date:
	Font Bold	True
TextBox	(ID)	TxtDate
Label	(ID)	lblFruit
	Text	A kind of fruit:
	Font Bold	True
DropDownList	(ID)	cboFruit
	Items	O controle DropDownList possui uma caixa de diálogo que o ajudará a inserir itens nele. Dê um clique na propriedade Items e, em seguida, no botão resultante Build. Veja a Figura 10.5 para visualizar a caixa de diálogo que surgirá. Adicione várias frutas, dando um clique no botão Add e, então, configure a propriedade Text. Repita isso para cerca de dez itens. Adicionei as frutas: Mango (manga), Orange (laranja), Banana, Currant (groselha), Berry (cereja), Kumquat, Peach (pêssego), Kiwi, Apricot (damasco) e Plum (ameixa).
Label	(ID)	lblNumber
	Text	A number from 100 to 1000:
	Font Bold	True
TextBox	(ID)	TxtNumber
	Text	500
Label	(ID)	lblEmotion
	Text	An emotional state:
	Font Bold	True
RadioButtonList	(ID)	rlstEmotion

TABELA 10.2 Controles Usados no Formulário Madlib da Web (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Label	Items	A propriedade Items de RadioButtonList é semelhante a de DropDownList e possui o mesmo editor. Adicione alguns de seus estados emocionais favoritos aqui. Inserir o seguinte: Excited (excitado), Frustrated (frustrado), Intrigued (intrigado), Saddened (triste), Panicky (aterrorizado), Ecstatic (extático), Angry (zangado), Jealous (enciumado), Frightened (apavorado), Happy (feliz), Shocked (chocado) e Blue (melancólico).
	RepeatColumns	3
	(ID)	lblVerb
	Text	A verb:
TextBox	Font Bold	True
	(ID)	TxtVerb
Label	(ID)	lblOccupation
	Text	An occupation:
	Font Bold	True
TextBox	(ID)	txtOccupation
Button	(ID)	cmdWrite
	Text	Write Story
Button	ID	cmdClear
	Text	Clear
Label	(ID)	lblResult
	Text	Deixe este campo em branco (ou seja, exclua o valor da propriedade Text)
	BorderStyle	Groove
	Width	75%

Além disso, você pode querer inserir uma linha em certos pontos para organizar melhor os controles na página. Veja a Figura 10.5 para a visualização de um exemplo. Se estiver familiarizado com a HTML, também poderá adicionar os controles em uma tabela para obter possibilidades de formatação ainda mais adequadas.

**NOTA**

Há outra técnica que pode ser usada na inserção de controles em um formulário da Web. Se examinar as propriedades do formulário da Web (localize `DOCUMENT` na lista suspensa de objetos que se encontra na parte superior da janela `Property`), você verá uma com o nome `pageLayout`. Seu padrão é `LinearLayout`. A alternativa, `GridLayout`, pode ajudá-lo a criar formulários sofisticados na Web. Se `pageLayout` for configurada como `GridLayout`, você poderá inserir controles no formulário da Web exatamente como em um formulário `Windows`.

FIGURA 10.5
*Adicionado itens a
DropDownList.*



A maioria das propriedades usadas tem um sentido próprio; no entanto, algumas precisam de uma explicação adicional. Muitos controles que trabalham com listas podem ser ‘destinados’ a um conjunto de itens. Em geral, isso significa que eles poderão ser associados às informações recuperadas em bancos de dados, porém também podem fazer referência a outros conjuntos, como os arrays. Os controles que possuem essa capacidade podem ser facilmente identificados, já que possuem um conjunto `Items`. Esse conjunto aparece na janela `Property` e permite que você adicione itens sem vincular um controle a um array ou a outro conjunto. Essa é a maneira mais fácil de adicionar itens se eles não puderem ser alterados. Se apresentarem a possibilidade de alteração, deverão ser armazenados em um banco de dados e ser recuperados no tempo de execução para, em seguida, ser vinculados ao controle. Examinaremos esse procedimento no Dia 12, “Acessando Dados com a Plataforma .NET”.

O controle `RadioButtonList` possui uma propriedade relativamente rara: `RepeatColumns`. Você pode configurá-la para controlar a quantidade de colunas usadas na exibição da lista de itens. Essa pode ser uma ótima maneira de economizar algum espaço na tela, mesmo enquanto todos os itens são exibidos. Internamente, o controle `RadioButtonList` gera um código HTML para executar essa tarefa. Esse é um dos recursos que tornam esses controles mais fáceis de usar do que escrever seu próprio código HTML.

A próxima etapa no desenvolvimento de seu aplicativo da Web é adicionar códigos. Você só inserirá o código dos dois botões. Começaremos com o botão Clear. Esse botão apagará as informações de todos os controles TextBox e dos resultados de Label. Dê um clique duplo no botão Clear e adicione o código mostrado na Listagem 10.1.

CÓDIGO**LISTAGEM 10.1** Código para o Botão Clear

```
1 Private Sub cmdClear_Click(_  
2     ByVal sender As System.Object,_  
3     ByVal e As System.EventArgs)_  
4     Handles cmdClear.Click  
5         txtName.Text = " "  
6         txtDate.Text = " "  
7         txtVerb.Text = " "  
8         txtNumber.Text = " "  
9         txtOccupation.Text = " "  
10        lblResult.Text = " "  
11 End Sub
```

ANÁLISE

O código para esse botão é simples. Tudo que ele faz é configurar a propriedade Text de todos os controles TextBox e os resultados de Label com " ". Isso esvaziará esses controles.

O código da Listagem 10.2 também é simples. A idéia básica é que você crie uma string longa contendo toda a história, e que será exibida nos resultados de Label.

CÓDIGO**LISTAGEM 10.2** Código para o Botão Write Story

```
1 Private Sub cmdWrite_Click(_  
2     ByVal sender As System.Object,_  
3     ByVal e As System.EventArgs)_  
4     Handles cmdWrite.Click  
5     'é aqui que associamos as opções  
6     'que o usuário selecionou formando a história final  
7     Dim sTemp As String  
8     sTemp = "Diário de " & txtName.Text &  
9         " de " & DateTime.Today.ToString & "<br>"  
10    sTemp &= "Em " & txtDate.Text &  
11        " Comecei a programar no Visual Basic.NET. "  
12    sTemp &= "Estou " & rlstEmotion.SelectedItem.Text & "! "  
13    sTemp &= "Acho que vou sair e " & txtVerb.Text  
14    sTemp &= " meu novo " & txtNumber.Text  
15    sTemp &= " PicoHertz " & cboFruit.SelectedItem.Text  
16    sTemp &= " e me tornar um " & txtOccupation.Text & "."
```


CÓDIGO**LISTAGEM 10.2** Código para o Botão Write Story (*continuação*)

```

17      'a história final é armazenada no controle Label
18      lblResult.Text = sTemp
19  End Sub

```

ANÁLISE

O processo começa na linha 7, onde você declara a string. Em seguida, ela é construída da linha 8 à 16, e o resultado é inserido na propriedade Text do controle lblResult da linha 18. Um símbolo que provavelmente parecerá estranho é &= que se encontra nas linhas 10 à 16. Sendo um novo recurso do Visual Basic .NET, esse atalho é utilizado para executar inserções em uma string. O código das linhas 10 à 11, por exemplo,

```

10      sTemp &= "Em " & txtDate.Text &_
11      " Comecei a programar no Visual Basic.NET. "

```

é equivalente a:

```

10      sTemp = sTemp & "Em " & txtDate.Text &_
11      " Comecei a programar no Visual Basic.NET."

```

Você pode usar o operador &= para tornar seu código mais curto quando estiver adicionando mais informações ao final de uma string existente.

Depois que tiver adicionado o código, você já estará pronto para construir e testar seu programa. Selecione Build no menu Build e, em seguida, execute o programa. Isso deve iniciar um navegador e carregar sua página. Insira alguns itens e dê um clique no botão Write Story para ver sua história. A Figura 10.6 mostra o formulário da Web em ação.

FIGURA 10.6

O formulário Madlib em ação.

http://random/Madlib/Madlib.aspx - Microsoft Internet Explorer - Daily Build 6.00.2432.0002

File Edit View Favorites Tools Help

Back Forward Stop Home Personal Bar Search Favorites

Address http://random/Madlib/Madlib.aspx Links »

Madlib

A Mad Lib is a game where one player selects a series of words (usually by type of word, or a description). These words are then plugged into a story at specific spots to create a (hopefully) amusing, personalized end result.

Your first name: A date:

A kind of fruit: A number from 100 to 1000:

An emotional state:

☐ Excited ☐ Panicky ☐ Frightened

☐ Frustrated ☐ Ecstatic ☐ Happy

☐ Intrigued ☐ Angry ☐ Shocked

☐ Saddened ☐ Jealous ☐ Blue

A verb: An occupation:

Done Local intranet

Usando os Controles Avançados dos Formulários da Web

Apesar de ser fácil criar um formulário com os controles que estão disponíveis como parte da HTML, os formulários da Web se tornam ainda mais úteis (além de coloridos e fáceis de usar) quando são aplicados alguns dos controles mais avançados, como `Calendar`, `AdRotator` ou `Data`. Embora sejam construídos com o uso de controles mais simples, eles facilitam a criação das interfaces com o usuário.

O controle `Calendar` mostra, por estranho que pareça, um calendário mensal. Esse controle permite que o usuário visualize e selecione datas mais facilmente do que com o uso de uma `TextBox`. Além disso, empregando `Calendar`, você reduz a chance de o usuário inserir uma data em um formato inválido. O controle `Calendar` possui uma grande quantidade de propriedades, no entanto, a maioria delas diz respeito a sua exibição. Quase tudo que está visível no controle pode ser ajustado – as cores, a grafia do dia da semana ou do mês e assim por diante. A Tabela 10.3 resume algumas das propriedades mais úteis do controle `Calendar`.

TABELA 10.3 Propriedades Importantes do Controle `Calendar`

<i>Item</i>	<i>Propriedade</i>	<i>Descrição</i>
<code>SelectedDate</code>	Propriedade	A data que será retornada por esse controle.
<code>VisibleDate</code>	Propriedade	A data mostrada no controle. Embora em geral ela seja a mesma de <code>SelectedDate</code> , pode ser diferente, principalmente se você estiver tentando configurá-la por meio de código.

Atualizemos o projeto `MadLib` de modo que use um controle `Calendar` em vez de `TextBox` para a data inserida. Você pode editar o formulário antigo ou copiá-lo se quiser visualizar os dois.

Exclua a caixa de texto `Date` e adicione um controle `Calendar`. Neste ponto, você pode escolher entre testar as propriedades que afetam a aparência de `Calendar` ou facilitar as coisas e selecionar o link `AutoFormat`, que se encontra na parte inferior da janela `Properties` quando selecionar o controle `Calendar`. Optei por dar um clique no link e selecionei `Colorful 2`. (Por que complicar e criar algo estranho quando um profissional já trabalhou em uma opção de aparência adequada – seria essa uma de minhas estratégias de programação?) Configure a propriedade `Calendar's Name` como `calDate`.

Você também terá de alterar um pouco o código das Listagens 10.1 e 10.2 por causa da mudança no nome do controle. As Listagens 10.3 e 10.4 mostram o novo código alterado.

CÓDIGO**LISTAGEM 10.3** Código Alterado do Botão Clear

```

1 Private Sub cmdClear_Click(_
2     ByVal sender As System.Object,_
3     ByVal e As System.EventArgs)_
4     Handles cmdClear.Click
5         txtName.Text = " "
6         calDate.SelectedDate = DateTime.Today
7         txtVerb.Text = " "
8         txtNumber.Text = " "
9         txtOccupation.Text = " "
10        lblResult.Text = " "
11 End Sub

```

ANÁLISE

Apenas uma linha de código foi alterada. Já que você não tem mais a caixa de texto txtDate, não pode configurá-la com " ". Em vez disso, você pode reconfigurar Calendar para selecionar o dia atual (DateTime.Today), como fez na linha 6.

CÓDIGO**LISTAGEM 10.4** Código Alterado do Botão Write Story

```

1 Private Sub cmdWrite_Click(_
2     ByVal sender As System.Object,_
3     ByVal e As System.EventArgs)_
4     Handles cmdWrite.Click
5     'é aqui que associamos as opções
6     'que o usuário selecionou formando a história final
7     Dim sTemp As String
8     sTemp = "Diário de " & txtName.Text &
9         " de " & DateTime.Today.ToString & "<br>"
10    sTemp &= "Em " & calDate.SelectedDate &
11        " Comecei a programar no Visual Basic.NET. "
12    sTemp &= "Estou " & rstEmotion.SelectedItem.Text & "! "
13    sTemp &= "Acho que vou sair e " & txtVerb.Text
14    sTemp &= " meu novo " & txtNumber.Text
15    sTemp &= " PicoHertz " & cboFruit.SelectedItem.Text
16    sTemp &= " e me tornar um " & txtOccupation.Text & "."
17    'a história final é armazenada no controle Label
18    lblResult.Text = sTemp
19 End Sub

```

ANÁLISE

Novamente, a única alteração na Listagem 10.4 foi na linha 10. Em vez de recuperar a data em TextBox, o código recupera a data selecionada em Calendar com calDate.SelectedDate.

Não é verdade que as propriedades possuem um sentido intrínseco? É por isso que os criadores não chamaram o sistema de Visual Complexo .NET.

Usando os Controles Validator

Quando você criar formulários de entrada de dados para a Web, em geral será necessário assegurar que eles sejam preenchidos de modo correto. Isso pode significar campos específicos preenchidos, ou apenas alguns deles, mas com valores que estejam dentro de um intervalo. No passado, poderíamos fazer isso escrevendo um código no servidor ou no cliente. Se o código estivesse no servidor, poderia fazer com que as informações fossem passadas entre o cliente e o servidor desnecessariamente. Se, em vez disso, colocássemos o código no cliente, enfrentaríamos problemas com a incompatibilidade de alguns navegadores.

Vários controles Validator que tornam a validação de formulários muito mais fácil foram incluídos no Visual Basic .NET. Os controles processarão a validação no servidor ou no cliente se determinarem que o navegador tem essa capacidade. Cinco controles de validação estão disponíveis no Visual Basic .NET.

- **RequiredFieldValidator** Assegura que um campo tenha sido preenchido. Você pode usá-lo sempre que quiser se certificar se o usuário concluiu o preenchimento de um formulário antes de enviá-lo.
- **CompareValidator** Assegura que dois campos estejam coincidindo ou que um campo seja comparado a algum valor. No primeiro caso, a comparação entre campos será útil quando você quiser que o usuário digite sua senha duas vezes. Comparar um campo com algum valor será adequado se desejar que o usuário insira um número positivo ou se a entrada do usuário tiver de incluir um tipo especial de informação (por exemplo, uma data).
- **RangeValidator** Assegura que o valor digitado em um campo esteja dentro de um intervalo. O intervalo pode ficar entre dois valores (como uma data inicial e uma final) ou entre dois controles. Por exemplo, você pode ter um controle no qual o usuário deve inserir um valor mínimo, e um segundo para o valor máximo. Em seguida, o controle Validator se certificaria se o valor digitado em um terceiro controle estaria entre os outros dois. Isso pode ser útil como parte de um relatório, no qual pode-se querer que o usuário selecione uma data que esteja dentro do intervalo de informações armazenadas em um banco de dados.
- **RegularExpressionValidator** Assegura que o valor digitado tenha o formato desejado. O valor é comparado com uma expressão regular. Se coincidir, será considerado válido. Pode ser útil para valores que precisem apresentar uma certa estrutura, como números telefônicos, ISBNs ou números de peças.
- **CustomValidator** Permite que você adicione seu próprio código para validar o campo. Sendo o mais flexível dos controles de validação, esse código é executado no servidor ou no cliente. Pode ser útil quando um dos outros controles de validação não atende sua necessidade ou as informações válidas precisem ser determinadas por meio de algum pro-

cesso – por exemplo, se o valor tiver de ser uma entre várias entradas que se encontrem em um banco de dados.

Além desses cinco, também há o controle `ValidationSummary`, que exibe todas as mensagens de erro de todos os controles `Validator` na mesma página. É útil por fornecer um único local para todas essas informações.

Os cinco controles de validação possuem várias propriedades importantes em comum. Elas estão relacionadas com o controle que monitoram e com a maneira de exibir o erro. As mais importantes entre essas propriedades estão descritas na Tabela 10.4.

TABELA 10.4 Propriedades Comuns aos Controles de Validação

<i>Propriedade</i>	<i>Descrição</i>
<code>ControlToValidate</code>	Esta é a propriedade mais importante de todos os controles de validação. Deve ser configurada para apontar para outro controle (pelo nome) do mesmo formulário. Esse é o controle que será monitorado pelo controle de validação. Use a lista suspensa da janela <code>Property</code> para selecionar o controle monitorado.
<code>ErrorMessage</code>	É a mensagem a ser exibida se houver um erro com o controle de validação, por exemplo, se o campo for deixado em branco. Ela deve ter informações suficientes para permitir que o usuário determine o que está errado e como corrigir o erro.
<code>Display</code>	Esta é uma propriedade um pouco estranha que define como o controle <code>Validator</code> aparecerá na página da Web. Por padrão, ela é configurada como <code>Static</code> ; no entanto, há duas outras opções, <code>Dynamic</code> ou <code>None</code> . Se o valor for definido como <code>Static</code> , o espaço ocupado por <code>ErrorMessage</code> estará sempre preenchido, mesmo se essa mensagem não for exibida. Isso será útil se você quiser garantir a organização de sua página da Web. O valor <code>Dynamic</code> significa que o controle não ocupará espaço até que a propriedade <code>ErrorMessage</code> seja exibida. É adequado quando não se deseja espaços em branco no formulário. Para concluir, se esta propriedade for configurada como <code>None</code> , <code>ErrorMessage</code> nunca será exibida. Essa configuração só é útil com o controle <code>ValidationSummary</code> (que mostrará o erro).

Você pode usar alguns desses controles para encerrar o aplicativo `MadLib` do exemplo. Pode empregar o controle `RequiredFieldValidator` para assegurar que o usuário tenha inserido as informações nos campos certos, e `RangeValidator` para se certificar de que um número apropriado foi digitado no campo `txtNumber`. Para concluir, é possível abranger todos os erros em um resumo com o controle `ValidationSummary`.

Mais uma vez, copie ou abra o projeto ou o formulário anterior para editá-lo. Você adicionará os controles `Validator` a ele.

Arraste um controle `RequiredFieldValidator` para perto de cada um dos controles `TextBox` remanescentes (`txtName`, `txtNumber`, `txtVerb` e `txtOccupation`). Adicione um controle `RangeValidator` próximo ao controle `RequiredFieldValidator` que você posicionou perto do campo

txtNumber. Por fim, adicione um controle ValidationSummary em uma linha própria entre os botões e o controle lblResult. Configure as propriedades de cada um desses controles como mostrado na Tabela 10.5.

TABELA 10.5 Propriedades para os Controles Validator do Formulário Madlib

Controle	Propriedade	Valor
RequiredFieldValidator	ID	reqName
	ControlToValidate	txtName
	ErrorMessage	Please enter a name
RequiredFieldValidator	(ID)	reqNumber
	ControlToValidate	txtNumber
	ErrorMessage	Please enter a number
	Display	Dynamic
RangeValidator	(ID)	rngNumber
	ControlToValidate	txtNumber
	ErrorMessage	Please enter a number from 100 and 1000
	Display	Dynamic
	Type	Integer (inteiro)
	MaximumValue	1000
	MinimumValue	100
RequiredFieldValidator	(ID)	reqVerb
	ControlToValidate	txtVerb
	ErrorMessage	Please enter a verb
	Display	Dynamic
RequiredFieldValidator	(ID)	reqOccupation
	ControlToValidate	txtOccupation
	ErrorMessage	Please enter an occupation
	Display	Dynamic
ValidationSummary	(ID)	valSummary

Precisamos fazer uma pausa para descrever as três propriedades de RangeValidator que você não atribuiu antes. Embora seja *evidente* o sentido de MaximumValue e MinimumValue com relação a algo chamado RangeValidator (validação de intervalo), a propriedade Type não é tão óbvia. Já que RangeValidator pode ser usado para testar vários tipos diferentes de valores (como inteiros,

valores financeiros ou datas), a propriedade `Type` identifica o tipo de informação que será comparada. Ela pode ter um dos valores a seguir:

- **String** O padrão; faz com que o controle verifique se o valor está alfabeticamente entre as duas extremidades.
- **Integer (inteiro)** Compara o valor com as duas extremidades para assegurar de que faz parte do intervalo. Só os valores inteiros são usados.
- **Double (duplo)** O mesmo que o inteiro, porém incluindo a parte decimal do valor e das extremidades do intervalo.
- **Currency (moeda)** O mesmo que o inteiro, porém incluindo as quatro primeiras casas decimais do valor.
- **Date (data)** Compara os valores como se fossem datas, portanto, um valor igual a 27 de agosto de 1964: seria aceito em um intervalo entre 23 de novembro de 1963 e 1º de abril de 1986.

Agora verificaremos por que os controles `Validator` são tão úteis. Para fazê-los funcionar, você não precisa adicionar nenhum outro código. Construa e visualize a nova página da Web (veja a Figura 10.7). Tente deixar alguns campos em branco e exclua o algarismo 500 que é o valor-padrão para o campo do número. Você verá mensagens de erro vermelhas aparecerem no formulário. Se isso não acontecer, tente dar um clique no botão `Write Story`. Provavelmente surgirá algo semelhante ao formulário da Figura 10.8. Tente inserir um valor que esteja fora do intervalo aceito para o campo do número. Por fim, digite valores corretos em todos os campos e selecione o botão `Write Story`. Todas as mensagens de erro devem desaparecer e nossa história será exibida.

FIGURA 10.7
O formulário `Madlib` mostrando os controles de validação.

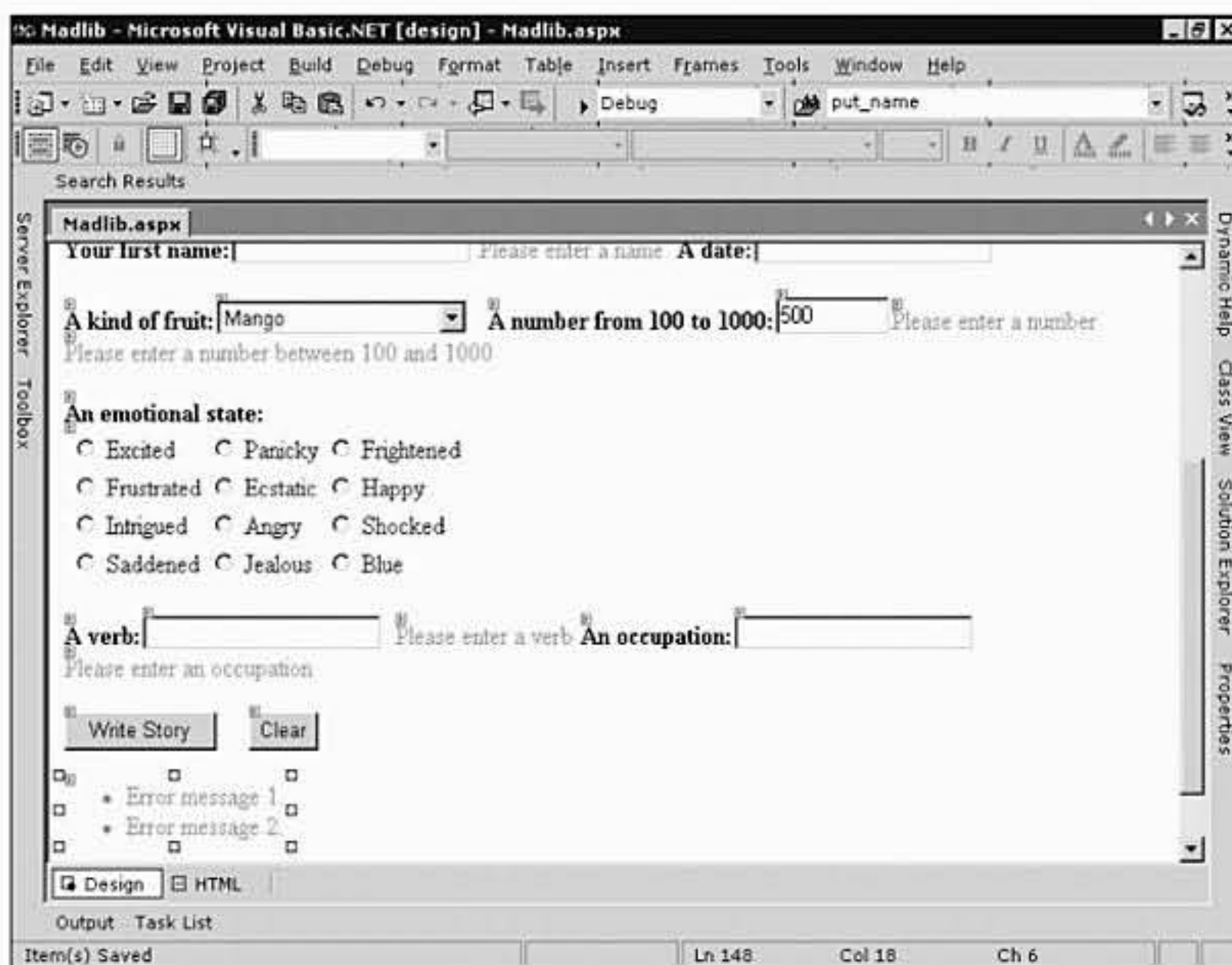


FIGURA 10.8
O formulário Madlib mostrando os erros encontrados na validação.

http://random/Madlib/Madlib.aspx - Microsoft Internet Explorer - Daily Build 6.00.2432.0002

File Edit View Favorites Tools Help

Back Forward Stop Home Personal Bar Search Favorites

Address http://random/Madlib/Madlib.aspx Links

A Mad Lib is a game where one player selects a series of words (usually by type of word, or a description). These words are then plugged into a story at specific spots to create a (hopefully) amusing, personalized end result.

Your first name: A date:

A kind of fruit: A number from 100 to 1000: Please enter a number between 100 and 1000

An emotional state:

☒ Excited ☐ Paracky ☐ Frightened

☐ Frustrated ☐ Ecstatic ☐ Happy

☐ Intrigued ☐ Angry ☐ Shocked

☐ Saddened ☐ Jealous ☐ Blue

A verb: Please enter a verb An occupation: Please enter an occupation

- Please enter a number between 100 and 1000
- Please enter a verb
- Please enter an occupation

Done Local intranet

Resumo

Os aplicativos da Web estão cada vez mais importantes, e os formulários da Web tornam sua criação bastante fácil. Eles permitem que você use as mesmas técnicas que empregaria para desenvolver aplicativos para microcomputadores na construção de um aplicativo da Web que funcione em qualquer navegador. Colocando o código novamente em um servidor (que você possa controlar), é possível extrair o melhor que há nas duas alternativas para chegar ao objetivo pelo qual os desenvolvedores se empenharam – uma interface com o usuário sofisticada e recursos compatíveis com vários navegadores.

Embora apenas simular uma experiência como a que se obtém no Windows já seja o suficiente para a maioria das pessoas, os formulários da Web vão além, ajudando-o a criar com maior facilidade rotinas de validação e controles complexos.

Na próxima lição, você começará a examinar a programação orientada a objetos. Aprenderá qual a relação que a POO tem com o Visual Basic .NET e com a programação em geral. Perceberá que até aqui já esteve investigando os objetos, na forma do .NET Framework, assim como dos formulários e controles que tem usado.

P&R

P Preciso ter um servidor Web disponível para usar os formulários da Web?

R Sim, os aplicativos da Web precisam de um servidor Web que reconheça o ASP.NET, como o Internet Information Server (IIS). O IIS vem com um servidor gratuito da Web que faz parte do Windows NT Server ou do Windows 2000 Server.

P Posso usar o Windows 98 ou o Windows Me para criar e implantar os aplicativos da Web?

R Esses sistemas operacionais já possuem, ou disponibilizam, o Personal Web Server (PWS). Não é possível usar o PWS para criar aplicativos da Web. Os sistemas Windows 9x podem ser empregados na geração de aplicativos da Web, mas não são plataformas boas para sua implantação. Você deve criar aplicativos da Web utilizando o IIS no Windows NT 4 Server ou no Windows 2000.

P Como posso aprender mais sobre escrever códigos HTML?

R Embora os formulários da Web tornem o conhecimento da HTML quase opcional, ter uma boa formação prática nessa linguagem ajudará. Examine um dos bons livros que se encontram no mercado para aprender mais.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Se você pode adicionar texto a uma página da Web apenas por meio da digitação, por que precisaria dos controles Label?
2. Que controles o ajudarão a navegar entre as páginas de um aplicativo da Web?
3. Como posso exibir figuras em um formulário da Web?

Exercício

Crie um aplicativo para execução de registros. Esse novo aplicativo da Web deve permitir que o usuário digite:

- Seu nome. Esse campo é de preenchimento obrigatório.
- Um nome de usuário para usar em seu site. Esse campo é de preenchimento obrigatório e não pode ter mais do que 10 caracteres.
- Uma senha nova (lembre-se de que a senha deve ser mantida em segredo). O usuário deve inserir sua senha duas vezes para garantir que tenha sido digitada de modo correto. Os dois campos são de preenchimento obrigatório e devem ter a mesma informação.

Depois que as informações tiverem sido inseridas, o usuário deve dar um clique em um botão. Elas serão então mostradas para ele.

Semana 2

DIA 11

Introdução aos Bancos de Dados

Computadores são boas ferramentas para trabalhar com dados, principalmente em grandes quantidades, e, no decorrer da história desses dispositivos, a maioria dos programas foi escrita exatamente para fazer isso. Mesmo hoje, em que se utilizam os computadores para uma enorme variedade de tarefas, o processamento de dados ainda faz parte de quase todos os aplicativos empresariais e de muitos outros com finalidades diferentes. Nesta lição você aprenderá sobre os bancos de dados, incluindo:

- Termos importantes relacionados aos bancos de dados.
- Aspectos básicos da SQL.
- Dicas sobre projetos de bancos de dados.
- Uma breve introdução aos dados na plataforma .NET.

Um Banco de Dados É a Solução para Todos os Problemas Cotidianos

Nos últimos anos, um dos autores deste livro foi ficando cada vez mais atrapalhado com a sua coleção de CDs. Só há pouco tempo, ele percebeu que quase sempre ouvia apenas um ou dois CDs específicos porque não sabia exatamente o que possuía ou, quando sabia, não conhecia sua localização. A preocupação com a coleção de CDs se tornou real quando ele decidiu limpar sua grande pilha de discos e tentou organizá-los em algo parecido com uma classificação.

Porém, organizar os CDs em um único fichário não é uma tarefa tão simples quanto pode parecer, já que há muitas maneiras diferentes de ordená-los. Talvez por artista possa ser melhor (certificando-se de que os Barenaked Ladies venham antes do Boney M.), ou por categoria, separando os discos em grupos de pop, rock, músicas natalinas e por aí afora. Há tantas decisões a serem tomadas. Brian Adams deve ser colocado na letra A (levando-se em consideração o sobrenome) ou na B (considerando-se o primeiro nome)? E os CDs novos? Depois de colocar esse mesmo CD do Brian Adams em um dos poucos espaços iniciais de seu fichário, o que se deve fazer se ele lançar mais um álbum? Será necessário mover todos os outros CDs para a frente a fim de criar espaço, ou será interessante apenas colocá-lo no final e prejudicar seu ótimo sistema de posicionamento alfabético? No final, pode ser mais simples decidir não comprar mais discos, o que parece ter sido a escolha feita pelos pais do autor nos idos dos anos 70.

A necessidade de tomar uma decisão se tornou óbvia quando, durante a limpeza de CDs, nosso conhecido autor se deparou com dois (2!) CDs “Queen: Classic Hits, Volume 1”. Pois bem – e isso não é nada contra o Queen –, ninguém precisa de dois CDs iguais de melhores sucessos, não importa quanto ele goste de ouvir “Bohemian Rhapsody”. Algo tinha de ser feito rapidamente.

A Decisão É Tomada

Todas as opções acabaram sendo muito trabalhosas, e o autor chegou a uma conclusão surpreendente: as informações sobre o CD deveriam ser inseridas em um programa de computador! Para ser sincero, não é assim tão surpreendente – ele chegou a essa conclusão em muitas outras situações (talvez muitas, se você perguntar a sua esposa ou a um de seus poucos amigos que continuaram a sê-lo). No entanto, a idéia de armazenar dados em um computador não é tão nova; na verdade ela sempre foi uma das finalidades mais comuns dos sistemas de computador.

Você pode organizar os dados de muitas maneiras, mas a vantagem de usar um computador para executar a organização é que, depois que estiverem no sistema, poderão ser visualizados de várias maneiras. Isso elimina muitos dos problemas enfrentados pelo pobre autor no cenário descrito anteriormente. Se um novo CD tiver de ser incluído, poderá ser adicionado ao final do arquivo de dados, mas será exibido em sua localização apropriada com base na visualização selecionada. Se você quiser visualizar os CDs por artista, apenas ordene os dados usando um campo chamado Artista. Se preferir vê-los pelo título do CD, a reordenação pode ser facilmente executada.

Nesse caso, vamos supor que todos os dados tenham sido inseridos em um arquivo de texto no computador, com cada linha desse arquivo representando um CD. Os campos e entradas possíveis teriam a aparência a seguir:

```
ArtistName, ArtistFirstName, ArtistLastName, CDTitle, CDReleaseYear,  
➡ MusicCategories  
Sting, Sting, -, Brand New Day, 2000, Pop  
Queen, Queen, -, Classic Hits Volume 1, 1987, Rock/Pop/Best Of  
Sting, Sting, -, Fields Of Gold, 1994, Pop/Best Of
```




Em muitos bancos de dados, não são permitidos espaços nos nomes das tabelas ou dos campos, portanto o nome dos campos teve de ser alterado para evitar problemas de compatibilidade.

A primeira linha desse arquivo, chamada *linha do cabeçalho*, indica que partes da informação estão armazenadas em cada linha. Uma nova linha marca o início do registro de cada álbum. As vírgulas são usadas para separar os campos, tornando esse arquivo um arquivo *separado por vírgulas* (ou *delimitado por vírgulas*). Tão simples quanto parece, esse arquivo é um banco de dados e poderia ser utilizado por um programa de computador que registrasse os álbuns. Já que se trata de um banco de dados, você pode usá-lo para aprender alguns dos termos que empregará daqui em diante para se referir às diferentes partes de um banco de dados:

NOVO TERMO

Cada CD listado nesse arquivo é um *registro* individual, às vezes chamado de *linha* porque os registros em geral são armazenados ou exibidos como linhas separadas com as informações.

NOVO TERMO

Dentro de cada registro, vários trechos da informação sobre cada CD são registrados, e cada item desses é chamado de *campo* (nesse caso, ArtistName, CDTitle, CDReleaseYear e outros). Como os registros, os campos em geral são conhecidos por outro nome, *colunas*, porque são com frequência armazenados e exibidos como colunas individuais com informações.

NOVO TERMO

O grupo completo de registros (todos com o mesmo conjunto de campos) é chamado de *tabela*, e um banco de dados pode conter muitas delas, embora o exemplo desta lição, até o momento, só apresente uma.

11

A Passagem para um Banco de Dados Real

Certo, então agora você conhece todos os termos que precisará para descrever seu pequeno banco de dados, que contém uma única tabela de registros de CDs que chamaremos de Disc. (Espero que a utilização que se faz do termo ‘registro’ para descrever tanto os lançamentos musicais quanto uma parte do banco de dados não gere confusão.) Com seus termos e o banco de dados definidos, como iremos trabalhar com esses dados em seus programas de computador?

Como já mencionei, o banco de dados do exemplo é um simples arquivo de texto, e esse tipo de arquivo é relativamente fácil de ler. (Veja o Dia 4, “Controlando o Fluxo dos Programas”, para ver um exemplo da leitura de um arquivo ou o Dia 17, “Usando o .NET Framework”, para obter mais detalhes sobre as excelentes classes de leitura de arquivos do Framework.) No entanto, como o exemplo se encontra no momento, você mesmo teria de analisar cada linha dos dados e manipular todas as colunas. Mesmo se pudesse ler o arquivo, seria preferível poder manipulá-lo usando alguma das maneiras-padrão, inclusive encontrar registros específicos, adicionar novos itens, excluir outros e alterar todo ou parte de um registro. Para implementar esses recursos, seria necessário escrever uma codificação extensa, mas não será preciso manipular nada disso se con-

vertermos seus dados para o formato de um banco de dados como o Access ou o SQL Server. Esses sistemas gerenciadores de bancos de dados podem manipular o armazenamento de suas informações e também responder solicitações de recuperação e alteração desses dados, gerenciando todos os detalhes de layout de arquivo e armazenamento. Os sistemas de bancos de dados possuem muitos recursos, mas o serviço comum que eles fornecem é manipular os detalhes de armazenamento de dados e liberá-lo de qualquer interação física com as informações.

Uma Introdução à SQL

Depois que os dados forem armazenados em um banco de dados, você ainda terá de manipulá-los a partir do programa, mas não precisará mais se preocupar com todos os detalhes. A SQL (Structured Query Language) foi desenvolvida para fornecer uma linguagem comum de acesso aos bancos de dados. Essa linguagem contém comandos para manipulação de registros nas tabelas do banco de dados e tem suporte até certo ponto de quase todos os softwares gerenciadores de bancos de dados disponíveis. Muitos comandos diferentes da SQL podem ser usados, mas os que abordarei aqui manipulam:

- A recuperação dos registros dos dados.
- A inclusão de novos registros.
- A alteração dos registros existentes.
- A exclusão de registros.

Recuperando Registros com a Instrução SELECT

Em SQL, a instrução SELECT é usada para recuperar dados de um banco de dados. Esta instrução, em sua forma básica, tem a aparência a seguir:

SINTAXE

```
SELECT <nome dos campos> from <Tabela>
```

Com o banco de dados de CDs como exemplo, se você quisesse recuperar todos os títulos de CDs junto com os nomes de seus artistas, poderia executar esta instrução SQL:

```
SELECT ArtistName, CDTitle FROM Disc
```

Ela retornaria um conjunto de dados com dois campos, e uma linha para cada registro da tabela Disc. Como alternativa, se você quisesse obter todas as colunas de uma tabela, poderia usar * em vez de uma lista de campos, produzindo uma instrução como a da linha a seguir:

```
SELECT * FROM Disc
```

Embora você provavelmente irá ver instruções SQL como essa em muitos exemplos e até em aplicativos, não é uma prática recomendável recuperar todos os campos de uma vez. O que se deseja sempre é recuperar a menor quantidade possível de infor-

mações, e em geral isso não significa todos os campos. Seja cuidadoso para não prejudicar a longo prazo o desempenho de seu código apenas para obter uma economia no tempo que leva a digitação em seu programa.

Faça	Não Faça
Especifique cada campo que quiser recuperar em uma instrução SQL.	Não use * para indicar todos os campos, a menos que esteja certo de que precisará de cada campo que existe na tabela e de todos os outros que possam ser adicionados posteriormente.
Recupere a menor quantidade de dados ou de campos e registros possível para fornecer um desempenho melhor.	

Ordenando os Resultados

A ordem dos registros retornados não foi especificada nas duas consultas mostradas até agora, mas você pode adicionar uma cláusula à instrução SQL para configurar a ordem da classificação. Essa cláusula, `ORDER BY`, usa uma lista de campos, permitindo que possa ser especificado mais de um na ordenação. A instrução SQL a seguir recupera os campos `ArtistName` e `CDTitle` de todos os registros da tabela `Disc`, ordenando-os primeiro pelo nome do artista (`ArtistName`) e, em seguida, pelo título do CD (`CDTitle`) dentro de cada conjunto de registros de artistas:

```
SELECT ArtistName, CDTitle FROM Disc ORDER BY ArtistName, CDTitle
```

Por padrão, um campo especificado na cláusula `ORDER BY` será classificado em ordem crescente (em relação ao valor ou à ordem alfabética, até atingir o último registro retornado), de A a Z se o campo armazenar texto. Se você quiser que a ordem seja inversa, decrescendo de Z a A, então, especifique uma palavra-chave `DESC` próxima ao trecho da cláusula `ORDER BY` que deseja que seja classificada na ordem inversa. O exemplo a seguir ordenará por artista em ordem alfabética inversa e, em seguida, por título de CD em ordem crescente.

```
SELECT ArtistName, CDTitle FROM Disc ORDER BY ArtistName Desc, CDTitle
```

A ordem-padrão, crescente, também pode ser especificada com o uso da palavra-chave `ASC` exatamente como `Desc` é empregada. A instrução SQL a seguir possui o mesmo efeito que a anterior, mas é mais explícita sobre o que está acontecendo.

```
SELECT ArtistName, CDTitle  
FROM Disc  
ORDER BY ArtistName Desc, CDTitle ASC
```

Faça	Não Faça
Especifique <code>ASC</code> mesmo se estiver usando <code>DESC</code> ; ajudará a evitar confusão.	Não ordene mais campos do que os necessários porque isso afetará o desempenho.

Especificando um Critério

As instruções SELECT mostradas até agora recuperavam todos os registros da tabela Disc, mas e se você quiser recuperar só um determinado registro ou um conjunto de registros com base em um critério específico? Outra cláusula pode ser adicionada a uma instrução SELECT para se encarregar disso, a cláusula WHERE. Nessa cláusula, podem ser especificados quantos critérios se desejar para restringir que linhas serão retornadas. Usando esse recurso, a instrução SELECT a seguir só recuperará os CDs do Sting:

```
SELECT CDTitle, CDReleaseYear FROM Disc
WHERE ArtistName = 'Sting'
```

Observe que o campo usado para restringir os resultados não foi um dos retornados; você pode utilizar o campo que quiser. É possível combinar vários critérios empregando os operadores AND/OR e usar parênteses para determinar como esses critérios serão aplicados. Para recuperar os discos lançados depois de 1984 cujo nome do artista é Sting e os lançados antes de 1984 pelo grupo The Police, a instrução SELECT a seguir poderia ser empregada:

```
SELECT CDTitle, CDReleaseYear FROM Disc
WHERE (ArtistName = 'Sting' AND CDReleaseYear >= 1984)
      OR (ArtistName = 'The Police' AND CDReleaseYear < 1984)
```

Essa consulta levanta uma questão relativa aos bancos de dados que sempre foi incômoda: o disco deve ser inserido como Police ou Police, The, representando o valor do campo ArtistName? Detalharei esse e outros problemas comuns relacionados aos bancos de dados, além das melhores maneiras que conheço para resolvê-los, ainda nesta lição em “Problemas Comuns dos Bancos de Dados e Suas Soluções”.

Adicionando Novos Registros

A ordem na qual esta lição aborda essas instruções SQL se apresenta um pouco invertida. Você aprendeu a recuperar dados no banco de dados antes de saber como inseri-los! Embora alguns deles sejam apenas de leitura, na maioria dos casos os registros novos são adicionados às tabelas regularmente. O comando SQL empregado para inserir esses registros novos é chamado INSERT e apresenta este formato básico:

```
INSERT INTO <Tabela> (campo 1,campo 2,...) VALUES (valor 1,valor 2,...)
```

Voltando ao exemplo da biblioteca de registro de CDs, você poderia usar uma instrução INSERT para adicionar um disco novo:

```
INSERT INTO Disc (ArtistName, ArtistFirstName, ArtistLastName, CDTitle,
➡CDReleaseYear, MusicCategories) VALUES ('Lenny Kravitz','Lenny',
➡'. 'Kravitz','Greatest Hits', 2000, 'Rock/Pop/Best Of')
```

Observe as aspas simples usadas em todos os valores exceto em CDReleaseYear. Essas aspas são necessárias em qualquer valor textual empregado nas instruções SQL para diferenciá-lo dessas instruções. Já que CDReleaseYear não é um valor textual, não precisa ser colocado entre aspas.

É possível saltar a lista dos nomes dos campos, contanto que a de valores esteja na ordem correta. No entanto, esse é outro exemplo de como você pode economizar tempo na criação do código, porém tornando sua manutenção mais difícil e fazendo com ele esteja mais propenso a apresentar alguma falha no futuro.

Alterando Registros

A SQL fornece a instrução `UPDATE` para que modificações sejam feitas em registros existentes, permitindo que você altere de imediato um ou mais campos de quantos registros quiser. Essa instrução possui a sintaxe básica a seguir:

SINTAXE

```
UPDATE <Nome Tabela> SET <nome campo> = <valor novo>
```

Como na instrução `SELECT`, a instrução `UPDATE` pode ter uma cláusula `WHERE`. Sem essa cláusula, ela executará sua atualização em todos os registros da tabela. Portanto, a instrução

```
UPDATE Disc SET CDTitle = 'Novo Título'
```

configuraria todos os campos `CDTitle` como 'Novo Título', que provavelmente não é o resultado pretendido. Embora em geral seja atualizado apenas um registro, é útil poder alterar vários deles com uma instrução `UPDATE`. Novos valores configurados com o uso dessa instrução podem ser baseados no conteúdo existente no campo. Como exemplo, considere esta atualização em uma tabela fictícia que fornece a cada funcionário 10% de aumento:

```
UPDATE Funcionario SET Salario = Salario * 1.1
```

Na maioria dos casos, no entanto, será preferível especificar uma cláusula `WHERE` para restringir os registros afetados a um grupo menor ou mesmo a um único item. A instrução SQL a seguir atualiza todos os registros dos Barenaked Ladies na tabela `Disc` alterando seu valor em `MusicCategories` para `Pop`:

```
UPDATE Disc SET MusicCategories = 'Pop' WHERE ArtistName = 'Barenaked Ladies'
```

Esse tipo de reavaliação de categorias e agrupamentos em um banco de dados é algo que você pode querer fazer de vez em quando para evitar confusão.

Embora os exemplos desta lição tenham sido simples, a instrução `UPDATE` pode executar ações mais complexas. Por exemplo, você pode atualizar vários campos de uma só vez, separando cada atribuição com uma vírgula, e tornar uma cláusula `WHERE` tão intrincada quanto quiser.


```
UPDATE Disc SET ArtistName = 'Sting',  
               ArtistFirstName = 'Gordon',  
               ArtistLastName = 'Sumner'  
WHERE ArtistName = 'Sting' OR ArtistName = 'Mr.Sting'
```

Removendo Registros Indesejados

Você recuperou, adicionou e alterou seus registros, sem executar uma operação básica, a exclusão. A SQL fornece uma instrução DELETE que usa uma sintaxe simples:

SINTAXE

```
DELETE <Tabela> WHERE <critério>
```

Nesse caso, como com UPDATE, a cláusula WHERE é opcional, porém importante porque sem ela, você excluirá todos os registros da tabela especificada.

Na tabela Disc, você poderia usar essa instrução para remover um certo grupo de registros ou, o que é mais comum, excluir apenas um. A instrução SQL mostrada aqui removeria todos os discos de Gordon Lightfoot (exceto um):

```
DELETE Disc  
WHERE ArtistFirstName = 'Gordon'  
       AND ArtistLastName = 'Lightfoot'  
       AND CDTitle != 'Summertime Dream'
```

Para Onde Ir a Partir Daqui Abordando a SQL

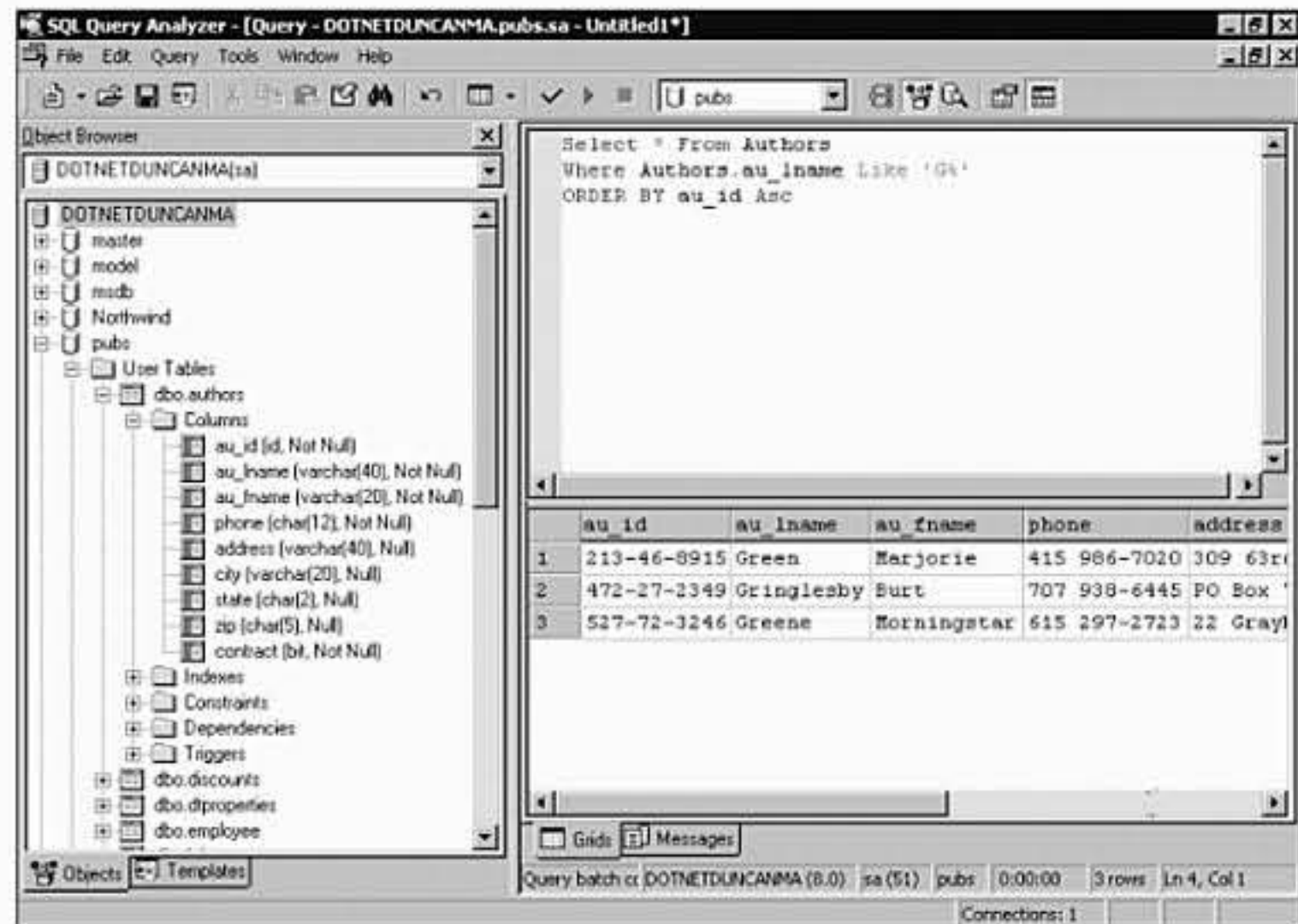
A SQL é realmente importante, mas também é um tópico extenso e complexo em sua abrangência. Você aprenderá mais detalhes sobre ela no decorrer desta lição, mas esses exemplos não fornecem uma abordagem completa do assunto. Quase todos os bancos de dados usam a SQL em algum nível, e há vários livros dedicados somente a essa linguagem. Alguns muito bons são:

- *Desenvolvendo Bancos de Dados na Web – Passo a Passo*, Buyens
- *Dominando SQL Server 2000*, Gunderloy
- *MS SQL Server 2000 – Passo a Passo*, Riordan

Outra boa fonte, que você provavelmente já possui, é o material da biblioteca Microsoft Solution Developers Network (MSDN), que faz parte da instalação do Visual Studio. Tente pesquisar os comandos SQL descritos anteriormente, rastreando a seção de referências à SQL da biblioteca MSDN. Se quiser praticar a SQL fora dos exemplos que executará neste livro, poderá fazê-lo com qualquer programa de banco de dados que tiver disponível. Se tiver o SQL Server instalado, execute o Query Analyzer (veja a Figura 11.1), que é uma das ferramentas que vem com o SQL Server e está disponível como um ícone na pasta do menu desse programa. Essa ferramenta foi projetada para permitir a execução de consultas e a visualização dos resultados, embora seja recomendável ter cautela ao executar algo que não seja uma consulta SELECT até que já se tenha mais segurança para especificar as cláusulas WHERE.

FIGURA 11.1

O Query Analyser permite que você execute instruções SQL direcionadas a seu banco de dados, mas o recurso mais avançado se encontra na sua capacidade de analisar o desempenho e a estrutura das consultas.



Problemas Comuns dos Bancos de Dados e Suas Soluções

Os softwares de bancos de dados armazenarão suas tabelas, campos e registros, mas como os dados serão estruturados é algo que só compete a você e, portanto, é aí que a maioria dos problemas ocorre. Há muitas maneiras diferentes pelas quais é possível organizar o mesmo conjunto de dados, mas apenas algumas delas funcionarão bem em um sistema de banco de dados. O objetivo é produzir um banco de dados que seja eficiente para a pesquisa, mas é aconselhável dificultar a introdução de dados inconsistentes. A inconsistência é o maior problema dos bancos de dados. Se partes diferentes dele não estiverem em concordância, os dados não serão confiáveis. Considere o exemplo de uma tabela, na qual intencionalmente cometemos alguns dos erros mais comuns. Nessa tabela, é fácil gerar dados inconsistentes porque ela não foi estruturada para evitar ou pelo menos minimizar esses problemas. Percorrerei cada um dos principais problemas de uma tabela desse tipo e mostrarei como evitá-los nos bancos de dados que forem criados.

11

Inconsistências de Atualização

Suponha que um usuário decida atualizar uma linha da tabela Disc porque o valor de Artist está incorreto. Talvez Sting tenha sido especificado quando o certo seria Bruce Willis. O usuário poderia executar essa instrução SQL:

```
UPDATE Disc SET ArtistName = 'Bruce Willis'
WHERE ArtistName = 'Sting' and CDTitle = 'The Return Of Bruno'
```

Essa é uma instrução SQL perfeitamente adequada, mas, se a linha original de dados fosse como a descrita a seguir:

ArtistName, ArtistFirstName, ArtistLastName, CDTitle, CDReleaseYear,
 ➡ MusicCategories

agora teria esta aparência:

Bruce Willis, Gordon, Sumner, The Return Of Bruno, 1986, Rock/Pop

ArtistName reflete a informação correta, mas ArtistFirstName e ArtistLastName não o fazem, o que é uma inconsistência. Isso poderia ter sido evitado se todos os três campos tivessem sido atualizados, mas o usuário ou o software teria de assegurar que os dados fossem consistentes. Você tem de fazer tudo que puder no nível do projeto do banco de dados para evitar a inconsistência.

Faça	Não Faça
Estruture seu banco de dados para evitar a inconsistência.	Não confie no fato de que seus usuários, inclusive seus próprios programas, irão trabalhar com os dados exatamente da maneira esperada.

Outra inconsistência comum que pode ocorrer quando se usa uma estrutura de banco de dados como a da tabela Disc é que podem existir muitas variações de dados que deveriam ser idênticos. Aqui estão três linhas da tabela Disc:

Barenaked Ladies, -, -, Maroon, 2000, Pop
 The Barenaked Ladies, -, -, Gordon, 1993, Pop
 Barenaked Ladies, -, -, Stunt, 1999, Rock/Pop

Uma consulta SQL criada para encontrar todos os álbuns dos Barenaked Ladies poderia ser escrita da seguinte maneira:

```
SELECT * FROM Disc WHERE ArtistName = 'Barenaked Ladies'
```

Esse código retornaria a primeira e a última das três linhas listadas do código anterior, mas não encontraria o CD chamado 'Gordon'. Isso resultaria em um erro exibido para o usuário ou algo parecido. O usuário simplesmente não conseguiria a informação correta.

O terceiro problema com relação aos campos Artist ocorrerá se um artista alterar seu nome. (Essa situação não é comum quando se trata de música, mas se você estivesse lidando com departamentos de uma empresa ou nomes de produtos em outros bancos de dados, seria um grande problema.) Suponha que um artista, selecionemos de modo aleatório o Prince, altere seu nome da simples palavra 'Prince' para um símbolo impronunciável (que, infelizmente, não faz parte do código ASCII e, portanto, você não poderá inseri-lo em seu banco de dados). Seria necessário atualizar cada registro que no momento tivesse o nome Prince nele. É claro que essa seria uma instrução simples:

```
UPDATE Disc SET ArtistName = 'O artista anteriormente conhecido como Prince'
WHERE ArtistName = 'Prince'
```

Pressupondo que ninguém tenha escrito Prince errado em algum local, essa instrução SQL atualizaria todos os registros necessários. Novamente, no entanto, você está confiando no fato de que o usuário ou o software esteja ciente de que é preciso executar várias alterações. Pode ser que

eles não usem uma instrução UPDATE como a anterior. Eles podem alterar apenas uma entrada para o novo valor, em vez de todos os discos de Prince, tornando o banco de dados mais uma vez inconsistente. A situação fica ainda pior quando descobrimos que precisamos alterar todos os registros retornando-os para 'Prince' no final!

Todas essas questões resultam de um problema na estrutura do banco de dados – as informações sobre o artista são armazenadas com o registro de cada disco, embora o nome do artista na verdade seja um conjunto independente de dados. Para evitar os problemas descritos, você deve separar as informações sobre o artista para que fiquem em sua própria tabela e armazenar apenas um valor com o disco para vincular as duas tabelas. Para esse valor, poderia ser usado o campo ArtistName, mas isso trará problemas se ele for alterado em algum momento.

Em geral é melhor escolher um valor que com certeza não vá ser alterado a menos que o registro vinculado, na tabela Artist, seja excluído. Ao examinarmos o conjunto atual de informações sobre o artista – os valores de ArtistName, ArtistFirstName e ArtistLastName –, não há nada que possamos garantir que não será alterado. Portanto, você adicionará um campo exclusivo para esse fim. Com frequência é isso que acontece, um campo adicional tem de ser criado apenas para atuar como um identificador exclusivo de um registro, e o método mais comum é gerar um campo contendo um número que seja automaticamente incrementado. Inserindo esse campo a seu conjunto de informações sobre o artista, obteremos a tabela Artist a seguir:

```
ArtistID, ArtistName, ArtistFirstName, ArtistLastName
1, Sting, Gordon, Sumner
2, The Barenaked Ladies, -, -
3, Bon Jovi, -, -
4, Queen, -, -
5, The Police, -, -
...
45, Janet Jackson, Janet, Jackson
```

Cada registro da tabela Disc ainda precisa ser associado a um artista, portanto, você adicionará um campo a essa tabela que terá um valor de ArtistID para cada registro.

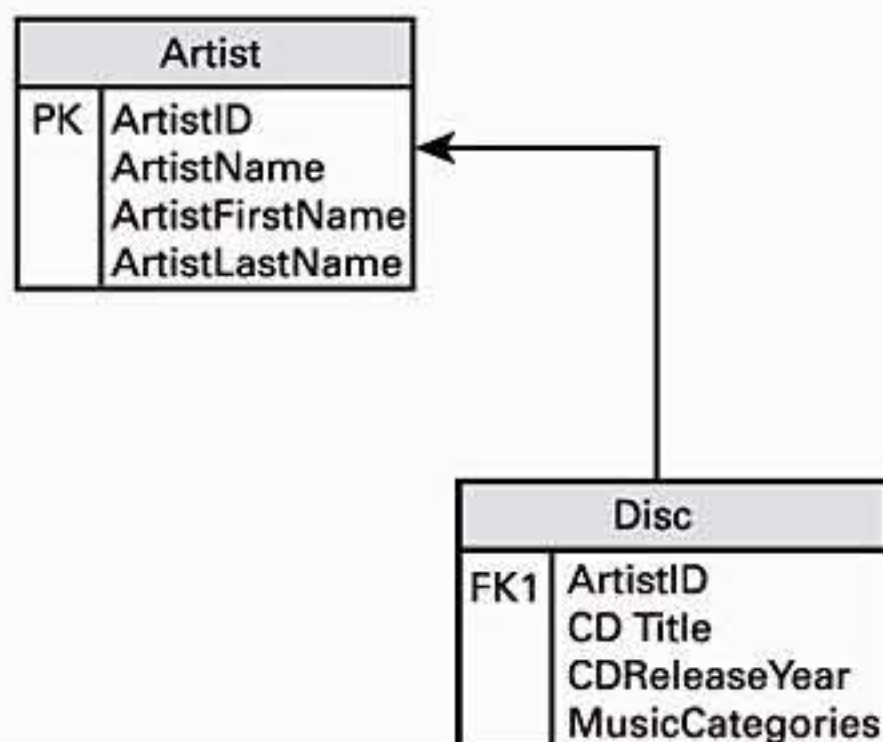
A tabela Disc agora teria a aparência a seguir:

```
ArtistID, CDTitle, CDReleaseYear, MusicCategories
1, Brand New Day, 2000, Pop
4, Greatest Hits, 1988, Rock/Pop
...
```

O campo ArtistID é chamado de chave porque é usado para identificar uma entidade individual. Na tabela Artist, em que ArtistID identifica apenas um artista específico, esse campo seria denominado *chave primária*. Na tabela Disc, ele pode ser chamado de *chave externa* porque é empregado como um vínculo de retorno a uma tabela diferente (veja a Figura 11.2). Uma tabela em geral contém várias chaves externas, vinculando-a a outras tabelas, porém possui só uma chave primária (embora mais de um campo possa ser usado ao mesmo tempo para formar uma chave primária).

FIGURA 11.2

As tabelas *Artist* e *Disc* contêm campos-chave que vinculam uma à outra.

**NOVO TERMO**

A *chave* é um valor, em geral um campo, que identifica uma entidade, como o nome de um estudante, de um autor ou um número de seguro social. Quando um campo identifica exclusivamente cada registro de uma tabela, é chamado de *chave primária* da tabela. A *chave primária* de uma tabela que é inserida em um campo de outra tabela para vincular as duas entidades é conhecida na última tabela como *chave externa*.

NOVO TERMO

Quando duas tabelas estão vinculadas pelo uso de chaves primárias e externas, diz-se que elas estão *relacionadas*.

Por causa da transferência dos dados do campo *Artist* para uma tabela separada, as instruções necessárias para adicionar e recuperar informações terão de ser alteradas. Para adicionar dados, será preciso executar duas instruções *INSERT*, uma na tabela *Artist* e outra na tabela *Disc*, mas a de *Artist* ocorrerá apenas uma vez a cada artista, e não a cada disco. A criação de tabelas vinculadas separadas elimina os problemas discutidos anteriormente, mas gera uma nova maneira de introduzir dados inconsistentes – chaves externas inválidas. Se um usuário ou programa remover um valor do campo *Artist*, então, todos os discos que fazem referência a essa identificação do artista repentinamente se referirão a um registro da tabela *Artist* que não existe. Discutirei como lidar com esse problema um pouco mais adiante nesta lição, na seção “Integridade Referencial”. A maneira de recuperar dados necessários no banco de dados também foi alterada, já que as informações que você deseja agora estão contidas em duas tabelas. Os métodos que usaremos para obter esses dados também serão abordados ainda nesta lição, na seção “Associações: Consultando Várias Tabelas de uma Só Vez”.

Inconsistência entre Locais *Versus* Dados Incorretos

No novo layout da tabela, as informações sobre o artista estão armazenadas na tabela Artist, separadas das relacionadas ao CD, que se encontram em Disc. O objetivo dessa separação é coibir informações inconsistentes. Queremos evitar situações em que duas partes diferentes do banco de dados sejam discordantes sobre o mesmo trecho de dados. Não estamos tentando impedir a ocorrência de dados incorretos, embora esse seja um problema real.

Na verdade não é um problema de um projeto de banco de dados se o campo FirstName de Sting for configurado como George em vez do nome certo Gordon, contanto que todo o banco de dados esteja consistente e armazenando apenas um valor, no caso George. Isso pode parecer um pouco estranho porque os dados incorretos são uma questão de importância relevante, mas o objetivo do projeto do banco de dados é evitar a introdução de erros devido ao layout dos dados. O software e seus usuários é que precisam estar atentos para evitar os dados incorretos.

Campos Multivalorados

Um dos campos, MusicCategories, pode conter mais de um valor de uma só vez, por meio do formato a seguir: Rock/Pop. Esse tipo de campo precisa ser removido de seu banco de dados, ou você terá vários problemas com ele. O primeiro problema ocorrerá quando o usuário tentar recuperar dados com base nas informações contidas ali:

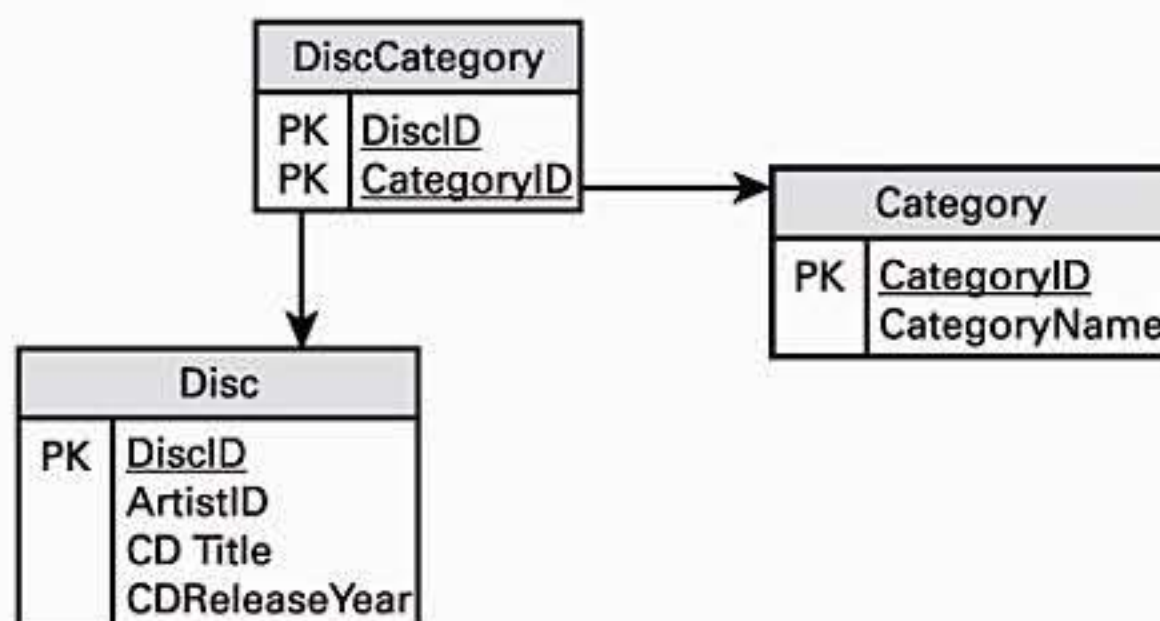
```
SELECT * FROM Disc Where MusicCategories = 'Rock'
```

Isso retornaria os discos em que o valor de MusicCategories fosse igual a Rock, mas não os de Rock/Pop, Best of/Rock ou qualquer outra informação relativa a mais que apenas uma categoria. É preciso ter algum conhecimento sobre o objetivo do banco de dados. Nesse caso, a finalidade do campo MusicCategories era agrupar os CDs de um ou mais tipos de música. Se o usuário quisesse encontrar todos os seus discos de música pop ou que se enquadrassem tanto no conjunto rock quanto em 'clássicos' (seria Rock/Best Of ou Best Of/Rock?), os problemas resultantes tornariam essa categorização inútil. Atualizar os dados de um campo multivalorado também é quase tão difícil quanto a situação comentada. Terminaríamos tendo de adicionar o novo valor a um já existente, nos certificando de que ele já não estivesse lá! Tudo ficaria ainda mais complicado se em algum momento decidíssemos renomear uma categoria, alterando-a de Pop para Popular por exemplo, uma tarefa que seria extremamente difícil de realizar com uma instrução UPDATE.

Para corrigir essa instrução multivalorada e evitar todos os problemas decorrentes dela, separaremos os dados em outra tabela. Diferente das informações sobre o artista, no entanto, só uma tabela nova não pode manipular o relacionamento entre as categorias e os discos. Cada artista pode ser associado a um número qualquer de discos (ou a absolutamente nenhum). Esse relacionamento é chamado *um-para-muitos* ou, o que seria mais correto, *nenhum-para-muitos*. O relacionamento entre o disco e a categoria é mais complexo; uma única categoria pode ser associada a muitos discos e apenas um disco pode ser associado a muitas categorias. Isso produz um relacionamento muitos-para-muitos que você poderá modelar usando duas tabelas novas. Uma armazenará as próprias categorias, a outra vinculará os discos às categorias (veja a Figura 11.3).

FIGURA 11.3

Modelar um relacionamento muitos-para-muitos requer três tabelas: as duas entidades que estão relacionadas e uma tabela especial que exista apenas para conectá-las.



As tabelas resultantes obedecem um padrão que sempre poderá ser usado para representar um relacionamento muitos-para-muitos. Detalharei como fazer consultas em relacionamentos do tipo um-para-muitos e muitos-para-muitos na seção a seguir.

Associações: Consultando Várias Tabelas de uma Só Vez

Agora que você separou de modo adequado seus dados em várias tabelas, será preciso mais do que as simples consultas SELECT que executou anteriormente para obter as informações que deseja. Neste momento há algumas tabelas distintas, porém uma se relacionando com a outra através de *campos-chave*. Usando esses mesmos campos-chave já podemos associar os dados de uma tabela com os correspondentes da outra tabela, ou tabelas, e produzir um único conjunto de resultados. Há diversas maneiras de construir a instrução SELECT quando se quer trabalhar com múltiplas tabelas, mas veja a seguir um exemplo:

CÓDIGO

```
SELECT Artist.ArtistName Disc.CDTitle FR M Artist Disc
WHERE Artist.ArtistID = Disc.ArtistID
```

RESULTADO

```
Sting          Nothing Like The Sun
Sting          Brand New Day
The Barenaked Ladies Maroon
The Barenaked Ladies Gordon
...
```

NOTA

Quando você tiver mais de uma tabela envolvida em uma instrução SELECT, é possível que um campo com o mesmo nome exista em várias delas. Nesse caso, é preciso especificá-lo utilizando o formato `tabela.campo` em sua instrução SELECT para assegurar que o banco de dados compreenda qual o campo desejado. No entanto, essa sintaxe pode ser usada em qualquer situação, mesmo quando só houver uma tabela envolvida.

Essa instrução informa ao banco de dados que recupere todos os registros da tabela Disc e, para cada um deles, faça uma pesquisa (usando a identificação do artista – `ArtistID` – armazenada na tabela Disc) na tabela Artist em busca do nome correspondente do artista (`ArtistName`). Isso pa-

rece trabalhoso, mas, felizmente, todo o trabalho é manipulado pelo banco de dados, e ele o faz de modo muito rápido. Portanto, embora você tenha removido cuidadosamente `ArtistName` da tabela `Disc`, ainda poderá produzir os mesmos resultados associando as tabelas `Disc` e `Artist`.

NOVO TERMO

Vincular duas tabelas em uma consulta para recuperar dados relacionados chama-se *associação*.

A última instrução SQL usava uma cláusula `WHERE` para associar as duas tabelas, o que com certeza funcionará, mas não é a melhor maneira de criar associações. Em vez disso, você aprenderá uma sintaxe nova com a qual poderá especificar as informações sobre a associação como parte da cláusula `FROM` de sua instrução SQL. A reconstrução da instrução anterior com a sintaxe correta produzirá

```
SELECT Artist.ArtistName, Disc.CDTitle  
FROM Disc INNER JOIN Artist ON (Artist.ArtistID = Disc.ArtistID)
```

Essa instrução gerará os mesmos resultados que a do exemplo anterior que usava a cláusula `WHERE`, e pode ser chamada de *associação interna*. É possível especificar `Disc JOIN Artist`, e a palavra `INNER` ‘omitida’ será incorporada, porém o Access não aceita essa forma abreviada.

Associações Internas *versus* Externas

Com uma associação interna, a consulta retorna apenas os registros em que uma correspondência foi encontrada nas duas tabelas. No exemplo, isso significa que se um artista foi listado na tabela `Artist`, mas não teve *nenhum* registro correspondente na tabela `Disc`, ele não fará parte do resultado. Se esse não for o resultado esperado, se em vez disso você quisesse todos os artistas listados na tabela `Artist` em seus resultados (mesmo se não tivesse nenhum disco de alguns deles), poderia usar uma associação externa. As associações externas vinculam as tabelas utilizando as mesmas informações, mas se não conseguirem encontrar nenhum valor correspondente em uma delas, mesmo assim retornarão os campos da primeira e anularão (removerão) os valores dos campos que estiverem na segunda tabela. A palavra-chave `OUTER` pode ser precedida por `LEFT` ou `RIGHT` para especificar em que direção a associação deve trabalhar.

Você deve estar se perguntando qual tabela seria a primeira e qual seria a segunda. Bem, se você especificar `RIGHT OUTER JOIN`, então, será considerada a primeira tabela aquela que estiver à direita da instrução `JOIN`. Se especificar `LEFT OUTER JOIN`, a tabela à esquerda será a primeira.

Digamos que, no banco de dados do exemplo, você tivesse o artista Limp Bizkit na tabela `Artist`, mas não houvesse nenhum disco na tabela `Disc` executado por ele. A instrução `INNER JOIN` mostrada no código a seguir produziria um conjunto de registros, mas o artista Limp Bizkit não seria mencionado nos resultados.

```
SELECT Artist.ArtistName, Disc.CDTitle  
FROM Disc INNER JOIN Artist ON (Artist.ArtistID = Disc.ArtistID)  
ORDER BY Artist.ArtistName
```


Por outro lado, se você usasse uma instrução `OUTER JOIN` como mostro a seguir, o campo `Artist-Name` seria retornado contendo `Limp Bizkit`. Já que nenhum registro correspondente seria encontrado na tabela `Disc`, `CDTitle` teria um valor nulo para esse registro.

```
SELECT Artist.ArtistName, Disc.CDTitle  
FROM Disc LEFT OUTER JOIN Artist ON (Artist.ArtistID = Disc.ArtistID)
```

Como você viu nos exemplos, o resultado de uma associação interna pode ser enganoso porque inclui apenas os valores em que uma correspondência foi encontrada entre as duas tabelas. Mesmo assim, esse tipo de associação é adequado para a maioria das consultas.

Relacionamentos Muitos-para-Muitos

Para modelar o relacionamento muitos-para-muitos dos CDs com as categorias, você terminou com três tabelas distintas: `Disc`, `Category` e uma terceira criada apenas para esse relacionamento, chamada `DiscCategory`. Para fazer consultas nessas tabelas, será preciso usar duas associações — uma entre `Disc` e `DiscCategory` e outra entre `DiscCategory` e `Category`. Por exemplo, a instrução SQL para recuperar todos os CDs que foram atribuídos à categoria `Pop` é mostrada a seguir:

```
SELECT Disc.CDTitle  
FROM Category INNER JOIN  
    (Disc INNER JOIN DiscCategory ON Disc.DiscID = DiscCategory.DiscID)  
    ON Category.CategoryID = DiscCategory.CategoryID  
WHERE Category.Category = "Pop"
```

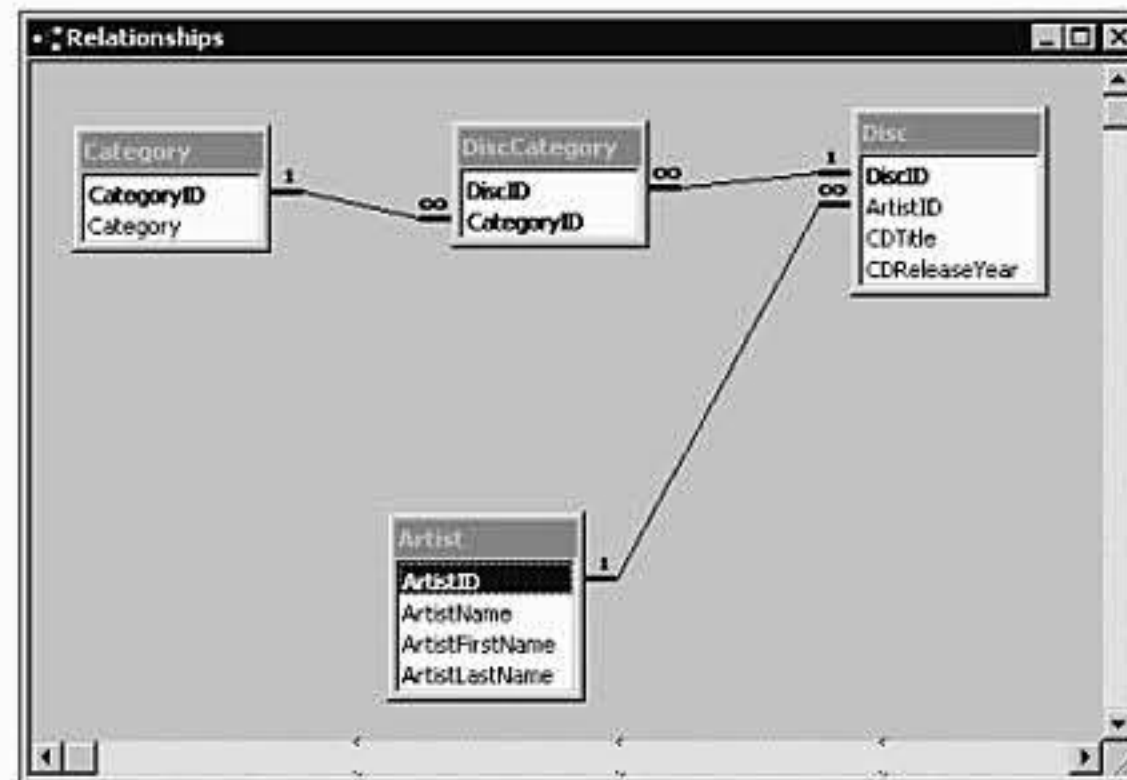
Quando você tiver várias tabelas relacionadas, poderá usar instruções de associação aninhadas como as mostradas no código anterior, no qual uma tabela está na verdade sendo associada ao resultado da associação de duas outras.

Integridade Referencial

Nos casos em que existem várias tabelas, há mais probabilidade de serem encontradas inconsistências. Como discutimos, se um artista for referenciado por sua identificação em um ou mais registros da tabela `Disc`, então você terá dados inconsistentes se ele for excluído. Para evitar isso, faça com que o banco de dados tenha conhecimento dos vínculos entre suas tabelas definindo as chaves e o relacionamento (veja a Figura 11.4). Quando o banco de dados tiver informações dos relacionamentos existentes em seu sistema, ele poderá garantir sua integridade impedindo a exclusão de um registro se esse tiver correspondências em outras tabelas. No sistema específico que criamos, ter os relacionamentos no banco de dados impediria (o banco de dados retornaria um erro) que o usuário excluísse um registro da tabela `Artist` se a tabela `Disc` possuísse algum registro com a identificação desse artista.

FIGURA 11.4

Em seu programa de banco de dados, sendo ele o Access ou o SQL Server, você pode criar relacionamentos para que esses possam ter sua integridade garantida.



Criando Chaves Primárias

No exemplo, você terminou com chaves numéricas para todas as tabelas (`ArtistID`, `CategoryID`, `DiscID`), mas o que não abordamos é a necessidade de obter ou gerar essas chaves sempre que um novo registro for inserido. Há muitas maneiras pelas quais é possível manipular esse requisito: recursos internos do banco de dados, cálculo na hora da inserção, usar GUIDs em vez de números simples, ou um sistema que gerencie e incremente as chaves chamado gerenciador de chaves.

Recurso do Banco de Dados

O primeiro método para gerenciar as chaves está embutido em muitos bancos de dados e é um tipo especial de campo que incrementa automaticamente seu valor quando uma nova linha é adicionada. Esses campos são sempre numéricos; eles podem ser configurados para acrescentar uma quantidade definida, quase sempre 1, e para iniciar a contagem em um valor específico, que também é quase sempre 1. Essa é uma boa maneira de manipular as chaves porque o banco de dados realizará todo o trabalho, mas, sob certas circunstâncias, uma chave de geração automática pode causar problemas. Por exemplo, se você tivesse de adicionar um novo CD à coleção e ele fosse de um artista não existente em seu banco de dados, seriam necessárias duas instruções `INSERT` – uma na tabela `Artist` e, em seguida, outra na tabela `Disc`. Usando as chaves de geração automática, não adicionaríamos um valor para `ArtistID` na primeira inserção; em vez disso, ele seria gerado de modo automático. Entretanto, na segunda inserção (em `Disc`), precisaríamos que o novo valor de `ArtistID` vinculasse o novo CD ao novo artista. Empregando apenas a SQL, essa identificação não seria retornada.

Você poderia fazer outra consulta no banco de dados, mas teria de usar campos como `ArtistName` para encontrar os dados novamente, e isso estaria propenso a erro. Essa é uma razão pela qual é necessário um método alternativo de obter as chaves; outra razão será discutida ainda nesta lição em “Identificadores Exclusivos (GUIDs – Globally Unique Identifiers)”.

Chaves Calculadas

Como alternativa às chaves geradas por bancos de dados, você pode usar as instruções SQL para determinar a próxima identificação na sequência antes de fazer a inserção necessária. Isso envolve o emprego de um tipo especial de operador SQL chamado *operador de agregação*, que recupera um valor calculado com base em todos os campos encontrados por uma cláusula SELECT. Há vários desses operadores de agregação, mas utilizaremos o chamado de MAX, que retorna para um campo o valor mais alto extraído entre todos os registros selecionados.

Para determinar a próxima identificação, você poderia executar esta instrução (usando a tabela Artist como exemplo):

```
SELECT MAX(ArtistID) FROM Artist
```

Essa instrução retornaria apenas o registro de um campo contendo o valor mais alto relativo às identificações existentes atualmente na tabela. Adicione um a esse valor, e você terá a próxima identificação, que então será usada em uma inserção.

Essa abordagem, embora válida em algumas circunstâncias, apresenta dois problemas principais. O primeiro ocorre porque mais de um programa poderia estar usando ao mesmo tempo esse banco de dados – dois ou mais usuários poderiam estar adicionando artistas no mesmo momento. Em uma situação dessas, seria possível que alguns usuários executassem sua primeira instrução SELECT antes que o usuário anterior tivesse executado sua instrução INSERT, portanto eles teriam obtido o mesmo valor para a identificação.

Esse problema pode ser evitado com o uso de transações, outro dos conceitos de bancos de dados que não abordarei nesta lição.

O segundo problema não pode ser evitado com tanta facilidade. Ele ocorre quando um ou mais registros do final da tabela (as identificações mais altas) são excluídos. Qualquer registro novo primeiro selecionará MAX (ArtistID) e obterá os valores de identificação anteriormente atribuídos aos registros excluídos. Em um sistema em que se pretende que os valores das identificações sejam exclusivos, ter registros diferentes compartilhando a mesma identificação em momentos distintos não é muito aconselhável.

Identificadores Exclusivos (GUIDs – Globally Unique Identifiers)

Em algumas situações, uma chave crescente – criada pelo banco de dados ou por meio de seus próprios cálculos – não é adequada. Um exemplo em que isso acontece seria quando temos bancos de dados distribuídos, nos quais há várias cópias do banco de dados e essas são regularmente unificadas. As inserções podem ocorrer em quaisquer das cópias. Se você usasse uma chave com o recurso incremental, então, sempre que as inserções ocorressem em mais de um local, seriam produzidas identificações duplicadas. Uma solução seria empregar alguma forma de identificação gerada ao acaso, aleatória o suficiente para que pudesse ser criada em vários locais com pouca ou nenhuma chance de ser duplicada. O método em geral aceito para a criação de identificações aleatórias é a utilização de um GUID (Globally Unique Identifier), um número

aleatório de 128 bits gerado pelo banco de dados ou pelo sistema operacional. O GUID é aleatório o bastante para que seja usado em um cenário de bancos de dados distribuídos.

Gerenciador de Chaves

O último esquema para a criação de identificações que discutirei é adequado para sistemas que não são distribuídos (veja a seção anterior sobre os GUIDs), mas para os quais você precisará realmente obter a chave no momento exato da inserção do registro. (Isso torna problemática uma chave gerada pelo banco de dados.) Esse método, chamado sistema gerenciador de chaves, usa uma tabela adicional em seu banco de dados para armazenar os valores atualmente mais altos das identificações de todas as outras tabelas (veja a Figura 11.5).

FIGURA 11.5

O método utilizado pelo gerenciador de chaves para gerá-las requer uma tabela adicional que contenha o valor mais alto de identificação de cada tabela que usará esse método.

Key Manager	
TableName	HighestID
Disc	35
Artist	7
Category	12

Para obter novas identificações usando esse método, você terá de procurar na tabela do gerenciador de chaves o valor da identificação atual, acrescentar uma unidade a ele, inserir seu novo registro e atualizar essa tabela com o valor da identificação incrementado. As quatro ações separadas significam que esse método compartilha o mesmo problema da existência de vários usuários encontrado na chave calculada. O problema de reutilizar as chaves devido às exclusões é evitado porque o valor da identificação armazenado no gerenciador de chaves nunca é decrescido, e as identificações nunca são reutilizadas.

Procedimentos Armazenados e Chaves de Geração Automática

Enquanto descrevia o tipo de dado de incremento automático como opção para a criação de chaves, mencionei que é difícil obter a chave depois de executada uma instrução `INSERT`, e que essa é uma das razões principais para escolher um método alternativo de geração de chaves. Embora esses comentários anteriores sejam realmente procedentes, há um método em geral usado para obtenção de uma chave gerada pelo sistema depois de uma inserção. Esse método só estará disponível se você empregar um procedimento armazenado para inserir seus registros, o que não será abordado com detalhes neste livro. Se quiser obter mais informações sobre esse método de inserção de registros e obtenção da nova identificação gerada, pesquise a biblioteca do MSDN.

Criando o Banco de Dados de Exemplo

Para os exemplos do restante desta lição e para o código que você irá escrever no Dia 12, “Acessando Dados com a Plataforma .NET”, será preciso sua própria cópia do banco de dados do exemplo. Embora o processo de criação do banco de dados possa ser percorrido manualmente – e essa não é uma idéia ruim se os bancos de dados forem um assunto novo para você –, tentarei tornar isso um pouco mais fácil fornecendo três opções diferentes de banco de dados que poderão ser configuradas em sua máquina.

As três versões foram projetadas para ser usadas em três sistemas de bancos de dados diferentes: Microsoft SQL Server (2000), Microsoft Database Engine (MSDE 2000) ou Microsoft Access. O SQL Server e o MSDE darão suporte à versão de download do mesmo arquivo, permitindo que seja criada apenas uma configuração para os dois sistemas. Todas essas opções funcionarão, mas o melhor produto para se ter a fim de atender à finalidade atual e para a prática futura é o SQL Server. Ele é um servidor popular de banco de dados e provavelmente será seu instrumento de trabalho quando começar a desenvolver sistemas reais. O MSDE é gratuito, o que é um ótimo *preço* por um sistema de banco de dados bem completo, mas fornecerá só algumas ferramentas para o trabalho com dados externos a seu programa. O Access vem com várias versões diferentes do Office 2000/XP e disponibiliza várias ferramentas para a visualização de dados, criação e alteração de tabelas, e até para a construção de suas instruções SQL (o Query Builder), portanto não será uma opção ruim se você já o tiver.

Descubra que produto você tem e, em seguida, siga a direção apropriada nas próximas seções. Se não tiver o Access, o MSDE ou o SQL Server, poderá simplesmente fazer o download do arquivo .mdb (parte do Access abordada na próxima seção) e usá-lo sem instalar o restante do software do banco de dados.

- **CD.mdf e CD.ldf** Juntos, esses dois arquivos contêm todos os dados que serão usados com o SQL Server ou o MSDE.
- **AttachDB.vbs** Antes que os arquivos mdf e ldf possam ser usados a partir de seu código, esse arquivo VBScript (.vbs) deve ser executado para incluí-los em seu SQL Server ou no MSDE local.
- **CD.mdb** O banco de dados Access, que você pode usar se não tiver o SQL Server ou o MSDE.
- **TestDB.vb** O exemplo de um programa no Visual Basic .NET que você pode usar para testar a configuração de seu banco de dados.
- **Orders.txt** Este arquivo é usado no Exercício 2 no fim desta lição.

Access 2000 ou Access 2002

Para os usuários que possuem o Access, ou que não tiverem nenhum sistema de banco de dados, não será necessário criar uma configuração; você só precisará saber o caminho para o arquivo .mdb que usará nos exemplos. Se quiser empregar o Access para trabalhar com os dados, o único

problema que pode encontrar será se tiver uma versão diferente daquela em que o exemplo foi criado. Se esse for o caso, por padrão, só poderá visualizar e não editar as diversas tabelas e outros objetos de seu banco de dados por meio do Access. Se quiser, utilize as ferramentas de conversão de banco de dados do Access a fim de atualizar o banco de dados para a mesma versão que você estiver executando.

MSDE e SQL Server 2000

Tanto para o MSDE quanto para o SQL Server, os dois arquivos (`cd.mdf` e `cd.ldf`) representam um banco de dados isolado. É possível incluir novamente esse banco de dados na sua cópia do SQL ou do MSDE. O código fornecido no arquivo de script `AttachDB.vbs` vinculará esses dois arquivos a seu sistema de banco de dados. Embora o arquivo deva funcionar ‘como está’ para a maioria das pessoas, na verdade ele contém a identificação de usuário e a senha do administrador do sistema em seu código, tendo como padrão as configurações `sa` e uma senha em branco. Se seu sistema não tiver uma senha em branco para a conta `sa`, você precisará editar esse arquivo. Esse arquivo de script também foi projetado para ser executado (com um clique duplo nele) no mesmo diretório dos arquivos `.mdf` e `.ldf` e na mesma máquina de seu sistema de banco de dados. Vá em frente e dê um clique duplo nesse arquivo, depois que tiver feito todas as alterações necessárias na identificação do usuário e na senha armazenadas no código.

Testando a Configuração com System.Data

Agora que você fez o download e configurou os dados de seu exemplo, poderá criar um pequeno aplicativo de console no Visual Basic .NET para confirmar se tudo funciona como deveria. Para tanto, será preciso usar o espaço de nome `System.Data` com o .NET Framework. Essas classes contêm tudo que é necessário para acessar dados em quase todos os tipos de banco de dados e são divididas em duas áreas principais – as classes projetadas para acessar o Microsoft SQL Server (incluindo o MSDE) e as que acessam qualquer banco de dados que possua um driver de OLE DB (o que significa quase todos os bancos de dados).

Não me deterei muito em detalhes sobre como essas classes funcionam. Em vez disso, o exemplo a seguir apenas testará se seu banco de dados está configurado corretamente antes de passarmos para a próxima lição, na qual me aprofundarei no acesso aos dados por meio da plataforma .NET.

O código que será usado para testar o banco de dados estabelecerá uma conexão com ele empregando o conjunto de classes do OLE DB do espaço de nome `System.Data.OleDb`. As classes `System.Data.SqlClient` podem ser utilizadas se você tiver o SQL Server ou o MSDE, mas, já que o Access também pode estar sendo usado, será mais simples utilizar as classes que podem acessar os dois tipos de bancos de dados. Quando empregar um dos métodos de conexão com o banco de dados, o trecho-chave da informação é o que é chamado de *string de conexão*, um texto que fornece todas as informações necessárias para a conexão com seu banco de dados. Esse é o único trecho do código que terá de ser alterado para que se encaixe em sua configuração específi-

ca. A Listagem 11.1 contém o procedimento completo, e fornecerei informações adicionais sobre os valores da string de conexão depois desta listagem.

LISTAGEM 11.1 TextDB.vb

```
1 'Substitua a string de conexão pelo valor apropriado
2 'à sua configuração (explicado nesta lição)
3 'e, em seguida, compile com:
4 ' "vbc /r:System.DLL /r:System.Data.DLL TestDB.vb"
5 Imports System
6 Imports System.Data
7 Module TestDB
8     Sub Main()
9         Dim sConnectionString, sSQL As String
10        sConnectionString = <consulte abaixo >
11
12        sSQL = "SELECT CDTITLE FROM Disc ORDER BY ArtistID Asc"
13
14        Dim connCD As New OleDb.OleDbConnection(sConnectionString)
15        Dim cmdCD As New OleDb.OleDbCommand(sSQL, connCD)
16        Dim drCD As OleDb.OleDbDataReader
17        connCD.Open()
18        drCD = cmdCD.ExecuteReader()
19        Do While drCD.Read()
20            Console.WriteLine(drCD.Item("CDTITLE"))
21        Loop
22
23        drCD.Close()
24        connCD.Close()
25        Console.ReadLine()
26    End Sub
27 End Module
```

Se você usar o banco de dados do Access (CD.mdb), sua string de conexão incluirá o caminho completo para esse arquivo como descrito a seguir:

Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\chapter11\cd.mdb

Certifique-se de alterar essa string de conexão a fim de indicar o caminho apropriado para CD.mdb em sua máquina.

Se, por outro lado, você usar o SQL Server ou o MSDE, e já tiver executado AttachDB.vbs com sucesso, então, a string a seguir deve funcionar em sua máquina:

Provider=SQLOLEDB.1;User ID=sa;Initial Catalog=CD;Data Source=(local)

Esta string de conexão SQL (que é dividida em duas linhas neste livro, mas deve ter apenas uma linha em seu código) pressupõe uma senha em branco para a conta sa, podendo ser alterada para que especifique uma senha se necessário:

```
Provider=SQLOLEDB.1;User ID=sa;Password=peanutbutter;  
➡Initial Catalog=CD;Data Source=(local)
```

Insira a string de conexão apropriada na Listagem 11.1. Observe que esse código está incluído, como `testdb.vb`, na pasta `Day11` que você descarregou anteriormente do site do livro na Web. Agora compile o código como foi orientado nos comentários do início do arquivo, e você deverá ver um arquivo `.exe` que poderá usar para testar sua conexão com o banco de dados. Se tudo der certo, você deve obter o seguinte resultado:

RESULTADO

```
Left Of The Middle  
the tragically hip  
Road Apples  
Day for Night  
Phantom Power  
Brand New Day  
Mercury Falling  
Fields of Gold  
Classic Queen  
20 Greatest Christmas Songs  
Gordon  
Born on a Pirate Ship  
Maroon  
Tails  
Firecracker  
Janet  
The Velvet Rope  
Design Of A Decade 1986-1996  
Mad Season  
Music
```

Com esse resultado, você terá criado um aplicativo que estabelece com sucesso uma conexão com o banco de dados e estará pronto para passar para tópicos mais avançados sobre bancos de dados.

Resumo

Esta lição apresentou uma introdução aos bancos de dados e sua função no desenvolvimento de aplicativos .NET. Ao dar continuidade na criação de sistemas no Visual Basic .NET, você em geral irá usar um banco de dados, independentemente do segmento empresarial para o qual desenvolver seus programas. Com frequência escreverá programas que empregarão um banco de dados já existente.

Às vezes, seu trabalho exigirá que você mesmo projete e implemente esse banco de dados, uma tarefa que pode ser bem complexa de realizar corretamente. Na lição seguinte, continuaremos a trabalhar com bancos de dados, passando da teoria diretamente para a prática escrevendo programas que usam as classes `System.Data`.

P&R

P Nesta lição, você mencionou três tipos diferentes de bancos de dados, mas os bancos de dados da Microsoft são o único tipo com o qual posso me conectar?

R Absolutamente não. Qualquer banco de dados no qual você tenha um driver do OLE DB (ou do ODBC) estará disponível para conexão com as classes `System.Data.OleDb`. Se você não tiver um provedor do OLE DB, mas possuir um driver do ODBC, também poderá se conectar passando por uma camada adicional e usando o provedor do OLE DB para as conexões do ODBC.

Para obter mais informações sobre o OLE DB, acesse <http://www.microsoft.com/data>.

P No banco de dados dos CDs, você removeu as informações sobre o artista da tabela `Disc`, mas, em seguida, teve de retorná-las em quase todas as saídas usando uma associação em sua instrução SQL. Não seria melhor apenas manter uma cópia das informações sobre o artista em cada registro de `Disc`?

R O armazenamento de dados em mais de um local significa que, para evitar inconsistências, você terá de se certificar de atualizar todos os locais possíveis ao mesmo tempo. Os benefícios conseguidos evitando a associação em suas consultas serão perdidos pelo trabalho adicional necessário sempre que atualizar os registros na tabela `Disc` ou na `Artist`.

P A XML foi projetada para armazenar dados. Ela substitui os bancos de dados?

R A XML é uma maneira de armazenar dados e, embora seja certamente uma substituta para alguns sistemas de bancos de dados não relacionais, ela não substitui sistemas de bancos de dados relacionais como o SQL Server. A XML será usada com mais frequência como uma maneira de intercambiar dados entre sistemas e representará a saída gerada por seus sistemas tradicionais de banco de dados.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Em uma tabela de banco de dados, um ou mais campos são usados para identificar de maneira exclusiva cada registro. Como esses campos, ou campo, são chamados?
2. Dadas as duas tabelas mostradas na Figura 11.6, qual seria o resultado da instrução SQL que vemos aqui?

```
SELECT Make.MakeName, Model.ModelName
FROM Make LEFT OUTER JOIN Model ON Make.MakeID = Model.Make
```

FIGURA 11.6

Exibição das tabelas referentes à pergunta 2 do teste.

Make	
MakeID	MakeName
1	Ford
2	Audi
3	BMW
4	Pontiac
5	Toyota

Model		
ModelID	Make	ModelName
1	1	Mustang
2	1	Explores
3	4	Grand Ann
4	4	Grand Prix
5	4	Azek
6	5	Rav 4
7	5	Camry

3. Examinando novamente as tabelas da Figura 11.6, qual seria a saída desta instrução SQL?

```
SELECT Make.MakeName, Model.ModelName
FROM Make INNER JOIN Model ON Make.MakeID = Model.Make
```

Exercícios

1. Se você quisesse expandir sua biblioteca de registro de CDs para que armazenasse a coleção de CDs de mais de um usuário, como alteraria seu banco de dados? Considere que precisaria registrar informações sobre os próprios usuários e sobre a propriedade dos discos.
2. Dado o conjunto de informações contido em `Orders.txt`, como você projetaria um banco de dados para armazenar as mesmas informações?

PÁGINA EM BRANCO

SEMANA 2

DIA 12

Acessando Dados com a Plataforma .NET

Na lição anterior apresentei uma introdução aos bancos de dados e mostrei como eles podem ser usados em vários aplicativos. Nesta lição, examinaremos como seus programas trabalharão com dados do Visual Basic .NET. Esta lição incluirá:

- Uma visão geral da arquitetura de acesso a dados da plataforma .NET.
- Conexão a um banco de dados.
- Execução de instruções SQL.
- O uso da vinculação de dados com os formulários da Web e do Windows.

Além desses tópicos, no final da lição você aprenderá alguns dos conceitos e técnicas mais avançados para o trabalho com bancos de dados da plataforma .NET.

Uma Visão Geral do Acesso aos Dados na Plataforma .NET

Os bancos de dados são usados em quase todos os aplicativos empresariais que já foram criados e em muitos sistemas pessoais e de microcomputadores. Eles já eram encontrados em uma dessas formas mesmo antes da criação do Visual Basic. Esse longo histórico dos bancos de dados e de seu uso em programas de computador assegurou que os programas do Visual Basic já acessem dados e bancos de dados desde suas primeiras versões. Da mesma maneira, a tecnologia

de acesso a dados do Visual Basic evoluiu com o passar do tempo, passando por muitas versões diferentes antes que a plataforma .NET entrasse em cena.

O ADO e o OLEDB

Não obteríamos grandes vantagens percorrendo a história completa que envolve o Visual Basic e o acesso a dados, mas vale a pena fazer um exame resumido da tecnologia mais recente de acesso a dados (anterior à plataforma .NET). Antes da existência dessa plataforma era extensivamente usada pelos programadores do Visual Basic 6.0 uma biblioteca de acesso a dados chamada ActiveX Data Objects (ADO), considerada o principal meio de conectar os sistemas do Visual Basic ou do Visual Basic for Applications (VBA) a quase todos os bancos de dados de back-end. O ADO era uma biblioteca do COM que encapsulava a funcionalidade da tecnologia real de acesso a dados, o OLE DB, e que tinha sido projetada para ser uma maneira fácil de trabalhar com essa nova tecnologia.

Para evitar a necessidade de se criar um código para cada banco de dados com o qual você pudesse se deparar, o ADO/OLEDB usava uma abordagem semelhante ao ODBC (<http://www.microsoft.com/data/odbc>), fornecendo uma única interface de programação independente do banco de dados e, em seguida, usando *drivers* específicos de banco de dados para trabalhar com cada um deles. Esse mesmo conceito já tinha se tornado popular e quase todo sistema de banco de dados possuía um driver de ODBC disponível. O ADO também requer drivers para funcionar, embora eles sejam conhecidos como *provedores* de OLEDB, e muitos dos bancos de dados mais conhecidos disponibilizam o(s) provedor(es) necessário(s). Mas já que o ADO é mais novo do que o ODBC, não existem tantos drivers desse tipo. No entanto, possuir apenas o driver do ODBC já é suficiente, já que o ADO vem com um provedor de OLEDB para os drivers de ODBC. Por meio da combinação dos provedores de OLEDB disponíveis com todos os drivers de ODBC fornecidos, o ADO poderia ser usado para estabelecer uma conexão com quase todos os bancos de dados existentes.

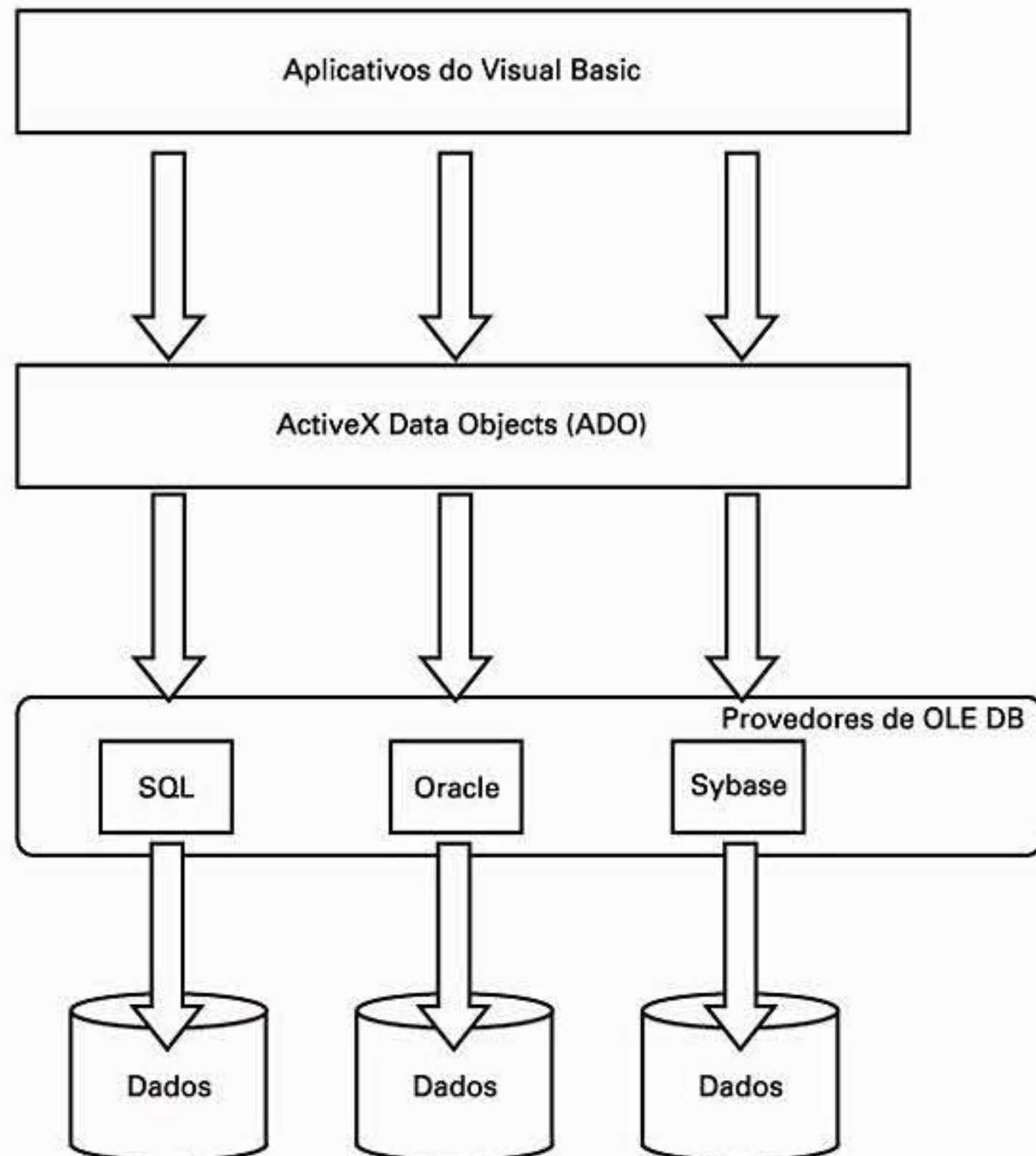
A arquitetura integral do ADO foi estabelecida no fato de que o código que você escrevesse só trabalharia com o ADO, e não com o sistema de banco de dados, porém, o ADO usaria um provedor de OLEDB para converter e transferir suas solicitações para o sistema de banco de dados (veja a Figura 12.1). O ADO expunha vários objetos-chave que representavam a conexão com o banco de dados, os comandos relacionados ao banco de dados e os resultados obtidos nas consultas – todos projetados para ser fáceis de usar como parte de um programa do Visual Basic.

ADO.NET

O ADO.NET é a tecnologia de acesso a dados que faz parte do .NET Framework. Ele será sua maneira de obter dados quando programar no Visual Basic.NET e representa o estágio posterior ao ADO/OLEDB. A tecnologia OLEDB subjacente ainda permanece em seu lugar, e os provedores de OLEDB ainda são o principal método pelo qual o ADO.NET se comunica com sistemas específicos de banco de dados, mas acima dessa camada há muito pouca semelhança com os objetos anteriores do ADO.

FIGURA 12.1

O ADO foi projetado como uma camada acima do OLEDB, que usa tanto drivers de OLEDB quanto de ODBC para se conectar com os bancos de dados.

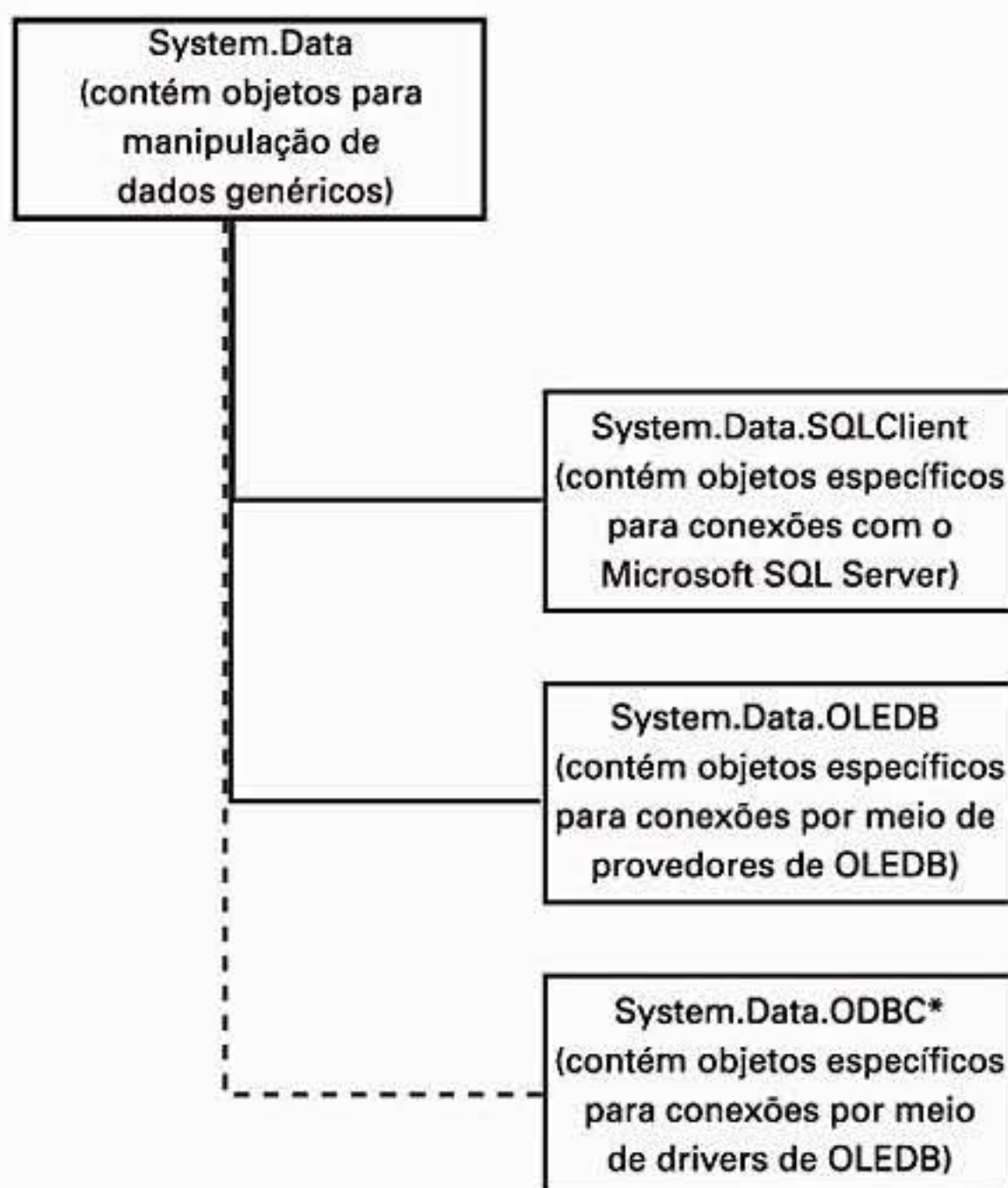


Na plataforma .NET, o acesso a banco de dados é manipulado pelas classes existentes no espaço de nome `System.Data`. Esse espaço de nome é dividido em duas áreas distintas: o conjunto de classes `System.Data.OleDb` e as classes `System.Data.SqlClient`. O primeiro conjunto, `System.Data.OleDb`, foi projetado para permitir que você se conecte a qualquer banco de dados para o qual possua um provedor de OLEDB ou (por meio do provedor de OLEDB para o ODBC) um driver de ODBC e é tecnicamente o equivalente à camada de acesso a dados do ADO original. A segunda área, `System.Data.SqlClient`, foi projetada para trabalhar apenas com o Microsoft SQL Server, mas fornece um conjunto de recursos semelhante às classes OLEDB. A arquitetura geral das classes de acesso da plataforma .NET é mostrada na Figura 12.2, ilustrando os dois sistemas principais.

Nos exemplos desta lição, você aprenderá a trabalhar tanto com bancos de dados OLEDB quanto com sistemas do SQL Server ou do MSDE. Mesmo se só tiver o SQL Server, ainda poderá testar os dois métodos porque é possível se conectar com o SQL Server por meio de seu provedor de OLEDB ou com as novas classes da plataforma .NET. Além das classes `SqlClient` e `OleDb` para acessar os bancos de dados, `System.Data` também inclui várias outras classes projetadas para trabalhar com os dados independentemente de sua fonte específica. Também examinaremos essas outras classes, principalmente `DataSet`, como parte dos exemplos desta lição.

FIGURA 12.2

O ADO.NET é dividido em duas seções principais, OLEDB e SQL, que permitem que você se conecte a uma ampla variedade de fontes de dados.



* Como na versão Beta 2, não incluída no Visual Studio .NET mas disponível como download da Web, o ADO.NET é dividido em duas seções principais, OLEDB e SQL, que permitem que você se conecte a uma ampla variedade de fontes de dados.

Tarefas-padrão dos Bancos de Dados

Em vez de abordar as classes `System.Data` em um formato referencial que examine cada classe, esta seção enfocará as tarefas que normalmente são executadas. Essas tarefas, como a conexão a um banco de dados, a execução de uma instrução SQL ou a recuperação de dados, fornecerão um ponto de partida natural para a discussão dos objetos subjacentes.

Conectando-se ao Banco de Dados

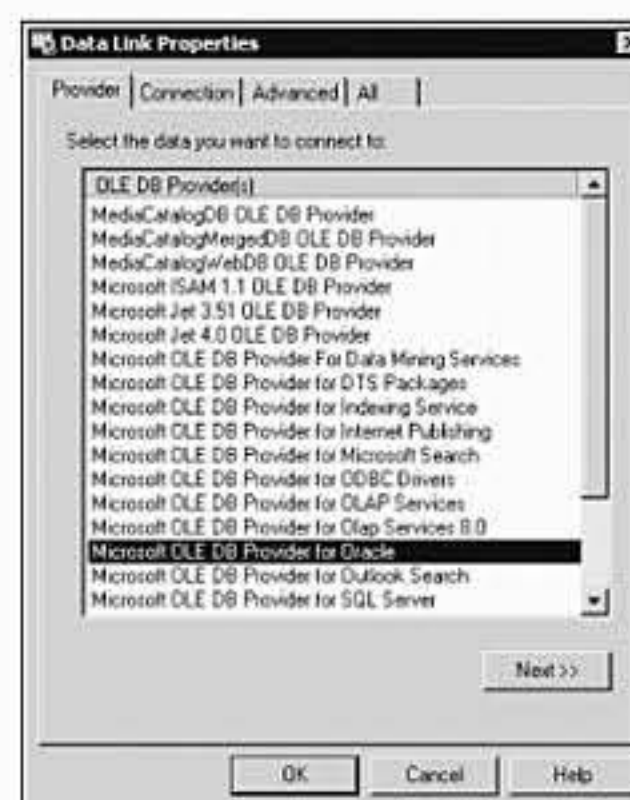
Antes que você possa começar a trabalhar com algum dado, precisará estabelecer uma conexão. A conexão representa e contém todas as configurações necessárias para que seu banco de dados seja encontrado e acessado. Em versões anteriores do ADO, ela envolvia dois itens principais, o objeto da conexão e uma string de conexão. A string de conexão, uma linha de texto que inclui algumas ou todas as informações necessárias para o acesso a seu banco de dados, ainda é a chave para configurar uma conexão de banco de dados no ADO.NET, embora agora existam dois objetos diferentes de conexão (um para o OLEDB e outro para o SQL Server). Antes de começarmos a codificar, devemos obter a string apropriada de conexão com o seu banco de dados. Mesmo sendo apenas um valor textual e podendo, portanto, ser criado manualmente, a maneira mais fácil de obter uma string correta é usar um pouco de astúcia.

Apenas por ter o ADO instalado em seu computador – e ele está incluído como parte da instalação do .NET Framework, portanto você o tem –, um tipo especial de arquivo, o Microsoft Data Link, foi registrado. Os arquivos desse tipo foram projetados para conter informações sobre a conexão com o banco de dados e se você criar um vazio, então, uma interface gráfica com o usuário adequada será fornecida para a geração e edição de todos os detalhes da conexão. Depois de ter trabalhado com essa interface e selecionado as opções apropriadas para seu banco de dados, o arquivo terá a string de conexão que poderá ser copiada e usada em seu aplicativo do Visual Basic .NET. Siga estas etapas para empregar esse artifício:

1. Crie um arquivo Microsoft Data Link vazio por meio de um novo arquivo de texto (dê um clique com o botão direito do mouse na área de trabalho e selecione New, Text Document no menu que aparecerá) e renomeie-o com um título que possua uma extensão .udl (New.udl seria perfeito). Isso será difícil de fazer se as extensões de arquivo não estiverem visíveis para que possam ser ativadas pelo painel de controle Folder Options. Você deve ficar atento para o ícone de alteração do arquivo depois que tiver mudado sua extensão, indicando seu novo tipo de arquivo.
2. Dê um clique duplo no arquivo novo, e uma caixa de diálogo aparecerá com um conjunto de quatro guias para a criação e edição das informações sobre a conexão com seu banco de dados.
3. Na caixa de diálogo que acabou de ser aberta (veja a Figura 12.3), comece com a primeira guia, Provider, e configure as informações corretas para seu banco de dados:
 - Para o arquivo CD.mdb do banco de dados do Access, selecione o provedor Microsoft Jet 4.0.
 - Tanto para o banco de dados do MSDE quanto para o do SQL Server, selecione Microsoft OLEDB Provider for SQL Server.

FIGURA 12.3

A caixa de diálogo de propriedades do Data Link permite que você configure graficamente os detalhes de sua conexão.



4. Na guia Connection, você verá opções diferentes dependendo de qual dos dois provedores disponíveis selecionou na etapa anterior. Para o Access, só será preciso inserir o caminho para o arquivo CD.mdb (veja a Figura 12.4).

FIGURA 12.4

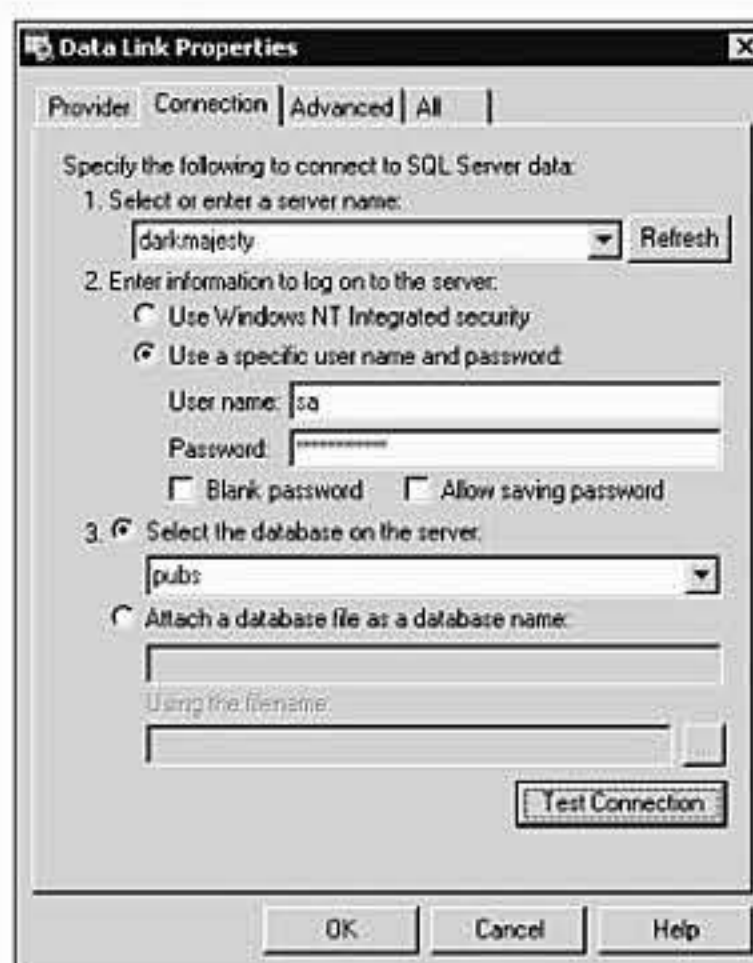
A guia Connection será diferente dependendo do provedor que você selecionar; as opções para o Access incluem o caminho para o arquivo do banco de dados.



Para o SQL, você precisa informar o nome do servidor ou (local) se ele estiver no mesmo computador, a identificação e a senha do usuário e com que banco de dados específico (CD neste caso) quer se conectar. A Figura 12.5 mostra a guia Connection configurada para um banco de dados de um servidor SQL local que usa 'sa' como sua identificação de usuário e uma senha que foi inserida mas não pode ser lida.

FIGURA 12.5

Para o SQL Server, os detalhes da conexão incluem o nome do servidor, do banco de dados específico e a identificação e senha de usuário necessárias para a conexão.



- Embora haja mais duas guias, você concluiu a digitação de informações. Dê um clique no botão Test Connection para ver se as informações que inseriu estão corretas. Se o teste for bem-sucedido, então, feche a caixa de diálogo dando um clique em OK. Agora, use o Bloco de notas ou algum outro editor de textos para abrir o arquivo .udl de modo que possa examinar seu conteúdo. O conteúdo exato dependerá das informações que foram inseridas na caixa de diálogo, mas devemos ver um texto como o descrito a seguir:


```
[oledb]
; Everything after this line is an OLE DB initstring
Provider=Microsoft.Jet.OLEDB.4.0;
  Data Source=C:\CD.mdb;
  Persist Security Info=False
```

A última linha desse arquivo, que começa em `Provider`, é uma string de conexão do OLEDB, que é o que você precisará usar em seu programa do Visual Basic .NET. Se estiver utilizando um banco de dados do Access, então, a string anterior deverá ser a mesma exceto pelo caminho. O conteúdo de um arquivo .udl configurado para o SQL Server é mostrado no exemplo a seguir, mas pode ser preciso alterar ao valores da identificação do usuário (`userid`), da senha (`password`) e da fonte de dados (`Data Source`) para que esta string funcione em seu sistema:

```
[oledb]
; Everything after this line is an OLE DB initstring
Provider=SQLOLEDB.1;Password=password;
  Persist Security Info=True;User ID=sa;
  Initial Catalog=CD;Data Source=(local)
```

Observe que a última linha é na verdade uma longa linha de texto que foi dividida em várias outras para que a clareza fosse mantida.

Depois que você tiver sua string de conexão, e que souber que ela está correta porque o botão `Test Connection` executou um teste bem-sucedido com base nessas informações, poderá começar a escrever algum código para seu banco de dados.

Para gerar uma conexão, você precisa criar uma nova instância do objeto `System.Data.OleDb.OleDbConnection` ou de `System.Data.SqlClient.SqlConnection`. Se estiver se conectando com o MSDE ou com o SQL Server, então, a string de conexão precisará ser um pouco alterada antes que possamos usá-la com o objeto `SqlConnection`. A string é utilizada com o `OleDb` e, portanto, com qualquer tipo de banco de dados, o que significa que ela apresenta uma seção `Provider`. A especificação de um provedor não é necessária, e deve ser removida quando houver uma conexão através das classes SQL.

A Listagem 12.1 cria uma nova instância de `OleDbConnection` e fornece a string de conexão no construtor, produzindo um objeto `Connection` que já está configurado com todas as informações que você precisa para se comunicar com seu banco de dados.

LISTAGEM 12.1 Uma String de Conexão

```
1 Module Module1
2     Private Const sConnection As String = "Provider=SQLOLEDB.1;" &_
3                                           "Password=password;" &_
4                                           "Persist Security Info=True;" &_
5                                           "User ID=sa;" &_
6                                           "Initial Catalog=CD;" &_
7                                           "Data Source=(local)"
```


LISTAGEM 12.1 Uma String de Conexão (*continuação*)

```
8
9     Sub Main()
10         Dim objConn As New System.Data.OleDb.OleDbConnection(sConnection)
11     End Sub
12 End Module
```

Nesse momento, você criou o objeto de conexão e o carregou com os detalhes de seu sistema, mas nenhuma comunicação real ocorreu com o servidor do banco de dados. Para estabelecer o vínculo com o banco de dados, é preciso chamar o método `Open` do objeto `OleDbConnection`:

`objConn.Open()`

Essa linha em particular é a primeira que causará uma interação com seu banco de dados, e você deve inseri-la em uma estrutura de tratamento de erros. A Listagem 12.2 mostra um exemplo completo do estabelecimento de uma conexão com um tratamento simples de erros.

LISTAGEM 12.2 Abrindo uma Conexão

```
1 Module Module1
2     Private Const sConnection As String = "Provider=SQLOLEDB.1;" & _
3                                           "Password=password;" & _
4                                           "Persist Security Info=True;" & _
5                                           "User ID=sa;" & _
6                                           "Initial Catalog=CD;" & _
7                                           "Data Source=(local)"
8     Sub Main()
9         Dim objConn As New System.Data.OleDb.OleDbConnection(sConnection)
10        Try
11            objConn.Open()
12        Catch myException As System.Exception
13            Console.WriteLine(myException.Message)
14        End Try
15        Console.ReadLine()
16    End Sub
17 End Module
```

Para obter mais informações sobre o tratamento de erros, veja o Dia 6, “O Que Fazer Quando Programas Bons Apresentam Problemas e para Se Certificar de Que Isso Não Aconteça”. Se você quiser testar o código por sua própria conta, apenas crie um novo aplicativo do console usando o Visual Basic .NET e insira esse código em `Module1.vb`. Lembre-se de substituir o valor de `sConnection` na Listagem 12.2 pela string que criou anteriormente nesta lição. Para ficar ainda mais divertido, tente alterar alguns valores da string de conexão, como a senha, para fazer

com que a tentativa de conexão falhe. Seu tratamento de erros deve produzir uma linha de saída adequada mostrando o que houve de errado em seu programa.

Para essa tarefa específica, abrir uma conexão, as classes `SQLClient` poderão ser usadas com apenas algumas pequenas alterações. Primeiro, remova a seção `Provider` de sua string de conexão e, em seguida, altere sua declaração de `objConn` para que se refira ao objeto `System.Data.SqlClient.SqlConnection` em vez da classe `OleDb`. A Listagem 12.3 mostra o código completo, que produz essencialmente o mesmo resultado.

LISTAGEM 12.3 O Tratamento de Erros É Essencial Quando Se Lida com Bancos de Dados

```
1 Imports System
2 Module Module1
3     Private Const sConnection As String
4         sConnection = "Password=password;" & _
5             "Persist Security Info=True;" & _
6             "User ID=sa;" & _
7             "Initial Catalog=CD;" & _
8             "Data Source=(local)"
9     Sub Main()
10         Dim objConn As New Data.SqlClient.SqlConnection(sConnection)
11         Try
12             objConn.Open()
13         Catch myException As Exception
14             Console.WriteLine(myException.Message)
15         End Try
16         Console.ReadLine()
17     End Sub
18 End Module
```

Estabelecer uma conexão será uma etapa inicial comum, mas sozinha ela não é particularmente útil; você precisará de alguns outros objetos de banco de dados para começar a trabalhar com os dados.

Executando uma Instrução SQL

Agora que você tem uma conexão estabelecida, com certeza desejará ter a capacidade de adicionar, excluir, alterar e recuperar registros. Para conseguir realizar qualquer uma dessas tarefas, você poderá usar um objeto de comando (`SqlCommand` ou `OleDbCommand`) para representar e, em seguida, executar uma instrução SQL. Antes de poder empregar um objeto de comando, precisará da instrução SQL que quer executar e de um objeto de conexão (`SqlConnection` ou `OleDbConnection`). O objeto de conexão não precisa ser aberto quando o objeto de comando estiver sendo criado, mas terá de ser antes da execução efetiva do comando. Se tivermos nossa instrução SQL

e nossa conexão, então, tudo que precisamos é criar uma nova instância do tipo apropriado de objeto de comando. Temos a opção de passar a instrução SQL e o objeto de conexão para o construtor quando criarmos o objeto (veja a Listagem 12.4) ou configurá-los depois do fato (veja a Listagem 12.5).

LISTAGEM 12.4 Executando Consultas no Banco de Dados

```
1 Module Module1
2     Private Const sConnection As String = "Provider=SQLOLEDB.1;" & _
3                                           "Password=;" & _
4                                           "User ID=sa;" & _
5                                           "Initial Catalog=CD;" & _
6                                           "Data Source=(local)"
7     Sub Main()
8         Dim sSQL As String
9         sSQL = "INSERT INTO CATEGORY (CategoryID, Category)" & _
10              "VALUES (7,'Electronic')"
11         Dim objConn As New _
12             System.Data.OleDb.OleDbConnection(sConnection)
13         Dim objCmd As New _
14             System.Data.OleDb.OleDbCommand(sSQL, objConn)
15         Try
16             objConn.Open()
17         Catch myException As System.Exception
18             Console.WriteLine(myException.Message)
19         End Try
20
21         Console.Write("Pressione Return para executar a consulta: {0}", sSQL)
22         Console.ReadLine()
23
24
25
26         If objConn.State = ConnectionState.Open Then
27             Try
28                 objCmd.ExecuteNonQuery()
29             Catch myException As System.Exception
30                 Console.WriteLine(myException.Message)
31                 Console.ReadLine()
32             End Try
33         End If
34
35     End Sub
36 End Module
```

LISTAGEM 12.5 Configurando as Propriedades Individualmente

```
1 Dim objCmd As New OleDbCommand()  
2 objCmd.Connection = objConn  
3 objCmd.CommandText = sSQL
```

A execução da instrução INSERT é efetuada pelo método `ExecuteNonQuery` do objeto de comando (linha 13), mas esse método foi projetado para ser usado apenas com os comandos da SQL que não retornem linhas.

Recuperando Dados

Se você quiser executar uma consulta SQL que retorne linhas de dados, como SELECT, então, terá de usar o método `ExecuteReader`. Esse método existe nas duas variações dos objetos leitores e retorna `OleDbDataReader` ou `SqlDataReader`, o que for mais apropriado. Depois que o objeto leitor tiver sido retornado, poderá ser empregado na execução de um laço pelos dados para recuperar os valores de alguns ou todos os campos e linhas no resultado de sua consulta. O objeto leitor possui muitos métodos e propriedades, mas os membros-chave são `Read()` e os métodos de recuperação de conjuntos de dados `GetString`, `GetDouble`, `GetDateTime` além de outros, todos utilizando um número ordinal para indicar de que campo devem retornar seus dados.

O método `Read` avança o leitor de dados para o próximo registro e retorna um valor booleano indicando se alguma outra linha está disponível. Depois que você estiver em um registro (depois que tiver chamado `Read()`), poderá usar os vários métodos `Get<tipo de dados>` para recuperar os valores de campos individuais. O código mostrado na Listagem 12.6 foi expandido para incluir a execução de uma nova consulta SELECT e de um laço para percorrer e exibir os resultados da consulta no console.

LISTAGEM 12.6 Acessando os Resultados de uma Consulta ao Banco de Dados

```
1 Module Module1  
2     Private Const sConnection As String = "Provider=SQLOLEDB.1;" &  
3                                           "Password=;" &  
4                                           "User ID=sa;" &  
5                                           "Initial Catalog=CD;" &  
6                                           "Data Source=(local)"  
7  
8     Sub Main()  
9         Dim sSQL As String  
10        sSQL = "SELECT Artist.ArtistID, " &  
11              "Artist.ArtistName, Disc.CDTitle " &  
12              "FROM Artist INNER JOIN Disc ON " &  
13              "Artist.ArtistID = Disc.ArtistID;"
```

LISTAGEM 12.6 Acessando os Resultados de uma Consulta ao Banco de Dados
(*continuação*)

```
14 Dim objConn As New _
15     System.Data.OleDb.OleDbConnection(sConnection)
16 Dim objCmd As New _
17     System.Data.OleDb.OleDbCommand(sSQL, objConn)
18 Dim objReader As _
19     System.Data.OleDb.OleDbDataReader
20
21 Try
22     objConn.Open()
23 Catch myException As System.Exception
24     Console.WriteLine(myException.Message)
25 End Try
26
27 Console.Write("Pressione Return para executar SQL: {0}", sSQL)
28 Console.ReadLine()
29
30 If objConn.State = ConnectionState.Open Then
31     Try
32         objReader = objCmd.ExecuteReader()
33         Do While objReader.Read()
34             Console.WriteLine("{0} {1} - {2}", _
35                 objReader.GetInt32(0), _
36                 objReader.GetString(1), _
37                 objReader.GetString(2))
38         Loop
39         Console.ReadLine()
40     Catch myException As System.Exception
41         Console.WriteLine(myException.Message)
42         Console.ReadLine()
43     End Try
44 End If
45 End Sub
46 End Module
```

Os leitores de dados são maneiras rápidas, apenas de leitura ordenada, de recuperar dados, mas também são conjuntos de dados totalmente conectados. Isso significa que quando você criar um leitor de dados e executar um laço em suas linhas, cada linha será recuperada diretamente do banco de dados. Na verdade, enquanto estiver trabalhando com um leitor de dados, sua conexão com o banco de dados estará ocupada e não poderá ser usada para qualquer outro fim. Esse não é um grande problema e faz parte da razão principal pela qual esses leitores são tão rápidos (nada é armazenado na memória do cliente), mas às vezes é preciso obter dados que proporcionem um

pouco mais de controle. Como alternativa aos leitores de dados, um modelo desconectado de trabalhar com os dados também está disponível por meio dos DataSets.

Trabalhando com Data Sets

Na seção anterior, você usou um leitor de dados para recuperar informações em seu banco de dados. Esse foi um tipo de acesso a dados conectado, em que é preciso estar conectado com o banco de dados durante todo o processo. Conseqüentemente, esse estilo de acesso a dados é mais adequado quando se quer recuperar rapidamente um conjunto de informações e não há planos de continuar a usá-las por um tempo mais longo (mais de alguns segundos). Como alternativa, se recuperarmos dados e, em seguida, continuarmos processando-os, alterando, exibindo, filtrando, executando ou ainda manipulando esses dados por muito tempo ou retendo-os como uma forma de cache, então, o tipo de acesso a dados conectado não será a melhor escolha.

Como outra opção, a plataforma .NET fornece um modelo de acesso a dados totalmente desconectado por meio do Data Set (`System.Data.DataSet`). É possível carregar informações nos Data Sets, em geral de um banco de dados, embora possam provir de outras fontes, tornando-as independentes de sua origem. A independência da fonte dos dados é o motivo pelo qual essas informações são consideradas desconectadas e podem ser mantidas pelo tempo necessário em um ambiente totalmente off-line. Se você já usou a versão anterior do ADO, então, isso pode parecer semelhante ao objeto desconectado `RecordSet` dessa tecnologia. É parecido com esse objeto de muitas maneiras, mas possui muitos outros recursos. Um Data Set pode conter várias tabelas com informações, monitorar os relacionamentos entre elas e até manter múltiplas visualizações diferentes de cada tabela de dados. Esses objetos são como bancos de dados completos que usam a memória e ainda são capazes de fornecer uma utilização simples com um conjunto único de resultados.

Nas seções a seguir, você aprenderá a carregar dados em um DataSet e, em seguida, tendo armazenado-os, a navegar por essas informações. Depois de abordados esses aspectos, usaremos alguns dos recursos mais complexos, incluindo tabelas, visualizações e relacionamentos múltiplos. Para concluirmos, examinaremos como retornar ao banco de dados, depois que tivermos editado, excluído e adicionado linhas, as alterações efetuadas em um dataset.

Inserindo Dados em um DataSet

Para carregar as informações de seu banco de dados em um DataSet, você precisará usar outro tipo de objeto, um adaptador de dados. Os objetos adaptadores de dados, `SQLDataAdapter` e `OleDbDataAdapter`, foram projetados para fornecer a união ou vínculo entre seu banco de dados e um objeto DataSet. Esse vínculo funciona em duas etapas, preenchendo o DataSet e, em seguida, retornando as alterações do DataSet à fonte dos dados para atualizar o registro original. Aprenderemos como funciona o estágio de atualização desse processo ainda nesta lição, mas antes que possamos atualizar alguma coisa, temos de carregar os dados. Primeiro, devemos criar um

LISTAGEM 12.7 Preenchendo um DataSet com os Resultados de uma Consulta
(continuação)

```
5                                     "Initial Catalog=CD;" & _
6                                     "Data Source=(local)"
7
8     Sub Main()
9         Dim sSQL As String
10        sSQL = "SELECT Artist.ArtistID, " & _
11              "Artist.ArtistName, Disc.CDTitle " & _
12              "FROM Artist INNER JOIN Disc ON " & _
13              "Artist.ArtistID = Disc.ArtistID;"
14
15        Dim objConn As New _
16        OleDb.OleDbConnection(sConnection)
17        Dim objDataAdapter As New _
18        OleDb.OleDbDataAdapter(sSQL,objConn)
19        Dim objDS As New DataSet("CDs")
20
21        Try
22            objConn.Open()
23        Catch myException As System.Exception
24            Console.WriteLine(myException.Message)
25        End Try
26
27        If objConn.State = ConnectionState.Open Then
28            Try
29                objDataAdapter.Fill(objDS,"Disc")
30                objConn.Close()
31                Console.WriteLine("{0}Rows", _
32                objDS.Tables("Disc").Rows.Count)
33            Catch myException As System.Exception
34                Console.WriteLine(myException.Message)
35            End Try
36            Console.ReadLine()
37        End If
38    End Sub
39 End Module
```

Como em todos os exemplos demonstrados até agora nesta lição, você pode executar esse código por sua própria conta colocando-o em um módulo de um novo aplicativo do console.

Navegando pelos Dados

Depois que você carregar alguns dados em seu DataSet, pode querer examinar essas informações. O objeto com o qual trabalhará na verdade não é o DataSet, porque ele representa potencialmente várias tabelas diferentes, mas sim o objeto DataTable correspondente aos dados carregados nos quais estiver interessado. O código a seguir mostra como poderá obter um DataTable a partir do DataSet que o contém:

```
Dim objTable As DataTable
objTable = objDS.Tables("Disc")
```

Já que o código que inseriu os dados no DataSet forneceu o nome "Disc" para a tabela recém-carregada, você pode usar esse nome para acessar a mesma tabela. O objeto DataTable disponibiliza dois conjuntos que são particularmente úteis: o conjunto Rows, que contém todos os registros da tabela, e o conjunto Columns, que possui um grupo de objetos DataColumn descrevendo cada campo da tabela. O conjunto Rows pode ser empregado de uma entre várias maneiras para executar um laço através de todos os registros da tabela:

Usando um laço For Each

```
Dim objRow As DataRow
For Each objRow In objTable.Rows
    Console.WriteLine(objRow.Item("CDTitle"))
Next
```

Usando um laço For simples

```
Dim i As Integer
Dim objRow As DataRow

For i = 0 To objTable.Rows.Count - 1
    objRow = objTable.Rows(i)
    Console.WriteLine(objRow.Item("CDTitle"))
Next
```

Qualquer um dos métodos produz o mesmo resultado nesse caso. Cada registro da tabela é representado por um objeto DataRow. Esse objeto concederá acesso a seus campos por meio da propriedade Item (na qual você pode fornecer o índice ou o nome do campo que deseja recuperar), e também fornece várias propriedades úteis, como RowState, que retorna o estado de edição atual do registro.

O conjunto Columns contém detalhes dos campos de DataTable, cada um representado como um objeto DataColumn. Por meio desse conjunto, e dos objetos contidos nele, você obterá todas as informações sobre a estrutura de sua tabela. Como no conjunto Rows anterior, é possível executar um laço através dos campos da tabela usando For Each ou For:

```
Dim objColumn As DataColumn
For Each objColumn In objTable.Columns
```



```

Console.WriteLine("Column Name: {0} Data Type: {1}",_
    objColumn.ColumnName,_
    objColumn.DataType.FullName)

```

Next

O código do procedimento `DisplayTable` do exemplo descrito na Listagem 12.8 mostra como você poderia usar as propriedades `Rows` e `Columns` de um `DataTable` para escrever um procedimento genérico a fim de exibir todas as informações sobre a tabela com os cabeçalhos da coluna.

LISTAGEM 12.8 Exibindo o Conteúdo de um `DataTable`

```

1  Sub DisplayTable(ByRef objTable As DataTable)
2      Dim objRow As DataRow
3      Dim objCol As DataColumn
4      Dim i, j As Integer
5
6      For j = 0 To objTable.Columns.Count - 1
7          objCol = objTable.Columns(j)
8          Console.Write("{0}:{1} ",_
9              objCol.ColumnName,_
10             objCol.DataType.Name)
11      Next
12      Console.WriteLine()
13
14      For i = 0 To objTable.Rows.Count - 1
15          objRow = objTable.Rows(i)
16          For j = 0 To objTable.Columns.Count - 1
17              Console.Write("{0} ",objRow.Item(j))
18          Next
19          Console.WriteLine()
20      Next
21  End Sub

```

Você pode testar esse procedimento alterando o exemplo para carregar um `DataSet` que inclua uma chamada a esse novo procedimento, como mostro na Listagem 12.9.

LISTAGEM 12.9 A Função `DisplayTable`

```

1 Private Const sConnection As String = "Provider=SQLOLEDB.1;" &_
2     "Password=;" &_
3     "Persist Security Info=True;" &_
4     "User ID=sa;" &_
5     "Initial Catalog=CD;" &_
6     "Data Source=(local)"
7
8 Sub Main()

```


LISTAGEM 12.9 A Função DisplayTable (*continuação*)

```
9 Dim sSQL As String
10 sSQL = "SELECT Artist.ArtistID, " & _
11 "Artist.ArtistName, Disc.CDTitle " & _
12 "FROM Artist INNER JOIN Disc ON " & _
13 "Artist.ArtistID = Disc.ArtistID;"
14
15 Dim objConn As New _
16 OleDb.OleDbConnection(sConnection)
17 Dim objDataAdapter As New _
18 OleDb.OleDbDataAdapter(sSQL,objConn)
19 Dim objDS As New
20 DataSet("CDs")
21
22 Try
23     objConn.Open()
24 Catch myException As System.Exception
25     Console.WriteLine(myException.Message)
26 End Try
27
28 If objConn.State = ConnectionState.Open Then
29     Try
30         objDataAdapter.Fill(objDS,"Disc")
31         objConn.Close()
32         Dim objTable As DataTable
33         objTable = objDS.Tables("Disc")
34
35         DisplayTable(objTable)
36
37     Catch myException As System.Exception
38         Console.WriteLine(myException.Message)
39     End Try
40     Console.ReadLine()
41 End If
42 End Sub
```

O objeto DataRow é importante; além de ser o meio pelo qual você acessa os valores do campo (como foi feito na Listagem 12.8), ele também é usado quando se edita, exclui e adiciona registros a um objeto DataTable. Na seção a seguir, aprenderemos a alterar o conteúdo de seu DataSet usando tanto o objeto DataTable quanto DataRow.

Editando Dados (Adicionar, Editar e Excluir)

Nesta seção, você escreverá um código para alterar seu banco de dados por meio do objeto DataSet. No entanto, o banco de dados propriamente dito não tem conhecimento algum desses objetos DataSets e acabará sendo alterado pelas instruções SQL. As alterações que forem feitas no DataSet serão convertidas pelo DataAdapter em instruções SQL que terão de ser executadas visando ao banco de dados. É claro que seria possível apenas executar cada instrução UPDATE, INSERT e DELETE diretamente, mas o modelo DataSet possui os benefícios de ser desconectado (evitando o uso de recursos do servidor enquanto aguarda a interação com o usuário) e simples (usando métodos do objeto, como DELETE, em vez de instruções SQL personalizadas).

Usando os três tipos de alteração de dados – adicionar, editar e excluir – um de cada vez, você verá como pode manipular facilmente seus dados por meio dos métodos dos objetos DataTable e DataRow.

Adicionando Registros

Depois de criar uma conexão, um adaptador de dados e carregar alguns dados em seu DataSet, você poderá acessar diretamente um objeto DataTable:

```
objDataAdapter.Fill(objDS, "Disc")
Dim objTable As DataTable
objTable = objDS.Tables("Disc")
```

Depois que tiver esse objeto DataTable, poderá acessar seu conteúdo por meio do conjunto Rows, que retornará um objeto DataRowCollection:

```
Dim drRows As DataRowCollection
drRows = objTable.Rows
```

Esse objeto representa todos os registros da tabela como um conjunto de objetos DataRow. O próprio conjunto fornece um método, Add, para criar novos registros. Esse método pode ser chamado de uma entre duas maneiras: com um array de valores para o campo do novo registro ou com um único objeto DataRow representando as informações a adicionar. O segundo método, que é chamado com um DataRow, requer uma maneira de obter um novo objeto de linha que possua o mesmo esquema (os mesmos campos, com os mesmos tipos de dados e tamanhos) do resto das linhas da tabela, e o próprio objeto DataTable fornece esse recurso por meio do método NewRow. Você verá um exemplo dos dois estilos nas Listagens 12.10 e 12.11. A Listagem 12.10 mostra o uso de um array, que nos forçará a conhecer a ordem dos campos e resultados em um código que é muito menos claro que o segundo método. A Listagem 12.11 mostra o segundo método – empregando um objeto DataRow.

LISTAGEM 12.10 Usando um Array (listagem parcial do código)

```
1 Dim drRows As DataRowCollection
2 Dim objNewRow As DataRow
3 drRows = objTable.Rows
4
5 'Neste caso, temos 3 colunas
6 'ArtistID
7 'ArtistName
8 'CDTitle
9
10 'Usando o método do array produziremos
11 Dim arrFields(3) As Object
12 arrFields(0) = 10
13 arrFields(1) = "Sting"
14 arrFields(2) = "Dream of Blue Turtles"
15 objNewRow = drRows.Add(arrFields)
```

LISTAGEM 12.11 Fornecendo um Objeto DataRow como Parâmetro para o Método Add

```
1 Dim drRows As DataRowCollection
2 Dim objNewRow As DataRow
3 drRows = objTable.Rows
4
5 'Neste caso, temos 3 colunas
6 'ArtistID
7 'ArtistName
8 'CDTitle
9
10 objNewRow = objTable.NewRow()
11 objNewRow("ArtistID") = 3
12 objNewRow("ArtistName") = "George Orwell"
13 objNewRow("CDTitle") = "Party like it's 1984"
14 drRows.Add(objNewRow)
```

Nos dois casos, os dados foram adicionados à tabela, mas isso não significa que serão retornados ao banco de dados. As alterações efetuadas no banco de dados do servidor depois do momento em que os dados foram carregados nesse DataSet poderiam impedir que os registros adicionados fossem válidos (você poderia especificar um artista que tivesse sido excluído da tabela Artist, por exemplo), mas você não veria esse tipo de erro até o estágio da atualização. Já que está trabalhando em um estado desconectado, ou off-line, quando usa um DataSet, qualquer problema relacionado ao servidor não será descoberto até que haja a tentativa de atualizá-lo.

Editando Registros

Editar registros é semelhante a adicionar; trabalha-se através dos objetos `DataRow` que representam os registros da tabela. No caso da edição, no entanto, queremos trabalhar com um objeto `DataRow` existente, em vez de criar um novo como fizemos na seção anterior, “Adicionando Registros”. Como nos exemplos anteriores, precisamos primeiro criar uma conexão, um comando, um adaptador de dados, e então carregar os dados em um objeto `DataSet` para criar um objeto `DataTable`. Depois que tivermos esse `DataTable`, teremos acesso a seu objeto `DataRowCollection`, e estaremos prontos para começar a editar os registros.

```
objDataAdapter.Fill(objDS, "Disc")
Dim objTable As DataTable
objTable = objDS.Tables("Disc")
Dim drRows As DataRowCollection
Dim objRow As DataRow
drRows = objTable.Rows
objRow = drRows(5)
```

Cada objeto `DataRow` possui vários métodos que afetam diretamente a edição: `BeginEdit`, `EndEdit` e `CancelEdit`. `BeginEdit` e `EndEdit` colocam o objeto `DataRow` dentro ou fora do modo de edição, que é um estado especial no qual a linha contém as informações sobre a edição em progresso e, portanto, eventos de alteração não serão acionados para cada modificação de um campo. Você aprenderá mais sobre os eventos de alteração ainda nesta lição, na seção “Vinculação de Dados”. `BeginEdit` é opcional, mas indica sua intenção de maneira clara. É possível acessar e alterar os valores por meio da propriedade `Item` usando o nome ou a posição do campo:

```
objRow.BeginEdit()
objRow("au_lname") = "Exciting"
objRow.EndEdit()
```

Quando `EndEdit` for chamado, se `BeginEdit` tiver sido usado, as alterações da linha serão confirmadas dentro do objeto `DataTable`, momento em que qualquer erro que tenha ocorrido durante a edição será lançado pelo ADO.NET. Sem a utilização de `BeginEdit` e `EndEdit`, todos os erros ocorrerão no instante em que cada campo for editado.

Você aprenderá a usar um `DataSet` contendo campos alterados, registros novos e até excluídos, e atualizar o banco de dados de origem posteriormente nesta lição.

Excluindo Registros

A exclusão de um registro em uma tabela de dados é efetuada pelo método `Delete` do objeto `DataRow` em questão. Isso é simples de fazer, mas há um problema com o qual se preocupar. O método `Delete` marca a linha como excluída, mas, na verdade, não remove o objeto `DataRow` de `DataRowCollection`. Isso é crítico porque permite que o objeto não seja excluído e fornece as informações necessárias para atualizar o banco de dados de origem excluindo esse registro quando ele estiver novamente na tabela original. Há outros métodos, `Remove` e `RemoveAt`, da classe `Data-`

RowCollection, que realmente removem o objeto DataRow específico do conjunto. Usar Remove ou RemoveAt será adequado quando você empregar um objeto DataTable sem um banco de dados de back-end, mas se a intenção for usar todas as alterações feitas no objeto DataSet para atualizar a(s) tabela(s) de origem, será preciso utilizar o método Delete.

A Listagem 12.12 fornece o exemplo de um laço em uma tabela, que localiza certas linhas específicas e, em seguida, as exclui. Nenhuma dessas exclusões afetará os dados da fonte, o SQL Server, por exemplo, até que você atualize o servidor.

LISTAGEM 12.12 O Método Delete Marca uma Linha como Excluída

```

1 Module Module1
2     Private Const _
3         sConnection As String = _
4             "Provider=SQLOLEDB.1;" & _
5             "Password=;" & _
6             "Persist Security Info=True;" & _
7             "User ID=sa;" & _
8             "Initial Catalog=CD;" & _
9             "Data Source=(local)"
10
11     Sub Main()
12         Dim sSQL As String
13         sSQL = "SELECT ArtistID, DiscID, CDTitle From Disc"
14
15         Dim objConn _
16             As New OleDb.OleDbConnection(sConnection)
17         Dim objDataAdapter _
18             As New OleDb.OleDbDataAdapter(sSQL, objConn)
19         Dim objDS _
20             As New DataSet("CDs")
21
22         Try
23             objConn.Open()
24         Catch myException As System.Exception
25             Console.WriteLine(myException.Message)
26         End Try
27
28         If objConn.State = ConnectionState.Open Then
29             Try
30                 objDataAdapter.Fill(objDS, "Disc")
31                 objConn.Close()
32                 Dim objTable As DataTable
33                 objTable = objDS.Tables("Disc")
34                 Dim drRows As DataRowCollection
35                 Dim objCurrentRow As DataRow

```


LISTAGEM 12.12 O Método Delete Marca uma Linha como Excluída (*continuação*)

```

36         drRows = objTable.Rows
37
38         DisplayTable(objTable)
39
40         Console.WriteLine("Se pretende excluir, pressione return para iniciar!")
41         Console.ReadLine()
42
43         For Each objCurrentRow In drRows
44             If CType(objCurrentRow("ArtistID"), Integer) = 3 Then
45                 objCurrentRow.Delete()
46             End If
47         Next
48
49         Console.WriteLine("Após a exclusão, pressione return para visualizar")
50         Console.ReadLine()
51
52         DisplayTable(objTable)
53
54         Catch myException As System.Exception
55             Console.WriteLine(myException.Message)
56         End Try
57         Console.ReadLine()
58     End If
59
60 End Sub
61 Sub DisplayTable(ByRef objTable As DataTable)
62     Dim objRow As DataRow
63     Dim objCol As DataColumn
64     Dim i, j As Integer
65
66     For j = 0 To objTable.Columns.Count - 1
67         objCol = objTable.Columns(j)
68         Console.WriteLine("{0}:{1} ", _
69             objCol.ColumnName, _
70             objCol.DataType.Name)
71     Next
72     Console.WriteLine()
73     For i = 0 To objTable.Rows.Count - 1
74         objRow = objTable.Rows(i)
75         Select Case objRow.RowState
76             Case DataRowState.Deleted
77                 Console.WriteLine("[Deleted]      ")
78             Case DataRowState.Modified
79                 Console.WriteLine("[Modified]    ")
80             Case DataRowState.Added

```


LISTAGEM 12.12 O Método Delete Marca uma Linha como Excluída (*continuação*)

```
81         Console.WriteLine("[New]      ")
82         Case DataRowState.Unchanged
83             Console.WriteLine("[Unchanged]  ")
84     End Select
85     For j = 0 To objTable.Columns.Count - 1
86         If objRow.RowState <> DataRowState.Deleted Then
87             Console.WriteLine("{0} ", _
88                 objRow.Item(j))
89         Else
90             Console.WriteLine("{0} ", _
91                 objRow.Item(j, DataRowVersion.Original))
92         End If
93     Next
94     Console.WriteLine()
95 Next
96 End Sub
97 End Module
```

Observe que na linha 90, dentro da rotina `DisplayTable`, uma sintaxe diferente foi usada para acessar o valor de um campo. Pela inclusão do segundo parâmetro, `DataRowVersion.Original`, esse código especifica que o valor do campo após a atualização mais recente, ou no momento em que for carregado, deve ser exibido em vez do valor do campo atual (o padrão). Nesse caso, isso é particularmente importante porque uma linha excluída não possui valores válidos para o campo atual, e um erro será gerado se você tentar acessá-los.

Atualizando o Banco de Dados

Depois de adicionar, editar ou excluir os registros, você terá de atualizar a fonte original dos dados (o banco de dados) antes que essas alterações tenham efeito. O procedimento que usa um `DataSet` alterado para atualizar o banco de dados de origem é o mesmo, independentemente das modificações feitas, e envolve o uso da classe de adaptadores de dados.

Especificando os Comandos de Atualização, Inserção e Exclusão

Quando você criou um adaptador de dados para carregar informações em seu `DataSet`, forneceu um objeto de comando ou uma instrução SQL que o adaptador, em seguida, usou para recuperar os dados de seu banco de dados. Esse comando na verdade é um entre os quatro aos quais o adaptador de dados pode dar suporte, tendo cada um sido projetado para manipular uma tarefa diferente na transmissão de dados entre o banco de dados e os `DataSets`. Só um dos comandos, aquele que já foi usado, manipula a extração de dados do banco de dados e é chamado de `Select-`

Command. Os outros três (UpdateCommand, DeleteCommand e InsertCommand) manipulam edições, exclusões e inserções no banco de dados. Cada um desses comandos deve conter um valor apropriado, seja uma instrução SQL ou uma chamada a um procedimento armazenado no servidor, antes que a ação que lhe compete possa ser executada. Cada propriedade dessas representa um objeto Command do tipo apropriado (OleDbCommand ou SqlCommand), que será criado ou inicializado antes de ser atribuído à propriedade.

NOVO TERMO

Para que você mesmo crie esses comandos, precisará usar um tipo de consulta que não foi abordada até agora nesta lição, um *comando com parâmetros*. Um comando dessa espécie deixa espaços reservados especiais em sua instrução SQL – parâmetros – que são preenchidos com valores específicos antes que o comando seja executado. Os pontos de interrogação são usados na instrução SQL para representar os parâmetros, como mostrado na Listagem 12.13:

LISTAGEM 12.13 Os Adaptadores de Dados Precisam de Comandos com Espaços Reservados

```
1 Dim objDeleteCommand As New OleDb.OleDbCommand()  
2 Dim sDeleteSQL As String  
3  
4 sDeleteSQL = "DELETE FROM Artist WHERE ArtistID = ?"  
5  
6 objDeleteCommand.Connection = objConn  
7 objDeleteCommand.CommandText = sDeleteSQL
```

Os detalhes devem ser fornecidos para cada espaço reservado pela inclusão de um objeto de parâmetro (OleDbParameter ou SqlParameter, dependendo do ambiente em que você trabalha) no conjunto de parâmetros do comando. Cada objeto de parâmetro é vinculado a um campo especificado do DataSet por meio da propriedade SourceColumn, e também pode ser associado a uma versão específica do campo por meio da propriedade SourceVersion. A Listagem 12.14 mostra o código necessário para adicionar um parâmetro e configurar algumas das propriedades disponíveis.

LISTAGEM 12.14 Criando Parâmetros em um Objeto de Comando

```
1 Dim objParam As OleDb.OleDbParameter  
2  
3 objParam = objDeleteCommand.Parameters.Add(_  
4     "@ArtistID", OleDb.OleDbType.Integer)  
5  
6 objParam.SourceColumn = "ArtistID"  
7 objParam.SourceVersion = DataRowVersion.Original
```

Nesse exemplo, o único ponto de interrogação fornecido para a SQL foi associado a esse novo objeto de parâmetro. Os espaços reservados foram associados aos parâmetros com base na posi-

ção: o primeiro espaço reservado foi vinculado ao primeiro parâmetro. Depois que o objeto de parâmetro foi criado, o campo e sua versão correspondentes foram especificados nas linhas 5 e 6. Já que estamos lidando com linhas excluídas aqui, exatamente como na Listagem 12.12 anterior, é importante especificar `SourceVersion`. A versão-padrão é a do valor atual (`DataRowVersion.Current`), que não está disponível para uma linha excluída. A última etapa que você terá de executar antes do comando ser usado pelo adaptador será atribuí-lo à propriedade `DeleteCommand`:

```
objDataAdapter.DeleteCommand = objDeleteCommand
```

A Listagem 12.15 mostra um exemplo completo e muito extenso que cria uma conexão com o banco de dados e um adaptador e, em seguida, configura os comandos individuais para `UpdateCommand`, `DeleteCommand` e `InsertCommand`. Por fim, para testar esses comandos, um dataset é criado, carregado com dados e alterado e, depois, a fonte de dados é atualizada pelo método `Update` do adaptador.

LISTAGEM 12.15 As Alterações Não Têm Efeito até Que o Método `Update` Seja Chamado

```

1 Module Module1
2     Private Const _
3     sConnection As String = _
4         "Provider=SQLOLEDB.1;" & _
5         "Password=;" & _
6         "Persist Security Info=True;" & _
7         "User ID=sa;" & _
8         "Initial Catalog=CD;" & _
9         "Data Source=(local)"
10
11 Sub Main()
12     Dim sSQL As String
13     sSQL = "SELECT ArtistID, ArtistName From Artist"
14     Dim objConn _
15         As New OleDb.OleDbConnection(sConnection)
16     Dim objDataAdapter _
17         As New OleDb.OleDbDataAdapter(sSQL, objConn)
18     Dim objDS _
19         As New DataSet("Artists")
20     Dim objDeleteCommand _
21         As New OleDb.OleDbCommand()
22     Dim objUpdateCommand _
23         As New OleDb.OleDbCommand()
24     Dim objInsertCommand _
25         As New OleDb.OleDbCommand()
26     Dim sDeleteSQL As String
27     Dim sUpdateSQL As String
28     Dim sInsertSQL As String

```


LISTAGEM 12.15 As Alterações Não Têm Efeito até Que o Método Update Seja Chamado (*continuação*)

```
29
30     sDeleteSQL = "DELETE FROM Artist WHERE ArtistID = ?"
31
32     objDeleteCommand.Connection = objConn
33     objDeleteCommand.CommandText = sDeleteSQL
34
35     Dim objParam As OleDb.OleDbParameter
36     objParam = objDeleteCommand.Parameters.Add(_
37         "@ArtistID", OleDb.OleDbType.Integer)
38     objParam.SourceColumn = "ArtistID"
39     objParam.SourceVersion = DataRowVersion.Original
40
41     objDataAdapter.DeleteCommand = objDeleteCommand
42
43     sUpdateSQL = "Update Artist SET ArtistName = ? " & _
44         "WHERE ArtistID = ?"
45
46     objUpdateCommand.Connection = objConn
47     objUpdateCommand.CommandText = sUpdateSQL
48
49     objParam = objUpdateCommand.Parameters.Add(_
50         "@ArtistName", OleDb.OleDbType.Char)
51     objParam.SourceColumn = "ArtistName"
52     objParam.SourceVersion = DataRowVersion.Current
53
54     objParam = objUpdateCommand.Parameters.Add(_
55         "@ArtistID", OleDb.OleDbType.Integer)
56     objParam.SourceColumn = "ArtistID"
57     objParam.SourceVersion = DataRowVersion.Original
58
59     objDataAdapter.UpdateCommand = objUpdateCommand
60
61     sInsertSQL = "INSERT INTO Artist " & _
62         "(ArtistID, ArtistName) " & _
63         "VALUES (?, ?) "
64
65     objInsertCommand.Connection = objConn
66     objInsertCommand.CommandText = sInsertSQL
67
68     objParam = objInsertCommand.Parameters.Add(_
69         "@ArtistID", OleDb.OleDbType.Integer)
70     objParam.SourceColumn = "ArtistID"
71     objParam.SourceVersion = DataRowVersion.Current
```


LISTAGEM 12.15 As Alterações Não Têm Efeito até Que o Método Update Seja Chamado (*continuação*)

```
72
73     objParam = objInsertCommand.Parameters.Add(_
74         "@ArtistName", OleDb.OleDbType.Char)
75     objParam.SourceColumn = "ArtistName"
76     objParam.SourceVersion = DataRowVersion.Current
77
78     objDataAdapter.InsertCommand = objInsertCommand
79
80     Try
81         objConn.Open()
82     Catch myException As System.Exception
83         Console.WriteLine(myException.Message)
84     End Try
85
86     Try
87         Dim sNewArtistSQL As String
88         sNewArtistSQL = "INSERT INTO Artist " & _
89             "(ArtistID, ArtistName, " & _
90             "ArtistFirstName, ArtistLastName)" & _
91             "VALUES (11, 'Weird Al Yankovich'," & _
92             " 'Al', 'Yankovich')"
```

```
93
94         Dim objNewArtistCmd _
95             As New OleDb.OleDbCommand(sNewArtistSQL, objConn)
96         objNewArtistCmd.ExecuteNonQuery()
97     Catch e As Exception
98         'Pode causar um erro porque
99         'ArtistID #11 já existe
100        'Mas se ocorrer, será correto.
101        Console.WriteLine(e.Message)
102    End Try
103
104    If objConn.State = ConnectionState.Open Then
105        Try
106            objDataAdapter.MissingSchemaAction _
107                =MissingSchemaAction.AddWithKey
108            objDataAdapter.Fill(objDS, "Artist")
109            objConn.Close()
110            Dim objTable As DataTable
111            objTable = objDS.Tables("Artist")
112            Dim drRows As DataRowCollection
113            Dim objCurrentRow As DataRow
114            drRows = objTable.Rows
```


LISTAGEM 12.15 As Alterações Não Têm Efeito até Que o Método Update Seja Chamado (*continuação*)

```
115
116         DisplayTable(objTable)
117
118         Console.WriteLine("Se pretende excluir," &_
119             " pressione return para iniciar!")
120         Console.ReadLine()
121
122         'Exclua a linha com chave primária = 11
123         drRows.Find(11).Delete()
124
125         Console.WriteLine("Após a exclusão," &_
126             " pressione return para visualizar os resultados")
127         Console.ReadLine()
128
129         DisplayTable(objTable)
130
131         Console.WriteLine("Se pretende inserir," &_
132             " pressione return para iniciar!")
133         Console.ReadLine()
134
135         Dim drRow As Data.DataRow
136         drRow = objTable.NewRow
137         drRow("ArtistID") = 40
138         drRow("ArtistName") = "Kent Sharkey"
139         objTable.Rows.Add(drRow)
140
141         Console.WriteLine("Depois de inserir," &_
142             " pressione return para visualizar!")
143         Console.ReadLine()
144         DisplayTable(objTable)
145
146
147         Console.WriteLine("Se pretende atualizar," &_
148             " pressione return para iniciar!")
149         Console.ReadLine()
150
151         'Acesse a linha com chave primária igual a 7 (Lisa Loeb)
152         drRow = drRows.Find(7)
153
154         drRow.BeginEdit()
155         drRow("ArtistName") = "John Doe"
156         drRow.EndEdit()
157
```


LISTAGEM 12.15 As Alterações Não Têm Efeito até Que o Método Update Seja Chamado (*continuação*)

```
158         Console.WriteLine("Depois de atualizar," & _
159             " pressione return para visualizar os resultados")
160         Console.ReadLine()
161         DisplayTable(objTable)
162
163         objConn.Open()
164         objDataAdapter.Update(objDS, "Artist")
165
166         Catch myException As System.Exception
167             Console.WriteLine(myException.Message)
168         End Try
169         Console.WriteLine("Pressione Return para finalizar.")
170         Console.ReadLine()
171     End If
172 End Sub
173 Sub DisplayTable(ByRef objTable As DataTable)
174     Dim objRow As DataRow
175     Dim objCol As DataColumn
176     Dim i, j As Integer
177
178     For j = 0 To objTable.Columns.Count - 1
179         objCol = objTable.Columns(j)
180         Console.WriteLine("{0}:{1} ", _
181             objCol.ColumnName, _
182             objCol.DataType.Name)
183     Next
184     Console.WriteLine()
185     For i = 0 To objTable.Rows.Count - 1
186         objRow = objTable.Rows(i)
187         Select Case objRow.RowState
188             Case DataRowState.Deleted
189                 Console.WriteLine("[Deleted ]      ")
190             Case DataRowState.Modified
191                 Console.WriteLine("[Modified ]    ")
192             Case DataRowState.New
193                 Console.WriteLine("[New ]        ")
194             Case DataRowState.Unchanged
195                 Console.WriteLine("[Unchanged ]  ")
196         End Select
197         For j = 0 To objTable.Columns.Count - 1
198             If objRow.RowState = DataRowState.Deleted Then
199                 Console.WriteLine("{0} ", _
200                     objRow.Item(j))
```


LISTAGEM 12.15 As Alterações Não Têm Efeito até Que o Método Update Seja Chamado (*continuação*)

```
201         Else
202             Console.Write("{0} ",_
203                 objRow.Item(j, DataRowVersion.Original))
204         End If
205     Next
206     Console.WriteLine()
207 Next
208 End Sub
209 End Module
```

ANÁLISE

O código da Listagem 12.15 está disponível para download no site deste livro na Web, portanto, não é necessário digitá-lo a menos que você queira mesmo fazê-lo. A primeira seção do código está bem dentro do padrão utilizado em todos os outros escritos nesta lição. A string de conexão foi criada, devendo ser substituída pelos valores apropriados para seu sistema, e usada para instanciar um novo objeto `OleDbConnection` (linha 14). Um objeto `OleDbDataAdapter` e um `DataSet` também foram gerados e inicializados para tornar possível se conectar com o banco de dados e retornar os resultados da instrução `SELECT` para seu programa. Como parte do código de inicialização, três novos objetos `OleDbCommand` também foram criados e serão empregados na representação dos comandos `UPDATE`, `DELETE` e `INSERT` para `OleDbDataAdapter`. No entanto, esses objetos de comando não foram realmente gerados nesse ponto; essa etapa é manipulada conforme cada comando vai sendo configurado posteriormente.

As linhas 30 a 41 criam e configuram o objeto de comando `Delete`, adicionando o único objeto de parâmetro para `ArtistID` (a chave primária). Observe como a linha 39 especifica que `SourceVersion` seja `DataRowVersion.Original` porque os dados atuais não são válidos para uma linha excluída. As linhas 43 a 59 configuram o comando `Update` e, em seguida, as linhas 61 a 78 manipulam o comando `Insert`. É bom ressaltar que para `Update` e `Insert`, muitos parâmetros são necessários, devido ao fato de que todos os campos da tabela têm de ser enviados nesses comandos.

Voltando ao código que é comum a todos os exemplos desta lição, as linhas 80 a 84 configuram a conexão com o banco de dados. A seguir, uma única linha de dados é inserida no banco de dados pelo método `ExecuteNonQuery()` de um objeto de comando. Essa linha é adicionada de modo que possa ser excluída posteriormente, porque a tentativa de remover qualquer outro registro de `Artist` falharia devido à existência de registros-filhos (CDs desse artista na tabela `Disc`). Em vez de usar o dataset e o adaptador de dados no modo desconectado, essa inserção é feita de imediato por uma operação direta.

A linha 106, que ocorre depois que a conexão é aberta e que o único registro novo tiver sido inserido, informa ao adaptador de dados como manipular a configuração de uma tabela quando não existe nenhum esquema (as informações sobre o tipo de dados e o layout referente aos dados). Essa configuração específica, `AddWithKey`, comunica ao adaptador para criar automaticamente o

esquema básico (informações sobre o campo) e incluir também informações sobre a chave primária. A seguir, na linha 108, o DataSet é carregado com os dados pelo método Fill e com um parâmetro adicional para especificar o nome da tabela recém-criada, 'Artist'.

Apenas para se certificar de que está claro que esse é um processo desconectado, a linha 109 encerra a conexão com o banco de dados antes que a verdadeira manipulação de dados ocorra. Depois desse ponto, o código exclui, altera e adiciona registros ao objeto DataTable para fornecer um teste de todos os três tipos de edições que podem ocorrer em um DataTable. Para encontrar a linha correta a excluir (linha 123) e editar (linha 152), o método Find do objeto DataRowCollection é usado. Esse método retorna um objeto DataRow dado um valor de chave primária válido para pesquisa, um processo que requer que o objeto DataTable tenha informações sobre a chave primária (leia o parágrafo anterior para obter uma explicação de como a linha 106 fornece essa informação).

Para concluir, depois de todas as edições, a linha 164 usa o método Update de OleDbDataAdapter para aplicar todas as alterações de dados ao DataSet. Esse método usa um DataSet como parâmetro, junto a uma string indicando que tabela dentro do DataSet deve ser utilizada para atualizar o banco de dados. Se o nome da tabela não for fornecido, o adaptador tentará atualizar o banco de dados com todas as tabelas disponíveis.

O resultado desse código deve ser uma sequência de instruções SQL executadas em um banco de dados, cada uma correspondendo a um dos comandos do adaptador. Se você estiver usando o SQL Server e quiser ver o que acontece, poderá empregar o Profiler para observar todos os comandos sendo processados no servidor enquanto esse código estiver em execução. Fornecer detalhes sobre como utilizar o Profiler está além do escopo deste livro, mas o resultado deve ser semelhante ao mostrado a seguir:

```
INSERT INTO Artist (ArtistID, ArtistName, ArtistFirstName, ArtistLastName)
VALUES (11, 'Weird Al Yankovich', 'Al', 'Yankovich')
go
SELECT ArtistID, ArtistName From Artist
go
exec sp_executesql
N'Update Artist SET ArtistName = @P1
WHERE ArtistID = @P2', N'@P1 char(8),@P2 int', 'John Doe', 7
go
exec sp_executesql
N'DELETE FROM Artist WHERE ArtistID = @P1', N'@P1 int', 11
go
exec sp_executesql
N'INSERT INTO Artist (ArtistID, ArtistName) VALUES (@P1,@P2)',
N'@P1 int,@P2 char(12)', 40, 'Kent Sharkey'
go
```

Nesse exemplo, só um registro foi inserido, alterado e excluído, mas o método Update simplesmente chamaria os comandos apropriados para cada registro alterado.

Usando Comandos Gerados Automaticamente

Como alternativa a você mesmo especificar os comandos, é possível deixar que o ADO.NET crie automaticamente os comandos necessários, o que pode ser mais simples, embora haja algumas etapas a seguir para se alcançar esse objetivo. O primeiro requisito para os comandos gerados de modo automático é que seu DataSet deve conter pelo menos informações sobre a chave primária das tabelas relevantes. Essas informações podem ser fornecidas de uma entre várias maneiras, já que todas elas produzirão o efeito desejado.

Primeiro, você pode especificá-las manualmente usando a propriedade `PrimaryKey` de um objeto `DataTable`. Essa propriedade é configurada com o campo apropriado, que pode ser obtido pelo conjunto `Columns` de `DataTable`:

```
Dim PK(0) As DataColumn
PK(0) = objTable.Columns("ArtistID")
objTable.PrimaryKey = PK
```

Você também pode deixar que o adaptador de dados obtenha as informações apropriadas na fonte dos dados, incluindo a chave primária e muitos outros detalhes sobre a tabela, por meio do método `FillSchema`. Esse método cria o objeto `DataTable` antes de carregá-lo e preenche a tabela com sua lista de colunas, informações sobre a chave primária e qualquer outra restrição que possa descobrir na fonte de dados:

```
objDataAdapter.FillSchema(objDS, SchemaType.Mapped, "Artist")
```

Para concluir, você pode fazer com que o adaptador crie as informações necessárias para o esquema durante o carregamento dos dados, configurando o valor da propriedade `MissingSchemaAction` antes de chamar `Fill`.

Essa propriedade possui algumas configurações interessantes, mas `AddWithKey` é necessária para esse fim. `Add` (o valor padrão) cria automaticamente as informações sobre a coluna (nomes e tipos de dados) se esses detalhes já não existirem na tabela, enquanto `AddWithKey` adiciona informações sobre essa mesma coluna e também configura a propriedade `PrimaryKey` a cada tabela que preencher.

```
objDataAdapter.MissingSchemaAction _
    = MissingSchemaAction.AddWithKey
objDataAdapter.Fill(objDS, "Artist")
```

Independentemente do método que você usar, depois que o objeto `DataTable`, ao ser atualizado, tiver informações sobre a chave primária, os comandos `Update`, `Insert` e `Delete` poderão ser gerados de modo automático. Primeiro, uma instância da classe apropriada do criador do comando (`OleDbCommandBuilder` ou `SqlCommandBuilder`) deve ser gerada e inicializada e, em seguida, quando o método `Update` do adaptador for chamado, se algum comando necessário estiver faltando, ele será criado. A Listagem 12.16 demonstra esses conceitos carregando alguns dados simples em um DataSet, fazendo algumas alterações e chamando `Update`.

LISTAGEM 12.16 O Método Update Executa os Comandos Correspondentes no Banco de Dados de Origem

```
1 Module Module1
2     Private Const _
3     sConnection As String = _
4         "Provider=SQLOLEDB.1;" & _
5         "Password=;" & _
6         "Persist Security Info=True;" & _
7         "User ID=sa;" & _
8         "Initial Catalog=CD;" & _
9         "Data Source=(local)"
10
11     Sub Main()
12         Dim sSQL As String
13         sSQL = "SELECT ArtistID, ArtistName From Artist"
14         Dim objConn _
15             As New OleDb.OleDbConnection(sConnection)
16         Dim objDataAdapter _
17             As New OleDb.OleDbDataAdapter(sSQL, objConn)
18         Dim objCommandBuilder _
19             As New OleDb.OleDbCommandBuilder(objDataAdapter)
20         Dim objDS _
21             As New DataSet("Artists")
22
23         Try
24             objConn.Open()
25         Catch myException As System.Exception
26             Console.WriteLine(myException.Message)
27         End Try
28
29         If objConn.State = ConnectionState.Open Then
30             Try
31                 objDataAdapter.MissingSchemaAction _
32                     = MissingSchemaAction.AddWithKey
33                 objDataAdapter.Fill(objDS, "Artist")
34
35                 objConn.Close()
36                 Dim objTable As DataTable
37                 objTable = objDS.Tables("Artist")
38                 Dim drRows As DataRowCollection
39                 Dim objCurrentRow As DataRow
40                 drRows = objTable.Rows
41
42                 DisplayTable(objTable)
43     
```


LISTAGEM 12.16 O Método Update Executa os Comandos Correspondentes no Banco de Dados de Origem (*continuação*)

```

44      Console.Write("Se pretende editar," & _
45          " pressione return para iniciar!")
46      Console.ReadLine()
47
48      'Exclua a linha com chave primária = 11
49      drRows.Find(7)("ArtistName") = "Kent Sharkey"
50
51      Console.Write("Depois de editar," & _
52          " pressione return para visualizar os resultados")
53      Console.ReadLine()
54
55      DisplayTable(objTable)
56
57      objConn.Open()
58      objDataAdapter.Update(objDS, "Artist")
59
60      Catch myException As System.Exception
61          Console.WriteLine(myException.Message)
62      End Try
63      Console.Write("Pressione Return para finalizar.")
64      Console.ReadLine()
65  End If
66 End Sub
67 Sub DisplayTable(ByRef objTable As DataTable)
68     Dim objRow As DataRow
69     Dim objCol As DataColumn
70     Dim i, j As Integer
71
72     For j = 0 To objTable.Columns.Count - 1
73         objCol = objTable.Columns(j)
74         Console.Write("{0}:{1} ", _
75             objCol.ColumnName, _
76             objCol.DataType.Name)
77     Next
78     Console.WriteLine()
79     For i = 0 To objTable.Rows.Count - 1
80         objRow = objTable.Rows(i)
81         Select Case objRow.RowState
82             Case DataRowState.Deleted
83                 Console.Write("[Deleted ]      ")
84             Case DataRowState.Modified
85                 Console.Write("[Modified ]    ")
86             Case DataRowState.Added

```

LISTAGEM 12.16 O Método Update Executa os Comandos Correspondentes no Banco de Dados de Origem (*continuação*)

```
87         Console.Write("[New ]           ")
88         Case DataRowState.Unchanged
89             Console.Write("[Unchanged ]   ")
90     End Select
91     For j = 0 To objTable.Columns.Count - 1
92         If objRow.RowState <> DataRowState.Deleted Then
93             Console.Write("{0} ", _
94                 objRow.Item(j))
95         Else
96             Console.Write("{0} ", _
97                 objRow.Item(j, DataRowVersion.Original))
98         End If
99     Next
100     Console.WriteLine()
101 Next
102 End Sub
103 End Module
```

Como em todos os exemplos desta lição, esse código poderia ser modificado para trabalhar com as classes `SQLClient` precisando apenas de algumas alterações menores. A string de conexão teria de ser alterada pela remoção da seção `Provider`, e todos os objetos declarados como classes `OleDb` precisariam ser substituídos por seus equivalentes `SQLClient`.

Trabalhando com Múltiplas Tabelas

Quando os dados são carregados em um `DataSet`, são inseridos em tabelas individuais, o que permite que um único `DataSet` possua os resultados de muitas consultas diferentes. Depois que você tiver um `DataSet` com mais de uma tabela, será possível especificar relacionamentos entre elas, como entre as tabelas `Artist` e `Disc` (baseado nos exemplos que desenvolvemos até agora). Todos os relacionamentos que existem dentro de um `DataSet` são representados por objetos do conjunto `Relations`, que pode ser acessado como uma propriedade do próprio objeto `DataSet`. Para criar um relacionamento entre duas tabelas, use o método `Add` do conjunto `Relations`. Esse método possui muitas opções de parâmetros, permitindo que seja chamado de sete maneiras diferentes, mas se quisermos configurar um relacionamento simples com base em um único campo (coluna) da tabela-pai e um da tabela-filha, então, esta sintaxe funcionará bem:

```
objNewRelation = objDS.Relations.Add(RelationName, ParentColumn, ChildColumn)
```

Como exemplo dessa situação, a Listagem 12.17 cria uma conexão com o banco de dados e carrega as tabelas `Artist` e `Disc` em um novo `DataSet`. Depois que tudo for carregado, um relacionamento será estabelecido entre as duas tabelas.

LISTAGEM 12.17 O DataSet Reflete Algo Semelhante à Fonte de Dados

```

1 Module Module1
2
3     Private Const _
4     sConnection As String = _
5     "Provider=SQLOLEDB.1;" & _
6     "Password=321dweksp302axn;" & _
7     "Persist Security Info=True;" & _
8     "User ID=sa;" & _
9     "Initial Catalog=CD;" & _
10    "Data Source=(local)"
11
12    Sub Main()
13        Dim sSQLArtist As String
14        Dim sSQLDisc As String
15
16        sSQLArtist = "SELECT ArtistID, ArtistName From Artist"
17        sSQLDisc = "SELECT ArtistID, CDTitle From Disc"
18
19        Dim objConn _
20        As New OleDb.OleDbConnection(sConnection)
21        Dim objDataAdapterArtist _
22        As New OleDb.OleDbDataAdapter(sSQLArtist, objConn)
23        Dim objDataAdapterDisc _
24        As New OleDb.OleDbDataAdapter(sSQLDisc, objConn)
25        Dim objDS _
26        As New DataSet("CD")
27
28        objDataAdapterArtist.MissingSchemaAction = _
29        MissingSchemaAction.AddWithKey
30        objDataAdapterDisc.MissingSchemaAction = _
31        MissingSchemaAction.AddWithKey
32
33        objDataAdapterArtist.Fill(objDS, "Artist")
34        objDataAdapterDisc.Fill(objDS, "Disc")
35
36        objDS.Relations.Add("DiscArtist", _
37        objDS.Tables("Artist").Columns("ArtistID"), _
38        objDS.Tables("Disc").Columns("ArtistID"))
39    End Sub
40 End Module

```

Com os relacionamentos definidos, é possível usar essas informações para permitir o acesso estruturado de uma linha de dados-pai a qualquer linha relacionada com ela na tabela-filha. O código da Listagem 12.18, que pode ser adicionado no local exato que antecede o final do

procedimento da Listagem 12.17, executa um laço por meio dos registros da tabela Artist e exibe os CDs associados a esse artista.

LISTAGEM 12.18 Acessando as Linhas-Filhas de Qualquer Linha-Pai Específica por Meio do Método GetChildRows

```
1 Dim objDRParent As DataRow
2 Dim objDRChild As DataRow
3 Dim objChildRows() As DataRow
4
5 For Each objDRParent In objDS.Tables("Artist").Rows
6     Console.WriteLine("{0} {1}", _
7         objDRParent("ArtistID"), _
8         objDRParent("ArtistName"))
9     objChildRows = objDRParent.GetChildRows("DiscArtist")
10    For Each objDRChild In objChildRows
11        Console.WriteLine("    {0}", _
12            objDRChild("CDTitle"))
13    Next
14 Next
15 Console.ReadLine()
```

A parte essencial do código desse exemplo está na linha 9, em que o método `GetChildRows` do objeto `DataRow` é usado para retornar um array de `DataRows` da tabela-filha. Esse array inclui todas as linhas da tabela-filha em que os campos `ArtistID` da tabela-pai e da filha têm uma correspondência.

Visualizações

O ADO.NET inclui o conceito de visualizações de dados, a capacidade de especificar a ordem de classificação, um filtro sobre as linhas e um filtro com base no estado da linha (alterada, excluída, inalterada ou nova). Essa visualização (`DataView`), quando criada, pode ser acessada pela propriedade `DefaultView` da própria tabela ou pelo conjunto `Relations`, que é uma propriedade do próprio `DataSet`.

A Listagem 12.19 apresenta o código-padrão para a abertura de um banco de dados, o preenchimento de uma tabela com dados e, em seguida, a criação de um objeto `DataView` usando a tabela como seu construtor. Depois que a visualização for gerada, ela poderá ser configurada com uma combinação aleatória de três propriedades diferentes: `RowStateFilter`, `Sort` e/ou `RowFilter`. A visualização propriamente dita pode ser vinculada a um controle dos formulários Windows (veja a próxima seção para obter informações sobre a vinculação de dados), ou você pode acessar seu conteúdo diretamente como um conjunto de objetos `DataRowView`.

LISTAGEM 12.19 Vários Objetos DataView Apontando para o Mesmo DataTable

```
1 Module Module1
2     Private Const _
3     sConnection As String = _
4     "Provider=SQLOLEDB.1;" & _
5     "Password=;" & _
6     "Persist Security Info=True;" & _
7     "User ID=sa;" & _
8     "Initial Catalog=CD;" & _
9     "Data Source=(local)"
10
11 Sub Main()
12     Dim sSQLArtist As String
13     sSQLArtist = "SELECT ArtistID, ArtistName From Artist"
14     Dim objConn _
15         As New OleDb.OleDbConnection(sConnection)
16     Dim objDataAdapterArtist _
17         As New OleDb.OleDbDataAdapter(sSQLArtist, objConn)
18     Dim objDS _
19         As New DataSet("CD")
20     objDataAdapterArtist.MissingSchemaAction = _
21         MissingSchemaAction.AddWithKey
22     objDataAdapterArtist.Fill(objDS, "Artist")
23
24     Dim objDR As DataRow
25     Console.WriteLine("Na tabela")
26     For Each objDR In objDS.Tables("Artist").Rows
27         Console.WriteLine("{0} {1}", _
28             objDR("ArtistID"), _
29             objDR("ArtistName"))
30     Next
31
32     Dim objDV As DataView
33     objDV = New DataView(objDS.Tables("Artist"))
34     objDV.Sort = "ArtistName"
35     objDV.RowFilter = "ArtistID < 8"
36
37     Dim objDRV As DataRowView
38     Console.WriteLine("No objeto DataView")
39     For Each objDRV In objDV
40         Console.WriteLine("{0} {1}", _
41             objDRV("ArtistID"), _
42             objDRV("ArtistName"))
43     Next
44     Console.ReadLine()
```

LISTAGEM 12.19 Vários Objetos DataView Apontando para o Mesmo DataTable
(*continuação*)

```
45 End Sub
46 End Module
```

Até a linha 22 não temos nada tão diferente dos outros exemplos que vimos até aqui: uma conexão com o banco de dados foi aberta, e os dados foram carregados em um DataSet. Depois do carregamento de dados, um laço rápido (linhas 26 a 30) os exibiu sem dar a impressão de que percorreu um objeto DataView (na verdade percorreu a visualização de dados-padrão). Em seguida, um objeto DataView foi criado e configurado com uma ordem e um filtro sobre os valores das linhas (linhas 32 a 35). Para concluir, esse objeto DataView foi usado para acessar as linhas (linhas 39 a 43), dessa vez exibindo-as aplicando sua ordem e filtro. O resultado desse código, presumindo que você tivesse feito todos os exemplos até este ponto, deve se parecer com o apresentado a seguir:

```
Through Table
1 Natalie Imbruglia
2 The Tragically Hip
3 Sting
4 Queen
5 Boney M.
6 Barenaked Ladies
7 Kent Sharkey
8 Janet Jackson
9 Matchbox Twenty
10 Madonna
40 Kent Sharkey
Through DataView
6 Barenaked Ladies
5 Boney M.
7 Kent Sharkey
1 Natalie Imbruglia
4 Queen
3 Sting
2 The Tragically Hip
```

Mesmo se sua saída não estiver exatamente igual ao texto anterior, observe que na lista Through Table, 11 linhas foram exibidas por ordem de identificação do artista. Em Through DataView, as linhas foram acessadas por meio da visualização recém-criada, e o resultado reflete tanto o filtro quanto a ordem.

Vinculação de Dados

A vinculação de dados fornece uma maneira de associar um elemento da interface com o usuário, como uma grade, a uma fonte de dados, como um objeto DataTable ou DataSet. Depois que

um elemento da interface é vinculado a uma fonte de dados, as alterações nessa fonte serão refletidas na interface com o usuário, e as que afetarem os dados por meio da interface serão, por sua vez, retornadas à fonte de dados. Para o desenvolvedor, a vinculação de dados resolve todos os problemas complexos de manipular as interações do usuário com os dados e elimina a necessidade de escrever um código extenso apenas para fornecer um meio de o usuário interagir com seus dados. Um recurso interessante e poderoso da vinculação de dados da plataforma .NET é que ela não é limitada a trabalhar com `DataSets`. Na verdade, uma ampla variedade de objetos pode usar a vinculação de dados, incluindo `Arrays`, `Collections`, `DataViews`, `DataSets`, `DataTable` e outros. Em nossos exemplos, trabalharemos apenas com `DataSets`, `DataViews` e `DataTables`, mas existe o potencial para vincular todo tipo de informações



Quando um objeto `DataSet` ou `DataTable` é vinculado a um controle da interface com o usuário, a vinculação na verdade se dá com a visualização-padrão (objeto `DataView` ou `DataSetView`) de `DataTable` ou `DataSet`.

No universo da plataforma .NET, dois tipos principais de interfaces com o usuário podem ser criados: o cliente Windows sofisticado (`System.Windows.Forms`) e o cliente da Internet `WebForms`. Os detalhes da vinculação serão abordados na próxima seção junto a um exemplo passo a passo ocorrendo entre um dataset e uma grade. A vinculação de dados nos formulários da Web e o acesso geral aos dados do ASP.NET não serão detalhados nesta lição.

Vinculação de Dados com os Formulários Windows

Todos os controles dos formulários Windows que dão suporte à vinculação de dados vêm em duas versões principais: os que dão suporte à vinculação simples e os que podem manipular a vinculação complexa. A vinculação simples acontece quando uma propriedade de um controle é associada ao campo de uma fonte de dados. A vinculação complexa ocorre quando o controle é associado a uma fonte de dados completa (como um objeto `DataTable`), e entre os objetos embutidos ela têm suporte apenas dos controles `DataGrid` e `ComboBox`.

Vinculação Simples

Os controles que dão suporte à vinculação simples expõem um conjunto `DataBindings` que pode ser usado para vincular as propriedades a campos de dados. A Listagem 12.20 fornece um exemplo desse tipo de vinculação de dados como parte de um aplicativo com formulários Windows. Para você testar esse código, crie um novo aplicativo Windows no Visual Basic e insira apenas uma caixa de texto e um botão no formulário. Deixe os controles com seus nomes-padrão (`textBox1` e `Button1`) e inclua esse código no evento `Click` do botão.

LISTAGEM 12.20 A Vinculação de Dados Pode Ser Criada Dinamicamente em Código

```
1 Private Sub button1_Click(ByVal sender As System.Object,_
2                             ByVal e As System.EventArgs)_
3                             Handles button1.Click
4     Dim sConnection As String =_
5     "Provider=SQLOLEDB.1;" &_
6     "Password=;" &_
7     "Persist Security Info=True;" &_
8     "User ID=sa;" &_
9     "Initial Catalog=CD;" &_
10    "Data Source=(local)"
11
12    Dim sSQL As String
13    sSQL = "SELECT ArtistID, ArtistName From Artist"
14    Dim objConn _
15        As New OleDb.OleDbConnection(sConnection)
16    Dim objDataAdapter _
17        As New OleDb.OleDbDataAdapter(sSQL, objConn)
18    Dim objDS _
19        As New DataSet("Artists")
20    Dim objDV _
21        As DataView
22
23    Try
24        objConn.Open()
25    Catch myException As System.Exception
26        Windows.Forms.MessageBox.Show(myException.Message)
27    End Try
28
29    If objConn.State = ConnectionState.Open Then
30        Try
31            objDataAdapter.Fill(objDS, "Disc")
32            objConn.Close()
33            Dim objTable As DataTable
34            objTable = objDS.Tables("Disc")
35            objDV = objTable.DefaultView
36            textBox1.DataBindings.Add("Text",_
37                                    objDV,_
38                                    "ArtistName")
39        Catch myException As System.Exception
40            Windows.Forms.MessageBox.Show(myException.Message)
41        End Try
42    End If
43 End Sub
```

Além do trecho inteiro de código que você já conhece (linhas 1 a 34) abrindo a conexão com o banco de dados e obtendo um DataSet, a linha no final do procedimento (linha 37) é que realmente cria a vinculação de dados. O método Add do conjunto DataBindings permite a especificação de uma propriedade do controle e da fonte de dados (nesse caso a visualização de dados-padrão da tabela). Depois que isso for feito, o valor do campo ArtistName será exibido (veja a Figura 12.6).

Isso não é tão útil assim, considerando que você está visualizando só o primeiro registro e não tem como se mover para a frente ou para trás através dos dados. Para fornecer uma funcionalidade como essa quando você usar a vinculação simples de dados, será preciso acessar o conjunto BindingContext do formulário-pai. Ao especificar a fonte de dados na qual está interessado, você poderá obter um objeto BindingContext para ela.

FIGURA 12.6

Podemos ter uma vinculação simples de dados com vários controles diferentes ao mesmo tempo, permitindo que você exiba os campos de seus dados de várias maneiras.



No exemplo a seguir, a fonte de dados original não é uma variável no nível do formulário, portanto, o código a acessa percorrendo o conjunto DataBindings da caixa de texto. Só há uma vinculação de dados, assim, especificar um índice igual a zero retornará somente a fonte de dados disponível. Com esse objeto de contexto, se tem acesso a várias propriedades e métodos para criação de uma interface que trabalhe com os dados vinculados. Uma propriedade em particular, Position, é útil porque permite que você se mova para a frente e para trás através dos dados conforme desejar. Se adicionar um segundo botão ao formulário e inserir um código nele, você terá a capacidade de se mover para adiante na lista de artistas. Observe que não há nenhum recurso avançado nesse código; ele será interrompido se você tentar ultrapassar os dados disponíveis:

```
Private Sub button2_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) _  
    Handles button2.Click  
    Me.BindingContext(textBox1.DataBindings(0).DataSource).Position += 1  
End Sub
```

Esse objeto de contexto expõe outros recursos como AddNew e CancelCurrentEdit, tornando-o um objeto com o qual você trabalhará frequentemente.

Vinculação Complexa de Dados

NOVO TERMO

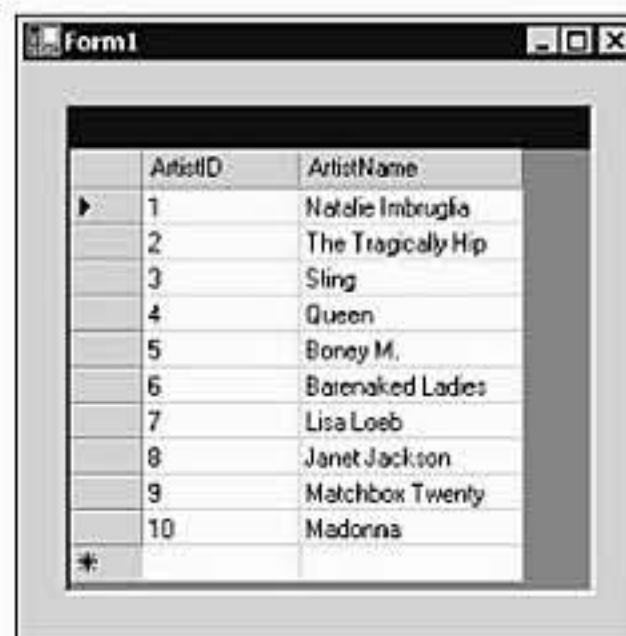
Vincular controles a uma fonte de dados inteira, em vez de a um único campo, é chamado de *vinculação complexa de dados*. Esse tipo de vinculação de dados tem suporte de dois dos controles que vêm com o Visual Studio .NET, DataGrid e ComboBox. Para configurar a vinculação de dados com um desses controles, você definirá a propriedade DataSource do controle com o objeto apropriado (DataView neste caso, mas poderia ser Array, Collection, DataSet e outros). Se DataSource tiver mais de um objeto que puder ser vinculado (como um DataSet com mais de uma tabela), então, especifique o item apropriado como um valor alfanumérico na propriedade DataMember. Portanto, se vincular DataGrid a um DataSet, terá de fornecer o nome da tabela como a propriedade DataMember. O código para fazer isso seria idêntico ao da Listagem 12.20, exceto pelo fato de ser preciso substituir as linhas 36 a 38 pelas descritas a seguir:

```
dataGrid1.DataSource = objDS  
dataGrid1.DataMember = "Disc"
```

Para que você mesmo teste a vinculação complexa, crie um novo aplicativo Windows e insira um botão e um controle DataGrid no formulário (os dois estão disponíveis na caixa de ferramentas). Deixe tudo com os nomes-padrão e insira o código da Listagem 12.20, incluindo as alterações descritas anteriormente no evento Click de seu botão. Como resultado desse código, depois que o botão for clicado, o controle DataGrid será vinculado a ele e, portanto, exibirá a tabela Artists de seu DataSet. Seu formulário pode não ter exatamente a mesma aparência da Figura 12.7, mas deve ficar semelhante.

FIGURA 12.7

Um controle DataGrid cria um ambiente de aparência profissional sem ser necessária muita codificação.



Resumo

O acesso a dados foi totalmente redesenhado para a plataforma .NET, mas ainda dá suporte a tecnologia subjacente do OLEDB. Usando os provedores de OLEDB (drivers), você poderá criar códigos que serão executados em quase todos os bancos de dados necessários, sendo preciso apenas aprender uma maneira de programar aplicável a todos eles. Empregando a natureza desconectada dos DataSets, é possível gerar sistemas completamente interativos que não ocupem

conexões valiosas com os bancos de dados. Ao construir interfaces reais com o usuário, a vinculação de dados lhe proporcionará uma maneira rápida e fácil de exibir seus dados em um formulário Windows.

P&R

- P** Em todos os exemplos desta lição, minha string de conexão estava embutida em código. Já que ela inclui minha identificação de usuário e senha, isso não me parece muito seguro.
- R** Excelente observação. Nos códigos desta lição, a string de conexão foi inserida diretamente no arquivo .vb, que é bem pouco seguro. Uma idéia melhor, quando você não estiver apenas desenvolvendo exemplos, seria inserir o valor da string de conexão no Registro ou em um arquivo de configurações no disco. Isso é feito não só por razões de segurança; não inserindo sua string de conexão no código, será possível alterar os parâmetros da conexão com o banco de dados, assim como de outras conexões, sem recompilar seu sistema.
- P** Ao atualizar o banco de dados, o que acontecerá se o registro a ser atualizado tiver sido alterado depois que você extraiu seus dados originais?
- R** Quando lidar com atualizações em um banco de dados, você deve estar atento aos problemas de simultaneidade, em que duas ou mais pessoas alteram os mesmos dados no mesmo momento. No caso dos DataSets, já que foram projetados para ser usados off-line, é provável que o período entre o download e a atualização seja relativamente longo. Durante esse tempo, outro usuário poderia ter alterado esses mesmos dados de alguma maneira. Se usarmos os comandos de atualização gerados de modo automático, eles incluirão um código que verifica esse problema antes de atualizar. Caso contrário, se construirmos os comandos manualmente, nós mesmos teremos de confrontar os dados originais com os atuais no banco de dados.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Que propriedade você precisa configurar (e com que valor) para forçar o adaptador de dados a criar informações sobre a chave primária quando carregar os dados em um DataSet?

2. Qual é a diferença entre o método `Delete` de um objeto `DataRow` e o método `Remove` do objeto `DataRowCollection`?

3. Como você deve executar esta instrução SQL?

```
DELETE FROM Artist Where ArtistID=3?
```

Exercícios

1. Crie um aplicativo Windows que use as classes que você aprendeu nesta lição para fornecer uma interface com o usuário que visualize, altere, adicione e exclua itens de seu banco de dados de CDs.

SEMANA 2

DIA 13

Usando o Server Explorer

Um dos principais objetivos de um ambiente integrado de desenvolvimento (IDE – Integrated Development Environment) é fornecer ferramentas que não estão disponíveis para os desenvolvedores que usam compiladores de linha de comandos com um editor de texto. Elas podem ser recursos aperfeiçoados de edição e depuração – ferramentas que permitem aos programadores desenvolver códigos melhores e mais rápidos. Um bom IDE também pode incluir recursos que ajudem os desenvolvedores a gerar códigos sem que seja preciso escrevê-los. Os suplementos e assistentes que fazem parte do IDE do Visual Basic .NET são todos desse tipo. O Server Explorer também se encontra nessa categoria. Ele é um recurso do Visual Basic .NET que proporciona um controle fácil dos vários serviços que compõem o sistema operacional. Além disso, ele permite que o desenvolvedor crie rápida e comodamente códigos para usar esses serviços. Nesta lição, aprenderemos

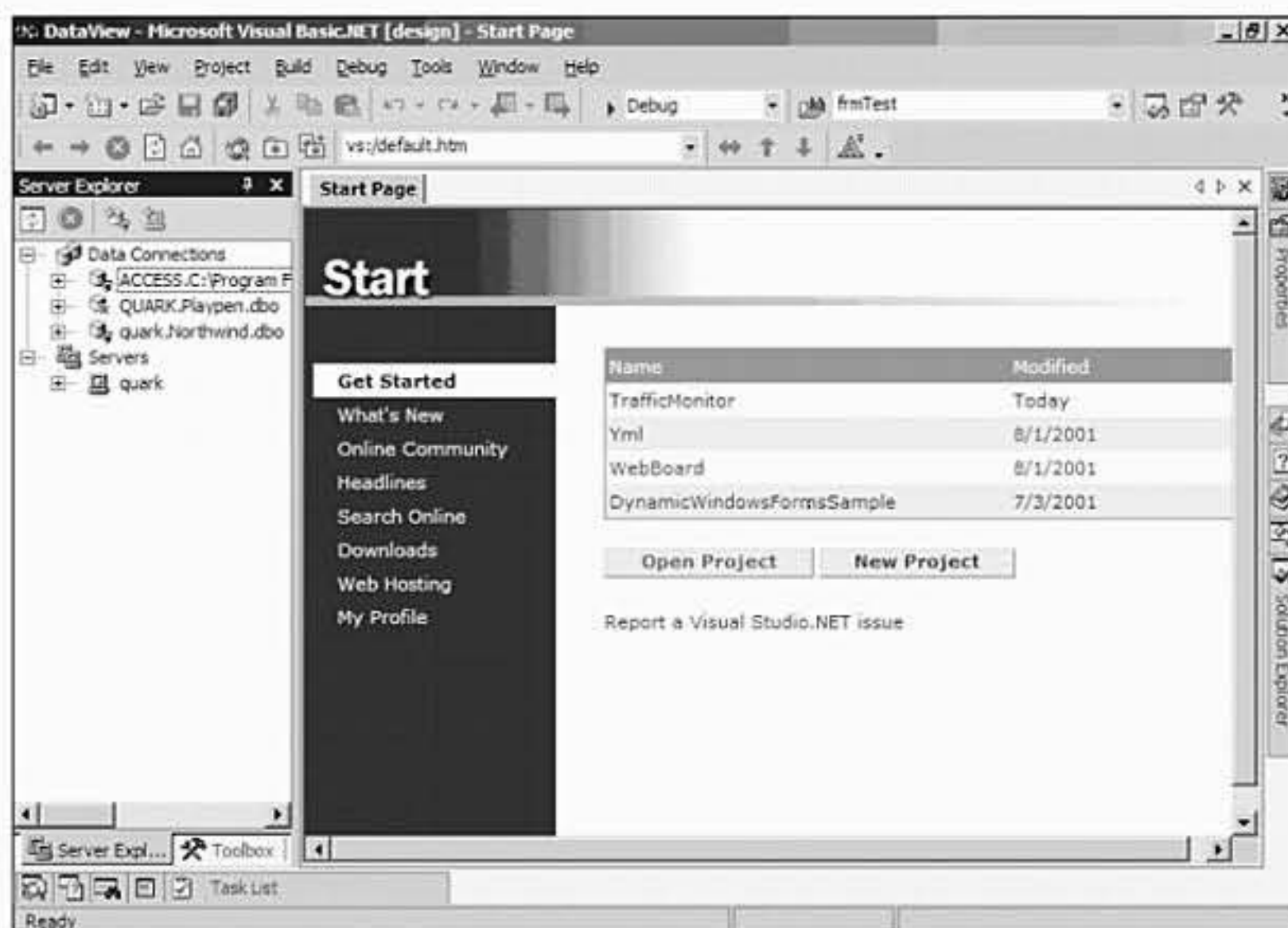
- O que é o Server Explorer.
- A explorar os serviços.
- A trabalhar com os serviços.
- A escrever programas que usem os serviços.

O Que É o Server Explorer

O Server Explorer é uma ferramenta que está disponível no IDE do Visual Basic .NET. Como o Explorer e o Internet Explorer, ele permite a pesquisa de um conjunto de recursos. Em vez de

pesquisar pastas em um disco rígido ou site da Web, disponibiliza a pesquisa em vários servidores de sua rede e a visualização dos serviços que estão em execução neles. Além disso, você pode controlar ou escrever um código para usar alguns serviços, tudo a partir do Server Explorer. Ele pode ser visto na Figura 13.1. Se não estiver visível, pode ter sido fechado anteriormente. Abra-o selecionando View e Server Explorer no menu.

FIGURA 13.1
O Server Explorer.



O Server Explorer apresenta uma lista hierárquica dos serviços disponíveis no computador selecionado. Por padrão, ele só exibe os serviços que se encontram no computador de desenvolvimento, mas você poderá adicionar outros quando necessário. Esses podem ser divididos em duas amplas categorias:

- **Conexões de dados** É semelhante ao ambiente de dados disponível no Visual Basic 6 (presumindo que você tenha usado essa versão). Permite a conexão e visualização dos bancos de dados sem ser necessário sair do ambiente do Visual Basic .NET. Este recurso possui algumas finalidades óbvias caso você tenha de desenvolver uma programação extensa para acessar e atualizar os bancos de dados, que é um dos principais usos no Visual Basic .NET.
- **Servidores** Esta é a lista de servidores com os quais você está conectado e os serviços que estão em execução (ou disponíveis) em cada um deles. As informações desta seção incluem a capacidade de acessar os registros de eventos e a monitoração do desempenho nesses computadores, assim como outros serviços, por exemplo, filas de mensagens, o SQL Server e assim por diante.

Usaremos o Server Explorer nesta lição concentrando-nos em duas de suas principais funções: visualização de informações e desenvolvimento de programas.

O Que É um Serviço?

NOVOTERMO

O *serviço* é um programa executado em segundo plano, que fornece algum recurso para complementar o sistema operacional. Esses recursos podem incluir programas que tradicionalmente são considerados servidores, como servidores de bancos de dados (por exemplo, o Microsoft SQL Server), servidores Web (como o Microsoft IIS) ou servidores de correio (como o Microsoft Exchange). Como alternativa, o serviço pode fornecer algum recurso adicional que outros programas podem utilizar, como a monitoração do desempenho ou o registro de eventos. O Windows 95, o 98 e o ME fornecem muito menos serviços do que o Windows NT ou o 2000.

Em geral os serviços possuem um programa que é usado na visualização de informações sobre ele. Por exemplo, o Event Viewer permite que você visualize os registros de eventos de um computador, ou o SQL Server Enterprise Manager disponibiliza a visualização e administração de um SQL Server. O Server Explorer complementa essas ferramentas permitindo a utilização dos serviços sem ser necessário sair do IDE.

Examinando os Serviços

Para visualizar o Server Explorer, selecione-o no menu View. Por padrão, ele aparecerá no lado esquerdo do IDE. Como acontece com outras telas, ele pode ser 'mantido aberto' ou configurado para deslizar automaticamente para fora do campo visual quando não for mais necessário. Em um computador que estiver executando o Windows 2000 com o Microsoft SQL Server instalado, ele aparecerá como mostrado na Figura 13.2. Em outro computador, com serviços diferentes instalados, esses provavelmente aparecerão.

FIGURA 13.2
O Server Explorer.

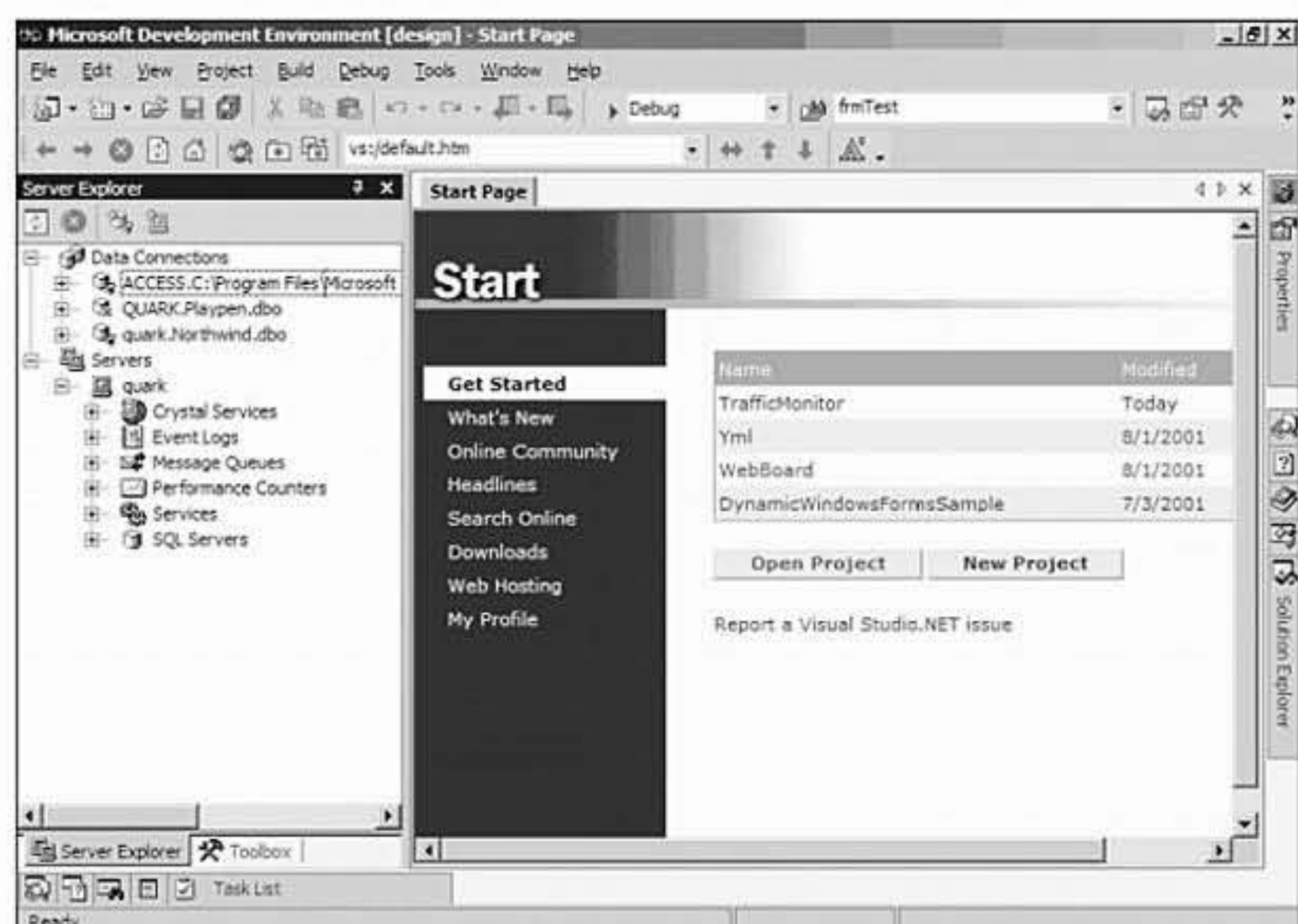
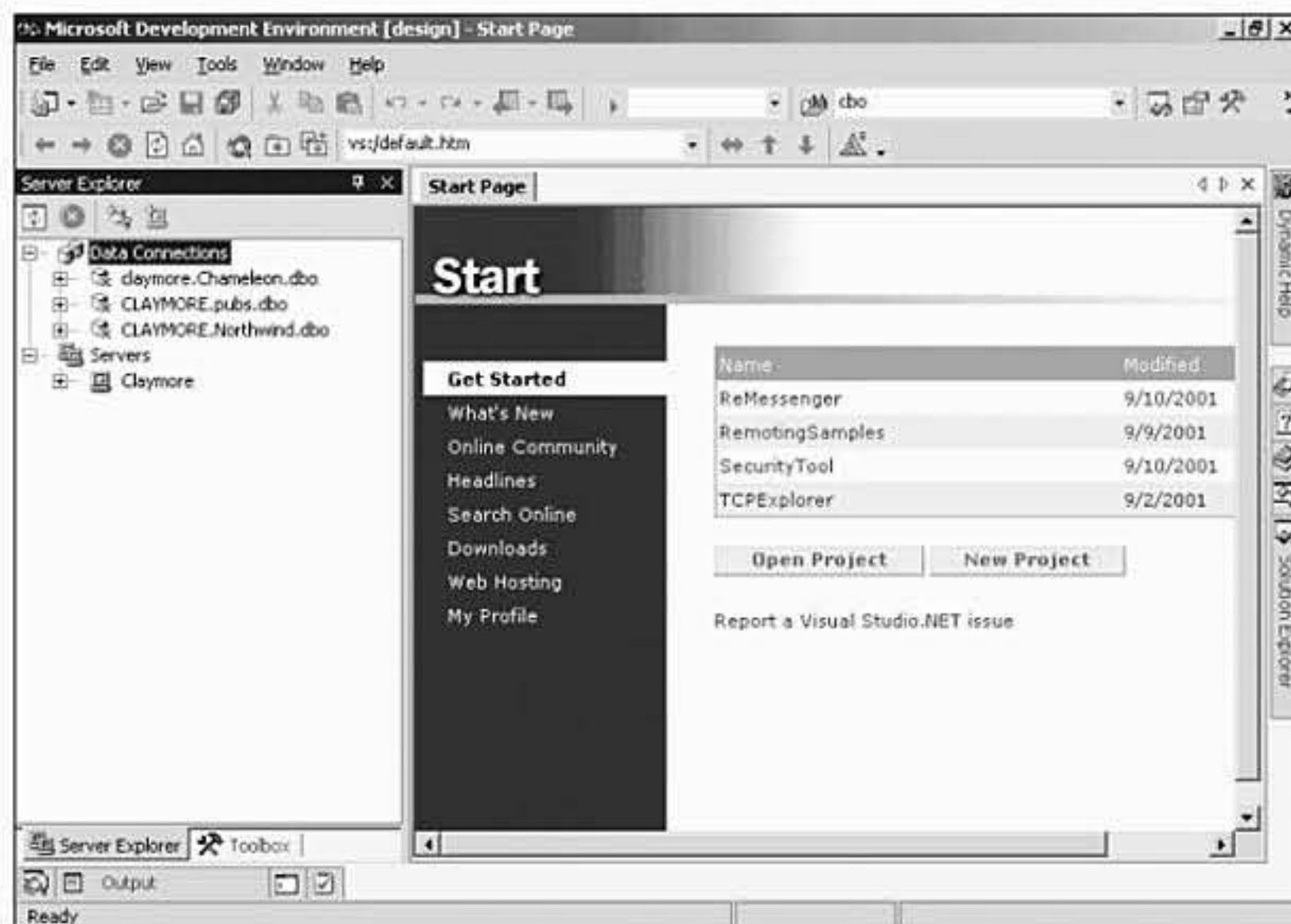


FIGURA 13.3

Visualizando conexões de dados com o Server Explorer.



Em princípio, só um link, Add Connection, deve estar disponível na pasta Data Connections. Dar um clique duplo nessa opção trará a caixa de diálogo Data Link Properties (veja a Figura 13.4). Essa caixa de diálogo permitirá que você configure uma nova conexão de dados. As opções que são selecionadas nessa caixa variam dependendo do tipo de dados com os quais se quer conectar. As configurações estão divididas entre várias guias da caixa de diálogo.

A guia Provider é usada para selecionar o provedor que será empregado na conexão com o banco de dados. Dependendo de quais provedores você tiver instalado, pode haver muitas opções. A Figura 13.4 mostra os provedores disponíveis em minha máquina.

FIGURA 13.4

Provedores instalados para acesso a dados.



Alguns dos provedores mais usados para o acesso aos dados são:

- Microsoft Jet 4.0 OLE DB Provider Permite o acesso aos bancos de dados do Microsoft Access (arquivos .mdb).
- Microsoft OLE DB Provider for ODBC Databases Permite o acesso a qualquer banco de dados que possua um driver de ODBC instalado. Muitos drivers de ODBC estão disponíveis, portanto, esse provedor pode ser usado para acessar quase todos os bancos de dados existentes. Porém, ele só deve ser empregado em último caso porque acessa as informações utilizando tanto o OLE DB quanto o ODBC. Já que há mais camadas entre seu programa e o banco de dados, esse em geral é o provedor mais lento.
- Microsoft OLE DB Provider for Oracle Permite o acesso aos bancos de dados do Oracle.
- Microsoft OLE DB Provider for SQL Server Permite o acesso aos bancos de dados do Microsoft SQL Server.

Dependendo do provedor que você selecionar nessa etapa, diferentes opções estarão disponíveis nas outras guias. A Tabela 13.1 resume as opções disponíveis na guia Connection.

TABELA 13.1 Opções da Caixa de Diálogo Data Link Properties

<i>Campo</i>	<i>Provedor</i>	<i>Finalidade</i>
Select, or enter a database name	Access	Este campo identifica o banco de dados do Access (arquivo MDB) com o qual se conectar.
User name	Todos	A conta do usuário a ser usada na conexão com o banco de dados.
Password	Todos	A senha a ser usada na conexão com o banco de dados.
Select, or enter a server name	SQL Server	Este campo identifica o servidor com o qual se conectar. Selecione '(local)' para se conectar com seu próprio computador se você estiver executando o SQL Server localmente.
Select the database on the server	SQL Server	Já que o SQL Server pode conter vários bancos de dados, este campo identifica com qual dos existentes no servidor se conectar. As guias Advanced e All raramente são usadas. A guia Advanced contém configurações que a maioria dos usuários não precisa definir, baseadas no provedor selecionado. A guia All é apenas outra maneira de acessar todas as propriedades. Em vez de ter vários campos para editar, essa guia é uma lista de todas as propriedades, que você pode editar.
Test Connection	Todos	Este botão permite que você teste os parâmetros da conexão. Isso fará com que o IDE tente se conectar com o banco de dados. Uma conexão bem-sucedida garantirá que o banco de dados possa ser acessado.

Depois de inserir todas as informações necessárias para acessar seu banco de dados, dê um clique no botão Test Connection para confirmar. Se o resultado for uma caixa de diálogo exibindo 'Test Connection Succeeded', dê um clique em OK para fechá-la e continuar. Caso contrário, a caixa de diálogo deve sugerir uma maneira de corrigir o erro (em geral a causa é a falta de definição de uma configuração em algum local). Corrija o erro e teste a conexão novamente.

Depois de se conectar a um banco de dados, ele deve aparecer na seção Data Connections do Server Explorer. Dependendo de seu tipo, a entrada possuirá várias seções. A Figura 13.5 mostra os itens existentes para alguns dos tipos de bancos de dados disponíveis.

A Tabela 13.2 descreve os tipos de itens que você deve ver para os diversos bancos de dados com os quais normalmente pode se conectar.

FIGURA 13.5
Data Connections.

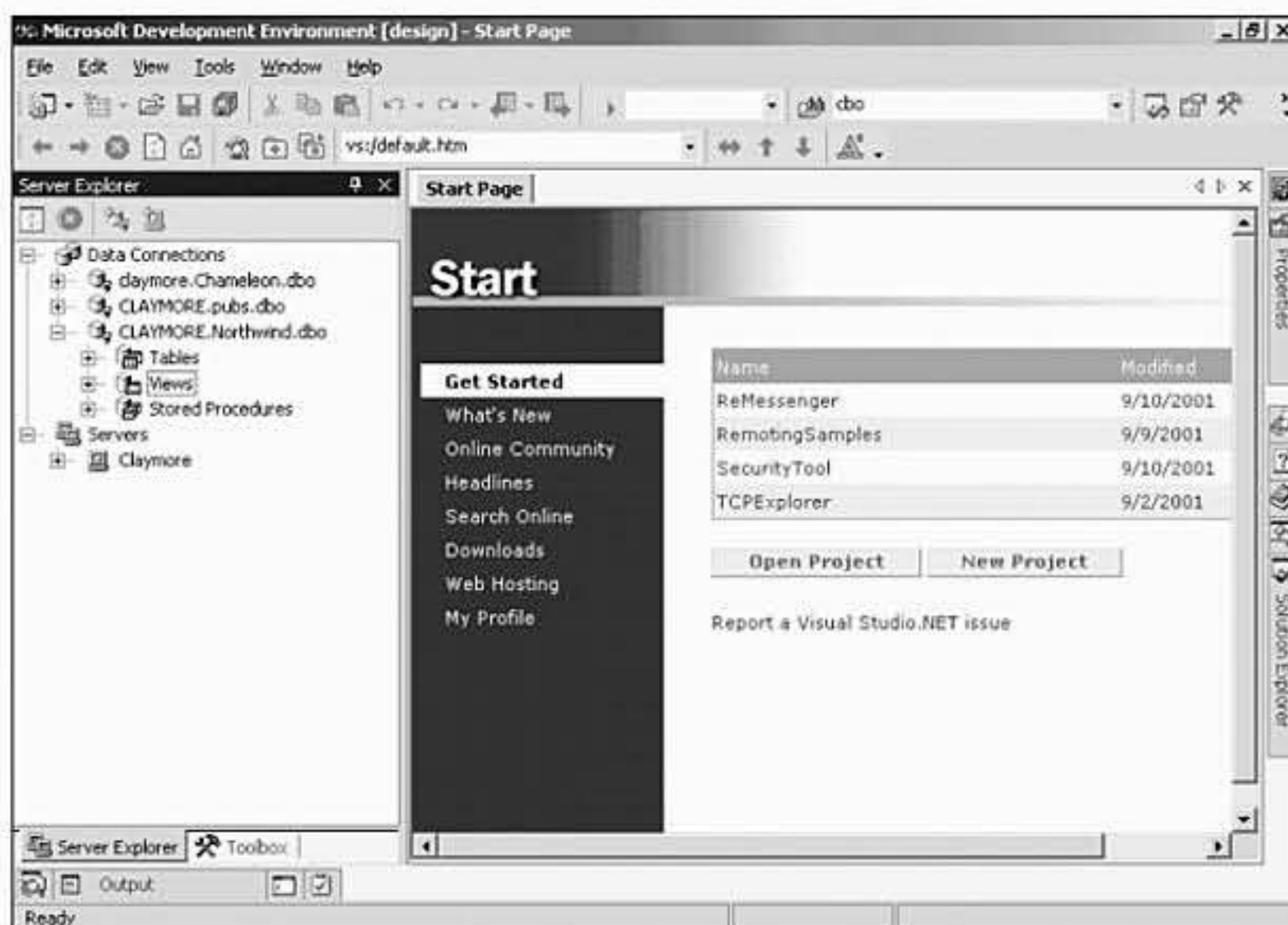


TABELA 13.2 Objetos de bancos de dados

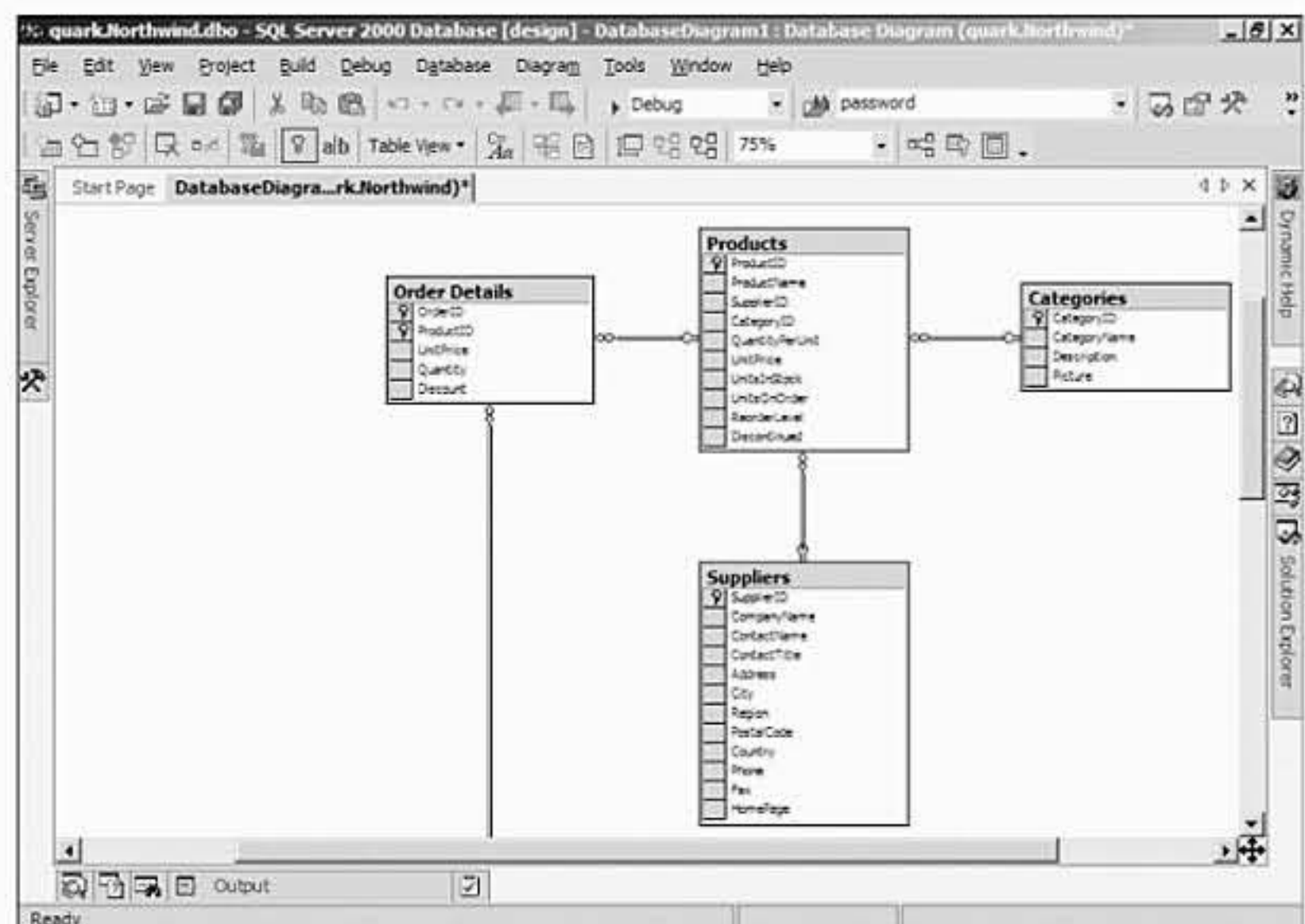
<i>Objeto</i>	<i>Descrição</i>
Tables (tabelas)	Contém uma lista de todas as tabelas armazenadas em um banco de dados. Uma tabela é um agrupamento de informações. Por exemplo, ela pode conter informações sobre os funcionários de uma empresa, os produtos que ela vende ou itens semelhantes relacionados. Em geral, esta é a seção disponível mais importante de Data Connections.
Views (visualizações)	Contém uma lista das visualizações de dados do banco de dados. A visualização é uma maneira de examinar as informações – por exemplo, ela pode não exibir todos os campos, ordenar as informações de um modo diferente ou incluir dados de várias tabelas.
Stored procedures (procedimentos armazenados)	Contém uma lista dos programas armazenados no banco de dados. Aí estão incluídas as consultas que podem retornar informações, assim como programas que podem excluir, atualizar ou inserir dados.

TABELA 13.2 Objetos de bancos de dados (*continuação*)

<i>Objeto</i>	<i>Descrição</i>
Functions (funções)	Semelhante aos procedimentos armazenados, porém, as funções são específicas para alguns bancos de dados. Em geral são usadas para retornar pequenos trechos de informação e, por enquanto, são parecidas com as funções que usamos no Visual Basic .NET.
Database diagrams (diagramas de bancos de dados)	Em geral só são encontrados no SQL Server. São usados para documentar o banco de dados e para fornecer uma descrição visual de suas tabelas. (Veja a Figura 13.6 para um exemplo.)

FIGURA 13.6

Diagrama de banco de dados.



Caminho a Ser Percorrido para Estabelecer uma Conexão com o Banco de Dados

Como descrevi anteriormente, a seção mais importante de Data Connections são as tabelas. Elas representam um conjunto de informações armazenadas no banco de dados. Com esse objeto, você pode visualizar os dados do banco de dados, alterar as tabelas e assim por diante. Usaremos o Data Connections para a conexão e visualização de um banco de dados. Empregaremos o banco de dados Northwind que vem com o Visual Basic .NET como exemplo.

1. Dê um clique duplo em Add Connection para acessar a caixa de diálogo Data Link Properties.
2. Selecione a guia Provider e Microsoft Jet 4.0 OLE DB Provider. Dê um clique em Next.
3. Dê um clique no botão próximo ao campo Select or Enter a Database Name. Navegue para encontrar um banco de dados do Access. Deve haver um em %directory to Visual

Basic.NET%\Common7\Tools\Bin\nwind.mdb. Dê um clique em OK para aceitar esse banco de dados.

- 4 Dê um clique no botão Test Connection. Uma caixa de diálogo deve aparecer informando que a conexão foi bem-sucedida. Caso contrário, certifique-se de que haja um banco de dados no local selecionado. Dê um clique em OK para aceitar essa conexão de dados.
5. A conexão resultante possui várias tabelas, visualizações e procedimentos armazenados. Abra a pasta Tables e dê um clique duplo na pasta Employees. Uma nova janela deve ser aberta, mostrando a lista de funcionários da empresa Northwind. Também deve ser possível dar um clique em outras tabelas para abri-las e visualizar os dados nelas armazenados.

Faça	Não Faça
	Não faça nenhuma alteração no banco de dados agora. Se você alterar esse banco de dados, outros exemplos ou aplicativos podem não funcionar. Apenas examine as informações por enquanto. Se quiser tentar fazer alterações em um banco de dados, crie um para teste.

Trabalhando com os Serviços

Além de permitir que você visualize os bancos de dados, o Server Explorer também torna fácil o acesso, a visualização e o controle dos serviços.

Bem, o que é um serviço? Um serviço é um aplicativo que é executado em segundo plano, fornecendo alguns recursos ao sistema operacional. Eles são familiares aos desenvolvedores que têm usado o Windows NT e o Windows 2000. No entanto, os que utilizam o Windows 95, o 98 ou o ME podem não conhecê-los tanto porque esses sistemas operacionais tendem a ter menos serviços em execução. Alguns dos serviços processados no Windows NT e no 2000 são o registro de eventos (um registro central que envolve o sistema, a segurança e os aplicativos), os contadores de desempenho (registram diversos valores significativos relativos ao sistema operacional e outros aplicativos) e o spooler de impressão (gerencia os trabalhos de impressão). Além desses, outros serviços podem ser instalados, como o IIS, o SQL Server (banco de dados) ou o Message Queue Server (permite a comunicação assíncrona, ou desconectada, de um aplicativo com o outro, semelhante ao correio eletrônico).

No Windows 95, no 98 e no ME, os serviços fazem parte do próprio sistema operacional ou são programas que costumam aparecer na barra de tarefas.

Visualizando os Serviços

Exatamente como as opções do Data Connections, os serviços do Server Explorer variam, dependendo do que estiver instalado e em execução no computador. Alguns dos itens do Server Explorer estão descritos na Tabela 13.3. Esses são os que estão disponíveis enquanto escrevia este texto; muitos outros poderão ser disponibilizados posteriormente.

TABELA 13.3 Serviços do Server Explorer

<i>Serviço</i>	<i>Descrição</i>
Registro de eventos (Event Log)	Fornece acesso aos arquivos de registro dos aplicativos (Application), da segurança (Security) e do sistema (System) do Windows NT ou do 2000. Esses arquivos de registro contêm erros e outras informações sobre os aplicativos que estão em execução no computador. Isso proporciona a mesma funcionalidade do aplicativo Event Viewer, sem que seja necessário sair do IDE do Visual Basic .NET.
Módulos carregados (Loaded Modules)	Fornece uma lista de todas as DLLs carregadas na memória e dos programas que as utilizam. Essa lista pode ser longa, dependendo do que você esteja executando em um dado momento. Pode ser útil para se saber se uma DLL específica está carregada na memória e, caso esteja, se está sendo usada por um aplicativo.
Dados de gerenciamento (Management Data)	Permite ao servidor acessar o Windows Management Information (WMI). O WMI é um meio de consultar informações em um computador. Essas informações variam de valores simples, como a CPU ou o sistema operacional instalado, a programas em execução, e são precisas como as configurações das impressoras instaladas. O WMI é uma ferramenta poderosa e melhor quando usada para recuperar informações, e não para alterá-las.
Contadores de desempenho (Performance Counters)	Concede acesso a dados sobre o desempenho do computador. O Windows NT e o Windows 2000 registram constantemente informações quando você executa seus programas. Aí estão incluídas informações sobre o tempo necessário para executar tarefas, a memória utilizada e muito mais. Podem ser acessados para avaliar a utilização da CPU, a memória e assim por diante.
Processos (Processes)	Fornece uma lista de todos os aplicativos em execução. Você pode se surpreender ao ver quantos programas são processados em seu computador.
Serviços (Services)	Mostra uma lista dos serviços em execução no computador. Esses são os aplicativos processados em segundo plano. Um ótimo recurso é que você pode usar essa seção para iniciar ou encerrar os serviços (dependendo da segurança).
Bancos de dados do SQL Server (SQL Server Databases)	Fornece mais uma maneira (a seção Data Connections é a outra) de acessar os bancos de dados do SQL Server. Disponibiliza informações semelhantes à Data Connections (no entanto, só sobre os bancos de dados do SQL Server).
Serviços da Web (Web Services)	Fornece uma lista dos serviços da Web instalados no computador (para obter mais detalhes, veja o Dia 21, "Criando Serviços da Web com o Visual Basic .NET").
Filas de mensagens (Message Queues)	Fornece uma lista das filas de mensagens disponíveis. Elas são usadas quando são criados aplicativos com base em mensagens (porém não se aplicam a aplicativos de correio eletrônico).

Conectando-se com Outro Servidor

Para examinar alguns dos serviços listados na Tabela 13.3, pode ser necessário se conectar a um servidor. Dê um clique duplo no item Add Server do Server Explorer, digite um nome de servidor válido na caixa de diálogo (veja a Figura 13.7) e dê um clique em OK. Você deve poder (dependendo da segurança da rede) visualizar os serviços carregados nessa máquina, assim como os da sua. Você pode especificar uma conta de usuário diferente para ser usada se esse tiver direito de acesso a outro computador.

FIGURA 13.7
Adicionando outro servidor.



Você pode usar esse recurso para se conectar aos servidores com os quais trabalha. Por exemplo, muitos desenvolvedores precisam acessar um servidor de banco de dados, um de teste, e possivelmente um servidor Web quando desenvolvem um programa. O Server Explorer permitiria a conexão com todos os três quando fosse preciso monitorar e/ou empregar os serviços.

Escrevendo Programas Que Usam os Serviços

Um dos recursos mais interessantes do Server Explorer é que ele permite que você não só visualize os serviços e trabalhe graficamente com eles, mas também que escreva com facilidade aplicativos que se comuniquem com eles. Por exemplo, os objetos disponibilizados pelo Server Explorer podem ser usados para incluir em um aplicativo o acesso aos dados. Além disso, é possível adicionar objetos a seu aplicativo que possam controlar ou interagir com os serviços – como ler contadores de desempenho ou ainda ler ou gravar informações nos registros de eventos do Windows NT ou do Windows 2000. Esses são recursos que precisavam de uma codificação extensa e agora podem ser obtidos em algumas linhas, com a alteração das propriedades e execução dos métodos dos objetos que os representam.

Escrevendo Códigos de Acesso a Dados com o Server Explorer

Embora o Server Explorer seja útil quando se quer visualizar os serviços, se torna muito mais quando a intenção é acessá-los. Ele pode ser usado na criação de códigos extensos, poupando-o de ter de desenvolvê-los – por exemplo, na criação de aplicativos de acesso a dados. Em geral, é

necessário escrever (como você viu no Dia 11, “Introdução aos Bancos de Dados”, e no Dia 12, “Acessando Dados com a Plataforma .NET”) o mesmo tipo de código para recuperar informações e exibi-las para o usuário. Por que perder tempo com isso, se seu computador pode fazê-lo para você?

Criaremos um aplicativo simples para visualizar os dados de uma tabela em um banco de dados. Usarei o banco de dados Northwind; você pode escolher qualquer banco de dados com o qual tenha uma conexão de sua máquina.

Depois que tiver uma conexão válida com um banco de dados no Server Explorer, crie um novo projeto de aplicativo Windows. O meu se chama DataView; você pode usar esse nome (não está patenteado ou algo semelhante) ou escolher outro. Depois de criar o projeto, exclua o formulário original dando um clique com o botão direito do mouse sobre ele no Solution Explorer e selecionando Delete. Aceite o aviso. A seguir, dê um clique com o botão direito do mouse no projeto e selecione Add e Add Windows Form. Nomeie o formulário resultante como Customers. Dê um clique com o botão direito do mouse no projeto pela última vez e selecione Properties. Configure o objeto de inicialização (Startup object) como Customers e selecione OK.

Como sempre, a primeira etapa na construção de um aplicativo do Visual Basic .NET é adicionar os controles ao formulário. Em nosso caso, teremos uma interface com o usuário simples – nada a não ser uma grade. Selecione o controle DataGridView na caixa de ferramentas e adicione-o a seu formulário. Configure as propriedades como mostra a Tabela 13.4.

TABELA 13.4 Propriedades da Grade de Dados

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Form	Text	Customer listing
DataGridView	Name	dbgCustomers
	Dock	Fill
	CaptionText	Customers

Você também pode querer usar o recurso AutoFormat (ou configurar manualmente os valores) para fazer com que a grade fique com uma boa aparência.

A seguir, queremos adicionar o código para o acesso aos dados. Em vez de fazê-lo manualmente, no entanto, usaremos o Server Explorer. Selecione uma tabela em seu banco de dados (escolhi a tabela Customers de Northwind), arraste-a do Server Explorer e solte-a no formulário. Dois objetos novos aparecerão no formulário – OleDbConnection1 e OleDbDataAdapter1 (veja a Figura 13.8).

O objeto OleDbConnection1 é, como você já deve ter adivinhado, a conexão com o banco de dados. O objeto OleDbDataAdapter1 é o componente que extrai (e salva) as informações do banco de dados. Altere o nome dos controles conforme mostra a Tabela 13.5.

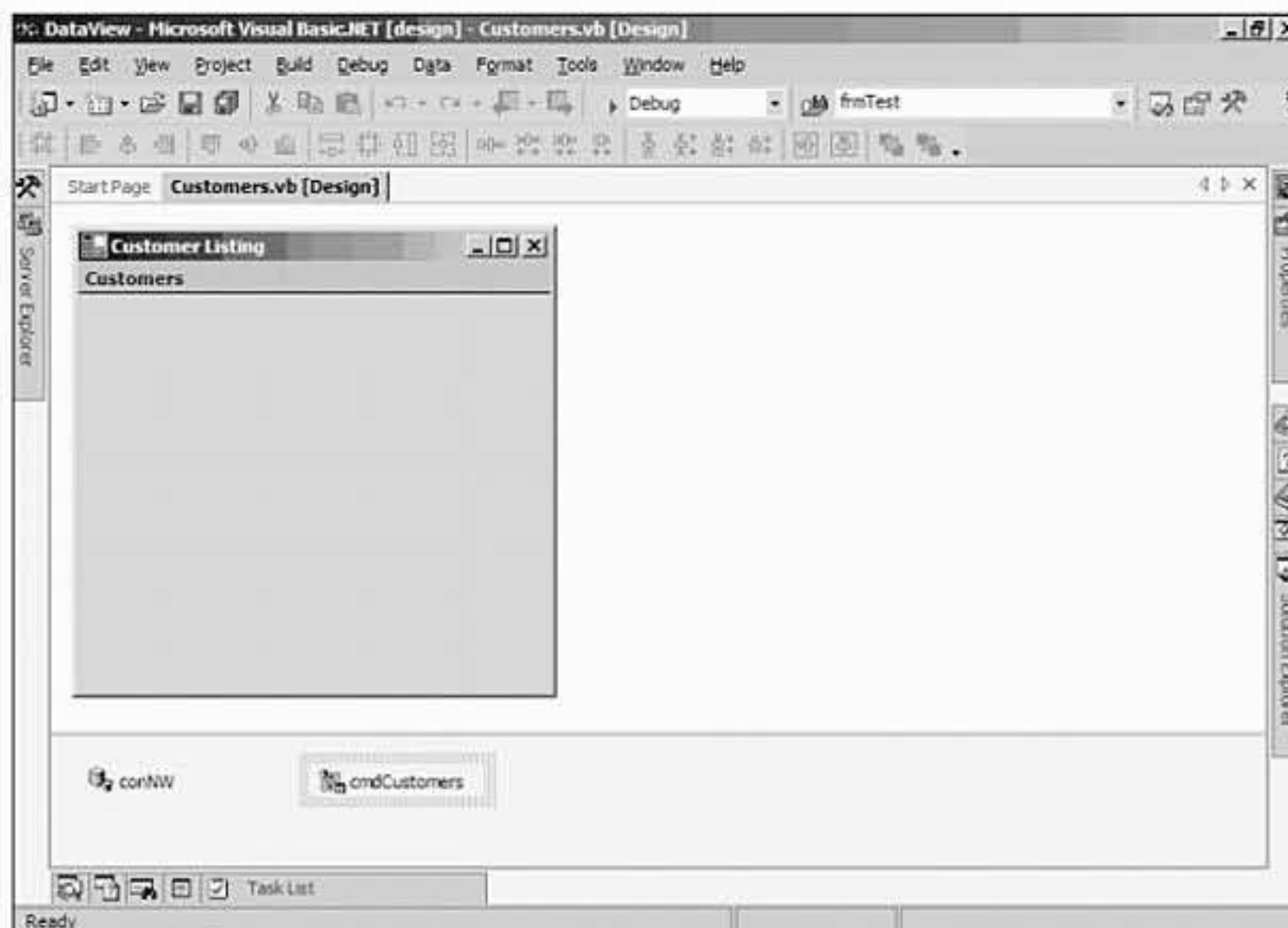
TABELA 13.5 Propriedades da Conexão e do Comando

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
OleDbConnection	Name	ConNW
OleDbDataAdapter	Name	CmdCustomers

Agora vem a parte divertida. Dê um clique com o botão direito do mouse no adaptador de dados cmdCustomers e selecione Generate DataSet; configure o nome como CustomerSet e marque a caixa que adiciona uma instância ao projeto (veja a Figura 13.9). Você deve ouvir mais alguns ruídos no disco rígido e um novo dataset CustomerSet1 será adicionado à janela (veja a Figura 13.10). Altere o nome para dsCustomers.

FIGURA 13.8

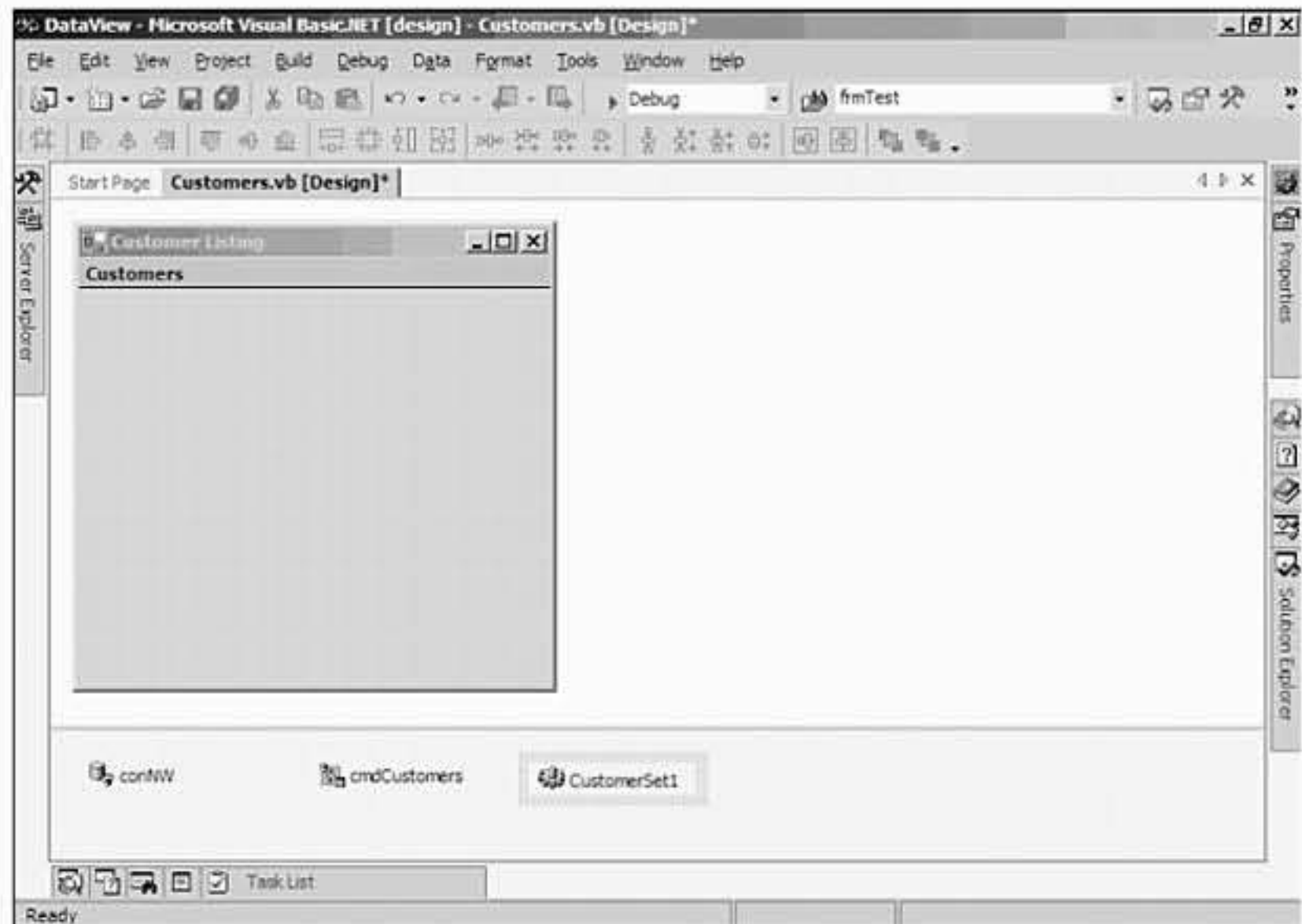
Objetos de conexão adicionados.

**FIGURA 13.9**

Adicionando o DataSet.

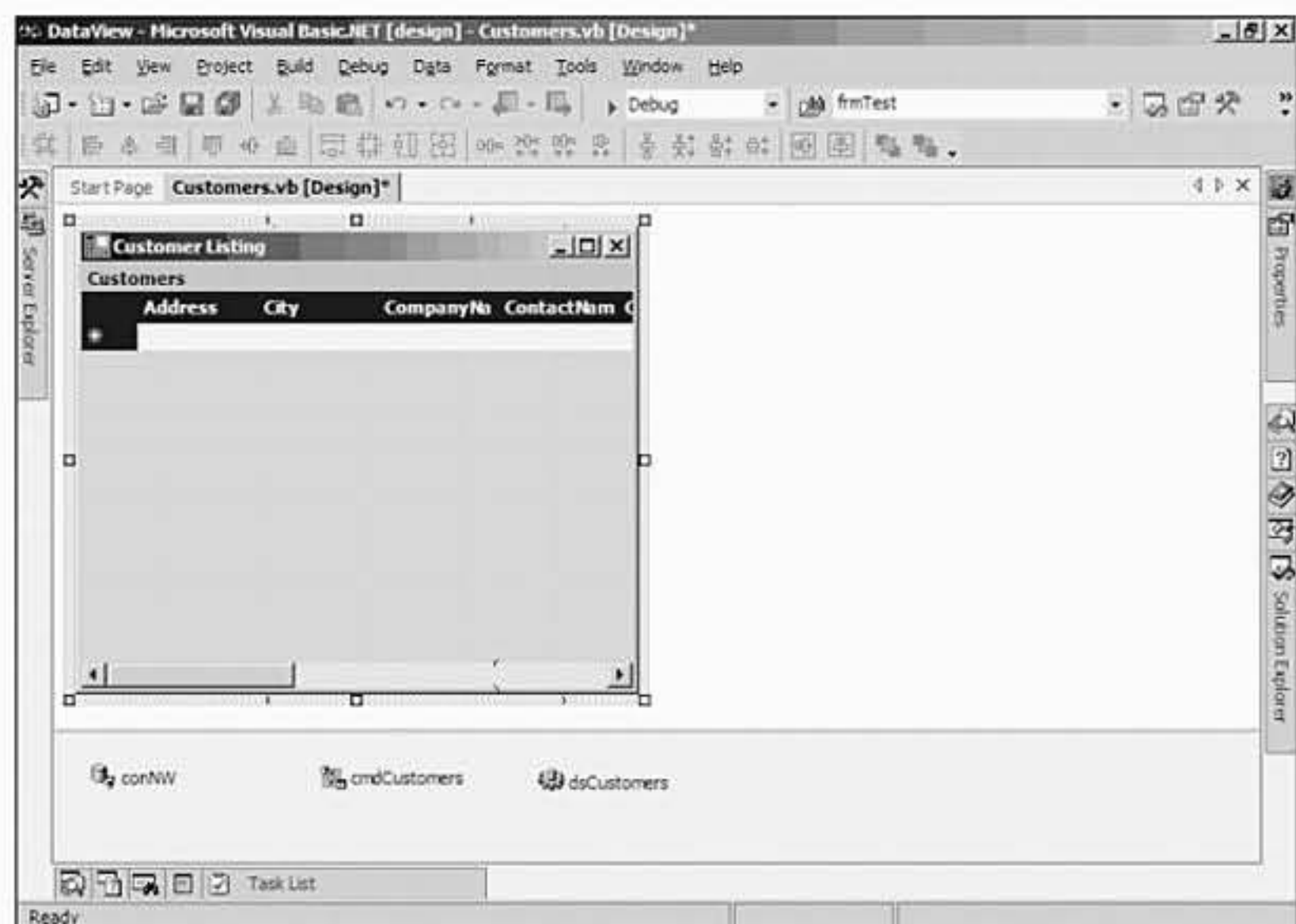


FIGURA 13.10
O DataSet
adicionado.



Antes de escrevermos o código, devemos conectar o banco de dados e o DataSet ao DataGrid. Configure a propriedade DataSource do DataGrid como dsCustomers, e DataMember como Customers. Ao fazê-lo, os cabeçalhos da coluna devem aparecer no DataGrid (veja a Figura 13.11).

FIGURA 13.11
Conectando o DataSet
ao DataGrid.



CÓDIGO

Agora vem a parte trabalhosa – ainda precisaremos escrever um código. Localize o procedimento `Form_Load` e atualize-o para que preencha o DataSet como mostrado na Listagem 13.1.

CÓDIGO**LISTAGEM 13.1** Alterações no Novo Procedimento

```

1 Private Sub Customers_Load(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles MyBase.Load
3     cmdCustomers.Fill(dsCustomers)
4 End Sub

```

ANÁLISE

O procedimento `Form_Load` será padrão para os aplicativos Windows, com a adição da linha 3. Ela chama o método `Fill` do adaptador de dados `cmdCustomers`. O Data-Set `dsCustomers` foi criado quando selecionamos `Generate DataSet`. Em outras palavras, em vez de termos de digitar o que precisávamos nessa linha, foi tudo gerado para nós. Construa e execute seu programa. Se tudo funcionar, ele deve ter a aparência da Figura 13.12.

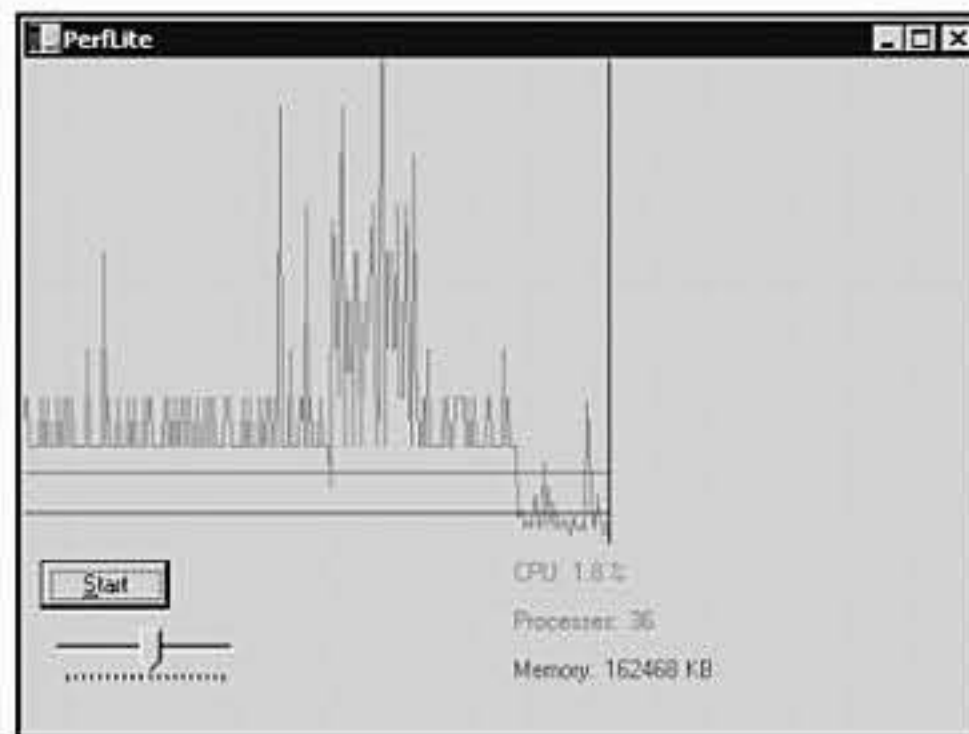
FIGURA 13.12

Executando o aplicativo DataView.

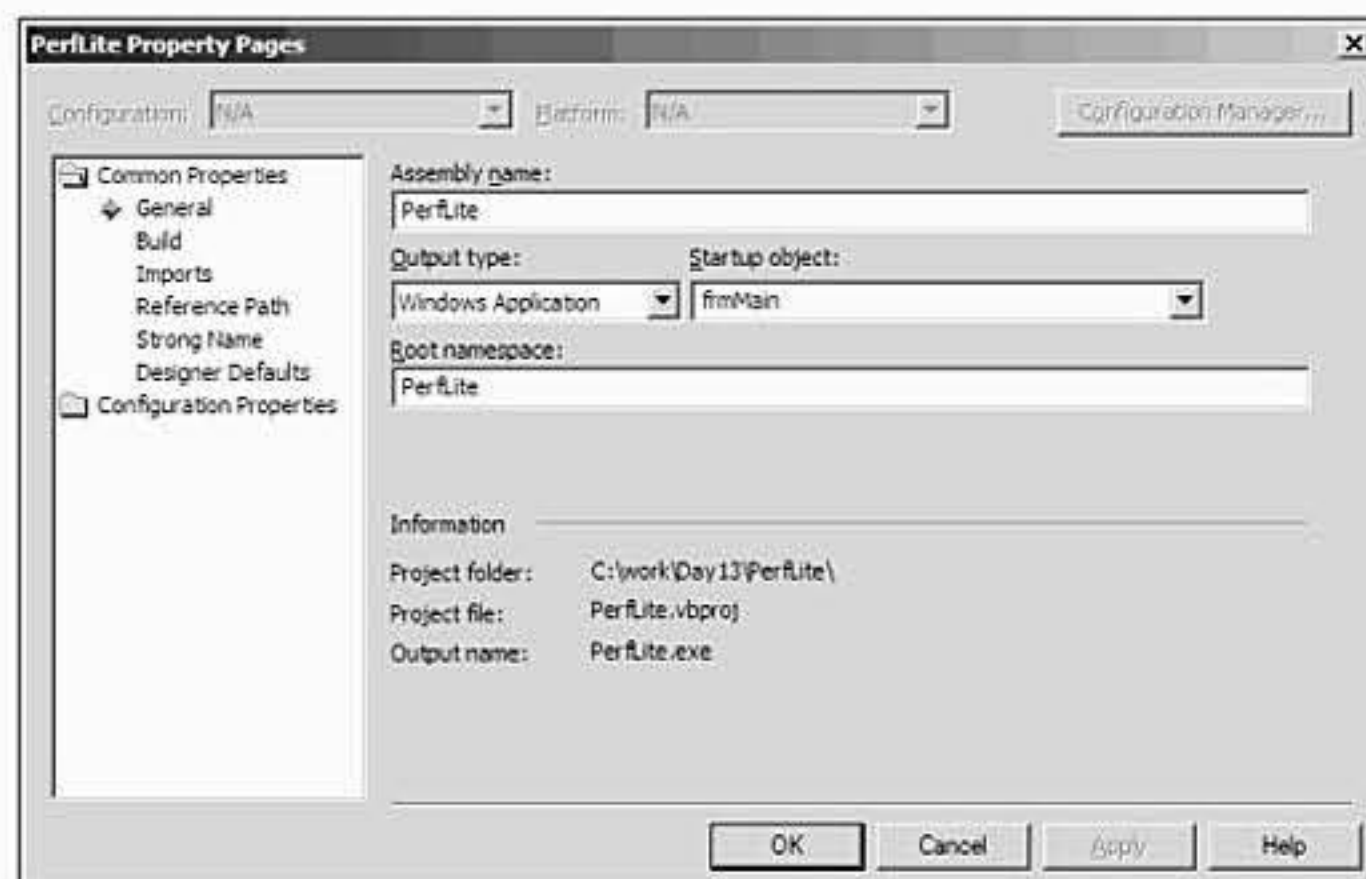
Address	City	CompanyName	ContactName	ContactTitle	Country	CustomerID	Fax
Obere Str. 57	Berlin	Alfreds Futterk	Maria Anders	Sales Represe	Germany	ALFKI	030-0076
Avda. de la Co	México D.F.	Ana Trujillo E	Ana Trujillo	Owner	Mexico	ANATR	(5) 555-3
Mataderos 23	México D.F.	Antonio Moren	Antonio Moren	Owner	Mexico	ANTON	(null)
120 Hanover	London	Around the Ho	Thomas Hardy	Sales Represe	UK	AROUT	(171) 555
Berguvsvägen	Luleå	Berglunds sna	Christina Bergl	Order Adminis	Sweden	BERGS	0921-12 1
Forsterstr. 57	Mannheim	Blauer See Del	Hanna Moos	Sales Represe	Germany	BLAUS	0621-089
24, place Kléb	Strasbourg	Blondel père e	Frédérique Cit	Marketing Ma	France	BLONP	88.60.15.
C/ Araquil, 67	Madrid	Bóldo Comida	Martín Somme	Owner	Spain	BOLID	(91) 555 !
12, rue des Bo	Marseille	Bon app'	Laurence Lebl	Owner	France	BONAP	91.24.45.
23 Tsawassen	Tsawassen	Bottom-Dollar	Elizabeth Unc	Accounting Ma	Canada	BOTTM	(604) 555
Fauntleroy Cir	London	B's Beverages	Victoria Ashwo	Sales Represe	UK	BSBEV	(null)
Cerrito 333	Buenos Aires	Cactus Comid	Patricio Simps	Sales Agent	Argentina	CACTU	(1) 135-4
Sierras de Gra	México D.F.	Centro comerc	Francisco Cha	Marketing Ma	Mexico	CENTC	(5) 555-7
Hauptstr. 29	Bern	Chop-suey Chi	Yang Wang	Owner	Switzerland	CHOPS	(null)
Av. dos Lusíad	São Paulo	Comércio Mine	Pedro Afonso	Sales Associat	Brazil	COMMI	(null)
Berkeley Gard	London	Consolidated	Elizabeth Bro	Sales Represe	UK	CONSH	(171) 555
Walsenweg 21	Aachen	Drachenblut D	Sven Ottlieb	Order Adminis	Germany	DRACD	0241-059
67, rue des Ci	Nantes	Du monde ent	Janine Labrun	Owner	France	DUMON	40.67.69.
35 King Georg	London	Eastern Conne	Ann Devon	Sales Agent	UK	EASTC	(171) 555
Kirchgasse 6	Graz	Ernst Handel	Roland Mendel	Sales Manager	Austria	ERNSH	7675-342
Rua Orós, 92	São Paulo	Família Arquib	Aria Cruz	Marketing Assi	Brazil	FAMIA	(null)
C/ Moralzarzal	Madrid	FISSA Fabrica	Diego Roel	Accounting Ma	Spain	FISSA	(91) 555 !

Acessando os Contadores de Desempenho e os Registros de Eventos

Para examinarmos os benefícios das ferramentas do Server Explorer, construiremos um aplicativo pequeno que monitorará alguns dos contadores de desempenho que o sistema operacional mantém sobre ele próprio. Além disso, registraremos a data e a hora em que o programa começar e terminar de registrar os eventos do aplicativo. O resultado final deve ter uma aparência semelhante à da Figura 13.13.

FIGURA 13.13*O PerfLite em ação.*

Crie um novo aplicativo Windows. Chame o novo projeto de PerfLite porque ele será uma versão mais enxuta (ou 'Lite' na linguagem de marketing) do aplicativo Performance Monitor que vem com o Windows NT e o Windows 2000. Quando o projeto estiver pronto, feche o formulário-padrão e renomeie-o como `frmMain.vb`. A seguir, dê um clique com o botão direito do mouse no arquivo e selecione View Code. Altere todas as referências a `Form1` para `frmMain`. Usar o comando Replace ajudará (lembre-se de selecionar Search Hidden Text). Para concluir, dê um clique com o botão direito do mouse sobre o projeto no Solution Explorer, selecione Properties e altere o Startup Object no formulário resultante para `frmMain` (veja a Figura 13.14).

FIGURA 13.14*Propriedades de PerfLite.*

Compile o aplicativo para certificar-se de que todas as alterações foram feitas.

Depois de tudo compilado, estaremos prontos para começar a adicionar os controles e configurar suas propriedades. Configure as propriedades do formulário como mostra a Tabela 13.6.

TABELA 13.6 Propriedades do Formulário PerfLite

<i>Propriedade</i>	<i>Valor</i>
Text	PerfLite
Size	480, 360

Exibiremos algumas informações sobre o desempenho desenhando-as em um painel. Além disso, temos de controlar a velocidade da exibição. Adicione os controles da Tabela 13.7 e configure suas propriedades como mostrado na tabela.

TABELA 13.7 Controles para o Formulário PerfLite

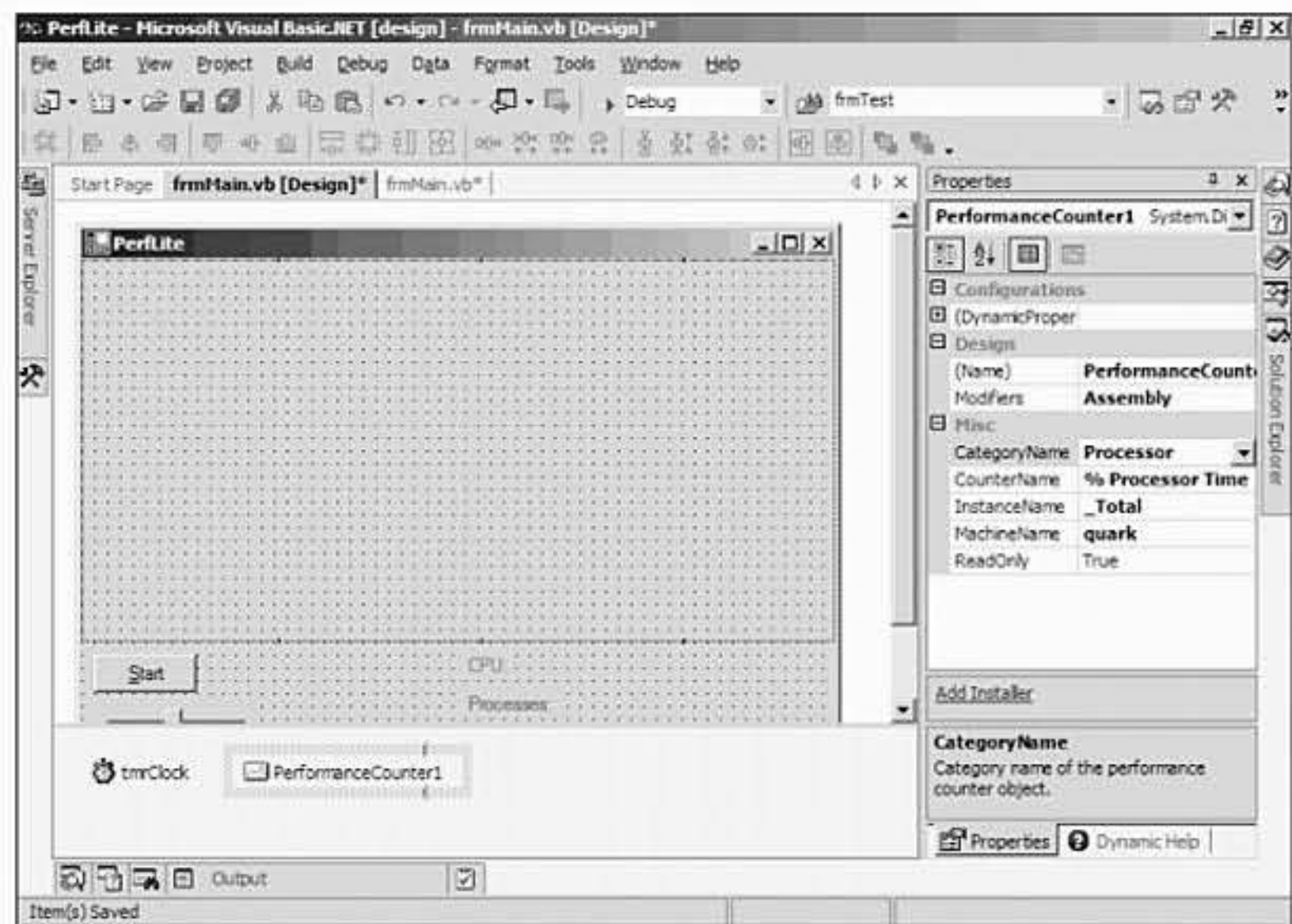
<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Panel	Name	pnlSweep
	Dock	Top
	Height	240
Button	Name	cmdSweep
	Text	&Start
	Location	8, 248
	Size	64, 24
TrackBar	Name	trkSpeed
	Location	8, 280
	Size	104, 42
	Minimum	1
	Maximum	20
Label	Value	10
	Name	lblCPU
	Location	240, 248
	Autosize	True
Label	Text	CPU:
	Forecolor	Red (ou algo que você goste/possa ver)
	Name	lblProc
	Location	240, 272
Label	Autosize	True
	Text	Processes:
	Forecolor	Green (novamente, ou algo atrativo)
	Name	lblMem
Label	Location	240, 296
	Autosize	True
	Text	Memory:
	Forecolor	Blue
Timer	Name	tmrClock
	Interval	1000

Para exibirmos as informações sobre o desempenho, precisamos adicionar os contadores apropriados ao aplicativo:

1. Abra o Server Explorer e selecione um servidor (pode ser seu próprio computador).
2. Abra a seção Performance Counters.
3. Role para baixo até encontrar o item Processor e abra-o.
4. Abra o item % Processor Time e selecione o item _Total.
5. Arraste esse item sobre o formulário e solte-o sobre ele. Um item chamado PerformanceCounter1 deve ficar visível em uma nova seção abaixo do formulário (veja a Figura 13.15).

FIGURA 13.15

Depois de incluir o contador de desempenho.



Há outra maneira de adicionar um item do Server Explorer a seu aplicativo. Localize o contador de desempenho System\Processes. Observe que ele não possui nenhum item, como _Total em % Processor Time. Dê um clique com o botão direito do mouse no contador Processes e selecione Add to Designer. Em seguida, você deve ver outro objeto Performance Counter adicionado próximo ao primeiro.

Para concluir a inserção de nossos contadores de desempenho, selecione e adicione o contador de desempenho Memory\Committed Bytes.

Renomeie os três contadores de desempenho como prfCPU, prfProcs e prfMem.

Além dos contadores de desempenho, também podemos adicionar objetos que tornem o acesso aos dados muito mais fácil do que já possa ter sido. Abra a seção Event Logs do Server Explorer e selecione Application. Arraste o item Application para o formulário ou dê um clique com o botão direito do mouse e selecione Add To Designer (o que preferir). Um novo item deve aparecer

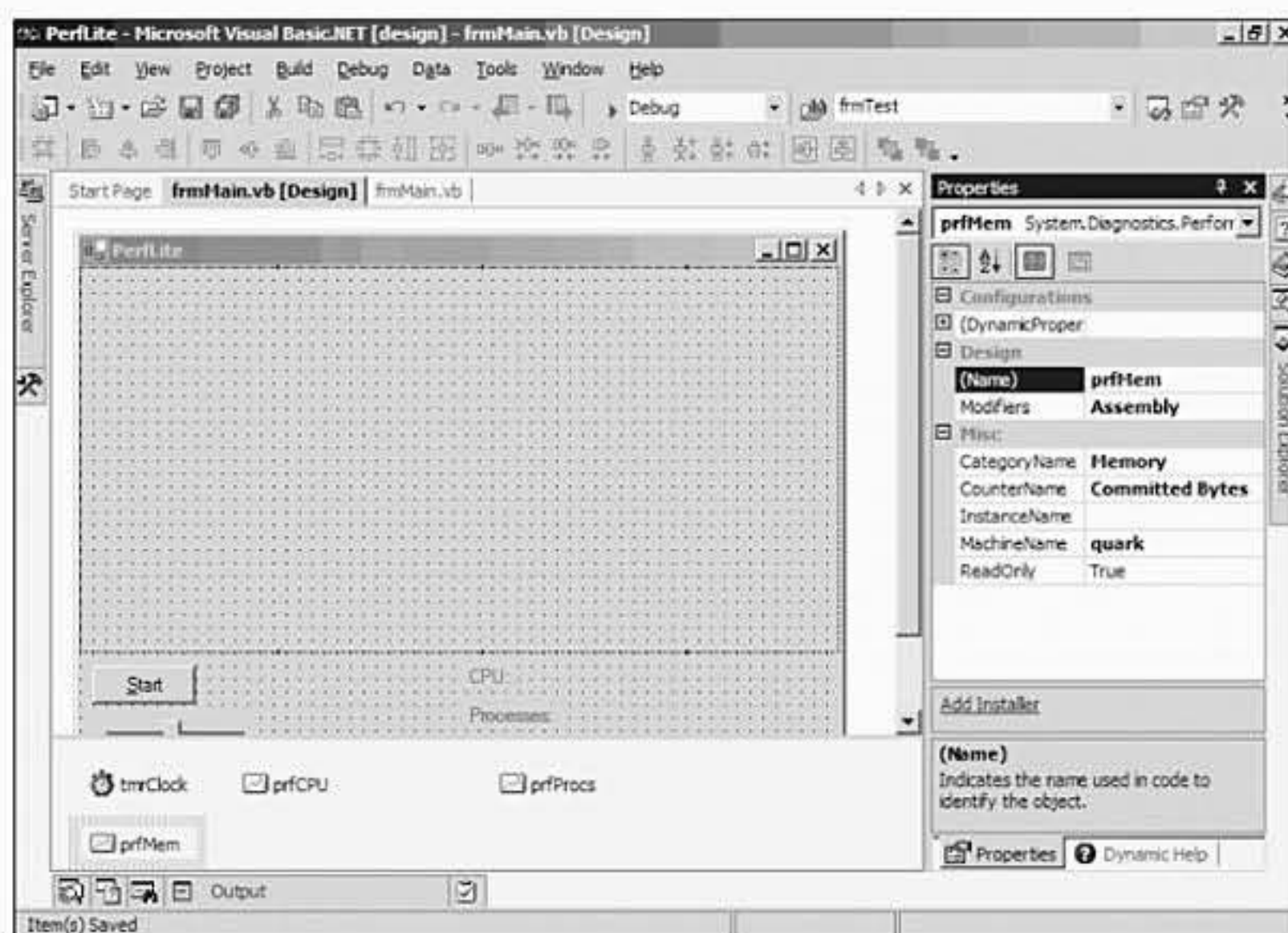
próximo aos três contadores de desempenho. Configure as propriedades desse novo objeto conforme descrito na Tabela 13.8. O formulário final deve se parecer com o mostrado na Figura 13.16.

TABELA 13.8 Propriedades do Objeto Event Log

Propriedade	Valor
Name	logApp
Source	PerfLite

FIGURA 13.16

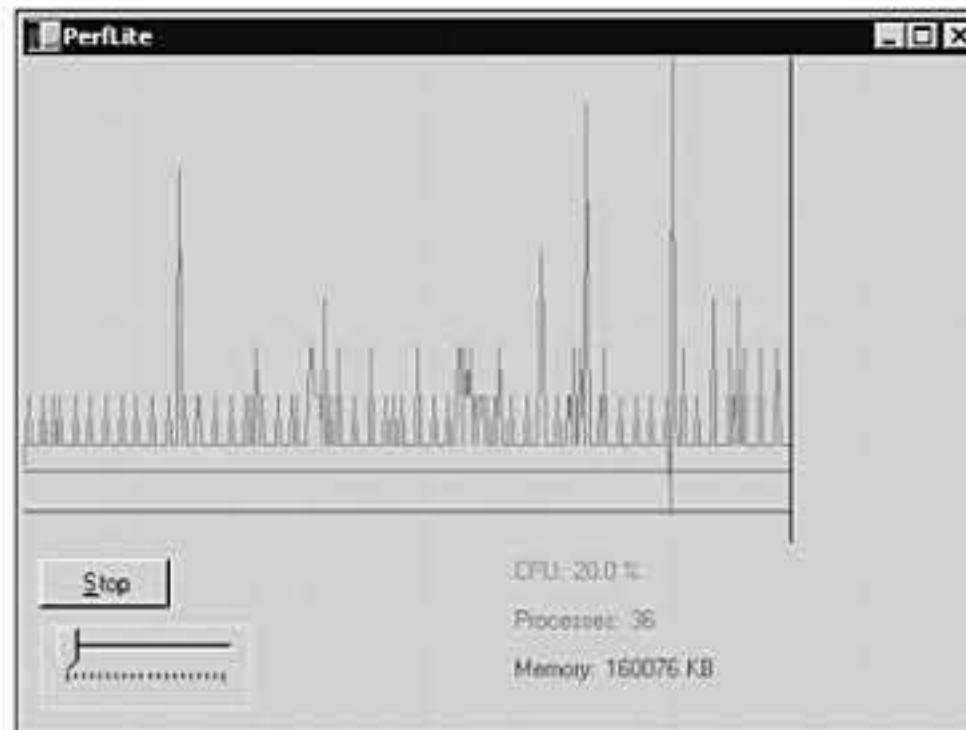
O PerfLite, pronto para a codificação.



Agora que a interface com o usuário está pronta, é hora de começarmos a adicionar a codificação para fazer nosso programa executar algo. Nesse aplicativo, adicionaremos um código para três dos controles e um procedimento para auxiliar. O resultado final mostrará uma linha delimitadora se movendo pela extensão do painel, com três linhas coloridas acompanhando-a atrás (veja a Figura 13.17).

Para que a linha delimitadora seja exibida periodicamente, usaremos o evento Timer do controle de mesmo nome. O código está na Listagem 13.2.

FIGURA 13.17
O exemplo sendo executado.

**CÓDIGO****LISTAGEM 13.2** Código do Controle Timer

```

1 Private Sub tmrClock_Tick(ByVal sender As System.Object, _
2     ByVal e As System.EventArgs) Handles tmrClock.Tick
3     Dim sngCPU As Single = prfCPU.NextValue() / 100
4     Dim sngProcs As Single = prfProcs.NextValue()
5     Dim sngMem As Single = prfMem.NextValue / 1024
6     'desenhe o painel delimitador
7     DrawSweep(sngCPU, sngProcs, sngMem)
8     'atualize os títulos
9     lblCPU.Text = "CPU: " & sngCPU.ToString("p")
10    lblProc.Text = "Processes: " & sngProcs
11    lblMem.Text = "Memory: " & sngMem.ToString("f0") & "KB"
12 End Sub

```

ANÁLISE

O código desse procedimento tem como sua preocupação básica duas finalidades: converter os valores dos contadores de desempenho em variáveis com o tipo de dado simples e atualizar os controles Label. As linhas 3 a 5 recuperam os valores de cada contador de desempenho. Observe que estamos atribuindo os valores às variáveis enquanto as criamos. Em seguida, passamos esses valores para a rotina DrawSweep (linha 7 – veja a Listagem 13.3) para a exibição. Para concluir, cada valor é formatado e exibido em cada um dos controles Label (linhas 9 a 11). Dois itens que parecem estranhos são as chamadas ToString nas linhas 9 e 11. O método ToString permite que um formato opcional seja aplicado quando o número é convertido em uma string. A linha 9 formata o valor como um percentual, enquanto a linha 11 o formata como um número com uma quantidade fixa de casas decimais, e essa quantidade igual a 0.

CÓDIGO**LISTAGEM 13.3** Desenhando a Linha Delimitadora

```

13 Private Sub DrawSweep(ByVal CPU As Single, _
14     ByVal Processes As Single, _
15     ByVal Memory As Single)
16     Dim oGrafix As Graphics = pnlSweep.CreateGraphics()

```


CÓDIGO**LISTAGEM 13.3** Desenhando a Linha Delimitadora (*continuação*)

```

17      Dim sngHeight As Single = pnlSweep.Height
18      'para os pontos
19      Dim sngCPUY As Single
20      Dim sngProcsY As Single
21      Dim sngMemY As Single
22      'apague a linha delimitadora anterior
23      oGrafix.DrawLine(penBack, m_sngX, 0, m_sngX, sngHeight)
24      'desenhe os pontos dos dados
25      sngCPUY = sngHeight - (CPU * sngHeight) - 1
26      oGrafix.DrawLine(penCPU, _
27          m_sngX - increment, m_sngCPUY, m_sngX, sngCPUY)
28      m_sngCPUY = sngCPUY
29      sngProcsY = sngHeight - Processes
30      oGrafix.DrawLine(penProcs, _
31          m_sngX - increment, m_sngProcsY, m_sngX, sngProcsY)
32      m_sngProcsY = sngProcsY
33      'o número 10000 é para dar a memória um valor adequado em minha máquina
34      '    você pode precisar alterá-lo se a linha referente à memória
35      '    não for exibida corretamente
36      sngMemY = sngHeight - (Memory / 10000)
37      oGrafix.DrawLine(penMem, _
38          m_sngX - INCREMENT, m_sngMemY, m_sngX, sngMemY)
39      m_sngMemY = sngMemY
40      'aumente x
41      m_sngX += increment
42      If m_sngX > pnlSweep.Width Then
43          'reinicialize para voltar ao ponto de origem
44          m_sngX = 0
45          'e limpe a superfície de exibição
46          oGrafix.Clear(SystemColors.Control)
47      End If
48      'desenhe a linha nova
49      oGrafix.DrawLine(penFore, m_sngX, 0, m_sngX, sngHeight)
50  End Sub

```

ANÁLISE

Como você já deve ter deduzido, essa rotina é o ponto central do aplicativo. Não se assuste com a extensão ou as chamadas gráficas – é uma rotina bem simples.

Começamos recuperando o objeto Graphics do controle Panel (linha 16). Todos os elementos gráficos do Visual Basic .NET são criados em um objeto Graphics. Qualquer controle que possa ser desenhado (como os controles Image e Panel ou os formulários) expõe esse objeto Graphics

com um método `CreateGraphics`. Por sua vez, ele expõe outros objetos e métodos (e também o controle do contêiner) para que se possa desenhar nele.

Antes de discutirmos o código propriamente dito, iremos determinar o que se pretende fazer. O resultado final será uma linha vertical que se move (horizontalmente) pela extensão do formulário. Enquanto se move, ela é seguida por três linhas coloridas, cada uma representando um dos três contadores de desempenho. Quando a linha (delimitadora) vertical chegar ao limite do formulário, deverá ser iniciada mais uma vez no lado esquerdo dele, sendo esse limpo para a nova exibição. No fim deveremos ter algo parecido com o monitor das batidas de um coração (sem o ruído 'ping'). Portanto, precisaremos de um código para desenhar a linha vertical (e movê-la pela extensão do formulário) e as três linhas de desempenho.

Para mover a linha delimitadora, precisamos desenhar uma outra linha (com a cor do plano de fundo) para apagar a anterior, passar um pouco para cima e iniciar uma nova exibição. Na Listagem 13.3, a linha 23 do código apaga a linha que já havia sido exibida, e uma nova é iniciada e movida das linhas 25 a 41. A razão pela qual isso se torna tão extenso é que também precisamos testar se estaremos movendo a linha em direção ao lado direito do formulário. Se isso estiver ocorrendo (a instrução `If` na linha 42), limparemos o formulário e voltaremos ao lado esquerdo para começar uma nova exibição.

As linhas do contador de desempenho também são desenhadas nessa rotina. A diferença existente nessas chamadas é que também precisamos determinar onde desenhar a linha e lembrar do valor anterior para que possamos conectar a linha nova a já exibida. Usando o primeiro contador de desempenho como exemplo (linhas 25 a 28), isso se resume a três etapas:

1. Decida qual deve ser o novo valor de *Y* (a altura) para a nova linha. A linha 25 faz isso para o contador da CPU determinando que percentual da altura total será ocupado pelo valor referente à CPU (lembre-se, o contador da CPU é um percentual).
2. Desenhe a linha, conectando a anterior à recém-criada. As linhas 26 e 27 fazem isso desenhando uma linha com a cor definida para a CPU com base nos valores anteriores de *X* e *Y* para iniciar os novos. O novo valor de *X* é determinado pelo incremento (15 unidades) do valor anterior, enquanto o novo valor de *Y* foi calculado na etapa 1.
3. Armazene o valor recém-calculado de *Y* para que seja empregado no próximo laço (linha 28). Ele será usado na criação de uma nova linha.

As rotinas para o desenho das linhas dos outros dois contadores de desempenho (linhas 29 a 35 e 36 a 39) são semelhantes – diferindo apenas na maneira como o novo valor de *Y* é calculado.

CÓDIGO**LISTAGEM 13.4** Ativando e Desativando a Linha Delimitadora

```
51 Private Sub cmdSweep_Click(ByVal sender As Object, _  
52     ByVal e As System.EventArgs) Handles cmdSweep.Click  
53     'Alterne entre o texto e o timer no botão (ativado ou desativado)  
54     If cmdSweep.Text = "&Start" Then
```


CÓDIGO**LISTAGEM 13.4** Ativando e Desativando a Linha Delimitadora
(*continuação*)

```

55         cmdSweep.Text = "&Stop"
56         tmrClock.Enabled = True
57     Else
58         cmdSweep.Text = "&Start"
59         tmrClock.Enabled = False
60     End If
61 End Sub

```

ANÁLISE

A finalidade dessa rotina é permitir que o usuário inicie e interrompa a exibição. Portanto, ela é relativamente simples. Com base no texto do botão, alternaremos entre ele e nosso timer. Isto é, se o timer estiver ativado, o interromperemos. Se não estiver, o ativaremos. Poderíamos usar outra variável para registrar esse status, mas, em vez disso, empregaremos o texto do botão. Se a propriedade Text dele for igual a &Start (o valor inicial), iniciaremos o timer e alteraremos Text para &Stop (observe que os dois valores possuem a mesma chave de acesso, a letra S). Como alternativa, interromperemos o timer e configuraremos o texto de volta com &Start.

A seguir, precisamos permitir que o usuário altere a velocidade da linha delimitadora. Isso é feito na Listagem 13.5.

CÓDIGO**LISTAGEM 13.5** Configurando a Velocidade

```

62 Private Sub trkSpeed_Scroll(ByVal sender As Object, _
63     ByVal e As System.EventArgs) Handles trkSpeed.Scroll
64     Dim iValue As Integer
65     iValue = CInt(trkSpeed.Value)
66     'configure o intervalo de tempo com o valor que escolheu
67     tmrClock.Interval = iValue * 100 'ms
68 End Sub

```

ANÁLISE

O código para configurar a velocidade é bem básico. As linhas 64 e 65 apenas se certificam de que o valor que usamos (a configuração atual para a velocidade) tenha sido um inteiro. Em seguida, utilizamos a velocidade para ajustar o intervalo definido para o timer (linha 67). Intervalos menores significam que a linha delimitadora será exibida mais rapidamente.

Para concluir, há algumas variáveis de que precisaremos em todo o aplicativo. Elas armazenarão as coordenadas atuais X e Y, usadas quando desenharmos, e as canetas empregadas na criação das linhas. Adicione o código da Listagem 13.6 ao formulário, exatamente abaixo da região marcada como código gerado pelo Windows Form Designer e antes de qualquer outro trecho de seu código.

CÓDIGO**LISTAGEM 13.6** Variáveis no Nível do Formulário

```

69 Dim m_sngX As Single
70 Dim m_sngY As Single
71
72 Dim m_sngCPUY As Single
73 Dim m_sngProcsY As Single
74 Dim m_sngMemY As Single
75
76 Dim penCPU As New System.Drawing.Pen(Color.Red)
77 Dim penProcs As New System.Drawing.Pen(Color.Green)
78 Dim penMem As New System.Drawing.Pen(Color.Blue)
79 Dim penFore As New System.Drawing.Pen(SystemColors.WindowText)
80 Dim penBack As New System.Drawing.Pen(SystemColors.Control)
81
82 Const INCREMENT As Single = 1

```

ANÁLISE

As variáveis das linhas 69 a 74 são usadas para armazenar as posições atuais da caneta quando não estiverem desenhando as linhas. Uma variável armazena a posição atual de X, e as outras as posições atuais de Y para cada contador de desempenho. As canetas são empregadas no desenho de cada linha de desempenho e da linha delimitadora. Por fim, a constante da linha 82 é utilizada para registrar o espaço definido para o avanço da linha delimitadora enquanto ela se move pelo formulário.

Depois que o código estiver estruturado, compile o programa e execute-o. Dê um clique no botão Start, e observe por alguns segundos. Você começará a ver a barra delimitadora se mover da esquerda para a direita no formulário, sendo seguida por três linhas – uma para a CPU (percentual de utilização), outra para os processos (programas em execução) e mais uma para a memória (em uso). O texto no botão deve alterar-se para Stop. Altere a barra de registro (configure-a como maior ou menor) para observar como ela afeta a exibição; tente dar um clique no botão Stop para ver se ele se altera novamente para Start. Você pode tentar adicionar alguns outros contadores à exibição como um exercício.

Resumo

Os sistemas operacionais mais modernos como o Windows possuem vários serviços. Esses serviços fornecem recursos que se estendem ao sistema operacional, adicionando funcionalidades, como bancos de dados ou registros, que enriquecem as capacidades centrais. O Server Explorer permite que você visualize ou altere o estado desses serviços em seu próprio computador de desenvolvimento ou em outros sem ter de sair do IDE do Visual Basic .NET. Além disso, ele simplifica o desenvolvimento de códigos que usem muitos dos serviços disponíveis.

Na próxima lição, começaremos nossa saga na descoberta da verdadeira natureza orientada a objetos do Visual Basic .NET aprendendo o que significa ser ‘realmente’ orientado a objetos.

P&R

P Tenho alguns itens em meu Server Explorer que não foram mencionados (ou não tenho alguns que foram mencionados). O que são eles?

R É possível que outras empresas criem componentes para ‘associar’ ao Server Explorer. Isso permitiria ao desenvolvedor trabalhar com esses serviços diretamente, tal como se pode fazer com os serviços comuns. Por outro lado, se o servidor com o qual você se conectar não fornecer um determinado serviço, ele poderá não aparecer na lista.

P Se os dados do Server Explorer forem alterados, isso alterará o banco de dados?

R Sim, por favor, tenha cuidado.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Cite duas seções do Server Explorer que informariam a você que programas estão em execução em seu computador.
2. O que é um serviço?
3. O que seria exibido se o código a seguir fosse adicionado a um programa e executado?

```
Dim prfCPU As New PerformanceCounter("processor", __  
    "%Processor Time", "_total")  
Console.WriteLine(prfCPU.NextValue())
```

Exercícios

1. Use o Data Connections para se conectar a um banco de dados disponível e pesquisar as tabelas e outros itens cujo acesso for concedido (lembre-se de não fazer nenhuma alteração, a menos que você realmente precise).
2. Pesquise o Server Explorer para descobrir o que está disponível em seu computador, assim como os servidores que você costuma usar para desenvolvimento.

SEMANA 2

DIA 14

Introdução à Programação Orientada a Objetos

Na versão .NET, o Visual Basic foi reescrito a partir do zero. Uma das principais razões disso foi para adicionar recursos orientados a objetos profundamente integrados. Esta lição abordará:

- Uma visão geral da programação orientada a objetos (POO).
- Conceitos importantes da POO.
- Como incorporar a POO nos sistemas de sua autoria.

Além desses tópicos, no final da lição, você aprenderá técnicas da POO para ajudá-lo a construir aplicativos.

Visão Geral da Programação Orientada a Objetos

A programação orientada a objetos (POO) não é uma tecnologia ou linguagem específica; é uma maneira de projetar e construir aplicativos. Para torná-la ainda mais abstrata, você poderia pensar nela como um modo de considerar o projeto e o desenvolvimento de aplicativos. A POO possui um histórico acadêmico e não tentarei fornecer uma data exata na qual foi concebida pela primeira vez, mas ela foi adotada de maneira ampla pela indústria de softwares nos anos 80. Agora quase sempre presume-se que a POO está sendo usada, mas com frequência o termo é empregado com pouca compreensão do que realmente significa.

Na essência da POO está o conceito de *objeto*, um bloco da programação que combina informações específicas e um conjunto de comportamentos relacionados. Na POO, os cenários são considerados em termos desses objetos em vez da abordagem linear passo a passo que era (e ainda é) usada na maioria dos programas de computador. Os objetos em geral são empregados para descrever entidades, que podem ser reais (um veículo) ou abstratas (um voo reservado em uma companhia aérea nacional). Essas entidades possuem atributos, como a cor do veículo ou a data de saída do voo, que descrevem o objeto. Esses objetos também apresentam certas ações que podem ser executadas neles, como ‘Vender’ para o carro e ‘Cancelar’ para o voo. Na POO, as entidades se tornam objetos, os atributos são conhecidos como *propriedades* (que você pode configurar ou recuperar) e as ações são chamadas de *métodos*.

NOVO TERMO

Um *objeto* é uma representação de uma entidade real ou abstrata junto às propriedades dessa entidade e as ações relacionadas a ela que podem ser executadas.

Comparando a Programação Orientada a Objetos com a Linear

A diferença entre a programação linear ou procedural e a POO pode parecer simples, mas é um conceito complexo em geral mal compreendido. Mostrarei a você duas descrições referentes a um processo de cálculo do pagamento de uma hipoteca. Um é descrito em etapas lineares (também chamado de *programação procedural*) e o outro a partir de um ponto de vista orientado a objetos:

O processo linear segue estas etapas:

1. Obter a quantia principal.
2. Obter a taxa anual de juros.
3. Obter o período em anos.
4. Calcular a taxa de juros por mês.
5. Calcular a quantidade de pagamentos (anos * 12).
6. Calcular os pagamentos.

O processo orientado a objetos inclui estas etapas:

1. Criar uma nova hipoteca.
2. Configurar as propriedades referentes à quantia principal, à taxa de juros e ao período da hipoteca.
3. Recuperar a propriedade do pagamento.

Em código, as duas abordagens também são diferentes. Considere os dois blocos de código mostrados nas Listagens 14.1 e 14.2, que poderiam ser executados em um formulário Windows ou da Web para calcular os pagamentos da hipoteca.

LISTAGEM 14.1 Abordagem procedural para um Programa de Cálculo de Hipoteca

```
1  'Estilo linear
2  Private Sub btnCalc_Click(ByVal sender As System.Object, _
3  ByVal e As System.EventArgs) Handles btnCalc.Click
4      Dim iYears As Integer
5      Dim iMonths As Integer
6      Dim dblInterestRate As Double
7      Dim curPrincipal As Decimal
8      Dim curPayment As Decimal
9
10     iYears = CInt(txtPayments.Text)
11     iMonths = iYears * 12
12     dblInterestRate = CType(txtInterest.Text, Double)
13     curPrincipal = CType(txtPrincipal.Text, Decimal)
14
15     'divida a taxa de juros por 12 para obter a taxa mensal
16     dblInterestRate /= 12
17
18     curPayment = curPrincipal * _
19         ((1 - (1 + dblInterestRate)) _
20         / (1 - ((1 + dblInterestRate) ^ iMonths)) _
21         + dblInterestRate)
22     lblMonthlyPayment.Text = curPayment.ToString()
23 End Sub
```

LISTAGEM 14.2 Abordagem Orientada a Objetos para um Programa de Cálculo de Hipoteca

```
1  'Estilo orientado a objetos
2  Private Sub btnCalc_Click(ByVal sender As System.Object, _
3  ByVal e As System.EventArgs) Handles btnCalc.Click
4      Dim iYears As Integer
5      Dim dblInterestRate As Double
6      Dim curPrincipal As Decimal
7      Dim curPayment As Decimal
8
9      iYears = CInt(txtPayments.Text)
10     dblInterestRate = CType(txtInterest.Text, Double)
11     curPrincipal = CType(txtPrincipal.Text, Decimal)
12
```

LISTAGEM 14.2 Abordagem Orientada a Objetos para um Programa de Cálculo de Hipoteca (*continuação*)

```
13      Dim objMort As New Mortgage()  
14  
15      objMort.AnnualInterestRate = dblInterestRate  
16      objMort.NumberOfYears = iYears  
17      objMort.Principal = curPrincipal  
18  
19      lblMonthlyPayment.Text = objMort.PaymentAmount.ToString  
20  End Sub
```

A Listagem 14.2 cria uma instância do objeto Mortgage (hipoteca) para fazer seus cálculos, mas as operações matemáticas subjacentes são as mesmas. É bom ressaltar que já que o objeto Mortgage ainda não foi definido, você não poderá executar o código da Listagem 14.2. Embora um resultado final semelhante pudesse ser obtido se escrevêssemos uma função com o código procedural, como `CalculateMortgagePayment()`, isso ainda não seria uma POO. Só com o uso do objeto Mortgage, que combina as informações sobre a hipoteca com o código para processá-las, estaremos realmente usando a POO.

Esse estilo de programação não é novo para você, se esteve acompanhando a primeira metade deste livro; quase tudo no Visual Basic .NET é construído dessa maneira. Considere o código usado para trabalhar com controles em um formulário Windows: `txtPayment.Text = lblResult.Text`. Os dois controles são objetos, com uma propriedade `Text`, exatamente como o objeto Mortgage do exemplo anterior. Empregar os objetos fornecidos pelo Visual Basic .NET e pelo .NET Framework não fará com que seu código seja orientado a objetos, embora muitas pessoas possam lhe dizer que sim. É possível utilizar os objetos e ainda programar com um estilo procedural passo a passo.

Usando os Objetos na Organização do Código

Na programação procedural, sem o uso das técnicas da POO, é comum dividir os recursos em sub-rotinas ou funções que poderão, em seguida, ser chamadas de qualquer parte do programa. Também não é raro agrupar essas sub-rotinas (em geral com outras sub-rotinas relacionadas) em módulos, DLLs ou outra forma de estrutura de código. A seguir, você poderia adicionar esses grupos de sub-rotinas a seu programa como unidades individuais e chamar qualquer uma delas quando necessário. Portanto, todas as suas funções matemáticas poderiam ser agrupadas em um arquivo `math.dll`, e todas as funções de strings em `string.dll`, criando um sistema organizado.

Essa forma de codificação ainda seria procedural em vez de orientada a objetos. Isso não reduz seu valor como uma boa maneira de organizar o código, mas começa a ficar muito semelhante à POO com chamadas como `Math.Sqrt(x)` em execução. Ao programar na plataforma .NET, você precisará usar objetos para agrupar os procedimentos desse modo. Incluir 20 procedimentos diferentes como método de um objeto é uma técnica útil, mas lembre-se de que esse es-

tilo de agrupamento não é POO; é simplesmente como o código é organizado na plataforma .NET. Seu código só será orientado a objetos quando empregar objetos para representar as entidades e conceitos que compõem seu aplicativo.

Na plataforma .NET, os objetos que não representam uma entidade real ou abstrata, existindo apenas para agrupar código, em geral designam a si mesmos e a todos os seus métodos como estáticos ou compartilhados. Você aprenderá mais sobre esses tipos de objetos na próxima lição (Dia 15, “Criando Objetos no Visual Basic .NET”), mas apenas lembre-se de que há uma maneira específica de construir essas bibliotecas procedurais de código.

Conceitos Importantes na POO

Alguns termos foram definidos para serem usados na POO, e eles são relativamente universais para todas as tecnologias e linguagens de programação. Nesta seção, explicarei os termos mais comuns e fornecerei um exemplo de como cada um funciona.

Classes, Objetos e Instâncias

Os primeiros termos que precisam de discussão são aqueles usados em qualquer material que envolva a POO: classe, objeto e instância. Esses conceitos serão a base para seu trabalho com a POO, e como tal devem ficar claros antes de prosseguirmos.

A *classe* é um modelo para um objeto; ela descreve a estrutura básica do objeto. Muitas analogias diferentes são usadas para descrever o relacionamento entre a classe, o objeto e a instância. Uma das mais comuns é considerar esses conceitos em termos de casas e da construção delas. Empregando uma analogia desse tipo, a classe seria o projeto para a casa, e a casa propriamente dita seria um objeto. Muitas casas poderiam ser criadas com base no mesmo projeto, e muitos objetos podem ser gerados com base na mesma classe. Cada objeto criado a partir de uma classe é chamado *instância* dessa classe. Para examinar isso nos termos do Visual Basic, considere a Listagem 14.3.

LISTAGEM 14.3 Inserindo Várias Classes em um Único Arquivo

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         x = New myFirstClass()
5     End Sub
6 End Module
7
8 Public Class myFirstClass
9     ' <a definição da classe e o código entram aqui>
10 End Class
```


A instrução `Public Class myFirstClass` (linha 8) define uma classe nova; um projeto para os objetos. A linha 3 cria uma variável do tipo `myFirstClass`, que pode conter uma instância de `myFirstClass`. A linha 4 gera uma nova instância de `myFirstClass` (criando um objeto) e a atribui a variável `x`. Agora `x` se refere a uma instância da classe `myFirstClass`. Outro exemplo do uso de objetos é mostrado na Listagem 14.4.

LISTAGEM 14.4 Variáveis e Instâncias Não São a Mesma Coisa

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         Dim y As myFirstClass
5         x = New myFirstClass()
6         y = New myFirstClass()
7     End Sub
8 End Module
```

Na Listagem 14.4, duas variáveis do tipo `myFirstClass` são declaradas (nas linhas 3 e 4) e, em seguida, duas novas instâncias de `myFirstClass` são criadas (linhas 5 e 6). Cada instância é referenciada por uma variável. A variável `x` contém uma referência a um objeto diferente de `y`, mas os dois objetos são instâncias de `myFirstClass`. Esse conceito, instâncias comparadas a variáveis, será examinado mais uma vez em outro exemplo na Listagem 14.5.

LISTAGEM 14.5 Instâncias Comparadas com Variáveis

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         Dim y As myFirstClass
5         x = New myFirstClass()
6         y = x
7     End Sub
8 End Module
```

A Listagem 14.5 torna tudo um pouco mais confuso. Agora duas variáveis foram declaradas, ambas podendo se referir a uma instância da classe `myFirstClass`. Em seguida, uma nova instância dessa classe é criada (por meio da palavra-chave `New`) e uma referência ao novo objeto é inserida em `x`. A seguir, `y` é atribuída a `x`, e o resultado é que `y` agora contém uma referência à mesma instância de `myFirstClass` que `x` referencia. Só há um objeto e, portanto, uma área da memória, mas duas variáveis que se referem (ou apontam) a ele. Isso será demonstrado na próxima seção quando você aprender as propriedades.

Propriedades

As classes podem definir as propriedades que cada instância delas deve ter. A *propriedade* é um valor que existe como parte de um objeto, e você pode recuperar ou configurá-la por meio desse objeto.

A Listagem 14.6 adiciona uma propriedade à definição de `myFirstClass` por uma variável pública. Essa não é a única maneira de adicionar uma propriedade, mas abordaremos os outros métodos no Dia 15.

LISTAGEM 14.6 Adicionando uma Propriedade à Definição de `myFirstClass`

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         Dim y As myFirstClass
5         x = New myFirstClass()
6         y = New myFirstClass()
7         x.Name = "Fred"
8         y.Name = "Joe"
9         Console.WriteLine("x.Name = {0}", x.Name)
10        Console.WriteLine("y.Name = {0}", y.Name)
11        Console.ReadLine()
12    End Sub
13 End Module
14
15 Public Class myFirstClass
16     Public Name As String
17 End Class
```

Como antes, duas variáveis são declaradas, `x` e `y`, duas novas instâncias de `myFirstClass` são criadas, e são inseridas referências nas duas variáveis. A seguir, nas linhas 7 e 8, um valor é inserido na propriedade `Name` de cada instância de `myFirstClass`, e essa propriedade é acessada por meio das variáveis que fazem referência ao objeto. As instruções `Console.WriteLine` (linhas 9 e 10) produzem a saída a seguir:

```
x.Name = Fred
y.Name = Joe
```

Cada instância de uma classe possui sua própria parte na memória associada a ela e, portanto, os valores da propriedade são armazenados independentemente com cada instância. Retornando à analogia da casa, todas as instâncias de uma casa são criadas do mesmo projeto, mas cada uma pode ser pintada de uma cor diferente. A cor da casa é uma propriedade, e seu valor não é determinado pelo projeto; é um atributo de cada instância. A Listagem 14.7 deve ajudar a ilustrar como as propriedades estão associadas a uma instância individual de uma classe.

LISTAGEM 14.7 As Variáveis *x* e *y* Apontam para a Mesma Instância de um Objeto

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         Dim y As myFirstClass
5         x = New myFirstClass()
6         y = x
7         x.Name = "Fred"
8         y.Name = "Joe"
9         Console.WriteLine("x.Name = {0}", x.Name)
10        Console.WriteLine("y.Name = {0}", y.Name)
11        Console.ReadLine()
12    End Sub
13 End Module
```

A única diferença entre as Listagens 14.7 e 14.6 está na linha 6, em que é atribuída à variável *y* uma referência à *x* em vez de a uma nova instância de *myFirstClass*. Essa diferença significa que nas linhas 7 e 8, o código trabalha com a propriedade *Name* da mesma instância, e a saída dessa rotina será

```
x.Name = Joe
y.Name = Joe
```

As duas instruções *WriteLine* produzem o mesmo valor para a propriedade porque tanto *x* quanto *y* se referem à mesma instância do objeto. Esse conceito específico, em que múltiplas variáveis se referem ao mesmo objeto, é outro tópico importante, porém confuso. Verifique a seção 6.1 do Visual Basic Language Specification (parte da documentação da plataforma .NET; pesquise em “Tipos de Valores e Tipos de Referência” para encontrá-la) para obter mais detalhes.

Na plataforma .NET, as propriedades podem ser apenas de leitura, apenas de gravação ou de leitura/gravação, permitindo a existência de um controle sobre como esses valores são acessados. Você aprenderá a controlar o acesso às propriedades de suas próprias classes no Dia 15.

Métodos

Além das propriedades, as classes também podem ter comportamentos ou ações associados a elas. Conhecidas como *métodos*, essas ações permitem que uma classe possua alguma lógica incorporada além das informações que são armazenadas por suas propriedades. Uma casa não é o melhor exemplo quando se quer falar sobre métodos, mas uma classe casa criada para ser usada por um aplicativo de uma imobiliária pode possuir métodos que listem a casa na Web (*myHouse.List()*), vendam a casa (*myHouse.Sell()*) ou até exibam uma propaganda (*myHouse.PrintBrochure("Printer1")*). Esses métodos são exatamente como qualquer outro procedimento; eles podem aceitar parâmetros quando você os chama e retornar valores também.

Na Listagem 14.8, um método (DoSomething) foi adicionado a `myFirstClass` por meio de um procedimento `Public (Sub)` na definição da classe.

LISTAGEM 14.8 Adicionado um Método à `myFirstClass`

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         x = New myFirstClass()
5         x.DoSomething()
6     End Sub
7 End Module
8
9 Public Class myFirstClass
10     Public Name As String
11     Public Sub DoSomething()
12         'o código entraria aqui
13     End Sub
14 End Class
```

Em seguida, esse método poderia ser chamado de qualquer instância da classe. Exatamente como com as propriedades, no entanto, o método é definido na classe, porém chamado na instância. Isso significa que se o método usar alguma das propriedades (privadas ou públicas) da classe, então, estará utilizando as informações armazenadas na instância em questão.

Herança

Um conceito essencial na POO, mas que não era fácil de implementar no Visual Basic antes da versão .NET, é a herança.

NOVO TERMO

Herança é o conceito pelo qual você pode basear uma classe em outra, em geral para alterar ou adicionar algo nos recursos disponíveis da classe original.

Acabei de almoçar, portanto acho que deixarei de lado a analogia da casa e tentarei confundi-lo com uma baseada em sanduíches. Considere o hambúrguer básico: dois pães com um pedaço de carne colocado entre eles.

Quando o restaurante decide começar a vender cheeseburgers, ele não é criado do nada. O cozinheiro é informado para ‘fazer um hambúrguer, porém adicionando uma fatia de queijo’. Assim, uma nova classe de sanduíche é criada, o cheeseburger, que herda características da classe hambúrguer. Para estender essa analogia até que fiquemos saturados de sanduíches, o restaurante poderia agora decidir oferecer um hambúrguer de luxo, que herdasse características do cheeseburger e incluísse alface e tomate. Em cada caso, a classe básica é usada como origem.

Dessa maneira, você pode criar uma hierarquia de objetos, em que a classe utilizada pode estar no final de várias camadas de herança. É possível ter quantas classes desejar com base em uma única classe, portanto o restaurante poderia oferecer um sanduíche vegetariano ou de galinha, os dois derivados da classe hambúrguer e sobrepostos ao tipo de recheio utilizado anteriormente.

NOVO TERMO

Sobreposição é um termo usado para indicar que uma classe-filha fornece sua própria implementação de um recurso da classe básica, quando comparado à simples adição de novos recursos.

No Dia 15, você aprenderá a criar seus próprios objetos no Visual Basic .NET usando a sobreposição, a herança e outros recursos, mas, por enquanto, fornecerei um exemplo simples do emprego da herança na Listagem 14.9.

LISTAGEM 14.9 Usando a Herança para Criar Hierarquias de Objetos

```
1 Module AllAboutObjects
2     Public Sub Main()
3
4         Dim mySnippet As CodeSnippet
5         mySnippet = New CodeSnippet()
6
7         With mySnippet
8             .Contents = "txtEntry.Text = lblInfo.Text"
9             .Author = "Joe"
10            .Language = "VB.NET"
11            .Purpose = "Inserir o texto do título na caixa de texto"
12        End With
13    End Sub
14 End Module
15
16 Public Class Snippet
17     Public Contents As String
18     Public Author As String
19 End Class
20
21 Public Class CodeSnippet
22     Inherits Snippet
23     Public Language As String
24     Public Purpose As String
25 End Class
```

Como você pode ver, CodeSnippet, que é derivada de Snippet, adiciona duas novas propriedades. Quando for criada uma nova instância da classe CodeSnippet (linha 5), ela terá as propriedades das duas classes disponíveis.

Em geral, a classe nova precisa sobrepor algum recurso da classe básica, apenas porque a funcionalidade básica não leva em consideração as alterações posteriores feitas na classe. Considere um método, chamado `GetSnippet()` que existe na classe `Snippet` e foi criado para retornar a classe `Snippet` completa como uma `String`, como mostrado na Listagem 14.10.

LISTAGEM 14.10 Inclua Todos os Recursos de Sua Nova Classe

```
1 Public Class Snippet
2     Public Contents As String
3     Public Author As String
4     Public Function GetSnippet() As String
5         Dim sTmp As String
6         sTmp = "Author: " & Author _
7             & System.Environment.NewLine _
8             & Contents
9         Return sTmp
10    End Function
11 End Class
```

Essa função exibe apenas as duas propriedades da classe básica, portanto se você não as sobrepuer à nova classe, suas duas novas propriedades não serão manipuladas. A classe `CodeSnippet` da Listagem 14.11 manipula isso sobrepondo a função `GetSnippet` da classe básica.

LISTAGEM 14.11 Sobreponha uma Função para Personalizá-la a Fim de Que Manipule os Novos Recursos de Sua Classe

```
1 Module AllAboutObjects
2     Public Sub Main()
3
4         Dim mySnippet As CodeSnippet
5         mySnippet = New CodeSnippet()
6
7         With mySnippet
8             .Contents = "txtEntry.Text = lblInfo.Text"
9             .Author = "Joe"
10            .Language = "VB.NET"
11            .Purpose = "Inserir o texto do título na caixa de texto"
12        End With
13    End Sub
14 End Module
15
16 Public Class Snippet
17     Public Contents As String
18     Public Author As String
19     Public Overridable Function GetSnippet() As String
```

LISTAGEM 14.11 Sobreponha uma Função para Personalizá-la a Fim de Que Manipule os Novos Recursos de Sua Classe (*continuação*)

```

20      Dim sTmp As String
21      sTmp = "Author: " & Author _
22            & System.Environment.NewLine _
23            & Contents
24      Return sTmp
25  End Function
26 End Class
27
28 Public Class CodeSnippet
29     Inherits Snippet
30     Public Language As String
31     Public Purpose As String
32
33     Public Overrides Function GetSnippet() As String
34         Dim sTmp As String
35         sTmp = MyBase.GetSnippet() & _
36               System.Environment.NewLine & _
37               "Language: " & Language & _
38               System.Environment.NewLine & _
39               "Purpose: " & Purpose
40         Return sTmp
41     End Function
42 End Class
  
```

Para que essa função seja sobreposta, ela deve ser marcada com `Overridable` na classe básica (linha 19) e, em seguida, você precisa fornecer uma nova implementação na classe derivada (marcada com `Overrides`, linha 33). Para obter o resultado da função da classe básica como parte da implementação dessa classe, a palavra-chave especial `MyBase` pode ser usada (linha 35) para acessar as propriedades e métodos da classe básica. Outra palavra-chave especial, que não foi empregada nesse exemplo, é `Me`. Ela se refere ao objeto atual, portanto `Me.Name` que foi utilizada no procedimento `GetSnippet` referia-se à propriedade `Name` da instância atual dessa classe.

Um recurso útil da herança é que as classes que são derivadas de uma classe básica específica podem ser usadas como se fossem essa classe básica. As instâncias da nova classe podem ser inseridas em variáveis que utilizem o tipo de dado da classe anterior e passadas como parâmetros para os procedimentos que esperem o tipo de dado anterior, como mostra a Listagem 14.12.

LISTAGEM 14.12 Use uma Classe Herdada em Qualquer Local Que Der Suporte a Sua Classe Básica

```

1 Option Strict On
2 Option Explicit On
  
```


LISTAGEM 14.12 Use uma Classe Herdada em Qualquer Local Que Der Suporte a Sua Classe Básica (*continuação*)

```
3
4 Module AllAboutObjects
5     Public Sub Main()
6
7         Dim mySnippet As CodeSnippet
8         mySnippet = New CodeSnippet()
9         With mySnippet
10             .Contents = "txtEntry.Text = lblInfo.Text"
11             .Author = "Joe"
12             .Language = "VB.NET"
13             .Purpose = "Inserir título do texto na caixa de texto"
14         End With
15         PrintSnippet(mySnippet)
16         Console.ReadLine()
17     End Sub
18
19     Public Sub PrintSnippet(ByVal objSnippet As Snippet)
20         Console.WriteLine(objSnippet.GetSnippet())
21     End Sub
22 End Module
```

Quando um objeto é passado como seu tipo básico, como na linha 15 da Listagem 14.12, as propriedades e métodos adicionados pela classe herdada não ficam disponíveis. Portanto, `objSnippet.Language` não seria compilada se você a inserisse na linha 20 já que ela não é uma propriedade válida para a classe `Snippet`. Os métodos sobrepostos ficam disponíveis porque existem na definição da classe básica, mas a implementação da classe derivada será chamada em vez da básica. A Listagem 14.12 produziria o seguinte resultado:

```
Author: Joe
txtEntry.Text = lblInfo.Text
Language: VB.NET
Purpose: Inserir texto do título na caixa de texto
```

Você aprenderá mais sobre a herança e a capacidade de tratar um objeto derivado como se fosse sua classe básica, durante a próxima lição (Dia 15).

Construtores

Outro recurso novo orientado a objetos nesta versão do Visual Basic é a inclusão de construtores para os objetos.

NOVO TERMO

O *construtor* é uma rotina chamada quando a instância de uma classe é gerada, e essa pode especificar que parâmetros fornecer na hora da criação. Esse conceito proporciona uma maneira de inicializar seu objeto na hora da criação.

No Visual Basic .NET, o construtor é representado por um procedimento `New` na definição da classe, e (exatamente como qualquer procedimento) você pode ter a quantidade de versões sobrepostas que quiser dele, fornecendo várias maneiras pelas quais o objeto pode ser criado. Na Listagem 14.13, diversos construtores são adicionados às classes `Snippet` e `CodeSnippet`, permitindo que esses objetos sejam inicializados de muitas maneiras.

LISTAGEM 14.13 Os Construtores Criam e Inicializam Seu Objeto

```
1 Option Strict On
2 Option Explicit On
3
4 Module AllAboutObjects
5     Public Sub Main()
6         Dim mySnippet As CodeSnippet
7         mySnippet = _
8             New CodeSnippet("txtEntry.Text = lblInfo.Text", _
9                 "Joe", "VB.NET", "Inserir texto do título na caixa de texto")
10        PrintSnippet(mySnippet)
11        Console.ReadLine()
12    End Sub
13    Public Sub PrintSnippet(ByVal objSnippet As Snippet)
14        Console.WriteLine(objSnippet.GetSnippet())
15    End Sub
16 End Module
17
18 Public Class Snippet
19     Public Contents As String
20     Public Author As String
21
22     Public Sub New()
23
24     End Sub
25
26     Public Sub New(ByVal Contents As String)
27         Me.Contents = Contents
28     End Sub
29
30     Public Sub New(ByVal Contents As String, _
31         ByVal Author As String)
32         Me.Contents = Contents
33         Me.Author = Author
34     End Sub
```


LISTAGEM 14.13 Os Construtores Criam e Inicializam Seu Objeto (*continuação*)

```
35
36     Public Overridable Function GetSnippet() As String
37         Dim sTmp As String
38         sTmp = "Author: " & Author _
39             & System.Environment.NewLine _
40             & Contents
41         Return sTmp
42     End Function
43 End Class
44
45 Public Class CodeSnippet
46     Inherits Snippet
47     Public Language As String
48     Public Purpose As String
49
50     Public Sub New()
51         MyBase.New()
52     End Sub
53
54     Public Sub New(ByVal Contents As String)
55         MyBase.New(Contents)
56     End Sub
57
58     Public Sub New(ByVal Contents As String, _
59         ByVal Author As String)
60         Me.New(Contents)
61         Me.Author = Author
62     End Sub
63
64     Public Sub New(ByVal Contents As String, _
65         ByVal Author As String, _
66         ByVal Language As String)
67         Me.New(Contents, Author)
68         Me.Language = Language
69     End Sub
70
71     Public Sub New(ByVal Contents As String, _
72         ByVal Author As String, _
73         ByVal Language As String, _
74         ByVal Purpose As String)
75         Me.New(Contents, Author, Language)
76         Me.Purpose = Purpose
77     End Sub
78
79     Public Overrides Function GetSnippet() As String
```


LISTAGEM 14.13 Os Construtores Criam e Inicializam Seu Objeto (*continuação*)

```
80      Dim sTmp As String
81      sTmp = MyBase.GetSnippet() & _
82          System.Environment.NewLine & _
83          "Language: " & Language & _
84          System.Environment.NewLine & _
85          "Purpose: " & Purpose
86      Return sTmp
87  End Function
88 End Class
```

As linhas 22 a 34 são os construtores para a classe Snippet, e ilustram a maneira-padrão de inicializar um objeto. As linhas 50 a 77 são os construtores para a classe CodeSnippet, e contêm alguns artifícios em como a inicializam. Primeiro, você pode chamar o construtor da classe básica por meio da palavra-chave MyBase (linhas 51 e 55), o que assegura que os códigos sejam executados na rotina da classe básica e, em seguida, também sejam processados para sua classe derivada. O segundo artifício, empregado em todos os outros construtores, é chamar seus construtores mais simples dentre os mais complexos. Utilizar esse estilo de codificação evita a repetição de códigos; a inicialização genérica é inserida no primeiro construtor, e qualquer código adicional, requerido para a manipulação de cada parâmetro novo, pode ser incluído apenas no primeiro construtor no qual todos os parâmetros novos aparecem.

Projetando um Aplicativo com o Uso da POO

A programação orientada a objetos não é um conceito técnico e não está relacionada apenas à programação de um sistema. A POO está envolvida na criação de um sistema com base em um projeto conceitual. Você deve decidir se vai projetar um sistema usando as técnicas orientadas a objetos, e a implementação não está envolvida nessa decisão. Antes da plataforma .NET, o Visual Basic não possuía recursos para permitir a herança, a sobreposição de procedimentos e outros conceitos da POO, mas os programadores ainda podiam projetar um aplicativo empregando esses conceitos e, em seguida, implementá-los utilizando várias etapas ou artifícios para conseguir o que o Visual Basic .NET possui embutido atualmente. Depois que você tiver sua fase de abrangência/escopo e o trabalho de análise dos requisitos concluídos em um projeto, passará para o estágio conceitual. Durante a fase de projeto, você começará a incorporar objetos em seus aplicativos. Siga estas etapas para criar esse projeto:

1. Identifique as entidades.
2. Determine as propriedades e métodos.
3. Crie hierarquias de objetos.
4. Modele seus objetos.

Cada etapa será abordada individualmente à medida que você percorrer o processo de descrição de um jogo de vinte-e-um usando objetos. Seria útil que você compreendesse esse jogo de cartas, antes que lesse estas seções. Se não souber nada sobre ele, consulte as regras procurando “Black-Jack” na Encarta (<http://encarta.msn.com>).

Identificando os Objetos

A primeira tarefa é descrever os requisitos de seu sistema, que é muito mais trabalhoso do que pode parecer. Nesse estágio, você precisa esboçar o sistema, aplicativo ou componente proposto com detalhes suficientes para determinar toda as entidades principais. Um artifício que gosto de usar para iniciar esse processo é empregar todos os substantivos que aparecem nas descrições de meu sistema. Utilizando o vinte-e-um como exemplo, dê uma olhada nessa breve e provavelmente incompleta descrição do jogo (recorra ao site da Encarta para obter informações mais completas).

O vinte-e-um é um jogo de cartas com dois ou mais participantes, um deles dando as cartas. O objetivo do jogo é fazer com que o valor de suas cartas chegue o mais próximo possível de 21 sem ultrapassar essa marca, e ele é jogado entre os participantes incluindo também o jogador que distribui as cartas. Em caso de empate (quando o jogador que dá as cartas e os outros obtêm o mesmo valor próximo a 21), quem ganha é aquele que distribui as cartas. Antes de cada rodada, os jogadores apostam uma quantia em dinheiro em sua probabilidade de ter uma mão melhor que a do jogador que dá as cartas. Um ou mais baralhos são embaralhados e, em seguida, o jogador que dá as cartas distribui uma para cada participante e para ele mesmo.

Nesse ponto, os participantes fazem suas apostas depois de conhecer somente uma carta. Em seguida, o jogador que dá as cartas distribui mais uma para cada participante e para ele mesmo. A rodada termina se alguém chegar a um total igual a 21 com apenas duas cartas. Se ninguém conseguir esse resultado, chamado de ‘natural’, então, o jogador que dá as cartas as distribui em ordem para cada participante, uma de cada vez, até que o jogador decida parar ao atingir 21 ou quando chegar perto. Se ele exceder 21 com as novas cartas que recebeu, perderá imediatamente essa rodada. Todos os participantes passam por esse processo, pedindo mais cartas até que decidam parar ou que excedam o valor.

Depois que todos os jogadores decidirem parar de receber cartas ou tiverem sido eliminados da rodada, o jogador que dá as cartas continua a recebê-las, decidindo se pára ou permanece, segundo as regras fixadas. Depois que ele e os outros jogadores já tiverem recebido todas as cartas, ele terá de pagar a todos os jogadores com resultados mais altos que os dele, e os que tiverem resultados mais baixos terão de pagar a ele. Portanto, já que o jogador que dá as cartas sempre ganha em um empate, se ele terminar com 21, os outros pagarão a ele.

É claro que essa é uma descrição breve, não abordando quando as apostas são dobradas, divididas e outras características do jogo, mas está detalhada o bastante para atender a essa discussão. Considerando esse exemplo como a parte de uma declaração necessária a seu trabalho, você pode começar a procurar as entidades que poderiam ser transformadas em objetos. Lembre-se de

meu conselho sobre os substantivos e tente por sua própria conta criar uma lista de objetos antes de examinar a que consegui.

Percorrer o exemplo apenas procurando os substantivos produzirá mais entidades do que queremos, mas posso fazer uma seleção nessa lista e obter as sugestões a seguir para os objetos:

- Participante
- Jogador que dá as cartas (seria incluído como um participante?)
- Rodada (uma jogada)
- Jogo (o jogo completo)
- Baralho
- Mão (cartas do participante ou do jogador que dá as cartas)
- Carta (uma única carta do jogo)
- Aposto (representando uma aposta do participante)

Dada essa lista, passaríamos para a determinação das propriedades e métodos desses objetos, que em geral ajudam a definir quais deles, se houver algum, estão relacionados.

Determinando as Propriedades e Métodos

Atendo-se apenas a um subconjunto das entidades encontradas, começarei a examinar Participante, Jogador que dá as cartas, Baralho, Mão e Carta. Cada uma delas possui vários atributos que poderão ser propriedades, e diversas ações prováveis que podem se tornar métodos. Um Participante pode ter um Nome, uma Mão atual, uma Posição (já que o jogador que dá as cartas o faz em uma ordem específica entre os participantes, a posição pode ser necessária) e o Total em dinheiro como propriedades. Os métodos do objeto Participante poderiam incluir Permanecer, Parar, Apostar e outros. O Jogador que dá as cartas pode ter o Nome, a Mão atual e o Total em dinheiro também, além dos métodos Permanecer e Parar do objeto Participante. Só dois objetos, e já estamos vendo um certo padrão. Por causa das semelhanças entre Participante e Jogador que dá as cartas, poderia ser melhor que eles fossem o mesmo objeto ou que existisse um relacionamento de herança entre eles.

Perceber esses relacionamentos é uma parte essencial do processo de identificação dos objetos, e pode afetar muito o aplicativo resultante. Se os objetos Jogador que dá as cartas e Participante tivessem sido herdados do mesmo objeto básico, ou Jogador que dá as cartas fosse herdado de Participante, então, você poderia escrever um código que lidasse com as duas entidades apenas tratando-as como instâncias da classe básica.

Continuando a percorrer a lista de objetos, o Baralho teria um conjunto de objetos Carta e uma propriedade Quantidade (para indicar a quantidade de cartas atualmente no baralho, diminuindo quando elas fossem distribuídas). De maneira semelhante, o objeto Mão, representando as cartas que um participante ou o jogador que dá as cartas estivesse segurando no momento, também teria um conjunto de objetos Carta e uma propriedade Quantidade. Você também poderia querer

uma propriedade Total no objeto Carta, para indicar o valor total das cartas, mas a regra ‘o ás pode valer 1 ou 11’ tornaria essa propriedade difícil de implementar e seria necessário um trabalho maior no projeto. Os métodos do objeto Baralho incluiriam Adicionar/Remover para o conjunto de cartas e um método Embaralhar para misturar esse mesmo conjunto. O método Adicionar/Remover seria aplicado ao objeto Mão, mas Embaralhar não seria tão útil.

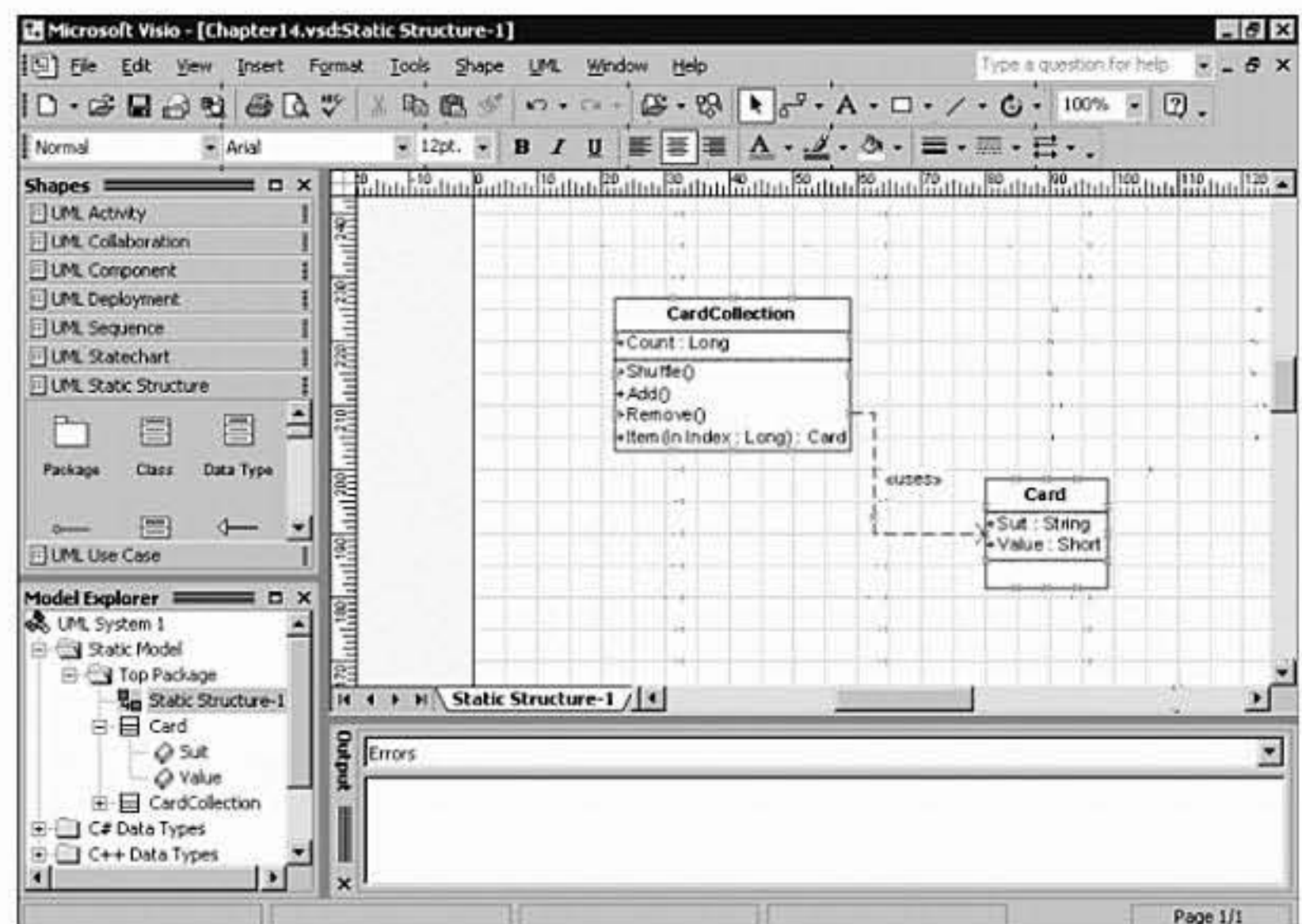
Os objetos Baralho e Mão não são apenas semelhantes, eles são quase idênticos. Em vez de pensar na herança, uma opção seria transformar os dois na mesma classe, porém contendo quantidades diferentes de cartas. Como alternativa, você poderia criar Mão e Baralho como classes derivadas de uma classe ConjuntoCartas mais genérica. A classe Carta é bem fácil de definir: naipe (Copas, Espadas, Ouros e Paus) e valor (2-10, K, Q, J, A), e não uma necessidade real de que seja herdada de qualquer outro objeto.

Modelando Seus Objetos

Ao percorrer o processo de determinação dos objetos na análise de requisitos, seria útil que você criasse diagramas ou modelos para ilustrar os objetos e seus relacionamentos. O Visual Studio .NET Architect Edition fornece ferramentas de modelagem, e elas também estão disponíveis em uma cópia autônoma do Microsoft Visio. Por enquanto, para demonstrar a aparência de um diagrama de modelagem de objetos, veja a Figura 14.1.

FIGURA 14.1

A modelagem UML (Universal Modeling Language) atende a uma finalidade relacionada ao projeto, fornecendo a você uma visualização do estado atual do modelo de objeto de seu aplicativo, e representa a documentação do funcionamento interno de seu sistema.



O diagrama mostrado na Figura 14.1 foi criado com o uso de um estilo de modelagem chamado UML ou Universal Modeling Language. Esse tipo de diagrama pode ser gerado empregando-se qualquer uma das diversas ferramentas existentes, incluindo o Visio e o Rational Rose. A vantagem de modelar seu aplicativo é fornecer um diagrama de seu projeto inicial dos objetos e utili-

z -lo quando fizer altera  es com o passar do tempo. Em um ambiente que envolva uma equipe grande, os recursos de gera  o de c digos da maioria das ferramentas de modelagem tamb m podem ser  teis, permitindo que voc  crie o c digo da estrutura de qualquer conjunto de objetos de seus diagramas. Ainda com rela  o   gera  o do c digo, o Visio e o Rational Rose (e provavelmente outros) d o suporte   engenharia reversa de c digos existentes, e podem usar essa tecnologia para produzir um diagrama de objetos a partir desses c digos. A engenharia reversa   uma  tima maneira de documentar um sistema quando n o se emprega a modelagem desde o in cio.

Resumo

Os objetos s o uma das principais partes do Visual Basic .NET, mas uma abordagem orientada a objetos   diferente de uma procedural comum. Deve-se focar primeiro o projeto, na tentativa de visualizar os conceitos e entidades de um sistema como objetos, antes de se preocupar com os detalhes da implementa  o do conceito desses objetos. No Dia 15, voc  aprender  mais sobre a cria  o de objetos usando os diversos recursos da POO discutidos em toda esta li  o.

P&R

P Ouvi falar que a C# e a C++ fornecem melhores recursos de programa  o orientada a objetos do que o Visual Basic .NET. Isso   verdade?

R Antes da plataforma .NET, isso era verdade com rela  o a C++; ela fornece recursos de POO (como a heran a) que o Visual Basic n o possu a. Agora, todas as linguagens .NET apresentam a mesma funcionalidade de POO porque ela est  sendo fornecida pela Common Language Specification (com a qual todas as linguagens .NET devem estar em conformidade) em vez de ser espec fica de cada linguagem. Isso significa que na plataforma .NET, o Visual Basic .NET   apenas uma linguagem t o boa para a POO quanto a C++ ou a C#.

P A POO   melhor ou mais r pida do que o estilo de programa  o ‘linear’ ou comum?

R N o necessariamente. A POO n o produz resultados diferentes em um aplicativo;   simplesmente outra abordagem para projetar e construir esse aplicativo. Em geral, acredita-se que a POO gera sistemas que possuem uma manuten  o mais simples e s o mais f ceis de expandir, mas isso decerto estar  mais relacionado com o c digo criado por voc .

Workshop

O Workshop foi planejado para ajud -lo a antecipar poss veis d vidas, revisar o que j  aprendeu e come ar a pensar em como colocar seu conhecimento em pr tica. As respostas do teste est o no Ap ndice A, “Respostas dos Testes/Exerc cios”.

Teste

1. Que palavra-chave permite que você se refira à classe básica quando escreve um código dentro de uma classe derivada?
2. Que nome é dado quando são criadas várias versões do mesmo método com conjuntos diferentes de parâmetros?
3. Qual seria o resultado deste código?

```
1 Module AllAboutObjects
2     Sub Main()
3         Dim x As myFirstClass
4         Dim y As myFirstClass
5         x = New myFirstClass()
6         y = x
7         y.Name = "Fred"
8         x.Name = y.Name
9         Console.WriteLine("x.Name = {0}", x.Name)
10        Console.WriteLine("y.Name = {0}", y.Name)
11        Console.ReadLine()
12    End Sub
13 End Module
14
15 Public Class myFirstClass
16     Public Name As String
17 End Class
```

Exercícios

Descreva o que poderia ser um projeto com objetos que registrassem os eventos de um grande estádio. Leve em consideração vários tipos diferentes de eventos e produza uma pequena lista dos objetos que você poderia projetar para essa finalidade.

PÁGINA EM BRANCO

SEMANA 2

Revisão

Na Semana 2, abrangemos uma grande área, indo da introdução a uma boa mostra das classes do .NET Framework aos detalhes sobre os recursos avançados orientados a objetos do Visual Basic .NET. No geral, você aprendeu o bastante para criar um aplicativo completo de produção com uma interface com o usuário, conexões com bancos de dados e até algumas conexões com as funções do servidor como o registro de eventos.

O Dia 8 apresentou várias das partes mais úteis do .NET Framework e também o conduziu pelo processo de exploração do Framework por sua própria conta. Esse conhecimento permitirá que você encontre qualquer recurso de que precisar no Framework; uma aptidão inestimável quando tiver de fazer algo que este livro não teve tempo de abordar.

Embora os aplicativos da Web sejam a tendência atual, os aplicativos Windows ainda são comuns, e o Dia 9 mostrou-lhe como construí-los no Visual Basic .NET. Essa abordagem dos formulários Windows introduziu a razão original pela qual a palavra ‘Visual’ foi adicionada ao Visual Basic – a criação de interfaces com o usuário com operações de arrastar e soltar. Usando os controles, propriedades e alguma codificação, já é possível criar seu aplicativo Windows.

É claro que você também tinha de aprender a construir uma interface com o usuário com base na Web, e o Dia 10 foi exatamente a lição de que precisava. Sua introdução aos formulários da Web mostrou como criar um aplicativo da Web que pudesse ser programado com o mesmo modelo dirigido a eventos que os aplicativos Windows usaram durante anos.

Nos Dias 11 e 12, você estudou os bancos de dados – alguma teoria, porém muito mais prática. Essas lições mostraram como as classes `System.Data` do Framework podem ser usadas para estabelecer uma conexão com um banco de dados, recuperar as informações necessárias e, em seguida, manipular esses dados quando preciso. No decorrer do caminho, conseguimos saber mais do que talvez quiséssemos sobre o gosto musical de pelo menos um dos autores (e não disse qual).

No Dia 13, abordamos outro dos principais recursos novos do IDE do Visual Studio, o Server Explorer, e como você pode usá-lo no Visual Basic .NET para se conectar aos servidores, visualizar e criar contadores de desempenho, e trabalhar com o Event Logs do Windows XP/2000/NT. Agora os aplicativos que você criar poderão agir como programas no nível empresarial, fornecendo informações sobre desempenho, erros e de auditoria por meio das ferramentas com as quais os administradores de sistemas já estão familiarizados.

Na última lição da Semana 2, o Dia 14, você retornou ao assunto dos objetos no Visual Basic .NET, que começamos lá atrás no Dia 7. Nessa lição avançada, abordamos a estrutura e os recursos dos objetos e como usá-los para construir aplicativos. Mais informações sobre os objetos ainda estão por vir. Leia o Dia 15 para obter detalhes sobre como é possível desenvolver seus objetos e sistemas com base em objetos no Visual Basic .NET.

No final da Semana 2, você já tem uma compreensão substancial do Visual Basic .NET, o que permite a criação de sistemas que vão além de simples exemplos na área dos aplicativos reais. O projeto de bônus da Semana 2, uma biblioteca de registros de CDs/DVDs on-line, colocará à prova seu conhecimento aperfeiçoado na construção de um aplicativo com uma estrutura interna mais complicada e uma interface com o usuário tanto no Windows quanto na Web. Verifique todos os três projetos de bônus na Web em http://www.sampublishing.com/detail_sams.cfm?item=0672320665.

SEMANA 3

Visão Geral

Durante esta última semana, você examinará vários tópicos avançados para complementar seu conhecimento sobre o Visual Basic e o .NET Framework. Primeiro, você aprenderá a criar suas classes de biblioteca e componentes no Dia 15. O Dia 16 irá mostrar-lhe os detalhes da geração de interfaces com o usuário mais complexas para aplicativos Windows por meio das classes dos formulários Windows. Essa lição inclui uma discussão sobre menus, aplicativos com interfaces de documentos múltiplos (MDI – multiple document interface) e vários dos controles mais complicados fornecidos pelas classes dos formulários Windows.

Como você viu no decorrer dos primeiros 14 capítulos, o .NET Framework é amplo e complexo, mas essa estrutura fornece todos os recursos que serão necessários em seus aplicativos. Por essa razão, o Dia 17 explora mais áreas do Framework, incluindo o trabalho com a funcionalidade de manipulação de arquivos e figuras da plataforma .NET.

Os Dias 18 e 19 fornecem as informações sobre finalização de que você precisa para concluir seu aplicativo e instalá-lo na máquina que será o destino final, seja ele um servidor Web ou os microcomputadores de vários usuários. Uma discussão sobre a redistribuição do .NET Framework e sobre os requisitos básicos do cliente também está incluída nesses capítulos.

O Dia 20 é uma introdução à XML, a linguagem comum para os dados que fluem por toda a plataforma .NET. Esse capítulo aborda tanto a própria XML quanto as partes da plataforma .NET (System.Xml) que facilitam ao seu código a leitura, gravação e manipulação de informações XML.

O último capítulo deste livro, o Dia 21, aborda os serviços da Web, códigos que podem ser chamados pela Web por meio de tecnologias-padrão da indústria como o SOAP, a XML e o HTTP.

O lugar que os serviços da Web ocupam em seus sistemas foi discutido no Dia 5, mas o material do Dia 21 mostrará a você como criar e testar um serviço da Web simples usando o Visual Basic .NET.

15

16

17

18

19

20

21

Para concluir, há o terceiro e último projeto de bônus do livro, Hangman, um programa complexo que usa os conceitos da XML do Dia 20, requer algum trabalho com arquivos e figuras (Dia 7) e envolve a criação de uma interface com o usuário avançada (Dia 16). Esse programa também deve ser divertido. Você escreverá um jogo que poderá mostrar para sua família, amigos e colegas, e conseguir que eles se divirtam com um programa que você criou!

SEMANA 3

DIA 15

Criando Objetos no Visual Basic .NET

Nas lições anteriores, você aprendeu a usar objetos e projetá-los, mas em algum momento pode querer ou precisará começar a construir os seus próprios. Esta lição abordará:

- A definição de objetos no Visual Basic .NET.
- O uso de seus próprios objetos em seu código.
- O desenvolvimento de suas classes em montagens e o uso dessas montagens a partir de outro aplicativo.

Para ilustrar esses conceitos, esta lição incluirá a criação de vários exemplos de classes e a geração de uma montagem para armazená-las.

Criando Objetos

Os objetos não são criados diretamente no Visual Basic .NET; em vez disso você construirá classes que se tornarão a definição do objeto. Em seguida, os objetos serão gerados a partir da classe, transformando-se em uma instância dela. Portanto, as etapas para definir seus próprios objetos começam com a criação de uma nova classe do Visual Basic .NET.

Declarando uma Classe Nova no Visual Basic .NET

Criar uma classe nova é tão fácil quanto digitar `Public Class Test` na janela de edição de seu código (dentro de um arquivo do Visual Basic .NET) e pressionar Enter. Pressupondo que você não tenha tentado isso dentro de um procedimento real (sub-rotina ou função), poderia dizer que foi bem-sucedido porque o Visual Basic .NET adiciona uma linha `End Class` automaticamente. Essas instruções declaram uma nova classe e indicam o início e o fim da definição da classe. Dentro dessa classe são definidos os métodos, as propriedades e os eventos, mas antes é preciso compreender a declaração da classe.

Escopo

Digitar `Public Class Test` cria uma classe vazia com escopo público (`Public`), o que significa que qualquer pessoa com uma referência a esse projeto pode gerar uma instância de `Test` apenas executando `myTest = New Test()`. Isso em geral é o que se quer, portanto você provavelmente criará classes de escopo público, mas essa não é a sua única opção. Também é possível criar classes privadas, usando a palavra-chave `Private` para produzir uma classe que só possa ser utilizada por outro código em um escopo no mesmo nível da declaração, o que quase sempre significa dentro do mesmo arquivo. Se tivéssemos declarado sua classe dentro da declaração de outra (aninhando classes), então, qualquer código da classe externa poderia acessar o novo objeto. A palavra-chave `Private` é empregada quando não queremos que uma classe seja criada fora de nosso bloco de código atual. Para concluir, a outra opção de escopo que mencionarei é `Friend`. As classes, as variáveis e os procedimentos declarados com o escopo `Friend` podem ser acessados por qualquer código dentro do mesmo programa (montagem, serviço da Web, aplicativo Windows e assim por diante), mas de maneira nenhuma por algum outro código.

Herança

Como parte da declaração da classe, você pode especificar apenas uma classe básica para a nova herdar características dela, se a classe for derivada de alguma outra. A Listagem 15.1 mostra um exemplo disso.

LISTAGEM 15.1 Uma Classe Pode Herdar Características de Outra Classe

```
1 Public Class Snippet
2 'definição da classe
3
4 End Class
5
6 Public Class CodeSnippet
7     Inherits Snippet
8 'definição da classe
9 End Class
```

Além disso, por meio da palavra-chave `NotInheritable`, adicionada depois do trecho do escopo na declaração da classe, você também pode especificar que sua classe não permite que características possam ser herdadas. Se especificar essa palavra-chave, então, ninguém poderá criar uma nova classe derivada dessa. Isso é equivalente ao termo 'Sealed', que é usado na C# para indicar que essa nova classe não permite que características sejam herdadas dela e é empregado na definição de muitos objetos da plataforma .NET como `System.String`.

O contrário da criação de classes lacradas é a geração das abstratas, que devem ser usadas como classes básicas e não podem ser criadas diretamente. Se você definir os objetos básicos que não quer que sejam empregados sozinhos, apenas como parte de sua hierarquia de objetos, então, a palavra-chave `MustInherit` será perfeita. Uma utilização comum dessa palavra-chave é quando a classe básica não é funcional isoladamente, e um ou mais métodos-chave precisam implementar classes derivadas antes de surgir algo útil.

A Listagem 15.2 ilustra o uso da palavra-chave `MustInherit`. Ela será útil para demonstrar que você pode empregar a palavra-chave `MustOverride` como parte da declaração de métodos, a fim de forçar a inclusão deles em todas as versões herdadas dessa classe. Se utilizar `MustOverride`, não terá de fornecer nenhuma implementação desse método ou propriedade.

LISTAGEM 15.2 `MustInherit` e `MustOverride` Permitem Que Você Especifique Que uma Classe Não Pode Ser Usada Diretamente

```
1 Public MustInherit Class BaseShape
2     Public MustOverride ReadOnly Property NumSides() As Integer
3     Public MustOverride Sub Draw()
4 End Class
5
6 Public Class Square
7     Inherits BaseShape
8     Public Overrides Sub Draw()
9         'desenhe um quadrado
10    End Sub
11    Public Overrides ReadOnly Property NumSides() As Integer
12        Get
13            Return 4
14        End Get
15    End Property
16 End Class
```

ANÁLISE

A classe `Shape` é inútil sozinha; está apenas definindo o modelo geral de todos os objetos que a compartilham como classe básica. Ela é uma classe básica útil porque permite a codificação de várias rotinas a partir de uma única classe e automaticamente dá suporte a todas as classes derivadas. Definindo essa classe como `MustInherit`, você assegurará que os programadores não possam contrariar suas intenções e criar uma instância de `Shape` diretamente.

É provável que ninguém tente gerar uma instância de Shape – e se alguém o fizesse, não haveria muitos problemas –, mas suas intenções se tornaram claras evitando a utilização incorreta. Marcar o método Draw como MustOverride indica que, quando essa classe for herdada desse método, ele terá de estar presente na nova classe derivada. Mais uma vez, essa é uma maneira de indicar sua intenção empregando os recursos do Visual Basic .NET em vez de confiar que os outros programadores sempre ajam da maneira correta.

Adicionando Propriedades

Há duas maneiras principais de adicionar propriedades a uma classe: criando variáveis públicas (que se tornam propriedades diretamente) ou rotinas completas de propriedades. Criar propriedades por meio das variáveis públicas funcionará, mas você não terá nenhum controle sobre essas variáveis expostas e não haverá maneira de executar os códigos quando seus valores forem alterados. A Listagem 15.3 apresenta esse método de definição de propriedades, que é adequado quando se está apenas testando as classes no Visual Basic .NET.

LISTAGEM 15.3 As Variáveis Públicas Se Tornam Propriedades Quando São Declaradas Dentro de uma Classe

```
1 Public Class Person
2     Public Name As String
3     Public FirstName As String
4     Public SecondName As String
5 End Class
```

Criar propriedades por meio de uma rotina própria é um método mais complexo que proporcionará um controle total sobre a propriedade. Primeiro, determine o nome e o tipo de dado da propriedade que você quer adicionar e, em seguida, digite esta linha dentro dos limites de uma classe (fazendo as substituições apropriadas para o nome e o tipo de dado da propriedade):

```
Public Property Name() As String
```

Quando você pressionar Enter ao terminar essa linha, o Visual Basic .NET criará a estrutura completa de um procedimento de propriedade:

```
Public Property Name() As String
    Get

    End Get
    Set(ByVal Value As String)

    End Set
End Property
```

As duas partes desse procedimento, Get e Set, manipulam a atribuição de um valor de propriedade e o recebimento de um valor (objTest.Name = "Fred"), respectivamente. Já que você não usa-

rá uma variável pública quando empregar esse tipo de rotina de propriedade, provavelmente precisará de uma variável privada declarada para conter o valor da propriedade. Adicioná-la, como `m_sName`, à classe de seu exemplo permitirá a criação do código que será inserido na rotina `Get/Set` de sua propriedade. A variável da classe privada usa o prefixo `m_` (que representa *variável membro*) para indicar que ela é uma representação interna de uma propriedade exposta. Nessa forma mais simples, o código inserido na rotina da propriedade (veja a Listagem 15.4) será usado apenas para transferir valores para fora e para dentro da variável interna.

LISTAGEM 15.4 Os Procedimentos de Propriedade São uma Alternativa ao Uso de Variáveis Públicas

```
1 Public Class Test
2     Private m_sName As String
3
4     Public Property Name() As String
5         Get
6             Return m_sName
7         End Get
8         Set(ByVal Value As String)
9             m_sName = Value
10        End Set
11    End Property
12 End Class
```

A rotina de propriedade da Listagem 15.4 é pública (`Public`), o que provavelmente será preferido, mas você também pode defini-la como privada (`Private`), restringindo, dessa maneira, seu uso fora da própria classe.

As verdadeiras vantagens das rotinas de propriedade sobre simplesmente usar uma variável pública estão no fato de que você pode criar propriedades de leitura ou de gravação e executar a validação dos dados em cada tentativa de gravação em uma propriedade.

Criando Propriedades de Leitura e de Gravação

As rotinas de propriedade são definidas como de leitura ou de gravação na declaração da propriedade. Se a propriedade for declarada sem palavras-chave adicionais, como na Listagem 15.4, então, ela será de leitura/gravação, e você precisará incluir tanto o trecho `Get` quanto `Set` na rotina. Como alternativa, é possível especificar `Public ReadOnly Property` para criar uma propriedade que não possa ser lida. As propriedades apenas de leitura, embora não sejam tão comuns quanto as de leitura/gravação, podem ser úteis em várias situações. A Listagem 15.5 mostra como três propriedades, duas das quais de leitura/gravação e uma apenas de leitura, podem ser empregadas para produzir uma classe útil.

LISTAGEM 15.5 As Propriedades Podem Ser de Leitura/Gravação, Apenas de Leitura ou Apenas de Gravação

```
1 Option Strict On
2 Option Explicit On
3
4 Module AllAboutObjects
5     Public Sub Main()
6
7         Dim objSample As New Person()
8
9         objSample.FirstName = "Fred "
10        objSample.LastName = "Jones "
11        Console.WriteLine(objSample.DisplayName)
12        Console.ReadLine()
13    End Sub
14 End Module
15 Public Class Person
16     Private m_sName As String
17     Private m_sFirstName As String
18     Private m_sLastName As String
19
20     Public ReadOnly Property DisplayName() As String
21         Get
22             Return String.Format("{0} {1}", m_sFirstName, m_sLastName)
23         End Get
24     End Property
25
26     Public Property FirstName() As String
27         Get
28             Return m_sFirstName
29         End Get
30         Set(ByVal Value As String)
31             m_sFirstName = Value
32         End Set
33     End Property
34
35     Public Property LastName() As String
36         Get
37             Return m_sLastName
38         End Get
39         Set(ByVal Value As String)
40             m_sLastName = Value
41         End Set
42     End Property
43 End Class
```

As propriedades apenas de gravação são um pouco mais confusas; pessoalmente não vejo muitas razões para a existência de uma propriedade que o usuário pode configurar, mas não visualizar. Se ele pode configurá-la, então, já deve saber o valor e, portanto, por que você iria querer impedi-lo de visualizá-la? A única situação que posso imaginar na qual uma propriedade apenas de gravação poderia ser útil é quando lidamos com senhas. Considere uma classe que vai ser usada para estabelecer uma conexão com um banco de dados de back-end e uma de suas propriedades é a senha do banco de dados. Em algumas situações, pode-se querer passar uma instância desse objeto à outra rotina, com suas propriedades já preenchidas, sem expor nenhuma informação de segurança.

Usando Rotinas de Propriedade para Validar Dados

O uso das rotinas de propriedade em vez de variáveis públicas apresenta outro grande benefício; com uma rotina de propriedade, você pode executar um código sempre que o usuário quiser recuperar ou configurar o valor dessa propriedade. Isso permite que muitas coisas sejam feitas, inclusive a validação de dados antes do acesso a seus objetos. A Listagem 15.6, por exemplo, verifica o número de um cartão de crédito antes do acesso a um objeto, certificando-se de que ele é válido e rejeitando a tentativa se não for.

LISTAGEM 15.6 Os Procedimentos de Propriedade Permitem a Validação de Dados

```

1 Option Explicit On
2 Option Strict On
3
4 Public Class CreditCardValidation
5     Public Enum CardTypes
6         ccvVisa
7         ccvMasterCard
8         ccvDiscover
9         ccvAMEX
10        ccvDinersClub
11        ccvEnRoute
12        ccvUndefined
13    End Enum
14
15    Private Const Numbers As String = "0123456789"
16    Private Const sInvalidChecksumError As String _
17        = "O número do cartão de crédito contém um erro " & _
18        "em um ou mais dígitos (Checksum Error)"
19    Private Const sLengthError As String _
20        = "O número do cartão de crédito não tem o tamanho certo" & _
21        "para esse tipo de cartão (Length Error)"
22
23    Private sErrorMsg As String
24    Private mCardType As CardTypes

```

LISTAGEM 15.6 Os Procedimentos de Propriedade Permitem a Validação de Dados
(*continuação*)

```
25 Private sCardNumber As String
26 Private bValid As Boolean
27
28 Public Property CardNumber() As String
29     Get
30         Return sCardNumber
31     End Get
32     Set(ByVal Value As String)
33         bValid = ValidCreditCard(Value)
34         If bValid Then
35             sCardNumber = Value
36         Else
37             Throw New System.ArgumentException(sErrorMsg, _
                                                    "CardNumber")
38         End If
39     End Set
40 End Property
41
42 Private Function CTPrefix(ByVal sCard As String) As CardTypes
43     If CType(Left(sCard, 2), Integer) > 50 AndAlso _
44         CType(Left(sCard, 2), Integer) < 56 Then
45         Return CardTypes.ccvMasterCard
46     ElseIf Left(sCard, 1) = "4" Then
47         Return CardTypes.ccvVisa
48     ElseIf Left(sCard, 4) = "6011" Then
49         Return CardTypes.ccvDiscover
50     ElseIf Left(sCard, 2) = "34" OrElse _
51         Left(sCard, 2) = "37 " Then
52         Return CardTypes.ccvAMEX
53     ElseIf Left(sCard, 2) = "36" Then
54         Return CardTypes.ccvDinersClub
55     Else
56         Return CardTypes.ccvUndefined
57     End If
58 End Function
59
60
61 Private Function Prefix(ByVal sTest As String, _
62     ByVal sArg As String) As Boolean
63     If Left(sArg, Len(sTest)) = sTest Then
64         Prefix = True
65     Else
66         Prefix = False
```


LISTAGEM 15.6 Os Procedimentos de Propriedade Permitem a Validação de Dados
(*continuação*)

```
67      End If
68  End Function
69
70  Private Function ValidCreditCard(ByVal sNumber As String)_
71      As Boolean
72      Dim sTemp As String
73      Dim iTemp As Integer
74      Dim sCreditCard As String
75      Dim iCardLength As Integer
76      Dim Checksum As Integer
77      Dim i As Integer
78
79      sTemp = sNumber
80      sCreditCard = " "
81      Checksum = 0
82
83      For i = 1 To Len(sTemp)
84          If InStr(Numbers, Mid(sTemp, i, 1)) = 0 Then
85              sCreditCard = sCreditCard & Mid(sTemp, i, 1)
86          End If
87      Next
88
89      mCardType = CTPrefix(sCreditCard)
90      sCardNumber = sCreditCard
91      iCardLength = Len(sCreditCard)
92
93      Select Case mCardType
94          Case CardTypes.ccvAMEX
95              If iCardLength < 15 Then
96                  ValidCreditCard = False
97                  sErrorMsg = sLengthError
98              End If
99
100         Case CardTypes.ccvVisa
101             If (iCardLength < 13) AndAlso (iCardLength <> 16) Then
102                 ValidCreditCard = False
103                 sErrorMsg = sLengthError
104             End If
105         Case CardTypes.ccvMasterCard, CardTypes.ccvDiscover
106             If iCardLength <> 16 Then
107                 ValidCreditCard = False
108                 sErrorMsg = sLengthError
109             End If
110     End Select
```


LISTAGEM 15.6 Os Procedimentos de Propriedade Permitem a Validação de Dados
(*continuação*)

```
111
112     sCreditCard = Right("0000000000000000" & sCreditCard, 16)
113
114     For i = 1 To Len(sCreditCard) - 1
115         iTemp = CInt(Mid(sCreditCard, 16 - i, 1))
116         iTemp = iTemp * (1 + (i Mod 2))
117         If iTemp >= 10 Then
118             iTemp = iTemp - 9
119         End If
120         Checksum = Checksum + iTemp
121     Next i
122     Checksum = (10 - (Checksum Mod 10)) Mod 10
123     If Checksum = CInt(Right(sCreditCard, 1)) Then
124         ValidCreditCard = True
125         sErrorMsg = " "
126     Else
127         ValidCreditCard = False
128         sErrorMsg = sInvalidChecksumError
129     End If
130 End Function
131
132 Public Function CardTypeName(ByVal sCardNumber As String) As String
133     Dim sTmp As String
134
135     If ValidCreditCard(sCardNumber) Then
136         Select Case mCardType
137             Case CardTypes.ccvAMEX
138                 sTmp = "American Express"
139             Case CardTypes.ccvVisa
140                 sTmp = "Visa"
141             Case CardTypes.ccvMasterCard
142                 sTmp = "MasterCard"
143             Case CardTypes.ccvDinersClub
144                 sTmp = "Diners Club"
145             Case CardTypes.ccvDiscover
146                 sTmp = "Discover Card"
147             Case CardTypes.ccvEnRoute
148                 sTmp = "enRoute"
149             Case CardTypes.ccvUndefined
150                 sTmp = "Unknown"
151         End Select
152     Else
153         Throw New ArgumentException(sErrorMsg, "Card Number")
```


LISTAGEM 15.6 Os Procedimentos de Propriedade Permitem a Validação de Dados
(*continuação*)

```
154         End If
155         Return sTmp
156     End Function
157 End Class
```

ANÁLISE

Esse código apenas verifica se o número é válido, e não se é mesmo de um cartão de crédito. Esse número também pode se referir a uma conta de cartão de crédito cancelada, expirada ou ainda inválida. O trecho `Set` dessa rotina de propriedade aceita uma string e, em seguida, usa uma soma de verificação para determinar se ela é um número válido de cartão de crédito, somando os dígitos de uma maneira específica para verificar o total no último dígito. Se a string fornecida for inválida, então, uma exceção será lançada. Essa exceção deve ser capturada pelo programa com a configuração da propriedade `CardNumber` exatamente como aparece nesse exemplo.

Em geral, não se devem usar variáveis públicas como propriedades; funcionará, mas você terminará sem um controle e sem opções. O melhor a fazer é empregar sempre as rotinas de propriedade e utilizar copiar/colar para manipular a digitação adicional.

Criando Métodos

Para adicionar métodos a sua classe, você criará procedimentos `Sub` e `Function`; o tipo exato dependendo de sua situação específica. Exatamente como com as classes (veja a seção “Escopo” apresentada no início desta lição), é possível criar métodos que possuam um escopo `Public`, `Private` ou `Friend`. Os métodos declarados com escopo `Public` podem ser acessados por qualquer pessoa por meio de um objeto desse tipo. Os métodos `Private` ficam disponíveis apenas dentro da própria classe, e os métodos `Friend` só podem ser usados por outro código que esteja dentro da mesma montagem ou programa executável. A Listagem 15.7 mostra a mesma classe de exemplos anteriores (com alguns trechos de código não pertinentes removidos, como as rotinas `Property`) com um método `Public` adicionado para estabelecer uma conexão com o Outlook e tentar encontrar um endereço de correio eletrônico para a pessoa em questão.

**NOTA**

Para usar esse código, você precisa incluir uma referência ao Outlook em seu projeto (e precisa do Outlook 2000 ou do XP instalado e funcionando). Para adicionar essa referência a seu projeto, siga estas etapas:

1. Dê um clique com o botão direito do mouse na pasta References de seu projeto no Solution Explorer e dê um clique em Add Reference no menu suspenso.
2. Dê um clique na guia COM e localize Microsoft Outlook na lista.
3. Dê um clique no botão Select e, em seguida, feche a caixa de diálogo dando um clique em OK. Provavelmente você será questionado se quer criar uma montagem de interoperabilidade, e deve dar um clique em Yes para responder a essa pergunta.

LISTAGEM 15.7 Esta Função Recupera Informações no Outlook

```

1 Public Class Person
2 'código não pertinente removido da listagem
3     Public Function LookUpInOutlook() As String
4         Dim objOutlook As New Outlook.Application()
5         Dim objSession As Outlook.NameSpace
6         Dim objPerson As Outlook.Recipient
7         Dim sEmailAddress As String
8
9         objOutlook.Session.Logon(NewSession:=False)
10        objSession = CType(objOutlook.Session, Outlook.NameSpace)
11        objPerson = objSession.CreateRecipient(Me.DisplayName)
12        Try
13            sEmailAddress = String.Format("{0}:{1}", _
14                objPerson.AddressEntry.Type, _
15                objPerson.AddressEntry.Address)
16            Return sEmailAddress
17        Catch objException As Exception
18            'Endereço não encontrado
19            Return " "
20        End Try
21    End Function
22 End Class
23 Module AllAboutObjects
24     Public Sub Main()
25
26         Dim objSample As New Person()
27         objSample.FirstName = "Joel"
28         objSample.LastName = "Semeniuk"
29         Console.WriteLine(objSample.DisplayName)
30         Console.WriteLine(objSample.LookUpInOutlook())

```


LISTAGEM 15.7 Esta Função Recupera Informações no Outlook (*continuação*)

```
31
32     Console.ReadLine()
33 End Sub
34 End Module
```

ANÁLISE

Quando você executar esse código, o Outlook poderá exibir um aviso (veja a Figura 15.1) de que um programa está tentando acessar endereços de correio eletrônico, o que é verdade porque isso é exatamente o que essa sub-rotina faz.

FIGURA 15.1

Os novos recursos de segurança do Outlook o alertarão e permitirão que você impeça o acesso programático a seu endereço de correio eletrônico e informações de contato.



Se você estiver certo de que esse código é a causa do aviso, então, poderá dar um clique em OK se quiser que ele obtenha com sucesso um endereço de correio eletrônico. É preciso fornecer um nome que exista em sua pasta Contatos ou em uma de suas listas de endereços, caso contrário esse código não conseguirá encontrar ninguém, e o endereço não será determinado. A linha 4 cria uma instância nova do Outlook; em seguida, a linha 9 obtém a sessão MAPI atual. A linha 10 captura um objeto Namespace que é essencial no modelo de objetos do Outlook e expõe tudo que é necessário para o acesso. Usar o método CreateRecipient (linha 11) é o equivalente a digitar o valor do nome na caixa Para de uma mensagem de correio eletrônico permitindo que o Outlook determine esse nome. Se CreateRecipient falhar (o usuário pode não ser determinado), então, o bloco Try/Catch que inclui a obtenção do endereço deverá impedir que alguma mensagem de erro seja exibida.

Se você quiser permitir que as classes derivadas sobreponham um método definido em sua classe básica, então, o método básico precisará ser marcado com a palavra-chave `Overridable`, e o da classe derivada terá de especificar a palavra-chave `Overrides`. Na Listagem 15.8, a classe `Mortgage` é definida para manipular os cálculos da hipoteca (essa classe também foi usada no Dia 14, “Introdução à Programação Orientada a Objetos”) e, em seguida, na Listagem 15.9, uma nova classe, `AccelMortgage`, é derivada de `Mortgage`. Nessas listagens, as palavras-chave `Overridable` e `Overrides` são usadas para que a nova classe possa acomodar uma agenda de pagamentos que não seja mensal.

LISTAGEM 15.8 A Herança É uma Boa Maneira de Adicionar Funcionalidade

```
1 Public Class Mortgage
2     'Programa muito simples de cálculo de hipoteca
3     'Configure AnnualInterestRate, NumberOfYears e Principal
4     'Em seguida, recupere a propriedade PaymentAmount
5     Private m_dblInterestRate As Double
6     Private m_iDuration_Years As Integer
7     Private m_curPrincipal As Decimal
8
9     Public Property AnnualInterestRate() As Double
10        Get
11            Return m_dblInterestRate
12        End Get
13        Set(ByVal Value As Double)
14            m_dblInterestRate = Value
15        End Set
16    End Property
17
18    Public Property NumberOfYears() As Integer
19        Get
20            Return m_iDuration_Years
21        End Get
22        Set(ByVal Value As Integer)
23            m_iDuration_Years = Value
24        End Set
25    End Property
26
27    Public Property Principal() As Decimal
28        Get
29            Return m_curPrincipal
30        End Get
31        Set(ByVal Value As Decimal)
32            m_curPrincipal = Value
33        End Set
34    End Property
35
36    Public Overridable Function PaymentAmount() As Decimal
37        Dim iNumPaymentsPerYear As Integer = 12
38        Dim iPayments As Integer
39        Dim dblFractionalInterestRate As Double
40        Dim curPayment As Decimal
41
42        iPayments = m_iDuration_Years * iNumPaymentsPerYear
43        dblFractionalInterestRate = m_dblInterestRate / iNumPaymentsPerYear
44
45        curPayment = m_curPrincipal * _
```


LISTAGEM 15.8 A Herança É uma Boa Maneira de Adicionar Funcionalidade
(continuação)

```

46      ((1 - (1 + dblFractionalInterestRate))_
47      /(1 - ((1 + dblFractionalInterestRate) ^ iPayments))_
48      + dblFractionalInterestRate)
49
50      Return curPayment
51  End Function
52 End Class

```

ANÁLISE

Os detalhes da Listagem 15.8 não são muito relevantes, exceto pela maneira como essa classe teve de ser escrita para permitir que você baseasse nela uma nova classe de antecipação da hipoteca. Na linha 36, a função `PaymentAmount` é declarada por meio da palavra-chave `Overridable`. Sem essa palavra-chave, `AccelMortgage` (veja a Listagem 15.9) não poderia fornecer sua própria versão da função `PaymentAmount`.

LISTAGEM 15.9 `AccelMortgage` Adiciona a Capacidade de Calcular Mais Rapidamente um Plano de Pagamento da Hipoteca

```

1 Public Class AccelMortgage
2     Inherits Mortgage
3     Private m_iNumberOfPaymentsInYear As Integer = 12
4
5     Public Property PaymentsInYear() As Integer
6         Get
7             Return m_iNumberOfPaymentsInYear
8         End Get
9         Set(ByVal Value As Integer)
10            m_iNumberOfPaymentsInYear = Value
11        End Set
12    End Property
13
14    Public Overrides Function PaymentAmount() As Decimal
15        Dim iPayments As Integer
16        Dim dblFractionalInterestRate As Double
17        Dim curPayment As Decimal
18
19        iPayments = Me.NumberOfYears * Me.PaymentsInYear
20        dblFractionalInterestRate = _
21            Me.AnnualInterestRate / Me.PaymentsInYear
22        curPayment = Me.Principal * _
23            ((1 - (1 + dblFractionalInterestRate))_
24            /(1 - ((1 + dblFractionalInterestRate) ^ iPayments))_

```

LISTAGEM 15.9 AccelMortgage Adiciona a Capacidade de Calcular Mais Rapidamente um Plano de Pagamento da Hipoteca (*continuação*)

```
25         + dblFractionalInterestRate)
26
27     Return curPayment
28 End Function
29 End Class
```

ANÁLISE

A Listagem 15.9 ilustra a sobreposição de um método de uma classe derivada, e a linha 14 faz exatamente isso declarando um método que existe na classe básica e especificando a palavra-chave `Overrides`. A própria função `PaymentAmount` foi reescrita para dar suporte a pagamentos que não sejam mensais. Nessa versão revista da função, as propriedades necessárias da classe são acessadas por meio da palavra-chave `Me` (linhas 19, 20 e 22), em vez de se usar as variáveis internas diretamente, porque uma classe derivada não tem acesso às variáveis privadas da classe básica. A Listagem 15.10 é um programa cliente simples que usa essas duas classes, projetado para comparar uma hipoteca comum (com pagamentos mensais) com a mesma hipoteca quitada com pagamentos antecipados bimestrais.

LISTAGEM 15.10 O Programa Cliente Chama os Dois Cálculos da Hipoteca

```
1 Module Main
2     Sub Main()
3         Dim objMortgage As New Mortgage()
4         With objMortgage
5             .AnnualInterestRate = 0.07
6             .NumberOfYears = 25
7             .Principal = 140000
8             Console.WriteLine("{0:C}", .PaymentAmount())
9         End With
10
11        Dim objNewMortgage As New AccelMortgage()
12        With objNewMortgage
13            .AnnualInterestRate = 0.07
14            .NumberOfYears = 25
15            .PaymentsInYear = 26
16            .Principal = 140000
17            Console.WriteLine("{0:C}", .PaymentAmount())
18        End With
19        Console.ReadLine()
20    End Sub
21 End Module
```

Decidir qual a função de uma propriedade e a de um método nem sempre é fácil e muitos desenvolvedores fazem a escolha errada. Não há ‘regra’ que informe o que fazer, mas posso oferecer-lhe a diretriz que utilizo e a lógica existente nela. Emprego um teste simples quando projeto meus objetos: “as propriedades devem fazer com que algo aconteça”. Gosto de escrever classes de maneira que configurar uma propriedade três vezes apresente o mesmo resultado final que configurá-la uma vez.

Essa diretriz única evita situações em que a ordem da configuração das propriedades de um objeto pode afetar o resultado. Na classe `Mortgage`, por exemplo, o código poderia ser escrito de modo que a configuração da propriedade `Principal` fizesse com que o valor do pagamento tivesse de ser recalculado. Se esse fosse o caso, então, você poderia querer se certificar de ter configurado as outras propriedades (`AnnualInterestRate`, `NumberOfYears`) antes de `Principal`. Evito inserir qualquer código em uma rotina de propriedade que altere outras partes internas de minha classe, que não sejam sua variável interna correspondente. A função `PaymentAmount` da classe `Mortgage` poderia certamente ser uma propriedade ou um método porque não altera nada na classe quando efetua seu cálculo. Criei-a como um método nesta lição, porém como uma propriedade no Dia 14; em geral há situações em que qualquer uma das opções é adequada.

Adicionando Eventos

Os eventos são uma parte importante dos objetos na plataforma .NET, principalmente em interfaces gráficas com o usuário, mas no restante do Framework também. Eles permitem que um objeto informe a todos os interessados que algo aconteceu, o que é uma interação muito mais ativa entre um objeto e o código que o utiliza do que a alternativa em que as propriedades do objeto são chamadas repetidamente. Os eventos não são adequados para todas as situações, por exemplo, as classes `Mortgage` e `AccelMortgage` não apresentam uma necessidade real de possuírem eventos; seu código é embutido, e só executado se requisitado. Quando você precisar de que seu objeto informe algo a seus usuários, os eventos serão um ótimo mecanismo para fazê-lo.

Considere uma classe projetada para se comunicar com um programa de correio e alertar você quando chegar uma correspondência que atenda a um critério específico. Uma propriedade poderia ser configurada como um flag para quando essa nova mensagem chegar, porém, seu programa teria de continuar verificando esse valor. Em vez disso, é possível criar um evento que sua classe possa lançar com alguns trechos de dados para descrever o que ocorreu. Seu programa principal não fará nenhuma chamada do objeto; apenas escreva o código em um manipulador de eventos para esse evento de seu objeto. Para criar um evento, adicione uma declaração a sua classe, como no exemplo da Listagem 15.11.

LISTAGEM 15.11 Adicionando uma Declaração de um Evento

```
1 Public Class OutlookMessageWatcher
2
3     Public Event EmailArrived(ByVal From As String, _
4         ByVal Subject As String, _
```

LISTAGEM 15.11 Adicionando uma Declaração de um Evento (*continuação*)

```
5     ByVal Message As Outlook.MailItem)
6
7 End Class
```

Quando esse evento for lançado, você poderá fornecer valores para cada um dos parâmetros, e esses valores serão enviados para qualquer rotina que venha a manipulá-lo. Para lançar o evento, o que será feito em sua classe, use a instrução `RaiseEvent`, como mostra a Listagem 15.12 (a classe `OutlookMessageWatcher` completa).

LISTAGEM 15.12 Listagem Completa do Outlook Message Watcher

```
1 Imports System
2 Imports System.Windows.Forms
3
4 Public Class OutlookMessageWatcher
5     Private WithEvents objInboxItems As Outlook.Items
6     Private objOutlook As Outlook.Application
7     Public Event EmailArrived(ByVal From As String, _
8         ByVal Subject As String, _
9         ByVal Message As Outlook.MailItem)
10
11     Public Sub New()
12         objOutlook = New Outlook.Application()
13         objOutlook.Session.Logon(NewSession:=False)
14         objInboxItems = objOutlook.Session.GetDefaultFolder _
15             (Outlook.OlDefaultFolders.olFolderInbox).Items
16     End Sub
17
18     Private Sub objInboxItems_ItemAdd(ByVal Item As Object) _
19         Handles objInboxItems.ItemAdd
20         Dim objNewMail As Outlook.MailItem
21         Try
22             objNewMail = CType(Item, Outlook.MailItem)
23             RaiseEvent EmailArrived(objNewMail.SenderName, _
24                 objNewMail.Subject, _
25                 objNewMail)
26         Catch objException As Exception
27             MessageBox.Show(objException.Message)
28         End Try
29     End Sub
30 End Class
```

Depois de criar um evento em sua classe, você poderá escrever um código para manipular esses eventos em outras classes declarando seu objeto com a palavra-chave `WithEvents` e, em seguida, gerar uma rotina de evento por meio de `Handles`. Por exemplo, a Listagem 15.13 mostra como essa classe nova poderia ser empregada como parte de um formulário Windows.

LISTAGEM 15.13 Testando o Message Watcher

```
1 Public Class Form1
2     Inherits System.Windows.Forms.Form
3     Private WithEvents objOMW As OutlookMessageWatcher
4
5     'O código gerado pelo Windows Form Designer foi omitido
6
7     Private Sub Form1_Load(ByVal sender As System.Object, _
8     ByVal e As System.EventArgs) Handles MyBase.Load
9         objOMW = New OutlookMessageWatcher()
10    End Sub
11
12    Private Sub objOMW_EmailArrived(ByVal From As String, _
13        ByVal Subject As String, _
14        ByVal Message As Outlook.MailItem) _
15        Handles objOMW.EmailArrived
16        1bMailItems.Items.Add(Subject)
17    End Sub
18 End Class
```

Os dois requisitos para o uso do(s) evento(s) de seu objeto são declarar o objeto por meio de `WithEvents` e criar um manipulador de eventos empregando a palavra-chave `Handles`. Se você estiver trabalhando no IDE do Visual Studio, ele poderá criar o procedimento do evento automaticamente se o objeto for declarado com o uso de `WithEvents`. Selecione o objeto na lista suspensa da direita e, em seguida, seu evento na lista suspensa da esquerda.

Definindo e Usando Interfaces

Um dos recursos mais importantes da programação orientada a objetos (POO) é a capacidade de tratar um objeto como se fosse uma instância de outra classe. Quando você lidar com a herança, isso significa que poderá tratar qualquer classe derivada como se fosse algum de seus antecessores. Isso é útil e em geral um estimulante essencial na criação de classes básicas, mas seu uso é limitado às classes com relacionamento de herança.

Outra maneira de produzir um resultado semelhante, classes que podem ser tratadas como se fossem outras, é usar interfaces. Uma *interface* é um tipo especial de classe que não contém código, mas é empregada como uma alternativa para descrever a aparência de um objeto. Em segui-

da, outras classes podem implementar uma ou mais dessas interfaces, permitindo que sejam tratadas como se fossem um desses objetos.

As interfaces são usadas em todo o .NET Framework, permitindo que uma quantidade indefinida de classes declare o fornecimento de um conjunto específico de recursos. Uma das interfaces definidas no .NET Framework é `IComparable` (os nomes das interfaces em geral possuem um `I` maiúsculo como prefixo), que fornece apenas um método (`CompareTo`) empregado para comparar o objeto atual com outra instância da mesma classe. Qualquer classe que implemente a interface `IComparable` estará essencialmente declarando que duas instâncias dessa classe podem ser comparadas, e um relacionamento maior que, menor que ou igual a pode ser determinado.

Para implementar uma interface, uma classe quase sempre precisa fornecer sua própria versão de um ou mais métodos. As classes que implementarem `IComparable` terão de apresentar sua versão do método `CompareTo`. A Listagem 15.14 mostra uma classe que implementa `IComparable` e a versão personalizada de `CompareTo` que a forneceu.

LISTAGEM 15.14 `IComparable` Permite Que Sua Classe Funcione

```
1 Public Class Person
2     Implements IComparable
3     Private m_sName As String
4     Private m_sFirstName As String
5     Private m_sLastName As String
6
7     Public ReadOnly Property DisplayName() As String
8         Get
9             Return String.Format("{0} {1}", _
10 m_sFirstName, m_sLastName)
11         End Get
12     End Property
13
14     Public Property FirstName() As String
15         Get
16             Return m_sFirstName
17         End Get
18         Set(ByVal Value As String)
19             m_sFirstName = Value
20         End Set
21     End Property
22
23     Public Property LastName() As String
24         Get
25             Return m_sLastName
26         End Get
27         Set(ByVal Value As String)
28             m_sLastName = Value
```


LISTAGEM 15.14 IComparable Permite Que Sua Classe Funcione (*continuação*)

```

29      End Set
30  End Property
31
32  Public Function CompareTo(ByVal obj As Object) As Integer _
33      Implements System.IComparable.CompareTo
34      'Compare esta instância com obj, retorne um número
35      'menor que zero para indicar obj < me, 0 para indicar
36      'obj = me, e maior que zero para indicar obj > me
37      Dim objOtherPerson As Person
38      objOtherPerson = CType(obj, Person)
39
40      If objOtherPerson.LastName < Me.LastName Then
41          Return -1
42      ElseIf objOtherPerson.LastName > Me.LastName Then
43          Return 1
44      Else
45          If objOtherPerson.FirstName < Me.FirstName Then
46              Return -1
47          ElseIf objOtherPerson.FirstName > Me.FirstName Then
48              Return 1
49          Else
50              Return 0
51          End If
52      End If
53  End Function
54 End Class

```

Implementar `IComparable` faz com que a classe `Person` possa ser passada para funções como se fosse um objeto do tipo `IComparable` e usada com outros objetos que precisam dessa interface para ser implementados. Um exemplo de como isso poderia ser útil é encontrado na classe `SortedList` do .NET Framework. Essa classe permite que seja criada uma lista de objetos, cada uma com uma chave associada, e mantém a lista ordenada pelos valores das chaves. O segredo é que os objetos fornecidos como chaves devem dar suporte à `IComparable` porque `SortedList` depende do método `CompareTo` exposto por essa interface para executar sua ordenação. Esse requisito não será um problema se sua chave for um tipo de dado simples, como uma string, um inteiro ou uma data, já que todos eles dão suporte à interface `IComparable`, mas se você quiser empregar uma classe de sua autoria como chave, ela não funcionará a menos que implemente `IComparable`. A Listagem 15.15 mostra como poderíamos criar uma lista ordenada utilizando instâncias da classe `Person` como valores das chaves.

LISTAGEM 15.15 Implementar IComparable Permite Que uma Classe Funcione em uma Lista Ordenada

```
1 Option Strict On
2 Option Explicit On
3
4 Module AllAboutObjects
5     Public Sub Main()
6         Dim objPerson1 As New Person()
7         Dim objPerson2 As New Person()
8         Dim objPerson3 As New Person()
9         Dim Val1, Val2, Val3 As Integer
10
11         Val1 = 234
12         Val2 = 13
13         Val3 = 500
14
15         objPerson1.FirstName = "John"
16         objPerson1.LastName = "Adams"
17         objPerson2.FirstName = "Quincy"
18         objPerson2.LastName = "Wallace"
19         objPerson3.FirstName = "Arlene"
20         objPerson3.LastName = "Ratuski"
21
22         Dim s1People As New SortedList()
23
24         s1People.Add(objPerson1, Val1)
25         s1People.Add(objPerson2, Val2)
26         s1People.Add(objPerson3, Val3)
27
28         Dim objDE As DictionaryEntry
29
30         For Each objDE In s1People
31             Console.WriteLine("{0} {1}", _
32                 CType(objDE.Key, Person).DisplayName, _
33                 CType(objDE.Value, Integer))
34         Next
35         Console.ReadLine()
36     End Sub
37 End Module
```

As interfaces devem ser usadas no lugar da herança quando sua intenção for indicar que uma classe fornece alguma forma de funcionalidade comum (como a capacidade de ser comparada). A classe Person dos exemplos anteriores tem uma finalidade completamente desvinculada de

seu suporte a ser comparada, de modo que esse suporte é demonstrado melhor por meio de uma interface.

O .NET Framework fornece uma ampla variedade de interfaces, mas você também pode criar as suas de maneira semelhante ao processo de definição de uma classe nova. A definição de uma interface começa com `Public Interface <nome da interface>`, termina com `End Interface` e contém declarações de métodos, propriedades e eventos entre o início e o fim. Nenhum código entra em uma interface independentemente do que você esteja fazendo, portanto as declarações de método e propriedade são compostas apenas da primeira linha de uma declaração comum. A Listagem 15.16 define uma interface simples, `IDebugInfo`, para fornecer algumas informações gerais sobre a classe que tem como finalidade a depuração.

15

LISTAGEM 15.16 As Interfaces Se Parecem com Classes Vazias

```
1 Option Explicit On
2 Option Strict On
3
4 Public Interface IDebugInfo
5     ReadOnly Property ClassName() As String
6     ReadOnly Property Author() As String
7     ReadOnly Property SourceFile() As String
8     ReadOnly Property Description() As String
9 End Interface
```

As interfaces são semelhantes às classes em alguns aspectos, mas como você pode ver na Listagem 15.16, bem diferentes em outros. Dentro da declaração `Interface` (linhas 4 a 9), os membros são definidos apenas com sua declaração, e nenhum código de implementação é fornecido. O escopo também não é incluído como parte dessas declarações, mas `Public` será o padrão quando essa interface for implementada em uma de suas classes. Independentemente do escopo escolhido em sua classe, esses membros estarão disponíveis por meio da interface `IDebugInfo`. No caso dessa interface, as propriedades serão as mesmas para todas as instâncias de uma única classe e estarão destinadas a descrever essa classe específica. A natureza dessas propriedades permite valores apenas de leitura, mas as interfaces podem apresentar o tipo de propriedade que for necessário. Um exemplo contendo uma rotina de registros, mostrado na Listagem 15.17, armazena informações de rastreamento sobre qualquer classe que dê suporte à `IDebugInfo`.

LISTAGEM 15.17 Um Procedimento de Registros Aceita Qualquer Classe Que Implemente `IDebugInfo`

```
1 Option Explicit On
2 Option Strict On
3
4 Imports System
5 Imports System.IO
```

LISTAGEM 15.17 Um Procedimento de Registros Aceita Qualquer Classe Que Implemente IDebugInfo (*continuação*)

```

6
7 Public Class LoggingRoutine
8     Const DefaultLogFile As String = "C:\LogFile.txt"
9     Shared Sub LogMsgToFile(ByVal Message As String, _
10         ByVal FileName As String, _
11         ByVal Source As IDebugInfo)
12         Dim objLogFile As StreamWriter
13         If File.Exists(FileName) Then
14             objLogFile = File.AppendText(FileName)
15         ElseIf Directory.Exists _
16             (Path.GetDirectoryName(FileName)) Then
17             objLogFile = File.CreateText(FileName)
18         Else
19             'error, use default log file
20             objLogFile = File.AppendText(DefaultLogFile)
21         End If
22
23         objLogFile.WriteLine("-----")
24         objLogFile.WriteLine(Message)
25         objLogFile.WriteLine("From {0} ({1}) by {2}", _
26             Source.ClassName, Source.SourceFile, Source.Author)
27         objLogFile.WriteLine("Description: {0}", _
28             Source.Description)
29         objLogFile.WriteLine("-----")
30
31         objLogFile.Flush()
32         objLogFile.Close()
33     End Sub
34 End Class
  
```

ANÁLISE

Essa classe de registros mostra como o uso de uma interface comum para vários objetos pode ser parte do projeto de um aplicativo. O procedimento compartilhado `LogMsgToFile` (linha 9) aceita vários parâmetros incluindo um objeto do tipo `IDebugInfo`, permitindo, portanto, que qualquer instância de uma classe que implemente essa interface seja passada para essa rotina. A rotina em si não está fazendo nada excepcional, mas algumas partes dela podem ser interessantes. As classes `Path`, `File` e `Directory` são todas do espaço de nome `System.IO`, e cada uma expõe várias funções compartilhadas ou estáticas, portanto, você pode usar esses objetos sem precisar criar uma instância deles. O código da Listagem 15.18 mostra o exemplo de uma classe que implementa `IDebugInfo`.

LISTAGEM 15.18 Usando a Interface IDebugInfo

```
1 Option Explicit On
2 Option Strict On
3
4 Module Main
5     Sub Main()
6         Dim objPerson As New Person()
7         objPerson.FirstName = "Duncan"
8         objPerson.LastName = "Mackenzie"
9         Console.WriteLine(objPerson.DisplayName())
10    End Sub
11 End Module
12
13 Public Class Person
14     Implements IComparable
15     Implements IDebugInfo
16
17     Private m_sName As String
18     Private m_sFirstName As String
19     Private m_sLastName As String
20
21     Public Sub New()
22         LoggingRoutine.LogMsgToFile _
23             ("Started Up", "C:\test.txt", Me)
24     End Sub
25
26     Public ReadOnly Property DisplayName() As String
27     Get
28         LoggingRoutine.LogMsgToFile _
29             (String.Format("DisplayName called, {0} {1}output", _
30                 m_sFirstName, m_sLastName), "C:\test.txt", Me)
31
32         Return String.Format("{0} {1}", m_sFirstName, m_sLastName)
33     End Get
34 End Property
35
36     Public Property FirstName() As String
37     Get
38         Return m_sFirstName
39     End Get
40     Set(ByVal Value As String)
41         m_sFirstName = Value
42     End Set
43 End Property
44
45     Public Property LastName() As String
```


LISTAGEM 15.18 Usando a Interface IDebugInfo (*continuação*)

```
46         Get
47             Return m_sLastName
48         End Get
49         Set(ByVal Value As String)
50             m_sLastName = Value
51         End Set
52     End Property
53
54     Public Function CompareTo(ByVal obj As Object) As Integer _
55         Implements System.IComparable.CompareTo
56         'Compare essa instância com obj, retorne um número
57         'menor que zero para indicar que obj < me, igual a 0 para indicar que
58         'obj = me e maior que zero para indicar que obj > me
59         Dim objOtherPerson As Person
60         objOtherPerson = CType(obj, Person)
61
62         If objOtherPerson.LastName < Me.LastName Then
63             Return -1
64         ElseIf objOtherPerson.LastName > Me.LastName Then
65             Return 1
66         Else
67             If objOtherPerson.FirstName < Me.FirstName Then
68                 Return -1
69             ElseIf objOtherPerson.FirstName > Me.FirstName Then
70                 Return 1
71             Else
72                 Return 0
73             End If
74         End If
75     End Function
76
77     Public ReadOnly Property Author() As String _
78         Implements IDebugInfo.Author
79         Get
80             Return "Duncan Mackenzie"
81         End Get
82     End Property
83
84     Public ReadOnly Property ClassName() As String _
85         Implements IDebugInfo.ClassName
86         Get
87             Return "Person"
88         End Get
89     End Property
90
```


LISTAGEM 15.18 Usando a Interface IDebugInfo (*continuação*)

```
91     Public ReadOnly Property Description() As String _
92         Implements IDebugInfo.Description
93     Get
94         Return "A classe Person foi projetada para representar" _
95             & " um cliente ou um funcionário"
96     End Get
97 End Property
98
99     Public ReadOnly Property SourceFile() As String _
100         Implements IDebugInfo.SourceFile
101     Get
102         Return "\\liquidsoap\bigproject\Source\Person.vb"
103     End Get
104 End Property
105 End Class
```

ANÁLISE

Para usar esse programa, reúna a interface IDebugInfo (Listagem 15.16), a rotina de registros (Listagem 15.17) e esse código (Listagem 15.18) em um projeto, porém em arquivos separados. Vá até as propriedades do projeto e certifique-se de que a rotina `Main` da Listagem 15.18 esteja selecionada como objeto inicial (Startup object), e tudo deve funcionar bem. Lembre-se de que esse código pode ser descarregado da página do livro, portanto, se quiser, você poderá evitar a digitação.

A Listagem 15.18 apresenta dois itens: o exemplo de uma classe (Person) que implementa IDebugInfo (linhas 13 a 105) e de um trecho de código (linhas 4 a 11) que cria uma instância da classe Person e, em seguida, chama o método `DisplayName`. Como descrito, se você incluir todo esse código em um projeto – três arquivos .vb separados serão adequados – e definir `Sub Main` como seu objeto inicial, então, o código deverá ser executado sem problemas e produzir um arquivo (`c:\test.txt`) contendo suas informações de rastreamento.

Tanto as interfaces quanto a herança fornecem a capacidade de se ter muitos objetos diferentes compartilhando atributos em comum, mas duas diferenças essenciais as tornam adequadas para funções distintas. As interfaces não incluem qualquer implementação junto a esses atributos; a definição Interface não contém código, e as classes que implementam essa interface devem fornecer elas mesmas toda a funcionalidade. Essa com certeza é uma vantagem da herança, o compartilhamento da funcionalidade implementada na classe básica com as derivadas, porém, as interfaces compensam essa diferença com uma flexibilidade muito maior, permitindo que uma única classe implemente quantas interfaces quisermos.

No final, as duas tecnologias têm alguma semelhança, porém alguns procedimentos diferentes. Use interfaces quando quiser indicar que uma classe tem determinados (normalmente mais do que um) recursos, semelhantes a um relacionamento como “minha classe suporta a interface

IdebugInfo”. Inheritance, por outro lado, indica “é um” relacionamento. Frases como “Cliente é uma pessoa” e “Funcionário é uma pessoa” poderiam descrever o relacionamento entre duas classes derivadas e suas classes-base.

Usando os Objetos Que Você Criou

Depois de definir um conjunto de classes e interfaces, provavelmente você desejará usá-las como parte de um aplicativo. Nos exemplos que vimos até agora, apenas um projeto esteve envolvido, portanto, utilizar suas classes foi tão simples quanto declarar novas variáveis. Esse poderá ser o caso em seus sistemas reais, mas fazer com que as definições de classes sejam parte de cada projeto não será útil e decerto não funcionará bem quando quisermos que muitos projetos compartilhem as mesmas definições de um conjunto de classes e interfaces. Para tornar seu código facilmente disponível para qualquer projeto que precise usá-lo, é necessário criar um projeto Class Library. Esse tipo de projeto não tem como finalidade ser executado isoladamente, mas, em vez disso, contém um código que poderá ser usado por outros projetos.

1. Selecione File, New e Project no menu, o que deve abrir a caixa de diálogo New Project (veja a Figura 15.2).
2. Selecione o modelo de projeto Class Library e observe a descrição adequada do texto (“Um projeto para a criação de classes a serem usadas em outros aplicativos”) que confirma a finalidade desse modelo.
3. Forneça um nome para esse novo projeto, no lugar do padrão ClassLibrary1, e dê um clique em OK para criá-lo.

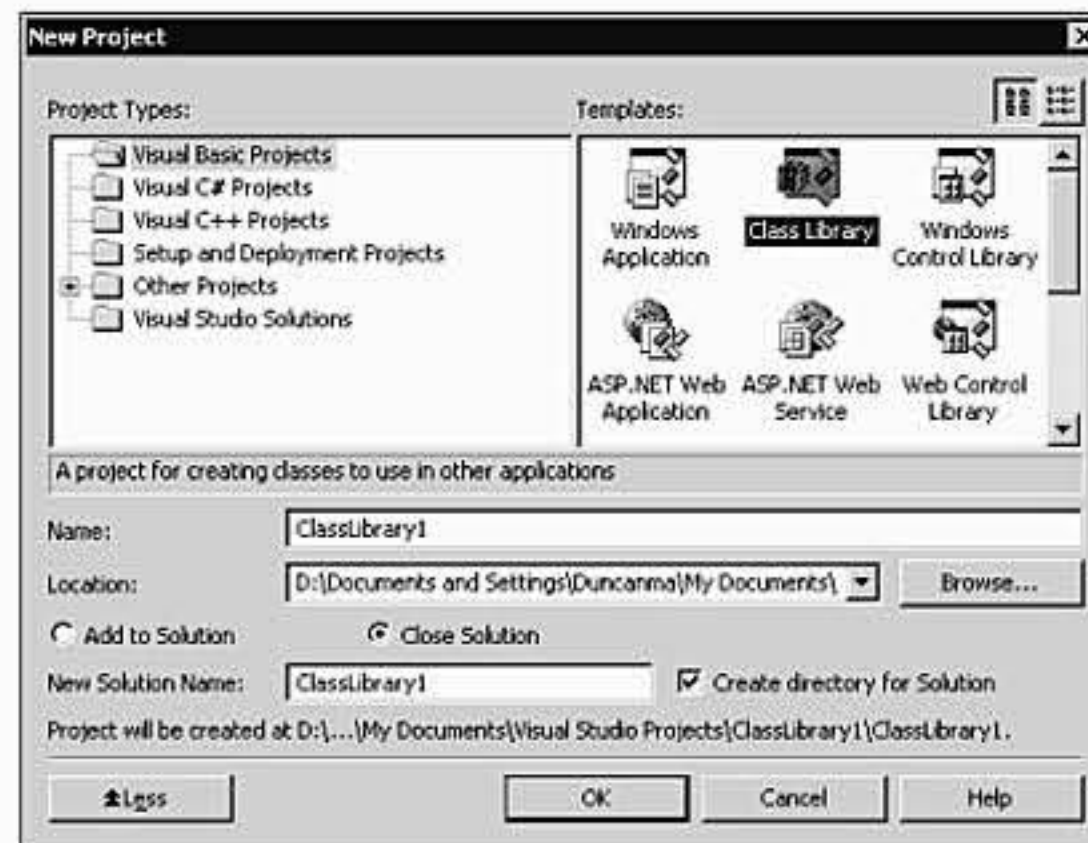
Os projetos desse tipo são compostos de quantos arquivos de código repletos de classes você quiser adicionar e nenhum objeto inicial. É possível disponibilizar suas classes tendo apenas uma delas em um arquivo ou reunindo todo o seu código; o resultado final é o mesmo.

Espaços de Nome

Uma maneira útil de criar uma organização para suas classes é usar espaços de nome. A definição de espaços de nome já é um mecanismo para a estruturação do código, mas quando você criar bibliotecas de classes, a organização será ainda mais importante. O .NET Framework emprega muito os espaços de nome, produzindo classes organizadas de maneira clara pelos espaços de nome que as contêm (System, System.Data, System.IO e assim por diante), e podemos fazer o mesmo apenas envolvendo seu código com as definições dos espaços de nome. Eles serão declarados em seu código por meio de Namespace *<nome do espaço de nome>* e End Namespace. Poderão ser aninhados (o que permitiria a criação de uma estrutura na forma *Espaço1.Espaço2.Espaço3*) pela especificação do nome completo (*Espaço1.Espaço2*), já que não é possível aninhar declarações Namespace. Quando definidos envolvendo uma classe (veja a Listagem 15.19) ou várias, elas serão consideradas parte desse espaço de nome e poderão ser referenciadas como *Espaço1.Classe1* em seu código.

FIGURA 15.2

A caixa de diálogo New Project inclui modelos para a criação de bibliotecas de classes.



LISTAGEM 15.19 Os Espaços de Nome Ajudarão a Organizar Suas Classes

```

1 Option Explicit On
2 Option Strict On
3 Namespace libPerson
4     Public Class Class2
5
6     End Class
7 End Namespace
8
9 Module mod1
10     Sub Main()
11         Dim objClass2 As libPerson.Class2
12
13     End Sub
14 End Module

```

Você também pode usar o mesmo nome para um espaço de nome em vários arquivos de código e até em diversas bibliotecas de códigos, e o Visual Studio .NET considerará cada definição desse espaço de nome como parte do todo. As Listagens 15.20 e 15.21 ilustram isso, representando dois arquivos distintos.

LISTAGEM 15.20 O Mesmo Espaço de Nome Pode Ser Usado em Vários Arquivos

```

1 Option Explicit On
2 Option Strict On
3 Namespace libPerson
4     Public Class Class1
5

```

LISTAGEM 15.20 O Mesmo Espaço de Nome Pode Ser Usado em Vários Arquivos
(*continuação*)

```
6 End Class
7 End Namespace
```

LISTAGEM 15.21 Para o Programador Que Usar Seu Código, Tudo Parecerá Parte de um Único Espaço de Nome

```
1 Option Explicit On
2 Option Strict On
3 Namespace libPerson
4     Public Class Class2
5
6     End Class
7 End Namespace
8
9 Module mod1
10     Sub Main()
11         Dim objClass2 As libPerson.Class2
12         Dim objClass1 As libPerson.Class1
13     End Sub
14 End Module
```

No final da Listagem 15.21, podemos ver o código declarando e usando as duas classes (linhas 11 e 12). Quando você quiser construir uma biblioteca de códigos e compilá-la em uma DLL, poderá envolver todo o seu código em declarações Namespace para facilitar ao usuário a localização de suas classes. Como examinaremos um pouco mais adiante, suas classes já parecerão estar agrupadas depois que forem inseridas em uma biblioteca e fizerem referência a ela em outro projeto.

Criando e Usando uma DLL da Biblioteca

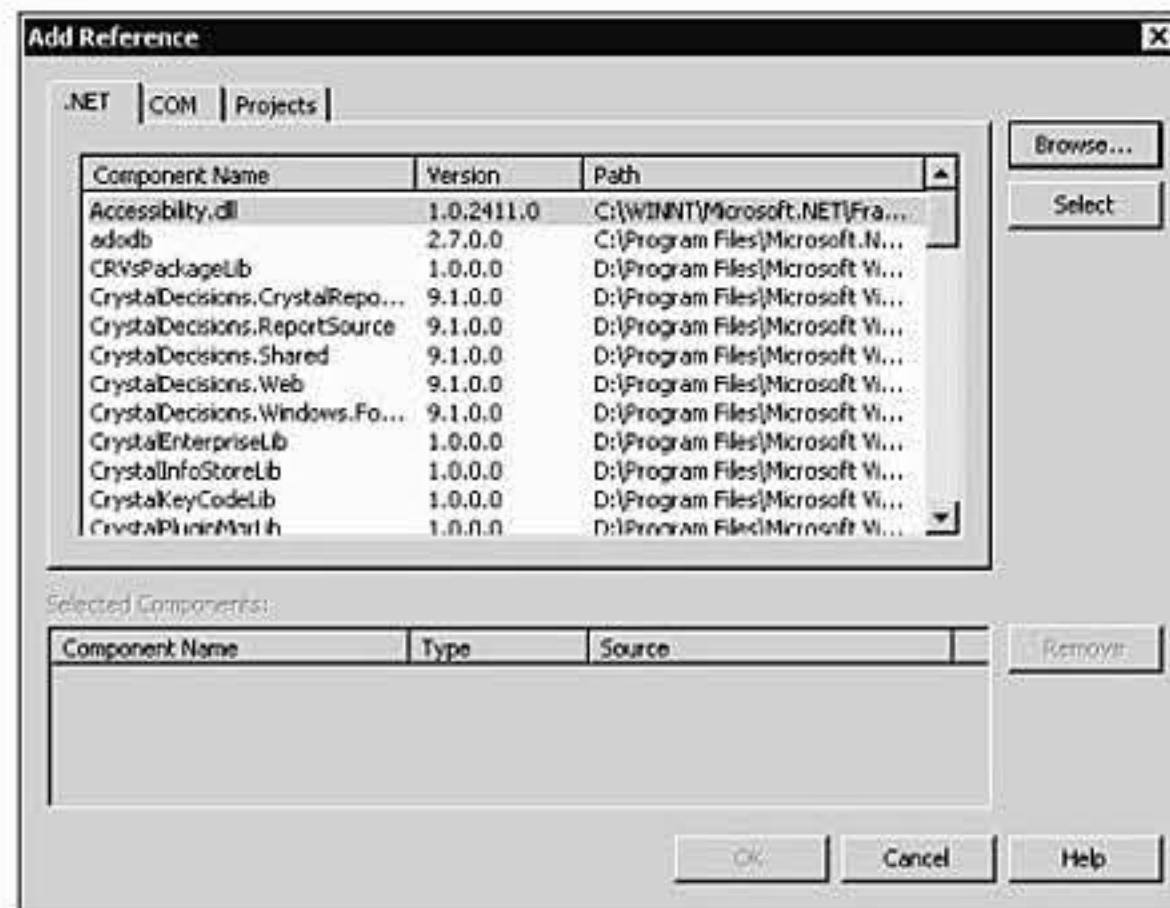
Depois de adicionar todas as suas classes e interfaces a seu projeto de biblioteca, para construir a DLL final você só terá de selecionar Build no menu Build ou pressionar as teclas Ctrl+Shift+B. Isso cria uma DLL, em geral chamada <nome do projeto>.dll, localizada no diretório bin sob seu projeto. Esse local e o nome são adequados, mas para testar sua nova biblioteca, será preciso fechar essa solução e abrir um novo projeto de aplicativo do console.

1. Selecione New e Project no menu File e crie um novo aplicativo do console no Visual Basic .NET.
2. Depois que o novo projeto tiver sido aberto, dê um clique com o botão direito do mouse na pasta References da tela do Solution Explorer em que ele aparecer.

3. Selecione Add Reference no menu que aparecerá, e você verá a caixa de diálogo Add Reference (veja a Figura 15.3).
4. Dê um clique no botão Browse e encontre o arquivo libPerson.dll que foi criado.

FIGURA 15.3

A caixa de diálogo Add Reference pode ser usada para que sejam adicionadas referências a componentes e referências que conduzam diretamente a outros projetos.



5. Depois de localizar a DLL desejada, dê um clique em OK para confirmá-la na caixa de diálogo Add Reference; em seguida, dê um clique em OK mais uma vez para retornar a seu código com a recém-adicionada referência disponível para seu projeto.

Para usar essa biblioteca de classes, você pode tratá-la como qualquer parte do .NET Framework. As classes propriamente ditas serão encontradas no final do caminho `<nome do projeto>.<espaço de nome>.<classe>`. Supondo que seu projeto seja chamado de `libPerson` e uma referência a esse nome seja adicionada, então, a Listagem 15.22 ilustra como as instâncias das classes de sua biblioteca poderiam ser manipuladas.

LISTAGEM 15.22 Usando Sua Biblioteca

```

1 Option Explicit On
2 Option Strict On
3
4 Module Module1
5     Sub Main()
6         Dim objClass As libPerson.libPerson.Class1
7     End Sub
8 End Module

```

Observe que o uso do espaço de nome `libPerson` resultou em um agrupamento secundário que provavelmente é desnecessário. Empregue espaços de nome em suas bibliotecas para criar agrupamentos mais granulares do que a biblioteca em sua totalidade.

Resumo

Fazer a POO trabalhar para você, em seus aplicativos, não é uma questão técnica, e sim de projeto. Quando for criar o projeto conceitual de seu aplicativo, precisará pensar em termos de objetos. Se conseguir fazer isso, então, desenvolver o sistema usando objetos não será uma mudança completa de paradigma, mas apenas uma passagem do conceitual para a implementação.

P&R

P Ouvi o termo ‘polimorfismo’ relacionado à programação orientada a objetos. Isso significa que preciso ter mais de uma esposa para trabalhar com o Visual Basic .NET?

R Não tema, isso nada tem a ver com suas relações familiares. O polimorfismo descreve a capacidade de tratar muitos objetos diferentes como se fossem do mesmo tipo, permitindo que você tenha classes diferentes, como *Círculo*, *Quadrado* e *Triângulo*, porém tratando todas como o tipo *Forma*. Esse conceito é útil e fornecido pelas tecnologias de herança e interfaces.

P Tenho de programar com objetos para usar o Visual Basic .NET?

R Para ser breve, sim. O Visual Basic .NET, e todo o .NET Framework, foi desenvolvido em torno dos objetos, e sempre que você usar qualquer das classes do .NET Framework estará trabalhando com eles. Os programas de sua autoria, no entanto, podem empregar muito ou pouco os objetos, conforme desejar. Todos os códigos criados ficarão armazenados em classes (ou módulos, que são apenas uma forma um pouco estranha de classes) e, portanto, não se pode evitar completamente os objetos, mas a opção de construir todo o seu sistema com entidades representadas como objetos é sua.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Se você tiver entidades em sua empresa que sirvam para várias finalidades (*Pedidos*, *Clientes*, *ItensDosPedidos*, *Eventos*), mas todas compartilharem alguns recursos ou capacidades, qual conceito da POO deve ser usado em sua criação?

2. Quando você tiver um relacionamento de herança, como Funcionário e Cliente, os dois derivados de Pessoa, como assegurará de que certos métodos ou propriedades da classe básica sejam implementados nas classes derivadas?
3. Novamente, em um relacionamento de herança, como você cria uma classe básica que não possa ser gerada, forçando os usuários de seu código a criar apenas classes que tenham sido derivadas da básica?
4. Como você indica que nenhuma das características de uma classe pode ser herdada? Como isso também é conhecido?

Exercícios

Crie um objeto, ou uma hierarquia de objetos, destinado a representar um cliente. Tente construí-lo usando uma ou mais classes com um enfoque na reutilização de seu código por outras partes de seu aplicativo.

PÁGINA EM BRANCO

SEMANA 3

DIA 16

Formulários Windows Avançados

No Dia 9, “Desenvolvendo uma Interface com o Usuário com os Formulários Windows”, você viu como construir aplicativos usando os formulários Windows (os Windows Forms). Como já deve ter imaginado, há muito mais a aprender nessa área. Nesta lição, examinaremos outros tópicos relacionados à criação de aplicativos com formulários Windows. Em particular, nos dedicaremos aos

- Menus.
- Programas com várias interfaces de documentos.
- Controles avançados dos formulários Windows.

Menus

Se você usou alguma versão do Windows, deve estar familiarizado com os menus. Eles aparecem em quase todo aplicativo e concedem acesso a maioria dos recursos do programa. Você vai querer adicionar menus a muitos de seus programas.

Adicionando um Menu a um Formulário

Você pode adicionar um menu a seu aplicativo da mesma maneira que insere qualquer outro controle, com duas pequenas diferenças. Ao dar um clique com o botão direito do mouse no controle `MainMenu` da caixa de ferramentas para adicionar um novo menu a seu formulário, irá se deparar com a primeira diferença. Em vez de ver um exemplo do menu no formulário, o controle

aparecerá em uma nova seção do IDE (veja a Figura 16.1). A segunda diferença entre adicionar um controle MainMenu e a maioria dos outros controles é que em geral só é inserido um único controle MainMenu em cada formulário.

**NOTA**

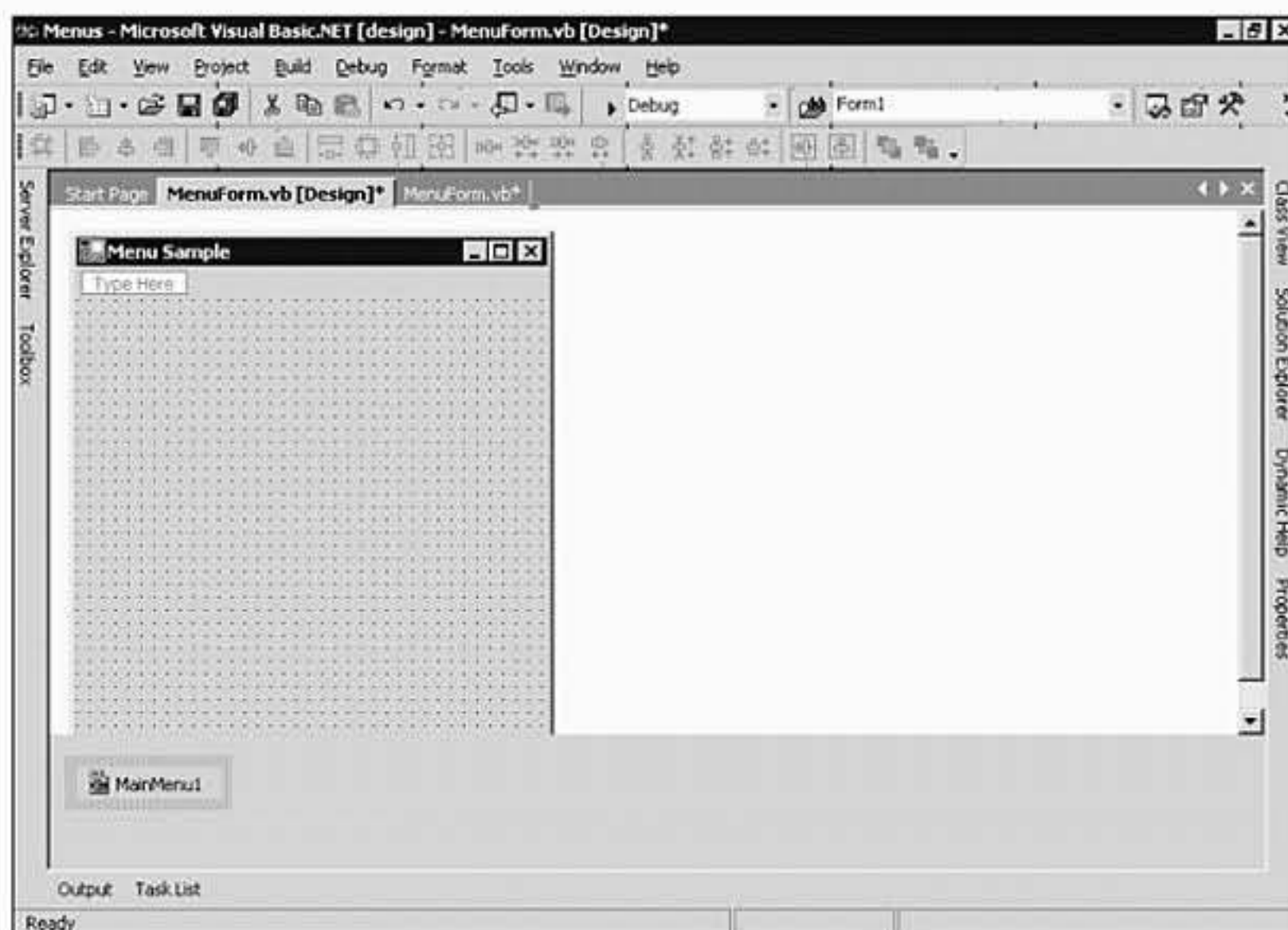
Embora você realmente possa adicionar vários controles MainMenu a um formulário, só um pode estar ativo em um dado momento. É possível alternar os controles MainMenu configurando a propriedade Menu do formulário com o menu desejado.

```
Me.Menu = mnuSecond
```

Você desejará fazer isso nos casos em que seja usado apenas um formulário para realizar múltiplas tarefas – por exemplo, se você carregar diferentes tipos de arquivos em um controle, no formulário. Você poderá alterar os menus para refletir as operações que poderão ser executadas no arquivo. Entretanto, uma solução prática para esse problema poderia ser ocultar e exibir os itens de menu de acordo com sua necessidade.

FIGURA 16.1

O formulário depois que o controle MainMenu foi adicionado.

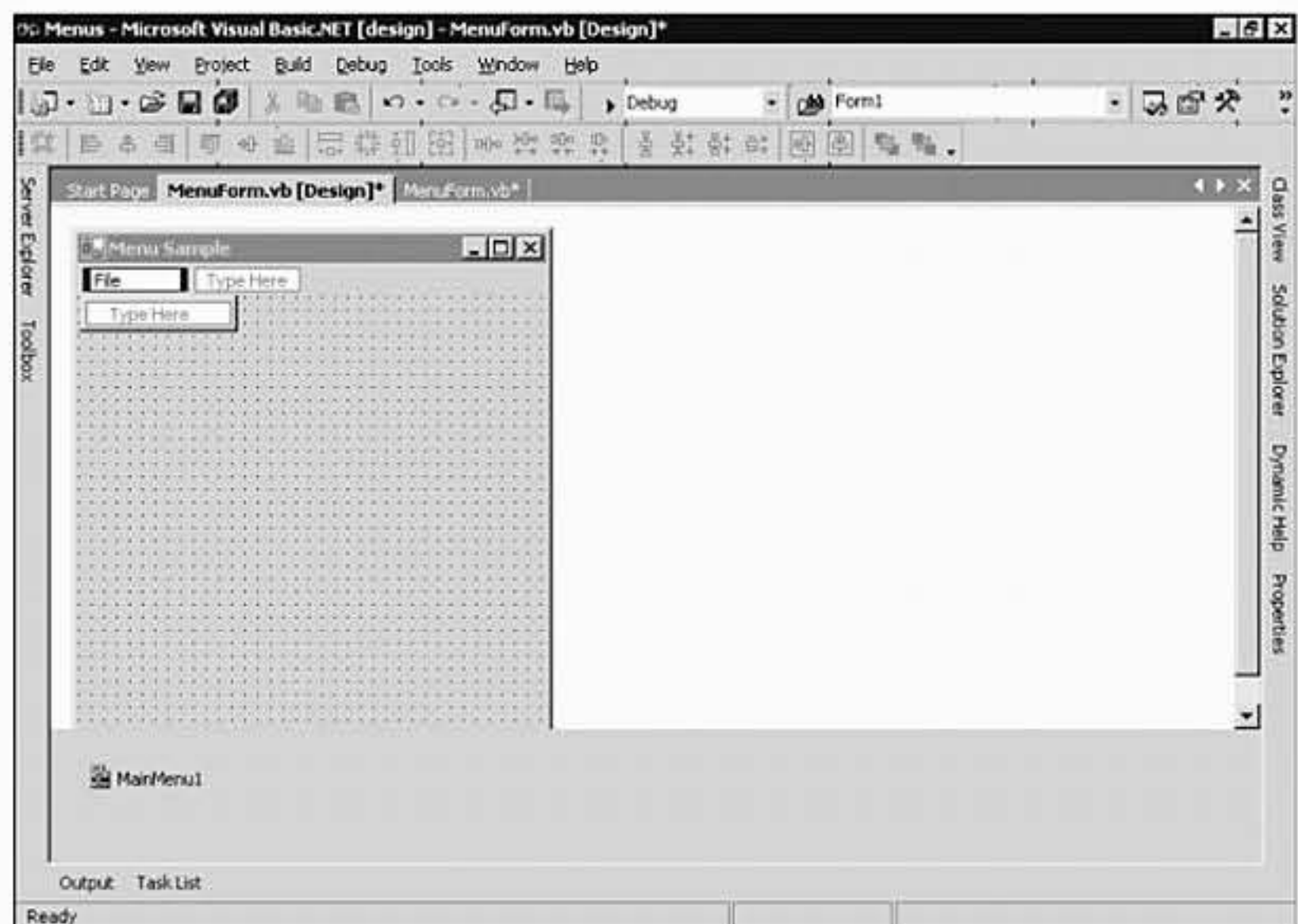


Embora o controle MainMenu não apareça no formulário, ele realmente o altera. Depois de adicionar um controle MainMenu, você deve ver uma nova faixa no formulário, logo abaixo da barra de título, como mostra a Figura 16.1.

Você adiciona itens ao menu construindo-o visualmente. Quando o controle MainMenu for selecionado no IDE, você deverá ver uma área de texto com as palavras *Type Here* no canto superior esquerdo do formulário que estiver projetando, onde o menu File normalmente se encontra. Dê um clique nesse espaço e digite o texto que deseja ver no menu. Ao fazê-lo, mais duas áreas de

texto surgirão: uma ao lado do item que acabou de ser criado e outra abaixo (veja a Figura 16.2). Os itens que forem adicionados à faixa do menu serão os menus de nível mais alto de seu aplicativo, enquanto os que forem inseridos abaixo deles serão as opções do menu. Por exemplo, digite File (**Arquivo**) na caixa inicial. Abaixo, adicione os itens New (**Novo**) e Exit (**Sair**). Dê um clique no espaço próximo ao menu File e digite View (**Visualizar**) para criar um novo menu. Abaixo de View, insira os itens Red (**Vermelho**), Green (**Verde**) e Blue (**Azul**). Enquanto estiver trabalhando em um menu, os outros menus de nível mais alto serão fechados. No entanto, poderão ser abertos novamente se precisarmos adicionar novos itens.

FIGURA 16.2
Edição de menus preparada.



Exatamente como nos outros controles, você deve estabelecer uma convenção para nomeação dos itens de seu menu. Sugiro utilizar a desta lição:

- Os itens do menu de nível superior devem ser nomeados usando-se o prefixo de três letras **mnu**, seguido do texto para o menu – por exemplo, **mnuArquivo**, **mnuEditar**, **mnuAjuda**.
- Itens de menus-filhos devem ser nomeados com base no menu de nível superior em que estão contidos, mais o texto do novo item. Você pode querer abreviar ou escolher apenas uma parte do texto, esperando que ainda assim o nome resultante seja exclusivo. Por exemplo, **mnuArquivoAbrir**, **mnuAjudaSobre**, **mnuJanelaHorizontal**. Os itens repetidos do menu, como os separadores (as linhas horizontais usadas para agrupar seções de um menu), devem ser nomeados com o acréscimo de um número no final, a menos que possamos dar a eles um nome exclusivo.

Os Teclados e os Menus

Embora os menus proporcionem acesso aos recursos, na verdade, podem limitá-lo para alguns usuários. Os menus são mesmo úteis, mas podem bloquear os usuários que não tenham experiência ou aptidão de usar o mouse. Por outro lado, os usuários avançados podem não querer alternar constantemente entre o teclado e o mouse. Ao projetar menus para seus aplicativos, lembre-se desses dois grupos.

O suporte ao teclado pode ser adicionado aos menus de duas maneiras. Primeiro, devem ser adicionadas teclas de acesso a todos os itens do menu. As *teclas de acesso* são os itens sublinhados do menu, por exemplo, o A do menu Arquivo. Elas permitirão que as pessoas que não saibam usar um mouse ou não tenham um, utilizem seus menus. Em geral, a primeira letra de um menu ou de um item de menu é uma tecla de acesso. No entanto, para evitar ter vários itens com a mesma tecla de acesso, você pode ter de usar outra letra da palavra. Por exemplo, em muitos produtos da Microsoft, a tecla de acesso do menu Arquivo é a letra A, enquanto o menu Ferramentas utiliza a tecla M.

Dê um clique no menu File criado anteriormente. Altere a propriedade Text para **File**. Um travessão aparecerá embaixo do F. Você também pode adicionar esse sublinhado dando um clique em um item do menu para selecioná-lo, aguardando um pouco e, em seguida, dando um clique novamente para poder editar o texto. Adicione um E comercial (&) antes da letra que quiser que seja a tecla de acesso desse item do menu. Associe teclas de acesso aos menus já criados conforme o descrito na Tabela 16.1.

TABELA 16.1 Teclas de Acesso do Menu de Exemplo (em inglês)

<i>Item do Menu</i>	<i>Tecla de Acesso</i>
File	F
New	N
Exit	x
View	V
Red	R
Green	G
Blue	B



NOTA

Ao observarmos a lista de teclas de acesso, percebemos que a maioria delas é o primeiro caractere, com exceção de Exit (Sair). Nesse caso o padrão é usar as letras X como teclas de acesso.

A segunda maneira de adicionar o suporte do teclado aos menus é por meio de teclas de atalho ou, como são às vezes conhecidas, aceleradores. As teclas de atalho em geral são associadas às

teclas de função (como F1 para ajuda) e são pressionadas junto com a tecla Ctrl – por exemplo, Ctrl+C costuma ser a combinação de teclas que cria o atalho para o comando Copy (Copiar). Elas fornecem aos usuários uma maneira rápida de acessar comandos frequentemente usados. Não são obrigatórias, mas proporcionam atalhos para ajudar os usuários, principalmente os acostumados a lidar exclusivamente com teclados ou os usuários experientes. Embora não tenhamos sempre uma tecla de atalho associada a cada item de menu, você deve empregá-las nos mais usados. Além disso, é bom adicioná-las aos itens do menu em que costumamos vê-las, como nos comandos Cut (Recortar), Copy (Copiar) e Paste (Colar).

As teclas de atalho são adicionadas aos itens de menu por meio da propriedade Shortcut. Selecione Shortcut na lista suspensa. Crie as teclas de atalho como descrito na Tabela 16.2.

TABELA 16.2 Teclas de Atalho

<i>Item do Menu</i>	<i>Atalho</i>
New	CtrlN
Exit	CtrlQ
Red	CtrlR
Green	CtrlG
Blue	CtrlB



NOTA

Ao examinarmos a lista, perceberemos novamente que só uma tecla de atalho parece não seguir o padrão – Exit. Preferiu-se o uso de Ctrl+Q em vez de Ctrl+X porque essa combinação em geral é associada ao comando Recortar (Cut).

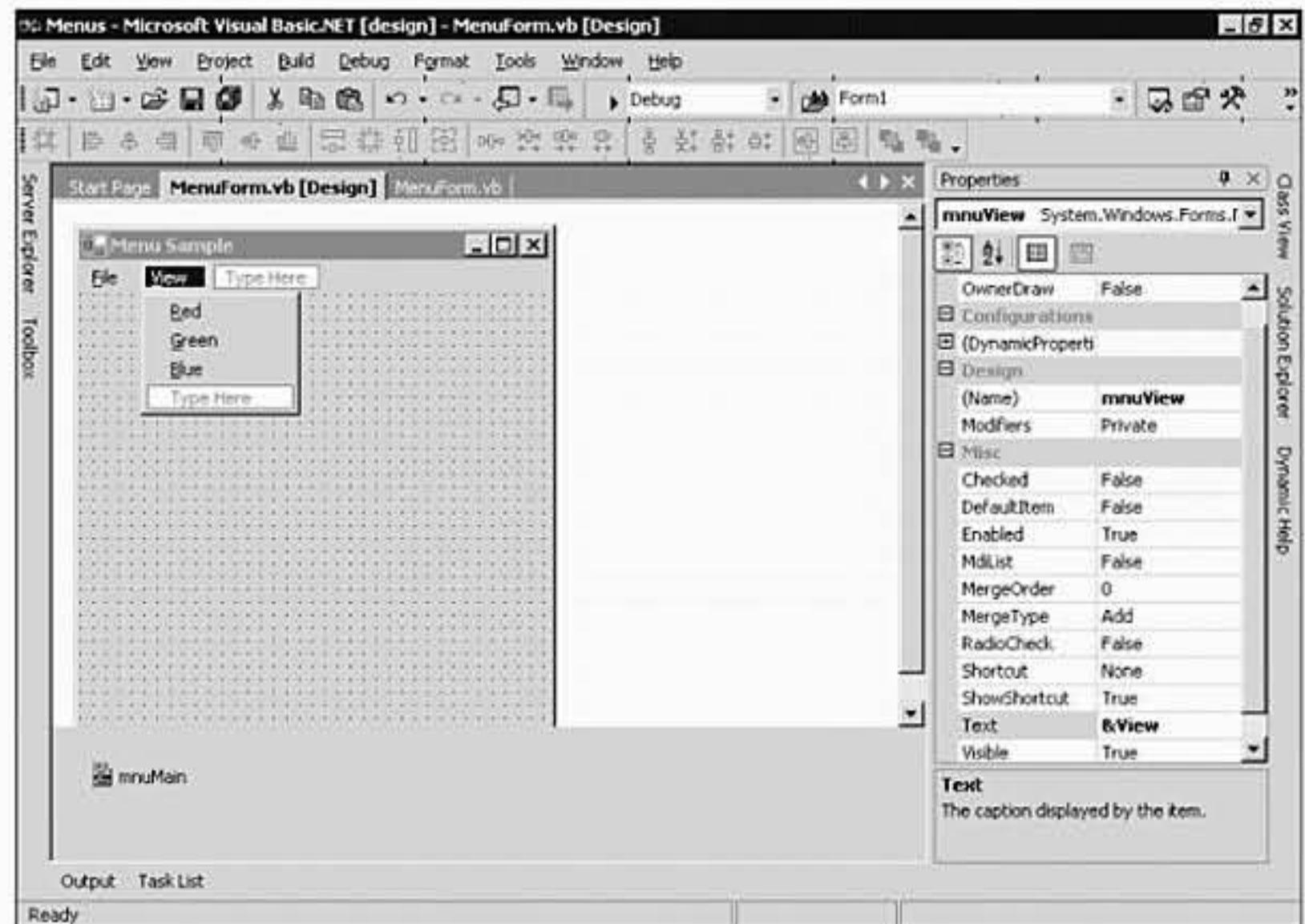
As teclas selecionadas não aparecem no item de menu no tempo de projeto (veja a Figura 16.3); no entanto, surgirão quando o aplicativo for executado (veja a Figura 16.4).

Adicionando Código

Tanto os menus de nível superior quanto os itens de menu dão suporte a vários eventos. Entretanto, em geral só o evento Click é usado. Ele ocorre quando o usuário seleciona o item de menu. Adicione um código ao manipulador desse evento para que ele execute alguma ação que seja necessária. Dê um clique duplo nos itens de menu para abrir a janela de códigos e adicione um ao evento Click. Adicione o código da Listagem 16.1 ao formulário.

FIGURA 16.3

Teclas de atalho no tempo de projeto.

**FIGURA 16.4**

Teclas de atalho no tempo de execução.



LISTAGEM 16.1 Código para o Menu de Exemplo

```

1 Private Sub mnuFileNew_Click(ByVal sender As System.Object, _
2     ByVal e As System.EventArgs) Handles mnuFileNew.Click
3     Dim frmNew As New frmMenu()
4     frmNew.Show()
5 End Sub
6 Private Sub mnuFileExit_Click(ByVal sender As Object, _
7     ByVal e As System.EventArgs) Handles mnuFileExit.Click
8     Me.Close()
9 End Sub
10 Private Sub mnuViewRed_Click(ByVal sender As System.Object, _
11     ByVal e As System.EventArgs) Handles mnuViewRed.Click
12     Me.BackColor = Color.Red

```


LISTAGEM 16.1 Código para o Menu de Exemplo (*continuação*)

```

13 End Sub
14 Private Sub mnuViewGreen_Click(ByVal sender As System.Object, _
15     ByVal e As System.EventArgs) Handles mnuViewGreen.Click
16     Me.BackColor = Color.Green
17 End Sub
18 Private Sub mnuViewBlue_Click(ByVal sender As System.Object, _
19     ByVal e As System.EventArgs) Handles mnuViewBlue.Click
20     Me.BackColor = Color.Blue
21 End Sub

```

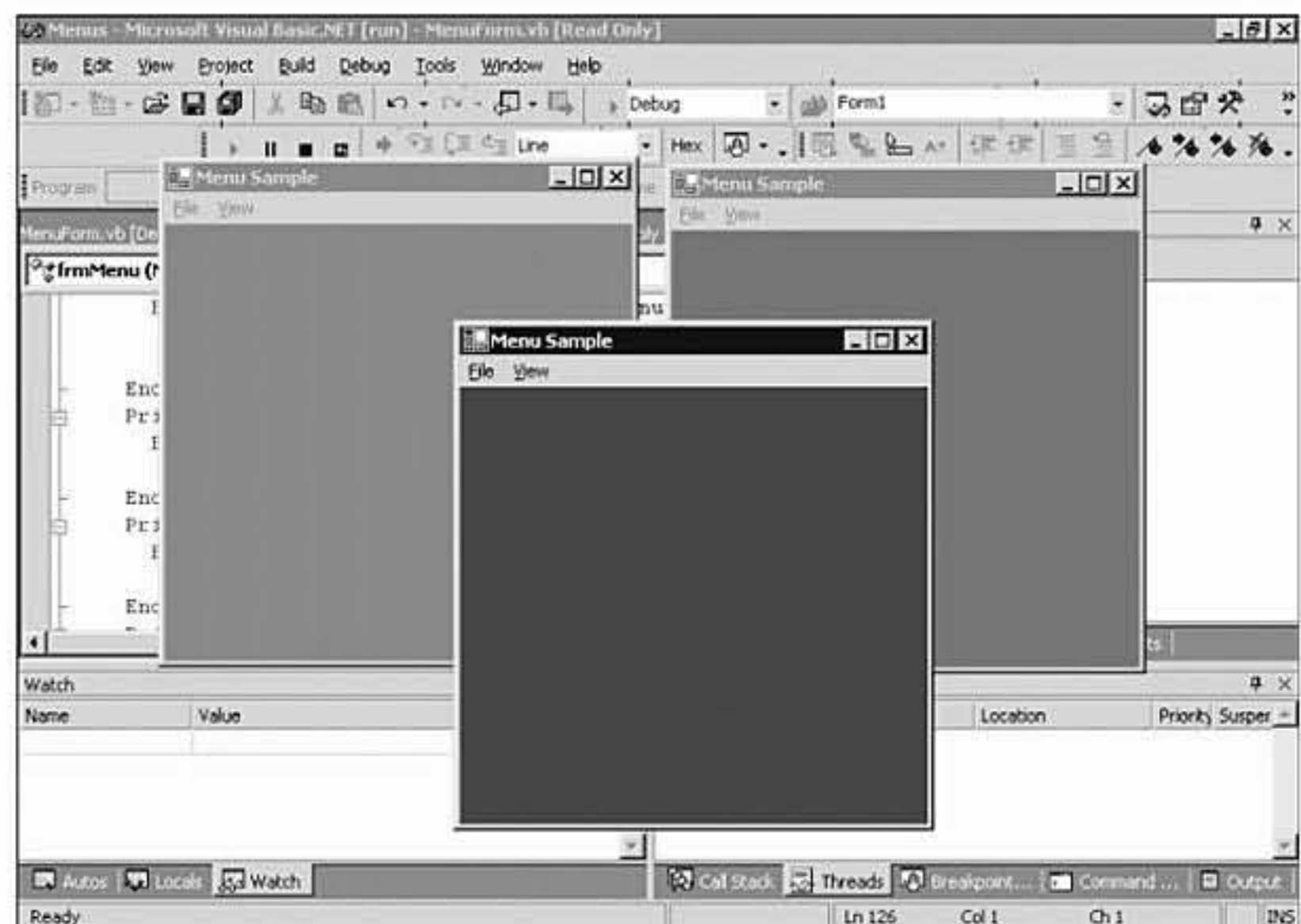
ANÁLISE

O manipulador de eventos `mnuFileNew_Click` está relacionado aos itens `File` e `New` do menu. O código para esse evento (linhas 1 a 5) cria uma nova instância do formulário atual e a exibe. Isso permite que você veja várias cópias do formulário simultaneamente. Por outro lado, o código do evento `mnuFileExit_Click` (itens `File` e `Exit` das linhas 6 a 9) fecha o formulário ativo. O aplicativo é encerrado quando o último formulário é fechado. Os três manipuladores de eventos finais estão relacionados aos itens `Red`, `Green` e `Blue` do menu. Eles apenas alteram a cor do plano de fundo do formulário ativo para a cor que foi selecionada.

Compile e execute o aplicativo. Use os itens `File` e `New` (ou pressione `Ctrl+N`) para abrir várias telas do menu. Tente alterar a cor dos formulários usando os itens `Red`, `Green` e `Blue` do menu (lembre-se de que também existem teclas de atalho para esses itens). A Figura 16.5 mostra o aplicativo em execução. Para concluir, feche todos os formulários usando o item `Exit` do menu para encerrar o aplicativo.

FIGURA 16.5

Executando o menu de exemplo.

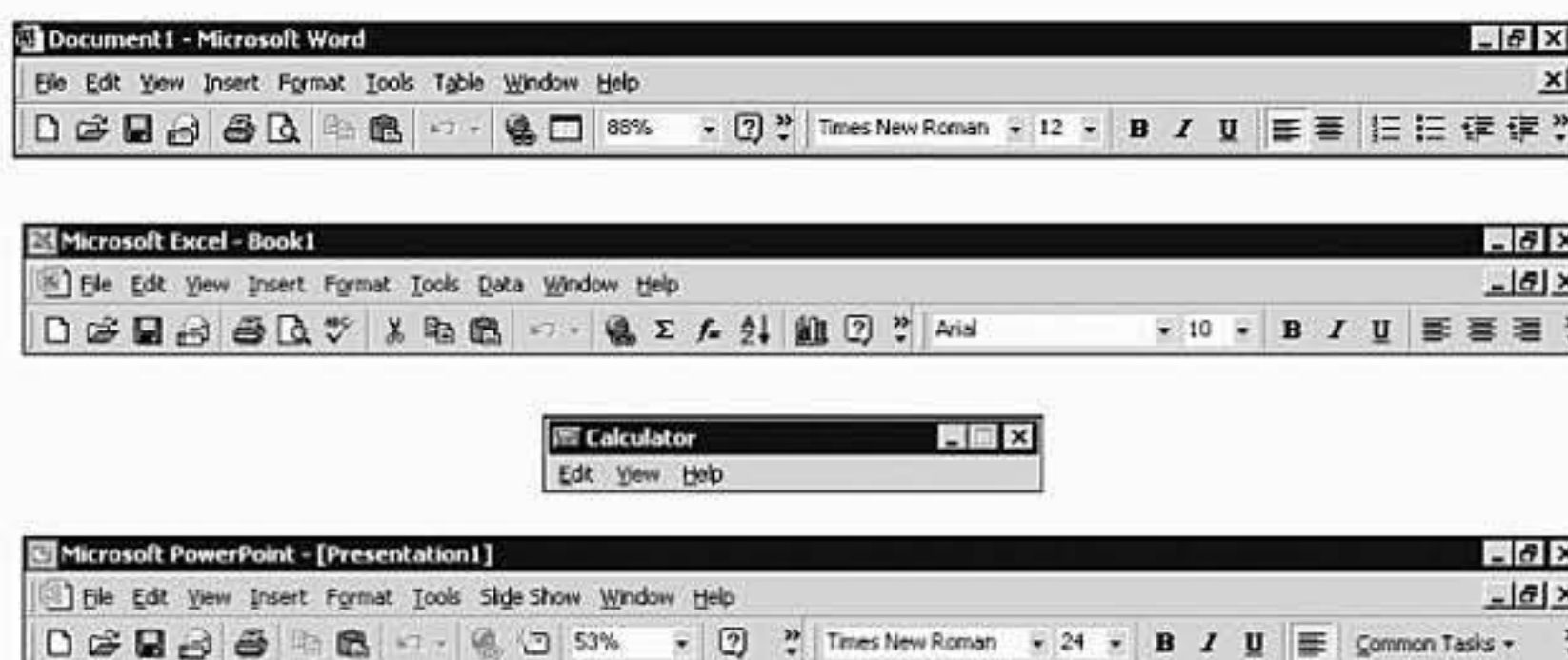


Algumas Sugestões

Como você viu, os menus fornecem uma maneira comum de acesso aos recursos de seu programa. Contudo, se tornam mais úteis quando as pessoas encontram os comandos no mesmo local em todos os programas. Portanto, será uma boa prática configurar seus menus de maneira semelhante à de outros aplicativos. Isso ajudará seus usuários a conhecerem seu aplicativo porque eles encontrarão os comandos mais rapidamente se esses estiverem onde se espera encontrá-los.

Isso significa que você deve ter alguns menus comuns e que eles devem estar localizados no mesmo lugar em que são colocados em outros programas. A Figura 16.6 mostra alguns dos principais menus comuns de programas como o Word, o Excel, o PowerPoint, a Calculadora (todos em inglês) e o Visual Basic .NET. Tente reproduzir aplicativos como esses, se isso for apropriado para o seu aplicativo.

FIGURA 16.6
Menus comuns.



É claro que se seu aplicativo não usar arquivos ou tiver telas alternativas, você poderá não achar apropriados alguns desses menus. Por exemplo, o aplicativo Calculadora que vem no Windows não tem um menu File (Arquivo). Da mesma maneira, muitos aplicativos não precisam de outros menus comuns, como Edit (Editar), Tools (Ferramentas) ou Help (Ajuda). Não será necessário incluir esses menus se não precisarmos deles; no entanto, se eles realmente existirem, devem ficar nos locais esperados.

Exatamente como há menus comuns de nível superior, você também deve construir menus para seus aplicativos de modo que sejam semelhantes ao de outros aplicativos utilizados pelos seus usuários. Por exemplo, o menu Arquivo em geral deve conter os itens Novo, Abrir, Salvar e Sair. Da mesma maneira, o menu Editar deve apresentar os itens Recortar, Copiar e Colar.

Faça

Examine outros aplicativos para obter sugestões de estruturas de menus comuns quando definir os seus. Use essas estruturas de menus para ajudar seus usuários a encontrar comandos de que precisem.

Não Faça

NÃO encare isso como uma regra absoluta. Se um menu não for apropriado, não hesite em deixá-lo de fora. Por exemplo, se seu aplicativo não usar arquivos, você pode evitar a utilização do menu Arquivo. No entanto, ainda seria possível incluir o item Sair como o último item do primeiro menu.

Programas de Interface de Documentos Múltiplos

16

Com frequência, precisamos criar um programa que lide com vários itens semelhantes, como um editor de texto que precise de várias janelas abertas ou uma ferramenta de banco de dados que permita ao usuário visualizar os resultados de diversas consultas. Embora não seja obrigatório, esses tipos de programas podem ser projetados como um aplicativo MDI (Multiple Document Interface) para ajudar seus usuários a visualizar o relacionamento entre as janelas abertas. A Figura 16.7 mostra um aplicativo MDI.

FIGURA 16.7

Aplicativo com interface de documentos múltiplos.



O Que É uma Interface de Documentos Múltiplos?

Para descrever a MDI, devemos compará-la com um aplicativo SDI (Single Document Interface – interface de apenas um documento). Em um aplicativo comum, como o Bloco de notas, cada

documento aberto é exibido em sua própria janela. Portanto, um formulário contém apenas um documento. Em um aplicativo MDI, pressupõe-se que um formulário possa conter vários documentos. Basicamente, isso é verdade. No entanto, uma definição melhor seria que uma janela contém vários outros formulários, cada um com apenas um documento. As versões mais antigas do Microsoft Word usavam essa estratégia. De maneira semelhante, o próprio Visual Basic .NET é um aplicativo MDI. Você pode ter vários documentos abertos ao mesmo tempo. Cada um deles é exibido em sua própria janela dentro da janela-pai do Visual Basic .NET.

Adicionando o Formulário-pai

A única diferença entre criar um aplicativo Windows comum e um aplicativo MDI é o contêiner da MDI. O contêiner ou formulário-pai é o formulário de seu aplicativo que contém seus filhos. Nenhum formulário interno pode ser retirado do pai MDI. Temos duas razões para isso:

- Ele diminui as chances de um ou mais formulários serem perdidos por usuários iniciantes. Os usuários menos familiarizados com a navegação no Windows às vezes perdem um formulário selecionando outro e em geral não conseguem localizar o anterior usando a barra de tarefas.
- Reforça a idéia de que os formulários-filhos estão relacionados entre si e associados ao formulário-pai. Se os formulários não puderem ser transferidos de modo independente, serão considerados como dependentes uns dos outros.

Em geral, todos os outros formulários do aplicativo são filhos desse formulário-pai. No entanto, você pode não querer que alguns deles sejam um filho. Isso seria desejável se o formulário não estivesse contido logicamente no formulário-pai. Por exemplo, no Visual Basic .NET, uma janela de código está logicamente contida dentro dele e, portanto, deve ser um formulário-filho. Como caixa de diálogo para abrir arquivos, por outro lado, não está logicamente contida no aplicativo e, portanto, não é uma janela-filha.

Você pode criar um novo formulário-pai MDI configurando a propriedade `IsMdiContainer` como `True`. Fazer isso adicionará um novo controle ao formulário, um controle `MdiClient`. Esse controle ocupa todo o formulário e é o contêiner dos formulários-filhos.

Depois que você tiver um formulário-pai MDI, poderá tornar outros formulários-filhos dele configurando sua propriedade `MdiParent` antes de exibí-los. Configure essa propriedade para apontar para o formulário-pai. A Listagem 16.2 mostra um exemplo disso.

LISTAGEM 16.2 Criando um Formulário-filho

```
1 Private Sub mnuFileOpen_Click(_  
2     ByVal sender As System.Object,_  
3     ByVal e As System.EventArgs)_  
4     Handles mnuFileOpen.Click  
5  
6     Dim oForm As New frmView()
```


LISTAGEM 16.2 Criando um Formulário-filho (*continuação*)

```
7      oForm.MdiParent = Me
8      oForm.Show()
9  End Sub
```

ANÁLISE

Para exibir outro formulário, a primeira etapa é criar uma nova instância do formulário desejado (linha 6). A seguir, a propriedade `MdiParent` é configurada como formulário-pai. Esse formulário deve ter sua propriedade `IsMdiContainer` configurada como `True`. Para concluir, o formulário é exibido (linha 8). Ele não pode ser removido dos limites do formulário-pai.

**NOTA**

Embora os aplicativos MDI forneçam muitos benefícios, você pode não querer usá-los na maioria de seus aplicativos. A Microsoft reduziu bastante o uso que faz de aplicativos MDI, declarando que muitos usuários iniciantes ficam confusos por causa deles. Por exemplo, tanto o Microsoft Word quanto o Excel eram aplicativos MDI. No entanto, agora são aplicativos SDI e geram uma janela por documento.

Portanto, como qualquer tecnologia, use a MDI onde apropriado e quando seus usuários a compreenderem.

A MDI e os Menus

Como é de se esperar, quando apenas uma janela puder conter várias janelas-filhas diferentes, os aplicativos MDI possuirão configurações especiais de menu. Elas permitirão que você organize as janelas-filhas mais facilmente. Em geral, os aplicativos MDI possuem um menu `Window` (Janela) que dá acesso a esses comandos.

O menu `Window` padrão tem quatro comandos:

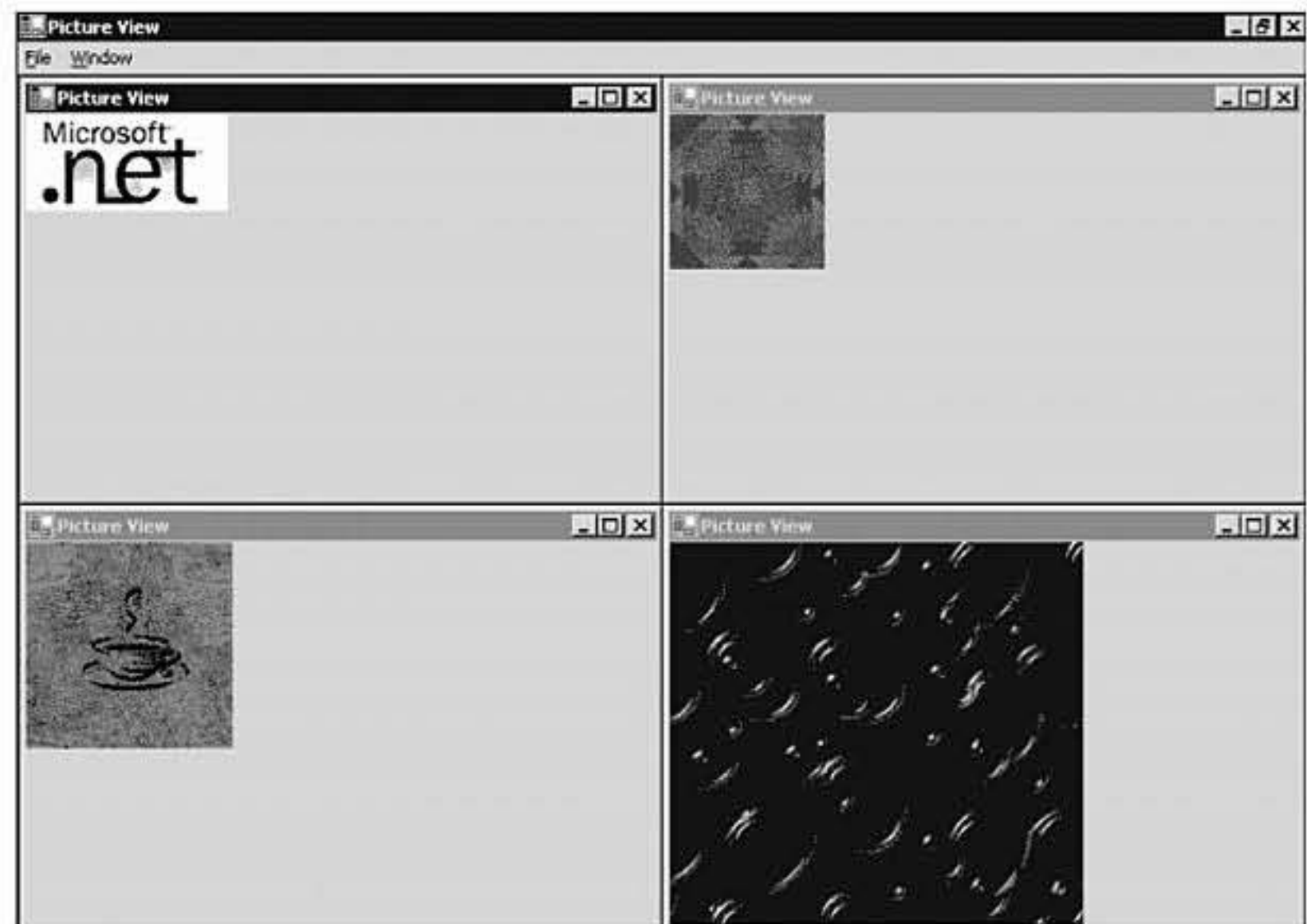
- **Tile Horizontally** (Lado a lado horizontalmente) Organiza as janelas-filhas de modo que cada uma seja uma faixa horizontal ocupando a tela (veja a Figura 16.8). Isso será útil quando você tiver de comparar informações que precisem ser exibidas em uma linha inteira, como dois documentos para o processamento de textos.
- **Tile Vertically** (Lado a lado verticalmente) Organiza as janelas-filhas de modo que cada uma seja uma faixa vertical do lado da outra ao ocupar a tela (veja a Figura 16.9). Isso será útil ao se comparar pequenas áreas de dois ou mais documentos ou quando você executar uma varredura rápida em dois documentos para encontrar alterações.

FIGURA 16.8

*O comando Tile
Horizontally aplicado
aos formulários-filhos.*

**FIGURA 16.9**

*O comando Tile
Vertically aplicado aos
formulários-filhos.*



- **Cascade (Em cascata)** Organiza as janelas-filhas de modo que tenham todas o mesmo tamanho. Cada janela é posicionada um pouco abaixo e à direita das que estão atrás dela, resultando em uma disposição na qual você pode visualizar as barras de título de cada janela-filha (veja a Figura 16.10). Isso é útil para demonstrar que documentos estão abertos no momento.

FIGURA 16.10

Formulários-filhos em cascata.



- **Arrange Icons (Organizar ícones)** Organiza todas as janelas minimizadas de modo que fiquem dispostas em linha ao longo da parte inferior da janela-pai (veja a Figura 16.11). É útil para organizar janelas-filhas que não estejam em uso no momento.

FIGURA 16.11

O comando Arrange Icons aplicado aos formulários-filhos.



Além dos itens comuns, o menu Window normalmente também apresenta uma lista de todas as janelas-filhas (veja a Figura 16.12). Em geral esse recurso é chamado de lista de janelas. Ele lista todas as janelas-filhas abertas e mantém uma marca de seleção próxima à que estiver ativa. Em vez de você mesmo desenvolver e fazer a manutenção do código que produzirá esse resultado, o

Visual Basic .NET facilita a inclusão dessa lista. Apenas configure a propriedade `MidList` como `True` para o menu de nível superior onde quiser adicionar a lista de janelas.

FIGURA 16.12

Menu com a lista de janelas.



Para saber como se cria um aplicativo MDI, você desenvolverá um visualizador simples de figuras. Crie um novo aplicativo Windows e chame-o de `MultiPicView`. Altere o nome do formulário no Solution Explorer para `MdiParent`. Abra a tela de códigos do formulário e altere o nome da classe de `Form1` para `MdiParent`. Para concluir, abra as propriedades do projeto e configure `Startup Object` como `MdiParent`. Configure as propriedades do formulário como mostra a Tabela 16.3.

TABELA 16.3 Propriedades do Formulário MDI-pai

Propriedade	Valor
Size	480, 360
Text	Picture View
IsMdiContainer	True

O usuário poderá acessar um menu que lhe permitirá abrir novos arquivos a fim de visualizá-los. Adicione os controles `MainMenu` e `OpenFileDialog` ao formulário MDI-pai. Renomeie o objeto `MainMenu` como `mnuMain` e construa o menu como mostra a Tabela 16.4. Configure as propriedades de `OpenFileDialog` como descrito na Tabela 16.5.

TABELA 16.4 Estrutura do Menu do Formulário MDI-pai

Item do Menu	Propriedade	Valor
Nível superior	(Name)	MnuFile
	Text	File
Abaixo de File	(Name)	MnuFileOpen
	Text	&Open
	Shortcut	Ctrl+O
Abaixo de Open	(Name)	MnuFileSep
	Text	-
Abaixo do separador	(Name)	MnuFileExit
	Text	E&xit
	Shortcut	Ctrl+Q

TABELA 16.4 Estrutura do Menu do Formulário MDI-pai (*continuação*)

<i>Item do Menu</i>	<i>Propriedade</i>	<i>Valor</i>
Menu de nível superior	(Name)	MnuWindow
	Text	&Window
	MdiList	True
Abaixo de Window	(Name)	MnuWindowHorizontal
	Text	Tile &Horizontally
Abaixo de Tile Horizontally	(Name)	MnuWindowVertical
	Text	Tile &Vertically
Abaixo de Tile Vertically	(Name)	MnuWindowCascade
	Text	&Cascade
Abaixo de Cascade	(Name)	MnuWindowArrange
	Text	&Arrange Icons

TABELA 16.5 Propriedades de OpenFileDialog

<i>Propriedade</i>	<i>Valor</i>
(Name)	DlgOpen
Filter	Graphics Files *.gif;*.bmp;*.jpg All files *.*

O menu File será usado para abrir novos arquivos de figuras e para encerrar o aplicativo. Adicione o código da Listagem 16.3 ao formulário para os itens do menu.

LISTAGEM 16.3 Código para os Comandos do Menu File de MultiPicView

```

1 Private Sub mnuFileOpen_Click(_
2     ByVal sender As System.Object,_
3     ByVal e As System.EventArgs)_
4     Handles mnuFileOpen.Click
5
6     If dlgOpen.ShowDialog() = DialogResult.OK Then
7         Dim oForm As New frmView()
8         oForm.MdiParent = Me
9         oForm.Picture = Image.FromFile(dlgOpen.FileName)
10        oForm.Show()
11    End If
12 End Sub

```


LISTAGEM 16.3 Código para os Comandos do Menu File de MultiPicView (*continuação*)

```
13
14 Private Sub mnuFileExit_Click(_
15     ByVal sender As System.Object,_
16     ByVal e As System.EventArgs)_
17     Handles mnuFileExit.Click
18
19     Me.Close()
20
21 End Sub
```

ANÁLISE

O código deve ser semelhante ao que já vimos. O manipulador do evento File, Open exibe OpenFileDialog (linha 6). Se o usuário selecionar um arquivo, o código criará um novo formulário frmView, que geraremos logo após, e configurará sua propriedade MdiParent com o formulário atual (linha 8). Em seguida, ele configura sua propriedade Picture com o arquivo das figuras e exibe o novo formulário (linha 10). A figura é carregada na linha 9 por meio do método FromFile da classe Image que recupera uma figura armazenada em um arquivo.

O manipulador do evento File, Exit é ainda mais simples, fechando o formulário atual. Isso faz com que todas as janelas-filhas abertas também sejam fechadas, encerrando o aplicativo.

Além de escrever o código do menu File, você também deve criar o código dos comandos do menu Window. A Listagem 16.4 mostra esse código.

LISTAGEM 16.4 Código para os Comandos do Menu Windows de MultiPicView

```
22 Private Sub mnuWindowHorizontal_Click(_
23     ByVal sender As System.Object,_
24     ByVal e As System.EventArgs)_
25     Handles mnuWindowHorizontal.Click
26
27     Me.LayoutMdi(MdiLayout.TileHorizontal)
28 End Sub
29
30 Private Sub mnuWindowVertical_Click(_
31     ByVal sender As System.Object,_
32     ByVal e As System.EventArgs)_
33     Handles mnuWindowVertical.Click
34
35     Me.LayoutMdi(MdiLayout.TileVertical)
36 End Sub
37
38 Private Sub mnuWindowCascade_Click(_
39     ByVal sender As System.Object,_
```


LISTAGEM 16.4 Código para os Comandos do Menu Windows de MultiPicView
(continuação)

```

40      ByVal e As System.EventArgs) _
41      Handles mnuWindowCascade.Click
42
43      Me.LayoutMdi(MdiLayout.Cascade)
44  End Sub
45
46  Private Sub mnuWindowArrange_Click(_
47      ByVal sender As System.Object, _
48      ByVal e As System.EventArgs) _
49      Handles mnuWindowArrange.Click
50
51      Me.LayoutMdi(MdiLayout.ArrangeIcons)
52  End Sub

```

16

ANÁLISE

Cada um dos quatro menus Window apenas executa o método `LayoutMdi` do formulário MDI-pai, configurando a disposição desejada para os formulários-filhos.

Agora você está pronto para criar o formulário que será usado para exibir as figuras. Adicione um segundo formulário ao projeto selecionando os itens `Project` e `Add Windows Form` no menu. Quando o nome for solicitado, nomeie o novo formulário como `PictureView`. Altere sua propriedade `Text` para `Picture View`. Adicione um controle `PictureBox` ao formulário e configure as propriedades desse controle como mostra a Tabela 16.6.

TABELA 16.6 Propriedades do Controle `PictureBox`

Propriedade	Valor
(Name)	PicView
Dock	Fill

Acesse a tela do código e adicione uma propriedade pública ao formulário chamada `Picture` que representa a propriedade `Image` de `PictureBox`. Devemos terminar com algo semelhante ao código da Listagem 16.5.

LISTAGEM 16.5 Propriedade `Picture`

```

1  Public Property Picture() As Image
2      Get
3          Return picView.Image
4      End Get
5      Set(ByVal Value As Image)
6          picView.Image = Value

```


LISTAGEM 16.5 Propriedade Picture (*continuação*)

```
7      End Set  
8      End Property
```

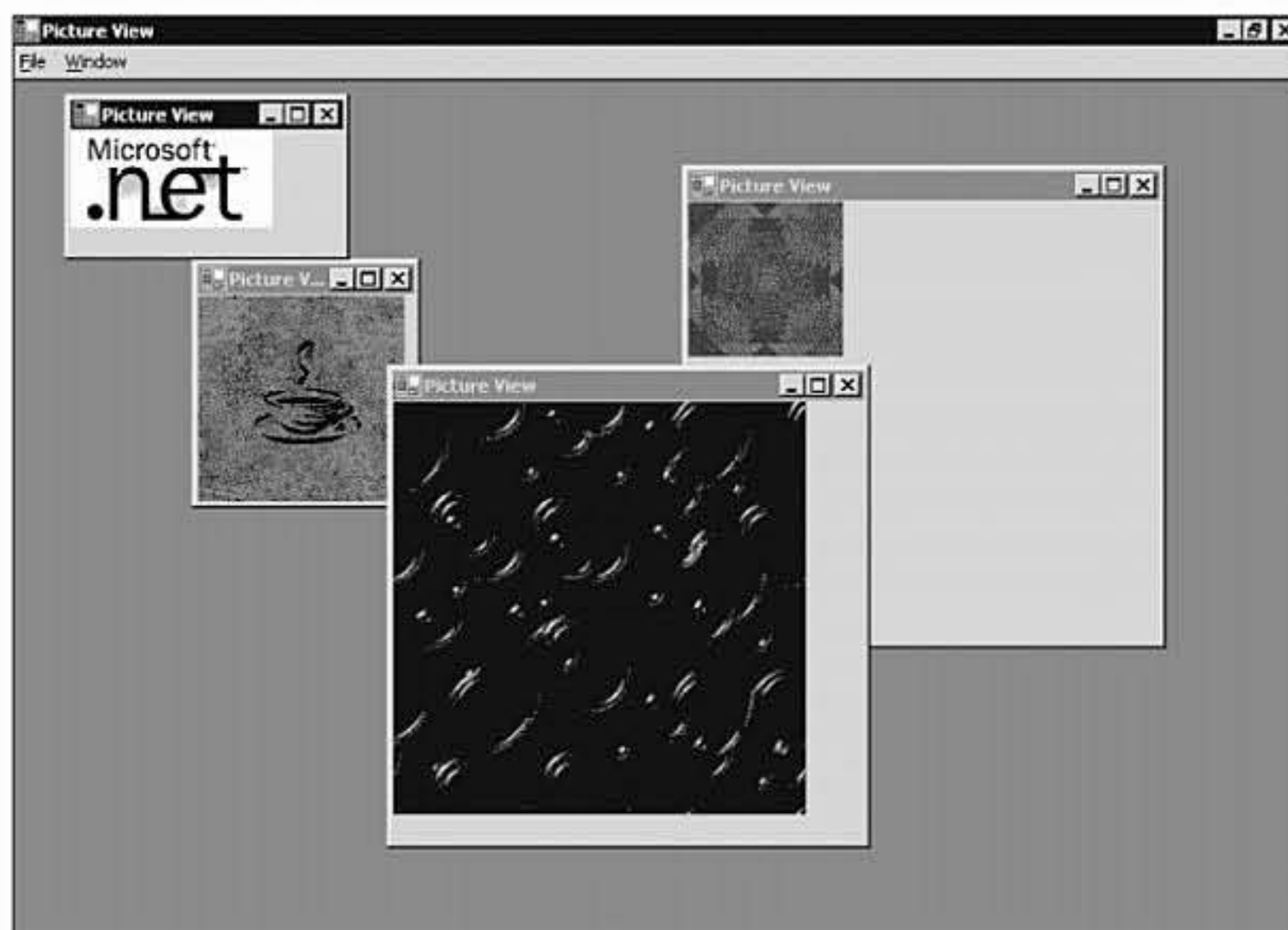
ANÁLISE

Do mesmo modo que você pode ter propriedades que representem valores privados armazenados dentro da classe Form, também é possível ter propriedades que representem os controles no formulário ou propriedades exclusivamente relacionadas aos controles. Nesse caso, a propriedade Picture foi usada para acessar a propriedade Image do controle `picView`. Embora isso pudesse ter sido resolvido se tornássemos público o controle, seria proporcionado um nível muito amplo de acesso a ele. Isso permite que o código leia ou altere com facilidade a figura, sem deixar que o código de um formulário diferente modifique outras informações sobre `PictureBox`.

Execute o aplicativo. Provavelmente você poderá abrir vários arquivos de figuras, como vemos na Figura 16.13.

FIGURA 16.13

MultiPicView em ação.



Controles Avançados dos Formulários Windows

Embora seja quase sempre possível desenvolver um aplicativo usando apenas os controles mais comuns, como `Label`, `TextBox` e `PictureBox`, você provavelmente precisará de outros em certas

situações. Existem muitos controles disponíveis no Visual Basic .NET que não foram abordados neste livro e vários outros podem ser comprados. Seria difícil, se não impossível, abordar todos os controles disponíveis em um único livro, ou como parte desta lição. Portanto, mostrarei alguns dos mais importantes e como usá-los.

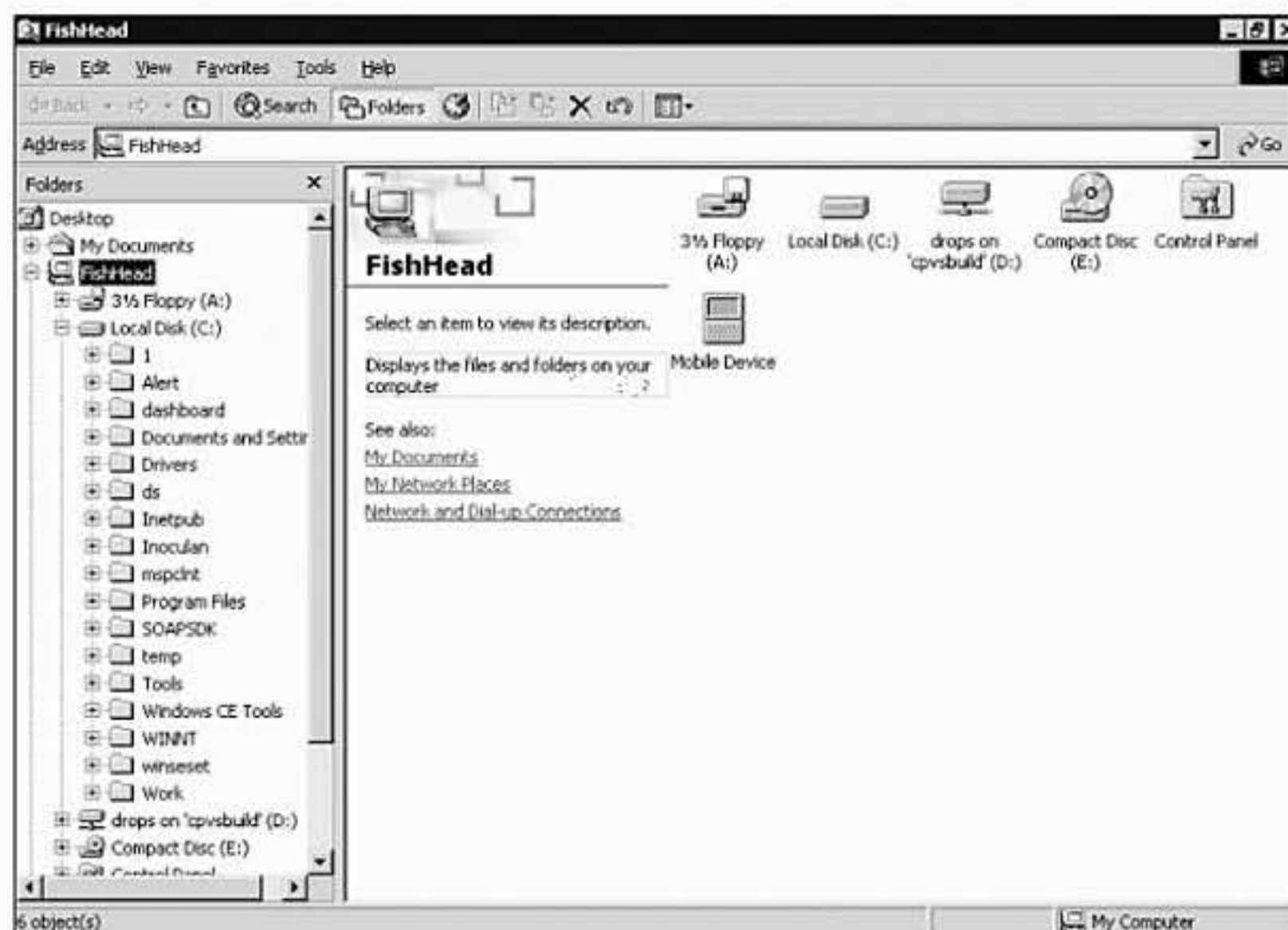
TreeView

O controle `TreeView` deve ser familiar para qualquer pessoa que tenha usado o Windows durante algum tempo. É o que aparece no lado esquerdo da janela do Explorer, assim como em qualquer outro lugar em que um modelo hierárquico é exibido. Ele permite que você mostre ao usuário uma lista de itens e seus relacionamentos. As figuras exibidas para cada item da lista podem ser alteradas ou não exibidas de maneira alguma. De maneira semelhante, as linhas que conectam os itens podem ser personalizadas ou removidas. O controle `TreeView` deve ser considerado sempre que for preciso exibir muitos itens relacionados e mostrar esses relacionamentos para os usuários – por exemplo, para permitir que o usuário pesquise os diretórios, as pastas de correio eletrônico ou crie a estrutura de um documento. A Figura 16.14 mostra o controle `TreeView` em uso, exibindo uma lista de pastas de um disco rígido.

16

FIGURA 16.14

O controle `TreeView` em ação.



O controle `TreeView` possui muitas propriedades, métodos e eventos. A Tabela 16.7 descreve aqueles usados com mais frequência:

TABELA 16.7 Membros do Controle TreeView

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Checkboxes	Adiciona caixas de texto a cada item do controle TreeView. Isso facilita a criação de uma seleção múltipla.
FullRowSelect	Se esta propriedade for igual a True, a linha inteira será selecionada quando um item for marcado. Como alternativa, apenas o texto de cada item marcado será selecionado.
Nodes	Esta é a propriedade mais importante do controle TreeView. Ela contém a lista de todos os itens de nível superior desse controle.
PathSeparator	Usada quando um caminho é recuperado em um nó do controle TreeView. Esse caractere é empregado entre cada um dos nós. O padrão utilizado é a barra invertida (\).
SelectedNode	O nó selecionado atualmente no controle TreeView.
ShowLines	Determina se serão traçadas linhas entre os nós do controle TreeView.
ShowPlusMinus	Determina se os caracteres + e – serão exibidos próximos aos nós do controle TreeView.
ShowRootLines	Determina se serão traçadas linhas para conectar os nós do nível superior do controle TreeView.
Sorted	Determina se os itens adicionados ao controle TreeView devem ser ordenados quando inseridos.
Métodos	
CollapseAll	Fecha todos os três nós.
ExpandAll	Expande todos os três nós.
Eventos	
BeforeCollapse	Ocorre antes que uma subárvore seja fechada. Pode ser usado para cancelar o evento ou atualizar outros controles.
BeforeExpand	Ocorre antes que uma subárvore seja aberta. É um ótimo manipulador de eventos para conduzir à atualização os itens da subárvore antes de executar a exibição para o usuário.
BeforeSelect	Ocorre antes que um nó seja selecionado.
AfterCollapse	Ocorre depois que uma subárvore foi fechada. Esse é um bom momento se você precisar liberar os recursos usados pela subárvore, como encerrar as conexões com bancos de dados.
AfterExpand	Ocorre depois que uma subárvore é aberta. Pode ser usado para atualizar outros controles com os nós da árvore recentemente exibidos.
AfterSelect	Ocorre depois que um nó da árvore é selecionado. Pode ser usado para atualizar outros controles com base no nó selecionado.

Como descrito na Tabela 16.7, Nodes é a propriedade mais importante do controle TreeView. Cada item desse conjunto é um objeto TreeNode, e esse objeto é o ponto central de grande parte do que é executado com o controle TreeView. A Tabela 16.8 descreve as propriedades e métodos importantes da classe TreeNode.

TABELA 16.8 Propriedades e Métodos de TreeNode

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Checked	Usada junto com a propriedade Checkboxes do controle TreeView. Esta propriedade será igual a True se o objeto TreeNode for selecionado.
FullPath	Retorna uma string que contém cada um dos nós abaixo do que foi selecionado, todos separados pela propriedade PathSeparator do controle TreeView. Em geral, isso cria uma string semelhante a um caminho.
Nodes	O conjunto de TreeNodes desse TreeNode.
Text	O texto desse TreeNode.
Métodos	
Collapse	Recolhe a árvore começando com o nó atual.
Expand	Expande a árvore começando com o nó selecionado.
Toggle	Altera a árvore começando com o nó selecionado para minimizado ou expandido; o estado em que não estiver no momento.

Não são só os objetos TreeNode e todo o controle TreeView que devem ser considerados importantes, um objeto adicional também o é quando lidamos com esse controle. Ele é o conjunto representado pelas propriedades Nodes – os objetos TreeNodeCollection. Um desses conjuntos se encontra no nível máximo da árvore e é refletido na propriedade Nodes de TreeView. Além disso, cada TreeNode desse conjunto possui seu próprio conjunto Nodes, representando todos os nós-filhos desse TreeNode. A Tabela 16.9 descreve algumas das propriedades e métodos desse importante conjunto.

TABELA 16.9 Métodos e Propriedades de TreeNodeCollection

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Count	Determina quantos itens existem no conjunto.
Item	Retorna um dos TreeNodes do conjunto. Esta propriedade pode ser aninhada – por exemplo, <code>twvFolders.Nodes.Item(3).Nodes.Item(1)</code> seria o segundo nó abaixo do quarto nó de TreeView. (Lembre-se de que os conjuntos começam em 0.)
Métodos	
Add	Adiciona um novo nó ao final do conjunto atual. Pode ser um objeto TreeNode ou apenas o texto a ser exibido para o novo nó.
Clear	Remove todos os nós do conjunto selecionado.
IndexOf	Retorna o número referente à posição do nó selecionado no conjunto.
Insert	Adiciona um novo TreeNode à lista de nós em uma posição definida.
Remove	Remove do conjunto um nó solicitado.
RemoveAt	Remove um nó pelo índice no conjunto.

ListVie

O controle `ListView` também deve ser familiar para a maioria dos usuários do Windows como o lado esquerdo da janela Explorer, assim como em muitos outros locais. Ele é semelhante a `List-Box`, no fato de poder conter vários itens. No entanto, fornece mais recursos, já que permite a exibição dos itens de diversas maneiras. Os itens podem ser exibidos como uma lista, associados a ícones grandes ou pequenos. Também é possível disponibilizar uma lista com detalhes adicionais de cada item. Para concluir, podemos usar `ListView` no lugar de um controle de grade porque ele também pode exibir linhas de grade. A Tabela 16.10 mostra algumas das propriedades, métodos e eventos mais importantes do controle `ListView`.

TABELA 16.10 Propriedades, Métodos e Eventos Importantes do Controle `ListView`

<i>Membro</i>	<i>Descrição</i>
Propriedades	
<code>Checkboxes</code>	Determina se as caixas de seleção serão exibidas próximas a cada item da lista. Essa é uma maneira excelente de criar uma lista com seleção múltipla.
<code>CheckIndices</code>	Retorna o valor do índice de cada item selecionado em <code>ListView</code> . Isso permite que você saiba que itens foram selecionados quando usar <code>ListView</code> para criar listas com seleção múltipla.
<code>Columns</code>	Quando <code>ListView</code> estiver no modo de exibição 'Detalhe', as colunas poderão aparecer para que a lista fique organizada.
<code>FullRowSelect</code>	Determina se a linha inteira ou apenas o texto do item será marcado quando um item for selecionado.
<code>GridLines</code>	Determina se as linhas de grade serão exibidas quando <code>ListView</code> estiver no modo de exibição 'Detalhe'. Isso permitirá que você faça <code>ListView</code> se parecer com um simples controle <code>Grid</code> .
<code>Items</code>	Representa o conjunto de todos os itens de <code>ListView</code> .
<code>View</code>	Determina como <code>ListView</code> será exibido. As opções possíveis são <code>Large Icons</code> , <code>Small Icons</code> , <code>List</code> ou <code>Detail</code> .
Métodos	
<code>Clear</code>	Remove todos os itens de <code>ListView</code> .
Eventos	
<code>ColumnClick</code>	Ocorre quando o usuário dá um clique no cabeçalho de uma coluna.
<code>SelectedIndex-Changed</code>	Ocorre quando o usuário seleciona um novo item em <code>ListView</code> . Permite que você atualize outros controles com base nessa seleção.

Pode ser usado para alterar a ordem das colunas. Exatamente como o conjunto `Nodes` do controle `TreeView`, a propriedade mais importante de `ListView` é o conjunto `Items`. Cada item do conjunto `Items` é um objeto `ListViewItem`. Esse objeto, por sua vez, possui métodos e propriedades que representam o que é possível fazer com cada item de `ListView`. A Tabela 16.11 descreve as mais importantes entre essas propriedades.

TABELA 16.11 Propriedades de ListViewItem

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Checked	Usada junto com a propriedade Checkboxes de ListView, esta propriedade será igual a True se um item for selecionado.
Index	Retorna a posição do item em ListView.
Selected	Determina se ListViewItem foi selecionado.
SubItems	Usada junto com o modo de exibição 'Detalhes' e a propriedade Columns. Esta propriedade contém as informações que serão exibidas nas colunas adicionais.

Enquanto TreeView possui objetos TreeNodeCollection refletindo os conjuntos de objetos TreeNode, ListView apresenta objetos ListViewItemCollection. Entretanto, esse conjunto é menos importante do que a propriedade que corresponde a ele em TreeView, já que ListView não apresenta a mesma hierarquia de nós. Em vez disso, há basicamente apenas um objeto ListViewItemCollection. A Tabela 16.2 descreve algumas das propriedades e métodos importantes desse conjunto.

TABELA 16.12 Propriedades e Métodos de ListViewItemCollection

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Count	A quantidade de itens em ListViewCollection.
Item	Retorna o objeto ListViewItem da posição solicitada.
Métodos	
Add	Adiciona um novo objeto ListViewItem ao final do conjunto. O novo item pode ser um objeto ListViewItem ou o texto a ser exibido para o item.
Clear	Remove todos os itens do conjunto.
IndexOf	Retorna o índice, ou posição, do objeto ListViewItem solicitado.
Insert	Adiciona um novo objeto ListViewItem ao conjunto em uma posição solicitada.
Remove	Remove um objeto ListViewItem do conjunto.
RemoveAt	Remove um item do conjunto pela posição.

Controles Splitter

Embora seja possível que você já tenha visto os controles TreeView e ListView, é menos provável que conheça o controle Splitter; no entanto, pode tê-lo usado. O controle Splitter é utilizado para redimensionar os controles-filhos de uma janela no tempo de execução. Por exemplo, no Explorer, se passarmos o mouse sobre a linha cinza que separa TreeView e ListView, poderemos

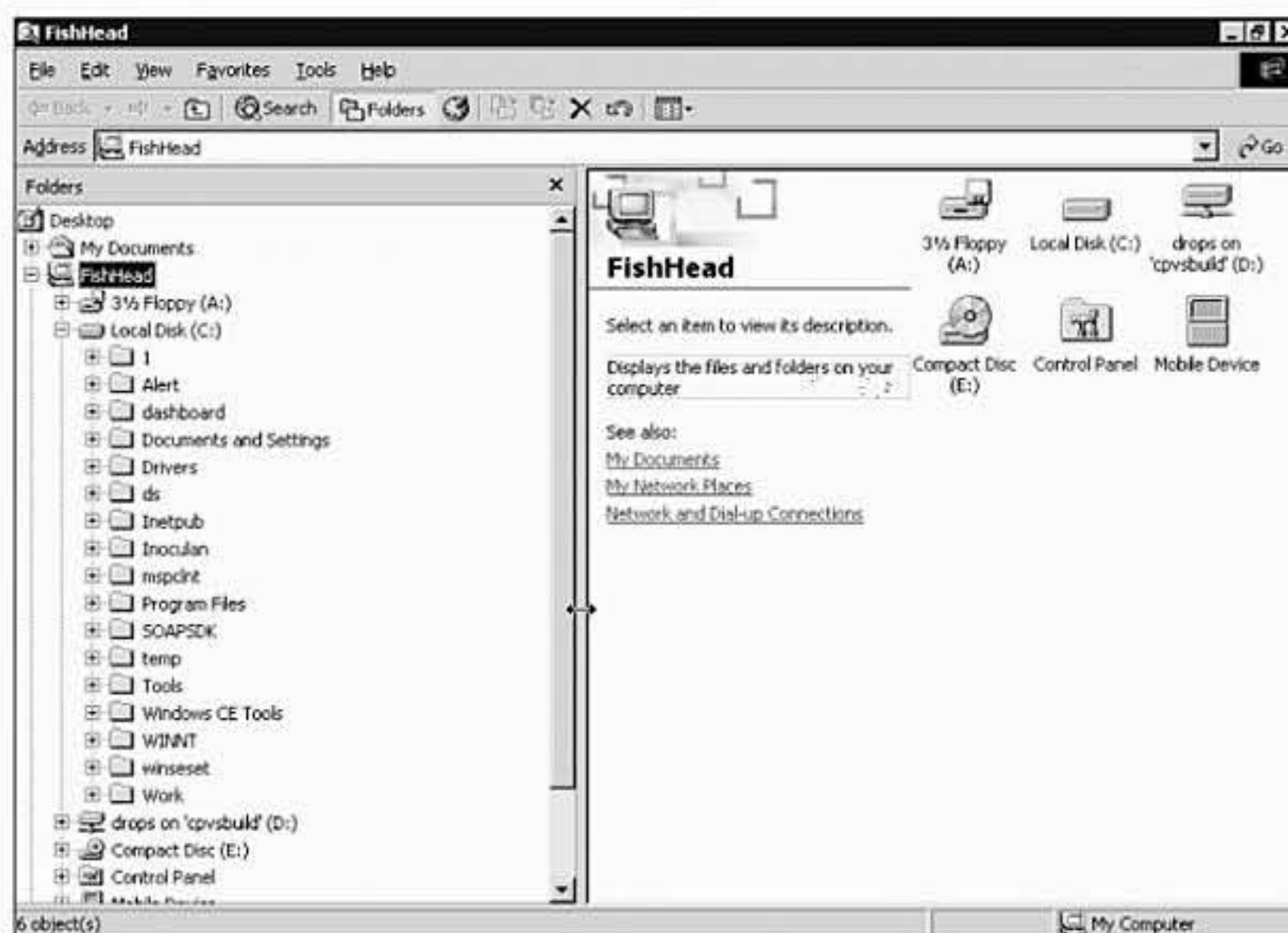
arrastar esse separador de um lado para outro, alterando o tamanho relativo dos dois controles (veja a Figura 16.15). Esse é o controle `Splitter`. Ele é um controle simples, cuja única finalidade é permitir esse tipo de redimensionamento dinâmico. É usado junto com dois controles encaixados. Um fica ancorado a um dos lados do formulário-pai, enquanto o outro tem sua propriedade `Dock` configurada com `Fill`.

É raro ter de usar qualquer das propriedades do controle `Splitter`. No entanto, há algumas propriedades que você pode querer configurar para personalizar o comportamento do controle. Elas são descritas na Tabela 16.13.

TABELA 16.13 Propriedades Importantes do Controle `Splitter`

<i>Membro</i>	<i>Descrição</i>
<code>BackColor</code>	Como nos outros controles, esta propriedade define a cor de plano de fundo do controle <code>Splitter</code> . É útil para ajudar o usuário a perceber a existência desse controle.
<code>Dock</code>	Configura a direção do controle <code>Splitter</code> . Se configurado para um lado, será usado na alteração da altura dos dois controles. Se for encaixado na parte superior ou inferior do formulário, será empregado para alterar a largura dos dois controles.
<code>minSize</code>	Define um tamanho mínimo para o controle que não for configurado como <code>Fill</code> . Será útil se você não quiser que as informações de um lado do controle <code>Splitter</code> fiquem ocultas.

FIGURA 16.15
*Usando o controle
`Splitter`.*



Para ver como esses três controles podem ser usados no desenvolvimento rápido de um aplicativo, criaremos um clone do Explorer, permitindo que o usuário navegue pelas unidades, examinando os arquivos armazenados nelas.

Crie um novo aplicativo Windows. Chamei o meu de Pioneer, mas você pode dar o nome que quiser ao seu. Altere o nome do arquivo do formulário gerado para PioneerForm, e a classe criada para frmPioneer. Configure o Startup Object do projeto como frmPioneer usando a caixa de diálogo Project Properties. Compile o aplicativo uma vez para se certificar de que não haja erros.

A interface com o usuário será simples – composta dos controles TreeView, ListView e Splitter (veja a Figura 16.16).

Redimensione o formulário para torná-lo maior que o padrão. Configure essa e outras propriedades conforme o descrito na Tabela 16.14.

FIGURA 16.16

Interface com o usuário para Pioneer.

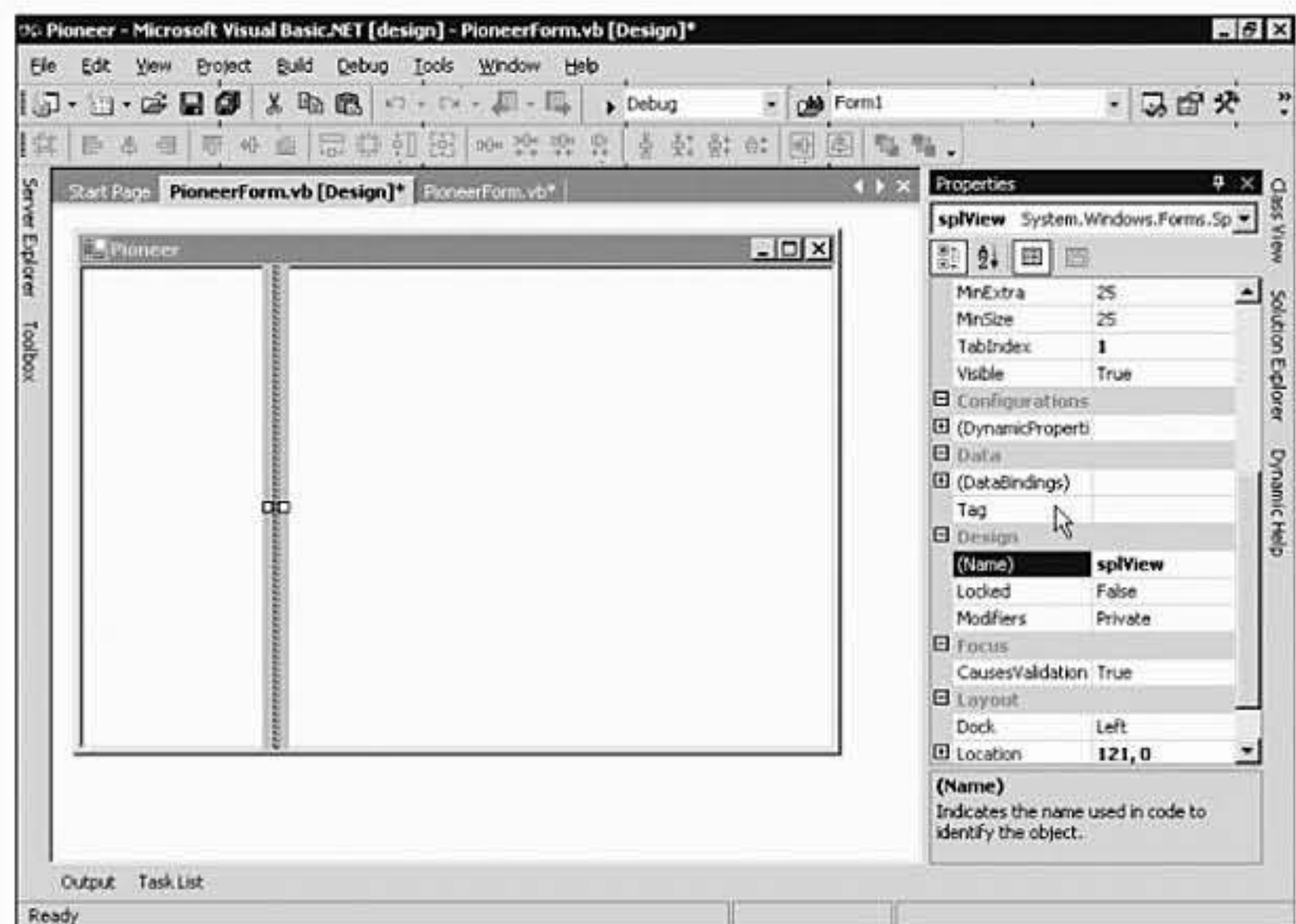


TABELA 16.14 Propriedades do Formulário Pioneer

Propriedade	Valor
Name	FrmPioneer
Text	Pioneer
Size	480, 330

Adicione um controle TreeView ao formulário. Configure as propriedades como as que vemos na Tabela 16.15.

TABELA 16.15 Propriedades do Controle TreeView

<i>Propriedade</i>	<i>Valor</i>
Name	TwvFolders
Dock	Left
Width	121

Adicione o controle `Splitter` ao formulário e altere seu nome para `splView`. Suas outras propriedades já apresentam padrões adequados. Agora adicione um controle `ListView` ao formulário. Configure as propriedades como o descrito na Tabela 16.16.

TABELA 16.16 Propriedades do Controle ListView

<i>Propriedade</i>	<i>Valor</i>
Name	LvwFiles
Dock	Fill
View	List

Quando o primeiro formulário for carregado, o controle `TreeView` deve exibir as unidades disponíveis (veja a Listagem 16.6). Como cada unidade selecionada, seus diretórios serão exibidos em `TreeView`, e os arquivos de cada diretório em `ListView`. Em princípio, no entanto, apenas `TreeView` será preenchido com as unidades disponíveis. Já que o objeto `Directory` é usado nesse código, você também deve incluir `Imports System.IO` no início dele. Isso permitirá que você empregue o objeto `Directory` sem ter de qualificá-lo totalmente (isto é, sem ter de escrever `System.IO.Directory`).

LISTAGEM 16.6 Adicionando a Lista de Unidades a TreeView

```

1 Private Sub PioneerForm_Load(ByVal sender As System.Object, _
2     ByVal e As System.EventArgs) _
3     Handles MyBase.Load
4
5     'adicione as unidades à visualização da árvore
6     Dim sDrives() As String = Directory.GetLogicalDrives()
7     Dim sDrive As String
8     Dim tvwNode As TreeNode
9
10    For Each sDrive In sDrives
11        tvwNode = tvwFolders.Nodes.Add(sDrive)
12        'adicione um nó fictício
13        tvwNode.Nodes.Add("dummy")
14    Next
15
16 End Sub

```


ANÁLISE

O método `GetLogicalDrives` da classe `Directory` é compartilhado e retorna um array contendo os nomes de todas as unidades disponíveis no computador. Aí estão incluídas unidades de disquete, de CD-ROM, de rede mapeada e outros dispositivos que se pareçam com uma unidade de disco rígido. A linha 6 preenche esse array e o bloco `For Each...Next` executa um laço nele. Adicionamos isso ao controle `TreeView` como um objeto `TreeNode` (linha 11) e inserimos na linha 13 um nó-filho com o texto `dummy`. Esse nó-filho serve a dois propósitos: marca os nós que ainda não abrimos e assegura que haja um caractere próximo a todos os nós. Se não houver nenhum nó-filho em uma árvore, o controle não adicionará um símbolo de adição e não ficará evidente a possível existência de filhos. Adicionar um nó fictício que será removido posteriormente é uma estratégia comum para fazer com que todos os nós pareçam ter filhos.

O controle `TreeView` apresenta duas funções. Quando um diretório é selecionado, seus arquivos são adicionados a `ListView`. E quando um nó é expandido em `TreeView`, os diretórios-filhos do diretório selecionado são adicionados ao controle `TreeView`. A Listagem 16.7 mostra esses dois manipuladores de eventos.

16**LISTAGEM 16.7** Manipuladores de Eventos de `TreeView`

```

17 Private Sub tvwFolders_BeforeExpand(ByVal sender As Object, _
18     ByVal e As System.Windows.Forms.TreeViewCancelEventArgs) _
19     Handles tvwFolders.BeforeExpand
20
21     'examine se já sabemos quais são os filhos
22     ' (se ainda houver um nó fictício aqui é porque não sabemos)
23     Dim oNode As TreeNode = CType(e.Node, TreeNode)
24     If oNode.Nodes(0).Text = "dummy" Then
25         'remova o nó fictício
26         oNode.Nodes(0).Remove()
27         'adicione os filhos reais
28         GetChildren(oNode)
29     End If
30 End Sub
31
32 Private Sub tvwFolders_AfterSelect(ByVal sender As Object, _
33     ByVal e As System.Windows.Forms.TreeViewEventArgs) _
34     Handles tvwFolders.AfterSelect
35
36     Dim sFiles() As String
37     Try
38         sFiles = Directory.GetFiles(tvwFolders.SelectedNode.FullPath)
39     Catch ex As Exception
40         'simplesmente ignore a exceção
41         'a causa mais provável da exceção
42         ' será 'A unidade não está pronta' se você estiver acessando um

```


LISTAGEM 16.7 Manipuladores de Eventos de TreeView (*continuação*)

```
43         ' disquete sem que ele esteja na unidade
44     End Try
45
46     If Not IsNothing(sFiles) then
47         Dim sFile As String
48         Dim oItem As ListViewItem
49
50         lvwFiles.Items.Clear()
51
52         For Each sFile In sFiles
53             oItem = lvwFiles.Items.Add(StripPath(sFile))
54         Next
55     End If
56 End Sub
57
58 Private Sub GetChildren(ByVal node As TreeNode)
59     Dim sDirs() As String = Directory.GetDirectories(node.FullPath)
60     Dim sDir As String
61     Dim oNode As TreeNode
62
63     For Each sDir In sDirs
64         oNode = node.Nodes.Add(StripPath(sDir))
65         'adicione um nó fictício como filho
66         oNode.Nodes.Add("dummy")
67     Next
68 End Sub
69
70 Private Function StripPath(ByVal path As String) As String
71     'remova o caminho principal do nome do arquivo
72     Dim iPos As Integer
73     'encontre o último caractere \
74     iPos = path.LastIndexOf("\")
75     'tudo que estiver depois dele é o nome real do arquivo
76     Return path.Substring(iPos + 1)
77 End Function
78 End Class
```

ANÁLISE

Um dos recursos do controle TreeView é que para cada operação que afete os nós da árvore há um evento Before e um After. Os eventos Before, como o manipulador de eventos BeforeExpand (linhas 17 a 30 da listagem do código), fornecem uma chance para cancelar o evento ou fazer alterações antes que seus efeitos sejam exibidos. No caso do evento BeforeExpand, você pode alterar o conteúdo dos filhos do nó antes que a lista seja expandida. Lembre-se de que anteriormente adicionamos um nó fictício para garantir que cada nó tivesse um sinal de adi-

ção próximo a ele, implicando que poderia ser expandido. O manipulador de eventos `BeforeExpand` é o melhor local para remover esse nó fictício e adicionar os reais. Depois, se não houver nós fictícios, não teremos de fazer nenhuma alteração porque `TreeView` conhecerá os nós adicionados. A rotina começa pela atribuição do nó afetado a uma variável temporária. Isso não é totalmente necessário porque o código poderia ter sido escrito com o uso de `e.Node` em vez de `oNode`. No entanto, usar uma variável temporária como essa ou o bloco `With...End With` é um bom hábito, já que é um pouco mais eficiente. A seguir, o código tenta encontrar o nó fictício (linha 24). Se ele existir, será removido, e os filhos reais serão adicionados por meio da sub-rotina `GetChildren` (descrita mais à frente). Se o nó fictício não existir, nada será feito porque já preenchemos `TreeView` com os nós-filhos apropriados.

O manipulador de eventos `twvFolders_AfterSelect` é onde `ListView` é preenchido com os arquivos do `TreeNode` selecionado. No início ele recupera os nomes de todos os arquivos usando o método compartilhado `GetFiles` da classe `Directory` (linhas 21 e 22). Esse método retorna o caminho completo para todos os arquivos em um diretório solicitado. O segundo parâmetro para o manipulador de eventos `AfterSelect`, `e`, possui duas propriedades de interesse:

- **Action** Por que o nó foi selecionado? Isto é, isso aconteceu durante uma operação de expansão, de recolhimento ou apenas quando o usuário selecionou o nó?
- **Node** O nó que foi selecionado. Essa é uma maneira mais fácil de recuperar o nó selecionado do que a alternativa `twvFolders.SelectedItem`.

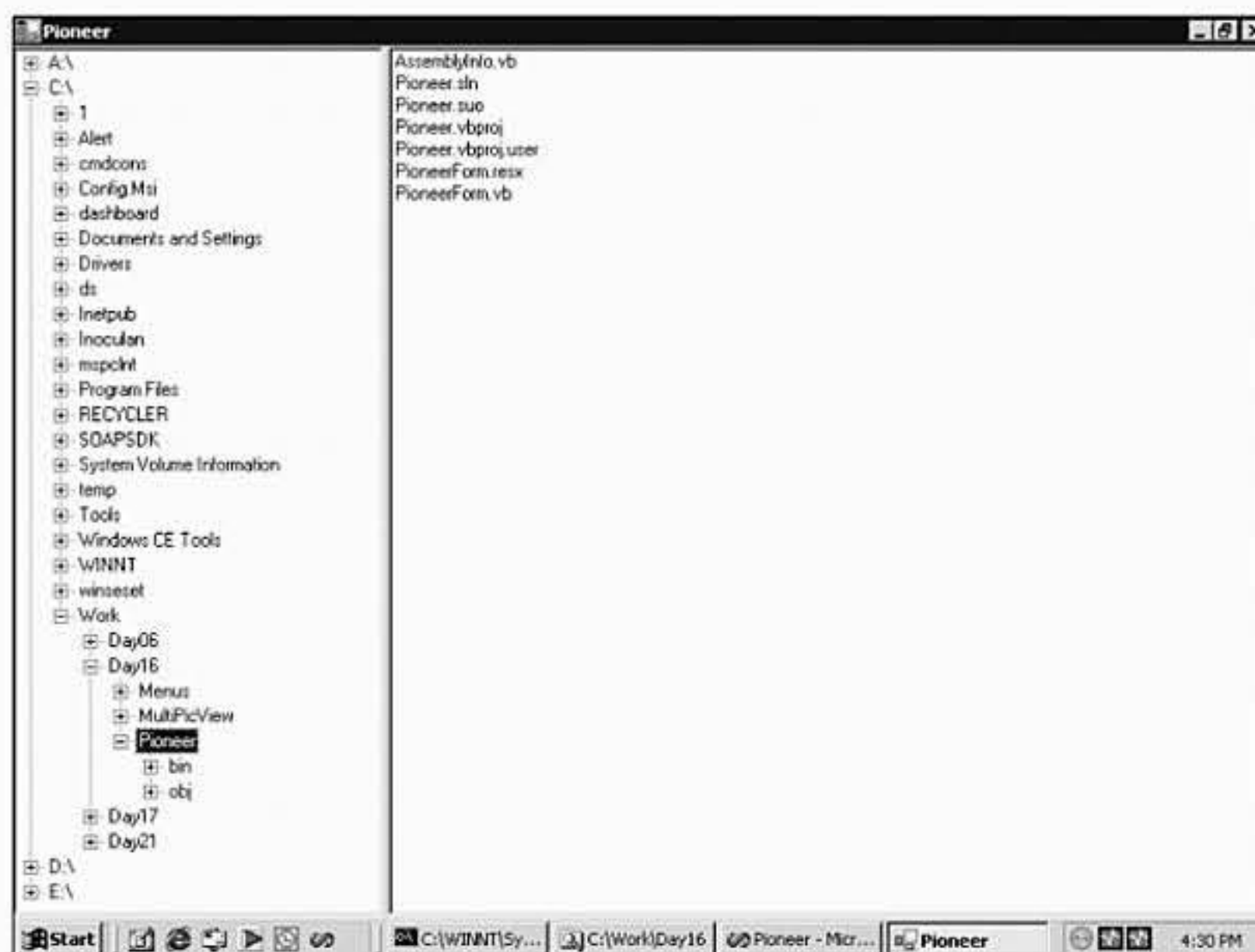
Qualquer nó de um controle `TreeView` pode recuperar seu caminho nesse local. O caminho é composto de todos os caminhos-pais, começando com o nó-raiz, separados pelo caractere `PathSeparator` do controle `TreeView`. Já que ele tem como padrão o caractere de barra invertida, essa é uma maneira fácil de recuperar um caminho semelhante ao da unidade. Por exemplo, se o nó `System32` for filho do nó `Winnt`, que é filho de `C:\`, a propriedade `FullPath` do nó `System32` seria `C:\Winnt\System32`. Esse caminho é passado para o método `GetFiles`, que é um método compartilhado da classe `Directory` e retorna um array de strings com todos os arquivos do diretório. Um laço `For Each...Next` (linhas 27 a 29) é usado para preencher `ListView` com todos os nomes de arquivo, porém, primeiro o conteúdo completo de `ListView` é apagado (linha 25). Observe que o conteúdo de cada diretório é recuperado sempre que `TreeNode` é selecionado; nenhuma tentativa de armazenar essa lista é feita.

O procedimento `GetChildren` é usado para preencher um `TreeNode` com seus nós-filhos (linhas 58 a 68). O nó que será adicionado é passado para a rotina a fim de ajudar a determinar os subdiretórios e identificar o `TreeNode` que será preenchido. Observe que o caminho existente do nó selecionado não é passado porque isso pode ser determinado por meio do método `FullPath` do nó (linha 59) como fizemos anteriormente no manipulador de eventos `AfterSelect`. O método compartilhado `Directory.GetDirectories` é usado para recuperar um array de strings que contém os subdiretórios do diretório selecionado. O código restante é semelhante ao usado quando as unidades foram adicionadas a `TreeView`. O caminho é removido do nome do diretório selecionado (linha 64), e um nó fictício é adicionado para assegurar que o nó tenha a aparência de que pode ser expandido (linha 66).

A função `StripPath` é usada por outros procedimentos para remover o caminho de um nome de arquivo, deixando apenas o nome. Essa rotina (linhas 70 a 78) usa o método `LastIndexOf` do objeto de string para encontrar a última posição de um caractere selecionado, nesse caso a barra invertida, “\”. Tudo que estiver antes dessa última barra invertida deve ser o caminho, enquanto o que houver depois dela pode ser o nome real do arquivo. Para concluir, a linha 76 extrai e retorna o texto começando pelo caractere após a última barra invertida.

Compile e execute o aplicativo. Você deve ver algo semelhante à Figura 16.17, dependendo de seu conjunto de unidades e diretórios. Você também pode navegar pelos diretórios, e os arquivos de cada um deles será exibido. Agora temos um ponto de partida para criar seu próprio explorador de arquivos. Talvez queira estender *Pioneer* para adicionar outros recursos, como a capacidade de inserir diretórios, alterar a visualização dos arquivos, adicionar figuras e assim por diante.

FIGURA 16.17
O Pioneer em ação.



Resumo

É claro que há muitos outros tópicos a aprender relacionados aos formulários Windows. Além disso, vários outros controles que estão disponíveis na caixa de ferramentas não foram descritos. Teste-os para ver como podem ajudá-lo em seus aplicativos. Se você encontrar em outro aplicativo um recurso que goste, tente encontrar o controle, ou possivelmente a configuração da propriedade, que permitirá reproduzi-lo em seus próprios aplicativos.

Você deve ter percebido que, na verdade, temos usado objetos em todos os nossos aplicativos. No entanto, você deve aprender a criar seus próprios objetos. No próximo capítulo (Dia), continuaremos nosso aprendizado sobre o .NET Framework e algumas classes importantes que serão usadas com frequência.

P&R

P Como você associaria **Ctrl+B** a um item de menu de modo que pressionando essa combinação de teclas um código fosse executado para esse item?

R Adicione uma tecla de atalho a um item de menu por meio da propriedade `ShortcutKey`. Selecione o item `CtrlB` na lista suspensa dessa propriedade.

P O que acontece se a propriedade `MdiParent` de um formulário não for configurada antes de sua exibição?

R Se você não definir `MdiParent` para um formulário com a propriedade `IsMdiContainer` configurada como `True`, o novo formulário não ficará armazenado dentro de `MdiParent` e poderá ser transferido para fora dos limites do formulário-pai.

P Qual seria o resultado da execução deste código se `tv Folders` fosse um controle `TreeView` de um formulário?

```
Dim oItem as TreeNode
oItem = tvwFolders.Nodes.Add("One")
With oItem.Nodes.Add("Two")
    .Nodes.Add("Three")
    .Nodes.Add("Four")
End With
```

R Quatro itens seriam adicionados a `TreeView`. O nó-raiz teria o texto `One` e um nó-filho, `Two`, que possuiria dois nós-filhos, `Three` e `Four`. O bloco `With...End...With` é uma alternativa a armazenar um nó em uma variável `TreeNode` quando nós-filhos forem adicionados.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Qual a melhor maneira de organizar as janelas-filhas de um aplicativo MDI para visualizar as barras de título de todos os filhos?

Exercícios

1. Encontre exemplos de aplicativos MDI e SDI que você normalmente use. Fica mais fácil ou mais difícil manter o registro de vários documentos quando se trabalha com eles?
2. Atualize o aplicativo Pioneer que você desenvolveu durante esta lição para que exiba os arquivos por meio do modo de exibição ‘Detalhe’ do controle `ListView`. Adicione colunas para apresentar os atributos `ReadOnly`, `Hidden` e `System` de cada arquivo. Além disso, mostre em outra coluna a última vez que o arquivo foi alterado.

SEMANA 3

DIA 17

Usando o .NET Framework

No Dia 8, “Introdução ao .NET Framework”, você examinou pela primeira vez o .NET Framework e algumas das classes que ele disponibiliza. Como deve ter percebido, discutimos classes naquela lição. Agora, aprenderemos alguns dos grupos mais comuns de classes que são usados regularmente. Entre eles estão:

- Os arquivos e as classes relacionadas.
- O desenho com as classes de figuras.

Fluxos e Arquivos

Muitos aplicativos do Visual Basic .NET que você escrever terão de lidar com arquivos de algum tipo. Pode ser preciso criar arquivos, ler arquivos de configuração ou encontrar arquivos nas unidades do usuário. Portanto, aprender a trabalhar com arquivos é uma habilidade importante, necessária ao desenvolvedor do Visual Basic .NET.

Quando você começar a trabalhar com as classes do .NET Framework, principalmente ao escrever aplicativos para lidar com arquivos, se deparará com frequência com itens que usam a palavra ‘Stream’, por exemplo: `FileStream`, `TextStream`, `StreamWriter` e `NetworkStream`. Como deve ter percebido, eles estão relacionados. Os projetistas do Visual Basic .NET e do .NET Framework tentaram facilitar o uso do Framework definindo tudo da maneira mais consistente possível. Uma maneira pela qual fizeram isso foi empregando os mesmos tipos de objetos em muitos locais. Um grupo desses objetos é o objeto `Stream`, seus filhos e as classes utilizadas para trabalhar com Streams. Essas outras classes incluem várias classes `Reader` e `Writer`, `Encoders` e mui-

tas outras. Enfocaremos as essenciais, mas à medida que avançarmos mais, será perceptível a familiaridade de outras classes dessa família.

O Que É um Stream?

O termo ‘streams’, como muitos usados em computação, é uma tentativa de nomear algo que lembre uma equivalência do mundo real. Um stream no mundo real é uma pequena quantidade de água, fluindo. Um fluxo de computador é semelhante – é um fluxo de informações que passam de maneira seqüencial. No entanto, a menos que você tenha um arquivo sobre os rios do mundo, tenho certeza de que provavelmente não pensará em água quando examinar os arquivos da unidade de seu disco rígido. Porém, imagine-se como parte de sua unidade de disco rígido que lê arquivos. Tudo o que ela vê é uma série de informações, fluindo seqüencialmente – um fluxo.

Depois de imaginar os arquivos como um fluxo, será mais fácil trabalhar com eles. Então, seja o arquivo binário ou algum que contenha texto, será possível lidar com ele da mesma maneira. Isto é, abra o arquivo, manipule-o e grave nele utilizando os mesmos métodos porque são todos fluxos. Além disso, essas habilidades também se aplicam a outros itens que se parecem com fluxos. Não os abordaremos neste livro, mas aí se incluem as comunicações de rede, a memória de seu computador, dados criptografados e muitos outros recursos. Aprendendo a ler e gravar arquivos simples de texto, já saberemos como trabalhar com qualquer um desses fluxos.

Arquivos e Diretórios

Antes de começarmos a criar fluxos, no entanto, passaremos algum tempo examinando como encontrar, gerar e excluir diretórios e arquivos. Isso permitirá que você localize arquivos e crie diretórios para armazenar seus aplicativos.

Como vimos no Dia 8, o .NET Framework é dividido em uma série de espaços de nomes. Cada um contém várias classes que estão relacionadas de algum modo. Lembre-se de que no Dia 8, trabalhamos com conjuntos (como `ArrayList` e `Queue`), e eles foram reunidos no espaço de nome `System.Collections`. De maneira semelhante, todas as classes de arquivos e de diretórios (e fluxos) estão agrupadas no espaço de nome `System.IO`. Esse espaço de nome não está, como se pode pensar, relacionado à lua altamente vulcânica de Júpiter, mas, em vez disso, à entrada e à saída. O espaço de nome `System.IO` é um dos mais usados no .NET Framework por essa razão. Embora um espaço de nome relacionado com a lua de Júpiter pudesse ser útil, não seria tão utilizado. Sempre que quiser abrir, ler, gravar ou apenas manipular um arquivo ou fluxo, você precisará desse espaço de nome. Felizmente, o Visual Basic .NET sempre o inclui quando cria um novo aplicativo Windows, portanto ele em geral está disponível.

Se você pesquisar o espaço de nome `System.IO` na ajuda on-line, verá duas classes que não irão surpreendê-lo – `File` e `Directory`. Essas duas classes são convertidas em arquivos e diretórios. Elas possuem propriedades e métodos que representam o que se espera de arquivos e diretórios. Por exemplo, a classe `Directory` possui métodos para recuperar subdiretórios, acessar a lista de arquivos do diretório e assim por diante. De maneira semelhante, a classe `File` tem métodos que

são usados para abrir o arquivo, copiá-lo e recuperar informações sobre ele. A Tabela 17.1 lista os métodos importantes do objeto `Directory`, enquanto a Tabela 17.2 mostra os métodos importantes da classe `File`.

TABELA 17.1 Métodos da Classe `Directory`

<i>Método</i>	<i>Descrição</i>
<code>CreateDirectory</code>	Cria um ou mais diretórios. Uma das funções mais poderosas desta classe é gerar uma árvore inteira de diretórios.
<code>Delete</code>	Remove um diretório.
<code>Exists</code>	Retorna <code>True</code> se o diretório existir.
<code>GetCurrentDirectory</code>	Retorna o caminho completo do diretório atual.
<code>GetDirectories</code>	Retorna um array contendo os diretórios-filhos do diretório desejado.
<code>GetFiles</code>	Retorna um array contendo os arquivos do diretório solicitado.

TABELA 17.2 Métodos da Classe `File`

<i>Método</i>	<i>Descrição</i>
<code>Copy</code>	Copia um arquivo.
<code>Create</code>	Cria um novo arquivo.
<code>CreateText</code>	Uma versão especial de <code>Create</code> que cria um arquivo de texto.
<code>Delete</code>	Exclui um arquivo.
<code>Exists</code>	Retorna <code>True</code> se o arquivo existir.
<code>Open</code>	Abre um arquivo para leitura, gravação ou ambos.
<code>OpenRead</code>	Versão especializada de <code>Open</code> que sempre abre o arquivo para leitura.
<code>OpenText</code>	Versão especializada de <code>Open</code> que abre arquivos de texto apenas para leitura. Esse seria um atalho prático caso você estivesse escrevendo um aplicativo que precisasse ler informações sobre configuração ou um arquivo de registros.
<code>OpenWrite</code>	Versão especializada de <code>Open</code> que sempre abre o arquivo para gravação.

Criando Arquivos

Como você pode ver nas Tabelas 17.1 e 17.2, vários métodos das classes `File` e `Directory` podem ser usados na pesquisa e criação de arquivos e diretórios. Esse é um ótimo exemplo da orientação a objetos em ação – classes que representam objetos do mundo real agindo como esse objeto. Por exemplo, um diretório deve ter conhecimento de que diretórios-filhos possui. De maneira semelhante, deve-se poder solicitar a um arquivo que abra a si mesmo.

**NOTA**

Muitos dos métodos das classes `File` e `Directory` são compartilhados (shared). Isso significa que você não precisa criar um objeto para usá-los; apenas empregue a classe para acessá-los. Por exemplo, o método `Exists` da classe `File` é compartilhado. Para saber se um arquivo existe, não é necessário declarar uma instância da classe `File`; utilize a própria classe. Portanto, em vez de escrever:

```
Dim oFile As New File()
bExists = oFile.Exists("algumarquivo.txt")
```

determine se um arquivo existe com o código a seguir:

```
bExists = File.Exists("algumarquivo.txt")
```

Lendo um Arquivo de Texto

Depois que você criar um arquivo, precisará ter condições de ler seu conteúdo. É aí que as classes `Stream` entram em cena, assim como as várias classes `Reader` e `Writer`. Quando um arquivo for aberto, uma dessas classes invariavelmente será passada, quase sempre uma classe `Stream` de um tipo ou de outro. Em seguida, será possível ler as informações do arquivo usando a classe `Stream` ou aplicando `StreamReader` a ela. Já que a classe `Stream` com a qual é mais provável que lidemos é `FileStream`, passaremos mais tempo examinando-a.

`FileStream` é uma classe `Stream` que se obtém quando um arquivo é lido. O arquivo pode ser de texto ou binário. `FileStream` possui vários métodos que permitirão que você leia o arquivo e propriedades que estarão relacionadas a ele. A Tabela 17.3 descreve os mais importantes.

TABELA 17.3 Métodos e Propriedades de `FileStream`

Nome	Descrição
<code>CanRead</code> (Propriedade)	Igual a <code>True</code> se o arquivo puder ser lido. Esta é uma propriedade adequada em testes para evitar uma exceção que poderá ocorrer se o arquivo estiver bloqueado ou for aberto apenas para gravação.
<code>CanSeek</code> (Propriedade)	Igual a <code>True</code> se você puder pesquisar (isto é, mover-se para a frente e para trás) no arquivo. É uma boa propriedade em testes para evitar que uma exceção ocorra quando um arquivo no qual não for possível mover-se para trás for lido. Isso é raro nos arquivos, mas freqüente em alguns outros tipos de fluxos.
<code>CanWrite</code> (Propriedade)	Igual a <code>True</code> se você puder gravar no arquivo. Essa é uma boa propriedade em testes para evitar uma exceção que poderá ocorrer se o arquivo estiver bloqueado ou for aberto apenas para leitura.
<code>Length</code> (Propriedade)	Quantidade de bytes do arquivo.
<code>Position</code> (Propriedade)	A posição atual do arquivo.
<code>Close</code> (Método)	Finaliza <code>FileStream</code> . Sempre encerra a classe <code>FileStream</code> (ou qualquer classe <code>Stream</code>) quando você termina de usá-la.

TABELA 17.3 Métodos e Propriedades de FileStream (*continuação*)

<i>Nome</i>	<i>Descrição</i>
Read (Método)	Lê vários bytes de FileStream. Eles são retornados para você em um array.
Seek (Método)	Provoca a movimentação para a frente e para trás em um arquivo.
Write (Método)	Grava vários bytes em FileStream.

O problema dos métodos Read da classe FileStream é que tendem a não ser muito convenientes porque todos lidam com bytes. Em vez de usá-los, você pode aplicar StreamReader à FileStream para ler as informações do arquivo de uma maneira mais natural. A Tabela 17.4 mostra os métodos importantes de StreamReader.

TABELA 17.4 Métodos Importantes de StreamReader

<i>Nome</i>	<i>Descrição</i>
Close	Finaliza StreamReader. Sempre encerra suas classes StreamReader quando você termina de usá-las.
Read	Lê o próximo caractere de Stream. Será adequado se você estiver lendo um caractere das informações de cada vez.
ReadBlock	Lê um bloco de caracteres de Stream. Pode ser uma maneira rápida de ler as informações de uma classe Stream.
ReadLine	Lê a próxima linha de Stream. É uma maneira adequada de manipular arquivos que fornecem informações organizadas por linhas.
ReadToEnd	Lê todos os caracteres de Stream de uma só vez. É o meio mais rápido de extrair todas as informações de Stream e inserir em uma variável.

A Listagem 17.1 mostra uma maneira comum de abrir um arquivo de texto e ler o conteúdo em uma variável string.

CÓDIGO**LISTAGEM 17.1** Lendo um Arquivo

```

1 Dim oFile As FileStream
2 Dim oReader As StreamReader
3 Dim sContents As String
4 oFile = New FileStream("MyFile.txt", FileMode.OpenOrCreate, FileAccess.Read)
5 oReader = New StreamReader(oFile)
6 sContents = oReader.ReadToEnd()
7 oReader.Close()
8 oReader = Nothing
9 oFile = Nothing

```


Gravando em um Arquivo de Texto

Exatamente como na leitura, você pode usar o objeto `Stream` para gravar algo nele mesmo. No entanto, é muito mais fácil empregar `StreamWriter` para fazer isso. Da mesma maneira que com `StreamReader`, aplique `StreamWriter` a um objeto `Stream` existente e utilize os métodos `Write` e `WriteLine` para adicionar as informações. A Tabela 17.5 resume os métodos significativos de `StreamWriter`.

TABELA 17.5 Métodos Importantes da Classe `StreamWriter`

Nome	Descrição
<code>Close</code>	Finaliza <code>StreamWriter</code> . Sempre encerrará alguma classe <code>StreamWriter</code> que você possa ter criado. Se isso falhar, poderá causar a perda das alterações feitas no arquivo.
<code>Write</code>	Grava em <code>Stream</code> .
<code>WriteLine</code>	Grava em <code>Stream</code> , finalizando as informações adicionadas com uma nova linha.

A Listagem 17.2 mostra um exemplo típico de uma gravação em um arquivo de texto.

CÓDIGO

LISTAGEM 17.2 Gravando em um Arquivo

```

1 Dim oFile As FileStream
2 Dim oWriter As StreamWriter
3 oFile = New FileStream("MyFile.txt", _
4     FileMode.OpenOrCreate, FileAccess.Write)
5 oWriter = New StreamWriter(oFile)
6 'Grava o inteiro 123 no arquivo
7 oWriter.Write(123)
8 'Grava a string "Customer" no arquivo
9 oWriter.Write("Customer")
10 'Grava a string "John Bull" no arquivo, as próximas gravações ficarão em uma
    linha nova
11 oWriter.WriteLine("John Bull")
12 oWriter.Close()
13 oWriter = Nothing
14 oFile = Nothing

```

Os métodos de `StreamWriter` devem ser familiares; são os mesmos que usamos na classe `Console`. Essa também é uma classe `StreamWriter` – projetada para gravar no console.

Agora reúna leitura e gravação e crie um aplicativo simples que use as três classes principais que discutimos: `FileStream`, `StreamReader` e `StreamWriter`. Como exemplo, geraremos um substitui-

to simples para o Bloco de notas que permitirá a leitura e gravação de arquivos de texto. Posteriormente ele poderá ser estendido para permitir a criação de outros tipos de informações.

Crie um novo aplicativo Windows e chame-o de Note. Afinal, é um programa seu. Como sempre, a primeira etapa após a criação do projeto é se certificar de que o formulário principal não seja chamado de Form1. Abra o formulário na tela do código, localize e substitua Form1 por frmMain. Da mesma maneira, renomeie o arquivo como frmMain.vb. Para concluir, abra a caixa de diálogo das propriedades do projeto e configure Startup Object como frmMain. Execute uma compilação para assegurar que não haja erros.

Adicione um controle TextBox ao formulário e configure suas propriedades conforme o descrito na Tabela 17.6.

TABELA 17.6 Propriedades de TextBox

<i>Propriedade</i>	<i>Valor</i>
(Name)	TxtText
Multiline	True
Text	(deixe em branco)
Scrollbars	Vertical
Dock	Fill

Para fazer com que seu bloco de notas fique igual ao do Windows, você precisará de um menu. Dê um clique duplo no item MainMenu da caixa de ferramentas para adicionar um. Como sempre, altere o nome para mnuMain. Adicione os itens mostrados na Tabela 17.7.

TABELA 17.7 O Menu Principal

<i>Item</i>	<i>Propriedade</i>	<i>Valor</i>
Menu de nível superior	Caption	&File
	Name	mnuFile
Abaixo de &File	Caption	&New
	Name	mnuFileNew
	Shortcut	CtrlN
Abaixo de &New	Caption	&Open...
	Name	mnuFileOpen
	Shortcut	CtrlO
Abaixo de &Open...	Caption	&Save
	Name	mnuFileSave
	Shortcut	CtrlS
Abaixo de &Save	Caption	Save &As...

TABELA 17.7 O Menu Principal (*continuação*)

<i>Item</i>	<i>Propriedade</i>	<i>Valor</i>
Abaixo de Save &As...	Name	mnuFileSaveAs
	Caption	-
Abaixo de -	Name	mnuFileSep
	Caption	E&xit
	Name	mnuFileExit
	Shortcut	CtrlQ
Item de nível superior	Caption	&Edit
	Name	mnuEdit
Abaixo de &Edit	Caption	Cu&t
	Name	mnuEditCut
	Shortcut	CtrlX
Abaixo de Cu&t	Caption	&Copy
	Name	mnuEditCopy
	Shortcut	CtrlC
Abaixo de &Copy	Caption	&Paste
	Name	mnuEditPaste
	Shortcut	CtrlV
Item de nível superior	Caption	&Help
	Name	mnuHelp
Abaixo de &Help	Caption	&About
	Name	mnuHelpAbout

O resultado final deve se parecer com os menus mostrados na Figura 17.1.

Para concluir o formulário, adicione um controle `FileOpenDialog` e um `FileSaveDialog` a ele e configure as propriedades como mostra a Tabela 17.8.

FIGURA 17.1

Executando o novo exemplo.

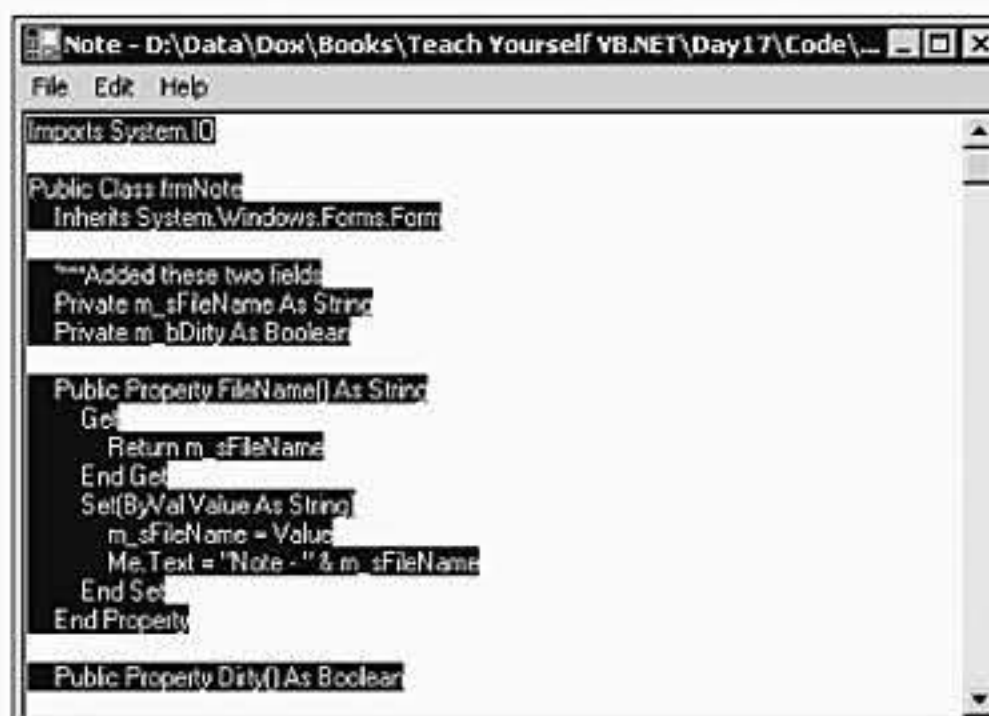


TABELA 17.8 Outros Controles do Formulário

<i>Objeto</i>	<i>Propriedade</i>	<i>Valor</i>
FileOpenDialog	Name	dlgOpen
	Filter	Text Files *.txt All Files *.*
FileSaveDialog	Name	DlgSave
	Filter	Text Files *.txt All Files *.*
	FileName	notel

Agora que a interface com o usuário está configurada, precisamos adicionar o código.

Primeiro, adicionaremos o espaço de nome System.IO ao projeto e ao formulário. Abra a pasta References no Solution Explorer. Você deve ver um espaço de nome System incluído na listagem, mas não System.IO, o que é bom. Os itens da caixa de diálogo References representam as bibliotecas (DLLs) que serão realmente usadas para localizar o código quando seu aplicativo estiver em compilação. Cada uma dessas DLLs pode conter vários espaços de nome. Um deles é System.IO. Para tornar tudo mais fácil no momento da codificação, no entanto, também podemos querer importar os espaços de nome que empregaremos. Isso permitirá a utilização de atalhos quando fizermos referência às classes. Para obter mais detalhes, veja a nota a seguir. Importe o espaço de nome System.IO para economizar na digitação. Adicione a Imports System.IO a primeira linha, antes de Public Class frmMain.



Muitas pessoas que começam a usar o Visual Basic .NET confundem-se com as referências e quando empregá-las no lugar da instrução Imports. As referências são adicionadas por meio do Solution Explorer ou do menu Project. Elas representam os outros componentes que seu aplicativo usará. Se você for empregar um código que exista em outra DLL – qualquer dos espaços de nome que vêm com o .NET Framework que não sejam automaticamente adicionados ou um código que foi escrito por outro desenvolvedor ou empresa –, será preciso adicioná-la às referências de seu projeto. A instrução Imports pode ser considerada como uma ferramenta para encurtar seu código. Utilizando essa instrução, é possível evitar a necessidade de escrever nomes longos de classes. Por exemplo, quando o controle TextBox for adicionado a um formulário, o Visual Basic .NET usará a linha de código a seguir:

```
Private WithEvents txtText As System.Windows.Forms.TextBox
```

Portanto, o nome completo da classe é System.Windows.Forms.TextBox. No entanto, se você adicionar Imports System.Windows.Forms ao início do arquivo, poderá encurtar isso para:

```
Private WithEvents txtText As TextBox
```

Quando o Visual Basic .NET ler TextBox, usará o espaço de nome adicionado com a instrução Imports para procurar a classe TextBox.

Assim, se você usar um espaço de nome, ele deverá ser incluído na seção References. Se quiser encurtar seu código, utilize a instrução Imports.

A seguir, adicione duas propriedades novas ao formulário. Adicione uma propriedade para controlar o nome do arquivo (com o caminho) e se algo foi alterado. Acrescente o código da Listagem 17.3 ao programa logo após a linha `Inherits System.Windows.Forms.Form`.

CÓDIGO**LISTAGEM 17.3** Propriedades do Aplicativo Note

```
1 Private m_sFileName As String
2 Private m_bDirty As Boolean
3 Public Property FileName() As String
4     Get
5         Return m_sFileName
6     End Get
7     Set(ByVal Value As String)
8         m_sFileName = Value
9         Me.Text = "Note - " & m_sFileName
10    End Set
11 End Property
12 Public Property Dirty() As Boolean
13     Get
14         Return m_bDirty
15     End Get
16     Set(ByVal Value As Boolean)
17         m_bDirty = Value
18         Me.Text = "Note - " & m_sFileName & "*"
19     End Set
20 End Property
21 Public Sub NewFile()
22     Me.txtText.Text = " "
23     Me.FileName = " "
24     Me.Dirty = False
25 End Sub
```

ANÁLISE

O código é um conjunto de instruções de propriedade relativamente simples. As duas primeiras linhas declaram campos privados que armazenarão os valores das propriedades. As linhas 3 a 11 representam a propriedade `FileName`. Essa propriedade salva o nome do arquivo para acesso posterior no campo `m_sFileName`. Ao ser configurada, `FileName` também é incluída no título do formulário (linha 9).

De maneira semelhante, a propriedade `Dirty` (linhas 12 a 20) armazena suas informações no campo `m_bDirty` e adiciona um asterisco ao final da barra de título.

A seguir, adicione a caixa About (Sobre). Afinal, ela é a parte mais importante de muitos aplicativos: permite que as pessoas saibam quem é o responsável por eles. Para simplificar, use MessageBox para mostrar a informação.

Dê um clique duplo no item Help, do menu About, para acessar o editor de códigos. Adicione o código mostrado na Listagem 17.4.

CÓDIGO**LISTAGEM 17.4** Item Help, do menu About

```
1 Private Sub mnuHelpAbout_Click(ByVal sender As System.Object, _  
2     ByVal e As System.EventArgs) Handles mnuHelpAbout.Click    Dim sMessage  
3     As String  
4     sMessage = "Note – um editor de texto simples" & ControlChars.CrLf & _  
5         "Exclusivo do livro Aprenda Visual Basic .NET em 21 Dias." & _  
6         ControlChars.CrLf & _  
7         "copyright 2001.Direitos totais de distribuição fornecidos pelo autor."  
8     MessageBox.Show(sMessage, _  
9         "Note", _  
10        MessageBoxButtons.OK, MessageBoxIcon.Information)  
11 End Sub
```

ANÁLISE

A caixa About do aplicativo Note é apenas uma caixa de mensagens. No código, criamos e preenchemos uma variável string (linhas 3 a 6). Observe que estamos adicionando novas linhas à mensagem inserindo `ControlChars.CrLf` no meio da string. Esse símbolo representa uma combinação de retorno de carro (Carriage-return) e alimentação de linha (line-feed) incluída na string, o que cria uma linha nova. Para concluir, exibimos MessageBox (linha 7), incluindo o botão OK e o ícone Information.

**NOTA**

Qual a intenção de se usar 'Carriage-return, line feed'? Por que não empregar apenas newline ou algo mais simples? Como sempre, as razões estão baseadas nos dias longínquos e obscuros da computação (talvez 30 anos atrás). Antigamente, quando as pessoas usavam computadores, não trabalhavam com monitores e mouses. Utilizavam máquinas de teletipo, que funcionavam de maneira muito parecida com as máquinas de escrever com as quais se assemelhavam. Como qualquer pessoa que já tenha trabalhado com uma máquina de escrever sabe, quando chegamos ao final de uma linha e ouvimos o sinal, é hora de empurrar a alavanca de retorno. A alavanca de retorno executa duas operações, nos leva de volta ao início da linha e para a linha seguinte.

Retorno de carro = mover o carro (o objeto que contém o papel e o cilindro) de volta à posição de retorno (ou início da linha). Alimentação de linha = passar para a próxima linha. Portanto, retorno de carro, alimentação de linha significa "passe para o início da próxima linha". Carriage-return, line-feed é uma grande quantidade de palavras até para os profissionais de computação de modo que costumava ser abreviada para CRLF. No Visual Basic .NET, esse recurso está disponível como `ControlChars.CrLf`.

Agora é hora de fornecer alguma funcionalidade ao programa. Comece com o menu File, como na Listagem 17.5.

CÓDIGO**LISTAGEM 17.5** Comandos do Menu File

```
1 Private Sub mnuFileNew_Click(ByVal sender As System.Object,_
2   ByVal e As System.EventArgs) Handles mnuFileNew.Click
3   If Me.Dirty = True Then
4       If MessageBox.Show(_
5           "Você fez alterações no arquivo que serão perdidas. " & _
6           "Deseja continuar?",_
7           "New File",_
8           MessageBoxButtons.YesNo, MessageBoxIcon.Question) = _
9           DialogResult().Yes Then
10      NewFile()
11  End If
12 Else
13     NewFile()
14 End If
15 End Sub

16 Private Sub mnuFileOpen_Click(ByVal sender As System.Object,_
17   ByVal e As System.EventArgs)_
18   Handles mnuFileOpen.Click
19   Dim oFile As FileStream
20   Dim oReader As StreamReader
21   If Me.dlgOpen.ShowDialog = DialogResult().OK Then
22       'Certo, podemos tentar abrir e ler o arquivo
23       Try
24           Me.FileName = Me.dlgOpen.FileName
25           oFile = File.OpenRead(Me.FileName)
26           oReader = New StreamReader(oFile)
27           Me.txtText.Text = oReader.ReadToEnd
28       Catch ex As Exception
29           'exibiremos apenas o erro por enquanto
30           MessageBox.Show(ex.Message,_
31               "Erro ao abrir o arquivo",_
32               MessageBoxButtons.OK,_
33               MessageBoxIcon.Error)
34       Finally
35           'lembre-se de sempre encerrar suas classes de leitura e arquivos
36           oReader.Close()
37           oFile.Close()
38       End Try
39   End If
40 End Sub
```


CÓDIGO

LISTAGEM 17.5 Comandos do Menu File (*continuação*)

```
41 Private Sub mnuFileSave_Click(ByVal sender As System.Object,_  
42     ByVal e As System.EventArgs)_  
43     Handles mnuFileSave.Click  
44         'só devemos tentar salvar esse arquivo se ele tiver um nome  
45         If Me.FileName <> "Untitled" Then  
46             'Certo, tentemos salvar o arquivo  
47             Dim oFile As FileStream  
48             Dim oWriter As StreamWriter  
49             Try  
50                 oFile = File.OpenWrite(Me.FileName)  
51                 'converta o conteúdo de TextBox em um array de bytes  
52                 oWriter = New StreamWriter(oFile)  
53                 'e grave no arquivo  
54                 oWriter.Write(Me.txtText.Text)  
55                 'agora não teremos mais alterações  
56                 Me.Dirty = False  
57             Catch ex As Exception  
58                 'por enquanto, só exibiremos uma mensagem de erro  
59                 MessageBox.Show(ex.Message,_  
60                     "Erro ao salvar o arquivo",_  
61                     MessageBoxButtons.OK,_  
62                     MessageBoxIcon.Error)  
63             Finally  
64                 'lembre-se de encerrar todas as classes de gravação e de fluxo  
65                 oWriter.Close()  
66                 oFile.Close()  
67             End Try  
68         Else  
69             'se ainda não houver, crie um nome  
70             mnuFileSaveAs_Click(sender,e)  
71         End If  
72 End Sub  
  
73 Private Sub mnuFileSaveAs_Click(ByVal sender As System.Object,_  
74     ByVal e As System.EventArgs)_  
75     Handles mnuFileSaveAs.Click  
76     If Me.dlgSave.ShowDialog = DialogResult().OK Then  
77         'se derem um clique em OK, definiremos o nome do arquivo e salvaremos  
78         Me.FileName = Me.dlgSave.FileName  
79         'use o código já existente no item File, Save para salvar o arquivo  
80         mnuFileSave_Click(sender,e)  
81     End If  
82 End Sub
```


CÓDIGO**LISTAGEM 17.5** Comandos do Menu File (*continuação*)

```

83 Private Sub mnuFileExit_Click(ByVal sender As System.Object, _
84     ByVal e As System.EventArgs) _
85     Handles mnuFileExit.Click
86     Me.Close()
87 End Sub

```

ANÁLISE

Esse código pode parecer muito extenso, mas analisaremos algumas etapas isoladamente para ver o que ele faz, começando com o comando New do menu File das linhas 1 a 15. Adicionamos a propriedade Dirty para que nos permita determinar se o texto foi alterado de alguma maneira. Se tiver sido, precisamos nos certificar de que o usuário não tenha acidentalmente eliminado essas alterações. Portanto, se ele solicitar um novo arquivo depois de fazer as alterações, devemos avisá-lo. Esse nada mais é do que o comportamento adequado – proteger o usuário de pressionar uma tecla por acidente. Se o arquivo estiver sujo (alterado), perguntaremos se não há problemas em perder as alterações (linhas 3 a 9) e criar um arquivo novo. Se não houver problema (linha 10 da listagem) ou se não houver alterações (linha 13 da listagem), criaremos NewFile. O procedimento NewFile será gerado posteriormente. Já que há vários locais onde podemos criar um arquivo novo, é uma boa idéia inserir esse código em um procedimento próprio.

A seguir, vemos o item Open do menu File (linhas 16 a 40). Aqui começamos a usar as classes para trabalhar com os arquivos. Empregaremos também um objeto FileStream (linha 19) e um StreamReader (linha 20). Lembre-se de que FileStream na verdade é apenas um tipo especial de objeto Stream. Depois de declarar essas duas variáveis, permitimos que o usuário selecione um arquivo a ser aberto (linha 21) utilizando o controle OpenFileDialog adicionado ao formulário. Se ele tiver selecionado um arquivo, poderemos abri-lo. Observe que estamos tentando abrir esse arquivo dentro de um bloco Try...End Try (linhas 23 a 38). Abrir arquivos pode ser propenso a exceções – o arquivo pode não existir, não estar disponível, ter sido bloqueado por outro usuário e assim por diante. Qualquer dessas ocorrências poderia levar a uma exceção. Portanto, sempre que você quiser abrir um arquivo, deve usar um bloco Try...End Try.

Faça**Não Faça**

USE um bloco Try...End Try para capturar exceções ao lidar com arquivos – quando for abrir, excluir ou salvá-los. Empregue sempre a seção Finally do bloco para fechar o arquivo e outras classes de fluxo, leitura ou gravação que tenha utilizado para acessá-lo.

Dentro do bloco `Try...End Try`, primeiro recuperamos o nome do arquivo (inclusive o caminho) para que seja aberto na caixa de diálogo (linha 24). Esse nome será exibido na barra de título da janela de `Note`. A seguir (linha 25), usamos o método compartilhado `OpenRead` para abrir o arquivo. Lembre-se de que os métodos compartilhados não precisam de uma instância do arquivo, mas dele próprio. Essa é uma área que poderia levar a uma exceção, portanto é bom que esteja no bloco `Try...End Try`. A variável `oFile` da linha 25 é do tipo `FileStream` de modo que representa uma série de bytes do arquivo (isto é, um fluxo). Para ler esse fluxo, passamos esse objeto `FileStream` para o construtor de um `StreamReader` (linha 26). Essa variável `StreamReader` (`oReader`) pode ser empregada para ler todo o conteúdo de `FileStream` (linha 27) e passá-lo para o principal `TextBox` utilizado no formulário.

Se uma exceção ocorrer na abertura ou leitura do arquivo (mais provável que seja na abertura), a capturaremos na linha 28 e apenas a exibiremos para o usuário nas linhas 30 a 33. Uma rotina de tratamento de exceções ideal provavelmente tentaria ajudar o usuário a corrigir o erro. Por exemplo, se o usuário estivesse proibido de abrir o arquivo devido a razões de segurança (uma `SecurityException` seria lançada), você poderia informá-lo de que seria preciso solicitar permissão ao proprietário do documento antes de tentar abrir o arquivo. Para concluir, nas linhas 36 a 37, encerramos os objetos `StreamReader` e `FileStream`. Feche sempre esses objetos para assegurar que não permaneçam abertos, podendo impedir outros programas de visualizar ou alterar o arquivo.

No final do item `Open` do menu `File`, já devemos ter selecionado um arquivo para editar, e o conteúdo provavelmente estará em `TextBox`. A seguir, precisamos lidar com duas maneiras relacionadas de salvar o arquivo – elas são os itens `Save` e `Save As` do menu `File`.

Dos dois itens do menu `File` relacionados a salvar o arquivo, `Save` é o mais complicado. Na verdade, como você verá, o item `Save As` não precisa de nada além dele para executar sua ação. O código do item `Save` do menu `File` é mostrado nas linhas 41 a 72. Primeiro tente saber se o usuário já nomeou o arquivo (linha 45). Se ele não tiver feito isso, chamaremos a rotina `mnuFileSaveAs_Click` (isto é, o item `Save As` do menu `File`) para obter um nome (linha 70). Supondo que o arquivo tenha um nome, ele será aberto para edição na linha 50. Depois de aberto, criaremos um objeto `StreamWriter` para tornar possível a edição do arquivo (linha 52) e a gravação do conteúdo de `TextBox` no arquivo recém-criado (linha 54). Já que o arquivo não possui mais nenhuma alteração, configuramos o flag `Dirty` como `False`.

Se uma exceção ocorrer quando da gravação no arquivo, a capturaremos na linha 57. Exatamente como fizemos com qualquer exceção que pudesse ocorrer na leitura do arquivo, apenas exibiremos o erro (linha 58 a 62). Mais uma vez, como o aplicado à leitura, é essencial fechar os objetos `StreamWriter` e `FileStream` (linhas 65 e 66). Isso é ainda mais importante na gravação do que na leitura porque as alterações feitas podem não ser gravadas até que os objetos de fluxo e o arquivo estejam fechados.

No final do item `Save` do menu `File`, o conteúdo de `TextBox` deve estar gravado no arquivo solicitado. Você pode confirmar isso abrindo o arquivo no Bloco de notas ou em algum outro editor.

O código do item Save As do menu File é muito mais simples do que o usado para salvar o arquivo porque ele utiliza o trabalho executado em outra rotina. Esse código é empregado apenas para gerar um novo nome para o arquivo. Usamos `FileSaveDialog`, que adicionamos ao formulário na linha 76, para permitir que o usuário selecione um local e um nome para o arquivo. Se ele der um clique em OK para fechar a caixa de diálogo, daremos prosseguimento configurando a propriedade `FileName` com o novo nome (linha 78) e, em seguida, chamando a rotina `mnuFileSave_Click` para realmente salvar o arquivo (linha 80). Como alternativa, poderíamos ter colocado o conteúdo completo do código salvo aqui, mas isso teria gerado um nível alto de duplicação.

Para concluir (o menu File), ocorre o código do comando Exit do menu File. Nesse momento, apenas fechamos o formulário, encerrando o aplicativo (linha 86).

Grande parte do código relacionado a Stream está no menu File (onde se imaginava), permitindo que você abra e salve arquivos. Na Listagem 17.6, examinaremos o código do menu Edit que permite recortar, copiar e colar da mesma maneira que em um aplicativo como o Bloco de notas ou o Word.

CÓDIGO**LISTAGEM 17.6** Comandos do Menu Edit

```

1 Private Sub mnuEditCut_Click(ByVal sender As System.Object, _
2   ByVal e As System.EventArgs) Handles mnuEditCut.Click
3     Clipboard.SetDataObject(Me.txtText.SelectedText)
4     Me.txtText.SelectedText = " "
5 End Sub
6
7 Private Sub mnuEditCopy_Click(ByVal sender As System.Object, _
8   ByVal e As System.EventArgs) Handles mnuEditCopy.Click
9     Clipboard.SetDataObject(Me.txtText.SelectedText)
10 End Sub
11
12 Private Sub mnuEditPaste_Click(ByVal sender As System.Object, _
13   ByVal e As System.EventArgs) Handles mnuEditPaste.Click
14     Me.txtText.SelectedText = _
15       CType(Clipboard.GetDataObject.GetData(DataFormats.Text), String)
16 End Sub

```

ANÁLISE

Felizmente, o código dos comandos Edit é bem menor do que o de File. Além disso, podemos ver outro objeto poderoso do .NET Framework – o objeto `Clipboard`.

Começando com o código Cut do menu Edit, pegamos o texto selecionado em `TextBox` e o copiamos para a área de transferência (linha 3). A classe `Clipboard` é uma abstração da área de transferência do Windows compartilhada por todos os aplicativos. Ela possui métodos que permitirão a você atribuir programaticamente texto, figuras ou outras informações à área de transferência e recuperar os dados desse local. A Tabela 17.9 descreve esses dois métodos. Depois de copiar as

informações para a área de transferência, excluimos o texto selecionado em TextBox (linha 4). Isso cria o comportamento esperado de recorte do texto – adicioná-lo à área de transferência e removê-lo do formulário.

TABELA 17.9 Métodos da Classe Clipboard

<i>Método</i>	<i>Descrição</i>
SetDataObject	Atribui informações à área de transferência. Esse método usa um objeto e, portanto, você pode gravar o que quiser na área de transferência. É bom ressaltar que na verdade há duas versões desse método. Ou seja, ele é um método sobreposto. A segunda versão inclui um valor booleano que determina se o conteúdo deve permanecer depois que o aplicativo for encerrado. O comportamento normal é manter as informações na área de transferência depois que o programa for interrompido.
GetDataObject	Usado para recuperar as informações da área de transferência ou sobre o seu conteúdo. Na verdade, esse método retorna uma interface – IDataObject, que é empregada na execução efetiva da recuperação. A Tabela 17.10 descreve os métodos importantes da interface IDataObject.

TABELA 17.10 Métodos de IDataObject

<i>Método</i>	<i>Descrição</i>
GetData	Recupera as informações armazenadas na área de transferência. Formulários intermediários permitem que você as converta em um formato específico.
GetFormats	Usado para determinar que formatos estão armazenados na área de transferência. Por exemplo, se uma figura tiver sido copiada para a área de transferência, este método retornará todos os formatos com os quais ela pode ser colada.
GetDataPresent	Determina se algum dado de um formato específico está armazenado na área de transferência. Este método poderia ser usado para você definir se permitirá que o usuário cole informações em seu aplicativo. Por exemplo, se houver informações gráficas armazenadas na área de transferência, não deixaremos que o usuário as cole em nosso aplicativo Note.

O código Copy do menu Edit é semelhante ao do comando Cut do menu Edit (linha 9), exceto por não excluirmos o texto selecionado. Os dois comandos, no entanto, resultam na cópia das informações para a área de transferência.

O código do comando Paste do menu Edit parece complexo, mas se torna simples quando dividimos as linhas 14 e 15 em uma série de etapas. A primeira etapa para a recuperação de informações da área de transferência é obter, por meio de GetDataObject, a interface IDataObject armazenada nesse local. A seguir, usamos o método GetData de IDataObject para recuperar as informações, solicitando-as no formato de texto simples. Embora tenhamos definido esse formato, se trata de um objeto. Portanto, para atribuí-lo a TextBox, devemos convertê-lo em uma string. Poderíamos ter usado a função CStr ou CType, mas optamos pela última. Para concluir-

mos, utilizamos o texto resultante da substituição de `SelectedText` em `TextBox`. Empregamos `SelectedText`, e não `Text` porque, se substituirmos `Text`, então, o conteúdo de `TextBox` será composto apenas das informações da área de transferência. Esse não é o comportamento esperado quando se usa a área de transferência. Por meio de `SelectedText`, só substituímos o texto realçado. Se nada estiver realçado, o texto da área de transferência será adicionado à posição do cursor.

O último dos menus que usaremos é `Help`. A Listagem 17.7 mostra o código do procedimento `About` do menu `Help`.

ENTRADA**LISTAGEM 17.7** Comando `About` do Menu `Help`

```
1 Private Sub mnuHelpAbout_Click(ByVal sender As System.Object, _  
2     ByVal e As System.EventArgs) Handles mnuHelpAbout.Click  
3     Dim sMessage As String  
4     sMessage = "Note - - um editor de texto simples" &  
5         ControlChars.CrLf &  
6         "Exclusivo do livro Aprenda Visual Basic .NET em 21 Dias." &  
7         ControlChars.CrLf &  
8         "copyright 2001. Direitos totais de distribuição fornecidos pelo  
        autor."  
9     MessageBox.Show(sMessage, _  
10        "Note", _  
11        MessageBoxButtons.OK, _  
12        MessageBoxIcon.Information)  
13 End Sub
```

ANÁLISE

Embora não fosse necessário nesse exemplo, adicionamos o comando `About` do menu `Help` para que ele ficasse completo. É uma rotina simples que cria uma string (linhas 4 a 8) e a exibe para o usuário (linhas 9 a 12). Em um aplicativo mais avançado, você teria um formulário só para executar a exibição para o usuário.

Para concluir, a Listagem 17.8 mostra três rotinas que você pode adicionar ao aplicativo para complementar a funcionalidade.

LISTAGEM 17.8 Outro Código para o Aplicativo `Note`

```
1 Public Sub NewFile()  
2     Me.txtText.Text = " "  
3     Me.FileName = "Untitled"  
4     Me.Dirty = False  
5 End Sub  
6  
7 Private Sub txtText_TextChanged(ByVal sender As System.Object, _  
8     ByVal e As System.EventArgs) Handles txtText.TextChanged  
9     Me.Dirty = True
```


LISTAGEM 17.8 Outro Código para o Aplicativo Note (*continuação*)

```

10 End Sub
11
12 Private Sub frmNote_Closing(ByVal sender As System.Object, _
13     ByVal e As System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
14     If Me.Dirty = True Then
15         If MessageBox.Show(_
16             "Você fez alterações no arquivo que serão perdidas. Deseja continuar?", _
17             "New File", _
18             MessageBoxButtons.YesNo, _
19             MessageBoxIcon.Question, _
20             MessageBoxDefaultButton.Button2) = DialogResult.No Then
21             e.Cancel = True
22         End If
23     End If
24 End Sub

```

ANÁLISE

Como você deve se lembrar, o procedimento `NewFile` é chamado por `mnuFileNew_Click` quando um novo arquivo é criado. Esse procedimento atribui os padrões ao arquivo novo: apaga as informações de `TextBox` (linha 2), configura a propriedade `FileName` como 'Untitled' (linha 3) e a propriedade `Dirty` como `False` (linha 4).

Queremos que o aplicativo e o usuário tenham conhecimento quando o documento for alterado. Essa é a finalidade da propriedade `Dirty`. Podemos nos beneficiar do evento `TextChanged` para alterar o valor dessa propriedade. Se alguma alteração ocorrer, por causa de uma edição, cópia de informações da área de transferência ou digitação do usuário, esse evento será acionado, configurando o flag `Dirty` como `True`.

Para concluirmos, mais uma vez queremos nos certificar de proteger o usuário de pressionar teclas aleatoriamente. O evento `Closing` ocorre quando o formulário é fechado. É um ótimo momento para determinar se o usuário quer salvar as informações. Só devemos solicitar a ele que salve as alterações se houver alguma, portanto, nossa primeira etapa (linha 14) é descobrir se foram feitas alterações desde a última vez que as informações foram salvas. Se não houver, não precisamos fazer nada, e o formulário poderá ser fechado. No entanto, se houver alterações, teremos de perguntar ao usuário se deseja salvá-las antes de sair. As linhas 15 a 20 criam e exibem uma caixa de mensagens com os botões `Yes` e `No`. Se o usuário selecionar o botão `No`, significa que não deseja fechar o arquivo. Portanto, podemos cancelar esse evento (e o conseqüente fechamento do formulário) configurando a propriedade `Cancel` do objeto `EventArgs` (passada para todos os eventos) como `True`, como na linha 21.

Aí está – cerca de 100 linhas de código, e você tem um editor de texto funcionando que pode ser usado como base para uma implementação mais completa. Essa versão permite abrir e salvar arquivos, e usar uma área de transferência. Utilizamos os objetos `FileStream`, `StreamReader` e `StreamWriter`

para manipular os arquivos de texto com os quais ela trabalha. Criar um aplicativo semelhante para abrir e salvar arquivos binários, como os gerados pelas classes de figuras, é quase idêntico.

Desenhando com as Classes de Figuras

O Visual Basic .NET (na verdade, o .NET Framework) possui um conjunto sofisticado de ferramentas de desenho que o ajudarão a criar linhas, caixas, círculos e outras formas em virtualmente qualquer superfície de seus formulários. Acesse essas ferramentas por meio das classes de `System.Drawing` e de outros espaços de nome relacionados. As ferramentas fornecem uma ampla variedade de recursos para desenhar elementos gráficos e trabalhar com diversos arquivos de figuras. O espaço de nome `System.Drawing` contém muitas das classes essenciais que serão necessárias para qualquer tipo de desenho, incluindo: `Color`, `Brush`, `Image` e outras. Examinaremos essas classes importantes em breve.

O próximo espaço de nome em importância relacionado às figuras é `System.Drawing.Drawing2D`. Ele contém as classes referentes ao desenho efetivo das linhas, formas e assim por diante. Você ainda usará muitas dessas classes nesta lição e sempre que quiser adicionar formas simples a um formulário ou figura.

Examinando as Classes de Figuras

Com um aplicativo Windows aberto, inicie o Pesquisador de Objetos (selecione `View`, `Other Windows`, `Object Browser`). Encontre o espaço de nome `System.Drawing` e abra-o, como vemos na Figura 17.2. A quantidade total de classes, estruturas e enumerações pode parecer enorme à primeira vista. No entanto, muitas dessas classes raramente são usadas. Na verdade, só alguns itens desse espaço de nome devem ser considerados como de conhecimento necessário. Os mais importantes entre eles estão descritos na Tabela 17.11.

FIGURA 17.2
O espaço de nome
`System.Drawing`.

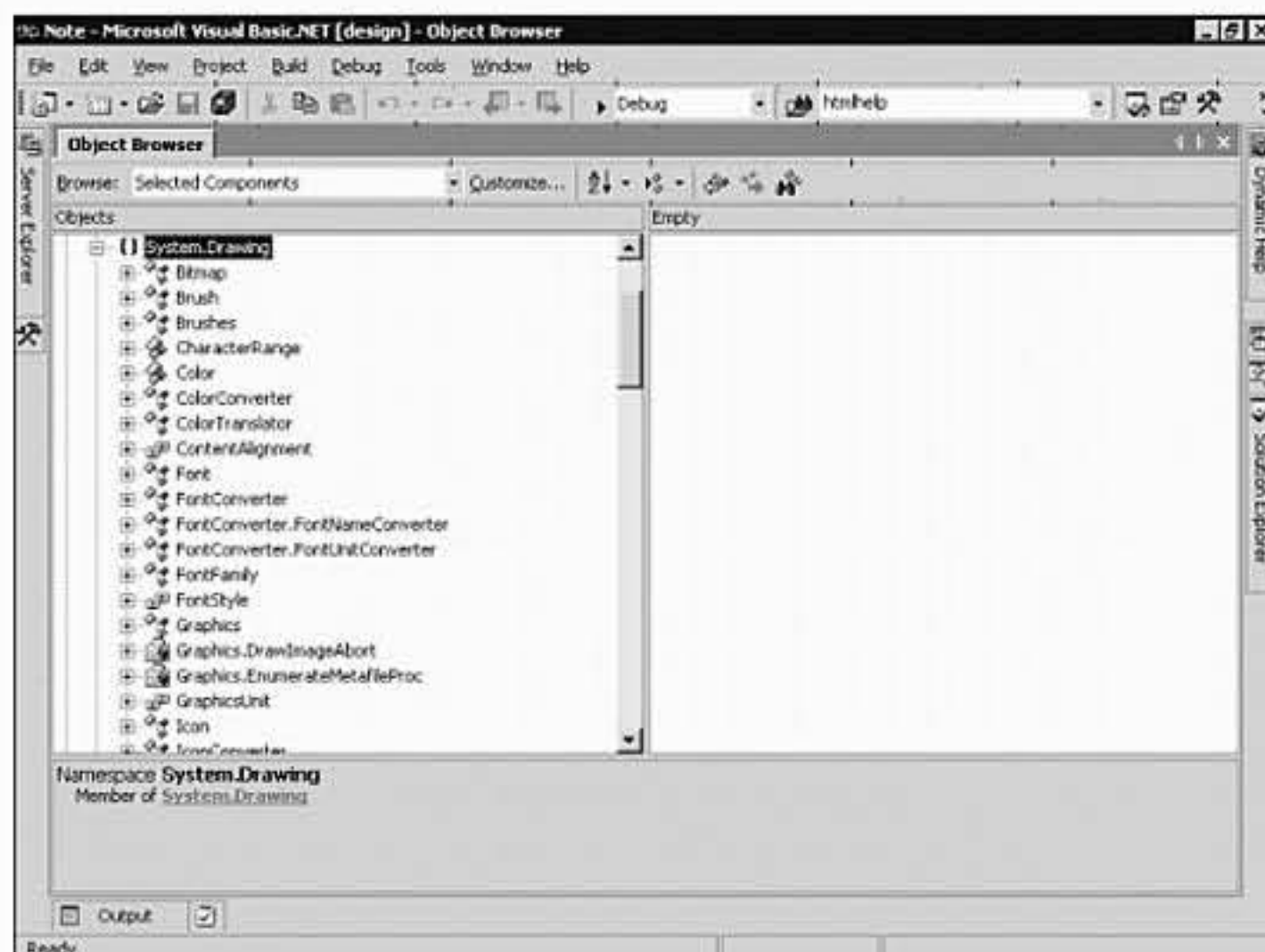


TABELA 17.11 Classes e Estruturas Importantes do Espaço de Nome System.Drawing

<i>Item</i>	<i>Tipo</i>	<i>Descrição</i>
Bitmap	Classe	O bitmap que representa uma figura, como os formatos GIF, BMP ou JPG. Esta classe é usada em geral quando se carrega e trabalha com figuras.
Brush	Classe	Usada quando áreas são preenchidas com cores. Há tipos diferentes de classes Brush, incluindo as que empregam cores sólidas e outras que aplicam texturas. Essas texturas em geral se baseiam em um bitmap.
Color	Estrutura	A estrutura Color contém informações sobre todas as cores comuns que já possuem uma denominação definida, assim como pode criar outras novas.
Font	Classe	A classe Font será usada sempre que você quiser inserir texto em uma figura.
Graphics	Classe	A mais importante das classes de System.Drawing. Esta é a classe que contém todos os métodos usados para desenhar e constitui a superfície sobre a qual serão feitos os desenhos.
Pen	Classe	Usada no desenho de linhas coloridas.
Point	Estrutura	Usada para representar um local em uma superfície de desenho. Define coordenadas X e Y para o local.
Rectangle	Estrutura	Representa uma área retangular em um local preestabelecido. É definida como uma região começando em um ponto, com largura e altura conhecidas.
Size	Estrutura	Representa uma área retangular. Essa área possui altura e largura.

Você usará muito as classes da Tabela 17.11. Por exemplo, se desenhar um quadrado na tela, ele será criado em um objeto Graphics, com uma classe Pen específica, com uma estrutura Color atribuída, dentro de uma estrutura Rectangle. Essa última, por sua vez, seria composta de uma estrutura Point e uma Size. Para concluir, seria possível preencher o quadrado com a classe Brush que fosse selecionada.

Para demonstrarmos o uso das classes de figuras, criaremos um programa simples de desenho. A Figura 17.3 mostra esse aplicativo em execução. Você pode estendê-lo posteriormente para criar desenhos mais complexos.

FIGURA 17.3
O programa Scribble
em ação.



Comece criando um novo projeto de aplicativo Windows. Dê o nome de Scribble a ele. Como você pode ver na Figura 17.3, o usuário tem três ferramentas de desenho disponíveis: uma caneta, formas e texto. Todas elas possuem seu próprio conjunto de opções que afetarão o modo de desenhar. Além disso, cada uma pode ser usada com uma cor selecionada. A ampla área branca de desenho é um controle PictureBox.

Altere sempre o nome do formulário. Feche o gerador de formulários. Altere o nome do formulário no Solution Explorer para frmScribble. Passe para o modo de visualização do código e altere todas as referências a Form1 para frmScribble. Dê um clique com o botão direito do mouse sobre o projeto no Solution Explorer e selecione Properties. Na guia General Properties, altere o Startup Object para frmScribble. Compile e execute o aplicativo para assegurar que todas as alterações tenham sido processadas.

Agora podemos começar a adicionar controles ao formulário. Abra-o no modo estrutura e insira os controles como descrito na Tabela 17.12. Use a Figura 17.3 para ajudá-lo, mas não se preocupe se eles não ficarem no lugar exato.

TABELA 17.12 Controles do Aplicativo Scribble

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Form	Height	332
	Width	500
	Text	Scribble
	BorderStyle	FixedSingle
Menu	Name	mnuMain
Menu File	Name	mnuFile
File, New	Name	mnuFileNew
	Shortcut	CtrlN
	Text	&New
File, Open	Name	mnuFileOpen
	Shortcut	CtrlO
	Text	&Open...
File, Save	Name	mnuFileSave
	Shortcut	CtrlS
	Text	&Save
File, Exit	Name	mnuFileExit
	Shortcut	CtrlQ
	Text	E&xit
OpenFileDialog	Name	dlgOpen
	DefaultExt	Bmp
	Filter	Bitmap files *.bmp PNG files *.png
SaveFileDialog	Name	dlgSave

TABELA 17.12 Controles do Aplicativo Scribble (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
RadioButton	DefaultExt	Bmp
	Filename	Scribble1
	Filter	Bitmap files *.bmp PNG files *.png
	Dock	DockStyle.Left
	Width	64
	Name	optPen
	Appearance	Button
	Checked	True
	Location	8,8
	Size	48,36
RadioButton	Text	Pen
	TextAlign	MiddleCenter
	Name	optShape
	Appearance	Button
RadioButton	Location	8,48
	Size	48,36
	Text	Shape
	TextAlign	MiddleCenter
RadioButton	Name	optText
	Appearance	Button
	Location	8,88
	Size	48,36
	Text	Text
	TextAlign	MiddleCenter
PictureBox	Name	picDraw
	BackColor	White
	BorderStyle	Fixed3D
	Dock	Fill
	Name	pnlOptions
Panel	Dock	Bottom
	Height	72
	Name	lblColor
Label	Location	8, 8
	Size	48, 16
	Text	Color:
	Name	cboColors
ComboBox	DropDownStyle	DropDownList

TABELA 17.12 Controles do Aplicativo Scribble (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Panel	Location	8, 24
	Width	136
	Name	pnlPenOptions
	Location	160, 8
Label	Size	274, 56
	Name	lblPenWidth
	Location	8, 8
	Size	80, 16
NumericUpDown	Text	Pen Width:
	Name	updpnWidth
	Location	96, 8
	Maximum	10
	Minimum	1
	Size	48, 20
Panel	Value	1
	Name	pnlShapeOptions
	Location	160, 8
	Size	274, 56
Label	Name	lblShapeType
	Location	8, 8
	Size	48, 16
	Text	Type:
DropDown	Name	cboShapeType
	DropDownStyle	DropDownList
	Location	64, 8
	Width	121
Label	Name	lblShapeHeight
	Location	8, 32
	Size	48, 16
	Text	Height:
NumericUpDown	Name	updShapeHeight
	Location	64, 32
	Maximum	1000
	Minimum	1
	Size	48, 20
	Value	20
Label	Name	lblShapeWidth

TABELA 17.12 Controles do Aplicativo Scribble (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
NumericUpDown	Location	128, 32
	Size	48, 16
	Text	Width:
	Name	updShapeWidth
	Location	184, 32
	Maximum	1000
	Minimum	1
	Size	48, 20
Panel	Value	20
	Name	pnlTextOptions
	Location	160, 8
Label	Size	274, 56
	Name	lblText
	Location	8, 8
	Size	40, 16
TextBox	Text	Text:
	Name	txtText
	Location	48, 8
	Size	208, 20
	Text	Scribble!
Label	Name	lblTextFont
	Location	8, 32
	Size	40, 16
	Text	Font:
DropDown	Name	cboTextFont
	DropDownStyle	DropDownList
	Location	48, 32
	Width	104
Label	Name	lblFontSize
	Location	160, 32
	Size	40, 16
	Text	Size:
NumericUpDown	Name	updFontSize
	Location	208, 32
	Maximum	72
	Minimum	6
	Size	48, 20
	Value	12



Quando adicionar os controles `RadioButton` ao painel de ferramentas, lembre-se de dar um clique sobre o controle na caixa de ferramentas e, em seguida, arrastá-lo até o painel. Isso assegurará que ele fique situado nesse local. Você também deve fazer isso com os controles do painel de opções.

Outro ponto a ressaltar é que os três painéis de opções (`pnlPenOptions`, `pnlShapeOptions` e `pnlTextOptions`) são todos do mesmo tamanho e ficam no mesmo local. No tempo de execução, só um ficará visível a cada vez. No entanto, isso pode ser difícil de conseguir na hora da criação. A maneira mais fácil de resolver esse problema é trabalhar com eles individualmente. Quando um dos painéis estiver concluído, configure a propriedade `Location` de modo que o painel fique fora da tela. Configurar a propriedade `Left` com `-10000` é um modo simples de fazer isso. Quando todos os três painéis estiverem concluídos, configure `Location` novamente com o valor original deles.

Onde Posso Desenhar?

Você pode desenhar em quase todos os controles dos formulários Windows, assim como em muitos dos controles de formulários da Web. Isso significa que é possível alterar facilmente um dos controles existentes ou apenas adicionar uma figura ou símbolo a quase tudo que fizer parte de seus programas. Esse recurso é ao mesmo tempo útil e flexível, permitindo que seus formulários e controles sejam personalizados de maneira simples adicionando-se uma figura e um toque exclusivos a eles.

Sua primeira etapa ao desenhar sempre será obter um objeto `Graphics`. Esse objeto pode ser obtido como parte de alguns eventos, como o evento `Paint`, ou por meio do método `CreateGraphics`. Esse método é compartilhado por todos os controles, inclusive por formulários.

Sempre que o Windows determinar que precisa redesenhar todos os seus formulários ou controles, ou apenas parte deles, você acionará o evento `Paint` para o item que será alvo da operação. Isso acontece quando uma nova janela é criada e em qualquer momento em que um formulário adicional for aberto em sua janela. Por exemplo, se você exibir uma caixa de mensagem, quando ela for fechada, a parte de seu formulário que ficou embaixo dela terá de ser redesenhada. Como consequência, o evento `Paint` de todos os controles que ficaram cobertos e do formulário será chamado. O evento `Paint` de um formulário possui a estrutura a seguir:

```
Private Sub frmScribble_Paint(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.PaintEventArgs) _  
    Handles MyBase.Paint  
End Sub
```

Nesse exemplo, o formulário tem o nome `frmScribble`. Como em todos os eventos, são passados dois objetos – `sender` e `EventArgs`. No caso do evento `Paint`, não recebemos o objeto básico `EventArgs`, mas um dos filhos, `PaintEventArgs`. Ele possui duas propriedades apenas de leitura:

- **ClipRectangle** O retângulo que deve ser redesenhado. Isto permite que você otimize o seu código de desenho para só atualizar a área alterada, em vez de redesenhar sempre a figura completa. Não usaremos este parâmetro em nossos aplicativos.
- **Graphics** A superfície em que a figura será desenhada. É usada para o desenho das formas que serão necessárias.

Como alternativa, você pode usar o método `CreateGraphics` de um controle para criar seus próprios objetos `Graphic` de desenho.

```
Dim oGraphics As System.Drawing.Graphics  
oGraphics = picDraw.CreateGraphics()
```

Qualquer método que você usar para criar um objeto `Graphic` poderá ser empregado para desenhar várias formas.

Armazenaremos o objeto `Graphics` no aplicativo `Scribble` para que seja usado por várias ferramentas de desenho. Além disso, precisamos registrar as ferramentas ativas e disponíveis. Adicione o código da Listagem 17.9 ao projeto `Scribble`.

ENTRADA**LISTAGEM 17.9** Propriedades e Código Geral do Aplicativo `Scribble`

```
1  Public Enum DrawingTools  
2      Pen  
3      Shape  
4      Text  
5  End Enum  
6  
7  Private m_sFileName As String  
8  Private m_bDrawing As Boolean  
9  Private m_eCurrentTool As DrawingTools  
10  
11 Private oGraphics As System.Drawing.Graphics  
12 Private oTool As Object  
13 Private sngX As Single  
14 Private sngY As Single  
15  
16 Public Property FileName() As String  
17     Get  
18         Return m_sFileName  
19     End Get  
20     Set(ByVal Value As String)  
21         m_sFileName = Value  
22     End Set  
23 End Property  
24  
25 Public Property Drawing() As Boolean  
26     Get
```


ENTRADA

LISTAGEM 17.9 Propriedades e Código Geral do Aplicativo Scribble
(continuação)

```
27         Return m_bDrawing
28     End Get
29     Set(ByVal Value As Boolean)
30         m_bDrawing = Value
31     End Set
32 End Property
33
34 Public Property CurrentTool() As DrawingTools
35     Get
36         Return m_eCurrentTool
37     End Get
38     Set(ByVal Value As DrawingTools)
39         m_eCurrentTool = Value
40         'elimine a ferramenta existente
41         If Not oTool Is Nothing Then
42             CType(oTool, IDisposable).Dispose()
43         End If
44     End Set
45 End Property
46
47 Private Sub frmScribble_Load(ByVal sender As Object, _
48     ByVal e As System.EventArgs) _
49     Handles MyBase.Load
50     'configure a caixa de combinação de cores
51     FillLists()
52     'crie a figura sobre a qual desenharemos
53     Me.picDraw.Image = New Bitmap(picDraw.Width, _
54         picDraw.Height, _
55         System.Drawing.Imaging.PixelFormat.Format24bppRgb)
56     Me.oGraphics = Graphics.FromImage(Me.picDraw.Image)
57     'configure o plano de fundo com a cor branca
58     Me.oGraphics.Clear(Color.White)
59     'configure a ferramenta inicial como Pen
60     optPen_Click(Nothing, Nothing)
61 End Sub
62
63 Private Sub optPen_Click(ByVal sender As Object, _
64     ByVal e As System.EventArgs) _
65     Handles optPen.Click
66     'configure a ferramenta como Pen
67     Me.CurrentTool = DrawingTools.Pen
68     'oculte todos os outros painéis de ferramentas
69     pnlPenOptions.Visible = True
```


ENTRADA

LISTAGEM 17.9 Propriedades e Código Geral do Aplicativo Scribble
(continuação)

```
70     pnlShapeOptions.Visible = False
71     pnlTextOptions.Visible = False
72 End Sub
73
74 Private Sub optShape_Click(ByVal sender As Object, _
75     ByVal e As System.EventArgs) _
76     Handles optShape.Click
77     'configure a ferramenta como Shape
78     Me.CurrentTool = DrawingTools.Shape
79     'oculte todos os outros painéis de ferramentas
80     pnlPenOptions.Visible = False
81     pnlShapeOptions.Visible = True
82     pnlTextOptions.Visible = False
83 End Sub
84
85 Private Sub optText_Click(ByVal sender As Object, _
86     ByVal e As System.EventArgs) _
87     Handles optText.Click
88     'configure a ferramenta como Text
89     Me.CurrentTool = DrawingTools.Text
90     'oculte todos os outros painéis de ferramentas
91     pnlPenOptions.Visible = False
92     pnlShapeOptions.Visible = False
93     pnlTextOptions.Visible = True
94 End Sub
95
96 Private Sub FillLists()
97     With cboColors.Items
98         .Add("Black")
99         .Add("Red")
100        .Add("Green")
101        .Add("Blue")
102    End With
103    cboColors.SelectedIndex = 0
104
105    With cboShapeType.Items
106        .Add("Rectangle")
107        .Add("Ellipse")
108    End With
109    cboShapeType.SelectedIndex = 0
110
111    With cboTextFont.Items
112        .Add("Arial")
```


ENTRADA**LISTAGEM 17.9** Propriedades e Código Geral do Aplicativo Scribble
(*continuação*)

```
113         .Add("Times New Roman")
114         .Add("Courier New")
115     End With
116     cboTextFont.SelectedIndex = 0
117 End Sub
```

ANÁLISE

Primeiro declaramos uma nova enumeração (linhas 1 a 5). Essa enumeração será usada para registrar as ferramentas disponíveis e assegurar que a propriedade `CurrentTool` (declarada posteriormente) só possa ser configurada com um dos valores válidos. Se depois você quiser adicionar outras ferramentas de desenho, deverá fazê-lo nessa enumeração.

As linhas 7 a 9 são as variáveis membro das três propriedades expostas pelo formulário. Essas propriedades (linhas 16 a 23, 25 a 32 e 34 a 45) são basicamente simples, com uma exceção. No trecho `Set` da propriedade `CurrentTool`, as linhas 41 a 43 se destacam. As ferramentas de desenho como canetas, pincéis e fontes se encontram em um local especial no Windows. Só uma quantidade limitada desses componentes fica disponível. Portanto, você deve sempre se lembrar de chamar o método `Dispose` desses componentes para assegurar que eles sejam liberados para o próximo usuário. Em geral, é possível chamar `Dispose` diretamente nesses objetos. O código da linha 42 é necessário porque `oTool` foi declarada como um objeto na linha 12. Esse é um exemplo dos sacrifícios feitos quando se escreve um código genérico; às vezes isso significa mais trabalho adiante.

O evento `Form Load` (linhas 47 a 61) é o local onde configuramos o formulário, carregando as listas suspensas e preparando os elementos gráficos do desenho. A linha 51 chama a rotina `FillLists` que escreveremos posteriormente. A seguir, nas linhas 53 a 55, criamos um novo `Bitmap` e o atribuímos a `PictureBox`. Esse é o `Bitmap` no qual desenharemos. Os parâmetros usados definem o tamanho da nova figura e o tipo de `Bitmap` que será criado. Aqui geramos uma figura de 24 bits. Isso significa que podemos empregar qualquer uma das 16777216 cores quando selecionarmos uma.

A linha 56 extrai o objeto `Graphics` de `Bitmap` e o atribui ao formulário. Usaremos esse objeto mais tarde quando formos desenhar. Para criar uma superfície adequada e limpa para o desenho, aplicaremos `Clear` ao objeto `Graphics` configurado-o como `White`, e para assegurar que tenhamos uma ferramenta definida, selecionamos `Pen`.

Todas as três rotinas a seguir executam a mesma tarefa. São usadas quando o usuário seleciona uma nova ferramenta. Essas rotinas configuram `CurrentTool` como a ferramenta desejada e asseguram que o conjunto correto de opções da ferramenta atual fique visível e os outros não.

O procedimento `FillLists` (linhas 96 a 105) preenche os controles `DropDownList` com várias cores, formas e fontes e seleciona o primeiro item de cada. Você pode estender isso se quiser incluir outras cores, formas e fontes.

Desenhando Formas

Depois que você tiver um objeto `Graphics`, precisará saber como usá-lo. O objeto `Graphics` possui vários métodos que são empregados para incluir diversas linhas, formas e texto em uma superfície. Esses métodos se encontram em duas categorias amplas:

- **Métodos Draw** Estes métodos são utilizados para o desenho com o uso da ferramenta Pen (caneta). Em geral, só criam uma forma vazia.
- **Métodos Fill** Estes métodos são utilizados para o desenho com o uso da ferramenta Brush (pincel). Criam uma forma que é preenchida com uma cor ou textura.

A Tabela 17.13 descreve alguns dos métodos mais usados da classe `Graphics`.

TABELA 17.13 Métodos da Classe `System.Drawing.Graphics`

<i>Método</i>	<i>Descrição</i>
<code>Clear</code>	Remove todo o conteúdo do objeto <code>Graphics</code> , substituindo-o pela cor solicitada.
<code>DrawEllipse</code>	Desenha uma elipse ou círculo no objeto <code>Graphics</code> usando a caneta que for atribuída. Se a altura e a largura forem iguais, você terminará com um círculo; caso contrário, ela será oval.
<code>DrawLine</code>	Desenha uma linha no objeto <code>Graphics</code> usando a caneta atribuída.
<code>DrawRectangle</code>	Desenha um retângulo ou quadrado no objeto <code>Graphics</code> usando a caneta atribuída. Se a altura e a largura forem iguais, você terá um quadrado; senão, aparecerá um retângulo.
<code>DrawString</code>	Escreve um texto no objeto <code>Graphics</code> .
<code>FillEllipse</code>	Preenche uma área oval ou circular do objeto <code>Graphics</code> usando o pincel atribuído.
<code>FillRectangle</code>	Preenche uma área retangular ou quadrada do objeto <code>Graphics</code> usando o pincel atribuído.

Agora estamos prontos para adicionar o código do aplicativo `Scribble` que fará realmente o desenho. Quando estivermos no modo `Shape` ou `Text`, desenharemos um retângulo ou uma elipse, ou adicionaremos um texto onde o usuário der um clique. No entanto, o desenho de linhas é um pouco diferente. Na verdade não podemos desenhar uma linha quando o usuário der um clique no botão do mouse; devemos aguardar que ele mova o mouse. Em vez disso, quando o mouse estiver com o botão pressionado, apenas o definiremos como inativo. Depois, quando o usuário mover o mouse, se seu botão estiver pressionado, começaremos a desenhar. A Listagem 17.10 mostra o código dos manipuladores de evento de `PictureBox`.

CÓDIGO**LISTAGEM 17.10** Manipuladores de Evento de PictureBox

```
1 Private Sub picDraw_MouseMove(ByVal sender As Object,_
2     ByVal e As System.Windows.Forms.MouseEventArgs)_
3     Handles picDraw.MouseMove
4     If Me.Drawing Then
5         'isso só será verdadeiro se a ferramenta atual for uma caneta
6         oGraphics.DrawLine(oTool, sngX, sngY, e.X, e.Y)
7         sngX = e.X
8         sngY = e.Y
9         'force um redesenho
10        Me.picDraw.Refresh()
11    End If
12 End Sub
13
14 Private Sub picDraw_MouseUp(ByVal sender As Object,_
15     ByVal e As System.Windows.Forms.MouseEventArgs)_
16     Handles picDraw.MouseUp
17     'agora podemos parar de desenhar
18     Me.Drawing = False
19 End Sub
20
21 Private Sub picDraw_MouseDown(ByVal sender As Object,_
22     ByVal e As System.Windows.Forms.MouseEventArgs)_
23     Handles picDraw.MouseDown
24     'comece a desenhar
25     'Shape e Text são formas, Pen funcionará ao mover o mouse
26     'precisamos criar a ferramenta
27     'e desenhar ou estar prontos para isso
28     Select Case Me.CurrentTool
29     Case DrawingTools.Shape
30         Select Case Me.cboShapeType.Text
31         Case "Rectangle"
32             oGraphics.FillRectangle(_
33                 New SolidBrush(_
34                     Color.FromName(Me.cboColors.Text)),_
35                 e.X,_
36                 e.Y,_
37                 Me.updShapeWidth.Value,_
38                 Me.updShapeHeight.Value)
39         Case "Ellipse"
40             oGraphics.FillEllipse(_
41                 New SolidBrush(_
42                     Color.FromName(Me.cboColors.Text)),_
43                 e.X,_
44                 e.Y,_
```


CÓDIGO**LISTAGEM 17.10** Manipuladores de Evento de PictureBox (continuação)

```

45         Me.updShapeWidth.Value, _
46         Me.updShapeHeight.Value)
47     Case Else
48     End Select
49     'force um redesenho
50     Me.picDraw.Refresh()
51 Case DrawingTools.Text
52     'crie uma fonte
53     oTool = New System.Drawing.Font(_
54         Me.cboTextFont.Text, Me.updFontSize.Value)
55     'escreva o texto no local em que o mouse está agora
56     oGraphics.DrawString(Me.txtText.Text, _
57         oTool, _
58         New SolidBrush(_
59             Color.FromName(Me.cboColors.Text)), _
60         e.X, e.Y)
61     'force um redesenho
62     Me.picDraw.Refresh()
63 Case DrawingTools.Pen
64     'crie a caneta (para desenhar quando o mouse se mover)
65     oTool = New System.Drawing.Pen(_
66         Color.FromName(Me.cboColors.Text), _
67         Me.updPenWidth.Value)
68     sngX = e.X
69     sngY = e.Y
70     Me.Drawing = True
71 End Select
72 End Sub

```

ANÁLISE

As linhas são desenhadas durante o evento `MouseMove` de `PictureBox`. Só queremos desenhar a linha se estivermos no modo de desenho; portanto, a primeira etapa é testar a propriedade `Drawing` (linha 4). Essa propriedade será configurada como `True` posteriormente no evento `MouseDown`.

No modo de desenho, a primeira etapa é desenhar uma linha usando a caneta atual (linha 6). As variáveis `sngX` e `sngY` são variáveis privadas no nível do formulário usadas para registrar a última posição do mouse. Sempre que desenharmos uma linha, o faremos a partir dessa posição armazenada até a posição atual do mouse. A posição atual do mouse é fornecida como propriedade da classe `MouseEventArgs` passada para o manipulador de eventos. Copiamos a posição atual no local de armazenamento temporário nas linhas 7 e 8 para a próxima vez que o evento for chamado. Para concluirmos, na linha 10, forçamos uma atualização de `PictureBox`. Isso garante que qualquer alteração que tenhamos feito seja exibida para o usuário.

O manipulador do evento `MouseUp` (linhas 14 a 19) é simples. Só será usado para encerrar o modo de desenho se estivermos nele. Posteriormente, os eventos `MouseMove` sem `Drawing` configurado como `True` não possibilitarão desenho algum.

Se a ferramenta atual for `Shape`, precisaremos determinar o tipo de forma a selecionar. O código do desenho é semelhante. Usamos o método `Fill` para desenhar um retângulo ou uma elipse na posição atual do mouse com os tamanhos selecionados nos controles `NumericUpDown`. A posição atual do mouse é fornecida pelo parâmetro `MouseEventArgs` do manipulador do evento `MouseDown`.

Se a ferramenta atual for `Text`, a primeira coisa a fazer será criar uma fonte nova (linhas 53 a 54). Essa fonte será usada quando escrevermos o texto em `PictureBox` (linhas 56 a 60). Ele será escrito na fonte gerada, com um pincel recém-criado.

Para concluir, se estivermos desenhando linhas com a ferramenta `Pen`, teremos de criar o novo objeto `Pen` (linhas 65 a 67). Ele terá a cor selecionada e a largura da caneta. A seguir, armazenaremos a posição atual do mouse para uso posterior no manipulador do evento `MouseMove` (linhas 68 a 69) e ativaremos o modo de desenho (linha 70).

Salvando Figuras

Depois de criar uma figura, use o método `Save` do objeto `Bitmap` para armazená-la como um arquivo. Esse método permite que a figura seja salva com um tamanho e formato específicos.

Vários formatos diferentes têm suporte do objeto `Bitmap`. Entre eles estão:

- **Bitmap** Um formato comum de figuras usado no Windows, sendo normalmente bem amplo. Pode dar suporte a uma quantidade ilimitada de cores, e a extensão utilizada é `BMP`.
- **GIF** Abreviatura de `Graphics Interchange Format`; este é um formato comum de figuras usado na Internet. Normalmente, os desenhos `GIF` são pequenos porque são compactados; no entanto, só podem empregar 256 cores por figura. A extensão utilizada para esses arquivos é `GIF`.
- **JPEG** Abreviatura de `Joint Photographic Experts Group`; este é outro formato comum de figura usado na Internet. O formato `JPEG` normalmente é empregado para armazenar fotografias ou outras figuras que precisem de muitas cores para ser exibidas. Ele pode ser bastante compactado; no entanto, arquivos altamente compactados podem perder informações ou qualidade. A extensão para estes arquivos é `JPG`.
- **PNG** Abreviatura de `Portable Network Graphics`; este é um formato de arquivo relativamente incomum. Foi criado para substituir os arquivos `GIF` e `TIFF`. São arquivos bastante pequenos, embora em geral não tanto quanto nos formatos `GIF` ou `JPEG`. O suporte é raro, mesmo com o Internet Explorer e outros programas podendo exibi-los. A extensão para estes arquivos é `PNG`.
- **TIFF** Abreviatura de `Tagged Image File Format`; este é um formato raramente encontrado que costumava ser comum. Os arquivos `TIFF` podem ser grandes ou pequenos, de-

pendendo de qual das muitas variantes desse formato seja empregada para armazená-los. A extensão TIF é a usada nestes arquivos.



NOTA

Em geral, quando exibimos a figura em uma página da Web, a salvamos no formato GIF (se tiver menos de 256 cores) ou JPEG. Como alternativa, se você salvar a figura para usar em um aplicativo de microcomputador, deve testar todos os formatos para ver qual fornece a melhor combinação de tamanho do arquivo e qualidade para atender suas necessidades. Normalmente, o formato JPEG é adequado quando a figura usa muitas cores (como uma fotografia), enquanto o PNG ou o BMP são interessantes para figuras mais simples. Como exemplo, a Tabela 17.14 mostra as dimensões do mesmo bitmap simples salvo em cada um dos formatos.

TABELA 17.14 Comparação dos Formatos de Figuras

<i>Formato</i>	<i>Tamanho do arquivo</i>
BMP	340.134
GIF	10.088
JPG	20.971
PNG	10.262
TIFF	15.492

17

Agora podemos terminar nosso aplicativo adicionando o código aos comandos do menu File como vemos na Listagem 17.11.

ENTRADA

LISTAGEM 17.11 Comandos do Menu

```

1 Private Sub mnuFileNew_Click(ByVal sender As System.Object, _
2     ByVal e As System.EventArgs) Handles mnuFileNew.Click
3     Me.oGraphics.Clear(Color.White)
4     'force a atualização
5     Me.picDraw.Refresh()
6 End Sub
7
8 Private Sub mnuFileOpen_Click()
9     If Me.dlgOpen.ShowDialog = DialogResult.OK Then
10         Me.picDraw.Image.FromFile(Me.dlgOpen.FileName)
11     End If
12 End Sub
13
14 Private Sub mnuFileSave_Click(ByVal sender As Object, _
15     ByVal e As System.EventArgs) Handles mnuFileSave.Click
16     Dim sFileName As String
17     Dim oFormat As System.Drawing.Imaging.ImageFormat
18
19     'obtenha o nome do arquivo com o qual salvará
20     If dlgSave.ShowDialog = DialogResult.OK Then

```


ENTRADA**LISTAGEM 17.11** Comandos do Menu (*continuação*)

```

21      sFileName = dlgSave.FileName
22      Select Case dlgSave.FilterIndex
23          Case 0 'salve como bitmap
24              oFormat = System.Drawing.Imaging.ImageFormat.Bmp
25          Case 1 'salve como PNG
26              oFormat = System.Drawing.Imaging.ImageFormat.Png
27          Case Else
28              'não deve acontecer
29      End Select
30      'possível exceção ao salvar
31      Try
32          Me.picDraw.Image.Save(sFileName, oFormat)
33
34      Catch ex As Exception
35          'por enquanto apenas exiba
36          MessageBox.Show(ex.Message, Erro ao salvar arquivo", _
37                          MessageBoxButtons.OK, MessageBoxIcon.Error)
38      End Try
39  End If
40 End Sub
41
42 Private Sub mnuFileExit_Click(ByVal sender As System.Object, _
43     ByVal e As System.EventArgs) Handles mnuFileExit.Click
44     Me.Close()
45 End Sub

```

ANÁLISE

As linhas 1 a 6 mostram o código do comando New, do menu File. Ele é simples; tudo o que faz é apagar o conteúdo existente no objeto `oGraphics`, na linha 3, substituindo-o por um novo e adequado plano de fundo branco. Como antes, forçamos a atualização de `PictureBox` (linha 5) para que exiba a figura alterada.

Abrir o arquivo é quase tão fácil quanto apagar a figura. Na linha 9 do código exibimos a caixa de diálogo File Open. Se o usuário selecionar um arquivo, o carregaremos no objeto `Image` de `PictureBox`. Esse controle determinará o tipo do arquivo e o exibirá adequadamente.

O código necessário para salvar a figura é um pouco mais longo, mas continua simples. Começamos (linha 20) exibindo a caixa de diálogo-padrão File Save. Depois que o usuário tiver selecionado o local e o tipo do arquivo a ser salvo, poderemos continuar. Para resumirmos as últimas linhas, salvamos o nome do arquivo selecionado em uma variável. Nossa caixa de diálogo solicita ao usuário para selecionar entre dois formatos diferentes de figura – Bitmap ou PNG. Usamos essa opção na configuração de outra variável para o formato que empregaremos ao salvar a figura. Por fim, estamos prontos para salvá-la. Para o caso de uma exceção, como falta de espaço em

disco, inseriremos a figura em um bloco Try...End Try. A linha 32 representa a operação efetiva de salvar, se beneficiando da capacidade de que todos os objetos possuem de salvar a si mesmos.

Finalmente, o comando Exit do menu File fecha o formulário, encerrando o aplicativo.

Salvar figuras armazenadas em um objeto Image é fácil, com o é carregar uma nova figura. Na verdade, esse é um dos melhores exemplos dos benefícios de usar objetos porque trabalha-se com o que seria uma tarefa complexa e o código necessário é encapsulado em objetos Graphics, Image e Bitmap.

Resumo

Aprender todas as classes, estruturas, propriedades, métodos e eventos que compõem o .NET Framework leva algum tempo. No entanto, conhecer algumas classes fundamentais o ajudará sempre, já que os conceitos representados por elas se repetem em todo o Framework. Depois que você tiver conhecido um fluxo, toda vez que se deparar com um, saberá o que fazer. De maneira semelhante, após aprender a desenhar com as classes Graphics, poderá fazê-lo em qualquer local.

No próximo capítulo, você aprenderá a adicionar os 'retoques finais' a nossos aplicativos.

P&R

P O Que É Unicode?

R Um problema mundial é que nunca houve uma linguagem com a qual todos pudessem reconhecer. Muitas delas usam o mesmo conjunto de caracteres ou um semelhante ao empregado na América do Norte e Europa Ocidental. Esses caracteres foram definidos como o conjunto ASCII (American Standard Code for Information Interchange). Ele utiliza 255 caracteres (de um byte) para representar os que são usados nessas linguagens. Por exemplo, a letra 'A' é representada pelo algarismo 65 em todos os conjuntos de caracteres que usam a codificação ASCII.

No entanto, outros idiomas, como o tâmil ou o japonês, precisam de uma quantidade muito maior de caracteres. Conseqüentemente, o conjunto de caracteres Unicode foi definido pela International Standards Organization (ISO). O Unicode usa dois bytes para codificar cada símbolo de qualquer dos idiomas terrestres (e alguns extraterrestres – há uma conversão de Klingon para Unicode em <http://anubis.dkuug.dk/jtc1/sc2/wg2/docs/n1643/n1643.htm>). Por exemplo, o caractere Unicode 20AC sempre representará o euro. Os primeiros 256 caracteres da codificação Unicode são os mesmos da ASCII por conveniência.

Portanto, uma resposta direta seria que é um valor de dois bytes capaz de identificar um símbolo usado em qualquer conjunto de caracteres. Por que se preocupar? Em algum momento, você verá um grupo de caracteres que se parecerão com ?????, e já saberá o motivo. Provavelmente serão Unicode, e é preciso saber como convertê-los em ASCII ou exibir os caracteres nativos.

P Quero desenhar uma forma complexa. Há comandos em `System.Drawing` para desenhar formas como octógonos também?

R Vários métodos do objeto `Graphics` podem ser usados para criar formas mais complexas. Entre eles estão incluídos `DrawArc`, `DrawCurve`, `DrawLines`, `DrawPath` e `DrawPolygon`. Para desenhar um octógono, você deve empregar o método `DrawPolygon`, passando a caneta que utilizará no desenho, e um array de pontos (`Points`). Esse array descreverá cada ângulo do octógono.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Qual dos trechos de código a seguir você usaria para saber se um arquivo existe?

A. `Dim oFile As New File`

`Dim bExists As Boolean = oFile.Exists("algum arquivo")`

B. `Dim bExists As Boolean = Directory.Exists("algum arquivo")`

C. `Dim bExists As Boolean = File.Exists("algum arquivo")`

D. `Dim oDir As New Directory`

`Dim bExists As Boolean = oDir.Exists("algum arquivo")`

2. Qual é o relacionamento entre as classes `Stream`, `StreamReader` e `StreamWriter`?

3. Que formatos de figuras você pode criar (ou exibir em um controle `Image`)?

4. Qual é a diferença entre `Point`, `Size` e `Rectangle`?

Exercícios

1. Um dos problemas do programa `Note` que escrevemos é que ele permite o uso dos comandos `Cut` (Recortar), `Copy` (Copiar) e `Paste` (Colar) sem haver nenhum texto selecionado ou na área de transferência. Veja se consegue descobrir uma maneira fácil de ativar e desativar esses itens de menu antes que o menu seja exibido para o usuário. (Dica: adicione um evento ao item de menu `mnuEdit`.)

2. Altere o formato usado para salvar figuras e compare os tamanhos dos arquivos resultantes. Qual é o menor formato para desenhos simples em preto-e-branco? E para fotos?

SEMANA 3

DIA 18

Retoques Finais

Este livro é dedicado a ensiná-lo a programar com a plataforma .NET, mas há muitos aspectos a considerar no desenvolvimento de um aplicativo completo. Esta lição abordará alguns detalhes que é preciso compreender para a criação de um sistema real, principalmente se for grande e complexo, incluindo:

- A documentação de seu aplicativo.
- O uso do controle do código-fonte.

Esses dois tópicos são opcionais; você não precisa executá-los para produzir um sistema que funcione, mas fazem parte de quase todo grande projeto de desenvolvimento.

Documentando Seu Aplicativo

Às vezes, quando se cria um sistema, todo o código é escrito e distribuído para os usuários, e ninguém o vê novamente. Essa é uma exceção, e não a regra, e provavelmente só ocorre com sistemas pequenos. É mais comum que o código seja revisto várias vezes quando o sistema original for alterado para a inclusão de recursos e correção de erros, podendo ser usado até mesmo como base para versões completamente novas do mesmo sistema. Para fazer alterações e reutilizar seu código da maneira mais fácil possível para você ou outros programadores, é recomendável documentar seu sistema.

Esse tipo de documentação é diferente de registrar o sistema para seus usuários, o que discutirei posteriormente nesta lição. Ela fornece informações sobre o funcionamento de seu sistema para

as pessoas que têm acesso ao código e varia amplamente na quantidade de detalhes porque não há um método exclusivo de documentá-lo. Um caminho para se chegar a uma documentação adequada é considerar primeiro o motivo pelo qual essas informações são fornecidas. A documentação existe para assegurar que todo o conhecimento sobre um sistema fique registrado – sejam os requisitos da empresa ou a lógica por trás das decisões técnicas – para simplificar a manutenção por meio de mais informações disponíveis sobre seu código. A documentação pode ser dividida em algumas diretrizes essenciais (não necessariamente por ordem de importância), e a primeira não está de maneira alguma ligada à documentação:

- Crie a solução mais simples que puder para resolver o problema.
- Documente seu sistema com o menor nível de especulação possível.
- Use seu tempo de maneira sensata. Não comente o óbvio; em vez disso, empenhe-se em esclarecer as partes complexas de seu sistema.
- Explicações detalhadas de seu código serão irrelevantes se não houver uma compreensão da finalidade geral do sistema. Documente o sistema, cada um de seus componentes e o código propriamente dito.

Crie Soluções Mais Simples

Se o motivo da documentação de seu código for facilitar sua compreensão, não seria melhor criá-lo da maneira mais simples possível desde o início? Seu objetivo deve ser um código que seja inteligível sem qualquer explicação adicional, e os comentários devem complementá-lo onde você não conseguir atingir esse resultado. Há duas maneiras principais de chegar a isso: escrever um código claro e desenvolver soluções simples. Abordarei o conceito de código limpo primeiro, mas a questão mais significativa decerto é a criação de soluções simples.

Escrevendo um Código Limpo

Considere as Listagens 18.1 e 18.2, sabendo-se que as duas foram projetadas para executar a mesma tarefa.

LISTAGEM 18.1 O Código Pode Ser Mais Complexo do Que É Necessário

```
1 Option Explicit On
2 Option Strict On
3
4 Imports System
5 Imports System.IO
6
7 Module Module1
8
9     Sub Main()
10         Dim Path1 As String = "C:\file.txt"
11         Dim Path2 As String = "C:\file_new.txt"
```


LISTAGEM 18.1 O Código Pode Ser Mais Complexo do Que É Necessário (*continuação*)

```
12 Dim File1 As New StreamReader(Path1)
13 Dim File2 As New StreamWriter(Path2)
14 Dim x As String
15 x = File1.ReadLine()
16 Do Until x Is Nothing
17     File2.WriteLine(x)
18     x = File1.ReadLine()
19 Loop
20 File2.Close()
21 File1.Close()
22 End Sub
23
24 End Module
```

LISTAGEM 18.2 Tornando Seu Código Mais Fácil de Compreender, Você Poderá Ter uma Documentação Menor

```
1 Option Explicit On
2 Option Strict On
3
4 Imports System
5 Imports System.IO
6
7 Module WorkWithFiles
8
9     Sub CopyFiles()
10         Dim sInputFilePath As String = "C:\test.txt"
11         Dim sOutputFilePath As String = "C:\output.txt"
12         Dim fileInput As New StreamReader(sInputFilePath)
13         Dim fileOutput As New StreamWriter(sOutputFilePath)
14         Dim sLine As String
15
16         sLine = fileInput.ReadLine()
17         Do Until sLine Is Nothing
18             fileOutput.WriteLine(sLine)
19             sLine = fileInput.ReadLine()
20         Loop
21
22         fileOutput.Close()
23         fileInput.Close()
24
25     End Sub
26
27     Sub Main()
```

LISTAGEM 18.2 Tornando Seu Código Mais Fácil de Compreender, Você Poderá Ter uma Documentação Menor (*continuação*)

```
28      CopyFiles()  
29  End Sub  
30  
31 End Module
```

ANÁLISE

As Listagens 18.1 e 18.2 lêem um arquivo e o gravam em outro, porém a segunda é muito mais clara do que o mesmo código na Listagem 18.1. A primeira razão é o próprio nome do módulo e da sub-rotina; os valores-padrão não fornecem nenhuma informação, enquanto o nome `CopyFiles` para o procedimento é uma indicação bem direta da finalidade da rotina. A próxima diferença é o espaçamento. Na Listagem 18.2, as linhas em branco são usadas para reunir trechos relacionados no código, agrupando o código de inicialização (linhas 10 a 14), o de leitura de arquivo (linhas 16 a 20) e o de limpeza (linhas 22 e 23). Essas linhas em branco não terão absolutamente nenhum efeito sobre o resultado final desse sistema – os dois exemplos serão executados com a mesma velocidade obtendo resultados iguais –, mas agrupar o código o torna mais legível.

A diferença essencial restante entre os dois exemplos de código está na nomeação da variável, um tópico sobre o qual discorri brevemente em lições anteriores. As variáveis podem ser nomeadas de qualquer forma, contanto que você não use uma palavra-chave ou outro valor reservado à plataforma .NET. Podemos considerar o nome da variável como um mecanismo interno de documentação. A Listagem 18.1 é um exemplo extremado; grande parte dos códigos que vemos não é tão ruim no que diz respeito à nomeação de variáveis. Mas essa é uma tentativa de ilustrar o efeito de variáveis mal nomeadas. Referir-se aos dois arquivos como `fileInput` e `fileOutput` (comparados com `File1` e `File2`) documenta a finalidade das variáveis sem a necessidade de qualquer comentário adicional. Opções semelhantes foram feitas para os nomes das outras variáveis, e quando elas forem usadas juntas (como na linha 12 da Listagem 18.2), a finalidade do código estará quase completamente explicada apenas pelos nomes empregados.

Uma prática comum e útil na codificação também é usada na escolha dos nomes das variáveis; a cada nome é dado um prefixo com um ou mais caracteres para indicar o tipo de dado da variável. Na Listagem 18.2, as strings têm o prefixo `s`, e os objetos `StreamReader/StreamWriter` empregam `file` para indicar sua finalidade de ler e gravar arquivos. Esse método de nomeação de variáveis é baseado na *notação húngara*, em que os caracteres iniciais são empregados para descrever o tipo de dado e o escopo (global *versus* local, público *versus* privado) das variáveis. Digo que é baseado nesse tipo de notação, em vez de dizer que ‘é’ a Notação Húngara porque em geral aplico um padrão de nomeação muito menos rígido do que o conjunto original de regras descrito como notação húngara por seu criador (Charles Simonyi da Microsoft). Abordarei a nomeação de variáveis ainda nesta lição, na seção “Melhores Práticas e Padrões de Codificação”.

Uma questão final sobre a criação de um código simples, que não é demonstrada no exemplo anterior, é como a estrutura de seu código pode facilitar a sua compreensão. Gosto de gerar meus sistemas usando muitos procedimentos, mesmo em códigos que não serão necessariamente reutilizados de algum modo. O resultado de estruturar seu sistema dessa maneira é que procedimentos de nível mais alto podem ser lidos e compreendidos, ao mesmo tempo em que também evitam os detalhes mais complexos (e mais confusos). A Listagem 18.3 fornece um exemplo.

LISTAGEM 18.3 Os Procedimentos São Outra Maneira de Facilitar a Compreensão de Seu Código

```
1 Module StartUp
2     Public Sub Main()
3         Dim sPassword, sData, sFilePath As String
4         sFilePath = "C:\Input.txt"
5         InitializeSystemVariables()
6
7         sPassword = GetPassword()
8         sData = ReadData(sFilePath)
9         sData = EncryptData(sPassword, sData)
10        WriteData(sFilePath, sData)
11    End Sub
12 End Module
```

A Listagem 18.3 não executa nada sozinha; a essência real desse sistema está contida nos procedimentos à que ele faz referência. Apenas ler essa rotina dará a você uma boa idéia do que o sistema realiza, e é possível examinar as rotinas individuais se necessário. Se seu código for estruturado de modo correto, rotinas de nível superior poderão agir como uma visão da *Reader's Digest* de todas as rotinas de nível inferior que elas chamam. Vale a pena observar que, embora possamos ler o código da Listagem 18.3 e compreender o que está ocorrendo, apenas com uma documentação apropriada teríamos certeza de sua finalidade!

Simplificando Seus Sistemas

Criar um código limpo é um aspecto da primeira diretriz, mas não é o mais importante. O mais relevante é que a solução efetiva gerada, ou seja, como você decidiu atingir os objetivos do sistema, seja a mais simples possível.

Evitando Conjecturas

Ao começar a documentar seu código, é provável que tenha em mente (de maneira consciente ou inconsciente) um público para esses comentários. Poderá escrever como se fosse o único a ler esses comentários, ou talvez você possa imaginar que outros desenvolvedores de sua equipe atual sejam as pessoas que precisarão compreender seu código para efetuar manutenções. O problema

em considerar um público-alvo é que haverá a tendência de escrever uma documentação diferente para cada tipo de pessoa.

Não pressuponha que os comentários serão lidos apenas por pessoas com experiência em C++ ou mesmo em Visual Basic .NET. Não imagine que eles serão examinados por indivíduos que compreendam o objetivo final do sistema ou que conheçam algo, em qualquer nível, sobre o lado operacional da abordagem. Suas pressuposições podem parecer válidas – talvez você tenha uma equipe só de especialistas –, mas tudo muda. Seu código poderia terminar nas mãos de uma firma de consultoria encarregada de atualizá-lo para uma empresa que comprasse aquela em que você está atualmente, ou um trecho individual de seu código poderia ser usado em outro projeto.

Mesmo as suposições mais simples, que o leitor compreenda os termos de bancos de dados, por exemplo, devem ser abordadas no começo de qualquer documento que você criar. É claro que há um limite para essa regra; se você não pudesse pressupor pelo menos um conhecimento básico de Visual Basic, então, teria de explicar cada aspecto individual do código.

Não Comente o Óbvio, Só o Que For Confuso

Esta regra, na verdade, está relacionada intimamente com a primeira porque quanto mais simples e claro você puder tornar seu código, menos comentários serão necessários. Considere as Listagens 18.1 e 18.2 e como poderia comentá-las para assegurar que tenham uma manutenção fácil. No final, precisará gastar muito mais tempo comentando o primeiro exemplo, explicando o significado de variáveis como `Path1` e `Path2`, enquanto o segundo exemplo já é de alguma maneira auto-explicativo. Um nome de variável como `sInputFilePath` informa tudo que é preciso saber, sem ser necessário adicionar qualquer comentário. O uso da notação húngara, abordada posteriormente nesta lição, junto a nomes significativos para as variáveis fornece até informações adicionais, permitindo saber que, nesse caso, trata-se de uma variável string e reduzindo também a necessidade de comentários.

Documente o Sistema, e Não Apenas o Seu Código

A questão final de nossos princípios gerais de documentação é que você precisa sempre se concentrar no objetivo, que é facilitar a outras pessoas a compreensão de seu sistema. Documentar cada variável com detalhes e cada laço não me ajudará a entender seu sistema se eu não souber qual a sua finalidade. Forneça uma documentação com vários níveis distintos, desde a abordagem e escopo originais por trás do sistema até os detalhes de cada linha de código. Os diferentes trechos de uma documentação que deveriam estar disponíveis sobre um sistema incluem os descritos a seguir:

- Abordagem/escopo
- Requisitos/situação na qual utilizá-lo
- Projeto detalhado
- Algum modo de solicitação de um histórico/alterações do sistema

A definição de cada um desses tipos de documentação varia bastante para metodologias de desenvolvimento e pessoas diferentes, mas a intenção básica do documento é minha única preocupação aqui. Os detalhes sobre o objetivo geral dessa documentação são fornecidos nas seções a seguir, porém a pergunta que você provavelmente se fará quando os examinar será: “Tenho de escrever tudo isso? Pensei que tivesse terminado!”. Essa é uma boa pergunta. Se não existir nenhuma documentação e seu sistema estiver concluído, não há chance de que ela seja criada. Mas não é assim que deve funcionar. A idéia é que essa documentação seja gerada antes que o sistema seja desenvolvido, a sua maioria antes que qualquer código efetivo seja escrito a não ser um protótipo e alguma parte dela durante a própria fase de codificação. No Dia 1, “Bem-Vindo ao Visual Basic .NET”, abordei o ciclo de vida do desenvolvimento dos softwares, mas veja novamente como fonte de referência (veja a Figura 18.1).

FIGURA 18.1

O ciclo de vida no desenvolvimento de softwares é um processo que não tem fim, em geral representado como um círculo que se conecta novamente a ele próprio para indicar um ciclo contínuo.



Abordagem/Escopo

A documentação sobre a abordagem/escopo descreve o objetivo definido para o sistema e deve ser determinada no início do desenvolvimento. Como referência para fins de manutenção, é bom começar a compreender qual a finalidade do sistema desde o início. Um exemplo de abordagem e escopo em seu nível mais simples poderia ser:

“O sistema de Registro das Habilidades do Funcionário armazenará uma listagem dos funcionários e registrará seu nível de especialização em várias aptidões relacionadas ao cargo com a finalidade de desenvolver equipes e planejar projetos.”

Observe que essa declaração não explica todos os detalhes: que habilidades? Quem usará o sistema? Onde será obtida a lista de funcionários? Essas e outras perguntas não fazem parte da abordagem. Elas precisam ser respondidas, mas serão encontradas em outra documentação.

Requisitos e Situações em Que Será Utilizado

A documentação sobre os requisitos e a utilização pode ser feita em separado ou em conjunto. Independentemente de sua escolha, a mesma finalidade será atendida.

Os requisitos de um sistema documentam as funções individuais que os usuários executarão com ele, e as situações em que será utilizado tentam descrever interações representativas que acontecerão com o seu uso. Cada um fornece uma listagem de funções que o sistema tem de realizar e pretende passar uma idéia de como essas funções devem funcionar. Esses documentos disponibilizam uma visão geral do aplicativo do ponto de vista do usuário, mas será preciso procurar informações adicionais sobre como o sistema executa realmente essas funções.

Projeto Detalhado

Muitos documentos se enquadram nessa categoria – qualquer item que forneça mais detalhes do que a documentação sobre os requisitos. A intenção dos documentos dessa categoria é detalhar como o sistema funciona: que etapas segue internamente para executar os requisitos do usuário descritos no documento anterior. Espere ver fluxogramas, diagramas de estado e listas de componentes, procedimentos e outros elementos de código como parte desse material.

Histórico de Alterações

Se os aplicativos atendessem exatamente suas especificações quando fossem implantados, seria um milagre. Se há uma constante no desenvolvimento de aplicativos, é que sempre são feitas alterações. Grande parte do trabalho de desenvolvimento de um sistema é registrar e controlar essa alteração, impedindo-a de desviar totalmente o projeto do que foi planejado. Esse rastreamento é em geral conhecido como *controle de alterações* e envolve o registro de cada alteração solicitada, obedecendo algum processo de aprovação para determinar quais delas puderam ser incorporadas ao sistema e, em seguida, a documentação dessas decisões e o efeito que cada alteração produz no planejamento do projeto. Se as que ocorrerem em seu sistema forem documentadas, esse material será um recurso inestimável para os desenvolvedores que estiverem trabalhando em novos recursos e na manutenção dos existentes, e para os projetistas e gerentes que planejarão novas versões de seu software.

Melhores Práticas e Padrões de Codificação

Quando se trata da documentação de seu código, haverá algumas ‘práticas mais adequadas’ que vale a pena usar ou pelo menos considerar. A primeira é estabelecer um conjunto de padrões comuns (comuns pelo menos para sua equipe) para nomear variáveis, controles e objetos. A segunda é fornecer blocos de cabeçalho em seu código – comentários no início dos módulos, classes e procedimentos que usem um formato comum e disponibilizem informações sobre esse código em um único local. Abordarei cada uma delas e descreverei alguns exemplos de como manipulá-las dentro de sua equipe.

Nomeação de Variáveis, Controles e Objetos

A questão mais importante com relação ao modo de nomeação é que ele tem de ser um padrão; todas as pessoas de sua equipe precisam usar as mesmas convenções de nomeação sem exceção. Na verdade, a nomeação deve ser apenas uma parte de um documento maior que detalhe os comentários, a manipulação de exceções, o controle do código-fonte e mais – tudo que o desenvolvedor precisar saber para funcionar como parte de sua equipe. Se você puder fazer com que todos empreguem seu padrão de nomeação, o estabelecimento efetivo desse padrão será uma facilidade.

A notação húngara é uma maneira de nomear, abordada de modo resumido anteriormente nesta lição, que usa prefixos para fornecer informações sobre a variável, controle ou objeto. Esses prefixos são compostos de uma ou mais letras e são projetados para representar pelo menos o tipo de dado e às vezes também o escopo do objeto retirado. Em meu código, emprego um conjunto bem simples de convenções de nomeação que evoluíram para um conjunto de convenções que realmente utilizo. Aprofundarei minha abordagem das convenções, mas há algumas diretrizes gerais para discutir antes.

- Só uso essa notação para indicar o tipo de dado; não adiciono informações sobre o escopo, a não ser incluir o símbolo `m_` para indicar uma variável membro (privada) de uma classe. Não especificar o escopo é uma questão de simplificação porque raramente emprego algo diferente de variáveis no nível do procedimento ou da classe. Dessa maneira não preciso ter nenhum tipo de indicador do escopo nas variáveis em nível de procedimento.
- Todas essas convenções de nomeação se aplicam a nomes internos, como variáveis privadas, controles de seu formulário e assim por diante. Ao criar classes, propriedades públicas e métodos públicos, você definirá uma interface externa, e esses nomes devem ser simples e legíveis por qualquer pessoa. Por exemplo, posso chamar a variável de um contador interno de `iCount` (para indicar que é um número inteiro), mas uma propriedade semelhante em uma classe pública deveria ser chamada apenas de `Count`.
- O objetivo final é tornar seu código mais e não menos legível. Além da notação húngara, certifique-se de que os próprios nomes das variáveis tenham um significado!

Em todos os exemplos de nomeação a seguir *alterno maiúsculas e minúsculas*, onde o prefixo fica em minúsculas e o nome da variável em maiúsculas, produzindo nomes mais legíveis.

Tipos de Dados

Em geral uso o menor número de caracteres possível em minha notação dos tipos de dados e, portanto, nem mesmo tento empregar a mesma quantidade para todos eles (diferentemente da referência anterior ao artigo da Microsoft). Lembre-se de que forneço minha notação na Tabela 18.1, mas incluo alternativas comuns entre parênteses, onde apropriado. Não seria prático abordar todos os tipos de dados da plataforma .NET, inclusive todas as classes do .NET Framework, de modo que tentarei mostrar exemplos suficientes para lhe dar uma idéia de como nomeio minhas variáveis.

TABELA 18.1 Prefixos de Tipos de Dados Comuns

<i>Tipo de Dado</i>	<i>Notação Húngara</i>
String	S [str]
Integer (Inteiro), Int32	I [int]
Double (Duplo)	Dbl
Single (Simples)	sng [f (para flutuante, um tipo da C++)]
Long (Longo), Int64	L [lng]
Boolean (Booleano)	B [bool]
Date/Time (Data/Hora)	dt
System.Data.DataSet	ds
System.Data.DataColumn	dc
Object (Objeto)	obj

Para algo que seja bem incomum, que eu use freqüentemente, costumo empregar apenas um nome descritivo com o prefixo obj, como em objWebRequest.

Controles

Os exemplos da Tabela 18.2 se aplicam igualmente aos controles usados nos formulários da Web e aos controles Windows.Forms.

TABELA 18.2 Prefixos de Controles Comuns

<i>Controle</i>	<i>Notação Húngara</i>
Botão	btn [cmd (em versões anteriores ao VB.NET)]
Título (Label)	lbl
Caixa de texto	txt
Caixa de lista (List Box)	lb
Caixa de combinação	cbo
Botões de opção	opt
Caixas de seleção (Check Boxes)	chk
Grade de dados	dg
Itens de menu	mnu

Blocos de Comentário

No início de cada procedimento e classe, você pode incluir um comentário amplo de várias linhas descrevendo esse trecho do código. Esses blocos de comentário são muito usados como o local onde se fornece qualquer tipo de informação aplicável ao trecho completo do código em vez de a uma única linha ou estrutura. Em geral utilizo as informações a seguir nos cabeçalhos de comentário da classe (incluindo o módulo):

- Nome (da classe)
- Dependências (referências requeridas por essa classe, outras classes, requisitos do sistema como o SQL Server)
- Finalidade
- Autor (em geral o autor original)
- Quem editou pela última vez
- Data da última edição
- Comentários (observações sobre problemas, erros encontrados e corrigidos, qualquer item que não seja específico de uma linha dentro da classe e que valha a pena ressaltar)

A Listagem 18.4 mostra um exemplo de um bloco de comentário de uma classe.

LISTAGEM 18.4 Blocos de Comentário Consistentes Valem o Esforço

```

1 Imports System
2 Imports System.Data
3 Imports System.Data.SqlClient
4 '*****
5 'Nome da classe:           GetAuthors
6 'Dependências:           System.Data, o SQL Server deve estar disponível
7 'Finalidade:             Acessar o SQL Server e recuperar dados
8 '                       na tabela Authors do banco de dados Pubs
9 '                       Sample
10 'Autor:                  Duncan Mackenzie
11 'Editado pela última vez por:Duncan Mackenzie
12 'Editado pela última vez em:09/08/2001
13 'Comentários:            Alterado para que o nome do SQL Server seja passado
14 '                       em vez de estar embutido no código.
15 '*****
16
17 Public Class GetAuthors
18 End Class

```

18

No bloco de comentário de um procedimento, incluo um conjunto um pouco diferente de informações:

- Nome (do procedimento)
- Dependências (referências requeridas por essa sub-rotina ou função, outras classes, requisitos do sistema como o SQL Server)
- Finalidade
- Autor (em geral o autor original)
- Parâmetros (incluindo todos os possíveis se você tiver um bloco de comentário para várias versões sobrepostas, ou apresentará apenas a versão atual se um bloco de comentário for inserido antes de cada versão sobreposta)
- Quem editou pela última vez
- Data da última edição
- Comentários (observações sobre problemas, erros encontrados e corrigidos, qualquer item que não seja específico de uma linha dentro do procedimento e que valha a pena ressaltar)

Não é difícil escrever um suplemento para a plataforma .NET que lhe forneça pelo menos o modelo de um bloco de comentário e o poupe de alguma digitação, portanto se você tiver tempo, pode achar interessante desenvolver um como projeto de fim de semana.

Usando o Controle do Código-fonte

O código-fonte que você criar será valioso; ele representa o resultado de seu esforço e a base para trabalhos futuros. Tanto antes do desenvolvimento quanto depois do fato consumado, é preciso encontrar uma maneira de armazenar seu código. Os requisitos principais para armazenar esse código são que ele esteja seguro e as alterações fiquem registradas de modo que seja possível comparar duas versões diferentes. No Visual Studio .NET, são fornecidos recursos para o acesso a um sistema de controle do código-fonte, e o que a Microsoft disponibiliza é essencialmente o mesmo que foi incluído no Visual Studio 6.0, o Visual SourceSafe (VSS) 6.0c.

Para ativar o suporte ao VSS na plataforma .NET, apenas instale o servidor e o cliente (ou apenas o cliente, pressupondo que um servidor já exista em algum outro local) em sua máquina, e serão ativados os comandos do menu apropriado no Visual Studio .NET. Nesta seção, percorrerei os aspectos básicos do uso do controle do código-fonte, abordando como introduzir e extrair um código desse sistema, e como retornar uma alteração anterior.

Extraindo o Código

Depois de ativar o controle do código-fonte no Visual Studio .NET instalando a parte relativa ao cliente do VSS, você terá disponíveis algumas novas opções de menu. No menu File, um submenu chamado Source Control fornecerá várias opções, dependendo de um projeto estar aberto no momento. Caso contrário, haverá a opção de abrir um diretamente do sistema de controle do código-fonte, porém, na primeira vez em que se usa o VSS, é mais sensato acessar um projeto e adicioná-lo a esse sistema. Qual projeto será aberto não é realmente importante no momento, mas

também podemos apenas criar um novo aplicativo do console para usá-lo neste exemplo. Chame-o de SCC Sample e insira o código mostrado na Listagem 18.5 em seu módulo, permitindo que eu aproveite e introduza uma pequena demonstração de acesso a conteúdo da Web enquanto examinamos o controle do fonte.

LISTAGEM 18.5 Recuperando uma Página da Web

```
1 Imports System.Net
2 Imports System.IO
3 Module SCC_Sample
4
5     Sub Main()
6         Dim sOutput As String
7         Dim sURL As String
8         sURL = "http://msdn.microsoft.com"
9         Try
10            Dim objNewRequest _
11                As WebRequest = HttpWebRequest.Create(sURL)
12            Dim objResponse _
13                As WebResponse = objNewRequest.GetResponse
14            Dim objStream _
15                As New StreamReader(objResponse.GetResponseStream())
16            sOutput = objStream.ReadToEnd()
17        Catch eUFE As UriFormatException
18            sOutput = "Error in URL Format: [" & sURL & "]" & _
19                System.Environment.NewLine() & eUFE.Message
20        Catch eWEB As WebException
21            sOutput = "Error With Web Request: " & _
22                & System.Environment.NewLine() & eWEB.Message
23        Catch e As Exception
24            sOutput = e.ToString
25        Finally
26            Console.Write(sOutput)
27        End Try
28        Console.ReadLine()
29    End Sub
30 End Module
```

Agora, salve todos os seus arquivos selecionando File, Save All no menu. Com o projeto iniciado e contendo alguns dados, a próxima etapa é inseri-lo em seu sistema de controle do código-fonte. Selecione a opção Microsoft Visual SourceSafe no menu File e, em seguida, escolha Source Control e finalmente dê um clique em Add Solution to Source Control.

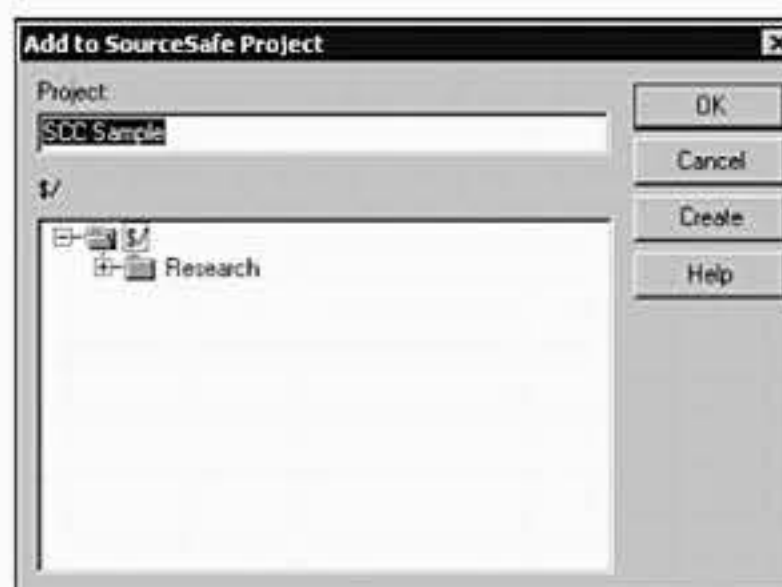


As identificações e senhas do usuário são necessárias no Visual Source Safe, e você pode configurá-las executando o programa Administrator na pasta Visual Source Safe do menu Programs. Por enquanto, se for solicitado a inserir uma identificação de usuário ou senha, use apenas Admin e um espaço em branco para a senha (que estão configurados como o padrão).

Isso abrirá uma caixa de diálogo que lhe permitirá escolher um caminho dentro do sistema de código-fonte para armazenar sua solução e os projetos dela (veja a Figura 18.2).

FIGURA 18.2

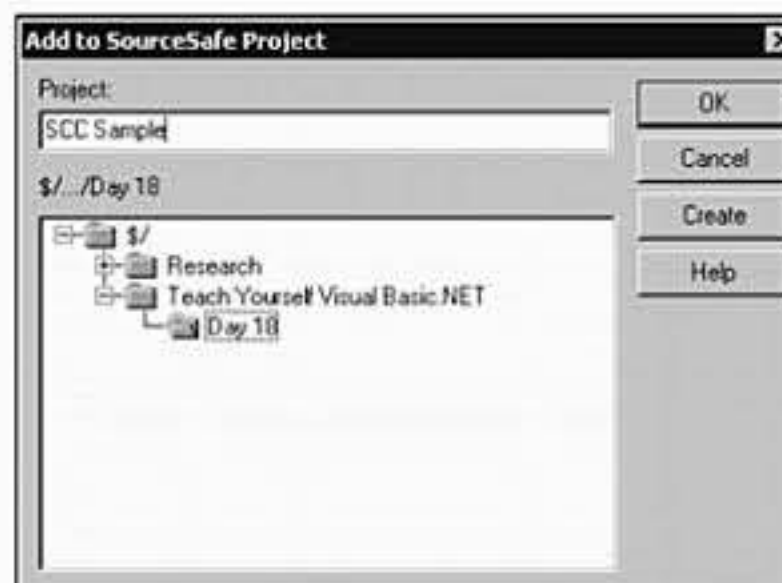
Ao incluir seu código, você terá de escolher onde armazenar o código-fonte.



Sua caixa de diálogo provavelmente não exibirá nada além da raiz da árvore de controle do código-fonte, \$/, mas já existem alguns projetos em meu sistema de controle. É possível criar subdiretórios digitando o nome desejado e dando um clique em Create. Certifique-se de que a raiz esteja selecionada na caixa de diálogo e em seguida, digite **Teach ourself Visual basic .NET** (Ensinamos a você o Visual Basic .NET) e dê um clique em Create. Agora, o novo diretório estará selecionado, que é o que queríamos, portanto, digite **Day 18** e dê um clique em Create. Para concluir, digite o nome que deseja para sua solução (SCC Sample é uma boa escolha; veja a Figura 18.3) e dê um clique em OK. Você será questionado se o projeto já não existe e se quer criá-lo automaticamente, que é o desejado, então, dê um clique em Yes.

FIGURA 18.3

Você pode criar uma hierarquia completa para armazenar seus projetos com base nas categorias que queira usar.

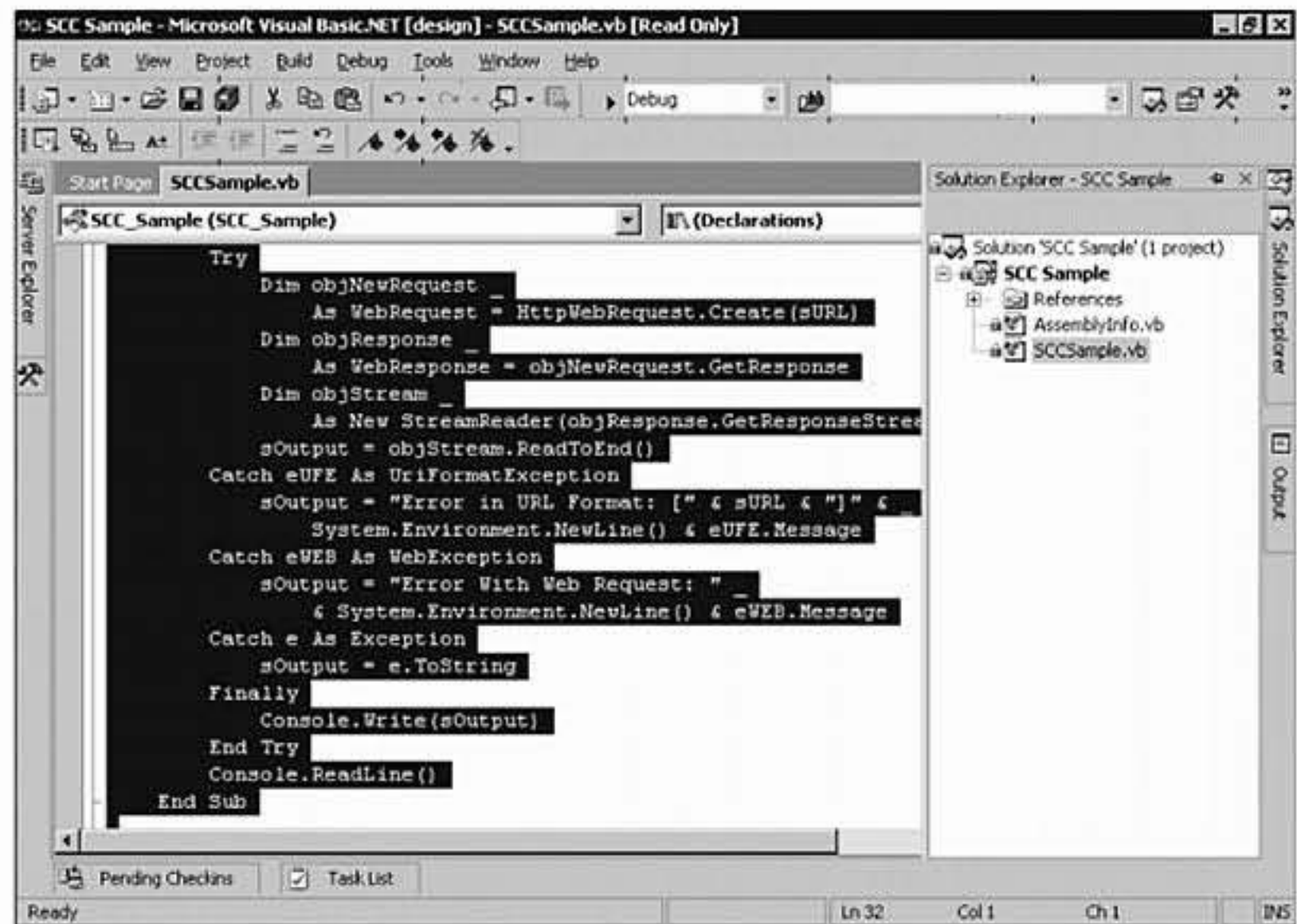


Agora, o Visual Studio .NET adicionará esse projeto ao controle do código-fonte e o armazenará ao mesmo tempo, o que significa que você terá de extraí-lo antes de poder alterá-lo. Observe que

seus arquivos a partir desse momento serão apenas de leitura, como indicado na barra de título do Visual Studio .NET e do Solution Explorer (veja a Figura 18.4).

FIGURA 18.4

Uma das intenções do controle do código-fonte é permitir que o desenvolvedor só edite um código de cada vez. Portanto, apenas o código que for extraído poderá ser alterado.



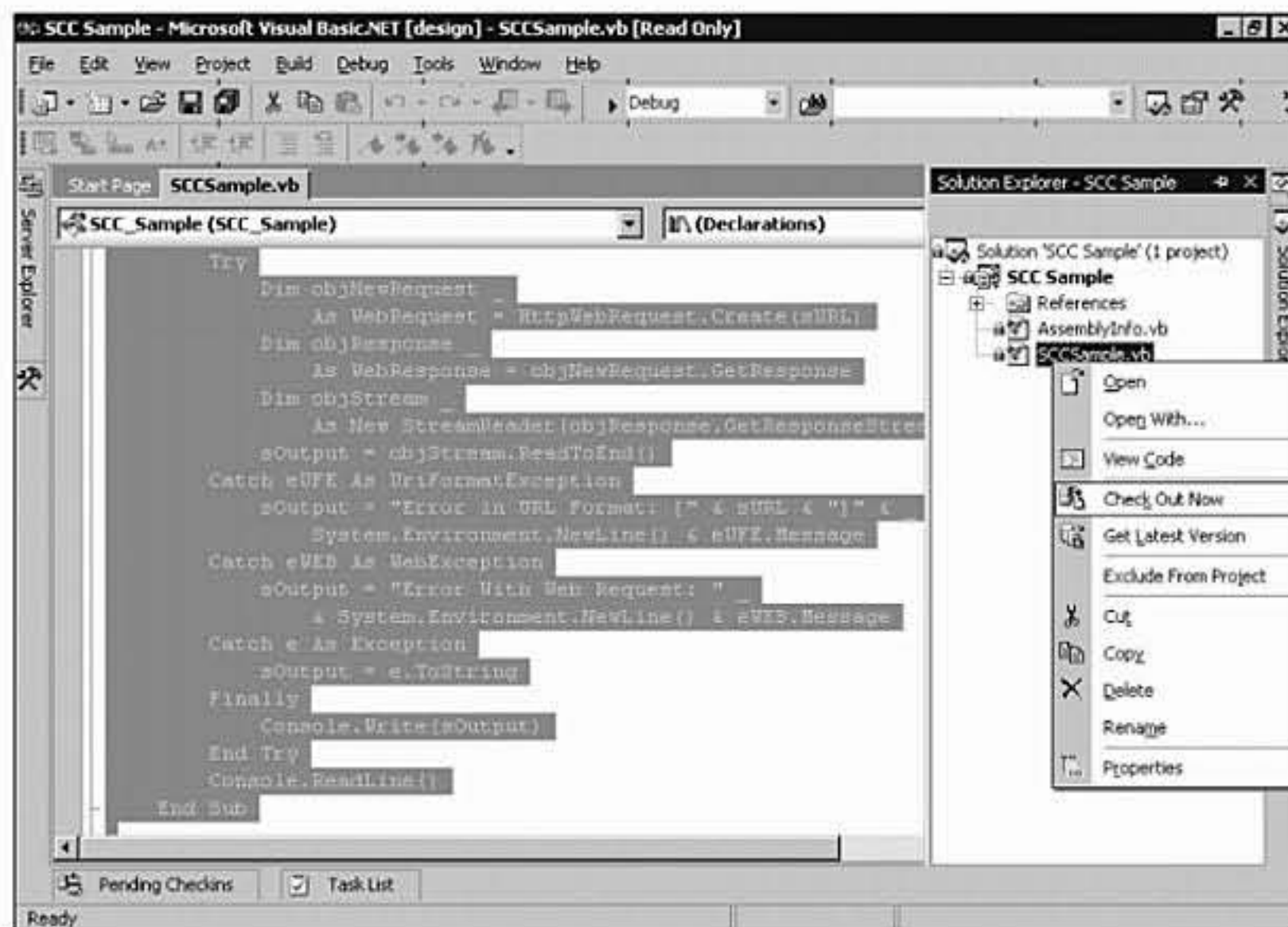
Nesse momento, você poderá extrair seu código, porque por fim o inseriu completamente pela primeira vez. Selecione o módulo no Solution Explorer, que nomeei como `SCCSample.vb` em meu projeto, e dê um clique nele com o botão direito do mouse. No menu suspenso que surgirá, haverá uma nova opção, **Check Out Now** (veja a Figura 18.5), que permitirá que só esse arquivo seja extraído e editado. Até que um arquivo seja extraído, ele será apenas de leitura e quando for acessado, poderá ser editado somente na máquina onde for recuperado.

Outra opção, que pode ser mais útil, é **Check Out Now (Recursive)**, mostrada na Figura 18.6, que extrai a solução ou projeto e todos os arquivos dele, e estará disponível se você der um clique com o botão direito do mouse no projeto ou solução. Essa opção aparecerá, para que seja tudo extraído de uma só vez, apenas se **Display Silent Check-Out Command in Menus** for selecionado. Esse recurso encontra-se na pasta **Source Control** da caixa de diálogo de opções, que pode ser acessada selecionando-se os itens de menu **Tools** e **Options**.

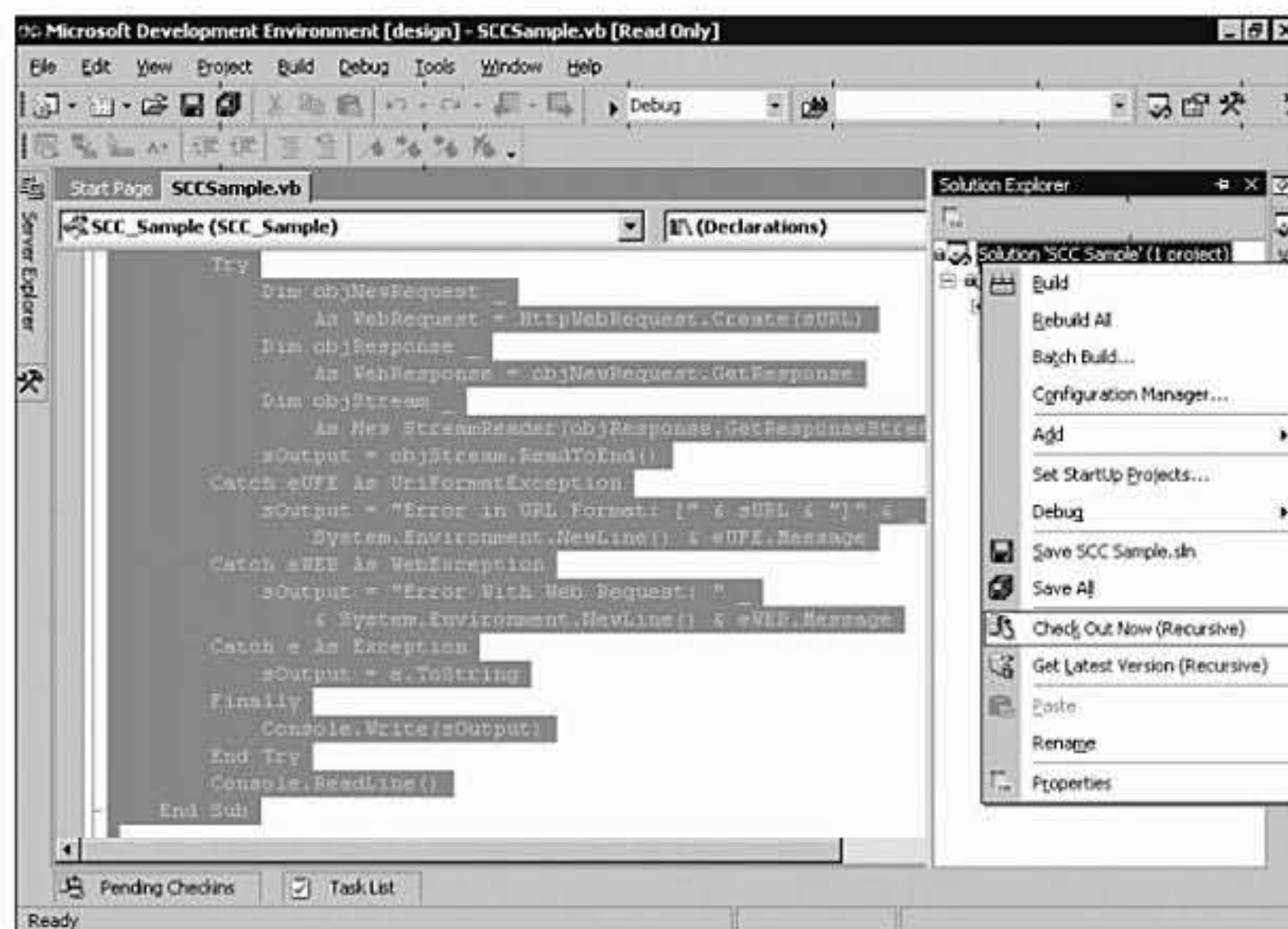
Depois que um arquivo for extraído, ele apresentará esse status por meio de um novo ícone próximo a seu nome no Solution Explorer, como mostrado na Figura 18.7, e pelo fato de que não será mais apenas de leitura. Agora você pode fazer quantas alterações quiser nesses arquivos, mas elas não serão refletidas no sistema **Source Safe** até que eles sejam introduzidos novamente nesse local.

FIGURA 18.5

Você pode extrair individualmente qualquer arquivo com o qual deseje trabalhar.

**FIGURA 18.6**

Usar a extração recursiva para acessar todos os arquivos de seu projeto de uma só vez, em geral é mais fácil do que extraí-los um de cada vez.

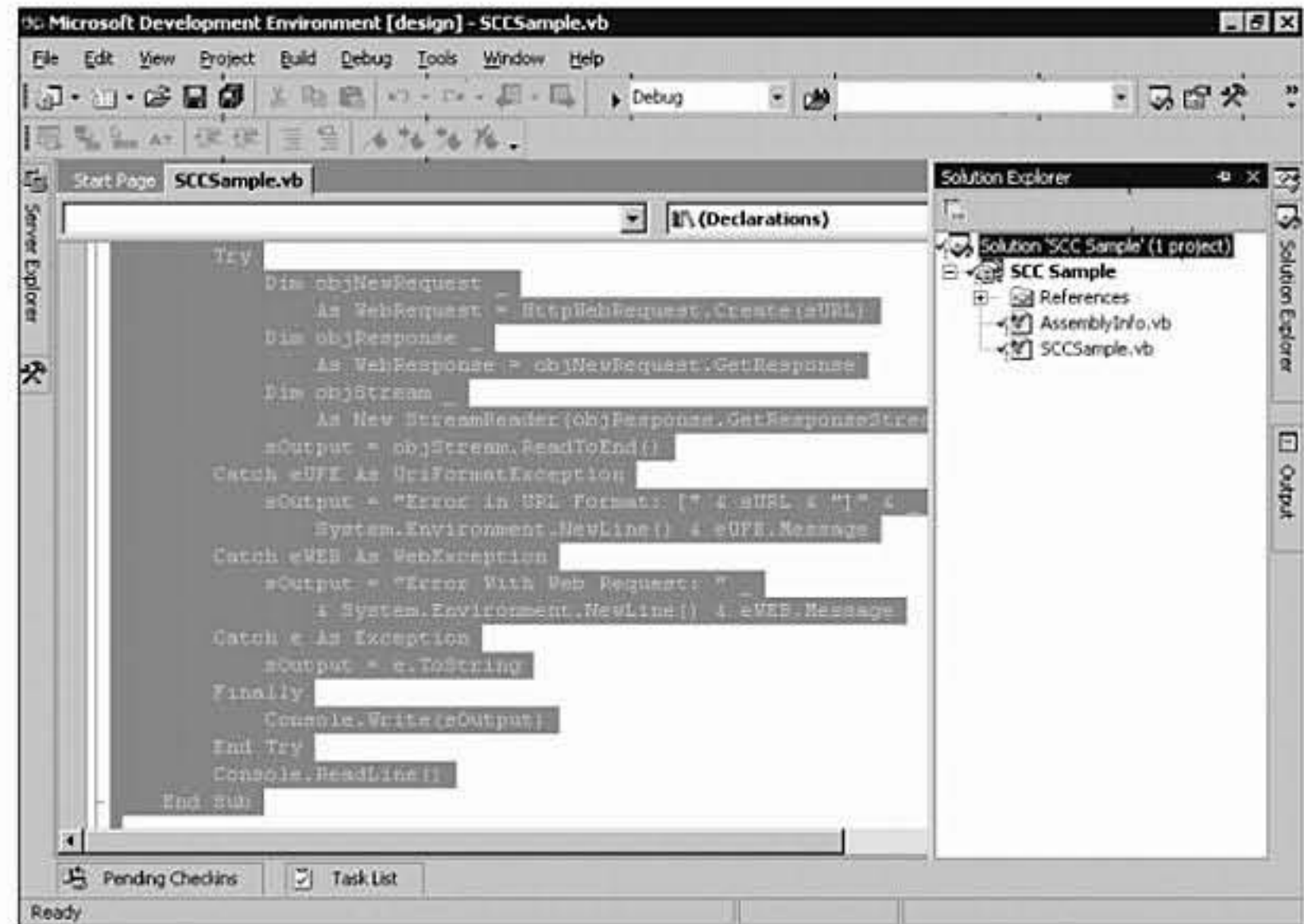


Armazenando o Código

Quando um código for extraído, outros programadores que usarem o sistema do código-fonte só terão acesso a uma cópia de leitura dele; você terá acesso exclusivo à cópia que extraiu. Não é nossa intenção manter os arquivos fora do sistema se não os estivermos usando, a menos que conscientemente não pretendamos que eles sejam alterados por mais ninguém, portanto armazene seu código mais uma vez depois que tiver feito (e, em geral, testado) suas alterações.

FIGURA 18.7

O Solution Explorer indicará se um arquivo está armazenado ou não, depois que você optar por usar o controle do código-fonte em seu projeto.

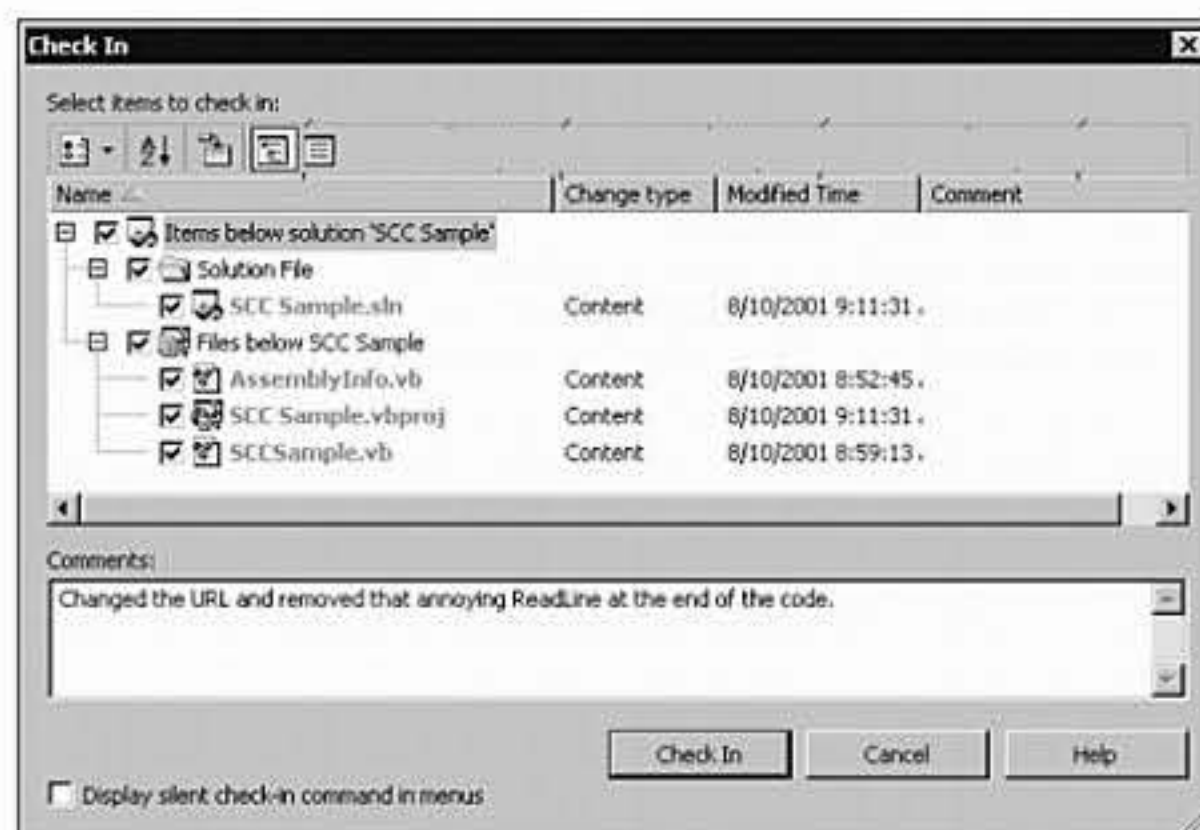


No código de seu exemplo, que, em conformidade com o desenrolar desta explicação, no momento não foi extraído, faça algumas alterações no módulo, como remover `Console.ReadLine` da linha 28 e alterar o trecho do URL na linha 8 para `http://www.microsoft.com`. Agora, salve seus arquivos e, em seguida, você poderá armazená-los novamente em Source Safe. Há várias maneiras de armazenar arquivos, mas a mais simples é dar um clique com o botão direito do mouse na solução ou projeto (o mesmo nível do qual foram retirados) e selecionar Check In. Isso abrirá uma caixa de diálogo (veja a Figura 18.8) que permitirá a especificação de que arquivos da solução devem ser incluídos juntamente com um comentário explicando o que foi alterado e qual o motivo.

18

FIGURA 18.8

Quando você armazenar novamente os arquivos, poderá adicionar comentários para documentar as alterações que fez.

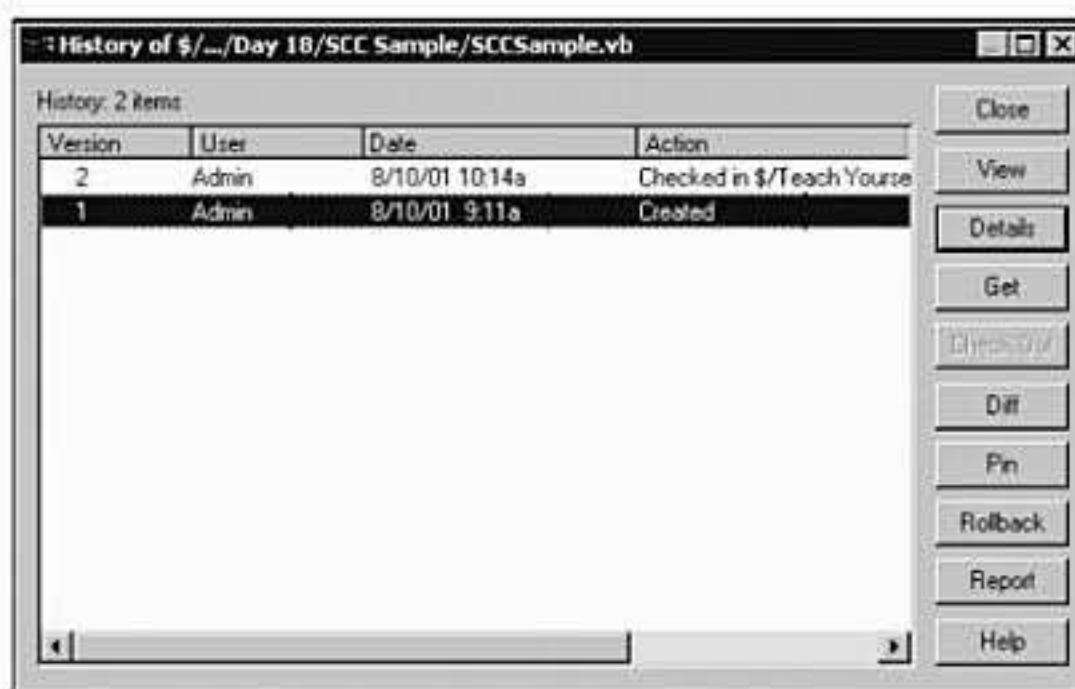


Visualizando e Retornando Suas Alterações

Agora, com seus arquivos armazenados novamente, você pode manipular alguns dos recursos mais avançados da utilização do controle de código-fonte. O primeiro que você poderá testar é a visualização do histórico de um arquivo, exibindo cada inclusão/extração em que alterações foram feitas. Selecione o módulo no Solution Explorer e, em seguida, selecione File, Source Control e History no menu. Isso fará com que você possa configurar a listagem do histórico, mas apenas dar um clique em OK trará uma listagem das versões disponíveis desse arquivo (veja a Figura 18.9).

FIGURA 18.9

Toda vez que você incluir novamente um código-fonte alterado no sistema de controle, essa nova versão será mantida para referências futuras.

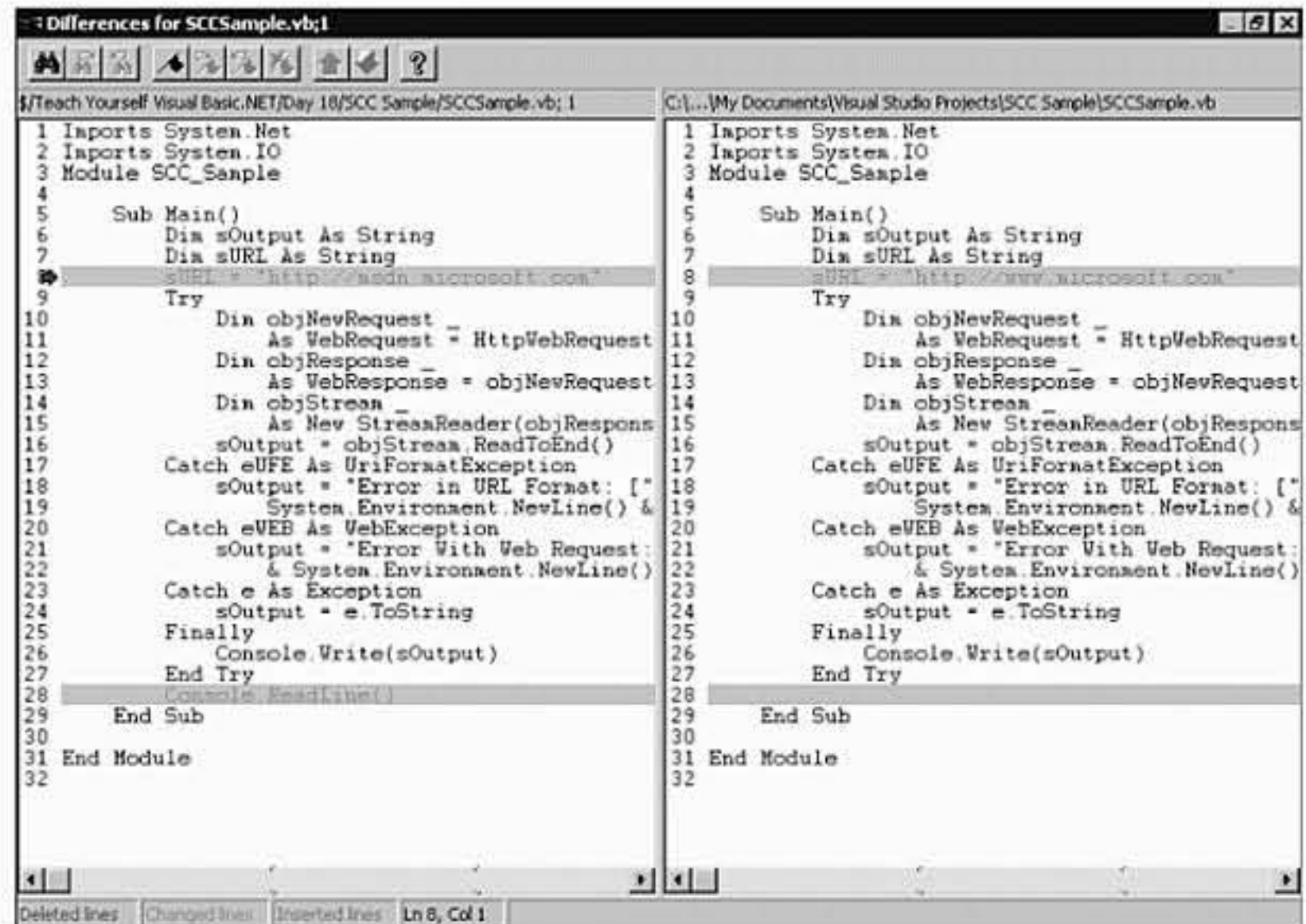


Selecione a primeira versão, que representa o arquivo original introduzido em Source Safe e, em seguida, dê um clique no botão Diff. Isso abrirá uma caixa de diálogo na qual será possível se certificar de que a opção Visual está selecionada para que, em seguida, você possa dar um clique em OK. Agora, será exibida uma janela (veja a Figura 18.10) que detalha o estado atual do arquivo e como ele era na versão selecionada e descreve as diferenças. Esse formato visual torna claro o que foi alterado, mas ocupa muito espaço para mostrar as alterações de um arquivo-fonte grande. Os outros dois formatos (Unix e Source Safe) fornecem um breve resumo das alterações entre as duas versões, o que pode ser mais de seu gosto.

Voltando à janela History, você pode retornar uma versão anterior de um arquivo, se quiser desfazer muitas alterações e não se preocupar em perder tudo o que foi alterado de uma versão para a outra. Retornar a versão, comparado a visualizar as diferenças e desfazer manualmente uma quantidade selecionada de alterações, é uma operação radical, mas às vezes é a única maneira de voltar a um estado conhecido que funcione. Selecione a primeira versão desse arquivo na lista do histórico e, em seguida, dê um clique no botão Rollback. Será exibido um aviso de que essa ação não pode ser desfeita, mas vá em frente e execute-a de qualquer modo. Agora, só haverá uma versão. Suas cópias locais serão atualizadas para essa versão, e se você ou outra pessoa acessasse esse arquivo em Source Safe, essa é a versão que receberiam. Todas as alterações feitas após a versão selecionada foram totalmente removidas.

FIGURA 18.10

O Source Safe pode fornecer uma interface visual para detalhar as alterações entre uma versão e outra. Isso torna fácil localizar a origem de um problema em uma versão específica.



Considerações sobre Segurança no Uso do Visual Source Safe

O usuário que armazenou o arquivo aparecerá na caixa de diálogo History, e nesse caso ele será sempre 'Admin'. É importante que todos se conectem com seu próprio nome de modo que você possa saber quem fez quais alterações em seu código-fonte e para que as senhas sejam configuradas a fim de que ninguém possa fazer alterações usando o nome de outra pessoa. O Visual Source Safe não parece poder ser integrado à segurança do Windows 2000/NT, mas é possível contornar isso criando usuários com os mesmos nomes das identificações de seus desenvolvedores na rede e fazendo-os usá-las quando se conectarem ao VSS. Utilizando a interface do Visual Source Safe, podemos configurar a segurança no nível do projeto com base na identificação de logon do usuário, o que permite limitar o acesso a um projeto quando necessário.

Resumo

A documentação é considerada por muitas pessoas como um mal necessário na programação. Em geral é feita no final de um projeto, de maneira apressada, fornecendo uma quantidade mínima de comentários para que você possa dizer que está tudo concluído. A chave para tornar a documentação útil e muito menos difícil de ser terminada é trabalhar nela durante o desenvolvimento. Seguir as diretrizes desta lição fará com que seu código fique tão fácil de compreender quanto for possível e os comentários preencherão qualquer falha no entendimento que possa existir.

P&R

P O comentário não é apenas uma maneira de compensar um código confuso? Se seu código for bem escrito, não deverá precisar de nenhum comentário.

R Tenho ouvido essa declaração algumas vezes e ela é parcialmente verdadeira. Quanto mais claro for seu código, menos comentários serão necessários. Coloque em prática esse pensamento, e você concluirá que um código realmente claro não precisa de comentários. Concordo, mas creio que qualquer programa composto de mais de 10 linhas de código nunca conseguirá ser tão claro porque o público em potencial que irá lê-lo é muito variado. Faça com que seu código fique tão claro quanto possível, diminuindo assim os comentários, mas comentar um pouco sempre será necessário.

P Achei que a plataforma .NET incluísse comentários com base em XML, mas você não a abordou nesta lição.

R Infelizmente, o comentário com base em XML é um recurso da C#, e não do Visual Basic .NET (ou de qualquer outra linguagem .NET). Se você estiver interessado, examine os materiais de referência da C# que vêm com o .NET Framework para obter mais detalhes sobre os comentários em XML.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Com base nos padrões de nomeação abordados nesta lição, o que você pode dizer de uma variável chamada `sConnectionInfo`?
2. O Visual Source Safe fornece serviços de controle do código-fonte ao Visual Studio .NET. Cite no mínimo duas razões pelas quais você gostaria de usar um sistema de controle do código-fonte.
3. Qual o nome dado quando se usa o sistema de código-fonte para que a versão anterior de um código volte a ser usada?

SEMANA 3

DIA 19

Implantando Seu Aplicativo

Depois de desenvolver um aplicativo, você terá de fazê-lo chegar a seus usuários. O processo de passar um sistema da etapa de produção à utilização é conhecido como *implantação* e em geral é considerada uma das fases mais difíceis de qualquer projeto de desenvolvimento. Esta lição abordará as etapas básicas da implantação de um aplicativo por meio do Visual Studio .NET, incluindo:

- A introdução à implantação.
- O desenvolvimento de um programa de instalação no Windows.
- As opções de distribuição de um programa de instalação.

A plataforma .NET conseguiu reduzir bastante as dificuldades relacionadas à implantação de aplicativos, mas esse ainda é um processo necessário e geralmente mais complicado do que parece.

Introdução à Implantação

Implantar um aplicativo é, falando de maneira bem simples, disponibilizá-lo para as pessoas que irão usá-lo. A implantação terá significados distintos para tipos diferentes de aplicativos. Para um aplicativo da Web, ela pode significar colocar seu código e páginas em um servidor Web, enquanto para um aplicativo Windows, pode ser disponibilizar seu arquivo .exe para todos os usuários. Em geral, lidamos com vários arquivos diferentes, e cada um apresenta seus próprios requisitos, complicando todo o processo. Uma *dependência* acontece quando um arquivo, que poderia ser seu executável, precisa de outro, como uma biblioteca (.dll), para ser executado cor-

retamente. Portanto, seu aplicativo pode depender de um ou mais arquivos, mas esses, por sua vez, também podem ter dependências e assim por diante. Implantar seu arquivo .exe de modo que ele funcione de maneira correta pode requerer a instalação de vários outros arquivos!

A dependência mais comum que encontraremos, porque ela será necessária a todos os aplicativos .NET que forem criados, é o .NET Framework. Como você deve se lembrar, o .NET Framework foi introduzido como parte da instalação do Visual Studio .NET e é composto de muitos arquivos que contêm todas as bibliotecas de classes, o Common Language Runtime (CLR) e outros componentes importantes. Para evitar que seja preciso incluir todos esses arquivos diferentes e suas dependências individualmente, um módulo de intercalação foi fornecido.

NOVO TERMO

O *módulo de intercalação* é um arquivo único que representa essencialmente todo um programa de instalação, podendo ser incluído como parte de sua implantação. Os módulos de intercalação (que são armazenados como arquivos .msm em disco) são uma maneira excelente de empacotar todas as configurações, dependências e arquivos necessários para uma biblioteca ou aplicativo. Você pode criar seus próprios módulos de intercalação, se já tiver uma biblioteca ou outro projeto que geralmente é incluído em outras instalações, mas o Visual Studio .NET possui muitos desses arquivos MSM prontos para serem usados. Esses módulos de intercalação, armazenados em `\Program Files\Commom Files\Merge Modules\`, incluem as partes do .NET Framework necessárias para implantar seu aplicativo em outra máquina.

Antes da plataforma .NET, a instalação quase sempre envolvia o registro de componentes do COM, o que significava a criação de entradas no registro do sistema para esses componentes de modo que eles pudessem ser encontrados e gerados pelos aplicativos que quisessem usá-los. Na plataforma .NET, essa necessidade de alterar o registro para instalar um aplicativo foi removida; todas as configurações e metadados sobre um aplicativo ou componente (como uma biblioteca de classes) são armazenados dentro do próprio arquivo ou como arquivos adicionais no mesmo diretório. Essa eliminação do registro permite instalações 'X-Copy', em que os aplicativos podem ser completamente instalados apenas por meio da cópia dos arquivos de uma máquina para outra.

Com isso em mente, a necessidade de criar uma instalação real não fica muito clara. Em geral, instalar um aplicativo significa mais do que apenas conseguir que ele seja executado na máquina de destino: precisam ser criados ícones na área de trabalho e no menu Iniciar, dependências como o .NET Framework têm de ser instaladas, uma opção de desinstalação precisa ser fornecida e, às vezes, até entradas no registro para as configurações do aplicativo têm de ser criadas. Todas essas ações, recursos comuns que podem fazer parte da instalação de muitos aplicativos, ficam disponíveis com a criação de um programa completo de instalação.

No Visual Studio .NET, a implantação é manipulada pela criação de arquivos do Windows Installer. O Windows Installer é uma maneira relativamente nova de tecnologia de instalação que tem um suporte maior do sistema operacional e está em uso por muitos produtos, inclusive o próprio Visual Studio .NET, o Microsoft Office e muitos outros. É suficiente dizer que ao em-

pregar essa tecnologia, seus programas de instalação poderão se beneficiar de muitos recursos excelentes, todos fazendo parte da plataforma SDK.

As páginas MSDN que fazem parte da plataforma SDK apresentam uma tendência incômoda de ter seus URLs alterados com o tempo, portanto o conduzirei na descida pela árvore da biblioteca em vez de fornecer um link direto. Acesse <http://msdn.microsoft.com/library> e, em seguida, localize o nó da árvore (no lado esquerdo da página) chamado Setup and System Administration. Abra esse nó e, então, prossiga na descida através de Setup, Windows Installer e SDK Documentation. No final, você estará no Windows Installer (ele aparece duas vezes no caminho; não se incomode com isso). O bom é que há informações disponíveis sobre o Windows Installer, mas talvez a melhor maneira de aprender seja testá-lo por sua própria conta.

Criando uma Instalação Simples

Pararei de falar sobre implantação e o conduzirei pela criação da instalação básica de um aplicativo simples. A fim de que esse seja um exemplo perfeito, você precisa de mais uma máquina (sem ser a que tem o ambiente .NET instalado), de preferência que não possua a plataforma .NET. Essa segunda máquina, se houver, deve ser conectada a uma primeira rede, ou será necessário algum outro meio de transferir arquivos grandes, como um gravador de CDs. Porém, se só houver uma máquina, o processo também poderá ser executado, mas a demonstração não estará tão próxima da realidade em que são processados os programas reais de implantação.

O objetivo deste exemplo é criar um aplicativo .NET simples, sem que seu conteúdo real tenha alguma importância. No entanto, você já leu neste livro que não criaríamos nenhum aplicativo do tipo 'hello world!' como versão de demonstração. Com isso em mente, precisamos gerar um aplicativo real para desenvolver nosso exemplo a partir dele. Entretanto, a simplicidade ainda é o objetivo, de modo que usaremos apenas um formulário conectado a um banco de dados do SQL Server e que exiba uma lista de autores (do banco de dados Pubs do exemplo). Isso será feito em apenas um formulário, o que não é a melhor maneira de escrever um aplicativo para uso no mundo real, mas queremos manter o exemplo simples e desenvolveremos outro mais apropriado um pouco mais adiante nesta lição. Abordarei o código brevemente; para obter mais detalhes, retorne ao Dia 12, "Acessando Dados com a Plataforma .NET", no qual encontrará uma abordagem do trabalho com bancos de dados e classes System.Data.

O exemplo será criado com a colocação de um novo aplicativo Windows em sua própria solução. Percorrerei cada etapa relevante, e você poderá acompanhar ou fazer o download do código completo que se encontra no site deste livro na Web.

Crie o Projeto e a Solução

Se você estiver com uma solução aberta, selecione File e Close Solution no menu para fechá-la. Agora, sem nada aberto, selecione New e Project no menu File. Aparecerá a caixa de diálogo New Project; selecione a pasta Visual Basic Projects e o modelo de projeto Windows Application. A seguir, certifique-se de que a versão completa dessa caixa de diálogo foi exibida dando um

clique no botão More se ele já não tiver sido expandido. Dê um nome apropriado ao projeto, como AuthorList e, em seguida, assegure-se de que a opção Create Directory For Solution esteja selecionada, permitindo que um nome seja fornecido para sua solução. Crie um nome para ela, como Pubs Demo, indicando que algo mais do que apenas esse projeto pode existir nesse local. Para concluir, dê um clique em OK a fim de gerar a solução e o projeto. Crie um diretório para qualquer solução que tenha mais de um projeto de modo que promova uma organização automática em seus arquivos, situando todas as propriedades que fazem parte da mesma solução juntas em um diretório. Um projeto e uma solução serão criados com os nomes que você escolheu.

Configure o Novo Formulário

No projeto do aplicativo Windows, um novo formulário terá sido criado por padrão, com o nome Form1. Renomeie esse formulário com frmDisplayAuthors que é mais explicativo e, em seguida, adicione um controle caixa de lista ao, até então, formulário vazio. Configure o nome da nova caixa de lista como lbAuthors, sua propriedade Dock como Fill, e IntegralHeight como False. Essas configurações criarão uma grade de dados que abrangerá todo o formulário, mesmo se ele for redimensionado pelo usuário. A configuração de IntegralHeight é exclusiva para caixas de lista e determina se o controle deve ser sempre dimensionado como um múltiplo da altura de um único item da lista, de modo que nenhum item parcial dela fique visível. Se você não configurar IntegralHeight como False, a caixa de lista nem sempre preencherá toda a altura do formulário, o que não dá uma aparência muito adequada.

Adicione o Código

A finalidade deste aplicativo é exibir uma lista de autores, portanto, o código tem de estabelecer a conexão com o banco de dados, acessar essa lista e exibi-la no controle da caixa de lista. Todas essas etapas foram abordadas como parte do Dia 12, de modo que só mostrarei para você o código concluído (veja a Listagem 19.1), que é chamado no construtor do formulário. Adicione o código da linha 8 à sub-rotina New de seu formulário e, em seguida, insira a rotina LoadAuthors em qualquer local dele, exceto dentro de outro procedimento.

LISTAGEM 19.1 Adicionando o Código à Sub-rotina New de Seu Formulário

```
1 Public Sub New()  
2     MyBase.New()  
3  
4     'Esta chamada é requisitada pelo criador do formulário Windows.  
5     InitializeComponent()  
6  
7     'Adicione o que tiver de ser inicializado depois da chamada  
8     'InitializeComponent()  
9     LoadAuthors()  
10 End Sub
```


LISTAGEM 19.1 Adicionando o Código à Sub-rotina New de Seu Formulário
(continuação)

```
11 Private Sub LoadAuthors()  
12     Dim sConnectionString As String  
13     sConnectionString = "Data Source=(local);" & _  
14         "Initial Catalog=Pubs;" & _  
15         "User ID=sa;password=daneel"  
16     Dim connPubs As New SqlConnection(sConnectionString)  
17     Dim cmdAuthors As New SqlCommand(_  
18         "Select au_fname + ' ' + au_lname as Name from Authors", _  
19         connPubs)  
20     Dim daAuthors As New SqlDataAdapter(cmdAuthors)  
21     Dim dsAuthors As New DataSet("Authors")  
22  
23     connPubs.Open()  
24     daAuthors.Fill(dsAuthors, "Authors")  
25  
26     lbAuthors.DataSource = dsAuthors.Tables(0).DefaultView  
27     lbAuthors.DisplayMember = "Name "  
28 End Sub
```

Preciso desviar um pouco do tópico e discutir algo comum quando chegamos aos estágios finais de um aplicativo. Há um item no código mostrado na Listagem 19.1 que causará problemas quando esse for implantado. Ele usa (local) para especificar o nome do SQL Server (Data Source na string de conexão da Listagem 19.1). Isso informa à plataforma .NET que você quer se conectar com o SQL Server na máquina local, o que provavelmente é correto quando estamos desenvolvendo, mas quando esse código estiver em execução em um grupo de máquinas clientes diferentes, pode ser melhor que todas acessem o mesmo SQL Server central. Há duas maneiras de corrigir isso: a rápida que embutirá no código o nome do SQL Server desejado, ou a um pouco mais complicada que permitirá que o nome do servidor seja alterado sem a recompilação do código. É claro que quero mostrar as duas, portanto, começarei com o método mais fácil, em que apenas alteraremos a string da conexão para que inclua o nome real do computador de seu SQL Server:

```
sConnectionString = "Data Source=Olivaw;" & _  
    "Initial Catalog=Pubs;" & _  
    "User ID=sa;password=daneel"
```

Nada mais precisa ser alterado, e agora, independentemente de em que máquina esse código seja executado, ele tentará localizar o servidor com base no nome "Olivaw".

É possível que o nome desse servidor seja alterado ou você pode preferir usar um servidor diferente para este aplicativo. Em geral, o servidor que é empregado no desenvolvimento e o utiliza-

do na produção são duas máquinas diferentes. Seria benéfico poder alterar o servidor com o qual esse aplicativo se conecta sem que fosse preciso recompilar o código. Antes da plataforma .NET, esse tipo de funcionalidade era obtida com o uso do Registro. Essa opção ainda está disponível, mas um dos principais conceitos novos do VS .NET é que seu aplicativo deve poder ser instalado apenas por meio da cópia de seu diretório em uma máquina (que tenha o .NET Framework). Possuir informações que precisam estar disponíveis no Registro pode impedir que esse tipo de instalação funcione corretamente.

Perseguindo o objetivo de fornecer instalações e reinstalações fáceis para os aplicativos, a plataforma .NET introduziu um novo sistema para armazenamento das configurações de seu aplicativo, que é mais parecido com a tecnologia antiga dos arquivos .ini do que com a do Registro. Esse novo sistema funciona com o uso de arquivos de configuração, documentos XML que podem ser associados a toda uma máquina ou a aplicativos individuais. Depois que você concluir essa instalação simples, abordarei a utilização dos arquivos de configuração na próxima seção.

Teste o Projeto

Agora voltemos ao objetivo original de criar um arquivo de instalação. Edite a string de conexão da maneira certa para certificar-se de que ela poderá estabelecer corretamente uma conexão com um SQL Server e encontrar o banco de dados Pubs do exemplo. Concluída essa etapa, execute o projeto para ter certeza de que está funcionando e de que você possa ver uma lista de nomes de autores no controle de caixa de lista. Se tudo der certo e você tiver fornecido algo diferente de (local) como nome do servidor em sua string de conexão, então, teremos um aplicativo concluído para o qual poderemos desenvolver um programa de instalação.

Adicionando o Projeto de Instalação

No VS.NET, adicione os projetos de instalação à solução que possua um ou mais projetos que você queira implantar. Nesse caso, só temos um projeto que produzirá um arquivo .exe, e o resultado final precisa ser a instalação desse arquivo na máquina de destino. A fim de desenvolver uma instalação para esse aplicativo, adicione um novo projeto à solução atual (selecione File, Add Project e New Project no menu) e selecione o Setup Wizard (veja a Figura 19.1) porque esse é um bom ponto de partida.

Quando você adicionar esse projeto, ele iniciará automaticamente o assistente (veja a Figura 19.2).

Esse assistente tem conhecimento de que faz parte de uma solução e suas opções são baseadas nos projetos disponíveis nela. A configuração-padrão para o tipo de instalador (veja a Figura 19.3) é criar uma instalação para um aplicativo Windows, e é isso que você quer usar neste exemplo. Na próxima tela (veja a Figura 19.4) serão exibidas as opções as quais o seu programa de instalação irá implantar e aquelas que refletem as várias partes disponíveis de outros projetos dessa solução. Neste exemplo, queremos implantar a opção de saída (resultado) de nosso aplicativo Windows simples, que é um arquivo .exe neste caso, mas também poderíamos usar o insta-

lador para distribuir nosso código-fonte, arquivos de depuração ou outro conteúdo. Para este exemplo, selecione Primary Output from AuthorList.

FIGURA 19.1

Você pode usar o Setup Wizard ou simplesmente selecionar Setup Project para adicionar individualmente todos os arquivos de que precisa.



FIGURA 19.2

O Setup Wizard o conduzirá pela criação de um projeto simples de instalação para sua solução.



FIGURA 19.3

Se você quiser criar uma instalação gráfica em uma máquina Windows, selecione a instalação-padrão para um aplicativo Windows.

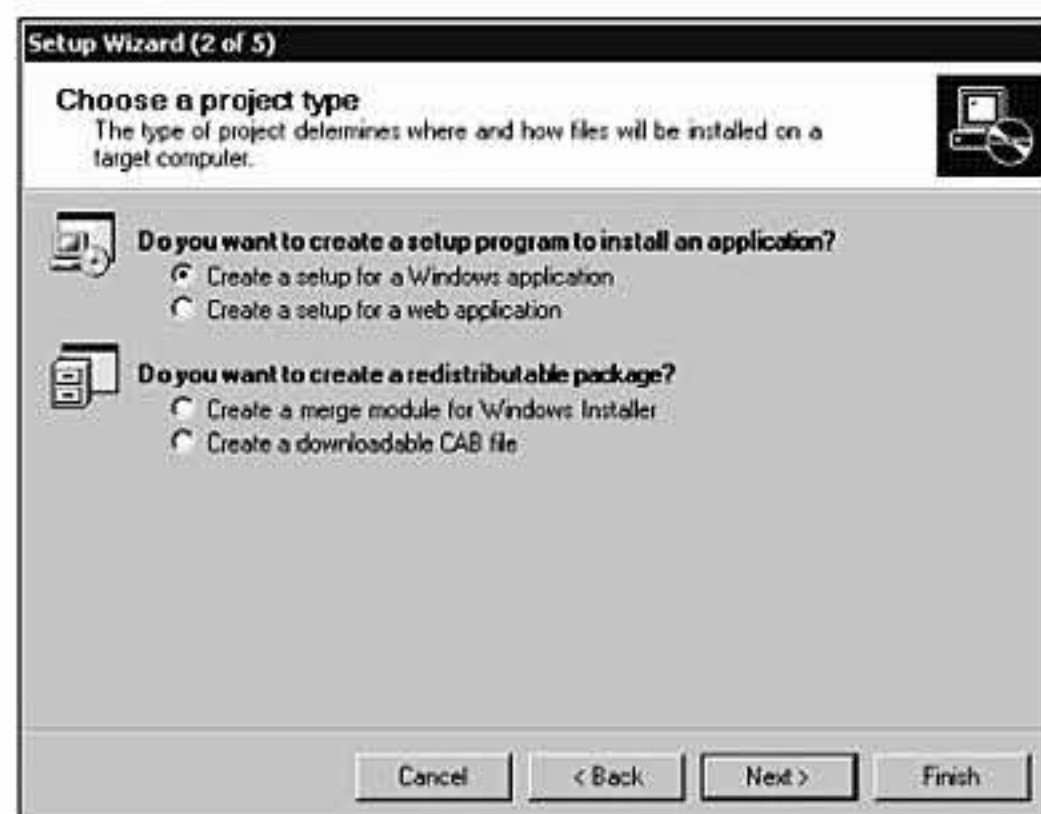
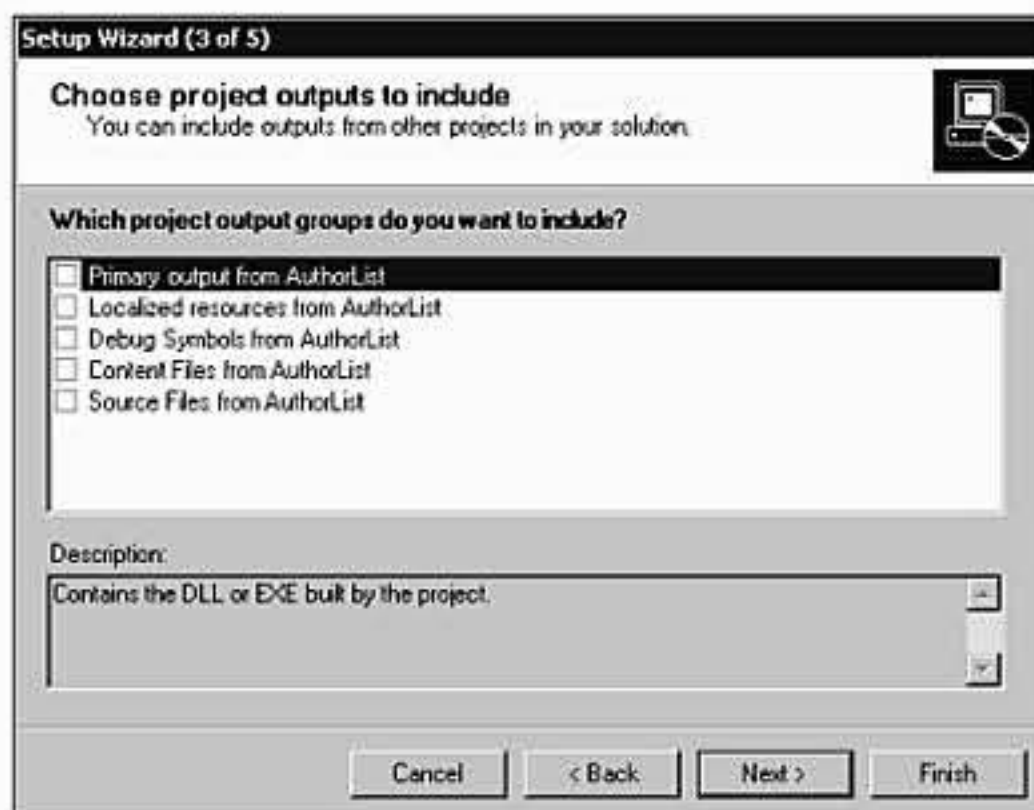


FIGURA 19.4

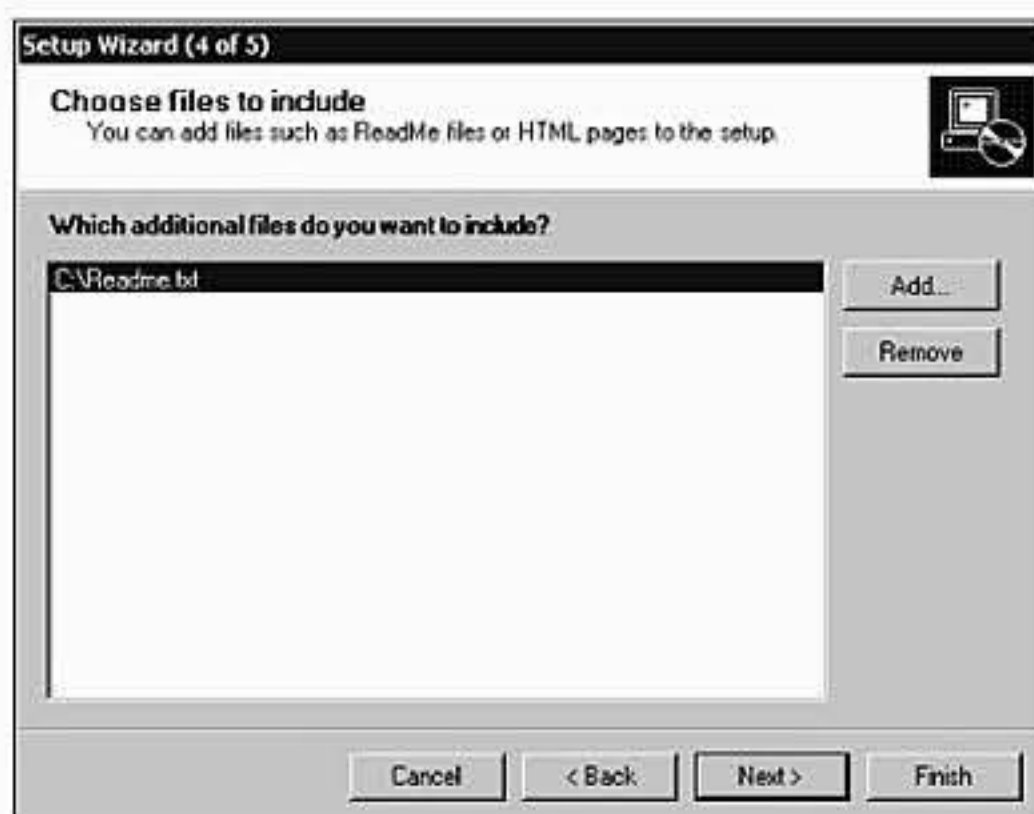
Os resultados de um projeto são consequência de seu desenvolvimento, seja um arquivo .exe ou .dll.



Além dos resultados de um ou mais de nossos projetos (se houvesse mais de um projeto na solução, que não fosse o de instalação, então, todos eles teriam sido incluídos na lista de saídas), você também pode selecionar qualquer arquivo adicional que queira incluir (veja a Figura 19.5). Em meu exemplo, incluí um arquivo readme, mas esse poderia ser seu método de distribuir quase todos os tipos de arquivos inclusive algum conteúdo XML ou até um banco de dados do Access (arquivo .mdb).

FIGURA 19.5

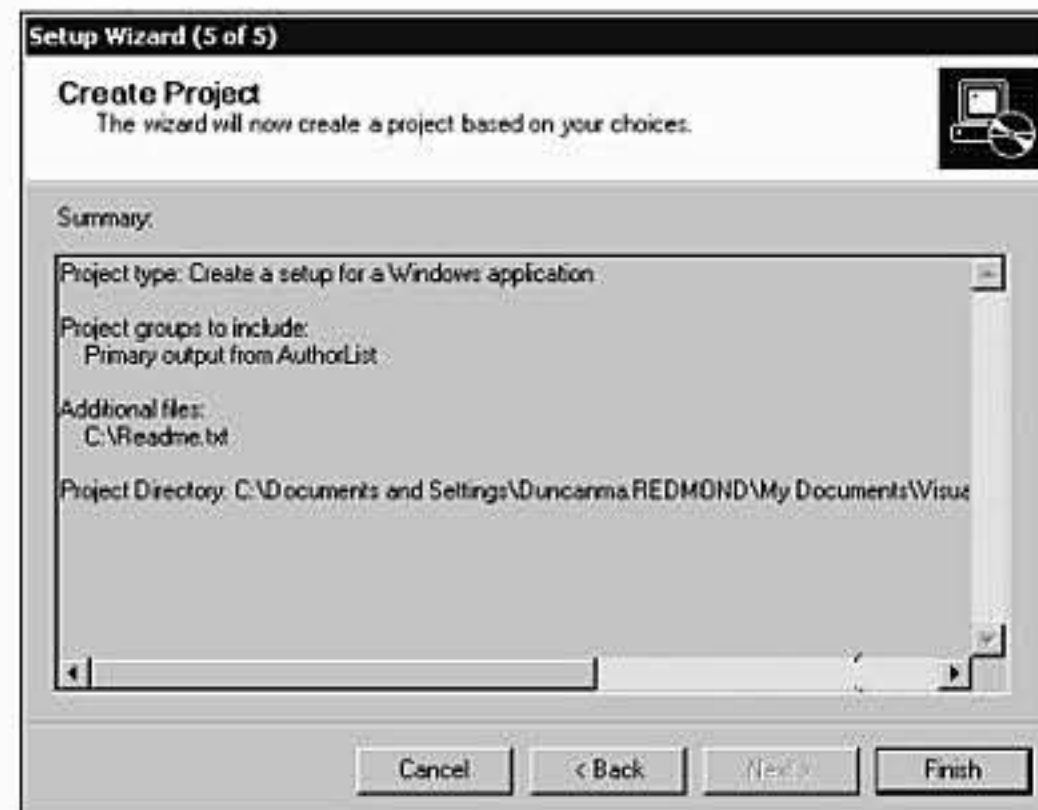
Adicionar arquivos .config, readme e documentação são apenas algumas das utilidades da seção de arquivos adicionais do projeto de instalação.



Para concluir, um resumo (veja a Figura 19.6) é exibido, e o assistente será fechado quando você der um clique em Finish. Nesse ponto, o projeto de instalação terá de se esforçar um pouco para determinar as dependências das saídas selecionadas, examinando os arquivos do projeto correspondente. Quando essa operação estiver concluída, será possível ver o projeto de instalação em seu Solution Explorer, e dentro da subpasta Dependencies, as dependências que encontrou. Na janela de propriedades do próprio projeto e de seus vários arquivos, pode-se alterar como o programa de instalação será criado, mas por enquanto, deixarei tudo como está.

FIGURA 19.6

A janela do resumo fornecerá algumas informações básicas sobre o que o projeto de instalação criará, mas o trabalho efetivo não começará até que você dê um clique em Finish.



Compilando a Instalação

Exatamente como em qualquer outro projeto, para gerar a saída deste projeto de instalação, você precisa compilá-lo. Pressione Ctrl+Shift+B ou selecione Build e Build Solution no menu para compilar todos os projetos da solução, inclusive o novo projeto de instalação. Essa compilação leva algum tempo devido basicamente ao projeto de instalação, que compacta todos os arquivos .NET redistribuíveis necessários em um único arquivo .msi. Vários outros arquivos serão compilados e incluídos no diretório da saída de seu projeto de instalação mas, na verdade, eles são apenas arquivos de suporte para a tecnologia do Windows Installer.

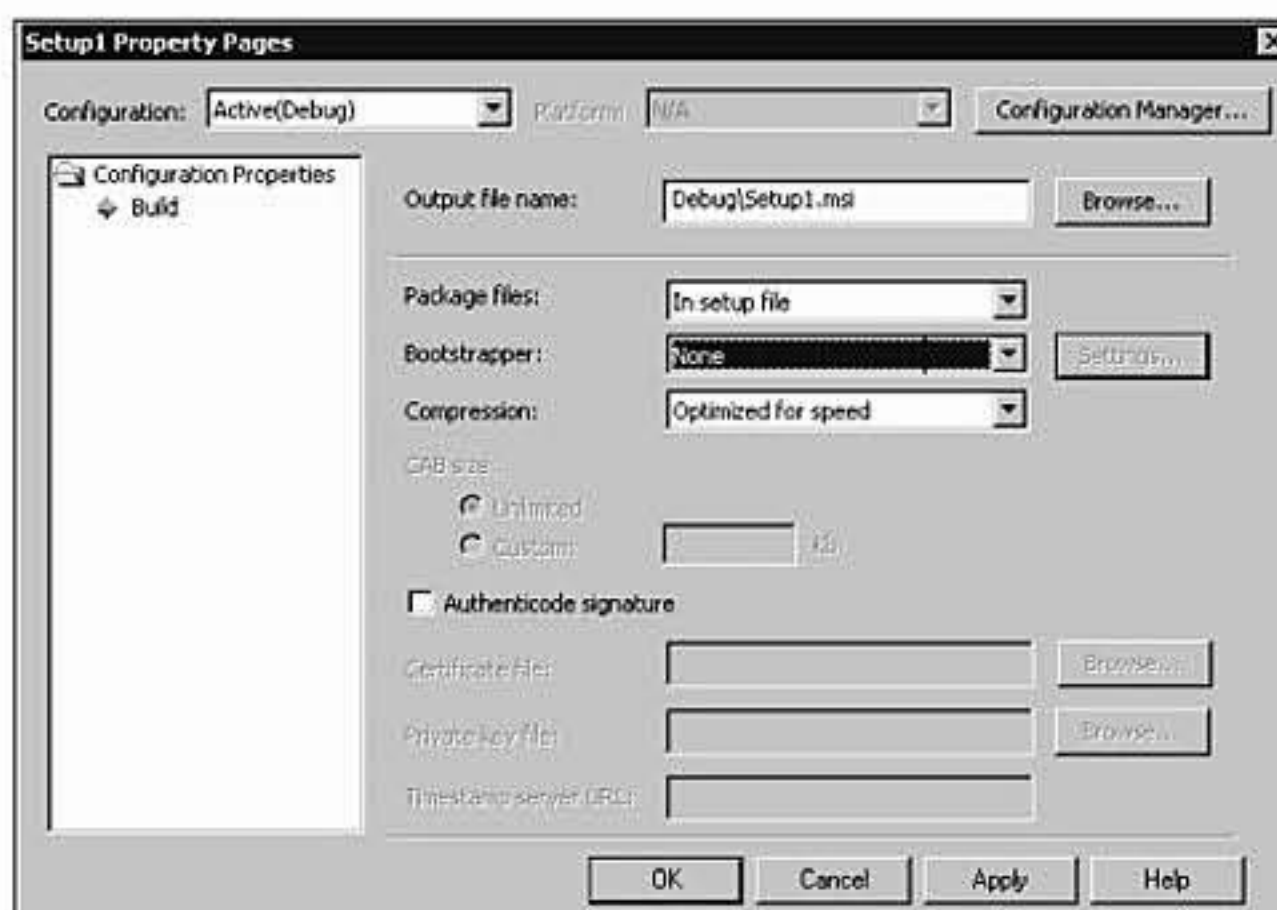
O Visual Studio .NET usa a versão 2.0 do Windows Installer, que pode ser a instalada no momento em sua máquina de destino. Se ela já não estiver na máquina, esses arquivos de suporte permitirão que o usuário a instale. Se você puder se certificar de que todos os usuários possuem a versão correta do Windows Installer, devido a sua versão do sistema operacional, por exemplo, ou porque têm outro programa .NET instalado, então poderá optar por incluir apenas o arquivo .msi em sua implantação. As configurações que controlam a inclusão dessas instalações podem ser encontradas quando você verificar se seu projeto de instalação está selecionado e, em seguida, selecionar Properties no menu Project (veja a Figura 19.7).



O termo *máquina de destino* em geral se refere à máquina que executará o aplicativo e que precisará de seus arquivos instalados nela. O alvo fica bem claro quando falamos em aplicativos Windows – as máquinas de todos os usuários –, mas lembre-se de que para os aplicativos da Web (formulários da Web, por exemplo), a máquina de destino é o servidor Web, e não a do usuário final.

FIGURA 19.7

Se você não estiver certo de que versão (se houver alguma) do Windows Installer estará disponível na máquina de destino, certifique-se de incluir os arquivos necessários para atualizar ou instalar a versão apropriada.



Na pasta de dependências de seu projeto de instalação, você verá vários arquivos, todos eles necessários para que seu aplicativo seja executado. Agora, será possível saber que não é preciso instalar um certo arquivo ou arquivos, mas o programa de instalação os incluirá independentemente disso. Se quiser forçar a exclusão de um arquivo, dê um clique nele com o botão direito do mouse e selecione **Exclude** no menu que aparecerá. Também podemos visualizar as propriedades de um arquivo e alterar a propriedade **Exclude** para **True** se quisermos: as duas operações terão o mesmo efeito. Uma razão comum para a exclusão de arquivos é que sabemos que o .NET Framework já está instalado e queremos tentar reduzir o tamanho da instalação para que fique adequada.

Isso conclui a criação de um programa de instalação simples, mas o teste real é pegar aquele arquivo .msi (que, no momento em que escrevi este texto, estava compactado em cerca de 25 MB) e implantá-lo em uma máquina de destino. Apenas copie o arquivo (que estará no diretório de depuração da pasta do projeto de instalação) para o destino ou insira-o em um CD. Em seguida, na máquina de destino, dê um clique com o botão direito do mouse no arquivo .msi e selecione **Install**. Nesse momento, você receberá uma mensagem de erro se não tiver a versão mais recente do Windows Installer na máquina de destino, caso contrário, a instalação deve prosseguir e inserir o .NET Framework e seu arquivo .exe nessa máquina.

Você pode ter alguns problemas para encontrar o que foi instalado, mas os arquivos serão inseridos em `\Program Files\<Autor do projeto de instalação>\<Título da instalação>` na máquina de destino, que por padrão será `\Program Files\Microsoft\Setup1`. Antes de compilar um programa de instalação que atenda a seus propósitos, certifique-se de alterar as propriedades **Author** e **Setup Title**, que estão disponíveis na janela de propriedades do projeto de instalação.

Arquivos de Configuração

Os arquivos de configuração são documentos XML associados ao seu aplicativo .NET com base no nome e local. Eles usam o formato simples de nomeação com o nome completo dos aplicativos (AuthorList.exe, por exemplo) seguido de .config (produzindo AuthorList.exe.config) e fi-

cam localizados no mesmo diretório dos aplicativos propriamente ditos. O conteúdo do documento é de XML padrão, mas o formato real (ou *esquema*, usando os termos da XML) é definido pela plataforma .NET. A procura por 'esquema de arquivos de configuração' na documentação do .NET Framework gera vários resultados úteis que detalham como criar arquivos de configuração.

Uma introdução básica seria que o arquivo é composto de uma ou mais seções, cada uma com seu próprio nome e contendo configurações relacionadas. Cada seção de configuração deve ser descrita como uma linha dentro de um conjunto de tags `<configSections></configSections>`. No entanto, também existem arquivos de configuração no nível da máquina, de modo que as seções descritas nesse nível podem ser usadas (sem serem descritas novamente) nos arquivos de configuração específicos do aplicativo. Uma das seções principais, `AppSettings`, é pré-descrita no arquivo de configuração da máquina, portanto você pode apenas seguir em frente e usá-la na configuração de seu aplicativo. Construí um arquivo de configuração para o aplicativo Author List, que mostro a seguir, como um exemplo de como poderíamos empregar esse tipo de arquivo. Ele contém, na seção `AppSettings`, apenas uma linha com o nome "Server" e o valor "Olivaw", representando o nome de meu SQL Server:

```
<configuration>
  <appSettings>
    <add key="Server" value="Olivaw" />
  </appSettings>
</configuration>
```

Agora, voltando a meu código, preciso fazer alterações para que o nome do servidor embutido no código da string de conexão seja, de modo alternativo, carregado a partir do arquivo de configuração. A versão alterada de `LoadAuthors`, que foi reescrita para usar o arquivo de configuração a fim de determinar o nome do SQL Server, é fornecida na Listagem 19.2.

LISTAGEM 19.2 Usando um Arquivo de Configuração para Tornar Seu Código Mais Flexível

```
1 Private Sub LoadAuthors()
2     Dim sConnectionString As String
3     sConnectionString = "Data Source=" & GetServerName() & ";" & _
4         "Initial Catalog=Pubs;" & _
5         "User ID=sa;password=danee1"
6     Dim connPubs As New SqlConnection(sConnectionString)
7     Dim cmdAuthors As New SqlCommand(_
8         "Select au_fname + ' ' + au_lname as Name from Authors ", _
9         connPubs)
10    Dim daAuthors As New SqlDataAdapter(cmdAuthors)
11    Dim dsAuthors As New DataSet("Authors")
12
13    connPubs.Open()
14    daAuthors.Fill(dsAuthors, "Authors")
```

LISTAGEM 19.2 Usando um Arquivo de Configuração para Tornar Seu Código Mais Flexível (*continuação*)

```
15
16     lbAuthors.DataSource = dsAuthors.Tables(0).DefaultView
17     lbAuthors.DisplayMember = "Name"
18 End Sub
19
20 Private Function GetServerName() As String
21     Dim sServerName As String
22     sServerName = _
23         Configuration.ConfigurationSettings.AppSettings.Get("Server")
24     Return sServerName
25 End Function
```

ANÁLISE

A linha 22 executa todo o trabalho que usa o arquivo de configuração, trabalho esse que ficou muito mais simples porque meu arquivo de configuração só utilizou seções já existentes. Se eu quisesse incluir minhas próprias seções, teria de descrever a nova seção como uma linha dentro do bloco `configSections` dos arquivos de configuração do aplicativo ou máquina e, em seguida, adicionar as configurações individuais como linhas dentro dessa nova seção. Nesse caso, a linha 22 retorna como uma string qualquer que seja o valor do atributo na linha do arquivo de configuração, de modo que posso empregar essa string na criação de minha conexão (linha 3).

Implantações de Múltiplos Projetos

O aplicativo de exemplo criado nesta lição se encontrava completamente dentro de um único projeto, mas é comum o desenvolvimento de sistemas em que alguns recursos do aplicativo são colocados em um ou mais componentes diferentes. Quando você desenvolver esses sistemas no Visual Studio .NET, em geral terá todos os projetos envolvidos (pelo menos os que estiverem sob seu controle ou de sua equipe) como parte da mesma solução. Em uma situação dessas, as opções apresentadas pelo Setup Wizard serão diferentes porque ele não consegue definir exatamente que projeto é o aplicativo ‘principal’.

Para ilustrar essa questão, dividirei aquele aplicativo individual em dois projetos, removendo o código de acesso a dados do primeiro e inserindo-o em uma biblioteca de códigos separada. O código, como está agora, é mostrado na Listagem 19.2 e se encontra encapsulado na sub-rotina `LoadAuthors`. Adicione um novo projeto à solução atual, uma biblioteca de classes do Visual Basic, e dê a ele o nome `PubsDataAccess`. Esse novo projeto terá um único arquivo de classes, que no início estará basicamente vazio e será chamado de `Class1.vb`. Renomeie o arquivo e a própria classe para `GetAuthors` (`GetAuthors.vb` para o nome do arquivo) e, em seguida, substitua seu conteúdo pelo código da Listagem 19.3.

LISTAGEM 19.3 Usando uma Classe Separada para Isolar Seu Código de Acesso a Dados

```
1 Option Explicit On
2 Option Strict On
3
4 Imports System
5 Imports System.Data
6 Imports System.Data.SqlClient
7
8 Public Class GetAuthors
9
10     Public Function AuthorList(ByVal sServerName As String) As DataSet
11         Dim sConnectionString As String
12         sConnectionString = "Data Source=" & sServerName & ";" & _
13             "Initial Catalog=Pubs;" & _
14             "User ID=sa;password=daneel"
15         Dim connPubs As New SqlConnection(sConnectionString)
16         Dim cmdAuthors As New SqlCommand(_
17             "Select au_fname + ' ' + au_lname as Name from Authors", _
18             connPubs)
19         Dim daAuthors As New SqlDataAdapter(cmdAuthors)
20         Dim dsAuthors As New DataSet("Authors")
21
22         connPubs.Open()
23         daAuthors.Fill(dsAuthors, "Authors")
24
25         Return dsAuthors
26
27     End Function
28 End Class
```

ANÁLISE

Observe que o único método dessa classe, `AuthorList` (linha 10), usa o nome de um servidor como parâmetro, permitindo que passemos essa informação do aplicativo principal. Agora o aplicativo original precisa ser reescrito para usar a biblioteca nova, o que também requer que adicionemos uma referência ao projeto. Para adicionar a referência, dê um clique com o botão direito do mouse na pasta `References` (do projeto original) e selecione `Add Reference`. Isso abrirá a caixa de diálogo `Add Reference`, e nesse momento você poderá encontrar o novo projeto de biblioteca de classes na guia `Projects` (veja a Figura 19.8). Certifique-se de que o projeto `PubsDataAccess` esteja realçado e, em seguida, dê um clique em `OK` para adicionar uma referência ao projeto principal.

Com essa referência adicionada, você poderá reescrever o código de seu formulário para que use a biblioteca, criando o código descrito na Listagem 19.4.

LISTAGEM 19.4 Adicionando o Código para Carregar os Valores em Seu Aplicativo

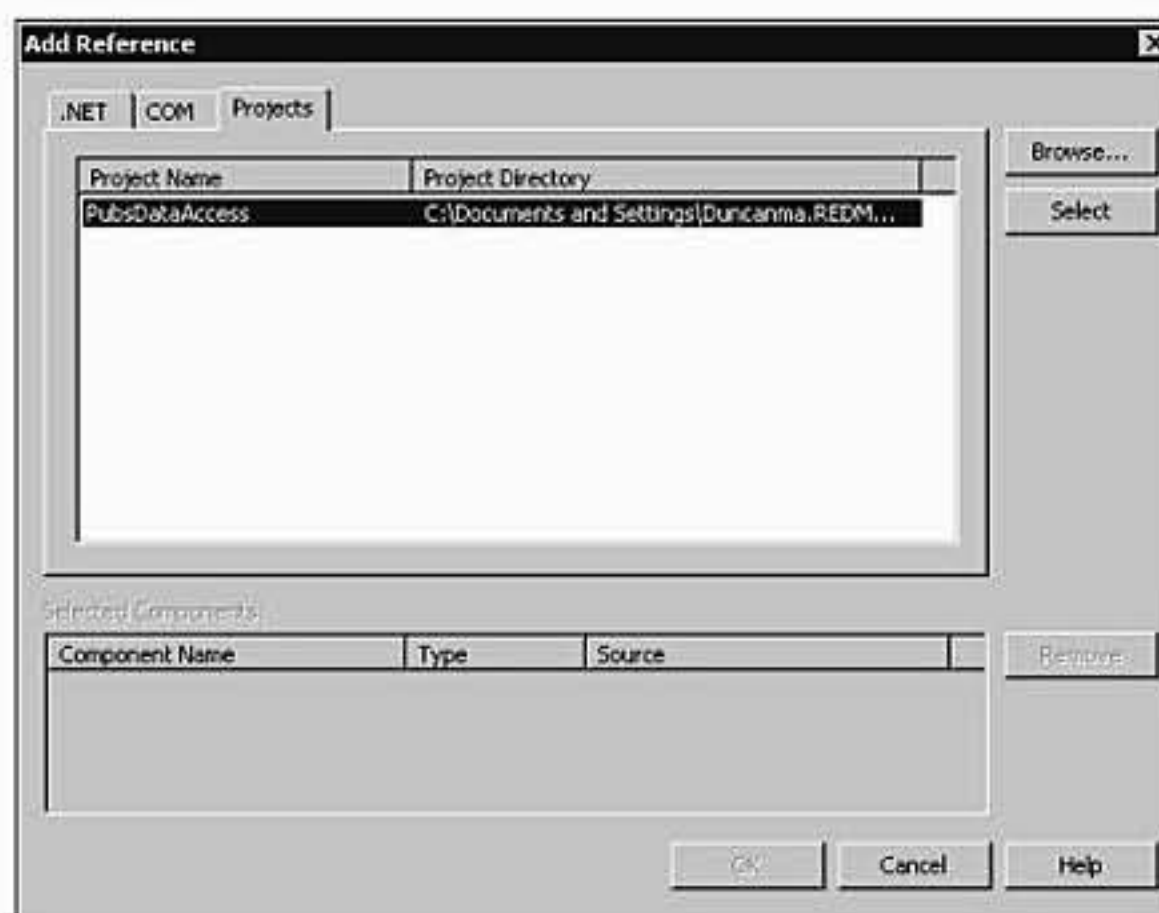
```

1 Dim sServerName As String
2 Private Sub LoadAuthors()
3     Dim dsAuthors As DataSet
4     Dim objDA As New PubsDataAccess.GetAuthors()
5
6     dsAuthors = objDA.AuthorList(sServerName)
7
8     lbAuthors.DataSource = dsAuthors.Tables(0).DefaultView
9     lbAuthors.DisplayMember = "Name"
10:End Sub
11
12 Private Sub frmDisplayAuthors_Load _
13 (ByVal sender As System.Object, _
14  ByVal e As System.EventArgs) _
15 Handles MyBase.Load
16     sServerName = _
17     Configuration.ConfigurationSettings.AppSettings.Get("Server")
18 End Sub

```

FIGURA 19.8

A guia Projects permite que você referencie diretamente outros projetos de sua solução, ou de outro local, mesmo se seus resultados não tiverem sido compilados.

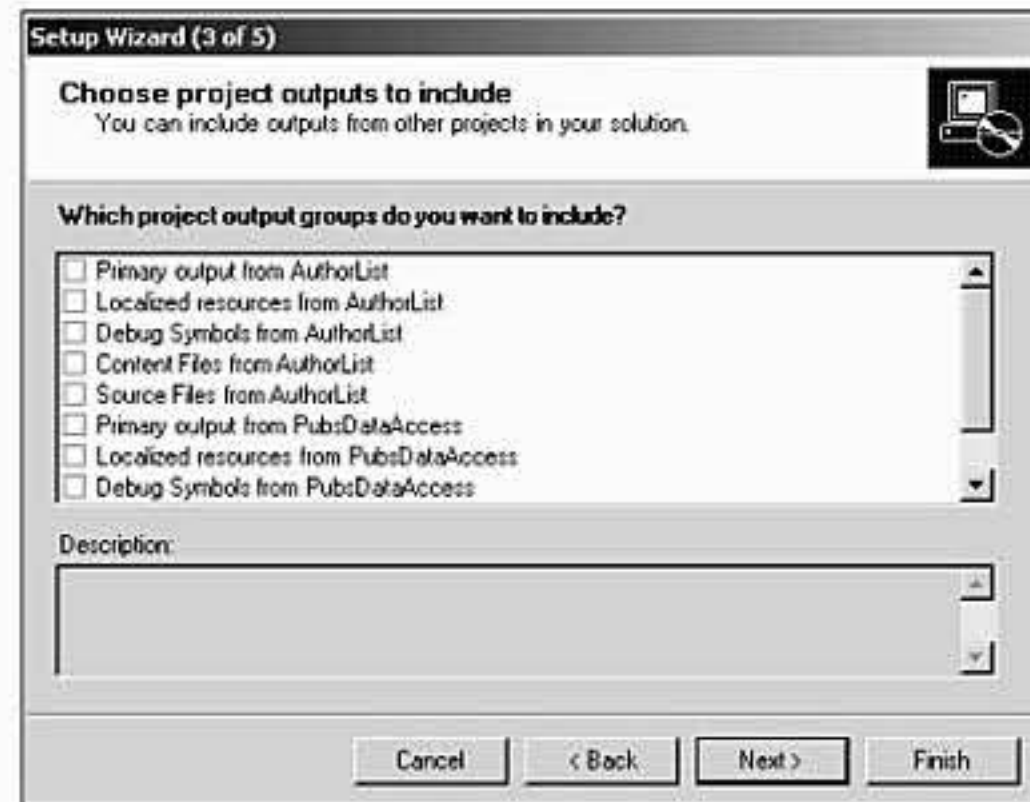


Para que possamos adicionar outro projeto de instalação, dê um clique no existente e selecione Remove para retirá-lo da solução atual. Dê um clique com o botão direito do mouse na solução, selecione Add e New Project no menu suspenso que aparecerá e abra a caixa de diálogo New Project. Selecione o Setup Wizard para obter um tipo de projeto, insira um nome diferente do padrão para evitar o conflito com o projeto criado anteriormente e, em seguida, dê um clique em OK. O mesmo Setup Wizard que já usamos deve aparecer, mas uma das etapas, Choose Project Outputs to Include, exibirá algumas informações diferentes das da última vez que esse assistente

foi executado. Dessa vez, os arquivos de resultado tanto do aplicativo principal quanto da nova biblioteca serão exibidos (veja a Figura 19.9).

FIGURA 19.9

Quando desenvolvemos uma instalação em uma solução com mais de um projeto (excluindo a própria instalação), todas as saídas possíveis de todos os projetos ficam disponíveis para o programa de instalação.



Apesar das diferenças e de todas as opções que ficaram visíveis na etapa Choose Project Outputs, você só deve selecionar o resultado básico do aplicativo principal. Nesse caso, ele é AuthorList, o projeto que contém apenas um formulário Windows, portanto é preciso selecionar apenas as saídas principais desse projeto. Depois que as saídas desejadas forem selecionadas, feche o assistente. Em seguida, o Setup Wizard determinará as dependências da(s) saída(s) selecionada(s) e, em nosso exemplo, encontrará uma dependência adicional que é a DLL PubsDataAccess. Por causa dessa dependência, a DLL necessária será instalada, mesmo não tendo sido selecionada como uma das saídas desejadas. Se fosse necessário selecionar as duas saídas, isso não faria com que a instalação falhasse, mas um aviso seria exibido durante a compilação de seu arquivo de instalação: **Aviso:** Dois ou mais objetos possuem o mesmo local de destino ('[targetdir]\pubsdataaccess.dll').

A instalação desse projeto não parecerá diferente para o usuário final, mesmo sendo instalada uma biblioteca adicional, e tudo irá para a mesma máquina de destino.

Resumo

Implantar sistemas é uma das etapas finais da criação de um aplicativo e é em geral muito mais complicado do que os desenvolvedores esperam. A plataforma .NET tornou a implantação muito mais simples, mas só desenvolvendo seu aplicativo, sua instalação e, em seguida, testando a implantação em quantas máquinas for possível você terá alguma certeza de que seu estágio real de implantação foi bem-sucedido.

P&R

P A tecnologia Windows Installer parece ter muito mais opções do que é possível encontrar nos projetos Setup e Deployment. Como posso desenvolver uma instalação com recursos como os que vejo nas instalações comerciais?

R Os recursos de instalação fornecidos pelo VS .NET são excelentes, mas eles abrangem apenas as necessidades básicas de um programa de instalação. Embora haja mais recursos nos projetos de instalação do VS .NET do que mostrei nesta lição, para conseguir todos os recursos disponíveis na tecnologia Windows Installer, será preferível que você trabalhe com um programa de instalação de outros fornecedores como o Install Shield (<http://www.installshield.com>) ou o Wise Installer (<http://www.wisesolution.com>).

P **Achei que não iria ter de escrever programas de instalação com a plataforma .NET. Achei que pudesse apenas usar um comando XCOPY.**

R Isso é verdade, você não tem de criar um programa de instalação para seu aplicativo, apenas copiar seu diretório deve fornecer a funcionalidade de que precisa. Mas a instalação do .NET Framework é um pouco mais complexa e será necessária em muitas máquinas antes que seu aplicativo possa ser executado. Além disso, um arquivo .bat executando um comando XCOPY deve funcionar, mas uma instalação gráfica com caixas de diálogo e outras opções é uma escolha muito mais profissional para mostrar a seus usuários.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Se você não estiver certo de que o .NET Framework estará disponível em todas as suas máquinas de destino, terá de desenvolver duas instalações (uma com ele e outra sem)?
2. Se você criar um aplicativo composto de apenas um projeto principal e vários projetos de biblioteca que ele use, terá de gerar múltiplas instalações?
3. Como configurar uma única instalação para que tenha vários tipos, como, por exemplo, a normal, a completa e a mínima?
4. Como você incluiria um arquivo de configuração (.config) ou outros arquivos adicionais junto a sua instalação?

Exercícios

1. Desenvolva um aplicativo com formulários Windows que use um arquivo de configuração para determinar que nome (propriedade Text do formulário) inserir na barra de título do Windows e, em seguida, crie um arquivo .config para ele e um programa de instalação. Implante o projeto e, então, tente alterar o conteúdo do arquivo .config e executar novamente o aplicativo.

SEMANA 3

DIA 20

Introdução à XML

A menos que você tenha evitado, durante os últimos anos, qualquer pessoa no mesmo prédio que tenha olhado, tocado ou estado com um computador, pode ter ouvido falar do acrônimo de três letras XML. A XML (eXtensible Markup Language) está em todos os lugares atualmente – aparecendo até em revistas de variedades do grande mercado. Nesta lição tentarei desmistificar um pouco do sensacionalismo e examinar em que a XML pode auxiliar seus programas. Em particular, esta lição enfocará:

- O que é XML?
- A XML na prática.

O Que É XML?

A Extensible Markup Language (XML – se analisarmos deveria ser EML, mas essa abreviatura não chegaria nem perto do nome moderno que é dado atualmente) é uma maneira de adicionar informações ao texto. XML é o termo popular usado nos softwares – linhas de produtos e empresas inteiras foram criadas ou modificadas apenas para adicionar XML ao seu material de propaganda –, e há muitos mitos e mal-entendidos relacionados ao que ela pode fazer e qual sua finalidade.

No entanto, antes de tratarmos dos mal-entendidos, examinaremos primeiro o que ela faz. A XML é uma maneira de adicionar informações ao texto. Que tipo de informações? Informações sobre o texto. Pode soar como um círculo vicioso, mas estou falando sério, e isso na verdade tem um nome – metadado. O metadado é a informação sobre informações. A definição formal é

Dados sobre dados. O metadado descreve como, quando e por quem um conjunto específico de dados foi coletado e como eles estão formatados. O metadado é essencial para a compreensão das informações armazenadas em depósitos de dados (definição encontrada na Wikipédia do Internet.com).

Para apreendermos o conceito do metadado, examinaremos o livro que você está lendo neste momento. É claro que há uma parte essencial no livro – as informações que ele contém. Elas são os dados. Além delas, ainda há as informações sobre o livro – a quantidade de lições, a existência dos cabeçalhos nas seções e assim por diante. Até a formatação fornece informações sobre o livro. Se apliquei **negrito** a alguma parte do texto, você perceberá que ela é mais importante do que o restante. Essas são as informações sobre o texto do livro – o metadado. O metadado é uma idéia poderosa; fornece às informações uma estrutura e finalidade.

Você pode adicionar metadados de muitas maneiras. Por exemplo, em um banco de dados, o metadado se encontra na forma de nomes e dimensões referentes às colunas das tabelas. Ele informa que tipo de dados deve existir nesse local, qual seu tamanho e às vezes um nome que induz ao conteúdo. A HTML também fornece metadados de acordo com a importância relativa de alguns textos. Isto é, eles ficam em uma tag de cabeçalho e, nesse caso, indicam o nível. No entanto, a HTML não é um exemplo adequado porque foi projetada para ser usada na formatação, e não na identificação de informações.

Algo que a HTML realmente nos mostra, entretanto, é a relevância de usar simples tags para adicionar a formatação, ou seja, metadados, ao texto. As tags são os itens que possuem colchetes angulares (dessa <forma>) que vemos quando nos deparamos com a HTML. Se nunca tiver visto um código HTML ou só quiser ter uma idéia de sua aparência, dê uma olhada no código-fonte de qualquer página da Web. É possível fazer isso selecionando Exibir e Código fonte no Internet Explorer. Você deve ver várias tags. A HTML usa essas tags para identificar qual texto deve ficar em negrito ou em itálico, onde uma tabela ou figura deve ser inserida e assim por diante. Ela também demonstra o final de cada uma dessas seções. Portanto, se examinássemos o código-fonte da página da Web mostrada na Figura 20.1, veríamos

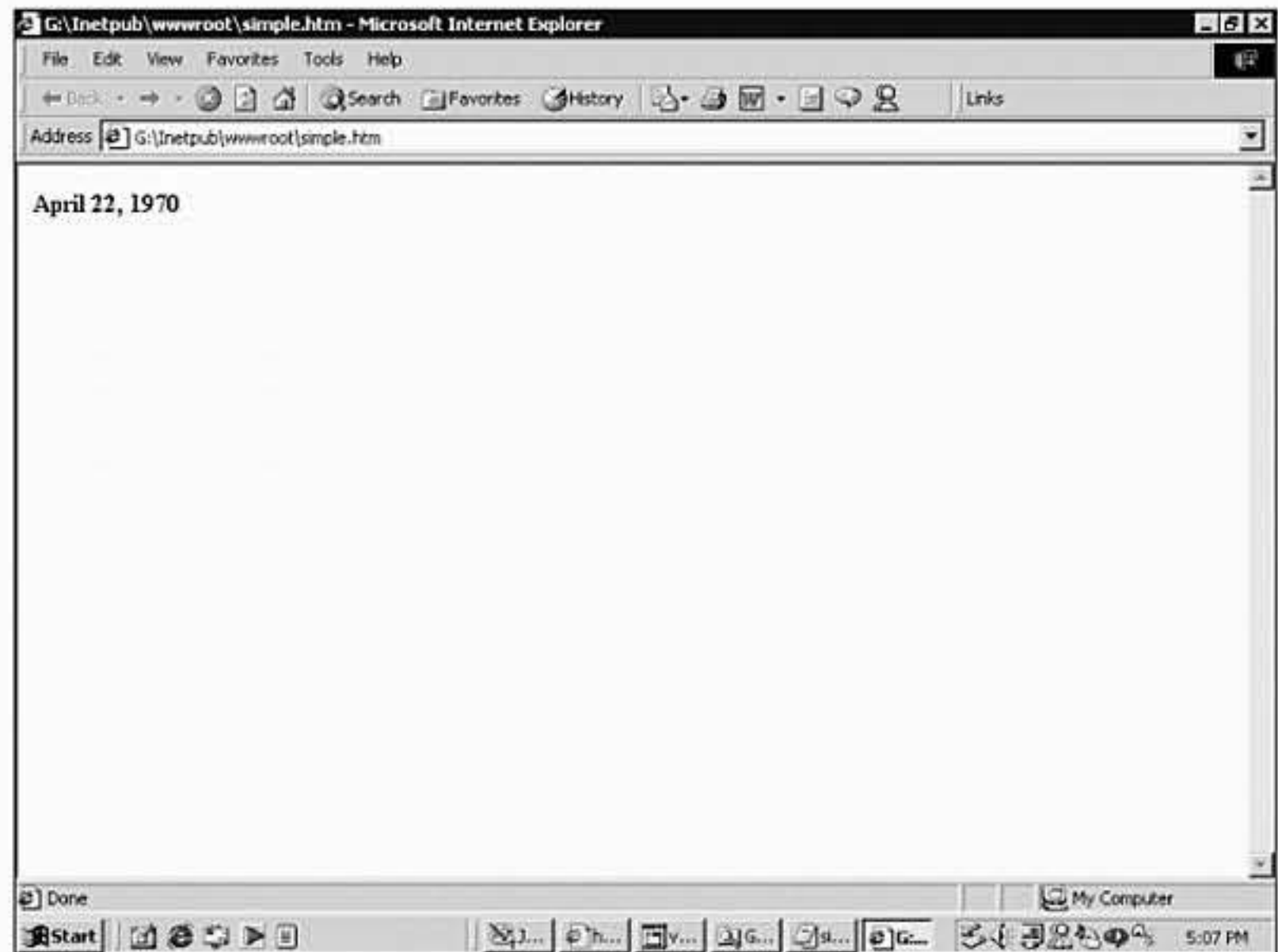
```
<html><body><strong>April 22, 1970</strong></body></html>
```

A tag marca tudo que estiver entre ela e a tag de fechamento como algo importante, e a maioria dos navegadores exibiria esse trecho em negrito. Essa tag adicionou informações ao texto, portanto é metadado.

A XML é semelhante a HTML; na verdade, elas estão relacionadas. Há muitos anos, um pouco depois da idade das trevas (nos anos 60), vários pesquisadores, preocupados com a quantidade de maneiras pelas quais um ponto importante poderia ser identificado, criaram algo chamado Standard Generalized Markup Language (SGML). A SGML introduziu muitas idéias inovadoras:

FIGURA 20.1

Uma data a ser lembrada.



- Devemos identificar o começo de uma seção de informações com um marcador.
- O marcador deve identificar a si mesmo como tal usando um conjunto de caracteres que raramente aparecem na escrita comum. Em seu caso, os autores da SGML escolheram os colchetes angulares ($\langle \rangle$).
- O marcador também deve ter algumas informações, não importando se forem resumidas, sobre os dados contidos.
- Devemos identificar o término das informações com outro marcador.
- O marcador relacionado ao significado acima deve identificar a si próprio como referente ao marcador anterior usando as informações já empregadas, adicionando ainda um caractere que não tenha probabilidades de aparecer nas informações de origem.

O resultado final pode se parecer muito com o descrito a seguir:

```
<strong>April 22, 1970</strong>
```

Como você já deve ter adivinhado, a HTML é descendente da SGML. É um dos muitos tipos de utilização especial da SGML. Veja bem, a SGML não era na verdade uma maneira de marcar documentos específicos, mas, em vez disso, o modo de definir os tipos de tags que poderiam ser adicionados a um documento e, portanto, o modo de identificar os metadados dele. A seguir, seria criada a definição de um documento (isto é, uma lista das tags que poderiam ser usadas) que começaria a ser empregada na criação desses documentos. Conseqüentemente, a HTML apresentava várias tags que poderiam ser utilizadas para a formatação, e que foram usadas pelas pessoas para criar todas as páginas excelentes que existem na World Wide Web (WWW).

A XML é parente da HTML – ou seja, as duas herdaram muitas das idéias da SGML. No entanto, a XML é mais parecida com a SGML do que a HTML. Em vez de definir um tipo específico de documento (como uma página da Web), ela estabelece uma maneira de identificar as partes dele

(como a SGML fazia). Ela usa os mesmos conceitos básicos da SGML; emprega tags para marcar o começo e o final de uma seção de informações. Essas tags são palavras entre colchetes angulares que devem fornecer informações sobre os dados que contêm. Em outras palavras, a XML tem a mesma aparência dos trechos anteriores de HTML e SGML.

Por que os autores da XML criaram algo tão parecido com os dois padrões já existentes? Eles fizeram isso porque a SGML é complexa, e a HTML, na verdade, só tem a finalidade de adicionar a um documento informações sobre a formatação. A XML foi projetada para ser uma SGML simplificada que poderia ser usada na criação de várias linguagens ou conjuntos de tags específicas de um segmento da indústria ou de uma tecnologia. Muitas dessas linguagens estão disponíveis, inclusive as que descrevem equações matemáticas (MathML), receitas e até uma versão da HTML que foi definida com o uso da XML (XHTML). Cada uma delas define os metadados que podem ser aplicados às informações; no entanto, todas fazem isso conforme as regras da XML.

Então, quais são as regras da XML? Elas foram oficialmente definidas da seguinte maneira:

- A XML deve ser simples de usar na Internet.
- Deve dar suporte a uma ampla variedade de aplicativos.
- Deve ser compatível com a SGML.
- Deve facilitar a criação de programas que processem documentos XML.
- A quantidade de recursos opcionais da XML deve ser mantida em um mínimo rigoroso, cujo ideal seria nenhum.
- Os documentos XML devem ser suficientemente claros e de fácil leitura pelas pessoas.
- O projeto XML deve ficar pronto rapidamente.
- O projeto que usar XML deve ser formal e conciso.
- Os documentos XML devem ser fáceis de criar.
- A abreviação na marcação XML tem uma importância mínima.

Portanto, isso é o que é a XML. E aqueles mal-entendidos mencionados anteriormente? Parece haver muitos deles. Vários surgiram pela maneira com que diversos departamentos de marketing promoveram o uso que sua empresa fazia da XML; outros foram causados pelos escritores que passaram para as pessoas idéias estranhas sobre ela, mas é claro que não sou um *deles*.

Os principais mal-entendidos que vejo com frequência são

- Você tem de escolher entre o Visual Basic .NET e a XML. Embora a XML seja uma linguagem, não é uma linguagem de programação como o Visual Basic .NET. As pessoas parecem ficar confusas com essa parte da ‘linguagem’ e começam a fazer perguntas como, “Devo aprender Visual Basic .NET ou XML?”. A XML não é uma linguagem de programação, só uma maneira de descrever informações.
- A XML é complicada e difícil de ler. Por que adicionar toda essa ‘confusão’ ao documento quando *X* (alguma outra técnica) faria o mesmo? A resposta é simples. É uma maneira fácil de identificar as partes de um documento. Por exemplo, muitas pessoas sugerem que algo chamado de *Valores separados por vírgula* (CSV – Comma-separated values) é me-

lhor do que a XML por ser mais simples, fácil de produzir e os arquivos serem menores (veja o próximo item). No entanto, qual dessas duas listagens descreve mais detalhes sobre as informações exibidas:

```
<funcionários>
  <identificação funcionário="1">
    <primeironome>John</primeironome>
    <sobrenome>Bull</sobrenome>
  </funcionário>
  <identificação funcionário="2">
    <primeironome>Mary</primeironome>
    <sobrenome>Tell</sobrenome>
  </funcionário>
</funcionários>
1, John, Bull
2, Mary, Tell
```

Na minha opinião, a primeira (a XML) é muito mais compreensível.

- A XML torna o trabalho lento (ou muito extenso). Essa afirmação em geral é proferida por aqueles veteranos que acham que se estivermos usando um caractere a mais, estaremos desperdiçando velocidade/esforço da memória/processador. Embora às vezes isso me preocupe, a XML não é uma das áreas onde ocorre. O argumento que você pode ouvir é algo como, “Todas essas tags de abertura e fechamento ocupam muito espaço e seu uso consome tempo. Devemos usar *X* (colocam algo aqui de que eles gostam mais) em vez disso”. A maioria das pessoas que estiver lendo este texto provavelmente se lembrará do resultado de um pensamento desse tipo: ele foi chamado de bug Y2K. Realmente há o que se chama (em minha opinião) de código superotimizado. Em algumas situações, usar uma técnica menos eficiente que torne mais fácil compreender a intenção produz uma resposta bem superior (mais uma vez, é apenas minha opinião). A XML é uma dessas técnicas menos eficientes, porém mais inteligíveis.
- A resposta é XML. Qual é a pergunta? A XML não vai resolver o problema da fome mundial, impedir a guerra ou mesmo fazer café para você de manhã. Tudo que pretende é gerar informações sobre um bloco de texto. Muitas pessoas tentam tornar a XML a solução para tudo ou aplicá-la onde não pode, ou deve, ser usada. Se parece que ela irá causar mais problemas do que se deseja resolver ou se não estiver nem mesmo resolvendo problema algum, não a utilize. Encontre uma solução melhor.

Elementos

Vários itens podem ser usados na criação de um documento XML; no entanto, você encontrará duas partes importantes em quase todo documento XML – elementos e atributos.

Um *elemento* é o bloco de construção fundamental de todo arquivo XML. Ele é composto de uma tag inicial, de uma tag de fechamento e do conteúdo entre elas. Como exemplo, há três elementos na XML a seguir:

```
<identificação funcionário="1">  
  <primeironome>John</primeironome>  
  <sobrenome>Bull</sobrenome>  
</funcionário>
```

Um elemento é funcionário. O elemento funcionário começa com a tag funcionário (<funcionário>), termina com a tag de fechamento funcionário (</funcionário>) e contém as duas outras tags, <primeironome> e <sobrenome>. O elemento primeiro nome começa com a tag <primeironome>, termina com a tag de fechamento </primeironome> e contém o texto John.

Os elementos definem dois ‘itens’ importantes do arquivo XML. É claro que a XML anterior descreve o funcionário de uma empresa e que ele tem um primeiro nome e um sobrenome. Para associar isso às lições dos capítulos passados – os elementos são semelhantes aos objetos e propriedades que examinamos no Dia 7, “Trabalhando com Objetos”. Esse exemplo poderia descrever um objeto funcionário que possui duas propriedades.

Atributos

Embora grande parte de qualquer arquivo XML seja de elementos, também podem existir atributos nele. Os atributos são usados para fornecer informações adicionais sobre um elemento.

Se você usou HTML no passado, provavelmente saberá o que é um atributo. Por exemplo, no fragmento HTML

```
<a href="http://msdn.microsoft.com/vbasic">Home Page do Visual Basic</a>
```

o termo href é um atributo que define um local para o elemento a (ou âncora). Ele estabelece para onde o elemento âncora deve direcionar seu navegador quando você der um clique nele. Para os que não conhecem HTML, a linha anterior de código adiciona um hiperlink (um daqueles links sublinhados) para uma página da Web.

De maneira semelhante, outros atributos fornecem informações para elementos diferentes. Isso leva a uma das maiores diferenças entre os atributos e elementos: os atributos não são independentes e precisam ser aplicados aos elementos. Um atributo sempre aparece dentro da tag de abertura de um elemento. Além disso, ele sempre apresenta a forma a seguir:

```
nome="valor"
```

Ou seja, um atributo é composto de duas partes: um nome e um valor. O nome deve ser exclusivo dentro de cada elemento, embora dois elementos possam ter o mesmo atributo. Além disso, um elemento pode ter muitos atributos.



A comunidade XML se divide em duas facções. De um lado estão as pessoas que sugerem que devemos usar os elementos em todos os locais. Todos os trechos de informações devem ser elementos contidos dentro de outros elementos. Essa lógica em geral pode ser resumida como "Ao mesmo tempo que pode se parecer com uma propriedade agora, posteriormente você pode precisar ter algo como uma propriedade dela". Por exemplo, considere que ainda precisemos dividir uma propriedade, que represente um endereço, em propriedades como rua, cidade e código postal. Essas pessoas que acham que 'não existe nada a não ser elementos' definiriam nosso exemplo do funcionário John Bull usando o trecho XML a seguir:

```
<funcionário>
  <identificação>1</identificação>
  <primeironome>John</primeironome>
  <sobrenome>Bull</sobrenome>
</funcionário>
```

Do outro lado estão aqueles que sugerem que se um trecho de informações for sobre outro elemento (como uma propriedade é para um elemento), ele deve ser um atributo. Isso torna os arquivos XML menores e em algumas situações mais compreensíveis. Essas pessoas definiriam nosso funcionário Johnny da maneira a seguir:

```
<identificação funcionário="1" primeironome="John" sobrenome="Bull" />
```

É claro que você é livre para usar qualquer uma das duas alternativas, ou até um híbrido delas, como seu estilo. No entanto, lembre-se de alguns pontos quando decidir se algo deve ser um elemento ou atributo:

- Um atributo não pode ter filhos. Isso significa que definir algo como um atributo é uma via de mão única. Você não pode resolver posteriormente adicionar novas informações sem alterar a estrutura de um arquivo. Os elementos, por outro lado, permitem que sejam adicionados em um momento posterior novos atributos ou elementos-filhos quando necessário.
- Um documento com muitos atributos em geral é menor do que um com vários elementos. Quanto mais atributos você tiver em seu código XML comparado com a quantidade de elementos, menor ele será. Isso acontece principalmente porque os atributos não possuem uma tag de fechamento.

Esquemas

Embora os esquemas na verdade não façam parte da XML, desempenham um papel importante em sua utilização, e isso se tornará mais relevante quando eles se tornarem um padrão oficial. No momento em que escrevo este texto, os esquemas XML acabaram de ser definidos como um padrão recomendado. Os esquemas XML são, em primeiro lugar, um aplicativo em XML. Exatamente como qualquer outro código XML, eles são informações sobre informações (metadados – já ficou cansado de ouvir este termo?). Os esquemas XML definem uma maneira apropriada de estruturar um arquivo XML. Isso inclui os tipos de dados para cada elemento e atributo, o que pode ser um elemento-filho de qualquer outro elemento, e até quais são os elementos válidos para um arquivo específico.

Alguns de vocês podem se perguntar por que os esquemas são necessários. Eu não disse que você poderia criar um arquivo XML contendo qualquer combinação de elementos e atributos? Bem, na verdade, sim. No entanto, há uma diferença entre criar um arquivo XML para conter al-

guns dados e um código XML que defina um tipo específico de dado. Por exemplo, se trabalhássemos com um arquivo que tivesse dados de funcionários, poderíamos esperar ver certos itens – nome, identificação do funcionário, algumas informações de contato e assim por diante. Entretanto, não temos idéia do local em que se encontram essas informações no arquivo, nem que tags podem ser chamadas. Sem um esquema, teríamos de examinar o arquivo e escrever um código específico para ler cada um deles. Com um esquema, conheceremos a estrutura dos dados antecipadamente. Nem todos os campos serão necessários ou que eles estejam exatamente no mesmo local em todos os arquivos, mas saberemos com certeza quais desses campos serão chamados e teremos informações suficientes para recuperá-los por meio do DOM ou usando um objeto `XMLTextReader`. Examinaremos esses objetos em breve. A Figura 20.2 mostra um esquema XML simples.

**NOTA**

Se você leu algo antes ou se decidir fazer uma leitura adicional sobre XML, se deparará com o conceito chamado Document Type Definition (DTD). Os DTDs foram uma tentativa anterior de definir uma estrutura apropriada para arquivos XML. Na verdade, os DTDs datam da época da SGML. Eles parecem executar a mesma tarefa que os esquemas XML; no entanto, há algumas diferenças importantes:

- Os esquemas XML são escritos com o uso da XML; os DTDs não. Eles empregam um formato semelhante, mas que não é a XML, para a definição da estrutura adequada de um arquivo XML. Isso é importante porque significa que você precisa de um conjunto de ferramentas para trabalhar com os DTDs, e outro para manipular a XML resultante. Já que os esquemas XML são XML, é possível trabalhar com eles usando as mesmas ferramentas.
- Os DTDs não definem os tipos de informações. Eles descrevem a estrutura delas (isto é, que elementos podem estar contidos em outro), mas não se um dado elemento deve ser um inteiro, string ou outro tipo de dado. Os esquemas XML fornecem a capacidade de adicionar essa informação.

Como já foi descrito muitas vezes neste livro, um dos principais objetivos do Visual Basic .NET é tornar mais fácil o trabalho com tecnologias complexas. Portanto, não deve surpreender o fato de ele possuir um editor que permita a visualização e edição de esquemas XML. A criação de esquemas também produz um efeito agradável que é tornar mais simples o trabalho com os documentos XML. A Figura 20.2 mostra um esquema XML básico. Ele se encontra em um editor gráfico que permite a geração de vários dos tipos e estruturas que compõem os esquemas XML. Depois de criar um esquema, você poderá associá-lo a um arquivo XML, e o Visual Basic .NET fornecerá os recursos de Intellisense para os elementos e atributos definidos nesse esquema (veja a Figura 20.3).

FIGURA 20.2
Criando um esquema
XML com o Visual
Basic .NET.

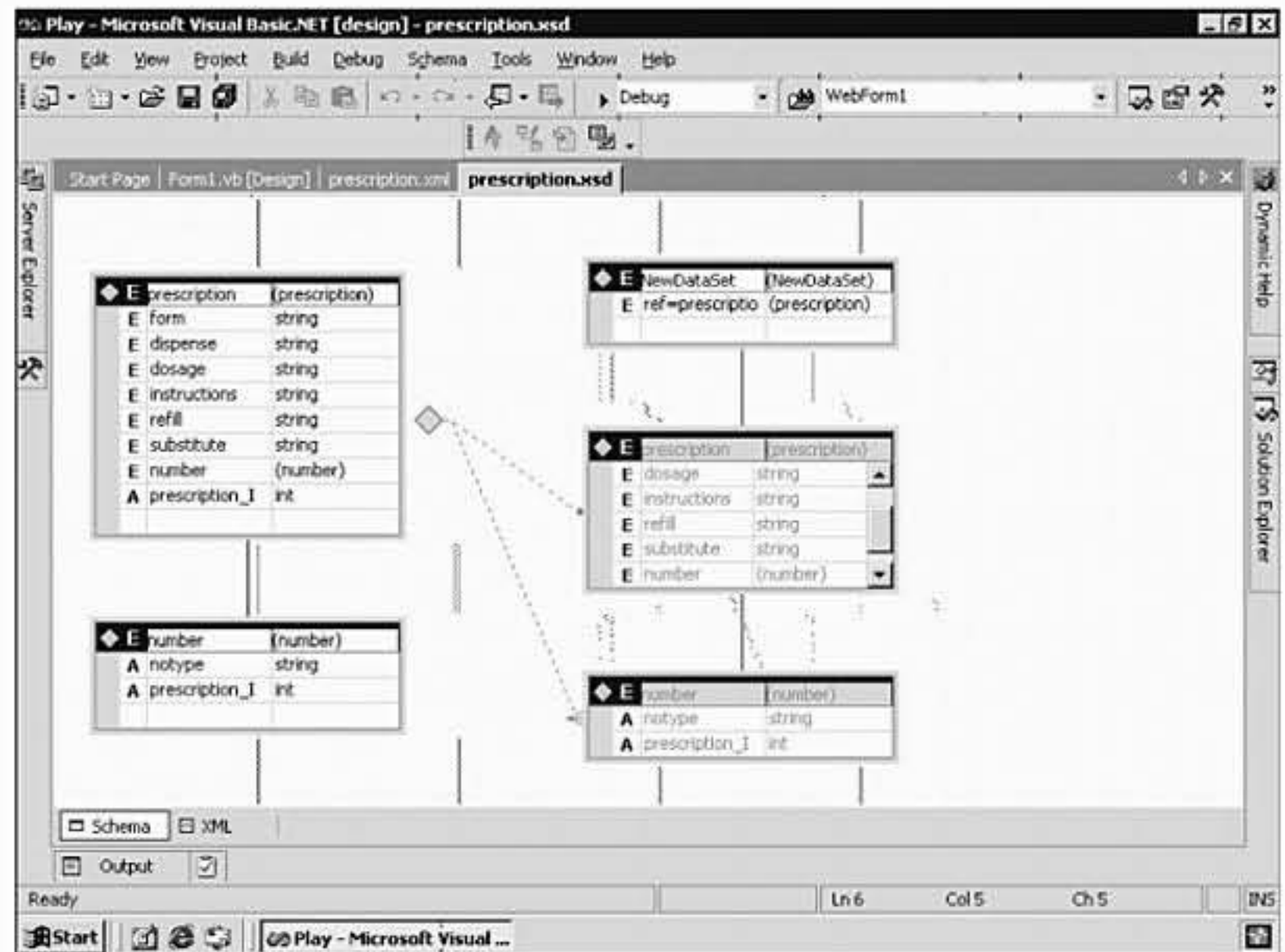
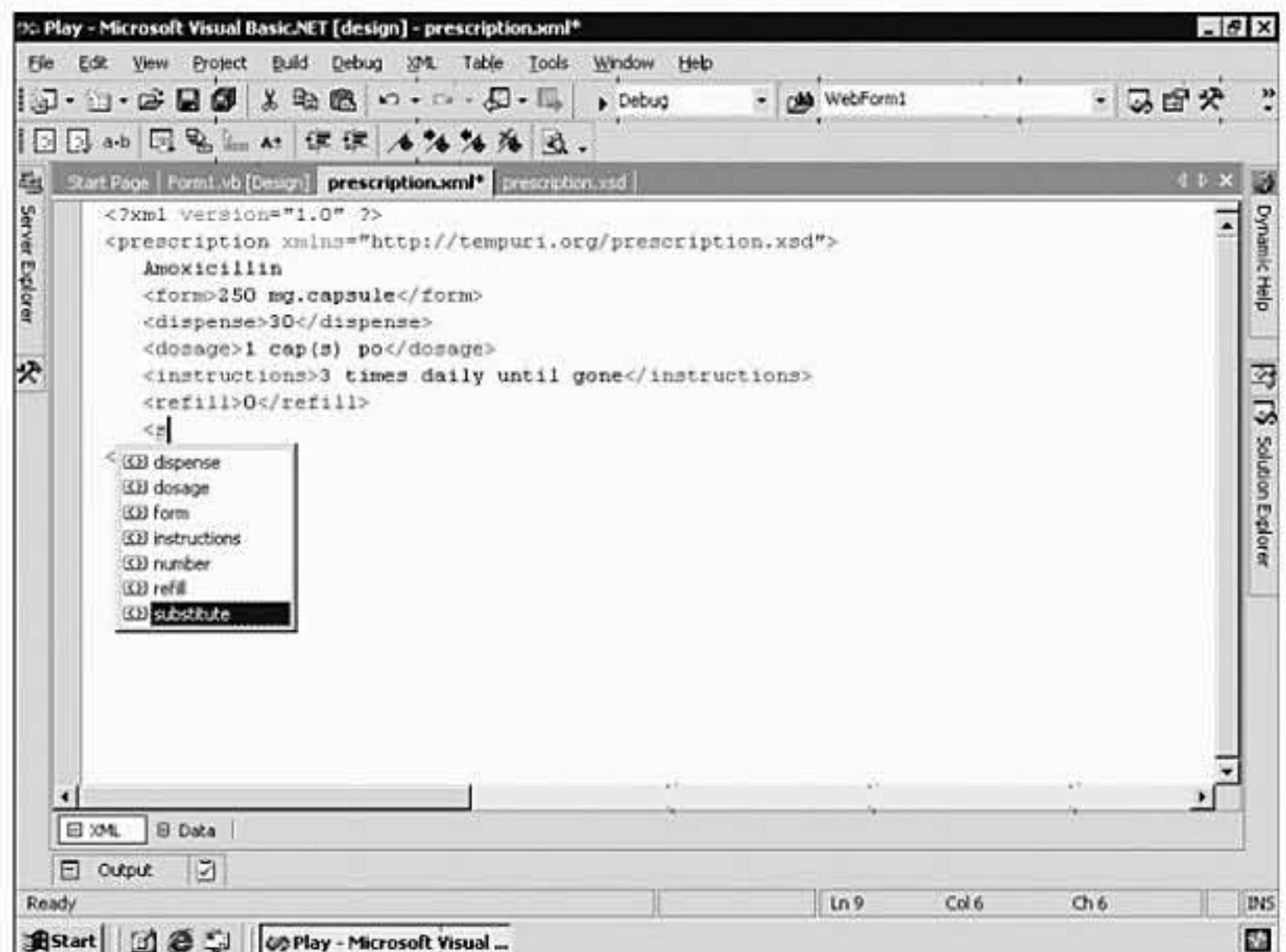


FIGURA 20.3
O Intellisense e a XML.



Trabalhando com a XML

Agora que você possui uma compreensão geral do que seria na verdade esse monstro chamado XML, como poderá usá-lo em seus aplicativos? A XML é flexível em sua utilização. Há muitas aplicações possíveis, mas as mais comuns são:

- Informações sobre configuração – Muitas pessoas usam a XML para armazenar informações sobre configuração. Na verdade, o próprio Visual Basic .NET emprega a XML em dados sobre configuração. Aí se incluem informações como as configurações de segurança para aplicativos Web, locais de módulos necessários e assim por diante. Utilizar a XML para formatar seus arquivos de configuração permitirá que você descreva melhor os valores. Pode-se usar o arquivo de configuração existente ou criar um. Gerar seu próprio arquivo em geral será mais seguro porque a configuração definida não será afetada.
- Transferência de dados – A XML é um formato excelente para a transferência de informações entre computadores. Qualquer computador que a receber poderá ler as informações contidas independentemente do sistema operacional ou da linguagem de programação. Examinaremos um exemplo sobre esse assunto no próximo capítulo quando abordarmos o SOAP (Simple Object Access Protocol); no entanto, você pode usar seu próprio formato para enviar XML de uma maneira tão fácil quanto essa.
- Armazenamento de dados – A XML é um grande substituto dos bancos de dados pequenos. Ela permite a criação de formatos de arquivo simples que sejam autodescritivos e possibilita a definição de relacionamentos, podendo ser editada por programas ou manualmente.

Nesta lição, examinaremos duas maneiras pelas quais o Visual Basic .NET torna fácil trabalhar com arquivos XML. Nos concentraremos mais na leitura de XML, mas também veremos como é possível criá-la. As duas técnicas que o Visual Basic .NET fornece para se trabalhar com a XML são:

- O Document Object Model
- Objetos de leitura e de gravação

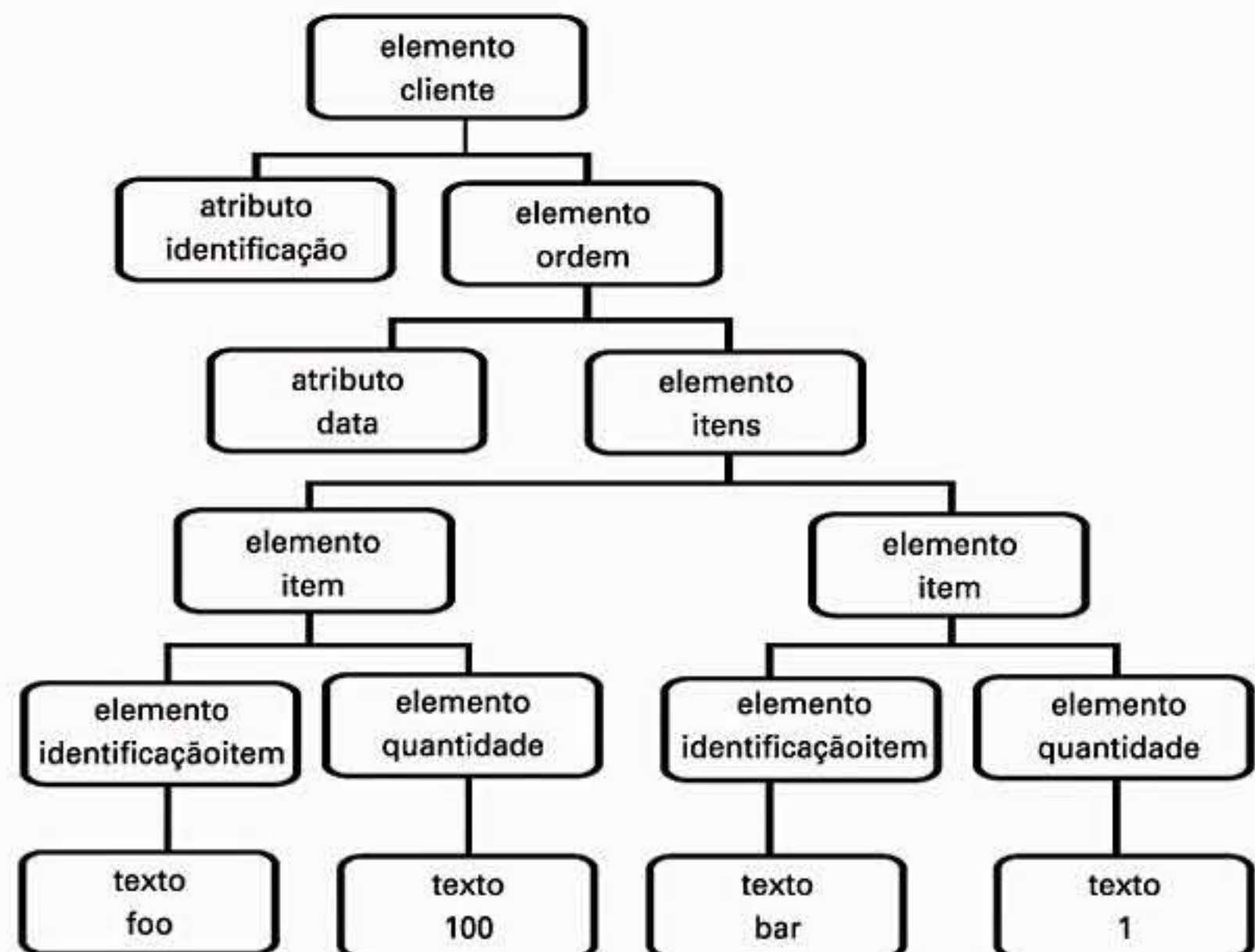
O Document Object Model

O Document Object Model (DOM) é uma maneira desajeitada de dizer ‘a forma-padrão de ler e criar XML’, mas é isso que ele é. O DOM representa a interface de programação-padrão recomendada pelo World Wide Web Consortium (www.w3.org) para se trabalhar com XML. Essa é a mesma organização que padronizou a XML e todas as tecnologias relacionadas com ela. Existe uma versão do DOM disponível nos sistemas operacionais mais populares, que pode ser utilizada pela maioria das linguagens de programação, inclusive o Visual Basic .NET. Ela é uma Application Programming Interface (API) para consultas a arquivos XML.

A idéia existente por trás do DOM consiste em que qualquer arquivo XML pode ser descrito em termos de uma árvore com nós. A Figura 20.4 mostra como isso poderia ser representado conceitualmente, dado o exemplo XML a seguir:

FIGURA 20.4

Estrutura lógica de um arquivo XML.



```

<?xml version="1.0"?>
<identificação cliente="12345">
  <data ordem="01/04/01">
    <itens>
      <item>
        <identificaçãoitem>foo</identificaçãoitem>
        <quantidade>100</quantidade>
      </item>
      <item>
        <identificaçãoitem>bar</identificaçãoitem>
        <quantidade>1</quantidade>
      </item>
    </itens>
  </ordem>
</cliente>
  
```

Há duas maneiras de examinar qualquer um dos nós de um DOM. De certo modo, todos são nós. No entanto, também são tipos específicos de nós. Por exemplo, o nó ordem representa um nó de elemento, enquanto o nó data é um nó de atributo. Cada tipo de nó dá suporte a todos os recursos de um nó genérico, assim como a métodos e propriedades específicos de seu tipo. A Tabela 20.1 descreve os tipos de nó mais usados.

TABELA 20.1 Nós Mais Usados em Arquivos XML

<i>Tipo de Nó</i>	<i>Descrição</i>
XmlElement	Representa um elemento de um arquivo XML.
XmlAttribute	Representa um atributo de um arquivo XML.
XmlDocument	Representa o documento como um todo.
XmlComment	Representa um comentário de um arquivo XML. Os comentários em XML começam com <!-- e terminam com -->.

Um benefício de ter vários tipos de nós herdados de um único nó é que todos compartilham um conjunto de propriedades e métodos. Isso permitirá que você aprenda mais facilmente como trabalhar com o DOM. A Tabela 20.2 resume as propriedades e métodos usados com mais frequência.

TABELA 20.2 Propriedades e Métodos Comuns a Todos os Nós XML

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Attributes	Conjunto de todos os atributos desse nó. Permite a navegação pela lista de atributos em cada nível da hierarquia.
ChildNodes	Conjunto de todos os nós-filhos desse nó. Essa é uma das maneiras principais de navegar pelo DOM, percorrendo os filhos com um laço For...Next.
FirstChild	Retorna o primeiro nó-filho do nó atual. Essa é outra das principais maneiras de navegar pelo DOM, encontrando o primeiro nó-filho e, em seguida, executando um laço enquanto NextSibling (discutido posteriormente nesta tabela) retorna um valor.
InnerText	O texto contido dentro de cada nó. Aí se inclui qualquer nó-filho que ele possua.
Name	O texto existente nos colchetes angulares do nó atual.
NextSibling	Retorna o próximo nó disponível no mesmo nível do atual. Por exemplo, na Figura 20.4, se o nó atual estivesse armazenado na variável oNode e apontando para o nó item, e você chamasse oNode=oNode.NextSibling, oNode seria apontado para o próximo nó item.
NodeType	Retorna o tipo do nó atual – por exemplo, XmlElement, XmlAttribute e assim por diante.
Métodos	
AppendChild	Adiciona um novo filho ao nó atual. É assim que é possível estender um conjunto de nós existente.

Usar `XmlDocument` para trabalhar com a XML é semelhante ao que vemos na árvore de nós da Figura 20.4. No topo da hierarquia se encontra um único nó-raiz, de nome `cliente`. Ele e todos os outros nós do DOM possuem um conjunto de nós-filhos. Por sua vez, todos esses nós também têm nós-filhos e assim por diante. Os objetos possuem métodos que o ajudarão a navegar nessa hierarquia. A Tabela 20.3 descreve alguns desses métodos e propriedades. Além disso, `XmlDocument` dá suporte às mesmas propriedades e métodos da classe `XmlNode`.

TABELA 20.3 Métodos e Propriedades de `XmlDocument`

<i>Membro</i>	<i>Descrição</i>
Propriedades	
<code>DocumentElement</code>	O nó-raiz do documento.
Métodos	
<code>CreateNode</code>	Usado para criar novos nós que serão adicionados ao documento. Este método genérico permitirá que você gere qualquer tipo de <code>XmlNode</code> .
<code>CreateElement</code>	Semelhante a <code>CreateNode</code> ; no entanto, <code>CreateElement</code> é usado para criar elementos.
<code>CreateAttribute</code>	Semelhante a <code>CreateNode</code> ; no entanto, <code>CreateAttribute</code> é usado para criar atributos.
<code>Load</code>	Carrega o conteúdo do arquivo XML em um documento.
<code>LoadXml</code>	Carrega em um documento o conteúdo de uma string que contém XML.
<code>Save</code>	Salva o conteúdo do documento XML em um arquivo.

Ao usar `XmlDocument` e o DOM para ler XML, você em geral empregará `ChildNodes` ou `FirstChild/NextSibling` para percorrê-la e encontrar os nós nos quais estiver interessado. Utilizar o DOM pode dar um pouco de trabalho porque só o nó `XmlDocument` é capaz de criar os vários elementos, atributos e assim por diante. Depois que o nó específico for criado, será possível adicioná-lo ao conjunto desejado com o método `AppendChild`.

O DOM é uma maneira-padrão de ler e criar arquivos XML. Em geral, ele é útil quando se trabalha com arquivos que são relativamente pequenos.

Objetos de Leitura e de Gravação

Embora o DOM seja uma maneira poderosa de ler e criar XML, na verdade apresenta várias limitações. A mais significativa delas é que quando se carrega um documento XML no DOM, esse documento tem de ser carregado integralmente na memória. Nesse momento, o DOM constrói a árvore inteira descrevendo o documento, com todos os nós, listas de nós e assim por diante. Como você deve ter deduzido, se o arquivo XML for grande, isso poderá ocupar muita memória.

Esse processo também consome muito tempo, principalmente se tudo o que você precisar for de um ou dois itens do documento.

O Visual Basic .NET possui outra estratégia para trabalhar com a XML – objetos de leitura e de gravação. Para ser mais específico, `XmlTextReader` e `XmlTextWriter`. O objeto `XmlTextReader` concede acesso veloz só de avanço a blocos XML, enquanto `XmlTextWriter` fornece uma maneira rápida e resumida de criar XML. Embora em algumas situações eles não sejam tão intuitivos quanto usar o DOM, na verdade são mais aperfeiçoados que ele em duas áreas principais:

- Não é preciso carregar todo o arquivo antes que o documento comece a ser processado – ele não precisa nem mesmo estar disponível. Isso poderá acontecer quando for executado o download de um arquivo XML da Internet. Por não precisar de todo o arquivo na memória, `XmlTextReader` e `XmlTextWriter` podem reduzir enormemente os requisitos gerais de memória de seu aplicativo.
- Os objetos `XmlTextReader` e `XmlTextWriter` são rápidos. Muito rápidos. Isso é devido ao fato de não precisarem construir todas as estruturas de memória, como as listas de nós, que o DOM usa.

No entanto há uma desvantagem em se trabalhar com `XmlTextReader` e `XmlTextWriter`. Já que eles não armazenam o documento integral na memória, às vezes pode se tornar difícil relacionar uma parte dele com a outra. Se você precisar manter o registro de várias partes do documento e ainda usar `XmlTextReader` e `XmlTextWriter`, terá de escrever um código para fazer isso.

Por meio de `XmlTextReader`, você pode ler o arquivo executando um laço com o método `Read`, parando em cada nó do documento XML. A seguir, decida se está interessado no nó atual e, caso esteja, aplique a ele as outras propriedades e métodos. A Tabela 20.4 mostra alguns dos métodos e propriedades de `XmlTextReader` mais usados.

TABELA 20.4 Propriedades e Métodos Comuns de `XmlTextReader`

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Name	Nome do nó atual.
NodeType	O tipo de nó atual – se é elemento, atributo e assim por diante.
Métodos	
Read	Lê o próximo nó do documento XML. A cada vez que este método é chamado, ele passa para o próximo nó.

O objeto `XmlTextWriter` é usado na construção do documento XML, acrescentando as tags individuais de abertura e fechamento dos elementos. Ele pode ser uma maneira natural de criar um

documento chamando métodos para gerar cada parte. A Tabela 20.5 descreve as propriedades e métodos de `XmlTextWriter` que são geralmente usados.

TABELA 20.5 Propriedades e Métodos Comuns de `XmlTextWriter`

<i>Membro</i>	<i>Descrição</i>
Propriedades	
Formatting	Usada para configurar se a XML resultante será formatada. Em geral, quando a finalidade for sua leitura pelas pessoas, esta propriedade deverá ser configurada como <code>Formatting.Indented</code> , já que essa configuração cria um documento mais legível. Como alternativa, se esta propriedade não estiver configurada ou se seu valor for <code>Formatting.None</code> , toda a XML aparecerá em uma única linha.
Métodos	
Close	Encerra o objeto de edição. Como com os outros objetos de edição que examinou, você deve sempre encerrá-lo quando não for mais usá-lo.
WriteElementString	Cria um elemento no documento. Esse elemento novo possuirá as tags de abertura e de fechamento, assim como qualquer texto que se refira a ele. Este é o método que você deve usar quando criar um elemento independente que não tenha nenhum nó-filho.
WriteStartElement	Cria um elemento no documento. Este método só gera a tag de abertura para o elemento. Você deve usá-lo quando criar elementos que tenham nós-filhos.
WriteEndElement	Cria a tag de fechamento para o elemento atual.

Lendo XML

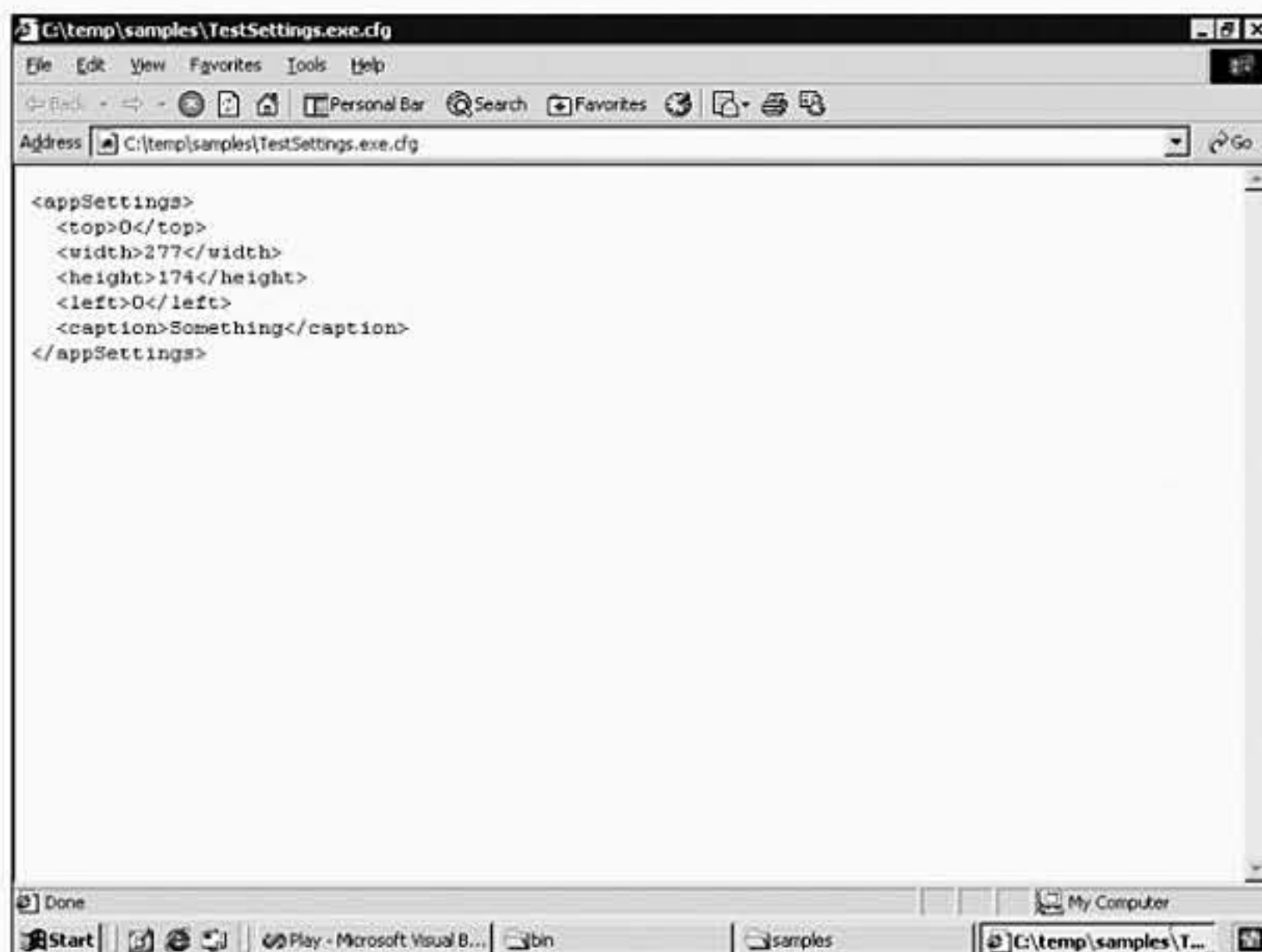
Embora você pudesse usar os métodos `IndexOf` e `Substring` da classe `String` para ler XML e conhecer seu conteúdo, essa não é a maneira mais eficiente de fazer isso. Como alternativa, é recomendável usar o DOM ou os objetos `XmlTextReader` para ler XML. Essas duas famílias de objetos podem interpretar com maior facilidade arquivos XML.

O DOM será mais apropriado se você tiver de ler um arquivo XML que esteja todo na memória e trabalhar com ele. Como descrevi, o DOM nos permite percorrer o arquivo avançando ou retornando, relacionar uma seção com outra e alterar o arquivo no momento em que estivermos lendo. Por outro lado, `XmlTextReader` é mais apropriado quando precisamos apenas ler o arquivo percorrendo-o uma única vez.

Para saber como os dois conjuntos de objetos funcionam, seria bom vê-los em ação em um exemplo. Muitos aplicativos precisam ser personalizados pelos usuários, como pode ser feito por meio das configurações de Tools na caixa de diálogo Options. Essas informações podem ser facilmente armazenadas em um arquivo XML, permitindo que você ou seus usuários leiam ou

alterem as configurações sem problemas. Inserir o código dessa funcionalidade em um componente proporcionará uma reutilização mais fácil dela. A classe Settings permite o armazenamento de definições de configuração em um arquivo e que essas sejam retornadas para leitura posteriormente. O conteúdo do arquivo será semelhante ao da Figura 20.5. Criar uma classe dessas o ajudará a permitir que o usuário personalize o aplicativo conforme suas necessidades. A Listagem 20.1 mostra as propriedades e métodos básicos dessa classe. Depois adicionaremos a ela a capacidade para leitura e edição das configurações.

FIGURA 20.5
Arquivo de configuração.



LISTAGEM 20.1 Classe AppSettings

```
1 Imports System.Xml
2
3 Public Class Settings
4     Private m_sFileName As String
5     Private m_oItems As Hashtable
6
7     Public Sub New()
8         Me.FileName = AppDomain.CurrentDomain.BaseDirectory &
9             AppDomain.CurrentDomain.FriendlyName & ".cfg"
10    End Sub
11
12    Public Sub New(ByVal fileName As String)
13        Me.FileName = fileName
```


LISTAGEM 20.1 Classe AppSettings (*continuação*)

```
14     End Sub
15
16     Public Property FileName() As String
17         Get
18             Return m_sFileName
19         End Get
20         Set(ByVal Value As String)
21             m_sFileName = Value
22             LoadDOM()
23         End Set
24     End Property
25
26     Public Property Items() As Hashtable
27         Get
28             Return m_oItems
29         End Get
30         Set(ByVal Value As Hashtable)
31             m_oItems = Value
32         End Set
33     End Property
34
35 End Class
```

ANÁLISE

A classe possui dois construtores (linhas 7 a 10 e 12 a 14). Isso foi feito para adicionar flexibilidade. O programador pode criar uma instância dessa classe usando `oSettings = New Settings()`, caso em que o arquivo empregado é o padrão, composto do nome do aplicativo, seguido pela extensão `.cfg`. Como alternativa, ele pode gerar o objeto `Settings` nomeando o arquivo que contém as informações sobre a configuração, como em `oSettings = New Settings("AlgumArquivo.xml")`. De qualquer uma das duas maneiras, a propriedade `FileName` é configurada. Depois que isso é feito (linhas 20 a 23), a variável interna `m_sFileName` é configurada, e o método `LoadDOM` (que será criado em breve) é chamado. Esse método carrega no conjunto `HashTable` interno as configurações que foram estabelecidas no arquivo desejado. `HashTable` é um dos conjuntos definidos em `System.Collections`. Ele armazena vários itens por nome, como a propriedade `Tables` do objeto `DataSet`. Isso permitirá que você recupere itens da propriedade `Items` usando um código parecido com `oSettings.Items("AlgumaConfiguração")`.

É claro que antes de poder usar essa classe, você deve adicionar a funcionalidade que carregará as configurações. A Listagem 20.2 mostra como fazer isso com `XmlTextReader`, enquanto a Listagem 20.3 descreve o mesmo, porém empregando o DOM.

LISTAGEM 20.2 Carregando as Configurações Por Meio de XmlTextReader

```
36 Public Sub LoadReader()  
37     'carrega o conteúdo do arquivo de configuração em  
38     ' HashTable  
39     m_oItems = New Hashtable()  
40     Dim oReader As XmlTextReader  
41     Dim sLastElement As String  
42  
43     Try  
44         oReader = New XmlTextReader(FileName)  
45         While (oReader.Read)  
46             Select Case oReader.NodeType  
47                 Case XmlNodeType.Element  
48                     If oReader.Name <> "appSettings" Then  
49                         m_oItems.Add(oReader.Name, Nothing)  
50                         sLastElement = oReader.Name  
51                     End If  
52                 Case XmlNodeType.Text  
53                     m_oItems.Item(sLastElement) = oReader.Value  
54                 Case Else  
55                     End Select  
56             End While  
57         Catch ex As Exception  
58         Finally  
59             oReader.Close()  
60             oReader = Nothing  
61         End Try  
62     End Sub
```

ANÁLISE

Primeiro o método LoadReader inicializa Hashtable (linha 39) e declara o objeto XmlTextReader além de uma variável string que será usada durante o processamento.

Já que esse processamento é executado em um arquivo que pode ou não existir, sua abertura e leitura são protegidas pela inserção em um bloco Try...End Try (linhas 43 a 61). Como antes, a cláusula Finally desse bloco (que começa na linha 58) garante que o objeto de leitura seja fechado e eliminado, mesmo se um erro ocorrer.

Grande parte desse método é representada pelo laço While...End While das linhas 45 a 56. Ele lê cada nó do arquivo. Quando um nó é lido, primeiro o código determina seu tipo (linha 46). Nesse código, só nos preocupamos com nós de elemento e de texto porque eles são os únicos tipos de nós que existem no arquivo. Se o nó atual for um elemento e não for o nó appSettings (o nó-raiz), o código adicionará um novo item a Hashtable (linha 49) e salvará o nome desse item recém-adicionado na variável string. O próximo método Read deve retornar o nó de texto asso-

ciado ao elemento, e isso servirá para adicionar o valor ao último item inserido em HashTable. No final da rotina, HashTable deve conter todas as configurações armazenadas no arquivo.

LISTAGEM 20.3 Carregando Configurações com o Uso do DOM

```
63 Public Sub LoadDOM()
64     m_oItems = New Hashtable()
65     Dim oDoc As New XmlDocument()
66     Dim oNode As XmlElement
67
68     Try
69         oDoc.Load(FileName)
70
71         oNode = oDoc.DocumentElement.FirstChild
72         Do While Not IsNothing(oNode)
73             m_oItems.Add(oNode.Name, oNode.InnerText)
74             oNode = oNode.NextSibling
75         Loop
76
77     Catch ex As Exception
78     Finally
79         oDoc = Nothing
80     End Try
81 End Sub
```

ANÁLISE

O código começa de maneira semelhante ao da listagem anterior, inicializando HashTable. Aqui, no entanto, ele cria um objeto XmlDocument e um XmlElement para armazenar os nós quando o arquivo for lido. Novamente, um bloco Try...End Try é usado como proteção caso um erro ocorra na leitura do arquivo. Primeiro, o arquivo é carregado (linha 69). Lembre-se de que nesse estágio o arquivo inteiro é carregado na memória, e o DOM é construído. Na linha 71, duas operações de navegação são definidas. Em primeiro lugar, DocumentElement (o nó appSettings) é selecionado; em seguida, o primeiro filho desse nó, FirstChild, também o é. Quando esse código for concluído, deverá apresentar como resultado oNode apontando para o primeiro nó contido dentro do nó appSettings. A seguir, o código executa um laço pelos irmãos desse nó, adicionando itens a HashTable. Quando não houver irmãos, o método NextSibling retornará Nothing, e o laço terminará. Como antes, XmlDocument é configurado como Nothing (linha 79) no final da execução.

Agora você pode adicionar essa classe a outros projetos, ou a uma biblioteca de classes de sua autoria e convertê-la em uma DLL. Quando terminar, será possível usar a classe como mostra a Listagem 20.4 para configurar seu aplicativo.

LISTAGEM 20.4 Usando a Classe Settings

```
1: Private Sub frmTest_Load(ByVal sender As System.Object, _  
2:     ByVal e As System.EventArgs) _  
3:     Handles MyBase.Load  
4:     ' 'comentários: blah  
5:     oSettings = New Settings()  
6:     With Me  
7:         .Height = oSettings.Items("height")  
8:         .Width = oSettings.Items("width")  
9:         .Left = oSettings.Items("left")  
10        .Top = oSettings.Items("top")  
11        .Text = oSettings.Items("caption")  
12    End With  
13 End Sub
```

ANÁLISE

Para ler as configurações e atribuí-las ao aplicativo atual, primeiro inicialize o objeto Settings (linha 5). Lembre-se de que ele carrega as configurações nesse momento. Em seguida, leia cada configuração e atribua-as à propriedade desejada do formulário, a um controle dele ou use-as quando necessário (linhas 6 a 12).

Gravando XML

Exatamente como quando lemos a XML, é possível criá-la apenas usando strings. Isto é, você pode gerar um código XML adicionado as tags apropriadas de abertura e fechamento do elemento a uma variável string. No entanto, será muito mais provável que você crie um arquivo XML corretamente se empregar o DOM ou as classes `XmlTextWriter`.

A Listagem 20.5 mostra o código necessário para salvar as informações de configuração no objeto Settings usando `XmlTextWriter`, enquanto a Listagem 20.6 descreve o mesmo código para objetos `XmlDocument`. Esses códigos devem ser adicionados ao objeto Settings criado anteriormente.

LISTAGEM 20.5 Criando Definições de Configuração com `XmlTextWriter`

```
82 Public Sub SaveWriter()  
83     Dim oWriter As XmlTextWriter  
84     Dim oItem As DictionaryEntry  
85  
86     oWriter = New XmlTextWriter(m_sFileName, Nothing)  
87     oWriter.Formatting = Formatting.Indented  
88     oWriter.WriteStartElement("appSettings")  
89  
90     For Each oItem In m_oItems  
91         oWriter.WriteElementString(oItem.Key, oItem.Value)  
92     Next
```

LISTAGEM 20.5 Criando Definições de Configuração com XmlTextWriter
(continuação)

```

93      oWriter.WriteEndElement()
94
95      oWriter.Flush()
96      oWriter.Close()
97
98  End Sub

```

ANÁLISE

O método SaveWriter começa instanciando as variáveis necessárias. Uma variável que pode parecer estranha é o objeto `oItem` declarado na linha 84. Os itens de `HashTable` são os declarados como `DictionaryEntry`; portanto, para permitir o uso de um laço `For Each...Next` no processamento da tabela, precisamos de um desses objetos.

O código da gravação começa abrindo o arquivo (linha 86) e atribuindo que a formatação deve ser recuada. Lembre-se de que isso é para beneficiar a pessoa que for ler, e não é necessário para nenhum código. A seguir, o elemento inicial do nó-raiz é criado (linha 88), e todos os itens são gravados no arquivo XML durante o laço `For Each...Next` (linhas 90 a 92). O método `WriteElementString` cria tanto a tag de abertura quanto a de fechamento com base no nome do item de `HashTable`, assim como o texto contido nas tags baseado no valor desse item. Para concluir, a tag de fechamento do nó-raiz é criada (linha 93), e o código é encerrado, gravando as informações no arquivo real.

LISTAGEM 20.6 Criando Definições de Configuração com o DOM

```

99  Public Sub SaveDOM()
100      Dim oDoc As New XmlDocument()
101      Dim oRoot As XmlElement
102      Dim oItem As DictionaryEntry
103      Dim oNode As XmlElement
104
105      oRoot = oDoc.CreateElement("appSettings")
106      oDoc.AppendChild(oRoot)
107
108      For Each oItem In m_oItems
109          oNode = oDoc.CreateElement(oItem.Key)
110          oNode.InnerText = oItem.Value
111          oRoot.AppendChild(oNode)
112      Next
113
114      oDoc.Save(m_sFileName)
115  End Sub

```

ANÁLISE

O código para edição com o DOM começa criando o nó-raiz e inserindo-o no documento. Todos os outros nós serão acrescentados ao nó `appSettings`. Isso é feito em um laço `For Each...Next` como aconteceu no método `SaveWriter`. Os nós são criados (linha 109); o texto é atribuído (110) e adicionado ao nó `appSettings`.

Agora que o código para salvar as configurações foi adicionado, você pode alterar o programa que o usará para que as salve na próxima vez que o aplicativo for executado. Em geral isso é feito no evento `Closing` de um formulário, como vemos na Listagem 20.7.

LISTAGEM 20.7 Código para Salvar as Configurações

```
14 Private Sub frmTest_Closing(ByVal sender As Object, _  
15     ByVal e As System.ComponentModel.CancelEventArgs) _  
16     Handles MyBase.Closing  
17     With Me  
18         oSettings.Items("height") = .Height  
19         oSettings.Items("width") = .Width  
20         oSettings.Items("left") = .Left  
21         oSettings.Items("top") = .Top  
22         oSettings.Items("caption") = .Text  
23     End With  
24     oSettings.SaveWriter()  
25 End Sub
```

ANÁLISE

Antes que você possa salvar as configurações no arquivo, elas devem ser copiadas com os valores reais. Normalmente, isso seria feito quando a caixa de diálogo `Options` da opção `Tools` fosse fechada ou o aplicativo fosse encerrado. Quando todas as definições de configurações forem copiadas, poderemos usar o método `SaveWriter` ou `SaveDOM` para salvar as configurações no arquivo. Depois de adicionar esse código, tente executar o projeto (veja a Figura 20.6). Altere o tamanho do formulário e mova-o para outra posição na tela. Feche o formulário, encerrando o aplicativo, e execute-o mais uma vez. O formulário deve aparecer na mesma posição da tela, com o tamanho para o qual foi alterado. Fazer com que seu aplicativo saiba as posições do formulário dessa maneira, em geral é apreciado pelos usuários.

FIGURA 20.6
Testando a classe
Settings.



Resumo

A XML é uma dessas tecnologias que podem ser examinadas em muitos níveis. É simples, ainda que tenha efeitos profundos. Você pode criá-la com um programa ou em um simples editor de texto. De maneira semelhante, é possível escrever um programa para processá-la e ao mesmo tempo ela ainda poderá ser lida pelas pessoas. No entanto, criando ou processando-a, a XML fornece uma maneira poderosa de formatar informações. À parte todo o exagero, acho que ela é uma das mais importantes inovações em computação, ficando atrás apenas do ASCII. Dessa maneira, é sua responsabilidade aprender pelo menos um pouco do que ela pode fazer para ajudá-lo e a seus aplicativos.

No próximo capítulo, concluiremos nosso estudo do Visual Basic .NET examinando uma nova maneira de desenvolver aplicativos – os serviços Web. O aprendizado da XML será útil porque os serviços Web a utilizam bastante.

P&R

P Li sobre outra maneira de processar XML. O que é SAX?

R O SAX (Simple API for XML) é ainda outra maneira de ler XML que se tornou popular graças aos desenvolvedores de Java e às versões posteriores do MSXML (analisador XML para o COM da Microsoft). Ele foi desenvolvido por David Megginson e a lista de distribuição XML-DEV é uma maneira simples de ler XML. É semelhante a `XmlTextReader` por não carregar o documento inteiro na memória. No entanto, quando usado, é em muitos aspectos o oposto de `XmlTextReader`. Enquanto esse objeto é um modelo de ‘recuperação’, extraíndo trechos da XML, o SAX é um modelo de ‘eventos’. Quando um processador SAX lê um arquivo XML, ele aciona eventos, como `BeginElement`, `EndElement`, `BeginDocument` e assim por diante. Você deve criar os manipuladores de eventos nos quais estiver interessado para processar a XML.

P Ouvi falar de vários outros acrônimos relacionados à XML, como XSLT, XPath, SOAP e outros. Onde posso encontrar mais informações sobre eles?

R Há muitos padrões estabelecidos e propostos para tratamento da XML – uma quantidade muito grande para que abordemos aqui. Se você estiver interessado em aprender mais sobre como usar esses outros padrões, deve procurar um bom livro sobre XML (existem muitos deles) ou encontrar um site interessante de alguma comunidade na Web que forneça essas informações (o endereço <http://msdn.microsoft.com/xml> seria um bom ponto de partida).

P Disseram-me que algumas linguagens, como a Java e a C#, permitem que sejam adicionados documentos a um arquivo-fonte quando ele estiver sendo escrito por meio da conversão dos comentários em documentação. Posso fazer isso com o Visual Basic .NET?

R Infelizmente, o Visual Basic .NET por enquanto não dá suporte a esse recurso. No entanto, uma versão simples de como adicioná-lo a seu aplicativo está on-line. Esse aplicativo permite que você adicione tags <summary> a cada propriedade e classe dos métodos de seu programa inserindo-as em um documento XML que poderá ser utilizado como a base da documentação de suas classes (veja as Figuras 20.7 e 20.8).

FIGURA 20.7
Documentação XML.

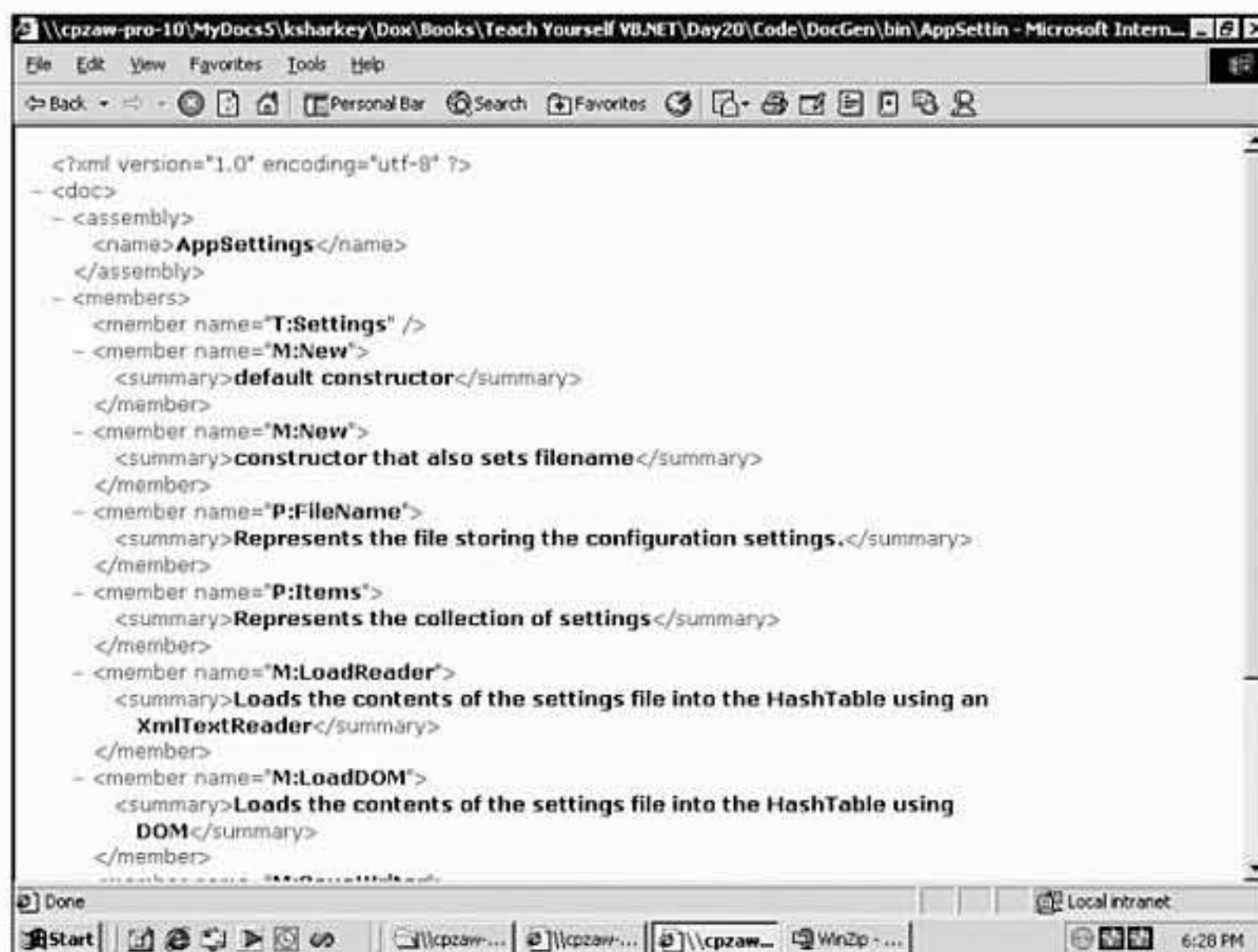
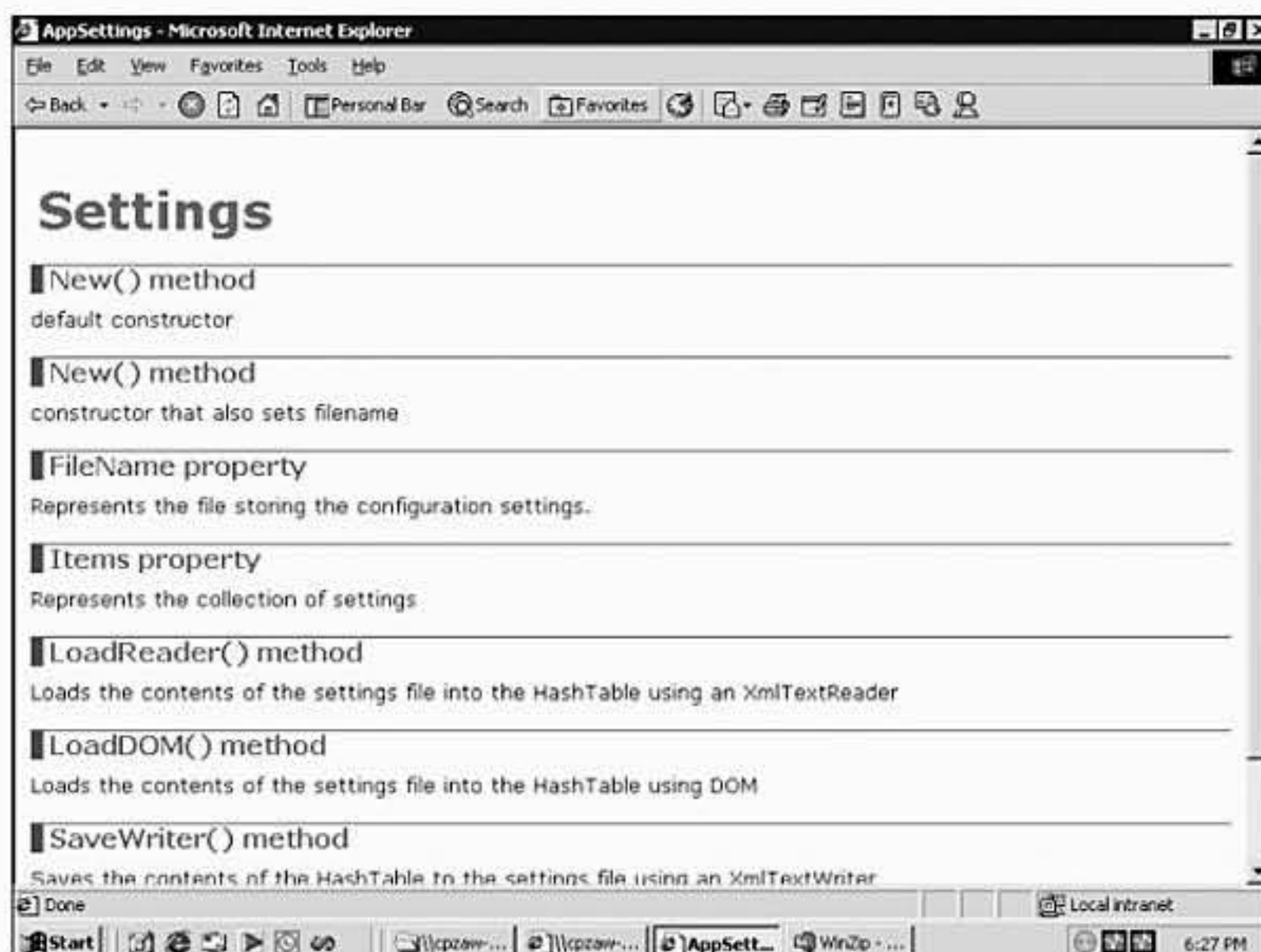


FIGURA 20.8
Documentação XML com uma folha de estilo aplicada a ela.



Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Como posso adicionar rapidamente um novo elemento, incluindo as tags de abertura e de fechamento e algum texto, a um arquivo XML usando `XmlTextWriter`?
2. Em que situação você optaria por usar o DOM para ler um arquivo XML em vez de empregar a classe `XmlTextReader`?
3. O que é metadado?

Exercícios

O código mostrado nesta lição, que testa a classe `Settings`, no momento define o uso do método `LoadDOM` para carregar as configurações e do método `SaveWriter` para salvá-las. Escreva outro formulário de teste que salve várias configurações e permita que o usuário selecione qual dos dois métodos empregar para salvar e recuperar.

PÁGINA EM BRANCO

SEMANA 3

DIA 21

Criando Serviços Web com o Visual Basic .NET

A menos que você tenha vivido em uma caverna (ou em uma ilha do Pacífico Sul, aguardando resgate) nos últimos anos, provavelmente ouviu falar da Internet. É aquela invenção maravilhosa – imagine uma rede mundial – ah, ouviu comentários sobre ela. De qualquer modo, a Internet é uma ferramenta valiosa para os desenvolvedores se beneficiarem dela. Fornece uma rede essencialmente gratuita que pode ser usada para a transferência de informações de seus programas para outros, em qualquer lugar do planeta onde possam estar. Portanto, criar programas que utilizem a Internet é obviamente muito importante para os desenvolvedores do Visual Basic .NET. Nesta lição, abordaremos:

- O que é um Serviço Web.
- O SOAP (e qual sua relação com os Serviços Web).
- Como criar um Serviço Web no Visual Basic .NET.
- Como criar um Serviço Web cliente no Visual Basic .NET.

O Que É um Serviço Web?

É óbvio para qualquer pessoa que já tenha usado a World Wide Web que há muitos serviços disponíveis nesse local. Eles podem ser utilizados em quase qualquer lugar; serviços de mecanismos de busca, de compras, serviços que recuperam informações e assim por diante. Portanto, por que dedicar uma lição à criação de Serviços Web em um livro sobre o Visual Basic? Esta lição explica a diferença entre um serviço na Web e um Serviço Web.



À guisa de esclarecimento, me referirei aos serviços comuns na Web em letras minúsculas e aos Serviços Web criados no Visual Basic .NET com uma combinação de maiúsculas e minúsculas.

Interagimos com um serviço ‘normal’ na Web navegando em alguma página na Internet, preenchendo um formulário e enviando essas informações a qualquer outra página da Web. É fácil, e essa simplicidade no uso contribuiu definitivamente para o crescimento e a popularidade da Internet. No entanto, não há uma maneira fácil de pegar as informações que você obtém em um serviço na Web e usá-la em um programa. Por exemplo, é possível utilizar um serviço na Web que procure o valor atual de uma ação na qual podemos estar interessados. Em geral, digitamos o símbolo da ação, damos um clique em um botão e acessamos uma página da Web que apresenta o preço atual dela. Não raro, também obtemos outras informações, como os preços de abertura e fechamento, as alterações desde o último fechamento e às vezes um demonstrativo com as alterações recentes da ação. Tudo isso é muito prático quando navegamos na Web, mas, se o desejado fosse apenas obter o preço atual da ação e usá-lo em outro programa, seríamos forçados a tomar uma entre duas medidas:

- Fazer com que o usuário procure a informação, preencha o formulário, aguarde o resultado e, em seguida, digite-o em seu programa. O problema aqui é bem óbvio – a maioria dos usuários não fará isso com frequência. Imagine tentar executar essas operações com algum valor, como o preço de ações, em que as alterações são constantes. Ou tentar persuadir alguém a inserir informações em uma página da Web, dar um clique em um botão e digitar um valor em seu programa com uma frequência maior do que uma ou duas vezes por hora.
- Fazer com que seu programa pesquise a página da Web (em segundo plano), a analise (leia) e extraia as informações que você quer. Isso em geral é conhecido como *varrer* uma página da Web. O problema encontrado aqui ocorrerá se os autores da página alterarem o layout, forçando-o a alterar seu programa para que analise o novo layout. Além disso, se resolver alterar o site da Web que estiver usando para suas informações provavelmente também terá de reescrever o código que analisa a página da Web.

Certo, e os Serviços Web? Esses são programas que fornecem algum serviço na Internet, mas são projetados para permitir que outros programas se comuniquem com eles, em vez de precisar que pessoas façam isso. Portanto, agora, alguém poderia criar um Serviço Web que permitisse a outros programas recuperar preços de ações, executar pesquisas e assim por diante. Um recurso adequado dos Serviços Web é que podem ser escritos em qualquer sistema operacional e linguagem de programação. Eles usam um formato de mensagem que pode ser lido e criado facilmente.

O Visual Basic .NET torna a criação desses Serviços Web tão fácil quanto gerar qualquer outro tipo de aplicativo. Os detalhes da criação e leitura da mensagem têm sua visualização bloqueada pelo próprio Visual Basic .NET. De maneira semelhante, o Visual Basic .NET faz com que o uso dos Serviços Web seja tão simples quanto chamar algum outro objeto, porém ele poderia estar na

Internet e até ter sido escrito em outra linguagem de programação ou executado em um sistema operacional diferente.

Os Serviços Web não só permitem que dois programas se comuniquem pela Internet, mas também que um programa reúna informações de vários Serviços Web em um único aplicativo, aumentando potencialmente o valor dos dados. Por exemplo, um aplicativo que manipulasse ações poderia não só exibir o valor atual de várias ações diferentes (possivelmente recuperadas de diversos Serviços Web que lidassem com ações), como também incluir as recomendações atuais de analistas com base nas ações exibidas, empresas semelhantes, notícias sobre a empresa e assim por diante, tudo recuperado de vários Serviços Web diferentes.

Os Serviços Web se comunicam uns com os outros e com os clientes por meio dos protocolos-padrão da Internet. Eles podem empregar o protocolo comum HTTP, o mesmo que os navegadores utilizam. Nesse caso, eles usam os comandos GET e POST que os navegadores da Web empregam. Como alternativa, um Serviço Web pode utilizar o SOAP para se comunicar com um cliente ou outro serviço. Usar o SOAP permite que a comunicação seja mais sofisticada porque ele deixa que objetos sejam transferidos entre dois aplicativos, e o HTTP não.

O Simple Object Access Protocol

O SOAP é um protocolo relativamente novo. Ele define uma maneira de formatar a XML para que seja usada como uma mensagem. Os Serviços Web em geral empregam o SOAP para comunicar a solicitação e a resposta entre dois programas. Ele utiliza os protocolos comuns da Internet, como o HTTP, que é usado pelos navegadores da Web, ou o SMTP, que manipula a correspondência na Internet. Porém, o SOAP permite uma comunicação mais sofisticada do que a que existiria se você só utilizasse mensagens simples.

O SOAP foi desenvolvido inicialmente por várias empresas, inclusive a Microsoft, para permitir uma comunicação sofisticada de objeto a objeto entre programas. De certo modo, o objetivo do SOAP era definir uma maneira de trabalhar com objetos na Internet, por meio dos protocolos e formatos-padrão, em vez de formatos binários e protocolos proprietários. Depois de ficar disponível por algum tempo, os desenvolvedores melhoraram seus recursos. Agora ele é um formato geral de troca de mensagens que pode ser usado por qualquer protocolo e atende a muitas necessidades além da criação de Serviços Web. No entanto, esta lição examinará o SOAP apenas no contexto de sua utilização na criação e uso de Serviços Web. Para obter mais detalhes sobre o SOAP, leia a especificação. Ela está disponível na Internet em <http://www.w3.org/TR/SOAP>.

O Protocolo

Embora os detalhes reais do protocolo SOAP estejam ocultos no .NET Framework, você pode ficar curioso para descobrir o que acontece em segundo plano. Portanto, examinemos qual a aparência de uma mensagem SOAP e como ela pode ser usada para enviar um objeto pela rede.

A mensagem SOAP nada mais é do que uma mensagem XML que usa um formato específico, ou *esquema*. O esquema define a estrutura geral da mensagem (veja Dia 20, “Introdução à XML”, para obter mais detalhes sobre os esquemas). Esse formato permite a inserção de uma grande quantidade de informações na mensagem, incluindo objetos e dados adicionais sobre como cada lado da comunicação deve interpretá-la. Por exemplo, a mensagem pode conter informações sobre segurança, por meio da identificação do usuário que fez a solicitação. A Listagem 21.1 mostra uma solicitação SOAP simples.

CÓDIGO**LISTAGEM 21.1** Uma Solicitação SOAP Simples

```
1  xml version "1.0" encoding "utf-8"
2  soap:Envelope
3  xmlns:xsi "http://www.w3.org/2001/XMLSchema-instance"
4  xmlns:xsd "http://www.w3.org/2001/XMLSchema"
5  xmlns:soap "http://schemas.xmlsoap.org/soap/envelope/"
6  soap:body
7      Add xmlns="http://tempuri.org/"
8          x int
9          y int
10         Add
11     soap:body
12 soap:Envelope
```

ANÁLISE

Nessa mensagem há três seções principais. A primeira (nas linhas 2 a 5) é a tag `Envelope`. Esse é o elemento-raiz da mensagem SOAP. Os espaços de nome incluídos definem a estrutura da mensagem, o tipo de codificação e a utilização de esquemas XML.

A segunda seção ocorre na linha 6, com a declaração do corpo (Body) da mensagem, que contém a solicitação real. Essa solicitação, que é a terceira seção, é definida nas linhas 7 a 10. Nesse caso, é uma solicitação `Add`, com dois valores, `x` e `y`.

Web Service Description Language (WSDL)

Depois que tiver uma maneira de se comunicar, você precisará de um meio para descrever a mensagem. Quando se trata de linguagens, chamamos isso de *gramática*. A gramática define como construir uma mensagem e identificar partes importantes dela.

Embora o SOAP defina o formato da mensagem, você também pode precisar de uma maneira de descrever a gramática das mensagens SOAP. Isso permitirá que os clientes e os servidores gerem e reconheçam automaticamente mensagens SOAP válidas. A *Web Service Description Language* (WSDL) é uma maneira de descrever Serviços Web. Ela define as mensagens (*métodos*) que têm suporte de um Serviço Web e os componentes (*parâmetros*) de cada mensagem. De certo modo, a WSDL é o ‘contrato’ que conecta o cliente ao servidor.

A WSDL depende de outro padrão, os esquemas XML, que, como você aprendeu no Dia 20, definem a estrutura das mensagens XML. Ela usa os esquemas para definir as duas mensagens que compõem a comunicação típica do Serviço Web, a solicitação e a resposta. Os esquemas para a solicitação e a resposta são uma parte do arquivo WSDL. Outra parte desse arquivo descreve o fato de duas estruturas estarem relacionadas. Para concluir, o arquivo WSDL identifica o destino ao qual o cliente deve enviar a mensagem SOAP a fim de que ela possa ser processada. A Listagem 21.2 mostra um exemplo de contrato WSDL.



Se você teve a oportunidade de trabalhar com versões Beta anteriores do Visual Basic .NET, como a lançada na Professional Developers' Conference (PDC), pode ter ouvido falar de algo chamado SDL (Service Description Language). A SDL é um formato mais antigo que foi substituído pela WSDL.

LISTAGEM 21.2 Um Contrato WSDL

```

1: <?xml version="1.0" encoding="utf-8" ?>
2:   <definitions xmlns:s="http://www.w3.org/2001/XMLSchema"
3:     xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
4:     xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
5:     xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
6:     xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
7:     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
8:     xmlns:s0="http://tempuri.org/"
9:     targetNamespace="http://tempuri.org/"
10:    xmlns="http://schemas.xmlsoap.org/wsdl/">
11:    <types>
12:      <s:schema attributeFormDefault="qualified"
13:        elementFormDefault="qualified"
14:        targetNamespace="http://tempuri.org/">
15:        <s:element name="Add2Ints">
16:          <s:complexType>
17:            <s:sequence>
18:              <s:element minOccurs="1" maxOccurs="1"
19:                name="X" type="s:int" />
20:              <s:element minOccurs="1" maxOccurs="1"
21:                name="Y" type="s:int" />
22:            </s:sequence>
23:          </s:complexType>
24:        </s:element>
25:        <s:element name="Add2IntsResponse">
26:          <s:complexType>
27:            <s:sequence>
28:              <s:element minOccurs="1" maxOccurs="1"
29:                name="Add2IntsResult" type="s:int" />

```


LISTAGEM 21.2 Um Contrato WSDL (*continuação*)

```
30:         </s:sequence>
31:     </s:complexType>
32: </s:element>
33:     <s:element name="int " type="s:int" />
34: </s:schema>
35: </types>
36: <message name="Add2IntsSoapIn">
37:     <part name="parameters" element="s0:Add2Ints" />
38: </message>
39: <message name="Add2IntsSoapOut">
40:     <part name="parameters" element="s0:Add2IntsResponse" />
41: </message>
42: <message name="Add2IntsHttpGetIn">
43:     <part name="X" type="s:string" />
44:     <part name="Y" type="s:string" />
45: </message>
46: <portType name="Service1Soap">
47:     <operation name="Add2Ints">
48:         <input message="s0:Add2IntsSoapIn" />
49:         <output message="s0:Add2IntsSoapOut" />
50:     </operation>
51: </portType>
52: <binding name="Service1Soap" type="s0:Service1Soap">
53:     <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
54:         style="document" />
55:     <operation name="Add2Ints">
56:         <soap:operation soapAction="http://tempuri.org/Add2Ints"
57:             style="document" />
58:         <input>
59:             <soap:body use="literal" />
60:         </input>
61:         <output>
62:             <soap:body use="literal" />
63:         </output>
64:     </operation>
65: </binding>
66: <service name="Service1">
67:     <port name="Service1Soap" binding="s0:Service1Soap">
68:         <soap:address
69:             location="http://localhost/Add2Ints/Service1.asmx" />
70:     </port>
71: </service>
72: </definitions>
```


Discovery

Depois de definir uma mensagem e uma maneira de descrevê-la, o próximo recurso que as pessoas esperarão obter é poderem encontrar o Serviço Web. De certo modo, isso é semelhante a ter um número de telefone ou fax em um cartão de visitas. Pela inclusão dos números no cartão, estamos anunciando essencialmente o fato de que há um Serviço Web com o qual se comunicar usando essas ferramentas. Para o SOAP e os Serviços Web, esse cartão de visitas é o Discovery, ou DISCO na abreviatura. O DISCO, além de ser um estilo de música dos anos 70, é um formato XML (poderia ser algo mais?) que descreve os serviços que têm suporte de um servidor. Outros computadores podem empregar esse arquivo DISCO para descobrir os recursos de uma máquina específica. O arquivo DISCO é em geral colocado no diretório-raiz de um site da Web, permitindo que os clientes o encontrem com facilidade.

O Visual Basic .NET dá suporte a dois tipos de arquivos DISCO. O primeiro é editado manualmente, como o da Listagem 21.3. Esse arquivo lista cada serviço que tem suporte. Você pode criar com facilidade um desses arquivos adicionando um novo arquivo Disco a um projeto e inserindo manualmente os serviços desejados como vemos na listagem.

CÓDIGO

LISTAGEM 21.3 Um Arquivo DISCO Simples

```

1  xml version "1.0"
2  discovery xmlns "http: schemas.xmlsoap.org DISC "
3  contractRef ref " demos Async Add.asmx sdl"
4      docRef " demos Async Add.asmx"
5      xmlns "http: schemas.xmlsoap.org DISC scl "
6  contractRef ref " demos Interop CCValidator.asmx sdl"
7      docRef " demos Interop CCValidator.asmx"
8      xmlns "http: schemas.xmlsoap.org DISC scl "
9  contractRef ref " demos ebServices Async 99 ottles.asmx sdl"
10      docRef " demos ebServices Async 99 ottles.asmx"
11      xmlns "http: schemas.xmlsoap.org DISC scl "
12  discovery
```

ANÁLISE

Cada arquivo DISCO é composto de vários contratos. Cada *contrato* (ou *contractRef*) identifica um arquivo WSDL que descreve um Serviço Web. Para cada serviço Web listado, há três atributos importantes:

- **ref** – O local de um arquivo WSDL
- **docRef** – O local de um Serviço Web
- **xmlns** – O espaço de nome que define o formato do arquivo WSDL. Isso permite que um único arquivo DISCO seja usado com várias linguagens de descrição de serviço.

O segundo tipo de arquivo DISCO à que o Visual Basic .NET dá suporte é o de busca dinâmica. Esse é o tipo de arquivo de busca criado quando se inicia um projeto de Serviço Web. Para dife-

reenciá-lo do arquivo DISCO manual, ele apresenta a extensão vsdisco. Esse arquivo pesquisa todos os diretórios do servidor Web, e localiza todos os Serviços Web expostos. Em seguida, retorna essa lista (em formato DISCO simples) para o cliente. Esse tipo de arquivo DISCO é conveniente porque torna desnecessária qualquer alteração para que se adicione um novo Serviço Web. Ele será encontrado de modo automático pelo arquivo DISCO dinâmico. Você pode preferir não usar esse tipo de arquivo DISCO se quiser controlar mais rigidamente os serviços que disponibilizar. A Listagem 21.4 mostra o exemplo de um arquivo DISCO dinâmico.

CÓDIGO**LISTAGEM 21.4** Um Arquivo DISCO Dinâmico

```
1  xml version "1.0 "  
2  dynamicDiscovery xmlns "urn:schemas-dynamicdiscovery:DISC .2000-03-17"  
3  exclude path " vti cnf"  
4  exclude path " vti pvt"  
5  exclude path " vti log"  
6  exclude path " vti script"  
7  exclude path " vti txt"  
8  exclude path " eb References"  
9  dynamicDiscovery
```

ANÁLISE

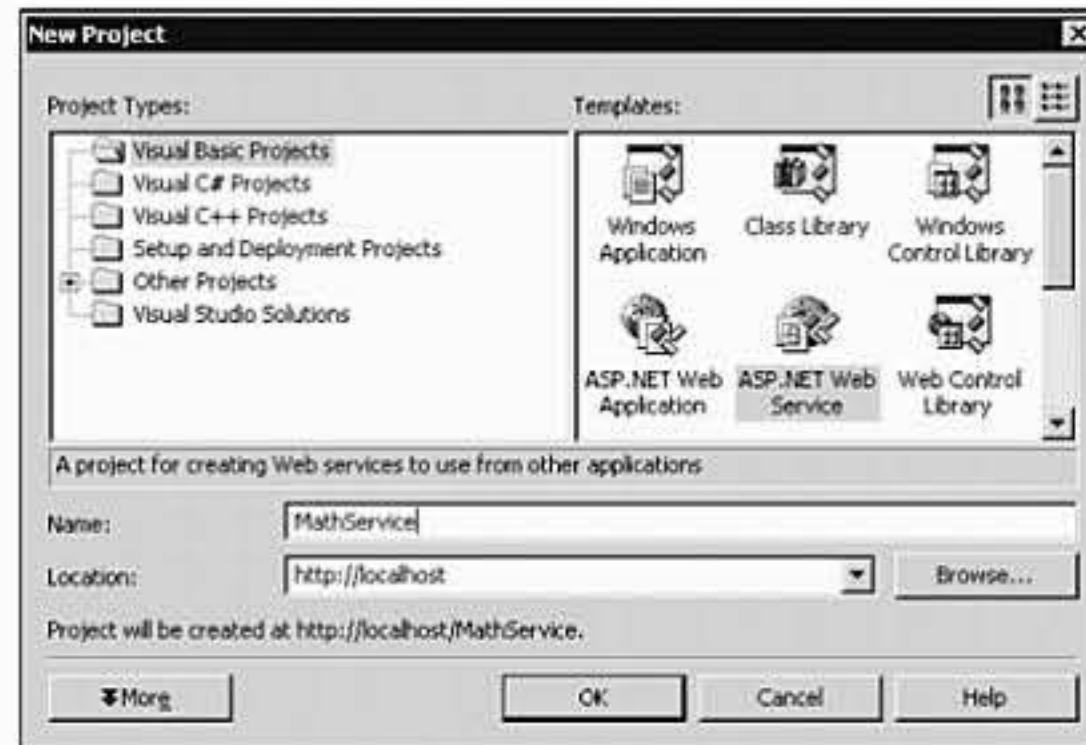
A manutenção dessa forma de arquivo DISCO é muito mais fácil do que a do arquivo de busca simples. Em vez de identificar todos os Serviços Web (ou os arquivos WSDL) que têm suporte, ele lista os locais onde não se deve procurar arquivos WSDL. Cada linha `<exclude path=" " />` listada identifica um diretório que não deve ser pesquisado. Esse tipo de arquivo DISCO é um pouco mais lento que o manual porque precisa que o mecanismo de busca pesquise todos os subdiretórios, em vez de apenas verificar os listados na instrução `<exclude path=" " />`.

Agora que você tem tudo preparado (ou pelo menos conhece os protocolos que usará), pode utilizar o Visual Basic .NET para criar um Serviço Web.

Criando um Serviço Web Simples

Você pode criar o aplicativo de um Serviço Web, por estranho que pareça, adicionando um projeto de Serviço Web à solução ou gerando uma que já tenha um projeto desse tipo (veja a Figura 21.1).

FIGURA 21.1
Criando um Projeto de Serviço Web.



Para saber o que está envolvido em sua criação e utilização, você gerará um Serviço Web bem básico – um serviço ‘matemático’. Esse é um exemplo simples de um Serviço Web. Embora seja um serviço básico, ele demonstra bem como é seu funcionamento. Qualquer funcionalidade mais complexa não será muito diferente. A lógica existente na criação de um Serviço Web é sempre a mesma.

Criando o Projeto

Quando estiver no ambiente de desenvolvimento, selecione New Project e ASP.NET Web Service. Chame o novo Serviço Web de MathService.

Após sua criação, o projeto apresentará vários arquivos.

- **Web.Config** Este é o arquivo usado pelo processador ASP.NET para sua configuração. Ele pode ser empregado na alteração das configurações do diretório virtual utilizado para esse Serviço Web. Por exemplo, você poderia ativar Tracing, o que permitiria uma melhor monitoração da utilização. O exemplo não alteraria este arquivo.
- **Global.asax** Este é o arquivo usado pelo processador ASP.NET para o armazenamento de manipuladores de eventos. Você pode empregar este arquivo na criação de códigos que serão executados quando eventos importantes ocorrerem durante a existência do aplicativo Web. O exemplo não alteraria este arquivo.
- **MathService.vsdisco** Este é o arquivo de busca do Serviço Web. Ele permite que o cliente encontre os serviços Web expostos nesse diretório virtual. O arquivo DISCO criado no projeto é de busca dinâmica, portanto, você não precisa fazer nenhuma alteração para expor mais serviços.
- **Service1.asmx** Este é o arquivo que você editará para criar a funcionalidade do Serviço Web.

A Listagem 21.5 mostra o código do arquivo Service1.asmx.

CÓDIGO**LISTAGEM 21.5** O Arquivo de um Serviço Básico

```
1 Imports System.Web.Services
2
3 Public Class Service1
4     Inherits System.Web.Services.WebService
5
6     #Region " Web Services Designer Generated Code "
7
8     Public Sub New()
9         MyBase.New()
10
11         'Esta chamada é necessária para o gerador de Serviços Web.
12         InitializeComponent()
13
14         'Adicione seu próprio código de inicialização depois da chamada
15         'InitializeComponent()
16
17     End Sub
18
19     'Necessária para o gerador de Serviços Web
20     Private components As System.ComponentModel.Container
21
22     'OBSERVAÇÃO:O procedimento a seguir é necessário para o gerador de Serviços Web
23     'Ele pode ser alterado com o uso do gerador de Serviços Web.
24     'Não o altere usando o editor de códigos.
25     <System.Diagnostics.DebuggerStepThrough()>Private Sub InitializeComponent()
26         components = New System.ComponentModel.Container()
27     End Sub
28
29     Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
30         'CODEGEN: Este procedimento é necessário para o gerador de Serviços Web
31         'Não o altere usando o editor de códigos.
32     End Sub
33
34     #End Region
35
36     ' EXEMPLO DE SERVIÇO WEB
37     ' O serviço HelloWorld() do exemplo retorna a string Hello World.
38     ' Para compilá-lo, elimine os caracteres de comentário das linhas a seguir,
39     ' salve e compile o projeto.
40     ' Para testar este serviço da Web, certifique-se de que o arquivo .asmx é a
41     ' página inicial
42     ' e pressione F5.
43     '
44     ' <WebMethod()>Public Function HelloWorld() As String
```


CÓDIGO**LISTAGEM 21.5** O Arquivo de um Serviço Básico (*continuação*)

```

42 '           HelloWorld = "Hello World"
43 ' End Function
44
45 End Class

```

ANÁLISE

O módulo do Serviço Web começa como a maioria dos arquivos do Visual Basic .NET, com os espaços de nome incluídos. O espaço de nome mais importante existente nesse caso é `System.Web.Services`. Ele contém as rotinas essenciais que adicionam a funcionalidade do Serviço Web.

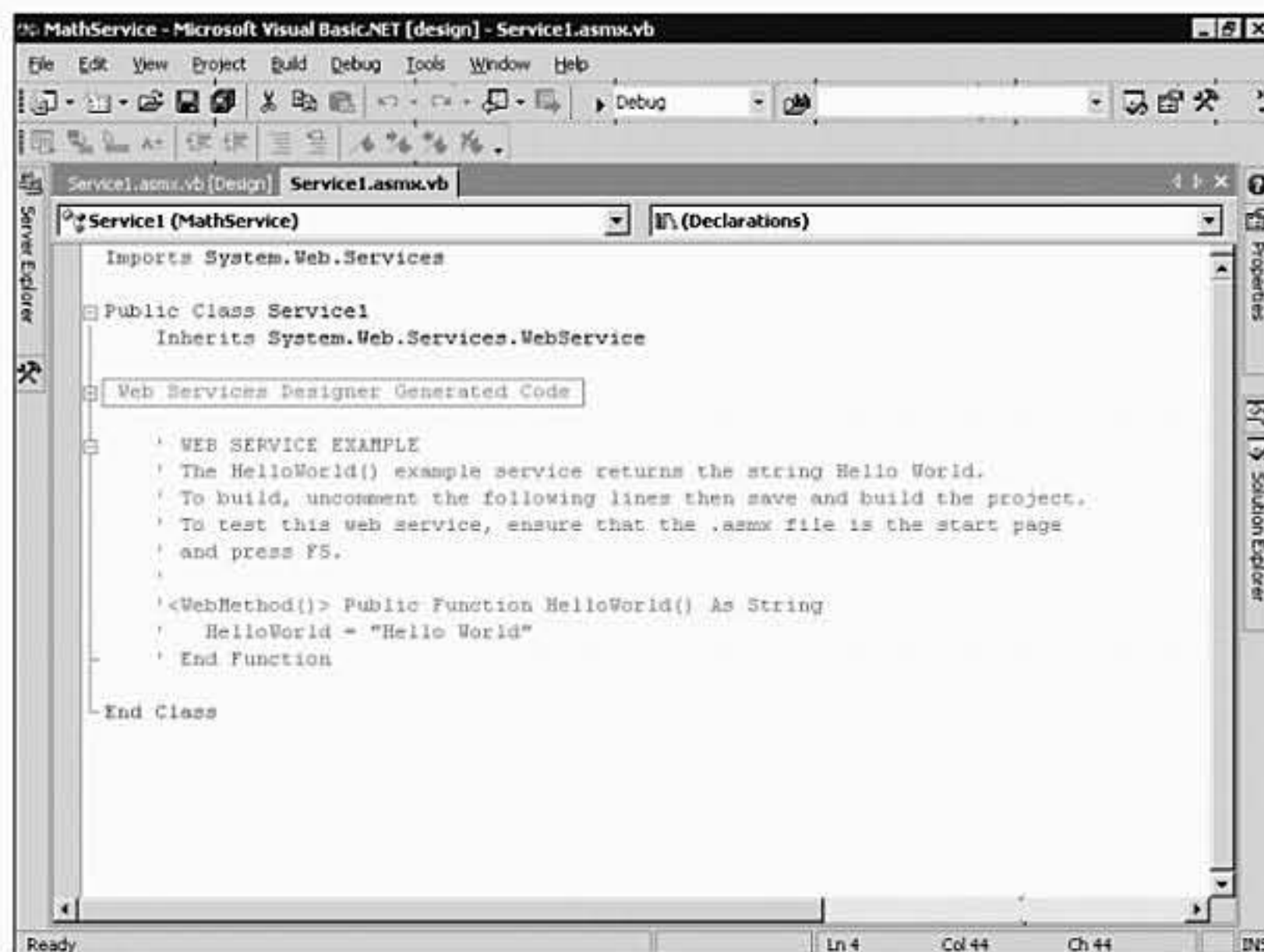
Observe na linha 4 que a classe herda características de `System.Web.Services.WebService`. Isso fornece várias propriedades a seu Serviço Web, como pode ser visto na Tabela 21.1. A maioria permite que ele acesse o interior do ASP.NET. Entre essas estão as propriedades `Request`, `Response`, `Session` e `Application`.

TABELA 21.1 Propriedades do Serviço Web

<i>Propriedade</i>	<i>Descrição</i>
<code>Application</code>	Fornece acesso às variáveis armazenadas no objeto <code>Application</code> dentro do ASP.NET. É uma propriedade que resume <code>Context.Application</code> .
<code>Context</code>	Concede acesso a todos os dados fornecidos no ASP.NET. Isso inclui os objetos <code>Request</code> e <code>Response</code> reais, o objeto <code>Cache</code> e outras propriedades da solicitação.
<code>Server</code>	Fornece acesso ao objeto <code>Server</code> . Esse objeto é usado em consultas ao servidor Web que estiver hospedando o Serviço Web ou na codificação de mensagens (por exemplo, para converter uma string comum em uma string de consulta (Querystring) válida, adicionando todos os caracteres + e %20 apropriados além de outros mais). É uma propriedade que resume <code>Context.Server</code> .
<code>Session</code>	Fornece acesso às variáveis armazenadas no objeto <code>Session</code> dentro do ASP.NET. É uma propriedade que resume <code>Context.Session</code> .
<code>User</code>	Fornece acesso às informações sobre o usuário que estiver fazendo a solicitação.

O código das linhas 6 a 33 da Listagem 21.5 foi inserido em uma área `#Region`. Esse código na verdade é minimizado no ambiente de desenvolvimento (veja a Figura 21.2). Essa é uma técnica útil, nova no Visual Basic .NET, que lhe permite tornar seu código menos complexo ocultando os detalhes que possam não ser relevantes. Nesse caso, ela oculta o código usado pelo próprio IDE. Esse é o código que permite arrastar e soltar itens na superfície visível. Em geral, ele não deve ser alterado manualmente, portanto faz sentido ocultá-lo no editor.

FIGURA 21.2
O Serviço Web no
estágio inicial.



O código das linhas 41 a 43 é um exemplo de como você adicionaria um novo método a esse Serviço Web. Observe que ele fica desativado por padrão. Se quiser expor esse método, apenas remova os caracteres de comentário.

Adicionando o Código

Agora que você possui um Serviço Web básico, precisa adicionar o código que executará as tarefas dele. Isso é feito em um Serviço Web pela inserção de métodos públicos em sua classe. Também poderíamos adicionar propriedades públicas, porém teria um significado menos relevante (como veremos posteriormente). Os métodos públicos de um Serviço Web são os mesmos de qualquer outra classe, com mais um item, o atributo `<WebMethod()>`.

A Sintaxe dos Métodos do Serviço Web

SINTAXE

A sintaxe do método do Serviço Web é

```
<WebMethod()> Public Function MethodName(ByVal Parameter As Type)As Return
Type
    'Código do método
End Function
```

na qual *MethodName* é o nome do método, *Parameter* é a lista de parâmetros do método, *Type* é o tipo de cada parâmetro e *ReturnType* é o tipo retornado pelo método Web.

A inclusão em um método comum é indicada por `<WebMethod()>` na declaração normal. Esse é o exemplo de um *atributo*, um elemento .NET que adiciona um novo comportamento a uma clas-

se, método ou propriedade. Na sintaxe do exemplo, o atributo adiciona o Serviço Web para que estabeleça a comunicação com um método específico por meio do SOAP. Qualquer método marcado com esse atributo será exposto publicamente como parte do Serviço Web.

No exemplo a seguir, adicionei apenas um método simples ao Serviço Web, incluindo o código da Listagem 21.6 no projeto do Serviço Web, depois do método do exemplo. Como alternativa, você pode excluir o comentário e substituí-lo por esse método.

LISTAGEM 21.6 Código para um Exemplo de Método de Serviço Web

```
1 <WebMethod()> Public Function Add(_  
2   ByVal X As Integer,_  
3   ByVal Y As Integer) As Integer  
4     Return X + Y  
5 End Function
```

Esse método apenas soma dois números, retornando o total. Embora você possa desconsiderá-lo por ser muito básico, ele possui tudo que em geral é comum em todos os Serviços Web:

- Contém o atributo <WebMethod> na declaração
- Retorna um valor

Você poderia adicionar mais métodos, ou métodos mais complexos, neste momento. No entanto, isso não é necessário. Escreveremos um Serviço Web mais realista ainda nesta lição.

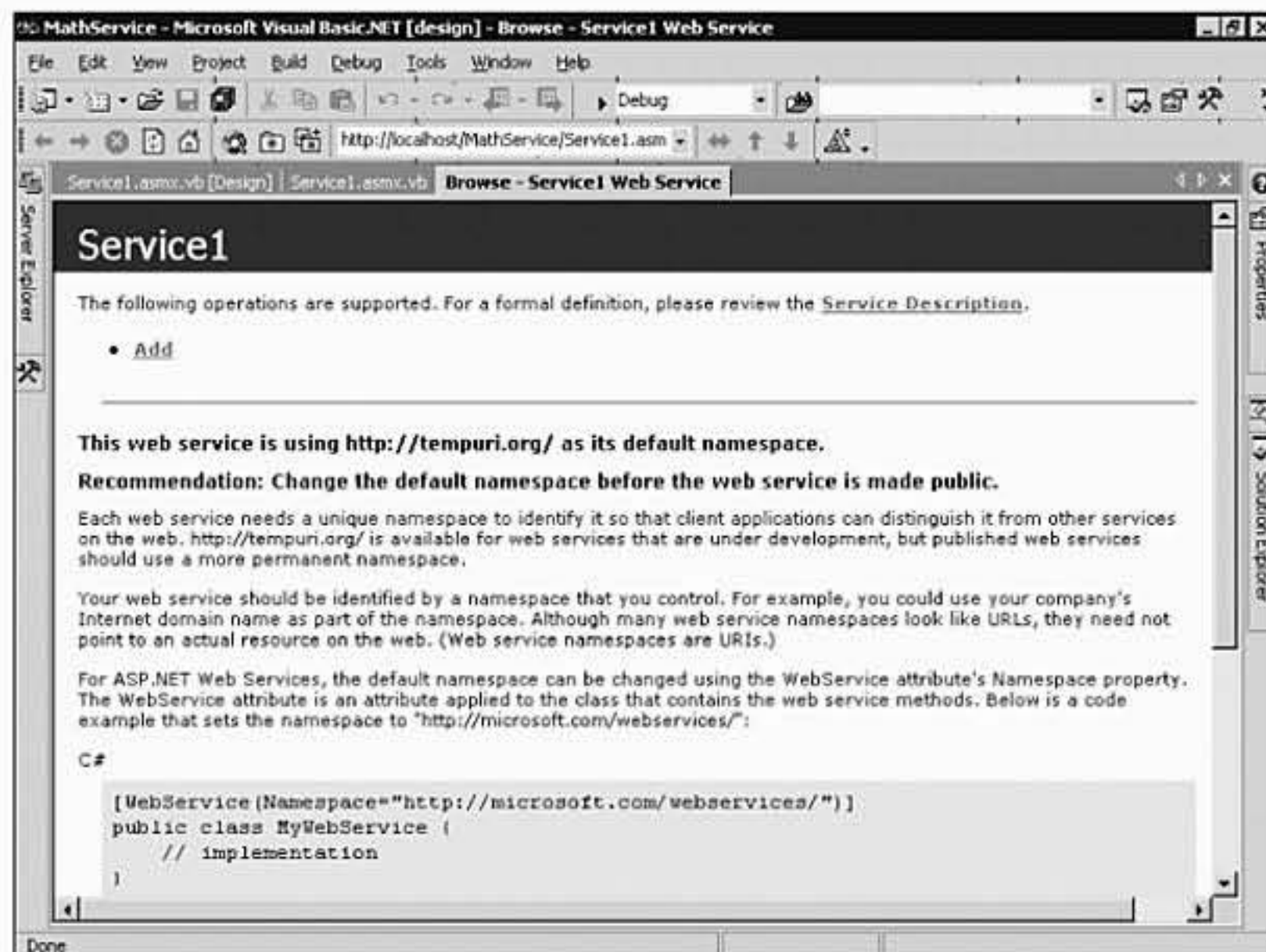
Compilando o Serviço Web

O Serviço Web é compilado da mesma maneira que outros projetos. Selecione Build no menu Build. Isso criará uma DLL (que será inserida no diretório bin sob o diretório-raiz do Serviço Web) contendo o código efetivo do Serviço Web. Depois de compilar o serviço, você poderá pesquisar o arquivo ASMX criado. Dê um clique com o botão direito do mouse sobre o arquivo ASMX no Server Explorer e selecione View in Browser para vê-lo em ação. O arquivo ASMX é aquele que contém a funcionalidade do Serviço Web, do mesmo modo que o arquivo .exe é um programa ou o ASPX é um formulário Web. As Figuras 21.3 e 21.4 mostram visualizações do projeto de exemplo. O URL visualizado é

<http://localhost/MathService/service1.asmx>

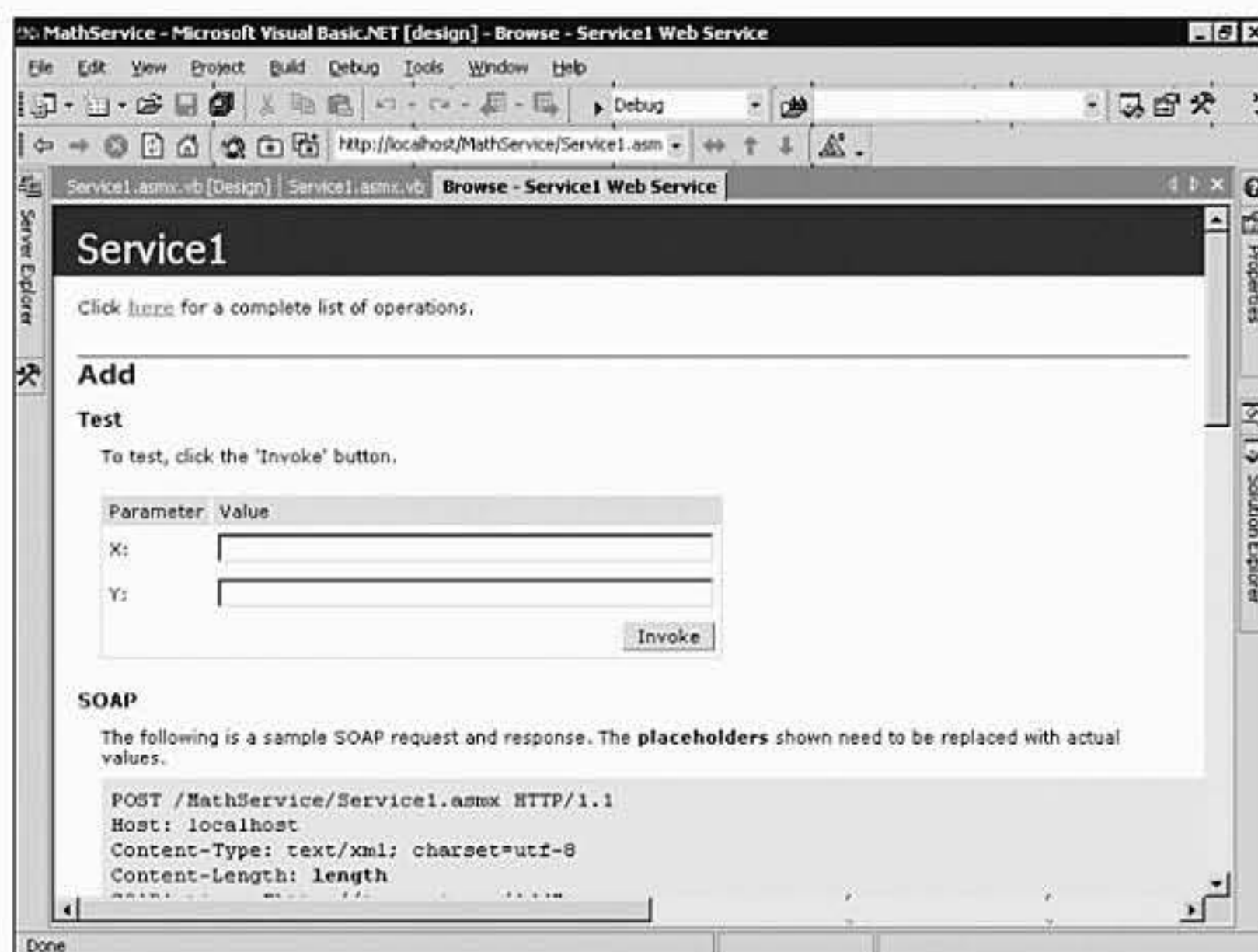
A primeira tela que veremos mostrará a descrição do Serviço Web, junto aos nomes de todos os métodos aos quais foi adicionado o atributo WebMethod (veja a Figura 21.3).

FIGURA 21.3
Página de teste do
Serviço Web.



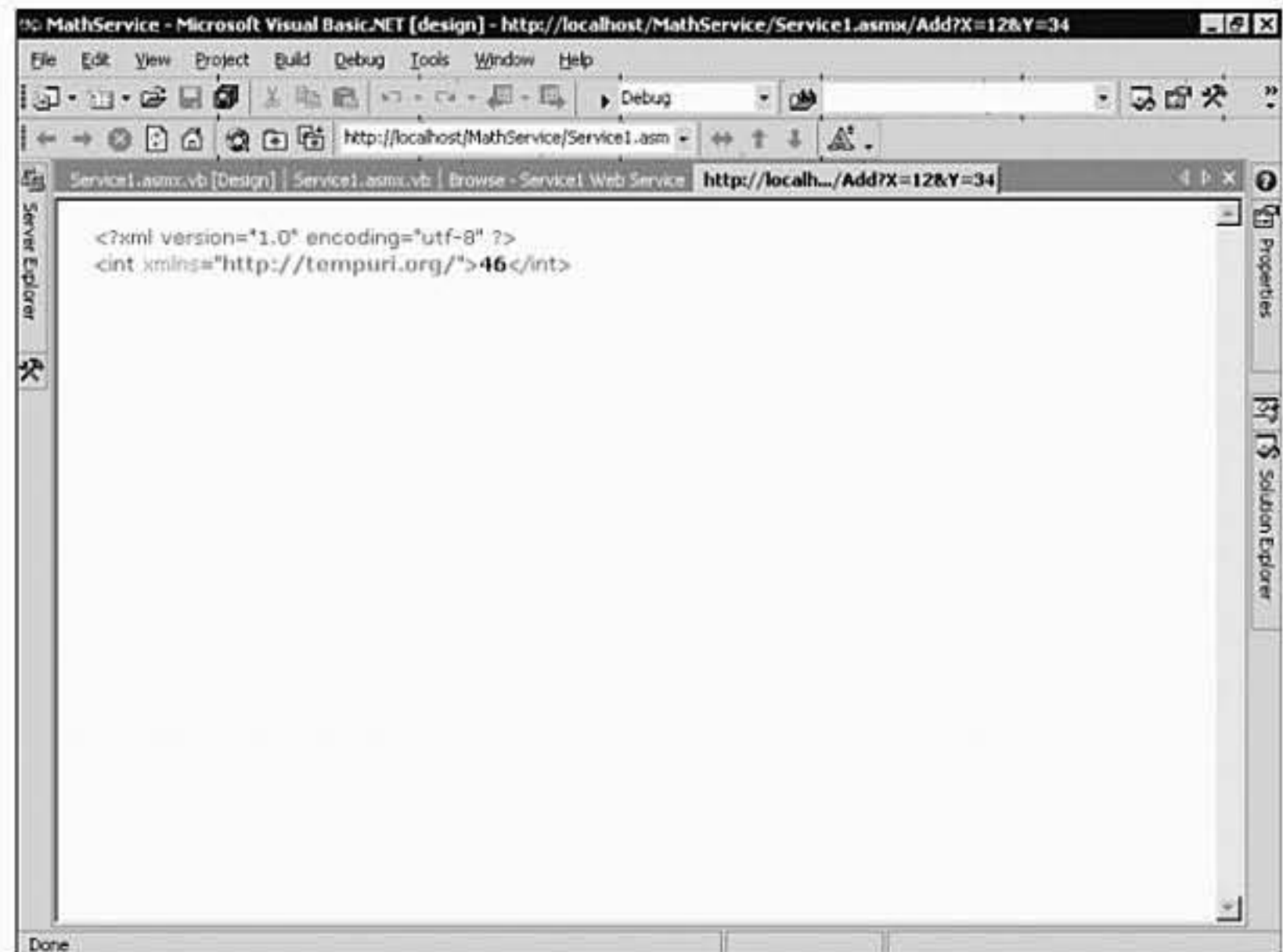
Se você selecionar um dos métodos, como o método `Add` recém-criado, deve ver outra página de teste que possui um formulário que permite que sejam inseridos valores de teste no Serviço Web (veja a Figura 21.4). Além disso, ela exibe um exemplo das mensagens SOAP de solicitação e resposta desse método.

FIGURA 21.4
Inserindo valores na
página de teste.



Essa página de teste será gerada automaticamente quando você visualizar um arquivo ASMX. Ela fornecerá informações sobre os métodos que o Serviço Web disponibiliza. Além disso, cria um formulário de teste para cada método, permitindo que alguns valores sejam inseridos e a função executada. Digite alguns valores nos campos de texto da seção do método Add e dê um clique no botão Invoke. Outra janela do navegador deve ser aberta mostrando o total, como na Figura 21.5.

FIGURA 21.5
*Adicionando valores
com o Serviço Web.*



Criando um Serviço Web Cliente

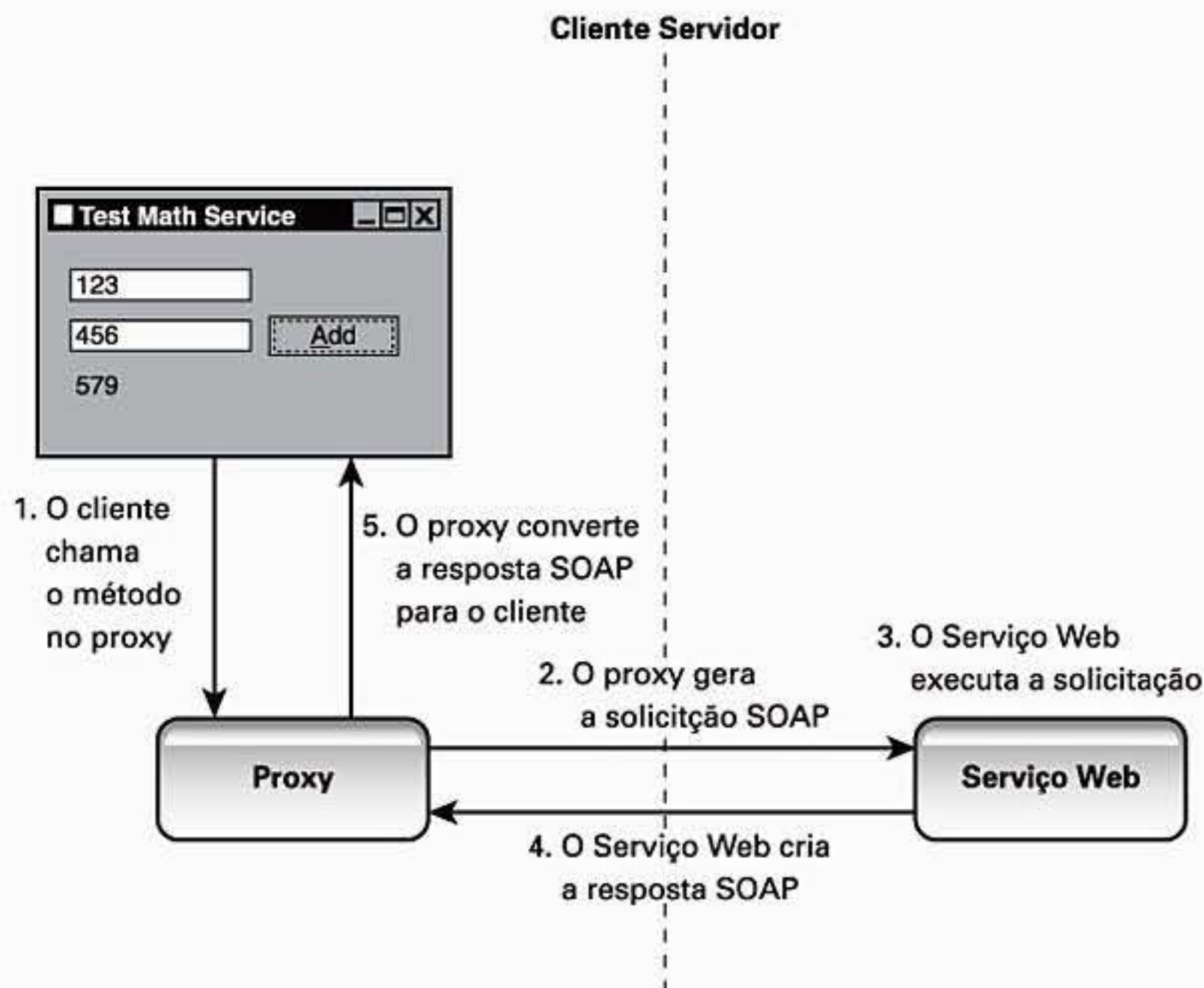
Embora a página de teste criada para o arquivo ASMX seja útil como demonstração, não é recomendável que os usuários a utilizem para trabalhar com seu Serviço Web. Em vez disso, você deve criar um cliente (um formulário Windows ou Web) que se comunicará com o Serviço Web. A comunicação entre esse cliente e o Serviço Web será executada com o auxílio de uma classe intermediária, também conhecida como proxy.

NOVO TERMO

Sob um ponto de vista genérico, o *proxy* é um substituto ou representante autorizado. É um intermediário. Para os Serviços Web, um *proxy* é um pequeno componente no lado do cliente que se parece com o Serviço Web para o programa cliente. Quando o programa cliente faz uma chamada no proxy, ele gera a solicitação SOAP apropriada para o Serviço Web e a transmite. Quando o Serviço Web reage com uma resposta SOAP, ele a converte novamente para o valor de retorno esperado pela função e envia isso para o cliente. A Figura 21.6 mostra esse processo.

FIGURA 21.6

Fluxo de informações entre o cliente, o proxy e o Serviço Web.



Criando o Projeto

Embora você pudesse gerar um cliente usando um formulário Windows ou Web (até mesmo outra biblioteca ou Serviço Web), nesta seção, examinaremos a criação de um Serviço Web cliente por meio de um formulário Windows. Esse será o exemplo de um dos usos possíveis para os Serviços Web, em que ele representará um meio de estender os clientes já existentes (ou futuros) de um microcomputador, como o Microsoft Office. Criar um cliente usando os formulários Web é semelhante.

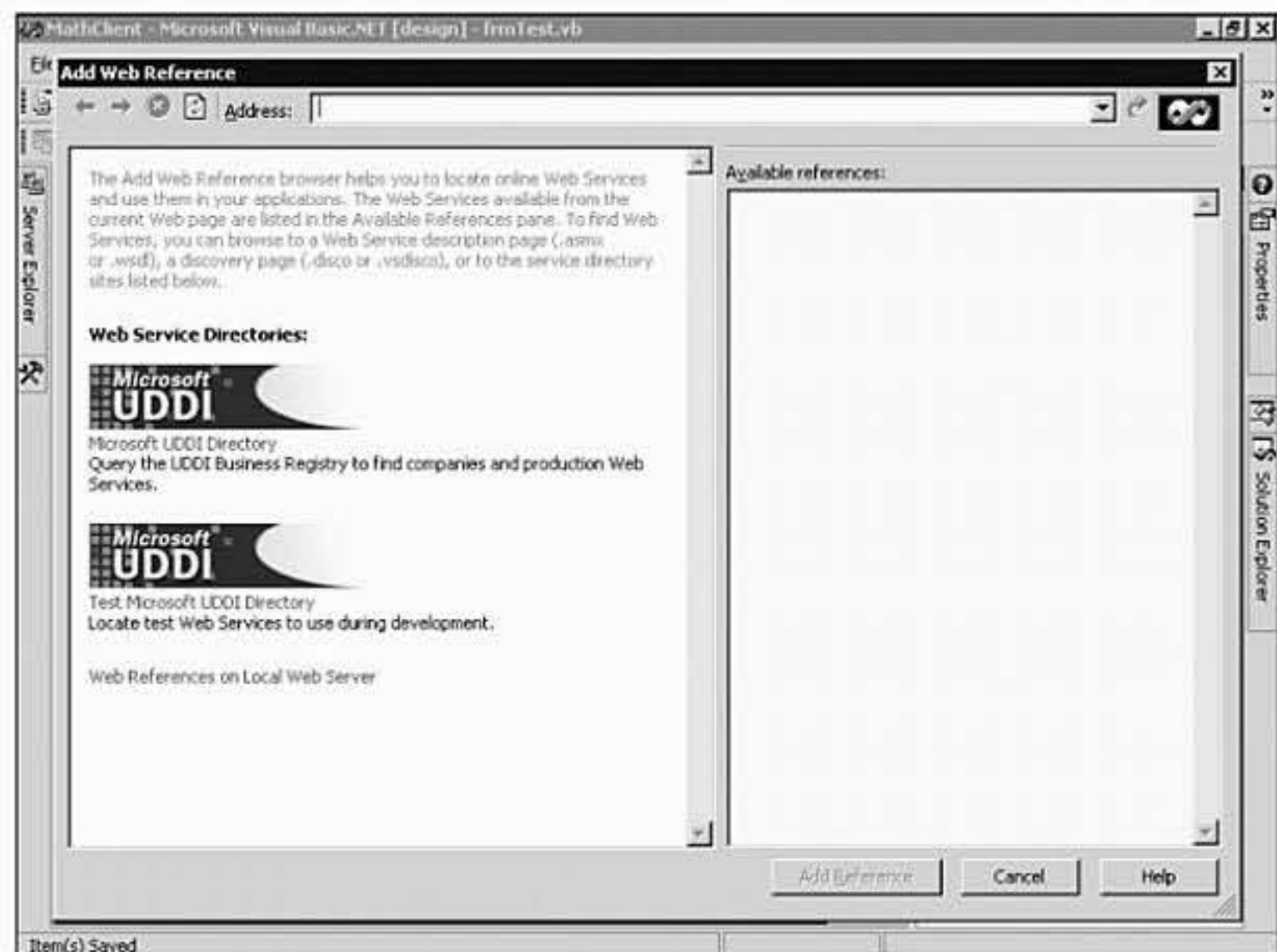
Crie o projeto de um novo aplicativo Windows e chame-o de MathClient. Você pode optar por criá-lo na mesma solução de MathService para manter os dois juntos, embora isso não seja necessário. Selecione File, New e Project. Na caixa de diálogo New Project, selecione Windows Application e digite **MathClient** no campo do nome. Certifique-se de que Add to Solution esteja selecionado e, em seguida, dê um clique em OK. O Visual Basic .NET deve criar um projeto com apenas um formulário. Antes de prosseguir, feche o gerador de formulários e renomeie o arquivo Form1.vb como **frmTest.vb**. A seguir, abra o formulário no modo de exibição do código e altere todas as referências de Form1 para **frmTest**. Depois, dê um clique com o botão direito do mouse no projeto MathClient e selecione Properties. Na página General, altere o Startup Object para **frmTest** e selecione OK. Para concluir, dê um clique novamente com o botão direito do mouse no projeto MathClient e selecione Set as Startup Project. Isso assegurará que esse projeto seja o primeiro a ser executado quando a solução for iniciada.

Adicionando o Código

Quando chamamos outro objeto no Visual Basic .NET, em geral temos de carregar uma referência a ele. Isso permite que o IDE conheça as várias propriedades e métodos do objeto. No entanto, você não pode fazer isso com um Serviço Web, porque ele não possui todas as informações expostas de um objeto local. Em vez disso, é preciso carregar uma referência da Web, que é um ponteiro para um arquivo WSDL. Lembre-se de que esse arquivo descreve todos os métodos de um Serviço Web e seus parâmetros.

Adicionar uma referência da Web é fácil: dê um clique com o botão direito do mouse no projeto MathClient e selecione Add Web Reference. Você deve ver a caixa de diálogo da Figura 21.7.

FIGURA 21.7
Adicionando uma referência da Web.



A caixa de diálogo solicita o URL para um arquivo DISCO ou para o local da WSDL do Serviço Web. Normalmente, eles seriam fornecidos pelo autor do serviço. No caso de MathService, o URL deve ser: `http://localhost/MathService/MathService.vsdisco`. Digite esse URL e dê um clique na seta de busca. Se funcionar da maneira ideal, depois de uma pausa, ele deve encontrar o Serviço Web, como mostra a Figura 21.8.

Depois de encontrada a referência da Web, dê um clique no botão Add Web Reference. Isso adicionará uma nova seção ao projeto (Web References), que deve ter três arquivos na seção Localhost: `Service1.wsdl`, `MathService.vsdisco` e `Reference.Map`. O compilador usará esses arquivos no tempo de compilação para adicionar um proxy ao programa cliente, que saiba como se comunicar com o Serviço Web.

Agora que seu programa cliente sabe como se comunicar com o Serviço Web, você pode adicionar o código para chamar o serviço. Para tanto, adicione duas caixas de texto, um título e um botão ao formulário. Configure as propriedades como na Tabela 21.2.

FIGURA 21.8
Localizando a referência da Web.

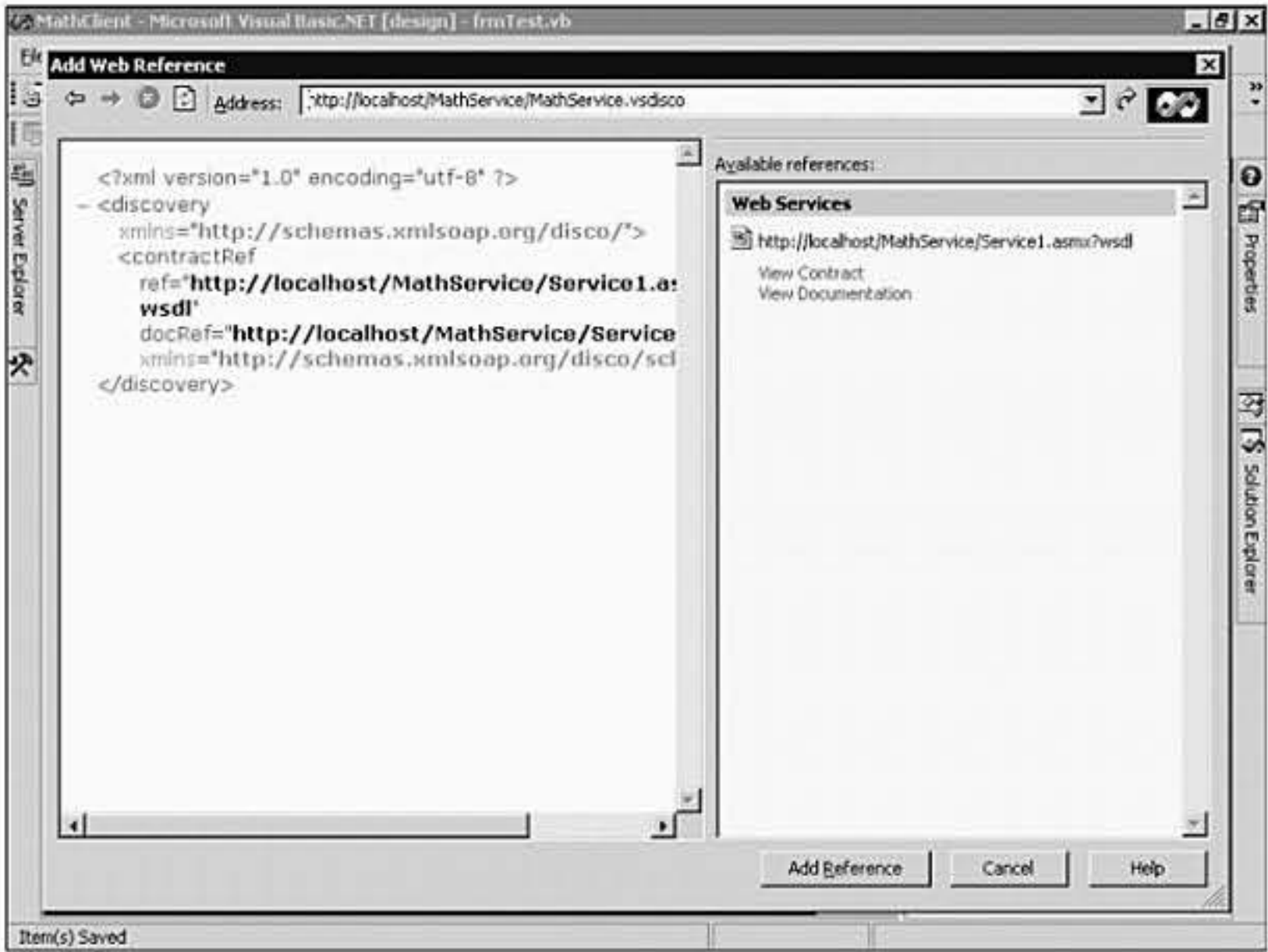


TABELA 21.2 Propriedades do Formulário MathClient

Controle	Propriedade	Valor
Form	Text	Test Math Service
	Size	248, 160
	BorderStyle	FixedSingle
TextBox	Name	TxtX
	Text	123
	Location	16, 24
	Size	100, 20
TextBox	Name	TxtY
	Text	456
	Location	16, 56
	Size	100, 20
Button	Name	CmdAdd
	Text	&Add

TABELA 21.2 Propriedades do Formulário MathClient (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Label	Location	128, 56
	Size	75, 23
	Name	LblResult
	Text	0
	Location	16,88
	Size	100, 23

Por fim, você está pronto para adicionar o código da Listagem 21.7, que chamará o Serviço Web. Dê um clique com o botão direito do mouse no botão Add para adicionar este código.

ENTRADA**LISTAGEM 21.7** Chamando o Serviço Web

```

1 Private Sub cmdAdd_Click(_
2     ByVal sender As System.Object, _
3     ByVal e As System.EventArgs) _
4     Handles cmdAdd.Click
5     Dim oMath As New localhost.Service1()
6     lblResult.Text = oMath.Add(CInt(txtX.Text), CInt(txtY.Text))
7 End Sub

```

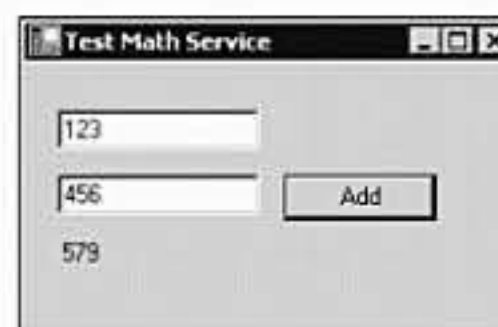
ANÁLISE

Algo que deve ficar imediatamente óbvio é que o código é muito semelhante ao usado para o acesso a objetos comuns. A única diferença entre o código das linhas 5 e 6 e o mesmo código que acessa um objeto local é que o nome completo do objeto tem como base o servidor que fornece o Serviço Web, nesse caso, `localhost.Service1`. Um item importante a observar é que o nome do Serviço Web será alterado com base no host do serviço.

O programa resultante deve poder encontrar e chamar o Serviço Web, estando ele na mesma máquina, como mostrei aqui, ou procurar pela Internet. A Figura 21.9 mostra o resultado.

FIGURA 21.9

Usando o cliente para chamar o Serviço Web.



Embora esse serviço que foi adicionado pareça simples (e é), como declarei anteriormente, ele apresenta toda a lógica de criação e uso de um Serviço Web:

- Cria um projeto para o Serviço Web (ou adiciona o Serviço Web em um projeto Web existente)
- Adiciona um ou mais métodos públicos com o atributo `<WebMethod()>`
- Adiciona uma referência Web a um aplicativo cliente, que aponte para o WSDL do Serviço Web
- Cria uma nova variável que aponte para o Serviço Web e executa um ou mais de seus métodos.

Um Serviço Web Mais Complexo

Para mostrar que realmente aprendeu tudo sobre a criação de Serviços Web, você criará um que faça mais do que apenas somar dois números. Será gerado um Serviço Web que poderá ser exposto a partir de um depósito, nesse caso, o Northwind. Ele permitirá que as tarefas a seguir sejam executadas:

- Recuperação de uma lista de categorias de produtos
- Recuperação de uma lista de produtos de uma única categoria
- Pedido de um produto

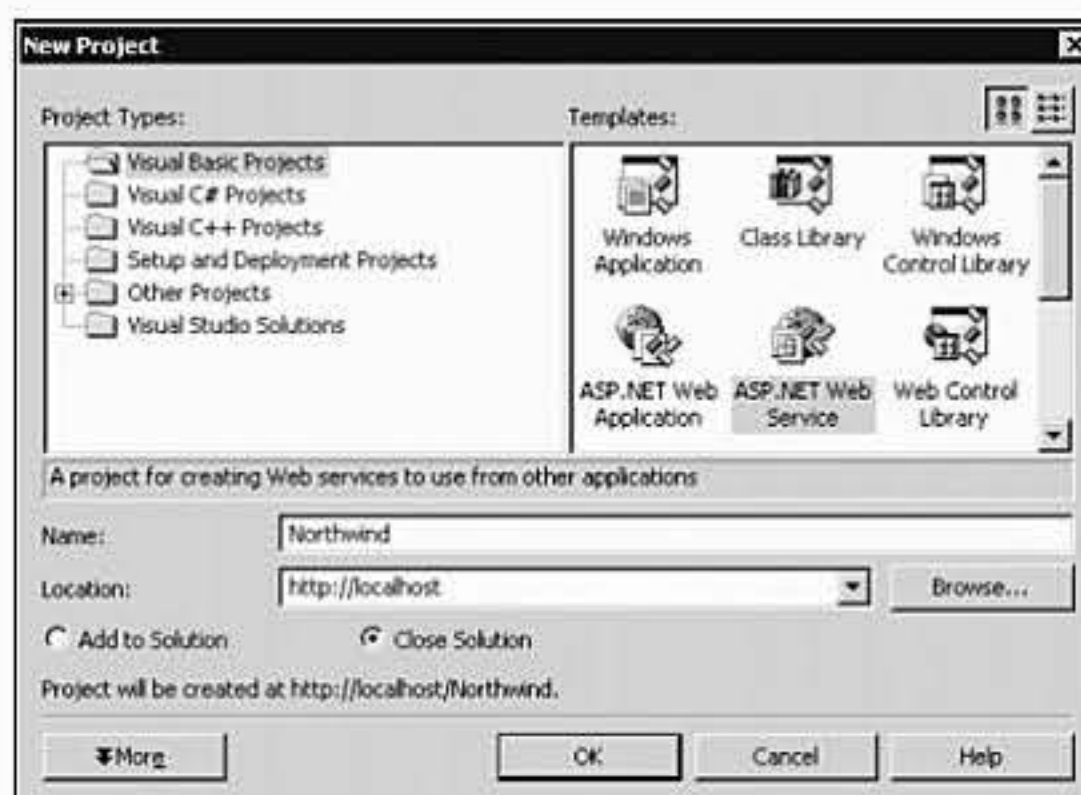
Uma diferença entre este Serviço Web e o do exemplo anterior é que o atual mostra que você pode recuperar objetos. Qualquer objeto interno ou criado pelo usuário pode ser enviado ou retornado de um Serviço Web. O objeto é convertido para XML de modo que passe pela rede e volte a ser um objeto para uso do cliente.

Criando o Serviço

Como no serviço simples, você começará criando o Serviço Web. No Visual Basic .NET crie um projeto de Serviços Web. Esse poderá ser chamado de Northwind. A Figura 21.10 mostra a caixa de diálogo New Project.

FIGURA 21.10

Criando o Serviço Web Northwind.



Feche o gerador e altere o nome do arquivo `Service1.asmx` para `Products.asmx`. Abra o arquivo `Products.asmx` no modo de exibição do código e altere a linha

```
Public Class Service1
```

para

```
Public Class ProductManager
```

A seguir, você adicionará os três métodos. O primeiro retornará um array contendo todas as categorias. O segundo retornará um `DataSet` contendo todas as informações sobre os produtos de uma categoria específica e o terceiro inserirá novos dados na tabela `Categories`. A Listagem 21.8 mostra esses três métodos. Para obter mais informações sobre o código de acesso a dados desses métodos, veja o Dia 12, “Acessando Dados com a Plataforma .NET”. Adicione esse código após a região marcada como ‘Web Services Designer Generated Code’, porém antes da linha `End Class`. Além disso, será preciso adicionar duas instruções `Imports`, uma para importar o espaço de nome `System.Data` e a outra para importar `System.Data.SqlClient`. Insira essas duas instruções logo após a linha `Imports System.Web.Services`.

CÓDIGO

LISTAGEM 21.8 O Código para o Serviço Web dos Produtos

```
1 Private Const DSN As String
2   "server localhost database north ind user id sa pass ord "
3
4   ebMethod    Public Function GetCategories    As String
5       Dim oCon As Ne  SqlConnection DSN
6       Dim sSQL As String
7       sSQL    "SELECT CategoryName FR M Categories"
8       Dim oCmd As Ne  SqlDataAdapter sSQL  oCon
9       Dim oDS As Ne  DataSet
10      Dim sReturn  As String
11      Dim I As Integer
12
13      oCmd.Fill oDS  "Categories"
14      ReDim sReturn oDS.Tables 0 .Ro s.Count
15      For I    0 To oDS.Tables 0 .Ro s.Count - 1
16          sReturn I    CStr oDS.Tables 0 .Ro s I .Item 0
17      Next
18      Return sReturn
19 End Function
20
21 ebMethod
22 Public Function GetProducts
23     yVal categoryName As String  As DataSet
24     Dim oCon As Ne  SqlConnection DSN
25     Dim sSQL As String
26     sSQL    "SELECT ProductID  ProductName  nitPrice "
```


CÓDIGO**LISTAGEM 21.8** O Código para o Serviço Web dos Produtos
(continuação)

```

27      "QuantityPer nit Discontinued CategoryName "
28      "FR M Categories INNER    IN Products "
29      " N Categories.CategoryID  Products.CategoryID "
30      " HERE Discontinued 0 AND "
31      " CategoryName LI E      categoryName " "
32      Dim oCmd As Ne SqlDataAdapter sSQL oCon
33      Dim ods As Ne DataSet
34
35      oCmd.Fill ods "products"
36      Return ods
37 End Function
38
39  ebMethod
40 Public Function InsertCategory
41     yVal categoryName As String
42     yVal description As String As oolean
43     Dim oCon As Ne SqlConnection DSN
44     Dim sSQL As String
45     sSQL "INSERT INT Categories"
46         " CategoryName Description "
47         "VAL ES " categoryName " "
48         " " description " "
49     Dim oCmd As Ne SqlCommand sSQL oCon
50     Try
51         oCon. pen
52         oCmd.ExecuteNonQuery
53         Return True
54     Catch
55         Return False
56     End Try
57 End Function

```

ANÁLISE

O primeiro dos três métodos, `GetCategories`, retorna um array de strings contendo os nomes de cada categoria de produtos. Ele poderia ter retornado um `DataSet` com a mesma facilidade (na verdade, teria sido mais fácil), no entanto, fizemos assim para mostrar que arrays também podem ser retornados.

A string de conexão que será usada para o acesso ao banco de dados foi criada como uma constante nas linhas 1 e 2. Isso terá de ser alterado se seu banco de dados estiver armazenado em outro local ou se forem necessárias credenciais específicas para a conexão com ele. Os objetos apropriados para o acesso aos dados foram criados nas linhas 5 a 11. Aqui empregamos os métodos de acesso ao banco de dados SQL. Portanto, será necessário incluir o espaço de nome

`System.Data.SqlClient` nesse arquivo. Além disso, se você for acessar um banco de dados que não seja SQL, deve utilizar os objetos de conexão do ADO, que estão armazenados no espaço de nome `System.Data.OleDb`. Observe que recuperamos apenas uma coluna da tabela `Categories` (linha 7).

A linha 13 recupera os dados e preenche o `DataSet`. As linhas 14 a 17 extraem as informações do `DataSet` para usá-las no preenchimento do array. Por fim, o array é retornado para a função da chamada na linha 18.

O segundo método, `GetProducts`, é mais simples do que o primeiro. Nele, é retornado um `DataSet` preenchido com todos os produtos ainda válidos da categoria solicitada. A parte mais complexa do método é o código das linhas 26 a 31, em que a instrução SQL é criada para recuperar a lista. O ideal seria que isso fosse mantido como uma visualização ou um procedimento armazenado no banco de dados. A string final teria a aparência descrita a seguir:

```
SELECT ProductID, ProductName, UnitPrice, ➡  
QuantityPerUnit, Discontinued, CategoryName ➡  
FROM Categories INNER JOIN Products ➡  
ON Categories.CategoryID = Products.CategoryID ➡  
WHERE (Discontinued = 0) AND (CategoryName LIKE 'Bev%'
```

A instrução SQL resultante retornaria todos os campos solicitados em que o produto ainda fosse válido e o nome da categoria começasse com `Bev`.

O último método, `InsertCategory`, demonstra que você também pode solicitar que o Serviço Web faça algo diferente de uma recuperação. Nesse caso, será inserida uma nova categoria no banco de dados.

Começamos de maneira semelhante aos dois métodos anteriores, pela criação de uma conexão com o banco de dados e escrevendo a instrução SQL. Nesse caso, a instrução final criada nas linhas 45 a 48 se parecerá com a demonstrada a seguir:

```
INSERT INTO Categories(CategoryName, Description) ➡  
VALUES('Algum produto', 'Outro produto que estamos vendendo')
```

Uma diferença nesse método é que em vez de criar um adaptador `SqlDataAdapter`, você gera um comando `SqlCommand`. Os comandos `SqlCommand`, como aprendemos no Dia 12, são mais adequados para inserir, excluir e alterar dados, do que para recuperá-los. Outro ponto que vale a pena ressaltar é o bloco `Try...Catch` das linhas 50 a 56. Nesse caso, tentamos executar a instrução SQL. Se ela for bem-sucedida, o método retornará `True`. Se uma exceção ocorrer, por alguma razão, ele retornará `False`. Normalmente, não ocorreria uma exceção nesse método, a menos que o banco de dados não estivesse em execução ou se não houvesse espaço em disco.

Testando o Serviço Web

Exatamente como foi feito no Serviço Web simples, é possível usar a funcionalidade interna para testar o serviço Web que acabamos de criar. Compile o projeto, abra um navegador Web e pesquise em `http://localhost/northwinds/products.asmx`. Você deve ver a página de teste gerada na Figura 21.11.

É bom testar cada um dos três métodos antes de continuar. As Figuras 21.12, 21.13 e 21.14 mostram o resultado de cada método em ação. O segundo foi testado com a string `Bev`, e o terceiro com as strings `Algum produto` e `Outro produto` que estamos vendendo.

FIGURA 21.11
Página de teste do Serviço Web Northwind.

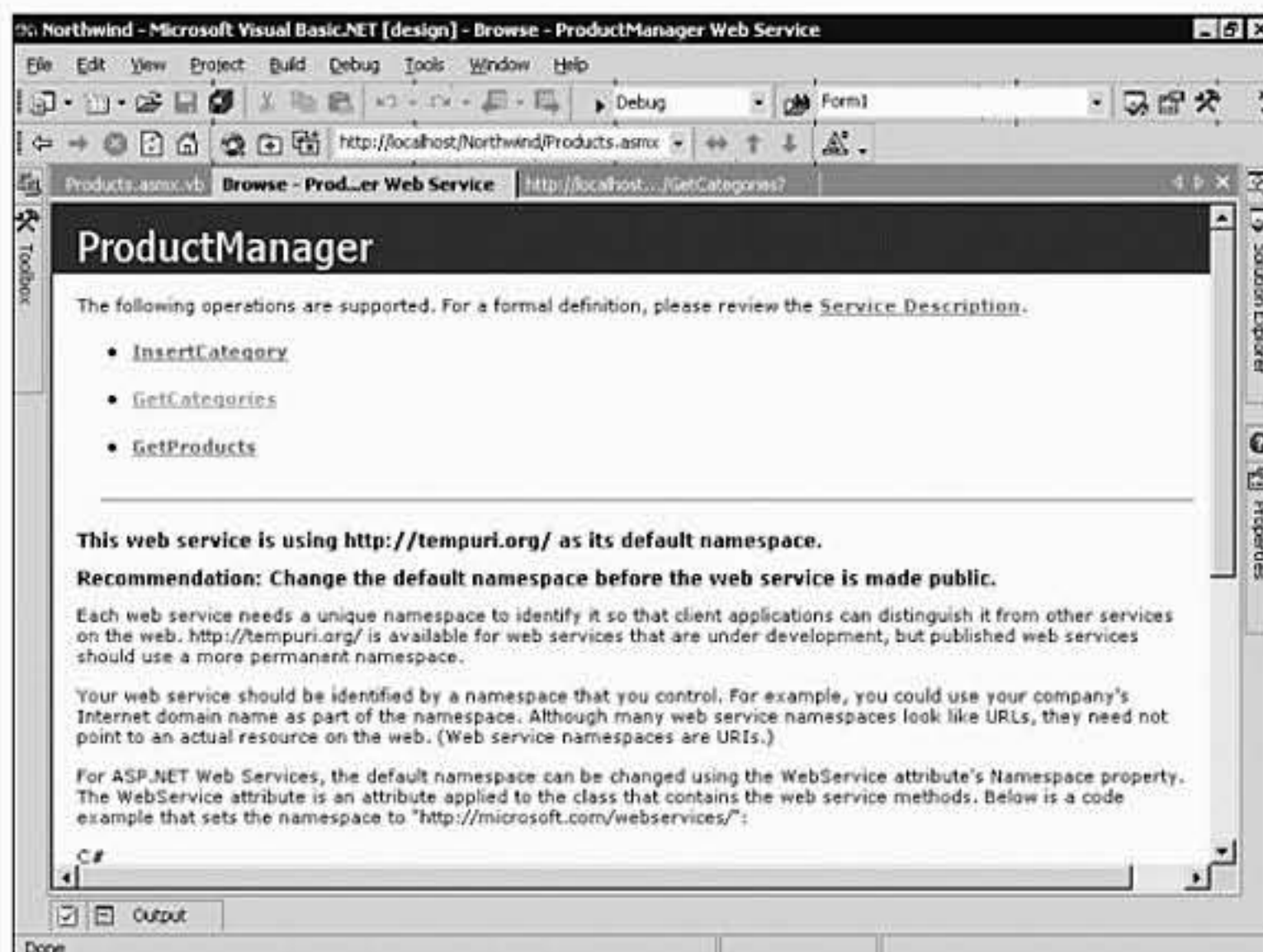
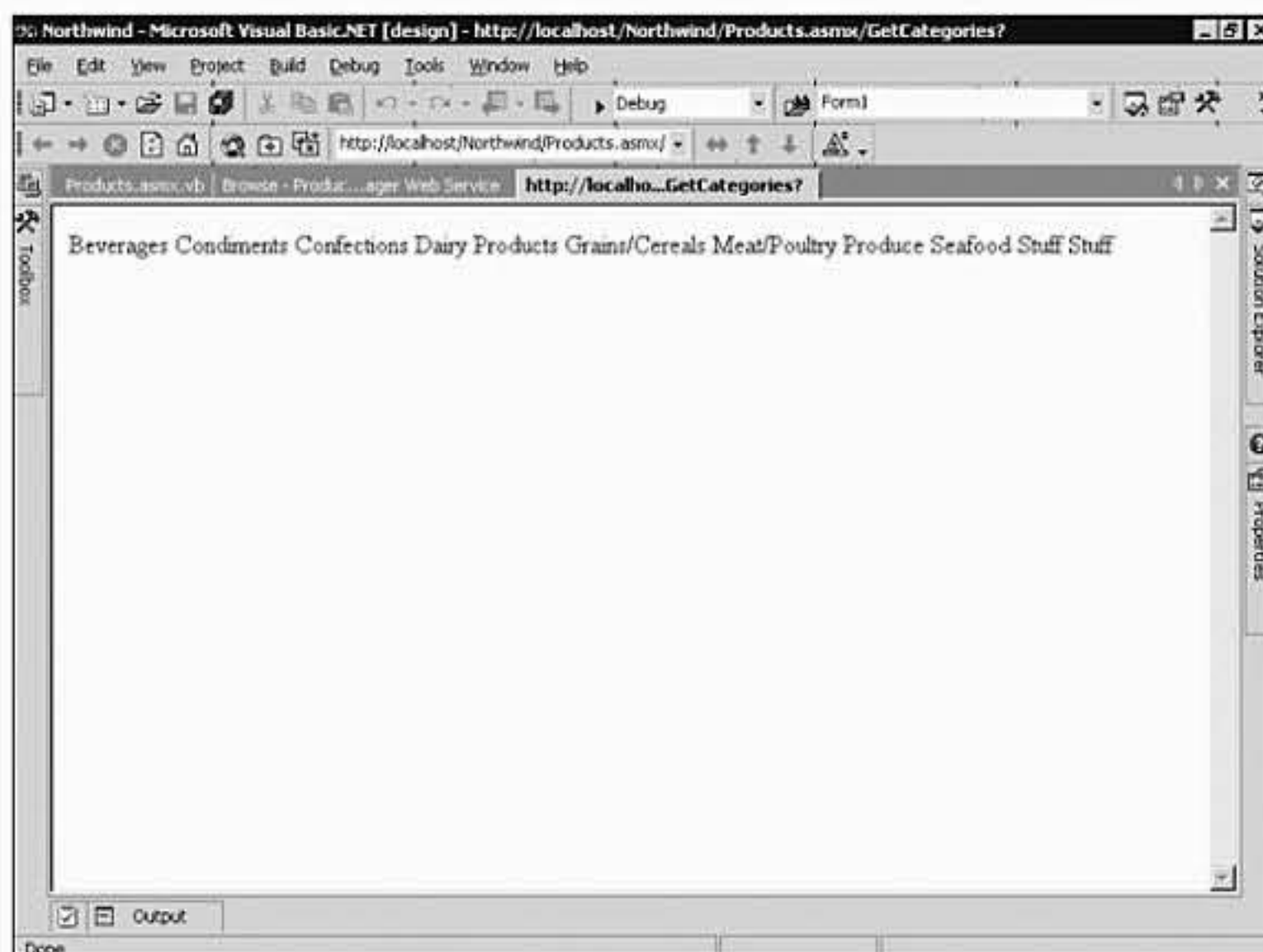


FIGURA 21.12
Testando GetCategories.



Criando o Cliente

Como no caso anterior, agora você está pronto para criar seu Serviço Web. Dessa vez, no entanto, criará um cliente com base nos formulários Web. Ele poderia ser um site usando um Serviço Web para que esse lhe fornecesse conteúdo, porém empregando seu próprio layout. Como alternativa, poderíamos representar esse cliente que usa formulários Web como um aplicativo de intranet.

FIGURA 21.13
Testando GetProducts.

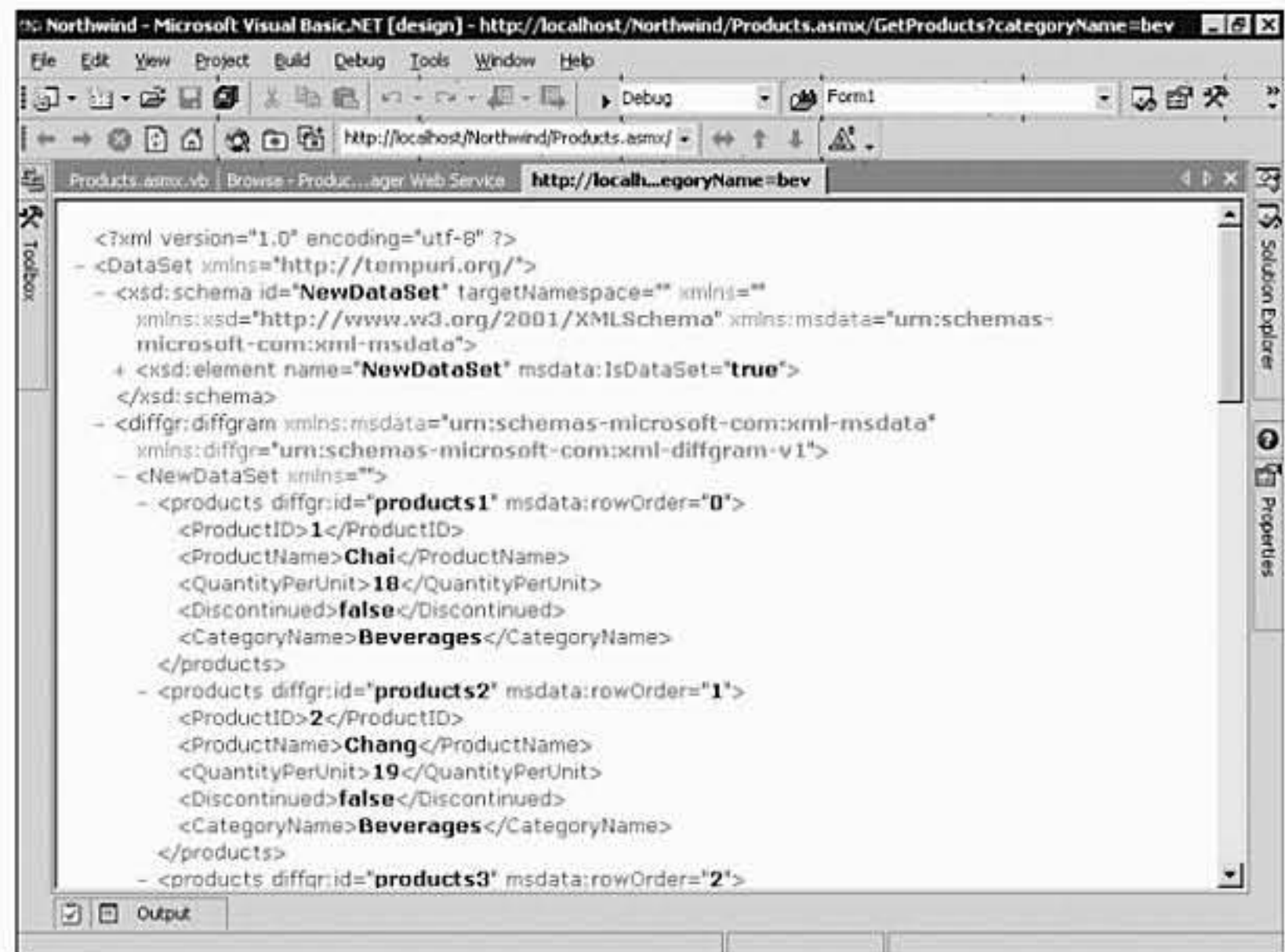
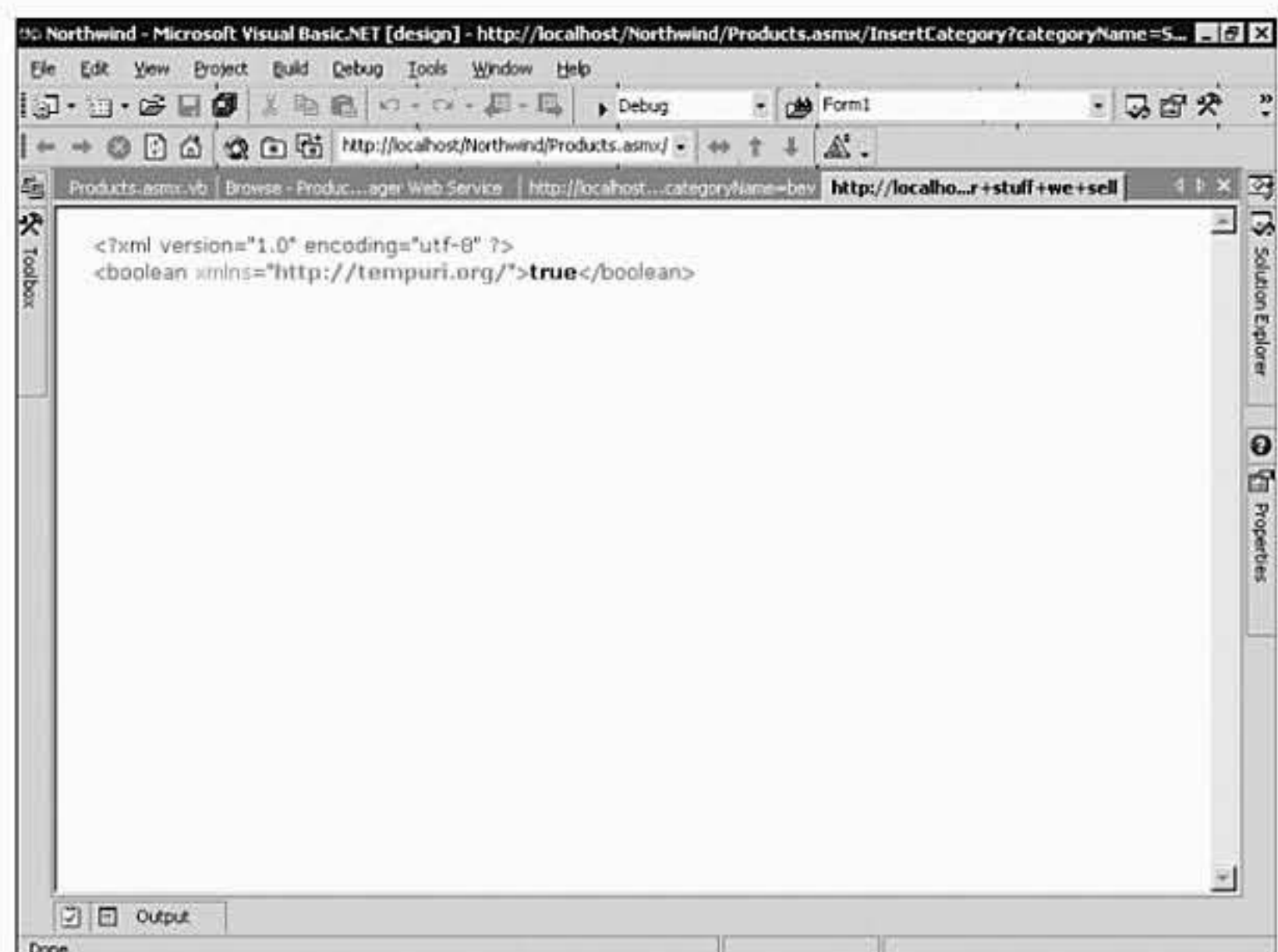


FIGURA 21.14
*Testando
InsertCategory.*



Inicie a criação do cliente adicionando um novo projeto de aplicativo Web. Esse você chamará de `NWCClient` (que provém de cliente Northwind). Dê um clique com o botão direito do mouse no nome do projeto e selecione `Set as Startup Project`. Feche o gerador e renomeie o arquivo `WebForm1.aspx` como **`Products.aspx`**. Para concluir, abra mais uma vez o gerador e gere uma interface bem simples. Faremos assim principalmente porque tenho poucas habilidades gráficas, mas também para limitar a quantidade de código adicional que pode complicar o exemplo.

As propriedades dos controles da página estão descritas na Tabela 21.3.

TABELA 21.3 Propriedades do Formulário Web `NWCClient`

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Document	<code>title</code>	Northwind Products
Panel	<code>(ID)</code>	<code>pn1NewCategory</code>
	<code>Height</code>	181px
	<code>Width</code>	368px
Label	<code>(ID)</code>	<code>1b1NewCategory</code>
	<code>Text</code>	New Category
Label	<code>(ID)</code>	<code>1b1Name</code>
	<code>Text</code>	Name:
TextBox	<code>(ID)</code>	<code>txtName</code>
	<code>Height</code>	24px
	<code>Width</code>	213px
Label	<code>(ID)</code>	<code>1b1Description</code>
	<code>Text</code>	Description:
TextBox	<code>(ID)</code>	<code>txtDescription</code>
	<code>Height</code>	41px
	<code>Width</code>	222px
	<code>TextMode</code>	MultiLine
Button	<code>(ID)</code>	<code>cmdAdd</code>
	<code>Text</code>	Add
Label	<code>(ID)</code>	<code>1b1Result</code>
	<code>Height</code>	19px
	<code>Width</code>	329px
	<code>BackColor</code>	Silver
Label	<code>(ID)</code>	<code>1b1Categories</code>

TABELA 21.3 Propriedades do Formulário Web NWClient (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
DropDownList	Text	Categories
	(ID)	cboCategory
	Height	22px
	Width	202px
DataGrid	AutoPostBack	True
	(ID)	dbgProducts
	Height	265px
	Width	438px

ANÁLISE

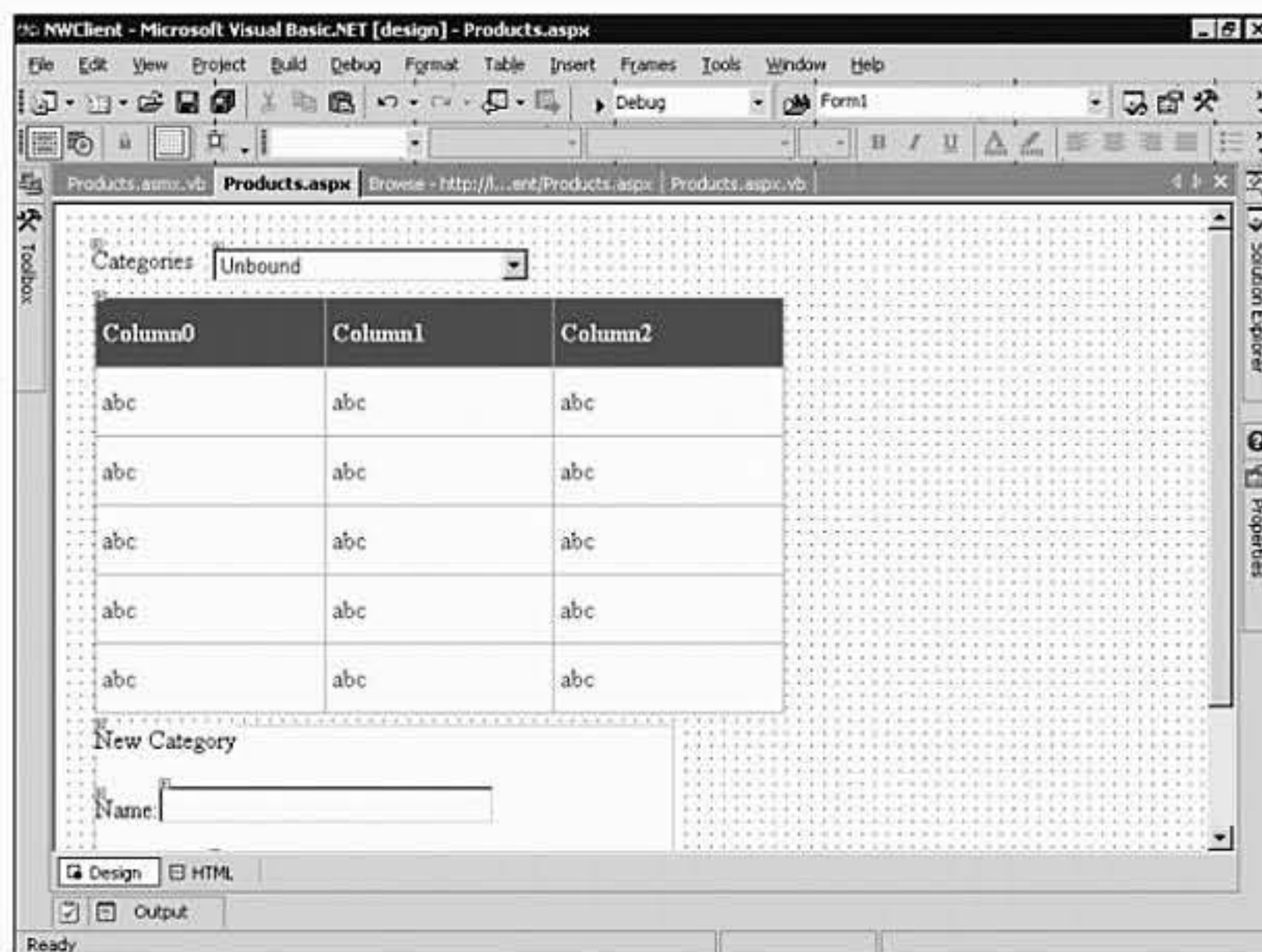
Você adicionará ao formulário os controles ComboBox (DropDownList) e DataGrid. Eles serão usados para exibir o resultado das chamadas a GetCategories e GetProducts. Além disso, haverá um painel contendo duas caixas de texto, um botão e um título. Esses serão utilizados pelo método InsertCategory.

Uma propriedade de DropDownList que vale a pena ressaltar é AutoPostBack. Como você aprendeu no Dia 10, “Construindo a Interface com o Usuário com os Formulários da Web”, AutoPostBack faz com que o formulário seja retornado ao servidor imediatamente, em vez de aguardar que um botão Submit seja pressionado. Em nosso exemplo, o formulário será enviado sempre que o usuário alterar o conteúdo de DropDownList.

Depois de DropDownList, definimos o controle DataGrid. Você também pode usar o recurso AutoFormat do gerador do DataGrid para tornar sua aparência mais adequada do que a usada no padrão.

A Figura 21.15 mostra qual deve ser a aparência da interface com o usuário no Visual Basic .NET.

FIGURA 21.15
Cliente Northwind.



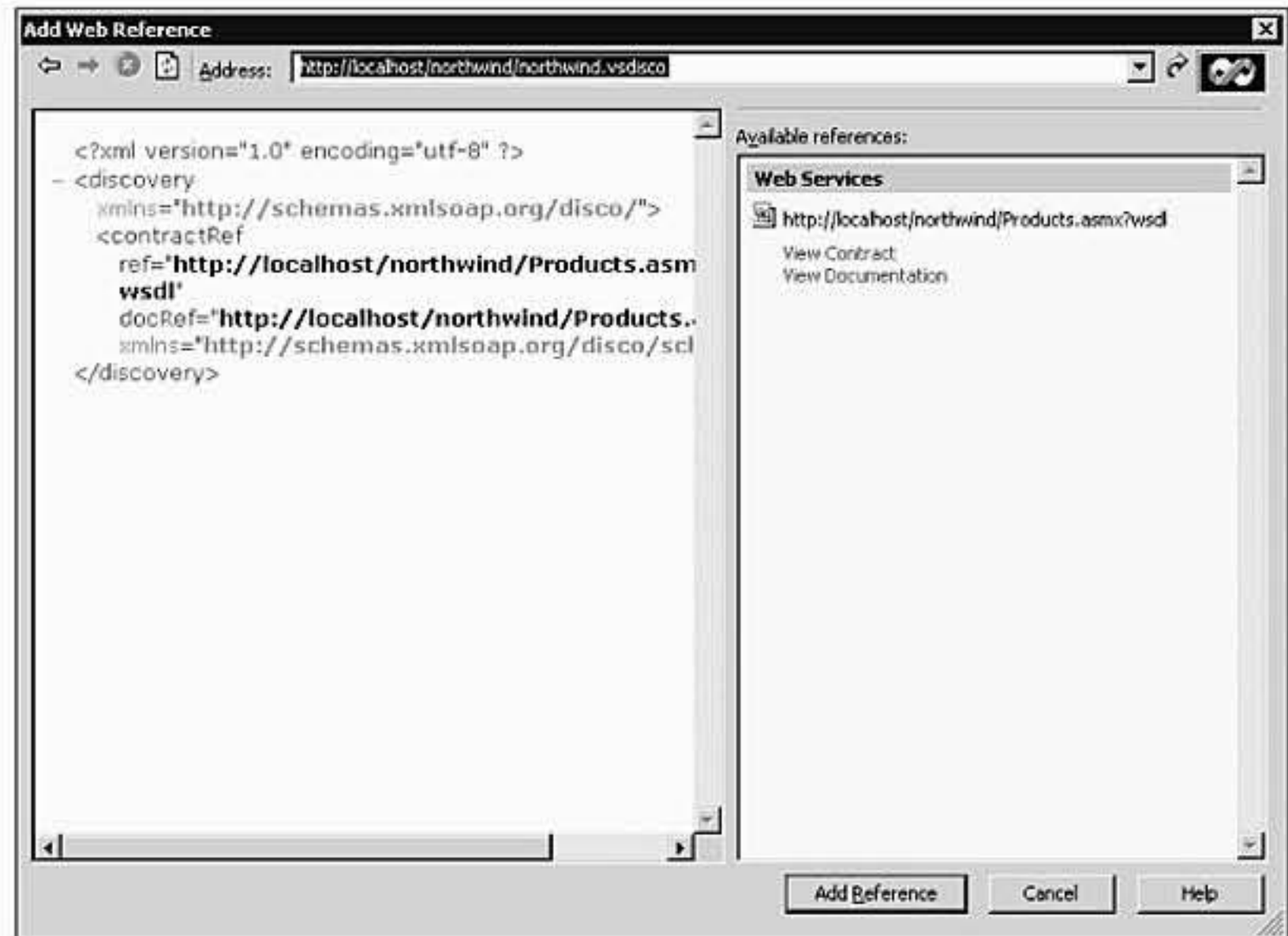
Adicionando o Código

Estamos quase lá. Tudo que precisamos fazer é conectar algumas rotinas. Queremos que a caixa de combinação Categories seja preenchida quando o usuário examinar a página pela primeira vez, e que a grade apresente os resultados da categoria. Além disso, queremos que a grade seja atualizada sempre que o usuário selecionar uma nova categoria. Para concluir, é nossa intenção adicionar uma nova categoria quando o usuário der um clique no botão Add.

Antes que você possa adicionar o código, deve acrescentar uma referência da Web ao Serviço Web. Dê um clique com o botão direito do mouse no projeto NWClient e selecione Add Web Reference. O arquivo DISCO criado para o Serviço Web é <http://localhost/northwind/northwind.vsdisco>. A Figura 21.16 mostra o resultado da consulta a esse arquivo.

Em vez de começar preenchendo a caixa de combinação, inicie com a grade. Você já decidiu que quer atualizar a grade sempre que o conteúdo da caixa de combinação for alterado. O evento apropriado ao qual o código deve ser adicionado é `SelectedIndexChanged`. Dê um clique duplo na caixa de combinação e atualize o manipulador de evento como vemos na Listagem 21.9.

FIGURA 21.16
Localizando o serviço Northwind.

**CÓDIGO****LISTAGEM 21.9** Atualizando a Grade

```

1 Private Sub cboCategory_SelectedIndexChanged
2     ByVal sender As System.Object
3     ByVal e As System.EventArgs
4     Handles cboCategory.SelectedIndexChanged
5     Dim oProducts As New localhost.ProductManager
6     Dim oDS As DataSet
7     oDS = oProducts.GetProducts cboCategory.SelectedItem.Text
8     dbgProducts.DataSource = oDS.Tables(0).DefaultView
9     dbgProducts.DataBind
10 End Sub

```

ANÁLISE

Esse método começa instanciando um novo ProductManager (o Serviço Web). Em seguida, recupera a lista de produtos (como um DataSet) do Serviço Web, usando o método GetProducts (linha 7). Para concluir, nas linhas 8 e 9, ele conecta DataSet ao controle DataGridView e faz com que esse seja atualizado.

Quando a grade estiver sendo atualizada automaticamente, será a hora de nos preocuparmos com a caixa de combinação. Ela deve ser preenchida com a lista de categorias quando o usuário carregar a página pela primeira vez. O evento apropriado para fazer com que isso aconteça é WebForm1_Load. A Listagem 21.10 mostra esse método.

CÓDIGO**LISTAGEM 21.10** Carregando as Categorias

```
11 Private Sub Page_Load
12     ByVal sender As System.Object
13     ByVal e As System.EventArgs Handles MyBase.Load
14     If Not IsPostBack Then
15         Dim oProducts As New localhost.ProductManager
16         Dim sCategories As String
17         sCategories = oProducts.GetCategories
18         cboCategory.DataSource = sCategories
19         cboCategory.DataBind
20         cboCategory.SelectedIndex = 0
21     End If
22 End Sub
```

ANÁLISE

Como você aprendeu no Dia 10, a propriedade `IsPostBack` é usada para determinar se essa é a primeira vez que uma página da Web é visualizada ou se ela está retornando. Em geral, só queremos executar tarefas como preencher controles uma vez, portanto adicionamos o código à seção `If Not IsPostBack`. No evento que carrega a página, inicialmente criamos uma nova instância do Serviço Web (linha 15). A seguir, chamamos o método `GetCategories` (linha 17), que retorna um array de strings.

Uma prática interessante é usada nas linhas 18 e 19 no evento que carrega a página. Os controles não só podem ser conectados a `DataSets` como também a `Arrays` (e a conjuntos). Portanto, você pode conectar um array de categorias diretamente a uma caixa de combinação.

O código da linha 20 merece alguma explicação. Quando a caixa de combinação é preenchida, o índice selecionado não é considerado como alterado. Portanto, chamamos explicitamente o manipulador de evento para a caixa de combinação a fim de assegurar que a grade seja preenchida quando o usuário visitar a página pela primeira vez.

A Figura 21.17 mostra o resultado que obteremos ao carregar a página da Web, enquanto a Figura 21.18 ilustra o que será exibido se mudarmos a categoria.

FIGURA 21.17
Visualizando o cliente Northwind.

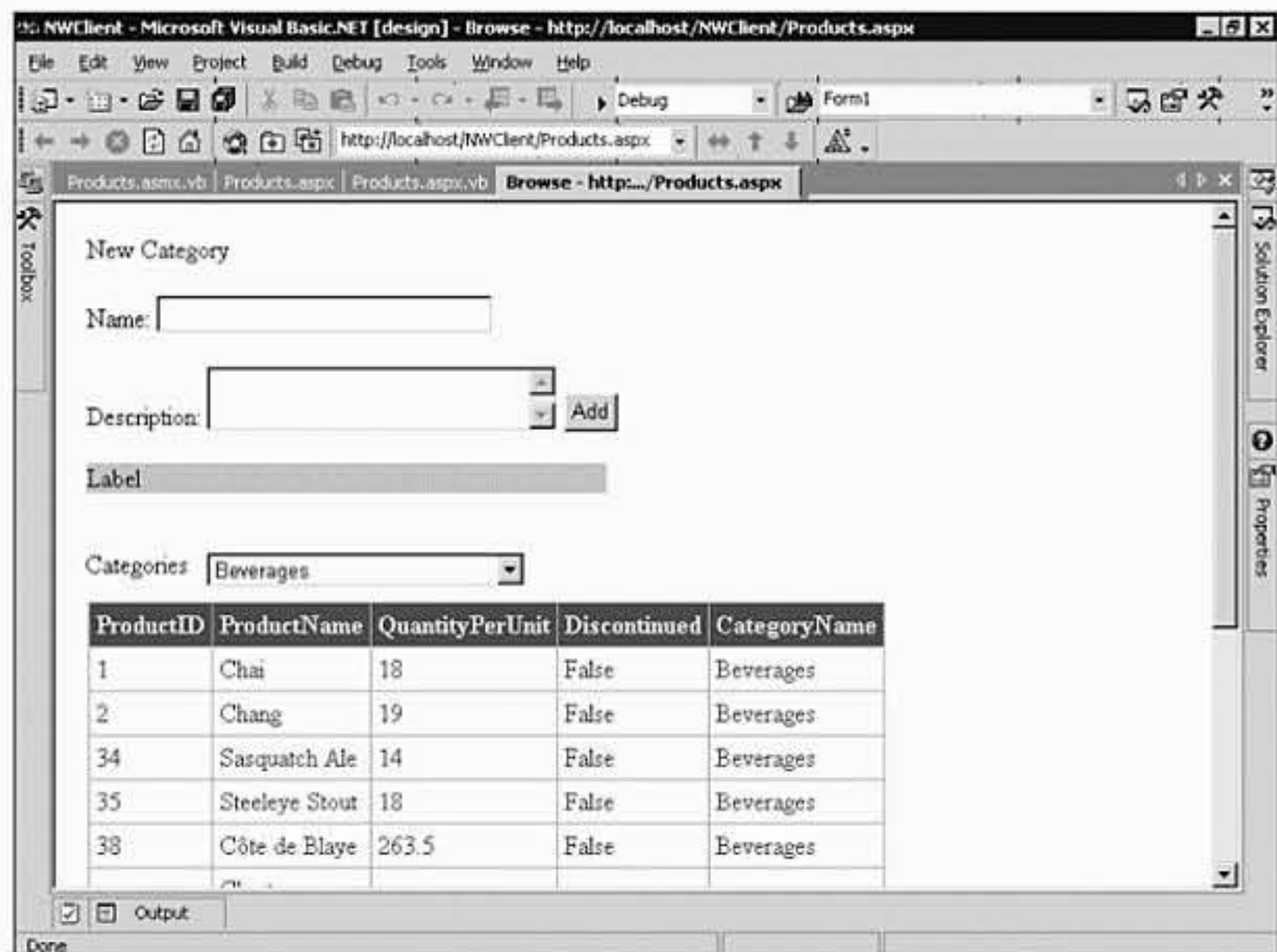
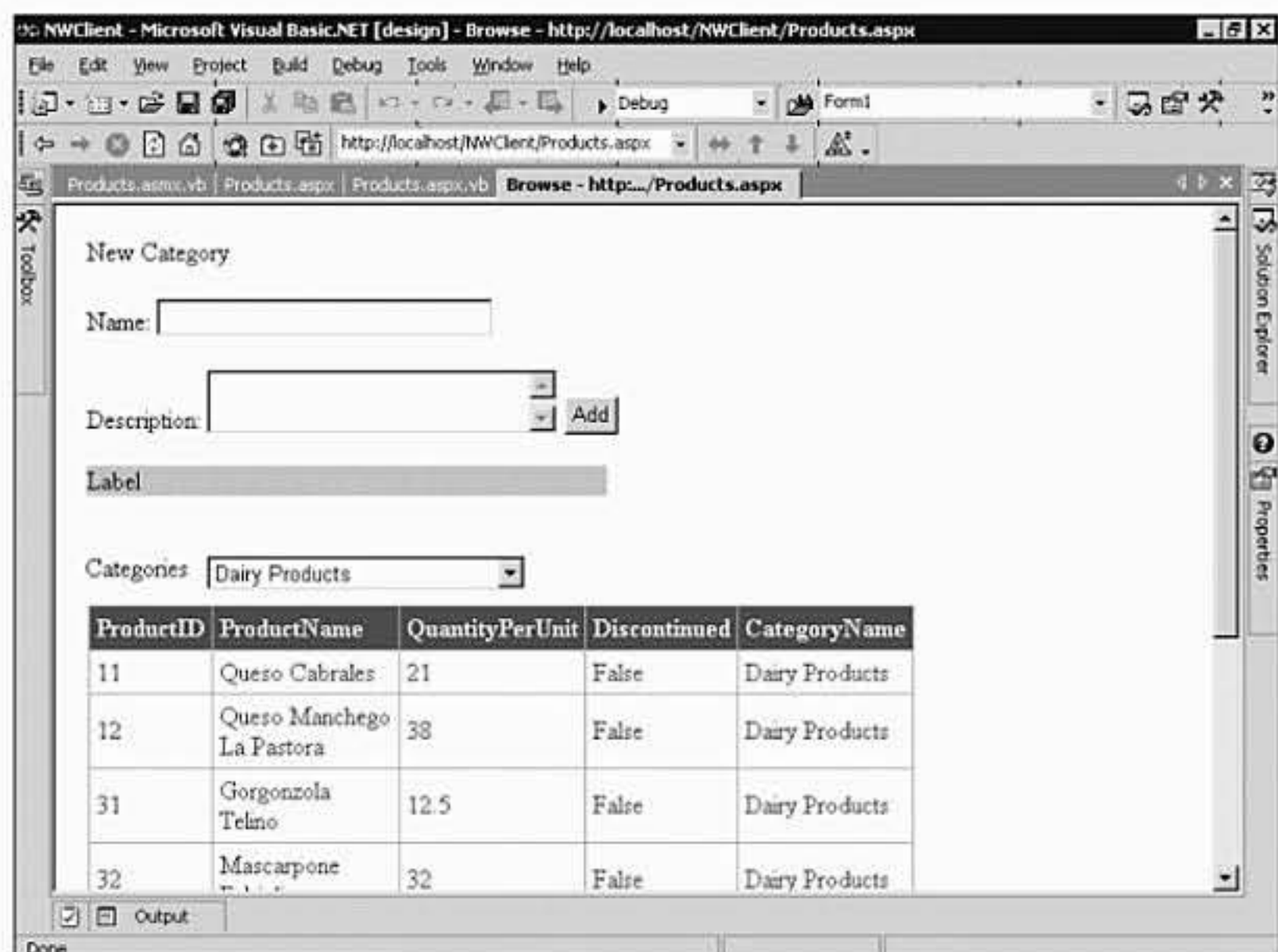


FIGURA 21.18
Alterando a categoria selecionada.



Para concluir, você adicionará o código que fará o botão Add acrescentar uma nova categoria ao banco de dados. Dê um clique duplo no botão Add para abrir a janela do código. Adicione o código da Listagem 21.11 ao procedimento.

CÓDIGO

LISTAGEM 21.11 O Botão Add

```

23 Private Sub cmdAdd Click
24     yVal sender As System. b ect
25     yVal e As System.EventArgs
26     Handles cmdAdd.Click
27     Dim oProducts As Ne localhost.ProductManager
28     If oProducts.InsertCategory txtName.Text txtDescription.Text Then
29         lblResult.Text "Adicionada nova categoria: " txtName.Text
30     Else
31         lblResult.Text "Não foi poss vel adicionar nova categoria"
32     End If
33 End Sub

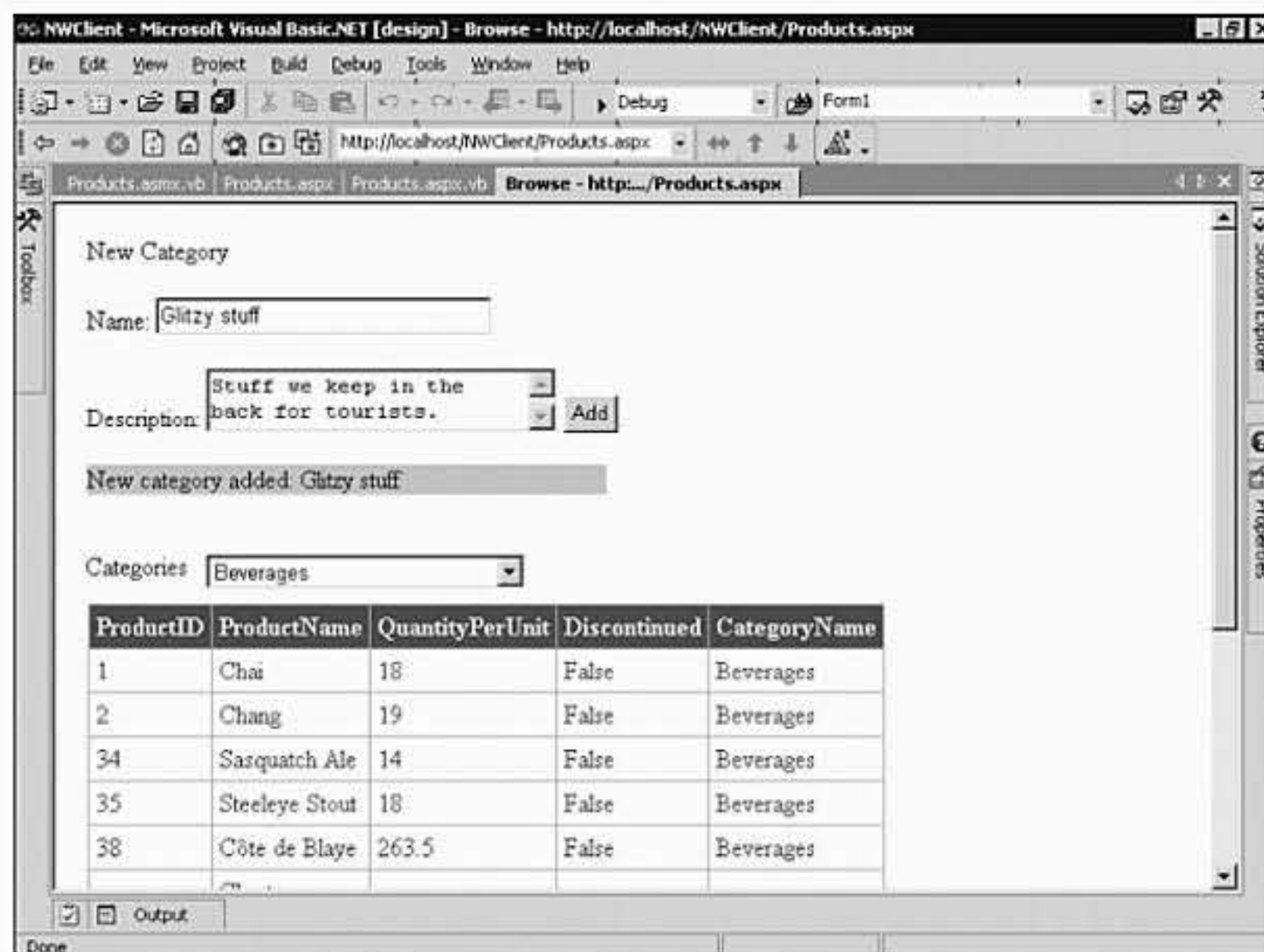
```

ANÁLISE

Felizmente, não há nada muito complexo aqui. Você criou uma nova instância do Serviço Web e chamou o método `InsertCategory`, passando o conteúdo dos dois campos. Quando ele retornar (`True` ou `False`), o conteúdo de `Label` será alterado com uma mensagem de que a operação foi bem-sucedida ou falhou. A Figura 21.19 mostra o resultado da execução desse método.

FIGURA 21.19

Dando um clique no botão Add.



Resumo

Os Serviços Web são uma nova maneira de acessar programas e componentes por meio de uma rede ou da Internet. Eles são úteis tanto na Web quanto em aplicativos tradicionais de microcom-

putadores. Fornecem simplicidade no desenvolvimento e utilização de clientes Web com objetos em várias plataformas. Um benefício essencial é que eles podem ser escritos em qualquer linguagem de programação e proporcionam interoperabilidade com todos os sistemas operacionais. Podemos esperar que os Serviços Web se tornem mais relevantes conforme forem mais difundidos.

P&R

P Posso proteger um Serviço Web para controlar quem tem permissão para acessá-lo?

R Sim, os Serviços Web dão suporte integral ao uso do HTTPS (e do SSL) para o controle de acesso.

P Os Serviços Web só são adequados para soluções que usem os produtos da Microsoft?

R Não, os Serviços Web podem ser usados para a comunicação com qualquer sistema operacional. Contanto que o cliente seja capaz de interpretar a solicitação SOAP (e criar uma resposta SOAP), ele poderá ser utilizado na interoperação com um cliente ou servidor escrito com o Visual Basic .NET.

P O que aconteceria se eu tentasse acessar um Serviço Web, e ele estivesse inativo?

R Infelizmente, há pouco que se possa fazer nessas situações. Em geral, é preciso adicionar um código ao cliente para que ele reaja de modo apropriado a esse problema. O cliente pode falhar ou armazenar em cache e reutilizar a última resposta quando adequado.

Workshop

O Workshop foi planejado para ajudá-lo a antecipar possíveis dúvidas, revisar o que já aprendeu e começar a pensar em como colocar seu conhecimento em prática. As respostas do teste estão no Apêndice A, “Respostas dos Testes/Exercícios”.

Teste

1. Qual a finalidade do arquivo DISCO?
2. Qual o objetivo de um proxy?
3. Por que o SOAP é importante para os Serviços Web?

Exercícios

Tente adicionar outros métodos aos dois Serviços Web que criamos. Por exemplo, acrescente os métodos Subtract, Multiply e Divide a MathService. Atualize seu cliente para que trabalhe com os novos métodos.

SEMANA 3

Resumo

A Semana 3 abordou um conjunto pesado de tópicos, mencionando vários recursos com os quais será preciso gastar ainda mais tempo no futuro. Para começar, você aprendeu como criar seus próprios objetos na plataforma .NET, construindo blocos de código que podem se tornar autônomos e ser reutilizados por outros projetos quando necessário. O Dia 16 o conduziu por algumas das mais avançadas técnicas e tecnologias para a criação de aplicativos com base no Windows por meio da plataforma .NET, como os menus e os controles de divisão.

O Dia 17 forneceu uma abordagem de duas áreas do .NET Framework freqüentemente necessárias: a biblioteca de figuras, em que mostramos como manipular imagens e a aparência de seus aplicativos, e os recursos de manipulação de arquivos da plataforma .NET, com detalhes de como criar, abrir e alterar esses arquivos. No Dia 18, você aprendeu a tornar a manutenção e o suporte de seu aplicativo tão fáceis quanto possível, incluindo a documentação de seu código e o uso do controle do código-fonte para proteger seu precioso trabalho. O Dia 19 deu prosseguimento ao tópico distribuição, incluindo várias considerações sobre a redistribuição dos arquivos .NET necessários para que seu programa funcione corretamente. Criar programas de instalação, aprender as opções de instalação remota e fornecer a documentação do usuário, todos esses itens fizeram parte do Dia 19.

Os Dias 20 e 21 abordaram duas tecnologias que são a base do conceito .NET: a XML e os Serviços Web. Você aprendeu como a plataforma .NET fornece classes para leitura, criação, alteração e compilação de documentos XML, e que a XML pode ser usada em seus aplicativos. O Dia 21 mostrou os Serviços Web, um dos principais recursos da plataforma .NET, e aprendemos como eles permitem a exposição da funcionalidade de seu sistema na Web por meio dos padrões HTTP e XML. O desenvolvimento de Serviços Web e o uso deles em seu próprio aplicativo também foram discutidos, embora a abordagem completa desse tópico esteja além do escopo deste livro.

Para concluir, o projeto de bônus da Semana 3, que está disponível na Web (veja a Introdução), mostrou-lhe como os conceitos abordados nos capítulos anteriores, como a utilização de arquivos, as figuras, a XML e até algumas tarefas avançadas com os formulários Windows, podem ser usados na criação de um programa completo. Nesse caso, criamos um jogo, mas essa foi apenas uma maneira mais agradável de aprender as técnicas que podem ser aplicadas a quase todo tipo de sistema que for preciso gerar.

Com a conclusão deste livro, você também chegou ao fim dos 21 dias de Visual Basic .NET e já obteve uma compreensão geral da linguagem, da estrutura subjacente e de como pode usar as duas para começar a criar seus próprios sistemas. No desenvolvimento, o aprendizado é um processo contínuo que nunca termina realmente, mas passamos por uma introdução bem profunda a uma linguagem fantástica. É recomendável sair do nível introdutório de estudo por algum tempo e se concentrar em desenvolver vários aplicativos (de exemplo ou reais, dependendo de seu trabalho atual) para fixar todas as informações que abordamos nestas três semanas. Certifique-se de estar codificando na plataforma .NET o máximo que puder; não há maneira melhor de ganhar confiança e uma compreensão adicional em uma linguagem e uma tecnologia novas.

PÁGINA EM BRANCO

APÊNDICE A

Respostas dos Testes/Exercícios

Respostas do Dia 1

Teste

1. Primeiro a Microsoft produziu um compilador para a linguagem BASIC original, chamando esse produto de Microsoft BASIC. A versão seguinte desse compilador, que antecedeu o Visual BASIC, foi o Quick BASIC, ou QBASIC.
2. Todas as linguagens .NET compartilham esses recursos porque o Common Language Runtime os fornece. É por meio dessa funcionalidade de tempo de execução que todas essas linguagens trabalham com a plataforma .NET.
3. Esse processo é chamado de compilação.
4. Se você não declarar explicitamente um arquivo de saída, usando `/out:<nomedoarquivo>`, o compilador empregará o nome do arquivo-fonte no lugar do executável. Nesse caso, a saída seria `MySourceCode.exe`.

Exercícios

1. Para gerar um arquivo executável específico, você só precisa de `/out:<nomedoarquivo>` no compilador da linha de comando. A fim de executar essa tarefa, acesse o console (prompt de comando). Navegue (usando o comando `cd`) até a pasta que contém seus arquivos de trabalho relacionados aos exemplos anteriores, especificamente `step3.vb`.

Agora, execute o compilador digitando **`vbc step3.vb t:exe out: hat S.exe`** e pressionando Enter. Isso compilará `step3`, mas salvará o resultado compilado em um arquivo chamado `WhatOS.exe`.

2. Há muitas propriedades diferentes disponíveis na parte Environment do .NET Framework, portanto existem várias maneiras pelas quais você poderia ter respondido a essa pergunta. No final, entretanto, todas as respostas se pareceriam com a solução a seguir, que exibe a versão do .NET Framework:

```
Public Class FrameworkVersion
    Shared Sub Main()
        System.Console.WriteLine(System.Environment.Version.ToString())
    End Sub
End Class
```

Esse código produz o resultado a seguir (esse resultado é gerado quando a versão Beta 2 da plataforma .NET é usada; você obterá resultados diferentes se empregar a versão oficial):

CÓDIGO/ RESULTADO

```
C:\T V \C1 vbc FrameworkVersion.vb t:exe
Microsoft (R) Visual Basic.NET Compiler version 7.00.9254
for Microsoft (R).NET CLR version 1.00.2914.16
Copyright (C)Microsoft Corp 2001.All rights reserved.

C:\T V \C1 FrameworkVersion
1.0.2914.16
```

Como alternativa, você pode querer exibir o caminho para o diretório System ou um dos muitos outros valores, todos seguindo o mesmo padrão básico. Um exemplo do uso do valor `SystemDirectory` é mostrado a seguir:

```
Public Class SysDir
    Shared Sub Main()
        System.Console.WriteLine(System.Environment.SystemDirectory)
    End Sub
End Class
```

CÓDIGO/ RESULTADO

```
C:\T V \C1 vbc SysDir.vb t:exe
Microsoft (R) Visual Basic.NET Compiler version 7.00.9254
for Microsoft (R).NET CLR version 1.00.2914.16
Copyright (C) Microsoft Corp 2001.All rights reserved.
```


C:\TV\CL SysDir
C:\WINNT\System32

Respostas do Dia 2

Teste

A

1. Você usaria o Solution Explorer, que mostra todos os projetos abertos e os arquivos que existem neles. Podemos tornar essa janela visível pressionando a combinação de teclas CTRL+R.
2. Por padrão, os projetos são colocados sob a opção My Documents folder\Visual Studio Projects do usuário atual e em uma nova pasta com o nome do projeto. Quando você criar um projeto novo, esse caminho será exibido na caixa de diálogo New Project, podendo ser alterado quando for conveniente.
3. Escolha um ícone na caixa de diálogo Project Properties, que você pode acessar dando um clique com o botão direito do mouse sobre o projeto no Solution Explorer e selecionando Properties no menu suspenso. Também é possível escolher um arquivo .ico (de ícone) em Common Properties\Build.
4. Na janela Command, digitar >cmd e pressionar Return o colocará no modo Command. Digitar immed e pressionar Return o conduzirá novamente para o modo Immediate.

Exercício

Com a introdução de uma chamada `MessageBox.Show` no evento `TextChanged` de uma caixa de texto, uma mensagem será exibida sempre que o conteúdo dessa caixa for alterado. Embora esse seja um local útil para códigos que validem a entrada de texto do usuário, exibir uma mensagem dentro desse evento logo se tornaria incômodo!

Respostas do Dia 3

Teste

1. Você deve usar o tamanho e o tipo de variável mais apropriados entre os tipos disponíveis.
2. A e C estão corretas. Quando uma função é chamada, os parâmetros têm de ser colocados entre parênteses. Além disso, a função retornará um valor, portanto em geral é preciso que se faça algo com o resultado. Atribua-o a outra variável ou exiba-o.
3. Se uma variável for declarada com a palavra-chave `Private`, estará disponível para todo o módulo ou classe onde foi especificada.

Exercícios

Uma solução possível para o exercício do programa de cálculo de pagamentos seria

```
1  Module Payment
2
3      Private dblAnnualInterest As Double = 0
4      Private iYears As Integer = 0
5      Private dblLoanAmount As Double = 0
6      Private dblMonthlyDeposit As Double = 0
7
8      Sub Main()
9          Dim dblResult As Double
10
11          'armazena os valores informados
12          GetInputValues()
13
14          'calcula
15          dblResult = CalculatePayment(dblLoanAmount, _
16                                     dblAnnualInterest, _
17                                     iYears)
18          'proceda a exibição do resultado
19          DisplayResults(dblResult)
20
21      End Sub
22
23      Private Function CalculatePayment(Byval LoanAmount As Double, _
24                                     ByVal AnnualInterest As Double, _
25                                     ByVal Years As Integer) As Double
26          'divida por 1200 para torná-lo um percentual mensal
27          Dim dblMonthlyInterest As Double = CDec(AnnualInterest / 1200)
28          Dim iMonths As Integer = Years * 12
29          Dim dblTemp As Double
30          Dim dblReturn As Double
31
32
33          'precisaremos desse valor em alguns locais
34          dblTemp = CDec(((1 + dblMonthlyInterest) ^ iMonths))
35          dblReturn = LoanAmount * _
36                     (dblMonthlyInterest * dblTemp / (dblTemp - 1))
37          Return dblReturn
38
39      End Function
40
41      Private Sub GetInputValues()
42          Console.WriteLine()
43          dblLoanAmount = CDec(GetValue("Loan Amount: "))
```



```
44         dblAnnualInterest = _
45             CDb1(GetValue("Annual Interest (e.g. for 5%, enter 5): "))
46         iYears = CInt(GetValue("Years of loan: "))
47         Console.WriteLine()
48     End Sub
49
50     Private Function GetValue(ByVal Prompt As String) As String
51
52         Console.Write(Prompt)
53         Return Console.ReadLine
54
55     End Function
56
57     Private Sub DisplayResults(ByVal Result As Double)
58         Console.WriteLine()
59
60         Console.WriteLine("If you borrow {0:c}, ", dblLoanAmount)
61         Console.WriteLine("at {0}% interest.", dblAnnualInterest)
62         Console.WriteLine("for {0}years", iYears)
63
64         Console.WriteLine()
65         Console.WriteLine("Your monthly payment would be: {0:c}", Result)
66
67     End Sub
68
69 End Module
```

A

Respostas do Dia 4

Teste

1. O laço For é mais adequado para esse tipo de solução, embora qualquer dos laços pudessem ser usados.
2. Embora os laços possam ser usados para simular uns aos outros, o laço Do é mais flexível devido a seu suporte às cláusulas While e Until e às posições de entrada e saída onde poderá ser inserida a expressão condicional.
3. O bloco interno é o código que será executado em cada iteração do laço, portanto, o desempenho obtido ou a quantidade de iterações do laço apresentará um resultado melhor dentro do laço.
4. Considerando-se CalculatedTotalNetWorth() um pouco mais complexa do que o código executado em dtCurrentDate.Hour(), seria melhor que você invertesse os dois lados

da expressão. Isso assegurará que, se a primeira expressão, `dtCurrentDate.Hour() > 12`, for falsa, a rotina maior e mais demorada não seja nem mesmo executada.

Exercício

1. Há muitas maneiras diferentes pelas quais você poderia escrever o programa Number-Guesser, mas o resultado geral seria semelhante ao descrito a seguir:

```
Public Class GamePlayer
```

```
    Shared Sub Main()
```

```
        Dim iUpperBound As Integer
```

```
        Dim iLowerBound As Integer
```

```
        Dim iCurrentGuess As Integer
```

```
        Dim sInput As String
```

```
        Dim sGuessStatus As String
```

```
        System.Console.WriteLine("Please enter the lower bound: ")
```

```
        sInput = System.Console.ReadLine()
```

```
        iLowerBound = CInt(sInput)
```

```
        System.Console.WriteLine("Please enter the upper bound: ")
```

```
        sInput = System.Console.ReadLine()
```

```
        iUpperBound = CInt(sInput)
```

```
        'Solicite ao usuário que escolha um número
```

```
        System.Console.WriteLine("Pick a number between " & iLowerBound & _  
            " and " & iUpperBound & ", and remember it!")
```

```
        System.Console.WriteLine("Hit Return To Continue")
```

```
        System.Console.ReadLine()
```

```
        iCurrentGuess = 0
```

```
        sGuessStatus = " "
```

```
        Do Until sGuessStatus = "="
```

```
            iCurrentGuess = GetMyGuess(iLowerBound, iUpperBound)
```

```
            System.Console.WriteLine("My Guess Is: " & iCurrentGuess)
```

```
            System.Console.WriteLine("How did I do ? ")
```

```
            sGuessStatus = System.Console.ReadLine()
```

```
            sGuessStatus = sGuessStatus.ToUpper()
```

```
            Select Case sGuessStatus
```

```
                Case "H"
```

```
                    iUpperBound = iCurrentGuess - 1
```

```
                Case "L"
```

```
                    iLowerBound = iCurrentGuess + 1
```

```
            End Select
```



```
        Loop
    End Sub
    Shared Function GetMyGuess(iUpper As Integer, iLower As Integer)
        ↪As Integer
        If iUpper = iLower Then
            GetMyGuess = iLower
        Else
            GetMyGuess = iLower + ((iUpper - iLower - 1)\2)
        End If
    End Function
End Class
```

A

Respostas do Dia 5

Teste

1. Há várias vantagens, porém a mais importante é que não é necessário distribuir nada antes que o aplicativo possa ser usado ou para atualizá-lo. Outro benefício essencial está relacionado aos aplicativos com uma interface Web poderem ser acessados em plataformas que não sejam Windows.
2. Ainda assim será um sistema de uma camada. Os níveis diferentes dos aplicativos são baseados nos locais onde o código e/ou o processamento são executados. Nesse exemplo, nenhum tipo de processamento é manipulado pelo servidor; ele apenas contém o arquivo que será usado pelo aplicativo cliente. Toda a parte ligada à lógica, ao processamento e ao código é tratada somente no sistema cliente.
3. O requisito básico é o .NET Framework, que deve ser instalado em qualquer sistema que venha a executar um código .NET, inclusive um servidor que processe um site ASP.NET na Web.

Respostas do Dia 6

Teste

1. As janelas Local, Watch, Immediate e Quickwatch e a janela suspensa Watch.

Exercícios

1. A Listagem 6.11 mostra uma solução possível.

CÓDIGO**LISTAGEM 6.11** Tabela de Multiplicação Concluída

```
1 Imports System
2 Imports Microsoft.VisualBasic.ControlChars
3
4 Module modTable
5
6 Sub Main()
7     Dim iLow,iHigh As Integer
8     Dim sInput As String
9
10    'Permita múltiplas execuções na geração da tabela
11    Do
12        'solicite valores
13        Console.WriteLine("Low value (maximum of 20, 0 to end): ")
14        sInput = Console.ReadLine()
15        iLow = CInt(sInput)
16
17        If iLow < 0 Then
18            Console.WriteLine("High value (maximum of 20): ")
19            sInput = Console.ReadLine()
20            iHigh = CInt(sInput)
21
22            OutputTable(iLow, iHigh)
23        End If
24        Loop Until iLow = 0
25        Console.WriteLine("Press ENTER to continue")
26        Console.ReadLine()
27    End Sub
28
29 Private Sub OutputTable(ByVal MinValue As Integer,_
30     ByVal MaxValue As Integer)
31     Dim iCount, iCount2 As Integer
32
33     Console.WriteLine()
34     Console.WriteLine("Multiplication Table  ({0}to {1})",_
35         MinValue, MaxValue)
36
37     'escreva o cabeçalho
38     For iCount = MinValue To MaxValue
39         Console.Write(Tab & CStr(iCount))
40     Next
41     Console.WriteLine()
42
43     'Crie cada linha da tabela
44     For iCount = MinValue To MaxValue
```


CÓDIGO**LISTAGEM 6.11** Tabela de Multiplicação Concluída (*continuação*)

```
45      For iCount2 = MinValue To MaxValue
46          Console.Write(Tab & CStr(iCount * iCount2))
47      Next
48      Console.WriteLine()
49  Next
50 End Sub
51
52 End Module
```

A

Respostas do Dia 7

Teste

1. Se um método inserido em uma classe for marcado com `MustOverride`, então, a classe deve ser marcada com `MustInherit`. Isso acontece porque o primeiro requisito, que todas as classes clientes devem implementar seu próprio código para esse método, força a inclusão da palavra-chave `MustInherit` já que a classe básica não sobrepõe o método.
2. A palavra-chave `MyBase` permite que sejam chamados os membros da classe-pai ou da classe básica. Você pode usar `MyBase.New()` para chamar o construtor-padrão de sua classe básica.
3. Dois dos construtores sobrepostos são efetivamente os mesmos para o compilador do Visual Basic e, portanto, não são permitidos.

```
Public Sub New(ByVal sColor As String)
    m_dtManufactured = System.DateTime.Now()
    m_sColor = sColor
End Sub
```

```
Public Sub New(ByVal sName As String)
End Sub
```

Nesse caso, já que os dois procedimentos compartilham o mesmo nome e os parâmetros, o compilador do Visual Basic os considerará iguais, portanto, não permitirá que ambos existam. O fato de que um argumento é chamado `sColor` e o outro `sName` é irrelevante para a finalidade dessa questão.

Exercícios

Há várias maneiras que você poderia usar para criar esse exemplo, mas é apresentada a seguir uma implementação possível (veja a Figura 7.4):

Namespace Biology

Public Class Animal

Public Overridable Property Name() As String

Get

End Get

Set(ByVal Value As String)

End Set

End Property

Public Overridable Property NumberOfLimbs() As Integer

Get

End Get

Set(ByVal Value As Integer)

End Set

End Property

Public Overridable Property Cor() As String

Get

End Get

Set(ByVal Value As String)

End Set

End Property

Public Overridable Property Weight() As Single

Get

End Get

Set(ByVal Value As Single)

End Set

End Property

Public Overridable Property AirBreather() As Boolean

Get

End Get

Set(ByVal Value As Boolean)

End Set

End Property

End Class

Public Class Reptile

Inherits Animal

End Class

Public Class Mammal

Inherits Animal

Public Overrides Property AirBreather() As Boolean

Get

Return True

End Get

Set(ByVal Value As Boolean)

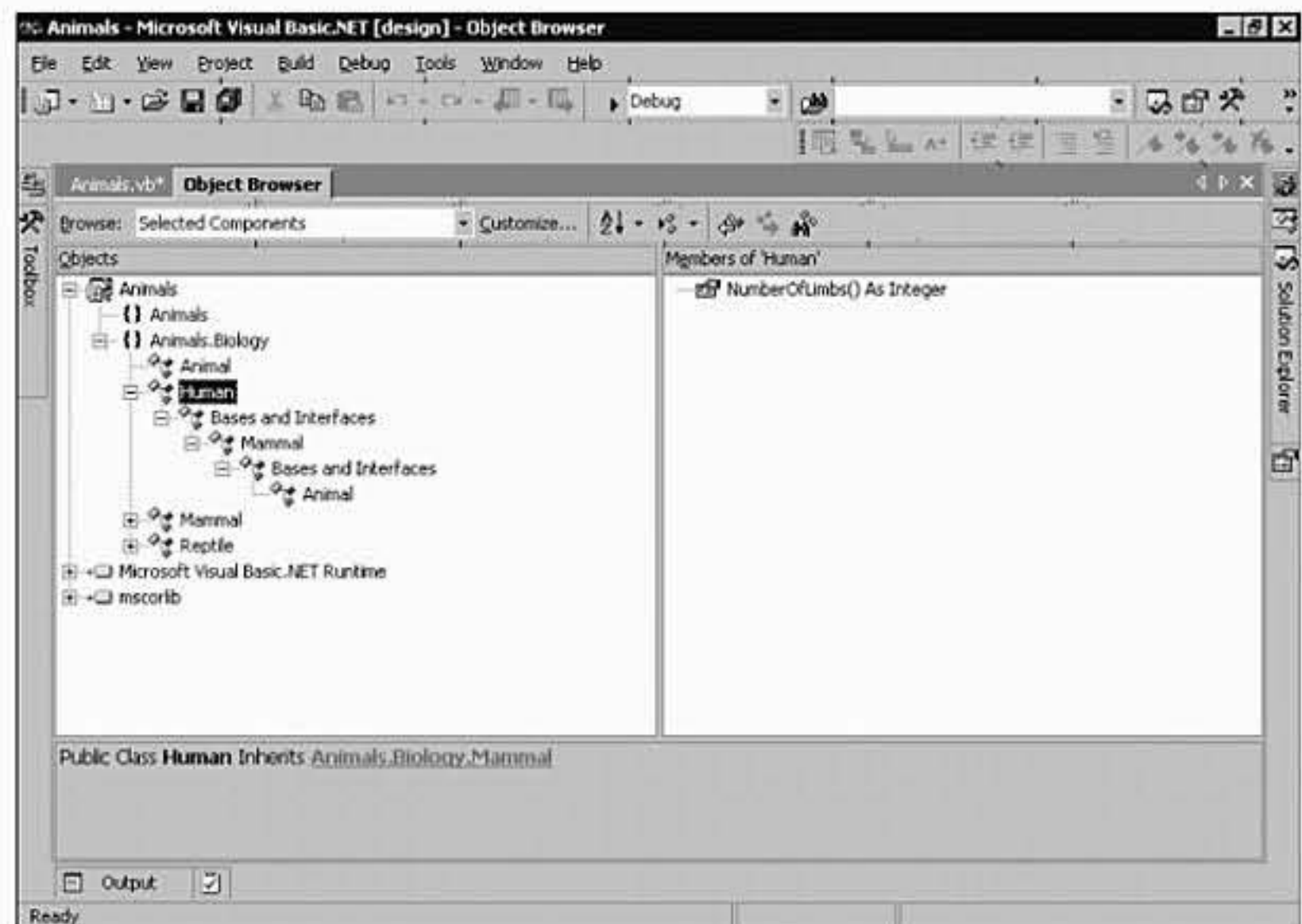
Throw New System.Exception("All mammals breathe air")

End Set


```
End Property  
End Class  
  
Public Class Human  
    Inherits Mammal  
    Public Overrides Property NumberOfLimbs() As Integer  
        Get  
            Return 4  
        End Get  
        Set(ByVal Value As Integer)  
        End Set  
    End Property  
End Class  
End Namespace
```

FIGURA 7.4

O Object Browser mostra as classes que compõem esse exemplo e seu relacionamento umas com as outras.



Não se preocupe se seu código não ficar tão parecido com esse exemplo, contanto que funcione e você possa usar alguns dos recursos dos objetos descritos na lição do Dia 7.

Respostas do Dia 8

Teste

1. Você poderia redirecionar os fluxos de resultados ou erro para um arquivo com a finalidade de registrá-los.

2. O conjunto `SortedList` ordenará de modo automático os itens em uma lista à medida que forem adicionados, portanto, você não tem de se lembrar de chamar repetidamente o método `Sort`. O conjunto `ArrayList` não possui essa função.
3. Um número entre 2 e 12 seria exibido no console.

Exercícios

Para criar o programa do console descrito, você usaria o método `GetCommandArgs` da classe `Environment` e o método `Sort` da classe `ArrayList`. Dessa maneira, seriam necessários os espaços de nome `System` e `System.Collections`. Siga estas etapas:

1. Crie um aplicativo do console. O meu se chama `SortedArgs`.
2. Altere o nome do módulo de `Module1` para `modMain`. Salve o arquivo resultante como `Main.vb`.
3. Dê um clique com o botão direito do mouse sobre o nome do projeto no `Solution Explorer` e selecione `Properties`. Altere o `Startup Object` para `modMain`. Selecione `OK` para fechar a caixa de diálogo.
4. O método `GetCommandArgs` da classe `Environment` retorna um array de strings de cada argumento listado na linha de comando (incluindo o nome do aplicativo). Você pode usar isso na criação de um conjunto `ArrayList` e empregar a funcionalidade de ordenação interna para organizar sua lista.
5. Já que o nome do aplicativo está em seu conjunto `ArrayList`, você deve removê-lo antes de ordenar a lista.
6. Depois de ordená-la, você poderá usar o método `Console.WriteLine` para exibir a lista resultante. Empregue um laço `For...Next` para visualizar todos os itens dela.

O código-fonte completo para o exemplo está na Listagem 8.4.

CÓDIGO

LISTAGEM 8.4 Ordenando os Argumentos da Linha de Comando

```
1 Module modMain
2     Sub Main()
3         Dim oArgs As New ArrayList(Environment.GetCommandLineArgs())
4         Dim I As Integer
5         With oArgs
6             .RemoveAt(0)
7             .Sort()
8             For I = 0 To .Count - 1
9                 Console.WriteLine(.Item(I))
10            Next
11        End With
12    End Sub
13 End Module
```


ANÁLISE

A Listagem 8.4 é pequena, mas demonstra várias técnicas para a manipulação de conjuntos, assim como a recuperação de argumentos da linha de comando.

A linha 3 cria um novo conjunto `ArrayList`, copiando o array de strings retornado pelo método `GetCommandLineArgs`. Essa técnica está disponível para muitos dos conjuntos definidos em `System.Collections`. Ela permite que você pegue um array existente, insira-o em um conjunto (`ArrayList`, `Queue` ou `Stack`) e defina seu comportamento adicional. Nesse caso, ela executa a remoção do item indesejado (linha 6) e ordena os restantes (linha 7). Para concluir, o código que vai da linha 8 a 10 exibe o array resultante. A execução do aplicativo deve produzir o resultado a seguir:

A**RESULTADO**

```
1 SortedArgs Asimov Heinlein radley Niven
2 Pournelle Clarke Herbert Card LeGuin Haldeman
3 Asimov
4 Bradley
5 Card
6 Clarke
7 Haldeman
8 Heinlein
9 Herbert
10 LeGuin
11 Niven
12 Pournelle
```

Respostas do Dia 9

Teste

1. Modal *versus* não-modal: quando uma janela é exibida de maneira modal, o usuário não pode interagir com nenhuma parte do aplicativo até que a janela seja fechada. Códigos que exibem formulários modais têm sua execução interrompida até o momento em que esse formulário fique oculto ou seja fechado. Quando um formulário não-modal é exibido, o usuário não é impedido de trabalhar com qualquer outra parte do aplicativo, e o código que o exibiu continuará a ser executado sem ter de esperar que o formulário seja fechado.
2. A propriedade `CausesValidation` é configurada como `True` para indicar que desejamos assegurar a existência de dados válidos nos campos de entrada de dados quando o usuário usar esse controle. Se você configurar essa propriedade em um botão `Cancel`, então, o usuário terá de inserir informações válidas mesmo quando quiser apenas cancelar o processo!

3. A instrução `Handles`, seguida de um ou mais nomes de eventos específicos, é usada para indicar que o procedimento é um manipulador de eventos. Ter `Handles btnOpen.Click` no final de um procedimento significa que ele será chamado sempre que o evento `Click` de `btnOpen` ocorrer (quando o usuário der um clique em `btnOpen`).

Exercícios

1. A estratégia para manipular os três eventos com apenas uma rotina é determinar que evento específico ocorreu quando ela foi chamada. Pela conversão do parâmetro `sender` em um objeto `System.Windows.Forms.Control`, você poderá acessar sua propriedade `Name`, que fornecerá o nome do controle que é a origem do evento. O novo procedimento de eventos, que substituiria os outros três, é mostrado na Listagem 9.21.

LISTAGEM 9.21 Procedimento de Eventos

```
1 Private Sub DoEverything(ByVal sender As Object,_  
2 ByVal e As System.EventArgs)_  
3 Handles btnCopy.Click, btnMove.Click, btnDelete.Click  
4  
5     Dim sEventSource As String _  
6     = CType(sender, Control).Name  
7     Dim sSource As String  
8     Dim sDestination As String  
9     sSource = txtSource.Text()  
10    sDestination = txtDestination.Text()  
11  
12    Select Case sEventSource  
13        Case "btnCopy"  
14            File.Copy(sSource, sDestination)  
15        Case "btnMove"  
16            File.Move(sSource, sDestination)  
17        Case "btnDelete"  
18            File.Delete(sSource)  
19        Case Else  
20            'não faça nada  
21    End Select  
22 End Sub
```


Respostas do Dia 10

Teste

1. Os controles `Label` fornecerão a você o recurso de alterar posteriormente o texto de maneira dinâmica.
2. Os controles `LinkButton` e `HyperLink`.
3. Use os controles `Image` ou `ImageButton` (se a figura tiver como finalidade ser clicada), ou apenas selecione `Insert` e `Image` no menu.

Exercícios

Por causa das condições do formulário, devem ser usados os controles `Validator`. Adicione-os a todos os outros controles. A maior parte será de controles `RequiredFieldValidators`, mas também será necessário um `CompareValidator` para assegurar que as duas senhas coincidam. Além disso, você verá como se cria um campo de senha.

Crie um projeto (ou adicione um novo formulário a um projeto existente). Adicione uma tabela nova (`Table`, `Insert`, `Table`) ao formulário, com três colunas e seis linhas. Veja a Figura 10.9 para obter detalhes.

Adicione três títulos ao novo formulário, um para cada uma das três primeiras linhas da primeira coluna. De maneira semelhante, adicione quatro controles `TextBox` às primeiras quatro linhas da segunda coluna. O resultado final deve ficar parecido com a Figura 10.10.

FIGURA 10.9

Caixa de diálogo `Insert Table`.

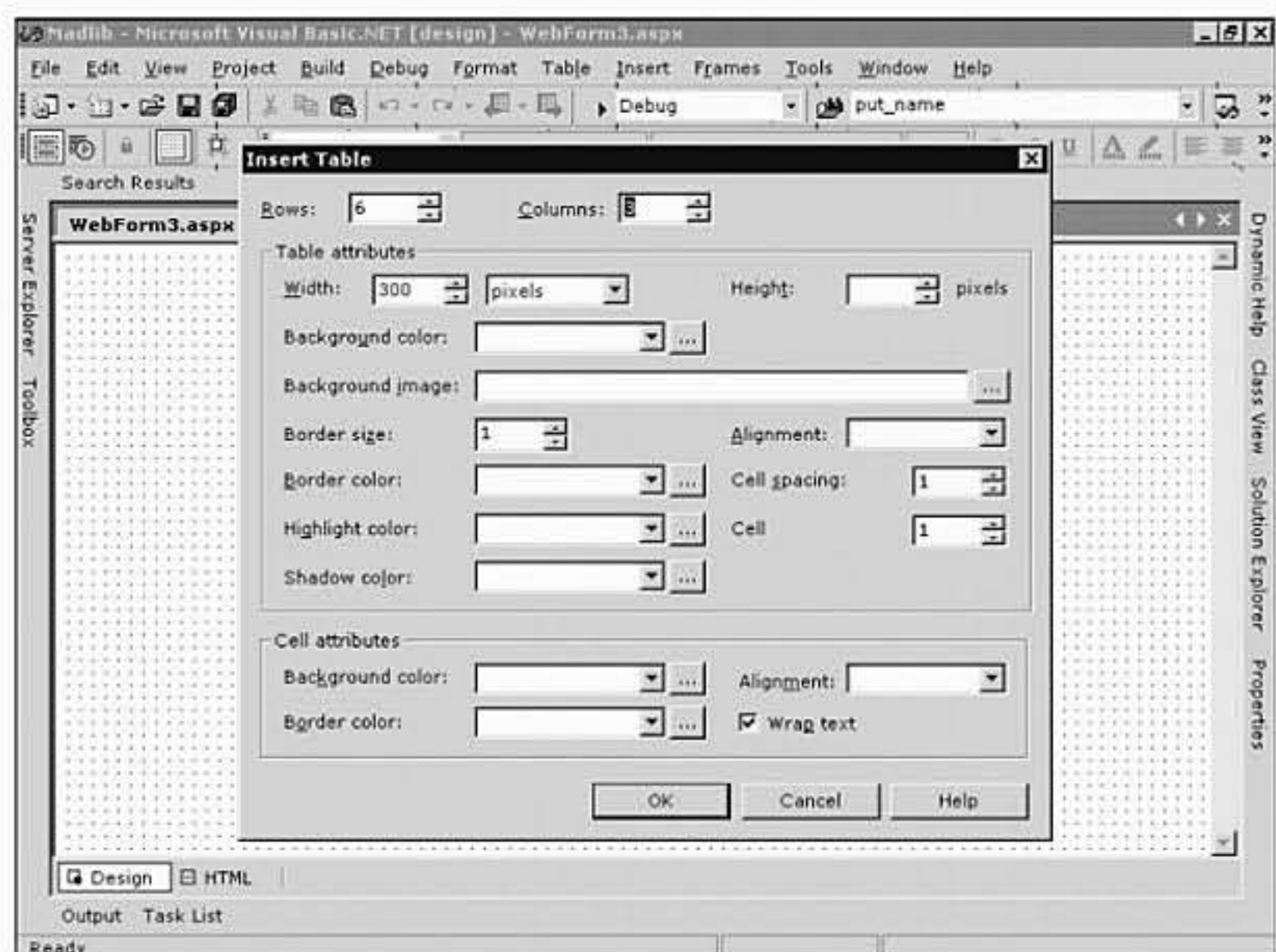
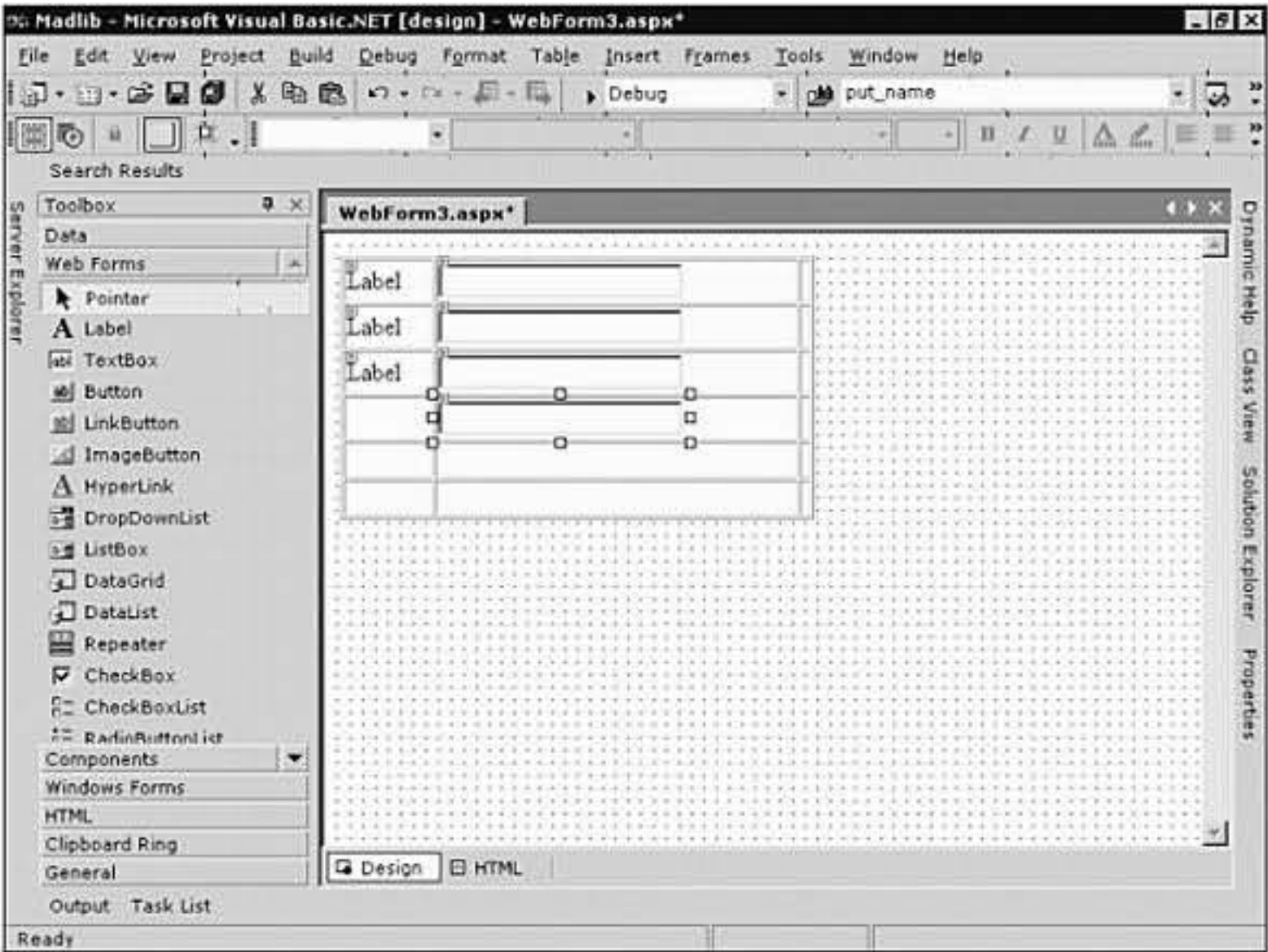


FIGURA 10.10
*O formulário Web
sendo criado.*



Para concluir a segunda coluna, adicione um controle Button e outro Label. Por fim, na terceira coluna, adicione (de cima para baixo) três controles RequiredFieldValidators e um CompareValidator. Configure as propriedades de todos os controles como descrito na Tabela 10.6.

TABELA 10.6 Propriedades do Formulário de Registro

Controle	Propriedade	Valor
Label	(ID)	lblName
	Text	Name:
	Font Bold	True
Label	(ID)	lblAlias
	Text	Alias:
	Font Bold	True
Label	(ID)	lblPassword
	Text	Password:
	Font Bold	True
TextBox	(ID)	txtName
TextBox	(ID)	txtAlias
	MaxLength	10
TextBox	(ID)	txtPassword1
	TextMode	Password
TextBox	(ID)	txtPassword2
	TextMode	Password

TABELA 10.6 Propriedades do Formulário de Registro (*continuação*)

<i>Controle</i>	<i>Propriedade</i>	<i>Valor</i>
Button	(ID)	cmdRegister
	Text	Register
Label	(ID)	lblSummary
	BackColor	#E0E0E0
	BorderColor	Silver
	BorderStyle	Groove
	BorderWidth	1px
	Text	<blank>
Validator	(ID)	reqName
	ControlToValidate	txtName
	ErrorMessage	Name is a required field.
Validator	(ID)	reqAlias
	ControlToValidate	txtAlias
	ErrorMessage	Alias is a required field.
Validator	(ID)	reqPassword
	ControlToValidate	txtPassword1
	ErrorMessage	You must enter a password.
CompareValidator	(ID)	cmpPassword
	ControlToCompare	txtPasssword1
	ControleToValidate	txtPasssword2
	ErrorMessage	Two passwords must match.

Por fim, você está pronto para adicionar o código. Como antes, não é necessário inserir um que manipule a validação porque os controles gerenciam isso. Portanto, não há um código extenso a ser adicionado a página. Só é preciso criar aquele que será executado quando o usuário der um clique no botão de registro (Register). Ele será processado apenas se todos os erros forem corrigidos e só exibirá uma mensagem no campo do resultado (summary). Dê um clique duas vezes no botão Register e adicione o código da Listagem 10.5.

CÓDIGO**LISTAGEM 10.5** Código do Botão Register

```

1  Public Sub cmdRegister_Click(ByVal sender As Object, ByVal e As
    ➤System.EventArgs)
2      lblSummary.Text = "Welcome, " & txtAlias.Text
3  End Sub

```


Compile e execute o projeto. Quando o navegador estiver disponível, tente usar valores diferentes nos campos. Exclua qualquer entrada que tiver inserido nos campos Name e Alias. Você deve ver as mensagens de erro desses controles. Digite duas senhas diferentes para ver o controle CompareValidator em ação (veja a Figura 10.11). Para concluir, insira valores nos campos Name e Alias, e senhas iguais para ver a mensagem final. Veja a Figura 10.12.

FIGURA 10.11

Formulário de registro com erros.

The screenshot shows a Microsoft Internet Explorer window with the address bar displaying `http://random/Madlib/WebForm3.aspx`. The browser's title bar indicates it is a "Daily Build 6.00.2432.0002". The form contains the following fields and messages:

Name:	<input type="text" value="Me"/>	
Alias:	<input type="text"/>	Alias is a required field
Password:	<input type="password" value="*"/>	
	<input type="password"/>	Two passwords must match
	<input type="button" value="Register"/>	

The status bar at the bottom shows "Done" and "Local intranet".

FIGURA 10.12

Formulário de registro preenchido corretamente.

The screenshot shows the same Microsoft Internet Explorer window with the address bar displaying `http://random/Madlib/WebForm3.aspx`. The form now displays the following:

Name:	<input type="text" value="Me"/>	
Alias:	<input type="text" value="him"/>	
Password:	<input type="password"/>	
	<input type="password"/>	
	<input type="button" value="Register"/>	
	<input type="text" value="Welcome, him"/>	

The status bar at the bottom shows "Done" and "Local intranet".

Respostas do Dia 11

Teste

1. O campo (ou campos) usado para identificar de maneira exclusiva o registro de um banco de dados é chamado de chave primária.
2. Esse código SQL executou uma instrução LEFT OUTER JOIN entre essas duas tabelas, especificando que todos os registros de Marca e Modelo que atendam à condição da associação sejam incluídos no resultado. Os resultados a seguir serão gerados:

<i>Marca</i>	<i>Modelo</i>
Ford	Mustang
Ford	Explorer
Audi	<Nulo>
BMW	<Nulo>
Pontiac	Grand Am
Pontiac	Grand Prix
Pontiac	Aztek
Toyota	Rav 4
Toyota	Camry

3. Essa instrução SQL especifica uma associação interna, que é o tipo mais comum de união. Ela produzirá resultados apenas com os registros das duas tabelas em que a condição da associação for atendida, como vemos a seguir:

<i>Marca</i>	<i>Modelo</i>
Ford	Mustang
Ford	Explorer
Pontiac	Grand Am
Pontiac	Grand Prix
Pontiac	Aztek
Toyota	Rav 4
Toyota	Camry

Observe que Audi e BMW, que não têm registros na tabela Modelo, não foram incluídos no resultado da segunda instrução SQL.

Exercícios

1. Há muitas maneiras pelas quais você poderia estender seu banco de dados de CDs para abranger os usuários; veja a seguir a mais simples:

Primeiro, adicione uma tabela de usuários (User) que terá o campo da identificação do usuário (UserID) como chave primária. Que outros campos serão incluídos nessa tabela é irrelevante, mas provavelmente ela terá informações como o nome e endereço eletrônico do usuário, e talvez uma senha, de modo que você possa garantir a segurança e impedir que os usuários alterem a coleção de CDs de outras pessoas.

A seguir, você trabalhará com base no fato de que se dois usuários tiverem o álbum “Fields of Gold” do Sting, deve haver apenas uma entrada relacionada a Sting na tabela Artist, e apenas uma em Disc referente a esse CD específico. Com isso definido, no final teremos um relacionamento muitos-para-muitos entre User e Disc, que pode ser expresso como uma tabela de relacionamentos UserDisc. O banco de dados alterado é mostrado na Figura 11.7.

2. Mesmo com as informações de Orders.txt sendo apresentadas de maneira inadequada, não é raro receber dados nesse formato, principalmente quando são originários de uma planilha. Para colocá-los em um banco de dados eficiente, você tem de separar as informações sobre o cliente das referentes ao pedido e ainda inserir os itens de cada pedido (e se um cliente quisesse pedir quatro itens no mesmo pedido?) em sua própria tabela. Durante o processo, também poderiam ser removidos alguns campos que produzem resultados calculados (total do pedido, custo total do item X) que em vez disso seriam obtidos como parte de uma consulta. O layout desse banco de dados é mostrado na Figura 11.8.

FIGURA 11.7

Só precisamos adicionar duas tabelas para transformar seu banco de dados de CDs em um sistema multiusuário.

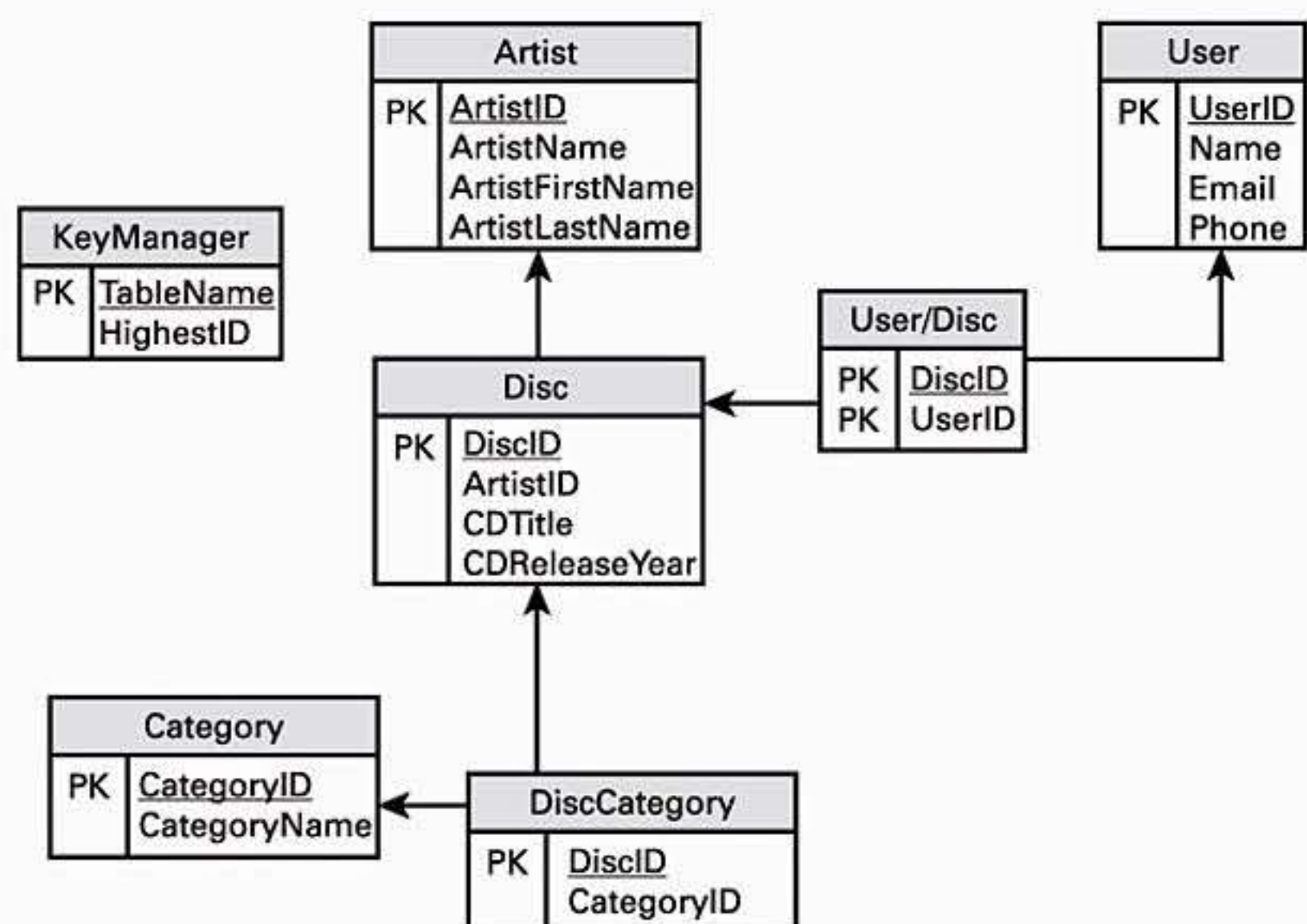
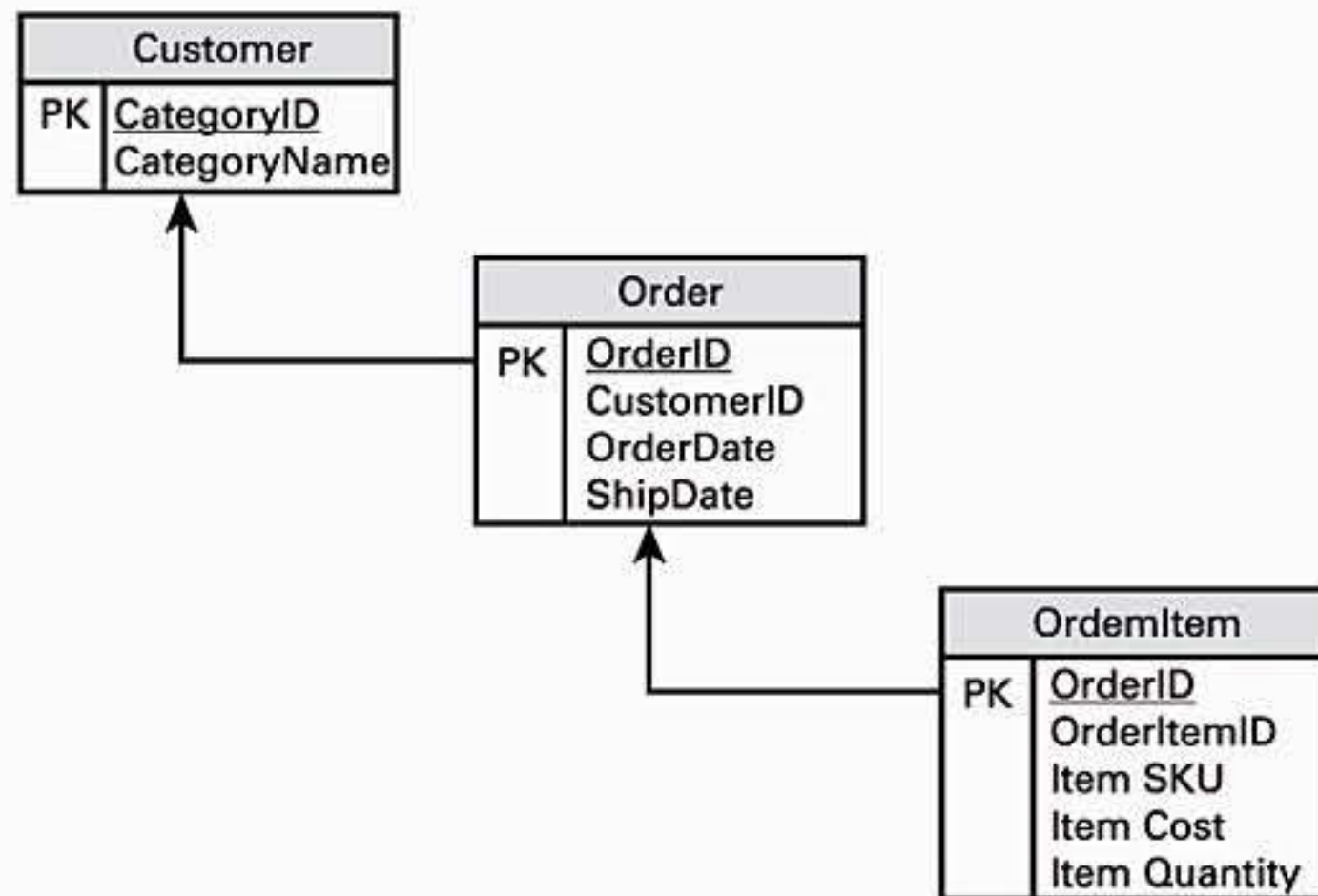


FIGURA 11.8

Banco de dados de pedidos reestruturado.



Se quisesse, você poderia ir adiante com esse banco de dados, criando uma tabela de itens para armazenar um registro de cada SKU composto do preço e da descrição do item e remover esses campos da tabela OrderItem. Observe que em um banco de dados de entrada de pedidos às vezes é necessário existir alguma duplicação – armazenar o custo dos itens com os itens do pedido, por exemplo – para permitir que os preços dos itens sejam alterados com o tempo sem que isso afete as informações de pedidos anteriores.

Respostas do Dia 12

Teste

1. A propriedade em questão é `MissingSchemaAction`, e a configuração de `AddWithKey` é que assegurará que as informações da chave primária sejam criadas.
2. O método `Delete` marca a linha como excluída, mas não a remove da tabela; ela permanece para que a rotina `Update` saiba que tem de excluir essa linha do banco de dados. O método `Remove` extrai imediatamente o objeto `DataRow` do conjunto `Rows`, porém ele não é removido do banco de dados quando `Update` é chamado.
3. Esse é um comando que não retorna registros, portanto, você deve chamá-lo usando o método `ExecuteNonQuery` do objeto `Command`. Seu código pode ficar com a aparência a seguir:

```

Sub Main()
    Dim sSQL As String
    sSQL = "DELETE FROM Artist Where ArtistID=3"
    Dim objConn As New
        System.Data.OleDb.OleDbConnection(sConnection)
    Dim objCmd As New
        System.Data.OleDb.OleDbCommand(sSQL, objConn)

```



```
objConn.Open()  
objCmd.ExecuteNonQuery()  
End Sub
```

Exercícios

Você poderia criar uma interface para seu banco de dados de registro de CDs de muitas maneiras, mas as partes essenciais serão as mesmas em todos os casos.

Respostas do Dia 13

Teste

1. Loaded Modules e Processes (e você também pode obter essa informação no Management Information).
2. O serviço é um programa executado em segundo plano no seu computador, que fornece algum recurso. Por exemplo, o SQL Server disponibiliza recursos de bancos de dados.
3. O valor atual do percentual de CPU em uso (% Processor Time) seria exibido no console.

Respostas do Dia 14

Teste

1. A palavra-chave MyBase fornece a funcionalidade desejada e pode ser usada para se referir a propriedades ou métodos da classe básica.
2. Criar múltiplas versões do mesmo procedimento, porém com diferentes conjuntos de parâmetros, é chamado de sobreposição.
3. O resultado será
x.Name = Fred
y.Name = Fred

Tanto x quanto y se referem à mesma instância de myFirstClass, e x.Name = y.Name não altera nada porque é equivalente a x.Name = x.Name.

Exercícios

Um caminho possível para o projeto seria manipular os diversos tipos de evento usando uma hierarquia de objetos, em vez de empregar apenas uma propriedade de tipo ou categoria em uma única classe Event. Você poderia criar uma classe básica Event, com algumas propriedades comuns como o nome do evento e talvez as informações de contato referentes a esse evento. Deri-

vadas dessa classe, outras poderiam ser criadas como `SportingEvent` (herda de `Event`), `ConcertEvent`, `ComedyEvent`, `PublicSpeakingEvent` e assim por diante. A hierarquia poderia ser levada ainda mais para baixo, conduzindo a classes como `BaseballGame` (herda de `SportingEvent`), `PoliticalEvent` (herda de `PublicSpeakingEvent`) e outras. As propriedades individuais das diversas classes derivadas podem ser descritas no nível de detalhe desejado, mas devem evitar a repetição. Se duas classes tiverem propriedades semelhantes, considere passar as propriedades em comum para uma classe básica.

A

Respostas do Dia 15

Teste

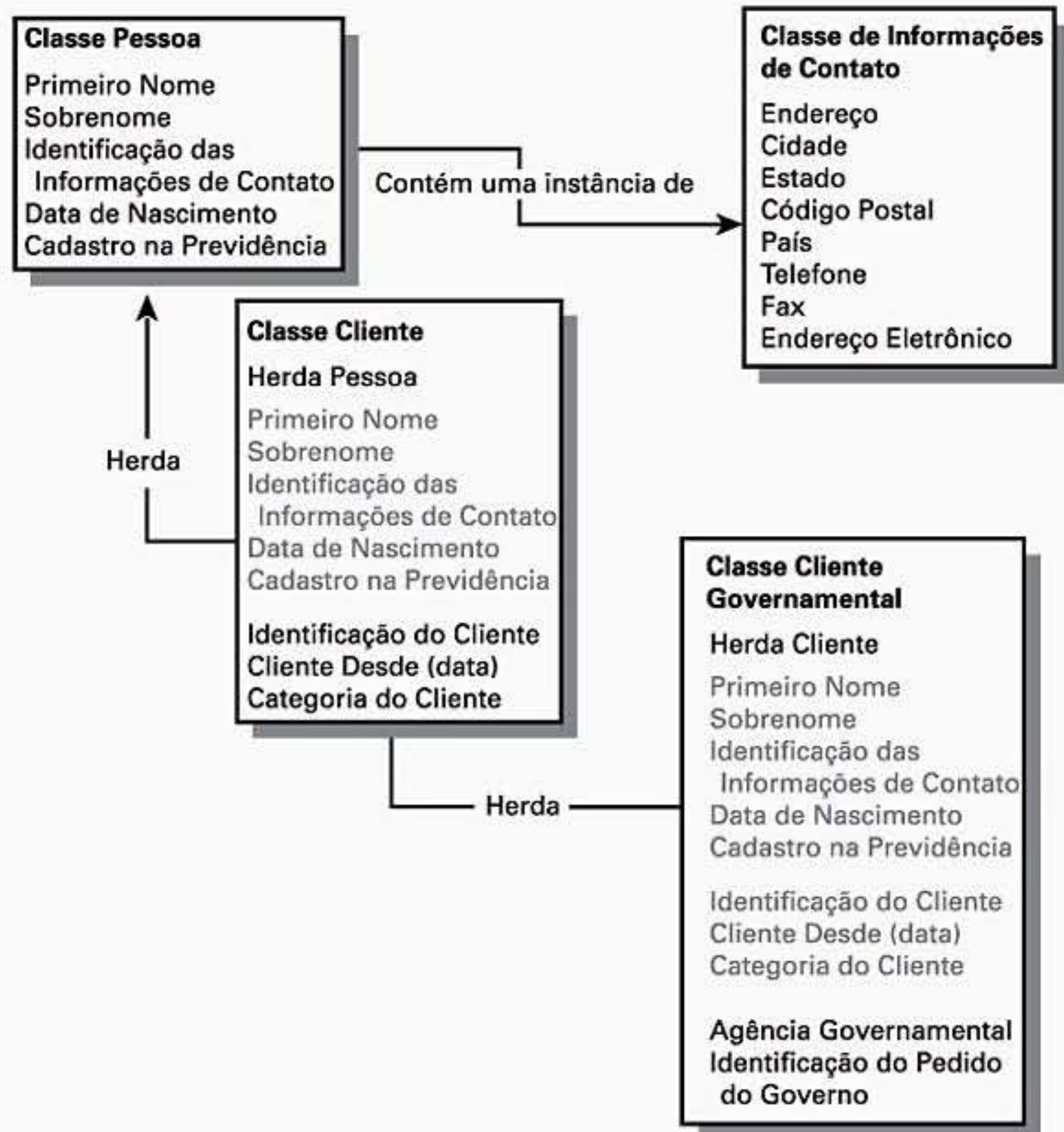
1. As interfaces fornecem esse tipo de funcionalidade, permitindo a demonstração de que muitas classes diferentes compartilham alguns recursos. Elas são bastante adequadas a essa tarefa porque uma única classe pode implementar várias interfaces distintas.
2. A palavra-chave `MustOverride` pode ser usada para indicar que uma propriedade ou método específico deve ser sobreposto em uma classe derivada.
3. A palavra-chave `MustInherit` transforma uma classe básica em 'abstrata', que não pode ser instanciada, mas, em vez disso, deve ter classes derivadas dela.
4. As classes marcadas como `NotInheritable` também são conhecidas como 'lacradas', e não podem ser usadas como uma classe básica (outras classes não podem herdar características dela).

Exercícios

Há muitas maneiras pelas quais você poderia criar uma hierarquia de objetos que incluísse o cliente, mas a Figura 15.4 ilustra um exemplo. Nesse caso, a classe `Cliente` foi derivada de uma classe `Pessoa` mais genérica, e os aspectos das informações de contato foram gerados diretamente nessa classe básica de origem.

FIGURA 15.4

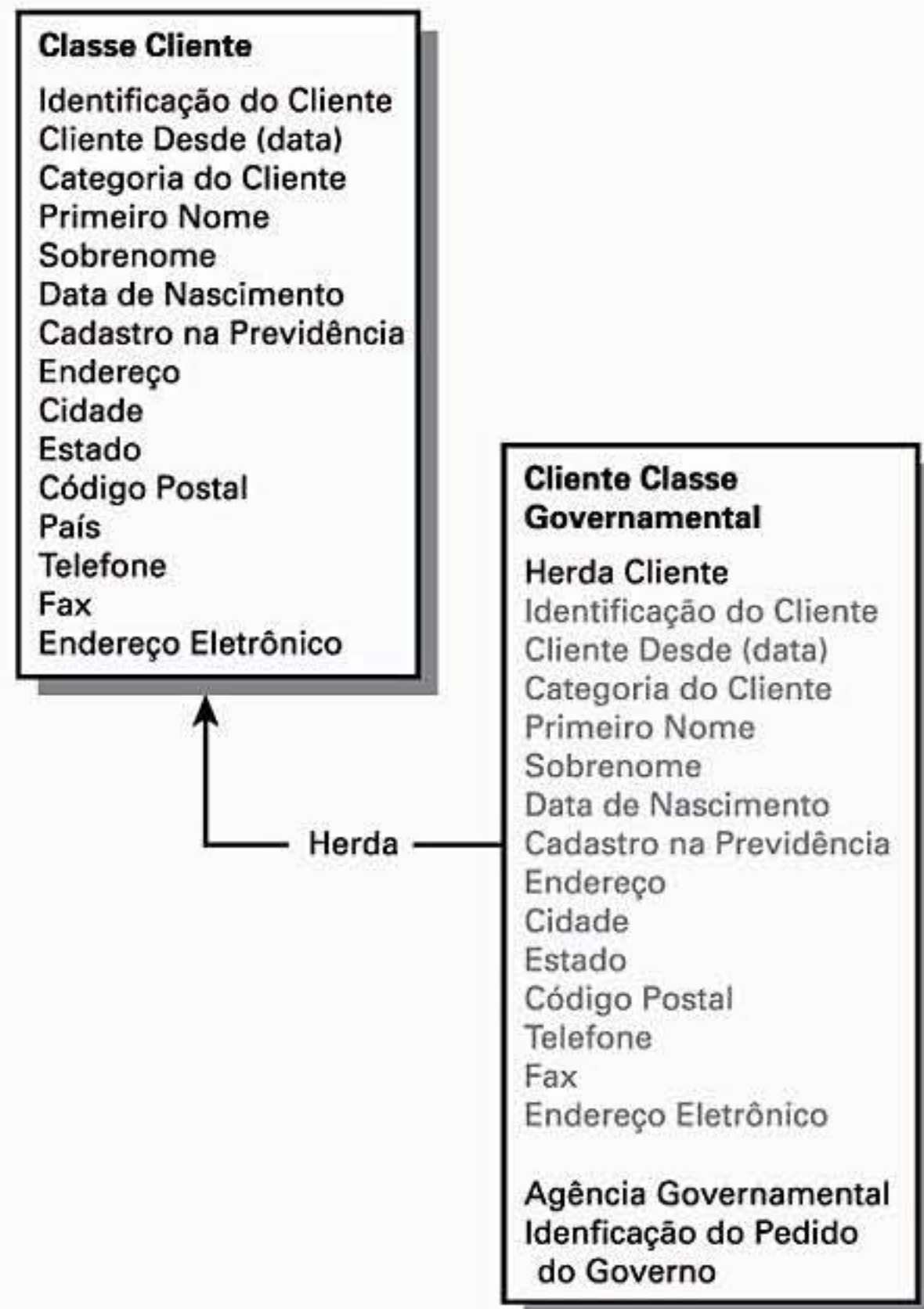
Projetar classes para serem reutilizadas às vezes pode gerar uma estrutura mais complexa.



Um projeto alternativo, na Figura 15.5, ilustra como uma versão mais simples com as mesmas informações pode ser estruturada.

FIGURA 15.5

O mesmo conjunto de informações pode ser modelado de várias maneiras.



A

Respostas do Dia 16

Teste

1. Use o método `LayoutMdi` do formulário MDI-pai para configurar o layout como `MdiLayout.Cascade`. Isso organizará as janelas-filhas de modo que todas as suas barras de título fiquem visíveis.

Exercícios

1. Na verdade, é uma preferência pessoal. Em geral, prefiro o modelo SDI porque posso ver as janelas individualmente na barra de tarefas. No entanto, em alguns aplicativos que podem abrir múltiplos arquivos, como um que use figuras, prefiro o MDI, para armazenar todas elas no mesmo local.
2. Os métodos `GetAttributes` e `GetLastWriteTime` da classe `File` são úteis quando essas informações são recuperadas em um arquivo. Os dois são métodos compartilhados da classe `File`, portanto você não precisa criar a instância de um arquivo antes de usá-los. Para

adicionar essa funcionalidade, você pode alterar o aplicativo Pioneer original ou copiar todos os arquivos do diretório dele em um novo diretório e alterá-los.

Altere a propriedade View de ListView para Detail, Isso ativará o recurso com o qual você poderá adicionar várias colunas à ListView. Adicione três colunas à ListView usando o editor da propriedade Columns. A Tabela 16.17 descreve as propriedades dessas colunas.

TABELA 16.17 Propriedades da Coluna

<i>Coluna</i>	<i>Propriedade</i>	<i>Valor</i>
Primeira	Name	hdrName
	Text	File Name
	Width	180
Segunda	Name	hdrAttributes
	Text	Attributes
	TextAlign	Center
Terceira	Name	hdrModified
	Text	Last Modified
	Text Align	Right
	Width	80

O único código necessário é o que adiciona as informações às colunas Attributes e Last Modified. Altere o manipulador de eventos tvwFolders_AfterSelect para criar essas colunas (veja a Listagem 16.8).

LISTAGEM 16.8 Criando Colunas Adicionais

```

1 Private Sub tvwFolders_AfterSelect(ByVal sender As Object, _
2     ByVal e As System.Windows.Forms.TreeViewEventArgs) _
3     Handles tvwFolders.AfterSelect
4
5     Dim sFiles() As String = _
6         Directory.GetFiles(tvwFolders.SelectedNode.FullPath)
7     Dim sFile As String
8     Dim oItem As ListViewItem
9
10    lvwFiles.Items.Clear()
11
12    For Each sFile In sFiles
13        oItem = lvwFiles.Items.Add(StripPath(sFile))
14        If lvwFiles.Items.Count > 0 Then

```


LISTAGEM 16.8 Criando Colunas Adicionais (*continuação*)

```
15         oItem.SubItems.Add(GetAttributeString(sFile))
16         oItem.SubItems.Add(File.GetLastWriteTime(sFile))
17     End If
18 Next
19 End Sub
```

ANÁLISE

Os trechos adicionados foram a declaração de `ListViewItem` na linha 8 e o código das linhas 13 a 17. Quando um novo objeto `ListViewItem` é adicionado à `ListView`, o item inserido é retornado pelo método `Add`. Se você adicionar apenas um item a lista, como fizemos antes, não será importante salvá-lo em uma nova variável. No entanto, já que estará adicionando subitens a esse novo item, deve armazená-lo em uma variável, como vemos na linha 13. Depois de adicionar o item, usamos a propriedade `SubItems` de cada `ListViewItem` para inserir novos valores nas colunas criadas anteriormente. Primeiro, o código chama a função `GetAttributeString`, que será criada em breve, para adicionar a string que representa cada atributo do arquivo. Em seguida, a linha 16 insere na coluna `Last Modified` a hora e a data em que o arquivo foi gravado pela última vez.

Para concluir, você deve criar uma rotina com a finalidade de exibir os atributos definidos para o arquivo. Adicione o código da Listagem 16.9 a fim de executar essa operação.

LISTAGEM 16.9 Função `GetAttributeString`

```
20 Private Function GetAttributeString(ByVal fileName As String) _
21     As String
22     Dim sReturn As String
23     Dim oAttr As FileAttributes = File.GetAttributes(fileName)
24
25     If (oAttr And FileAttributes.ReadOnly) = _
26         FileAttributes.ReadOnly Then
27         sReturn = "R"
28     Else
29         sReturn = "-"
30     End If
31
32     If (oAttr And FileAttributes.Hidden) = _
33         FileAttributes.Hidden Then
34         sReturn += "H"
35     Else
36         sReturn += "-"
37     End If
38
39     If (oAttr And FileAttributes.System) = _
40         FileAttributes.System Then
```


LISTAGEM 16.9 Função GetAttributeString (*continuação*)

```

41         sReturn += "S"
42     Else
43         sReturn += "-"
44     End If
45
46     Return sReturn
47
48 End Function

```

ANÁLISE

A função GetAttributeString recupera os atributos de cada arquivo e os converte em uma string para visualização. O método File.GetAttributes retorna um valor FileAttributes que contém a soma de todos os atributos definidos para o arquivo (linha 23). A instrução If das linhas 25 e 26 testa esse valor para saber se ele inclui o atributo ReadOnly. Esse teste usa o operador lógico AND para comparar o valor integral que contém todos os atributos com o atributo FileAttributes.ReadOnly. Se o valor de FileAttributes.ReadOnly fizer parte do total, esse teste o retornará. Isto é, se você associar dois números ao operador AND, o valor retornado será o comum aos dois ou 0 se não compartilharem nada. Por exemplo, 33 AND 1 retornaria 1, enquanto 33 AND 2 retornaria 0. Se o arquivo realmente possuir o atributo ReadOnly configurado, o código acrescentará uma letra R à string; caso contrário, acrescentará um hífen. Os testes dos atributos Hidden e System são idênticos, exceto pelo atributo desejado. Para concluir, na linha 46, a string completa é retornada para ser adicionada à coluna ListView.

NOTA

O valor retornado de GetAttributes é um bitmask de todos os atributos de um arquivo. *Bitmask* é o termo aplicado a um valor em que cada bit é usado como um flag, para marcar alguma característica. Por exemplo, se você examinar alguns dos valores reais de cada atributo da enumeração de FileAttributes (veja a Tabela 16.18), verá que há intervalos entre eles.

TABELA 16.18 Possíveis Valores dos Atributos do Arquivo

<i>Membro</i>	<i>Valor</i>
ReadOnly	1
Hidden	2
System	4
Directory	16
Archive	32
Device	64
Normal	128
Temporary	256
Sparse File	512
Reparse Point	1024
Compressed	2048
Offline	4096
Not Content Indexed	8192
Encrypted	16384
