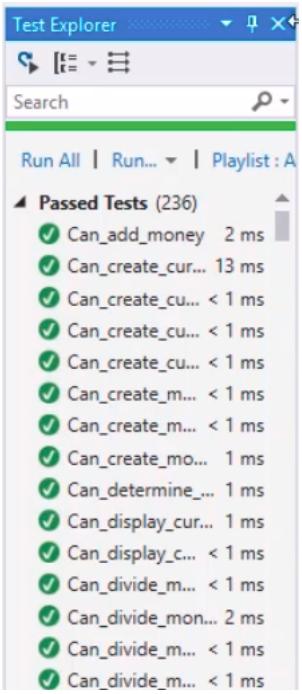
07

Extrair Método 2

Transcrição

Modificamos o projeto `caelum-stella-csharp` refatorando o seu código, podemos garantir que não quebramos nenhuma funcionalidade existe ao rodar os *testes de unidade* desse projeto.

No Visual Studio, vamos em "Test > Run > All Tests" para rodar esses testes de unidade.



Com esses testes de unidade que deram certo, podemos seguir em frente em segurança com a refatoração. Podemos fechar a janela de testes e o arquivo `Moeda.cs` , e abrir um novo projeto da aplicação chamado de `refatoracao` . Esse projeto contém uma pasta chamada `Aula01` e dentro dela temos `R01.ExtractMethod". Dentro deste, teremos mais duas pastas: antes e depois`.

Trabalharemos com o arquivo `Relatorio.cs` que será refatorado e se encontra em `depois` . Desta forma, poderemos comparar com o arquivo da pasta `antes` e ver o que foi refatorado nessa classe.

Ao abrirmos `Relatorio.cs` , veremos `Imprimir()` , que cria um pedido, adiciona e imprime itens, além de depois imprimir detalhes como *nome* e *valor*.

Como esse método é longo,iremos simplificá-lo extraindo métodos. Começaremos com o trecho que *imprime itens*:

```
class Relatorio
{
    void Imprimir()
    {
        // primeira parte do código

        // imprimir itens
        Console.WriteLine("******************************");
        Console.WriteLine("********** Itens **********");
        Console.WriteLine("******************************");
```

```
            foreach (var item in pedido.Itens)
            {
                decimal valorItem = item.Quantidade * item.PrecoBase;
                Console.WriteLine($"{item.Desconto}: {item.Quantidade} unidades, R$ {valorItem}'
                total = total + valorItem;
            }
        }
    }
```

Ao selecionar esse trecho, utilizaremos o atalho "Ctrl + ." para extrair a um novo método.

```
class Relatorio
{
    void Imprimir()
    {
        decimal total = 0.0m;
        var pedido = new Pedido("José da Silva");
        pedido.AddItem("Dentozap", 2, 10m, 0m, 3m);
        pedido.AddItem("Voldax", 3, 10m, 0m, 3m);
        pedido.AddItem("Translab", 7, 10m, 0m, 3m);

        // imprimir itens
        total = (total, pedido);
```

A variável `total` está sendo passada como parâmetro desse novo método, que está declarada em cima, na primeira linha do método, e também a variável `pedido` que contém o nome do cliente e os itens do pedido. Porém, existe um detalhe: a variável `total` 'só é usada, a partir do trecho extraído. O que era possível fazer para melhorar a chamada?

Poderíamos ter movido a variável `total` para próximo o trecho extraído, pois assim, ela seria extraída juntamente, eliminando a necessidade de passar o parâmetro `total` . Vamos desfazer essa alteração, para que possamos mover a declaração do `total` .

```
class Relatorio
{
    void Imprimir()
    {
        // primeira parte do código

        // imprimir itens
        Console.WriteLine("*****************************");
        Console.WriteLine("********** Itens **********");
        Console.WriteLine("*****************************");
        decimal total = 0.0m;
        foreach (var item in pedido.Itens)
        {
            decimal valorItem = item.Quantidade * item.PrecoBase;
            Console.WriteLine($"{item.Desconto}: {item.Quantidade} unidades, R$ {valorItem}'
            total = total + valorItem;
        }
    }
}
```

Em seguida, selecionaremos o trecho que imprime os itens, e utilizaremos o atalho "Ctrl + ." para refatorar clicando em *Extract Method*.

Como podemos ver, teremos somente um único parâmetro: `pedido`. O novo método será `ImprimirItens()`.

```
class Relatorio
{
    void Imprimir()
    {
        var pedido = new Pedido("José da Silva");
        pedido.AddItem("Dentozap", 2, 10m, 0m, 3m);
        pedido.AddItem("Voldax", 3, 10m, 0m, 3m);
        pedido.AddItem("Translab", 7, 10m, 0m, 3m);

        decimal total = ImprimirItens(pedido);

        // imprimir detalhes
```

Criamos o seguinte método:

```
private static decimal ImprimirItens(Pedido pedido
{
    Console.WriteLine("*****************************");
    Console.WriteLine("*********** Itens ***********");
    Console.WriteLine("*****************************");
    decimal total = 0.0m;
    foreach (var item in pedido.Itens)
    {
        decimal valorItem = item.Quantidade * item.PrecoBase;
            Console.WriteLine($"{item.Desconto}: {item.Quantidade} unidades, R$ {valorItem}'
        total = total + valorItem;
    }
}
```

Após a chamada do método `ImprimirItens()`, teremos um bloco de código que imprime os detalhes. Iremos selecioná-lo para extrair um método, utilizando o atalho "Ctrl + .". Chamaremos-no de `ImprimirDetalhes()`.

No começo do método `Imprimir()`, teremos um bloco de código que cria um pedido. Novamente, vamos selecionar todas as linhas do trecho e extrair para um novo método. Ao extraí-lo, renomearemos esse novo método para `CriarPedido()`.

O nosso código ficará assim:

```
class Relatorio
{
    void Imprimir()
    {
        Pedido pedido = CriarPedido();
        decimal total = ImprimirItens(pedido);
        ImprimirDetalhes(pedido, total);
    }
```

```
        private static Pedido CriarPedido()
        {
            var pedido = new Pedido("José da Silva");
            pedido.AddItem("Dentozap", 2, 10m, 0m, 3m);
            pedido.AddItem("Voldax", 3, 10m, 0m, 3m);
            pedido.AddItem("Translab", 7, 10m, 0m, 3m);
            return pedido;
        }

        private static void ImprimirDetalhes(Pedido pedido, decimal total)
        {
            Console.WriteLine("******************************");
            Console.WriteLine("********** Resumo **********");
            Console.WriteLine("******************************");
            Console.WriteLine("nome: " + pedido.Cliente);
            Console.WriteLine("valor: " + total);
        }

            private static void ImprimirItens(Pedido pedido, decimal total)
        {
            Console.WriteLine("******************************");
            Console.WriteLine("********** Itens **********");
            Console.WriteLine("******************************");
            decimal total = 0.0m;
            foreach (var item in pedido.Itens)
            {
                decimal valorItem = item.Quantidade * item.PrecoBase;
                Console.WriteLine($"{item.Desconto}: {item.Quantidade} unidades, R$ {valorItem}'
                total = total + valorItem;
            }
            return total;
        }
```

Será que podemos usar o *Extract Method* em qualquer lugar?

Temos o cálculo do `valorItem`, que é o *produto* da quantidade pelo preço base desse item. Então, o que faremos é *extrair* um método a partir da expressão.

Mas algo interessante aconteceu durante essa extração. O Visual Studio identificou que estamos atribuindo essa expressão à variável `valorItem`, e então, automaticamente foi criado o método `GetValorItem()`. Após aplicar a refatoração, teremos dentro de `GetValorItem()`, o trecho com a expressão do calculo do `valorItem`:

```
    private static decimal GetValorItem(Item item)
    {
        return item.Quantidade * item.PrecoBase;
    }
```

Essa mudança melhorou a leitura do código. Ainda no mesmo código, teremos uma outra expressão: `total = total + valorItem`. Será que vale a pena extrair essa expressão para um método?

Vamos *testar* essa refatoração com `total + valorItem`, por meio do atalho "Ctrl + .". Chamaremos esse novo método de `TotalMaisValorItem`:

```
        foreach (var item in pedido.Itens)
        {
            decimal valorItem = item.Quantidade * item.PrecoBase;
            Console.WriteLine($"{item.Desconto}: {item.Quantidade} unidades, R$ {valorItem}");
            total = TotalMaisValorItem(total, valorItem);
        }
        return total;
    }

    private static decimal TotalMaisValorItem(decimal total, decimal valor)
    {
        return total + valorItem;
    }
```

Esse novo método retorna o total mais o valor do item, e seu nome também será `TotalMaisValorItem` , ou seja, o método faz exatamente o que o nome diz. Complicamos o código desnecessariamente, pois o corpo do método é evidente. Por isso, vamos nos desfazer dessa refatoração.

Aplicaremos a refatoração somente onde for necessário, onde irá simplificar a leitura do código, ou explicar uma expressão complexa, ou então eliminar a quantidade de itens de um método grande. Podemos ainda eliminar a necessidade de ter comentários no código.