02

Validando CPF

Transcrição

Um dos assuntos abordados nessa aula, é a *validação de CPF*. Como já sabemos, o CPF é um documento legal que é utilizado aqui no Brasil para identificar pessoas. Temos aqui três números que precisamos descobrir se é um CPF ou não.

```
86288366757
98745366797
22222222222
```

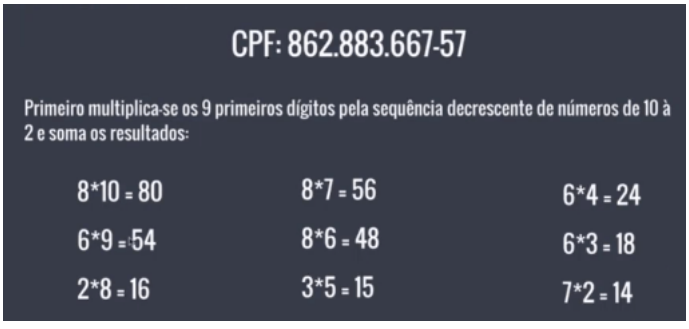
Uma forma bem simples e rápida de ver se esses os números são realmente CPFs, é utilizar um cadastro que faça essa validação. É o caso do site da Alura (https://www.alura.com.br/), onde para que eu me cadastre como aluno, preciso fornecer meu Nome, Telefone, Email e CPF.

Então podemos pegar cada um dos três números que temos, e colocar no campo do CPF.

Dos exemplos, somente o primeiro passa na validação do site sem mostrar mensagens de erro, certo? Então, sabemos que o `86288366757` é um CPF válido.

O que o site da Alura (https://www.alura.com.br/) está fazendo para verificar se o CPF é válido ou não? Aqui está sendo utilizado uma *fórmula* bastante conhecida que é definida por uma legislação. Na da página do Wikipedia (https://pt.wikipedia.org/wiki/Cadastro_de_pessoas_f%C3%ADsicas), sobre o **Cadastro de Pessoa Física**, encontraremos o "cálculo do dígito verificador", uma fórmula matemática que descreve em detalhes, como podemos fazer para descobrir se o número que digitamos no campo de CPF é válido ou não.

Existe uma maneira mais fácil e simples, e mostraremos adiante:



Primeiro, pegaremos os **9** primeiros dígitos do CPF, (do 8 até o primeiro 7), e multiplicaremos cada um desses dígitos por uma sequência de números que vai de **10** até **2**.

Depois, pegamos cada um dos produtos dessas multiplicações e somamos.

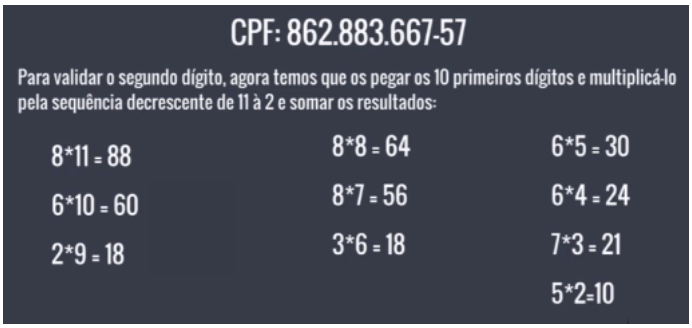
Somando todos os resultados, chegamos no valor: **325**!

Após ter chegado à esse resultado, temos que multiplicá-lo por **10**.

Com o total de **3250**, temos que dividí-lo pelo número primo *11*. Encontraremos o número **295**. Esse resultado será descartado, pois o que queremos na verdade, é o resto da divisão pelo número 11.

O resto dessa divisão será o **5**! E o 5 também é o primeiro dígito verificador que foi digitado no site da Alura! Agora veremos se o segundo verificador 7, também bate com a conta que faremos a seguir.

Observe a imagem:



Faremos o mesmo cálculo, entretanto, a nossa sequência de 9 números agora será de `10`, começando de `11` indo até o número `2`.

Somando todos os 10 produtos, obteremos o valor de **389**. Mais uma vez multiplicaremos 389 por 10, e o dividiremos por 11 para encontrar o resto. O resto da divisão é **7**! Este é o *segundo dígito verificador*, portanto, o CPF é válido.

O que temos aqui é uma maneira simples de mostrar como essa validação foi feita.

Como nós podemos fazer para implementar essas regras utilizando o C#?

No Visual Studio, criaremos um novo projeto para demonstrar essa solução. Chamaremos esse projeto de **Validador de Documentos**.

"Add > New Project > Windows Classic Desktop > Console App (.NET Framework)".

Temos aqui a `Program.cs` com o método `main()`. Dentro da *main*, colocaremos os três CPFs em variáveis do tipo String:

```
class Program
{
    static void Main(string[] args)
    {
        string cpf1 = "86288366757";
        string cpf2 = "98745366797";
        string cpf3 = "22222222222";
    }
}
```

Para validar esses os números de CPF no código, podemos pegar o algoritmo descrito no *Wikipedia* e implementá-lo no código C#. Um melhor caminho seria utilizar uma **biblioteca** de terceiros que já foi escrita e testada para agilizarmos o nosso processo.

Como fazemos normalmente no Visual Studio, vamos importar um pacote do **NuGet**.

Clicando com o botão direito no projeto `ValidadorDeDocumentos`, escolheremos a opção "Manage NuGet Package..." para escolher o pacote de nosso interesse, e em `Browse`, procuramos por "Caelum". Depois, vamos encontrar a opção **Caelum.Stella.CSharp**! A Stella é uma API .Net justamente para a validação e formatação de CPF, CNPJ, etc. Vamos clicar no botão "Install" para instalar.

Aparecerá uma tela de Aceitação de Licenças e clicaremos em "I Accept". Depois que foi instalado tudo o que era necessário, podemos fechar o NuGet. E assim, voltamos ao nosso código.

Então, instanciaremos o validador de CPF do Stella. Criaremos uma nova classe e importaremos o `using Caelum.Stella.CSharp.Validation` .

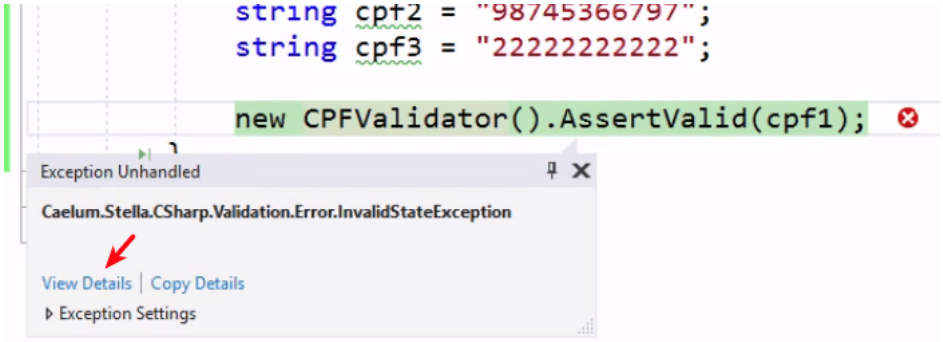
```
static void Main(string[] args)
{
    string cpf1 = "86288366757";
    string cpf2 = "98745366797";
    string cpf3 = "22222222222";

    new CPFValidator().AssertValid
}
```

Chamamos o método de validação `AssertValid()` passando como parâmetro, o primeiro CPF.

```
    new CPFValidator().AssertValid(cpf1);
```

Podemos rodar a aplicação para ver se está tudo certo com o nosso código. Não aconteceu nada, não é mesmo? Isso é porque o `AssertValid()` não informa se o CPF é válido. Entretanto, se alterarmos o último dígito, e rodarmos novamente a aplicação, veremos que dará uma exceção `InvalidStateException` . Clicando em "View Details":



Veremos a mensagem "Dígito de verificação inválido", ou seja, o `AssertValid()` serve para permitir que o seu programa prossiga na execução caso o CPF seja válido, entretanto, se ele for inválido, ele lançará uma exceção. Para isso, precisamos tratar e encapsular a linha do `AssertValid()` dentro de um bloco `try catch` para ter o controle, e não permitir que a minha aplicação pare de executar por causa disso.

```
try
{
    new CPFValidator().AssertValid(cpf1);
}
catch (Exception)
{
    throw;
}
```

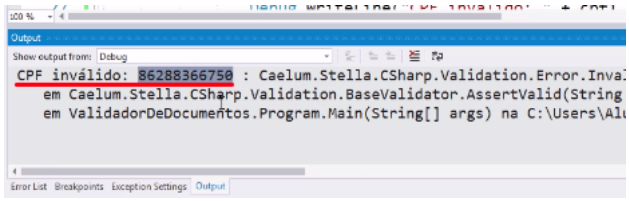
Adicionaremos um "Debug" logo abaixo do `AssertValid()` , acertamos a referência importando o `using System.Diagnostics` . Depois, incluiremos o método `.WriteLine()` , pois queremos que uma frase seja exibida:

```
try
{
    new CPFValidator().AssertValid(cpf1);
    Debug.WriteLine("CPF válido: " + cpf1);
}
```

Queremos também uma mensagem no `catch()` caso o meu CPF seja inválido. Acrescentaremos uma variável de exceção:

```
catch (Exception exc)
{
    Debug.WriteLine("CPF inválido: " + cpf1 + " : " + exc.ToString());
}
```

Muito bem, agora vamos rodar a aplicação. Expandindo a janela *Output*, teremos a informação:



Além de nos mostrar o número do CPF que é inválido, ainda temos a informação do motivo do CPF estar inválido.

Antes de tudo, vamos voltar o número 7 sendo ele o último dígito do primeiro CPF. Testaremos os outros CPFs para ver se eles são válidos ou não. Como não queremos uma repetição de método, vamos **extraí-lo**.

Selecionamos todo o `try catch` e extrai o método com o "Quick Actions and Refactorings... > Extract Method", e chamaremos esse novo método de `ValidarCPF()` :

```
static void Main(string[] args)
{
    string cpf1 = "86288366757";
    string cpf2 = "98745366797";
    string cpf3 = "22222222222";

    ValidarCPF(cpf1);
}

private static void ValidarCPF(string cpf)
{
    try
    {
        new CPFValidator().AssertValid(cpf);
        Debug.WriteLine("CPF válido: " + cpf);
    }
    catch (Exception exc)
    {
        Debug.WriteLine("CPF inválido: " + cpf + " : " + exc.ToString());
    }
}
```

Agora, simplesmente podemos copiar a linha que invoca o método, e passar os dois CPFs restantes:

```
static void Main(string[] args)
{
    string cpf1 = "86288366757";
    string cpf2 = "98745366797";
    string cpf3 = "22222222222";

    ValidarCPF(cpf1);
    ValidarCPF(cpf2);
    ValidarCPF(cpf3);
}
```

Após rodarmos novamente o projeto, vemos no *Output* que o primeiro CPF é válido, e os dois últimos não! Também vimos que o segundo CPF trouxe o motivo de "Dígito de verificação inválido" e o terceiro trouxe o motivo "Dígito repetido". Em que um CPF válido não pode conter números repetidos.