06

Formatando Documentos

Transcrição

Nesta primeira aula, estamos vendo como fazer validações de documentos como **CPF**, **CNPJ** e **Título Eleitoral**. **Mas, por que é importante validar documentos?**

Porque pode acontecer do usuário digitar algo errado por distração ou proposital, e mais pra frente isso pode nos dar dor de cabeça. Então, precisamos *garantir* que o nosso sistema tenha somente documentos e informações válidas!

Veremos uma outra peculiaridade agora: **a formatação desses documentos**. Cada um deles têm um formato específico, por exemplo, o CPF tem 11 dígitos, já o CNPJ tem 14 dígitos, e por fim, o Título Eleitoral tem 12 dígitos.

Entretanto, cada documento é exibido na tela ou no formulário de forma diferente, com (`.`) e (`-`) chamados de **separadores** cada um com as suas características.

Adicionaremos algumas linhas para mostrar como é a formatação desses documentos. Em primeiro lugar, trabalharemos com o **CPF**.

Imprimiremos o `cpf1` no console do nosso *Debug*.

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

    Debug.WriteLine(cpf1);
}
```

Rodando a aplicação, o `cpf1` original será exibido na tela. O próximo passo é exibir esse CPF já formatado. Já que temos a biblioteca **Caelum.Stella.CSharp** instalada em nosso projeto, podemos simplesmente usar o *formatador de CPF* dessa biblioteca. Para isso, **instanciaremos** um novo *Formatador de CPF*.

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

    Debug.WriteLine(cpf1);
    new CPFFormatter().Format(cpf1);
}
```

Depois de instanciar, dentro de `CPFFormatter()`, chamamos o método `Format()` passando o `cpf1`.

Como esse método retorna uma string, vamos imprimi-la no console do Debug.

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

    Debug.WriteLine(cpf1);
    Debug.WriteLine(new CPFFormatter().Format(cpf1));
}
```

Quando rodamos a aplicação, temos o CPF formatado no *Output* aparecendo dessa maneira:

```
862.883.667-57
```

Mas, o que acontece se tentamos formatar novamente esse CPF já formatado?

Vamos fazer algumas mudanças. Armazenaremos `new CPFFormatter().Format(cpf1)` em uma **variável local**. Selecionando esse conteúdo, clicaremos com o botão direito, logo em seguida em "Quick Actions and Refactorings...", e *introduziremos* essa variável local para essa expressão.

Automaticamente, aparecerá a variável `v` do tipo string, onde vamos modificar seu nome para `cpfFormatado`.

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

    Debug.WriteLine(cpf1);
    string cpfFormatado = new CPFFormatter().Format(cpf1);
    Debug.WriteLine(cpfFormatado);
}
```

Legal! Pegaremos esse CPF formatado e o formataremos novamente, chamando a instância do `CPFFormatter()` e chamaremos também o `Format()` para formatar o que já estava formatado. No final, mandaremos o resultado para o console do Debug.

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

    Debug.WriteLine(cpf1);
    string cpfFormatado = new CPFFormatter().Format(cpf1);
    Debug.WriteLine(cpfFormatado);
    Debug.WriteLine(new CPFFormatter().Format(cpfFormatado));
}
```

Rodando a aplicação, o resultado será um CPF sem formatação, o segundo CPF com a formatação, e por último o mesmo CPF reformatado. O que podemos entender com isso?

Que não houve mudanças! Podemos garantir a segurança de que se rodarmos mais uma vez o método `Format()`, não vamos gerar uma exceção ou estragar o nosso código. Essa é uma característica da computação chamada de *Idempotência*, ou seja, o mesmo método foi aplicado mais de uma vez, e ele obteve o mesmo resultado da primeira aplicação.

Aprenderemos agora a fazer o caminho inverso, que é pegar o **CPF Formatado**, e transformá-lo de volta ao seu valor original, sem estar formatado.

Chamaremos o formatador de CPF, e chamaremos o método inverso, que é o `Unformat()`, passando o cpfFormatado como parâmetro. Não podemos nos esquecer de chamar o método do console do Debug.

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

    Debug.WriteLine(cpf1);
    string cpfFormatado = new CPFFormatter().Format(cpf1);
    Debug.WriteLine(cpfFormatado);
    Debug.WriteLine(new CPFFormatter().Format(cpfFormatado));
    Debug.WriteLine(new CPFFormatter().Unformat(cpfFormatado);)
}
```

Após rodar a aplicação, temos o resultado do CPF **desformatado**, na forma original.

Como seria essa formatação para o CNPJ?

Da mesma forma! Porém usaremos um método diferente. Primeiro, podemos imprimir o `cnpj1` no console, para comparações posteriores. Segundo, vamos imprimir o mesmo CNPJ, porém já formatado.

```
static void Main(string[] args)
{
    // formatações do cpf

    Debug.WriteLine(cnpj1);
    Debug.WriteLine(new CNPJFormatter().Format(cnpj1));
}
```

Temos esse resultado:

```
5124175800015
51.241.758/0001-52
```

Repare que o CNPJ, diferente do CPF, tem uma barra que separa um grupo de quatro dígitos.

Agora, falta aplicarmos a formatação para o **Título Eleitoral**.

```
        static void Main(string[] args)
        {
            // formatações do cpf

            // formatação do cnpj

            Debug.WriteLine(titulo1);
            Debug.WriteLine(new TituloEleitoralFormatter().Format(titulo1));
        }
```

Quando rodamos a aplicação, temos o Título Eleitoral formatado:

```
041372570132
0413725701/32
```

Nesta aula, vimos como formatar um CPF, CNPJ e um Título Eleitoral. E por que é importante saber como formatar, tal como *desformatar* esses documentos? Pois geralmente, nós armazenamos esses documentos sem a formatação em nosso banco de dados. E na hora de exibir esses dados, sempre exibimos para o usuário, com a sua formatação, pois assim facilita a leitura. Nas próximas aulas, veremos como trabalhar com números.