01

Trabalhando com Dinheiro

Transcrição

Veremos como trabalhar com dinheiro utilizando **C#**. Criaremos um novo projeto do tipo *Console App* que se chamará **Dinheiro**!

Definiremos como um projeto inicial (*StartUp Project*). Então, como trabalharemos com dinheiro. E como vamos fazer isso? Geralmente, em **C#**, trabalhamos com o *decimal*, porém vamos trabalhar com o Money.

Adicionaremos uma referência em nosso projeto Dinheiro. Com o botão direito, clicamos em "Manage NuGet Package", e vamos procurar a referência para a Caelum, instalaremos e vamos aceitar a licença!

Para começar, declararemos um novo tipo: `Money` :

```
static void Main(string[] args)
{
    Money money = 10.00;
}
```

Sabemos que `10.00` é um tipo *Double*. Temos a informação de que existe um **operador implícito** na estrutura *Money* do `Caelum Stella CSharp` . Esse operador implícito nos permite atribuir um valor *Double* a um valor *Money*. Podemos fazer isso sem trabalhar com construtores, simplesmente utilizando o operador de atribuição `=` .

Em seguida, vamos exibi-lo no console com o `Debug.WriteLine(money)` , e imprimir o dinheiro que acabamos de criar.

```
static void Main(string[] args)
{
    Money money = 10.00;
    Debug .WriteLine(money);
}
```

Rodaremos o projeto para ver o resultado.

```
R$10,00
```

Apareceu para nós o valor de 10 reais com o `R` e o `$` e também com a vírgula. É interessante pois não precisamos colocar o símbolo **R$** e nem a vírgula. Essa é uma das *vantagens* de trabalhar com a **estrutura money** em relação ao **decimal**. Pois com ele, teríamos que formatar o número, colocando o `$` e vírgulas, para adaptá-lo ao modelo brasileiro.

O que mais podemos fazer com o money?

Podemos somar valores! Como exemplo, definiremos agora dois valores *Double* e uma variável para a soma:

```
static void Main(string[] args)
{
```

```
    Money money = 10.00;
    Debug .WriteLine(money);

    double valor1 = 10.00;
    double valor2 = 20.00;
    Money total = valor1 + valor2;
    Debug.WriteLine(total);
}
```

Fazendo isso, temos o resultado de:

```
R$30,00
```

Perceba que esse valor também está formatado, apesar de ter sido exibido com o `R$` , não significa que o `money` seja uma string! Ele é um valor realmente, e podemos fazer operações com ele. Adicionaremos uma **subtração** a nossa `main` .

Em vez de **Double**, trabalharemos com **Decimal**! O `minuendo` é o valor original de onde será subtraído um outro valor.

```
static void Main(string[] args)
{
    Money money = 10.00;
    Debug .WriteLine(money);

    double valor1 = 10.00;
    double valor2 = 20.00;
    Money total = valor1 + valor2;
    Debug.WriteLine(total);

    decimal minuendo = 20m;
}
```

O "m" minúsculo simboliza o decimal.

Em seguida, acrescentaremos o valor que vai ser subtraído dele. Depois jogaremos essa diferença em outra variável do tipo `Money` .

```
static void Main(string[] args)
{
    Money money = 10.00;
    Debug .WriteLine(money);

    double valor1 = 10.00;
    double valor2 = 20.00;
    Money total = valor1 + valor2;
    Debug.WriteLine(total);

    decimal minuendo = 20m;
    decimal subtraendo = 15m;
    Money diferenca = minuendo - subtraendo;
```

```
    Debug.WriteLine(diferenca);
}
```

Como resultado, temos:

```
R$5,00
```

Agora imagine que temos que trabalhar com outro tipo de moeda, por exemplo, o **euro**, como faremos?

Primeiramente, precisamos declarar uma variável do tipo `Money`, que chamaremos de `euro`. Neste caso, precisaremos passar também no construtor o valor e o tipo da moeda, que será um enumerador do tipo **Currency.EUR**.

```
static void Main(string[] args)
{
    //soma
    //subtração

    Money euro = new Money(Currency.EUR, 1000);
    Debug.WriteLine(euro);
}
```

Será impresso assim:

```
1.000,00 €
```

Veremos que ele já formatou para nós com o símbolo € no lugar correto. Isso é interessante para trabalhar com moedas diferentes. Imagine que agora temos que imprimir **1000 dólares**.

Da mesma forma, definiremos uma variável `dolar` do tipo `Money`, passando a moeda dólar com o **Currency.USD**.

```
static void Main(string[] args)
{
    //soma
    //subtração

    Money euro = new Money(Currency.EUR, 1000);
    Debug.WriteLine(euro);

    Money dolar = new Money(Currency.USD,1000);
    Debug.WriteLine(dolar);
}
```

Em nosso console, aparecerá como:

```
$1.000,00
```

Sabemos que o formato americano usa esses separadores em lugares diferentes. Como podemos usá-lo?

Temos que mudar a **Cultura** da Thread atual, e com o `.` acessamos a classe `DefaultThreadCurrentCulture`, e atribuímos uma nova `CultureInfo()` passando a cultura americana.

```
static void Main(string[] args)
{
    //soma
    //subtração

    Money euro = new Money(Currency.EUR, 1000);
    Debug.WriteLine(euro);

    Money dolar = new Money(Currency.USD,1000);
    Debug.WriteLine(dolar);

    CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("en-US");
    Debug.WriteLine(dolar);
}
```

Como isso aparecerá na tela?

```
$1.000,00
$1,000.00
```

Então foi impresso no console, o valor de 1000 dólares no formato americano, sendo a vírgula para separar os milhares e o ponto para separar os centavos.

Para voltar para o formato brasileiro, copiaremos a linha do `CultureInfo` e trocar de "en-US" para "pt-BR". E se quisermos somar **duas moedas diferentes**? O que acontece se somarmos euro com dólar?

Declaramos uma nova variável que chamaremos de `somaMoedasDiferentes`. Essa variável irá conter a soma de `euro + dolar`.

```
static void Main(string[] args)
{
    //soma
    //subtração

    Money euro = new Money(Currency.EUR, 1000);
    Debug.WriteLine(euro);

    Money dolar = new Money(Currency.USD,1000);
    Debug.WriteLine(dolar);

    CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("en-US");
    Debug.WriteLine(dolar);
    CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("pt-BR");

    Money somaMoedasDiferentes = euro + dolar;
    Debug.WriteLine(somaMoedasDiferentes);
}
```

Após rodar o projeto, vimos que houve uma exceção de que a moeda de ambos argumentos, deve coincidir para que essa operação seja efetuada. Com isso, concluímos que não podemos realizar operações com moedas diferentes. Isso é muito bom, pois em nosso dia a dia, também não podemos realizar operações com moedas distintas. Temos que convertê-las para depois trabalhar com o resultado.

Nesta aula, vimos como trabalhar com dinheiro utilizando a biblioteca `Caelum Stella CSharp`, que fornece esse tipo `Money` - bastante útil para trabalharmos com conversão de moedas -, impedindo operações entre moedas diferentes. Ela também facilitará a exibição, formatando e colocando o símbolo da moeda. Temos a vantagem de ter o arredondamento automático.

Na próxima aula, trabalharemos com CEPs e obtendo endereços da internet, a partir de um CEP formado que será enviado para um serviço. O mesmo retornará o resultado para nós, nos permitindo utilizá-lo na aplicação.