01

Substituir Método por Objeto Método

Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://github.com/alura-cursos/csharp-refatorando-codigo/archive/60fdf55b7c6b33b88f03c138c9b4d912a4c4f119.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

A oitava técnica de refatoração se refere a **Substituição do método por método-objeto**. Abrindo o Visual Studio encontraremos o arquivo `Produto.cs` no projeto `refatoracao`, localizado em "Aula04 > R08.ReplaceMethodWithMethod > depois". A classe `Produto` é responsável por armazenar informações do produto.

Temos também o método `Preco()` que nos retorna um valor decimal, e que tem muitas responsabilidades.

```
decimal Preco(decimal precoBase, decimal acrescimo, decimal desconto)
{
    var resultado = precoBase;

    if (this.promocional && desconto > 0)
    {
        throw new Exception("Produto já é promocional e não pode ter desconto!")
    }

    if (desconto >  20)
    {
        desconto = 20;
    }

    f (acrescimo > 15)
    {
        acrescimo = 15;
    }

    return precoBase + precoBase * (acrescimo - desconto);
}
```

Vimos que essa classe está acumulando responsabilidades demais, pois além de armazenar informações do produto, ela também realiza o calculo do preço. A nossa tarefa é *aliviar* a carga dessa classe, transportando a funcionalidade de cálculo de preço para uma outra classe.

Para isso, criaremos uma nova classe que conterá a funcionalidade que pertencia ao método `Preco()`. No final desse arquivo, o `Produto.cs`, criaremos a `CalculadoraDePrecos`. Em seguida, declararemos alguns campos dessa classe.

```
class Produto
{
    // escopo da classe
}

class CalculadoraDePrecos
```

```
{

}
```

O primeiro campo que precisamos declarar, é a instância de `Produto` :

```
class CalculadoraDePrecos
{
    private readonly Produto produto;
}
```

Agora vamos pegar os parâmetros do método `Preco()` , e transformá-los em **campos** da classe `CalculadoraDePrecos` :

```
class CalculadoraDePrecos
{
    private readonly Produto produto;
    private decimal precoBase;
    private decimal acrescimo;
    private decimal desconto;
}
```

É preciso criar um construtor para a classe - o que faremos facilmente com a ajuda do Visual Studio. Selecionamos os campos, clicamos na lâmpada, onde encontraremos a opção "Generate constructor":

```
class CalculadoraDePrecos
{
    private readonly Produto produto;
    private decimal precoBase;
    private decimal acrescimo;
    private decimal desconto;

    public CalculadoraDePrecos(Produto produto, decimal precoBase, decimal acrescimo, decima
    {
        this.produto = produto;
        this.precoBase = precoBase;
        this.acrescimo = acrescimo;
        this.desconto = desconto;
    }
}
```

O próximo passo é criar o método de cálculo do preço logo abaixo do construtor. O seu corpo executará o que o método antigo `Preco()` executa atualmente:

```
public decimal Calcular()
{
    var resultado = precoBase;

    if (this.promocional && desconto > 0)
    {
        throw new Exception("Produto já é promocional e não pode ter desconto!")
```

```
        }

        if (desconto >  20)
        {
            desconto = 20;
        }

        f (acrescimo > 15)
        {
            acrescimo = 15;
        }

        return precoBase + precoBase * (acrescimo - desconto);
    }
```

Como percebemos, o método dependia da propriedade privada `promocional` do objeto `produto`, e por isso, ocorreu o erro de compilação em nosso código. Como temos a instância de `produto`, podemos substituir a referência `this` pela referência `produto`:

```
    public decimal Calcular()
    {
        var resultado = precoBase;

        if (produto.Promocional && desconto > 0)
        {
            throw new Exception("Produto já é promocional e não pode ter desconto!")
        }

        // demais ifs
```

E assim, temos o calculo do produto. No método original, `Preco()`, vamos eliminar todas as linhas, e retornaremos uma nova instância da `CalculadoraDePrecos()` passando os parâmetros. Depois, chamaremos o método `Calcular()`:

```
    decimal Preco(decimal precoBase, decimal acrescimo, decimal desconto)
    {
        return new CalculadoraDePrecos(this, precoBase, acrescimo, desconto).Calcular()
    }
```

Com isso, conseguimos com sucesso, realizar a refatoração de substituir o método pelo método-objeto.