06

Substituir Variável Por Método de Consulta

Transcrição

Vamos para a quinta técnica, conhecida como **substituir variável por consulta**. Para esta, usaremos o projeto `caelum-stella-csharp`. Dentro da pasta `Http`, encontraremos a classe `CEP.cs`, onde encontramos o construtor:

```
public CEP(string cepAsString)
{
    string unformattedCEP = cepAsString.Replace("-", "");
    if (cepAsString == null)
        this.cepAsString = null;
    else if (Regex.IsMatch(cepAsString, RegexFormatted))
        this.cepAsString = unformattedCEP;
    else if (new Regex(RegexUnformatted).IsMatch(cepAsString))
        this.cepAsString = cepAsString;
    else
        throw new InvalidZipCodeFormat();
}
```

Como vemos, há algumas condições para montar e validar um CEP. A variável `unformattedCEP` guarda o CEP não formatado (desformatado). Ela pegará o CEP formatado e removerá o hífen, substituindo-o por uma String vazia.

Existe algum problema de usar a variável `unformattedCEP`? Nenhum. No entanto, se quisermos utilizar a expressão de desformatação `cepAsString.Replace("-", "")` em um outro local da classe além do Construtor?

Desta vez, não daria certo porque a variável `unformattedCEP` é **local**, o que significa: **ela está no escopo do construtor dessa classe**. Então, para utilizar a mesma expressão em outro lugar, seria necessário **mover a variável `unformattedCEP` para o escopo da classe**.

Mesmo assim, teríamos problema pois o método `Replace()` só será executado uma vez.

Para obter sempre o CEP formatado ou desformatado a partir de um *CEP formatado*, é preciso utilizar **um método** no lugar de uma variável temporária, assim como estamos fazendo no código. Chamaremos esse método de *Consulta*, pois é um método utilizado para obter valor.

A nossa tarefa será extrair um método da expressão `cepAsString.Replace("-", "")`. Ao selecionar essa expressão, clicaremos na lâmpada ao lado para "Extrair método". Vamos chamá-lo de `UnformatCEP()`, uma função que desformatará o CEP formatado.

```
public CEP(string cepAsString)
{
    string unformattedCEP = UnformatCEP(cepAsString);
    // ifs e elses
}

private static string UnformatCEP(string cepAsString)
{
    return cepAsString.Replace("-", "");
}
```

Agora esse método já pode ser utilizado em outros lugares da classe. Ainda temos uma tarefa pendente, precisamos eliminar a necessidade da variável temporária `unformattedCEP`, que recebe o valor vindo da consulta.

Clicaremos na variável, e depois escolher a opção "*Inline temporary variable*" clicando na lâmpada. Ao selecionar essa opção, o Visual Studio automaticamente elimina a variável e substitui pela chamada direta ao método `UnformatCEP()`, os lugares onde a variável estava situada.

Quando alteramos o código, é importante sempre lembrar-se de executar os testes de unidade para ter certeza de que nenhuma funcionalidade foi quebrada. Clicaremos em "Test > Run > All Tests".

Os testes foram executados com sucesso. Agora, recapitularemos o assunto abordado até o momento. Nós vimos a refatoração **Extrair Variável**, que é utilizada quando temos uma expressão complexa, ou quando temos algum código que podemos armazenar um resultado temporário dentro de uma variável.

Fazemos isso para **explicar o que as expressões estão executando dentro do código**. Podemos pegar expressões que estão dentro de `If`s, ou dentro de uma expressão maior, e então quebrá-la em pequenas variáveis temporárias que podem facilitar a leitura.

Também temos o "porém" quando temos variáveis demais, podendo prejudicar a limpeza da classe.

Abordamos a função inversa que é **Incorporar Uma Variável Temporária** sempre que alguma variável é muito simples.

A terceira técnica de refatoração que vimos, é **Substituir uma variável por consulta a método**, pois a variável temporária está limitada ao escopo em que ela foi declarada, seja um construtor ou um método. Por um outro lado, o método de consul pode ser consumido em qualquer lugar de uma classe.