01

# Extrair Classe

### Transcrição

Começando deste ponto? Você pode fazer o DOWNLOAD (https://github.com/alura-cursos/csharp-refatorando-codigo/archive/c7616c5559602e5105feb3bf3d9dc3472d66a7e3.zip) completo do projeto do capítulo anterior e continuar seus estudos a partir deste capítulo.

---

Nesse capítulo, veremos a décima segunda técnica de refatoração chamada de **Extrair Classe**.

Imagine uma criança brincando com vários brinquedos espalhados pelo chão, e então pedimos para que ela guarde todos os brinquedos em uma caixa.


Da mesma maneira, também podemos fazer o mesmo com o nosso código, espalhando dados por uma classe e depois colocá-los novamente dentro de uma nova classe. A técnica de extrair classes resolve vários problemas, por exemplo, quando uma classe é muito grande e está fazendo o trabalho de duas, violando o *princípio da responsabilidade única*.

Um outro problema quando a classe é muito longa, além do problema de leitura, é a *duplicação de dados*. Com a extração de classe conseguimos resolvê-lo. Também podemos ter dados aglomerados, onde uma classe extraída organizará melhor esses dados. Por último, a *obsessão por tipos primitivos* também é um problema, pois podemos nos esquecer facilmente da orientação a objetos.

Faremos a extração de classes, mas tomando cuidado para não extrair demais e gerar classes supérfluas.

No projeto `refatorcao` dentro do Visual Studio, temos a pasta "Aula06 > R12.ExtractClass > depois" que guarda o arquivo `Empresa.cs` que contém uma série de informações. Imagine que essa empresa tenha `EndEntrega` e `EndComercial`. São dois grupos de informações que deveriam estar separados.

```
class Empresa
{
    public Empresa(string razaoSocial, string cNPJ, string endEntregaLogradouro, string endEntr
    {
```

```
            RazaoSocial = razaoSocial;
            CNPJ = cNPJ;
            EndEntregaLogradouro = endEntregaLogradouro;
            EndEntregaNumero = endEntregaNumero;
            EndEntregaComplemento = endEntregaComplemento;
            EndEntregaBairro = endEntregaBairro;
            EndEntregaCEP = endEntregaCEP;
            EndEntregaMunicipio = endEntregaMunicipio;
            EndEntregaUF = endEntregaUF;
            EndComercialLogradouro = endComercialLogradouro;
            EndComercialNumero = endComercialNumero;
            EndComercialComplemento = endComercialComplemento;
            EndComercialBairro = endComercialBairro;
            EndComercialCEP = endComercialCEP;
            EndComercialMunicipio = endComercialMunicipio;
            EndComercialUF = endComercialUF;
        }

        public string RazaoSocial { get; private set; }
        public string CNPJ { get; private set; }

        // propriedades do endereço de entrega

        // propriedades do endereço comercial
    }
```

Como podemos ver, existe uma duplicação de dados muito grande, pois a própria classe `Empresa` está tentando gerenciar informações de *endereço*. Uma grande parte dessa classe é dedicada a gerenciar endereços. Podemos evitar que a classe fique muito longa, a duplicação e código e a obsessão por tipos primitivos, como `string` que está sendo muito usado, criando uma nova classe chamada `Endereco` no mesmo arquivo, logo abaixo da classe `Empresa` .

Para a nova classe, moveremos as partes que ela irá conter, como o bloco de propriedades do endereço de entrega:

```
class Empresa {...}

class Endereco
{
    public string EndEntregaLogradouro { get; private set }
    public string EndEntregaNumero { get; private set }
    public string EndEntregaComplemento { get; private set }
    public string EndEntregaBairro { get; private set }
    public string EndEntregaCEP { get; private set }
    public string EndEntregaMunicipio { get; private set }
    public string EndEntregaUF { get; private set }
}
```

Vamos remover a parte em que se refere ao **endereço de entrega**, deixando as propriedades mais genéricas:

```
class Empresa {...}

class Endereco
{
    public string Logradouro { get; private set }
```

```
    public string Numero { get; private set }
    public string Complemento { get; private set }
    public string Bairro { get; private set }
    public string CEP { get; private set }
    public string Municipio { get; private set }
    public string UF { get; private set }
}
```

Criaremos um *construtor* para a classe `Endereco`. Selecionamos essas propriedades, e com o atalho "Ctrl + .", o Visual Studio nos permite gerar um construtor:

```
class Endereco
{
    public Endereco(string logradouro, string numero, string complemento, string bairro, string
    {
        Logradouro = logradouro;
        Numero = numero;
        Complemento = complemento;
        Bairro = bairro;
        CEP = cEP;
        Municipio = municipio;
        UF = uF;
    }

    public string Logradouro { get; private set }
    public string Numero { get; private set }
    public string Complemento { get; private set }
    public string Bairro { get; private set }
    public string CEP { get; private set }
    public string Municipio { get; private set }
    public string UF { get; private set }
}
```

Eliminaremos o trecho que contém as propriedades do *endereço de entrega* na classe `Empresa` e a substituiremos por uma nova propriedade pública `EndEntrega`:

```
    public string RazaoSocial { get; private set; }
    public string CNPJ { get; private set; }

    public Endereco EndEntrega { get; private set }

        // propriedades do endereço comercial
```

Faremos o mesmo com as propriedades do endereço comercial. Vamos selecionar o trecho que contém as propriedades do endereço comercial em `Empresa`, e utilizar o atalho "Ctrl + X". Logo depois de ter recortado, no mesmo lugar adicionaremos um propriedade pública referente ao endereço comercial:

```
    public string RazaoSocial { get; private set; }
    public string CNPJ { get; private set; }

    public Endereco EndEntrega { get; private set }
```

```
    public Endereco EndComercial { get; private set }
```

Após essa mudança, geramos uma série de problemas pois a classe `Empresa` não estava preparada para isso. No construtor da `Empresa`, instanciaremos um novo endereço de entrega e um endereço comercial:

```
class Empresa
{
    public Empresa(string razaoSocial, string cNPJ, string endEntregaLogradouro, string endEntre
    {
        RazaoSocial = razaoSocial;
        CNPJ = cNPJ;
        EndEntrega = new Endereco(endEntregaLogradouro, endEntregaNumero, endEntregaComplemento
        EndComercial = new Endereco(endComercialLogradouro, endComercialNumero, endComercialCom

        EndEntregaLogradouro = endEntregaLogradouro;
        EndEntregaNumero = endEntregaNumero;
        EndEntregaComplemento = endEntregaComplemento;
        // outros objetos.
    }
```

Com essa mudança, eliminamos uma série de código desnecessário. Confira como o construtor da `Empresa` está após essas mudanças:

```
public Empresa(string razaoSocial, string cNPJ, string endEntregaLogradouro, string endEntregaN
    {
        RazaoSocial = razaoSocial;
        CNPJ = cNPJ;
        EndEntrega = new Endereco(endEntregaLogradouro, endEntregaNumero, endEntregaComplemento
        EndComercial = new Endereco(endComercialLogradouro, endComercialNumero, endComercialCom
    }
```

Temos o método chamado de `UpdateEnderecoEntrega()` e o método `UpdateEnderecoComercial()` na `Empresa`. O que iremos fazer agora é passar esses métodos para a classe `Endereco`. A fim de realizar uma refatoração mais inteligente, reaproveitaremos o construtor de `Endereco` para extrair um novo método chamado `Update()`.

```
class Endereco
{
    public Endereco(string logradouro, string numero, string complemento, string bairro, string
    {
        Update(logradouro, numero, complemento, bairro, cEP, municipio, uF);
    }

    public void Update(string logradouro, string numero, string complemento, string bairro, str
    {
        Logradouro = logradouro;
        Numero = numero;
        Complemento = complemento;
        Bairro = bairro;
        CEP = cEP;
        Municipio = municipio;
```

```
            UF = uF;
        }
    }
```

Depois disso, podemos consumir o método `Update()` a partir da classe `Empresa`, passando os parâmetros que recebemos em `UpdateEnderecoEntrega()`.

```
class Empresa
{
    public void UpdateEnderecoEntrega(string endEntregaLogradouro, string endEntregaNumero, str
    {
        EndEntrega.Update(endEntregaLogradouro, endEntregaNumero, endEntregaComplemento, endEnt
        EndEntregaLogradouro = endEntregaLogradouro;
        EndEntregaNumero = endEntregaNumero;
        EndEntregaComplemento = endEntregaComplemento;
        EndEntregaBairro = endEntregaBairro;
        EndEntregaCEP = endEntregaCEP;
        EndEntregaMunicipio = endEntregaMunicipio;
        EndEntregaUF = endEntregaUF;
    }
}
```

Dessa forma, não será mais necessário os dados que virão a seguir. O método ficará enxuto:

```
class Empresa
{
    public void UpdateEnderecoEntrega(string endEntregaLogradouro, string endEntregaNumero, str
    {
        EndEntrega.Update(endEntregaLogradouro, endEntregaNumero, endEntregaComplemento, endEnt
    }
}
```

Faremos o mesmo processo com o método `UpdateEnderecoComercial()`:

```
class Empresa
{
    public void UpdateEnderecoEntrega(string endEntregaLogradouro, string endEntregaNumero, str
    {
        EndEntrega.Update(endEntregaLogradouro, endEntregaNumero, endEntregaComplemento, endEnt
    }

    public void UpdateEnderecoComercial(string endComercialLogradouro, string endComercialNumer
    {
        EndComercial.Update(endComercialLogradouro, endComercialNumero, endComercialComplemento
    }
}
```

Para fechar, também temos que resolver os problemas que surgiram nos métodos `GetTextoEnderecoEntrega()` e em `GetTextoEnderecoComercial()`. Vamos copiar o corpo do método de qualquer um deles, para a classe `Endereco` em um

novo método chamado `GetTexto()`, deixando-o bem genérico:

```
class Endereco
{
    public void Update(){...}

    public string GetTexto()
    {
        return $"Endereço: {Logradouro} {Numero} {Complemento} - {Bairro} - CEP {CEP} - {Municip
    }
}
```

Resolvido! Nos métodos `GetTextoEnderecoComercial()` e `GetTextoEnderecoEntrega()`, vamos chamar o `GetTexto()` da classe `Endereco`:

```
    public Endereco EndEntrega { get; private set; }

    public Endereco EndComercial { get; private set; }

    public string GetTextoEnderecoComercial()
    {
        return EndComercial.GetTexto();
    }

    public string GetTextoEnderecoEntrega()
    {
        return EndEntrega.GetTexto();
    }
```

Podemos compilar a aplicação com o atalho "Ctrl + Shift + B". O nosso projeto foi compilado com sucesso.