08

Faça o que eu fiz na aula

Mover Método

Na pasta mover_metodo, temos três arquivos, Correntista.php, ContaCorrente.php e index.php. No vídeo, nós vimos que o método `exibeNomeCorrentista` está na classe `ContaCorrente` e faz uso de dois campos da classe `Correntista`, ele não utiliza nenhum dado da `ContaCorrente` e não faz sentido estar nesta classe.

Além disso ele também viola o princípio Diga, não pergunte (Tell, don't ask), que é um dos princípios principais da orientação a objetos, quebrando o encapsulamento da classe `Correntista` expondo diretamente os seus campos.

Vamos resolver isso aplicando a refatoração Mover Método, e vamos criar um método com o nome de `getNomeCompleto` na classe correntista:

```
public function getNomeCompleto(): string
{
    return $this->nome . " " . $this->sobrenome;
}
```

Agora no método `exibeNomeCorrentista` da classe `ContaCorrente`, vamos apagar a chamada aos campos, e chamar o método que acabamos de criar:

```
public function exibeNomeCorrentista(): string
{
    return $this->correntista->getNomeCompleto();
}
```

Se acessarmos a página pelo navegador agora, podemos ver que o comportamento continua igual ao anterior à refatoração.

Mover Campo

Na pasta mover_campo, temos um arquivo index.php, Contato.php e Usuario.php. No vídeo, vimos que na classe Usuario, existe um método chamado `imprimirInformacoes`, e nele estão sendo feitas as impressões de texto na tela do usuário:

```
public function imprimirInformacoes(): void
{
    echo "<p>-- Informações de contato --</p>";
    echo "<p>Usuário: " . $this->nome . "</p>";
    echo "<p>Email: " . $this->email . "</p>";
    echo $this->contato;
}
```

Só que vimos que o e-mail não faz muito sentido estar nesta classe, já que ele é uma informação de contato, seria mais organizado se ele ficasse dentro da classe `Contato`. Para isso, podemos utilizar uma refatoração chamada Mover Campo.

Vamos remover o e-mail da classe `Usuario`, vamos começar pela impressão na tela, que deve ficar assim:

```
    public function imprimirInformacoes(): void
    {
        echo "<p>-- Informações de contato --</p>";
        echo "<p>Usuário: " . $this->nome . "</p>";
        echo "<p>Email: " . $this->email . "</p>";
        echo $this->contato;
    }
```

Vamos também remover o método `getEmail`, já que ele não será mais utilizado.

```
    public function getEmail(): string
    {
        return $this->email;
    }
```

Agora ainda falta remover o campo e o parâmetro no construtor, que devem ficar assim:

```
    private $nome;
    private $sobrenome;
    private $contato;

    public function __construct(string $nome, string $sobrenome, Contato $contato)
    {
        $this->nome = $nome;
        $this->sobrenome = $sobrenome;
        $this->contato = $contato;
    }
```

Na classe `Contato`, podemos adicionar o campo `email`, adicionar a atribuição desse campo no construtor e imprimí-lo no método `__toString`.

```
class Contato
{
    private $endereco;
    private $telefone;
    private $email;

    public function __construct(string $endereco, string $telefone, string $email)
    {
        $this->endereco = $endereco;
        $this->telefone = $telefone;
        $this->email = $email;
    }

    public function __toString()
    {
        return "<p>Email: $this->email</p>
                <p>Endereço: $this->endereco</p>
                <p>Telefone: $this->telefone</p>";
    }
}
```

Após a alteração na classe, devemos alterar também as instâncias no arquivo index.php:

```
$contato = new Contato('Rua Vergueiro 3185', '115571-2751', "giovanni@alura.com.br");
$usuario = new Usuario('Giovanni', 'Tempobono', $contato);
```

E se recarregarmos a página no navegador, devemos ter o mesmo comportamento, indicando que a refatoração foi bem sucedida.

Extrair Classe

Na pasta extrair_classe, temos um arquivo index.php e um arquivo chamado Usuario.php contendo uma classe.

No vídeo, vimos que esta classe Usuario.php tem muitas responsabilidades, ela quebra o princípio da responsabilidade única, uma boa prática de desenvolvimento de software que diz que uma classe só deve ter uma razão para ser modificada.

Neste caso, vamos acabar modificando a classe se precisarmos trocar algum cadastro do usuário ou algum cadastro de informações de contato, então vamos separar em duas classes, começando marcando quais são os campos que devem ser extraídos:

```
class Usuario
{
    private $nome;
    private $sobrenome;
    private $endereco; //extrair
    private $cep; //extrair
    private $telefone; //extrair
    private $tipoTelefone; //extrair
    private $ddd; //extrair
```

Então temos agora que extrair todas essas propriedades da classe `Usuario` para uma nova classe chamada Contato.php, atribuindo pelo construtor e criando os getters:

```
class Contato
{
    private $endereco;
    private $cep;
    private $telefone;
    private $tipoTelefone;
    private $ddd;

    public function __construct(string $endereco, string $cep, string $telefone, string $tipoTelefor
    {
        $this->endereco = $endereco;
        $this->cep = $cep;
        $this->telefone = $telefone;
        $this->tipoTelefone = $tipoTelefone;
        $this->ddd = $ddd;
    }

    public function getTipoTelefone(): string
    {
```

```
            return $this->tipoTelefone;
        }

        public function getEnderecoECep(): string
        {
            return "$this->endereco $this->cep";
        }

        public function getTelefoneDdd(): string
        {
            return "($this->ddd) $this->telefone";
        }
    }
```

Agora na classe `Usuario`, precisamos remover as referências que temos para aquelas propriedades que adicionamos na classe `Contato`, então vamos remover os seguintes campos e adicionar um para a classe `Contato`:

```
    private $nome;
    private $sobrenome;
    private $contato;
```

Vamos fazer a mesma modificação no construtor da classe Usuario:

```
    public function __construct(string $nome, string $sobrenome, Contato $contato)
    {
        $this->nome = $nome;
        $this->sobrenome = $sobrenome;
        $this->contato = $contato;
    }
```

Podemos também remover os métodos getter desses campos, pois eles não existem mais. Também modificaremos os argumentos do arquivo index.php para casar com os parâmetros da classe.

```
$contato = new Contato("Rua Vergueiro 3185", "04101-300", "5571-2751", "celular", "11");
$usuario = new Usuario("Giovanni", "Tempobono", $contato);
```

Com isso temos um código mais desacoplado, respeitando o princípio da responsabilidade única, e se recarregarmos a página no navegador, teremos o mesmo comportamento, indicando o sucesso da refatoração.

Incorporar Classe

Vamos abrir a classe Contato da pasta incorporar_classe e vamos tentar aplicar a refatoração Extrair Classe nela, vamos extrair o `ddd` e o `telefone` para uma nova classe chamada `Telefone` e o `cep` para uma nova classe chamada Cep.

Porém, vimos no vídeo que criar uma classe só para um campo é um grande exagero, pois não apresenta nenhuma vantagem em questão de legibilidade, adicionando complexidade desnecessária.

Podemos aplicar a refatoração chamada Incorporar Classe, para mover o campo `cep` da sua própria classe para a classe `Contato` novamente.

Então a classe `Cep` deixa de existir, e a classe `Contato` fica com o campo `Cep`:

```
class Contato
{
    private $endereco;
    private $cep;

    public function __construct(string $endereco, string $cep)
    {
        $this->endereco = $endereco;
        $this->cep = $cep;
    }

    public function getEndereco(): string
    {
        return $this->endereco;
    }

    public function getCep(): string
    {
        return $this->cep;
    }
}
```

E se recarregarmos a página, devemos ter o mesmo resultado no navegador.