07

Faça o que eu fiz na aula

Ocultar Delegado

Neste vídeo, vamos utilizar o projeto que está dentro da pasta ocultar_delegado, nesta pasta, temos três classes, a classe `Gerente` , `Departamento` e `Pessoa` , onde uma depende da outra. Para ser criada, a classe `Pessoa` depende de `Departamento` , e `Departamento` depende de `Gerente` .

No arquivo index.php, temos os requires destas classes, instanciamos um objeto `Pessoa` que tem um `Departamento` e um `Gerente` chamado José.

Se precisarmos encontrar o nome do gerente temos que chamar um método antes para encontrar o departamento, esse tipo de encadeamento de mensagens é um problema, pois quando precisarmos de um objeto teremos que chamar outro, para chamar outro objeto, e chamar outro, e assim vai. Isso viola a Lei de Demeter, que estabelece que cada objeto só pode acessar o objeto que está próximo a ele.

Para resolvermos isso, podemos usar uma refatoração chamada Esconder Delegado, vamos ao arquivo Usuario.php e dentro da classe usuário, criar um método chamado `getNomeDoGerente` :

```
public function getNomeDoGerente(): string
 {
     return $this->departamento->getGerente()->getNome();
 }
```

Neste método vamos ocultar aquela chamada de métodos gigante que estávamos fazendo, vamos chamar o departamento, pegar o gerente e retornar o nome dele como uma string.

Com este método feito, podemos simplificar a API pública da nossa classe Usuario, e no nosso arquivo index.php, vamos mudar a chamada de método para somente este método novo.

```
$maria = new Pessoa(new Departamento(new Gerente('José')));

echo $maria->getNomeDoGerente();
```

Remover Intermediário

Neste vídeo vamos utilizar o código que está dentro da pasta remover_intermediario, dentro dessa pasta temos uma classe `Funcionario` e uma classe `Empregado` .

Para instanciarmos um novo empregado, temos que instanciar também um empregado, se formos na definição da classe `Empregado` , vamos ver que ela precisa de um objeto `Funcionario` no construtor, mas os dois métodos dessa classe `Empregado` só chamam propriedades da classe `Funcionario` .

Essa classe não tem nenhum comportamento próprio que a justifique, a classe `Empregado` está sendo somente um intermediário para chamarmos os métodos de `Funcionario` .

Vimos no vídeo que essa é uma boa oportunidade para refatorar. Podemos simplificar o nosso código utilizando uma refatoração chamada Remover Intermediário, vamos deletar a classe `Empregado` e no nosso arquivo index.php, vamos chamar diretamente a classe `Funcionario` .

```
$maria = new Funcionario('Giovanni', 'Instrutor');

echo "<p>{$maria->getNome()};</p>";
echo "<p>{$maria->getCargo()};</p>";
```

Com isso, removemos uma complexidade desnecessária da nossa classe e fizemos com que ele fique mais enxuto.

Substituir Número Mágico

Neste vídeo vamos para a pasta substituir_numero_magico, e nessa pasta, temos dois arquivos PHP, um arquivo index.php e um arquivo CalculadoraDeSalario.php. No arquivo index.php, é incluída e instanciada a classe CalculadoraDeSalario com um salário base, e depois, é chamado um método que aplica os descontos no salário.

Nessa classe, temos um campo chamado salário, que é inicializado pelo construtor, e temos uma função de aplicar descontos.

Nesta função será pego o salário, e descontado 8% do salário e depois 7,5% do salário. Aí vem a pergunta, o que são estes descontos? porque 8% e 7,5? o que eles significam? o código não nos ajuda muito a descobrir isso, não é?

Sempre que utilizamos estes números dessa maneira isso é algo que vai dificultar a leitura do código posteriormente, afetando também a manutenção futura do sistema.

Vimos no vídeo que podemos utilizar uma refatoração chamada Substituir Número Mágico para nos ajudar, para fazermos esta refatoração, vamos declarar nessa classe, dois campos de constantes:

```
private const INSS = 0.08;
private const IR = 0.075:
```

E depois, podemos trocar estes números dentro do método para `self::IR` e `self::INSS`

```
    public function aplicaDescontos()
    {
        return $this->salario - ($this->salario * self::INSS) - ($this->salario * self::IR);
    }
```

Assim o código fica mais legível e mais fácil de entender, e se acessarmos pelo navegador, o código vai se comportar da mesma maneira.