**Reescrita e Polimorfismo**

**Reescrita de método**

Precisamos exibir mais uma informação em nossa listagem de produtos, que é o valor de imposto de cada um deles. A regra é bem simples, o imposto do produto é equivalente a 19,5% de seu valor. Podemos calcular isso na própria listagem, no lugar de onde era calculado o imposto, como por exemplo:

<tr>

<td><?= $produto->getNome() ?></td>

<td><?= $produto->getPreco() ?></td>

<td><?= $produto->preco \* 0.195 ?></td>

<td><?= substr($produto->getDescricao(), 0, 40) ?></td>

<td><?= $produto->getCategoria()->getNome() ?></td>

<!-- restante do html omitido -->

Mas já vimos que expor nossa regra de negócio dessa forma não é legal, seria muito mais interessante encapsular esse comportamento em um método. Assim evitamos duplicação de código e facilitamos a manutenção e evolução desse comportamento.

Vamos criar o método calculaImposto, e apagar o valorComDesconto, na classe Produto:

public function calculaImposto() {

return $this->preco \* 0.195;

}

Perfeito, para testar vamos mudar nosso produto-lista.php para mostrar o valor do imposto com esse novo método:

<tr>

<td><?= $produto->getNome() ?></td>

<td><?= $produto->getPreco() ?></td>

<td><?= $produto->calculaImposto() ?></td>

<td><?= substr($produto->getDescricao(), 0, 40) ?></td>

<td><?= $produto->getCategoria()->getNome() ?></td>

<!-- restante do html omitido -->

Mas há uma nova regra. Quando o produto for um livro, seu desconto será de 6.5%. Se deixarmos o código como está, um livro terá o desconto de 19.5%, já que um livro **é um produto**, ou seja, herda esse comportamento. Como corrigir isso?

Dentre algumas outras formas, poderíamos criar um outro método na classe Livro, chamadocalculaImpostoDeLivro, e fazer um if no HTML para chamar esse método caso o produto iterado seja do tipoLivro. O código ficaria parecido com:

<tr>

<td><?= $produto->getNome() ?></td>

<td><?= $produto->getPreco() ?></td>

<td><?= substr($produto->getDescricao(), 0, 40) ?></td>

<td>

<?php

if ($produto->temIsbn()) {

echo $produto->calculaImpostoDeLivro();

// calcula 6.5% de imposto

}

else {

echo $produto->calculaImposto();

// calcula 19.5% de imposto

}

?>

</td>

<td><?= substr($produto->getDescricao(), 0, 40) ?></td>

<!-- restante do html omitido -->

Mas além de confundir a todos que forem usar esse método, precisaríamos lembrar de escrever esse if sempre que fosse necessário saber o desconto do produto. Não parece uma solução ideal. Além disso, já pensou em como ficaria esse código conforme novos tipos de produto com regras de imposto diferentes forem aparecendo? Um caos.

O que queremos na verdade é modificar o comportamento herdado da **superclasse**. Podemos **reescrever** (ou**sobrescrever**, como também é comumente dito) todo comportamento que não quisermos herdar diretamente. É bem simples, basta criar o mesmo método (com a mesma assinatura) na classe Livro:

class Livro extends Produto {

private $isbn;

function getIsbn() {

return $this->isbn;

}

function setIsbn($isbn) {

$this->isbn = $isbn;

}

public function calculaImposto() {

return $this->preco \* 0.065;

}

}

Mas ainda há um problema. Ao executar esse código receberemos o seguinte erro:

Undefined property: Livro::$preco in

/Users/turini/Desktop/cursophp/loja

/Livro.php on line 9

Mas espera ai, o livro não tem preço? A resposta é **sim**, livro herda o atributo preço da classe Produto, porém não consegue acessá-lo dessa forma, afinal se trata de um atributo **privado**. E agora, precisamos mudar preço para**public**? Não precisamos e nem devemos quebrar o encapsulamento de nossas classes por causa disso. Há duas formas de resolver, uma delas seria modificando a visibilidade do preco para **protected** na classe Produto:

class Produto {

private $id;

private $nome;

protected $preco;

private $descricao;

private $categoria;

private $usado;

// restante do código

}

O protected é um meio termo entre public e private, onde apenas a classe **e também as subclasses** podem acessar o atributo. Assim o código já funcionará como estamos esperando. Mas e quanto à outra alternativa?

É ainda mais simples, basta não acessar a propriedade da classe mãe diretamente. Você pode consultar o preço do produto utilizando a **interface pública** da classe, isto é, seus métodos. No lugar de fazer:

public function calculaImposto() {

return $this->preco \* 0.065;

}

Podemos utilizar o método get:

public function calculaImposto() {

return $this->getPreco() \* 0.065;

}

Perfeito! Agora temos o mesmo resultado, mesmo mantendo a propriedade preço como private. Sempre prefira usar a interface pública da classe (*getters*, *setters* e outros métodos públicos) ao invés de afrouxar a regra do encapsulamento com a visibilidade protected, ainda que apenas as classes filhas acessem. Idealmente apenas a própria classe deveria acessar e modificar seu estado diretamente.

Vamos testar a alteração? Basta abrir a listagem. Agora os valores estão sendo calculados conforme esperamos, 6.5% para livros e 19.5% para os demais produtos.

**Invocando métodos da superclasse**

Você sempre pode utilizar a palavra reservada parent para invocar métodos estáticos da **superclasse**. Por exemplo, imagine que o imposto de um Livro precise ser o dobro do imposto da classe Produto. Até poderíamos ver o valor escrito no método da classe mãe e escrever o dobro dele na classe filha, mas e quando o valor da classe mãe atualizar? Precisaríamos lembrar de atualizar a classe filha. Não vai acontecer.

E que tal fazer assim:

class Livro extends Produto {

private $isbn;

// getIsbn e setIsbn

function calculaImposto() {

return parent::calculaImposto() \* 2;

}

}

Problema resolvido, com isso o imposto da classe Livro será o resultado do calculaImposto da classe Produto, que é o *parent*, multiplicado por dois.

**Polimorfismo**

Perceba que apesar de estarmos chamando o mesmo método na página, ele tem um comportamento diferente de acordo com o **tipo** da classe que foi instanciada, seja um Produto ou Livro. Sem esse recurso, seria necessário encadear uma série de ifs, um para cada tipo. O código ficaria parecido com:

<tr>

<td><?= $produto->getNome() ?></td>

<td><?= $produto->getPreco() ?></td>

<td><?= substr($produto->getDescricao(), 0, 15) ?></td>

<td>

<?php

if ($produto->temIsbn()) {

// calcula 6.5% de imposto

}

if ($produto->algumaOutraCoisa()) {

// calcula 12% de imposto

} else {

// calcula 19.5% de imposto

}

?>

</td>

<td><?= substr($produto->getDescricao(), 0, 40) ?></td>

<!-- restante do html omitido -->

A pior parte é que a cada novo tipo de produto, precisaríamos mudar esse código, adicionando mais um if. Isso seria um pesadelo, desde a verbosidade dessas várias linhas de código, complexidade, legibilidade, etc. Tudo isso reflete diretamente na manutenibilidade do código.

Agora veja da forma que está, com uma única chamada ao método calculaImposto. Dessa forma estamos aproveitamos a estrutura do objeto, que com certeza é um Produto, mas mudando apenas o que for preciso em suas implementações.

<td><?= $produto->calculaImposto() ?></td>

O PHP sempre vai procurar pelo método na classe mais específica, que foi instanciada com o *new*. Na ausência desse método, aí sim ele vai procurar na classe mãe. A ordem é sempre essa. Apesar de parecer algo muito simples, isso nos possibilita economizar montanhas de códigos como aqueles ifs todos que vimos. Esse recurso é conhecido como **polimorfismo**, que é a capacidade de um objeto poder ser referenciado de várias formas.