**Métodos**

**Definindo comportamentos com métodos**

Vimos que o uso de classes já torna algumas coisas mais interessantes em nosso sistema. Temos uma forma forte de representar elementos importantes em nosso código. Mas até agora nossas classes Produto e Categoria só tem propriedades, ou seja, só guardam estado. Uma classe também pode, e deve, ter comportamentos. Por exemplo, como faríamos para aplicar um desconto de 10% no valor real do produto? Uma forma simples de fazer seria:

* 1. $produto->preco -= $produto->preco \* 0.1;

Estamos subtraindo da propriedade $preco o seu próprio valor, multiplicado por 0.1. O código funciona, mas vamos copiar e colar em todos os lugares que precisamos dessa conta? Não é uma boa ideia. Há outro problema: O que acontecerá se algum desenvolvedor fornecer um valor negativo de desconto? Vamos adicionar juros ao produto! E se fosse um valor maior que 100%? O produto ficará com valor negativo! Um exemplo:

* 1. $produto = new Produto();
  2. $produto->preco = 59.90;
  3. $produto->preco -= $produto->preco \* 1.1; // -5,99
  4. $produto->preco -= $produto->preco \* - 1; // 119,8

Não podemos permitir que nenhuma dessas situações aconteça. Podemos criar uma função para que seja possível reutilizar esse comportamento uma vez definido, e dentro dela já validar que o valor de desconto é válido, antes de aplicar ao preço do produto:

* 1. function subtraiDesconto($produto)
  2. {
  3. $valor = 0.1;
  4. if ($valor > 0 && $valor < 1) {
  5. $produto->preco -= $produto->preco \* $valor;
  6. return $produto->preco;
  7. }
  8. }

Estamos no caminho certo, mas ainda há um problema. Onde essa função deverá ser definida? No arquivo que ela será usada? E se forem vários arquivos? Criamos um arquivo novo só pra ela? Adicionamos no banco-produto.php? Mas ela não é uma função de banco. Nenhuma das soluções parecem muito satisfatória, não é?

E que tal criamos essa função dentro da própria classe Produto? Dessa forma todos que tem acesso ao Produtoterão acesso a esse comportamento em comum. Além disso, a função já terá acesso fácil a propriedade $preco, também definida nessa classe. Essa é uma solução muito mais interessante do que deixar a função em um local desatrelado das informações do Produto. Assim mantemos estado e comportamento juntos.

Chamamos de **método** as funções que definimos dentro de uma classe, já que elas definem a forma (método) de fazer operações em determinado objeto.

Vamo fazer a alteração. A classe Produto deve ficar assim:

* 1. class Produto
  2. {
  3. public $id;
  4. public $nome;
  5. public $preco;
  6. public $descricao;
  7. public $categoria;
  8. public $usado = false;
  9. function subtraiDesconto()
  10. {
  11. $valor = 0.1;
  12. if ($valor > 0 && $valor < 1) {
  13. $this->preco -= $this->preco \* $valor;
  14. return $this->preco;
  15. }
  16. }
  17. }

Repare que no lugar de recebermos um $produto como argumento, agora estamos usando a pseudo variável$this para nos referenciar ao objeto que está chamando essa função. Para executar o método subtraiDesconto, também utilizaremos o ->. O termo usado para isso é **invocação de método**.

* 1. $produto->subtraiDesconto();

Podemos deixar esse método ainda mais mais flexivel, recebendo o valor do desconto como um argumento (parâmetro) no lugar de sempre aplicar o valor fixo de 0.1.

* 1. function subtraiDesconto($valor)
  2. {
  3. if ($valor > 0 && $valor < 1) {
  4. $this->preco -= $this->preco \* $valor;
  5. return $this->preco;
  6. }
  7. }

Podemos passar quantos argumentos, ou **parâmetros** como são chamados, forem necessários para nossos métodos. Um parâmetro nada mais é do que uma variável comum, chamada também de temporária ou local, pois, ao final da execução desse método, ela deixa de existir.

Agora, ao invocar o método subtraiDesconto, podemos passar o valor que deverá ser calculado. Um exemplo seria:

* 1. $produto->subtraiDesconto(0.1);

Vamos colocar esse código em prática na listagem de produtos. Podemos criar uma nova coluna, ao lado do valor real do produto, mostrando seu valor com desconto. No arquivo produto-lista.php:

* 1. <table class="table table-striped table-bordered">
  2. <?php
  3. $produtos = listaProdutos($conexao);
  4. foreach ($produtos as $produto) :
  5. ?>
  6. <tr>
  7. <td><?= $produto->nome ?></td>
  8. <td><?= $produto->preco ?></td>
  9. <td><?= $produto->subtraiDesconto(0.1) ?></td>
  10. <!-- continuação da tabela -->

**Para Saber Mais: Sobrecarga com parâmetros opcionais**

Um recursos interessante e que pode ser bastante útil são os parâmetros opcionais. Quando necessário, você pode atribuir um valor padrão para o parâmetro como a seguir:

* 1. function subtraiDesconto($valor=0.05)
  2. {
  3. if ($valor > 0 && $valor < 1) {
  4. $this->preco -= $this->preco \* $valor;
  5. return $this->preco;
  6. }
  7. }

Neste exemplo, caso o valor do desconto não tenha sido fornecido ele será de 5%. Esse recurso é bastante utilizado quando precisamos fazer uma sobrecarga de método, já que em PHP não podemos ter métodos com o mesmo nome.

Uma regra importante é que quando temos os dois tipos de parâmetros (opcionais e não opcionais) declarados, os opcionais tem que ser os ultimos na lista de parâmetros. Por exemplo:

* 1. function subtraiDesconto($paramObrigatorio, $valor=0.05)
  2. {
  3. // ...
  4. }

Declarar o $paramObrigatorio por último resultaria em um erro.