**Herança**

**Regras próprias de um Livro**

Livro é um tipo de produto que aparece com bastante frequência em nosso estoque, mas estamos deixando um detalhe importante passar. Todo livro precisa ter um número de **ISBN** (**I**nternational **S**tandard **B**ook **N**umber), que é seu identificador único. É fundamental manter e exibir essa informação.

Precisamos modificar a tabela produtos, adicionando uma nova coluna para o **isbn**. Podemos fazer isso acessando o phpMyAdmin e, na aba **SQL**, executando a seguinte instrução:

ALTER TABLE produtos ADD COLUMN isbn VARCHAR(255);

Vamos adicionar essa propriedade também na classe Produto, assim como seus métodos get e set. O código ficará assim:

class Produto {

// outras propriedades

private $isbn;

// demais métodos

public function getIsbn() {

return $this->isbn;

}

public function setIsbn($isbn) {

$this->isbn = $isbn;

}

}

Ótimo, já temos o novo campo na tabela e na classe. Mas como saber se o produto é um livro ou não? Afinal, só faz sentido manter e exibir o **isbn** se for um livro. Verificamos se o campo isbn é nulo? Não sabemos o que fazer. Uma forma simples seria adicionar mais uma coluna na tabela produtos, que vai indicar o tipo do produto que estamos trabalhando.

ALTER TABLE produtos ADD COLUMN tipoProduto VARCHAR(255);

Agora, na classe Produto, podemos adicionar a propriedade tipoProduto, assim como seus métodos get e set.

class Produto {

// outras propriedades

private $tipoProduto;

// demais métodos

public function setTipoProduto($tipoProduto) {

$this->tipoProduto = $tipoProduto;

}

public function getTipoProduto() {

return $this->tipoProduto;

}

}

Como já temos o tipoProduto declarado na classe, como ficamos sabendo se ele é ou não um livro? Podemos fazer um if comparando seu o tipo com "Livro", mas sempre que quisermos saber o tipo de um produto, precisaremos repetir esse mesmo if? Isso não é legal, não é? Duplicar código nunca é. Podemos criar um novo método chamado isLivro, que já faz essa comparação para nós. Assim essa lógica fica **encapsulada** na classeProduto:

public function isLivro() {

return $this->tipoProduto === "Livro";

}

Mas vamos tentar usar esse método quando o tipo do produto é livro, com l minúsculo. Rode um exemplo para testar:

$produto = new Produto("", "", "", new Categoria, "");

$produto->setTipoProduto("livro");

echo $produto->isLivro();

A comparação não será verdadeira, por quê? É simples, quando comparamos dessa forma, com o ===, ele leva em consideração o *case* da string (é *case-sensitive*). Em outras palavras, **l**ivro, com "l" minúsculo, será diferente de **L**ivro.

Uma alternativa que o PHP nos oferece de comparar strings, é utilizando os métodos strcmp e strcasecmp, sendo esse último para comparações *case-insensitive*. Vamos tentar:

function isLivro() {

return strcasecmp($this->tipoProduto, "Livro") == 0;

}

Veja que estamos comparando o resultado dessa função com 0 (zero). Isso é necessário pois, diferente do ===, que sempre retorna **true** ou **false**, o strcasecmp retorna zero quando os valores são iguais, um inteiro positivo quando for maior e um inteiro negativo quando a comparação for menor. Mas vamos utilizar a comparação do === mesmo, vocês verão o porquê.

Quase pronto, mas para que tudo funcione precisamos alterar os métodos de adicionar, alterar e listar os produtos. Vamos começar com o arquivo produto-formulario-base.php. Nele podemos adicionar mais dois *inputs*, um para o**isbn** e outro para o **tipoProduto**.

<tr>

<td>Tipo do produto</td>

<td>

<select name="tipoProduto" class="form-control">

<option value="Livro">Livro</option>

<option value="Produto">Produto</option>

</select>

</td>

</tr>

<tr>

<td>ISBN (caso seja um livro)</td>

<td>

<input type="text" name="isbn" class="form-control"

value="<?=$produto->getIsbn()?>" />

</td>

</tr>

Agora precisamos modificar o arquivo adiciona-produto.php, chamando o set desses dois campos novos. A parte que deve mudar ficará assim:

$nome = $\_POST['nome'];

$preco = $\_POST['preco'];

$descricao = $\_POST['descricao'];

$categoria = new Categoria;

$categoria->setId($\_POST['categoria\_id']);

if(array\_key\_exists('usado', $\_POST)) {

$usado = "true";

} else {

$usado = "false";

}

$isbn = $\_POST['isbn'];

$tipoProduto = $\_POST['tipoProduto'];

$produto = new Produto($nome, $preco, $descricao, $categoria, $usado);

$produto->setIsbn($isbn);

$produto->setTipoProduto($tipoProduto);

$produtoDao = new ProdutoDAO($conexao);

if($produtoDao->insereProduto($produto)) { ?>

// ...

Por fim, vamos mudar as funções insereProduto, alteraProduto e listaProdutos do nosso ProdutoDAO. No momento de inserir e alterar, vamos mudar o SQL adicionando os dados das colunas isbn e tipoProduto:

function insereProduto(Produto $produto) {

$query = "insert into produtos (nome, preco, descricao, categoria\_id,

usado, isbn, tipoProduto) values ('{$produto->getNome()}',

{$produto->getPreco()}, '{$produto->getDescricao()}',

{$produto->getCategoria()->getId()}, {$produto->isUsado()},

'{$produto->getIsbn()}', '{$produto->getTipoProduto()}')";

return mysqli\_query($this->conexao, $query);

}

function alteraProduto(Produto $produto) {

$query = "update produtos set nome = '{$produto->getNome()}',

preco = {$produto->getPreco()}, descricao = '{$produto->getDescricao()}',

categoria\_id= {$produto->getCategoria()->getId()},

usado = {$produto->isUsado()}, isbn = '{$produto->getIsbn()}',

tipoProduto = '{$produto->getTipoProduto()}'

where id = '{$produto->getId()}'";

return mysqli\_query($this->conexao, $query);

}

Na listagem, basta chamar o *setter* desses dois campos:

function listaProdutos() {

$produtos = array();

$resultado = mysqli\_query($this->conexao, "select p.\*,c.nome as categoria\_nome

from produtos as p join categorias as c on c.id=p.categoria\_id");

while($produto\_atual = mysqli\_fetch\_assoc($resultado)) {

$id = $produto\_atual['id'];

$nome = $produto\_atual['nome'];

$preco = $produto\_atual['preco'];

$descricao = $produto\_atual['descricao'];

$categoria = new Categoria;

$categoria->setNome($produto\_atual['categoria\_nome']);

$usado = $produto\_atual['usado'];

$isbn = $produto\_atual['isbn'];

$tipoProduto = $produto\_atual['tipoProduto'];

$produto = new Produto($nome, $preco, $descricao, $categoria, $usado);

$produto->setId($id);

$produto->setIsbn($isbn);

$produto->setTipoProduto($tipoProduto);

array\_push($produtos, $produto);

}

return $produtos;

}

Quase pronto, agora só falta adicionar a nova coluna no arquivo produto-lista.php.

<tr>

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

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

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

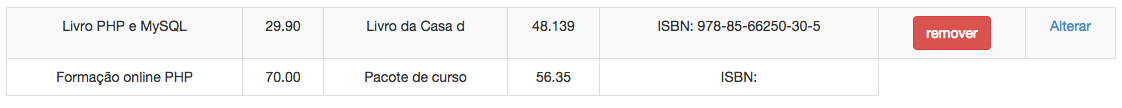
<td><?= substr($produto->getDescricao(), 0, 40) ?></td>

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

<td> ISBN: <?= $produto->getIsbn() ?></td>

<!-- continuação do código -->

Parece tudo certo, mas ao testar:



Veja que a tabela quebrou na segunda linha. O seguinte erro foi exibido no terminal:

/loja/produto-lista.php - Call to undefined method

Produto::getIsbn() in /Users/turini/Desktop/cursophp

/loja/produto-lista.php on line 17

Faz sentido, afinal só podemos chamar o getIsbn de um Livro, o Produto não tem esse método. E agora?

Já criamos o método isLivro para nos ajudar nesse trabalho, lembra? Só precisamos de um if chamando esse método condicionando a chamada do getIsbn:

<tr>

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

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

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

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

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

<td>

<?php

if ($produto->isLivro()) {

echo "ISBN: ".$produto->getIsbn();

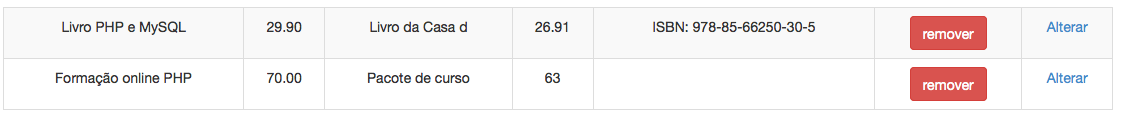
}

?>

</td>

<!-- continuação do código -->

Vamos testar? Podemos adicionar um novo livro chamado Livro PHP e MySQL, com preço 29.90, uma descrição qualquer e isbn 978-85-66250-30-5. Ao abrir a listagem:



Sucesso, a nova coluna com o número de ISBN foi adicionada, apenas no produto que é do tipo Livro. Mas será que essa forma que fizemos é a melhor opção?

**Herança entre classes**

Nosso código está funcionando, mas o problema dessa nossa alternativa é que agora temos o **isbn** para todos os produtos. É totalmente possível alguém esquecer de chamar o método isLivro e chamar o getIsbn para um produto que não é livro. Estranho, não é?

Em outras palavras, o problema é que a classe Produto agora tem propriedades e métodos que só dizem respeito aos produtos do tipo Livro. Se Livro é tão importante e tem informações e características próprias, nada mais justo do que criar uma classe para representar esse tipo especial de produto. Em orientação a objetos, podemos e devemos criar classes para representar os elementos importantes de nosso contexto, como é o caso do Produto, da Categoria e agora do Livro.

Vamos tirar os atributos isbn e tipoProduto, assim como seus *getters* e *setters*, da classe Produto, já que essa classe deve manter o estado e comportamento apenas do que for válido para todos os tipos de produto. O isbnserá agora propriedade de uma nova classe, que vamos chamar de Livro:

class Livro {

private $isbn;

function getIsbn() {

return $this->isbn;

}

function setIsbn($isbn) {

$this->isbn = $isbn;

}

}

Perfeito! Mas espera um pouco, um objeto do tipo Livro também vai precisar de um nome, preco, descricao e todas as outras propriedades que já existem na classe Produto. Afinal, livro **é um** produto. Uma forma de resolver esse problema seria copiando e colando as propriedades e métodos, mas veja como o código fica repetitivo!

class Livro {

private $id;

private $nome;

private $preco;

private $descricao;

private $categoria;

private $usado;

private $isbn;

function getIsbn() {

return $this->isbn;

}

function setIsbn($isbn) {

$this->isbn = $isbn;

}

// todos os outros métodos, getters e setters

}

Além disso, se tivéssemos mais outro tipo de produto que tem características diferentes do produto comum, seria necessário criar uma outra classe e copiar todo esse código novamente! Imagine como seria trabalhoso evoluir o código, ao adicionar uma propriedade na classe Produto, teríamos que lembrar de replicar essa mudança em todos os outros tipos de produto.

Em orientação a objetos, há uma forma muito mais elegante de resolver esse problema. Podemos relacionar as classes Produto e Livro, dizendo que um Livro tem tudo (**herda**) o que a classe Produto tiver. Podemos dizer que um Livro é uma **extensão** de Produto.

class Livro extends Produto {

private $isbn;

function getIsbn() {

return $this->isbn;

}

function setIsbn($isbn) {

$this->isbn = $isbn;

}

}

Com isso, estamos estabelecendo uma relação de classe mãe (**superclasse**) e classe filha (**subclasse**), que neste caso são ,respectivamente, Produto e Livro. A partir do momento que dissemos que um Livro é um (**extends**)Produto, podemos chamar qualquer método que tenha sido declarado na classe Produto a partir de uma instância de Livro. Veja um exemplo:

$livro = new Livro("Livro de PHP e MySQL", "", "", new Categoria, "");

echo "O nome é ". livro->getNome();

Mesmo sem ter esses métodos diretamente declarados em sua classe, uma instância de livro pode usar os *getters* e*setters* de todas as propriedades declaradas em sua **superclasse** Produto.

Um detalhe importante é que não existe herança multipla em PHP, isso significa que você só pode herdar de uma única classe. Claro, isso não te impede de encadear heranças, ou seja, herdar de uma classe que herda de outra.**Uma classe pode ter várias filhas, mas pode ter apenas uma mãe**.

**O mau uso da herança**

Herança é um recurso bastante interessante e que pode trazer uma série de benefícios para nosso projeto, mas**cuidado para não abusar desse recurso**. Imagine que a classe Pessoa tenha um ou mais comportamentos em comum com a classe Produto. Fazemos com que Produto herde de Pessoa para aproveitar esses métodos? Isso não seria nem um pouco interessante, já que Pessoa tem nome e outras propriedades que não fazem sentido paraProduto. No contexto real, um produto não é uma pessoa. Por que em nosso código seria?

Vamos colocar essa mudança em prática? Podemos começar pelo produto-formulario-base.php, o value do *input*do **ISBN** está invocando o getIsbn, mas ele agora só existe se o produto for um Livro, então temos que colocar essa condição:

<tr>

<td>Tipo do produto</td>

<td>

<select name="tipoProduto" class="form-control">

<option value="Livro">Livro</option>

<option value="Produto">Produto</option>

</select>

</td>

</tr>

<tr>

<td>ISBN (caso seja um livro)</td>

<td>

<input type="text" name="isbn" class="form-control"

value="<?php if ($produto->isLivro()) { echo $produto->getIsbn(); } ?>" />

</td>

</tr>

Podemos começar pela adição de produtos. No arquivo adiciona-produto.php, podemos fazer um if para verificar o tipo do produto. Assim, damos *new* no Livro e setamos o isbn quando houver, ou damos *new* emProduto nas demais situações. O código pode ficar assim:

$nome = $\_POST['nome'];

$preco = $\_POST['preco'];

$descricao = $\_POST['descricao'];

$categoria = new Categoria;

$categoria->setId($\_POST['categoria\_id']);

$isbn = $\_POST['isbn'];

$tipoProduto = $\_POST['tipoProduto'];

if(array\_key\_exists('usado', $\_POST)) {

$usado = "true";

} else {

$usado = "false";

}

if ($tipoProduto == "Livro") {

$produto = new Livro($nome, $preco, $descricao, $categoria, $usado);

$produto->setIsbn($isbn);

} else {

$produto = new Produto($nome, $preco, $descricao, $categoria, $usado);

}

$produtoDao = new ProdutoDAO($conexao);

if($produtoDao->insereProduto($produto)) { ?>

// ...

Já podemos testar. Primeiro vamos adicionar um produto do tipo Livro, com o isbn preenchido. O produto deve ser adicionado conforme esperado. Ótimo! Mas ao testar adicionar um produto que não seja Livro, recebemos o seguinte erro:

Call to undefined method Produto::getIsbn()

in /Users/turini/Desktop/cursophp/loja/

banco-produto.php on line 36

Ops, estamos chamando o método getIsbn sempre, independente se o produto é ou não um Livro! Vamos corrigir, antes de escrever a *query*, basta verificar se o tipo do parâmetro $produto tem **isbn**. Mas e para pegar o tipo do produto? Não temos mais o método getTipoProduto. Para isso, basta pegar a classe do produto, através do método get\_class, passando o produto por parâmetro:

function insereProduto(Produto $produto) {

if ($produto->temIsbn()) {

$isbn = $produto->getIsbn();

} else {

$isbn = "";

}

$tipoProduto = get\_class($produto);

$query = "insert into produtos (nome, preco, descricao, categoria\_id,

usado, isbn, tipoProduto) values ('{$produto->getNome()}',

{$produto->getPreco()}, '{$produto->getDescricao()}',

{$produto->getCategoria()->getId()}, {$produto->isUsado()},

'{$isbn}', '{$tipoProduto}')";

return mysqli\_query($this->conexao, $query);

}

Fazemos a mesma coisa no método de alteração:

function alteraProduto(Produto $produto) {

if ($produto->temIsbn()) {

$isbn = $produto->getIsbn();

} else {

$isbn = "";

}

$tipoProduto = get\_class($produto);

$query = "update produtos set nome = '{$produto->getNome()}',

preco = {$produto->getPreco()}, descricao = '{$produto->getDescricao()}',

categoria\_id= {$produto->getCategoria()->getId()},

usado = {$produto->isUsado()}, isbn = '{$isbn}',

tipoProduto = '{$tipoProduto}'

where id = '{$produto->getId()}'";

return mysqli\_query($this->conexao, $query);

}

Mas sem nos esquecer de também alterar o altera-produto.php:

$id = $\_POST['id'];

$nome = $\_POST['nome'];

$preco = $\_POST['preco'];

$descricao = $\_POST['descricao'];

$categoria = new Categoria;

$categoria->setId($\_POST['categoria\_id']);

$isbn = $\_POST['isbn'];

$tipoProduto = $\_POST['tipoProduto'];

if(array\_key\_exists('usado', $\_POST)) {

$usado = "true";

} else {

$usado = "false";

}

if ($tipoProduto == "Livro") {

$produto = new Livro($nome, $preco, $descricao, $categoria, $usado);

$produto->setIsbn($isbn);

} else {

$produto = new Produto($nome, $preco, $descricao, $categoria, $usado);

}

$produto->setId($id);

$produtoDao = new ProdutoDAO($conexao);

if($produtoDao->alteraProduto($produto)) { ?>

// ...

O método temIsbn precisa verificar se o produto em questão é uma instância de Livro. Podemos fazer isso com oinstanceof. Vamos remover o isLivro e fazer:

class Produto {

// todo o código omitido

function temIsbn() {

return $this instanceof Livro;

}

}

**Lembre-se de trocar por temIsbn() onde estava utilizando o isLivro().**

Teste mais uma vez para garantir que tudo está funcionando. Perfeito, agora conseguimos adicionar, mas e a listagem? Ao tentar acessá-la, recebemos um erro:

::1:56458 [500]: /loja/produto-lista.php -

Call to undefined method Produto::setTipoProduto()

in /Users/turini/Desktop/cursophp/loja/banco-produto.php

on line 23

Faz sentido, ainda não mudamos o método listaProdutos. Precisamos instanciar um Livro, se o seu tipo for um "Livro", ou Produto no caso contrário. O código ficará bem parecido com o que já fizemos. Algo como:

function listaProdutos() {

$produtos = array();

$resultado = mysqli\_query($this->conexao, "select p.\*,c.nome as categoria\_nome

from produtos as p join categorias as c on c.id=p.categoria\_id");

while($produto\_atual = mysqli\_fetch\_assoc($resultado)) {

$id = $produto\_atual['id'];

$nome = $produto\_atual['nome'];

$preco = $produto\_atual['preco'];

$descricao = $produto\_atual['descricao'];

$categoria = new Categoria;

$categoria->setNome($produto\_atual['categoria\_nome']);

$usado = $produto\_atual['usado'];

$isbn = $produto\_atual['isbn'];

$tipoProduto = $produto\_atual['tipoProduto'];

if ($tipoProduto == "Livro") {

$produto = new Livro($nome, $preco, $descricao, $categoria, $usado);

$produto->setIsbn($isbn);

} else {

$produto = new Produto($nome, $preco, $descricao, $categoria, $usado);

}

$produto->setId($id);

array\_push($produtos, $produto);

}

return $produtos;

}

Tudo certo, mas ainda falta atualizar o HTML. Não podemos mais fazer aquele if com o isLivro, já que esse método não existe mais. No lugar disso vamos usar o método temIsbn, lá no produto-lista.php:

<td>

<?php

if ($produto->temIsbn()) {

echo "ISBN: ".$produto->getIsbn();

}

?>

</td>

Fechado, agora nosso código todo voltou a funcionar, faltando apenas mudar o produto-altera-formulario.php, mas isso faremos no exercício :)

Vamos colocar isso tudo em prática?

**Acoplamento**

Essa é uma discussão um pouco mais avançada, mas desde já perceba que o uso de herança aumenta o**acoplamento** entre as classes, isto é, o quanto uma classe depende de outra. Como há uma relação muito forte entre a classe mãe e filha, acaba sendo necessário conhecermos muitos dos detalhes de implementação da classe mãe para que tudo funcione como esperado. E o problema vai além, imagine que um certo dia a classe mãe adicione um método com o nome igual a um que já existe na classe filha. Quando alguém chamar esse método, esperando o comportamento definido na classe mãe, poderá na verdade estar chamando o da classe filha. Isso pode bagunçar toda a lógica. Em outras palavras, a classe mãe também precisa conhecer detalhes de implementação de suas filhas - que podem ainda nem existir.

Veremos mais à frente alguns outros recursos que nos ajudam a resolver esse problema.