EBOOK

ETEBRAS
Escola Técnica do Brasil

Orientação a objeto

Bruno Silva

Versão 1, janeiro/2015

ESCOLA TÉCNICA DO BRASIL – ETEBRAS

Diretora-Presidente
Maria Antonieta Alves Chiappetta

Diretor Administrativo Financeiro
Marco Antônio Rodrigues

Diretor de Educação a Distância
George Bento Catunda

Secretária Escolar
Maria Patrícia Farias Bastos

Professor Autor
Bruno Silva

Imagem da Capa
http://developingux.com/wp-content/uploads/2010/01/image_thumb.png

Sumário

Introdução

Prezado (a) aluno (a),

Com programação podemos construir diversos aplicativos para facilitar e enriquecer nossas experiências do dia a dia. Esta disciplina visa justamente mostrar a vocês como construir sistemas que têm como meio as páginas web, utilizando o paradigma orientado a objetos.

Ela também pretende ampliar os conceitos aprendidos na matéria de programação em PHP, pois utiliza a noção de orientação a objetos como forma de modelar os problemas a serem resolvidos. Desta forma, este caderno apresenta um material introdutório sobre programação orientada a objetos (OO) utilizando a linguagem PHP. Portanto, é essencial que você entenda bem os conceitos básicos da linguagem PHP e seja capaz de criar programas nesta linguagem, utilizando um ambiente de programação e teste. Então, caso nas disciplinas anteriores tenha ficado alguma dúvida, releia os cadernos de estudo e procure na internet outras explicações para ajudar na compreensão. O sucesso depende de sua dedicação.

O conteúdo do caderno é dividido em três partes principais.

Introdução, onde são apresentados de forma introdutória os conceitos fundamentais do paradigma orientado a objetos e um pouco de sua história.

Orientação a Objetos, na qual são mostrados os conceitos fundamentais do paradigma OO como, por exemplo, métodos e atributos. Conceitos relacionados

à reutilização de código onde são mostradas técnicas de reutilização de código largamente utilizadas nesse tipo de paradigma (Orientada a Objetos).

Por fim, uma atividade de projeto é proposta para que você aplique os conhecimentos aprendidos nesta disciplina.

Capítulo 01: Princípios e importância da orientação a objetos.

Este material apresenta uma introdução sobre programação orientada a objetos, utilizando a linguagem de programação PHP. Neste material, assumiremos que o leitor possua conhecimentos prévios relacionados à programação web (HTML, CSS, PHP estruturado).

Para pessoas que sabem programar em uma linguagem de programação estruturada, como C ou PHP básico, pode surgir uma pergunta.

"Por que eu aprenderia uma forma diferente de programar (i.e., utilizando objetos), se até então nunca surgiu essa necessidade?"

Se o projeto envolver muitas funcionalidades (muitas rotinas) e algumas delas possuírem um comportamento similar, a orientação a objetos é uma boa alternativa a ser utilizada nesse projeto. A utilização de orientação a objetos permite ao sistema uma maior **reusabilidade**, **manutenibilidade** e **qualidade** do código.

Histórico e comparação com linguagens estruturadas.

Em 2004, o modelo de programação orientado a objetos foi adicionado ao PHP 5 e permitiu que aplicações WEB complexas fossem desenvolvidas de uma forma simples, modular e reutilizável.

Com o surgimento do release PHP 5, programadores finalmente tinham o mesmo poder de programação que linguagens proeminentes como o Java e C#. Desta forma, PHP finalmente possuiria uma infraestrutura de programação orientada a objetos (POO) **completa**.

Em linguagens estruturadas costumamos programar tentando 'reproduzir' a forma como um computador 'pensa'. Ou seja, programamos uma sequência de comandos para executar a programação. A partir da execução dessa sequência de comandos, temos o resultado da computação.

Por outro lado, a linguagem de programação orientada a objetos é um paradigma de computação que se assemelha à forma de pensar dos seres humanos. Para realizar a computação, ao invés de uma série de comandos, utilizamos objetos que possuem dados associados e se comunicam entre si.

Durante os anos 70 e 80, a metodologia de engenharia de software predominante era a ***programação estruturada***, que se baseava nos seguintes princípios: sequência, decisão e interação.

A sequência quer dizer a ordem principal de execução de um programa, ou seja, o conjunto de passos para execução do programa. A decisão envolve questões que possam surgir no meio do caminho. Geralmente, existe um fluxo principal na execução do programa, mas pode haver desvios. Os mecanismos de

decisão auxiliam nessa tarefa. Já a iteração permite que uma mesma função seja chamada várias vezes durante a execução do programa.

Abaixo, uma das formas de resolver certo problema utilizando o modelo de programação estruturada.

Para resolver um problema complexo, quebre este problema em pequenas partes e trabalhe nessas partes separadamente.

Para resolver cada parte, trate este problema como um novo problema e se possível quebre este em novos subproblemas.

Repita este processo até que cada subparte esteja resolvida e junte todo o resultado.

Essa abordagem de programação também é conhecida como abordagem *top-down* (cima para baixo). Apesar de ser uma técnica bastante útil na escrita de código estruturado, a abordagem *top-down* possui algumas limitações como listadas a seguir.

Esta é focada quase que inteiramente em produzir **instruções** necessárias para se resolver um problema. A modelagem das estruturas de dados que compõe o problema geralmente é deixada em segundo plano na fase de projeto do sistema.

É bastante **complicado reutilizar um código** escrito utilizando essa abordagem em outros projetos, dado que os problemas a serem resolvidos são bastante específicos para o domínio do sistema projetado.

Dadas essas limitações técnicas, os engenheiros da época combinaram técnicas *top-down* com abordagens *bottom-up* (de baixo para cima). Nesse tipo de projeto (*bottom-up*), problemas similares são reutilizados em outras partes de código, fazendo com que estes sejam resolvidos utilizando o mesmo trecho de código. Assim, os componentes do sistema se tornavam o mais **modular**

possível, considerando que um módulo é um componente que interage com o resto do sistema de uma forma simples e bem definida.

A ideia básica é a de que um módulo seja facilmente "plugado" ao sistema. Portanto, para que um módulo seja utilizado, deve-se conhecer a sua forma de interação com outros componentes (também conhecida como **interface do sistema)**, ou seja, quais os parâmetros necessários para utilizar este módulo e quais as funcionalidades que este módulo disponibiliza.

No início dos anos 80, foi criada a programação orientada a objetos (POO), na qual módulos guardavam estruturas de dados de forma protegida e com interface bem definida. O conceito chave da abordagem POO é o objeto, que é um tipo de módulo que possui uma estrutura de dados e sub-rotinas associados. Assim, um objeto é um tipo de entidade autossuficiente que possui um estado interno (seus dados constituintes) e que pode responder a mensagens (chamadas e sub-rotinas).

Para resolver um determinado problema utilizando a abordagem POO deve-se:

- Identificar os objetos envolvidos no problema;
- Identificar as mensagens que são trocadas entre esses objetos.

A solução do problema vai resultar em um conjunto de objetos que possuem o seu próprio dado associado e um conjunto de responsabilidades. Os objetos interagem entre si por meio de troca de mensagens. As principais vantagens desse tipo de modelo de programação são:

- Facilidade na escrita do programa;
- Facilidade de entendimento do código;
- Menor quantidade de erros (no caso geral);
- Alto índice de reuso de código.

Para conhecer um pouco mais sobre o histórico das linguages orientadas a objetos acesse:

- http://pt.wikibooks.org/wiki/Programa%C3%A7%C3%A3o_Orientada_a_Objetos/Introdu%C3%A7%C3%A3o
- http://social.stoa.usp.br/poo2013pos/aulas/aula-2-historia-da-programacao-orientada-a-objetos

Exemplo: Programação estruturada versus POO

Suponha que você queira criar um sistema que calcule o valor dos descontos mensais de um funcionário de uma locadora de filmes. Por simplicidade, será assumido que o trabalhador irá pagar o Imposto de Renda calculado como 27,5% do salário bruto mais a contribuição da Previdência Social, que varia de trabalhador pra trabalhador. Serão exibidas várias formas de se calcular os descontos no salário deste funcionário.

Exemplo 1.

```
<?php
$joaoSalario      = 1000;
$joaoPrevidencia  = 100;
$joaoNome     = "João Filho";
$joaoDescontos      =
round($joaoSalario*0.275 + $joaoPrevidencia,2);

$mariaSalario      = 2000;
$mariaPrevidencia  = 200;
$mariaNome     = "Maria Rute";
$mariaDescontos      =
round($mariaSalario*0.275 + $mariaPrevidencia,2);

$joseSalario      = 3000;
$josePrevidencia  = 400;
$joseNome     = "José Salgado";
$joseDescontos      =
round($joseSalario*0.275 + $josePrevidencia,2);

echo "O valor do desconto de $joaoNome é $joaoDescontos. <br>";
echo "O valor do desconto de $mariaNome é $mariaDescontos. <br>";
echo "O valor do desconto de $joseNome é $joseDescontos. <br>";
?>
```

O resultado da execução desse código é mostrado a seguir:

O valor do desconto de João Filho é 375.
O valor do desconto de Maria Rute é 750.
O valor do desconto de José Salgado é 1225.

Observe que todas as vezes em que precisarmos calcular os descontos de um novo funcionário, um novo cálculo deve ser adicionado. Se a forma de calcular o salário do trabalhador for mudada, essa alteração terá que ser feita em muitos lugares diferentes.

Exemplo 2.

```
<?php
function calcularDescontos($salario, $previdencia)
{
      return round($salario*0.275 + $previdencia,2);
}



$joaoNome     = "João Filho";
$joaoDescontos      =
      calcularDescontos(1000, 100);

$mariaNome     = "Maria Rute";
$mariaDescontos      =
      calcularDescontos(2000, 200);

$joseNome     = "José Salgado";
$joseDescontos      =
      calcularDescontos(3000, 300);

echo "O valor do desconto de $joaoNome é $joaoDescontos. <br>";
echo "O valor do desconto de $mariaNome é $mariaDescontos. <br>";
echo "O valor do desconto de $joseNome é $joseDescontos. <br>";
?>
```

Neste exemplo, criamos uma função para calcular os descontos do funcionário ao invés de repetir o cálculo várias vezes. A saída do programa permanece a mesma que a mostrada no exemplo anterior, contudo a informação

associada aos salários e previdência de cada funcionário é perdida quando a rotina “calcular descontos” é chamada.

Além disso, caso o número de funcionários dessa empresa fosse grande, seria bastante complicado manipular as variáveis relativas a cada funcionário. Assim, uma estrutura como *array* seria útil para guardar o conjunto de funcionários dessa empresa. O código a seguir apresenta essa mesma implantação, utilizando *arrays*.

Exemplo 3.

```
<?php


function calcularDescontos($salario, $previdencia)

{

      return round($salario*0.275 + $previdencia,2);

}


$funcionarios = array();

$funcionarios[0] = array("João Filho", 1000, 100);

$funcionarios[1] = array("Maria Rute", 2000, 200);

$funcionarios[2] = array("José Salgado", 3000, 300);


for($i = 0; $i < count(funcionarios); $i++)

{

      $nome = funcionarios[$i][0];

      $descontos = calcularDescontos(funcionarios[$i][1], funcionarios[$i][2]);


      echo "O valor do desconto de $nome é $descontos. <br>";


?>
```

Este exemplo mostra uma forma mais elaborada de calcular o mesmo salário do funcionário da locadora. Apesar do código do Exemplo 3 possuir pouca repetição de código, este ainda não pode ser considerado um módulo reutilizável do sistema. Isso se deve ao fato de os dados serem agrupados sem uma

formatação bem definida e a rotina de calcular descontos não estar associada a um funcionário específico.

O código que será apresentado a seguir mostra como esse problema é resolvido utilizando orientação a objetos.

Anotações

Exemplo 4.

```
<?php
class Funcionario{
	$nome;
	$salario;
	$previdencia;
	$descontos;

	function __construct($nome, $salario, $previdencia)
	{
		$this->nome = $nome;
		$this->salario = $salario;
		$this->previdencia = $previdencia;
		$this->descontos = $this->calcularDescontos();
	}

	function calcularDescontos()
	{
		return round($this->salario*0.275 + $this->previdencia,2);
	}
}

$funcionarios = array();
$funcionarios[0] = new Funcionario("João Filho", 1000, 100);
$funcionarios[1] = new Funcionario("Maria Rute", 2000, 200);
$funcionarios[2] = new Funcionario("José Salgado", 3000, 300);

for($i = 0; $i < count($funcionarios); $i++)
{
	$func = $funcionarios[$i];
	echo "O valor do desconto de $func->nome é $func->descontos  <br>";
```

Observe que para esse exemplo específico (Exemplo 4) a classe Funcionario foi criada. Essa classe descreve quais as características de cada objeto em termos de propriedades (atributos) e funcionalidades (métodos). Nesse caso, os atributos da classe Funcionario são nome, salário base e previdência e o método é calcularDescontos.

Dessa forma, para cada funcionário existe um número de previdência, nome e número associado. Maiores detalhes sobre a criação de classes, atributos e métodos serão discutidos na próxima seção. O método __construct será detalhado nas seções posteriores. Porém, por enquanto, basta saber que este método inicializa os atributos da classe Funcionario. Por exemplo, para o funcionário "João Filho", o construtor é chamado passando os parâmetros **Funcionario("João Filho", 1000, 100).**

Assuma agora que o cargo do funcionário seja dependente do salário que este ganha. Portanto, para funcionários com salário entre 1000 e 1999 teremos um colaborador júnior. Para salários maiores ou iguais a 2000 e menores que 2999, o funcionário é pleno e se o salário for maior ou igual a 3000 temos um funcionário sênior.

Como o tipo do funcionário será definido na sua criação, podemos definir o tipo de funcionário utilizando o método construtor para fazer tal operação. O trecho de código abaixo apresenta o código que realiza essas operações.

Exemplo 5.

```
<?php
class Funcionario{
	public $nome;
	public $salario;
	public $previdencia;
	public $descontos;
	public $tipoFuncionario;

	function __construct($nome, $salario, $previdencia) {
		$this->nome = $nome;
		$this->salario = $salario;
		$this->previdencia = $previdencia;
		$this->descontos = $this->calcularDescontos();

		if($this->salario >= 1000 && $this->salario < 2000) {
			$this->tipoFuncionario = "Júnior";
		}
		else if($this->salario >= 2000 && $this->salario < 3000) {
			$this->tipoFuncionario = "Pleno";
		}
		else if($this->salario >= 3000) {
			$this->tipoFuncionario = "Sênior";
		}
	}
	function calcularDescontos(){
		return round($this->salario*0.275 + $this->previdencia,2);
	}
}
```

Exemplo 5. Continuação

```
$funcionarios = array();
$funcionarios[0] = new Funcionario("João Filho", 1000, 100);
$funcionarios[1] = new Funcionario("Maria Rute", 2000, 200);
$funcionarios[2] = new Funcionario("José Salgado", 3000, 300);

for($i = 0; $i < count($funcionarios); $i++)
{
      $func = $funcionarios[$i];
      echo "O valor do desconto de $func->nome é $func->descontos
      e seu tipo é $func->tipoFuncionario<br>";
}
?>
```

Exercício:

Utilize como base os exemplos anteriores e construa dois programas PHP (um de forma estruturada e outro orientado a objetos) que calculem o IMC de uma determinada pessoa. O cálculo do IMC é avaliado como peso dividido pela altura ao quadrado.

Nos links a seguir o aluno poderá visulalizar outras explicações que mostram a diferença entre o paradigma estruturado e orientado a objetos.

- http://fabrica.ms.senac.br/2013/04/programacao-estruturada-versus-programacao-orientada-a-objetos/
- http://www.devmedia.com.br/programacao-orientada-a-objetos-x-programacao-estruturada/11747

Veja esses videos demonstrativo sobre diferença entre programação estrutura e orientada a objetos.

https://www.youtube.com/watch?v=vhvmZfxZhPw

https://www.youtube.com/watch?v=PmefpISZ7Ew

Capítulo 02: Conceitos de classe, herança, objeto, atributo e método.

Comentado [IPC1]: Esta seria a competência 2?

Esta seção aborda os conceitos fundamentais do paradigma de programação orientado a objetos. Embora os conceitos aqui estejam sendo apresentados com uso da linguagem PHP, os assuntos abordados são genéricos e podem ser extrapolados para outras linguagens orientadas a objetos.

Conceitos Fundamentais.

Objetos são porções de software, descritos por variáveis e métodos. Objetos em software modelam os objetos do mundo real através do seu comportamento e seus componentes. Em linguagem estruturada, até então, vimos os conceitos de atributos e métodos. Por outro lado, quando falamos em componentes de um objeto temos métodos (relativos às funções) e atributos (relativos às variáveis).

Na prática, um objeto é uma representação em memória do computador de uma entidade do mundo real.

O que é uma classe?

Classe é o modelo ou protótipo que define as variáveis e métodos comuns a um conjunto de objetos de certo tipo. Em geral, uma classe define como é cada objeto, sua estrutura e seu comportamento, enquanto os objetos são as entidades "vivas" dentro do programa. Uma classe pode ter vários representantes (objetos) dentro de um mesmo programa, cada objeto com seus atributos em particular.

As classes normalmente têm a seguinte forma (Exemplo 6):

Exemplo 6.

```
<?php
#class [NomeDaClasse] {
#        [lista de atributos]
#       [lista de métodos]
#}

//Exemplo:
class Conta{
      public $numero;
      public $saldo;

      function __construct($numero, $saldo)
      {
            $this->numero = $numero;
            $this->saldo = $saldo;
      }
}
?>
```

No corpo da classe há basicamente três componentes básicos: Atributos, Construtores e Métodos.

Atributos:

- Também conhecidos como variáveis de instância descrevem as características dos objetos.
- O tempo de vida dos atributos é igual ao tempo de vida do objeto.
- Cada objeto tem seus próprios valores de atributos em tempo de execução.

No Exemplo 6, os atributos da classe **Conta** são **número** e **saldo** e definem os dados associados a este objeto. Portanto, em uma aplicação de banco, para o nosso exemplo, toda conta possuirá número e saldo

Comentado [AE2]: destacar os termos do exemplo para não confundir o aluno.

associados. Os valores dos atributos são específicos para cada objeto e podem ou não coincidir com algum outro objeto criado, dependendo de cada projeto.

Métodos:

Os métodos são os componentes da classe que realizam as computações (funções).

- Métodos recebem parâmetros, ou seja, valores para que podem ser utilizados durante a computação.
- Pode ou não retornar um resultado, assim como funções não associadas a métodos.

Exemplo 7.

```
<?php
#[modificadores] function [nome do metodo]([lista de parametros])


//Exemplo:
class ContaCliente{
	public $numero;
	public $saldo;


	function creditar($valor)
	{
		$this->saldo = $this->saldo + $valor;
	}
}
?>
```

Os métodos possuem uma assinatura, que correspondem a uma descrição do que o método deve faze e ainda os valores necessários para a computação e o retorno da computação. A forma básica da assinatura dos métodos é mostrada no Exemplo 7. No caso da função creditar um determinado valor, é recebido como parâmetro e adicionado ao saldo atual da conta.

Assim como funções que não são associadas a objetos, a lista de parâmetros de um método pode ser vazia. Para retornar um resultado de um método, basta adicionar a sentença result no final do método (observe o Exemplo 8). Após a

criação do objeto calculadora (`$calc = new Calculadora();`) a função somar é chamada e o resultado é impresso na tela.

Anotações

Exemplo 8.

```
<?php
class Calculadora{

      function somar($operador1, $operador2){
            return $operador1 + $operador2;
      }
}

$calc = new Calculadora();
$soma = $calc->somar(2, 2);
echo "SOMA : $soma";
?>
```

Saída:

SOMA : 4

Para conhecer um pouco mais sobre classes e objetos em PHP acesse. http://php.net/manual/pt_BR/language.oop5.basic.php

Veja também esse vídeos:

https://www.youtube.com/watch?v=9bQNsfYqVc4

https://www.youtube.com/watch?v=UM9YBFqIwQ4

Trabalhando com Objetos.

Em PHP, uma variável inicializada como objeto guarda uma referência para o mesmo e pode ser utilizada para invocar seus métodos e alterar/visualizar seus atributos. A Figura 1 apresenta uma representação gráfica de como uma variável que aponta para um objeto (referência) é esquematizada na memória do computador.

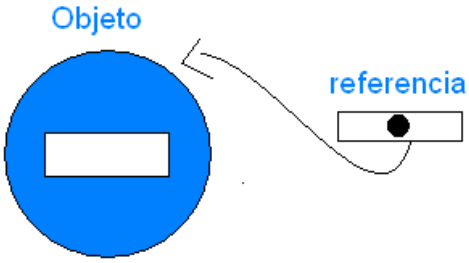

Figura 1 Acessando Objetos
Fonte: Próprio autor

No Exemplo 8, a variável $calc permite que o usuário acesse os métodos/atributos do objeto Calculadora.

Inicializando Objetos:

Para que uma variável que referencia um objeto seja utilizada, é necessário que este objeto exista, ou seja, é preciso que ele seja criado e esteja armazenado na memória do computador. PHP dispõe do comando new, que cria um novo objeto, inicializando os atributos da classe segundo o construtor da classe.

Construtor

Tipo especial de método que inicializa as variáveis do objeto, quando instanciado (inicializado). Como mencionado anteriormente, a nossa classe Conta possui dois atributos principais (número e saldo).

Exemplo 9.

```
<?php
class Filme{
	public $nome;
	public $saldo;

	function __construct($nome, $saldo)
	{
		$this->nome = $nome;
		$this->saldo = $saldo;
	}

	function incrementarSaldo($valor)
	{
		$this->saldo = $this->saldo + $valor;
	}
}

	$filme = new Filme("Exterminador", 100);
	$filme->incrementarSaldo(150);

	echo "Nome do filme: $filme->nome <br>";
	echo "Saldo: $filme->saldo";
?>
```

Saída:
Nome do filme: “Exterminador”
Saldo: 250

Construtores não possuem um retorno, e utilizam a instrução __construct. Para que um construtor seja chamado é necessário o uso da palavra chave **new** seguida pelo nome da classe e a lista de parâmetros.

No Exemplo 9, o construtor da classe Filme é chamado em $filme = new Filme(1, 100). Isso invoca o método __construct($nome, $saldo), passando nome "Exterminador" e saldo 100. Os valores do nome e saldo da variável $filme são inicializados no código do construtor ($this->nome = $nome; e $this->saldo = $saldo;).

Veja esse site que apresenta uma descrição mais detalhada sobre construtores em PHP

- http://php.net/manual/pt_BR/language.oop5.decon.php

O operador $this:

O operador *$this* do PHP indica o que a variável/método que está sendo utilizada em determinada parte da classe é pertencente ao objeto atual e somente a ele. Por exemplo, o método __construct do Exemplo 9 possui dois parâmetros (número e saldo - não confunda com os atributos da classe Conta). Para que seja possível a diferenciação entre os parâmetros do método construtor e os atributos da classe Conta o operador $this se faz necessário. Dessa forma, quando utilizamos o $this para acessar alguma entidade da classe (método ou atributo), de fato estamos referenciando as entidades pertencentes a este (*this* em inglês) objeto que está sendo usado no momento.

Comentado [JV3]: não seria atributos?

Assim no trecho de código a seguir, as variáveis marcadas em vermelho referem-se aos parâmetros do construtor, enquanto que as variáveis em azul pertencem à classe.

```
function __construct($nome, $saldo)
{
        $this->nome = $nome;
        $this->saldo = $saldo;
```


```
}
```

O operador é útil, pois:

- Facilita a leitura do código;
- Pode ser decisivo em casos de ambiguidade.

Exemplo 10.

```
<?php

class Conta{

        public $numero;

        public $saldo;


        function __construct($numero, $saldo){

                $this->numero = $numero;

                $this->saldo = $saldo;

        }


        function creditar($valor){

                $this->saldo = $this->saldo + $valor;

        }


        function debitar($valor){

                $this->saldo = $this->saldo - $valor;

        }


        function transferir($outraConta, $valor){

                if($this->saldo > $valor)

                {

                        $this->debitar($valor);

                        $outraConta->creditar($valor);

                }

                //Se não entrou no if é pq o saldo

                //é insuficiente.

        }

}

        $conta1 = new Conta(1, 100);

        $conta2 = new Conta(2, 100);
```

Anotações

```
        echo "Saldo da conta $conta1->numero: $conta1->saldo <br>"
        echo "Saldo da conta $conta2->numero: $conta2->saldo <br>";

        $conta1->transferir($conta2, 50);
        echo "Transferência efetuada de 50 da conta1 para conta2 <br>";

        echo "Saldo da conta $conta1->numero: $conta1->saldo <br>"
        echo "Saldo da conta $conta2->numero: $conta2->saldo <br>";

?>
```

Saída:

Saldo da conta 1: 100 Saldo da conta 2: 100

Transferência efetuada de 50 da conta1 para conta2

Saldo da conta 1: 50

Saldo da conta 2: 150

No Exemplo 10, criamos uma classe conta que possui número e saldo e esta efetua operações de débito, de crédito e também de transferências entre contas. Observe neste caso a importância do operador $this no método transferir. No caso geral, quando efetuamos uma transferência bancária, fazemos um débito na nossa conta corrente e efetuamos um crédito na conta de destino. Nesse caso particular, fizemos um débito de 50 na conta atual ($this->debitar($valor)) e efetuamos um crédito de 50 na outra conta ($outraConta->creditar($valor)). Além disso, para a nossa aplicação, uma transferência só pode ser efetuada se há saldo suficiente para operação.

Atributos e métodos estáticos

Quando um atributo/método pertence não somente a um objeto, como no exemplo do número de uma Conta (ver Exemplo 10), mas a todos os objetos pertencentes à classe, então utilizamos um atributo/método do tipo estático. A declaração de um atributo ou método estático faz com que este seja acessível sem que haja a necessidade de criação de um objeto. É importante mencionar que um método estático pode ser acessado por um objeto, mas um atributo não. Dado que atributos/métodos estáticos podem ser acessados sem a necessidade de um objeto existir, a palavra-chave $this não pode ser utilizada dentro de um método estático. Além disso, atributos estáticos não podem ser utilizados através do operador (->).

O Exemplo 11 apresenta uma forma de utilização de atributo/método estático. Observe que nenhum objeto foi criado e, mesmo assim, foi possível acessar os atributos da classe. Observe que nesse caso o valor $pi não está relacionado com nenhum objeto específico e, por isso, foi associado à classe, diferentemente de um saldo da conta que pertence a um objeto específico.

Exemplo 11.

```
<?php
class AreaCalc{
	public static $pi = 3.141517;

	static function areaCircunferencia($raio)
	{
		return $raio*$raio*(AreaCalc::$pi);
	}
}

$raioAtual = 10;
$area = AreaCalc::areaCircunferencia($raioAtual);
$valorPi = AreaCalc::$pi;
echo "Area circunferencia de raio $raioAtual eh $area<br>";
echo "Valor de pi $valorPi<br>";
?>
```

Saída:

Area circunferência de raio 10 eh 314.1517

Valor de pi 3.141517

Links úteis

- http://www.devmedia.com.br/introducao-a-orientacao-a-objetos-em-php/26762

- http://www.kadunew.com/blog/php/introducao-php-orientado-a-objetos-objetos-e-classes

Herança

A redução de custos com software não está em contratar programadores baratos e mal qualificados. Isso pode prejudicar bastante o tempo de entrega do projeto e sua usabilidade, além de dificultar a manutenção do código.

A redução dos custos de um determinado software se dá através do reuso e adaptabilidade do código. De fato, um dos custos mais efetivos do processo de desenvolvimento de software está relacionado ao capital humano. Para isso, entre outros aspectos, é necessário dispor de uma linguagem que dê suporte a fatores de qualidade como facilidade de reuso e manutenção, incluindo adaptabilidade.

Uma das formas de reuso de software é a herança. Relações como ObjetoA é um tipo de ObjetoB são claros indicativos de que a herança pode ser utilizada. Ex: Poupança (Objeto A) é um tipo de Conta (Objeto B). Ou seja, o ObjetoA tem tudo que ObjetoB tem, incluindo algumas funcionalidades extras.

Quando uma classe herda da outra temos uma relação de dependência entre as duas, na qual existe uma superclasse (classe que é herdada) e uma subclasse (classe que herda). Com esse recurso é possível aproveitar uma estrutura existente de uma classe e replicar em outra. Dessa forma, conseguimos reutilizar códigos, sem a necessidade de reescrever ou duplicar trechos do programa. Os relacionamentos entre as subclasses e a superclasses em PHP são:

1. A subclasse herda o comportamento (atributos e métodos), diretamente de apenas uma superclasse;
2. Funciona como uma hierarquia.

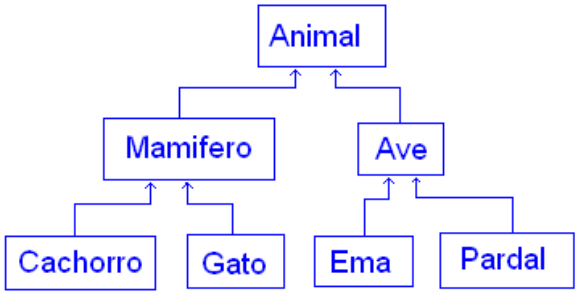

Figura 2: Hierarquia de classes

Fonte: Próprio autor

- No exemplo acima (Figura 2) Cachorro herda diretamente de Mamífero, mas também herda indiretamente de animal, haja vista que a herança é transitiva.
- Usa a palavra reservada ***extends*** exemplo: class Poupanca extends Conta{}

O Exemplo 12 estende o Exemplo 10 e mostra como seria criada uma classe Poupança que herda da classe Conta. Como se sabe, uma poupança bancária é um tipo especial de conta que possui uma taxa de juros que é aplicada ao saldo em períodos determinados.

Neste exemplo, veremos como é simples herdar o comportamento da classe Conta na classe Poupanca. Observe que na classe poupança, apenas as funcionalidades adicionais são codificadas e todo o restante das funcionalidades da classe Conta são aplicadas implicitamente ao código da classe Poupanca. Os métodos creditar e debitar são adicionados implicitamente na classe poupança. Portanto, pode ser chamada diretamente sem maiores problemas.

Exemplo 12.


```
<?php
class Conta{
	public $numero;
	public $saldo;

	function __construct($numero, $saldo){
		$this->numero = $numero;
		$this->saldo = $saldo;
	}

	function creditar($valor) {
		$this->saldo = $this->saldo + $valor;
	}

	function debitar($valor) {
		$this->saldo = $this->saldo - $valor;
	}

	function transferir($outraConta, $valor){
		if($this->saldo > $valor)
		{
			$this->debitar($valor);
			$outraConta->creditar($valor);
```

Exemplo 12. Cont.

```
class Poupanca extends Conta{

	public $juros = 0.05;


	function atualizarJuros()

	{

		$this->saldo = $this->saldo*(1 + $this->juros);

	}

}


	$conta = new Conta(1, 150);

	$poupanca = new Poupanca(2, 150);


	$conta->creditar(50);

	$conta->debitar(100);


	//os métodos debitar e creditar são utilizados sem problemas

	//mesmo que não sejam escritos explicitamente.

	$poupanca->creditar(50);

	$poupanca->debitar(100);

	$poupanca->atualizarJuros();


	echo "Saldo da conta $conta->numero: $conta->saldo <br>";

	echo "Saldo da conta $poupanca->numero: $poupanca->saldo <br>";


?>
```

Saída do programa:

Saldo da conta 1: 100
Saldo da conta 2: 105

Podemos observar que no caso da conta poupança, o valor do saldo foi atualizado com juros (que nesse caso é fixo em 5%). A **superclasse** Conta tem como **subclasse** Poupanca.

Se ao invés de adicionarmos um valor fixo para a taxa de juros, quiséssemos inicializar o valor da taxa com um valor dado pelo usuário? Utilizaríamos um construtor como no Exemplo 13. O construtor inicializa o número, saldo e juros com um valor definido pelo usuário.

Exemplo 13.

```
<?php
class Conta{
	public $numero;
	public $saldo;

	function __construct($numero, $saldo){
		$this->numero = $numero;
		$this->saldo = $saldo;
	}

	function creditar($valor){
		$this->saldo = $this->saldo + $valor;
	}

	function debitar($valor){
		$this->saldo = $this->saldo - $valor;
	}

	function transferir($outraConta, $valor)  {
		if($this->saldo > $valor)
		{
			$this->debitar($valor);
			$outraConta->creditar($valor);
		}
	}
}
```

Exemplo 13. Cont.

```
class Poupanca extends Conta{
        public $juros;

        function __construct($numero, $saldo, $juros)
        {
                $this->numero = $numero;
                $this->saldo = $saldo;
                $this->juros = $juros;
        }

        function atualizarJuros()
        {
                $this->saldo = $this->saldo*(1 + $this->juros);
        }
}

        $conta = new Conta(1, 150);
        $poupanca = new Poupanca(2, 150, 0.10);

        $conta->creditar(50);
        $conta->debitar(100);

        $poupanca->creditar(50);
        $poupanca->debitar(100);
        $poupanca->atualizarJuros();

        echo "Saldo da conta $conta->numero: $conta->saldo <br>";
        echo "Saldo da conta $poupanca->numero: $poupanca->saldo <br>";
?>
```

Saída do programa:

Saldo da conta 1: 100
Saldo da conta 2: 110

Apesar de estar sintaticamente correta, a forma de inicializar os atributos da classe Conta no construtor de Poupanca não está sendo feita da maneira mais adequada. Estamos reescrevendo o código do construtor de Conta no construtor da classe Poupanca. Poderíamos, ao invés disso, reutilizar o construtor de poupança fazendo uma chamada do construtor da classe pai utilizando o operador (**parent::**). Com o uso deste operador podemos acessar qualquer método da classe pai (até mesmo o construtor). No Exemplo 14, mostramos como o mesmo código seria executado de uma forma mais elegante pela utilização do operador (**parent::**).

Exemplo 14.

```
<?php
class Conta{
	public $numero;
	public $saldo;

	function __construct($numero, $saldo)
	{
		$this->numero = $numero;
		$this->saldo = $saldo;
	}

	function creditar($valor)
	{
		$this->saldo = $this->saldo + $valor;
	}

	function debitar($valor)
	{
		$this->saldo = $this->saldo - $valor;
	}
```

Exemplo 14. Cont.

```
        function transferir($outraConta, $valor){
                if($this->saldo > $valor)
                {
                        $this->debitar($valor);
                        $outraConta->creditar($valor);
                }
        }
}
class Poupanca extends Conta{
        public $juros;
        function __construct($numero, $saldo, $juros)
        {
                parent::__construct($numero, $saldo);
                $this->juros = $juros;
        }
        function atualizarJuros()
        {
                $this->saldo = $this->saldo*(1 + $this->juros);
        }
}
        $conta = new Conta(1, 150);
        $poupanca = new Poupanca(2, 150, 0.10);
        $conta->creditar(50);
        $conta->debitar(100);
        $poupanca->creditar(50);
        $poupanca->debitar(100);
        $poupanca->atualizarJuros();
```

A saída do Exemplo 14 é idêntica à do Exemplo 13. Porém, a segunda forma de se escrever o código é mais eficiente dado que, caso haja uma alteração no código do construtor da classe Conta, a alteração seria feita apenas na classe Conta.

É possível também alterar o comportamento de uma determinada classe reutilizando o código da superclasse. Por exemplo, se no caso da poupança todo crédito fosse seguido de uma atualização de juros. Nesse caso, teríamos uma classe Poupanca como mostrada no Exemplo 15 (insira a classe Poupanca do Exemplo 14 e veja você mesmo a diferença no resultado).

Anotações

Exemplo 15.

```
class Poupanca extends Conta{

	public $juros;


	function __construct($numero, $saldo, $juros)

	{

		parent::__construct($numero, $saldo);

		$this->juros = $juros;

	}


	function creditar($valor)

	{

		parent::creditar($valor);

		$this->atualizarJuros();

	}


	function atualizarJuros()

	{

		$this->saldo = $this->saldo*(1 + $this->juros);

	}

}
```

Saída do programa:

Saldo da conta 1: 100
Saldo da conta 2: 132

Nesse caso, fizemos uma sobrecarga do método creditar. Onde o método da superclasse (Conta) foi alterado para um comportamento diferente na subclasse (Poupança). É importante salientar que a chamada do método da superclasse com o operador parent:: não é obrigatório e o método sobrescrito pode ser totalmente diferente do da classe pai.

Modificadores de Acesso:

A fim de garantir que algumas variáveis, atributos e até mesmo classes só sejam visíveis nos locais necessários, PHP dispõe de modificadores de acesso que indicam a visibilidade de cada atributo, método ou até mesmo da própria classe. Essa funcionalidade é importante para garantir que somente o necessário seja visível fora da classe. Isso garante que a classe PHP seja visualizada como um módulo isolado e apenas o que interessa seja visível a outras partes do código.

Os modificadores de acesso da linguagem PHP são:

- Private, visível apenas dentro da classe.
- Protected, visível dentro da classe e em todas as classes que herdam desta classe.
- Public, visível em todos os lugares do sistema. Quando não se especifica nenhuma opção, então o atributo/método é assumido como public.

Os exemplos a seguir auxiliarão o entendimento das variáveis public, protected e private.

Exemplo 16.

Comentado [AE4]: No último comentário, poderia dizer quais as linhas que não seriam visíveis.

```
<?php
class Visibilidade{
	public $varPublic;
	protected $varProtected;
	private $varPrivate;

	public function __construct($va1, $var2, $var3)
	{
		$this->varPublic = $va1;
		$this->varProtected = $var2;
		$this->varPrivate = $var3;
	}
}

	$visibilityTest = new Visibilidade(1,2,3);
	echo "Atributo Public : $visibilityTest->varPublic";
	//echo "Atributo Protected : $visibilityTest->varProtected";
	//echo "Atributo Public : $visibilityTest->varPrivate";

	//As duas linhas acima não funcionariam pois as variáveis não
	//são visíveis
?>
```

Saída:

Atributo Public : 1

No Exemplo 16 observa-se que apenas a variável $varPublic é visível, portanto apenas esta pode ser acessada de fora da classe. O mesmo vale para o Exemplo 17 onde apenas o método público pode ser invocado de fora da classe.

Exemplo 17.

```
<?php
class Visibilidade{

	public function publicFunction()
	{
		echo "public method<br>";
	}

	protected function protectedFunction()
	{
		echo "protected method<br>";
	}

	private function privateFunction()
	{
		echo "private method<br>";
	}
}

	$visibilityTest = new Visibilidade();
	$visibilityTest->publicFunction();
	//$visibilityTest-> protectedFunction();
	//$visibilityTest-> privateFunction();

	//As duas lihas acima não funcionam pois as variáveis não
	//são visíveis
?>
```

Comentado [AE5]:
Problema no final do exemplo:

Está comentado, porém ao retirar o comentário, estaria chamando a classe que não existe porque escreveu o nome do método errado:
`protecredFunction()`
`e seria`
`protectedFunction():`

A letra "r" pela "t"

Anotações

Comentado [I6]: No exemplo acima não seria?
```
//$visibilityTest->protecredFunction();
//$visibilityTest->privateFunction();
```

No Exemplo 18, modificamos o Exemplo 15 e vemos um exemplo de um atributo protected ($juros) que é utilizado pela subclasse de forma direta. Se este atributo fosse private, haveria um erro e a saída não seria gerada.

Exemplo 18.

```
class Poupanca extends Conta{

	protected $juros;


	function __construct($numero, $saldo, $juros)

	{

		parent::__construct($numero, $saldo);

		$this->juros = $juros;

	}


	function creditar($valor)

	{

		parent::creditar($valor);

		$this->atualizarJuros();

	}


	function atualizarJuros()

	{

		$this->saldo = $this->saldo*(1 + $this->juros);

	}

}
```

Métodos get e set:

Os links abaixo apresentam informações adicionais a respeito dos conceitos de orientação a objetos em PHP e herança.

- http://php.net/manual/pt_BR/language.oop5.basic.php
- http://josemalcher.net/heranca-php/
- http://php.net/manual/pt_BR/language.oop5.inheritance.php

Para um melhor controle das variáveis dos objetos, é recomendado que elas sejam declaradas como private. Com esse procedimento, é garantido que nenhuma outra parte do programa faça uso indevido das variáveis. O que você acha se uma variável como saldo bancário, por exemplo, fosse declarada com public? Qualquer outro código do programa poderia atribuir qualquer valor para a variável saldo, isto seria um desastre não é?

Para controlar o acesso das variáveis como boa prática de programação são declarados nas classes os métodos "get" e "set", que são métodos que retornam o valor da variável e atribuem um novo valor a ela, respectivamente.

Dessa forma, o programador define quem pode acessar as variáveis e quais são as condições para que uma variável seja alterada ou lida. O Exemplo 19 mostra como o saldo de uma conta de banco poderia ser lido mesmo que a variável saldo seja private (usaremos métodos get e set). Observe que no Exemplo 14, por outro lado, o saldo da conta pode ser manipulado sem problemas em qualquer parte do programa.

Exemplo 19.

```
<?php
class Conta{
	private $numero;
	private $saldo;

	function __construct($numero, $saldo)
	{
		$this->numero = $numero;
		$this->saldo = $saldo;
	}

	public function getSaldo()
	{
		//Aqui poderia conter um codigo de verificacao
		//de autencidade do usuario
		return $this->saldo;
	}
```

Exemplo 19 Cont.

```
        public function setSaldo($novoSaldo)
        {
                //Aqui poderia conter um codigo de verificacao
                //do valor depositado
                $this->saldo = $novoSaldo;
        }
}

        $conta = new Conta(1, 100);

        //$conta->saldo = 300; A variável não e visível - ERRO
        $conta->setSaldo(300);

        //echo "Saldo: $conta->saldo"; A variável não e visível - ERRO
        $saldoAtual = $conta->getSaldo();
        echo "Saldo: $saldoAtual";

?>
```

Classes abstratas:

Um dos maiores benefícios do conceito de orientação a objetos é o uso de tipos abstratos. Uma abstração de uma classe é um nome elegante para um agregado de dados e informações bem encapsulados. Muitas vezes você não está interessado em como as operações são realizadas, mas sim no comportamento

desse agrupamento de dados. Em PHP as abstrações de dados são implantadas com classes abstratas e interfaces.

Vejamos um exemplo de abstração de dados: suponha que queiramos representar uma classe comida. Sabe-se que a comida não pode ser imaginada sem que um dos seus representantes seja pensado. Ou seja, só existe uma comida se houver um tipo de comida específica (ex. Arroz, Feijão). O que seria um objeto Comida? Comida representa um conceito abstrato daquilo que se pode comer.

Em PHP, quando há algum comportamento (método) em alguma classe que não podemos representar a menos que uma classe que a herde implante, temos uma grande candidata à classe abstrata. Fica a cargo das suas subclasses o trabalho de redefinir o método dito abstrato.

Por exemplo, suponha que precisássemos de uma classe do tipo figura geométrica para calcularmos sua área e perímetro. Nesse caso, para calcularmos a área temos que especificar que tipo de figura geométrica temos. Portanto, a classe FiguraGeometrica seria uma classe abstrata. Como subclasses desta temos, para esse exemplo, Quadrado e Circunferencia. A Figura 3 detalha a estrutura de classes.

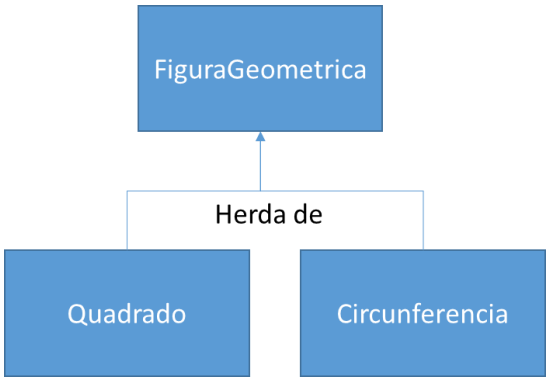

Figura 3. Esquema de classes para Figura Geométrica.

Anotações

Comentado [AE7]: agrupamento?

Comentado [AE8]: De acordo com a figura abaixo, seria Quadrado e Circunferência.

Outra questão é que poderia ser utilizado Circulo ao invés de Circunferência, pois Círculo seria um tipo de figura geométrica já circunferência seria uma medida.

Comentado [WU9R8]: A circunferência está sendo utilizada de forma correta. Circulos são os pontos internos de uma circunferência. A forma geométrica de fato é a circunferência. Vide http://educacao.uol.com.br/disciplinas/matematica/circunferencias-definicao-e-propriedades.htm

Fonte: Próprio autor

Observe que no caso do Quadrado e Circunferencia pode-se calcular sua área e perímetro. O Exemplo 20 apresenta o código referente às três classes mostradas a seguir.

Anotações

Comentado [AE10]: Veja comentário anterior.

Exemplo 20.

```
<?php

abstract class FiguraGeometrica{

	private $tipo;


	public function __construct($tipo)

	{

		$this->tipo = $tipo;

	}

	public function printCaracteristicas()

	{

		$areaTemp = $this->area();

		$perimetroTemp = $this->perimetro();

		echo "Tipo: $this->tipo, Area $areaTemp, Perimetro
$perimetroTemp<br>";

	}

	public abstract function area();

	public abstract function perimetro();

}

class Circunferencia extends FiguraGeometrica{
```

Exemplo 20 Continuação

```
class Retangulo extends FiguraGeometrica{

        private $lado1, $lado2;


        public function __construct($tipo, $lado1, $lado2)

        {

                parent::__construct($tipo);

                $this->lado1 = $lado1;

                $this->lado2 = $lado2;

        }


        public function area()

        {

                return $this->lado1*$this->lado2;

        }


        public function perimetro()

        {

                return 2*$this->lado1 + 2*$this->lado2;

        }

}


        $circ = new Circunferencia("Circunferencia", 10);

        $circ->printCaracteristicas();



        $retangulo = new Retangulo("Retangulo", 10, 20);

        $retangulo->printCaracteristicas();


?>
```


Anotações

Saída:

Tipo: Circunferencia, Area 314, Perimetro 62.8
Tipo: Retangulo, Area 200, Perimetro 60

Para definir a classe como abstrata utilizamos a palavra chave abstract antes da definição da classe. O mesmo vale para os métodos abstratos. Observe também que o método possui apenas assinatura e não possui corpo. Como ambas as classes Circunferencia e Retangulo herdam de FiguraGeometrica, o método printCaracteristicas pode ser chamado sem problemas através de suas classes filhas.

Interfaces:

E se ao invés de abstrair parte da implantação, como fazemos com classes abstratas, quiséssemos abstrair toda a implantação de todos os métodos da classe? Ou seja, se não estivéssemos interessados nas implantações das classes, mas sim nos serviços que a classe pode oferecer, usaríamos interfaces que são espécies de contratos entre duas classes. Uma oferece serviços à outra. Como, por exemplo, repositórios onde só estamos interessados em inserir, remover e atualizar. Não estamos interessados se o repositório é feito com arrays, banco de dados, vetores, ou qualquer que seja a estrutura de dados na qual os objetos são armazenados. As interfaces podem ser pensadas relativamente como uma classe abstrata só com métodos abstratos.
Exemplo de Interface:

```
interface IRepositorioContas{
    public function cadastrar($conta);
```


```
        public function remover($conta);
        public function atualizar($conta);
}
```

Todos os repositórios que implantam os serviços definidos em IRepositórioContas devem conter os métodos definidos na interface acima.

Por exemplo, uma classe Agencia poderia implantar esta interface em particular. Precisamos informar que a classe implanta a interface em questão, usando a palavra chave **implements** como descrito abaixo.

```
public Agencia implements IRepositorioContas {
        public function cadastrar($conta)
        {
                //Corpo da classe
        }
        public function remover($conta)
        {
                //Corpo da classe
        }

        public function atualizar($conta)
        {
                //Corpo da classe
        }
```


```
}
```

Assim como objetos que herdam de alguma classe podem ser referenciados pela superclasse, um objeto que referencia uma interface pode ser referenciado por uma variável com o tipo da Interface em questão. Por exemplo:

```
$contas1 = new Agencia();
$contas2 = new ObjetoQueImplementaIRepositorioContas();
$contas2->inserir(new Conta());
$contas1->inserir(new Conta());
//posso chamar sem medo os métodos de IRepositorioContas.
```

Exercício: Implemente em PHP a classe Funcionario, que tem como atributos o número de dias trabalhados, valor do ganho diário e seu respectivo nome. Adicione à classe Funcionario um método que calcula o seu salário com base no ganho diário e nos dias trabalhados.

Exercício: Pense na implementação da classe gerente, que é um tipo especial de funcionário, pois tem os mesmos atributos de um funcionário, mas seu salário é calculado de forma diferente, seu

salário é fixado em 1000.00 reais. Tente implementar esta classe reusando o código da classe Funcionario.

Exercício: Digamos que devemos calcular a média entre dois números e não estamos interessados em como esta média é calculada. Defina três classes Mediageometrica, MediaAritmetica e MediaHarmonica cada um com um método CalcularMedia. Escreva uma tela de Teste com um método para testar os três métodos usando a mesma referencia a um Tipo Media. Faça a questão usando interface e depois use Classe abstrata.

Anotações

Capítulo 03: Utilizando técnicas de orientação a objetos em uma aplicação.

O projeto utilizado por esta disciplina corresponde a uma locadora de vídeos. Vamos neste caderno criar uma aplicação para aluguel de vídeos que utiliza orientação a objetos. Consideramos que você conheça os pré-requisitos necessários para elaboração do projeto, ou seja, você já sabe utilizar o XAMPP, Aptana Studio, MySQL. Vamos fazer um cadastro completo de filmes para a locadora virtual utilizando classes e objetos ao invés de puramente PHP estruturado. A Figura 4 apresenta a tela principal do programa.

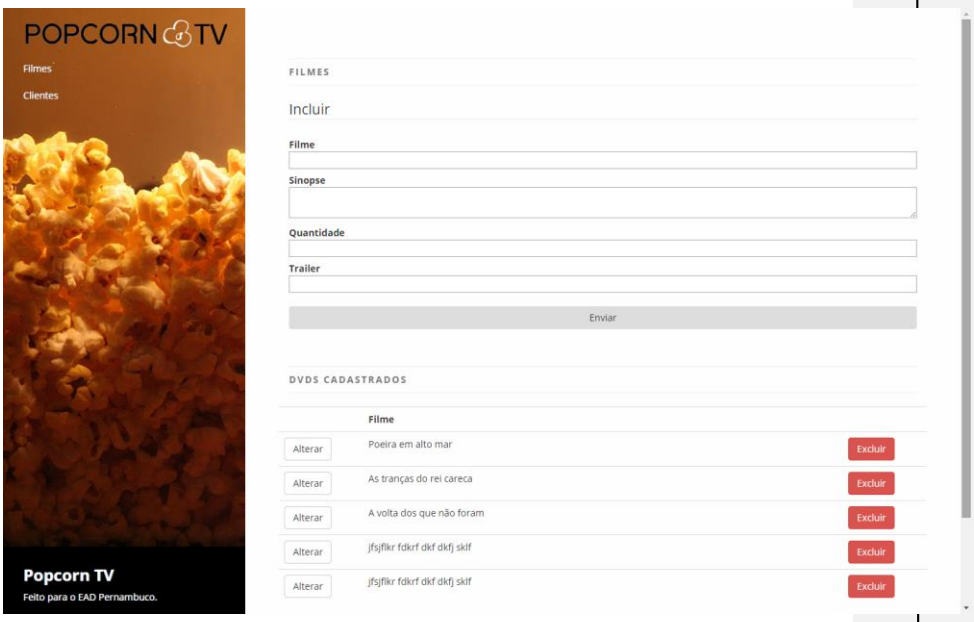

Figura 4 – Popcorn TV – Projeto da disciplina.
Fonte: Próprio autor.

Utilize o projeto base que está localizado no sistema de gerenciamento de aulas e importe para o Aptana.

Conectando ao MySQL

Assumimos que você já possua uma base de dados chamada popcorntv previamente configurada utilizando o XAMPP. Para o caso orientado a objetos, ao invés de utilizar comandos estruturados para conectar ao banco de dados, vamos criar uma classe que faça conexão com o banco de dados. Todos os comandos de envio de consultas SQL serão por esta classe. A classe é apresentada na Figura 5. Crie um novo arquivo chamado conexao.php e digite o código abaixo. Nos comentários do código estão as explicações de cada função e variável. Observe que foram criados métodos para conectar ao banco, executar consultas e todas as variáveis relativas ao banco ficam encapsuladas no objeto conexão quando este for criado.

Depois de criada a classe, basta que o programador crie um novo objeto conexão, conecte ao banco e execute as consultas desejadas. Por exemplo:

```
//Cria o objeto conexão que será responsável pelas chamadas ao banco de dados
$conexao = new Conexao("localhost", "root", "", "popcorntv");

//Conecta ao banco de dados
if($conexao->conectar() == false)
{
        //Caso ocorra algum erro, uma mensagem é mostrada ao usuário
        echo "Erro" . mysqli_error();
}

$sql = "DELETE FROM filme WHERE codigo = '2'";
```


```
//Executa a consulta
$conexao->executarQuery($sql);
```

Você deve ter o cuidado de saber o endereço de seu banco, a senha, o login e o nome do banco. Caso algum deles esteja errado, irá aparecer uma mensagem de erro informando. Verifique também se o XAMPP está ligado com o Apache e o MySQL.

conexao.php

```php
<?php

/**
 * Cria uma classe chamada conexão que conecta ao banco de dados.
 * Observe a diferença entre esse formato de codificação e o uti-
 * lizado na linguagem estrutural (veja o caderno de programação
 * web).
 */
class Conexao{
    //Atributos da classe conexão.
    //Essas informações são relevantes para acesso ao banco. Por
    //exemplo, host, usuário e senha.
    private $host;
    private $usuario;
    private $senha;
    private $banco;
    private $conexao;

    //Construtor da classe conexão. Este método é responsável
    //por inicializar as variáveis do objeto conexão a ser criado
    function __construct($host, $usuario, $senha, $banco)
    {
        //Observe a utilização da variável $this. Neste caso,
        //ela está servindo para diferenciar o atributo da
        //classe (o que está apontado por this) e os argu-
        //mentos da função __construct
        $this->host = $host;
        $this->usuario = $usuario;
        $this->senha = $senha;
        $this->banco = $banco;

        //Observe que o atributo conexão não está sendo ini-
        //cializado. Este será inicializado no método conectar.
    }
```

```php
    //Método que conecta ao banco de dados
    //Retorna true se a concexão foi realizada com sucesso e false
    //caso contrário
    function conectar()
    {
        //Realiza solicita conexão ao banco de dados
        $this->conexao = mysqli_connect(
            $this->host,
            $this->usuario,
            $this->senha,
            $this->banco);
        //Caso tenha ocorrido algum erro, avisa o usuário sobre o
        //erro e retorna false.
        if (mysqli_connect_errno($this->conexao))
        {

            return false;
        }
        //Se tudo deu certo! Seta a codificação apra UTF8 e retorna true.
        else
        {
            mysqli_query($this->conexao, "SET NAMES 'utf8';");
            return true;
        }
    }

    //Função que executa queries ("buscas") no banco de dados.
    function executarQuery($sql)
    {
        return mysqli_query($this->conexao, $sql);
    }

    //Funcao que executa uma busca e retorna o primeiro registro encontrado.
    function obtemPrimeiroRegistroQuery($query)
    {
        // Enviamos o pedido pela conexéo e guardamos a resposta em $linhas
        $linhas = $this->executarQuery($query);
        // retorna a primeira linha
        return mysqli_fetch_array($linhas);
    }
}
?>
```

Figura 5.Classe Conexão. Fonte: Próprio autor.

No link abaixo você verá um artigo completo sobre conexão com banco de dados MySQL, utilizando PHP orientado a objetos. Leia o texto e se concentre nos aspectos teóricos do artigo dado porque utilizamos uma forma diferente de escrever o código.

http://www.escolacriatividade.com/php-orientado-a-objetos-conexao-a-banco-de-dados/

Classe Básica Filme

A classe básica filme serve para encapsularmos os dados dos filmes de nossa loja. Os dados da classe filme são: título, código, sinopse, quantidade e trailer. Para cada atributo da classe serão criados métodos get e set e o resultado você pode conferir na Figura 6. Observe que não serão explicados novamente esses conceitos neste ponto do texto, pois já foram apresentados na seção de orientação a objetos.

```php
<?php
    //Classe que encpsula os dados do objeto filme
    class Filme
    {
        //Dados de filme
        private $titulo;
        private $codigo;
        private $sinopse;
        private $quantidade;
        private $trailer;

        //Construtuor
        public function __construct($titulo,$codigo,$sinopse,$quantidade,$trailer)
        {
            $this->titulo = $titulo;
            $this->codigo = $codigo;
            $this->sinopse = $sinopse;
            $this->quantidade = $quantidade;
            $this->trailer = $trailer;
        }

        //Daqui em diante serão criados métodos get e set
        //para cada atributo de Filme.
        public function setTitulo($titulo)
        {
            $this->titulo = $titulo;
        }

        public function getTitulo()
        {
            return $this->titulo;
        }

        public function setCodigo($codigo)
        {
            $this->codigo = $codigo;
        }

        public function getCodigo()
        {
            return $this->codigo;
        }

        public function setSinopse($sinopse)
        {
            $this->sinopse = $sinopse;
        }

```


```php
        public function getSinopse()
        {
            return $this->sinopse;
        }

        public function setQuantidade($quantidade)
        {
            $this->quantidade = $quantidade;
        }

        public function getQuantidade()
        {
            return $this->quantidade;
        }

        public function setTrailer($trailer)
        {
            $this->trailer = $trailer;
        }

        public function getTrailer()
        {
            return $this->trailer;
        }
    }
?>
```

Figura 6. Classe Filme. Fonte: Próprio autor.

Repositório de dados

Para acesso dos dados do banco vamos utilizar uma classe para manipulação dos mesmos. Assim, sempre que quisermos ler ou alterar um elemento do banco de dados, vamos acessar essa classe que, em seguida, utilizará a classe Conexao. Essa classe é chamada de repositório e funciona como uma 'ponte' entre a aplicação e o local onde os dados são guardados. A Figura 6 apresenta sua relação com o restante da aplicação.

Figura 6. Esquema de acesso aos dados do Sistema.

Fonte: Próprio autor.

A partir da utilização do repositório, a aplicação não necessita executar comandos SQL diretamente. Portanto, há uma separação entre o código da aplicação e o controle da persistência dos dados. O código do repositório é mostrado na Figura 7 e já contém métodos para inserir, alterar e deletar filmes da nossa locadora virtual. Além disso, foi criada uma interface para que outros tipos de repositórios possam ser criados, além do repositório que utiliza MySQL. Caso outra forma de guardar os dados da aplicação fosse necessária (arquivos, por exemplo), não precisaríamos alterar nada acima do repositório (Figura 6). Apenas o repositório seria alterado. Crie no seu projeto um arquivo repositorio_filmes.php e adicione o código abaixo. Ele está todo comentado e não será difícil entender os seus conceitos, dado que você chegou até aqui.


```php
<?php
    require 'conexao.php';
    include 'filme.php';

    //Interface de repositorio para filmes. Todos
    //os repositorios de filmes devem conter pelo menos
    //os métodos aqui listados. Caso contrário, o código
    //nem compila.
    interface IRepositorioFilmes{
        public function cadastrarFilme($filme);
        public function removerFilme($filme);
        public function atualizarFilme($filme);
        public function buscarFilme($codigo);
        public function getListaFilmes();
    }

    //Classe de repositório de filmes que implementa IRepositorioFilmes
    class RepositorioFilmesMySQL implements IRepositorioFilmes{

        //Objeto conexão que guardará a conexão com o banco
        private $conexao;

        //Construtor do repositório de filmes
        public function __construct()
        {
            //Cria o objeto conexão que será responsável pelas chamadas ao banco de dados
            $this->conexao = new Conexao("localhost", "root", "", "popcorntv");
            //Conecta ao banco de dados
            if($this->conexao->conectar() == false)
            {
                echo "Erro" . mysqli_error();
            }

        }

        //Cadastra um novo filme. Observe que a SQL é preparada e enviada para o banco
        //isso valo para o restante dos métodos dessa classe
        public function cadastrarFilme($filme)
        {
            $titulo = $filme->getTitulo();
            $sinopse = $filme->getSinopse();
            $quantidade = $filme->getQuantidade();
            $trailer = $filme->getTrailer();

            $sql = "INSERT INTO filme (titulo, codigo, sinopse,quantidade, trailer) VALUES
                ('$titulo', NULL, '$sinopse', '$quantidade', '$trailer')";

            $this->conexao->executarQuery($sql);
        }
```

```php
50
51          //Remove um filme do banco de dados
52          public function removerFilme($codigo)
53          {
54              $sql = "DELETE FROM filme WHERE codigo = '$codigo'";
55              $this->conexao->executarQuery($sql);
56          }
57
58          //Atualiza a informação de um dado filme no banco de dados
59          public function atualizarFilme($filme)
60          {
61              $titulo = $filme->getTitulo();
62              $codigo = $filme->getCodigo();
63              $sinopse = $filme->getSinopse();
64              $quantidade = $filme->getQuantidade();
65              $trailer = $filme->getTrailer();
66
67              $sql = "UPDATE filme SET titulo='$titulo', sinopse='$sinopse',
68                      quantidade='$quantidade', trailer='$trailer'
69                      WHERE codigo='$codigo'";
70
71              $this->conexao->executarQuery($sql);
72          }
73
74          //Busca um novo filme a partir de seu código.
75          //Observe que um novo objeto filme é criado baseado com o que
76          //é retornado do banco.
77          public function buscarFilme($codigo)
78          {
79              //Obtem o primeiro registro do select abaixo
80              $linha = $this->conexao->obtemPrimeiroRegistroQuery
81                  ("SELECT * FROM filme WHERE codigo='$codigo'");
82
83              //Cria um novo objeto filme baseado na busca acima
84              $filme = new Filme(
85                  $linha['titulo'],
86                  $linha['codigo'],
87                  $linha['sinopse'],
88                  $linha['quantidade'],
89                  $linha['trailer']);
90              return $filme;
91          }
```


```
92
93          public function getListaFilmes()
94          {
95              //Obtem a lista de todos os filmes cadastrados
96              $listagem = $this->conexao->executarQuery("SELECT * FROM filme");
97              //Cria um novo array onde os objetos filmes serão guardados
98              $arrayFilmes = array();
99
100             //Varre a lista de entradas da tabela filmes e
101             //cria um novo objeto filme para cada entrada da
102             //tabela
103             while($linha = mysqli_fetch_array($listagem)){
104                 $filme = new Filme(
105                     $linha['titulo'],
106                     $linha['codigo'],
107                     $linha['sinopse'],
108                     $linha['quantidade'],
109                     $linha['trailer']);
110                 array_push($arrayFilmes, $filme);
111             }
112
113             //Retorna o array de filmes
114             return $arrayFilmes;
115         }
116     }
117
118     //Cria o objeto repositório de filmes. Esse objeto será
119     //acessado pelo restante da aplicação para receber e enviar
120     //objetos filme ao banco de dados.
121     $repositorio = new RepositorioFilmesMySQL();
    ?>
```

Figura 7. Classe de repositório dos dados. Fonte: Próprio autor.

O repositório de dados é um padrão de projeto muito utilizado em orientação a objetos. Veja mais conceitos relacionados a este tema em:

http://sergiotaborda.wordpress.com/desenvolvimento-de-software/java/patterns/repository/

Listando dados

Vamos agora listar os dados da tabela filmes e apresentá-la na tela. Vamos fazer essa mesma listagem utilizando código orientado a objetos. Para isso, vamos alterar a página 'index.php' e fazer duas alterações. A primeira diz respeito ao início da página que irá criar o repositório para obter a lista de todos os filmes cadastrados no banco. O trecho de código é mostrado a seguir:

```
<?php
        //Cria um objeto repositório e inclui as classes básicas
        require 'repositorio_filmes.php';

        //Obtem a lista de todos os filmes cadastrados
        $filmes = $repositorio->getListaFilmes();
?>
```

A outra mudança (código logo abaixo) trata-se de uma alteração referente à tabela de filmes mostrada na página html. Enquanto o projeto base possui uma lista de filmes fixos, nós vamos fazer com que essa tabela possua dados trazidos do banco. Observe que existe um loop (while) que varre todos os elementos do array de filmes do banco. Para cada elemento do array, um botão alterar, para o nome do filme, e um botão excluir são inseridos. O código final é mostrado na Figura 8.

```
while($filmeTemporario = array_shift($filmes))
{
?>
        <tr>
```


```
                <td class="col-md-1">
                        <a class="btn btn-default" href="#" role="button">Alterar</a>
                </td>

                <td class="col-md-6"> <?php echo $filmeTemporario->getTitulo() ?> </td>

                <td class="col-md-1">
                        <a class="btn btn-danger" href="#" role="button">Excluir</a>
                </td>
        </tr>
<?php
}
?>
```

Observe que o nome do filme é obtido por meio do método getTitulo(), pois o atributo titulo da classe Filme é privado. Além disso, por hora, os botões alterar e excluir estão sem funcionalidade implementada.

Fique atento! Algumas partes do código não foram copiadas (observe o número das linhas). Copie o código para o seu projeto com cuidado. O local onde o código foi alterado está marcado.

index.php

```php
<?php
    //Cria um objeto repositório e inclui as classes
    require 'repositorio_filmes.php';

    //Obtem a lista de todos os filmes cadastrados
    $filmes = $repositorio->getListaFilmes();
?>

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <meta charset="utf-8">
        <title>Popcorn TV</title>
        <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
        <link href="css/bootstrap.min.css" rel="stylesheet">
        <!--[if lt IE 9]>
            <script src="//html5shim.googlecode.com/svn/trunk/html5.js"></script>
        <![endif]-->
        <link href="css/styles.css" rel="stylesheet">
    </head>
    <body>
<div class="wrapper">
    <div class="box">
        <div class="row">
```

Anotações

```
116                 <table class="table table-hover">
117                     <tr>
118                         <th></th>
119                         <th>Filme</th>
120                         <th></th>
121                     </tr>
122                     <?php
123
124                 while($filmeTemporario = array_shift($filmes))
125                 {
126                 ?>
127                     <tr>
128                         <td class="col-md-1">
129                             <a class="btn btn-default" href="#" role="button">Alterar</a>
130                         </td>
131
132                         <td class="col-md-6"> <?php echo $filmeTemporario->getTitulo() ?> </td>
133
134                         <td class="col-md-1">
135                             <a class="btn btn-danger" href="#" role="button">Excluir</a>
136                         </td>
137                     </tr>
138                 <?php
139                 }
140                 ?>
141                 </table>
142
143
144                 <hr>
145
146                 <div class="row" id="footer">
147                   <div class="col-sm-6">
148
149                   </div>
150                   <div class="col-sm-6">
151                     <p>
152                     <a href="#" class="pull-right">©Copyright Inc.</a>
153                     </p>
154                   </div>
155                 </div>
156
157               <hr>
```

Figura 8. Código de index.html para listar os dados do banco.

Fonte: próprio autor.

O resultado pode ser visto na Figura 9.

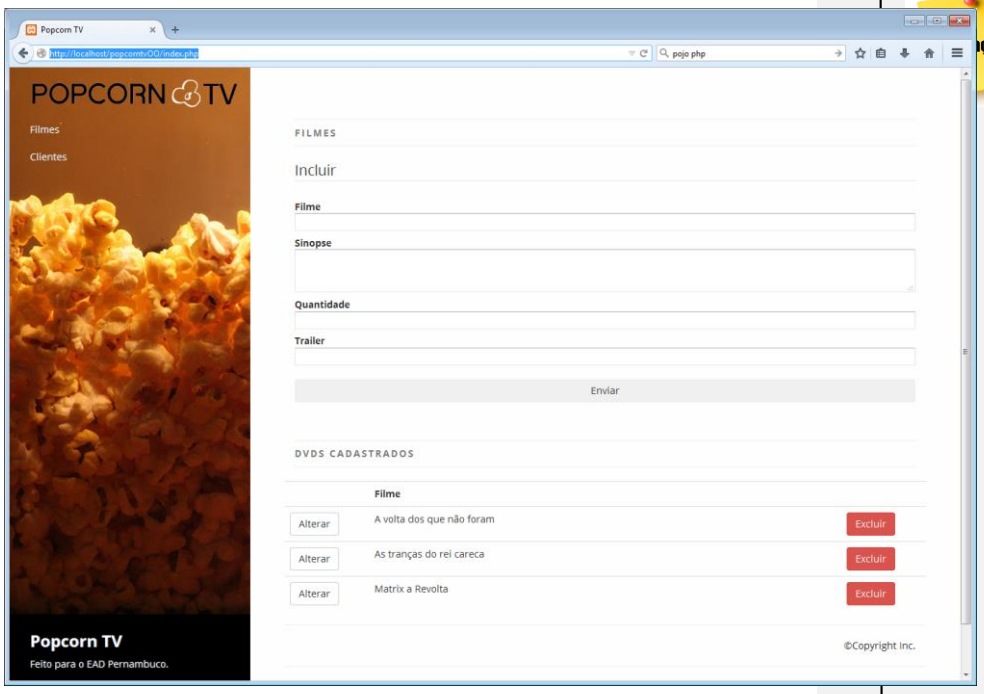

Figura 9. Listagem de Filmes. Fonte: Próprio autor.

Incluindo novos filmes.

Agora que podemos ver os dados no banco vamos adicionar dados. O processo é muito parecido, mas antes teremos que receber dados do usuário. Precisaremos de um formulário que receberá os valores fornecidos pelo usuário.

O modelo já contém um formulário em HTML que vamos modificá-lo para nossas necessidades. Ele vai servir tanto para incluir como para alterar os dados. O formulário em HTML irá enviar os dados pelo método **post** para a página **inserir_filme.php** que faremos mais a frente, será nele que conectaremos com o banco. Por agora você deve terminar a tag **<form>** incluindo os parâmetros action e post. A Figura 10 apresenta o código que envia os dados de inserção.

index.php × | template_topo.php | template_rodape.php | conexao.php | listar_produto.php

```
25
26              <!-- main -->
27              <div class="column col-sm-9" id="main">
28                  <div class="padding">
29                      <div class="full col-sm-9">
30
31                          <!-- content -->
32                          <div class="col-sm-12" id="featured">
33                            <div class="page-header text-muted">Filmes</div>
34                            <form class="form-horizontal" action="incluir_filme.php" method="post">
35                              <fieldset>
36
37                              <!-- Form Name -->
38                              <legend>Incluir</legend>
39
40                              <!-- Text input-->
41                              <div class="control-group">
42                                <label class="control-label" for="filme">Filme</label>
43                                <div class="controls">
44                                  <input id="filme" name="filme" type="text" />
45
46                                </div>
47                              </div>
48
49                              <!-- Textarea -->
50                              <div class="control-group">
51                                <label class="control-label" for="sinopse">Sinopse</label>
52                                <div class="controls">
53                                  <textarea id="sinopse" name="sinopse"></textarea>
54                                </div>
55                              </div>
```

Figura 10. Alterações no formulário para incluir um novo filme.

Fonte: Próprio autor

Anotações

Além do código de inserção do filme, é necessário que se crie uma página php que receba os dados do formulário e envie para o repositório. Essa classe é mostrada na Figura 11.

index.php | popcorntvBase | index.php | popcorntvOO | incluir_filme.php ×

```php
<?php
    require 'repositorio_filmes.php';

    //Cria um novo objeto com os dados recebidos pelo form
    $filmeRecebido = new Filme($_REQUEST['titulo'], null, $_REQUEST['sinopse'], $_REQUEST['quantidade'], $_REQUEST['trailer']);

    //Envia para o repositorio
    $repositorio->cadastrarFilme($filmeRecebido);

    header('Location: index.php');
    exit;
?>
```

Figura 11. Código de incluir_filme.php. Fonte: Próprio autor.

Removendo Filmes

Para que possamos remover um filme, é necessário do código do filme. Assim, para cada entrada da tabela de filmes em 'index.php' vamos inserir um hiperlink com o código do filme apresentado para a página 'excluir_filme.php'. A Figura 12 apresenta a alteração em questão.

```php
while($filmeTemporario = array_shift($filmes))
{
?>
    <tr>
        <td class="col-md-1">
            <a class="btn btn-default" href="#" role="button">Alterar</a>
        </td>

        <td class="col-md-6"> <?php echo $filmeTemporario->getTitulo() ?> </td>

        <td class="col-md-1">
            <a class="btn btn-danger" href="excluir_filme.php?codigo=<?= $filmeTemporario->getCodigo(); ?>" role="button">Excluir</a>
        </td>
    </tr>
<?php
}
?>
```

Figura 12. Envio de código do filme para exclusão no banco de dados.

Fonte: Próprio autor.

Anotações

O código da página excluir_filme é mostrado a seguir (Figura 13). Observe que apenas o código do filme foi necessário para que o repositório delete a entrada.

excluir_filme.php

```php
<?php
    require 'repositorio_filmes.php';

    $repositorio->removerFilme($_REQUEST['codigo']);

    header('Location: index.php');
    exit;
?>
```

Figura 13. Código fonte de excluir_filme.php. Fonte: Próprio autor.

Alterando filmes

Para a alteração de filmes vamos utilizar uma variável $destino que indicará se o formulário vai enviar os dados para a página 'incluir_filme.php' ou 'alterar_filme.php' (veja a seguir). As respectivas fontes são mostradas nas Figuras 14 e 15. Na Figura 15 temos o arquivo contendo todas as funcionalidades anteriores (exemplo: remover, inserir).

alterar_filme.php

```php
<?php
    require 'repositorio_filmes.php';

    //Cria um objeto filme com as entradas recebidas pelo usuário
    $filmeRecebido = new Filme($_REQUEST['titulo'], $_REQUEST['codigo'], $_REQUEST['sinopse'], $_REQUEST['quantidade'], $_REQUEST['trailer']);

    //Atualiza o filme existente no banco com esse recebido pelo form
    $repositorio->atualizarFilme($filmeRecebido);

    header('Location: index.php');
    exit;
?>
```

Figura 14. Código de alterar filme. Fonte próprio autor.

index.php

```php
<?php
    require 'repositorio_filmes.php';

    //Obtem a lista de todos os filmes cadastrados
    $filmes = $repositorio->getListaFilmes();

    // A variável $destino aponta para onde vamos enviar os dados do formulério.
    // Inicialmente vai para inserir_filme.php
    $destino = "incluir_filme.php";

    // Se recebermos uma variável codigo pelo metodo GET faça o seguinte
    if (isset($_GET['codigo'])) {
        $codigo = $_GET['codigo']; // Guardamos o codigo enviado na variével $codigo
        // Obtemos o objeto filme relativo ao código
        $filme = $repositorio->buscarFilme($codigo);
        // Agora o formulário enviar os dados para alterar_filme.php
        $destino = "alterar_filme.php";
        // Vamos acrescentar este campo oculto no formulário que contem o codigo do registro
        $oculto = '<input type="hidden" name="codigo" value="'. $codigo .'" />';
    }
?>

<!DOCTYPE html>
<html lang="en">
    <head>
        <meta http-equiv="content-type" content="text/html; charset=UTF-8">
        <meta charset="utf-8">
        <title>Popcorn TV</title>
        <meta name="viewport" content="width=device-width, initial-scale=1, maximum-scale=1">
        <link href="css/bootstrap.min.css" rel="stylesheet">
        <!--[if lt IE 9]>
            <script src="//html5shim.googlecode.com/svn/trunk/html5.js"></script>
        <![endif]-->
        <link href="css/styles.css" rel="stylesheet">
    </head>
    <body>
<div class="wrapper">
    <div class="box">
        <div class="row">

            <!-- sidebar -->
            <div class="column col-sm-3" id="sidebar">
                <a class="logo" href="#"><span>Popcorn TV</span></a>
                <ul class="nav">
                    <li class="active"><a href="index.php">Filmes</a>
                    </li>
                    <li><a href="#">Clientes</a>
                    </li>
                </ul>
```

Anotações

```
            <ul class="nav hidden-xs" id="sidebar-footer">
                <li>
                  <a href="#"><h3>Popcorn TV</h3>Feito para o EAD Pernambuco.</a>
                </li>
            </ul>
        </div>
        <!-- /sidebar -->

        <!-- main -->
        <div class="column col-sm-9" id="main">
            <div class="padding">
                <div class="full col-sm-9">

                    <!-- content -->
                    <div class="col-sm-12" id="featured">
                      <div class="page-header text-muted">Filmes</div>
                      <form class="form-horizontal" action="<?=$destino; ?>" method="post">
                        <?= @$oculto; ?>
                        <fieldset>

                        <!-- Form Name -->
                        <legend>Incluir</legend>

                        <!-- Text input-->
                        <div class="control-group">
                          <label class="control-label" for="titulo">Filme</label>
                          <div class="controls">
                            <input id="titulo" name="titulo" type="text" value="<?php echo isset($filme)?$filme->getTitulo():""; ?>"
                                placeholder="" class="input-xxlarge">
                          </div>
                        </div>

                        <!-- Textarea -->
                        <div class="control-group">
                          <label class="control-label" for="sinopse">Sinopse</label>
                          <div class="controls">
                            <textarea id="sinopse" name="sinopse"><?php echo isset($filme)?$filme->getSinopse():""; ?></textarea>
                          </div>
                        </div>

                        <!-- Text input-->
                        <div class="control-group">
                          <label class="control-label" for="quantidade">Quantidade</label>
                          <div class="controls">
                            <input id="cartaz" name="quantidade" type="text" value="<?php echo isset($filme)?$filme->getQuantidade():""; ?>"
                                placeholder="" class="input-xxlarge">

                          </div>
                        </div>

                        <!-- Text input-->
                        <div class="control-group">
                          <label class="control-label" for="trailer">Trailer</label>
                          <div class="controls">
                            <input id="trailer" name="trailer" type="text"
                                value="<?php echo isset($filme)?$filme->getTrailer():""; ?>" placeholder="" class="input-xxlarge">

                          </div>
                        </div>

                        <!-- Button -->
                        <div class="control-group">
                          <label class="control-label" for=""></label>
                          <div class="controls">
                            <input type="submit" class="btn btn-inverse" value="Enviar" />
                          </div>
                        </div>

                        </fieldset>
                        </form>
                    </div>

                    <!--/top story-->


                    <div class="col-sm-12" id="stories">
                      <div class="page-header text-muted divider">
                        DVDs Cadastrados
                      </div>
                    </div>

                    <table class="table table-hover">
                        <tr>
                            <th></th>
                            <th>Filme</th>
                            <th></th>
                        </tr>
```

Anotações

```
                        <?php

                        while($filmeTemporario = array_shift($filmes))
                        {
                        ?>
                            <tr>
                                <td class="col-md-1">
                                    <a class="btn btn-default"
                                    href="index.php?codigo=<?= $filmeTemporario->getCodigo(); ?>" role="button">Alterar</a>
                                </td>

                                <td class="col-md-6"> <?php echo $filmeTemporario->getTitulo() ?> </td>

                                <td class="col-md-1">
                                    <a class="btn btn-danger"
                                    href="excluir_filme.php?codigo=<?= $filmeTemporario->getCodigo(); ?>" role="button">Excluir</a>
                                </td>
                            </tr>
                        <?php
                        }
                        ?>

                    </table>


                    <hr>

                    <div class="row" id="footer">
                      <div class="col-sm-6">

                      </div>
                      <div class="col-sm-6">
                        <p>
                        <a href="#" class="pull-right">©Copyright Inc.</a>
                        </p>
                      </div>
                    </div>

                  <hr>

                  <h3 class="text-center">
                  <a href="#" target="ext">EAD Pernambuco</a>
                  </h3>

                  <hr>


                </div><!-- /col-9 -->
              </div><!-- /padding -->
          </div>
          <!-- /main -->
        </div>
    </div>
</div>
    <!-- script references -->
        <script src="//ajax.googleapis.com/ajax/libs/jquery/2.0.2/jquery.min.js"></script>
        <script src="js/bootstrap.min.js"></script>
    </body>
</html>
```

Figura 15. Código fonte index.php. Fonte: Próprio autor.

Conclusão

Este caderno apresenta um guia introdutório sobre orientação à objetos utilizando como base a linguagem PHP. Apesar deste caderno ser focado na linguagem PHP, os conceitos apresentados aqui são genéricos o suficiente para serem utilizados em outras linguagens que utilizam o mesmo paradigma.

Foram discutidos conceitos como: classes, atributos, métodos, herança, interfaces entre outros. Além disso, no final do caderno é apresentada uma aplicação prática multidisciplinar com os conceitos aprendidos em várias disciplinas como por exemplo banco de dados.

Referências

1 – Dall'oglio, Pablo, PHP Programando com Orientação a Objetos, Ed. Novatec. 2ª Ed., 2009.

2 - Anthony Sintes, Anthony, Aprenda programação orientada a objetos em 21 dias, Ed. Pearson, 2002.

3 – Mendes, Douglas Rocha, Programação Java - Com Ênfase Em Orientação a Objetos, Ed. Novatec, 2009.

4 – Xavier, Fabrício S. V., Php - do Básico à Orientação de Objetos, Ed. Ciência Moderna, 2008.

5 – Araújo, Everton Coimbra, Orientação a Objetos Com Java, Ed. Visual Books, 2008.

6- Curso online PHP orientado a objetos, Por Hugo vasconcelos link: https://www.youtube.com/watch?v=rAkJOH5Gz7U

Anotações

CURRÍCULO DO PROFESSOR-PESQUISADOR

Bruno Silva é formado em Engenharia da Computação pela UFPE, com mestrado em Ciência da Computação pela UFPE e doutorando pela UFPE e a universidade Technische universitat ilmenau. Atualmente é professor da Universidade Federal Rural de Pernambuco. Participa de projetos de cooperação científica entre Brasil e Empresas internacionais como EMC, HP e Foxconn.