PHP Orientado a Objetos para Iniciantes

Programação orientada a objetos é um estilo de programação que permite os desenvolvedores agruparem tarefas semelhantes em **classes**. Isso ajuda a mantermos-nos dentro do princípio ***não se repita***, além de facilitar a manutenção do código.

Compreendendo Objetos e Classes

Há uma grande confusão na POO: quando programadores de longa data falam sobre objetos e classes, esses termos parecem permutáveis entre si. Porém, não é bem assim, mesmo que a diferença entre eles seja um pouco complicada de perceber, no início.

Uma classe, por exemplo, é como uma **planta baixa para uma casa**. Ela define a forma da casa no papel, com as relações entre as diferentes partes da casa, claramente definidas e planejadas, mesmo a casa ainda não existindo.

Um objeto, por outro lado, **seria a casa de verdade**, construída de acordo com a planta baixa.

Estruturando Classes

A sintaxe para criar uma classe é bem direta: declare-a usando a palavra chave **class**, seguida do nome da classe e um par de chaves ({}):

```php
<?php

class MinhaClasse
{
      // As propriedades e métodos da Classe vem aqui
}
?>
```

Após criar a classe, ela pode ser instanciada e guardada em alguma variável usando a palavra chave new:

```php
$obj = new MinhaClasse;
```

Para vermos o conteúdo da classe, usamos var_dump():

> Esta função exibe informações estruturadas sobre uma ou mais expressões que incluem seu tipo e valor.

```php
var_dump($obj);
```

Experimente isso, colocando todo o código anterior em um único arquivo php, chamado teste.php na sua pasta local de /htdocs/**Aula2and3**

```php
<?php

class MinhaClasse
{
  // As propriedades e métodos da Classe vem aqui
}

$obj = new MinhaClasse;

var_dump($obj);

?>
```

Teste no navegador: http://localhost/Aula2and3/teste.php

De uma forma bem simples, você acabou de criar seu primeiro código em POO.

Definindo as Propriedades da Classe

Para adicionar dados à classe, usamos as **propriedades**, que são variáveis específicas à classe.

Elas funcionam de forma parecida às variáveis normais, exceto que elas estão ligadas ao objeto e só podem ser acessadas usando o objeto.

Para adicionar uma propriedade a **MinhaClasse,** adicione o seguinte trecho de código ao seu código:

```
<?php

class MinhaClasse
{
        public $prop1 = "Sou um propriedade de classe!";
}

$obj = new MinhaClasse;

var_dump($obj);

?>
```

Teste no navegador: http://localhost/Aula2and3/teste.php

A palavra chave public determina a visibilidade da propriedade, a qual você aprenderá mais sobre, nas próximas seções. Membros de classes declarados como públicos podem ser acessados de qualquer lugar.

Modifique o código do arquivo teste.php para ler a propriedade ao invés de mostrar todo o conteúdo da classe, dessa forma:

```
<?php

class MinhaClasse
{
  // As propriedades e métodos da Classe vem aqui
        public $prop1 = "Sou um propriedade de classe!";
}

$obj = new MinhaClasse;

echo $obj->prop1; // Mostra a saída/conteúdo da propriedade
?>
```

Teste no navegador: http://localhost/Aula2and3/teste.php

Definindo Métodos de Classe

Métodos são funções específicas das classes. Ações particulares que os objetos serão capazes de executar são definidas dentro das classes na forma de métodos.

Por exemplo, para criar métodos que atribuam e retornem o valor de uma propriedade, crie um novo arquivo em Aula2and3 chamado **TesteMetodo.php** e adicione o código a seguir:

```php
<?php

class MinhaClasse2
{
    public $propriedade1 = "Sou uma propriedade de classe";

    public function setPropriedade1($v)
    {
      $this->propriedade1 = $v;
    }

    public function getPropriedade1()
    {
      return "Método que imprime: ". $this->propriedade1 . "<br />";
    }
}//fim da classe

$obj = new MinhaClasse2;

echo "Olá.  $obj->propriedade1. <br>";
echo $obj->getPropriedade1();

  //atualizando a propriedade
  $obj->setPropriedade1("Novo valor a propriedade");
  echo "Olá ".$obj->propriedade1."<br>";
  echo $obj->getPropriedade1();
?>
```

"O poder da POO mostra-se ao usar múltiplas instâncias da mesma classe."

```php
<?php

class MinhaClasse2
{
  public $propriedade1 = "Sou uma propriedade de classe!";

  public function setPropriedade1($v)
  {
     $this->propriedade1 = $v;
  }

     public function getPropriedade1()
    {
        return "Método que imprime: ". $this->propriedade1 . "<br />";
    }
}// fim da minhaClasse2

   $obj = new MinhaClasse2;
   $obj2 = new MinhaClasse2;

   //atualizando a propriedade 1
   $obj->setPropriedade1("Novo valor a propriedade do objeto 1");
   echo "Olá ". $obj->propriedade1"<br>";
   echo $obj->getPropriedade1();
   echo "<br><br>";
   //atualizando a propriedade 2
   $obj2->setPropriedade1("Novo valor a propriedade do objeto 2");
   echo "Olá ". $obj2->propriedade1"<br>";
   echo $obj2->getPropriedade1();
?>
```

Usando Construtores e Destruidores

Quando um objeto é instanciado, é desejável que algumas coisas ocorram de cara.

Para lidar com isso, o PHP provê o método __construct(), que é chamado automaticamente quando um novo objeto é criado.

Para ilustrar os conceitos dos construtores, crie um novo arquivo chamado **TesteConstrutor.php** e adicione um construtor à classe MinhaClasse3 que mostre uma mensagem toda vez que uma nova instância for criada:

```
<?php

class MinhaClasse3
{
  public $prop1="Sou uma propriedade de classe!";

  public function __construct()
  {
     echo "A classe, ".  __CLASS__." ,  foi instanciada!<br />";
  }

  public function setProp1($newval)
  {
     $this->prop1 = $newval;
  }

  public function getProp1()
  {
     return $this->prop1 . "<br />";
  }
}

// Cria um novo objeto
$obj = new MinhaClasse3;

// Mostra o valor de $prop1
echo $obj->getProp1();

// Mostra uma mensagem ao final do arquivo
echo "Fim do arquivo.<br />";

?>
```

Nota : __CLASS__ retorna o nome da classe na qual foi usado.

Exemplo de construtor inicializando propriedades. Crie um arquivo chamado **TestaConstrutorInitialize.php:**

```
<?php
class Ponto {
    public  $x;
    public  $y;

     public function __construct($x, $y = 0) {
        $this->x = $x;
        $this->y = $y;
    }
}

// Passagem de ambos os argumentos.
$p1 = new Ponto(4, 5);

// Passar somente o argumento obrigatório, $y terá o valor padrão zero.
$p2 = new Ponto(4);
var_dump($p1);

echo "<br><br>";

var_dump($p2);
?>
```

Atividade: Implemente os métodos sets e gets para as duas propriedades $x e $y, e faça um teste para validar o código.

Para chamar uma função quando um objeto for destruído, o método __destruct() está disponível. Ele é útil para finalizar as tarefas da classe (encerrar uma conexão com a base de dados, por exemplo).

Mostre uma mensagem quando um objeto for destruído usando o método __destruct() no arquivo **TestaDestrutor.php**:

```php
<?php
class MinhaClasse{
  public $prop1 = "Sou uma propriedade de classe!";

  public function __construct()
  {
     echo 'A classe "', __CLASS__, '" foi instanciada!<br />';
  }

  public function __destruct()
  {
     echo 'A classe "', __CLASS__, '" foi destruída.<br />';
  }

  public function setProp1($newval)
  {
     $this->prop1 = $newval;
  }

  public function getProp1()
  {
     return $this->prop1 . "<br />";
  }
}

// Cria um novo objeto
$obj = new MinhaClasse();

// Mostra o valor de $prop1
echo $obj->getProp1();

// Mostra uma mensagem ao final do arquivo
echo "Fim do arquivo.<br />";
?>
```

Você deve usar um destrutor sempre que tem algum recurso que precisa de uma finalização, ou seja, precisa fechar uma conexão, um arquivo, etc. Mas, caso não o faça, o PHP irá se encarregar por si só. Digamos que seja apenas uma forma de você programador ter certeza que será feito no momento que deseja.

Atividade

Implemente as 4 operações básicas da matemática: soma, subtração, multiplicação e divisão.

- Crie um arquivo chamado **operacoes.php**
- Implemente uma classe chamada **Operacoes**
- Implemente 3 propriedades na classe: **a**, **b** e **c**
- Para cada operação, um método deve ser implementado para manipular as 3 propriedades da classe
- Faça pelo menos um caso de teste para cada operação implementada

Atividade a ser entregue via moodle.

Estudo de caso: Aluno

Crie um arquivo chamado aluno.php:

```
<?php
class Aluno{
	public $nome;
	public $cpf;
	public $matricula;
	public $endereco;

	//construtor da classe
	public function  __construct(){
		$this->preparaAluno();
	}

	function preparaAluno(){
		$this->nome = "Patrick";
		$this->cpf = "99999999999";
		$this->endereco = "Rua Fulano de Tal número 0 apt 999";
		$this->matricula = "02353";
	}
}
?>
```

Abaixo segue um exemplo de como instanciar a classe e acessar os atributos. Para iniciarmos, crie outro arquivo chamado imprimeDados.php.

```
<?php
require_once 'aluno.php';

$aluno = new Aluno();

echo $aluno->nome;
echo "<br>";
echo $aluno->cpf;
echo "<br>";
echo $aluno->endereco;
echo "<br>";
echo $aluno->matricula;
?>
```

Teste via navegador : http://localhost/Aula2and3/imprimeDados.php

Outro conceito importante em orientação a objetos é a visibilidade, a visibilidade de um método ou um atributo podem ser definidos ao pré-fixarmos as palavras chave, **public**, **private** ou **protected**, se não prefixados, por padrão o PHP entende que seja **public**.

- A visibilidade public significa que pode ser acessado por todo mundo;
- A visibilidade protected limita o acesso a classes herdadas;
- A visibilidade private limita o acesso apenas para a classe que definiu o item.

Para entendermos melhor visibilidade, temos que entender o conceito de herança.

A herança permite que classes compartilhem métodos e atributos, ela é peça fundamental para o reaproveitamento de código, com ela as classes que a estendem podem utilizar os métodos.
No exemplo abaixo incrementamos a classe Aluno Utilizando visibilidade e herança.

```php
<?php
class Aluno{
	public $nome;
	protected $cpf;
	public $matricula;
	private $endereco;
	public function  __construct(){
		$this->preparaAluno();
	}

	function preparaAluno(){
		$this->nome = "Patrick";
		$this->cpf = "99999999999";
		$this->endereco = "Rua Fulano de Tal número 0 apt 999";
		$this->matricula = "02353";
	}

	public function setNome($n){
		$this->nome=$n;
	}
	public function getNome(){
		return $this->nome;
	}

	public function setCpf($n){
		$this->cpf=$n;
	}
	public function getCpf(){
		return $this->cpf;
	}

	public function setMatricula($n){
		$this->matricula=$n;
	}
	public function getMatricula(){
		return $this->matricula;
	}
	public function setEndereco($n){
		$this->endereco=$n;
	}
	public function getEndereco(){
		return $this->endereco;
	}
} ?>
```

A seguir, mostramos como serão recuperados os dados em outra classe que vai instanciar a classe Aluno. Vejamos o exemplo abaixo em recuperaAluno.php:

```
<?php
require_once 'aluno.php';

class AcessaAluno{

	function imprimeAluno(){
		$aluno = new Aluno();
		echo "$aluno->nome <br>";
		echo "$aluno->cpf <br>";
		echo "$aluno->matricula <br>";
		echo "$aluno->endereco <br>";
	}
}
	$acesso = new AcessaAluno();
	$acesso->imprimeAluno();

?>
```

teste via navegador http://localhost/Aula2and3/recuperaAluno.php

No código acima ocorreram dois erros fatais, um na hora de imprimir o cpf e outro ao tentar imprimir o endereço, apenas o nome e matrícula será impresso, pois é o único atributo público da classe. Para acessarmos os outros atributos deveremos fazer o que mostra o código abaixo.

```
<?php
require_once 'aluno.php';
class AcessaAluno{
	function imprimeALuno(){
		$aluno = new Aluno();
		echo $aluno->getNome()."<br>";
		echo $aluno->getCpf(). "<br>";
		echo $aluno->getMatricula() ."<br>";
		echo $aluno->getEndereco() ."<br>";
	}
}//fim classe
	$acesso = new AcessaAluno();
	$acesso->imprimeAluno();
?>
```

teste via navegador http://localhost/Aula2and3/recuperaAluno.php

A seguir veremos um exemplo prático de herança. Por exemplo, podemos criar uma classe para aluno de graduação, que é um tipo de aluno. Vejamos o código de alunoGraduacao.php:

```
<?php
require_once 'aluno.php';

class alunoGraduacao extends Aluno{
	protected $escolaridade; // ensino médio ou técnico

	//construtor da classe
	public function  __construct($escolaridade){
		parent::__construct();
		$this->escolaridade = $escolaridade;
	}

	public function setEscolaridade($n){
		$this->escolaridade = $n;
	}
	public function getEscolaridade(){
		return $this->escolaridade;
	}
}
?>
```

Aluno de pós-graduação, que é um tipo de aluno. Vejamos o código de alunoPosGraduacao.php:

```php
<?php
require_once 'aluno.php';

class alunoPosGraduacao extends Aluno{
	protected $formacao;
	protected $programa;// especialização, mestrado, doutorado

	//construtor da classe
	public function  __construct($formacao, $prog){
		parent::__construct();
		$this->formacao = $formacao;
		$this->programa = $prog;
	}

	public function setFormacao($n){
		$this->formacao = $n;
	}
	public function getFormacao(){
		return $this->formacao;
	}
	public function setPrograma($p){
		$this->programa = $p;
	}
	public function getPrograma(){
		return $this->programa;
	}
}
?>
```

Para recuperarmos os dados, vamos implementar um arquivo de teste chamado testeHeranca.php:

```php
<?php

require_once 'alunoPosGraduacao.php';
require_once 'alunoGraduacao.php';

class Testando {
	function ImprimeAlunoGraduacao(){
		$graduacao = new alunoGraduacao("Ensino Médio");
		echo "Aluno de Graduação:<br>";
		echo "Nome:".$graduacao->getNome()."<br>";
		echo "CPF:".$graduacao->getCpf(). "<br>";
		echo "Matrícula:".$graduacao->getMatricula() ."<br>";
		echo "Endereço:".$graduacao->getEndereco() ."<br>";
		echo "Escolaridade:".$graduacao->getEscolaridade()."<br>";
	}
	function ImprimeAlunoPosGraduacao(){
		$posgrad = new alunoPosGraduacao("Geografia", "Especialização");
		echo "Aluno de Pós Graduação:<br>";
		echo "Nome:".$posgrad->getNome()."<br>";
		echo "CPF:".$posgrad->getCpf(). "<br>";
		echo "Matrícula:".$posgrad->getMatricula() ."<br>";
		echo "Endereço:".$posgrad->getEndereco() ."<br>";
		echo "Formação:".$posgrad->getFormacao()."<br>";
		echo "Programa:".$posgrad->getPrograma()."<br>";
	}
}
 $obj= new Testando();
 $obj->ImprimeAlunoGraduacao();
 echo "<br>**************************<br>";

 $obj->ImprimeAlunoPosGraduacao();
?>
```