

PHPOO – Parte 1

DIOGO CEZAR TEIXEIRA BATISTA
<http://inf.cp.utfpr.edu.br/diogo>
diogo@diogocezar.com

REVISÃO DE ORIENTAÇÃO A OBJETOS

objeto:

- representa alguma coisa física, tangível, uma idéia ou conceito;
- possui um estado (o que ele sabe? → atributos);
- possui um comportamento (o que ele é capaz de fazer, como ele reage a estímulos externos → métodos).

REVISÃO DE ORIENTAÇÃO A OBJETOS

classe:

- é um "molde" para a criação de objetos;
- fornecendo o seu comportamento padrão e a definição de todos os seus estados possíveis;
- exemplo: Classe Cliente.

instância:

- é uma ocorrência particular, identificada, de um objeto de uma determinada classe, com seu estado particular, independente de outras instâncias da mesma classe;
- exemplo: o objeto Cliente “Fernando Almeida”.

encapsulamento:

- encapsular: esconder os dados contidos nas propriedades (atributos) de uma classe;
- exemplo: não é necessário conhecer todo o funcionamento interno de um carro para poder dirigi-lo:
 - para isso é escondido por baixo da lataria tudo que faz com que o carro funcione, deixando apenas para o usuário o que é realmente necessário para se dirigir;
- na programação orientada a objetos é possível utilizar do encapsulamento dando permissões de acessos aos atributos das classes: private, protected e public, que serão estudados posteriormente.

```
proteger atributos privados  
com get e set
```


ocultamento de informações:

- deve ser possível utilizar um objeto apenas pelo conhecimento da sua estrutura externa (isto é, sua interface);
- mudanças na estrutura interna de um objeto (isto é, sua implementação) não devem afetar aos usuários do objeto.

polimorfismo:

- classes diferentes podem tratar uma entrada de dados de acordo com a sua necessidade (sobrescrita de um método);
- a entrada de dados deve resultar em uma saída de dados esperada.

herança ou especialização:

- uma nova classe pode ser definida em termos de uma classe pai, herdando o seus métodos e atributos;
- a nova classe especializa a classe pai, definindo apenas onde o seu comportamento deve ser diferente.

agregação e composição:

- objetos podem conter outros objetos como partes constituintes, imitando o mundo real onde objetos são construídos em função de outros objetos.
- Exemplo: Classe Funcionário contém Objeto Dependente

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

- a seguir uma pequena lista das características da programação orientada a objetos (POO) em PHP5.

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

- em PHP 5 podemos utilizar nomes pré-definidos para os métodos construtores e destrutores;
 - __construct (Construtor);
 - __destruct (Destrutor).
- ainda é possível utilizar o construtor com uma função com o nome idêntico a classe utilizada, por exemplo (descontinuado):

```
class Pessoa{  
    function Pessoa() {}  
}
```

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

acesso :

- public ← todos acessam
- private ← só é acessado dentro da classe
- protected ← só é acessado dentro da classe ou por seus herdeiros
- podemos utilizar os modificadores de acesso habituais da POO;
- estes modificadores servem para definir que métodos e propriedades das classes são acessíveis desde cada meio.

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

possibilidade de uso de interfaces:

- especifica quais métodos e variáveis outras classes devem implementar, sem definir como serão tratados;
- uma classe pode implementar várias interfaces ou conjuntos de métodos.
 - Na prática, o uso de interfaces é utilizado para suprir a falta de herança múltipla de linguagens como PHP ou Java.

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

métodos e classes final:

- pode-se indicar que um método é "final".
- com isso, não se permite sobrescrever esse método, em uma nova classe que o herde;
- se a classe é "final", o que se indica é que esta classe não permite ser herdada por outra classe.

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

atributos e métodos static:

- podemos fazer uso de atributos e métodos "static";
- são as propriedades e funcionalidades as quais se pode acessar a partir do nome de classe, sem a necessidade de haver instanciado um objeto de tal classe;
- pode-se dizer que tais métodos ou atributos pertencem a classe e não a um de seus objetos.

ALGUMAS CARACTERÍSTICAS DO TRABALHO COM POO EM PHP 5

classes e métodos abstratos:

- também é possível criar classes e métodos abstratos;
- as classes abstratas não são instanciadas, costumam ser utilizadas para herdá-las de outras classes que implementarão seu conteúdo;
- os métodos abstratos não podem ser chamados, utilizam-se para serem herdados por outras classes que implementarão seu conteúdo.

CLASSES E OBJETOS EM PHP

ContaCorrente.php

```
<?php
class ContaCorrente{

    public $saldo;

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

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

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

CLASSES E OBJETOS EM PHP

index.php

```
<?php
include("class/ContaCorrente.php");
$CC = new ContaCorrente(1000);
$CC->saque(500);
$CC->saque(500);
$CC->saque(10);
$CC->deposito(150);
echo $CC->saldo; // imprime 150
?>
```


CLASSES E OBJETOS EM PHP

- a palavra-chave class indica uma declaração de classe, delimitada por chaves;
- a classe deve utilizar a variável \$this para referenciar seus próprios métodos e atributos;
- para referenciar atributos e métodos utilizamos o operador ->

ESPECIFICADORES DE ACESSO

- public: pode ser acessado por qualquer classe;
- protected: pode ser acessado somente por quem estende sua classe;
- private: pode ser acessado somente por sua classe.

MÉTODOS E VARIÁVEIS ESTÁTICAS

- métodos e variáveis estáticas (static) podem ser acessados de qualquer lugar do código, sem a necessidade de se instanciar um objeto.

```
<?php
class Estatica{
    static $var = "Variável Estática";

    static function getStatic(){
        return Estatica::$var;
    }
}
echo Estatica::getStatic();
?>
```

- note que o operador para acessar métodos ou variáveis estáticas é ::

MÉTODOS E CLASSES FINAIS

- métodos final: não poderão ser sobre-escritos;
- classes final: não poderão ser herdadas;

```
<?php
final class ClasseFinal{
    // essa classe não poderá ser herdada
    final function getFinal() {
        // esse método não poderá ser sobrescrito
        return "Metodo Final";
    }
}
$FC = new ClasseFinal();
echo $FC->getFinal();
?>
```

CONSTRUTORES E DESTRUTORES

```
<?php
class ConsDestructor{
    function __construct(){
        echo "Metodo Construtor Invocado<br>";
    }
    function __destruct(){
        echo "Metodo Destrutor Invocado<br>";
    }
}
$ConsDestrutores = new ConsDestructor();
unset($ConsDestrutores);
?>
```

unset usado para
invocar o método
destrutor

Metodo Construtor Invocado

CLASSE ABSTRATA

```
<?php
abstract class Abstrata{
    protected $nome;
    public abstract function setNome($nome);
    public function getNome(){
        return $this->nome;
    }
}

class ClasseAbstrata extends Abstrata{
    public function setNome($nome){
        $this->nome = $nome;
    }
}

$classeAbstrata = new ClasseAbstrata();
$classeAbstrata->setNome("Pedro");
echo $classeAbstrata->nome;
echo $classeAbstrata->getNome();
?>
```

Todos os métodos declarados como abstract, **deverão** ser implementados na classe filha.

Pode-se implementar uma função.

Pode-se declarar atributos. **protected**, será visível na classe herdeira.

Pedro

INTERFACES

```
<?php
interface IPessoa{
    public function setNome($nome);
}
interface ITipo{
    public function setTipo($tipo);
}
class ClassePessoa implements IPessoa, ITipo{
    private $nome, $tipo;
    public function setNome($nome){
        $this->nome = $nome;
    }
    public function setTipo($tipo){
        $this->tipo = $tipo;
    }
    public function __toString(){
        $retorno .= "Nome: {$this->nome}<br>";
        $retorno .= "Tipo: {$this->tipo}";
        return $retorno;
    }
}
$IP = new ClassePessoa();
$IP->setNome("Carlos");
$IP->setTipo("Pessoa Física");
echo $IP;
?>
```

o método `__toString()`
possibilita a chamada de
`echo $IP`

```
Nome: Carlos
Tipo: Pessoa Física
```

DIFERENÇA ENTRE CLASSES ABSTRATAS E INTERFACES

- uma classe pode herdar múltiplas interfaces;
- uma interface é utilizada quando não existe a necessidade das classes derivadas herdarem métodos já implementados;
- interfaces não permitem a declaração de atributos, como as classes abstratas.

ATIVIDADES

- sabendo que, a função `get_object_vars($this)`, retorna um array com os atributos da classe atual, desenvolva um método `__toString()` genérico, que imprima todos os atributos do objeto da seguinte forma:

```
Nome: Diogo Cezar  
Endereco: Rua anchieta, 1369  
Telefone: (43)3523-2956  
Email: xgordo@gmail.com
```