

Programação II

Programação Orientada a Objetos



Programação Orientada a Objetos

- POO é um paradigma de programação que representa conceitos como “**objetos**”, os quais são formados por **atributos** (valores que descrevem as características dos objetos) e operações (conhecidas como **métodos**).
- Os objetos interagem uns com os outros a fim de atingir o objetivo do programa.
- As **classes** definem de forma abstrata a estrutura dos objetos, incluindo suas características (atributos, campos ou propriedades) e seu comportamento (métodos, operações).
 - *template* ou modelo
- Um **objeto** particular é uma instância de uma classe, criada em tempo de execução na memória.



Programação Orientada a Objetos

- Os valores dos atributos de um objeto em particular formam seu “estado”.
 - Atributos são variáveis da classe.
 - Um objeto na memória é uma unidade formada por seu estado e pelo comportamento definido em sua classe.
- O comportamento é definido pelos métodos, que determinam as “habilidades” destes objetos
 - Métodos são trechos de código (usualmente *functions*) que fazem parte da estrutura de uma classe e, conseqüentemente, das suas instâncias.



Pilares da POO

- Abstração
 - Representação de um objeto real.
 - Capacidade de descrever características essenciais do objeto que o distinguem de outros tipos de objetos.
- Encapsulamento
 - Disponibilizar uma interface pública para que outros objetos possam interagir com o objeto em questão, sem no entanto precisar conhecer seu funcionamento interno.
 - Evitar acesso direto às propriedades de um objeto.
 - Utilizar modificadores de visibilidade (*public*, *protected*, *private*), associados a métodos de acesso (*getters*) e métodos modificadores (*setters*).
- Herança (especialização, generalização)
 - Relação pai/filho, geral/específico.
 - Reutilização de código: o objeto abaixo na hierarquia irá herdar características de todos os objetos acima dele (seus “ancestrais”).
- Polimorfismo
 - Significa “muitas formas”.
 - Alterar a implementação de um método herdado, de modo que se comporte de modo diferente, de acordo com quem chamou a operação.

POO em PHP

- O PHP passou a suportar orientação a objetos a partir da versão 5. Antes disto, no modo não OO, os programadores utilizavam apenas programação estruturada, com a criação de arquivos de funções para facilitar o reuso de código.
- Todas as funções não OO da linguagem são definidas globalmente, levando à necessidade de utilizar prefixos em seus nomes. Pesquise por date no manual do PHP e veja que há inúmeras funções com nomes do tipo por `date_algumaCoisa`.

```
<?php
```

```
// exemplo não OO
```

```
date_default_timezone_set("America/Sao_Paulo");
```

```
$nextWeek = time() + (7 * 24 * 60 * 60);
```

```
echo "Hoje: ".date("d-m-Y")."<br>";
```

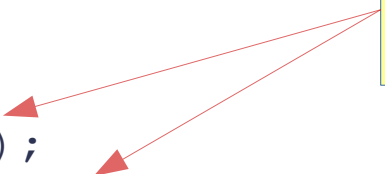
```
echo "Próxima semana: ".date("d-m-Y", $nextWeek)."<br>";
```

```
?>
```

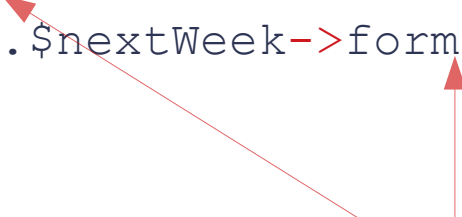
POO em PHP

```
<?php
// exemplo OO
$now = new DateTime();
$nextWeek = new DateTime("today + 1 week");
echo "Hoje: ".$now->format("d-m-Y")."<br>";
echo "Próxima semana: ".$nextWeek->format("d-m-Y")."<br>";
?>
```

Duas instâncias (ou objetos)
diferentes da classe DateTime



A classe DateTime possui o método
format. Portanto, os dois objetos
referenciados por \$now e \$nextWeek
também o possuem. Ao cada
chamada, o método se aplicará aos
dados do objeto que o chamou.



Saiba mais sobre POO em PHP:

<http://php.net/manual/en/language.oop5.php>

Criando Classes em PHP

- Uma classe em PHP é um conjunto de variáveis e funções relacionadas a essas variáveis.
 - As variáveis (atributos) e funções (métodos) são chamados genericamente de “membros” da classe.
- Cada classe se torna um tipo de dados.
- Para definir uma classe, deve-se utilizar a seguinte sintaxe (por convenção, o nome da classe deve ter inicial em maiúsculo):

```
class Nome_da_classe {  
    var $variavel1;  
    var $variavel2;  
  
    function funcao1 ($parâmetro) {  
        /*corpo da função*/  
    }  
}
```

Operadores

- **new**

- Variáveis do tipo de uma classe são chamadas de objetos, e devem ser criadas utilizando o operador new.

```
$variável = new $nome_da_classe;
```

- **\$this**

- é uma constante predefinida que somente pode ser utilizada dentro de um método da própria classe.
- Significa “esta instância” ou “este objeto”.

```
return this -> nome;
```

- **->**

- acessa membros (atributos ou métodos) de um objeto.

```
$variável->funcao1();
```

- **::**

- acessa membros estáticos e constantes de uma classe.

```
Classe::atributo
```


Exemplo

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

    function setNumero($n){
        $this->numero = $n;
    }
    function getSaldo() {
        return $this->saldo;
    }
    function credito($valor) {
        $this->saldo += $valor;
    }
}

$minhaconta = new Conta();
$minhaconta->setNumero(1234);
echo $minhaconta->getSaldo(); // não imprime nada
$minhaconta->credito(50);
$minhaconta->credito(75);
echo $minhaconta->getSaldo(); // imprime 125
?>
```

Ciclo de vida

- Objetos são criados, usados e descartados.
- Há métodos especiais chamados quando:
 - O objeto é criado (**construtor**)
 - O objeto é destruído (**destrutor**)
 - O destrutor não precisa ser chamado explicitamente, pois há um *garbage collector* que retira os objetos da memória depois de algum tempo sem utilização.

Construtor

- Antes da versão 5 do PHP, por convenção, o construtor era um método que tinha o mesmo nome da classe. A partir da versão 5 podemos definir o método construtor da forma antiga ou usar o método a seguir (atenção para os dois *underscore*):

```
class Conta {  
    private $numero;  
    private $saldo;  
  
    function __construct($n, $s) {  
        $this->numero = $n;  
        $this->saldo = $s;  
    }  
    ...  
}
```

A partir da inclusão do construtor, ao criar uma conta, deve-se passar os valores para o número e para o saldo:

```
$minhaconta = new Conta(1234, 0);
```

Para tornar os valores opcionais, já que não é possível a sobrecarga em PHP, basta definir um valor default:

```
function __construct($n = 0, $s = 0)
```

Destrutor

- O PHP 5 introduziu um conceito de destrutor similar ao outras linguagens orientadas a objeto.
- O método destrutor será chamado assim que todas as referências a um objeto particular forem removidas ou quando o objeto for explicitamente destruído (unset).

```
function __destruct() {  
    echo "destruindo conta ".$this->numero;  
}
```

__toString()

- O PHP possui várias funções com nomes começando com __ , conhecidas como funções “mágicas”. Veja mais em:
http://www.php.net/manual/pt_BR/language.oop5.magic.php
- O método `__toString()` permite que uma classe decida como se comportar quando convertida para uma string.

```
function __toString() {  
    return "Conta: ".$this->numero." - Saldo: ".$this->saldo;  
}
```

- Com isso, o comando `echo $varConta` imprimirá o retorno do método acima.

Built in functions

- Funções para verificação de objetos:

```
$c1 = new Conta(1234, 200);  
if (is_a($c1, "Conta")) {  
    echo "Eu sou uma conta";  
}  
if (property_exists($c1, "saldo")) {  
    echo " que tem um saldo ";  
}  
if (method_exists($c1, "credito")) {  
    echo " e pode receber $$$ ";  
}
```

Encapsulamento

- A visibilidade de um método ou um atributo pode ser definida ao prefixá-los com as palavras-chave:
 - **public**: pode ser acessado de fora da classe, dentro da classe e nas classes derivadas;
 - **protected**: pode ser acessado dentro da própria classe e nas classes derivadas (mas não de fora);
 - **private**: pode ser acessado somente dentro da própria classe.
- Métodos com visibilidade não definida serão públicos.
- Propriedades com visibilidade não definida serão públicas (neste caso devem ser precedidas da palavra **var**).
- Normalmente recomenda-se manter os atributos como privados e os métodos como públicos.
 - Assim, para acessar um atributo privado utiliza-se um método de acesso (get) e para alterá-lo, um método modificador (set).

Encapsulamento

- Métodos de acesso (*getters*)

```
function getNumero() {  
    return $this->numero;  
}
```

- Métodos modificadores (*setters*)

```
function setNumero($numero) {  
    $this->endereco=$numero;  
}
```


Herança

- Uma classe pode ser uma extensão de outra. Isso significa que ela herdará todas as variáveis e funções da outra classe, e ainda terá as que forem adicionadas pelo programador.
- Em PHP não é permitido utilizar herança múltipla, ou seja, uma classe só pode ter uma superclasse.
- Para criar uma classe estendida ou subclasse, deve ser utilizada a palavra reservada `extends`. Nesse exemplo, a classe `ContaEspecial` é derivada da classe `Conta` (superclasse), tendo as mesmas funções e variáveis, com a adição da variável `$limite` e a função `getLimite()`.

```
class ContaEspecial extends Conta
{
    private $limite;

    function getLimite() {
        return $this->limite;
    }
}
```

Herança e Construtores

- O construtor da superclasse é herdado pela subclasse. Porém, se desejarmos definir na subclasse um construtor mais específico, podemos referenciar o construtor da superclasse através da palavra-chave **parent** (pai; similar ao *super* da linguagem Java).

```
class ContaEspecial extends Conta {  
    public $limite;  
  
    function __construct ($numero, $saldo, $limite){  
        parent::__construct($numero, $saldo);  
        $this->limite = $limite;  
    }  
    ...  
}
```

Sobrescrita (*override*)

- A sobrescrita ocorre quando redefinimos, na subclasse, o valor de uma propriedade ou o comportamento de um método herdado da superclasse.

```
<?php
class Forma {
    public $temLados = true;
}

class Quadrado extends Forma {
    public $temLados = 4;
}
```

```
class Veiculo {
    public function buzina() {
        return "BIIIII!";
    }
}

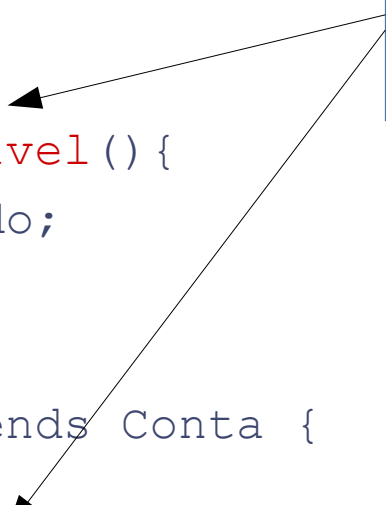
class Bicicleta extends Veiculo {
    public function buzina(){
        return "trim trim!";
    }
}
```

Sobrescrita (*override*)

- O método sobrescrito precisa ter a mesma assinatura (nome e lista de parâmetros).

```
class Conta {  
    private $numero;  
    private $saldo;  
    function saldoDisponivel() {  
        return $this->saldo;  
    }  
}  
  
class ContaEspecial extends Conta {  
    public $limite;  
    function saldoDisponivel() {  
        return $this->getSaldo() + $this->limite;  
    }  
}
```

Polimorfismo: mesmo método com 2 comportamentos, dependendo do tipo do objeto a partir do qual o método for chamado



final

- A palavra-chave final impede que um método ou propriedade seja sobrescrito.

Impede
sobrescrita

```
class Veiculo {  
    final public function buzina() {  
        return "BIIII!";  
    }  
}  
  
class Bicicleta extends Veiculo {  
    public function buzina(){  
        return "trim trim!";  
    }  
}
```

Vai gerar um
erro. (Fatal error:)

Constantes

- Constantes podem ser definidas através da palavra-chave **const**. O nome da constante não possui o símbolo \$.
- Para acessar a constante, utiliza-se a sintaxe **classe::constante**.

Definição de uma
constante

```
class Imortal extends Pessoa {  
    // Imortais não morrem!  
    const vivo = true;  
}  
  
// Se está vivo  
if (Imortal::vivo) {  
    echo "Sempre vivo!";  
}
```

Operador de escopo.
Acessa a constante "vivo" da classe Imortal

static

- Variáveis estáticas são compartilhadas por todas as instâncias (objetos) da classe.
- Tanto variáveis quanto métodos estáticos podem ser utilizados sem instanciar um objeto, utilizando `self::$var` (dentro da classe) ou `Classe::$var` (de fora da classe).

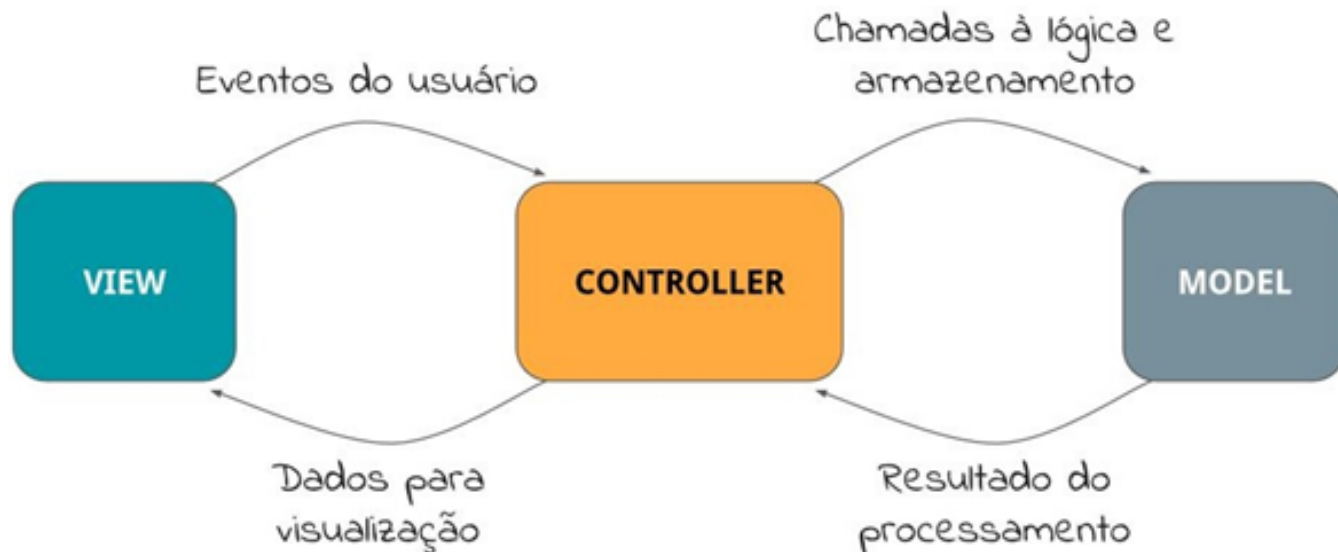
Permite acesso sem
instanciar

```
class Pessoa {  
    public static $vivo = "sim";  
    public static function cumprimento() {  
        echo "Olá!";  
    }  
}  
  
echo Pessoa::$vivo; // imprime "sim"  
Pessoa::cumprimento(); // imprime "Olá!"
```

Objeto não foi instanciado

MVC

- **Model View Controller**
- É um padrão arquitetural (ou *design pattern*) que separa o código em três camadas: o modelo, a visão e o controlador.
- Largamente utilizado e presente em diversos frameworks.
- Possui vantagens como: reutilização de código, facilidade de manutenção, desenvolvimento rápido, etc.



MVC

- Model
 - Local onde fica toda a lógica da aplicação e onde são implementadas as regras de negócio.
 - Responsável pela leitura, escrita e validação dos dados, consultas ao BD, lógica de disparo de e-mail, etc.
 - O *model* sabe como realizar as tarefas, mas não sabe quando isso deve ser feito.
- View
 - Responsável pela exibição dos dados e pela interação com o usuário.
 - Não sabe o que a aplicação está fazendo nem em que momento renderizar as saídas; apenas recebe dados (HTML, XML, JSON, PDF, etc) e os exibe.
 - Também se comunica de volta com o *model* e com o *controller* para reportar o seu estado.
- Controller
 - Determina quando as ações devem acontecer.
 - Interpreta as entradas do usuário e mapeia essas ações em comandos que são enviados para o *model* e/ou para a *view*.