SI401

Programação para a Web

2º Semestre - 2024

Programação *Back-End*: PHP
Unidade 16

Prof. Guilherme Palermo Coelho
guilherme@ft.unicamp.br

Roteiro

- Orientação a Objetos em PHP;
- Referências.

Orientação a Objetos em PHP

Orientação a Objetos em PHP

- A linguagem PHP possui suporte a programação orientada a objetos;
  - A partir da versão 5, PHP passou a utilizar um novo modelo de objetos, o que resultou em ganhos significativos de velocidade e desempenho;
- Neste tópico **não** serão revistos os conceitos fundamentais de orientação a objetos;
  - Serão tratados os mecanismos básicos de PHP para programação dentro deste paradigma;
  - Para informações mais detalhadas, consultar:

http://php.net/manual/en/language.oop5.basic.php

Orientação a Objetos em PHP

- Sintaxe para criação de classes:

```
class nome_classe
{
        // métodos e atributos
}
```

- Métodos e atributos devem ser definidos dentro do *bloco* da classe:
  - Devem ter um dos seguintes modificadores de visibilidade:
    - **public**: membro pode ser acessado de qualquer lugar;
    - **protected***:* acesso dentro da classe onde foi definido e das subclasses (herança);
    - **private***:* acesso apenas dentro da classe onde foi definido.

Orientação a Objetos em PHP

- Criação de um objeto:
  - **$objeto** = new **Nome_da_Classe()**;
    - A partir de PHP7, é obrigatória a definição de um *construtor* para a classe, como será visto adiante;
- Acesso a atributos e métodos:
  - **$objeto**->nome_atributo;
  - **$objeto**->nome_metodo();
- A palavra reservada ***this*** se refere ao próprio objeto dentro da classe.

Orientação a Objetos em PHP

```php
class Estoque
{
        public $itens;

        // Falta o construtor!

        function adiciona($codigo, $quantidade)
        {
                if(isset($this->itens[$codigo]))
                        //isset() verifica se a variável existe;
                        $this->itens[$codigo] += $quantidade;
                else
                        $this->itens[$codigo] = $quantidade;

        }
}
```

Orientação a Objetos em PHP

```
<?php
        …
        $est = new Estoque();

        $est->adiciona("bermuda", 2);
        $est->adiciona("camiseta", 3);

        echo "A loja tem ".$est->itens["bermuda"]. " bermudas.";
        echo "<br />";
        echo "A loja tem ".$est->itens["camiseta"]. " camisetas.";
        echo "<br />";
        …
?>
```

```
A loja tem 2 bermudas.
A loja tem 3 camisetas.
```

PHP O.O.: Construtores e Destrutores

- ***Construtores*** são métodos que são chamados sempre que um novo objeto de uma determinada classe é criado;
  - Em PHP, este método deve ***obrigatoriamente*** ser chamado de "***_ _construct()***";
    - Versões anteriores usavam construtores com o mesmo nome da classe, mas isto é considerado *deprecated* e não deve mais ser usado.
  - Como em outras linguagens orientadas a objetos, construtores também podem receber parâmetros.
    - Valores para os parâmetros são passados após o nome da classe, dentro de parêntesis, na criação do objeto.

PHP O.O.: Construtores e Destrutores

```
class Estoque
{
        public $itens;
        public $nome;

        function __construct($nome)
        {
                $this->nome = $nome;
                echo "Bem vindo à loja $this->nome <br />";
        }
        ...
}
...
$est = new Estoque("Mesbla");
```

```
Bem vindo à loja Mesbla
```

PHP O.O.: Construtores e Destrutores

- ***Destrutores*** são métodos que são chamados após a eliminação da última referência a um objeto, **antes** da liberação da memória;
  - Em PHP, este método deve ***obrigatoriamente*** ser chamado de "***_ _destruct()***";
  - Destrutores podem ser úteis para:
    - Fins de depuração;
    - Fechar conexões com banco de dados;
    - Fechar arquivos;
    - ...

PHP O.O.: Atributos e Métodos Estáticos

- Quando criamos um objeto de uma classe, este objeto passa a ter uma ***cópia própria*** das propriedades da classe (atributos e métodos);
- No entanto, uma classe pode ter também ***propriedades estáticas***:
  - São métodos e atributos que ***pertencem à classe em si***, e não a um dado objeto.
- Propriedades estáticas são definidas em PHP através da palavra-chave ***static***:
  - A referência a um atributo estático deve ser precedida sempre pelo ***nome da classe*** seguido de "**::**";
  - Se um atributo estático estiver sendo acessado de dentro de um método da própria classe, pode-se utilizar "**self::**"

PHP O.O.: Atributos e Métodos Estáticos

```
class Classe {
        public static $var1 = 10;
        public static $var2 = 8;
        public function soma($numero) {
                Classe::$var1 += $numero;
                self::$var2 += $numero;
        }
}
$obj = new Classe();
$obj->soma(2);
$obj2 = new Classe();
$obj2->soma(2);
print Classe::$var1;
print Classe::$var2;
```

```
14
12
```

PHP O.O.: Atributos e Métodos Estáticos

- Além de atributos, também podemos ter ***métodos*** estáticos;
- Estes métodos podem ser utilizados sem necessidade de criação de um objeto;
  - Chamada de um método estático: "**Classe::metodo_estatico()**";

- **ATENÇÃO**: como métodos estáticos pertencem a uma classe e não a um objeto, ***this*** não pode ser utilizado.

PHP O.O.: Atributos e Métodos Estáticos

```
class Printer {
        static function printHelloWorld() {
                print "Hello World!";
                self::printNewLine();
        }

        static function printNewLine() {
                print "\n";
        }
}

Printer::printHelloWorld();
Printer::printHelloWorld();
```

```
Hello World!
Hello World!
```

PHP O.O.: Herança e Métodos Abstratos

- ***Herança*** é um conceito bem estabelecido em O.O. e também está presente em PHP;
  - Quando uma classe estende outra, ela ***herda*** todos os métodos e atributos *public* e *protected*;
    - Os métodos herdados mantêm a **mesma funcionalidade** da classe original, a menos que sejam sobrepostos (*override*) na subclasse.
    - Para que se possa acessar atributos ou métodos (***inclusive o construtor***) da ***classe pai***, a palavra reservada ***parent*** deve ser usada:
      - Ex.: "**parent::imprime()**";
  - **ATENÇÃO**: PHP não permite ***sobrecarga*** de métodos (*overload*), como outras linguagens O.O.

PHP O.O.: Herança e Métodos Abstratos

```
class Classe1 {
          private $var1 = "Olá, var1!\n";
          protected $var2 = "Olá, var2!\n";

          function imprime() {
                    print "Classe 1: " . $this->var1 . "<br />";
                    print "Classe 1: " . $this->var2 . "<br /><br />";
          }
}

class Classe2 extends Classe1 {
          function imprime() {
                    parent::imprime();
                    print "Classe 2: ". $this->var1 ."<br />";
                    print "Classe 2: ". $this->var2 ."<br /><br />";
          }
}

$obj = new Classe1();
$obj->imprime();
$obj = new Classe2();
$obj->imprime();
```

```
Classe 1: Olá, var1!
Classe 1: Olá, var2!

Classe 1: Olá, var1!
Classe 1: Olá, var2!

Classe 2:
Classe 2: Olá, var2!
```

PHP O.O.: Herança e Métodos Abstratos

- PHP também permite a definição de ***métodos abstratos***, através do uso da palavra reservada ***abstract***;
  - Métodos abstratos são apenas declarados, ***cabendo sua implementação a outra classe herdeira*** da classe que originalmente contém o método abstrato;
  - As ***classes*** que contêm ***métodos abstratos*** também devem ser definidas como do tipo ***abstract***:
    - Não podem ser usadas para criar objetos diretamente.

PHP O.O.: Herança e Métodos Abstratos

```
abstract class Abstrata {
        abstract public function teste();
}

class Implementacao extends Abstrata {
        public function teste() {
                echo "Método teste() chamado! <br />";
        }
}

$obj = new Implementacao();
$obj->teste();
```

PHP O.O.: Interfaces

- PHP também tem suporte a ***interfaces***;
  - São estruturas que contêm apenas constantes e ***declarações*** de métodos (sempre devem ser *public*);
  - Devem ser definidas com a palavra reservada ***interface*** (ao invés de ***class***);
  - Uma classe pode *implementar* mais de uma interface:
    - Para isso é usado o operador ***implements***;
    - Ex.:
      - **class** A **implements** B, C // *onde B e C são interfaces*

PHP O.O.: Interfaces

```
interface MinhaInterface {
        public function teste();
}

class Implementacao implements MinhaInterface {
        public function teste() {
                //...
        }
}
```

- **ATENÇÃO**: A menos que seja declarada ***abstrata***, uma classe que implementa uma dada interface deve, ***obrigatoriamente***, implementar **todos** os métodos declarados na interface.

PHP O.O.: A palavra-chave *final*

- A palavra-chave ***final*** pode ser aplicada tanto a ***métodos*** quanto a classes;
  - Quando aplicada a ***métodos***, implica que as subclasses **não poderão sobrescrever estes métodos**;
  - Quando aplicada a uma ***classe***, implica que esta classe ***não pode ter subclasses associadas***.

```
final class Classe {
        // Esta classe não pode ter subclasses
}

class Classe {
        final function teste() {
                // Este método não pode ser sobrescrito
}}
```

Tratamento de Exceções

Tratamento de Exceções

- A partir da versão 5, PHP também passou a oferecer novos mecanismos de ***tratamento de exceções***;
  - Baseados em comandos ***try***, ***catch*** e ***throw***;
  - Muito semelhante ao de outras linguagens de programação OO.
- Como em outras linguagens, trechos de códigos sujeitos a erros específicos são monitorados (em blocos ***try***);
- Exceções que ocorrem nestes blocos são ***lançadas*** (com o comando ***throw***) e então capturadas (método ***catch***) para receberem tratamento adequado.

Tratamento de Exceções

```
// Código que lança uma exceção:
<?php
    function checkNum($number)
        {
                if($number>1)
                {
                        throw new Exception("Valor maior que 1");
                }
                return true;
        }
    //Lançamento explícito de uma exceçao
    checkNum(2);
?>
```

// Mensagem de erro de PHP:

Fatal error: Uncaught exception 'Exception' with message 'Valor maior que 1' in /Applications/MAMP/htdocs/testException.php:14 Stack trace: #0 /Applications/MAMP/htdocs/testException.php(20): checkNum(2) #1 {main} thrown in **/Applications/MAMP/htdocs/testException.php** on line **14**

Tratamento de Exceções

- No exemplo anterior, a mensagem de erro foi ***lançada*** mas não foi devidamente tratada pelo programa;
  - Isso deveria ter sido feito com os comandos ***try ... catch*.**

```
// Código que lança uma exceção:
<?php
    function checkNum($number) {...}

    try // O código dentro deste bloco é monitorado:
    {
        checkNum(2);
    }
    catch(Exception $e) // Tratamento da exceção lançada”
    {
        echo 'Mensagem de erro: ' .$e->getMessage();
    }
?>
```

```
// Mensagem de PHP:
Mensagem de erro: Valor maior que 1
```

Tratamento de Exceções

- PHP permite que se defina uma ***função-padrão*** (criada pelo usuário) a ser chamada toda vez que for lançada uma exceção não capturada;
  - *Top Level Exception Handler*;
  - Sintaxe: ***set_exception_handler***(‘*nome_da_funcao*’);

```
// Lançamento de uma exceção personalizada:
<?php
    function myException($exception)
    {
        echo "<b>Exceção:</b> " . $exception->getMessage();
    }

    set_exception_handler('myException');

    throw new Exception('Exceção não capturada.');
?>
```

Tratamento de Exceções

- Regras gerais para exceções em PHP:
  - Cada bloco ***try*** ou comando ***throw*** deve estar acompanhado de uma bloco ***catch*** correspondente;
  - Múltiplos blocos ***catch*** podem ser usados para capturar exceções de ***classes diferentes***;
  - Exceções podem ser ***relançadas*** de dentro de um bloco ***catch***:
    - Este bloco ***catch*** deve estar dentro de um bloco ***try***.

Referências

Referências

[1] Niederauer, J. “Desenvolvendo Websites com PHP”, 2ª ed. Novatec, 2011.

[2] W3Schools PHP Tutorial:

http://w3schools.com/php/default.asp

[3] Manual do PHP:

http://php.net/manual/pt_BR/index.php

[4] Hudson, P. “Hacking with PHP”:

http://www.hackingwithphp.com/

[5] Prettyman, S. "Learn PHP 7“, Apress, 2016:

https://link.springer.com/book/10.1007/978-1-4842-1730-6