Introdução à Orientação a Objetos em PHP

Veja neste artigo uma introdução à Programação Orientada a Objetos, com apresentação dos conceitos básicos desse paradigma e de exemplos práticos na linguagem PHP.

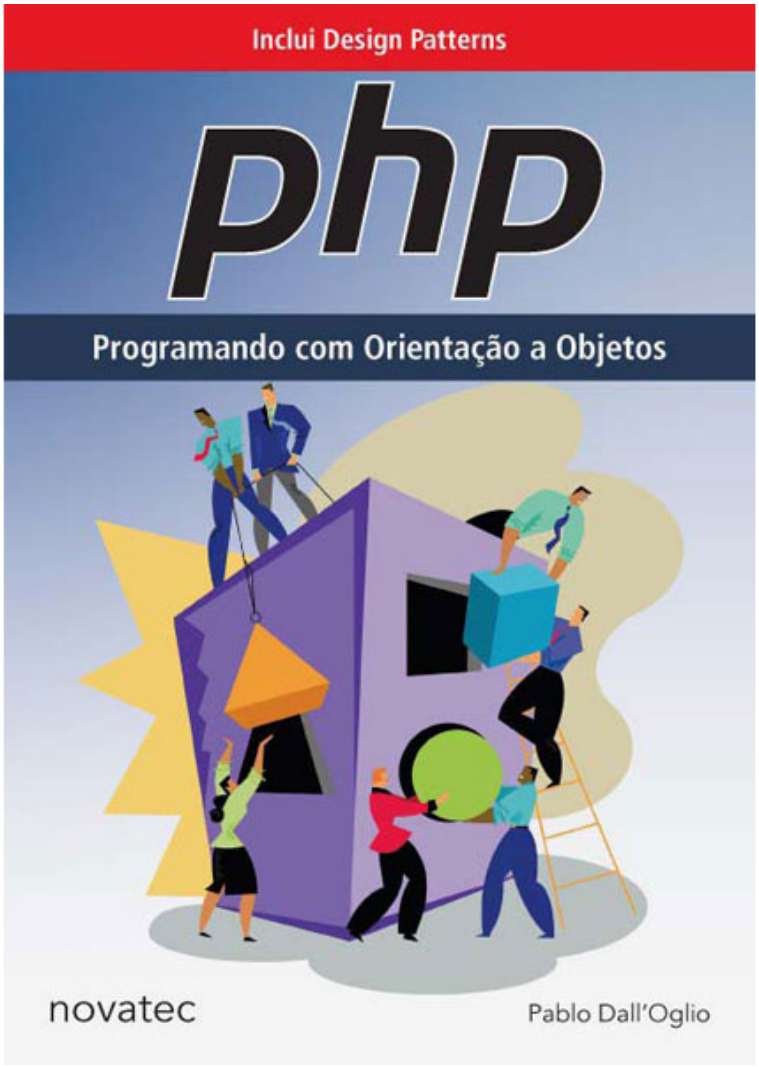

Figura 1: Livro "PHP - Programando com Orientação a Objetos". Ótimo material de estudo

Para iniciar esta introdução, vejamos um pouco de história: Como o PHP não é uma linguagem que foi criada para ser orientada a objetos (só começou a suportar orientação a objetos na versão 3, sendo aprimorada na versão 4, na versão 5.3e o suporte a orientação a objetos está excelente), os programadores PHP utilizavam ou a programação estruturada ou orientação a funções (nomenclatura usada por estudantes para definir um método de desenvolvimento). Este método basicamente organiza as funções mais utilizadas em arquivos específicos, como por exemplo, um arquivo chamado funções de banco e neste arquivo são colocadas as funções de insert, update e delete, depois bastava incluir o arquivo no local onde deseja utilizar as

funções. Para isso utiliza-se os métodos include, include_once, require ou require_once do PHP e chamar as funções.

O include tenta incluir o arquivo, caso não ache, retorna um Warning (warning é apenas um alerta do PHP, a aplicação não é interrompida quando acontece). O require por sua vez retorna um Fatal Error (o fatal error interrompe a aplicação e não executa o resto dos comandos), o include_once e require_once tentam incluir o arquivo, porém se o arquivo já foi incluso ele retorna false e não o inclui novamente.

Saindo um pouco da parte histórica e indo para a parte mais acadêmica, vamos estudar um pouco dos conceitos básicos de orientação a objetos para em seguida fazer alguns trechos de código para fixar melhor os conceitos. O primeiro e mais importante conceito de orientação a objetos é a classe, uma abstração do software de objetos similares, ou seja, um template do qual os objetos serão criados. Crie um arquivo chamado usuário.php para começarmos os exemplos abaixo.

Listagem 1: Primeiro passo criar uma classe em PHP

```
<?php
class Usuario{


}
?>
```

Acima temos um simples exemplo de como criar uma classe. Antes de incrementar o exemplo iniciado acima, abordaremos mais alguns conceitos de orientação a objetos.

Atributos de classe são elementos que definem uma classe, também são conhecidos como variáveis de classe. O exemplo abaixo mostra como criá-los e acessá-los através de funções, também cria o construtor da classe.

Listagem 2: Definição de atributos da classe

```
<?php
class Usuario{
        public $nome;
```

```
        public $cpf;
        public $endereco;
        //construtor da classe
        function Usuario(){
                $this->preparaUsuario();
        }

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

} ?>
```

Acima podemos ver o construtor da classe, que será o método executado assim que a classe for chamada. É neste método que colocamos os passos fundamentais para a inicialização da classe. No exemplo é executada a função preparaUsuário que apenas preenche os atributos da classe, em seguida foram criados alguns métodos para que estes atributos possam ser utilizados quando a classe for instanciada.

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

Listagem 3: Testando a classe criada

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

$usuario = new Usuario();

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

Primeiramente, assim como no Java é preciso fazer o "import" do arquivo que contém a classe, ou seja, especificarmos para o PHP onde se encontram as classes que queremos utilizar e em seguida instanciar a classe. Após isto feito, basta apenas acessar os métodos.

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

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

Para entendermos melhor visibilidade temos que entender melhor 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 públicos e privados, como foi explicado acima. No exemplo abaixo incrementaremos a classe Usuario utilizando visibilidade e herança.

Listagem 4: Classe com visibilidade dos atributos modificada

```
<?php
class Usuario{
        public      $nome;
        protected $cpf;
        private     $endereco;

//construtor da classe
        function Usuario(){
                $this->preparaUsuario();
        }

        private function preparaUsuario(){
                $this->nome = "Octavio";
                $this->cpf = "99999999999";
                $this->endereco = "Rua Fulano de Tal número 0 apt 999";
        }

        public function getCpf (){
```

```
                return $this->cpf;
        }

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

        function getEndereco(){
                return $this->endereco;
        }

} ?>
```

Acima foi escrito um pequeno exemplo de criação de uma classe com seus atributos, getters e setters. A seguir vamos dar uma incrementada no código, mostrando como serão recuperados os dados em outra classe que vai instanciar a classe Usuario. Vejamos o exemplo abaixo.

Listagem 5: Classe que acessa Usuario

```
require_once 'usuario.php';

class AcessaUsuario{
        function imprimeUsuario(){
                $usuario = new Usuario;
                echo $usuario->nome;
                echo $usuario->cpf;
                echo $usuario->endereco;
        }
}
```

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

Listagem 6: Acessando os getters da classe Usuario

```
require_once 'usuario.php';
```

```
class AcessaUsuario{
        function imprimeUsuario(){
                $usuario = new Usuario;
                echo $usuario->nome;
                echo $usuario->getCpf();
                echo $usuario->getEndereco();
        }
}
```

Pronto, este código acima não apresentaria nenhum erro e imprimiria todos os valores dos atributos da classe Usuario.

A seguir veremos um exemplo prático de herança. Como explicado anteriormente herança é a possibilidade de classes e objetos compartilharem funções e atributos. Por exemplo, podemos criar uma classe chamada assistente, que é um tipo de usuário, porém possui algumas particularidades, como um atributo chamado ramal. Vejamos o código.

Listagem 7: Classe Assistente herdando de Usuario

```
require_once 'usuario.php';
class Assistente extends Usuario{
        protected $ramal;

        function Assistente (){
                parent::Usuário();
                $this->ramal = '099';
        }

        public function getRamal(){
                return $this->ramal;
        }

}
```

No construtor da classe executamos o construtor da classe pai, utilizando o parent:: e em seguida preenchemos o atributo ramal. Para recuperarmos os dados da classe Assistente basta fazer o que mostra o exemplo abaixo:

Listagem 8: Utilizando a classe herdada

```
require_once 'assistente.php';
class AcessaAssistente(){
        function imprimeAssistente(){
                $assistente = new Assistente();
                echo $assistente->getRamal();
                echo $assistente->nome;
                echo $assistente->getCpf();
                echo $assistente->getEndereco();
        }
}
```

Bom, esse foi o artigo introdutório de PHP orientado a objetos. Nele vimos construção de classe, herança e visibilidade que são alguns dos princípios fundamentais da orientação a objetos, independente da linguagem. Até a próxima.

Octávio Waldomiro De Almeida