PHP 5 e a utilização de Orientação a Objetos

Daves Santos Vieira[1]

Resumo

As reflexões desse artigo centram-se no uso do desenvolvimento orientado a objetos implementado no PHP 5, mostrando que a linguagem vem se desenvolvendo ao longo dos anos, buscando o seu espaço nas empresas de médio e grande porte. Até a versão 4 do PHP, a programação com o só poderia ser feita de modo estruturado. Esse tipo de programação preconiza a utilização de funções e subrotinas dentro do desenvolvimento e foi muito utilizada por linguagens como o PASCAL e o COBOL. Com a versão 5 e a aplicação do uso da programação orientada a objetos, o PHP começa a ganhar mais importância e grandes empresas como adeptas.
Palavras-chave: PHP. Orientado a objetos. Desenvolvimento.

Abstract

The reflections in this article focus on the use of object-oriented design implemented in PHP 5, showing that the language has evolved over the years, seeking its place in medium and large. Until version 4 of PHP, the programming can only be done in a structured way. This type of programming advocates the use of functions and subroutines in the development and was widely used by languages such as COBOL and PASCAL. With version 5 of the application and use of object-oriented programming, PHP is beginning to gain more importance and large companies as adept.
Keywords: PHP. Object-oriented. Development.

1 Introdução

1.1 O Que é o PHP ?

O PHP foi criado por Rasmus Lerdorf no ano de 1994. A princípio, era formada por um conjunto de scripts voltado ao desenvolvimento de páginas dinâmicas que o autor utilizava para monitorar o acesso ao seu currículo na internet. Com o crescimento das funcionalidades, Rasmus Lerdorf desenvolveu uma implementação em C, facilitando a criação de aplicações para a web. Essa versão foi nomeada de PHP/FI (Personal Home Pages/Forms Interpreter) e o seu código foi disponibilizado no ano de 1995, para compartilhar com outros desenvolvedores e receber ajuda na correção de alguns bugs.

Em 1997, ano em que a segunda versão do PHP foi lançada, aproximadamente 1% da internet (cerca de 50 mil domínios) já utilizavam essa linguagem. Nesse mesmo ano,

[1] Graduado em Sistemas para Internet e Analista de Sistemas da Secretaria de Estado da Justiça e de Defesa ao Consumidor do Estado de Sergipe

dois estudantes que utilizavam PHP em um projeto de comércio eletrônico (Zeev Suraski e Andi Gutmans) resolveram colaborar com Rasmus e criaram a terceira versão do PHP.

Em 1999, desenvolveram o Zend Engine, que é o motor do PHP 4, abandonando o projeto PHP 3, dando um maior poder a máquina da linguagem e um maior número de recursos de orientação a objetos. O maior problema dessa versão foi a criação de cópias de objetos, pois a mesma ainda não trabalhava com apontadores (handlers), como é na linguagem Java.

O problema foi resolvido com a criação do PHP 5, que já trabalha com handlers. Quando ocorre a cópia de um objeto, é feita a cópia de um apontador, pois, em caso de mudanças na versão original do objeto, as outras cópias também sofrerão alterações, o que não acontecia na versão 4.

1.2 Orientação a Objetos

A orientação a objetos é um paradigma que representa toda uma filosofia para construção de sistemas (DALL'OGLIO, 2007).

A programação orientada a objetos (POO), ou ainda em inglês *Object-Oriented Programming (OOP)* é um protótipo de análise, projeto e programação de sistemas baseado na composição e interação entre diversas unidades, denominadas de objetos.

Em linguagens estruturadas, como Cobol, Clipper e Pascal, os sistemas eram formados por um conjunto de procedimentos e variáveis agrupadas de acordo com o contexto. No modelo OO, utilizamos uma visão mais próxima da realidade, permitindo assim uma maior compreensão do que vai ser desenvolvido. Na POO, o programador é responsável por moldar o mundo dos objetos, e explicar para esses objetos como eles devem interagir entre si. Os objetos se comunicam uns com os outros através do envio de mensagens e o principal papel do programador é especificar quais são as mensagens que cada objeto pode receber e qual a ação que ele deve realizar ao receber uma mensagem específica.

2 Características da POO no PHP 5

2.1 Classes

Em POO, uma classe representa um conjunto de objetos com características similares. Em outras palavras, uma classe descreve os serviços providos por seus objetos e quais informações eles podem armazenar.

Para definir o comportamento de um objeto, uma classe utiliza métodos e atributos. Um método é uma subrotina que é executada por um objeto ao receber uma mensagem. Os atributos são os elementos que definem a estrutura de uma classe. Eles também são conhecidos como variáveis da classe. As mensagens enviadas a um objeto podem mudar o valor de um ou mais atributos, alterando o estado de um objeto.

QUADRO 1

Exemplo de Classe

```
class Exemplo_01 {

        var $nome;

        function DigaOla(){
                return "Olá, $this->nome";
        }
}

$ola = new Exemplo_01();
$ola->nome("João");
print $ola->DigaOla();
// Será impresso: Olá, João
```

Fonte: Criação Própria

2.2 Construtores e Destrutores

Segundo PEDROSO (2007), os construtores e destrutores são utilizados para realizar as tarefas de inicialização e destruição dos objetos. No PHP 5, existem nomes pré-definidos para os construtores e destrutores de uma classe. Eles se chamam *__construct* e

__destruct, respectivamente. Observem que os métodos possuem dois underlines (subscritos) antes de cada nome.

Um exemplo disso pode ser visto no quadro abaixo:

QUADRO 2

Exemplo de Construtores e Destrutores

```
class Exemplo_02 {
        var $nome;

        function __construct(){
                echo “Classe construída”;
        }

        function DigaOla(){
                return "Olá, $this->nome";
        }

        function __destruct(){
                echo “Classe destruída”;
        }

}
$ola = new Exemplo_02();
// Será impresso: Classe construída
$ola->nome(“João”);
print $ola->DigaOla();
// Será impresso: Olá, João
unset($ola);
// Será impresso: Classe destruída
```

Fonte: Criação Própria

2.3 Herança

A herança é a criação de subclasses que herdam atributos e operações da super-classe, ou seja, um filho obter algumas características do pai. Um bom exemplo disso é a criação das classes aluno e professor, que possuem atributos semelhantes como nome,

endereço e telefone. Nesse caso, para as semelhanças, será criada uma classe pessoa e as classes professor e aluno irão herdar as características da mesma.

QUADRO 3

Exemplo de Herança

```
class Exemplo_03_pai {
        var $nome;

        function DigaOla(){
                return "Olá, $this->nome";
        }
}

Class Exemplo_03_filho extends Exemplo_03_pai {
        Function DigaOi(){
                return “Oi, parent::$nome”;
        }
}

$ola = new Exemplo_03_filho();
$ola->nome(“João”);
print $ola->DigaOla();
// Será impresso: Olá, João
print $ola->DigaOi();
// Será impresso: Oi, João
```

Fonte: Criação Própria

2.4 Visibilidade

A visibilidade no PHP possui 3 níveis de acesso: *public*, *protected* e *private*.

Todo método ou atributo com a definição *public* pode ser acessado de qualquer lugar da aplicação, ou seja, o mesmo é considerado global. Com a definição *protected*, ele só pode ser acessado dentro da classe que o implementa e dentro das classes que o herdam. Todavia, com a definição *private*, os métodos ou atributos só podem ser acessados dentro da classe que o definem. No caso de uma tentativa de acesso externo ou através de uma classe filha, será exibido um erro.

QUADRO 4

Exemplo de Visibilidade

```
class Exemplo_04
	{
		//Atributos com nível de acesso
		public $publico = "Acesso publico ! <br />";
		protected $protegido = "Acesso protegido ! <br />";
		private $privado = "Acesso privado ! <br />";

	public function __construct()
		{
		}
	}

$classe = new Exemplo_04();

echo $classe ->publico;
//Retorna Acesso publico !

echo $classe ->protegido;
//Retorna Fatal error: Cannot access protected property Access1::$protegido
echo $classe ->privado;
//Retorna Fatal error: Cannot access private property Access1::$privado
```

Fonte: Criação Própria

2.5 Interfaces

O uso de interfaces no PHP permite o desenvolvimento de um código que define quais são os métodos e atributos que uma classe deve implementar, dispensando a obrigação de definir como esses métodos serão tratados. Uma classe não pode implementar mais de uma interface que possuam o mesmo nome, pois isso causaria ambigüidade.

QUADRO 5

Exemplo do uso de Interfaces

```
interface if_05
        {
    public function somar($a, $b)
    public function subtrair($a, $b)
        }

class Exemplo_05 implements if_05
        {

        public function somar($a, $b)
        {
          return ($a + $b);
        }
        public function subtrair($a, $b)
        {
          return ($a - $b);
        }

        }

$classe = new Exemplo_05();

echo $classe->somar(5,3); //Retorna 8
echo $classe->subtrair(5,3); //Retorna 2
```

Fonte: Criação Própria

2.6 Constantes

As constantes são variáveis imutáveis, ou seja, os valores declarados sempre serão os mesmos. As classes e interfaces podem utilizar constantes em seus escopos.

QUADRO 6

Exemplo do uso de Constantes

```
class Exemplo_06
        {
        const pi = 3.1415926536;

        public function ObterCircunferencia($raio)
        {
          return (2 * self::pi * $raio);
        }
        }

$classe = new Exemplo_06();
echo $classe->ObterCircunferencia(10);
//Resultado:  62.831853072
```

Fonte: Criação Própria

No PHP, existem as constantes pré-definidas, que são aquelas que possuem alguns dados do servidor como a versão, o sistema operacional, o arquivo em execução, entre outros. Para ter acesso a todas as constantes pré-definidas, utilize a função *phpinfo()*. Uma tabela será exibida com as constantes e os seus respectivos valores.

2.7 Lambda

As lambdas são conhecidas como funções anônimas. Elas podem ser definidas em qualquer ponto da aplicação e podem ser atribuídas a uma variável. Uma lambda só existe no escopo em que ela for criada. Caso a variável fique fora do escopo, a função não estará disponível.

QUADRO 7

Exemplo de Lambda

```
$lambda = function () { return “Exemplo de Lambda”;};
print $lambda;  // Resultado: Exemplo de Lambda

$lambda_advanced = array_map(function($value) {
            return $value + 2;
           }, array(2, 3, 4, 5, 6));
print_r($lambda_advanced); // Resultado: 4, 5, 6, 7, 8
```

Fonte: Criação Própria

A grande vantagem do uso de lambdas é a de criar funções durante a execução do programa, quando não será preciso qualquer referência a variável.

2.8 Closures

Os lambdas, em teoria, não acrescentam em nada já que a funcionalidade poderia ser reproduzida na versão 4 pela função create_function. Closures trabalham como lambdas de uma forma mais perspicaz, pois elas permitem a interação com variáveis que estão fora do ambiente onde estão definidos.

QUADRO 8

Exemplo de Closures

```
$a = 1;
$closure = function() (use &$a){$a += 1};
$closure();
echo $a ; // 2
$closure;
echo $a; // 3
```

Fonte: Criação Própria

As closures são muito úteis em projetos que utilizam callbacks ou em refatoração de códigos velhos com o intuito de um melhor reaproveitamento do código.

3 Preconceito contra o PHP

Apesar de se ter uma idéia do grande poder que o PHP possui, existe uma pergunta que não quer calar: Por que o mundo do desenvolvimento “não-PHP” tem tanta desconsideração pela comunidade PHP? Irei citar alguns pontos que são de grande relevância para aqueles que criticam a linguagem:

- A mistura entre a POO e o desenvolvimento procedural
- A falta de algumas soluções que outras linguagens já possuíam (closures e namespacing)
- A sintaxe que não é muito agradável (sintaxe feia)
- A grande maioria dos projetos em PHP serem de baixa qualidade

Um dos grandes problemas do PHP para a sua aceitação em um mercado de médio e grande porte foi a cultura geral em volta da linguagem. Os desenvolvedores pareciam ter encorajado práticas ruins, ou seja, as estruturas de código do PHP tendem a ser precárias. Dessa maneira, seria difícil uma grande empresa ter qualquer tipo de interesse em investir milhões em um projeto que possivelmente não teria futuro. Seria como desenvolver uma torre de babel.

O próprio criador do PHP, Rasmus Lerdorf, faz apologia a não-utilização de frameworks e o uso do PHP como uma linguagem de templates. Diante dessa idelologia, existe a vantagem de ser ter um código que traduz velocidade e escalabilidade. Por outro lado, teremos uma pilha de código procedural em projetos de manutenção impossível. Durante os 10 primeiros anos, após o nascimento do PHP, eram assim que os projetos eram desenvolvidos.

Logo após essa fase, surge um novo problema: A linguagem ganha uma série de novos adeptos. Como o PHP permite o desenvolvimento sem uma padronização do código, qualquer pessoa poderia começar a programar para web. Isso acarretou em uma série de projetos desenvolvidos por uma equipe que não possuia o mínimo de conhecimento em boas práticas.

Diante dos fatos anteriormente abordados, podemos dizer que o legado do PHP foi o seu maior problema. A maioria dos desenvolvedores daquela época migrou para outras linguagens como Java, Python, e Ruby e não olhou para trás para ver o desenvolvimento que o PHP teve ao longo dos anos.

Com a criação de frameworks como o Zend e o Code Igniter, o PHP começa a caminhar para a direção certa. Com certeza, a documentação dessas duas ferramentas é de causar inveja a muitas outras. O vasto uso do desenvolvimento nos padrões MVC no mundo PHP é algo atual, e podemos agradecer ao advento do Rails por isso.

Hoje, o PHP tem ao seu favor uma ótima documentação, velocidade e escalabilidade, baixa curva de aprendizado (em relação a outras linguagens) e padrões de projeto, proporcionando ao desenvolvedor o poder criar soluções que atendam a empresas de todos os portes.

4 Conclusão

O PHP, uma das linguagens mais utilizadas na web, tem crescido e amadurecido ao logo desses 16 anos. Muita coisa tem mudado no mundo da programação para a WEB e o PHP tem acompanhado essa evolução, demonstrando o seu potencial e, conseqüentemente, sendo utilizado por grandes empresas como Yahoo!, Oracle, IBM, entre outras. Com a versão 5 do PHP, a programação estruturada abre espaço para a programação orientada a objeto, trazendo mais profissionalismo e qualidade na produção de aplicativos para a web.

Apesar de sofrer com o preconceito pela falta de informação de alguns programadores, o PHP mostra que pode ser bom como qualquer outra linguagem. O que realmente se diferencia nas grandes linguagens de programação são as diretrizes de cada uma. A qualidade da aplicação vai depender da equipe que está desenvolvendo.

4 Referências

DALL'OGLIO, Pablo. **PHP Programando com Orientação a Objetos**: Inclui Design Patterns. 1.ed. São Paulo: Novatec, 2007.

ESCOBAR, Igor. **PHP não é coisa de moleque!**. 2009. Disponível em <http://imasters.uol.com.br/artigo/11171/php/php_nao_e_coisa_de_moleque/>. Acessado em 15 fev. 2009.

KATSGRAU, KENNY. **Why PHP was a ghetto**. Disponível em <http://codefury.net/2011/04/why-php-was-a-ghetto/>. Acessado em 08 mai. 2011.

MELO, Alexandre Altair de; NASCIMENTO, Mauricio G. F. **PHP Profissional**: Aprenda a desenvolver sistemas profissionais orientados a objetos com padrões de projeto. 1.ed. São Paulo: Novatec, 2007.

PEDROSO, Aguinelo. **PHP5 + OOP – Parte 2 (Construtores e Destrutores)**. 2007. Disponível em <http://imasters.uol.com.br/artigo/6329/php/php5__oop_-_parte_02_construtores_e_destrutores/>. Acessado em 17 fev. 2009.

PEDROSO, Aguinelo. **PHP5 + OOP – Resolução de Escopos**. 2007. Disponível em < http://imasters.uol.com.br/artigo/6860/php/php5__oop_-_resolucao_de_escopos/ >. Acessado em 17 fev. 2009.

SICA, Carlos. **PHP Orientado a Objetos – Fale a Linguagem da Internet**. 1.ed. Rio de Janeiro: Ciência Moderna, 2006.

VIANA, Bruno. **Tutorial de POO no PHP**: Herança e Polimorfismo. 2009. Disponível em: <http://www.htmlstaff.org/ver.php?id=23368>. Acessado em 17 fev. 2009.

VIVAS, Mauricio. **Iniciando no PHP:** Constantes no PHP. 2006. Disponível em: <http://www.htmlstaff.org/ver.php?id=1787>, Acessado em 17 fev. 2009.