UM ESTUDO PARA A EVOLUÇÃO DO PHP COM A LINGUAGEM ORIENTADA A OBJETOS

Jean Henrique Zenzeluk*
Sérgio Ribeiro**

Resumo. Este artigo descreve os conceitos de Orientação a Objetos na linguagem de programação PHP . São passados, descritos e explicados os principais conceitos da programação orientada a objetos, junto também de alguns exemplos de implementação e código utilizados. O artigo é escrito de modo explicativo e objetivo, tendo como função passar a compreensão e entendimento da Orientação a Objetos contida no PHP5. Como resultados, pode-se perceber o quanto o PHP evoluiu e está avançado.

Palavras-chave: evolução PHP, Orientação a Objetos, Tecnologia Web, Linguagem de Programação, PHP sintaxe.

*Estudante - Cursando Tec. em Análise e Desenvolvimento de Sistemas - Faculdade Guairacá

**Professor - Graduado em Tecnologia da Informação pela PUC/Campinas, Técnico em Administração de Empresas pela UNOPAR e Especialista em Educação Especial pela Faculdade São Luis. Mestrando em Computação Aplicada pela UEPG.

1. Introdução

A orientação a objetos é uma maneira de programar que trata de modelar os processos de programação de uma maneira próxima à realidade: tratando a cada componente de um programa como um objeto com suas características e funcionalidades.

Com as primeiras 2 versões de PHP, PHP 3 e PHP 4, conseguiram uma plataforma potente e estável para a programação de páginas do lado do servidor. Estas versões serviram muito de ajuda para a comunidade de desenvolvedores, tornando possível que PHP seja a linguagem mais utilizada na web para a realização de páginas avançadas.

Entretanto, ainda existiam pontos negros no desenvolvimento de PHP que trataram de solucionar com a versão 5, aspectos que fizeram falta na versão 4, quase desde o dia de seu lançamento. Referimo- nos principalmente à programação orientada a objetos (POO) que, apesar de estar suportada a partir de PHP3, só implementava uma parte muito pequena das características deste tipo de programação.

O principal objetivo de PHP5 foi melhorar os mecanismos de POO para solucionar as carências das versões anteriores. Um passo necessário para conseguir que PHP seja uma linguagem apta para todo tipo de aplicações e meios, inclusive os mais exigentes.

2. Metodologia

A pesquisa foi desenvolvida a partir de Livros da biblioteca da Faculdade, e também a partir da internet com eBooks e apostilas. Os tópicos da pesquisa foram escolhidos nos livros a partir dos principais tópicos da Orientação a Objetos.

As apostilas reforçaram o entendimento do caso, e os livros tinham o conteúdo totalmente explicado. Como foram mais de um livro, o conteúdo da pesquisa foi feito com a união dos principais segmentos de Orientação a Objetos que cada um listava, buscando a melhor facilidade de compreensão e explicação sobre os temas.

A história envolvida entre a evolução do PHP foi descrita a partir dos principais conteúdos listados em cada livro utilizado, fazendo com que fosse criado uma união entre os conhecimentos a parte que cada livro trouxe.

3. Fundamentação Teórica

3.1 PHP Orientado a Objetos

Um dos problemas mais básicos das versões anteriores de PHP era a clonagem de objetos, que se realizava ao atribuir um objeto a outra variável ou ao passar um objeto por parâmetro em uma função. Para resolver este problema PHP5 usa os manipuladores de objetos (Object handles), que são uma espécie de ponteiros que apontam os espaços de memória onde residem os objetos. Quando se atribui um manipulador de objetos ou

se passa como parâmetro em uma função, se duplica o próprio object handle e não o objeto em si.

Também conhecida pelas iniciais POO , a Programação Orientada a Objetos torna-se progressivamente um modelo seguido pelos desenvolvedores de diversas linguagens. Algumas na verdade exigem que seja feito assim.

Na POO o código é divido em pequenos blocos de procedimentos bastante específicos. Eles são chamados de objetos. O encadeamento das ações disparadas pelos objetos é que faz o programa funcionar.

3.2 Classe

As classes em Programação orientada a objetos (POO) são definições dos elementos que formam um sistema, neste caso, definições dos objetos que vão intervir em nossos programas.

Um objeto se define indicando que propriedades e funcionalidades têm. Justamente essas declarações são o que é uma classe. Quando se faz uma classe simplesmente se especifica que propriedades e funcionalidades têm.

A palavra reservada *class* informa que estamos criando uma Classe. O papel da classe é reunir as características e comportamentos comuns dos objetos.

Na programação utilizamos o mesmo conceito. A classe pode ter diversas funções que pesquisam em múltiplas tabelas, mas em comum dividem o mesmo propósito e pode acontecer também de trabalharem com as mesmas variáveis.

Ao trabalhar com classes e programação orientada a objetos, é comum chamar variável de *atributo*. Se o valor do atributo muda, o resultado final também é alterado. Em outras palavras, as características do objeto instanciado mudam. Sendo assim, ao somar todos os atributos temos o estado do objeto.

As funções, por sua vez, são chamadas de *métodos*. Responsáveis pelo comportamento do objeto, o que ele faz.

Como um exemplo de Classe na realidade, pode-se usar um cachorro que é da Classe animal. Os comportamentos do cachorro, como latir e pular, são os estados, e as características do cachorro, como cor dos olhos, cor do pelo, altura, são os atributos.

Ao instanciar uma classe é criando um objeto. Este objeto tem um estado, referente ao que é atribuído à variável e um comportamento que são as chamadas de funções.

Exemplo de Classe:

```
class Cachorro{              // Classe Cachorro
    var $corDosOlhos;    // Atributo do Cachorro
    var $altura;             // Atributo do Cachorro
```

```
    // latir() é o método, o comportamento do cachorro
        function latir(){
            //aqui o código do método latir
        }
}
```

Agora uma Classe em si não é útil, para isto é preciso instanciá-la. Quando uma Classe é instanciada, é criado um objeto.

Exemplo de Instância:

```
$akita = new Cachorro();          // é instanciado/criado um Akita do tipo Cachoro

$pastorAlemao = new Cachorro();    // é instanciado/criado um Pastor Alemão do tipo Cachoro
```

3.3 Objetos

Classes servem de modelo para a criação dos objetos, é onde é definida toda a estrutura e comportamento de um objeto. Os objetos então são gerados a partir de classes.

A identificação de um objeto ocorre segundo suas características(atributos) e comportamentos(métodos). Na modelagem de sistemas, um objeto é qualquer coisa existente no mundo real. É possível a existência de objetos concretos ou abstratos.

3.4 Atributos

É como são chamadas as características ou propriedades de um objeto. Estes identificam o estado de um objeto e representam uma abstração do tipo de dados do mesmo, e conforme aumenta o número de atributos, mais fácil é de identificar este objeto.

3.5 Constantes Pré-definidas

O PHP possui algumas constantes pré-definidas, indicando a versão do PHP, o Sistema Operacional do servidor, o arquivo em execução, e diversas outras informações. Para ter acesso a todas as constantes pré-definidas, pode-se utilizar a função phpinfo(), que exibe uma tabela contendo todas as constantes pré-definidas, assim como configurações da máquina, sistema operacional, servidor http e versão do PHP instalada.

3.6 Definindo Constantes

Para definir constantes utiliza-se a função *define*. Uma vez definido, o valor de uma constante não poderá mais ser alterado. Uma constante só pode conter valores escalares, ou seja, não pode conter nem um array nem um objeto.

Exemplo:

int define(string nome_da_constante, mixed valor);

3.7 Propriedade estáticas

Propriedades estáticas são atributos de uma classe; são dinâmicas como as propriedades de um objeto, mas estão relacionadas à classe. Como a classe é a estrutura comum a todos os objetos dela derivados, propriedades estáticas são compartilhadas entre todos os objetos de uma mesma classe.

3.8 Métodos estáticos

Métodos estáticos podem ser invocados diretamente da classe, sem a necessidade de instanciar um objeto para isso. Eles não devem referenciar propriedade internas pelo operador $this, porque este operador é utilizado para referenciar instâncias da classe (objetos), mas não a própria classe; são limitados a chamarem outros métodos estáticos da classe ou utilizar apenas propriedade estáticas. Para executar um método estático, basta utilizar a sintaxe *NomeDaClasse::NomeDoMétodo()*.

3.9 Construtores

Os construtores se encarregam de resumir as ações de iniciação dos objetos. Quando instanciamos um objeto, temos que realizar vários passos em sua iniciação, por exemplo, dar valores a seus atributos e isso é o que se encarrega o construtor. Os construtores podem receber dados para iniciar os objetos como se deseje em cada caso.

A sintaxe para a criação de construtor varia em relação a do PHP 3 e 4, pois deve se chamar com um nome fixo: __construct. (São dois hífens baixos(underline), antes da palavra "construct").Exemplo de Construtor:

```
class Cachorro{
        var $corDosOlhos;
        var $peso;

        function __construct($corDosOlhos, $peso){
            $this->corDosOlhos = $corDosOlhos;
            $this->peso  = $peso;
        }
}
```

Quando uma Classe for instanciada, já deverá ser passado os valores para o construtor.

Exemplo:

```
$akita = new Cachorro("azuis", "5 kg");
```

3.10 This

Na definição de uma classe, pode-se utilizar a variável *$this*, que é o próprio objeto. Assim, quando uma classe é instanciada em um objeto, e uma função desse objeto na definição da classe utiliza a variável *$this*, essa variável significa o objeto que estamos utilizando.

3.11 Destrutor

Os destrutores são funções que se encarregam de realizar as tarefas que se necessita executar quando um objeto deixa de existir. Quando um objeto já não está referenciado por nenhuma variável, deixa de ter sentido que esteja armazenado na memória, portanto, o objeto deve ser destruído para liberar seu espaço. No momento de sua destruição, a função se chama destrutor, que pode realizar as tarefas que o programador estime oportuno realizar.

A criação do destrutor é opcional. Somente devemos criá-lo, se desejarmos fazer alguma coisa quando um objeto se elimine da memória.

O destrutor é como qualquer outro método da classe, embora deve se declarar com um nome fixo: __destruct.

3.12 Modificadores de acesso a métodos e propriedades

São os Public, Protected e Private, que se podem conhecer porque já se utilizam em outras linguagens orientados a objetos.

Um dos princípios da programação orientada a objetos é o encapsulamento, que é um processo no qual se ocultam as características internas de um objeto àqueles elementos que não têm porque conhecê-las. Os modificadores de acesso servem para indicar as permissões que terão outros objetos para acessar a seus métodos e propriedades.

3.13 Modificador public

É o nível de acesso mais permissivo. Serve para indicar que o método ou atributo da classe é público.

Neste caso pode-se acessar a este atributo, para visualizá-lo ou editá-lo, por qualquer outro elemento do programa. É o modificador que se aplica se não se indica outra coisa.

Vejamos um exemplo de classe onde declaramos como public seus elementos, um método e uma propriedade. Trata-se da classe "dado", que tem um atributo com sua pontuação e um método para tirar o dado o obter uma nova pontuação aleatória.

3.14 Modificador private

É o nível de acesso mais restritivo. Serve para indicar que essa variável somente vai poder ser acessada pelo próprio objeto, nunca de fora. Se tentarmos acessar um método

ou atributo declarado private de fora do próprio objeto, obteremos uma mensagem de erro indicando que não é possível a este elemento.

3.15 Modificador protected

Este indica nível de acesso médio e um pouco mais especial que os anteriores. Serve para que o método ou o atributo seja público dentro do código da própria classe e de qualquer classe que herde daquela onde está o método ou propriedade protected. É privado e não acessível de qualquer outra parte.

Ou seja, um elemento protected é público dentro da própria classe e em suas heranças.

3.16 Herança

Quando falamos em herança, a primeira imagem que nos aparece na memória é a de uma árvore genealógica com avós, pais, filhos e nas características que são transmitidas geração após geração.

O que deve ser levado em consideração dobre herança em orientação a objetos é o compartilhamento de atributos e comportamentos entre as classes de uma mesma hierarquia(árvore). As classes inferiores da hierarquia automaticamente herdam todas as propriedade e os métodos das classes superiores, chamadas de superclasses.

Então Basicamente a herança é um processo pelo qual os objetos podem herdar as características de outros, de modo que se podem fazer objetos especializados, baseados em outros mais gerais.

A herança é um dos mecanismos fundamentais da programação orientada a objetos. Por meio da herança, podem se definir classes a partir da declaração de outras classes. As classes que herdam incluem os métodos como as propriedades da classe a partir da qual estão definidos.

Como um exemplo pode-se usar animais. Pensando que existe a classe cachorro, e a classe animal, cachorro é um animal, então este pode herdar da classe animal.

O normal em sistemas de herança é que as classes que herdam de outras incluam novas características e funcionalidades, à parte dos atributos e métodos herdados. Porém, isto não é imprescindível, de modo que podem se criar objetos que herdem de outros e não incluam nada novo.

Em PHP, para herdarmos umas classe deve ser usada a palavra reservada *extends*.

Exemplo:

```
class Cachorro extends Animal{}
```

3.17 Polimorfismo

Polimorfismo é um termo grego que significa muitas formas (poli: muitas, morphos: formas). Na programação é o mesmo que dizer que várias classes podem possuir a mesma estrutura e comportamentos diferentes.

Ao utilizar herança não apenas podemos reutilizar métodos da classe pai, como também podemos sobrescrever os métodos da classe pai, fazendo assim com que algumas características sejam modificadas.

Basicamente, por exemplo podem ser criados métodos com o mesmo nome, sendo que só é preciso ter a assinatura/conteúdo diferentes.

3.18 Abstração

No paradigma de orientação a objetos se prega o conceito da "abstração". Para construir um sistema orientado a objetos, não devemos projetar o sistema como sendo uma grande pela monolítica; devemos separá-lo em partes, concentrando-nos nas peças mais importantes e ignorando os detalhes, para podermos construir peças bem-definidas que possam ser reaproveitadas mais tarde, formando uma estrutura hierárquica.

3.19 Classes Abstratas

Classes abstratas basicamente são classes que não podem ser instanciadas, ou seja, não podem ser criados objetos a partir desta.

3.20 Classes Finais

A classe final não pode ser uma superclasse, ou seja, não pode ser base em uma estrutura de herança. Se definirmos uma classe como final pelo operador *final*, ela não poderá ser especializada.

Exemplo:

final class Cachorro extends Animal()

3.21 Métodos Abstratos

Um método abstrato consiste na definição de uma assinatura na classe abstrata. Este método deverá conter uma implementação na classe-filha, mas não deve possuir implementação na classe em que ele é definido.

3.22 Métodos Finais

Um método final não pode ser sobrescrito, ou seja, não pode ser redefinido na classe-filha. Para marcar um método como final, basta utilizar o operador *final* no início de sua declaração.

3.23 Associação

Associação é a relação mais comum entre dois objetos, de modo que um possui uma referência à posição da memória onde o outro se encontra, podendo visualizar seus atributos ou mesmo acionar uma de suas funcionalidade (métodos). A forma mais comum de implementar uma associação é ter um objeto como atributo de outro.

3.24 Agregação

Agregação é o tipo de relação entre objetivos conhecidos como todo/parte. Na agregação, um objetivo agrega outro objetivo, ou seja, torna um objetivo externo parte de si mesmo pela utilização de um dos seus métodos. Assim, o objeto-pai poderá utilizar funcionalidades do objeto agregado. Nesta relação, um objeto poderá agregar uma ou muitas instâncias de um outro objeto. Para agregar muitas instâncias, a forma mais simples é utilizando arrays. Criamos um array como atributo da classe, sendo que o papel deste array é armazenar iúmeras instâncias de uma outra classe.

3.25 Composição

Composição também é uma relação que demonstra uma relação todo/parte. A diferença em relação à agregação é que, na composição, o objeto-pai ou "todo" é responsável pela criação e destruição de suas partes. O objeto-pai realmente "possui" a(s) instância(s) de suas partes. Diferentemente da agregação, na qual as instâncias do "todo" e das " partes" são independentes.

Na agregação, ao destruirmos o objeto "todo", as "partes" permanecem na memória por terem sido criadas fora do escopo da classe "todo". Já na composição, qiando o objeto "todo" é destruído, suas "partes" também são, justamente por terem sido criadas pelo objeto "todo".

3.26 Intercepções

O PHP5 introduziu o conceito de intercepção em operações realizadas por objetos por meio dos métodos __set(), __get() e __call(), vistos a seguir.

3.27 Método __set()

O método *__set()* intercepta a atribuição de valores a propriedades do objeto. Sempre que for atribuído um valor a uma propriedade do objeto, automaticamente esta atribuição passa pelo método __set(), o qual recebe o nome da propriedade e o valor a ser atribuído, podendo atribuí-lo ou não.

3.28 Método __get()

O método *__get()* intercepta requizições de propriedades do objeto. Sempre que for requisitada uma propriedade, automaticamente essa requisição passará pelo método __get(), o quel recebe o nome da propriedade requisitada, podendo retorná-la ou não.

3.29 Método __call()

O método *__call()* intercepta a chamada a métodos. Sempre que for executado um método que não existir no objeto, automaticamente a execução será direcionada para ele, que recebe dois parâmetros, o nome do método requisitado e o parâmetro recebido, podendo decidir o procedimento a realizar.

4. Resultados

Com esta pesquisa, o conceito de orientação a objetos em PHP foi altamente reforçado, e as dúvidas e compreensão dos tópicos puderam ser de fácil entendimento. Como os tópicos foram descritos a partir de mais de um livro foram feitas uniões entre os conteúdos, podendo-se ter até mesmo uma opinião própria, gerada com a compreensão e explicação únicos de cada livro.

Durante a realização e desenvolvimento da pesquisa até mesmo outros tópicos, que não são os de orientação a objetos em PHP, puderam ser aprendidos e compreendidos. Isto porque a busca de informação faz com que se adentre sobre os vários conteúdos (que não são de seu conhecimento) contidos nos livros, criando uma certa curiosidade sobre outros assuntos, e então fazendo com que hajam mais buscas, aumentando a o conhecimento de quem as faz.

5. Conclusão

Ao final desta pesquisa pôde-se concluir que o PHP evoluiu bastante desde a versão PHP3 ate chegar na versão do PHP5, e o mesmo junto da orientação a objetos também cresceu bastante.

Com o PHP5 foram melhoras os mecanismos de PHP orientado a objetos, fazendo com que muitas das faltas contidas em versões anteriores pudessem ser preenchidas, deixando a linguagem cada vez mais apta e completa.

6. Referências

BARRETO< Mauricio Bas de Souza. Curso de Linguagem PHP. Abril de 2000, 67 páginas.

DALL' OGLIO, Pablo. PHP Programando com Orientação a Objetos, São Paulo, Novatec, 2007, 580 páginas.

ESPINHA, Diogo. Como funciona o PHP. Disponível em <http://www.escolacriatividade.com/como-funciona-o-php/>. Acesso em 14 de agosto 2013.

SICA, Carlos. PHP Orientado a Objetos, Rio de Janeiro, Ciência Moderna, 2006, 216 páginas.

XAVIER, Fabricio S. V. PHP Do Básico a Orientação a Objetos, Rio de Janeiro, Ciência Moderna Edit, 2008, 244 páginas.