**PHP7 e Orientação a Objetos**

**Considerações preliminares:**

Paradigma – padrão, conduta, uma forma de se fazer alguma coisa;

No PHP podem ser desenvolvidos códigos baseados em um paradigma procedural e um paradigma de orientação a objetos dentro do mesmo script;

Paradigmas esses que são os mais difundidos no mercado.

Podemos distinguir qual é o paradigma empregado nos scripts analisando os princípios aplicados ao código; princípios de cada um:

**-Procedural/Estruturado/Imperativo**

Sequência – interpretação gravitacional e chamada de funções;

Decisão – aplicação de comandos como ifelse ou switch;

Repetição - aplicação de comandos como for, while e do while.

(Funcional e lógico)

**-Orientado a objetos**

Princípios serão falados ao longo da seção;

Princípios são atendidos pelo PHP.

(Concorrente e reativo)

É fato que o PHP suporta o paradigma procedural e de orientação a objetos, pois ele possui os recursos necessários para atender aos princípios dos 2.

**Introdução a orientação a objetos**

***Paradigma Procedural***

Chamada de procedimentos para manipulação de dados (métodos ou funções).

***Paradigma de Orientação a Objetos***

Estrutura de dados com comportamentos (métodos ou funções).

**Princípios -> Pilares:**

**Abstração**: forma como interpretamos as coisas do mundo real e transportamos para dentro da aplicação.

– **Entidade:** compreensão de que um objeto do mundo real deve passar para dentro da aplicação, identificada a partir de um processo de abstração (observação), não é o objeto em si; **Identidade:** Estabelecer uma referência que futuramente nos permitir acessar o objeto; **Características:** atributos que especificam as características da entidade (ex.:tipo, categoria, título, nome...), o nível dos atributos dependem da regra de negócio da aplicação (pode-se ter + ou - atributos de acordo com as necessidades de negócio); **Ações:** métodos/comportamentos da entidade, ações do objeto; se esse objeto fosse um produto (exibirResumoProduto, alterarValorProduto).

//acessando atributos ou métodos (->):

$y->resumirCadFunc ();

**$this-><atributo que quero receber o valor>** é utilizado para recuperar atributo do objeto.

**public $nome = 'José';**

**public $numFilhos = 2;**

**return "$this->nome possui $this->numFilhos filho(s)";**

**//atribuindo ao atributo numFilhos do objeto o valor de $numFilhos que é o parâmetro recebido pelo método(modificarNumFilhos)**

**$this->numFilhos = $numFilhos;**

**class Funcionario**

**{**

**//class - é seguido um padrão onde a primeira letra da palavra é maiúscula, isso segue não importa quantas palavras houver**

**//atributos**

**//primeira palavra sempre minúscula, mas se houver mais de uma, a/s outra/s começam com letra maiúscula**

**public $nome = 'José';**

**public $telefone = '11 99999-8888';**

**public $numFilhos = 2;**

**//métodos - na prática são funções**

**//padrão de definição de nomes segue mesma regra da definição de variáveis-atributos**

**function resumirCadFunc()**

**{**

**}**

**function modificarNumFilhos($numFilhos)**

**{**

**//afetar um atributo do objeto**

**$this->numFilhos = $numFilhos;**

**//atribuindo ao atributo numFilhos do objeto o valor de $numFilhos que é o parâmetro recebido pelo método(modificarNumFilhos)**

**}**

**//variável recebe instância da classe Funcionário**

**$y = new Funcionario();**

**//criando objeto com base no modelo Funcionário e atribuindo ele a variável $y**

**//a partir dessa variável estamos recebendo os atributos e métodos do objeto**

**//acessando atributos ou métodos(->):**

**echo $y->resumirCadFunc(); //exibindo resumo do cadastro do funcionario**

**echo '<br/>';**

**$y->modificarNumFilhos(3); //modificando num de filhos**

**echo $y->resumirCadFunc(); //exibindo  cadastro novamente (com informação de numFilhos atualizada)**

**?>**

**//nova instância de objeto / novo funcionário**

**$x = new Funcionario();**

**echo $x->resumirCadFunc();**

**echo '<br/>';**

**echo $x->modificarNumFilhos(1);**

**echo $x->resumirCadFunc();**

dados distintos dentro da aplicação que podem ser evoluídos de forma isolada.

**É ideal que as classes sejam genéricas, não tenham atributos engessados (atributos com valor fixo)**;

**A ideia seria evoluir os atributos no momento de criação do objeto ou após a criação desse objeto**.

No exemplo desse projeto:

**$nome = null;**

**$telefone = null;**

**$numFilhos = null;**

**Métodos Getters e Setters**

Nos permitem atender a uma convenção que diz qual a melhor forma de acessar e manipular os atributos dos objetos;

Não é uma regra a manipulação de atributos com getter e setter, é opcional;

**Setter** – setar ou **definir** os **valores** de um atributo de um objeto. Recebe e manipula um valor, **sem retorno**. ***Estrutura:***

Palavra reservada **function**;

Nome do método: set<NomeDoMétodo> - sendo que o nome do método **inicia sempre com letra maiúscula** (por sempre começar na segunda palavra);

Abre e fecha parênteses;

Escopo do método definido dentro de chaves.

function setNome(){

}

$this->nome = $nome;

    } //para manipular um atributo do próprio objeto basta utilizar a palavra reservada this e atribui-se a ela o valor passado por parâmetro na função

A palavra this indica que estamos utilizando uma palavra reservada do próprio objeto;

**Getter** – Nos permitem recuperar valores dos atributos;

Mesma **sintaxe do set**, só muda que **ao invés de set é get**;

A palavra utilizada para fazer o retorno da variável é ‘**return’**.

function setNumFilhos($numFilhos)

    {

        $this->numFilhos = $numFilhos;

    }

    function getNome()

    {

        return $this->nome;

    }

$y->setNumFilhos(2);

echo $y->getNome() . ' possui ' . $y->getNumFilhos() . ' filho(s) ';

**Getters e Setters mágicos (overloading de atributos e métodos)**

Nos exemplos dado, na classe funcionário haviam apenas 3 atributos e foram implementados get e set apenas para $nome e $numFilhos. Implementação simples;

Se houvesse uma classe com diversos atributos e para cada um tivéssemos que implementar os métodos getter e setter, fazendo com que fique extenso e a codificação fique redundante;

Pensando nisso, as pessoas começaram a utilizar os métodos getters e setters de uma forma mais inteligente. Utilizando o conceito de overloading ou sobrecarga;

A ideia desse conceito é criar um único método get e um único método set capazes de se adaptar ao atributo que será manipulado.

**Nome de método deve ser essencial para seu fim.** Pois sempre que ele for utilizado dentro da aplicação ou estiver fazendo a manutenção em outras ações, provavelmente o método responsável por fazer o set dos atributos do objeto, terá esse nome.

function \_\_set($atributo, $valor)

    {

    $this->$atributo = $valor;

    }

$y = new Funcionario();

$y->\_\_set('nome', 'Jose');

$y->\_\_set('numFilhos', 2);

echo $y->resumirCadFunc();

echo '<br/>';

echo $y->\_\_get('nome') . ' possui ' . $y->\_\_get('numFilhos') . ' filho(s) ';

**Chamar métodos internamente**

        return $this->\_\_get('nome'). "possui". $this->\_\_get('numFilhos'). "filho(s)";

**Exibindo método (passando variáveis)**

$y = new Funcionario();

$y->\_\_set('nome', 'Jose');

$y->\_\_set('numFilhos', 2);

echo $y->resumirCadFunc();

Encapsulamento

Herança

Polimorfismo