**Magic methods**

**Magic methods do PHP: Construtores e destrutores**

Sempre que instânciamos uma novo objeto, utilizando a palavra reservada new, o PHP executa um método especial chamado \_construct em sua classe. Você pode definir esse método para executar alguma ação toda vez que um objeto daquele tipo for criado. Um exemplo seria:

class Produto

{

private $id;

// outras propriedades

function \_\_construct()

{

echo "construindo um Produto";

}

// outros métodos

}

Agora que temos esse método definido, toda vez que alguém criar um novo Produto, a mensagem ::construindo um Produto:: será exibida. É como um evento executado sempre que uma classe desse tipo é instânciada.

Outro ponto interessante é que o método \_construct pode receber parâmetros, sendo assim podemos inicializar nossas propriedades ou qualquer outra informação que seja necessária. Por exemplo, vamos criar um construtor que já receba o nome e preço do Produto:

class Produto

{

private $nome;

private $preco;

// outras propriedades

function \_\_construct($nome, $preco)

{

$this->setNome($nome);

$this->setPreco($preco);

}

// outros métodos

}

Note que agora, ao tentar criar um Produto sem passar nome ou preco como parâmetro, receberemos o seguinte ::warning:::

Warning: Missing argument 1 for Produto::\_\_construct(), called in

/Users/turini/Desktop/cursophp/loja/index.php on line 9 and defined in

/Users/turini/Desktop/cursophp/loja/produto.php on line 12

Warning: Missing argument 2 for Produto::\_\_construct(), called in

/Users/turini/Desktop/cursophp/loja/index.php on line 9 and defined in

/Users/turini/Desktop/cursophp/loja/produto.php on line 12

Para resolver isso, tudo que precisamos fazer é mudar a forma que estamos criando o produto. No lugar de fazer:

$produto = new Produto();

$produto->setNome("Livro da Casa do Código");

$produto->setPreco(59.90);

Passaremos a fazer:

$produto = new Produto("Livro da Casa do Código", 59.90);

**Construtores à moda antiga**

Nas versões anteriores ao PHP 5, você declarava seus construtores criando uma função com o mesmo nome da classe. Ou seja, o construtor da classe Produto ficaría assim:

function Produto($nome, $preco)

{

$this->setNome($nome);

$this->setPreco($preco);

}

Para manter a compatibilidade, a partir do PHP 5 ele procura o método \_ \_contruct e, apenas caso não encontre, procura uma função com o nome da classe.

Normalmente utilizamos o construtor quando queremos forçar a passagem de um valor obrigatório. Se todoProduto obrigatóriamente precisa pelo menos de um nome e preco para existir, como garantir que essas informações serão passadas? Podemos criar um método \_construct que receba esses dois parâmetros.

Outro uso comum dos construtores, é a inicialização de parâmetros que já precisam ter algum valor inicial após a criação do objeto. A propriedade $categoria por exemplo, é uma referência a nossa classe Categoria. Porém o PHP não sabe disso, até que tenhamos uma instância de $produto para nele colocar uma nova categoria$produto->categoria = new Categoria();.

Como o \_construct pode nos ajudar a ter um objeto Categoria já pronto no momento de instânciar um Produto? Criando uma instância de Categoria no \_construct do Produto. Nosso código ficará assim:

class Produto

{

private $nome;

private $preco;

private $categoria;

// outras propriedades

function \_\_construct($nome, $preco)

{

$this->setNome($nome);

$this->setPreco($preco);

$this->setCategoria( new Categoria() );

}

// outros métodos

}

Dessa forma, evitamos acessar um atributo $produto->getCategoria()->getId() e correr o risco dogetCategoria() estar null ou recebermos um erro como o abaixo:

PHP Fatal error: Call to a member function getId() on a non-object

Uma vez que estaremos garantindo que o atributo $categoria sempre terá uma instância e também que é um objeto e possui o método getId().

**Para saber mais: \_\_destruct**

Outro recurso interessante do PHP são os métodos destrutores. Você pode criar um método chamado \_destructem sua classe, que será chamado assim que todas as referências para esse objeto específico forem removidas, ou ainda caso ele tenha sido destruído explicitamente.

Um exemplo seria:

class Produto

{

private $id;

// outras propriedades

function \_\_construct($nome, $preco)

{

$this->setNome($nome);

$this->setPreco($preco);

$this->setCategoria( new Categoria() );

}

function \_\_destruct()

{

echo "Destruindo o produto ".$this->getNome();

}

// outros métodos

Após utilizar o produto que criamos, a seguinte mensagem será exibida:

Destruindo o produto Livro da Casa do Código

**Representando o objeto como um texto**

Outra necessidade comum do dia a dia é representar nossos objetos como um texto. Sabemos que para mostrar o nome do Produto, por exemplo, podemos fazer:

echo $produto->getNome();

Mas o que acontece se eu mandar imprimir a variável produto inteira?

echo $produto;

Bem, nesse caso a saída não será muito agradável. Receberemos o **fatal error** a seguir:

Catchable fatal error: Object of class Produto could not be converted to string

in /Users/turini/Desktop/cursophp/loja/index.php on line 24

O Produto não pode ser convertido como um texto. Claro, podemos usar o var\_dump para ter um resultado mais perto do que esperamos. Seria algo como:

var\_dump($produto);

Neste caso a saída (formatada, para melhor legibilidade) será:

object(Produto)#1 (6) {

["id":"Produto":private]=> NULL

["nome":"Produto":private]=> string(24) "Livro da Casa do Código"

["preco":"Produto":private]=> float(59.9)

["descricao":"Produto":private]=> NULL

["categoria\_id":"Produto":private]=> int(1)

["usado":"Produto":private]=> bool(false)

}

Já é bem melhor, mas queremos algo mais simples. Quando imprimir um produto seria interessante mostrarmos apenas seu nome e preco, como a seguir:

Nome: Livro da Casa do Código, Preço: 59.90

Podemos utilizar outro método bastante comum para nos ajudar nesta tarefa, o \_toString. Ele ensina ao PHP como se comportar caso algum objeto desse tipo seja convertido para uma string. Neste caso faremos:

function \_\_toString()

{

return "Nome: ".$this->getNome().", Preço: ".$this.getPreco();

}

Tudo pronto, qual a saída ao executar a instrução echo $produto?

**Para saber mais: Magic methods**

Como vimos, os métodos \_construct, \_destruct e \_toString tem um signifcado especial no PHP. Eles são conhecidos como ::Magic Methods::. Se quiser, você pode ler mais a respeito e conhecer outros métodos mágicos pela documentação da linguagem, em:

<http://php.net/manual/pt_BR/language.oop5.magic.php>