POO
Programação para a WEB - servidor (server-side)

Sérgio da Silva Nogueira


php

CLASSE

Uma classe é uma abstração ou uma estrutura que define um tipo de dados.

Cada classe pode conter atributos(variáveis) e também métodos(funções) para manipular esses atributos.

ESTRUTURA DE UMA CLASSE

```php
<?php

class MyClass
{
    // Class properties and methods go here
}

?>
```

OBJETO

Um objeto pode ser qualquer entidade física, como um computador ou um funcionário, ou um entidade, como um ficheiro.

O objeto é caracterizado pelas suas características(atributos) e pelo seus comportamentos(métodos).

INSTANCIAR UMA CLASSE

No contexto de prorgamação orientada a objetos, um objeto é uma instância de uma classe. Cada instância possui uma identidade única e um estado que represanto pelos valores atuais dos atributos do objeto.

O objeto é criado através da seguinte forma:

```php
<?php

$obj = new MyClass;

?>
```


PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

VER CONTEÚDO DE UMA CLASSE

```php
<?php

var_dump($obj);

?>
```

ATRIBUTOS

Um atributo é qualquer propriedade, característica ou qualidade que possa ser atribuída ao objeto.

Exemplos

- A idade de uma pessoa.
- O número de portas de um carro.

ATRIBUTOS

```php
<?php

// Class name and definition
class MyClass
{
    // Class Properties
    var $name = "I'm a class property!";

    // Class Constructor
    public function __construct() {
        $this->name = "Property";
    }
}

$obj = new MyClass;

echo $obj->name;

?>
```


PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

ATRIBUTOS

No PHP Orientado a Objetos, todos os atributos de uma classes devem ser manipulados pela variável *$this*.

```
$this->_name = ”Sérgio";
```

VISIBILIDADE

- A visibilidade é utilizada para indicar o nível de acessibilidade de um determinado atributo ou método.

PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

TIPOS DE VISIBILIDADE PRIVATE

- Somente os objetos da classe detentora do atributo ou método poderão ver ou aceder.

```php
<?php

// Class name and definition
class MyClass
{
    // Class Properties
    private $_name = "I'm a class property!";

    private function getName(){
        //...
    }
}

?>
```


PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

TIPOS DE VISIBILIDADE PROTECTED

- Determina que além dos objetos da classe detentora dos atributos ou métodos também os objetos de suas subclasses poderão ter acesso ao mesmo

```php
<?php

// Class name and definition
class MyClass
{
    // Class Properties
    protected $_name = "I'm a class property!";

    protected function getName(){
        //...
    }
}

?>
```

TIPOS DE VISIBILIDADE PUBLIC

- A visibilidade pública determina que o atributo ou método pode ser utilizado por qualquer objeto.

```php
<?php

// Class name and definition
class MyClass
{
    // Class Properties
    public $_name = "I'm a class property!";

    public function getName(){
        //...
    }
}

?>
```

ENCAPSULAMENTO

O *encapsulamento* é uma das principais técnicas que define a programação orientada a objetos. É um dos elementos que adicionam segurança à aplicação em uma programação orientada a objetos pelo facto de esconder as propriedades, criando uma espécie de caixa negra.

A maior parte das linguagens orientadas a objetos implementam o encapsulamento baseado em propriedades privadas, ligadas a métodos especiais chamados *getters* e *setters*, que irão retornar e definir o valor da propriedade, respectivamente. Esta definição evita o acesso direto a propriedades do objeto, adicionando uma outra camada de segurança à aplicação.

PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

ENCAPSULAMENTO

Para fazermos um paralelo com o que vemos no mundo real, temos o encapsulamento em outros elementos. Por exemplo, quando clicamos no botão ligar da televisão, não sabemos o que está a acontecer internamente.

Podemos então dizer que os métodos que ligam a televisão estão encapsulados.

ENCAPSULAMENTO

Os atributos da classe nunca devem ser acedidos diretamente de fora do scope de uma classe Para aceder aos atributos privados ou protegidos é preciso criar os métodos

- *Get* - Método para obter valores dos atributos
- *Set* - Método para atribuir valores aos atributos

MÉTODOS

```php
// Class name and definition
class MyClass
{
    // Class Properties
    private $_name;

    // Class Method Setter
    public function setName($name){
        $this->_name = $name;
    }

    // Class Method Getter
    public function getName(){
        return $this->_name;
    }
}

$obj = new MyClass();
$obj->setName( name: "Sérgio");
echo $obj->getName();
```

MULTIPLAS INSTÂNCIAS

```php
<?php

$firstPerson = new MyClass();
$firstPerson->setName("Sérgio");
echo $firstPerson->getName();

$secondPerson = new MyClass();
$secondPerson->setName("Nogueira");
echo $secondPerson->getName();

?>
```

CONSTRUTORES

O construtor é método especial utilizado para definir o comportamento inicial de um objeto, ou seja, ações executado durante a criação de um objeto. O método construtor é chamado automaticamente no momento em que instaciamos um objeto através do operador *new*. Quando o construtor não for definido, todos os atributos do objeto criado são inicialiazadas com o valor NULL

CONSTRUTORES

```php
// Class name and definition
class MyClass
{
    // Class Properties
    private $_name;

    public function __construct($name)
    {
        $this->_name = $name;
    }

    // Class Method Setter
    public function setName($name){
        $this->_name = $name;
    }

    // Class Method Getter
    public function getName(){
        return $this->_name;
    }
}
```

MULTIPLOS CONSTRUTORES

Para construir vários construtores para mesma classe é preciso criar o construtor que identifique quantos parâmetros estão a ser passados para o construtor e em seguida é chamado o construtor correspondente.

MULTIPLOS CONSTRUTORES

```php
public function __construct()
{
    $this->_name = "asd";
    $this->_age    = 1;

    $arguments               = func_get_args();
    $numberOfArguments = func_num_args();

    if (method_exists($this, $function = '__construct'.$numberOfArguments)) {
        call_user_func_array(array($this, $function), $arguments);
    }
}
```

MULTIPLOS CONSTRUTORES

```
public function __construct1($name)
{
   $this->_name = $name;
   $this->_age    = 2;
}

public function __construct2($name, $age)
{
   $this->_name = $name;
   $this->_age    = $age;
}
```

MULTIPLOS CONSTRUTORES

```php
$obj1 = new MyClass();
echo $obj1->getAge();

$obj2 = new MyClass("John");
echo $obj2->getAge();

$obj3 = new MyClass("John", 3);
echo $obj3->getAge();
```

DESTRUTORES

O destrutor é um método especial executado quando a objeto é removido da memória. O destrutor é chamado quando:

- Atribuir NULL ao objeto,
- Utilizar a função unset() sobre o objeto,
- O programa é finalizado,
- O método é utilizado para finalizar ligações, apagar ficheiros temporários, etc.

DESTRUTORES

```
public function __destruct(){
   echo "Boom";
}
```

DESTRUTORES

```
$obj      = new MyClass();
$objeto = null; // Calling __destruct


$objNew    = new MyClass();
unset($objNew); // Calling __destruct
```

__toString

O método __toString() permite que uma classe decida como se comportar quando convertida para uma string. Por exemplo, o que echo $obj; irá imprimir. Este método precisa retornar uma string, senão um erro nível E_RECOVERABLE_ERROR é gerado.

__toString

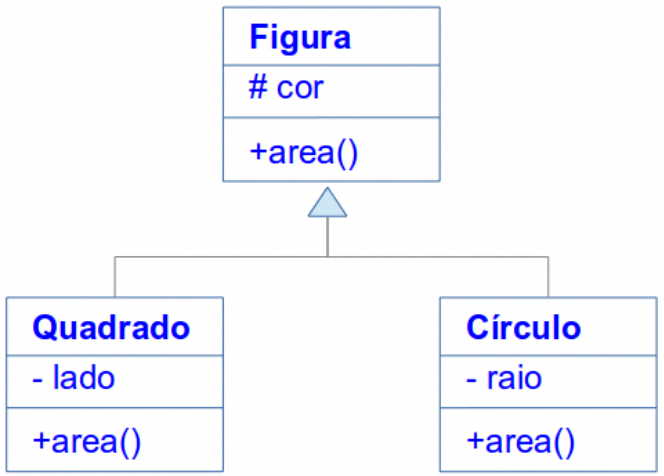
```php
public function __toString()
{
   return "Using the toString method: {$this->getAge()}";
}


$obj    = new MyClass();
echo $obj;
```

HERANÇA

A herança é um recurso da programação orientada à objetos que permite a partilha de atributos e métodos entre classes de uma mesma hierarquia (árvore). As classes inferiores da hierarquia (Subclasses) automaticamente herdam todas as propriedades e métodos das classes superiores (Superclasses)

HERANÇA



Nesse exemplo, a classe Figura é uma Superclasse para classes Quadrado e Círculo.

HERANÇA

Hieraquia

• Superclasse (Classe Pai)

– Possui os atributos e métodos que serão herdados.

– Desconhece as suas subclasses.

• Subclasse (Classe Filha)

– Classe considerada como uma especialização da Superclasse.

– Herda todas as propriedades e métodos das classes superiores.

– É possível adicionar na Subclasse comportamentos que não são específicados na Superclasse – A subclasse pode também ser uma Superclasse de futuras subclasses

PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

HERANÇA

Todos os atributos e métodos protegidos de uma Superclasse podem ser acessados pelas suas subclasses.

```
// Class Properties
protected $_name;
```

HERANÇA

```
class MyOtherClass extends MyClass
{
    public function newMethod()
    {
        echo "From a new method in " . __CLASS__ . ".<br />";
    }
}
```

HERANÇA

```php
// Create a new object
$newobj = new MyOtherClass;

// Output the object as a string
echo $newobj->newMethod();

// Use a method from the parent class
echo $newobj->getAge();
```

PARENT

Em PHP, o construtor da Superclasse não é chamado implicitamente pelo construtor da Subclasse. O construtor da Superclasse pode ser chamado pela Subclasse através da palavra-chave *parent*:

```
public function __construct() {
   parent::__construct();
}
```

PARENT

Assim como os construtores, o destrutor da Superclasse também não pode ser herdado pelas Subclasses. O destrutor da Subclasse deve chamar o destrutor da sua Superclasse através da palavra-chave *parent*

```
public function __destruct() {
    parent::__destruct();
}
```

POLIMORFISMO

Polimorfismo é uma palavra derivada do grego que significa “muitas formas”.

Na programação orientação a objeto é o principio que permite que Subclasses tenham métodos iguais, ou seja métodos que possuem a mesma assinatura da Superclasse, mas com comportamentos diferentes. Os métodos que sofrem o polimorfismo são redefinidos em cada uma das subclasses.

POLIMORFISMO

```
class MyClass
{

    public function myClassName() {
        return "Hello i'm MyClass";
    }

}
```

POLIMORFISMO

```
class MyOtherClass extends MyClass
{

    public function myClassName() {
        return "Hello i'm MyOtherClass";
    }

}
```

POLIMORFISMO

```php
// Create a new object
$obj    = new MyClass();
$newobj = new MyOtherClass;

// Output the object as a string
echo $obj->myClassName();

echo "<br>";

// Use a method from the parent class
echo $newobj->myClassName();
```

POLIMORFISMO

```php
// Create a new object
$obj    = new MyClass();
$newobj = new MyOtherClass;

echo $obj->myClassName();

echo "<br>";

echo $newobj->myClassName();
```

POLIMORFISMO

```
Hello i'm MyClass
Hello i'm MyOtherClass
```

STATIC

Pode aceder-se a um método estático sem instanciar a classe. Fornece-se simplesmente o nome da classe, o operador :: e o nome do método.

```
// Class name and definition
class MyClass
{
      public static $count = 0;
}


 echo MyClass::$count
```


PORTUGAL 2020
UNIÃO EUROPEIA
Fundo Social Europeu

FINAL

Um método pode ser marcado como um método final utilizando a palavra-chave *final* no início da sua declaração. Um método final impede a sobrescrita do método nas Subclasses. Sintaxe

```
final function myClassName() {
    return "Hello i'm MyClass";
}
```

FINAL

Uma classe pode evitar ter Subclasses se ela ser marcado como uma classe final. Como método final, basta adicionar a palavra-chave *final* no início da sua declaração.

```
final class ClassName {

}
```

CLASSE ABSTRATA

Uma classe abstrata é uma classe que serve de base para outras classe. Uma classe abastrata nunca pode ser instanciada na forma de objeto, somente suas Subclasses poderam ser instaciadas. A linguagem PHP irá automaticamente impedir que se instanciem objetos de classes abastratas. Sintaxe

```
abstract class ClassName {

}
```

MÉTODO ABSTRATO

Um método abstrato é um método que fornece assinatura para um método, mas sem nenhuma implementação. Qualquer classe que possuir métodos abstratos também deve ser abstrata. A implementação deverá ser feita nas subclasses. O principal uso de classes e métodos abstratos é garantir que cada subclasse so-brescreva os métodos abstratos da superclasse. Sintaxe

MÉTODO ABSTRATO

```
// Class name and definition
abstract class ClassA {
   abstract function show();
    //...
}
```

INTERFACES

Interfaces são um conjuntos de métodos que determinadas classes deverão implementar. Na declaração desses métodos devem ser informados a visibilidade, os nomes dos métodos e os respectivos parâmetros, sem qualquer implementação. A classe que implementar uma interface tem a obrigação de implementar todos os métodos definidos pela interface.

INTERFACES

```php
interface PersonRestrictions
{
    function setName($name);

    function getName();
}
```

```php
class Person implements PersonRestrictions
{
    private $name;

    function setName($name)
    {
        $this->name = $name;
    }

    function getName()
    {
        return $this->name;
    }

}
```

DocBlocks

Um Doblock é definido utilizando um comentário de bloco que começa com um asterisco adicional

```
/**
 * This is a very basic DocBlock
 */
```

DocBlocks

Um Doblock é definido utilizando um comentário de bloco que começa com a capacidade de utilizar tags, que começa com um símbolo (@) imediatamente seguido do nome da tag e do datatype. As tags documentais permitem aos Developers definir autores de um ficheiro, a licença para uma classe, a informação sobre propriedades ou métodos e outras informações úteis, ou seja, um asterisco adicional:

DocBlocks - TAGS

- **@author**: The author of the current element (which might be a class, file, method, or any bit of code) are listed using this tag. Multiple author tags can be used in the same DocBlock if more than one author is credited. The format for the author name is John Doe <john.doe@email.com>.
- **@copyright**: This signifies the copyright year and name of the copyright holder for the current element. The format is 2010 Copyright Holder.
- **@license**: This links to the license for the current element. The format for the license information is http://www.example.com/path/to/license.txt License Name.
- **@var**: This holds the type and description of a variable or class property. The format is type element description.

DocBlocks - TAGS

- **@var**: This holds the type and description of a variable or class property. The format is type element description.
- **@param**: This tag shows the type and description of a function or method parameter. The format is type $element_name element description.
- **@return**: The type and description of the return value of a function or method are provided in this tag. The format is type return element description.

DocBlocks - Exemplo

```
/**
 * A simple class
 *
 * This is the long description for this class,
 * which can span as many lines as needed. It is
 * not required, whereas the short description is
 * necessary.
 *
 * It can also span multiple paragraphs if the
 * description merits that much verbiage.
 *
 * @author Jason Lengstorf <jason.lengstorf@ennuidesign.com>
 * @copyright 2010 Ennui Design
 * @license http://www.php.net/license/3_01.txt PHP License 3.01
 */

class SimpleClass
```

DocBlocks - Exemplo

```
class SimpleClass
{
    /**
     * A public variable
     *
     * @var string stores data for the class
     */
    public $foo;
```

DocBlocks - Exemplo

```php
/**
 * Sets $foo to a new value upon class instantiation
 *
 * @param string $val a value required for the class
 * @return void
 */
public function __construct($val)
{
    $this->foo = $val;
}
```

DocBlocks - Exemplo

```php
/**
 * Multiplies two integers
 *
 * Accepts a pair of integers and returns the
 * product of the two.
 *
 * @param int $bat a number to be multiplied
 * @param int $baz a number to be multiplied
 * @return int the product of the two parameters
 */
public function bar($bat, $baz)
{
    return $bat * $baz;
}
}
```