PHP 8 - PARTE 2

Prof. Ruan Carlos Martins

19/08/2022

FUNÇÕES

Funções são blocos de código identificados por um nome e uma aasinatura, que permitem que o código seja realizado e organizado de uma forma muito mais consistente.

```php
<?php
//Esses blocos são executados quando são solicitados.
//Definimos uma função usando a palavra chave function,
//seguida de um nome, um conjunto de parênteses e um
//bloco de código.

function funcao()
{
    //código da função
}
//exemplo da sua aplicação
echo 'Início da nossa aplicação';
echo '<br>';
executar();
function executar()
{
    echo 'A função foi executada';
}
```

FUNÇÕES

No PHP o nome das funções é case insensitive.

```php
<?php


function Falar()
{
}
//São as mesmas funções mas esse tipo de operação não é permitido
function falar ()
{
}

//Podem ser usadas vários padrões para o nome das funções.
//Os mais comuns são o Camel Case e o Snake Case
function camelCase(){

}

function snake_case(){

}
```

PARÂMETROS DE FUNÇÕES

Dentro dos parênteses das funções, nós podemos definir um conjunto de parâmetros (assinatura da função)

```php
<?php
function adicao($a, $b){
    //código da função aqui
}

//Nesta função adicao, temos dois parâmetros $a e $b.
//São duas variáveis de PHP que vão existir dentro da função.
//Os parâmetros permitem passar valores para o interior de uma função, quando e
//Os parâmetros são separados por vírgulas.

adicionar(10,20);

function adicionar($a, $b){
    echo "$a + $b =" . $a + $b; //no PHP 7 emite um aviso
    echo "$a + $b =" . ($a + $b);
}
```

PARÂMETROS DE FUNÇÕES

No PHP 8 já não são necessários os parênteses nas operações.

```php
<?php

$nome = ['Ruan', 'Ana','Carlos'];

foreach($nomes as $nome){
    saudacao($nome);
}

function saudacao($valor)
{
    echo "Bom dia, $valor.<br>";
}

// Parâmetros são os nomes das variáveis existentes na definição da função.
//Argumentos são os dados passados quando a função é chamada
```

PARÂMETROS DE FUNÇÕES

PARÂMETROS OPCIONAIS

```php
<?php

//Podemos criar funções que tem valores pré definidos.
// A chamada dessas funções não obriga a definir argumentos para esses parâmetr

function multiplicar($a, $b = 2)
{
    //código aqui
    //$a será o valor passado por argumento
    //$b será:
    //O valor do argumento se ele for passado na chamada
    //O valor 2 se não for passado o argumento

    echo $a * $b;
}

multiplicar(10); //$a = 10 e $b = 2
echo'<br>';
multiplicar(10,20); //$a = 10 e $b = 20
echo '<hr>';
```

PARÂMETROS DE FUNÇÕES

PARÂMETROS OPCIONAIS

```php
<?php

//Os parâmetros opcionais tem que ser definidos após os
//parâmetros não opcionais.

function dividir($a = 2, $b)
{
    echo "$a e $b";
}
divir(2,10);
//No PHP 7 ainda era possível fazer isso sem avisos,
//mas no PHP 8, aparecerá um aviso.
```

PARÂMETROS DE FUNÇÕES

NAMED ARGUMENTS PHP8

```php
<?php

//O PHP 8 introduz uma novidade: named arguments.

function adicionar($a, $b = 10, $c = 30)
{
    echo $a + $b + $c;
}

//No PHP 7, se quisermos alterar o valor de $c e manter o valor de $b,
//temos que fornecer todos os valores
adicionar(100, 10, 300);

echo '<br>';

//No PHP 8 podemos fazer da seguinte forma:
    adicionar(c: 1000, a: 0);

    echo '<br>';
    adicionar(500, c: 1000);
```

PARÂMETROS DE FUNÇÕES

NOTAS ADICIONAIS SOBRE PARÂMETROS E VARIADIC PARAMETER

```php
<?php
//Uma função nunca pode ser chamada sem que os valores obrigatórios sejam
//fornecidos

function funcao($a, $b){
    //código aqui...
}

funcao('teste');
//isto é errado - só foi fornecido o valor de $a
//contudo, embora pouco frequente, podemos passar mais argumentos do que
//o solicitado
```

PARÂMETROS DE FUNÇÕES

NOTAS ADICIONAIS SOBRE PARÂMETROS E VARIADIC PARAMETER

```php
<?php
function outraFuncao($a){
    $x = func_get_arg(0);
    $y = func_get_arg(1);
    $z = func_get_arg(2);

    echo "$x - $y - $z";

    echo '<br>';
    echo func_num_args(10,20,30); //avalia quantos argumentos foram passados pa
}

outraFuncao(10,20,30);
echo '<br>';
```

PARÂMETROS DE FUNÇÕES

NOTAS ADICIONAIS SOBRE PARÂMETROS E VARIADIC PARAMETER

```php
<?php
//Também não muito comum, mas sendo possível, podemos usar um argumento
//especial designado por variadic parameter

function minha_funcao(...$argumentos){
    foreach($argumentos as $v){
        echo "$v<br>";
    }
}

minha_funcao(10,20,30,40,50);
```

PARÂMETROS DE FUNÇÕES

A EXPRESSÃO RETURN NUMA FUNÇÃO

```php
<?php
//A declaração return provoca o fim da execução do código de uma função,
//retornando ao local onde a função foi chamada.
function falar()
{
    return;
    echo 'Não vai ser apresentado este texto';
}
//Opcionalmente podemos usar o return para devolver um valor.
//Desta forma, uma função pode ser responsável, por exemplo,
//por desenvolver um conjunto de cálculos e desenvolver resultados.

function adicionar($a, $b)
{
    $resultado = $a + $b;
    return $resultado;
    //ou
    return $a + $b;
}
 echo adicionar(10,20); // = 30
```

PARÂMETROS DE FUNÇÕES

A EXPRESSÃO RETURN NUMA FUNÇÃO

```php
<?php
//A EXPRESSÃO RETURN NUMA FUNÇÃO
$nome = 'Ruan';
if(avaliar_nome($nome)){
    echo 'O cliente está correto';
}
function avaliar_nome($n){
    if($n == 'Ruan'){
        return true;
    } else {
        false;
    }
}
// Uma função sem qualquer valor de retorno, devolve sempre um valor null
function teste(){
    //código aqui..
}
if(falar() === null){
    echo 'Função com retorno igual a nulo';
}
```

ESCOPO DE UMA VARIÁVEL E EXPRESSAO GLOBAL

Normalmente uma variável de php passa a existir a partir do local onde é iniciada e existe até o final da página. No entanto, quanto mais temos variáveis dentro de uma função, elas tem um ciclo de vida limitado

```php
<?php
$variavel = 10;
echo "<p>$variavel<\p>";

funcao(20, 30);
echo "<p>¨$a e $b<\p>"; // estas variáveis existem dentro da função,
//mas não existem fora

function funcao($a, $b){
    echo "<p>$variavel</p>"; // Esta variável não é conhecida dentro da função
    echo "<p>$a e $b</p>";
}
function funcao2(){
    $v = 100;
}
funcao2();
echo "<p>$v</p>";
```

ESCOPO DE UMA VARIÁVEL E EXPRESSAO GLOBAL

EXPRESSAO GLOBAL

```php
<?php
//Neste exemplo, a variável $a existe em dois escopos diferentes
$a = 10;

function funcao()
{
    // global $a;
    $a = 20;
}
funcao();
echo "<p>$a</p>";

//Outro exemplo
$b = 100;
function funcao2()
{
     $GLOBALS['b'] = 200;
}
funcao2();
echo"<p>$b<\p>";
```

ESCOPO DE UMA VARIÁVEL E EXPRESSAO GLOBAL

ESCOPO E CICLO DE VIDA DE UMA VARIÁVEL

```php
<?php
//Ao contrário de outras linguagens, no PHP, as variáveis definidas
//dentro de loops ou instruções condicionais, não são destruídas
//dentro do escopo.

if(true){
    $a = 10;
}
echo "<p>$a<\p>";

for ($i = 0; $i < 10; $i++){
    $x = 1000;
}

echo "<p>$i e $x<\p>";

// Existem ainda um outro contexto de escopo de variáveis
// Quando estas são definidas dentro de uma classe,
//passando a ser designadas por propiedades de uma classe.
```

FUNÇÕES ANÔNIMAS

A partir do PHP 5.3 foram introduzidos as funções anônimas. Uma função anônima não tem nome e pode ser definida com valor a atribuir a uma variável.

```php
<?php

$a = function()
{
    echo '<p>Olá!</p>';
};
//Importante: vejam como, neste caso, após a chave de fecho da função tem que s
$a();

//Outro exemplo
$falar = function($mensagem)
{
    echo"<p>Eu digo: $mensagem</p>";
};

$falar('Estou aqui');
```

FUNÇÕES ANÔNIMAS

A partir do PHP 5.3 foram introduzidos as funções anônimas. Uma função anônima não tem nome e pode ser definida com valor a atribuir a uma variável.

```php
<?php
//As funções anônimas são particulamente úteis quando
// as queremos passar como argumentos para uma função.

$a = function()
{
    return'<p>Função A<\p>';
};
function falar($x){
    echo $x;
}

falar($a());
```

CLOSURES E ARROW FUNCTIONS

FUNÇÕES CLOSURE

```php
<?php

//São funções anônimas que podem usar variáveis do escopo global

$x = 20;
$x = 30;

$minhaClosure = function($z) use($x, $y)
{
    echo "$z - $x - $y";
    $y += 1000; // esta instrução não vai alterar o valor de $y

};

$minhaClosure(10);
echo "<p>$y<\p>"; //O valor de $y não foi alterado na closure
```

CLOSURES E ARROW FUNCTIONS

ARROW FUNCTIONS

```php
<?php

//São funções anônimas escritas de uma forma mais suscinta
//Foram introduzidas no PHP 7.4
//Suportam as mesmas características de uma função CLOSURE,
//Com a diferença que capturam automaticamente as variáveis globais.

$x = 20;
$y = 30;

$minhaFuncao = fn($z) => "$x - $y - $z";
echo $minhaFuncao(10);

//Usam palavra reservada fn e não necessitam do return nem das chaves.
```

GENERATORS

Um gerador é uma função que permite gerar séries de valores. Cada valor é devolvido pela função através da instrução. yield. Ao contrário do return, a instrução yield guarda o estado da função permitindo que a função continue a partir do estado onde ficou na última chamada.

```php
<?php

function buscar_numero(){
    for($i = 0; $i < 10; $i++){
        yield $i;
    }
}

//A função geradora funciona como um iterador, podendo ser usada num ciclo
//até que o gerador não tenha mais valores para desenvolver com o yield.

foreach(buscar_numero() as $numero){
    echo "$numero<br>";
}
```

GENERATORS

Os geradores foram introduzidos na versão 5.5 do PHP, mas foram melhores na versão 7 com a introdução do yield from, que permitem outro tipo de retorno:

Devolver valores de outro gerador,
Devolver valores de um array

```php
<?php
function buscar_nomes(){
    yield 'joao';
    yield 'Maria';
    yield from ['carlos', 'ana', 'ruan'];
    yield 'fernando';
}
echo '<hr>';
foreach(buscar_nomes() as $nome){
    echo "$nome<br>";
}
```

GENERATORS

Os geradores foram introduzidos na versão 5.5 do PHP, mas foram melhores na versão 7 com a introdução do yield from, que permitem outro tipo de retorno:
Devolver valores de outro gerador,
Devolver valores de um array

```php
<?php

// Como as funções geradoras não necessitam de tratar todos os dados
// de uma vez, quando usadas, podem aumentar substancialmente a performace do
//nosso script.

//NOTA: O PHP vem acompanhando de uma imensa coleção de funções que estão
// sempre disponíveis para realizar operações com arrays, com string, comunicar
//com bases de dados encriptação,
//operar com ficheiros e pastas, etc.
```

INTRODUÇÃO A CLASSES, PROPRIEDADES E MÉTODOS

Uma classe é um molde/forma/modelo a partir do qual criamos objetos. Exemplo: a classe Humano é um modelo para, a partir dela, criarmos um conjunto de homens e mulheres, cada um partilhando o mesmo "molde"mas com propiedades e funções com valores diferentes. Homem e mulher tem ambos cabelo, mas o homem pode ter cabelo escuro e a mulher cabelo claro.

```php
<?php
//Teoricamente:
// class Humano:
// > cabelo
// > genero
// > peso
// > caminhar

// Homem -> Humano
// cabelo - castanho
// genero - masculino
// peso - 75
// caminhar()
```

CLASSES

As classes são definidas pela declaração class, o seu nome e o bloco de código que contém as suas propiedades e métodos por convensão PSR-1, o nome de uma classe deve ser sempre atribuido na forma se StudlyCaps/MixedCase. O corpo deve ser definido da seguinte forma:

```php
<?php

class Humano
{
    //Propiedade e método
}

class JogadorFutebol
{
    //Propiedade e métodos
}
```

CLASSES

Propiedade e métodos

```php
<?php

//PROPIEDADES: São variáveis que guardam as características do objeto.
//MÉTODOS: São funções que definem o que o objeto pode fazer.

// As propiedades são também conhecidas como fields (campos) ou atributos
// de uma classe no PHP, as propiedades tem que ter um nível de acesso
// especifidado. veremos o que significa mais à frente

 class FiguraGeometrica
{
    public $largura, $altura;
    public $x;
    public $y;

    function novaArea($a, $b){
        return $a * $b;
    }
}
```

CRIAÇÃO DE OBJETOS, INSTANCIAÇÃO

Para aceder às propiedades de uma classe, dentro dos métodos da classe, é usada a pseudo variável "*this*" seguida do operador seta ->

```php
<?php
class Humano
{
    public $nome = "Ruan";
    public $apelido = "Martins";

    public function nomeCompleto(){
        return $this->nome . ' '. $this->apelido;
    }
}
/* As classes recorrem à utilização de Access Modifiers - Níveis de acesso
Os níveis de acesso aos dados indicam se podemos ver os dados apenas dentro da
se podemos ver fora da classe ou se estão protegidos por algum motivo extra.
Veremos mais à frente com estas informações são importantes.
*/
```

INSTANCIAR UM OBJETO

Um objeto é uma variável criada a partir de uma classe. Instanciar um objeto significa criar um objeto a partir de uma classe atribuindo à variàvel a expressão new e o nome da classe.

```php
<?php

$homen = new Humano();

//veremos depois porque dos parênteses.
//com a implementação anterior podemos agora aceder às propiedades e métodos

echo $homem->nomeCompleto(); // Ruan Martins

//Tal como acontece com as funções, os objetos podem ser instanciados mesmo
// se a definição da classe aparecer mais abaixo na script
```

CONSTRUCTOR

O constroctor é um método especial dentro de uma classe que é sempre executado automaticamente quando é criado um objeto a partir de uma classe. Este método é definido de uma forma especial com (*doisunderscores*) .São chamados métodos mágicos porque tem uma execução específica ou automática associada.

```php
<?php
class Humano
{
    private $nome;
    private $apelido;

    function __construct($n, $a)
    {
        $this->nome = $n;
        $this->apelido = $a;
    }

    public function nomeCompleto(){
        return $this->nome . ' ' . $this->apelido;
    }
}
```

CONSTRUCTOR

```php
<?php
$homem = new Humano ('Ruan','Martins');
$mulher = new Humano('Ana','Martins');

echo $homem->nomeCompleto();
echo'<br>';
echo $mulher->nomeCompleto();

/*Para classes que tem construtor sem parâmetros, podemos
instanciar da seguinte forma:
*/
class Humano
{
    function __construct()
    {
        //código
    }
}
$homem = new Humano;
//ou
$homem = new Humano();
```

PROPERTY PROMOTION NO PHP 8

Exemplos a seguir

```php
<?php
class Humano1
{
    public $nome;
    public $apelido;
}
class Humano2
{
    public function falar()
    {
        //código
    }
    private function andar()
    {
        //método apenas acessível dentro da class
    }
    public function movimento(){
        $this->andar();
    }
}
```

PROPERTY PROMOTION NO PHP 8

Exemplos a seguir

```php
<?php

/* Podemos ter classes sem qualquer tipo
de método e podemos ter classes sem qualquer tipo de propiedades
*/

/*Ao instanciar-mos a classe Humano1, podemos acessar diretamente às suas
propiedades,pelo fato de estarem indentificadas com o access modifier public
*/

$homem = new Humano1();
$homem->nome = "Ruan";
$homem->apelido = "Martins";

$mulher = new Humano2();
$mulher->andar();
```

PROPERTY PROMOTION NO PHP 8

Com PHP8 foi introduzido o conceito de property promotion no constructor. Isso permite definir propiedades diretamente nos parâmetros do construtor. Vejamos um exemplo "*antes*" e "*depois*"

```php
<?php
class Humano1
{
    public $nome;
    public$apelido;

    function __construct($n, $a)
    {
        $this->nome = $n;
        $this->apelido = $a;
    }
}
```

PROPERTY PROMOTION NO PHP 8

No PHP podemos criar a mesma classe da seguinte forma

```php
<?php
class Humano2
{
    function __construct(public $nome, public $apelido)
    {
        $this->nome = $nome;
        $this->apelido = $apelido;
    }
}
$h1 = new Humano1('Ruan','Martins');
$h2 = new Humano2('Ana', 'Martins');

echo $h1->nome . ' ' . $h1->apelido;

/* IMPORTANTE: Ao contrário do nome das variáveis, o nome das
classes é case insensitive.
*/
$h3 = new humano1('nome','apelido');
echo '<br>';
echo $h3->nome . ' ' . $h3->apelido;
```

CLASSES ANÔNIMAS

Com o PHP7 foi introduzido um conceito usado em outras linguagens e que se designa por classes anônimas. Este tipo de classes só faz sentido quando queremos instanciar apenas um objeto dessa classe.

```php
<?php
//Exemplo "normal";
class Objeto1
{
    function teste()
    {
        echo 'teste - normal';
    }
}

$a = new Objeto1();
```

CLASSES ANÔNIMAS

CLASSES ANÔNIMAS

```php
<?php
//Exemplo com classes anônimas

$b = new class
{
    function teste()
    {
        echo 'teste - classe anônima';
    }
}; //Nota: é importante fechar o código com ;

$a->teste();
echo '<br>';
$b->teste();
```

HERANÇA DE CLASSES

A herança é um mecanismo através do qual podemos criar classes que herdam propiedades e métodos de outra classe.

```php
<?php

/*A classe inicial a partir da qual outras vão ser criadas, é
designada por classe base
*/

/*Todas as classes que vão herdar propiedades e métodos da classe
base são designados por classes derivadas
*/
```

HERANÇA DE CLASSES

```php
<?php

class Animal
{
    public $especie;
    public $peso;

    function tipoEspecie(){
        return "Este animal é da espécie {$this->especie}";
    }
}

$animal = new Animal();
$animal->especie = 'Mamíferos';

echo $animal ->tipoEspecie();
```

HERANÇA | INHERITANCE

Para herdar uma classe a partir de outra, usamos a expressão extends.

```php
<?php

//Exemplo (uma classe base):

class Animal
{
    public $especie;
    public $peso;

    function tipoEspecie(){
        return "Este animal é da espécie ($this->especie)";
    }
}
```

HERANÇA | INHERITANCE

```php
<?php

// Exemplo (Uma classe com herança):
class Mamifero extends Animais
{
    //não é necessário voltar a definir
    //as propiedades e métodos que já existem na classe base

    //podemos acrescentar outras propiedades e outros métodos
    public $quantidade_pernas;
    public $tem_pelo;

    function temQuantasPernas(){
        return "O animal da espécie {$this->especie} tem {$this->quantidade_pe
    }
}
$mamiferos = new Mamifero();
$mamifero->especie = 'Cavalo';
$mamifero->quantidade_pernas = 4;
echo $mamifero->temQuantasPernas();
```

OVERRIDE EM CLASSES

```php
<?php
/* O mecanismo de overriding permite a uma classe
derivada ter métodos rescritos especificamente para
essa classe.
 Por exemplo, podemos ter um método TESTE na classe
base e ter o mesmo método TESTE com código diferente
na classe derivada
*/
class Animal
{
    function mover(){
        echo 'Mover a partir da classe base.';
    }
}
class Mamifero extends Animal
{
}
```

OVERRIDE EM CLASSES

```php
<?php
class Peixe extends Animal
{
    function mover(){
        echo 'Mover a partir da classe peixe';
    }
}

$animal = new Animal();
echo $animal -> mover();
echo'<br>';

$mamifero = new Mamifero();
echo $mamifero->mover();
echo '<br>';

$peixe = new Peixe();
echo $peixe->mover();
```

OVERRIDE EM CLASSES

Este sistema também é aplicável às propiedades.

```php
<?php

class Animal
{
    public $especie = 'Ave';
}

class Mamifero extends Animal
{
    public $especie = 'Cavalo';
}
$a = new Animal();
echo $a->especie;
echo '<br>';
$b = new Mamifero();
echo $b->especie;
```

OVERRIDE

Para além do conceito da classe base e classe derivada, temos o conceito de parent class (classe pai) É a classe a partir da qual fazemos a derivação

```php
<?php
//Vamos ver um exemplo com contrutor deste tipo de classes

class Retangulo
{
    public $largura, $altura;
    function __construct($l, $a)
    {
        $this->largura = $l;
        $this->altura = $a;
    }

    function calcularArea(){
        return $this->largura * $this->altura;
    }
}
```

OVERRIDE

```php
<?php
class Quadrado extends Retangulo
{
    function __construct($l)
    {
        $this->largura = $l;
        $this->altura = $l;
    }
}
$ret = new Retangulo(10,20);
$quad = new Quadrado(10);

echo $ret->calcularArea();
echo '<br>';
echo $quad->calcularArea();
echo '<br>';
```

IMPEDIR HERANÇA COM A EXPRESSÃO FINAL

Para impedir que uma classe derivada possa fazer override de métodos podemos utilizar a expressão FINAL da seguinte forma:

```php
<?php
class Veiculo
{
   final function mover(){
        //código aqui
    }
}
class Bicicleta extends Veiculo
{
    function mover(){
        //override do código original
    }
}
```

IMPEDIR HERANÇA COM A EXPRESSÃO FINAL

Podemos inclusive, definir uma class como não sendo possível de ser herdada colocando FINAL antes da CLASS

```php
<?php

final class Humano
{
    //código aqui
    function teste(){
        echo 'teste';
    }
}
class Homem extends Humano
{
    //código aqui
}

$a = new Homem();
$a->teste();
```

ACCESS LEVELS - NÍVEIS DE ACESSO

Existem 3 tipos de níveis de acesso a elementos dentro de uma classe public, protected e private

```php
<?php

class MinhaClass
{
    public $v1;
    protected $v2;
    private $v3;
}

$a = new MinhaClass();

$a->v1 = '111'; //possível
$a->v2 = '222'; //não é possível. Resultado em erro.
$a->v3 = '333'; //não é possível. Resultado em erro.
```

ACCESS LEVELS - NÍVEIS DE ACESSO

Os membros públicos (propiedades ou métodos) de uma classe estão sempre acessíveis. Se criarmos um objeto a partir da classe, temos acesso direto a esses elementos.

```php
<?php
class TudoPublico
{
    public $propiedade;
    public function metodo(){
        echo 'Método público';
    }
}
$obj1 = new TudoPublico();
$obj1->propiedade = "a";
$obj1->metodo();
```

ACCESS LEVELS - NÍVEIS DE ACESSO

Um elemento protected pode ser alcançado dentro da classe e dentro de uma qualquer classe que seja uma extensão de outra classe

```php
<?php
class ClasseTeste
{
    public $public = "a";
    protected $protegida = "b";
    private $privada = "c";
}

class ClasseDerivada extends ClasseTeste
{
    function teste()
    {
        echo $this->public; //Possível
        echo $this->protegida; //Possível
        echo $this->privada; //erro
    }
}
```

ACCESS LEVELS - NÍVEIS DE ACESSO

```php
<?php

$a = new ClasseTeste();
$a->publica ="1"; //Possível
$a->protegida = "2";//gera erro
$a->privada = "3"; //gera erro

$b = new ClasseDerivada();
$a->publica = "a"; //Possível
$a->protegida = "b";//erro
$a->privada = "c"; //erro

$b-> teste();

//PRIVATE

/* Só pode ser vísivel dentro da própia classe.
Não é visível nos objetos instanciados nem em
outras classes herdadas.
*/
```

ACCESS LEVELS - NÍVEIS DE ACESSO

PRIVATE: Só pode ser vísivel dentro da própia classe. Não é visível nos objetos instanciados nem em outras classes herdadas.

```php
<?php

/* Ao contrário das propiedades, os métodos não
necessitam ter um nível de acesso específico
claramente. Se não for identificado, por defeito
método público.
*/

class Teste
{
    private $valor = 'a';

    function mover(){
        //código aqui
        echo $this->valor;
    }
}
$a = new Teste();
$a->mover(); //Possível
```

VAR E OBJECT ACCESS

VAR Keyword:
Tem o mesmo comportamento de public, mas apenas existe por retrocompatibilidade devido ao código escrito antes do PHP5. Não é recomendável usar porque poderá em breve passar a ser não suportado

```php
<?php
class Homem
{
    var $nome, $apelido;
}

$eu = new Homem();
$eu->nome = 'Ruan';
$eu->apelido ="Martins";
```

VAR E OBJECT ACCESS

No PHP, um objeto instanciado a partir de uma classe pode acessar a elementos privados e protegidos de outro objeto criado a partir da mesma classe. Este comportamento não acontece na maior parte das linguagens de programação.

```php
<?php
class Humano
{
    private $nome = 'x';

    function setPrivate($objeto, $valor){
        $objeto->nome = $valor;
    }

    function apresentar(){
        echo $this->nome;
    }
}
$a = new Humano();
$b = new Humano();
$a->setPrivate($b,'Ruan');
```

GETTERS E SETTERS

Uma boa prática é criar o menor número de propiedades de uma classe como públicas. Colocar uma propiedade como pública é expor em de em demasia essa propiedade. Um exemplo: imaginem que querem que uma propiedade de uma classe seja sempre um número.

```php
<?php

//ACCESS LEVEL - Aspectos importantes a memorizar

class Humano
{
    public $idade;
}
$eu = new Humano();
$eu->idade = 'Olá mundo!'; /*isto é possível porque o PHP não é strongly typed
*/
```

GETTERS E SETTERS

Para definir corretamente este tipo de situação, podemos usar métodos para definir e ir buscar os valores das propiedades privadas. Estes métodos permitem garantir que a integridade das propiedades sejam mantidas.

```php
 <?php
//EX1:
class Humano1
{
    private $idade;
}

$eu1->idade = 'Ruan'; //Não é possível porque a propiedade é privada
```

GETTERS E SETTERS

```php
<?php
//EX2:
class Humano2
{
    private $idade = 0;

    //Verifica se o $valor é número
    public function setIdade($valor){
        if(is_numeric($valor)){
            $this->idade = $valor;
        }
    }

    public function getIdade(){
        return $this->idade;
    }
}

$eu2 = new Humano2();
$eu2->setIdade(23);
echo $eu2->getIdade();
```

GETTERS E SETTERS

```php
<?php
//EX3:
class Tempo
{
    function setSegundos($valor){
        if(!is_numeric(($valor) || $valor<0)){
            $this->segundos = 0;
        } else{
            $this->segundos = $valor;
        }
    }//------------------
    function getMinutos(){
        return $this->segundos / 60;
     }//-----------------
     function setMinutos($valor){
        if($valor == 0){
            $this->segundos = 0;
        } else {
            $this->segundos = $valor * 60;
        }
     }
}
```