MÓDULO

PROGRAMAÇÃO PHP

UNIDADE

PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

Instituto Tecnológico

ÍNDICE

OBJETIVOS

Com esta unidade didática, pretende-se que desenvolva os seguintes objetivos de aprendizagem:

- Compreender o sistema de programação orientado a objetos.
- Criar e utilizar classes em programação.
- Criar propriedades e métodos para as classes.
- Utilizar os métodos de herança, visibilidade e clonagem das classes.

INTRODUÇÃO

A Programação Orientada a Objetos permite criar um sistema de programação modular dividido em classes que facilitam a programação e minimizam a quantidade de código a ser escrito.

Em PHP, a utilização deste tipo de programação é muito frequente, visto que permite uma escrita mais estruturada e limpa.

Nesta unidade, aprenderá como funciona o PHP no modelo de Programação Orientada a Objetos. Este modelo pode ser utilizado em qualquer aplicação ou página que pretender fazer, e torna a programação e a limpeza do código muito mais fácil.

Quando utilizado com frequência, este sistema de programação também economiza tempo ao programador, uma vez que as suas classes genéricas podem ser criadas para serem reutilizadas noutros programas. Por exemplo, uma classe que gere uma base de dados.

1. PROGRAMAÇÃO ORIENTADA A OBJETOS

A Programação Orientada a Objetos é baseada, como o próprio nome sugere, na criação de objetos. São entidades que contêm propriedades (valores) e métodos (funções). Um objeto irá conter tudo o que é necessário para se poder criar e, uma vez criado, distingui-lo dos outros objetos da mesma classe.

Uma classe é o array ou o esquema de um objeto que contém as suas propriedades e métodos, e é o elemento a partir do qual são criadas as instâncias que serão os objetos.

A Programação Orientada a Objetos ou POO (como será chamada de aqui em diante) permite que os programadores não tenham de repetir código. Sem este sistema, um programador que, por exemplo, criasse uma página web de aluguer de livros iria repetir os mesmos padrões de código inúmeras vezes, enquanto, se utilizasse a POO, poderia procurar esses padrões de código comuns e criar uma classe para reutilizar o código sempre que pretendesse e utilizá-lo na página simplesmente criando uma instância da classe criada.

Para entender melhor a POO, é mais fácil pensar num objeto como se se tratasse de um objeto real: por exemplo, se pensar numa biblioteca que necessita de alugar livros. O livro seria o objeto que é preciso instanciar (criar), logo, seria criada a classe "livro"; esta classe teria as suas propriedades, como título, autor, se está ou não alugado, ano, entre outras, e os métodos seriam todas as opções de uso do livro, como alugar, devolver, cancelar, entre outros.

Assim, uma POO começa por definir muito especificamente quais os objetos que podem ser agrupados numa classe e por criar essas mesmas classes.

2. CRIAR CLASSES, PROPRIEDADES E MÉTODOS

Como visto no ponto anterior, a POO é baseada no uso de classes e a criação de uma classe é feita com a palavra reservada **class** seguida pelo identificador da classe e do código entre chavetas.

```
<?php
class novaClasse {
}
?>
```

O identificador da classe pode ser qualquer tag, exceto as palavras reservadas do PHP (echo, this, print, etc.), e, além disso, deve começar com uma letra ou um underscore (_) seguido pelas letras, números e underscores que pretender.

Para criar uma instância de uma classe, é utilizada a palavra reservada **new**. As classes devem ser definidas antes da chamada de construção.

```
<?php
class livro{}
$liv = new livro();
?>
```

As propriedades são variáveis ou constantes dentro do objeto que permitem definir os seus parâmetros e são definidas utilizando uma das seguintes palavras reservadas:

- **public**: podem ser acedidos a partir de qualquer lugar.
- **protected**: só podem ser acedidos a partir da própria classe, classes herdadas ou classes pai.
- **private**: só podem ser acedidas a partir da classe que os definiu.

```
<?php
class livro{
public $titulo;
}
?>
```

Os métodos são funções dentro da classe que são atribuídas automaticamente a cada instância (objeto) dessa classe e que ajudam a interagir com o objeto. Para os criar, são definidos da mesma forma como se fossem funções normais, e para aceder às propriedades do objeto é utilizada a palavra reservada **$this**, que se refere à instância da classe.

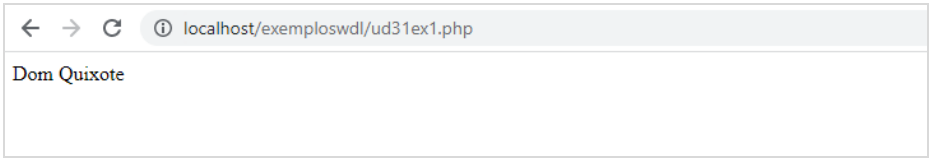
```
<?php
class livro{
public $titulo;
public function mostrarTitulo(){
echo $this->titulo;
}
}
?>
```

Na classe do exemplo anterior verá que foi utilizada a palavra reservada $this para aceder à propriedade "titulo" do objeto, mas, ao aceder, não é utilizado o cifrão ($) antes da palavra "titulo", pois já fora utilizado na palavra reservada $this.

Neste exemplo, será criada a classe "livro", com a qual poderá interagir para descobrir o nome do livro.

```
<html>
<body>
<p>
<?php
class livro{
public $titulo;
public function mostrarTitulo(){
echo $this->titulo;
}
}
$liv1 = new livro();
$liv1->titulo = 'Dom Quixote';
$liv1->mostrarTitulo();
?>
</p>
</body>
</html>
```

O resultado desta página será:



Dentro da classe também é possível definir uma constante e, como o próprio nome indica, se deve ser um valor constante, ou seja, não ser uma variável ou o resultado de uma operação matemática, ou uma função.

Para definir uma constante age-se da mesma forma usada para definir uma constante normal, sendo que estas são definidas sem utilizar o cifrão ($) e utilizando a palavra reservada **const**.

```
<?php
class novaClasse {
const constante = ‘valor’;
}
?>
```

Uma prática muito comum recomendada por programadores é criar as classes como ficheiros .php separados do código original e fazer um include ou um require para poder utilizar essas classes. Desde a versão 5 do PHP que existe um método de carregamento automático de classes para evitar ter de colocar uma lista de inclusões no início das páginas. Este método chama-se **__autoload** e é invocado sempre que uma instância de classe é criada.

A função __autoload será utilizada para carregar a classe “livro” que foi criada num ficheiro PHP separado do código da página.

Exemplo:

```
<?php
function __autoload($classe){
include $classe.‘.php’;
}
$liv1 = new livro();
$liv1->titulo = 'Dom Quixote';
$liv1->mostrarTitulo();
?>
```

Neste exemplo será utilizada a classe “livro” inserida no ficheiro “livro.php”, tendo em consideração que, para que o ficheiro funcione, o seu nome deve coincidir com o nome da classe.

- livro.php.

```
<?php
class livro{
public $titulo;
public function mostrarTitulo(){
echo $this->titulo;
}
}
?>
```

Se criar uma função que contém um método, ele será chamado exatamente da mesma forma que a classe, e será executado quando a classe for criada, desde que um construtor não seja definido (explicado no ponto seguinte desta unidade), e podem ser passados parâmetros.

```
<?php
class novaClasse {
public $var;
public function novaClasse($dado){
$this->var = $dado;
}
}
$obj = new novaClasse('teste');
echo $obj->var;
?>
```

3. CONSTRUTORES E DESTRUIDORES

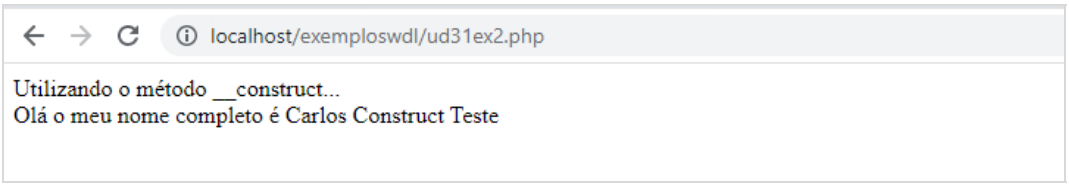
No PHP, uma classe com a instância new criará um objeto dessa mesma classe. Desde a versão 5 do PHP, existem alguns métodos das classes que são os construtores (**__construct**) e os destruidores (**__destruct**) que são chamados ao criar uma instância (desde que sejam definidos na classe) e os parâmetros podem ser passados para estes métodos. O destruidor é invocado no final de toda a execução do código da página e o objeto é libertado.

Neste exemplo, será utilizado o construtor do objeto para criar instâncias do elemento "livro" com o título já inserido na propriedade correspondente. Neste caso, a definição da classe ficará na própria página.

```
<html>
<body>
<p>
<?php
class Pessoa {
private $nome;
private $apelidos;
// Constructor. É executado ao criar uma instância desta classe
function __construct($nome = '', $apelidos = '') {
$this->nome = $nome;
$this->apelidos = $apelidos;
}
```

```
function setNome($nome) {
$this->nome = $nome;
}
function getNome() {
return $this->nome;
}
function setApelidos($apelidos) {
$this->apelidos = $apelidos;
}
function getNomeCompleto() {
return $this->nome.' '.$this->apelidos;
}
}
$Pessoa = new Pessoa('Carlos','Construct Teste');
echo 'Utilizando o método __construct... <br>';
echo 'Olá o meu nome completo é '.$Pessoa->getNomeCompleto();
?>
</p>
</body>
</html>
```

O resultado será:



4. CLASSES ABSTRATAS

O conceito de classes abstratas é bastante utilizado em POO. Qualquer classe para a qual um método abstrato é definido também deve ser criada como uma classe abstrata. Uma classe abstrata determina a estrutura da classe, mas não a implementação da classe, e não pode ser instanciada.

Ao herdar uma classe abstrata de uma classe não abstrata, todos os métodos da classe pai devem ser redefinidos com a mesma visibilidade que tinham na classe pai.

Pode parecer que o uso de classes abstratas não faz sentido, mas, se pensar em exemplos da vida real, como um carro, uma mota ou uma bicicleta, estes têm aspetos em comum, com os quais poderia criar uma classe abstrata designada "veículo" e classes herdadas para cada tipo de veículo.

Neste exemplo, verá as duas opções: com e sem o uso de classes abstratas. O exercício consiste em criar uma página com classes que permitam representar uma figura geométrica, utilizando a base e a altura como propriedades e o cálculo da área da figura como método.

- Sem classes abstratas:

```
<html>
<body>
<p>
<?php
```

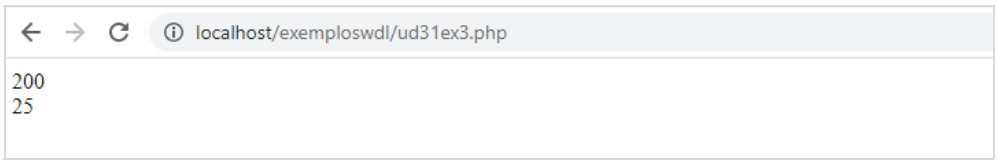
```
class figura {
private $base;
private $altura;
public function __construct($base, $altura) {
$this->base = $base;
$this->altura = $altura;
}
public function area($tipo_figura) {
switch ($tipo_figura) {
case 'rectangulo':
return $this->base * $this->altura;
break;
case 'triangulo':
return ($this->base * $this->altura) / 2;
break;
}
}
}
$objeto = new figura(20,10);
echo $objeto->area('rectangulo').'<br/>';
$objeto1 = new figura(10,5);
echo $objeto1->area('triangulo');
?>
</p></body>
</html>
```

- Com classe abstrata:

```
<html>
<body>
<p>
<?php
abstract class figura {
```

```
protected $base;
protected $altura;
abstract public function area();
}
class rectangulo extends figura {
public function __construct($base, $altura) {
$this->base = $base;
$this->altura = $altura;
}
public function area() {
return $this->base * $this->altura;
}
}
class triangulo extends figura {
public function __construct($base, $altura) {
$this->base = $base;
$this->altura = $altura;
}
public function area() {
return ($this->base * $this->altura) / 2;
 }
}
$objeto = new rectangulo(20,10);
echo $objeto->area().'<br/>';
$objeto2 = new triangulo(10,5);
echo $objeto2->area();
?>
</p></body>
</html>
```

Em ambos os casos o resultado será:



Como pode verificar neste exemplo, embora o método de criação com classes abstratas crie mais código, deixa o código muito mais limpo e é mais rápido na execução.

5. VISIBILIDADE E HERANÇA

Visibilidade

A visibilidade pode ser atribuída a uma propriedade ou um método. Para tal são utilizadas as palavras:

- **public**: podem ser acedidos de qualquer lugar.
- **protected**: só podem ser acedidos a partir da própria classe, classes herdadas ou classes pai.
- **private**: só podem ser acedidos a partir da classe que os definiu.

No caso das propriedades, se não forem declaradas com qualquer uma destas palavras, por defeito, a declaração será considerada public.

```
class novaClasse {
public $valorPublico;
private $valorPrivado;
function __construct($publico, $privado) {
$this->valorPublico = $publico;
$this->valorPrivado = $privado;
}
function setPrivado($privado) {
$this->valorPrivado = $privado;
```

```
}
function getPrivado() {
return $this->valorPrivado;
}
private function metodoPrivado() {
echo ' --- Olá através do método privado ---';
}
function metodoPublico() {
echo ' --- Olá através do método público, e daqui chamo o método
privado: --- <br /> ';
$this->metodoPrivado();
}
function soma() {
return $this->valorPublico + $this->valorPrivado;
}
}
$novaClasse = new novaClasse(4,5);
echo 'O valor público é '.$novaClasse->valorPublico;
// Remova o comentário da linha a seguir para verificar o erro
//echo $novaClasse->valorPrivado;
echo '<br> Propriedade privada não pode ser acedida!!';
echo '<br> Para obter valor privado, utilize o método getPrivado()
:'.$novaClasse->getPrivado();
echo '<br> Os métodos podem operar em propriedades públicas e
privadas. Exemplo soma(): '.$novaClasse->soma();
echo '<br> O mesmo acontece com os métodos e com as propriedades ';
echo '<br> É possível chamar os públicos $novaClasse-
>metodoPublico():<br>'.$novaClasse->metodoPublico();
echo '<br> Mas os privados não $novaClasse->metodoPrivado()';
// Remova o comentário da linha a seguir para verificar o erro
//echo $novaClasse->metodoPrivado();
```

Os métodos de uma classe também podem ser definidos como public, private ou protected, e terão a mesma visibilidade das propriedades; se um método não for criado com nenhuma visibilidade, por defeito será public.

A herança

O conceito é uma classe que é gerada a partir da herança, ou extensão, de outra já existente. As propriedades e métodos da classe principal são passados para a classe herdada, e podem ser acedidos a partir da segunda classe.

No exemplo seguinte verá como criar duas classes: uma normal e outra herdada.

```
<html>
<body>
<p>
<?php
class livro{
public $titulo;
function __construct(){
$this->titulo = func_get_arg(0);
}
public function mostrarTitulo(){
echo $this->titulo.'<br>';
}
}
class livro_ficcao extends livro{
public $tematica;
public function mostrarTematicaFiccao(){
echo $this->tematica.'<br>';
}
}
$liv1 = new livro('Dom Quixote');
$liv2 = new livro_ficcao('Encontros Imediatos de Terceiro Grau');
$liv2->tematica = 'Extraterrestres';
```

```
$liv1->mostrarTitulo();
//$liv1->mostrarTematicaFiccao();
$liv2->mostrarTitulo();
$liv2->mostrarTematicaFiccao();
?>
</p>
</body>
</html>
```

Se no exemplo anterior remover o comentário, a linha que tenta mostrar o tema do **$liv1**, verá que devolve um erro, pois este método encontra-se apenas nos objetos da classe **livro_ficcao**, enquanto os seus objetos têm acesso à função **mostrarTitulo**, uma vez que a herdaram da classe pai.

Os métodos herdados manterão a funcionalidade original da classe pai, a menos que sejam substituídos.

A visualização será:

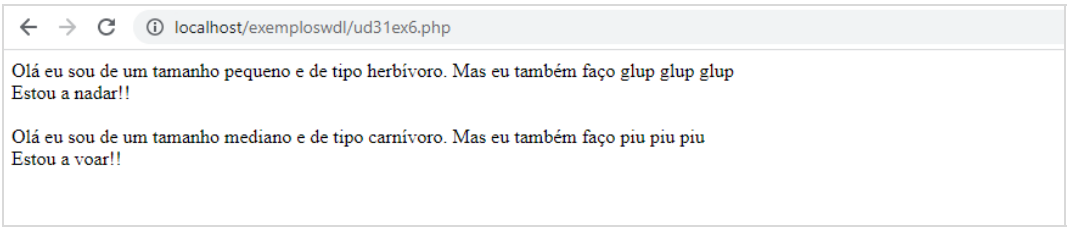
Segue-se outro exemplo de herança em classes:

```
<html>
<body>
<p>
<?php
class Animal {
private $tamanho = '';
```

```
private $tipo = '';

function __construct($tamanho, $tipo) {

$this->tamanho = $tamanho;

$this->tipo = $tipo;

}

function cumprimentar() {

echo 'Olá eu sou de um tamanho '.$this->tamanho.' e de tipo
'.$this->tipo.'.';}

}

class Peixe extends Animal {

// pode usar e modificar os métodos da classe principal

function cumprimentar() {

parent::cumprimentar();

echo ' Mas eu também faço glup glup glup';

}

// método que só tem esta classe

function nada() { echo 'Estou a nadar!!'; }

}

class Passaro extends Animal {

// pode usar e modificar os métodos da classe principall

function cumprimentar() {

parent::cumprimentar();

echo ' Mas eu também faço piu piu piu';

}

// método que só tem esta classe

function voa() { echo 'Estou a voar!!'; }

}

$peixe = new Peixe('pequeno','herbívoro');

$passaro = new Passaro('mediano','carnívoro');

// Peixe

$peixe->cumprimentar();

echo '<br>';
```

```
$peixe->nada();
echo '<br><br>';
// Pássaro
$passaro->cumprimentar();
echo '<br>';
$passaro->voa();

?>
</p>
</body>
</html>
```

A visualização será:



6. OPERADORES DE AMBIENTE

O operador de ambiente permite aceder a elementos estáticos e constantes de uma classe sem ter de criar instâncias dela, ou seja, pode aceder aos valores das constantes definidas ou às propriedades de tipo de variável e métodos definidos como static.

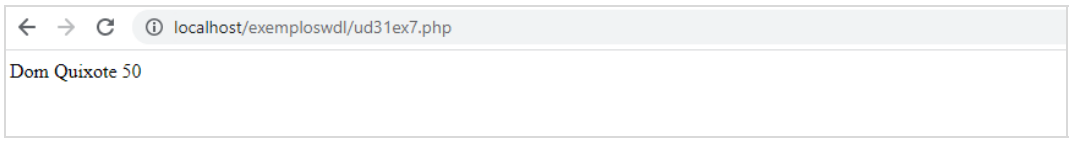
Para aceder a estes métodos ou propriedades da página, utiliza-se o nome da classe seguido por dois pontos duplos (: :) e o nome do método ou propriedade.

Exemplo:

```
<html>
<body>
<p>
<?php
class livro{
const titulo = 'Dom Quixote';
public static function vezesAlugado(){
echo ' 50';
}
}
echo livro::titulo;
echo livro::vezesAlugado();
```

```
?>
</p>
</body>
</html>
```

A visualização será:



Existem duas palavras reservadas que são utilizadas em POO para aceder aos dados entre classes:

- **self**: permite aceder às propriedades da própria classe da mesma forma que a palavra reservada $this.
- **parent**: válido apenas em classes herdadas, permite fazer referência às propriedades da classe pai a que pertence.

7. MÉTODOS COMUNS

Em POO, existem vários métodos comuns que podem ser definidos numa classe, alguns deles já referidos aqui anteriormente, como o __construct e o __destruct. Seguem-se alguns dos mais usados na programação:

- **__toString**: permite que uma classe defina como se deve comportar quando tratada como string. Por exemplo, se utilizar **echo $objeto**, este método vai devolver uma string.

```
<?php
class novaClasse {
public $cor;
public function __construct($valor){
$this->cor = $valor;
}
public function __toString(){
return $this->cor;
}
$objeto = new novaClasse('azul');
echo $objeto; //devolve azul
?>
```

- **__invoke**: chamado quando a instância de uma classe é chamada como se fosse uma função.

```
<?php
class novaClasse {
public $cor;
public function __construct($valor){
$this->cor = $valor;
}
public function __invoke($numero){
echo ‘comprar’.$numero.’de cor’.$this->cor;
}
}
$objeto = new novaClasse(‘azul’);
$objeto(4);
?>
```

Este método é válido apenas para a versão PHP 5.3.0 e posteriores (para saber qual a versão que está a utilizar, deverá escrever “echophpinfo();”).

8. CLONAR OBJETOS

O PHP dá a opção de clonar objetos utilizando a palavra reservada **clone**. O que essa função faz é uma cópia superficial das propriedades do objeto clonado, mantendo referências entre as variáveis. Assim que o processo de clonagem for concluído, o método comum **__clone** será chamado, desde que seja definido na classe de que foi clonado.

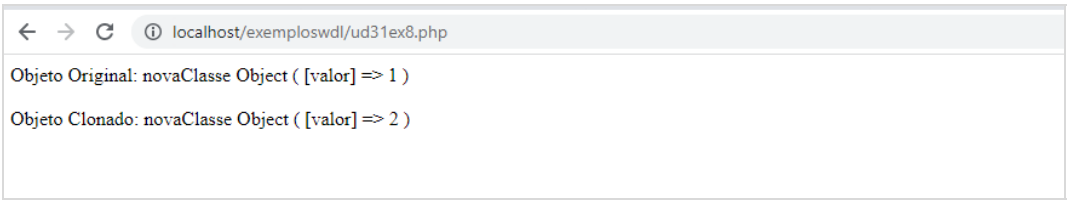
Neste exemplo, irá clonar-se uma classe e no seu método comum irá aumentar-se um parâmetro da mesma classe para ver como isso afeta a clonagem de elementos.

```
<html>
<body>
<p>
<?php
class novaClasse{
static $instancia = 0;
public $valor;
public function __construct() {
$this-> valor = ++self::$instancia;
}
public function __clone() {
$this-> valor = ++self::$instancia;
}
```

```
}
$objeto = new novaClasse();
$objeto2 = clone $objeto;
print("Objeto Original:\n");
print_r($objeto);
echo '<br/><br/>';
print("Objeto Clonado:\n");
print_r($objeto2);
?>
</p>
</body>
</html>
```

Como visto no exemplo, ao executá-lo pela primeira vez, a instância vale 1, enquanto ao exibir o objeto clonado vale 2, já que no método __clone o valor da instância é aumentado em um valor.

A visualização será:



O exemplo abaixo é um conjunto de tudo o que foi visto nesta unidade didática, aplicado a um caso real: a partir da classe “Pessoa”, irá criar uma classe “Professor” que estende (herda) de “Pessoa” os métodos e propriedades.

```
<html>
<body>
<p>
<?php
```

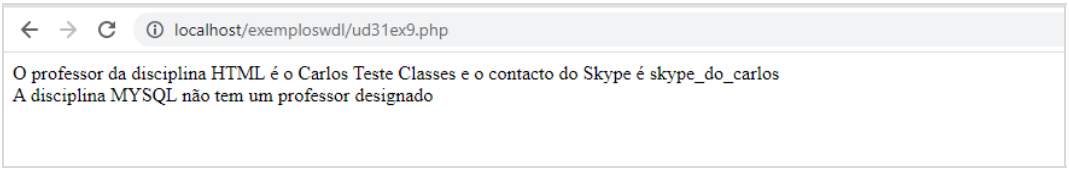
```
class disciplina {
private $nome;
private $Professor = null;
function __construct($nome) {
$this->nome = $nome;
}
function getNome() {
return $this->nome;
}
function setProfessor(Professor $professor) {
$this->Professor = $professor;
}
function getProfessor() {
return $this->Professor;
}
}
class Pessoa {
private $nome;
private $apelidos;
// Construtor. É executado ao criar uma instância desta classe
function __construct($nome = '', $apelidos = '') {
$this->nome = $nome;
$this->apelidos = $apelidos;
}
function setNome($nome) {
$this->nome = $nome;
}
function getNome() {
return $this->nome;
}
function setApelidos($apelidos) {
$this->apelidos = $apelidos;
```

```
}
function getNomeCompleto() {
return $this->nome.' '.$this->apelidos;
}
}
class Professor extends Pessoa{
private $skype;
function __construct($nome='', $apelidos='', $skype='') {
// Chama o construtor da classe pai Pessoa
parent::__construct($nome,$apelidos);
$this->skype = $skype;
}
function getSkype() {
return $this->skype;
}
}
// Embora não seja usado neste exemplo, pode imaginar as classes
que poderia criar que se estendem de Pessoa
class Aluno extends Pessoa{
private $skype;
private $numeroMatricula;
function __construct($nome='', $apelidos='', $skype='') {
// Chama o construtor da classe pai Person
parent::__construct($nome,$apelidos);
$this->skype = $skype;
}
function setNumeroMatricula($numeroMatricula) {
$this->numeroMatricula = $numeroMatricula;
}
function getNumeroMatricula() {
return $this->numeroMatricula;
}
```

```
}

// Definir um professor

$professor1 = new Professor('Carlos','Teste Classes',
'skype_do_carlos');

// Definir as disciplinas

$disciplina1 = new disciplina('HTML');

$disciplina2 = new disciplina('PHP');

$disciplina3 = new disciplina('MYSQL');

// Atribuir um professor às disciplinas

$disciplina1->setProfessor($professor1);

$disciplina2->setProfessor($professor1);

// Não é atribuido um professor à disciplina1, é obtido um objeto
do tipo Professor

$professorDisciplina = $disciplina1->getProfessor();

// Se o objeto não for nulo, existe um professor

if(!is_null($professorDisciplina)) {

echo 'O professor da disciplina '.$disciplina1->getNome();

// Chamar um método da classe Pessoa que foi herdado pela classe
Professor

echo ' é o '.$professorDisciplina-> getNomeCompleto();

// Por último chamar o método da mesma classe

echo ' e o contacto do Skype é '.$professorDisciplina->getSkype();

} else {

echo 'A disciplina '.$disciplina1-> getNome().' ainda não tem um
professor designado';

}

echo '<br />';

// Fazer o mesmo para a disciplina3

$professorDisciplina = $disciplina3->getProfessor();

if(!is_null($professorDisciplina)) {

echo 'O professor da disciplina '.$disciplina3->getNome();

echo ' é o '.$professorDisciplina->getNomeCompleto();

echo ' e o contacto do Skype é '.          $professorDisciplina-
>getSkype();
```

```
} else {

echo 'A disciplina '.$disciplina3->getNome().' não tem um professor
designado';

}

?>

</p>

</body>

</html>
```

A visualização será:



CONCLUSÃO

As classes são um dos recursos mais utilizados em PHP, pois permitem criar códigos reutilizáveis e organizar melhor as páginas.

O uso da POO, embora exija, inicialmente, mais tempo de estudo até que possa começar a criar, será bastante vantajoso no futuro.

AUTOAVALIAÇÃO

1. Em que se baseia a POO?

a) Na criação de classes a partir de objetos (estrutura) para os quais se pode apenas definir métodos.

b) Na criação de objetos a partir de classes (estrutura) para os quais se pode definir propriedades e métodos.

c) Na criação de objetos a partir de classes (estrutura) para os quais se pode apenas definir propriedades.

d) Na criação de objetos a partir de classes (estrutura) para os quais se pode apenas definir métodos.

2. Qual é a forma correta de definir uma classe?

a) def class novaClasse{}.

b) class novaClasse{}.

c) class novaClasse(){}.

d) classe novaClasse{}.

3. Que palavra é utilizada para criar um objeto de uma classe?

a) create.

b) make.

c) new.

d) newObj.

4. Se dentro de um método de uma classe se colocar "$this-> cor", o que é que acontece?

a) Acede ao atributo cor da classe.

b) Executa o método cor desta classe.

c) Acede ao valor da variável cor existente fora da classe.

d) Não é possível colocar uma classe dentro de um método, pois dará erro.

5. O que é que se pode utilizar como propriedade de uma classe?

a) Apenas variáveis.

b) Apenas constantes.

c) Variáveis e constantes, mas apenas aquelas de um tipo de cada vez.

d) Variáveis e constantes indiscriminadamente.

6. Que método se utiliza para carregar classes, num website, que são definidas em arquivos separados, sem ter de colocar um include para cada classe?

a) _load.

b) _autoload.

c) __autoload.

d) __load.

7. Como é que se pode criar objetos de uma classe sem utilizar o método __construct, mas passando os parâmetros iniciais?

a) Não é possível.

b) Criando a partir de um método que é chamado da mesma forma que a classe.

c) Criando um método que é chamado da mesma forma que as propriedades da classe que se pretende inicializar.

d) Criando variáveis com o mesmo nome das propriedades da classe antes de se criar a instância do objeto.

8. Qual é o método que permite executar um código específico sempre que um objeto é destruído?

a) _destruct.

b) __delete.

c) __ondelete.

d) __destruct.

9. Se se atribuir a visibilidade protected a uma propriedade de uma classe, a partir de onde estará acessível?

a) De qualquer lugar.

b) Da própria classe e das classes herdadas.

c) Da própria classe, classes herdadas ou classes pai.

d) Da própria classe.

10. Se se pretender que um método seja acessível apenas a partir da classe que o criou, que visibilidade é utilizada?

a) private.

b) protected.

c) public.

d) alone.

SOLUÇÕES

1.	b	**2.**	b	**3.**	c	**4.**	a	**5.**	d
6.	c	**7.**	b	**8.**	d	**9.**	c	**10.**	a

PROPOSTAS DE DESENVOLVIMENTO DO ESTUDO

Para saber mais sobre Programação Orientada a Objetos, visite o seguinte website:

- https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_orientada_a_objetos

BIBLIOGRAFIA

- Cabezas, L. M. (2010), *Php 5.* Madrid: AnayaMultimedia.
- The PHP Group (2001), "Classes e Objetos". Disponível em: https://www.php.net/manual/pt_BR/language.oop5.php Consultado a 17 de março de 2021.