O que é PHP Orientado a Objetos?

OOP Significa programação orientada a objetos.
A programação procedural é sobre a escrita de procedimentos ou funções que executam operações nos dados, enquanto a orientação a objetos é sobre a criação de objetos que contenham dados e funções.

A programação orientada a objetos tem várias vantagens sobre a programação procedural:

- OOP é mais rápido e fácil de se executar.
- OOP fornece uma estrutura clara para os programadores.
- OOP ajuda a manter o código “seco”, isto é, que não se repetirá e torna mais fácil de manter, modificar e depurar.
- OOP torna possível criar aplicações reutilizáveis completos com menos código e menos tempo de desenvolvimento.

Dica: O princípio “não se repita” é reduzir a repetição de código. Você deve extrair os códigos que são comuns para o aplicativo, e colocá-los em um único lugar para reutilizá-los em vez de repeti-los.

O que são classes e objetos?

Classe e objeto são os dois principais aspectos da programação orientada a objetos.
Exemplo:

Assim, classe é modelo para um objeto, e um objeto é uma instância de uma classe. Quando os objetos individuais são criados, eles herdam todas as propriedades e comportamentos de uma classe, mas cada objeto terá valores diferentes para as propriedades.

Uma classe é um modelo para objetos, e um objeto é uma instância de classe.

Definindo classes

Uma classe é definida usando a palavra-chave, seguida pelo nome da classe e um par de chaves({}). Todas as suas propriedades e métodos são para dentro da palavra reservada class.
Exemplo:

```
class Fruit {
  // code goes here...
}
```

Abaixo declaramos uma classe chamada Fruit composta por duas propriedades ($name e $color) e dois métodos set_name() e get_name() para selecionar e obter as propriedades $name.

Nota: em uma classe, variáveis são chamadas propriedades e funções são chamadas métodos

Definindo objetos

As classes não são nada sem objetos! Podemos criar múltiplos objetos de uma classe. Cada objeto contém todas as propriedades e métodos definidos na classe, mas eles terão diferentes valores de propriedade.
Objetos de uma classe são criados usando a palavra reservada new.

No exemplo abaixo, $apple e $banana são instâncias da classe fruit.

```php
<?php
    class Fruit{
        public $name;
        public $color;

        function __construct($name, $color){
            $this->name = $name;
            $this->color = $color;
        }
    }

    $apple = new Fruit('Apple', 'Red');

    echo $apple->name;
?>
```

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit();
$banana = new Fruit();
$apple->set_name('Apple');
$banana->set_name('Banana');

echo $apple->get_name();
echo "<br>";
echo $banana->get_name();
?>
```

No exemplo abaixo, nós adicionamos mais dois métodos para a classe Fruit, para setar e obter a propriedade $color:

```php
<?php
class Fruit {
  // Properties
  public $name;
  public $color;

  // Methods
  function set_name($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
  function set_color($color) {
    $this->color = $color;
  }
  function get_color() {
    return $this->color;
  }
}

$apple = new Fruit();
$apple->set_name('Apple');
$apple->set_color('Red');
echo "Name: " . $apple->get_name();
echo "<br>";
echo "Color: " . $apple->get_color();
?>
```

A palavra-chave $this

$this faz referência ao objeto atual, e isso apenas é possível dentro dos métodos.
Olhe para os exemplos:

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
?>
```

Então, onde podemos mudar o valor da propriedade $nome? há dois caminhos, você pode adicionar tanto dentro da classe, quanto fora da classe, exemplo:

```php
<?php
class Fruit {
  public $name;
  function set_name($name) {
    $this->name = $name;
  }
}
$apple = new Fruit();
$apple->set_name("Apple");
?>
```

```php
<?php
class Fruit {
  public $name;
}
$apple = new Fruit();
$apple->name = "Apple";
?>
```

Instanceof

Você pode usar a palavra reservada instanceof para checar se um objeto pertence a uma classe ou não, retornando sempre true ou false.

```php
<?php
$apple = new Fruit();
var_dump($apple instanceof Fruit);
?>
```

Constructor e Destructor

São duas palavras reservadas para funções em uma classe, são muito importantes. Veja a seguir

Constructor

Um construtor permite que você inicialize uma propriedade na criação de um objeto.

Se você cria uma function __construct(), o PHP automaticamente irá chamar a função no momento em que você criar o objeto.

Exemplo:

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function get_name() {
    return $this->name;
  }
}

$apple = new Fruit("Apple");
echo $apple->get_name();
?>
```

Destructor

Um destrutor permite ser chamado quando o objeto é destruído, ou seja, quando o script é finalizado.

Se você cria uma function __destruct(), o PHP automaticamente irá chamar essa função no final do script.

Exemplo:

```php
<?php
class Fruit {
  public $name;
  public $color;

  function __construct($name) {
    $this->name = $name;
  }
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}

$apple = new Fruit("Apple");
?>
```

Modificadores de acesso

Propriedades e métodos podem ser acessads com modificadores na qual podem controlar quem pode ser acessado.
Há três modificadores de acesso, listados do menos ao mais protegido:

- Public - Pode ser acessado por todos.
- Protected - Pode ser acessado dentro da classe e pelas classes derivadas da classe.
- Private - Apenas pode ser acessada dentro da classe.

Exemplo:

```php
<?php
class Fruit {
  public $name;
  protected $color;
  private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango'; // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300'; // ERROR
?>
```

Herança

Quando uma classe é derivada de outra classe.
A classe-filho irá herdar todas as propriedades e métodos que estão como public e protected da classe parente.
Uma classe herdada é definida por uma palavra reservada extends.
Exemplo:

```php
<?php
class Fruit {
  public $name;
  public $color;
  public function __construct($name, $color) {
    $this->name = $name;
    $this->color = $color;
  }
  public function intro() {
    echo "The fruit is {$this->name} and the color is {$this->color}.";
  }
}

// Strawberry is inherited from Fruit
class Strawberry extends Fruit {
  public function message() {
    echo "Am I a fruit or a berry? ";
  }
}
$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
?>
```

Herança e Protected

No capítulo anterior, aprendemos que propriedades e métodos protected podem ser acessados pela classe ou em classes derivadas. O que isso significa?

Podem sim ser acessados de classes derivadas, caso contrário irá retornar um erro, diferente de public, que funcionará normalmente.

Herança e substituições

Métodos podem ser reescritos nas classes herdadas utilizando o mesmo nome na classe filha.

A palavra reservada final

A palavra final pode ser utilizada para previnir que classes sejam herdadas ou previne que métodos sejam reescritos. (métodos constantes)

Basta adicionar a palavra reservada final antes de class.

Constantes

Constantes não podem ser modificadas uma vez que são declaradas.

Classes const são são case-sensitive. Contudo, é recomendável que constantes sejam escritas sempre com letras maiúsculas.

Para acessar uma constante fora da classe usando o nome da classe junto do operador (::)

Exemplo:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
}

echo Goodbye::LEAVING_MESSAGE;
?>
```

Ou, podemos acessar a constante dentro da classe usando a palavra-chave self.

Exemplo:

```php
<?php
class Goodbye {
  const LEAVING_MESSAGE = "Thank you for visiting W3Schools.com!";
  public function byebye() {
    echo self::LEAVING_MESSAGE;
  }
}

$goodbye = new Goodbye();
$goodbye->byebye();
?>
```

Classes e métodos abstratos

É quando uma classe parente tem um método nomeado, mas necessita de classes-filho para realizar as tarefas.

Uma classe abstrata é uma classe que contém pelo menos um método abstrato. O método abstrato é declarado, mas não implementado no código.

Exemplos:

```php
<?php
abstract class ParentClass {
  abstract public function someMethod1();
  abstract public function someMethod2($name, $color);
  abstract public function someMethod3() : string;
}
?>
```

ou:

```php
abstract class Car {
  public $name;
  public function __construct($name) {
    $this->name = $name;
  }
  abstract public function intro() : string;
}

// Child classes
class Audi extends Car {
  public function intro() : string {
    return "Choose German quality! I'm an $this->name!";
  }
}
```

Traços

O PHP suporta apenas uma herança, isto é, uma classe-filho pode herdar de apenas uma classe parente. Mas e se uma classe necessitasse herdar mais comportamentos? Os traços resolvem isso!

Exemplos:

```php
<?php
trait TraitName {
  // some code...
}
?>
```

```php
<?php
class MyClass {
  use TraitName;
}
?>
```

Métodos estáticos

Métodos estáticos podem ser chamados diretamente, sem precisar criar uma instância de uma classe.
Exemplo:

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

// Call static method
greeting::welcome();
?>
```

Uma classe pode conter métodos seja eles estáticos ou não. Um método estático também pode ser acessado na classe, novamente utilizando a palavra reservada self.

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }

  public function __construct() {
    self::welcome();
  }
}

new greeting();
?>
```

Métodos estáticos também podem ser chamados por métodos de outras classes, para isso, o método deverá estar como public ou protected.
Exemplo:

```php
<?php
class greeting {
  public static function welcome() {
    echo "Hello World!";
  }
}

class SomeOtherClass {
  public function message() {
    greeting::welcome();
  }
}
?>
```

Para chamar um método estático através de uma classe-filho, utilize a palavra-reservada parent dentro da classe filho.

```php
<?php
class domain {
  protected static function getWebsiteName() {
    return "W3Schools.com";
  }
}

class domainW3 extends domain {
  public $websiteName;
  public function __construct() {
    $this->websiteName = parent::getWebsiteName();
  }
}

$domainW3 = new domainW3;
echo $domainW3 -> websiteName;
?>
```

Propriedades estáticas

Propriedades estáticas podem ser chamadas diretamente, assim como os métodos estáticos, sem precisar criar uma instância de uma classe.

```php
<?php
class pi {
  public static $value = 3.14159;
}

// Get static property
echo pi::$value;
?>
```

Namespaces

Namespaces são qualificadores que resolvem dois problemas diferentes:

1. Eles permitem uma boa organização por um grupo de classes que trabalham juntas, para execução de tarefas.
2. Eles permitem que o mesmo nome pode ser usado para mais de uma classe.

Por exemplo, você pode ter um conjunto de classes que descrevem uma tabela HTML, como Tabela, Linha e Coluna, além de ter outro conjunto de classes para descrever móveis, como Mesa, Cadeira e Cama. Os namespaces podem ser utilizados para organizar as classes em dois grupos diferentes, além de impedir que duas classes da table e table sejam misturadas.

Declarando Namespaces

Exemplo:

```
namespace Html;
```

Obs: Um namespace deve ser declarado na primeira linha de código do arquivo PHP.

Constantes, classes e funções declaradas neste arquivo separado irão pertencer ao namespace html. Veja:

```
<?php
namespace Html;
class Table {
  public $title = "";
  public $numRows = 0;
  public function message() {
    echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
  }
}
$table = new Table();
$table->title = "My table";
$table->numRows = 5;
?>

<!DOCTYPE html>
<html>
<body>

<?php
$table->message();
?>

</body>
</html>
```

Para uma maior organização, é possível ter namespaces em ninhos separados:

```
namespace Code\Html;
```

Usando Namespaces

Qualquer código que conter após a palavra reservada namespace estará operando dentro de um namespace.
Para acessar classes fora de um namespace, a classe necessita de ter um namespace anexado a ele.

```
$table = new Html\Table()
$row = new Html\Row();
```

Namespaces com Aliases

Você também pode atribuir um apelido ao namespace, veja:

```
use Html as H;
$table = new H\Table();
```

Ou:

```
use Html\Table as T;
$table = new T();
```