Primeiros Passos em PHP Parte III

Thiago H. P. Silva

Execução

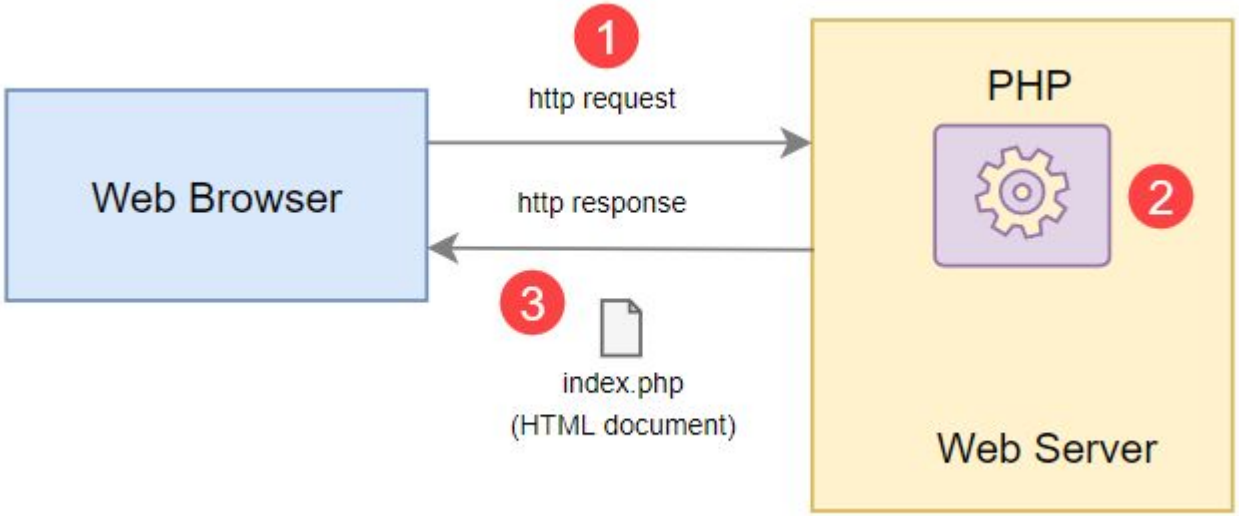
- LAMP: Linux + Apache + Mysql + PHP
- WAMP: Windows + Apache + Mysql + PHP
  - https://www.apachefriends.org/pt_br/download.html

- Rodar os programas online
  - https://paiza.io/

```
<?php
    echo "oi";
?>
```

PHP

- General-purpose language
- Origem: **P**ersonal **H**ome **P**age
- PHP: Hypertext Preprocessor
- Multi-paradigm: imperative, functional, procedural, reflective…



PHP

Tem também o paradigma POO

Programação Orientada a Objetos

Questão Problematizadora

Como modelar os problemas do mundo real combinando **estrutura** e **comportamento** em uma única entidade?

Programação Orientada a Objetos

- Sequencial
- Estruturado
- Orientado a objetos

Programação Orientada a Objetos

- Sequencial
  - Desvios incondicionais (GOTO e JUMP).
  - Solução rápida para problemas de pequeno porte.
  - Dificuldade em organizar o código.

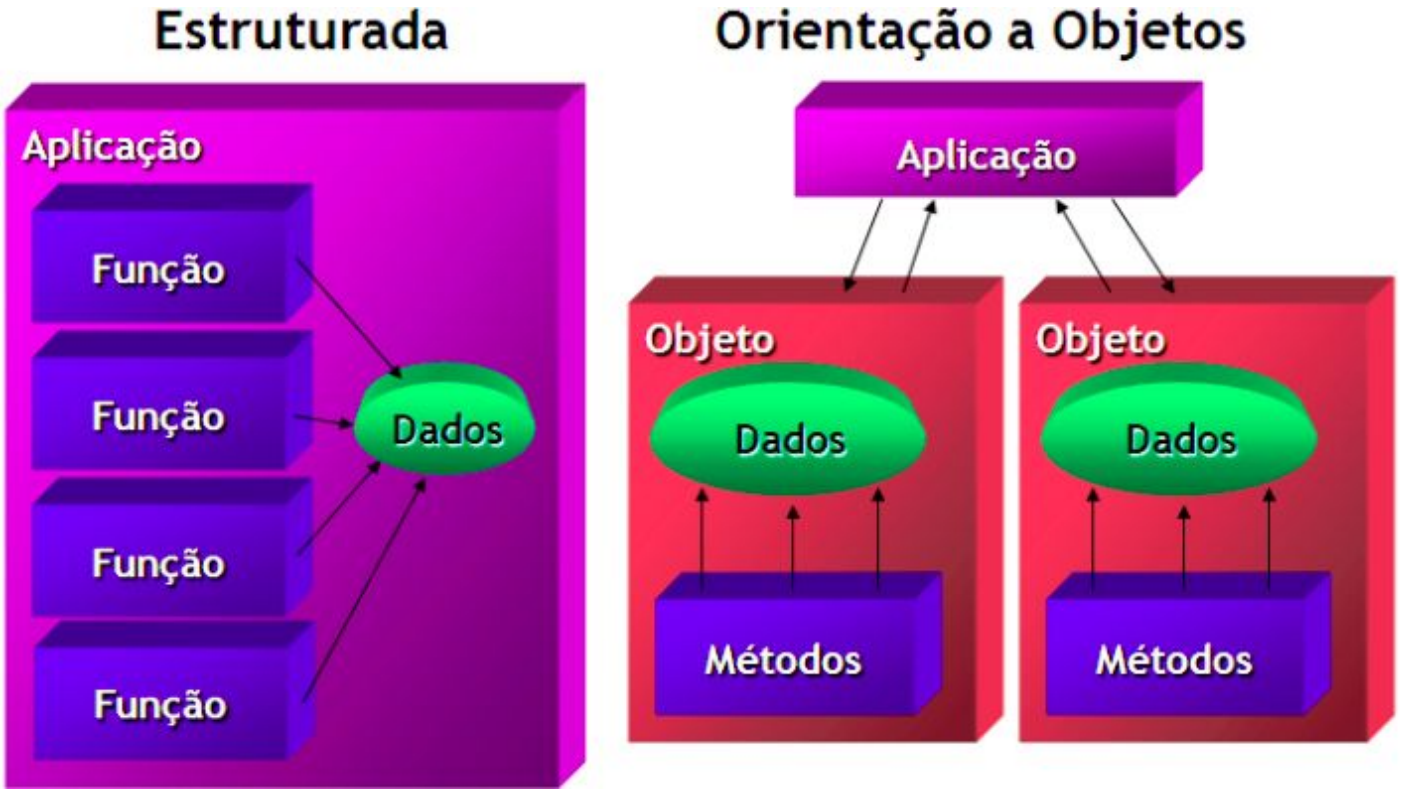
Programação Orientada a Objetos

- Estruturado
  - Dividir para conquistar: dividir o problema em partes menores
  - Estruturas
    - Sequência: Uma tarefa é executada após a outra, linearmente.
    - Decisão: Desvios condicionais.
    - Iteração: Repetição de trecho de código.
  - Função → solução para uma pequena parte
  - Vantagem → Execução mais rápida.
  - Desvantagens:
    - Códigos confusos: Dados misturados com comportamento.
    - Baixa reutilização de código.

Programação Orientada a Objetos

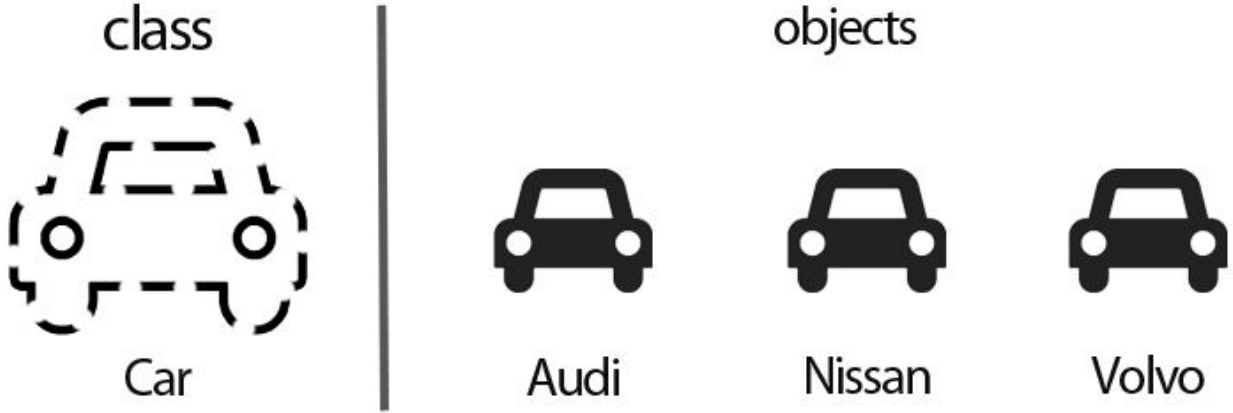
- Orientado a objetos
  - Todo sistema de software funcionasse como um ser vivo.
  - Cada célula do sistema poderia interagir com outras células, através do envio de mensagens e cada célula consistiria ainda em um sistema autônomo.
  - Células interconectadas, denominadas **objetos** → **comunicação entre os objetos** é possível realizar uma **tarefa computacional** completa.

Programação Orientada a Objetos

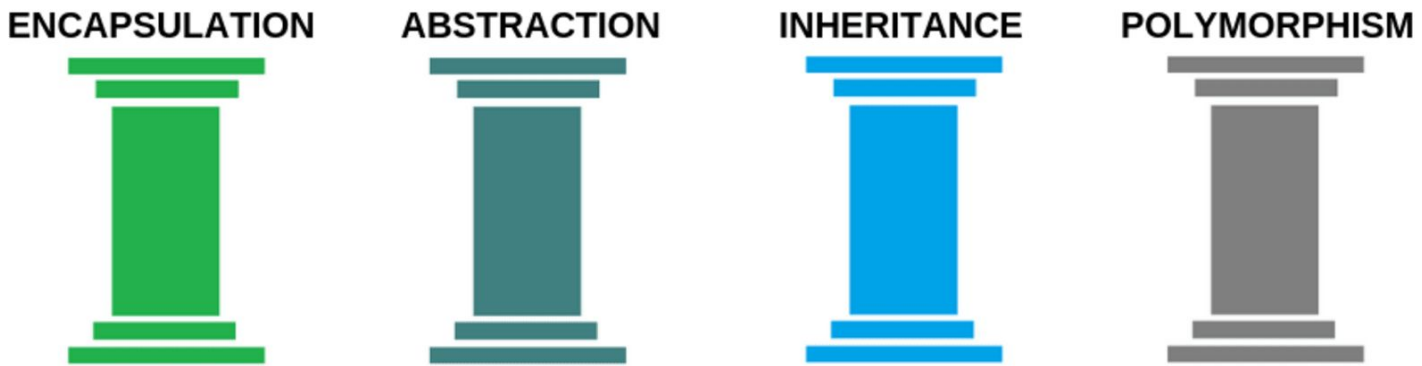


Programação Orientada a Objetos

- **Objetos**: Representação computacional de algo do mundo real.
- **Classe**: Definição de um bloco de construção básico. Um modelo ou planta (projeto) que indica como os objetos deverão ser construído.

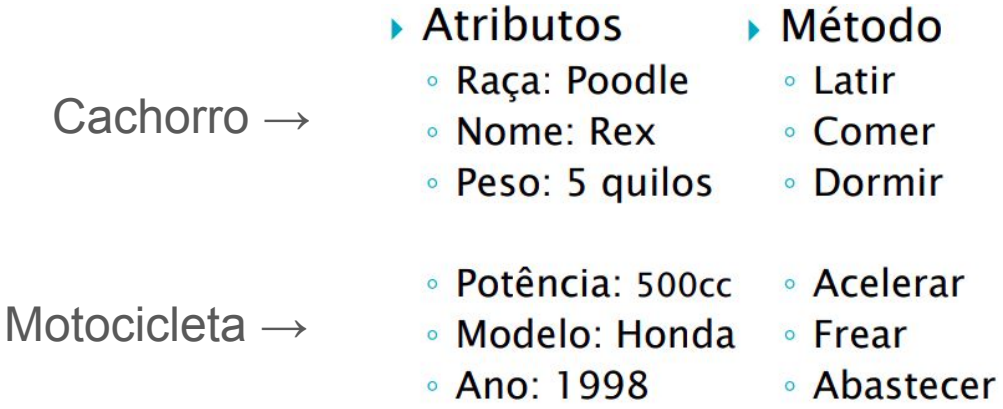


Programação Orientada a Objetos



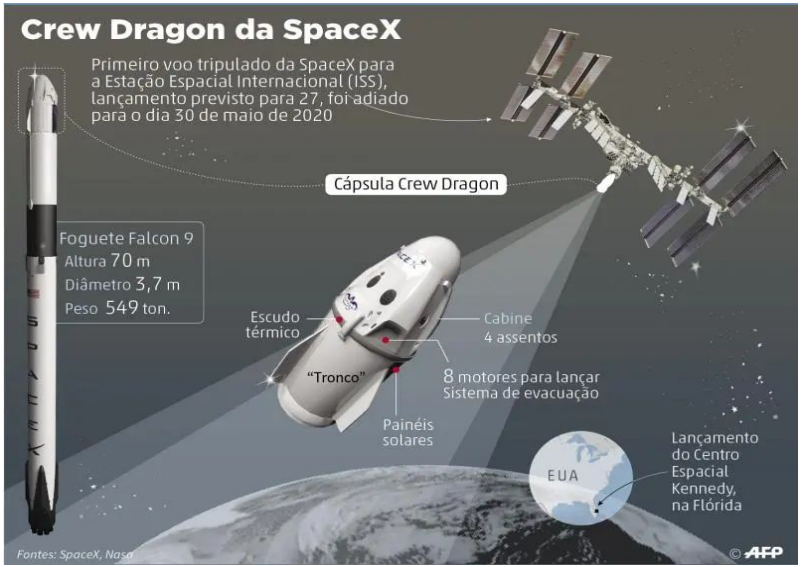
Programação Orientada a Objetos

- Abstração
  - Transformar aquilo que observamos **realidade** para a **virtualidade**
  - Objeto: Modelo → Características + Comportamento
  - Estado →Atributos (Características)
  - Operações → Métodos (Comportamentos)



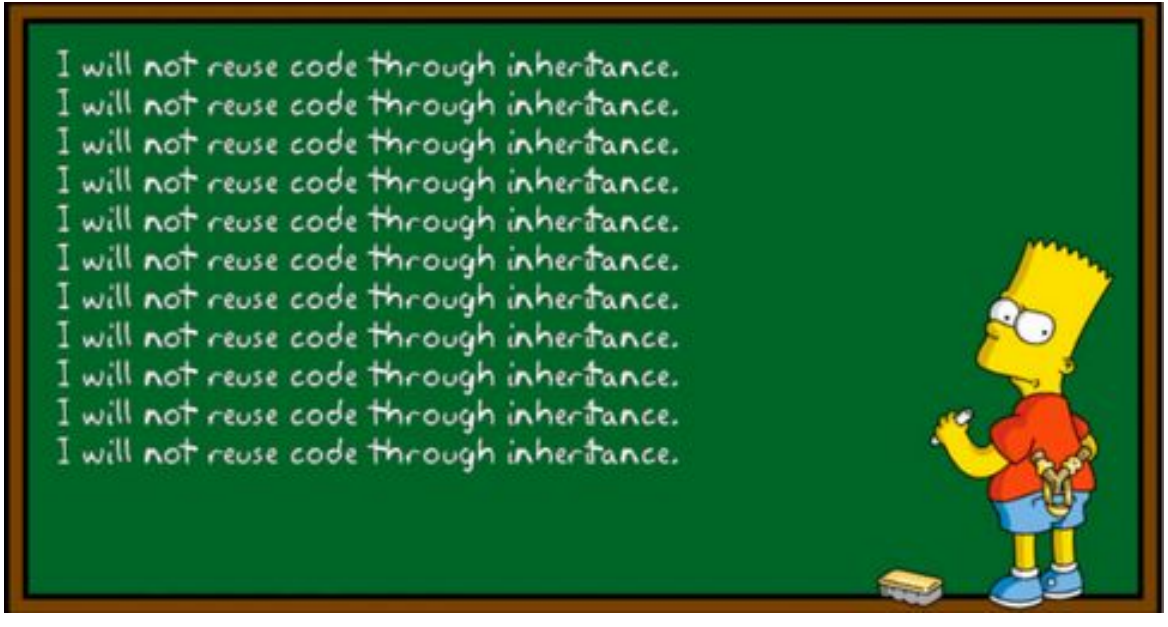
Programação Orientada a Objetos

- Encapsulamento
  - Um objeto “encapsula” todo o seu estado e o comportamento.
  - Os dados e as implementações são protegidos/escondidos.
  - Só importa o resultado.
  - Modificadores de visibilidade.



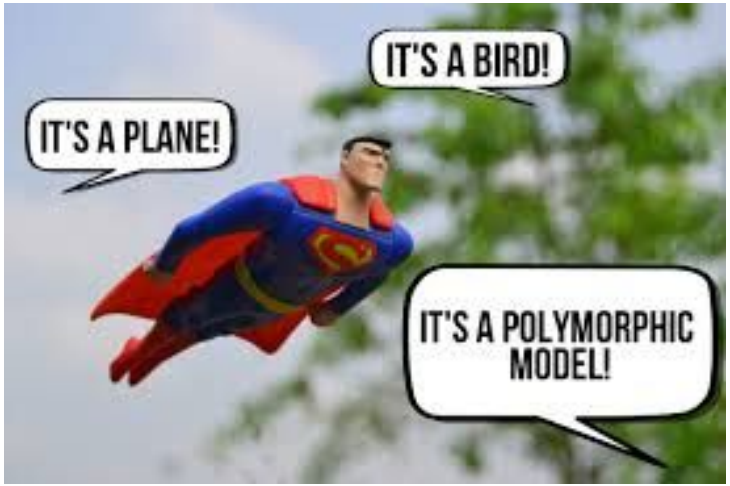
Programação Orientada a Objetos

- Herança
  - Criança → herda as características de seus pais.
  - O reuso de código!!!



Programação Orientada a Objetos

- Polimorfismo
  - Herdar as características e ações de “ancestrais” → **engessado**?
  - Capacidade de alterar sua forma conforme a necessidade.



PHP

- PHP objects → same way as references or handles
- Each variable contains an object reference (NOT a copy of the entire object)

PHP

- Classe

```
class <name> {

    ...

}
```

PHP

- Classe
- Propriedades → constantes, variáveis

```
class <name> {
        public $var = "Oi";
        ...


}
```

PHP

- Classe
- Propriedades → constantes, variáveis
- Métodos → funções

```
class <name> {
        public $var = "Oi";
        public function imprimirVar() {
                echo $this->var;
        }

}
```

PHP

- Construtor→executa quando um objeto é instanciado

```
class <name> {
        …
        public function __construct() {
                echo "Objeto instanciado!";
        }

}
```

PHP

1. Crie uma classe Calculadora que aceite como argumentos dois parâmetros e que tenha como métodos a adição, subtração, multiplicação.

```
$mycalc = new MyCalculator( 12, 6);

echo $mycalc->add(); // Mostra 18
```

PHP - Herança

- Classe herda constantes, métodos e propriedades: **extends**
- Há herança de apenas uma classe base em PHP
- Polimorfismo: métodos e propriedades podem ser sobrescritos (final?)

PHP - Herança

```php
class SimpleClass
{
    // property declaration
    public $var = 'a default value';

    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
```

```php
class ExtendClass extends SimpleClass
{
    // Redefine the parent method
    function displayVar()
    {
        echo "Extending class\n";
    }
}

$extended = new ExtendClass();
$extended->displayVar();
```

PHP - Herança

- Acessando a classe “pai” → parent::

```php
class SimpleClass
{
    // property declaration
    public $var = 'a default value';

    // method declaration
    public function displayVar() {
        echo $this->var;
    }
}
```

```php
class ExtendClass extends SimpleClass
{
    // Redefine the parent method
    function displayVar()
    {
        echo "Extending class\n";
        parent::displayVar();
    }
}
```

PHP - Herança

- Sobrescrita tem que ser compatível

```php
class Base
{
    public function foo(int $a = 5) {
        echo "Valid\n";
    }
}

class Extend extends Base
{
    function foo()
    {
        parent::foo(1);
    }
}
```

PHP - Herança

- Sempre validar se não há valores nulos
- Exemplo: acessar o nome do usuário com id igual a 5 no repositório
- Nullsafe Operator

```php
if (is_null($repository)) {
    $result = null;
} else {
    $user = $repository->getUser(5);
    if (is_null($user)) {
        $result = null;
    } else {
        $result = $user->name;
    }
}
```

PHP - Herança

- Sempre validar se não há valores nulos
- Exemplo: acessar o nome do usuário com id igual a 5 no repositório
- Nullsafe Operator

```php
if (is_null($repository)) {
    $result = null;
} else {
    $user = $repository->getUser(5);
    if (is_null($user)) {
        $result = null;
    } else {
        $result = $user->name;
    }
}
```

```php
$result = $repository?->getUser(5)?->name;
```

PHP - Propriedades

- Visibilidade (variáveis e métodos)
  - public (default)
  - protected (herdeiros podem acessar)
  - private (apenas na própria classe)

PHP - Propriedades

```php
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}
```

```php
$obj = new MyClass();
echo $obj->public;
echo $obj->protected;
echo $obj->private;
$obj->printHello();
```

PHP - Propriedades

- read-only: só é permitido inicializar uma vez

```php
class User {
    public readonly int $uid;

    public function __construct(int $uid) {
        $this->uid = $uid;
    }
}
```

PHP - Propriedades

- const: não é permitido alterar

```php
class MyClass
{
    const CONSTANT = 'constant value';

    function showConstant() {
        echo  self::CONSTANT . "\n";
    }
}

echo MyClass::CONSTANT . "\n";
```

PHP - Propriedades

- Operador **::**
  - Acesso a conteúdo estático, constante, e propriedades e métodos sobreescritos
  - Fora da classe:
    - <nome-da-classe>::<o que deseja acessar>
  - Dentro da classe:
    - self::<o que deseja acessar>

PHP - Propriedades

- Operador **::**
  - Acesso a conteúdo estático, constante, e propriedades e métodos sobreescritos
  - Fora da classe:
    - <nome-da-classe>::<o que deseja acessar>
  - Dentro da classe:
    - self::<o que deseja acessar>

```php
class OtherClass extends MyClass
{
    public static $my_static = 'static var';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}
```

```php
OtherClass::doubleColon();
```

PHP - Propriedades

- non-static properties: **->** (object operator)
  - $this->property
- Static properties: **::** (double colon)

PHP - Propriedades

- non-static properties: **->** (object operator)
  - $this->property
- Static properties: **::** (double colon)

```php
<?php
class Foo {
    public static function aStaticMethod() {
        // ...
    }
}

Foo::aStaticMethod();
$classname = 'Foo';
$classname::aStaticMethod();
?>
```

```php
class Foo
{
    public static $my_static = 'foo';

    public function staticValue() {
        return self::$my_static;
    }
}
```

PHP

2. Crie uma classe Animal que tenha a propriedade de tamanho e a funcionalidade de emitir som (string). Crie as classes Cachorro e Gato que sobrescreva a classe base. Instancie e teste todas as classes.
3. Crie uma nova calculadora que, além de herdar as funcionalidades da calculadora anterior, adicione a função n!.
4. Crie uma fun¸c˜ao que recebe um array chamado alunos que contenha o nome e duas notas (N1 e N2) de cada aluno e calcula a m´edia (N1+2*N2)/3 e retorna um array contendo o nome e a m´edia dos alunos aprovados. Seu script deve imprimir a lista dos aprovados em ordem alfabética com as médias finais.

PHP

5. Crie um script contendo as classes Automóvel, Motor, Rodas. Cada carro deve possuir um motor e quatro rodas. Utilizando as classes do exerc´ıcio anterior, crie a classe Caminhonete e Taxi, onde a primeira possui uma capacidade para carregar carga e a segunda pode levar até 4 passageiros.
6. Crie uma classe para processar um número indeterminado de argumentos. Crie um método para concatenar todos os elementos com uma string passada como parâmetro.