

Primeiros Passos em PHP

Parte IV

Thiago H. P. Silva

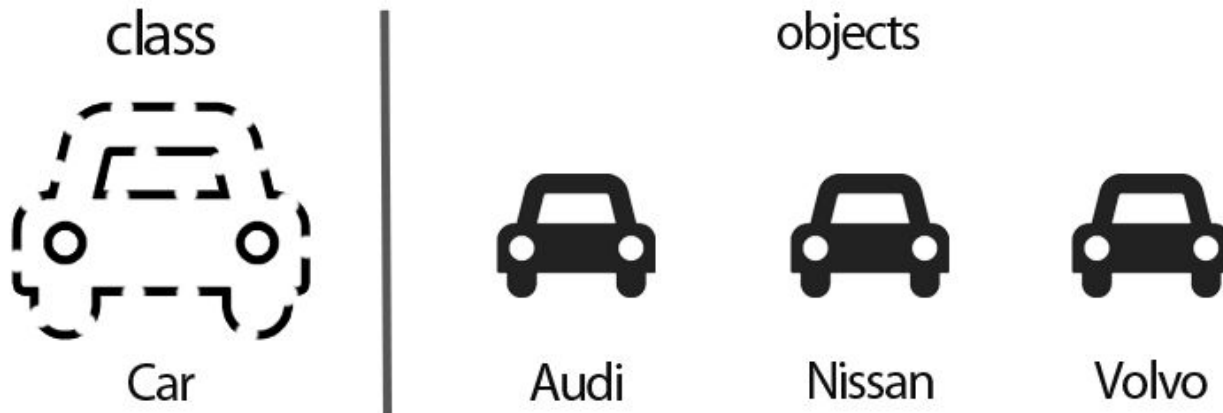
Execução

- 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";  
?>
```

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.



PHP

- Métodos estáticos

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

```
Foo::aStaticMethod();
```

PHP

- Propriedades Estáticas

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

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

PHP

- Propriedades Estáticas

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

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

```
print Foo::$my_static . "\n";
```

PHP

- Propriedades Estáticas

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

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

```
print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";
```

PHP

- Propriedades Estáticas

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

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

```
print Foo::$my_static . "\n";

$foo = new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static . "\n";
```

Undefined "Property" my_static

PHP

- Propriedades Estáticas

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

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

```
class Bar extends Foo
{
    public function fooStatic() {
        return parent::$my_static;
    }
}
```

PHP

- Classe Abstrata

PHP

- Classe Abstrata
 - Não pode ser instanciada
 - Modelos para as suas classes derivadas
 - Os métodos devem ser implementados pelo herdeiro (**obrigatoriamente?**)
 - Quando usar
 - Quando há um estreito relacionamento entre várias classes
 - Uma hierarquia de classes com muitos atributos/dados em comum
 - Quando não usar
 - Todas as classes podem ser instanciadas

PHP

- Classe Abstrata
 - Métodos abstratos: definem a assinatura (não definem a implementação)
 - Toda classe que possui pelo menos um método abstrato DEVE ser abstrata

PHP

- Classe Abstrata

- Métodos abstratos: definem a assinatura (não definem a implementação)
- Toda classe que possui pelo menos um método abstrato DEVE ser abstrata

```
abstract class AbstractClass
{
    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut() {
        print $this->getValue() . "\n";
    }
}
```

PHP

```
abstract class AbstractClass
{
    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut() {
        print $this->getValue() . "\n";
    }
}
```

```
class ConcreteClass1 extends AbstractClass
{
    protected function getValue() {
        return "ConcreteClass1";
    }

    public function prefixValue($prefix) {
        return "{$prefix}ConcreteClass1";
    }
}
```

PHP

- Interface

PHP

- Interface
 - Descreve de todas as funções que um objeto DEVE
 - Impõe certas propriedades em um objeto (classe)
 - **Contrato!**

PHP

- Interface

- Descreve de todas as funções que um objeto DEVE
- Impõe certas propriedades em um objeto (classe)
- **Contrato!**
- Vantagem: Facilita a criação de objetos de diferentes classes que podem ser usados de forma intercambiável
 - Exemplo: Serviços bancários que usam vários serviços de acesso a BD, vários gateways de pagamento ou diferentes estratégias de armazenamento em cache → Diferentes implementações podem ser trocadas sem exigir nenhuma alteração no código que as utiliza.

PHP

- Interface
 - Definido da mesma forma que uma classe: **interface**
 - Todos os métodos declarados em uma interface devem ser públicos

PHP

- Interface
 - Definido da mesma forma que uma classe: **interface**
 - Todos os métodos declarados em uma interface devem ser públicos
 - Vantagens
 - Para permitir que os desenvolvedores criem objetos de diferentes classes que podem ser usados de forma intercambiável
 - Exemplo: Serviços bancários que usam vários serviços de acesso a BD, vários gateways de pagamento ou diferentes estratégias de armazenamento em cache
→ Diferentes implementações podem ser trocadas sem exigir nenhuma alteração no código que as utiliza.

PHP

- Interface
 - É possível que as interfaces tenham constantes
 - As interfaces podem ser estendidas como classes usando o operador extends
 - Classes podem implementar mais de uma **interface**
 - Deve declarar todos os métodos na interface com uma **assinatura compatível**

PHP

```
// Declare the interface 'Template'
```

```
interface Template
{
    public function setVariable($name, $var);
    public function getHtml($template);
}
```

```
class WorkingTemplate implements Template
{
    private $vars = [];

    public function setVariable($name, $var)
    {
        $this->vars[$name] = $var;
    }

    public function getHtml($template)
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{ ' . $name . ' }', $value, $template);
        }

        return $template;
    }
}
```

PHP

```
interface A
{
    const B = 'Interface constant';
}
```

```
// Prints: Interface constant
echo A::B;
```

```
class B implements A
{
    const B = 'Class constant';
}
```

```
// Prints: Class constant
// Prior to PHP 8.1.0, this will however not work because it was not
// allowed to override constants.
echo B::B;
```

PHP

```
interface A
{
    public function foo(string $s): string;

    public function bar(int $i): int;
}
```

// An abstract class may implement only a portion of an interface.
// Classes that extend the abstract class must implement the rest.

```
abstract class B implements A
{
    public function foo(string $s): string
    {
        return $s . PHP_EOL;
    }
}
```

```
class C extends B
{
    public function bar(int $i): int
    {
        return $i * 2;
    }
}
```

PHP

- **Traits:** reutilização de código

PHP

- **Traits:** reutilização de código
 - **Micro interface??**
 - `LinkedList<String> lista;`



Parametrizando a classe LinkedList

PHP

- **Traits:** reutilização de código
 - **Micro interface??**

```
1 public interface HasComments<R extends HasComments<R>> {  
2  
3     // one method that parameterize the provided behaviour  
4     List<Comment> getComments();  
5  
6     // two methods that implement the behaviour  
7     default R add(Comment comment) {  
8         getComments().add(comment);  
9         return (R) this;  
10    }
```

PHP

- **Traits:** reutilização de código
 - Especificação de modelos que fornecem a capacidade de herdar de mais de uma (trait-)class → Pseudo herança múltipla
 - Reduzir algumas limitações da herança única → Reutilizar conjuntos de métodos livremente em várias classes independentes (diferentes hierarquias de classes)

PHP

```
class Base {  
    public function sayHello() {  
        echo 'Hello ';  
    }  
}
```

```
trait SayWorld {  
    public function sayHello() {  
        parent::sayHello();  
        echo 'World!';  
    }  
}
```

```
class MyHelloWorld extends Base {  
    use SayWorld;  
}
```

```
$o = new MyHelloWorld();  
$o->sayHello();
```

PHP

```
trait Hello {  
    public function sayHello() {  
        echo 'Hello ';  
    }  
}
```

```
trait World {  
    public function sayWorld() {  
        echo 'World';  
    }  
}
```

```
class MyHelloWorld {  
    use Hello, World;  
    public function sayExclamationMark() {  
        echo '!';  
    }  
}
```

```
$o = new MyHelloWorld();  
$o->sayHello();  
$o->sayWorld();  
$o->sayExclamationMark();
```

PHP

- Iterando objetos

```
class MyClass
{
    public $var1 = 'value 1';
    public $var2 = 'value 2';
    public $var3 = 'value 3';

    protected $protected = 'protected var';
    private $private = 'private var';

    function iterateVisible() {
        echo "MyClass::iterateVisible:\n";
        foreach ($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$class = new MyClass();
$class->iterateVisible();
```

PHP

- Iterando objetos

```
MyClass::iterateVisible:  
var1 => value 1  
var2 => value 2  
var3 => value 3  
protected => protected var  
private => private var
```

```
class MyClass  
{  
    public $var1 = 'value 1';  
    public $var2 = 'value 2';  
    public $var3 = 'value 3';  
  
    protected $protected = 'protected var';  
    private $private = 'private var';  
  
    function iterateVisible() {  
        echo "MyClass::iterateVisible:\n";  
        foreach ($this as $key => $value) {  
            print "$key => $value\n";  
        }  
    }  
}  
  
$class = new MyClass();  
$class->iterateVisible();
```

PHP

- Iterando objetos

```
$class = new MyClass();

foreach($class as $key => $value) {
    print "$key => $value\n";
}
```

```
class MyClass
{
    public $var1 = 'value 1';
    public $var2 = 'value 2';
    public $var3 = 'value 3';

    protected $protected = 'protected var';
    private $private = 'private var';

    function iterateVisible() {
        echo "MyClass::iterateVisible:\n";
        foreach ($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$class = new MyClass();
$class->iterateVisible();
```


PHP

- Iterando objetos

```
$class = new MyClass();

foreach($class as $key => $value) {
    print "$key => $value\n";
}
```

```
var1 => value 1
var2 => value 2
var3 => value 3
```

```
class MyClass
{
    public $var1 = 'value 1';
    public $var2 = 'value 2';
    public $var3 = 'value 3';

    protected $protected = 'protected var';
    private $private = 'private var';

    function iterateVisible() {
        echo "MyClass::iterateVisible:\n";
        foreach ($this as $key => $value) {
            print "$key => $value\n";
        }
    }
}

$class = new MyClass();
$class->iterateVisible();
```

PHP

- Final methods

```
class BaseClass {  
    public function test() {  
        echo "BaseClass::test() called\n";  
    }  
  
    final public function moreTesting() {  
        echo "BaseClass::moreTesting() called\n";  
    }  
}
```

Fatal Error



```
class ChildClass extends BaseClass {  
    public function moreTesting() {  
        echo "ChildClass::moreTesting() called\n";  
    }  
}
```

PHP

- Final methods

```
final class BaseClass {  
    public function test() {  
        echo "BaseClass::test() called\n";  
    }  
  
    // As the class is already final, the final keyword is redundant  
    final public function moreTesting() {  
        echo "BaseClass::moreTesting() called\n";  
    }  
}
```

Fatal Error



```
class ChildClass extends BaseClass {  
}
```

PHP

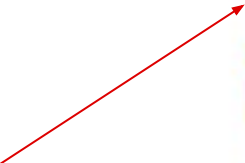
- Enumerations

```
enum Suit
{
    case Hearts;
    case Diamonds;
    case Clubs;
    case Spades;
}
```

PHP

- Enumerations

```
enum Suit
{
    case Hearts;
    case Diamonds;
    case Clubs;
    case Spades;
}
```

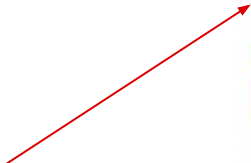


```
$val = Suit::Diamonds;
```

PHP

- Enumerations

```
enum Suit
{
    case Hearts;
    case Diamonds;
    case Clubs;
    case Spades;
}
```



```
$val = Suit::Diamonds;
```

```
Suit::cases();  
// Produces: [Suit::Hearts, Suit::Diamonds, Suit::Clubs, Suit:Spades]
```

PHP

- Enumerations

```
enum Suit: string
{
    case Hearts = 'H';
    case Diamonds = 'D';
    case Clubs = 'C';
    case Spades = 'S';
}
```

PHP

- Enumerations

```
enum Suit: string
{
    case Hearts = 'H';
    case Diamonds = 'D';
    case Clubs = 'C';
    case Spades = 'S';
}

print Suit::Clubs->value;
```


PHP

- Enumerations

```
interface Colorful
{
    public function color(): string;
}

enum Suit: string implements Colorful
{
    case Hearts = 'H';
    case Diamonds = 'D';
    case Clubs = 'C';
    case Spades = 'S';

    // Fulfills the interface contract.
    public function color(): string
    {
        return match($this) {
            Suit::Hearts, Suit::Diamonds => 'Red',
            Suit::Clubs, Suit::Spades => 'Black',
        };
    }
}
```

PHP

- Enumerations

```
enum Size
{
    case Small;
    case Medium;
    case Large;

    public static function fromLength(int $cm): static
    {
        return match(true) {
            $cm < 50 => static::Small,
            $cm < 100 => static::Medium,
            default => static::Large,
        };
    }
}
```

PHP

1. Defina dois números inteiros a e b. Em seguida, imprima uma lista de todos os números inteiros entre eles.
 - Inteiros -5 e 6. Saída: [-4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
 - Inteiros 1 e 2. Saída: []
2. Crie uma que receba um inteiro >1 e exiba um padrão de triângulo conforme o exemplo abaixo:
 - Inteiro igual a 5
 - Saída:

```
1
12
123
1234
12345
```

PHP

3. Dada uma string e um inteiro. Caso o tamanho da string seja divisível por n, então a divida em n partes.
- String: “a123b123”, inteiro: 2
 - Saída:
a123
b123
 - String: “abc”, inteiro: 3
 - Saída:
a
b
c

PHP

4. Dado uma string, realize uma compressão dos dados contando o número de caracteres repetidos.
- String: qqwwweeeerrrrtyy
 - Saída: q2w3e4r5ty2
5. Escreva um programa que gere a série de Fibonacci até um dado número n
- N igual a 5
 - Saída:
0
1
1
2
3

$$F_0 = 0 \text{ e } F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}$$