Este curso é de propriedade do aplicativo SoloLearn e foi traduzido por Ana Beatriz Augusto usando os recursos Reverso Context e Google Tradutor.

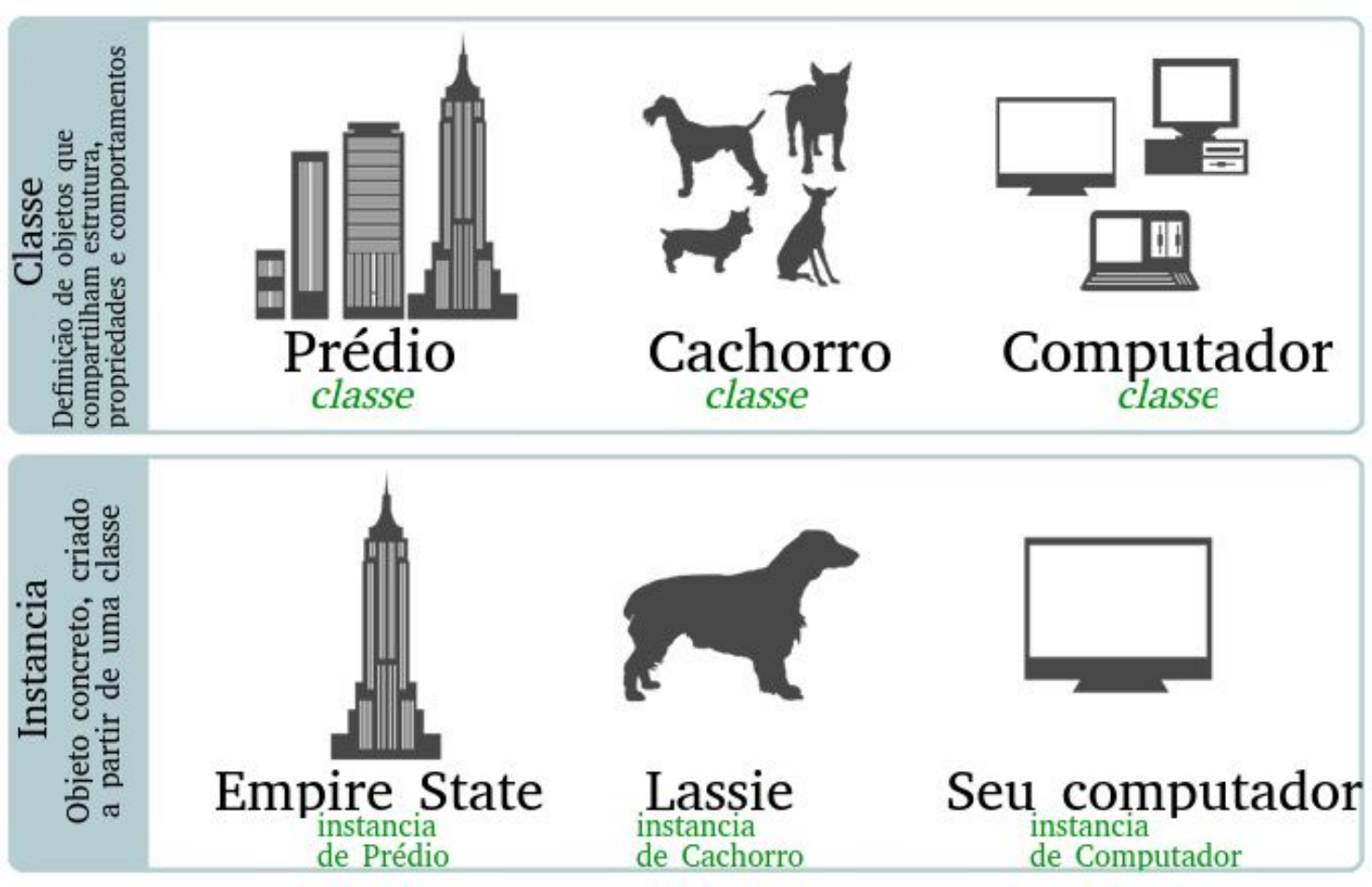
# Classes e objetos

Programação Orientada a Objetos (POO) é um estilo de programação que tem a intenção de deixar o mundo real e a programação mais próximos.

Objetos são criados usando classes, que são o foco da POO.
A classe descreve o que o objeto será, mas é separada do objeto em si.

Em outras palavras, uma classe pode ser pensada como o projeto do objeto, descrição ou definição.

# Veja os seguintes exemplos:



Aqui, Prédio é uma classe. Define as características de um prédio genérico e como deve se comportar.
O prédio Empire States é um objeto específico (ou instância) dessa classe.

Você pode usar a mesma classe como um projeto para criar múltiplos objetos diferentes.

Em PHP, uma classe pode incluir variáveis chamadas de propriedades para definir as características de um objeto e funções, chamadas de métodos, para definir o comportamento de um objeto. A definição de uma classe começa com a palavra class, seguida pelo nome da classe. Chaves envolvem as definições das propriedades e métodos que pertencem a classe.
Por exemplo:

```php
class Pessoa {
  public $idade; //propriedade
  public function falar() { //método
    echo "Oi!"
  }
}
```

O código da página anterior define uma classe `Pessoa` que inclui uma propriedade de idade e um método `falar()`

Um nome válido para classe começa com uma letra ou underline, seguido por qualquer número de letras, números ou underlines.

Note a palavra `public` em frente ao método falar; é um especificador de visibilidade.
A palavra `public` específica se o componente pode ser acessado de qualquer lugar no código.
Há outras palavras que definem visibilidade e vamos aprender sobre elas nas próximas páginas.

O processo de criação de um objeto é chamado de instanciação. Para instanciar um objeto de uma classe, use a palavra `new`, como no exemplo abaixo:

```php
$bob = new Pessoa();
```

No código acima, `$bob` é um objeto da classe `Pessoa`.

Para acessar as propriedades e métodos de um objeto, use a seta `->`, como em:

```php
echo $bob->idade;
```

Essa afirmação mostra o valor da propriedade idade para `$bob`. Se você quer atribuir um valor a uma propriedade use o operador `=` como você usaria com qualquer variável.

Vamos definir a classe `Pessoa`, instanciar um objeto, fazer uma atribuição e chamar o método `falar()`:

```php
class Pessoa {
  public $idade;
  function falar() {
    echo "Oi!";
  }
}
$p1 = new Pessoa(); //instância um objeto
$p1->idade = 23; // atribuição
echo $p1->idade; // 23
$p1->falar(); // Oi!
```

`$this` é uma pseudo variável que é uma referência ao objeto que está sendo chamado. Quando se trabalha dentro de um método, use `$this` da mesma maneira que usaria um nome de objeto fora da classe.

Por exemplo:

```php
class Cachorro {
  public $pernas = 4;
  public function mostre() {
    echo $this->pernas;
  }
}
```

```php
$d1 = new Cachorro();
$d1->mostre(); //4
```

```php
$d2 = new Cachorro();
$d2->pernas = 2;
$d2->mostre(); //2
```

Criamos dois objetos da classe `Cachorro` e chamamos seus métodos `mostre()`. Pois, o método `mostre()` usa `$this`, o valor de `pernas` referenciado ao valor da propriedade do objeto que foi chamado.

Como pode ver, cada objeto pode ter seus próprios valores para as propriedades da classe.

## Construtor e destrutor

PHP fornece o mágico método construtor `__construct()`, que é chamado automaticamente quando um novo objeto é instanciado. Por exemplo:

```
class Pessoa {
  public function __construct() {
    echo "Objeto criado";
  }
}
$p = new Pessoa();
```

O método `__construct()` é frequentemente usado para qualquer inicialização que o objeto talvez precise antes de ser usado. Parâmetros podem ser incluídos no `__construct()` para aceitar valores quando o objeto é criado.

No código ao lado, o construtor usa argumentos para inicializar propriedades da classe correspondente.

Por exemplo:

```php
class Pessoa {
  public $nome;
  public $idade;
  public function
__construct($n, $i) {
    $this->nome = $n;
    $this->idade = $i;
  }
}
$p = new Pessoa("David",
42);
```

Você não pode escrever múltiplos métodos `__construct()` com diferente números de parâmetros. Um comportamento diferente deve ser lidado com lógica dentro de um único método `__construct()`

Similar ao construtor, há um destruidor método mágico `__destruct()`, que é automaticamente chamado quando um objeto é destruído.

Por exemplo:

```
class Pessoa {
  public function __destruct() {
    echo "Objeto destruido";
  }
}
$p = new Pessoa();
```

O código da página anterior cria um novo objeto da classe `Pessoa`. Quando o script termina, o objeto é automaticamente destruído, que chama o destruidor e mostra a mensagem "Objeto destruído". Para explicitamente disparar o destruidor, você pode destruir o objeto usando a função `unset()`, similar a: `unset($p)`

Destruidores são úteis para performar certas tarefas quando o objeto termina seu ciclo de vida. Por exemplo, libera recursos, escreve arquivos de registro, fecha uma conexão com o banco de dados e assim em diante.
PHP libera todos os recursos quando um script termina sua execução.

# Herança de classe

Classes podem herdar os métodos e propriedades de outra classe. A classe que herda os métodos e propriedades é chamada de subclasse. A classe que herdou é chamada de super classe.

A herança pode ser feita usando a palavra `extends`.
Por exemplo:

```php
class Animal {
  public $nome;
  public function oi() {
    echo "Oi de Animal";
  }
}
```

```php
class Cachorro extends Animal{
}

$d = new Cachorro();
$d->oi();
```

A classe `Cachorro` herda da classe `Animal`. Como você pode ver, todas as propriedades e métodos de `Animal` são acessíveis por objetos da classe `Cachorro`.

Construtores da super classe não são chamados implicitamente se a subclasse define um construtor. Entretanto, se a subclasse não define um construtor, então será herdado da super classe se não for declarado privado.

Note que todas propriedades e métodos tem visibilidade pública. Para controlar objetos, declare métodos e propriedades com palavras de visibilidade.

Visibilidade controla como e de onde as propriedades e métodos podem ser acessados.
Até aqui, usamos a palavra `public` para especificar que uma propriedade ou método é acessível de qualquer lugar.
Há mais duas palavras que declaram visibilidade:
`protected`: Faz com que propriedades e métodos sejam acessíveis apenas dentro da classe, por herança e por super classes.
`private`: Faz com que propriedades e métodos sejam acessíveis apenas pela classe que a definiu

As propriedades devem sempre ter um tipo de visibilidade. Métodos declarados sem qualquer palavra que define visibilidade, são definidos como públicos.

Uma interface especifica uma lista de métodos que uma classe deve implementar. Entretanto, a interface não contém implementações de métodos. Esse é um aspecto importante de interfaces pois, permite que um método seja lidado de maneira diferente em cada classe que usa a interface.

A palavra `interface` define uma interface.
A palavra `implements` é usada em uma classe para implementar uma interface.

Por exemplo, `AnimalInterface` é definida com uma declaração para a função `fazerSom()`, mas essa função não é implementada até ser usada em uma classe:

```php
<?php
interface AnimalInterface{
  public function fazerSom();
}
class Cachorro implements
AnimalInterface {
  public function fazerSom(){
    echo "Au! <br />";
  }
}
```

```php
class Gato implements
AnimalInterface {
  public function
fazerSom() {
    echo "Miau! <br />";
  }
}
$myObj1 = new Cachorro();
$myObj1->fazerSom();
$myObj2 = new Gato();
$myObj2->fazerSom();
?>
```

Uma classe pode implementar múltiplas interfaces.
Mais que uma interface pode ser especificada sendo separadas por vírgulas. Por exemplo:

```
class Demo implements AInterface, BInterface,
CInterface {
  // Funções devem ser definidas aqui
}
```

Uma interface pode herdar outra interface usando a palavra `extends`.

Todos os métodos especificados em uma interface requerem visibilidade pública.

Classes abstratas podem ser herdadas mas não podem ser instanciadas.
Elas oferecem a vantagem de serem capazes de conter métodos com definições e métodos abstratos que não são definidos até serem herdados.

A classe que herda de uma classe abstrata deve implementar todos os método abstratos.

A palavra `abstract` é usada para criar uma classe abstrata ou um método abstrato.

Por exemplo:

```php
<?php
abstract class Fruta{
  private $cor;
  abstract public function
comer();
  public function setCor($c){
    $this->cor = $c;
  }
}
```

```php
class Maca extends
Fruta{
  public function
comer(){
    echo "Comendo...";
  }
}
$obj = new Maca();
$obj->comer();
?>
```

Funções abstratas apenas podem aparecer em uma classe abstrata

# static e final

A palavra `static` no PHP, define propriedades estáticas e métodos estáticos. Uma propriedade estática/um método estático de uma classe pode ser acessado sem a criação de um objeto dessa classe.

Uma propriedade estática ou método é acessado usando o operador de resolução do escopo `::` entre o nome da classe e o nome da propriedade/método.

Por exemplo:

```php
<?php
class minhaClasse {
  static
$minhaPropriedadeEstatica =
42;
}
echo
minhaClasse::$minhaPropried
adeEstatica;
?>
```

A palavra `self` é necessária para acessar uma propriedade estática de um método estático em uma definição de classe.

Por exemplo:

```php
<?php
class minhaClasse {
  static $minhaPropriedade = 42;
  static function meuMetodo() {
    echo self::$minhaPropriedade;
  }
}
minhaClasse::meuMetodo();
?>
```

Objetos de uma classe não podem acessar propriedades estáticas na classe mas eles podem acessar métodos estáticos.

A palavra final no PHP define métodos que não podem ser sobrescritos em subclasses.
Classes que são definidas final não podem ser herdadas.

Esse exemplo demonstra que um método final não pode ser sobrescrito em uma classe filha:

```php
<?php
class minhaClasse {
  final function minhaFuncao() {
    echo "Mae";
  }
}
// Erro pois, um método final não pode ser sobrescrito em subclasses.
class minhaClasse2 extends minhaClasse {
  function minhaFuncao() {
    echo "Filha";
  }
}
?>
```

O seguinte código demonstra que uma classe `final` não pode ser herdada:

```php
<?php
final class minhaClasseFinal {
}
// Erro pois, uma classe final não pode ser herdada.
class minhaClasse extends minhaClasseFinal {
}
?>
```

Diferente das classes e métodos, as propriedades não podem ser marcadas como `final`.

Este curso é de propriedade do aplicativo SoloLearn e foi traduzido por Ana Beatriz Augusto usando os recursos Reverso Context e Google Tradutor.