



---

---


---

---


---

---

---

Introdução a Orientação a Objetos 

- Problemas da Programação Procedural
  - Descentralização (repetição) de código
  - Dificuldade de manutenção
  - Dificuldade em substituir desenvolvedores
  - Muitas pessoas responsáveis por mesmos códigos em diferentes partes do projeto
  - Pouco reaproveitamento de código



---

---


---

---


---

---

---

Introdução a Orientação a Objetos 

- Orientação a Objetos
  - Resolve os problemas apresentados da Programação Procedural
  - Concentra responsabilidades nos locais certos
  - Flexibiliza a aplicação
  - Encapsula a lógica de negócio
  - Melhora a comunicação entre desenvolvedores
  - Aumenta a reutilização de código
  - Aumenta o ciclo de vida dos projetos
  - Menor custo de desenvolvimento e criação



---

---

---

---

---

---

---

## Orientação a Objetos



- Siglas
  - OO: Orientação a Objetos
  - LOO: Linguagem Orientada a Objetos
  - POO: Programação Orientada a Objetos
- Características
  - Códigos pertencem a classes
  - Classes possuem propriedades próprias
  - Classes possuem funcionalidades próprias
  - Classes interagem com outras classes

---

---

---

---

---

---

---

## Como funciona (visão geral)



### Modelo Procedural

nome  
sobrenome  
velocidade  
endereco  
anos

INSERT INTO USUARIOS...

INSERT INTO CARROS...

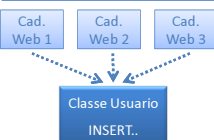


### Modelo OO

usuario->nome  
usuario->sobrenome  
carro->velocidade  
carro->endereco  
usuario->anos

usuario->salvar()

carro->salvar()




---

---

---

---

---

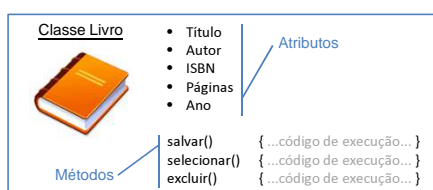
---

---

## Classes



- Estrutura de dados
- Representa um tipo de dado
- Comportamento próprio




---

---

---

---

---

---

---

## Atributos e Métodos



- **Atributos**
  - Características da classe
  - Tipos primitivos e/ou outras classes
  - Geralmente representados por **substantivos**
  - Exemplos: *nome, idade, endereço, cor, tamanho*
- **Métodos**
  - Ações que a classe pode realizar
  - Podem receber parâmetros e retornar valores
  - Geralmente representados por **verbos**
  - Exemplos: *salvar, ler, abrir, fechar, emprestar, devolver*

---

---

---

---

---

---

---

---

## Criando classes



- Operador **class**

```
class NomeDaClasse {
    /* Atributos */
    /* Métodos */
}
```

```
class Usuario {
    /* Atributos */
    public $nome;
    public $idade;
    /* Métodos */
    public function apresentar() {
        echo "Meu nome é " . $this->nome;
    }
}
```

---

---

---

---

---

---

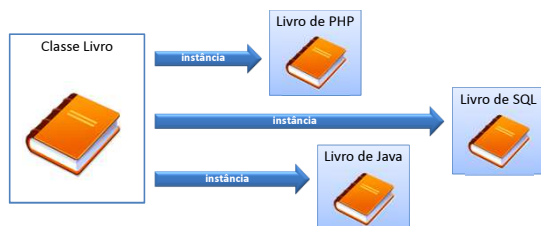
---

---

## Objetos



- Classe não é um objeto
- Classe é um modelo de objeto (template)
- Objetos são instâncias de classes




---

---

---

---

---

---

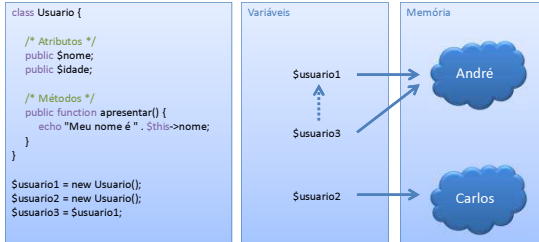
---

---

## Criando objetos



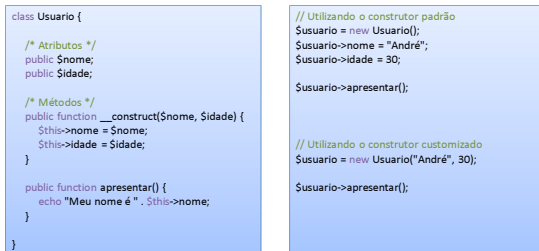
- Operador **new**
- Cria uma instância a partir de uma classe



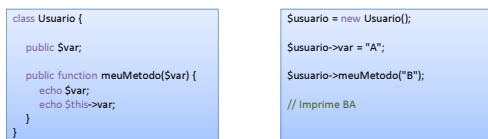
## Construtores



- Método **\_\_construct()**
- Customiza o código de inicialização da classe

Operador **\$this**

- Diferencia um atributo do objeto de um atributo do método
- Fornece a referência do próprio objeto para outro método
- Funciona somente dentro do próprio objeto



## Modificadores de acesso



- Definem o acesso aos atributos e métodos
  - private**: acesso apenas para a própria classe
  - public**: acesso público

```
class Modificadores {
    public $publicVar;
    private $privateVar;

    public function publicMethod() {
        echo $this->privateVar;
    }

    private function privateMethod() {
        echo "Hello from privateMethod";
    }
}
```

```
$meuObj = new Modificadores();

echo $meuObj->publicVar; // Imprime A
echo $meuObj->publicMethod(); // Imprime B
echo $meuObj->privateVar; // Fatal Error
echo $meuObj->privateMethod(); // Fatal Error
```

## Forma de utilização



- Geralmente utilizados da seguinte forma:
  - Atributos são declarados como **private**
  - Métodos são declarados como **public**
  - Métodos auxiliares são declarados como **private**

## Exemplo



Informação interna do carro

Forma de interação

Função interna do carro

```
class Carro {
    private $tanqueGasolina;

    public function acelerar() {
        $this->injetarGasolinaMotor();
        $this->ativarQueimaDaVela();
    }

    private function injetarGasolinaMotor() {
        // Código de execução
    }

    private function ativarQueimaDaVela() {
        // Código de execução
    }
}
```

## Getters e Setters



- Permitem interagir com os atributos
- Segurança: validações no lugar certo

```
class Carro {
    private $tanqueGasolina;

    public function setGasolina($tanqueGasolina) {
        // Validações
        $this->tanqueGasolina = $tanqueGasolina;
    }

    public function getGasolina() {
        return $this->tanqueGasolina;
    }
}
```

---

---

---

---

---

---

---

---

## Getters e Setters booleanos



- **set** para definir valor
- **is** para consultar, ao invés de get

```
class Carro {
    private $carroLigado;

    public function setCarroLigado($carroLigado) {
        // Validações
        $this->carroLigado = $carroLigado;
    }

    public function isCarroLigado() {
        return $this->carroLigado;
    }
}
```

---

---

---

---

---

---

---

---

## Aulas práticas e manuais on-line



Assista agora as aulas práticas.

[Clique aqui](#) para visualizar as aulas práticas disponíveis

---

---

---

---

---

---

---

---