DESENVOLVIMENTO DE SISTEMAS

PARADIGMAS DE PROGRAMAÇÃO, ORIENTAÇÃO A OBJETOS E PHPH

Prof. André Aparecido da Silva
Aula sobre PHP e orientação a objetos
Disponível em: https://www.oxnar.com.br/aulas/php

ORIENTAÇÃO A OBJETOS

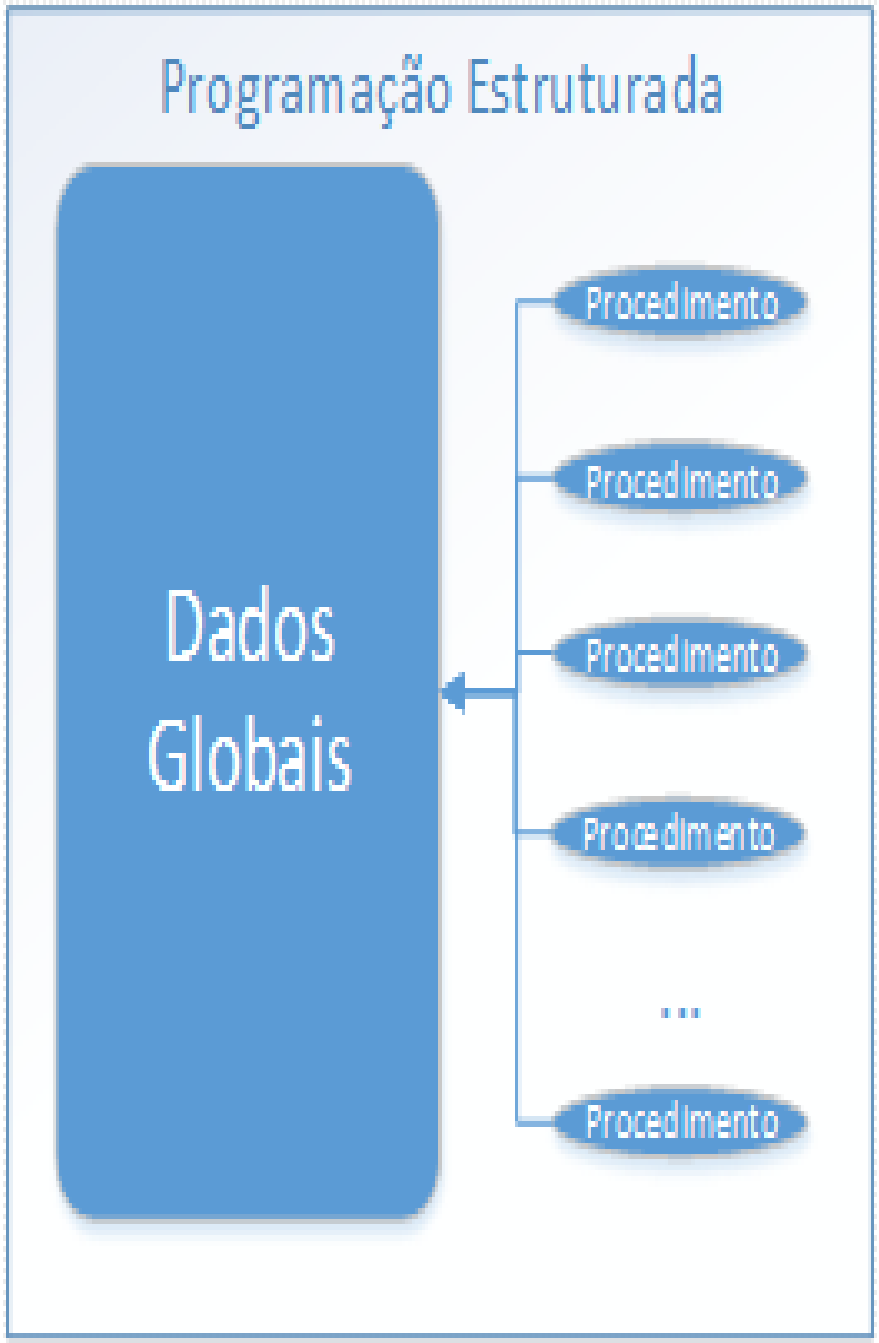
Orientação a Objetos (OO) é um paradigma de programação que organiza o código ao redor de **objetos**, em vez de ações ou lógica funcional.

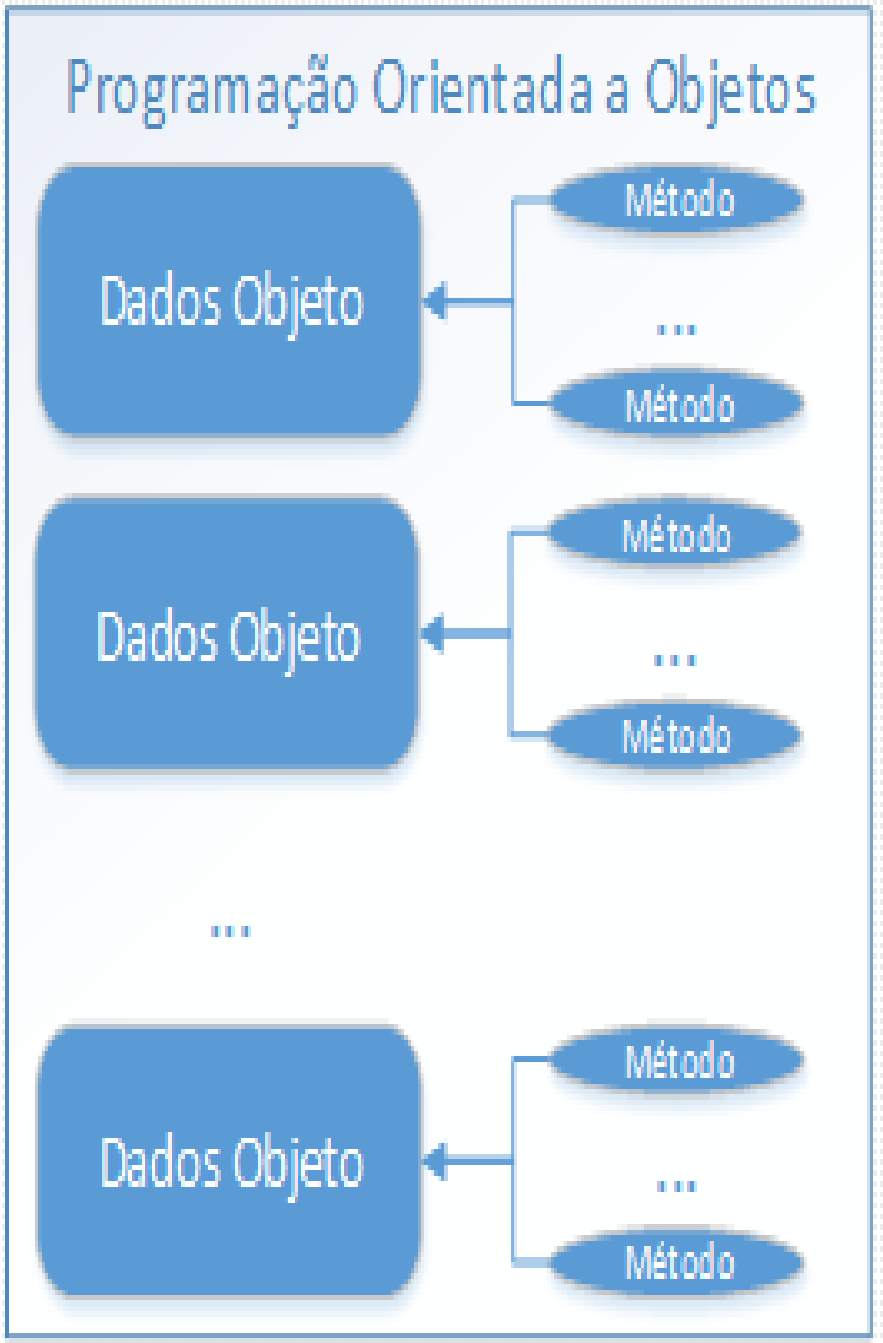
OUTRA DEFINIÇÃO

Programação orientada a objetos (POO) é um paradigma de programação que organiza o código em torno de objetos, que são instâncias de classes. Esses objetos possuem atributos (dados) e métodos (comportamentos) que podem interagir entre si.

Esses objetos representam entidades do mundo real ou conceitos, com suas **características** (atributos) e **comportamentos** (métodos). A ideia é modelar o software de forma mais próxima de como pensamos as coisas no dia a dia..

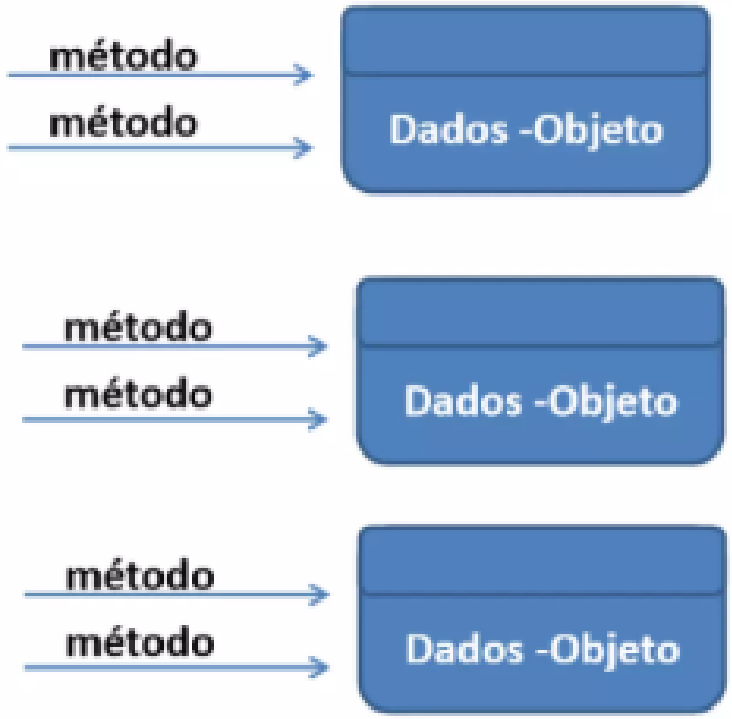
COMPARAÇÃO ENTRE PARADIGMAS





COMPARAÇÃO ENTRE PARADIGMAS

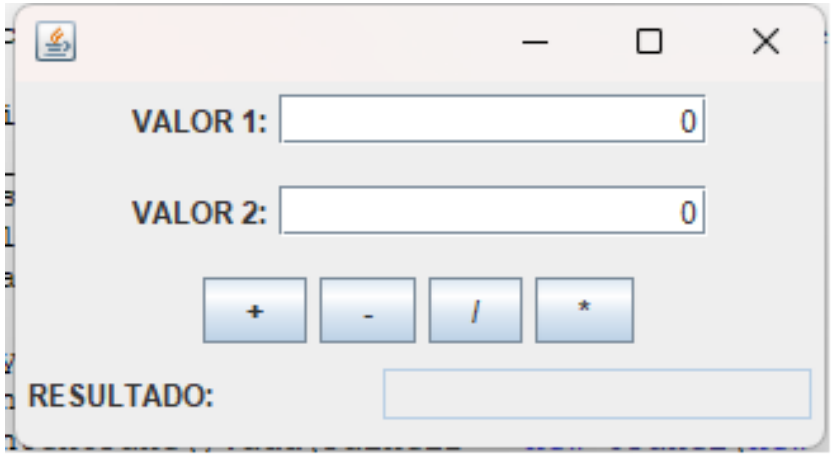
Paradigma
Estrutural x Orientação a Objeto



EXEMPLO DE OO EM OUTRA LINGUAGEM

```
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.*;

public class Tela_Calc10 extends JFrame implements ActionListener
{
        JButton bt_Mais, bt_Menos, bt_Dividir, bt_Vezes;
        JTextField tf_Valor1, tf_Valor2, tf_Resultado;
        JLabel lbl_Resultado;
        JPanel Painel1, Painel2, Painel3, Painel4;
        public Tela_Calc10 ()
        {
                setLayout(new GridLayout(4, 1));
                getContentPane().add(Painel1 = new JPanel(new FlowLayout()));
                getContentPane().add(Painel2 = new JPanel(new FlowLayout()));
                getContentPane().add(Painel3 = new JPanel(new FlowLayout()));
                getContentPane().add(Painel4 = new JPanel(new FlowLayout()));

                Painel1.add(new JLabel ("VALOR 1:"));
                Painel1.add(tf_Valor1 = new JTextField("0", 15));
                tf_Valor1.setHorizontalAlignment(tf_Valor1.RIGHT); // Alinhamento a direita

                Painel2.add(new JLabel ("VALOR 2:"));
                Painel2.add(tf_Valor2 = new JTextField("0", 15));
                tf_Valor2.setHorizontalAlignment(tf_Valor2.RIGHT); // Alinhamento a direita

                Painel3.add(bt_Mais = new JButton("+"));
                Painel3.add(bt_Menos  = new JButton("-"));
                Painel3.add(bt_Dividir = new JButton("/"));
                Painel3.add(bt_Vezes = new JButton("*"));

                //
                Painel4.add(new JLabel("RESULTADO:"));
```

EXEMPLO DE OO EM OUTRA LINGUAGEM



OUTROS PARADIGMAS DE PROGRAMAÇÃO

1. Programação Imperativa

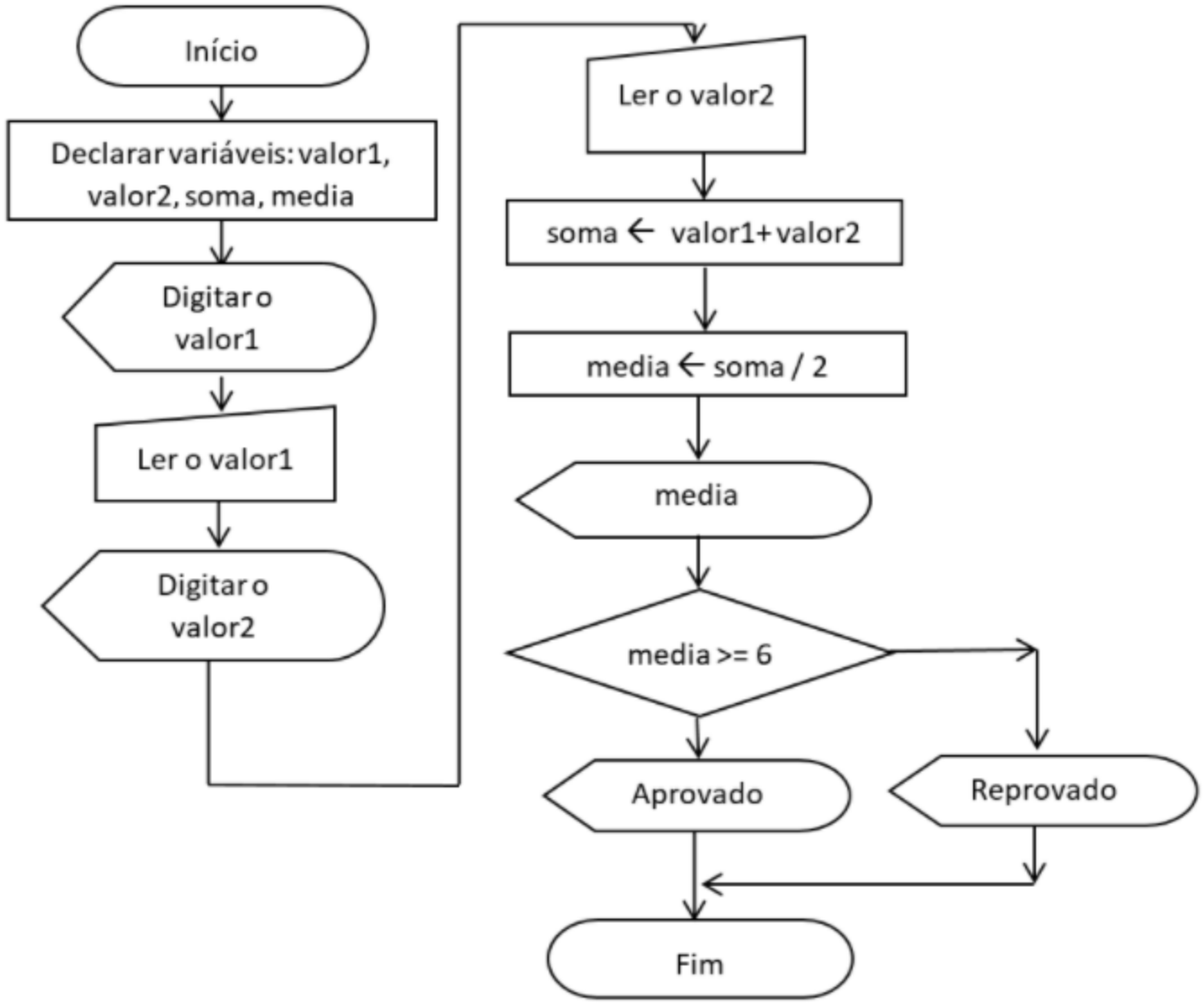
•**Descrição:** Foca em **como** o programa deve fazer algo, usando instruções sequenciais.

•**Exemplo de linguagens:** C, Fortran, Assembly, Python (em parte).

•**Características:**

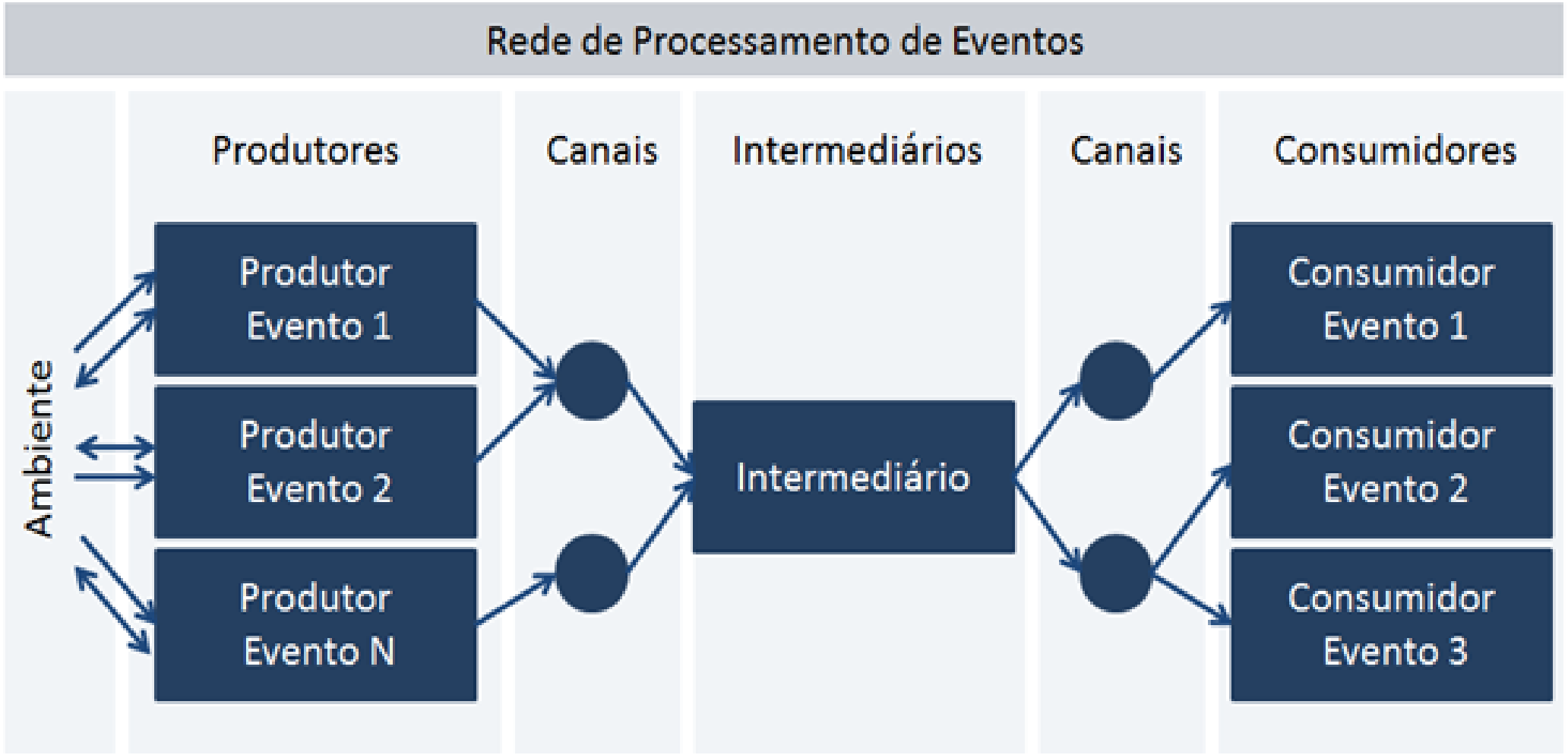
- Uso de variáveis, estruturas de controle (if, for, while)
- Manipulação direta do estado do programa

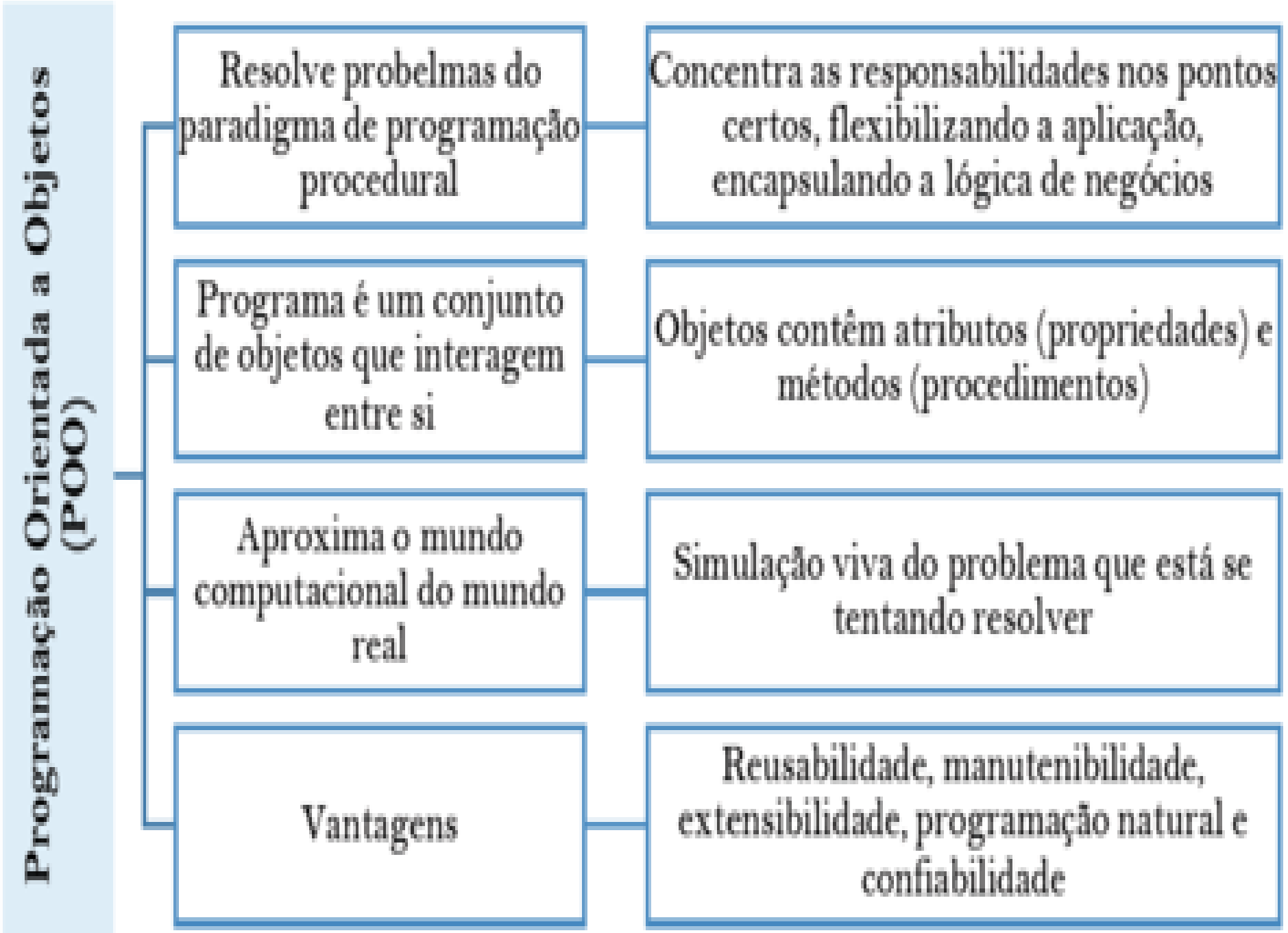
PROGRAMAÇÃO ESTRUTURADA



PROGRAMAÇÃO ORIENTADA A EVENTOS

É um paradigma de programação onde o fluxo do programa é determinado por eventos, como cliques do mouse, pressionamentos de teclas ou mensagens recebidas. Em vez de seguir uma sequência linear de instruções, o programa reage a esses eventos, executando ações específicas em resposta a eles.


Rede de Processamento de Eventos
Produtores
Canais
Intermediários
Canais
Consumidores
Ambiente
Produtor Evento 1
Produtor Evento 2
Produtor Evento N
Intermediário
Consumidor Evento 1
Consumidor Evento 2
Consumidor Evento 3


Programação Orientada a Objetos (POO)
Resolve probelmas do paradigma de programação procedural
Concentra as responsabilidades nos pontos certos, flexibilizando a aplicação, encapsulando a lógica de negócios
Programa é um conjunto de objetos que interagem entre si
Objetos contêm atributos (propriedades) e métodos (procedimentos)
Aproxima o mundo computacional do mundo real
Simulação viva do problema que está se tentando resolver
Vantagens
Reusabilidade, manutenibilidade, extensibilidade, programação natural e confiabilidade

ORIENTAÇÃO A OBJETOS

A Programação Orientada a Objetos (POO) é um paradigma ou estilo de programação que nos ajuda a trabalhar com classes e objetos para criar vários objetos de uma única classe e assim reutilizar código.

Classes e Objetos

Em PHP, uma classe é um modelo (ou estrutura) que define como um objeto deve se comportar. Ela serve como uma "fábrica" para criar objetos, agrupando atributos (variáveis) e métodos (funções) que representam as características e comportamentos desse objeto.

Extensão dos arquivos

Extensão de Arquivos

- .php
  - Arquivo PHP contendo um programa;
- class.php
  - Arquivo PHP contendo uma classe;
- inc.php
  - Arquivo a ser incluído, pode incluir constantes ou configurações;

Exemplo de classe simples

```php
<?php
class Carro
{
        // Atributos
        public $marca;
        public $cor;
        // Método
        public function buzinar()
                {echo "Bii Bii!";}
        public function mostrarDetalhes()
                {echo "Marca: $this->marca, Cor: $this->cor"; }
}

// Continua no próximo slide
```

Exemplo de classe simples

```
// Criando um objeto da classe Carro
$meuCarro = new Carro();
$meuCarro->marca = "Toyota";
$meuCarro->cor = "Preto";

$meuCarro->buzinar();          // Saída: Bii Bii!
$meuCarro->mostrarDetalhes();  // Saída: Marca: Toyota, Cor: Preto
?>
```

SOBRE AS CLASSES

Componentes principais de uma classe:

class: palavra-chave usada para definir uma classe.

Atributos: variáveis dentro da classe que armazenam dados.

Métodos: funções dentro da classe que executam ações.

$this: referência ao próprio objeto, usada para acessar atributos e métodos dentro da classe.

Por que usar classes?

Organização do código.

Reutilização (você pode criar vários objetos a partir da mesma classe).

Facilidade na manutenção e expansão (por meio de herança, encapsulamento etc.).

class

A definição de uma classe começa com a palavra chave class, seguida do nome da classe, seguido de um par de chaves que englobam as definições de propriedades e métodos pertencentes à classe.

O nome de uma classe pode ser qualquer identificador válido, que não seja uma palavra reservada do PHP. A partir do PHP 8.4.0, usar um único sublinhado _ como nome de classe foi descontinuado.

class

Um nome de classe válido começa com uma letra ou sublinhado, seguido de qualquer sequência de letras, números e sublinhados. Como uma expressão regular, pode ser expressada assim: ^[a-zA-Z_\x80-\xff][a-zA-Z0-9_\x80-\xff]*$.

Expressões regulares

Expressões regulares em PHP são padrões usados para buscar, comparar e manipular strings com base em regras definidas por esses padrões. Elas são extremamente úteis para validar formatos de dados (como e-mails, telefones, CEPs), extrair partes de textos ou substituir trechos de strings.

Exemplo simples de expressão regular

```
$texto = "Meu número é 99999-1234";
$padrao = "/[0-9]{5}-[0-9]{4}/"; // Encontra padrão de
telefone

if (preg_match($padrao, $texto, $resultado))
{
        echo "Telefone encontrado: " . $resultado[0];
}
```

NESTE EXEMPLO

preg_match verifica se o texto contém o padrão especificado.

O padrão /[0-9]{5}-[0-9]{4}/ representa 5 dígitos, um hífen e mais 4 dígitos.

Principais funções com expressões regulares em PHP

preg_match() — Verifica se há correspondência com o padrão.

preg_match_all() — Encontra todas as correspondências.

preg_replace() — Substitui partes do texto com base no padrão.

preg_split() — Divide uma string com base no padrão.

Exemplo com validação de e-mail

```
$email = "teste@exemplo.com";
$padrao = "/^[\w\-\.]+@([\w\-]+\.)+[\w\-]{2,4}$/";

if (preg_match($padrao, $email))
        {echo "E-mail válido";}
else
        {echo "E-mail inválido";}
```

Elementos comuns em expressões regulares

Símbolo	Significado
.	Qualquer caractere
^	Início da string
$	Fim da string
*	Zero ou mais ocorrências
+	Uma ou mais ocorrências
?	Zero ou uma ocorrência
[]	Conjunto de caracteres
{n}	Exatamente n ocorrências

Para qual tipo de dado ou informação você quer criar a expressão regular?

Exemplos comuns:

- CPF
- CNPJ
- E-mail
- Telefone
- CEP
- Placa de carro
- Senha segura (com letras, números e símbolos)
- Data (ex: dd/mm/aaaa)

O que é Programação Orientada a Objetos (POO)?

É um paradigma de programação baseado em **objetos**, que são instâncias de **classes**. Você modela o código como se estivesse representando elementos do mundo real.

Exemplo básico – Classe pessoa

```
class Pessoa
{
        public $nome;
        public $idade;

        // Método construtor
        public function __construct($nome, $idade)
        {
                $this->nome = $nome;
                $this->idade = $idade;
        }
        public function apresentar()
        {
                 return "Olá, meu nome é {$this->nome} e tenho {$this->idade} anos.";
        }
}
```

Switch

Aritméticos: +, -, *, /, %

Lógicos: &&, ||, !

Comparação: ==, ===, !=, !==, <, >, <=, >=

Atribuição: =, +=, -=, etc.

Exemplo: Classe- Objeto - Método - Atributo.

Classes Abstratas e Concretas

A classe abstrata é uma classe que não possui objetos instanciados a partir dela, enquanto a classe concreta possui objetos instanciados (*criados*) a partir dela.

Exemplo: No mundo real, por exemplo, existem automóveis e aviões, mas nada que seja simplesmente um veiculo (*em outras palavras, se não for um carro ou avião, não é de nosso interesse*). Isso significa que podemos instanciar objetos como carros e aviões, mas nunca iremos criar objetos veículos. As classes abstratas são criadas quando necessitamos de uma classe que implemente recursos comuns a duas ou mais classes.

EXEMPLO DE CLASSE CARRO EM PHP

```php
<?php

    class Carro
    {
        // Atributos (propriedades)
        private $marca;
        private $modelo;
        private $ano;
        // Construtor
        public function __construct($marca, $modelo, $ano)
        {
            $this->marca = $marca;
            $this->modelo = $modelo;
            $this->ano = $ano;
        }
```

CONTINUA NO PRÓXIMO SLIDE

CONTINUAÇÃO

```
// Métodos (comportamentos)
public function exibirDetalhes() {
    echo "Carro: {$this->marca} {$this->modelo}, Ano: {$this->ano}<br>";
}
// Getters e Setters
public function getMarca() {
    return $this->marca;
}
public function setMarca($marca) {
    $this->marca = $marca;
}
public function getModelo() {
    return $this->modelo;
}
```

EXEMPLO DE CLASSE CARRO EM PHP

CONTINUAÇÃO

```
    public function getAno()
            {return $this->ano;}

    public function setAno($ano)
            {$this->ano = $ano;}
}
// Criando um objeto da classe Carro
            $meuCarro = new Carro("Toyota", "Corolla", 2020);

// Usando o método
            $meuCarro->exibirDetalhes();

// Modificando valores com os setters
            $meuCarro->setAno(2022);
            $meuCarro->exibirDetalhes();

?>
```

O que esse exemplo mostra:

private: Encapsulamento dos atributos (só podem ser acessados por métodos da própria classe).

__construct: Método especial chamado ao instanciar o objeto.

get e set: Métodos para acessar e modificar atributos privados.

new Carro(...): Criação de um objeto (instância da classe).

Uma classe de interface

```php
<?php

interface VeiculoInterface {
    public function acelerar();
    public function frear();
}

class Moto implements VeiculoInterface {
    public function acelerar() {
        echo "Moto acelerando...<br>";
    }

    public function frear() {
        echo "Moto freando...<br>";
    }
}

$moto = new Moto();
$moto->acelerar();
$moto->frear();

?>

```

Exemplo classe fabricante

```php
<?php
class Fabricante {
    private $nome;
    private $endereco;
    private $documento;

    public function __construct( $nome, $endereco, $documento ) {
        $this->nome      = $nome;
        $this->endereco  = $endereco;
        $this->documento = $documento;
    }

    public function getNome() {
        return $this->nome;
    }
}
```

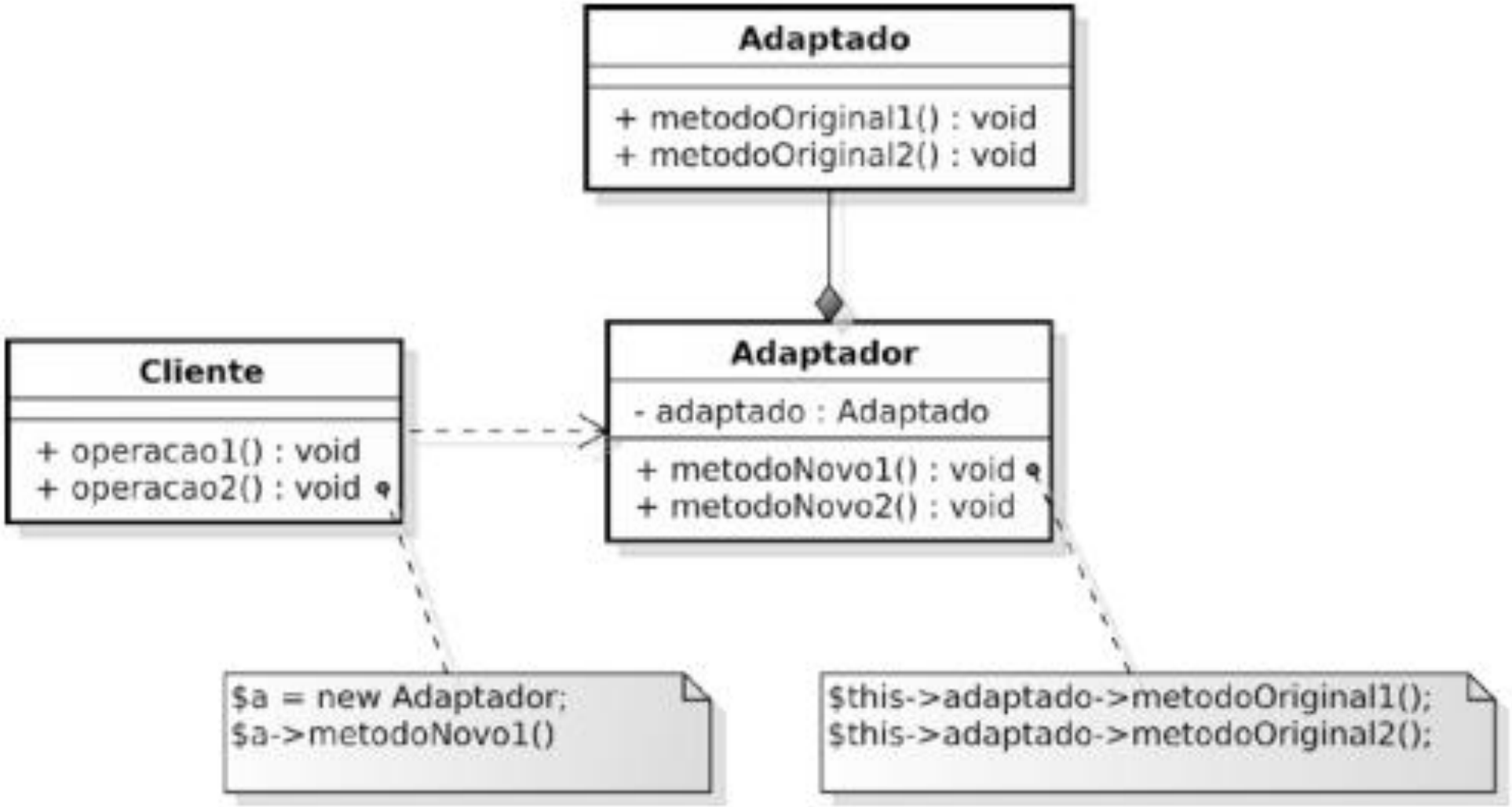
Encapsulamento

Para atingir o encapsulamento, uma das formas é definir a visibilidade das propriedades e dos métodos de um objeto.

A visibilidade define a forma como essas propriedades devem ser acessadas. Existem três formas de acesso:

Encapsulamento

Visibilidade	Descrição
`public`	Membros declarados como `public` poderão ser acessados livremente a partir da própria classe em que foram declarados, a partir de classes descendentes e a partir do programa que faz uso dessa classe (manipulando o objeto em si). Na UML, simbolizamos com um + na frente da propriedade.
`private`	Membros declarados como `private` somente podem ser acessados dentro da própria classe em que foram declarados. Não poderão ser acessados a partir de classes descendentes nem a partir do programa que faz uso dessa classe (manipulando o objeto). Na UML, simbolizamos com um - na frente da propriedade.
`protected`	Membros declarados como `protected` somente podem ser acessados dentro da própria classe em que foram declarados e a partir de classes descendentes, mas não poderão ser acessados a partir do programa que faz uso dessa classe (manipulando o objeto em si). Na UML, simbolizamos com um caractere # na frente da propriedade.


Adaptado
+ metodoOriginal1() : void
+ metodoOriginal2() : void
Adaptador
- adaptado : Adaptado
+ metodoNovo1() : void
+ metodoNovo2() : void
Cliente
+ operacao1() : void
+ operacao2() : void
$a = new Adaptador;
$a->metodoNovo1()
$this->adaptado->metodoOriginal1();
$this->adaptado->metodoOriginal2();

Exercícios

Exercício 1

Classe Carro

Objetivo: Simular aceleração de um carro.

Exercício 2

Classe Retângulo

Objetivo: Calcular área de um retângulo.

Exercício 3

Criar uma classe com atributos e um método para exibir dados.

solução

Exercício 3

Criar uma classe com atributos e um método para exibir dados.

solução

Exercício 3

```php
<?php
class Pessoa {
    public $nome;
    public $idade;

    public function apresentar() {
        echo "Olá, meu nome é $this->nome e tenho $this->idade anos.";
    }
}

// Criando objeto
$pessoa1 = new Pessoa();
$pessoa1->nome = "João";
$pessoa1->idade = 30;

$pessoa1->apresentar();
?>
```