

# Apresentação do Componente Curricular + Nivelamento

UNIVERSIDADE DO OESTE DE SANTA CATARINA - UNOESC

Ciência da Computação | Programação II | 2025/2

Prof. Leandro Otavio Cordova Vieira

```
object|erei| 9>
<ohpect-orgofesylrow 07'> e2 iquestis 07">
sobpocetienacion 07">
<pnti ]ot>"l>
<otcouE| 008">
<ohpectetectouahuniveabgysned|, 07"">
emp jottectoett l[noaccio]=
2 antY(closto-guisisn loþofernet |
2 <cupofeiert eurt"Lycect"Leota)>
3 anct"l eat luhmoccioes auutilesé]>
2 <oprcceiiluarniosJrcn 07">
<ftiles-camen |>
0 > 2 <opecteocoial 077">snecliiie]|,
0 > 2 ent"Calculationerivecation>
```

## Unlock your potential with PHP

Expert PHP development, tailored solutions,  
and seamless integration.

# Objetivos da Aula

1

## Apresentar objetivos e estrutura da disciplina

Conhecer o plano de ensino, metodologia e critérios de avaliação para o semestre.

2

## Nivelar conhecimentos prévios sobre paradigmas

Identificar o nível de experiência prévia da turma com diferentes abordagens de programação.

3

## Introduzir a base da Programação Orientada a Objetos

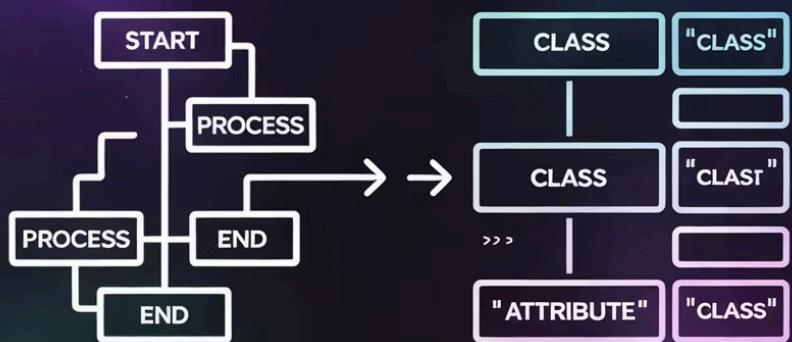
Compreender os fundamentos e conceitos essenciais que formam a base do paradigma orientado a objetos.

4

## Contextualizar o PHP na evolução da POO

Entender como o PHP evoluiu para incorporar recursos avançados de programação orientada a objetos.

# Programação Estruturada vs. POO



## Programação Estruturada

- Foco em procedimentos e funções
- Baseada em sequência, condição e repetição
- Dados e funções são entidades separadas
- Código organizado de cima para baixo

## Programação Orientada a Objetos

- Foco em objetos e suas interações
- Organização por responsabilidades
- Dados e comportamentos unidos em objetos
- Reutilização através de herança

Qual paradigma você utilizou mais até hoje em seus projetos?

# Mesa Redonda: Nivelamento

## 1 Qual linguagem você conhece melhor?

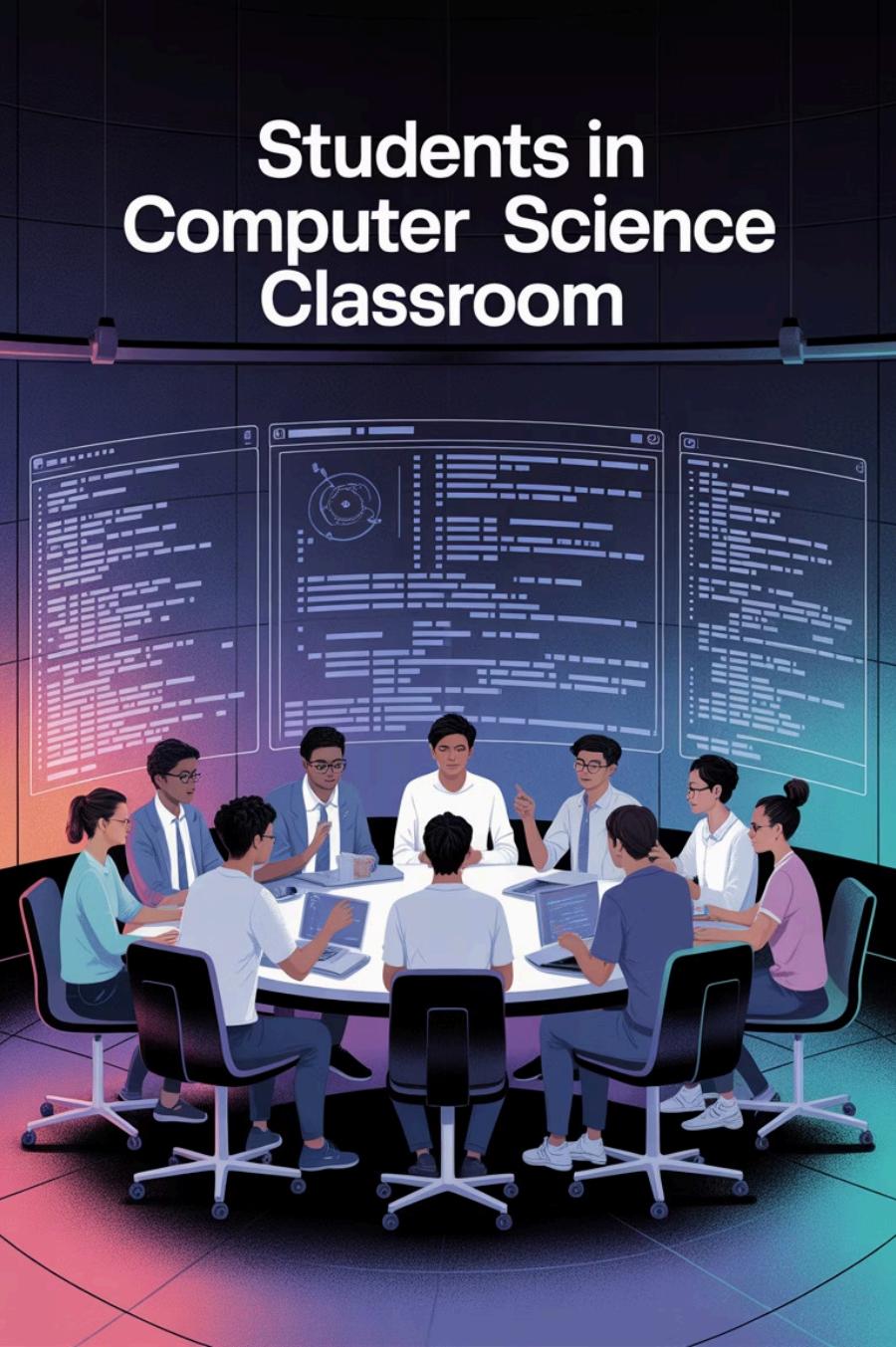
Compartilhe sua experiência com diferentes linguagens de programação e em quais você se sente mais confortável.

## 2 Já usou classes e objetos?

Relate suas experiências anteriores com o paradigma orientado a objetos em qualquer linguagem.

## 3 Já programou com PHP? Em qual versão?

Compartilhe seu conhecimento sobre PHP e as versões com as quais já trabalhou.



**Students in  
Computer Science  
Classroom**

# Paradigmas de Programação

## 1 Imperativo (1950s-1960s)

Baseado em comandos sequenciais que alteram o estado do programa. Foco em "como" fazer.

Exemplos: Fortran, COBOL, Assembly

## 2 Estruturado (1960s-1970s)

## Orientado a Objetos (1980s-presente)

Evolução do imperativo com estruturas de controle melhoradas e modularização.

Exemplos: C, Pascal

Organização em objetos que combinam dados e comportamentos. Surgiu para gerenciar a crescente complexidade dos sistemas.

Exemplos: Java, C++, PHP (moderno)

1

2

3

4

## Funcional e Lógico (1980s-presente)

Abordagens alternativas baseadas em funções matemáticas e lógica formal.

Exemplos: Haskell, Prolog

A POO surgiu como resposta à necessidade de lidar com a crescente complexidade dos sistemas de software.

# O que é Programação Orientada a Objetos?

É um paradigma que organiza o software baseando-se em "coisas" do mundo real, modelando entidades como objetos que possuem:

- Características (atributos/propriedades)
- Comportamentos (métodos/funções)
- Estados (valores atuais dos atributos)

Uma classe funciona como um molde ou planta que define a estrutura para criar objetos específicos.



- ⓘ A POO permite modelar software de forma semelhante a como percebemos o mundo real, facilitando o entendimento e a manutenção do código.

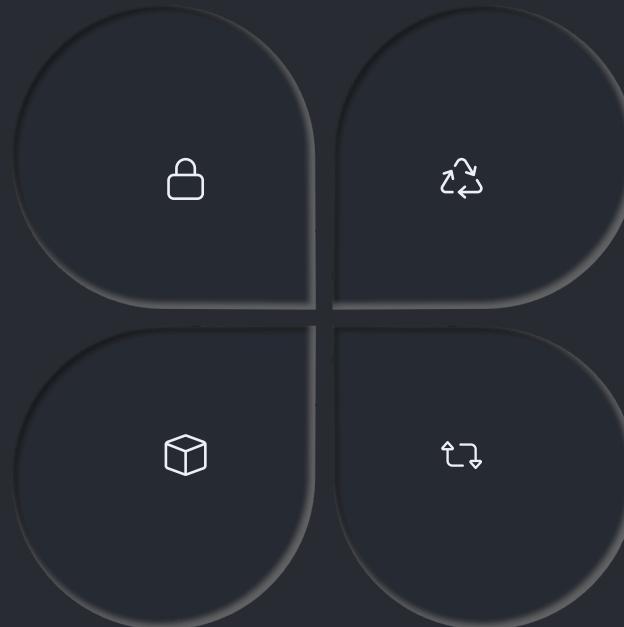
# Por que usar POO?

## Encapsulamento

Protege os dados internos do objeto, permitindo acesso apenas através de métodos controlados, aumentando a segurança.

## Organização e manutenção

Estrutura modular facilita o entendimento, a depuração e a manutenção do código a longo prazo.



## Reutilização de código

Através de herança e composição, permite aproveitar código existente para criar novas funcionalidades.

## Flexibilidade com polimorfismo

Permite que objetos de diferentes classes sejam tratados de maneira uniforme, facilitando extensões futuras.

# Elementos Básicos da POO

## Classe



Modelo ou "planta" que define a estrutura de um tipo de objeto. Define atributos e métodos que os objetos terão.

```
class Usuario { ... }
```

## Objeto



Instância concreta de uma classe. Representa uma entidade específica com seus próprios valores de atributos.

```
$usuario1 = new Usuario();
```

## Atributo



Variável pertencente à classe que armazena dados específicos do objeto.

```
public $nome; public $email;
```

## Método



Função definida na classe que determina o comportamento dos objetos.

```
public function fazerLogin() { ... }
```

## Construtor



Método especial chamado quando um objeto é criado, geralmente usado para inicializar atributos.

```
public function __construct() { ... }
```

# Evolução da POO no PHP

- 1** **PHP 3 (1998)**

Linguagem predominantemente procedural e estruturada. Sem suporte real para POO.

Focada em scripts simples para web.
  - 2** **PHP 4 (2000)**

Primeiros passos em direção à POO, com classes básicas e objetos.

Sem visibilidade de métodos ou herança robusta.
  - 3** **PHP 5 (2004)**

Revolução na POO: suporte completo para visibilidade (public, private, protected).

Introdução de interfaces, classes abstratas e tratamento de exceções.
  - 4** **PHP 7/8 (2015-2020)**

Aperfeiçoamento da POO: tipagem rígida, return types, nullable types.

Performance drasticamente melhorada e novos recursos avançados.

# PHP Evolution: Object-oriented features



# Novidades no PHP 8+



## Tipagem nas propriedades

```
class Usuario {  
    public string $nome;  
    private int $id;  
}
```

## Construtores promovidos

```
class Produto {  
    public function __construct(  
        public string $nome,  
        public float $preco  
    ) {}  
}
```

## Union types

```
public function setValor(  
    int|float $valor  
) {}
```

## Traits e interfaces aprimorados

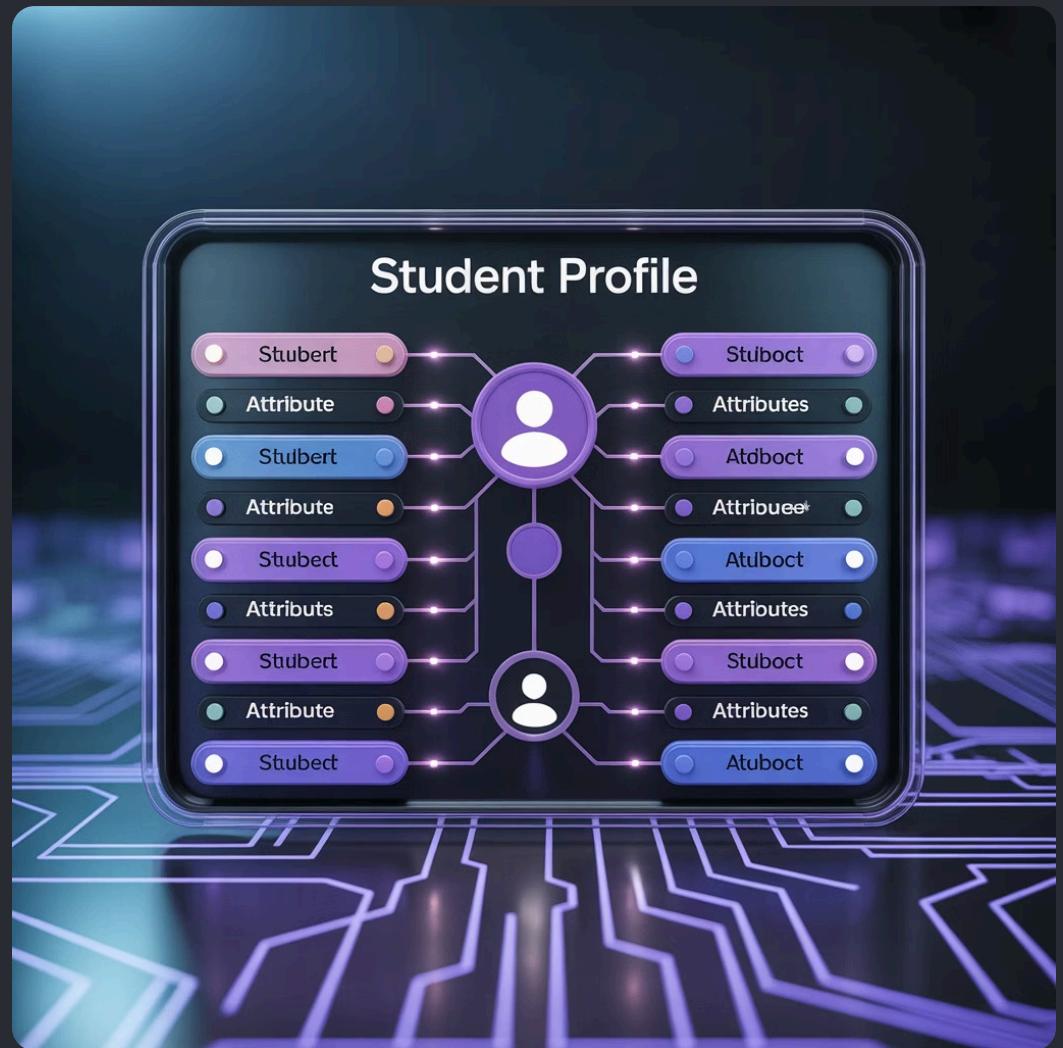
Melhor suporte para composição e polimorfismo.

# Exemplo Intuitivo: Classe Aluno

```
class Aluno {  
    public $nome;  
    public $curso;  
  
    public function apresentar() {  
        return "Olá, eu sou $this->nome  
        do curso $this->curso.";  
    }  
}
```

Esta classe define o modelo para objetos do tipo "Aluno" com:

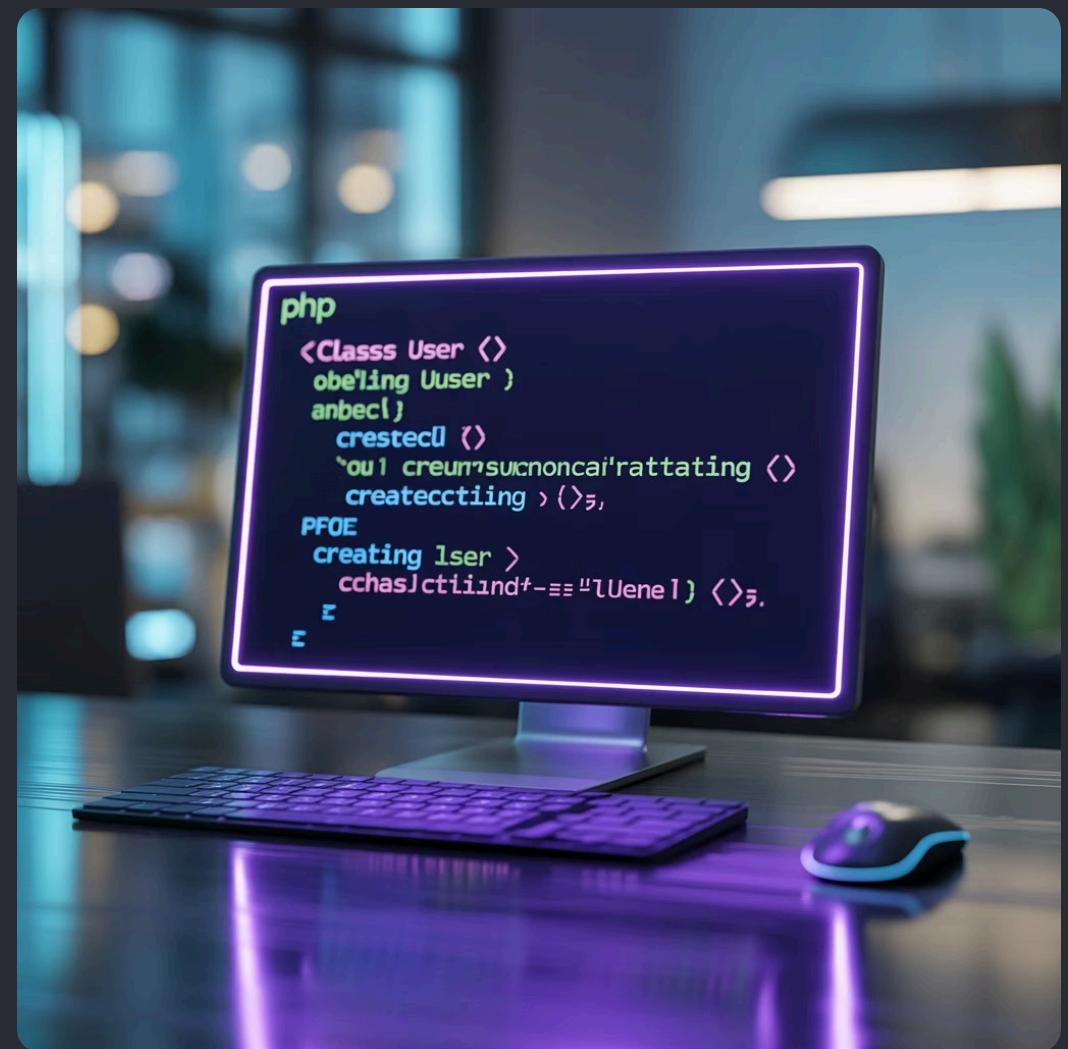
- Dois atributos: nome e curso
- Um método: apresentar()



- ❑ No PHP, a palavra-chave **\$this** refere-se ao objeto atual e é usada para acessar suas propriedades e métodos.

# Objeto a partir da Classe

```
// Criando um objeto da classe Aluno  
$leandro = new Aluno();  
  
// Definindo valores para os atributos  
$leandro->nome = "Leandro";  
$leandro->curso = "CCO";  
  
// Chamando o método apresentar  
echo $leandro->apresentar();  
  
// Resultado:  
// "Olá, eu sou Leandro do curso CCO."
```



Neste exemplo, criamos uma **instância** da classe Aluno, atribuímos valores aos seus atributos e chamamos seu método.

O objeto **\$leandro** é uma entidade concreta criada a partir do molde (classe) Aluno.

# Diagrama Conceitual (Classe vs. Objeto)



## 1 Classe: Aluno

**Definição:** É o modelo que define como todos os alunos serão estruturados.

**Atributos:**

- \$nome (string)
- \$curso (string)

**Métodos:**

- apresentar()

## 2 Objeto: \$leandro

**Instância:** É um aluno específico criado a partir da classe.

**Valores dos atributos:**

- \$nome = "Leandro"
- \$curso = "CCO"

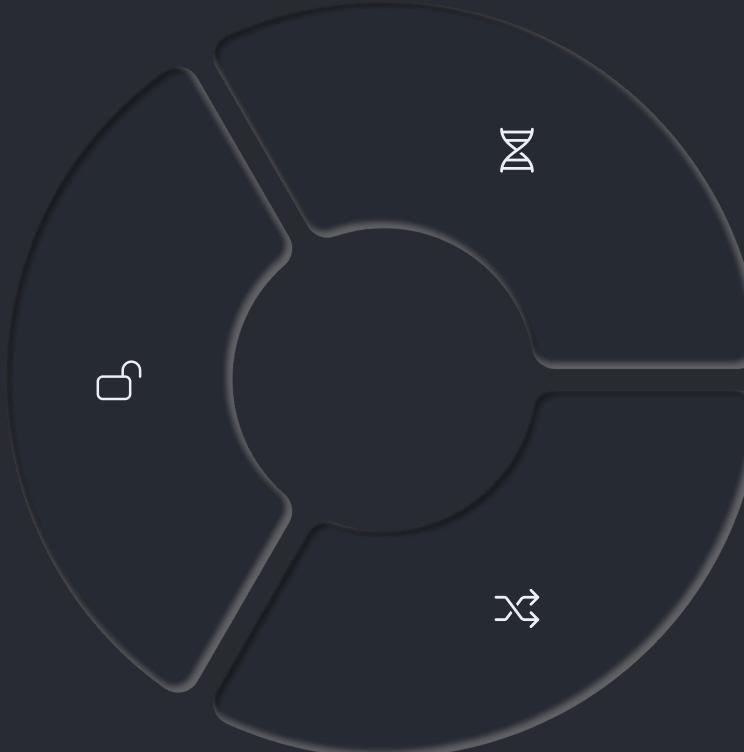
**Comportamento:** Quando chamamos \$leandro->apresentar(), obtemos "Olá, eu sou Leandro do curso CCO."

# Três Pilares da POO

## Encapsulamento

Proteção dos dados internos do objeto, controlando o acesso através de modificadores de visibilidade.

- public: acessível de qualquer lugar
- private: acessível apenas dentro da classe
- protected: acessível na classe e em suas descendentes



## Herança

Mecanismo que permite criar novas classes a partir de classes existentes, aproveitando seus atributos e métodos.

- Evita repetição de código
- Cria hierarquias de classes
- Em PHP: "class Filho extends Pai"

## Polimorfismo

Capacidade de objetos de classes diferentes responderem à mesma mensagem de maneiras distintas.

- Sobrescrita de métodos
- Interfaces
- Flexibilidade no código

Estes conceitos serão aprofundados nas próximas aulas.

# Vantagens no Desenvolvimento Moderno

## Organização e clareza no código

Estruturas bem definidas facilitam a compreensão do código, especialmente em projetos grandes.

## Facilidade de manutenção e testes

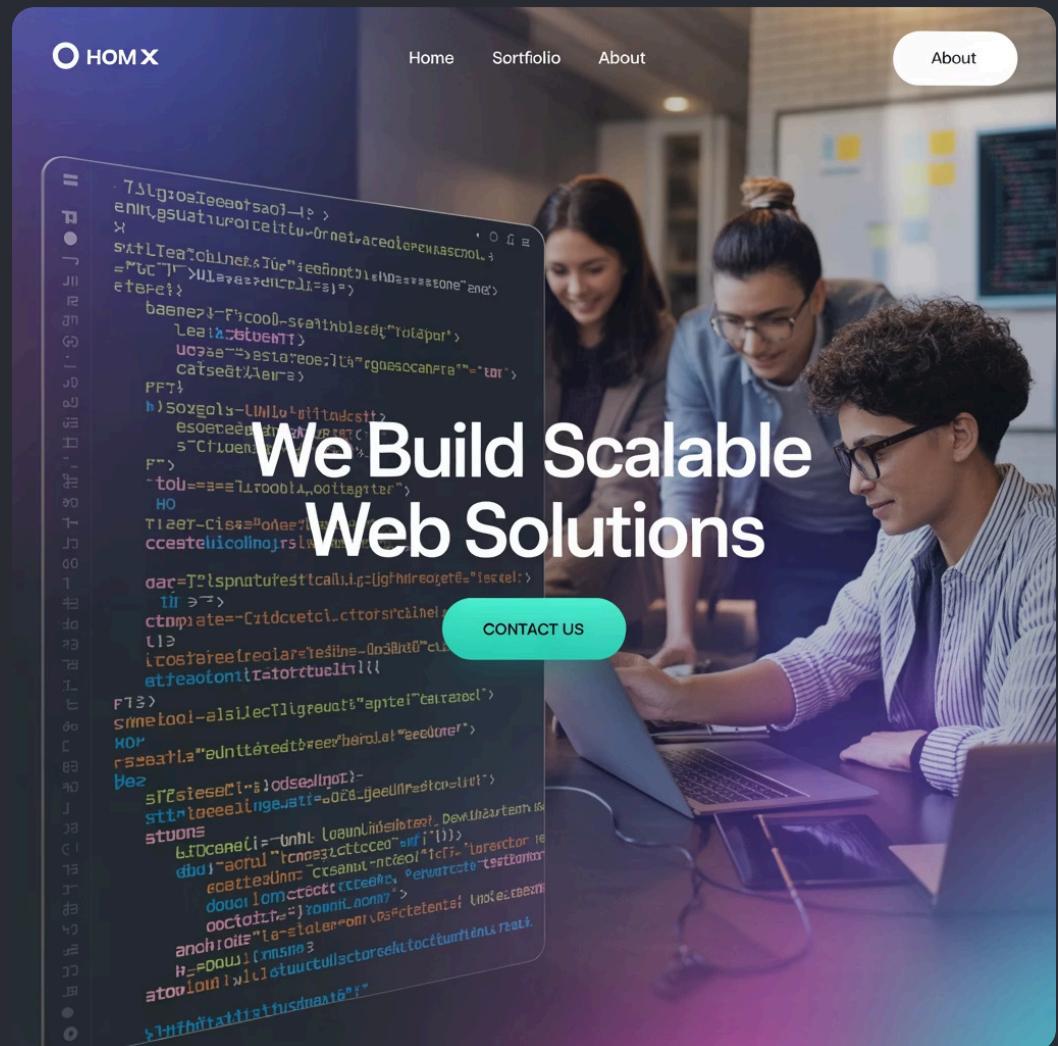
Componentes isolados podem ser testados individualmente e modificados sem afetar todo o sistema.

## Reuso de código

Herança e composição permitem reutilizar estruturas existentes, economizando tempo de desenvolvimento.

## Flexibilidade e expansão

Sistemas orientados a objetos são mais fáceis de estender com novas funcionalidades.



A POO é essencial em frameworks PHP modernos como Laravel, Symfony e CodeIgniter, que dependem fortemente dos conceitos orientados a objetos.

# Prática Guiada (atividade em duplas)

## Análise de código PHP estruturado:

```
// Código PHP estruturado  
$preco = 100;  
$desconto = 0.1;  
$final = $preco - ($preco * $desconto);  
echo $final;  
  
// Resultado: 90
```

Este é um exemplo típico de código estruturado, onde temos:

- Variáveis independentes (\$preco, \$desconto)
- Operações sequenciais
- Sem agrupamento lógico entre dados e comportamentos

Na próxima etapa, vamos transformar este código em uma abordagem orientada a objetos.

# Desafio: Transformar para POO

Em duplas, identifiquem:

## 1 Qual seria a classe?

Pense em qual entidade do mundo real este código representa.

## 2 Quais atributos e métodos?

Identifique os dados (variáveis) e comportamentos (operações) que fariam parte desta classe.

## 3 Esboce a estrutura OOP correspondente

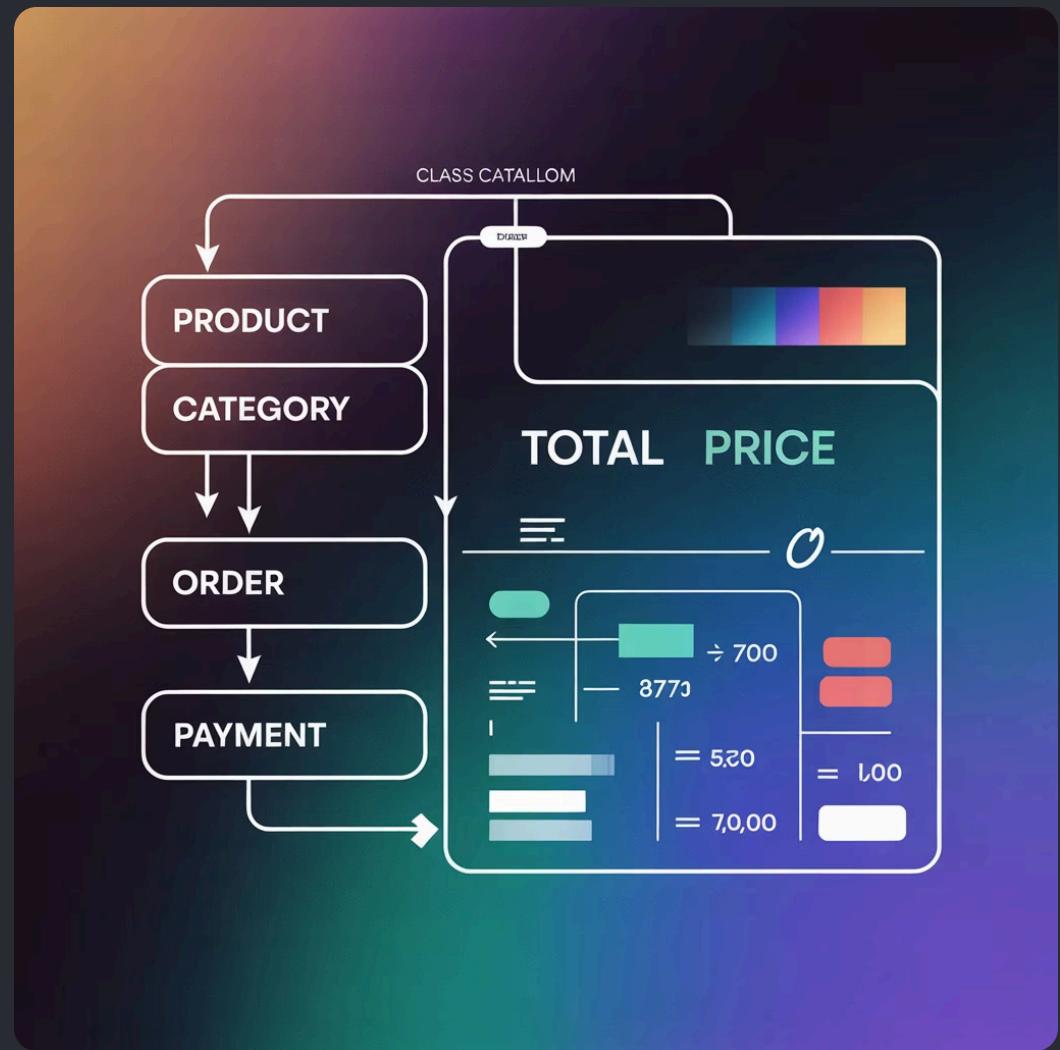
Crie a estrutura básica da classe com seus atributos e métodos.



② Reflexão: Como você agruparia estes elementos em uma classe que represente um conceito do mundo real?

# Modelo Esperado (exemplo)

```
class Produto {  
    public $preco;  
    public $desconto;  
  
    public function precoFinal() {  
        return $this->preco -  
            ($this->preco * $this->desconto);  
    }  
}  
  
// Uso da classe  
$camiseta = new Produto();  
$camiseta->preco = 100;  
$camiseta->desconto = 0.1;  
echo $camiseta->precoFinal();  
  
// Resultado: 90
```



Nesta versão orientada a objetos:

- Criamos a classe **Produto** que agrupa dados e comportamentos
- Os dados (preço e desconto) são atributos da classe
- O cálculo do preço final é um método que opera sobre os atributos
- Criamos um objeto específico (`$camiseta`) a partir da classe

# Reflexão em Grupo

## O que muda ao usar POO nesse exemplo?

### Organização

O código fica mais organizado e coeso, agrupando dados e comportamentos relacionados em uma única estrutura.

Facilita encontrar todos os elementos relacionados a um conceito em um único lugar.

### Reaproveitamento

A classe pode ser instanciada múltiplas vezes para criar diferentes produtos.

O método de cálculo pode ser utilizado por todos os objetos da classe sem duplicação de código.

### Clareza

O código comunica melhor sua intenção, refletindo entidades do mundo real.

A estrutura do programa se torna mais compreensível para novos desenvolvedores.



# Encerramento e Tarefa

## Leitura para próxima aula

Capítulo 2 (2.1 e 2.2) do eBook disponível no Ambiente Virtual de Aprendizagem.

## Preparação prática

Prepare seu ambiente de desenvolvimento PHP para criar suas primeiras classes na próxima aula.

## Participação no fórum

Acesse o fórum no AVA e compartilhe suas percepções iniciais sobre Programação Orientada a Objetos.



- ⓘ Na próxima aula, exploraremos mais a fundo a criação de classes em PHP, construtores e modificadores de acesso.

Dúvidas podem ser enviadas pelo AVA ou trazidas para a próxima aula!