🧩 Exercício 1 — Classe Abstrata Base

Crie uma **classe abstrata** chamada `Pessoa` que servirá de base para outras classes.

Estrutura esperada:

```
<?php
abstract class Pessoa {
    protected $nome;
    protected $idade;
    protected $sexo;

    public function __construct($nome, $idade, $sexo) {
        $this->nome = $nome;
        $this->idade = $idade;
        $this->sexo = $sexo;
    }

    // Método comum (não abstrato)
    final public function fazerAniversario() {
        $this->idade++;
        echo "<p>Parabéns, {$this->nome}! Agora você tem
{$this->idade} anos.</p>";
    }

    // Método abstrato
    abstract public function apresentar();
}
```

Tarefas:

1. Crie a classe acima. (VS Code)

2. Explique por que não é possível instanciá-la diretamente (`new Pessoa()` deve gerar erro).
   Porque ela é uma classe abstrata, ou seja, não pode ser instanciada.

3. Identifique o papel do método **`final`**.
   Ele é um método que impede de ser sobrescrito por outras classes, elas herdam o método, mas não podem sobrescrever, garantindo que esse método seja o mesmo em todas as subclasses

4. Explique a função de um método **abstrato**.
   Um método abstrato define uma função sem implementação e serve para obrigar as classes filhas a criarem sua própria versão desse método, garantindo que todas sigam um mesmo padrão de comportamento.

🧩 Exercício 2 — Herança de Implementação

Crie uma classe chamada `Visitante` que **herda de `Pessoa`** e **implementa** o método abstrato `apresentar()`.

Estrutura esperada:

```
class Visitante extends Pessoa {
    public function apresentar() {
        echo "<p>Sou um visitante chamado {$this->nome}.</p>";
    }
}
```

Tarefas:

1. Implemente `Visitante`. (VS Code)

2. Instancie um visitante e teste os métodos herdados (`fazerAniversario()` e `apresentar()`). (VS Code)

3. Confirme que `Visitante` é uma **classe concreta** (instanciável).
   A classe Visitante herda de Pessoa e implementa o método abstrato apresentar(). Como todos os métodos abstratos foram implementados, Visitante se torna uma classe concreta, ou seja, pode ser instanciada normalmente com new Visitante(...).

4. Essa herança é **pobre** ou **por diferença**? Justifique.
   Essa é uma herança pobre, isso acontece porque Visitante herda tudo da classe Pessoa, mas não adiciona nenhum novo atributo ou comportamento, apenas implementa o método abstrato obrigatório. Ou seja, não há diferença significativa entre Pessoa e Visitante, apenas o mínimo necessário para torná-la instanciável.

🧩 Exercício 3 — Herança por Diferença

Crie uma classe `Aluno` que também **herda de `Pessoa`**, mas **acrescenta novos atributos e métodos**.

Estrutura esperada:

```
class Aluno extends Pessoa {
    protected $matricula;
    protected $curso;

    public function __construct($nome, $idade, $sexo, $matricula,
$curso) {
        parent::__construct($nome, $idade, $sexo);
        $this->matricula = $matricula;
        $this->curso = $curso;
    }

    public function apresentar() {
        echo "<p>Sou o aluno {$this->nome}, do curso de
{$this->curso}.</p>";
    }

    public function pagarMensalidade() {
        echo "<p>Mensalidade de {$this->nome} paga com
sucesso!</p>";
    }
}
```

Tarefas:

1. Implemente `Aluno` conforme o modelo. (VS Code)

2. Explique o uso de `parent::__construct()`.
   O comando parent::__construct() é utilizado dentro do construtor da classe filha (Aluno) para chamar o construtor da classe pai (Pessoa).Isso evita repetição de código e garante que os atributos herdados (nome, idade, sexo) sejam corretamente inicializados pela lógica já existente na classe pai. Assim, o construtor da filha cuida apenas dos novos atributos (matrícula e curso).

3. Teste a criação de um objeto e verifique o método `fazerAniversario()` herdado. (VS Code)

4. Por que `fazerAniversario()` não pode ser sobrescrito?
   O método fazerAniversario() foi declarado com o modificador final na classe Pessoa, o que significa que nenhuma subclasse pode redefini-lo.Isso garante que o comportamento desse método permaneça idêntico em todas as classes filhas, evitando alterações acidentais ou intencionais na sua lógica.

🧩 Exercício 4 — Subclasse com Sobrescrita e Novo Método

Crie uma classe `Bolsista` que **herda de `Aluno`**, adicionando um atributo e sobrescrevendo um método.

Estrutura esperada:

```
class Bolsista extends Aluno {
    private $bolsa;

    public function __construct($nome, $idade, $sexo, $matricula, $curso, $bolsa) {
        parent::__construct($nome, $idade, $sexo, $matricula, $curso);
        $this->bolsa = $bolsa;
    }

    public function renovarBolsa() {
        echo "<p>Bolsa renovada para {$this->nome}!</p>";
    }

    public function pagarMensalidade() {
        echo "<p>{$this->nome} é bolsista! Pagamento com desconto de {$this->bolsa}%.</p>";
    }
}
```

Tarefas:

1. Implemente `Bolsista` e teste os métodos. (VS Code)

2. Identifique o conceito de **polimorfismo** no método `pagarMensalidade()`.
   O polimorfismo ocorre quando um método herdado é redefinido (sobrescrito) em uma subclasse, assumindo comportamentos diferentes conforme o tipo do objeto que o executa.
   Nesse caso, a classe Aluno tem um método pagarMensalidade() com um comportamento padrão. A classe Bolsista, que herda de Aluno, sobrescreve esse

método para aplicar o desconto da bolsa. Assim, se chamarmos pagarMensalidade() em um objeto do tipo Aluno, o comportamento será um; mas se chamarmos no tipo Bolsista, o comportamento será diferente.

3. Análise: o método `fazerAniversario()` pode ser sobrescrito? Por quê?
   O método fazerAniversario() foi declarado com o modificador final na classe Pessoa, o que significa que nenhuma subclasse pode redefini-lo.Isso garante que o comportamento desse método permaneça idêntico em todas as classes filhas, evitando alterações acidentais ou intencionais na sua lógica

🧩 Exercício 5 — Classe Final e Hierarquia Completa

Crie uma classe `Professor` que **herda de `Pessoa`**, e **declare-a como `final`**, impedindo novas heranças.

Estrutura esperada:

```
final class Professor extends Pessoa {
    private $especialidade;
    private $salario;

    public function __construct($nome, $idade, $sexo, $esp, $salario) {
        parent::__construct($nome, $idade, $sexo);
        $this->especialidade = $esp;
        $this->salario = $salario;
    }

    public function apresentar() {
        echo "<p>Sou o professor {$this->nome}, especialista em {$this->especialidade}.</p>";
    }

    public function receberAumento($valor) {
        $this->salario += $valor;
        echo "<p>O salário de {$this->nome} foi reajustado para R$ {$this->salario}.</p>";
    }
}
```

Tarefas:

1. Crie a classe `Professor` com `final`. (VS Code)

2. Tente criar uma classe `Coordenador extends Professor` e observe o erro.
   Geral um erro fatal, pois por Professor ser uma classe final, não pode servir de base para novas heranças.
   Fatal error: Class Coordenador may not inherit from final class (Professor)

3. Crie um vetor com um `Visitante`, um `Aluno`, um `Bolsista` e um `Professor`. (VS Code)

4. Use `get_class($obj)` e `instanceof` para identificar a hierarquia. (VS Code)

5. Identifique qual é a **raiz** e quais são as **folhas** da árvore de herança.

   Pessoa
   ├── Visitante
   ├── Aluno
   │ └──Bolsista
   └── Professor (final)

Raiz: Pessoa (classe abstrata que dá origem a todas as outras)
Folhas: Visitante, Bolsista e Professor
Visitante e Bolsista são folhas porque não têm subclasses.
Professor é uma folha por estar marcada como final, impedindo herança futura.