

# Atividades de Prog 21/10/25

---

## Exercício 1 – Criando a Classe Base

```
<?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;
    }

    final public function fazerAniversario() {
        $this->idade++;
        echo "Parabéns, $this->nome! Agora você tem $this->idade
anos!";
    }
}
```

### Explicação:

A classe Pessoa é abstrata, o que significa que não pode ser instanciada diretamente. Isso ocorre porque a classe contém um método abstrato e/ou incompleto.

O método fazerAniversario é final, o que significa que ele não pode ser sobrescrito em subclasses.

O construtor inicializa os atributos `$nome`, `$idade` e `$sexo`.

---

## Exercício 2 – Criando a Primeira Subclasse (Herança de Implementação)

```
<?php

class Visitante extends Pessoa {

}

$v1 = new Visitante('Maria', 25, 'Feminino');

echo "Nome: " . $v1->nome . "<br>";
echo "Idade: " . $v1->idade . "<br>";
echo "Sexo: " . $v1->sexo . "<br>";

$v1->fazerAniversario();
?>
```

### Explicação:

A classe Visitante herda diretamente de Pessoa sem adicionar nenhum comportamento novo.

A classe Visitante pode acessar os métodos e atributos da classe Pessoa, como o fazerAniversario().

O exercício mostra uma herança de implementação onde a subclasse não altera ou adiciona novos comportamentos.

---

## Exercício 3 – Adicionando Especialização (Herança por Diferença)

```
<?php

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 pagarMensalidade() {
        echo "Mensalidade paga com sucesso por $this->nome!";
    }
}

$a1 = new Aluno('Lucas', 20, 'Masculino', '2023001', 'Engenharia');

$a1->pagarMensalidade();
?>

```

#### Explicação:

A classe Aluno é uma especialização de Pessoa, adicionando os atributos \$matricula e \$curso, e o método pagarMensalidade().

A herança por diferença é demonstrada porque a subclasse (Aluno) adiciona novos comportamentos (atributos e métodos) além dos herdados de Pessoa.

---

#### Exercício 4 – Criando uma Subclasse Especializada (Sobrescrita)

```

<?php

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 pagarMensalidade() {
        echo "Mensalidade paga com desconto por $this->nome.";
    }
}

```

```

        public function renovarBolsa() {
            echo "Bolsa renovada com sucesso!";
        }
    }

    $b1 = new Bolsista('Ana', 22, 'Feminino', '2022001', 'Medicina',
    '50%');

    $b1->renovarBolsa();
    echo "<br>";
    $b1->pagarMensalidade();
    ?>

```

#### Explicação:

- A classe Bolsista herda de Aluno, mas sobrescreve o método pagarMensalidade() para adicionar um comportamento especializado, ex: "com desconto".
- A classe Bolsista também adiciona o método renovarBolsa().
- A sobrescrita é usada aqui para modificar o comportamento de um método da classe pai (Aluno).

---

### Exercício 5 – Ampliando a Hierarquia (Desafio Integrador)

```

<?php
class Professor extends Pessoa {
    protected $especialidade;
    protected $salario;

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

    public function receberAumento($valor) {

```

```

        $this->salario += $valor;
        echo "Novo salário de $this->nome: $this->salario!";
    }
}

$p1 = new Professor('Carlos', 45, 'Masculino', 'Matemática', 5000);
$a1 = new Aluno('Mariana', 19, 'Feminino', '2023002', 'Direito');
$b1 = new Bolsista('José', 23, 'Masculino', '2022002', 'Física',
'80%');

$vetor = [$a1, $b1, $p1];

foreach ($vetor as $obj) {
    echo "Tipo: " . get_class($obj) . "<br>";

    if ($obj instanceof Aluno) {
        $obj->pagarMensalidade();
    }
    if ($obj instanceof Bolsista) {
        $obj->renovarBolsa();
    }
    if ($obj instanceof Professor) {
        $obj->receberAumento(1000);
    }
    echo "<br>";
}
?>

```

### Explicação:

A classe Professor herda de Pessoa, mas adiciona novos atributos (especialidade, salario) e um novo método (receberAumento).

Um vetor é criado contendo objetos de várias classes e, em seguida, é percorrido para executar ações específicas dependendo do tipo do objeto. O `get_class($obj)` é usado para identificar a classe do objeto.

A classe raiz dessa hierarquia é Pessoa, e as folhas são as classes Visitante, Aluno, Bolsista, e Professor.