

# Herança e Classes Abstratas em PHP

**Curso:** Programação Orientada a Objetos

**Tema da Aula:** Herança (Parte 2) – Classes Abstratas, Finais e Tipos de Herança

## 1 Retomando Conceito de Herança

Ela permite que uma **classe filha** (ou *subclasse*) aproveite atributos e métodos de uma **classe mãe** (ou *superclasse*). Dessa forma, é possível **reutilizar código** e **especializar comportamentos** sem precisar reescrever tudo do zero.

Exemplo simples:

```
class Pessoa {  
    public $nome;  
    public $idade;  
}  
  
class Aluno extends Pessoa {  
    public $matricula;  
}
```

👉 Aqui, **Aluno herda** de **Pessoa** e, portanto, tem acesso a **\$nome** e **\$idade**. Isso significa que um aluno **é um tipo de pessoa**.

## 2 Tipos de Herança

### ◆ Herança de Implementação (ou Herança Pobre)

Ocorre quando a subclasse **não adiciona nada novo**, apenas utiliza os métodos e atributos herdados da superclasse.

```
class Pessoa {  
    public $nome;  
    public $idade;  
    public function fazerAniversario() {  
        $this->idade++;  
    }  
}  
  
class Visitante extends Pessoa {  
    // Nenhum atributo ou método novo  
}
```

Neste exemplo, **Visitante** apenas herda o que já existe em **Pessoa**. Não há novos comportamentos — é uma **herança simples**, apenas para implementação.

### ◆ Herança por Diferença

Acontece quando a subclasse **adiciona novos atributos ou métodos**, tornando-se uma versão mais específica da classe mãe.

```
class Pessoa {  
    public $nome;  
    public $idade;  
}  
  
class Aluno extends Pessoa {  
    public $matricula;  
    public $curso;  
  
    public function pagarMensalidade() {  
        echo "Mensalidade paga com sucesso!<br>";  
    }  
}
```

Aqui, **Aluno** mantém os atributos de **Pessoa**, mas **acrescenta novos** (**matricula**, **curso**) e **métodos específicos** (**pagarMensalidade()**).

## 3 Hierarquia de Herança

Quando várias classes são relacionadas, formamos uma **árvore hierárquica**, como no exemplo a seguir:

```
Pessoa  
├── Visitante  
├── Aluno  
│   ├── Bolsista  
│   └── Tecnico  
└── Professor
```

- **Pessoa** é a classe **raiz** (ou classe base).
- **Aluno**, **Professor** e **Visitante** são **subclasses** de **Pessoa**.
- **Bolsista** e **Técnico** são **especializações** de **Aluno**.

## 4 Classes e Métodos Abstratos

### ♦ Classe Abstrata

Uma **classe abstrata** serve apenas como modelo. Ela **não pode ser instanciada**, mas pode ser herdada.

```
abstract class Pessoa {  
    protected $nome;  
    protected $idade;  
    protected $sexo;  
  
    public final function fazerAniversario() {  
        $this->idade++;  
    }  
}
```

- ♦ `abstract` define a classe como abstrata.
- ♦ `final` em um método indica que ele **não pode ser sobrescrito**.
- ♦ Essa classe **não pode gerar objetos diretamente**, apenas ser herdada por outras.

Exemplo de uso incorreto:

```
$p1 = new Pessoa(); // ❌ Erro: classe abstrata não pode ser  
instanciada
```

### ♦ Método Abstrato

É um **método declarado, mas não implementado**.  
Toda subclasse **deve implementar** esse método.

```
abstract class Animal {  
    abstract public function emitirSom();  
}
```

```
class Cachorro extends Animal {  
    public function emitirSom() {  
        echo "Latindo...<br>";  
    }  
}
```

```
class Gato extends Animal {  
    public function emitirSom() {  
        echo "Miando...<br>";  
    }  
}
```

- `emitirSom()` é **abstrato** em `Animal`, mas **obrigatório** em `Cachorro` e `Gato`.
- Isso garante **padronização** nas subclasses.

## 5 Classe e Método Final

### ◆ Classe Final

Uma **classe final** não pode ser herdada.

```
final class Visitante extends Pessoa {  
    // Nenhuma outra classe pode herdar de Visitante  
}
```

### ◆ Método Final

Um **método final** não pode ser sobrescrito em subclasses.

```
class Pessoa {  
    public final function fazerAniversario() {  
        echo "Parabéns, mais um ano de vida!<br>";  
    }  
}
```

Mesmo que uma classe herde `Pessoa`, ela **não pode redefinir** esse método.

## 6 Exemplo Completo

```
abstract class Pessoa {  
    protected $nome;  
    protected $idade;  
    protected $sexo;  
  
    public final function fazerAniversario() {  
        $this->idade++;  
    }  
}  
  
class Aluno extends Pessoa {  
    protected $matricula;  
    protected $curso;  
  
    public function pagarMensalidade() {  
        echo "<p>Mensalidade de {$this->nome} paga com  
sucesso!</p>";  
    }  
}
```

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

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

    public function pagarMensalidade() {
        echo "<p>{$this->nome} é bolsista! Pagamento
facilitado.</p>";
    }
}

$aluno1 = new Aluno();
$aluno1->nome = "Maria";
$aluno1->pagarMensalidade();

$bolsista1 = new Bolsista();
$bolsista1->nome = "João";
$bolsista1->pagarMensalidade();
```

## Atividades de Fixação

### Atividade 1 – Conceitos

Explique com suas palavras:

1. O que é uma **classe abstrata**?

É uma classe que não pode ser instanciada, mas é criada para que possa ser herdada, servindo como um modelo.

2. Qual é a diferença entre uma **classe abstrata** e uma **classe final**?

Uma classe abstrata funciona como um modelo que pode ser herdado por outras classes, enquanto uma classe final pode ser instanciada, porém não pode ser herdada.

### Atividade 2 – Interpretação de Código

Explique o que ocorre ao executar:

```
$p = new Pessoa();
```

Erro, já que “Pessoa” é uma classe abstrata

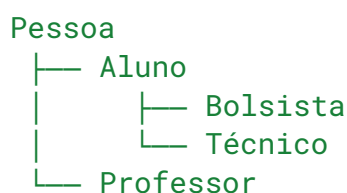
E por que o comando abaixo **funciona normalmente**:

```
$v = new Visitante();
```

“Visitante” herda a classe “Pessoa”, por isso, pode ser instanciada.

### Atividade 3 – Identificação Hierárquica

Na estrutura a seguir:



Responda:

- a) Quem é a **superclasse** de **Aluno**? Pessoa
- b) Quem é a **subclasse** de **Aluno**? Bolsista e Técnico
- c) Quem é **ancestral** de **Bolsista**? Aluno

### Atividade 4 – Complete o Quadro

Situação	Tipo correto
----------	--------------

Serve apenas como modelo base	Classe abstrata
Não pode ser herdada	Classe final
Não pode ser instanciada	Classe abstrata
Método que deve ser implementado nas filhas	Método abstrato
Método que não pode ser sobrescrito	Método final

## Atividade 5 – Prática em PHP

Crie uma classe abstrata chamada `Veiculo` com os atributos `modelo`, `ano` e um método abstrato `mover()`.

Crie duas subclasses (`Carro` e `Bicicleta`) que implementem o método `mover()` com mensagens diferentes.

No final, instancie ambas as classes e chame o método `mover()`.

*Link do github anexado na atividade.*