

HERANÇA E POLIMORFISMO

IFBA – Campus Ilhéus
Curso: Back End com Python

Objetivo da Aula

- Explicar os conceitos de herança e polimorfismo para reutilização e extensão de código.

Herança

- É um mecanismo que permite que uma classe (filha) herde atributos e métodos de outra classe (mãe).
- Assim, há o **reaproveitamento de código**, evitando repetições e criando uma **hierarquia de classes** mais clara.

Herança

⦿ Vantagens:

- Reutilização de código.
- Organização do sistema.
- Facilidade para manter e estender funcionalidades.

Exemplo 1 — Classe Mãe e Subclasse

Classe Pai

class Pessoa:

def __init__(self, nome, idade):

self.nome = nome

self.idade = idade

def apresentar(self):

print(f"Olá! Meu nome é {self.nome} e
tenho {self.idade} anos.")

Exemplo 1 — Classe Mãe e Subclasse

```
# Subclasse (Classe Filha)
class Aluno(Pessoa):
    def __init__(self, nome, idade, matricula):
        # reaproveita o construtor da classe pai
        super().__init__(nome, idade)
        self.matricula = matricula

    def estudar(self):
        print(f"{self.nome} está estudando...")
```

Exemplo 1 — Classe Mãe e Subclasse

```
# Programa principal
```

```
aluno1 = Aluno("Maria", 20, "A123")
```

```
aluno1.apresentar() # método herdado
```

```
aluno1.estudar()    # método da subclasse
```

Sobrescrita de Métodos (Override)

- ⦿ A **subclasse** pode **redefinir** um método herdado da classe pai para **alterar** ou **especializar** o **comportamento**.

Exemplo 2 — Sobrescrita

```
class Pessoa:  
    def falar(self):  
        print("A pessoa está falando algo.")
```

```
class Professor(Pessoa):  
    def falar(self): # sobrescreve o método da  
        classe pai  
        print("O professor está explicando a  
        matéria.")
```

Exemplo 2 — Sobrescrita

```
# Programa principal
```

```
p1 = Pessoa()
```

```
p2 = Professor()
```

```
p1.falar()
```

```
p2.falar()
```

Polimorfismo

- ⦿ **Polimorfismo** significa “**muitas formas**”.
- ⦿ Na prática, é quando **métodos com o mesmo nome se comportam de maneira diferente**, dependendo do **objeto** que os chama.

Exemplo 3 — Polimorfismo

```
class Animal:  
    def emitir_som(self):  
        print("O animal faz algum som.")
```

```
class Cachorro(Animal):  
    def emitir_som(self):  
        print("O cachorro late: Au au!")
```

```
class Gato(Animal):  
    def emitir_som(self):  
        print("O gato mia: Miau!")
```

Exemplo 3 — Polimorfismo

```
# Programa principal
```

```
animais = [Cachorro(), Gato(), Animal()]
```

```
for a in animais:
```

```
    a.emitir_som()
```

```
# Saída:
```

```
# O cachorro late: Au au!
```

```
# O gato mia: Miau!
```

```
# O animal faz algum som.
```

Resumo da Aula

Conceito	O que é	Exemplo
Herança	Uma classe herda atributos e métodos de outra	<pre>class Aluno(Pessoa)</pre>
<code>super()</code>	Chama métodos da classe pai	<pre>super().__init__()</pre>
Sobrescrita (Override)	Redefinir método herdado	<pre>def falar(self): ...</pre>
Polimorfismo	Mesmo método com comportamentos diferentes	<pre>a.emitir_som()</pre> em classes distintas

Exercício

- ⦿ A classe `Funcionario` é a classe pai, com o método genérico `calcular_pagamento()`.
- ⦿ As classes `Horista` e `Mensalista` herdam da classe `Funcionario`.
- ⦿ Cada subclasse sobreescreve o método `calcular_pagamento()` com sua própria lógica.
- ⦿ No laço `for`, o mesmo comando `f.calcular_pagamento()` executa comportamentos diferentes conforme o tipo do funcionário — isso é polimorfismo.

Exercício - continuação

⦿ Classe base (superclasse):

Funcionario:

- Deve conter o método `calcular_pagamento()`.
- Esse método **não precisa fazer nada específico** (pode apenas imprimir uma mensagem genérica ou retornar 0).
- Serve de **modelo** para as subclasses.

Exercício - continuação

⦿ Subclasse Horista

- Atributos: horas_trabalhadas e valor_hora.
- O método calcular_pagamento() deve retornar horas_trabalhadas * valor_hora.

⦿ Subclasse Mensalista

- Atributo: salario_fixo.
- O método calcular_pagamento() deve retornar o valor de salario_fixo.

Exercício - continuação

⦿ Programa principal:

- Crie uma lista com objetos de ambos os tipos (Horista e Mensalista).
- Percorra a lista e chame o método `calcular_pagamento()` para cada funcionário.
- Observe que o mesmo método se comporta de forma diferente, dependendo do tipo do funcionário.