Programação Orientada a Objetos

Herança e Polimorfismo

Profa. Andriele Busatto do Carmo

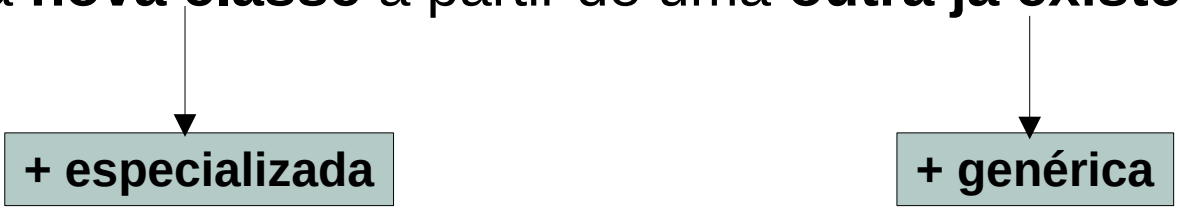
acarmo@unisinos.br

UNISINOS

Herança

Herança

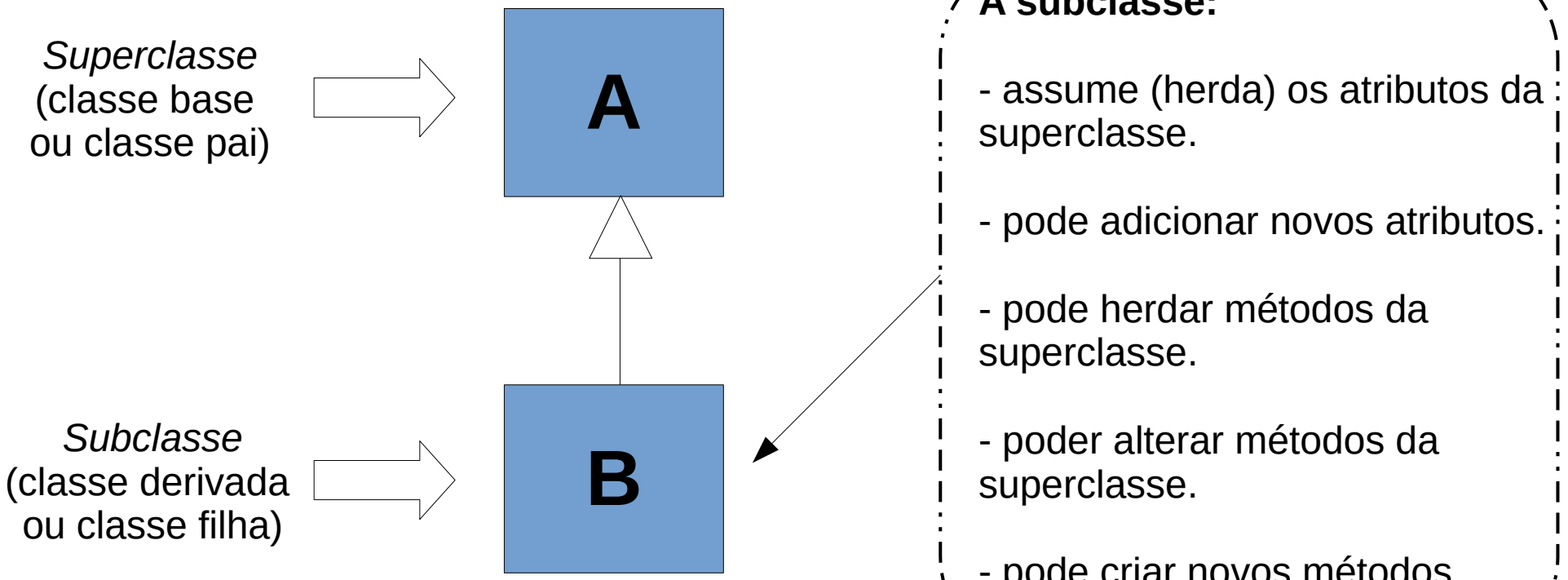
- Ideia:
  - derivar uma **nova classe** a partir de uma **outra já existente**.



Classe original: **SUPERCLASSE**
Classe derivada: **SUBCLASSE**

- Por que o nome herança?
  - Porque a **subclasse HERDA** características da superclasse.

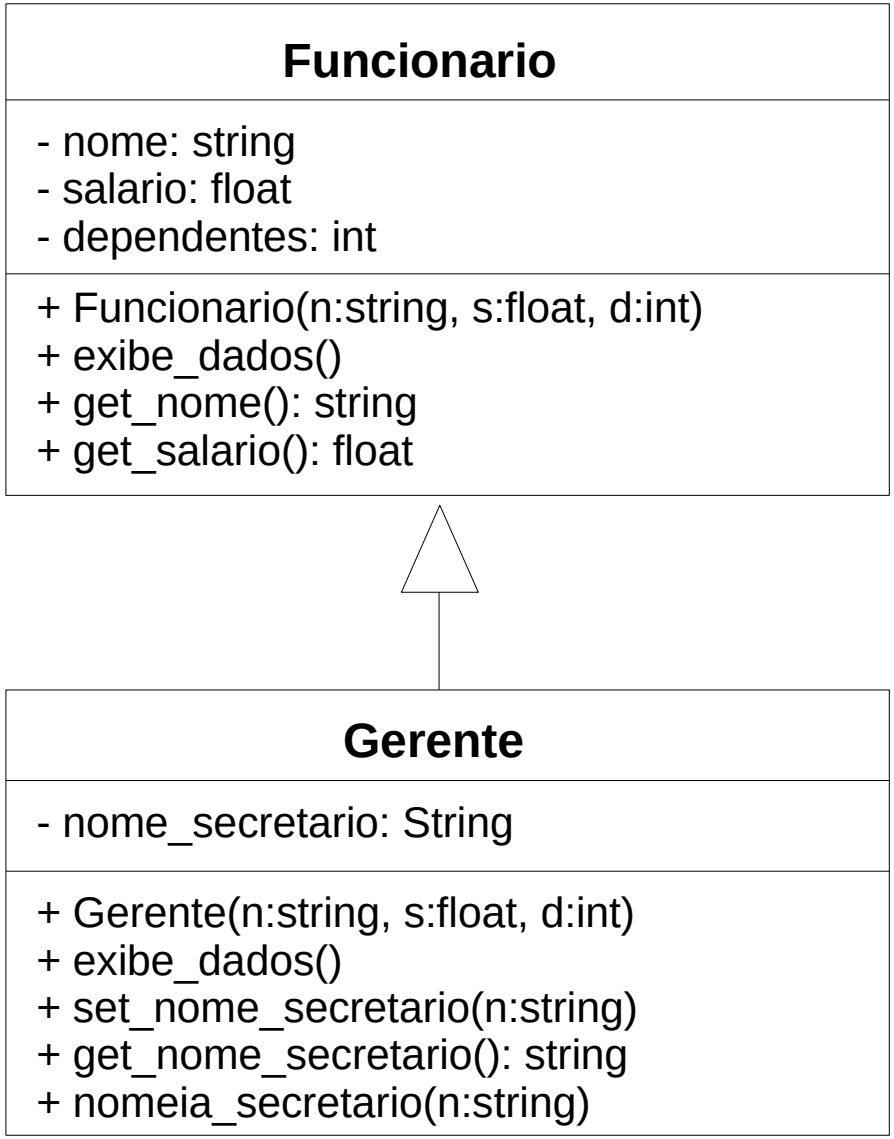
Herança



Todo objeto da subclasse B também É UM objeto da superclasse A

Logo, a subclasse B apresenta caracterísitcas **ADICIONAIS** e **EXCLUSIVAS** em relação àquelas da superclasse (por isso é mais especializada).

Herança



- **Gerente** é um tipo especial de Funcionário, com características adicionais e exclusivas.
- O método exibe_dados() é **SOBRESCRITO**, ou seja, **REDEFINIDO**.
- O método nomeia_secretario() é **EXCLUSIVO** de Gerente.

Herança

- Como dizer que uma classe HERDA outra em Python?
  - Utilizando o nome da superclasse entre parênteses na definição da subclasse.

- Exemplo:

```
class Gerente(Funcionario):
    .
    .
    .
```

Atributos da subclasse

- A subclasse HERDA todos os atributos públicos e protegidos da superclasse. Objetos da subclasse terão todos esses atributos (ex.: *nome*, *salario*, *dependentes*).

- A subclasse PODE ter atributos adicionais exclusivos. Só objetos da subclasse terão tais atributos (ex.: *nome_secretario*).

Métodos da subclasse

- A subclasse PODE sobrescrever métodos da superclasse (ex.: *exibe_dados()*) – mesma assinatura (nome e parâmetros) e o mesmo tipo de retorno, mas implementação diferente (não confundir com **sobrecarga**).

- Todo método da superclasse que não é sobrescrito na subclasse é herdado por esta (ex.: *get_nome()*).

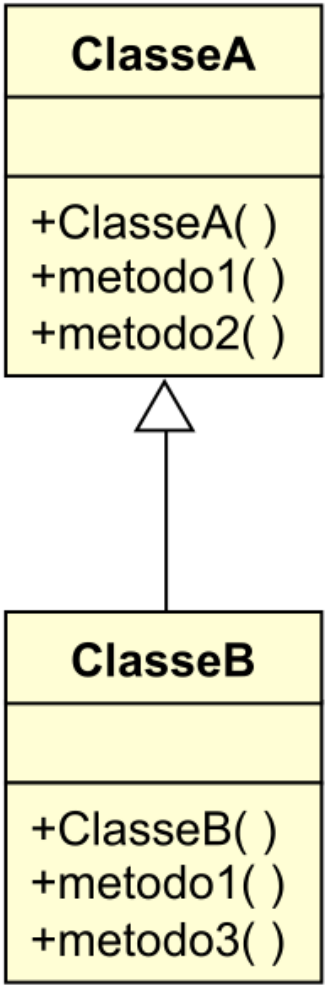
- A subclasse PODE ter métodos novos, exclusivos, que não aparecem na superclasse (ex.: *nomeia_secretario()*).

Chamando métodos da subclasse

- Quando um objeto da subclasse chama um método, temos:
  - se o método foi **sobrescrito** pela subclasse, o da subclasse é acionado;
  - se o método só existe na **superclasse**, o da superclasse será acionado;
  - se o método é **exclusivo da subclasse**, este será acionado;
  - se o método **não existe** nem na superclasse e nem na subclasse, ocorre um erro.

Responda:

- O que vai ocorrer em cada uma das linhas abaixo?




```
b = ClasseB()

b.metodo1()

b.metodo2()

b.metodo3()

b.metodo4()
```

Construtores da subclasse

- Os construtores da superclasse não são herdados pela subclasse.
- Cada subclasse deve ter seu(s) próprio(s) construtor(es) para inicializar seus atributos.
- Mas, ao instanciar um objeto da subclasse, devemos inicializar os atributos que ela está herdando.
- O construtor da subclasse faz isso chamando o construtor da superclasse com a chamada **`super()`**.

Construtores da subclasse

- Exemplo:

```
def __init__(self, n: String, s: double, d: int):
    super().__init__(n, s, d) #chama o construtor da superclasse
    nome_secretario = “Petrônio”
```

Herança

- **Quais atributos de sua superclasse uma subclasse pode acessar?**
  - Os atributos do tipo **privado** são acessíveis diretamente apenas para a classe que os possuir. Qualquer outra classe somente tem acesso a estes atributos utilizando os métodos de acesso públicos (**get**, por exemplo).

Herança

- **Quais atributos de sua superclasse uma subclasse pode acessar?**
  - Logo, suponha o método `exibe_dados()` na classe `Gerente`:

```
def exibe_dados(self) -> None:
    print(f“Nome: {nome}”) #ERRO
    print(f“Salário: {salario}”) #ERRO
    print(f“Dependentes: {dependentes}”) #ERRO
    print(f“Secretário: {nome_secretario}”)
```

Herança

- **Quais atributos de sua superclasse uma subclasse pode acessar?**
  - Porém, podemos chamar o método `exibe_dados()` da superclasse:

```
def exibe_dados(self) -> None:
    super().exibe_dados()
    print(f“Secretário: {nome_secretario}”)
```

  - Outra opção é definir os atributos da superclasse como **protegidos**. Um atributo **protegido** é acessível à classe e às suas subclasses.

Herança

- **Sobrescrita:**
  - Um método de uma subclasse **sobrescreve** um da sua superclasse se tem exatamente a mesma assinatura (nome, parâmetros e tipos de retorno) do método da superclasse (tem apenas outra implementação).
  - Não pode haver sobrescrita de métodos na mesma classe ou sobrescrita de construtores.

Herança

- Exemplo no Python:

```python
class Caixa:
    def __init__(self, largura = 0, altura = 0):
        self.largura = float(largura)
        self.altura = float(altura)

    def setLargura(self, largura):
        self.largura = float(largura)

    def setAltura(self, altura):
        self.altura = float(altura)

    def getLargura(self):
        return self.largura

    def getAltura(self):
        return self.altura
```

```python
class CaixaCor(Caixa):
    def __init__(self, largura = 0, altura = 0, cor = 0):
        super().__init__(largura, altura)
        self.cor = cor

    def setCor(self, cor):
        self.cor = cor

    def getCor(self):
        return self.cor
```

Herança

- Exemplo no Python:

```
if __name__ == "__main__":
    caixa1 = Caixa()
    caixa1.setLargura(10)
    caixa1.setAltura(20)

    caixa2 = Caixa(10.30)
    caixa2.setAltura(20.40)

    caixa3 = CaixaCor(100, 100.10)
    caixa3.setCor(3)
```

```
print("########################")
print("Caixa 1:")
print("Largura: {}" .format(caixa1.getLargura()))
print("Altura: {}" .format(caixa1.getAltura()))

print("########################")
print("Caixa 2:")
print("Largura: {}" .format(caixa2.getLargura()))
print("Altura: {}" .format(caixa2.getAltura()))


print("########################")
print("Caixa 3:")
print("Largura: {}" .format(caixa3.getLargura()))
print("Altura: {}" .format(caixa3.getAltura()))
print("Cor: {}" .format(caixa3.getCor()))
```

Polimorfismo

Polimorfismo

- **Ideia:**
  - Polimorfismo é a realização de uma tarefa de formas diferentes (*poli* – muitas; *morphos* – formas).
  - Utilização importante: nas hierarquias de herança.
  - Um objeto da superclasse e outro da subclasse respondem diferentemente à mesma chamada de um método.

Polimorfismo

- **Conceito:**
  - Sendo B derivado de A, todos os membros disponíveis em A estarão em B.
  - B é um super-conjunto de A: todas as operações que podem ser feitas com objetos de A também podem ser feitas com por meio de objetos de B.
  - Um objeto da classe B também é um objeto da classe A: isso significa a possibilidade de converter um objeto B para A.

Polimorfismo

- **Exemplo:**
  - Numa academia, os atletas são categorizados pelo peso, conforme tabela da esquerda. Os lutadores, porém, têm categorias definidas pela da direita. Mas, é claro que todo lutador **é um** atleta.

Atleta

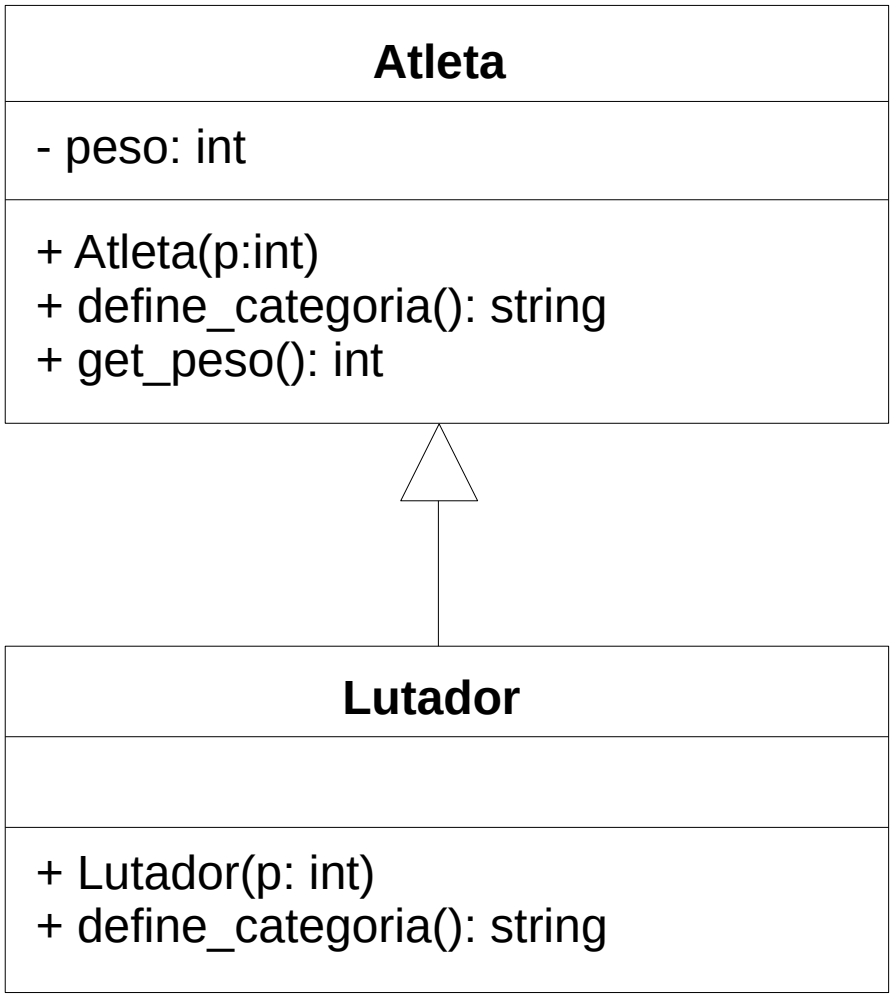
Faixa de Peso	**Categoria**
até 50kg	Infantil
acima de 50, até 65kg	Juvenil
acima de 65kg	Adulto

Lutador

Faixa de Peso	**Categoria**
até 54kg	Pluma
acima de 54, até 60kg	Leve
acima de 60, até 75kg	Meio-leve
acima de 75kg	Pesado

Polimorfismo

- **Exemplo:**



Polimorfismo

- **Exemplo:**

```
from random import randint

def main():
        n = int(input(“Informe a quantidade de atletas: ”))

        for i in range(n):
            p = int(input(“Peso: ”)
            r = randint(0,9)

            if(r % 2 == 0)
                a = Atleta(p)
            else
                a = Lutador(p)

            print(f“Categoria: {a.define_categoria()}”)


if __name__ == ‘__main__’:
    main()
```

Referências

- Material construído com base nos originais de:
  - Programação I
    - Professor Aníbal Cardoso (Unisinos)
  - Programação Orientada a Objetos
    - Professor Márcio Garcia Martins (Unisinos)