



Programação Orientada a Objetos

Herança

Prof. Paulo Vinicius Vieira
paulo.vieira@faculdadeimpacta.com.br

Herança

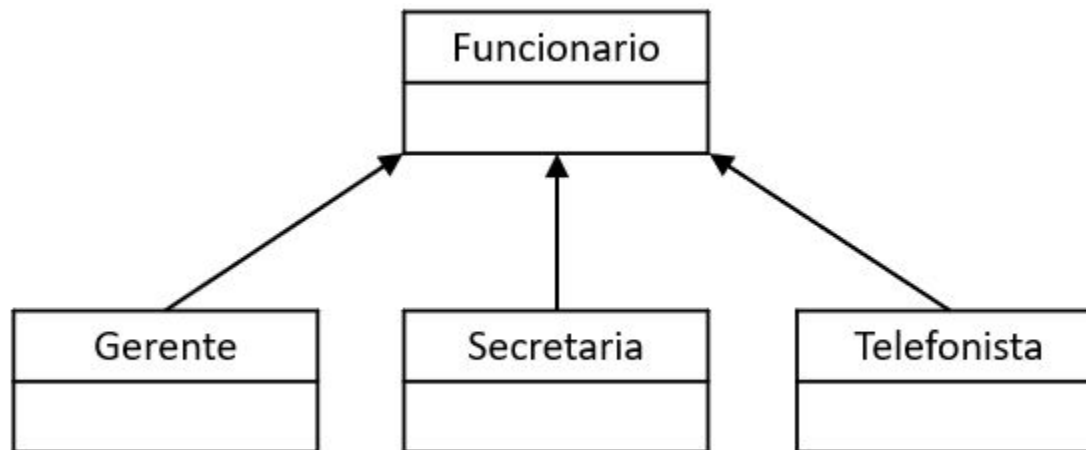
- A Programação Orientada a Objetos se baseia em 4 pilares:
 - Abstração:
 - Processo de representar entidades do mundo real
 - Encapsulamento:
 - Permite Ocultar detalhes internos de uma classe
 - Herança:
 - Facilita a reutilização de código de uma classe
 - Polimorfismo:
 - Adicionam a possibilidade de alteração no funcionamento interno de objetos

Herança

- Herança é o princípio da programação orientada a objetos que permite criar uma nova classe a partir de uma já existente.
 - A herança permite que uma classe **herde** atributos e métodos **públicos** de outra classe.
- Principal vantagem: **reutilização do código**.
 - Atributos e métodos de uma classe serão iguais para as outras.
 - Pode se agrupar as características comuns em um classe mais genérica e, a partir dela, gerar classes mais específicas.

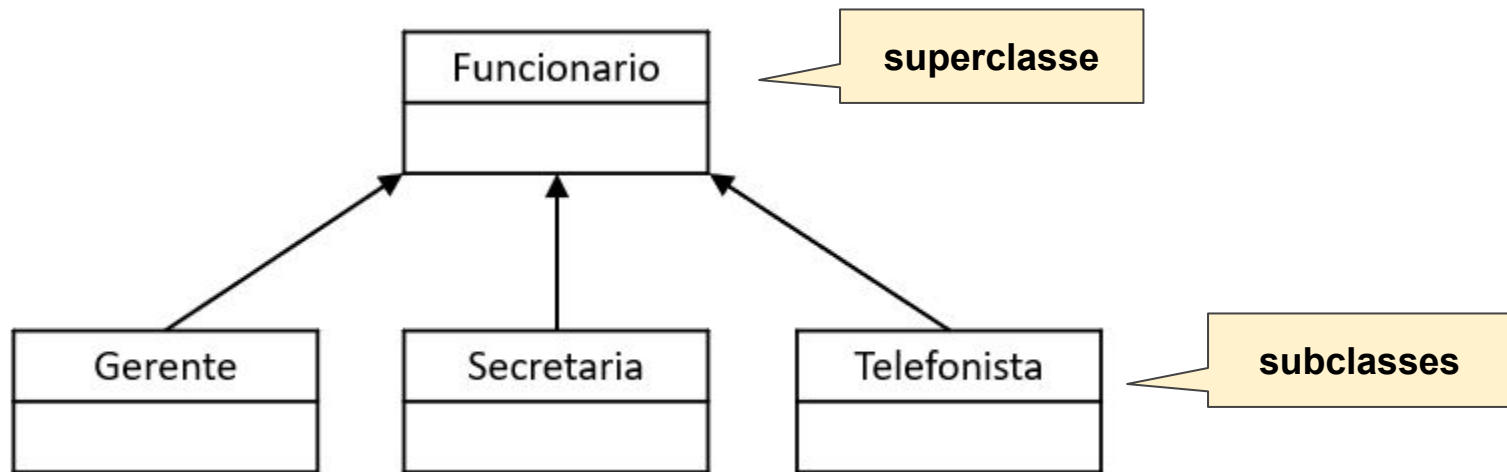
Herança

- A herança permite criar uma **hierarquia de classes**
 - A classe mais genérica é chamada de **superclasse**, ou **classe mãe**, ou **classe base**.
 - As classes mais específicas herdam as características da superclasse e são chamadas de **subclasse**, ou **classe filha**, ou **classe derivada**.



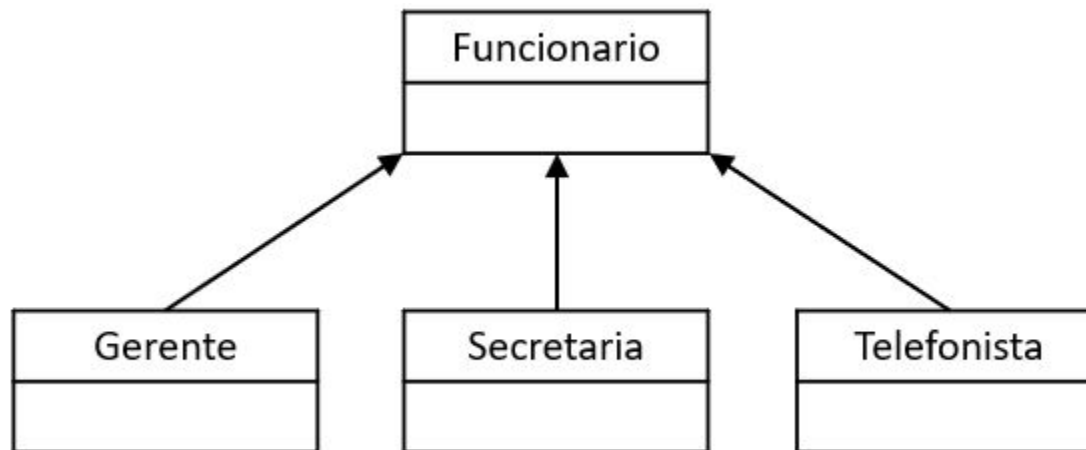
Herança

- A herança permite criar uma **hierarquia de classes**
 - A classe mais genérica é chamada de **superclasse**, ou **classe mãe**, ou **classe base**.
 - As classes mais específicas herdam as características da superclasse e são chamadas de **subclasse**, ou **classe filha**, ou **classe derivada**.



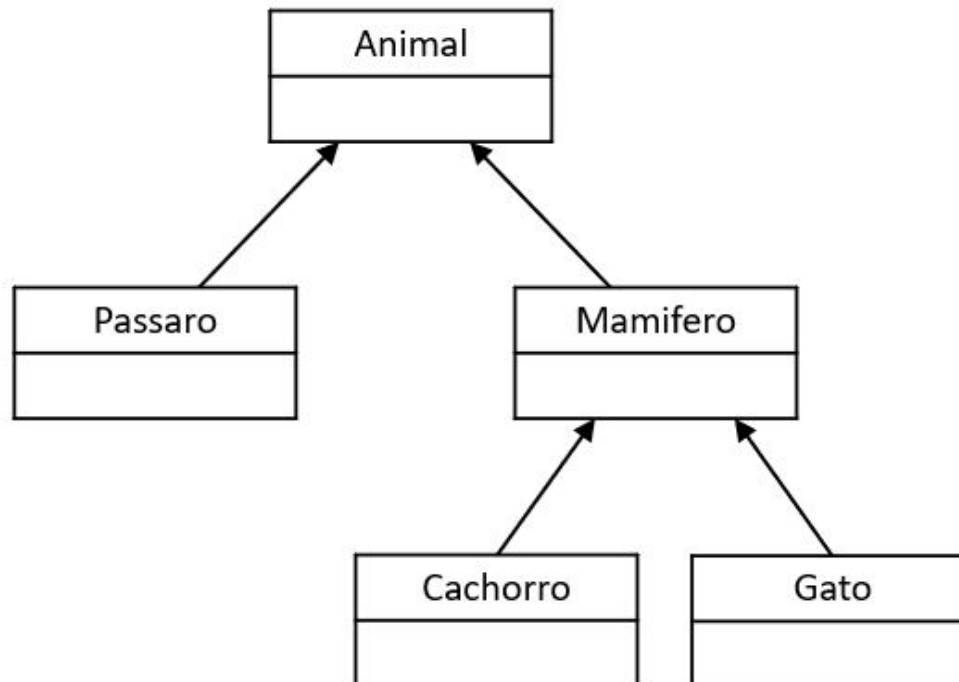
Herança

- Diagrama de Classes
 - A relação de herança é indicada por uma linha com um triângulo em uma das extremidades, sempre apontando para a superclasse



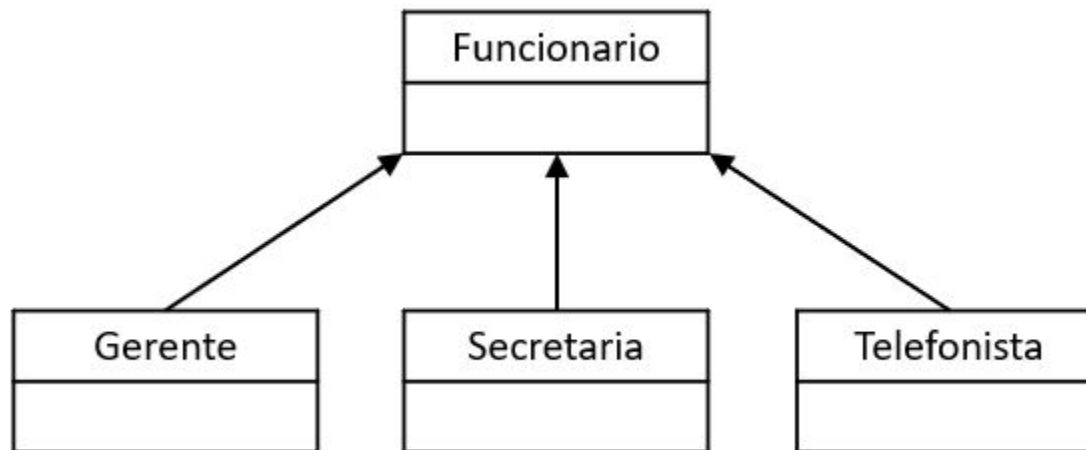
Herança

- Diagrama de Classes
 - Uma hierarquia de classes pode ter vários níveis.



Herança

- Para aplicar a herança não basta que as classes tenham características iguais
 - É necessário que a subclasse seja um tipo específico da superclasse
 - Podemos utilizar o teste conhecido como “**É UM**”.
 - Ajuda a detectar conceitualmente se a herança foi aplicada de forma correta
 - A classe Gerente herda da classe Funcionário, porque gerente **É UM** tipo específico de funcionário.



Herança

- Generalização:
 - Consiste em identificar atributos e métodos iguais em diferentes classes e agrupá-los em uma superclasse
 - Exemplo: Considere as classes Ave e Cachorro

Ave
Nome
Cor
Numero de pés
Cor do bico
Tipo de pé
Andar
Voar
Piar
Por ovos
Beber
comer

Cachorro
Nome
Cor
Numero de pés
Comprimento do pêlo
Comprimento do rabo
Andar
Correr
Pular
Farejar
Beber
Comer

Herança

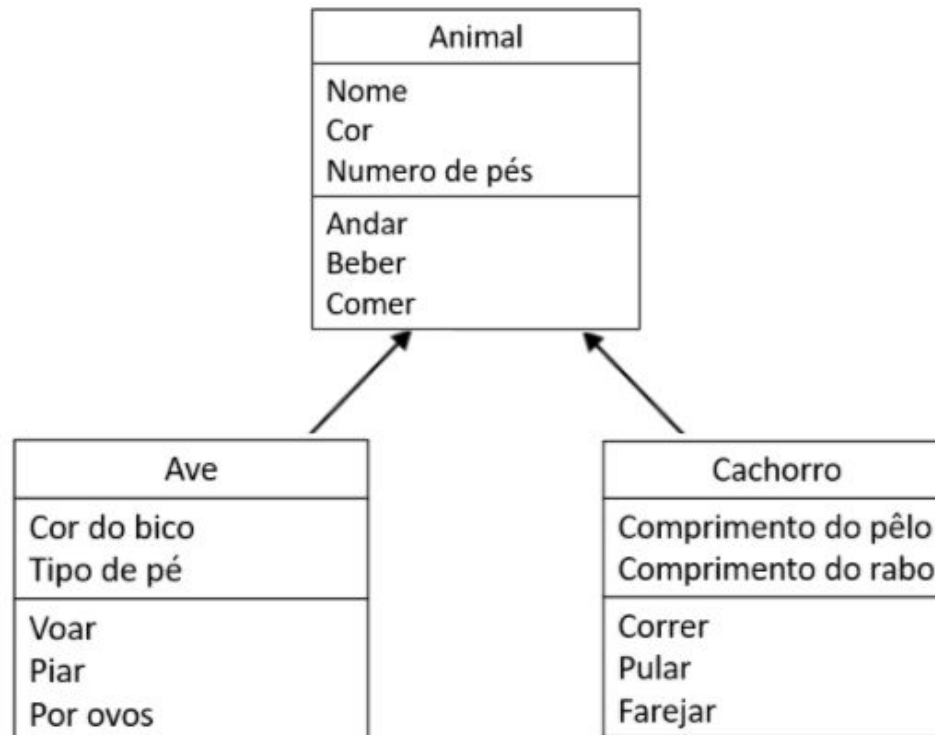
- Generalização:
 - Consiste em identificar atributos e métodos iguais em diferentes classes e agrupá-los em uma superclasse
 - Exemplo: Considere as classes Ave e Cachorro
 - As classes possuem alguns atributos e métodos iguais, e outros diferentes

Ave
Nome Cor Numero de pés Cor do bico Tipo de pé
Andar Voar Piar Por ovos Beber Comer

Cachorro
Nome Cor Numero de pés Comprimento do pêlo Comprimento do rabo
Andar Correr Pular Farejar Beber Comer

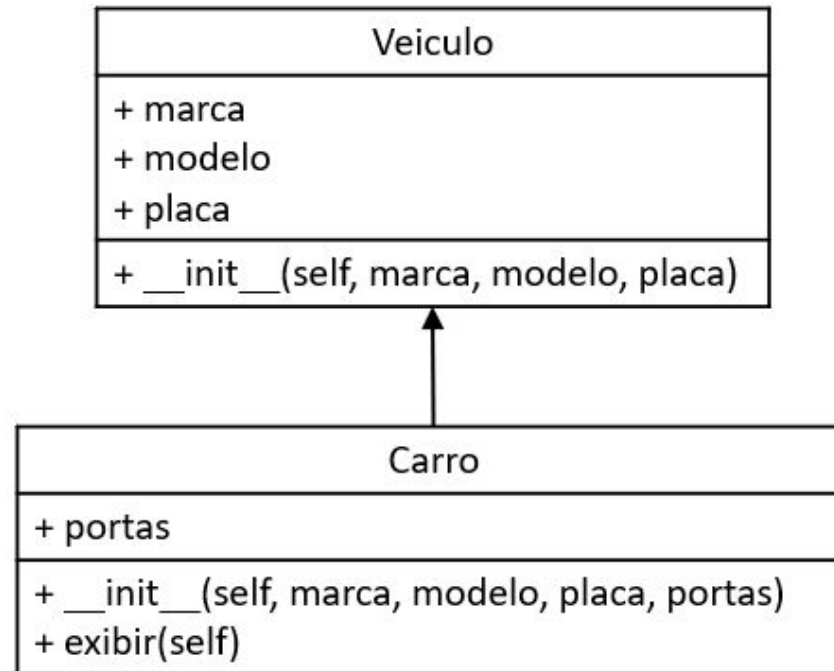
Herança

- Generalização:
 - Podemos agrupar o que é igual em uma superclasse, e manter o que é diferente nas subclasses



Herança

- Exemplo



Herança

```
class Veiculo:
    def __init__(self, marca, modelo, placa):
        self.marca = marca
        self.modelo = modelo
        self.placa = placa
```

Herança

```
class Veiculo:
    def __init__(self, marca, modelo, placa):
        self.marca = marca
        self.modelo = modelo
        self.placa = placa
```

```
class Carro(Veiculo):                                # classe Carro herda da classe Veiculo
    def __init__(self, marca, modelo, placa, portas):
        super().__init__(marca, modelo, placa)
        self.portas = portas                         # atributo específico do Carro

    def exibir(self):
        print('-----')
        print('Marca: ', self.marca)
        print('Modelo: ', self.modelo)
        print('Placa: ', self.placa)
        print('Portas: ', self.portas)
```

A função **super()** invoca um método da superclasse.

O construtor da classe Carro está repassando os dados para a superclasse através do **super().__init__**

Herança

```
class Veiculo:
    def __init__(self, marca, modelo, placa):
        self.marca = marca
        self.modelo = modelo
        self.placa = placa
```

```
class Carro(Veiculo):                                # classe Carro herda da classe Veiculo
    def __init__(self, marca, modelo, placa, portas):
        super().__init__(marca, modelo, placa)
        self.portas = portas                          # atributo específico do Carro

    def exibir(self):
        print('-----')
        print('Marca: ', self.marca)
        print('Modelo: ', self.modelo)
        print('Placa: ', self.placa)
        print('Portas: ', self.portas)
```


```
# Programa Principal
carro1 = Carro("Ford", "Ka", "AAA-1234", 4)
carro1.exibir()
```

Herança

```
class Veiculo:
    def __init__(self, marca, modelo, placa):
        self.marca = marca
        self.modelo = modelo
        self.placa = placa
```

```
class Carro(Veiculo):
    # classe Carro herda da classe Veiculo
    def __init__(self, marca, modelo, placa, portas):
        super().__init__(marca, modelo, placa)
        self.portas = portas
        # atributo específico do Carro

    def exibir(self):
        print('-----')
        print('Marca: ', self.marca)
        print('Modelo: ', self.modelo)
        print('Placa: ', self.placa)
        print('Portas: ', self.portas)
```



```
# Programa Principal
carro1 = Carro("Ford", "Ka", "AAA-1234", 4)
carro1.exibir()
```


Herança

```
class Veiculo:
    def __init__(self, marca, modelo, placa):
        self.marca = marca
        self.modelo = modelo
        self.placa = placa
```

```
class Carro(Veiculo):
    # classe Carro herda da classe Veiculo
    def __init__(self, marca, modelo, placa, portas):
        super().__init__(marca, modelo, placa)
        self.portas = portas
        # atributo específico do Carro

    def exibir(self):
        print('-----')
        print('Marca: ', self.marca)
        print('Modelo: ', self.modelo)
        print('Placa: ', self.placa)
        print('Portas: ', self.portas)
```

```
# Programa Principal
carro1 = Carro("Ford", "Ka", "AAA-1234", 4)
carro1.exibir()
```