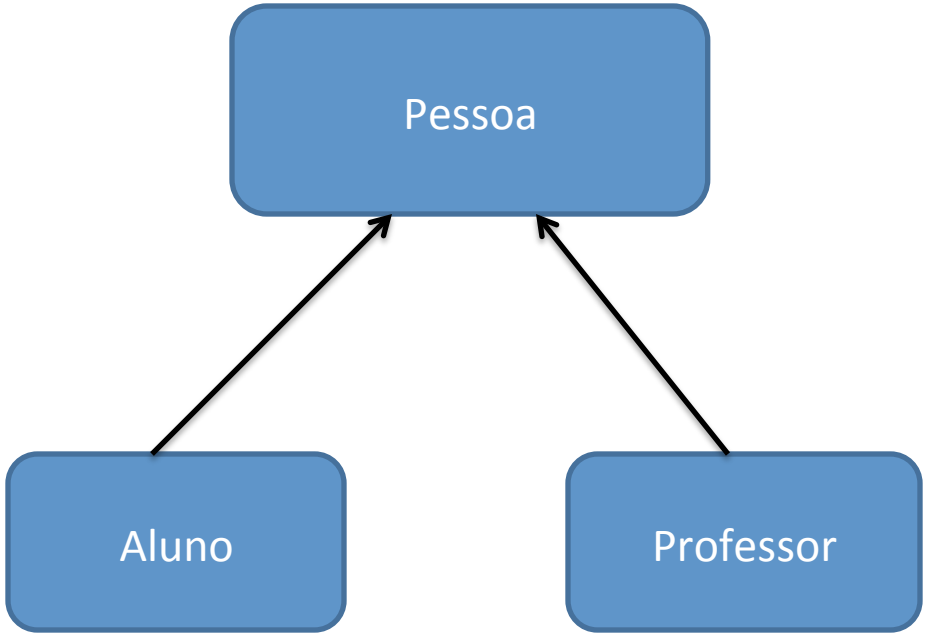
Python – Orientação a Objetos – Parte 3

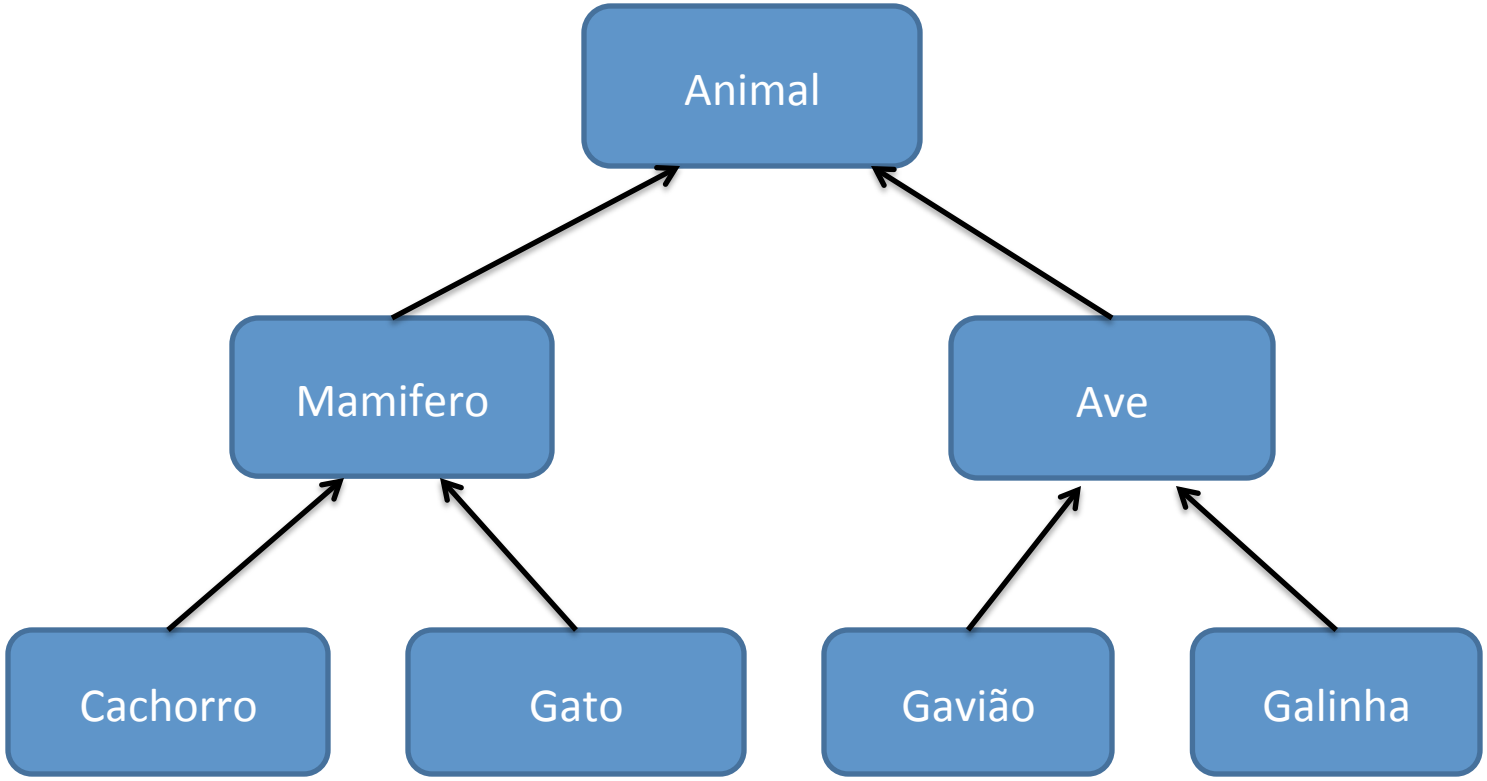
Introdução à Programação

SI1

Herança



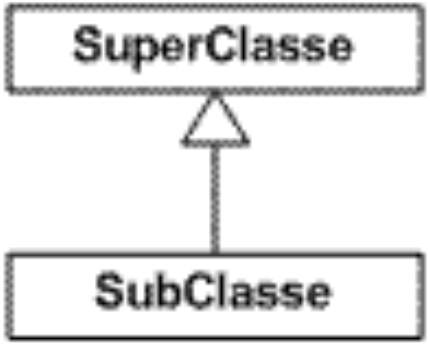
Herança



Herança

- Uma classe pode **herdar** a definição de outra classe:
  - Permite uso ou extensão de métodos e atributos previamente definidos em outra classe
  - Nova classe
    - **Subclasse**
  - Original
    - Classe pai, ancestral ou **superclasse**
- Permite herança múltipla

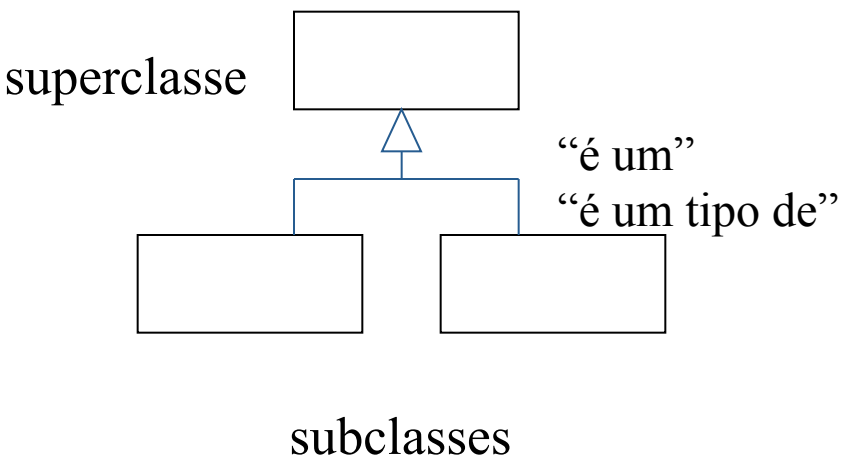
Herança

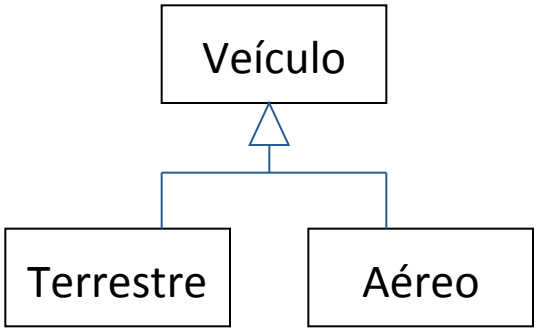


- Uma classe pode ser **definida** a partir de outra já **existente**
- Abstrai classes genéricas (**superclasse**), a partir de classes com propriedades (atributos e operações) **semelhantes**
  - Modelar similaridades entre classes, preservando diferenças
- As **subclasses herdam** todas as **propriedades** de sua **superclasse**
  - E possuem as suas próprias

Herança

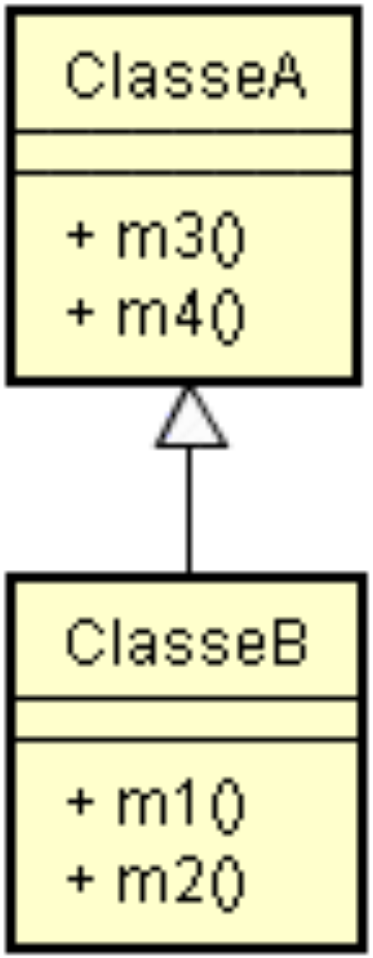
- Relacionamento entre itens gerais (**superclasses**) e itens mais específicos (**subclasses**)





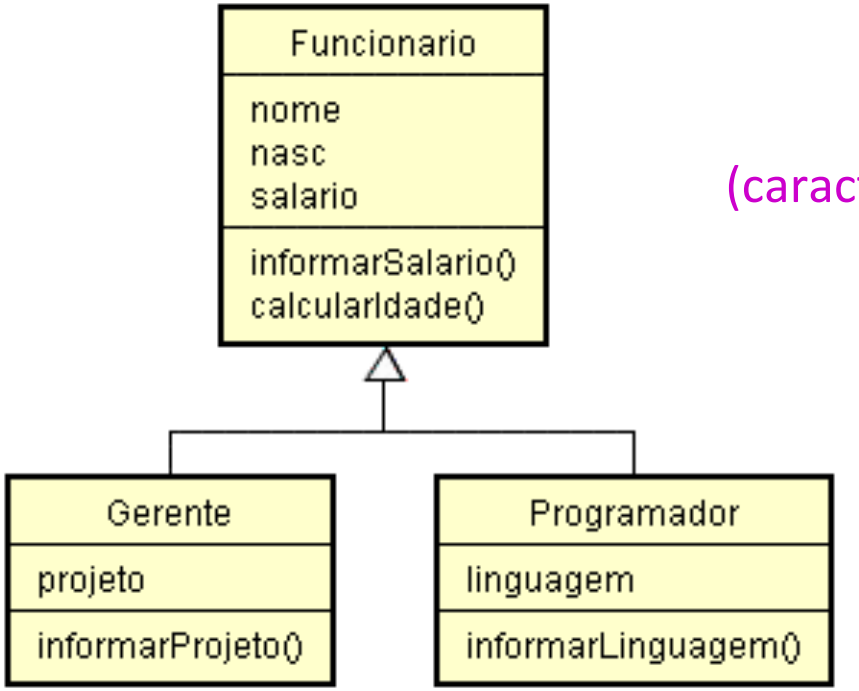
Herança

- Suponha que a classe **ClasseB** herda de **ClasseA**
- Um objeto da **ClasseB** também **é um** objeto da **ClasseA**
- Alterar **m3()** basta modificar a **ClasseA**



Herança

- *Exemplo:*



Superclasse
(características comuns)

Subclasses
(características específicas)

Herança

```
class Veiculo:
    def andar(self):
        print("andei")

class Carro(Veiculo):
    _nrodas = 4
```

```
>>>gol = Carro()
>>>gol.andar()
andei
```

Sobrecarga

- Redefinição de métodos já existente, com o mesmo nome.
- Quando um método da classe pai é redefinido na classe filha
- Diz-se que o método foi sobrecarregado (overloaded).

Redefinindo Métodos

- Você pode redefinir métodos declarados na superclasse

```
class Veiculo:
    def andar(self):
        print("andei")

class Carro(Veiculo):
    _nrodas = 4
    def andar(self):
        print("andei de carro")
```

```
>>>gol = Carro()
>>>gol.andar()
andei de carro
```

Redefinindo Métodos

- Você pode chamar o método da superclasse

```
class Veiculo:
    def andar(self):
        print("andei")

class Carro(Veiculo):
    _nrodas = 4
    def andar(self):
        Veiculo.andar(self)
```

```
>>>gol = Carro()
>>>gol.andar()
andei
```

Construtores

- Construtor da classe filha tem que chamar o da classe pai

```
class Sequence:
    def __init__(self, nome, seq):
        self.nome = nome
        self.seq = seq

class DNA(Sequence):

    def __init__(self, nome, seq):
        Sequence.__init__(self, nome, seq)
```

Chamando construtor da superclasse

```
class Veiculo:
    numPassageiros = None

    def __init__(self, numPassageiros):
        self.numPassageiros = numPassageiros

    def andar(self):
        print("andei")

class Carro(Veiculo):
    _nrodas = None

    def __init__(self, nrodas, numPassageiros):
        Veiculo.__init__(self, numPassageiros)
        self._nrodas = nrodas

    def mostraQtdPassageiros(self):
        print self.numPassageiros
```

Chamada ao Superconstrutor. (Sempre na primeira linha do construtor)

```
>>>gol = Carro(4, 6)
>>>gol.mostraQtdPassageiros()
6
```

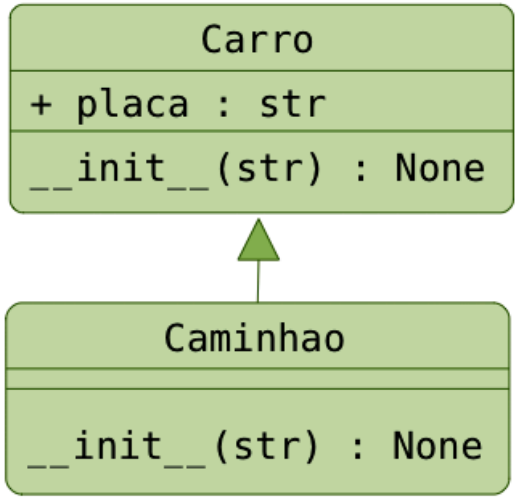
Herança

- Java

```
public class Caminhao extends Carro {
   public Caminhao(String placa) {
      super(placa);
   }
}
```

- Python

```
class Caminhao (Carro):
   def __init__(self, placa):
      Carro.__init__(self, placa)
```



Exemplo

```
class ContaCorrente:
    def __init__(self, numero):
        self.numero = numero
        self.saldo = 0.0

    def creditar(self, valor):
        self.saldo = self.saldo + valor

    def debitar(self, valor):
        self.saldo = self.saldo - valor
```

Exemplo

```
class Poupanca(ContaCorrente):
    #A classe Poupanca tem um atributo
    #taxaJuros que é especifico

    def __init__(self, numero, taxa):
        ContaCorrente.__init__(self, numero)
        self.taxaJuros = taxa

    #E tem tambem um metodo para render taxaJuros
    def renderJuros(self):
        self.saldo = self.saldo + self.taxaJuros*(self.saldo/100)
```

Exemplo

```
>>> p = Poupanca("1234",10)
>>> p.saldo
0.0
>>> p.taxaJuros
10
>>> p.creditar(1500)
>>> p.debitar(300)
>>> p.saldo
1200.0
>>> p.renderJuros()
>>> p.saldo
1320.0
```

Polimorfismo

- Polimorfismo literalmente significa várias formas.
- Em Python, um método é polimórfico se ele tem diferentes implementações numa família de classes
- Ex:
  - O operador '+' é polimórfico → se refere a diferentes operações quando usado, por exemplo, em inteiros e strings

Exemplo

```
class Mamifero:
    def som(self):
        print('emitir um som')

class Homem(Mamifero):
    def som (self):
        print('Oi')

class Cachorro(Mamifero):
    def som(self):
        print('Wufff! Wufff!')

class Gato(Mamifero):
    def som(self):
        print('Meawwww!')

mamifero = Mamifero()
mamifero.som()

animais = [Homem(), Cachorro(), Gato()]
for animal in animais:
    animal.som()
```

Informação sobre classes e instâncias

- Podemos perguntar se um objeto pertence a uma classe:

```
obj1 = Gato()
obj2 = Mamifero()
obj3 = Cachorro()
```

```
>>> isinstance(obj1, Gato)
True
>>> isinstance(obj1, Mamifero)
True
>>> isinstance(obj1, Cachorro)
False
>>> isinstance(obj2, Gato)
False
```

EXERCÍCIOS

Exercícios

1. Crie uma classe chamada Ingresso, que possui um valor em reais e um método imprimeValor()
   - Crie uma classe VIP, que herda de Ingresso e possui um valor adicional. Crie um método que retorne o valor do ingresso VIP (com o adicional incluído)

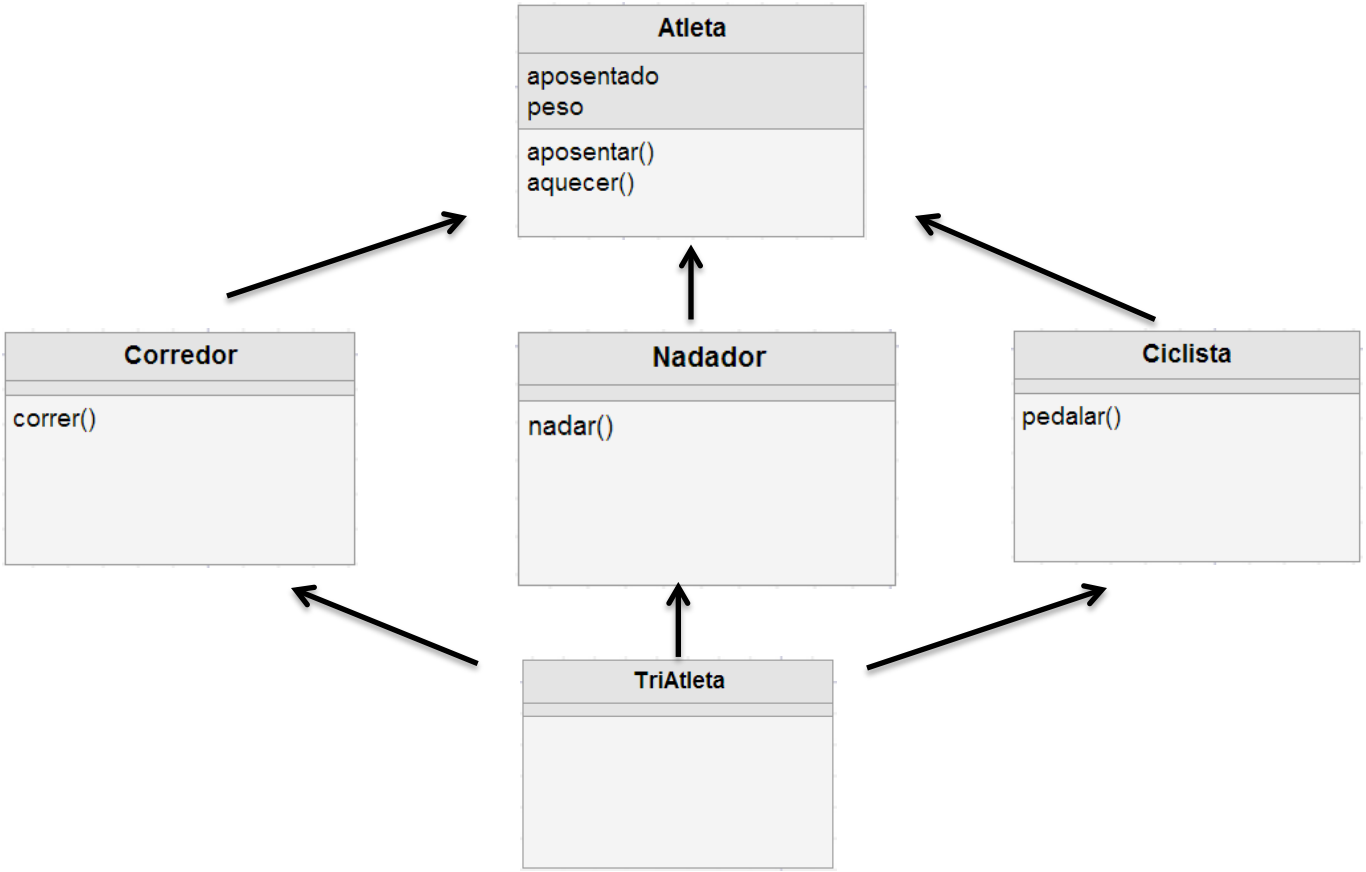
Exercícios

2. Crie uma classe chamada Forma, que possui os atributos area e perimetro.
   - Implemente as subclasses Retangulo e Triangulo, que devem conter os métodos calculaArea e calculaPerimetro. A classe Triangulo deve ter também o atributo altura.

No código de teste crie um objeto da classe Triangulo e outro da Classe Retangulo. Verifique se os dois são mesmo instancias de Forma (use isinstance) , e calcule a área de cada um.

Exercícios

3. Crie um programa que implemente o seguinte diagrama de classes:



Exercícios

4. Considere as classes ContaCorrente e Poupanca apresentadas em sala de aula. Crie uma classe ContaImposto que herda de conta e possui um atributo percentualImposto. Esta classe também possui um método calculaImposto() que subtrai do saldo, o valor do próprio saldo multiplicado pelo percentual do imposto. Crie um programa para criar objetos, testar todos os métodos e exibir atributos das 3 classes (ContaCorrente, Poupanca e ContaImposto).