Paradigmas de Linguagem de Programação em Python

Paradigma Orientado a Objetos: Herança e Polimorfismo

Prof. Henrique Mota

Criando Classes e Atributos

```
class Taxi:
 def __init__(self, modelo, cor, motorista, cidades):
   self.modelo = modelo
   self.cor = cor
   self.motorista = motorista
   self.cidades = cidades
   self.num_passageiros = 0
```

Criando Classes e Atributos

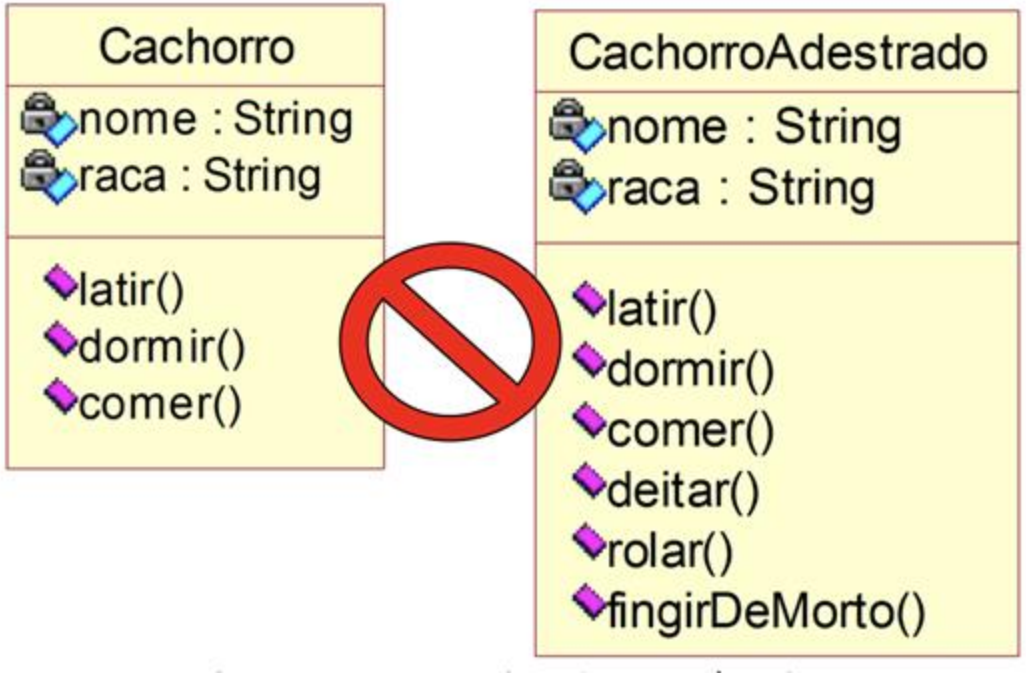
```
class Onibus:
 def __init__(self, modelo, cor, motorista, num_assentos):
   self.modelo = modelo
   self.cor = cor
   self.motorista = motorista
   self.num_assentos = num_assentos
```

Criando Classes e Atributos

```
class Taxi:
 def __init__(self, modelo, cor, motorista, cidades):
   self.modelo = modelo
   self.cor = cor
   self.motorista = motorista
   self.cidades = cidades
   self.num_passageiros = 0


class Onibus:
 def __init__(self, modelo, cor, motorista, num_assentos):
   self.modelo = modelo
   self.cor = cor
   self.motorista = motorista
   self.num_assentos = num_assentos
```

Criando Classes, Atributos

Criando Classes, Atributos [e métodos]



Herança

- Possibilita o **reuso de classes** (código-fonte)
- É uma relação entre uma **classe pai** (***parent class*** → **e.g., Veículo**) e suas classes filhos (***children classes*** → **e.g., Taxi e Ônibus**)
- Utilizado quando:
  - Desejamos **estender** funcionalidades ou características a partir de um tipo de dado (classe) existente
  - Identificamos vários tipos de dados (classes) com características e funcionalidades **comuns**, porém, cada um contendo outras características **particulares**

Herança

- Quando dizemos que:
  - Uma classe B **herda** de uma classe; ou
  - Uma classe B **é subtipo de** uma classe A; ou
  - Uma classe B **é subclasse** de uma classe A; ou
  - Uma classe A **é supertipo** de uma classe B.
- Significa dizer que todos os atributos e métodos que foram definidos em A também fazem parte de B.

Herança
Em Python...

- Cria-se uma *parent class*, como Veiculo, com todos os atributos e métodos em comum

Herança
Em Python...

```
class Veiculo:
 def __init__(self, modelo, cor, motorista):
   self.modelo = modelo
   self.cor = cor
   self.motorista = motorista
```

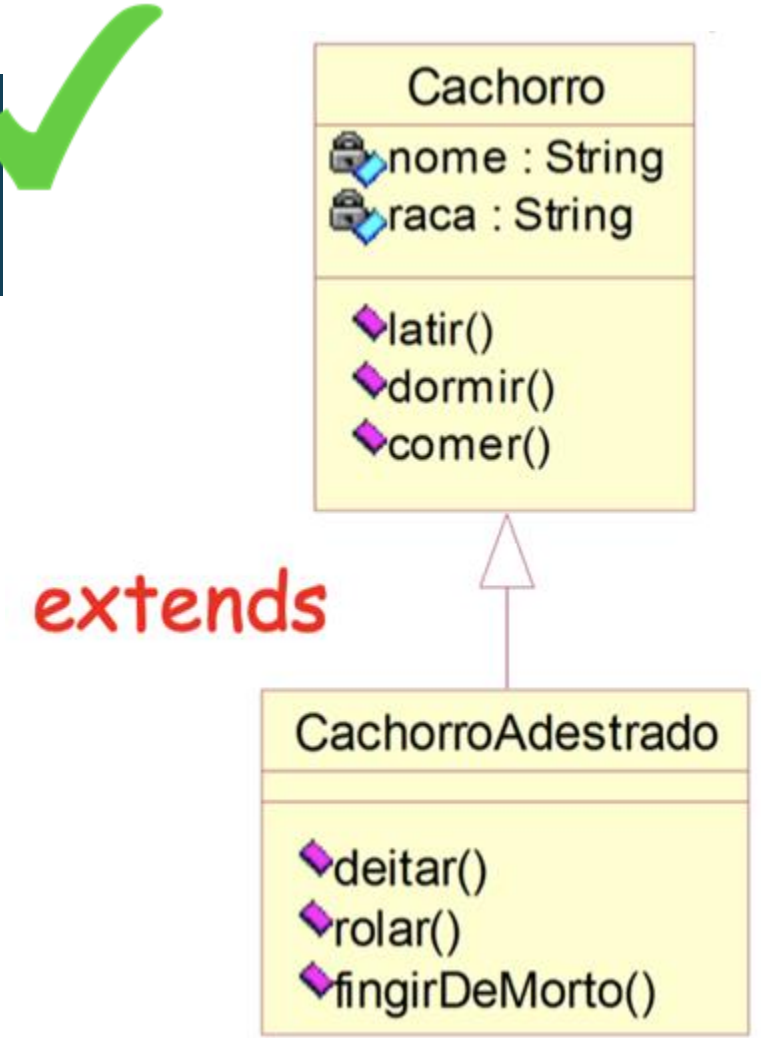
Herança
Em Python...

- Cria-se uma *parent class*, como Veiculo, com todos os atributos e métodos em comum
- Cria-se as *children classes*, como Taxi e Onibus, estendendo os atributos e métodos
  - Explicita-se a definição da *parent class* em cada *child*
    - e.g., `class Onibus(Veiculo):`
  - Define-se os atributos da parent class através do uso do **`super()`**

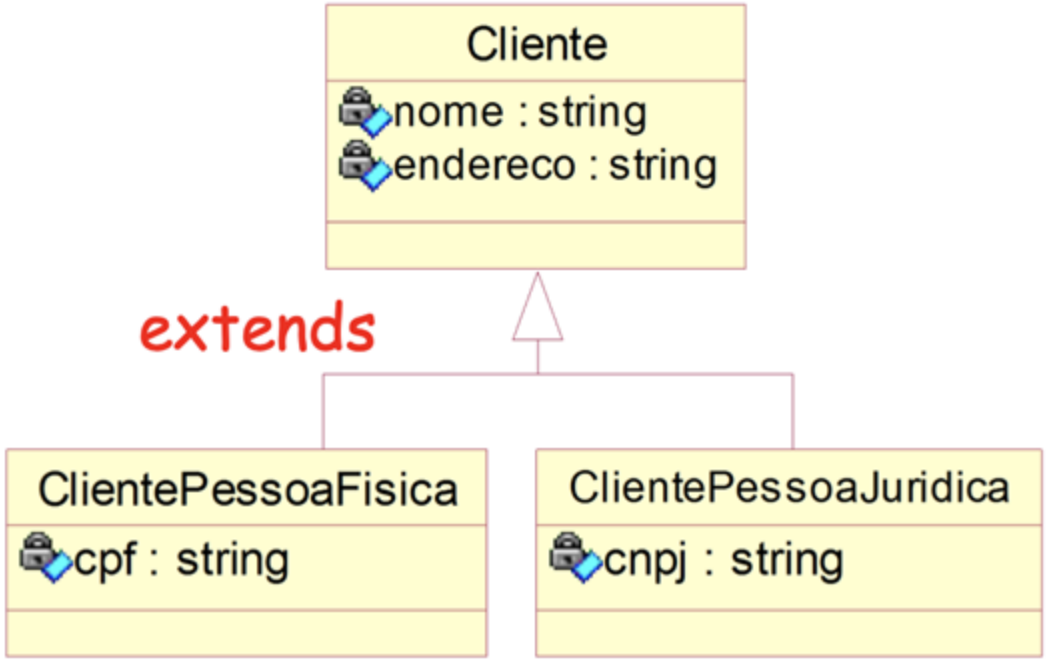
Herança
Em Python...

```
class Veiculo:
 def __init__(self, modelo, cor, motorista):
   self.modelo = modelo
   self.cor = cor
   self.motorista = motorista

class Onibus(Veiculo):
 def __init__(self, modelo, cor, motorista, num_assentos):
   super().__init__(modelo, cor, motorista)
   self.num_assentos = num_assentos

class Taxi(Veiculo):
 def __init__(self, modelo, cor, motorista, cidades):
   super().__init__(modelo, cor, motorista)
   self.cidades = cidades
   self.num_passageiros = 0
```

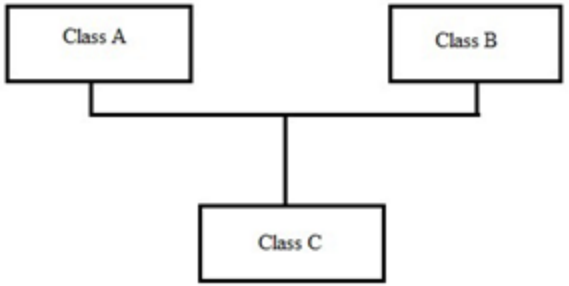
Herança


Cachorro
nome : String
raca : String
latir()
dormir()
comer()
extends
CachorroAdestrado
deitar()
rolar()
fingirDeMorto()

Herança


Cliente
nome : string
endereco : string
extends
ClientePessoaFisica
cpf : string
ClientePessoaJuridica
cnpj : string

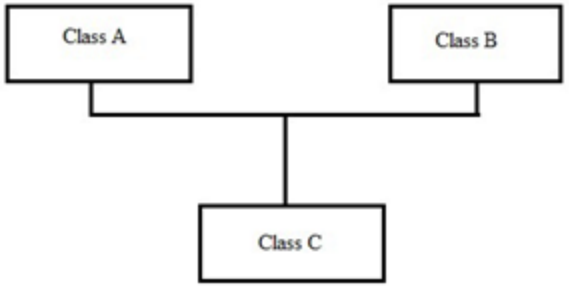
Herança Multipla



- É a capacidade de herdar características de mais de uma classe → **Class C** herdaria características de **Class A** e **Class B**
- Embora a herança múltipla seja um recurso presente em diversas linguagens de programação, o seu uso pode facilmente se tornar um problema

  → Diamante da Morte

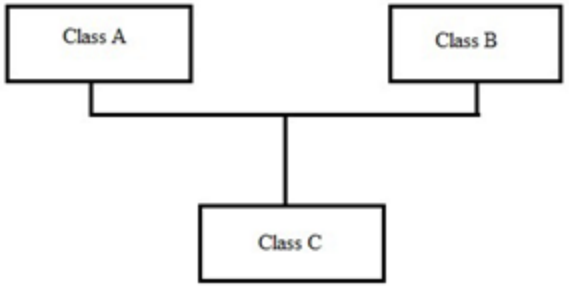
Herança Multipla



→ Diamante da Morte

```
class Ingles:
 def cumprimentar(self):
   print("Hi!")

class Portugues:
 def cumprimentar(self):
   print("Ola!")

class Bilingue(Ingles, Portugues)
 pass

pessoa = Bilingue()
pessoa.cumprimentar()
```

Herança Multipla

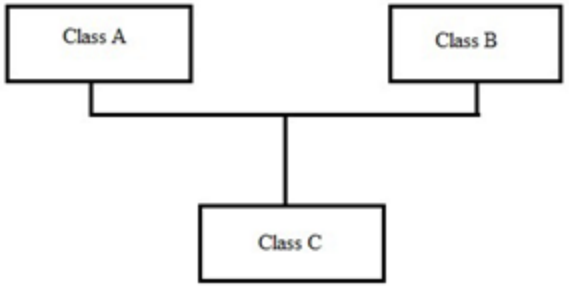


- **Problema:** Diamante da Morte
- **Solução do Python:** definição de prioridade da esquerda para a direita

```
class Bilingue(Ingles, Portugues)
 pass

pessoa = Bilingue()
pessoa.cumprimentar()
```

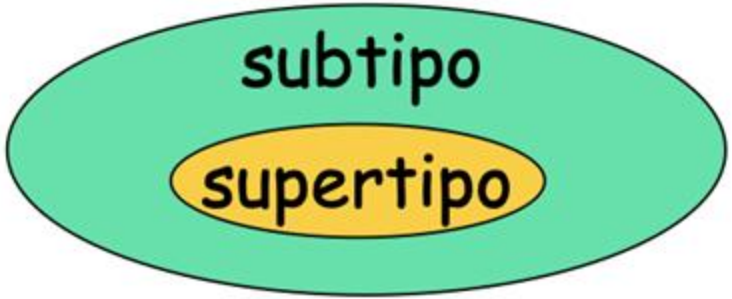
- No exemplo, inglês tem prioridade, e aparecerá o print "Hi!"

Herança Multipla
Conclusão



- Por convenção, programadores utilizam herança múltipla de forma graular, utilizando o conceito de Mixins
- Mesmo assim, **evitem utilizar herança múltipla...exceto em casos que é estritamente necessário (e após verificar que a linguagem trata esses potenciais conflitos)**

Herança e Polimorfismo

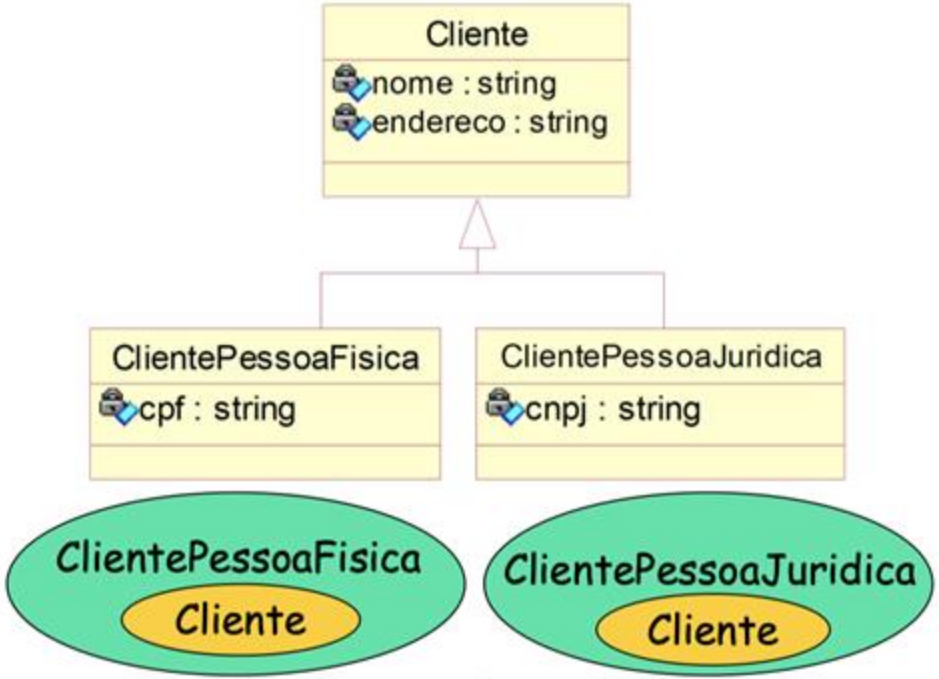


- Qualquer elemento, tal como um **atributo, variável** ou **parâmetro de método,** que pode referenciar **valores de tipos diferentes** durante o curso de execução de um programa pode ser considerado como **polimórfico**

- Os **tipos diferentes** que podem ser referenciados por um elemento polimórfico são, exatamente, o **supertipo** e todos os seus **subtipos**

- **Esse é o tipo de poliformismo "tradicional"**

Herança e Polimorfismo



Polimorfismo e Python

- Outras linguagens, como o Java, são estáticas e fortemente tipadas, enquanto python é uma linguagem dinâmica
  - Isso obriga o interpretador manter o rastreios de todos os tipos das variáveis
- Sendo assim, quase tudo em Python é polimórfico, incluindo operadores, etc

```
1+2
'key'+'board'
[1,2,3] + [4,5,6]
(1, 2, 3) + (4, 5, 6)
{A: "a", B: "b"} + {C: "c", D: "d"}
```

Polimorfismo e Python

- De modo geral, o único jeito de "destruir" o poliformismo é forçar checagem de tipos com o "type", "isinstance" e "issubclass" com o intuito de realizar operações que fogem do controle do interpretador e seu design original

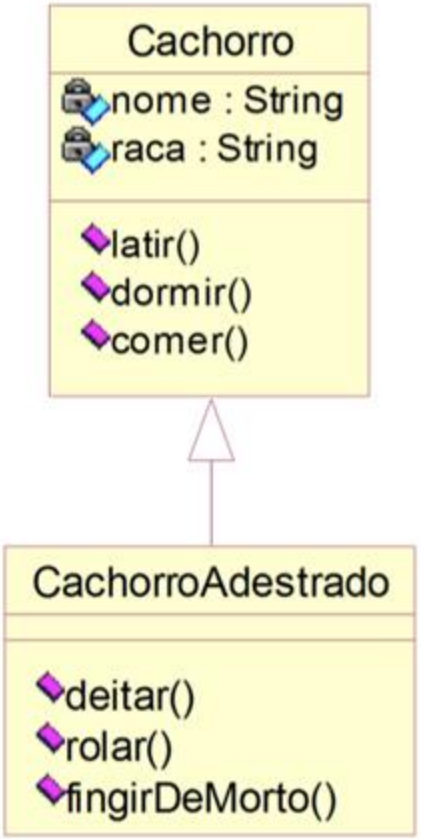
Exercício: Implemente as classes do modelo.

OBS: Usar encapsulamento

A. Atributos tipo String não podem receber valores None ou String vazia

B. Implementar Construtor (__init__) que recebe valores para todos os atributos da classe

C. Implementar um main que cria dois objetos: um do tipo Cachorro e outro do tipo CachorroAdestrado. Ao final, deve-se imprimir os dados dos objetos criados fazendo chamadas ao método **__str__**



Obrigado!

Alguma dúvida?

Prof. Henrique Mota

profhenriquemota@gmail.com