Algoritmo e Programação

Profª Danielle Couto

Aula 5

Agenda

- Orientação a objetos (OO)
  - Objeto
  - Classe
    - Herança
    - Encapsulamento
  - Vamos criar um game em Python?

Motivação

- O desenvolvimento de aplicações de software estão cada vez mais complexas;
- Cresceram as demandas por metodologias que pudessem abstrair e modularizar as estruturas básicas de programas; e
- A maioria das linguagens de programação suportam orientação a objetos: Python, Java, C++, PHP, Ruby, Pascal, entre outras.

Principais Vantagens

- Aumento de produtividade;
- Reúso de código;
- Redução das linhas de código programadas;
- Separação de responsabilidades;
- Componentização;
- Maior flexibilidade do sistema; e
- Facilidade na manutenção

Paradigma OO

- Um paradigma é uma **forma de abordar um problema**
- O paradigma da **orientação a objetos** surgiu no fim dos anos 60
  - Praticamente suplantou o anterior: **paradigma estruturado**
- Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada **analogia biológica**
  - "Como seria um sistema de **software** que funcionasse como um **ser vivo**?"

Objetos

- É a metáfora para se compreender a tecnologia orientada a objetos; Estamos rodeados por objetos: mesa, carro, livro, pessoa, etc;
- Os objetos do mundo real têm duas características em comum: Estado – representa as propriedades (nome, peso, altura, cor, etc.); e Comportamento – representa ações (andar, falar, calcular, etc.).




Orientação a Objeto

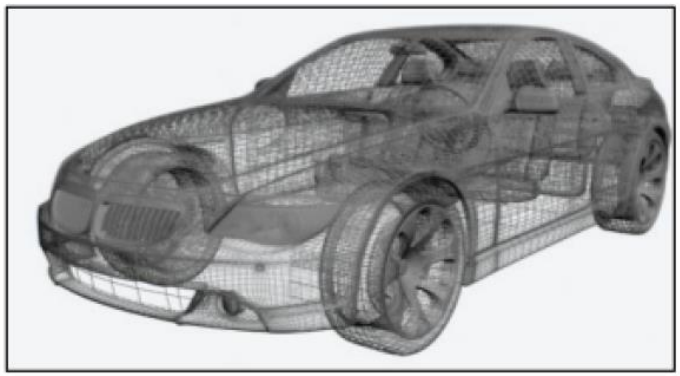
- É um paradigma para o desenvolvimento de software que baseia-se na utilização de componentes individuais (objetos) que colaboram para construir sistemas mais complexos.
  - A colaboração entre os objetos é feita através do envio de mensagens;
  - Descreve uma série de técnicas para estruturar soluções para problemas computacionais; e
  - É um paradigma de programação no qual um programa é estruturado em objetos.

Pilares OO

1) Abstração;
2) Encapsulamento;
3) Herança;
4) Polimorfismo.

Abstração - Classes

- A estrutura fundamental para definir novos objetos é a classe;
- Uma classe é definida em código-fonte.



Classe

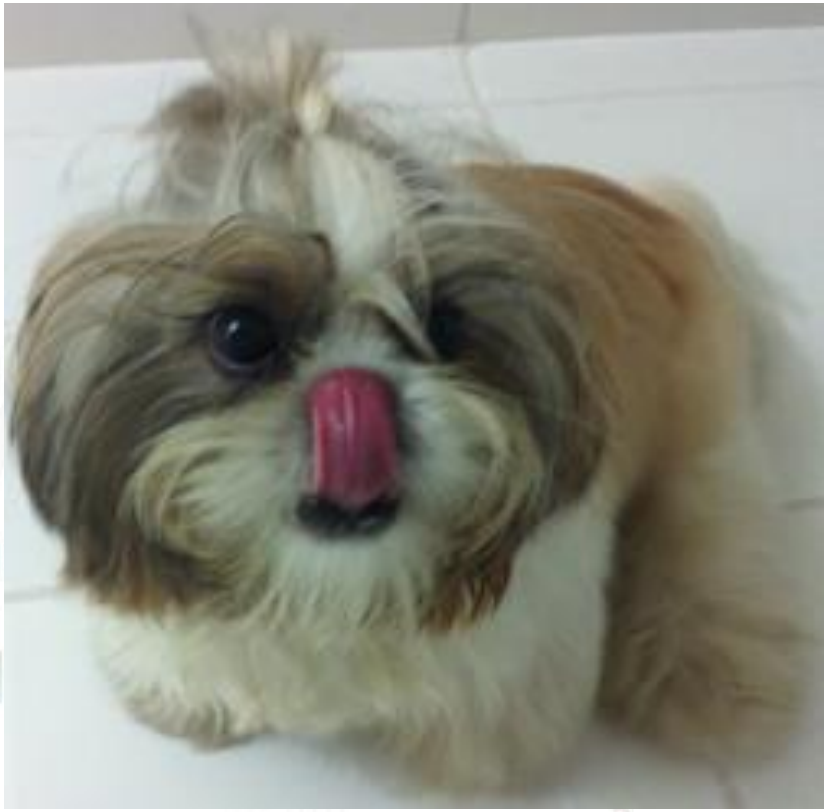
Objeto

Classes

- Podemos descrever o cachorro Garu em termos de seus **atributos** físicos:
  - é pequeno
  - sua cor principal é branco quase encardido
  - olhos pretos
  - orelhas pequenas e caídas,
  - rabo pequeno

Classes

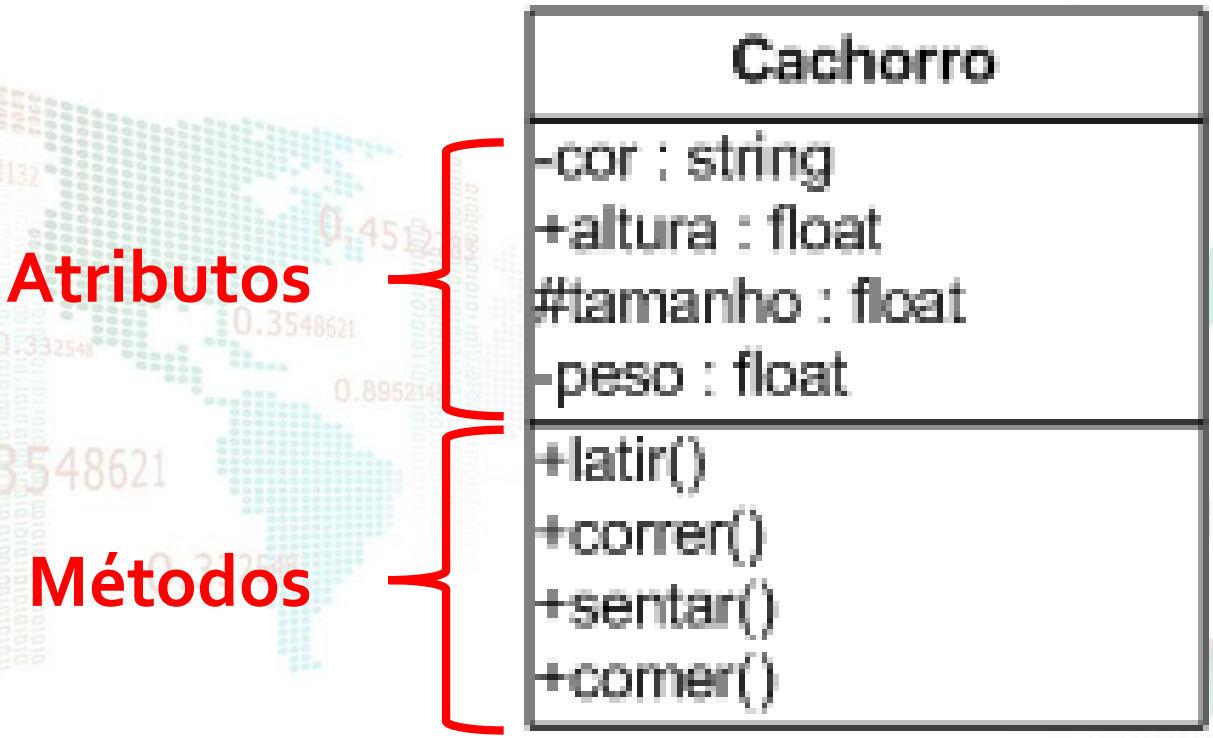
- Podemos também descrever algumas **ações** que ele faz (temos aqui os métodos):
- balança o rabo
- foge e se deita quando leva reclamação
- late quando ouve um barulho ou vê outro cão ou gato
- atende quando o chamamos pelo seu nome



Classes

- Representação do cachorro Garu:
  - **Propriedades** : [Cor do corpo: malhado; cor dos olhos: pretos; altura: 30 cm; comprimento: 40 cm largura : 24 cm]
  - **Métodos** : [balançar o rabo, latir, correr, deitar , sentar]

Representação de Classe



Classe em Python

- Estrutura

```
class nome_da_classe:
    atributos
    construtor
    métodos
```

- Exemplo

```
class Conta:
    numero = None
    saldo = None
```

Instância

- Uma instância é um objeto criado com base em uma classe definida;
- Classe é apenas uma estrutura, que especifica objetos, mas que não
- pode ser utilizada diretamente;
- Instância representa o objeto concretizado a partir de uma classe;
  - Uma instância possui um ciclo de vida:
    - Criada;
    - Manipulada; e
    - Destruída.

Estrutura
variável = Classe()

Exemplo

- Telefone

Características:

cor:azul

discagem: pulso

Comportamento:

tocar()

discar()

- Em Python

```
conta = Conta()
conta.numero = 1
conta.saldo = 10
print(conta.numero)
print(conta.saldo)
```

Orientação a Objetos

- Em resumo, a expressão **orientada a objetos** significa que
  - o aplicativo é organizado como uma coleção de objetos que incorporam tanto a estrutura como o comportamento dos dados
- Objetos pertencem à **classes**
- Abstrações utilizadas para representar um conjunto de objetos com ***características*** *e* ***comportamento idênticos***
- Uma classe pode ser vista como uma **"fábrica de objetos"**

Classe

- Objetos são “**instâncias**” de uma classe
  - Todos os objetos são instâncias de alguma classe
- Todos os objetos de uma classe são **idênticos** no que diz respeito a sua interface e implementação
  - o que difere um objeto de outro é seu **estado** e sua **identidade**

Classe - Exemplo

Instância da classe (objeto)

Características:

cor:azul

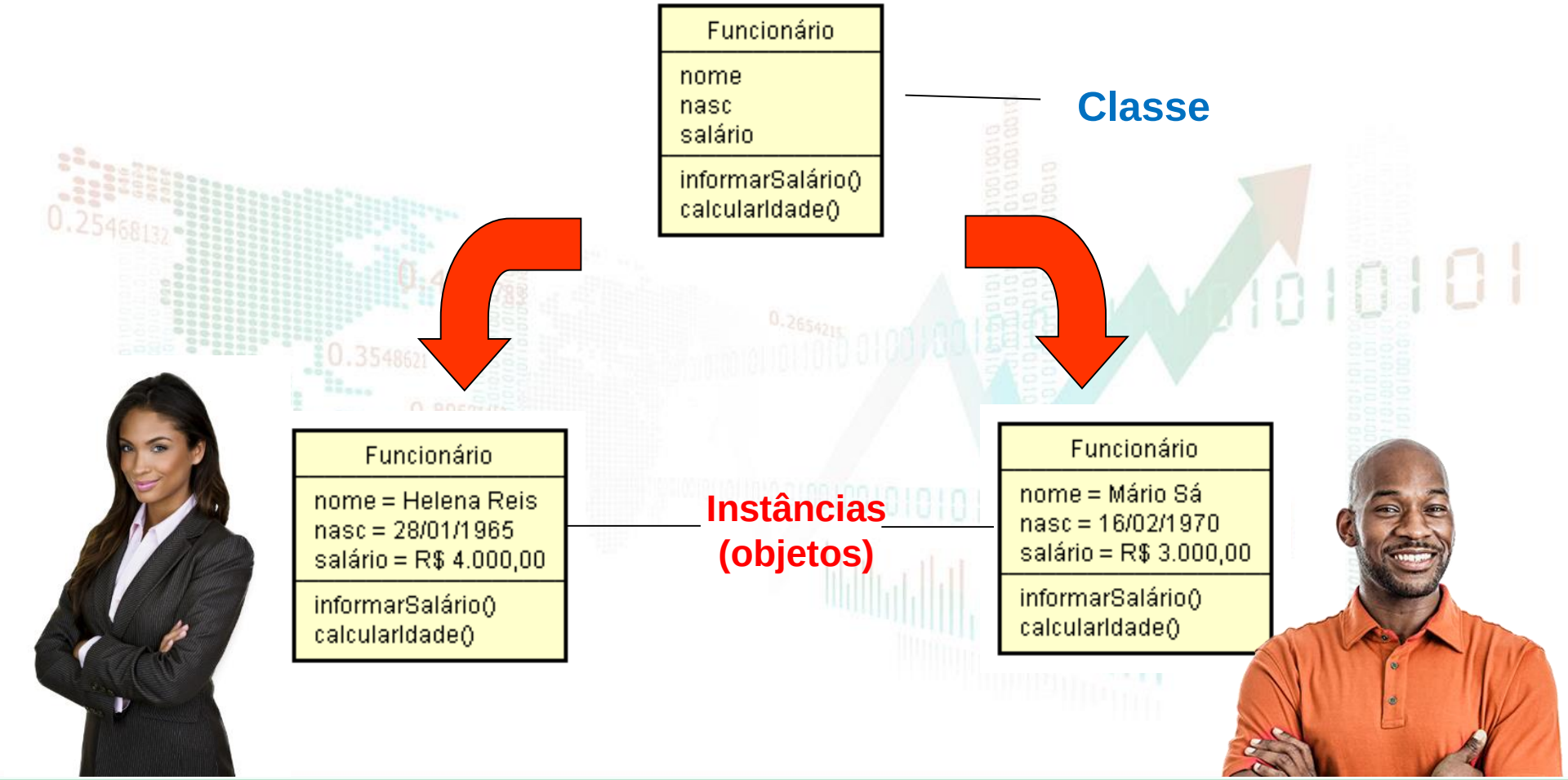
discagem: pulso

marca: siemens

Classe

Telefone
marca numero discagem
tocar() discar()

Classes



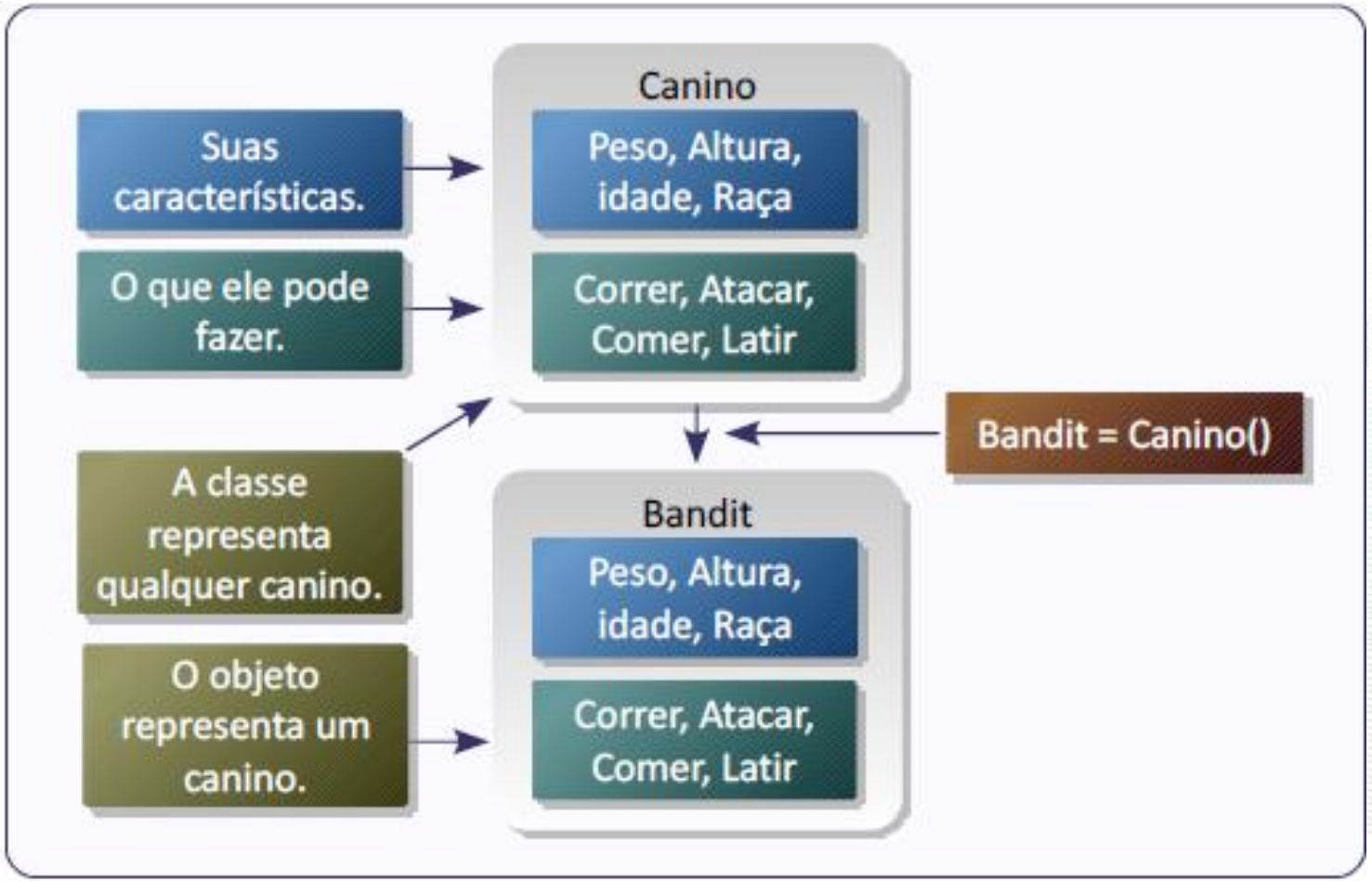
Classe - Atributos

- Descrevem as **características** das instâncias de uma classe
- Seus valores definem o **estado** do objeto
- O estado de um objeto pode mudar ao longo de sua existência
- A identidade de um objeto, contudo, nunca muda

Classe - Operações

- Representam o comportamento das instâncias de uma classe
- Correspondem às **ações** das instâncias de uma classe

Classe - Cachorro



Encapsulamento

- Na terminologia da orientação a objetos, diz-se que um objeto possui uma **interface**.
- A interface de um objeto é como ele **aparece** para os demais objetos:
  - Suas características, sem **detalhes internos**
- A interface de um objeto define os **serviços** que ele pode realizar e conseqüentemente as **mensagens que ele recebe**
  - **Um objeto é "visto" através de seus métodos**

Encapsulamento

- Um objeto é utilizado através de sua **interface**
  - Não precisamos conhecer a implementação dos seus ***métodos***
- Um objeto encapsula tanto dados como algoritmos (**seus métodos**)

```
>>> s = "quem parte e reparte, fica com a maior parte"
>>> s.find("parte")
5
```

Encapsulamento

- Encapsulamento é a **proteção** dos atributos ou métodos de uma classe.
- Em Python existem somente o **public** e o **private** e eles são definidos no próprio nome do atributo ou método.
- Atributos ou métodos iniciados por no máximo **dois sublinhados** (underline) são **privados** e todas as outras formas são **públicas**

Exemplo

```
class Poupanca(ContaCorrente):
    # A classe Poupança tem um atributo
    # taxaJuros que é específico
    def __init__(self, numero, taxa):
        ContaCorrente.__init__(self, numero)
        self.taxaJuros = taxa
    # E tem também um método para
    # render os juros
    def renderJuros(self):
        self.saldo = self.saldo +
        self.taxaJuros*self.saldo/100
```

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

Herança

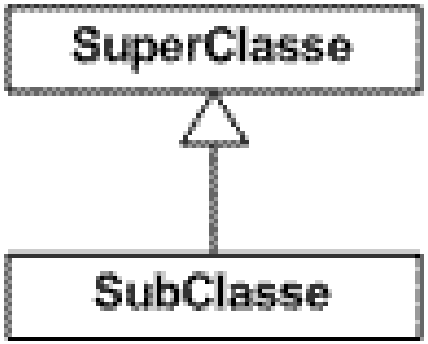
- É o que permite construir objetos que são **especializações** de outro objeto
  - Objetos especializados ***herdam*** dos objetos genéricos
- Por exemplo: um objeto que representa uma forma geométrica. Ele tem **características** (área, perímetro, etc.)
  - Um **polígono é uma forma geométrica**,
    - Portanto, herda todas as características de formas geométricas
    - Deve suportar também **características específicas** como número de lados e comprimento de arestas

Generalização/Especialização

- **Generalização** é um processo que ajuda a identificar as classes principais do sistema
- Ao identificar as partes comuns dos objetos, a generalização ajuda a reduzir as redundâncias, e promove a reutilização.
  - Criar **classes genéricas**
- O processo inverso a generalização é a **especialização**.
- A especialização foca na criação de classes mais individuais
  - Criar **classes especializadas**

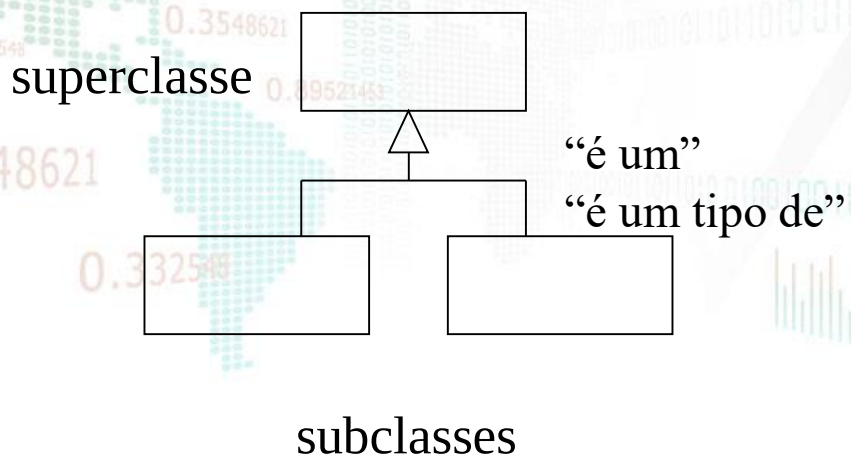
Herança

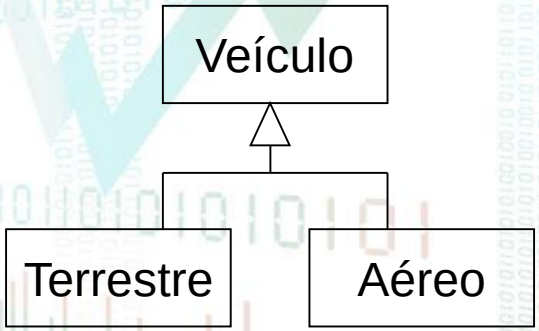
- Uma classe pode ser **definida** a partir de outra já **existente**
- Abstrai classes genéricas (**superclasse**), a partir de classes com propriedades (atributos e operações) **semelhantes**
  - Modelar similaridades entre classes, preservando diferenças
- As **subclasses herdam** todas as **propriedades**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



Herança

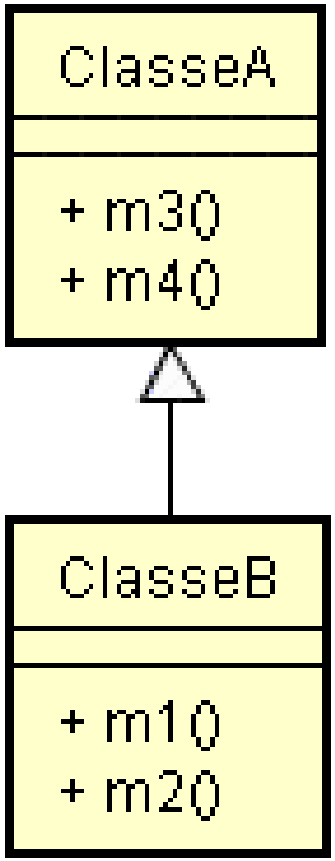
- Relacionamento entre itens gerais (**superclasses**) e itens mais específicos (**subclasses**)





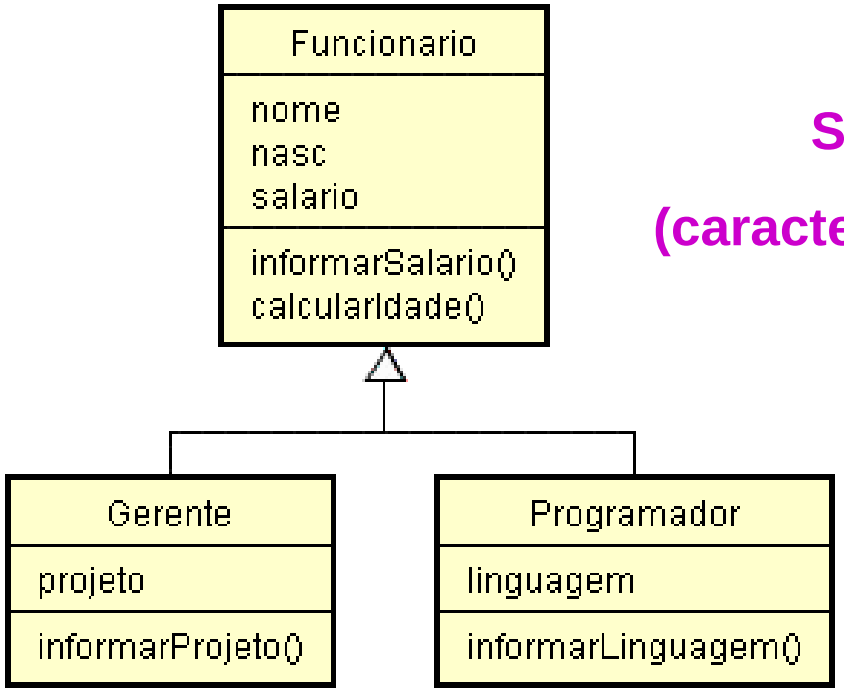
Herança

- 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

- Para fazer uma classe C herdar de outra B, basta declarar **C** como:

  class C(B):

  . . .

- **C** herda todos os atributos de B
- A especialização de C se dá acrescentando-se novos atributos e métodos ou alterando-se seus métodos
- Se na classe C, for necessário invocar um método m() de B :
  - pode-se utilizar a notação B.m() para diferenciar do m *de* **C**, referido como C.m()

Polimorfismo

- É o que permite que dois **objetos diferentes** possam ser usados de forma semelhante
  - Por exemplo, tanto **listas** quanto **strings** podem ser indexadas por um número e usam o **len()**
  - Se escrevemos ...

```
for i in range(len(X)):
    print i, X[i]
```

  - ...não é possível saber de antemão **se X é uma lista ou uma string**
- Um algoritmo para ser aplicado a um objeto X, então também pode ser aplicado a um objeto Y

Polimorfismo

- É originário do grego e significa “muitas formas” (poli = muitas, morphos = formas);
- Indica a capacidade de abstrair **várias implementações** diferentes em uma única interface;
- É o princípio pelo qual duas ou mais classes derivadas de uma mesma **superclasse** podem invocar métodos que têm a mesma identificação (assinatura) mas comportamentos distintos;
- Quando polimorfismo está sendo utilizado, o comportamento que será adotado por um método só será definido durante a execução.

Classe em Python

- A maneira mais simples é:

```
class nomeClasse:
    var = valor
    ...
    var = valor
    def metodo (self, ... arg):
        ...
    def metodo (self, ... arg):
        ...
```

Classe em Python

- As **variáveis** e os **métodos** são escritos precedidos pelo **nome da classe** e por um **ponto** (.)
  - A variavel ***v*** definida numa classe ***C*** é escrita ***C.v***
- Os métodos sempre têm **self** como primeiro argumento
  - **self** se refere a uma instância da classe
- Uma nova instância da classe é criada usando **nomeClasse()**

Construtores

- O método **inicia** foi usado para inicializar atributos e é conhecido como ***construtor*** da classe
- Python suporta construtores que podem ser chamados **automaticamente** na criação de instâncias
  - Basta definir na classe um método chamado **__init__**
  - Este método é chamado **automaticamente** durante a criação de um nova instância da classe, sendo que os **argumentos** são passados **entre parênteses** após o nome da classe

Construtores em Python

- Determina que ações devem ser executadas quando da criação de um objeto;
- Pode possuir ou não parâmetros.

Estrutura

```
def __init__(self,parâmetros):
```

Exemplo

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

conta = Conta(1)
print(conta.numero)
print(conta.saldo)
```

Métodos em Python

- Representam os comportamentos de uma classe;
- Permitem que acessemos os atributos, tanto para recuperar os valores, como para alterá-los caso necessário;
- Podem retornam ou não algum valor;
- Podem possuir ou não parâmetros.

Estrutura

```
def nome_do_método(self,parâmetros):
```

Importante

O parâmetro **self** é obrigatório.

Métodos em Python

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

    def consultar_saldo(self):
        return self.saldo

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

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

    def transferir(self,conta,valor):
        self.saldo -= valor
        conta.saldo += valor

conta1 = Conta(1)
conta1.creditar(10)
conta2 = Conta(2)
conta2.creditar(5)
print(conta1.consultar_saldo())
print(conta2.consultar_saldo())
conta1.transferir(conta2,5)
print(conta1.consultar_saldo())
print(conta2.consultar_saldo())
```

Desafios

1. Classe Retangulo: Crie uma classe que modele um retângulo:
   - Atributos: LadoA, LadoB (ou Comprimento e Largura, ou Base e Altura, a escolher)
   - Métodos: Mudar valor dos lados, Retornar valor dos lados, calcular Área e calcular Perímetro;
- Crie um programa que utilize esta classe. Ele deve pedir ao usuário que informe as medidas de um local. Depois, deve criar um objeto com as medidas e imprimir suas área e perímetro

Desafios

2. Considere as classes ContaCorrente e Poupança apresentadas nesta 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).

Bibliografia

- Livro “Como pensar como um Cientista de Computação usando Python” – Capítulo 12 http://pensarpython.incubadora.fapesp.br/portal
- Python Tutorial http://www.python.org/doc/current/tut/tut.html
- Dive into Python http://www.diveintopython.org/
- Curso em Vídeo https://www.youtube.com/channel/UCrWvhVmt0Qac3HgsjQK62FQ
- Python Brasil http://www.pythonbrasil.com.br/moin.cgi/DocumentacaoPython#head5a7ba2746c5191e7703830e02d0f5328346bcaac

Próxima Aula..

- Prática Python com OO
- Ferramentas Python