Programação Orientada a Objetos

Conceitos de Orientação a Objetos

Profa. Andriele Busatto do Carmo

acarmo@unisinos.br

UNISINOS

Conceitos de Orientação a Objetos

Construtores e destrutores

- Método construtor:
  - Método especial usado para inicializar objetos quando estes são criados.
    - Aloca recursos do objeto
    - Define o estado inicial dos objetos
    - Garante a existência dos atributos
  - No Python:

```
def __init__(self, nome_atributo):
    self._atributo = nome_atributo
```

- Método destrutor:
  - Método especial chamado quando a instância do objeto é destruída.
    - Libera recursos do objeto
    - Permite executar alguma tarefa antes que o objeto seja eliminado
  - No Python:

```
def __del__(self):
    #código a ser executado
```

→ *Pouco usado, pois Python implementa garbage collector.*

Encapsulamento

- Conceito implementado pelas linguagens orientadas a objetos:
  - para "proteger" os dados e permitir que sejam acessados somente através de métodos do próprio objeto.
- O programa executa trocando mensagens entre objetos.
- Para acessá-los, primeiro identifica-se o objeto e depois o dado que se deseja acessar.
- O mesmo ocorre com os métodos (funções).

Restrições de visibilidade

- Modificadores de acesso:
  - público: permite atributos e/ou métodos sejam acessados a partir de qualquer classe que pertença ao mesmo programa.
  - protegido: permite atributos e/ou métodos sejam acessados a partir de qualquer classe que pertença ao mesmo pacote/módulo ou a mesma descendência de classes.
  - privado: permite que atributos e/ou métodos de uma classe sejam acessados somente dentro da classe onde foram definidos.

Restrições de visibilidade no Python

- Para indicar que um atributo ou método é privado:
  - Usa-se dois *underscores* "__" no início do nome do atributo ou método (função).
  - Exemplo:
    - Atributo: nome → público

      __nome → privado

- Para acessar o atributo, deve ser utilizado um método da classe, que usualmente é público.

Métodos de acesso – *getters*

- Métodos de acesso retornam o valor de um atributo.

- Exemplo:

```
def get_nome(self):
    return self.__nome
```

Métodos modificadores – *setters*

- Trocam o valor atual de um atributo por um novo valor, passado por parâmetro.

- Exemplo:

```
def set_nome(self, nome):
    self.__nome = nome
```

Métodos *get* e *set* no Python

- Atributos escritos com dois *underscores* não são verdadeiramente privados no Python.

- Então, por convenção, programadores Python usam apenas um *underscore* para “proteger” um atributo.
  - Esse único *underscore* não tem significado para o interpretador, mas indica ao programador que esse é um atributo que deve ser tratado como um atributo privado e só deve ser acessado através dos métodos de acesso e modificadores.

Métodos *get* e *set* no Python

- Exemplo:

```
class ContaBancaria:
    def __init__(self, saldo =0 ):
        self._saldo = saldo

    def get_saldo(self):
        return self._saldo

    def set_saldo(self, saldo):
        self._saldo = saldo
```

```
cb1 = ContaBancaria(100)
cb2 = ContaBancaria(2000)
cb3 = ContaBancaria()
cb3.set_saldo(cb1.get_saldo() + cb2.get_saldo())
```

Legibilidade ruim

Métodos *get* e *set* no Python

- Solução para legibilidade:
  - Uso de *decorators*. O Python utiliza um decorador chamado **property**.

- Logo, métodos *get* e *set* ficam da seguinte forma:

```
@property
def nome_atributo(self):
    return self._nome_atributo
```

```
@nome_atributo.setter
def nome_atributo(self, nome_atributo):
    self._nome_atributo = nome_atributo
```

Não é obrigatório que tenha o mesmo nome do atributo.

Métodos *get* e *set* no Python

- No nosso exemplo, então:

```
class ContaBancaria:
    def __init__(self, saldo =0 ):
        self._saldo = saldo

    @property
    def saldo(self):
        return self._saldo

    @saldo.setter
    def saldo(self, saldo):
        self._saldo = saldo
```

```
cb1 = ContaBancaria(100)
cb2 = ContaBancaria(2000)
cb3 = ContaBancaria()
cb3.saldo = cb1.saldo + cb2.saldo
```

UML

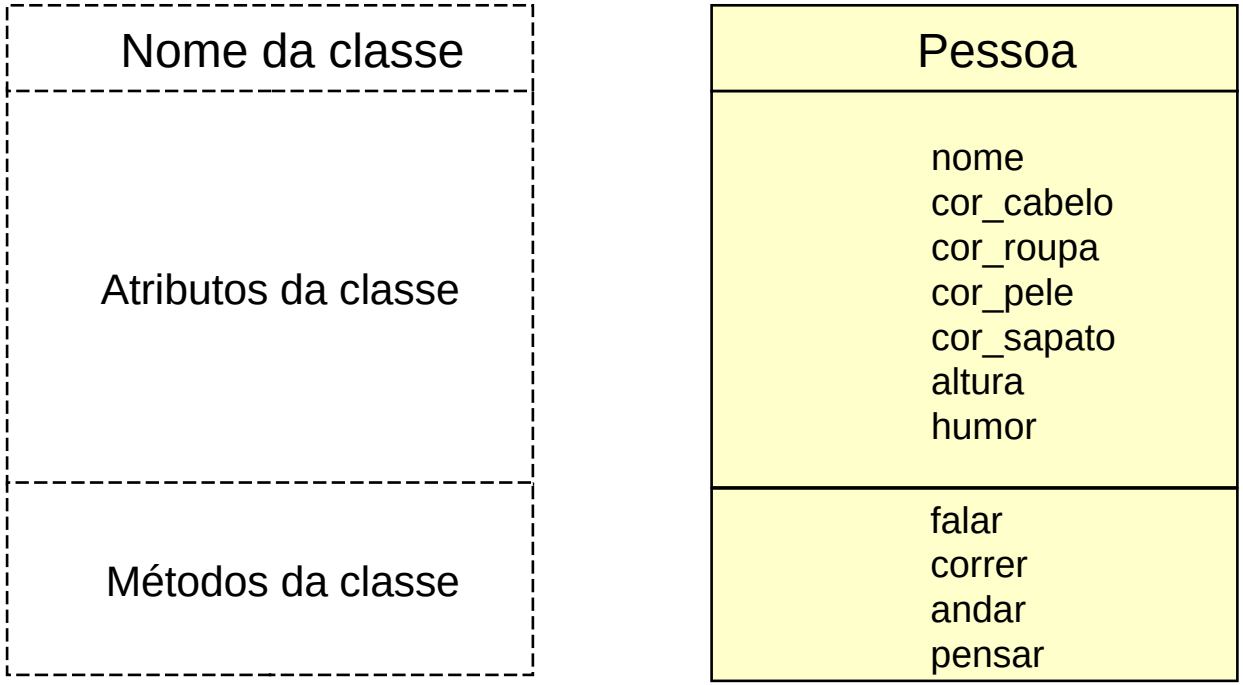
UML

- *Unified Modeling Language* ou, Linguagem de Modelagem Unificada.

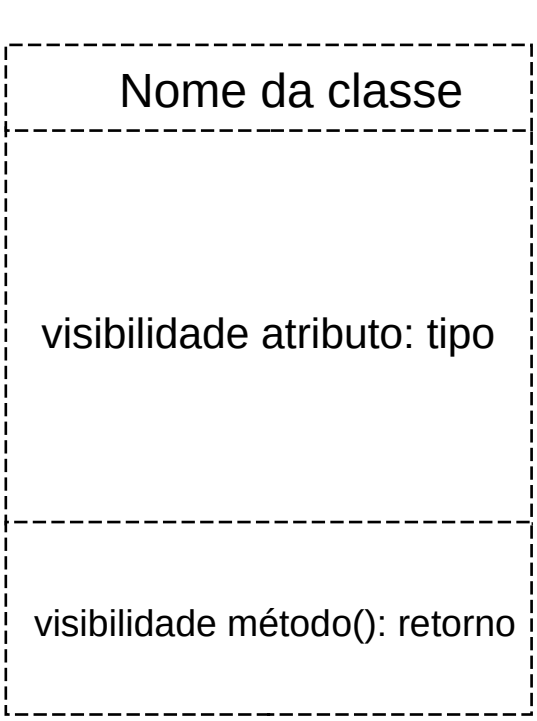
- Permite visualizar, especificar e documentar em projetos de desenvolvimento de software.

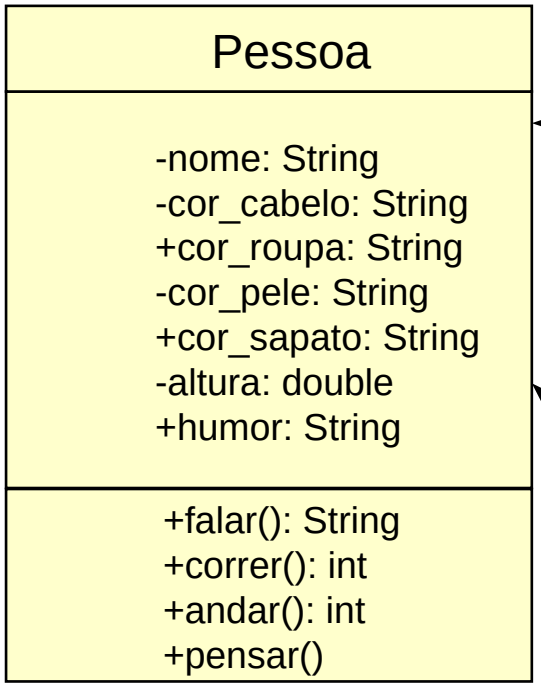
- Apresenta diferentes diagramas:
  - de casos de uso
  - **de classes**
  - de sequência
  - ...

UML

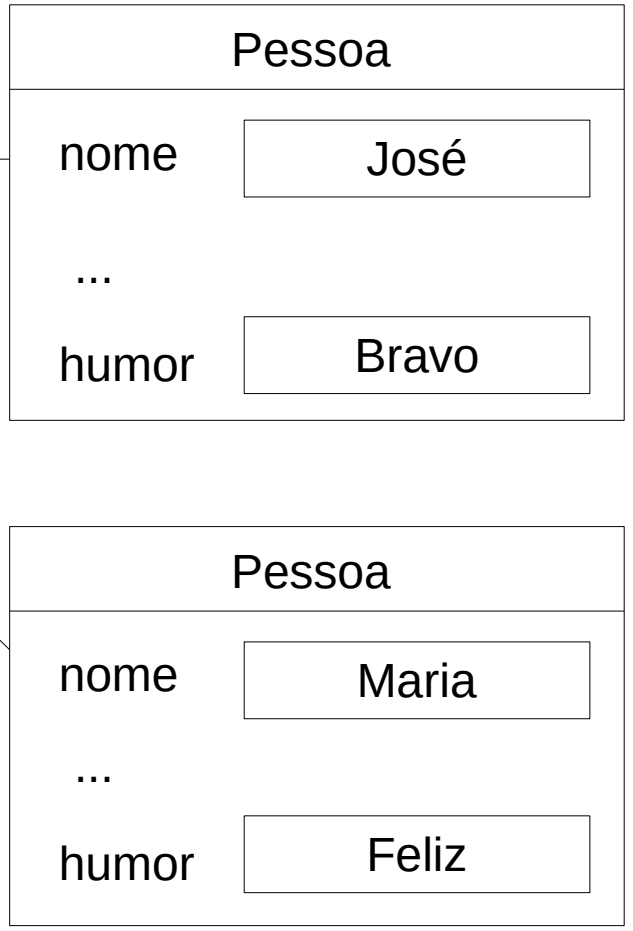

Nome da classe
Atributos da classe
Métodos da classe
Pessoa
nome
cor_cabelo
cor_roupa
cor_pele
cor_sapato
altura
humor
falar
correr
andar
pensar

UML





INSTÂNCIAS



Modificadores de acesso:

+: *público*
-: *privado*
#: *protegido*

UML

- Relacionamento:
  - Uma classe pode se relacionar com outras de diversas maneiras. Na UML, a associação entre classes é representada por linhas e setas específicas.
  - Por questão de simplicidade, utilizaremos apenas uma linha contínua para ligar as classes que possuem algum relacionamento.