

CC8210 – NCA210

Programação Avançada I

Prof. Reinaldo A. C. Bianchi
Prof. Isaac Jesus da Silva
Prof. Danilo H. Perico

Introdução à Programação Orientada a Objetos

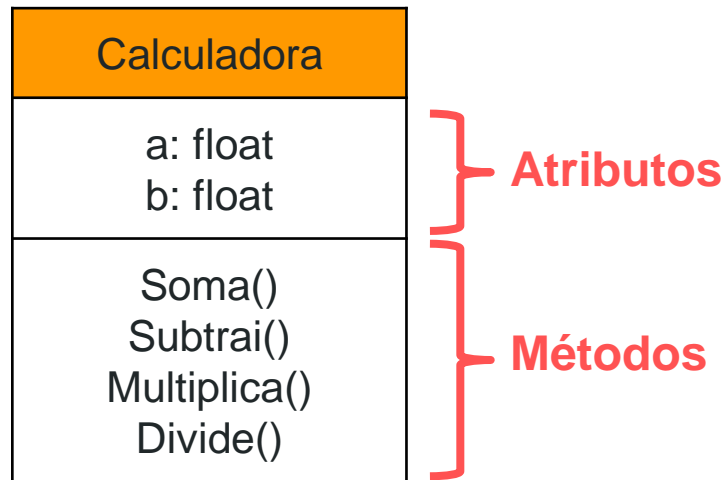
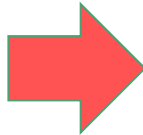
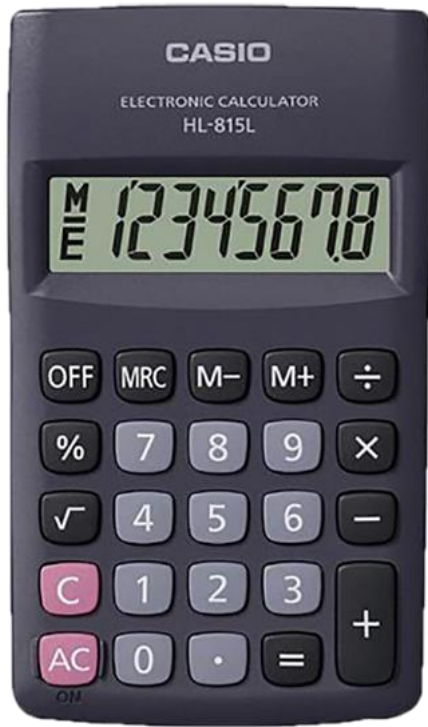
Revisão de POO

Estrutura de uma Classe

Nome da Classe
- Atributos
- Métodos

- **Atributos** são **variáveis** que armazenam informações do objeto.
- **Métodos** são as operações (**funções**) que o objeto pode realizar.

Exemplo de Classe



Definindo uma Classe em Python

Em Python uma classe é declarada da seguinte forma:

```
1  class NomeClasse:  
2      # atributos  
3  
4      # métodos
```

Construtores e Destrutores em Python

```
class Robot:
```

```
    name = ""
```

```
    positionX = 0.0
```

```
    positionY = 0.0
```

```
    direction = 0.0
```



Atributos

```
    def __init__(self, nome = "Tiago++"):
```

```
        self.name = nome
```

```
        print("Construindo o %s :-)" % self.name)
```

```
    def __del__(self):
```

```
        print ("Bye bye!!")
```

Exemplo

```
1 #coding: utf-8
2 class Aluno:
3     nome = ""
4     ra = 0
5
6     def __init__(self, nome="", ra=0):
7         self.nome = nome
8         self.ra = ra
9
10    def mostraAluno(self):
11        print("Nome: %s" % self.nome)
12        print("R.A.: %d" % self.ra)
13 #####
14 aluno = Aluno("Danilo", 1234567890)
15 aluno.mostraAluno()
```

```
fei CursoFerias2016 $ python exemplo.py
Nome: Danilo
R.A.: 1234567890
fei CursoFerias2016 $
```



Encapsulamento

Encapsulamento

- Encapsulamento é um dos 3 pilares de Programação Orientada a Objetos, juntamente com Herança e Polimorfismo.
- Encapsulamento vem de encapsular, que em programação orientada a objetos significa separar o programa em partes, o mais isolado possível.
- A ideia é tornar o software mais flexível, fácil de modificar e de criar novas implementações.

Encapsulamento

- O Encapsulamento serve para controlar o acesso aos atributos e métodos de uma classe.
- É uma forma eficiente de proteger os dados manipulados dentro da classe, além de determinar onde esta classe poderá ser manipulada.
- O encapsulamento que é dividido em dois níveis:
 - Nível de classe: quando determinamos o acesso de uma classe inteira.
 - Nível de membro: Quando determinamos o acesso aos atributos ou métodos de uma classe.

Encapsulamento em Python

- Python não possui mecanismos de encapsulamento formais!!!!
- No entanto, existe uma convenção que é seguida pela maioria dos programas em Python:
 - Um nome prefixado com **um** sublinhado deve ser tratado como uma parte não-pública da API (seja uma função, um método ou um atributo de dados).
 - Tais nomes devem ser considerados um detalhe de implementação e sujeito a alteração sem aviso prévio.

Encapsulamento em Python

- Python não possui mecanismos de encapsulamento formais!!!!
- No entanto, existe uma convenção que é seguida pela maioria dos programas em Python:
 - Um nome prefixado com **dois** sublinhados invocará as regras desfiguração (mangling) de nomes do Python.
 - Python desfigura esses nomes com o nome da classe: se a classe Foo tem um atributo chamado `__a`, ele não pode ser acessado por `Foo.__a`.

Exemplo 1: "_"

```
class Robot:  
    _name = "Tiago++"  
    _positionX = 0.0  
    _positionY = 0.0  
    _direction = 0.0
```



Atributos privados

Exemplo

```
C3_PELE    = Robot ()  
R2D_DUNGA = Robot ()  
ROBOMARIO = Robot ()  
  
print (C3_PELE._positionX)  
print (C3_PELE._positionY)  
pass
```

Exemplo de Saída:

```
Construindo o Robo. Aguarde :-)  
Construindo o Robo. Aguarde :-)  
Construindo o Robo. Aguarde :-)  
0.0  
0.0  
Bye bye!!  
Bye bye!!  
Bye bye!!
```

Exemplo 2: "__"

```
class Robot:  
    __name = "Tiago++"  
    __positionX = 0.0  
    __positionY = 0.0  
    __direction = 0.0
```



**Atributos privados
e mengled**

Exemplo

```
C3_PELE    = Robot()
R2D_DUNGA  = Robot()
ROBOMARIO  = Robot()

print (C3_PELE.__positionX)
print (C3_PELE.__positionY)
pass
```

Exemplo de Saída:

Traceback (most recent call last):

File

"/Users/rbianchi/Documents/AULAS/CC8210 - PROGRAMAÇÃO AVANÇADA I/Semana 09/robots_encapsulados.py", line 37, in <module>

print (C3_PELE.__positionX)

AttributeError: 'Robot' object has no attribute '__positionX'

Exemplo 3: "__" com função de acesso

```
class Robot:
    __name = "Tiago++"
    __positionX = 0.0
    __positionY = 0.0
    __direction = 0.0
```

**Atributos privados
e mangled**

```
def printPosition(self):
    print ("X = %f" % self.__positionX)
    print ("Y = %f" % self.__positionY)
```

**Método
de
Acesso**

Exemplo

```
C3_PELE    = Robot ()  
R2D_DUNGA = Robot ()  
ROBOMARIO = Robot ()  
  
C3_PELE.printPosition()  
  
pass
```

Exemplo de Saída:

```
Construindo o Tiago++ :-)  
Construindo o Tiago++ :-)  
Construindo o Tiago++ :-)  
X = 0.000000  
Y = 0.000000  
Bye bye!!  
Bye bye!!  
Bye bye!!
```

Herança em POO

Reutilização (ou Reuso)

- Reutilização de código é o “Santo Graal” da programação.
- Diversas maneiras:
 - Cópia direta do código (método porco).
 - Composição (bibliotecas de funções).
 - Herança (linguagens Orientadas a Objetos).

Reutilização via Composição

- Permite a reutilização de programas, criando objetos dentro de novas classes.
- Maneira mais simples de reutilização de programas:
 - Reutiliza o código.
- É a maneira pela qual já utilizamos os objetos de dados (arrays, ...) definidos na biblioteca NumPy.

Composição = “Tem um?”

- Quando utilizar a composição?
- Regra: realizar a pergunta “Tem um?”
- Exemplos:
 - Gorgonzola **tem** Mofo?
 - Carro **tem** Volante?
- Para a composição criamos uma instância de uma classe, dentro da nova classe:
 - A nova classe terá um objeto.

Exemplo de Composição

```
class RoboticTeam:
```

```
    C3_PELE      = Robot("C3 Pelé")
```

```
    R2D_DUNGA   = Robot("R2D Dunga")
```

```
    ROBOMARIO   = Robot("Robomário")
```

```
    def ataque(self):
```

```
        print ("Chuta para frente")
```

```
    def defesa(self):
```

```
        print ("Realiza uma defesa")
```

} Objetos



Exemplo RoboticTeam

```
ROBOFEI = RoboticTeam()  
ROBOFEI.ataque()  
pass
```

Exemplo de Saída:

```
Construindo o C3 Pelé :-)  
Construindo o R2D Dunga :-)  
Construindo o Robomário :-)  
Chuta para frente  
Bye bye!!  
Bye bye!!  
Bye bye!!
```

Exemplo de Composição

```
class RoboticTeam:
```

```
    C3_PELE      = Robot("C3 Pelé")
```

```
    R2D_DUNGA   = Robot("R2D Dunga")
```

```
    ROBOMARIO   = Robot("Robomário")
```

} Objetos

```
def ataque(self):
```

```
    print ("Chuta para frente")
```

```
    self.C3_PELE.moveForward()
```

```
    self.R2D_DUNGA.moveBackward()
```

```
    self.ROBOMARIO.turnLeft()
```

```
def defesa(self):
```

```
    print ("Realiza uma defesa")
```



Exemplo RoboticTeam

```
ROBOFEI = RoboticTeam()  
ROBOFEI.ataque()  
pass
```

Exemplo de Saída:

```
Construindo o C3 Pelé :-)  
Construindo o R2D Dunga :-)  
Construindo o Robomário :-)  
Chuta para frente  
Anda para frente  
Anda para tras  
Girando para a esquerda  
Bye bye!!  
Bye bye!!  
Bye bye!!
```

Conceitos de Herança

- É uma forma de **reutilização** de código!
- O conceito de herança se baseia no princípio de que toda codificação mais genérica pode ser transmitida para classes mais específicas.
- Cria uma **nova classe** a partir de uma classe existente.
- Relação **é um**


Herança

- Cria uma nova classe como uma **extensão** de uma classe já existente.
- A nova classe é um tipo da classe já existente.
- Permite utilizar a forma da classe existente, adicionando código, sem destruir a classe existente.
- A nova classe herda os atributos e os métodos da classe existente.

Exemplo - Carros

- Uma Ferrari é um **Carro**
- Uma BMW é um **Carro**
- Um Fusca é um **Carro**

Podemos ter uma classe **Carro** que tem características comuns a todos esses veículos!



A classe carro pode conter os seguintes atributos:

- **Rodas**
- **Cor**
- **ano de fabricação**

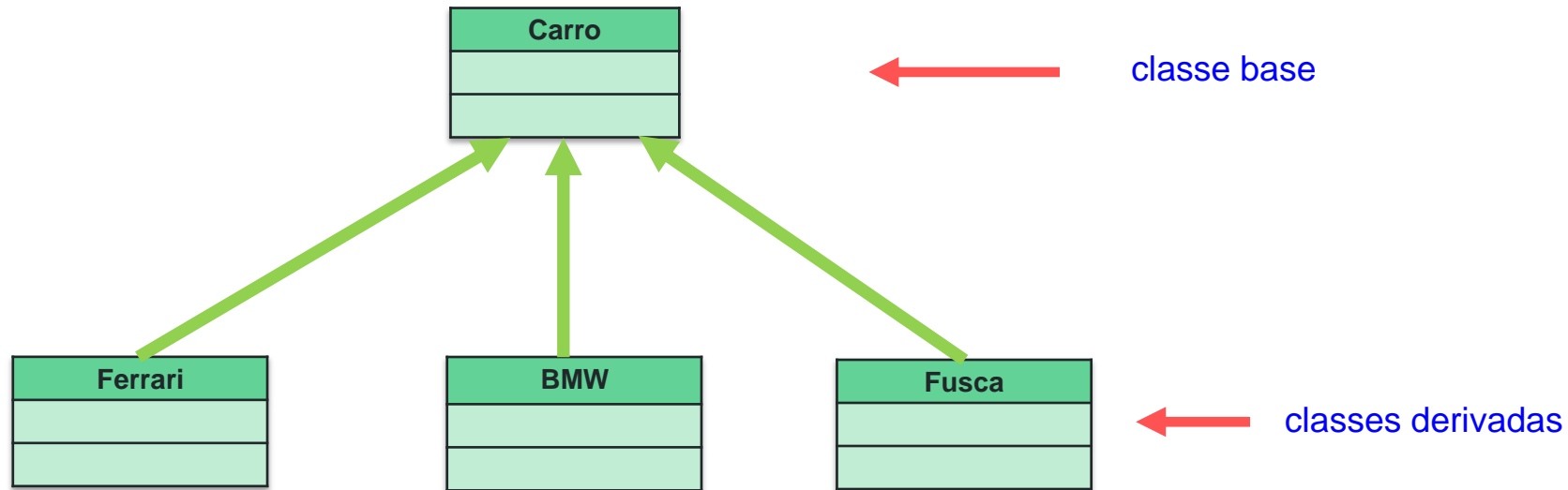
Superclasse e Subclasse

- ❑ Considerando os exemplos dados:
 - As classes Mamífero e Carro são **superclasses** / **classes bases**
 - As classes Ferrari, BMW, Fusca são **subclasses** / **classes derivadas**
- ❑ **Superclasses** tendem a ser **mais genéricas**
- ❑ **Subclasses** tendem a ser **mais específicas**

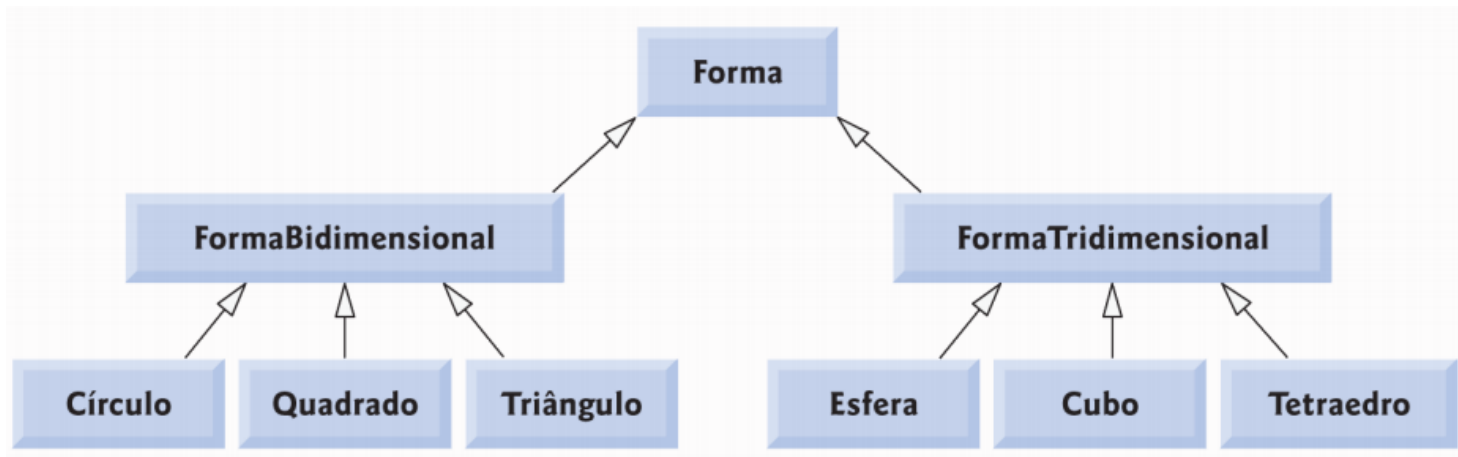
A Herança Permite que as Subclasses:

- ❑ Herdem os **atributos** e os **métodos** da superclasse:
 - **atributos** e **métodos** herdados podem ser diretamente utilizados - não é preciso escrevê-los novamente.
- ❑ Definam novos **atributos** e **métodos**.
- ❑ Modifiquem um **método** definido na superclasse (sobrescrita - override)

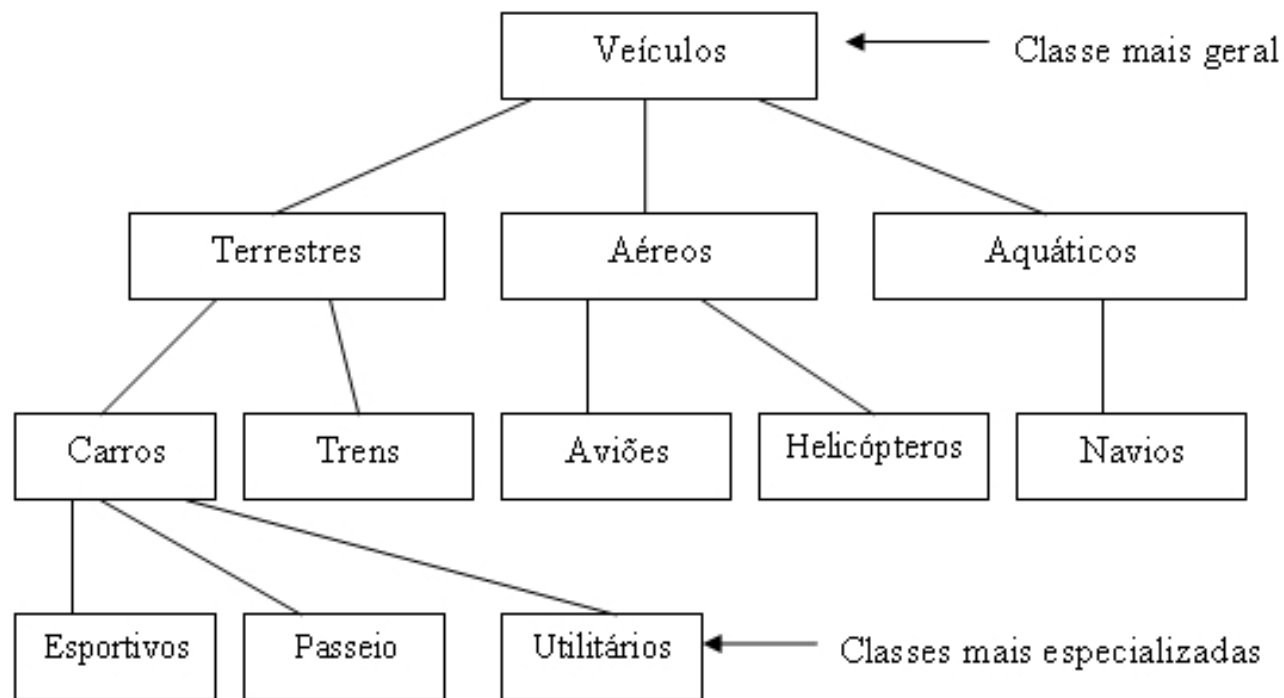
Herança - UML



Exemplo de Herança



Exemplo de Herança



Herança

- ❑ Quando utilizar a herança?
- ❑ Realizar a pergunta “É um/uma?”
- ❑ Exemplos:
 - Funcionario é uma Pessoa?
 - Carro é um Veículo?
 - Aluno é uma Pessoa?
 - Gerente é um Empregado?

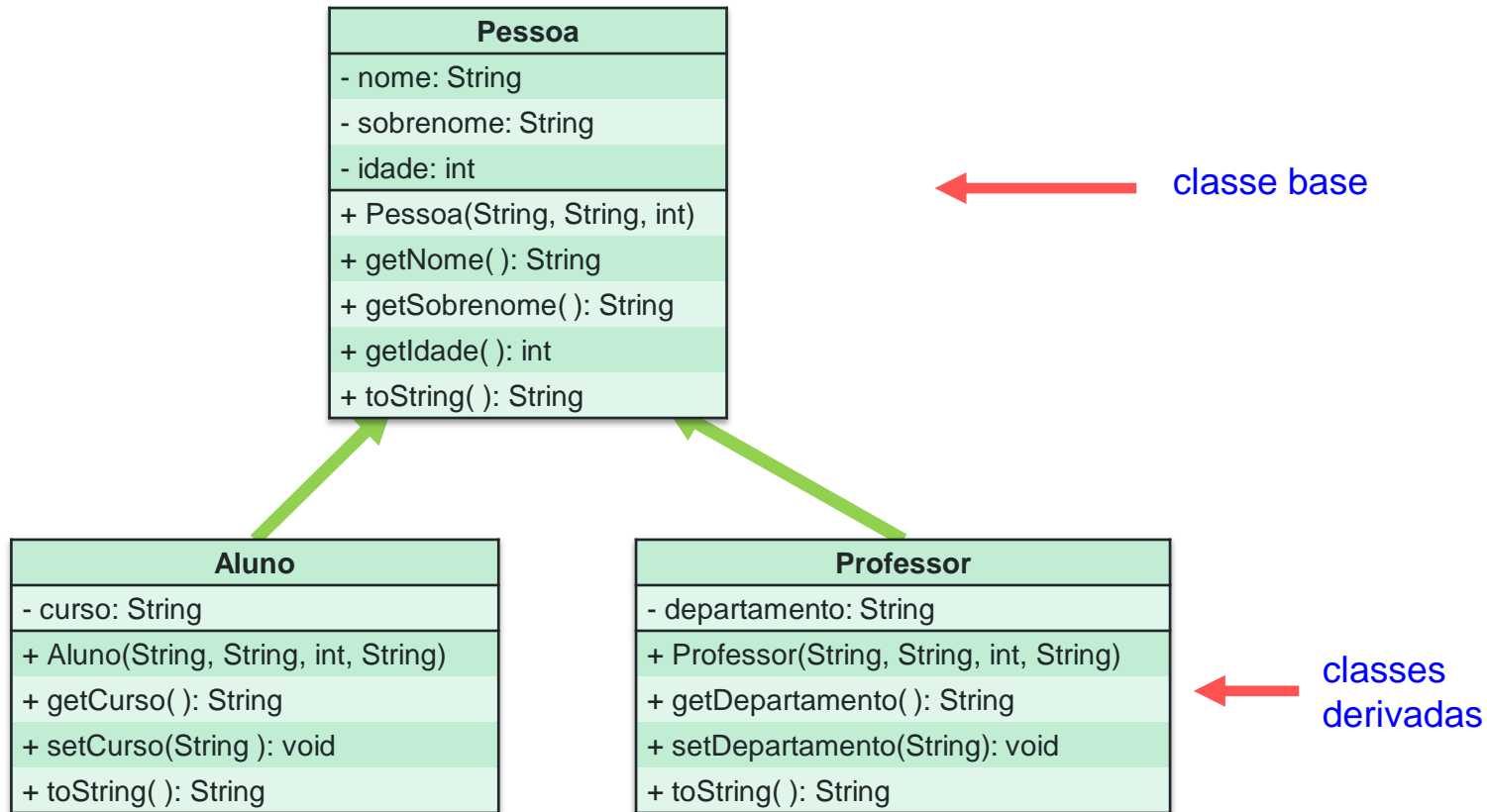
Exemplo

Duas classes que possui vários atributos e métodos iguais.

Aluno
- nome: String
- sobrenome: String
- idade: int
- curso: String
+ Aluno(String, String, int, String)
+ getName(): String
+ getSobrenome(): String
+ getIdade(): int
+ getCurso(): String
+ setCurso(String): void
+ toString(): String

Professor
- nome: String
- sobrenome: String
- idade: int
- departamento: String
+ Professor(String, String, int, String)
+ getName(): String
+ getSobrenome(): String
+ getIdade(): int
+ getDepartamento(): String
+ setDepartamento(String): void
+ toString(): String

Exemplo de Herança



Sintaxe

- Para criar uma classe herdeira de outra:

```
class ClasseDerivada (ClasseBase) :  
    <statement-1>  
  
    . . .  
  
    <statement-N>
```

- A classe ClasseDerivada é uma sub-classe de ClasseBase
- A classe ClasseBase é a super-classe de ClasseDerivada

Declarando uma Classe em Python

```
class Robot:  
    name = "Tiago++"  
    positionX = 0.0  
    positionY = 0.0  
    direction = 0.0
```

```
    def moveForward(self):  
        print ("Anda para frente")  
    def moveBackward(self):  
        print ("Anda para tras")  
    ...
```



Atributos



Métodos

Derivando uma Classe em Python

```
class RoboNey (Robot):  
    surname = "Santos"  
  
    def cai(self):  
        print ("Xi... Cai!")  
  
    def cavaPenalty(self):  
        print ("Oba! Cai na area!")  
  
...
```

Exemplo RoboticTeam

```
ROBOMAR = RoboNey("Neymar")
```

```
ROBOMAR.cai()
```

```
ROBOMAR.cavaPenalty()
```

```
print (ROBOMAR.name)
```

```
print (ROBOMAR.surname)
```

```
pass
```

Exemplo de Saída:

Construindo o Neymar :-)

Xi... Cai!

Oba! Cai na area!

Neymar

Santos

Bye bye!!

Derivando uma Classe reutilizando métodos

```
class RoboNey (Robot):  
    surname = "Santos"  
  
    def cai(self):  
        print ("Xi... Cai!")  
  
    def cavaPenalty(self):  
        self.moveForward()  
        self.turnLeft()  
        self.moveBackward()  
        print ("Oba! Cai na area!")  
    ...
```

Exemplo RoboticTeam

```
ROBOMAR = RoboNey("Neymar")
```

```
ROBOMAR.cavaPenalty()
```

```
pass
```

Exemplo de Saída:

Construindo o Neymar :-)
Anda para frente
Girando para a esquerda
Anda para tras
Oba! Cai na area!
Bye bye!!

Dicas:

- Em um grupo de classes relacionadas, coloque os atributos e métodos comuns na super-classe.
- Use a herança para criar sub-classes sem ter que repetir código.
- Herde somente da classe mais parecida com a que você precisa.
 - Herdar classes maiores desperdiça memória e processamento.

Inicialização

- Devemos inicializar todas as classes utilizadas em uma hierarquia.
- Para isso, fazemos uma “cascata” de construtores.

Derivando uma Classe reutilizando métodos

```
class Robot:
    name = ""
    def __init__(self, nome = "Tiago++"):
        self.name = nome
        print("Construindo o %s :-)" % self.name)

class RoboNey (Robot):
    surname = ""
    def __init__(self, nome = "Thiago2", sobrenome = "Pal"):
        self.surname = sobrenome
        Robot.__init__(self, nome)
```

Conclusão

Herança vs. Composição

é um versus tem um

é um (Herança)

- O objeto da classe derivada pode ser tratado com objeto da classe básica.
- Exemplo: O carro é um veículo.
 - Os atributos e métodos de veículos também se aplicam a carro

tem um (Composição)

- O objeto contém um ou mais objetos de outras classes como membros.
- Exemplo: O carro tem (uma) direção.

Conclusão

- Foi realizada uma breve apresentação dos 4 elementos principais de POO:
 - Classes
 - Objetos
 - Encapsulamento
 - Herança

Exercícios

Exercício 01

- Implemente a hierarquia de herança do sistema político.

