

XDES02

Programação Orientada a Objetos

Aula 07: Criação de classes

POO

- O paradigma OO provê uma série de conceitos que facilitam a criação e a manipulação de objetos. Entre eles está o conceito de classes e instâncias
- Classe
 - Modelo para vários objetos com características similares. Agrupa todas as características e funcionalidades de um conjunto particular de objetos

Um modelo

Lampada
<ul style="list-style-type: none">- estadoDaLampada
<ul style="list-style-type: none">- acende()- apaga()- mostraEstado()

Modelo lâmpada

O pseudocódigo

Modelo lampada // representa uma lâmpada em uso

Início do modelo

 dado do estadoDaLampada; // indica se a lâmpada está ligada ou não

 operacao acende() // acende a lâmpada

 início

 estadoDaLampada = aceso;

 fim

 operacao mostraEstado() // mostra estado da lâmpada

 início

 se (estadoDaLampada == aceso)

 imprime “A lâmpada está acesa”;

 senão

 imprime “A lâmpada está apagada”;

 fim

 fim do modelo

Outro modelo

Data	
-	dia
-	mes
-	ano
-	inicializaData(d, m, a)
-	dataEValida(d, m, a)
-	mostraData()

Modelo data

O pseudocódigo

```
Modelo Data
Inicio do modelo
    dado dia, mes, ano; // componentes da data
    // inicializa simultaneamente todos os dados para esta operação
    operacao inicializaData(umDia, umMes, umAno) // argumentos para esta operação
    inicio
        se dataEValida(umDia, umMes, umAno) // repassa os argumentos a operação
            inicio
                dia = umDia;
                mes = umMes;
                ano = umAno;
            fim
        senão
            inicio
                dia = 0;
                mes = 0;
                ano = 0;
            fim
    fim
```

Modelo data (cont)

O pseudocódigo

```
operacao dataEValida(umDia, umMes, umAno) // argumentos para a operação
    inicio
        // Se a data passada for válida, retorna verdadeiro
        se ((dia >= 1) e (dia <= 31) e (mes >= 1) e (mes <= 12))
            retorna verdadeiro;
        senão // senão retorna falso
            retorna falso;
    fim
```

```
operacao mostraData( ) // mostra a data imprimindo valores de seus dados
    inicio
        imprime dia;
        imprime "/";
        imprime mes;
        imprime "/";
        imprime ano;
    fim
```

Fim do modelo

Classe = Modelo

- Então...
 - Classe define um **modelo** de dados e as **operações** sobre esses dados
- Operação de instanciação
 - Cria representações concretas do modelo abstrato
 - Objetos

Criando uma classe

- Motocicleta.py

Motocicleta
- marca : String - cor : String - motorLigado : boolean

```
class Motocicleta:  
  
    # construtor  
    def __init__(self, marca, cor, motorLigado):  
        self.__marca = marca  
        self.__cor = cor  
        self.__motorLigado = motorLigado
```

Criando uma classe

- Acrescentando comportamento
 - Através da **criação de métodos** é possível determinar o que se pode fazer com cada objeto da classe criada

```
def ligaMotor(self):  
    if self.__motorLigado == True:  
        print('O motor já está ligado!')  
    else:  
        self.__motorLigado = True  
        print('O motor acaba de ser ligado!')
```

Criando uma classe

Motocicleta
- marca : String
- cor : String
- motorLigado : boolean
+ ligaMotor() : void

```
class Motocicleta:

    # construtor
    def __init__(self, marca, cor, motorLigado):
        self.__marca = marca
        self.__cor = cor
        self.__motorLigado = motorLigado

    # método de instância
    def ligaMotor(self):
        if self.__motorLigado == True:
            print('O motor já está ligado!')
        else:
            self.__motorLigado = True
            print('O motor acaba de ser ligado!')
```

Criando uma classe

- Esta classe já é capaz de caracterizar seus objetos (através de atributos) e já possui um comportamento (método) especificado
 - Este **comportamento** tem como função **mudar o estado** de um objeto através da modificação do valor de um de seus atributos
- É possível especificar métodos que realizam ações com um objeto **sem mudar seu estado**
 - Exemplo: método `mostraAtributos()`

Criando uma classe

Motocicleta
- marca : String - cor : String - motorLigado : boolean
+ ligaMotor() : void + mostraAtributos() : void

```
def mostraAtributos(self):  
    print('Esta motocicleta é uma {} {}'.format(self.__marca, self.__cor))  
    if(self.__motorLigado):  
        print('Seu motor está ligado')  
    else:  
        print('Seu motor está desligado')
```

Criando uma classe

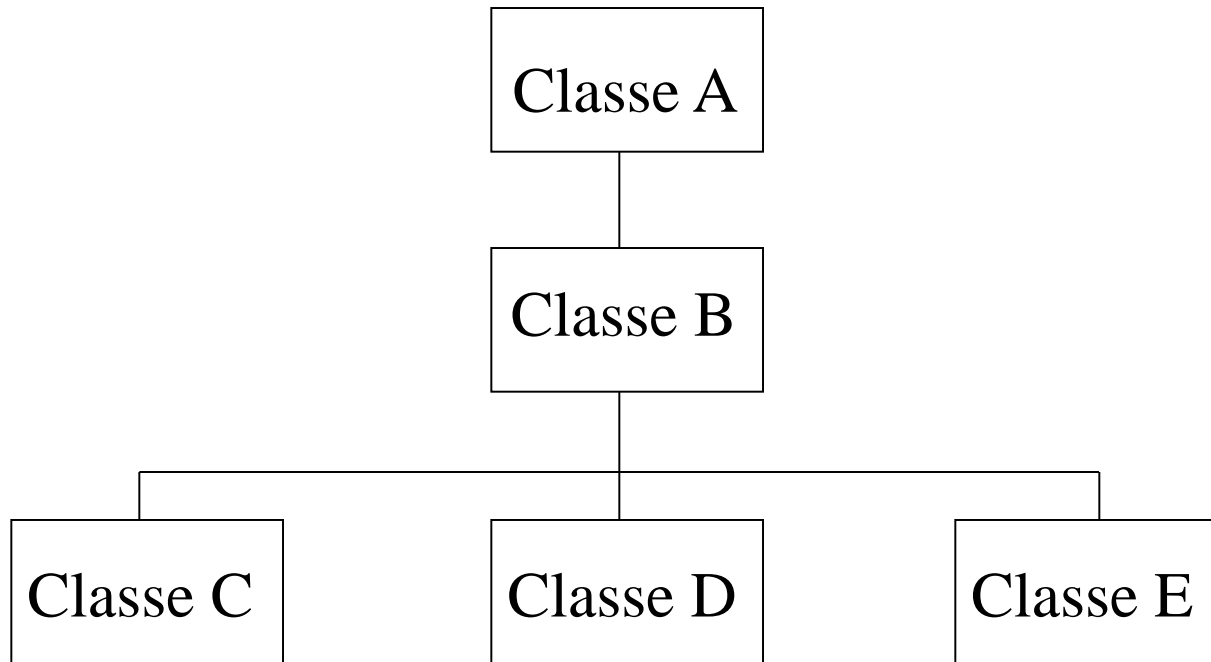
- Vamos agora instanciar nossa classe e chamar seus métodos

```
moto1 = Motocicleta('Honda', 'vermelha', False)
moto1.mostraAtributos()
print()
moto1.ligaMotor()
print()
moto1.mostraAtributos()
print()
moto1.ligaMotor()
```

Herança

- Conceito básico da POO através do qual pode-se definir uma classe (classe filha ou **subclasse**) a partir de uma ou mais classes já existentes (classes pai ou **superclasses**)
- Segundo este conceito, uma subclasse herda atributos e comportamentos de sua(s) classe(s) pai

Herança



Herança

- Quando se define uma **subclasse**, esta automaticamente **herda** da superclasse (e de suas superclasses) seus **atributos** e **métodos**, podendo usá-los diretamente sem que tenha que definí-los novamente
 - A subclasse estende a superclasse provendo novos atributos e comportamentos. Ex:
 - Árvore
 - Árvore frutífera

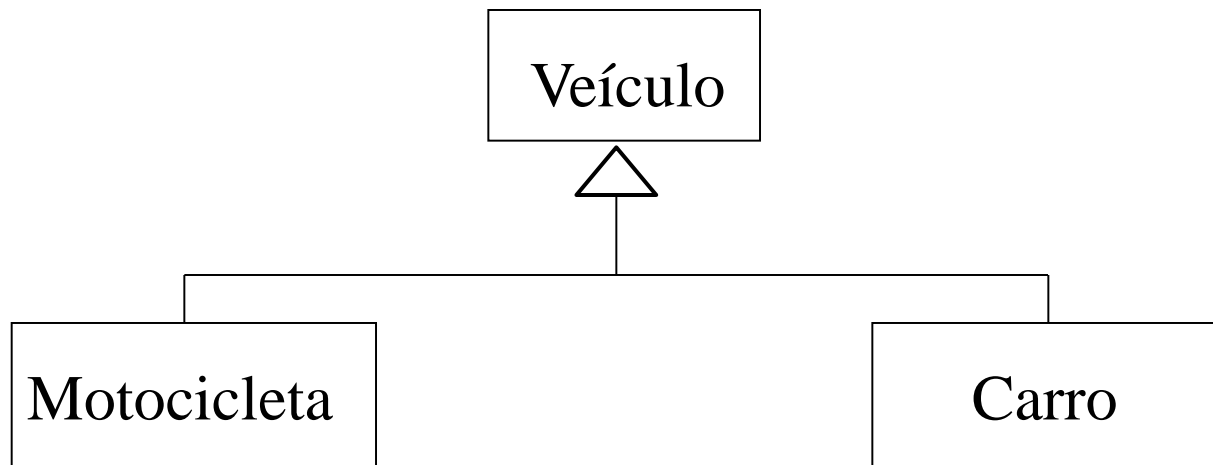
Herança

- Criação da classe carro
 - Carro: um carro possui uma série de coisas em comum com uma moto:
 - motor
 - rodas
 - velocímetro
 - marca/cor/modelo
 - Posso criar a classe carro **copiando** parte do código da classe Motocicleta?

Herança

- Criação da classe carro
 - Copiando código você teria informação em duplicidade e estaria indo na contra mão da POO
 - Uma melhor solução, em concordância com os princípios da POO, seria criar uma **superclasse** para as classes *Motocicleta* e *Carro*, na qual pudessem ser colocados os atributos e comportamentos comuns às duas classes em questão

Herança



Exemplo de Herança

```
class Animal:

    # construtor
    def __init__(self, nome):
        self.__nome = nome

    # getters
    def getNome(self):
        return self.__nome

    # metodo
    def fazerCarinho(self):
        print("{} está recebendo carinho".format(self.__nome))
```

```
if __name__ == "__main__":
    gato = Gato("Perola")
    gato.miar()
    print(gato.getNome())
    cao = Cao("Cleitin")
    cao.latir()
    print(cao.getNome())
    cao.fazerCarinho()
```

```
class Gato(Animal):

    # construtor
    def __init__(self, nome):
        # super pertmír utilizar metodos da superclasse
        super().__init__(nome)

    # metodo
    def miar(self):
        print("Meow Meow")
```

```
class Cao(Animal):

    # construtor
    def __init__(self, nome):
        # super pertmír utilizar metodos da superclasse
        super().__init__(nome)

    # metodo
    def latir(self):
        print("Au Au")
```

Exercício 1

- Elabore um programa contendo quatro classes:
 - Veículo
 - Carro
 - Motocicleta
- Considere os seguintes atributos
 - marca
 - cor
 - motorLigado (boolean)
 - estilo: trail, naked, custom
 - portaMalasCheio (boolean)
- Considere as seguintes operações
 - Liga/desliga motor
 - enche/esvazia porta malas
 - mostraAtributos
- Deve-se instanciar um carro e uma moto. Deve-se ligar a moto e mostrar seus atributos. Em seguida, deve-se encher o porta malas do carro, ligá-lo e mostrar seus atributos
- Nota: Atributos e operações comuns devem ficar na classe de mais alto nível na hierarquia.