Aula 05

Roteiro

- Programação Orientada a Objetos
  - Classes e Objetos
  - Atributos e Métodos
  - Encapsulamento e Visibilidade
  - Herança
  - Polimorfismo

PROGRAMAÇÃO ORIENTADA A OBJETOS (POO)

Programação Orientada a Objetos

- A linguagem de programação *Python* é ***multiparadigma***, assim, é possível programar usando:
  - Paradigma Procedural
  - Paradigma Funcional
  - Paradigma Orientada a Objetos
- Ou ainda,
  - Misturar todos estes paradigmas

Programação Orientada a Objetos

- ***Classes***
  - Modelo formal para criação de objetos.
  - Tipo abstrato de dados.
- ***Objetos***
  - Abstração de objetos reais existentes.
  - Instancias das classes

Programação Orientada a Objetos

- ***Atributos***
  - Responsável pela manipulação e armazenamento dos dados dos objetos.
- ***Métodos***
  - utilizados para definir os comportamentos que serão executados pelos objetos.

Classes, Objetos, Atributos e Métodos

```
public class Carro {

    private String cor;
    private double potencia;
    private int ano;
    private String modelo;
    private String fabricante;

    public Carro() {
    }
    public void ligar(){
        System.out.println("Carro ligado");
    }
    public void desligar(){
        System.out.println("Carro desligado");
    }
    public void acelerar(){
        System.out.println("Acelerando...");
    }
    public void frear(){
        System.out.println("Freando...");
    }
    public void trocarmarcha(){
        System.out.println("Trocar marcha");
    }
}
```

Programação Orientada a Objetos

- Definição de uma ***classe*** em ***Python***

```python
class Pessoa:
    def setNome(self, nome):
        self.nome = nome

    def getNome(self):
        return self.nome

    def setIdade(self, idade):
        self.idade = idade

    def getIdade(self):
        return self.idade
```

Programação Orientada a Objetos

- Definição de uma ***classe*** em ***Python***

```
1   class Pessoa:
2       def setNome(self, nome):
3           self.nome = nome
4
5       def getN
6           retu
7
8       def setI
9           self
10
11      def getIdade(self):
12          return self.idade
```

A instrução ***self*** é utilizada para indicar que a definição da ***função*** deve ser tratada como ***método de uma classe***.

Programação Orientada a Objetos

- ***Instância*** de objetos

```
17    joao = Pessoa()
18    joao.nome = "João da Silva"
19    joao.idade = 22
20
21    print("%s %d" %(joao.nome,joao.idade))
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMIN

```
João da Silva 22
```

Programação Orientada a Objetos

- ***Instância*** de objetos

```
17    joao = Pessoa()
18    joao.setNome("João da Silva")
19    joao.setIdade(22)
20    print(joao)
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERM

```
<__main__.Pessoa object at 0x0000025F2E2AF400>
```

Programação Orientada a Objetos

- ***Instância*** de objetos

```
17    joao = Pessoa()
18    joao.setNome("João da Silva")
19    joao.setIdade(22)
20    print(joao)
21
```

PROBLEMS OUTPUT DEBUG CONSOLE TERM

```
<__main__.Pessoa object at 0x0000025F2E2AF400>
```

Resultado da chamada do método ***str*** da classe ***object***.

Programação Orientada a Objetos

- ***Sobreposição*** de métodos
  - A linguagem *Python* possui uma série de métodos com __ (***underscores***), tais como:
    - __str__ retorna dados no formato str
    - __len__ retorna o tamanho de uma variável
    - Dentre outros
  - Estes métodos são denominados ***métodos mágicos*** e resolvem uma série de problemas.

Programação Orientada a Objetos

- ***Sobreposição*** de métodos
  - Para substituir a implementação padrão destes métodos é necessário utilizar o conceito de ***polimorfismo por sobreposição***.

```
24          #sobreposição do método __str__
25          def __str__(self):
26              return "%s %s" %(self.nome,self.idade)
27
28      joao = Pessoa("João da Silva", 22)
29      print(joao)
```

Programação Orientada a Objetos

- ***Instância*** de objetos

Os parâmetros indicados nos métodos são tratados como ***atributos públicos***.

```
17    joao = Pessoa()
18    joao.nome = "João da Silva"
19    joao.idade = 22
20
21    print("%s %d" %(joao.nome,joao.idade))
22
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMIN

```
João da Silva 22
```

Programação Orientada a Objetos

- ***Visibilidade***
  - ***Público***: acessível em qualquer parte do código.
  - ***Protegido:*** acessível na própria classe e nas classes derivadas.
  - ***Privado***: acessível apenas na própria classe.

```
def metodo(self, a,b,c):
    self.a = a        #público
    self._b = b       #protegido
    self.__c = c      #privado
```

Programação Orientada a Objetos

- ***Construtores***

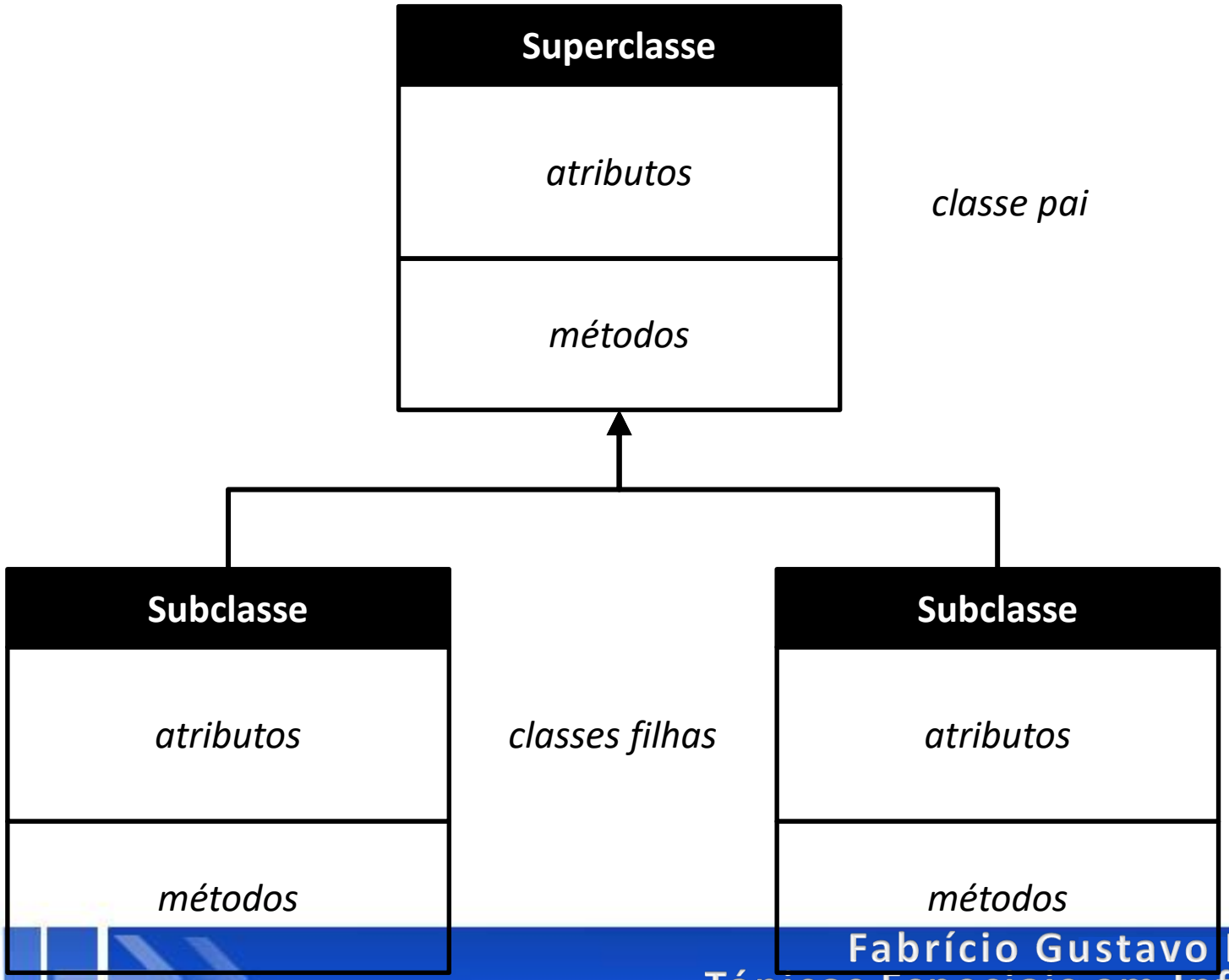
```
4   class Pessoa:
5
6       #construtor
7       def __init__(self, nome, idade):
8           self.nome = nome
9           self.idade = idade
10
```

```
23   joao = Pessoa("João da Silva", 22)
```

Programação Orientada a Objetos

- ***Herança***
  - Metodologia clara e objetiva para reutilização de software.
  - Especificação de novas classes a partir de definições de classes existentes.
  - Novas classes criadas *herdam* características e funcionalidades de classes previamente especificadas.
  - Todas as classes em Python ***herdam*** implicitamente da classe ***object.***

Herança

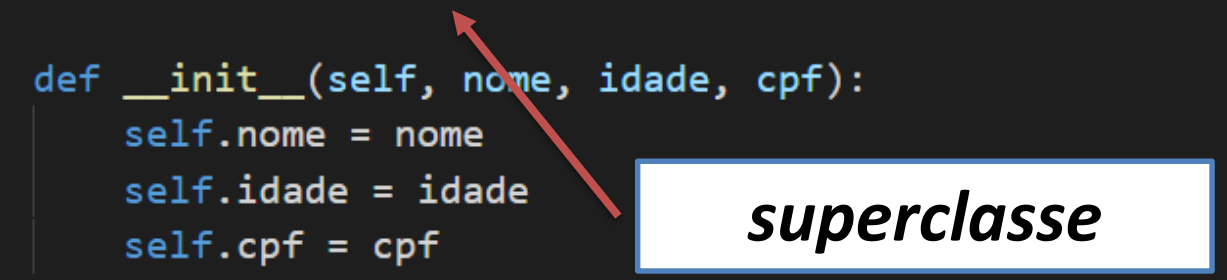

Superclasse
atributos
métodos
classe pai
Subclasse
atributos
métodos
classes filhas
Subclasse
atributos
métodos

```python
class PessoaFisica(Pessoa):

    def __init__(self, nome, idade, cpf):
        self.nome = nome
        self.idade = idade
        self.cpf = cpf

    def setCpf(self, cpf):
        self.cpf = cpf

    def getCpf(self):
        return self.cpf

    def __str__(self):
        return "%s \n%s" %(super().__str__(), self.cpf)

joao = PessoaFisica("João da Silva", 22, "123.456.789-00")
print(joao)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
João da Silva 22
123.456.789-00
```

```python
class PessoaFisica(Pessoa):

    def __init__(self, nome, idade, cpf):
        self.nome = nome
        self.idade = idade
        self.cpf = cpf

    def setCpf(self, cpf):
        self.cpf = cpf

    def getCpf(self):
        return self.cpf

    def __str__(self):
        return "%s \n%s" %(super().__str__(), self.cpf)

joao = PessoaFisica("João da Silva", 22, "123.456.789-00")
print(joao)
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
João da Silva 22
123.456.789-00
```

```python
class PessoaFisica(Pessoa):

    def __init__(self, nome, idade, cpf):
        self.nome = nome
        self.idade = idade
        self.cpf = cpf

    def setCpf(self, cpf):
        self.cpf = cpf

    def getCpf(self):
        return self.cpf

    def __str__(self):
        return "%s \n%s" %(super().__str__(), self.cpf)

joao = PessoaFisica("João da Silva", 22, "123.456.789-00")
print(joao)
```

Chamada explícita de métodos da superclasse

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
João da Silva 22
123.456.789-00
```

Programação Orientada a Objetos

- Funções Auxiliares

```
55  #Verificar se um objeto é de uma determinada classe
56  p = PessoaFisica("Ana Maria", 21, "123.456-789-00")
57  print(isinstance(p, Pessoa))
58  print(isinstance(p, PessoaFisica))
59
60  #Descobrir quem é a superclase
61  print(PessoaFisica.__bases__)
62
63  #Descobrir se uma classe é subclasse de outra
64  print(issubclass(PessoaFisica,Pessoa))
65
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
True
True
(<class '__main__.Pessoa'>,)
True
```

ATIVIDADE PRÁTICA

Atividade Prática

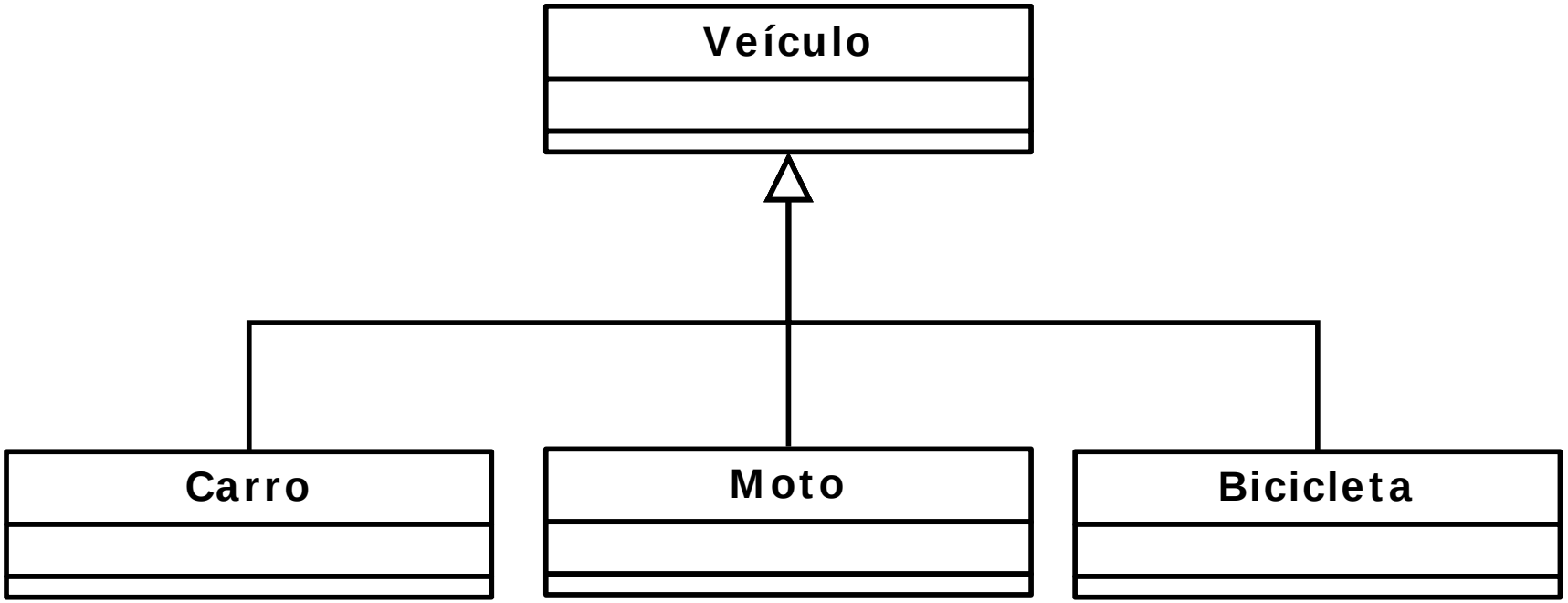
- ***Exercício 1:***
  - Escreva uma classe denominada ***Televisor*** que contenha dois atributos volume e canal.
  - Especifique os seguintes métodos:
    - ***aumentarVolume()***
    - ***reduzirVolume()***
    - ***trocarCanal(int*** canal)
  - Por fim, demonstre a utilização da classe

Atividade Prática

- ***Exercício 2:***
  - Escreva uma classe denominada ***Funcionario*** que contenha os atributos encapsulados: ***nome, salario*** e ***ano de contratação***.
  - Especifique os métodos modificadores de acesso ***set/get*** para cada atributo, bem como, os seguintes métodos:
    - ***getBonificacao()***
      - 5% de bonificação para mais de 5 anos
      - 10% de bonificação para mais de 10 anos
      - 20% de bonificação para mais de 20 anos
    - ***getSalarioTotal()***
  - Por fim, demonstre a utilização da classe

Atividade Prática

- ***Exercício 3***
  - Considere a seguinte hierarquia de classes:



Atividade Prática

- ***Exercício 3 (continuação)***
  - Especifique as classes e seus respectivos atributos. Cada classe deverá conter pelo menos dois atributos.
  - Na classe Carro crie um método denominado *velocidadeMaxima* capaz de indicar se o veículo atingiu a velocidade máxima permitida.