Programação Orientada a Objetos

Polimorfismo

Classes Abstratas

Métodos Abstratos

Prof. Paulo Viniccius Vieira

paulo.vieira@faculdadeimpacta.com.br

Polimorfismo

- A Programação Orientada a Objetos se baseia em 4 pilares:
  - Abstração:
    - Processo de representar entidades do mundo real
  - Encapsulamento:
    - Permite Ocultar detalhes internos de uma classe
  - Herança:
    - Facilita o reuso do código
  - **Polimorfismo:**
    - Possibilidade de um objeto se comportar de diferentes formas

Polimorfismo

- Polimorfismo é um conceito muito importante dentro da programação.

- O polimorfismo não ocorre somente na POO.
  - Pode ocorrer em programação estruturada.

- Refere-se a possibilidade de uma entidade (método, operador ou objeto) ser utilizada para representar diferentes tipos de dados em diferentes cenários.

Polimorfismo

- Na POO, o polimorfismo permite que objetos se comportem de acordo com a classe à qual pertencem, ou de acordo com uma superclasse mais genérica (o objeto se comporta de formas diferentes).

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def comer(self):
        print("Animal Comendo")

class Cachorro(Animal):
    def __init__(self, nome, raca):
        super().__init__(nome)
        self.raca = raca

    def latir(self):
        print("O Cachorro está latindo")

cachorro = Cachorro("Rex", "Rottweiler")
cachorro.comer()                    # Animal Comendo
cachorro.latir()                    # O Cachorro está latindo
```

Comportamento da classe Animal

Comportamento da classe Cachorro

Polimorfismo

- O Polimorfismo também ocorre quando uma subclasse sobrescreve algum método herdado da superclasse.

```
class Animal:
    def __init__(self, nome):
        self.nome = nome

    def comer(self):
        print("Animal Comendo")


class Cachorro(Animal):
    def __init__(self, nome, raca):
        super().__init__(nome)
        self.raca = raca

    def comer(self):
        print("Cachorro", self.nome, "Comendo")


cachorro = Cachorro("Rex", "Rottweiler")
cachorro.comer()                    # Cachorro Rex Comendo
```

Método comer foi sobrescrito na classe-filha

Classes e Métodos Abstratos

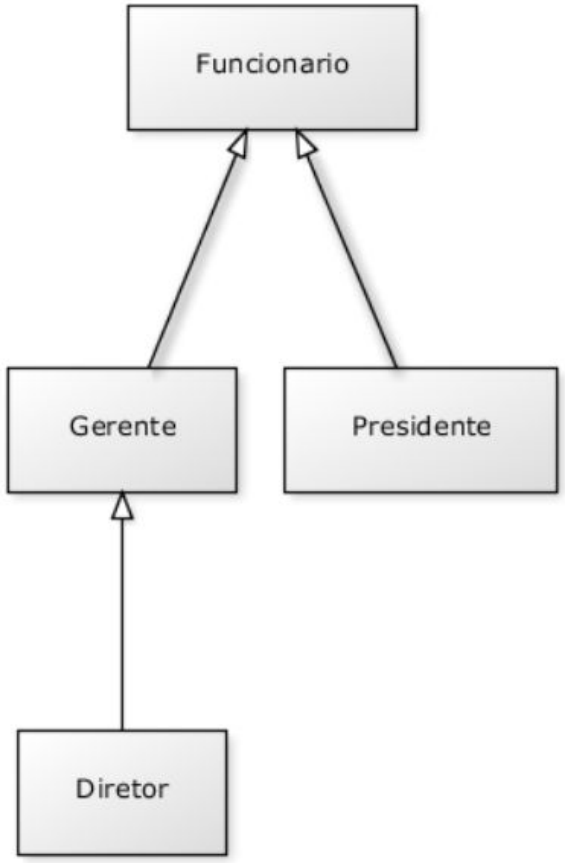
Classes Abstratas

- **Classe Abstrata** é uma classe que não pode ser instanciada, ou seja, não gera objetos.
  - Geralmente são utilizadas para definir uma estrutura básica para suas classes filhas.

- **Classe Concreta** é uma classe que pode ser instanciada diretamente, ou seja, pode gerar objetos.
  - Classes que utilizamos até agora

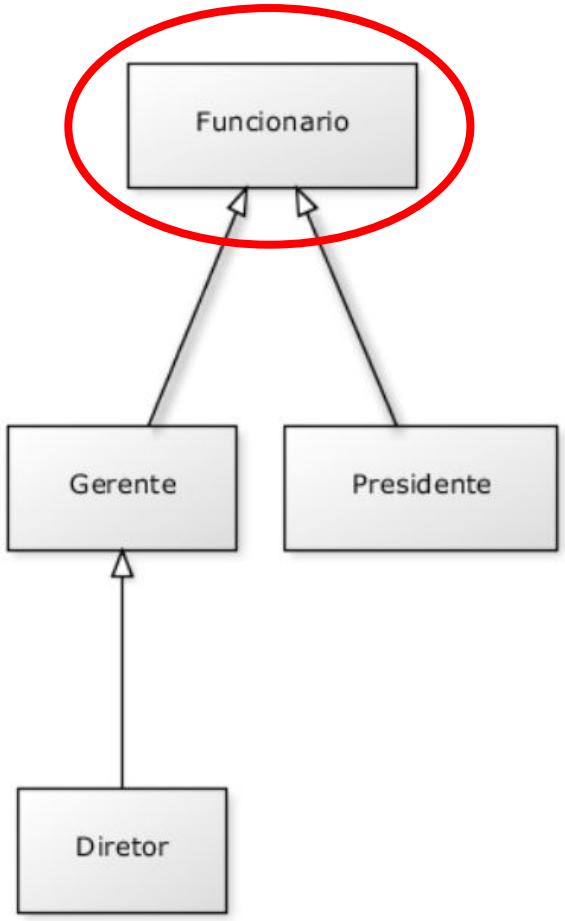
Classes Abstratas

- Exemplo: Numa hierarquia de classes, todas as classes concretas podem ser instanciadas
  - Nesse exemplo, podemos criar objetos das classes **`Funcionario`**, **`Gerente`**, **`Presidente`** e **`Diretor`**



Classes Abstratas

- Exemplo: Se a classe **`Funcionario`** for abstrata, ela não poderá ser instanciada.
  - Nesse caso, somente poderíamos criar objetos das classes-filhas **`Gerente`**, **`Presidente`** e **`Diretor`**

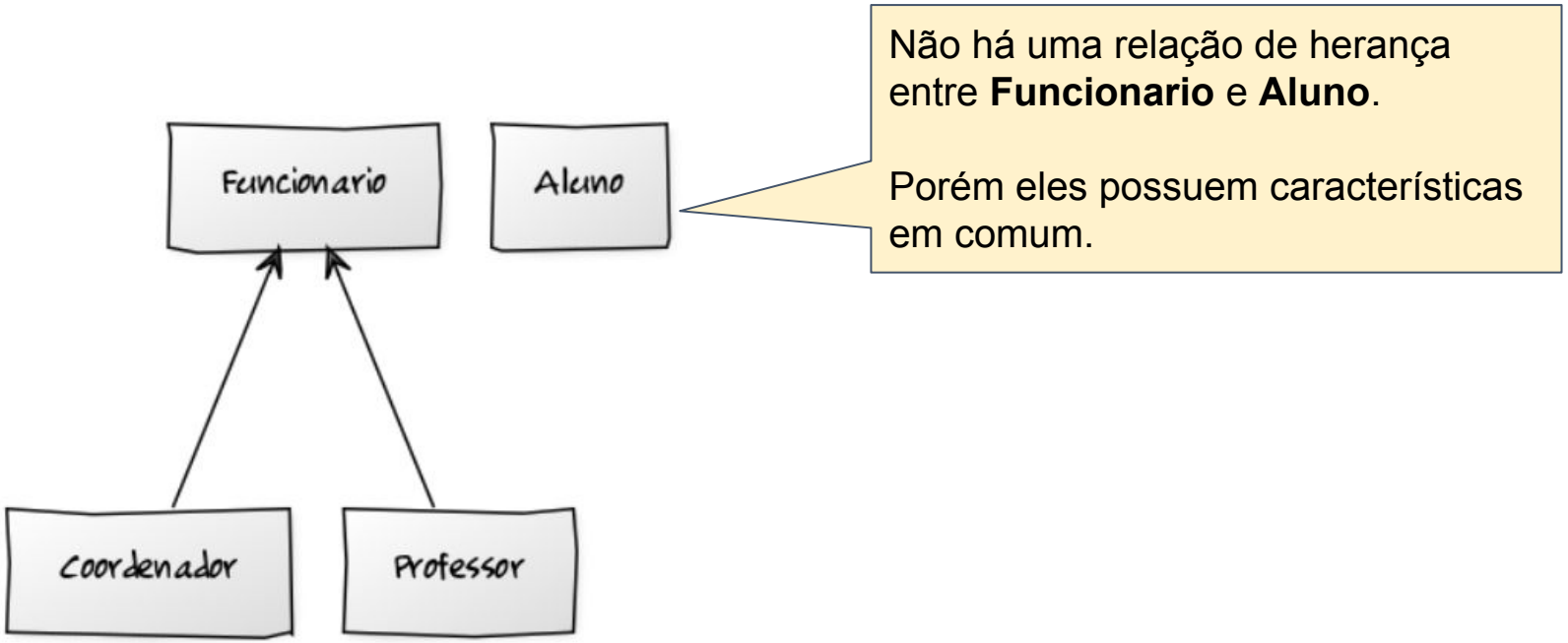


Classes Abstratas

- A decisão de transformar uma classe em abstrata depende do domínio do problema.
  - Em um determinado sistema pode fazer sentido ter um determinado objeto.
  - E em outro sistema, esse mesmo objeto pode não fazer sentido.

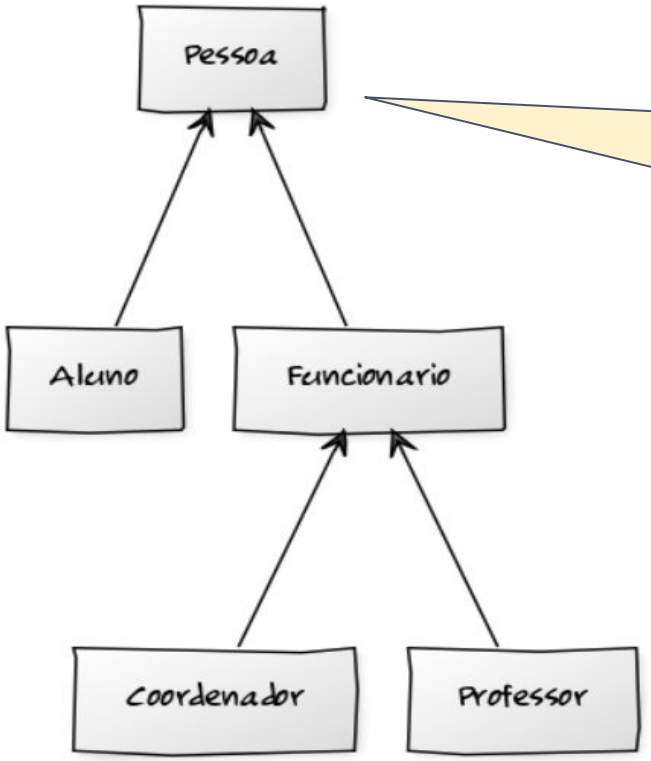
Classes Abstratas

- Uma classe abstrata pode servir para agrupar informações comuns às classes filhas, quando não faz sentido existir na aplicação objetos instanciados diretamente dela.



Classes Abstratas

- Uma classe abstrata pode servir para agrupar informações comuns às classes filhas, quando não faz sentido existir na aplicação objetos instanciados diretamente dela.



Características em comum do **Funcionario** e **Aluno** podem ser colocadas na classe **Pessoa**.

Porém não faz sentido ter um objeto **Pessoa** no sistema (Pessoa pode ser então uma classe abstrata).

Classes Abstratas

Devemos importar a classe **ABC** do módulo **abc** (abstract base classes)

A classe abstrata deve herdar da classe **ABC**

Somente as classes filhas devem ser instanciadas

```
from abc import ABC

class Pessoa(ABC):
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

class Funcionario(Pessoa):
    def __init__(self, nome, idade, salario):
        super().__init__(nome, idade)
        self.salario = salario

class Aluno(Pessoa):
    def __init__(self, nome, idade, disciplina):
        super().__init__(nome, idade)
        self.disciplina = disciplina

funcionario = Funcionario("João", 25, 1500.0)
aluno = Aluno("Pedro", 19, "Programação")
```

Métodos Abstratos

- As classes abstratas podem ser utilizadas para definir uma estrutura básica para as suas classes filhas.
  - Para isso, uma classe abstrata pode definir métodos abstratos

- **Método abstrato** é um método que não possui implementação.
  - As classes filhas são obrigadas a implementar os métodos abstratos da classe-mãe.

```python
from abc import ABC, abstractmethod

class Pessoa(ABC):
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    @abstractmethod
    def exibir_nome(self):
        pass

class Funcionario(Pessoa):
    def __init__(self, nome, idade, salario):
        super().__init__(nome, idade)
        self.salario = salario

    def exibir_nome(self):
        print("Nome do Funcionario: ", nome)

class Aluno(Pessoa):
    def __init__(self, nome, idade, disciplina):
        super().__init__(nome, idade)
        self.disciplina = disciplina

    def exibir_nome(self):
        print("Nome do Aluno: ", nome)

funcionario = Funcionario("João", 25, 1500.0)
aluno = Aluno("Pedro", 19, "Programação")
```

Devemos importar a classe **`abstractmethod`** do módulo **`abc`**

Para definir o método abstrato utiliza-se o comando **`@abstractmethod`** antes da definição do método

O método abstrato deve ser **vazio** (contém apenas um *pass*).

Os métodos abstratos devem ser implementados, **obrigatoriamente**, em todas as classes filhas

Interfaces

- Uma interface **define** um mecanismo da programação orientada a objetos que permite o reuso da declaração de métodos e estabelece o comportamento básico de um conjunto de classes.
  - Python não possui uma sintaxe própria para determinar uma interface.
  - No entanto, podemos definir uma interface implementando uma classe abstrata que possui apenas métodos abstratos.

- Todas as classes que herdam da interface precisam implementar os métodos definidos.

Interfaces

```
from abc import ABC, abstractmethod

class Animal(ABC):
    def __init__(self, nome, idade):
        self.nome = nome
        self.idade = idade

    @abstractmethod
    def comer(self):
        pass

    @abstractmethod
    def andar(self):
        pass

    @abstractmethod
    def dormir(self):
        pass
```

Qualquer classe que herdar da classe **`Animal`** será obrigada a implementar os métodos abstratos que foram definidos na 'interface'.