

Herança e Classes Abstratas

Programação Orientada a Objetos

Prof. Jônata Tyska Carvalho



UFSC



CTC • UFSC
Informática e estatística

Herança: Agenda

1. Intuição e conceito
2. Benefícios e características
3. Classes Abstratas



Herança: Agenda

1. **Intuição e conceito**
2. Benefícios e características
3. Classes Abstratas



Herança - Generalização

Generalização:



A é um tipo de B

Composição:



A contém B, A é composto por B
B é propriedade exclusiva de A
A instancia e destrói B

Agregação:



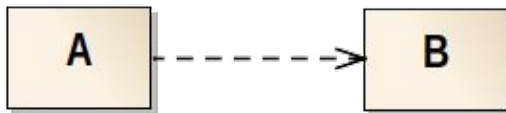
A contém B, A é formado por B
B é compartilhável

Associação:



A tem atributo
do tipo B

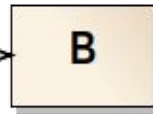
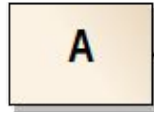
Dependência:



Dependência de A para B
não é permanente

Herança - Generalização

Generalização:



A é um tipo de B



Herança

Exemplo

Carro
-modelo: str -cor: str -placa: str -velocidade: int -sujo: bool
+lavar() +buzinar() +acelerar() +frear()

Bicicleta
-modelo: str -cor: str -velocidade: int -sujo: bool -marchas: int -amortecedor: bool
+lavar() +pedalar() +frear()

Herança

Intuição e conceito – O que é isso?



Herança

Intuição e conceito – O que é isso?



Herança

Intuição e conceito – O que é isso?



Herança

Intuição e conceito – O que é isso?



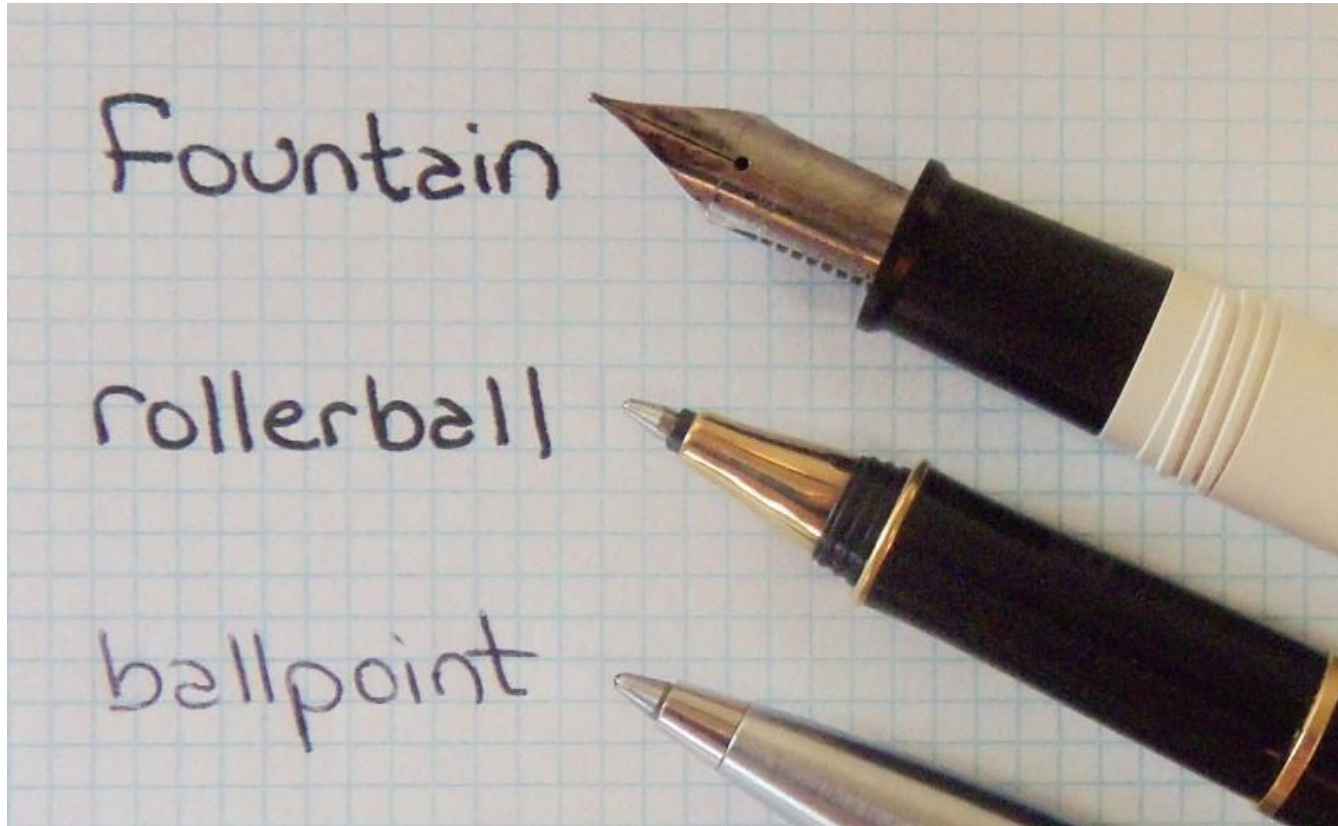
Intuição e conceito – Características comuns

Caneta



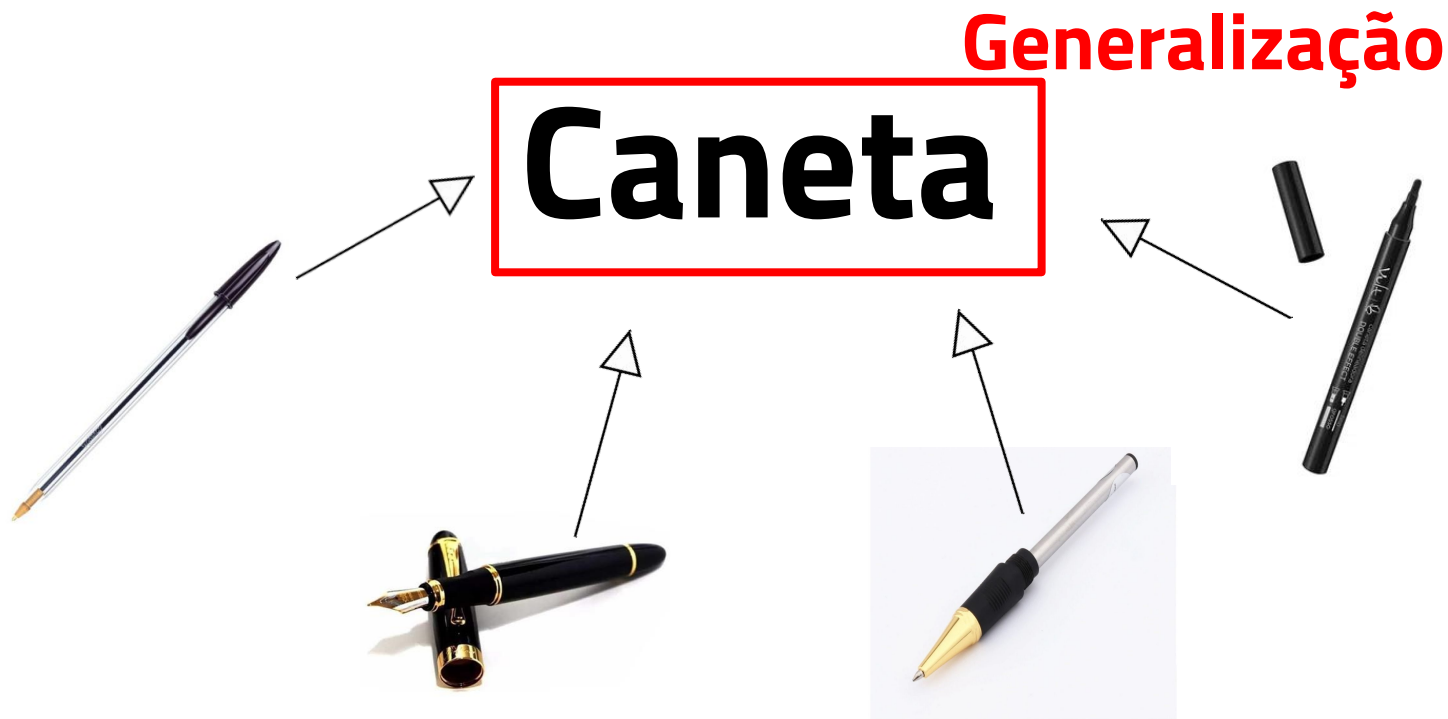
Herança

Intuição e conceito – Características Específicas



Herança

Intuição e conceito – abstração



Herança

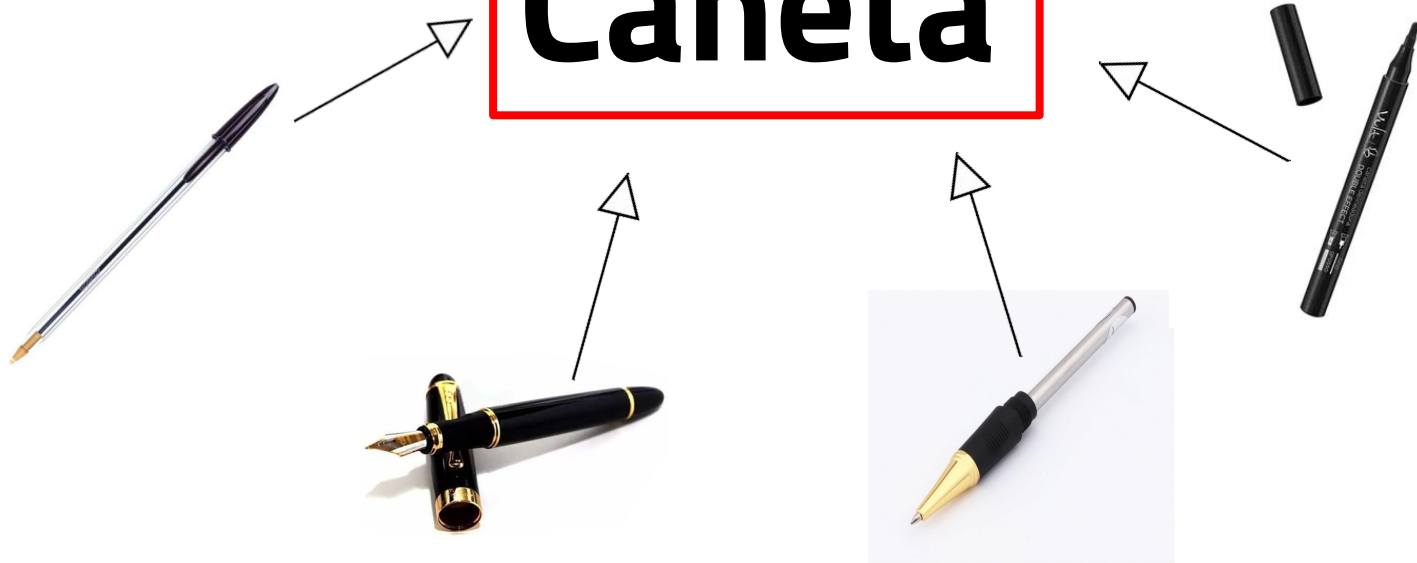
Intuição e conceito – abstração

Evitar a replicação

Reusabilidade

Generalização

Caneta



Herança

Intuição e conceito – especializar – refinar

Reusabilidade
Extensibilidade

Caneta



Especializações

Herança: Agenda

1. Intuição e conceito
- 2. Benefícios e características**
3. Classes Abstratas



Herança – Benefícios e Características

Ideia básica

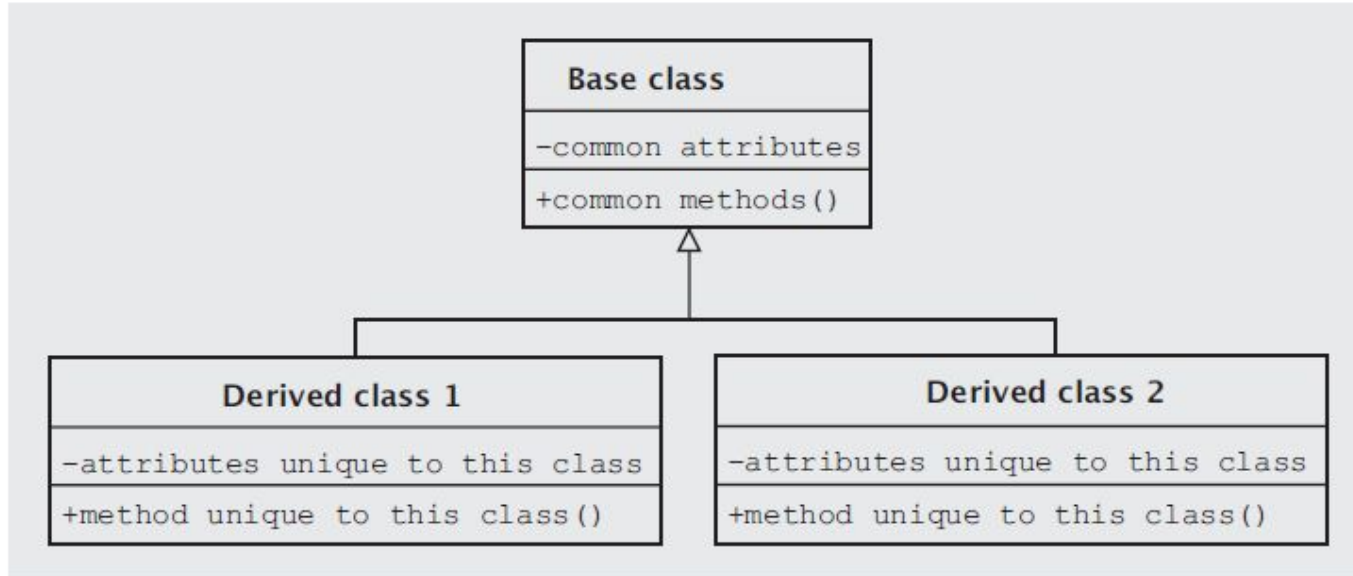


Figure 3.5 *Basic idea of inheritance*

Herança – Benefícios e Características

Ideia básica – termos

Superclasse
Classe pai
Classe base

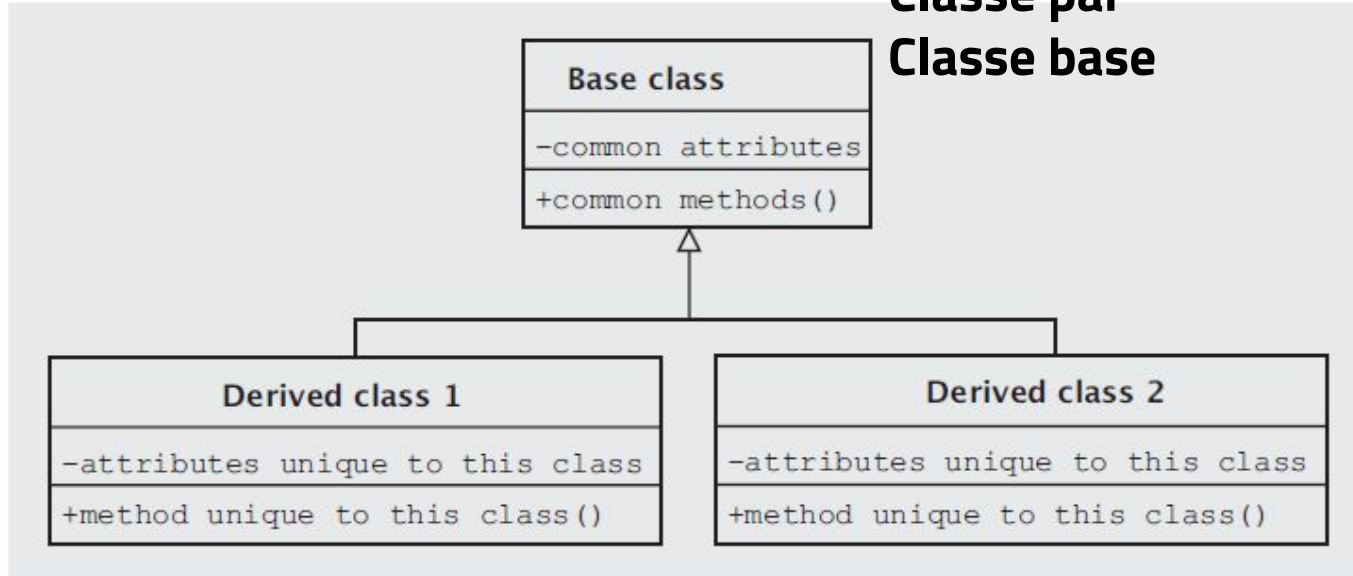
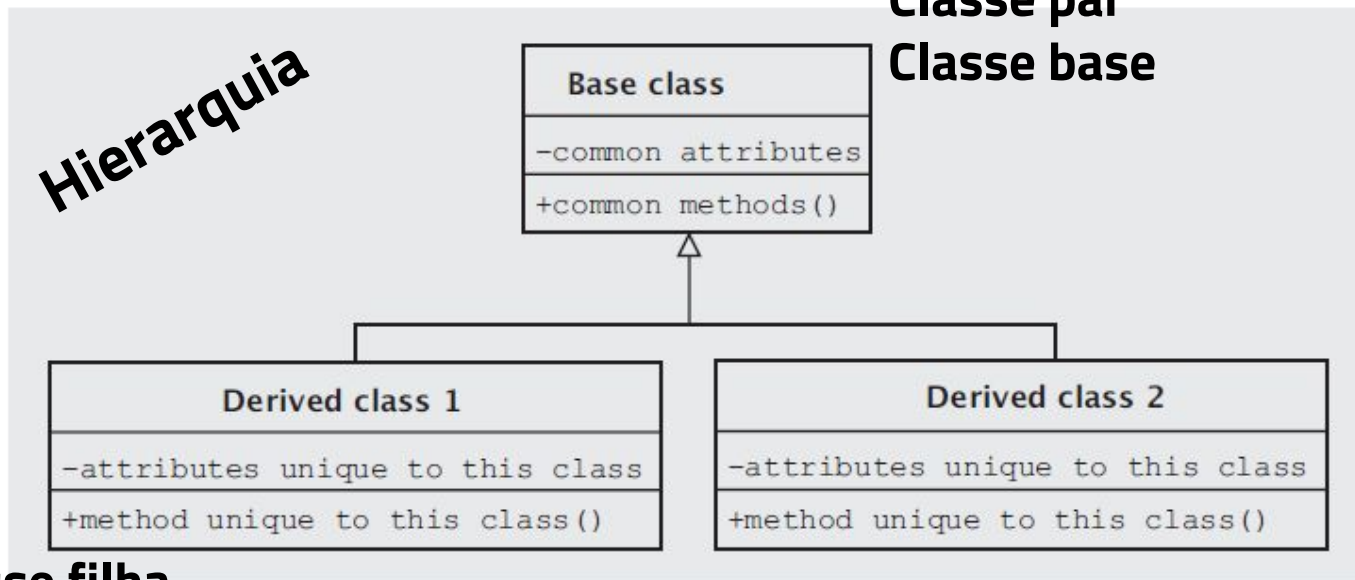


Figure 3.5 *Basic idea of inheritance*

Herança – Benefícios e Características

Ideia básica – termos



Superclasse
Classe pai
Classe base

Classe filha

Classe derivada

Subclasse

Figure 3.5 Basic idea of inheritance

DATHAN, B.; RAMNATH, S. Object-Oriented Analysis, Design and Implementation. Cham: Springer, 2015

Herança – Benefícios e Características

Exemplo simples

Carro
-modelo: str -cor: str -placa: str -velocidade: int -sujo: bool
+lavar() +buzinar() +acelerar() +frear()

Bicicleta
-modelo: str -cor: str -velocidade: int -sujo: bool -marchas: int -amortecedor: bool
+lavar() +pedalar() +frear()

Herança – Benefícios e Características

Exemplo simples

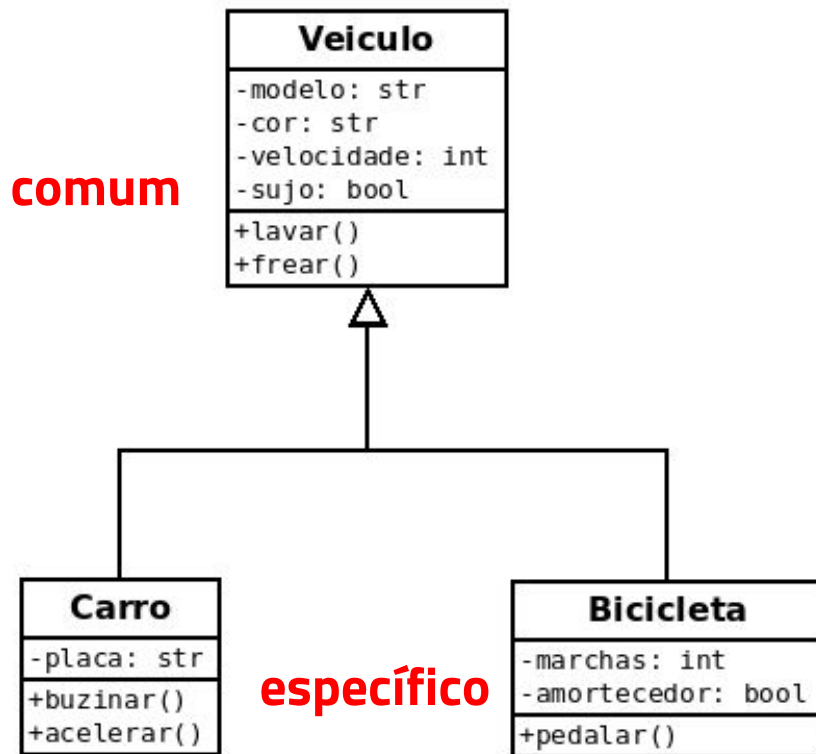
Carro
-modelo: str -cor: str -placa: str -velocidade: int -sujo: bool
+lavar() +buzinar() +acelerar() +frear()

Bicicleta
-modelo: str -cor: str -velocidade: int -sujo: bool -marchas: int -amortecedor: bool
+lavar() +pedalar() +frear()

Como usar **generalização (herança)** para reduzir a redundância?

Herança - Benefícios e Características

Exemplo simples



Herança – Benefícios e Características

Benefícios

- Evitar replicação de código
- Aumentar reusabilidade
- Facilitar a extensibilidade



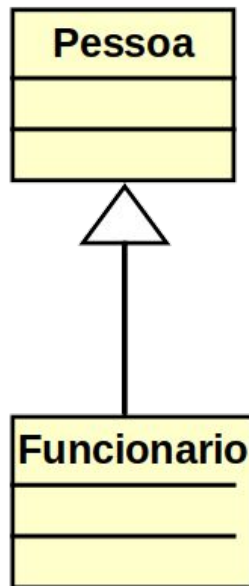
Herança – Benefícios e Características

Características

Qual é o sentido da dependência?

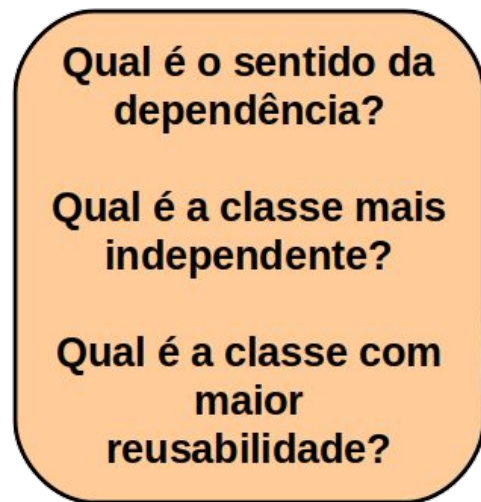
Qual é a classe mais independente?

Qual é a classe com maior reusabilidade?

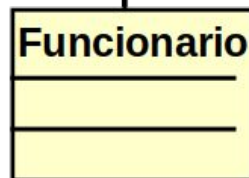
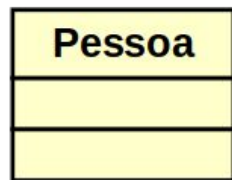


Herança – Benefícios e Características

Características



Maior reusabilidade, Mais genérico
Maior abstração



Menor reusabilidade, Mais específico,
Menor abstração



Sentido da dependência

Herança – Benefícios e Características

Características

- Subclasse herda atributos e métodos
 - SUBCLASSE NÃO TEM ACESSO AOS PRIVADOS
- Subclasse é um tipo da superclasse - tipagem estática
- É possível **criar** novos atributos e métodos (especializar)
 - E também **redefinir**: sobreposição e sobrecarga
 - Fundamentais para o *polimorfismo*!

Herança – Benefícios e Características

Características

- **Sobreposição (*override*)**

redefinição de métodos
nas subclasses

```
class Veiculo:
    # ...

    def lavar(self):
        total = self.dono.dinheiro
        if total >= 30:
            self.dono.dinheiro = total - 30
            self.__sujo = False

    def frear(self):
        pass

    # ...

class Carro(Veiculo):
    # ...

    def frear(self):
        self.velocidade -= 10
        if self.velocidade < 0:
            self.velocidade = 0

class Bicicleta(Veiculo):
    # ...

    def lavar(self):
        total = self.dono.dinheiro
        if total >= 10:
            self.dono.dinheiro = total - 10
            self.sujo = False

    def frear(self):
        self.velocidade -= 1
        if self.velocidade < 0:
            self.velocidade = 0
```

Herança – Benefícios e Características

Características

- **Sobrecarga (*overload*)** – definição de múltiplas assinaturas para um mesmo método (não existe em Python)

Overriding

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }
}

class Hound extends Dog{
    public void sniff(){
        System.out.println("sniff ");
    }

    public void bark(){
        System.out.println("bowl");
    }
}
```

Same Method Name,
Same parameter

Overloading

```
class Dog{
    public void bark(){
        System.out.println("woof ");
    }

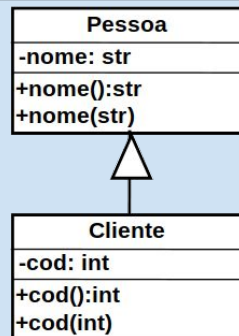
    //overloading method
    public void bark(int num){
        for(int i=0; i<num; i++)
            System.out.println("woof ");
    }
}
```

Same Method Name,
Different Parameter

Tipos de herança

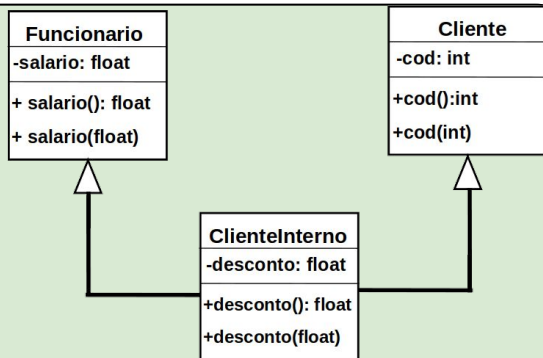
- **Simples**

subclasse herda
de uma superclasse



- **Múltipla**

herda de duas ou mais
superclasses



Herança: Agenda

1. Intuição e conceito
2. Benefícios e características
- 3. Classes Abstratas**



Você já viu
um **veículo**?



Você já viu
uma **forma**
geométrica?



Você já viu
um **animal**?



Classes Abstratas

- Abstrato demais (incompleto) para ter objetos
 - Métodos podem ser abstratos
- Permite definir uma base para hierarquia de classes
- Pode ter atributos e métodos definidos
- Python não possui palavra chave - usa ABC e decorators

Classes Abstratas

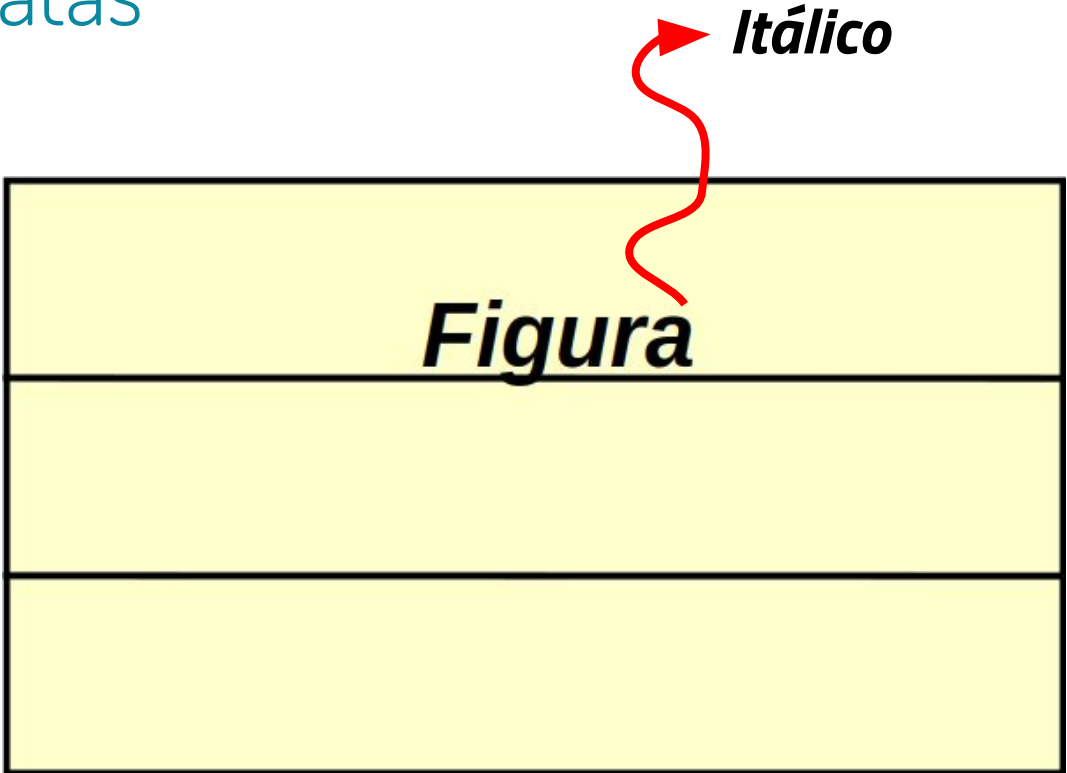
{abstract}
Figura



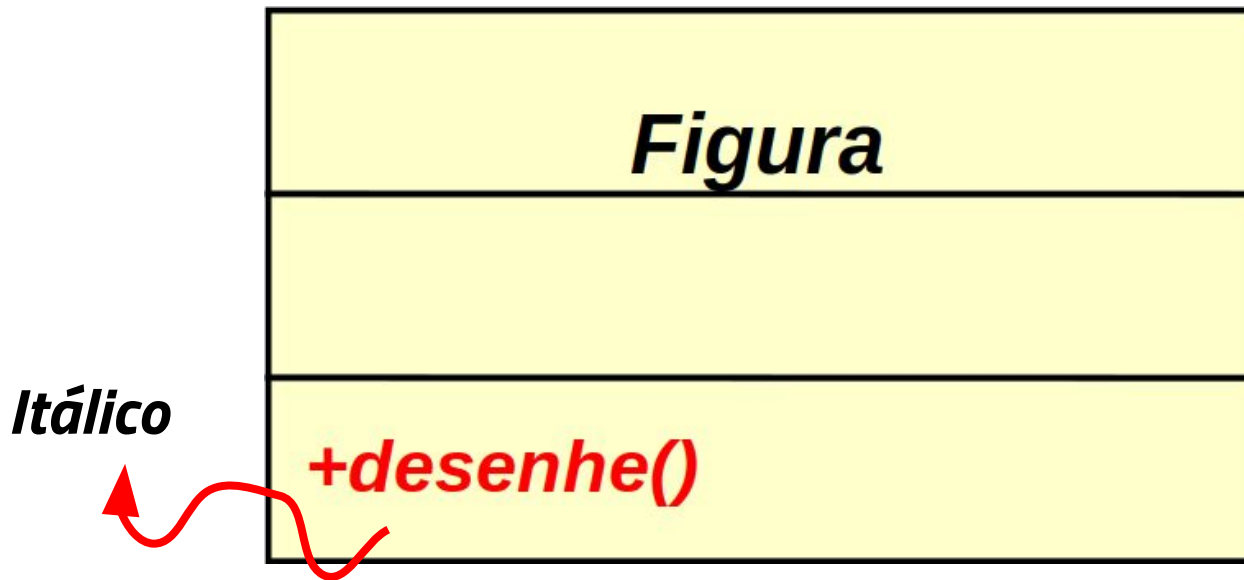
```
classDiagram
    class Figura {
        <<abstract>>
    }
    class Figura {
        <<abstract>>
    }
    class Figura {
        <<abstract>>
    }
```

The diagram shows a UML class box for an abstract class named 'Figura'. The box is yellow and has a black border. The top section contains the text '{abstract}' and 'Figura'. Below this section are two empty sections, separated by horizontal lines, representing the area for attributes and methods.

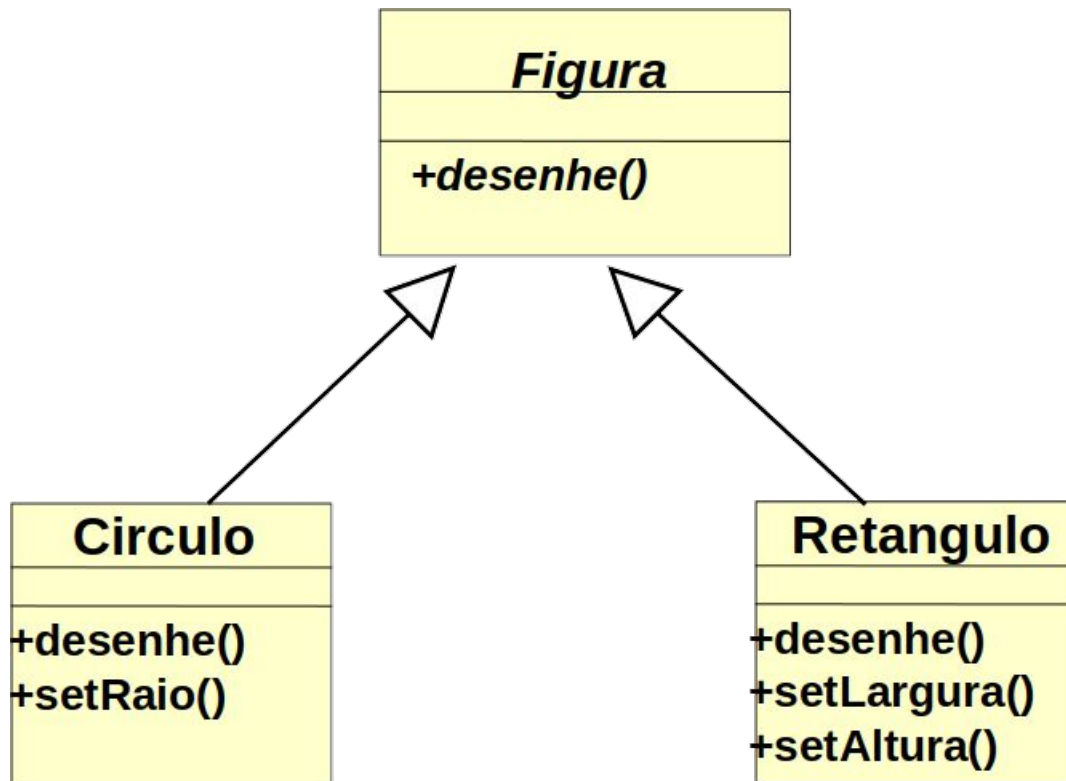
Classes Abstratas



Classes Abstratas



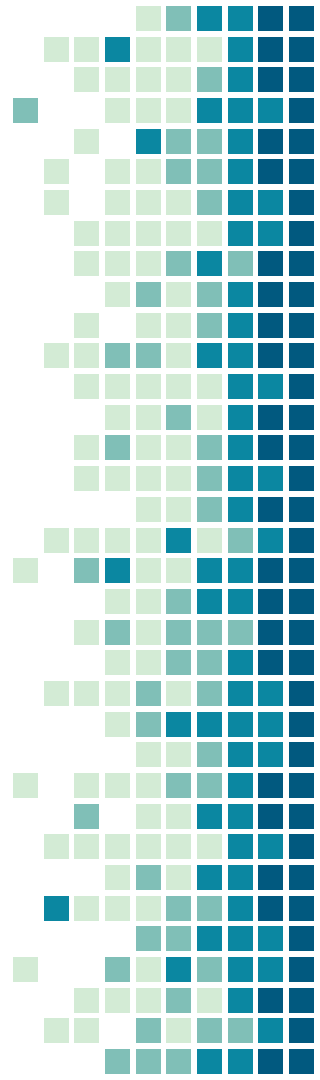
Classes Abstratas



Classes Abstratas – Python

```
from abc import ABC, abstractmethod
```

```
class Figura(ABC):  
    @abstractmethod  
    def __init__(self):  
        pass  
  
    @abstractmethod  
    def desenhe(self):  
        pass
```



Classes Abstratas - Python

```
from abc import ABC, abstractmethod
```

```
class Figura(ABC):  
    @abstractmethod  
    def __init__(self):  
        pass  
  
    @abstractmethod  
    def desenha(self):  
        pass
```

Herança de
ABC indica
que a **Classe**
é **abstrata**

Classes Abstratas - Python

```
from abc import ABC, abstractmethod
```

```
class Figura(ABC):  
    @abstractmethod  
    def __init__(self):  
        pass  
  
    @abstractmethod  
    def desenha(self):  
        pass
```

@abstractmethod
método abstrato
impede de
instanciar a classe

Classes Abstratas - Python

```
class Circulo(Figura):  
    def __init__(self, raio: int):  
        super().__init__()  
        self.__raio = raio  
  
    @property  
    def raio(self):  
        return self.__raio  
  
    @raio.setter  
    def raio(self, raio):  
        self.__raio = raio  
  
    def desenha(self):  
        return "circulo de raio {0:d}".format(self.__raio)
```

Implementação obrigatória dos métodos que eram abstratos na classe-pai

Classes Abstratas - Python

```
class Retangulo(Figura):
    def __init__(self, lado1=0, lado2=0):
        super().__init__()
        self.__lado1 = lado1
        self.__lado2 = lado2

    @property
    def lado1(self):
        return self.__lado1

    @lado1.setter
    def lado1(self, lado1):
        self.__lado1 = lado1

    ...

    def desenhe(self):
        return "retangulo com lados {0:d} e {1:d}".\
            format(self.__lado1, self.__lado2)
```

Implementação
obrigatória dos
métodos que
eram abstratos na
classe-pai

Classes Abstratas - Python

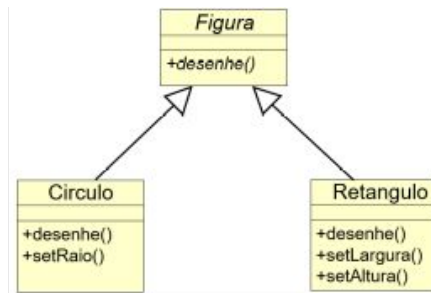
```
figuras = []

retangulo = Retangulo(1, 2)

circulo = Circulo(2)

figuras.append(retangulo)
figuras.append(circulo)

for figura in figuras:
    print(figura.desenhe())
```



Classes Abstratas - Python

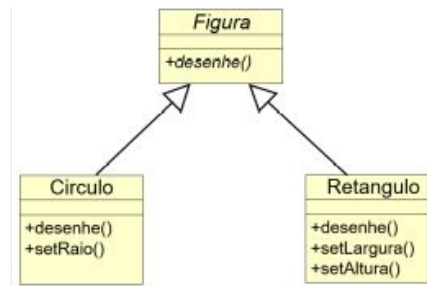
```
figuras = []

retangulo = Retangulo(1, 2)

circulo = Circulo(2)

figuras.append(retangulo)
figuras.append(circulo)

for figura in figuras:
    print(figura.desenhe())
```



Polimorfismo:
garantido pela
herança de
Figura

Classes Abstratas – Java

```
abstract class Shape {  
    String name;  
    abstract public double area();  
    public String name() {  
        return name;  
    }  
  
    //other stuff  
}  
  
class Square extends Shape {  
    double length;  
    public Square(double length) {  
        this.length = length;  
    }  
    public double area() {  
        return length*length;  
    }  
}
```

Output

The area of square s is 156.25

Referências

DATHAN, B.; RAMNATH, S. **Object-Oriented Analysis, Design and Implementation**. Cham: Springer, 2015

SEIDL, M. et al. **UML@ Classroom: An Introduction to Object-Oriented Modeling**. Cham: Springer, 2015.

