

FIA/P GRADUAÇÃO

DOMAIN DRIVEN DESIGN

Prof. Me. Thiago T. I. Yamamoto

#09 - HERANÇA

TRAJETÓRIA



- ✓ Orientação a Objetos
- ✓ Introdução ao Java
- ✓ IDE e Tipos de Dados
- ✓ Classes, atributos e métodos
- ✓ Encapsulamento
- ✓ Construtores
- ✓ Conversões e Tomada de decisões
- ✓ Manipulação de Strings
- ✓ Herança

#09 - AGENDA

- Conceito de Herança
- Herança no Java
- Palavra reservada super

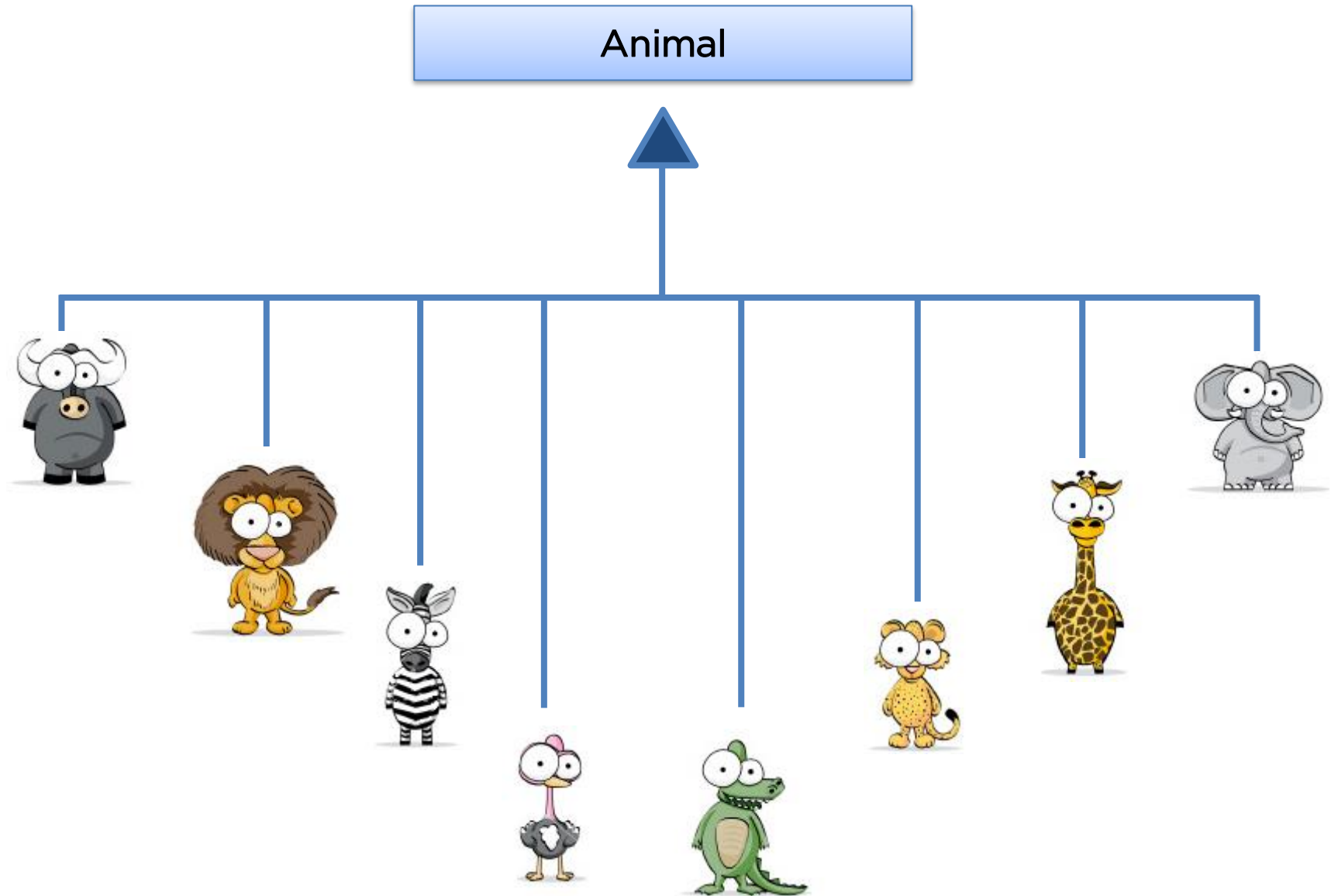


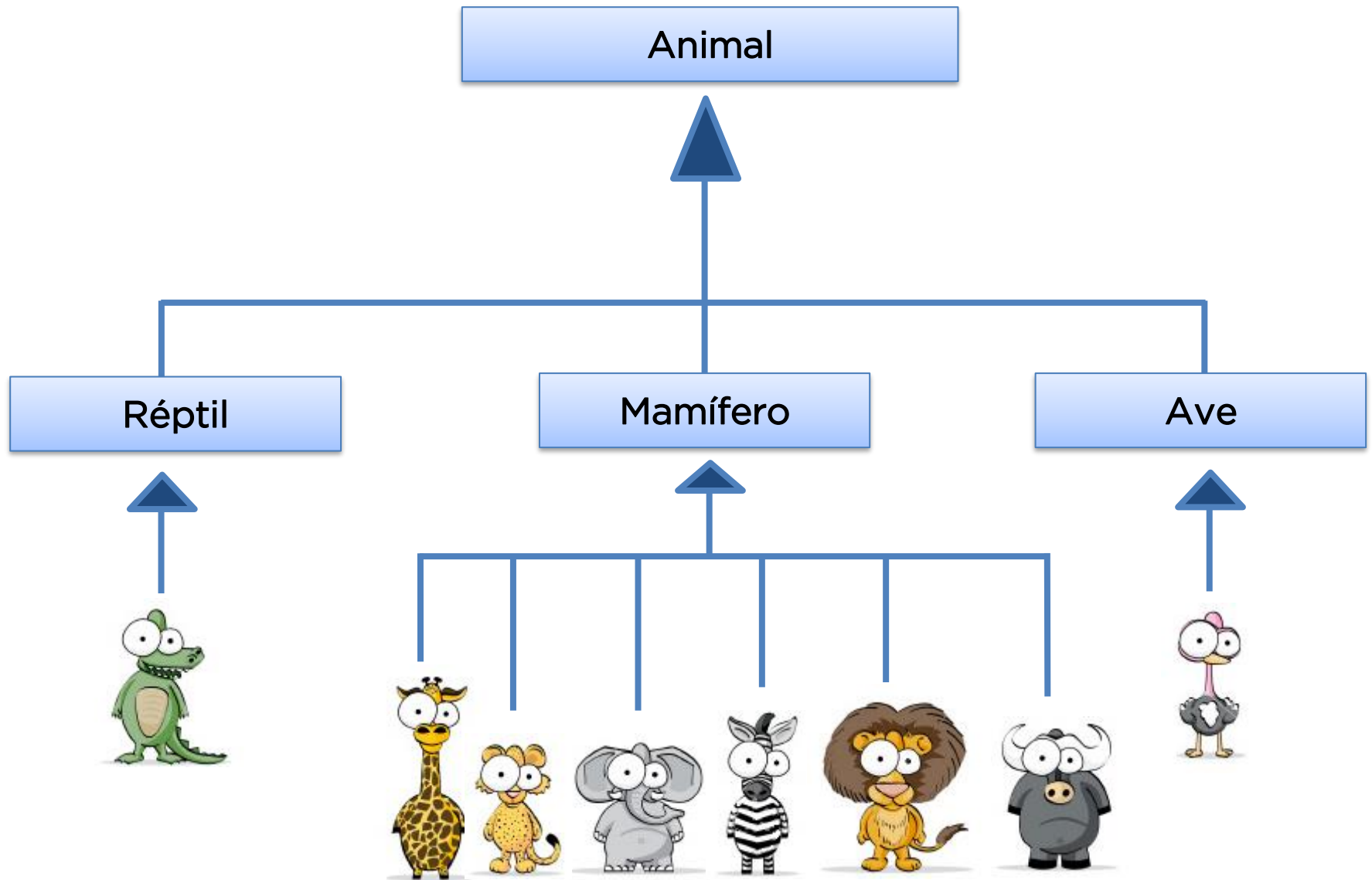


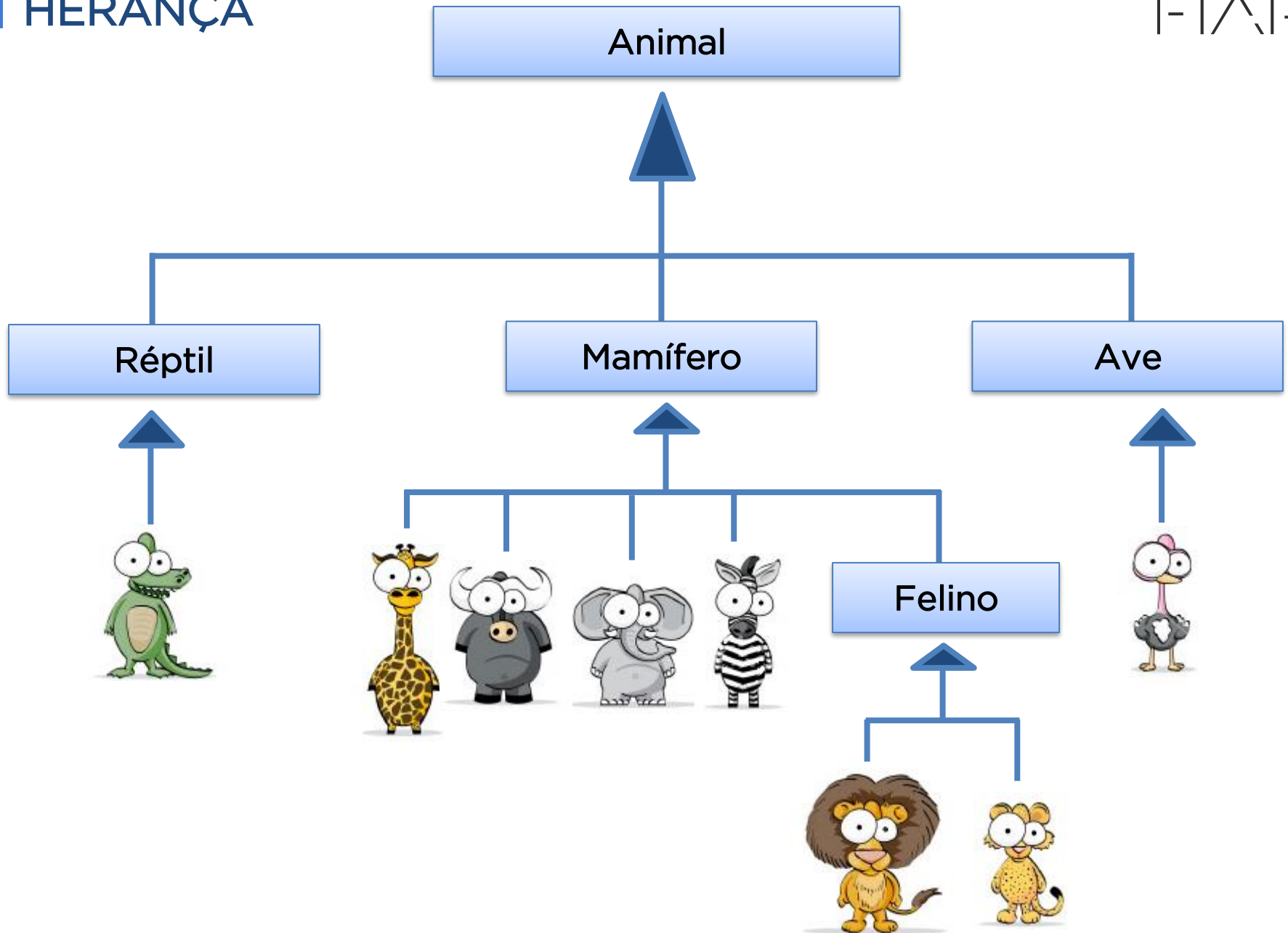
HERANÇA

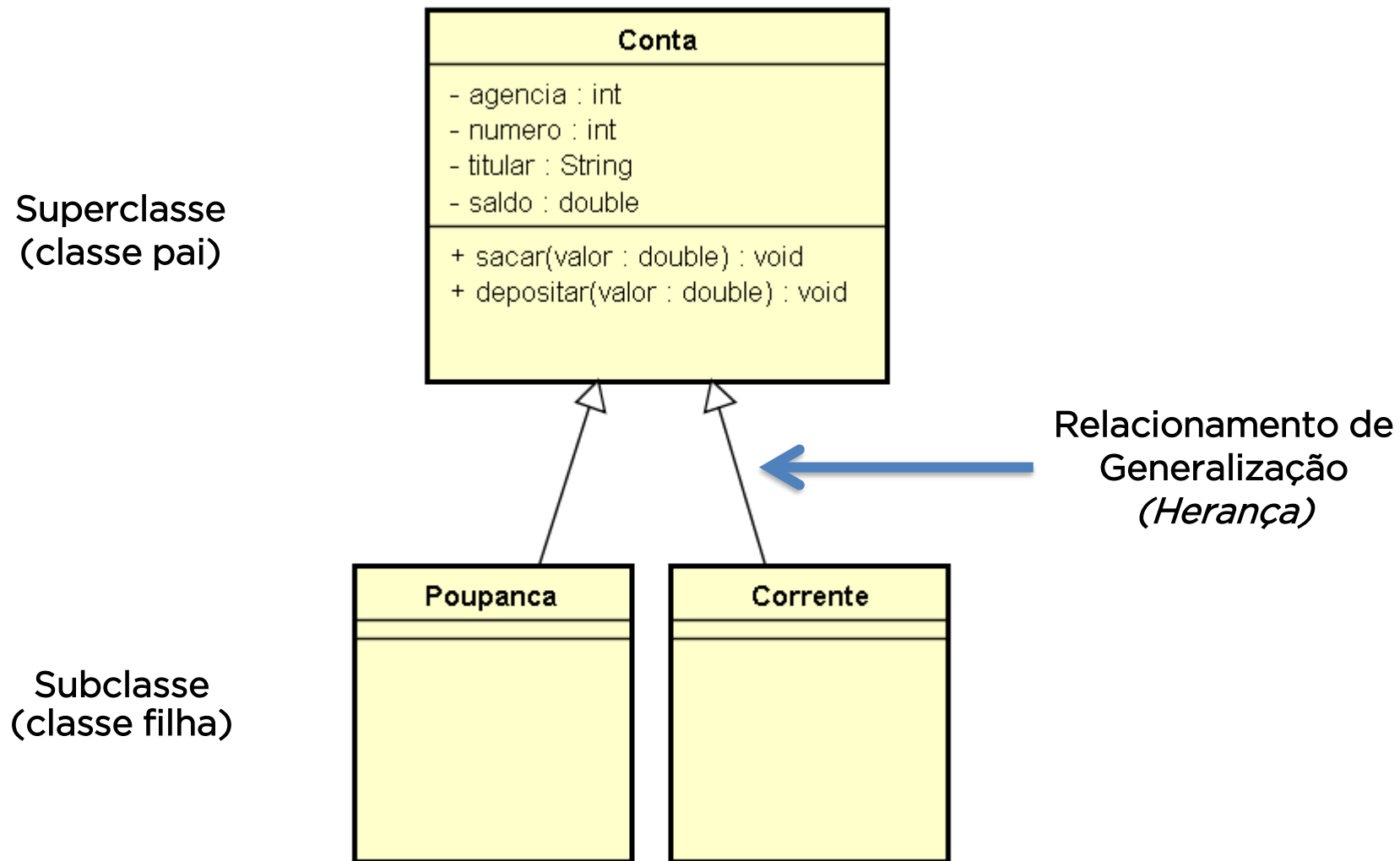
- **Herança** é um dos mecanismos **fundamentais** para as **linguagens** que suportam o **paradigma OO**;
- Este mecanismo possibilita a **criação de novas classes a partir de uma já existente**;
- A **herança** é utilizada como forma de **reutilizar os atributos e métodos** de classes já definidas, permitindo assim derivar uma **nova classe mais especializada** a partir de outra classe mais genérica existente;
- **Aplicar herança** sempre envolve basicamente dois elementos: uma **superclasse** (classe pai) e uma **subclasse** (classe filha);
- **Superclasse** é também conhecida como classe ancestral ou **classe pai**. Apresenta as características genéricas de um conjunto de objetos;
- **Subclasse** é também conhecida como classe descendente ou **classe filha**. Elas estende a superclasse para incluir suas características

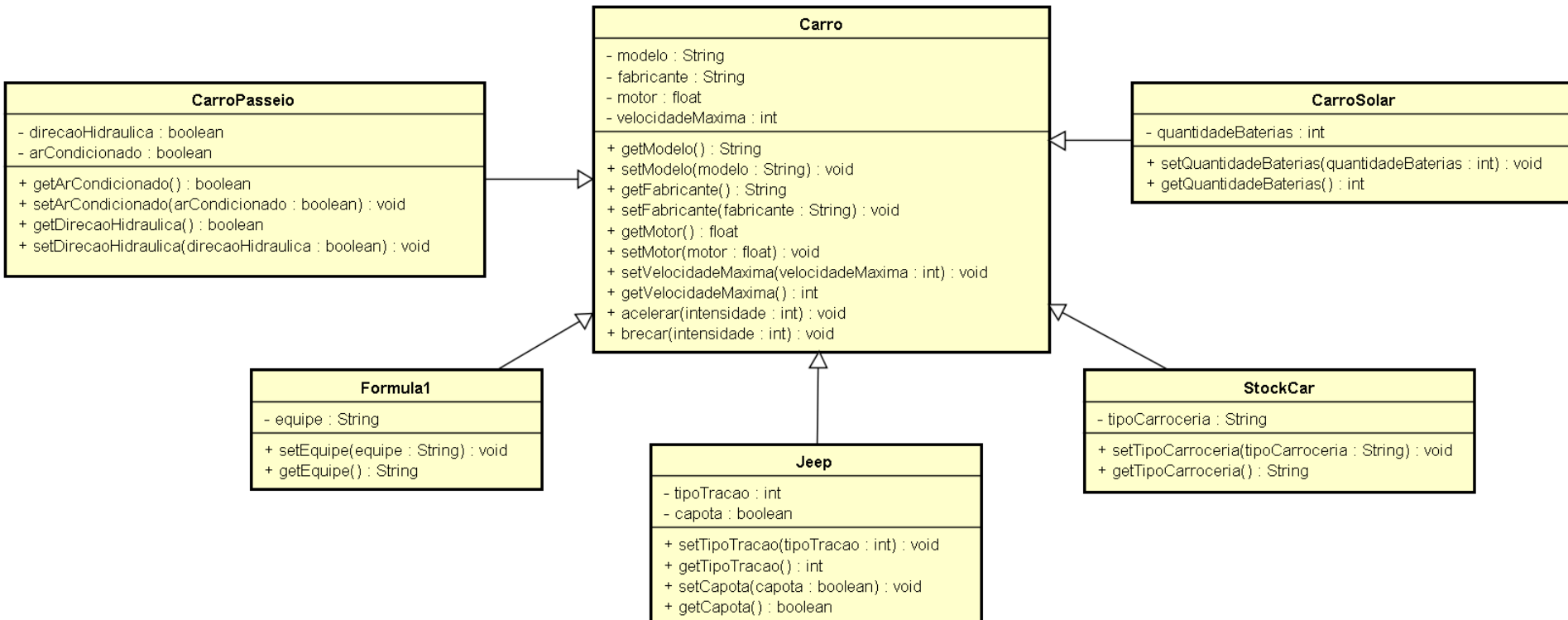
- A subclasse:
 - Herda os atributos;
 - Permite **adicionar novos atributos** (que será visível somente na subclasse);
 - Em relação aos **métodos**, a subclasse poderá **utilizá-los/herdá-los** (superclasse), bem como **criar novos** métodos e alterá-los;
 - Métodos **construtores não são herdados** (porém podemos chamá-los dentro do construtor da subclasse).



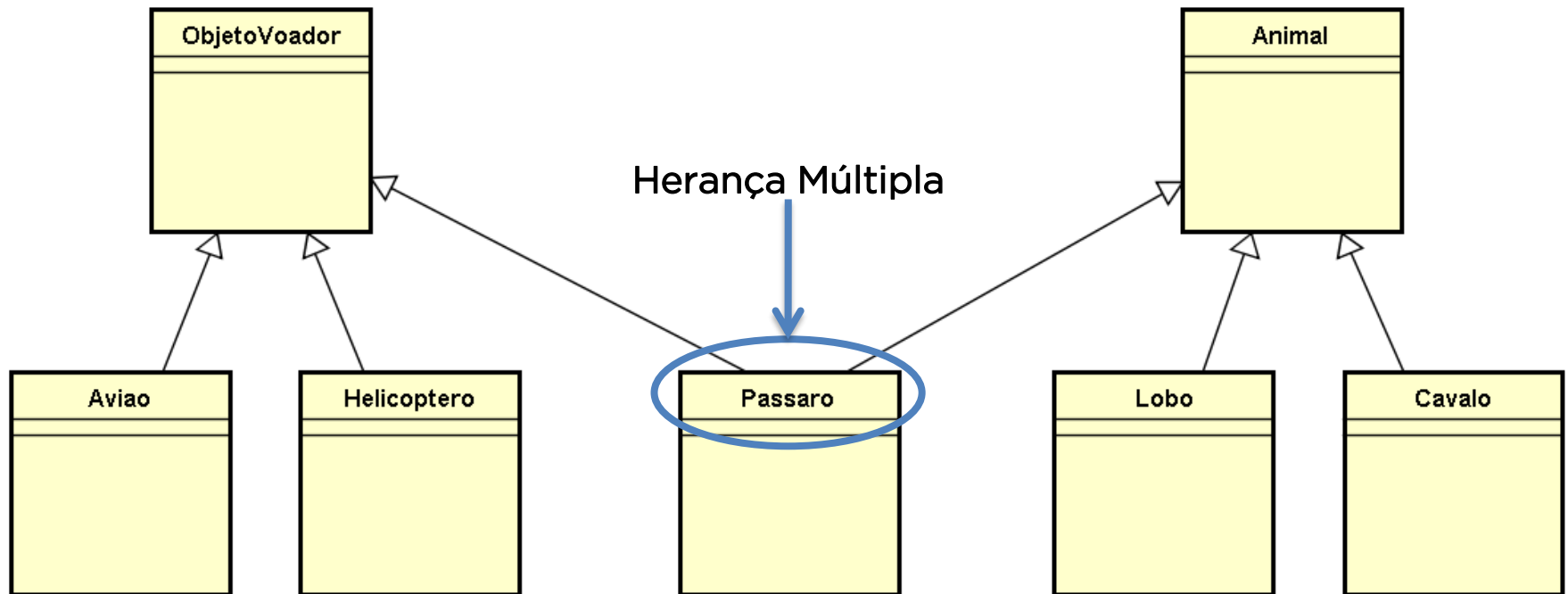








- Herança múltipla significa que uma classe pode herdar de várias outras classes;
- O Java não permite a herança múltipla;

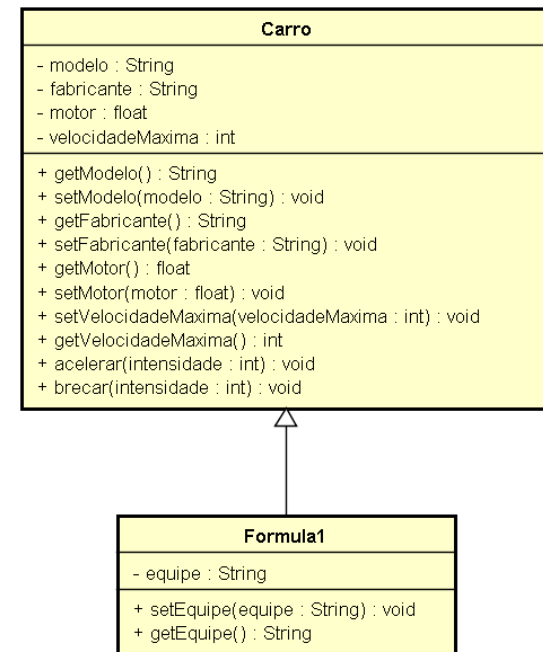


- Toda **classe criada no Java** é estendida a partir da classe **Object**;
- A palavra-chave **extends** é utilizada na declaração de uma classe para especificar quem é sua superclasse;
- Caso a palavra-chave seja omitida, a classe **Object** será assumida como a **superclasse** da nova classe;
- Sintaxe:

```
[public] [abstract | final] class <subclasse> extends <superclasse>
{
}
```

- No exemplo abaixo, a classe **Formula1** estende a classe **Carro**:

```
public class Formula1 extends Carro {  
  
    private String equipe;  
  
    public void setEquipe(String equipe) {  
        this.equipe = equipe;  
    }  
  
    public String getEquipe() {  
        return equipe;  
    }  
  
}
```



- O construtor da classe estendida lida apenas com as variáveis definidas na classe, e o construtor da superclasse lida com as variáveis que são herdadas;
- Um construtor da classe estendida pode invocar diretamente um dos construtores da superclasse;
- Construtores não são herdados e precisam ser implementados na subclasse;
- Construtores da subclasse “sempre” utilizam algum construtor da superclasse;
- A referência `super(<parâmetros>)` é utilizada para invocar o construtor da superclasse;


```
public class Conta {  
    public Conta(String numero, double saldo, Cliente cliente){  
    }  
}  
  
public class Poupanca extends Conta{  
  
    public Poupanca(String numero, double saldo, Cliente cliente){  
        super(numero, saldo, cliente);  
        System.out.println("Classe: Poupanca - Construtor:  
        Poupanca(String numero, double saldo, Cliente cliente)");  
    }  
}
```

- **super** chama o **construtor** da superclasse
se **super** não for **chamado**, o compilador acrescenta uma chamada ao construtor *default*: **super()**;
se não existir um construtor *default* na superclasse, haverá um **erro de compilação**;

- Permite que **atributos** e **métodos** da **superclasse** sejam **referenciados** pelos métodos da **subclasse**;
 - Sintaxe:
 - `super.<atributo>;`
 - `super.<método>;`
- Uso sucessivo de `super` **não** é permitido;
 - `super.super.nomeMetodo;`
- Caso queira referir-se a um **construtor** da **superclasse**, a sintaxe é diferente. Deve ser utilizada apenas a **referência seguida de um par de parênteses**;
 - `super();`

```
public class ClasseA{
    protected int atributo3 = 3;
}

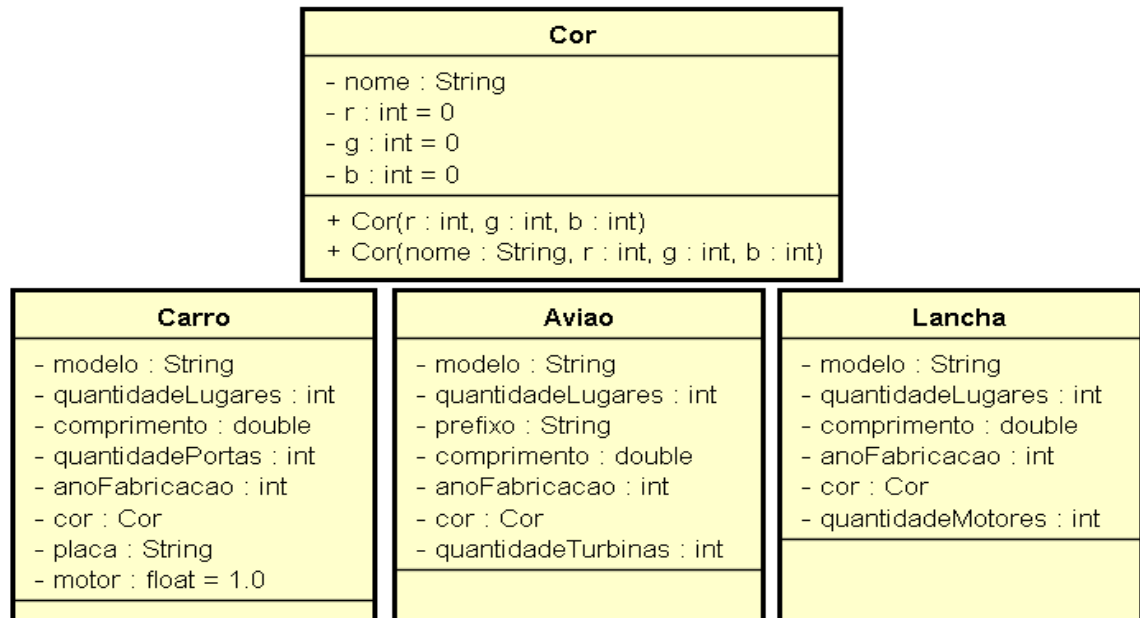
public class ClasseB extends ClasseA{
    public void testeSuper(){
        super.atributo3 = 33;
        System.out.println("super.atributo3=" + this.atributo3);
    }
}

public class TesteSuper {
    public static void main(String args[]){
        ClasseB cla = new ClasseB();
        cla.testeSuper();
    }
}
```

PRÁTICA

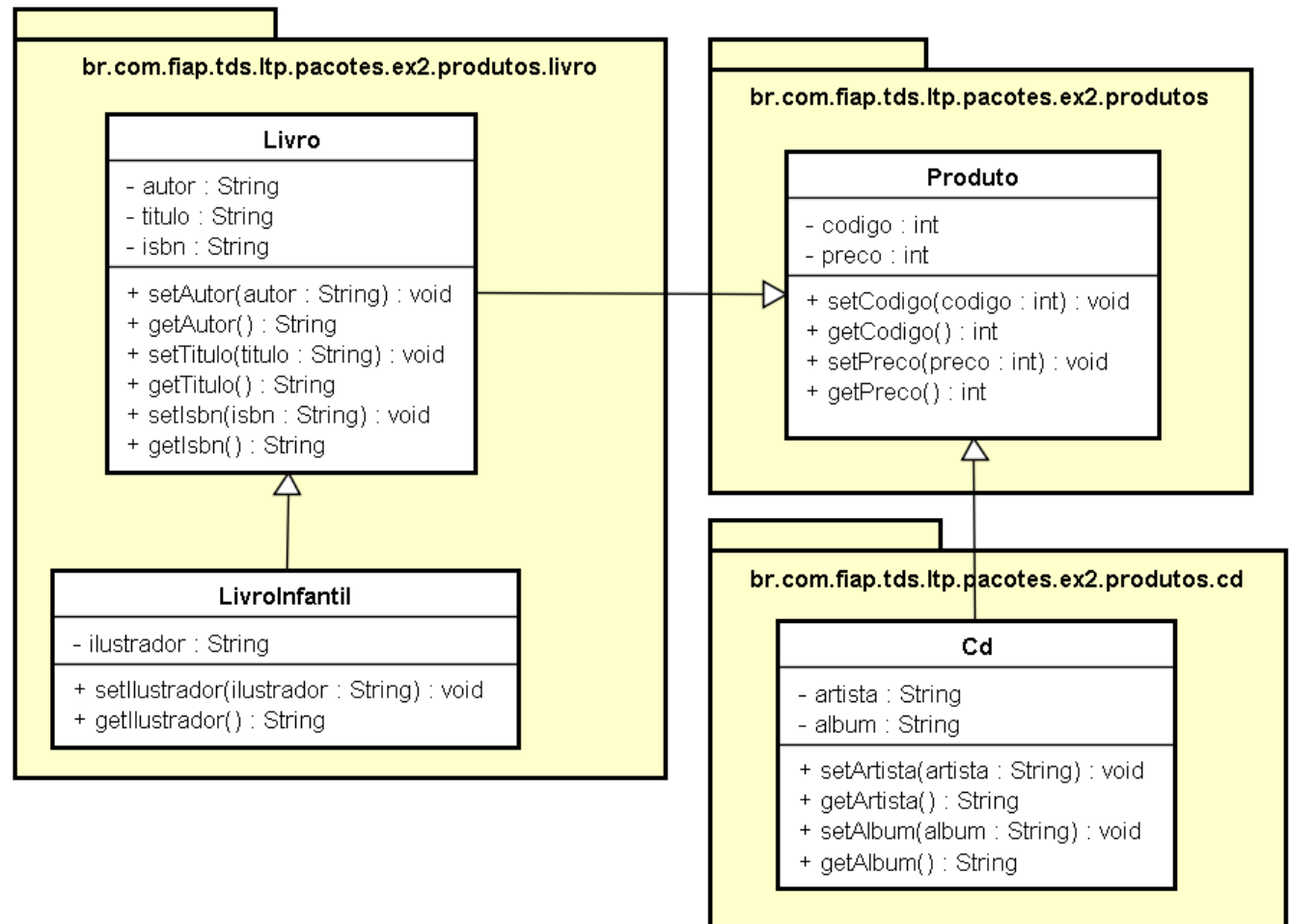


- Utilizando seus conhecimentos sobre herança, analise as classes exibidas abaixo e em seguida implemente uma nova hierarquia de classes de forma que o conceito de herança seja aplicado e evite as duplicações expostas.



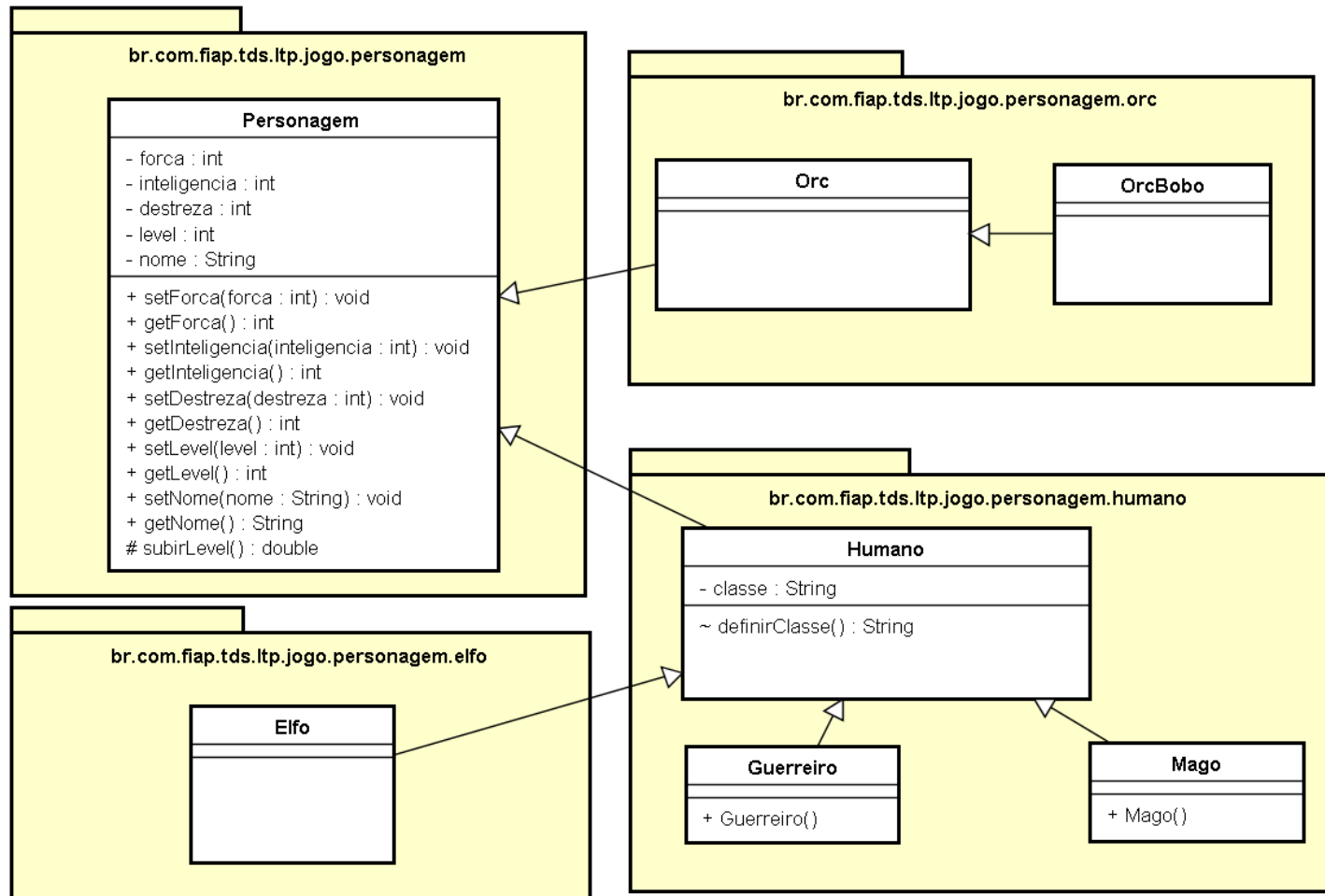
PRÁTICA

- Implemente o diagrama de classes abaixo:



PRÁTICA

- Implemente o diagrama de classes abaixo:



Copyright © 2020 - 2022 Prof. Thiago T. I. Yamamoto

Todos direitos reservados. Reprodução ou divulgação total ou parcial deste documento é expressamente proibido sem o consentimento formal, por escrito, do Professor (autor).

*"A lógica pode levar de um ponto A a um ponto B.
A imaginação pode levar a qualquer lugar"*
Albert Einstein