Herança

Herança

```
class Carro {
    String matricula;
    int cilindrada;
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota {
    String matricula;
    int cilindrada;
}
```

Certas classes têm coisas em comum:

variáveis

métodos

Normalmente isso reflecte um super-tipo no qual ambas as classes se inserem

Herança

```
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro {
    String matricula;
    int cilindrada;
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota {
    String matricula;
    int cilindrada;
}
```

Carro e Mota são ambos Veículos!

Se:

- todos os veículos têm matricula e cilindrada
- carro e mota são veículos

Então:

- Carro e mota têm matrícula e cilindrada

Herança

```
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {
}
```

Carro e Mota herdam (extends) de Veiculo

Todas as características (variáveis e métodos) de Veiculo passam automaticamente para os "herdeiros"

Herança

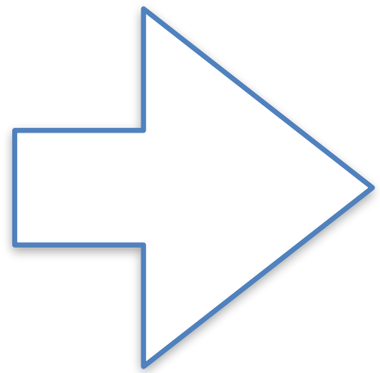
```java
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {
}

public class App {
    public static void main(String[] args) {

        Mota mota = new Mota();
        mota.matricula = "34-GH-21";
        mota.cilindrada = 500;
        mota.dimensaoDaMala = 100;

        Carro carro = new Carro();
        carro.matricula = "45-HJ-21";
        carro.cilindrada = 1300;
        carro.dimensaoDaMala = 100;
    }
}
```

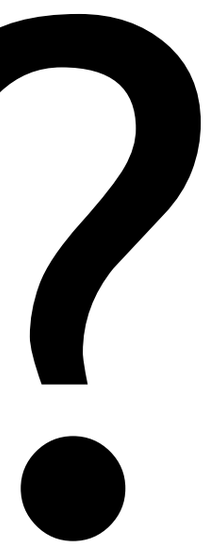
Errado! Só Carro é que tem esta variável!

Herdado de Veiculo

UML

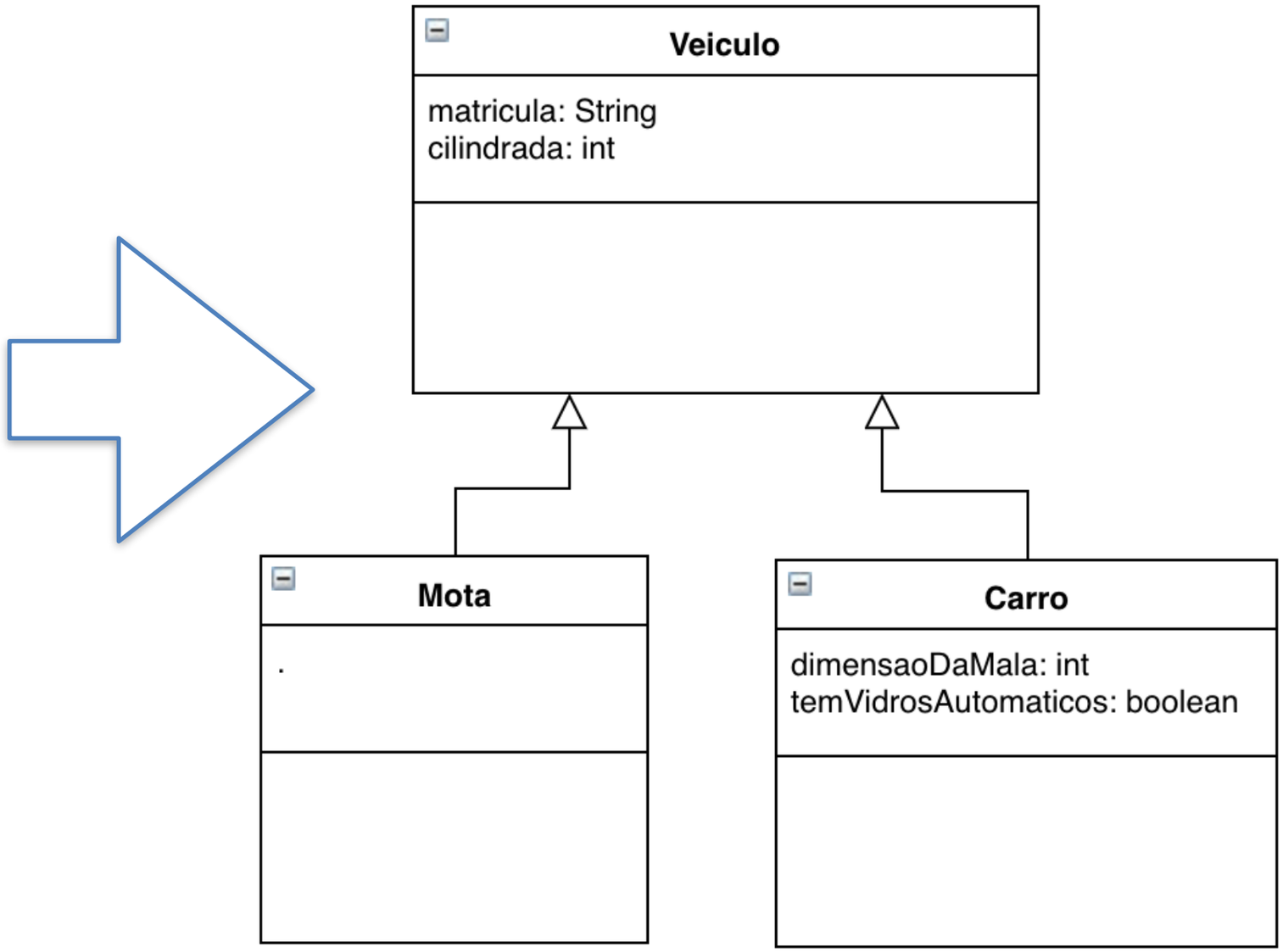
```
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {
}
```



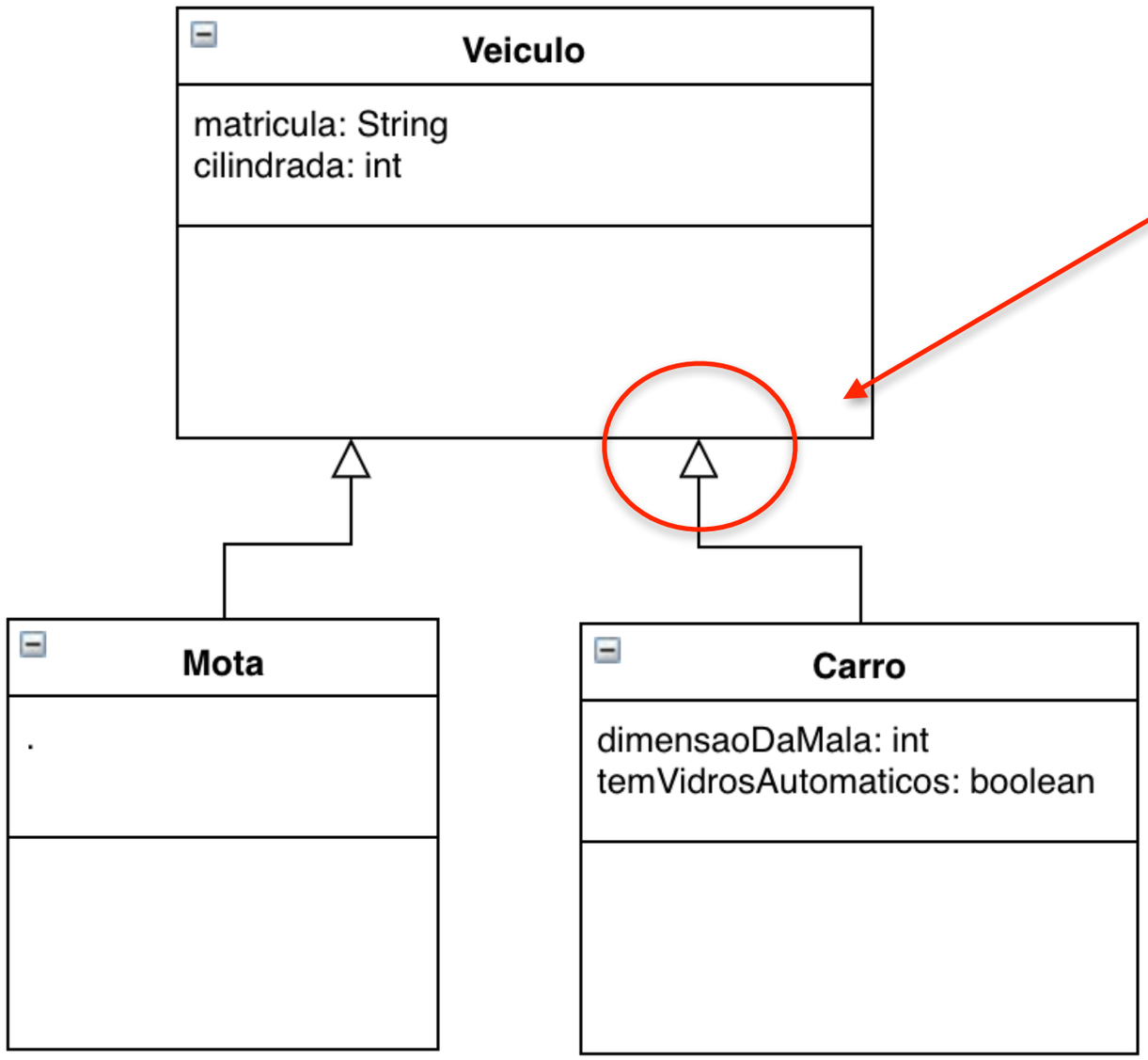


UML

```
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {
}
```



Herança



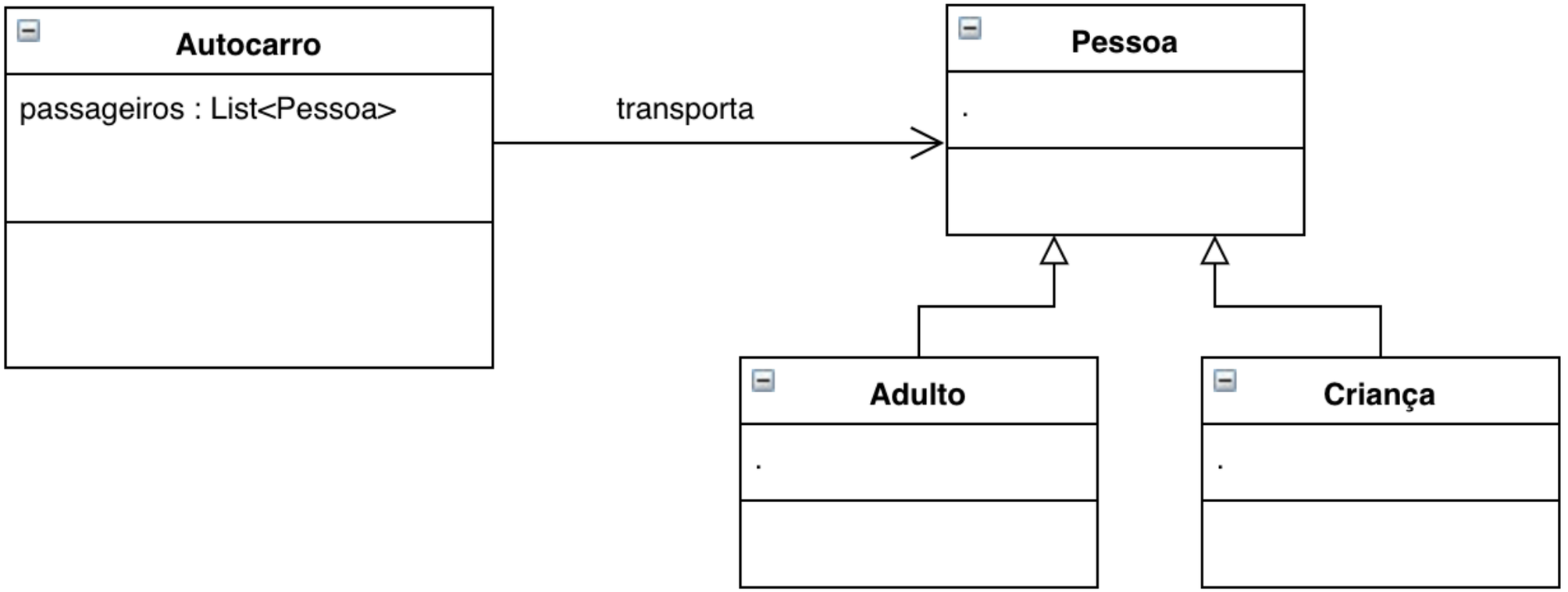
Atenção à seta de herança. É diferente da seta de relação

Carro **é um** Veiculo.
Mota **é um** Veiculo.

Veiculo **pode ser** uma Mota ou um Carro.

Nota: Construtores e métodos omitidos para simplificação

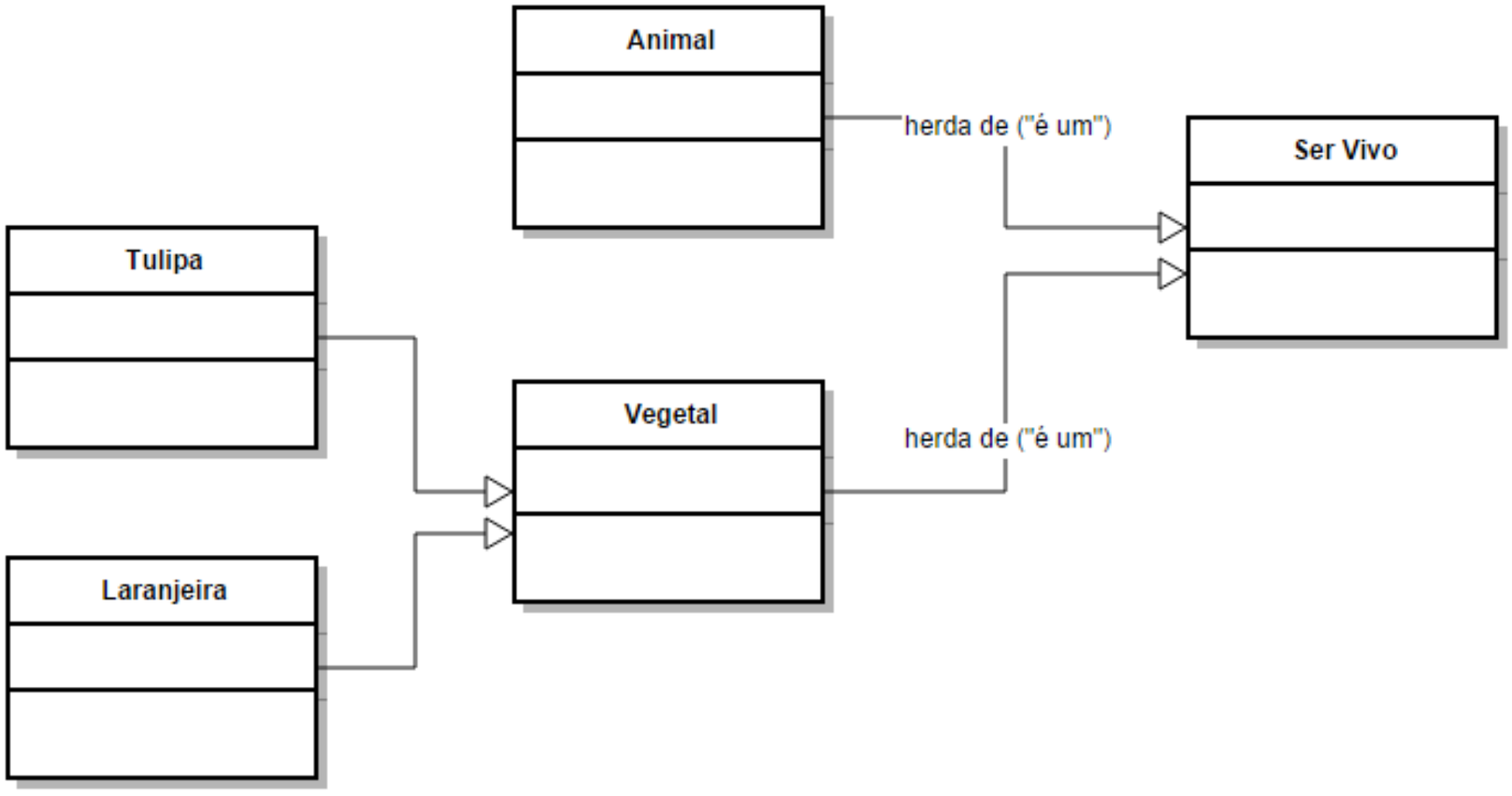
Relação vs Herança



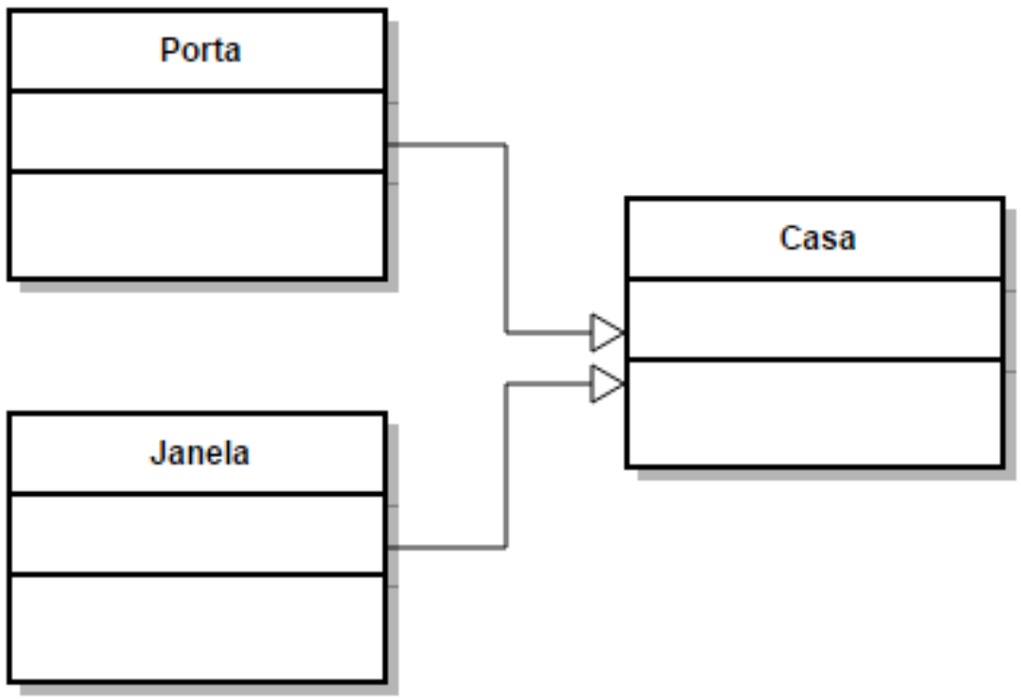
relação entre duas entidades definida pelo seu nome (ex: tem, transporta, pára em, …) - implica criação de uma variável que represente essa relação (ex: passageiros)

relação de herança - pode ser lida como “é um” (ex: um adulto é uma pessoa) - não implica criação de variável

Herança - Seres Vivos



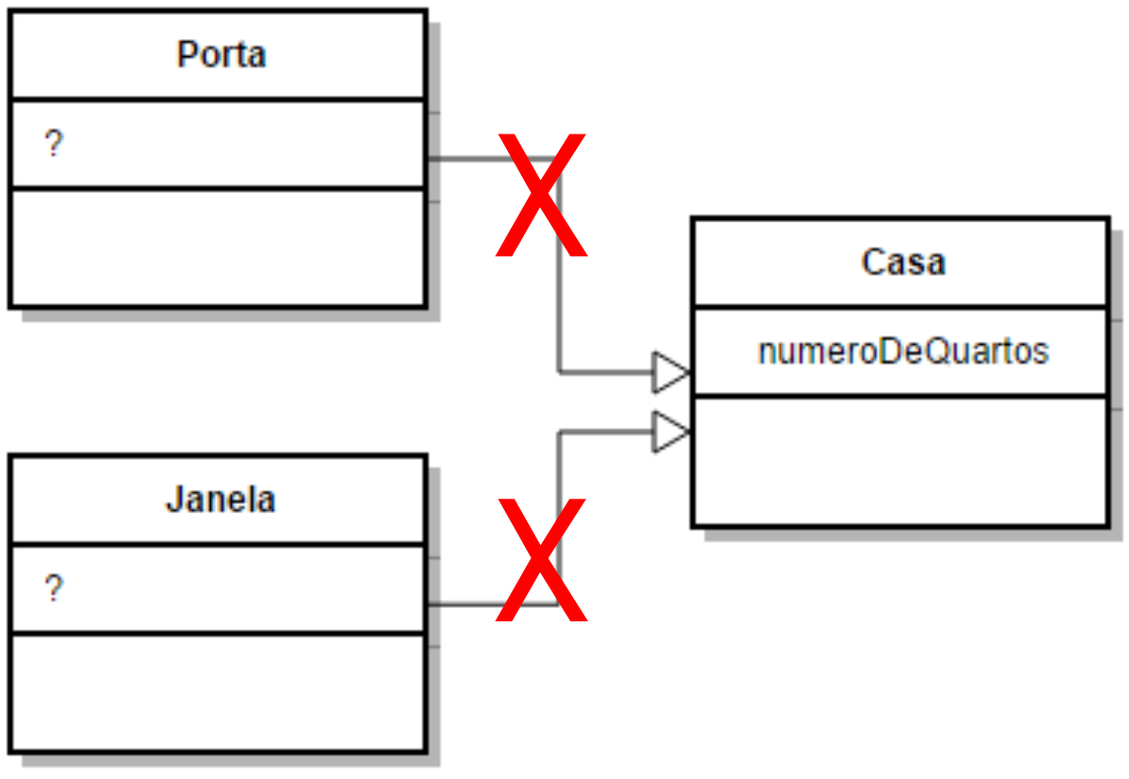
Herança



Isto está correto?

Herança

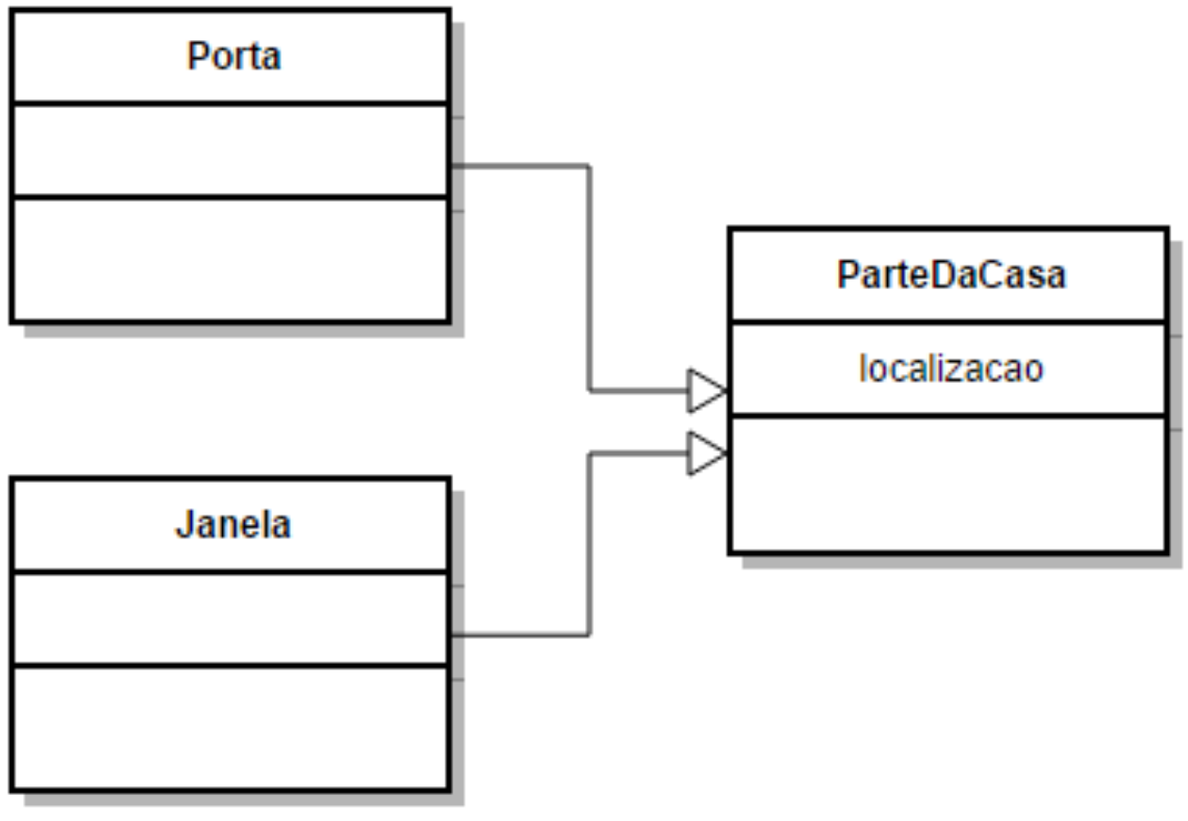
Nota: Tipos das variáveis omitidos para simplificação



Porta **não é uma** Casa. Janela **não é uma** Casa.

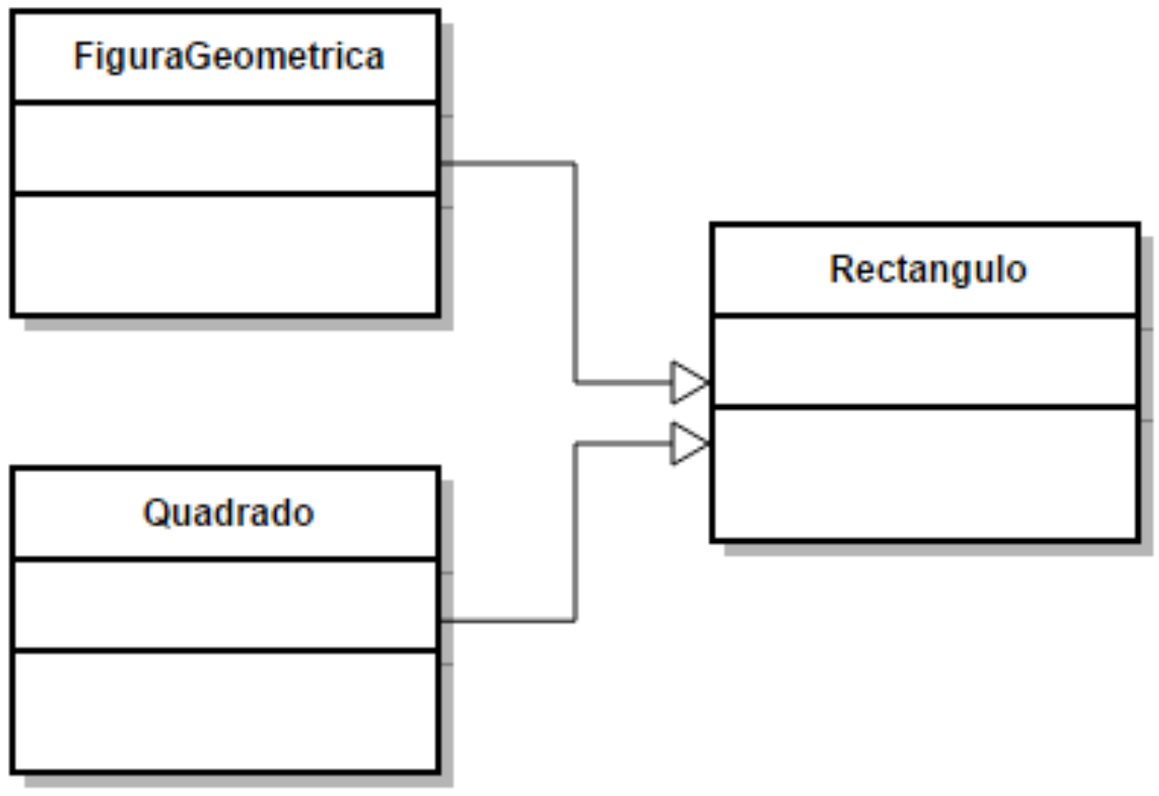
Herança

Nota: Tipos das variáveis omitidos para simplificação



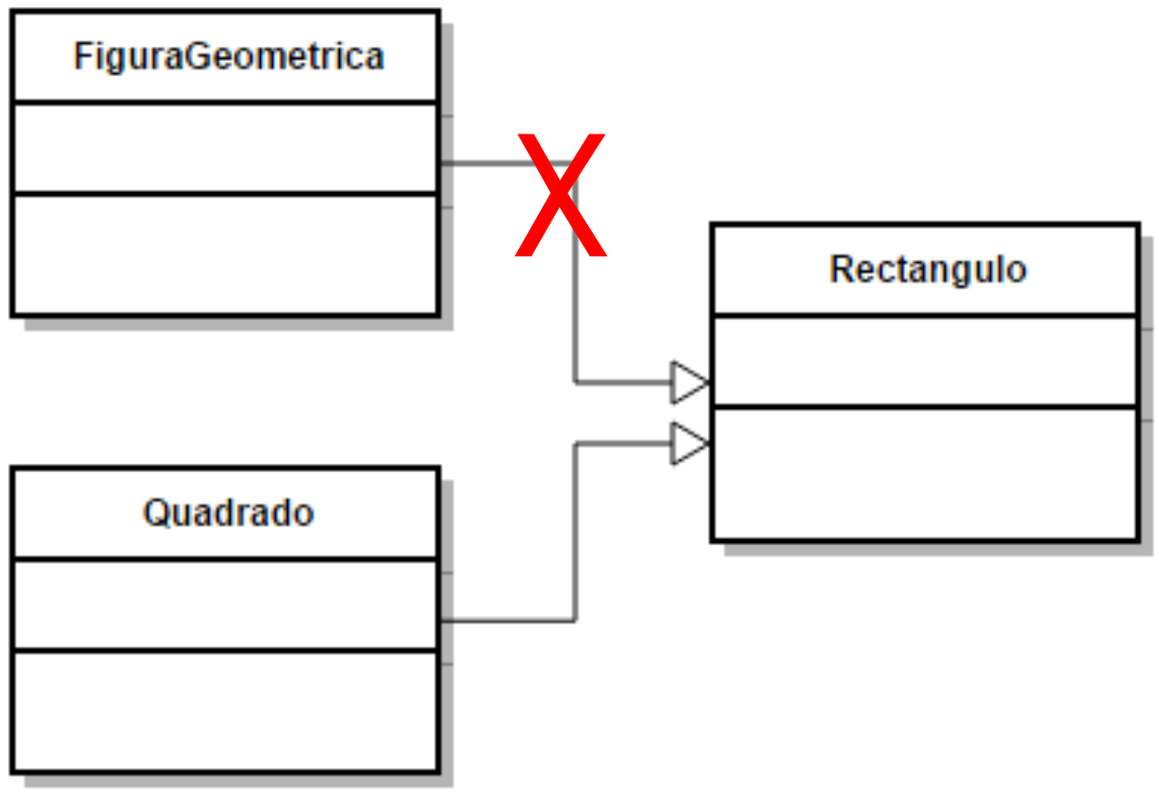
Porta **<u>é uma</u>** Parte da Casa. Janela **<u>é uma</u>** Parte da Casa.

Herança



Isto está correto?

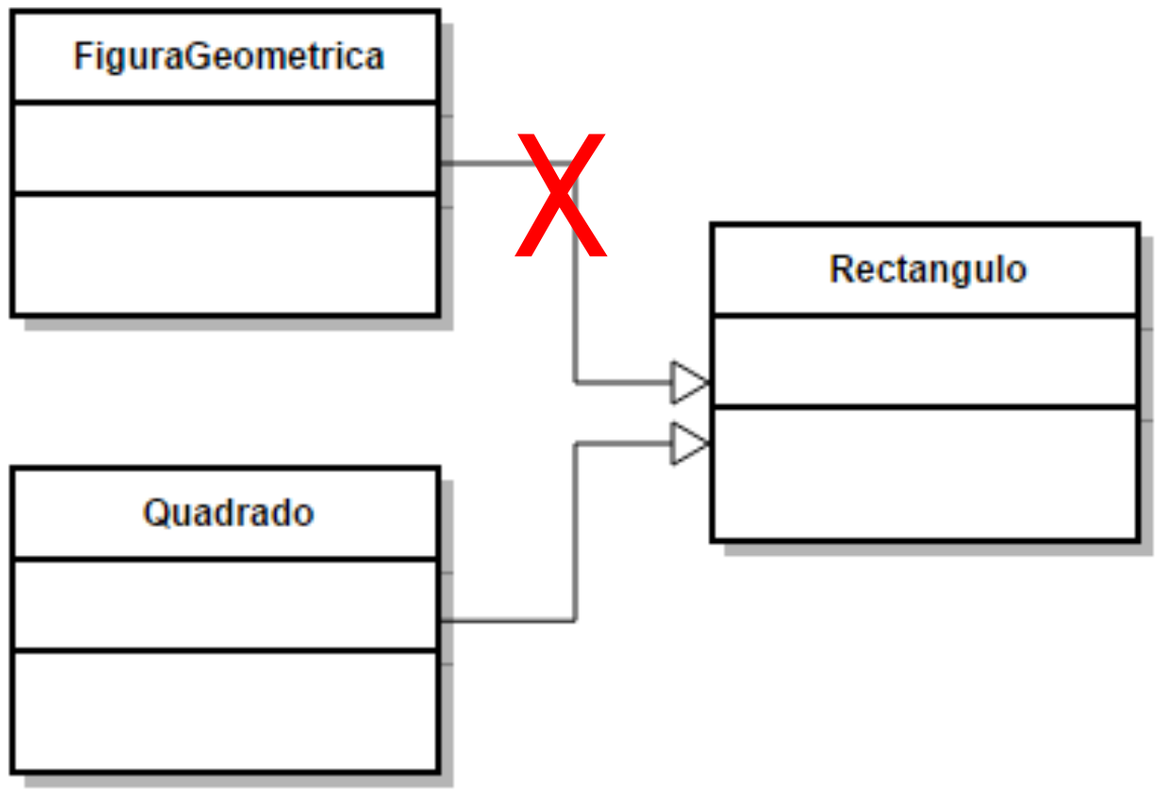
Herança



FiguraGeometrica **não é (necessariamente) um** Rectangulo.

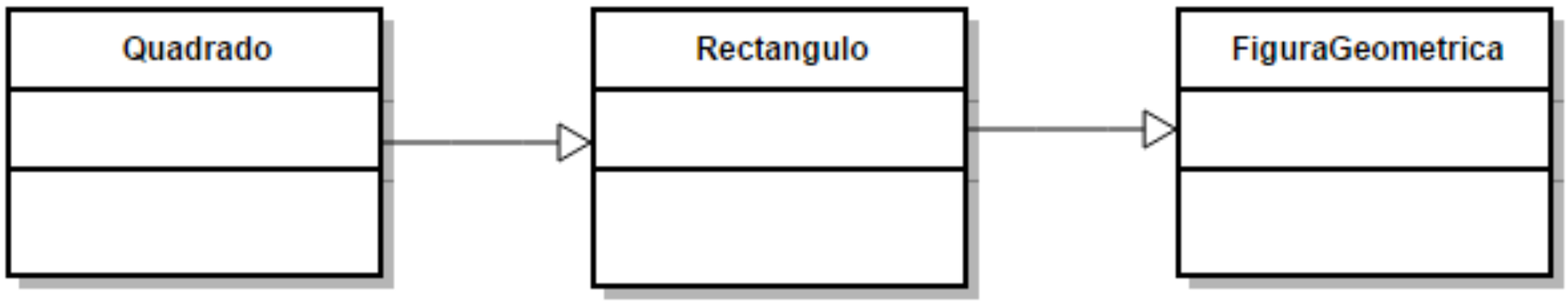
FiguraGeometrica **pode ser** um Rectangulo (mas pode ser outras coisas)

Herança



Quadrado **é um** Rectangulo (com arestas iguais)

Herança



Quadrado **<u>é um</u>** Rectangulo (com arestas iguais)
Rectangulo **<u>é uma</u>** FiguraGeometrica
Por transitividade, Quadrado **<u>é uma</u>** FiguraGeometrica

Herança

```
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {
}
```

Carro e Mota herdam (extends) de Veiculo

Todas as características (variáveis e métodos) de Veiculo passam automaticamente para os "herdeiros"

?

Herança

```
class Veiculo {
    String matricula;
    int cilindrada;

    boolean temAltaCilindrada() {
        return (cilindrada > 2000);
    }
}

class Carro extends Veiculo {
}

public class App {
    public static void main(String[] args) {

        Carro carro = new Carro();
        carro.matricula = "45-HJ-21";
        carro.cilindrada = 1300;
        if (carro.temAltaCilindrada()) {
            System.out.println("O carro " + carro.matricula + " tem alta cilindrada!");
        }

    }
}
```

Carro herda de Veiculo:

- Estado: **matricula**, **cilindrada**
- Comportamento: **temAltaCilindrada()**

Vantagens da Herança

- Evita duplicação de código
- Código fica mais fácil de manter

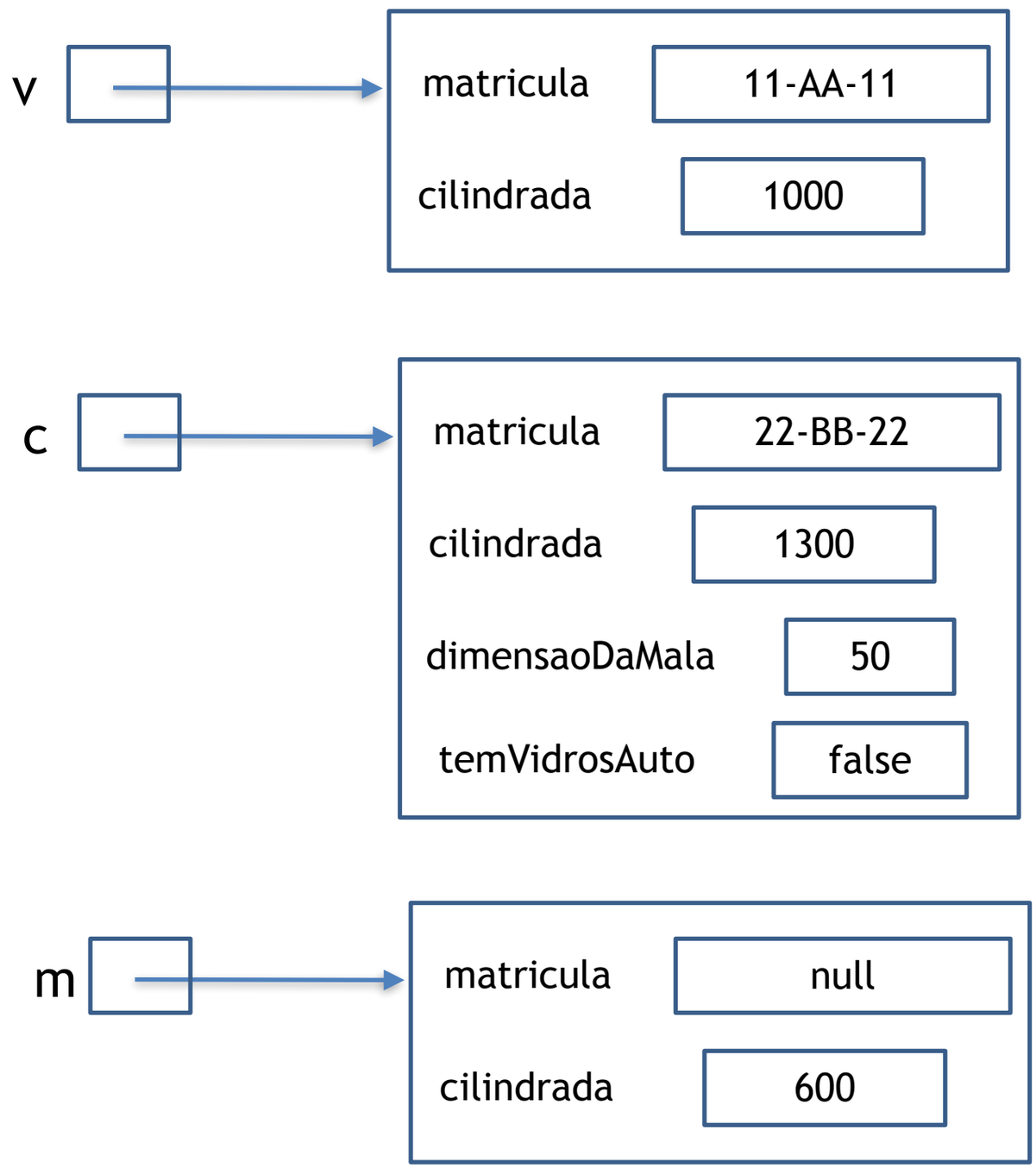
Herança + estrutura de memória

```java
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {

}

public class App {
    public static void main(String[] args) {
        Veiculo v = new Veiculo();
        v.matricula = "11-AA-11";
        v.cilindrada = 1000;

        Carro c = new Carro();
        c.matricula = "22-BB-22";
        c.dimensaoDaMala = 50;
        c.temVidrosAutomaticos = false;
        c.cilindrada = 1300;

        Mota m = new Mota();
        m.cilindrada = 600;
    }
}
```

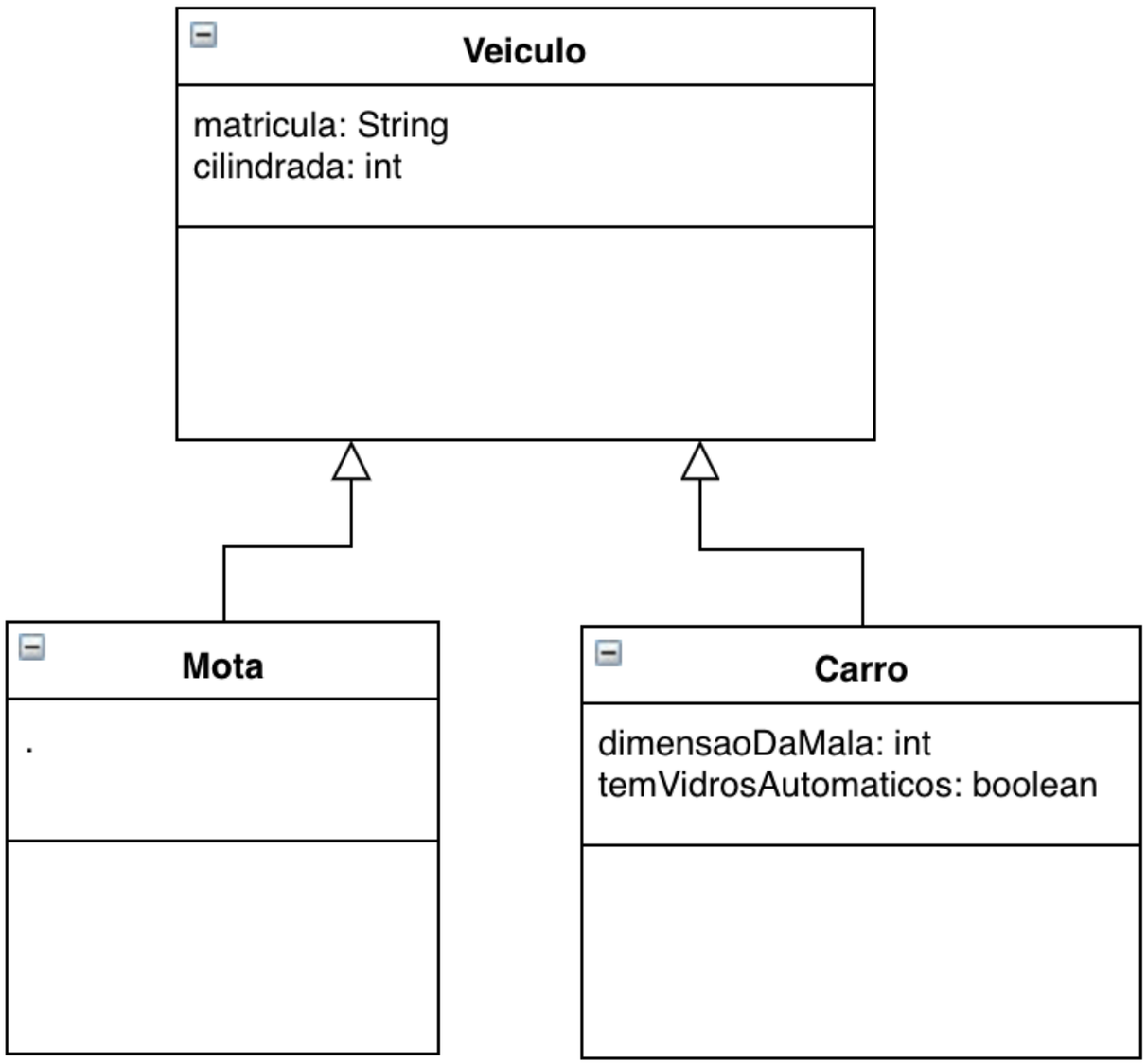
?

Herança + estrutura de memória

```
class Veiculo {
    String matricula;
    int cilindrada;
}

class Carro extends Veiculo {
    int dimensaoDaMala;
    boolean temVidrosAutomaticos;
}

class Mota extends Veiculo {

}

public class App {
    public static void main(String[] args) {
        Veiculo v = new Veiculo();
        v.matricula = "11-AA-11";
        v.cilindrada = 1000;

        Carro c = new Carro();
        c.matricula = "22-BB-22";
        c.dimensaoDaMala = 50;
        c.temVidrosAutomaticos = false;
        c.cilindrada = 1300;

        Mota m = new Mota();
        m.cilindrada = 600;
    }
}
```



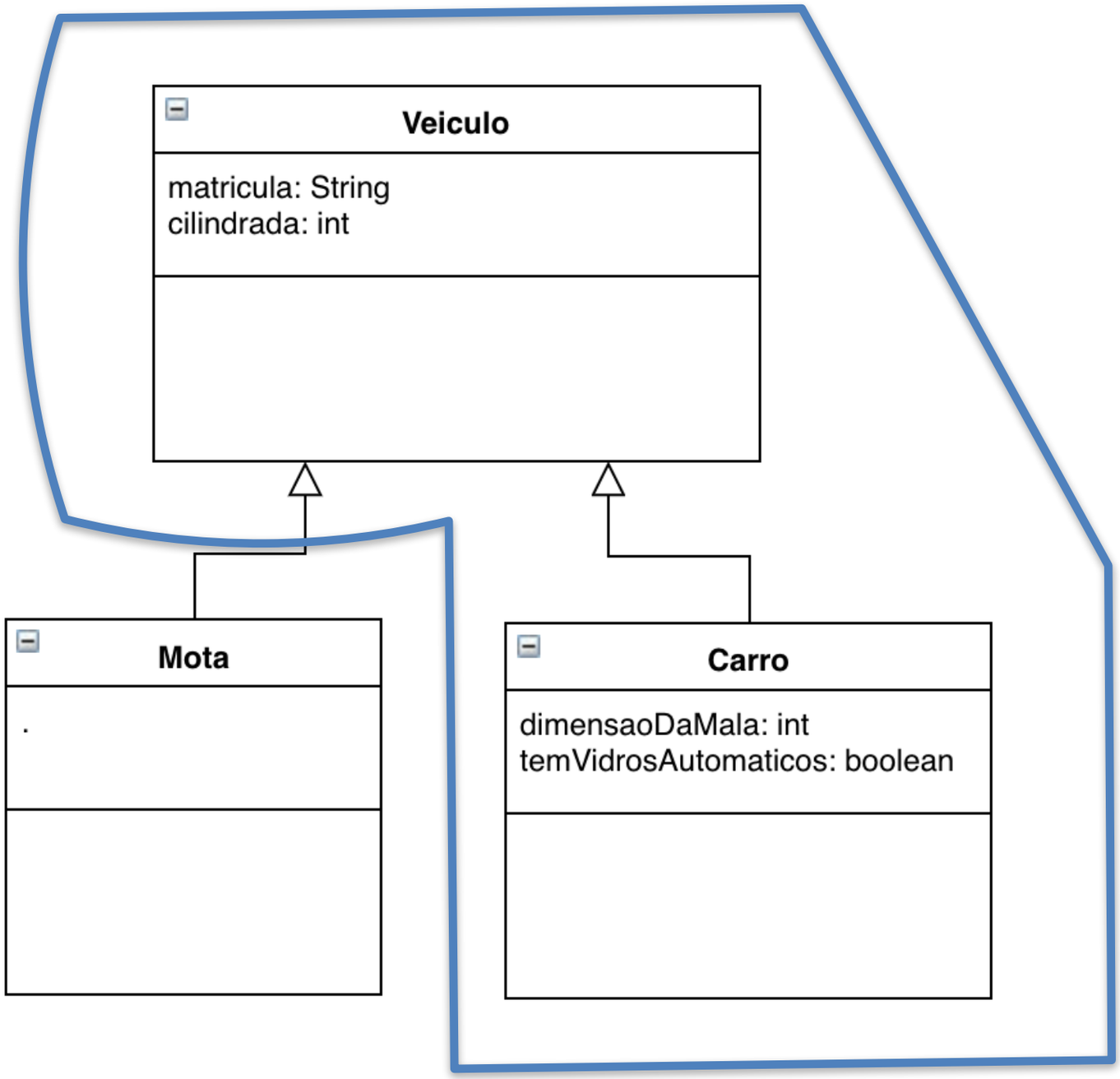
Herança + estrutura de memória



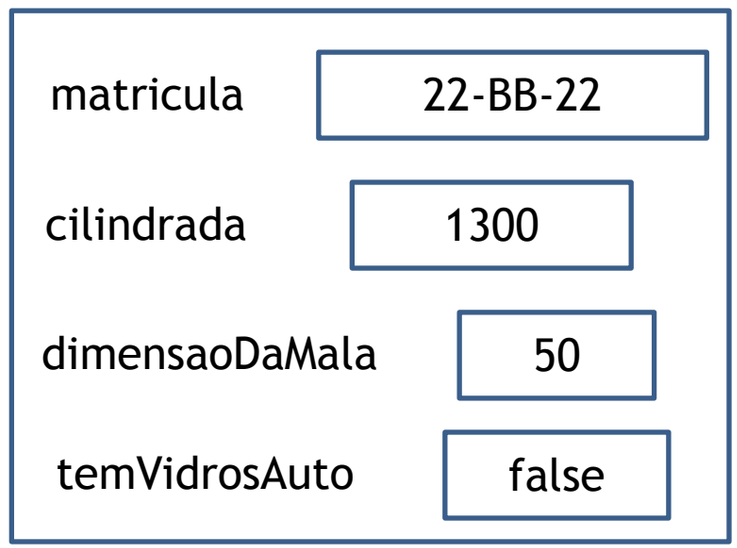
Pensamento errado:
Como são duas caixas de UML, são duas caixas em memória

Pensamento correcto:
Cada “new” dá origem a uma única caixa na memória

Herança + estrutura de memória




```
Carro c = new Carro();
```



Exercício com herança

breakout rooms

<u>Colaborativamente</u> pensar e desenhar o diagrama que resolve o problema

1. Entram no breakout room, onde devem estar mais 4 ou 5 colegas
2. Descarregam o ficheiro que coloquei no chat, que contém este e o próximo slide (pois vão deixar de ver o écran do professor)
3. Escolhem quem vai desenhar o diagrama (o desenhador)
4. O desenhador acede a draw.io e partilha o seu écran
5. Em conjunto discutem a solução enquanto o desenhador o desenha
6. Logo após aviso do professor, o desenhador envia captura de écran do diagrama pelo teams (p4997), indicando os elementos do grupo (número e nome)
7. Voltam à sala principal

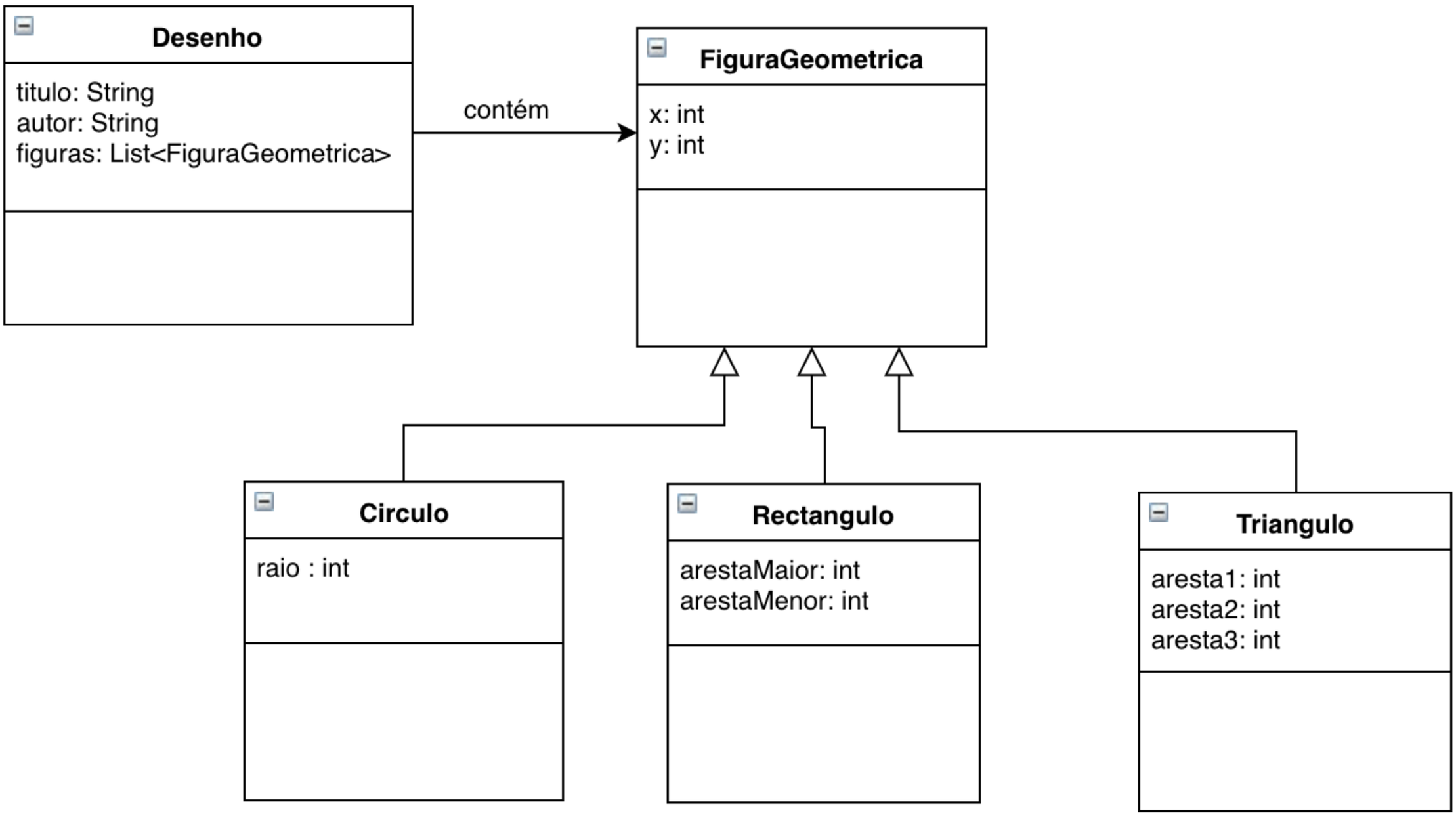
Exercício com herança

breakout rooms

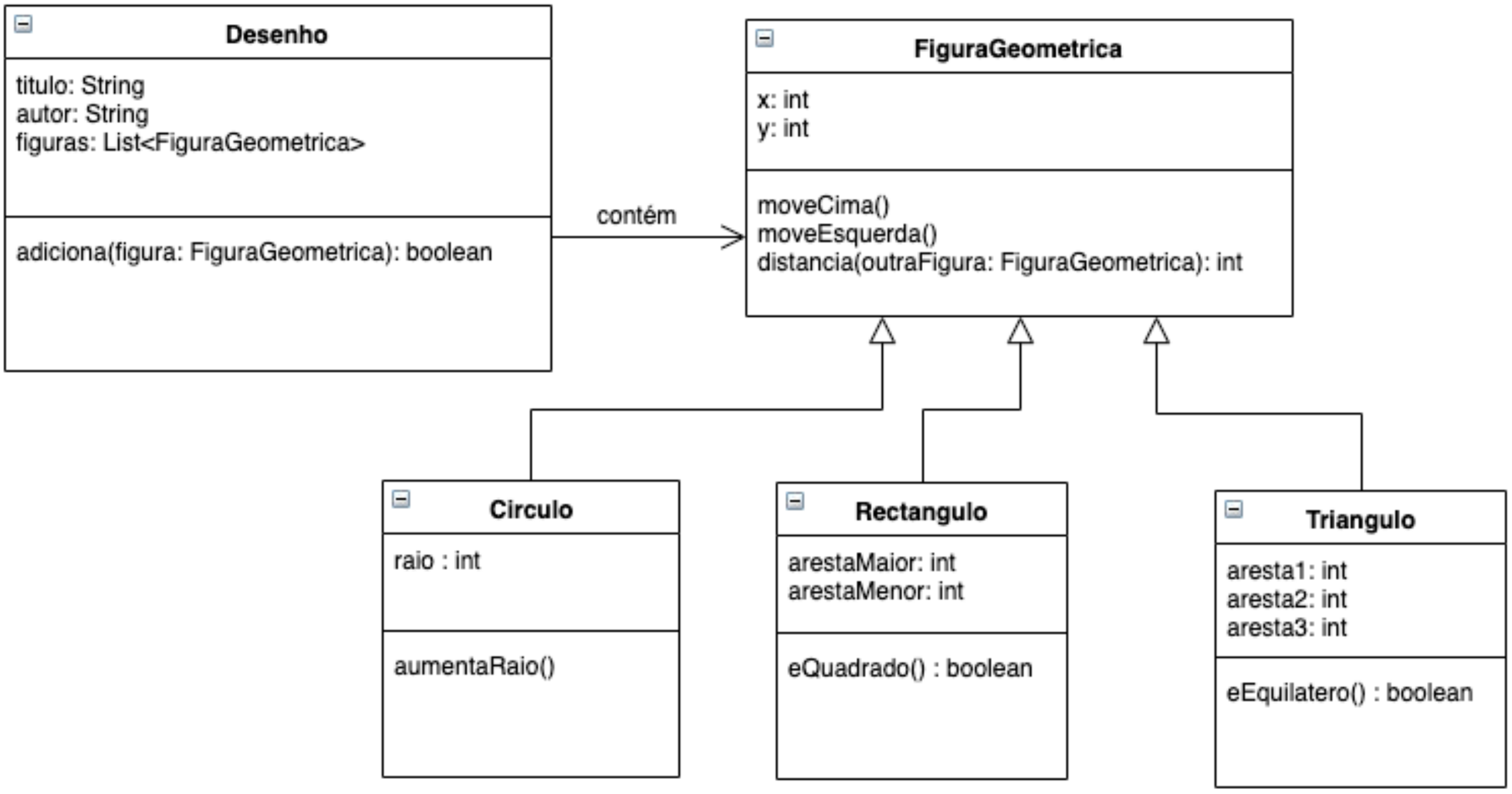
Pretende-se desenvolver um programa de desenho geométrico. Nesse programa, cada desenho está associado a um título e um autor e é constituído por várias figuras geométricas. As figuras geométricas podem ser círculos, rectângulos ou triângulos.

Desenhem em UML as cinco classes necessárias para implementar este programa. Deverão usar herança. Todas as classes têm que ter pelo menos 1 atributo e 1 método (não poder ser um getter nem um setter). Não é necessário incluir os construtores.

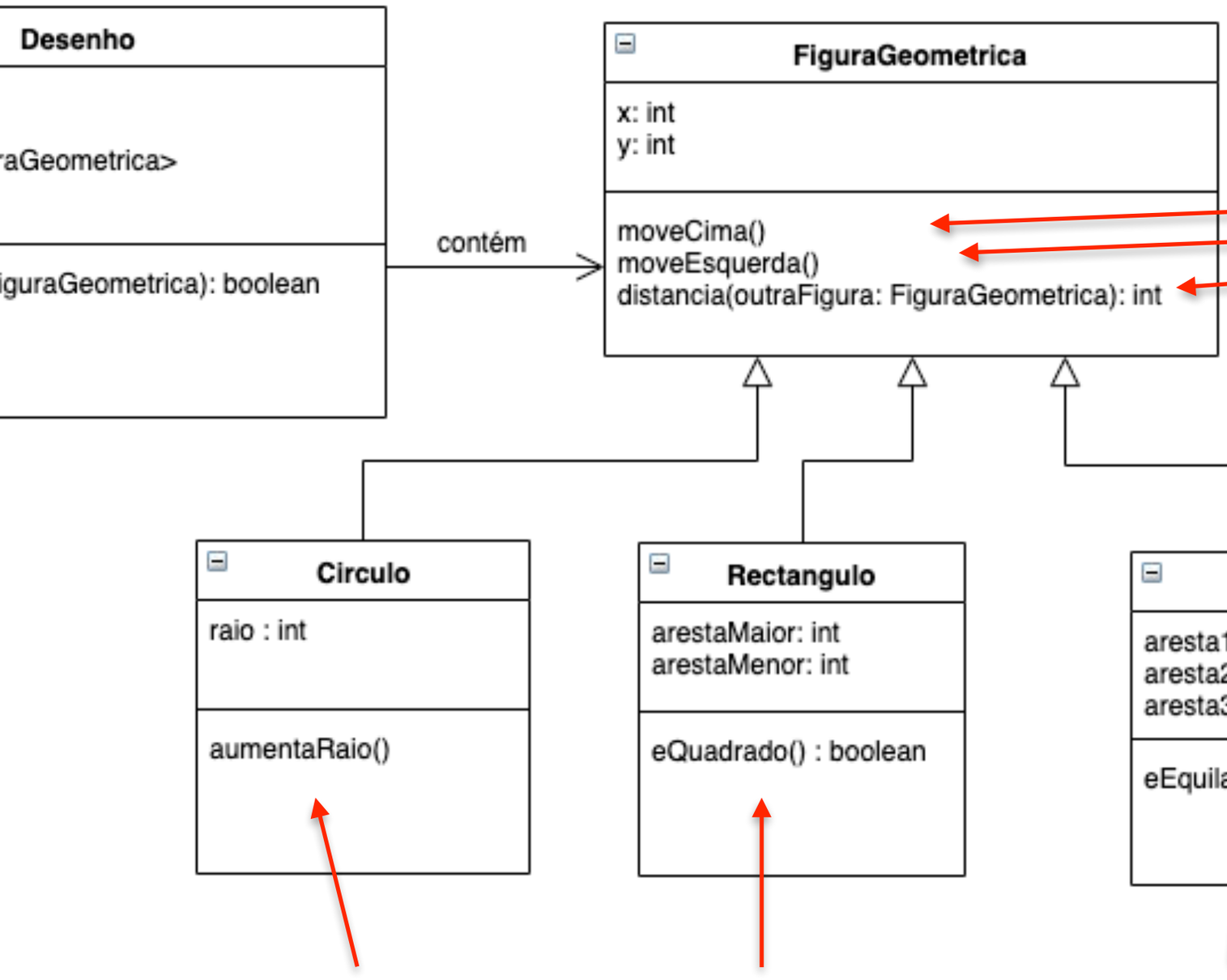
Resolução



Resolução



Resolução



Estes métodos funcionam da mesma forma independentemente da figura geométrica. Por isso estão no "pai".

Estes métodos só fazem sentido na figura específica ao qual pertencem. Por isso estão no "filho".

Próxima aula

Todas as figuras geométricas têm uma área mas o seu cálculo varia de figura para figura