




Alisson RS

[www.alissonrs.com.br](http://www.alissonrs.com.br)

[Moodle](#) 

# Linguagem de programação II



## Programação Orientada a Objetos: Herança



# POO Herança



“É um” ou “Tem um”? 





# POO

## características

Herde, encapsule e reutilize

# POO características



- Composição
- Herança
- Encapsulamento
- Polimorfismo
- Abstração



Mecanismos poderosos que facilitam a construção de código complexo.

# POO características



- **Composição**
- **Herança**
- **Encapsulamento**
- Polimorfismo
- Abstração



Mecanismos poderosos que facilitam a construção de código complexo.

# POO características



## Cada um cuida de si

- Classes devem cuidar e manipular suas próprias propriedades
- Sempre que possível os atributos devem ser privados.
  - Métodos Get e Set devem fornecer acesso as informações fora da classe.
- Encapsulamento facilita o controle dos dados, a manutenção das classes tornando o código mais reutilizável, estável e com maior capacidade de evoluir.



## Cada um cuida de si

- Em Java
  - Utilize atributos privados.
    - Se preciso implemente métodos públicos set e get.
  - Métodos públicos somente se necessários fora da classe.
  - Use visibilidade protected em caso de herança.
    - Cuidado com suas falhas (mais nessa apresentação)



# Composição

- Instanciar objetos dentro de uma classe.
  - Ou seja, uma classe que possui outros objetos em seu interior.
  - Utiliza a nomenclatura “TEM UM”



# Composição

- Instanciar objetos dentro de uma classe.

```
public class CalculadoraPares{  
    TextField txtResultado = new TextField("X ? Y = ?");  
    Button bt0 = new Button("0");  
    Button bt2 = new Button("2");  
    Button bt4 = new Button("4");  
    Button bt6 = new Button("6");  
    Button bt8 = new Button("8");  
  
    Button btMais = new Button("+");  
    Button btMenos = new Button("-");  
    Button btMult = new Button("*");  
    Button btDiv = new Button("/");  
    Button btIgual = new Button("=");  
}
```

os em



# Composição

- Instanciar objetos dentro de uma classe.

```
public class CalculadoraPares{  
    TextField txtResultado = new TextField("X ? Y = ?");  
    Button bt0 = new Button("0");  
    Button bt2 = new Button("2");  
    Button bt4 = new Button("4");  
    Button bt6 = new Button("6");  
    Button bt8 = new Button("8");  
  
    Button btMais = new Button("+");  
    Button btMenos = new Button("-");  
    Button btMult = new Button("*");  
    Button btDiv = new Button("/");  
    Button btIgual = new Button("=");  
}
```

CalculadoraPares  
"TEM UM" Button bt0  
"TEM UM" Button bt1

os em





# Herança

Para não ter que reinventar a roda

# Herança

Mecanismo que permite reutilizar código de forma eficiente.

Classes são reutilizadas sem necessidade de copiar e colar.

Toda classe em Java utiliza herança.





# Hierarquia de Classes

FILHO



PAI

ou



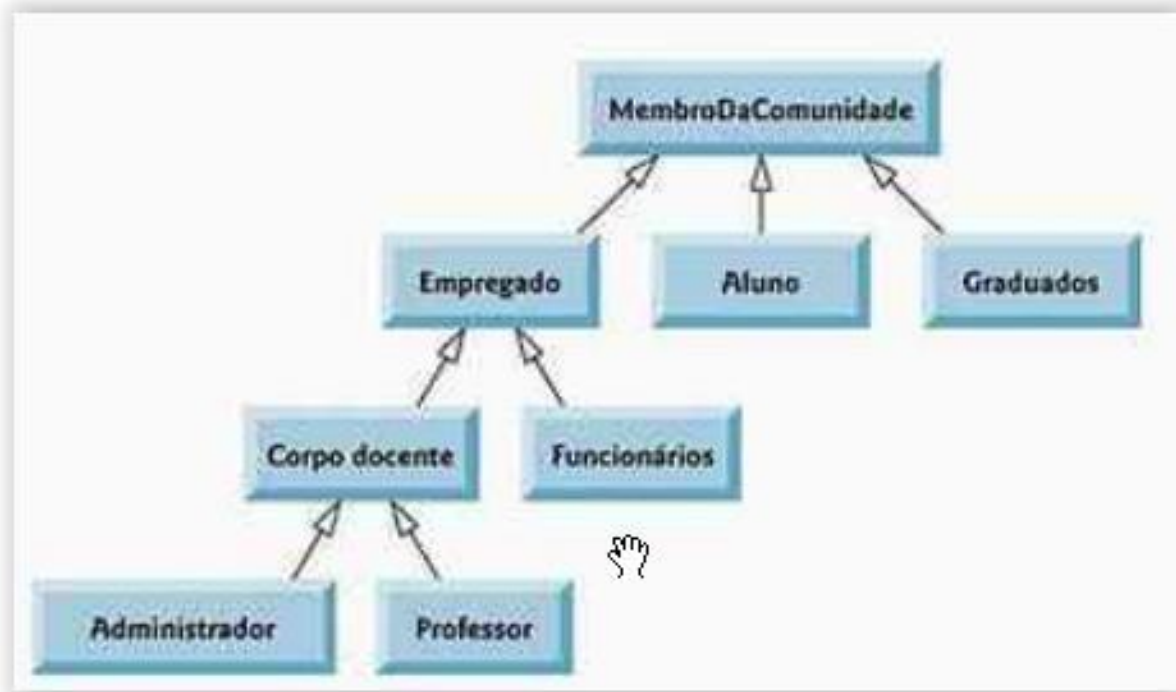
A classe apontada é a classe pai ou classe de base.

A classe que aponta é a classe filha ou classe derivada.

# Hierarquia de Classes

Subclasse  Superclasse

ou

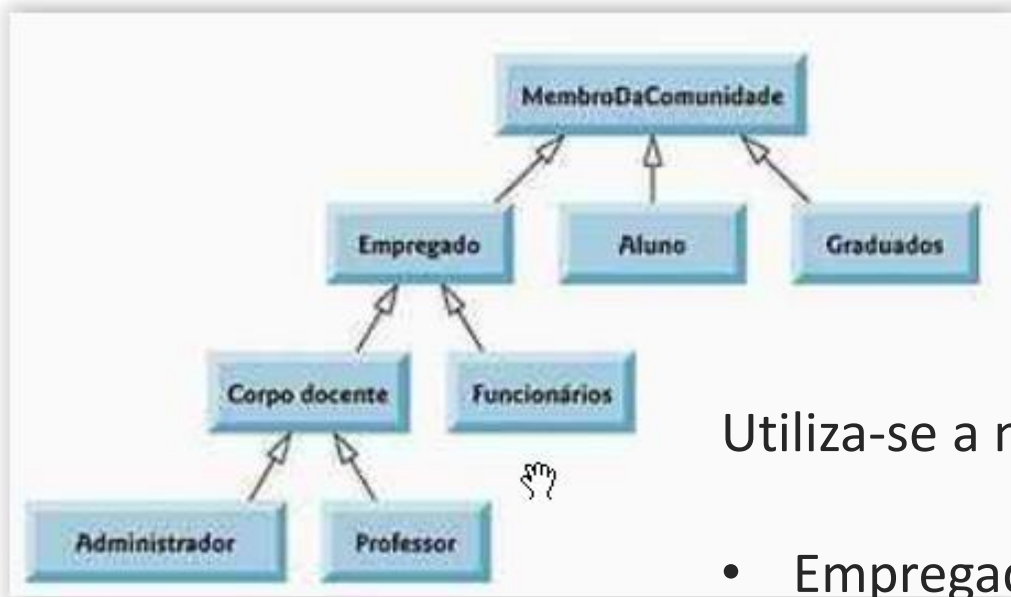


A classe apontada é a **superclasse** ou classe de base.

A classe que aponta é a **subclasse** ou classe derivada.



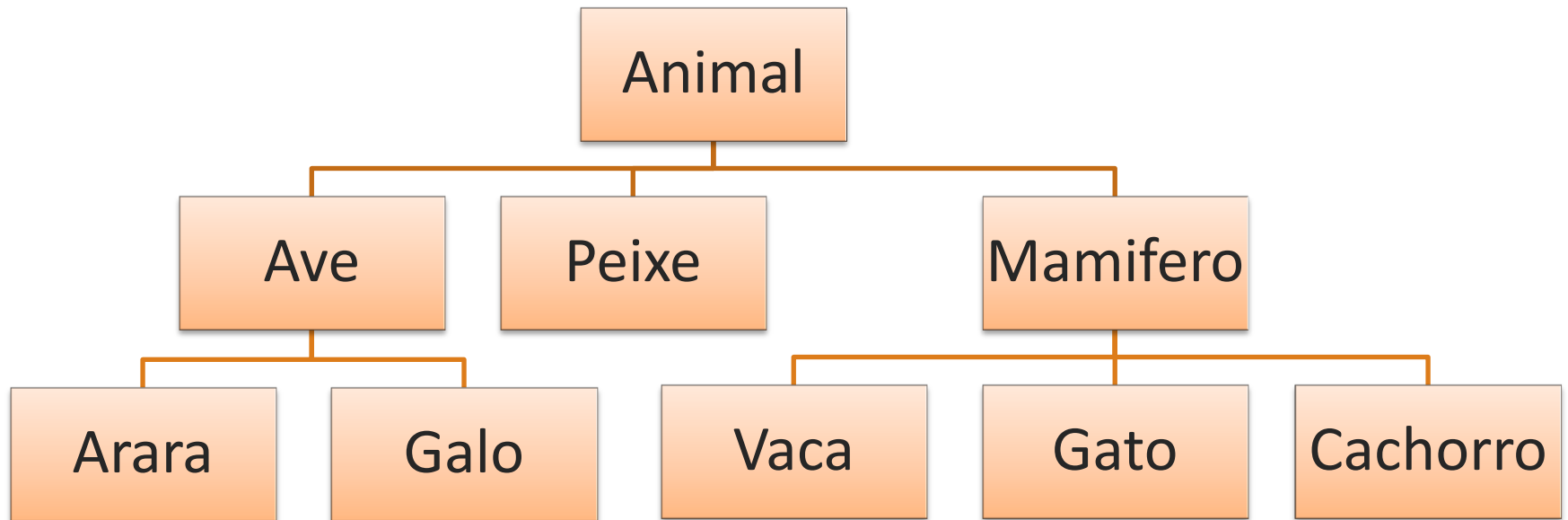
# Hierarquia de Classes



Utiliza-se a nomenclatura **É UM**

- Empregado **É UM** MembrodaComunidade
- Funcionário **É UM** Empregado

# Herança



```
1 package br.com.alissonrs.aula14;
2
3 public class Animal{
4
5     protected int idade;
6     protected String nome;
7
8     public Animal() {
12
13     public void reclamar(){
16 }
```

Arara

Galo

Vaca

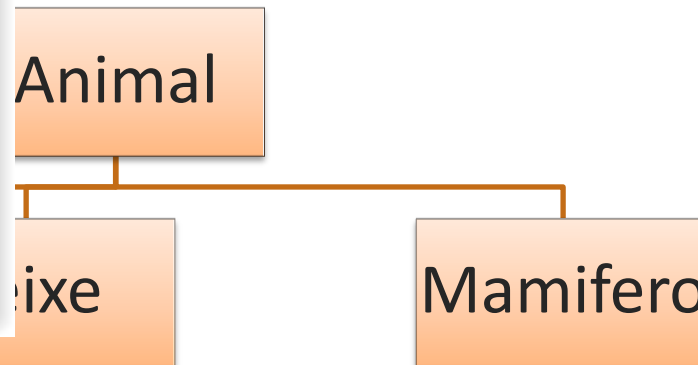
Gato

Cachorro

Mamifero



```
1 package br.com.alissonrs.aula14;
2
3 public class Animal{
4
5     protected int idade;
6     protected String nome;
7
8     public Animal() {..
12
13     public void reclamar(){..
16 }
```



```
1 package br.com.alissonrs.aula14;
2
3 public class Mamifero extends Animal {
4     public void mamar() {..
7 }
```

```
1 package br.com.alissonrs.aula14;
2
3 public class Animal{
4
5     protected int idade;
6     protected String nome;
7
8     public
12
13     public
16 }
```

```
1 package
2
3 public
4
7 }
```

```
1 package br.com.alissonrs.aula14;
2
3 public class Cachorro extends Mamifero {
4
5     private static int contar = 0;
6
7     private int numeroRegistro = 0;
8
9     public Cachorro() {..}
11
12     public Cachorro(String novoNome) {..}
16
18     public void reclamar() {..}
21
22     public void setNumeroRegistro(int numeroRegistro) {..}
25
26     public int getNumeroRegistro() {..}
29 }
```

# Herança

```
1 package br.com.alissonrs.aula14;
2
3 public class Zoologico {
4
5     public static void main(String[] args) {
6         Mamifero mama;
7         Cachorro cachorro;
8         Vaca vaca;
9
10        mama = new Mamifero();
11        cachorro = new Cachorro("Snoopy");
12        vaca = new Vaca();
13
14        mama.mamar();
15        cachorro.mamar();
16        vaca.mamar();
17
18        mama.reclamar();
19        cachorro.reclamar();
20        vaca.reclamar();
21    }
22 }
```

<terminated> Zoologico (6) [Java Application] C:\Program Files\  
xuc, xuc, xuc  
xuc, xuc, xuc  
xuc, xuc, xuc  
Inominado diz: zzzzzzzzzzzzzzzzzzz!!!  
Au, au, rrrrrr!  
Muuuuuuuuuu!!

# Herança : Sobrecarga x Sobrescrita

- Sobrecarregar método
  - Dois métodos com o mesmo nome, mas com assinatura diferente.

```
+ public Cachorro() {  
+  
+ public Cachorro(String novoNome) {  
+  
+ }
```



# Herança : Sobrecarga x Sobrescrita

---

- Sobrescrever método
  - Método com o mesmo nome e mesma assinatura, mas em outro nível na hierarquia de classes.
  - Sobrepuõ o método original.
    - Método original pode ser chamado por “super.NOMEDOMETODO()”.
  - @Override
    - Ajuda o compilador a verificar se existe o método na hierarquia.



# Herança : Sobrecarga x Sobrescrita

- Sobrescrever método

```
public class Animal {  
  
    public void reclamar() {  
        System.out.println(nome + " diz: zzzzzzzzz!!!");  
    }  
}
```

Presente na  
classe Animal

```
public class Mamifero extends Animal {
```

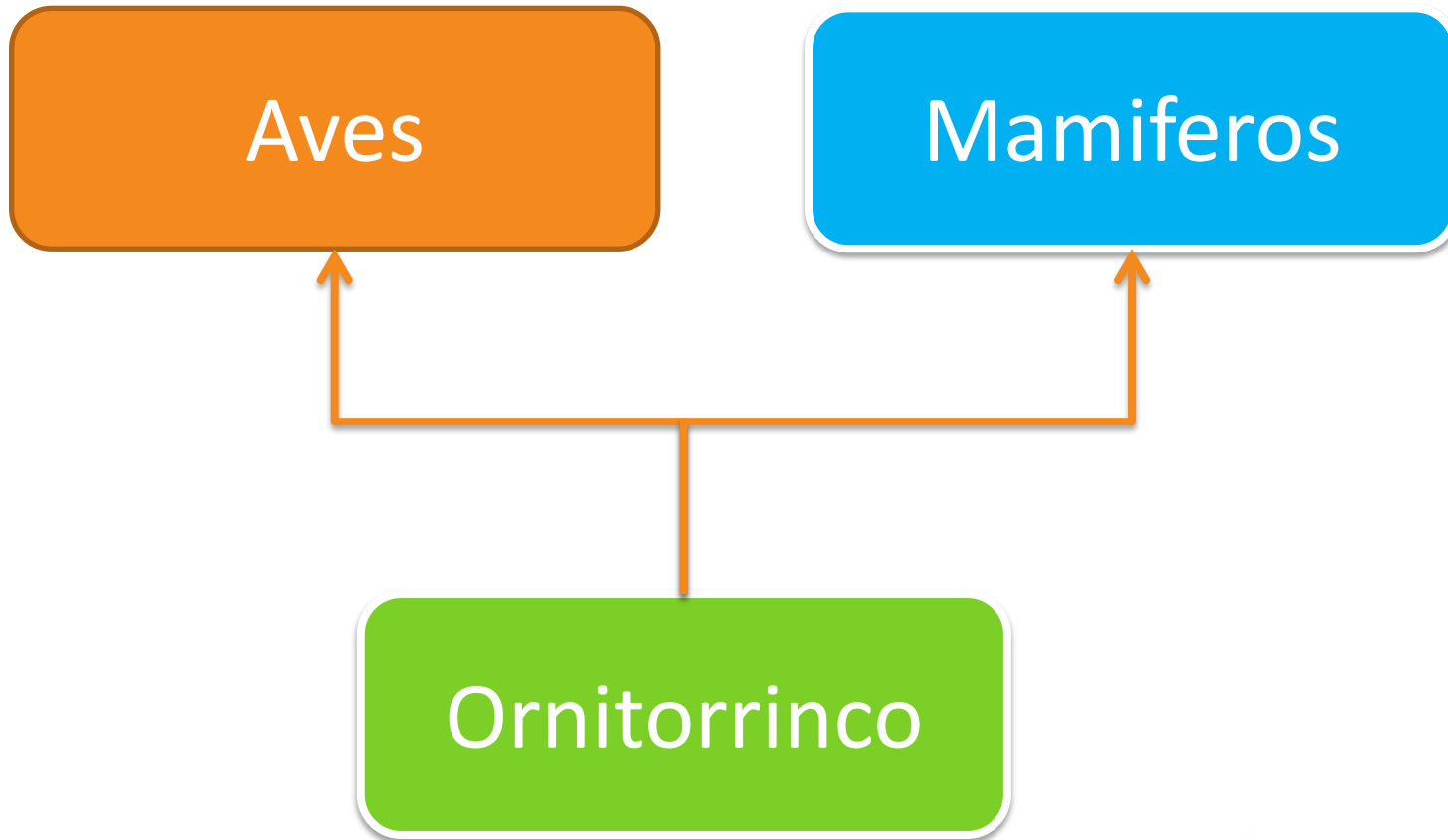
Não tem implementação  
na classe Mamifero

Sobrescrito na classe  
Cachorro.

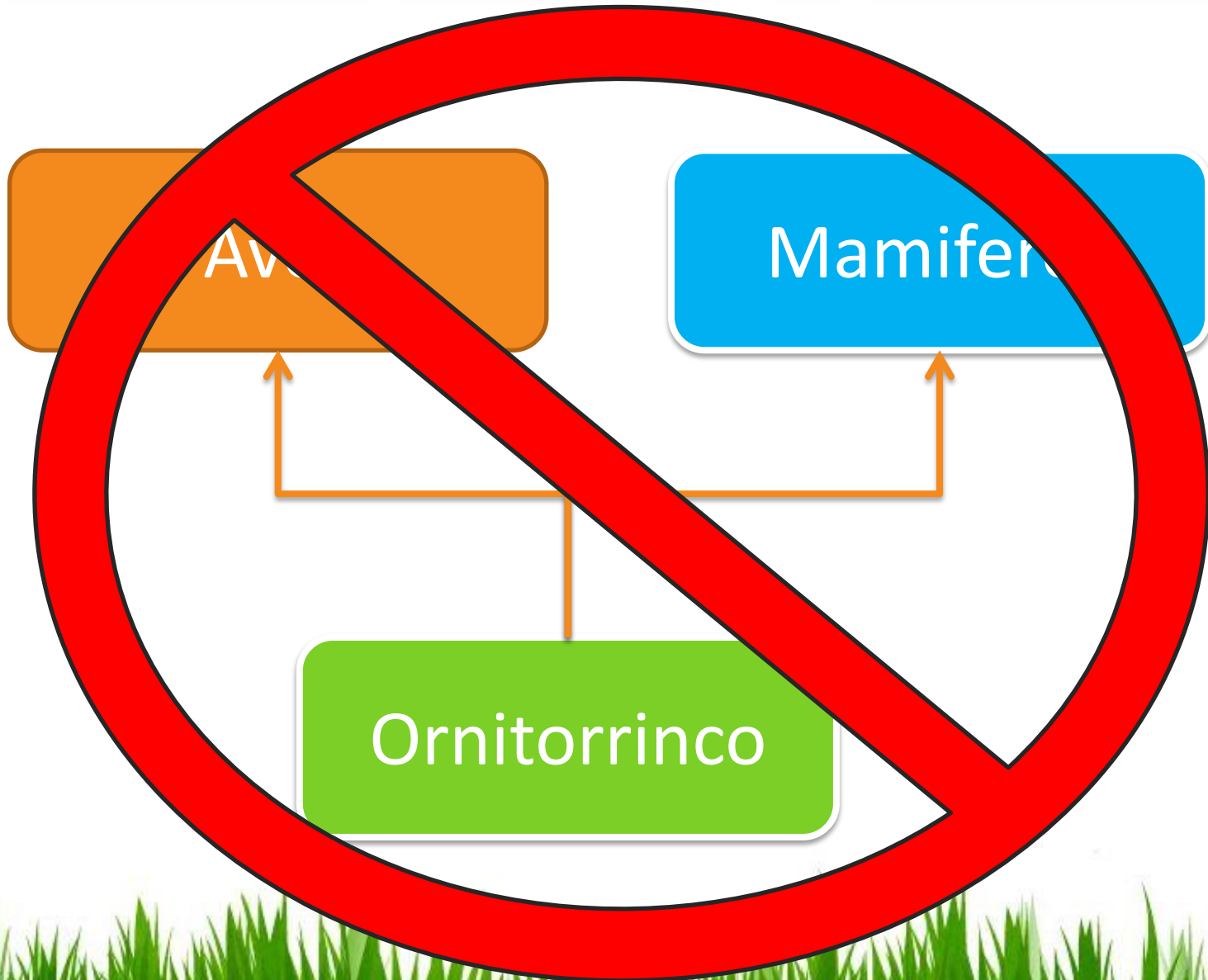
```
public class Cachorro extends Mamifero {  
  
    @Override  
    public void reclamar() {  
        System.out.println("Au, au, rrrrrr!");  
    }  
}
```

# Herança : Herança múltipla

- Herdar de duas classes diferentes



# Herança : Herança múltipla



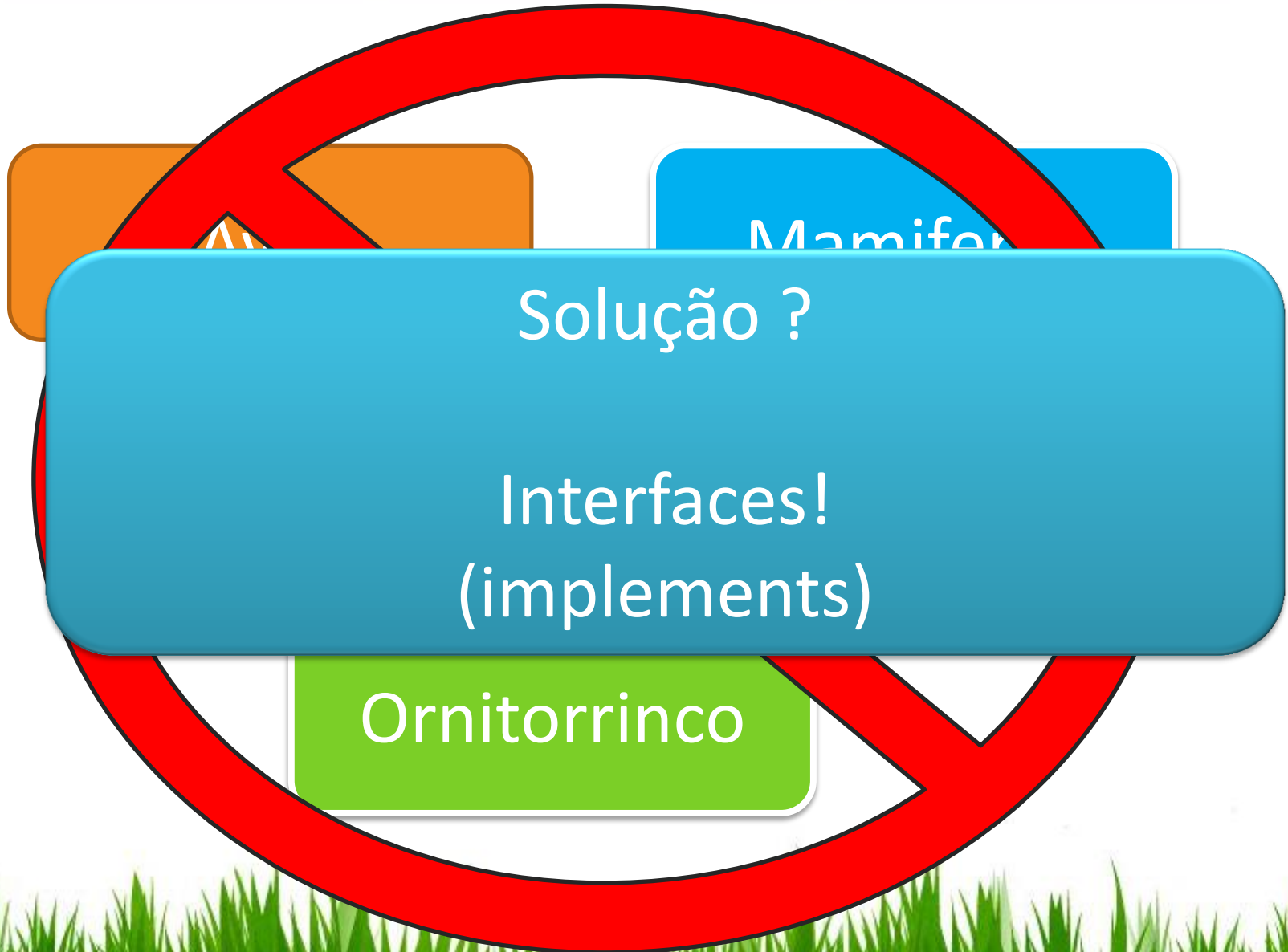
# Herança : Herança múltipla



Não é permitido em Java!  
Para Java esse animal não existe!

Ornitorrinco

# Herança : Herança múltipla





# Classe Object

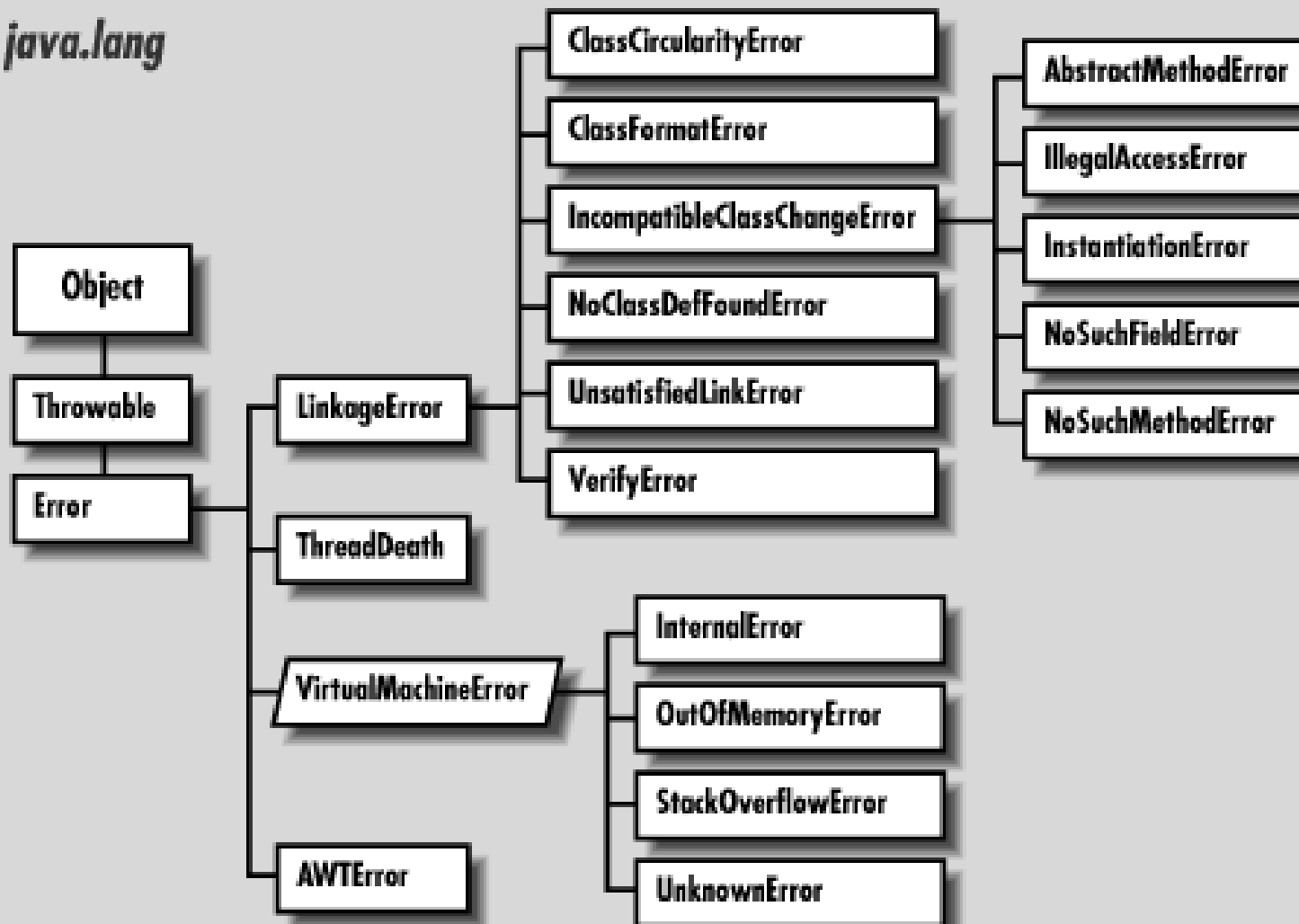
A classe mãe







*java.lang*

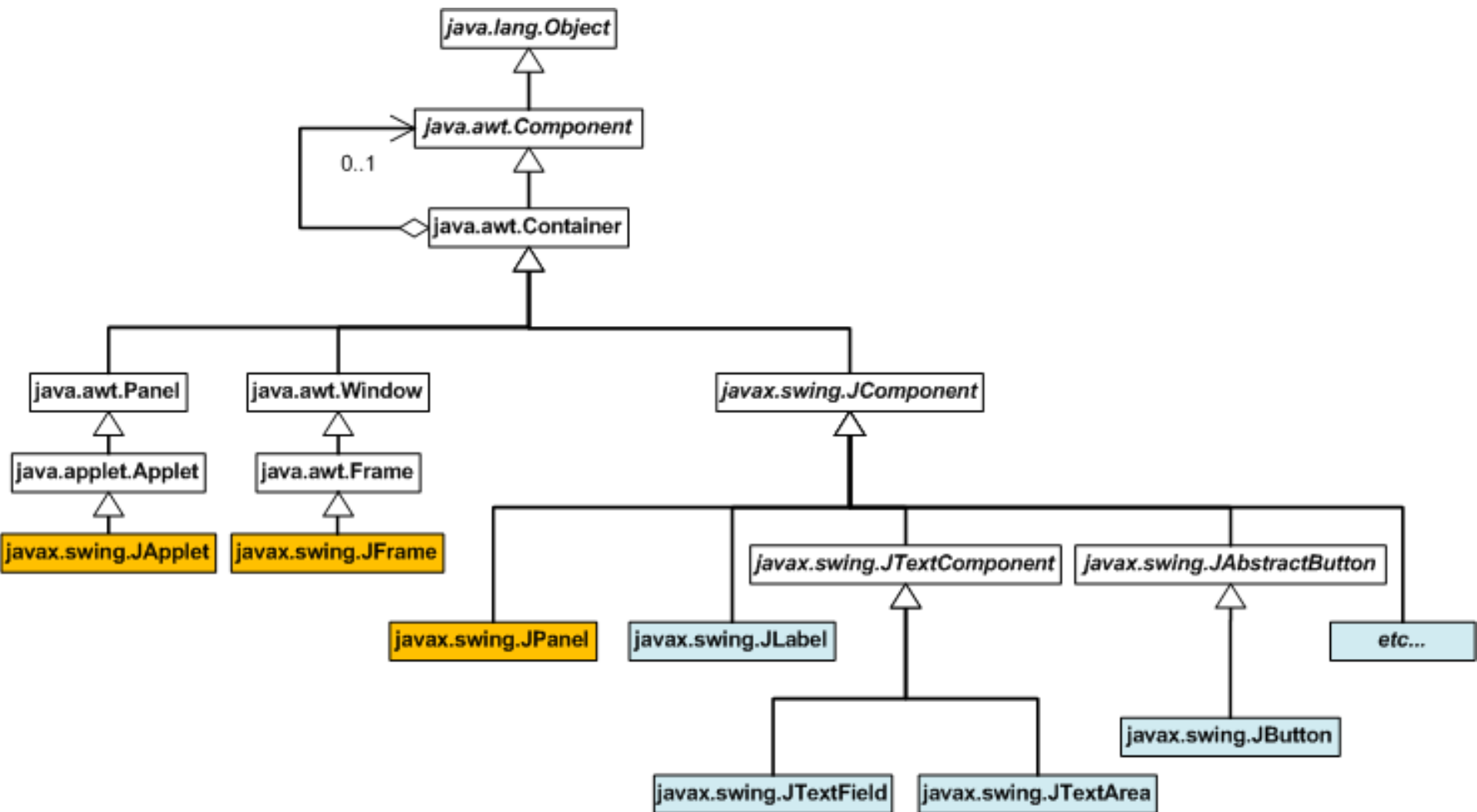


**KEY**

CLASS

ABSTRACT CLASS

extends



# A classe Object

- Base para todas as demais classes
- Traz métodos importantes
  - Alguns para serem sobrescritos

## Object as a Superclass

The `Object` class, in the `java.lang` package, sits at the top of the class hierarchy tree. Every class is a subclass of `Object`. If you choose to do so, you may need to override them with code that is specific to your class. The methods

- `protected Object clone() throws CloneNotSupportedException`  
Creates and returns a copy of this object.
- `public boolean equals(Object obj)`  
Indicates whether some other object is "equal to" this one.
- `protected void finalize() throws Throwable`  
Called by the garbage collector on an object when garbage collection determines that there are no more references to the object
- `public final Class getClass()`  
Returns the runtime class of an object.
- `public int hashCode()`  
Returns a hash code value for the object.
- `public String toString()`  
Returns a string representation of the object.

# A classe Object

---

- protected Object clone() throws CloneNotSupportedException
- public boolean equals(Object obj)
- protected void finalize() throws Throwable
- public final Class getClass()
- public int hashCode()
- public String toString()

# A classe Object

---

## Métodos para controle de programação multitarefa

- `public final void notify()`
- `public final void notifyAll()`
- `public final void wait()`
- `public final void wait(long timeout)`
- `public final void wait(long timeout, int nanos)`







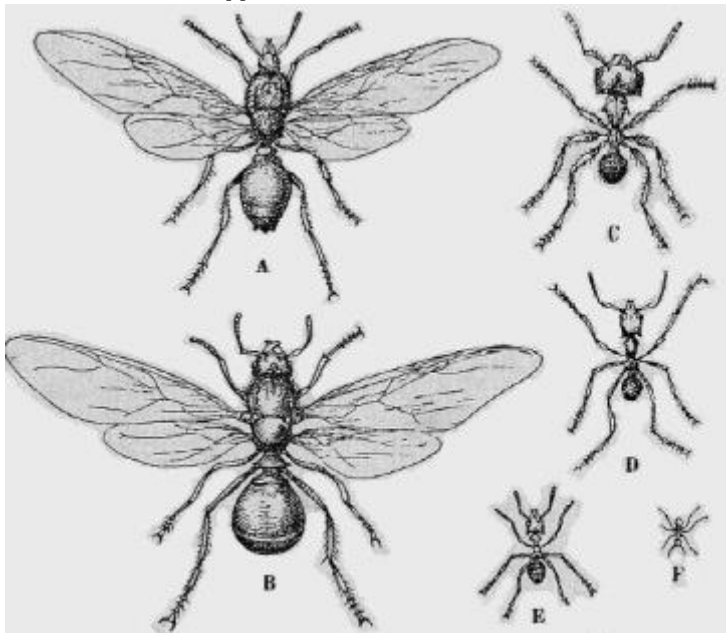
**Mas espere...**  
Há mais!

Porque ainda está muito fácil.

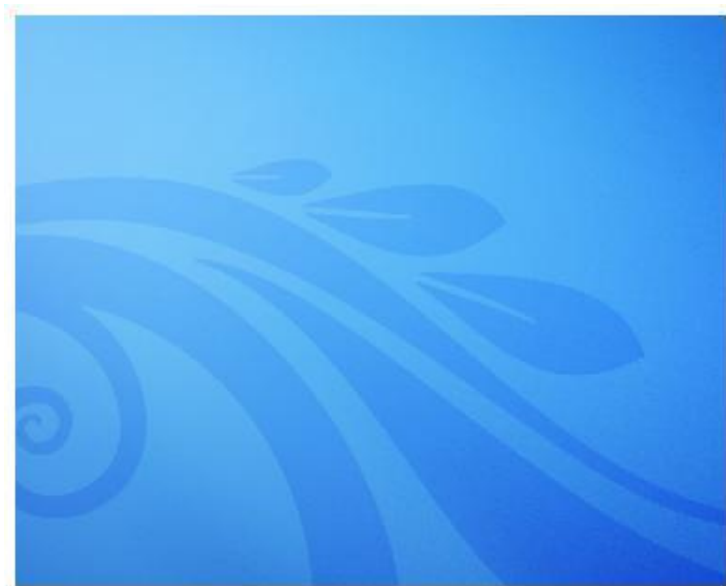


# Polimorfismo – um nome, várias formas.

- » Vaca é um Mamifero
- » **Mamifero pode ser uma vaca?**
- » E se misturar?
- » Mamifero mamaVaca = new Vaca();
- » Mamifero mamaCao = new Cachorro();



- » Animal animal = new Peixe() ;
- » Peixe peixe = (Peixe) animal;
- » ....



**Perguntas?**

