FIAP GRADUAÇÃO

Agenda

- Encapsulamento
- Modificadores de Acesso
- JavaBeans
- Herança

Encapsulamento

É o conceito de ocultar ou expor os os atributos e métodos das classes evitando que outras classes tenham acesso direto a eles.

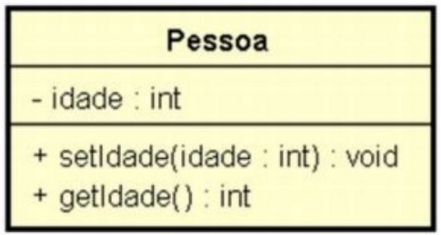
Encapsulamento - Benefícios

- “Esconde” detalhes da implementação da classe
- Força outras classes a utilizarem os métodos para acessar as propriedades
- Permite definir, quando, como e se os atributos devem ser alterados ou acessados

Encapsulamento - getters / setters

- Para encapsular um atributo podemos utilizar os métodos get/set. No exemplo abaixo, a única forma de buscar ou atualizar o valor da idade é através dos métodos:

```
public class Pessoa {

   private int idade;

   public void setIdade(int idade){
      this.idade = idade;
   }

   public int getIdade(){
      return idade;
   }

}
```



Encapsulamento - getters / setters

Não precisamos sempre que os métodos get e set sejam sempre iguais.

Podemos implementar a lógica destes métodos conforme a necessidade.

```
public class Pessoa {

    private String ddd;
    private String numero;

    //getters and setters

    public String getTelefoneFormatado() {
        return "(" + ddd + ") " + numero;
    }
```

Encapsulamento - getters / setters

Nem sempre vamos devemos declarar os getters e setters. Devemos analisar cada caso.

```
public class ContaCorrente {

    private double saldo;

    public double getSaldo(){
        return this.saldo;
    }

    public void depositar(double valor){
        this.saldo = saldo + valor;
    }

    public void sacar(double valor){
        this.saldo = saldo - valor;
    }

}
```

Modificadores de acesso

No Java podemos declarar qual o nível de acesso aos nossos atributos, métodos, construtores, classes e interfaces.

Até agora utilizamos os modificadores “public” (público) para visibilidade total e “private” (privado) para visibilidade apenas na classe.

Modificadores de acesso

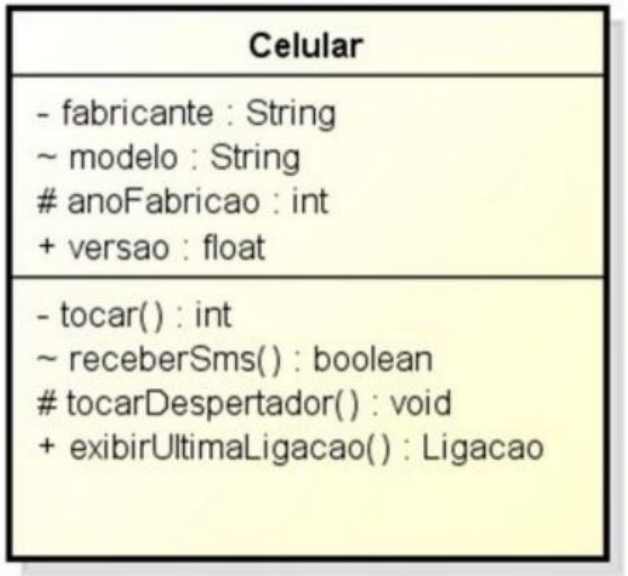
- No Java temos 4 níveis de modificadores:

Símbolo	Palavra-chave	Descrição
-	**private**	Atributos e métodos são acessíveis somente nos métodos da própria classe. Este é o nível mais rígido de encapsulamento.
~		Atributos e métodos são acessíveis somente nos métodos das classes que pertencem ao pacote em que foram criados.
#	**protected**	Atributos e métodos são acessíveis nos métodos da própria classe e suas subclasses.
+	**public**	Atributos e métodos são acessíveis em todos os métodos de todas as classes. Este é o nível menos rígido de encapsulamento.

Modificadores de acesso

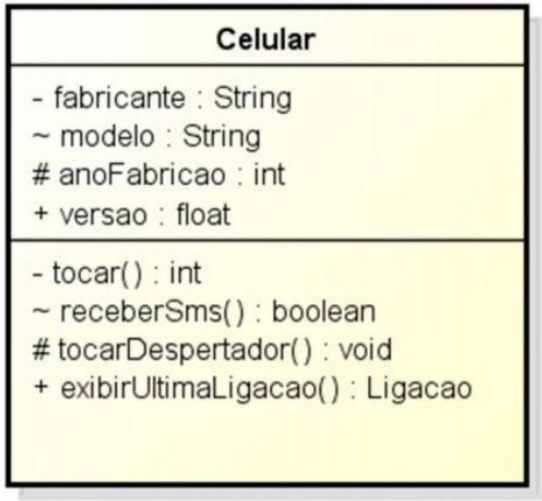
Forma de representação no UML

- (private)
~ (*default*)
# (protected)
+ (public)



Modificadores de acesso

```
private String fabricante;

String modelo;

protected int anoFabricacao;

public float versao;



private int tocar(){ ... }

boolean receberSms(){ ... }

protected void tocarDespertador(){ ... }

public Ligacao exibirUltimaLigacao(){ ... }
```



Java Beans

São componentes reutilizáveis que isolam e encapsulam um conjunto de funcionalidades.

Pode também ser definido como um conjunto de convenções de design e nomeação da especificação Java Single Edition (JSE).

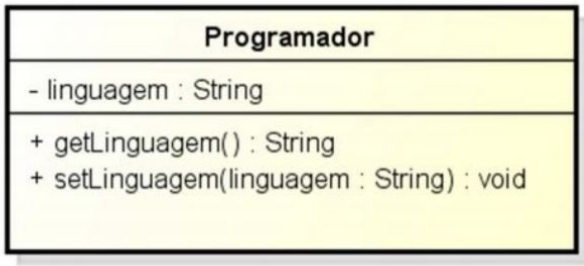
Java Beans

Para que uma classe seja considerada um Java Bean, ela deve seguir 3 regras:

- Os atributos devem ser todos privados e acessados através de getters e setters
- Deve ter um construtor que não recebe nenhum parâmetro
- Implementar a interface Serializable

Java Beans

```java
import java.io.Serializable;

public class Programador implements Serializable {

   private String linguagem;

   public String getLinguagem(){
      return linguagem;
   }

   public void setLinguagem(String linguagem){
      this.linguagem = linguagem;
   }

}
```



Herança

É um mecanismo da Programação orientada a objetos. Através da herança podemos criar novas classes a partir de classes já existentes, desta forma, reutilizando os atributos e métodos já existentes.

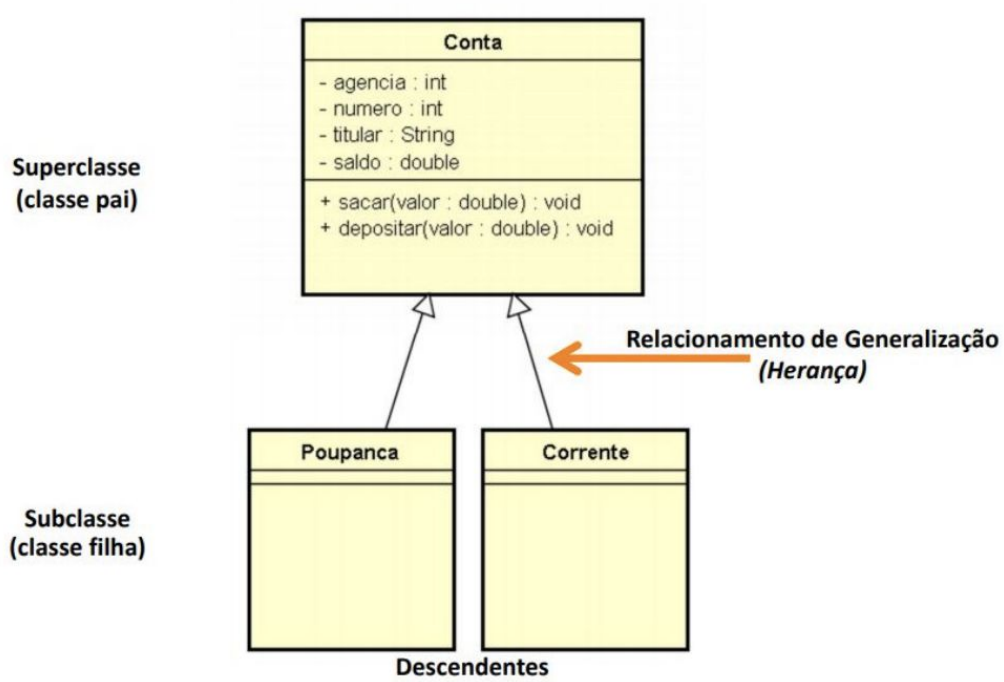
Herança - Super e sub classe

Para utilizar a herança precisamos de duas classes, uma super classe (pai) e uma sub classe (filha).

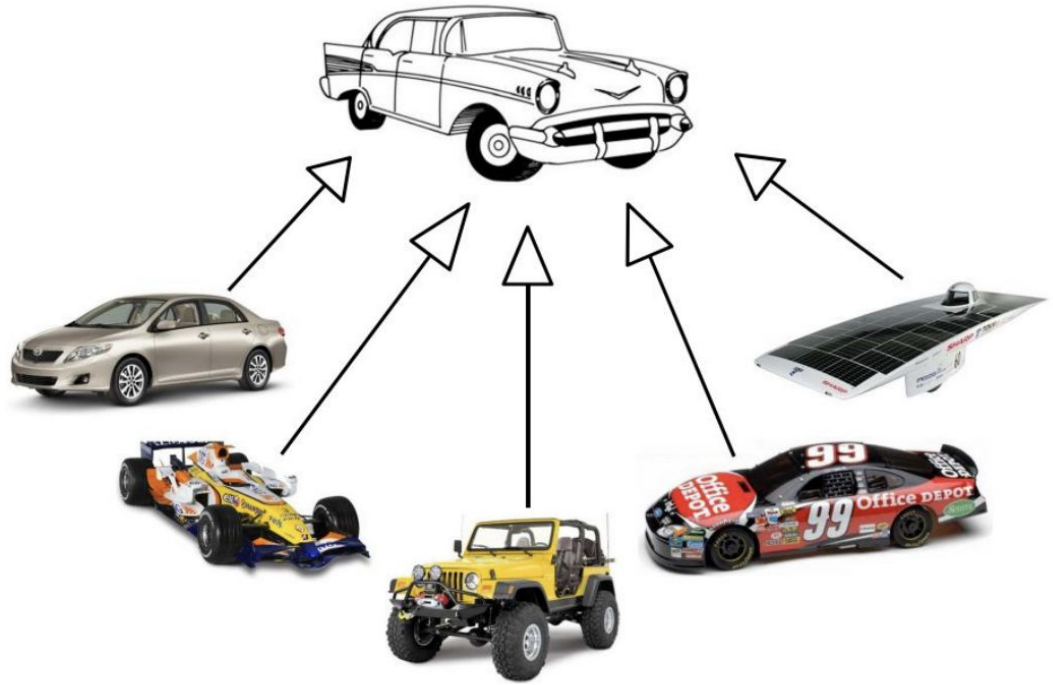
A sub classe “herda” as características e os comportamentos da super classe, podendo acrescentar novas características e comportamentos ou mesmo alterar os comportamentos existentes na classe pai.

Importante: No Java só podemos herdar de uma única classe pai

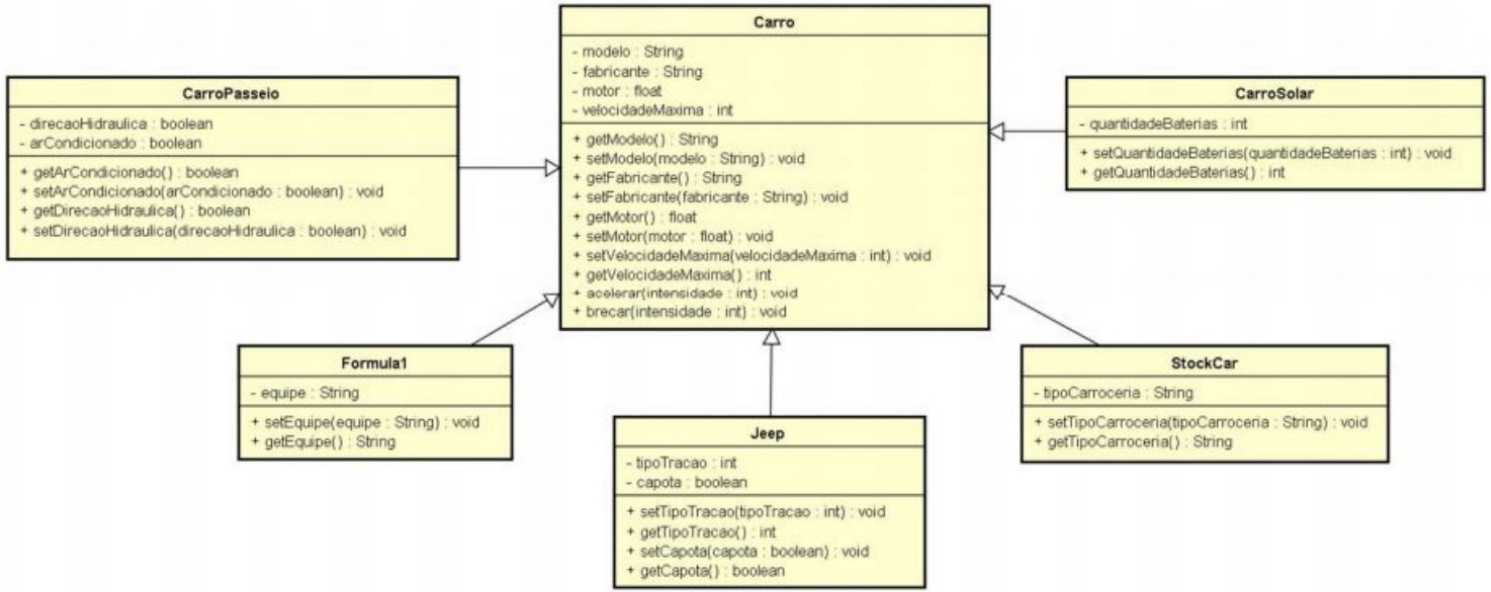
Herança - UML


Conta
- agencia : int
- numero : int
- titular : String
- saldo : double
+ sacar(valor : double) : void
+ depositar(valor : double) : void
Superclasse
(classe pai)
Relacionamento de Generalização
(Herança)
Poupanca
Corrente
Subclasse
(classe filha)
Descendentes

Herança - Exemplo



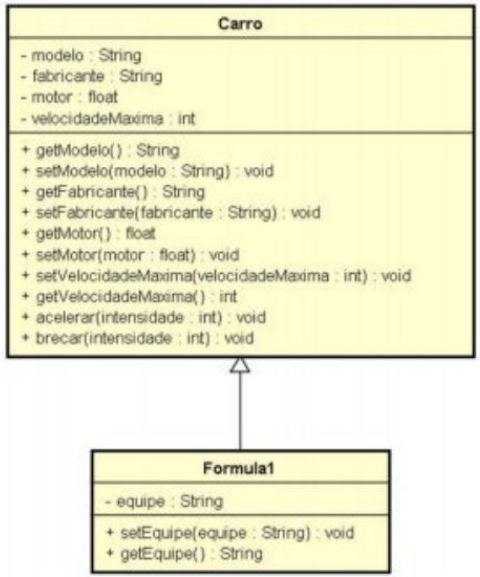
Herança - Exemplo



Herança - Sintaxe

Para utilizar herança no Java temos a palavra reservada “extends”.

```
public class Formula1 extends Carro {

    private String equipe;

    public void setEquipe(String equipe) {
        this.equipe = equipe;
    }

    public String getEquipe() {
        return equipe;
    }

}
```



Herança - Classe Object

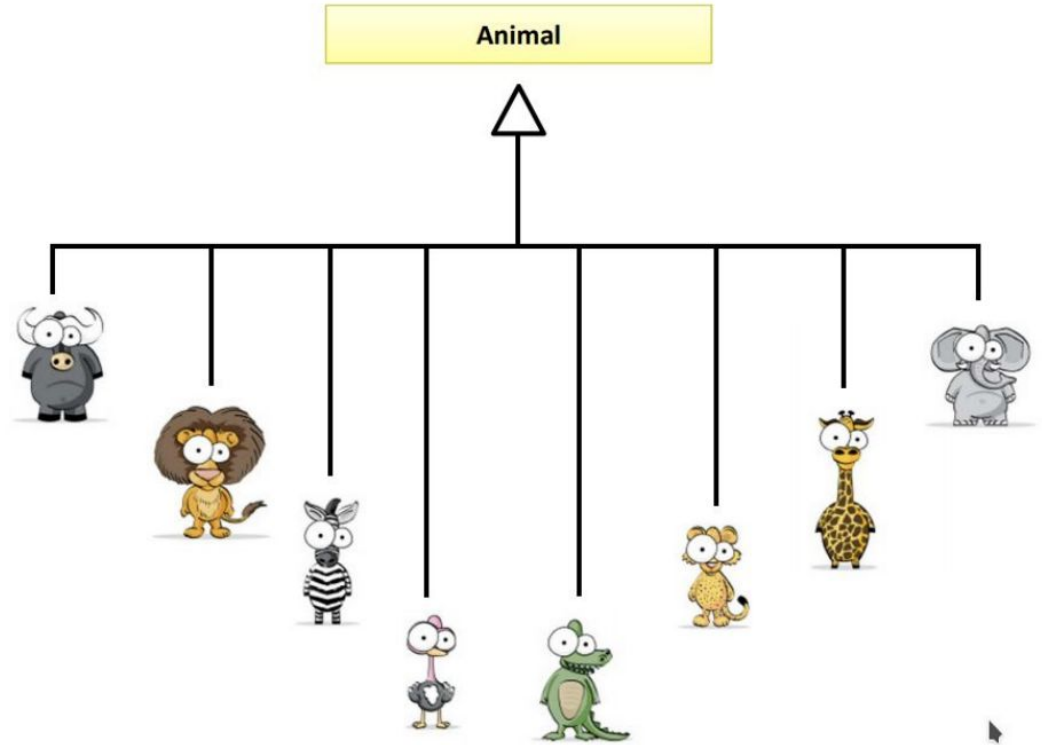
Todas as classes que criamos até hoje já são classes que herdam da super classe Object.

Se omitimos a palavra chave “extends”, automaticamente a nossa classe herda da classe Object.

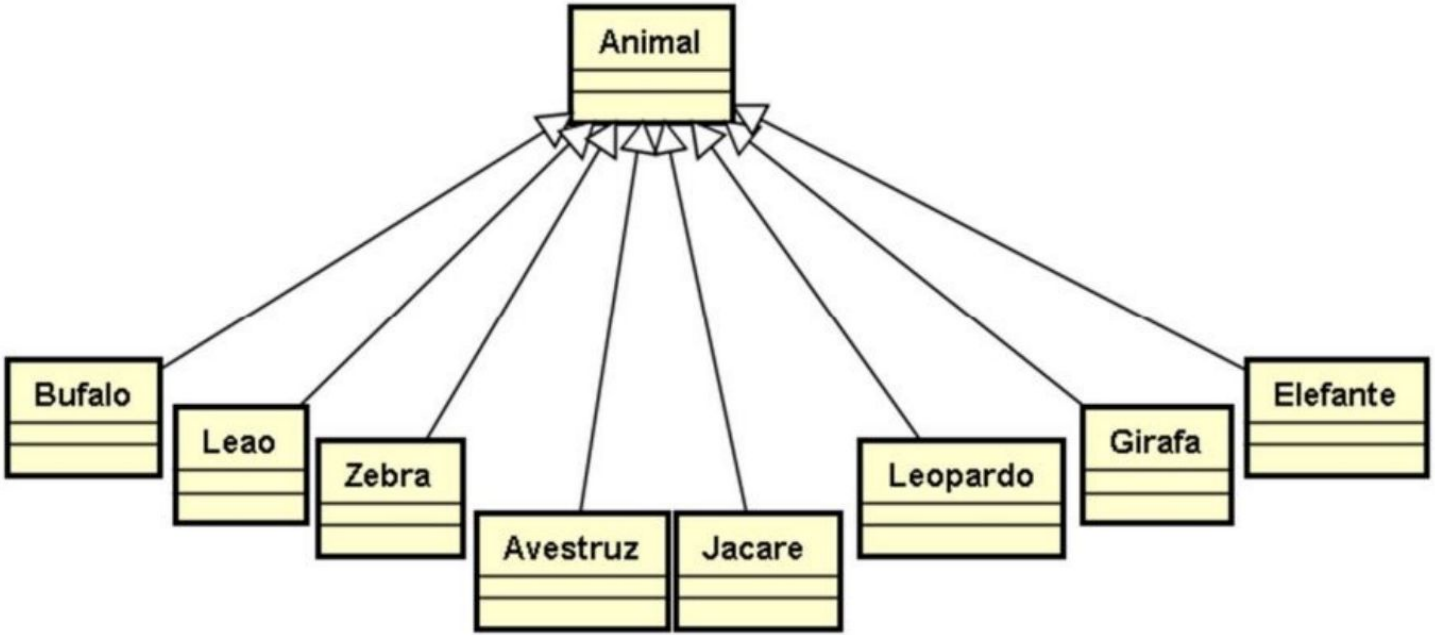
É por este motivo que todas as nossas classes já nascem com algum métodos como toString(), equals() entre outros.

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

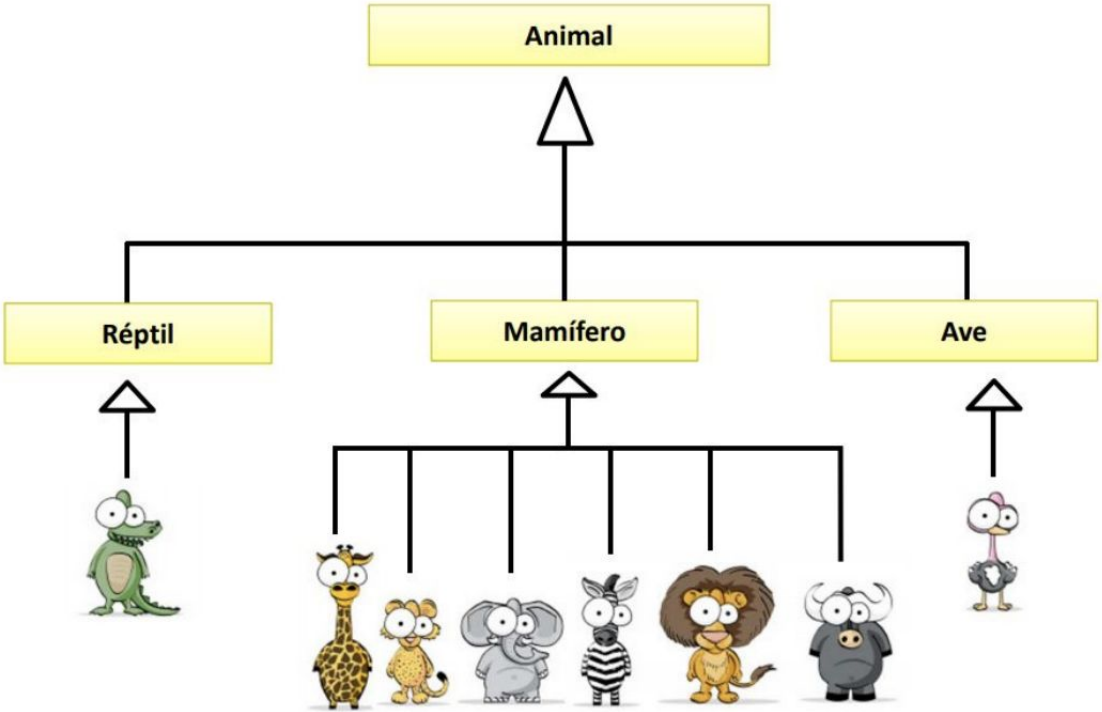
Herança - Exemplo



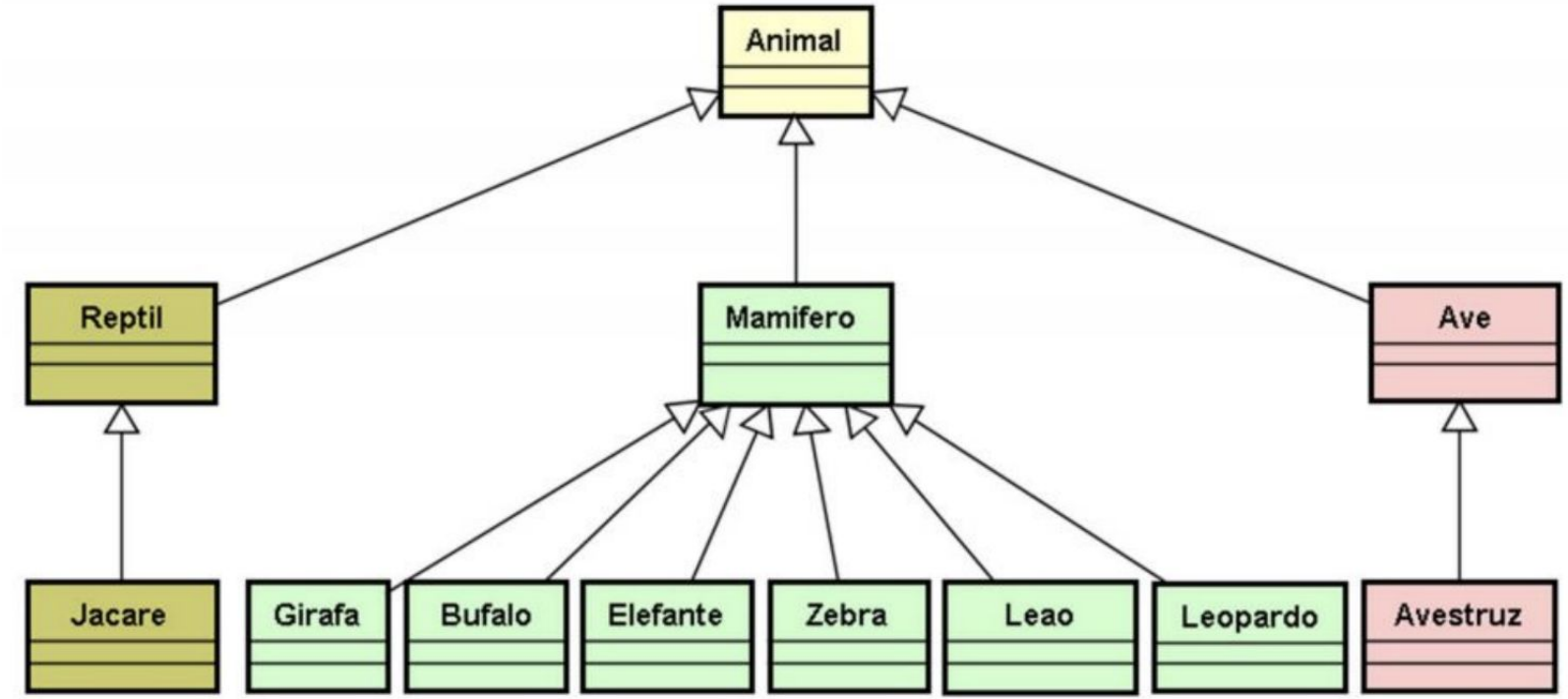
Herança - Exemplo



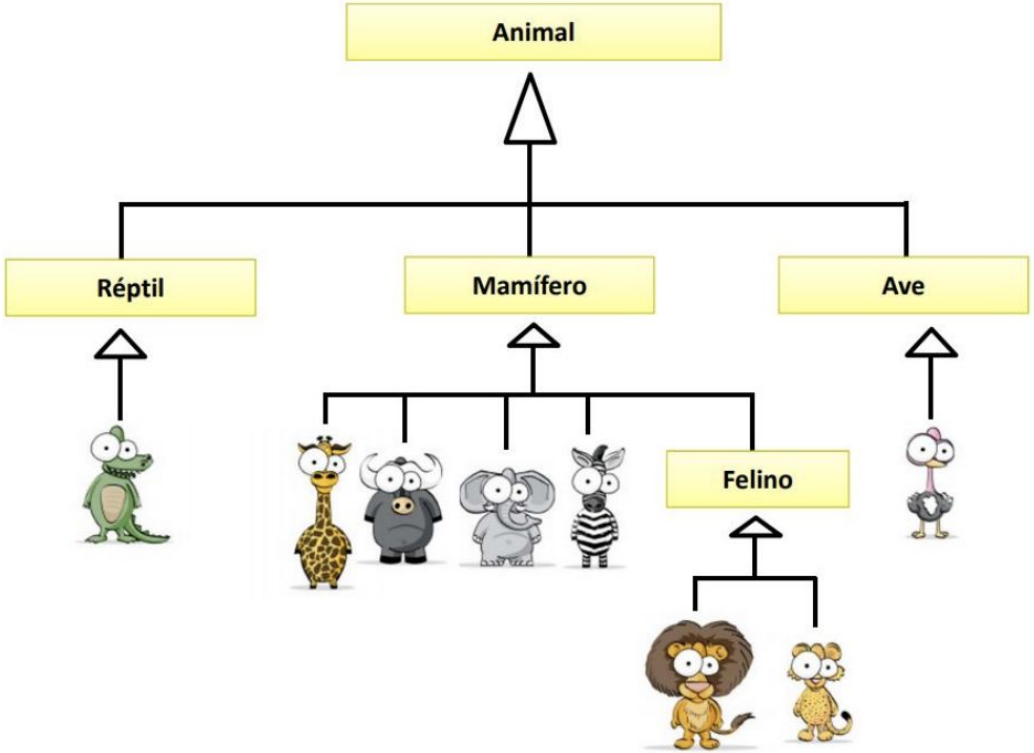
Herança - Exemplo



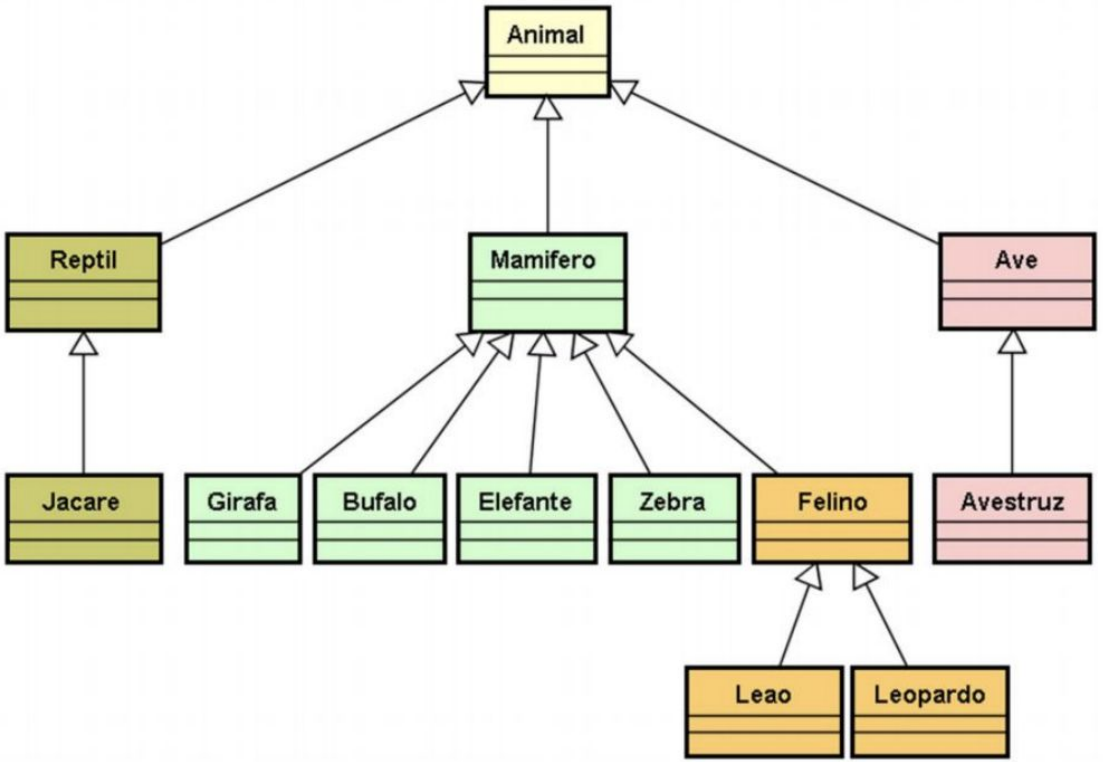
Herança - Exemplo



Herança - Exemplo



Herança - Exemplo



OBRIGADO

FIAP