# F | \ 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;
   }
}
```

# Pessoa - idade : int + setIdade(idade : int) : void + getIdade() : int



#### 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 <u>saldo</u>;
    public double getSaldo(){
        return this.saldo;
    public void depositar(double valor){
        this.saldo = saldo + valor;
    public void sacar(double valor){
        this.saldo = saldo - valor;
```



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.



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 <u>mais rígido</u> 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.



Forma de representação no UML

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

#### Celular

- fabricante : String
- ~ modelo : String
- # anoFabricao : int
- + versao : float
- tocar(): int
- ~ receberSms(): boolean
- # tocarDespertador(): void
- + exibirUltimaLigacao(): Ligacao



```
private String fabricante;
                                                     Celular
String modelo;
                                         - fabricante : String
protected int anoFabricacao;
                                         ~ modelo : String
                                         # anoFabricao : int
public float versao;
                                         + versao : float
                                         - tocar(): int
                                         ~ receberSms(): boolean
                                        # tocarDespertador(): void
                                         + exibirUltimaLigacao(): Ligacao
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 n\u00e3o recebe nenhum par\u00e1metro
- Implementar a interface Serializable



#### Java Beans

```
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;
```

#### Programador

- linguagem : String
- + getLinguagem(): String
- + setLinguagem(linguagem : String) : void



#### 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 Superclasse (classe pai) + sacar(valor : double) : void + depositar(valor : double) : void Relacionamento de Generalização (Herança) Poupanca Corrente Subclasse (classe filha) Descendentes





Carro - modelo : String fabricante : String CarroPasseio - motor : float CarroSolar velocidadeMaxima : int. - direcaoHidraulica : boolean quantidadeBaterias : int + getModelo(): String arCondicionado : boolean + setQuantidadeBaterias(quantidadeBaterias : int) : void + setModelo(modelo : String) : void + getQuantidadeBaterias() int + getArCondicionado() : boolean + getFabricante(): String + setArCondicionado(arCondicionado : boolean) : void + setFabricante(fabricante : String) : void + getDirecaoHidraulica() : boolean + getMotor(): float \* setDirecaoHidraulica(direcaoHidraulica : boolean) : void + setMotor(motor : float) : void + setVelocidadeMaxima(velocidadeMaxima : int) : void + getVelocidadeMaxima() : int + acelerar(intensidade : int) : void + brecar(intensidade : int) : void Formula1 StockCar - equipe : String - tipoCarroceria : String + setTipoCarroceria(tipoCarroceria : String) : void + setEquipe(equipe : String) : void Jeep + getTipoCarroceria(): String + getEquipe() : String - tipoTracao : int - capota : boolean + setTipoTracao(tipoTracao : int) : void + getTipoTracao(): int + setCapota(capota : boolean) : void + getCapota() : boolean



#### 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;
```

```
Carro
- modelo : String

    fabricante String

- motor : float

    velocidadeMaxima : int

+ getModelo(): String
+ setModelo(modelo : String) : void
+ getFabricante(): String
+ setFabricante(fabricante : String) : void
+ getMotor(): float
+ setMotor(motor : float) : void
+ setVelocidadeMaxima(velocidadeMaxima : int) : void
+ getVelocidadeMaxima(): int
+ acelerar(intensidade : int) : void
+ brecar(intensidade : int) : void
                          Formula1
            - equipe : String
            + setEquipe(equipe : String) : void
            + getEquipe() : String
```



#### Herança - Classe Object

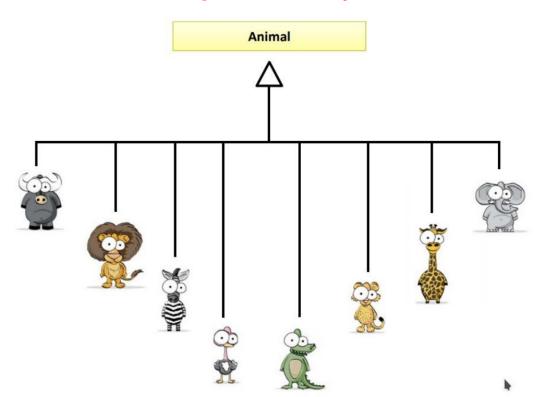
Todas as classes que criamos até hoje já são classes que herdam da super classe Object.

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.

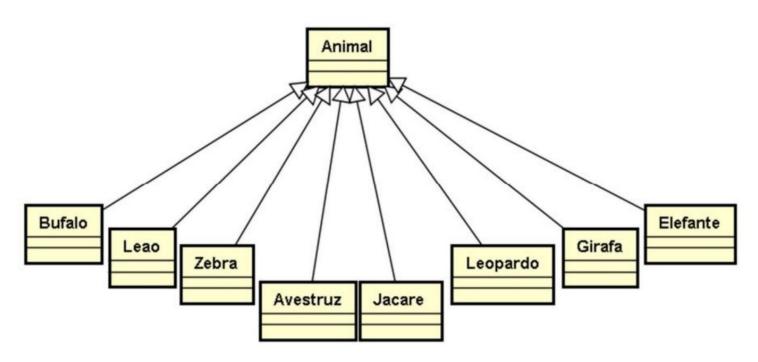
https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html



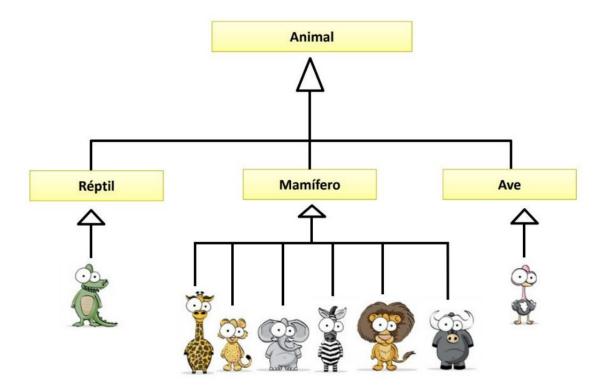




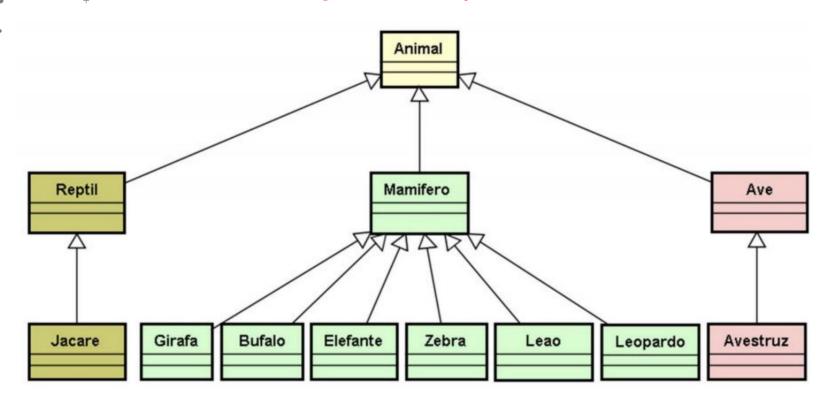
•



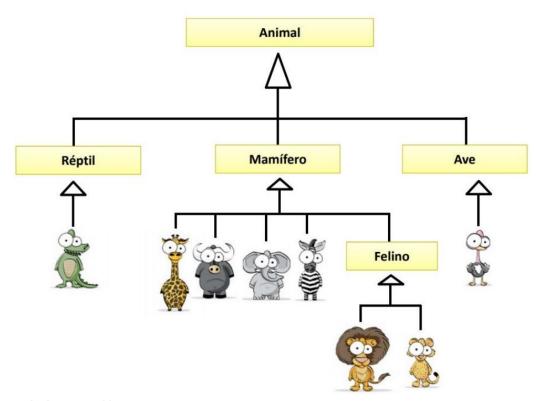




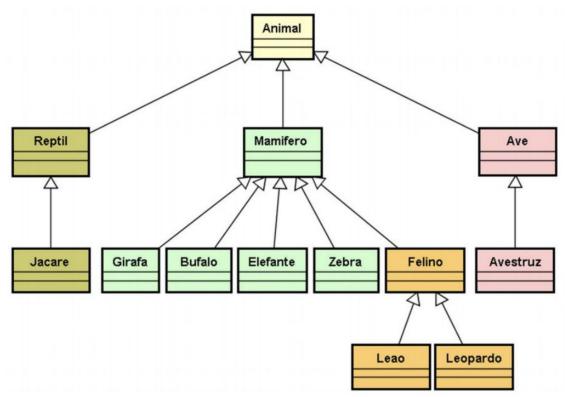












# **OBRIGADO**



Copyright © 2020 | Professor Fabio Tadashi Miyasato
Todos os direitos reservados. Reprodução ou divulgação total ou parcial deste documento, é expressamente
proibido sem consentimento formal, por escrito, do professor/autor.