



# **Estrutura de dados e Programação orientada a objetos**

Prof. Wendley S. Silva – [wendley@ufc.br](mailto:wendley@ufc.br)

# **Introdução ao paradigma de programação: Orientado a Objetos**

# Roteiro

- Paradigma de construção de software
- Orientação a Objetos
- Conceitos de Orientação a Objetos (OO)
- Classe, Objeto e Mensagem
- Os pilares da Orientação a Objetos (OO)
- Reuso de Implementação

# Conceitos da Orientação a Objetos

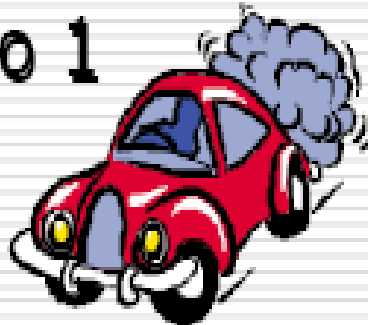
- Classe
- Objeto (Instância)
- Mensagem
- Encapsulamento
- Herança
- Polimorfismo
- ...

# Classe (1/3)

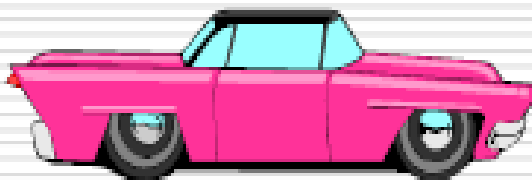
- A classe é a implementação de tipo abstrato de dados (TAD) no paradigma orientado a objetos.
- Uma classe Java é um molde para a criação de objetos. A classe define as propriedades (atributos) e os comportamentos (métodos).
- Além disso, uma classe Java define como produzir (instanciar) objetos a partir dela.

# Classe (2/3)

Objeto 1



Objeto 2



Classe  
Automovel

numeroPortas  
cor  
fabricante  
ano  
placa

# Classe (3/3)

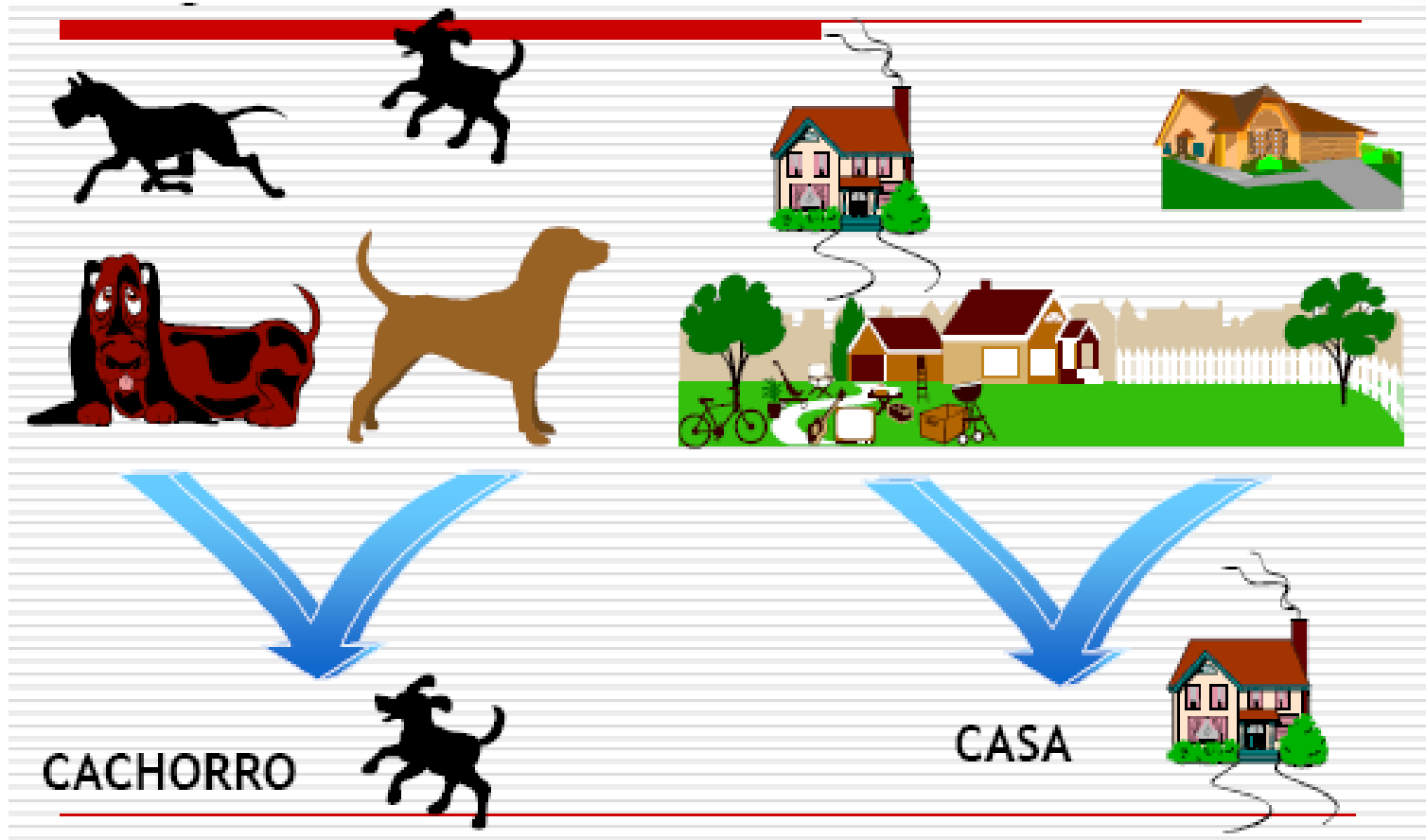


# Objeto (1/3)

- Um objeto é uma construção de software que encapsula **estado** e **comportamento**, através respectivamente de **propriedades** (atributos) e **operações** (métodos);
- **Estado de um Objeto**: composto por suas **propriedades** e seus respectivos **valores**;
- **Comportamento**: a maneira como o objeto reage quando o seu estado é **alterado** ou quando uma mensagem é **recebida**.



# Objeto (2/3)



## Objeto (3/3)

☐ Um objeto possui operações:



- ☐ Ligar;
- ☐ Desligar;
- ☐ Travar portas;
- ☐ Acelerar;
- ☐ Frear...

# Mensagens

- Mecanismo através do qual os objetos se **comunicam**, invocando as **operações desejadas**;
- Um objeto (**Emissor**) envia uma mensagem a outro (**Receptor**) que executará uma tarefa.

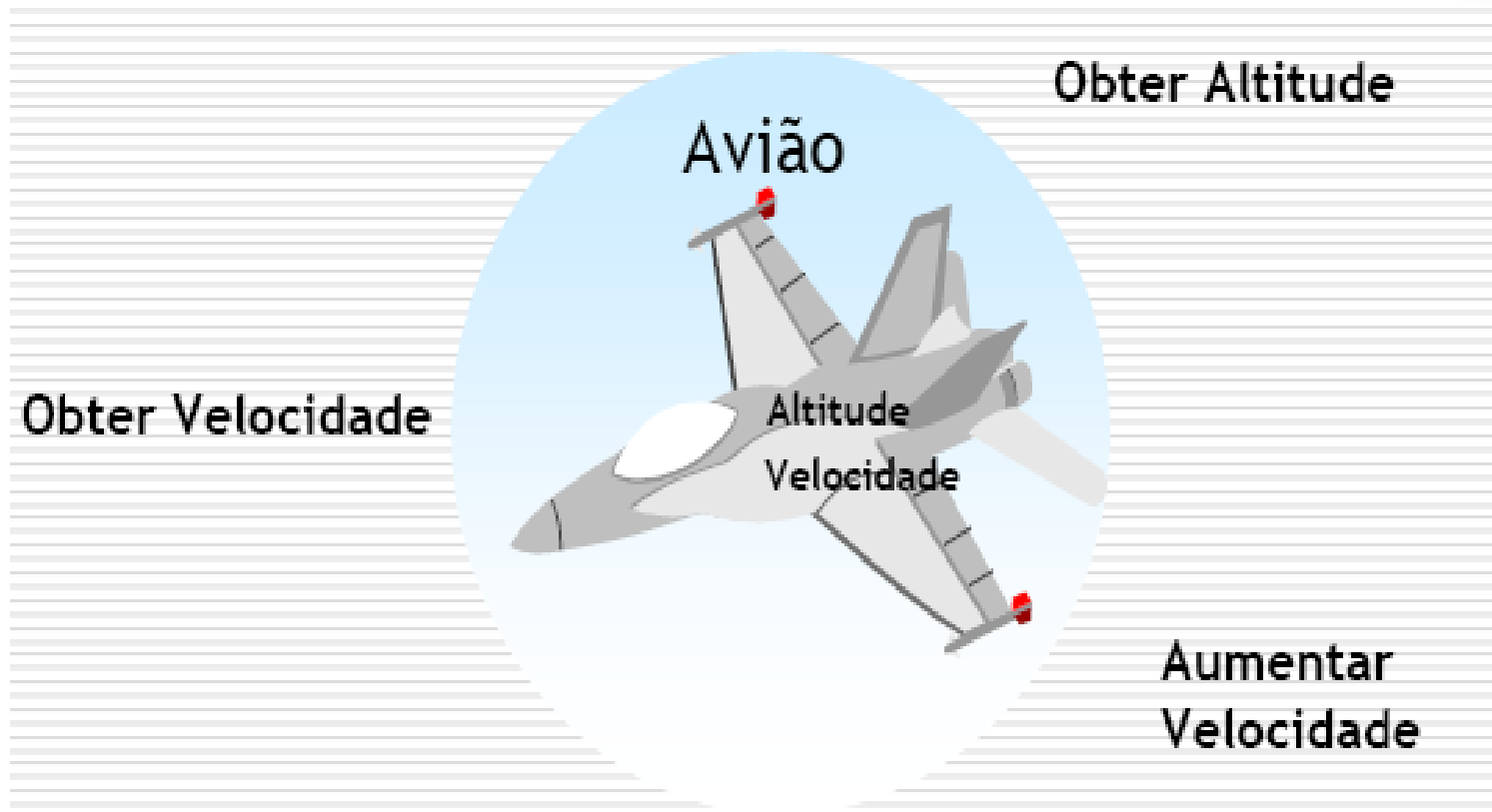
# Os pilares da OO

- Os pilares da OO são mecanismos fundamentais que garantem a filosofia de Orientação a Objetos. São eles:
  - Encapsulamento;
  - Herança;
  - Polimorfismo.

# Encapsulamento (1/2)

- **Resumindo:** “Não mostre as cartas de seu baralho”
- **Objetivos:**
  - Ocultar do mundo externo ao objeto os detalhes de implementação e restringir o acesso às propriedades e aos métodos;
  - Permitir a criação de programas com **menos erros e mais clareza**.
- **Vantagens:**
  - Segurança no acesso ao objeto;
  - Melhor consistência no estado interno, pois evita **alterações incorretas** nos valores das propriedades.

# Encapsulamento (2/2)

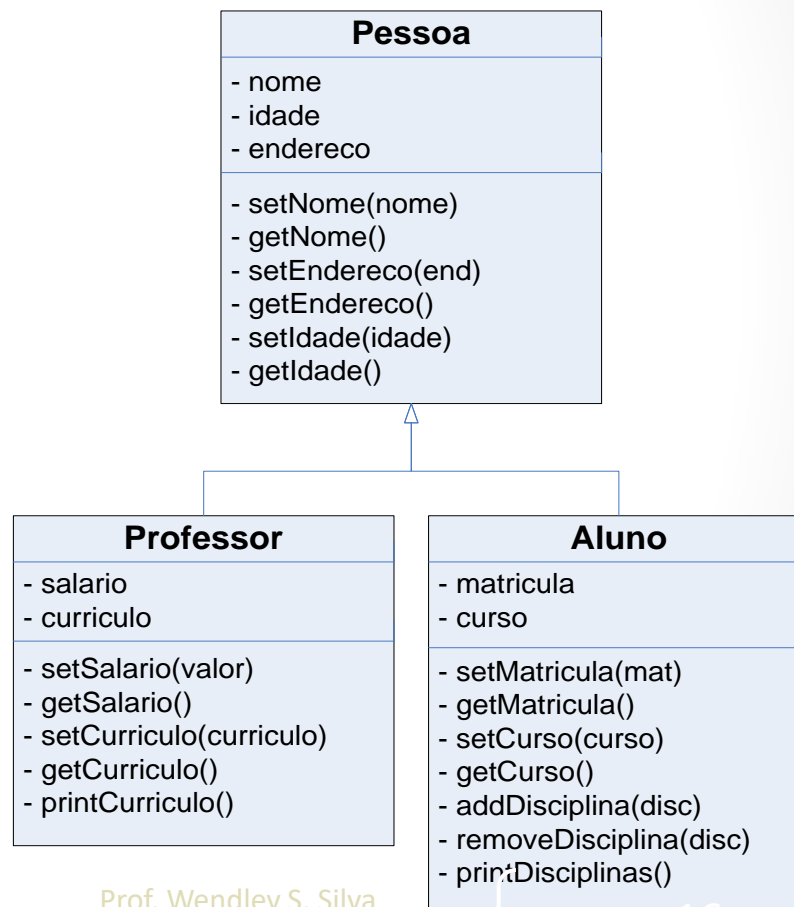


# Herança (1/3)

- **Resumindo:** “Filho de peixe, peixe é”.
- Permite definir **novas classes** (subclasses) a partir de uma classe já existente (superclasse).
- A subclasse herda as **propriedades comuns** da superclasse e pode ainda **adicionar novos métodos** ou **reescrever métodos herdados**.
- **Objetivo:** evitar que classes que possuam atributos ou métodos **semelhantes** sejam **repetidamente criados**.

# Herança (2/3)

- Pode ser: **Simples**





# Herança (3/3)

- Pode ser: **Múltipla**



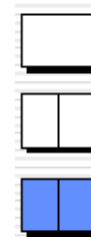
# Polimorfismo

- **Resumindo:** “Vamos nos adaptar”.
- Permite a um método ter **várias implementações** as quais são selecionadas com base na **quantidade de parâmetros** e **seus tipos** que é passado para a invocação do método.

Janela ( )

Janela ( 1 x 2 , 2 )

Janela ( 1 x 2 , 2, Azul )





# Prática

# Criar novo projeto

- Criar projeto **Exemplos**
- Criar nova classe **Contas**, com as seguintes variáveis:
  - Nº da conta (inteiro)
  - Nome do titular (String)
  - Valor do saldo (double)
- Criar método **depositar**

```
package exemplos;

public class Conta {

    int numero;
    String nome_titular;
    double saldo;

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

# Método sacar

```
boolean sacar(double valor) {  
    if (this.saldo >= valor) {  
        this.saldo -= valor;  
        return true;  
    }  
    else  
        return false;  
}
```

Figura 2.5: Método *sacar*

# Utilizando objetos

- **Declarar uma variável que referenciará o objeto:** assim como fazemos com tipos primitivos, é necessário declarar o objeto.
- **Instanciar o objeto:** alocar o objeto em memória. Para isso utilizamos o comando *new* e um construtor.

```
1 package exemplos;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6         Conta c; //Declarando a variável que conterà a r
7         c = new Conta();//Instanciando um objeto em memó
8         c.nome_titular = "Zé";
9         c.depositar(100);
10        System.out.println("Titular: "+ c.nome_titular);
11        System.out.println("Saldo Atual: "+ c.saldo);
12    }
13 }
```

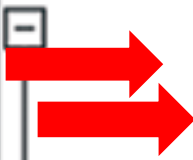


Figura 2.6: Criação de instâncias da classe *Conta*



# Usando o método sacar

```
1 package exemplos;
2
3 public class Programa {
4
5     public static void main(String[] args) {
6
7         Conta c = new Conta();
8         c.depositar(200);
9         boolean saque_efetuado = c.sacar(250);
10        if (saque_efetuado)
11            System.out.println("Saque Efetuado com Sucesso");
12        else
13            System.out.println("Saque não efetuado! Saldo insuficiente!");
14    }
15 }
```

Figura 2.7: Utilização do valor retornado por um método

# Criar os métodos

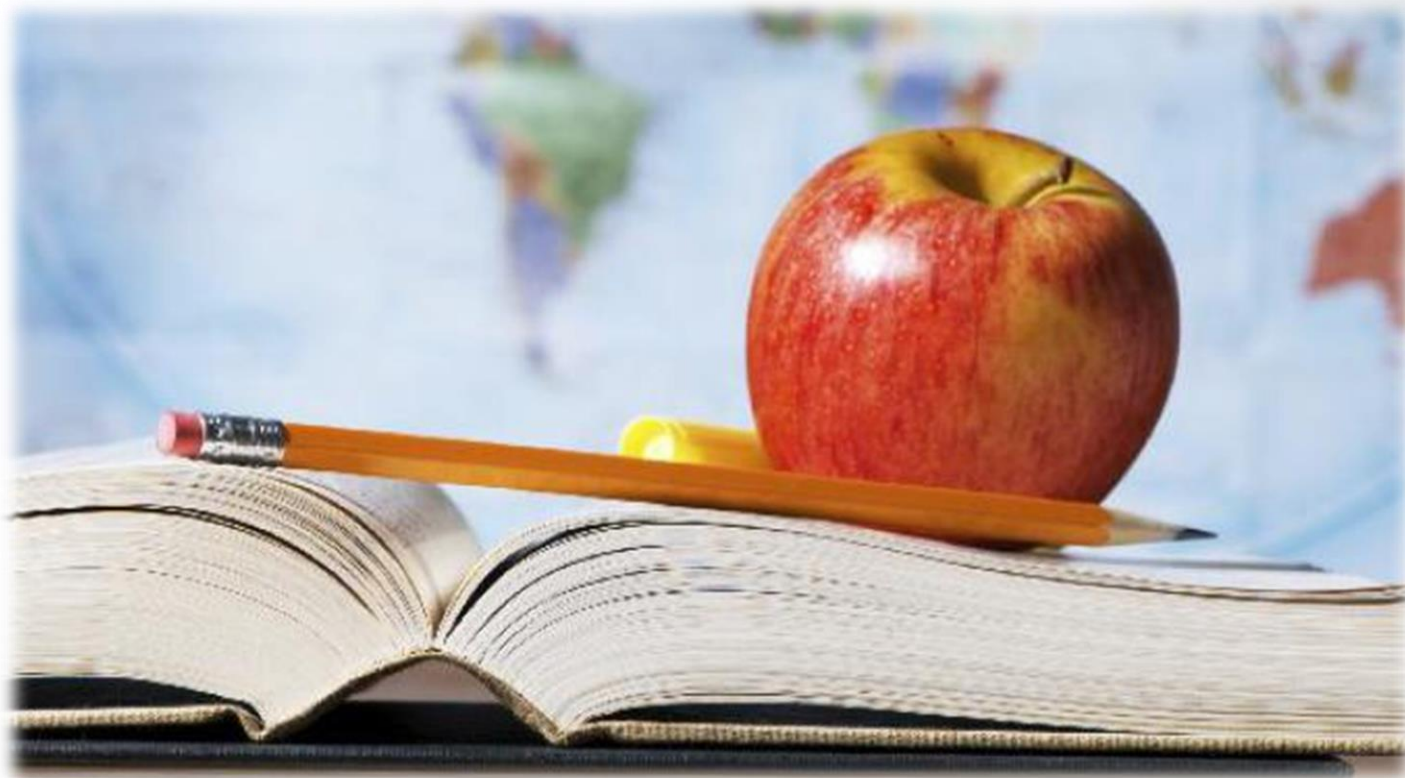
- **getSaldo()**
- **getNumero()**
- **getNomeTitular()**

# Polimorfismo

```
public Conta(int numero, String nome_titular, double saldo){
    this.numero = numero;
    this.nome_titular=nome_titular;
    this.saldo = saldo;
}

public Conta(int numero, String nome_titular){
    this.numero = numero;
    this.nome_titular=nome_titular;
    saldo = 0;
}
```

Figura 3.1: Métodos construtores para a classe *Conta*



# Exercício

# Exercício: Classe Carro

- **Atributos:**
  - Modelo
  - Preço
  - Cor
- **Métodos:**
  - getModelo()
  - getPreco()
  - getCor()
- Criar uma classe **Aplicacao** que adicione alguns modelos de carros e apresente na tela

```
public class Carro{

    private String cor;
    private double preco;
    private String modelo;

    /* CONSTRUTOR PADRÃO */
    public Carro(){
    }

    /* CONSTRUTOR COM 2 PARÂMETROS */
    public Carro(String modelo, double preco){
        //Se for escolhido o construtor sem a COR do veículo definimos a PRETA
        this.cor = "PRETA";
        this.modelo = modelo;
        this.preco = preco;
    }

    /* CONSTRUTOR COM 3 PARÂMETROS */
    public Carro(String cor, String modelo, double preco){
        this.cor = cor;
        this.modelo = modelo;
        this.preco = preco;
    }
}
```

```
public class Aplicacao {  
  
    public static void main(String[] args) {  
        //Construtor sem parâmetros  
        Carro prototipoDeCarro = new Carro();  
  
        //Construtor com 2 parâmetros  
        Carro celtaPreto = new Carro("Celta", "30000");  
  
        //Construtor com 3 parâmetros  
        Carro golfAmarelo = new Carro("Amarelo", "Golf",  
"70000");  
    }  
}
```