

# Classes x Objetos



**Conteudista:** Prof. Esp. Alexander Gobbato Albuquerque

**Revisão Textual:** Prof.ª M.ª Rosemary Toffoli

## Objetivos da Unidade:

- Aprender o que é classe e o que é objeto;
- Entender como trazer o problema do mundo real para o mundo da orientação a objetos;
- Compreender o conceito de atributos, “métodos de acesso”, “parâmetros” e outros.

Contextualização

Material Teórico

Material Complementar

Referências



# Contextualização

---

Vimos na unidade anterior como é a estrutura de um programa orientado a objetos e de um programa estruturado. Agora vamos apresentar como criar os métodos, usar encapsulamento e chamar os métodos criados através de objetos.

# Material Teórico

## Classes x Objetos

### Mecanismo de Trabalho

A JVM interpreta os *bytecodes* gerados pelo compilador.

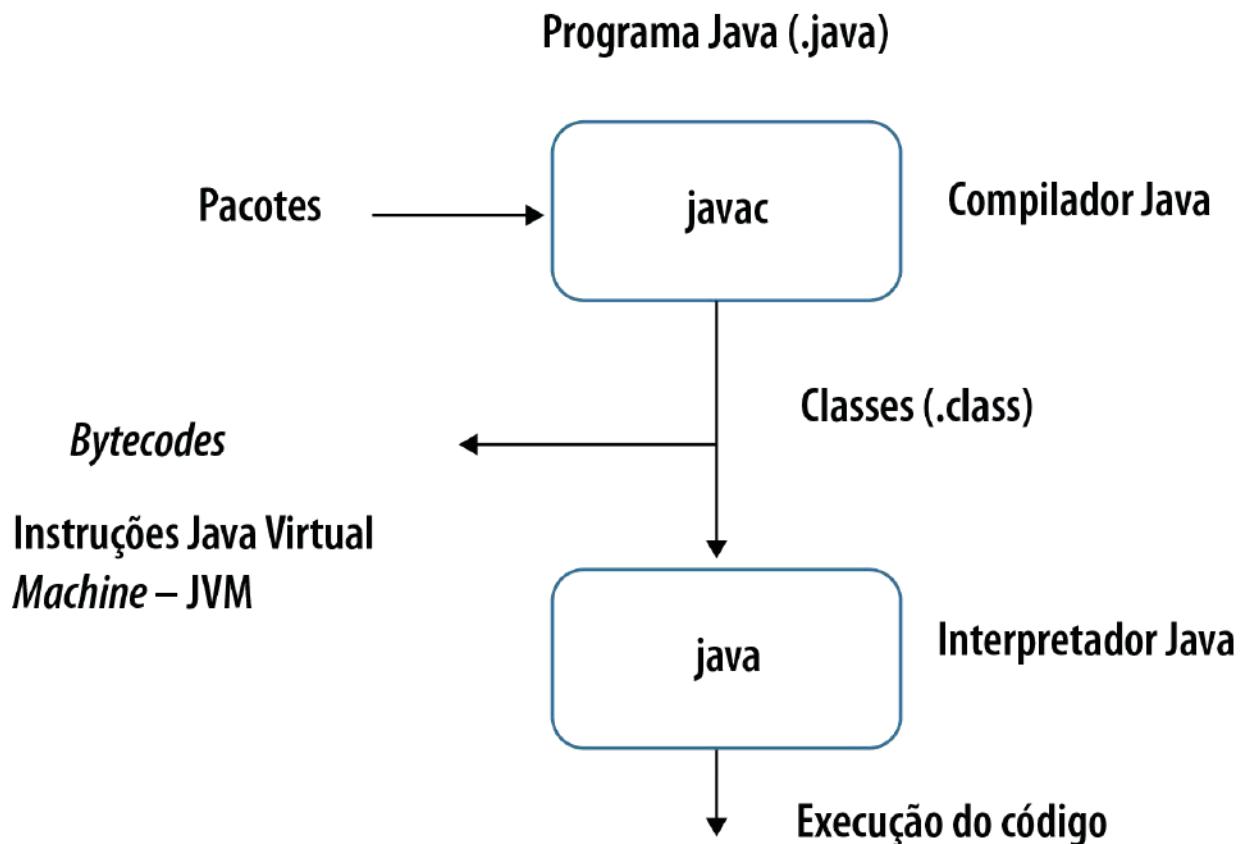


Figura 1

O objetivo da JVM é permitir que qualquer sistema operacional possa executar uma aplicação Java

## Classe Java

```
public class BemVindo {  
    public static void main(String args[]) {  
        System.out.println("Bem vindo a Java.");  
    }  
}
```

---

### Importante!

Lembre-se o Java é *case sensitive* (há diferença entre maiúsculas e minúsculas).

## Programa = Classe

```
public class BemVindo {  
    public static void main(String args[]){  
        System.out.println("Bem vindo"); /* esta linha impirme uma mensagem na tela*/  
    }  
}
```

---

## **Importante!**

**Os arquivos Java têm que ter o nome da classe principal.**

No exemplo acima, o nome do arquivo seria Benvindo.java e depois de compilado ficaria Benvindo.class

## **Nome da classe = Nome dos Arquivos**

```
public class BemVindo {  
    public static void main(String[ ] args){  
        System.out.println("Benvindo");  
    }  
}
```

## **Método main**

```
public class BemVindo {  
    public static void main(String[ ] args){  
        System.out.println("Benvindo");  
    }  
}
```

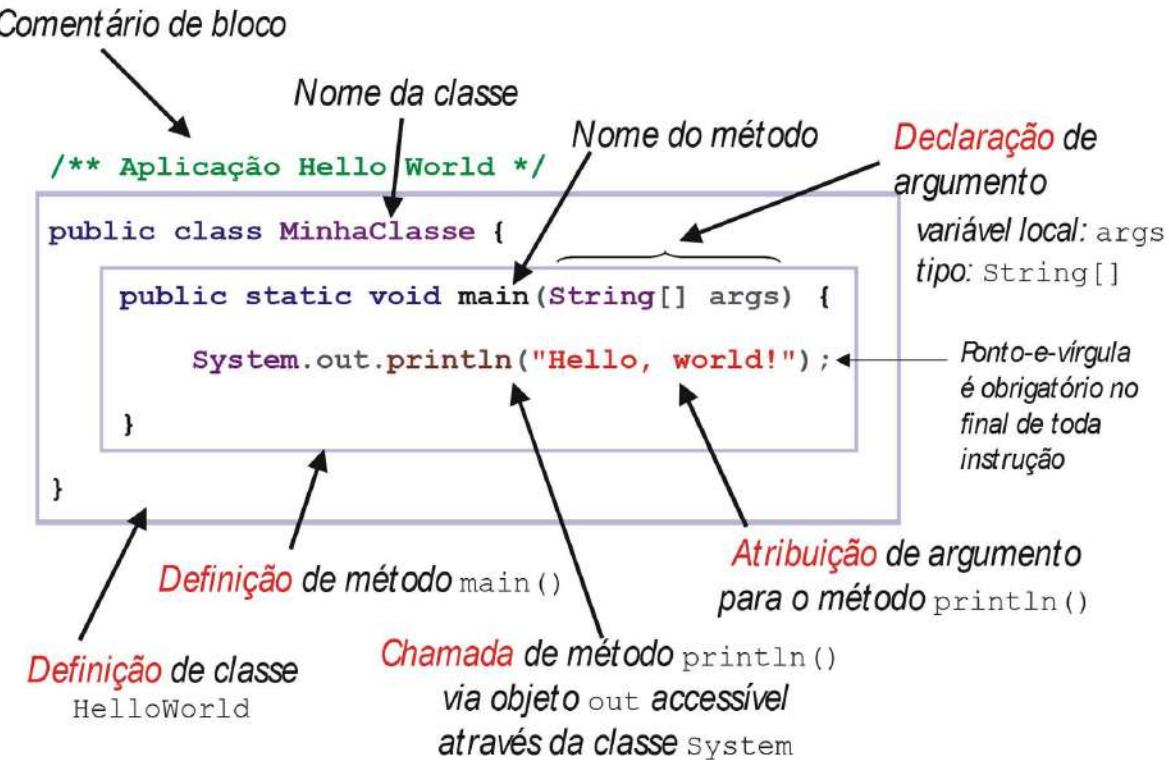


Figura 2

## Principais Conceitos

### O que são Objetos?

São quaisquer coisas na natureza que possuam propriedades (características) e comportamentos (operações).

Exemplos de Objetos: um bolo, um cachorro, um livro, etc.

### Orientação a Objetos

O termo orientação a objetos significa organizar o mundo real como uma coleção de objetos que incorporam estrutura de dados e um conjunto de operações que manipulam estes dados, como

exemplo podemos montar os seguintes objetos pessoas e pássaros.

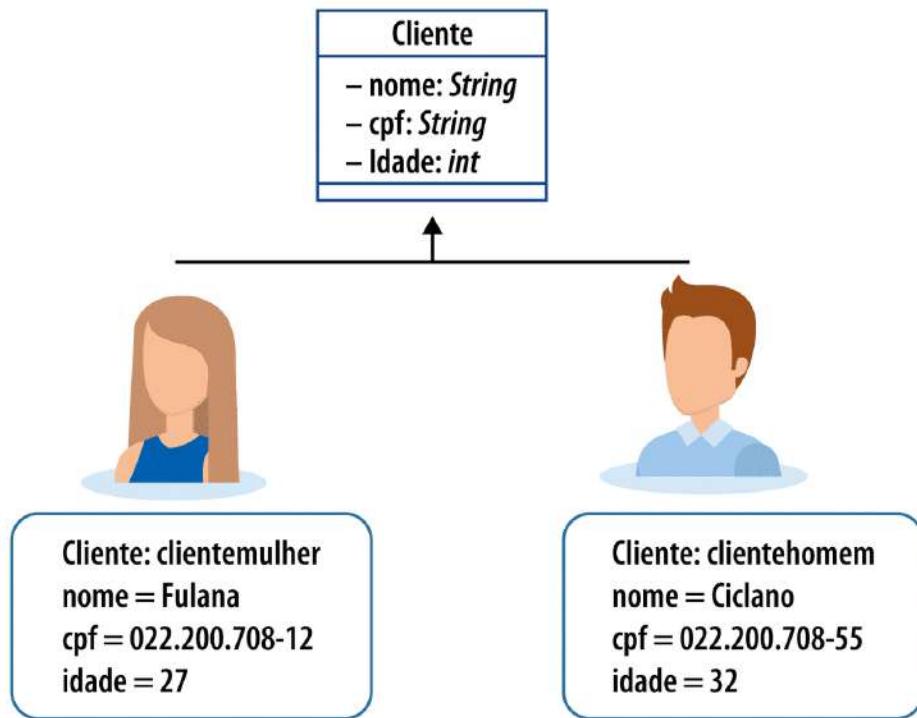
Tabela 1 – Classe: Pessoa

Propriedades	Comportamento
Nome	Andar
Profissão	Correr
Data de Nascimento	Trabalhar
Altura	Chorar
Peso	Dançar
Espécie	Andar
Cor das penas	Correr
Tamanho	Voar
Peso	Pousar

## Estrutura de um Objeto

Um objeto tem identificação, dados e comportamento, atributos e métodos.

Podemos dizer que objeto é espécie da classe, ou seja, uma instância da classe.



**Figura 3**

Fonte: Adaptada de Freepik

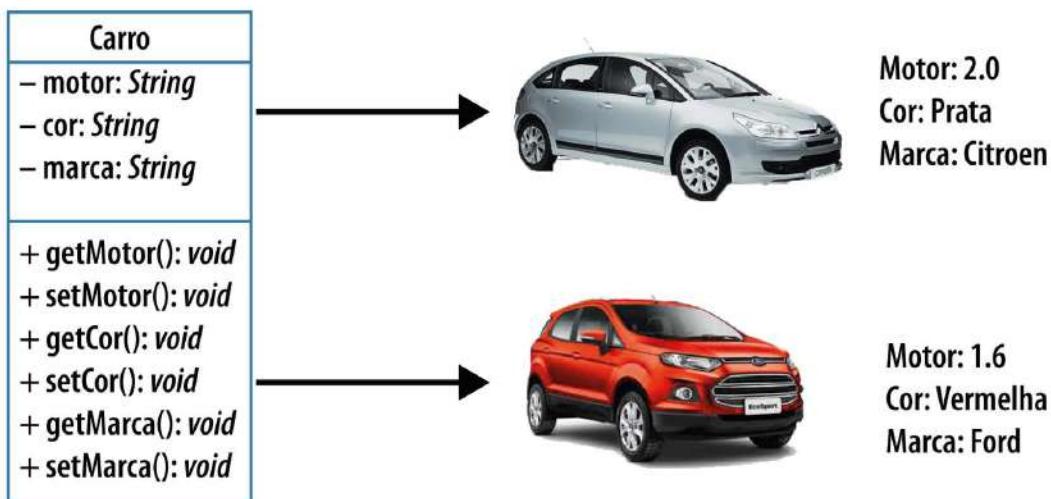
Para finalizarmos as comparações entre Classe e Objetos temos os seguintes exemplos:

**Tabela 2**

Classes	Objetos
Funcionário	Os funcionários Pitoco Aguiar e José da Silva
Carro de passeio	O astra, vinho, placa CIL 8445

Classes	Objetos
Nota Fiscal	A nota fiscal #29470 referente a um computador <i>notebook</i>

Em resumo, o objeto é definido pelo conjunto de valores dos seus atributos em determinado instante. O comportamento é definido pelo conjunto de métodos. Ao criar uma classe, ela passa a ser um tipo abstrato de dados que pode instanciar objetos. Para instanciar objetos usamos o operador *new*.



**Figura 4**

Fonte: Adaptado de Freepik

Uma coleção de carros pode ser representada por uma classe chamada Carro. Cada carro desta coleção é a “instância” (objeto) da classe Carro.

# Atributos

Os atributos são as características dos objetos, o estado do objeto equivale aos valores de todos os atributos. No caso do exemplo acima os atributos são representado pelo motor, cor e marca.

## Construindo a Classe em Java

Um arquivo em *Java* precisa de uma classe pública com o mesmo nome do arquivo, abaixo um exemplo de como criar o arquivo.

```
public class <nome_da_classe> {  
    <lista de atributos>  
    <lista de métodos>  
}
```

---

## Importante!

Por padrão as classes e atributo em java devem seguir algumas regras:

- Somente letras, números e *underline*;
- Não pode começar por números;
- Não pode ter espaços em branco;
- Não pode ser palavra reservada;
- **Classes:** Possuem as iniciais maiúsculas e sem espaços ou caracteres especiais;

- **Atributos:** São sempre em letras minúsculas.

---

## Tipos de Dados em Java

Java possui 8 tipos primitivos que podem ser usados como tipos de atributos e que são eles:

- **Byte:** 8 bits;
- **Short:** 16 bits;
- **Int:** 32 bits;
- **Long:** 64 bits;
- **Float:** ponto flutuante de 32 bits;
- **Double:** ponto flutuante de 64 bits;
- **Char:** Unicode de 16 bits;
- **Boolean:** true/false;
- **String:** classe.

Vejamos um exemplo de criação da classe carro e a instância da classe.



```

public class Carro {
    String motor;
    String cor;
    String marca;
}

```

```

public class TestarCarro{
    public static void main(String args[]){
        Carro c1 = new Carro();
        Carro c2 = new Carro();

        c1.motor = '2.0';
        c1.cor = 'Prata'
        c1.marca = 'Citroen';

        c1.motor = '1.6';
        c1.cor = 'Vermelho'
        c1.marca = 'Ford';
    }
}

```

**Figura 5**

Fonte: Reprodução

## Classes: Atributos (Propriedades) + Métodos (Comportamento)

Método é a implementação de uma operação. Os métodos expressam os comportamentos que todos os objetos dessa classe possuem.

```

tipo nome (parâmetros) {
    instruções;
}

```

```
<return resposta>;  
}
```

## Set e Get

Servem como métodos de leitura/escrita aos atributos de classes.

Um método de leitura para um atributo deve ser chamado de *getXxx* (onde *Xxx* é o nome do atributo). Este método não recebe nada como parâmetro, e retorna o mesmo tipo do atributo.

Já um método de gravação deve ser chamado *setXxx*, não retorna nada (geralmente), e recebe como parâmetro o valor que deve ser armazenado no atributo.

Na programação orientada a objetos, os atributos da classe quase nunca estão visíveis para os usuários, é necessário criar um método público para que a aplicação possa acessá-lo. Temos basicamente 3 modificadores de acesso:

- **Private ( - )**: é o mais restritivo de todos, atributos e métodos com esse modificador são visíveis somente dentro da definição da própria classe, acessando-o diretamente ou através de uma instância da mesma classe;
- **Protected ( # )**: define que atributos e métodos somente podem ser acessados por subclasses da classe onde está sendo definido;
- **Public ( + )**: é o mais abrangente de todos os tipos de acesso, declara que elementos que o utilizam são acessíveis de qualquer classe Java.

Vejamos a implementação do código:

```
public class Carro {  
    private String motor;  
    private String cor;  
    private String marca;  
  
    public String getMotor(){  
        return motor;  
    }  
  
    public void setMotor(String m){  
        motor = m;  
    }  
  
    public String getCor(){  
        return cor;  
    }  
  
    public void setCor(String c){  
        cor = c;  
    }  
  
    public String getMarca(){  
        return marca;  
    }  
  
    public void setMarca(String mc){  
        marca = mc;  
    }  
}
```

## **Figura 6**

Fonte: Reprodução

---

Agora os atributos da classe Carro são privados, não é mais possível atribuir informações diretamente:

```
public class TestarCarro{
    public static void main(String args[]){
        Carro c1 = new Carro();
        Carro c2 = new Carro();

        c1.motor = '2.0';
        c1.cor = 'Prata';
        c1.marca = 'Citroen';

        c1.motor = '1.6';
        c1.cor = 'Vermelho'
        c1.marca = 'Ford';
    }
}

public class TestarCarro{
    public static void main(String args[]){
        Carro c1 = new Carro();
        Carro c2 = new Carro();

        c1.setMotor('2.0')
        c1.setCor('Prata')
        c1.setMarca('Citroen');

        c1.setMotor('1.6')
        c1.setCor('Vermelho')
        c1.setMarca('Ford');
    }
}
```

**Figura 7**

Fonte: Reprodução

---

Os métodos também possuem padrão de nomenclatura. Os métodos possuem sempre a primeira inicial minúscula e as outras iniciais maiúsculas.

pegarInformacao(), executarComandoInicial()

Os métodos geralmente são ações que podem ser efetuadas entre os atributos do objeto.

Métodos que não retornam valores, apenas executam ações, são do tipo *void*.

Por exemplo, na classe Carro, podemos criar um método que imprima seus atributos na tela, ou como vimos anteriormente, mostre o estado do objeto.

```
public class Carro {  
    private String motor;  
    private String cor;  
    private String marca;  
  
    public String getMotor() ..  
    public void setMotor(String m) ..  
    public String getCor() ..  
    public void setCor(String c) ..  
    public String getMarca() ..  
    public void setMarca(String mc) ..  
  
    public void imprimeDados(){  
        System.out.println("Motor: " + motor);  
        System.out.println("Cor: " + cor);  
        System.out.println("Marca: " + marca);  
    }  
}
```

**Figura 8**

Fonte: Reprodução

---

## Métodos – Procedimentos

Os métodos podem ou não assumir tipos de dados, caso não assumam, são chamados de procedimentos, pois executam um conjunto de instruções sem devolverem valor algum a quem os chamou. Um método sem tipo recebe em sua definição a palavra-chave void no lugar do tipo.

```
//Procedimento sem parâmetro
void frase() {
    System.out.println("Procedimento sem parâmetros");
}
```

**Figura 9**

Fonte: Reprodução

---

## Métodos – Funções

Quando os métodos assumem algum tipo, eles são chamados de funções e precisam do comando return para devolver o valor resultante da execução de suas instruções internas.

```
//Função sem parâmetro
String frase() {
    String mensagem = "Função sem parâmetro";
    return mensagem;
}
```

**Figura 10**

Fonte: Reprodução

---

## Métodos – Parâmetros

Os métodos podem receber dados para serem utilizados internamente, os quais são chamados de parâmetros ou de argumentos.

Quando os parâmetros são passados para os métodos, é criada uma cópia dos valores.

Podemos passar vários parâmetros para os métodos, inclusive de tipos diferentes.

```
//Procedimento com parâmetro          //Função com parâmetro
void numero (int n) {                  int soma(int num1, int num2) {
    int resposta;                      int resul;
    resposta = n * 5;                  resul = n + m;
    System.out.println(resposta);      return resul ;
}
}
```

**Figura 11**

Fonte: Reprodução

---

## Material Complementar

---

Indicações para saber mais sobre os assuntos abordados nesta Unidade:

---

### Livros

#### **Aprenda Programação Orientada a Objetos em 21 dias**

SINTES A. *Aprenda programação orientada a objetos em 21 dias*. Ed. Pearson; 1. ed. 2002.

#### **Java como Programar**

DEITEL P.; DEITEL H. *Java como Programar*. Ed: Pearson Universidades; 8. ed. 2009.

#### **Core Java**

HORSTMAN C.; CORNELL H. *Core Java*. Ed: Pearson Universidades; 8. ed. 2009.

---

### Leitura

## *Lesson: Object-Oriented Programming Concepts*

Clique no botão para conferir o conteúdo.

ACESSE

## Referências

---

SINTES, T. (2002) **Aprenda Programação Orientada a Objetos em 21 dias.** 1 ed. São Paulo: Pearson Education do Brasil, 2002, v. 1.

DEITEL, P.; DEITEL, H. (2010) **Java Como Programar**, 8 ed. São Paulo: Pearson Education do Brasil, 2010.

HORSTMANN, C.S.; CORNELL, G. (2010) **Core Java**. 8 ed. São Paulo: Pearson Education do Brasil, 2010, v. 1.