



# Introdução a Java

Programação Orientada a Objetos – Francisco Molina



# Por que aprender Java hoje?

Java não é só uma linguagem, é um ecossistema.

Java continua sendo uma das linguagens mais populares e demandadas no mercado global.

**+ 90% das empresas Fortune 500 usam Java**

Em 2025, Java está na **3ª posição** entre as linguagens mais requisitadas por recrutadores.

Onde o Java brilha:

- Aplicações escaláveis para grandes corporações
- Infraestrutura robusta que suporta tráfego massivo
- Soluções que exigem alto nível de segurança e confiabilidade
- Sistemas que precisam evoluir e se adaptar ao longo do tempo



## Backend & Cloud

Microserviços e sistemas robustos para milhões de usuários

Ex: Netflix, Uber



## Android

Linguagem oficial para o sistema mobile mais usado no mundo

+ 2.5 bilhões de dispositivos



## Big Data

Ferramentas essenciais como Hadoop, Spark e Kafka são em Java

Processamento de dados em larga escala



## Desktop & Científica

Aplicações robustas para análises complexas e simulações

Usado em finanças e pesquisa

# O que é Java? A Grande Ideia

"Write Once, Run Anywhere"

Java é uma linguagem de programação **Orientada a Objetos**, desenvolvida pela Sun Microsystems em 1995 e hoje mantida pela Oracle.

Sua principal característica é a **independência de plataforma**, permitindo que aplicações Java funcionem em qualquer dispositivo ou sistema operacional.

Como isso é possível? Graças à **JVM (Java Virtual Machine)**:

- Um "computador virtual" presente em cada plataforma
- Interpreta o bytecode Java, independente do hardware
- Gerencia memória e garante segurança
- Isola o código das peculiaridades do sistema operacional

Como funciona o processo de compilação e execução:



A mesma aplicação Java roda em:



# Seu Primeiro Programa: "Olá, Mundo!"

Vamos codificar!

```
// Todo programa Java vive dentro de uma "classe"
public class OlaMundo {
    // O "main" é o ponto de partida do seu programa
    public static void main(String[] args) {
        // Comando para imprimir texto no console
        System.out.println("Olá, Mundo Java!");
    }
}
```

▶ Executar



Mostrar o código primeiro cria uma conexão prática antes de mergulhar na teoria da sintaxe.

## Classe (`public class OlaMundo`)

Em Java, todo código precisa estar dentro de uma classe. A classe é o modelo/estrutura básica do seu programa.

## Método Main (`public static void main`)

É o ponto de entrada do seu programa - onde a execução começa. Sem ele, a JVM não sabe por onde começar.

## Saída de Dados (`System.out.println`)

Imprime texto no console. É como o programa "fala" com o usuário, mostrando informações na tela.

Resultado no console:

Olá, Mundo Java!

# Os Blocos de Construção: Variáveis e Tipos

Armazenando Informações

## Tipos Primitivos em Java

byte	-128 a 127	short	-32768 a 32767
int	$-2^{31}$ a $2^{31}-1$	long	$-2^{63}$ a $2^{63}-1$
float	$\pm 3.4e^{-38}$ a $\pm 3.4e^{38}$	double	$\pm 1.7e^{-308}$ a $\pm 1.7e^{308}$
char	0 a 65535	boolean	true ou false

E outros tipos não-primitivos:

String

Arrays

Classes

Interfaces

## A Inferência de Tipo com

var

Desde o **Java 10**, podemos usar o `var` para simplificar declarações, deixando o compilador inferir o tipo.

Antes do Java 10



Com var (Java 10+)

Exemplos de uso:

- Variáveis locais em métodos e blocos
- Inicializações com valores literais Expressões
- com retorno de tipo específico Loops e estruturas de iteração

# Os Blocos de Construção: Variáveis e Tipos

## O jeito clássico


```
// Declaração tradicional de variáveis
String mensagem = "Isso é uma String";
int numero = 10;
double valor = 3.14;
boolean ativo = true;

// Para collections
List<String> nomes = new ArrayList<>();
```

## Com 'var' (Java 10+)

```
// O compilador infere o tipo automaticamente
var mensagem = "Isso é uma String";
var numero = 10; // inferido como int
var valor = 3.14; // inferido como double
var ativo = true; // inferido como boolean

// Para collections
var nomes = new ArrayList<String>();
```

 **Dica:** Use `var` para tornar seu código mais limpo, mas sem sacrificar a legibilidade. O tipo ainda é estático e verificado em tempo de compilação.

# Operadores: Fazendo as Coisas Acontecerem

Matemática, Comparações e Lógica no Java

## Matemáticos



Adição

int soma = 10 + 5; // 15



Subtração

int dif = 10 - 5; // 5



Multiplicação

int prod = 10 \* 5; // 50



Divisão

int div = 10 / 5; // 2



Resto (Módulo)

int resto = 10 % 3; // 1

## = Relacionais



Maior que

boolean maior = 10 > 5; // true



Menor que

boolean menor = 10 < 5; // false



Igual a

boolean igual = 10 == 10; // true



Diferente de

boolean dif = 10 != 5; // true



Maior ou igual

boolean maiorIg = 10 >= 10; // true

## Lógicos



AND (E)

boolean and = true && true; // true

Ambas as condições precisam ser verdadeiras



OR (OU)

boolean or = true || false; // true

Pelo menos uma condição precisa ser verdadeira



NOT (NÃO)

boolean not = !false; // true

Inverte o valor da expressão

# Operadores: Fazendo as Coisas Acontecerem

## Exemplo Prático: Calculando Aprovação

```
// Notas de um aluno
double nota1 = 7.5;
double nota2 = 6.8;
double media = (nota1 + nota2) / 2;

// Verificando aprovação (média precisa ser >= 7.0)
boolean aprovado = media >= 7.0;
boolean recuperacao = !aprovado && media >= 5.0;

System.out.println("Média: " + media); // Média: 7.15
System.out.println("Aprovado? " + aprovado); // Aprovado? true
```



# Controlando o Fluxo do Programa

Tomando decisões e repetindo tarefas com estruturas de controle

## Estruturas de Decisão

```
// Estrutura if/else
double nota = 7.5;

if (nota >= 7.0) {
    System.out.println("Aprovado!");
} else if (nota >= 5.0) {
    System.out.println("Recuperação");
} else {
    System.out.println("Reprovado");
}
```

## Estruturas de Repetição

```
// Loop for
for (int i = 0; i < 5; i++) {
    System.out.println("Iteração: " + i);
}

// Loop while
int contador = 0;
while (contador < 3) {
    System.out.println("Contagem: " + contador);
    contador++;
}
```

# Controlando o Fluxo do Programa

## ⇌ Switch Expressions

Switch Tradicional:

```
char nota = 'B';
switch (nota) {
    case 'A':
        System.out.println("Excelente!");
        break;
    case 'B':
        System.out.println("Bom");
        break;
    default:
        System.out.println("Pode melhorar");
}
```

## ★ Switch Expression (Moderno)

```
String feedback = switch (nota) {
    case 'A' -> "Excelente!";
    case 'B' -> "Bom";
    default -> "Pode melhorar";
};
System.out.println(feedback);
```


### Vantagens:


- Sintaxe mais concisa
- Não precisa de break (elimina erros)
- É uma expressão, retorna valor
- Compatível com pattern matching (Java 21+)

# O Coração do Java: A Orientação a Objetos

## Organizando o mundo em "objetos"

A orientação a objetos é o paradigma central do Java, onde organizamos nosso código em estruturas que combinam dados e comportamentos.

 **Classe** é como uma planta de uma casa  
Define a estrutura, mas não é um objeto real. É apenas o projeto.

 **Objeto** é como a casa construída  
Uma instância real criada a partir da classe, ocupando espaço na memória.

## Exemplo Prático:

```
// A "planta" (Classe)
class Carro {
    String modelo;
    int ano;

    void ligar() {
        System.out.println("O " + modelo + " está ligando!");
    }
}

// Criando "objetos" (casas construídas)
Carro meuCarro = new Carro();
```

## Pilares da OO

  
Planta (Classe)



  
Casa (Objeto)

  
**Encapsulamento**

Protege dados dentro de classes  
com modificadores de acesso

  
**Herança**

Reutilização de código entre  
classes com relação pai/filho

  
**Polimorfismo**

Um objeto pode assumir  
diferentes formas conforme o  
contexto

  
**Abstração**





Simplificar a realidade focando  
apenas no essencial

# As Superpotências do Java

## Características que Fazem a Diferença







### Robusto e Seguro

-  Garbage Collector gerencia automaticamente a memória
-  Tipagem forte evita erros comuns em tempo de execução
-  Sistema de exceções detalhado e rastreável
-  Arquitetura de segurança incorporada







### Ecossistema Gigante

-  Maven/Gradle para gerenciamento de dependências
-  Spring Framework domina o desenvolvimento corporativo
-  Milhares de bibliotecas prontas para qualquer necessidade
-  Comunidade ativa e suporte abundante



### Concorrência Simplificada

-  Suporte nativo a Threads desde sua criação
-  API concurrency com abstrações de alto nível
-  Project Loom com virtual threads (Java 21+)
-  Ideal para sistemas que fazem várias tarefas simultaneamente

“ Java continua evoluindo para atender às demandas dos sistemas modernos, mantendo sua reputação de confiabilidade e performance.

# Resumo e Próximos Passos

## Sua Jornada Apenas Começou

### O que vimos:

- Java: linguagem poderosa, multiplataforma, com ecossistema gigante
- Conceitos-chave: JVM, Bytecode, Orientação a Objetos
- Recursos modernos: inferência de tipo (var), switch expressions
- Mercado: alta demanda em diversos setores tecnológicos

### Por que Java continuará relevante:

- Evolução contínua (Java 21, 25) com recursos modernos
- Comunidade ativa e suporte corporativo (Oracle)
- Base de código legada massiva em empresas globais
- Expansão para novas áreas (IA, Cloud Native)

## Próximos passos sugeridos:



### Classes e Herança

Aprofunde-se nos conceitos de OO e como criar hierarquias de classes eficientes



### Collections

Domine as estruturas de dados como ArrayList, HashMap e Stream API



### Exceções

Aprenda a tratar erros e criar código robusto com try-catch



### Projeto Prático

Construa uma aplicação real usando Spring Boot, JPA e APIs REST