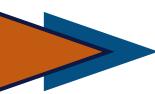




JAVA: A JORNADA DOS CÓDIGOS

ANNA ANDRADE



Introdução

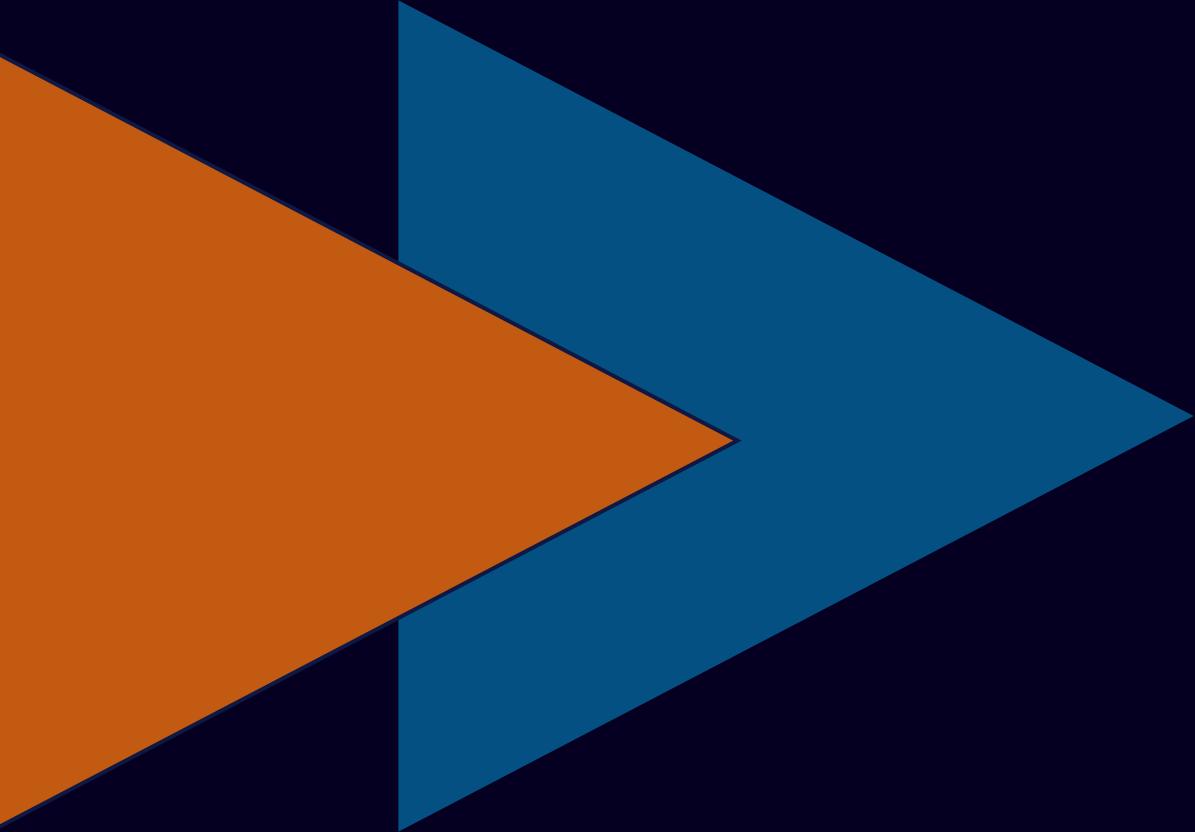
Java é uma linguagem de programação poderosa e versátil, usada para desenvolver aplicativos para diversas plataformas. Neste ebook, vamos explorar os principais comandos em Java e entender o básico necessário para que um código funcione. Vamos seguir uma abordagem simples e prática, com exemplos de código em contextos reais. Pronto para começar essa jornada? Vamos lá!

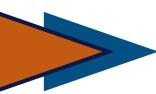


01

CAPÍTULO

Hello, World!





O primeiro passo

O primeiro passo em qualquer linguagem de programação é criar o programa "Olá, Mundo!". Este é um exemplo clássico e simples que ajuda a entender a estrutura básica de um programa na linguagem. Em Java, o "Olá, Mundo!" ensina você a criar uma classe, um método e imprimir uma mensagem no console.

Estrutura básica de um Programa Java

Um programa Java básico é estruturado da seguinte forma:

1. Classe: Toda aplicação Java começa com uma classe. A classe é um bloco de construção que pode conter variáveis e métodos.
2. Método *main*: O ponto de entrada do programa. É aqui que a execução do programa começa.
3. Instrução *System.out.println*: Utilizada para imprimir uma mensagem no console.

```
● ● ●

public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Olá, Mundo!");
    }
}
```

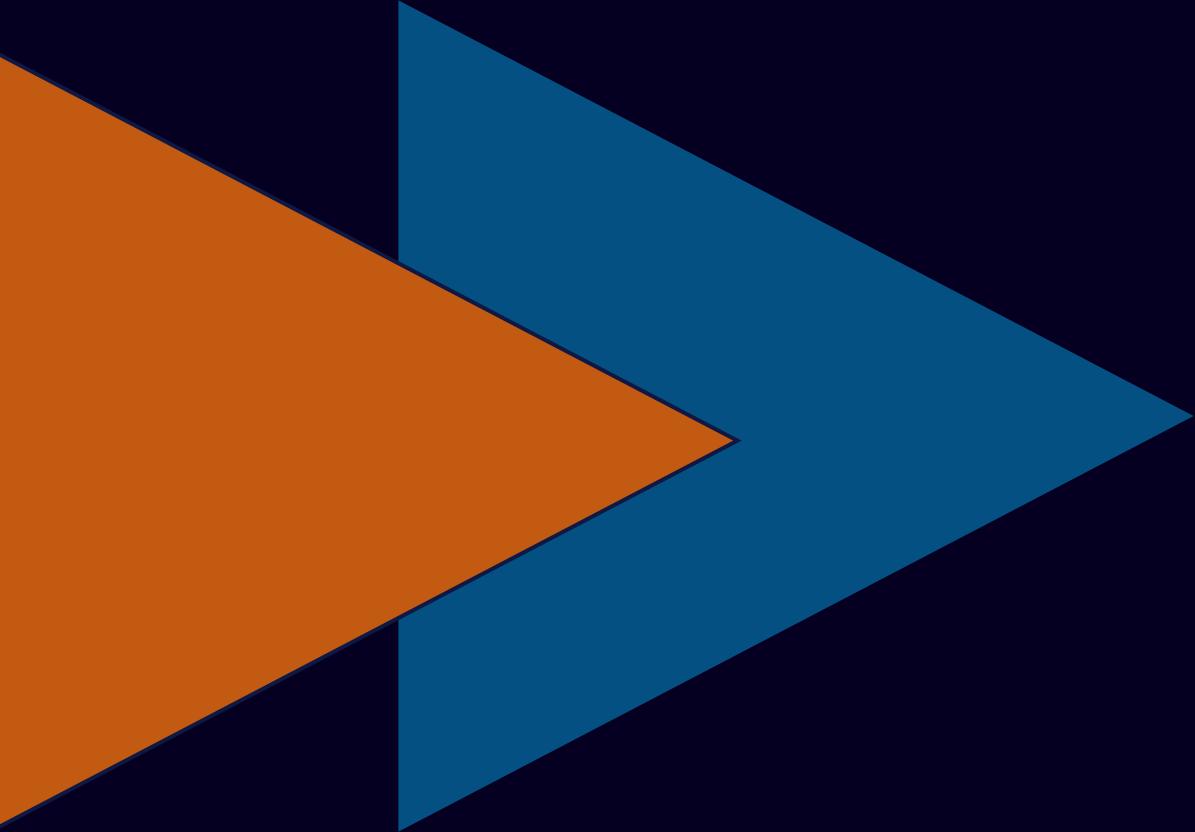
Contexto real

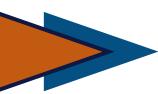
Este exemplo é um ponto de partida para entender como a estrutura básica de um programa Java funciona. Mesmo que pareça simples, ele prepara você para entender conceitos mais avançados, como a criação de métodos, classes e objetos.

02

CAPÍTULO

Variáveis e tipos de dados





Variáveis e tipos

As variáveis são essenciais em qualquer linguagem de programação, pois permitem armazenar e manipular dados. Em Java, você precisa declarar o tipo de dado de uma variável antes de usá-la. Este capítulo aborda os tipos de dados mais comuns em Java e como usar variáveis de forma eficaz.

Tipos de dados

Java é uma linguagem fortemente tipada, o que significa que você deve especificar o tipo de dado de cada variável.

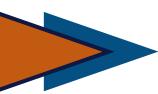
Os principais tipos de dados em Java são:

1. Tipos Primitivos
2. Tipos de Referência

Tipos primitivos

Os tipos primitivos são os tipos de dados mais básicos em Java. Eles incluem:

- int: Armazena números inteiros.
- double: Armazena números de ponto flutuante (decimais).
- char: Armazena um único caractere.
- boolean: Armazena valores true ou false.



Variáveis e tipos

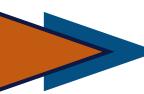
Tipos de dados

Tipos primitivos

```
public class TiposPrimitivos {  
    public static void main(String[] args) {  
        int numero = 10; // Inteiro  
        double preco = 19.99; // Decimal  
        char letra = 'A'; // Caractere  
        boolean ativo = true; // Booleano  
  
        System.out.println("Número: " + numero);  
        System.out.println("Preço: " + preco);  
        System.out.println("Letra: " + letra);  
        System.out.println("Ativo: " + ativo);  
    }  
}
```

Tipos de Referência

Os tipos de referência são mais complexos e incluem objetos e arrays. O tipo mais básico de referência é o tipo String, que armazena texto.



Variáveis e tipos

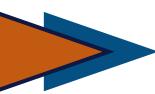
Tipos de dados

Tipos de Referência

```
public class TiposReferencia {  
    public static void main(String[] args) {  
        String mensagem = "Olá, Java!";  
  
        System.out.println(mensagem);  
    }  
}
```

Manipulação de variáveis

Você pode realizar várias operações com variáveis, como concatenar strings ou realizar cálculos com números.



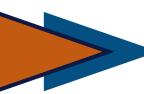
Variáveis e tipos

Manipulação de variáveis

```
public class ManipulacaoVariaveis {  
    public static void main(String[] args) {  
        int a = 5;  
        int b = 3;  
        int soma = a + b;  
  
        String nome = "João";  
        String saudacao = "Olá, " + nome + "!";  
  
        System.out.println("Soma: " + soma);  
        System.out.println(saudacao);  
    }  
}
```

Conversão de Tipos

Às vezes, você precisa converter entre diferentes tipos de dados. Isso pode ser feito usando métodos de conversão, como parseInt para converter uma String em int.



Variáveis e tipos

Conversão de tipos

```
public class ConversaoTipos {  
    public static void main(String[] args) {  
        String numeroStr = "123";  
        int numero = Integer.parseInt(numeroStr);  
  
        System.out.println("Número convertido: " + numero);  
    }  
}
```

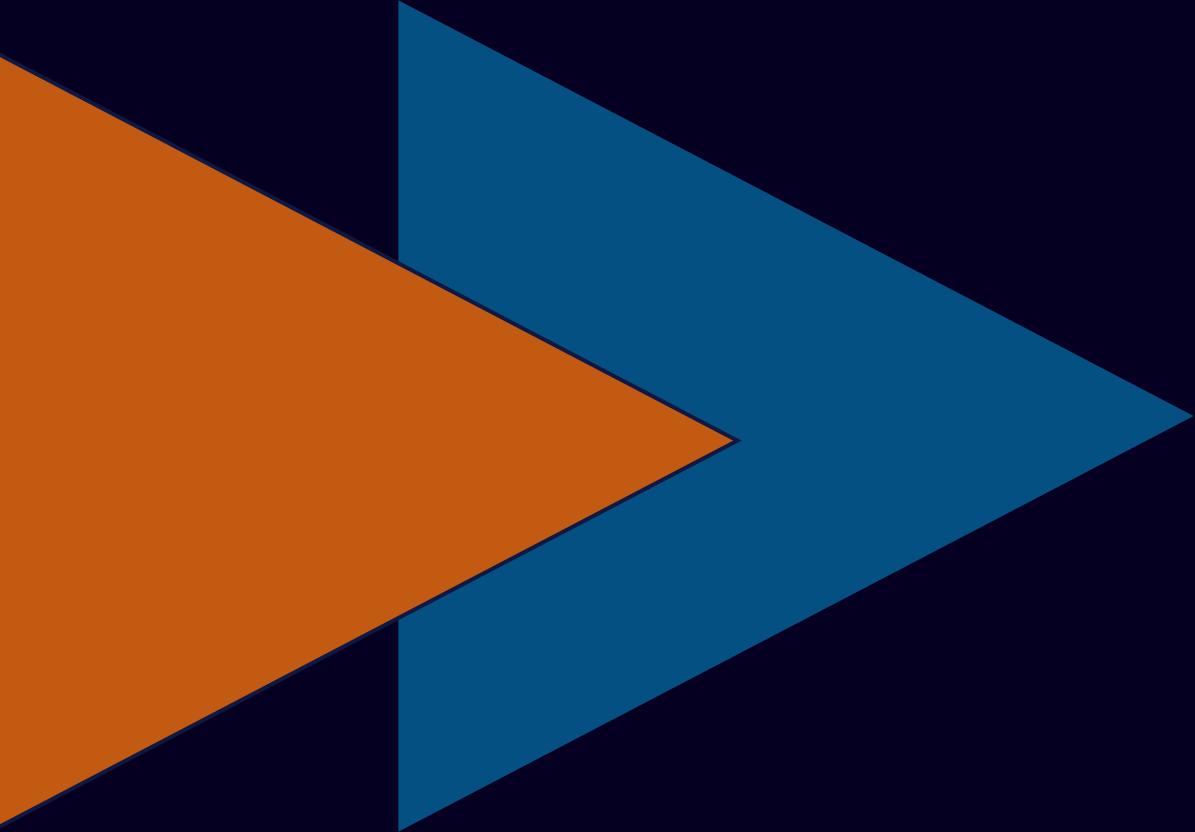
Contexto Real

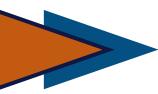
Em aplicativos reais, você usa variáveis para armazenar informações que serão processadas e manipuladas ao longo do programa. Por exemplo, em um aplicativo de e-commerce, você pode usar variáveis para armazenar o preço dos produtos, a quantidade em estoque e o nome do cliente.

03

CAPÍTULO

Estruturas de Controle





Estruturas de controle

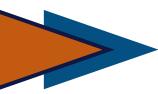
As estruturas de controle em Java permitem alterar o fluxo de execução do programa com base em condições ou repetições. As principais estruturas de controle são as condições (if, else, switch) e os laços de repetição (for, while, do-while).

IF e ELSE

A estrutura if é usada para executar um bloco de código se uma condição for verdadeira. A estrutura else é usada para executar um bloco de código alternativo quando a condição do if é falsa.



```
public class Condicionais {  
    public static void main(String[] args) {  
        int idade = 20;  
  
        if (idade >= 18) {  
            System.out.println("Você é maior de idade.");  
        } else {  
            System.out.println("Você é menor de idade.");  
        }  
    }  
}
```



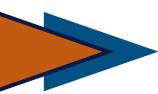
Estruturas de controle

ELSE IF

O else if permite testar várias condições. Se a condição do if inicial for falsa, o else if pode verificar uma nova condição.



```
public class CondicionaisComElseIf {  
    public static void main(String[] args) {  
        int nota = 85;  
  
        if (nota >= 90) {  
            System.out.println("Nota A");  
        } else if (nota >= 80) {  
            System.out.println("Nota B");  
        } else if (nota >= 70) {  
            System.out.println("Nota C");  
        } else {  
            System.out.println("Nota D");  
        }  
    }  
}
```



Estruturas de controle

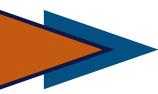
SWITCH

O switch é uma alternativa ao if-else if quando você precisa comparar uma variável com vários valores possíveis.

```
public class SwitchExemplo {
    public static void main(String[] args) {
        int dia = 3;
        String nomeDia;

        switch (dia) {
            case 1:
                nomeDia = "Domingo";
                break;
            case 2:
                nomeDia = "Segunda-feira";
                break;
            case 3:
                nomeDia = "Terça-feira";
                break;
            case 4:
                nomeDia = "Quarta-feira";
                break;
            default:
                nomeDia = "Dia inválido";
                break;
        }

        System.out.println("O dia da semana é: " + nomeDia);
    }
}
```

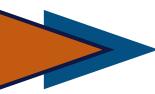


Estruturas de controle

FOR

O laço for é usado quando você sabe quantas vezes deseja repetir um bloco de código.

```
public class ForLoop {  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            System.out.println("Número: " + i);  
        }  
    }  
}
```



Estruturas de controle

WHILE

O laço while executa um bloco de código enquanto uma condição é verdadeira. Use-o quando não souber de antemão quantas vezes o código deve ser repetido.



```
public class WhileLoop {  
    public static void main(String[] args) {  
        int i = 0;  
        while (i < 5) {  
            System.out.println("Número: " + i);  
            i++;  
        }  
    }  
}
```



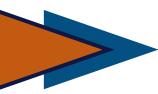
Estruturas de controle

DO WHILE

O laço do-while é semelhante ao while, mas garante que o bloco de código seja executado pelo menos uma vez, pois a condição é verificada após a execução.



```
public class DoWhileLoop {  
    public static void main(String[] args) {  
        int i = 0;  
        do {  
            System.out.println("Número: " + i);  
            i++;  
        } while (i < 5);  
    }  
}
```



Estruturas de controle

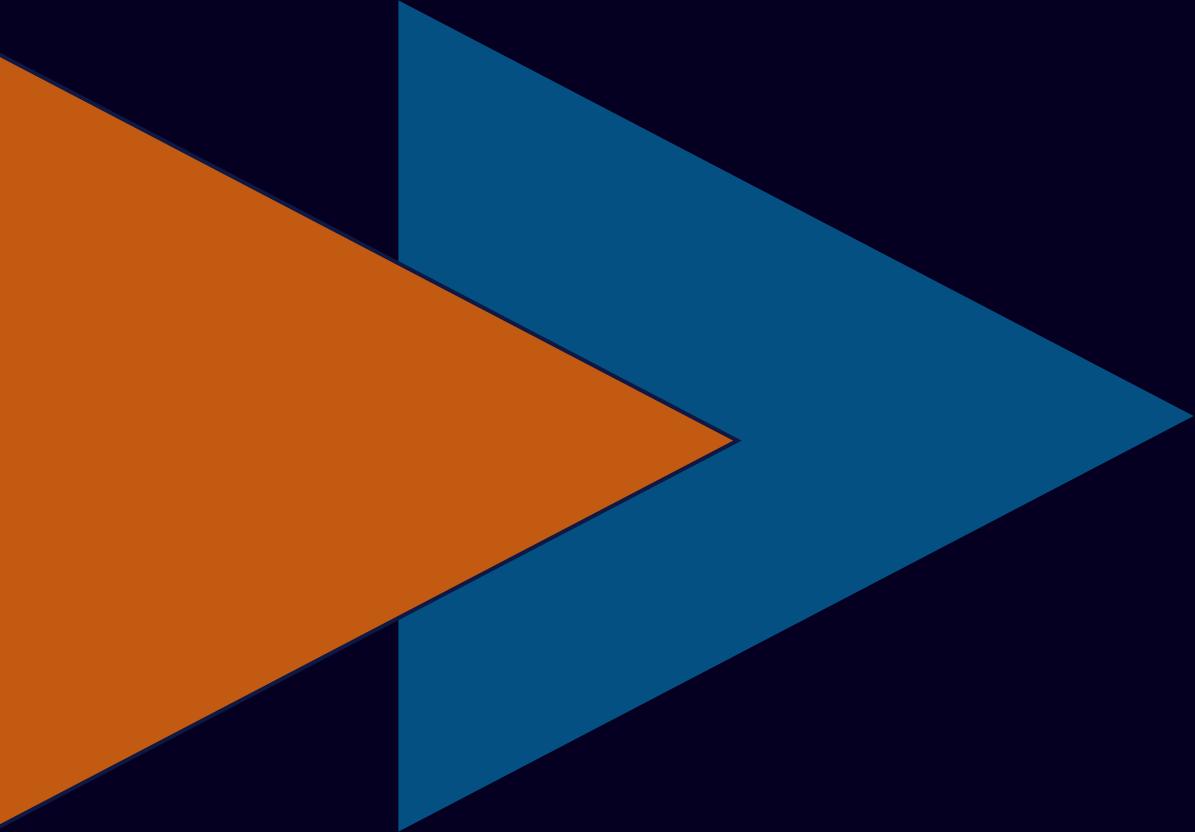
Contexto Real

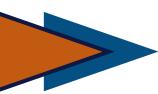
Estruturas de controle são amplamente usadas para criar lógica em aplicativos. Por exemplo, em um sistema de login, você pode usar `if` para verificar se o usuário e a senha estão corretos. Em um jogo, você pode usar `for` para criar um loop que atualiza a posição do personagem a cada quadro.

04

CAPÍTULO

Métodos





Métodos e funções

Métodos são blocos de código que realizam uma tarefa específica e podem ser reutilizados em diferentes partes do programa. Eles ajudam a organizar o código, evitando duplicação e tornando o código mais legível e fácil de manter.

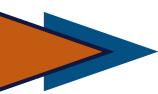
Declaração e definição

Os métodos em Java são definidos dentro de uma classe e podem ser chamados a partir de outros métodos ou objetos. Um método pode retornar um valor ou não.

```
public class ExemploMetodos {
    public static void main(String[] args) {
        saudar(); // Chama o método saudar
        int resultado = somar(5, 3); // Chama o método somar e armazena o resultado
        System.out.println("Resultado da soma: " + resultado);
    }

    // Método sem parâmetros e sem retorno
    public static void saudar() {
        System.out.println("Olá, Mundo!");
    }

    // Método com parâmetros e com retorno
    public static int somar(int a, int b) {
        return a + b;
    }
}
```



Métodos e funções

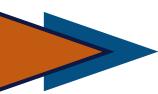
Sobrecarga de Métodos

A sobrecarga de métodos permite que você tenha múltiplos métodos com o mesmo nome, mas com diferentes parâmetros. Isso é útil para realizar a mesma operação com diferentes tipos ou quantidades de dados.

```
public class SobrecargaExemplo {
    public static void main(String[] args) {
        System.out.println(adicionar(5, 3)); // Chama o método com dois parâmetros
        System.out.println(adicionar(1.5, 2.5)); // Chama o método com dois parâmetros do tipo
double
    }

    // Método para somar dois inteiros
    public static int adicionar(int a, int b) {
        return a + b;
    }

    // Método sobreescrito para somar dois doubles
    public static double adicionar(double a, double b) {
        return a + b;
    }
}
```



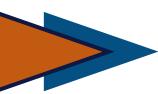
Métodos e funções

Métodos Estáticos

Métodos estáticos pertencem à classe, não a instâncias da classe. Eles podem ser chamados sem criar um objeto da classe. Use métodos estáticos para operações que não dependem do estado de um objeto.



```
public class MetodosEstaticos {  
    public static void main(String[] args) {  
        int resultado = calcularQuadrado(4);  
        System.out.println("Quadrado de 4 é: " + resultado);  
    }  
  
    // Método estático para calcular o quadrado de um número  
    public static int calcularQuadrado(int numero) {  
        return numero * numero;  
    }  
}
```

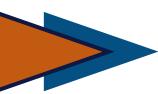


Métodos e funções

Métodos de Instância

Métodos de instância são associados a uma instância específica da classe. Eles podem acessar e modificar variáveis de instância.

```
public class MetodoInstancia {  
    int valor;  
  
    public static void main(String[] args) {  
        MetodoInstancia obj = new MetodoInstancia();  
        obj.valor = 10;  
        obj.exibirValor(); // Chama o método de instância  
    }  
  
    // Método de instância para exibir o valor  
    public void exibirValor() {  
        System.out.println("O valor é: " + valor);  
    }  
}
```

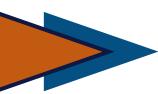


Métodos e funções

Parâmetros de Métodos

Métodos podem aceitar parâmetros que são passados quando o método é chamado. Esses parâmetros são usados dentro do método para realizar operações.

```
public class MetodoComParametros {  
    public static void main(String[] args) {  
        saudacao("Ana"); // Chama o método saudacao com um parâmetro  
    }  
  
    // Método com um parâmetro  
    public static void saudacao(String nome) {  
        System.out.println("Olá, " + nome + "!");  
    }  
}
```



Métodos e funções

Retornos de Métodos

Métodos que retornam valores devem ter um tipo de retorno especificado. O valor retornado pode ser usado onde o método é chamado.

```
public class MetodoComRetorno {  
    public static void main(String[] args) {  
        int resultado = multiplicar(4, 5);  
        System.out.println("Resultado da multiplicação: " + resultado);  
    }  
  
    // Método que retorna o produto de dois números  
    public static int multiplicar(int a, int b) {  
        return a * b;  
    }  
}
```

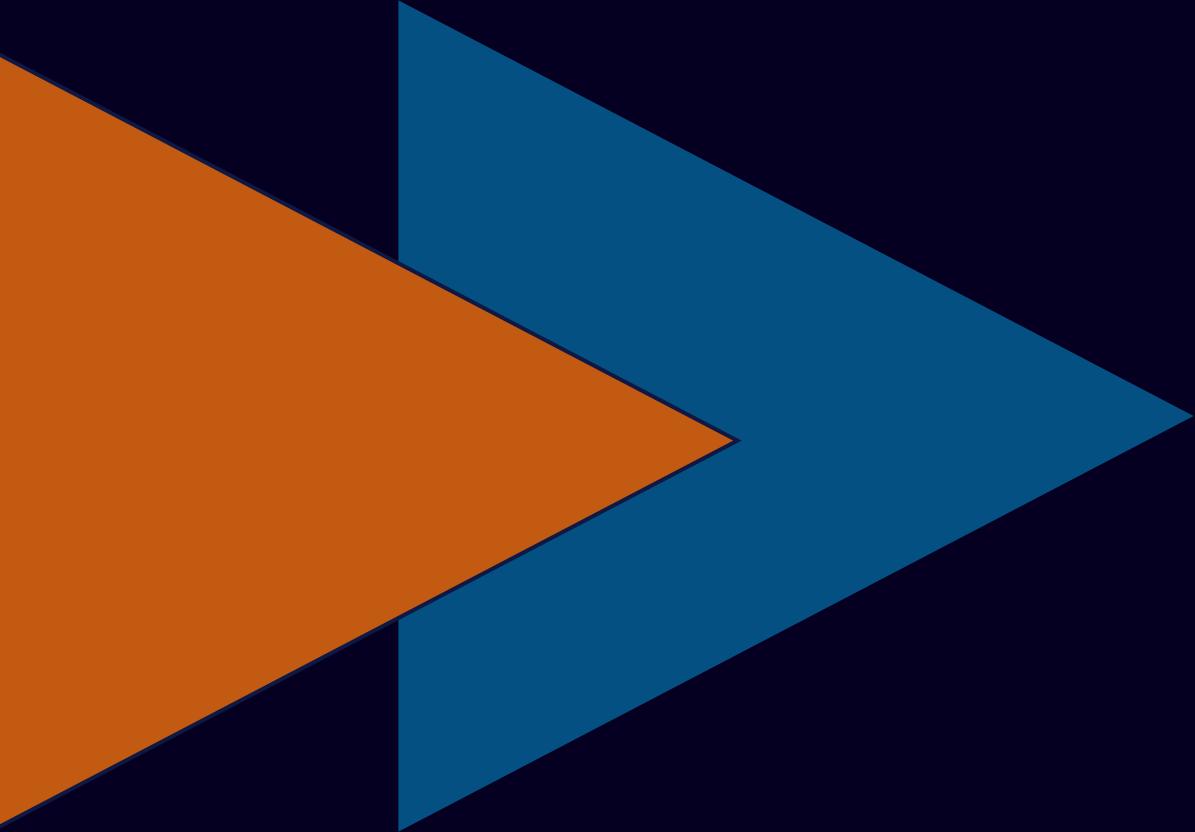
Contexto Real

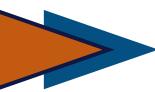
Métodos são usados em praticamente todos os programas Java. Eles ajudam a modularizar o código, facilitam a reutilização e a manutenção. Por exemplo, em um sistema de gerenciamento de usuários, você pode ter métodos para adicionar, remover e listar usuários.

05

CAPÍTULO

Arrays e Coleções





Arrays e Coleções

Arrays e coleções são estruturas importantes em Java para armazenar e gerenciar conjuntos de dados. Arrays fornecem uma maneira simples de armazenar múltiplos valores do mesmo tipo, enquanto as coleções oferecem mais flexibilidade e funcionalidades avançadas para manipulação de dados.

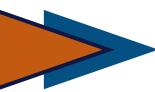
Arrays

Arrays são estruturas de dados que armazenam múltiplos valores em uma única variável, com todos os valores sendo do mesmo tipo. Eles têm um tamanho fixo que deve ser especificado na criação.



```
public class ArraysExemplo {
    public static void main(String[] args) {
        // Declaração e inicialização de um array de inteiros
        int[] numeros = {1, 2, 3, 4, 5};

        // Acessando e imprimindo os valores do array
        for (int i = 0; i < numeros.length; i++) {
            System.out.println("Número: " + numeros[i]);
        }
    }
}
```



Arrays e Coleções

Arrays Multidimensionais

Arrays podem ter mais de uma dimensão, permitindo criar tabelas ou matrizes.

```
public class ArraysMultidimensionais {
    public static void main(String[] args) {
        // Declaração e inicialização de um array bidimensional
        int[][] matriz = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        // Acessando e imprimindo os valores da matriz
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Arrays e Coleções

Coleções

Coleções fornecem uma maneira mais flexível de trabalhar com grupos de dados, comparado aos arrays. A biblioteca de coleções de Java inclui várias interfaces e classes úteis.

Listas

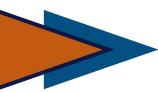
A interface List representa uma coleção ordenada de elementos. A classe mais comum que implementa List é ArrayList.

```
import java.util.ArrayList;

public class ListaExemplo {
    public static void main(String[] args) {
        // Criação de uma ArrayList
        ArrayList<String> lista = new ArrayList<>();

        // Adicionando elementos à lista
        lista.add("Java");
        lista.add("Python");
        lista.add("JavaScript");

        // Imprimindo todos os elementos da lista
        for (String linguagem : lista) {
            System.out.println(linguagem);
        }
    }
}
```



Arrays e Coleções

Coleções

Conjuntos

A interface Set representa uma coleção que não permite elementos duplicados. A classe mais comum que implementa Set é HashSet.

```
import java.util.HashSet;

public class ConjuntoExemplo {
    public static void main(String[] args) {
        // Criação de um HashSet
        HashSet<String> conjunto = new HashSet<>();

        // Adicionando elementos ao conjunto
        conjunto.add("Java");
        conjunto.add("Python");
        conjunto.add("JavaScript");
        conjunto.add("Java"); // Este elemento será ignorado, pois conjuntos não permitem
                            // duplicatas

        // Imprimindo todos os elementos do conjunto
        for (String linguagem : conjunto) {
            System.out.println(linguagem);
        }
    }
}
```



Arrays e Coleções

Coleções

Mapas

A interface Map representa uma coleção de pares chave-valor. A classe mais comum que implementa Map é HashMap.

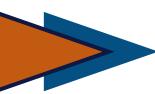


```
import java.util.HashMap;

public class MapaExemplo {
    public static void main(String[] args) {
        // Criação de um HashMap
        HashMap<String, String> mapa = new HashMap<>();

        // Adicionando pares chave-valor ao mapa
        mapa.put("Java", "Linguagem de Programação");
        mapa.put("Python", "Linguagem de Programação");
        mapa.put("JavaScript", "Linguagem de Programação");

        // Imprimindo todos os pares chave-valor do mapa
        for (String chave : mapa.keySet()) {
            String valor = mapa.get(chave);
            System.out.println(chave + ": " + valor);
        }
    }
}
```



Arrays e Coleções

Coleções

Iteradores

Iteradores são usados para percorrer elementos de coleções de maneira segura. Eles fornecem métodos para iterar, remover e acessar elementos.

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteradorExemplo {
    public static void main(String[] args) {
        ArrayList<String> lista = new ArrayList<>();
        lista.add("Java");
        lista.add("Python");
        lista.add("JavaScript");

        // Criando um iterador para a lista
        Iterator<String> iterador = lista.iterator();

        // Usando o iterador para percorrer a lista
        while (iterador.hasNext()) {
            String linguagem = iterador.next();
            System.out.println(linguagem);
        }
    }
}
```



Arrays e Coleções

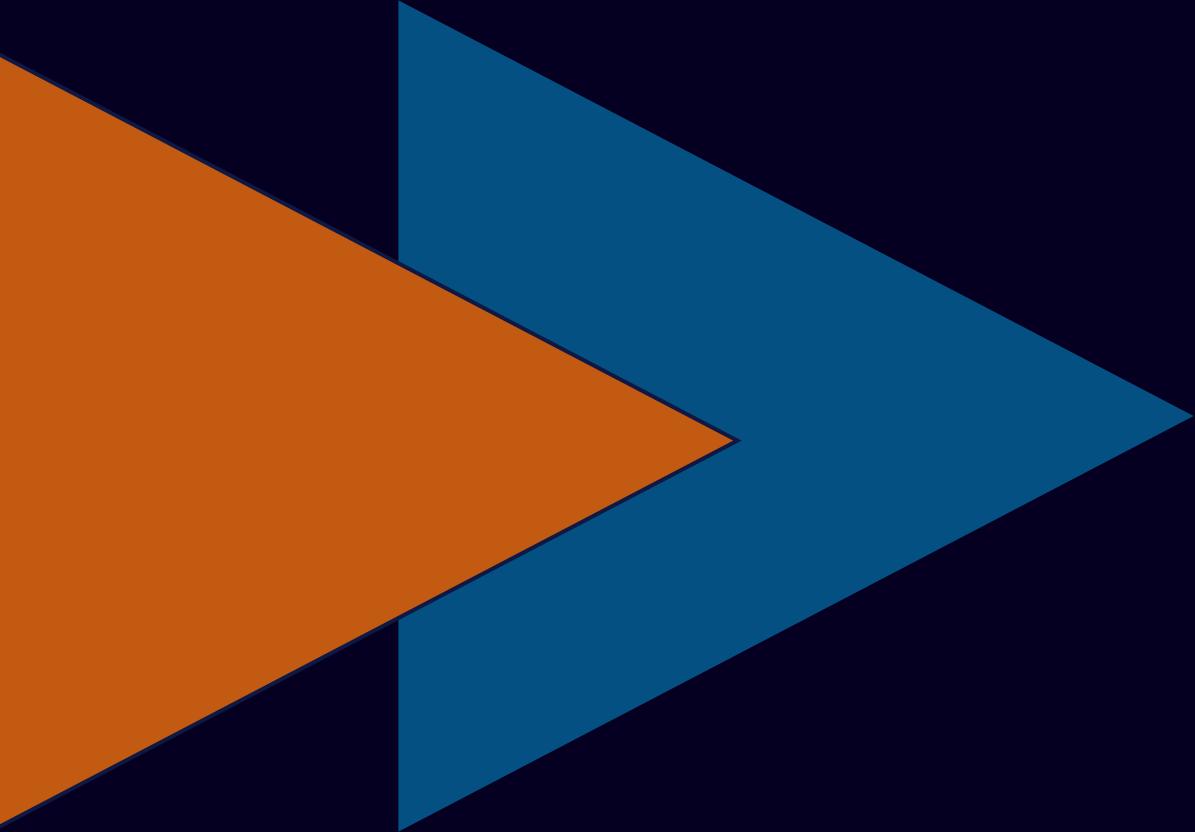
Contexto Real

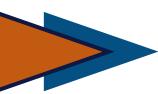
Arrays e coleções são amplamente utilizados em aplicativos Java para gerenciar dados. Por exemplo, em uma aplicação de gerenciamento de contatos, você pode usar um ArrayList para armazenar uma lista de contatos ou um HashMap para associar nomes a números de telefone.

06

CAPÍTULO

Classes e objetos





Classes e objetos

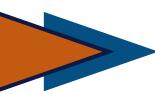
Em Java, a programação orientada a objetos (POO) é um paradigma fundamental que organiza o código em classes e objetos. As classes definem a estrutura e o comportamento dos objetos, enquanto os objetos são instâncias dessas classes.

O que é uma classe?

Uma classe é um molde ou um modelo para criar objetos. Ela define os atributos (dados) e os métodos (comportamentos) que os objetos daquela classe terão.

Para declarar uma classe, você usa a palavra-chave `class`, seguida pelo nome da classe e pelo corpo da classe.

```
public class Carro {  
    // Atributos  
    String modelo;  
    int ano;  
  
    // Método  
    void exibirInformacoes() {  
        System.out.println("Modelo: " + modelo);  
        System.out.println("Ano: " + ano);  
    }  
}
```



Classes e objetos

Criando e Usando Objetos

Objetos são instâncias de classes. Para criar um objeto, você usa o operador new seguido do construtor da classe.



```
public class Main {  
    public static void main(String[] args) {  
        // Criação de um objeto da classe Carro  
        Carro meuCarro = new Carro();  
  
        // Definindo valores para os atributos  
        meuCarro.modelo = "Fusca";  
        meuCarro.ano = 1970;  
  
        // Chamando o método para exibir informações  
        meuCarro.exibirInformacoes();  
    }  
}
```



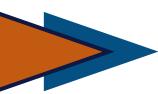
Classes e objetos

Construtores

Construtores são métodos especiais que são chamados quando um objeto é criado. Eles inicializam os atributos do objeto.



```
public class Carro {  
    // Atributos  
    String modelo;  
    int ano;  
  
    // Construtor  
    public Carro(String modelo, int ano) {  
        this.modelo = modelo;  
        this.ano = ano;  
    }  
  
    // Método  
    void exibirInformacoes() {  
        System.out.println("Modelo: " + modelo);  
        System.out.println("Ano: " + ano);  
    }  
}
```



Classes e objetos

Construtores

• • •

```
public class Main {  
    public static void main(String[] args) {  
        // Criação de um objeto da classe Carro usando o construtor  
        Carro meuCarro = new Carro("Fusca", 1970);  
  
        // Chamando o método para exibir informações  
        meuCarro.exibirInformacoes();  
    }  
}
```



Classes e objetos

Métodos e Atributos de Instância

Métodos e atributos de instância pertencem a um objeto específico. Cada objeto tem seus próprios valores para os atributos e pode executar seus próprios métodos.

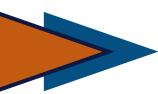
```
public class Pessoa {  
    // Atributos  
    String nome;  
    int idade;  
  
    // Método  
    void apresentar() {  
        System.out.println("Olá, meu nome é " + nome + " e eu tenho " + idade + " anos.");  
    }  
}
```

Encapsulamento

Encapsulamento é o conceito de esconder os detalhes internos de uma classe e expor apenas o que é necessário. Isso é feito usando modificadores de acesso e métodos getter e setter.

Modificadores de acesso

Os modificadores de acesso controlam a visibilidade dos atributos e métodos. Os mais comuns são `private`, `protected`, e `public`.



Classes e objetos

Encapsulamento

Modificadores de acesso

```
public class ContaBancaria {  
    // Atributos privados  
    private double saldo;  
  
    // Método público para depositar dinheiro  
    public void depositar(double valor) {  
        if (valor > 0) {  
            saldo += valor;  
        }  
    }  
  
    // Método público para obter o saldo  
    public double getSaldo() {  
        return saldo;  
    }  
}
```



Classes e objetos

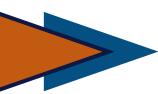
Herança

Herança é um mecanismo que permite criar uma nova classe com base em uma classe existente. A classe existente é chamada de classe base (ou superclasse), e a nova classe é chamada de classe derivada (ou subclasse).



```
// Classe base
public class Animal {
    void comer() {
        System.out.println("O animal está comendo.");
    }
}

// Subclasse que herda de Animal
public class Cachorro extends Animal {
    void latir() {
        System.out.println("O cachorro está latindo.");
    }
}
```



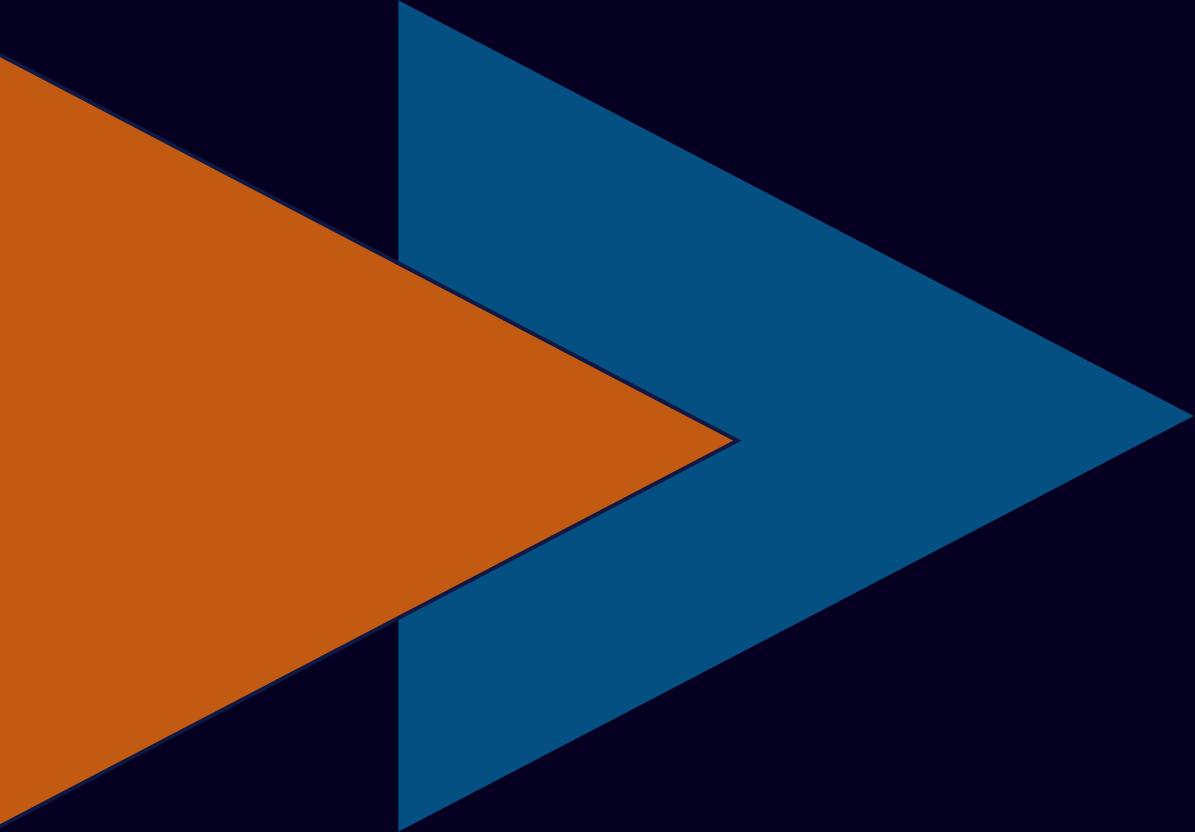
Classes e objetos

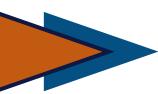
Contexto Real

Classes e objetos são a base da programação orientada a objetos em Java e são amplamente utilizados para modelar conceitos do mundo real. Por exemplo, em um sistema de biblioteca, você pode ter classes como Livro, Autor e Leitor, onde cada classe define atributos e comportamentos específicos relacionados ao seu conceito.

07

CONCLUSÃO





Conclusão

Parabéns por chegar até aqui! Agora você conhece os principais comandos e conceitos básicos para começar a programar em Java. Continue praticando e explorando mais sobre essa linguagem poderosa. Boa jornada na sua aventura pelo mundo do Java!