



Java – Comandos Básicos para seu conhecimento

Este material foi elaborado para apoiar alunos iniciantes no aprendizado da linguagem Java, reunindo **50 exemplos práticos** que abordam os principais fundamentos exigidos para o **nível júnior**. Os tópicos estão organizados de forma didática e categorizada, cobrindo desde a sintaxe básica até conceitos mais avançados como orientação a objetos, tratamento de exceções e manipulação de arquivos.

O objetivo é proporcionar uma base sólida e acessível para quem está começando, com exemplos curtos, diretos e fáceis de entender. Cada item representa uma habilidade essencial para a construção de programas Java e serve como ponto de partida para projetos mais elaborados no futuro.

Dominar os conceitos apresentados neste guia é um passo fundamental para qualquer pessoa que deseje se tornar desenvolvedor (a) Java.

Ao praticar cada um dos 50 exemplos, o estudante desenvolve a lógica de programação, compreende o funcionamento da linguagem e se prepara para enfrentar desafios mais complexos em ambientes acadêmicos ou profissionais. Recomenda-se experimentar, modificar e combinar os exemplos para reforçar o aprendizado. A prática constante é a chave para transformar conhecimento teórico em experiência prática de desenvolvimento.

1. Sintaxe Básica

1. **HelloWorld** – Primeiro programa que imprime "Olá Mundo". Serve para verificar se o ambiente Java está funcionando
2. **Scanner** – Permite ler dados digitados pelo usuário, como números e textos.
3. **Conversão de tipos** – Transformar valores de um tipo para outro, como int para String e vice-versa.
4. **Operações matemáticas** – Usar operadores como +, -, *, / para fazer contas simples.
5. **Math.random() e Math.pow()** – Gerar números aleatórios ou calcular potências.

2. Estruturas de Controle

6. **if, else if, else** – Executa ações diferentes com base em condições.
7. **switch case** – Escolhe o que fazer com base no valor de uma variável.
8. **for** – Repetição com número fixo de vezes.
9. **while** – Repetição enquanto a condição for verdadeira.
10. **do-while** – Semelhante ao while, mas garante pelo menos uma execução.

3. Operadores e Expressões

11. **Operadores relacionais** – Comparam valores (ex: ==, !=, >, <).
12. **Operadores lógicos** – Verificam condições compostas (ex: &&, ||, !).
13. **Operador ternário** – Um if resumido: condição ? valor1 : valor2.
14. **Incremento/Decremento** – Aumenta ou diminui uma variável (x++, x--).
15. **break/continue** – break encerra o laço; continue pula para a próxima repetição.



4. Métodos

- 16. **Método com retorno int** – Retorna um valor inteiro ao ser chamado.
- 17. **Método void** – Executa algo, mas não retorna valor.
- 18. **Parâmetros por valor** – Passa cópia dos dados ao método (os originais não mudam).
- 19. **Sobrecarga** – Dois métodos com mesmo nome, mas diferentes parâmetros.
- 20. **Recursão** – Método que chama ele mesmo, usado para cálculos como fatorial.

5. Arrays e Coleções

- 21. **Array unidimensional** – Lista com vários elementos do mesmo tipo.
- 22. **Array bidimensional (matriz)** – Tabela com linhas e colunas.
- 23. **ArrayList** – Lista dinâmica (pode crescer ou diminuir).
- 24. **for-each** – Forma mais simples de percorrer arrays ou listas.
- 25. **Arrays.sort()** – Ordena um array de forma crescente.

6. Programação Orientada a Objetos (POO)

- 26. **Classe com atributos** – Define o que um objeto tem (ex: nome, idade).
- 27. **Criar objeto** – Instancia um novo elemento com base na classe.
- 28. **Encapsulamento (get/set)** – Protege os dados e permite acessá-los com métodos.
- 29. **Construtor** – Método especial que inicializa objetos.
- 30. **Herança (extends)** – Uma classe herda atributos e métodos de outra.

7. Polimorfismo e Interfaces

- 31. **Classe abstrata** – Classe incompleta que serve de base para outras.
- 32. **Interface** – Define métodos que outras classes devem implementar.
- 33. **Sobrescrita (@Override)** – Redefinir um método herdado com novo comportamento.
- 34. **Casting de objetos** – Conversão entre tipos relacionados (superclasse e subclasse).
- 35. **instanceof** – Verifica se um objeto é de um determinado tipo.

8. Manipulação de Strings

- 36. **.equals() vs ==** – .equals() compara conteúdo; == compara se são o mesmo objeto.
- 37. **.replace()** – Substitui partes da string por outras.
- 38. **.split()** – Divide a string em partes (ex: separar palavras).
- 39. **.substring()** – Pega apenas parte da string.
- 40. **.toUpperCase() / .toLowerCase()** – Deixa a string toda maiúscula ou minúscula.

9. Entrada/Saída de Arquivos

- 41. **Leitura com Scanner** – Lê arquivos linha por linha.
- 42. **Escrita com PrintWriter** – Escreve dados simples em arquivos.
- 43. **Leitura com BufferedReader** – Lê arquivos de forma mais eficiente.
- 44. **Escrita com FileWriter** – Escreve dados em arquivos, com mais controle.
- 45. **Manipulação com File** – Verifica se um arquivo existe, cria ou deleta arquivos.

10. Exceções e Erros

- 46. **try-catch** – Trata erros para evitar que o programa pare.
- 47. **finally** – Bloco que sempre roda (usado para fechar arquivos, por exemplo).
- 48. **Exceção personalizada** – Criar sua própria classe de erro.
- 49. **throw** – Lança uma exceção manualmente.
- 50. **throws** – Declara que um método pode gerar exceções.



Agora vamos aos códigos:

HelloWorld.java

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

EntradaScanner.java

```
import java.util.Scanner;  
  
public class EntradaScanner {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.print("Digite seu nome: ");  
        String nome = sc.nextLine();  
        System.out.println("Olá, " + nome);  
        sc.close();  
    }  
}
```

ConversaoTipos.java

```
public class ConversaoTipos {  
    public static void main(String[] args) {  
        int numero = 10;  
        String texto = String.valueOf(numero);  
        int convertido = Integer.parseInt(texto);  
        System.out.println("Texto: " + texto + ", Número: " + convertido);  
    }  
}
```

OperacoesMatematicas.java

```
public class OperacoesMatematicas {  
    public static void main(String[] args) {  
        int a = 5, b = 2;  
        System.out.println("Soma: " + (a + b));  
        System.out.println("Subtração: " + (a - b));  
        System.out.println("Multiplicação: " + (a * b));  
        System.out.println("Divisão: " + (a / b));  
    }  
}
```

MathFuncoes.java

```
public class MathFuncoes {  
    public static void main(String[] args) {  
        double aleatorio = Math.random();  
        double potencia = Math.pow(2, 3);  
        System.out.println("Aleatório: " + aleatorio);  
    }  
}
```



```
        System.out.println("2 elevado a 3: " + potencia);
    }
}
```

CondicionallfElse.java

```
public class CondicionallfElse {
    public static void main(String[] args) {
        int idade = 18;
        if (idade < 18) {
            System.out.println("Menor de idade");
        } else if (idade == 18) {
            System.out.println("Tem 18 anos");
        } else {
            System.out.println("Maior de idade");
        }
    }
}
```

SwitchCase.java

```
public class SwitchCase {
    public static void main(String[] args) {
        int dia = 3;
        switch (dia) {
            case 1: System.out.println("Domingo"); break;
            case 2: System.out.println("Segunda"); break;
            case 3: System.out.println("Terça"); break;
            default: System.out.println("Outro dia");
        }
    }
}
```

ForLoop.java

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

WhileLoop.java

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



DoWhileLoop.java

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

OperadoresRelacionais.java

```
public class OperadoresRelacionais {  
    public static void main(String[] args) {  
        int a = 5, b = 10;  
        System.out.println(a == b);  
        System.out.println(a != b);  
        System.out.println(a > b);  
    }  
}
```

OperadoresLogicos.java

```
public class OperadoresLogicos {  
    public static void main(String[] args) {  
        boolean a = true, b = false;  
        System.out.println(a && b);  
        System.out.println(a || b);  
        System.out.println(!a);  
    }  
}
```

OperadorTernario.java

```
public class OperadorTernario {  
    public static void main(String[] args) {  
        int idade = 20;  
        String status = (idade >= 18) ? "Adulto" : "Menor";  
        System.out.println(status);  
    }  
}
```

IncrementoDecremento.java

```
public class IncrementoDecremento {  
    public static void main(String[] args) {  
        int x = 5;  
        x++;  
        System.out.println("Incrementado: " + x);  
        x--;  
        System.out.println("Decrementado: " + x);  
    }  
}
```



BreakContinue.java

```
public class BreakContinue {  
    public static void main(String[] args) {  
        for (int i = 0; i < 5; i++) {  
            if (i == 2) continue;  
            if (i == 4) break;  
            System.out.println("Valor: " + i);  
        }  
    }  
}
```

MetodoComRetorno.java

```
public class MetodoComRetorno {  
    public static int somar(int a, int b) {  
        return a + b;  
    }  
    public static void main(String[] args) {  
        System.out.println(somar(2, 3));  
    }  
}
```

MetodoVoid.java

```
public class MetodoVoid {  
    public static void mostrarMensagem() {  
        System.out.println("Método void chamado.");  
    }  
    public static void main(String[] args) {  
        mostrarMensagem();  
    }  
}
```

ParametrosPorValor.java

```
public class ParametrosPorValor {  
    public static void alterarValor(int x) {  
        x = 10;  
    }  
    public static void main(String[] args) {  
        int numero = 5;  
        alterarValor(numero);  
        System.out.println(numero);  
    }  
}
```

SobrecargaMetodos.java

```
public class SobrecargaMetodos {  
    public static int somar(int a, int b) { return a + b; }  
    public static double somar(double a, double b) { return a + b; }  
    public static void main(String[] args) {  
        System.out.println(somar(2, 3));  
        System.out.println(somar(2.5, 3.5));  
    }  
}
```



```
}  
}
```

RecursaoFatorial.java

```
public class RecursaoFatorial {  
    public static int fatorial(int n) {  
        if (n <= 1) return 1;  
        return n * fatorial(n - 1);  
    }  
    public static void main(String[] args) {  
        System.out.println(fatorial(5));  
    }  
}
```

ArraySimples.java

```
public class ArraySimples {  
    public static void main(String[] args) {  
        int[] numeros = {1, 2, 3};  
        System.out.println(numeros[0]);  
    }  
}
```

ArrayBidimensional.java

```
public class ArrayBidimensional {  
    public static void main(String[] args) {  
        int[][] matriz = {{1, 2}, {3, 4}};  
        System.out.println(matriz[0][1]);  
    }  
}
```

ArrayListOperacoes.java

```
import java.util.ArrayList;  
public class ArrayListOperacoes {  
    public static void main(String[] args) {  
        ArrayList<String> lista = new ArrayList<>();  
        lista.add("A");  
        lista.add("B");  
        lista.remove("A");  
        System.out.println(lista.contains("B"));  
    }  
}
```

ForEachArray.java

```
public class ForEachArray {  
    public static void main(String[] args) {  
        int[] numeros = {1, 2, 3};  
        for (int num : numeros) {  
            System.out.println(num);  
        }  
    }  
}
```



```
}  
}
```

OrdenacaoArray.java

```
import java.util.Arrays;  
public class OrdenacaoArray {  
    public static void main(String[] args) {  
        int[] numeros = {3, 1, 2};  
        Arrays.sort(numeros);  
        System.out.println(Arrays.toString(numeros));  
    }  
}
```

ClasseComAtributos.java

```
public class ClasseComAtributos {  
    String nome;  
    int idade;  
}
```

CriarObjeto.java

```
public class CriarObjeto {  
    public static void main(String[] args) {  
        ClasseComAtributos pessoa = new ClasseComAtributos();  
        pessoa.nome = "João";  
        pessoa.idade = 30;  
        System.out.println(pessoa.nome);  
    }  
}
```

Encapsulamento.java

```
public class Encapsulamento {  
    private int idade;  
    public int getIdade() { return idade; }  
    public void setIdade(int idade) { this.idade = idade; }  
}
```

Construtores.java

```
public class Construtores {  
    String nome;  
    public Construtores(String nome) {  
        this.nome = nome;  
    }  
    public static void main(String[] args) {  
        Construtores obj = new Construtores("Ana");  
        System.out.println(obj.nome);  
    }  
}
```




HerancaSimples.java

```
class Animal {  
    void som() { System.out.println("Som genérico"); }  
}  
public class HerancaSimples extends Animal {  
    public static void main(String[] args) {  
        HerancaSimples cachorro = new HerancaSimples();  
        cachorro.som();  
    }  
}
```

ClasseAbstrata.java

```
abstract class Forma {  
    abstract void desenhar();  
}  
public class ClasseAbstrata extends Forma {  
    void desenhar() {  
        System.out.println("Desenhando...");  
    }  
    public static void main(String[] args) {  
        ClasseAbstrata c = new ClasseAbstrata();  
        c.desenhar();  
    }  
}
```

InterfaceImplementacao.java

```
interface Animal {  
    void emitirSom();  
}  
public class InterfaceImplementacao implements Animal {  
    public void emitirSom() {  
        System.out.println("Som!");  
    }  
    public static void main(String[] args) {  
        new InterfaceImplementacao().emitirSom();  
    }  
}
```

PolimorfismoOverride.java

```
class Animal {  
    void falar() { System.out.println("Animal"); }  
}  
public class PolimorfismoOverride extends Animal {  
    @Override  
    void falar() { System.out.println("Cachorro"); }  
    public static void main(String[] args) {  
        Animal a = new PolimorfismoOverride();  
        a.falar();  
    }  
}
```



CastingObjetos.java

```
class Pai {}  
class Filho extends Pai {}  
public class CastingObjetos {  
    public static void main(String[] args) {  
        Pai p = new Filho();  
        Filho f = (Filho) p;  
        System.out.println("Casting feito com sucesso.");  
    }  
}
```

InstanceofUso.java

```
public class InstanceofUso {  
    public static void main(String[] args) {  
        String texto = "Olá";  
        System.out.println(texto instanceof String);  
    }  
}
```

ComparacaoStrings.java

```
public class ComparacaoStrings {  
    public static void main(String[] args) {  
        String a = "Java";  
        String b = new String("Java");  
        System.out.println(a == b);  
        System.out.println(a.equals(b));  
    }  
}
```

SubstituicaoStrings.java

```
public class SubstituicaoStrings {  
    public static void main(String[] args) {  
        String texto = "Olá Mundo";  
        System.out.println(texto.replace("Mundo", "Java"));  
    }  
}
```

DivisaoStrings.java

```
public class DivisaoStrings {  
    public static void main(String[] args) {  
        String frase = "Aprender Java é legal";  
        String[] palavras = frase.split(" ");  
        for (String p : palavras) {  
            System.out.println(p);  
        }  
    }  
}
```

SubstringUso.java



```
public class SubstringUso {  
    public static void main(String[] args) {  
        String texto = "Java é ótimo";  
        System.out.println(texto.substring(5));  
    }  
}
```

TransformacaoStrings.java

```
public class TransformacaoStrings {  
    public static void main(String[] args) {  
        String nome = "João";  
        System.out.println(nome.toUpperCase());  
        System.out.println(nome.toLowerCase());  
    }  
}
```

LeituraScannerArquivo.java

```
import java.io.File;  
import java.util.Scanner;  
public class LeituraScannerArquivo {  
    public static void main(String[] args) throws Exception {  
        File arquivo = new File("exemplo.txt");  
        Scanner sc = new Scanner(arquivo);  
        while (sc.hasNextLine()) {  
            System.out.println(sc.nextLine());  
        }  
        sc.close();  
    }  
}
```

EscritaPrintWriter.java

```
import java.io.PrintWriter;  
public class EscritaPrintWriter {  
    public static void main(String[] args) throws Exception {  
        PrintWriter pw = new PrintWriter("saida.txt");  
        pw.println("Texto escrito com PrintWriter");  
        pw.close();  
    }  
}
```

LeituraBufferedReader.java

```
import java.io.*;  
public class LeituraBufferedReader {  
    public static void main(String[] args) throws Exception {  
        BufferedReader br = new BufferedReader(new FileReader("exemplo.txt"));  
        String linha;  
        while ((linha = br.readLine()) != null) {  
            System.out.println(linha);  
        }  
        br.close();  
    }  
}
```



```
}  
}
```

EscritaFileWriter.java

```
import java.io.FileWriter;  
public class EscritaFileWriter {  
    public static void main(String[] args) throws Exception {  
        FileWriter fw = new FileWriter("saida2.txt");  
        fw.write("Texto com FileWriter");  
        fw.close();  
    }  
}
```

ManipulacaoFile.java

```
import java.io.File;  
public class ManipulacaoFile {  
    public static void main(String[] args) {  
        File f = new File("saida2.txt");  
        System.out.println("Existe: " + f.exists());  
    }  
}
```

TryCatch.java

```
public class TryCatch {  
    public static void main(String[] args) {  
        try {  
            int x = 5 / 0;  
        } catch (ArithmeticException e) {  
            System.out.println("Erro: " + e.getMessage());  
        }  
    }  
}
```

FinallyUso.java

```
public class FinallyUso {  
    public static void main(String[] args) {  
        try {  
            System.out.println("Tentando...");  
        } finally {  
            System.out.println("Finalizado");  
        }  
    }  
}
```

ExcecaoPersonalizada.java

```
class MinhaExcecao extends Exception {  
    public MinhaExcecao(String msg) { super(msg); }  
}  
public class ExcecaoPersonalizada {  
    public static void main(String[] args) throws MinhaExcecao {
```



```
        throw new MinhaExcecao("Erro personalizado");
    }
}
```

ThrowUso.java

```
public class ThrowUso {
    public static void checarIdade(int idade) {
        if (idade < 18) {
            throw new IllegalArgumentException("Menor de idade");
        }
    }
    public static void main(String[] args) {
        checarIdade(15);
    }
}
```

ThrowsEmMetodo.java

```
public class ThrowsEmMetodo {
    public static void metodo() throws Exception {
        throw new Exception("Erro lançado");
    }
    public static void main(String[] args) throws Exception {
        metodo();
    }
}
```

Java – Orientação Objetos

A Programação Orientada a Objetos (POO) é um dos pilares da programação moderna e é amplamente utilizada em linguagens como Java. Diferente da programação estruturada, que foca em funções e procedimentos, a POO se baseia em objetos – entidades que combinam dados (atributos) e comportamentos (métodos).

Este material foi elaborado para guiar estudantes e desenvolvedores iniciantes pelos conceitos essenciais da orientação a objetos em Java.

Os exemplos abordam desde a criação de classes e objetos até temas mais avançados, como herança, polimorfismo, abstração e interfaces.

O objetivo é proporcionar uma base prática e sólida para quem deseja compreender e aplicar a POO de forma eficiente no desenvolvimento de sistemas.

Ao compreender e aplicar os conceitos apresentados neste material, o estudante desenvolve uma visão mais estruturada e reutilizável sobre como escrever código.

A POO facilita a manutenção, expansão e organização de aplicações, tornando-se essencial para projetos reais em Java e outras linguagens orientadas a objetos.

Praticar com os exemplos fornecidos é um excelente ponto de partida.

Com o tempo, o uso da orientação a objetos se tornará natural, e a construção de sistemas mais robustos será uma consequência direta desse aprendizado.



1. Definição e uso de classe simples

- ClasseSimples.java – Cria uma classe Pessoa com atributos e método.
- InstanciarObjeto.java – Cria um objeto da classe Pessoa e chama o método.

2. Encapsulamento com Getters e Setters

- EncapsulamentoPessoa.java – Classe PessoaEncapsulada com atributos privados e métodos get/set.
- TestaEncapsulamento.java – Cria e manipula um objeto usando encapsulamento.

3. Herança

- HerancaAnimal.java – Classe base Animal e subclasse Cachorro que sobrescreve um método.
- TestaHeranca.java (está dentro do mesmo arquivo HerancaAnimal.java) – Demonstra polimorfismo com Animal a = new Cachorro();.

4. Polimorfismo

- PolimorfismoVeiculo.java – Classe Veiculo com métodos sobrescritos em Carro e Bicicleta, usando array de referência para demonstrar polimorfismo.

5. Abstração

- ClasseAbstrataForma.java – Classe abstrata Forma e sua implementação Circulo. Demonstra conceito de contrato abstrato.

6. Interface

- InterfaceExemplo.java – Interface Operação implementada pela classe Soma. Demonstra abstração e polimorfismo com interface.

Agora vamos aos códigos

ClasseSimples.java

Cria uma classe Pessoa com atributos e método apresentar().

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



InstanciarObjeto.java

Instancia um objeto da classe Pessoa e chama seu método.

```
public class InstanciarObjeto {  
    public static void main(String[] args) {  
        Pessoa p = new Pessoa();  
        p.nome = "Ana";  
        p.idade = 25;  
        p.apresentar();  
    }  
}
```

EncapsulamentoPessoa.java

Classe PessoaEncapsulada com atributos privados e métodos get/set.

```
public class PessoaEncapsulada {  
    private String nome;  
    private int idade;  
  
    public String getNome() { return nome; }  
    public void setNome(String nome) { this.nome = nome; }  
  
    public int getIdade() { return idade; }  
    public void setIdade(int idade) { this.idade = idade; }  
}
```

TestaEncapsulamento.java

Cria e manipula um objeto com acesso controlado aos atributos.

```
public class TestaEncapsulamento {  
    public static void main(String[] args) {  
        PessoaEncapsulada p = new PessoaEncapsulada();  
        p.setNome("Carlos");  
        p.setIdade(30);  
        System.out.println(p.getNome() + " - " + p.getIdade());  
    }  
}
```

HerancaAnimal.java

Classe Animal e subclasse Cachorro com sobrescrita de método.

```
class Animal {  
    void emitirSom() {  
        System.out.println("Som genérico");  
    }  
}  
  
class Cachorro extends Animal {  
    void emitirSom() {  
        System.out.println("Latido");  
    }  
}
```



```
public class TestaHeranca {  
    public static void main(String[] args) {  
        Animal a = new Cachorro();  
        a.emitirSom();  
    }  
}
```

PolimorfismoVeiculo.java

Demonstra polimorfismo com as classes Veiculo, Carro e Bicicleta.

```
class Veiculo {  
    void mover() { System.out.println("Veículo em movimento"); }  
}  
class Carro extends Veiculo {  
    void mover() { System.out.println("Carro andando"); }  
}  
class Bicicleta extends Veiculo {  
    void mover() { System.out.println("Bicicleta pedalando"); }  
}  
  
public class TestaPolimorfismo {  
    public static void main(String[] args) {  
        Veiculo[] veiculos = {new Carro(), new Bicicleta()};  
        for (Veiculo v : veiculos) {  
            v.mover();  
        }  
    }  
}
```

ClasseAbstrataForma.java

Classe abstrata Forma e sua implementação Circulo.

```
abstract class Forma {  
    abstract void desenhar();  
}  
class Circulo extends Forma {  
    void desenhar() {  
        System.out.println("Desenhando círculo");  
    }  
}  
  
public class TestaForma {  
    public static void main(String[] args) {  
        Forma f = new Circulo();  
        f.desenhar();  
    }  
}
```




InterfaceExemplo.java

Interface Operacao implementada pela classe Soma.

```
interface Operacao {  
    int executar(int a, int b);  
}  
class Soma implements Operacao {  
    public int executar(int a, int b) {  
        return a + b;  
    }  
}  
public class TestaInterface {  
    public static void main(String[] args) {  
        Operacao op = new Soma();  
        System.out.println(op.executar(5, 3));  
    }  
}
```

EducaCiência FastCode para a comunidade