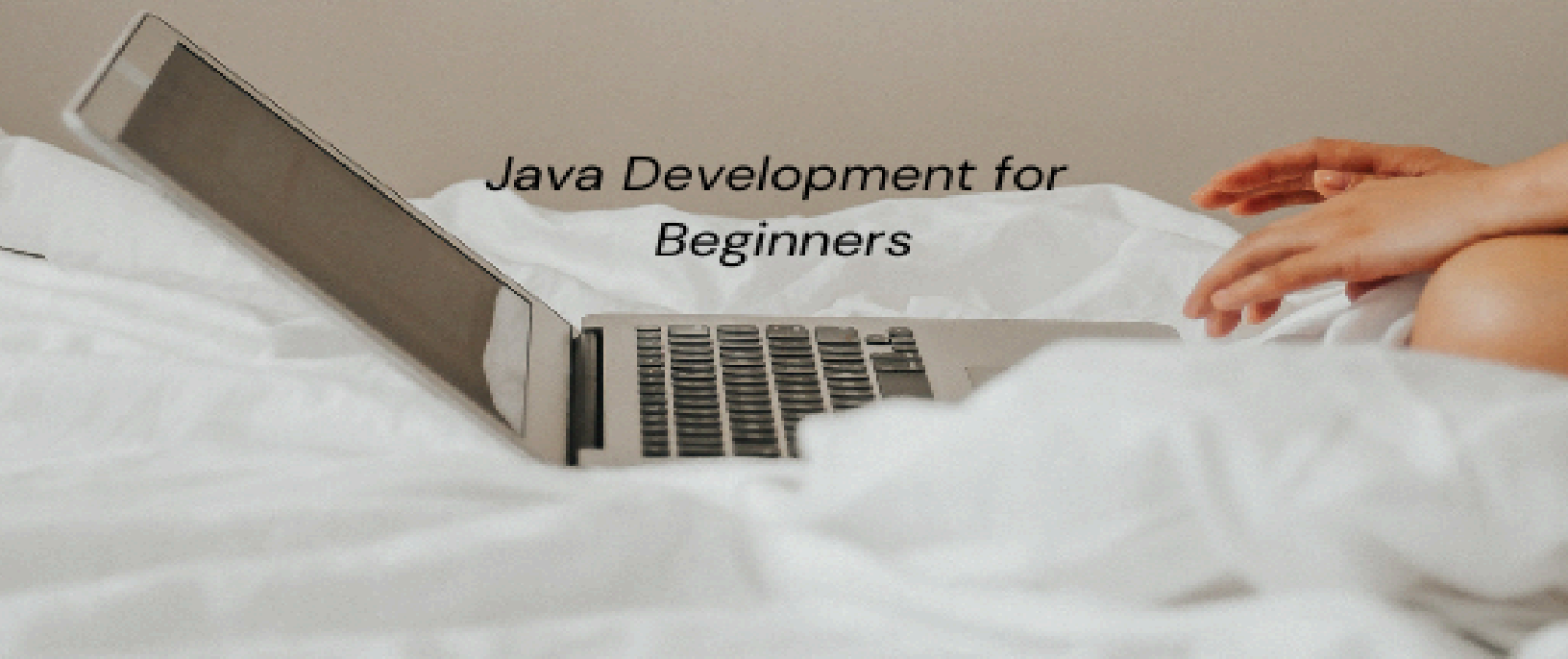
E-BOOK

JAVA PARA INICIANTES: FUNDAMENTOS DA PROGRAMAÇÃO E CONFORMIDADE COM NORMAS TÉCNICAS

Java Development for Beginners



PEDRO HENRIQUE ALVES PASSERINI
VITOR YOSHIAKI NISHIDA
PATRYCK SHOITY TAKIMOTO FARONI

Java para Iniciantes: Fundamentos da Programação e Conformidade com Normas Técnicas

Sinopse

Este e-book didático explora os conceitos iniciais da linguagem de programação Java, focando em sua aplicação prática e no entendimento dos mecanismos centrais, como a Máquina Virtual Java (JVM) e o Kit de Desenvolvimento Java (JDK). O conteúdo é organizado de forma progressiva, partindo da configuração do ambiente de desenvolvimento, passando pela sintaxe básica, tipos de dados primitivos, estruturas de controle de fluxo (if, else, switch, *loops* for, while, do-while), até culminar na introdução dos conceitos fundamentais da Orientação a Objetos (classes, objetos, métodos e encapsulamento). Códigos de exemplo detalhados acompanham cada tópico, servindo como base para a consolidação do conhecimento prático do leitor.

Palavras-Chave: Java, Programação Orientada a Objetos, Sintaxe, Algoritmos, JDK, Estruturas de Dados.

Sumário

B. ELEMENTOS TEXTUAIS: DESENVOLVIMENTO

1 INTRODUÇÃO

A linguagem Java destaca-se no cenário tecnológico global por sua robustez e versatilidade, sendo amplamente utilizada no desenvolvimento de aplicações corporativas, sistemas móveis (Android) e soluções de grande escala. Lançada pela Sun Microsystems (atualmente de propriedade da Oracle), a principal promessa do Java é a portabilidade, encapsulada no lema *"Write Once, Run Anywhere"* (WORA).

A característica WORA é alcançada através de uma arquitetura que insere uma camada de abstração entre o código e o hardware do sistema operacional. O código-fonte Java é primeiramente compilado em um formato intermediário chamado **Bytecode**. Este Bytecode não é executado diretamente pelo sistema operacional hospedeiro, mas sim pela **Máquina Virtual Java (JVM)**. É a JVM que interpreta o Bytecode em tempo de execução, permitindo que o mesmo código opere em diferentes plataformas sem necessidade de recompilação específica para cada sistema. Essa abordagem estabelece Java como uma linguagem compilada, mas que depende de uma interpretação para a execução final.

2 PREPARANDO O AMBIENTE DE DESENVOLVIMENTO

O desenvolvimento em Java exige a configuração de ferramentas específicas que trabalham em conjunto para compilar, executar e depurar as aplicações. Para o iniciante, o primeiro passo é a instalação do software necessário, tipicamente o Kit de Desenvolvimento Java.

2.1 O Trio Essencial: JDK, JRE e JVM

O ecossistema Java é sustentado por três componentes interconectados [3]:

1. **JDK (Java Development Kit):** O kit de ferramentas essencial para o desenvolvimento. Ele inclui o compilador (javac), o depurador e todas as ferramentas necessárias para criar aplicações Java.

2. **JRE (Java Runtime Environment):** O ambiente de tempo de execução. Contém as bibliotecas básicas e a JVM. É o que permite que um usuário final execute uma aplicação Java já compilada.
3. **JVM (Java Virtual Machine):** O componente crucial responsável por interpretar o Bytecode e executá-lo no sistema operacional hospedeiro, garantindo a portabilidade.[3]

2.2 Passo a Passo da Instalação e Configuração (JDK)

Para iniciar o desenvolvimento, é fundamental instalar o JDK, geralmente obtido diretamente do site da Oracle. A instalação manual do Java em um computador Windows, por exemplo, exige que o usuário tenha acesso de administrador.[4]

A instalação segue um processo padrão de *download* do instalador seguido pela execução do assistente. Uma vez concluída, o ambiente está tecnicamente presente no sistema, mas ainda é necessário configurá-lo para ser acessível globalmente a partir de qualquer diretório.

2.3 Configurando Variáveis de Ambiente (JAVA_HOME e PATH)

Um passo que frequentemente causa frustração em iniciantes é a falha em configurar as variáveis de ambiente.[3] O sistema operacional precisa saber onde as ferramentas do Java (como o compilador javac) estão localizadas, especialmente quando o desenvolvedor tenta executar comandos a partir de diretórios arbitrários no terminal.

A variável JAVA_HOME é utilizada para apontar para o diretório raiz da instalação do JDK.[3] Mais importante ainda, a variável PATH precisa ser atualizada para incluir o subdiretório bin do JDK. Se essa configuração for negligenciada, o sistema operacional não conseguirá encontrar o compilador ao digitar javac, resultando em erros de "comando não encontrado". O processo de configuração dessas variáveis estabelece o elo de ligação entre o sistema operacional e as ferramentas de desenvolvimento, transformando a instalação em um ambiente de trabalho funcional.

3 MEU PRIMEIRO PROGRAMA JAVA

O ponto de partida para qualquer linguagem de programação é o programa "Hello, World!", que demonstra a estrutura fundamental e o fluxo de execução de um aplicativo.

3.1 Estrutura Mínima de um Código

Todo programa Java deve residir dentro de uma classe, a menor unidade estrutural da linguagem. O código abaixo ilustra a estrutura básica [5]:

Quadro 3.1
Exemplo Básico de Programa Java

public class HelloWorld {
public static void main(String args) {
System.out.println("Hello, World!");
}
}

O bloco de código System.out.println("Hello, World!"); é a instrução que imprime o texto desejado no console. System.out representa o fluxo de saída padrão, e println é o método que imprime uma linha de texto.[5]

3.2 Compilação e Execução

Antes de ser executado, o arquivo deve ser salvo com o nome exato da classe seguido da extensão .java (neste caso, HelloWorld.java). Java é *case sensitive*, portanto, a correspondência exata é obrigatória.[5]

O processo segue duas etapas via terminal/prompt de comando [5]:

1. **Compilação:** O comando javac HelloWorld.java invoca o compilador, que traduz o código-fonte em Bytecode, gerando o arquivo HelloWorld.class.
2. **Execução:** O comando java HelloWorld invoca a JVM, que carrega e executa o Bytecode contido no arquivo .class.

3.3 Entendendo o Método main()

O método main() é o ponto de entrada obrigatório para qualquer aplicação Java autônoma (standalone).[5] A JVM inicia a execução do código a partir das instruções contidas dentro deste método.[6] Se a sintaxe estiver incorreta, o programa não será sequer executado.[8]

3.3.1 O Significado de public, static e void

Embora os iniciantes possam memorizar a assinatura do método (public static void main(String args)) inicialmente [7], a compreensão de seus modificadores é vital para avançar na Orientação a Objetos:

- **public:** Garante que o método possa ser chamado de qualquer lugar, incluindo pela JVM, que está fora da classe.[5]
- **static:** Permite que o método seja chamado sem a necessidade de criar uma instância (objeto) da classe HelloWorld. Como a JVM precisa de um ponto de partida antes de instanciar qualquer objeto, o método principal deve ser estático. Este modificador é a primeira pista da Orientação a Objetos para o leitor, demonstrando que a execução começa fora do contexto de um objeto específico.
- **void:** Indica que o método não retorna nenhum valor após sua execução.[5]
- **String args:** É um array de *strings* que recebe argumentos passados ao programa via linha de comando.

4 FUNDAMENTOS DA LINGUAGEM: VARIÁVEIS E OPERADORES

Uma das características de Java é ser uma **linguagem estaticamente tipada**.[9] Isso significa que toda variável deve ter seu tipo de dado declarado explicitamente antes de ser utilizada, e essa verificação de tipo ocorre em tempo de compilação.

4.1 Tipos de Dados em Java: Primitivos vs. Referência

Os tipos de dados são divididos em duas categorias: primitivos e de referência. Os tipos primitivos são os mais simples e armazenam valores individuais. Java possui 8 tipos primitivos.[9]

4.1.1 Detalhando os Tipos Primitivos

A diversidade de tipos inteiros, como byte, short, int e long, reflete um princípio de *software engineering*: a otimização de memória.[9] Em cenários onde se sabe que o valor guardado será pequeno (e.g., sistemas embarcados), o uso de byte economiza recursos em comparação com int.[9]

Tabela 4.1
Resumo dos Tipos de Dados Primitivos em Java

Tipo de Dado	**Tamanho (bits)**	**Descrição**	**Valor Mínimo/Máximo**
byte	8	Inteiro pequeno	-128 a 127 [9]
short	16	Inteiro curto	-32.768 a 32.767 [9]
int	32	Inteiro padrão (mais comum)	$\pm$ 2 bilhões [9]
long	64	Inteiro longo	Valores muito

			grandes
float	32	Ponto flutuante de precisão simples	(Decimal)
double	64	Ponto flutuante de precisão dupla (padrão)	(Decimal de alta precisão)
char	16	Caractere Unicode	(Caractere único)
boolean	1	Valor lógico	true ou false [10]

4.2 Declaração e Inicialização de Variáveis

A sintaxe para declarar uma variável é simples, exigindo o tipo de dado seguido pelo nome da variável. A inicialização pode ser feita no momento da declaração:

Java

```
// Exemplos de declaração e inicialização
byte a;
char t;
int valor;
int dia = 20;
char sexo = 'F';
String nome = "Fábio";
```

O tipo String é um exemplo de tipo de dado de **referência**, enquanto os demais são primitivos.[11]

4.3 Regras de Nomeação e Boas Práticas (Camel Case)

A legibilidade e manutenibilidade do código dependem diretamente do respeito às convenções de nomeação. As variáveis em Java devem seguir as seguintes regras e convenções [12]:

- Podem conter letras, números e o caractere sublinhado (_), mas não podem começar com um número.
- **Convenção:** Devem ser declaradas em minúsculo. Caso sejam nomes compostos, utiliza-se o padrão **Camel Case** (a primeira letra de todas as palavras, exceto a primeira, deve ser maiúscula), como em _salarioBase.[12]
- Java é *case sensitive*: a variável numeroUm é tratada de forma diferente de numero um.Seguir rigorosamente as regras e convenções é crucial para evitar erros de compilação e garantir a clareza do código a longo prazo.

4.4 Operadores Essenciais

Os operadores são símbolos que instruem o compilador a executar operações matemáticas ou lógicas específicas.

4.4.1 Operadores Aritméticos e de Atribuição

Os operadores aritméticos básicos são [13]: + (Adição), - (Subtração), * (Multiplicação), / (Divisão) e % (Resto ou Módulo).

Java também suporta operadores de atribuição composta, que são atalhos para operações matemáticas seguidas de atribuição. Por exemplo, x += 5 é equivalente a x = x + 5.[13]

4.4.2 Operadores de Comparação e Lógicos

Operadores de Comparação retornam um valor booleano (true ou false) e incluem: == (Igualdade), != (Diferente), > (Maior que), < (Menor que), >= (Maior ou igual a), e <= (Menor ou igual a).[13]

Operadores Lógicos são usados para combinar múltiplas expressões booleanas: && (E

lógico), || (OU lógico) e ! (NÃO lógico).[14]

5 CONTROLE DE FLUXO E REPETIÇÃO

As estruturas de controle de fluxo são elementos essenciais que permitem aos programas tomar decisões e executar código de forma condicional ou repetitiva.[15]

5.1 Estruturas Condicionais: Tomada de Decisão

As estruturas condicionais determinam qual bloco de código será executado com base na avaliação de uma expressão booleana.

5.1.1 O Bloco if e else

A estrutura if verifica uma condição: se ela for avaliada como true, o código dentro do bloco é executado. O bloco else é opcional e fornece um caminho alternativo a ser executado caso a condição seja false.[15]

Quadro 5.1
Exemplo if-else (Maioridade)

public class ControleIfElse {
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.");
}
}
}
```

5.1.2 Múltiplas Condições: else if

Para encadear múltiplas condições, o else if é utilizado, permitindo testar sequencialmente diversas possibilidades até que uma condição verdadeira seja encontrada.[15]

5.1.3 O Seletor switch

A estrutura switch é utilizada para selecionar entre vários blocos de código com base no valor de uma única variável. É altamente recomendável que, ao utilizar o switch, o desenvolvedor sempre inclua um bloco default.[17] A inclusão do default assegura que o programa tenha uma saída válida e controlada (nem que seja uma mensagem de erro) mesmo que a variável contenha um valor inesperado, aumentando a robustez do sistema.[17]

5.2 Estruturas de Repetição (Loops)

As estruturas de repetição permitem que um bloco de código seja executado múltiplas vezes.

5.2.1 O Loop for

O for é o mais indicado quando o número de iterações necessárias é **fixo e conhecido** de antemão. Sua sintaxe inclui a inicialização, a condição de término e o

incremento/decremento.[18]

5.2.2 O Loop while

O loop while é preferido quando o número de iterações é **desconhecido** e o loop deve continuar enquanto uma condição booleana permanecer verdadeira. A condição é verificada *antes* da execução do corpo do loop. Se a condição for falsa desde o início, o bloco de código jamais será executado.[18]

5.2.3 O Loop do-while

A principal distinção do do-while é que ele garante que o bloco de código seja executado **pelo menos uma vez**, independentemente da condição. A condição de repetição é verificada *após* a primeira execução do código.[18]

Quadro 5.2
Exemplo do-while (Soma de Positivos)

import java.util.Scanner;
class Main {
public static void main(String args) {
int sum = 0;
int number = 0;
Scanner input = new Scanner(System.in);
do {
sum += number;

```
System.out.println("Digite um número (negativo para sair):");
number = input.nextInt();
} while(number >= 0);
System.out.println("Soma = " + sum);
input.close();
}
}
```

A estruturação do código requer a preferência por condições claras e objetivas, o que é fundamental para evitar a duplicação de código e confusão, mesmo em códigos de iniciante.[17]

6 ESTRUTURAS DE DADOS FUNDAMENTAIS

Estruturas de dados são cruciais para organizar informações de forma eficiente em memória.

6.1 Arrays: Estruturas Estáticas

Arrays são coleções de elementos do mesmo tipo, armazenados em posições de memória contíguas. Uma característica fundamental dos arrays em Java é que eles possuem **tamanho fixo**, definido no momento da inicialização.[20]

A declaração básica é feita especificando o tipo de dado e o nome do array, seguido de colchetes: tipo nomeDoArray;.[20]

A inicialização define o tamanho do array: nomeDoArray = new tipo[tamanho];.[20]

Os elementos são acessados através de seu índice, que começa em 0. Para exibir o primeiro elemento de um array chamado numeros, utiliza-se: System.out.println(numeros);.[20] O

tamanho do array é obtido pelo atributo length.

6.2 Introdução ao ArrayList: Estruturas Dinâmicas

A principal limitação dos arrays (tamanho fixo) é resolvida pelo uso de coleções dinâmicas, das quais o ArrayList é a primeira e mais comum que o iniciante deve dominar. O ArrayList faz parte do *Collections Framework* de Java e reside no pacote java.util.[21] Ele é uma implementação de matriz redimensionável, oferecendo armazenamento dinâmico.

A transição de arrays estáticos para o ArrayList é o primeiro passo para o entendimento da arquitetura de classes de referência em Java.

6.2.1 Importando e Criando um ArrayList

Para usar o ArrayList, é necessário importá-lo: import java.util.ArrayList;.[21]

Ao criar um ArrayList, é obrigatório usar **Wrapper Classes** (classes empacotadoras) em vez dos tipos primitivos. Por exemplo, deve-se usar Integer em vez de int, e String em vez de string (embora String já seja uma classe de referência).[22] Isso ocorre porque as *Collections* em Java são projetadas para trabalhar apenas com objetos, e não com tipos primitivos.

Criação: ArrayList<String> fruits = new ArrayList<>();.[21]

6.2.2 Operações Básicas: Adicionar, Remover e Acessar

O ArrayList fornece métodos convenientes para manipulação de elementos [21]:

- add(): Usado para inserir elementos.
- remove(): Usado para excluir um elemento.
- get(index): Usado para recuperar um elemento pelo seu índice.

Quadro 6.1
Exemplo Básico de ArrayList

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

```
public class BasicArrayListExample {
public static void main(String args) {
ArrayList<String> frutas = new ArrayList<>();
frutas.add("Maçã");
frutas.add("Banana");
frutas.add("Laranja");
System.out.println("Frutas: " + frutas);
frutas.remove("Banana");
System.out.println("Após remoção: " + frutas);
System.out.println("Primeira fruta: " + frutas.get(0));
}
}
```

7 INTRODUÇÃO À ORIENTAÇÃO A OBJETOS (OO)

Java é fundamentalmente uma linguagem Orientada a Objetos. A compreensão de Classes, Objetos, Atributos e Métodos é indispensável para a evolução do desenvolvedor.

7.1 Classes e Objetos: O Molde e a Instância

A Orientação a Objetos é baseada na ideia de modelar entidades do mundo real.

Uma **Classe** funciona como um molde, ou planta baixa, que define as propriedades e comportamentos comuns a um conjunto de entidades.[24]

Um **Objeto** é uma instância concreta da classe. Ele é criado a partir do molde usando a palavra-chave new, seguida pelo construtor da classe.[24]

Exemplo de instanciação: Carro meuCarro = new Carro("Sedan", 2024);

7.2 Atributos (Estado) e Métodos (Comportamento)

Os objetos em Java são definidos por dois componentes principais [25]:

1. **Atributos:** São as variáveis declaradas dentro da classe que definem o estado do objeto. Por exemplo, em uma classe Carro, modelo e ano seriam atributos. Cada objeto instanciado terá seus próprios valores para esses atributos.[25]
2. **Métodos:** São blocos de código que definem o comportamento ou as ações que o objeto pode realizar. Por exemplo, o método ligar() em um objeto Carro.[25]

7.3 Entendendo Métodos: Ação e Retorno

Em Java, que é uma linguagem Orientada a Objetos, o termo correto para rotinas de código que pertencem a uma classe é **método**. O termo **função** é geralmente reservado para linguagens de programação procedurais (como C).[26] Como em Java, a essência do desenvolvimento é sempre ligada a uma classe ou objeto, o comportamento está intrinsecamente ligado a essa entidade.[27]

7.4 Construtores: Inicializando Objetos

O **Construtor** é um tipo especial de método que possui o mesmo nome da classe e não retorna valor (nem mesmo void). Sua finalidade é inicializar o objeto no momento de sua criação (new).[28]

Se o desenvolvedor não definir explicitamente um construtor, Java fornece automaticamente um **construtor padrão (implícito)**, que inicializa todos os atributos com seus valores padrão

(0 para numéricos, null para referências e false para booleanos).[28] Contudo, definir um construtor **explícito** permite um controle total sobre o processo de configuração inicial dos objetos, garantindo que eles sejam criados em um estado válido.[28]

7.5 Encapsulamento: Getters e Setters

O encapsulamento é um dos pilares da Orientação a Objetos e se refere ao princípio de proteger o estado interno de um objeto, controlando o acesso aos seus dados.

Isso é alcançado definindo os atributos internos da classe como private (privados), o que restringe o acesso direto de códigos externos.[29] Para permitir a leitura e modificação desses atributos de forma controlada, são criados métodos públicos: **Getters** (para obter o valor) e **Setters** (para definir o valor).[25]

O encapsulamento não é apenas uma forma de organização, mas uma ferramenta de segurança e manutenibilidade. Ao forçar o acesso através de métodos, o desenvolvedor pode incluir lógica de validação dentro dos métodos *setter* antes de permitir que o estado do objeto seja alterado.

Quadro 7.1
Exemplo de Classe, Atributos e Encapsulamento

public class Carro {
private String modelo; // Atributo privado, protegido
public int ano; // Atributo público
// Getter: Permite ler o atributo 'modelo'
public String getModelo() {
return modelo;
}

```
// Setter: Permite definir o atributo 'modelo' com validação
public void setModelo(String modelo) {
this.modelo = modelo;
}

public void ligar() {
System.out.println("O carro " + modelo + " está ligado.");
}
}
```

A Tabela 7.1 resume a visibilidade dos principais modificadores de acesso [29]:

Tabela 7.1
Modificadores de Acesso em Java (Visibilidade)

Modificador	**Mesma Classe**	**Mesmo Pacote**	**Pacotes Diferentes**
private	Sim	Não	Não
public	Sim	Sim	Sim

8 CONSIDERAÇÕES FINAIS E PRÓXIMOS PASSOS

O domínio dos fundamentos apresentados neste e-book — desde a configuração do ambiente (JDK/JVM) e a compreensão da sintaxe básica (variáveis, operadores, controle de

fluxo) até a introdução à Orientação a Objetos (Classes, Métodos, Encapsulamento) — estabelece uma base sólida para a jornada de desenvolvimento em Java.

A progressão didática, que move o leitor de estruturas estáticas (Arrays) para estruturas dinâmicas (ArrayList), ilustra como as classes de referência em Java resolvem as limitações dos tipos primitivos. De maneira análoga, a distinção entre métodos e funções reforça a natureza inerentemente orientada a objetos da linguagem.

Para o aprofundamento do conhecimento, recomenda-se que o iniciante explore os seguintes tópicos de forma progressiva:

1. **Pilares Avançados de OO:** Herança, Polimorfismo e Abstração.
2. **Tratamento de Exceções:** Uso de try-catch para lidar com erros em tempo de execução.
3. **IO e Arquivos:** Leitura e escrita de dados em sistemas de arquivos.
4. **Frameworks:** Estudar ambientes como Spring Boot, amplamente utilizados para desenvolvimento de aplicações *backend* corporativas.

REFERÊNCIAS (ABNT NBR 6023)

A listagem de referências a seguir inclui todas as fontes utilizadas para a elaboração e validação técnica e normativa deste e-book.

BOSON TREINAMENTOS. **Declaração de variáveis em Java**. Disponível em: https://www.bosontreinamentos.com.br/java/declaracao-de-variaveis-em-java/. Acesso em: 10 out. 2024..[11]

CODEGYM. **Os modificadores do método main()**. Disponível em: https://codegym.cc/pt/groups/posts/pt.1002.mtodo-java-main. Acesso em: 10 out. 2024..[8]

COMOPROGRAMARJAVA. **Arrays em Java: declaração e uso**. Disponível em: https://comoprogramarjava.com.br/arrays-em-java-declaracao-e-uso/. Acesso em: 10 out. 2024..[20]

DATACAMP. **ArrayList em Java**. Disponível em: https://www.datacamp.com/pt/doc/java/arraylist. Acesso em: 10 out. 2024..[21]

DATACAMP. **Creating your first Java program - Hello World**. Disponível em: https://www.datacamp.com/doc/java/first-java-program-hello-world. Acesso em: 10 out. 2024..[5]

DATACAMP. **Java Operators**. Disponível em:

https://www.datacamp.com/pt/doc/java/java-operators. Acesso em: 10 out. 2024..[14]

DATACAMP. **Classes and Objects**. Disponível em: https://www.datacamp.com/pt/doc/java/classes-and-objects. Acesso em: 10 out. 2024..[24]

DEV.TO. **Classe e Instância de Objetos em Java**. Disponível em: https://dev.to/emanoelcarvalho/classe-e-instancia-de-objetos-em-java-6a6. Acesso em: 10 out. 2024..[28]

DEV.TO. **Tipos de Dados Primitivos com Java**. Disponível em: https://dev.to/monokai_dev/tipos-de-dados-primitivos-com-java-3o4g. Acesso em: 10 out. 2024..[9]

DEVMEDIA. **Java: variáveis e constantes**. Disponível em: https://www.devmedia.com.br/java-variaveis-e-constantes/38311. Acesso em: 10 out. 2024..[12]

DEVMEDIA. **Métodos, atributos e classes no Java: Modificadores de acesso**. Disponível em: https://www.devmedia.com.br/metodos-atributos-e-classes-no-java/25404. Acesso em: 10 out. 2024..[29]

DEVMEDIA. **Trabalhando com arrays em Java**. Disponível em: https://www.devmedia.com.br/trabalhando-com-arrays-em-java/25530. Acesso em: 10 out. 2024..[30]

DIO. **Classes, atributos, métodos, pacotes**. Disponível em: https://www.dio.me/articles/classes-atributos-metodos-pacotes. Acesso em: 10 out. 2024..[25]

DIO. **Operadores em Java: Um Guia Completo para Desenvolvedores Iniciantes**. Disponível em: https://www.dio.me/articles/operadores-em-java-um-guia-completo-para-desenvolvedores-iniciantes. Acesso em: 10 out. 2024..[13]

GASPAR BARANCELLI. **Estruturas de Controle de Fluxo em Java: If, Else e Switch Explicados**. Disponível em: https://gasparbarancelli.com/post/estruturas-de-controle-de-fluxo-em-java-if,-else-e-switch-explicados. Acesso em: 10 out. 2024..[15]

IONOS. **Método Java main: O que é e como funciona**. Disponível em: https://www.ionos.com/pt-br/digitalguide/sites-de-internet/desenvolvimento-web/metodo-java-main/. Acesso em: 10 out. 2024..[6]

IONOS. **Java primitives: Tudo sobre dados primitivos em Java**. Disponível em: https://www.ionos.com/pt-br/digitalguide/sites-de-internet/desenvolvimento-web/java-primitiv

es/. Acesso em: 10 out. 2024..[10]

JAVA.COM. **Como posso baixar e instalar manualmente o Java em meu computador Windows?** Disponível em: https://www.java.com/pt-br/download/help/windows_manual_download.html. Acesso em: 10 out. 2024..[4]

MAURÍCIO GENEROSO. **Configurando Java #1: Instalação do JRE e do JDK no Windows**. Disponível em: https://mauriciogeneroso.medium.com/configurando-java-1-instala%C3%A7%C3%A3o-do-jre-e-do-jdk-no-windows-38cacace0377. Acesso em: 10 out. 2024..[3]

PROGRAMIZ. **Hello World Program in Java**. Disponível em: https://www.programiz.com/java-programming/hello-world. Acesso em: 10 out. 2024..[7]

PROGRAMIZ. **Java do...while Loop**. Disponível em: https://www.programiz.com/java-programming/do-while-loop. Acesso em: 10 out. 2024..[19]

PROGRAMIZ. **Java ArrayList**. Disponível em: https://www.programiz.com/java-programming/arraylist. Acesso em: 10 out. 2024..[22]

SCHOLARHAT. **Java Loops: For, While, Do-while**. Disponível em: https://www.scholarhat.com/tutorial/java/java-loops-for-while-do-while. Acesso em: 10 out. 2024..[18]

UFAPE. **Guia de Orientação de Normalização para Trabalhos Acadêmicos**. 2025..

Referências citadas

1. Normas ABNT: Regras de formatação para trabalhos acadêmicos - Blog da Mettzer, acessado em novembro 12, 2025, https://blog.mettzer.com/normas-abnt/
2. Espaçamento ABNT: Veja a importância e como fazê-lo certo - Blog da Mettzer, acessado em novembro 12, 2025, https://blog.mettzer.com/espacamento-abnt/
3. Configurando Java #1: Instalação do JRE e do JDK no Windows | by Maurício Generoso, acessado em novembro 12, 2025, https://mauriciogeneroso.medium.com/configurando-java-1-instala%C3%A7%C3%A3o-do-jre-e-do-jdk-no-windows-38cacace0377
4. Como posso baixar e instalar manualmente o Java em meu computador Windows?, acessado em novembro 12, 2025, https://www.java.com/pt-br/download/help/windows_manual_download.html
5. First Java Program: Hello World - DataCamp, acessado em novembro 12, 2025, https://www.datacamp.com/doc/java/first-java-program-hello-world
6. O que é e como funciona o método Java main() - IONOS, acessado em novembro 12, 2025, https://www.ionos.com/pt-br/digitalguide/sites-de-internet/desenvolvimento-web

/metodo-java-main/
7. Hello World - Your First Java Program - Programiz, acessado em novembro 12, 2025, https://www.programiz.com/java-programming/hello-world
8. Método Java main() - CodeGym, acessado em novembro 12, 2025, https://codegym.cc/pt/groups/posts/pt.1002.mtodo-java-main
9. Tipos de dados primitivos com Java - DEV Community, acessado em novembro 12, 2025, https://dev.to/monokai_dev/tipos-de-dados-primitivos-com-java-3o4g
10. Java primitives: Tipos de dados primitivos em Java - IONOS, acessado em novembro 12, 2025, https://www.ionos.com/pt-br/digitalguide/sites-de-internet/desenvolvimento-web/java-primitives/
11. Declaração de Variáveis em Java - Bóson Treinamentos, acessado em novembro 12, 2025, https://www.bosontreinamentos.com.br/java/declaracao-de-variaveis-em-java/
12. O que são variáveis e constantes no Java? Descubra agora! - DevMedia, acessado em novembro 12, 2025, https://www.devmedia.com.br/java-variaveis-e-constantes/38311
13. Operadores em Java: Um Guia Completo para Desenvolvedores Iniciantes | Lilian Rodrigues | Operadores lógicos | Java | DIO, acessado em novembro 12, 2025, https://www.dio.me/articles/operadores-em-java-um-guia-completo-para-desenvolvedores-iniciantes
14. Operadores Java - DataCamp, acessado em novembro 12, 2025, https://www.datacamp.com/pt/doc/java/java-operators
15. Estruturas de Controle de Fluxo em Java: if, else e switch Explicados - Gaspar Barancelli, acessado em novembro 12, 2025, https://gasparbarancelli.com/post/estruturas-de-controle-de-fluxo-em-java-if,-else-e-switch-explicados
16. Estruturas de Controle e Repetição em Java: while, do-while, if, else e switch case - DIO, acessado em novembro 12, 2025, https://www.dio.me/articles/estruturas-de-controle-e-repeticao-em-java-while-do-while-if-else-e-switch-case-928acdb59b36
17. Java Course #09 - Conditional Structures in Java (If, Else, Switch) - YouTube, acessado em novembro 12, 2025, https://www.youtube.com/watch?v=SLBwd5ONgNU
18. Looping Statements in Java - For, While, Do-While Loop in Java - ScholarHat, acessado em novembro 12, 2025, https://www.scholarhat.com/tutorial/java/java-loops-for-while-do-while
19. Java while and do...while Loop - Programiz, acessado em novembro 12, 2025, https://www.programiz.com/java-programming/do-while-loop
20. Arrays em Java: Declaração e Uso - Como Programar Java, acessado em novembro 12, 2025, https://comoprogramarjava.com.br/arrays-em-java-declaracao-e-uso/
21. Java ArrayList - DataCamp, acessado em novembro 12, 2025, https://www.datacamp.com/pt/doc/java/arraylist
22. Java ArrayList (With Examples) - Programiz, acessado em novembro 12, 2025,

https://www.programiz.com/java-programming/arraylist
23. Java ArrayList - DataCamp, acessado em novembro 12, 2025, https://www.datacamp.com/es/doc/java/arraylist
24. Classes e objetos Java - DataCamp, acessado em novembro 12, 2025, https://www.datacamp.com/pt/doc/java/classes-and-objects
25. Classes, Atributos, Métodos, Pacotes | Lilian Rodrigues | Java - DIO, acessado em novembro 12, 2025, https://www.dio.me/articles/classes-atributos-metodos-pacotes
26. Qual a diferença entre uma função e um método? : r/learnjava - Reddit, acessado em novembro 12, 2025, https://www.reddit.com/r/learnjava/comments/u2mfky/what_is_the_difference_between_a_function_and_a/?tl=pt-br
27. Difereça entre métodos e funções | Java I: Primeiros passos | Solucionado - Alura, acessado em novembro 12, 2025, https://cursos.alura.com.br/forum/topico-difereca-entre-metodos-e-funcoes-36265
28. Classe e Instância de Objetos em Java - DEV Community, acessado em novembro 12, 2025, https://dev.to/emanoelcarvalho/classe-e-instancia-de-objetos-em-java-6a6
29. Métodos, atributos e classes no Java: Modificadores de acesso - DevMedia, acessado em novembro 12, 2025, https://www.devmedia.com.br/metodos-atributos-e-classes-no-java/25404
30. Como operar com Arrays em Java - DevMedia, acessado em novembro 12, 2025, https://www.devmedia.com.br/trabalhando-com-arrays-em-java/25530
31. Guia de Orientação de Normalização para Trabalhos Acadêmicos - 2025.pdf, acessado em novembro 12, 2025, https://ufape.edu.br/sites/default/files/2025-02/Guia%20de%20Orienta%C3%A7%C3%A3o%20de%20Normaliza%C3%A7%C3%A3o%20para%20Trabalhos%20Acad%C3%AAmicos%20-%202025.pdf