Introdução

Seja bem-vindo ao ebook de 200+ exercícios de Java! Com este material você irá se preparar para o mercado de trabalho e polir muito seu conhecimento em Java.

Preparei exercícios que são similares aos problemas que os desenvolvedores vivenciam nas empresas, então a grande maioria deles tem este foco no mercado de trabalho.

Os que não tem, visam trabalhar com partes que os programadores costumam ter dificuldade em Java, como conceitos mais avançados que utilizados na hora certa podem ser úteis demais.

Cada capítulo trata de um recurso de forma 'macro', sendo dividido em seções para explorar o recurso ao máximo. Como por exemplo o capítulo 4, que trabalha com controle de fluxo, ele tem as seções:

- If, else e else if;
- Switch;
- Loops;

Onde os exercícios têm normalmente dificuldade progressiva, para você se sentir desafiado.

> Quero te convidar também a conhecer meu **Curso Completo de Java**, que será um ótimo material aliado a este para você dominar a linguagem.

E para você que está lendo este ebook há uma condição especial, clique no link acima e veja como o curso é completo e como ele vai te ajudar a programar melhor em Java.

Todos os nossos cursos possuem valores acessíveis, suporte para dúvidas, certificado de conclusão, exercícios e projetos. Recursos que vão amplificar seu aprendizado.

Quer conteúdo gratuito e de qualidade de programação? Se inscreva no meu canal de YouTube e me siga no Instagram: @horadecodar =)

Dada a introdução, vamos prosseguir?

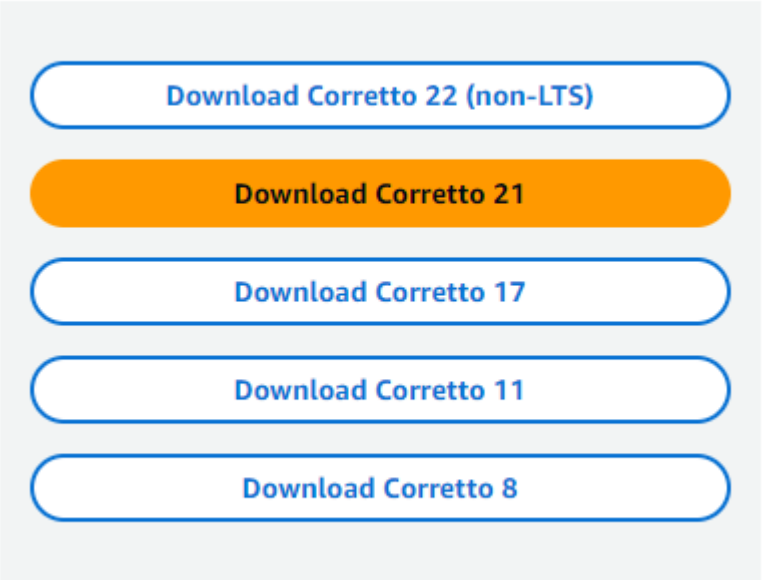
Capítulo 1: Configuração do Ambiente de Desenvolvimento

Se você já tem o Java instalado, passe para o segundo capítulo e comece os exercícios! =)

Instalando o Java (Amazon Corretto) no Windows

Passo 1: Baixar o Amazon Corretto

1. Acesse o site oficial do Amazon Corretto: https://aws.amazon.com/corretto/
2. Escolha a versão do Amazon Corretto que você deseja instalar.
3. Para um ambiente moderno e sem bugs, eu indico o Java 21 é a versão LTS (Long Term Support) ou a versão que estiver em LTS.
4. Clique no link de download correspondente à sua versão do Windows (64 bits).



Veja que a Amazon indica a versão LTS com o botão dourado.

Passo 2: Instalar o Amazon Corretto

1. Após o download, abra o instalador.
2. Siga as instruções do instalador, aceitando os termos de uso e selecionando a pasta de instalação padrão ou uma personalizada.
3. Certifique-se de que a opção "Set JAVA_HOME" esteja marcada, pois isso define a variável de ambiente necessária para que o sistema reconheça o Java.
4. Conclua a instalação clicando em "Install" e depois em "Finish".

Obs: Você pode deixar todas as opções marcadas, elas vão deixar o seu ambiente mais completo.

Passo 3: Verificar a Instalação do Java

1. Abra o Prompt de Comando:
2. Pressione Win + R, digite cmd e pressione Enter.
3. Digite o comando abaixo para verificar a versão instalada do Java:

```
java -version
```

Desta maneira:

```
C:\Users\mathe>java -version
openjdk version "21.0.4" 2024-07-16 LTS
OpenJDK Runtime Environment Corretto-21.0.4.7.1 (build 21.0.4+7-LTS)
OpenJDK 64-Bit Server VM Corretto-21.0.4.7.1 (build 21.0.4+7-LTS, mixed mode, sharing)
```

Se a instalação estiver correta, você verá algo como:

> openjdk version "21.0.x" 202x-xx-xx
> OpenJDK Runtime Environment Corretto-21.x.x.x (build 21.0.x+xx)
> OpenJDK 64-Bit Server VM Corretto-21.x.x.x (build 21.0.x+xx, mixed mode)

Instalando o Visual Studio Code (VS Code)

Passo 1: Baixar o VS Code

1. Acesse o site oficial do Visual Studio Code: https://code.visualstudio.com/
2. Clique em Download for Windows para baixar a versão do instalador para Windows.

Passo 2: Instalar o VS Code

1. Após o download, execute o instalador do VS Code.
2. Marque a opção "Add to PATH" durante a instalação para que o VS Code possa ser aberto diretamente pelo terminal.
3. Siga as instruções e conclua a instalação clicando em "Install".

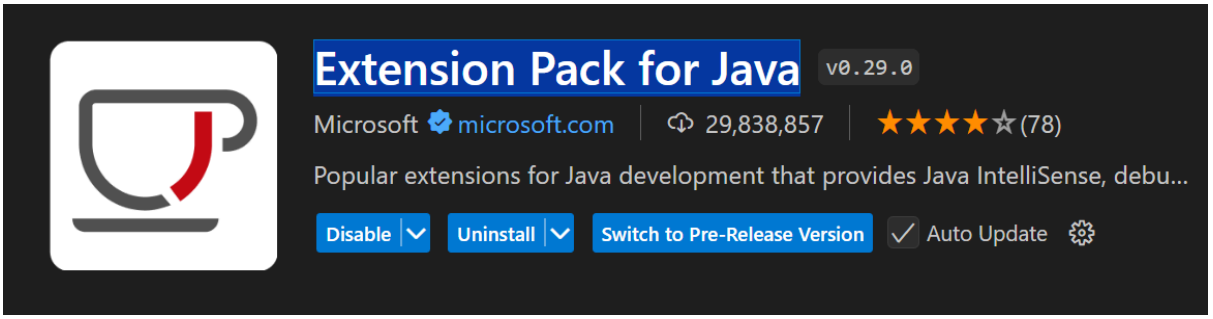
Configurando o Ambiente Java no VS Code

Passo 1: Abrir o VS Code

1. Abra o Visual Studio Code através do atalho no desktop ou pelo menu Iniciar.
2. Vá até a aba Extensões clicando no ícone de blocos no lado esquerdo do editor ou usando o atalho Ctrl + Shift + X.

Passo 2: Instalar o Extension Pack for Java

1. Na aba de extensões, pesquise por Extension Pack for Java.
2. Clique no resultado e, em seguida, clique no botão Instalar.



Esta é a extensão

Esta extensão instala automaticamente um conjunto de ferramentas essenciais para desenvolvimento Java no VS Code, incluindo:

- Language Support for Java (by Red Hat)
- Debugger for Java
- Java Test Runner
- Java Dependency Viewer
- Maven for Java

Isso vai fazer com que você consiga executar o Java pelo VS Code e ter highlight da linguagem, o que facilita as coisas.

Passo 3: Verificar a Instalação da Extensão

1. Após a instalação, reinicie o VS Code.
2. Para verificar se tudo está instalado corretamente, abra um novo terminal no VS Code (Ctrl + ~) e digite:

```
java -version
```

Se a instalação do Java e da extensão estiver correta, a versão do Java instalada será exibida no terminal.

Criando e Executando um Programa Java no VS Code

Passo 1: Criar um Novo Projeto Java

1. Crie uma pasta no seu computador
2. Abra o VS Code nela

Passo 2: Escrever o Código Java

Após a criação do projeto, crie um novo arquivo chamado Main.java na pasta.

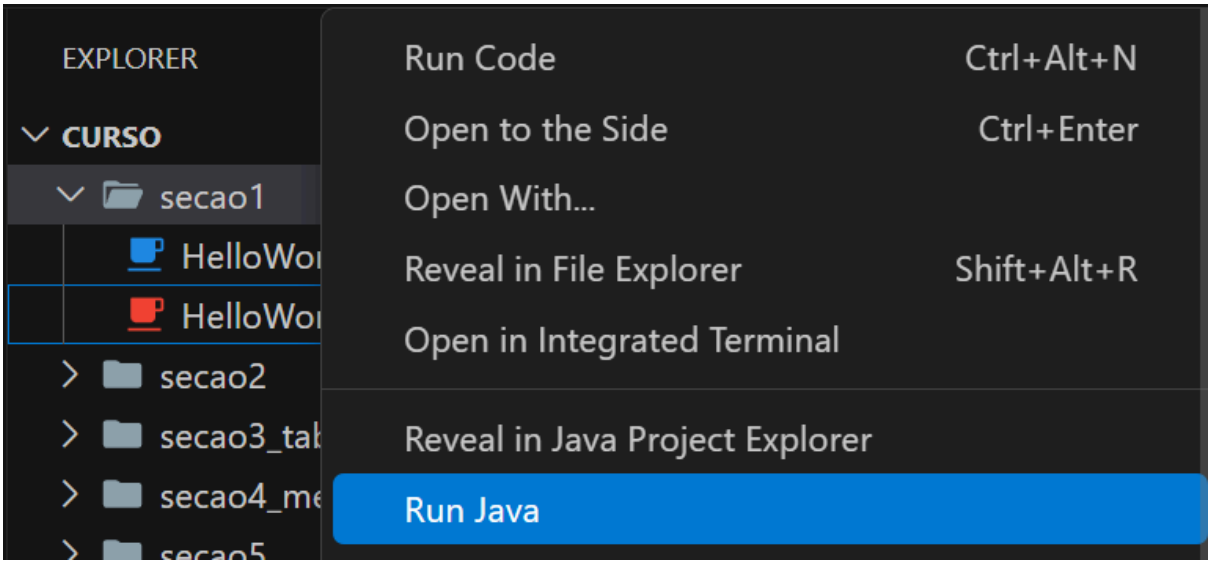
Adicione o código Java abaixo ao arquivo Main.java:

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

Passo 3: Compilar e Executar o Programa

Para compilar e executar, basta clicar no arquivo com o botão direito do mouse e escolher a opção Run Java

Veja:



Agora você já está apto a utilizar o Java no seu computador, vamos aos exercícios!

Nota importante

1) Aqui no nosso ebook eu fiz os exercícios separadamente por classes. Mas você pode unir vários em uma classe só, para não ficar tão burocrático.

Por exemplo: Divida todos os exercícios de cada capítulo em uma classe.

Os arquivos de conferência no GitHub estão organizados desta forma também.

2) Alguns exercícios solicitam os dados do usuário, utilize a classe Scanner para obter os dados.

> Quero te convidar também a conhecer meu **Curso Completo de Java**, que vai te fazer superar qualquer dificuldade nos exercícios deste ebook, e ainda está numa condição especial para os leitores deste material, aproveite! =)

Capítulo 2: Introdução ao Java

Neste capítulo, você dará os primeiros passos na linguagem de programação Java, uma das mais populares e versáteis do mundo. A jornada começa com a compreensão dos fundamentos que são essenciais para qualquer programador que deseja dominar Java.

O que você vai aprender?

Você será introduzido aos conceitos básicos, como a criação do seu primeiro programa, a importância dos comentários no código, e a declaração e manipulação de variáveis. Além disso, abordaremos operações aritméticas, a utilização de constantes, e como realizar a leitura e a exibição de dados no console.

Desafios que você enfrentará:

Neste capítulo, os desafios serão focados em entender e aplicar conceitos fundamentais. Você aprenderá a lidar com diferentes tipos de dados e variáveis, a realizar operações básicas, e a formatar a saída de dados de forma adequada. Alguns exercícios também irão desafiá-lo a entender as diferenças entre variáveis locais e globais, além de aprender a trabalhar com strings e a manipular a entrada de dados do usuário.

Ao concluir os exercícios deste capítulo, você estará familiarizado com a estrutura básica de um programa Java e entenderá como os diferentes tipos de dados e operações funcionam na prática. Esta base sólida é essencial para enfrentar desafios mais complexos nos próximos capítulos. Prepare-se para começar sua jornada na programação Java!

Exercício 1: Primeiros Passos

Crie um programa Java que exiba a mensagem "Olá, Mundo!" no console. Em seguida, modifique o programa para exibir seu nome.

Código de solução:

```java
public class PrimeirosPassos {
    public static void main(String[] args) {
        // Exibindo "Olá, Mundo!" no console
        System.out.println("Olá, Mundo!");

        // Modificando para exibir o nome
        System.out.println("Meu nome é Matheus.");
    }
}
```

Explicação: Neste exercício, você cria um programa simples que exibe uma mensagem no console usando System.out.println. Primeiro, a mensagem "Olá, Mundo!" é exibida, e depois é modificada para exibir seu nome. Este exercício introduz a estrutura básica de um programa Java e a função de saída no console.

Exercício 2: Comentários no Código

Escreva um programa Java que contenha três tipos de comentários: comentário de linha, comentário de bloco e comentário de documentação. Explique brevemente o uso de cada um desses comentários dentro do código.

Código de solução:

```java
public class Comentarios {
    public static void main(String[] args) {
        // Este é um comentário de linha, usado para comentar uma única
linha

        /*
         * Este é um comentário de bloco,
         * pode ser usado para comentar várias linhas
         */

        /**
         * Este é um comentário de documentação,
         * usado para gerar documentação automática
         * usando ferramentas como o Javadoc
         */
        System.out.println("Comentários adicionados ao código!");
    }
}
```

Explicação: Este exercício demonstra o uso de três tipos de comentários em Java. Comentários de linha (//) são usados para breves anotações em uma única linha.

Comentários de bloco (/* ... */) são utilizados para comentar várias linhas de código. Comentários de documentação (/** ... */) são empregados para gerar documentação com ferramentas como o Javadoc.

Exercício 3: Variáveis e Tipos de Dados

Crie um programa que declare e inicialize variáveis de todos os tipos primitivos em Java (int, double, char, boolean, etc.). Exiba o valor de cada variável no console.

Código de solução:

```
public class TiposDeDados {
    public static void main(String[] args) {
        int idade = 25;
        double altura = 1.75;
        char inicial = 'M';
        boolean estudante = true;

        System.out.println("Idade: " + idade);
        System.out.println("Altura: " + altura);
        System.out.println("Inicial do nome: " + inicial);
        System.out.println("É estudante? " + estudante);
    }
}
```

Explicação: Neste exercício, você declara variáveis de diferentes tipos primitivos em Java (int, double, char, boolean) e inicializa com valores. Em seguida, o programa exibe os valores dessas variáveis no console usando System.out.println. Este exercício ajuda a compreender a utilização de diferentes tipos de dados em Java.

Exercício 4: Conversão de Tipos

Escreva um programa que converta um valor double em int e outro valor int em double. Exiba os resultados das conversões e explique a diferença entre conversão explícita e implícita.
Dica: procure por type casting em Java.

Código de solução:

```
public class ConversaoDeTipos {
    public static void main(String[] args) {
        double valorDouble = 9.99;
```

```
        int valorInt = (int) valorDouble;  // Conversão explícita de
double para int

        int numero = 10;
        double numeroConvertido = numero;  // Conversão implícita de int
para double

        System.out.println("Valor double: " + valorDouble);
        System.out.println("Valor convertido para int: " + valorInt);
        System.out.println("Número int: " + numero);
        System.out.println("Número convertido para double: " +
numeroConvertido);
    }
}
```

Explicação: Este exercício demonstra a conversão de tipos em Java. A conversão de double para int é feita explicitamente, truncando o valor decimal. A conversão de int para double é implícita, e o valor é convertido sem perda de informação. Este exercício reforça o conceito de conversão de tipos e suas implicações.

Exercício 5: Operações Aritméticas

Desenvolva um programa que declare duas variáveis int e realize as operações de soma, subtração, multiplicação, divisão e módulo entre elas. Exiba os resultados de cada operação.

Código de solução:

```
public class OperacoesAritmeticas {
    public static void main(String[] args) {
        int a = 10;
        int b = 3;

        int soma = a + b;
        int subtracao = a - b;
        int multiplicacao = a * b;
        int divisao = a / b;
        int modulo = a % b;

        System.out.println("Soma: " + soma);
        System.out.println("Subtração: " + subtracao);
        System.out.println("Multiplicação: " + multiplicacao);
        System.out.println("Divisão: " + divisao);
        System.out.println("Módulo: " + modulo);
```

```
    }
}
```

Explicação: Neste exercício, são realizadas operações aritméticas básicas (+, -, *, /, %) entre duas variáveis inteiras. O resultado de cada operação é exibido no console. Este exercício ajuda a solidificar o entendimento das operações matemáticas em Java e como utilizá-las em variáveis.

Exercício 6: Constantes

Crie um programa que utilize a palavra-chave final para declarar uma constante que representa a velocidade da luz no vácuo. Tente alterar o valor da constante e observe o comportamento do compilador.

Código de solução:

```
public class Constantes {
    public static void main(String[] args) {
        final double VELOCIDADE_DA_LUZ = 299792458; // em metros por segundo

        System.out.println("Velocidade da luz: " + VELOCIDADE_DA_LUZ + " m/s");

        // VELOCIDADE_DA_LUZ = 300000000; // Esta linha causará um erro de compilação
    }
}
```

Explicação: Neste exercício, você declara uma constante usando a palavra-chave final, que impede a modificação do valor após a inicialização. A tentativa de alterar a constante resultará em um erro de compilação, reforçando o conceito de imutabilidade associado às constantes em Java.

Exercício 7: Entrada de Dados

Escreva um programa que leia um número inteiro e um número decimal do teclado e, em seguida, exiba a soma desses números no console.

Dica: utilize o pacote/classe Scanner.

Código de solução:

```java
import java.util.Scanner;

public class EntradaDeDados {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int numeroInt = scanner.nextInt();

        System.out.print("Digite um número decimal: ");
        double numeroDouble = scanner.nextDouble();

        double soma = numeroInt + numeroDouble;
        System.out.println("A soma dos números é: " + soma);
    }
}
```

Explicação: Este exercício introduz a leitura de dados do usuário utilizando a classe Scanner. O programa lê um número inteiro e um número decimal, realiza a soma e exibe o resultado no console. Este exercício é útil para entender como lidar com a entrada de dados em Java.

Exercício 8: Strings e Concatenação

Crie um programa que peça ao usuário para digitar seu nome e sobrenome. O programa deve exibir uma mensagem de boas-vindas concatenando o nome e o sobrenome do usuário.

Código de solução:

```java
import java.util.Scanner;

public class ConcatenaStrings {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite seu nome: ");
        String nome = scanner.nextLine();

        System.out.print("Digite seu sobrenome: ");
        String sobrenome = scanner.nextLine();

        String mensagem = "Bem-vindo, " + nome + " " + sobrenome + "!";
        System.out.println(mensagem);
```

```
    }
}
```

Explicação: Neste exercício, você aprende a trabalhar com strings e concatenação em Java. O programa solicita ao usuário que digite seu nome e sobrenome, e em seguida, exibe uma mensagem de boas-vindas. A concatenação de strings é uma operação comum e fundamental no desenvolvimento de software.

Exercício 9: Tipos de Variáveis

Escreva um programa que declare variáveis locais e globais (dentro de uma classe). Inicialize e exiba o valor de ambas as variáveis no console.

Dica: as variáveis globais ficam fora do método main.

Código de solução:

```
public class TiposDeVariaveis {
    // Variável global
    static int variavelGlobal = 10;

    public static void main(String[] args) {
        // Variável local
        int variavelLocal = 5;

        System.out.println("Valor da variável global: " +
variavelGlobal);
        System.out.println("Valor da variável local: " + variavelLocal);
    }
}
```

Explicação: Este exercício demonstra a diferença entre variáveis globais e locais. As variáveis globais são acessíveis em qualquer parte da classe, enquanto as variáveis locais só existem dentro do escopo onde foram declaradas. O código exibe os valores de ambas as variáveis no console.

Exercício 10: Formatação de Saída

Desenvolva um programa que exiba o valor de uma variável double com duas casas decimais. Utilize formatação para garantir que o valor seja exibido corretamente.

Código de solução:

```java
public class FormatacaoDeSaida {
    public static void main(String[] args) {
        double valor = 123.456789;

        System.out.printf("Valor formatado: %.2f%n", valor);
    }
}
```

Explicação: Neste exercício, você aprende a formatar a saída de um valor numérico usando System.out.printf. O código exibe o valor de uma variável double com apenas duas casas decimais, que é um requisito comum ao trabalhar com valores monetários ou medidas precisas.

> Está com alguma dificuldade? Quero te convidar também a conhecer meu **Curso Completo de Java**, nele você dominará todos os conceitos dos exercícios deste ebook.

Capítulo 3: Operadores

Neste capítulo, você aprofundará seu conhecimento sobre operadores em Java, uma das ferramentas mais fundamentais na programação.

Operadores são símbolos especiais que nos permitem realizar operações em variáveis e valores, como somar números, comparar valores, ou mesmo tomar decisões baseadas em certas condições.

O que você vai aprender?

Você será introduzido aos diferentes tipos de operadores que Java oferece, incluindo:

- **Operadores Aritméticos:** usados para realizar operações matemáticas básicas como adição, subtração, multiplicação, divisão e módulo;
- **Operadores de Comparação e Lógicos:** que permitem comparar valores e construir expressões lógicas para controlar o fluxo do seu programa;
- **Operadores de Atribuição e Incremento/Decremento:** que facilitam a manipulação de valores armazenados em variáveis, permitindo atualizar esses valores de maneira eficiente;

Desafios que você enfrentará:

Neste capítulo, você enfrentará desafios que exigem:

- **Cálculos matemáticos:** aplicando operadores aritméticos para resolver problemas práticos, como calcular áreas, médias e realizar conversões;
- **Tomada de decisões:** utilizando operadores de comparação e lógicos para desenvolver programas que tomam decisões com base em condições específicas;

- **Manipulação de variáveis:** aprendendo a utilizar operadores de atribuição e incrementos para modificar e atualizar valores de variáveis de maneira controlada e eficiente;

Esses exercícios o ajudarão a entender como utilizar operadores para manipular dados e controlar o comportamento do seu programa.

Exercícios de Operadores Aritméticos

Exercício 11: Calculadora Simples

Crie um programa que leia dois números inteiros do usuário e exiba a soma, subtração, multiplicação, divisão e o módulo desses números no console.

Código de solução:

```
import java.util.Scanner;

public class CalculadoraSimples {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        int soma = num1 + num2;
        int subtracao = num1 - num2;
        int multiplicacao = num1 * num2;
        int divisao = num1 / num2;
        int modulo = num1 % num2;

        System.out.println("Soma: " + soma);
        System.out.println("Subtração: " + subtracao);
        System.out.println("Multiplicação: " + multiplicacao);
        System.out.println("Divisão: " + divisao);
        System.out.println("Módulo: " + modulo);
    }
}
```

Explicação: Este exercício envolve a leitura de dois números inteiros do usuário e a realização de operações aritméticas básicas (soma, subtração, multiplicação, divisão e módulo) entre eles. Os resultados são então exibidos no console.

Exercício 12: Média Aritmética

Escreva um programa que leia três números inteiros do usuário e calcule a média aritmética deles. Exiba o resultado no console.

Dica: Sempre que os exercícios pedirem para 'ler' algo, você deve utilizar a classe Scanner.

Código de solução:

```
import java.util.Scanner;

public class MediaAritmetica {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        System.out.print("Digite o terceiro número inteiro: ");
        int num3 = scanner.nextInt();

        double media = (num1 + num2 + num3) / 3.0;

        System.out.println("A média aritmética é: " + media);
    }
}
```

Explicação: Neste exercício, o programa lê três números inteiros do usuário, calcula a média aritmética desses números e exibe o resultado no console. A divisão por 3.0 garante que a média seja calculada como um valor double, preservando casas decimais.

Exercício 13: Área de um Retângulo

Desenvolva um programa que leia a largura e a altura de um retângulo e calcule a área. Exiba o resultado no console.

Dica: area = largura x altura.

Código de solução:

```java
import java.util.Scanner;

public class AreaRetangulo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a largura do retângulo: ");
        double largura = scanner.nextDouble();

        System.out.print("Digite a altura do retângulo: ");
        double altura = scanner.nextDouble();

        double area = largura * altura;

        System.out.println("A área do retângulo é: " + area);
    }
}
```

Explicação: Neste exercício, o programa lê a largura e a altura de um retângulo, calcula a área multiplicando esses valores, e exibe o resultado no console.

Exercício 14: Conversão de Temperatura

Crie um programa que converta uma temperatura em graus Celsius para Fahrenheit. A fórmula de conversão é: F = (C * 9/5) + 32. Exiba o resultado no console.

Código de solução:

```java
import java.util.Scanner;

public class ConversaoTemperatura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a temperatura em graus Celsius: ");
        double celsius = scanner.nextDouble();

        double fahrenheit = (celsius * 9/5) + 32;

        System.out.println("A temperatura em Fahrenheit é: " +
```

```
fahrenheit);
    }
}
```

Explicação: Este exercício requer a conversão de uma temperatura em graus Celsius para Fahrenheit usando a fórmula fornecida. O resultado é então exibido no console.

Exercício 15: Potenciação

Escreva um programa que leia dois números inteiros do usuário e exiba o resultado da potenciação do primeiro número elevado ao segundo número (use o método Math.pow).

Dica: pow recebe dois argumentos, o primeiro a base e o segundo o expoente.

Código de solução:

```
import java.util.Scanner;

public class Potenciacao {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o número base: ");
        int base = scanner.nextInt();

        System.out.print("Digite o expoente: ");
        int expoente = scanner.nextInt();

        double resultado = Math.pow(base, expoente);

        System.out.println("O resultado de " + base + " elevado a " +
expoente + " é: " + resultado);
    }
}
```

Explicação: Neste exercício, o programa utiliza o método Math.pow para calcular a potenciação, ou seja, o primeiro número elevado ao segundo número. O resultado é exibido no console.

Exercícios Operadores de Comparação e Lógicos

Exercício 16: Comparação Simples

Crie um programa que leia dois números inteiros e exiba se o primeiro é maior, menor ou igual ao segundo.

Código de solução:

```
import java.util.Scanner;

public class ComparacaoSimples {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        if (num1 > num2) {
            System.out.println("O primeiro número é maior que o
segundo.");
        } else if (num1 < num2) {
            System.out.println("O primeiro número é menor que o
segundo.");
        } else {
            System.out.println("Os dois números são iguais.");
        }
    }
}
```

Explicação: Este exercício compara dois números inteiros fornecidos pelo usuário e exibe uma mensagem indicando se o primeiro número é maior, menor ou igual ao segundo.

Exercício 17: Verificação de Paridade

Escreva um programa que leia um número inteiro e exiba se ele é par ou ímpar.

Dica: Você pode utilizar a divisão de resto, com o operador %.

Código de solução:

```java
import java.util.Scanner;

public class VerificacaoParidade {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int num = scanner.nextInt();

        if (num % 2 == 0) {
            System.out.println("O número é par.");
        } else {
            System.out.println("O número é ímpar.");
        }
    }
}
```

Explicação: Este programa verifica se um número inteiro fornecido pelo usuário é par ou ímpar usando o operador de módulo (%).

Exercício 18: Maior de Três Números

Desenvolva um programa que leia três números inteiros e exiba o maior deles.

Código de solução:

```java
import java.util.Scanner;

public class MaiorDeTres {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        System.out.print("Digite o terceiro número inteiro: ");
        int num3 = scanner.nextInt();

        int maior = num1;

        if (num2 > maior) {
```

```
            maior = num2;
        }

        if (num3 > maior) {
            maior = num3;
        }

        System.out.println("O maior número é: " + maior);
    }
}
```

Explicação: Este exercício determina qual dos três números inteiros fornecidos pelo usuário é o maior e exibe o resultado.

Exercício 19: Elegibilidade para Votação

Crie um programa que leia a idade de uma pessoa e verifique se ela é elegível para votar (idade igual ou superior a 18 anos).

Código de solução:

```
import java.util.Scanner;

public class ElegibilidadeVotacao {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a sua idade: ");
        int idade = scanner.nextInt();

        if (idade >= 18) {
            System.out.println("Você é elegível para votar.");
        } else {
            System.out.println("Você não é elegível para votar.");
        }
    }
}
```

Explicação: Este programa verifica se a idade fornecida pelo usuário é igual ou superior a 18 anos, determinando a elegibilidade para votação.

Exercício 20: Verificação de Intervalo

Escreva um programa que leia um número inteiro e verifique se ele está entre 10 e 20 (inclusive). Exiba uma mensagem informando se o número está dentro ou fora do intervalo.

Código de solução:

```
import java.util.Scanner;

public class VerificacaoIntervalo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int num = scanner.nextInt();

        if (num >= 10 && num <= 20) {
            System.out.println("O número está dentro do intervalo.");
        } else {
            System.out.println("O número está fora do intervalo.");
        }
    }
}
```

Explicação: Este exercício verifica se um número inteiro fornecido pelo usuário está dentro do intervalo de 10 a 20 (inclusive) e exibe a mensagem correspondente.

Exercício 21: Comparação de Strings

Desenvolva um programa que leia duas strings do usuário e verifique se elas são iguais. Exiba uma mensagem informando o resultado da comparação.

Código de solução:

```
import java.util.Scanner;

public class ComparacaoStrings {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a primeira string: ");
        String string1 = scanner.nextLine();
```

```
        System.out.print("Digite a segunda string: ");
        String string2 = scanner.nextLine();

        if (string1.equals(string2)) {
            System.out.println("As strings são iguais.");
        } else {
            System.out.println("As strings são diferentes.");
        }
    }
}
```

Explicação: Este exercício compara duas strings fornecidas pelo usuário e determina se elas são iguais, utilizando o método equals.

Exercício 22: Operadores Lógicos AND e OR

Crie um programa que leia três números inteiros e verifique se pelo menos um deles é maior que 10 (usando o operador ||). Em seguida, verifique se todos são maiores que 10 (usando o operador &&).

Código de solução:

```
import java.util.Scanner;

public class OperadoresLogicos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        System.out.print("Digite o terceiro número inteiro: ");
        int num3 = scanner.nextInt();

        if (num1 > 10 || num2 > 10 || num3 > 10) {
            System.out.println("Pelo menos um dos números é maior que
10.");
        } else {
            System.out.println("Nenhum dos números é maior que 10.");
        }
```

```
        if (num1 > 10 && num2 > 10 && num3 > 10) {
            System.out.println("Todos os números são maiores que 10.");
        } else {
            System.out.println("Nem todos os números são maiores que
10.");
        }
    }
}
```

Explicação: Este programa utiliza operadores lógicos para verificar se pelo menos um dos números é maior que 10 e se todos os números são maiores que 10.

Exercício 23: Verificação de Maioria

Escreva um programa que leia a idade de três pessoas e verifique se pelo menos duas delas são maiores de idade (18 anos ou mais).

Código de solução:

```
import java.util.Scanner;

public class VerificacaoMaioria {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a idade da primeira pessoa: ");
        int idade1 = scanner.nextInt();

        System.out.print("Digite a idade da segunda pessoa: ");
        int idade2 = scanner.nextInt();

        System.out.print("Digite a idade da terceira pessoa: ");
        int idade3 = scanner.nextInt();

        int maioridade = 0;

        if (idade1 >= 18) maioridade++;
        if (idade2 >= 18) maioridade++;
        if (idade3 >= 18) maioridade++;

        if (maioridade >= 2) {
            System.out.println("Pelo menos duas pessoas são maiores de
idade.");
        } else {
```

```
            System.out.println("Menos de duas pessoas são maiores de
idade.");
        }
    }
}
```

Explicação: Este exercício conta quantas pessoas, entre três, são maiores de idade e verifica se pelo menos duas delas têm 18 anos ou mais.

Exercício 24: Ano Bissexto

Crie um programa que leia um ano e verifique se ele é bissexto. Um ano é bissexto se for divisível por 4, mas não por 100, exceto se for divisível por 400.

Código de solução:

```
import java.util.Scanner;

public class VerificacaoAnoBissexto {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um ano: ");
        int ano = scanner.nextInt();

        if ((ano % 4 == 0 && ano % 100 != 0) || (ano % 400 == 0)) {
            System.out.println("O ano é bissexto.");
        } else {
            System.out.println("O ano não é bissexto.");
        }
    }
}
```

Explicação: Este programa verifica se um ano fornecido pelo usuário é bissexto, aplicando as regras de divisibilidade específicas.

Exercício 25: Verificação de Números Positivos

Escreva um programa que leia três números inteiros e verifique se pelo menos dois deles são positivos.

Código de solução:

```java
import java.util.Scanner;

public class VerificacaoPositivos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        System.out.print("Digite o terceiro número inteiro: ");
        int num3 = scanner.nextInt();

        int positivos = 0;

        if (num1 > 0) positivos++;
        if (num2 > 0) positivos++;
        if (num3 > 0) positivos++;

        if (positivos >= 2) {
            System.out.println("Pelo menos dois dos números são
positivos.");
        } else {
            System.out.println("Menos de dois dos números são
positivos.");
        }
    }
}
```

Explicação: Este programa verifica quantos dos três números fornecidos pelo usuário são positivos e exibe uma mensagem indicando se pelo menos dois deles são maiores que zero.

Operadores de Atribuição e Incremento/Decremento

Exercício 26: Atribuição Simples

Enunciado: Crie um programa que declare uma variável inteira, atribua um valor a ela e, em seguida, modifique o valor utilizando os operadores de atribuição (+=, -=, *=, /=, %=). Exiba o resultado após cada operação.

Código de solução:

```
public class AtribuicaoSimples {
    public static void main(String[] args) {
        int valor = 10;

        valor += 5;
        System.out.println("Após valor += 5: " + valor);

        valor -= 3;
        System.out.println("Após valor -= 3: " + valor);

        valor *= 2;
        System.out.println("Após valor *= 2: " + valor);

        valor /= 4;
        System.out.println("Após valor /= 4: " + valor);

        valor %= 3;
        System.out.println("Após valor %= 3: " + valor);
    }
}
```

Explicação: Neste exercício, uma variável inteira é declarada e inicializada. Em seguida, seu valor é modificado usando os operadores de atribuição (+=, -=, *=, /=, %=). Após cada operação, o novo valor da variável é exibido no console.

Exercício 27: Incremento e Decremento

Enunciado: Escreva um programa que declare uma variável inteira, aplique o operador de incremento (++) e decremento (--), e exiba o valor da variável antes e depois de cada operação.

Código de solução:

```
public class IncrementoDecremento {
    public static void main(String[] args) {
        int numero = 15;

        System.out.println("Valor inicial: " + numero);
```

```
        numero++;
        System.out.println("Após incremento (++numero): " + numero);

        numero--;
        System.out.println("Após decremento (--numero): " + numero);
    }
}
```

Explicação: Neste exercício, uma variável inteira é declarada e incrementada com o operador ++ e decrementada com o operador --. O valor da variável é exibido antes e após cada operação para mostrar as mudanças.

Exercício 28: Soma Acumulada

Enunciado: Desenvolva um programa que leia cinco números inteiros do usuário, um por vez, e acumule a soma deles usando o operador de atribuição +=. Exiba o total acumulado ao final.

Dica: Utilize o Scanner para pedir os dados, e você pode utilizar um loop for para repetir a solicitação de dados cinco vezes.

Código de solução:

```
import java.util.Scanner;

public class SomaAcumulada {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int somaAcumulada = 0;

        for (int i = 1; i <= 5; i++) {
            System.out.print("Digite o " + i + "º número inteiro: ");
            int valorDigitado = scanner.nextInt();
            somaAcumulada += valorDigitado;
        }

        System.out.println("Soma acumulada: " + somaAcumulada);
    }
}
```

Explicação: Este programa lê cinco números inteiros do usuário, um por vez, e acumula a soma desses valores usando o operador +=. O valor acumulado é exibido ao final.

Exercício 29: Pré-incremento e Pós-incremento

Enunciado: Crie um programa que demonstre a diferença entre o pré-incremento (++variavel) e o pós-incremento (variavel++). Utilize exemplos práticos e exiba os resultados no console.

Código de solução:

```
public class PrePosIncremento {
    public static void main(String[] args) {
        int valorPrePos = 10;

        System.out.println("Valor inicial: " + valorPrePos);

        int preIncremento = ++valorPrePos;
        System.out.println("Após pré-incremento (++valor): " +
preIncremento);

        int posIncremento = valorPrePos++;
        System.out.println("Após pós-incremento (valor++): " +
posIncremento);

        System.out.println("Valor final após pós-incremento: " +
valorPrePos);
    }
}
```

Explicação: Este exercício demonstra a diferença entre o pré-incremento, onde a variável é incrementada antes de ser usada, e o pós-incremento, onde a variável é usada antes de ser incrementada. Os resultados de ambas as operações são exibidos para ilustrar a diferença.

Exercício 30: Operadores Compostos

Enunciado: Escreva um programa que leia dois números inteiros do usuário e aplique operadores compostos (e.g., +=, -=, *=, /=, %=) para modificar o valor da primeira variável em relação à segunda. Exiba o resultado após cada operação.

Dica: Use o Scanner para receber os números.

Código de solução:

```
import java.util.Scanner;
```

```java
public class OperadoresCompostos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int valor1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int valor2 = scanner.nextInt();

        valor1 += valor2;
        System.out.println("Após valor1 += valor2: " + valor1);

        valor1 -= valor2;
        System.out.println("Após valor1 -= valor2: " + valor1);

        valor1 *= valor2;
        System.out.println("Após valor1 *= valor2: " + valor1);

        valor1 /= valor2;
        System.out.println("Após valor1 /= valor2: " + valor1);

        valor1 %= valor2;
        System.out.println("Após valor1 %= valor2: " + valor1);
    }
}
```

Explicação: Neste exercício, dois números inteiros são lidos do usuário, e operadores compostos (+=, -=, *=, /=, %=) são aplicados à primeira variável em relação à segunda. Após cada operação, o novo valor da primeira variável é exibido.

Conclusão do Capítulo 3

Neste capítulo, você explorou uma variedade de operadores em Java, que são fundamentais para manipular e controlar dados em seus programas. Os exercícios foram projetados para reforçar o entendimento dos seguintes conceitos:

Operadores Aritméticos: Você aprendeu a realizar operações matemáticas básicas, como adição, subtração, multiplicação, divisão e módulo, além de trabalhar com expressões mais complexas, como potenciação e cálculo de médias.

Operadores de Comparação e Lógicos: Você praticou a comparação de valores e a construção de expressões lógicas que permitem tomar decisões baseadas em condições

específicas. Isso incluiu o uso de operadores de igualdade, maior/menor, e combinações lógicas utilizando AND, OR, e NOT.

Operadores de Atribuição e Incremento/Decremento: Você viu como modificar valores de variáveis de forma eficiente utilizando operadores compostos e incrementos/decrementos, entendendo a diferença entre pré-incremento e pós-incremento.

Ao dominar esses operadores, você desenvolveu habilidades essenciais para controlar o fluxo de seus programas, realizar cálculos e manipular dados de forma eficaz.

Porém, isso é apenas o começo, temos muito o que explorar no universo de Java através de exercícios! Vejo você no próximo capítulo.

Capítulo 4: Controle de Fluxo

Neste capítulo, você irá aprofundar seu conhecimento sobre o controle de fluxo em Java, um dos aspectos mais importantes da programação. O controle de fluxo permite que você tome decisões, repita ações e altere o comportamento do seu programa com base em diferentes condições e cenários.

O que você vai aprender?
Estruturas Condicionais (if, else if, else): Você começará com as estruturas condicionais, que são fundamentais para a tomada de decisões em seus programas. Essas estruturas permitem que seu código execute diferentes blocos de instruções com base em condições específicas.

Switch Case: Em seguida, você aprenderá a usar a estrutura switch case, que é especialmente útil quando você precisa avaliar uma única variável contra várias opções possíveis. Isso torna seu código mais legível e organizado em situações onde múltiplas comparações são necessárias.

Laços de Repetição (for, while, do-while): Finalmente, você explorará os laços de repetição, que permitem que você execute um bloco de código várias vezes, seja um número específico de vezes ou até que uma determinada condição seja atendida. Isso inclui o uso dos laços for, while e do-while, cada um com suas próprias características e aplicações.

Desafios que você enfrentará
Neste capítulo, os desafios incluirão desde a criação de simples tomadas de decisão, como verificar a validade de uma condição, até o uso de laços para repetir ações complexas. Você será desafiado a implementar soluções que requerem um pensamento lógico robusto e a utilização eficiente das estruturas de controle de fluxo.

Com a conclusão deste capítulo, você terá uma compreensão sólida sobre como controlar o comportamento dos seus programas de forma dinâmica e eficiente, o que é essencial para resolver problemas de programação no mundo real. Prepare-se para aplicar essas

ferramentas poderosas em uma variedade de situações que encontrará no desenvolvimento de software.

Estruturas Condicionais (if, else if, else)

Exercício 31: Verificação de Positivo ou Negativo

Enunciado: Escreva um programa que leia um número inteiro e verifique se ele é positivo, negativo ou zero. Exiba uma mensagem apropriada para cada caso.

Código de solução:

import java.util.Scanner;

```
public class VerificacaoPositivoNegativo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int numero = scanner.nextInt();

        if (numero > 0) {
            System.out.println("O número é positivo.");
        } else if (numero < 0) {
            System.out.println("O número é negativo.");
        } else {
            System.out.println("O número é zero.");
        }
    }
}
```

Explicação: O programa lê um número inteiro e, usando estruturas condicionais, verifica se o número é positivo, negativo ou zero. A mensagem correspondente é exibida para cada situação.

Exercício 32: Par ou Ímpar

Enunciado: Crie um programa que leia um número inteiro e exiba se o número é par ou ímpar.

Código de solução:

```java
import java.util.Scanner;

public class ParOuImpar {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int numero = scanner.nextInt();

        if (numero % 2 == 0) {
            System.out.println("O número é par.");
        } else {
            System.out.println("O número é ímpar.");
        }
    }
}
```

Explicação: Este programa usa o operador módulo (%) para determinar se o número fornecido é divisível por 2 (par) ou não (ímpar), exibindo a mensagem correspondente.

Exercício 33: Cálculo de Desconto

Enunciado: Desenvolva um programa que leia o valor de uma compra e aplique um desconto de 10% se o valor for superior a R$100,00. Exiba o valor final com ou sem desconto.

Código de solução:

```java
import java.util.Scanner;

public class CalculoDesconto {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o valor da compra: ");
        double valorCompra = scanner.nextDouble();

        if (valorCompra > 100.00) {
            valorCompra *= 0.90;  // Aplicando desconto de 10%
            System.out.println("Desconto aplicado! Valor final: R$ " +
valorCompra);
        } else {
            System.out.println("Sem desconto. Valor final: R$ " +
valorCompra);
```

```
        }
    }
}
```

Explicação: Este programa verifica se o valor da compra excede R$100,00 e aplica um desconto de 10% quando necessário. Caso contrário, o valor original é mantido e exibido.

Exercício 34: Verificação de Maioridade

Enunciado: Escreva um programa que leia a idade de uma pessoa e exiba uma mensagem informando se ela é menor de idade (menor que 18 anos), maior de idade (18 anos ou mais) ou idosa (60 anos ou mais).

Código de solução:

```
import java.util.Scanner;

public class VerificacaoMaioridade {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a sua idade: ");
        int idade = scanner.nextInt();

        if (idade >= 60) {
            System.out.println("Você é idoso.");
        } else if (idade >= 18) {
            System.out.println("Você é maior de idade.");
        } else {
            System.out.println("Você é menor de idade.");
        }
    }
}
```

Explicação: O programa lê a idade do usuário e usa estruturas condicionais para verificar se ele é menor de idade, maior de idade ou idoso, exibindo a mensagem apropriada.

Exercício 35: Classificação de Notas

Enunciado: Crie um programa que leia uma nota de 0 a 100 e exiba uma mensagem de aprovação se a nota for maior ou igual a 60. Caso contrário, exiba uma mensagem de reprovação.

Código de solução:

```java
import java.util.Scanner;

public class ClassificacaoNotas {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a sua nota (0 a 100): ");
        int nota = scanner.nextInt();

        if (nota >= 60) {
            System.out.println("Você está aprovado.");
        } else {
            System.out.println("Você está reprovado.");
        }
    }
}
```

Explicação: O programa lê a nota do usuário e verifica se é maior ou igual a 60 para determinar se ele está aprovado ou reprovado, exibindo a mensagem correspondente.

Exercício 36: Comparação de Três Números

Enunciado: Desenvolva um programa que leia três números inteiros e exiba o maior deles. Caso dois ou mais números sejam iguais, exiba uma mensagem indicando que há números iguais.

Código de solução:

```java
import java.util.Scanner;

public class ComparacaoTresNumeros {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número inteiro: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número inteiro: ");
        int num2 = scanner.nextInt();

        System.out.print("Digite o terceiro número inteiro: ");
```

```
        int num3 = scanner.nextInt();

        if (num1 == num2 && num2 == num3) {
            System.out.println("Todos os números são iguais.");
        } else if (num1 >= num2 && num1 >= num3) {
            System.out.println("O maior número é: " + num1);
        } else if (num2 >= num1 && num2 >= num3) {
            System.out.println("O maior número é: " + num2);
        } else {
            System.out.println("O maior número é: " + num3);
        }
    }
}
```

Explicação: Este programa lê três números e determina qual é o maior deles. Se todos os números forem iguais, uma mensagem especial é exibida.

Exercício 37: Avaliação de Temperatura

Enunciado: Escreva um programa que leia a temperatura atual em graus Celsius e exiba uma mensagem dizendo se o clima está "Frio" (abaixo de 15ºC), "Agradável" (entre 15ºC e 30ºC) ou "Quente" (acima de 30ºC).

Código de solução:

```
import java.util.Scanner;

public class AvaliacaoTemperatura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a temperatura em graus Celsius: ");
        double temperatura = scanner.nextDouble();

        if (temperatura < 15) {
            System.out.println("Clima está Frio.");
        } else if (temperatura <= 30) {
            System.out.println("Clima está Agradável.");
        } else {
            System.out.println("Clima está Quente.");
        }
    }
}
```

Explicação: O programa classifica a temperatura inserida pelo usuário em três categorias (Frio, Agradável e Quente) com base em valores pré-definidos.

Exercício 38: Verificação de Nota Escolar

Enunciado: Desenvolva um programa que leia uma nota escolar (de 0 a 10) e classifique-a como "Insuficiente" (menor que 5), "Suficiente" (entre 5 e 7) ou "Bom" (maior que 7).

Código de solução:

```
import java.util.Scanner;

public class VerificacaoNotaEscolar {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite sua nota escolar (0 a 10): ");
        double notaEscolar = scanner.nextDouble();

        if (notaEscolar < 5) {
            System.out.println("Classificação: Insuficiente.");
        } else if (notaEscolar <= 7) {
            System.out.println("Classificação: Suficiente.");
        } else {
            System.out.println("Classificação: Bom.");
        }
    }
}
```

Explicação: Este programa lê a nota escolar e classifica o desempenho do aluno em três categorias: Insuficiente, Suficiente ou Bom.

Exercício 39: Comparação de Idades

Enunciado: Escreva um programa que leia as idades de duas pessoas e exiba quem é mais velho. Caso as idades sejam iguais, exiba uma mensagem informando que as duas pessoas têm a mesma idade.

Código de solução:

```
import java.util.Scanner;
```

```java
public class ComparacaoIdades {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a idade da primeira pessoa: ");
        int idade1 = scanner.nextInt();

        System.out.print("Digite a idade da segunda pessoa: ");
        int idade2 = scanner.nextInt();

        if (idade1 > idade2) {
            System.out.println("A primeira pessoa é mais velha.");
        } else if (idade1 < idade2) {
            System.out.println("A segunda pessoa é mais velha.");
        } else {
            System.out.println("Ambas têm a mesma idade.");
        }
    }
}
```

Explicação: O programa compara as idades de duas pessoas e exibe qual é mais velha ou se ambas têm a mesma idade.

Exercício 40: Avaliação de Velocidade

Enunciado: Crie um programa que leia a velocidade de um veículo e exiba uma mensagem dizendo se o veículo está dentro do limite de velocidade (até 60 km/h), acima do limite (entre 61 km/h e 80 km/h) ou muito acima do limite (acima de 80 km/h).

Código de solução:

```java
import java.util.Scanner;

public class AvaliacaoVelocidade {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a velocidade do veículo (km/h): ");
        int velocidade = scanner.nextInt();

        if (velocidade <= 60) {
            System.out.println("Veículo está dentro do limite de
velocidade.");
        } else if (velocidade <= 80) {
```

```
            System.out.println("Veículo está acima do limite de
velocidade.");
        } else {
            System.out.println("Veículo está muito acima do limite de
velocidade.");
        }
    }
}
```

Explicação: O programa lê a velocidade do veículo e exibe se ele está dentro do limite, acima do limite ou muito acima do limite, com base em intervalos predefinidos.

Switch Case

Exercício 41: Dia da Semana

Enunciado: Escreva um programa que leia um número inteiro de 1 a 7 e exiba o nome do dia da semana correspondente (1 para domingo, 2 para segunda-feira, etc.).

Código de solução:

```
import java.util.Scanner;

public class DiaDaSemana {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número de 1 a 7 para o dia da
semana: ");
        int diaSemana = scanner.nextInt();

        switch (diaSemana) {
            case 1:
                System.out.println("Domingo");
                break;
            case 2:
                System.out.println("Segunda-feira");
                break;
            case 3:
                System.out.println("Terça-feira");
                break;
            case 4:
```

```
                System.out.println("Quarta-feira");
                break;
            case 5:
                System.out.println("Quinta-feira");
                break;
            case 6:
                System.out.println("Sexta-feira");
                break;
            case 7:
                System.out.println("Sábado");
                break;
            default:
                System.out.println("Número inválido! Digite um número de
1 a 7.");
        }
    }
}
```

Explicação: O programa lê um número de 1 a 7 e exibe o nome correspondente ao dia da semana usando a estrutura switch case. Se o número estiver fora do intervalo de 1 a 7, uma mensagem de erro é exibida.

Exercício 42: Classificação de Nota

Enunciado: Crie um programa que leia uma nota de 0 a 10 e classifique a nota de acordo com as seguintes categorias:

- 10: Excelente
- 8 e 9: Muito bom
- 6 e 7: Bom
- 5: Regular
- 0 a 4: Insuficiente

Código de solução:

```
import java.util.Scanner;

public class ClassificacaoDeNota {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma nota de 0 a 10: ");
```

```
        int nota = scanner.nextInt();

        switch (nota) {
            case 10:
                System.out.println("Excelente");
                break;
            case 9:
            case 8:
                System.out.println("Muito bom");
                break;
            case 7:
            case 6:
                System.out.println("Bom");
                break;
            case 5:
                System.out.println("Regular");
                break;
            case 4:
            case 3:
            case 2:
            case 1:
            case 0:
                System.out.println("Insuficiente");
                break;
            default:
                System.out.println("Nota inválida.");
        }
    }
}
```

Explicação: O programa lê uma nota de 0 a 10 e a classifica de acordo com as categorias estabelecidas, usando a estrutura switch case. Notas fora do intervalo resultam em uma mensagem de erro.

Exercício 43: Operações Matemáticas

Enunciado: Desenvolva um programa que leia dois números e um operador (+, -, *, /) e realize a operação correspondente. Exiba o resultado no console.

Código de solução:

```
import java.util.Scanner;

public class OperacoesMatematicas {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número: ");
        double numero1 = scanner.nextDouble();

        System.out.print("Digite o segundo número: ");
        double numero2 = scanner.nextDouble();

        System.out.print("Digite o operador (+, -, *, /): ");
        char operador = scanner.next().charAt(0);

        switch (operador) {
            case '+':
                System.out.println("Resultado: " + (numero1 + numero2));
                break;
            case '-':
                System.out.println("Resultado: " + (numero1 - numero2));
                break;
            case '*':
                System.out.println("Resultado: " + (numero1 * numero2));
                break;
            case '/':
                if (numero2 != 0) {
                    System.out.println("Resultado: " + (numero1 /
numero2));
                } else {
                    System.out.println("Divisão por zero não é
permitida.");
                }
                break;
            default:
                System.out.println("Operador inválido.");
        }
    }
}
```

Explicação: O programa lê dois números e um operador matemático e usa switch case para realizar a operação correspondente. A divisão por zero é tratada com uma verificação adicional.

Exercício 44: Estação do Ano

Enunciado: Escreva um programa que leia um número de 1 a 4 e exiba o nome da estação do ano correspondente:

- 1: Verão
- 2: Outono
- 3: Inverno
- 4: Primavera

Código de solução:

```
import java.util.Scanner;

public class EstacaoDoAno {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número de 1 a 4 para a estação do
ano: ");
        int estacao = scanner.nextInt();

        switch (estacao) {
            case 1:
                System.out.println("Verão");
                break;
            case 2:
                System.out.println("Outono");
                break;
            case 3:
                System.out.println("Inverno");
                break;
            case 4:
                System.out.println("Primavera");
                break;
            default:
                System.out.println("Número inválido. Digite um número de
1 a 4.");
        }
    }
}
```

Explicação: O programa lê um número de 1 a 4 e exibe a estação correspondente, usando a estrutura switch case. Números fora do intervalo resultam em uma mensagem de erro.

Exercício 45: Classificação de Idade

Enunciado: Crie um programa que leia a idade de uma pessoa e classifique-a em uma das seguintes categorias:

- 0 a 12: Criança
- 13 a 17: Adolescente
- 18 a 59: Adulto
- 60 ou mais: Idoso

Dica: Aqui veremos porque o switch case não funciona muito bem com intervalos de valores, e o if deve ser a estrutura preferida.

Código de solução:

```
import java.util.Scanner;

public class ClassificacaoDeIdade {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a idade: ");
        int idade = scanner.nextInt();

        switch (idade) {
            case 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12:
                System.out.println("Criança");
                break;
            case 13, 14, 15, 16, 17:
                System.out.println("Adolescente");
                break;
            case 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
50, 51, 52, 53, 54, 55, 56, 57, 58, 59:
                System.out.println("Adulto");
                break;
            default:
                System.out.println("Idoso");
        }
    }
}
```

Explicação: O programa lê a idade de uma pessoa e a classifica em uma das quatro categorias (Criança, Adolescente, Adulto, Idoso) usando switch case. Qualquer idade acima de 59 é classificada como "Idoso".

Laços de Repetição

Exercício 46: Contagem Crescente

Enunciado: Escreva um programa que exiba os números de 1 a 10, em ordem crescente, usando um laço for.

Código de solução:

```
public class ContagemCrescente {
    public static void main(String[] args) {
        for (int i = 1; i <= 10; i++) {
            System.out.println(i);
        }
    }
}
```

Explicação: O laço for começa com o valor 1 e vai até 10, incrementando 1 a cada iteração. O valor de i é exibido a cada passo, mostrando os números de 1 a 10 em ordem crescente.

Exercício 47: Contagem Decrescente

Enunciado: Crie um programa que exiba os números de 10 a 1, em ordem decrescente, usando um laço for.

Código de solução:

```
public class ContagemDecrescente {
    public static void main(String[] args) {
        for (int i = 10; i >= 1; i--) {
            System.out.println(i);
        }
    }
}
```

Explicação: Neste exercício, o laço for inicia com o valor 10 e vai até 1, decrementando 1 a cada iteração. O valor de i é exibido em cada passo, mostrando os números de 10 a 1 em ordem decrescente.

Exercício 48: Números Pares

Enunciado: Desenvolva um programa que exiba todos os números pares de 1 a 20.

Código de solução:

```
public class NumerosPares {
    public static void main(String[] args) {
        for (int i = 2; i <= 20; i += 2) {
            System.out.println(i);
        }
    }
}
```

Explicação: O laço for começa em 2 e vai até 20, incrementando de 2 em 2. Dessa forma, apenas os números pares são exibidos no console.

Exercício 49: Soma de Números

Enunciado: Escreva um programa que calcule a soma de todos os números de 1 a 100.

Código de solução:

```
public class SomaDeNumeros {
    public static void main(String[] args) {
        int soma = 0;
        for (int i = 1; i <= 100; i++) {
            soma += i;
        }
        System.out.println("Soma de 1 a 100: " + soma);
    }
}
```

Explicação: O programa utiliza um laço for para percorrer os números de 1 a 100. A cada iteração, o valor de i é somado à variável soma, e o resultado total é exibido ao final.

Exercício 50: Tabuada

Enunciado: Crie um programa que exiba a tabuada de um número fornecido pelo usuário, de 1 a 10.

Código de solução:

```
import java.util.Scanner;

public class Tabuada {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite um número para a tabuada: ");
        int numero = scanner.nextInt();

        for (int i = 1; i <= 10; i++) {
            System.out.println(numero + " x " + i + " = " + (numero *
i));
        }
    }
}
```

Explicação: O programa pede ao usuário que forneça um número e, em seguida, usa um laço for para calcular e exibir a tabuada desse número de 1 a 10.

Exercício 51: Fatorial

Enunciado: Desenvolva um programa que calcule o fatorial de um número inteiro fornecido pelo usuário.

Código de solução:

```
import java.util.Scanner;

public class Fatorial {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite um número para calcular o fatorial: ");
        int num = scanner.nextInt();
        int fatorial = 1;

        for (int i = 1; i <= num; i++) {
            fatorial *= i;
        }

        System.out.println("Fatorial de " + num + " é: " + fatorial);
    }
}
```

Explicação: O programa lê um número fornecido pelo usuário e usa um laço for para multiplicar os números de 1 até o valor dado, calculando o fatorial. O resultado é exibido no final.

Exercício 52: Números Ímpares

Enunciado: Escreva um programa que exiba todos os números ímpares entre 1 e 50.

Código de solução:

```
public class NumerosImpares {
    public static void main(String[] args) {
        for (int i = 1; i <= 50; i += 2) {
            System.out.println(i);
        }
    }
}
```

Explicação: O laço for começa em 1 e vai até 50, incrementando de 2 em 2, garantindo que apenas os números ímpares sejam exibidos no console.

Exercício 53: Média de Números

Enunciado: Crie um programa que leia 5 números inteiros do usuário e calcule a média deles.

Código de solução:

```
import java.util.Scanner;

public class MediaDeNumeros {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int soma = 0;

        for (int i = 1; i <= 5; i++) {
            System.out.print("Digite o número " + i + ": ");
            int numero = scanner.nextInt();
            soma += numero;
        }

        double media = soma / 5.0;
        System.out.println("A média é: " + media);
    }
}
```

Explicação: O programa lê 5 números inteiros fornecidos pelo usuário, soma-os e, em seguida, calcula a média, que é exibida ao final.

Exercício 54: Contagem de Múltiplos de 3

Enunciado: Desenvolva um programa que conte quantos números entre 1 e 100 são múltiplos de 3.

Código de solução:

```
public class ContagemMultiplosDeTres {
    public static void main(String[] args) {
        int contagem = 0;

        for (int i = 1; i <= 100; i++) {
            if (i % 3 == 0) {
                contagem++;
            }
        }

        System.out.println("Quantidade de múltiplos de 3 entre 1 e 100:
" + contagem);
    }
}
```

Explicação: O programa usa um laço for para verificar todos os números de 1 a 100. A cada iteração, ele verifica se o número é múltiplo de 3 e aumenta a contagem, exibindo o total ao final.

Exercício 55: Sequência de Fibonacci

Enunciado: Escreva um programa que exiba os primeiros 10 termos da sequência de Fibonacci.

Código de solução:

```
public class Fibonacci {
    public static void main(String[] args) {
        int termo1 = 0, termo2 = 1;

        System.out.print(termo1 + " " + termo2 + " ");

        for (int i = 3; i <= 10; i++) {
            int proximoTermo = termo1 + termo2;
```

```
            System.out.print(proximoTermo + " ");
            termo1 = termo2;
            termo2 = proximoTermo;
        }
    }
}
```

Explicação: Este programa exibe os primeiros 10 termos da sequência de Fibonacci. Ele começa com 0 e 1, e em cada iteração, calcula o próximo termo somando os dois anteriores.

Exercício 56: Produto de Números

Enunciado: Crie um programa que calcule o produto dos números inteiros de 1 a 10.

Código de solução:

```
public class ProdutoDeNumeros {
    public static void main(String[] args) {
        int produto = 1;

        for (int i = 1; i <= 10; i++) {
            produto *= i;
        }

        System.out.println("O produto dos números de 1 a 10 é: " +
produto);
    }
}
```

Explicação: O laço for percorre os números de 1 a 10, multiplicando cada número pelo valor atual de produto. O resultado final é exibido no console.

Exercício 57: Soma de Pares e Ímpares Separadamente

Enunciado: Escreva um programa que some todos os números pares de 1 a 100 e, separadamente, todos os números ímpares de 1 a 100.

Código de solução:

```
public class SomaParesImpares {
    public static void main(String[] args) {
```

```
        int somaPares = 0, somaImpares = 0;

        for (int i = 1; i <= 100; i++) {
            if (i % 2 == 0) {
                somaPares += i;
            } else {
                somaImpares += i;
            }
        }

        System.out.println("Soma dos números pares: " + somaPares);
        System.out.println("Soma dos números ímpares: " + somaImpares);
    }
}
```

Explicação: Neste programa, os números de 1 a 100 são verificados. Se o número for par, ele é adicionado à somaPares. Caso contrário, ele é adicionado à somaImpares. As somas finais são exibidas no console.

Exercício 58: Validação de Entrada (while)

Enunciado: Crie um programa que leia um número inteiro entre 1 e 10. Caso o valor seja inválido, continue pedindo a entrada até que um número válido seja fornecido.

Código de solução:

```
import java.util.Scanner;

public class ValidacaoEntradaWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite um número entre 1 e 10: ");
        int numero = scanner.nextInt();

        while (numero < 1 || numero > 10) {
            System.out.println("Número inválido. Tente novamente.");
            System.out.print("Digite um número entre 1 e 10: ");
            numero = scanner.nextInt();
        }

        System.out.println("Número válido: " + numero);
    }
}
```

Explicação: O laço while garante que, enquanto o número fornecido pelo usuário estiver fora do intervalo de 1 a 10, o programa continuará solicitando um novo número. O laço só termina quando um número válido é fornecido.

Exercício 59: Soma de Números (while)

Enunciado: Desenvolva um programa que leia números inteiros do usuário e exiba a soma acumulada. O programa deve terminar quando o usuário digitar o número zero.

Código de solução:

```
import java.util.Scanner;

public class SomaNumerosWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int somaAcumulada = 0;
        int valorDigitado;

        System.out.println("Digite números para somar. Digite 0 para
parar.");
        System.out.print("Digite um número: ");
        valorDigitado = scanner.nextInt();

        while (valorDigitado != 0) {
            somaAcumulada += valorDigitado;
            System.out.print("Digite outro número: ");
            valorDigitado = scanner.nextInt();
        }

        System.out.println("Soma total: " + somaAcumulada);
    }
}
```

Explicação: O laço while soma os números fornecidos pelo usuário até que o valor zero seja digitado, momento em que o laço é encerrado e o resultado da soma acumulada é exibido.

Exercício 60: Contagem de Números Positivos (while)

Enunciado: Escreva um programa que leia números inteiros e exiba quantos desses números são positivos. O programa deve parar quando o usuário digitar um número negativo.

Código de solução:

```
import java.util.Scanner;

public class ContagemPositivosWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int contagemPositivos = 0;
        int numero;

        System.out.print("Digite um número (negativo para parar): ");
        numero = scanner.nextInt();

        while (numero >= 0) {
            if (numero > 0) {
                contagemPositivos++;
            }
            System.out.print("Digite outro número (negativo para parar):
");
            numero = scanner.nextInt();
        }

        System.out.println("Quantidade de números positivos: " +
contagemPositivos);
    }
}
```

Explicação: Neste programa, o laço while continua a ler números até que um número negativo seja inserido. A cada número positivo inserido, a contagem de positivos é incrementada.

Exercício 61: Raiz Quadrada Aproximada (while)

Enunciado: Crie um programa que leia um número inteiro positivo e encontre a raiz quadrada aproximada desse número. Continue a tentativa até encontrar a aproximação correta.

Código de solução:

```
import java.util.Scanner;

public class RaizQuadradaAproximadaWhile {
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite um número inteiro positivo: ");
        int numero = scanner.nextInt();

        int raizAprox = 0;
        while (raizAprox * raizAprox < numero) {
            raizAprox++;
        }

        if (raizAprox * raizAprox == numero) {
            System.out.println("Raiz quadrada exata de " + numero + " é:
" + raizAprox);
        } else {
            System.out.println("Raiz quadrada aproximada de " + numero +
" é: " + raizAprox);
        }
    }
}
```

Explicação: O programa usa um laço while para incrementar a variável raizAprox até que seu quadrado seja igual ou superior ao número fornecido pelo usuário. Ele determina a raiz quadrada exata ou uma aproximação.

Exercício 62: Multiplicação por Acumulação (while)

Enunciado: Desenvolva um programa que leia um número e multiplique esse número por 2 repetidamente até o valor exceder 1000.

Código de solução:

```
import java.util.Scanner;

public class MultiplicacaoAcumulacaoWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite um número: ");
        int numero = scanner.nextInt();

        while (numero <= 1000) {
            numero *= 2;
            System.out.println("Valor após multiplicação: " + numero);
        }

        System.out.println("Valor final após multiplicação acumulada: "
```

```
+ numero);
    }
}
```

Explicação: O laço while multiplica o número fornecido por 2 a cada iteração até que ele ultrapasse o valor 1000. O valor atualizado é exibido a cada passo.

Exercício 63: Senha Correta (do-while)

Enunciado: Escreva um programa que peça ao usuário para digitar uma senha. Continue pedindo a senha até que a senha correta seja digitada.

Código de solução:

```
import java.util.Scanner;

public class SenhaCorretaDoWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String senhaCorreta = "1234";
        String senhaDigitada;

        do {
            System.out.print("Digite a senha: ");
            senhaDigitada = scanner.nextLine();
            if (!senhaDigitada.equals(senhaCorreta)) {
                System.out.println("Senha incorreta. Tente novamente.");
            }
        } while (!senhaDigitada.equals(senhaCorreta));

        System.out.println("Senha correta! Acesso concedido.");
    }
}
```

Explicação: Neste caso, o laço do-while é usado para continuar pedindo a senha até que o usuário insira a senha correta. A senha é verificada a cada iteração.

Exercício 64: Menu de Opções (do-while)

Enunciado: Crie um programa que exiba um menu de opções e permita ao usuário escolher uma ação (como somar dois números, subtrair, etc.). O menu deve continuar sendo exibido até o usuário escolher a opção de sair.

Código de solução:

```
import java.util.Scanner;

public class MenuOpcoesDoWhile {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int opcao;

        do {
            System.out.println("Menu de Opções:");
            System.out.println("1. Somar dois números");
            System.out.println("2. Subtrair dois números");
            System.out.println("3. Multiplicar dois números");
            System.out.println("4. Dividir dois números");
            System.out.println("5. Sair");
            System.out.print("Escolha uma opção: ");
            opcao = scanner.nextInt();

            if (opcao >= 1 && opcao <= 4) {
                System.out.print("Digite o primeiro número: ");
                double num1 = scanner.nextDouble();
                System.out.print("Digite o segundo número: ");
                double num2 = scanner.nextDouble();

                switch (opcao) {
                    case 1:
                        System.out.println("Resultado: " + (num1 +
num2));
                        break;
                    case 2:
                        System.out.println("Resultado: " + (num1 -
num2));
                        break;
                    case 3:
                        System.out.println("Resultado: " + (num1 *
num2));
                        break;
                    case 4:
                        if (num2 != 0) {
                            System.out.println("Resultado: " + (num1 /
num2));
                        } else {
                            System.out.println("Erro: Divisão por zero
não é permitida.");
```

```
                        }
                        break;
                }
            } else if (opcao != 5) {
                System.out.println("Opção inválida. Tente novamente.");
            }

            System.out.println();
        } while (opcao != 5);

        System.out.println("Programa encerrado.");
    }
}
```

Explicação: O programa apresenta um menu com operações matemáticas. Usando do-while, o menu é exibido repetidamente até que o usuário selecione a opção de sair.

Conclusão do Capítulo 4: Controle de Fluxo

Neste capítulo, você explorou as principais ferramentas de controle de fluxo em Java, que são essenciais para criar programas dinâmicos e adaptáveis. As Estruturas Condicionais (if, else if, else) permitiram que você tomasse decisões com base em condições específicas, enquanto o Switch Case trouxe uma forma mais clara e direta de lidar com múltiplas opções possíveis.

Nos Laços de Repetição (for, while, do-while), você praticou a execução repetida de blocos de código, seja para iterar sobre números, validar entradas ou realizar cálculos complexos. Cada um desses laços tem seu próprio uso ideal e você agora entende quando aplicar cada um deles.

Capítulo 5: Arrays e Coleções

Neste capítulo, você irá explorar o mundo dos **Arrays e Coleções** em Java, ferramentas essenciais para armazenar e manipular grandes volumes de dados de forma eficiente. Iniciaremos com os **Arrays Unidimensionais**, que permitem armazenar e acessar múltiplos valores de um mesmo tipo de dado de forma estruturada. A seguir, veremos os **Arrays Multidimensionais (2D)**, utilizados para organizar dados em matrizes e tabelas, aumentando as possibilidades de manipulação.

Além disso, aprenderemos a realizar operações essenciais de **Manipulação de Arrays**, como busca de elementos, ordenação de valores e outras técnicas que são amplamente usadas em desenvolvimento de software para organizar e processar dados de forma

otimizada. Ao final deste capítulo, você será capaz de trabalhar com arrays de diferentes dimensões e manipular esses dados de forma eficaz em diversos cenários práticos.

Arrays Unidimensionais

Exercício 65: Criação e Inicialização de um Array

Enunciado: Crie um programa que declare um array de 5 números inteiros. Atribua valores a esse array e, em seguida, exiba os valores no console.

Código de solução:

```
public class CriacaoInicializacaoArray {
    public static void main(String[] args) {
        int[] numeros = {10, 20, 30, 40, 50};

        for (int i = 0; i < numeros.length; i++) {
            System.out.println("Elemento " + i + ": " + numeros[i]);
        }
    }
}
```

Explicação: O programa cria um array de 5 números inteiros e inicializa seus valores diretamente. Um laço for é usado para percorrer e exibir cada valor do array.

Exercício 66: Soma de Elementos de um Array

Enunciado: Desenvolva um programa que leia 5 números inteiros do usuário, armazene-os em um array e calcule a soma de todos os elementos.

Código de solução:

```java
import java.util.Scanner;

public class SomaElementosArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[5];
        int soma = 0;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
            soma += numeros[i];
        }

        System.out.println("A soma dos elementos do array é: " + soma);
    }
}
```

Explicação: O programa solicita que o usuário insira 5 números inteiros, que são armazenados em um array. Em seguida, a soma de todos os elementos do array é calculada e exibida.

Exercício 67: Média de Valores

Enunciado: Escreva um programa que leia 10 números inteiros e calcule a média dos valores inseridos, utilizando um array para armazenar os números.

Código de solução:

```java
import java.util.Scanner;

public class MediaDeValores {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
        int soma = 0;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
            soma += numeros[i];
```

```
        }

        double media = soma / (double) numeros.length;
        System.out.println("A média dos valores é: " + media);
    }
}
```

Explicação: Este programa lê 10 números inteiros, armazena-os em um array e calcula a média somando os valores e dividindo pela quantidade de elementos no array.

Exercício 68: Valores Pares em um Array

Enunciado: Crie um programa que leia 8 números inteiros e exiba todos os valores pares armazenados no array.

Código de solução:

```
import java.util.Scanner;

public class ValoresParesArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[8];

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        System.out.println("Números pares no array:");
        for (int numero : numeros) {
            if (numero % 2 == 0) {
                System.out.println(numero);
            }
        }
    }
}
```

Explicação: O programa lê 8 números inteiros do usuário e armazena-os em um array. Depois, percorre o array e exibe apenas os números que são pares (divisíveis por 2).

Exercício 69: Menor e Maior Valor

Enunciado: Desenvolva um programa que leia 10 números inteiros e armazene-os em um array. Encontre e exiba o menor e o maior valor presentes no array.

Código de solução:

```
import java.util.Scanner;

public class MenorMaiorValor {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
        int maior, menor;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        maior = menor = numeros[0];

        for (int numero : numeros) {
            if (numero > maior) {
                maior = numero;
            }
            if (numero < menor) {
                menor = numero;
            }
        }

        System.out.println("Maior valor: " + maior);
        System.out.println("Menor valor: " + menor);
    }
}
```

Explicação: O programa lê 10 números inteiros e encontra o maior e o menor valor no array, usando um laço for-each para comparar os elementos.

Exercício 70: Contagem de Valores Positivos e Negativos

Enunciado: Escreva um programa que leia 15 números inteiros e, em seguida, exiba quantos desses números são positivos e quantos são negativos.

Código de solução:

```
import java.util.Scanner;

public class PositivosNegativos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[15];
        int positivos = 0, negativos = 0;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        for (int numero : numeros) {
            if (numero > 0) {
                positivos++;
            } else if (numero < 0) {
                negativos++;
            }
        }

        System.out.println("Quantidade de números positivos: " +
positivos);
        System.out.println("Quantidade de números negativos: " +
negativos);
    }
}
```

Explicação: O programa lê 15 números inteiros e conta quantos são positivos e quantos são negativos, exibindo o resultado final.

Exercício 71: Inversão de Array

Enunciado: Crie um programa que leia 6 números inteiros e armazene-os em um array. Depois, exiba os valores do array na ordem inversa.

Código de solução:

```java
import java.util.Scanner;

public class InversaoArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[6];

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        System.out.println("Números em ordem inversa:");
        for (int i = numeros.length - 1; i >= 0; i--) {
            System.out.println(numeros[i]);
        }
    }
}
```

Explicação: O programa lê 6 números inteiros, armazena-os em um array e, em seguida, exibe os elementos do array na ordem inversa, começando do último elemento.

Exercício 72: Contagem de Ocorrências de um Número

Enunciado: Desenvolva um programa que leia 10 números inteiros e armazene-os em um array. O programa deve pedir ao usuário para inserir um número extra e contar quantas vezes esse número aparece no array.

Código de solução:

```java
import java.util.Scanner;

public class ContagemOcorrencias {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
        int ocorrencias = 0;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        System.out.print("Digite um número para contar as ocorrências:
");
        int numeroProcurado = scanner.nextInt();

        for (int numero : numeros) {
            if (numero == numeroProcurado) {
                ocorrencias++;
            }
        }

        System.out.println("O número " + numeroProcurado + " aparece " +
ocorrencias + " vezes no array.");
    }
}
```

Explicação: O programa lê 10 números inteiros do usuário e, em seguida, solicita um número adicional. Ele percorre o array para contar quantas vezes esse número aparece.

Exercício 73: Duplicação de Valores

Enunciado: Escreva um programa que crie um array de 5 números inteiros e multiplique todos os seus valores por 2, exibindo o novo array no console.

Código de solução:

```java
import java.util.Scanner;
```

```java
public class DuplicacaoDeValores {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[5];

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        System.out.println("Valores duplicados:");
        for (int i = 0; i < numeros.length; i++) {
            numeros[i] *= 2;
            System.out.println(numeros[i]);
        }
    }
}
```

Explicação: O programa lê 5 números inteiros, multiplica cada número por 2, e exibe os novos valores no console.

Exercício 74: Elementos em Posições Ímpares

Enunciado: Crie um programa que leia 10 números inteiros e exiba apenas os valores que estão em posições ímpares no array.

Código de solução:

```java
import java.util.Scanner;

public class ElementosPosicoesImpares {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        System.out.println("Valores nas posições ímpares:");
```

```
        for (int i = 1; i < numeros.length; i += 2) {
            System.out.println(numeros[i]);
        }
    }
}
```

Explicação: O programa lê 10 números inteiros e exibe apenas os elementos que estão nas posições ímpares do array (índices 1, 3, 5, etc.).

Exercício 75: Substituição de Valores em um Array

Enunciado: Desenvolva um programa que crie um array de 10 números inteiros. O programa deve pedir ao usuário que forneça dois números: um número para buscar no array e outro para substituir o número encontrado. Se o número for encontrado, ele deve ser substituído.

Código de solução:

```
import java.util.Scanner;

public class SubstituicaoValoresArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        System.out.print("Digite o número a ser substituído: ");
        int numeroAntigo = scanner.nextInt();
        System.out.print("Digite o novo número: ");
        int numeroNovo = scanner.nextInt();

        boolean encontrado = false;
        for (int i = 0; i < numeros.length; i++) {
            if (numeros[i] == numeroAntigo) {
                numeros[i] = numeroNovo;
```

```
                encontrado = true;
            }
        }

        if (encontrado) {
            System.out.println("O número foi substituído. Novo array:");
            for (int numero : numeros) {
                System.out.println(numero);
            }
        } else {
            System.out.println("Número não encontrado no array.");
        }
    }
}
```

Explicação: O programa solicita ao usuário que forneça um número para buscar e substituir no array. Se o número for encontrado, ele é substituído e o array atualizado é exibido.

Exercício 76: Verificação de Elementos Repetidos

Enunciado: Escreva um programa que leia 10 números inteiros e verifique se algum valor é repetido no array. Se houver repetições, exiba uma mensagem informando.

Código de solução:

```
import java.util.Scanner;

public class VerificacaoRepetidos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
        boolean repetido = false;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }
```

```
        for (int i = 0; i < numeros.length; i++) {
            for (int j = i + 1; j < numeros.length; j++) {
                if (numeros[i] == numeros[j]) {
                    repetido = true;
                    break;
                }
            }
        }

        if (repetido) {
            System.out.println("Há valores repetidos no array.");
        } else {
            System.out.println("Não há valores repetidos no array.");
        }
    }
}
```

Explicação: O programa compara cada valor do array com os outros, verificando se há repetições. Se algum valor repetido for encontrado, o programa exibe uma mensagem indicando isso.

Exercício 77: Produto dos Elementos de um Array

Enunciado: Crie um programa que leia 6 números inteiros e calcule o produto de todos os valores do array.

Código de solução:

```
import java.util.Scanner;

public class ProdutoElementosArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[6];
        int produto = 1;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
            produto *= numeros[i];
```

```
        }

        System.out.println("O produto dos elementos do array é: " +
produto);
    }
}
```

Explicação: O programa lê 6 números inteiros e calcula o produto (multiplicação) de todos os valores armazenados no array.

Exercício 78: Comparação de Arrays

Enunciado: Desenvolva um programa que crie dois arrays de 5 números inteiros. O programa deve comparar os dois arrays e exibir quais posições possuem valores iguais.

Código de solução:

```
import java.util.Scanner;

public class ComparacaoArrays {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] array1 = new int[5];
        int[] array2 = new int[5];

        System.out.println("Preencha o primeiro array:");
        for (int i = 0; i < array1.length; i++) {
            System.out.print("Elemento " + (i + 1) + ": ");
            array1[i] = scanner.nextInt();
        }

        System.out.println("Preencha o segundo array:");
        for (int i = 0; i < array2.length; i++) {
            System.out.print("Elemento " + (i + 1) + ": ");
            array2[i] = scanner.nextInt();
        }

        System.out.println("Comparação de valores nas mesmas
posições:");
```

```
        for (int i = 0; i < array1.length; i++) {
            if (array1[i] == array2[i]) {
                System.out.println("Posição " + i + ": " + array1[i] + "
= " + array2[i]);
            }
        }
    }
}
```

Explicação: O programa lê dois arrays e compara seus elementos nas mesmas posições, exibindo quando os valores são iguais.

Exercício 79: Verificação de Ordem Crescente

Enunciado: Escreva um programa que leia 8 números inteiros e verifique se os valores estão em ordem crescente. Exiba uma mensagem indicando se os números estão ou não em ordem.

Código de solução:

```
import java.util.Scanner;

public class VerificacaoOrdemCrescente {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[8];
        boolean emOrdem = true;

        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        for (int i = 0; i < numeros.length - 1; i++) {
            if (numeros[i] > numeros[i + 1]) {
                emOrdem = false;
                break;
            }
        }
```

```
        if (emOrdem) {
            System.out.println("Os números estão em ordem crescente.");
        } else {
            System.out.println("Os números NÃO estão em ordem
crescente.");
        }
    }
}
```

Explicação: O programa verifica se os números lidos estão em ordem crescente, comparando cada elemento com o próximo no array. Se algum número estiver fora de ordem, ele exibe uma mensagem indicando isso.

Arrays Multidimensionais (2D) (10 exercícios)

Dica: Arrays multidimensionais podem ser criados utilizando a sintaxe: int[][] x = [2][2], sendo assim este array terá 2 linhas e duas colunas.

Exercício 80: Criação de uma Matriz

Enunciado: Crie um programa que declare uma matriz 3x3 e permita que o usuário insira valores inteiros para preencher essa matriz. Em seguida, exiba os valores da matriz no console.

Código de solução:

```
import java.util.Scanner;

public class CriacaoMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[3][3];

        // Preenchendo a matriz
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
            }
```

```
        }

        // Exibindo a matriz
        System.out.println("Matriz 3x3:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Explicação: O programa permite que o usuário preencha uma matriz 3x3 com valores inteiros e depois exibe a matriz no console em formato de tabela.

Exercício 81: Soma de Elementos de uma Matriz

Enunciado: Desenvolva um programa que leia uma matriz 3x3 de inteiros e calcule a soma de todos os elementos da matriz.

Código de solução:

```
import java.util.Scanner;

public class SomaMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[3][3];
        int soma = 0;

        // Preenchendo a matriz
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
                soma += matriz[i][j];
            }
        }

        // Exibindo a soma dos elementos
        System.out.println("A soma de todos os elementos da matriz é: "
+ soma);
```

```
    }
}
```

Explicação: O programa lê uma matriz 3x3, calcula a soma de todos os elementos da matriz e exibe o resultado.

Exercício 82: Soma das Linhas e Colunas

Enunciado: Escreva um programa que leia uma matriz 3x3 e exiba a soma dos elementos de cada linha e de cada coluna.

Código de solução:

```
import java.util.Scanner;

public class SomaLinhasColunas {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[3][3];
        int[] somaLinhas = new int[3];
        int[] somaColunas = new int[3];

        // Preenchendo a matriz
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
                somaLinhas[i] += matriz[i][j];
                somaColunas[j] += matriz[i][j];
            }
        }

        // Exibindo a soma das linhas
        for (int i = 0; i < 3; i++) {
            System.out.println("Soma da linha " + i + ": " +
somaLinhas[i]);
        }

        // Exibindo a soma das colunas
        for (int j = 0; j < 3; j++) {
            System.out.println("Soma da coluna " + j + ": " +
somaColunas[j]);
        }
```

```
    }
}
```

Explicação: Este programa lê uma matriz 3x3 e, durante o preenchimento, acumula a soma de cada linha e coluna. Em seguida, exibe a soma de cada uma delas separadamente.

Exercício 83: Matriz Identidade

Enunciado: Crie um programa que gere e exiba uma matriz identidade 4x4 (valores 1 na diagonal principal e 0 nos outros elementos).

Código de solução:

```
public class MatrizIdentidade {
    public static void main(String[] args) {
        int[][] matriz = new int[4][4];

        // Gerando matriz identidade
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (i == j) {
                    matriz[i][j] = 1;
                } else {
                    matriz[i][j] = 0;
                }
            }
        }

        // Exibindo a matriz identidade
        System.out.println("Matriz identidade 4x4:");
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                System.out.print(matriz[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Explicação: O programa gera uma matriz identidade 4x4, onde os elementos da diagonal principal são 1 e todos os outros elementos são 0. Em seguida, exibe a matriz no console.

Exercício 84: Busca em uma Matriz

Enunciado: Desenvolva um programa que permita ao usuário preencher uma matriz 4x4 com valores inteiros. O programa deve pedir ao usuário um número para buscar na matriz e informar em qual posição ele foi encontrado (linha e coluna).

Código de solução:

```
import java.util.Scanner;

public class BuscaMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[4][4];

        // Preenchendo a matriz
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
            }
        }

        // Buscando um número na matriz
        System.out.print("Digite o número para buscar: ");
        int numero = scanner.nextInt();
        boolean encontrado = false;

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                if (matriz[i][j] == numero) {
                    System.out.println("Número encontrado na posição ["
+ i + "][" + j + "]");
                    encontrado = true;
                }
            }
        }

        if (!encontrado) {
            System.out.println("Número não encontrado na matriz.");
        }
    }
}
```

Explicação: O programa permite que o usuário preencha uma matriz 4x4 com valores e, em seguida, realiza uma busca por um número fornecido pelo usuário. Se o número for encontrado, exibe a posição (linha e coluna) do número na matriz.

Exercício 85: Matriz Transposta

Enunciado: Escreva um programa que leia uma matriz 3x3 e exiba a sua matriz transposta (inversão das linhas com as colunas).

Código de solução:

```
import java.util.Scanner;

public class MatrizTransposta {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[3][3];
        int[][] transposta = new int[3][3];

        // Preenchendo a matriz
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
                transposta[j][i] = matriz[i][j];  // Preenchendo a
transposta
            }
        }

        // Exibindo a matriz transposta
        System.out.println("Matriz transposta 3x3:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(transposta[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Explicação: O programa lê uma matriz 3x3 e, enquanto preenche a matriz original, também calcula a matriz transposta, invertendo as linhas e colunas. Em seguida, exibe a matriz transposta.

Exercício 86: Diagonal Principal e Secundária

Enunciado: Crie um programa que leia uma matriz 4x4 e exiba os elementos da diagonal principal e da diagonal secundária.

Código de solução:

```
import java.util.Scanner;

public class DiagonalMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[4][4];

        // Preenchendo a matriz
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
            }
        }

        // Exibindo a diagonal principal
        System.out.println("Diagonal principal:");
        for (int i = 0; i < 4; i++) {
            System.out.println(matriz[i][i]);
        }

        // Exibindo a diagonal secundária
        System.out.println("Diagonal secundária:");
        for (int i = 0; i < 4; i++) {
            System.out.println(matriz[i][3 - i]);
        }
    }
}
```

Explicação: O programa lê uma matriz 4x4 e exibe os elementos da diagonal principal (que vai da posição superior esquerda até a inferior direita) e da diagonal secundária (que vai da posição superior direita até a inferior esquerda).

Exercício 87: Multiplicação de Matrizes

Enunciado: Desenvolva um programa que leia duas matrizes 2x2 e calcule o produto entre elas, exibindo o resultado.

Código de solução:

```
import java.util.Scanner;

public class MultiplicacaoMatrizes {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matrizA = new int[2][2];
        int[][] matrizB = new int[2][2];
        int[][] resultado = new int[2][2];

        // Preenchendo a matriz A
        System.out.println("Preenchendo a matriz A:");
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matrizA[i][j] = scanner.nextInt();
            }
        }

        // Preenchendo a matriz B
        System.out.println("Preenchendo a matriz B:");
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matrizB[i][j] = scanner.nextInt();
            }
        }

        // Multiplicação das matrizes A e B
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                resultado[i][j] = 0;
                for (int k = 0; k < 2; k++) {
                    resultado[i][j] += matrizA[i][k] * matrizB[k][j];
                }
            }
        }

        // Exibindo o resultado da multiplicação
        System.out.println("Resultado da multiplicação das matrizes A e
```

```
B:");
        for (int i = 0; i < 2; i++) {
            for (int j = 0; j < 2; j++) {
                System.out.print(resultado[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Explicação: O programa lê duas matrizes 2x2, realiza a multiplicação entre elas (usando a regra de multiplicação de matrizes) e exibe a matriz resultante.

Exercício 88: Soma de Duas Matrizes

Enunciado: Escreva um programa que leia duas matrizes 3x3 e calcule a soma entre elas, exibindo a matriz resultante.

Código de solução:

```
import java.util.Scanner;

public class SomaMatrizes {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matrizA = new int[3][3];
        int[][] matrizB = new int[3][3];
        int[][] soma = new int[3][3];

        // Preenchendo a matriz A
        System.out.println("Preenchendo a matriz A:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matrizA[i][j] = scanner.nextInt();
            }
        }

        // Preenchendo a matriz B
        System.out.println("Preenchendo a matriz B:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print("Digite o valor para a posição [" + i +
```

```
"][" + j + "]: ");
                matrizB[i][j] = scanner.nextInt();
            }
        }

        // Somando as duas matrizes
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                soma[i][j] = matrizA[i][j] + matrizB[i][j];
            }
        }

        // Exibindo a matriz resultante
        System.out.println("Soma das matrizes A e B:");
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                System.out.print(soma[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Explicação: O programa lê duas matrizes 3x3, soma elemento por elemento de cada uma e exibe a matriz resultante, que contém a soma de ambas as matrizes.

Exercício 89: Contagem de Elementos Pares

Enunciado: Crie um programa que leia uma matriz 5x5 e conte quantos números pares existem na matriz. Exiba o total de números pares encontrados.

Código de solução:

```
import java.util.Scanner;

public class ContagemParesMatriz {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[][] matriz = new int[5][5];
        int contagemPares = 0;

        // Preenchendo a matriz
        for (int i = 0; i < 5; i++) {
            for (int j = 0; j < 5; j++) {
```

```
                System.out.print("Digite o valor para a posição [" + i +
"][" + j + "]: ");
                matriz[i][j] = scanner.nextInt();
                if (matriz[i][j] % 2 == 0) {
                    contagemPares++;
                }
            }
        }

        // Exibindo o número de pares
        System.out.println("Total de números pares na matriz: " +
contagemPares);
    }
}
```

Explicação: O programa lê uma matriz 5x5, conta quantos elementos pares existem na matriz e exibe o total de números pares encontrados.

Manipulação de Arrays (busca, ordenação, etc.)

Exercício 90: Busca Linear em um Array

Enunciado: Crie um programa que leia 10 números inteiros e um número adicional. O programa deve realizar uma busca linear no array para verificar se o número adicional está presente. Exiba a posição do número, se encontrado.

Código de solução:

```
import java.util.Scanner;

public class BuscaLinear {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
        boolean encontrado = false;

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }
```

```
        // Lendo o número adicional
        System.out.print("Digite o número que deseja buscar: ");
        int numeroBuscado = scanner.nextInt();

        // Realizando a busca linear
        for (int i = 0; i < numeros.length; i++) {
            if (numeros[i] == numeroBuscado) {
                System.out.println("Número encontrado na posição: " +
i);
                encontrado = true;
                break;
            }
        }

        if (!encontrado) {
            System.out.println("Número não encontrado.");
        }
    }
}
```

Explicação: A busca linear percorre o array elemento por elemento até encontrar o número buscado. O algoritmo é simples e funciona mesmo para arrays desordenados, pois verifica cada elemento individualmente. O programa lê 10 números inteiros, insere um número adicional para buscar, e caso o número esteja presente, ele exibe a posição. A busca linear tem complexidade de tempo O(n), onde n é o número de elementos no array, o que significa que, no pior caso, pode ser necessário percorrer todos os elementos para encontrar o número.

Exercício 91: Busca Binária em um Array Ordenado

Enunciado: Desenvolva um programa que leia 10 números inteiros, os ordene em ordem crescente e, em seguida, utilize o método de busca binária para encontrar um número fornecido pelo usuário. Exiba a posição do número se ele for encontrado.

Código de solução:

```
import java.util.Arrays;
import java.util.Scanner;

public class BuscaBinaria {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
```

```
        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Ordenando o array
        Arrays.sort(numeros);

        // Lendo o número a ser buscado
        System.out.print("Digite o número que deseja buscar: ");
        int numeroBuscado = scanner.nextInt();

        // Realizando a busca binária
        int posicao = Arrays.binarySearch(numeros, numeroBuscado);

        if (posicao >= 0) {
            System.out.println("Número encontrado na posição: " +
posicao);
        } else {
            System.out.println("Número não encontrado.");
        }
    }
}
```

Explicação: A busca binária só pode ser aplicada em arrays ordenados. A estratégia divide repetidamente o array ao meio, verificando se o número buscado é maior, menor ou igual ao valor central. Se o número for maior, a busca continua na metade direita do array; se for menor, na metade esquerda. A busca binária tem complexidade de tempo O(log n), o que a torna muito eficiente para grandes conjuntos de dados. No programa, o array é ordenado com Arrays.sort(), e a busca binária é realizada com Arrays.binarySearch(). A função retorna o índice se o número for encontrado e um valor negativo se não for encontrado.

Exercício 92: Ordenação de um Array com Bubble Sort

Enunciado: Implemente o algoritmo de ordenação Bubble Sort para ordenar um array de 10 números inteiros em ordem crescente.

Código de solução:

```
import java.util.Scanner;

public class BubbleSort {
```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Algoritmo de ordenação Bubble Sort
        for (int i = 0; i < numeros.length - 1; i++) {
            for (int j = 0; j < numeros.length - 1 - i; j++) {
                if (numeros[j] > numeros[j + 1]) {
                    int temp = numeros[j];
                    numeros[j] = numeros[j + 1];
                    numeros[j + 1] = temp;
                }
            }
        }

        // Exibindo o array ordenado
        System.out.println("Array ordenado:");
        for (int numero : numeros) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: O Bubble Sort é um dos algoritmos de ordenação mais simples, mas também um dos menos eficientes em grandes conjuntos de dados, com complexidade de tempo $O(n^2)$. O algoritmo percorre o array diversas vezes e compara elementos adjacentes, trocando-os se estiverem fora de ordem. A cada iteração, o maior elemento "bolha" até o final do array. O processo continua até que o array esteja completamente ordenado. No programa, a cada iteração interna, o maior valor é colocado em sua posição final, até que todo o array esteja ordenado.

Exercício 93: Ordenação de um Array com Selection Sort

Enunciado: Escreva um programa que leia 10 números inteiros e os ordene utilizando o algoritmo Selection Sort. Exiba o array ordenado ao final.

Código de solução:

```java
import java.util.Scanner;

public class SelectionSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Algoritmo de ordenação Selection Sort
        for (int i = 0; i < numeros.length - 1; i++) {
            int minIndex = i;
            for (int j = i + 1; j < numeros.length; j++) {
                if (numeros[j] < numeros[minIndex]) {
                    minIndex = j;
                }
            }

            // Troca o menor elemento com o elemento da posição i
            int temp = numeros[i];
            numeros[i] = numeros[minIndex];
            numeros[minIndex] = temp;
        }

        // Exibindo o array ordenado
        System.out.println("Array ordenado:");
        for (int numero : numeros) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: O Selection Sort é um algoritmo de ordenação que seleciona o menor (ou maior, dependendo da ordem) elemento de uma lista e o coloca na primeira posição. O processo se repete para o restante do array. Em cada iteração, o algoritmo percorre o array para encontrar o menor elemento e troca-o com o primeiro elemento não ordenado. A complexidade do Selection Sort também é $O(n^2)$, mas ele é mais eficiente em termos de número de trocas, pois faz, no máximo, n-1 trocas. No programa, o menor valor encontrado em cada iteração é colocado na sua posição correta no array.

Exercício 94: Ordenação de um Array com Insertion Sort

Enunciado: Desenvolva um programa que leia 10 números inteiros e os ordene utilizando o algoritmo Insertion Sort. Exiba o array ordenado ao final.

Código de solução:

```
import java.util.Scanner;

public class InsertionSort {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Algoritmo de ordenação Insertion Sort
        for (int i = 1; i < numeros.length; i++) {
            int chave = numeros[i]; // Número a ser inserido na posição
correta
            int j = i - 1;

            // Desloca os elementos maiores que a chave uma posição à
frente
            while (j >= 0 && numeros[j] > chave) {
                numeros[j + 1] = numeros[j];
                j--;
            }
            numeros[j + 1] = chave; // Insere a chave na posição correta
        }

        // Exibindo o array ordenado
        System.out.println("Array ordenado:");
        for (int numero : numeros) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: O Insertion Sort é um algoritmo de ordenação que constrói o array final um elemento por vez. Ele pega cada elemento do array e o insere na posição correta em

relação aos elementos que já foram ordenados. Em cada iteração, o algoritmo seleciona um elemento e o compara com os elementos à sua esquerda, movendo-os se necessário, até encontrar a posição correta para inseri-lo. A complexidade de tempo é O(n²) no pior caso, mas é eficiente para listas pequenas ou quase ordenadas, pois faz menos trocas do que outros algoritmos, como o Bubble Sort.

Exercício 95: Inversão de um Array

Enunciado: Crie um programa que leia 10 números inteiros, armazene-os em um array e inverta a ordem dos elementos no array. Exiba o array invertido.

Código de solução:

```java
import java.util.Scanner;

public class InversaoArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Invertendo o array
        for (int i = 0; i < numeros.length / 2; i++) {
            int temp = numeros[i];
            numeros[i] = numeros[numeros.length - 1 - i];
            numeros[numeros.length - 1 - i] = temp;
        }

        // Exibindo o array invertido
        System.out.println("Array invertido:");
        for (int numero : numeros) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: Para inverter um array, o algoritmo percorre metade do array e troca cada elemento da primeira metade com o elemento correspondente da segunda metade. Isso é feito usando uma variável temporária (temp) para armazenar temporariamente um valor enquanto ocorre a troca. A inversão tem complexidade de tempo O(n), pois cada elemento

é visitado uma única vez. O programa lê 10 números, realiza as trocas e exibe o array invertido.

Exercício 96: Remoção de Elemento de um Array

Enunciado: Escreva um programa que leia 10 números inteiros e remova um número específico informado pelo usuário. Após a remoção, exiba o array resultante.

Código de solução:

```
import java.util.Scanner;

public class RemocaoElementoArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Lendo o número a ser removido
        System.out.print("Digite o número a ser removido: ");
        int numeroRemover = scanner.nextInt();
        boolean encontrado = false;

        // Criando um novo array sem o número removido
        int[] novoArray = new int[numeros.length - 1];
        int indiceNovo = 0;

        for (int i = 0; i < numeros.length; i++) {
            if (numeros[i] == numeroRemover && !encontrado) {
                encontrado = true;
            } else {
                novoArray[indiceNovo++] = numeros[i];
            }
        }

        // Exibindo o novo array
        if (encontrado) {
            System.out.println("Array após remoção:");
            for (int numero : novoArray) {
```

```
                System.out.print(numero + " ");
            }
        } else {
            System.out.println("Número não encontrado no array.");
        }
    }
}
```

Explicação: O programa lê 10 números e um número específico para ser removido. Se o número for encontrado, ele é ignorado na cópia para um novo array, enquanto os demais números são copiados normalmente. A remoção é feita criando um novo array com tamanho reduzido em um, e o número é removido ao não ser copiado para o novo array. A complexidade de tempo é O(n), já que o programa precisa percorrer todo o array para realizar a remoção.

Exercício 97: Inserção de Elemento em um Array Ordenado

Enunciado: Desenvolva um programa que leia 10 números inteiros, os ordene em ordem crescente e insira um novo número no array mantendo a ordem. Exiba o array resultante.

Código de solução:

```
import java.util.Arrays;
import java.util.Scanner;

public class InsercaoElementoOrdenado {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Ordenando o array
        Arrays.sort(numeros);

        // Lendo o número a ser inserido
        System.out.print("Digite o número a ser inserido: ");
        int numeroInserir = scanner.nextInt();

        // Criando um novo array com espaço adicional
```

```
        int[] novoArray = new int[numeros.length + 1];
        int i = 0, j = 0;

        // Inserindo o número no array ordenado
        while (i < numeros.length && numeros[i] < numeroInserir) {
            novoArray[j++] = numeros[i++];
        }
        novoArray[j++] = numeroInserir; // Insere o novo número
        while (i < numeros.length) {
            novoArray[j++] = numeros[i++];
        }

        // Exibindo o novo array
        System.out.println("Array após a inserção:");
        for (int numero : novoArray) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: O programa lê um array de 10 números, os ordena, e insere um novo número mantendo a ordem. A inserção é feita percorrendo o array até encontrar a posição correta, copiando os números menores para o novo array, inserindo o novo número, e em seguida copiando o restante. A complexidade de tempo é O(n), pois é necessário percorrer o array original uma vez para encontrar a posição de inserção e copiar os elementos.

Exercício 98: Contagem de Elementos em um Intervalo

Enunciado: Crie um programa que leia 15 números inteiros e conte quantos números estão dentro de um intervalo fornecido pelo usuário. Exiba a quantidade.

Código de solução:

```
import java.util.Scanner;

public class ContagemIntervalo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[15];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
```

```
        }

        // Lendo o intervalo
        System.out.print("Digite o limite inferior do intervalo: ");
        int limiteInferior = scanner.nextInt();
        System.out.print("Digite o limite superior do intervalo: ");
        int limiteSuperior = scanner.nextInt();

        // Contando os números no intervalo
        int contagem = 0;
        for (int numero : numeros) {
            if (numero >= limiteInferior && numero <= limiteSuperior) {
                contagem++;
            }
        }

        // Exibindo a contagem
        System.out.println("Quantidade de números no intervalo: " +
contagem);
    }
}
```

Explicação: O programa lê 15 números e dois valores que definem um intervalo. Ele percorre o array verificando se cada número está dentro desse intervalo, e, se estiver, incrementa um contador. A complexidade de tempo é O(n), já que todos os números precisam ser verificados para determinar se estão no intervalo.

Exercício 99: Duplicação de Elementos de um Array

Enunciado: Escreva um programa que leia 10 números inteiros e duplique todos os valores do array. Exiba o array duplicado.

Código de solução:

```
import java.util.Scanner;

public class DuplicacaoArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
```

```
            numeros[i] = scanner.nextInt();
        }

        // Duplicando os valores
        for (int i = 0; i < numeros.length; i++) {
            numeros[i] *= 2;
        }

        // Exibindo o array duplicado
        System.out.println("Array após a duplicação:");
        for (int numero : numeros) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: O programa lê 10 números e duplica cada valor multiplicando-o por 2. A operação é feita diretamente no array, e o resultado é exibido no console. O algoritmo é simples, com complexidade de tempo O(n), pois percorre todo o array uma vez para realizar a multiplicação.

Exercício 100: Interseção de Dois Arrays

Enunciado: Desenvolva um programa que leia dois arrays de 10 números inteiros e exiba a interseção entre os dois arrays (os números que aparecem em ambos).

Código de solução:

```
import java.util.Scanner;

public class IntersecaoArrays {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] arrayA = new int[10];
        int[] arrayB = new int[10];

        // Preenchendo o array A
        System.out.println("Preencha o array A:");
        for (int i = 0; i < arrayA.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            arrayA[i] = scanner.nextInt();
        }
```

```
        // Preenchendo o array B
        System.out.println("Preencha o array B:");
        for (int i = 0; i < arrayB.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            arrayB[i] = scanner.nextInt();
        }

        // Encontrando a interseção
        System.out.println("Interseção dos arrays:");
        for (int i = 0; i < arrayA.length; i++) {
            for (int j = 0; j < arrayB.length; j++) {
                if (arrayA[i] == arrayB[j]) {
                    System.out.print(arrayA[i] + " ");
                    break;
                }
            }
        }
    }
}
```

Explicação: O programa compara os elementos de dois arrays para encontrar os valores que estão presentes em ambos (interseção). Para isso, percorre cada elemento do primeiro array e verifica se ele está presente no segundo array. A complexidade de tempo é $O(n^2)$, pois para cada elemento de um array, o programa percorre o segundo array.

Exercício 101: União de Dois Arrays

Enunciado: Crie um programa que leia dois arrays de 10 números inteiros e exiba a união dos dois arrays (todos os números, sem repetição).

Código de solução:

```
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class UniaoArrays {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] arrayA = new int[10];
        int[] arrayB = new int[10];
        Set<Integer> uniao = new HashSet<>();

        // Preenchendo o array A
```

```
        System.out.println("Preencha o array A:");
        for (int i = 0; i < arrayA.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            arrayA[i] = scanner.nextInt();
            uniao.add(arrayA[i]);
        }

        // Preenchendo o array B
        System.out.println("Preencha o array B:");
        for (int i = 0; i < arrayB.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            arrayB[i] = scanner.nextInt();
            uniao.add(arrayB[i]);
        }

        // Exibindo a união dos arrays
        System.out.println("União dos arrays (sem repetição):");
        for (int numero : uniao) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: Este programa cria a união de dois arrays, ou seja, inclui todos os números presentes em ambos, sem repetição. A estrutura HashSet é usada para garantir que não haja duplicatas, pois a inserção em um Set não permite valores repetidos. A complexidade de tempo é O(n) para inserir todos os elementos no Set, onde n é o número de elementos no array.

Exercício 102: Frequência de Elementos em um Array

Enunciado: Escreva um programa que leia 10 números inteiros e exiba a frequência de cada número, ou seja, quantas vezes cada número aparece no array.

Código de solução:

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class FrequenciaElementos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        int[] numeros = new int[10];
        Map<Integer, Integer> frequencia = new HashMap<>();

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
            frequencia.put(numeros[i],
frequencia.getOrDefault(numeros[i], 0) + 1);
        }

        // Exibindo a frequência de cada número
        System.out.println("Frequência de cada número:");
        for (Map.Entry<Integer, Integer> entry : frequencia.entrySet())
{
            System.out.println("Número " + entry.getKey() + " aparece "
+ entry.getValue() + " vez(es).");
        }
    }
}
```

Explicação: O programa utiliza um HashMap para armazenar a frequência de cada número. O método getOrDefault é utilizado para verificar se um número já está no mapa. Se o número já existir, o valor correspondente (frequência) é incrementado. Caso contrário, o número é adicionado com valor inicial 1. A complexidade de tempo é O(n), pois cada número é inserido e contado uma vez.

Exercício 103: Separação de Pares e Ímpares

Enunciado: Desenvolva um programa que leia 10 números inteiros e separe os números pares dos números ímpares em dois arrays diferentes. Exiba os dois arrays.

Código de solução:

```
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class SeparacaoParesImpares {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];
        List<Integer> pares = new ArrayList<>();
```

```
        List<Integer> impares = new ArrayList<>();

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Separando os números pares e ímpares
        for (int numero : numeros) {
            if (numero % 2 == 0) {
                pares.add(numero);
            } else {
                impares.add(numero);
            }
        }

        // Exibindo os números pares
        System.out.println("Números pares:");
        for (int par : pares) {
            System.out.print(par + " ");
        }
        System.out.println();

        // Exibindo os números ímpares
        System.out.println("Números ímpares:");
        for (int impar : impares) {
            System.out.print(impar + " ");
        }
    }
}
```

Explicação: O programa lê 10 números e os separa em dois arrays dinâmicos (ArrayList): um para os números pares e outro para os ímpares. Isso é feito verificando o resto da divisão por 2 de cada número. Os números são então adicionados às suas respectivas listas e exibidos ao final. A complexidade de tempo é O(n), já que cada número é verificado uma vez.

Exercício 104: Rotação de um Array

Enunciado: Crie um programa que leia 10 números inteiros e rotacione os elementos do array para a direita (o último elemento passa a ser o primeiro). Exiba o array após a rotação.

Código de solução:

```java
import java.util.Scanner;

public class RotacaoArray {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] numeros = new int[10];

        // Preenchendo o array
        for (int i = 0; i < numeros.length; i++) {
            System.out.print("Digite o número " + (i + 1) + ": ");
            numeros[i] = scanner.nextInt();
        }

        // Rotacionando o array
        int ultimoElemento = numeros[numeros.length - 1];
        for (int i = numeros.length - 1; i > 0; i--) {
            numeros[i] = numeros[i - 1];
        }
        numeros[0] = ultimoElemento;

        // Exibindo o array rotacionado
        System.out.println("Array após a rotação:");
        for (int numero : numeros) {
            System.out.print(numero + " ");
        }
    }
}
```

Explicação: O programa realiza uma rotação simples para a direita. O último elemento do array é armazenado em uma variável temporária, enquanto os outros elementos são deslocados uma posição para a direita. O último elemento é então colocado na primeira posição. A complexidade de tempo é O(n), já que todos os elementos são deslocados uma vez.

Conclusão do Capítulo 5: Arrays e Coleções

Neste capítulo, exploramos profundamente o uso de arrays unidimensionais e multidimensionais, além de técnicas avançadas de manipulação de arrays. Ao longo dos exercícios, os alunos foram desafiados a lidar com conceitos como declaração e inicialização de arrays, operações básicas de iteração, busca, e ordenação. A compreensão e implementação de algoritmos como Bubble Sort, Selection Sort, e Insertion Sort reforçam a capacidade de resolver problemas com eficiência.

Nos arrays multidimensionais, práticas como o uso de matrizes para cálculos de soma, multiplicação, e identificação de elementos em diagonais permitiram uma aplicação mais visual e complexa dos conceitos. A manipulação de arrays trouxe desafios mais avançados, como busca binária, remoção e inserção de elementos, rotação de arrays, além de interseção e união de arrays.

Capítulo 6: Métodos e Funções

Neste capítulo, exploraremos o conceito de métodos e funções em Java, elementos fundamentais para a construção de programas modulares e reutilizáveis. Através dos exercícios propostos, você aprenderá a declarar e chamar métodos, entender a passagem de parâmetros e como os métodos retornam valores, além de descobrir a sobrecarga de métodos, uma técnica que permite criar múltiplas versões de um mesmo método para diferentes tipos ou quantidades de parâmetros.

Dividido em três seções, este capítulo começará com a construção e uso básico de métodos, avançará para o conceito de parâmetros e valores de retorno, e finalizará com a sobrecarga, permitindo que o aluno ganhe domínio sobre o poder dos métodos em programação orientada a objetos. Os exercícios serão desafiadores e práticos, reforçando a importância de modularizar o código e promover a reutilização de trechos, tornando os programas mais organizados e eficientes.

Declaração e Chamada de Métodos

Exercício 105: Saudação Simples

Enunciado: Crie um método chamado saudacao() que exiba a mensagem "Olá, seja bem-vindo!" no console. Em seguida, no método main(), chame este método para exibir a saudação.

Solução:

```
public class SaudacaoSimples {
    public static void main(String[] args) {
        saudacao();  // Chamada do método
    }

    // Declaração do método saudacao
    public static void saudacao() {
        System.out.println("Olá, seja bem-vindo!");
```

```
    }
}
```

Explicação: Neste exercício, o objetivo é criar e chamar um método simples que exibe uma mensagem no console. O método saudacao() não recebe parâmetros e não retorna nenhum valor, sua única tarefa é exibir uma mensagem. Este tipo de método é usado para modularizar partes do código que podem ser reutilizadas em diferentes pontos, facilitando a manutenção e a legibilidade do programa.

Exercício 106: Exibir Número

Enunciado: Implemente um método chamado exibirNumero() que receba um número inteiro como parâmetro e exiba esse número no console. No método main(), chame o método passando diferentes valores para testar.

Solução:

```java
public class ExibirNumero {
    public static void main(String[] args) {
        exibirNumero(5);   // Chamando o método com o valor 5
        exibirNumero(20);  // Chamando o método com o valor 20
    }

    // Declaração do método exibirNumero
    public static void exibirNumero(int numero) {
        System.out.println("O número é: " + numero);
    }
}
```

Explicação: Aqui, estamos criando um método que recebe um parâmetro (um número inteiro) e o exibe no console. O conceito de passar parâmetros para métodos é fundamental, pois permite que o mesmo método seja reutilizado com diferentes valores. O método exibirNumero(int numero) é chamado duas vezes no main(), passando valores distintos (5 e 20), demonstrando como um único método pode ser reutilizado de forma eficiente para processar diferentes dados.

Exercício 107: Soma de Dois Números

Enunciado: Crie um método chamado somar() que receba dois números inteiros como parâmetros, calcule a soma deles e exiba o resultado no console. No método main(), solicite dois números do usuário, chame o método e exiba o resultado.

Solução:

```java
import java.util.Scanner;

public class SomaDoisNumeros {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número: ");
        int num1 = scanner.nextInt();

        System.out.print("Digite o segundo número: ");
        int num2 = scanner.nextInt();

        somar(num1, num2);  // Chamada do método somar
    }

    // Declaração do método somar
    public static void somar(int a, int b) {
        int soma = a + b;
        System.out.println("A soma é: " + soma);
    }
}
```

Explicação: Este exercício ensina como passar dois parâmetros para um método. O método somar(int a, int b) recebe dois números, calcula a soma e exibe o resultado no console. A interação com o usuário acontece no método main(), onde solicitamos os números, que são passados como argumentos para o método somar(). Esse exercício demonstra como os métodos ajudam a separar a lógica de cálculo da interação com o usuário, melhorando a organização e legibilidade do código.

Exercício 108: Verificação de Paridade

Enunciado: Implemente um método chamado verificarParidade() que receba um número inteiro como parâmetro e exiba se o número é par ou ímpar. No método main(), chame o método para testar com diferentes números.

Solução:

```java
public class VerificacaoParidade {
    public static void main(String[] args) {
        verificarParidade(7);  // Chamando o método com o valor 7
        verificarParidade(10); // Chamando o método com o valor 10
    }
```

```
    // Declaração do método verificarParidade
    public static void verificarParidade(int numero) {
        if (numero % 2 == 0) {
            System.out.println("O número " + numero + " é par.");
        } else {
            System.out.println("O número " + numero + " é ímpar.");
        }
    }
}
```

Explicação: Este exercício introduz a lógica de condicional dentro de métodos. O método verificarParidade(int numero) verifica se o número é par ou ímpar usando o operador % (módulo), que calcula o resto da divisão. Se o resto for 0, o número é par; caso contrário, é ímpar. Esse tipo de função é útil para encapsular lógica específica, o que facilita o uso repetido em diferentes partes do programa sem duplicação de código.

Exercício 109: Cálculo de Área de Retângulo

Enunciado: Desenvolva um método chamado calcularAreaRetangulo() que receba a largura e a altura de um retângulo como parâmetros e exiba a área no console. No método main(), solicite os valores ao usuário, chame o método e exiba o resultado.

Solução:

```
import java.util.Scanner;

public class CalculoAreaRetangulo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a largura do retângulo: ");
        double largura = scanner.nextDouble();

        System.out.print("Digite a altura do retângulo: ");
        double altura = scanner.nextDouble();

        calcularAreaRetangulo(largura, altura);  // Chamada do método
calcularAreaRetangulo
    }

    // Declaração do método calcularAreaRetangulo
    public static void calcularAreaRetangulo(double largura, double
```

```
altura) {
        double area = largura * altura;
        System.out.println("A área do retângulo é: " + area);
    }
}
```

Explicação: Neste exercício, o aluno aprende a passar múltiplos parâmetros de diferentes tipos para um método. O método calcularAreaRetangulo(double largura, double altura) calcula a área de um retângulo multiplicando a largura pela altura e exibe o resultado. Ao separar a lógica do cálculo da interação com o usuário, o código se torna mais modular e reutilizável. Além disso, ao usar tipos de dados double, o programa é capaz de trabalhar com valores decimais, o que amplia sua flexibilidade.

Passagem de Parâmetros e Retorno de Valores

Exercício 110: Retorno de Saudação

Enunciado: Crie um método chamado obterSaudacao() que retorne a string "Olá, seja bem-vindo!". No método main(), armazene o valor retornado em uma variável e exiba a saudação.

Solução:

```
public class SaudacaoComRetorno {
    public static void main(String[] args) {
        String saudacao = obterSaudacao();  // Chamando o método e
armazenando o retorno
        System.out.println(saudacao);       // Exibindo a saudação
    }

    // Declaração do método obterSaudacao
    public static String obterSaudacao() {
        return "Olá, seja bem-vindo!";
    }
}
```

Explicação: Neste exercício, o método obterSaudacao() retorna uma string contendo a mensagem de saudação. Em vez de simplesmente exibir a mensagem diretamente no método, ela é retornada e utilizada no main(). Esse conceito de retorno de valores é fundamental, pois permite que os métodos façam cálculos ou processem dados e devolvam o resultado para ser utilizado em outras partes do programa. Retornar valores em métodos

aumenta a flexibilidade e a reutilização do código, pois o mesmo método pode ser chamado em diversos contextos e o valor pode ser manipulado de diferentes formas.

Exercício 111: Soma com Retorno

Enunciado: Implemente um método chamado somar() que receba dois números inteiros como parâmetros, calcule a soma e retorne o resultado. No método main(), capture o valor retornado e exiba-o.

Solução:

```
public class SomaComRetorno {
    public static void main(String[] args) {
        int resultado = somar(10, 5);  // Chamada do método somar com
retorno
        System.out.println("A soma é: " + resultado);
    }

    // Declaração do método somar
    public static int somar(int a, int b) {
        return a + b;  // Retornando a soma
    }
}
```

Explicação: O método somar() recebe dois parâmetros inteiros, calcula a soma e retorna o resultado para o ponto de chamada, que neste caso é o main(). Esse conceito de retorno permite que o valor seja capturado em uma variável, podendo ser exibido ou reutilizado em outras partes do programa. O uso de parâmetros junto com o retorno cria métodos que são altamente reutilizáveis, permitindo a modularização do código.

Exercício 112: Verificar Paridade com Retorno

Enunciado: Desenvolva um método chamado ehPar() que receba um número inteiro como parâmetro e retorne true se o número for par e false se for ímpar. No método main(), utilize o valor retornado para exibir a mensagem correspondente.

Solução:

```
public class VerificacaoParidadeComRetorno {
    public static void main(String[] args) {
        boolean resultado = ehPar(10);  // Chamada do método ehPar com
```

```
retorno
        if (resultado) {
            System.out.println("O número é par.");
        } else {
            System.out.println("O número é ímpar.");
        }
    }

    // Declaração do método ehPar
    public static boolean ehPar(int numero) {
        return numero % 2 == 0;  // Retornando true se for par
    }
}
```

Explicação: Neste exercício, o método ehPar() retorna um boolean indicando se o número é par (true) ou ímpar (false). O conceito de retorno de valores booleanos permite que o método encapsule a lógica de verificação, e o main() use esse valor retornado para exibir a mensagem correta. Esse tipo de abordagem facilita a reutilização do método, tornando o código mais limpo e separando a lógica da exibição.

Exercício 113: Cálculo de Área com Retorno

Enunciado: Escreva um método chamado calcularAreaRetangulo() que receba a largura e a altura de um retângulo e retorne a área calculada. No método main(), exiba a área retornada.

Solução:

```
public class AreaRetanguloComRetorno {
    public static void main(String[] args) {
        double area = calcularAreaRetangulo(5.0, 3.0);  // Chamada do
método com retorno
        System.out.println("A área do retângulo é: " + area);
    }

    // Declaração do método calcularAreaRetangulo
    public static double calcularAreaRetangulo(double largura, double
altura) {
        return largura * altura;  // Retornando a área do retângulo
    }
}
```

Explicação: O método calcularAreaRetangulo() recebe a largura e a altura do retângulo, calcula a área (multiplicação dos dois valores) e retorna o resultado para o main(). O uso de

tipos double permite trabalhar com valores decimais, tornando o método flexível para diferentes entradas. Esse tipo de implementação separa a lógica do cálculo da exibição dos resultados, tornando o código mais modular e fácil de testar.

Exercício 114: Potenciação com Retorno

Enunciado: Implemente um método chamado potencia() que receba dois números inteiros (base e expoente) e retorne o resultado da potenciação. No método main(), capture e exiba o valor retornado.

Solução:

```
public class PotenciaComRetorno {
    public static void main(String[] args) {
        int resultado = potencia(2, 3);  // Chamada do método potencia
        System.out.println("O resultado da potenciação é: " +
resultado);
    }

    // Declaração do método potencia
    public static int potencia(int base, int expoente) {
        int resultado = 1;
        for (int i = 0; i < expoente; i++) {
            resultado *= base;  // Multiplicando a base pelo número de
vezes do expoente
        }
        return resultado;
    }
}
```

Explicação: O método potencia() calcula a exponenciação usando um loop for, onde a base é multiplicada pelo número de vezes correspondente ao valor do expoente. Esse valor é então retornado e exibido no main(). Esse exercício mostra como algoritmos mais complexos podem ser encapsulados em métodos, retornando valores que podem ser utilizados em outras partes do programa.

Exercício 115: Conversão de Temperatura

Enunciado: Desenvolva um método chamado converterCelsiusParaFahrenheit() que receba uma temperatura em graus Celsius e retorne o valor em Fahrenheit. No método main(), exiba a temperatura convertida.

Solução:

```java
public class ConversaoTemperatura {
    public static void main(String[] args) {
        double fahrenheit = converterCelsiusParaFahrenheit(25.0);  // Chamada do método com retorno
        System.out.println("A temperatura em Fahrenheit é: " + fahrenheit);
    }

    // Declaração do método converterCelsiusParaFahrenheit
    public static double converterCelsiusParaFahrenheit(double celsius) {
        return (celsius * 9/5) + 32;  // Fórmula de conversão de Celsius para Fahrenheit
    }
}
```

Explicação: O método converterCelsiusParaFahrenheit() recebe a temperatura em Celsius e aplica a fórmula de conversão para Fahrenheit. O resultado é retornado para o main(), onde é exibido. Esse exercício demonstra como encapsular cálculos específicos dentro de métodos reutilizáveis, permitindo que a lógica de conversão seja facilmente utilizada em diferentes partes de um programa.

Exercício 116: Verificar Maior Número

Enunciado: Crie um método chamado obterMaior() que receba dois números inteiros e retorne o maior deles. No método main(), capture o valor retornado e exiba o maior número.

Solução:

```java
public class MaiorNumero {
    public static void main(String[] args) {
        int maior = obterMaior(10, 20);  // Chamada do método obterMaior
        System.out.println("O maior número é: " + maior);
    }

    // Declaração do método obterMaior
    public static int obterMaior(int num1, int num2) {
        if (num1 > num2) {
            return num1;
        } else {
            return num2;
        }
```

```
    }
}
```

Explicação: O método obterMaior() recebe dois números inteiros e compara qual deles é maior, retornando o valor correspondente. A comparação é feita usando um condicional if-else, garantindo que o maior número seja retornado. Esse exercício reforça o conceito de métodos que podem retornar diferentes resultados dependendo das condições lógicas. Ele é útil em cenários onde precisamos comparar valores de forma eficiente e retornar um resultado apropriado.

Exercício 117: Média Aritmética com Retorno

Enunciado: Escreva um método chamado calcularMedia() que receba três números inteiros e retorne a média aritmética deles. No método main(), exiba a média retornada.

Solução:

```
public class MediaAritmetica {
    public static void main(String[] args) {
        double media = calcularMedia(10, 20, 30);  // Chamada do método calcularMedia
        System.out.println("A média é: " + media);
    }

    // Declaração do método calcularMedia
    public static double calcularMedia(int num1, int num2, int num3) {
        return (num1 + num2 + num3) / 3.0;  // Calculando e retornando a média
    }
}
```

Explicação: O método calcularMedia() recebe três números inteiros e calcula a média aritmética, retornando o resultado. Neste caso, a divisão é feita por 3.0 (um número de ponto flutuante) para garantir que o resultado seja um valor double, preservando casas decimais. Esse exercício mostra como métodos podem ser usados para cálculos mais complexos, encapsulando a lógica de uma operação e retornando o valor processado.

Exercício 118: Verificar Letra Maiúscula

Enunciado: Desenvolva um método chamado ehMaiuscula() que receba um caractere e retorne true se for uma letra maiúscula, e false se for uma letra minúscula. No método main(), utilize o valor retornado para exibir a mensagem correspondente.

Solução:

```
public class VerificarMaiuscula {
    public static void main(String[] args) {
        boolean resultado = ehMaiuscula('A');  // Chamada do método
ehMaiuscula
        if (resultado) {
            System.out.println("O caractere é maiúsculo.");
        } else {
            System.out.println("O caractere é minúsculo.");
        }
    }

    // Declaração do método ehMaiuscula
    public static boolean ehMaiuscula(char caractere) {
        return Character.isUpperCase(caractere);  // Verifica se o
caractere é maiúsculo
    }
}
```

Explicação: O método ehMaiuscula() usa o método da classe Character.isUpperCase() para verificar se um caractere é maiúsculo. Ele retorna true se o caractere for maiúsculo e false se for minúsculo. O retorno de valores booleanos simplifica a lógica de verificação, separando a lógica de checagem e permitindo que o main() manipule o resultado. Esse exercício reforça o uso de métodos nativos da linguagem para resolver problemas específicos.

Exercício 119: Raiz Quadrada com Retorno

Enunciado: Implemente um método chamado calcularRaizQuadrada() que receba um número inteiro e retorne a raiz quadrada desse número. No método main(), capture e exiba o valor retornado.

Solução:

```
public class RaizQuadrada {
    public static void main(String[] args) {
        double raiz = calcularRaizQuadrada(16);  // Chamada do método
calcularRaizQuadrada
        System.out.println("A raiz quadrada é: " + raiz);
    }
```

```
    // Declaração do método calcularRaizQuadrada
    public static double calcularRaizQuadrada(int numero) {
        return Math.sqrt(numero);  // Calculando e retornando a raiz
quadrada
    }
}
```

Explicação: O método calcularRaizQuadrada() usa o método Math.sqrt() da biblioteca padrão do Java para calcular a raiz quadrada de um número. O resultado é retornado para o main(), que o exibe. Este exercício demonstra como métodos de bibliotecas nativas podem ser integrados em funções personalizadas para realizar cálculos mais complexos, como a raiz quadrada, que seria difícil de implementar manualmente.

Exercício 120: Calcular Fatorial

Enunciado: Crie um método chamado calcularFatorial() que receba um número inteiro e retorne o fatorial desse número. No método main(), exiba o valor retornado.

Solução:

```
public class CalculoFatorial {
    public static void main(String[] args) {
        int fatorial = calcularFatorial(5);  // Chamada do método
calcularFatorial
        System.out.println("O fatorial é: " + fatorial);
    }

    // Declaração do método calcularFatorial
    public static int calcularFatorial(int numero) {
        int resultado = 1;
        for (int i = 1; i <= numero; i++) {
            resultado *= i;  // Multiplicando os valores sucessivos
        }
        return resultado;  // Retornando o fatorial
    }
}
```

Explicação: O método calcularFatorial() calcula o fatorial de um número usando um loop for. Ele multiplica os números sucessivos de 1 até o número fornecido e retorna o resultado. O fatorial é uma operação matemática que multiplica um número por todos os números positivos menores que ele. O uso de loops dentro de métodos permite realizar cálculos iterativos de forma eficiente, encapsulando a lógica em uma função que pode ser reutilizada em diferentes contextos.

Exercício 121: Verificar Número Primo

Enunciado: Desenvolva um método chamado ehPrimo() que receba um número inteiro e retorne true se o número for primo, e false caso contrário. No método main(), utilize o valor retornado para exibir a mensagem correspondente.

Solução:

```
public class VerificarPrimo {
    public static void main(String[] args) {
        boolean resultado = ehPrimo(7);  // Chamada do método ehPrimo
        if (resultado) {
            System.out.println("O número é primo.");
        } else {
            System.out.println("O número não é primo.");
        }
    }

    // Declaração do método ehPrimo
    public static boolean ehPrimo(int numero) {
        if (numero < 2) {
            return false;  // Números menores que 2 não são primos
        }
        for (int i = 2; i <= Math.sqrt(numero); i++) {
            if (numero % i == 0) {
                return false;  // Se divisível por qualquer número além
de 1 e ele mesmo
            }
        }
        return true;  // Se não encontrou nenhum divisor, o número é
primo
    }
}
```

Explicação: O método ehPrimo() verifica se um número é primo, retornando true se for e false caso contrário. Um número primo é aquele que só é divisível por 1 e por ele mesmo. Para otimizar a verificação, o laço percorre os números de 2 até a raiz quadrada do número, pois se não houver divisores até essa raiz, o número não terá divisores maiores. Este tipo de verificação é útil em diversas áreas da matemática e programação.

Exercício 122: Contagem de Vogais

Enunciado: Implemente um método chamado contarVogais() que receba uma string como parâmetro e retorne a quantidade de vogais presentes na string. No método main(), exiba o valor retornado.

Solução:

```
public class ContagemVogais {
    public static void main(String[] args) {
        int totalVogais = contarVogais("Programacao");  // Chamada do
método contarVogais
        System.out.println("O total de vogais é: " + totalVogais);
    }

    // Declaração do método contarVogais
    public static int contarVogais(String texto) {
        int contagem = 0;
        String vogais = "aeiouAEIOU";  // Definindo as vogais
        for (int i = 0; i < texto.length(); i++) {
            if (vogais.indexOf(texto.charAt(i)) != -1) {  // Verifica se
o caractere é uma vogal
                contagem++;
            }
        }
        return contagem;  // Retorna o número total de vogais
    }
}
```

Explicação: O método contarVogais() percorre uma string e conta quantas vogais ela contém. Para isso, ele verifica se cada caractere pertence à string "aeiouAEIOU", que contém todas as vogais possíveis (minúsculas e maiúsculas). O método retorna a quantidade de vogais encontradas. Esse tipo de função é útil para a análise de textos e processamento de strings.

Exercício 123: Conversão de Moedas

Enunciado: Crie um método chamado converterRealParaDolar() que receba um valor em reais e retorne o valor convertido em dólares. No método main(), exiba o valor convertido. Considere a taxa de câmbio fixa de 1 dólar = 5,00 reais.

Solução:

```java
public class ConversaoMoeda {
    public static void main(String[] args) {
        double valorEmDolar = converterRealParaDolar(50.0);  // Chamada do método converterRealParaDolar
        System.out.println("O valor em dólares é: " + valorEmDolar);
    }

    // Declaração do método converterRealParaDolar
    public static double converterRealParaDolar(double valorReal) {
        double taxaCambio = 5.0;  // Taxa de câmbio fixa
        return valorReal / taxaCambio;  // Retorna o valor convertido para dólares
    }
}
```

Explicação: O método converterRealParaDolar() recebe um valor em reais, aplica a taxa de câmbio de 5 reais para 1 dólar e retorna o valor convertido em dólares. Essa função é simples, mas muito útil em aplicações que lidam com conversões de moeda, onde o valor de entrada em uma moeda precisa ser transformado em outra. A modularização desse tipo de lógica torna o código reutilizável e fácil de atualizar caso a taxa de câmbio mude.

Exercício 124: Menor de Três Números

Enunciado: Desenvolva um método chamado obterMenor() que receba três números inteiros e retorne o menor deles. No método main(), capture o valor retornado e exiba o menor número.

Solução:

```java
public class MenorDeTres {
    public static void main(String[] args) {
        int menor = obterMenor(15, 7, 10);  // Chamada do método obterMenor
        System.out.println("O menor número é: " + menor);
    }

    // Declaração do método obterMenor
    public static int obterMenor(int num1, int num2, int num3) {
        int menor = num1;
        if (num2 < menor) {
            menor = num2;
        }
```

```
        if (num3 < menor) {
            menor = num3;
        }
        return menor;  // Retorna o menor número
    }
}
```

Explicação: O método obterMenor() compara três números e retorna o menor deles. Isso é feito inicializando a variável menor com o primeiro número e, em seguida, comparando com os outros dois. Esse tipo de função é útil em problemas de seleção, onde é necessário determinar o menor (ou maior) valor entre um conjunto de números. A lógica é simples, mas pode ser aplicada em muitos contextos de programação.

Sobrecarga de métodos

Exercício 125: Sobrecarga do Método Soma

Enunciado: Implemente uma classe com dois métodos chamados somar(). O primeiro método deve receber dois números inteiros como parâmetros e retornar a soma deles. O segundo método deve receber três números inteiros e retornar a soma dos três. No método main(), teste os dois métodos chamando-os com diferentes quantidades de parâmetros.

Solução:

```
public class SobrecargaSoma {
    public static void main(String[] args) {
        System.out.println("Soma de dois números: " + somar(5, 10));
// Chamando o método com 2 parâmetros
        System.out.println("Soma de três números: " + somar(3, 7, 2));
// Chamando o método com 3 parâmetros
    }

    // Sobrecarga do método somar para dois parâmetros
    public static int somar(int num1, int num2) {
        return num1 + num2;  // Retorna a soma de dois números
    }

    // Sobrecarga do método somar para três parâmetros
    public static int somar(int num1, int num2, int num3) {
        return num1 + num2 + num3;  // Retorna a soma de três números
    }
```

```
}
```

Explicação: Neste exercício, a sobrecarga de métodos permite criar duas versões do método somar() com a mesma assinatura, mas com diferentes números de parâmetros. O primeiro método recebe dois parâmetros e retorna a soma deles, enquanto o segundo método recebe três parâmetros e retorna a soma dos três. Ao sobrecarregar métodos, o Java determina qual método chamar com base no número de parâmetros fornecidos durante a chamada. Isso oferece flexibilidade ao programador, permitindo que diferentes operações sejam realizadas com métodos de mesmo nome, mas diferentes entradas.

Exercício 126: Sobrecarga do Método Calcular Área

Enunciado: Desenvolva uma classe que contenha dois métodos sobrecarregados chamados calcularArea(). O primeiro método deve calcular e retornar a área de um quadrado, recebendo o comprimento de um dos lados como parâmetro. O segundo método deve calcular e retornar a área de um retângulo, recebendo a largura e a altura como parâmetros. No método main(), teste os dois métodos.

Solução:

```
public class SobrecargaArea {
    public static void main(String[] args) {
        System.out.println("Área do quadrado: " + calcularArea(5));
// Chamando o método com 1 parâmetro
        System.out.println("Área do retângulo: " + calcularArea(5, 10));
// Chamando o método com 2 parâmetros
    }

    // Sobrecarga do método calcularArea para o quadrado (1 parâmetro)
    public static int calcularArea(int lado) {
        return lado * lado;  // Calcula e retorna a área do quadrado
    }

    // Sobrecarga do método calcularArea para o retângulo (2 parâmetros)
    public static int calcularArea(int largura, int altura) {
        return largura * altura;  // Calcula e retorna a área do
retângulo
    }
}
```

Explicação: A sobrecarga de métodos permite que o método calcularArea() tenha duas versões: uma que calcula a área de um quadrado (com base no comprimento de um lado) e outra que calcula a área de um retângulo (com base na largura e altura). Ao invocar o

método calcularArea(), o compilador diferencia qual método chamar com base na quantidade de argumentos fornecidos. Isso é útil quando se deseja criar métodos genéricos para cálculos relacionados, mas com diferentes entradas, como a área de diferentes formas geométricas.

Exercício 127: Sobrecarga do Método Exibir Mensagem

Enunciado: Crie uma classe com dois métodos sobrecarregados chamados exibirMensagem(). O primeiro método deve exibir uma mensagem padrão no console. O segundo método deve receber uma string como parâmetro e exibir essa string no console. No método main(), chame ambos os métodos para exibir as mensagens.

Solução:

```
public class SobrecargaMensagem {
    public static void main(String[] args) {
        exibirMensagem();               // Chamando o método sem
parâmetros
        exibirMensagem("Bem-vindo!");   // Chamando o método com uma
string como parâmetro
    }

    // Sobrecarga do método exibirMensagem sem parâmetros
    public static void exibirMensagem() {
        System.out.println("Mensagem padrão: Olá, mundo!");  // Exibe a
mensagem padrão
    }

    // Sobrecarga do método exibirMensagem com um parâmetro
    public static void exibirMensagem(String mensagem) {
        System.out.println("Mensagem personalizada: " + mensagem);  //
Exibe a mensagem recebida como parâmetro
    }
}
```

Explicação: A sobrecarga dos métodos exibirMensagem() permite que o mesmo método exiba diferentes tipos de mensagens, dependendo da forma como ele é chamado. Quando o método é chamado sem parâmetros, ele exibe uma mensagem padrão. Quando é chamado com uma string como parâmetro, exibe essa mensagem personalizada. A sobrecarga de métodos é uma forma poderosa de lidar com diferentes entradas em funções que realizam tarefas semelhantes, tornando o código mais flexível e reutilizável.

Conclusão do Capítulo 6: Métodos e Funções

Neste capítulo, exploramos os principais conceitos de métodos e funções em Java, passando pela criação, uso e sobrecarga de métodos.

Os exercícios proporcionaram prática na declaração e chamada de métodos, mostrando como modularizar o código, tornando-o mais organizado e reutilizável. Na seção de passagem de parâmetros e retorno de valores, aprendemos a enviar informações para os métodos e receber respostas, o que possibilita realizar cálculos e manipulações eficientes.

Por fim, na sobrecarga de métodos, vimos como utilizar o mesmo nome de método com diferentes assinaturas para lidar com diferentes tipos de operações.

Introdução ao Capítulo 7: Classes e Objetos

Neste capítulo, vamos explorar um dos pilares fundamentais da programação orientada a objetos (POO): a criação e manipulação de classes e objetos. Através de exemplos práticos, você aprenderá a definir classes, que são os modelos para criar objetos, e a compreender como os atributos e métodos de instância moldam o comportamento desses objetos no programa.

Você também verá como o encapsulamento, um dos conceitos mais importantes da POO, permite proteger os dados de uma classe, controlando seu acesso e modificação por meio de getters e setters. O encapsulamento promove segurança e flexibilidade, assegurando que os atributos sejam manipulados de maneira adequada.

Definição de Classes e Objetos

Exercício 128: Definindo uma Classe Simples

Enunciado: Crie uma classe chamada Carro. Defina três atributos para a classe: marca, modelo e ano. Em seguida, crie um objeto dessa classe no método main() e inicialize os atributos com valores.

Solução:

```
public class Carro {
    String marca;
    String modelo;
    int ano;
```

```
    public static void main(String[] args) {
        Carro meuCarro = new Carro();  // Criando o objeto Carro
        meuCarro.marca = "Toyota";
        meuCarro.modelo = "Corolla";
        meuCarro.ano = 2020;

        // Exibindo os valores dos atributos
        System.out.println("Marca: " + meuCarro.marca);
        System.out.println("Modelo: " + meuCarro.modelo);
        System.out.println("Ano: " + meuCarro.ano);
    }
}
```

Explicação: Aqui, criamos uma classe chamada Carro com três atributos: marca, modelo, e ano. Esses atributos são variáveis de instância que representam as características do carro. No método main(), criamos um objeto da classe Carro e inicializamos seus atributos diretamente. O conceito fundamental deste exercício é a instanciação de objetos e a atribuição de valores a atributos. A classe atua como um molde que descreve o que é um carro, e o objeto meuCarro é uma instância específica desse molde.

Exercício 129: Definindo Atributos e Métodos

Enunciado: Crie uma classe chamada Pessoa com os atributos nome e idade. Adicione um método chamado apresentar() que exiba no console: "Olá, meu nome é [nome] e tenho [idade] anos". No método main(), crie dois objetos Pessoa e chame o método apresentar() para ambos.

Solução:

```
public class Pessoa {
    String nome;
    int idade;

    // Método para apresentação
    public void apresentar() {
        System.out.println("Olá, meu nome é " + nome + " e tenho " +
idade + " anos.");
    }

    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa();
        pessoa1.nome = "Carlos";
```

```
        pessoa1.idade = 30;

        Pessoa pessoa2 = new Pessoa();
        pessoa2.nome = "Ana";
        pessoa2.idade = 25;

        pessoa1.apresentar();  // Chamada do método apresentar
        pessoa2.apresentar();  // Chamada do método apresentar
    }
}
```

Explicação: Neste exercício, a classe Pessoa tem dois atributos: nome e idade. Adicionamos o método apresentar(), que exibe uma mensagem personalizada. O método usa os valores dos atributos de instância para exibir uma mensagem de apresentação. No método main(), criamos duas instâncias da classe Pessoa e atribuímos diferentes valores a nome e idade. O foco deste exercício é mostrar como definir métodos que usam atributos de instância e como esses métodos podem ser chamados em diferentes objetos, permitindo reutilização de código.

Exercício 130: Definindo uma Classe com Construtor

Enunciado: Implemente uma classe chamada ContaBancaria com os atributos numeroConta e saldo. Crie um construtor que inicialize esses atributos ao criar um objeto. No método main(), crie uma conta bancária usando o construtor.

Solução:

```
public class ContaBancaria {
    int numeroConta;
    double saldo;

    // Construtor para inicializar os atributos
    public ContaBancaria(int numeroConta, double saldoInicial) {
        this.numeroConta = numeroConta;
        this.saldo = saldoInicial;
    }

    public static void main(String[] args) {
        ContaBancaria minhaConta = new ContaBancaria(12345, 500.0);  //
Criando uma conta bancária
        System.out.println("Número da conta: " +
minhaConta.numeroConta);
        System.out.println("Saldo inicial: " + minhaConta.saldo);
    }
```

```
}
```

Explicação: Neste exercício, introduzimos o conceito de construtores. O construtor é um método especial que é chamado automaticamente quando um objeto é criado. Ele é utilizado para inicializar os atributos da classe no momento da criação do objeto. A classe ContaBancaria possui um construtor que recebe os valores de numeroConta e saldoInicial, que são usados para inicializar os atributos do objeto. Esse exercício mostra como os construtores podem simplificar a criação de objetos, garantindo que os atributos sempre tenham valores definidos logo após a criação do objeto.

Exercício 131: Métodos para Manipular Dados

Enunciado: Adicione à classe ContaBancaria os métodos depositar(double valor) e sacar(double valor), que atualizam o saldo da conta. No método main(), simule um depósito e um saque, e exiba o saldo após cada operação.

Solução:

```
public class ContaBancaria {
    int numeroConta;
    double saldo;

    public ContaBancaria(int numeroConta, double saldoInicial) {
        this.numeroConta = numeroConta;
        this.saldo = saldoInicial;
    }

    // Método para depósito
    public void depositar(double valor) {
        saldo += valor;
    }

    // Método para saque
    public void sacar(double valor) {
        if (valor <= saldo) {
            saldo -= valor;
        } else {
            System.out.println("Saldo insuficiente.");
        }
    }

    public static void main(String[] args) {
        ContaBancaria minhaConta = new ContaBancaria(12345, 500.0);
```

```
        // Simulando operações de depósito e saque
        minhaConta.depositar(200.0);
        System.out.println("Saldo após depósito: " + minhaConta.saldo);

        minhaConta.sacar(100.0);
        System.out.println("Saldo após saque: " + minhaConta.saldo);
    }
}
```

Explicação: Este exercício expande a classe ContaBancaria, adicionando métodos para modificar o valor do saldo. O método depositar() aumenta o saldo com base no valor fornecido, e o método sacar() reduz o saldo, mas somente se o valor a ser sacado não exceder o saldo disponível. Este exercício ensina a manipulação de dados em objetos e como podemos definir comportamentos específicos para classes através de métodos. Ele também aborda a validação dentro dos métodos, garantindo que uma ação só seja realizada se determinadas condições forem atendidas.

Exercício 132: Classe Produto

Enunciado: Crie uma classe chamada Produto com os atributos nome, preco e quantidade. Adicione um método para calcular o valor total do estoque (preco * quantidade) e outro método para exibir os detalhes do produto. No método main(), crie dois objetos e teste os métodos.

Solução:

```
public class Produto {
    String nome;
    double preco;
    int quantidade;

    // Método para calcular o valor total do estoque
    public double calcularValorEstoque() {
        return preco * quantidade;
    }

    // Método para exibir detalhes do produto
    public void exibirDetalhes() {
        System.out.println("Produto: " + nome);
        System.out.println("Preço: R$ " + preco);
        System.out.println("Quantidade em estoque: " + quantidade);
        System.out.println("Valor total em estoque: R$ " +
calcularValorEstoque());
    }
```

```
    public static void main(String[] args) {
        Produto produto1 = new Produto();
        produto1.nome = "Notebook";
        produto1.preco = 3000.0;
        produto1.quantidade = 10;

        Produto produto2 = new Produto();
        produto2.nome = "Mouse";
        produto2.preco = 50.0;
        produto2.quantidade = 100;

        produto1.exibirDetalhes();  // Exibindo detalhes do produto 1
        produto2.exibirDetalhes();  // Exibindo detalhes do produto 2
    }
}
```

Explicação: A classe Produto contém três atributos e dois métodos. O método calcularValorEstoque() realiza o cálculo do valor total do estoque multiplicando o preço pela quantidade. O método exibirDetalhes() exibe informações detalhadas sobre o produto, incluindo o valor total do estoque. Esse exercício é útil para entender como organizar a lógica relacionada a um objeto em diferentes métodos, o que facilita o entendimento e a reutilização do código. Além disso, a manipulação de dados através de métodos torna o código mais flexível e fácil de manter.

Exercício 133: Relacionamento entre Classes

Enunciado: Crie duas classes: Aluno e Turma. A classe Turma deve ter o atributo nome e um método para exibir o nome da turma. A classe Aluno deve ter os atributos nome e turma, sendo turma do tipo Turma. No método main(), crie objetos das duas classes e exiba as informações.

Solução:

```
class Turma {
    String nome;

    // Método para exibir o nome da turma
    public void exibirTurma() {
        System.out.println("Turma: " + nome);
    }
}

class Aluno {
```

```
    String nome;
    Turma turma;

    // Método para exibir os dados do aluno
    public void exibirDados() {
        System.out.println("Aluno: " + nome);
        turma.exibirTurma();  // Chama o método da classe Turma
    }
}

public class RelacionamentoClasses {
    public static void main(String[] args) {
        Turma turma1 = new Turma();
        turma1.nome = "3A";

        Aluno aluno1 = new Aluno();
        aluno1.nome = "João";
        aluno1.turma = turma1;  // Relacionando o aluno à turma

        aluno1.exibirDados();  // Exibindo os dados do aluno e da turma
    }
}
```

Explicação: Aqui estamos introduzindo o conceito de relacionamento entre classes. A classe Aluno tem um atributo turma, que é do tipo Turma. Isso demonstra como uma classe pode ser composta por outra. No método exibirDados() da classe Aluno, chamamos o método exibirTurma() da classe Turma. Este exercício ensina a composição de objetos, um conceito central em programação orientada a objetos, onde um objeto pode ser composto por outros objetos.

Exercício 134: Classe Retângulo

Enunciado: Implemente uma classe Retangulo com os atributos largura e altura. Adicione métodos para calcular a área e o perímetro do retângulo. No método main(), crie um objeto Retangulo e exiba seus valores de área e perímetro.

Solução:

```
public class Retangulo {
    double largura;
    double altura;

    // Método para calcular a área do retângulo
```

```
    public double calcularArea() {
        return largura * altura;
    }

    // Método para calcular o perímetro do retângulo
    public double calcularPerimetro() {
        return 2 * (largura + altura);
    }

    public static void main(String[] args) {
        Retangulo retangulo = new Retangulo();
        retangulo.largura = 5.0;
        retangulo.altura = 3.0;

        System.out.println("Área: " + retangulo.calcularArea());
        System.out.println("Perímetro: " +
retangulo.calcularPerimetro());
    }
}
```

Explicação: Este exercício foca na criação de uma classe com métodos que realizam cálculos com base nos atributos. A classe Retangulo possui métodos para calcular a área e o perímetro. Esses métodos são chamados no main() para exibir os resultados. O objetivo deste exercício é mostrar como separar a lógica de cálculo em métodos específicos, o que torna o código mais organizado e fácil de manter.

Exercício 135: Definindo Métodos com Retorno

Enunciado: Crie uma classe Cilindro com os atributos raio e altura. Adicione um método calcularVolume() que retorne o volume do cilindro (π * raio² * altura). No método main(), crie um objeto Cilindro e exiba o volume.

Solução:

```
public class Cilindro {
    double raio;
    double altura;

    // Método para calcular o volume do cilindro
    public double calcularVolume() {
        return Math.PI * Math.pow(raio, 2) * altura;
    }

    public static void main(String[] args) {
```

```
        Cilindro cilindro = new Cilindro();
        cilindro.raio = 3.0;
        cilindro.altura = 10.0;

        System.out.println("Volume do cilindro: " +
cilindro.calcularVolume());
    }
}
```

Explicação: Aqui, introduzimos a matemática em métodos com o uso da biblioteca Math do Java para calcular o volume de um cilindro. O método calcularVolume() usa a fórmula do volume do cilindro e retorna o resultado. Este exercício reforça o uso de métodos com retorno de valores, permitindo que o cálculo seja feito dentro do método e o valor seja retornado para ser usado no main().

Exercício 136: Classe Aluno com Média

Enunciado: Implemente uma classe Aluno com os atributos nome, nota1 e nota2. Adicione um método calcularMedia() que calcule e retorne a média das duas notas. No método main(), crie um aluno e exiba sua média.

Solução:

```
public class Aluno {
    String nome;
    double nota1;
    double nota2;

    // Método para calcular a média das notas
    public double calcularMedia() {
        return (nota1 + nota2) / 2;
    }

    public static void main(String[] args) {
        Aluno aluno = new Aluno();
        aluno.nome = "Carlos";
        aluno.nota1 = 8.0;
        aluno.nota2 = 9.5;

        System.out.println("Média do aluno " + aluno.nome + ": " +
aluno.calcularMedia());
    }
}
```

Explicação: Neste exercício, criamos uma classe Aluno com atributos que representam as notas do aluno. O método calcularMedia() retorna a média das duas notas. Esse exercício foca na criação de métodos que retornam valores, aplicando uma lógica simples de cálculo. A reutilização do método para diferentes alunos demonstra a flexibilidade da POO para trabalhar com dados dinâmicos.

Exercício 137: Classe com Métodos Estáticos

Enunciado: Crie uma classe Calculadora com métodos estáticos somar(), subtrair(), multiplicar(), dividir(), que realizam as operações matemáticas básicas. No método main(), chame esses métodos sem criar um objeto.

Solução:

```
public class Calculadora {

    // Método estático para soma
    public static double somar(double a, double b) {
        return a + b;
    }

    // Método estático para subtração
    public static double subtrair(double a, double b) {
        return a - b;
    }

    // Método estático para multiplicação
    public static double multiplicar(double a, double b) {
        return a * b;
    }

    // Método estático para divisão
    public static double dividir(double a, double b) {
        if (b != 0) {
            return a / b;
        } else {
            System.out.println("Erro: Divisão por zero.");
            return 0;
        }
    }

    public static void main(String[] args) {
        System.out.println("Soma: " + Calculadora.somar(10, 5));
        System.out.println("Subtração: " + Calculadora.subtrair(10, 5));
        System.out.println("Multiplicação: " +
```

```
Calculadora.multiplicar(10, 5));
        System.out.println("Divisão: " + Calculadora.dividir(10, 5));
    }
}
```

Explicação: Neste exercício, introduzimos o conceito de métodos estáticos. Métodos estáticos pertencem à classe em si, e não a instâncias da classe, o que significa que podem ser chamados sem a criação de um objeto. A classe Calculadora contém métodos para realizar operações matemáticas básicas, que podem ser invocados diretamente no main() sem precisar criar um objeto Calculadora. Esse exercício ensina a diferença entre métodos de instância e métodos estáticos, mostrando como métodos estáticos podem ser úteis para funções gerais.

Exercício 138: Classe Livro

Enunciado: Implemente uma classe Livro com os atributos titulo, autor e numeroPaginas. Adicione um método exibirDetalhes() que exiba as informações do livro. No método main(), crie dois livros e exiba seus detalhes.

Solução:

```
public class Livro {
    String titulo;
    String autor;
    int numeroPaginas;

    // Método para exibir os detalhes do livro
    public void exibirDetalhes() {
        System.out.println("Título: " + titulo);
        System.out.println("Autor: " + autor);
        System.out.println("Número de páginas: " + numeroPaginas);
        System.out.println();
    }

    public static void main(String[] args) {
        Livro livro1 = new Livro();
        livro1.titulo = "Dom Quixote";
        livro1.autor = "Miguel de Cervantes";
        livro1.numeroPaginas = 992;

        Livro livro2 = new Livro();
        livro2.titulo = "1984";
        livro2.autor = "George Orwell";
```

```
        livro2.numeroPaginas = 328;

        livro1.exibirDetalhes();  // Exibindo detalhes do primeiro livro
        livro2.exibirDetalhes();  // Exibindo detalhes do segundo livro
    }
}
```

Explicação: Neste exercício, a classe Livro contém atributos que armazenam detalhes sobre um livro, como o título, o autor e o número de páginas. O método exibirDetalhes() exibe essas informações. Ao criar dois objetos da classe Livro e chamar o método exibirDetalhes(), conseguimos mostrar como diferentes objetos podem ter seus próprios dados e métodos para manipular esses dados. Este exercício reforça a criação de objetos com atributos distintos e o uso de métodos para exibir suas propriedades.

Exercício 139: Classe Funcionario

Enunciado: Crie uma classe Funcionario com os atributos nome, salario e cargo. Adicione um método aumentarSalario(double porcentagem) que aumenta o salário de acordo com a porcentagem dada. No método main(), crie um funcionário, aumente seu salário e exiba as informações.

Solução:

```
public class Funcionario {
    String nome;
    double salario;
    String cargo;

    // Método para aumentar o salário de acordo com a porcentagem
    public void aumentarSalario(double porcentagem) {
        salario += salario * (porcentagem / 100);
    }

    // Método para exibir os detalhes do funcionário
    public void exibirDetalhes() {
        System.out.println("Nome: " + nome);
        System.out.println("Cargo: " + cargo);
        System.out.println("Salário: R$ " + salario);
    }

    public static void main(String[] args) {
        Funcionario funcionario = new Funcionario();
        funcionario.nome = "João";
        funcionario.salario = 3000.0;
```

```
        funcionario.cargo = "Analista de Sistemas";

        // Aumentando o salário em 10%
        funcionario.aumentarSalario(10);

        funcionario.exibirDetalhes();  // Exibindo as informações do
funcionário
    }
}
```

Explicação: A classe Funcionario tem três atributos (nome, salario e cargo) e um método aumentarSalario(), que ajusta o salário com base em uma porcentagem fornecida. Este exercício introduz o conceito de métodos que alteram o estado interno de um objeto, mostrando como as variáveis de instância podem ser modificadas por métodos da própria classe. Ao aumentar o salário e exibir os detalhes do funcionário, vemos como os métodos podem trabalhar juntos para modificar e exibir dados de objetos.

Exercício 140: Classe Fatura

Enunciado: Implemente uma classe Fatura com os atributos numeroItem, descricao, quantidade, precoPorItem. Adicione um método calcularTotal() que retorne o valor total da fatura (quantidade * precoPorItem). No método main(), crie uma fatura e exiba o valor total.

Solução:

```
public class Fatura {
    String numeroItem;
    String descricao;
    int quantidade;
    double precoPorItem;

    // Método para calcular o valor total da fatura
    public double calcularTotal() {
        return quantidade * precoPorItem;
    }

    public static void main(String[] args) {
        Fatura fatura = new Fatura();
        fatura.numeroItem = "001";
        fatura.descricao = "Notebook";
        fatura.quantidade = 2;
        fatura.precoPorItem = 2500.0;
```

```
        double total = fatura.calcularTotal();
        System.out.println("Valor total da fatura: R$ " + total);
    }
}
```

Explicação: A classe Fatura tem atributos que descrevem um item de uma fatura. O método calcularTotal() retorna o valor total da fatura, multiplicando a quantidade pelo preço por item. Este exercício demonstra a importância de encapsular a lógica de cálculo dentro de métodos da classe, permitindo que o cálculo do valor total seja feito de maneira isolada e reutilizável. Essa prática ajuda a manter o código organizado e modular.

Exercício 141: Classe ContaCorrente

Enunciado: Crie uma classe ContaCorrente com os atributos numeroConta, saldo e limite. Adicione um método verificarLimite() que exiba se o saldo está dentro do limite. No método main(), crie uma conta e teste o método.

Solução:

```
public class ContaCorrente {
    int numeroConta;
    double saldo;
    double limite;

    // Método para verificar se o saldo está dentro do limite
    public void verificarLimite() {
        if (saldo > limite) {
            System.out.println("Saldo acima do limite.");
        } else {
            System.out.println("Saldo dentro do limite.");
        }
    }

    public static void main(String[] args) {
        ContaCorrente conta = new ContaCorrente();
        conta.numeroConta = 123456;
        conta.saldo = 2000.0;
        conta.limite = 3000.0;

        conta.verificarLimite();  // Verificando se o saldo está dentro
do limite
    }
}
```

Explicação: Este exercício introduz a classe ContaCorrente, que possui atributos relacionados à conta bancária e um método verificarLimite(). O método compara o saldo com o limite da conta e exibe uma mensagem apropriada. O foco aqui é a validação de dados dentro de métodos, onde é feita uma verificação condicional para determinar o comportamento correto.

Exercício 142: Classe Filme

Enunciado: Implemente uma classe Filme com os atributos titulo, diretor e duracao. Adicione um método exibirInformacoes() que exiba as informações do filme. No método main(), crie dois filmes e exiba seus detalhes.

Solução:

```
public class Filme {
    String titulo;
    String diretor;
    int duracao;  // duração em minutos

    // Método para exibir as informações do filme
    public void exibirInformacoes() {
        System.out.println("Título: " + titulo);
        System.out.println("Diretor: " + diretor);
        System.out.println("Duração: " + duracao + " minutos");
        System.out.println();
    }

    public static void main(String[] args) {
        Filme filme1 = new Filme();
        filme1.titulo = "O Poderoso Chefão";
        filme1.diretor = "Francis Ford Coppola";
        filme1.duracao = 175;

        Filme filme2 = new Filme();
        filme2.titulo = "Pulp Fiction";
        filme2.diretor = "Quentin Tarantino";
        filme2.duracao = 154;

        filme1.exibirInformacoes();  // Exibindo detalhes do primeiro
filme
        filme2.exibirInformacoes();  // Exibindo detalhes do segundo
filme
    }
}
```

Explicação: A classe Filme tem atributos que descrevem as características de um filme, como o título, o diretor e a duração. O método exibirInformacoes() exibe essas características no console. Este exercício ensina a manipulação de dados relacionados a objetos e a exibição organizada de informações usando métodos que processam e formatam os atributos do objeto.

Atributos e Métodos de Instância

Exercício 143: Classe Pessoa com Atributos

Enunciado: Crie uma classe chamada Pessoa com os atributos nome e idade. Adicione um método aniversario() que incremente a idade em 1. No método main(), crie uma pessoa, exiba a idade, chame o método aniversario() e exiba a nova idade.

Solução:

```
public class Pessoa {
    String nome;
    int idade;

    // Método para incrementar a idade
    public void aniversario() {
        idade++;
    }

    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa();
        pessoa.nome = "João";
        pessoa.idade = 25;

        // Exibindo a idade antes do aniversário
        System.out.println("Idade antes do aniversário: " +
pessoa.idade);

        // Chamando o método aniversario()
        pessoa.aniversario();

        // Exibindo a idade após o aniversário
        System.out.println("Idade após o aniversário: " + pessoa.idade);
    }
}
```

Explicação: Neste exercício, a classe Pessoa tem dois atributos: nome e idade. O método aniversario() é responsável por incrementar a idade em 1, simulando o efeito de um aniversário. A instância da classe é criada no método main(), onde o nome e a idade são definidos. O método aniversario() é chamado para modificar o valor da idade e, em seguida, o novo valor é exibido. Este exercício ensina como criar métodos que modificam os atributos de instância de um objeto e como esses métodos podem ser reutilizados ao longo da vida do objeto.

Exercício 144: Classe Produto com Desconto

Enunciado: Crie uma classe Produto com os atributos nome, preco e desconto. Adicione um método aplicarDesconto() que aplique o desconto ao preço do produto. No método main(), crie um produto, aplique o desconto e exiba o preço final.

Solução:

```
public class Produto {
    String nome;
    double preco;
    double desconto;

    // Método para aplicar o desconto ao preço do produto
    public void aplicarDesconto() {
        preco = preco - (preco * (desconto / 100));
    }

    public static void main(String[] args) {
        Produto produto = new Produto();
        produto.nome = "Celular";
        produto.preco = 2000.0;
        produto.desconto = 10.0;

        // Exibindo o preço antes do desconto
        System.out.println("Preço original: R$ " + produto.preco);

        // Aplicando o desconto
        produto.aplicarDesconto();

        // Exibindo o preço final após o desconto
        System.out.println("Preço com desconto: R$ " + produto.preco);
    }
}
```

Explicação: Neste exercício, a classe Produto tem os atributos nome, preco e desconto. O método aplicarDesconto() modifica o valor do preço aplicando a porcentagem de desconto

informada. No método main(), um produto é criado com um nome, preço e desconto. O preço original é exibido antes de aplicar o desconto, e o novo preço é exibido após a aplicação. Este exercício demonstra a capacidade de métodos de instância de alterar os atributos de um objeto com base em operações matemáticas e regras definidas pelo programador.

Exercício 145: Classe Carro com Consumo

Enunciado: Crie uma classe Carro com os atributos marca, modelo, consumoCombustivel (km por litro) e quantidadeCombustivel. Adicione um método dirigir() que reduza o combustível com base na distância percorrida e no consumo. No método main(), crie um carro, simule uma viagem e exiba o combustível restante.

Solução:

```
public class Carro {
    String marca;
    String modelo;
    double consumoCombustivel;  // Km por litro
    double quantidadeCombustivel;  // Em litros

    // Método para dirigir o carro, reduzindo o combustível
    public void dirigir(double distancia) {
        double combustivelNecessario = distancia / consumoCombustivel;
        if (combustivelNecessario <= quantidadeCombustivel) {
            quantidadeCombustivel -= combustivelNecessario;
            System.out.println("Viagem realizada. Combustível restante:
" + quantidadeCombustivel + " litros.");
        } else {
            System.out.println("Combustível insuficiente para realizar a
viagem.");
        }
    }

    public static void main(String[] args) {
        Carro carro = new Carro();
        carro.marca = "Honda";
        carro.modelo = "Civic";
        carro.consumoCombustivel = 12.0;  // 12 km por litro
        carro.quantidadeCombustivel = 50.0;  // 50 litros no tanque

        // Simulando uma viagem de 100 km
        carro.dirigir(100);

        // Exibindo o combustível restante
```

```
        System.out.println("Combustível restante no tanque: " +
carro.quantidadeCombustivel + " litros.");
    }
}
```

Explicação: Neste exercício, a classe Carro possui atributos que definem o consumo de combustível e a quantidade de combustível no tanque. O método dirigir() calcula quanto combustível é necessário para percorrer uma determinada distância e reduz o combustível restante, se possível. Caso contrário, informa que o combustível é insuficiente. Este exercício demonstra como os métodos de instância podem realizar cálculos e modificar os atributos do objeto com base em regras de negócios ou operações complexas.

Exercício 146: Classe ContaBancaria com Transferência

Enunciado: Crie uma classe ContaBancaria com os atributos saldo e numeroConta. Adicione um método transferir() que receba outra conta bancária e um valor a ser transferido, reduzindo o saldo da conta de origem e aumentando o saldo da conta destino. No método main(), crie duas contas e faça uma transferência.

Solução:

```
public class ContaBancaria {
    int numeroConta;
    double saldo;

    // Método para transferir dinheiro para outra conta
    public void transferir(ContaBancaria contaDestino, double valor) {
        if (saldo >= valor) {
            saldo -= valor;
            contaDestino.saldo += valor;
            System.out.println("Transferência de R$ " + valor + "
realizada com sucesso.");
        } else {
            System.out.println("Saldo insuficiente para a
transferência.");
        }
    }

    public static void main(String[] args) {
        ContaBancaria conta1 = new ContaBancaria();
        conta1.numeroConta = 12345;
        conta1.saldo = 1000.0;

        ContaBancaria conta2 = new ContaBancaria();
```

```
        conta2.numeroConta = 67890;
        conta2.saldo = 500.0;

        // Exibindo o saldo inicial das contas
        System.out.println("Saldo da Conta 1: R$ " + conta1.saldo);
        System.out.println("Saldo da Conta 2: R$ " + conta2.saldo);

        // Realizando uma transferência de R$ 200 da conta 1 para a
conta 2
        conta1.transferir(conta2, 200.0);

        // Exibindo o saldo final das contas
        System.out.println("Saldo da Conta 1 após transferência: R$ " +
conta1.saldo);
        System.out.println("Saldo da Conta 2 após transferência: R$ " +
conta2.saldo);
    }
}
```

Explicação: A classe ContaBancaria possui atributos de saldo e número da conta. O método transferir() permite que uma conta transfira um valor para outra conta, modificando o saldo de ambas as contas. No método main(), duas contas são criadas, e uma transferência de R$ 200,00 é feita entre elas. Este exercício ensina como criar métodos que interagem com outros objetos e modificam o estado de múltiplos objetos ao mesmo tempo, uma prática comum em sistemas bancários e financeiros.

Exercício 147: Classe Funcionario com Bônus

Enunciado: Implemente uma classe Funcionario com os atributos nome e salario. Adicione um método receberBonus(double percentual) que aumente o salário com base em um percentual. No método main(), crie um funcionário, aplique o bônus e exiba o novo salário.

Solução:

```
public class Funcionario {
    String nome;
    double salario;

    // Método para aplicar o bônus no salário
    public void receberBonus(double percentual) {
        salario += salario * (percentual / 100);
    }
```

```
    public static void main(String[] args) {
        Funcionario funcionario = new Funcionario();
        funcionario.nome = "Mariana";
        funcionario.salario = 5000.0;

        // Exibindo o salário antes do bônus
        System.out.println("Salário antes do bônus: R$ " +
funcionario.salario);

        // Aplicando um bônus de 10%
        funcionario.receberBonus(10);

        // Exibindo o salário após o bônus
        System.out.println("Salário após o bônus: R$ " +
funcionario.salario);
    }
}
```

Explicação: Neste exercício, a classe Funcionario tem dois atributos: nome e salario. O método receberBonus() é responsável por aumentar o salário com base em uma porcentagem fornecida. No método main(), um funcionário é criado, o salário é exibido antes e depois da aplicação do bônus de 10%. Este exercício demonstra como os métodos de instância podem modificar atributos com base em valores dinâmicos, como percentuais de aumento salarial, e como o valor resultante é retornado e utilizado dentro do código.

Exercício 148: Classe Aluno com Aprovação

Enunciado: Crie uma classe Aluno com os atributos nome, nota1 e nota2. Adicione um método verificarAprovacao() que calcule a média e retorne se o aluno foi aprovado (média maior ou igual a 7). No método main(), crie um aluno e verifique sua aprovação.

Solução:

```
public class Aluno {
    String nome;
    double nota1;
    double nota2;

    // Método para verificar se o aluno foi aprovado
    public boolean verificarAprovacao() {
        double media = (nota1 + nota2) / 2;
```

```
        return media >= 7.0;
    }

    public static void main(String[] args) {
        Aluno aluno = new Aluno();
        aluno.nome = "Lucas";
        aluno.nota1 = 8.0;
        aluno.nota2 = 6.5;

        // Verificando a aprovação
        if (aluno.verificarAprovacao()) {
            System.out.println(aluno.nome + " foi aprovado.");
        } else {
            System.out.println(aluno.nome + " foi reprovado.");
        }
    }
}
```

Explicação: Este exercício apresenta a classe Aluno, que possui dois atributos para as notas do aluno. O método verificarAprovacao() calcula a média entre nota1 e nota2 e verifica se o aluno está aprovado, retornando true ou false dependendo da média. No método main(), é criado um aluno e a verificação de aprovação é exibida no console. Esse exercício ensina a realizar cálculos simples e a tomar decisões com base no resultado, usando um método que retorna um valor booleano.

Exercício 149: Classe Retângulo com Alteração de Dimensões

Enunciado: Crie uma classe Retangulo com os atributos largura e altura. Adicione métodos alterarDimensoes(double novaLargura, double novaAltura) e calcularArea() para atualizar as dimensões e calcular a área. No método main(), altere as dimensões de um retângulo e exiba a nova área.

Solução:

```
public class Retangulo {
    double largura;
    double altura;

    // Método para alterar as dimensões do retângulo
    public void alterarDimensoes(double novaLargura, double novaAltura)
{
```

```
        largura = novaLargura;
        altura = novaAltura;
    }

    // Método para calcular a área do retângulo
    public double calcularArea() {
        return largura * altura;
    }

    public static void main(String[] args) {
        Retangulo retangulo = new Retangulo();
        retangulo.largura = 5.0;
        retangulo.altura = 3.0;

        // Exibindo a área inicial
        System.out.println("Área inicial: " + retangulo.calcularArea());

        // Alterando as dimensões
        retangulo.alterarDimensoes(10.0, 8.0);

        // Exibindo a nova área
        System.out.println("Nova área: " + retangulo.calcularArea());
    }
}
```

Explicação: Aqui, a classe Retangulo contém métodos que permitem alterar suas dimensões com alterarDimensoes() e calcular a área com calcularArea(). No método main(), o retângulo é criado com dimensões iniciais e sua área é calculada. Em seguida, as dimensões são alteradas e a nova área é exibida. Esse exercício mostra como métodos podem ser usados para modificar o estado interno de um objeto e como operações baseadas nos atributos atualizados podem ser realizadas.

Exercício 150: Classe Produto com Aumento de Preço

Enunciado: Implemente uma classe Produto com os atributos nome, preco e quantidade. Adicione um método aumentarPreco(double porcentagem) que aumente o preço com base na porcentagem. No método main(), crie um produto, aumente o preço e exiba o novo valor.

Solução:

```
public class Produto {
    String nome;
    double preco;
    int quantidade;
```

```
    // Método para aumentar o preço com base na porcentagem
    public void aumentarPreco(double porcentagem) {
        preco += preco * (porcentagem / 100);
    }

    public static void main(String[] args) {
        Produto produto = new Produto();
        produto.nome = "Geladeira";
        produto.preco = 1500.0;
        produto.quantidade = 20;

        // Exibindo o preço inicial
        System.out.println("Preço inicial: R$ " + produto.preco);

        // Aumentando o preço em 15%
        produto.aumentarPreco(15);

        // Exibindo o novo preço
        System.out.println("Preço após aumento: R$ " + produto.preco);
    }
}
```

Explicação: Neste exercício, a classe Produto tem o método aumentarPreco(), que aplica um aumento percentual ao preço do produto. No método main(), o produto é criado, e o preço é aumentado em 15%. O exercício demonstra como manipular o valor de atributos através de cálculos matemáticos em métodos, e como mudanças de estado podem ser refletidas no comportamento de um objeto.

Exercício 151: Classe Pessoa com Altura e Peso

Enunciado: Crie uma classe Pessoa com os atributos nome, altura e peso. Adicione métodos alterarPeso(double novoPeso) e alterarAltura(double novaAltura). No método main(), altere o peso e a altura de uma pessoa e exiba os valores atualizados.

Solução:

```
public class Pessoa {
    String nome;
    double altura;
    double peso;
```

```
    // Método para alterar o peso da pessoa
    public void alterarPeso(double novoPeso) {
        peso = novoPeso;
    }

    // Método para alterar a altura da pessoa
    public void alterarAltura(double novaAltura) {
        altura = novaAltura;
    }

    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa();
        pessoa.nome = "Joana";
        pessoa.altura = 1.70;
        pessoa.peso = 65.0;

        // Exibindo os valores iniciais
        System.out.println("Altura inicial: " + pessoa.altura + " m");
        System.out.println("Peso inicial: " + pessoa.peso + " kg");

        // Alterando peso e altura
        pessoa.alterarPeso(68.0);
        pessoa.alterarAltura(1.72);

        // Exibindo os novos valores
        System.out.println("Nova altura: " + pessoa.altura + " m");
        System.out.println("Novo peso: " + pessoa.peso + " kg");
    }
}
```

Explicação: Neste exercício, a classe Pessoa possui métodos que permitem alterar os atributos peso e altura. O exercício foca em como manipular diretamente os atributos do objeto e atualizar os valores dinamicamente através de métodos. No método main(), o peso e a altura da pessoa são modificados, e os novos valores são exibidos no console. Este exemplo ilustra como métodos podem ser usados para modificar o estado interno de objetos.

Exercício 152: Classe Banco com Saldo Total

Enunciado: Crie uma classe Banco que tenha uma lista de contas bancárias como atributo. Adicione um método calcularSaldoTotal() que some os saldos de todas as contas. No método main(), crie várias contas, adicione-as ao banco e exiba o saldo total.

Solução:

```java
import java.util.ArrayList;

public class Banco {
    ArrayList<ContaBancaria> contas = new ArrayList<>();

    // Método para calcular o saldo total do banco
    public double calcularSaldoTotal() {
        double saldoTotal = 0;
        for (ContaBancaria conta : contas) {
            saldoTotal += conta.saldo;
        }
        return saldoTotal;
    }

    public static void main(String[] args) {
        Banco banco = new Banco();

        // Criando contas bancárias
        ContaBancaria conta1 = new ContaBancaria();
        conta1.numeroConta = 12345;
        conta1.saldo = 1000.0;

        ContaBancaria conta2 = new ContaBancaria();
        conta2.numeroConta = 67890;
        conta2.saldo = 2000.0;

        // Adicionando as contas ao banco
        banco.contas.add(conta1);
        banco.contas.add(conta2);

        // Exibindo o saldo total do banco
        System.out.println("Saldo total no banco: R$ " +
banco.calcularSaldoTotal());
    }
}

class ContaBancaria {
    int numeroConta;
    double saldo;
}
```

Explicação: A classe Banco tem um atributo contas, que é uma lista de objetos da classe ContaBancaria. O método calcularSaldoTotal() percorre todas as contas e soma seus saldos. No método main(), duas contas bancárias são criadas, adicionadas ao banco, e o saldo total é calculado e exibido. Esse exercício ensina a trabalhar com coleções de objetos

e a manipular seus atributos através de métodos que operam em conjunto com outros objetos.

Exercício 153: Classe Livro com Empréstimo

Enunciado: Implemente uma classe Livro com os atributos titulo, autor e disponivel. Adicione um método emprestar() que marque o livro como indisponível e outro método devolver() que marque como disponível. No método main(), crie um livro e simule um empréstimo e uma devolução.

Solução:

```
public class Livro {
    String titulo;
    String autor;
    boolean disponivel;

    // Método para emprestar o livro
    public void emprestar() {
        if (disponivel) {
            disponivel = false;
            System.out.println("O livro foi emprestado.");
        } else {
            System.out.println("O livro já está emprestado.");
        }
    }

    // Método para devolver o livro
    public void devolver() {
        if (!disponivel) {
            disponivel = true;
            System.out.println("O livro foi devolvido.");
        } else {
            System.out.println("O livro já estava disponível.");
        }
    }

    public static void main(String[] args) {
        Livro livro = new Livro();
        livro.titulo = "1984";
        livro.autor = "George Orwell";
        livro.disponivel = true;

        // Simulando o empréstimo e a devolução do livro
        livro.emprestar();
```

```
        livro.devolver();
    }
}
```

Explicação: A classe Livro tem atributos que descrevem o estado de disponibilidade do livro. O método emprestar() altera o atributo disponivel para false, indicando que o livro não está mais disponível, enquanto o método devolver() o marca como disponível novamente. No método main(), o livro é emprestado e devolvido, simulando as mudanças de estado com base nas operações realizadas. Este exercício demonstra a manipulação de estados booleanos em objetos e como os métodos podem alterar os atributos de maneira condicional.

Exercício 154: Classe Veículo com Abastecimento

Enunciado: Crie uma classe Veiculo com os atributos modelo e combustivel (em litros). Adicione um método abastecer(double litros) que aumente a quantidade de combustível. No método main(), crie um veículo, simule um abastecimento e exiba o combustível atualizado.

Solução:

```
public class Veiculo {
    String modelo;
    double combustivel;

    // Método para abastecer o veículo
    public void abastecer(double litros) {
        combustivel += litros;
    }

    public static void main(String[] args) {
        Veiculo veiculo = new Veiculo();
        veiculo.modelo = "SUV";
        veiculo.combustivel = 20.0;

        // Exibindo o combustível inicial
        System.out.println("Combustível inicial: " + veiculo.combustivel
+ " litros");

        // Abastecendo o veículo com 30 litros
        veiculo.abastecer(30);

        // Exibindo o combustível após o abastecimento
        System.out.println("Combustível após abastecimento: " +
veiculo.combustivel + " litros");
```

```
    }
}
```

Explicação: A classe Veiculo contém um atributo combustivel, que representa a quantidade de combustível no tanque. O método abastecer() aumenta o valor do combustível com base no número de litros fornecido. No método main(), o veículo é criado com uma quantidade inicial de combustível, e o método abastecer() é chamado para aumentar o combustível disponível. Este exercício ilustra como métodos podem ser usados para incrementar atributos de instância.

Exercício 155: Classe Loja com Produtos

Enunciado: Crie uma classe Loja com o atributo nome e uma lista de produtos. Adicione um método adicionarProduto(Produto produto) que adicione um produto à lista e um método exibirProdutos() que exiba todos os produtos. No método main(), crie uma loja, adicione produtos e exiba a lista.

Solução:

```
import java.util.ArrayList;

public class Loja {
    String nome;
    ArrayList<Produto> produtos = new ArrayList<>();

    // Método para adicionar produto à lista
    public void adicionarProduto(Produto produto) {
        produtos.add(produto);
    }

    // Método para exibir todos os produtos
    public void exibirProdutos() {
        System.out.println("Produtos da loja " + nome + ":");
        for (Produto produto : produtos) {
            System.out.println(produto.nome + " - R$ " + produto.preco);
        }
    }

    public static void main(String[] args) {
        Loja loja = new Loja();
        loja.nome = "Loja de Eletrônicos";

        Produto produto1 = new Produto();
        produto1.nome = "Smartphone";
```

```
        produto1.preco = 1500.0;

        Produto produto2 = new Produto();
        produto2.nome = "Televisão";
        produto2.preco = 3000.0;

        // Adicionando produtos à loja
        loja.adicionarProduto(produto1);
        loja.adicionarProduto(produto2);

        // Exibindo os produtos da loja
        loja.exibirProdutos();
    }
}

class Produto {
    String nome;
    double preco;
}
```

Explicação: Neste exercício, a classe Loja contém uma lista de produtos, e o método adicionarProduto() permite adicionar novos produtos à lista. O método exibirProdutos() percorre a lista e exibe os detalhes de cada produto. No método main(), uma loja é criada, produtos são adicionados e, finalmente, a lista de produtos é exibida. O exercício ensina como trabalhar com coleções de objetos dentro de uma classe e como manipular listas de objetos.

Exercício 156: Classe Cesta de Compras com Total

Enunciado: Crie uma classe CestaDeCompras com uma lista de produtos. Adicione um método calcularTotal() que some o valor total dos produtos da cesta. No método main(), adicione produtos à cesta e exiba o valor total.

Solução:

```
import java.util.ArrayList;

public class CestaDeCompras {
    ArrayList<Produto> produtos = new ArrayList<>();
```

```
    // Método para calcular o valor total da cesta de compras
    public double calcularTotal() {
        double total = 0;
        for (Produto produto : produtos) {
            total += produto.preco;
        }
        return total;
    }

    public static void main(String[] args) {
        CestaDeCompras cesta = new CestaDeCompras();

        Produto produto1 = new Produto();
        produto1.nome = "Notebook";
        produto1.preco = 3000.0;

        Produto produto2 = new Produto();
        produto2.nome = "Fone de Ouvido";
        produto2.preco = 200.0;

        // Adicionando produtos à cesta
        cesta.produtos.add(produto1);
        cesta.produtos.add(produto2);

        // Exibindo o valor total da cesta
        System.out.println("Valor total da cesta: R$ " +
cesta.calcularTotal());
    }
}

class Produto {
    String nome;
    double preco;
}
```

Explicação: A classe CestaDeCompras mantém uma lista de produtos, e o método calcularTotal() percorre essa lista, somando os preços dos produtos para determinar o valor total. No método main(), a cesta é preenchida com produtos, e o valor total é calculado e exibido. Este exercício ilustra como trabalhar com listas e como métodos podem realizar operações sobre coleções de objetos.

Exercício 157: Classe Restaurante com Pedidos

Enunciado: Implemente uma classe Restaurante com uma lista de pedidos. Adicione um método adicionarPedido(String pedido) e outro método exibirPedidos() para exibir todos os pedidos. No método main(), crie um restaurante, adicione pedidos e exiba a lista.

Solução:

```
import java.util.ArrayList;

public class Restaurante {
    ArrayList<String> pedidos = new ArrayList<>();

    // Método para adicionar um pedido
    public void adicionarPedido(String pedido) {
        pedidos.add(pedido);
    }

    // Método para exibir todos os pedidos
    public void exibirPedidos() {
        System.out.println("Pedidos realizados:");
        for (String pedido : pedidos) {
            System.out.println(pedido);
        }
    }

    public static void main(String[] args) {
        Restaurante restaurante = new Restaurante();

        // Adicionando pedidos
        restaurante.adicionarPedido("Pizza");
        restaurante.adicionarPedido("Lasanha");
        restaurante.adicionarPedido("Hambúrguer");

        // Exibindo os pedidos
        restaurante.exibirPedidos();
    }
}
```

Explicação: A classe Restaurante armazena uma lista de pedidos como strings. O método adicionarPedido() adiciona um novo pedido à lista, e o método exibirPedidos() exibe todos os pedidos feitos. No método main(), o restaurante é criado, pedidos são adicionados, e a lista é exibida. Este exercício mostra como armazenar e manipular dados simples (strings) em uma lista, e como métodos podem operar sobre essas coleções para exibir informações de forma organizada.

Encapsulamento (getters e setters)

Exercício 158: Classe ContaBancaria com Encapsulamento

Enunciado: Implemente uma classe ContaBancaria com os atributos saldo e numeroConta, encapsulados com os métodos getSaldo(), setSaldo(), getNumeroConta() e setNumeroConta(). No método main(), crie uma conta bancária, defina seus valores e exiba o saldo usando os getters.

Solução:

```
public class ContaBancaria {
    private double saldo;
    private int numeroConta;

    // Getter para saldo
    public double getSaldo() {
        return saldo;
    }

    // Setter para saldo
    public void setSaldo(double saldo) {
        if (saldo >= 0) {
            this.saldo = saldo;
        } else {
            System.out.println("Saldo não pode ser negativo.");
        }
    }

    // Getter para número da conta
    public int getNumeroConta() {
        return numeroConta;
    }

    // Setter para número da conta
    public void setNumeroConta(int numeroConta) {
        this.numeroConta = numeroConta;
    }

    public static void main(String[] args) {
        ContaBancaria conta = new ContaBancaria();

        // Definindo valores através dos setters
        conta.setNumeroConta(12345);
```

```
        conta.setSaldo(1000.0);

        // Exibindo valores através dos getters
        System.out.println("Número da conta: " +
conta.getNumeroConta());
        System.out.println("Saldo da conta: R$ " + conta.getSaldo());
    }
}
```

Explicação: Neste exercício, a classe ContaBancaria encapsula os atributos saldo e numeroConta utilizando getters e setters. O método setSaldo() possui uma verificação para garantir que o saldo nunca seja negativo. No método main(), utilizamos os setters para definir os valores e os getters para exibi-los. O encapsulamento garante que os atributos não sejam acessados diretamente, mas sim controlados por métodos que podem validar e processar as informações antes de modificar o estado interno do objeto.

Exercício 159: Classe Produto com Validação de Preço

Enunciado: Crie uma classe Produto com os atributos nome e preco. O atributo preco deve ser encapsulado, e o método setPreco() deve garantir que o preço não seja negativo. No método main(), crie um produto, tente definir um preço negativo e exiba a mensagem de erro.

Solução:

```
public class Produto {
    private String nome;
    private double preco;

    // Getter para o preço
    public double getPreco() {
        return preco;
    }

    // Setter para o preço com validação
    public void setPreco(double preco) {
        if (preco >= 0) {
            this.preco = preco;
        } else {
            System.out.println("Erro: O preço não pode ser negativo.");
        }
    }

    // Getter e Setter para o nome
```

```
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public static void main(String[] args) {
        Produto produto = new Produto();

        // Definindo nome e tentando definir um preço negativo
        produto.setNome("Smartphone");
        produto.setPreco(-500.0);  // Deverá exibir uma mensagem de erro

        // Definindo um preço válido
        produto.setPreco(2000.0);

        // Exibindo o nome e preço do produto
        System.out.println("Produto: " + produto.getNome());
        System.out.println("Preço: R$ " + produto.getPreco());
    }
}
```

Explicação: Neste exemplo, o atributo preco está encapsulado, e o método setPreco() possui uma verificação para evitar valores negativos. Quando tentamos definir um preço negativo, uma mensagem de erro é exibida. No método main(), após o erro, um valor válido é atribuído, e o preço correto é exibido usando o getter. Isso garante a integridade dos dados e impede que o valor do preço seja modificado para valores inapropriados.

Exercício 160: Classe Pessoa com Validação de Idade

Enunciado: Implemente uma classe Pessoa com os atributos nome e idade. Encapsule a idade e adicione uma validação no método setIdade() para garantir que a idade seja maior que 0. No método main(), crie uma pessoa, defina uma idade inválida e exiba a mensagem de erro.

Solução:

```
public class Pessoa {
    private String nome;
    private int idade;

    // Getter para idade
```

```
    public int getIdade() {
        return idade;
    }

    // Setter para idade com validação
    public void setIdade(int idade) {
        if (idade > 0) {
            this.idade = idade;
        } else {
            System.out.println("Erro: A idade deve ser maior que 0.");
        }
    }

    // Getter e Setter para nome
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa();

        // Definindo nome e uma idade inválida
        pessoa.setNome("Carlos");
        pessoa.setIdade(-5);  // Deverá exibir uma mensagem de erro

        // Definindo uma idade válida
        pessoa.setIdade(25);

        // Exibindo os dados da pessoa
        System.out.println("Nome: " + pessoa.getNome());
        System.out.println("Idade: " + pessoa.getIdade());
    }
}
```

Explicação: A classe Pessoa encapsula o atributo idade, e o método setIdade() garante que a idade seja maior que 0. Quando uma idade inválida é fornecida, uma mensagem de erro é exibida. No método main(), primeiro definimos uma idade inválida para demonstrar a validação, e depois ajustamos o valor para uma idade válida. O uso de getters e setters permite que o controle sobre a idade seja centralizado e validado antes de ser atribuído.

Exercício 161: Classe Funcionario com Bônus Encapsulado

Enunciado: Crie uma classe Funcionario com os atributos nome, salario e bonus. O atributo bonus deve ser calculado como uma porcentagem do salário e encapsulado com getBonus() e setBonus(). No método main(), defina o salário, calcule o bônus e exiba os valores usando os getters.

Solução:

```
public class Funcionario {
    private String nome;
    private double salario;
    private double bonus;

    // Getter para o salário
    public double getSalario() {
        return salario;
    }

    // Setter para o salário
    public void setSalario(double salario) {
        this.salario = salario;
    }

    // Getter para o bônus
    public double getBonus() {
        return bonus;
    }

    // Setter para o bônus (calculado como 10% do salário)
    public void setBonus(double percentual) {
        this.bonus = salario * (percentual / 100);
    }

    // Getter e Setter para o nome
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public static void main(String[] args) {
        Funcionario funcionario = new Funcionario();
```

```
        // Definindo o nome e salário
        funcionario.setNome("Ana");
        funcionario.setSalario(5000.0);

        // Definindo o bônus como 10% do salário
        funcionario.setBonus(10);

        // Exibindo os dados do funcionário
        System.out.println("Funcionário: " + funcionario.getNome());
        System.out.println("Salário: R$ " + funcionario.getSalario());
        System.out.println("Bônus: R$ " + funcionario.getBonus());
    }
}
```

Explicação: Neste exercício, a classe Funcionario encapsula os atributos salario e bonus. O bônus é calculado como uma porcentagem do salário, e os métodos getBonus() e setBonus() garantem que o valor seja ajustado corretamente. No método main(), o nome, salário e bônus são definidos, e os valores são exibidos. Esse exercício ilustra como os setters podem ser usados para realizar cálculos automáticos ao definir atributos.

Exercício 162: Classe Veiculo com Encapsulamento de Combustível

Enunciado: Implemente uma classe Veiculo com o atributo combustivel. O combustível deve ser encapsulado, e o método setCombustivel() deve garantir que o valor esteja entre 0 e o valor máximo de combustível do tanque. No método main(), tente definir um valor maior que o permitido e exiba a mensagem de erro.

Solução:

```
public class Veiculo {
    private double combustivel;
    private final double capacidadeTanque = 50.0;  // Capacidade máxima
de combustível

    // Getter para combustível
    public double getCombustivel() {
        return combustivel;
    }

    // Setter para combustível com validação
    public void setCombustivel(double combustivel) {
        if (combustivel >= 0 && combustivel <= capacidadeTanque) {
            this.combustivel = combustivel;
```

```
        } else {
            System.out.println("Erro: Combustível deve estar entre 0 e "
+ capacidadeTanque + " litros.");
        }
    }

    public static void main(String[] args) {
        Veiculo veiculo = new Veiculo();

        // Tentando definir um valor de combustível inválido
        veiculo.setCombustivel(60.0);  // Deve exibir uma mensagem de
erro

        // Definindo um valor válido
        veiculo.setCombustivel(30.0);

        // Exibindo o valor de combustível
        System.out.println("Combustível no tanque: " +
veiculo.getCombustivel() + " litros.");
    }
}
```

Explicação: Neste exercício, a classe Veiculo encapsula o atributo combustivel e impõe uma validação no método setCombustivel() para garantir que o valor esteja entre 0 e a capacidade máxima do tanque, que é de 50 litros. Se um valor inválido for fornecido, uma mensagem de erro é exibida. No método main(), um valor inválido é tentado primeiro para demonstrar a validação, e depois um valor correto é atribuído e exibido. Esse exercício reforça a importância do controle de dados ao permitir que valores sejam inseridos em um objeto.

Conclusão do Capítulo 7: Classes e Objetos

Neste capítulo, exploramos os conceitos fundamentais da Programação Orientada a Objetos (POO), focando na criação de classes e objetos, definição de atributos e métodos, e a prática do encapsulamento por meio de getters e setters. Através de exercícios práticos, foi possível compreender como organizar o código em torno de objetos, permitindo uma estrutura mais modular, reutilizável e fácil de manter.

Primeiramente, aprendemos a definir classes e objetos, que são a base da POO. Os exercícios demonstraram como criar instâncias de classes, manipular atributos e métodos, e utilizar esses objetos para representar entidades reais, como carros, pessoas, e produtos.

Em seguida, abordamos o uso de atributos e métodos de instância, mostrando como encapsular informações e definir o comportamento dos objetos. Esse conceito permitiu

explorar a capacidade de alterar dinamicamente os atributos, utilizando métodos específicos para lidar com as interações entre os objetos.

Por fim, vimos como aplicar o encapsulamento com getters e setters, protegendo os dados internos das classes e garantindo que somente valores válidos sejam atribuídos aos atributos. O encapsulamento é essencial para garantir a integridade dos dados e proporcionar um controle maior sobre as modificações internas dos objetos.

Introdução ao Capítulo 8: Herança e Polimorfismo

No Capítulo 8, vamos explorar dois dos pilares fundamentais da Programação Orientada a Objetos (POO): Herança e Polimorfismo. Esses conceitos permitem a criação de sistemas mais flexíveis, reutilizáveis e eficientes.

Primeiramente, veremos a herança, que possibilita que uma classe derive características de outra, permitindo o reaproveitamento de código e a criação de hierarquias entre classes. A sobrescrita de métodos permitirá que classes filhas modifiquem o comportamento de métodos herdados da classe mãe.

Por fim, vamos aprender sobre o polimorfismo, que permite que objetos de diferentes classes sejam tratados de maneira uniforme, tornando o código mais dinâmico e adaptável. Este capítulo mostrará como esses conceitos se complementam para melhorar a estrutura e flexibilidade de sistemas orientados a objetos.

Herança Simples

Exercício 163: Classe Animal e Subclasse Cachorro

Enunciado: Crie uma classe Animal com os atributos nome e idade. Em seguida, crie uma subclasse Cachorro que herda de Animal e tenha o método latir(), que exibe uma mensagem no console. No método main(), crie um objeto Cachorro e chame seus métodos.

Código de Solução:

```
class Animal {
    String nome;
    int idade;

    public void exibirDetalhes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade + " anos");
```

```
    }
}

class Cachorro extends Animal {
    public void latir() {
        System.out.println("O cachorro está latindo: Au Au!");
    }
}

public class Main {
    public static void main(String[] args) {
        Cachorro cachorro = new Cachorro();
        cachorro.nome = "Rex";
        cachorro.idade = 3;

        cachorro.exibirDetalhes();
        cachorro.latir();
    }
}
```

Explicação: A classe Animal define os atributos nome e idade, e possui um método exibirDetalhes() que exibe essas informações. A classe Cachorro herda de Animal, portanto, também tem acesso aos atributos e métodos da classe pai. Além disso, Cachorro possui seu próprio método, latir(). No método main(), criamos uma instância de Cachorro, definimos o nome e a idade (herdados da classe Animal) e chamamos o método latir(). A herança permite que Cachorro reutilize o código já presente na classe Animal, evitando duplicação de código e promovendo a reutilização.

Exercício 164: Classe Veiculo e Subclasse Carro

Enunciado: Crie uma classe Veiculo com os atributos marca e modelo. Crie uma subclasse Carro que herda de Veiculo e tenha um atributo adicional ano. No método main(), crie um objeto Carro e exiba todos os atributos herdados e o atributo ano.

Código de Solução:

```
class Veiculo {
    String marca;
    String modelo;

    public void exibirDetalhes() {
        System.out.println("Marca: " + marca);
        System.out.println("Modelo: " + modelo);
    }
```

```
}

class Carro extends Veiculo {
    int ano;

    public void exibirAno() {
        System.out.println("Ano: " + ano);
    }
}

public class Main {
    public static void main(String[] args) {
        Carro carro = new Carro();
        carro.marca = "Toyota";
        carro.modelo = "Corolla";
        carro.ano = 2020;

        carro.exibirDetalhes();
        carro.exibirAno();
    }
}
```

Explicação: Neste exercício, a classe Veiculo possui dois atributos: marca e modelo. A classe Carro herda esses atributos e métodos de Veiculo, mas também possui seu próprio atributo ano e o método exibirAno(). A herança permite que Carro reutilize o código da classe Veiculo, garantindo que não seja necessário redefinir marca e modelo novamente em Carro. No método main(), criamos um objeto Carro, definimos os valores para os atributos herdados e o novo atributo ano, e exibimos todas as informações. Essa abordagem promove a reutilização e a organização de código.

Exercício 165: Classe Pessoa e Subclasse Funcionario

Enunciado: Implemente uma classe Pessoa com os atributos nome e idade. Crie uma subclasse Funcionario que herda de Pessoa e tenha o atributo salario. No método main(), crie um objeto Funcionario, atribua valores aos atributos e exiba as informações no console.

Código de Solução:

```
class Pessoa {
    String nome;
    int idade;

    public void exibirInformacoes() {
```

```
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade + " anos");
    }
}

class Funcionario extends Pessoa {
    double salario;

    public void exibirSalario() {
        System.out.println("Salário: R$ " + salario);
    }
}

public class Main {
    public static void main(String[] args) {
        Funcionario funcionario = new Funcionario();
        funcionario.nome = "Maria";
        funcionario.idade = 30;
        funcionario.salario = 4000.0;

        funcionario.exibirInformacoes();
        funcionario.exibirSalario();
    }
}
```

Explicação: A classe Pessoa contém os atributos nome e idade, e a classe Funcionario herda esses atributos, adicionando o atributo salario. O método exibirSalario() é específico da classe Funcionario, enquanto exibirInformacoes() é herdado da classe Pessoa. No método main(), criamos uma instância de Funcionario e definimos os atributos herdados e o novo atributo salario. Através da herança, Funcionario pode acessar tanto os métodos e atributos da classe pai quanto seus próprios métodos e atributos.

Exercício 166: Classe Conta e Subclasse ContaCorrente

Enunciado: Crie uma classe Conta com o atributo saldo e um método depositar(). Em seguida, crie uma subclasse ContaCorrente que herda de Conta e adicione um método sacar(). No método main(), crie um objeto ContaCorrente, realize um depósito e um saque, e exiba o saldo final.

Código de Solução:

```
class Conta {
    double saldo;
```

```
    public void depositar(double valor) {
        saldo += valor;
        System.out.println("Depósito de R$ " + valor + " realizado.
Saldo atual: R$ " + saldo);
    }
}

class ContaCorrente extends Conta {

    public void sacar(double valor) {
        if (valor <= saldo) {
            saldo -= valor;
            System.out.println("Saque de R$ " + valor + " realizado.
Saldo atual: R$ " + saldo);
        } else {
            System.out.println("Saldo insuficiente para realizar o
saque.");
        }
    }
}

public class Main {
    public static void main(String[] args) {
        ContaCorrente conta = new ContaCorrente();
        conta.depositar(1000.0);
        conta.sacar(500.0);
        conta.sacar(600.0);  // Tentativa de saque maior que o saldo
    }
}
```

Explicação: A classe Conta possui o atributo saldo e o método depositar(), que aumenta o saldo. A subclasse ContaCorrente herda esses elementos de Conta e adiciona o método sacar(), que permite a retirada de dinheiro desde que o saldo seja suficiente. No método main(), criamos uma instância de ContaCorrente, fazemos um depósito e realizamos saques. O uso da herança aqui permite que ContaCorrente reutilize o comportamento de Conta, enquanto adiciona novas funcionalidades específicas para contas correntes.

Exercício 167: Classe Produto e Subclasse Livro

Enunciado: Crie uma classe Produto com os atributos nome e preco. Crie uma subclasse Livro que herda de Produto e tenha o atributo autor. No método main(), crie um objeto Livro, atribua valores a seus atributos e exiba todas as informações.

Código de Solução:

```java
class Produto {
    String nome;
    double preco;

    public void exibirDetalhes() {
        System.out.println("Nome: " + nome);
        System.out.println("Preço: R$ " + preco);
    }
}

class Livro extends Produto {
    String autor;

    public void exibirAutor() {
        System.out.println("Autor: " + autor);
    }
}

public class Main {
    public static void main(String[] args) {
        Livro livro = new Livro();
        livro.nome = "O Senhor dos Anéis";
        livro.preco = 59.90;
        livro.autor = "J.R.R. Tolkien";

        livro.exibirDetalhes();
        livro.exibirAutor();
    }
}
```

Explicação: Neste exercício, a classe Produto define os atributos nome e preco, além de um método para exibir esses detalhes. A classe Livro, que herda de Produto, adiciona o atributo autor e um método específico para exibir o autor. No método main(), criamos uma instância de Livro, definimos todos os atributos e exibimos as informações. A herança permite que Livro reutilize os atributos e métodos de Produto, acrescentando apenas o que é específico para a subclasse, mantendo o código mais limpo e modular.

Exercício 168: Classe Pessoa e Subclasse Aluno

Enunciado: Implemente uma classe Pessoa com os atributos nome e cpf. Crie uma subclasse Aluno que herda de Pessoa e adicione o atributo matricula. No método main(), crie um objeto Aluno e exiba os dados herdados e o número de matrícula.

Código de Solução:

```
class Pessoa {
    String nome;
    String cpf;

    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("CPF: " + cpf);
    }
}

class Aluno extends Pessoa {
    String matricula;

    public void exibirMatricula() {
        System.out.println("Matrícula: " + matricula);
    }
}

public class Main {
    public static void main(String[] args) {
        Aluno aluno = new Aluno();
        aluno.nome = "João";
        aluno.cpf = "123.456.789-00";
        aluno.matricula = "20230001";

        aluno.exibirInformacoes();
        aluno.exibirMatricula();
    }
}
```

Explicação: A classe Pessoa contém os atributos nome e cpf, e a classe Aluno herda esses atributos, adicionando o atributo matricula e o método exibirMatricula(). No método main(), criamos um objeto Aluno, definimos os valores herdados e o valor de matricula, e exibimos todas as informações. A herança permite que Aluno reutilize os métodos e atributos da classe Pessoa, tornando o código mais reutilizável e organizado.

Sobrescrita de Métodos

Exercício 169: Sobrescrita de Método na Classe Animal

Enunciado: Crie uma classe Animal com o método som() que exibe "O animal está fazendo um som". Crie uma subclasse Gato que sobrescreve o método som() para exibir "O gato está miando". No método main(), crie objetos de ambas as classes e chame o método som().

Código de Solução:

```
class Animal {
    public void som() {
        System.out.println("O animal está fazendo um som.");
    }
}

class Gato extends Animal {
    @Override
    public void som() {
        System.out.println("O gato está miando.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Gato gato = new Gato();

        animal.som();  // Exibe som genérico de animal
        gato.som();    // Exibe som específico do gato
    }
}
```

Explicação: Neste exercício, a classe Animal define um método som() que exibe uma mensagem genérica sobre o som de um animal. A classe Gato herda de Animal e sobrescreve o método som() para exibir uma mensagem mais específica: "O gato está miando". No método main(), criamos um objeto de Animal e outro de Gato, chamando o método som() de ambos. A sobrescrita de método permite que Gato forneça uma implementação personalizada de som(), substituindo o comportamento herdado de Animal.

Exercício 170: Sobrescrita de Método na Classe Veiculo

Enunciado: Implemente uma classe Veiculo com o método acelerar() que exibe "O veículo está acelerando". Crie uma subclasse Moto que sobrescreva esse método para exibir "A moto está acelerando rapidamente". No método main(), crie um objeto de cada classe e chame o método acelerar().

Código de Solução:

```
class Veiculo {
    public void acelerar() {
        System.out.println("O veículo está acelerando.");
    }
}

class Moto extends Veiculo {
    @Override
    public void acelerar() {
        System.out.println("A moto está acelerando rapidamente.");
    }
}

public class Main {
    public static void main(String[] args) {
        Veiculo veiculo = new Veiculo();
        Moto moto = new Moto();

        veiculo.acelerar();  // Exibe aceleração genérica do veículo
        moto.acelerar();     // Exibe aceleração específica da moto
    }
}
```

Explicação: A classe Veiculo tem um método acelerar() que exibe uma mensagem genérica sobre a aceleração de um veículo. A subclasse Moto sobrescreve esse método para exibir uma mensagem mais específica: "A moto está acelerando rapidamente". No método main(), criamos instâncias de Veiculo e Moto e chamamos o método acelerar() de ambos. A sobrescrita permite que Moto tenha um comportamento específico ao acelerar, apesar de herdar a estrutura básica de Veiculo.

Exercício 171: Sobrescrita de Método na Classe Pessoa

Enunciado: Crie uma classe Pessoa com o método trabalhar() que exibe "A pessoa está trabalhando". Crie uma subclasse Professor que sobrescreva o método para exibir "O professor está dando aula". No método main(), crie objetos de ambas as classes e chame o método trabalhar().

Código de Solução:

```
class Pessoa {
    public void trabalhar() {
```

```
        System.out.println("A pessoa está trabalhando.");
    }
}

class Professor extends Pessoa {
    @Override
    public void trabalhar() {
        System.out.println("O professor está dando aula.");
    }
}

public class Main {
    public static void main(String[] args) {
        Pessoa pessoa = new Pessoa();
        Professor professor = new Professor();

        pessoa.trabalhar();   // Exibe trabalho genérico
        professor.trabalhar();  // Exibe trabalho específico do
professor
    }
}
```

Explicação: A classe Pessoa possui um método trabalhar() que exibe uma mensagem genérica. A classe Professor herda de Pessoa e sobrescreve o método trabalhar() para exibir uma mensagem específica: "O professor está dando aula". A sobrescrita permite que a subclasse forneça uma implementação especializada para o método trabalhar(), que é mais adequada ao contexto de um professor.

Exercício 172: Sobrescrita de Método na Classe Conta

Enunciado: Crie uma classe Conta com o método depositar() que exibe "Depositando na conta padrão". Crie uma subclasse ContaPoupanca que sobrescreva o método para exibir "Depositando na conta poupança". No método main(), crie objetos de ambas as classes e chame o método depositar().

Código de Solução:

```
class Conta {
    public void depositar() {
        System.out.println("Depositando na conta padrão.");
    }
}
```

```
class ContaPoupanca extends Conta {
    @Override
    public void depositar() {
        System.out.println("Depositando na conta poupança.");
    }
}

public class Main {
    public static void main(String[] args) {
        Conta conta = new Conta();
        ContaPoupanca contaPoupanca = new ContaPoupanca();

        conta.depositar();         // Exibe depósito padrão
        contaPoupanca.depositar();  // Exibe depósito específico na
poupança
    }
}
```

Explicação: A classe Conta possui um método depositar() que exibe uma mensagem genérica de depósito em conta. A subclasse ContaPoupanca sobrescreve esse método para exibir uma mensagem específica relacionada à conta poupança. No método main(), criamos instâncias de Conta e ContaPoupanca e chamamos o método depositar() de ambas. A sobrescrita aqui permite que cada classe apresente um comportamento específico relacionado ao tipo de conta.

Exercício 173: Sobrescrita de Método na Classe Produto

Enunciado: Implemente uma classe Produto com o método calcularPreco() que exibe "Calculando preço base". Crie uma subclasse Eletronico que sobrescreva o método para exibir "Calculando preço com taxa eletrônica". No método main(), crie objetos de ambas as classes e chame o método calcularPreco().

Código de Solução:

```
class Produto {
    public void calcularPreco() {
        System.out.println("Calculando preço base.");
    }
}

class Eletronico extends Produto {
    @Override
    public void calcularPreco() {
```

```
        System.out.println("Calculando preço com taxa eletrônica.");
    }
}

public class Main {
    public static void main(String[] args) {
        Produto produto = new Produto();
        Eletronico eletronico = new Eletronico();

        produto.calcularPreco();      // Exibe cálculo de preço base
        eletronico.calcularPreco();   // Exibe cálculo de preço
específico para eletrônicos
    }
}
```

Explicação: A classe Produto tem um método calcularPreco() que exibe uma mensagem genérica sobre o cálculo de preços. A subclasse Eletronico sobrescreve esse método para exibir uma mensagem específica relacionada a produtos eletrônicos. A sobrescrita permite que cada subclasse modifique o comportamento herdado, sem alterar o comportamento geral da classe pai. No método main(), criamos instâncias de Produto e Eletronico e chamamos o método calcularPreco() para cada uma, demonstrando como a sobrescrita altera o comportamento do método.

Polimorfismo

Exercício 174: Classe Animal com Polimorfismo

Enunciado: Crie uma classe Animal com um método fazerSom(). Implemente duas subclasses: Cachorro e Gato, cada uma sobrescrevendo o método fazerSom() para exibir sons diferentes. No método main(), crie uma lista de animais (misturando Cachorro e Gato) e utilize o polimorfismo para chamar o método fazerSom() em cada objeto da lista.

Código de Solução:

```
import java.util.ArrayList;

class Animal {
    public void fazerSom() {
        System.out.println("O animal está fazendo um som.");
    }
}
```

```
class Cachorro extends Animal {
    @Override
    public void fazerSom() {
        System.out.println("O cachorro está latindo.");
    }
}

class Gato extends Animal {
    @Override
    public void fazerSom() {
        System.out.println("O gato está miando.");
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Animal> animais = new ArrayList<>();
        animais.add(new Cachorro());
        animais.add(new Gato());

        for (Animal animal : animais) {
            animal.fazerSom();
        }
    }
}
```

Explicação: A classe Animal define o método fazerSom(). As subclasses Cachorro e Gato sobrescrevem esse método para exibir sons específicos de cada animal. No método main(), criamos uma lista de Animal e adicionamos objetos de Cachorro e Gato nela. Ao iterar sobre essa lista, o polimorfismo permite que o método correto seja chamado com base no tipo do objeto, exibindo o som adequado para cada animal. Isso demonstra como o polimorfismo permite manipular diferentes tipos de objetos de maneira uniforme.

Exercício 175: Classe Funcionario e Subclasses

Enunciado: Implemente uma classe Funcionario com um método calcularSalario(). Crie duas subclasses, FuncionarioCLT e FuncionarioPJ, que sobrescrevem o método para calcular o salário de forma diferente. No método main(), crie uma lista de funcionários (misturando ambos os tipos) e utilize o polimorfismo para calcular e exibir o salário de cada funcionário.

Código de Solução:

```java
import java.util.ArrayList;

class Funcionario {
    public void calcularSalario() {
        System.out.println("Calculando salário base.");
    }
}

class FuncionarioCLT extends Funcionario {
    @Override
    public void calcularSalario() {
        System.out.println("Salário CLT calculado com base no salário
fixo e benefícios.");
    }
}

class FuncionarioPJ extends Funcionario {
    @Override
    public void calcularSalario() {
        System.out.println("Salário PJ calculado com base nas horas
trabalhadas.");
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Funcionario> funcionarios = new ArrayList<>();
        funcionarios.add(new FuncionarioCLT());
        funcionarios.add(new FuncionarioPJ());

        for (Funcionario funcionario : funcionarios) {
            funcionario.calcularSalario();
        }
    }
}
```

Explicação: Neste exemplo, a classe Funcionario define um método genérico calcularSalario(). As subclasses FuncionarioCLT e FuncionarioPJ sobrescrevem o método com implementações específicas para cada tipo de contratação. No método main(), criamos uma lista de Funcionario contendo objetos de ambas as subclasses. O polimorfismo permite que o método adequado seja chamado, dependendo do tipo real do objeto, sem a necessidade de usar estruturas condicionais complexas.

Exercício 176: Classe Veiculo e Subclasses com Polimorfismo

Enunciado: Crie uma classe Veiculo com um método abastecer(). Implemente as subclasses Carro e Moto, onde cada uma sobrescreve abastecer() de forma diferente (por exemplo, litros diferentes para encher o tanque). No método main(), utilize polimorfismo para criar uma lista de veículos e chame abastecer() para cada veículo.

Código de Solução:

```
import java.util.ArrayList;

class Veiculo {
    public void abastecer() {
        System.out.println("Abastecendo o veículo.");
    }
}

class Carro extends Veiculo {
    @Override
    public void abastecer() {
        System.out.println("Abastecendo o carro com 50 litros de
combustível.");
    }
}

class Moto extends Veiculo {
    @Override
    public void abastecer() {
        System.out.println("Abastecendo a moto com 15 litros de
combustível.");
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Veiculo> veiculos = new ArrayList<>();
        veiculos.add(new Carro());
        veiculos.add(new Moto());

        for (Veiculo veiculo : veiculos) {
            veiculo.abastecer();
        }
    }
}
```

Explicação: A classe Veiculo define um método genérico abastecer(), enquanto as subclasses Carro e Moto fornecem implementações específicas para seus respectivos tipos

de veículos. No método main(), criamos uma lista de veículos contendo objetos das duas subclasses. O polimorfismo permite que o método apropriado seja chamado para cada tipo de veículo, sem a necessidade de verificar o tipo explicitamente. Isso demonstra como o polimorfismo facilita a manipulação de diferentes tipos de objetos de forma uniforme.

Exercício 177: Classe Forma e Subclasses com Cálculo de Área

Enunciado: Implemente uma classe abstrata Forma com um método abstrato calcularArea(). Crie as subclasses Circulo e Retangulo, cada uma com a implementação do método calcularArea(). No método main(), crie uma lista de formas e, usando polimorfismo, calcule e exiba a área de cada uma.

Código de Solução:

```
import java.util.ArrayList;

abstract class Forma {
    public abstract double calcularArea();
}

class Circulo extends Forma {
    private double raio;

    public Circulo(double raio) {
        this.raio = raio;
    }

    @Override
    public double calcularArea() {
        return Math.PI * raio * raio;
    }
}

class Retangulo extends Forma {
    private double largura;
    private double altura;

    public Retangulo(double largura, double altura) {
        this.largura = largura;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return largura * altura;
```

```
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Forma> formas = new ArrayList<>();
        formas.add(new Circulo(5));
        formas.add(new Retangulo(4, 6));

        for (Forma forma : formas) {
            System.out.println("Área: " + forma.calcularArea());
        }
    }
}
```

Explicação: Neste exercício, a classe Forma é abstrata e define um método abstrato calcularArea(). As subclasses Circulo e Retangulo implementam esse método para calcular a área de suas respectivas formas. No método main(), criamos uma lista de Forma contendo objetos de Circulo e Retangulo. O polimorfismo permite que o método correto seja chamado para cada tipo de forma, sem a necessidade de saber explicitamente qual tipo de forma estamos lidando. Isso exemplifica o uso de classes abstratas e polimorfismo para implementar comportamentos específicos em subclasses.

Exercício 178: Interface Pagamento e Implementações

Enunciado: Crie uma interface Pagamento com um método processarPagamento(). Implemente duas classes que implementam essa interface: PagamentoCartao e PagamentoBoleto. No método main(), crie uma lista de diferentes tipos de pagamento e utilize o polimorfismo para processar cada um deles chamando o método processarPagamento().

Código de Solução:

```
import java.util.ArrayList;

interface Pagamento {
    void processarPagamento();
}

class PagamentoCartao implements Pagamento {
    @Override
    public void processarPagamento() {
        System.out.println("Processando pagamento via cartão de
```

```
crédito.");
    }
}

class PagamentoBoleto implements Pagamento {
    @Override
    public void processarPagamento() {
        System.out.println("Processando pagamento via boleto
bancário.");
    }
}

public class Main {
    public static void main(String[] args) {
        ArrayList<Pagamento> pagamentos = new ArrayList<>();
        pagamentos.add(new PagamentoCartao());
        pagamentos.add(new PagamentoBoleto());

        for (Pagamento pagamento : pagamentos) {
            pagamento.processarPagamento();
        }
    }
}
```

Explicação: Aqui, definimos uma interface Pagamento com o método processarPagamento(). As classes PagamentoCartao e PagamentoBoleto implementam essa interface e fornecem suas próprias versões do método `

Conclusão do Capítulo 8: Herança e Polimorfismo

Neste capítulo, vimos dois conceitos fundamentais da Programação Orientada a Objetos (POO): herança e polimorfismo.

Aprendemos como a herança permite o reaproveitamento de código, criando hierarquias entre classes, e como a sobrescrita de métodos possibilita a personalização de comportamento nas subclasses.

O polimorfismo complementa esses conceitos, permitindo o tratamento uniforme de objetos de diferentes tipos, simplificando a estrutura do código e promovendo sua flexibilidade.

Introdução ao Capítulo 9: Manipulação de Strings

No Capítulo 9, exploraremos a manipulação de strings, uma das tarefas mais comuns na programação.

As strings representam sequências de caracteres e são amplamente utilizadas em diversas aplicações, desde processamento de textos até interação com o usuário. Começaremos com a manipulação básica, como concatenar, comparar e extrair partes de strings.

Em seguida, abordaremos os métodos da classe String, que oferecem uma variedade de ferramentas poderosas para trabalhar com textos de forma eficiente. Ao dominar esses conceitos, você será capaz de lidar com manipulações textuais de forma precisa e eficaz.

Manipulação básica de Strings

Exercício 179: Exibir String

Enunciado: Escreva um programa que receba uma string do usuário e exiba a mesma string no console.

Código de Solução:

```
import java.util.Scanner;

public class ExibirString {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.println("Você digitou: " + texto);
    }
}
```

Explicação: O programa utiliza a classe Scanner para receber uma string do usuário e, em seguida, exibe essa mesma string no console. O método nextLine() é utilizado para capturar a entrada completa do usuário, incluindo espaços.

Exercício 180: Comprimento da String

Enunciado: Crie um programa que receba uma string e exiba o número de caracteres que ela contém.

Código de Solução:

```
import java.util.Scanner;

public class ComprimentoString {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.println("Comprimento da string: " + texto.length());
    }
}
```

Explicação: Este programa usa o método length() da classe String para contar e exibir o número de caracteres na string fornecida pelo usuário. A string capturada é analisada e o número total de caracteres, incluindo espaços, é exibido.

Exercício 181: Concatenar Strings

Enunciado: Implemente um programa que receba duas strings do usuário e exiba a concatenação delas.

Código de Solução:

```
import java.util.Scanner;

public class ConcatenarStrings {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite a primeira string: ");
        String string1 = scanner.nextLine();

        System.out.print("Digite a segunda string: ");
        String string2 = scanner.nextLine();

        String resultado = string1 + string2;
        System.out.println("Resultado da concatenação: " + resultado);
    }
}
```

Explicação: O programa recebe duas strings do usuário e utiliza o operador + para concatená-las, criando uma nova string resultante. O operador de concatenação une as duas strings em uma só e exibe o resultado no console.

Exercício 182: Comparar Strings

Enunciado: Escreva um programa que receba duas strings e compare se elas são iguais, exibindo uma mensagem no console.

Código de Solução:

```
import java.util.Scanner;

public class CompararStrings {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite a primeira string: ");
        String string1 = scanner.nextLine();

        System.out.print("Digite a segunda string: ");
        String string2 = scanner.nextLine();

        if (string1.equals(string2)) {
            System.out.println("As strings são iguais.");
        } else {
            System.out.println("As strings são diferentes.");
        }
    }
}
```

Explicação: O método equals() é utilizado para comparar o conteúdo das duas strings. Se ambas as strings possuem o mesmo conteúdo, a comparação retorna true, e o programa exibe que as strings são iguais. Caso contrário, o programa informa que são diferentes.

Exercício 183: Converter para Maiúsculas

Enunciado: Crie um programa que receba uma string e exiba a versão da string convertida para letras maiúsculas.

Código de Solução:

```java
import java.util.Scanner;

public class ConverterMaiusculas {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        String textoMaiusculas = texto.toUpperCase();
        System.out.println("String em maiúsculas: " + textoMaiusculas);
    }
}
```

Explicação: O método toUpperCase() converte todos os caracteres da string para letras maiúsculas. O programa recebe uma string do usuário, aplica a conversão e exibe o resultado no console.

Exercício 184: Substring

Enunciado: Escreva um programa que receba uma string e um número inteiro do usuário. O programa deve exibir os primeiros N caracteres da string, onde N é o número fornecido.

Código de Solução:

```java
import java.util.Scanner;

public class SubstringExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite um número inteiro: ");
        int n = scanner.nextInt();

        String substring = texto.substring(0, n);
        System.out.println("Primeiros " + n + " caracteres: " +
substring);
    }
}
```

Explicação: O método substring() é utilizado para extrair os primeiros n caracteres da string. O programa recebe um número inteiro do usuário que indica quantos caracteres exibir, e a string resultante é extraída e exibida no console.

Exercício 185: Remover Espaços

Enunciado: Implemente um programa que receba uma string do usuário e remova todos os espaços em branco, exibindo o resultado.

Código de Solução:

```
import java.util.Scanner;

public class RemoverEspacos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        String textoSemEspacos = texto.replace(" ", "");
        System.out.println("String sem espaços: " + textoSemEspacos);
    }
}
```

Explicação: O método replace() substitui todos os espaços em branco (" ") por uma string vazia (""), removendo-os. O programa processa a string fornecida pelo usuário e exibe a versão sem espaços.

Exercício 186: Contar Ocorrências de um Caractere

Enunciado: Crie um programa que receba uma string e um caractere. O programa deve contar e exibir quantas vezes o caractere aparece na string.

Código de Solução:

```
import java.util.Scanner;

public class ContarOcorrencias {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite um caractere: ");
        char caractere = scanner.next().charAt(0);

        int contador = 0;
        for (int i = 0; i < texto.length(); i++) {
            if (texto.charAt(i) == caractere) {
                contador++;
            }
        }

        System.out.println("O caractere '" + caractere + "' aparece " +
contador + " vezes.");
    }
}
```

Explicação: O programa percorre cada caractere da string utilizando um laço for e compara com o caractere fornecido pelo usuário. A cada vez que uma correspondência é encontrada, o contador é incrementado. No final, o número total de ocorrências é exibido.

Exercício 187: Inverter String

Enunciado: Escreva um programa que receba uma string e exiba a versão invertida da string.

Código de Solução:

```
import java.util.Scanner;

public class InverterString {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        StringBuilder textoInvertido = new StringBuilder(texto);
        textoInvertido.reverse();

        System.out.println("String invertida: " + textoInvertido);
    }
}
```

Explicação: Utilizamos a classe StringBuilder que possui o método reverse() para inverter a string. Após receber a string do usuário, o método inverte sua ordem e exibe o resultado. A classe StringBuilder é ideal para manipulações de strings quando a mutabilidade é necessária.

Exercício 188: Substituir Caracteres

Enunciado: Implemente um programa que receba uma string e substitua todas as ocorrências de um caractere por outro caractere, fornecidos pelo usuário.

Código de Solução:

```
import java.util.Scanner;

public class SubstituirCaracteres {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite o caractere a ser substituído: ");
        char antigo = scanner.next().charAt(0);

        System.out.print("Digite o novo caractere: ");
        char novo = scanner.next().charAt(0);

        String resultado = texto.replace(antigo, novo);

        System.out.println("Resultado após substituição: " + resultado);
    }
}
```

Explicação: O método replace() da classe String é utilizado para substituir todas as ocorrências de um caractere por outro. O programa recebe uma string e dois caracteres (um para substituir e outro para ocupar o lugar do primeiro). O método percorre a string e substitui todas as ocorrências do caractere antigo pelo novo. Ao final, a string resultante é exibida no console.

Métodos da Classe String

Exercício 189: Usando o Método charAt()

Enunciado: Escreva um programa que receba uma string e um número inteiro do usuário. O programa deve exibir o caractere da string que está na posição indicada pelo número.

Código de Solução:

```
import java.util.Scanner;

public class CharAtExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite um número inteiro: ");
        int indice = scanner.nextInt();

        if (indice >= 0 && indice < texto.length()) {
            System.out.println("Caractere na posição " + indice + ": " +
texto.charAt(indice));
        } else {
            System.out.println("Índice fora do intervalo válido.");
        }
    }
}
```

Explicação: O método charAt() é utilizado para obter o caractere em uma posição específica da string. O programa solicita uma string e um número (índice) ao usuário, e exibe o caractere correspondente. O código verifica se o índice está dentro do intervalo válido (0 até texto.length() - 1) antes de acessar o caractere.

Exercício 190: Usando o Método equalsIgnoreCase()

Enunciado: Implemente um programa que receba duas strings do usuário e verifique se elas são iguais, ignorando a diferença entre maiúsculas e minúsculas.

Código de Solução:

```java
import java.util.Scanner;

public class EqualsIgnoreCaseExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a primeira string: ");
        String string1 = scanner.nextLine();

        System.out.print("Digite a segunda string: ");
        String string2 = scanner.nextLine();

        if (string1.equalsIgnoreCase(string2)) {
            System.out.println("As strings são iguais (ignorando
maiúsculas/minúsculas).");
        } else {
            System.out.println("As strings são diferentes.");
        }
    }
}
```

Explicação: O método equalsIgnoreCase() é usado para comparar duas strings, ignorando as diferenças entre letras maiúsculas e minúsculas. O programa solicita duas strings ao usuário e verifica se elas são iguais independentemente de como as letras estão capitalizadas.

Exercício 191: Usando o Método contains()

Enunciado: Escreva um programa que receba uma string e uma palavra. O programa deve verificar se a palavra está contida na string e exibir uma mensagem correspondente.

Código de Solução:

```java
import java.util.Scanner;

public class ContainsExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite uma palavra: ");
        String palavra = scanner.nextLine();
```

```
        if (texto.contains(palavra)) {
            System.out.println("A palavra \"" + palavra + "\" está
contida na string.");
        } else {
            System.out.println("A palavra \"" + palavra + "\" não está
contida na string.");
        }
    }
}
```

Explicação: O método contains() é utilizado para verificar se uma string contém outra string. O programa solicita uma string e uma palavra ao usuário, e verifica se a palavra está presente na string principal. Se a palavra for encontrada, uma mensagem é exibida informando o sucesso da busca.

Exercício 192: Usando o Método indexOf()

Enunciado: Implemente um programa que receba uma string e um caractere. O programa deve exibir a posição da primeira ocorrência do caractere na string, ou uma mensagem informando que o caractere não foi encontrado.

Código de Solução:

```
import java.util.Scanner;

public class IndexOfExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite um caractere: ");
        char caractere = scanner.next().charAt(0);

        int posicao = texto.indexOf(caractere);

        if (posicao != -1) {
            System.out.println("O caractere '" + caractere + "' aparece
pela primeira vez na posição: " + posicao);
        } else {
            System.out.println("O caractere '" + caractere + "' não foi
```

```
encontrado na string.");
        }
    }
}
```

Explicação: O método indexOf() retorna a posição da primeira ocorrência de um caractere em uma string. Se o caractere não for encontrado, o método retorna -1. O programa captura uma string e um caractere do usuário, e exibe a posição da primeira ocorrência do caractere na string.

Exercício 193: Usando o Método startsWith()

Enunciado: Crie um programa que receba uma string e verifique se ela começa com uma determinada palavra, fornecida pelo usuário. Exiba uma mensagem indicando o resultado.

Código de Solução:

```
import java.util.Scanner;

public class StartsWithExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite a palavra inicial: ");
        String palavra = scanner.nextLine();

        if (texto.startsWith(palavra)) {
            System.out.println("A string começa com \"" + palavra +
"\".");
        } else {
            System.out.println("A string não começa com \"" + palavra +
"\".");
        }
    }
}
```

Explicação: O método startsWith() verifica se uma string começa com a palavra fornecida. O programa recebe uma string e uma palavra, e verifica se a string começa com essa palavra. Se a verificação for bem-sucedida, uma mensagem é exibida no console.

Exercício 194: Usando o Método endsWith()

Enunciado: Desenvolva um programa que receba uma string e verifique se ela termina com uma determinada palavra, fornecida pelo usuário. Exiba o resultado no console.

Código de Solução:

```
import java.util.Scanner;

public class EndsWithExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite a palavra final: ");
        String palavra = scanner.nextLine();

        if (texto.endsWith(palavra)) {
            System.out.println("A string termina com \"" + palavra +
"\".");
        } else {
            System.out.println("A string não termina com \"" + palavra +
"\".");
        }
    }
}
```

Explicação: O método endsWith() é utilizado para verificar se uma string termina com a palavra fornecida pelo usuário. O programa captura uma string e uma palavra, e exibe uma mensagem indicando se a string termina com essa palavra ou não.

Exercício 195: Usando o Método replaceAll()

Enunciado: Implemente um programa que receba uma string e substitua todas as ocorrências de uma palavra por outra, ambas fornecidas pelo usuário.

Código de Solução:

```
import java.util.Scanner;
```

```java
public class ReplaceAllExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        System.out.print("Digite a palavra a ser substituída: ");
        String antiga = scanner.nextLine();

        System.out.print("Digite a nova palavra: ");
        String nova = scanner.nextLine();

        String resultado = texto.replaceAll(antiga, nova);
        System.out.println("Resultado: " + resultado);
    }
}
```

Explicação: O método replaceAll() substitui todas as ocorrências de uma palavra ou expressão em uma string por outra. O programa solicita uma string e duas palavras: uma a ser substituída e outra para ser colocada no lugar. Em seguida, a string resultante com as substituições é exibida.

Exercício 196: Usando o Método split()

Enunciado: Escreva um programa que receba uma string e separe as palavras usando espaços como delimitadores. Exiba cada palavra em uma linha separada.

Código de Solução:

```java
import java.util.Scanner;

public class SplitExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        String[] palavras = texto.split(" ");

        System.out.println("Palavras separadas:");
        for (String palavra : palavras) {
```

```
            System.out.println(palavra);
        }
    }
}
```

Explicação: O método split() divide a string em partes com base no delimitador fornecido (neste caso, um espaço " "). O programa captura uma string e usa o método split() para dividir a string em um array de palavras. Cada palavra é exibida em uma linha separada.

Exercício 197: Usando o Método trim()

Enunciado: Implemente um programa que receba uma string com espaços em excesso no início e no fim, e exiba a string "limpa", sem esses espaços, utilizando o método trim().

Código de Solução:

```
import java.util.Scanner;

public class TrimExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string com espaços em excesso: ");
        String texto = scanner.nextLine();

        String textoLimpo = texto.trim();
        System.out.println("String sem espaços em excesso: \"" +
textoLimpo + "\"");
    }
}
```

Explicação: O método trim() remove os espaços em branco no início e no final de uma string. O programa recebe uma string do usuário, aplica o método trim() e exibe a string sem os espaços em excesso.

Exercício 198: Usando o Método compareTo()

Enunciado: Escreva um programa que compare duas strings fornecidas pelo usuário e exiba o resultado da comparação lexicográfica entre elas.

Código de Solução:

```java
import java.util.Scanner;

public class CompareToExemplo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a primeira string: ");
        String string1 = scanner.nextLine();

        System.out.print("Digite a segunda string: ");
        String string2 = scanner.nextLine();

        int resultado = string1.compareTo(string2);

        if (resultado < 0) {
            System.out.println("\"" + string1 + "\" vem antes de \"" +
string2 + "\" lexicograficamente.");
        } else if (resultado > 0) {
            System.out.println("\"" + string1 + "\" vem depois de \"" +
string2 + "\" lexicograficamente.");
        } else {
            System.out.println("As strings são iguais
lexicograficamente.");
        }
    }
}
```

Explicação: O método compareTo() compara duas strings lexicograficamente (com base na ordem alfabética). O programa solicita duas strings ao usuário, realiza a comparação e exibe se uma string vem antes, depois ou se as duas são iguais.

Conclusão do Capítulo 9: Manipulação de Strings

Neste capítulo, exploramos a manipulação de strings, abordando tanto operações básicas quanto o uso dos métodos da classe String.

Na parte de manipulação básica, aprendemos a lidar com tarefas essenciais, como concatenar, comparar e modificar strings. Já nos métodos da classe String, vimos como utilizar funcionalidades poderosas para operações mais avançadas, como busca, substituição, e divisão de strings.
Ao dominar essas técnicas, você está apto a manipular e processar textos de maneira eficiente, essencial para uma grande variedade de aplicações na programação.

Introdução ao Capítulo 10: Desafios

Neste capítulo, você enfrentará desafios práticos que irão testar os conhecimentos adquiridos ao longo do curso.

Os exercícios serão mais complexos e exigirão uma combinação de habilidades, incluindo lógica de programação, manipulação de dados, e conceitos de orientação a objetos.

Estes desafios têm o objetivo de consolidar o que foi aprendido, incentivando o pensamento crítico e a resolução de problemas de forma criativa e eficiente.

Prepare-se para aplicar suas habilidades de forma prática e desafiadora, expandindo ainda mais seu domínio da programação.

Desafio 1: Calculadora Completa

Enunciado: Crie uma calculadora que permita ao usuário realizar operações matemáticas básicas (soma, subtração, multiplicação e divisão). O usuário deve poder inserir dois números e escolher a operação desejada.

Solução:

```
import java.util.Scanner;

public class CalculadoraCompleta {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite o primeiro número: ");
        double num1 = scanner.nextDouble();

        System.out.print("Digite o segundo número: ");
        double num2 = scanner.nextDouble();

        System.out.println("Escolha a operação: ");
        System.out.println("1: Soma");
        System.out.println("2: Subtração");
        System.out.println("3: Multiplicação");
        System.out.println("4: Divisão");
        int opcao = scanner.nextInt();

        double resultado;
```

```
        switch (opcao) {
            case 1:
                resultado = num1 + num2;
                System.out.println("Resultado: " + resultado);
                break;
            case 2:
                resultado = num1 - num2;
                System.out.println("Resultado: " + resultado);
                break;
            case 3:
                resultado = num1 * num2;
                System.out.println("Resultado: " + resultado);
                break;
            case 4:
                if (num2 != 0) {
                    resultado = num1 / num2;
                    System.out.println("Resultado: " + resultado);
                } else {
                    System.out.println("Erro: Divisão por zero.");
                }
                break;
            default:
                System.out.println("Operação inválida.");
        }
    }
}
```

Explicação: Este programa permite ao usuário inserir dois números e escolher a operação desejada (soma, subtração, multiplicação ou divisão). Utilizamos um switch para executar a operação com base na escolha do usuário. Para a divisão, verificamos se o segundo número é diferente de zero, evitando erro de divisão por zero.

Desafio 2: Verificador de Palíndromo

Enunciado: Implemente um programa que receba uma palavra ou frase e verifique se ela é um palíndromo (se lê da mesma forma de frente para trás).

Solução:

```
import java.util.Scanner;

public class VerificadorPalindromo {
    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma palavra ou frase: ");
        String texto = scanner.nextLine().replaceAll("\\s+",
"").toLowerCase();

        String invertido = new
StringBuilder(texto).reverse().toString();

        if (texto.equals(invertido)) {
            System.out.println("É um palíndromo.");
        } else {
            System.out.println("Não é um palíndromo.");
        }
    }
}
```

Explicação: O programa remove os espaços em branco da string usando o método replaceAll() e converte a string para letras minúsculas para garantir que a verificação não seja sensível a maiúsculas/minúsculas. Em seguida, a string é invertida utilizando o StringBuilder. Se a string original for igual à string invertida, ela é um palíndromo.

Desafio 3: Contador de Palavras

Enunciado: Crie um programa que receba uma string e conte quantas palavras existem nela.

Solução:

```
import java.util.Scanner;

public class ContadorDePalavras {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma frase: ");
        String frase = scanner.nextLine().trim();

        String[] palavras = frase.split("\\s+");
        System.out.println("A frase contém " + palavras.length + "
palavras.");
    }
}
```

Explicação: Este programa usa o método split() para dividir a string com base nos espaços em branco, considerando um ou mais espaços consecutivos. O método trim() remove os espaços extras no início e no fim da string, garantindo que as palavras sejam contadas corretamente. O número de palavras é exibido com base no comprimento do array gerado.

Desafio 4: Jogo da Adivinhação

Enunciado: Desenvolva um jogo em que o programa escolhe um número aleatório entre 1 e 100, e o usuário deve tentar adivinhar esse número. O programa deve fornecer dicas se o palpite for maior ou menor que o número secreto.

Solução:

```
import java.util.Scanner;
import java.util.Random;

public class JogoAdivinhacao {
    public static void main(String[] args) {
        Random random = new Random();
        Scanner scanner = new Scanner(System.in);

        int numeroSecreto = random.nextInt(100) + 1;
        int palpite;
        boolean acertou = false;

        System.out.println("Tente adivinhar o número entre 1 e 100!");

        while (!acertou) {
            System.out.print("Digite seu palpite: ");
            palpite = scanner.nextInt();

            if (palpite == numeroSecreto) {
                System.out.println("Parabéns! Você acertou!");
                acertou = true;
            } else if (palpite < numeroSecreto) {
                System.out.println("O número é maior.");
            } else {
                System.out.println("O número é menor.");
            }
        }
    }
}
```

Explicação: O programa gera um número aleatório entre 1 e 100 usando a classe Random. Em seguida, o usuário faz palpites até acertar o número secreto. O programa dá dicas se o palpite é maior ou menor que o número gerado. A execução continua até que o jogador acerte.

Desafio 5: Validador de Senha

Enunciado: Implemente um validador de senha que verifique se a senha do usuário atende aos seguintes critérios: no mínimo 8 caracteres, contém pelo menos um número, uma letra maiúscula, uma letra minúscula e um caractere especial.

Solução:

```
import java.util.Scanner;

public class ValidadorSenha {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma senha: ");
        String senha = scanner.nextLine();

        if (validarSenha(senha)) {
            System.out.println("Senha válida.");
        } else {
            System.out.println("Senha inválida. A senha deve ter no
mínimo 8 caracteres, conter uma letra maiúscula, uma minúscula, um
número e um caractere especial.");
        }
    }

    public static boolean validarSenha(String senha) {
        if (senha.length() < 8) return false;

        boolean temMaiuscula = false, temMinuscula = false, temNumero =
false, temEspecial = false;

        for (char c : senha.toCharArray()) {
            if (Character.isUpperCase(c)) temMaiuscula = true;
            if (Character.isLowerCase(c)) temMinuscula = true;
            if (Character.isDigit(c)) temNumero = true;
            if (!Character.isLetterOrDigit(c)) temEspecial = true;
        }
```

```
        return temMaiuscula && temMinuscula && temNumero && temEspecial;
    }
}
```

Explicação: Este programa valida a senha verificando se ela contém pelo menos 8 caracteres, uma letra maiúscula, uma minúscula, um número e um caractere especial. A verificação é feita através de um laço que percorre cada caractere da senha e atualiza variáveis booleanas para garantir que a senha atenda a todos os critérios.

Desafio 6: Ordenação de Números

Enunciado: Crie um programa que receba uma lista de números do usuário e os ordene em ordem crescente.

Solução:

```
import java.util.Arrays;
import java.util.Scanner;

public class OrdenacaoNumeros {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a quantidade de números: ");
        int n = scanner.nextInt();

        int[] numeros = new int[n];

        System.out.println("Digite os números:");
        for (int i = 0; i < n; i++) {
            numeros[i] = scanner.nextInt();
        }

        Arrays.sort(numeros);

        System.out.println("Números em ordem crescente: " +
Arrays.toString(numeros));
    }
}
```

Explicação: Este programa utiliza o método Arrays.sort() para ordenar os números inseridos pelo usuário. O usuário especifica a quantidade de números que deseja ordenar, insere-os, e o programa os exibe em ordem crescente. A função Arrays.toString() é usada para converter o array em uma string legível.

Desafio 7: Contagem de Vogais

Enunciado: Escreva um programa que receba uma string e conte quantas vogais existem nela.

Solução:

```
import java.util.Scanner;

public class ContagemVogais {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine().toLowerCase();

        int contagemVogais = 0;
        for (char c : texto.toCharArray()) {
            if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c ==
'u') {
                contagemVogais++;
            }
        }

        System.out.println("Número de vogais: " + contagemVogais);
    }
}
```

Explicação: Este programa recebe uma string e a converte para letras minúsculas para facilitar a contagem das vogais. Ele percorre a string com um laço for, verificando se cada caractere é uma vogal. Se for, o contador de vogais é incrementado. No final, o programa exibe o número total de vogais.

Desafio 8: Fatorial Recursivo

Enunciado: Desenvolva uma função recursiva que calcule o fatorial de um número fornecido pelo usuário.

Solução:

```java
import java.util.Scanner;

public class FatorialRecursivo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número: ");
        int numero = scanner.nextInt();

        System.out.println("O fatorial de " + numero + " é: " +
fatorial(numero));
    }

    public static int fatorial(int n) {
        if (n == 0) {
            return 1;
        }
        return n * fatorial(n - 1);
    }
}
```

Explicação: A função recursiva fatorial() chama a si mesma até que o valor de n seja 0, quando o cálculo é finalizado. O fatorial de 0 é definido como 1. A cada chamada recursiva, o número é multiplicado pelo valor de fatorial(n-1), criando a recursão necessária para calcular o fatorial.

Desafio 9: Conversor de Temperatura

Enunciado: Implemente um conversor de temperatura que converta valores entre Celsius, Fahrenheit e Kelvin. O usuário deve escolher a unidade de origem e destino.

Solução:

```java
import java.util.Scanner;

public class ConversorTemperatura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite a temperatura: ");
        double temperatura = scanner.nextDouble();

        System.out.println("Escolha a unidade de origem:");
```

```
        System.out.println("1: Celsius");
        System.out.println("2: Fahrenheit");
        System.out.println("3: Kelvin");
        int origem = scanner.nextInt();

        System.out.println("Escolha a unidade de destino:");
        System.out.println("1: Celsius");
        System.out.println("2: Fahrenheit");
        System.out.println("3: Kelvin");
        int destino = scanner.nextInt();

        double resultado = converterTemperatura(temperatura, origem,
destino);
        System.out.println("Temperatura convertida: " + resultado);
    }

    public static double converterTemperatura(double temp, int origem,
int destino) {
        if (origem == destino) {
            return temp;
        }
        // Convertendo para Celsius
        if (origem == 2) { // Fahrenheit para Celsius
            temp = (temp - 32) * 5/9;
        } else if (origem == 3) { // Kelvin para Celsius
            temp = temp - 273.15;
        }
        // Convertendo de Celsius para unidade destino
        if (destino == 2) { // Celsius para Fahrenheit
            return (temp * 9/5) + 32;
        } else if (destino == 3) { // Celsius para Kelvin
            return temp + 273.15;
        }
        return temp; // Se o destino for Celsius
    }
}
```

Explicação: O programa permite ao usuário escolher a unidade de temperatura de origem e destino. A função converterTemperatura() realiza a conversão para a unidade correta, convertendo primeiro a temperatura para Celsius (se necessário) e, em seguida, para a unidade de destino. Isso garante que todas as combinações de conversão sejam tratadas corretamente.

Desafio 10: Gerador de Senhas Aleatórias

Enunciado: Crie um programa que gere senhas aleatórias de 8 a 16 caracteres, incluindo letras, números e símbolos.

Solução:

```
import java.security.SecureRandom;
import java.util.Scanner;

public class GeradorSenhaAleatoria {
    private static final String CARACTERES =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*(
)_+-=<>?";

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        SecureRandom random = new SecureRandom();

        System.out.print("Digite o comprimento desejado para a senha
(entre 8 e 16): ");
        int comprimento = scanner.nextInt();

        if (comprimento < 8 || comprimento > 16) {
            System.out.println("Comprimento inválido. Deve ser entre 8 e
16.");
            return;
        }

        StringBuilder senha = new StringBuilder(comprimento);
        for (int i = 0; i < comprimento; i++) {

senha.append(CARACTERES.charAt(random.nextInt(CARACTERES.length())));
        }

        System.out.println("Senha gerada: " + senha.toString());
    }
}
```

Explicação: Este programa utiliza a classe SecureRandom para gerar números aleatórios de maneira segura. A senha é gerada concatenando caracteres aleatórios a partir de um conjunto de letras maiúsculas, minúsculas, números e símbolos. O usuário pode escolher o comprimento da senha, entre 8 e 16 caracteres, e a senha gerada é exibida no final.

Desafio 11: Cálculo de Média de Notas

Enunciado: Desenvolva um programa que receba as notas de um aluno e calcule sua média. O programa deve determinar se o aluno está aprovado (média ≥ 7), em recuperação (média entre 5 e 6.9) ou reprovado (média < 5).

Solução:

```java
import java.util.Scanner;

public class MediaNotas {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double soma = 0;
        int quantidadeNotas;

        System.out.print("Quantas notas você deseja inserir? ");
        quantidadeNotas = scanner.nextInt();

        for (int i = 0; i < quantidadeNotas; i++) {
            System.out.print("Digite a nota " + (i + 1) + ": ");
            soma += scanner.nextDouble();
        }

        double media = soma / quantidadeNotas;

        System.out.println("Média: " + media);

        if (media >= 7) {
            System.out.println("Aluno aprovado.");
        } else if (media >= 5 && media < 7) {
            System.out.println("Aluno em recuperação.");
        } else {
            System.out.println("Aluno reprovado.");
        }
    }
}
```

Explicação: O programa solicita o número de notas e, em seguida, calcula a média. Com base na média obtida, ele classifica o aluno como "aprovado", "em recuperação" ou "reprovado". O laço for é usado para coletar as notas, somá-las e, depois, calcular a média.

Desafio 12: Verificador de Número Primo

Enunciado: Implemente um programa que receba um número inteiro e verifique se ele é primo.

Solução:

```java
import java.util.Scanner;

public class VerificadorPrimo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int numero = scanner.nextInt();

        if (isPrimo(numero)) {
            System.out.println(numero + " é um número primo.");
        } else {
            System.out.println(numero + " não é um número primo.");
        }
    }

    public static boolean isPrimo(int numero) {
        if (numero < 2) return false;

        for (int i = 2; i <= Math.sqrt(numero); i++) {
            if (numero % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Explicação: Este programa verifica se um número é primo percorrendo todos os números até a raiz quadrada do número em questão, para otimizar o desempenho. Se o número for divisível por qualquer outro número além de 1 e ele mesmo, ele não é primo.

Desafio 13: Jogo Pedra, Papel e Tesoura

Enunciado: Crie uma versão digital do jogo Pedra, Papel e Tesoura, em que o usuário joga contra o computador. O computador deve fazer escolhas aleatórias e o programa deve determinar o vencedor de cada rodada.

Solução:

```java
import java.util.Random;
import java.util.Scanner;

public class PedraPapelTesoura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Random random = new Random();

        String[] opcoes = {"Pedra", "Papel", "Tesoura"};

        System.out.println("Escolha: 0 = Pedra, 1 = Papel, 2 =
Tesoura");
        int escolhaUsuario = scanner.nextInt();

        int escolhaComputador = random.nextInt(3);

        System.out.println("Você escolheu: " + opcoes[escolhaUsuario]);
        System.out.println("Computador escolheu: " +
opcoes[escolhaComputador]);

        if (escolhaUsuario == escolhaComputador) {
            System.out.println("Empate!");
        } else if ((escolhaUsuario == 0 && escolhaComputador == 2) ||
                   (escolhaUsuario == 1 && escolhaComputador == 0) ||
                   (escolhaUsuario == 2 && escolhaComputador == 1)) {
            System.out.println("Você venceu!");
        } else {
            System.out.println("Você perdeu!");
        }
    }
}
```

Explicação: Neste jogo, o usuário faz sua escolha (Pedra, Papel ou Tesoura), enquanto o computador escolhe aleatoriamente usando a classe Random. O programa compara as duas escolhas e determina o vencedor de acordo com as regras do jogo.

Desafio 14: Conversor de Moedas

Enunciado: Desenvolva um conversor de moedas que permita ao usuário converter valores entre diferentes moedas (por exemplo, de real para dólar, de euro para iene, etc.), utilizando taxas de câmbio fixas.

Solução:

```java
import java.util.Scanner;

public class ConversorMoedas {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Conversor de Moedas");
        System.out.println("1: Real para Dólar");
        System.out.println("2: Real para Euro");
        System.out.println("3: Dólar para Real");
        System.out.println("4: Euro para Real");
        System.out.print("Escolha a conversão: ");
        int opcao = scanner.nextInt();

        System.out.print("Digite o valor: ");
        double valor = scanner.nextDouble();

        double resultado = converterMoeda(opcao, valor);
        System.out.println("Valor convertido: " + resultado);
    }

    public static double converterMoeda(int opcao, double valor) {
        switch (opcao) {
            case 1: // Real para Dólar
                return valor / 5.20;
            case 2: // Real para Euro
                return valor / 5.90;
            case 3: // Dólar para Real
                return valor * 5.20;
            case 4: // Euro para Real
                return valor * 5.90;
            default:
                System.out.println("Opção inválida");
                return 0;
        }
    }
}
```

Explicação: Este conversor de moedas utiliza taxas de câmbio fixas para realizar as conversões entre real, dólar e euro. O usuário escolhe a conversão e insere o valor que deseja converter, e o programa aplica a taxa de câmbio correspondente.

Desafio 15: Contagem Regressiva

Enunciado: Implemente um programa que faça uma contagem regressiva de um número fornecido pelo usuário até 0, exibindo cada número no console.

Solução:

```
import java.util.Scanner;

public class ContagemRegressiva {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número para iniciar a contagem
regressiva: ");
        int numero = scanner.nextInt();

        for (int i = numero; i >= 0; i--) {
            System.out.println(i);
        }

        System.out.println("Contagem regressiva finalizada!");
    }
}
```

Explicação: O programa solicita um número do usuário e faz uma contagem regressiva de forma decrescente até zero usando um laço for. Cada número é exibido no console, e uma mensagem de conclusão é exibida ao final da contagem.

Desafio 16: Tabuada Automática

Enunciado: Crie um programa que exiba a tabuada de um número fornecido pelo usuário, de 1 a 10.

Solução:

```
import java.util.Scanner;

public class TabuadaAutomatica {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número para exibir sua tabuada: ");
        int numero = scanner.nextInt();
```

```
        System.out.println("Tabuada do " + numero + ":");
        for (int i = 1; i <= 10; i++) {
            System.out.println(numero + " x " + i + " = " + (numero *
i));
        }
    }
}
```

Explicação: O programa solicita um número ao usuário e exibe sua tabuada de 1 a 10. Utiliza-se um laço for para multiplicar o número fornecido pelo usuário pelos números de 1 a 10, exibindo o resultado no console.

Desafio 17: Soma dos Dígitos de um Número

Enunciado: Desenvolva um programa que receba um número inteiro e calcule a soma dos dígitos desse número.

Solução:

```
import java.util.Scanner;

public class SomaDigitos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite um número inteiro: ");
        int numero = scanner.nextInt();
        int soma = 0;

        while (numero != 0) {
            soma += numero % 10;
            numero /= 10;
        }

        System.out.println("Soma dos dígitos: " + soma);
    }
}
```

Explicação: Este programa usa um laço while para extrair cada dígito do número fornecido pelo usuário. A cada iteração, o dígito menos significativo (obtido pelo operador % 10) é somado, e o número é reduzido, removendo o último dígito. O processo continua até que o número seja zero.

Desafio 18: FizzBuzz

Enunciado: Implemente o desafio clássico FizzBuzz: exiba os números de 1 a 100, mas para múltiplos de 3 exiba "Fizz", para múltiplos de 5 exiba "Buzz", e para múltiplos de ambos exiba "FizzBuzz".

Solução:

```
public class FizzBuzz {
    public static void main(String[] args) {
        for (int i = 1; i <= 100; i++) {
            if (i % 3 == 0 && i % 5 == 0) {
                System.out.println("FizzBuzz");
            } else if (i % 3 == 0) {
                System.out.println("Fizz");
            } else if (i % 5 == 0) {
                System.out.println("Buzz");
            } else {
                System.out.println(i);
            }
        }
    }
}
```

Explicação: O programa percorre os números de 1 a 100. Para cada número, verifica se ele é divisível por 3, 5 ou ambos. Dependendo da condição atendida, ele exibe "Fizz", "Buzz" ou "FizzBuzz", ou apenas o número caso nenhuma condição seja atendida.

Desafio 19: Calculadora de IMC

Enunciado: Crie um programa que receba o peso e a altura do usuário e calcule seu Índice de Massa Corporal (IMC). O programa deve exibir a categoria de peso do usuário (abaixo do peso, peso normal, sobrepeso, etc.).

Solução:

```
import java.util.Scanner;

public class CalculadoraIMC {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Digite seu peso (kg): ");
        double peso = scanner.nextDouble();

        System.out.print("Digite sua altura (m): ");
        double altura = scanner.nextDouble();

        double imc = peso / (altura * altura);
        System.out.println("Seu IMC é: " + imc);

        if (imc < 18.5) {
            System.out.println("Abaixo do peso.");
        } else if (imc >= 18.5 && imc < 24.9) {
            System.out.println("Peso normal.");
        } else if (imc >= 25 && imc < 29.9) {
            System.out.println("Sobrepeso.");
        } else {
            System.out.println("Obesidade.");
        }
    }
}
```

Explicação: O programa calcula o IMC usando a fórmula IMC = peso / altura^2. Com base no valor do IMC, o programa classifica o usuário em categorias como "Abaixo do peso", "Peso normal", "Sobrepeso" ou "Obesidade", de acordo com os parâmetros estabelecidos pela Organização Mundial da Saúde (OMS).

Desafio 20: Contador de Caracteres Únicos

Enunciado: Escreva um programa que receba uma string e conte quantos caracteres únicos (sem repetição) existem nela.

Solução:

```
import java.util.HashSet;
import java.util.Scanner;

public class ContadorCaracteresUnicos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Digite uma string: ");
        String texto = scanner.nextLine();

        HashSet<Character> caracteresUnicos = new HashSet<>();
```

```
        for (char c : texto.toCharArray()) {
            caracteresUnicos.add(c);
        }

        System.out.println("Número de caracteres únicos: " +
caracteresUnicos.size());
    }
}
```

Explicação: Este programa usa um HashSet para armazenar caracteres únicos de uma string. O HashSet não permite duplicatas, então quando a string é convertida em um array de caracteres e os caracteres são inseridos no conjunto, apenas os caracteres únicos são mantidos. O tamanho do conjunto final indica o número de caracteres únicos.

Conclusão do Capítulo de Desafios

Neste capítulo, você enfrentou uma série de desafios práticos que abordaram diferentes aspectos fundamentais da programação. Desde o cálculo de médias, manipulação de strings, jogos interativos, até a criação de algoritmos clássicos como o FizzBuzz, cada exercício foi projetado para reforçar sua compreensão dos conceitos essenciais.

Esses desafios não apenas exigiram uma boa lógica e raciocínio, mas também trouxeram a necessidade de aplicar estratégias eficientes, como laços de repetição, condições, manipulação de arrays, e até mesmo a implementação de funções recursivas. Além disso, você trabalhou com uma variedade de tipos de dados e operações que simulam problemas do mundo real.

Parabéns por concluir o capítulo de Desafios!

> Por fim, fica o convite para conhecer meu **Curso Completo de Java**, que vai te fazer dominar a linguagem Java!