

Módulo 3: Tópicos Intermediários em Java (Detalhado)

Objetivo: Aprofundar o conhecimento em Java com recursos avançados para aplicações robustas.

Conteúdo:

- Coleções (Collections): List, Set, Map
- Exceções (Exceptions): Captura e criação de exceções personalizadas
- Java 8+ Features: Lambdas, Streams e Optional

1. Coleções (Collections)

As coleções em Java são estruturas de dados que permitem armazenar e manipular grupos de objetos. Elas oferecem diversas vantagens em relação aos arrays tradicionais, como tamanho dinâmico e métodos para manipulação dos elementos.

1.1 List

- **O que é:** Uma List é uma interface que representa uma coleção ordenada de elementos. Ela permite elementos duplicados e nulos. O ArrayList e o LinkedList são implementações comuns da interface List.
- **Como funciona:**
 - ArrayList: Utiliza um array dinâmico para armazenar os elementos. O acesso aos elementos por índice é rápido, mas a inserção e remoção de elementos no meio da lista pode ser lenta.
 - LinkedList: Utiliza uma lista duplamente ligada para armazenar os elementos. A inserção e remoção de elementos no meio da lista é rápida, mas o acesso aos elementos por índice é lento.
- **Importância:** A interface List é útil quando você precisa armazenar uma coleção ordenada de elementos. Ela é ideal para representar listas de objetos, como uma lista de compras ou uma lista de usuários.
- **ArrayList:**
 - **O que é:** O ArrayList é uma implementação da interface List que utiliza um array dinâmico para armazenar os elementos. Ele permite adicionar, remover e acessar elementos de forma eficiente.
 - **Como funciona:** O ArrayList mantém um array interno que armazena os elementos. Quando o array interno fica cheio, ele é automaticamente redimensionado para acomodar mais elementos.
 - **Importância:** O ArrayList é uma das coleções mais utilizadas em Java devido à sua flexibilidade e eficiência. Ele é ideal para armazenar listas de objetos quando a ordem dos elementos é importante e o tamanho da lista pode variar.

- **Exemplo:**

```
Java

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;

public class ExemploList {
    public static void main(String[] args) {
        // ArrayList
        List<String> nomesArrayList = new ArrayList<>();
        nomesArrayList.add("Alice");
        nomesArrayList.add("Bob");
        nomesArrayList.add("Charlie");

        System.out.println("ArrayList: " + nomesArrayList);

        // LinkedList
        List<String> nomesLinkedList = new LinkedList<>();
        nomesLinkedList.add("Alice");
        nomesLinkedList.add("Bob");
        nomesLinkedList.add("Charlie");

        System.out.println("LinkedList: " + nomesLinkedList);
    }
}
```

1.2 Set

- **O que é:** Um Set é uma interface que representa uma coleção de elementos únicos, ou seja, não permite elementos duplicados. O HashSet é uma implementação comum do Set.
- **Como funciona:** O HashSet utiliza uma tabela hash para armazenar os elementos. Isso garante que cada elemento seja único e permite acesso rápido aos elementos.
- **Importância:** O Set é útil quando você precisa armazenar uma coleção de elementos sem duplicatas. Ele é ideal para representar conjuntos de objetos, como um conjunto de usuários ou um conjunto de produtos.

- **Exemplo:**

```
Java

import java.util.HashSet;
import java.util.Set;

public class ExemploHashSet {
    public static void main(String[] args) {
        Set<String> nomes = new HashSet<>();
        nomes.add("Alice");
        nomes.add("Bob");
        nomes.add("Alice"); // Duplicado, ignorado

        System.out.println("HashSet: " + nomes);
    }
}
```

1.3 Map

- **O que é:** Um Map é uma interface que representa uma coleção de pares chave-valor. O HashMap é uma implementação comum do Map.
- **Como funciona:** O HashMap utiliza uma tabela hash para armazenar os pares chave-valor. Isso permite acesso rápido aos valores com base nas chaves.
- **Importância:** O Map é útil quando você precisa armazenar uma coleção de pares chave-valor. Ele é ideal para representar mapeamentos de objetos, como um mapeamento de nomes para idades ou um mapeamento de códigos de produtos para descrições.
- **Exemplo:**

```
Java

import java.util.HashMap;
import java.util.Map;

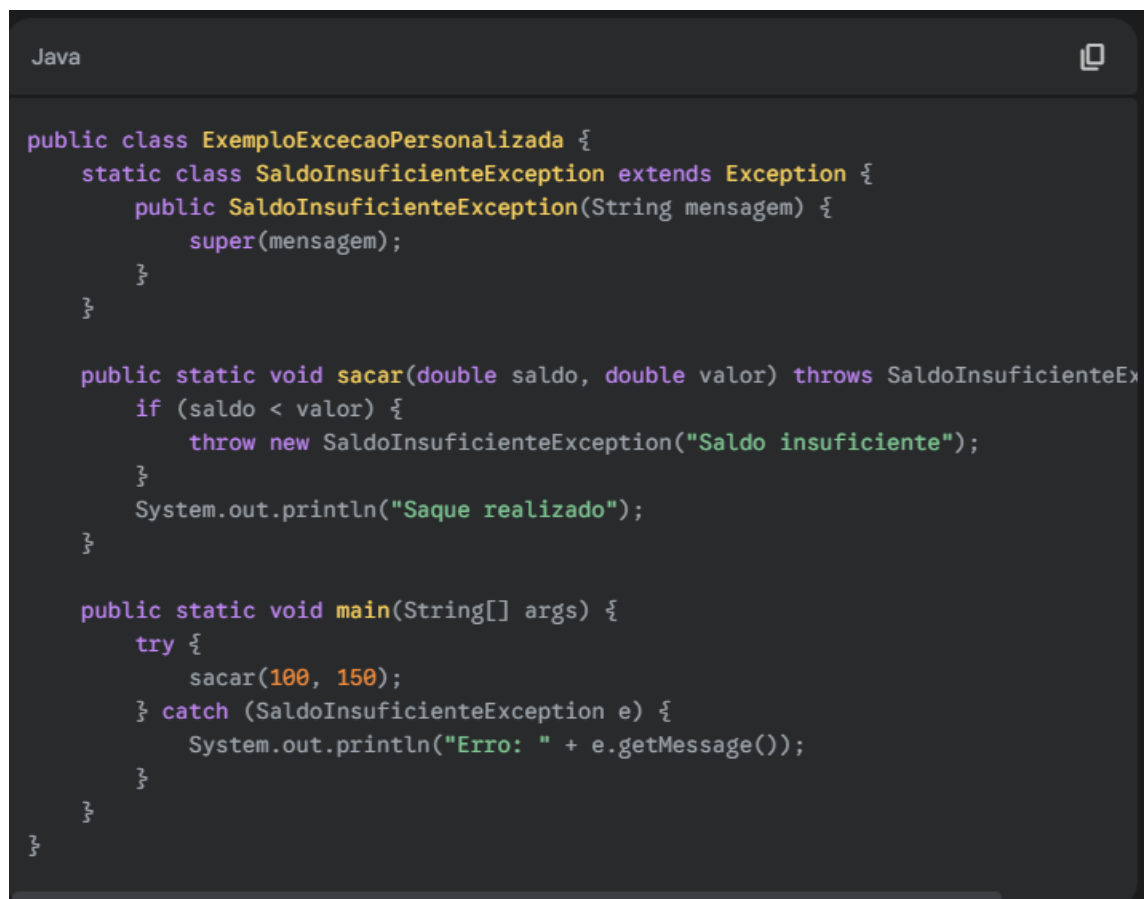
public class ExemploHashMap {
    public static void main(String[] args) {
        Map<String, Double> notas = new HashMap<>();
        notas.put("Alice", 8.5);
        notas.put("Bob", 9.0);
        notas.put("Charlie", 7.5);

        System.out.println("HashMap: " + notas);
        System.out.println("Nota de Bob: " + notas.get("Bob"));
    }
}
```

2. Exceções (Exceptions)

Exceções são eventos que interrompem o fluxo normal de execução de um programa. Java fornece mecanismos para capturar e tratar exceções, evitando que o programa seja encerrado abruptamente.

- **O que são:** Exceções são eventos que ocorrem durante a execução de um programa e que interrompem o fluxo normal de execução.
- **Como funcionam:** Quando ocorre uma exceção, o programa procura por um bloco catch que possa tratar a exceção. Se um bloco catch for encontrado, a exceção é tratada e o programa continua a execução. Se nenhum bloco catch for encontrado, o programa é encerrado.
- **Importância:** As exceções são importantes porque permitem que você escreva código mais robusto e tolerante a falhas. Elas permitem que você lide com erros de forma controlada, evitando que o programa seja encerrado abruptamente.
- **Exemplo:**



```
Java

public class ExemploExcecaoPersonalizada {
    static class SaldoInsuficienteException extends Exception {
        public SaldoInsuficienteException(String mensagem) {
            super(mensagem);
        }
    }

    public static void sacar(double saldo, double valor) throws SaldoInsuficienteException {
        if (saldo < valor) {
            throw new SaldoInsuficienteException("Saldo insuficiente");
        }
        System.out.println("Saque realizado");
    }

    public static void main(String[] args) {
        try {
            sacar(100, 150);
        } catch (SaldoInsuficienteException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

3. Java 8+ Features

Java 8 introduziu diversas funcionalidades que tornam o código mais conciso e expressivo.

3.1 Lambdas

- **O que são:** Lambdas são funções anônimas que podem ser usadas para simplificar a escrita de código.
- **Como funcionam:** Lambdas são funções que não possuem nome. Elas podem ser passadas como argumentos para métodos ou armazenadas em variáveis.
- **Importância:** Lambdas são úteis para simplificar a escrita de código para interfaces funcionais (uma única função abstrata). Elas são especialmente úteis para trabalhar com coleções e eventos.
- **Exemplo:**

```
Java

import java.util.Arrays;
import java.util.List;

public class ExemploLambda {
    public static void main(String[] args) {
        List<String> nomes = Arrays.asList("Alice", "Bob", "Charlie");
        nomes.forEach(nome -> System.out.println("Olá, " + nome));
    }
}
```

3.2 Streams

- **O que são:** Streams são sequências de elementos que podem ser processadas de forma declarativa.
- **Como funcionam:** Streams permitem que você execute operações em coleções de forma declarativa, ou seja, você especifica o que você quer fazer, mas não como fazer.
- **Importância:** Streams são úteis para processar coleções de forma eficiente e concisa. Eles permitem que você filtre, mapeie, reduza e outras operações em coleções de forma declarativa.
- **Exemplo:**

```
Java

import java.util.Arrays;
import java.util.List;

public class ExemploStream {
    public static void main(String[] args) {
        List<Integer> numeros = Arrays.asList(1, 2, 3, 4, 5);
        double media = numeros.stream()
```