Manual de Estudo de Java

Conceitos e Ferramentas Utilizados

Este manual foi criado para ajudar no estudo de conceitos fundamentais da linguagem Java, especialmente voltados para POO, tratamento de exceções e manipulação de coleções. Aqui, serão abordados pontos chave, explicações teóricas e práticas recomendadas.

1. Utilizando conceitos de POO

A Programação Orientada a Objetos (POO) é um paradigma de programação baseado em objetos, que são instâncias de classes que encapsulam dados e comportamentos. Os conceitos fundamentais de POO incluem:
- Classes e Objetos: Representam entidades do mundo real com atributos e comportamentos.
- Herança: Permite criar novas classes com base em classes existentes.
- Polimorfismo: Permite que objetos de diferentes classes possam ser tratados de forma uniforme.
- Encapsulamento: Restringe o acesso a detalhes internos de um objeto.

2. Biblioteca Padrão do Java

O Java oferece uma ampla biblioteca padrão, que facilita o desenvolvimento de software. Alguns dos componentes importantes incluem:
- ArrayList: Uma coleção que permite armazenar listas dinâmicas de objetos.
- Streams: Um recurso para processar dados de forma funcional com operações como forEach, filter, e map.
- Optional: Um contêiner que pode ou não conter um valor, ajudando a lidar com valores nulos.

```java
ArrayList<Produto> lista = new ArrayList<>();
lista.add(produto1);
```

```java
lista.stream().filter(p -> p.getPreco() > 100).forEach(System.out::println);
```

3. Tratamento de Exceções

O tratamento de exceções é crucial para lidar com erros inesperados no código de maneira controlada. A classe IOException, por exemplo, é uma exceção verificada usada para sinalizar problemas relacionados a entrada e saída de dados. Utilize blocos try-catch para capturar e tratar essas exceções, garantindo que o programa não quebre com erros inesperados.

```java
try {
    FileReader leitor = new FileReader("arquivo.txt");
} catch (IOException e) {
    System.out.println("Erro ao ler o arquivo: " + e.getMessage());
}
```

- **try-catch**: Captura exceções para prevenir falhas no sistema.

  Exemplo:

```java
try {
    int resultado = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Erro: Divisão por zero.");
}
```

4. Método toString()

O método toString() é usado para converter um objeto em sua representação textual. Ele é geralmente sobrescrito para fornecer uma saída legível que descreva o estado do objeto.

```java
@Override
public String toString() {
    return "Produto{" + "nome='" + nome + '\'' + ", preco=" + preco + '}';
}
```

5. Controle de Fluxo: Switch Case, While, e For

Switch Case: É uma estrutura condicional que permite selecionar uma entre várias opções baseadas no valor de uma expressão.
While: É uma estrutura de repetição que executa um bloco de código enquanto uma condição for verdadeira.
For: É usado para executar um bloco de código um número fixo de vezes, sendo muito útil para percorrer coleções.

```java
int opcao = 1;
switch (opcao) {
    case 1:
        System.out.println("Opção 1 escolhida.");
        break;
    case 2:
        System.out.println("Opção 2 escolhida.");
        break;
    default:
        System.out.println("Opção inválida.");
}
```

```java
int i = 0;
while (i < 5) {
    System.out.println(i);
    i++;
}
```

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}
```

6. Manipulação de Datas

A classe LocalDate faz parte da API de datas do Java e é usada para representar datas sem informação de horário. Com ela, é possível realizar operações como adição, subtração de dias e verificação de intervalos de tempo.

```java
LocalDate hoje = LocalDate.now();
LocalDate dataEspecifica = LocalDate.of(2024, 9, 30);
```

- **Period**: Representa a diferença entre duas datas.

  Exemplo:

```java
LocalDate dataInicio = LocalDate.of(2024, 1, 1);
LocalDate dataFim = LocalDate.of(2024, 9, 30);
Period periodo = Period.between(dataInicio, dataFim);
System.out.println(periodo.getMonths() + " meses e " + periodo.getDays() + " dias.");
```

7. Map e HashMap

Map é uma interface do Java que mapeia chaves para valores, garantindo que cada chave seja única. HashMap é uma implementação dessa interface que utiliza tabelas de hash para armazenar as chaves e seus respectivos valores. É recomendado utilizar HashMap quando for necessário recuperar dados rapidamente através de chaves únicas.

java

Copiar código

```java
HashMap<Integer, String> mapa = new HashMap<>();
mapa.put(1, "Produto 1");
mapa.put(2, "Produto 2");

// Acessando valores por chave
String produto = mapa.get(1);
System.out.println(produto);
```