RESIDÊNCIA EM TECNOLOGIA DA INFORMAÇÃO E COMUNICAÇÃO

ALBERT SILVA DE JESUS

JAVA, EXCEÇÃO

Atividade PI-P003 de programação em Java, questionário sobre exceções apresentada ao docente Alvaro degas, como requisito parcial para aprovação na disciplina de programação java.

Ilhéus – Bahia

2023

1º O que é uma exceção em Java e qual é o propósito de usá-las em programas?

R: Uma exceção em Java é um evento anormal que ocorre durante a execução do programa, representando condições de erro ou situações imprevistas. O propósito das exceções é oferecer um mecanismo de tratamento de erros, permitindo que o programa lide de maneira elegante com condições inesperadas, melhorando a robustez e a confiabilidade do software. As exceções também facilitam a separação do código normal do código de tratamento de erros, a propagação de erros entre métodos e a manutenção do fluxo de controle mesmo em caso de erros, além de notificar o código cliente sobre condições excepcionais. Existem duas categorias principais de exceções em Java: verificadas, que exigem tratamento explícito, e não verificadas, que geralmente indicam erros de programação.

2º Pesquise sobre a diferença entre exceções verificadas e não verificadas em Java. Dê exemplos de cada uma.

R: Em Java, exceções podem ser divididas em duas categorias principais: exceções verificadas (checked exceptions) e exceções não verificadas (unchecked exceptions).

Exceções Verificadas (Checked Exceptions):

- Exceções verificadas são exceções que o compilador Java obriga você a lidar explicitamente. Isso significa que você deve usar um bloco **try-catch** para capturar a exceção ou declarar que o método pode lançar essa exceção usando a cláusula **throws** no cabeçalho do método.
- Exemplos de exceções verificadas incluem **IOException**, **SQLException**, **FileNotFoundException**, entre outras.

Exemplo de exceção verificada (**IOException**):

```java
package semana3.p003;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ExcecaoVerificadaExemplo {
    Run | Debug
    public static void main(String[] args) {

        try {

            BufferedReader reader = new BufferedReader(new FileReader(fileName:"arquivo.txt"));
            String linha = reader.readLine();
            System.out.println("Conteúdo do arquivo: " + linha);
            reader.close();

        } catch (IOException e) {
            System.out.println("Erro de leitura: " + e.getMessage());
        }
    }
}
```

Neste exemplo, a leitura de um arquivo pode resultar em uma exceção **IOException**, que é uma exceção verificada. Portanto, é necessário tratá-la ou declará-la no cabeçalho do método.

Exceções Não Verificadas (Unchecked Exceptions):

- Exceções não verificadas são exceções que o compilador Java não exige que você lide explicitamente. Geralmente, são subclasses de **RuntimeException** e são frequentemente associadas a erros de programação que podem ser evitados corrigindo o código.
- Exemplos de exceções não verificadas incluem **NullPointerException**, **ArrayIndexOutOfBoundsException**, **ArithmeticException**, entre outras.

Exemplo de exceção não verificada (**NullPointerException**):

```java
package semana3.p003;

public class ExcecaoNaoVerificadaExemplo {
    Run | Debug
    public static void main(String[] args) {
        String str = null;

        try {

            // Tentar acessar o comprimento de uma string nula resultará em NullPointerException
            int comprimento = str.length();
            System.out.println("Comprimento da string: " + comprimento);

        } catch (NullPointerException e) {
            System.out.println(x:"Erro: A string é nula.");
        }
    }
}
```

Neste exemplo, a tentativa de acessar o comprimento de uma string nula resulta em uma exceção **NullPointerException**, que é uma exceção não verificada. Você não é obrigado a lidar explicitamente com ela, mas é uma boa prática verificar e evitar tais situações.

3º Como você pode lidar com exceções em Java? Quais são as palavras-chave e as práticas comuns para tratamento de exceções?

R: Em Java, o tratamento de exceções é realizado usando blocos **try**, **catch**, **finally** e, opcionalmente, **throws**. Aqui estão as principais palavras-chave e práticas comuns para o tratamento de exceções:

try: O bloco **try** contém o código que pode gerar uma exceção. Qualquer exceção lançada dentro deste bloco será tratada pelos blocos **catch** associados.

catch: O bloco **catch** é usado para capturar e tratar exceções. Pode haver vários blocos **catch** associados a um bloco **try**, cada um tratando um tipo específico de exceção.

finally: O bloco **finally** é usado para fornecer código que será executado independentemente de ocorrer ou não uma exceção. Este bloco é opcional.

throws: A palavra-chave **throws** é usada no cabeçalho de um método para declarar que o método pode lançar uma exceção específica. Isso informa aos chamadores do método que eles devem lidar com a exceção ou declará-la novamente em seus próprios cabeçalhos.

throw: A palavra-chave **throw** é usada para explicitamente lançar uma exceção em qualquer ponto do código. Pode ser usada para sinalizar condições de erro específicas.

Boas Práticas:

- Trate exceções de maneira apropriada para o contexto.

- Use blocos **try** e **catch** de maneira a manter o código legível e organizado.
- Evite capturar exceções genéricas, como **Exception**, sempre que possível. Capture exceções específicas para tratar casos específicos.
- Se possível, forneça mensagens de erro significativas ao criar ou lançar exceções.
- Utilize o bloco **finally** para garantir que recursos sejam liberados, mesmo em caso de exceção.

4º O que é o bloco "try-catch" em Java? Como ele funciona e por que é importante usá-lo ao lidar com exceções?

R: O bloco **try-catch** em Java é uma construção fundamental para o tratamento de exceções. Ele é usado para envolver um bloco de código que pode gerar exceções e fornece um mecanismo para capturar e lidar com essas exceções de maneira controlada.

Como Funciona:

try: O código que pode gerar uma exceção é colocado dentro do bloco **try**. Este bloco é monitorado quanto a exceções.

catch: Se uma exceção do tipo especificado ocorrer dentro do bloco **try**, o controle é transferido para o bloco **catch** correspondente. Vários blocos **catch** podem ser usados para tratar diferentes tipos de exceções.

finally: O bloco **finally** é opcional e é usado para conter código que deve ser executado independentemente de ocorrer ou não uma exceção. Este bloco é frequentemente usado para liberar recursos, como fechar arquivos ou conexões de banco de dados.

Por que é Importante Usar o bloco try-catch ao Lidar com Exceções?

I. **Prevenção de Encerramento Inesperado:** O uso do bloco **try-catch** evita que o programa seja encerrado abruptamente quando ocorre uma exceção. Isso é importante para garantir a robustez e confiabilidade do software.

II. **Tratamento Controlado de Exceções:** O bloco **catch** permite que você forneça um código específico para lidar com diferentes tipos de exceções. Isso ajuda a tomar ações apropriadas em resposta a condições excepcionais.

III. **Separação de Lógica de Tratamento de Erros:** O bloco **try-catch** facilita a separação da lógica principal do programa e da lógica de tratamento de erros. Isso melhora a legibilidade e a manutenção do código.

IV. **Propagação Controlada de Exceções:** O bloco **try-catch** permite que você capture exceções e tome medidas apropriadas, evitando que a exceção seja propagada para níveis superiores da pilha de chamadas, a menos que seja desejado.

V. **Liberação de Recursos:** O bloco **finally** é útil para garantir que recursos sejam liberados, mesmo em caso de exceção. Isso é crítico para evitar vazamentos de recursos, como não fechar um arquivo após a leitura.

VI. Ao utilizar o bloco **try-catch** de forma apropriada, você torna seu código mais robusto, controla melhor o fluxo de execução em situações excepcionais e facilita a manutenção do software.

5º Quando é apropriado criar suas próprias exceções personalizadas em Java e como você pode fazer isso? Dê um exemplo de quando e por que você criaria uma exceção personalizada.

R: É apropriado criar suas próprias exceções personalizadas em Java quando você está lidando com situações específicas em seu domínio de aplicação que não podem ser

adequadamente representadas por exceções existentes no Java ou quando você deseja criar um mecanismo mais informativo e específico para lidar com erros em seu código.

Para criar uma exceção personalizada em Java, você geralmente estende a classe **Exception** (ou uma de suas subclasses) e fornece construtores e métodos adicionais conforme necessário.

Exemplo:

- Vamos supor que você está desenvolvendo um sistema de gerenciamento de biblioteca e quer garantir que um livro só seja emprestado se estiver disponível. Se um usuário tentar emprestar um livro que não está disponível, você pode lançar uma exceção personalizada **LivroNaoDisponivelException**:
- Se estiver desenvolvendo uma aplicação financeira e precisar lidar com transações, pode criar uma exceção personalizada chamada **TransacaoInvalidaException**. Isso permitiria que você lançasse essa exceção em casos específicos, como tentativas de realizar uma transação com um valor negativo.