Respostas

Questão 1

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

Exceção é um evento não esperado que ocorre no sistema quando está em tempo de execução (Runtime). Geralmente quando o sistema captura alguma exceção o fluxo do código fica interrompido. Para conseguir capturar uma **exceção**, é preciso fazer antes o tratamento. O uso dos tratamentos é importante nos sistemas porque auxilia em falhas como: comunicação, leitura e escrita de arquivos, entrada de dados inválidos, acesso a elementos fora de índice, entre outros.

Questão 2

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

Uma exceção verificada em Java representa uma situação previsível e errônea que pode ocorrer mesmo que uma biblioteca de software seja usada como pretendido.

Por exemplo, se um desenvolvedor tentar acessar um arquivo, a biblioteca Java IO os obriga a lidar com o FileNotFoundException. Os desenvolvedores da API Java IO anteciparam que tentar acessar um arquivo que não existe seria uma ocorrência relativamente comum, então eles criaram uma exceção verificada para forçar um desenvolvedor a lidar com isso.

Em contraste com uma exceção verificada, uma exceção não verificadas representa um erro na lógica de programação, não uma situação errônea que pode ocorrer razoavelmente durante o uso adequado de uma API. Como o compilador não pode antecipar erros lógicos que surgem apenas em tempo de execução, ele não pode verificar esses tipos de problemas em tempo de compilação. É por isso que eles são chamados de exceções "não verificadas".

Exceções não verificadas resultam de lógica defeituosa que pode ocorrer em qualquer lugar de um programa de software. Por exemplo, se um desenvolvedor invoca um método em um objeto null, ocorre uma exceção NullPointerException. Se um desenvolvedor tentar acessar um elemento de matriz que não existe, o ArrayIndexOutOfBoundsException desmarcado ocorre.

Comparação verificada vs. exceções não verificadas

Critérios de crise	Exceção não verificada	Exceção com verificação
Propósito	Erros imprevistos na lógica que aparecem em tempo de execução	Problemas antecipados associados ao uso normal de uma API
Ancestralidade	Inclui Exceção de Runtime	Não inclui RuntimeException
Manuseio de	A semântica de tratamento de não é necessária	Deve ser manipulado em um bloco try-and-catch, ou ser jogado pelo método de invocação
A extensão	Pode ser personalizado estendendo RuntimeException	Pode ser personalizado estendendo o java.lang.Exception
Lista de exemplos	Exceção de NullPointer, ClassCastException,	Acepção ClassNotFoundException, SocketException, SQLException,

Critérios de crise	Exceção não verificada	Exceção com verificação
	AritméticaExceção, DataTimeException, ArrayStoreException	IOException, FileNotFoundException

Questão 3

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

Em Java, e em outras linguagens Orientação Objeto ou modernas, é necessário o tratamento de exceções, para ter controle sobre alguma adversidade no programa. Para Aqui estão algumas palavras-chave e práticas comuns para o tratamento de exceções em Java:

1. **try-catch:**
    - **try:** O bloco try contém o código onde uma exceção pode ocorrer.
    - **catch:** O bloco catch é usado para capturar e lidar com exceções específicas. Você pode ter vários blocos catch para tratar diferentes tipos de exceções.

Exemplo:

java

- try {
  // Código onde uma exceção pode ocorrer
} catch (TipoDeExcecao1 e1) {
  // Lidar com exceção do TipoDeExcecao1
} catch (TipoDeExcecao2 e2) {
  // Lidar com exceção do TipoDeExcecao2
} finally {
  // Bloco opcional: código que será executado independentemente de ocorrer ou não uma exceção
}

- **throw:**
- A palavra-chave throw é usada para explicitamente lançar uma exceção.

Exemplo:

java

- throw new MinhaExcecao("Isso é uma mensagem de exceção");

- **throws:**
- A palavra-chave throws é usada na assinatura de um método para indicar que o método pode lançar uma exceção de um tipo específico.

Exemplo:

java

- public void meuMetodo() throws MinhaExcecao {
  // Código do método que pode lançar MinhaExcecao
}

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

Exemplo:

java

```
try {
   // Código onde uma exceção pode ocorrer
} catch (TipoDeExcecao e) {
   // Lidar com a exceção
} finally {
   // Código que será executado independentemente de ocorrer uma exceção ou não
}
```

- **try-with-resources:**
- A partir do Java 7, você pode usar a declaração try-with-resources para simplificar a gestão de recursos que precisam ser fechados, como objetos que implementam AutoCloseable ou Closeable.

Exemplo:

java

```
try (FileReader reader = new FileReader("arquivo.txt");
    BufferedReader bufferedReader = new BufferedReader(reader)) {
   // Código que usa o leitor e o leitor de buffer
} catch (IOException e) {
   // Lidar com exceção de E/S
}
```

- **Criar suas próprias exceções:**
- É possível criar suas próprias classes de exceção estendendo Exception ou RuntimeException (se você quiser criar uma exceção de tempo de execução). Isso pode ser útil para sinalizar erros específicos em seu código.

Exemplo:

java

```
6. public class MinhaExcecao extends Exception {
       public MinhaExcecao(String mensagem) {
          super(mensagem);
       }
   }
```

Obs.: Ao tratar as exceções em Java, é uma prática recomendada ser específico sobre as exceções que você está tratando, em vez de usar blocos catch genéricos. Isso ajuda a identificar e corrigir problemas de maneira mais eficaz. Além disso, gerenciar recursos adequadamente usando finally ou try-with-resources é crucial para evitar vazamentos de recursos.

Questão 4

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

Importância do bloco try-catch ao lidar com exceções:

1. **Controle de Fluxo:** O bloco try-catch permite que você mantenha o controle do fluxo do programa mesmo quando exceções ocorrem. Isso evita que o programa termine abruptamente devido a uma exceção não tratada.
2. **Tratamento de Exceções:** O bloco catch oferece a oportunidade de lidar com exceções de maneira controlada. Isso pode incluir a impressão de mensagens de erro, a tomada de ações corretivas ou a notificação do usuário sobre o problema.
3. **Prevenção de Encerramento Inesperado:** Sem tratamento de exceções, erros inesperados podem levar ao encerramento abrupto do programa, tornando-o menos robusto e mais difícil de depurar.
4. **Manutenção de Recursos:** O uso de finally (ou try-with-resources) dentro do bloco try-catch ajuda a garantir que os recursos sejam adequadamente liberados, mesmo se uma exceção ocorrer. Isso é crucial para evitar vazamentos de recursos, como em operações de E/S (entrada/saída) com arquivos.

Portanto, o bloco try-catch é uma ferramenta essencial para lidar com exceções de maneira eficaz e manter a robustez e a confiabilidade do código Java.

Questão 5

5. 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.

Geralmente, o uso de exceção está relacionado a códigos que desempenham alguma interação com dados, acesso a arrays/banco de dados, cálculos, ou seja, tudo que o compilador verificar que existe algum risco, será necessário o uso dos blocos/cláusulas de tratamento de erros como os blocos try/catch/finally.

Aqui está uma explicação de como funciona:

1. **Bloco** try**:**
    - O código que pode gerar exceções é colocado dentro do bloco try. Se uma exceção ocorrer dentro deste bloco, a execução do código dentro do bloco try será interrompida.
2. **Bloco** catch**:**
    - Se uma exceção do tipo especificado (ou de um tipo relacionado, se o tipo especificado for uma classe base) ocorrer dentro do bloco try, o controle do programa será transferido para o bloco catch correspondente.
    - O bloco catch contém o código que será executado para lidar com a exceção. Aqui, você pode incluir lógica para tratar a exceção, como imprimir mensagens de erro, registrar informações de log ou tomar medidas corretivas.

Exemplo:

java

```
try {
    // Código que pode gerar uma exceção
    int resultado = dividir(10, 0);
    System.out.println("Resultado: " + resultado); // Esta linha não será alcançada devido à exceção
} catch (ArithmeticException e) {
    // Código para lidar com a exceção
    System.err.println("Erro aritmético: " + e.getMessage());
}
```

Neste exemplo, se a função dividir tentar dividir por zero, uma ArithmeticException será lançada, e o controle será transferido para o bloco catch, onde você pode lidar com a exceção de maneira apropriada.