RESOLUÇÃO DAS QUESTÕES DE 1 A 5 DE JAVA

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

R: Em Java, uma exceção é um evento anormal que ocorre durante a execução de um programa e interrompe o fluxo normal de instruções.
As exceções são usadas para lidar com situações de erro ou condições imprevistas que podem surgir durante a execução de um programa.
Existem dois tipos principais de exceções em Java: exceções verificadas (checked exceptions)
e exceções não verificadas (unchecked exceptions).

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

R: Exceções Verificadas:
São exceções que o compilador obriga o programador a lidar explicitamente.
Geralmente são relacionadas a condições que um programa pode prever e tratar,
como operações de entrada e saída que podem lançar exceções, como IOException.

```
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class ExemploCheckedException {
    public static void main(String[] args) {
        try {
            FileReader fileReader = new FileReader(new File("arquivo.txt"));
            // Código que manipula o arquivo
        } catch (IOException e) {
            // Tratamento da exceção
            e.printStackTrace();
        }
    }
}
```

Exceções Não Verificadas:
Não exigem que o programador lide explicitamente com elas.
Geralmente são erros de programação, como dividir por zero (ArithmeticException)
ou acessar um índice inválido em um array (ArrayIndexOutOfBoundsException).

```
public class ExemploUncheckedException {
    public static void main(String[] args) {
        int[] array = {1, 2, 3};
        int resultado = array[5]; // Isso lançará ArrayIndexOutOfBoundsException em tempo de
execução
```

```
    }
}
```

3. 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 por meio de blocos try, catch, finally, e opcionalmente,
throws nas assinaturas de métodos.
Aqui estão algumas palavras-chave e práticas comuns para o tratamento de exceções:
* O bloco try é usado para envolver um conjunto de instruções onde uma exceção pode ocorrer.
Se uma exceção ocorrer dentro do bloco try, o controle do programa será transferido para o bloco catch.
* O bloco catch é usado para capturar e lidar com exceções.
Pode haver vários blocos catch para tratar diferentes tipos de exceções.
* O bloco finally é usado para definir código que será executado, independentemente de uma exceção ter ocorrido ou não.
É útil para liberar recursos (como fechar arquivos ou conexões de banco de dados) que devem ser executados independentemente de ocorrer uma exceção.
* A palavra-chave throws é usada na declaração de um método para indicar que o método pode lançar uma exceção.
É usado para transferir a responsabilidade do tratamento da exceção para quem chama o método.
* Java possui uma hierarquia de exceções, com Exception sendo a classe base.
É uma prática comum capturar exceções mais específicas antes das mais genéricas, para evitar blocos de código que capturam exceções indesejadas.
* A partir do Java 7, é possível ter vários tipos de exceções no mesmo bloco catch.
* Para recursos que implementam a interface AutoCloseable (como FileInputStream, Socket, etc.),
* você pode usar o bloco try-with-resources para garantir que os recursos sejam fechados corretamente.

4. 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 utilizada para lidar com exceções,
que são eventos anormais ou erros que podem ocorrer durante a execução de um programa.
O objetivo do try-catch é permitir que um programa trate essas exceções de maneira controlada,
evitando que elas interrompam abruptamente a execução normal do código.
try: O bloco try envolve o código que pode gerar uma exceção. Se uma exceção ocorrer dentro deste bloco, o controle do programa é transferido para o bloco catch.
catch: O bloco catch é responsável por capturar e tratar a exceção. Ele especifica o tipo de exceção que pode ser tratado (por meio do tipo de exceção entre parênteses).

Quando uma exceção do tipo especificado ocorre no bloco try, o código dentro do bloco catch é executado.
Importância do try-catch ao lidar com exceções:
* Prevenção de falhas abruptas: O uso do try-catch evita que uma exceção não tratada interrompa abruptamente a execução do programa.
Em vez de o programa terminar devido a uma exceção não tratada, você pode fornecer um tratamento adequado e continuar a execução.

* Manuseio controlado de erros: O bloco catch permite que você forneça um código específico para lidar com um tipo particular de exceção.
Isso permite que você tome medidas apropriadas, como exibir uma mensagem de erro, registrar informações relevantes, ou tomar ações corretivas.

* Melhoria da robustez: O try-catch melhora a robustez do código, permitindo que ele lide com situações inesperadas de maneira controlada.
Isso é especialmente importante em ambientes de produção, onde é crucial evitar falhas não controladas.

* Facilitação de depuração: O bloco catch fornece um local centralizado para lidar com exceções, facilitando a depuração e a identificação de problemas no código.

* # 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.

R: Criar suas próprias exceções personalizadas em Java é apropriado quando você precisa representar erros ou situações específicas
ao domínio do seu aplicativo que não são adequadamente cobertos pelas exceções padrão do Java.
Isso ajuda a tornar seu código mais claro, expressivo e a facilitar o tratamento de erros específicos em sua aplicação.
A criação de uma exceção personalizada em Java envolve a criação de uma classe que estende a classe Exception ou uma de suas subclasses, como RuntimeException.

exemplo:

```
public class MeuErroPersonalizado extends Exception {
    public MeuErroPersonalizado() {
        super("Este é um erro personalizado!");
    }

    public MeuErroPersonalizado(String mensagem) {
        super(mensagem);
    }
}

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

```
        try {
            // bloco de codigo
            throw new MeuErroPersonalizado("Ocorreu um problema específico da minha
aplicação.");
        } catch (MeuErroPersonalizado e) {
            System.out.println("Tratando MeuErroPersonalizado: " + e.getMessage());
        }
    }
}
```