UD8 – Excepciones Java

Contenido

1. Introducción a las excepciones

Definición de una excepción.

Una excepción es un evento que ocurre durante la ejecución de un programa y que interrumpe el flujo normal de las instrucciones del programa. En Java, una excepción es un objeto que se crea cuando ocurre un error. Este objeto contiene información sobre el error, incluyendo su tipo y el estado del programa cuando ocurrió el error.

Diferencia entre errores y excepciones

- En Java, tanto los errores como las excepciones son subclases de la clase “Throwable”, pero hay una diferencia importante entre ellos:

Excepciones: Son condiciones que un programa razonablemente podría intentar manejar. Se dividen en dos categorías: verificadas (checked) y no verificadas (unchecked). Las excepciones verificadas son condiciones que un programa debería anticipar y recuperarse de ellas, como “FileNotFoundException”. Las excepciones no verificadas son errores de programación, como “NullPointerException”.

Errores: Son condiciones excepcionales que son externas al control de la aplicación, y que la aplicación generalmente no puede manejar. Por ejemplo, “OutOfMemoryError” ocurre cuando la JVM se queda sin memoria. Los errores no se deben capturar ni manejar en el código de la aplicación.

Cómo Java maneja las excepciones (mecanismo de manejo de excepciones)

- Java utiliza un mecanismo llamado "propagación de excepciones" para manejar las excepciones. Cuando ocurre una excepción, se crea un objeto de excepción y se pasa a lo largo de la pila de llamadas del método hasta que se encuentra un método que puede manejarla o hasta que se alcanza el método principal (“main”).

- Para manejar una excepción, se utiliza un bloque “try/catch”. El código que puede lanzar una excepción se coloca dentro del bloque “try”, y el código para manejar la excepción se coloca dentro del bloque “catch”.

- Un bloque “finally” puede ser añadido después de los bloques “catch” para ejecutar código que debe ser ejecutado independientemente de si una excepción fue lanzada o no.

Veamos un ejemplo:

```
public class Main {
    public static void main(String[] args) {
        try {
            int[] myNumbers = {1, 2, 3};
            System.out.println(myNumbers[10]); // Esto generará una
excepción ArrayIndexOutOfBoundsException
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Algo salió mal.");
        } finally {
            System.out.println("El bloque 'try catch' ha terminado.");
        }
    }
}
```

2. Tipos de excepciones en Java

Excepciones verificadas (Checked Exceptions)

- Las excepciones verificadas son aquellas de las que el código debe anticiparse y manejar de alguna manera, generalmente proporcionando una alternativa o dando un mensaje de error útil. Estas son excepciones que se pueden prever durante la compilación.

- Por ejemplo, si estás utilizando un método que puede lanzar una "IOException" (como un método de lectura de archivo), debes manejar esta excepción, ya sea con un bloque "try/catch" o declarando que tu método puede lanzar esta excepción.

- Algunas excepciones verificadas comunes son "IOException", "ClassNotFoundException", "InstantiationException", etc.

Excepciones no verificadas (Unchecked Exceptions)

- Las excepciones no verificadas son aquellas que ocurren debido a errores de programación. Estas son excepciones que no se pueden prever durante la compilación.

- Por ejemplo, si intentas acceder a un índice de un array que no existe, obtendrás una "ArrayIndexOutOfBoundsException". Este es un error de programación que debería haberse evitado en primer lugar.

- Las excepciones no verificadas son subclases de "RuntimeException", y algunas comunes incluyen "NullPointerException", "ArrayIndexOutOfBoundsException", "ClassCastException", etc.

Errores

- Los errores no son excepciones en absoluto en términos de que no se pueden (y no se deben) capturar y manejar. Son condiciones graves que ocurren en la JVM y hacen que la aplicación se detenga inmediatamente. Los errores no son causados por el programa en sí, sino que son problemas a nivel de sistema que no pueden ser manejados.

- Algunos ejemplos comunes de errores son "OutOfMemoryError" y "StackOverflowError".

3.- Bloques try, catch y finally

Cómo usar “try” y “catch” para manejar excepciones

- El bloque “try” se utiliza para encerrar el código que podría lanzar una excepción. Si ocurre una excepción en el bloque “try”, la ejecución del bloque se interrumpe y el control se pasa al primer bloque “catch” que coincida con el tipo de excepción lanzada.

- El bloque “catch” se utiliza para manejar la excepción. Puedes especificar qué tipo de excepción quieres manejar y proporcionar un bloque de código que se ejecutará en caso de que esa excepción ocurra.

```
try {
    // Código que puede lanzar una excepción
    int division = 10 / 0;
} catch (ArithmeticException e) {
    // Código para manejar la excepción
    System.out.println("No se puede dividir por cero");
}
```

Cómo el bloque “finally” se utiliza para limpiar recursos

- El bloque “finally” se ejecuta después de que los bloques “try” y “catch” se hayan ejecutado, independientemente de si se lanzó una excepción o no. Esto lo hace útil para limpiar recursos que deben cerrarse independientemente de si ocurrió una excepción, como streams de entrada/salida, conexiones a bases de datos, etc.

```
InputStream inputStream = null;
try {
    File file = new File("./test.txt");
    inputStream = new FileInputStream(file);

    // Código para leer del archivo
} catch (FileNotFoundException e) {
    System.out.println("El archivo no fue encontrado");
} finally {
    if (inputStream != null) {
        try {
            inputStream.close();
        } catch (IOException e) {
            System.out.println("Error al cerrar el inputStream");
        }
    }
}
```

En este ejemplo, el bloque “finally” se asegura de que el “InputStream” se cierre independientemente de si se lanzó una excepción o no.

4- La clase Throwable

Jerarquía de excepciones en Java

- En Java, todas las excepciones y errores son subclases de la clase “Throwable”, que es la superclase de toda la jerarquía de excepciones.

- “Throwable” tiene dos subclases directas: “Error” y “Exception”.

→ “Error” es la superclase de todos los errores que un programa normalmente no intenta manejar. Los errores son condiciones excepcionales que son externas al control de la aplicación.

→ “Exception” es la superclase de todas las excepciones que un programa normalmente intenta manejar. “Exception” tiene una subclase importante llamada “RuntimeException”. Todas las excepciones no verificadas son subclases de “RuntimeException”.

Métodos importantes de la clase Throwable

- “public String getMessage()”: Este método se utiliza para obtener el mensaje de detalle de la excepción. El mensaje de detalle es un String que se pasa al constructor de la excepción cuando se lanza.

- “public Throwable getCause()”: Este método se utiliza para obtener la causa de la excepción. La causa es la excepción que provocó que se lanzara esta excepción.

- “public void printStackTrace()”: Este método se utiliza para imprimir la pila de llamadas de la excepción. Esto puede ser útil para depurar, ya que muestra qué métodos estaban en la pila de llamadas en el momento en que se lanzó la excepción.

- “public StackTraceElement[] getStackTrace()”: Este método devuelve un array que contiene cada elemento en la pila de llamadas. Esto puede ser útil si quieres analizar la pila de llamadas en tu código.

Ejemplo:

```
try {
    // Código que puede lanzar una excepción
    int division = 10 / 0;
} catch (ArithmeticException e) {
    // Imprimir el mensaje de la excepción
    System.out.println(e.getMessage());

    // Imprimir la pila de llamadas
    e.printStackTrace();
}
```

Crear excepciones personalizadas

Cómo y por qué crear tus propias excepciones.

- En Java, puedes crear tus propias excepciones definiendo una clase que herede de la clase “Exception” (para excepciones verificadas) o de la clase “RuntimeException” (para excepciones no verificadas).

- Veamos un ejemplo de cómo crear una excepción personalizada:

```
try {
    // Código que puede lanzar una excepción
    int division = 10 / 0;
} catch (ArithmeticException e) {
    // Imprimir el mensaje de la excepción
    System.out.println(e.getMessage());

    // Imprimir la pila de llamadas
    e.printStackTrace();
}
```

Ejemplo de cómo lanzar y manejar esta excepción:

```
try {
    // Lanzar la excepción personalizada
    throw new MyException("Esto es una excepción personalizada");
} catch (MyException e) {
    // Manejar la excepción personalizada
    System.out.println(e.getMessage());
}
```

Por qué crear tus propias excepciones

- Crear tus propias excepciones puede ser útil por varias razones:

- **Especificidad**: Las excepciones personalizadas pueden tener nombres descriptivos que indican qué tipo de error ocurrió, lo que puede hacer que tu código sea más fácil de entender y depurar.
- **Información adicional**: Las excepciones personalizadas pueden contener información adicional sobre el error. Por ejemplo, podrías crear una excepción “InvalidUserInputException” que contenga el input del usuario que causó el error.
- **Control del flujo de programa**: Lanzar y manejar excepciones personalizadas puede ser una forma efectiva de controlar el flujo de tu programa. Puedes lanzar una excepción personalizada cuando ocurre una condición específica y luego manejarla en un bloque “catch” para tomar una acción específica.

6. Buenas prácticas para el manejo de excepciones

No ignorar las excepciones

Ignorar una excepción puede hacer que tu programa continúe en un estado inconsistente, lo que puede llevar a errores más difíciles de diagnosticar más adelante. Si un método lanza una excepción, debes manejarla de alguna manera, ya sea registrándola, mostrando un mensaje al usuario, lanzándola de nuevo, etc.

Evitar el uso de "catch (Exception e)"

Capturar "Exception" captura todas las excepciones, incluyendo las no verificadas que podrían ser causadas por errores de programación. En lugar de eso, debes capturar las excepciones específicas que puedes manejar de manera significativa. Esto también te permite proporcionar mensajes de error más útiles.

No usar excepciones para controlar el flujo de un programa

Las excepciones son para situaciones excepcionales, no para el control de flujo normal de un programa. Usar excepciones para el control de flujo puede hacer que tu código sea más difícil de entender y más lento, ya que lanzar y manejar excepciones es costoso en términos de rendimiento.

Cerrar los recursos en el bloque "finally" o usar try-with-resources

Si abres un recurso (como un stream de entrada/salida o una conexión a la base de datos) en un bloque "try", debes asegurarte de cerrarlo en un bloque "finally" para evitar fugas de recursos. A partir de Java 7, puedes usar la declaración try-with-resources para cerrar automáticamente los recursos, lo que hace que tu código sea más limpio y menos propenso a errores.

Aquí tienes un ejemplo de cómo usar try-with-resources:

```
try (FileInputStream fis = new FileInputStream("file.txt")) {
    // Código para trabajar con el FileInputStream
} catch (IOException e) {
    // Manejar la excepción
}
```

En este ejemplo, el "FileInputStream" se cerrará automáticamente al final del bloque "try", incluso si se lanza una excepción.

7. Ejemplos de código:

Manejo de excepciones try/catch

```
try {
    int division = 10 / 0;
} catch (ArithmeticException e) {
    System.out.println("Error: " + e.getMessage());
}
```

Uso del bloque finally para limpiar recursos

```
InputStream inputStream = null;
try {
    File file = new File("./test.txt");
    inputStream = new FileInputStream(file);
    // Código para leer del archivo
} catch (FileNotFoundException e) {
    System.out.println("El archivo no fue encontrado");
} finally {
    if (inputStream != null) {
        try {
            inputStream.close();
        } catch (IOException e) {
            System.out.println("Error al cerrar el inputStream");
        }
    }
}
```

Creación y uso de una excepción personalizada

```
public class MyException extends Exception {
    public MyException(String message) {
        super(message);
    }
}

// Uso de la excepción personalizada
try {
    throw new MyException("Esto es una excepción personalizada");
} catch (MyException e) {
    System.out.println(e.getMessage());
}
```

8. Lanzar excepciones:

En Java, puedes usar la palabra clave “throws” en la firma de un método para indicar que ese método puede lanzar ciertos tipos de excepciones. Esto es útil cuando un método realiza una operación que puede fallar de una manera que no puede manejar por sí mismo, pero que puede ser manejada por el código que llama al método.

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

```
        try {
            riskyMethod();
        } catch (Exception e) {
            System.out.println("Se capturó una excepción: " +
e.getMessage());
        }
    }

    static void riskyMethod() throws Exception {
        throw new Exception("¡Algo salió mal!");
    }
}
```

En este ejemplo, riskyMethod declara que puede lanzar una Exception. Dentro de riskyMethod, lanzamos una nueva Exception con un mensaje de error.

En el método main, llamamos a riskyMethod dentro de un bloque try. Si riskyMethod lanza una excepción, el bloque catch en main la captura y maneja imprimiendo un mensaje.