

Tratamiento de excepciones en Java

2º Desarrollo de aplicaciones multiplataforma

Autor: Rubén Agra Casal

10/10/2024

Sumario

1. Introducción al Manejo de Excepciones en Java.....	3
¿Qué es una excepción?.....	3
La importancia de manejar las excepciones.....	3
Errors y Exceptions.....	4
2. Jerarquía de excepciones en Java.....	4
Throwable, Exception, RuntimeException.....	4
Diferencias entre excepciones comprobadas (checked exceptions) y excepciones no comprobadas (unchecked exceptions).....	5
3. Estructuras de manejo de excepciones.....	6
¿Qué bloques podemos utilizar para manejar las excepciones?.....	6
4. Tipos comunes y excepciones de Java.....	9
5. Buenas prácticas para el manejo de excepciones.....	9
6. Excepciones personalizadas.....	10
7. Manejo avanzado de excepciones.....	11
8. Conclusión.....	11
Bibliografía.....	11

Índice de figuras

Figura 1: Jerarquía de la clase Throwable.....	5
Figura 2: Código de ejemplo excepción unchecked.....	5
Figura 3: Salida por consola excepcion unchecked.....	6
Figura 4: Código de ejemplo de excepciones checked.....	6
Figura 5: Bloque try-catch ejemplo.....	7
Figura 6: Bloque try-catch ejemplo salida por consola.....	7
Figura 7: Bloque finally ejemplo.....	8
Figura 8: Bloque finally ejemplo salida por consola.....	8
Figura 9: Bloque throw ejemplo.....	9
Figura 10: Bloque throw salida por consola.....	9
Figura 11: Bloque throws ejemplo.....	9
Figura 12: Construcción de la excepción personalizada.....	11
Figura 13: Ejemplo de uso de excepción personalizada.....	11
Figura 14: Salida por consola de la excepción personalizada.....	11

1. Introducción al Manejo de Excepciones en Java

¿Qué es una excepción?

Una excepción es un evento anómalo que puede ocurrir durante la ejecución de un programa, haciendo que altere el flujo normal de la ejecución. Java nos proporciona diferentes herramientas y métodos para poder gestionar estos eventos para hacer que el programa funcione correctamente y así evitar interrupciones.

La importancia de manejar las excepciones

Manejar las excepciones es de suma importancia, ya que repercute directamente en la buena experiencia del usuario ya que la aplicación funcionará correctamente y de en algún momento saltar un error, que se pueda saber a que se debe y como podemos solucionarlo. Las excepciones hacen que podamos manejar los diferentes errores de una manera ordenada, pudiendo separarlos de las demás funciones o incluso indicando en algún método, la excepción que puede surgir. Además de esto, puede proporcionar diferentes beneficios al código:

- Manejar los errores de forma estructurada.
- Mejora en la robustez y en la confiabilidad.
- Retroalimentación al usuario.
- Flexibilidad y adaptabilidad.

Errors y Exceptions

Tenemos varios tipos de excepciones que heredan de la clase Throwable como son las “Exceptions” (Checked y unchecked) y “Errors” en la que la principal diferencia es la gravedad de los errores que estos conllevan, haciendo que si salta un error de la clase “Error” sea algo más grave, que está fuera del alcance del programador y que no se deben manejar explícitamente. En cambio si es de la clase Exception, pueden ser manejables y son de menor gravedad.

2. Jerarquía de excepciones en Java

Throwable, Exception, RuntimeException...

Dentro de la clase Exception heredan dos clases RuntimeException que son las denominadas “Unchecked exception” (Los que heredan de la clase Error también son checked exception) y las demás que son “Checked exception”.

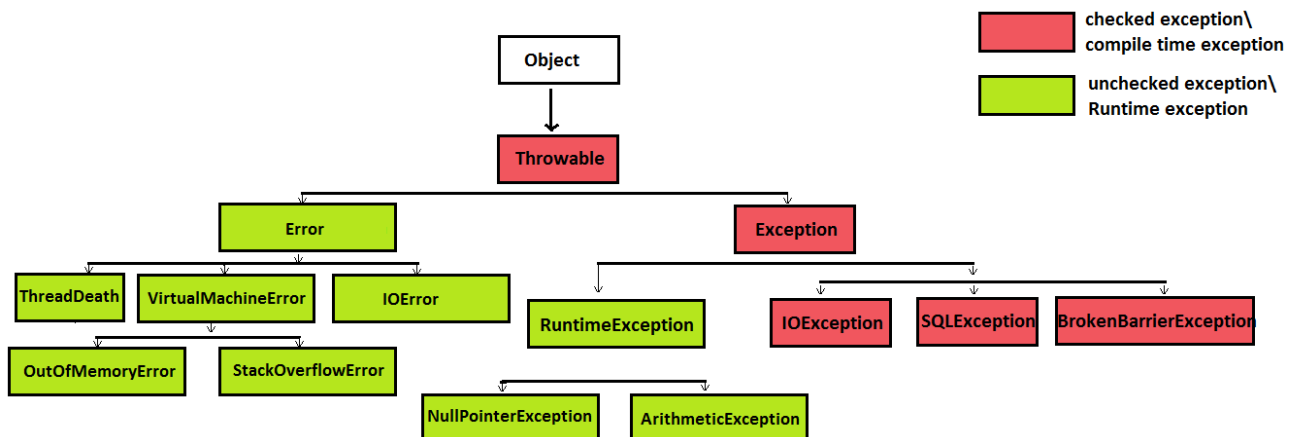


Figura 1: Jerarquía de la clase Throwable

Diferencias entre excepciones comprobadas (checked exceptions) y excepciones no comprobadas (unchecked exceptions)

La diferencia que podemos encontrar entre estos 2 tipos de excepciones es que si son unchecked, no hace falta manejarla de manera obligatoria, en cambio si es checked, si debemos manejarla ya sea con un bloque try-catch o con un throws

En el siguiente ejemplo se puede ver como al ser unchecked, no es obligatorio mencionarlo antes en el código.

```
import java.util.Scanner;

public class test {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Introduzca un número");
        int numero = scanner.nextInt();
    }
}
```

Figura 2: Código de ejemplo excepción unchecked

```
Introduzca un número
a
Exception in thread "main" java.util.InputMismatchException
    at java.base/java.util.Scanner.throwFor(Scanner.java:964)
    at java.base/java.util.Scanner.next(Scanner.java:1619)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2284)
    at java.base/java.util.Scanner.nextInt(Scanner.java:2238)
    at test.main(test.java:7)
```

Figura 3: Salida por consola excepcion unchecked

En cambio si es checked sí.

```
3 usages
public static Document getDocument(String route) throws ParserConfigurationException, IOException, SAXException {
    File file = new File(route);
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document doc;
    doc = builder.parse(file);
    doc.getDocumentElement().normalize();

    return doc;
}
```

Figura 4: Código de ejemplo de excepciones checked

3. Estructuras de manejo de excepciones

¿Qué bloques podemos utilizar para manejar las excepciones?

Para manejar estas excepciones, Java nos proporciona diferentes bloques para poder gestionarlas a nuestro gusto:

- **Bloque try-catch:** Este bloque hace que podamos capturar las excepciones y que tengamos más flexibilidad a la hora de manejarlas. Esto funciona como 2 bloques anidados, el bloque Try en el que pondremos el código que pueda hacer saltar la excepción y el bloque Catch para poder capturarla y manejarla si salta.

```

import java.util.InputMismatchException;
import java.util.Scanner;

public class test {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Introduzca un número");
            int numero = scanner.nextInt();
        } catch (InputMismatchException e) {
            System.out.println("Error: Debes introducir un número.");
        }
    }
}

```

Figura 5: Bloque try-catch ejemplo

```

Introduzca un número
a
Error: Debes introducir un número.

Process finished with exit code 0

```

Figura 6: Bloque try-catch ejemplo salida por consola

- **Finally (Opcional):** Este bloque hace que independientemente de si salta la excepción o no, el código que esté dentro de este bloque se ejecutará siempre. Este bloque es muy útil y se suele utilizar para cerrar bases de datos o streams aunque salte una excepción.

```

import java.util.InputMismatchException;
import java.util.Scanner;

public class test {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Introduzca un número");
            int numero = scanner.nextInt();
        } catch (InputMismatchException e) {
            System.out.println("Error: Debes introducir un número.");
        } finally {
            System.out.println("Código dentro del finally que se ejecuta igualmente.");
        }
    }
}

```

Figura 7: Bloque finally ejemplo

```

Introduzca un número
a
Error: Debes introducir un número.
Código dentro del finally que se ejecuta igualmente.

Process finished with exit code 0

```

Figura 8: Bloque finally ejemplo salida por consola

- **Throw:** Este bloque nos sirve para lanzar por nuestra cuenta una excepción. Esto proporciona más flexibilidad a nuestro código para que se cumplan determinadas funciones y también poder reutilizar código.

```

public class test {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.println("Introduzca un número");
            int numero = scanner.nextInt();
            if (numero < 0) {
                throw new InputMismatchException();
            }
        } catch (InputMismatchException e) {
            System.out.println("Error: Debes introducir un número mayor que 0.");
        } finally {
            System.out.println("Código dentro del finally que se ejecuta igualmente.");
        }
    }
}

```

Figura 9: Bloque throw ejemplo

```

Introduzca un número
-1
Error: Debes introducir un número mayor que 0.
Código dentro del finally que se ejecuta igualmente.

Process finished with exit code 0

```

Figura 10: Bloque throw salida por consola

- **Throws:** Esta palabra clave se utiliza para indicar que un método puede lanzar una o varias excepciones en concreto. Esto se utiliza que, si el programador quiere utilizar ese método, sea avisado de la excepción que puede saltar y así que pueda gestionarla como él desee.


```

3 usages
public static Document getDocument(String route) throws ParserConfigurationException, IOException, SAXException {
    File file = new File(route);
    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document doc;
    doc = builder.parse(file);
    doc.getDocumentElement().normalize();

    return doc;
}

```

Figura 11: Bloque throws ejemplo

4. Tipos comunes y excepciones de Java

Los tipos de excepciones más comunes que pueden aparecer son los siguientes:

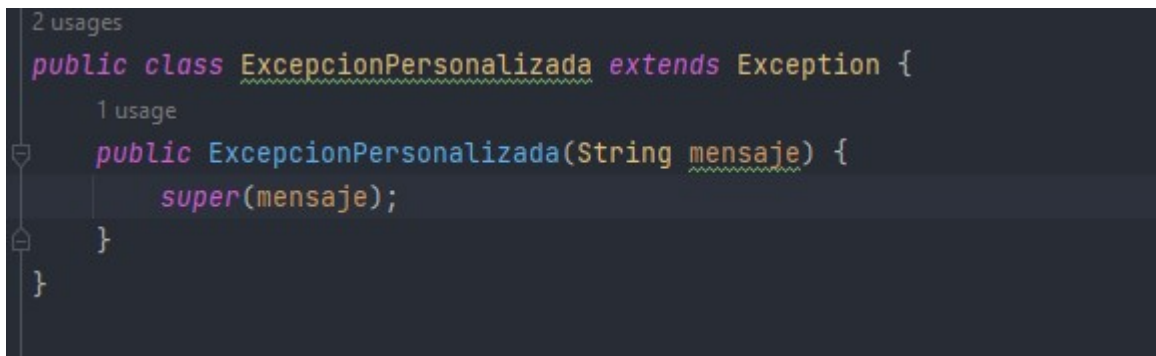
- **NullPointerException (Unchecked):** Esta excepción salta cuando la aplicación intenta usar un valor de un objeto que es nulo. Para evitar que ocurra esto, debemos asegurarnos que el Objeto al que se referencia no tiene un valor nulo.
- **ArrayIndexOutOfBoundsException (Unchecked):** Esta excepción salta cuando se intenta acceder a un índice que no sirve en un array o cuando el índice es más grande o más pequeño que la longitud total de un array. Para evitar esto, debemos tener cuidado con los índices que manejamos en los arrays, especialmente cuando lo recorremos con un bucle.
- **IOException (Checked):** Esta excepción salta cuando ocurre algún tipo de error en la entrada/salida de secuencias, archivos o directorios. Por ejemplo, cuando ponemos la ruta de un archivo mal y el programa no lo encuentra.
- **SQLException (Checked):** Esta excepción engloba errores producidos relacionados con el acceso a la base de datos o al consultar, modificar, etc. Por ejemplo, es bastante típico que esta excepción salte cuando intentamos hacer una consulta SQL y tenemos un error de sintaxis.
- **NumberFormatException (Unchecked):** Esta excepción salta cuando la aplicación intenta convertir un String a un número pero el String no es numérico. Para evitar esto, tenemos que asegurarnos de que el valor del String al que se intente hacer el cast sea numérico.

5. Buenas prácticas para el manejo de excepciones

- Intentar usar los tipos de excepción predefinidos.
- Colocar un String describiendo el mensaje en el bloque catch. Podemos usar también el método `printStackTrace()`; para que nos describa con más detalle el error.
- Utilizar nombres claros para las excepciones.
- No utilizar demasiados bloques try catch anidados. Si pasa esto, utilizar un tipo de excepción más general para que recoja todos o modificar el código de alguna manera para intentar evitar esto.
- Si la excepción es ambigua, utilizar los comentarios para aclarar lo que hace y en que caso saltaría.

6. Excepciones personalizadas

Java nos proporciona la ventaja de poder crear nuestras propias excepciones creando una clase nueva que herede de `Exception`. Esto puede resultar muy útil para aquellas excepciones que se den en un contexto muy específico de nuestro código que no encaje realmente con ninguno de los tipos predefinidos de excepciones que hay. Esto también puede generar que el código sea más limpio y legible.



```
2 usages
public class ExcepcionPersonalizada extends Exception {
    1 usage
    public ExcepcionPersonalizada(String mensaje) {
        super(mensaje);
    }
}
```

Figura 12: Construcción de la excepción personalizada

```
import java.util.Scanner;

public class test {
    public static void main(String[] args) throws ExcepcionPersonalizada {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Introduzca un número");
        int numero = scanner.nextInt();
        if (numero < 0) {
            throw new ExcepcionPersonalizada(mensaje: "Error número debe ser mayor que 0.");
        }
    }
}
```

Figura 13: Ejemplo de uso de excepción personalizada

```
Program Files (x86)\jdk-22\bin\java.exe -jar C:\Program Files (x86)\JetBrains\IntelliJ IDEA\bin\idea.exe
Introduzca un número
-3
Exception in thread "main" ExcepcionPersonalizada: Error número debe ser mayor que 0.
    at test.main(test.java:10)

Process finished with exit code 1
```

Figura 14: Salida por consola de la excepción personalizada

7. Manejo avanzado de excepciones

Hay varias formas más para poder tratar las excepciones, una de ellas es anidar bloques catch en un mismo bloque try-catch. Esto se puede hacer cuando un mismo bloque de código puede producir varias excepciones. Para poder hacer esto de forma correcta, debemos colocar los bloques catch en un orden que sea de más específico a más general.

También tenemos try-with-resources, un bloque try catch especial que de forma automática libera recursos como conexión a base de datos, archivos, etc.

8. Conclusión

Manejar las excepciones en un código es fundamental para su correcto funcionamiento, no podemos dejar un código con excepciones sin tratar ya que eso va a repercutir de manera negativa en el uso

por parte del usuario final. Además de esto, puede provocar que sea más robusto, seguro y más limpio pudiendo dividir el código de error con el resto.

Bibliografía

Java exception API: <https://docs.oracle.com/javase/8/docs/api/index.html?java/lang/Exception.html>

Java Throwable API: <https://docs.oracle.com/javase/8/docs/api/java/lang/Throwable.html>

Introducción a POO en Java: Excepciones: <https://openwebinars.net/blog/introduccion-a-poo-en-java-excepciones/>

Checked vs Unchecked exceptions in Java: <https://www.geeksforgeeks.org/checked-vs-unchecked-exceptions-in-java/>

Qué es un NullPointerException y cómo solucionarlo: <https://elbaultdelprogramador.com/que-es-un-nullpointerexception-y-como-solucionarlo/>

Java SQLException API: <https://docs.oracle.com/en/java/javase/11/docs/api/java.sql/java/sql/SQLException.html#:~:text=An%20exception%20that%20provides%20information,a%20string%20describing%20the%20error.>

Java NumberFormatException API:

<https://docs.oracle.com/javase/8/docs/api/java/lang/NumberFormatException.html>

Best practices for exceptions: <https://learn.microsoft.com/en-us/dotnet/standard/exceptions/best-practices-for-exceptions>

The try-with-resources statement:

<https://docs.oracle.com/javase/tutorial/essential/exceptions/tryResourceClose.html>