Manejo de Excepciones

¡Comencemos!

¿Qué aprenderás?

- Comprender las excepciones de Java para optimizar nuestras aplicaciones.
- Aplicar Excepciones utilizando Try-Catch y “throw” para controlar y capturar las excepciones.

Introducción

En este capítulo conoceremos todo sobre las excepciones: que son, para qué nos sirven y cómo manejarlas. En el mundo de la programación manejar las excepciones nos permite crear un sistema robusto y confiable donde la continuidad del software no se verá afectada por errores de programación como por errores de los usuarios que lo utilizan. El control de las excepciones nos ayudará a comprender el uso de las sentencias Try-Catch porque las excepciones van de la mano con las palabras claves de Java.

¡Vamos con todo!

Excepciones en Java

Cuando hablamos de excepciones en Java (o cualquier lenguaje que las ocupe), hablamos de control de errores en ejecución del programa; esto quiere decir que las excepciones nos permiten controlar errores de usuarios o de programación. Java nos ofrece una cantidad de excepciones para utilizar, listas para su uso.

Las excepciones más utilizadas en programación son las de tipo aritméticas, de validaciones de nulos y de archivos para controlar los errores. Las excepciones tienen jerarquía entre ellas. A continuación mostramos una imagen que ilustra tal orden:

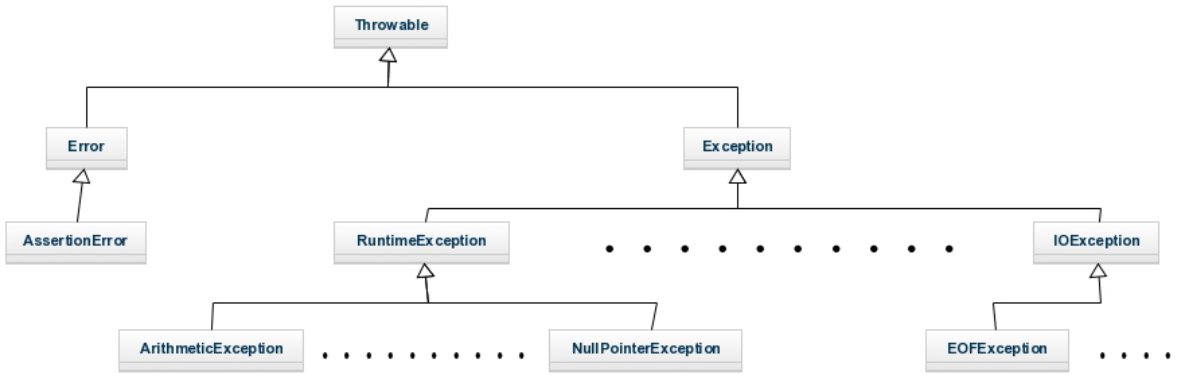

Imagen 1. Jerarquía en las Excepciones
Fuente: Desafío Latam.

Throwable

Es una clase base que representa todo lo que Java puede "lanzar", de hecho la palabra *throwable* se puede considerar en el español como "lanzar o arrojar". A su vez esta clase:

- Almacena un mensaje (variable de instancia de tipo String) que podemos utilizar para detallar qué error se produjo.
- Puede ser una causa, también de tipo Throwable, que permite representar el error que causó este error.

Tenemos dos grandes clases que nos permiten controlar errores y excepciones :

Error

Esta clase nos indica que el error es a nivel de hardware y no aplicativo, uno de los errores más comunes es el error de memoria. Por ejemplo, cuando un proceso toma más del tiempo habitual en ejecutarse y colapsa la memoria del programa.

Ejemplos: Memoria agotada, error interno de la JVM…

Exception

Por otro lado, tenemos todas las excepciones que podemos controlar a nivel de usuario y programación, con este tipo de Exception manejamos la mayor cantidad de errores controlables del software.

Exception y sus subclases indican situaciones que una aplicación debería tratar de forma razonable.

Los dos tipos principales de excepciones son:

- **RuntimeException**: Errores del programador, como una división por cero o el acceso fuera de los límites de un array.
- **IOException**: Errores que no puede evitar el programador, generalmente relacionados con la entrada/salida del programa.

Las excepciones que nos ofrece Java tiene métodos listos para su uso, algunos son:

1. **getMessage()**: Este método nos permite mostrar el error y dónde está pasando, esto quiere decir la línea de la clase de Java o el tipo de error.
2. **printTrace()**: Este método nos permite mostrar el error con más detalle, sin embargo utilizarlo en todo el código nos provoca un mayor uso de la memoria de la JVM, lo cual es recomendable utilizarlo cuando hay un error en particular que no se logra detectar.

Excepciones personalizadas

Java nos ofrece su clase Exception para su uso, pero también nos ofrece heredar de esta clase para crear nuestras propias clases con nuestros propios métodos. Esto nos permite controlar las excepciones según funcionalidad, evento o lógica en el programa.

```
public class MiExcepcion extends Exception {

	public MiExcepcion(String arg) {
		super(arg);
	}

	public String validaNulo(String arg) {
		String mensaje ="";
		if(arg == null) {
			mensaje= "campo nulo";
		}
		return mensaje;
	}

}
```

En este ejemplo tenemos el uso de Herencia y la clase `MiExcepcion` que hereda de Exception, sobreescribimos el constructor y creamos un método llamado "`validaNulo`", el cual valida si el campo es nulo.

Try-Catch

El uso de estas palabras nos permite relacionarlas con las excepciones ya que su uso siempre va ligado a este concepto.

Try

Dentro de este bloque escribiremos todo el código que debiese funcionar sin problemas, podemos hacer llamadas a otras clases, validaciones y escribir todo el código que debiese funcionar.

Esta palabra va acompañada con el uso de llaves de apertura y cerrado, y siempre se utiliza con la otra palabra reservada de Java Catch.

Catch

En esta cláusula es donde lanzamos las posibles excepciones del bloque de código dentro del `try`. Pueden existir tantos `catch` como Exception se quiera controlar, esto quiere decir que para utilizar Catch debemos siempre utilizar el nombre de la Excepción dentro del paréntesis, al lado de la palabra, y, a continuación, abrir y cerrar bloque de código con el uso de llaves como se muestra en la imagen a continuación.

Para ejemplificar un poco, usaremos una división por cero la cual, al ejecutarse, el código podría llegar a fallar. Si se dan las condiciones para que esta porción de código falle, se va a ejecutar otra porción de código (`catch`) para seguir ejecutando el programa y controlar este error.

Ejemplo: División por 0

```
try {
  int total = 3 / 0;
  }catch(Exception excepcion) {
  int total = 0;
  }
```

Se puede utilizar más de un catch a la vez, por ejemplo, si sabemos qué excepciones podría arrojar un bloque `try`, podemos agregar un `catch` con cada subclase de Exception que consideremos necesaria.

```
    try {
      Scanner sc = new Scanner(System.in);
      String variable = sc.nextLine();
       if(variable.isEmpty()){
        variable = null;
     }
      int total = 3 / Integer.parseInt(variable);
     }catch(NullPointerException ex1) {
     System.out.println("No se puede dividir por un valor nulo.");
       int total = 0;
     }catch(NumberFormatException ex2) {
     System.out.println("El valor de variable no es un número.");
       int total = 0;
     }catch(Exception ex3){
     System.out.println("Error inesperado: "+ex3.getMessage());
       int total = 0;
}
```

En este caso, el bloque catch a ejecutar dependerá del valor que ingrese el usuario dentro de la variable. Si el usuario no ingresa nada, el valor será null y se ejecutará el catch con NullPointerException y si, por ejemplo, ingresa una letra o un valor no numérico, se arrojará NumberFormatException y en cualquier otro caso, se arrojará el catch con Exception con una subclase de Exception.

La cláusula finally

En ocasiones, nos interesa ejecutar un fragmento de código independientemente de si se produce o no una excepción, si agregamos un `Finally` en el bloque Try-Catch, este se ejecutará siempre, ya que es independiente si se ejecuta el `try` o el `catch`. Se utiliza por ejemplo cuando se debe cerrar un fichero que estemos manipulando o cerrar una conexión de base de datos para liberar recursos.

Ejemplo de una conexión a una fuente de datos usando la cláusula finally:

```
Connection cn = null;
  try {
    cn = fuenteDeDatos.getConnection();
   }catch(Exception e){
    System.err.out("Ha ocurrido un error");
   } finally {
    cn.close();
}
```

Otro ejemplo con código:

```
public void validaEdad(String arg) {
            String mensaje =”prueba”
            try {
                  if((Integer.parseInt(arg)) >=18)
                  {
                  System.out.println("Edad es mayor a 18 y un número" +
mensaje);
                  }
            }
            catch (NumberFormatException  e) {
                   System.out.println(e.getMessage() + mensaje);
            }
      }
```

- Entrará al Catch siempre y cuando el parámetro sea algo distinto de un número.
- El código `Integer.parseInt(arg)` convierte un String a un int.
- Si lo ingresado es algo distinto a número, lanza una excepción de tipo `NumberFormatException`.

Este tipo de control nos permite manejar nuestras aplicaciones con errores controlados y así evitar que el programa se detenga y no se pueda utilizar.

Importante

Cabe mencionar que cada variable que se escriba dentro de los bloques de llave de apertura y cerrado, solo será vista y podrá utilizarse ahí. Se recomienda que al utilizar variables se declaren antes del bloque try-catch.

Throw

Las excepciones son errores durante la ejecución de un código y en la mayoría de los casos nos muestran la línea de código donde se originó el problema. Para lanzar un objeto de tipo Exception se necesita utilizar la palabra reservada de Java “Throw”, esta nos permite lanzar exception en ejecución y capturar la ejecución de un objeto.

Cuando se lanza una excepción:

1. Se sale inmediatamente del bloque de código actual.
2. Si el bloque tiene asociada una cláusula catch adecuada para el tipo de la excepción generada, se ejecuta el cuerpo de la cláusula catch.
3. Si no, se sale inmediatamente del bloque (o método) dentro del cual está el bloque en el que se produjo la excepción y se busca una cláusula catch apropiada.
4. El proceso continúa hasta llegar al método main de la aplicación. Si ahí tampoco existe una cláusula catch adecuada, la máquina virtual Java finaliza su ejecución con un mensaje de error.

```
public void validaEdad(String arg) {
      try {
            if((Integer.parseInt(arg)) >=18)
            {
                System.out.println("Edad es mayor a 18");
            }
      }
      catch (NumberFormatException  e) {
            throw new NumberFormatException(e.getMessage());
      }
}
```

Ejercicio Guiado: Validaciones con Try-Catch

Crear un método que nos permita dividir dos parámetros de tipo String y retornar el resultado de la división.

Explicación

Paso 1: Crear la variable local de método resultado.

Paso 2: Realizar la conversión de datos de String a Int dentro del bloque try.

Paso 3: Utilizar el primer catch con la Exception `NumberFormatException`, esta controlará la excepción cuando uno de los dos parámetros sea algo distinto a un número.

```
java.lang.NumberFormatException: Formato de número incorrecto  :For input string: "dos"
```

Paso 4: Utilizar el segundo Catch con la Exception `ArithmeticException` . Esta controlará la Exception cuando uno de los dos parámetros sea cero, dando la división de una excepción de tipo aritmética, por ejemplo 0/2.

```
java.lang.ArithmeticException: Error en aritmetico   :  / by zero
```

Paso 5: Si no entra a ninguna Exception, retorna el resultado.

```
public static void main (String [] args) {
            division("22","0");
           }
      public static  int  division(String valorUno, String valorDos) {
            int resultado = 0;
            try {
                  int uno = Integer.parseInt(valorUno);
                  int dos = Integer.parseInt(valorDos);
                  resultado  = uno/dos;
            }
            catch (NumberFormatException  e) {
      //se lanza cuando el parámetro  sea  distinto a una numero
                   throw new NumberFormatException("Formato de número
incorrecto  :" + e.getMessage());
            }
            catch (ArithmeticException  e) {
      // se lanzará cuando el parámetro sea un cero
                  throw new ArithmeticException("Error en aritmética   :
" +e.getMessage());
            }
            return resultado;
}
```