Excepciones en Java

ÍNDICE

1. Introducción

A la hora de escribir programas, uno de los mayores esfuerzos del programador debe ser evitar que se produzcan errores durante la ejecución. A pesar de esos esfuerzos, los errores suelen ocurrir. Algunos lenguajes de programación responden a esos errores con la finalización repentina de la ejecución del programa o incluso puede que la ejecución continúe pero de una forma impredecible. El lenguaje Java usa las **excepciones** para el tratamiento y manipulación de los errores.

Una excepción es una situación anómala a la que llega la ejecución de un programa. Estas situaciones anómalas pueden ser el intento de abrir un fichero que no existe, la división por cero, acceder a un índice que no existe en un array o el acceso a un objeto no inicializado (el famoso `null`).

2. Concepto de excepción

En este momento ya tenemos claro que una excepción es un suceso que ocurre durante la ejecución del programa y que rompe el flujo normal del mismo. Ejemplos de excepciones son intentar acceder a un elemento de una matriz con un índice mayor que su dimensión o convertir a entero un texto.

Cuando ocurren dichas situaciones, se crea una excepción y el control del programa pasa desde el punto donde el error se produjo a otro punto especificado por el programador. En tal caso, se dice que la excepción es "lanzada" (*throw*) desde el punto donde se creó y es "recogida" (*catch*) en el punto donde se transfiere el control. Java hereda la terminología y la sintaxis del tratamiento de excepciones del lenguaje C++.

Toda excepción es representada por un objeto de la clase `Throwable` o cualquier subclase suya (heredada). Este objeto contiene información sobre el tipo de la excepción y el estado del programa cuando se produjo. Además, puede ser utilizado para enviar información desde el punto donde el error se produjo hasta el punto donde será tratado.

Una vez que una excepción ha sido lanzada desde el interior de un método, el sistema se encarga automáticamente de buscar un manipulador de excepciones (*exception handler*) que sea capaz de responder a esa excepción. La búsqueda comienza en el método donde se produjo el error y en caso de no encontrarse dicho manipulador, la búsqueda continúa en el método en el que se hizo la llamada al método que provocó la excepción y así sucesivamente.

De esta forma, se produce una búsqueda ascendente hasta encontrar el manipulador apropiado. Un manipulador de excepciones es considerado apropiado si coinciden los tipos de la excepción producida y la excepción a la que responde el manipulador. En el caso de que el sistema termine la búsqueda y no sea capaz de encontrar el manipulador adecuado, la ejecución del programa terminará de forma no controlada.

3. Excepciones en Java

Java proporciona un mecanismo para detectar y solucionar las excepciones que se puede llegar a producir durante la ejecución de un programa. En Java estamos obligados a tratar las excepciones cuando se producen, bien gestionándolas directamente o bien desentendiéndonos de ellas, cediendo su manejo a un nivel superior, pero hasta esto último debemos hacerlo explícitamente en determinados casos.

En Java existen dos grandes tipos de situaciones no deseadas: los errores y las excepciones.

- Los **errores** son situaciones irrecuperables, por ejemplo fallos de la máquina virtual. Ante ellos no hay más alternativa que cerrar la aplicación, no estando obligados a gestionarlas.
- Las **excepciones** son situaciones anómales ante las cuales bien debemos reaccionar o bien nos desentendemos explícitamente. Cuando una excepción se produce se acompaña de toda la información relevante para que podamos gestionarla.

En Java existen muchos tipos de excepciones estándar. Algunas de las excepciones más comunes o usuales se muestran a continuación.

- `IOException`: hay un problema en la entrada o salida de los datos. Por ejemplo, al leer un fichero.
- `ArithmeticException`: provocada por una operación aritmética ilegal. Por ejemplo, división entre cero.
- `FileNotFoundException`: se produce cuando no se encuentra un fichero.
- `EOFException`: se lanza cuando se llega al final del fichero y se pretende leer más.
- `ClassCastException`: conversión de tipo ilegal.
- `IllegalArgumentException`: se produce al realizar la llamada a un método con un argumento inapropiado.
- `NumberFormatException`: intento de convertir una cadena de texto, que no tiene el formato adecuado, a un determinado tipo numérico.
- `IndexOutOfBoundsException`: índice fuera de rango.
- `NegativeArraySizeException`: intento de crear un array cuya longitud es negativa.
- `NullPointerException`: uso de una referencia nula (`null`) donde se requería la referencia a un objeto.

4. Causas de las excepciones

En un programa Java las excepciones pueden provocarse por el propio sistema cuando detecta un error o una condición de ejecución anormal, o explícitamente por el programador, utilizando la sentencia **throw**. Dentro del primer caso podrían identificarse las siguientes causas:

- Un índice fuera de los límites en una matriz.
- Un error cuando se carga una parte del programa.
- Un exceso en el uso de un cierto recurso. Por ejemplo, agotar la memoria en un bucle infinito.

5. Tratamiento de las excepciones

Una vez analizada la importancia de las excepciones, algunos de sus principales tipos y la necesidad de responder (en algunos casos) a cada excepción, esta sección se dedica a la exposición de cómo hacerlo.

Para ello, vamos a partir del programa siguiente, en el que se implementa el código necesario para hacer una división entera entre dos valores y mostrar el resultado por pantalla.

```
private static int dividir(int dividendo, int divisor) {
	return dividendo / divisor;
}

public static void main(String[] args) {
	int dividendo = solicitarEnteroAlUsuario();
	int divisor = solicitarEnteroAlUsuario();

	int cociente = dividir(dividendo, divisor);
	System.out.print("El cociente es " + cociente);
}
```

¿Qué ocurrirá en el método *dividir* si divisor toma el valor 0? Se producirá una excepción del tipo `ArithmeticException`.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at apuntes.DivisionApp.dividir(DivisionApp.java:7)
        at apuntes.DivisionApp.main(DivisionApp.java:15)
```

Sin embargo, el compilador no avisa respecto a que esa excepción pueda ocurrir y, por lo tanto, no es necesario tratarla *a priori*. En caso de que el programador desee no hacerlo, deberá asumir los riesgos que eso supone.

¿Cómo debemos actuar para evitar que se produzcan excepciones en nuestro código y que se pare la ejecución del programa? Capturando la excepción mediante un manejador de excepciones (*exception handler*).

5.1 Estructura de un manejador de excepciones

Todo manejador de excepciones tiene tres bloques básicos:

- El bloque **`try`**.
- El bloque o bloques **`catch`.**
- El bloque **`finally`**.

a) El primer bloque, bloque **`try`**, agrupa a todas las sentencias del método que pueden provocar una excepción.

b) Puede haber uno o varios bloques **`catch`** inmediatamente a continuación de cada bloque **`try`** para capturar la excepción o excepciones deseadas.

c) Por último, el bloque **`finally`** es opcional y permite ejecutar código adicional tras la ejecución (o no) de los anteriores bloques.

5.2 Implementación de un manejador de excepciones

A continuación, vamos a ver un ejemplo de código en el que capturamos la excepción `ArithmeticException` y cualquier otra (`Exception`) en el método dividir. Si se produce una excepción devolvemos el valor del divisor.

```
private static int dividir(int dividendo, int divisor) {
	int cociente = Integer.MIN_VALUE;

	try {
		cociente = dividendo / divisor;
	}
	catch(Exception ex) {
		System.out.println("Se ha producido una excepción no contemplada.");
	}

	return cociente;
}
```

Cada bloque `catch` requiere un argumento que se declara de la misma forma que la declaración de argumentos de un método. Este argumento, que acompaña a la sentencia `catch`, indica el tipo de excepción que es tratada en el correspondiente bloque y siempre debe ser una subclase de la clase `Throwable`, que se encuentra definida en el paquete `java.lang`.

Además de la clase de la excepción debe darse un nombre que puede ser usado para referirse a ella en las sentencias del bloque. A través de este nombre puede accederse a variables miembro o métodos de la clase de forma similar a como se haría en cualquier otra parte del método. Uno de tales métodos es `getMessage()`, el cual escribe información adicional sobre el error que provocó la excepción.

El anterior código lo podemos ampliar de la siguiente forma para incrementar la información ofrecida sobre la excepción correspondiente.

```
private static int dividir(int dividendo, int divisor) {
   int cociente = Integer.MIN_VALUE;

   try {
          cociente = dividendo / divisor;
   }
   catch(ArithmeticException ex) {
          System.out.println("Se ha producido una excepción aritmética.\n" + ex.getMessage());
   }
   catch(Exception ex) {
          System.out.println("Se ha producido una excepción no contemplada.\n" + ex.getMessage());
   }

   return cociente;
}
```

El orden en que se coloquen los distintos bloques `catch` es importante, ya que cuando se produce la excepción en una instrucción, el sistema abandona el flujo de ejecución normal y busca automáticamente el primer bloque `catch` cuyo tipo de excepción se ajusta al de la excepción que se ha producido ejecutando el código de dicho bloque `catch` únicamente (y no el de otros bloques `catch` que pudiesen existir) y las sentencias del bloque `finally` (si existe). La búsqueda del bloque comienza en el mismo método en el que produjo la excepción y continúa por los métodos de los que provenía la llamada.

6. Lanzamiento de excepciones

Una alternativa a la construcción de un manejador para una excepción es dejar que un método anterior en la sucesión de llamadas se encargue de responderla, utilizando el comando `throws`.

Cuando se quiere que un método declare varias excepciones pero que no responda a ellas con el o los correspondientes manejadores de excepciones, debe colocarse, a continuación del nombre del método y de la lista de sus argumentos, la palabra clave **`throws`** (con una ese al final) seguida de una lista separada por comas, de las excepciones.

Por ejemplo, un método definido de la forma siguiente, indica que en el método puede generarse una excepción `IndexOfBoundsException`, pero en ese caso, el bloque **`catch`** para tratarla debe buscarse en el método que hizo la llamada al método `S()` y no en el propio método `S()`:

```
public void S (int []) throws IndexOutOfBoundsException {
....
}
```

La sentencia **`throw`** (sin la ese al final) se utiliza dentro de los métodos para "lanzar" excepciones, es decir, cuando se detecta una situación que provocaría un error, debe crearse un objeto de la clase de excepción correspondiente e indicar que la excepción se ha producido mediante una sentencia de la forma:

```
throw Objeto;
```

donde *Objeto* debe pertenecer a alguna de las subclases de la clase `Throwable`.

Supongamos que queremos que únicamente se puedan utilizar valores positivos en los parámetros de *dividendo* y *divisor*. En el caso de que no sea así, deseamos lanzar una excepción. Nuestro código podría quedar de la siguiente forma:

```
private static int dividir(int dividendo, int divisor) throws Exception // IllegalArgumentException
{
	if(dividendo <= 0 || divisor <= 0) throw new Exception("Los valores del dividendo y el divisor deben ser positivos");

	return dividendo / divisor;
}

public static void main(String[] args) {
	int dividendo = solicitarEnteroAlUsuario();
	int divisor = solicitarEnteroAlUsuario();
	int cociente;

	try {
		cociente = dividir(dividendo, divisor);
		System.out.print("El cociente es " + cociente);
	}
	catch(Exception ex) {
		System.out.println("Se ha producido una excepción");
		ex.printStackTrace();
	}
	finally {
		System.out.println("Fin del programa. Siempre se ejecuta, haya o no haya excepción");
	}
}
```

Si el usuario introduce un valor inferior o igual a 0 en el dividendo o en el divisor, se producirá un mensaje como el siguiente.

```
Se ha producido una excepción
java.lang.Exception: Los valores del dividendo y el divisor deben ser positivos
	at apuntes.DivisionApp_v2.dividir(DivisionApp_v2.java:7)
	at apuntes.DivisionApp_v2.main(DivisionApp_v2.java:20)
Fin del programa. Siempre se ejecuta, haya o no haya excepción
```

7. Creación de excepciones propias

Toda excepción es un objeto de la clase `Throwable` o de alguna de sus subclases, bien sean incorporadas por librerías de Java o declaradas en el propio programa.

Como norma recomendada o buena práctica, los nombres de las clases de excepciones deben terminar con la palabra `Exception`. El programador puede crear su propios tipos de excepciones construyendo subclases de la clase `Throwable` que se adapten mejor a sus necesidades particulares.

Para crear una excepción propia debemos extender la clase `Exception` o la excepción más adecuada (por ejemplo `ArithmeticException`) y en el constructor de la clase llamar a la clase padre con el mensaje que se desee mostrar cuando se produzca la excepción.

Para lanzar una excepción explícitamente, utilizamos la palabra reservada `throw` e indicamos en la declaración del método que puede lanzar la excepción deseada.

Supongamos que estamos implementando un programa que únicamente puede ser usado por mayores de edad. En el caso de detectar que un usuario no es mayor de edad, queremos lanzar una excepción personalizada que nos indique esta circunstancia. Podemos implementar nuestra excepción de la siguiente forma:

```
public class MayorEdadException extends Exception
{
      private final static int LIMITE_MAYORIA_EDAD = 18;

      public MayorEdadException() {
        super("El usuario tiene menos de " + LIMITE_MAYORIA_EDAD + " años");
    }
}
```

Para utilizar dicha excepción podemos simular un código como el siguiente en el que se comprueba si el usuario es mayor de edad.

```
Usuario usuario = new Usuario();
if(usuario.getEdad() < 18) throw new MayorEdadException();
```