Unidad 10

Control y manejo de excepciones

Programación básica

Índice

Introducción

Una parte fundamental en el desarrollo de un programa es tener controlado cualquier situación anómala que pueda producirse en nuestro programa. Imaginemos que un usuario esta rellenando un formulario y al enviarlo se produce un error por introducir incorrectamente los datos solicitados, si nuestro programa no es capaz de responder en exactitud ante ese error y el usuario no puede conocer cuál ha sido el error, tenemos un programa que no cumple con su función. Para ello disponemos de las excepciones para captar cualquier error que se pueda producir, dar un tratamiento a ese error para corregirlo o realizar alguna actuación para interrumpir el proceso.

Al finalizar esta unidad

- Conoceremos que son las excepciones, cuando se producen y como tratarlas.
- Expondremos su clasificación y jerarquía.
- Los diferentes tratamientos que se pueden dar cuando sucede una excepción.
- Sabremos recomendaciones de uso para aplicar correctamente las excepciones

10.1. Excepciones

Una excepción es un evento ante una situación no prevista por un error funcional o lógico, unos parámetros con valores no definidos, un flujo incorrecto, una acción no prevista o cualquier problema externo de conectividad, hardware, etc. Que se produce en tiempo de ejecución. Por ejemplo, no se pueda establecer una conexión a una base de datos, no se pueda abrir un fichero, el programa se quede sin memoria, se tratarte un tipo de dato incorrecto, intentar acceder a una posición de un array inexistente, etc. Las excepciones son gestionadas en la mayoría de los lenguajes de programación, en concreto, java ya dispone de unas excepciones predefinidas que se explicará en los siguientes puntos.

Cuando se ejecuta un programa se empieza desde la clase que contiene el método main y se establece un flujo de llamadas a otras clases y métodos según la funcionalidad, denominado pila de llamadas. Cuando se produce una excepción en cualquier punto del código se propagará por toda la pila hasta llegar al método main y finalizar el programa.

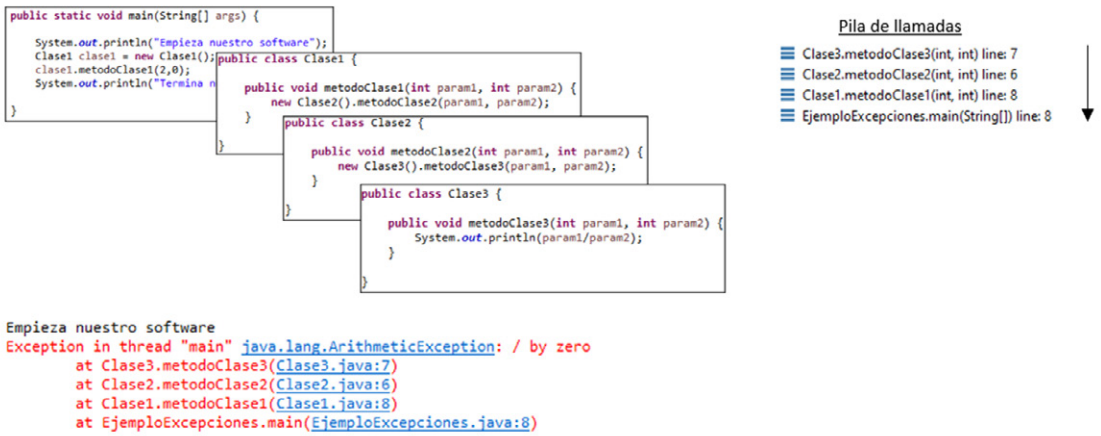

Imagen 1. Propagación de una excepción al dividir por cero

Para evitar que cada vez que se produzca una excepción, recorra toda la pila de llamadas y termine el programa, se realizará un tratamiento de excepciones para realizar acciones de corrección o alternativas para seguir con el correcto funcionamiento del programa.

Hacer un bueno desarrollo y control de excepciones permitirá tener un programa más robusto y evitará cualquier anomalía que haga que no funcione correctamente. Desde la etapa de codificación es recomendable tener un listado de todos los puntos donde puede producirse una excepción y diseñar su tratamiento si es necesario. Para ello es recomendable:

- Tener listado todas las excepciones posibles.
- Poner un identificador exclusivo de cada excepción.
- Saber el flujo de llamadas y como se propagará la excepción
- Separar el código destinado a controlar errores del resto de código.

10.2. Jerarquía de excepciones

En java por defecto se dispone de un conjunto de excepciones predefinidas con las que controlar ciertas situaciones, aun así, se puede definir nuestras propias excepciones.

Excepción	Descripción error
FileNotFoundException	No encuentra un fichero
ClassNotFoundException	No existe o no encuentra una clase
EOFException	Se intenta acceder a una parte final de un fichero
ArrayIndexOutOfBoundException	Posición inexistente de un array
NumberFormatException	Número en otro tipo de formato no esperado
NullPointerException	Se intenta acceder a un objeto no inicializado
IOException	Error de entrada o salida
ArithmeticException	Operación aritmética incorrecta
OutOfMemoryError	No hay suficiente memoria para su ejecución
StackOverFlowError	Sobrecarga de la pila de ejecución

Imagen 2. Principales excepciones en java

Todas las excepciones tienen una jerarquía usando como raíz java.lang.Throwable y por debajo Exception y Error. Si se define una nueva excepción ha de heredar obligatoriamente de Throwable o cualquiera de sus hijos. Por recomendación todas las excepciones que hereden de Exception su nomenclatura será "nombreException" y las que hereden de Error será "nombreError".

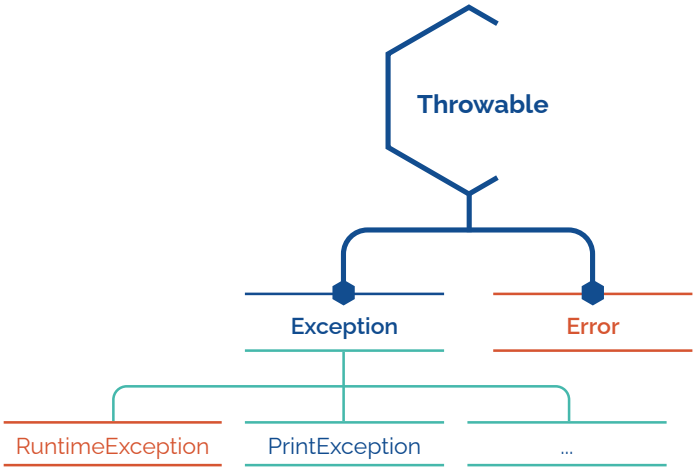

Imagen 3. Jerarquía de excepciones

Las excepciones se pueden clasificar según el tipo de grado de error:

- **Checked:** Son excepciones de consideración leve y controladas. En el caso que se produzca no debería de afectar al resto del programa. Es obligatorio hacer tratamiento de este tipo de excepción para permitir continuar con el programa. Son las excepciones que heredan de Exception. Por ejemplo, un usuario intenta hacer una división por cero en una calculadora.
- **Unchecked:** Son excepciones graves que pueden afectar a todo el programa y que interrumpa su ejecución. No es recomendable hacer un tratamiento a este tipo de excepciones, de hacerlo puede hacer que el programa no se ejecute correctamente. Son las excepciones que heredan de RuntimeExcepction o Error. Por ejemplo, un objeto no este inicializado y este null. Que no se encuentre una clase.

10.3. Miembros de una excepción

Las excepciones se representan mediante clases, y pueden tener atributos y métodos que sean necesarios. Por defecto y al heredar de java.lang.Throwable están disponible los siguientes métodos:

Métodos disponibles al heredar de Throwable	
Método	**Descripción**
Throwable getCause()	Devuelve un objeto Throwable con los datos de la causistica de la excepción.
String getMessage()	Devuelve el mensaje de la excepción.
printStackTrace()	Imprime todo el flujo de llamadas a los méto-dos hasta donde se ha producido el error.
StackTraceElement[] getStackTrace()	Obtiene una colección de StackTraceElement con cada flujo de excepcion.
setStackTrace(StackTraceElement[])	Establece una colección de StackTraceElement para indicar el flujo de la excepcion.

10.4. Manejo de excepciones

Como hemos explicado anteriormente las excepciones se pueden tratar e incluso en algunos casos es una acción obligatoria, pero antes de hacer el tratamiento hay que determinar que parte de código puede producir una excepción, si es necesario realizar alguna validación previa o incluso provocar o invocar una excepción por iniciativa nuestra. A continuación, se detalla diferentes formas de manejar excepciones.

10.4.1. Captura de excepciones

Se indicará que código va a producir una excepción, el tratamiento que se vaya a hacer si se produce y opcionalmente podemos indicar un tratamiento final se produzca o no la excepción. Estas capturas se realizarán con los bloques try/catch/finally

Sintaxis:

```
try {
        // Código que puede producir una excepción.
} catch (Tipo excepcion) {
        // Código para dar tratamiento a la excepción.
} [catch (Tipo excepcion1)] {
        // ..
} finally {
        // Código que se realizará si se produce la excepción o no.
}
```

La siguiente imagen muestra un ejemplo de captura de excepciones.

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

    float resultado = 0;
    int dividendo = 10;
    int divisor = 0;

    try {
        System.out.println("- Inicio de una división");
        resultado = dividendo / divisor;
        System.out.println("El resultado es: "+ resultado);
    } catch (ArithmeticException e1) {
        System.out.println("Se ha producido un error aritmético");
        System.out.println("Se ha intentado hacer la siguiente división: "+ dividendo +"/"+ divisor);
    } finally {
        System.out.println("- Fin de la división");
    }

}
```

```
- Inicio de una división
Se ha producido un error aritmético
Se ha intentado hacer la siguiente división: 10/0
- Fin de la división
```

Imagen 4. Bloque try/catch/finally para captura de excepciones

La parte del catch es posible definir más de un bloque para capturar diferentes excepciones del bloque try. El orden en que se especifique cada catch es importante según la jerarquía de excepciones. Si se indica como primera excepción la raíz todos los catch anteriores no se usarán nunca. Hay que especificar en orden de menor rango de la jerarquía a mayor.

Orden correcto de los Catch

```
try {
} catch (ArithmeticException e1) {
} catch (ArrayIndexOutOfBoundsException e2) {
} catch (Exception e) {
}
```

Orden incorrecto de los Catch

```
try {
} catch (Exception e) {
} catch (ArithmeticException e1) {
} catch (ArrayIndexOutOfBoundsException e2) {
}
```

Imagen 5. Uso de más de un bloque catch y su orden

Dependiendo de la versión de java es posible realizar multi-catch. Multicatch permite en un mismo bloque catch capturar más de una excepción.

```
try {

} catch (ArithmeticException | ArrayIndexOutOfBoundsException e ) {
    // Código para la excepciones Arithmetic y ArrayIndexOutOfBounds
} catch (Exception e) {
    // Código para el resto de excepciones de la clase padre de Exception
}
```

Imagen 6. Uso de más de un bloque catch y su orden

10.4.2. Propagación de excepciones

Si hay varias partes del código en diferentes métodos que contemplan que se pueda producir una misma excepción, no es necesario que se declare una sentencia try/catch en cada una de ellas. Se puede propagar la excepción para que sea el método principal que hizo la llamada para que pueda tratar la excepción.

Para propagar una excepción hay que indicar en la cabecera del método en la parte final la palabra reservada throws y el nombre de la clase de la excepción

```
public static void main(String[] args) {
    try {
        // Calculadora con las operaciones de: 0 - Suma, 1 - Resta, 2- Producto, 3- División
        calculadora(3, 0, 0);
    } catch (ArithmeticException e) {
        System.out.println("Se ha producido el siguiente error: "+e.getMessage());
        System.out.println("En: "+e.getClass());
        for (StackTraceElement traceElement : e.getStackTrace())
            System.out.println("\tat " + traceElement);
    }
}

public static int calculadora(int operacion, int param, int param2)
        throws ArithmeticException {

    switch (operacion) {
        case 0:
                return param + param2;
        case 1:
                return param - param2;
        case 2:
                return param * param2;
        case 3:
                return param / param2;

    }

    return -0;
}
```

```
Se ha producido el siguiente error: / by zero
En: class java.lang.ArithmeticException
        at EjemploPropagacion.calculadora(EjemploPropagacion.java:30)
        at EjemploPropagacion.main(EjemploPropagacion.java:8)
```

Imagen 7. Ejemplo de propagación de excepciones

10.4.3. Lanzamiento de excepciones

Otra opción que se dispone es de lanzar nosotros mismo una excepción ante un error que tenemos contemplado. Para lanzar una excepción tenemos que usar la palabra reservada throw junto con una instanciación de una excepción. Hay que ir con cuidado al confundir la propagación de excepciones con el lanzamiento. La propagación se usa la palabra throws y en el lanzamiento throw.

```
public static void main(String[] args) {
    try {
        // Calculadora con las operaciones de: 0 - Suma, 1 - Resta, 2- Producto, 3- División
        calculadora(9, 0, 0);
    } catch (Exception e) {
        System.out.println("Se ha producido el siguiente error: "+e.getMessage());
        System.out.println("En: "+e.getClass());
        for (StackTraceElement traceElement : e.getStackTrace())
            System.out.println("\tat " + traceElement);
    }
}

public static int calculadora(int operacion, int param, int param2) throws IllegalArgumentException {

    switch (operacion) {
        case 0:
                return param + param2;
        case 1:
                return param - param2;
        case 2:
                return param * param2;
        case 3:
                return param / param2;
        default:
                throw new IllegalArgumentException("Operacion no reconocida");
    }

}
```

```
Se ha producido el siguiente error: Operacion no reconocida
En: class java.lang.IllegalArgumentException
        at EjemploPropagacion.calculadora(EjemploPropagacion.java:28)
        at EjemploPropagacion.main(EjemploPropagacion.java:7)
```

Imagen 8. Ejemplo de lanzamiento de excepciones

10.4.4. Creación de clases de excepciones

A parte de las excepciones que vienen predefinidas en java, podemos crear nuevas excepciones según las necesidades que tenga nuestro programa.

```java
public class Ejemplo {

    public static void main(String[] args) {
        try {
            // Calculadora con las operaciones de: 0 - Suma, 1 - Resta, 2- Producto, 3- División
            calculadora(9, 5, 2);
        } catch (OperacionException e) {
            System.out.println("Se ha producido el siguiente error: "+e.getMessage());
            System.out.println("Código de error: "+ e.CODIGO);
            System.out.println("En: "+e.getClass());
            for (StackTraceElement traceElement : e.getStackTrace())
                System.out.println("\tat " + traceElement);
        }
    }

    public static int calculadora(int operacion, int param, int param2)  {

        switch (operacion) {
            case 0:
                    return param + param2;
            case 1:
                    return param - param2;
            case 2:
                    return param * param2;
            case 3:
                    return param / param2;
            default:
                    throw new OperacionException("Operacion no reconocida");
        }

    }

@SuppressWarnings("serial")
public class OperacionException extends IllegalArgumentException {

    public final static int CODIGO = -1;

    public OperacionException(String mensaje) {
        super(mensaje);
    }

}
```

```
Se ha producido el siguiente error: Operacion no reconocida
Código de error: 1
En: class OperacionException
        at EjemploPropagacion.calculadora(EjemploPropagacion.java:30)
        at EjemploPropagacion.main(EjemploPropagacion.java:8)
```

Imagen 9. Ejemplo de nuevas excepciones

10.5. Recomendaciones de uso de excepciones

El uso de excepciones es un buen mecanismo para tener controlada cualquier situación anómala que se pueda producir y afectar a la ejecución. Para ello es necesario tener un conocimiento previo de que situaciones anómalas se pueden producir y la mejor forma de gestionar esos errores. Si se hace un mal uso de las excepciones o aplicamos correctamente su tratamiento se producirá el efecto contrario y nuestro programa aun será más inestable y complejo de desarrollar.

Existen algunas recomendaciones de uso de excepciones y su tratamiento:

- **No abusar de excepciones.** En la mayoría de los casos se puede evitar excepciones estableciendo requisitos previos y validaciones. Es recomendable hacer un diseño de código sencillo, legible, optimo y reutilizable para evitar la mayor cantidad de excepciones.
- **Emplear correctamente los bloques try/catch.** Emplear los bloques justo en la parte de código que puede producirse el error. Un error inicial es pensar en poner todo el código de nuestro programa en un solo bloque de try/catch, de esta forma no podríamos tener el control de donde se produce el fallo e intentar recuperar la estabilidad del programa.
- **Usar siempre finally en el bloque try/catch para objetos de entrada y salida o conexiones.** Siempre que hay una conexión a una base de datos, red o nuestro programa haga uso de ficheros de entrada y salida es necesario abrir y cerrar este tipo de conexiones. Cuando se produce una excepción muchas veces no se puede cerrar la conexión establecida y se sigue consumiendo recursos. Al usar finally garantizamos que pase lo que pase se cerrara las conexiones y liberaremos recursos.
- **No crear nuevas excepciones y reutilizar las existentes.** Java aporta la mayoría de las excepciones que pueden producirse, solo en caso de que sea necesario por la funcionalidad de nuestro programa se recomiendo crear nuevas excepciones
- **Evitar tratar excepciones genéricas.** No hacer uso de captación de errores las clases padre como Throwable, Exception, Error. Es recomendable captar la excepción más detallada posible.
- **Cualquier situación anómala de consideración grave, no hacer tratamiento.** Si se produce una excepción grave que pueda dejar inestable nuestro programa se recomienda no hacer tratamiento. Por ejemplo, un error producido por falta de memoria, aún que se haga tratamiento no hay garantías que el programa siga funcionando correctamente.

www.universae.com