5.1 Excepciones

1. Introducción

Uno de los problemas más importantes al escribir aplicaciones es el tratamiento de los errores. Los errores detienen la ejecución del programa e impiden su desarrollo normal y, lo peor, además provocan que el usuario esté desinformado. Los programadores tiene que reconocer las situaciones que pueden provocar el fin de la ejecución normal del programa por un error no controlado. Dicho de otra forma, todos los posibles errores en un programa deben de estar controlados. A veces es imposible evitarlos (por ejemplo no hay papel en la impresora, o falla el disco duro), pero sí reaccionar de forma lógica para que el usuario reconozca lo que está ocurriendo.

Java nos echa una mano para ello a través de las excepciones. Se denomina **excepción** a un hecho que podría provocar la detención del programa; es decir una condición de error en tiempo de ejecución pero que puede ser controlable (a través de los mecanismos adecuados). En java sin embargo se denomina error a una condición de error incontrolable (ejemplos son el error que ocurre cuando no se dispone de más memoria o errores de
sintaxis).

Los errores de sintaxis son detectados durante la compilación, pero los errores de ejecución pueden provocar situaciones irreversibles, su control debe hacerse también en tiempo de ejecución y eso siempre ha sido problemático para la programación de aplicaciones.

En Java se puede preparar el código susceptible a provocar excepciones de modo que si ocurre una excepción, el código es lanzado (throw) a una determinada rutina previamente preparada por el programador, que permite
manipular esa excepción. Si la excepción no fuera capturada, la ejecución del programa se detendría irremediablemente (en muchas ocasiones la propia sintaxis de Java impide que la excepción no sea controlada; es decir, obliga a controlarla).

2. Try-catch

El control de las excepciones se realiza mediante las sentencias **try** y **catch**.

```
try {
    instrucciones susceptibles de provocar una excepción
}
catch (ClaseExcepción1 objetoQueCapturaLaExcepción) {
    instrucciones que se ejecutan si hay un error de tipo ClaseExcepción1
}
catch (ClaseExcepción2 objetoQueCapturaLaExcepción) {
    instrucciones que se ejecutan si hay un error de tipo ClaseExcepción2
}
...
```

Como se puede observar, puede haber más de una sentencia catch para un mismo bloque try debido a que un bloque de código puede ser susceptible a provocar varios tipos diferentes de excepciones.

Dentro del bloque try se colocan las instrucciones susceptibles de provocar una excepción, el bloque catch sirve para capturar esa excepción y evitar el fin de la ejecución del programa. Desde el bloque catch se maneja, en definitiva, la excepción.

Cada catch maneja un tipo de excepción. Cuando se produce una excepción, se busca el catch que posea el manejador de excepción adecuado, será el que utilice el mismo tipo de excepción que se ha producido. La búsqueda del catch se realiza en el orden en que se han escrito. Si se produce una excepción, primero se mira si cuadra en el primer catch. Si no cuadra, se pasa al siguiente, y así sucesivamente. Por este motivo, es importante el orden en que se coloquen los bloques catch.

Cuando acaba la ejecución del bloque catch, el programa continúa con la ejecución del código que le sigue al bloque del try-catch.

Ejemplo:

```
package tema5_1_Excepciones;

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

public class TryCatch {

    @SuppressWarnings("resource")
    public void show() {

        final String FIN = "fin";
        int base, exponent;
        String baseString;
        Scanner keyboard = new Scanner(System.in);

        try {
            System.out.println("Bienvenido al programa para calcular una
potencia.");
            System.out.print("Introduce la base o fin para terminar: ");
            baseString = keyboard.nextLine();
            if (!baseString.toLowerCase().equals(FIN)) {
                base = Integer.parseInt(baseString);
                System.out.print("Introduce el exponente: ");
                exponent = keyboard.nextInt();
```

```
                System.out.printf("%d elevado a %d es igual a %d", base,
exponent, (int) Math.pow(base, exponent));
            }
        } catch (NumberFormatException e) {
            System.err.println("Error en la base");
        } catch (InputMismatchException e) {
            System.err.println("Error en el exponente");
        }

    }

    public static void main(String[] args) {

        new TryCatch().show();

    }

}
```

Para la base, si el usuario no introduce *fin* y tampoco introduce un número entero, el `Integer.parseInt` al intentar convertir la cadena a número lanzará una excepción de tipo `NumberFormatException` y será manejada por el catch correspondiente. Para el exponente, si se introduce algo que no sea un número (una letra, un símbolo...), se producirá una excepción de tipo `InputMismatchException` y se manejará por su correspondiente catch. Si se produce un error de otro tipo, el programa se detendrá.

En este ejemplo se está utilizando *encadenamiento de llamadas a métodos*: `if(!baseString.toLowerCase().equals(FIN))` ya que el método `toLowerCase` devuelve la cadena convertida en minúsculas a la cual se le aplica después el método `equals`. En este caso nos sirve para que el usuario pueda finalizar utilizando *fin* tanto en minúsculas como en mayúsculas.

Puede ser que el programador quiera mostrar el mismo mensaje de error para ambas excepciones. En este caso se utiliza el **multi-catch**, incorporado en Java desde la versión 7:

```
package tema5_1_Excepciones;

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

public class MultiCatch {

    @SuppressWarnings("resource")
    public void show() {

        final String FIN = "fin";
        int base, exponent;
        String baseString;
        Scanner keyboard = new Scanner(System.in);

        try {
            System.out.println("Bienvenido al programa para calcular una
potencia.");
            System.out.print("Introduce la base o fin para terminar: ");
```

```
            baseString = keyboard.nextLine();
            if (!baseString.toLowerCase().equals(FIN)) {
                base = Integer.parseInt(baseString);
                System.out.print("Introduce el exponente: ");
                exponent = keyboard.nextInt();
                System.out.printf("%d elevado a %d es igual a %d", base,
exponent, (int) Math.pow(base, exponent));
            }
        } catch (NumberFormatException | InputMismatchException e) {
            System.err.println("Error en la base o en el exponente");
        }

    }

    public static void main(String[] args) {

        new MultiCatch().show();

    }

}
```

En el eclipse, si seleccionamos varias líneas de código, las podemos fácilmente insertar en un bloque try-catch: Menú *Source* → *Surround with*.

3. Manejo de excepciones

En la programación de aplicaciones en general siempre ha habido dos formas de manejar una excepción:

- Interrupción: en este caso se asume que el programa ha encontrado un error irrecuperable. La operación que dio lugar a la excepción se anula y se entiende que no hay manera de regresar al código que provocó la excepción.
- Reanudación: se puede manejar el error y regresar de nuevo al código que provocó el error.

La filosofía de Java es del tipo interrupción, pero se puede intentar emular la reanudación encerrando el bloque try-catch en un bucle que se repetirá hasta que el error deje de existir. Ejemplo:

```
package tema5_1_Excepciones;

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

public class Resumption {

    @SuppressWarnings("resource")
    public void show() {

        byte number = 0;
        Scanner keyboard = new Scanner(System.in);
        boolean error = false;

        do {
            try {
```

```
                System.out.print("Introduce un número de tipo byte, es decir,
entre -128 y 127: ");
                number = keyboard.nextByte();
                System.out.printf("Valor del número: %d\n", number);
                error = false; //Si se ha entrado antes en el catch, error está a
true
            } catch (InputMismatchException e) {
                System.err.println("Error");
                error = true;
                keyboard.nextLine(); //Limpieza del buffer
            }
        } while (error);

    }

    public static void main(String[] args) {

        new Resumption().show();

    }

}
```

Si el usuario introduce cualquier cosa que no sea un número, el método `nextByte()` lanza la excepción `InputMismatchException` y lo que ha metido el usuario se queda en el buffer. Para solucionarlo, se realiza una limpieza del buffer con `keyboard.nextLine()`.

4. Métodos de las excepciones

- getMessage: obtiene el mensaje descriptivo de la excepción.
- toString: devuelve una cadena sobre la situación de la excepción. Suele indicar la clase de excepción y el texto de getMessage.
- printStackTrace: escribe el método y mensaje de la excepción (la llamada información de pila). El resultado es el mismo mensaje que muestra el ejecutor (la máquina virtual de Java) cuando no se controla la excepción.

```
package tema5_1_Excepciones;

import java.util.Scanner;
import java.util.InputMismatchException;
import static tema1_11_EscrituraEnPantalla.colores.Colors.*;

public class MethodsOfExceptions {

    @SuppressWarnings("resource")
    public void show() {

        final String FIN = "fin";
        int base, exponent;
        String baseString;
        Scanner keyboard = new Scanner(System.in);

        try {
```

```
            System.out.println("Bienvenido al programa para calcular una
potencia.");
            System.out.print("Introduce la base o fin para terminar: ");
            baseString = keyboard.nextLine();
            if (!baseString.toLowerCase().equals(FIN)) {
                base = Integer.parseInt(baseString);
                System.out.print("Introduce el exponente: ");
                exponent = keyboard.nextInt();
                System.out.printf("%d elevado a %d es igual a %d", base,
exponent, (int) Math.pow(base, exponent));
            }
        } catch (NumberFormatException e) {
            System.err.println(GREEN + e.getMessage() + RESET);
            System.err.println(CYAN + e.toString() + RESET);
            e.printStackTrace();
        } catch (InputMismatchException e) {
            System.err.println(GREEN + e.getMessage() + RESET);
            System.err.println(CYAN + e.toString() + RESET);
            e.printStackTrace();
        }

    }

    public static void main(String[] args) {

        new MethodsOfExceptions().show();
    }

}
```

5. Cuándo se debe capturar una excepción

Una excepción se debe capturar cuando no sea un error de programación. Es decir, el programador debe distinguir si esa excepción es un error suyo de programación o no, porque si es un error de programación, no debe capturar el error sino solucionarlo. Veamos un ejemplo con la excepción *StringIndexOutOfBoundsException*.

Cuando se intenta acceder a una parte de la cadena que no existe, por ejemplo, si a la cadena "*hola*" le hacemos un *charAt(4)*, java nos lanza la excepción *StringIndexOutOfBoundsException*. El programador no debe capturar con un try-catch la excepción sino que debe corregir el error de programación que ha producido dicha excepción. Veamos un ejemplo:

```
package tema5_1_Excepciones;

public class StringIndexOutOfBoundsException1 {

    public void show() {

        String string = "hola";
        for (int i = 0; i <= string.length(); i++) {
            System.out.println(string.charAt(i));
        }

    }
```

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

        new StringIndexOutOfBoundsException1().show();

    }

}
```

La salida por consola es la siguiente:

```
h
o
l
a
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String
index out of range: 4
    at java.base/java.lang.StringLatin1.charAt(StringLatin1.java:47)
    at java.base/java.lang.String.charAt(String.java:702)
    at
tema5_1_Excepciones.StringIndexOutOfBoundsException1.show(StringIndexOutOfBoundsE
xception1.java:9)
    at
tema5_1_Excepciones.StringIndexOutOfBoundsException1.main(StringIndexOutOfBoundsE
xception1.java:16)
```

El programador no debe solucionarlo con un try-catch:

```
package tema5_1_Excepciones;

public class StringIndexOutOfBoundsException2 {

    public void show() {

        String string = "hola";
        try {
            for (int i = 0; i <= string.length(); i++) {
                System.out.println(string.charAt(i));
            }
        } catch (StringIndexOutOfBoundsException e) {//Esto no se debe hacer
            System.err.println("Esto no se debe hacer");
        }

    }

    public static void main(String[] args) {

        new StringIndexOutOfBoundsException2().show();

    }

}
```

Sino que debe corregir el error, es decir, quitar el = de la condición del for, ya que en la última iteración, *i* toma el valor de *string.length*, que es 4, cuando el *charAt(4)* no es un carácter válido ya que los caracteres válidos van de 0 a 3. Por lo tanto, el programador debe corregir el error:

```
package tema5_1_Excepciones;

public class StringIndexOutOfBoundsException3 {

    public void show() {

        String string = "hola";
        for (int i = 0; i < string.length(); i++) {
            System.out.println(string.charAt(i));
        }

    }

    public static void main(String[] args) {

        new StringIndexOutOfBoundsException3().show();

    }

}
```