Tema IX: CONTROL DE EXCEPCIONES

“No hay lenguaje de programación,
no importa su estructura,
que impida que los programadores hagan malos programas.”

CONCEPTO

Una **excepción** es un suceso que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias. Es un objeto que avisa de que ha ocurrido una condición inusual en la ejecución (objeto sin inicializar, lectura incorrecta, fuera de los límites de un array, etc). Cuando ocurre una excepción la ejecución del programa se detiene y se muestra el error.

Por ejemplo, cuando leemos un entero con el método nextInt() de la clase Scanner:

```
    System.out.println("Anota numero:");
    numero=sc.nextInt();
```

si el usuario introduce un dato que no es un entero, el programa termina al producirse una excepción:

```
<terminated> Principal1 [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (24 feb. 2021 20:34:53)
Anota numero:
f
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at excepciones1.Principal1.main(Principal1.java:16)
```

Pero podemos controlar una excepción, es decir, hacer que el programa no termine abruptamente, sino hacer que continúe de una forma sensata. A esta operación se la denomina manejar una excepción y se consigue usando las clausulas try catch.

```
try{
        instrrucciones en las que se puede producir una excepción
}
catch (ExceptionType e)
{
        manejador de la excepción e
}
catch ( ExceptionType d)
{
        manejador de la excepción d
}
finally
{
        siempre se ejecuta
}
```

El Bloque try

La sentencia o sentencias Java dentro de las cuales se puede producir una excepción, se sitúan dentro de un bloque **try**.

Los bloques catch

Un bloque catch maneja un tipo de excepción concreta. Si se produce una excepción en el bloque try el programa salta directamente al catch que se encarga de manejar dicha excepción, y no ejecuta el resto de instrucciones del try. Si no existe un bloque catch para esa excepción concreta el programa termina. Para indicar que un bloque catch maneja cualquier excepción se usa la clase Exception.

Si no se produce una excepción no se ejecuta el código en catch.

Por ejemplo, este fragmento de código lee un número entero y accede a la una posición de un array. Se pueden producir dos tipos de excepciones dentro del try:

`InputMismatchException si no se introduce un entero,`

`ArrayIndexOutOfBoundsException si nos salimos de los límites del array.`

Dependiendo de qué excepción se produzca se ejecutará el catch correspondiente:

```
System.out.println("Anota posición");

try {
    pos = sc.nextInt();
    System.out.println(numeros[pos]);
}catch (InputMismatchException e) {
    System.out.println("Tienes que introducir un número");
    sc.nextLine();
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("No existe esa posición, vuelve a anotar");
}
```

Ejemplos: EjExcepciones1, EjExcepciones2, EjExcepciones3.

El bloque finally

Si se añade la clausula finally, las instrucciones de dicho bloque siempre se ejecutan se produzca o no una excepción. Se usa fundamentalmente para cerrar los recursos abiertos pase lo que pase. (Ejemplo https://www.arquitecturajava.com/java-finally-y-el-cierre-de-recursos/)

La cláusula finally la veremos más adelante cuando trabajemos con ficheros.

TIPOS de EXCEPCIONES

Estas excepciones que hemos visto también se llaman **RuntimeException** (Excepciones en tiempo de ejecución), normalmente son excepciones que el programador puede evitar. Otras excepciones de este tipo son:

- Imposible convertir una cadena a un tipo de dato concreto: NumberFormatException, DateTimeParseException.
- No poder crear una fecha o una hora porque los datos son incorrectos (método of): DateTimeException
- Intentar acceder a un objeto que está sin instanciar: NullPointerException

Otro tipo de excepciones son las **Chequeadas** (errores que el programador no puede evitar). Por ejemplo:

- Creadas por el usuario.
- IOException.
- IllegalAccessException. (crear instancias, accede a campos o métodos sin tener acceso a la definición de la clase)

Este tipo de excepciones, si no las tratamos con un try-catch, estamos obligados a declararlas en la función, añadiendo la cláusula "throws nombre de la excepción".

Nos centramos ahora en las excepciones creadas por el usuario como un caso particular de excepciones chequeadas.

Excepciones creadas por el usuario

El lenguaje Java proporciona las clases que manejan casi cualquier tipo de excepción. Sin embargo, podemos imaginar situaciones en la que se producen excepciones que no están dentro del lenguaje Java.

El programador puede crear sus propias excepciones. Para ello tenemos que:

- Crear una clase que herede de Exception (de ésta heredan todas las excepciones en Java). En ella sólo hay que crear el método constructor que recibe un String con un mensaje que describe la causa de la excepción.

- Crear un método que cree un objeto excepción de la clase anterior y la lance a través de la sentencia **throw**. A este método además hay que añadirle en la declaración la palabra reservada **throws** seguido de la excepción o excepciones que puede lanzar.

Ej. EjExcepcionPropia

Excepciones chequeadas frente a las runTime

Si ocurre una RuntimeException en un método y no queremos tratarla el método acaba y la excepción pasa al método llamante y así sucesivamente hasta llegar a la MVJ. El programa termina y se nos muestra en pantalla el tipo de excepción ocurrida.

Si ocurre una excepción chequeada en un método, y no queremos tratarla, estamos obligados a pasarla al método llamante usando la cláusula "throws nombre de la excepción".

Por ejemplo, el método readLine() de la clase BufferedReader sirve para leer una línea de texto, y puede lanzar la excepción IOException, que es una excepción chequeada.

Si desde un método llamamos a la función readLine() y no queremos tratar la IOException tenemos que añadir en la declaración de dicho método throws IOException, si no es así no compila el programa.

Ejemplo: función que lee un double y si se produce error en el input pasamos el error al método llamante:

```
double leeDouble() throws IOException{
        BufferedReader br=new BufferedReader(new  InputStreamReader(System.ini));
        System.out.print("Introduce un double");
        // Cómo llamamos al método readLine() que puede lanzar la IOException y no
        // queremos tratarla con un catch, nos vemos obligados a introducir la clausula
        // throws IOException en la declaración de la función.
        double d=Double.valueOf(br.readLine().trim()).doubleValue();
        return d;
}
```

Con esto estamos diciendo el método leeDouble o retorna un double o retorna una IOException. Pero además de la IOException este método puede devolver otras excepciones, como la que se produce cuando se teclea un enter o un carácter no válido en un double, la NumberFormatException. Estas otras excepciones, como son del tipo runTimeException no son necesarias declararlas.

Ejemplo: Programa que lee un número double hasta que sea correcto.

```
// El main captura la excepción NumberFormatException dentro de un bucle.
// Se usa la clausula finally

import java.io.*;
class Leer{
public static void main(String[] arg) throws IOException {

        BufferedReader in=new BufferedReader(new InputStreamReader (System.in));
        boolean incorrecto;
        int i=1;
        do
        {
                incorrecto=false;
                try{
                        System.out.print("Introduce un double");
                        double d=Double.valueOf(in.readLine().trim()).doubleValue();
                        System.out.print("Número leido:"+d);
                        }
                catch (NumberFormatException e){
                        System.out.print("\nNúmero leido incorrecto:");
                        incorrecto=true;
                }
                finally{
```

```
                System.out.println("Número de veces que he leido:"+(i++));
            }
            System.out.println("------------");

        }while (incorrecto);
}
}
```

La Sentencia throw

Cómo hemos visto se pueden lanzar explícitamente las excepciones utilizando la sentencia throw seguida de un objeto Exception, dicho objeto es la excepción que se lanza.

También se pueden relanzar excepciones. Se trata de capturar una excepción pero además pasársela al método llamante, para ello en el bloque catch añadimos throw e:

```
try{
        ...
}
catch( ExceptionType e){
        ...
        throw e;
```

Una función puede lanzar varias excepciones, en ese caso se declaran las posibles excepciones en la función separadas por comas.