TEMA 9: EXCEPCIONES

1. INTRODUCCIÓN

A diferencia de otros lenguajes de programación orientados a objetos como C/C++, ***Java*** incorpora en el propio lenguaje la gestión de errores. El mejor momento para detectar los errores es durante la compilación. Sin embargo prácticamente sólo los errores de sintaxis son detectados durante este periodo. El resto de problemas surgen durante la ejecución de los programas.

Una *excepción* es un cierto tipo de error o una condición anormal que se ha producido durante la ejecución de un programa y que interrumpe el flujo de ejecución normal de las sentencias.

Algunas *excepciones* son fatales y provocan que se deba finalizar la ejecución del programa, como pueden ser serios problemas de hardware, por ejemplo la avería de un disco duro. En este caso conviene terminar ordenadamente y dar un mensaje explicando el tipo de error que se ha producido.

Otras, como por ejemplo no encontrar un fichero en el que hay que leer o escribir algo, o tratar de acceder a un elemento en un array fuera de sus limites, pueden ser recuperables. En este caso el programa debe dar al usuario la oportunidad de corregir el error (por ejemplo, indicando una nueva localización del fichero no encontrado).

Un buen programa debe gestionar correctamente todas o la mayor parte de los errores que se pueden producir. Hay dos "estilos" de hacer esto:

1. ***A la "antigua usanza"***: los métodos devuelven un código de error. Este código se chequea en el entorno que ha llamado al método, gestionando de forma cada uno de los posibles errores. Este sistema resulta muy complicado cuando hay varios niveles de llamadas a los métodos.
2. ***Con soporte en el propio lenguaje***: En este caso el propio lenguaje proporciona construcciones especiales para gestionar los errores. Suele ser lo habitual en lenguajes modernos, como C++, Visual Basic o ***Java***.

Para comprender mejor la utilidad de las excepciones, vamos a pararnos a pensar lo que habitualmente se hace en programación: se incluyen tantas instrucciones condicionales como sea necesario para conseguir que una aplicación sea robusta; de esta manera, por ejemplo, en cada división de un valor entre una variable, antes se comprueba que el denominador no sea cero.

Si utilizamos el mecanismo de excepciones, en nuestro ejemplo, en lugar de incluir una serie de instrucciones condicionales para evitar las distintas divisiones por cero que se puedan dar, se escribe el programa sin tener en cuenta esta circunstancia y, posteriormente, se escribe el código que habría que ejecutar si la situación "excepcional" se produce.

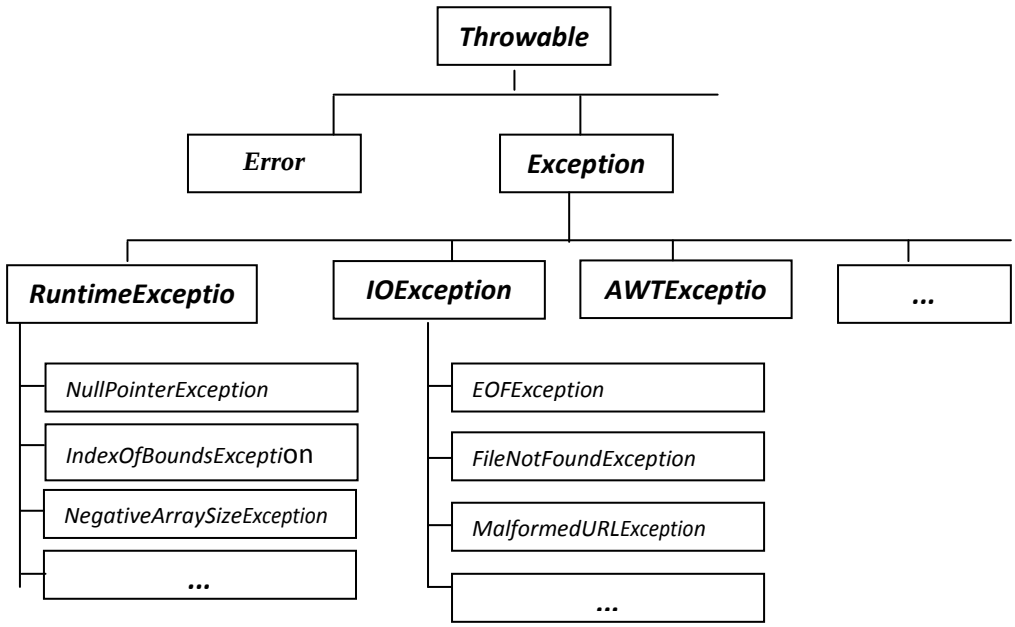
Al hacer uso de excepciones, el bloque que codifica la porción de aplicación resulta más sencillo de entender, puesto que no es necesario incluir las instrucciones condicionales que verifican si puede darse la situación de excepción. En definitiva, si utilizamos el mecanismo de excepciones, podemos separar la lógica del programa de las instrucciones de control de errores, haciendo la aplicación más legible y robusta.

Las excepciones son objetos (clases) que se crean cuando se produce una situación extraordinaria en la ejecución del programa. Estos objetos almacenan información acerca del tipo de situación anormal que se ha producido y el lugar donde ha ocurrido. Los objetos excepción se pasan automáticamente al bloque de tratamiento de excepciones.

En los siguientes apartados se examina cómo se trabaja con los bloques y expresiones ***try***, ***catch***, ***throw***, ***throws*** y ***finally***, cuándo se deben lanzar excepciones, cuándo se deben capturar y cómo se crean las clases propias de tipo ***Exception***.

2. EXCEPCIONES ESTÁNDAR EN JAVA.

Los errores se representan mediante dos tipos de clases derivadas de la clase ***Throwable***: ***Error*** y ***Exception***. La siguiente figura muestra parcialmente la jerarquía de clases relacionada con ***Throwable.***



La clase ***Error*** está relacionada con errores de compilación, del sistema o de la JVM. De ordinario estos errores son ***irrecuperables*** y no dependen del programador ni debe preocuparse de capturarlos y tratarlos.

La clase ***Exception*** tiene más interés. Dentro de ella se puede distinguir:

1. ***RuntimeException***: Son excepciones muy frecuentes, de ordinario relacionadas con errores de programación. Se pueden llamar ***excepciones implícitas***.
2. Las demás clases derivadas de ***Exception*** son ***excepciones explícitas***. ***Java*** obliga a tenerlas en cuenta y chequear si se producen.

El caso de ***RuntimeException*** es un poco especial. El propio ***Java*** durante la ejecución de un programa chequea y lanza automáticamente las excepciones que derivan de ***RuntimeException***. El programador no necesita establecer los bloques ***try***/***catch*** para controlar este tipo de excepciones, es decir, no es necesario realizar un tratamiento explícito de estas excepciones (de todas las demás clases derivadas de *Exception*, sí es necesario). Entre las clases derivadas de *RuntimeException* se encuentran:

- *ArithmeticException:* cuando ocurre una operación aritmética errónea, por ejemplo, división entre 0; con los valores reales no se produce esta excepción.
- *ArrayStoreException:* Intento de almacenar un valor de tipo erróneo en una matriz de objetos.
- *IllegalArgumentException:* Se le ha pasado un argumento ilegal o inapropiado a un método.

- *IndexOutBoundsException:* Cuando algún índice (por ejemplo, de array o String) está fuera de rango.
- *NegativeArraySzeException:* Cuando se intenta crear un array con un índice negativo.
- *NullPointerException:* Cuando se utiliza como apuntador una variable con valor *null*.

En realidad sería posible comprobar estos tipos de errores, pero el código se complicaría excesivamente si se necesitara chequear continuamente todo tipo de errores (que las ***referencias*** son distintas de ***null***, que todos los argumentos de los métodos son correctos, y un largo etcétera).

Las clases derivadas de ***Exception*** pueden pertenecer a distintos packages de ***Java***. Algunas perenecen a ***java.lang*** (*Throwable*, *Exception*, *RuntimeException*, …); otras a ***java.io*** (*EOFException*, *FileNotFoundException*, ...) o a otros packages. Por heredar de ***Throwable*** todos los tipos de excepciones pueden usar los métodos siguientes:

1. String ***getMessage()*** Extrae el mensaje asociado con la excepción.
2. String ***toString()*** Devuelve un String que describe la excepción.
3. void ***printStackTrace()*** Indica el método donde se lanzó la excepción.

3. LANZAR UNA EXCEPCIÓN.

Cuando en un método se produce una situación anómala es necesario ***lanzar una excepción***. El proceso de lanzamiento de una excepción es el siguiente:

1. Se crea un objeto ***Exception*** de la clase adecuada.
2. Se lanza la excepción con la sentencia ***throw*** seguida del objeto ***Exception*** creado.

```
// Código que lanza la excepción MyException una vez detectado el error
MiExcepcion me = new MiExcepcion("mensaje");
throw me;
```

Esta excepción deberá ser capturada (***catch***) y gestionada en el propio método o en algún otro lugar del programa (en otro método anterior en la ***pila*** o ***stack*** de llamadas), según se explica en el apartado siguiente.

Al lanzar una excepción el método termina de inmediato, sin devolver ningún valor. Solamente en el caso de que el método incluya los bloques ***try/catch/finally*** se ejecutará el bloque ***catch*** que la captura o el bloque ***finally*** (si existe).

Todo método en el que se puede producir uno o más tipos de excepciones (y que no utiliza directamente los bloques ***try/catch/finally*** para tratarlos) debe ***declararlas*** en el encabezamiento de la función por medio de la palabra ***throws***. Si un método puede lanzar varias excepciones, se ponen detrás de ***throws*** separadas por comas, como por ejemplo:

```
public void leerFichero(String fich) throws EOFException, FileNotFoundException {…}
```

Se puede poner únicamente una ***superclase de excepciones*** para indicar que se pueden lanzar excepciones de cualquiera de sus clases derivadas. El caso anterior sería equivalente a:

```
public void leerFichero(String fich) throws IOException {…}
```

Las excepciones pueden ser lanzadas directamente por ***leerFichero( )*** o por alguno de los métodos llamados por ***leerFichero ()***, ya que las clases ***EOFException*** y ***FileNotFoundException*** derivan de ***IOException***.

Se recuerda que no hace falta avisar de que se pueden lanzar objetos de la clases ***Error*** o ***RuntimeException*** (excepciones implícitas).

4. CAPTURAR UNA EXCEPCIÓN.

Como ya se ha visto, ciertos métodos de los packages de ***Java*** y algunos métodos creados por cualquier programador producen ("lanzan") excepciones. Si el usuario llama a estos métodos sin tenerlo en cuenta se produce un error de compilación con un mensaje del tipo: "… *Exception java.io.IOException must be caugth or it must be declared in the throws clause of this method*". El programa no compilará mientras el usuario no haga una de estas dos cosas:

1. ***Gestionar la excepción*** con una construcción del tipo ***try {…} catch {…}***.
2. ***Re-lanzar la excepción*** hacia un método anterior en el ***stack***, declarando que su método también lanza dicha excepción, utilizando para ello la construcción ***throws*** en el header del método.

El compilador obliga a capturar las llamadas ***excepciones explícitas***, pero no protesta si se captura y luego no se hace nada con ella. En general, es conveniente por lo menos imprimir un mensaje indicando qué tipo de excepción se ha producido.

4.1. Bloques try y catch.

En el caso de las excepciones que no pertenecen a las ***RuntimeException*** y que por lo tanto ***Java*** obliga a tenerlas en cuenta habrá que utilizar los bloques ***try***, ***catch*** y ***finally***. El código dentro del bloque ***try*** está "vigilado": Si se produce una situación anormal y se lanza por lo tanto una excepción el control salta o sale del bloque ***try*** y pasa al bloque ***catch***, que se hace cargo de la situación y decide lo que hay que hacer. Se pueden incluir tantos bloques ***catch*** como sean necesarios, cada uno de los cuales tratará un tipo de excepción.

Las excepciones se pueden capturar individualmente o en grupo, por medio de una superclase de la que deriven todas ellas.

El bloque ***finally*** es opcional. Si se incluye sus sentencias se ejecutan siempre, sea cual sea la excepción que se produzca o si no se produce ninguna. El bloque ***finally*** se ejecuta aunque en el bloque ***try*** haya un ***return***.

En el siguiente ejemplo se presenta un método que debe "controlar" una ***IOException*** relacionada con la lectura ficheros y una ***MyException*** propia:

```
void metodo1(){
     ...
     try {
          // Código que puede lanzar las excepciones IOException y MyException
     } catch (IOException e1) { // Se ocupa de IOException simplemente dando aviso
          System.out.println(e1.getMessage());
     } catch (MyException e2) {
          // Se ocupa de MyException dando un aviso y finalizando la función
          System.out.println(e2.getMessage()); return;
     } finally { // Sentencias que se ejecutarán en cualquier caso
         ...
     }
} // Fin del metodo1
```

4.2. Relanzar una Excepción.

Existen algunos casos en los cuales el código de un método puede generar una ***Exception*** y no se desea incluir en dicho método la gestión del error. ***Java*** permite que este método pase o relance (***throws***) la ***Exception*** al método desde el que ha sido llamado, sin incluir en el método los bloques ***try/catch*** correspondientes. Esto se consigue mediante la adición de ***throws*** más el nombre de la ***Exception*** concreta después de la lista de argumentos del método. A su vez el método superior deberá incluir los bloques ***try/catch*** o volver a pasar la ***Exception***. De esta forma se puede ir pasando la ***Exception*** de un método a otro hasta llegar al último método del programa, el método ***main( )***.

El ejemplo anterior (***metodo1***) realizaba la gestión de las excepciones dentro del propio método. Ahora se presenta un nuevo ejemplo (***metodo2***) que relanza las excepciones al siguiente método:

```
void metodo2() throws IOException, MyException {
     ...
     // Código que puede lanzar las excepciones IOException y MyException
     ...
} // Fin del metodo2
```

Según lo anterior, si un método llama a otros métodos que pueden lanzar excepciones (por ejemplo de un package de ***Java***), tiene 2 posibilidades:

1. ***Capturar*** las posibles excepciones y gestionarlas.
2. Desentenderse de las excepciones y ***remitirlas*** hacia otro método anterior en el ***stack*** para éste se encargue de gestionarlas.

Si no hace ninguna de las dos cosas anteriores el compilador da un error, salvo que se trate de una ***RuntimeException***.

4.3. Método finally.

El bloque ***finally {...}*** debe ir detrás de todos los bloques ***catch*** considerados. Si se incluye (ya que es opcional) sus sentencias se ejecutan siempre, sea cual sea el tipo de excepción que se produzca, o ***incluso si no se produce ninguna***. El bloque ***finally*** se ejecuta incluso si dentro de los bloques ***try/catch*** hay una sentencia ***continue***, ***break*** o ***return***. La forma general de una sección donde se controlan las excepciones es por lo tanto:

```
try {
     // Código "vigilado" que puede lanzar una excepción de tipo A, B o C
} catch (A a1) {
     // Se ocupa de la excepción A
} catch (B b1) {
     // Se ocupa de la excepción B
} catch (C c1) {
     // Se ocupa de la excepción C
} finally {
     // Sentencias que se ejecutarán en cualquier caso
}
```

El bloque ***finally*** es necesario en los casos en que se necesite recuperar o devolver a su situación original algunos elementos. No se trata de liberar la memoria reservada con ***new*** ya que de ello se ocupará automáticamente el *garbage collector*.

Como ejemplo se podría pensar en un bloque ***try*** dentro del cual se abre un fichero para lectura y escritura de datos y se desea cerrar el fichero abierto. El fichero abierto se debe cerrar tanto si produce una excepción como si no se produce, ya que dejar un fichero abierto puede provocar problemas posteriores. Para conseguir esto se deberá incluir las sentencias correspondientes a cerrar el fichero dentro del bloque ***finally***.

5. CREAR NUEVAS EXCEPCIONES.

El programador puede crear sus propias excepciones sólo con heredar de la clase ***Exception*** o de una de sus clases derivadas. Lo lógico es heredar de la clase de la jerarquía de ***Java*** que mejor se adapte al tipo de excepción. Las clases ***Exception*** suelen tener dos constructores:

1. Un ***constructor*** sin argumentos.
2. Un ***constructor*** que recibe un ***String*** como argumento. En este ***String*** se suele definir un mensaje que explica el tipo de excepción generada. Conviene que este constructor llame al constructor de la clase de la que deriva ***super(String)***.

Al ser clases como cualquier otra se podrían incluir variables y métodos nuevos. Por ejemplo:

```
class MiExcepcion extends Exception {
     public MiExcepcion() {       // Constructor por defecto
          super();
     }
     public MiExcepción(String s) {   // Constructor con mensaje
          super(s);
     }
}
```

6. HERENCIA DE CLASES Y TRATAMIENTO DE EXCEPCIONES.

Si un método redefine otro método de una super-clase que utiliza ***throws***, el método de la clase derivada no tiene obligatoriamente que poder lanzar todas las mismas excepciones de la clase base. Es posible en el método de la subclase lanzar ***las mismas excepciones o menos***, pero no se pueden lanzar más excepciones. No puede tampoco lanzar nuevas excepciones ni excepciones de una clase más general.

Se trata de una restricción muy útil ya que como consecuencia de ello el código que funciona con la clase base podrá trabajar automáticamente con referencias de clases derivadas, incluyendo el tratamiento de excepciones, concepto fundamental en la *Programación Orientada a Objetos* (***polimorfismo***).

7. EJEMPLO DE MANEJO DE EXCEPCIONES.

Veamos un ejemplo en el que se muestra como indicar si un método puede producir excepciones:

```
public class BDConexion {
  public void consulta(int arg1, String arg2) tHrows IOException, SQLException
   {
        // Este método puede producir dos excepciones
        //  · IOException (error de entrada y salida)
        //  · SQLException (error al efectuar una operación en la BD)
   }
   ...
}
```

Un método donde se ejecute el método consulta puede tratar la excepción (capturarla):

```
public class UsoBD {
  public void listado()
  {
        BDConexion bd = new BDConexion();
        try {
              bd.consulta(2,"Nombre");
              bd.mostrarDatos();
        }
        catch (IOException e) {
              // Tratamiento de la excepción IOException
```

```
                // “e” contiene el objeto IOException con la información
                // de la excepción que se ha producido
                System.out.println(e.toString());
        }
        catch (SQLException e) {
                // Tratamiento de la excepción SQLException
                System.out.println(e.getMessage();
                e.printStackTrace();
        }
        finally {
                // Código que se ejecuta se produzca o no error
                bd.cerrarConexion();
        }
  }
  ...
}
```

Pero en el método anterior podemos tomar la decisión de no tratar el error y dejarlo que el método que llame al método *listado* sea el que trate la excepción:

```
public class UsoBD {
  public void listado() throws IOException, SQLException
  {
        // Este método puede producir dos excepciones
        //  · IOException (error de entrada y salida)
        //  · SQLException (error al efectuar una operación en la BD)
        BDConexion bd = new BDConexion();
        bd.consulta(2,”Nombre”);
        bd.mostrarDatos();
   }
  ...
}
```

En el caso anterior no tratamos la excepción por lo tanto indicamos en la cláusula ***throws*** la excepciones que se pueden producir.

Pero y si somos nosotros mismos lo que queremos lanzar la excepción, es decir, mediante código. Para ello tendremos que crear un objeto de la clase a la que pertenece la excepción que queremos lanzar. Una vez creado el objeto sólo tenemos que lanzar la excepción para ello utilizamos la cláusula ***throw***. Nótose que este *throw* no lleva *s*.

Veamos un ejemplo:

```
public class UsoBD {
  public void listado() throws IOException, SQLException
  {
        // Este método puede producir dos excepciones
        //  · IOException (error de entrada y salida)
        //  · SQLException (error al efectuar una operación en la BD)
        BDConexion bd = new BDConexion();
        if (bd == null) throw new SQLException(“Conexión fallida”);
        bd.consulta(2,”Nombre”);
        bd.mostrarDatos();
   }
  ...
}
```

También hemos visto que para crear nuestras propias excepciones sólo tenemos que crear una clase que herede de la clase Exception. Veamos un ejemplo:

```
public class MiException extends Exception {
    public MiException()  {}
    public MiException(Strig msg)  {super(msg}; }
}
```

En este ejemplo creamos una clase para crear nuestras propias excepciones. La excepción que podremos crear es ***MiException***. El código anterior nos puede servir como base para crear nuestras propias excepciones.

Una posible clase cuyos métodos pueden producir una excepción como la creada sería:

```
public class PruebaExcepcion {
   public void metodo1() throws MiException {
        System.out.println("Metodo 1: Lanzando excepción MiExcepcion");
        throw new MiExcepcion();
   }
}
```

Como se puede observar el tratamiento de una excepción creada por nosotros mismo es exacta al tratamiento de cualquier otra excepción.

El código de una clase que utiliza un objeto de clase PruebaExcepcion debe hacer el tratamiento normal:

```
public class Prueba {
   public static void main(Srings args[]) {
        PruebaExcepcion p = null;
        try {
              // Capturamos la excepción
              p = new PruebaExcepcion();
              p.metodo1();
              // método que puede lanzar una excepción
        }
        catch(MiExcepcion e) {
              // Tratamiento de la excepción.
              e.printStackTrace();
              // Imprimimos la pila de llamadas
        }
    }
 }
```

Para capturar cualquier tipo de excepción podemos poner un catch de este tipo

```
        catch(Exception e) {
              // Tratamiento necesario
        }
```

Puesto que todas las excepciones heredan de la clase ***Exception***, todos los objetos excepciones son objetos de la clase Exception, por lo que conseguimos tratar cualquier tipo de excepción en un solo catch.