Excepciones en Java

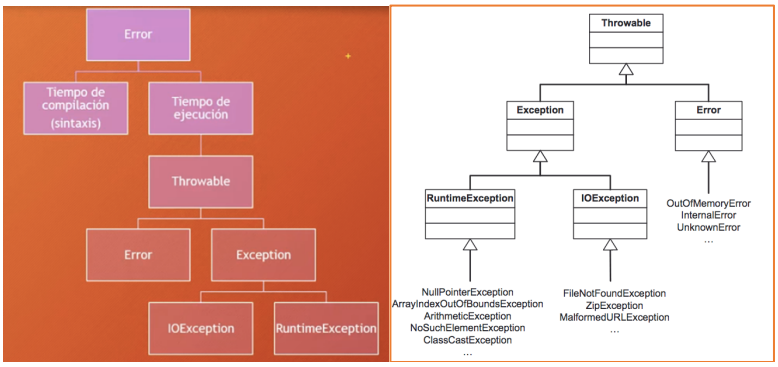
Qué son

Las excepciones son un mecanismo que permite a los métodos indicar que algo "anómalo" ha sucedido y que impide su correcto funcionamiento
Decimos en este caso, que el método ha lanzado(throw) una excepción.
Si en el código ninguna instrucción "atrapa" (catch) la excepción, el programa acaba su ejecución y se informa al usuario del error.
Cuando se produce un error en un método, "se lanza" un objeto Throwable.

¿Por qué debemos capturar las excepciones?
El manejo de errores y excepciones permite no ejecutar la parte de código que ha dado error y el programa podrá seguir ejecutando el resto.

Las excepciones no sirven para "corregir" errores de programación.
Su utilidad es informar sobre los errores o situaciones excepcionales que se producen durante la ejecución de un programa. Por lo tanto, deben ser entendidas como un sistema para informar sobre errores que han ocurrido en una aplicación, y no como una solución a los errores en la programación.

Tipos de excepciones



Error: Subclase de Throwable que indica problemas graves ajenos a la aplicación.
Exception: Exception y sus subclases indican situaciones que una aplicación sí debe gestionar. Los dos tipos principales de excepciones son:

- **RuntimeException**: Excepciones en tiempo de ejecución. Errores del programador, como una división por cero o el acceso fuera de los límites de un array. A este tipo de excepciones se les llama **excepciones no comprobadas**.

```java
public class Main {
  public static void main(String[] args) {
    int[] numeros = {1, 2, 3};
    System.out.println(numeros[10]);
  }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 10 out of bounds for length 3
        at Main.main(Main.java:4)
```

  Este tipo no es obligatorio que el programador las trate explícitamente pero si el código no las trata y se producen, el programa finaliza con mensaje de error.

- **IOException** (errores que no puede evitar el programador, generalmente relacionados con la entrada/salida del programa).
  A este tipo de excepciones se les llama **excepciones comprobadas**.
  No se considera que sean culpa del programador de Java. Por ejemplo, si tenemos que coger un fichero de una carpeta del ordenador, pero al ejecutar, coincide que alguien ha movido o borrado ese archivo de la carpeta, el programa falla y se dice que ese error es ajeno al programador.

Captura de excepciones: Bloques try...catch

Para capturar las excepciones que se hayan podido producir se utiliza el bloque de código delimitado por try y catch. Por ejemplo, el código que manipule ficheros, tendrá la siguiente estructura:

```
try {
  Codigo que abre y trata el fichero
} catch (IOExcepction ex) {
  Código que trata el error
}
```

Intenta(try) ejecutar esas instrucciones y, en caso de producirse un error en el tratamiento de los ficheros (se lanza una IOException), atrapa(catch) ese error y ejecuta el código de corrección. La cláusula catch recibe como argumento un objeto Throwable

El try catch "es como un if else". Si va todo bien haces la parte del try, y si va algo mal haces la parte del catch (captura una excepción)

La técnica básica consiste en colocar las instrucciones que podrían provocar problemas dentro de un bloque try, y a continuación uno o más bloques catch, de tal forma que si se provoca un error de un determinado tipo, lo que haremos será saltar al bloque catch capaz de gestionar ese tipo de error específico.
El bloque catch contendrá el código necesario para gestionar ese tipo específico de error. Si no se provocan errores en el bloque try, nunca se ejecutarán los bloques catch.

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] numeros = {1, 2, 3};
      int calculo =numeros[1]/0;
      //System.out.println(numeros[10]);
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Posición fuera del array");
    }
    catch (Exception e) {
      System.out.println("Algo fue mal");
    }
  }
}
```

```
Algo fue mal
```

```
public class Main {
  public static void main(String[] args) {
    try {
      int[] numeros = {1, 2, 3};
      //int calculo =numeros[1]/0;
      System.out.println(numeros[10]);
    }
    catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("Posición fuera del array");
    }
    catch (Exception e) {
      System.out.println("Algo fue mal");
    }
  }
}
```

```
Posición fuera del array
```

Si sale una excepción no comprobada (Runtime Exception), significa que podríamos haberlo programado mejor, podríamos haberlo evitado. No se considera buena práctica usar el try – catch, tienes que mejorar el código para evitar que de el error.

La **cláusula finally**: En ocasiones, nos interesa ejecutar un fragmento de código independientemente de si se produce o no una excepción (por ejemplo, cerrar un fichero que estemos manipulando)

```
// Bloque1
try {
  // Bloque 2
} catch (ArithmeticException ae) {
  // Bloque 3
} finally {
  // Bloque 4
}
// Bloque 5
```

Sin excepciones:	1 → 2 → 4 → 5
Excepción de tipo aritmético:	1 → 2* → 3 → 4 → 5
Excepción de otro tipo diferente:	1 → 2* → 4

Gestión de excepciones

Se puede lanzar una excepción manualmente

Ojo!: No confundir throw con throws

- La **sentencia throw** se utiliza en Java para lanzar objetos de tipo Throwable
- Si en el cuerpo de un método se lanza una excepción (de un tipo derivado de la clase Exception), en la cabecera del método hay que añadir una **cláusula throws** que incluya una lista de los tipos de excepciones que se pueden producir al invocar el método, por ejemplo: public String leerFichero (String nombreFichero) throws IOException

Ejemplo método que lanza una excepción, podrá, o bien devolver un valor double, o bien lanzar un objeto de la clase "Exception".

```
public static double division_real (double dividendo, double divisor) throws Exception{
        double aux;
        if (divisor != 0){
                aux = dividendo/divisor;
        }
        else {
                throw new Exception();
        }
        return aux;
}
```

El objeto que podemos lanzar por medio de "throw" no ha sido capturado dentro del método, es una **excepción propagada por el método:** le pasa el "marrón" al que le ha invocado. Todos los métodos que usen este método deberán tenerlo en cuenta a la hora de ser programados, gestionando la excepción que les podría llegar.

Supongamos que hemos invocado ese método desde el main. Si "lanzamos" la excepción, se lanza al método "main. El flujo de nuestra aplicación vuelve al main y no se llega a ejecutar la orden "return aux;" Y ahora tocaría realizar cambios en el main para que sea capaz de "capturar" la excepción que se ha lanzado.

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

        double x = 15.0;
        double y = 3.0;
        try{
                System.out.println ("El resultado de la division real de " + x +
                                    " entre " + y + " es " + division_real (x, y));
        }
        catch (Exception mi_excepcion){
                System.out.println ("Has intentado dividir por 0.0;");
                System.out.println ("El objeto excepcion lanzado: " +
                                    mi_excepcion.toString());
        }
}
```

Cuando dentro de un método aparece una excepción puede ser por 2 motivos:

- el propio método la ha creado y lanzado
- la excepción viene lanzada de algún otro método invocado desde ese método

En resumen, cuando un método se encuentra con una excepción que no sea de tipo "RunTimeException", tiene 2 opciones: propagar la excepción o puede capturarla y gestionarla. Si es de tipo "RunTimeException", el hecho de gestionarla es opcional, depende del programador.

El propio método captura y gestiona las excepciones

Otra opción para gestionar las excepciones es que el propio método capture la excepción en vez de propagarla. “modo juan palomo”, no es muy recomendado, salvo para debuguear.

```
public static double acceso_por_indice (double [] v, int indice){
      try {
            if ((0<=indice) && (indice <v.length)){
                return v [indice];
            }
            else {
                //Caso excepcional
                throw new Exception ("El indice " + indice +
                                        " no es una posicion valida");
            }
          }
      catch (Exception mi_excepcion){

             System.out.println(mi_excepcion.toString());
             System.out.println(mi_excepcion.getMessage());
      }
      finally {
            return 0.0;
      }
}
```

Como el método devuelve un “double”, en finally ponemos “return 0.0;” como valor por defecto del método.

Los clientes de este método ahora no deberán preocuparse de que el método lance una excepción, ya que el propio método las captura y gestiona.

El hecho de capturar la excepción de forma prematura ha hecho que se pierda parte de la utilidad de la misma, que es “avisar” a los clientes de nuestro método de que una situación excepcional se ha producido en el mismo. Por eso lo conveniente es poner el bloque try-catch fuera del método.

El método captura, gestiona y propaga las excepciones

Una tercera opción es que el propio método capture la excepción y además la propague.

```
public static double acceso_por_indice (double [] v, int indice) throws Exception{
        try {
                if ((0<=indice) && (indice <v.length)){
                        return v [indice];
                }
                else {
                        //Caso excepcional:
                        throw new Exception ("El indice " + indice
                                                + " no es una posicion valida");
                }
              }

                catch (Exception mi_excepcion){
                        System.out.println(mi_excepcion.toString());
                        System.out.println(mi_excepcion.getMessage());
                        throw mi_excepcion;
                }
}
```

Sobre qué opción es mejor que otra, aunque no hay una receta universal, en general se suele seguir la estrategia de generar excepciones y lanzarlas para que los métodos que los invoquen las gestionen como consideren oportuno.

Creación de nuestras propias excepciones

Hay que declarar una clase que herede de la clase Exception en Java (o de otro tipo, RuntimeException o IOException). Así creamos una clase de objetos que pueden ser lanzados con el comando “throw” y que podrán ser utilizados en situaciones anómalas de cualquier tipo.
Si hereda de RuntimeException el IDE no nos obliga a crear el try-catch, pero recordad que no se considera buena práctica.

```
Class comprueba_mail extends Exception {
        public comprueba_mail () {
                super ();
        }
        public comprueba_mail (String msj_error) {
                super (msj_error);
        }
}
static void comprueba_mail (String email) throws comprueba_mail {
        int contadorarroba=0;
        for (int=0; i<email.length();i++){
                If (email.charAt(i) == ‘@’ ) contadorarroba++;
        …
        If (contadorarroba !=1) throw new comprueba_mail(“Num. de @ en el email incorrecto”);

}
```

Al definir la clase de nuestra propia excepción podíamos haber redefinido alguno de los métodos de la misma, o haber añadido métodos nuevos. Por ejemplo, haber redefinido el método “toString ()” para que mostrase un mensaje distinto al que muestre en su definición por defecto.

Ejemplo gestión ficheros y gestión excepciones

```
import java.io.*;
import java.util.*;
public class Ejemplo_Ficheros{
public static void main (String [] args){
try{
        File file = new File("entrada_salida.txt");
        //Lanza NullPointerException, si la cadena es vacía
        boolean success = file.createNewFile();//Crea el fichero si no existe
        //Lanza IOException o SecurityException
        if (success) {
                // El fichero no existe y se crea:
                System.out.println("El fichero no existe y se crea");
                //Comprueba que el fichero se puede escribir y leer:
                System.out.println ("El fichero se puede escribir "  + file.canWrite());
                System.out.println ("El fichero se puede leer "  + file.canRead());
                //Le asociamos al fichero un búfer de escritura:
                BufferedWriter file_escribir =  new BufferedWriter (new FileWriter (file));
                //Lanza IOException
                //Escribimos cadenas de caracteres en el fichero, separadas por saltos de líneas:
                file_escribir.write("Una primera sentencia:"); //Lanza IOException
                file_escribir.newLine();
                file_escribir.write("8.5");//Lanza IOException
                file_escribir.newLine();
                file_escribir.write("6");//Lanza IOException
                file_escribir.newLine();
                file_escribir.flush(); //Lanza IOException
                file_escribir.close(); //Lanza IOException
                //Abrimos ahora el fichero para lectura por medio de la clase Scanner:
                Scanner file_lectura =  new Scanner (file);
                //Lanza FileNotFoundException
                //Leemos cadenas de caracteres del mismo:
                //En las 3 siguientes se lanza IllegalStateException o NoSuchElementException
                String leer = file_lectura.nextLine();
                String leer2 = file_lectura.nextLine();
                String leer3 = file_lectura.nextLine();
                //Intentamos convertir cada cadena a su tipo de dato original:
                double leer_double;
                int leer_int;
                leer_double = Double.parseDouble(leer2); //Lanza NumberFormatException
                leer_int = Integer.parseInt (leer3);
                //Mostramos por la consola las diversas cadenas:
                System.out.println ("La cadena leida es " + leer);
                System.out.println ("El double leido " + leer_double);
                System.out.println ("El entero leido " + leer_int);
                }
        else // El fichero ya existe:
                System.out.println("El fichero ya existe y no se creo");
        }
catch (FileNotFoundException f_exception) { //Ocurre si no encuentra el fichero al crear el Scanner
        System.out.println ("Las operaciones de lectura no  se han podido llevar a cabo,");
        System.out.println ("ya que ha sucedido un problema  al buscar el fichero para lectura");
        System.out.println (f_exception.toString());
        }
catch (IOException io_exception){ //Ocurre en las operaciones de escritura
        System.out.println("Ocurrio algun error de entrad y salida");
        System.out.println(io_exception.toString());
        }
catch (NumberFormatException nb_exception){ //Conversion de cadena de caracteres a tipo numerico
        System.out.println ("Ha ocurrido un error de  conversión de cadenas a numeros");
        System.out.println (nb_exception.toString());
        }
catch (NoSuchElementException nse_exception){ //Cuando el metodo nextLine() no encuentra una cadena
        System.out.println ("No se ha podido encontrar el  siguiente elemento del Scanner");
        System.out.println (nse_exception.toString());
        }
catch (Exception e_exception){ //Para capturar otro tipo de excepción
        System.out.println(e_exception.toString());
        }
}
```

Otro método que se suele utilizar para sacar información de la excepción:e.printStackTrace