



Unidad 6

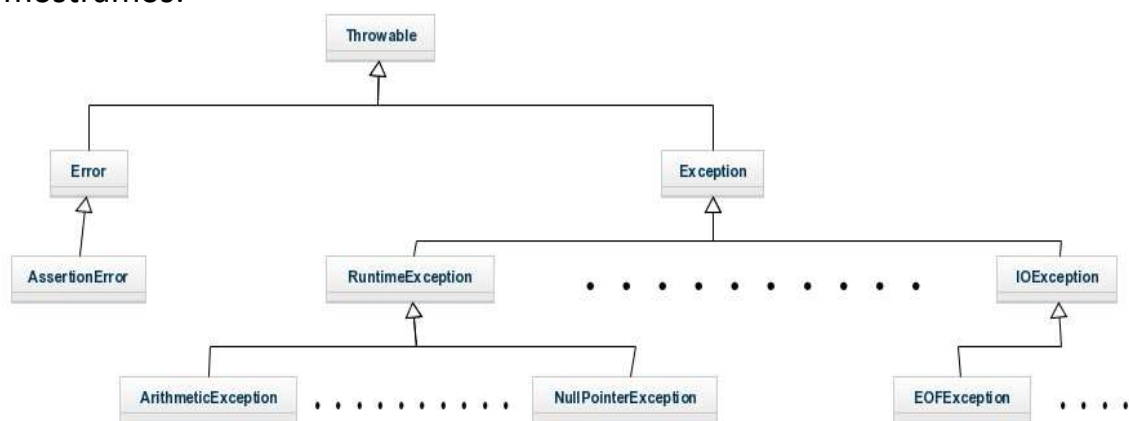
Excepciones. Bloque Catch , Try .

Excepciones :

En **Java** los errores en tiempo de ejecución (cuando se esta ejecutando el programa) se denominan **excepciones**, y esto ocurre cuando se produce un error en alguna de las instrucciones de nuestro programa, como por ejemplo cuando se hace una división entre cero, cuando un objeto es 'null' y no puede serlo... y hay otros

También se producen en el uso de la base de Datos, pero que no corresponden a este módulo.

Cuando en Java se produce una excepción se crear un objeto de una determina clase (dependiendo del tipo de error que se haya producido), que mantendrá la información sobre el error producido y nos proporcionará los métodos necesarios para obtener dicha información. Estas clases tienen como clase padre la clase **Throwable**, por tanto se mantiene una jerarquía en las excepciones. A continuación mostramos a modo de ejemplo algunas de las clases para que nos hagamos una idea de la jerarquía que siguen las excepciones, pero existen muchísimas más excepciones que las que mostramos:



Acá vemos algunas subclases derivadas de la clase Padre, donde hay excepciones de tipos Matemáticas, de entrada y salida de datos, de tiempo de ejecución, etc.



Uso de los comandos Catch y Try

¿Para qué se utiliza el Catch y el Try? El try catch en programación se utiliza para manejar fragmentos de código que son propensos a fallar, como puede ser: recibir un valor nulo, convertir un tipo de dato a otro o en tu caso, la lectura de un archivo, su uso esta ligadamente relacionado con **las excepciones**. También tenemos el Try Finally.

Al leer un archivo que esta almacenado en la memoria del dispositivo (interna o externa), se genera un flujo de bits (físicamente corriente) desde dicha memoria a la memoria ram. Ese flujo puede verse interrumpido por diferentes razones, lo que causaría que la tarea falle y que el programa se detenga. **El try catch lo que hace es asegurarse que aunque la tarea que se este ejecutando falle, el programa se siga ejecutando y no se detenga.**

Si el código que está dentro del try falla, se ejecuta el catch y el programa se sigue ejecutando. **Dentro del try se debe de colocar el código que es propenso a fallar y dentro del catch el código a ejecutarse si el try falla, como puede ser un mensaje de error.**

El try catch se debe utilizar cuando se tenga un fragmento de código que es propenso a fallar, que se sabe que por una u otra razón puede fallar. Como por ejemplo la conversión de un tipo de dato a otro (Casting):

```
try {  
  
    // Convierte un valor de tipo String a int  
    int numero = Integer.parseInt("a456");  
  
} catch(Exception e) {  
  
    System.out.println("ERROR: el valor de tipo String contiene caracteres no  
numéricos");  
  
}
```



En este caso el código que esta dentro del try fallaría y se ejecutaría el código que esta dentro del catch. Si no se usará el try catch en este caso, al producirse el error el programa se detendría. Pero al utilizar el try catch, aunque se produzca el error el programa se seguirá ejecutando normalmente.

¿En el siguiente código que función ejerce la clase try y catch? *Cópialo en el netbeans y ejecútalo para ver su funcionamiento.*

```
package Package;
import java.io.*;

public class Textos {

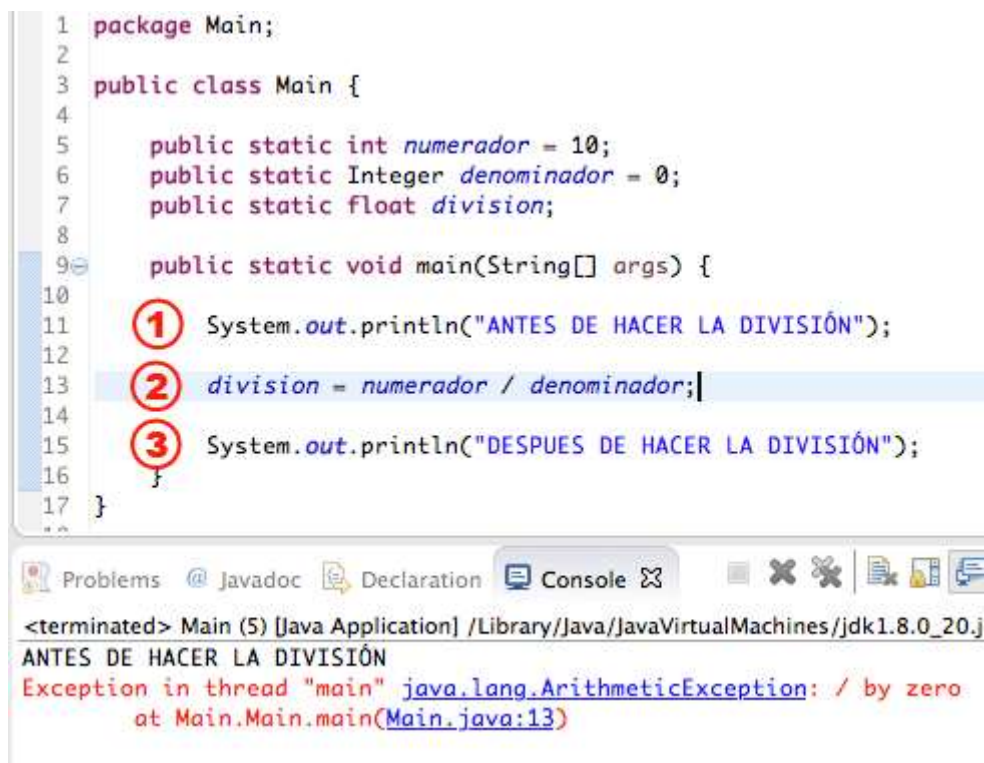
    public void leer (String nombreArchive) {
        try {

            FileReader r= new FileReader (nombreArchive);
            BufferedReader buffer = new BufferedReader (r);
            //System.out.println(buffer.readLine());

            String temp="";
            while(temp!=null) {
                temp = buffer.readLine();
                if (temp==null)
                    break;
                System.out.println(temp);
            }
        }
        catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Uso de los comandos Catch , Try y Finally

A continuación vamos a mostrar un ejemplo de como al hacer una división entre cero, se produce una excepción. Veamos la siguiente imagen en el que podemos ver un fragmento de código y el resultado de la ejecución del código:



```
1 package Main;
2
3 public class Main {
4
5     public static int numerador = 10;
6     public static Integer denominador = 0;
7     public static float division;
8
9     public static void main(String[] args) {
10
11         ❶ System.out.println("ANTES DE HACER LA DIVISIÓN");
12
13         ❷ division = numerador / denominador;
14
15         ❸ System.out.println("DESPUES DE HACER LA DIVISIÓN");
16     }
17 }
```

Problems Javadoc Declaration Console

<terminated> Main (5) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.j
ANTES DE HACER LA DIVISIÓN
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Main.Main.main(Main.java:13)

Como vemos en nuestro programa tenemos 3 instrucciones. La primera debe de imprimir por pantalla el mensaje "ANTES DE HACER LA DIVISIÓN", la segunda debe de hacer la división y la última debe de imprimir por pantalla el mensaje "DESPUES DE HACER LA DIVISIÓN". La primera instrucción la ejecuta perfectamente, pero al llegar a la segunda se produce una "**ArithmeticException**" (excepción de la clase ArithmeticException) y se detiene la ejecución del programa ya que estamos dividiendo un número entre '0'.



Por suerte Java nos permite hacer un control de las excepciones para que nuestro programa no se pare inesperadamente y aunque se produzca una excepción, nuestro programa siga su ejecución. Para ello tenemos la estructura "try – catch – finally" que la mostramos a continuación:

```
try {  
  
    // Instrucciones cuando no hay una excepción  
  
} catch (TypeException ex) {  
  
    // Instrucciones cuando se produce una excepcion  
  
} finally {  
  
    // Instrucciones que se ejecutan, tanto si hay como sino hay excepcion  
es
```

Respecto a la estructura "try – catch – finally", decimos que primero se ejecuta el bloque "try", si se produce una excepción se ejecuta el bloque "catch" y por último el bloque "finally". En esta estructura se puede omitir el bloque "catch" o el bloque "finally", **pero no ambos**.

Sabiendo esta estructura, podemos reescribir nuestro programa para que se ejecuten las tres instrucciones aunque se produzca una excepción. Previamente debemos de saber cual va a ser la clase de la excepción que puede aparecer que seria la "**ArithmeticException**" para definirla en la parte del "catch". Nuestro programa quedaría de la siguiente



forma y se ejecutaría sin problema obteniendo también la información de la excepción:

```
1 package Main;
2
3 public class Main {
4
5     public static int numerador = 10;
6     public static Integer denominador = 0;
7     public static float division;
8
9     public static void main(String[] args) {
10         System.out.println("ANTES DE HACER LA DIVISIÓN");
11         try {
12             division = numerador / denominador;
13         } catch (ArithmeticException ex) {
14             division = 0; // Si hay una excepción doy valor '0' al atributo 'division'
15             System.out.println("Error: "+ex.getMessage());
16         } finally {
17             System.out.println("División: "+division);
18             System.out.println("DESPUES DE HACER LA DIVISIÓN");
19         }
20     }
21 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (5) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Home/bin/java (!

ANTES DE HACER LA DIVISIÓN
Error: / by zero
División: 0.0
DESPUES DE HACER LA DIVISIÓN

Como vemos capturamos la excepción en un objeto "ex" de la clase "ArithmeticException" y podemos obtener el mensaje de error que nos da la excepción. Vemos también que el programa termina su ejecución aunque se haya producido una excepción.

Dentro de una misma estructura podemos definir todas las excepciones que queramos. En el caso anterior hemos definido solo la excepción "ArithmeticException"; pero por ejemplo, podemos definir también la excepción "NullPointerException", por si nos viene un valor a 'null' al hacer la división:



```
1 package Main;
2
3 public class Main {
4
5     public static int numerador = 10;
6     public static Integer denominador = null;
7     public static float division;
8
9     public static void main(String[] args) {
10         System.out.println("ANTES DE HACER LA DIVISIÓN");
11         try {
12             division = numerador / denominador;
13         } catch (ArithmeticException ex) {
14             division = 0; // Si hay una excepción doy valor '0' al atributo 'division'
15             System.out.println("Error: "+ex.getMessage());
16         } catch (NullPointerException ex) {
17             division = 1; // Si la excepción es de un null doy valor '1' al atributo 'division'
18             System.out.println("Error: "+ex.getMessage());
19         } finally {
20             System.out.println("División: "+division);
21             System.out.println("DESPUES DE HACER LA DIVISIÓN");
22         }
23     }
24 }
```

Problems @ Javadoc Declaration Console

<terminated> Main (5) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_20.jdk/Contents/Home/bin/java (12/10/2014 16:00)

ANTES DE HACER LA DIVISIÓN
Error: null
División: 1.0
DESPUES DE HACER LA DIVISIÓN

En resumen, hemos puesto en esta entrada un ejemplo muy sencillo para controlar un par de excepciones bastante obvias como la división entre '0' y un 'null', que perfectamente lo podríamos haber controlado con una sentencia de control "if" mirando el contenido de los atributos, pero la finalidad de esta entrada era ver como controlar las excepciones con la estructura **"try - catch - finally"**, que si lo sabemos utilizar nuestro programa deberá seguir funcionando aunque se produzcan excepciones. Decir también que es casi imposible aprenderse todas las excepciones que hay en Java, también las podemos encontrar en trabajando con ficheros y arrays.

Links de videos explicativos.

<https://www.youtube.com/watch?v=jYnHBbqHE3A>

<https://www.youtube.com/watch?v=BlivRaKyOIU>