UD14. EXCEPCIONES

1. Introducción.
2. Instrucción try..catch.
3. Tipos de excepciones.
4. Excepciones creadas por el programador.

1. Introducción

Las excepciones son el medio que ofrecen algunos lenguajes de programación para tratar situaciones anómalas que pueden suceder cuando ejecutamos un programa.

- llamar a un método sobre un objeto "null"
- intentar dividir por 0
- intentar abrir un fichero que no existe para leerlo
- quedarnos sin memoria en la JVM al ejecutar (bucle sin fin)
- intentar crear un fichero en una carpeta en la que no tenemos permiso de escritura
- etc.

En los programas que hemos realizado hasta ahora, se han producido en ocasiones algunas excepciones como NullPointerException o IndexOutOfBoundsException.

2. Instrucción try..catch

```
try {
        //Bloque de Instrucciones del try
} catch (TipoExcepción nombreVariable) {
        //Bloque de Instrucciones del primer catch
} catch (TipoExcepción nombreVariable) {
        //Bloque de Instrucciones del segundo catch
} .....
}[finally {
        //Bloque de Instrucciones de finally
}]
```

2. Instrucción try..catch

```
5   public class DivisionPorCero1 {
6       public static void main(String[] args) {
7       //Entorno:
8           int x, y, z;
9       //Algoritmo:
10          x = 15;
11          y = 0;
12
13          z = x / y;
14          System.out.println(x + " / " + y + " = " + z);
15          System.out.println("Terminar proceso");
16          System.out.println("Fin Programa");
17      }//Fin Programa
```

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivisionPorCero1.main(DivisionPorCero1.java:13)
Java Result: 1
```

2. Instrucción try..catch

```
6   public class DivisionPorCero2 {
7       public static void main(String[] args) {
8       //Entorno:
9           int x, y, z;
10      //Algoritmo:
11          x = 15;
12          y = 0;
13          try {
14              z = x / y;
15              System.out.println(x + " / " + y + " = " + z);
16              System.out.println("Terminar proceso");
17          } catch (ArithmeticException ae) {
18              System.out.println(" ");
19          }//Fin try-catch
20          System.out.println("Fin Programa");
21      }//Fin Programa
22  }
```

```
run:

Fin Programa
```

2. Instrucción try..catch

```
5   public class DivisionPorCero3 {
6       public static void main(String[] args) {
7        //Entorno:
8           int x, y, z;
9        //Algoritmo:
10          x = 15;
11          y = 0;
12          try {
13              z = x / y;
14              System.out.println(x + " / " + y + " = " + z);
15          } catch (NullPointerException ae) {
16              System.out.println("No se puede realizar la división");
17          }finally{
18              System.out.println("Terminar proceso");
19          }//Fin try-catch
20          System.out.println("Fin Programa");
21      }//Fin Programa
22  }
```

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
Terminar proceso
        at DivisionPorCero3.main(DivisionPorCero3.java:13)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Instrucción try..catch

```
5   public class DivisionPorCero4 {
6       public static int divide(int dividendo, int divisor) throws ArithmeticException {
7           return (dividendo / divisor);
8       }
9
10      public static void main(String[] args) {
11       //Entorno:
12          int x, y;
13       //Algoritmo:
14          x = 15;
15          y = 0;
16          System.out.println(x + " / " + y + " = " + divide(x, y));
17          System.out.println("Terminar proceso");
18          System.out.println("Fin Programa");
19      }//Fin Programa
20  }
```

No es necesario, se propaga siempre

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivisionPorCero4.divide(DivisionPorCero4.java:7)
        at DivisionPorCero4.main(DivisionPorCero4.java:16)
Java Result: 1
```

2. Instrucción try..catch

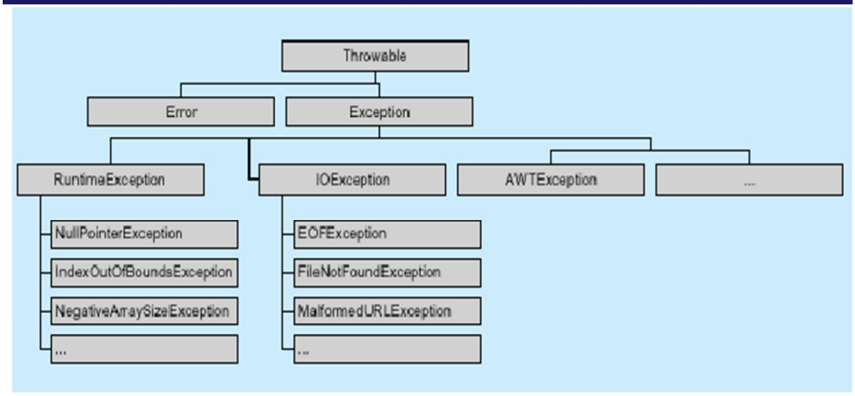
```
5   public class DivisionPorCero5 {
6       public static int divide(int dividendo, int divisor) throws ArithmeticException {
7       //Entorno:
8           int z;
9       //Algoritmo:
10          z=dividendo / divisor;
11          System.out.println("Aquí controlo yo");
12          return z;
13      }
14      public static void main(String[] args) {
15      //Entorno:
16          int x, y;
17      //Algoritmo:
18          x = 15;
19          y = 0;
20          try {
21              System.out.println(x + " / " + y + " = " + divide(x, y));
22          } catch (ArithmeticException ae) {
23              System.out.println("No se puede realizar la división");
24          }finally{
25              System.out.println("Terminar proceso");
26          }//Fin try-catch
27          System.out.println("Fin Programa");
28      }//Fin Programa
29  }
```

```
run:
No se puede realizar la división
Terminar proceso
Fin Programa
```

2. Instrucción try..catch

1) Hacer un programa que recorra la tabla de cadenas conteniendo *"12", "20", "8", "18"*. Y muestre por pantalla, la mitad de cada valor.
2) Ejecutar el programa anterior con los valores *"12", "m", "8", "18"*
3) Hacer un programa que recorra la tabla de cadenas conteniendo *"12", "m", "8", "18"*. Y muestre por pantalla, la mitad de cada valor. Si se encontrara algún valor del que no pueda calcularse la mitad, deberá mostrar "No es un número" y dejar de mostrar.
4) Hacer un programa que recorra la tabla de cadenas conteniendo *"12", "m", "8", "18"*. Y muestre por pantalla, la mitad de cada valor. Si se encontrara algún valor del que no pueda calcularse la mitad, deberá mostrar "No es un número" y continuar mostrando hasta el final.

3. Tipos de excepciones (jerarquía)



La clase Error ➔ errores de JVM no se gestionan.
La clase Exception ➔
RuntimeException ➔ errores de programación, el compilador no obliga a gestionarlos.
El resto de Excepciones ➔ el compilador de Java sí obliga a corregirlas.

3. Tipos de excepciones

1) La máquina Virtual de Java puede generar una excepción como producto de un error interno que está fuera de su control. Estas excepciones generalmente no pueden ser manejadas por el programa.
2) Excepciones estándar: Son excepciones que deben ser manipuladas, se producen cuando se ejecuta una división por cero o se trata de acceder a un array con un índice fuera de límites son generadas por errores en el código del programa.
3) El programador puede generar una excepción manualmente utilizando la estructura throw. Sin importar cómo se produjo la excepción, se maneja de la misma forma.

4. Excepciones creadas por el programador

```
class MiExcepcion extends Exception {

    public MiExcepcion() {
        super();
    }

    public MiExcepcion(String s) {
        super(s);
    }

    public String toString() {
        String msg=this.getMessage();
        if (msg==null){
            msg="Sin mensaje";
        }//Fin Si
        return "Se ha producido la excepción "
                + this.getClass().getName() + "\n"
                + "Con el siguiente mensaje: " + msg + "\n";
    }
}
```

```
throw new MiExcepcion("Denominador negativo");
throw new MiExcepcion();
```

4. Excepciones creadas por el programador

```
public class CrearExcepcion {
    public static int divide(int a, int b) throws MiExcepcion,ArithmeticException {
    //Entorno:
        int res;
    //Algoritmo:
        if (b < 1) {
            //throw new MiExcepcion("Denominador negativo");
            throw new MiExcepcion();
        } else {
            res = a / b;
        }//Fin Si
        return res;
    }
    public static void main(String[] args) {
    //Entorno:
        int x, y;
    //Algoritmo:
        x = 3;
        y = -1;
        try {
            System.out.println(divide(x, y));
        } catch (ArithmeticException ae){
            System.out.println(ae.getMessage());
        } catch (MiExcepcion m) {
            System.out.println(m.toString());
        }//Fin try-catch
    }//Fin Programa
}
```