Java

Tema 2

Control de Excepciones y pruebas

Control de Excepciones

El control de excepciones permite al programador controlar la ejecución del programa, para evitar que acabe de forma inesperada.

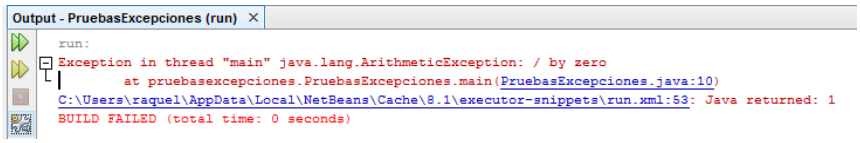
Control de Excepciones: estructura básica

```
try{

  //Código con riesgo de dar error

}catch(Exception   mi_variable){
  //Código que solo se ejecuta al
  //producirse el error
}
```

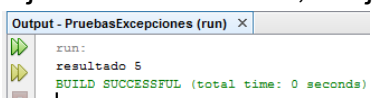
Clase de java: **Exception**

Control de Excepciones: ejemplo

```
int b=0,c;

 c = 10/b;  //error división por 0
System.out.println("resultado " + c);
```



Si ejecutamos con b=2, la ejecución es correcta



Control de Excepciones: ejemplo

```
int b=0,c;

try{

   c = 10/b;  //error división por 0
  System.out.println(“resultado " + c);

}catch(Exception  e){
   System.out.println("Error division por 0");
}
```

Ejercicio: Prueba a ejecutar b=0 y b=2

Control de varias Excepciones: ejemplo

Ejecutar el siguiente código tecleando para la variable b los siguientes valores: 3, 0 , a

```
int  b, c;
Scanner sc = new Scanner(System.in);

 System.out.println ("introduce un numero");
 b = sc.nextInt ();
 c  =  15 / b;

System.out.println("resultado " + c);
```

Control de varias Excepciones: muticatch

Excepción **"InputMismatchException"** generada al teclear: **un carácter a**

Debemos importar la clase:
import
java.util.InputMismatchException

```
Output - Pruebas (run)
run:
Introduce un numero:
a
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:909)
        at java.util.Scanner.next(Scanner.java:1530)
        at java.util.Scanner.nextInt(Scanner.java:2160)
        at java.util.Scanner.nextInt(Scanner.java:2119)
        at pruebas.Pruebas.main(Pruebas.java:24)
Java Result: 1
BUILD SUCCESSFUL (total time: 3 seconds)
```

Excepción **"ArithmeticException"** generada al teclear: **0**

```
Output - Pruebas (run)
run:
Introduce un numero:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at pruebas.Pruebas.main(Pruebas.java:26)
Java Result: 1
BUILD SUCCESSFUL (total time: 3 seconds)
```

7

Control de varias Excepciones: multicatch

```
int b, c;
Scanner sc = new Scanner(System.in);

try{
     System.out.println ("introduce un numero");
     b = sc.nextInt();
     c  =  15 / b;
     System.out.println("resultado " + c);
 } catch (ArithmeticException e){
     System.out.println("Excepcion aritmetica");
 } catch (InputMismatchException e){
     System.out.println("Entrada de datos erronea");
 } catch (Exception e){
     System.out.println("Otro tipo de excepcion");
 } finally {
    System.out.println(" Siempre se ejecuta finally");
 }
```

8

Control de Excepciones: muticatch

```
try{
     //Código con riesgo de dar error
}catch( subclaseException e1 ){
     //Control del error 1
}catch(subclaseException e2 ){
     //Control del error 2
}
……..
}catch(Exception e ){
     //Control de cualquier otro error
}[finally{
     //siempre se ejecuta
}]
```

finally es opcional

9

Control de Excepciones: multi-catch

❑ Se pueden controlar varias excepciones en un mismo try-catch

❑ Java evalúa los catch en orden desde el primero, tras la primera excepción que encaja ya no sigue evaluando. Por tanto el catch(Exception e) debe estar siempre el último

❑ Vaya bien o mal el código vigilado, el apartado finally siempre se ejecutará (si está presente)

10

Varios try en un mismo código

❑ En un mismo código pueden a existir varios try-catch

❑ Cuando salta un try-catch la ejecución sigue con normalidad.

Varios try en un mismo código

```
try{
     System.out.println (“¿Cuantos melocotones hay?");
     cuantos = sc.nextInt();
 } catch (InputMismatchException e){
     System.out.println(“has introducido una letra");
     cuantos = 0;
 }
 try{
     System.out.println (“¿Cuantas personas hay?");
     personas = sc2.nextInt();
     reparto  =  cuantos / personas;
     System.out.println(“reparto" + reparto);
 } catch (ArithmeticException  e){
     System.out.println(“Has introducido 0 personas");
 } catch (InputMismatchException  e){
     System.out.println(“Has introducido una letra");
 }
```

Try asociado a un bucle

```
int b, c=0;
boolean error = false;
Scanner sc = new Scanner(System.in);

do {
    try {
        System.out.println("Introduce un numero");
        c = sc.nextInt();
    } catch (InputMismatchException e) {
        System.out.println("Error en la recogida de datos");
        error=true;
        sc=new Scanner(System.in);
    }
 }while (error == true);

 b = 15 / c;
 System.out.println("resultado" + b);
```

Reiniciamos la consola para limpiar el carácter erróneo leído

13

Ejercicios Excepciones

Modifica los ejercicios de condicionales 1 y 2 del boletín B e incluye el control de las excepciones :

-introducción de carácter en lugar de un numero (`InputMismatchException`)

-y cualquier otra excepción (`Exception`)

14

Java

Pruebas
Debugger

Pruebas y depuración

Pruebas de software: Conjunto de procesos que permiten verificar y validar la calidad de un producto software. En los que podemos distinguir los siguientes fallos:

- Bugs o errores de programación: fallos en la semántica del código.
- Defectos de forma: El programa no hace lo que se espera de él

Debugger - depurador

Debugger: Herramienta integrada en el NetBeans que nos ayuda a ejecutar paso a paso el código para detectar el error

Video tutorial

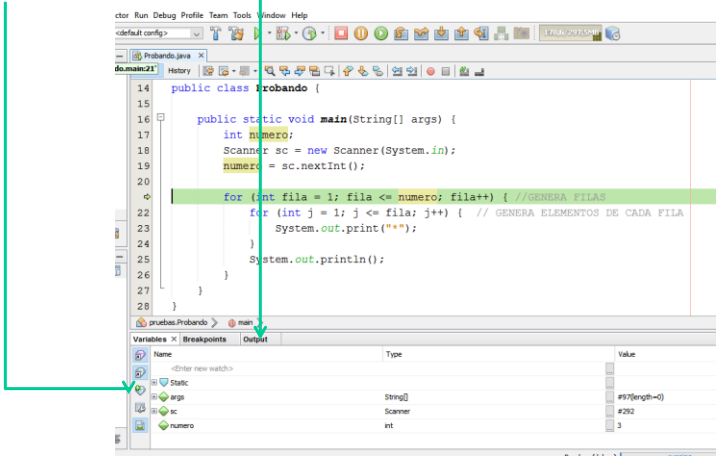
https://www.youtube.com/watch?v=w_-H4SCAUVw

17

Debugger - depurador

Si ejecutamos paso a paso (F7) podemos ir viendo la ejecucion poco a poco en la ventana output:

En la ventana Variables podemos ir controlando las variables que se van creando.



18

Uso del Debugger

Resumen de opciones más importantes:

Step Into → F7: ejecuta línea a línea
Step Out → Ctrl+F7: sale del método que se esta ejecutando
Run to cursor → F4: ejecuta hasta donde hemos dejado el cursor
Breakpoint → ctrl+F8: marca la línea donde queremos que se detenga la ejecución
- condicionales
- de conteo

Watches: visualizar valor de las variables elegidas

Uso del Debugger

EJERCICIOS: ejecuta mediante el debugger los ejercicios del boletin B:

-ejercicio 1: . ejecución paso a paso
. Añade parada cuando se calcula el bono de alimentación
. añade un breakpoint condicional cuando el nº de hijos sea 4 o superior

-ejercicio 4: . ejecución paso a paso
. añade un breakpoint de conteo y parar en la iteración 4 del primer bucle