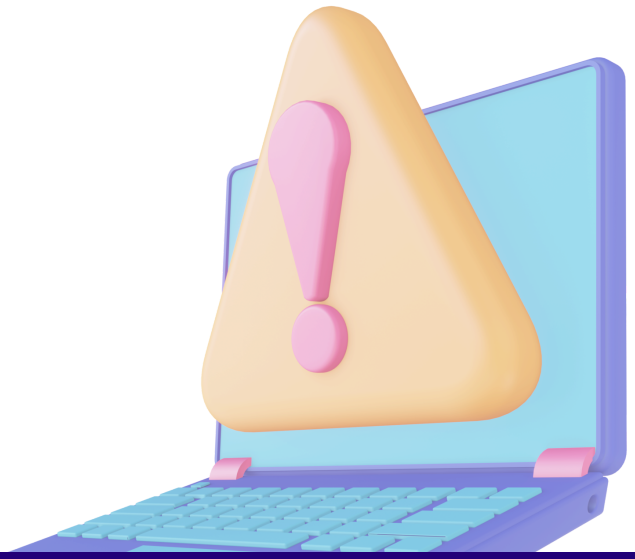
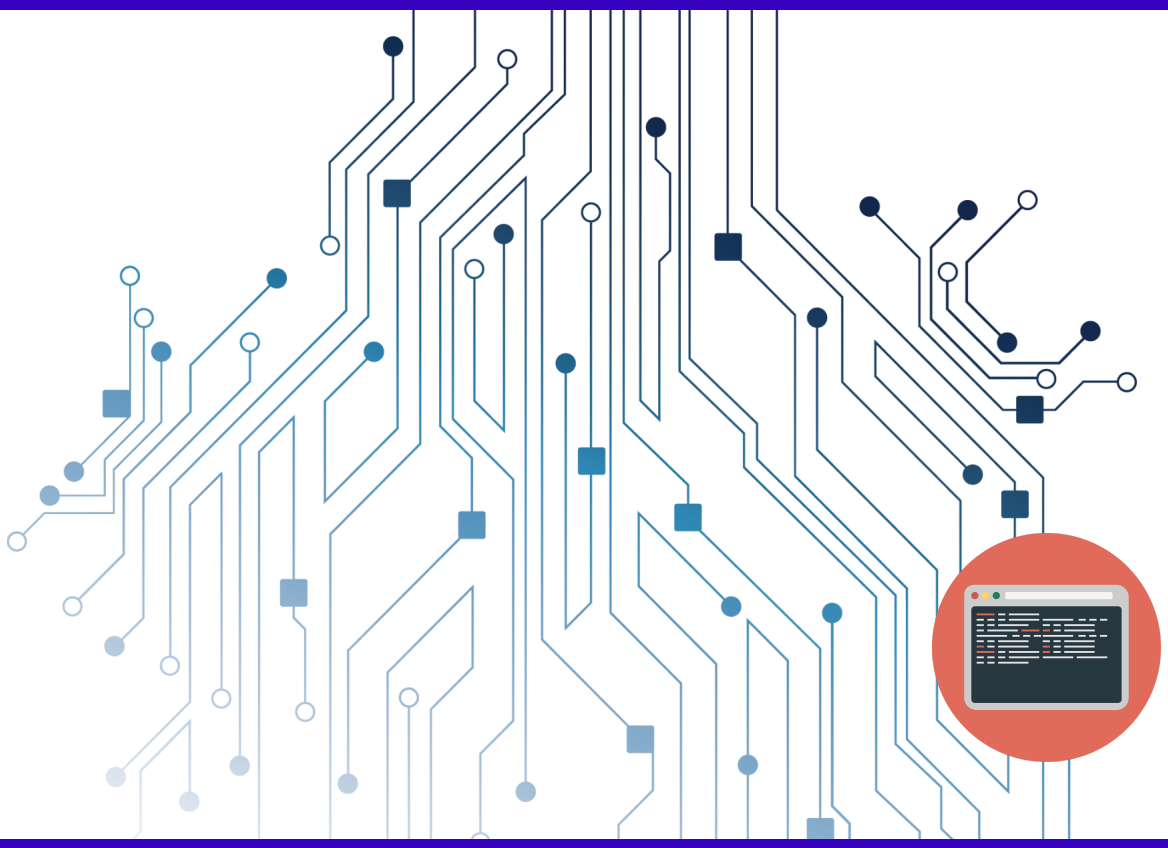


CFGS APLICACIONES MULTIPLATAFORMA - PROGRAMACIÓN

UNIDAD 6

EXCEPCIONES



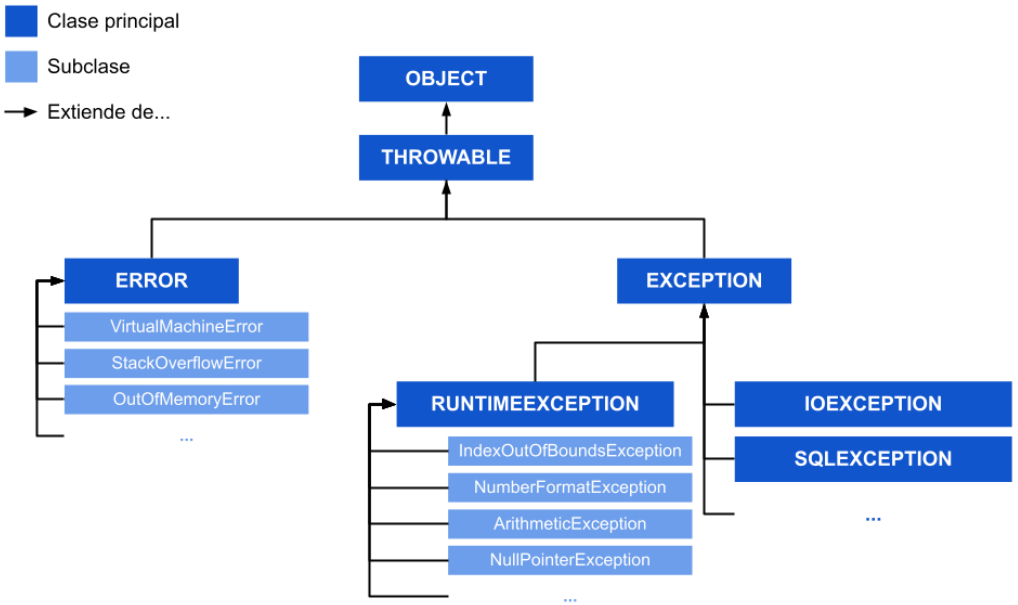
DIEGO VALERO ARÉVALO
BASADO EN APUNTES Y EJERCICIOS PROPORCIONADOS POR
Mª CARMEN DÍAZ GONZÁLEZ - IES VIRGEN DE LA PALOMA

U6 - **EXCEPCIONES**
ÍNDICE

6.1 - EXCEPCIONES

Una **excepción** es un **error semántico que se produce durante el tiempo de ejecución de un programa**.

Durante el curso y seguramente en tus propios programas has recibido algún tipo de error, tipo `OutOfBoundsException`, `NumberFormatException`, `InputMismatchException`, ... Todos estos son objetos de la clase **Exception**, que se generan cuando se detectan datos erróneos con los que el programa no puede trabajar. Otros tipos de errores internos provienen de la clase **Error**, los cuales vienen a su vez de la clase **Throwable**. Aquí tienes un diagrama con varios ejemplos de los dos:



Cuando se genera una excepción de cualquier tipo, decimos que esta se "lanza" (`throw`). Si no tratamos esos objetos, el programa se parará mostrando el error, por lo que existen formas de "cazar" (`catch`) esas excepciones para evitar ese parón.

Podemos **tratar** las excepciones que lance JAVA o **crear** las nuestras propias.

>> 6.1.1- MANEJO DE EXCEPCIONES (`TRY-CATCH-FINALLY`)

· 6.1.1.2- MANEJADORES DE EXCEPCIONES

Los **manejadores de excepciones** son mecanismos que trabajan en conjunto para tratar estas. Se conocen como el **bloque** `try-catch-finally`.

```
try{
    ...
}
catch(TipoException variable){
    ...
}
finally{
    ...

}
```

`try{ ... }`	El manejador **`try`** guarda **el código que queremos ejecutar** en el cual pueden darse excepciones.
`catch(TipoException variable){ ... }`	El manejador **`catch`** ejecutará su código si en el **`try`** se da la excepción especificada según su tipo.
`finally{ ... }`	El manejador **`finally`** ejecutará el código se dé o no la excepción. **No es obligatorio** y muchas veces no se pone.

El flujo de código es el siguiente:

TRY	>	CATCH	>	FINALLY
Se ejecuta el código Si en alguna línea se lanza una excepción, salta directamente al **`CATCH`**. Si no, continúa normal.		Se analiza la excepción y se ejecuta el código relacionado.		Una vez terminado el **`try-catch`**, el código sale y continúa.

EJEMPLO

Ejemplo.java

```
1  public class Ejemplo{
2     public static void main(String[] args){
3
4        int div;
5        int x = 6;
6        int y = 0;
7
8        div = x/y;
9
10       System.out.println("El resultado es "+div);
11
12       System.out.println("-Fin del programa-");
13    }
14 }
```

Console

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Ejemplo.main(Ej00ExTry.java:8)
```

En este ejemplo puedes ver que al ejecutar salta un error y la ejecución se detiene. Vamos a probar a hacer **`try-catch`** para que esto no sea tan agresivo.

EJEMPLO

Ejemplo.java

```
1  public class Ejemplo{
2     public static void main(String[] args){
3
4        int div;
5        int x = 6;
6        int y = 0;
7
8        try{
9           div = x/y;
10          System.out.println("El resultado es "+div);
11       }
12       catch(ArithmeticException e){
13          System.out.println("(!) No se puede dividir entre 0.");
```

```
14          }
15
16          System.out.println("-Fin del programa-");
17      }
18  }
19
```

Console

```
(!) No se puede dividir entre 0.
-Fin del programa-
```

Ahora mucho mejor. Podemos ver que **como se ha dado una excepción en el `try`**, el programa **ha saltado a los `catch`**, buscando la primera que corresponda y ejecutando el interior, que en este caso es un **`syso`** con un mensaje. Después de esto, el programa termina porque no tiene nada más, pero si tuviese el código apropiado podría continuar y hacer más cosas.

· 6.1.1.2- CATCH MÚLTIPLES

Podríamos manejar todos los tipos de excepciones simplemente especificando un **`catch`** de tipo genérico **`Exception`**, pero lo ideal es manejar **cada tipo que creamos útil de diferente manera**. Para ello podemos hacer **`catch`** múltiples.

```
try{
    ...
}
catch(TipoException1 variable){
    ...
}
catch(TipoException2 variable){
    ...
}
catch(TipoException3 variable){
    ...
}
```

Podemos tener varias excepciones especificadas y luego tratar aquellas que no contemplemos con la genérica **`Exception`**, pero esta debe estar **al final del resto**, ya que si no se ejecutará la primera y no buscará el tipo específico.

EJEMPLO

Ejemplo.java

```
1  public class Ejemplo{
2      public static void main(String[] args){
3
4          Scanner keyboard = new Scanner(System.in);{
5
6              int div;
7              int x;
8              int y;
9
10             try{
11                 System.out.println("Introduce el dividendo:");
12                 x = keyboard.nextInt();
13
14                 System.out.println("Introduce el divisor:");
15                 y = keyboard.nextInt();
16
17                 div = x/y;
18
```

```
19              System.out.println("El resultado es "+div);
20          }
21          catch(ArithmeticException e){
22              System.out.println("(!) No se puede dividir entre 0.");
23          }
24          catch(InputMismatchException e){
25              System.out.println("(!) Los datos deben ser números.");
26          }
27          catch(Exception e){
28              System.out.println("(!) Ha habido algún error.");
29          }
30
31          System.out.println("-Fin del programa-");
32
33      keyboard.close();}
34  }
35  }
```

Console

```
Introduce el dividendo:
6
Introduce el divisor:
a
(!) Los datos deben ser números.
-Fin del programa-
```

Ahora tenemos **múltiples catch** para **diferentes excepciones** que se puedan dar. La introducción de datos está en el `try` porque también se tiene que comprobar el dato que recogemos, si esperamos un `int` pero el usuario introduce cualquier otro caracter, la excepción saltará ahí. En este caso ha saltado en la línea 15.

>> 6.1.2- LANZAR EXCEPCIONES

Ahora que ya hemos visto el funcionamiento con excepciones por defecto, pasemos a **lanzar nuestras propias excepciones**. De esta manera podemos controlar aún más los datos que maneja el programa para no permitir ciertos valores, como por ejemplo que al pedir una edad no se permita un número negativo o que un DNI no tenga más dígitos de lo permitido.

Lanzar excepciones **no sirve para crear excepciones nuevas**. Simplemente usamos las que nos proporciona JAVA para adaptarlas a nuestro código y manejarlas de la forma que queramos.

Puedes consultar todos los tipos de excepciones y errores en la **API** de JAVA.

Para poder lanzar una **excepción** se escribe la siguiente línea:

```
throw new TipoException(["mensaje"]);
```

`throw new`	Los indicadores que indican el **lanzamiento de una excepción** y su instanciación.
`tipoException();`	El **tipo de excepción** que queremos tratar. Aquí podemos colocar cualquiera de las excepciones que nos proporciona JAVA.
`["mensaje"]`	Como opcional, podemos **escribir un string** que actúa como **mensaje** a imprimir por consola si se da la excepción.

La **línea** de lanzamiento de excepciones resume lo siguiente:

```
1  TipoException e = new TipoException();
2  throw e;
```

Los lanzamientos de excepciones se suelen usar **dentro de métodos** y forman parte de la **comprobación de los datos**. Estos **devolverán la excepción cuando se de la condición que le indiquemos para ser tratada en el `catch`**.

No podemos lanzar dos excepciones a la vez. Cada ámbito de código sólo puede tener una línea de lanzamiento de excepción. Si queremos preparar varias para lanzar, deberán estar en ámbitos diferentes. *(Por ejemplo, no podemos tener dos `throw new ...` en un `else`. Podemos tener uno pero el otro deberá estar en su propia condición).*

· 6.1.2.1- LANZAR EXCEPCIONES DESDE MÉTODOS

Para poder hacer que un método lance una excepción, debemos indicarlo en su cabecera

```
public/private [static] tipo/void nombreDelMetodo([parámetro/s]) throws TipoExcepcion {
    ...
}
```

`throws TipoExcepcion`	Debemos indicar la palabra **`throws`** junto a **qué excepción/es lanza** el método.

Un método puede lanzar **más de una excepción**, para indicarlo simplemente listamos todos los tipos en la cabecera **separados por comas** (`,`):

```
1   ... throws TipoExcepcion1, ..., TipoExcepcionN {
```

· 6.1.2.2- LLAMAR DESDE UN MÉTODO A OTRO QUE LANZA EXCEPCIONES

Si hacemos que un método lance una excepción y **lo llamamos desde otro método**, el método que está más arriba **también debe indicar que lanzará esas excepciones**.

EJEMPLO

Excepciones.java

```
1  public class Excepciones {
2     public static void main(String[] args) throws ArithmeticException {
3        Scanner keyboard = new Scanner(System.in);{
4
5           int n;
6
7           try {
8              System.out.println("Introduce un número positivo");
9              n = keyboard.nextInt();
10             compruebaPositivo(n);
11          }
12          catch(ArithmeticException e) {
13             System.out.println(e.getMessage());
14          }
15       keyboard.close();}
16    }
17
18    public static void compruebaPositivo(int n) throws ArithmeticException {
19       if(n>0) {
20          System.out.println("> ¡Bien hecho!");
21       }
22       else{
23          throw new ArithmeticException("> ¡Eh! ¡Ese número no vale!");
24       }
```

```
25      }
26  }
```

Console

```
Introduce un número positivo: -4
> ¡Eh! ¡Ese número no vale!
```

En este ejemplo, **el método `compruebaPositivo` lanza una `ArithmeticException`**, y como **se llama desde el `main`**, el propio `main` debe llevar un **`throws Exception()`**.

No es aconsejable que el **`main`** lance excepciones porque si se produce una excepción en específico, esta **se debe lanzar desde otra clase**. El **`main`** solo debe estar preparado para recibirla y poder trabajar con ella en las cláusulas **`try-catch`**.

· 6.1.2.3- IMPRIMIR LOS MENSAJES DE LAS EXCEPCIONES

Podemos imprimir por consola los **mensajes que traen las excepciones al "cazarlas"** a través del método **`.getMessage()`**:

```
nombreExcepcionCazada.getMessage()
```

Que se puede implementar de esta manera dentro de un **`catch`**:

```
catch (InvalidParameterException e) {
    System.out.println(e.getMessage());
}
```

>> 6.1.3- CREAR EXCEPCIONES PERSONALIZADAS

Podemos crear excepciones personalizadas a través de **clases** que **extiendan de la clase `Exception`**. Su estructura es la misma que cualquier otra clase, añadiendo la especificación de **`extends`** a la cabecera, un **identificador de serial** (opcional, que suele ser calculado por JAVA) y el **método** que definirá **qué ocurre cuando se llama a la excepción**, que suele ser un mensaje a través del constructor **`super()`**.

```
public class nombreExcepcion extends Exception {

   private static final long serialVersionUID = X;

   public nombreExcepcion(){
      super("mensaje");
   }
}
```

Cada clase de excepción personalizada sólo puede tener **una excepción**. Si queremos crear más, deberemos hacer que cada una tenga su propia clase.

Para usarla, simplemente la tratamos como un **tipo más de extensión**, de la misma manera que las demás.

EJEMPLO

ExcepcionNumeroNegativo.java

```
1  public class ExcepcionNumeroNegativo extends Exception {
2
3     private static final long serialVersionUID = 1L;
4
5     public ExcepcionNumeroNegativo(){
6        super("Los números no pueden ser negativos.");
```

```
    }
}
```

Excepciones.java

```
public class Excepciones {
    public static void main(String[] args) throws ExcepcionNumeroNegativo {
        Scanner keyboard = new Scanner(System.in);{

            int n;

            try {
                System.out.println("Introduce un número positivo");
                n = keyboard.nextInt();
                compruebaPositivo(n);
            }
            catch(ExcepcionNumeroNegativo e) {
                System.out.println(e.getMessage());
            }
        keyboard.close();}
    }

    public static void compruebaPositivo(int n) throws ExcepcionNumeroNegativo {
        if(n>0) {
            System.out.println("> ¡Bien hecho!");
        }
        else{
            throw new ExcepcionNumeroNegativo();
        }
    }
}
```

Console

```
Introduce un número positivo: -4
Los números no pueden ser negativos.
```

Como ves, hemos **cambiado la excepción** del ejemplo anterior por una **personalizada**.

>> 6.1.4- EJEMPLO DEL USO DE EXCEPCIONES

Veamos un ejemplo con todo lo que hemos visto sobre excepciones:

EJEMPLO

MetodosConExcepciones.java

```
import java.security.InvalidParameterException;

public class MetodosConExcepciones {

    public static void imprimePositivo(int p) throws InvalidParameterException {
        if(p>0) {
            System.out.println("> ¡Ese número es positivo!");
        }
        else{
            throw new InvalidParameterException("> ¡Eh! ¡Ese número no vale!");
        }
    }

    public static void imprimeNegativo(int p) throws InvalidParameterException {
        if(p<=0) {
            System.out.println("> ¡Ese número es negativo!");
        }
        else{
            throw new InvalidParameterException("> ¡Eh! ¡Ese número no vale!");
        }
```

```
21      }
22  }
23
```

PruebaExcepciones.java

```
 1  import java.security.InvalidParameterException;
 2  import java.util.InputMismatchException;
 3  import java.util.Scanner;
 4
 5  public class PruebaExcepciones {
 6     public static void main(String[] args) {
 7        Scanner keyboard = new Scanner(System.in);{
 8
 9           int n = 1;
10
11           while(true){
12              do {
13                 System.out.println("Introduce un número positivo: ");
14
15                 try {
16                    n = keyboard.nextInt();
17                    MetodosConExcepciones.imprimePositivo(n);
18                 }
19                 catch(InputMismatchException e) {
20                    System.err.println("(!) Carácter no válido.");
21                    n = -1;
22                 }
23                 catch (InvalidParameterException e) {
24                    System.out.println(e.getMessage());
25                 }
26
27                 keyboard.nextLine();
28                 System.out.println();
29
30              }while(n<0);
31
32              do {
33                 System.out.print("Introduce un número negativo: ");
34
35                 try {
36                    n = keyboard.nextInt();
37                    MetodosConExcepciones.imprimeNegativo(n);
38                 }
39                 catch(InputMismatchException e) {
40                    System.err.println("(!) Carácter no válido.");
41                    n = 1;
42                 }
43                 catch (InvalidParameterException e) {
44                    System.out.println(e.getMessage());
45                 }
46                 keyboard.nextLine();
47                 System.out.println();
48              }while(n>=0);
49           }
50        keyboard.close();}
51     }
52  }
```

Console

```
Introduce un número positivo: 2
> ¡Ese número es positivo!

Introduce un número negativo: -2
> ¡Ese número es negativo!

Introduce un número positivo: -4
> ¡Eh! ¡Ese número no vale!
```

```
Introduce un número positivo: 2
> ¡Ese número es positivo!

Introduce un número negativo: ...
```

Como has podido observar, este es un programa que **irá pidiendo un número positivo y otro negativo y comprobará** si se han introducido correctamente. Al introducir un dato erróneo, según el método, **lanzan excepciones** e imprimirá el mensaje que se les ha especificado en la clase **`MetodosConExcepciones`**.

U6 - **BATERÍA DE EJERCICIOS**

1	Implementa un programa que pida al usuario un valor entero A utilizando un nextInt() y luego muestre por pantalla el mensaje “Valor introducido: …”. Se deberá tratar la excepción InputMismatchException que lanza nextInt() cuando no se introduce un entero válido. En tal caso se mostrará el mensaje “Valor introducido incorrecto”.
2	Implementa un programa que pida dos valores int A y B utilizando un nextInt() (de Scanner), calcule A/B y muestre el resultado por pantalla. Se deberán tratar de forma independiente las dos posibles excepciones, InputMismatchException y ArithmeticException, mostrando en cada caso un mensaje de error diferente en cada caso.
3	Implementa un programa que cree un array tipo double de tamaño 5 y lo rellene con 5 valores que solicite al usuario. Tendrás que manejar la/las posibles excepciones y seguir pidiendo valores hasta rellenar completamente el vector.
4	Implementa un programa que cree un vector de enteros de tamaño N (número aleatorio entre 1 y 100) con valores aleatorios entre 1 y 10. Luego se le preguntará al usuario qué posición del vector quiere mostrar por pantalla, repitiéndose una y otra vez hasta que se introduzca un valor negativo. Maneja todas las posibles excepciones.
5	Implementa un programa con tres funciones: ◦ void imprimePositivo(int p): Imprime el valor p. Lanza una ‘Exception’ si p < 0 ◦ void imprimeNegativo(int n): Imprime el valor n. Lanza una ‘Exception’ si p >= 0 ◦ La función main para realizar pruebas. Puedes llamar a ambas funciones varias veces con distintos valores, hacer un bucle para pedir valores por teclado y pasarlos a las funciones, etc. Maneja las posibles excepciones.
6	Implementa una clase Gato con los atributos nombre y edad, un constructor con parámetros, los getters y setters, además de un método imprimir() para mostrar los datos de un gato. El nombre de un gato debe tener al menos 3 caracteres y la edad no puede ser negativa. Por ello, tanto en el constructor como en los setters, deberás comprobar que los valores sean válidos y lanzar una ‘Exception’ si no lo son. Luego, haz una clase principal con main para hacer pruebas: instancia varios objetos Gato y utiliza sus setters, probando distintos valores (algunos válidos y otros incorrectos). Maneja las excepciones