

UNIDAD 8.

EXCEPCIONES

Programación
CFGS DAW

ÍNDICE DE CONTENIDO

1. Introducción.....	4
1.1 Ejemplo 1.....	5
1.2 Ejemplo 2.....	6
1.3 Ejemplo 3.....	7
2. Lanzar excepciones (throw).....	8
2.1 ¿Por qué lanzar excepciones?.....	8
2.2 Cómo lanzar una excepción.....	8
2.3 Indicar la excepción en la cabecera del método.....	9
2.4 Lanzando distintos tipos de excepciones.....	10
2.5 Las excepciones se lanzan de método a método (la patata caliente).....	10
3. Manejar excepciones (try – catch – finally).....	11
3.1 Manejadores de excepciones.....	11
3.2 Ejemplo 4.....	12
3.3 Particularidades de la cláusula <i>catch</i>	12
3.4 Cláusulas <i>catch</i> múltiples.....	13
3.5 Ejemplo 5.....	14
3.6 El objeto <i>Exception</i>	15
4. Jerarquía y tipos de excepciones Java.....	16
5. Definir excepciones propias.....	18
5.1 Ejemplo 6.....	18
6. Agradecimientos.....	20

UD10. EXCEPCIONES

1. INTRODUCCIÓN

Una excepción es un error semántico que se produce en tiempo de ejecución. Aunque un código sea correcto sintácticamente (es código Java válido y puede compilarse), es posible que durante su ejecución se produzcan errores inesperados, como por ejemplo:

- Dividir por cero.
- Intentar acceder a una posición de un array fuera de sus límites.
- Al llamar al `nextInt()` de un `Scanner` el usuario no introduce un valor entero.
- Intentar acceder a un fichero que no existe o que está en un disco duro corrupto.
- Etc.

Cuando esto ocurre, la máquina virtual Java crea un objeto de la clase **Exception** (las excepciones en Java son objetos) y se notifica el hecho al sistema de ejecución. Se dice que se ha lanzado una excepción ("**Throwing Exception**"). Existen también los errores internos que son objetos de la clase **Error** que no estudiaremos. Ambas clases **Error** y **Exception** son clases derivadas de la clase base **Throwable**.

Un método se dice que es capaz de tratar una excepción ("**Catch Exception**") si ha previsto el error que se ha producido y las operaciones a realizar para "recuperar" el programa de ese estado de error. No es suficiente capturar la excepción, si el error no se trata tan solo conseguiremos que el programa no se pare, pero el error puede provocar que los datos o la ejecución no sean correctos.

En el momento en que es lanzada una excepción, la máquina virtual Java recorre la pila de llamadas de métodos en busca de alguno que sea capaz de tratar la clase de excepción lanzada. Para ello, comienza examinando el método donde se ha producido la excepción; si este método no es capaz de tratarla, examina el método desde el que se realizó la llamada al método donde se produjo la excepción y así sucesivamente hasta llegar al último de ellos. En caso de que ninguno de los métodos de la pila sea capaz de tratar la excepción, la máquina virtual Java muestra un mensaje de error y el programa termina.

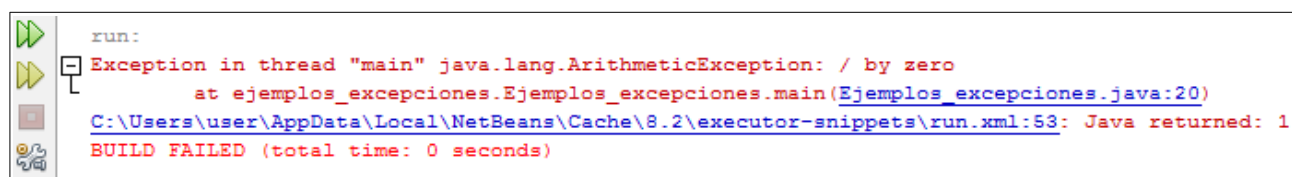
Los programas escritos en Java también pueden lanzar excepciones explícitamente mediante la instrucción **throw**, lo que facilita la devolución de un "código de error" al método que invocó el método que causó el error.

1.1 Ejemplo 1

Como primer encuentro con las excepciones, vamos a ejecutar el siguiente programa. En él vamos a forzar una excepción al intentar dividir un número entre 0:

```
12 public class Ejemplos_excepciones {  
13  
14     public static void main(String[] args) {  
15         int div, x, y;  
16  
17         x = 3;  
18         y = 0;  
19  
20         div = x / y;  
21  
22         System.out.println("El resultado es " + div);  
23     }  
24  
25 }
```

Siendo la salida:



```
run:  
Exception in thread "main" java.lang.ArithmeticException: / by zero  
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:20)  
    C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

Lo que ha ocurrido es que la máquina virtual Java (el programa que ejecuta código Java) ha detectado una condición de error, la división por 0, y ha creado un objeto de la clase *java.lang.ArithmeticException*. Como el método donde se ha producido la excepción no es capaz de tratarla, la máquina virtual Java finaliza el programa en la línea 20 y muestra un mensaje de error con la información sobre la excepción que se ha producido.

1.2 Ejemplo 2

A continuación vamos a forzar una excepción de conversión, para ello vamos a intentar pasar a entero una cadena que no sólo lleva caracteres numéricos:

```
12 public class Ejemplos_excepciones {  
13  
14     public static void main(String[] args) {  
15         String cadena = "56s";  
16         int num;  
17  
18         num = Integer.parseInt(cadena);  
19  
20         System.out.println("El número es " + num);  
21     }  
22  
23 }
```

Siendo la salida:

```
run:  
Exception in thread "main" java.lang.NumberFormatException: For input string: "56s"  
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)  
    at java.lang.Integer.parseInt(Integer.java:580)  
    at java.lang.Integer.parseInt(Integer.java:615)  
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)  
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

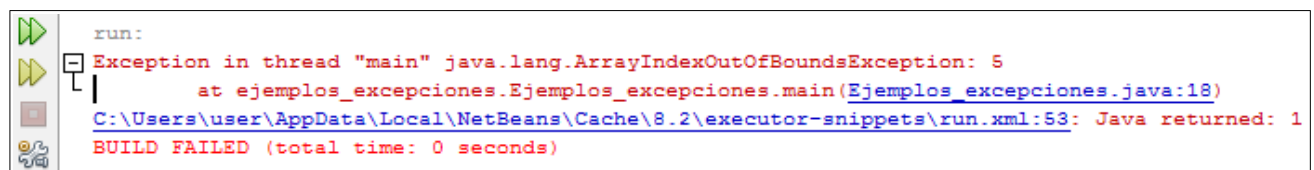
Debido a que la cadena no tiene el formato adecuado ("56s" no representa un número válido), el método `Integer.parseInt(...)` no puede convertirla a un valor de tipo `int` y lanza la excepción *NumberFormatException*. La máquina virtual Java finaliza el programa en la línea 18 y muestra por pantalla la información sobre la excepción que se ha producido.

1.3 Ejemplo 3

En este ejemplo vamos a forzar una excepción de límites del vector, para ello vamos a crear un vector e intentar acceder a una posición que no existe:

```
12 public class Ejemplos_excepciones {  
13  
14     public static void main(String[] args) {  
15         int v[] = {1,2,3};  
16         int elem;  
17  
18         elem = v[5];  
19  
20         System.out.println("El elemento es " + elem);  
21     }  
22 }  
23  
24 }
```

Siendo la salida:



```
run:  
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5  
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)  
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1  
BUILD FAILED (total time: 0 seconds)
```

Al intentar acceder a una posición que sobrepasa el tamaño del vector se produce una excepción de tipo `ArrayIndexOutOfBoundsException`. La máquina virtual de java finaliza el programa en la línea 18 y muestra el mensaje de error sobre la excepción que se ha producido.

2. LANZAR EXCEPCIONES (THROW)

2.1 ¿Por qué lanzar excepciones?

Un programador puede programar su código de forma que se lancen excepciones cuando se intente hacer algo incorrecto o inesperado (en ocasiones es recomendable). Por ejemplo, cuando los argumentos que se le pasan a un método no son correctos o no cumplen ciertos criterios.

Esto es habitual en la Programación Orientada a Objetos (POO): Recordad que una clase debe ser la responsable de la lógica de sus objetos: asegurar que los datos sean válidos, además de controlar qué está permitido y no está permitido hacer. **Por ejemplo si se instancia una clase con valores incorrectos** como un objeto Persona con un DNI no válido, una edad negativa, una cuenta bancaria con saldo negativo, etc. En esos casos **es conveniente que el constructor lance una excepción**.

También **puede ser apropiado lanzar excepciones en los setters** si el valor no es válido, **y en cualquier otro método en el que se intente hacer algo no permitido o que viole la integridad del objeto** como por ejemplo retirar dinero de una cuenta sin saldo suficiente.

⚡ Téngase en cuenta que **las excepciones pueden manejarse y controlarse sin que el programa se pare** (en el apartado 3 se explica cómo).

Es decir, lanzar una excepción no implica necesariamente que el programa terminará, es simplemente es una forma de avisar de un error. **Quien llame al método es responsable de manejar la excepción para que el programa no se pare**.

2.2 Cómo lanzar una excepción

Para lanzar la excepción se utiliza la palabra reservada **throw** seguido de un objeto de tipo *Exception* (o alguna de sus subclases como *ArithmeticException*, *NumberFormatException*, *ArrayIndexOutOfBoundsException*, etc.). Como las excepciones son objetos, deben instanciarse con “new”. Por lo tanto, **podemos lanzar una excepción genérica así**:

```
throw new Exception();
```

Esto es equivalente a primero instanciar el objeto *Exception* y luego lanzarlo:

```
Exception e = new Exception();  
throw e;
```

El constructor de *Exception* permite (opcionalmente) un argumento *String* para dar detalles sobre el problema. Si la excepción no se maneja y el programa se para, el mensaje de error se mostrará por la consola (esto es muy útil para depurar programas).

```
throw new Exception("La edad no puede ser negativa");
```

En lugar de lanzar excepciones genéricas (Exception) también es posible lanzar excepciones específicas de Java como por ejemplo `ArrayIndexOutOfBoundsException`, `ArithmeticException`, `NumberFormatException`, etc. En Java todas las clases de excepciones heredan de `Exception`.

```
throw new NumberFormatException("...");
```

⚡ En lugar de lanzar excepciones propias de Java (`ArithmeticException`, `NumberFormatException`, `ArrayIndexOutOfBoundsException`, etc.) **normalmente es preferible lanzar excepciones genéricas 'Exception' o mejor aún, utilizar nuestras propias excepciones** (apartado 6).

2.3 Indicar la excepción en la cabecera del método

Es obligatorio indicar en la cabecera del método que puede lanzar excepciones. Para ello hay que añadir, a la derecha de la cabecera y antes de las llaves, la palabra reservada **throws** seguido del **tipo de excepción** que puede lanzar (si puede lanzar distintos tipos de excepciones deben indicarse todas separadas por comas).

Por ejemplo, veamos el **método `setEdad(int edad)`** de la **clase `Persona`**. Como hemos decidido que la edad de una persona no puede ser negativa, lanzaremos una excepción si `edad < 0`.

```
// Setter del atributo edad. Debe ser >= 0
public void setEdad(int edad) throws Exception {
    if (edad < 0)
        throw new Exception("Edad negativa no permitida");
    else
        this.edad = edad;
}
```


Hay que tener en cuenta que **al lanzar una excepción se parará la ejecución de dicho método (no se ejecutará el resto del código del método) y se lanzará la excepción al método que lo llamó**. Si por ejemplo desde la función `main` llamamos a `setEdad()`, como puede suceder que `setEdad()` lance una excepción, entonces en la práctica es posible que el `main` lance una excepción (no directamente con un `throw`, sino por la excepción que nos lanza `setEdad()`). Por lo tanto, también tenemos que especificar en el `main` que se puede lanzar una excepción:

```
public static void main(String[] args) throws Exception {
    Persona p = new Persona("44193900L", "Pepito", 27);
    ...
    ...
    p.setEdad(valor); // Puede lanzar una excepción
}
```


2.4 Lanzando distintos tipos de excepciones

Un método puede lanzar distintos tipos de excepciones (si lo consideramos necesario). En tal caso hay que especificar todos los tipos posibles en la cabecera, separados por comas. Por ejemplo, imaginemos que el constructor de Persona toma como argumentos el dni y la edad, y queremos lanzar excepciones distintas según cada caso.

```
public Persona(String dni, int edad) throws InvalidDniException, InvalidEdadException {  
    if (!dni.matches("[0-9]{8}[A-Z]")) {  
        throw new InvalidDniException("DNI no válido: " + dni);  
    }  
    if (edad < 0) {  
        throw new InvalidEdadException("Edad no válida: " + edad);  
    }  
  
    this.dni = dni;  
    this.edad = edad;  
}
```

 Ojo, téngase en cuenta que *InvalidEdadException* e *InvalidDniException* no son tipos de excepciones de Java, sino excepciones propias que nosotros habríamos definido a medida para nuestro software (ver apartado 6).

2.5 Las excepciones se lanzan de método a método (la patata caliente)


Observese que el lanzamiento de excepciones se produce recursivamente a través de la secuencia de llamadas a métodos. Imaginemos que en el método A llamamos al método B, que llama al método C, etc. hasta llegar a E.

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$ (secuencia de llamadas de métodos)

Si el método E lanza una excepción, esta le llegará a D que a su vez se la lanzará a C, etc. recorriendo el camino inverso hasta llegar al método inicial A.

$A \leftarrow B \leftarrow C \leftarrow D \leftarrow E$ (secuencia de lanzamiento de excepciones)

Por lo tanto, como todos estos métodos pueden acabar lanzando una excepción, en sus cabeceras habrá que incluir el **throws Exception** (o el que corresponda según el tipo de excepción).

 ¡Ojo! No es recomendable dejar que las Excepciones de nuestro software lleguen de forma descontrolada hasta el main y terminen el programa.
Lo ideal es manejar las excepciones como veremos en el siguiente apartado.

3. MANEJAR EXCEPCIONES (TRY – CATCH – FINALLY)

3.1 Manejadores de excepciones

En Java se pueden manejar excepciones utilizando tres mecanismos llamados **manejadores de excepciones**. Existen tres y funcionan conjuntamente:

- Bloque **try** (intentar): código que podría lanzar una excepción.
- Bloque **catch** (capturar): código que manejará la excepción si es lanzada.
- Bloque **finally** (finalmente): código que se ejecuta tanto si hay excepción como si no.



Un **manejador de excepciones** es una bloque de código encargado de tratar las excepciones para intentar recuperarse del fallo y **evitar que la excepción sea lanzada descontroladamente hasta el main y termine el programa**.

Siempre que se utilice un **try** es obligatorio utilizar al menos un **catch**. El **finally** es opcional.

```
try {  
    // Instrucciones que podrían lanzar una excepción.  
}  
catch (TipoExcepción nombreVariable) {  
    // Instrucciones que se ejecutan cuando 'try' lanza una excepción.  
}  
finally {  
    // Instrucciones que se ejecutan tanto si hay excepción como si no.  
}
```

El bloque try intentará ejecutar el código. Si se produce una excepción se abandona dicho bloque (no se ejecutarán las demás instrucciones del try) y se saltará al bloque catch. Lógicamente, si en el try no se produce ninguna excepción el bloque catch se ignora.

El bloque catch capturará las excepciones del tipo *TipoExcepción*, evitando que sea lanzada al método que nos llamó. Aquí deberemos escribir las instrucciones que sean necesarias para manejar el error. Pueden especificarse varios bloques catch para distintos tipos de excepciones.

El bloque finally es opcional y se ejecutará tanto si se ha lanzado una excepción como si no.

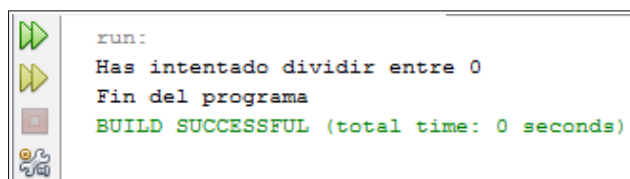
Veamos un ejemplo sencillo.

3.2 Ejemplo 4

Vamos a producir una excepción y tratarla haciendo uso de los manejadores *try-catch*:

```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int x = 1, y = 0;
16
17         try
18         {
19             int div = x / y;
20
21             System.out.println("La ejecución no llegará aquí.");
22         }
23         catch(ArithmeticException ex)
24         {
25             System.out.println("Has intentado dividir entre 0");
26         }
27
28         System.out.println("Fin del programa");
29     }
30 }
```

Siendo la salida:



```
run:
Has intentado dividir entre 0
Fin del programa
BUILD SUCCESSFUL (total time: 0 seconds)
```

Al intentar dividir por cero se lanza automáticamente una *ArithmeticException*. Como esto sucede dentro del bloque *try*, la ejecución del programa pasa al primer bloque *catch* porque coincide con el tipo de excepción producida (*ArithmeticException*). Se ejecutará el código del bloque *catch* y luego el programa continuará con normalidad.

3.3 Particularidades de la cláusula *catch*

Es importante entender que **el bloque *catch* solo capturará excepciones del tipo indicado**. Si se produce una excepción distinta no la capturará. Sin embargo **capturará excepciones heredadas del tipo indicado**. Por ejemplo, *catch (ArithmeticException e)* capturará cualquier tipo de excepción que herede de *ArithmeticException*. El caso más general es *catch (Exception e)* que capturará todo tipo de excepciones porque **en Java todas las excepciones heredan de *Exception***.

Sin embargo, es mejor utilizar excepciones lo más cercanas al tipo de error previsto, ya que lo que se pretende es recuperar el programa de alguna condición de error y si "se meten todas las excepciones en el mismo saco", seguramente habrá que averiguar después qué condición de error se produjo para poder dar una respuesta adecuada.

El objetivo de una cláusula *catch* es resolver la condición excepcional para que el programa pueda continuar como si el error nunca hubiera ocurrido.

3.4 Cláusulas *catch* múltiples

Se pueden especificar varias cláusulas *catch*, tantas como queramos, para que cada una capture un tipo diferente de excepción.

```
try {  
    // instrucciones que pueden producir distintos tipos de Excepciones  
}  
catch (TipoExcepción1 e1) {  
    // instrucciones para manejar un TipoExcepción1  
}  
catch (TipoExcepción2 e2) {  
    // instrucciones para manejar un TipoExcepción2  
}  
...  
}  
catch (TipoExcepciónN eN) {  
    // instrucciones para manejar un TipoExcepciónN  
}  
finally { // opcional  
    // instrucciones que se ejecutarán tanto si hay excepción como si no  
}
```

Cuando se lanza una excepción dentro del *try*, se comprueba cada sentencia *catch* en orden y se ejecuta la primera cuyo tipo coincida con la excepción lanzada. Los demás bloques *catch* serán ignorados. Luego se ejecutará el bloque *finally* (si se ha definido) y el programa continuará su ejecución después del bloque *try-catch-finally*.

⚡ Si el tipo excepción producida no coincide con ninguno de los *catch*, entonces la excepción será lanzada al método que nos llamó.

Veamos un ejemplo con cláusulas *catch* múltiples.

3.5 Ejemplo 5

Vamos a ejecutar distintas instrucciones dentro del bloque *try* y manejaremos las excepciones de división por cero y de si sobrepasamos el tamaño del vector.

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         int x, y, div, pos;
18         int[] v = {1,2,3};
19         Scanner in = new Scanner(System.in);
20
21         try
22         {
23             System.out.print("Introduce el numerador: ");
24             x = in.nextInt();
25
26             System.out.print("Introduce el denominador: ");
27             y = in.nextInt();
28
29             div = x / y;
30
31             System.out.println("La división es " + div );
32
33             System.out.print("Introduce la posición del vector a consultar: ");
34             pos = in.nextInt();
35
36             System.out.println("El elemento es " + v[pos] );
37
38         }
39         catch(ArithmeticException ex)
40         {
41             System.out.println("División por cero: " + ex);
42         }
43         catch(ArrayIndexOutOfBoundsException ex)
44         {
45             System.out.println("Sobrepasado el tamaño del vector: " + ex);
46         }
47
48         System.out.println("Fin del programa");
49     }
50 }
```

Pueden suceder tres cosas diferentes:

- El *try* se ejecuta sin excepciones, se ignoran los *catch* y se imprime “Fin del programa”.
- Se produce la excepción de división por cero (línea 29), el flujo de ejecución salta al 1er *catch*, se imprime el mensaje “División por cero...” y luego “Fin del programa”.
- Se produce la excepción de sobrepasar el vector (línea 36), el flujo de ejecución salta al 2º *catch*, se imprime el mensaje “Sobrepasado el tamaño del vector...” y “Fin del programa”.

3.6 El objeto Exception

Toda excepción genera un objeto de la clase Exception (o uno más específico que hereda de Exception). Dicho objeto contendrá detalles sobre el error producido. Puede ser interesante mostrar esta información para que la vea el usuario (que sepa qué ha sucedido) o el desarrollador (para depurar y corregir el código si es pertinente). En la cláusula catch tenemos acceso al objeto en caso de que queramos utilizarlo:

Los dos métodos de Exception más útiles son:

- **getMessage()** → Devuelve un String con un texto simple sobre el error.
- **printStackTrace()** → Es el que más información proporciona. Indica qué tipo de Excepción se ha producido, el mensaje simple, y también toda la pila de llamadas. Esto es lo que hace Java por defecto cuando una excepción no se maneja y acaba parando el programa.

```
...
catch (Exception e){

    // Mostramos el mensaje de la excepción
    System.err.println("Error: " + e.getMessage());

    // Mostramos toda la información, mensaje y pila de llamadas
    e.printStackTrace();
}
...
```

Los objetos de tipo Exception tienen sobrecargado el método toString() por lo que también es posible imprimirlos directamente mediante println().

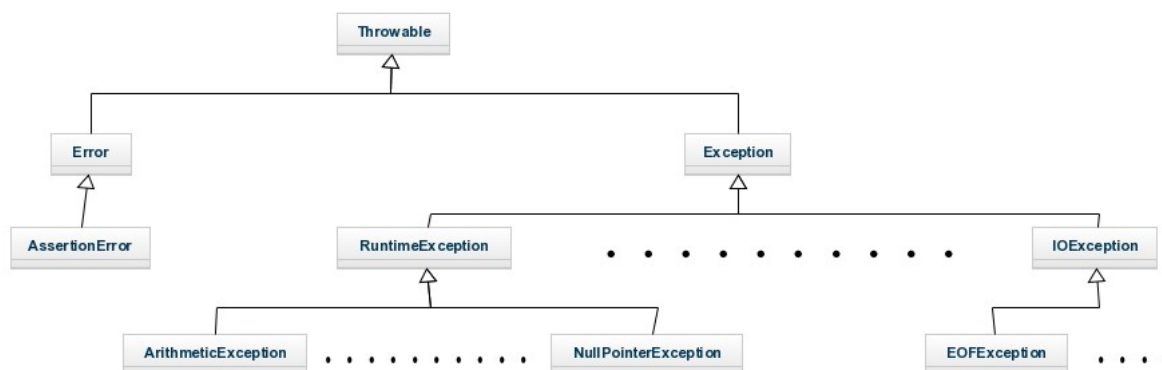
```
...
catch (Exception e){

    // Mostramos el mensaje de la excepción
    System.out.println(e);

}
...
```

4. JERARQUÍA Y TIPOS DE EXCEPCIONES JAVA

La clase `Exception` hereda de `Throwable`, y a su vez, todas las excepciones heredan de `Exception`.



Como *java.lang* es importado de forma implícita en todos los programas, la mayor parte de las excepciones derivadas de *RuntimeException* están disponibles de forma automática. Además **no es necesario incluirlas en ninguna cabecera de método mediante *throws***.

Las Excepciones pueden ser comprobadas y no comprobadas:


- **Excepciones comprobadas:** aquellas que Java comprueba durante la compilación, antes de la ejecución del programa.
- **Excepciones no comprobadas:** aquellas que Java no puede comprobar durante la compilación y se producirán durante la ejecución del programa.

Las **excepciones comprobadas** definidas en *java.lang* son:

Excepción	Significado
<i>ClassNotFoundException</i>	No se ha encontrado la clase.
<i>CloneNotSupportedException</i>	Intento de duplicado de un objeto que no implementa la interfaz clonable.
<i>IllegalAccessException</i>	Se ha denegado el acceso a una clase.
<i>InstantiationException</i>	Intento de crear un objeto de una clase abstracta o interfaz.
<i>InterruptedException</i>	Hilo interrumpido por otro hilo.
<i>NoSuchFieldException</i>	El campo solicitado no existe.
<i>NoSuchMethodException</i>	El método solicitado no existe.

Las **subclases de *RuntimeException* no comprobadas** son:

Excepción	Significado
<i>AithmeticException</i>	Error aritmético como división entre cero.
<i>ArrayIndexOutOfBoundsException</i>	Índice de la matriz fuera de su límite.
<i>ArrayStoreException</i>	Asignación a una matriz de tipo incompatible.
<i>ClassCastException</i>	Conversión inválida.
<i>IllegalArgumentException</i>	Uso inválido de un argumento al llamar a un método.
<i>IllegalMonitorStateException</i>	Operación de monitor inválida, como esperar un hilo no bloqueado.
<i>IllegalStateException</i>	El entorno o aplicación están en un estado incorrecto.
<i>IllegalThreadStateException</i>	La operación solicitada es incompatible con el estado actual del hilo.
<i>IndexOutOfBoundsException</i>	Algún tipo de índice está fuera de su rango o de su límite.
<i>NegativeArraySizeException</i>	La matriz tiene un tamaño negativo.
<i>NullPointerException</i>	Uso incorrecto de una referencia NULL.
<i>NumberFormatException</i>	Conversión incorrecta de una cadena a un formato numérico.
<i>SecurityException</i>	Intento de violación de seguridad.
<i>StringIndexOutOfBounds</i>	Intento de sobrepasar el límite de una cadena.
<i>TypeNotPresentException</i>	Tipo no encontrado.
<i>UnsupportedOperationException</i>	Operación no admitida.

 Es interesante conocer los distintos tipos de excepciones, pero **no es necesario sabérselas de memoria ni entender exactamente cuándo se produce cada una.**

Aunque las excepciones que incorpora Java, gestionan la mayoría de los errores más comunes, es probable que el programador prefiera crear sus propios tipos de excepciones para gestionar situaciones específicas de sus aplicaciones. Tan solo hay que definir una subclase de *Exception*, que naturalmente es subclase de *Throwable*.

5. DEFINIR EXCEPCIONES PROPIAS

Cuando desarrollamos software, sobre todo al desarrollar nuestras propias clases, es habitual que se puedan producir excepciones que no estén definidas dentro del lenguaje Java. Para crear una excepción propia tenemos que definir una clase derivada de la clase base *Exception*.

El uso de esta nueva excepción es el mismo que hemos visto.

5.1 Ejemplo 6

Vamos a ver un ejemplo sencillo de definición y uso de un nuevo tipo de excepción llamada *ExcepcionPropia* que utilizaremos cuando a se le pase a 'método' un valor superior a 10.

El main y el método que lanza la excepción:

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         try
18         {
19             metodo(1);
20             metodo(20);
21         }
22         catch (ExcepcionPropia e)
23         {
24             System.out.println("capturada :" + e);
25         }
26     }
27
28     static void metodo(int n) throws ExcepcionPropia
29     {
30         System.out.println("Llamado por metodo(" + n + ")");
31
32         if (n > 10)
33             throw new ExcepcionPropia(n);
34
35         System.out.println("Finalización normal");
36     }
37 }
```

La definición de la nueva excepción.

```
12 public class ExcepcionPropia extends Exception
13 {
14     private int num;
15
16     ExcepcionPropia(int n)
17     {
18         this.num = n;
19     }
20     public String toString()
21     {
22         return "Excepcion Propia[" + this.num + "]";
23     }
24 }
25
```

Siendo la salida:

```
run:
Llamado por metodo(1)
Finalización normal
Llamado por metodo(20)
capturada :Excepcion Propia[20]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Como se puede observar la excepción se lanza cuando el número es mayor que 10 y será tratada por la nueva clase creada que hereda de Exception. Además se sobrescribe el método toString que es el encargado de mostrar el mensaje asociado a la excepción.