

## Excepciones. Bloques try, catch, finally.

Una **excepción** es una **situación anómala** que puede provocar que el programa no funcione de forma correcta o termine de forma inesperada.

Ejemplos de situaciones que provocan una excepción:

- No hay memoria disponible para asignar
- Acceso a un elemento de un array fuera de rango
- Leer por teclado un dato de un tipo distinto al esperado
- Error al abrir un fichero
- División por cero
- Problemas de Hardware

**Si la excepción no se trata, el manejador de excepciones realiza lo siguiente:**

- Muestra la descripción de la excepción.
- Muestra la traza de la pila de llamadas.
- Provoca el final del programa.

Ejemplos de código Java que provocan una excepción:

```
public static void main(String[] args) {  
    int a = 4, b=0;  
    System.out.println(a/b);  
}
```

provoca:

Exception in thread "main" java.lang.ArithmeticException: / by zero

```
public static void main(String[] args) {  
    int [] array = new int [5];  
    array[5] = 1;  
}
```

provoca:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    System.out.print("Introduce un número entero: ");  
    int n = sc.nextInt();  
    System.out.print("Número introducido: " + n);  
}
```

Se espera un int. Si por ejemplo se lee 4,5 provoca:

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 excepciones1.Excepciones1.main(Excepciones1.java:7)

Ejemplos de excepciones en Java

El manejo de excepciones proporciona una **separación entre el código básico y el código que maneja los errores**, haciéndolo más legible.

Utilizando excepciones se consigue:

- Separar las instrucciones del programa de las del tratamiento de errores.
- Evitar llenar el código del programa de instrucciones de comprobación (if, switch, etc).
- El código es más simple de escribir ya que no se fuerza al programador a comprobar los errores constantemente.

### **¿Qué ocurre cuando se produce una excepción?**

Cuando ocurre una excepción:

- La Máquina Virtual Java crea un **objeto excepción** y lo lanza. El objeto excepción creado contiene información sobre el error. La ejecución normal del programa se detiene.
- El sistema busca en el método donde se ha producido la excepción un manejador de excepciones que capture ese objeto y trate la excepción.
- Si el método no contiene un manejador para la excepción se busca en el método que llamó a este y así sucesivamente en toda la pila de llamadas.
- Cuando se encuentra un manejador apropiado se le pasa la excepción. Un manejador de excepciones es considerado apropiado si el tipo de objeto excepción lanzado es compatible al tipo que puede manejar.
- Si no se encuentra un manejador adecuado la Máquina Virtual Java muestra el error y acaba el programa.

### **JERARQUÍA DE EXCEPCIONES**

Todas las excepciones lanzadas automáticamente en un programa Java son objetos de la clase **Throwable** o de alguna de sus clases derivadas.

La clase Throwable deriva directamente de Object y tiene dos clases derivadas directas: **Error** y **Exception**.

Resumen de la jerarquía de clases derivadas de Throwable.

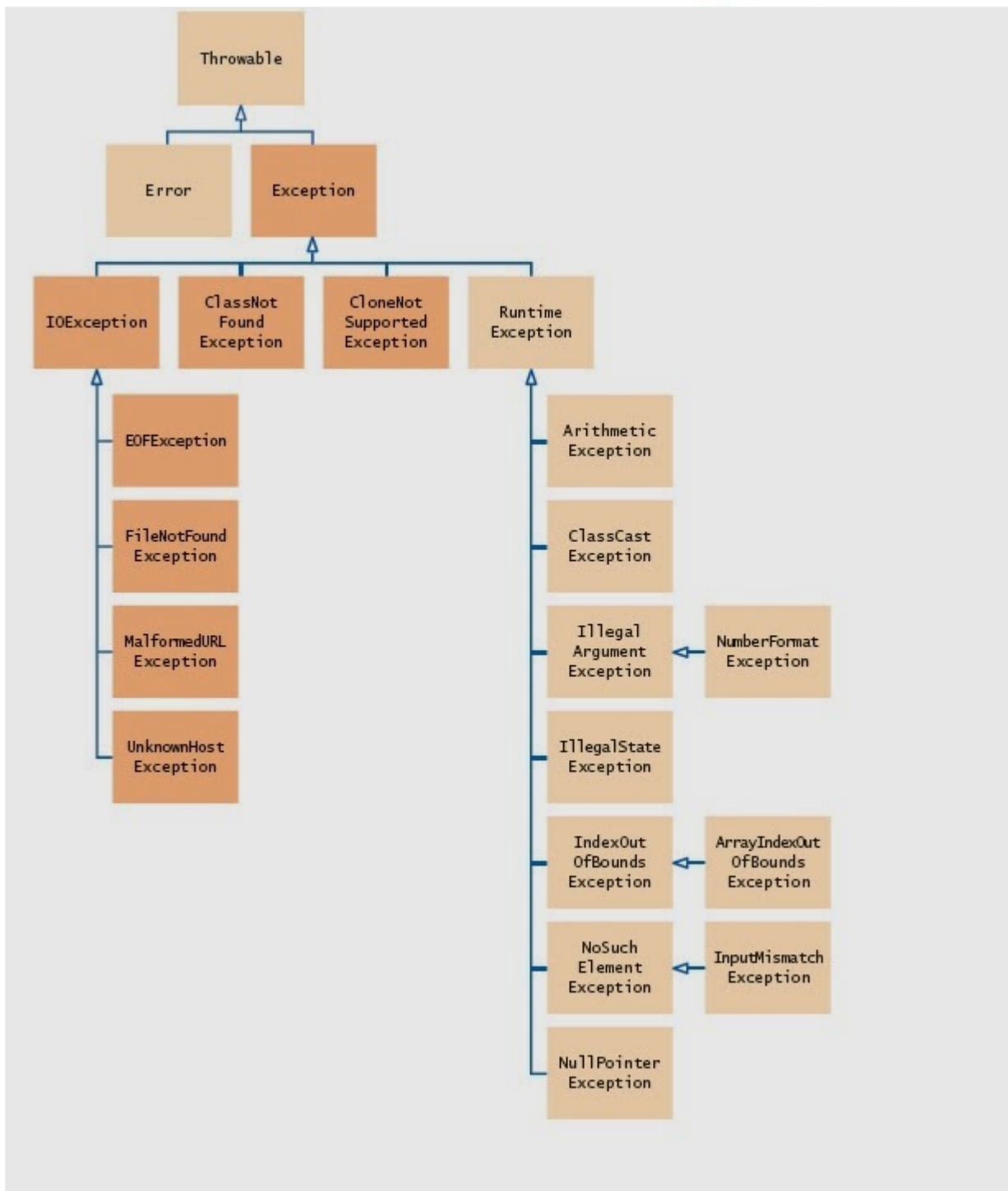


Imagen obtenida de <http://www.iro.umontreal.ca/~pift1025/bigjava/Ch15/ch15.html>

La clase **Error** está relacionada con errores de la máquina virtual de Java. Generalmente estos errores no dependen del programador por lo que no nos debemos preocupar por tratarlos, por ejemplo **OutOfMemoryError**, **StackOverflowError**, errores de hardware, etc.

En la clase **Exception** se encuentran las excepciones que se pueden lanzar en una aplicación. Tiene varias subclases entre ellas:

•**RuntimeException**: excepciones lanzadas durante la ejecución del programa. Por ejemplo: **ArithmeticException**, **NullPointerException**, **ArrayIndexOutOfBoundsException**, etc. Pertenecen al paquete **java.lang**.

•**IOException**: excepciones lanzadas al ejecutar una operación de entrada-salida. Pertenecen al paquete **java.io**.

•**ClassNotFoundException**: excepción lanzada cuando una aplicación intenta cargar una clase pero no se encuentra el fichero **.class** correspondiente

### Métodos de la clase Throwable

**Throwable** es la clase de la que derivan todos los demás tipos de excepciones.

Tiene los siguientes constructores:

**Throwable()** genera un objeto de la clase con un mensaje de error nulo.

**Throwable (String mensaje)** genera un objeto de la clase con un mensaje.

Los métodos de **Throwable** están disponibles en todas las clases que derivan de ella.

Algunos de estos métodos son:

**String getMessage()** Devuelve el mensaje que se asoció al objeto cuando se creó.

**String toString()** Devuelve una descripción del objeto. Suele indicar el nombre de la clase y el texto de **getMessage()**.

**void printStackTrace()** Es el método invocado por la máquina virtual cuando se produce una excepción del tipo **RuntimeException**. Aparece un listado con toda la pila de llamadas a métodos hasta que se llega al que provocó la excepción.

### TRATAMIENTO DE EXCEPCIONES

Bloques **try** – **catch** – **finally**

Un programa que trate las excepciones debe realizar los siguientes pasos:

1. Se **intenta ( try )** ejecutar un bloque de código.
2. Si se produce una circunstancia excepcional se **lanza ( throw )** una excepción. En caso contrario el programa sigue su curso normal.
3. Si se ha lanzado una excepción, la ejecución del programa es desviada al **manejador de excepciones** donde la excepción se **captura ( catch )** y se decide qué hacer al respecto.

El esquema general en Java es:

```
try{  
    //Instrucciones que se intentan ejecutar, si se produce una  
    //situación inesperada se lanza una excepción  
}  
catch(tipoExcepcion e){  
    //Instrucciones para tratar esta excepción  
}  
catch(otroTipoExcepcion e){  
    //Instrucciones para tratar esta excepción  
}
```

//Se pueden escribir tantos bloques catch como sean necesarios

**finally**{

// instrucciones que se ejecutarán siempre después de un bloque try

// se haya producido o no una excepción

}

### **Bloque try:**

En el bloque try se encuentran las **instrucciones que pueden lanzar una excepción**.

Solamente se pueden capturar las excepciones lanzadas dentro de un bloque try.

Una excepción se puede lanzar de forma automática o mediante la palabra reservada **throw**.

Cuando se lanza la excepción se transfiere la ejecución del programa desde el punto donde **se lanza** la excepción a otro punto donde **se captura** la excepción.

### **Bloque catch:**

Es el bloque de código donde se captura la excepción. El bloque catch es el **manejador** o **handler** de la excepción. Aquí se decide qué hacer con la excepción capturada. Puede haber varios bloques catch relacionados con un bloque try.

Una vez finalizado un bloque catch la ejecución no vuelve al punto donde se lanzó la excepción. La ejecución continúa por la primera instrucción a continuación de los bloques catch.

### **Bloque finally:**

Es opcional.

Debe aparecer a continuación de los bloques catch.

También puede aparecer a continuación de un bloque try si no hay bloques catch.

La ejecución de sus instrucciones queda garantizada independientemente de que el bloque try acabe o no su ejecución incluso en estos casos:

- Aunque el bloque try tenga una sentencia return, continue o break, se ejecutará el bloque finally
- Cuando se haya lanzado una excepción que ha sido capturada por un bloque catch. El finally se ejecuta después del catch correspondiente.
- Si se ha lanzado una excepción que no ha sido capturada, se ejecuta el finally antes de acabar el programa.

Un bloque finally se usa para dejar un estado consistente después de ejecutar el bloque try.

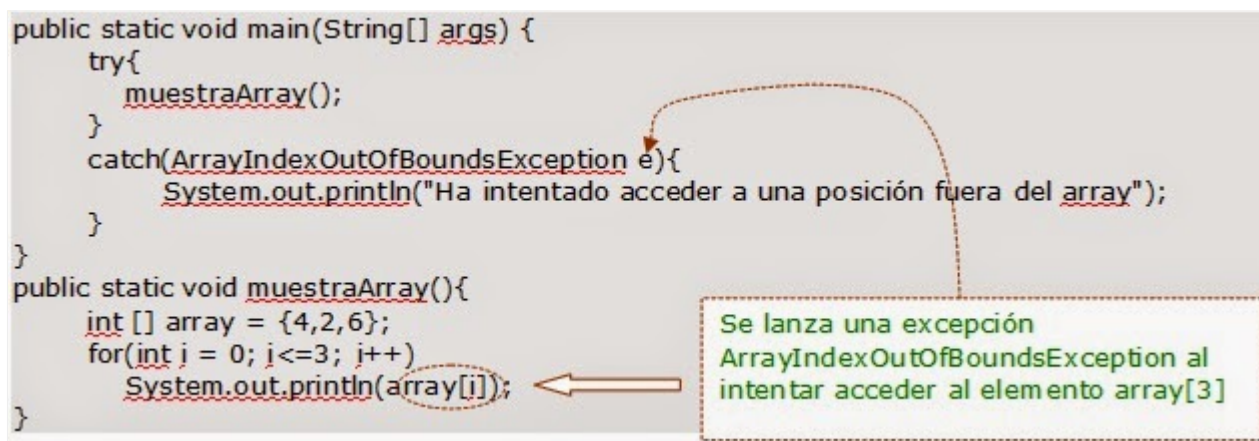
Un ejemplo de uso de bloques finally puede ser cuando estamos tratando con ficheros y se produce una excepción. Podemos escribir un bloque finally para cerrar el fichero. Este bloque se ejecutará siempre y se liberarán los recursos ocupados por el fichero.

### Ejemplo de tratamiento de excepciones:

El siguiente programa lee un número entero y lo muestra. Si en la instrucción `sc.nextInt()` se introduce un número de otro tipo o un carácter, se lanza una excepción `InputMismatchException` que es capturada por el bloque `catch`. En este bloque se realizan las instrucciones necesarias para resolver la situación y que el programa pueda continuar.

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    int n;  
    do{  
        try{  
            System.out.print("Introduce un número entero > 0: ");  
            n = sc.nextInt();  
            System.out.println("Número introducido: " + n);  
        }catch(InputMismatchException e){  
            sc.nextLine();  
            n = 0;  
            System.out.println("Debe introducir un número entero " + e.toString());  
        }  
    }while(n<=0);  
}
```

**Ejemplo de tratamiento de excepciones:** Excepción lanzada en un método y tratada en el que método que lo ha llamado.



```
public static void main(String[] args) {  
    try{  
        muestraArray();  
    }  
    catch(ArrayIndexOutOfBoundsException e){  
        System.out.println("Ha intentado acceder a una posición fuera del array");  
    }  
}  
  
public static void muestraArray(){  
    int [] array = {4,2,6};  
    for(int i = 0; i<=3; i++)  
        System.out.println(array[i]);  
}
```

Se lanza una excepción `ArrayIndexOutOfBoundsException` al intentar acceder al elemento `array[3]`

La salida del programa es:

4

2

6

Ha intentado acceder a una posición fuera del array



## CAPTURAR EXCEPCIONES

Un bloque try puede estar seguido de varios bloques catch, tantos como excepciones diferentes queramos manejar.

La estructura y el comportamiento de un bloque catch son similares al de un método.

**La excepción es capturada por el bloque catch cuyo argumento coincida con el tipo de objeto lanzado.**

La búsqueda de coincidencia se realiza sucesivamente sobre los bloques catch en el orden en que aparecen en el código hasta que aparece la primera concordancia.

Cuando acaba la ejecución de este bloque, el programa continúa después del último de los catch que sigan al bloque try que lanzó la excepción.

### Ejemplo de tratamiento de excepciones con varios catch:

En este programa se controla el error producido si se introduce un dato no entero y si se intenta acceder a una posición fuera del array.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int [] array = {4,2,6,7};
    int n;
    boolean repetir = false;
    do{
        try{
            repetir = false;
            System.out.print("Introduce un número entero > 0 y < " + array.length + " ");
            n = sc.nextInt();
            System.out.println("Valor en la posición " + n + ": " + array[n]);
        }
        catch(InputMismatchException e){
            sc.nextLine();
            n = 0;
            System.out.println("Debe introducir un número entero ");
            repetir = true;
        }
        catch(IndexOutOfBoundsException e){
            System.out.println("Debe introducir un número entero > 0 y < " + array.length + " ");
            repetir = true;
        }
    }while(repetir);
}
```

*nextInt() puede lanzar*

*array[n] puede lanzar*

El sistema de control de excepciones puede ser anidado a cualquier nivel.

Debe mantenerse la regla de que un bloque try debe ser seguido por un catch.

Pueden existir secuencias try-catch dentro de bloques try y de bloques catch.

### Ejemplo de secuencias anidadas en el bloque try:

```
public void metodo(){
    ...
    try{
```

```
try{
    ...
}
catch(tipoExcepcion e){
    ...
}
}
catch(tipoExcepcion e) {
    ...
}
}
```

### **Ejemplo de secuencias anidadas en el bloque catch**

```
public void metodo(){
    ...
    try{
        ...
    }
    catch(tipoExcepcion e){
        try{
            ...
        }
        catch(tipoExcepcion e1){
            ...
        }
    }
}
```

### **Ejemplo de anidamiento de bloques try-catch:**

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n1, n2;
    try {
        System.out.print("Introduce un número: ");
        n1 = sc.nextInt();
        try {
            System.out.print("Introduce otro número: ");
```



```
n2 = sc.nextInt();
System.out.println(n1 + " / " + n2 + " = " + n1/(double)n2);
}catch (InputMismatchException e) {
    sc.nextLine();
    n2 = 0;
    System.out.println("Debe introducir un número");
}catch (ArithmeticException e) {
    sc.nextLine();
    n2 = 0;
    System.out.println("No se puede dividir por cero");
}
}catch (InputMismatchException e) {
    sc.nextLine();
    n1 = 0;
    System.out.println("Debe introducir un número");
}
}
```

Cada catch debe tener como parámetro un objeto de la clase Throwable, de una clase derivada de ella o de una clase definida por el programador.

Cuando se lanza una excepción se captura por el primer bloque catch cuyo parámetro sea de la misma clase que el objeto excepción o de una clase base directa o indirecta. Por este motivo, **es importante el orden en que se coloquen los bloques catch**.

### **Las excepciones más genéricas se deben capturar al final**

Si no es necesario tratar excepciones concretas de forma específica se puede poner un **bloque catch de una clase base que las capture todas y las trate de forma general**. Esto se conoce como captura genérica de excepciones.

Ejemplo de captura genérica de excepciones:

Al programa anterior le añadiremos un nuevo bloque catch con un parámetro de tipo Exception. Este bloque se debe colocar al final ya que si se pone al principio capturaría cualquiera de las otras dos excepciones sin que sus bloques catch pudieran llegar a ejecutarse nunca.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int [] array = {4,2,6,7};
    int n;
    boolean repetir = false;
    do{
        try{
            repetir = false;
```

```
System.out.print("Introduce un número entero > 0 y < " + array.length + " ");
n = sc.nextInt();
System.out.println("Valor en la posición " + n + ": " + array[n]);
}catch(InputMismatchException e){
    sc.nextLine();
    n = 0;
    System.out.println("Debe introducir un número entero ");
    repetir = true;
}catch(IndexOutOfBoundsException e){
    System.out.println("Debe introducir un número entero > 0 y < " + array.length + "
");
    repetir = true;
}catch(Exception e){ //resto de excepciones de tipo Exception y derivadas
    System.out.println("Error inesperado " + e.toString());
    repetir = true;
}
}while(repetir);
}
```

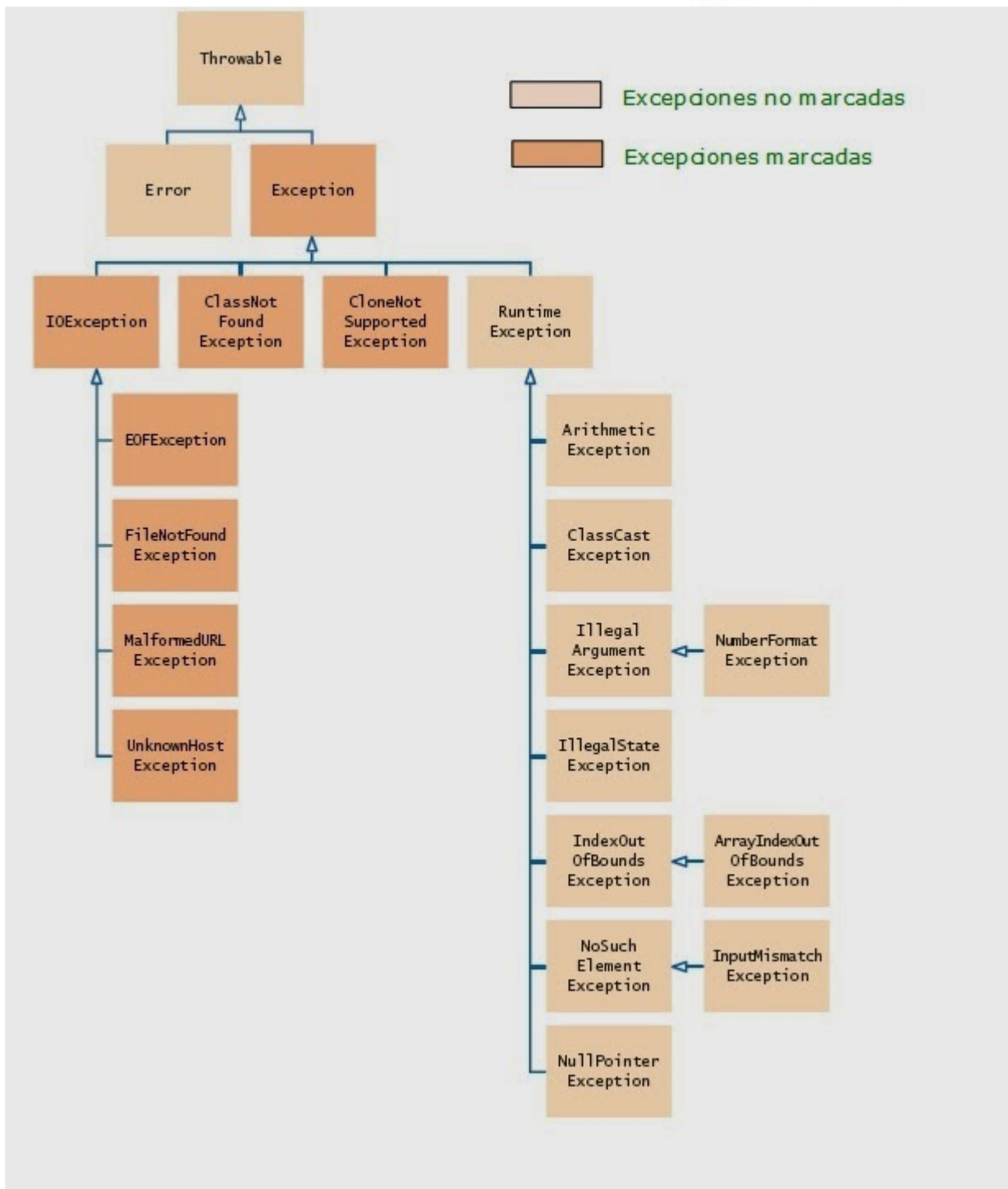
### **EXCEPCIONES MARCADAS Y NO MARCADAS (checked y unchecked)**

Excepciones no marcadas (unchecked) son aquellas que no estamos obligados a tratar. Pertenecen a la clase Error y a sus clases derivadas y a la clase RuntimeException y sus clases derivadas.

Excepciones marcadas (checked) son aquellas que estamos obligados a tratar. Son todas las clases derivadas de la clase Exception excepto las derivadas de RuntimeException.

#### **El compilador Java obliga a declarar o capturar las excepciones marcadas.**

El compilador muestra un mensaje de error si no se tratan o no se declara que son lanzadas de forma explícita.



## DECLARAR EXCEPCIONES

La declaración de las excepciones que puede lanzar un método se realiza escribiendo la palabra **throws** seguida del nombre de las excepciones separadas por comas.

Por ejemplo, la instrucción `System.in.read()` para leer un carácter lanza una excepción de tipo `IOException` que debemos capturar o declarar.

En este ejemplo no se captura la excepción por lo que estamos obligados a declararla:

```
public static void main(String[] args) throws IOException{
    char car;
```

```
System.out.println("Introduce un carácter");  
car = (char)System.in.read();  
}
```

La declaración de excepciones permite escribir métodos que no capturan las excepciones marcadas que se puedan producir.

Una excepción no capturada se lanza para que la trate algún método de la pila de llamadas.

### LANZAR EXCEPCIONES

Java permite al programador lanzar excepciones mediante la palabra reservada **throw**.

`throw objetoExcepcion;`

La excepción que se lanza es un objeto, por lo que hay que crearlo como cualquier otro objeto mediante `new`.

Ejemplo:

```
if(n==0) throw new ArithmeticException("División por cero");
```

Si en un método se usa la cláusula `throw` para lanzar un tipo de excepción marcada debe indicarse en su declaración con la cláusula `throws`.

```
public static void main(String[] args) throws IOException{  
    int n = 1;  
    char car = 'z';  
    if(car != 's') throw new IOException("Carácter no válido");  
    if(n==0) throw new ArithmeticException("División por cero");  
}
```

Se lanza una excepción marcada. Se debe declarar que el método lanza este tipo de excepciones

Se lanza una excepción no marcada. No es necesario declararla

### RELANZAR EXCEPCIONES

Si se ha capturado una excepción es posible desde el bloque `catch` relanzar la excepción (el mismo objeto recibido en el bloque `catch`) utilizando la instrucción **throw** `objetoExcepción`.

Ejemplo. Relanzar una excepción desde un método al método `main`:

```
public static void main(String[] args) {  
    try {  
        muestraArray();  
    }  
    catch (ArrayIndexOutOfBoundsException e){  
        System.out.println("Ha intentado acceder a una posición fuera del array ");  
    }  
}  
  
public static void muestraArray() {  
    Scanner sc = new Scanner(System.in);  
    int[] array = {4, 2, 6};  
    int n;  
    try {  
        System.out.println("Introduce posición ");  
        n = sc.nextInt();  
        System.out.println("Valor en esa posición: " + array[n]);  
    }  
    catch (ArrayIndexOutOfBoundsException e) {  
        throw e;  
    }  
}
```

Se lanza una excepción `ArrayIndexOutOfBoundsException` al intentar acceder a un elemento fuera del rango del array

Se relanza la excepción para que la siga tratando el método que ha llamado a éste

## CREAR NUESTRAS PROPIAS EXCEPCIONES

Aunque Java proporciona una gran cantidad de excepciones, en algunas ocasiones necesitaremos crear excepciones propias.

Las excepciones propias deben ser **subclases de la clase `Exception`**.

Normalmente crearemos excepciones propias cuando queramos manejar excepciones no contempladas por la librería estándar de Java. Por ejemplo, vamos a crear un tipo de excepción llamado `ValorNoValido` que se lanzará cuando el valor utilizado en una determinada operación no sea correcto.

La clase podría ser la siguiente:

```
public class ValorNoValido extends Exception{  
    public ValorNoValido(){ }  
    public ValorNoValido(String cadena){  
        super(cadena); //Llama al constructor de Exception y le pasa el contenido de cadena  
    }  
}
```

Un programa para probar la excepción creada podría ser este:

```
public static void main(String[] args) {  
    try {  
        double x = leerValor();  
        System.out.println("Raiz cuadrada de " + x + " = " + Math.sqrt(x));  
    }  
}
```

```
}catch (ValorNoValido e) {  
    System.out.println(e.getMessage());  
}  
}  
  
public static double leerValor() throws ValorNoValido {  
    Scanner sc = new Scanner(System.in);  
    System.out.print("Introduce número > 0 ");  
    double n = sc.nextDouble();  
    if (n <= 0) {  
        throw new ValorNoValido("El número debe ser positivo");  
    }  
    return n;  
}
```