

Tema 10 Excepciones

1. CONCEPTO DE EXCEPCIÓN

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 que provocan una excepción:

```
public class Excepciones1 {  
    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)

La forma de manejar las situaciones de error que hemos utilizado hasta ahora consiste en escribir a continuación de las instrucciones que pueden provocarlas, instrucciones de control de errores.

Por ejemplo, un método para leer un fichero y cargarlo en un array en memoria podría ser el siguiente (se escribe en pseudocódigo ya que no hemos visto aún ficheros)

```
int leerFichero() {
    codigoError = 0;
    abrir el fichero;
    si se ha abierto con éxito {
        calcular tamaño del fichero;
        si se ha podido calcular el tamaño {
            reservar memoria para el array;
            si hay memoria suficiente {
                leer el fichero y pasarlo al array;
                si se produce un error de lectura {
                    codigoError = -1;
                }
            }
        }
        sino {
            codigoError = -2;
        }
    }
    sino {
        codigoError = -3;
    }
    cerrar el fichero;
    si no se ha podido cerrar {
        codigoError = -4;
    }
}
sino {
    codigoError = -5;
}
return codigoError;
}
```

El método que llame a *leerFichero* deberá añadir instrucciones de tipo if o switch para comprobar el código de error devuelto y actuar en consecuencia.

Esta forma de tratar los errores puede llevar a obtener **código confuso**: se mezclan instrucciones de código básico con instrucciones de control de errores.

Además de los errores previstos en el método se pueden producir muchos más que no hemos previsto. Para manejarlos todos, la complejidad del código sería muy grande.



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

Método leerFichero manejando las excepciones que se puedan producir:

```
void leerFichero(){
    try{
        abrir el fichero;
        calcular tamaño del fichero;
        reservar memoria para el array;
        leer el fichero y pasarlo al array;
        cerrar el fichero;
    }
    catch(errorApertura) {tratar excepción}
    catch(errorCalculoTamaño) {tratar excepción}
    catch(errorMemoriaInsuficiente) {tratar excepción}
    catch(errorLectura) {tratar excepción}
    catch(errorCierre) {tratar excepción}
}
```

Código Básico

Tratamiento de errores

Utilizando tratamiento de 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 de excepción que puede manejar. (Similar a un método sobrecargado).
- Si no se encuentra un manejador adecuado, la Máquina Virtual Java muestra el error y acaba el programa.

2. JERARQUÍA DE EXCEPCIONES

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

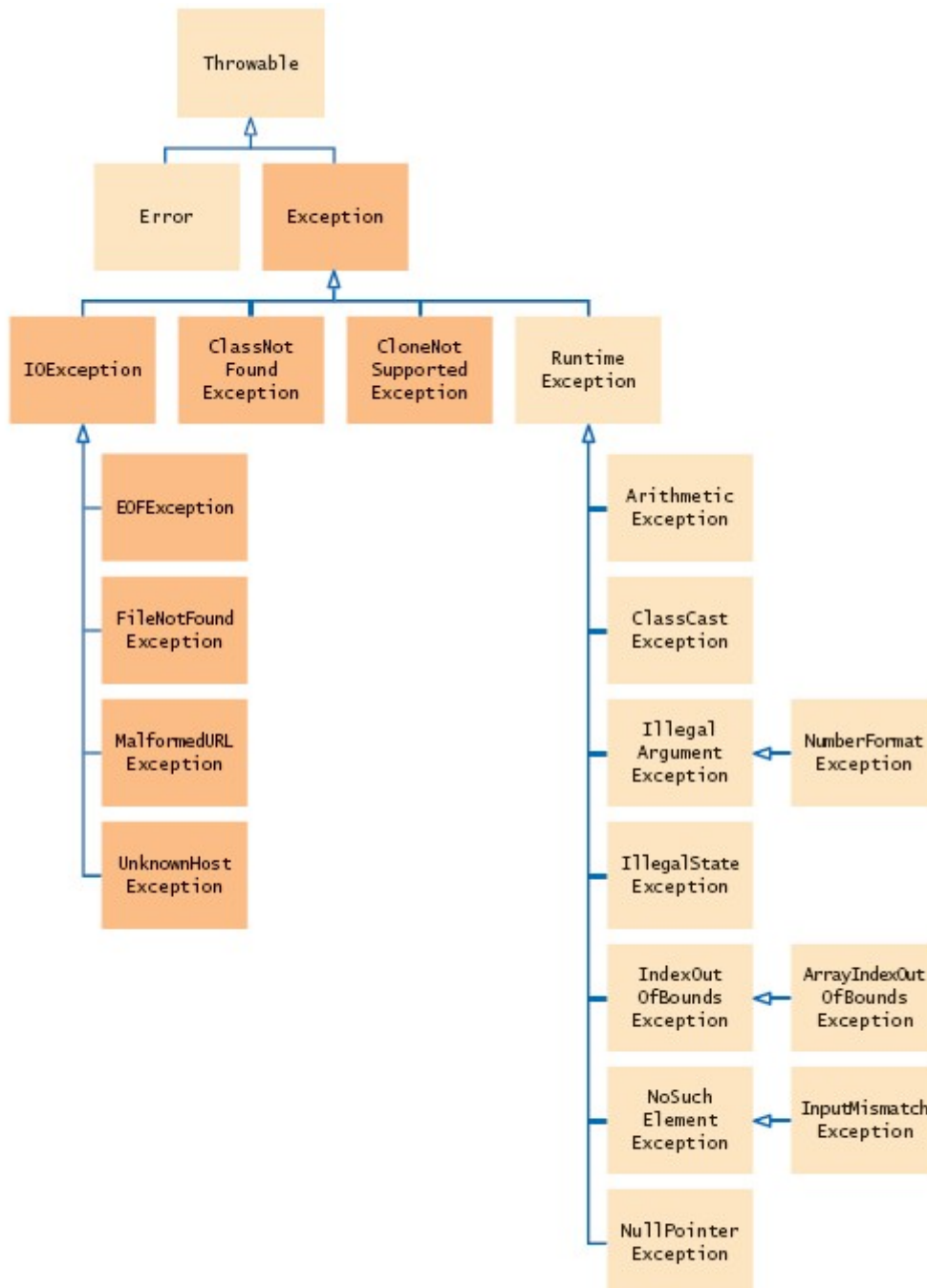


Figure 1 The Hierarchy of Exception Classes

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

La **clase Error** está relacionada con errores de la máquina virtual de Java y no con errores en el código. Generalmente estos errores no dependen del programador por lo que no nos debemos preocupar por tratarlos, por ejemplo 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**: clase base para las excepciones lanzadas al ejecutar una operación de entrada-salida. Pertenecen al paquete java.io.
- **ClassNotFoundException**: se lanza 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 descriptivo.

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 MVJ 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.

3. 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 (catch)** donde la excepción se **captura** 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(tipoExcepcion1 e){
    //Instrucciones para tratar este tipo de excepciones
}
catch(tipoExcepcion2 e){
    //Instrucciones para tratar este tipo de excepciones
}
//Se pueden escribir tantos bloques catch como sean necesarios
finally{ //bloque opcional
    // estas instrucciones se ejecutarán siempre tanto si se ha producido una excepción como si no.
}
```

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 la puede lanzar el sistema de forma automática o el programador 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.

Un 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.

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.

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 las instrucciones que contiene un bloque finally 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.
- Aunque 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.

Volviendo al ejemplo de lectura de un fichero visto antes:

```
leerFichero() {
```

```
    try{
        abrir el fichero;
        calcular tamaño del fichero;
        reservar memoria para el array;
        leer el fichero y pasarlo al array;
```

Código Básico

```
    }
```

```
    catch(errorApertura) {tratar excepción}
    catch(errorCalculoTamaño) {tratar excepción}
    catch(errorMemoriaInsuficiente) {tratar excepción}
    catch(errorLectura) {tratar excepción}
    catch(errorCierre) {tratar excepción}
```

Tratamiento de errores

```
    finally{
        si el fichero está abierto{
            cerrar el fichero
        }
    }
```

Se ejecuta siempre

```
}
```

Ejemplo de tratamiento de excepciones:

El siguiente programa lee un número entero y lo muestra por pantalla.

Si en la instrucción `sc.nextInt()` no se introduce un valor entero se lanza una excepción **InputMismatchException** que es capturada por el bloque `catch`. En este bloque `catch` 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(); //se limpia el buffer de entrada
```

```
            n = 0;
```

```
            System.out.println("Debe introducir un número entero " + e.toString());
```

```
        }
```

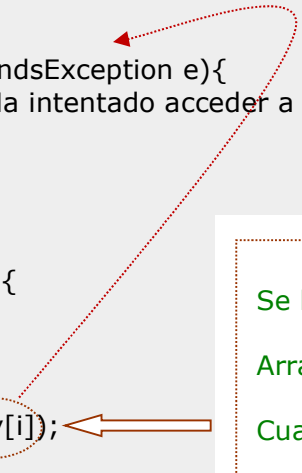
```
    }while(n<=0);
```

```
}
```

nextInt() lanza esta excepción si no se introduce un número entero

Ejemplo: La excepción se lanza en un método y se trata 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

Cuando se intenta acceder al elemento array[3]

La excepción pasa al método main

La salida del programa es:

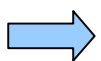
```
4  
2  
6  
Ha intentado acceder a una posición fuera del array
```

En este caso la excepción se ha lanzado en el método `muestraArray()`. Como en este método no hay un bloque `catch` para capturarla y tratarla, la excepción pasa al método desde donde se llamó a `muestraArray()`, en este caso es el método `main`. Aquí sí hay un bloque `catch` para capturarla. Si `main` no hubiese tenido el `catch` adecuado para capturarla el programa mostraría el error por pantalla y terminaría su ejecución.

4. 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() lanza esta excepción si no se introduce un número entero

Intentar acceder a la posición array[n] puede lanzar esta excepción si n no es una posición válida del array

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

Cuando se anidan instrucciones try-catch 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 (Excepcion e) {
            ...
        }
    }
    catch (Excepcion e1) {
        ...
    }
}
```

Ejemplo de secuencias anidadas en el bloque catch:

```
public void metodo() {  
    ...  
    try{  
        ...  
    }  
    catch (Excepcion e) {  
        try{  
            ...  
        }  
        catch (Excepcion 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 entero");  
        }  
        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 entero");  
    }  
}
```

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);
}
```

5. 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 estas excepciones no se tratan (mediante un bloque catch) o no se declaran que son lanzadas de forma explícita.

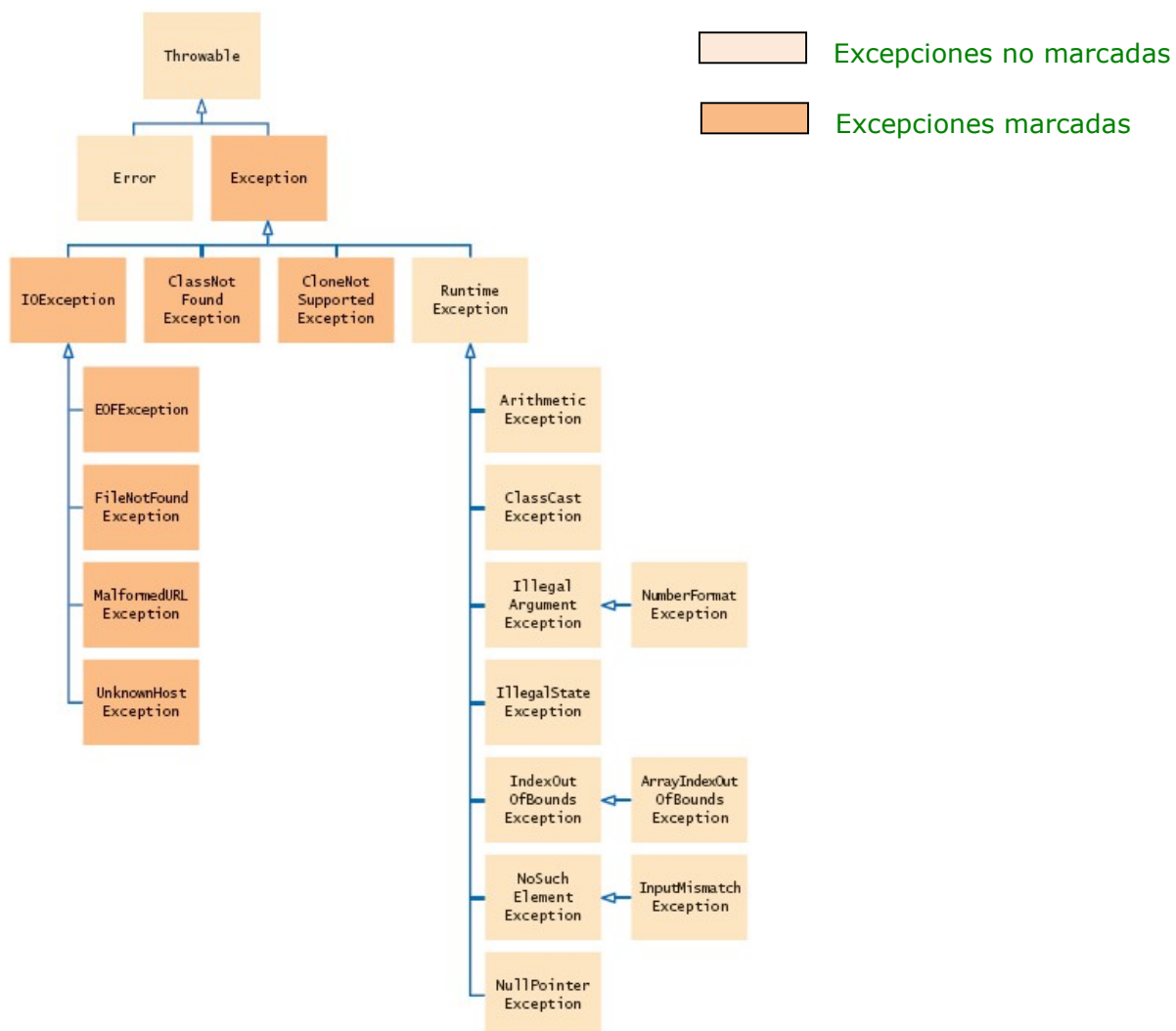


Figure 1 The Hierarchy of Exception Classes

6. 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` si se produce un error en la lectura.

Una excepción del tipo `IOException` es una excepción marcada y la debemos capturar o declarar.

Ejemplo: Lectura de un carácter por teclado utilizando el método `System.in.read()`. El carácter introducido se muestra por pantalla.

En este ejemplo no vamos a capturar la excepción por lo que estamos obligados a declararla.

```
public static void metodo() throws IOException {  
    char car;  
    System.out.println("Introduce un carácter");  
    car = (char)System.in.read();  
    System.out.println("Carácter introducido: " + car);  
}
```

Declaración de la excepción `IOException` que lanza el método `read()`

El método `read()` puede lanzar una `IOException` que al ser una excepción marcada, debemos tratar de alguna forma. En este caso la hemos declarado.

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

Una excepción no capturada en un método se lanza para que la trate algún método de la pila de llamadas. Si no hay ningún método que la trate, el programa mostrará el error y finalizará.

Si capturamos la excepción en el método en lugar de declararla, el método sería este:

```
public static void metodo() {  
    char car;  
    try {  
        System.out.println("Introduce un carácter");  
        car = (char) System.in.read();  
        System.out.println("Carácter introducido: " + car);  
    } catch (IOException ex) {  
        System.out.println("Error de lectura " + ex.toString());  
    }  
}
```

Capturamos la excepción que puede lanzar el método `do read()`

7. LANZAR EXCEPCIONES

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

```
throw objetoExcepcion;
```

La excepción que se lanza es un objeto, por lo que hay que crearla como cualquier otro objeto mediante el operador **new**.

Ejemplos:

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

```
if(n < 0) throw new Exception("Valor debe ser >= 0");
```



Si en un método se utiliza la cláusula **throw** para lanzar una excepción marcada, debe indicarse en la declaración del método con la cláusula **throws** o tener un bloque **catch** para capturarla.

Ejemplo:

```
public class LanzarExcepciones {

    public static void main(String[] args) {
        division();
        raizCuadrada();
    }

    //En este método si el divisor es cero se lanza una excepción
    //del tipo ArithmeticException
    //La excepción se captura en el propio método
    public static void division() {
        boolean repetir;
        do {
            repetir = false;
            try {
                System.out.println("División y resto de números enteros");
                int a = leerEntero("Introduce dividendo: ");
                int b = leerEntero("Introduce divisor: ");
                if (b == 0) {
                    throw new ArithmeticException("No se puede dividir por cero");
                }
                System.out.println("División: " + a / b);
                System.out.println("Resto: " + a % b);
            } catch (ArithmeticException ex) {
                System.out.println(ex.getMessage());
                repetir = true;
            }
        } while (repetir);
    }

    //En este método si el número introducido por teclado es < 0
    //se lanza una excepción del tipo Exception
    //La excepción se captura en el propio método
    public static void raizCuadrada() {
        boolean repetir;
        do {
            repetir = false;
            try {
                System.out.println("Raíz cuadrada de un número entero");
                int a = leerEntero("Introduce número >= 0: ");
                if (a < 0) {
                    throw new Exception("Debe introducir un valor >= 0");
                }
                System.out.println("Raiz cuadrada: " + Math.sqrt(a));
            } catch (Exception ex) {
                System.out.println(ex.getMessage());
                repetir = true;
            }
        } while (repetir);
    }
}
```

```
public static int leerEntero(String s) {
    Scanner sc = new Scanner(System.in);
    int n;
    while (true) {
        try {
            System.out.print(s);
            n = sc.nextInt();
            return n;
        } catch (InputMismatchException e) {
            System.out.println("Valor no válido");
        } finally {
            sc.nextLine();
        }
    }
}
```

El mismo ejemplo, pero en este caso no se capturan las excepciones en los métodos:

```
public class LanzarExcepciones2 {
    public static void main(String[] args) {
        boolean repetir;
        do {
            repetir = false;
            try {
                division();
                raizCuadrada();
                //se captura la excepción lanzada en el método division()
            } catch (ArithmeticException ex) {
                System.out.println(ex.getMessage());
                repetir = true;
                //se captura la excepción lanzada en el método raizCuadrada()
            } catch (Exception ex) {
                System.out.println(ex.getMessage());
                repetir = true;
            }
        } while (repetir);
    }
    //En este método si el divisor es cero se lanza una excepción
    //del tipo ArithmeticException
    //La excepción NO se captura en el propio método
    //Como se trata de una excepción NO MARCADA no es necesario declararla
    public static void division() {
        System.out.println("División y resto de números enteros");
        int a = leerEntero("Introduce dividendo: ");
        int b = leerEntero("Introduce divisor: ");
        if (b == 0) {
            throw new ArithmeticException("No se puede dividir por cero");
        }
        System.out.println("División: " + a / b);
        System.out.println("Resto: " + a % b);
    }
}
```

```
//En este método si el número introducido por teclado es < 0
//se lanza una excepción del tipo Exception
//La excepción NO se captura en el propio método
//Como Exception es una excepción MARCADA hay que declararla
//Las excepciones marcadas debemos capturarlas o declararlas
public static void raizCuadrada() throws Exception {
    System.out.println("Raíz cuadrada de un número entero");
    int a = leerEntero("Introduce número >= 0: ");
    if (a < 0) {
        throw new Exception("Debe introducir un valor >= 0");
    }
    System.out.println("Raíz cuadrada: " + Math.sqrt(a));
}

public static int leerEntero(String s) {
    Scanner sc = new Scanner(System.in);
    int n;
    while (true) {
        try {
            System.out.print(s);
            n = sc.nextInt();
            return n;
        } catch (InputMismatchException e) {
            System.out.println("Valor no válido");
        } finally {
            sc.nextLine();
        }
    }
}
```

En este ejemplo, en el método main podríamos eliminar el `catch (ArithmeticException ex)`



El programa funcionaría de forma correcta ya que el `catch(Exception ex)` capturaría también la `ArithmeticException` lanzada en el método división

8. RELANZAR EXCEPCIONES

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

throw *objetoExcepción*

Esto se suele utilizar en aquellas situaciones en las que además de capturar la excepción dentro del método, queremos comunicar al método que lo ha llamado que se ha producido un error.

Ejemplo:

Programa que modifica el valor de un elemento de un array de enteros.

El programa pide por teclado un número entero, muestra el valor del elemento del array que se encuentra en esa posición, pide el nuevo valor y realiza la modificación.

En el método mostrar, si se produce una excepción, además de capturarla, la relanza al método main.


```
public class RelanzarExcepciones {
    static int[] array = {4, 2, 6, 7, 0, 5};
    public static void main(String[] args) {
        boolean repetir;
        do {
            repetir = false;
            try {
                int posicion = leerEntero("Introduce posición: ");
                mostrar(posicion);
                int nuevoValor = leerEntero("Introduce nuevo valor para esa posición: ");
                array[posicion] = nuevoValor;
                System.out.println("Array modificado");
            } catch (Exception e) { //se captura la excepción relanzada en mostrar
                repetir = true;
            }
        } while (repetir);
    }

    public static void mostrar(int posicion) {
        try {
            System.out.println("Valor actual en posición " + posicion + ": " + array[posicion]);
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println("Ha intentado acceder a una posición fuera del array ");
            throw ex; //se relanza la excepción capturada
        }
    }

    public static int leerEntero(String s) {
        Scanner sc = new Scanner(System.in);
        int n;
        while (true) {
            try {
                System.out.print(s);
                n = sc.nextInt();
                return n;
            } catch (InputMismatchException e) {
                System.out.println("Valor no válido");
            } finally {
                sc.nextLine();
            }
        }
    }
}
```

Si se intenta acceder a un elemento fuera del rango del array se lanza una excepción `ArrayIndexOutOfBoundsException`
La excepción se captura en un catch dentro del método.

9. CREAR EXCEPCIONES PROPIAS

Aunque Java proporciona una gran cantidad de excepciones, podemos crear nuestras propias excepciones.

Normalmente crearemos excepciones propias cuando queramos manejar situaciones no contempladas por la librería estándar de Java.

Por ejemplo, vamos a crear una excepción llamada *ValorNoValido* que se lanzará cuando el valor utilizado en una determinada operación no sea correcto.

Para crear la excepción hay que crear una clase con ese nombre.



Las excepciones propias deben **heredar de la clase Exception**.

La clase tendrá un constructor que reciba el mensaje asignado a la excepción cuando se lanza:

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

Un programa para probar la excepción podría ser este:

```
public static void main(String[] args) {
    try {
        int edad = leerEntero("Introduce edad (> 0 y < 120): ");
        if (edad < 0 || edad > 120) {
            throw new ValorNoValido("La edad debe ser > 0 y < 120");
        }
        System.out.println("Edad introducida: " + edad);
        int nota = leerEntero("Introduce nota (>= 0 y <= 10) : ");
        if (nota < 0 || nota > 10) {
            throw new ValorNoValido("La nota debe ser >= 0 y <= 10");
        }
        System.out.println("Nota introducida: " + nota);
    } catch (ValorNoValido e) {
        System.out.println(e.getMessage());
    }
}

public static int leerEntero(String s) {
    Scanner sc = new Scanner(System.in);
    int n;
    while (true) {
        try {
            System.out.print(s);
            n = sc.nextInt();
            return n;
        } catch (InputMismatchException e) {
            System.out.println("Valor no válido");
        } finally {
            sc.nextLine();
        }
    }
}
```