Gestión de errores en Java

¿Qué son las excepciones?

Las excepciones son el método que tiene Java y la mayoría de los lenguajes de programación orientada a objetos de gestionar los errores que se producen en los programas. La filosofía básica de las excepciones es:

- Dar un aviso de que se puede producir un error (de red, de lectura de archivos, etc.)
- Transferir la gestión de los errores a fragmentos de código específicos destinados a ello (**gestor de excepción**).

Una excepción es una condición anormal que surge en una secuencia de código durante la ejecución de un programa. Cuando se produce una condición excepcional (error), se crea un objeto que representa la excepción y se la envía al método que la ha provocado.

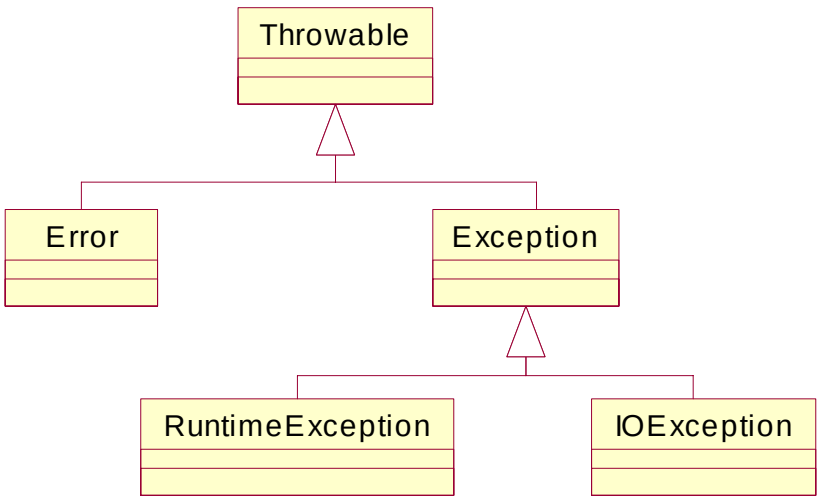
Las excepciones reflejan situaciones más o menos incontrolables como errores de los usuarios o de los dispositivos, o limitaciones físicas. También se pueden producir por errores en el código (acceso a componentes inexistentes de un array), pero estos errores se pueden evitar y conviene no usar las excepciones para gestionarlos.

Razones por las que se deben gestionar las excepciones:

1. Proporcionan un tratamiento de errores, **sin oscurecer el código**, aumentando la robustez de un programa
2. Al separar el código de gestión de errores (cláusulas *catch*) del flujo normal del programa, aumenta la **legibilidad, mantenimiento y reutilización** del código.

Tipos de excepciones

En Java, una excepción es un ejemplar de la clase Throwable. La subjerarquía (simplificada) de Throwable tiene el siguiente aspecto:



- class java.lang.**Throwable** (implements java.io.Serializable)
  - class java.lang.**Error**
    - class java.lang.**AssertionError**
    - class java.lang.**LinkageError**
      - class java.lang.**ClassCircularityError**
      - class java.lang.**ClassFormatError**
        - class java.lang.**UnsupportedClassVersionError**
      - class java.lang.**ExceptionInInitializerError**
      - class java.lang.**IncompatibleClassChangeError**
        - class java.lang.**AbstractMethodError**
        - class java.lang.**IllegalAccessError**
        - class java.lang.**InstantiationError**
        - class java.lang.**NoSuchFieldError**
        - class java.lang.**NoSuchMethodError**
      - class java.lang.**NoClassDefFoundError**
      - class java.lang.**UnsatisfiedLinkError**
      - class java.lang.**VerifyError**
    - class java.lang.**ThreadDeath**
    - class java.lang.**VirtualMachineError**
      - class java.lang.**InternalError**
      - class java.lang.**OutOfMemoryError**
      - class java.lang.**StackOverflowError**
      - class java.lang.**UnknownError**
  - class java.lang.**Exception**
    - class java.lang.**ClassNotFoundException**
    - class java.lang.**CloneNotSupportedException**
    - class java.lang.**IllegalAccessException**
    - class java.lang.**InstantiationException**
    - class java.lang.**InterruptedException**
    - class java.lang.**NoSuchFieldException**
    - class java.lang.**NoSuchMethodException**
    - class java.lang.**RuntimeException**
      - class java.lang.**ArithmeticException**
      - class java.lang.**ArrayStoreException**
      - class java.lang.**ClassCastException**
      - class java.lang.**IllegalArgumentException**
        - class java.lang.**IllegalThreadStateException**
        - class java.lang.**NumberFormatException**
      - class java.lang.**IllegalMonitorStateException**
      - class java.lang.**IllegalStateException**
      - class java.lang.**IndexOutOfBoundsException**
        - class java.lang.**ArrayIndexOutOfBoundsException**
        - class java.lang.**StringIndexOutOfBoundsException**
      - class java.lang.**NegativeArraySizeException**
      - class java.lang.**NullPointerException**
      - class java.lang.**SecurityException**
      - class java.lang.**UnsupportedOperationException**

La subjerarquía **Error** se refiere a los errores internos de la máquina virtual. No se deben usar, y cuando se produce uno de estos errores, no hay mucho que se pueda hacer aparte de salir del programa con un aviso al usuario.

Las excepciones del tipo **RuntimeException** se producen cuando hay un error en el código: una conversión (cast) no válida, un acceso fuera de los límites de un array (ArrayIndexOutOfBounds) o el acceso a un puntero nulo (NullPointerException). La regla general es:

Si se ha producido una RuntimeException, es culpa tuya.

Estas excepciones se pueden evitar con un código correcto, por lo que no se deben lanzar nunca. En cambio, si las lanza la máquina virtual Java, pueden usarse para localizar el error y depurar el código.

Las excepciones que no descienden de la clase RuntimeException incluyen situaciones como el intento de lectura después de un fin de archivo (IOException), el intento de apertura de una URL incorrecta, o el intento de encontrar un objeto de una clase que no existe.

Lanzamiento de excepciones

Los métodos en los que se puede producir un error deben avisar al compilador que éste se puede producir. Para ello utilizan la cláusula "**throws**". Por ejemplo, un método de lectura de un archivo podría lanzar una excepción de tipo IOException:

```
public String leer(FileInputStream archivo) throws IOException {
// ...
```

Se pueden lanzar varias excepciones en el mismo método:

```
public Image cargar(String s)
               throws EOFException, MalformedURLException {
// ...
```

En este ejemplo, se intenta cargar una imagen desde un archivo situado en una URL, y se pueden producir dos errores: que la URL no sea válida, y que el archivo no esté completo (por fallos en la red, por ejemplo).

Las excepciones se lanzan con la cláusula "**throw**". Por ejemplo, cuando estamos implementando un método que efectúa una lectura de un archivo de datos, y se llega inesperadamente a su final, podemos lanzar una EOFException:

```
public String leerDatos(DataInput archivo) throws EOFException {
     // ...
     while ( /* ... */ ) {
          if (ch == -1) { // Fin de archivo
               if (n < longitud)
                    throw new EOFException();
          }
     }
     return s;
}
```

Para lanzar una excepción, sólo hay que elegir su tipo, crear un ejemplar y lanzarla.

El programador puede crear sus propias excepciones, si resulta que ninguna de las predefinidas es adecuada. Para ello, define una clase que desciende de Exception (o de IOException, o de la clase deseada). Se suele agregar un constructor con el mensaje de la excepción, que se inicializa en el constructor llamando al de la clase padre. Además, toda excepción tiene un método getMessage() que devuelve un String con el mensaje.

Por ejemplo, se puede crear una excepción para dar un aviso cuando una pila está vacía. Ésta sería la excepción DesbordamientoInferior, que tendría el mensaje "Desapilar, Error: Pila vacia":

```
public void desapilar( ) throws DesbordamientoInferior
    {
        if( esVacia( ) )
            throw new DesbordamientoInferior( " Desapilar, Error:

                                                  Pila vacia " );
        cimaDePila--;
    }



public class DesbordamientoInferior extends Exception
{
public DesbordamientoInferior( String mensaje )
    {
        super( mensaje );
    }
}
```

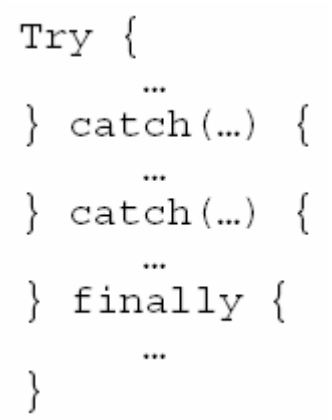
Gestión de excepciones

¿Qué hacer cuando se produce una excepción?

1. Se gestiona la excepción con un bloque try/catch,
2. Se propaga la excepción a la clase que llame al método poniendo la excepción en la cláusula throws del método.

Para capturar una excepción que se ha producido, se usa la cláusula try/catch. La estructura es la siguiente:

```
try {
// código
// ...
}
catch (TipoExcepcion e) {
// Gestor de la excepción
}
```



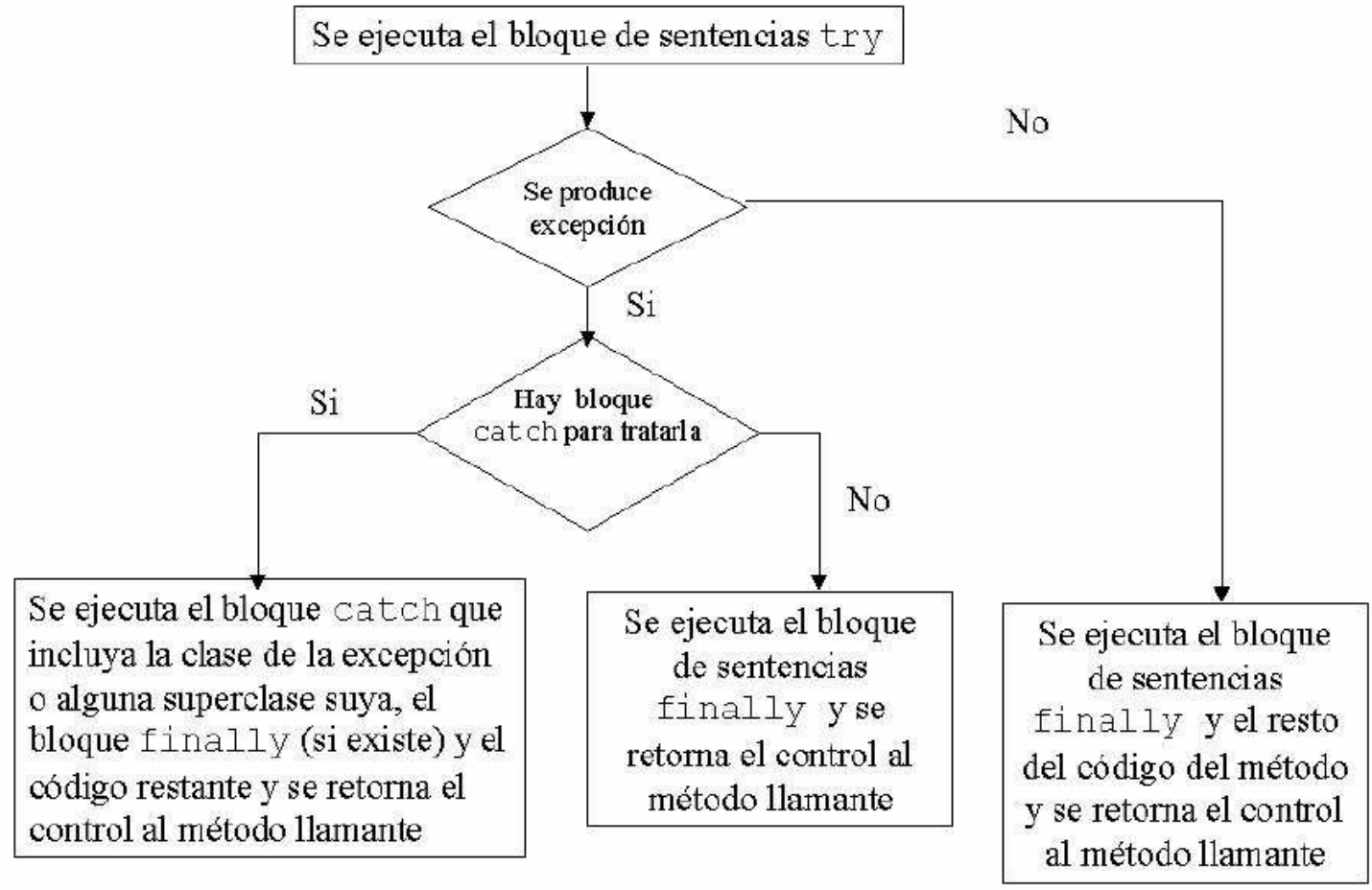
El comportamiento de Java es el siguiente: Si en la ejecución del código dentro del bloque try se produce una excepción de tipo Excepction o descendiente de éste, Java omite la ejecución del resto del código en el bloque try y ejecuta el código situado en el bloque catch.

Si no se desea capturar una excepción explícita, se debe propagar usando la cláusula throws en la cabecera del método. Es decir, si el código de un método contiene o bien un throw o bien una llamada a un método que lance una excepción, entonces:

1. se gestiona la excepción con un bloque try/catch,
2. se propaga la excepción a la clase que llame al método poniéndo la excepción en la cláusula throws del método.

Finalmente, es posible usar una cláusula finally para devolver recursos o acciones similares. El código de una cláusula finally se ejecuta siempre, independientemente de que se produzca excepción o no.

```
try {
// código
// ...
}
catch (TipoExcepcion e) {
// Gestor de la excepción
}
finally {
     // Devolver algún recurso
}
```



Tanto la cláusula ***catch*** como la cláusula ***finally*** son opcionales pero siempre deberá definirse por lo menos una cláusula ***catch*** o ***finally.*** Se puede utilizar más de una clausula ***catch***, pero no más de una ***finally.***

Nota sobre Eclipse: Si se quiere gestionar una excepción con un try-catch

1. Marcar con el ratón en bloque de código que se quiere gestionar.
2. Boton derecho del raton, → Codigo fuente → Rodear con bloque try-catch

```
try {
    entrada=l.leerCadena();
} catch (IOException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}
```

```
Ej: public void ejemExcep(int b, int c) {
    int a = 0;

    try {
        a = b / c;
        System.out.println("OK");
        System.out.println(a + " " + b + " " + c);
    } catch(ArithmeticException e) {
        System.out.println("¡Excepción artimética!");
    } finally {
        if (c != 0) {
           System.out.println("Resultado: " + a);
        } else {
           System.out.println("Resultado: infinito ");
        }
    }
    System.out.println("Código fuera del try ");
}
```

Para la llamada `ejemExcep(4,2)` el resultado sería:

```
> OK
> 2 4 2
> Resultado: 2
> Código fuera del try
```

Para la llamada `ejemExcep(4,0)` el resultado sería:

```
> ¡Excepción aritmética!
> Resultado: infinito
> Código fuera del try
```

```java
// Demostración del mecanismo try-catch-finally para manejar excepciones.
public class UsoExcepciones {
    public static void main(String args[]) {
        try {
            lanzarExcepcion(); // llamar al método lanzarExcepcion
        }
        // atrapar excepciones lanzadas por el método lanzarExcepcion
        catch (Exception excepcion) {
            System.err.println("La excepcion se manejo en main");
        }
        noLanzaExcepcion();
    }
    // demostrar try/catch/finally
    public static void lanzarExcepcion() throws Exception {
        // lanzar una excepción y atraparla inmediatamente
        try {
            System.out.println("El metodo lanzarExcepcion");
            throw new Exception(); // generar excepción
        }
        // atrapar la excepción lanzada en el bloque try
        catch (Exception excepcion) {
            System.err.println("Excepcion manejada en el metodo lanzarExcepcion");
            throw excepcion; // volver a lanzar para procesarla posteriormente
            // cualquier código aquí no llegaría a ejecutarse
        }
        // este bloque se ejecuta, sin importar lo que ocurra en try/catch
        finally {
            System.err.println("Finalmente se ejecuto en lanzarExcepcion");
        }
        // cualquier código aquí no llegaría a ejecutarse
    } // fin del método lanzarExcepcion
    // demostrar finally cuando no ocurre excepción
    public static void noLanzaExcepcion() {
        // el bloque try no lanza una excepción
        try {
            System.out.println("El metodo noLanzaExcepcion");
        }
        // catch no se ejecuta, porque no se lanzó una excepción
        catch (Exception excepcion) {
            System.err.println(excepcion);
        }
        // la cláusula this se ejecuta, sin importar lo que ocurra en try/catch
        finally {
            System.err.println("Finalmente se ejecuto en noLanzaExcepcion");
        }
        System.out.println("Fin del metodo noLanzaExcepcion");
    } // fin del método noLanzaExcepcion
} // fin de la clase UsoExcepciones
```

```
El metodo lanzarExcepcion
El metodo noLanzaExcepcion
Excepcion manejada en el metodo lanzarExcepcion
Finalmente se ejecuto en lanzarExcepcion
La excepcion se manejo en main
Finalmente se ejecuto en noLanzaExcepcionFin del metodo noLanzaExcepcion
```

```java
// Demostración de getMessage y printStackTrace de la clase Exception.
public class UsoExcepciones {

    public static void main(String args[]) {
        try {
            metodo1(); // llamar a metodo1
        }
        // atrapar las excepciones lanzadas desde metodo1
        catch (Exception excepcion) {
            System.err.println(excepcion.getMessage() + "\n");
            excepcion.printStackTrace();

            // obtener la información de rastreo de la pila
            StackTraceElement[] elementosRastreo = excepcion.getStackTrace();

            System.out
                    .println("\nRastreo de pila proveniente de getStackTrace:");
            System.out.println("Clase\t\tArchivo\t\t\tLinea\tMetodo");

            // iterar a través de elementosRastreo para obtener descripción de
            // las excepciones
            for (int i = 0; i < elementosRastreo.length; i++) {
                StackTraceElement elementoActual = elementosRastreo[i];
                System.out.print(elementoActual.getClassName() + "\t");
                System.out.print(elementoActual.getFileName() + "\t");
                System.out.print(elementoActual.getLineNumber() + "\t");
                System.out.print(elementoActual.getMethodName() + "\n");

            } // fin de la instrucción for
        } // fin de la instrucción catch
    } // fin del método main
    // llamar a metodo2; lanzar excepciones de vuelta hacia main
    public static void metodo1() throws Exception {
        metodo2();
    }
    // llamar al método3; lanzar excepciones de vuelta hacia metodo1
    public static void metodo2() throws Exception {
        metodo3();
    }
    // lanzar excepción de vuelta a metodo2
    public static void metodo3() throws Exception {
        throw new Exception("La excepcion se lanzo en metodo3");
    }

} // fin de la clase UsoExcepciones
```

```
La excepcion se lanzo en metodo3

java.lang.Exception: La excepcion se lanzo en metodo3
        at Java5Ejemplos.Cap15.fig15_05.UsoExcepciones.metodo3(UsoExcepciones.java:51)
        at Java5Ejemplos.Cap15.fig15_05.UsoExcepciones.metodo2(UsoExcepciones.java:46)
        at Java5Ejemplos.Cap15.fig15_05.UsoExcepciones.metodo1(UsoExcepciones.java:41)
        at Java5Ejemplos.Cap15.fig15_05.UsoExcepciones.main(UsoExcepciones.java:9)

Rastreo de pila proveniente de getStackTrace:
Clase           Archivo                 Linea   Metodo
Java5Ejemplos.Cap15.fig15_05.UsoExcepciones     UsoExcepciones.java     51      metodo3
Java5Ejemplos.Cap15.fig15_05.UsoExcepciones     UsoExcepciones.java     46      metodo2
Java5Ejemplos.Cap15.fig15_05.UsoExcepciones     UsoExcepciones.java     41      metodo1
Java5Ejemplos.Cap15.fig15_05.UsoExcepciones     UsoExcepciones.java     9       main
```

```java
// Demostración de las excepciones encadenadas.
public class UsoExcepcionesEncadenadas {

    public static void main(String args[]) {
        try {
            metodo1(); // llamar a metodo1
        }

        // atrapar las excepciones lanzadas desde metodo1
        catch (Exception excepcion) {
            excepcion.printStackTrace();
        }
    }

    // llamar al metodo2; lanzar excepciones de vuelta a main
    public static void metodo1() throws Exception {
        try {
            metodo2(); // llamar a metodo2
        }

        // atrapar la excepción lanzada desde metodo2
        catch (Exception excepcion) {
            throw new Exception("Excepcion lanzada en metodo1", excepcion);
        }
    }

    // llamar a metodo3; lanzar excepciones de vuelta a metodo1
    public static void metodo2() throws Exception {
        try {
            metodo3(); // llamar a metodo3
        }

        // atrapar la excepción lanzada desde metodo3
        catch (Exception excepcion) {
            throw new Exception("Excepcion lanzada en metodo2", excepcion);
        }
    }

    // lanzar excepción de vuelta a metodo2
    public static void metodo3() throws Exception {
        throw new Exception("Excepcion lanzada en metodo3");
    }

} // fin de la clase UsoExcepcionesEncadenadas
```

```
java.lang.Exception: Excepcion lanzada en metodo1
        at Java5Ejemplos.Cap15.fig15_06.UsoExcepcionesEncadenadas.metodo1(UsoExcepcionesEncadenadas.java:26)
        at Java5Ejemplos.Cap15.fig15_06.UsoExcepcionesEncadenadas.main(UsoExcepcionesEncadenadas.java:9)
Caused by: java.lang.Exception: Excepcion lanzada en metodo2
        at Java5Ejemplos.Cap15.fig15_06.UsoExcepcionesEncadenadas.metodo2(UsoExcepcionesEncadenadas.java:38)
        at Java5Ejemplos.Cap15.fig15_06.UsoExcepcionesEncadenadas.metodo1(UsoExcepcionesEncadenadas.java:21)
        ... 1 more
Caused by: java.lang.Exception: Excepcion lanzada en metodo3
        at Java5Ejemplos.Cap15.fig15_06.UsoExcepcionesEncadenadas.metodo3(UsoExcepcionesEncadenadas.java:44)
        at Java5Ejemplos.Cap15.fig15_06.UsoExcepcionesEncadenadas.metodo2(UsoExcepcionesEncadenadas.java:33)
        ... 2 more
```