5 Excepciones

Contenidos

1.- Introducción

Cuando un programa está en ejecución se pueden producir **situaciones inesperadas** que, de no tratarse de forma adecuada, producirán la **terminación anticipada y no deseada** del programa. Por ejemplo:

- El programa lee un número con nextInt() y el usuario introduce letras.
- El programa lee un número con nextInt() y el usuario introduce un valor con decimales.
- El programa realiza una división entre cero (división entera).
- Se intenta abrir un fichero que no existe.
- En un String, se intenta obtener el carácter correspondiente a una posición que no existe.
- El ordenador no tiene suficiente memoria para realizar determinada operación.

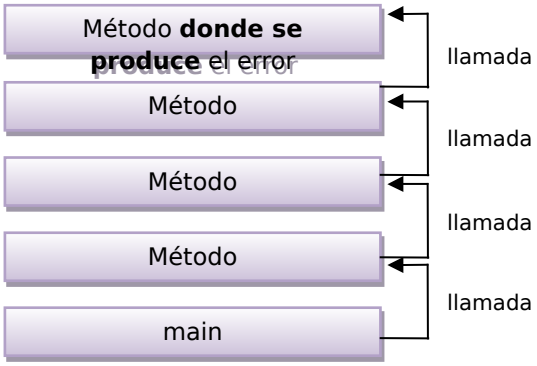
Una **excepción** es un evento que ocurre durante la ejecución de un programa e interrumpe el flujo normal de ejecución de sus instrucciones.

2.- Funcionamiento

Cuando se produce un error dentro de un método, el método crea un objeto y se lo pasa al sistema de tiempo de ejecución. El objeto, al que llamamos ***excepción***, contiene información sobre el error. Crear un objeto de excepción y *entregárselo* al sistema de ejecución es lo que llamamos *"lanzar una excepción"*. Los programas lanzan excepciones cuando se producen situaciones inesperadas como las descritas anteriormente.

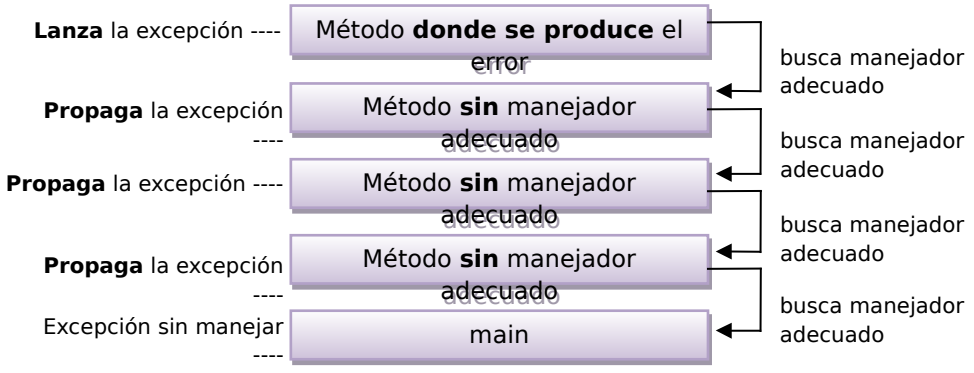
Cuando un método lanza una excepción, el sistema de ejecución intenta encontrar algo para manejarla. El conjunto de posibles "algo" para manejar

la excepción es la lista ordenada de los métodos que habían sido llamados hasta llegar al método donde se produjo el error. La lista de los métodos que se conoce como la ***pila de llamadas***.

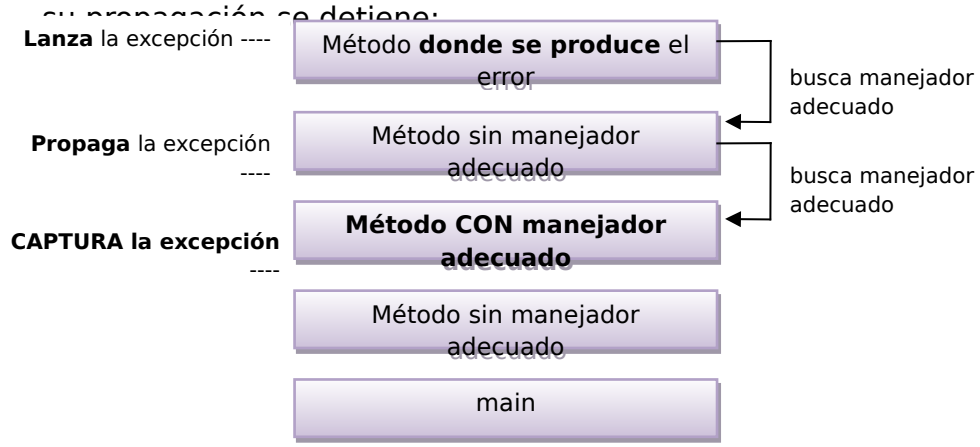


El sistema de ejecución busca en la pila de llamadas algún método que contenga un bloque de instrucciones para manejar la excepción. A este bloque de instrucciones se le llama ***manejador de excepciones***. La búsqueda comienza con el método en el que se produjo el error y procede a través de la pila de llamadas en el orden inverso en el que los métodos se llaman. Cuando se encuentra un manejador adecuado, el sistema pasa la excepción al manejador. Un manejador de excepciones se considera adecuado si el tipo del objeto de la excepción es compatible con el tipo que puede ser manejado por el manejador.

Si el sistema de tiempo de ejecución no encuentra un manejador de excepciones apropiado, como se muestra en la siguiente figura, el sistema de tiempo de ejecución (y, en consecuencia, el programa) termina de forma abrupta.

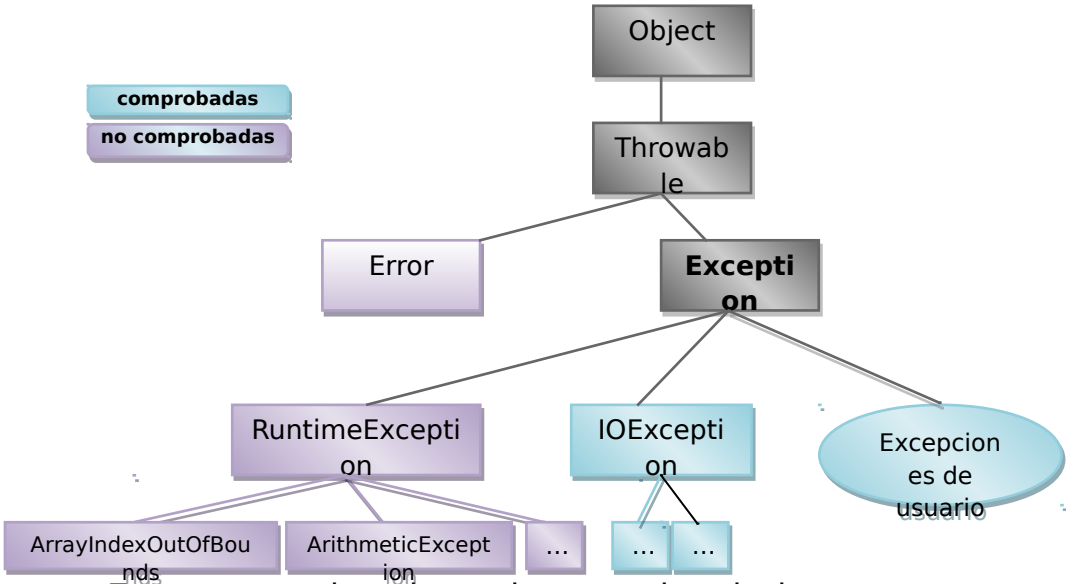


En cambio, si alguno de los métodos tiene un m*anejador* de excepción compatible con la excepción que se ha producido, la excepción se captura y su propagación se detiene:



3.- Jerarquía de excepciones.

Una **situación inesperada** se representa en Java como un **objeto** de la clase ***Throwable***. Para representar distintos tipos de situaciones inesperadas se utilizan distintas subclases de la clase *Throwable*:



- ***Error*** para situaciones inesperadas de las que no es posible recuperarse como por ejemplo que no exista memoria suficiente para continuar la ejecución.
- ***Exception*** para situaciones ante las que el programa tiene capacidad de respuesta:
  - ***RunTimeException***: Errores de programación, como una división por cero, el acceso a una posición de un array inexistente, etc.
  - ***IOException***: Errores de entrada salida, como el intento de escribir en un archivo protegido contra escritura.
  - **Excepciones de usuario**: Son clases derivadas de Exception creadas por el programador para modelizar situaciones de error que se producen en los programas que realiza.

4.- Gestión de excepciones.

Al programar hay que prever los errores, de manera que se evite la terminación abrupta del programa, se avise de la situación de error y, si es posible, se reconduzca el programa hacia una situación en la que no se produzca dicho error.

Hay distintas formas de actuar cuando se prevé que una instrucción o conjunto de instrucciones puede provocar un error (una excepción):

- **Evitar** que la excepción se produzca: Introducir las estructuras de programación necesarias, como una sentencia alternativa (if o switch) de manera que la excepción no pueda producirse.

- Dejar que la excepción se produzca y manejarla: **capturarla.**

- Dejar que la excepción se produzca y **propagarla**, dejando el manejo de la excepción a otro método.

4.1.- Evitar la excepción.

En ocasiones es posible adelantarse a la posibilidad de que la excepción se produzca, y evitar que suceda.

Por ejemplo, si hacemos la división de dos enteros, podemos asegurarnos de que el divisor es distinto de cero.

Observa la diferencia entre los dos siguientes fragmentos de código:

En el primero la instrucción (a/b) puede provocar una excepción si b es cero.

```
Scanner tec = new Scanner(System.in);

System.out.println("Introduzca un dividendo");
int a = tec.nextInt();
System.out.println("Introduzca un divisor (distinto de cero)");
int b = tec.nextInt();

System.out.println("Cociente: " + (a/b));
```

En este segundo ejemplo, en cambio, la excepción no se puede producir, puesto que se ha protegido con una sentencia *if.*

```
Scanner tec = new Scanner(System.in);

System.out.println("Introduzca un dividendo");
int a = tec.nextInt();
System.out.println("Introduzca un divisor (distinto de cero)");
int b = tec.nextInt();

if(b == 0) System.out.println("El divisor no puede ser cero");
else System.out.println("Cociente: " + (a/b));
```

4.2.- Capturar la excepción.

En la mayoría de ocasiones no es posible evitar que la excepción se produzca. Por ejemplo, si leemos un entero desde teclado con *nextInt()* y el usuario no introduce un número entero, se producirá la una excepción sin que podamos hacer nada para evitarla, ya que no tenemos el control de lo que teclea el usuario.

Lo que se puede hacer con una excepción que no podemos evitar es manejarla, capturándola. Para capturar una excepción, se encierran las instrucciones que pueden provocarla en un bloque **try-catch-finally.** El objetivo es que, cuando se produce una excepción, se ejecute un conjunto de instrucciones destinado a tratar el error que te tenido lugar.

```
01      Scanner tec = new Scanner(System.in);
02      try{
04              int num = tec.nextInt();
05              System.out.format("El doble de %d es %d", num, num*2);
06      } catch (InputMismatchException e){
07              System.err.println("Debía introducir un entero");
08      }
09      System.out.println("-- Fin del programa --");
```

Si la instrucción *nextInt()* (línea 04), provoca la excepción ***InputMismatchException*** (porque el usuario no introduce un número entero), la ejecución saltará inmediatamente al comienzo del bloque catch (línea 07), que se encarga de manejar el error. En este caso, muestra un mensaje por pantalla informando del hecho.

La instrucción de la línea 05 nunca se ejecutará si se produce excepción. Tanto si se produce la excepción como si no, el programa finalizará ejecutando la instrucción de la línea 09

Sintaxis de try - catch – finally

```
try {

        // Instrucciones que pueden provocar alguna excepción

} catch (TipoExcepcion1 nombreVble) {

        // instrucciones a ejecutar  si  se produce excepción de
        // tipo TipoExcepcion1 (o de un tipo compatible)

} catch (TipoExcepcion2 nombreVble){

        // instrucciones a ejecutar  si  se produce excepción de
        // tipo TipoExcepcion2 (o de un tipo compatible)

}

... //Tantos bloques catch como queramos

finally { //Opcional

        // instrucciones a ejecutar tanto si se producen
        // excepciones como si no

}
```

Se pueden poner tantos bloques *catch* como se desee.

El bloque *finally* es opcional. Puede o no aparecer.

¿Cómo funciona try-catch-finally?

- Se ejecuta el código del bloque **try** hasta que se produce una excepción o hasta que termina con éxito.
- Si no se produce excepción, se ejecutan las instrucciones del bloque finally y el programa continúa normalmente tras el bloque try-catch-finally
- Si se produce una excepción en el bloque **try**, se examinan las distintas cláusulas **catch**, por orden de aparición, comprobando si el tipo de la excepción lanzada corresponde con el tipo de la declarada en la cláusula catch (o con un tipo compatible).
    - Si se encuentra un bloque catch adecuado, se ejecuta el código asociado y no se comprueba ninguna otra cláusula **catch**. Después de ejecutarse el bloque catch el programa continúa con las instrucciones del bloque finally y el programa continua normalmente tras el bloque try-catch-finally.
    - Si no hay ningún catch que capture la excepción que se ha producido, la excepción se propaga al siguiente método de la pila de llamadas. Previemente se ejecutan las instrucciones del bloque finally, pero las instrucciones que haya a continuación del bloque try-catch-finally no se ejecutarán.

□ Observaciones:

- Cuando hay varios bloques catch, el orden en que aparecen es importante. Se ejecutará el primero de ellos que capture una excepción compatible con la que se ha producido.
- Solo se ejecutará un bloque catch.
- Tanto si se ha producido una excepción como si no, tanto si ésta se ha capturado como si no, se ejecutan las instrucciones asociadas a la cláusula **finally**. En la cláusula finally pondremos instrucciones que, de no existir, tendríamos que repetir en el bloque try y en cada uno de los bloques catch.

4.3.- Propagar la excepción.

Cuando en un método se produce una excepción, tenemos la posibilidad de propagar la excepción en lugar de capturarla. Propagamos una excepción cuando queremos que sea otro método de la pila de llamadas el que se encargue de su manejo.

Para indicar que un método propaga una excepción, se añade a su cabecera ***throws NombredelaExcepcion***.

Observa el siguiente ejemplo: En él, el método mostrarOperaciones no captura la ArithmeticException, sino que la propaga y deja que la capture el método main.

```java
public static void main(String[] args) {
    Scanner tec = new Scanner(System.in);
    System.out.println("Introduce dos enteros distintos de cero: ");

    try{
        int num1 = tec.nextInt();
        int num2 = tec.nextInt();
        mostrarOperaciones(num1,num2);
    } catch (InputMismatchException e){
        System.err.println("Debía introducir un numero real");
    } catch (ArithmeticException e){
        System.err.println("Los números han de ser distintos de cero");
    }
    System.out.println("-- Fin del programa --");
}

public static void mostrarOperaciones(int a, int b) throws ArithmeticException{
    System.out.format ("%d + %d = %d%n",a, b, a+b);
    System.out.format ("%d - %d = %d%n",a, b, a-b);
    System.out.format ("%d * %d = %d%n",a, b, a*b);
    System.out.format ("%d / %d = %d%n",a, b, a/b);
}
```

5.- Excepciones comprobadas y no comprobadas

Existen dos grupos de excepciones: comprobadas y no comprobadas (Observa la figura del apartado “Jerarquía de excepciones”

- Excepciones **comprobadas**, verificadas (o checked):
  - Su tratamiento es obligatorio y el compilador comprueba que se haga. Es necesario capturarlas (con try-catch) o propagarlas (con throws), de lo contrario se produce error de compilación.
  - Son excepciones que un programa bien escrito debería prever, tratar y recuperarse de ellas.
  - Supongamos por ejemplo que nuestro programa va a leer y mostrar por pantalla el contenido de un fichero cuyo nombre indica el usuario. En la mayoría de ocasiones el usuario indicará el nombre de un fichero existente y válido y el programa lo mostrará, pero es posible que en alguna ocasión el usuario se equivoque e indique el nombre de un fichero que no existe. En tal caso se producirá una excepción FileNotFoundException. El programa debería ser capaz de manejar la situación, informar al usuario y permitirle, si se estima oportuno, que introduzca un nombre de fichero válido.
  - Son comprobadas las derivadas de java.lang.IOException y las excepciones de usuario (que trataremos más adelante)
- Excepciones **no comprobadas**, no verificadas (o unchecked):
  - Su tratamiento no es obligatorio y el compilador no obliga a que se utilice un bloque try-catch o a que se anuncie su propagación usando *throws.* Aunque no es obligatorio, puede hacerse si se estima conveniente.
  - Son excepciones que suelen producirse porque nuestro programa contiene algún error. Es por eso que no se comprueban, pues el objetivo no es recuperarse de ellas, sino avisarnos de que estamos tratando de realizar alguna operación no posible. La solución no pasa por capturarlas, sino por modificar el programa.

- Siguiendo con el ejemplo anterior, supongamos que el usuario introduce un nombre de fichero correcto pero, por algún error en nuestro programa, al método encargado de leer el fichero en lugar de llegarle el nombre le llega el valor *null.* En ese caso se producirá un *NullPointerException*. No tiene demasiado sentido capturar la excepción, puesto que se produce porque el programa contiene algún error que hay que subsanar.
- Son no comprobadas las clases derivadas de java.lang.Error y de java.lang.RuntimeException

6.- Métodos útiles de la clase Exception

- public void printStackTrace();
  - Imprime la pila de llamadas por la salida estándar de errores. Incluye los números de línea y ficheros donde se ha producido la excepción.
- public String getMessage();
  - Devuelve una cadena de caracteres con la descripción de la excepción.
- public String toString();
  - Devuelve una cadena de caracteres con información de la excepción.

7.- Crear y lanzar excepciones de usuario:

Las excepciones de usuario son subclases de la clase Exception que podemos crear y lanzar en nuestros programas para avisar sobre determinadas situaciones.

7.1.- Crear una nueva excepción

Para crear una nueva excepción tenemos que crear una clase derivada (subclase) de la clase Exception.

La clase Exception tiene dos constructores, uno sin parámetros y otro que acepta un String con un texto descriptivo de la excepción. Todas las excepciones de usuario las crearemos de la siguiente forma:

```
class NombreExcepcion extends Exception {

  public NombreExcepcion(){ super();}

  public NombreExcepcion(String msg) {super(msg);}

}
```

7.2.- Lanzar una excepción

Las excepciones se lanzan mediante la instrucción throw. La sintaxis es

throw *new NombreExcepcion(Mensaje descriptivo de la situación inesperada);*

Ya que se tratará de una excepción comprobada, en la cabecera del método que lanza la excepción habrá que propagarla.