



USO DE EXCEPCIONES EN JAVA

Al ejecutar programas se producen errores. Los errores que se producen en tiempo de ejecución (cuando se está ejecutando el programa) se denominan **excepciones**. Estos errores pueden producirse en alguna de las instrucciones de nuestro programa, como por ejemplo cuando se hace una división por cero, cuando un objeto es 'null' y no puede serlo, cuando no se abre correctamente un fichero, cuando se pretende ejecutar un método inexistente, etc. Cuando se produce una excepción se muestra en la pantalla un mensaje de error y finaliza la ejecución del programa.

Cuando se produce un error en un método, “se lanza” un objeto Throwable. Cualquier método que haya llamado al método puede “capturar la excepción” y tomar las medidas que estime oportunas. Tras capturar la excepción, el control no vuelve al método en el que se produjo la excepción, sino que la ejecución del programa continúa en el punto donde se haya capturado la excepción. Para trabajar con excepciones en nuestros programas debemos identificar:

- el **bloque protegido** que es el trozo de código que interesa proteger para poder reaccionar ante la generación de errores
- el **tipo de excepción** que varía según el lenguaje de programación en el que se trabaje, en este caso Java
- un **manejador de excepción** que es el código que se ocupará de tratar el problema

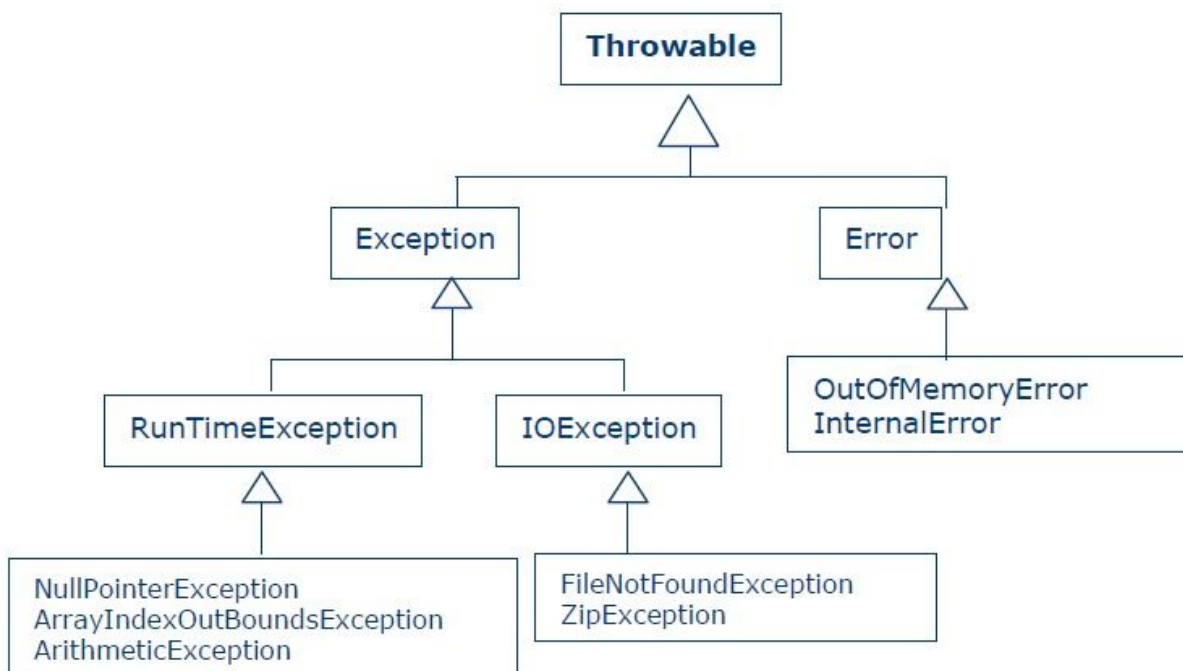
Programación Concurrente - Apunte de Excepciones en Java

Algunas Excepciones pueden presentarse cuando:

- El usuario escribe una palabra cuando se esperaba un número.
- En el programa se intenta leer un archivo que no existe.
- En el programa no se puede establecer una conexión de red.
- En el programa se intenta realizar una división por cero.
- Cuando se intenta calcular la raíz cuadrada de un número negativo.
- Cuando se excede del número de elementos de un arreglo.

En Java ...

Java presenta la siguiente jerarquía de clases para el manejo de excepciones:



Programación Concurrente - Apunte de Excepciones en Java

A continuación explicaremos brevemente cada una de las clases:

THROWABLE:

Es la clase base que representa todo lo que se puede “lanzar” en JAVA.

- Contiene una instantánea del estado de la pila en el momento en el que se creó el objeto (“stack trace” o “call chain”).
- Almacena un mensaje (variable de instancia de tipo String) que podemos utilizar para detallar qué error se produjo.
- Puede tener una causa, también de tipo Throwable, que permite representar el error que causó este error.

Uno de sus métodos más definidos en la clase Throwable y heredado por todas sus subclases es el método `getMessage`. Este método permite obtener información de una excepción. El siguiente ejemplo ilustra su uso:

```
try {  
    C c = new C();  
    c.m1();  
} catch (FileNotFoundException fnfe) {  
    System.out.println(fnfe.getMessage());  
}
```

En el bloque `catch` se capturan las excepciones de tipo `FileNotFoundException`, que se producen cuando se intenta abrir un archivo que no existe. En caso de que se produzca una excepción de este tipo, en la salida estándar del programa se escribirá algo como: `noexiste.dat (No such file or directory)` (suponiendo que el archivo que se intenta abrir se llame `noexiste.dat`).

Como se puede observar, el método `getMessage` permite obtener información útil sobre la excepción. El usuario puede utilizar esta información para intentar solucionar el error. Por ejemplo, en este caso, se muestra el nombre del archivo que se intenta abrir. Esta



Programación Concurrente - Apunte de Excepciones en Java

indicación podría servir para que el usuario recuerde que es necesario que el archivo de datos utilizado por el programa deba tener un nombre concreto, deba estar ubicado en algún directorio concreto, etc.

ERROR:

Subclase de Throwable que indica problemas graves que una aplicación no debería intentar solucionar. Ejemplos: Memoria agotada, error interno de JVM....

EXCEPTION:

Exception y sus subclases indican situaciones que una aplicación debería tratar de forma razonable. Los dos tipos principales de excepciones son:

- RuntimeException (errores del programador, como una división por cero o el acceso fuera de los límites de un array).
- IOException (errores que no puede evitar el programador, generalmente relacionados con la E/S del programa).

La mayoría de las clases derivadas de la clase Exception no implementan métodos adicionales ni proporcionan más funcionalidad, simplemente heredan de Exception. Por ello, la información más importante sobre una excepción se encuentra en la jerarquía de excepciones, el nombre de la excepción y la información que contiene la excepción.

Captura de Excepciones: try....catch

Se utilizan en Java para capturar las excepciones que se hayan podido producir en el bloque de código delimitado por **try** y **catch** (que es el bloque protegido).

En cuanto se produce la excepción, la ejecución del bloque try termina.

La cláusula catch recibe como argumento un objeto Throwable.

Programación Concurrente - Apunte de Excepciones en Java

Ejemplo 1:

- bloque protegido: Bloque 2
- tipo de excepcion: Exception
- bloque manejador de la excepcion: Bloque 3

```
//Bloque1
try{
    //Bloque 2
}catch (Exception error){
    //Bloque 3
}
//Bloque 4
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 4.
- Con una excepción en el Bloque 2: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 4.
- Con una excepción en el Bloque 1: se ejecutará Bloque 1*.

Ejemplo 2:

Pueden considerarse más de un manejador, para contemplar distinto tipo de excepciones, como se muestra a continuación

```
//Bloque1
try{
    //Bloque 2
}catch (ArithmeticException ae){
    //Bloque 3
}catch (NullPointerException ne){
    //Bloque 4
}
//Bloque 5
```

Programación Concurrente - Apunte de Excepciones en Java

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 5.
- Acceso a un objeto nulo (null): se ejecutará Bloque 1 – Bloque 2* - Bloque 4 – Bloque 5.
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2*.

Ejemplo 3:

```
//Bloque1
try{
    //Bloque 2
}catch (ArithmeticException ae){
    //Bloque 3
}catch (Exception error){
    //Bloque 4
}
//Bloque 5
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 5.
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2* - Bloque 4 – Bloque 5.

Programación Concurrente - Apunte de Excepciones en Java

Es importante tener en cuenta que las cláusulas se comprueban en orden, por lo que los tipos de excepción más específicos deben considerarse antes que los tipos más generales. En el ejemplo anterior el “*catch (Exception error)*” capturará las excepciones que no sean “*ArithmeticException*”

En el caso que sigue, ¿qué sucederá?

```
//Bloque1
try{
    //Bloque 2
}catch (Exception error){
    //Bloque 3
}catch (ArithmeticException ae){
    //Bloque 4
}
//Bloque 5
```

La cláusula finally

En ocasiones, nos interesa ejecutar un fragmento de código independientemente de si se produce o no una excepción (por ejemplo, cerrar un archivo en el que estemos trabajando).

Programación Concurrente - Apunte de Excepciones en Java

Ejemplo:

```
//Bloque1
try{
    //Bloque 2
}catch (ArithmeticException ae){
    //Bloque 3
}
finally{
    //Bloque 4
}
//Bloque 5
```

La ejecución puede ser:

- Sin excepción: se ejecutará Bloque1 - Bloque 2 - Bloque 4 – Bloque 5.
- Excepción de tipo aritmético: se ejecutará Bloque1 – Bloque 2* - Bloque 3 - Bloque 4 – Bloque 5
- Excepción de otro tipo diferente: se ejecutará Bloque 1 – Bloque 2* – Bloque 4.

Si el cuerpo del bloque try llega a comenzar su ejecución, el bloque finally siempre se ejecutará...

- Después del bloque try si no se producen excepciones.
- Después del bloque catch si este captura la excepción.
- Justo después de que se produzca la excepción si ninguna cláusula catch captura la excepción y antes de que la excepción se propague hacia afuera, en la cadena de llamadas.

Programación Concurrente - Apunte de Excepciones en Java

La sentencia throw

Se utiliza para lanzar objetos de tipo Throwable en Java:

```
throw new Exception ("Mensaje de error.....");
```

Cuando se lanza una excepción:

- Se sale inmediatamente del bloque de código actual.
- Si el bloque tiene asociada una cláusula catch adecuada para el tipo de la excepción generada se ejecuta el cuerpo de la cláusula catch.
- Si no, se sale inmediatamente del bloque (o método) dentro del cual está el bloque en el que se produjo la excepción y se busca una cláusula catch apropiada.
- El proceso continúa hasta llegar al método main de la aplicación. Si ahí tampoco existe una cláusula catch adecuada, la máquina virtual Java finaliza su ejecución con un mensaje de error.

Propagación de excepciones (throws)

Si en el cuerpo de un método se lanza una excepción (de un tipo derivado de la clase Exception), en la cabecera del método hay que añadir una cláusula throws que incluye una lista de los tipos de excepciones que se pueden producir al invocar el método.

Ejemplo:

```
public String leerArchivo (String nombreArchivo)  
  
    throws IOException  
  
...// continua el código
```

Las excepciones de tipo RuntimeException (que son muy comunes) no es necesario declararlas en la cláusula throws.

Programación Concurrente - Apunte de Excepciones en Java

Al implementar un método, hay que decidir si las excepciones se propagaran hacia afuera (throws) o se capturarán en el propio método (catch).

1. Método que propaga una excepción:

```
public void f() throws IOException{  
  
    //Fragmento de código que puede lanzar una excepción  
    de tipo IOException.  
  
}
```

Un método puede lanzar una excepción porque cree explícitamente un objeto Throwable y lo lance con throw, o bien porque llame a un método que genere la excepción y no la capture.

2. Método equivalente que no propaga la excepción:

```
public void f(){  
  
    //Fragmento de código libre de excepciones  
  
    try{  
  
        //Fragmento de código que puede lanzar una excepción  
        de tipo IOException }catch (IOException error){  
  
            //Tratamiento de la excepcion  
  
        }finally{  
  
            //liberar recursos  
  
        }  
  
}
```

Programación Concurrente - Apunte de Excepciones en Java

Creación de nuevos tipos de excepciones:

Un nuevo tipo de excepción puede crearse fácilmente: basta con definir una subclase de un tipo de excepción ya existente.

Por ejemplo:

```
public DivideByZeroException extends ArithmeticException {  
    public DivideByZeroException(String message) {  
        super(message)  
    }  
}
```

Una excepción de este tipo puede entonces lanzarse como cualquier otra excepción:

```
public double dividir(int num, int den) throws  
  
    DivideByZeroException {  
  
    if (den ==0)  
        throw new DivideByZeroException("Error!!!!!!");  
    return((double) num / (double)den);  
}
```

Las aplicaciones suelen definir sus propias subclases de la clase Exception para representar situaciones excepcionales específicas de cada aplicación.-