Exceptiones

Que es una excepción?

En la programación siempre se producen errores, más o menos graves, pero que hay que gestionar y tratar correctamente. Por ello en java disponemos de un mecanismo consistente en el uso de bloques try/catch/finally.
La técnica básica consiste en colocar las instrucciones que podrían provocar problemas dentro de un bloque try, y colocar a continuación uno o más bloques catch, de tal forma que si se provoca un error de un determinado tipo, lo que haremos será saltar al bloque catch capaz de gestionar ese tipo de error específico.

El bloque catch contendrá el código necesario para gestionar ese tipo específico de error. Suponiendo que no se hubiesen provocado errores en el bloque try, nunca se ejecutarían los bloques catch.

Ejemplos

El usuario escribe una palabra cuando se esperaba un número.
El programa intenta leer un fichero que no existe.
El programa no puede establecer una conexión de red.
Una conexión de red se pierde
El programa intenta realizar una división por cero.
Se intenta calcular la raíz cuadrada de un número negativo.
El programa se sale de los límites de un array.
El programa accede a los miembros de un objeto inexistente.

Terminología

- Cuando se produce una excepción, la excepción "se lanza" [*throw*].
- Cuando se prevé el posible lanzamiento de una excepción y se toman medidas al respecto en el código de nuestra aplicación, se "captura" la excepción [*catch*].
- El manejo/tratamiento de excepciones consiste en capturar una excepción y tomar las medidas adecuadas al respecto.

Uso de excepciones en Java

En Java, 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.

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 entrada/salida del programa).

Captura de excepciones: Bloques `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`.

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

La cláusula `catch` recibe como argumento un objeto `Throwable`.

```
// Bloque 1
try {
  // Bloque 2
} catch (Exception error) {
  // Bloque 3
}
// Bloque 4
```

Sin excepciones: $1 \rightarrow 2 \rightarrow 4$

Con una excepción en el bloque 2: $1 \rightarrow 2^* \rightarrow 3 \rightarrow 4$

Con una excepción en el bloque 1: 1^*

```
// Bloque 1
try {
  // Bloque 2
} catch (ArithmeticException ae) {
  // Bloque 3
} catch (NullPointerException ne) {
  // Bloque 4
}
// Bloque 5
```

Sin excepciones: $1 \rightarrow 2 \rightarrow 5$

Excepción de tipo aritmético: $1 \rightarrow 2^* \rightarrow 3 \rightarrow 5$

Acceso a un objeto nulo (`null`): $1 \rightarrow 2^* \rightarrow 4 \rightarrow 5$

Throwable

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.

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 fichero que estemos manipulando).

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

Sin excepciones: $1 \rightarrow 2 \rightarrow 4 \rightarrow 5$

Excepción de tipo aritmético: $1 \rightarrow 2^* \rightarrow 3 \rightarrow 4 \rightarrow 5$

Excepción de otro tipo diferente: $1 \rightarrow 2^* \rightarrow 4$

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

- Detrás del bloque `try` si no se producen excepciones
- Después de un bloque `catch` si éste 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 arriba.

La sentencia `throw`

Se utiliza en Java para lanzar objetos de tipo `Throwable`

```
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 leerFichero (String nombreFichero)
             throws IOException
...
```

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

Al implementar un método, hay que decidir si las excepciones se propagarán hacia arriba (`throws`) o se capturar en el propio método (`catch`)

1. Un método que propaga una excepción:

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

NOTA: 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. Un 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
      // (p.ej. Acceso a un fichero)
  } catch (IOException error) {
      // Tratamiento de la excepción
  } finally {
      // Liberar recursos (siempre se hace)
  }
}
```

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.

```
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);`
}
```

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