Tecnología de la Programación

Tema 6. Excepciones

Alberto Díaz
Purificación Arenas, Yolanda García,
Marco Antonio Gómez, Simon Pickin

Manejo de excepciones

- Las excepciones son el mecanismo que proporciona java para gestionar los errores de ejecución que se producen en un programa.
- Sirven para
  - Dar un aviso de que se puede producir un error
  - Transferir la gestión de los errores a fragmentos de código específicamente destinados a ello
- Una excepción es un evento que se produce durante la ejecución de un programa y que interrumpe el flujo normal de las instrucciones.
  - Un error de ejecución clásico es el intento de acceso a una posición de un array que no existe

Ejemplo

```
public class Desbordamiento {

public static void main(String[] args)
{
          int valores[] = {1, 2, 3};

          for (int i = 0; i <= 3; i++)
                    System.out.println(valores[i]);
}
}
```

```
Exception in thread "main" 1
2
3
java.lang.ArrayIndexOutOfBoundsException: 3
at ejemplos.Desbordamiento.main(Desbordamiento.java:14)
```


Exceptions

- Algunos errores son fatales y provocan que se deba finalizar la ejecución del programa.
  - En este caso conviene terminar ordenadamente y dar un mensaje explicando el tipo de error que se ha producido.

- Otras veces, los errores son recuperables.
  - Por ejemplo no encontrar un fichero en el que hay que leer o escribir algo
  - En este caso el programa debe dar al usuario la oportunidad de corregir el error (indicando una nueva localización del fichero no encontrado).

Exceptions

- Cuando se produce un error :
  - el método correspondiente crea un objeto con la información sobre el tipo de error y el estado del programa cuando se produjo.
- Este objeto se transfiere al sistema, es lo que se llama “lanzar” una excepción.
- Cuando un método lanza una excepción, el sistema trata de encontrar “algo” que lo capture y lo trate.
- Esta búsqueda se realiza en la cadena de métodos que han llevado a la ejecución del método donde se produjo el error.

Jerarquía de exceptions

- Los errores se representan mediante dos tipos de clases derivadas de la clase Throwable
- La siguiente figura muestra parcialmente la jerarquía de clases

Jerarquía de excepciones

- Las excepciones en Java son Objetos

- Clase Throwable
  - Superclase que engloba a todas las excepciones
- Clase Error
  - Representa los errores graves causados por el sistema (JVM, ...), no son tratados por los programas.
- Clase Exception
  - Define las excepciones que los programas deberían tratar
    - IOException, ArithmeticException, ...

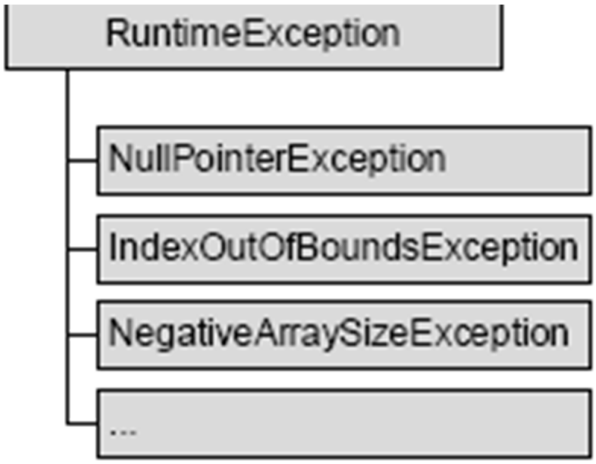
Jerarquía de exceptions

- Dentro de la clase Exception se puede distinguir las excepciones RunTimeException
  - Son excepciones muy frecuentes, de ordinario relacionadas con errores de programación.
  - Se pueden llamar excepciones implícitas, predefinidas o no comprobadas.

- Las demás clases derivadas de Exception son excepciones explícitas o comprobadas.
  - Java obliga a tenerlas en cuenta y chequear si se producen.

RuntimeException Class

- Sería posible comprobar estos tipos de errores, pero el código se complicaría excesivamente si se necesitaran chequear continuamente (referencias distintas de null, que se accede a una posición fuera del rango de un array, etc…)
- Esta clase agrupa las excepciones que son lanzadas automáticamente por Java.



RuntimeException Class

- El propio Java durante la ejecución de un programa chequea y lanza automáticamente las excepciones que derivan de RuntimeException.
- El programador no esta obligado a controlar este tipo de errores específicamente en el código.

RuntimeException Class

- Representan dos casos de errores de programación:
  - Un error que normalmente no suele ser chequeado por el programador, como por ejemplo recibir una referencia null en un método.
  - Un error que el programador debería haber chequeado al escribir el código, como sobrepasar el tamaño asignado de un array (genera un ArrayIndexOutOfBoundsException automáticamente).

Jerarquía de excepciones

- Las clases derivadas de Exception pueden pertenecer a distintos paquetes de Java.
  - Algunas pertenecen a java.lang
    - (Throwable, Exception, RuntimeException, …)
  - Otras a java.io
    - (EOFException, FileNotFoundException, …
  - y a otros paquetes …

- Por heredar de Throwable todos los tipos de excepciones tienen los métodos:
  - String getMessage()
    - Extrae el mensaje asociado con la excepción si existe, o null.
  - void printStackTree()
    - Muestra una traza que permite ver donde se generó el error

Manejo de excepciones

- Cuando se produce una situación anómala se lanza o eleva una excepción
  - Se crea un objeto Exception de la clase adecuada
  - Se lanza con la sentencia throw
    - MiException miExc = new MiException(“Mi excepción”);
    - throw miExc;
  - Al lanzar una excepción termina la ejecución del método

Lanzar una exception

- Todo método en el que se puede producir uno o más tipos de excepciones debe declararlas en el encabezamiento de la función por medio de la palabra throws.
- No es necesario para excepciones de las clases Error o RuntimeException.
- Si un método puede lanzar varias excepciones, se ponen detrás de throws separadas por comas:

```
public void leerFichero (String fich) throws EOFException, FileNotFoundException {
```

Lanzar una exception

- Se puede poner únicamente una superclase de excepciones para indicar que se pueden lanzar excepciones de cualquiera de sus clases derivadas:

```
public void leerFichero (String fich) throws IOException{

}
```

- ya que EOFException y FileNotFoundException derivan de IOException.

- Las excepciones pueden ser lanzadas por leerFichero() o por alguno de los métodos llamados por leerFichero().

Captura de exceptions

- Al lanzar una excepción el método termina de inmediato, sin devolver ningún valor, a menos que esta excepción sea capturada.

- El compilador obliga a capturar las llamadas excepciones explícitas, pero no protesta si se captura y luego no se hace nada con ella.
- En general, es conveniente por lo menos imprimir un mensaje indicando qué tipo de excepción se ha producido.

- Por tanto, una excepción deberá ser capturada y gestionada en el propio método o en algún otro lugar del programa.

Captura de exceptions

- La estructura try-catch-finally nos permitirá capturar excepciones, es decir, reaccionar a un error de ejecución.

```
try {
// Código que puede hacer que se lance la excepción
}
catch(TipoExcepcion e) {
// Gestor de la excepción
}
```

- Si en la ejecución del código dentro del bloque try se eleva una excepción de tipo TipoExcepcion (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 (gestor).

Captura de exceptions

```
public class EjemploCatch  {

    private static String mensajes[] = {"Primero", "Segundo", "Tercero" };

    public static void main(String[] args) {
        try {
            for(int i = 0; i <= 3; i++)
                System.out.println(mensajes[i]);
        }
        catch ( ArrayIndexOutOfBoundsException e ) {
            System.out.println("El asunto se nos ha desbordado");
        }
        finally {
            System.out.println("Ha finalizado la ejecución");
        }
    }
}
```

Bloque try-catch-finally

- try
  - El bloque de código donde se prevé que se eleve una excepción. Al encerrar el código en un bloque try es como si dijéramos:
    - El código dentro de este bloque try está “vigilado”
  - El bloque try tiene que ir seguido, al menos, por una cláusula catch o una cláusula finally.
  - Si se produce una situación anormal y se lanza por lo tanto una excepción el control sale del bloque try y pasa al bloque catch.

Bloque try-catch-finally

- catch
  - Se encarga de la situación y decide lo que hay que hacer.
  - Es el código que se ejecuta cuando se lanza la excepción
  - Controla cualquier excepción que cuadre con su argumento.
  - Se pueden incluir tantos bloques catch como sean necesarios, cada uno de los cuales tratará un tipo de excepción.
  - Las excepciones se pueden capturar individualmente o en grupo, por medio de una superclase de la que deriven todas ellas.

Captura de exceptions

- A catch le sigue, entre paréntesis, la declaración de una excepción.
- Es decir, el nombre de una clase derivada de Exception (o la propia Exception) seguido del nombre de una variable.
- Si se lanza una excepción que es la que deseamos capturar (o una derivada de la misma) se ejecutará el código que contiene el bloque.

```
catch ( Exception e ) {
    System.out.println("El asunto se nos ha desbordado");
}
```

- En este caso se ejecutará siempre que se produzca una excepción del tipo que sea, ya que todas las excepciones derivan de Exception.

Captura de exceptions

- Se pueden colocar varios bloques catch.
- Si es así, se comprobará, en el mismo orden en que se encuentren esos bloques catch, si la excepción elevada es la que se trata en el bloque catch; si no, se pasa a comprobar el siguiente.
- Sólo se ejecuta un bloque catch. En cuanto se captura la excepción se deja de comprobar el resto de los bloques.

```
catch ( ArrayIndexOutOfBoundsException e ) {
    System.out.println("El asunto se nos ha desbordado");
}
catch ( Exception e ) {
    System.out.println("Resto de excepciones");
}
```

Ejemplo: Sin try-catch

```
public class EjemploExcep  {

  public static void main(String[] args) {
          int a = args.length;
          System.out.println("a = " + a);
          int b = 42 / a;
          System.out.println("b = " + b);
          }
 }
```

Se produce el error y se interrumpe la ejecución….

- a = 0
- java.lang.ArithmeticException: / by zero
- at Excepcion1.main(Excepcion1.java:6)
- Exception in thread "main" Process Exit...

Ejemplo: Con try-catch

```
public class EjemploExcep  {

    public static void main(String[] args) {
        try {
            int a = args.length;
            System.out.println("a = " + a);
            int b = 42 / a;
            System.out.println("b = " + b);
        }
        catch (ArithmeticException e) {
            System.out.println("No dividas por 0 (" + e + ")");
        }
        System.out.println("La ejecución sigue ...");
    }
}
```

Se captura el error y se lanza la exception

Se captura la exception

Continua la ejecución

- a = 0
- No dividas por 0 (java.lang.ArithmeticException: / by zero)
- La ejecución sigue ...

Bloque try-catch-finally

- finally
  - Bloque que se ejecuta siempre, haya o no excepción.
  - Debe ir detrás de todos los bloques catch considerados
  - El bloque finally es opcional. Sus sentencias se ejecutan siempre, sea cual sea la excepción que se produzca o si no se produce ninguna.
  - El bloque finally se ejecuta aunque en el bloque try haya una sentencia continue, break o return.

Bloque try-catch-finally

- Como ejemplo se podría pensar en un bloque try dentro del cual se abre un fichero para lectura y escritura de datos y se desea cerrar el fichero abierto.
- El fichero abierto se debe cerrar tanto si produce una excepción como si no se produce, ya que dejar un fichero abierto puede provocar problemas posteriores.
- Para conseguir esto se deberá incluir las sentencias correspondientes a cerrar el fichero dentro del bloque finally.

Bloque try-catch-finally

- La forma general de una sección donde se controlan las excepciones es por lo tanto:

```
try {
// Código “vigilado” que puede lanzar una excepción de tipo A, B o C
}
catch (A a1) {
// Se ocupa de la excepción A
}
catch (B b1) {
// Se ocupa de la excepción B
}
catch (C c1) {
// Se ocupa de la excepción C
}
finally {
// Sentencias que se ejecutarán en cualquier caso
}
```


Relanzar una Exception

- Existen algunos casos en los cuales el código de un método puede generar una Exception y no se desea incluir en dicho método la gestión del error.
- Java permite que este método pase o relance la Exception al método desde el que ha sido llamado, sin incluir en el método los bloques try/catch correspondientes.
- Esto se consigue mediante la adición de throws más el nombre de la Exception concreta después de la lista de argumentos del método.
- A su vez el método superior deberá incluir los bloques try/catch o volver a pasar la Exception.

Relanzar una Exception

- De esta forma se puede ir pasando la Exception de un método a otro hasta llegar al último método del programa, el método main().

```
public void metodo1(){
    try {
        metodo2();
    }
    catch (IOException e) {
        // Se ocupa de IOException simplemente dando aviso
        System.out.println(e.getMessage());
    }
    finally { // Sentencias que se ejecutarán en cualquier caso
    }
}
```

```
public void metodo2() throws IOException {

        // Código que puede lanzar las excepciones IOException

} // Fin del metodo2
```


Relanzar una Exception

- Si un método llama a otros métodos que pueden lanzar excepciones tiene 2 posibilidades:
  - Capturar las posibles excepciones y gestionarlas.
  - Desentenderse de las excepciones y remitirlas hacia otro método anterior en el stack para que éste se encargue de gestionarlas.
- Si no hace ninguna de las dos cosas anteriores el compilador da un error, salvo que se trate de una RuntimeException.

Nuevas excepciones

- El programador puede crear sus propias excepciones personalizadas si ninguna de las predefinidas es adecuada.
- Para ello, se define una clase que deriva de Exception (o de la clase deseada).
- Lo lógico es heredar de la clase de la jerarquía de Java que mejor se adapte al tipo de excepción.

Nuevas excepciones

- Se agrega un constructor que recibe un String como argumento.
- En este String se suele definir un mensaje que explica el tipo de excepción generada.
- Conviene que este constructor llame al constructor de la clase de la que deriva super(String).

```
public class nuevaExcepcion extends Exception
    {
        public nuevaExcepcion(String texto)
        {
            super(texto);
        }
    }
```

Nuevas excepciones

```
ublic class UnaClase {

    public void metodo() throws MiExcepcion {
        System.out.println("Lanzo mi excepción desde aquí.");
        throw new MiExcepcion();
    }
}
```

```
public class MiExcepcion extends Exception {

    public MiExcepcion(){
        super("Error malísimo...");
    }
}
```

```
ublic class EjemploExcep  {

    public static void main(String[] args) {

        UnaClase c = new UnaClase();
        try {
            c.metodo(); // Invoco al método que eleva la excepción
        }
        catch(MiExcepcion e) {
            System.out.println("La tengo! " + e);
        }
        System.out.println("... y sigo.");
    }
```

- Salida:

```
Lanzo mi excepción desde aquí
La tengo! MiExcepcion: error malísimo...
... y sigo.
```