Excepciones en Java

Introducción - por qué usarlas?(1)

- En el mundo de la programación hay algo que siempre ocurre: **los errores en los programas.**
- Pero ¿qué sucede realmente después de que ha ocurrido el error? ¿Qué podemos hacer con él? ¿Cómo nos damos cuenta que se ha producido?
- El lenguaje Java utiliza excepciones para permitir trabajar mejor con los errores.

Introducción - por qué usarlas?(2)

- En la programación tradicional, la detección, el informe y el manejo de errores se convierte en un código muy liado. En pseudo-código:

```
leerFichero
{
   abrir el fichero;
   determinar su tamaño;
   asignar suficiente memoria;
   leer el fichero a la memoria;
   cerrar el fichero;
}
```

- A primera vista esta función parece bastante sencilla, pero ignora todos aquello errores potenciales.
  - ¿Qué sucede si no se puede abrir el fichero?
  - ¿Qué sucede si no se puede determinar la longitud del fichero?
  - ¿Qué sucede si no hay suficiente memoria libre?
  - ¿Qué sucede si la lectura falla?
  - ¿Qué sucede si no se puede cerrar el fichero?
- Para responder a estas cuestiones dentro de la función, tendríamos que añadir mucho código para la detección y el manejo de errores.

Introducción - por qué usarlas?(3)

- En caso de no tenerlas, como en otros lenguajes de programación como Perl, tendremos que utilizar incómodos bloques if para controlar los errores (peor, podemos elegir ignorar que estos errores pueden occurrir). Algo parecido a esto:

```
class InputFile {
      FileInputStream fis;
      InputFile(String filename) {
                fis = new FileInputStream(filename);
                if (fis == file_not_found) {
                  System.out.println("Error!, el fichero no existe");
                  return;
                }
                else if (fis == sin_memoria) {
                  System.out.println("Error, sin memoria");
                  return;
                }
                else if… (muchos más aún…)
      }
}
```

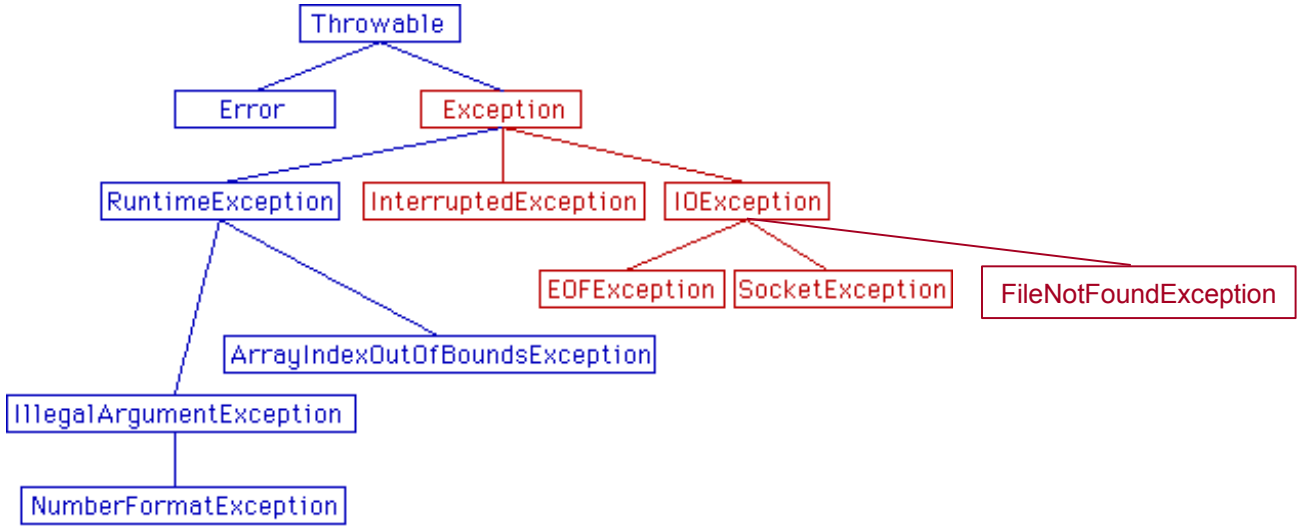
Introducción - por qué usarlas?(4)

- Mediante el uso de excepciones para controlar errores, los programas Java tienen las siguientes ventajas frente a las técnicas de manejo de errores tradicionales.
  - **Separar el Manejo de Errores del Código "Normal“.**
    - **Estará en una zona separada donde podremos tratar las excepciones como un código ‘especial’.**
  - **Propagar los Errores sobre la Pila de Llamadas**
    - **podemos propagar el error a la primera función que llamó a las diversas funciones hasta que llegamos al error.**
  - **Agrupar Errores y Diferenciación.**
    - **Gracias a esto tenemos todos los posibles errores juntos y podemos pensar una manera de tratarlos que sea adecuado.**

¿Qué es una excepción?

- Una **excepción** es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias.
  - O sea, algo que altera la ejecución normal.
- Muchas clases de errores pueden generar excepciones -- desde problemas de hardware, como la avería de un disco duro, a los simples errores de programación, como tratar de acceder a un elemento de un array fuera de sus límites.

Jerarquia de excepciones de Java



En rojo: son las excepciones que es obligatorio controlar. Checked exceptions

En azul: son las excepciones que no es obligatorio controlar. Unchecked exceptions

Uso de Excepciones Java (1)

- Este código

```
public class Hello {
        public static void main(String argv[]){
                int uno_diez[] = new int[10];
                        uno_diez[12] = 10;
        }
}
```

- Produciría esto.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 12
    at Hello.main(Hello.java:7)
```

- Qué podemos hacer al respecto?
  - Podemos decirle al compilador que queremos tratar esa excepción, y aunque ocurra, el programa no terminará y haremos lo que queremos con ella.

Uso de Excepciones Java (2)

- Y lo haríamos así, de manera que el programa no finalizaría con ese error, sino que podemos personalizar un mensaje y decidir si continuar o no.

```
public class Hello {
      public static void main(String argv[]){
        int uno_diez[] = new int[10];
        try {
          uno_diez[12] = 10;
        } catch(java.lang.ArrayIndexOutOfBoundsException) {
          System.out.println("Error, hemos sobrepasado el
   tamaño del array");
          // aquí podemos hacer lo que queramos
        }
      }
}
```

Uso de Excepciones Java (3)

- Hay algunos métodos/constructores como FileInputStream, que lanzan excepciones, y debemos tenerlo en cuenta, sino podemos encontrarnos con errores de compilación como este:

```
InputFile.java:8: Warning: Exception java.io.FileNotFoundException must be caught,
   or it must be declared in throws clause of this method.
   fis = new FileInputStream(filename);
         ^
```

- Clase InputFile:

```
public class InputFile {
  FileInputStream fis;
  InputFile(String filename) { // el constructor de la clase InputFile
      fis = new FileInputStream(filename);
  }
}
```

Uso de Excepciones Java (3)

- Como puedes ver en el listado, la clase InputFile ignora completamente el hecho de que el constructor de FileInputStream puede lanzar un excepción.
- Sin embargo, el lenguaje Java requiere que un método haga algo con las excepciones que se pueden lanzar desde el (por ejemplo, dentro del constructor InputFile, FileInputStream puede lanzar una excepción) .
- Se pueden hacer 2 cosas:
  - Lanzar la excepción al método llamante (con ***throws)***. Por ejemplo InputFile puede lanzar a su vez la excepción lanzada por FileInputStream. Eso es lo que hicimos en Sumar.java
  - Tratar todas las excepciones que pueden ser lanzadas desde dentro de su ámbito (como en la transparencia 9).
- Como la clase InputFile no hace ninguna de las dos cosas, el compilador rehúsa su compilación e imprime el mensaje de error.

Uso de Excepciones Java (4)

- El compilador dará el primer error en la primera línea que está en negrita (en la diapositiva 9). Esta línea crea un objeto FileInputStream y lo utiliza para abrir un fichero (cuyo nombre se pasa dentro del constructor de FileInputStream).
- Entonces, ¿Qué debe hacer el FileInputStream si el fichero no existe? Bien, eso depende de lo que quiera hacer el programa que utiliza el FileInputStream.
- Los implementadores de FileInputStream no tenían ni idea de lo que quiere hacer la clase InputFile si no existe el fichero.
  - ¿Debe FileInputStream terminar el programa?
  - ¿Debe intentar un nombre alternativo?
  - ¿deberá crear un fichero con el nombre indicado?
- No existe un forma posible de que los implemetadores de FileInputStream pudieran elegir una solución que sirviera para todos los usuarios de FileInputStream. Por eso ellos lanzaron una excepción. Esto es, si el fichero nombrado en el argumento del constructor de FileInputStream no existe, el constructor lanza una excepción **java.io.FileNotFoundException**. Mediante el lanzamiento de esta excepción, FileInputStream permite que el método llamador maneje ese error de la forma que considere más apropiada.

Cómo Especificar las Excepciones Lanzadas por un Método

- Podemos hacer que un método nuestro lance excepciones y lo especificaremos así (aquí se lanzan 2 excepciones de tipo IOException y ArrayIndexOutOfBoundsException):
  - public void writeList() throws IOException, ArrayIndexOutOfBoundsException {
- Todos los métodos Java utilizan la sentencia **throw** para lanzar una excepción.
  - Esta sentencia requiere un sólo argumento, un objeto Throwable. En el sistema Java, los objetos lanzables son ejemplares de la clase Throwable definida en el paquete java.lang, como por ejemplo:
    - throw **new EmptyStackException();**

Requerimientos Java para Capturar o Especificar Excepciones

- **Capturar (try-catch).** Un método puede capturar una excepción proporcionando un manejador para ese tipo de excepción.
- **Especificar (throws).** Si un método decide no capturar una excepción, debe especificar que puede lanzar esa excepción.
  - ¿Por qué hicieron este requerimiento los diseñadores de Java?
    - Porque una excepción que puede ser lanzada por un método es realmente una parte del interface de programación público del método. Así, en la firma del método debe especificar las excepciones que el método puede lanzar.

Capturar y Manejar Excepciones(1)

- **El bloque try**
  - El primer paso en la escritura de una manejador de excepciones es poner la sentencia Java dentro de la cual se puede producir la excepción dentro de un bloque **try**.
  - Se dice que el bloque **try** gobierna las sentencias encerradas dentro de él y define el ámbito de cualquier manejador de excepciones (establecido por el bloque **catch**) asociado con él.
- **Los bloques catch**
  - Después se debe asociar un manejador de excepciones con un bloque **try** proporcionándole uno o más bloques **catch** directamente después del bloque **try**.
- **El bloque finally**
  - El bloque **finally** de Java proporciona un mecanismo que permite a sus métodos limpiarse a si mismos sin importar lo que sucede dentro del bloque **try**. Se utiliza el bloque **finally** para cerrar ficheros o liberar otros recursos del sistema despues de que occurra una excepción.
- EJEMPLO (aquí capturamos y manejamos 2 tipos de excepciones):

```
try {
 . . .
}
catch (ArrayIndexOutOfBoundsException e) {
System.err.println("Caught ArrayIndexOutOfBoundsException: " + e.getMessage());
}
catch (IOException e) {
System.err.println("Caught IOException: " + e.getMessage());
}
```

Capturar y Manejar Excepciones(2)

- Supongamos que ocurre una excepción IOException dentro del bloque **try**.
  - El sistema de ejecución inmediatamente toma posesión e intenta localizar el manejador de excepción adecuado.
  - Empieza buscando al principio de la pila de llamadas.
  - Sin embargo, si el constructor de FileOutputStream no tiene un manejador de excepción apropiado por eso el sistema de ejecución comprueba el siguiente método en la pila de llamadas -- el método **writeList()**.

Capturar Varios Tipos de Excepciones con un Manejador

```
try {
. . .
}
catch (Exception e) { System.err.println("Exception caught: " + e.getMessage());
 }
```

- La clase Exception está bastante arriba en el árbol de herencias de la clase Throwable. Por eso, además de capturar los tipos de IOException y ArrayIndexOutOfBoundsException este manejador de excepciones, puede capturar otros muchos tipos. Generalmente hablando, los manejadores de excepción deben ser más especializados.

Un ejemplo

```
public void writeList()
{
    PrintStream pStr = null;
    try {
          int i;
          System.out.println("Entrando en la Sentencia try");
          pStr = new PrintStream(new BufferedOutputStream( new
                    FileOutputStream("OutFile.txt")));
          for (i = 0; i < size; i++)
                    pStr.println("Value at: " + i + " = " +
                              miVector.elementAt(i));
    } catch (ArrayIndexOutOfBoundsException e) {
          System.err.println("Caught ArrayIndexOutOfBoundsException: " +
                    e.getMessage());
    } catch (IOException e) {
          System.err.println("Caught IOException: " + e.getMessage()); }
    finally {
          if (pStr != null) { System.out.println("Cerrando PrintStream");
          pStr.close();
          } else {
                    System.out.println("PrintStream no está abierto");
          }
    }
}
```

Un ejemplo (explicación)

- El bloque **try** de este método tiene tres posibilidades de salida direrentes.
  - La sentencia **new FileOutputStream** falla y lanza una IOException.
  - La sentencia **victor.elementAt(i)** falla y lanza una ArrayIndexOutOfBoundsException.
  - Todo tiene éxito y la sentencia **try** sale normalmente.

Breve Resumen

- Las excepciones sirven para tratar errores en la ejecución del programa.
- Las funciones de clases que lanzan (throw) excepciones, deben ser llamadas desde métodos que las tratan con un bloque try… catch, o que vuelven a lanzan las excepciones.
- Podemos definir métodos que lanzan excepciones.

Más información

- Bradley Kjell, Curso Online, Capitulos 80 y 81:
  http://chortle.ccsu.edu/CS151/cs151java.html
- Handling Errors with Exceptions - tutorial de Sun:
  http://java.sun.com/docs/books/tutorial/essential/exceptions/