Unidad 8: Control de Errores Mediante Excepciones

Fundamentos de Programación. 1º de ASI

Para ver una copia completa de la licencia, acudir a la dirección
http://creativecommons.org/licenses/by-nc-sa/2.5/es/legalcode.es

(8)
control de errores

esquema de la unidad

(8.1) introducción a las excepciones

Uno de los problemas más importantes al escribir aplicaciones es el tratamiento de los errores. Los errores detienen la ejecución del programa e impiden su desarrollo normal y, lo peor, además provocan que el usuario esté desinformad. Toda programadora o programador tiene que reconocer las situaciones que pueden provocar el fin de la ejecución normal del programa por un error no controlado.

Dicho de otra forma, todos los posibles errores en un programa deben de estar controlado. A veces es imposible evitarlos (por ejemplo no hay papel en la impresora, o falla el disco duro), pero sí reaccionar de forma lógica para que el usuario o usuaria reconozca lo que está ocurriendo.

Java nos echa una mano para ello a través de las **excepciones**. Se denomina excepción a un hecho que podría provocar la detención del programa; es decir una condición de error en tiempo de ejecución pero que puede ser controlable (a través de los mecanismos adecuados). En java sin embargo se denomina **error** a una condición de error incontrolable (ejemplos son el error que ocurre cuando no se dispone de más memoria o errores de sintaxis).

Ejemplos de excepciones son:

- El archivo que queremos abrir no existe
- Falla la conexión a una red

- Se intenta dividir entre cero

Los errores de sintaxis son detectados durante la compilación, pero los errores de ejecución pueden provocar situaciones irreversibles, su control debe hacerse también en tiempo de ejecución y eso siempre ha sido problemático para la programación de aplicaciones.

En Java se puede preparar el código susceptible a provocar excepciones de modo que si ocurre una excepción, el código es ***lanzado*** *(**throw**)* a una determinada rutina previamente preparada por el programador, que permite manipular esa excepción. Si la excepción no fuera capturada, la ejecución del programa se detendría irremediablemente (en muchas ocasiones la propia sintaxis de Java impide que la excepción no sea controlada; es decir, obliga a controlarla).

En Java hay muchos tipos de excepciones (de operaciones de entrada y salida, de operaciones irreales,...). El paquete **java.lang.Exception** y sus subpaquetes contienen todos los tipos de excepciones.

Cuando se produce una excepción, se genera un objeto asociado a la misma. Este objeto es de clase **Exception** o de alguna de sus herederas. Este objeto se envía como parámetro al código que se ha definido para manejar la excepción. Dicho código puede manipular las propiedades del objeto **Exception**.

Hay una clase, la **java.lang.Error** y sus subclases que sirven para definir los errores irrecuperables más serios. Esos errores causan parada en el programa sin que el programador o programadora pueda evitarlo. Estos errores les produce el sistema y son incontrolables para el programador. Las excepciones son fallos más leves, y más manipulables.

(8.2) try y catch

El control de las excepciones se realiza mediante las sentencias **try** y **catch.** La sintaxis es:

```
try {
        instrucciones que se ejecutan salvo que haya un error
}
catch (ClaseExcepción objetoQueCapturaLaExcepción) {
        instrucciones que se ejecutan si hay un error
}
```

Puede haber más de una sentencia **catch** para un mismo bloque **try.**

Ejemplo:

```
try {
        readFromFile("arch");
        ...
}
catch(FileNotFoundException e) {
   //archivo no encontrado
   ...
}
catch (IOException e) {
   ...
}
```

Si en las instrucciones del bloque **try** hay un error causado por no encontrar el archivo se producirá una excepción de tipo **FileNotFound** y será manejada por el catch correspondiente, si se produce del otro tipo (**IOException**) se maneja por el siguiente, si se produce de otro tipo, el programa se detendrá.

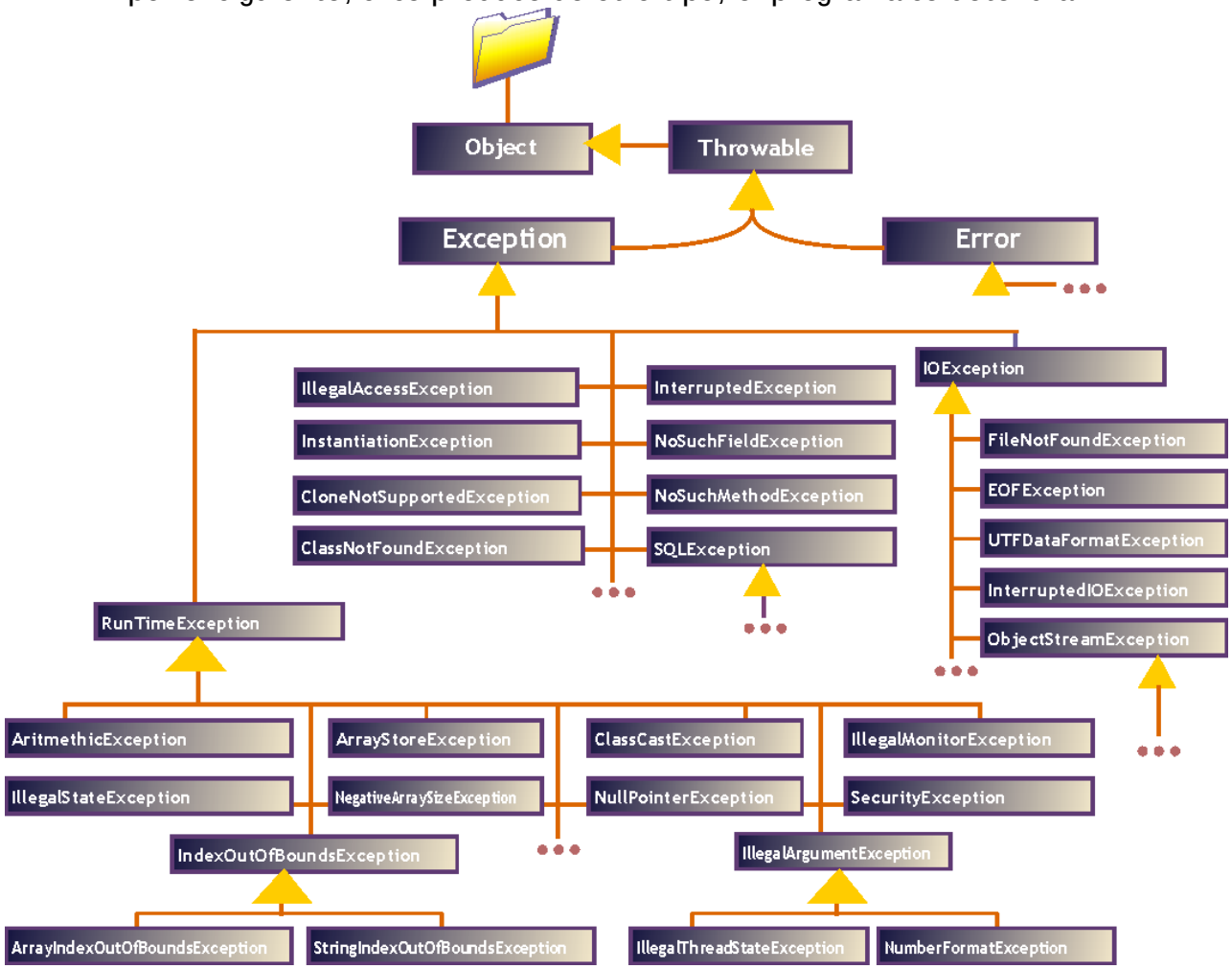

Ilustración 8-1, Diagrama de algunas de las clases fundamentales para el control de excepciones

Dentro del bloque **try** se colocan las instrucciones susceptibles de provocar una excepción, el bloque **catch** sirve para capturar esa excepción y evitar el fin de la ejecución del programa. Desde el bloque catch se maneja, en definitiva, la excepción.

Cada catch maneja un tipo de excepción. Cuando se produce una excepción, se busca el catch que posea el manejador de excepción adecuado, será el que utilice el mismo tipo de excepción que se ha producido. Esto puede causar problemas si no se tiene cuidado, ya que la clase **Exception** es la superclase de todas las demás. Por lo que si se produjo, por ejemplo, una excepción de tipo **AritmethicException** y el primer **catch** captura el tipo genérico **Exception**, será ese catch el que se ejecute y no los demás.

Por eso el último catch debe ser el que capture excepciones genéricas y los primeros deben ser los más específicos. Lógicamente si vamos a tratar a todas las excepciones (sean del tipo que sean) igual, entonces basta con un solo **catch** que capture objetos **Exception**. Ejemplo (de mal uso):

```
int x;
try{
   x=Integer.parseInt(
         JOptionPane.showInputDialog("Escriba un número"));
}
catch (Exception e) {
   JOptionPane.showMessageDialog(null, "Error indeterminado");
}
catch (NumberFormatException e) {
   JOptionPane.showMessageDialog(null, "El número no es válido");
}
```

El código del recuadro no es alcanzable, porque siempre se ejecutaría el primer catch. De hecho hoy en día Java marca como error ese código. Lo correcto es:

```
int x;
try{
   x=Integer.parseInt(
         JOptionPane.showInputDialog("Escriba un número"));
}
catch (NumberFormatException e) {
   JOptionPane.showMessageDialog(null, "El número no es válido");
}
catch (Exception e) {
   JOptionPane.showMessageDialog(null, "Error indeterminado");
}
```

Ahora sí

(8.3) manejo de excepciones

Siempre se debe controlar una excepción, de otra forma nuestro software está a merced de los fallos. En la programación de aplicaciones en general siempre ha habido dos formas de manejar la excepción:

- **Interrupción.** En este caso se asume que el programa ha encontrado un error irrecuperable. La operación que dio lugar a la excepción se anula y se entiende que no hay manera de regresar al código que provocó la excepción. Es decir, la operación que dio pies al error, se anula.
- **Reanudación.** Se puede manejar el error y regresar de nuevo al código que provocó el error.

La filosofía de Java es del tipo interrupción, pero se puede intentar emular la reanudación encerrando el bloque **try** en un bucle que se repetirá hasta que el error deje de existir. Ejemplo:

```
boolean indiceNoValido=true;
int i; //Entero que tomará nos aleatorios de 0 a 9
String texto[]={"Uno","Dos","Tres","Cuatro","Cinco"};
while(indiceNoValido){
   try{
         i=(int)(Math.round(Math.random()*9));
         System.out.println(texto[i]);
         indiceNoValido=false;
   }catch(ArrayIndexOutOfBoundsException exc){
         System.out.println("Fallo en el índice");
   }
}
```

En el código anterior, el índice ***i*** calcula un número aleatorio del 0 al 9 y con ese número el código accede al array ***texto*** que sólo contiene 5 elementos. Esto producirá continuamente (ya que el array es mucho más pequeño) una excepción del tipo **ArrayIndexOutOfBoundsException** que es manejada por el **catch** correspondiente.

Para buscar otro intento, el bloque **catch** está dentro de un bucle **while**, que permite otro intento y así hasta que no haya excepción (es decir hasta que el número esté entre 0 y 5), lo que provocará que ***indiceNovalido*** valga ***true*** y la salida, al fin, del **while**. Es un tanto enrevesado el código pero valga como ejemplo del funcionamiento de la captura de errores con reintento.

(8.4) métodos de la clase Exception

Como se observa en la Ilustración 8-1, la clase **Exception** es la superclase de todos los tipos de excepciones. Esto permite utilizar una serie de métodos comunes a todas las clases de excepciones:

- **String getMessage().** Obtiene el mensaje descriptivo de la excepción o una indicación específica del error ocurrido:

```
try{

} catch (IOException ioe){
   System.out.println(ioe.getMessage());
}
```

- **String toString().** Escribe una cadena sobre la situación de la excepción. Suele indicar la clase de excepción y el texto de **getMessage()**.
- **void printStackTrace()**. Escribe el método y mensaje de la excepción (la llamada información de pila). El resultado es el mismo mensaje que muestra el ejecutor (la máquina virtual de Java) cuando no se controla la excepción.

(8.5) throws

Al llamar a métodos, ocurre un problema con las excepciones. El problema es, si el método da lugar a una excepción, ¿quién la maneja? ¿El propio método?¿O el código que hizo la llamada al método?

Con lo visto hasta ahora, sería el propio método quien se encargara de sus excepciones, pero esto complica el código ya que descentraliza el control de excepciones y dificulta el mantenimiento del código. Por eso otra posibilidad es hacer que la excepción la maneje el código que hizo la llamada.

Esto se hace añadiendo la palabra **throws** tras la primera línea de un método. Tras esa palabra se indica qué excepciones puede provocar el código del método. Si ocurre una excepción en el método, el código abandona ese método y regresa al código desde el que se llamó al método. Allí se posará en el catch apropiado para esa excepción. Ejemplo:

```
void usarArchivo (String archivo) throws IOException,
   InterruptedException {
         ...
}
```

En este caso se está indicando que el método *usarArchivo* puede provocar excepciones del tipo **IOException** y **InterruptedException**. Lo cual **obliga** a que el código que invoque a este método deba preparar el (o los) **catch**

correspondientes. Cuando un método hace uso de **throws**, en el **javadoc** del mismo se documenta de esta forma:

```
/**
* Método que abre y prepara el archivo
* @param archivo Nombre del archivo a preparar
* @throws IOException Si ocurre un error en la apertura
* @throws InterruptedException  Si hay un error en el
* contenido del archivo
*/

void usarArchivo (String archivo) throws IOException,
   InterruptedException {
         ...
}
```

Para utilizar el método:

```
try{
   ...
   objeto.usarArchivo("C:\texto.txt");//puede haber excepción
   ..
}
catch(IOException ioe){
   ...
}
catch(InterruptedException ie){
   ...
}
...
```

(8.6) throw

Esta instrucción nos permite provocar a nosotros una excepción (o lo que es lo mismo, crear artificialmente nosotros las excepciones). Ante:

```
throw new Exception();
```

El flujo del programa se dirigirá a la instrucción **try…catch** más cercana. Se pueden utilizar constructores en esta llamada (el formato de los constructores depende de la clase que se utilice):

```
throw new Exception("Error grave, grave");
```

Eso construye una excepción con el mensaje indicado.

throw permite también ***relanzar*** excepciones. Esto significa que dentro de un catch podemos colocar una instrucción **throw** para lanzar la nueva excepción que será capturada por el catch correspondiente:

```
try{
   ...
} catch(ArrayIndexOutOfBoundsException exc){
   throw new IOException();
} catch(IOException){
   ...
}
```

El segundo catch capturará también las excepciones del primer tipo

(8.7) finally

La cláusula **finally** está pensada para limpiar el código en caso de excepción. Su uso es:

```
try{
   ...
}catch (FileNotFoundException fnfe){
   ...
}catch(IOException ioe){
   ...
}catch(Exception e){
   ...
}finally{
   ...//Instrucciones de limpieza
}
```

Las sentencias **finally** se ejecutan tras haberse ejecutado el catch correspondiente. Si ningún catch capturó la excepción, entonces se ejecutarán esas sentencias antes de devolver el control al siguiente nivel o antes de romperse la ejecución.

Hay que tener muy en cuenta que **las sentencias finally se ejecutan independientemente de si hubo o no excepción**. Es decir esas sentencias se ejecutan siempre, haya o no excepción. Son sentencias a ejecutarse en todo momento. Por ello se coloca en el bloque **finally** código común para todas las excepciones (y también para cuando no hay excepciones.