INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA DE INGENIERÍA Y CIENCIAS SOCIALES Y ADMINISTRATIVAS

Material para el curso de Programación Orientada a Objetos

Contenido

3.4 Excepciones

Son situaciones que pueden ocurrir durante la ejecución de una Aplicación o Applet, pero que a diferencia de los errores propiamente dichos, pueden ser interceptadas para que dicha ejecución no sea interrumpida. Sus causas son muy variadas:

- Condiciones no anticipadas de datos ilegales, como intentar dividir entre cero, o utilizar un valor inválido como índice en un arreglo.
- Acciones inapropiadas del usuario, como intentar abrir un archivo no existente.
- Condiciones normales que son detectadas a través de errores, como un usuario intentando escribir en un registro de Base de Datos bloqueado.

El mecanismo default de Java para manejar las problemas es enviar un mensaje, y dependiendo del problema, detener la Aplicación. La diferencia concreta entre estos 2 tipos de problemas es:

- Errores. No son recuperables, es decir, detienen la Aplicación, por ejemplo: VirtualMachineError, OutOfMemoryError.
- Excepciones. Sí pueden ser recuperables como veremos, por ejemplo: ArithmeticException, FileNotFoundException, IOException, SQLException.

Como su nombre nos sugiere, se trata de clases que heredan respectivamente de Error y Exception. Cuando escribimos código para atrapar y responder a una excepción, se dice que estamos manejando (handling) dicha excepción:

```
try {
        promedio= (float) acum/cont;
}
catch (ArithmeticException e) {    // si ocurre división entre 0
        promedio= 0.0f;
        System.out.println ("División entre 0, no hay alumnos");
}
catch (Exception e) {                  // cualquier otra
        promedio= 0.0f;
        System.out.println ("Otra excepción " + e.toString ());
}
finally {
        System.out.println ("El promedio es: " + promedio);
};
```

Esta estructura tiene 3 partes:

- El bloque *try*, que contiene el código donde podría ocurrir alguna excepción. Se recomienda que este bloque sea lo más breve posible, por cuestiones de performance o tiempo de ejecución.
- Uno o más bloques *catch*, los cuales responden a varios tipos de excepciones. Estos bloques deben ir de la excepción más particular (subclase) hasta la más general.
- Un bloque *finally* opcional, que contiene código que se ejecutara siempre, no importando si ocurrió o no alguna excepción.

Si una línea del código dentro del *try* causa una excepción, el flujo del programa salta al primer *catch* que corresponda.

Cuando es generada, una excepción existirá hasta que sea atrapada por un *catch*; si no se atrapa inmediatamente, pasara hacia donde se invoco ese método, subiendo en lo que se llama el *call stack* o pila de invocaciones; si llega hasta el método main y aun no es atrapada, Java despliega un mensaje y el *call stack.*

Métodos que lanzan excepciones

Algunos ejemplos de métodos que pueden lanzarlas:

CLASE	METODO	EXCEPCION
Thread	public static void sleep (long millisec)	InterruptedException
System	public static void setSecurityManager (SecurityManager s)	SecurityException
URL	URL(String specific)	MalformedURLException

Siempre que llamamos a un método que puede lanzar una excepción, esta debe ser atrapada (y si eso no puede ocurrir, **no debemos intentar atraparla**). Tenemos 2 maneras de hacerlo:

1. Poner la llamada dentro de un bloque *try*, como se vio en el ejemplo anterior (y siempre tratando de minimizar su tamaño).
2. Modificar la declaración del método que lo llama con la palabra *throws* para indicar que este último no la maneja sino que simplemente la deja pasar, transfiriendo la responsabilidad al método del nivel anterior. Por ejemplo:

```
public URL getURL(String specific) throws MalformedURLException {
        URL u;
        u = new URL(specific);      // este constructor puede lanzarla
        return u;

}
```

Incluso podemos en algunas situaciones manipular la excepción en un nivel más alto. Esto se hace volviendo a lanzar la excepción dentro del bloque *catch* después de ser atrapada, mediante la palabra *throw*, como en el siguiente ejemplo:

```
public int division (int x, int y) {
        int resul;
        try {
                result = x / y;
        }
        catch (ArithmeticException e) {
                resul = 0;
                throw e;        // debe ser atrapada por quien llame a división
        }
        finally {               // esto ocurre antes del throw
                return resul;
        };
}
```

Excepciones más comunes

EXCEPCION	
NullPointerException	Intento de accesar un miembro usando una variable que no referencia a un objeto
FileNotFoundException	Intento de leer un archivo que no existe
NumberFormatException	Intento de “parsear” o traducir un String que tiene formato ilegal, a un numero
ArithmeticException	Como vimos, tratar de dividir entre cero (para enteros)
SecurityException	Intento de un Applet de ejecutar alguna operación riesgosa
InterruptedException	No es propiamente una condición riesgosa, sino que el usuario interrumpió el método *sleep* mediante alguna combinación de teclas

Referencias

“Mastering Java 2.0”
Sun Microsystems