Captura de Excepciones

Anterior | Siguiente

Las excepciones lanzadas por un método que pueda hacerlo deben recoger en bloque `try/catch` o `try/finally`.

```
int valor;
try {
    for( x=0,valor = 100; x < 100; x ++ )
        valor /= x;
    }
catch( ArithmeticException e ) {
    System.out.println( "Matemáticas locas!" );
    }
catch( Exception e ) {
    System.out.println( "Se ha producido un error" );
    }
```

try

Es el bloque de código donde se prevé que se genere una excepción. Es como si dijésemos "intenta estas sentencias y mira a ver si se produce una excepción". El bloque *try* tiene que ir seguido, al menos, por una cláusula `catch` o una cláusula `finally`.

La sintaxis general del bloque *try* consiste en la palabra clave **`try`** y una o más sentencias entre llaves.

```
try {
    // Sentencias Java
    }
```

Puede haber más de una sentencia que genere excepciones, en cuyo caso habría que proporcionar un bloque *try* para cada una de ellas. Algunas sentencias, en especial aquellas que invocan a otros métodos, pueden lanzar, potencialmente, muchos tipos diferentes de excepciones, por lo que un bloque *try* consistente en una sola sentencia requeriría varios controladores de excepciones.

También se puede dar el caso contrario, en que todas las sentencias, o varias de ellas, que puedan lanzar excepciones se encuentren en un único bloque *try*, con lo que habría que asociar múltiples controladores a ese bloque. Aquí la experiencia del programador es la que cuenta y es el propio programador el que debe decidir qué opción tomar en cada caso.

Los controladores de excepciones deben colocarse inmediatamente después del bloque *try*. Si se produce una excepción dentro del bloque *try*, esa excepción será manejada por el controlador que esté asociado con el bloque *try*.

catch

Es el código que se ejecuta cuando se produce la excepción. Es como si dijésemos "*controlo cualquier excepción que coincida con mi argumento*". No hay código alguno entre un bloque *try* y un bloque *catch*, ni entre bloques *catch*. La sintaxis general de la sentencia **`catch`** en Java es la siguiente:

```
catch( UnTipoTrhowable nombreVariable ) {
    // sentencias Java
    }
```

El argumento de la sentencia declara el tipo de excepción que el controlador, el bloque *catch*, va a manejar.

En este bloque tendremos que asegurarnos de colocar código que no genere excepciones. Se pueden colocar sentencias `catch` sucesivas, cada una controlando una excepción diferente. No debería intentarse capturar todas las excepciones con una sola cláusula, como esta:

```
    catch( Excepcion e ) { ...
```

Esto representaría un uso demasiado general, podrían llegar muchas más excepciones de las esperadas. En este caso es mejor dejar que la excepción se propague hacia arriba y dar un mensaje de error al usuario.

Se pueden controlar grupos de excepciones, es decir, que se pueden controlar, a través del argumento, excepciones semejantes. Por ejemplo:

```
class Limites extends Exception {}
class demasiadoCalor extends Limites {}
class demasiadoFrio extends Limites {}
class demasiadoRapido extends Limites {}
class demasiadoCansado extends Limites {}
    .
    .
    .
    try {
        if( temp > 40 )
            throw( new demasiadoCalor() );
        if( dormir < 8 )
            throw( new demasiado Cansado() );
    } catch( Limites lim ) {
        if( lim instanceof demasiadoCalor ) {
            System.out.println( "Capturada excesivo calor!" );
            return;
            }
        if( lim instanceof demasiadoCansado ) {
            System.out.println( "Capturada excesivo cansancio!" );
            return;
            }
    } finally
        System.out.println( "En la clausula finally" );
```

La cláusula `catch` comprueba los argumentos en el mismo orden en que aparezcan en el programa. Si hay alguno que coincida, se ejecuta el bloque. El operador `instanceof` se utiliza para identificar exactamente cual ha sido la identidad de la excepción.

Cuando se colocan varios controladores de excepción, es decir, varias sentencias `catch`, el orden en que aparecen en el programa es importante, especialmente si alguno de los controladores engloba a otros en el árbol de jerarquía. Se deben colocar primero los controladores que manejen las excepciones más alejadas en el árbol de jerarquía, porque de otro modo, estas excepciones podrían no llegar a tratarse si son recogidas por un controlador más general colocado anteriormente.

Por lo tanto, los controladores de excepciones que se pueden escribir en Java son más o menos especializados, dependiendo del tipo de excepciones que traten. Es decir, se puede escribir un controlador que maneje cualquier clase que herede de **Throwable**; si se escribe para una clase que no tiene subclases, se estará implementando un controlador especializado, ya que solamente podrá manejar excepciones de ese tipo; pero, si se escribe un controlador para una clase *nodo*, que tiene más subclases, se estará implementando un controlador más general, ya que podrá manejar excepciones del tipo de la clase nodo y de sus subclases.

finally

Es el bloque de código que se ejecuta siempre, haya o no excepción. Hay una cierta controversia entre su utilidad, pero, por ejemplo, podría servir para hacer un *log* o un seguimiento de lo que está pasando, porque como se ejecuta siempre puede dejar grabado si se producen excepciones y si el programa se ha recuperado de ellas o no.

Este bloque *finally* puede ser útil cuando no hay ninguna excepción. Es un trozo de código que se ejecuta independientemente de lo que se haga en el bloque *try*.

A la hora de tratar una excepción, se plantea el problema de qué acciones se van a tomar. En la mayoría de los casos, bastará con presentar una indicación de error al usuario y un mensaje avisándolo de que se ha producido un error y que decida si quiere o no continuar con la ejecución del programa.

Por ejemplo, se podría disponer de un diálogo como el que se presenta en el código siguiente:

```
public class DialogoError extends Dialog {
    DialogoError( Frame padre ) {
        super( padre,true );
        setLayout( new BorderLayout() );

        // Presentamos un panel con continuar o salir
        Panel p = new Panel();
        p.add( new Button( "¿Continuar?" ) );
        p.add( new Button( "Salir" ) );

        add( "Center",new Label(
            "Se ha producido un error. ¿Continuar?" ) )
        add( "South",p );
        }

    public boolean action( Event evt,Object obj ) {
        if( "Salir".equals( obj ) ) {
            dispose();
            System.exit( 1 );
            }
        return( false );
        }
    }
```

Y la invocación, desde algún lugar en que se suponga que se generarán errores, podría ser como sigue:

```
try {
    // Código peligroso
    }
catch( AlgunaExcepcion e ) {
    VentanaError = new DialogoError( this );
    VentanaError.show();
    }
```

Lo cierto es que hay autores que indican la inutilidad del bloque *finally*, mientras que desde el Java Tutorial de Sun se justifica plenamente su existencia. El lector deberá revisar todo el material que esté a su alcance y crearse su propia opinión al respecto.

En el programa java904.java, se intenta demostrar el poder del bloque *finally*. En él, un controlador de excepciones intenta terminar la ejecución del programa ejecutando una sentencia `return`. Antes de que la sentencia se ejecute, el control se pasa al bloque *finally* y se ejecutan todas las sentencias de este bloque. Luego el programa termina. Es decir, quedaría demostrado que el bloque *finally* no tiene la última palabra palabra.

El programa redefine el método *getMessage()* de la clase **Throwable**, porque este método devuelve `null` si no es adecuadamente redefinido por la nueva clase excepción.

throw

La sentencia `throw` se utiliza para lanzar explícitamente una excepción. En primer lugar se debe obtener un descriptor de un objeto *Throwable*, bien mediante un parámetro en una cláusula `catch` o, se puede crear utilizando el operador `new`. La forma general de la sentencia **`throw`** es:

```
throw ObjetoThrowable;
```

El flujo de la ejecución se detiene inmediatamente después de la sentencia `throw`, y nunca se llega a la sentencia siguiente. Se inspecciona el bloque *try* que la engloba más cercano, para ver si tiene la cláusula `catch` cuyo tipo coincide con el del objeto o instancia *Thorwable*. Si se encuentra, el control se transfiere a ese sentencia. Si no, se inspecciona el siguiente bloque *try* que la engloba, y así sucesivamente, hasta que el gestor de excepciones más externo detiene el programa y saca por pantalla el trazado de lo que hay en la pila hasta que se alcanzó la sentencia `throw`. En el programa siguiente, java905.java, se demuestra como se hace el lanzamiento de una nueva instancia de una excepción, y también cómo dentro del gestor se vuelve a lanzar la misma excepción al gestor más externo.

```
class java905 {
    static void demoproc() {
        try {
            throw new NullPointerException( "demo" );
        } catch( NullPointerException e ) {
            System.out.println( "Capturada la excepcion en demoproc" );
            throw e;
            }
        }

    public static void main( String args[] ) {
        try {
            demoproc();
        } catch( NullPointerException e ) {
            System.out.println( "Capturada de nuevo: " + e );
            }
        }
    }
```

Este ejemplo dispone de dos oportunidades para tratar el mismo error. Primero, *main()* establece un contexto de excepción y después se llama al método *demoproc()*, que establece otro contexto de gestión de excepciones y lanza inmediatamente una nueva instancia de la excepción. Esta excepción se captura en la línea siguiente. La salida que se obtiene tras la ejecución de esta aplicación es la que se reproduce:

```
% java java905
Capturada la excepcion en demoproc
Capturada de nuevo: java.lang.NullPointerException: demo
```

throws

Si un método es capaz de provocar una excepción que no maneja él mismo, debería especificar este comportamiento, para que todos los métodos que lo llamen puedan colocar protecciones frente a esa excepción. La palabra clave **`throws`** se utiliza para identificar la lista posible de excepciones que un método puede lanzar. Para la mayoría de las subclase de la clase **Exception**, el compilador Java obliga a declarar qué tipos podrá lanzar un método. Si el tipo de excepción es `Error` o `RuntimeException`, o cualquiera de sus subclases, no se aplica esta regla, dado que no se espera que se produzcan como resultado del funcionamiento normal del programa. Si un método lanza explícitamente una instancia de **Exception** o de sus subclases, a excepción de la excepción de *runtime*, se debe declarar su tipo con la sentencia `throws`. La declaración del método sigue ahora la sintaxis siguiente:

```
type NombreMetodo( argumentos ) throws excepciones { }
```

En el ejemplo siguiente, java906.java, el programa intenta lanzar una excepción sin tener código para capturarla, y tampoco utiliza `throws` para declarar que se lanza esta excepción. Por tanto, el código no será posible compilarlo.

```
class java906 {
    static void demoproc() {
        System.out.println( "Capturada la excepcion en demoproc" );
```

```
        throw new IllegalAccessException( "demo" );
        }
    }

    public static void main( String args[] ) {
        demoproc();
    }
```

El error de compilación que se produce es lo suficientemente explícito:

```
% javac java906.java
java906.java:30: Exception java.lang.IllegalAccessException must be caught, or
        it must be declared in the throws clause of this method.
        throw new IllegalAccessException( "demo" );
        ^
```

Para hacer que este código compile, se convierte en el ejemplo siguiente, java907.java, en donde se declara que el método puede lanzar una excepción de *acceso ilegal*, con lo que el problema asciende un nivel más en la jerarquía de llamadas. Ahora *main()* llama a *demoproc()*, que se ha declarado que lanza una `IllegalAccessException`, por lo tanto colocamos un bloque *try* que pueda capturar esa excepción.

```
class java907 {
    static void demoproc() throws IllegalAccessException {
        System.out.println( "Dentro de demoproc" );
        throw new IllegalAccessException( "demo" );
        }

    public static void main( String args[] ) {
        try {
            demoproc();
        } catch( IllegalAccessException e ) {
            System.out.println( "Capturada de nuevo: " + e );
            }
        }
    }
```