Control de Excepciones

Índice:

1. Introducción.

Las aplicaciones pueden generar multitud de errores, ficheros que no existen, impresoras sin papel, una URL inaccesible, una división por cero… Cuando se produce un error, la máquina virtual Java comunica este hecho al programa mediante una excepción. El programa debe capturar la excepción y tratarla, de otra forma terminaría anormalmente.

Una excepción es un objeto que contiene información sobre el acontecimiento producido, y que se transmite al método que le ha llamado o al usuario a través del correspondiente mensaje.

Podemos dividir las excepciones en dos grupos:

- **checked (comprobadas).** Una excepción de este tipo representa un error del cual técnicamente la aplicación puede recuperarse. Por ejemplo, una operación de lectura/escritura en disco puede fallar porque el fichero no exista, porque este se encuentre bloqueado por otra aplicación, etc. Estas situaciones deben ser declaradas y manejadas.
- **unchecked (no comprobadas) o errores de programación.** Un ejemplo claro de este segundo tipo se produce cuando intentamos acceder a una posición inexistente de un array. El array tiene 10 elementos y estamos intentando acceder al elemento 11. Esto provoca una excepción del tipo ArrayIndexOutOfBoundsException. **Estas excepciones no deben ser declaradas ni comprobadas, es el código del programa el que debe estar correctamente escrito para evitar que se produzcan.**

Las aplicaciones deben tener, y en ocasiones obligadas por el compilador, un mecanismo de captura de excepciones. En unos casos simplemente se expulsan y en otros se capturan. Si el que expulsa la excepción es el método main se produce la finalización anormal de la aplicación.

2. Procesar excepciones con try-catch.

Para comprender mejor el funcionamiento de las excepciones estudiaremos el siguiente programa que incluye una división por cero, lo que provocará un error o excepción.

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

    System.out.println(5/0);

}
```

El resultado sería:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at clases.App.main(App.java:8)
```

Lo que ha ocurrido es que la máquina virtual ha detectado un error al dividir por cero y ha creado un objeto (excepción) de la clase java.lang.ArithmeticException. Como el método donde se ha producido la excepción (main) no la ha tratado, la máquina virtual de java muestra el mensaje de error y finaliza la ejecución del programa.

Para procesar excepciones utilizamos la estructura **try-catch** que tiene la siguiente sintaxis:

```
try {
        Bloque de Instrucciones
} catch (TipoExcepción1 nombreVariable) {
        Bloque de Instrucciones del primer catch
} catch (TipoExcepción2 nombreVariable) {
        Bloque de Instrucciones del segundo catch
}...
```

Cualquier excepción que se produzca dentro del bloque try será analizada por el bloque o bloques catch siguientes. En el momento en que se produzca la excepción, se abandona el bloque try y, por lo tanto, las instrucciones que sigan al punto donde se produjo la excepción no serán ejecutadas. La ejecución continúa en el bloque catch correspondiente al tipo de excepción que se haya producido. Estos bloques catch se ensayan en serie. Si la excepción producida no es del primer bloque, se pasa al siguiente y así sucesivamente. Normalmente el último bloque

lleva una excepción genérica de la clase Exception (clase de la que derivan todas las demás excepciones).

Para declarar el tipo de excepción que es capaz de tratar un bloque catch, se utiliza un objeto cuya clase es la clase de la excepción que se desea tratar o una de sus superclases.

Modificando un poco el ejemplo anterior quedaría así:

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

    int dividendo=7;
    int divisor=0;
    try {
        int resultado = dividendo/divisor;
        System.out.println(resultado);
    } catch (ArithmeticException e) {
        System.out.println(e.getMessage());
    }

}
```

Cuando se intenta dividir por cero, la máquina virtual Java genera un objeto de la clase ArithmeticException. Al producirse la excepción dentro de un bloque try, la ejecución del programa se pasa al primer bloque catch. Si la clase de la excepción se corresponde con la clase o alguna subclase de la clase declarada en el bloque catch, se ejecuta el bloque de instrucciones catch y a continuación se pasa el control del programa a la primera instrucción a partir de los bloques try-catch.

El bloque **finally** se utiliza para ejecutar un bloque de instrucciones que se ejecuta sea cual sea la excepción que se produzca. Este bloque se ejecutará, en cualquier caso, incluso si no se produce ninguna excepción. Sirve para no tener que repetir código en el bloque try y en los bloques catch

```
...
try {
Bloque de Instrucciones del try
} catch (TipoExcepción nombreVariable) {
Bloque de Instrucciones del primer catch
} catch (TipoExcepción nombreVariable) {
Bloque de Instrucciones del segundo catch
} .....
}finally {
Bloque de Instrucciones de finally
}
```

3. Expulsar excepciones

Las excepciones pueden ser capturadas y tratadas con try-catch o bien ser lanzadas desde el método donde se producen al método desde el que se hizo la llamada.

El mismo caso de división por cero pero lanzando la excepción hacia arriba.

```java
public class App {

    public static void main(String[] args) {

        int dividendo=7;
        int divisor=0;
        try {
            dividir(dividendo, divisor);
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }

    }
    /*************************************************/
    public static void dividir(int ddo, int dsor) throws ArithmeticException{
        int resultado = ddo/dsor;
        System.out.println(resultado);
    }

}
```

En esta ocasión desde el método main() se llama al método dividir. En la declaración de este último se observa la expresión “throws ArithmeticException”, que quiere decir que si en la ejecución del mismo se produce una excepción de este tipo, como la división por cero, el error no se trata aquí sino que se lanza (throw) al método que hizo la llamada, en nuestro caso main(), y es en este donde se atrapa (try-catch) la excepción. Como se observa la llamada al método dividir() está encerrada en una estructura try-catch.

4. Crear excepciones propias.

El lenguaje Java proporciona las clases que manejan casi cualquier tipo de excepción. Sin embargo, pueden producirse situaciones con excepciones que no están dentro del lenguaje Java. En estos casos, el programador puede crear sus propias excepciones y lanzarlas cuando se considere necesario.

Ilustraremos este tipo de excepciones mediante un ejemplo.

En esta aplicación se solicita un número (entre 1 y 10) al usuario para comprobar si es par o impar. En caso de que el número introducido no esté dentro de ese intervalo, se generará una excepción.

```
public class ExceptionFueraIntervalo extends Exception {

    public ExceptionFueraIntervalo(String mensaje) {
        super(mensaje);
    }

}
```

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

    Scanner entrada = new Scanner(System.in);
    System.out.println("Introduce un valor entre 1 y 10: ");
    int valor = entrada.nextInt();

    try {
        gestionNumero(valor);
    } catch (ExceptionFueraIntervalo e) {
        // TODO Auto-generated catch block
        System.out.println(e.getMessage());
    }

}
/*******************************************************/
public static void gestionNumero(int n) throws ExceptionFueraIntervalo
    if (n<1 || n>10) {
        throw new ExceptionFueraIntervalo("Error en el número");
    } else {
        if (n%2==0) {
            System.out.println("Es par");
        } else {
            System.out.println("Es impar");
        }

    }
}
```

Como se observa, la clase ExcepcionFueraIntervalo es una excepción propia. Se instancia un objeto de esta clase dentro del método “gestionNumero”, en caso de que el valor del parámetro esté fuera del rango [1,10].

Observa el uso de la palabra reservada **throw, con s al final y sin ella.**