Tema 11: Control de excepciones

¿Qué aprenderás?

- Gestionar los errores que se pueden producir cuando se ejecutan las aplicaciones.
- Crear tus propios tipos de errores.
- Dar diferentes soluciones a los errores de ejecución de la aplicación según dónde se produzcan y se capturen.

¿Sabías que…?

- La mayoría de los errores en un programa se originan en las primeras fases del desarrollo. Conforme avanza el proyecto, los errores suelen ser más costosos de arreglar.
- El mecanismo de control de excepciones de Java se amplió considerablemente con el lanzamiento del JDK 7 con las siguientes mejoras: (1) gestión automática de liberación de recursos, como archivos cuando dejan de ser necesarios; (2) la captura múltiple mediante la cual un mismo catch puede capturar dos o más excepciones; (3) regeneración más precisa.
- “El testing puede probar la presencia de errores, pero no la ausencia de ellos” (Edsger Dijkstra).

1. Introducción

Durante la ejecución de nuestros programas se pueden producir diversos errores, que deberemos controlar para evitar fallos. Algunos ejemplos de estos errores en tiempo de ejecución son:

- El usuario introduce datos erróneos (letras cuando se esperan números).
- Se intenta realizar una división entre cero.
- Se quiere abrir un archivo que no existe.

Todas estas situaciones se pueden controlar para que el programa no se cierre de manera inesperada, y actúe de la forma que nosotros le digamos.

Los errores antes comentados son lo que se conoce como excepciones en Java.

Java clasifica las excepciones en dos grupos:

- Error: producidos por el sistema y de difícil tratamiento.
- Exception: errores que se pueden llegar a dar por diversas situaciones, y que se pueden controlar y solucionar (formato numérico incorrecto, ausencia de fichero, error matemático, etc.).

2. Captura de excepciones

En nuestros programas debemos identificar situaciones en que se puedan producir errores, con el fin de desviar el flujo de ejecución y realizar las operaciones necesarias para solucionar el problema.

Esto se consigue con los bloques try/catch. La sintaxis es la siguiente:

```
try{
        /*Instrucciones que pueden producir un error*/
}
catch(TipoExcepcion e){
        /*Instrucciones a ejecutar si se da algún error en alguna de
        las instrucciones que hay en try de tipo TipoExcepcion*/
}
finally{
        /*Instrucciones que se ejecutarán siempre*/
}
```

Sintaxis de los bloques try-catch-finally.

En el momento en que se produjera un error de tipo TipoExcepcion en alguna de las instrucciones del bloque try, pasaría a ejecutarse el bloque catch.

Si no hay error en las instrucciones del bloque try, las del bloque catch no se ejecutan.

Por cada bloque try podemos poner varios bloques catch asociados. Así podremos dar diferente trato a ciertos tipos de excepciones, ejecutando código distinto. También podemos poner un único catch que capture todos los tipos de excepciones, usando la clase Exception, que es la clase de la que heredan el resto de excepciones.

En caso de poner varios bloques catch, el que capture la clase Exception deberá ir en último lugar, ya que, de otra forma, no se llegarían a ejecutar nunca las excepciones tratadas en los bloques catch siguientes.

Al final de los bloques catch podemos poner opcionalmente un bloque finally. En él incluimos instrucciones que queremos que se ejecuten siempre, independientemente de si se ha producido un error o no.

Veamos un programa de ejemplo que usa el bloque try-catch vista antes:

```
import java.io.*;
public class Excepciones {
    public static void main(String[] args) {
        int dividendo, divisor, result;
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        try{
            System.out.println("Introduce el dividendo de la división: ");
            dividendo = Integer.parseInt(br.readLine());
            System.out.println("Introduce el divisor de la división: ");
            divisor = Integer.parseInt(br.readLine());
            result = dividendo/divisor;
            System.out.println("La división" + dividendo + " entre " + divisor + " da " + result);
        }
        catch(ArithmeticException e){
            System.out.println("No se puede dividir entre 0");
        }
        catch(NumberFormatException e){
            System.out.println("Has insertado letras en vez de números");
        }
        catch(Exception e){
            System.out.println("Se ha producido un error inesperado.");
        }
        finally{
            System.out.println("Gracias por utilizar este programa.");
        }
    }
```

Ejemplo uso try-catch-finally.

Es importante poner el bloque catch correspondiente al tipo general Exception en último lugar. Así nos aseguramos que se ejecutarán los bloques anteriores en caso de error (siempre que el tipo de error corresponda con el de algún bloque catch).

Otra forma de capturar excepciones es poner las palabras throws Exception en la cabecera de un método (en el main por ejemplo). De esta forma no necesitamos poner los bloques try-catch, pero cuando se produzca un error éste no será capturado, y no podremos tratarlo como deseemos.

Un método puede lanzar más de una excepción:

```
public static void main(String args[]) throws IOException, NumberFormatException, …
```

Cabecera de función que puede lanzar varios tipos de excepciones

3. Creación de excepciones propias

Java nos proporciona una estructura de clases para tratar diversos tipos de excepciones. Exception es la superclase que gestiona cualquier tipo de excepción. De ella heredarán sus métodos un gran número de subclases encargadas en la gestión de excepciones específicas.

El programador puede crear sus propias clases de excepciones para tratar errores específicos que Java no contempla. Para ello hay que usar el concepto de herencia sobre la clase Exception. Normalmente, la nueva clase contendrá un único constructor con un parámetro de tipo String, que será el mensaje de error que queremos mostrar. Este mensaje lo mostraremos con el método getMessage() de la clase Exception.

Veamos un ejemplo de programa que crea y lanza excepciones propias:

```
public class MiExcepcion extends Exception{
    public MiExcepcion (String mensaje){
            super(mensaje);
    }
}
```

```
import java.io.*;
public class Main {
    public static void main(String[] args) {
            int nota;
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            try{
                System.out.println("Introduce una nota entre 0 y 10:");
                nota = Integer.parseInt(br.readLine());
                if(nota<0 || nota>10){
                        throw new MiExcepcion("Nota fuera de rango");
                }
            }
            catch(MiExcepcion e){
                System.out.println(e.getMessage());
            }
            catch(Exception e){
                System.out.println(e.getMessage());
            }
    }
}
```

Ejemplo de creación y uso de una excepción propia

4. Propagación de excepciones

Cuando se produce una excepción dentro de un método, se puede capturar dentro de éste y tratarlo. De no hacerse, la excepción se propaga automáticamente al método que lo llamó, y así sucesivamente hasta que un método captura la excepción o llegamos al main.

IMPORTANTE: los errores de la clase Exception no se propagan.

Observemos el siguiente ejemplo. ¿Qué ocurre cuando se produce un error de tipo NumberFormatException y no lo capturamos en el método leerNumero? Saldrá el mensaje de error que tenemos en el bloque catch en el main.

```
public static void main(String[] args) {
    int num;
    try{
            System.out.println("Introduce un número: ");
            num = leerNumero();
            System.out.println("El número es " + num);
    }
    catch(NumberFormatException e){
            System.out.println("Has insertado letras en vez de números");
    }
}
static int leerNumero(){
    int numero = 0;
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    try{
            numero = Integer.parseInt(br.readLine());
    }
    catch(IOException e){
            System.out.println("ERROR");
    }
    return numero;
}
```

Código de ejemplo de propagación de excepciones

¿Qué ocurrirá ahora si capturamos la excepción en el método leerNumero? En este caso no debe ejecutarse el bloque catch del main, por tanto, se ejecutará todo el código del try, incluido el System.out.println. Se mostrará un mensaje cuando no debería hacerse.

```
public static void main(String[] args) {
    int num;
    try{
            System.out.println("Introduce un número: ");
            num = leerNumero();
            System.out.println("El número es " + num);
    }
    catch(NumberFormatException e){
            System.out.println("Has insertado letras en vez de números");
    }
}
static int leerNumero(){
    int numero = 0;
    InputStreamReader isr = new InputStreamReader(System.in);
    BufferedReader br = new BufferedReader(isr);
    try{
            numero = Integer.parseInt(br.readLine());
    }
    catch(Exception e){
            //CAPTURAMOS EN ESTA FUNCIÓN CUALQUIER TIPO DE EXCEPCIÓN
            System.out.println("ERROR");
    }
    return numero;
}
```

Código de ejemplo de propagación de excepciones (II)

Con estos ejemplos vemos la utilidad de no capturar el error dentro del método donde se produce, sino dejarlo que se propague, debido a la necesidad de tener que controlarlo fuera.

Recursos y enlaces

- Clase Exception Java 10:
  https://docs.oracle.com/javase/10/docs/api/java/lang/Exception.html

- Clase Error Java 10:
  https://docs.oracle.com/javase/10/docs/api/java/lang/Error.html

Conceptos clave

- **Exception**: clase de Java que se encarga de la gestión de los errores en tiempo de ejecución que se pueden producir en los programas.
- **Propagación de excepciones**: cuando se produce una excepción y no se captura, ésta se propagará hacia el método que hizo la llamada. Esto se hará hasta llegar a un método que capture la excepción, o hasta el main.

Test de autoevaluación

Al crear excepciones propias, ¿qué forma tendrá el constructor?

a) No tendrá constructor, se usa el de la clase Exception
b) Un parámetro de tipo String y una llamada a la función getMessage de la clase Exception
c) Una llamada a la función System.out.println con el texto que queramos. No recibe ningún parámetro
d) Un parámetro de tipo String y una llamada a la función System.out.println para mostrar el String parámetro

¿Qué ocurre si no se captura una excepción en un método?

a) Que se propaga al método que lo llamó
b) Que se produce un error de compilación
c) Que ese tipo de excepción no se contempla en el programa
d) Que al ejecutarse, el método que lo llama da un error en tiempo de ejecución

Ponlo en práctica

Actividad 1

Crear un programa que pida los datos de dos alumnos, que son: nombre (letras), edad (entero) y altura (decimal). Se debe realizar un control de la entrada de datos, de tal forma que, si el usuario introduce números en el nombre, o letras en la edad o la altura, se vuelva a pedir el dato correspondiente hasta que la entrada sea correcta.

NOTA: puedes usar el método “matches” de la clase String para ver si una cadena de caracteres posee números o no.

Actividad 2

Modificar el programa anterior de manera que, si el usuario comete más de 5 errores al introducir datos, se muestre un mensaje informando de este hecho y se cierre el programa.

SOLUCIONARIOS

Test de autoevaluación

Al crear excepciones propias, ¿qué forma tendrá el constructor?

a) No tendrá constructor, se usa el de la clase Exception

b) Un parámetro de tipo String y una llamada a la función getMessage de la clase Exception

c) Una llamada a la función System.out.println con el texto que queramos. No recibe ningún parámetro

d) Un parámetro de tipo String y una llamada a la función System.out.println para mostrar el String parámetro

¿Qué ocurre si no se captura una excepción en un método?

e) Que se propaga al método que lo llamó

f) Que se produce un error de compilación

g) Que ese tipo de excepción no se contempla en el programa

h) Que, al ejecutarse, el método que lo llama da un error en tiempo de ejecución

Ponlo en práctica

Actividad 1

Crear un programa que pida los datos de dos alumnos, que son: nombre (letras), edad (entero) y altura (decimal). Se debe realizar un control de la entrada de datos, de tal forma que, si el usuario introduce números en el nombre, o letras en la edad o la altura, se vuelva a pedir el dato correspondiente hasta que la entrada sea correcta.

NOTA: puedes usar el método “matches” de la clase String para ver si una cadena de caracteres posee números o no.

El solucionario está disponible en la versión interactiva del aula.

Actividad 2

Modificar el programa anterior de manera que, si el usuario comete más de 5 errores al introducir datos, se muestre un mensaje informando de este hecho y se cierre el programa.

El solucionario está disponible en la versión interactiva del aula.