

UNIDAD 7.

EXCEPCIONES PERSONALIZADAS

Programación
CFGS DAM /DAW

Autores:

Carlos Cacho y Raquel Torres

Revisado por:

Lionel Tarazón – lionel.tarazon@ceedcv.es

Encarni Serrano – encarni.serrano@ceedcv.es


Licencia




Reconocimiento - NoComercial - CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

 Interesante

ÍNDICE DE CONTENIDO

1. Introducción.....	4
1.1 El objeto Exception.....	5
2. Jerarquía y tipos de excepciones Java.....	6
3. Definir excepciones personalizadas.....	8
3.1 Ejemplo.....	8
4. Agradecimientos.....	9

UD7. EXCEPCIONES PERSONALIZADAS.

1. INTRODUCCIÓN

Una excepción es un error semántico que se produce en tiempo de ejecución. Aunque un código sea correcto sintácticamente (es código Java válido y puede compilarse), es posible que durante su ejecución se produzcan errores inesperados, como por ejemplo:

- Dividir por cero.
- Intentar acceder a una posición de un array fuera de sus límites.
- Al llamar al `nextInt()` de un `Scanner` el usuario no introduce un valor entero.
- Intentar acceder a un fichero que no existe o que está en un disco duro corrupto.
- Etc.

Cuando esto ocurre, la máquina virtual Java crea un objeto de la clase ***Exception*** (las excepciones en Java son objetos) y se notifica el hecho al sistema de ejecución. Se dice que se ha lanzado una excepción ("***Throwing Exception***"). Existen también los errores internos que son objetos de la clase ***Error*** que no estudiaremos. Ambas clases ***Error*** y ***Exception*** son clases derivadas de la clase base ***Throwable***.

Un método se dice que es capaz de tratar una excepción ("***Catch Exception***") si ha previsto el error que se ha producido y las operaciones a realizar para "recuperar" el programa de ese estado de error. No es suficiente capturar la excepción, si el error no se trata tan solo conseguiremos que el programa no se pare, pero el error puede provocar que los datos o la ejecución no sean correctos.

En el momento en que es lanzada una excepción, la máquina virtual Java recorre la pila de llamadas de métodos en busca de alguno que sea capaz de tratar la clase de excepción lanzada. Para ello, comienza examinando el método donde se ha producido la excepción; si este método no es capaz de tratarla, examina el método desde el que se realizó la llamada al método donde se produjo la excepción y así sucesivamente hasta llegar al último de ellos. En caso de que ninguno de los métodos de la pila sea capaz de tratar la excepción, la máquina virtual Java muestra un mensaje de error y el programa termina.

Los programas escritos en Java también pueden lanzar excepciones explícitamente mediante la instrucción ***throw***, lo que facilita la devolución de un "código de error" al método que invocó el método que causó el error.

1.1 El objeto Exception

Toda excepción genera un objeto de la clase Exception (o uno más específico que hereda de Exception). Dicho objeto contendrá detalles sobre el error producido. Puede ser interesante mostrar esta información para que la vea el usuario (que sepa qué ha sucedido) o el desarrollador (para depurar y corregir el código si es pertinente). En la cláusula catch tenemos acceso al objeto en caso de que queramos utilizarlo:

Los dos métodos de Exception más útiles son:

- **getMessage()** → Devuelve un String con un texto simple sobre el error.
- **printStackTrace()** → Es el que más información proporciona. Indica qué tipo de Excepción se ha producido, el mensaje simple, y también toda la pila de llamadas. Esto es lo que hace Java por defecto cuando una excepción no se maneja y acaba parando el programa.

```
...
catch (Exception e){

    // Mostramos el mensaje de la excepción
    System.err.println("Error: " + e.getMessage());

    // Mostramos toda la información, mensaje y pila de llamadas
    e.printStackTrace();
}
...
```

Los objetos de tipo Exception tienen sobrecargado el método toString() por lo que también es posible imprimirlos directamente mediante println().

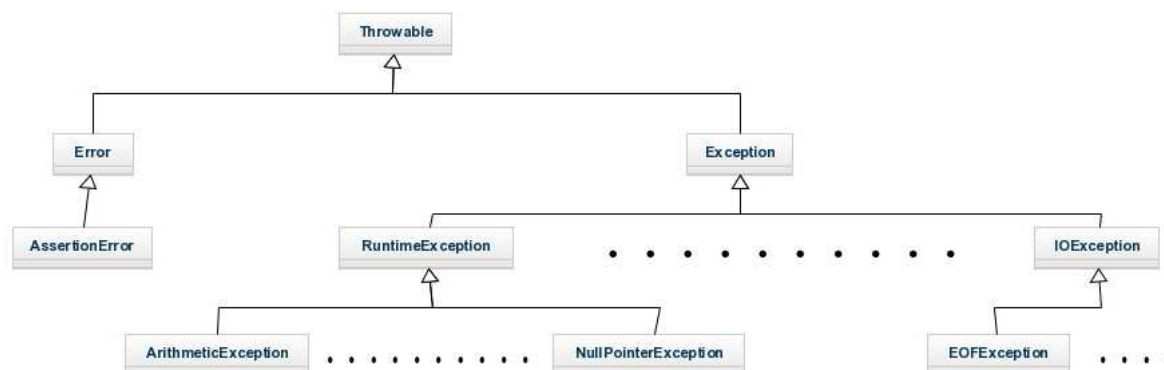
```
...
catch (Exception e){

    // Mostramos el mensaje de la excepción
    System.out.println(e);

}
...
```

2. JERARQUÍA Y TIPOS DE EXCEPCIONES JAVA

La clase `Exception` hereda de `Throwable`, y a su vez, todas las excepciones heredan de `Exception`.



Como *java.lang* es importado de forma implícita en todos los programas, la mayor parte de las excepciones derivadas de *RuntimeException* están disponibles de forma automática. Además **no es necesario incluirlas en ninguna cabecera de método mediante *throws***.

Las Excepciones pueden ser comprobadas y no comprobadas:

- **Excepciones comprobadas:** aquellas que Java comprueba durante la compilación, antes de la ejecución del programa.
- **Excepciones no comprobadas:** aquellas que Java no puede comprobar durante la compilación y se producirán durante la ejecución del programa.

Las **excepciones comprobadas** definidas en *java.lang* son:

Excepción	Significado
<i>ClassNotFoundException</i>	No se ha encontrado la clase.
<i>CloneNotSupportedException</i>	Intento de duplicado de un objeto que no implementa la interfaz clonable.
<i>IllegalAccessException</i>	Se ha denegado el acceso a una clase.
<i>InstantiationException</i>	Intento de crear un objeto de una clase abstracta o interfaz.
<i>InterruptedException</i>	Hilo interrumpido por otro hilo.
<i>NoSuchFieldException</i>	El campo solicitado no existe.
<i>NoSuchMethodException</i>	El método solicitado no existe.

Las **subclases de *RuntimeException* no comprobadas** son:

Excepción	Significado
<i>AithmeticException</i>	Error aritmético como división entre cero.
<i>ArrayIndexOutOfBoundsException</i>	Índice de la matriz fuera de su límite.
<i>ArrayStoreException</i>	Asignación a una matriz de tipo incompatible.
<i>ClassCastException</i>	Conversión inválida.
<i>IllegalArgumentException</i>	Uso inválido de un argumento al llamar a un método.
<i>IllegalMonitorStateException</i>	Operación de monitor inválida, como esperar un hilo no bloqueado.
<i>IllegalStateException</i>	El entorno o aplicación están en un estado incorrecto.
<i>IllegalThreadStateException</i>	La operación solicitada es incompatible con el estado actual del hilo.
<i>IndexOutOfBoundsException</i>	Algún tipo de índice está fuera de su rango o de su límite.
<i>NegativeArraySizeException</i>	La matriz tiene un tamaño negativo.
<i>NullPointerException</i>	Uso incorrecto de una referencia NULL.
<i>NumberFormatException</i>	Conversión incorrecta de una cadena a un formato numérico.
<i>SecurityException</i>	Intento de violación de seguridad.
<i>StringIndexOutOfBounds</i>	Intento de sobrepasar el límite de una cadena.
<i>TypeNotPresentException</i>	Tipo no encontrado.
<i>UnsupportedOperationException</i>	Operación no admitida.



Es interesante conocer los distintos tipos de excepciones, pero **no es necesario sabérselas de memoria ni entender exactamente cuándo se produce cada una.**

Aunque las excepciones que incorpora Java, gestionan la mayoría de los errores más comunes, es probable que el programador prefiera crear sus propios tipos de excepciones para gestionar situaciones específicas de sus aplicaciones. Tan solo hay que definir una subclase de *Exception*, que naturalmente es subclase de *Throwable*.

3. DEFINIR EXCEPCIONES PERSONALIZADAS.

Cuando desarrollamos software, sobre todo al desarrollar nuestras propias clases, es habitual que se puedan producir excepciones que no estén definidas dentro del lenguaje Java. Para crear una excepción propia tenemos que definir una clase derivada de la clase base *Exception*.

3.1 Ejemplo.

Vamos a ver un ejemplo sencillo de definición y uso de un nuevo tipo de excepción llamada *ExcepcionPropia* que utilizaremos cuando a se le pase a 'método' un valor superior a 10.

El main y el método que lanza la excepción:

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         try
18         {
19             metodo(1);
20             metodo(20);
21         }
22         catch (ExcepcionPropia e)
23         {
24             System.out.println("capturada :" + e);
25         }
26     }
27
28     static void metodo(int n) throws ExcepcionPropia
29     {
30         System.out.println("Llamado por metodo(" + n + ")");
31
32         if (n > 10)
33             throw new ExcepcionPropia(n);
34
35         System.out.println("Finalización normal");
36     }
37 }
```


La definición de la nueva excepción.

```
12 public class ExcepcionPropia extends Exception
13 {
14     private int num;
15
16     ExcepcionPropia(int n)
17     {
18         this.num = n;
19     }
20     public String toString()
21     {
22         return "Excepcion Propia[" + this.num + "]";
23     }
24 }
25 }
```

Siendo la salida:

```
run:
Llamado por metodo(1)
Finalización normal
Llamado por metodo(20)
capturada :Excepcion Propia[20]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Como se puede observar la excepción se lanza cuando el número es mayor que 10 y será tratada por la nueva clase creada que hereda de Exception. Además se sobrescribe el método toString que es el encargado de mostrar el mensaje asociado a la excepción.

4. AGRADECIMIENTOS

Apuntes actualizados y adaptados al CEEDCV a partir de la siguiente documentación:

[1] Apuntes Programación de José Antonio Díaz-Alejo. IES Camp de Morvedre.