

TEMA10. EXCEPCIONES

Programación
CFGS DAW

Profesores:

Carlos Cacho

Raquel Torres

carlos.cacho@ceedcv.es

raquel.torres@ceedcv.es

Versión:180309.1403

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
2.Lanzamiento (<i>Throw</i>).....	5
3.Tratamiento.....	5
3.1 El bloque <i>try</i>	6
3.2 El bloque <i>catch</i>	6
3.3 Cláusulas <i>catch</i> múltiples.....	7
3.4 Descripción de una excepción.....	7
3.5 El bloque <i>finally</i>	7
4.Jerarquía de excepciones.....	8
5.Excepciones de Java.....	9
6.Excepciones propias.....	12
7.Ejemplos.....	12
7.1 Ejemplo 1.....	12
7.2 Ejemplo 2.....	13
7.3 Ejemplo 3.....	13
7.4 Ejemplo 4.....	14
7.5 Ejemplo 5.....	15
7.6 Ejemplo 6.....	16
7.7 Ejemplo 7.....	17
8.Agradecimientos.....	19

UD10. EXCEPCIONES

1. INTRODUCCIÓN

Cuando un programa Java viola las restricciones semánticas del lenguaje (se produce un error), la máquina virtual Java comunica este hecho al programa mediante una **excepción**.

Muchas clases de errores pueden provocar una excepción:

- Desde un desbordamiento de memoria o un disco duro estropeado, hasta un usb protegido contra escritura.
- Un intento de dividir por cero.
- Intentar acceder a un vector fuera de sus límites.
-

Cuando esto ocurre, la máquina virtual Java crea un objeto de la clase **Exception** (las excepciones en Java son objetos de clases derivadas de la clase base **Exception**) 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 se ha previsto también 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.

Existe toda una jerarquía de clases derivada de la clase base *Exception*. Estas clases derivadas se ubican en dos grupos principales:

- Las **excepciones en tiempo de ejecución** que ocurren cuando el programador no ha tenido cuidado al escribir su código. Por ejemplo, cuando se sobrepasa la dimensión de un *array* se lanza una excepción *ArrayIndexOutOfBoundsException* o cuando se hace uso de una referencia a un objeto que no ha sido creado se lanza la excepción *NullPointerException*. Estas excepciones le indican al programador qué tipos de fallo tiene el programa y que debe arreglarlo antes de proseguir.
- El segundo grupo de **excepciones**, es el más interesante, ya **que indican** que ha sucedido **algo inesperado o fuera de control**.

Ver [ejemplo 1](#)

Ver [ejemplo 2](#)

Ver [ejemplo 3](#)

2. LANZAMIENTO (*THROW*)

Como se ha comentado anteriormente, un método también es capaz de lanzar excepciones.

En primer lugar, es necesario declarar todas las posibles excepciones que es posible generar en el método, utilizando la cláusula *throws* de la declaración de métodos.


Para lanzar la excepción es necesario crear un objeto de tipo *Exception* o alguna de sus subclases (por ejemplo *ArithmeticException*) y lanzarlo mediante la instrucción *throw*.

Ver [ejemplo 4](#)

3. TRATAMIENTO

En Java, se pueden tratar las excepciones previstas por el programador utilizando unos mecanismos, los **manejadores de excepciones**, que se estructuran en tres bloques:

- El bloque **try**.
- El bloque **catch**.
- El bloque **finally**.

 Un **manejador de excepciones** es una porción de código que se va a encargar de tratar las posibles excepciones que se puedan generar.

3.1 El bloque **try**

Lo primero que hay que hacer para que un método sea capaz de tratar una excepción generada por la máquina virtual Java, o por el propio programa mediante una instrucción *throw*, es encerrar las instrucciones susceptibles de generarla en un bloque *try*.

```
try {  
    Bloque_De_Instrucciones  
}
```

Cualquier excepción que se produzca dentro del bloque *try* será analizado por el bloque o bloques *catch* que se verá en el punto siguiente.

⚡ 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.

⚡ Cada bloque **try** debe tener asociado por lo menos un bloque **catch**.

3.2 El bloque **catch**

Por cada bloque *try* pueden declararse uno o varios bloques *catch*, cada uno de ellos capaz de tratar un tipo u otro de excepción.

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

Su estructura es la siguiente:

```
try {  
    Bloque_De_Instrucciones  
}
```

```
catch (TipoExcepción nombreVariable) {  
    Bloque_Catch  
}  
catch (TipoExcepción nombreVariable) {  
    Bloque_Catch  
} ...
```

Ver [ejemplo 5](#)

3.3 Cláusulas *catch* múltiples

Como se ha comentado antes, se pueden especificar varias cláusulas *catch*, para que cada una de ellas capture un tipo diferente de excepción. Cuando se lanza una excepción, se inspecciona cada sentencia *catch* por orden, y se ejecuta la primera cuyo tipo coincida con la excepción. Después de que se ejecute una sentencia *catch*, las demás se evitan y la ejecución continúa después del bloque *try/catch*.

⚡ Cuando se utilizan varias sentencias *catch*, es importante recordar que las subclases de excepción deberán ir delante de cualquiera de sus superclases.

Esto es así porque la sentencia *catch* que utiliza un superclase, capturará excepciones de este tipo más cualquiera de sus subclases, y por lo tanto, éstas no se ejecutarán si están después de la superclase.

Ver [ejemplo 6](#)

3.4 Descripción de una excepción

La clase *throwable* sobrescribe el método *toString()*, definido por *Object*, y devuelve una cadena que contiene la descripción de la excepción. La excepción puede presentarse en una sentencia *println()*, pasando simplemente la excepción como argumento. Por ejemplo:

```
...  
catch (AritmeticException e){  
    System.out.println("Excepción :" + e);  
}  
...
```

3.5 El bloque *finally*

✈ El bloque ***finally*** se utiliza para ejecutar un bloque de instrucciones 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*. La estructura es la siguiente:

```
try {  
    Bloque_De_Instrucciones  
}  
catch (TipoExcepción nombreVariable) {  
    Bloque_Catch  
}  
catch (TipoExcepción nombreVariable) {  
    Bloque_Catch  
}  
...  
}  
catch (TipoExcepción nombreVariable) {  
    Bloque_Catch  
}  
finally {  
    Bloque_Finally  
}
```

4. JERARQUÍA DE EXCEPCIONES

📖 Las excepciones son objetos pertenecientes a la clase *Throwable* o alguna de sus subclases.

Dependiendo del lugar donde se produzcan, existen dos tipos de excepciones:

- Las **excepciones síncronas** no son lanzadas en un punto arbitrario del

programa sino que, en cierta forma, son previsibles en determinados puntos del programa como resultado de evaluar ciertas expresiones o la invocación de determinadas instrucciones o métodos.

- Las **excepciones asíncronas** pueden producirse en cualquier parte del programa y no son tan "previsibles". Pueden producirse excepciones asíncronas debido a dos razones:
 1. La invocación del método *stop()* de la clase *Thread* que se está ejecutando.
 2. Un error interno en la máquina virtual Java.

Dependiendo de si el compilador comprueba o no que se declare un manejador para tratar las excepciones, se pueden dividir en:

- Las **excepciones comprobables** son repasadas por el compilador Java durante el proceso de compilación, de forma que si no existe un manejador que las trate, generará un mensaje de error.
- Las **excepciones no comprobables** son la clase *RuntimeException* y sus subclases junto con la clase *Error* y sus subclases.

También pueden definirse por el programador subclases de las excepciones anteriores. Las más interesantes desde el punto de vista del programador son las subclases de la superclase *Exception* ya que éstas pueden ser comprobadas por el compilador.

Las **excepciones predefinidas más frecuentes** son:

- ***ArithmeticException***: Las excepciones aritméticas son típicamente el resultado de división por 0.
- ***NullPointerException***: Se produce cuando se intenta acceder a una variable o método antes de ser definido.
- ***IncompatibleClassChangeException***: El intento de cambiar una clase afectada por referencias en otros objetos, específicamente cuando esos objetos todavía no han sido recompilados.
- ***ClassCastException***: El intento de convertir un objeto a otra clase que no es válida.
- ***NegativeArraySizeException***: Puede ocurrir si hay un error aritmético al cambiar el tamaño de un array.
- ***OutOfMemoryException***: ¡No debería producirse nunca! El intento de crear un objeto con el operador *new* ha fallado por falta de memoria. Siempre tendría que haber memoria suficiente porque el *garbage collector* se encarga de proporcionarla al ir liberando objetos que no se usan y devolviendo memoria al sistema.
- ***NoClassDefFoundException***: Se referenció a una clase que el sistema es incapaz de encontrar.
- ***ArrayIndexOutOfBoundsException***: Es la excepción que más

frecuentemente se produce. Se genera al intentar acceder a un elemento de un *array* más allá de los límites definidos inicialmente para ese *array*.

- **InternalException:** Este error se reserva para eventos que no deberían ocurrir. Por definición, el usuario nunca debería ver este error y esta excepción no debería lanzarse. El compilador Java obliga al programador a proporcionar el código de manejo o control de algunas de las excepciones predefinidas por el lenguaje.

5. EXCEPCIONES DE JAVA

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, en la lista de *throws*.

Hemos dicho que las excepciones no comprobadas son las que en tiempo de compilación no se comprueba si el método gestiona o lanza estas excepciones.

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

Excepción	Significado
<i>AritmeticException</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 invalida.
<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>StringIndexOutOfBoundsException</i>	Intento de sobrepasar el límite de una cadena.
<i>TypeNotPresentException</i>	Tipo no encontrado.
<i>UnsupportedOperationException</i>	Operación no admitida.

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.

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*.

En realidad no es necesario que estas subclases que crea el programador implementen nada; simplemente, su presencia en el sistema lo que permitirá es que se puedan utilizar como excepciones.

La clase *Exception* no define ningún método por si misma, pero, los hereda de la superclase *throwable*, por lo tanto todas las excepciones incluidas, las definidas por el programador pueden utilizar los métodos heredados.

Estos métodos que además pueden ser sobrescritos en las clases de excepción propias, se recogen en la siguiente tabla:

Método	Descripción
<i>Throwable fillInStackTrace()</i>	Devuelve un objeto con el trazado completo de la pila.
<i>Throwable getCause()</i>	Devuelve la excepción que suya en

	la excepción actual. Si no hay excepción subyacente devuelve NULL.
<i>String getLocalizedMessage()</i>	Devuelve una descripción localizada de la excepción.
<i>String getMessage()</i>	Devuelve una descripción de la excepción.
<i>Stack TraceElement[]</i>	Devuelve una matriz que contiene el trazado de la pila....
<i>getStackTrace()</i>	
<i>printStackTrace()</i>	
<i>printStackTrace(PrintStream stream)</i>	
<i>String toString</i>	Devuelve una cadena con la descripción de la excepción. Este método es llamado por <i>println()</i> cuando se desea imprimir un objeto de la clase <i>Throwable</i> .

6. EXCEPCIONES PROPIAS

Java proporciona las clases que manejan casi cualquier tipo de excepción. Sin embargo, podemos imaginar situaciones en la que se producen excepciones que no están dentro del lenguaje Java.

Para crear y lanzar una excepción propia tenemos que definir una clase derivada de la clase base *Exception*.

El uso de esta nueva excepción es el mismo que hemos visto.

Ver [ejemplo 7](#).

7. EJEMPLOS

7.1 Ejemplo 1

Como primer encuentro con las excepciones, vamos a ejecutar el siguiente programa, en él vamos a forzar una excepción al intentar dividir un número entre 0:

```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int div, x, y;
16
17         x = 3;
18         y = 0;
19
20         div = x / y;
21
22         System.out.println("El resultado es " + div);
23     }
24
25 }
```

Siendo la salida:

```
run:
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:20)
    C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

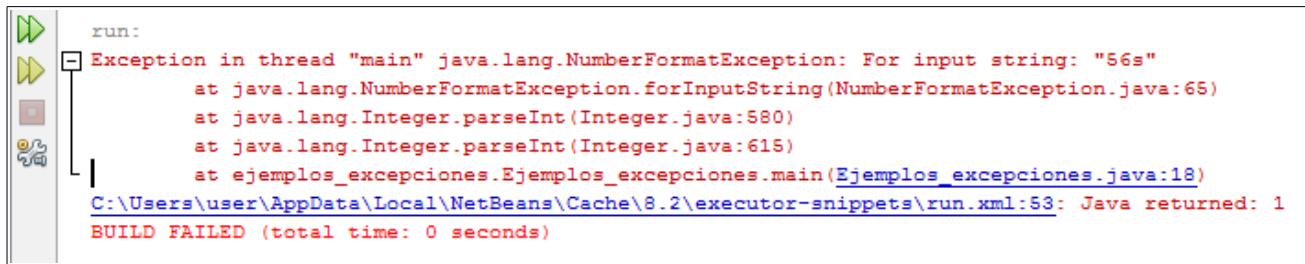
Lo que ha ocurrido es que la máquina virtual Java ha detectado una condición de error, la división por 0, y ha creado un objeto de la clase *java.lang.ArithmeticException*. Como el método donde se ha producido la excepción no es capaz de tratarla, es tratada por la máquina virtual Java, que muestra el mensaje de error anterior y finaliza la ejecución del programa.

7.2 Ejemplo 2

A continuación vamos a forzar una excepción de conversión, para ello vamos a intentar pasar a entero una cadena que no sólo lleva caracteres numéricos:

```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         String cadena = "56s";
16         int num;
17
18         num = Integer.parseInt(cadena);
19
20         System.out.println("El número es " + num);
21     }
22
23 }
```

Siendo la salida:



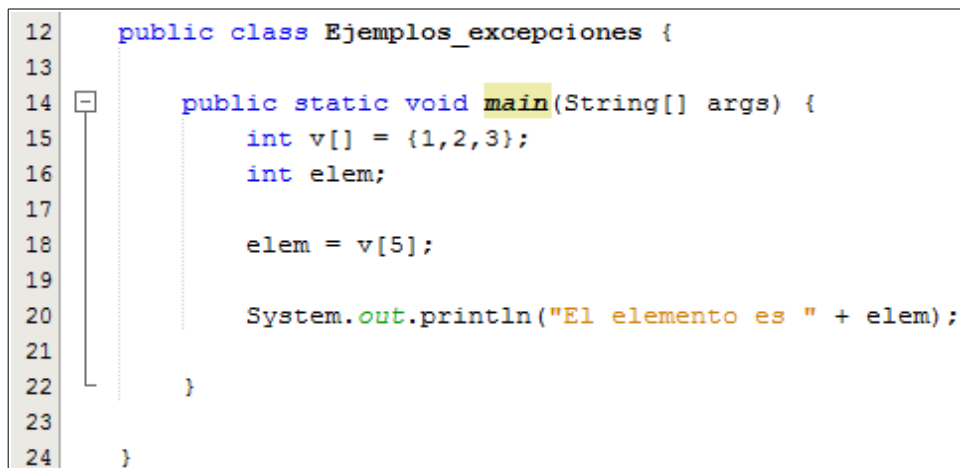
```
run:
Exception in thread "main" java.lang.NumberFormatException: For input string: "56s"
    at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    at java.lang.Integer.parseInt(Integer.java:580)
    at java.lang.Integer.parseInt(Integer.java:615)
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

Si se introducen caracteres no numéricos, o no se quitan los espacios en blanco al principio y al final del *String*, mediante el método *trim*¹, se lanza una excepción *NumberFormatException*.

El mensaje que aparece en la ventana nos indica el tipo de excepción *NumberFormatException* y el método que la ha lanzado, *Integer.parseInt*, que se llama dentro de *main*.

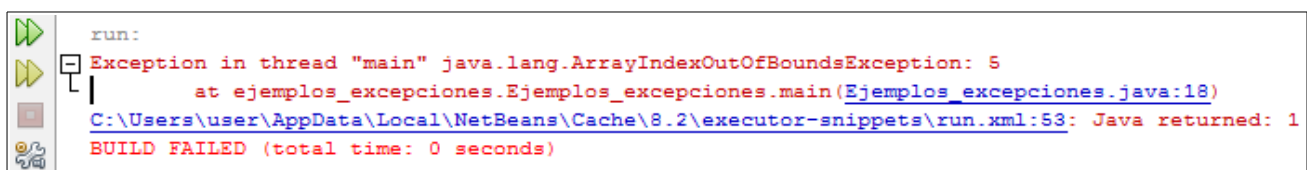
7.3 Ejemplo 3

En este ejemplo vamos a forzar una excepción de límites del vector, para ello vamos a crear un vector e intentar acceder a una posición que no existe:



```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int v[] = {1,2,3};
16         int elem;
17
18         elem = v[5];
19
20         System.out.println("El elemento es " + elem);
21     }
22 }
23
24 }
```

Siendo la salida:



```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

¹ [http://www.w3api.com/wiki/java:String.trim\(\)](http://www.w3api.com/wiki/java:String.trim())

Al intentar acceder a una posición que sobrepasa el tamaño del vector, el tamaño es 3 y la última posición es 2, se lanza la excepción.

[Volver](#)

7.4 Ejemplo 4

En este ejemplo vamos a generar una condición de error si el dividendo de una división es menor que el divisor:

```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) throws ArithmeticException {
15         int x = 1, y = 2;
16
17         if(x / y < 1)
18             throw new ArithmeticException();
19         else
20             System.out.println("El resultado es " + x / y);
21
22     }
23
24 }
```

Siendo la salida:

```
run:
Exception in thread "main" java.lang.ArithmeticException
    at ejemplos_excepciones.Ejemplos_excepciones.main(Ejemplos_excepciones.java:18)
    C:\Users\user\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1
BUILD FAILED (total time: 0 seconds)
```

En este caso hemos indicado mediante la palabra reservada **throws** que el método puede lanzar una excepción del tipo aritmética. Y con la palabra reservada **throw** lanzamos la excepción, creándonos un objeto de tipo **ArithmeticException**.

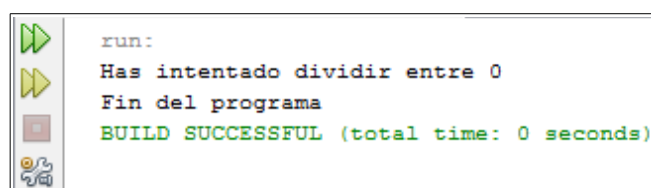
[Volver](#)

7.5 Ejemplo 5

A continuación vamos a capturar una excepción y tratarla haciendo uso de los manejadores *try-catch*:

```
12 public class Ejemplos_excepciones {
13
14     public static void main(String[] args) {
15         int x = 1, y = 0;
16
17         try
18         {
19             int div = x / y;
20
21             System.out.println("La ejecución no llegará aquí.");
22         }
23         catch(ArithmeticException ex)
24         {
25             System.out.println("Has intentado dividir entre 0");
26         }
27
28         System.out.println("Fin del programa");
29     }
30 }
```

Siendo la salida:



```
run:
Has intentado dividir entre 0
Fin del programa
BUILD SUCCESSFUL (total time: 0 seconds)
```

En este caso, 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 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*.

Como puede verse, también se podría haber utilizado en la declaración del bloque *catch*, una superclase de la clase *ArithmeticException*. Por ejemplo: *catch (RuntimeException ex)* o *catch (Exception ex)*. Sin embargo, es mejor utilizar excepciones lo más cercanas al tipo de error previsto, ya que lo que se pretende es recuperar el programa de alguna condición de error y si "se meten todas las condiciones en el mismo saco", seguramente habrá que averiguar después qué condición de error se produjo para poder dar una respuesta adecuada.

El objetivo de la mayoría de cláusulas *catch* bien construidas ha de ser el de resolver la condición excepcional y luego continuar como si el error nunca hubiera ocurrido.

[Volver](#)

7.6 Ejemplo 6

Vamos a ver un ejemplo de cláusulas *catch* múltiples:

```
14 public class Ejemplos_excepciones {
15
16     public static void main(String[] args) {
17         int x, y, div, pos;
18         int[] v = {1,2,3};
19         Scanner in = new Scanner(System.in);
20
21         try
22         {
23             System.out.print("Introduce el numerador: ");
24             x = in.nextInt();
25
26             System.out.print("Introduce el denominador: ");
27             y = in.nextInt();
28
29             div = x / y;
30
31             System.out.println("La división es " + div );
32
33             System.out.print("Introduce la posición del vector a consultar: ");
34             pos = in.nextInt();
35
36             System.out.println("El elemento es " + v[pos] );
37
38         }
39         catch(ArithmeticException ex)
40         {
41             System.out.println("División por cero: " + ex);
42         }
43         catch(ArrayIndexOutOfBoundsException ex)
44         {
45             System.out.println("Sobrepasado el tamaño del vector: " + ex);
46         }
47
48         System.out.println("Fin del programa");
49     }
50 }
```

En este caso estamos tratando las excepciones de división por cero y si sobrepasamos el tamaño del vector.

[Volver](#)

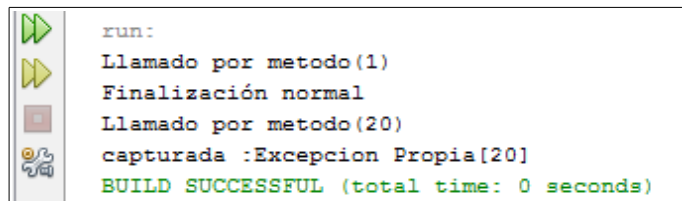
7.7 Ejemplo 7

Vamos a ver un ejemplo de creación y uso de un nuevo tipo de excepción, de tal forma que saltará si el número pasado al método es mayor que 10:

```
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 }
```

```
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.

8. 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.