
TEMA 3: Excepciones

Programación II - 2017/2018

Pedro Cuesta Morales, Baltasar García Pérez-Schofield,
Encarnación González Rufino (Dpto. de Informática)

Índice

1. Introducción
2. Excepciones básicas
3. Rebobinado del Stack
4. Tratamiento de excepciones
 - 4.1. Recuperarse tras una excepción
5. Limpieza de recursos

1. Introducción

Desarrollo de software → construir programas fiables y robustos

Fiabilidad:

- RAE: *“Probabilidad de buen funcionamiento de algo”*
- IEEE: “la probabilidad de que un bien funcione adecuadamente durante un período determinado bajo condiciones operativas específicas”

Probabilidad de fallo de un programa:

- Existencia de errores en el código
- Situaciones excepcionales fuera del alcance del programa

Robustez → sistema responde adecuadamente a los errores producidos

Excepciones

- Se utilizan para tratar los posibles errores que se pueden producir en un programa en JAVA
- Lenguajes OO: incorporan la gestión de excepciones como una forma más de manejar errores
- JAVA: es la única forma que está disponible, por lo que es una parte fundamental del lenguaje de programación

Excepción

Una **excepción** es un evento, que ocurre durante la ejecución de un programa, que interrumpe el flujo normal de ejecución de las instrucciones del programa

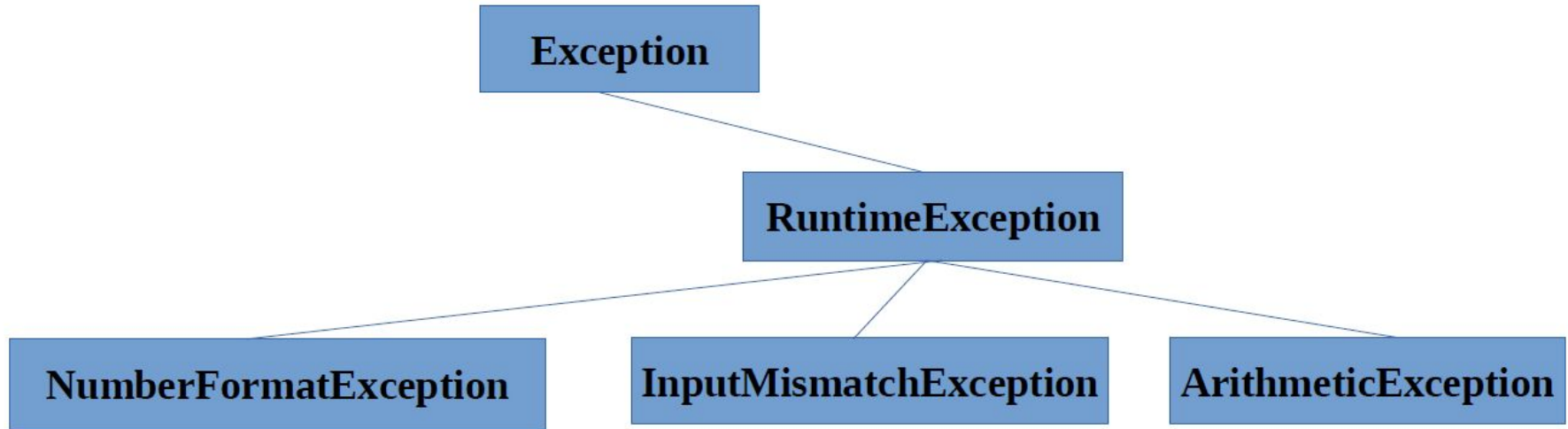
Cuando se produce un error en un método, el método crea un objeto y lo pasa al sistema de ejecución

El objeto (objeto excepción) contiene información sobre el error, incluyendo su tipo y el estado del programa cuando se produjo el error

Crear un objeto de excepción y pasárselo al sistema de ejecución se denomina **lanzar una excepción**.

2. Excepciones básicas

Las excepciones son objetos:



RuntimeException → excepciones **no controladas**

Java permitirá compilar un programa aún sabiendo que ciertas llamadas a métodos pueden provocar excepciones no controladas

Resto excepciones → excepciones **controladas**

Es obligatorio tratarlas

```
class ExcepcionNoControlada
```

```
{  
    public static int divide(int a, int b)  
    {  
        return a / b;  
    }  
}
```

COMPILA!

```
class Ideone
```

```
{  
    public static void main (String[] args) throws java.lang.Exception  
    {  
        ExcepcionNoControlada.divide( 5, 0 );  
    }  
}
```

EJECUTAR:
Se produce
una
excepción!

⚠ stderr

Exception in thread "main" java.lang.ArithmeticException: / by zero
 at ExcepcionNoControlada.divide(Main.java:12)
 at ExcepcionNoControlada.main(Main.java:17)

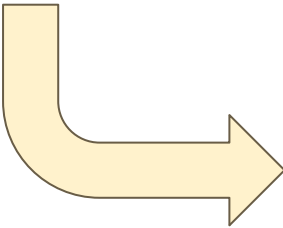
TERMINACIÓN
ANORMAL!

```
class ExcepcionNoControlada
{
    public static int divide(int a, int b)
    {
        return a / b;
    }
}

class Ideone
{
    public static void main (String[] args) throws ArithmeticException
    {
        try {
            ExcepcionNoControlada.divide( 5, 0 );
        }
        catch(ArithmeticException exc)
        {
            System.err.println( "\nERROR: " + exc.getMessage() );
        }
    }
}
```

CAPTURAR
LA
EXCEPCIÓN

GESTIONADO
EL ERROR



```
! stderr
ERROR: / by zero
```


Lanzar una excepción para evitar que se produzca de forma automática la excepción *ArithmeticException*

```
public static int divide2(int a, int b)
{
    if ( b == 0 ) {
        throw new Exception("divisor no puede ser cero");
    }
    return a / b;
}
```

ERROR DE
COMPILACIÓN

información de compilación

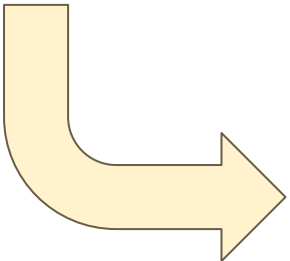
```
Main.java:16: error: unreported exception Exception; must be caught or declared to be
thrown
```

```
    throw new Exception("divisor no puede ser cero");
    ^
```

Se está lanzando un objeto exception sin especificar que el método divide2() puede lanzar una excepción de ese tipo → excepción controladas es obligatorio especificar que el método puede lanzar la excepción: ***throws TipoExcepcion***

```
public static int divide2(int a, int b) throws Exception
{
    if ( b == 0 ) {
        throw new Exception("divisor no puede ser cero");
    }
    return a / b;
}
```

El objetivo de una excepción es o ser tratada o parar el programa

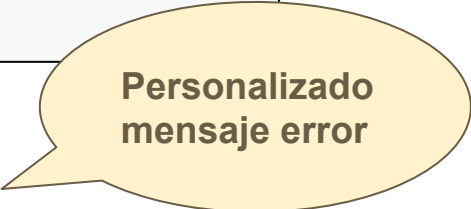


```
public static void main (String[] args) throws java.lang.Exception
{
    try {
        ExcepcionNoControlada.divide2( 5, 0 );
    }
    catch(Exception exc)
    {
        System.err.println( "\nERROR: " + exc.getMessage() );
    }
}
```



stderr

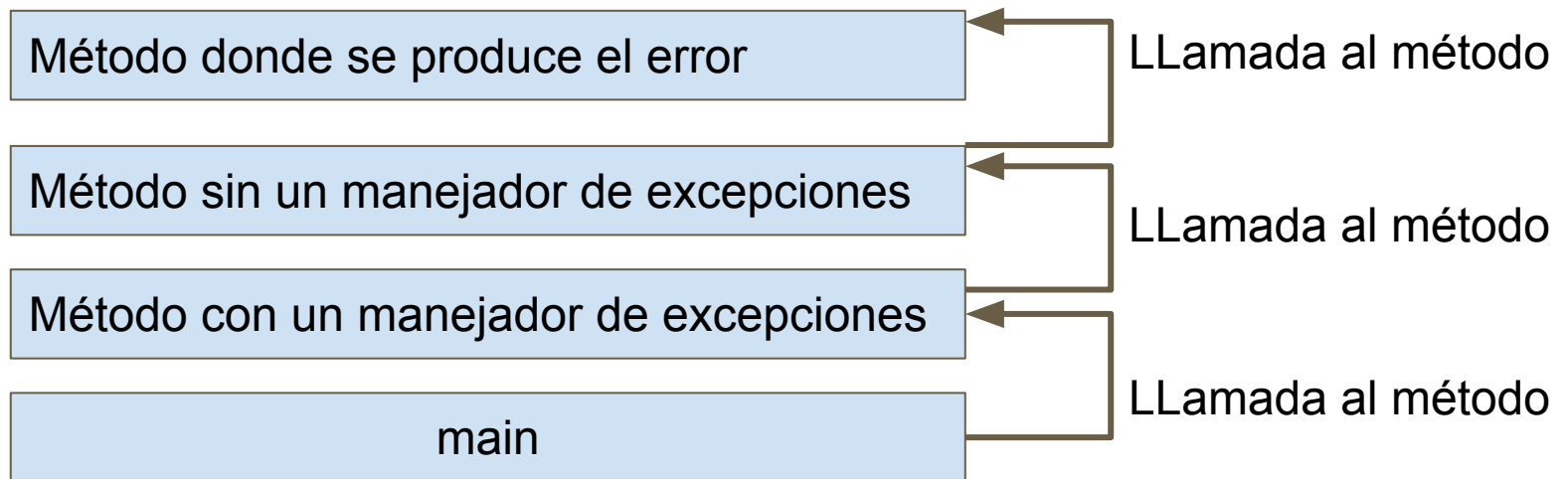
ERROR: divisor no puede ser cero



Personalizado
mensaje error

3. Rebobinado del Stack

Después de que un método lanza una excepción, el sistema de ejecución intenta encontrar un bloque de código para manejar la excepción (manejador de excepciones) → la lista ordenada de métodos que habían sido llamados para llegar al método donde ocurrió el error: **pila de llamadas** (orden inverso al que se llamaron los métodos)

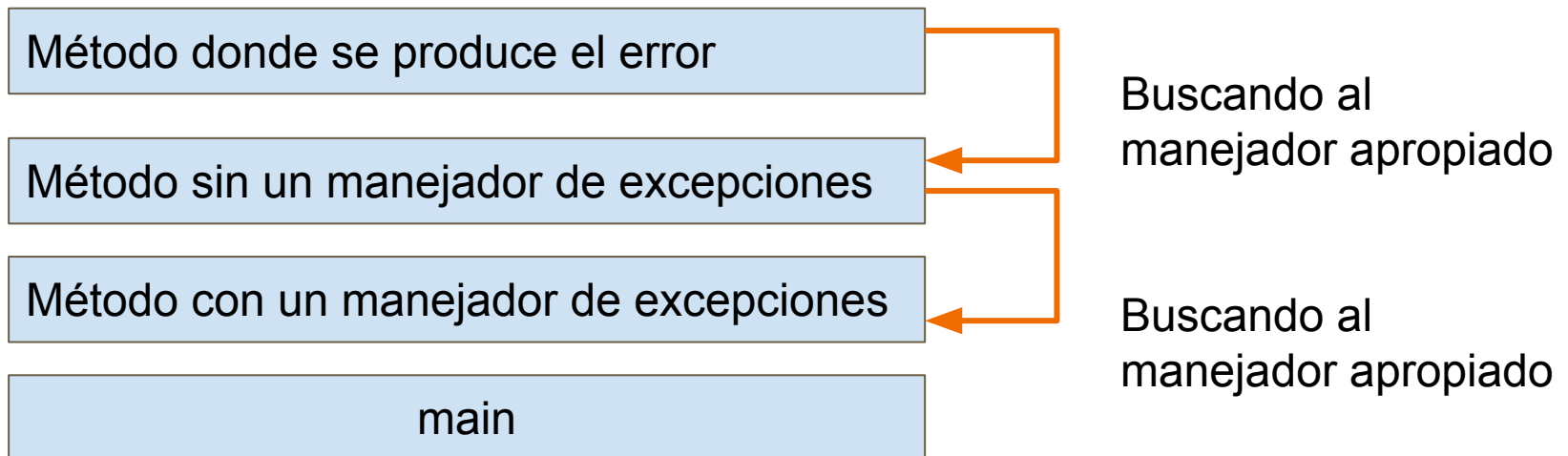


3. Rebobinado del Stack

Cuando se encuentra un manejador adecuado, el sistema de ejecución pasa la excepción al manejador.

Un manejador de excepciones se considera apropiado si el tipo del objeto excepción lanzado coincide con el tipo que puede ser manejado por el manejador: **capturar la excepción.**

Si no se encuentra un manejador de excepciones adecuado el sistema de ejecución (el programa) termina



```
3 public class ExcepcionNoControlada {
4     public static int divide(int a, int b)
5     {
6         return a / b;
7     }
8     public static void principal () {
9
10        System.out.println(divide( 4, 2 ));
11        System.out.println(divide( 5, 0 ));
12        System.out.println(divide( 8, 4 ));
13    }
14    public static void main (String[] args) {
15        principal();
16    }
17 }
```

run:

2

Exception in thread "main" java.lang.ArithmeticException: / by zero
at prueba.ExcepcionNoControlada.divide(ExcepcionNoControlada.java:6)
at prueba.ExcepcionNoControlada.principal(ExcepcionNoControlada.java:11)
at prueba.ExcepcionNoControlada.main(ExcepcionNoControlada.java:15)

C:\Users\pcuesta\AppData\Local\NetBeans\Cache\8.2\executor-snippets\run.xml:53: Java returned: 1

BUILD FAILED (total time: 0 seconds)

1. llamada desde main a principal
2. llamada desde principal a divide(4,2)
3. llamada desde principal a divide(5,0)
4. se produce la excepción dentro de divide
5. la excepción se pasa a la función principal (se detiene)
6. la excepción se pasa al método main: termina el programa informando de la excepción

4. Tratamiento de excepciones: bloque try...catch

Las excepciones pueden capturarse y tratarse, evitando que sigan propagándose por la pila de llamadas

- Código a controlar tiene que estar dentro de un bloque ***try {...}***
- detectar error → lanzar una excepción:
throw new objetoExcepcion(parametros);
- Habitual después del bloque try → método catch por cada una de las excepciones que se quieren capturar:
catch(tipoExcepcion e) {...}

Dado que la excepción va subiendo por la pila de llamadas, no es necesario tratarla exactamente donde se produce, sino que debe tratarse en el método donde se está preparado para tratarla adecuadamente

4. Tratamiento de excepciones: bloque try...catch

Lanzar una excepción supone una ruptura del flujo de ejecución (control) de un programa:

throw new objetoExcepcion(parametros);

Las sentencias que hay después del throw no se ejecutan.

El control continúa, buscando en la pila de llamadas, un manejador catch() que enlaza con el tipo de error lanzado:

catch(tipoExcepcion e) {...}

Si no se lanza ninguna excepción los manejadores (catch) no se ejecutan

El flujo de control continúa con la siguiente sentencia después de los catch, y si no hay más con el siguiente método en la pila de llamadas


```
class ExcepcionNoControlada {  
    private int x;  
    private int y;  
    private int z;
```

```
    public static int divide(int a, int b)  
    {  
        return a / b;  
    }
```

```
    public ExcepcionNoControlada()  
    {  
        try {  
            x = divide( 4, 2 );  
            y = divide( 5, 0 );  
            z = divide( 8, 4 );  
        }  
        catch(ArithmeticException exc)  
        {  
            x = y = z = -1;  
        }  
    }
```

```
    public static void main (String[] args) throws java.lang.Exception  
    {  
        ExcepcionNoControlada c = new ExcepcionNoControlada();  
        System.out.println(c);  
    }
```

<http://ideone.com/X14Mif>

Al crear un objeto de la clase *ExcepcionNoControlada*, se produce una excepción *ArithmeticException*, que es capturada dentro del propio constructor, poniendo el valor de los atributos a -1

El usuario de la clase *ExcepcionNoControlada* no es consciente de que se haya producido una excepción, lo deducirá de los valores de los atributos

4. Tratamiento de excepciones

En un bloque *try... catch* pueden existir varios catch para capturar varios tipos de excepciones

Orden catch:

Debe siempre ponerse primero aquellos más específicos, y después los más generales

Java no permitirá compilar si no se respeta esta norma

Capturar objeto Exception (más general) se capturará cualquier error que no se haya previsto

No es buena idea confiar en Exception para capturar todos los errores como norma general, ya que no permite distinguir entre tipos de errores

```

class Excepciones {
    private int x;
    private int y;
    private int z;
    public static int divide(String a, String b)
    {
        int op1 = Integer.parseInt( a );
        int op2 = Integer.parseInt( b );
        return op1 / op2;
    }
    public Excepciones()
    {
        try {
            x = divide( "4", "f" );
            y = divide( "5", "0" );
            z = divide( "8", "4" );
        }
        catch(ArithmeticException exc)
        {
            x = y = z = -1;
        }
        catch(NumberFormatException exc)
        {
            x = y = z = -2;
        }
    }
}

```

```

public static void main (String[] args) throws java.lang.Exception
{
    Excepciones c = new Excepciones();
    System.out.println(c);
}

```

<http://ideone.com/tTxlvb>

parseInt() lanza la excepción
NumberFormatException

Los atributos toman el valor -2
(el constructor acaba)

Si se desea capturar también la
excepción *Exception* su catch debe
ser el último

4.1 Recuperarse tras una excepción

Una vez que la ejecución ha llegado al código dentro de una cláusula catch, no es posible hacer nada para que la ejecución “vuelva atrás”, y corregir aquello que hizo que la excepción se produjera.

Si que es posible controlar el flujo de control cuando se producen interrupciones: anidando bloques try... catch()

Ejemplo: <http://ideone.com/Ar3tEV>

```
try {  
    for ( ...) {  
        try { ... }  
        catch( )  
    }  
} catch()
```

5. Limpieza de recursos

Cláusula *finally* de un bloque try... catch.

Cuando existe, se ejecutará siempre, haya habido una excepción o no.

El objetivo de las instrucciones dentro de la cláusula finally es siempre la de limpieza de recursos: asegurarse de que, se produzca una excepción o no, se van a cerrar todos los recursos que se hayan visto involucrados.

Ejemplo: <http://ideone.com/Ar3tEV>

se utiliza finally para asegurar que siempre se incluirá un salto de línea tras cada línea del informe de resultados (sólo para mostrar su funcionamiento)