

4. Tratamiento de Excepciones

Programación Modular y Orientada a Objetos

Mikel Larrañaga, Felipe Ibañez y Juan Miguel Lopez

mikel.larrañaga@ehu.es

felipe.anfurrutia@ehu.es

juanmiguel.lopez@ehu.es

Dpto. de Lenguajes y Sistemas Informáticos

UPV/EHU

Contenidos

1. Introducción
2. Tipos de excepciones
 - Excepciones predefinidas en Java
 - Excepciones definidas por el programador
3. Manejo de excepciones en Java
4. Ejemplos de tratamiento de excepciones

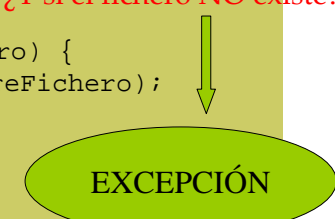
Introducción

- ▣ Objetivo: robustez y fiabilidad
 - Programación defensiva, prever el uso inadecuado
 - Un programa no debe parar ante un error Hw o Sw
- ▣ **Excepción**: suceso en tiempo de ejecución que puede hacer que una rutina fracase

Ejemplo

```
Programa principal {  
    ...  
    leerFichero("Pruebal.txt")  
}
```

```
Subrutina leerFichero(nombreFichero: String) {  
    linea: String  
    abrir(nombreFichero) ← ¿Y si el fichero NO existe?  
    mientras no_fin(nombreFichero) {  
        linea = leer_linea(nombreFichero);  
        tratar(linea);  
    }  
    cerrar(nombreFichero)  
}
```

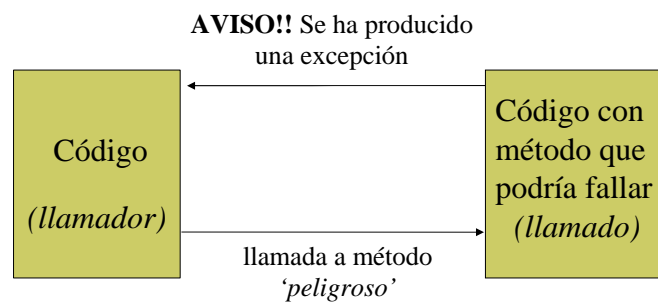


Introducción

▣ Tratamiento:

- Asumir el fracaso e informar a la rutina que ha llamado
- Cambiar el estado y reintentar

Introducción



Tipos de Excepciones

- ❑ Excepciones predefinidas en el lenguaje de programación
 - No hace falta definir las (están ya definidas)
 - El sistema las activa (lanza) automáticamente
 - También pueden ser activadas por el programador
 - El programador se ocupa de su gestión
- ❑ Excepciones definidas por el programador
 - Son las excepciones que define, activa y gestiona el programador

Excepciones en Java: clase **Exception**

- ❑ Las excepciones son clases Java
(<http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Exception.html>)
- ❑ Clase muy sencilla
- ❑ Tiene dos constructores (entre otros)
 - `Exception()`
 - `Exception(String mensaje)`
- ❑ Obtención del mensaje
 - `excepcion.getMessage();`
- ❑ Mostrar traza
 - `excepcion.printStackTrace();`

Manejo de excepciones en Java (I)

- Indicar el bloque de código en el que puede detectarse la excepción
 - Bloque **try**
- Introducir el código necesario para tratar las clases de excepciones que puedan producirse en el bloque **try**
 - Manejador de excepciones
 - Secuencia de bloques **catch**
- Después de un bloque **try** debe haber alguna sentencia **catch**
- Antes de un bloque de cláusulas **catch** debe haber un bloque **try**

Manejo de excepciones en Java (II)

```
try {  
    // código que pudiera fallar  
}  
catch (Exception e) {  
    // tratamiento de la excepción  
}
```

Capturar la excepción

Sólo se ejecutará si se produce la Excepción indicada en **catch**

10

Manejo de excepciones en Java (III)

- ❑ Si se activa una excepción dentro del bloque **try** se pasa el control a su manejador y el código interior al **try** no se ejecuta
 - Se busca en los bloques **catch** aquél que corresponda a la excepción
 - Si se encuentra el bloque **catch** correspondiente se ejecuta el código asociado y después se salta al final del bloque **try**
 - Se busca en el orden del código ➔ se deben programar las cláusulas **catch** en orden decreciente de especificidad
- ❑ Si no surge ninguna excepción en el bloque de código no se ejecutan las sentencias adicionales del bloque **catch**

Manejo de excepciones en Java (IV)

- ❑ Un método *llamador* capturará (**catch**) lo que otro (método *llamado*) propague
- ❑ Una excepción que está sin tratar se propagará al método *llamador* (**throws**)
- ❑ El método que *propaga* la excepción tiene que indicarlo

Ejemplo

```
public void métodoPeligroso() throws Excepcion {
    // ....
    if (algoVaMal ) {
        throw new Excepción();
    }
}

public void cruzarLosDedos() {
    try {
        unObjeto.métodoPeligroso();
    }
    catch (Excepción ex) {
        System.out.println("Mala suerte, no ha funcionado");
        ex.printStackTrace();
    }
}
```

LLamado

Indica que se propaga

Lanza la excepción

LLamador

Intenta ejecutar código que puede lanzar o propagar una excepción

Captura y tratamiento de la excepción

13

Manejo de excepciones en Java (V)

- ❑ Si una excepción llega “propagada” (sin tratar) al subprograma llamador, el control es transferido a su gestor de excepciones
- ❑ Si la excepción llega hasta el programa principal también se transfiere el control a su gestor
 - Si se ha previsto, se trata y el programa termina “normalmente”
 - Si no, se produce un error de ejecución

Excepciones predefinidas

- ❑ `java.lang.ArithmeticException`
- ❑ `java.lang.ArrayStoreException`
- ❑ `java.lang.IndexOutOfBoundsException`
 - `java.lang.ArrayIndexOutOfBoundsException`
 - `java.lang.StringIndexOutOfBoundsException`
- ❑ `java.lang.IllegalArgumentException`
 - `java.lang.NumberFormatException`
 - `java.lang.IllegalThreadStateException`
- ❑ `java.lang.NegativeArraySizeException`
- ❑ `java.lang.NullPointerException`
- ❑ `java.lang.InterruptedExecutionException`

15

Subclases de Exception en Java

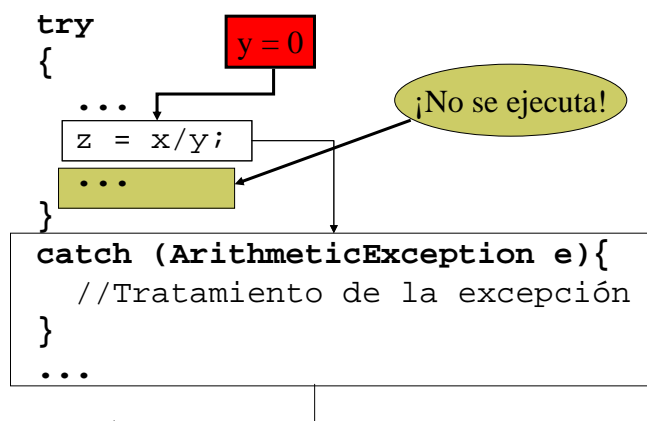
- ❑ `java.io.IOException`
 - `java.io.EOFException`
 - `java.io.FileNotFoundException`
 - `java.io.InterruptedExecutionException`
- ❑ `java.net.MalformedURLException`

Ejemplo

```
public double dividir () {  
    System.out.println("Teclea dos numeros: ");  
    /* Recoger los números tecleados y guardarlos en  
       las variables x, y */  
    try {  
        return (x/y);  
    }  
    catch (ArithmeticException e) {  
        //Tratamiento de la Excepción  
    }  
}
```

y=0 → ArithmeticException
TRATAR ERROR COMO EXCEPCION

Ejemplo



Ejemplo

```
public double dividir () throws ArithmeticException
{
    System.out.println("Teclea dos numeros: ");
    /* Recoger los números tecleados y guardarlos en
       las variables x, y */


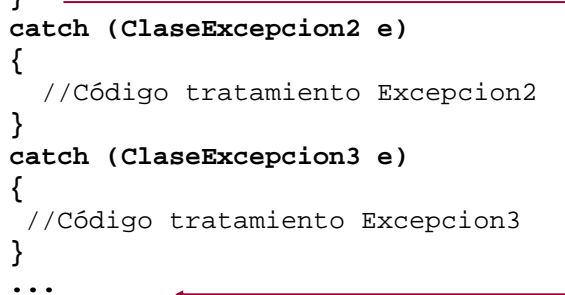

    return (x/y);
}
```

y=0 → ArithmeticException
TRATAR ERROR COMO EXCEPCION





Ejemplo

```
try{
    ... //Se produce una excepción Excepcion3
}
→ catch (ClaseExcepcion1 e) //no casa
{
    //Código tratamiento Excepcion1
}
→ catch (ClaseExcepcion2 e) //no casa
{
    //Código tratamiento Excepcion2
}
→ catch (ClaseExcepcion3 e) //casa
{
    //Código tratamiento Excepcion3
}
... ←
```

Ejemplo


```
try {  
    ... //Se produce una excepción Excepcion1  
}  catch (ClaseExcepcion1 e) //casa  
{  
    //Código tratamiento e1  
}  
 catch (ClaseExcepcion2 e)  
{  
    //Código tratamiento Excepcion2  
}  
catch (ClaseExcepcion3 e)  
{  
    //Código tratamiento Excepcion3  
}  
... 
```

Ejemplo

```
try {  
    ... //Se produce una excepción Excepcion8  
}  catch (ClaseExcepcion1 e) //no casa  
{  
    //Código tratamiento e1  
}  
 catch (ClaseExcepcion2 e) //no casa  
{  
    //Código tratamiento Excepcion2  
}  
 //La excepción se ha quedado sin tratar  
      
//Debe propagarse al método llamador  
... //El resto del método no se ejecuta
```

Ejemplo

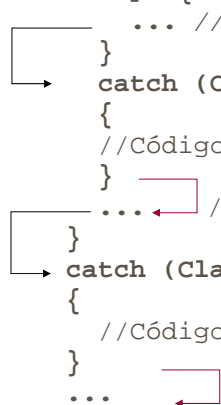
```
try {
    ...
    try {
        ... //Se produce una excepción Excepcion2
    }
    catch (ClaseExcepcion1 e) //no casa
    {
        //Código tratamiento e1
    }
    catch (ClaseExcepcion2 e) //casa
    {
        //Código tratamiento e2
    }
    ...
}
```



23

Ejemplo

```
try {
    ...
    try {
        ... //Se produce una excepción Excepcion1
    }
    catch (ClaseExcepcion1 e) //casa
    {
        //Código tratamiento Excepcion1
    }
    ... //Se produce una excepción Excepcion2
}
catch (ClaseExcepcion2 e) //casa
{
    //Código tratamiento Excepcion2
}
...
```



24

Cláusula finally

- ❑ La cláusula `finally` se utiliza para sentencias que se deben realizar siempre
 - Si **no** se ha producido una excepción: se ejecuta después del bloque `try`
 - Si se ha producido una excepción: se ejecuta después del `catch` correspondiente o, si no lo hay, después de que se haya activado la excepción
 - **Se ejecuta siempre**, no importa que existan sentencias `return` o `break` en el `try` o los `catch`

Cláusula finally

- ❑ Código de limpieza: recupera un estado apropiado para continuar la ejecución aunque sea con la excepción activada
- ❑ Por cada bloque `try` sólo puede haber una cláusula `finally`

Ejemplo

```
try {  
    ... //Finaliza sin excepciones  
}  
catch (ClaseExcepcion1 e){  
    //Código tratamiento Excepcion1  
}  
catch (ClaseExcepcion2 e){  
    //Código tratamiento Excepcion2  
}  
finally{  
    //Código de limpieza  
}  
...
```

Ejemplo

```
try {  
    ... //Se produce una excepción Excepcion1  
}  
catch (ClaseExcepcion1 e) //casa  
{  
    //Código tratamiento Excepcion1  
}  
catch (ClaseExcepcion2 e)  
{  
    //Código tratamiento Excepcion2  
}  
finally  
{  
    //Código de limpieza  
}  
...
```

Ejemplo

```
try{
    ... //Se produce alguna excepción
}
→ catch (ClaseExcepcion1 e) //no casa
{
    //Código tratamiento e1
}
→ catch (ClaseExcepcion2 e) //no casa
{
    //Código tratamiento e2
}
→ finally
{
    //Código de limpieza
}
... →
```

Tratamiento de excepciones

- ❑ Asumir el fracaso e informar al método llamador dejando el programa en un estado estable
- ❑ Cambiar de alguna manera la situación que ha creado la excepción y reintentar la ejecución del subprograma
 - Intentar volver a ejecutar la sentencia con el siguiente elemento (del fichero, de la tabla...)
 - Buscar un camino alternativo
 - Arreglar lo que ha producido el error
 - Esperar y reintentar

Ejemplo

Asumir el fracaso e informar al método llamador dejando el programa en un estado estable

```
public double dividir() throws ArithmeticException {  
    System.out.println("Teclea dos numeros: ");  
    /* Recoger los números tecleados y guardarlos en  
       las variables x, y */  
    try  
        return (x/y);  
    catch (ArithmeticException e) {  
        System.out.println("Imposible dividir")  
        throw (new ArithmeticException());  
    }  
}
```

y=0 → ArithmeticException

*Informa al método llamador,
pasándole la excepción*

Ejemplo

Arreglar lo que ha producido el error

```
public double dividir () {  
    System.out.println("Teclea dos numeros: ");  
    /* Recoger los números tecleados y guardarlos en  
       las variables x, y */  
    try  
        return (x/y);  
    catch{  
        y = 1;  
        return (x/y);  
    }  
}
```

y=0 → ArithmeticException

Ejemplo

Reintentar la ejecución del subprograma

```
public double dividir () {  
    System.out.println("Teclea dos numeros: ");  
    /* Recoger los números tecleados y guardarlos en  
       las variables x, y */  
    try  
        return (x/y);  
    catch{  
        dividir();  
    }  
}
```

y=0 → ArithmeticException

Excepciones definidas por el programador

- En algunas circunstancias el programador querrá definir sus propias excepciones
throw instanciaExcepcion
- El programador debe controlar cuándo deben activarse dichas excepciones
- Se tratan igual que las predefinidas

Excepciones definidas por el programador

Definición de una excepción del programador:

```
class DivisorPar extends Exception {  
    public DivisorPar(){  
        super();  
    }  
  
    public DivisorPar(String s){  
        super(s);  
    }  
}
```

Ejemplo

```
public double dividir(int x, int y) throws DivisorPar {  
    if (y % 2 == 0) PROPAGAR  
        throw new DivisorPar("Division  
                               entre un numero par");  
    } LANZAR
```

Ejemplo

```
public static void main(String[] args) {  
    double resul;  
    System.out.println("Teclea dos numeros: ");  
    /* Recoger los números tecleados y guardarlos en las  
    variables x, y */  
    try  
        resul=dividir(x, y);  
    catch (DivisorPar e) {  
        y = y + 1;  
        resul = dividir(x,y);  
    }  
    ...  
}
```

INTENTAR

CAPTURAR Y TRATAR

Excepciones definidas por el programador

- ❑ Pasos a seguir en la generación voluntaria de excepciones:
 1. Plantear la solución ideal
 2. Identificar situaciones excepcionales
 3. Programar el tratamiento de las situaciones excepcionales
 4. Aplicación/integración