Tema 5
Generación y tratamiento de excepciones

Programación II

Alicia Garrido Alenda

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Concepto de excepción

Excepción

- Es un error o evento que se produce durante la ejecución de un programa.
- Interrumpe el flujo normal del programa.

- Cuando se produce un error en ejecución el programa debe:
  - Presentar un mensaje indicando el problema y acabar adecuadamente.
  - Si el error es recuperable ofrecer la posibilidad de corregirlo.

La necesidad de tratar los errores

- Queremos hacer un método que lee un fichero y copia su contenido en memoria:

```
leeFichero(){
 Abrir el fichero;
   //¿Y si el fichero no se puede abrir?
 Obtener longitud fichero;
   //¿Y si no puedo determinar su longitud?
 Reservar espacio en memoria;
   //¿Y si no puedo reservar suficiente memoria?
 Copiar contenido;
   //¿Y si falla la lectura?
 Cerrar fichero;
   //¿Y si no puedo cerrar el fichero?
}
```


Tratamiento clásico de errores

- Mediante el uso de estructuras condicionales se determina el error:

```
int leeFichero(){
int codigoError=0;
 Abrir el fichero;
 if(fichero abierto){
  Obtener longitud fichero;
  if (longitud conseguida){
   Reservar espacio en memoria;
   if (memoria reservada){
    Copiar contenido;
    if (falla lectura)
     codigoError=-1;
   } else codigoError=-2;
  } else codigoError=-3;
  Cerrar fichero;
  if (fichero abierto && codigoError==0)
    codigoError=-4;
 } else codigoError=-5;
 return codigoError;
}
```

Tratamiento clásico de errores

- Este método:
  - Genera código difícil de leer.
  - Se pierde el flujo de ejecución lógico.
  - El código final puede resultar difícil de modificar.

El paradigma `throw-and-catch`

- Mediante el uso de excepciones:

```
int leeFichero(){
 int codigoError=0;
 try{
  Abrir el fichero;
  Obtener longitud fichero;
  Reservar espacio en memoria;
  Copiar contenido;
  Cerrar fichero;
  }
  catch(falla apertura fichero){codigoError=-5;}
  catch(falla determinar longitud){codigoError=-3;}
  catch(falla reservar memoria){codigoError=-2;}
  catch(falla lectura){codigoError=-1;}
  catch(falla cierre fichero){codigoError=-4;}
 return codigoError;
}
```

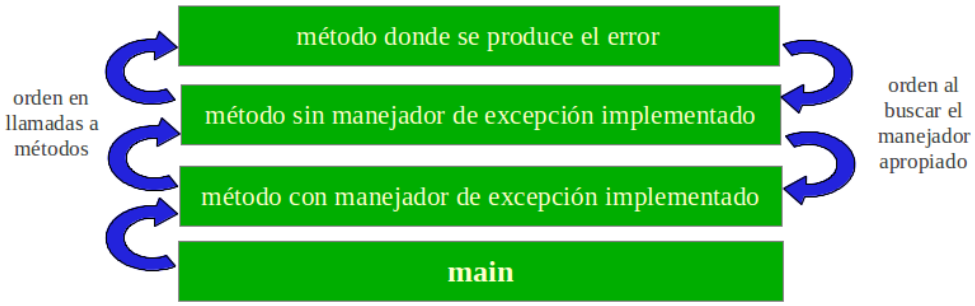

El paradigma `throw-and-catch`

- Este método:
  - No nos evita detectar, informar y manejar los errores.
  - Nos permite escribir el flujo principal de nuestro código de forma lógica.
  - Separa el código de la aplicación del código que trata los casos excepcionales (errores, excepciones).

¿Qué es una excepción en Java?

- Cuando se produce un error durante la ejecución de un método éste crea un objeto de tipo excepción *(crea una excepción)*.
- Lo devuelve al sistema *(lanza la excepción)*.
- Interrumpe el flujo de ejecución.
- Este objeto contiene información sobre el error, incluido su tipo y el estado del programa donde ocurrió.
- El sistema de ejecución recorre la pila de llamadas buscando un método que contenga un bloque de código que se ocupe de la excepción *(un manejador de la excepción)*.

¿Qué es una excepción?



Tipos de excepciones en Java

No comprobadas o de ejecución

Son irrecuperables y las detecta directamente Java, como por ejemplo las *excepciones de indexación* o las *excepciones de apuntadores*.

- No es obligatorio su tratamiento.

Comprobadas

Son comprobadas por el compilador y representan errores recuperables.

- Es obligatorio que sean especificadas o capturadas por algún método.

Captura y tratamiento de excepciones

```
import java.io.*;
public class VectorEnteros{
  private int[] vector;
  private static int SIZE = 20;
  public VectorEnteros(){
   vector = new int[SIZE];
   for(int i=0;i<SIZE;i++)
      vector[i]=i;
  }
  public static void setSIZE(int valor){
   if(valor>0)
      SIZE=valor;
  }
  public void resize(){
   vector = new int[SIZE];
   for(int i=0;i<SIZE;i++)
      vector[i]=i;
  }
  public void escribeVector(String name){
   FileWriter fdo = new FileWriter(name);
   String cadena=null;
   for (int i=0; i<SIZE ; i++){
      cadena="valor: "+i+" = "+vector[i];
      fdo.write(cadena);
   }
   fdo.close();
  }
}
```

- Este código no compila

Captura y tratamiento de excepciones

```
  public void escribeVector(String name){
   FileWriter fdo = new FileWriter(name);
   String cadena=null;
   for (int i=0; i<SIZE ; i++){
      cadena="valor: "+i+" = "+vector[i];
      fdo.write(cadena);
   }
   fdo.close();
  }
```

- Genera los siguientes errores de compilación:

```
VectorEnteros.java:18: error: unreported exception IOException; must be caught or
declared to be thrown
    fdo = new FileWriter(name);
          ^
VectorEnteros.java:22: error: unreported exception IOException; must be caught or
declared to be thrown
      fdo.write(cadena);
                ^
VectorEnteros.java:24: error: unreported exception IOException; must be caught or
declared to be thrown
    fdo.close();
              ^
3 errors
```

- Las excepciones comprobadas han de ser consideradas obligatoriamente en el código.

La instrucción `try-catch`

- Las instrucciones que pueden lanzar excepciones se deben incluir dentro de bloques **`try`**.
- Los manejadores de excepciones se incluyen en bloques **`catch`** asociados a éstos.

```
try{
  ...
}
catch(TipoExcepcion1 nombre1){
// Tratamiento para excepciones de este tipo
}
catch(TipoExcepcion2 | TipoExcepcion3 nombre2){
// El tratamiento para excepciones de estos dos tipos
    es la misma
}
...
```

- Las instrucciones de un bloque **`catch`** se ejecutan cuando alguna de las instrucciones del bloque **`try`** lanza una excepción de las tratadas.
- También podemos poner cada instrucción que puede lanzar una excepción en un bloque **`try`** diferente y proporcionar un manejador para cada uno.

Tratamiento de excepciones

- El manejador de una excepción como mínimo debe emitir un mensaje indicando el problema detectado:

```
public void escribeVector(String name){
 try{
  FileWriter fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
  fdo.close();
 } catch(ArrayIndexOutOfBoundsException e){
   System.err.println("Indice fuera de rango "+ i);
 }
 catch(IOException ex){
   System.err.println(name+" no se pudo abrir");
   System.exit(0);
 }
}
```

- Pueden hacer más: preguntar al usuario una alternativa, intentar recuperarse del error o terminar la ejecución del programa.

El bloque `finally`

- El bloque **`finally`** es opcional, y su objetivo es dejar la ejecución del programa en un estado correcto independientemente de lo que suceda dentro de un bloque **`try`** (cerrar ficheros, liberar recursos, ...).
- El bloque **`try`** de **`escribeVector`** del ejemplo anterior puede terminar de tres formas:
  - La instrucción **`new FileWriter(name)`** falla y lanza una excepción **`IOException`**.
  - La instrucción **`vector[i]`** falla y lanza una excepción **`ArrayIndexOutOfBoundsException`**.
  - No hay fallos y el bloque **`try`** termina normalmente.
- Cuando aparece un bloque **`finally`** siempre se ejecuta el código que contiene, tanto si hay excepciones como si no las hay.

El uso del bloque `finally`

- Nuestro ejemplo:

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
 } catch(ArrayIndexOutOfBoundsException e){
   System.err.println("Indice fuera de rango "+ i);
 }
 catch(IOException ex){
   System.err.println(name+" no se pudo abrir");
 }
 finally {
   if (fdo != null) {
    try{fdo.close();}
    catch(IOException ex){System.err.println(ex);}
   }
 }
}
```

Se lanza una `IOException`

- ¿Qué código se ejecuta si no se puede abrir el fichero?

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
 } catch(ArrayIndexOutOfBoundsException e){
   System.err.println("Indice fuera de rango "+ i);
 }
 catch(IOException ex){
   System.err.println(name+" no se pudo abrir");
 }
 finally {
   if (fdo != null) {
    try{fdo.close();}
    catch(IOException ex){System.err.println(ex);}
   }
 }
}
```


Se lanza una `IOException`

- Únicamente este:

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);




  }


 catch(IOException ex){
   System.err.println(name+" no se pudo abrir");
 }
 finally {
   if (fdo != null) {

   }
 }
}
```


Se lanza una `ArrayIndexOutOfBoundsException`

- ¿Qué código se ejecuta si SIZE es mayor que vector.length?

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
 } catch(ArrayIndexOutOfBoundsException e){
   System.err.println("Indice fuera de rango "+ i);
 }
 catch(IOException ex){
   System.err.println(name+" no se pudo abrir");
 }
 finally {
   if (fdo != null) {
    try{fdo.close();}
    catch(IOException ex){System.err.println(ex);}
   }
 }
}
```


Se lanza una `ArrayIndexOutOfBoundsException`

- El siguiente:

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
 } catch(ArrayIndexOutOfBoundsException e){
   System.err.println("Indice fuera de rango "+ i);
 }



 finally {
   if (fdo != null) {
    try{fdo.close();}
    catch(IOException ex){System.err.println(ex);}
   }
 }
}
```

El bloque `try` termina normalmente

- ¿Qué código se ejecuta?

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
 } catch(ArrayIndexOutOfBoundsException e){
   System.err.println("Indice fuera de rango "+ i);
 }
 catch(IOException ex){
   System.err.println(name+" no se pudo abrir");
 }
 finally {
   if (fdo != null) {
    try{fdo.close();}
    catch(IOException ex){System.err.println(ex);}
   }
 }
}
```


El bloque `try` termina normalmente

- El siguiente:

```
public void escribeVector(String name){
 FileWriter fdo = null;
 try{
  fdo = new FileWriter(name);
  String cadena=null;
  for (int i=0; i<SIZE ; i++){
    cadena="valor: "+i+" = "+vector[i];
    fdo.write(cadena);
  }
 }

 finally {
   if (fdo != null) {
    try{fdo.close();}
    catch(IOException ex){System.err.println(ex);}
   }
 }
}
```


Tratamiento de excepciones por defecto

```
public class DivisionPorCero{
  public void division(int n1,int n2){
   System.out.println(n1+" / "+n2+" = "+(n1/n2));
   System.out.println("Salimos de division");
  }
  public static void main(String[] args){
   DivisionPorCero objeto=new DivisionPorCero();

   objeto.division(10,0);
   System.out.println("Salimos de main");
  }
}
```

- Salida:

```
> java DivisionPorCero
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivisionPorCero.division(DivisionPorCero.java:3)
        at DivisionPorCero.main(DivisionPorCero.java:9)
```


`finally` y el tratamiento por defecto

```
public class DivisionPorCero{
  public void division(int n1,int n2){
   try{
       System.out.println(n1+" / "+n2+" = "+(n1/n2));
   } finally {
      System.out.println("finally de division hecho");
   }
   System.out.println("Salimos de division");
  }
  public static void main(String[] args){
   DivisionPorCero objeto=new DivisionPorCero();

   objeto.division(10,0);
   System.out.println("Salimos de main");
  }
}
```

- Salida:

```
> java DivisionPorCero
finally de division hecho
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at DivisionPorCero.division(DivisionPorCero.java:4)
        at DivisionPorCero.main(DivisionPorCero.java:13)
```

`finally` y el tratamiento por defecto

```
public void division(int n1,int n2){
 try{
     System.out.println(n1+" / "+n2+" = "+(n1/n2));
 } finally {
    System.out.println("finally de division hecho");
 }
 System.out.println("Salimos de division");
}
public static void main(String[] args){
 DivisionPorCero objeto=new DivisionPorCero();
 try{  objeto.division(10,0); }
 catch (ArithmeticException ex){
  System.out.println("Captura de excepcion");
 }
 System.out.println("Salimos de main");
}
}
```

- Salida:

```
> java DivisionPorCero
finally de division hecho
Captura de excepcion
Salimos de main
```


`finally` y la instrucción `return`

```
public int division(int n1,int n2){
 try{
   System.out.println(n1+" / "+n2+" = "+(n1/n2));
 }
 catch(ArithmeticException e) {
  System.out.println("Excepcion *"+e.getMessage()+"*");
  return 0;
 } finally {
  System.out.println("finally de division hecho");
 }
 System.out.println("Salimos de division");
 return 1;
}
public static void main(String[] args){
  DivisionPorCero objeto=new DivisionPorCero();
  int i=objeto.division(10,0);
  System.out.println("Devuelve "+i);
  System.out.println("Salimos de main");
 }
}
```

- Salida:

```
> java DivisionPorCero
```

¿qué salida mostrará?

Propagación de excepciones: la cláusula `throws`

- A veces interesa que las excepciones sean tratadas en otro método, y no donde se producen realmente.
- Para ello un método debe especificar qué excepciones puede lanzar y no va a tratar.
- Esto se indica con la cláusula **`throws`** en la cabecera.

```
public void escribeVector(String name)
                          throws IOException{
 FileWriter fdo = new FileWriter(name);
 String cadena=null;
 for (int i=0; i<SIZE ; i++){
   cadena="valor: "+i+" = "+vector[i];
   fdo.write(cadena);
 }
 fdo.close();
}
```

- Las excepciones comprobadas tienen que ser tratadas o propagadas.

Herencia y la cláusula `throws`

- Si un método de una superclase propaga excepciones con **`throws`**:
  - La sobreescritura de dicho método en sus subclases sólo puede propagar un subconjunto de las excepciones propagadas por el método de la superclase.

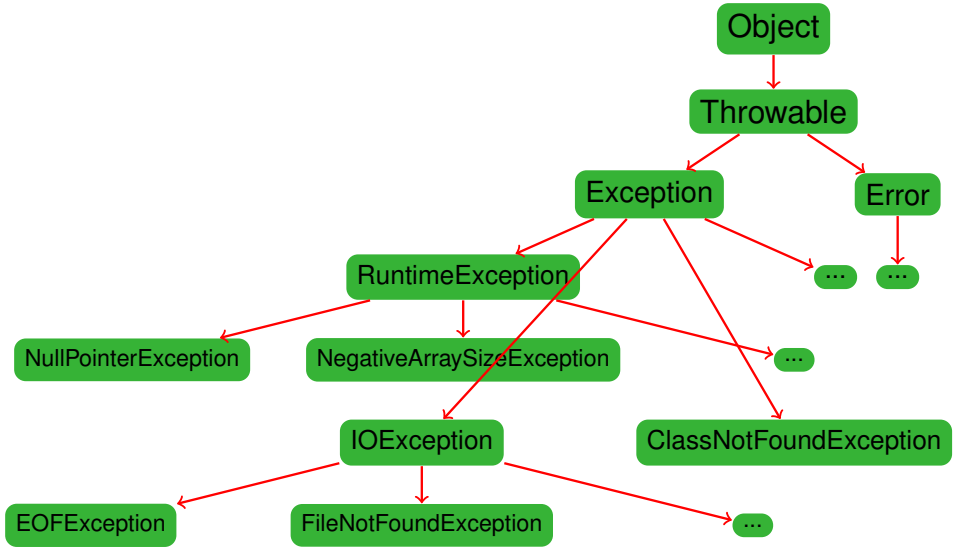
```
public class A{
  ...
  protected void propagaExcepcion()
                    throws Excep1, Excep2, Excep3 {
     ...
  }
  ...
}
public class B extends A{
  ...
  protected void propagaExcepcion() throws
                   Excep1, Excep2, subclaseExcep3 {
     ...
  }
  ...
}
```

La instrucción `throw`

- El programador puede lanzar excepciones cuando considere oportuno con la instrucción **`throw`**.

```
//EmptyStackException es no comprobada
//se propaga sola
 public int getElemento(){
  int e=Integer.MIN_VALUE;
  if (nElementos<=0)
   throw new EmptyStackException();
  nElementos--;
  e=vector[nElementos];
  vector[nElementos]=Integer.MIN_VALUE;
  return e;
 }
```


La clase `Throwable` y sus subclases



La clase `Throwable`

- Únicamente objetos que son instancias de la clase **`Throwable`** (o una subclase de ésta) pueden ser lanzados por la máquina virtual de Java o por una instrucción **`throw`**, y además sólo ellos pueden ser argumentos de las cláusulas **`catch`**.
- La clase **`Throwable`** y sus subclases tienen dos constructores:
  - Constructor sin argumentos.
  - Constructor con un argumento de tipo **`String`**: utilizado para emitir mensajes de error.
- Un objeto de la clase **`Throwable`** contiene el estado de la pila de ejecución en el momento que fue creado.

Métodos de la clase `Throwable`

`String` **`getMessage()`**: devuelve el texto con el mensaje de error del objeto.

`void` **`printStackTrace()`**: imprime el objeto receptor del mensaje y su traza en la salida de error estándar.

`void` **`printStackTrace(PrintWriter s)`**: imprime el objeto receptor del mensaje y su traza en el fichero especificado.

Información sobre excepciones: ejemplo

```
public class Ejemplo {
 public void e1(String paso){
  try{
    String cadena=paso;
    cadena.equals("");
  } catch(NullPointerException e) {
    System.out.println("Excepcion: "+e.getMessage());
    e.printStackTrace();
  }
 }
 public static void main(String[] args){
  (new Ejemplo()).e1(null);
 }
}
```

Salida:

```
> java Ejemplo
Excepcion: null
java.lang.NullPointerException
        at Ejemplo.e1(Ejemplo.java:5)
        at Ejemplo.main(Ejemplo.java:12)
```


Comprobación del tratamiento de excepciones

- En Java los métodos deben tratar, o al menos especificar en su cabecera, todas las excepciones que se pueden producir dentro de ellos.
- Hay un grupo de excepciones (**`RunTimeException`**) que se pueden producir en situaciones frecuentes, y por tanto el compilador de Java no exige su comprobación.
- Para todas las demás el compilador comprueba si se tratan dentro de los métodos en que se pueden producir, o bien son propagadas (especificadas en la cabecera).

Crear excepciones propias

- Un programador puede definir sus propias excepciones heredando de alguna de las clases de excepciones ya existentes.

```
public class miExcepcion extends Exception {
 public miExcepcion(int i){
    super(Integer.toString(i));
  }
}
```

- Y lanzarlas como las demás excepciones.

```
int indice=0;
...
throw new miExcepcion(indice);
...
```


Creando nuestra propia excepción: ejemplo

```
class ExcepcionDivisionPorCero extends Exception{
 ExcepcionDivisionPorCero(){
   super();
 }
 ExcepcionDivisionPorCero(String msg){
   super(msg);
 }
}
```

Lanzando nuestra propia excepción: ejemplo

```
public class DivisionPorCero{
  public void division(int n1,int n2){
   try{
    if (n2==0)
     throw new ExcepcionDivisionPorCero("n2 igual a 0");
    System.out.println(n1 + " / " + n2 + " = " + (n1/n2));
   } catch (ExcepcionDivisionPorCero e) {
     System.out.println("Tratamos "+e);
   }
   System.out.println("Salimos de division");
  }
  public static void main(String[] args){
   DivisionPorCero objeto=new DivisionPorCero();
   objeto.division(10,0);
   objeto.division(10,2);
   System.out.println("Salimos de main");
 }}
```

Salida:

```
> java DivisionPorCero
Tratamos ExcepcionDivisionPorCero: n2 igual a 0
Salimos de division
10 / 2 = 5
Salimos de division
Salimos de main
```


Definición de excepciones

- Se definen nuevas excepciones para:
  - Incluir más información útil.
  - Diferenciar excepciones por su tipo en las cláusulas **`catch`**.

Ejercicio

- Modifica este código para que no se invoque el método `division` cuando no se lance una **`ExcepcionDivisionPorCero`**:

```
public class DivisionPorCero{
 public void division(int n1,int n2) throws
     ExcepcionDivisionPorCero {
  if (n2==0)
   throw new ExcepcionDivisionPorCero("n2 igual a 0");
  System.out.println(n1+ " / " +n2+ " = " +(n1/n2));
  System.out.println("Salimos de division");
 }
 public static void main(String[] args){
  DivisionPorCero objeto=new DivisionPorCero();
  for(int j=0;j<args.length;j++){
    int i=Integer.parseInt(args[j]);
    try{
      objeto.division(10,i);
    }catch (ExcepcionDivisionPorCero ex){
      System.out.println("Captura de excepcion");
    }
  }
  System.out.println("Salimos de main");
 }
}
```