Fundamentos de Computadores y Lenguajes

Fundamentos de Computadores y Lenguajes

6. Tratamiento de errores

- 1. Tratamiento de errores por paso de parámetros
- 2. Excepciones
- 3. Bloques de tratamiento de excepciones
- 4. Errores o excepciones predefinidos
- 5. Instrucciones try anidadas
- 6. Lanzar excepciones
- 7. Usar tus propias excepciones
- 8. La cláusula finally
- 9. Principales formas de gestión de errores
- 10. Cuándo usar excepciones

1. Tratamiento de errores por paso de parámetros

En los lenguajes de programación más antiguos (C, Fortran, ...)

Cada método o función retorna un valor

- generalmente un código numérico indicando si había habido error o no, y cuál

Ejemplo de tratamiento de errores por paso de parámetros

```
error=funcion1(...);
if (error==1) {
   mostrar mensaje error 1;
} else if (error==2) {
   mostrar mensaje error 2;
} else {
   error=funcion2(..);
   if (error==1) {
      mostrar mensaje error 1;
   } else if (error==3) {
       mostrar mensaje error 3;
   } else {
       ...
   }
}
```

Inconvenientes

- el código de chequeo de error aparece por todas partes, mezclado con el código normal
- en muchos casos el chequeo de error se omite por "pereza" o desconocimiento
  - lleva a situaciones de error que pasan inadvertidas

```
funcion1();
funcion2();
...
```


2. Excepciones

Son un mecanismo especial para gestionar errores

- Separan claramente el tratamiento de errores del código normal
- Evitan que haya errores que pasen inadvertidos
- Propagan de forma automática los errores desde los métodos más internos a los más externos
- Permiten agrupar en un lugar común el tratamiento de errores que ocurren en varios lugares del programa

Las excepciones se lanzan para avisar de un error:

- automáticamente, cuando el sistema detecta un error
- explícitamente cuando el programador lo establezca

Presentes en lenguajes modernos (Java, Ada, C++,...)

Ejemplo de bloque de tratamiento de excepciones

```
try {
    funcion1();
    funcion2();
    ...
} catch (error e) {
    tratamiento del error;
}
```


Mecanismo

Las excepciones en Java son objetos de una clase extendida de otra clase especial llamada `Throwable`.

Cuando ocurre un error en una línea de código:

- Se crea una instancia de la excepción, y se lanza (`throw`) la excepción
- El bloque que contiene esa línea de código se aborta en ese momento

Mecanismo (cont.)

- El bloque donde ocurre la excepción puede decidir tratarla (`catch`) o dejarla pasar
  - Si se trata, se ejecutan las instrucciones de un manejador, la situación de error termina, y se sigue por el siguiente bloque
- Si se deja pasar, el siguiente bloque puede a su vez tratarla o dejarla pasar
  - Si nadie la trata, el programa se interrumpe y aparece un mensaje de error

Ejemplo de elevación implícita: División por cero

```
import fundamentos.*;
public class DivisionCero {
  public static void main(String[] args) {
    int i,j;
    Lectura leer=new Lectura ("Enteros");

    leer.creaEntrada("i",0);
    leer.creaEntrada("j",0);
    leer.espera("introduce datos");
    i=leer.leeInt("i");
    j=leer.leeInt("j");
    System.out.println("i/j="+i/j);
    System.out.println("Fin del programa");
  } // main
} // DivisionCero
```

Cuando j vale 0 se lanza ArithmeticException

Cuando se eleva la excepción esta línea no se ejecuta

3. Bloques de tratamiento excepciones

La forma general de escribir un bloque en el que se tratan excepciones es:

```
try {
    instrucciones;
} catch (ClaseExcepcion1 e) {
    instrucciones de tratamiento;
} catch (ClaseExcepcion2 e) {
    instrucciones de tratamiento;
}
```


Ejemplo: división por cero con tratamiento

```
int i,j;
Lectura leer=new Lectura ("Enteros");
try {
  leer.creaEntrada("i",0);
  leer.creaEntrada("j",0);
  leer.espera("introduce datos");
  i=leer.leeInt("i");
  j=leer.leeInt("j");
  System.out.println("i/j="+i/j);
} catch (ArithmeticException e) {
  Mensaje error = new Mensaje();
  error.escribe("Detectada excepcion: "+e);
}
System.out.println("Fin del programa");
```

Tratamiento general

Es posible poner un tratamiento específico para una excepción, como en el ejemplo, o uno general:

```
} catch (Exception e) {
    instrucciones de tratamiento;
}
```

El tratamiento general es cómodo pero *no es recomendable*,

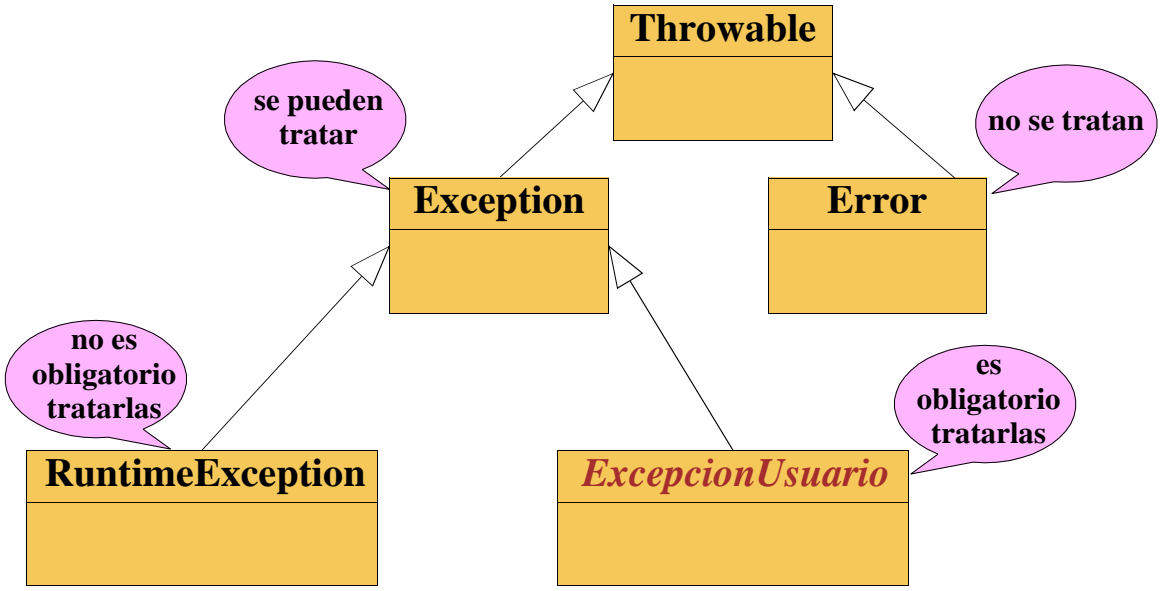
- puede ocurrir un tratamiento inadecuado para una excepción no prevista

Las excepciones se pueden concatenar a un `String`

- lo que las transforma en texto que describe el error

4. Errores o excepciones predefinidos

Jerarquía de las excepciones



Algunas excepciones no comprobadas (clase `RuntimeException`)

`ArithmeticException`	Error aritmético (x/0, ...)
`ArrayIndexOutOfBoundsException`	Índice de array fuera de límites (<0 o >=length)
`ArrayStoreException`	Tipo incorrecto al crear un array
`IllegalArgumentException`	Argumento ilegal en la llamada a un método
`IndexOutOfBoundsException`	Indice fuera de límites (p.e., en un ArrayList)
`NegativeArraySizeException`	Tamaño de array negativo
`NullPointerException`	Uso de una referencia nula
`NumberFormatException`	Formato de número incorrecto
`StringIndexOutOfBounds`	Indice usado en un String está fuera de límites

5. Instrucciones `try` anidadas

Las instrucciones `try` se pueden anidar cada tratamiento trata las excepciones de su bloque

Si la excepción no se trata se le pasa al bloque de fuera

El anidamiento puede ser explícito, como en el ejemplo, o implícito en la llamada a un método.

Sintaxis de las instrucciones try anidadas

```
try {
    instrucciones;
    try {
       instrucciones;
    } catch (Exception e) {
        manejador;
    } // try interno
    más instrucciones;
} catch (Exception e) {
      otro manejador;
} // try externo
```


6. Lanzar excepciones

Se pueden lanzar con:

```
throw objetoExcepcion;
throw new NombreExcepcion();
```

Y si es predefinida

```
throw new NombreExcepcion("mensaje");
```

Ejemplo:

```
...
if (clave==null) {
     throw new NullPointerException
       ("clave es nula");
}
```

Lanzar la misma excepción

UNIVERSIDAD DE CANTABRIA

En ocasiones se eleva en un manejador la misma excepción:

```
catch (Exception e) {
    System.out.println
      ("Detectada excepcion "+e);
    throw e;
}
```


7. Usar tus propias excepciones

UNIVERSIDAD DE CANTABRIA

Puedes crear tus propias excepciones y utilizarlas para indicar errores:

```
public class MiExcepcion extends Exception {};
```

Y se pueden usar desde un método en otra clase:

```
public class ClaseMia {
  public void metodo() {
    try {
      throw new MiExcepcion();
    } catch (MiExcepcion e) {
      System.out.println("excepcion "+e);
    } // try
  } // metodo
} // ClaseMia
```

Usar tus propias excepciones (cont.)

Si un método eleva una excepción que no sea `RuntimeException` (p.e. una tuya), debe:

- tratarla
- o declararla con una cláusula "`throws`"

Sintaxis para cláusula `throws`

```
public tipo nombreMetodo (parametros)
  throws NombreExcepcion1, NombreExcepcion2
{
    declaraciones;
    instrucciones; // lanzan las excepciones
}
```


Ejemplo

Clase `NoCabe`:

```
public class NoCabe extends Exception {};
```

Añadir un alumno en la clase `Curso`: lanza `NoCabe` si no caben más alumnos

```
public void anade() throws NoCabe {
  if (num==losAlumnos.length) {
    throw new NoCabe();
  } else {
    num++;
    Alumno nuevo=new Alumno();
    nuevo.leeDatos();
    losAlumnos[num-1]=nuevo;
  }
}
```

Ejemplo (cont.): Uso del método

```
try {
    primero.anade();
} catch (NoCabe e) {
    Mensaje msj = new Mensaje();
    msj.escribe("Demasiados alumnos");
}
```


8. La cláusula `finally`

En ocasiones una excepción podría acabar prematuramente un trozo de código

- La cláusula **`finally`** permite crear un bloque de código que se ejecuta siempre después de otro, haya habido excepción o no

Ejemplo:

```
try {
    operacion();
} catch (Exception e) {
    manejador;
} finally {
    siempre;
}
```

La cláusula `finally` (cont.)

- Si no hay error la secuencia es:
  ```
  operación -> siempre
  ```
- Si hay errores manejados la secuencia es:
  ```
  operación(incompleta) -> manejador -> siempre
  ```
- Si hay un error sin manejador:
  ```
  operación(incompleta) -> siempre ->
  se eleva la excepción en el siguiente bloque
  ```

La cláusula **`finally`** es opcional

Toda instrucción **`try`** debe tener al menos un **`catch`** o un **`finally`**

9. Principales formas de gestión de los errores

Según la gravedad del error:

- *recuperable*: se reintenta la operación
- *leve*: se notifica el error, pero la operación continúa
- *grave*: se notifica el error, pero se aborta la operación

Ejemplos: Ya hemos visto ejemplos de las dos últimas

Esquema de operación recuperable

```
while (true) {
    try {
        instrucciones a reintentar
        break o return
    } catch (ClaseExcepcion e) {
        manejador
    } // try
} // while
```


Ejemplo

Leer un número entero y reintentar si hay fallo

```
public static int leeIntBien() {
  while (true) {
    try {
      int valor;
      Lectura lec = new Lectura("Lee entero");
      lec.creaEntrada("Valor entero",0);
      lec.esperaYCierra("Introduce valor");
      valor=lec.leeInt("Valor entero");
      return valor;
    } catch (NumberFormatException e) {
      // no hacemos nada; leeInt ya pone un mensaje
    } // bloque try
  } // while
} // leeIntBien
```

Otro ejemplo

Leer un valor enumerado, reintentando si falla la lectura:

```
import fundamentos.*;

public enum Mes {

   ENERO, FEBRERO, MARZO, ABRIL, MAYO, JUNIO,
   JULIO, AGOSTO, SEPTIEMBRE, OCTUBRE,
   NOVIEMBRE, DICIEMBRE;
```


Otro ejemplo (cont.)

```
  public static Mes lee() {
    Lectura l=new Lectura("Lectura de un mes");
    l.creaEntrada("Mes","Enero");
    while (true) {
      try {
        l.esperaYCierra();
        return Mes.valueOf
           (l.leeString("Mes").toUpperCase());
      } catch (IllegalArgumentException e) {
        Mensaje m=new Mensaje();
        m.escribe("Mes incorrecto");
      }
    } // while
  } // lee
} // Mes
```

10. Cuándo usar excepciones

No deben usarse excepciones para de control de flujo

- por ejemplo para salirse de un lazo o una operación)

No deben usarse las excepciones en casos que no sean de error

- por eficiencia
- y porque hacen más difícil entender el programa

Tampoco debe ser habitual elevar una excepción y tratarla en el mismo método

- en lugar de elevar la excepción, realizar el tratamiento directamente
- excepto si ya hay un manejador escrito que sea adecuado

UNIVERSIDAD DE CANTABRIA