

Excepciones



Objetivos

- Conocer y comprender que son las excepciones en Java
- Aprender a utilizar las excepciones en Java para realizar el tratamiento de los posibles errores de un programa
- Conocer y comprender a capturar excepciones y lanzarlas
- Realizar pequeñas aplicaciones en Java que realicen tratamiento de excepciones



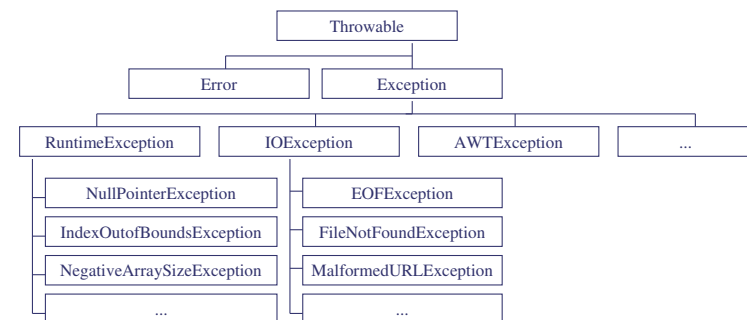
Excepciones - Aspectos Generales

- Una excepción es un error en tiempo de ejecución
- Tipos de excepciones:
 - Fatal: El programa debe finalizar → Terminar ordenadamente mostrando mensaje de error adecuado
 - Recuperable: Dar la oportunidad de corregir el error
- Muchos lenguajes imperativos finalizan el programa cuando surge un error
- Las excepciones permiten escribir código para manejar el error y continuar (si es conveniente) con la ejecución del programa
- Ejemplos: desbordamiento, acceso a una posición de una lista que no existe, ...



Tratamiento de Excepciones - Jerarquía

- Todas las excepciones son objetos de subclases de la clase *Throwable*



Tratamiento de Excepciones – Jerarquía – Excepciones vs Errores

- Clase *Throwable*: Superclase que engloba todas las excepciones
- Clase *Error*: Errores de compilación, del sistema o de la JVM normalmente irrecuperables (no hay que preocuparse de capturarlos ni de tratarlos)
- Clase *Exception*:
 - Define las excepciones que los programas deberían tratar (*IOException*, *ArithmeticException*, etc.) y pueden pertenecer a distintos paquetes
 - Por heredar de *Throwable*, pueden usar los métodos:
 - String `getMessage ()` → Mensaje asociado con la excepción
 - String `toString ()` → Descripción de la excepción
 - void `printStackTrace ()` → Devuelve una traza de la pila de llamadas



Tratamiento de Excepciones – Jerarquía – Excepciones vs Errores

- Clase *RuntimeException*:
 - Durante la ejecución de un programa, se comprueba y lanza automáticamente las excepciones que derivan de esta clase
 - El programador no necesita establecer los bloques `try/catch` para controlarlas. Representan dos tipos de errores:
 - Errores que no suelen ser chequeados por el programador (referencia null en un método)
 - Errores que se deberían chequear, como sobrepasar el tamaño asignado de un array.



Excepciones – Aspectos Generales - Ejemplo

- ¿Qué ocurre con el siguiente programa?

```
public class Desbordamiento {
    String mensajes[] = {"Primero", "Segundo", "Tercero" };
    public static void main(String[] args) {
        for (int i = 0; i <= 3; i++) ←iNo está definido mensajes[3]!
            System.out.println(mensajes[i]);
    }
}
```

Ejecución:

```
Primero Segundo Tercero Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException at
Desbordamiento.main(Desbordamiento.java, Compiled Code)
```



Excepciones – Aspectos Generales

- Cuando aparece un error:
 - Se crea un objeto *Exception* de la clase adecuada con información sobre el error ocurrido
 - Se lanza la excepción
- Propagación de errores en la pila de métodos
- En un programa se debe gestionar correctamente todas o la mayor parte de los errores que se pueden producir:
 - A la antigua usanza: los métodos devuelven un código de error, que se chequean con una serie de `if else if ...`
 - Con soporte en el propio lenguaje → Excepciones
- Beneficios del uso de excepciones:
 - Separación entre el tratamiento de código y los errores
 - Agrupación de tipos de errores y diferenciación



Captura de Excepciones

- Ciertos métodos de la API de JAVA y algunos métodos creados por el programador producen excepciones
- Si se llama a estos métodos sin tenerlo en cuenta se produce un error de compilación. Se resuelve de la siguiente manera:
 - Gestionar la excepción con try-catch-finally
 - Relanzar la excepción hacia un método anterior (llamador), declarando que su método también lanza dicha excepción (throws)
- Es posible capturar la excepción y no hacer nada con ella, aunque es conveniente al menos imprimir un mensaje de error



Captura de Excepciones – try ... catch ... finally

- El control de excepciones se realiza con bloques try – catch – finally
- El código susceptible de generar errores debe estar acotado en un bloque try
- Cuando se produce un error en el bloque try se genera un objeto del tipo de la excepción que se ha producido
- Esta excepción se puede recoger en un bloque catch
- Se pueden incluir tantos bloques catch como sean necesarios, cada uno de los cuales atrapará un tipo de excepción
- El ámbito del bloque catch está restringido a las sentencias del bloque try que le precede inmediatamente
- El bloque finally es opcional: Las sentencias de este bloque se ejecutan siempre (se produzca o no la excepción)



Captura de Excepciones – try ... catch ... finally - Ejemplo

```
public void leeFichero () {  
    try {  
        abrir el fichero;  
        leer el fichero en memoria;  
        cerrar el fichero;  
    } catch (falloAperturaFichero) {hacerAlgo;}  
    catch (falloLectura) { hacerAlgo;}  
    catch (falloCierreFichero) {hacerAlgo;}  
}
```



Captura de Excepciones – try ... catch ... finally - Ejemplo

```
class Excepcion {  
    public static void main(String args[]) {  
        int a,d;  
        d = 0;  
        a = 42 / d; // Error  
        System.out.println("Mensaje");  
    }  
}
```



Captura de Excepciones – try ... catch ... finally - Ejemplo

```
class Excepcion {
    public static void main(String args[]) {
        int d, a;
        try {
            d = 0;
            a = 42 / d;
            System.out.println("Mensaje.");
        } catch (ArithmeticException e) {
            System.out.println("Division por cero.");
        }
        System.out.println("Despues de catch.");
    }
}
```



Captura de Excepciones – try ... catch ... finally - Ejemplo

```
class ManejadoraErrores {
    public static void main(String args[]) {
        int a=0, b=0, c=0;
        Random r = new Random();
        for (int i=0; i<32000; i++) {
            try {
                b = r.nextInt();
                a = 12345 / (b/c);
            } catch (ArithmeticException e) {
                System.out.println("Division por cero."); a = 0;
            }
            System.out.println("a: " + a);
        }
    }
}
```

Si la división produce una excepción, se asigna 0 a la variable a, y continua la ejecución del bucle



Ejercicio 1 – Captura de Excepciones

- Tratar de corregir el error del siguiente código:

```
public class Desbordamiento
{
    static String mensajes[] = {"Primero", "Segundo",
    "Tercero" };
    public static void main(String[] args) {
        for (int i=0; i<=3; i++)
            System.out.println(mensajes[i]);
        System.out.println("Ha finalizado la ejecución");
    }
}
```



Captura de Excepciones – Múltiples catch

```
class Ejemplo {
    public static void main(String args[]) {
        try {
            int a = args.length;
            System.out.println("a = " + a);
            int b = 42 / a;
            int c[] = { 1 };
            c[42] = 99;
        } catch (ArithmeticException e) {
            System.out.println("Division por 0: " + e);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Desbordamiento: " + e);
        }
        System.out.println("Despues de try/catch ");
    }
}
```

Este código puede generar más de un tipo distinto de excepción

Si se pone **Exception** en el primer **catch**, el segundo nunca llegaría a ejecutarse



Captura de Excepciones – try anidados

```
class Anidados {
public static void main(String args[]) {
    try {
        int a = args.length;
        int b = 42 / a;
        System.out.println("a = " + a);
        try {
            if(a==1) a = a/(a-a);
            if(a==2) { int c[] = { 1 };      c[42] = 99; }
        }
        catch(ArrayIndexOutOfBoundsException e)
        { S.O.P.("Desbordamiento: " + e); }
        catch(ArithmeticException e) { S.O.P. ("Div por 0: " + e);}}
}
```

- Si **a=0**, se genera una división por 0 y lo captura el catch externo
- Si **a=1**, genera división por 0. Como no puede capturarlo el catch interno, lo captura el externo.
- Si **a=2**, se produce un desbordamiento que lo captura el catch interno



Lanzamiento de Excepciones - throws

- Si un método puede lanzar una o varias excepciones, debe/n declararse mediante una cláusula *throws*
- Si un método puede lanzar varias excepciones se ponen separadas por comas
- Ejemplo:
 - En la clase String, el método charAt lanza la excepción `IndexOutOfBoundsException` cuando:
 - El índice es negativo
 - El índice es menor de la longitud del array
 - Método:

```
public char charAt(int index)
    throws IndexOutOfBoundsException {...}
```



Ejercicio 2 - Lanzamiento de Excepciones - throws

- ¿Cuál es la salida del siguiente programa?

```
class Demo {
static void demoproc() {
    try { throw new NullPointerException("demo");}
    catch(NullPointerException e) {
        S.O.P. ("Dentro de demo");      throw e;
    }
}
public static void main(String args[]) {
    try { demoproc();}
    catch(NullPointerException e) { S.O.P. ("Nueva captura: " +
e);}
}
}
```



Captura de Excepciones – Creación de Excepciones Propias

- Se pueden crear excepciones propias sólo con heredar de la clase *Exception* o de alguna de sus clases derivadas.
- Las clases suelen tener dos constructores:
 - Un constructor sin argumentos
 - Un constructor que recibe un String como argumento (mensaje que explica el tipo de excepción generada). Conviene que este constructor llame al constructor de la clase de la que deriva.



Captura de Excepciones – Creación de Excepciones Propias

```
void metodo1(){
    ...
    try {
        // Código que puede lanzar IOException y MyException
    } catch (IOException e1) { S.O.P.(e1.getMessage());
    } catch (MyException e2) { S.O.P.(e2.getMessage()); return;
    } finally { ...// Sentencias que se ejecutarán siempre
    }
    ...
} // Fin del metodo1
```



Ejercicio 3 – Tratamiento de Excepciones

- Construir un programa que divida dos números representados mediante caracteres.
- El programa debe contemplar:
 - Que el carácter se corresponde con un número
 - Que el denominador no sea nulo ni menor que -5
 - Que el numerador no sea mayor que 100

