



Excepciones

Presentación basada en:

1. Como Programar en Java. Deitel y Deitel. Ed. Prentice-Hall. 1988
2. Java 2., Curso de programación. Fco. Javier Ceballos. Ed. Alfoomega&RA-MA, 2003.
3. Apuntes del Curso Programación Orientado a Objetos. Pablo Castells. Escuela Politécnica Superior, Universidad Autónoma de Madrid.
4. Apuntes del Curso de java. Luis Hernández y Carlos Cervigón. Facultad de Informática. Universidad Católica de Madrid.

1



Introducción

- Las excepciones son la manera que ofrece un programa (en nuestro caso Java) de manejar los errores en tiempo de ejecución.
- Muchos lenguajes imperativos simplemente detienen la ejecución de programa cuando surge un error.
- Las excepciones nos permiten escribir código que nos permita manejar el error y continuar (si lo estimamos conveniente) con la ejecución del programa.
- Ejemplo de Error
 - El error de ejecución más clásico en Java es el de **desbordamiento**, el intento de acceso a una posición de un vector que no existe.

2



Introducción

```
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]);  
    }  
}
```

- Este programa tendrá un serio problema cuando intente acceder a **mensajes[3]**, pues no existe dicho valor.
- Al ejecutarlo mostrará lo siguiente:

*Primero
Segundo
Tercero*

*Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: at
Desbordamiento.main(Desbordamiento.java, Compiled Code)*

3



Estructura: try ... catch ... finally

- En el ejemplo del *desbordamiento*
 - Se detecta un error de ejecución (lanza una excepción) al intentar acceder a la posición inexistente.
 - Cuando se detecta el error, por defecto se interrumpe la ejecución. Esto se puede evitar.
- La estructura **try-catch-finally** nos permite capturar excepciones, es decir, reaccionar a un error de ejecución.
- De este modo se puede imprimir mensajes de error "a la medida" y continuar con la ejecución del programa si consideramos que el error no es demasiado grave.

4



Estructura: try ... catch ... finally

- Para ver el funcionamiento de la estructura *try-catch-finally*, modifiquemos el ejemplo anterior asegurando que se capturan las excepciones.

```
try {  
    // Código que puede hacer que se eleve la excepción  
}  
catch(TipoExcepcion e) {  
    // Gestor de la excepción  
}
```

- Java se comporta de la siguiente manera:
 - Si en la ejecución del código dentro del bloque **try** se eleva una excepción de tipo **Tipo Excepcion** (o descendiente de éste), Java omite la ejecución del resto del código en el bloque **try** y ejecuta el código situado en el bloque **catch** (gestor).

5



Estructura: try ... catch ... finally

```
public class EjemploCatch {  
    static String mensajes[] = {"Primero", "Segundo", "Tercero" };  
    public static void main(String[] args) {  
        try {  
            for(int i = 0; i <= 3; i++)  
                System.out.println(mensajes[i]);  
        }  
        catch ( ArrayIndexOutOfBoundsException e ) {  
            System.out.println("Se han desbordado los mensajes");  
        }  
        finally {  
            System.out.println("Ha finalizado la ejecución");  
        }  
    }  
}
```

- Dentro del bloque de la sección **try** se coloca el código normal.
- Después de la sección **try** se debe colocar:
 - Al menos una sección **catch** o una **finally**
 - Se pueden tener ambos e incluso más de una sección **catch**.

6



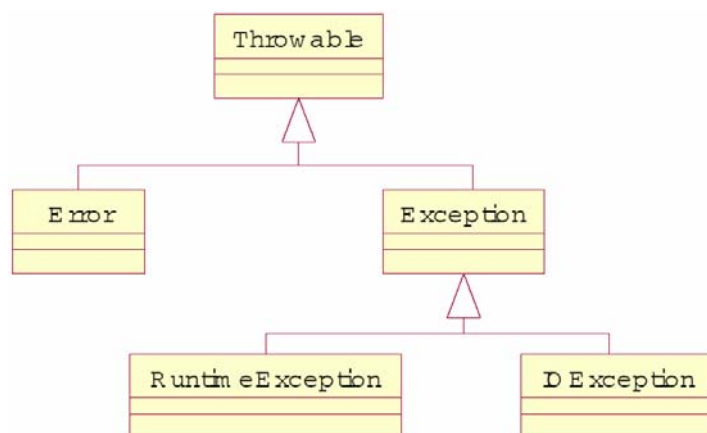
Estructura: try ... catch ... finally

- **try**
 - El bloque de código donde se prevé que se eleve una excepción.
 - Al encerrar el código en un bloque **try** es como si dijéramos: "Prueba a usar estas instrucciones y mira a ver si se produce alguna excepción".
 - El bloque **try** tiene que ir seguido, al menos, por una cláusula **catch** o una cláusula **finally**.
- **catch**
 - Es el código que se ejecuta cuando se eleva la excepción.
 - Controla cualquier excepción que cuadre con su argumento.
 - Se pueden colocar varios **catch** sucesivos, cada uno controlando un tipo de excepción diferente.
- **finally**
 - Bloque que se ejecuta siempre, haya o no excepción.
 - Existe cierta controversia sobre su utilidad, pero podría servir, por ejemplo, para hacer un seguimiento de lo que está pasando, ya que al ejecutarse siempre puede dejar grabado un registro (**log**) de las excepciones ocurridas y su recuperación o no.

7



Jerarquía de excepciones



8



Clases de excepciones

- **Throwable**
 - Superclase que engloba a todas las excepciones
- **Error**
 - Representa los errores graves causados por el sistema (JVM, ...)
 - No son tratados por los programas.
- **Exception**
 - Define las excepciones que los programas deberían tratar
 - (IOException, ArithmeticException, etcétera).

9



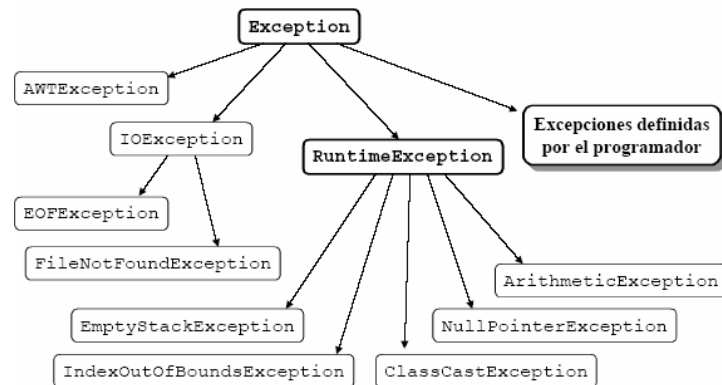
La clase Exception

- Cuando se eleva una excepción, lo que se hace es activar un ejemplar de **Exception** o de alguna de sus subclases.
- Normalmente las clases derivadas nos permiten distinguir entre los distintos tipos de excepciones.
- En el programa anterior, por ejemplo, en el bloque **catch** se captura una excepción del tipo *ArrayIndexOutOfBoundsException*, ignorando cualquier otro tipo de excepción.

10



Excepciones predefinidas



11

11



Captura de excepciones

- Al **catch** le sigue, entre paréntesis, la declaración de una excepción.
 - Es decir, el **nombre** de una **clase derivada de Exception** (o la propia `Exception`) seguido del **nombre** de una **variable**.
 - Si se lanza una excepción que es la que deseamos capturar (o una derivada de la misma) se ejecutará el código que contiene el bloque.
 - Ejemplo:
 - `catch(Exception e) { ... }`
 - se ejecutará siempre que se produzca una **excepción del tipo que sea**, ya que todas las excepciones se derivan de `Exception`.
 - **No es recomendable utilizar algo así**, ya que estaremos capturando cualquier tipo de excepción sin saber si eso afectará a la ejecución del programa o no.

12

Captura de excepciones

- Se pueden colocar varios bloques **catch**.
 - Si es así, se comprobará, en el mismo orden en que se encuentren esos bloques **catch**
 - Si la excepción elevada es la que se trata en el bloque **catch**; se dispara la excepción
 - Si no, se pasa a comprobar el siguiente.
 - **NOTA:** sólo se ejecuta un bloque **catch**. En cuanto se captura la excepción se deja de comprobar el resto de los bloques.
 - Por esta razón, el siguiente código no sería correcto:

```
catch(Exception e) {  
    ...  
}  
catch(DerivadaDeException e) {  
    ...  
}
```

- no es correcto, por que el segundo **catch** no se ejecutará nunca.

13

Ejemplo: Sin try-catch

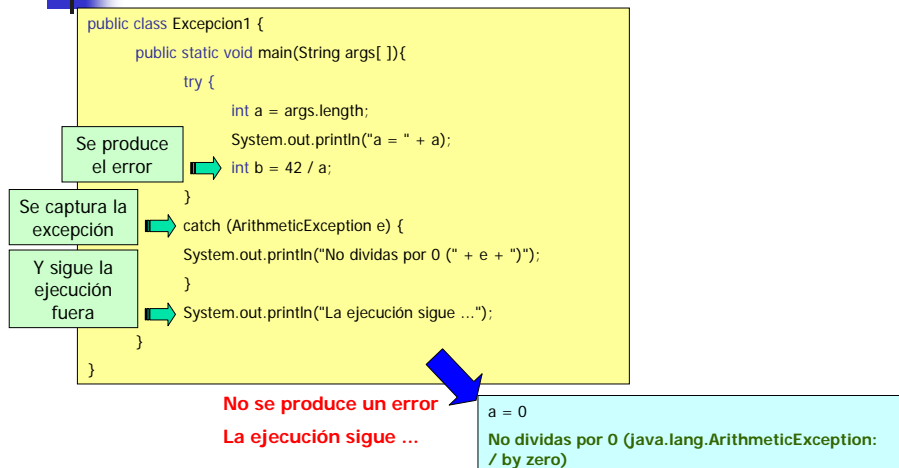
```
public class Excepcion1 {  
    public static void main(String args[ ]){  
        int a = args.length;  
        System.out.println("a = " + a);  
        ➡ int b = 42 / a;  
    }  
}
```

Se produce el error
y se interrumpe ...

```
a = 0  
java.lang.ArithmeticException: / by zero  
at Excepcion1.main(Excepcion1.java:6)  
Exception in thread "main" Process Exit...
```

14

Ejemplo: Con try-catch



15

Lanzamiento Explicito de Excepciones

- Los métodos en los que se puede producir un **error** deben avisar al compilador de que éste se puede producir.
 - Para ello se utiliza la cláusula **throws**.
- Ejemplo
 - un método de lectura de un archivo podría elevar una excepción de tipo **IOException**:

```
public String leer(FileInputStream archivo) throws IOException
{
    // ...
}
```
- Se pueden elevar varias excepciones en un mismo método:

```
public Image cargar(String s) throws EOFException,
    MalformedURLException
{
    // ...
}
```

16



Lanzamiento Explicito de Excepciones

- Las excepciones en los **métodos** se lanzan con la instrucción **throw**.
- Ejemplo
 - Cuando se está implementando un método que efectúa una lectura de un archivo de datos, y se llega inesperadamente a su final, se puede lanzar una **EOFException**:

```
public String leerDatos(DataInput archivo) throws EOFException {  
    // ...  
    while( /* ... */ ) {  
        if(ch == -1) { // Fin de archivo  
            if(n < longitud)  
                throw new EOFException();  
        }  
    }  
    return s;  
}
```

17



Excepciones definidas por el usuario

- El programador puede crear sus propias excepciones cuando ninguna de las predefinidas es adecuada.
- Pasos
 - Se define una clase que descende de **Exception** (o de la clase deseada).
 - Se suele agregar un constructor con el mensaje de la excepción, que se inicializa en el constructor llamando al de la clase padre.
 - Además, toda excepción tiene un método **getMessage()** que devuelve un **String** con el mensaje.

```
// Define una clase de excepción propia  
public class MiExcepcion extends Exception {  
    public MiExcepcion(){  
        super("error muy grave...");  
    }  
}
```

18



Excepciones definidas por el usuario

```
public class MiExcepcion extends Exception {
    public MiExcepcion(){
        super("error muy malo...");
    }
}

class UnaClase {
    public void metodo() throws MiExcepcion {
        System.out.println("Lanzo mi excepcion desde aqui.");
        throw new MiExcepcion();
    }
}

public class Excepcion4 {
    public static void main(String[] args) {
        UnaClase c = new UnaClase();
        try { c.metodo(); }
        // Invoco al método que eleva la excepción
        catch(MiExcepcion e) {
            System.out.println("La capture! " + e);
        }
        System.out.println("... y sigo.");
    }
}
```

```
C:\ARCHIV-1\XINOXS-1\JCREAT-2\GE2001.exe
Lanzo mi excepcion desde aqui.
La capture! MiExcepcion: error muy malo...
... y sigo.
Press any key to continue...
```

19



Ejemplo1

```
class Limites extends Exception {}
class demasiadoCalor extends Limites {}
class demasiadoFrio extends Limites {}
class demasiadoRapido extends Limites {}
class demasiadoCansado extends Limites {}

...
try {
    if(temp > 40 ) throw new demasiadoCalor();
    if(dormir < 8 ) throw new demasiadoCansado();
}
catch(Limites lim) {
    if(lim instanceof demasiadoCalor ) {
        System.out.println("Capturada excesivo calor!");
        return;
    }
    if( lim instanceof demasiadoCansado ) {
        System.out.println("Capturada excesivo cansancio!");
        return;
    }
}
finally System.out.println("En la cláusula finally"); ...
```

20

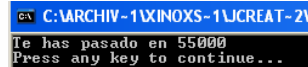


Ejemplo2

```
class NoHayDineroException extends Exception {
    private int dinero;
    public NoHayDineroException(int n) {
        super();
        dinero = n;
    }
    public int getDinero() { return dinero; }
}

class Cuenta {
    private int saldo;
    public Cuenta() { saldo = 0; }
    public Cuenta(int n) { saldo = n; }
    public void meterDinero(int n) { saldo += n; }
    public void sacarDinero(int n) throws NoHayDineroException {
        if(saldo > n) saldo -= n;
        else throw new NoHayDineroException(n - saldo);
    }
}

class Test {
    public static void main(String a[]) {
        Cuenta c = new Cuenta(5000);
        try {
            c.sacarDinero(60000);
        }
        catch(NoHayDineroException e) {
            System.out.println("Te has pasado en " + e.getDinero());
        }
    }
}
```



21



Las excepciones son parte de la interfaz de un objeto

- Si un método deja pasar una excepción, se debe declarar en la cabecera
- Solo los *ERROR'S* y las *RunTimeException'S* no requieren ser declarados
- Un método sobrescrito no puede declarar mas excepciones que (subclases de) las que declara la definición de una clase padre
- Si un método sobrescrito emite una excepción no declarada en el padre, es obligatorio procesarla aunque no se haga nada con ella.

```
class X extends Applet {
    public void start () { // start heredado de Applet
        try { ... } (catch IOException e) { /* vacio */ }
    }
}
```

22



Métodos de Throwable

- **Throwable (String):** Constructor que asigna un mensaje al objeto.
- **getMessage ():** Devuelve el mensaje del Objeto.
- **toString():** Devuelve un String incluyendo la clase del objeto más el mensaje.
- **printStackTrace():** Escribe la traza de ejecución en el standard error.
- Cuando una excepción no se procesa hasta el final, el programa se interrumpe y se ejecuta **printStackTrace()**.

23



Ventajas de las Excepciones

- Separación del tratamiento de errores del resto del código del programa.
 - Evitar manejo de códigos de error.
 - Evitar la alteración explícita del control de flujo.
- Propagación de errores a través de la pila de llamadas métodos.
 - Evitar el retorno de valores de error.
 - Evita la utilización de argumentos adicionales.
- Agrupamiento de tipos de errores, diferenciación de tipos de errores.
 - Jerarquía de clases de excepciones.
 - Tratar los errores a nivel de especificidad deseado.

24



Problema a resolver

- Indicar cuál es la salida del siguiente programa y explicar por qué.

```
class A {  
    public static void main (String args[]) throws X {  
        try { f (); throw new Z (); }  
        catch (Y ex) { System.out.println ("Y" + ex); }  
        catch (X ex) { System.out.println ("X" + ex); }  
    }  
    static f () throws X {  
        try { throw new Y (); } catch (X ex) { g (); }  
    }  
    static g () throws X {  
        try { throw new X (); } catch (Y ex) {}  
    }  
}  
class X extends Exception {  
    public String toString () { return "X"; }  
}  
class Y extends X {  
    public String toString () { return "Y"; }  
}  
class Z extends Y {  
    public String toString () { return "Z"; }  
}
```

25