

JAVA: Excepciones

Errores

- ❖ ¿ Qué ocurre si un programa está usando un archivo en CD y el usuario lo extrae antes de tiempo ?
- ❖ ¿ Qué ocurre si el usuario de un programa introduce un carácter por teclado, p.ej. 'W', cuando el programador ha previsto que se introduzca un número entero ?
- ❖ Las aplicaciones deben de ser robustas y saber manejar situaciones inesperadas (excepciones). En caso contrario:
 - ✓ El avión se cae en vuelo, el misil Tomahawk cae en la sede de la Cruz Roja, la nave espacial choca contra la superficie de Marte, etc....

El accidente del Ariane 5



Ariane 5 (Datos)

- El sistema de control de vuelo del Ariane 5 pertenece a un diseño estándar.
- Un Sistema de Referencia Inercial (SRI) mide la altitud del cohete y sus movimientos en el espacio.
- Los datos del SRI se transmiten a través de un bus de datos al Ordenador de a bordo (On-Board Computer OBC).
- El OBC ejecuta el programa de vuelo, controla las toberas de los tanques de combustible sólido y el motor criogénico Vulcain.

Ariane 5 (Datos)

- ❶ Para mejorar la fiabilidad hay una considerable redundancia de “maquinaria”.
- ❷ Por ejemplo hay dos SRIs operando en paralelo con idéntico hardware y software.
- ❸ Un SRI está activo mientras que el otro está en *stand-by*.
- ❹ Si el OBC detecta que el SRI activo ha fallado entonces utiliza inmediatamente el otro.
- ❺ Del mismo modo hay dos OBC, y otras unidades del Sistema de Control de Vuelo también están duplicadas.
- ❻ El diseño del SRI del Ariane 5 es prácticamente idéntico al del Ariane 4, especialmente en lo referido al software.

Ariane 5 (Vuelo 501)

- ❶ La lanzadera espacial comenzó a desintegrarse sobre H₀+39 segundos debido a la elevada carga aerodinámica causada por un ángulo de ataque de más de 20 grados que provocó la separación de los tanques lo que resultó en la activación del sistema de autodestrucción.
- ❷ El ángulo de ataque fue causado por una desviación en las toberas de los tanques de combustible sólido.
- ❸ Estas desviaciones fueron ordenadas por el software del OBC utilizando datos transmitidos por el SRI activo (SRI2). Parte de estos datos no contenían datos de vuelo sino un patrón de bits del ordenador del SRI2 que fue interpretado como datos de vuelo.

Ariane 5 (Vuelo 501)

- ❖ La razón de que el SRI2 no enviara datos correctos fue que la unidad había declarado un error debido a una **excepción software**.
- ❖ El OBC no pudo utilizar el SRI1 ya que esta unidad también había dejado de funcionar en el ciclo de datos anterior (72 milisegundos antes) por la misma razón que el SRI2.
- ❖ La excepción de software en el SRI fue causada durante la conversión de punto flotante de 64 bits a un valor entero con signo de 16 bits. El número en coma flotante que fue convertido tenía un valor mayor que el podía ser representado en un entero con signo de 16 bits. Resultando en un Operand Error (código Ada).

Ariane 5 (Vuelo 501)

- ❖ El Operand Error ocurrió debido a un valor inesperadamente elevado de una variable BH, Horizontal Bias, relacionada con la velocidad horizontal.
- ❖ El valor de BH era mucho más elevado que lo esperado porque la parte inicial de la trayectoria del Ariane 5 difiere de la del Ariane 4 y resulta en valores mayores de la velocidad horizontal.
- ❖ El procedimiento al encontrar un error establecía que (tratamiento de la excepción):
 - ✓ El fallo debe ser indicado en el bus de datos.
 - ✓ El contexto del fallo debe ser almacenado en una memoria EEPROM.
 - ✓ El procesador SRI debe apagarse.

¿Cómo se tratan los Errores?

• Opción 1:

No se tratan



Resultados impredecibles



• Opción 2:

Quien detecta el error decide qué hacer



Poco flexible

• Opción 3:

Quien detecta el error informa de que el error se ha producido.



Ejemplo de tratamiento (?)

```
public void desapilar () {  
    if ( this.esVacia() )  
        System.err.println ("Error: Desbordamiento inferior.");  
    else  
        laCima = laCima.siguiente();  
}
```

Detecta el posible error.

¿Trata el error?
¿Informa del error?
¿No hace nada?

Ejemplo de tratamiento (mejor)

```
public boolean desapilar () {  
    if ( this.esVacia() )  
        return ( false );  
    else {  
        laCima = laCima.siguiete();  
        return ( true );  
    }  
}
```

Detecta el posible error.

Informa del error:
Indica, a quien ha invocado
este método, que no ha
podido desapilar

Informar de un error NO quiere decir mostrar mensajes por pantalla.
Esto es útil durante el desarrollo pero no durante la ejecución.

Ejemplo (problema 1)

```
public boolean desapilar () {  
    if ( this.esVacia() )  
        return ( false );  
    else {  
        laCima = laCima.siguiete();  
        return ( true );  
    }  
}
```

¿Qué pasa si este método puede detectar varios tipos de error?
¿Cómo saber qué error se ha producido?

Ejemplo (solución 1 y problema 2)

```
public int desapilar () { //código  
}
```

¿Qué pasa si este método necesita devolver el dato desapilado?

No puede devolver 2 valores (error y dato).

Tratamiento adecuado: Excepciones

• Solución propuesta:

- ✓ Un método puede retornar dos tipos de información:
 - Datos (`return`)
 - Errores (`throw`)
- ✓ Analogía: `System.out` y `System.err` como dispositivos de salida con finalidad diferente.

• Los valores de retorno de error se denominan Excepciones.

Excepciones

- Una excepción es un problema que impide continuar con la ejecución de un método p.q. no tiene suficiente información para seguir adelante.
- Existen unos requisitos para que un método pueda funcionar correctamente, si estos requisitos no se cumplen se tiene una excepción.
- Ejemplo: Un método tiene como argumento un valor entero que representa un año.
 - ✓ Requisito: el valor del año no puede ser un valor entero negativo.
 - ✓ Excepción: si $\text{año} < 0$

Fases del tratamiento de Excepciones

1. Detectar e informar del error:

Lanzamiento de Excepciones → `throw`

Un método detecta una condición anormal que le impide continuar con su ejecución y finaliza "lanzando" un objeto Excepción.

2. Recoger el error y tratarlo:

Captura de Excepciones → `try-catch`

Un método recibe un objeto Excepción que le indica que otro método no ha terminado correctamente su ejecución y decide actuar en función del tipo de error.

Ejemplo

```
public double calculo ( double a, double b ) {  
    if ( (a-b) < 0 ) throw Excepcion  
    else  
        return sqrt (a-b);  
}
```

Detección y lanzamiento de la Excepción

```
try {  
    res = calculo (x,y);  
}  
catch ( Excepcion ) {  
    res = calculo (y,x);  
}  
System.out.print ( res );
```

Aplicación 1: tratamiento "ligero"

```
try {  
    res = calculo (a,b);  
}  
catch ( Excepcion ) {  
    System.exit(-1);  
}
```

Aplicación 2: tratamiento "fatal"

Lanzamiento de Excepciones

- Se puede lanzar una excepción mediante la sentencia:

```
throw referencia
```

Donde la referencia corresponderá a un objeto de alguna subclase de la clase `Exception`.

- El lanzamiento de una excepción implica que el método que se esté ejecutando finalice inmediatamente.
- Si un método puede lanzar una excepción hay que avisarlo:

```
public void desapilar () throws Exception  
{ //codigo }
```

Ojo,
"throws" = "lanza"
"throw" = "lanzar"

Subclases de `Exception`
que pueden ser lanzadas

Ejemplo desapilar ()

```
public void desapilar () throws Exception {  
    if ( this.esVacia() )  
        throw ( new Exception ("Pila vacia") );  
    else  
        laCima = laCima.siguiente();  
}
```

Lanza excepciones de tipo Exception

Lanzar un objeto de la clase Exception

Ejemplo cima ()

```
public Object cima () {  
    if ( this.esVacia() ) {  
        System.err.println ("Error: Desbordamiento inferior.");  
        return ( new Object() );  
    }  
    else  
        return ( laCima.dato() );  
}
```

Versión original, sin excepciones.

```
public Object cima () throws Exception {  
    if ( this.esVacia() )  
        throw ( new Exception ("Pila vacia") );  
    else  
        return ( laCima.dato() );  
}
```

Nueva versión, lanza excepciones.

Captura de Excepciones

- **Recoger la excepción:** Para poder recoger y tratar una excepción es preciso declarar un bloque `try` :

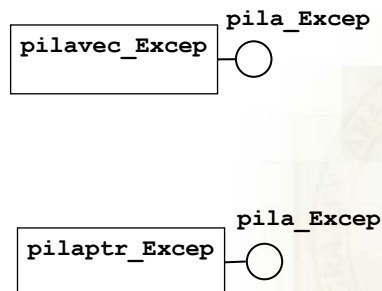
```
try {  
    //sentencias que pueden lanzar excepciones  
}
```

- **Si se produce una excepción en el bloque `try`, ésta es capturada y pasa a ser tratada**

- **El tratamiento de una excepción se realiza mediante “funciones” `catch`:**

```
catch (Exception e) {  
    //sentencias que tratan la excepción e  
}
```

Ejemplo: Pilas con Excepciones



Ejemplo Test

```
class test_pila_Excep {  
  
    public static void main (String[] args) {  
  
        pila_Excep mi_pila;  
  
        mi_pila = new pilavec_Excep();  
        try {  
            for (int veces = 0; veces < 2; veces++) {  
                for (int i = 1; i < 110; i++) //110 para generar error  
                    mi_pila.apilar ( new Integer (i) );  
                imprimir ( mi_pila );  
                mi_pila = new pilaptr_Excep();  
            }  
        }  
        catch ( Exception e ) {  
            System.err.println ();  
            System.err.println (e);  
        }  
  
        System.out.println("FIN.");  
    }  
}
```

Captura

Tratamiento

Ejemplo Test (cont.)

```
private static void imprimir ( pila_Excep p ) throws Exception {  
  
    System.out.print ("Contenido de la pila ");  
    if ( p instanceof pilavec_Excep )  
        System.out.println ("(con vectores):");  
    else  
        System.out.println ("(enlazada):");  
  
    while ( ! p.esVacia() ) {  
        System.out.print ( p.cima() + "," );  
        p.desapilar ();  
    }  
    System.out.println ();  
}  
  
} //fin clase
```

No hay captura ni tratamiento, las Excepciones son “relanzadas” tal y como llegan.
Hay que avisar del lanzamiento.

Crear nuevos tipos de Excepciones

- El nombre de la clase de la Excepción informa del problema.
- Si hay nuevos problemas puede convenir definir nuevas clases para las Excepciones.

```
class PilaLlenaException extends Exception {  
  
    PilaLlenaException ( Object x ) {  
        super ("Pila Llena");  
        dato = x;  
    }  
  
    public void apilar ( Object x ) throws PilaLlenaException {  
  
    }  
  
    public Object valor () {  
        return ( dato );  
    }  
  
    private Object dato;  
}
```

Nuevo tipo de Excepción

Finalidad: Almacenar el dato que genera el problema.

Revisión del Test

```
public static void main (String[] args) {  
    pila_Excep mi_pila;  
  
    mi_pila = new pilavec_Excep();  
    try {  
        for (int veces = 0; veces < 2; veces++) {  
            for (int i = 1; i < 110; i++ ) //110 para generar error  
                mi_pila.apilar ( new Integer (i) );  
            imprimir ( mi_pila );  
            mi_pila = new pilaptr_Excep();  
        }  
    }  
    catch ( PilaLlenaException e ) {  
        System.err.println ( e + "generada por: " + e.valor();  
        e.printStackTrace();  
    }  
    catch ( Exception e ) {  
        System.err.println (e);  
    }  
    finally {  
        System.out.println("Espero que llegue aqui. FIN.");  
    }  
}
```

¡Orden!

Traza de los métodos por los que ha pasado

finally: Se ejecuta haya o no Excepción.

Ejecución del Test

```
try {
    for (int veces = 0; veces < 2; veces++) {
        for (int i = 1; i < 110; i++) //110 para generar error
            mi_pila.apilar ( new Integer (i) );
        imprimir ( mi_pila );
        mi_pila = new pilaptr_Excep();
    }
}
catch ( PilaLlenaException e ) {
    System.err.println ( e + "generada por: " + e.valor();
    e.printStackTrace();
}
catch ( Exception e ) {
    System.err.println (e);
}
finally {
    System.out.println("Espero que llegue aqui. FIN.");
}
```

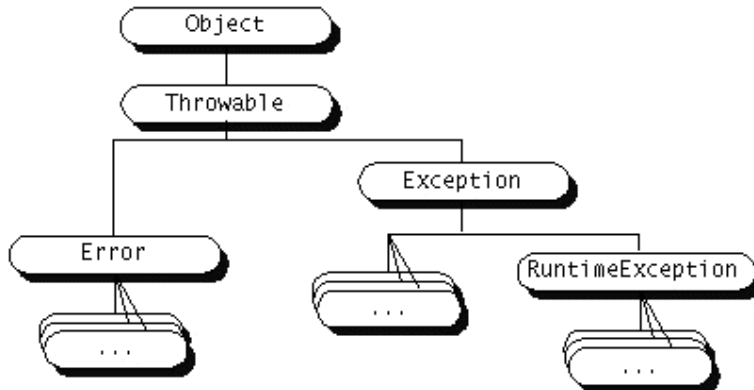
Lanza Excepción → fin try

PilaLlenaException: Pila Llena generada por: 101
PilaLlenaException: Pila Llena
at pilavec_Excep.apilar(pilavec_Excep.java:17)
at test_pila_Excep.main(test_pila_Excep.java:11)
Espero que llegue aqui. FIN.

Relevancia de las excepciones

- Cualquier excepción lanzada por un método es una parte pública de él: los usuarios del método deben conocer qué excepciones puede lanzar para poder decidir conscientemente qué hacer con ellas.
- Las excepciones que puede lanzar un método tienen el mismo nivel de importancia que los argumentos o el valor de retorno del método.
- Las excepciones deben de ser documentadas.

Jerarquía de clases “lanzables”



RuntimeException

- Las excepciones predefinidas de Java que derivan de la clase `RuntimeException` tienen un tratamiento especial.
- Las RTE representan problemas detectados por el sistema RT (Runtime System). Incluye:
 - ✓ Excepciones aritméticas (división por cero)
 - ✓ Excepciones de punteros: `NullPointerException`
 - ✓ Excepciones de indexación (índices inválidos en un array): `ArrayIndexOutOfBoundsException`
- Normalmente, el coste de comprobar RTE excede el beneficio de capturarlas o especificarlas. Por esa razón, el compilador no exige tratar las RTE :
 - ✓ Avisar (`throws`) de que un método lanza una RTE.
 - ✓ Tratar explícitamente (`catch`) una RTE.

Ventajas de las Excepciones

- Separar el Código que maneja los errores del Código “normal”
- Propagar los errores en la pila de llamadas
- Agrupar y diferenciar los tipos de errores

```
class Uno{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("42");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        }catch (NumberFormatException e){
            valor = valor + Integer.parseInt("42");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch (Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}
```

Diagrama de flujo de ejecución:

1. Inicio del método `metodo()` y declaración de `valor=0`.
2. Entrada al bloque `try`.
3. Salida del bloque `finally`.
4. Salida del método `metodo()` y entrada a `main`.
5. Ejecución de `metodo()` dentro de `main`.
6. Captura y manejo de la excepción en `main`.


```

class Uno{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("42");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        }catch (NumberFormatException e){
            valor = valor + Integer.parseInt("42");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch (Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

Diagram illustrating the execution flow of the `Uno` class:

- 1. `try` block starts, `valor` becomes 44.
- 2. `catch` block is executed, `valor` becomes 45.
- 3. `finally` block is executed, `valor` becomes 46.
- 4. `return` statement is executed, `valor` is 46.
- 5. `main` method calls `metodo()`, `valor` is 46.
- 6. `main` method catches the exception and prints the stack trace.

```

class Dos{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("W");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        }catch (NumberFormatException e){
            valor = valor + Integer.parseInt("42");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch (Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

Diagram illustrating the execution flow of the `Dos` class:

- 1. `try` block starts, `valor` becomes 1.
- 2. `catch` block is executed, `valor` becomes 2.
- 3. `finally` block is executed, `valor` becomes 3.
- 4. `return` statement is executed, `valor` is 4.
- 5. `main` method calls `metodo()`, `valor` is 5.
- 6. `main` method catches the exception and prints the stack trace.

```

class Dos{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("W");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        }catch(NumberFormatException e){
            valor = valor + Integer.parseInt("42");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch(Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

Diagram illustrating the execution flow of the `Dos` class:

- 1. `try` block starts, `valor` becomes 1.
- 2. `try` block continues, `valor` becomes 2.
- 3. `try` block ends, `valor` becomes 3.
- 4. `catch` block starts, `valor` becomes 4.
- 5. `finally` block starts, `valor` becomes 5.
- 6. `finally` block ends, `valor` becomes 6.
- 7. `return` statement is executed, `valor` becomes 7.
- 8. `main` method prints the result, `valor` becomes 8.

```

class Tres{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("W");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        }catch(NumberFormatException e){
            valor = valor + Integer.parseInt("W");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch(Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

Diagram illustrating the execution flow of the `Tres` class:

- 1. `try` block starts, `valor` becomes 1.
- 2. `try` block continues, `valor` becomes 2.
- 3. `try` block ends, `valor` becomes 3.
- 4. `catch` block starts, `valor` becomes 4.
- 5. `finally` block starts, `valor` becomes 5.
- 6. `finally` block ends, `valor` becomes 6.
- 7. `return` statement is executed, `valor` becomes 7.
- 8. `main` method prints the result, `valor` becomes 8.

```

class Tres{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("W");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        }catch (NumberFormatException e){
            valor = valor + Integer.parseInt("W");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch (Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

```

class Cuatro{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("W");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
            throw new IOException();
        }catch (IOException e){
            valor = valor + Integer.parseInt("42");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch (Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

```

class Cuatro{
    private static int metodo(){
        int valor=0;
        try{
            valor = valor +1;
            valor = valor + Integer.parseInt("W");
            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
            throw new IOException();
        }catch(IOException e){
            valor = valor + Integer.parseInt("42");
            System.out.println("Valor al final del catch: " + valor);
        }finally{
            valor = valor + 1;
            System.out.println("Valor al final de finally: " + valor);
        }
        valor = valor + 1;
        System.out.println("Valor antes del return: " + valor);
        return valor;
    }

    public static void main(String[] args){
        try{
            System.out.println(metodo());
        }catch(Exception e){
            System.err.println("Excepcion en metodo()");
            e.printStackTrace();
        }
    }
}

```

1

2

3

4

5

6

2