

13

Manejo de errores

fundacion@proydesa.org) has a
this Student Guide.

Informe de excepciones

Error de código:

```
int[] intArray = new int[5];  
intArray[5] = 27;
```

Salida en la consola:

```
Exception in thread "main"  
    java.lang.ArrayIndexOutOfBoundsException: 5  
        at TestErrors.main(TestErrors.java:17)
```

Informe de excepciones

Código de llamada en `main()`:

```
TestArray myTestArray = new TestArray(5);  
myTestArray.addElement(5, 23);
```

Clase `TestArray`:

```
public class TestArray {  
    int[] intArray;  
    public TestArray (int size) {  
        intArray = new int[size];  
    }  
    public void addElement(int index, int value) {  
        intArray[index] = value;  
    }  
}
```

Devolución de excepciones

Ejecución normal del programa:

1. El método de llamada llama al método de trabajo.
2. El método de trabajo no funciona.
3. El método de trabajo termina el trabajo y, a continuación, la ejecución se devuelve al método de llamada.

Cuando se produce una excepción, la siguiente secuencia cambia:

- La excepción se devuelve y:
 - Se transfiere un objeto Exception especial a un bloque catch similar a un método especial en el método actual o bien
 - La ejecución se devuelve al método de llamada

Tipos de excepciones

Existen tres tipos principales de objetos Throwable:

- Error
 - Normalmente, un error externo irrecuperable
 - No comprobada
- RuntimeException
 - Normalmente, un error de programación
 - No comprobada
- Exception
 - Error recuperable
 - Comprobada (capturada o devuelta)

OutOfMemoryError

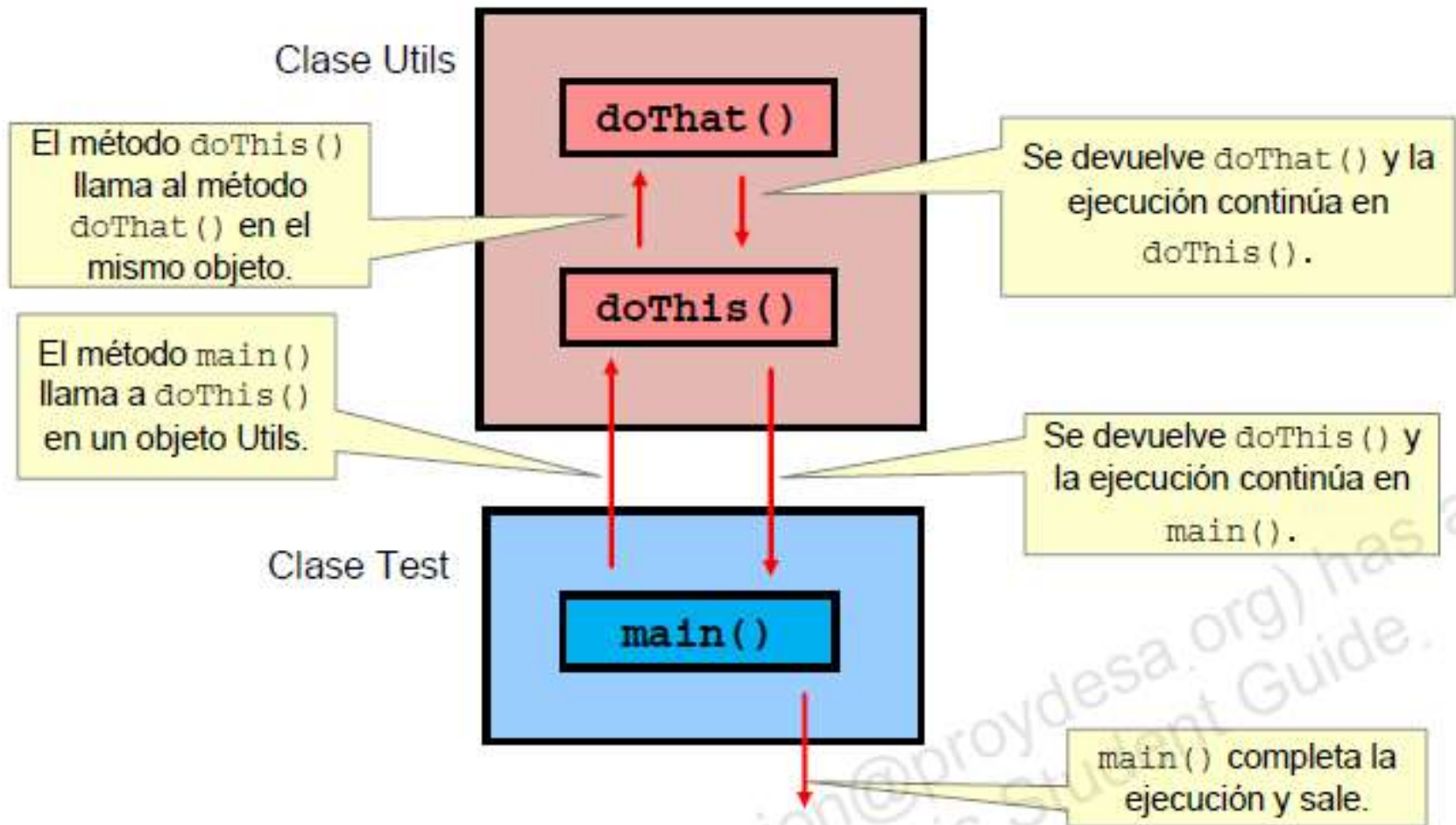
Error de programación:

```
ArrayList theList = new ArrayList();
while(true) {
    String theString = "A test String";
    theList.add(theString);
    if (theList.size()% 1000000 == 0) {
        System.out.println("List now has " +
            theList.size()/100000 + " million elements!");
    }
}
```

Salida en la consola:

```
List now has 240 million elements!
List now has 250 million elements!
Exception in thread "main" java.lang.OutOfMemoryError: Java
    heap space
```


Pila de métodos



Pila de llamadas: Ejemplo

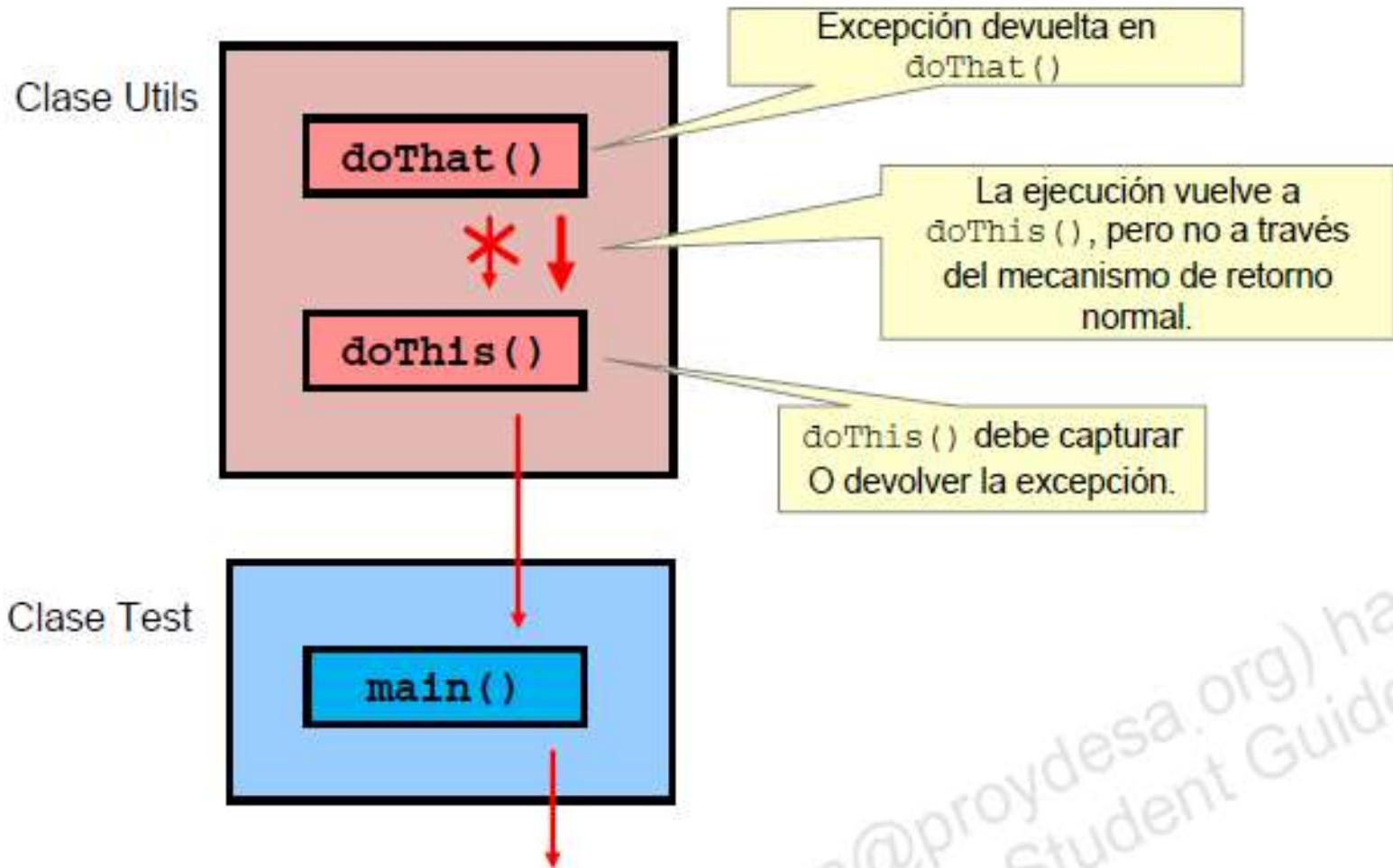
Clase Test:

```
public static void main (String args[]) {  
    Utils theUtils = new Utils();  
    theUtils.doThis();  
}
```

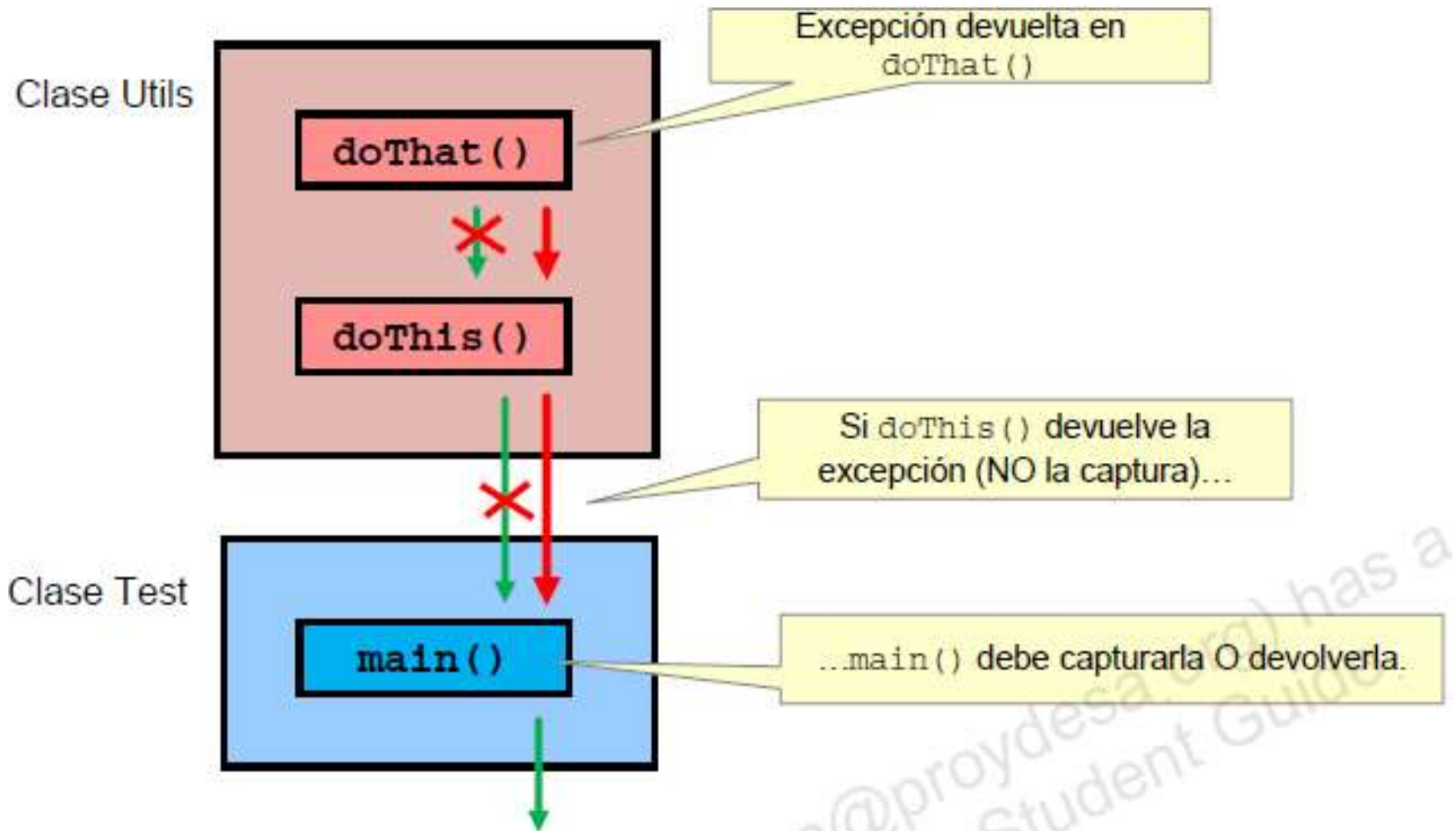
Clase Utils:

```
public void doThis() {  
    ...< code to do something >...  
    doThat();  
    return;  
  
    public void doThat() throws Exception{  
        ...< code to do something >...  
        if (some_problem) throw new Exception();  
        return;  
    }
```


Devolución de objetos Throwable



Devolución de objetos Throwable



Captura de una excepción

```
12 public void doThis() {  
13  
14     System.out.println("Arrived in doThis()");  
15     doThat();  
16     // unreported exception java.lang.Exception;  
17     // must be caught or declared to be thrown  
18  
19     // (Alt-Enter shows hints)  
20     public void doThat() throws Exception {  
21         System.out.println("In doThat()");  
22         throw new Exception();  
23     }  
24 }  
25
```

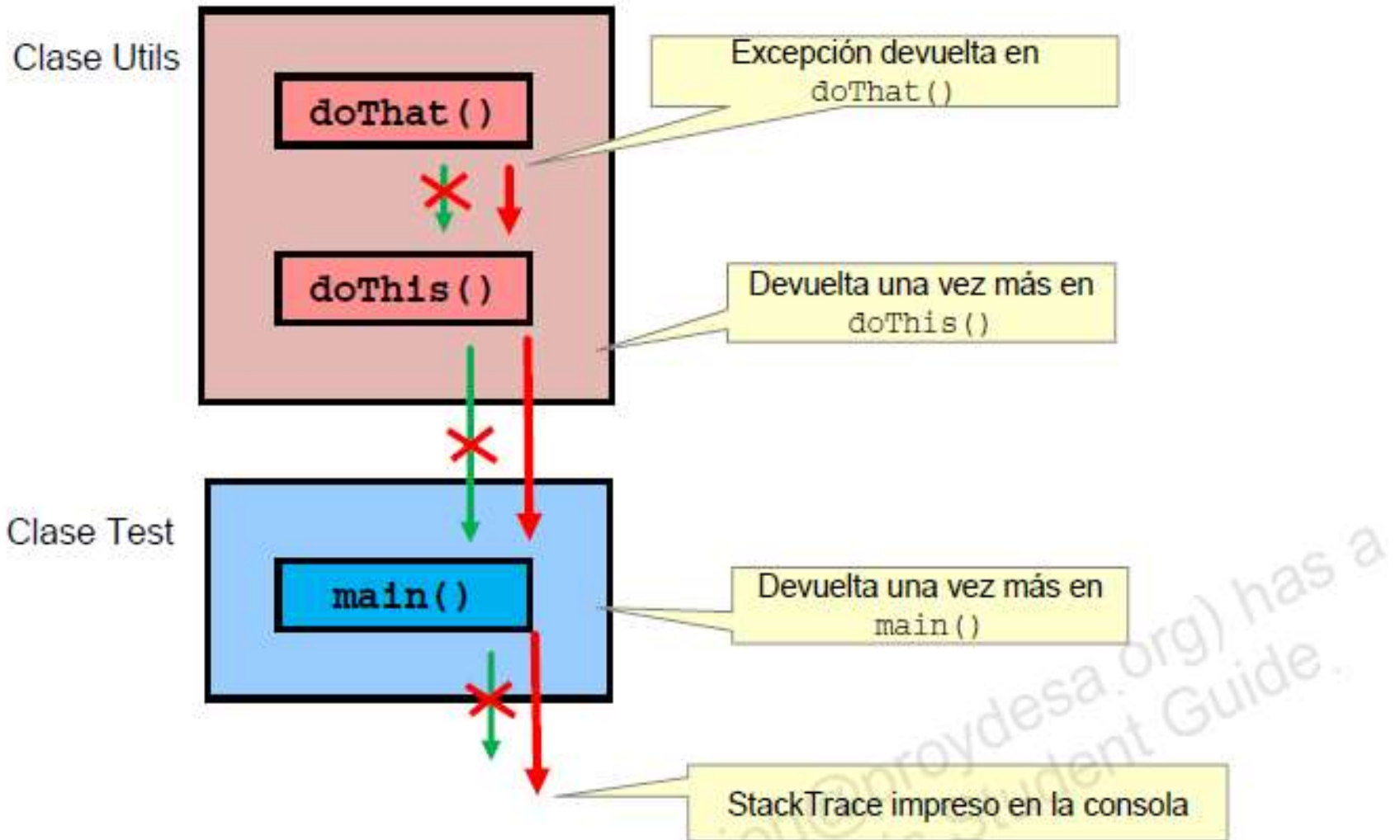
Ahora se debe manejar
la excepción en
doThis().

doThat ()
ahora devuelve
una excepción.

El bloque try/catch
captura una excepción y
la maneja.

```
12 public void doThis() {  
13  
14     System.out.println("Arrived in doThis()");  
15     try {  
16         doThat();  
17     }  
18     catch (Exception e) {  
19         System.out.println(e);  
20     }  
21     System.out.println("Back in doThis()");  
22  
23  
24  
25     public void doThat() throws Exception {  
26         System.out.println("In doThat()");  
27         throw new Exception();  
28     }  
29 }
```

Excepción no resuelta



Excepción impresa en la consola

Ejemplo de `main()` devolviendo una excepción.

```
10 public class Test {
11
12     public static void main (String args[] throws Exception {
13
14         System.out.println("Started in main()");
15         Utils myUtils = new Utils();
16         myUtils.doThis();
17         System.out.println("Back in main()");
18     }
19
20 }
```

Output - TestCode (run)

```
run:
Started in main()
Arrived in doThis()
In doThat()
Exception in thread "main" java.lang.Exception
    at Utils.doThat(Utils.java:27)
    at Utils.doThis(Utils.java:16)
    at Test.main(Test.java:16)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

`main()` ahora está definido para devolver una excepción.

Puesto que `main()` devuelve la excepción, ahora imprime una pila de llamadas en la consola.

Resumen de los tipos de excepciones

Throwable es un tipo especial de objeto Java:

- Es el único tipo de objeto que se utiliza como argumento en una cláusula catch.
- Es el único tipo de objeto que se puede “devolver” al método de llamada.
- Tiene dos subclases:
 - Error
 - Si se crea, se devuelve automáticamente al método de llamada.
 - Exception
 - Se debe devolver explícitamente al método de llamada.
 - - Se obtiene mediante el uso de un bloque try/catch.
 - Tiene una subclase RuntimeException que se devuelve automáticamente al método de llamada.

Excepciones en la documentación de la API de Java

Method Summary

Methods

Modifier and Type	Method and Description
boolean	<code>canExecute()</code> Tests whether the application can execute the file denoted by this abstract pathname.
boolean	<code>canRead()</code> Tests whether the application can read the file denoted by this abstract pathname.
boolean	<code>canWrite()</code> Tests whether the application can modify the file denoted by this abstract pathname.
int	<code>compareTo(File pathname)</code> Compares two abstract pathnames lexicographically.
boolean	<code>createNewFile()</code> Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.

Haga clic para
obtener los detalles
de
`createNewFile()`.

Observe las
excepciones
que se pueden
devolver.

createNewFile

```
public boolean createNewFile()
    throws IOException
```

Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist. The check for the existence of the file and the creation of the file if it does not exist are a single operation that is atomic with respect to all other filesystem activities that might affect the file.

Note: this method should not be used for file-locking, as the resulting protocol cannot be made to work reliably. The `FileLock` facility should be used instead.

Returns:

true if the named file does not exist and was successfully created; false if the named file already exists

Throws:

`IOException` - If an IO error occurred

`SecurityException` - If a security manager exists and its `SecurityManager.checkWrite(java.lang.String)` method denies write access to the file

Since:

1.2

Llamada a un método que devuelve una excepción

```
31  
32 public static void testCheckedException() {  
33  
34     File testFile = new File("//testFile.txt");  
35  
36     System.out.println("File exists: " + testFile.exists());  
37     testFile.delete();  
38     System.out.println("File exists: " + testFile.exists());  
39  
40 }  
41
```

El constructor no causa ningún problema de compilación.

```
31  
32 public static void testCheck  
33  
34     File testFile = new File  
35     testFile.createNewFile();  
36  
37     System.out.println("File exists: " + testFile.exists());  
38     testFile.delete();  
39     System.out.println("File exists: " + testFile.exists());  
40  
41 }
```

unreported exception java.io.IOException;
must be caught or declared to be thrown
(Alt-Enter shows hints)

createNewFile() puede devolver una excepción comprobada, por lo que debe devolverla o capturarla.

Trabajar con una excepción comprobada

Captura de IOException:

```
public static void main(String args[]) {  
    try {  
        testCheckedException();  
    }  
    catch (IOException e) {  
        System.out.println(e);  
    }  
}  
  
public static void testCheckedException() throws  
IOException{  
    File testFile = new File("//testFile.txt");  
    testFile.createNewFile();  
    System.out.println("File exists: " + testFile.exists());  
}
```

Prácticas recomendadas

- Capture la excepción real devuelta, no la excepción ni la superclase Throwable.
- Examine la excepción para conocer el problema exacto y así poder realizar la recuperación correctamente.
- No es necesario que capture todas las excepciones.
 - Un error de programación no se debe manejar. Se debe corregir.
 - Debe preguntarse si esta excepción representa el comportamiento del que desea que se recupere el programa.

Prácticas no recomendadas

```
public static void main (String args[]) {  
    try {  
        createFile("c:/testFile.txt");  
    }  
    catch (Exception e) {  
        System.out.println("Problem creating the file!");  
        ...< other actions >...  
    }  
}
```

¿Se captura
la superclase?

¿Se está procesando el objeto Exception?

```
public static void createFile(String fileName) throws  
    IOException {  
    File f = new File(fileName);  
    f.createNewFile();  
  
    int[] intArray = new int[5];  
    intArray[5] = 27;  
}
```


Prácticas no recomendadas

```
public static void main (String args[]) {  
    try {  
        createFile("c:/testFile.txt");  
    }  
    catch (Exception e) {  
        System.out.println(e);  
        ...< other actions >...  
    }  
}
```

¿Cuál es el
tipo de
objeto?

Se llama a
toString() en
este objeto.

```
public static void createFile(String fileName) throws  
IOException {  
    File f = new File(fileName);  
    System.out.println(fileName + " exists? " + f.exists());  
    f.createNewFile();  
    System.out.println(fileName + " exists? " + f.exists());  
    int[] intArray = new int[5];  
    intArray[5] = 27;  
}
```


Varias excepciones

El directorio debe permitir la escritura (IOException).

```
public static void createFile() throws IOException {  
    File testF = new File("c:/notWriteableDir");  
    File tempF = testFile.createTempFile("te", null, testF);  
    System.out.println("Temp filename: " +  
        tempFile.getPath());  
  
    int myInt[] = new int[5];  
    myInt[5] = 25;  
}
```

El índice de matriz debe ser válido (ArrayIndexOutOfBoundsException).

El argumento debe tener tres o más caracteres (IllegalArgumentException).

Captura de IOException

```
public static void main (String args[]) {  
    try {  
        createFile();  
    }  
    catch (IOException ioe) {  
        System.out.println(ioe);  
    }  
}  
  
public static void createFile() throws IOException {  
  
    File testF = new File("c:/notWriteableDir");  
    File tempF = testFile.createTempFile("te", null, testF);  
    System.out.println("Temp filename is " +  
        tempFile.getPath());  
    int myInt[] = new int[5];  
    myInt[5] = 25;  
}
```

Captura de IllegalArgumentException

```
public static void main (String args[]) {  
    try {  
        createFile();  
    }  
    catch (IOException ioe) {  
        System.out.println(ioe);  
    } catch (IllegalArgumentException iae) {  
        System.out.println(iae);  
    }  
}
```

```
public static void createFile() throws IOException {  
  
    File testF = new File("c:/writeableDir");  
    File tempF = testFile.createTempFile("te", null, testF);  
    System.out.println("Temp filename is " + tempFile.getPath());  
    int myInt[] = new int[5];  
    myInt[5] = 25;  
}
```

Captura de las excepciones restantes

```
public static void main (String args[]) {  
    try {  
        createFile();  
    }  
    catch (IOException ioe) {  
        System.out.println(ioe);  
    } catch (IllegalArgumentException iae) {  
        System.out.println(iae);  
    } catch (Exception e) {  
        System.out.println(e);  
    }  
}  
  
public static void createFile() throws IOException {  
    File testF = new File("c:/writeableDir");  
    File tempF = testFile.createTempFile("te", null, testF);  
    System.out.println("Temp filename is " + tempFile.getPath());  
    int myInt[] = new int[5];  
    myInt[5] = 25;  
}
```