Handling Exceptions

Objectives

After completing this lesson, you should be able to:

- Describe how Java handles unexpected events in a program
- List the three types of `Throwable` classes
- Determine what exceptions are thrown for any foundation class
- Describe what happens in the call stack when an exception is thrown and not caught
- Write code to handle an exception thrown by the method of a foundation class

Topics

- Handling exceptions: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

What Are Exceptions?

Java handles unexpected situations using exceptions.

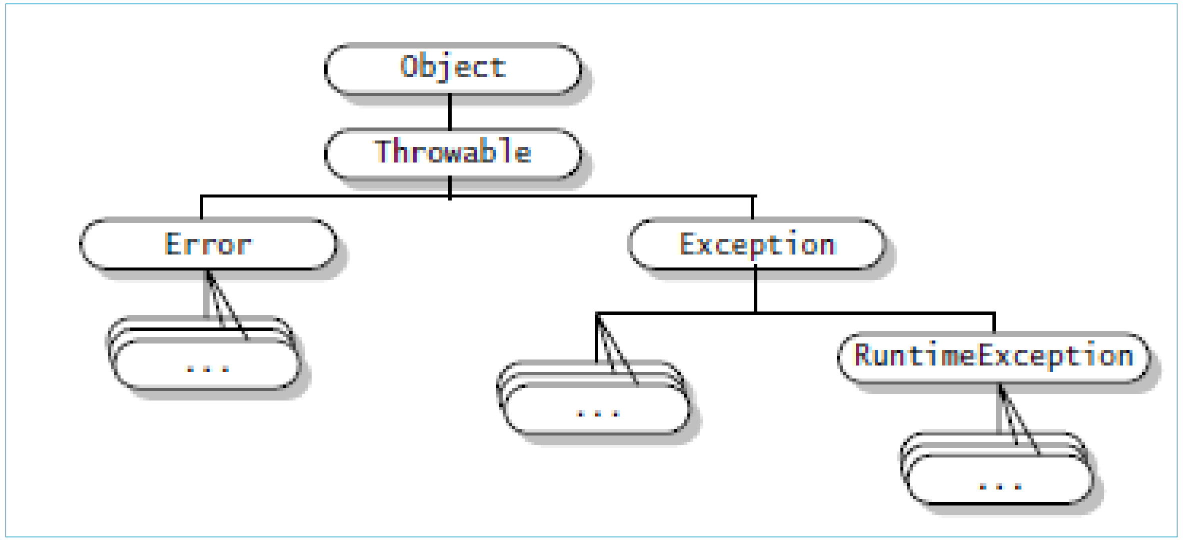
- Something unexpected happens in the program.
- Java doesn’t know what to do, so it:
  - Creates an exception object containing useful information and
  - Throws the exception to the code that invoked the problematic method
- There are several different types of exceptions.

Excepciones

Las excepciones son la manera que ofrece Java de manejar los errores en tiempo de ejecución.

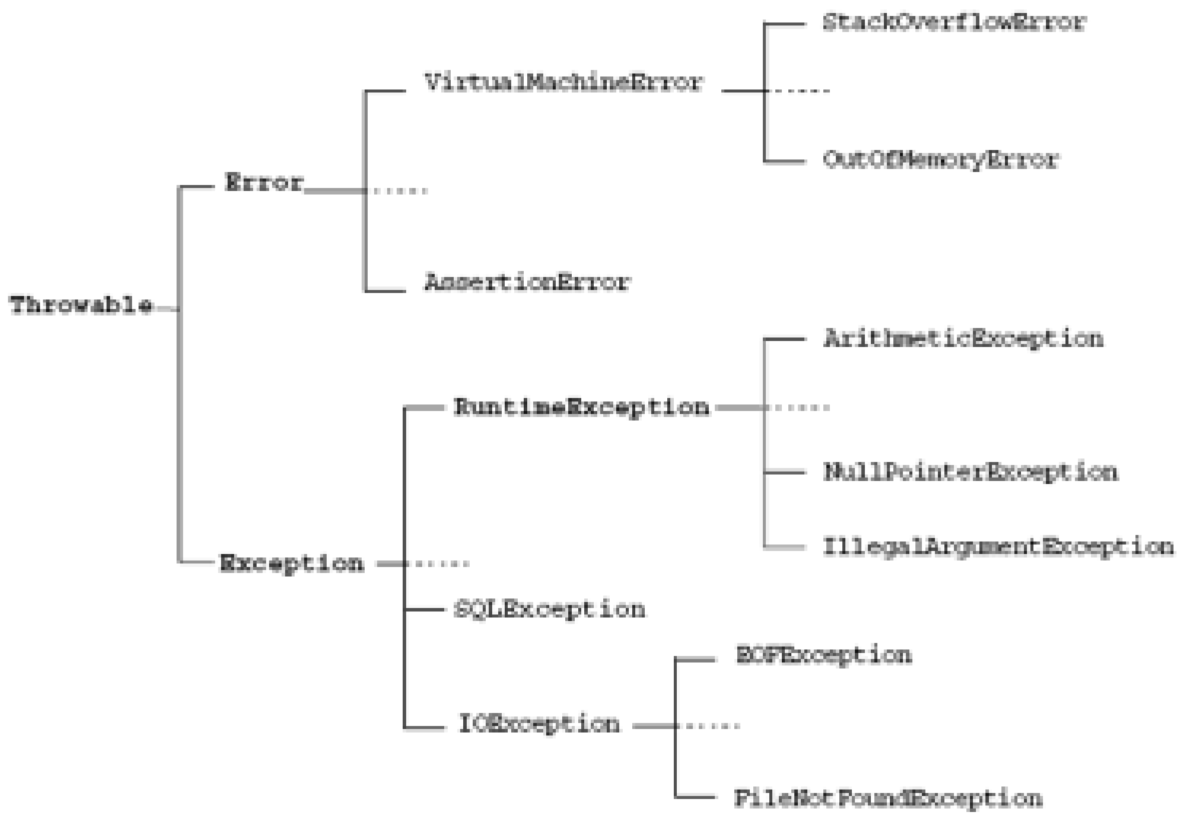
Las excepciones nos permiten indicar el código que se ejecutará en el caso de producirse un error y continuar con la ejecución del programa, si lo estimamos conveniente.

Excepciones



Throwable	**Superclase que engloba a todas las excepciones.**
Error	**Representa a los errores graves provocados por el sistema: OutOfMemoryError, InternalError, UnknownError,…**
Exception	**Define las excepciones que los programas deberían controlar obligatoriamente, salvo que hereden de RuntimeException**

Categorias de Exception



76

Examples of Exceptions

- `java.lang.ArrayIndexOutOfBoundsException`
  - Attempt to access a nonexistent array index
- `java.lang.ClassCastException`
  - Attempt to cast on object to an illegal type
- `java.lang.NullPointerException`
  - Attempt to use an object reference that has not been instantiated
- You can create exceptions, too!
  - An exception is just a class.

```
public class MyException extends Exception { }
```

Code Example

Coding mistake:

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

Output:

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

Another Example

Calling code in `main`:

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

`TestArray` class:

```
13 public class TestArray {
14   int[] intArray;
15   public TestArray (int size) {
16     intArray = new int[size];
17   }
18   public void addElement(int index, int value) {
19     intArray[index] = value;                     }
20 }
```

Stack trace:

```
    Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: 5
        at TestArray.addElement(TestArray.java:19)
        at TestException.main(TestException.java:20)
    Java Result: 1
```

Types of `Throwable` classes

Exceptions are subclasses of `Throwable`. There are three main types of `Throwable`:

- `Error`
  - Typically an unrecoverable external error
  - Unchecked
- `RuntimeException`
  - Typically caused by a programming mistake
  - Unchecked
- `Exception`
  - Recoverable error
  - Checked *(Must be caught or thrown)*

Error Example: `OutOfMemoryError`

Programming error:

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

Output in console:

```
List now has 156 million elements!
List now has 157 million elements!
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

Quiz

Which of the following objects are checked exceptions?

a. All objects of type `Throwable`

b. All objects of type `Exception`

c. All objects of type `Exception` that are not of type `RuntimeException`

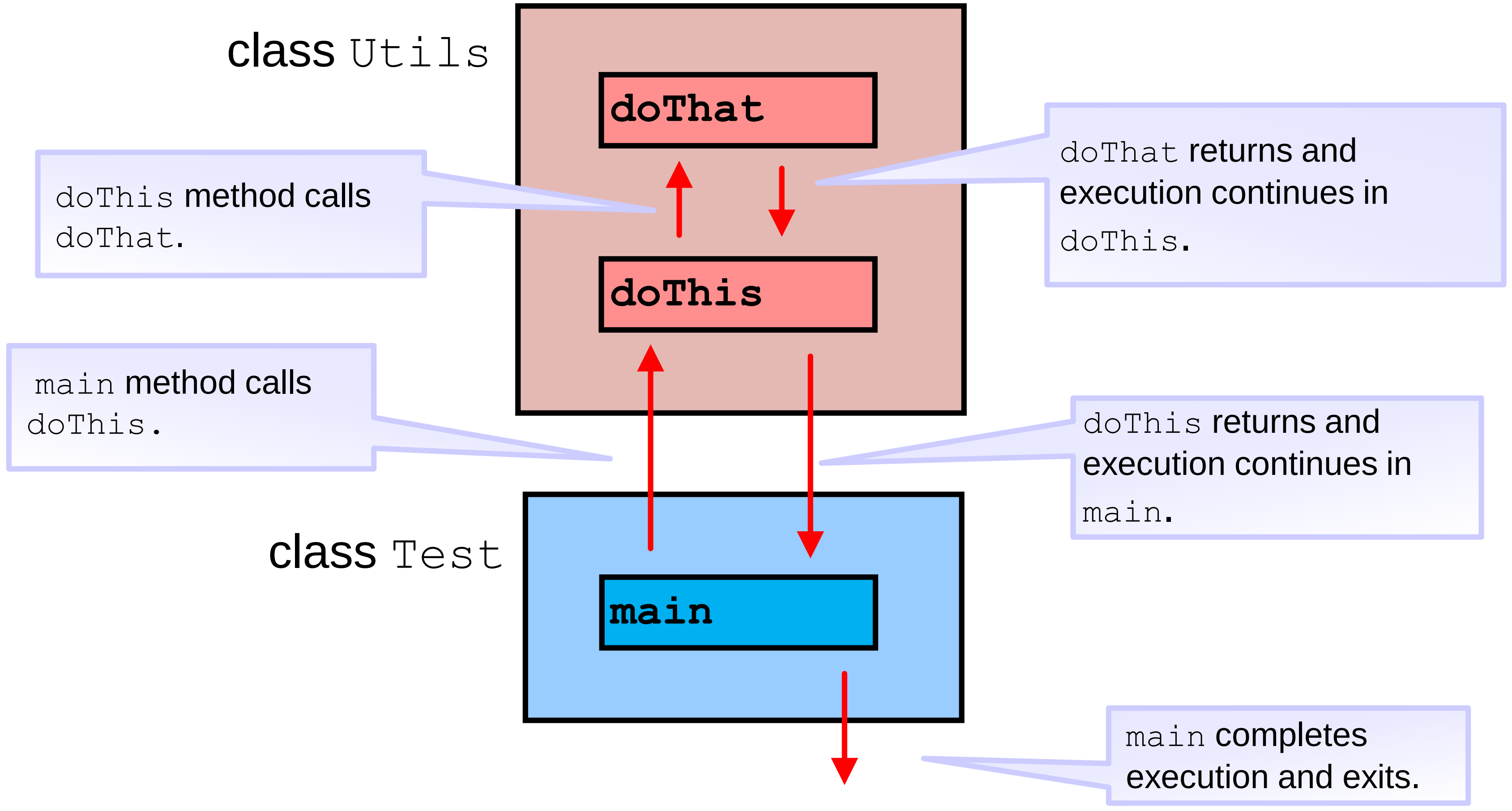
d. All objects of type `Error`

e. All objects of type `RuntimeException`

Topics

- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Normal Program Execution: The Call Stack


class Utils
doThat
doThis
class Test
main
doThis method calls doThat.
doThat returns and execution continues in doThis.
main method calls doThis.
doThis returns and execution continues in main.
main completes execution and exits.

How Exceptions Are Thrown

Normal program execution:

1. Caller method calls worker method.
2. Worker method does work.
3. Worker method completes work and then execution returns to caller method.

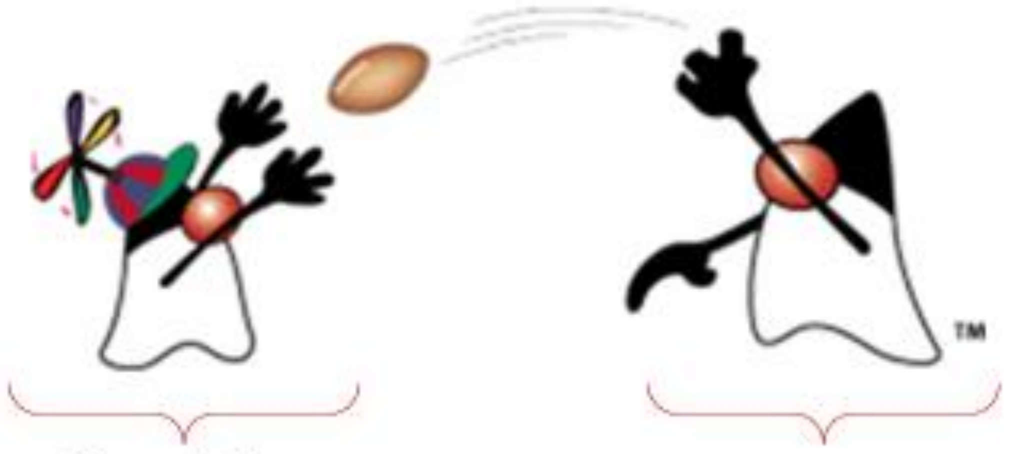
When an exception occurs, this sequence changes. An exception object is thrown and either:

- Passed to a `catch` block in the current method

or

- Thrown back to the caller method

THROWS



Try - catch

Intentando atrapar ....

?

77

Topics

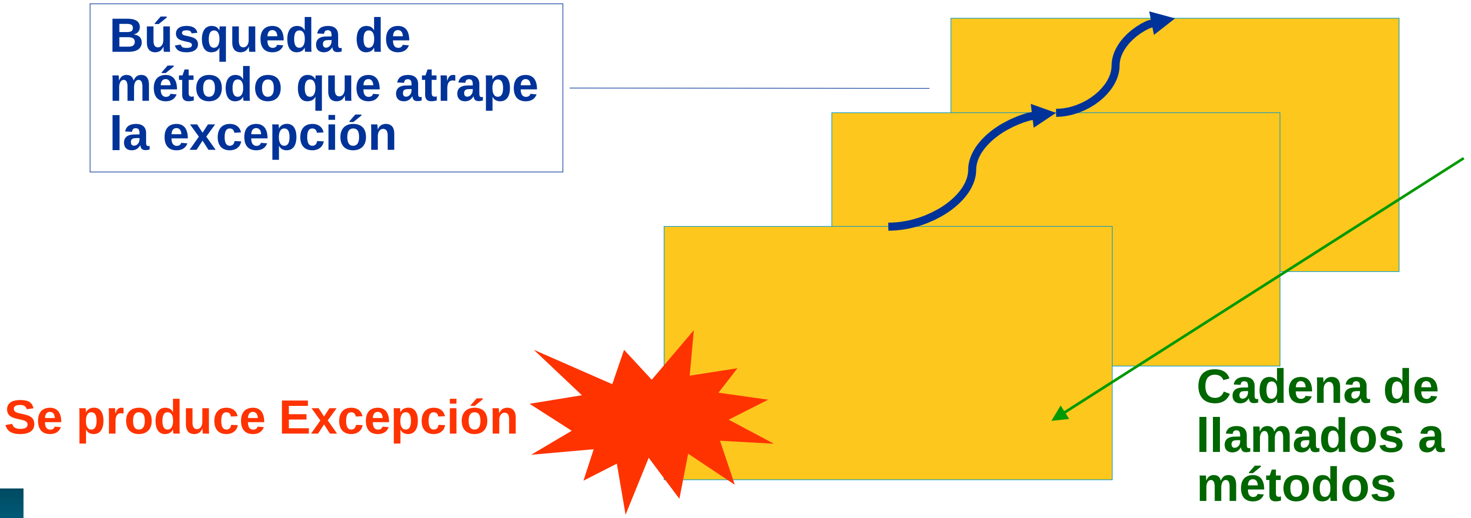
- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Excepciones

- En un método, qué se puede hacer si se produce una excepción?

Atrapar la excepción y manejarla

No calentarse, que se preocupe otro (método)


Búsqueda de método que atrape la excepción
Se produce Excepción
Cadena de llamados a métodos

Excepciones

Bloque try – catch – finally

Atrapar la excepción y manejarla

Se utiliza para detectar cuándo se produce una excepción y en tal caso, indicar las instrucciones que se deban ejecutar.

Si en la ejecución del código dentro del bloque try se produce una excepción de tipo TipoExcepcion (o descendiente de éste),

- **se interrumpe la ejecución del resto del código en el bloque try**
- **se ejecuta el código situado en el bloque catch**

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

Excepciones

Bloque try – catch – finally

Atrapar la excepción y manejarla

```
public class EjemploCatch {
  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("El asunto se nos ha desbordado");
    }
    finally {
      System.out.println("Ha finalizado la ejecución");
    }
  }
}
```

Excepciones

Atrapando Excepciones

Atrapar la excepción y manejarla

Se pueden colocar varios bloques catch. Se comprobará si matchea en el mismo orden en que se encuentren esos bloques catch

Sólo se ejecuta un bloque catch. En cuanto se captura la excepción se deja de comprobar el resto de los bloques.

Incorrrrrrecto

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

Excepciones

Ejemplo

Atrapar la excepción y manejarla

```
public class Excepcion1 {
  public static void main(String args[ ]){
    try {
      int a = args.length;
      System.out.println("a = " + a);
      int b = 42 / a;
    }
    catch (ArithmeticException e) {
      System.out.println("No dividas por 0 (" + e + ")");
    }
    System.out.println("La ejecución sigue ...");
  }
}
```

Se produce el error

Se captura la excepción

Y sigue la ejecución !

Excepciones

No calentarse, que se preocupe otro

Cláusula throws

Salvo que la exception herede de RuntimeException, el compilador exige que si un método no la atrapa DEBE decir explícitamente que no se calienta. Más formalmente, que continúa propagando esa Exception.

```
public String leer(FileInputStream archivo) throws IOException
{
  // ...
```

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

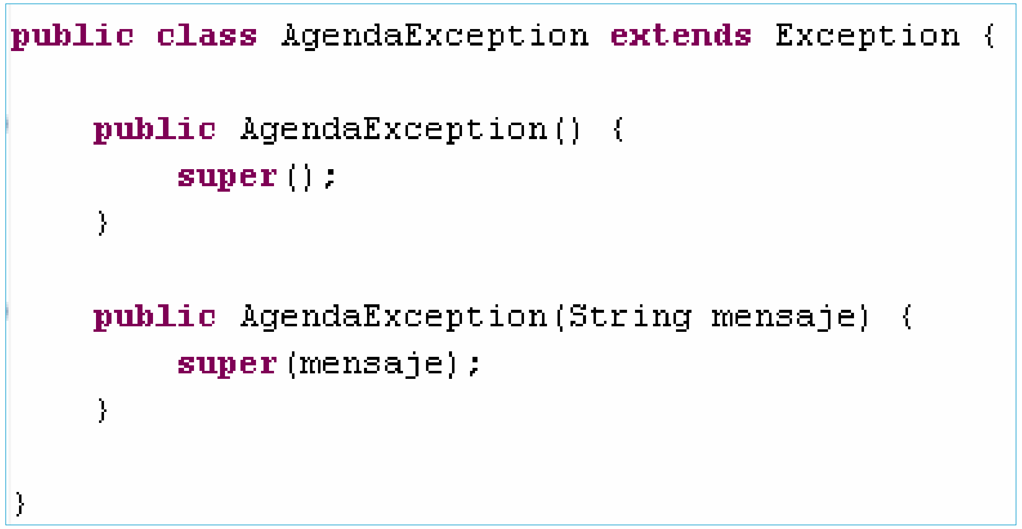
Excepciones

Lanzar 'manualmente' excepciones

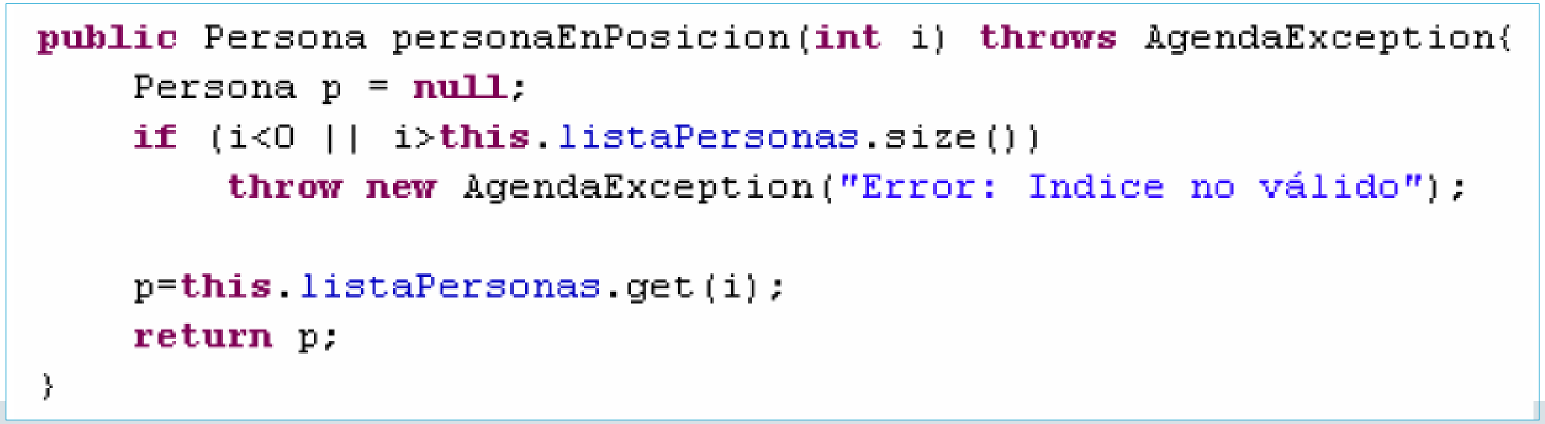
Las excepciones se lanzan utilizando la sentencia throw.

```
public Persona personaEnPosicion(int i) throws IndexOutOfBoundsException{
    Persona p = null;
    if (i<0 || i>this.listaPersonas.size())
        throw new IndexOutOfBoundsException();

    p=this.listaPersonas.get(i);
    return p;
}
```

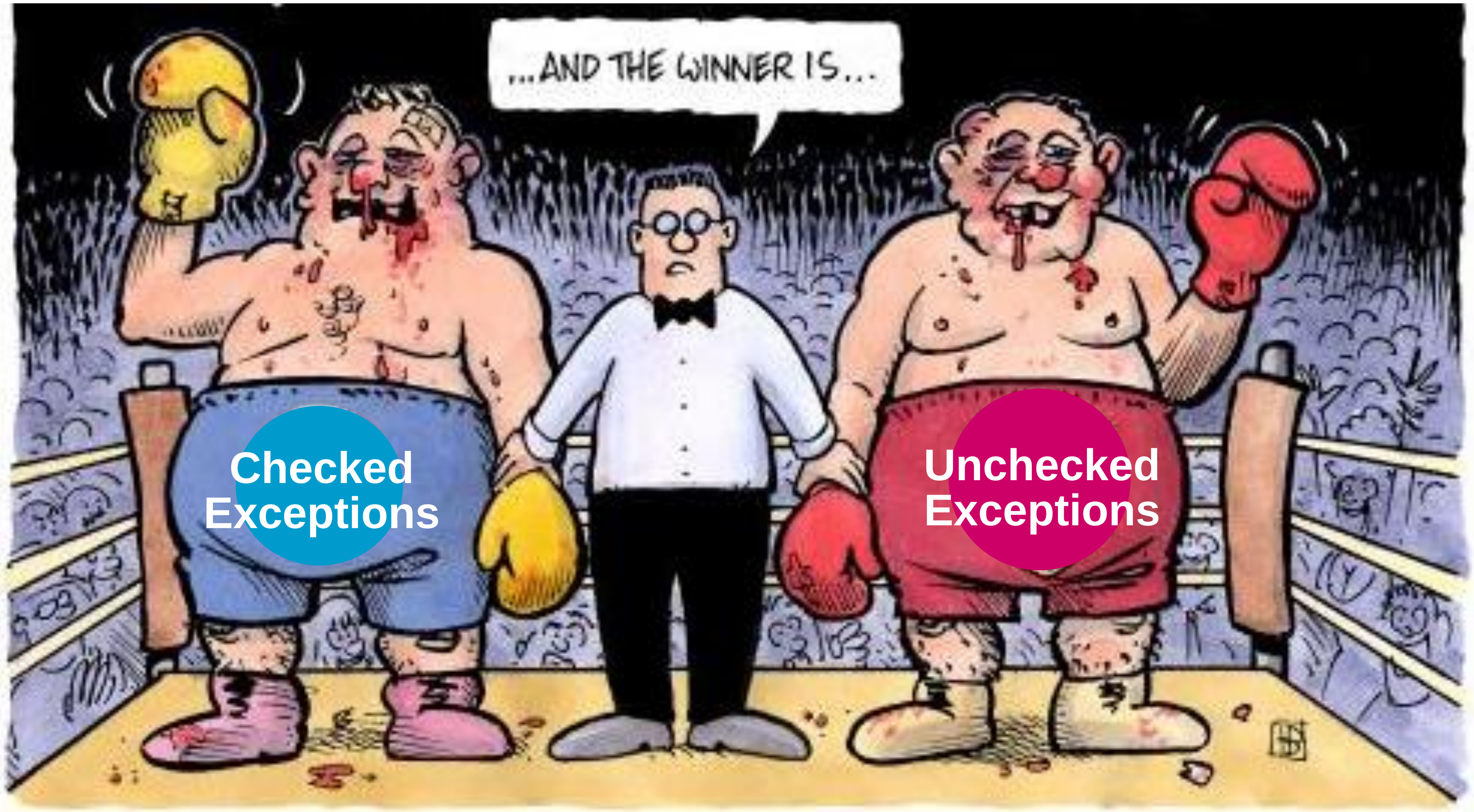
Excepciones definadas por el usuario



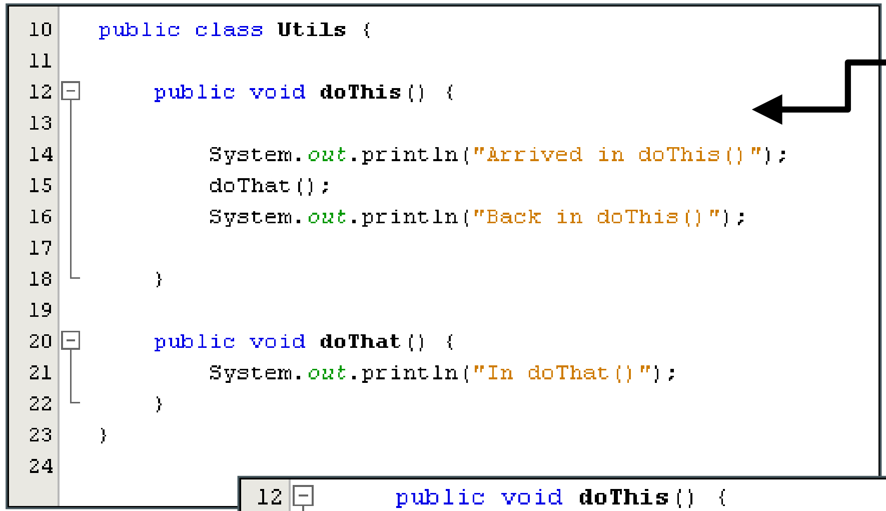
```
public class AgendaException extends Exception {

    public AgendaException() {
        super();
    }

    public AgendaException(String mensaje) {
        super(mensaje);
    }

}
```



```
public Persona personaEnPosicion(int i) throws AgendaException{
    Persona p = null;
    if (i<0 || i>this.listaPersonas.size())
        throw new AgendaException("Error: Indice no válido");

    p=this.listaPersonas.get(i);
    return p;
}
```

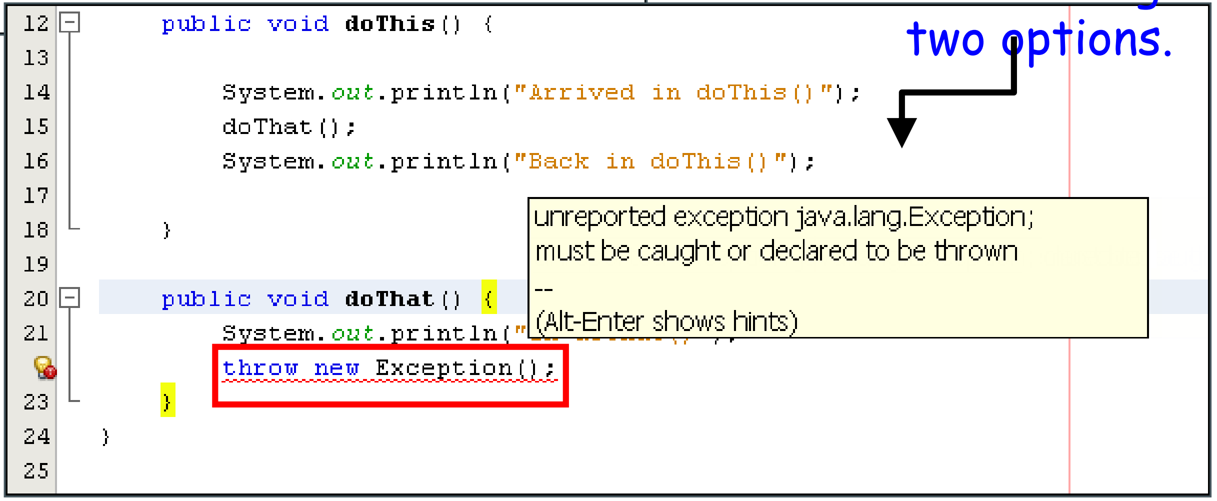

...AND THE WINNER IS...
Checked Exceptions
Unchecked Exceptions

Working with Exceptions in NetBeans



```
public class Utils {

    public void doThis() {

        System.out.println("Arrived in doThis()");
        doThat();
        System.out.println("Back in doThis()");

    }

    public void doThat() {
        System.out.println("In doThat()");
    }
}

```

No exceptions thrown; nothing needs be done to deal with them.



```
    public void doThis() {

        System.out.println("Arrived in doThis()");
        doThat();
        System.out.println("Back in doThis()");

    }

    public void doThat() {
        System.out.println("In doThat()");
        throw new Exception();
    }
}

```

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

When you throw an exception, NetBeans gives you two options.

The `try/catch` Block

Option 1: Catch the exception.

```
try {
   // code that might throw an exception
   doRiskyCode();
}
catch (Exception e){
    String errMsg = e.getMessage();
   // handle the exception in some way
}
```

`try` block

`catch` block

Option 2: Throw the exception.

```
public void doThat() throws Exception{
   // code that might throw an exception
   doRiskyCode();
}
```

Program Flow When an Exception Is Caught

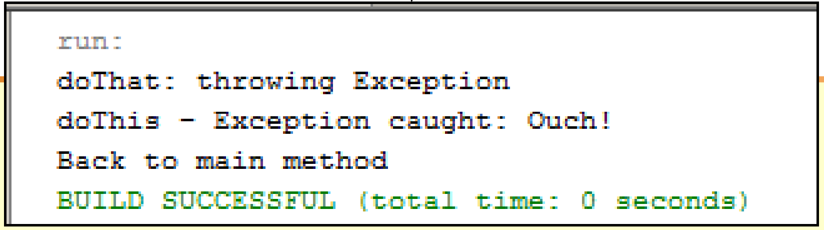
`main` method:

```java
Utils theUtils = new Utils();
theUtils.doThis();
System.out.println("Back to main method");
```

`Utils` class methods:

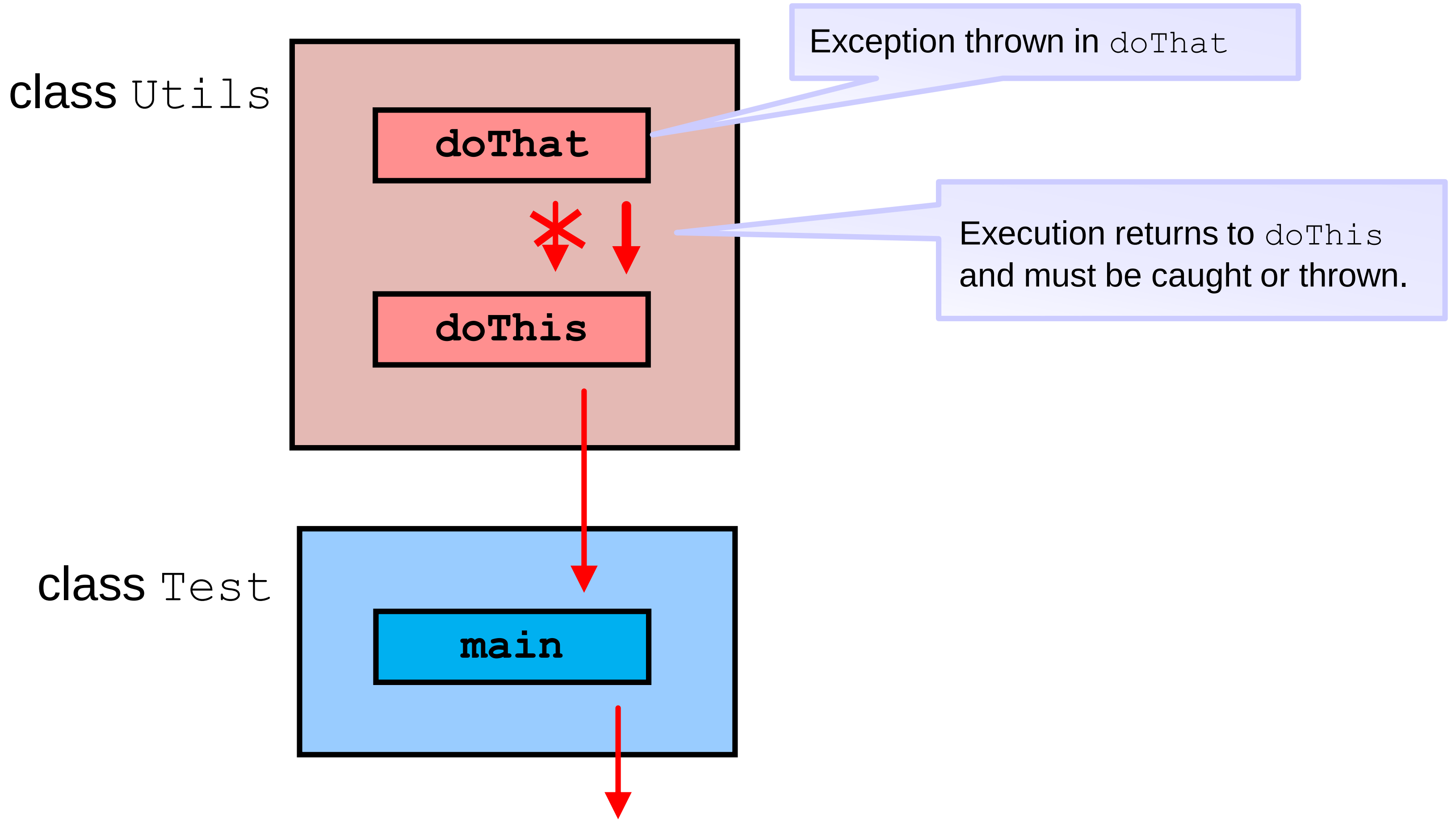
```java
public void doThis() {
  try{
      doThat();
  }catch(Exception e){
    System.out.println("doThis - "
     +" Exception caught: "+e.getMessage());
  }
}
public void doThat() throws Exception{
   System.out.println("doThat: Throwing exception");
   throw new Exception("Ouch!");
}
```

Output

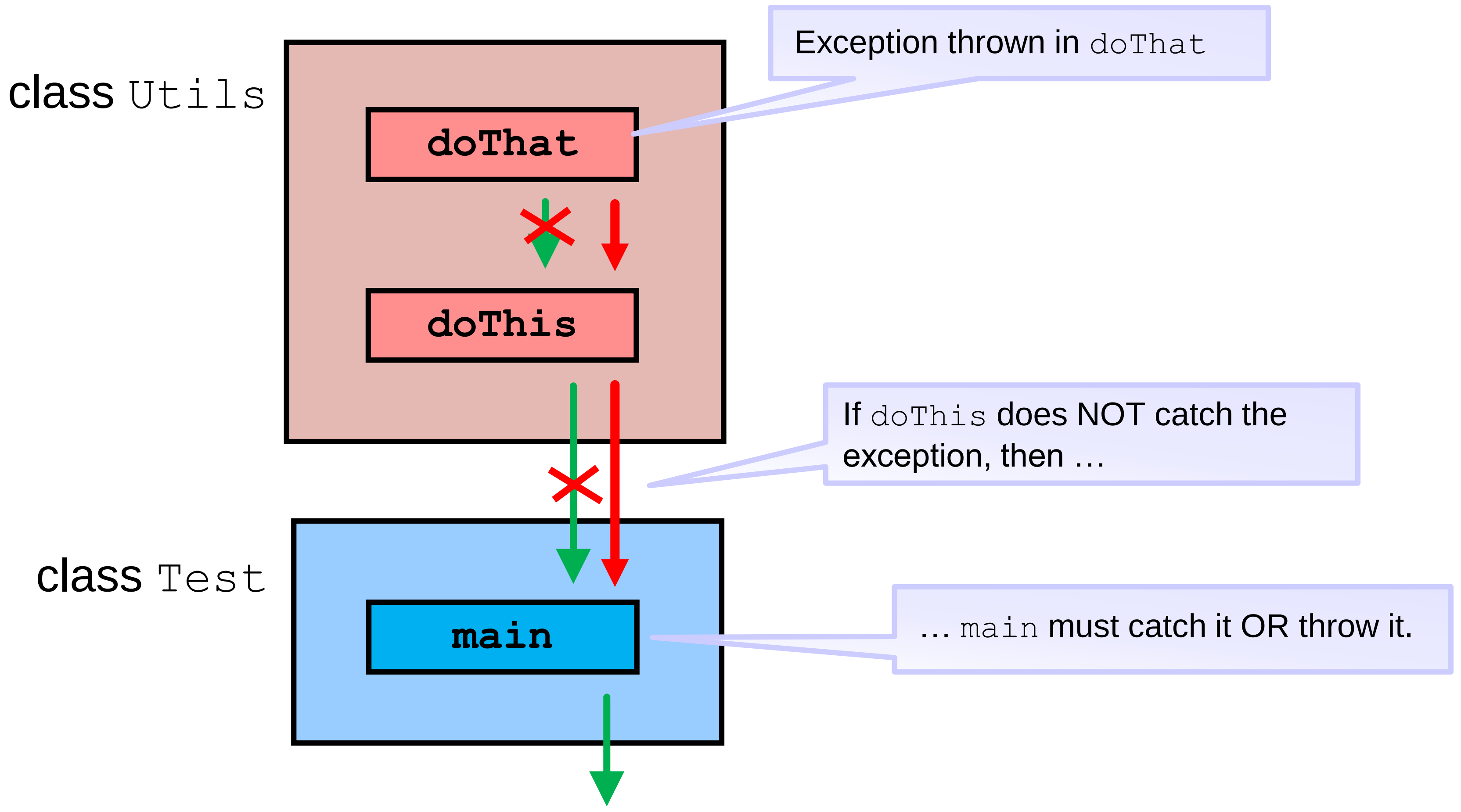


```
run:
doThat: throwing Exception
doThis - Exception caught: Ouch!
Back to main method
BUILD SUCCESSFUL (total time: 0 seconds)
```

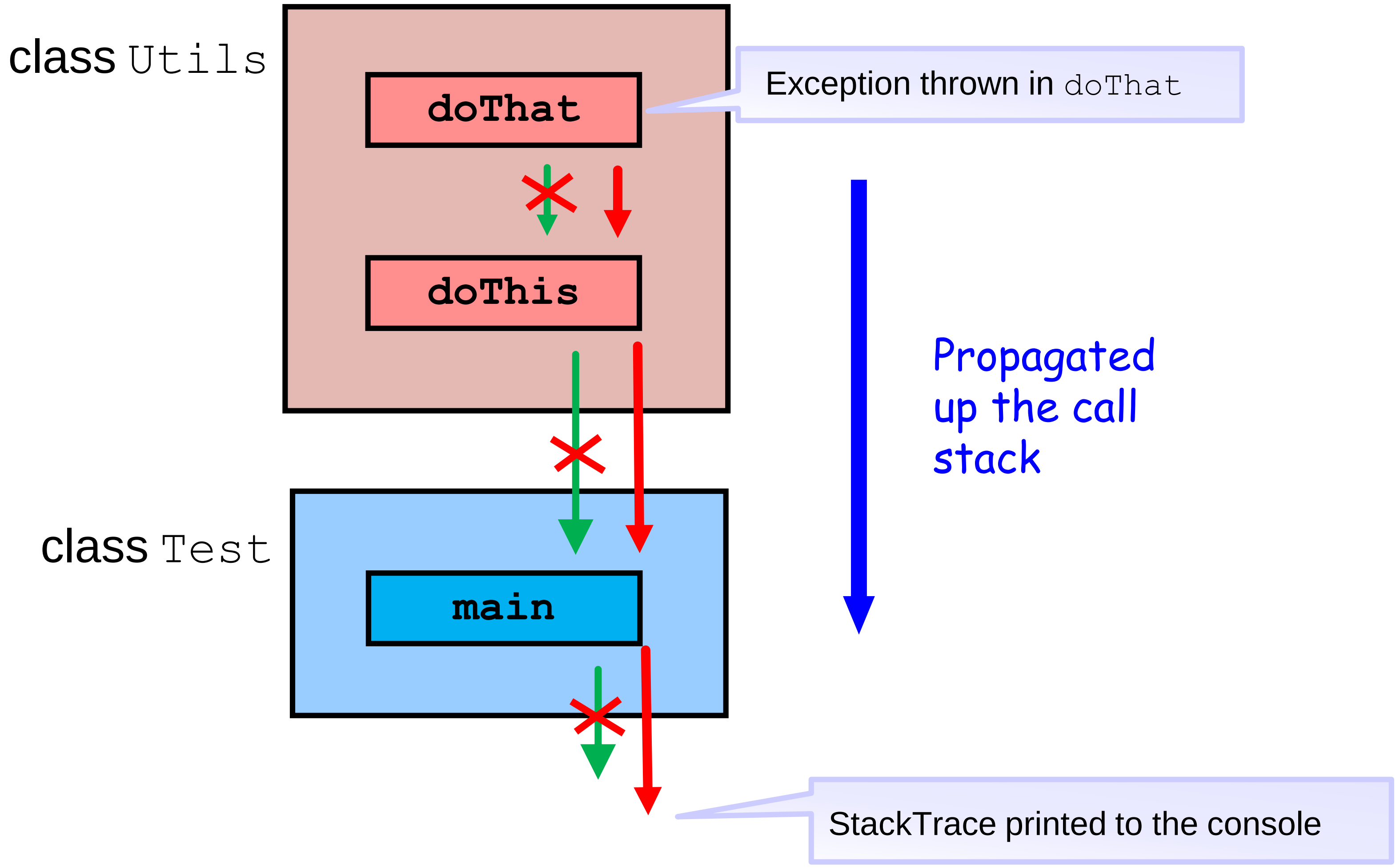
When an Exception Is Thrown


class Utils
doThat
doThis
class Test
main
Exception thrown in doThat
Execution returns to doThis and must be caught or thrown.

Throwing `Throwable` Objects

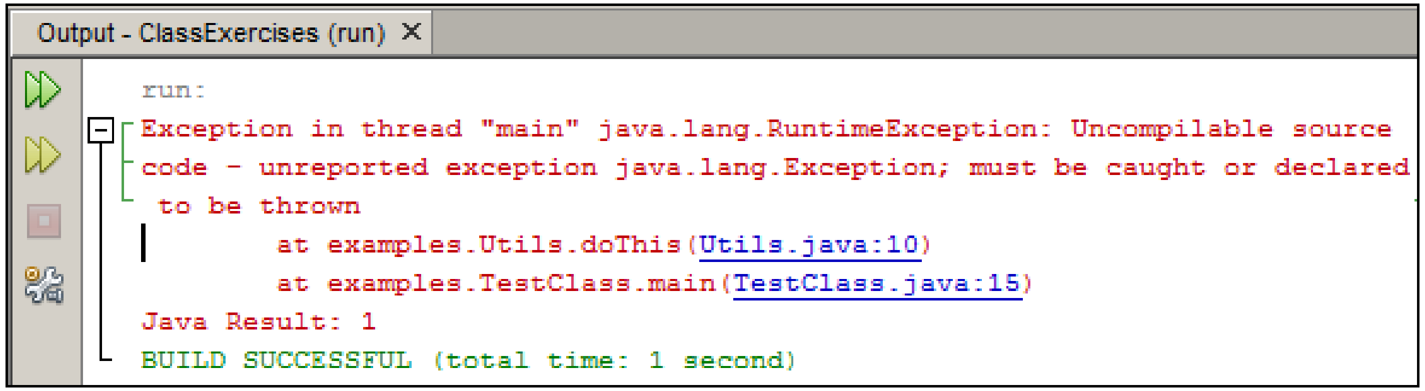

class Utils
doThat
doThis
class Test
main
Exception thrown in doThat
If doThis does NOT catch the exception, then …
… main must catch it OR throw it.

Uncaught Exception


class Utils
doThat
doThis
class Test
main
Exception thrown in doThat
Propagated up the call stack
StackTrace printed to the console

Exception Printed to Console

When the exception is thrown up the call stack without being caught, it will eventually reach the JVM. The JVM will print the exception’s output to the console and exit.



Output - ClassExercises (run)

```
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source
code - unreported exception java.lang.Exception; must be caught or declared
 to be thrown
        at examples.Utils.doThis(Utils.java:10)
        at examples.TestClass.main(TestClass.java:15)
Java Result: 1
BUILD SUCCESSFUL (total time: 1 second)
```

Summary of Exception Types

A `Throwable` is a special type of Java object.

- It is the only object type that:
  - Is used as the argument in a catch clause
  - Can be "thrown" to the calling method
- It has two direct subclasses:
  - `Error`
    - Automatically propagated up the call stack to the calling method
  - `Exception`
    - Must be explicitly handled and requires either:
      - A `try/catch` block to handle the error
      - A `throws` in the method signature to propagate up the call stack
    - Has a subclass `RuntimeException`
      - Automatically propagated up the call stack to the calling method

Quiz

Which one of the following statements is true?

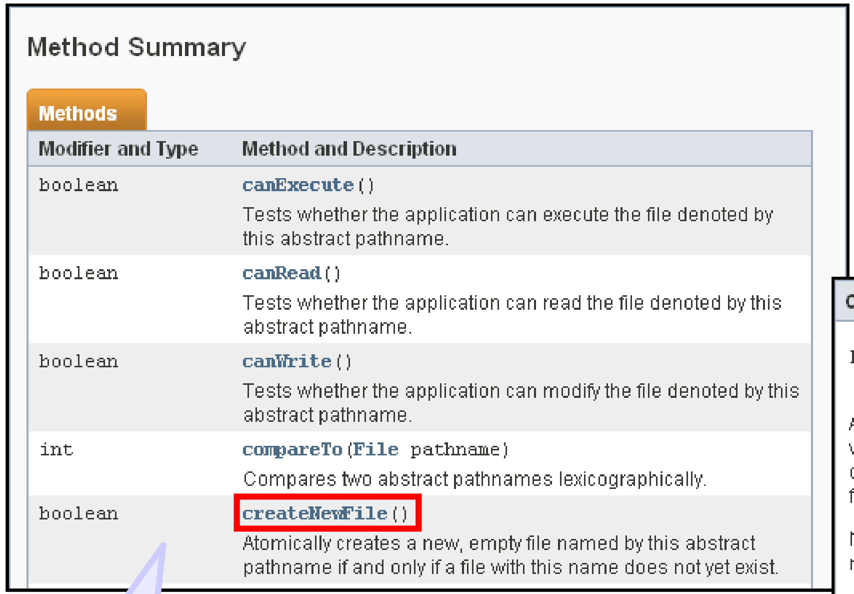
a. A `RuntimeException` must be caught.

b. A `RuntimeException` must be thrown.

c. A `RuntimeException` must be caught or thrown.

d. A `RuntimeException` is thrown automatically.

Exceptions in the Java API Documentation

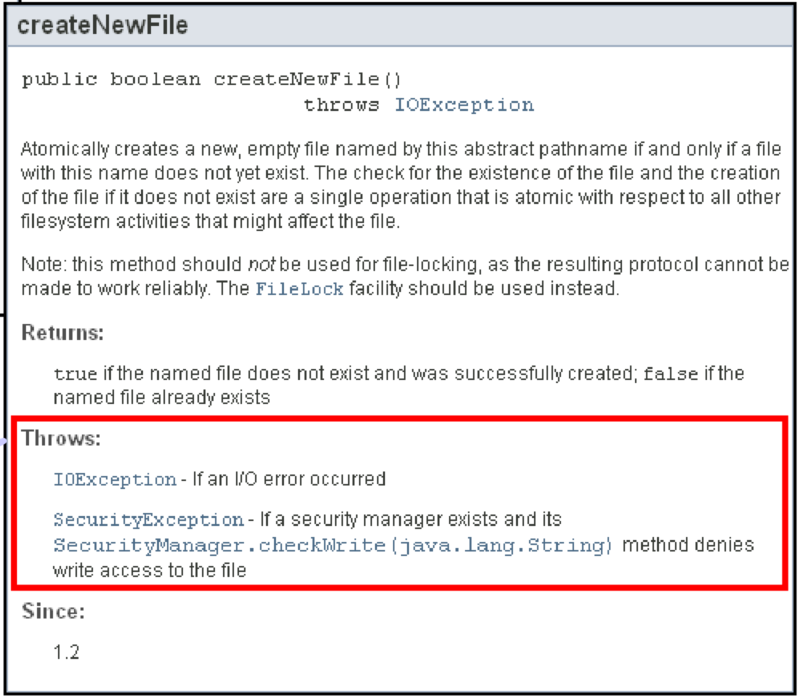


Method Summary

Methods

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

These are methods of the `File` Class.



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 I/O error occurred

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

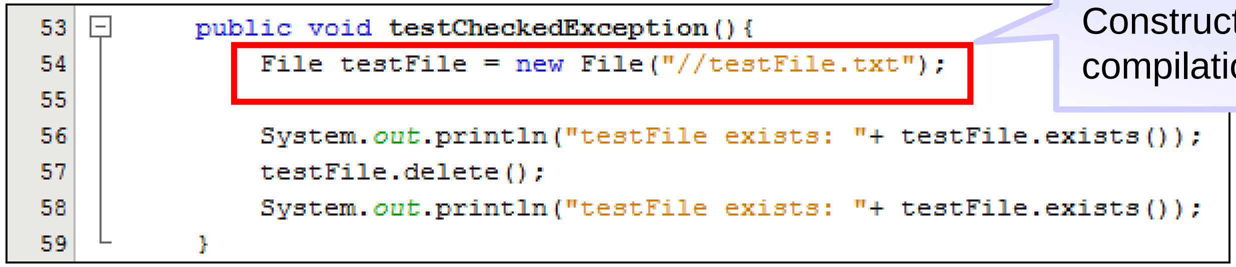
Since:

1.2

Click to get the detail of `createNewFile`.

Note the exceptions that can be thrown.

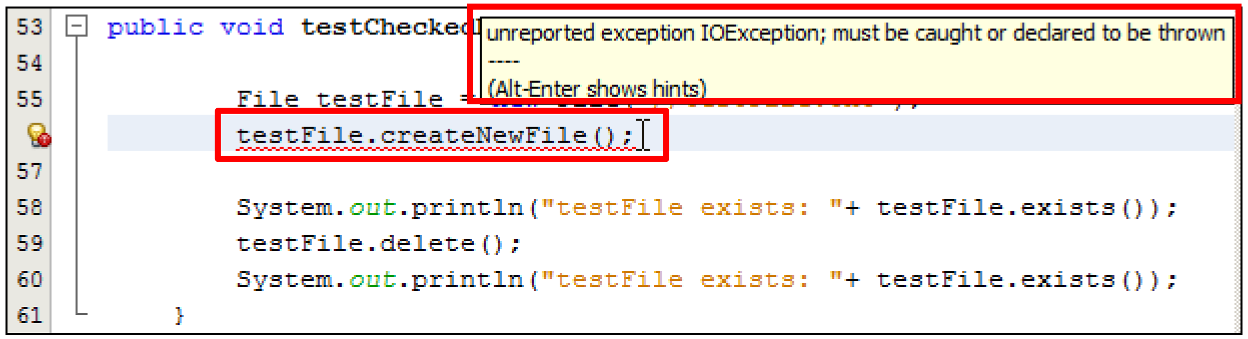
Calling a Method That Throws an Exception



```
public void testCheckedException(){
    File testFile = new File("//testFile.txt");

    System.out.println("testFile exists: "+ testFile.exists());
    testFile.delete();
    System.out.println("testFile exists: "+ testFile.exists());
}
```

Constructor causes no compilation problems.

`createNewFile` can throw a checked exception, so the method must throw or catch.



```
public void testChecked

        File testFile = 
        testFile.createNewFile();

        System.out.println("testFile exists: "+ testFile.exists());
        testFile.delete();
        System.out.println("testFile exists: "+ testFile.exists());
    }
```

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

Working with a Checked Exception

Catching `IOException`:

```
public static void main(String[] args) {
    TestClass testClass = new TestClass();

    try {
         testClass.testCheckedException();
     } catch (IOException e) {
         System.out.println(e);
     }
}

public void testCheckedException() throws IOException {
     File testFile = new File("//testFile.txt");
     testFile.createNewFile();
     System.out.println("testFile exists:"
        + testFile.exists());
}
```

Best Practices

- Catch the actual exception thrown, not the superclass type.
- Examine the exception to find out the exact problem so you can recover cleanly.
- You do not need to catch every exception.
  - A programming mistake should not be handled. It must be fixed.
  - Ask yourself, “Does this exception represent behavior I want the program to recover from?”

Bad Practices

```
public static void main(String[] args){
    try {
        createFile("c:/testFile.txt");
    } catch (Exception e) {
        System.out.println("Error creating file.");
    }
}
public static void createFile(String name)
        throws IOException{
    File f = new File(name);
    f.createNewFile();

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

Catching superclass?

No processing of exception class?

Somewhat Better Practice

```java
public static void main(String[] args){
    try {
        createFile("c:/testFile.txt");
    } catch (Exception e) {
        System.out.println(e);
    //<other actions>
    }
}
public static void createFile(String fname)
        throws IOException{
    File f = new File(name);
    System.out.println(name+" exists? "+f.exists());
    f.createNewFile();
    System.out.println(name+" exists? "+f.exists());
    int[] intArray = new int[5];
    intArray[5] = 27;
}
```

What is the object type?

`toString()` is called on this object.

Topics

- Handling errors: an overview
- Propagation of exceptions
- Catching and throwing exceptions
- Multiple exceptions and errors

Multiple Exceptions

Directory must be writeable:
IOException

```
public static void createFile()  throws IOException {
  File testF = new File("c:/notWriteableDir");

  File tempF = testF.createTempFile("te", null, testF);

  System.out.println
     ("Temp filename: "+tempF.getPath());
  int myInt[] = new int[5];
  myInt[5] = 25;
}
```

Arg must be greater than 3 characters:
IllegalArgumentException

Array index must be valid:
ArrayIndexOutOfBoundsException

Catching `IOException`

```java
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 = testF.createTempFile("te", null, testF);
    System.out.println("Temp filename: "+tempF.getPath());
    int myInt[] = new int[5];
    myInt[5] = 25;
}
```

Catching `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 = testF.createTempFile("te", null, testF);
    System.out.println("Temp filename: "+tempF.getPath());
    int myInt[] = new int[5];
    myInt[5] = 25;
}
```

Catching Remaining Exceptions

```java
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 = testF.createTempFile("te", null, testF);
   System.out.println("Temp filename: "+tempF.getPath());
   int myInt[] = new int[5];
   myInt[5] = 25;
}
```