

Exception Handling





Outlines

- › Common Errors by Java Programmers
- › Exception
- › Categories of Exceptions
- › Exception Handling: try-catch & throws
- › Try/Catch
 - Comparing to if-else
 - Finally
- › Throw
- › Create a new exception
- › JUnit with exception



Common Errors by Java Programmers

10

- Accessing non-static member variables from static methods (such as main)

9

- Mistyping the name of a method when overriding

8

- Comparison assignment (= rather than ==)

7

- Comparing two objects (== instead of .equals)

6

- Confusion over passing by value, and passing by reference



Common Errors by Java Programmers (cont.)

5

- Writing blank exception handlers

4

- Forgetting that Java is zero-indexed

3

- Preventing concurrent access to shared variables by threads

2

- Capitalization errors

1

- **Null pointers!**
 - Commonly caused by uninitialized objects



Common Errors by Java Programmers (cont.)

- › Syntax error
 - cannot compile

- › Logic error
 - wrong formula, wrong step, integer division, ...
 - unit test
 - fixed by programmer

- › Status of environment
 - network down
 - cannot open file
 - out of control by programmer

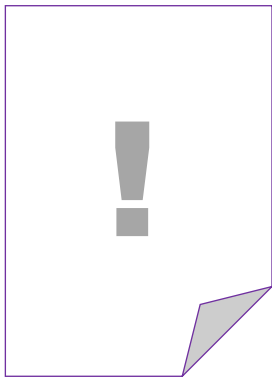
Though it is impossible to completely eliminate errors from the coding process, with care and practice you can avoid repeating the same ones.





Exception

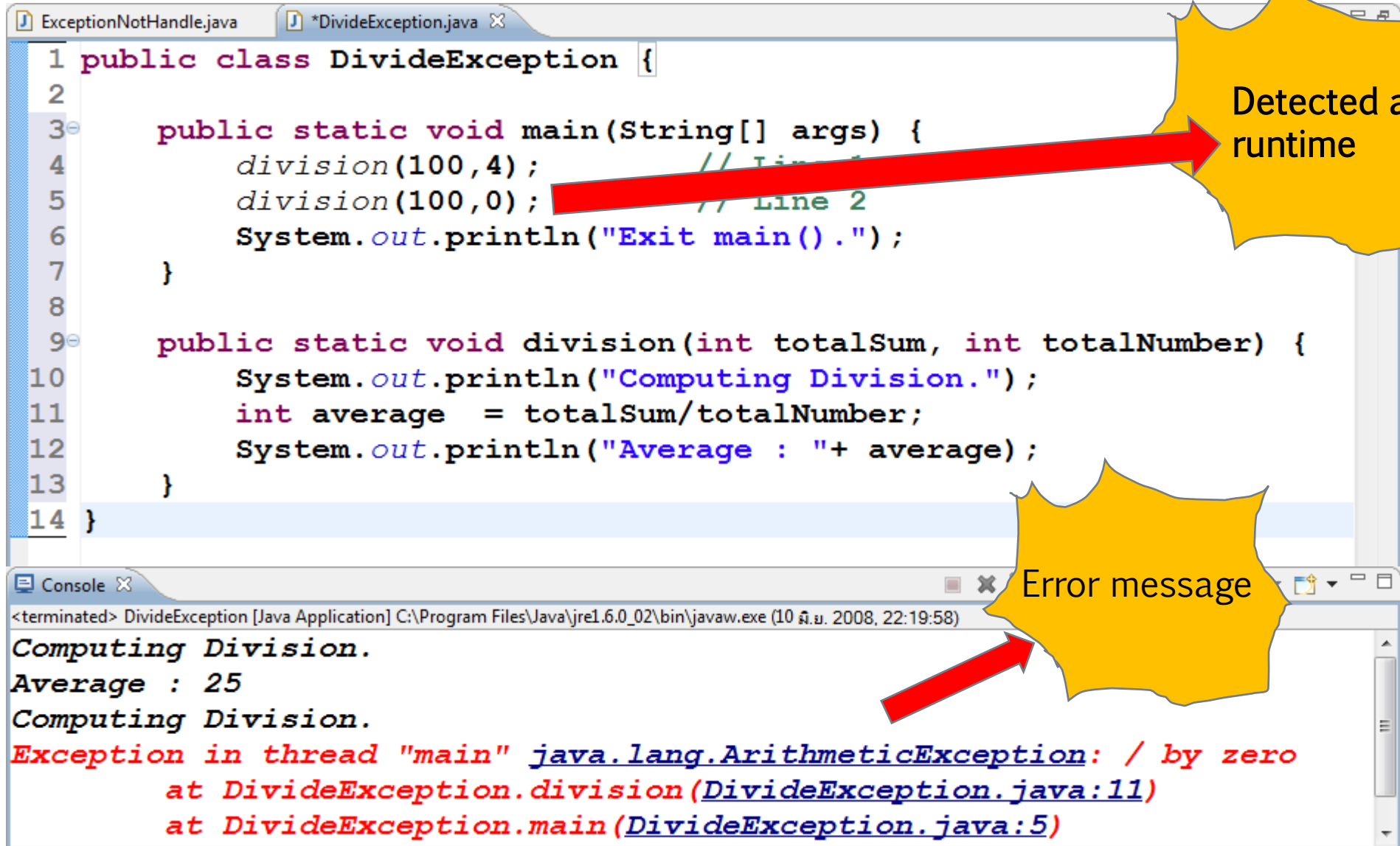
Exception



- › An **exception** is a problem that arises during the execution of a program.
- › There are many types of exceptions.
- › Therefore, there are many classes of Exception objects.
- › For example,
 - ArithmeticException
 - ArrayIndexOutOfBoundsException
 - FileNotFoundException



Exception (cont.): ArithmeticException



```
1 public class DivideException {
2
3     public static void main(String[] args) {
4         division(100,4); // Line 1
5         division(100,0); // Line 2
6         System.out.println("Exit main().");
7     }
8
9     public static void division(int totalSum, int totalNumber) {
10        System.out.println("Computing Division.");
11        int average = totalSum/totalNumber;
12        System.out.println("Average : "+ average);
13    }
14 }
```

Console

<terminated> DivideException [Java Application] C:\Program Files\Java\jre1.6.0_02\bin\javaw.exe (10 มิ.ย. 2008, 22:19:58)

Computing Division.
Average : 25
Computing Division.
Exception in thread "main" java.lang.ArithmeticException: / by zero
at DivideException.division(DivideException.java:11)
at DivideException.main(DivideException.java:5)



Exception (cont.): ArrayIndexOutOfBoundsException

```
1 /*  
2  * This class demonstrates what happen when  
3  * runtime exception occurs and not handled.  
4  */  
5  
6 public class RuntimeExceptionNotHandle {  
7  
8     public static void main(String[] args) {  
9  
10         int[] data = new int[4];  
11         data[4] = 7;  
12  
13         String courseNumber = "2110210";  
14         System.out.println(courseNumber.charAt(10));  
15     }  
16 }
```

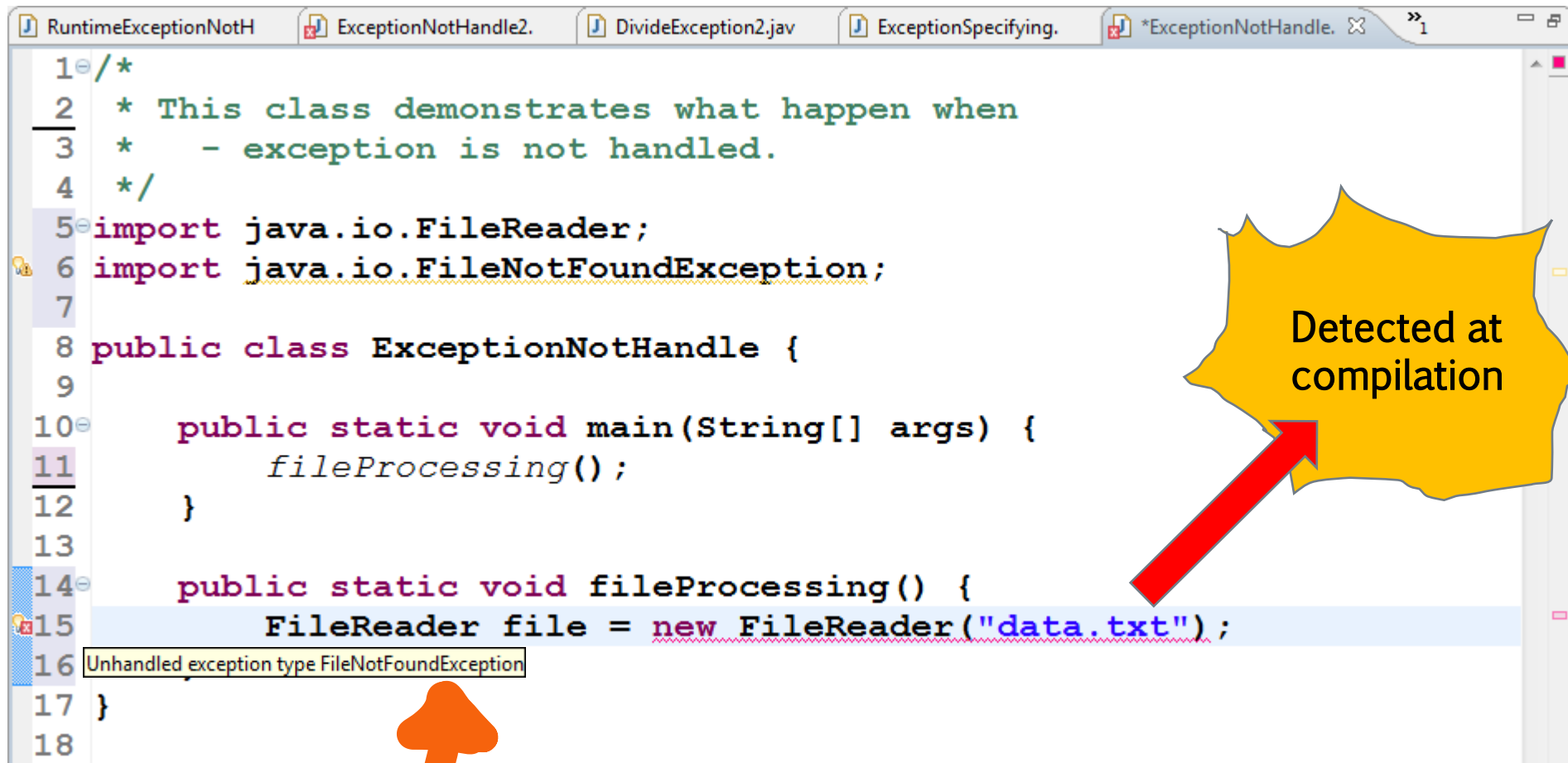
Detected at
runtime

This one will cause exception if
the first 2 lines are commented
out.

```
<terminated> RuntimeExceptionNotHandle [Java Application] C:\Program Files\Java\jre1.6.0_02\bin\javaw.exe (11 มิ.ย. 2008, 9:38:17)  
tion in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4  
at RuntimeExceptionNotHandle.main(RuntimeExceptionNotHandle.java:11)
```




Exception (cont.): FileNotFoundException



```
1 /*
2  * This class demonstrates what happen when
3  * - exception is not handled.
4  */
5 import java.io.FileReader;
6 import java.io.FileNotFoundException;
7
8 public class ExceptionNotHandle {
9
10     public static void main(String[] args) {
11         fileProcessing();
12     }
13
14     public static void fileProcessing() {
15         FileReader file = new FileReader("data.txt");
16     }
17 }
18
```

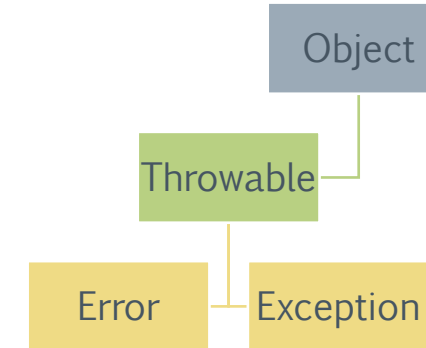
Detected at compilation

Unhandled exception type FileNotFoundException

This error is shown!



Categories of Exceptions



ERROR

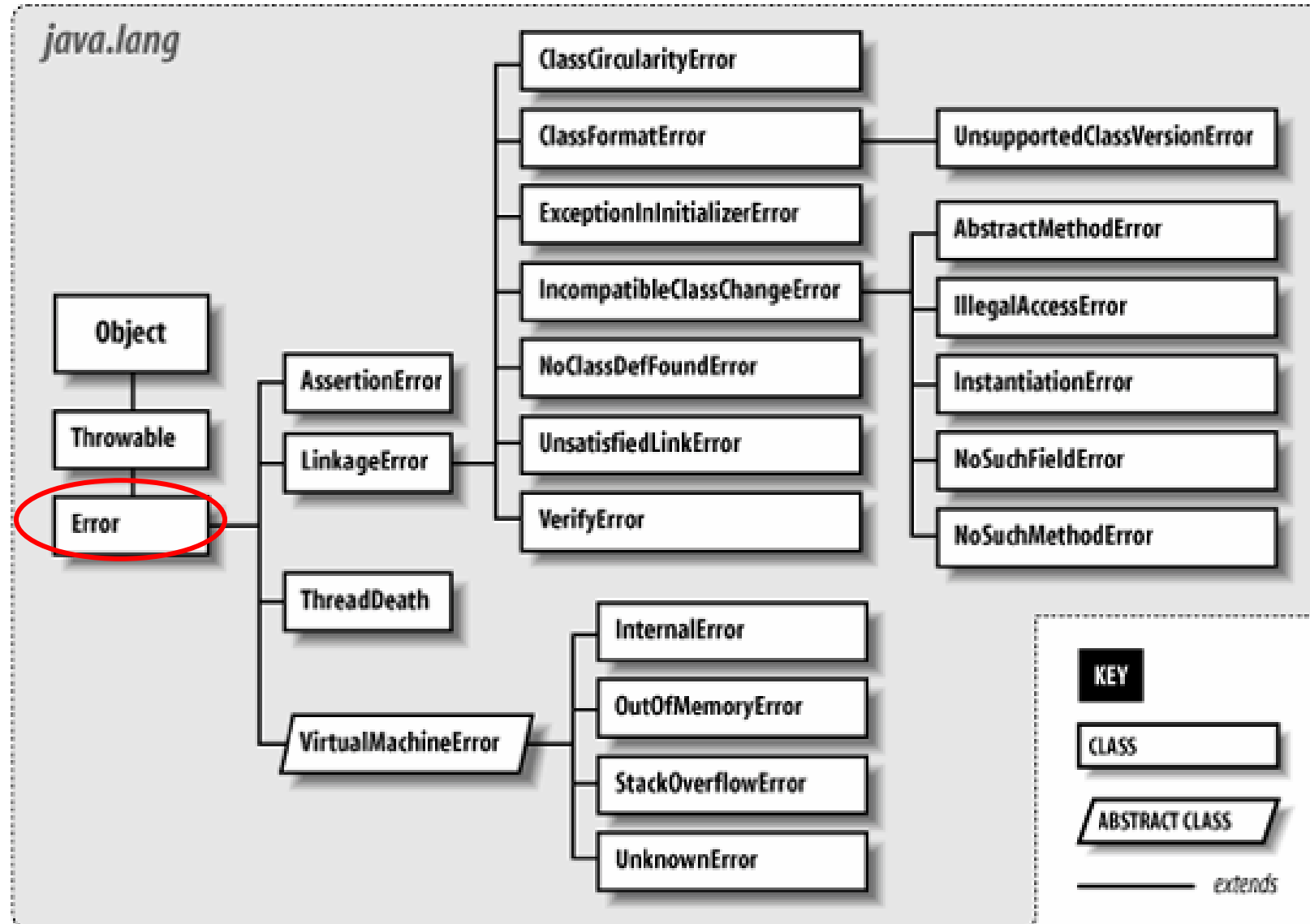
- › It is **a serious problem** that arise beyond the control of the user or the programmer.
- › It is typically ignored in your code because you can rarely do anything about an error

EXCEPTION

- › It is **an error that is less serious** than the Error class and can be control by the program.
 - Allow to “try/catch” or “throw”
- › There are two types:
 - Unchecked Exception
 - Checked Exception



Categories of Exceptions (cont.): Error

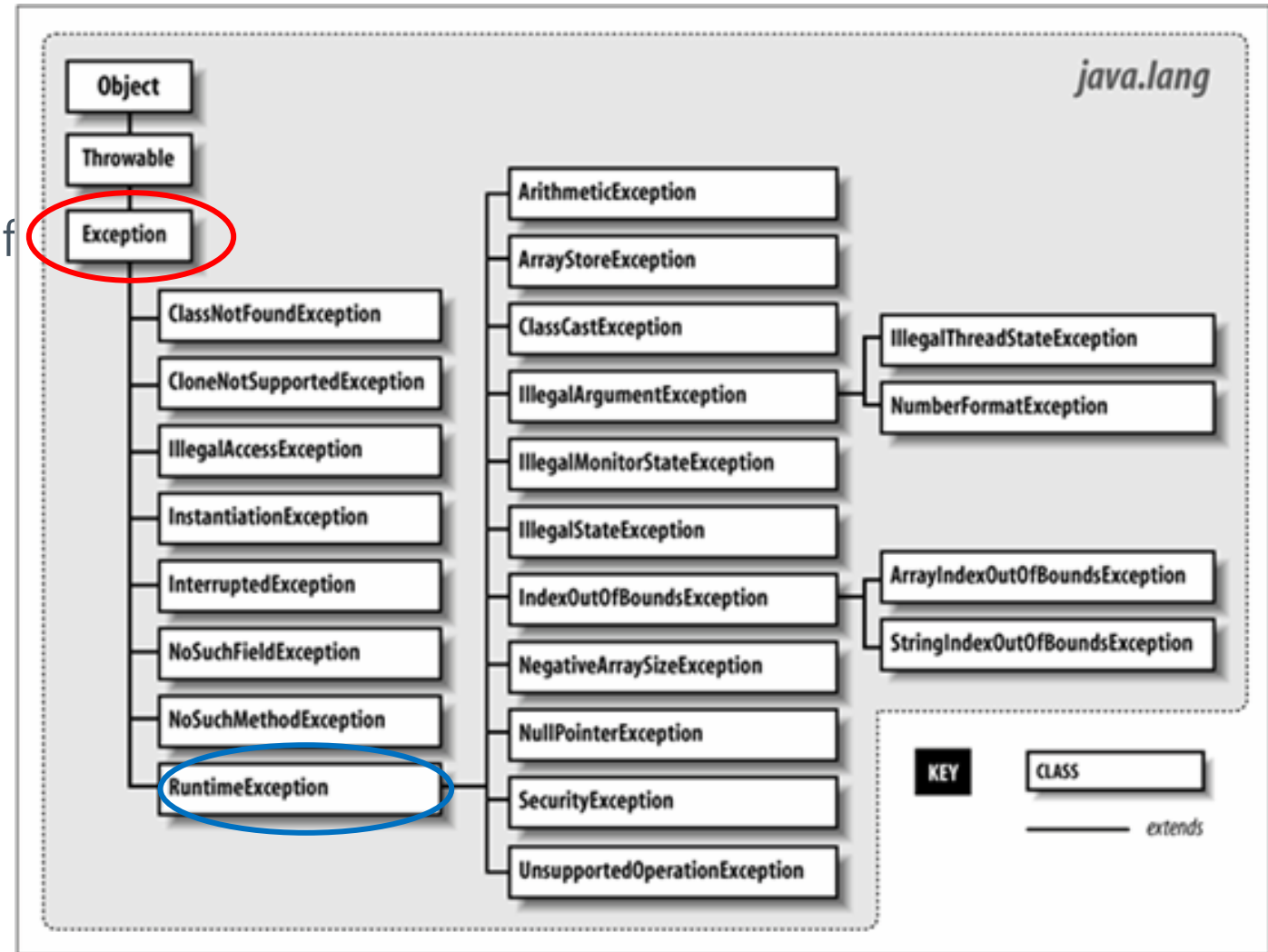




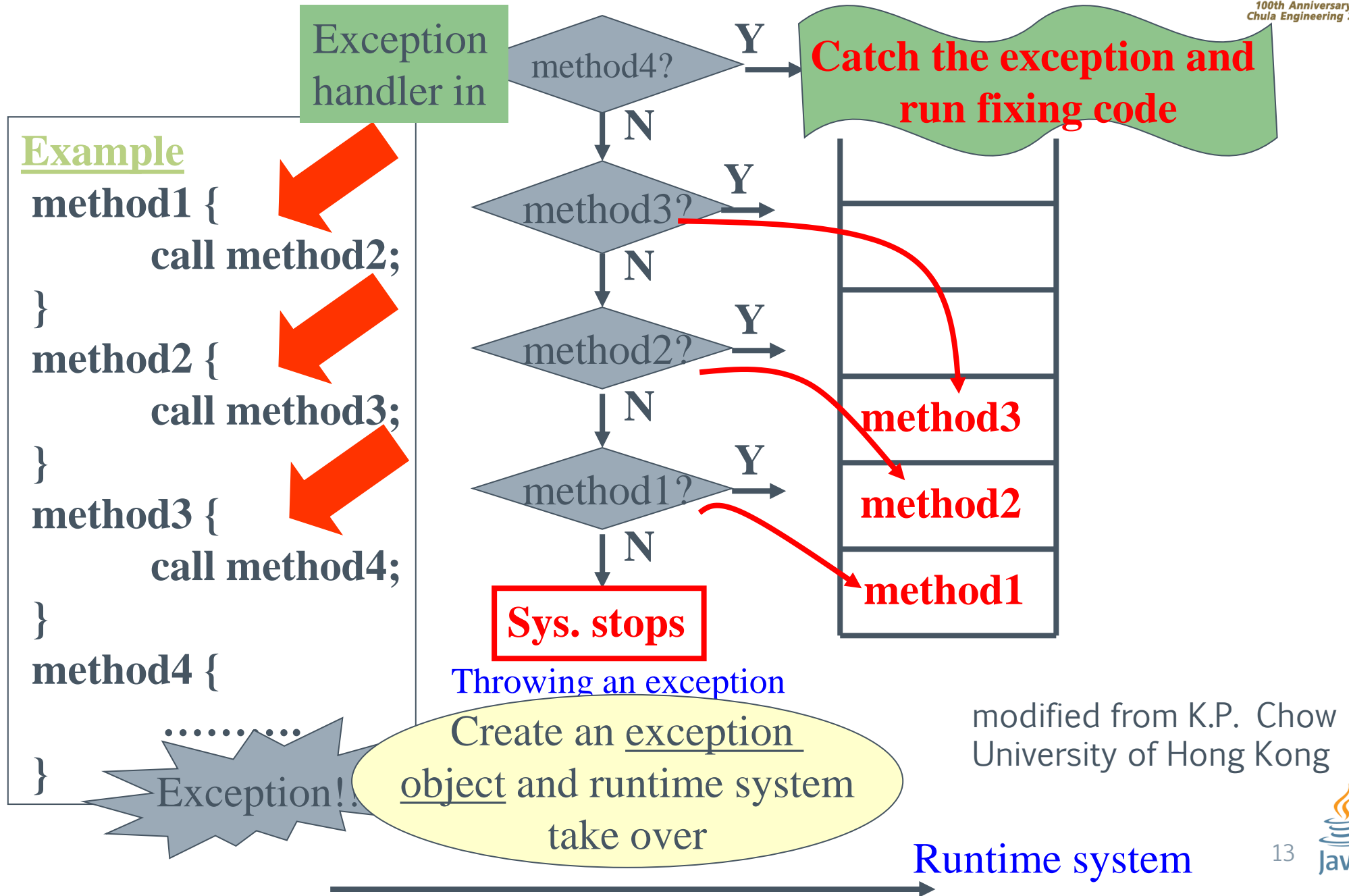
Categories of Exceptions (cont.): Exception

- › Unchecked exceptions:
 - They are ignored at the compilation time.
 - They are any subclasses of **RuntimeException**.

- › Checked exceptions:
 - These exceptions cannot simply be ignored at the time of compilation.
 - They **must be** handled (try-catch or throw).
 - They are Exception's subclasses, **except RuntimeException**.



What happens when an exception is generated?





Exception Handling

› Try-catch

```
try {  
    block of statements  
} catch (ExceptionType name) {  
    exception handler 1  
} catch (ExceptionType name) {  
    exception handler 2  
}
```

› Throws

in Integer class:

```
public static int parseInt(String s)  
    throws NumberFormatException;
```

› No handling at all

- unchecked exceptions only
- need to be carefully checked by programmers

› try/catch/finally

- handle normally

› Specifying the exception

- throws the exception to the caller
- Used when we **don't want to catch** the exception in this method



Try-Catch: Usage

```
System.out.println(aE.getMessage());
```

```
/ by zero
```

```
End
```

```
System.out.println(aE.toString());
```

```
java.lang.ArithmeticException: / by zero
```

```
End
```

```
aE.printStackTrace()
```

```
java.lang.ArithmeticException: / by zero
```

```
End
```

```
at Exception1.main(Exception1.java:7)
```

Exception1.java

```
public class Exception1 {  
    public static void main(String[] args) {  
        int s[] = new int[2];  
        try {  
            for (int i = 0; i < 3; ++i) {  
                s[i] = 1/i;  
                System.out.println(s[i]);  
            }  
        } catch (ArrayIndexOutOfBoundsException arrE) {  
            System.out.println(arrE.toString());  
        } catch (ArithmeticException aE) {  
            System.out.println(aE.toString());  
        } catch (Exception e) {  
            System.out.println(e.toString());  
        }  
        System.out.println("End");  
    }  
}
```

Try-Catch: Comparing to if-else



ReadFile1.java (pseudo code)

```

errorCodeType readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) errorCode = -1;    // read failed
            } else errorCode = -2;                // not enough memory
        } else errorCode = -3;                    // file size can't be determined
        close the file;
        if (theFileDidntClose && errorCode == 0) {
            errorCode = -4;                        // can't close file
        } else errorCode = errorCode and -4;    // can't close file + error
    } else errorCode = -5;                        // can't open file
    return errorCode;
}
    
```

Pseudo code to read file

```

readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
    
```

- **Spaghetti code**
 - difficulty to read
- What if a method needs to **return value**?
 - a method can return only a single value



Try-Catch: Comparing to if-else (cont.)

ReadFile2.java (pseudo code)

```
readFile {  
  try {  
    open the file;  
    determine its size;  
    allocate that much memory;  
    read the file into memory;  
    close the file;  
  } catch (fileOpenFailed) {  
    doSomething;  
  } catch (sizeDeterminationFailed) {  
    doSomething;  
  } catch (memoryAllocationFailed) {  
    doSomething;  
  } catch (readFailed) {  
    doSomething;  
  } catch (fileCloseFailed) {  
    doSomething;  
  }  
}
```

Pseudo code to read file

```
readFile {  
  open the file;  
  determine its size;  
  allocate that much memory;  
  read the file into memory;  
  close the file;  
}
```

Comparing ReadFile1.java & ReadFile2.java,
which one is better?





Try-Catch: Finally

TestFinally.java (main)

```
public class TestFinally {  
    public static void main(String[] args) {  
        functionWithFinally();  
    }  
}
```

Result (return)

```
catch  
finally
```

Result (System.exit(-1))

```
catch
```



Why do we need “finally”?
Can't we just move “finally code” to
be after the try-catch statement.

TestFinally.java (functionWithFinally)

```
public static void functionWithFinally() {  
    int result = 0;  
    for (int i = 0; i < 4; ++i) {  
        try {  
            result = 10 / i;  
            System.out.println("i=" + i + " and result=" + result);  
            if (i == 2) break;  
        } catch (ArithmeticException ae) {  
            System.out.println("catch");  
            return;  
        } finally {  
            System.out.println("finally");  
        }  
        System.out.println("End Step\n");  
    }  
    System.out.println("End Main Loop");  
}
```

```
public void writeList() {  
    try {  
        PrintWriter out = new PrintWriter(new FileWriter("out.txt"));  
        for (int i=0; i<SIZE; i++) {  
            out.println(v.elementAt(i));  
        }  
        out.close( );  
    } catch (ArrayIndexOutOfBoundsException e) {  
        System.err.println("Caught ArrayIndexOutOfBoundsException");  
    } catch (IOException e) {  
        System.err.println("Caught IOException");  
    }  
}
```

May not get executed!



use a finally block
(always will execute, even if
we jump out of try block)



Throws

- › When an exception occurs in the method, it will be *thrown* to the caller.
- › add **throws** clause to the method declaration if we do not want to catch exception within the current method.
- › Throw1.java
 - Caller: main()
 - Callee: greet()
 - › Checked Exception:
ClassNotFoundException, **InterruptedException**
- › Caller must handle **ALL** checked exception in the callee!

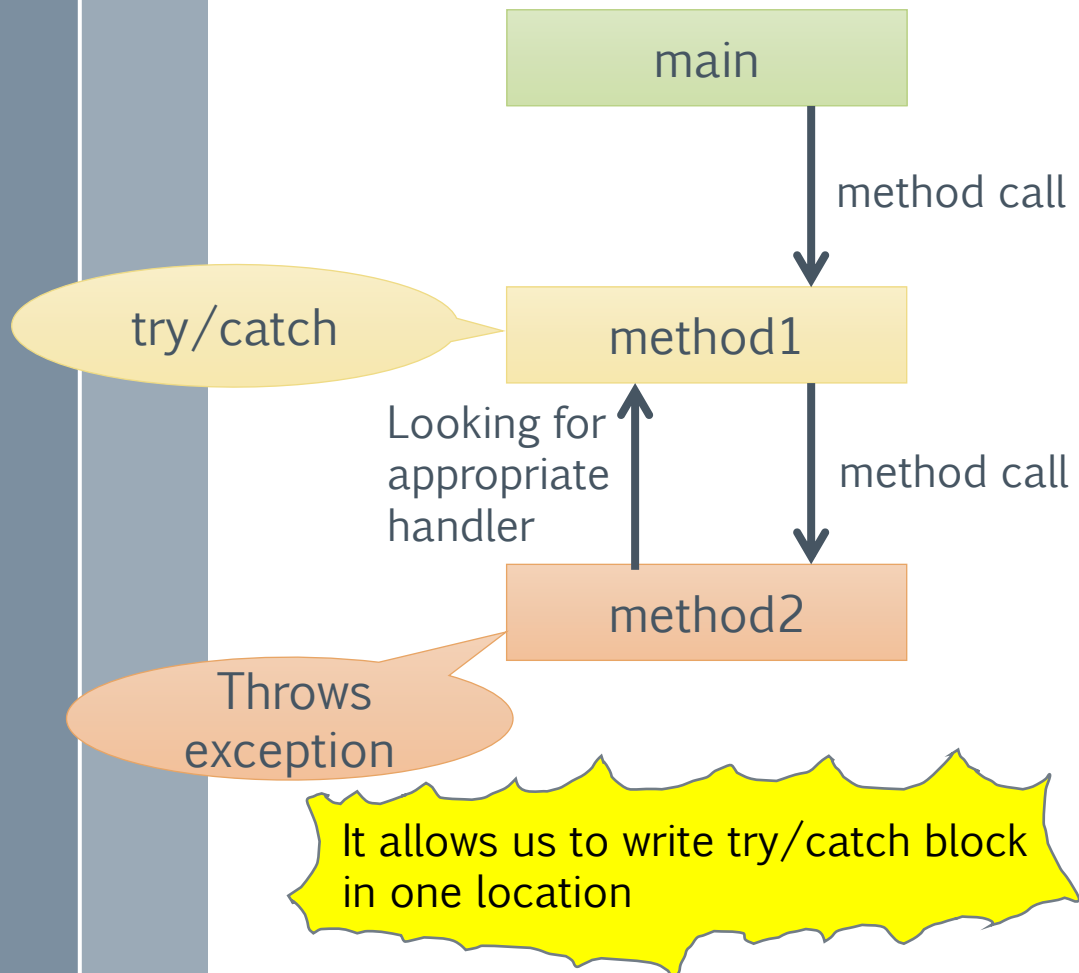
```
public class Throw1 {  
  
    static void greet(String name) throws ClassNotFoundException,  
    InterruptedException {  
  
        if (name.equals("John"))  
  
            throw new InterruptedException();  
  
        System.out.println("Hello! " + name);  
  
    }  
  
    public static void main(String[] args) throws  
    ClassNotFoundException {  
  
        try {  
  
            greet("John");  
  
        } catch (InterruptedException e) {  
  
            System.out.println("Bye.");  
  
        }  
  
    }  
  
}
```

Result

Bye



Throws: Chain Caller



ChainCaller.java

```
public class ChainCaller {  
    public static void main(String[] args) {  
        ChainCaller t = new ChainCaller();  
        t.method1(6, 3);  
        t.method1(6, 0);  
    }  
    public void method1(double a, double b) {  
        try {  
            System.out.println(method2(a, b));  
        } catch (ArithmeticException ae) {  
            System.out.println("Divided by zero not allowed");  
        }  
    }  
    public String method2(double a, double b)  
        throws ArithmeticException {  
        if (b == 0) throw new ArithmeticException();  
        else return a + "/" + b + "=" + a / b;  
    }  
}
```

Result

6.0/3.0=2.0

Divided by zero not allowed



What happens if we don't want to catch at all

```
import java.io.*;  
public void m1( ) {  
    m2( );  
}  
public void m2( ) {  
    m3( );  
}  
public void m3( ) throws IOException {  
    int b = System.in.read( );  
}
```

```
public void m1( ) throws IOException {  
    m2( );  
}
```

```
public void m2( ) throws IOException {  
    m3( );  
}
```

**Compile ok, but do not
handle the exception....**

Error!!

**m2 has to either catch or
throw IOException**

Error!!

**m1 has to either catch or
throw IOException**

modified from K.P. Chow
University of Hong Kong



Create a new exception



“Extends” can be applied.

TestMyException.java: MyException

```
class MyException extends Exception {  
    public MyException(String s) {  
        System.out.println("MyException = " + s);  
    }  
}
```

TestMyException.java

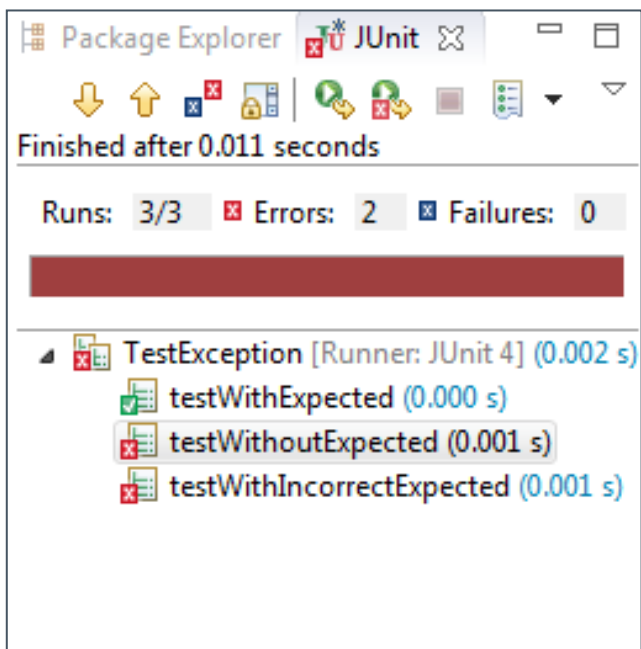
```
public class TestMyException {  
    static void welcome(String s) throws MyException {  
        if (s.equals("JAVA"))  
            System.out.println("Welome to JAVA World");  
        else  
            throw new MyException(s + " not allowed here");  
    }  
  
    public static void main(String[] args) {  
        try {  
            welcome("C#");  
        } catch (MyException e1) {  
            System.out.println("MyException.");  
        }  
    }  
}
```

Result

```
MyException = C# not allowed here  
MyException.
```



JUnit4 with exception



JUnit: TestException

```
import org.junit.Test;
```

```
public class TestException {
```

```
@Test(expected=ArithmeticException.class)
```

```
public void testWithExpected() {
```

```
    double a = 10/0;
```

```
}
```

```
@Test
```

```
public void testWithoutExpected() {
```

```
    double a = 10/0;
```

```
}
```

```
@Test(expected=ArrayIndexOutOfBoundsException.class)
```

```
public void testWithIncorrectExpected() {
```

```
    double a = 10/0;
```

```
}
```

```
}
```




JUnit5 with exception

Runs: 2/2 ✖ Errors: 1 ✖ Failures: 0

▼ ✖ TestException [Runner: JUnit 5] (0.018 s)
✖ testWithoutExpected() (0.001 s)
✔ testWithExpectedJava8() (0.017 s)

Finished after 0.149 seconds

Runs: 2/2 ✖ Errors: 1 ✖ Failures: 1

▼ ✖ TestException [Runner: JUnit 5] (0.016 s)
✖ testWithoutExpected() (0.004 s)
✖ testWithExpectedJava8() (0.010 s)

JUnit 5: TestException

@Test

```
public void testWithExpectedJava8() {
    Executable e = () -> System.out.println(10/0);
    assertThrows(ArithmeticException.class, e);
}
```

@Test

```
public void testWithExpectedJava8() {
    assertThrows(ArithmeticException.class,
        () -> {int x = 10/0;});
}
```

@Test

```
public void test03() {
    Executable e = () -> {
        int x = 1;
        int y = 9;
        int z = x/y;
        int xx = z/0;
        y = xx+1;
    };
    assertThrows(ArithmeticException.class, e);
}
```