100th Anniversary of
Chula Engineering 2013

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

from: C. Patanothai & http://www.javacoffeebreak.com/articles/toptenerrors.html

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

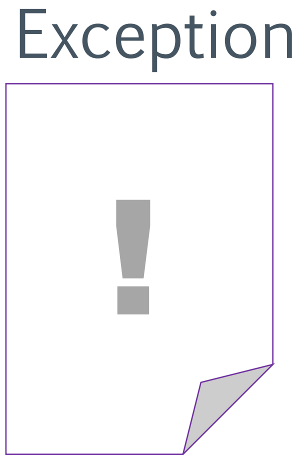
from: C. Patanothai & http://www.javacoffeebreak.com/articles/toptenerrors.html

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

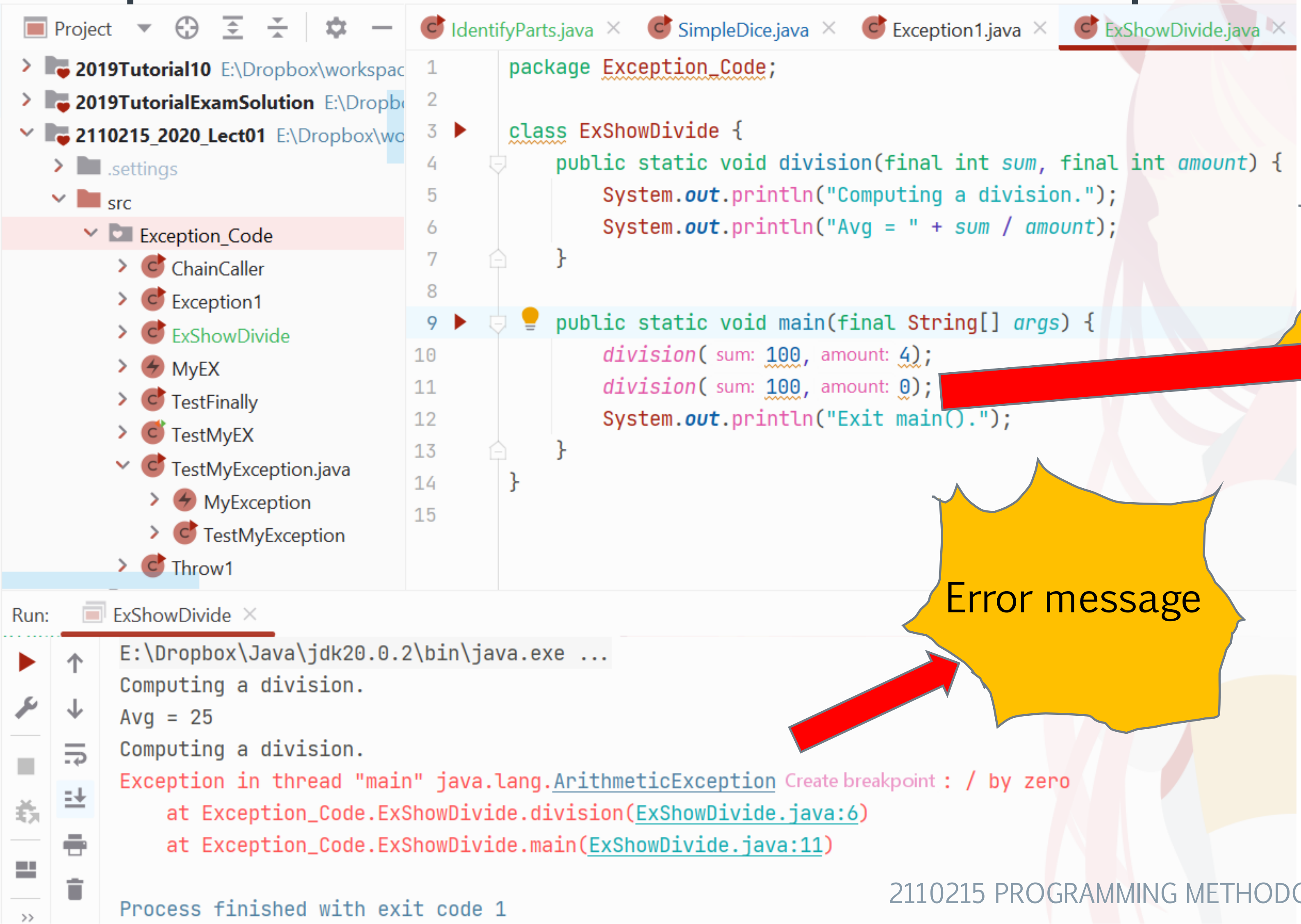
Though it is impossible to completely eliminate errors from the coding process, with care and practice you can avoid repeating the same ones.

Exception



- **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



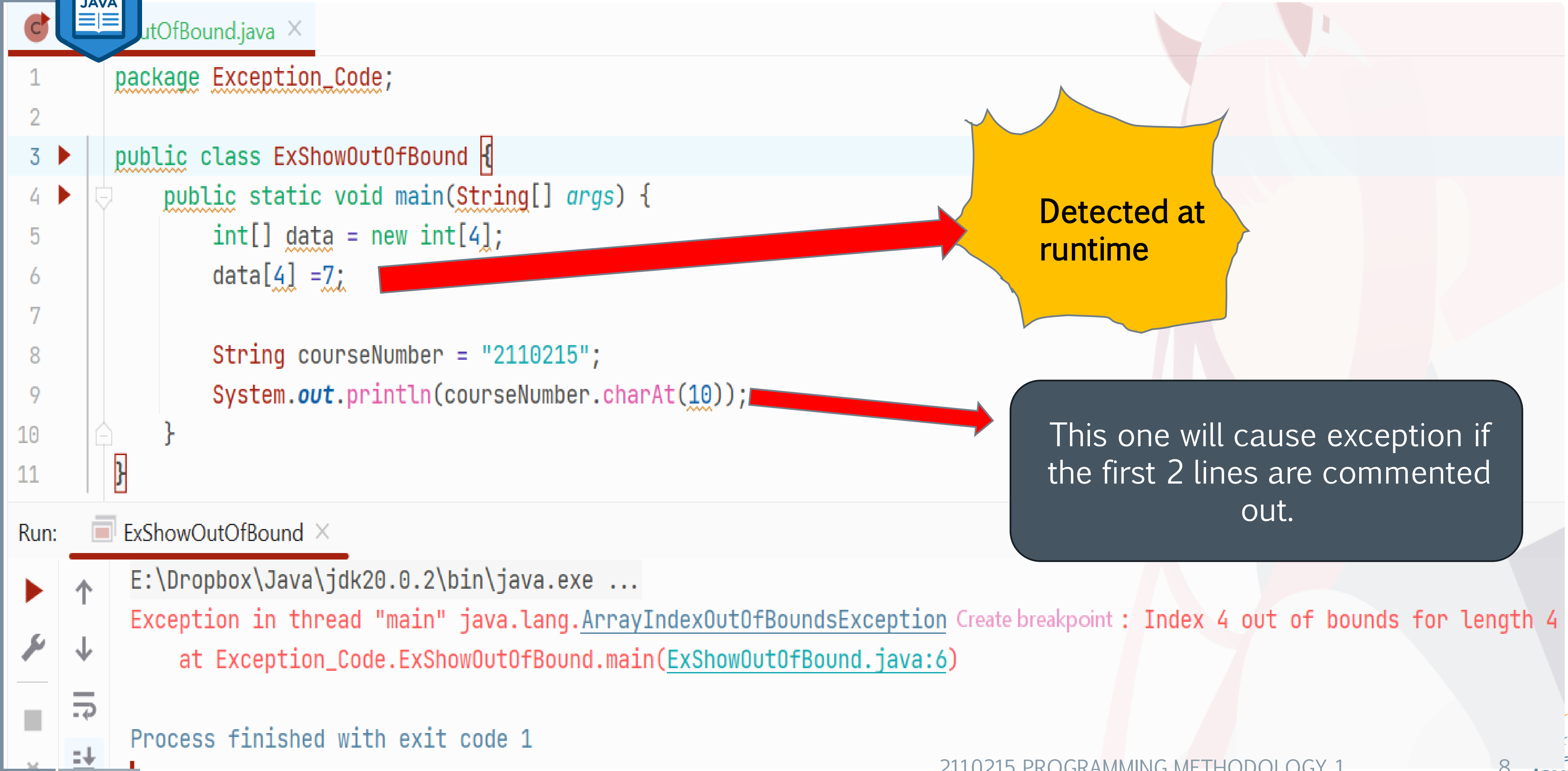
```java
package Exception_Code;

class ExShowDivide {
    public static void division(final int sum, final int amount) {
        System.out.println("Computing a division.");
        System.out.println("Avg = " + sum / amount);
    }

    public static void main(final String[] args) {
        division( sum: 100, amount: 4);
        division( sum: 100, amount: 0);
        System.out.println("Exit main().");
    }
}
```

```
E:\Dropbox\Java\jdk20.0.2\bin\java.exe ...
Computing a division.
Avg = 25
Computing a division.
Exception in thread "main" java.lang.ArithmeticException Create breakpoint : / by zero
    at Exception_Code.ExShowDivide.division(ExShowDivide.java:6)
    at Exception_Code.ExShowDivide.main(ExShowDivide.java:11)

Process finished with exit code 1
```

Detected at runtime

Error message

Exception (cont.): ArrayIndexOutOfBoundsException



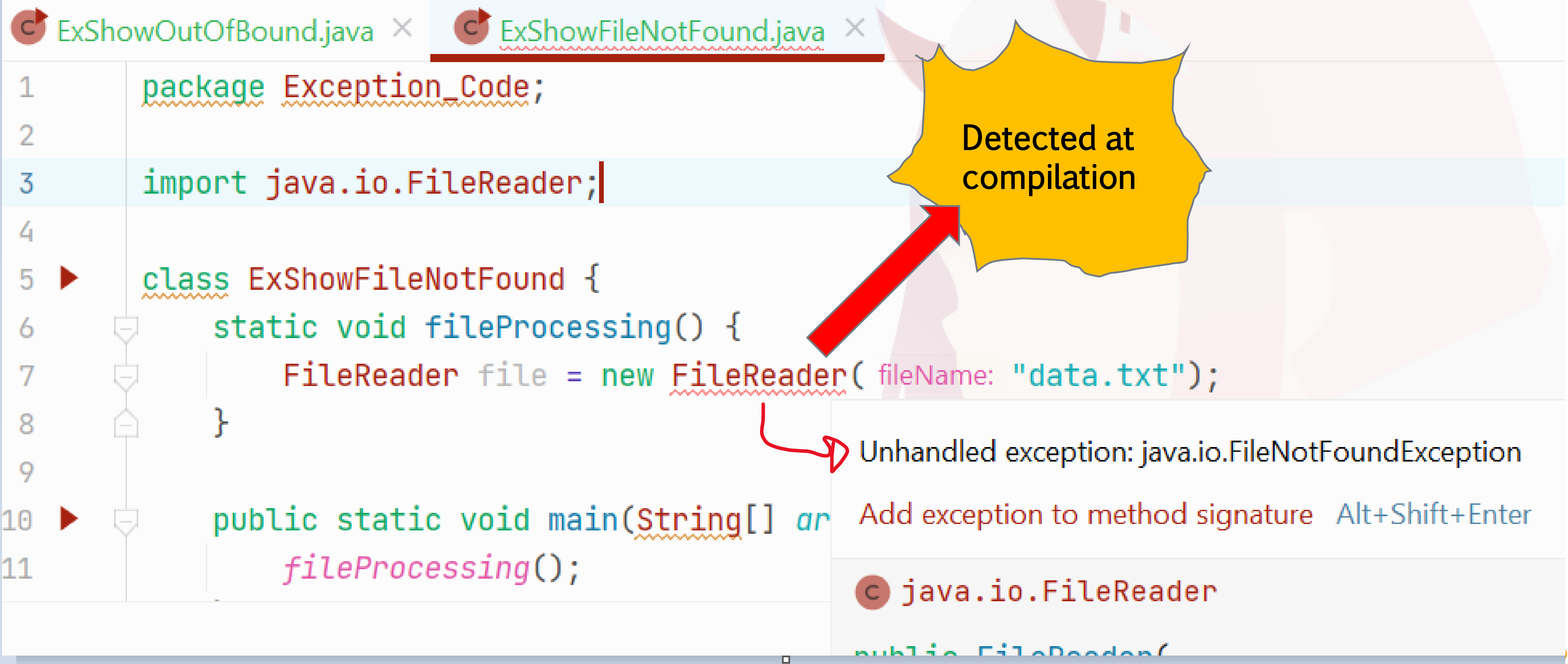
```java
package Exception_Code;

public class ExShowOutOfBound {
    public static void main(String[] args) {
        int[] data = new int[4];
        data[4] =7;

        String courseNumber = "2110215";
        System.out.println(courseNumber.charAt(10));
    }
}
```

```
E:\Dropbox\Java\jdk20.0.2\bin\java.exe ...
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 4 out of bounds for length 4
    at Exception_Code.ExShowOutOfBound.main(ExShowOutOfBound.java:6)

Process finished with exit code 1
```

Detected at runtime

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

Exception (cont.): FileNotFoundException



```
package Exception_Code;

import java.io.FileReader;

class ExShowFileNotFound {
    static void fileProcessing() {
        FileReader file = new FileReader( fileName: "data.txt");
    }

    public static void main(String[] ar
        fileProcessing();
```

Unhandled exception: java.io.FileNotFoundException

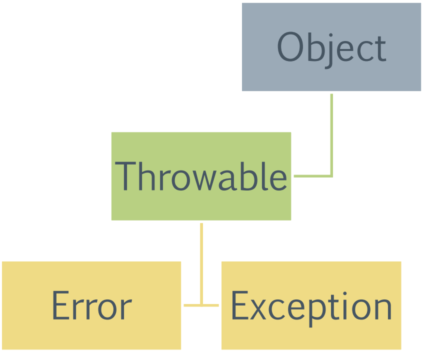
Add exception to method signature Alt+Shift+Enter

java.io.FileReader

public FileReader(

Detected at compilation

Categories of Exceptions



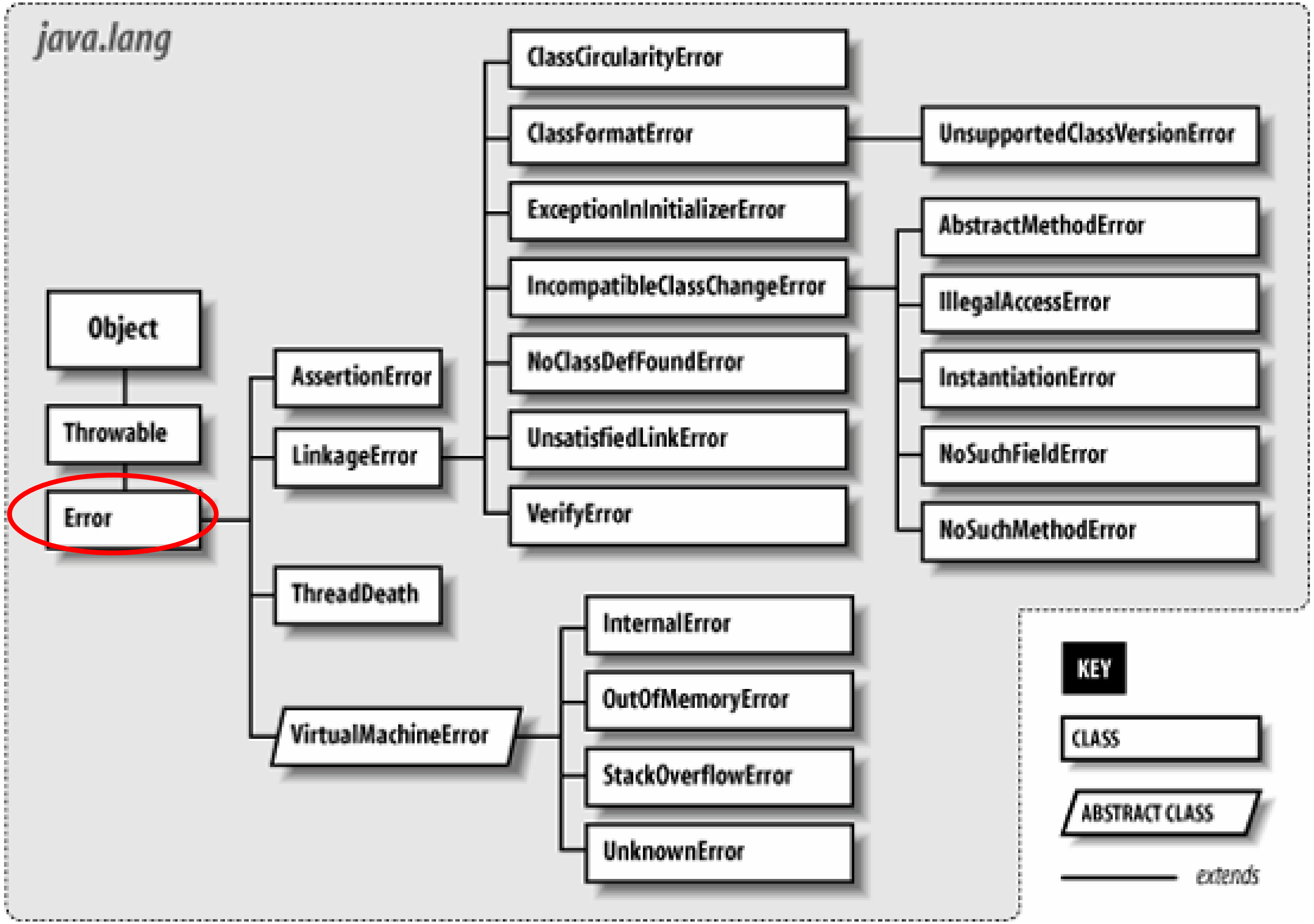
ERROR

- 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

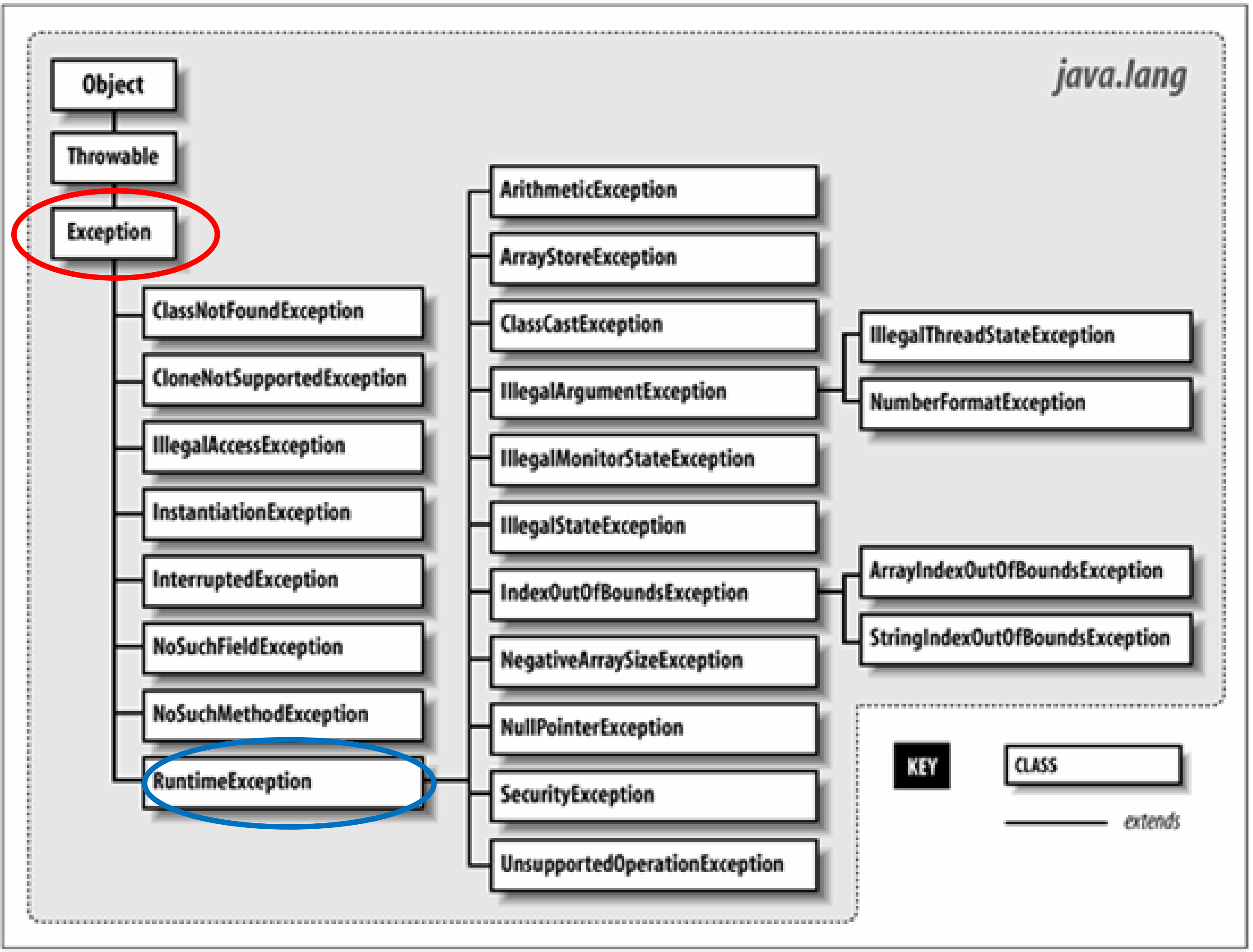


Categories of Exceptions (cont.): Exception

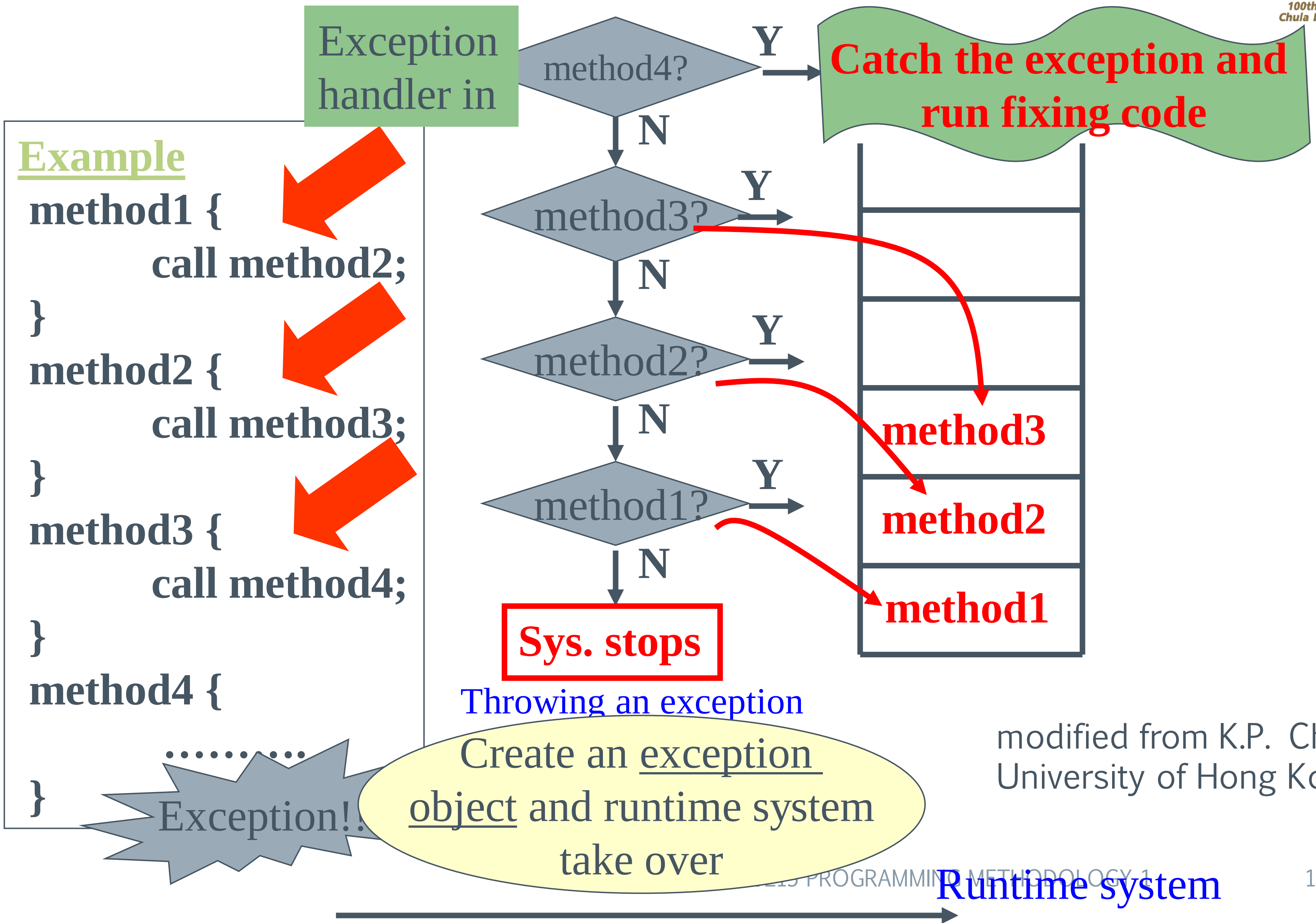
run time checking

- 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.

compile time checking



What happens when an exception is generated?



Example

```
method1 {
        call method2;
}
method2 {
        call method3;
}
method3 {
        call method4;
}
method4 {

        ..........
}
```

Throwing an exception

Create an exception object and runtime system take over

modified from K.P. Chow University of Hong Kong

Exception Handling

- Try-catch

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

- Throws

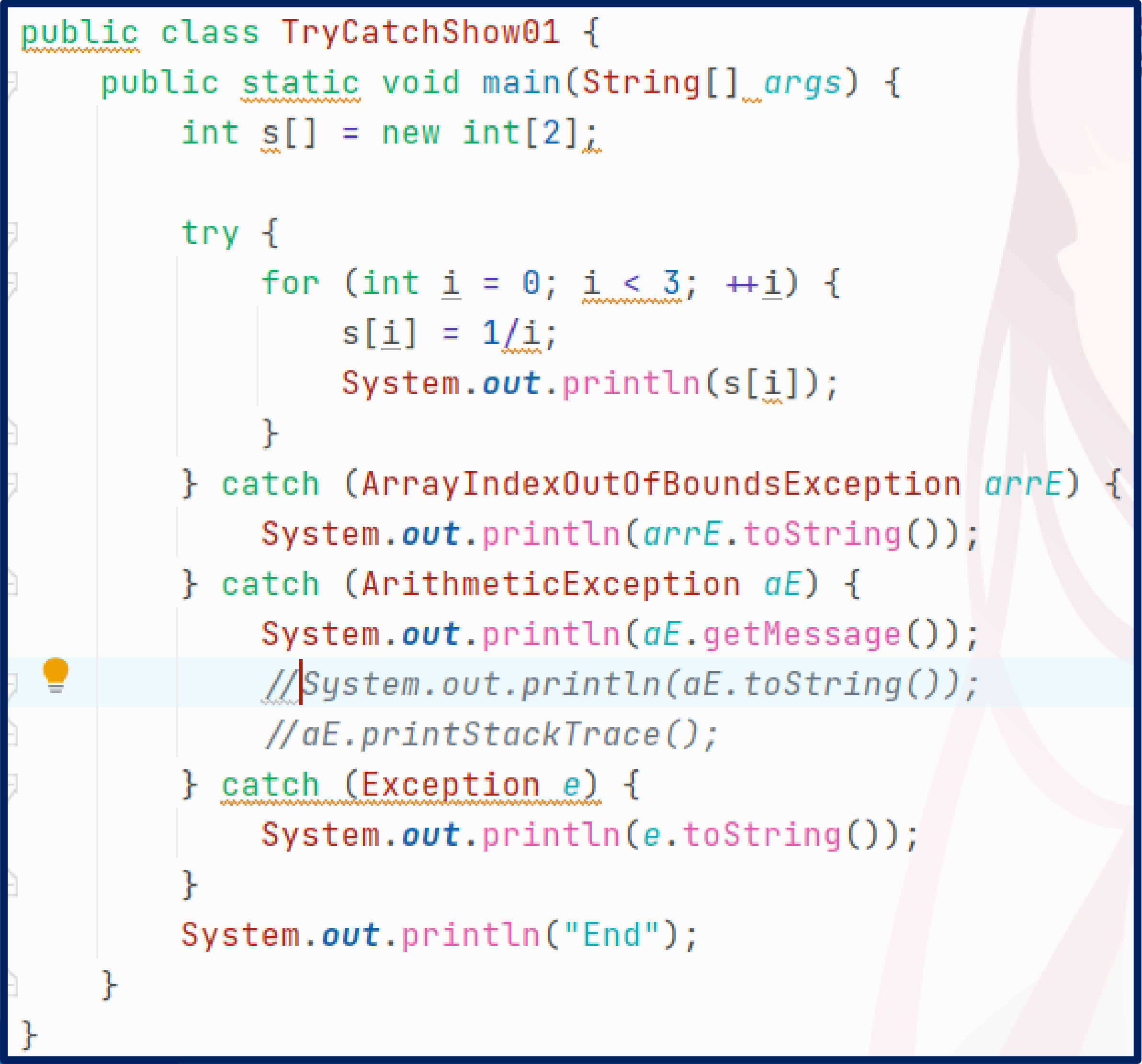
<u>in Integer class:</u>

```
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

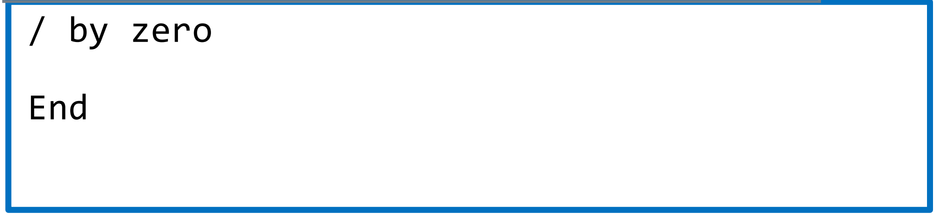
Try-Catch: Usage

TryCatchShow01.java



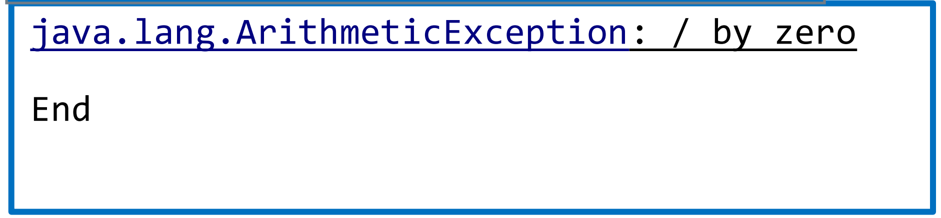
```java
public class TryCatchShow01 {
    public static void main(String[] args) {
        int s[] = new int[2];

        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.getMessage());
            //System.out.println(aE.toString());
            //aE.printStackTrace();
        } catch (Exception e) {
            System.out.println(e.toString());
        }
        System.out.println("End");
    }
}
```

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



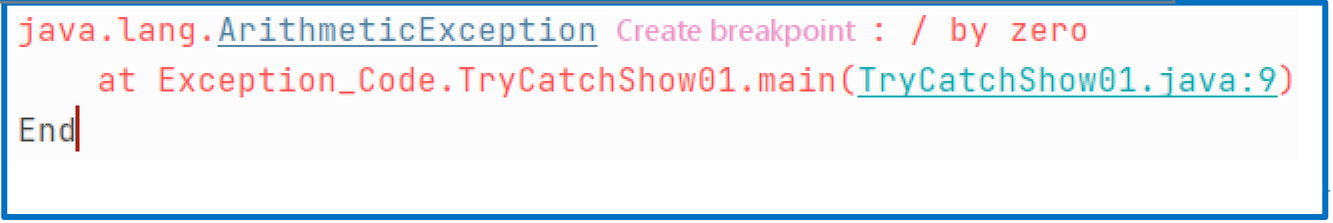
```
/ by zero

End
```

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



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

End
```

aE.printStackTrace()



```
java.lang.ArithmeticException Create breakpoint : / by zero
	at Exception_Code.TryCatchShow01.main(TryCatchShow01.java:9)
End
```

Try-Catch: Comparing to if-else

Pseudo code to read file

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

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 memor
    } 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;
}
```

- 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)

```java
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)

```java
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!

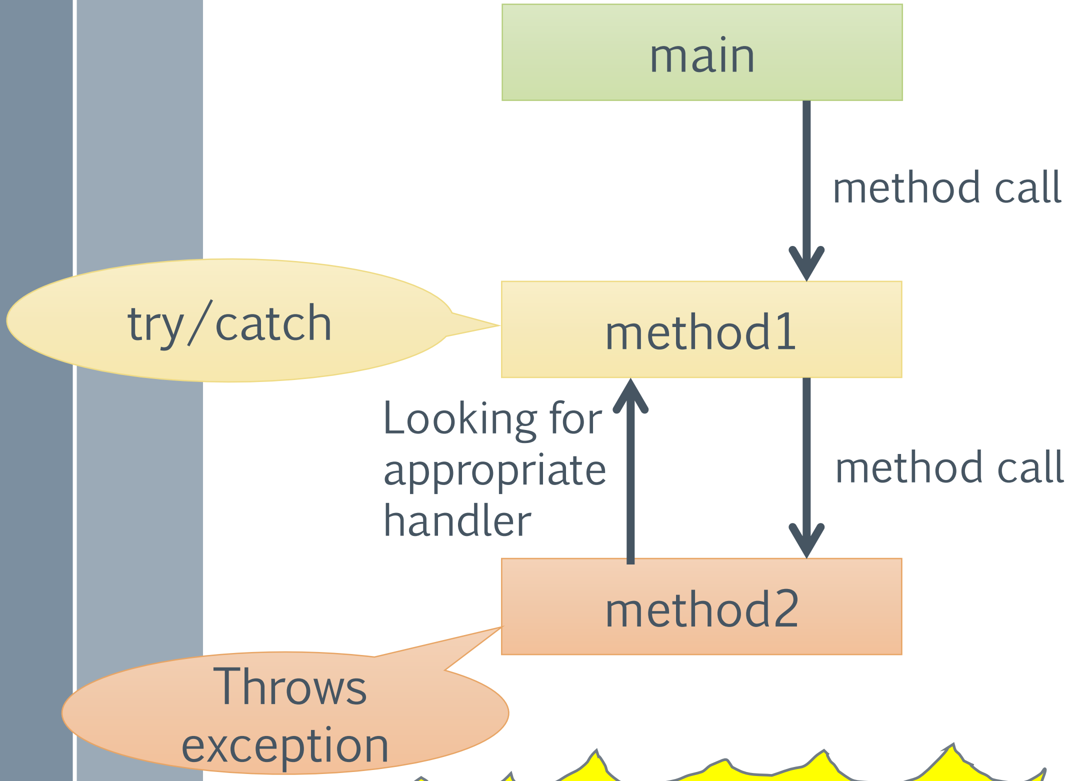
Throw1.java

```java
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



It allows us to write try/catch block in one location

ChainCaller.java

```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….

<u>Error!!</u>
m2 has to either catch or throw IOException

<u>Error!!</u>
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

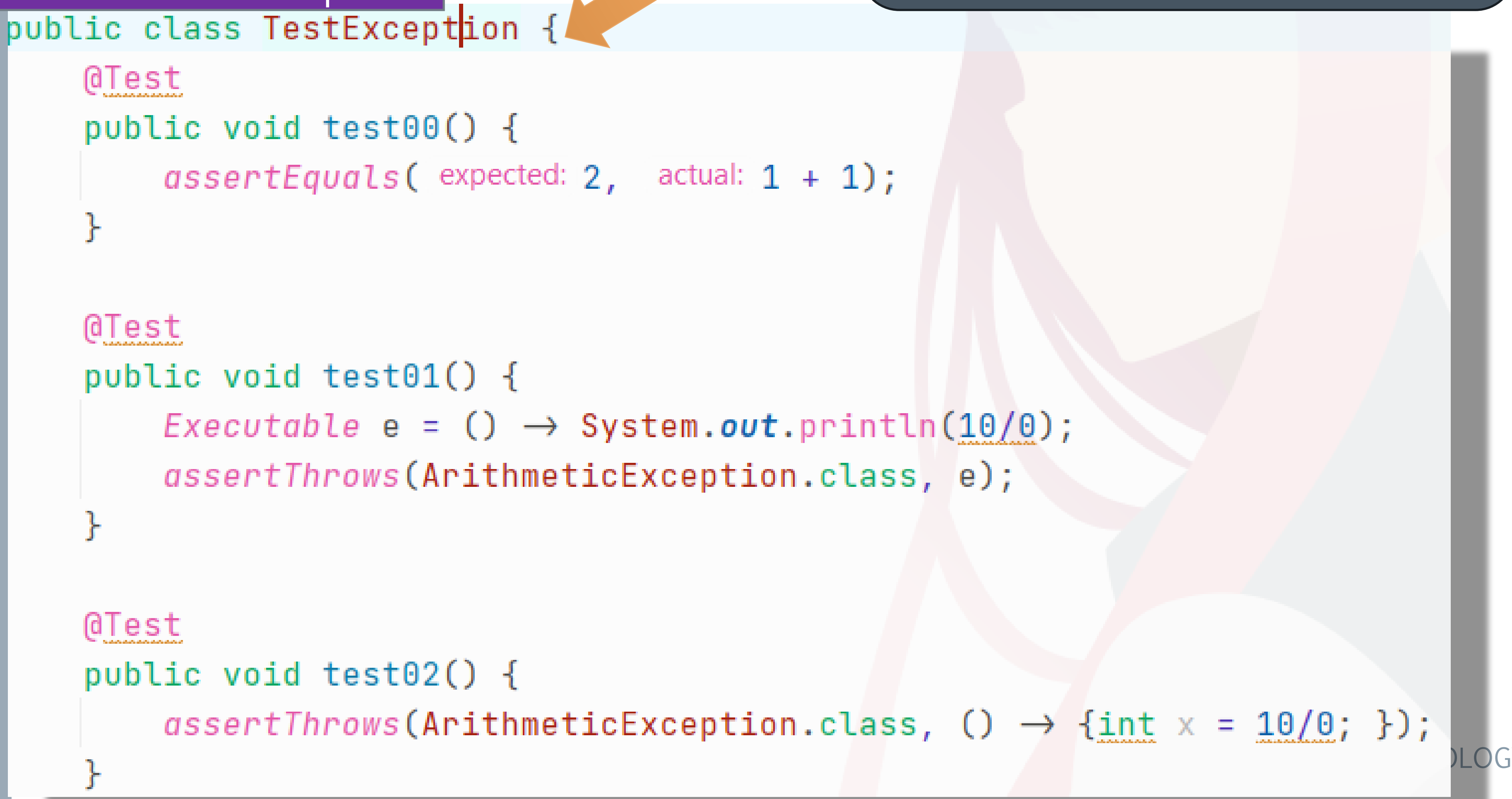
```
MyException = C# not allowed here
MyException.
```

Unit Testing (with-without Exception)

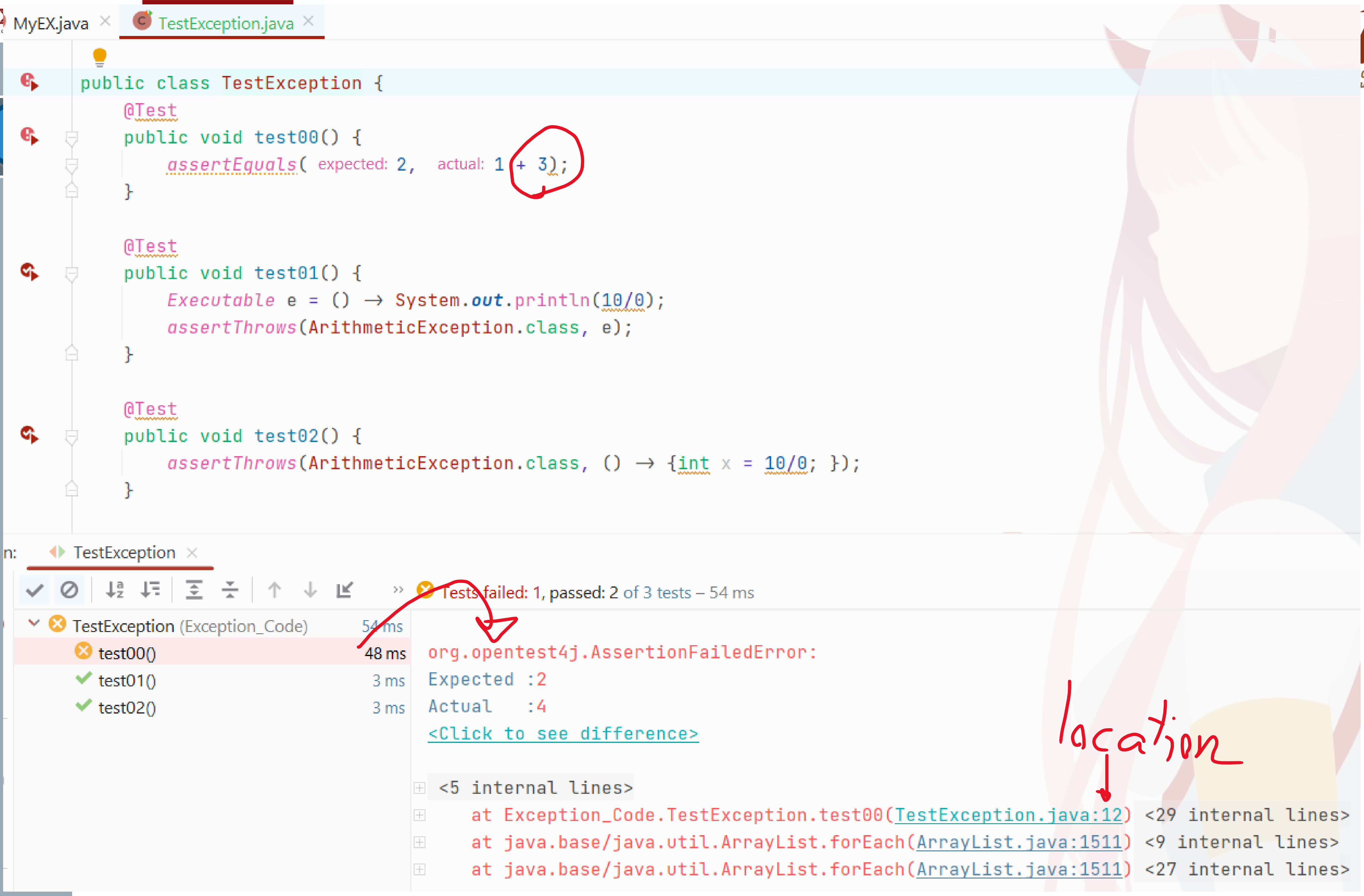
- JUnit is a way to test each Java method.
- It’s already setup in IntelliJ

Right click to run all tests. Or you can right click on a test to run only that test

Junit 5: TestException



```
public class TestException {
    @Test
    public void test00() {
        assertEquals( expected: 2,  actual: 1 + 1);
    }

    @Test
    public void test01() {
        Executable e = () -> System.out.println(10/0);
        assertThrows(ArithmeticException.class, e);
    }

    @Test
    public void test02() {
        assertThrows(ArithmeticException.class, () -> {int x = 10/0; });
    }
```



```
public class TestException {
    @Test
    public void test00() {
        assertEquals( expected: 2,  actual: 1 + 3);
    }

    @Test
    public void test01() {
        Executable e = () → System.out.println(10/0);
        assertThrows(ArithmeticException.class, e);
    }

    @Test
    public void test02() {
        assertThrows(ArithmeticException.class, () → {int x = 10/0; });
    }
```

```
Tests failed: 1, passed: 2 of 3 tests – 54 ms

TestException (Exception_Code)   54 ms
    test00()                     48 ms
    test01()                      3 ms
    test02()                      3 ms
```

```
org.opentest4j.AssertionFailedError: 
Expected :2
Actual   :4
<Click to see difference>

<5 internal lines>
    at Exception_Code.TestException.test00(TestException.java:12) <29 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>
    at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>
```

location