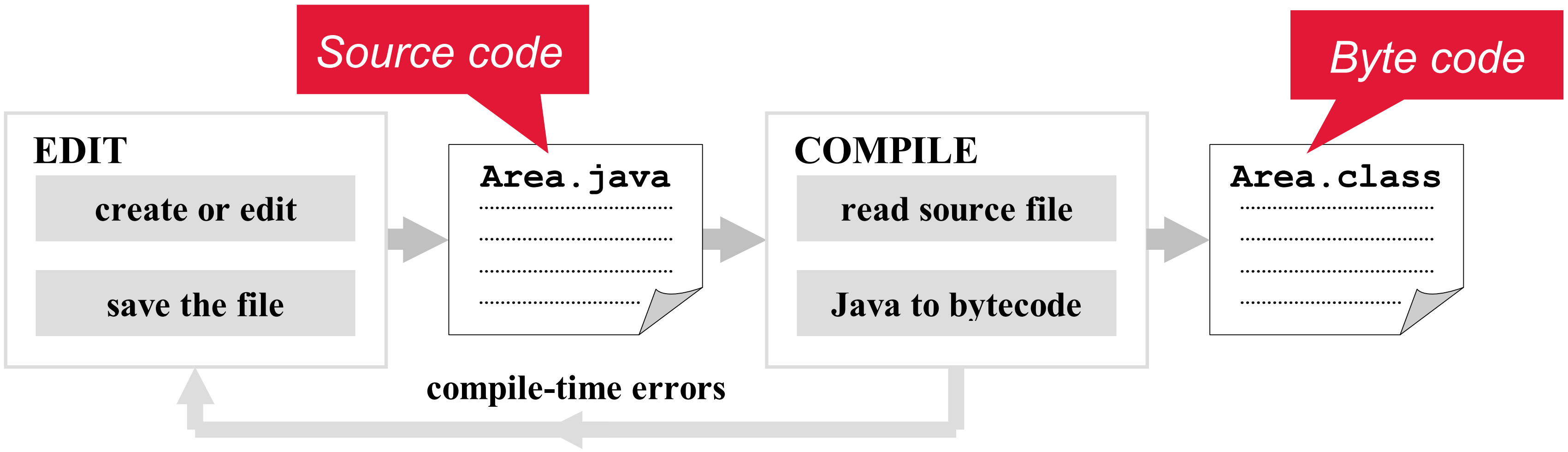
EECS 1720
Building Interactive Systems

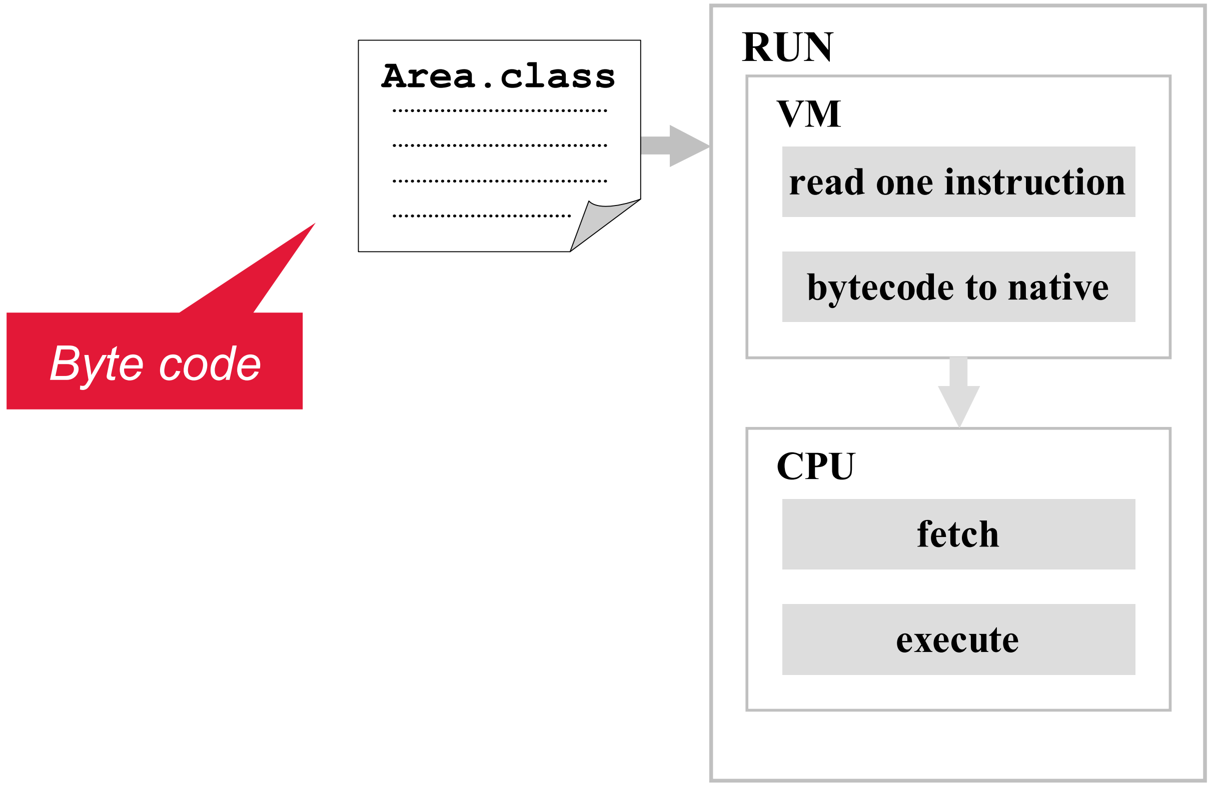
Lecture 5 :: Introduction to Exceptions (1)

Slides partially adapted based on Java by abstraction: A client-view approach (4th edition), H. Roumani (2015).

Source code vs. Byte Code (review)



Java Virtual Machine JVM (review)



JVM & Errors?

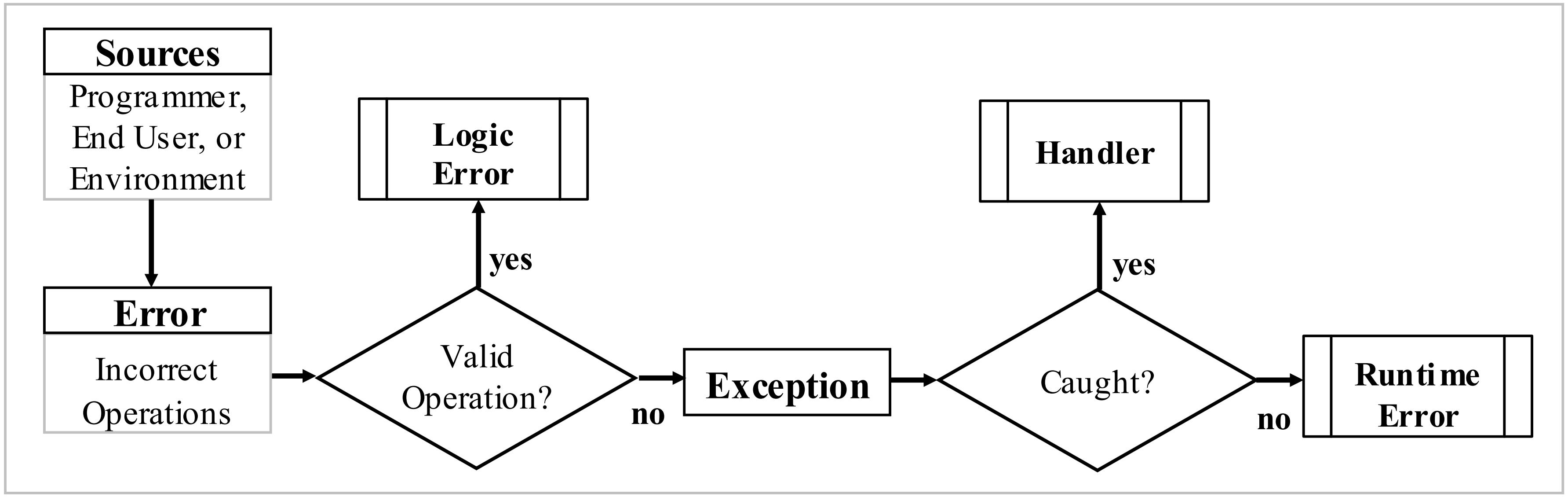
- Error Types?
  - Syntax
  - Semantic
  - Logic
- Caught/dealt with where?
  - Compiler `javac X.java`
  - JVM `java X`

Sources of these errors??

Compiler can detect limited errors only

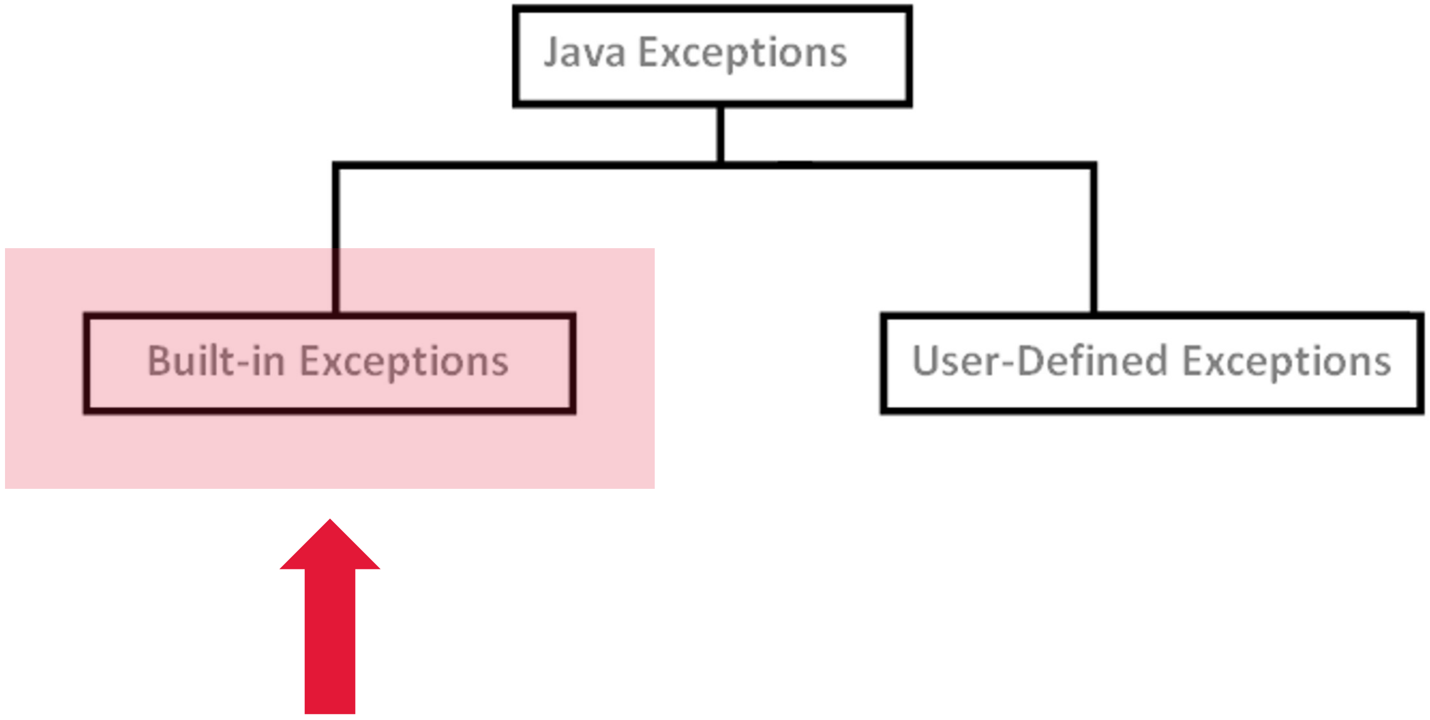
- The compiler:
  - checks syntax and turns syntactically-correct Java code into bytecode
  - does not check to see whether code could potentially raise exceptions
  - will not issue errors if the Java code may result in an exception at run-time

JVM – where are errors dealt with?



- An error source can lead to an incorrect operation
- An incorrect operations may be valid or invalid
- An invalid operation throws an exception
- An exception becomes a runtime error unless caught
- Caught exceptions can be handled gracefully

Exceptions (2 categories)



Our focus for now

Exceptions are classes that instantiate objects! These objects are subject to a special mechanism in java (throw/catch)

When something goes wrong, a special type of object (exception) is **instantiated** and made available (“thrown”) for processing by dedicated blocks of code.

Exception objects are meant to be “captured” and then handled by special blocks of code (try/catch statements)

If no such blocks are defined, then the program crashes with some (annoying) but useful information giving the user some clue as to what problem occurred

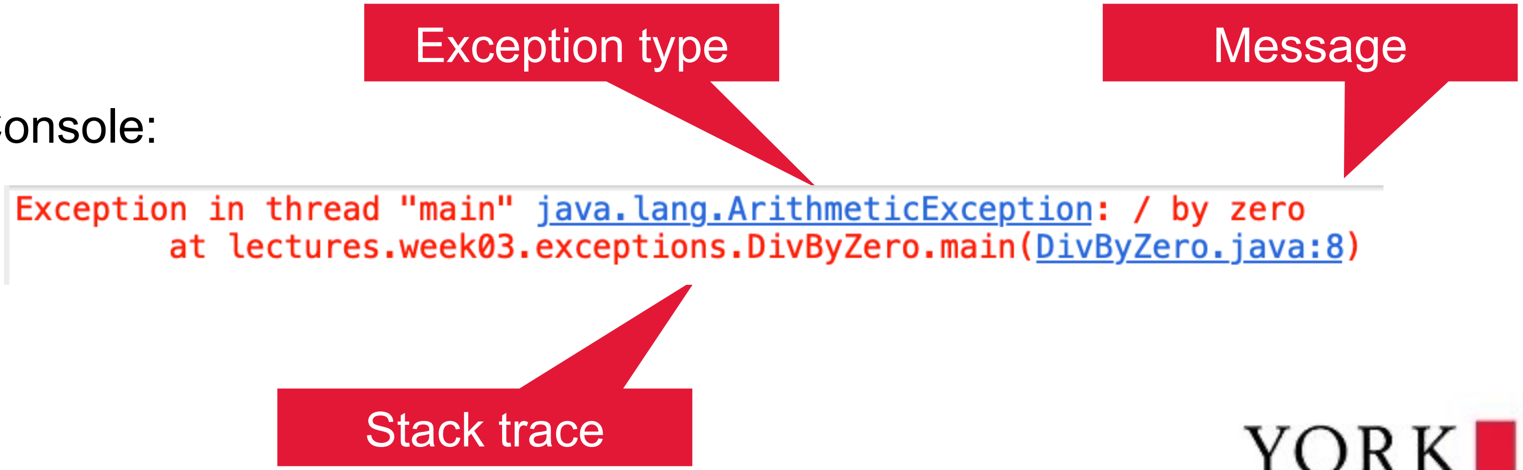
Example 1a

```
public class DivByZero {

    public static void main(String[] args) {
        int denom = 0;
        int result = 7 / denom;
    }
}
```

Example 1a

```java
public class DivByZero {

    public static void main(String[] args) {
        int denom = 0;
        int result = 7 / denom;
    }
}
```

Console:



```
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at lectures.week03.exceptions.DivByZero.main(DivByZero.java:8)
```

Example 1a

```
public class DivByZero {

    public static void main(String[] args) {
        int denom = 0;
        int result = 7 / denom;
    }
}
```

In this case:

- The error source is the **programmer**.
- The incorrect operation is invalid
- The exception was not caught

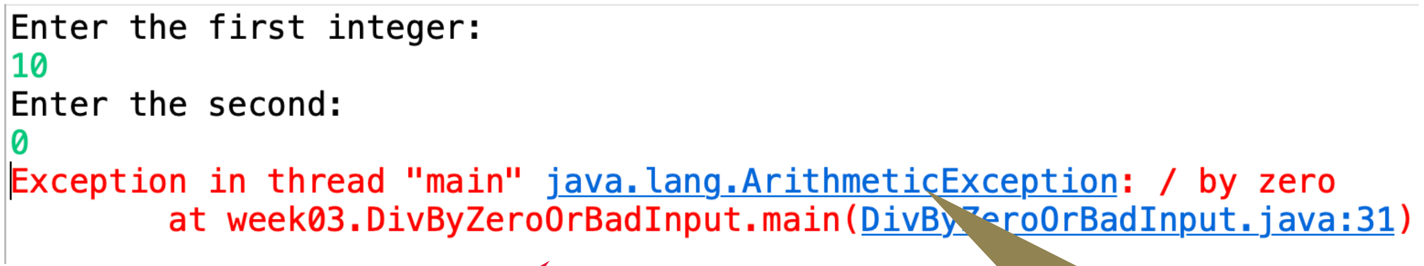
Exception not caught by JVM?

- When an exception is not caught/handled by the program
- It becomes a run-time error
  - Causes program to crash
  - Dumps output to console (red text)
    - Exception information
    - Stack Trace information
      - Stack trace is information about all the methods that are currently in progress when the program crashed

Example 1b

```
import java.util.Scanner;


public class DivByZeroOrBadInput {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);

        System.out.println("Enter the first integer:");
        int a = in.nextInt();

        System.out.println("Enter the second:");
        int b = in.nextInt();

        int c = a / b;
        System.out.println("Their quotient is: " + c);

    }

}
```

Example 1b

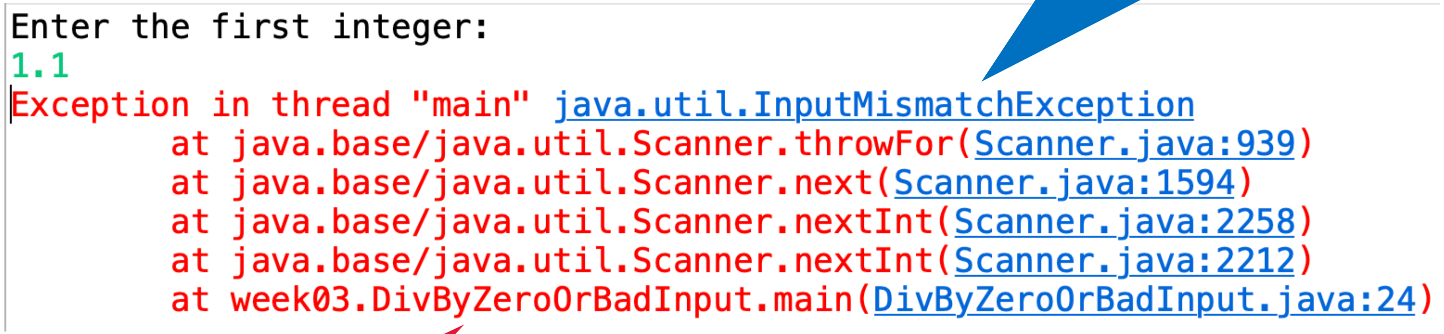


```
Enter the first integer:
10
Enter the second:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at week03.DivByZeroOrBadInput.main(DivByZeroOrBadInput.java:31)
```

Stack trace

Arithmetic Exception

InputMismatch Exception

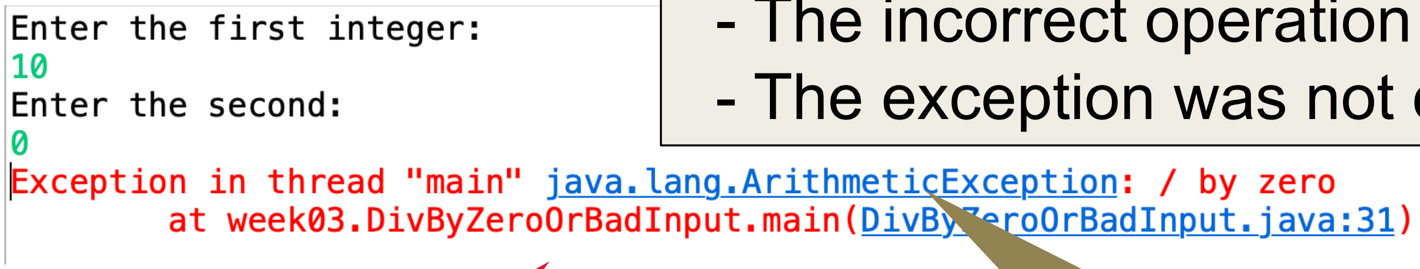


```
Enter the first integer:
1.1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at week03.DivByZeroOrBadInput.main(DivByZeroOrBadInput.java:24)
```

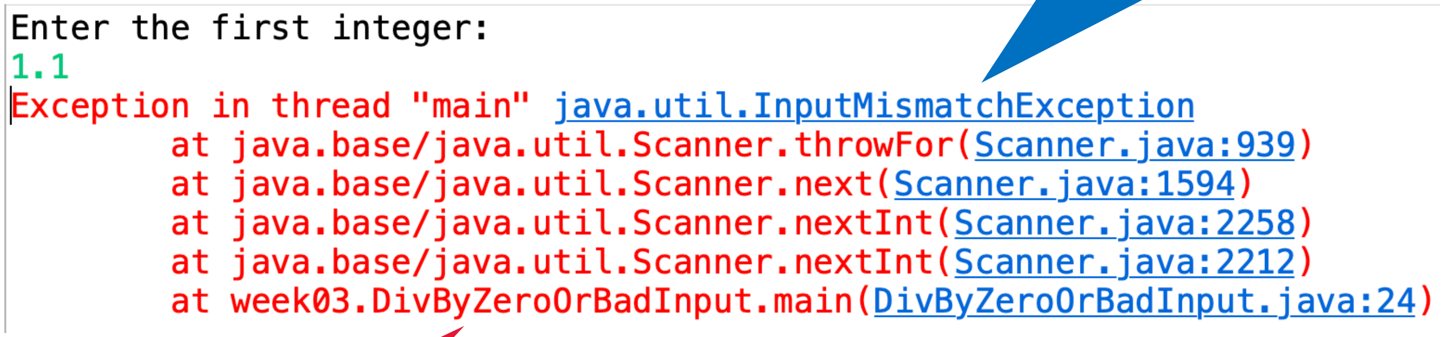
Stack trace

Example 1b

In this case:
- The error source is the **end user**.
- The incorrect operation is invalid
- The exception was not caught



```
Enter the first integer:
10
Enter the second:
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at week03.DivByZeroOrBadInput.main(DivByZeroOrBadInput.java:31)
```

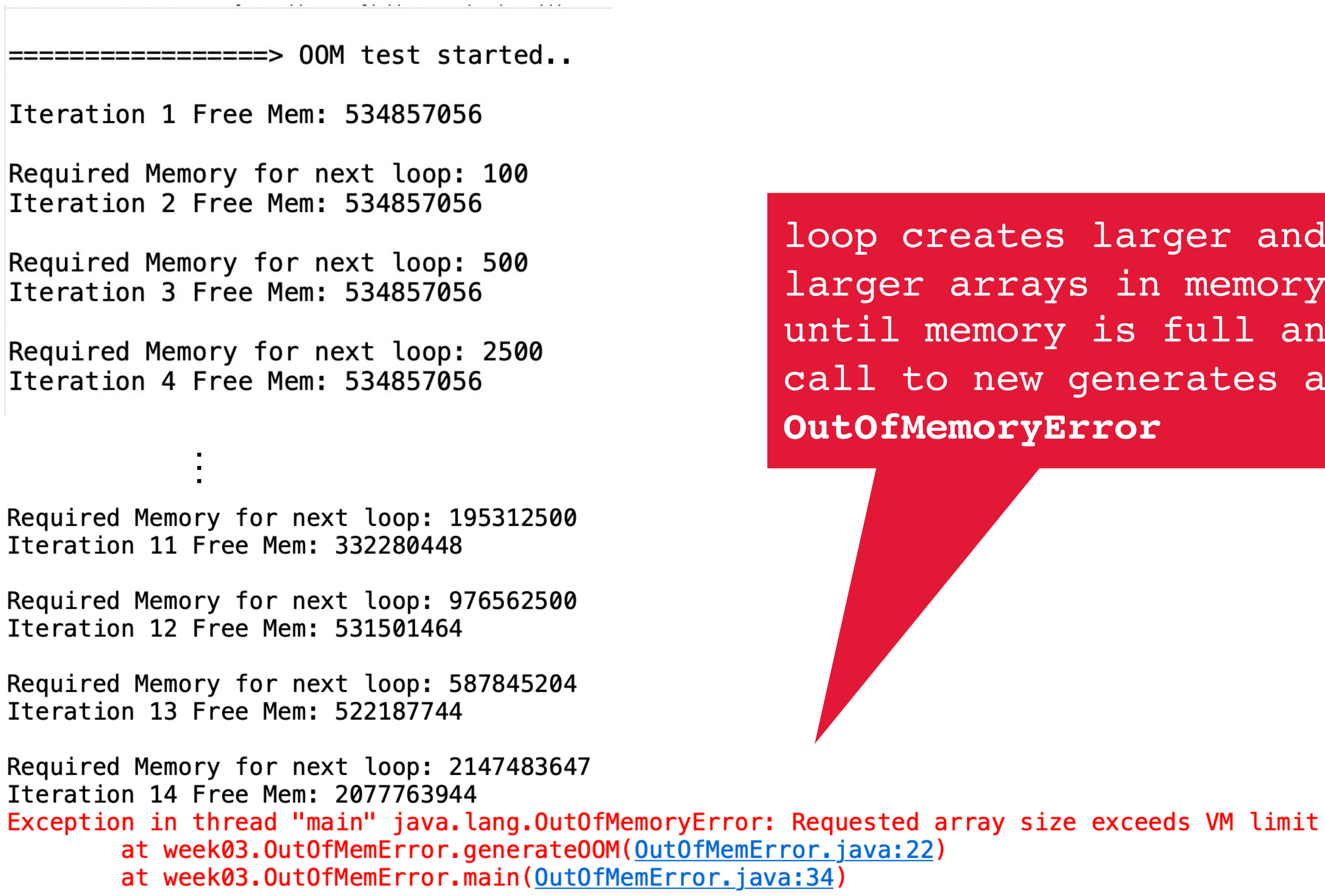


```
Enter the first integer:
1.1
Exception in thread "main" java.util.InputMismatchException
	at java.base/java.util.Scanner.throwFor(Scanner.java:939)
	at java.base/java.util.Scanner.next(Scanner.java:1594)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
	at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
	at week03.DivByZeroOrBadInput.main(DivByZeroOrBadInput.java:24)
```

Example 1c

```
public class OutOfMemError {

     public OutOfMemError() {
          // construct instance, no fields to set
     }

     public void generateOOM() throws Exception {
          int iteratorValue = 20;
          System.out.println("\n=================> OOM test started..\n");

          // create bigger and bigger arrays until no more memory!
          for (int outerIterator = 1; outerIterator < 20; outerIterator++) {
               System.out.println("Iteration " + outerIterator
                                + " Free Mem: " +  Runtime.getRuntime().freeMemory());
               int loop1 = 2;
               int[] memoryFillIntVar = new int[iteratorValue];   // create new array

               iteratorValue = iteratorValue * 5;
               System.out.println("\nRequired Memory for next loop: " + iteratorValue);
               Thread.sleep(1000);  // pauses briefly
          }
     }

     public static void main(String[] args) throws Exception {
          OutOfMemError memoryTest = new OutOfMemError();
          memoryTest.generateOOM();
     }
}
```

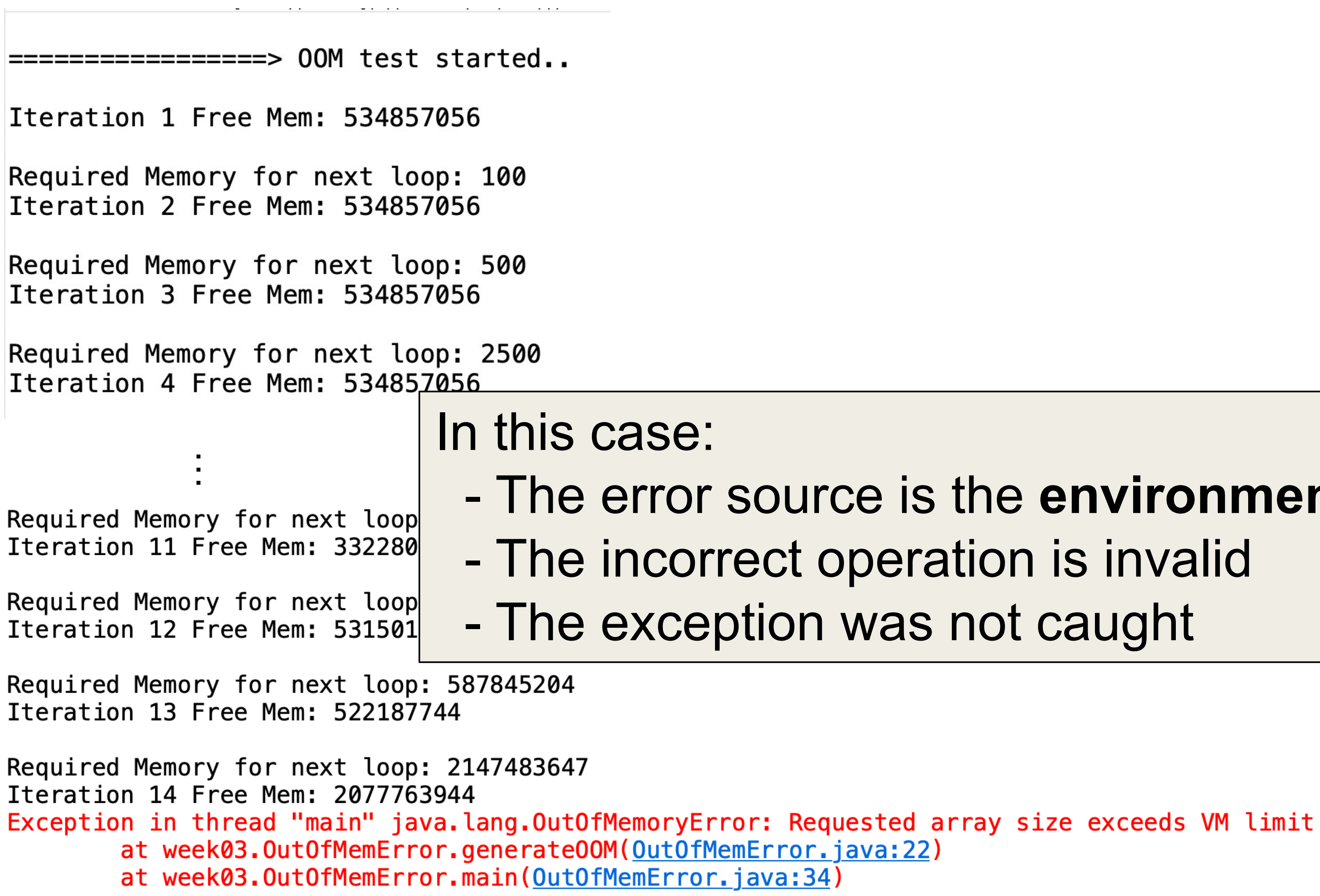
Example 1c



```
=================> OOM test started..

Iteration 1 Free Mem: 534857056

Required Memory for next loop: 100
Iteration 2 Free Mem: 534857056

Required Memory for next loop: 500
Iteration 3 Free Mem: 534857056

Required Memory for next loop: 2500
Iteration 4 Free Mem: 534857056

            ⋮

Required Memory for next loop: 195312500
Iteration 11 Free Mem: 332280448

Required Memory for next loop: 976562500
Iteration 12 Free Mem: 531501464

Required Memory for next loop: 587845204
Iteration 13 Free Mem: 522187744

Required Memory for next loop: 2147483647
Iteration 14 Free Mem: 2077763944
Exception in thread "main" java.lang.OutOfMemoryError: Requested array size exceeds VM limit
	at week03.OutOfMemError.generateOOM(OutOfMemError.java:22)
	at week03.OutOfMemError.main(OutOfMemError.java:34)
```

loop creates larger and larger arrays in memory until memory is full and a call to new generates an **`OutOfMemoryError`**

Example 1c



```
=================> OOM test started..

Iteration 1 Free Mem: 534857056

Required Memory for next loop: 100
Iteration 2 Free Mem: 534857056

Required Memory for next loop: 500
Iteration 3 Free Mem: 534857056

Required Memory for next loop: 2500
Iteration 4 Free Mem: 534857056

            ⋮

Required Memory for next loop
Iteration 11 Free Mem: 332280

Required Memory for next loop
Iteration 12 Free Mem: 531501

Required Memory for next loop: 587845204
Iteration 13 Free Mem: 522187744

Required Memory for next loop: 2147483647
Iteration 14 Free Mem: 2077763944
Exception in thread "main" java.lang.OutOfMemoryError: Requested array size exceeds VM limit
	at week03.OutOfMemError.generateOOM(OutOfMemError.java:22)
	at week03.OutOfMemError.main(OutOfMemError.java:34)
```

In this case:
- The error source is the **environment**.
- The incorrect operation is invalid
- The exception was not caught

CRASH vs. CLEAN EXIT

- If an exception is not caught & handled, the run-time error will cause the program to suddenly terminate (with a message) ~ **CRASH**

- A **CRASH** looks bad (to end user), and can also lead to data loss, file corruption, etc. as sudden termination prevents the opportunity for a CLEAN EXIT

- **CLEAN EXIT**: ability to end the program in a controlled way (save what needs to be saved, free memory, close open files, etc..)

How are Exceptions handled?

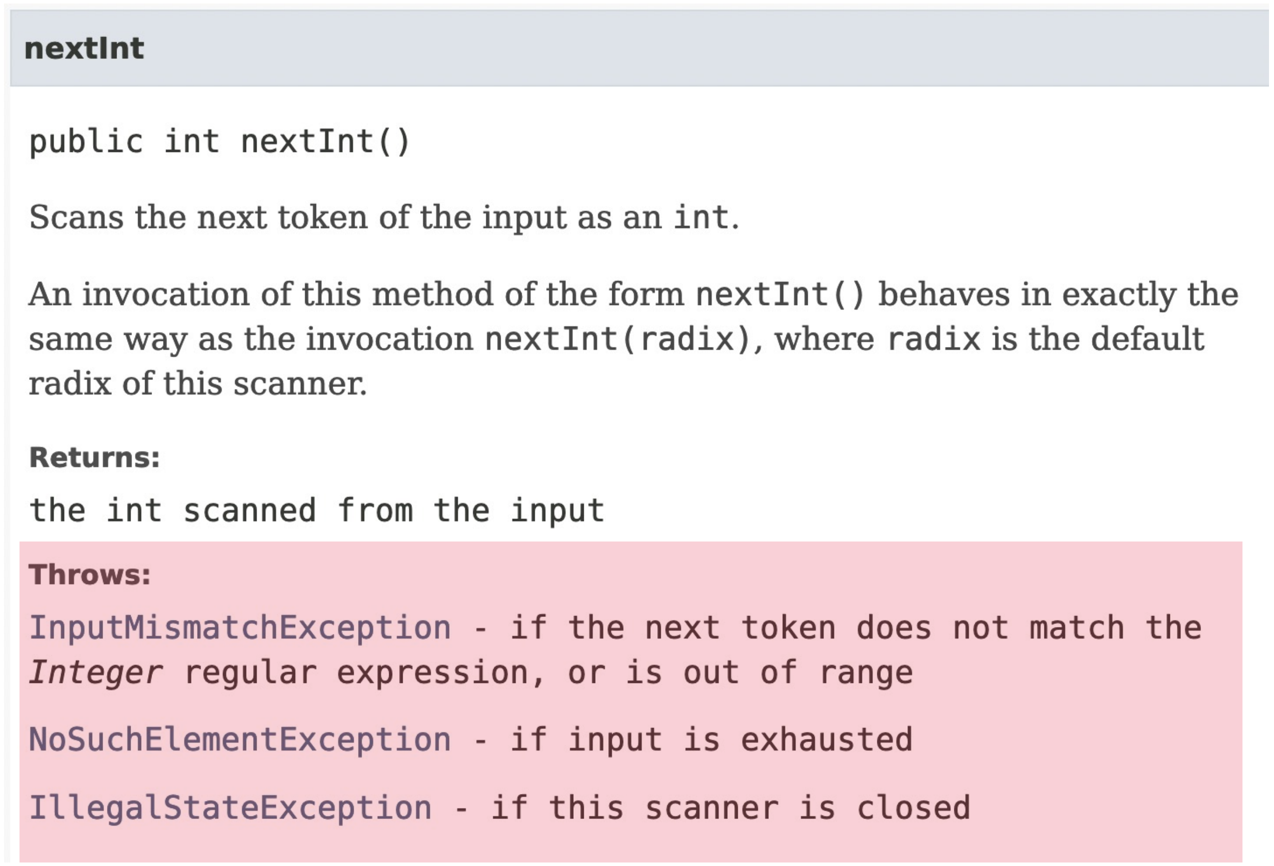
- THROW/CATCH → DELEGATION MODEL:
  - Client (main) invokes method A
    - Method A invokes method B
      - An invalid operation occurs in B !! (throws exception)
        - If B handles exception, all good (none the wiser)
      - otherwise B delegates (passes) exception back to A
    - If A handles, all good (again none the wiser), otherwise A passes exception back to client
  - Client has the option of handling the exception or not
  - If Nothing handled, then exception passed to JVM, and JVM causes a **RUN TIME ERROR**

Delegation Model Policy

HANDLE OR DELEGATE BACK

- Applies to all (components and client)
- The API must document any back delegation
- It does so under the heading: “Throws”

Example → Scanner API → nextInt()



nextInt

```
public int nextInt()
```

Scans the next token of the input as an `int`.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:

`the int scanned from the input`

Throws:

`InputMismatchException - if the next token does not match the` *`Integer`* `regular expression, or is out of range`

`NoSuchElementException - if input is exhausted`

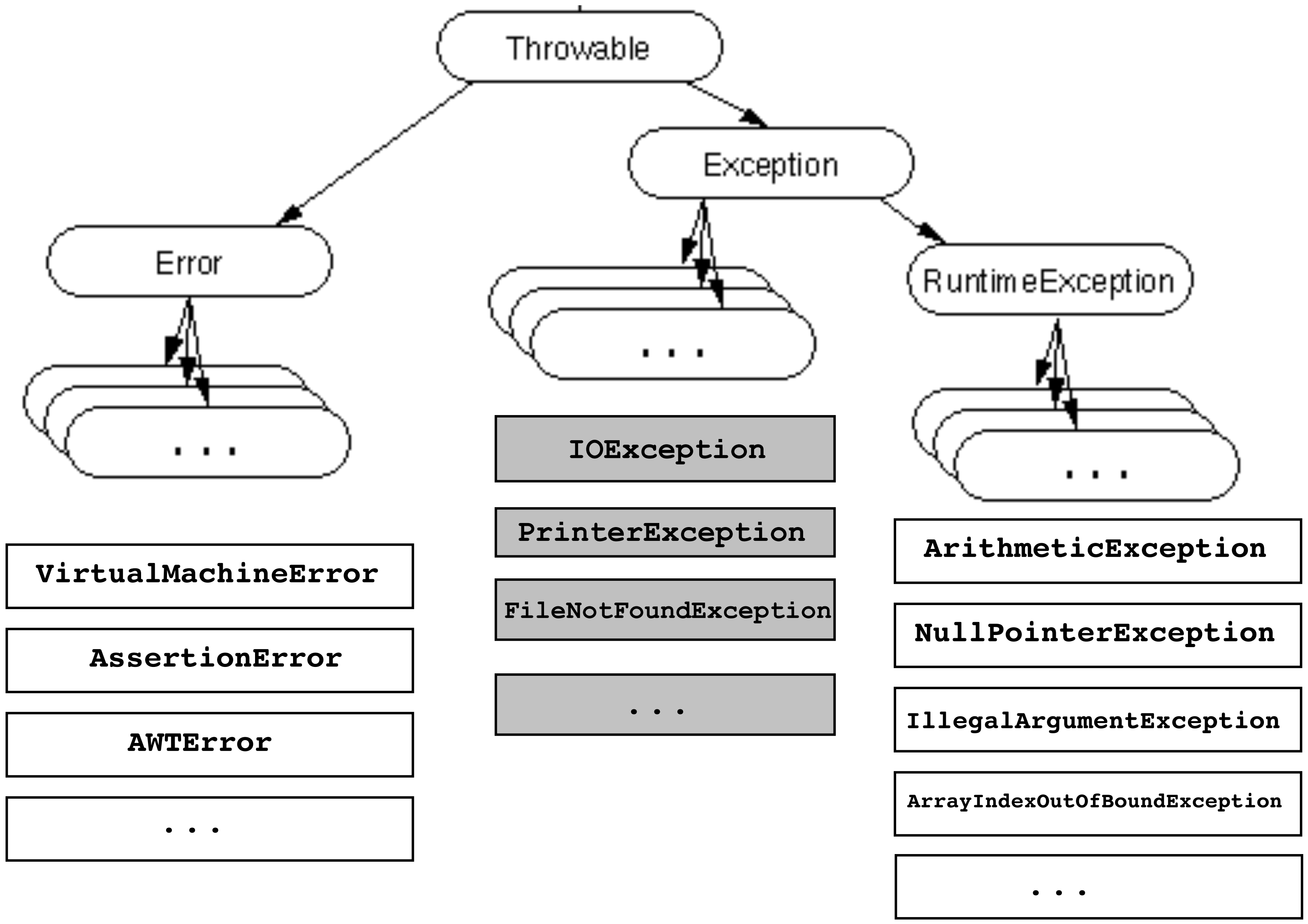
`IllegalStateException - if this scanner is closed`

To “throw” an exception means…

- To “spawn” or “create” a special type of object that indicates some kind of potential error/issue
    - a “thrown” object becomes available for processing via the delegation policy
- An Exception is really just a special type of <u>object</u> (by nature)
    - An object of a specific type of class
    - There are categories of different kinds of “throwable” classes that relate to Exceptions & Errors
- An Exception object has:
    - <u>state:</u> class fields that store information about the Exception/Error
    - <u>behavior:</u> methods that can be used to query the state of this Exception/Error

YORK UNIVERSITÉ UNIVERSITY

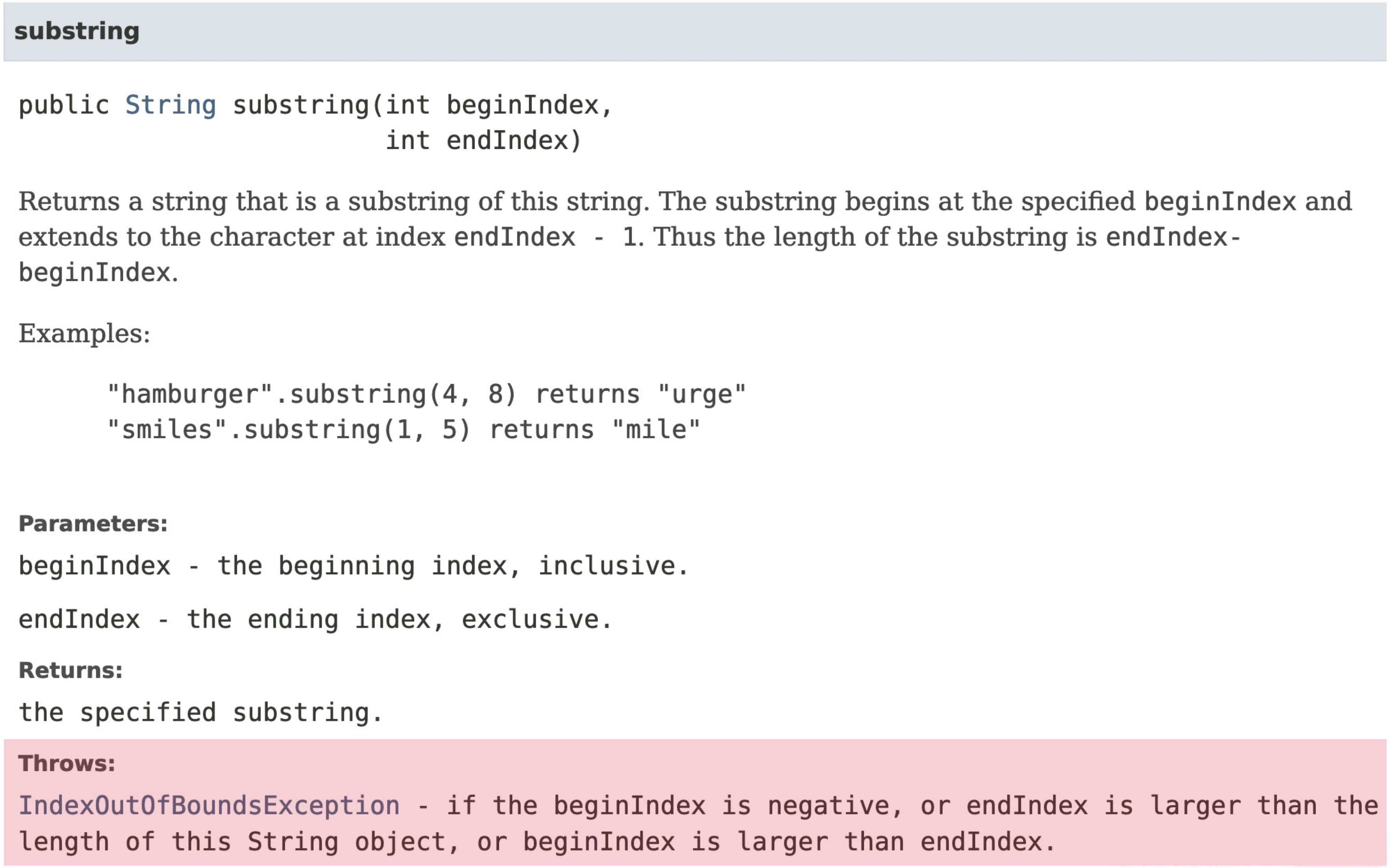
Categories of “throwable” errors/exceptions



Common built-in exceptions (in java.lang)

- **Arithmetic Exception**
  It is thrown when an exceptional condition has occurred in an arithmetic operation.
- **ArrayIndexOutOfBoundException**
  It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.
- **ClassNotFoundException**
  This Exception is raised when we try to access a class whose definition is not found
- **FileNotFoundException**
  This Exception is raised when a file is not accessible or does not open.
- **IOException**
  It is thrown when an input-output operation failed or interrupted
- **InterruptedException**
  It is thrown when a thread is waiting , sleeping , or doing some processing , and it is interrupted.
- **NoSuchFieldException**
  It is thrown when a class does not contain the field (or variable) specified
- **NoSuchMethodException**
  It is thrown when accessing a method which is not found.
- **NullPointerException**
  This exception is raised when referring to the members of a null object. Null represents nothing
- **NumberFormatException**
  This exception is raised when a method could not convert a string into a numeric format.
- **RuntimeException**
  This represents any exception which occurs during runtime.
- **StringIndexOutOfBoundsException**
  It is thrown by String class methods to indicate that an index is either negative than the size of the string

Example → String API → substring(..)



substring

```
public String substring(int beginIndex,
                        int endIndex)
```

Returns a string that is a substring of this string. The substring begins at the specified `beginIndex` and extends to the character at index `endIndex - 1`. Thus the length of the substring is `endIndex-beginIndex`.

Examples:

```
     "hamburger".substring(4, 8) returns "urge"
     "smiles".substring(1, 5) returns "mile"
```

Parameters:

`beginIndex` - the beginning index, inclusive.

`endIndex` - the ending index, exclusive.

Returns:

the specified substring.

Throws:

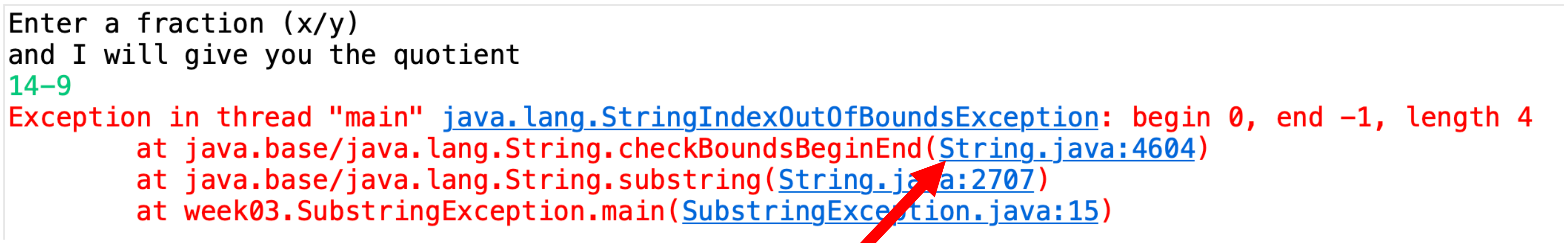
`IndexOutOfBoundsException` - if the `beginIndex` is negative, or `endIndex` is larger than the length of this `String` object, or `beginIndex` is larger than `endIndex`.

Example 2

```
import java.util.Scanner;

public class SubstringException {

    public static void main(String[] args) {

        Scanner in = new Scanner(System.in);
        System.out.println("Enter a fraction (x/y) ");
        System.out.println("and I will give you the quotient");
        String str = in.nextLine();

        int slash = str.indexOf("/") ;
        String left = str.substring(0, slash);
        String right = str.substring(slash + 1);

        int numer = Integer.parseInt(left);
        int denom = Integer.parseInt(right);
        int quotient = numer/denom;

        System.out.println("Quotient = " + quotient);
        in.close();
    }
}
```

Example 2

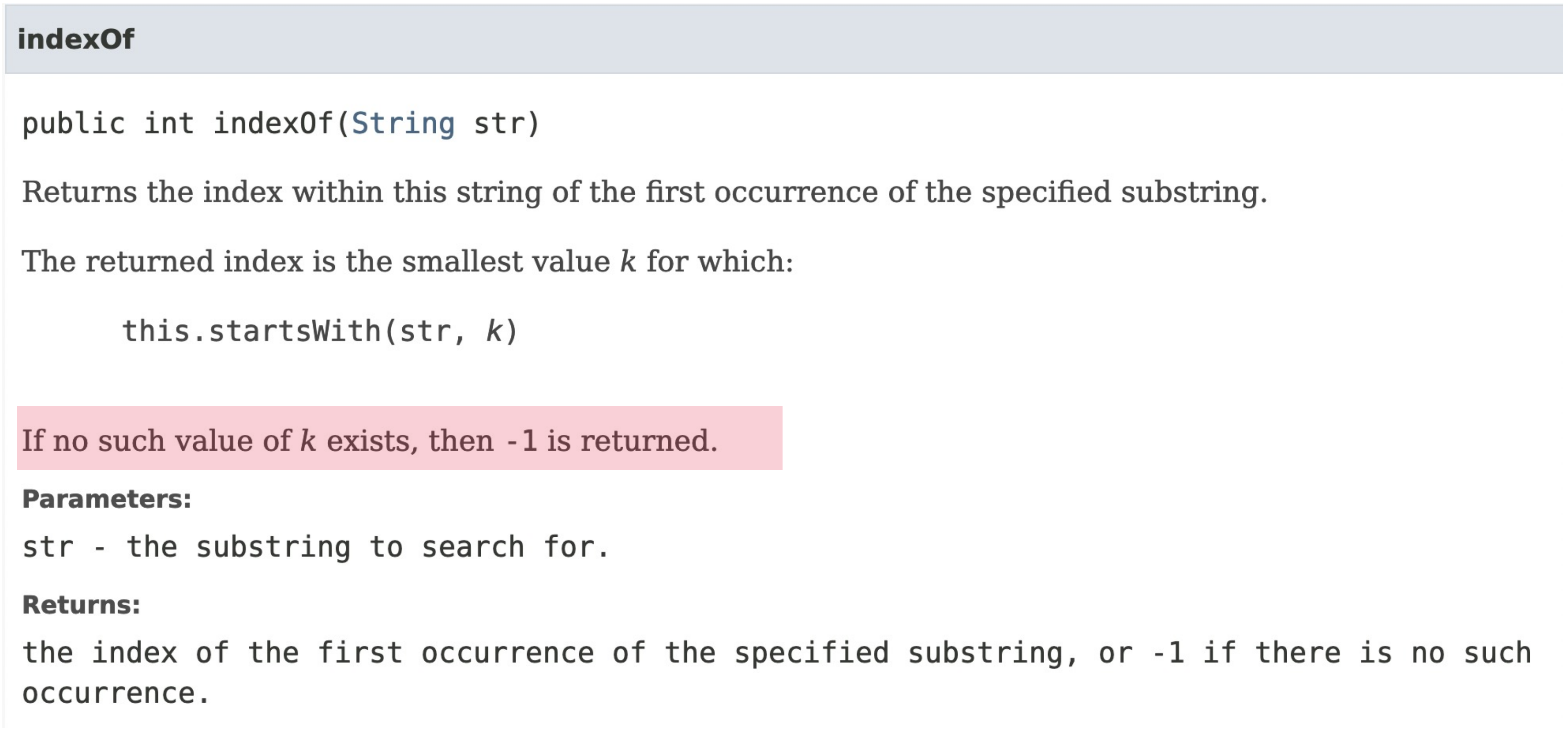
- Here is a sample run with input string: `“14-9”`



```
Enter a fraction (x/y)
and I will give you the quotient
14-9
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: begin 0, end -1, length 4
	at java.base/java.lang.String.checkBoundsBeginEnd(String.java:4604)
	at java.base/java.lang.String.substring(String.java:2707)
	at week03.SubstringException.main(SubstringException.java:15)
```

The trace follows the delegation from line 4604 within String class to line 15 within the client.

Why was the end index -1?



indexOf

```
public int indexOf(String str)
```

Returns the index within this string of the first occurrence of the specified substring.

The returned index is the smallest value *k* for which:

```
      this.startsWith(str, k)
```

If no such value of *k* exists, then -1 is returned.

Parameters:

`str - the substring to search for.`

Returns:

`the index of the first occurrence of the specified substring, or -1 if there is no such occurrence.`

So how to "handle" the exception? → with `try-catch` blocks:

```
try
{   ...
    code fragment
    ...
}
catch (SomeType e)
{   ...
    exception handler
    ...
}
program continues
```

Code that can possibly throw exception(s)

an exception argument (i.e. an exception type that might get thrown)

Alternative code to run if in fact an exception occurred within the try { }

Example 3:
add exception handling to Example 1

```
public class DivByZero {

    public static void main(String[] args) {


            int denom = 1;
            int result = 7 / denom;














    }
}
```

Example 3:
add exception handling to Example 1

```
public class DivByZeroHandled {

    public static void main(String[] args) {

        try {
            int denom = 1;
            int result = 7 / denom;
        }
        catch (ArithmeticException e) {

            System.out.println("I caught it!");




        }

        System.out.println("program finished.");

    }
}
```

Using the Exception object

```
public class DivByZeroHandled {

    public static void main(String[] args) {

        try {
            int denom = 1;
            int result = 7 / denom;
        }
        catch (ArithmeticException e) {

            System.out.println("I caught it!");

            e.printStackTrace();
            System.out.println(e.getMessage());


        }

        System.out.println("program finished.");

    }
}
```

Using the Exception object + graceful exit

```
public class DivByZeroHandled {

    public static void main(String[] args) {

        try {
            int denom = 1;
            int result = 7 / denom;
        }
        catch (ArithmeticException e) {

            System.out.println("I caught it!");

            e.printStackTrace();
            System.out.println(e.getMessage());
            System.out.println(".. exiting gracefully");
            System.exit(0);
        }

        System.out.println("program finished.");

    }
}
```

Example 4a:
add exception handling to Example 2

```
import java.util.Scanner;

public class SubstringException {

     public static void main(String[] args) {

          Scanner in = new Scanner(System.in);
          System.out.println("Enter a fraction (x/y) ");
          System.out.println("and I will give you the quotient");
          String str = in.nextLine();

          int slash = str.indexOf("/") ;
          String left = str.substring(0, slash);
          String right = str.substring(slash + 1);

          int numer = Integer.parseInt(left);
          int denom = Integer.parseInt(right);
          int quotient = numer/denom;

          System.out.println("Quotient = " + quotient);
          in.close();
     }
}
```

```java
import java.util.Scanner;

public class SubstringExceptionHandled {

     public static void main(String[] args) {

          Scanner in = new Scanner(System.in);

          try {
               System.out.println("Enter a fraction (x/y) ");
               System.out.println("and I will give you the quotient");
               String str = in.nextLine();

               int slash = str.indexOf("/") ;
               String left = str.substring(0, slash);
               String right = str.substring(slash + 1);

               int numer = Integer.parseInt(left);
               int denom = Integer.parseInt(right);
               int quotient = numer/denom;

               System.out.println("Quotient = " + quotient);
          }
          catch ( ?? ) {




          }
          System.out.println("Exiting..");
          in.close();
     }
}
```

```
import java.util.Scanner;

public class SubstringExceptionHandled {

     public static void main(String[] args) {

          Scanner in = new Scanner(System.in);

          try {
               System.out.println("Enter a fraction (x/y) ");
               System.out.println("and I will give you the quotient");
               String str = in.nextLine();

               int slash = str.indexOf("/") ;
               String left = str.substring(0, slash);
               String right = str.substring(slash + 1);

               int numer = Integer.parseInt(left);
               int denom = Integer.parseInt(right);
               int quotient = numer/denom;

               System.out.println("Quotient = " + quotient);
          }
          catch ( StringIndexOutOfBoundsException e ) {

               System.out.println("No slash in input!");


          }
          System.out.println("Exiting..");
          in.close();
     }
}
```

Handling Multiple Exceptions?

```
try
{   ...
}
catch (Type-1 e)
{   ...
}
catch (Type-2 e)
{   ...
}
...
catch (Type-n e)
{   ...
}
program continues
```

Example 4b:

- Given a string containing two slash-delimited integers, write a program that outputs their quotient.
- **Use exception handling to handle all possible input errors.**

Note that when exception handling is used, do not code defensively;

i.e. assume the world is perfect and then worry about problems.

This separates the program logic from validation.

```
import java.util.Scanner;

public class SubstringExceptionHandled2 {

	public static void main(String[] args) {

		Scanner in = new Scanner(System.in);

		try {
			System.out.println("Enter a fraction (x/y) ");
			System.out.println("and I will give you the quotient");
			String str = in.nextLine();

			int slash = str.indexOf("/") ;
			String left = str.substring(0, slash);
			String right = str.substring(slash + 1);

			int numer = Integer.parseInt(left);
			int denom = Integer.parseInt(right);
			int quotient = numer/denom;

			System.out.println("Quotient = " + quotient);
		}
		catch ( ?? ) {





		}
		System.out.println("Exiting..");
		in.close();
	}
}
```

```java
import java.util.Scanner;

public class SubstringExceptionHandled2 {

	public static void main(String[] args) {

		Scanner in = new Scanner(System.in);

		try {

			// not shown



		}
		catch (StringIndexOutOfBoundsException e ) {
				System.out.println("No slash in input!");
		}
		catch (NumberFormatException e ) {
			System.out.println("Non-integer operands!");
		}
		catch (ArithmeticException e) {
				System.out.println("Cannot divide by zero!");
		}
		System.out.println("Exiting..");
		in.close();
	}
}
```

Takeaways

- Different categories of Exceptions
  - Many classes use Exceptions to indicate unforeseen issues that occur at **runtime**
  - allows for possibility of recovery in code

- Exceptions are objects that get instantiated & “thrown”

- May be ”caught” using try{} / catch {} blocks
  - If code in try{} triggers an exception, code suspended and program re-routed to any catch() blocks immediately following
  - catch offers some alternative code that can be run instead of the statements that triggered the exception

- If not handled, the program typically will crash with some error messages (stack trace + info from exception that occurred)