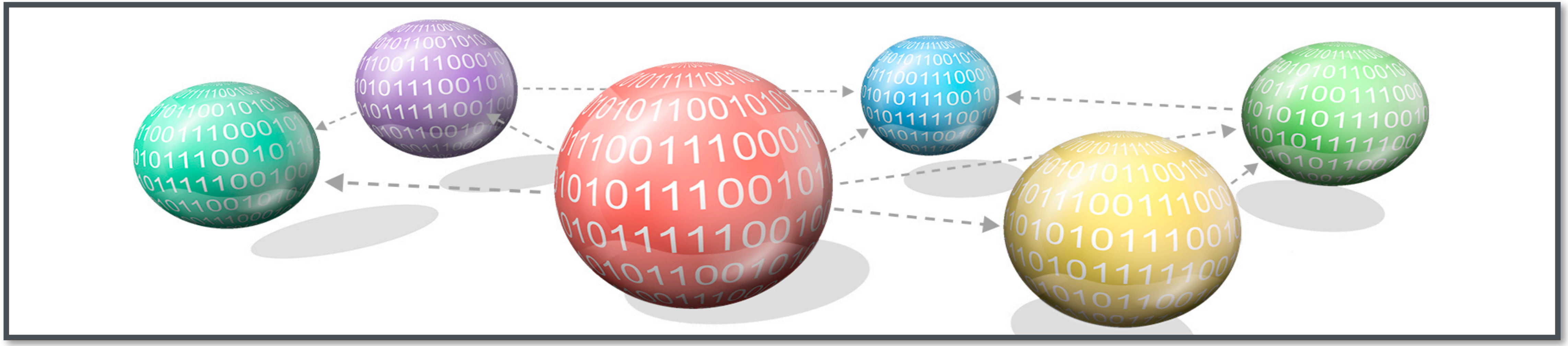
NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Chapter 4: Exception Handling

CE2002 Object Oriented Design & Programming

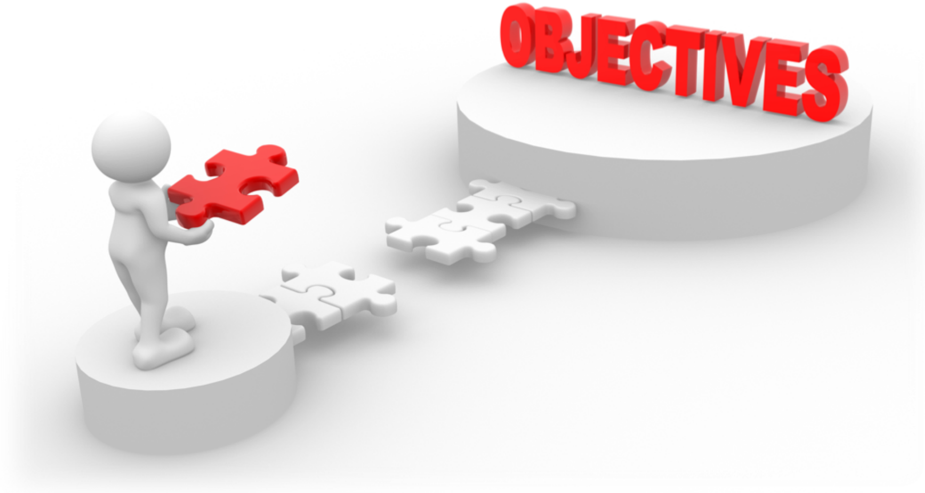
Dr Zhang Jie
Assoc Prof, School of Computer Science and Engineering (SCSE)



Learning Objectives

After the completion of this chapter, you should be able to:

- Handle error in Java.
- Handle Java’s exception.
- Summarise Java’s exception hierarchy.
- Use exception classes.
- Create your own exception classes.



Topic 1: Error Handling

Error Handling

- Exception: Run-time anomalies that a program may detect.
- Examples: Memory exhaustion, an input file cannot be opened, division by zero, etc.
  - May cause serious problems and disrupt normal execution of the program.
  - Well-designed programs should have error-handling code inside the program code to handle these run-time anomalies.
- In Java, exception handling is provided to catch and handle run-time exceptions.
- Exception handlers (or recovery procedure) may catch an exception in order to recover from the problem.

Computing Average Marks Version 1

```
import java.util.Scanner ;
public class AverageMarksV1 {
   public static void main( String[] args ) {
      int     i , numOfStudents ;
      double  totalMarks = 0 , avgMarks = 0 ;
      Scanner sc          = new Scanner( System.in );
      System.out.print( "Enter number of students: " );
      numOfStudents = sc.nextInt();
      System.out.print("Enter student marks: ");
      for ( i = 0 ; i < numOfStudents ; i++ )
         totalMarks += sc.nextDouble() ;
      avgMarks = totalMarks / (double)numOfStudents ;
                 // error if numOfStudents = 0
      System.out.println( "Average marks = " + avgMarks );
   }
}
```

```
Program Input and Output
Enter number of students: 5
Enter student marks: 70 80 90 60
50
Average marks = 70.0
```

Computing Average Marks Version 2

```
import java.util.Scanner ;
public class AverageMarksV2 {
  public static void main( String[] args ) {
    int     i , numOfStudents ;
    double  totalMarks = 0 , avgMarks = 0 ;
    Scanner sc = new Scanner( System.in );
    System.out.print( "Enter number of students: " );
    numOfStudents = sc.nextInt() ;
    if ( numOfStudents <= 0 ) {
      System.out.print(   "Error: no of students " );
      System.out.println( "must not equal to 0!"   );
      System.out.println( "Program Terminating!"   );
      System.exit( 0 ) ;
    }
    System.out.print( "Enter student marks: " );
    for ( i = 0 ; i < numOfStudents ; i++ )
      totalMarks += sc.nextDouble() ;
    avgMarks = totalMarks / (double)numOfStudents ;
    System.out.println( "Average marks = " + avgMarks );
  }
}
```

```
Program Input and Output
Enter number of students: 0
Error: no of students must not equal to 0!
Program Terminating!
```

Topic 2: Java’s Exception Handling

Java’s Exception Handling

- **Trying an Exception**:
  - Try to see if there is any exception.
  - Every method must first state the types of exceptions it can handle.

- **Throwing an Exception:**
  - When the method detects an error or exception on a statement contained in it, it creates (raises) an exception object that contains information on the type of the exception, and the state of the program when the error occurred, to signal the abnormal condition.
- **Catching an Exception**:
  - When an exception is thrown, the **JVM** looks for an **exception handler** that can **catch** and handle the exception. The exception handler must match the type of the exception thrown.

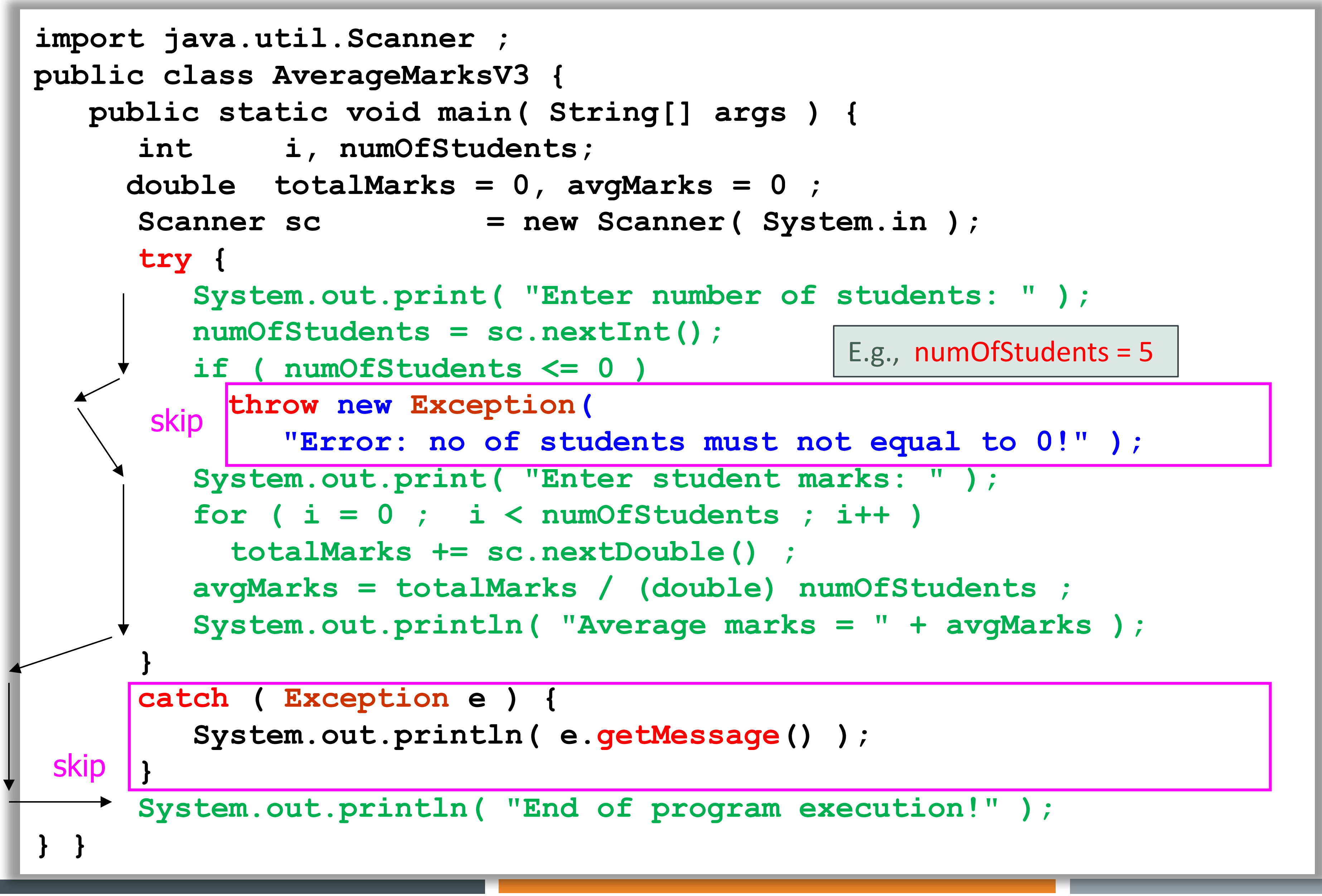
‘try/throw/catch’ Mechanism

```
try {
   // statements for normal flow of program execution
   // at least one statement should be capable of
   // throwing an exception based on some conditions
   if ( /* some conditions happen */ )
       throw new Exception_Name
     ( Optional_String_Arguments );
}
catch ( Exception_Class_Name1 Parameter_Name_1 ) {
   // statements to handle the exception
}
...
catch ( Exception_Class_NameN Parameter_Name_n ) {
   // statements to handle the exception
}
finally {
   // (optional) statements to be executed regardless
   // of whether an exception is thrown or not
}
```

Computing Average Marks Version 3

```
import java.util.Scanner ;
public class AverageMarksV3 {
   public static void main( String[] args ) {
      int      i, numOfStudents;
      double  totalMarks = 0, avgMarks = 0 ;
      Scanner sc         = new Scanner( System.in );
      try {
         System.out.print( "Enter number of students: " );
         numOfStudents = sc.nextInt();
         if ( numOfStudents <= 0 )
            throw new Exception(
               "Error: no of students must not equal to 0!" );
         System.out.print( "Enter student marks: " );
         for ( i = 0 ;  i < numOfStudents ; i++ )
           totalMarks += sc.nextDouble() ;
         avgMarks = totalMarks / (double) numOfStudents ;
         System.out.println( "Average marks = " + avgMarks );
      }
      catch ( Exception e ) {
         System.out.println( e.getMessage() );
      }
      System.out.println( "End of program execution!" );
} }
```

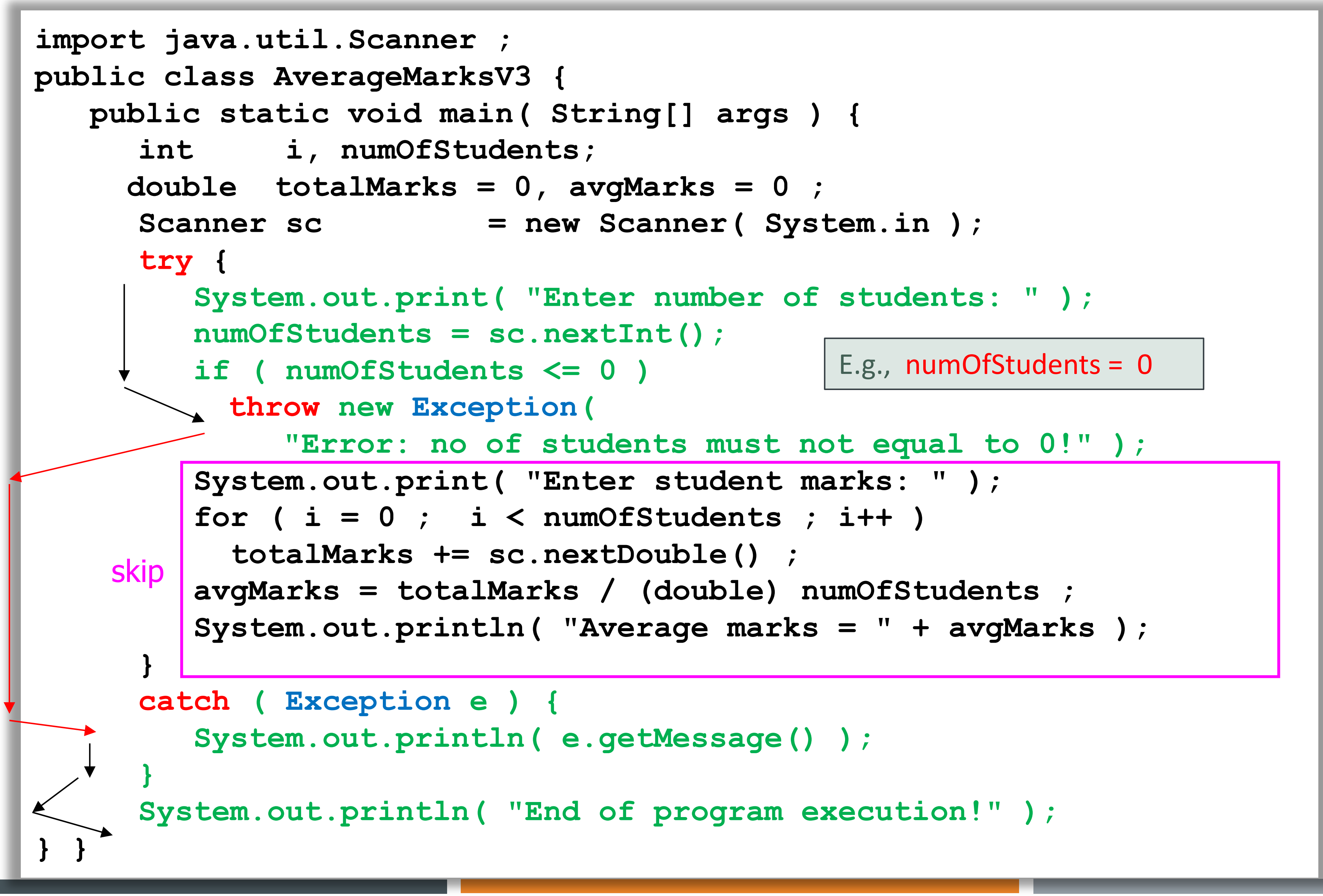
#1: Flow of Control: When No Exception



```
import java.util.Scanner ;
public class AverageMarksV3 {
   public static void main( String[] args ) {
      int     i, numOfStudents;
     double  totalMarks = 0, avgMarks = 0 ;
      Scanner sc         = new Scanner( System.in );
      try {
         System.out.print( "Enter number of students: " );
         numOfStudents = sc.nextInt();
         if ( numOfStudents <= 0 )
           throw new Exception(
              "Error: no of students must not equal to 0!" );
         System.out.print( "Enter student marks: " );
         for ( i = 0 ;  i < numOfStudents ; i++ )
           totalMarks += sc.nextDouble() ;
         avgMarks = totalMarks / (double) numOfStudents ;
         System.out.println( "Average marks = " + avgMarks );
      }
      catch ( Exception e ) {
         System.out.println( e.getMessage() );
      }
      System.out.println( "End of program execution!" );
} }
```

E.g., numOfStudents = 5

#2: Flow of Control: When Exception is Thrown



```
import java.util.Scanner ;
public class AverageMarksV3 {
   public static void main( String[] args ) {
      int      i, numOfStudents;
     double  totalMarks = 0, avgMarks = 0 ;
      Scanner sc          = new Scanner( System.in );
      try {
         System.out.print( "Enter number of students: " );
         numOfStudents = sc.nextInt();
         if ( numOfStudents <= 0 )
           throw new Exception(
              "Error: no of students must not equal to 0!" );
         System.out.print( "Enter student marks: " );
         for ( i = 0 ;  i < numOfStudents ; i++ )
           totalMarks += sc.nextDouble() ;
         avgMarks = totalMarks / (double) numOfStudents ;
         System.out.println( "Average marks = " + avgMarks );
      }
      catch ( Exception e ) {
         System.out.println( e.getMessage() );
      }
      System.out.println( "End of program execution!" );
} }
```

Computing Average Marks Version 3

Computing Average Marks Version 3

Program Input and Output

Case # 1

```
Enter number of students: 5
Enter student marks: 70 80 90 60 50
Average marks = 70.0
End of program execution!
```

Case # 2

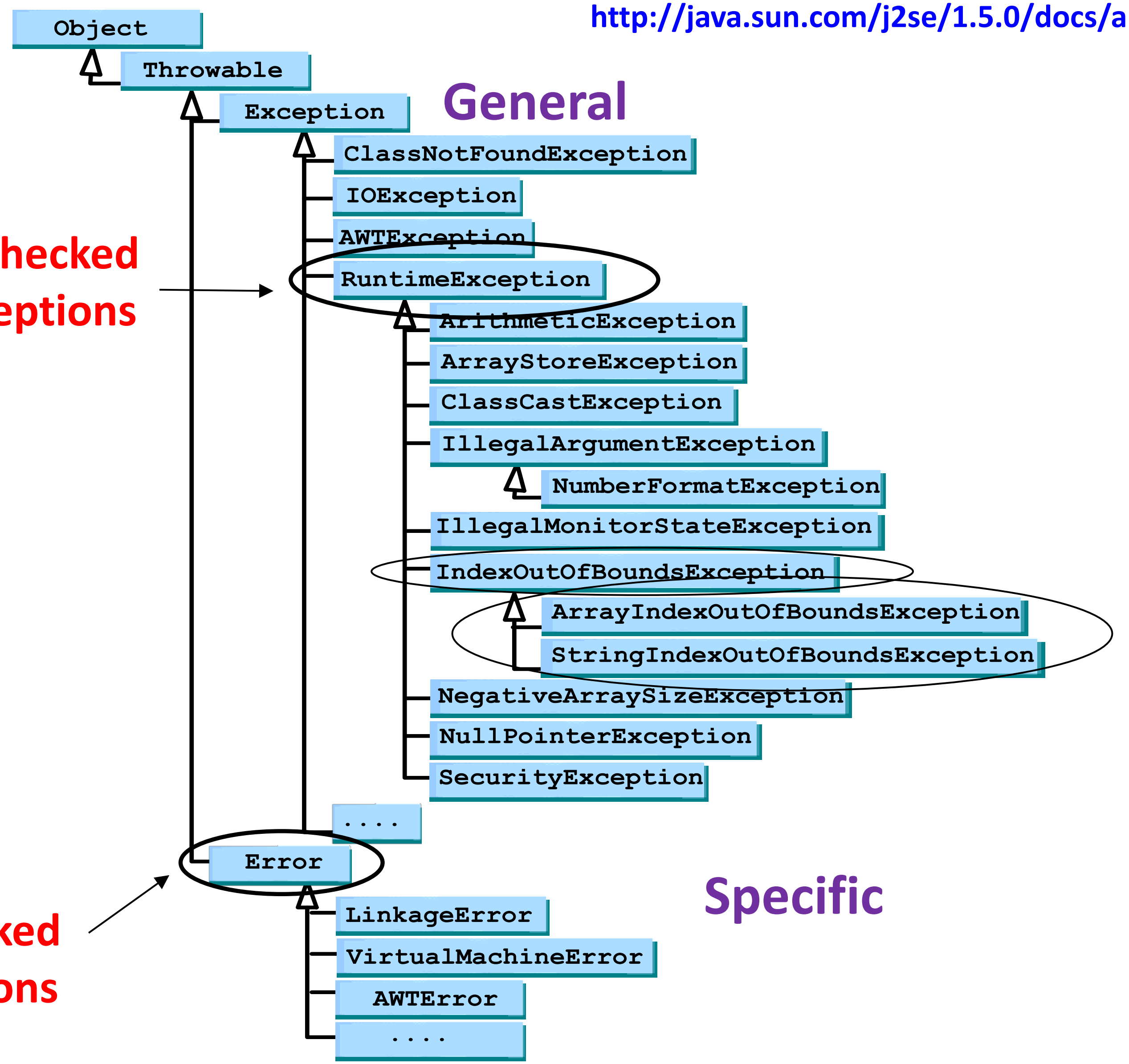
```
Enter number of students: 0
Error: no of students must not equal to 0!
End of program execution!
```

Topic 3: Java’s Exception Hierarchy

Java’s Exception Hierarchy

http://java.sun.com/j2se/1.5.0/docs/api/java/lang/package-tree.html

root/ancestor



- Object
  - Throwable
    - Exception
      - ClassNotFoundException
      - IOException
      - AWTException
      - RuntimeException
        - ArithmeticException
        - ArrayStoreException
        - ClassCastException
        - IllegalArgumentException
          - NumberFormatException
        - IllegalMonitorStateException
        - IndexOutOfBoundsException
          - ArrayIndexOutOfBoundsException
          - StringIndexOutOfBoundsException
        - NegativeArraySizeException
        - NullPointerException
        - SecurityException
      - ....
    - Error
      - LinkageError
      - VirtualMachineError
      - AWTError
      - ....

General

Unchecked exceptions

Specific

Unchecked exceptions

Some Useful Exceptions (System Generate)

Exception (Predefined)	Description
ArithmeticException	This indicates division by zero or some kinds of arithmetic exceptions.
IndexOutOfBoundsException	This indicates that an array or string index is out of bound.
ArrayIndexOutOfBoundsException	This indicates that an **array** index is less than zero or greater than or equal to the array's length.
StringIndexOutOfBoundsException	This indicates that a **string** index is less than zero or greater than or equal to the string's length.
FileNotFoundException	This indicates that the reference to a file cannot be found.
IllegalArgumentException	This indicates that an improper argument is used when calling a method.
NullPointerException	This indicates that an object reference has not been initialised yet.
NumberFormatException	This indicates that illegal num format is used.

Some Useful Exceptions (System Generate)

Exception (Predefined)	Description
ArithmeticException	This indicates division by zero or some kinds of arithmetic exceptions.
IndexOutOfBoundsException	This indicates that an array or string index is out of bound.
ArrayIndexOutOfBoundsException	This indic o or greater than or equal to the array's length.
StringIndexOutOfBoundsException	This indic o or greater t
FileNotFoundException	This indicates that the reference to a file cannot be found.
IllegalArgumentException	This indicates that an improper argument is used when calling a method.
NullPointerException	This indic een initialise
NumberFormatException	This indicates that illegal num format is used.

```
int [] array = new int[5];
int  j = array[5];
```

```
String str = “Hello” ;
 char c =  str.charAt(str.length()) ;
```

```
String str = null;
int j = st.length();
```

```
int  i  =  Integer.parseInt(“abc”);
```

Exception Class

- **Exception** is the root class of all exceptions.
- The **Exception class** has two constructors:

```
public Exception()
public Exception( String message )
```

- The Exception class contains some useful **instance methods** to get information related to the exception:
  - **String getMessage()**: Returns the message of the exception object.
  - **String toString()**: Returns a short description of the exception object.
  - **void printStackTrace()**: Print on screen a **trace** of all the methods that were called, leading up to the method that threw the exception.

The Exception Class

- **Example:**

```
try
{
   // statements that may throw ArithmeticException
}
catch (ArithmeticException e)
{
  // An example exception handling code:
  System.out.println( e.getMessage() );
  e.printStackTrace() ;
  System.exit(0);     // terminate the program
}
```

The Exception Class

- **Example:**

```
try
{
   // stater
}
catch (Arit
{
  // An exa
  System.ou
  e.printSt
  System.ex
}
```

```
public class  CreateException
{
     public static void main(String[] args) {
method1(3, 0)   ;
    }
     public static void method1(int i, int j) {
    method2(i,j);
     }
     public static void method2(int i, int j) {
    method3(i,j);
     }
     public static int method3(int i, int j) {
    int k = 0 ;
          try {
          k = i / j ;
          }catch (ArithmeticException e)   {
    System.out.println( e.getMessage() );
    e.printStackTrace() ;
    System.exit(0);    // terminate the program
          }
          return k ;
      }
}
```

The Exception Class

- **Example:**

```
try
{
   // stater
}
catch
{
  // /
  Sys
  e.p
  Sys
}
```

```
public class  CreateException
{
     public static void main(String[] args) {
method1(3, 0)   ;
    }
     public static void method1(int i, int j) {
    method2(i,j);
```

```
/ by zero
java.lang.ArithmeticException: / by zero
        at CreateException.method3(CreateException.java:36)
        at CreateException.method2(CreateException.java:31)
        at CreateException.method1(CreateException.java:28)
        at CreateException.main(CreateException.java:16)
Press any key to continue . . .
```

```
    System.out.println( e.getMessage() );
    e.printStackTrace() ;
    System.exit(0);   // terminate the program
          }
           return k ;
      }
}
```

Topic 4: Using Exception Classes

Using Exception Class

- There are basically **two types** of exceptions in Java's exception hierarchy:
  1) Checked Exceptions
  2) Unchecked Exceptions

- **Checked Exception**
  - It refers to those exceptions that **can be analysed** by the **compiler**.
  - For example, statements that might cause possible **IOException** when reading a file input data from users.
  - For checked exceptions:
    1. A checked exception **can be caught** within the method that **threw** the exception using the **try/catch** blocks.
    2. However, it is also possible to **delay** the handling of an exception when it is not clear how to handle the exception in the method (using the **throws** clause).

Unchecked Exceptions

- Refers to the exceptions that are **not checked** by the **compiler**.
  - Also refers to exceptions that belong to:
    1) any of the subclasses of class **RuntimeException** or
    2) class **Error**

  - These exceptions can be caused by actions, e.g., pressing a return key without entering any input or entering incorrect data.
    -> These actions might lead to exceptions such as **ArithmeticException** or **IndexOutOfBoundsException**.

  - These exceptions are usually **not easy to be checked explicitly** and are always avoided by programming.

Unchecked Exceptions (Cont`d)

- If **unchecked exceptions** are **not handled** inside the program (left uncaught), they will be handled by **Java's default exception handlers**. The program will terminate with an error message (with the name of the exception class).

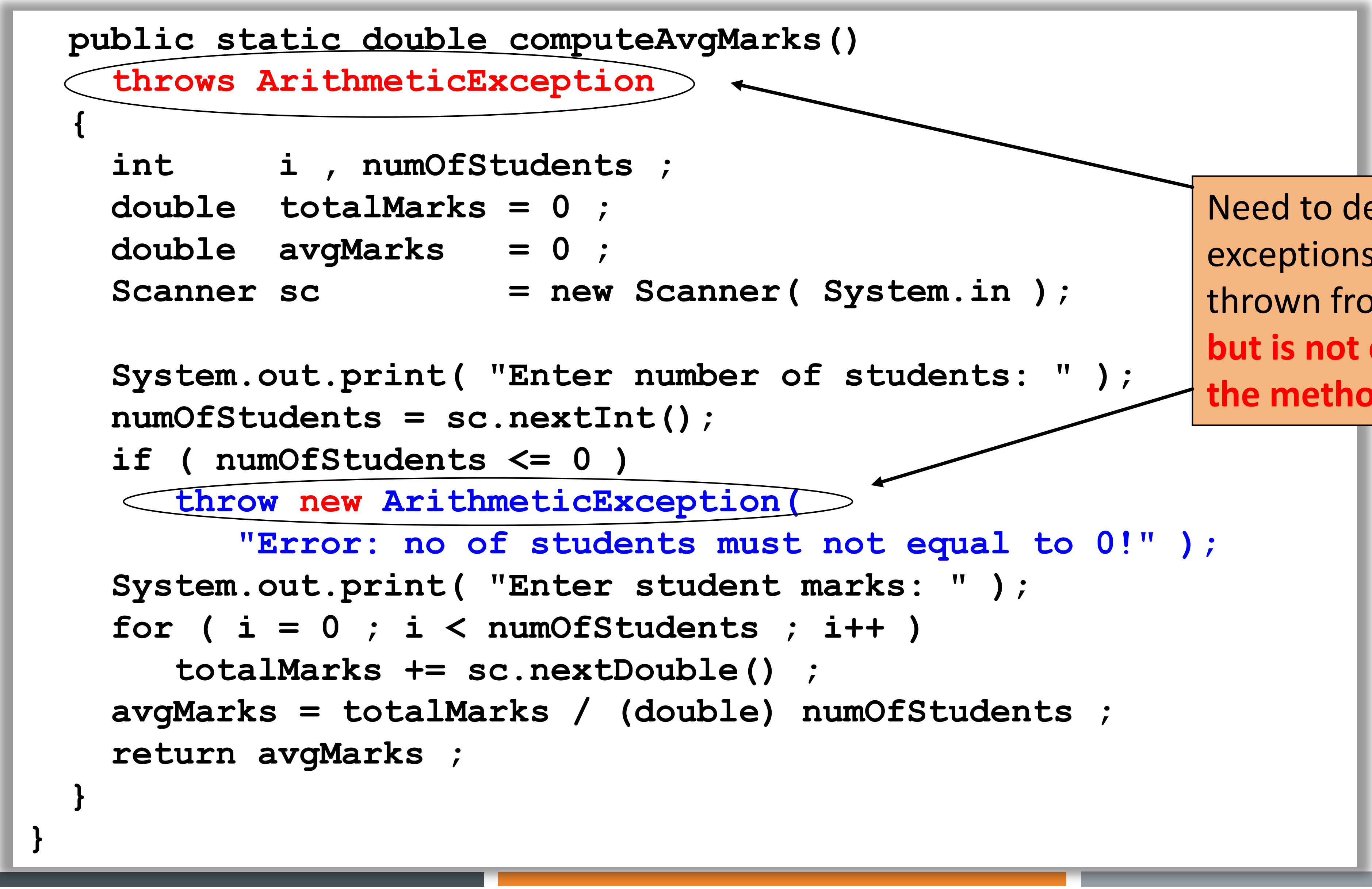
- **Guidelines for checked and unchecked exceptions**:
  - If the exception comes from a pre-defined class **RuntimeException** or the class **Error**, it does not necessary need to be caught within the programs (UNCHECKED EXCEPTIONS).
  - Otherwise, the exception must be either caught within the method in which it is thrown using a **catch block**, or declared that the exception might be thrown in the method using a **throws** clause (CHECKED EXCEPTIONS).

Computing Average Marks Version 4

```
import java.util.Scanner ;
public class AverageMarksV4 {
  public static void main( String[] args ){
     double average ;
     try {
       average = computeAvgMarks();
       System.out.println( "Average marks = " + average );
     }
     catch ( ArithmeticException e ) {
       System.out.println( e.getMessage() );
     }
     finally {
       System.out.println( "End of program execution!" );
     }
   }
   // to continue in next page
```

Calling this method must be prepared to handle any exceptions that may be thrown

Throwing Exceptions in Methods



```
  public static double computeAvgMarks()
    throws ArithmeticException
  {
    int      i , numOfStudents ;
    double   totalMarks = 0 ;
    double   avgMarks   = 0 ;
    Scanner sc          = new Scanner( System.in );

    System.out.print( "Enter number of students: " );
    numOfStudents = sc.nextInt();
    if ( numOfStudents <= 0 )
       throw new ArithmeticException(
          "Error: no of students must not equal to 0!" );
    System.out.print( "Enter student marks: " );
    for ( i = 0 ; i < numOfStudents ; i++ )
       totalMarks += sc.nextDouble() ;
    avgMarks = totalMarks / (double) numOfStudents ;
    return avgMarks ;
  }
}
```

Need to declare any exceptions that might be thrown from this method **but is not caught in the method**

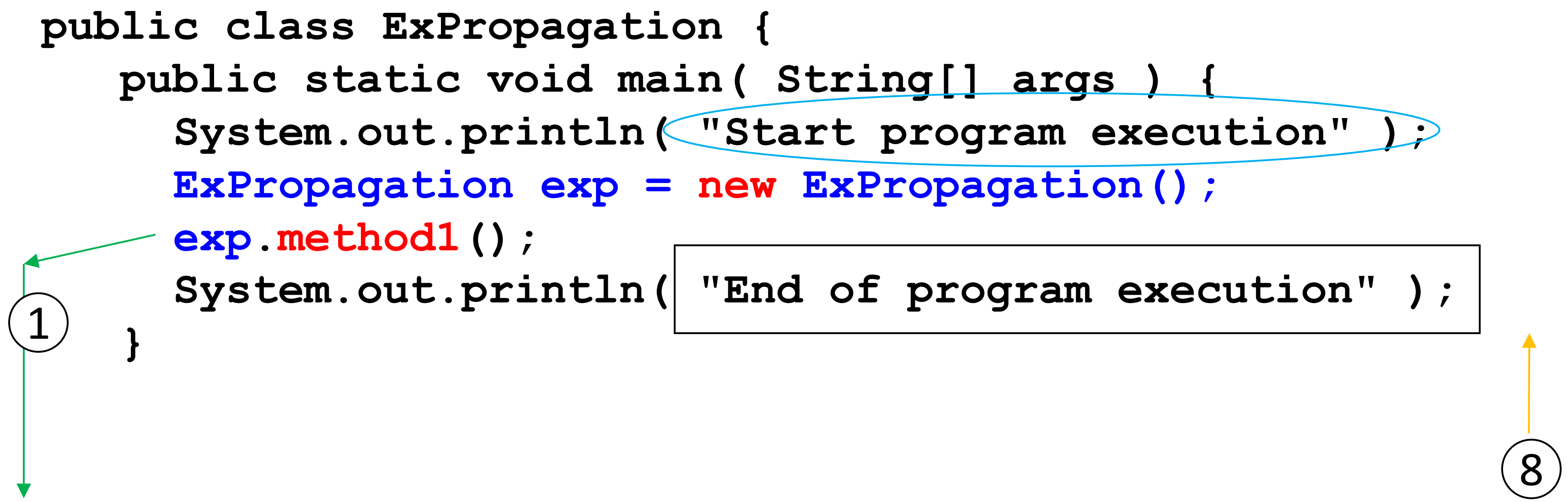
Throwing Exceptions in Methods

```
Program Input and Output
Enter number of students: 5
Enter student marks: 70 80 90 60 50
Average marks = 70
End of program execution!

Enter the number of students: 0
Error: no of students must not equal to 0!
End of program execution!
```

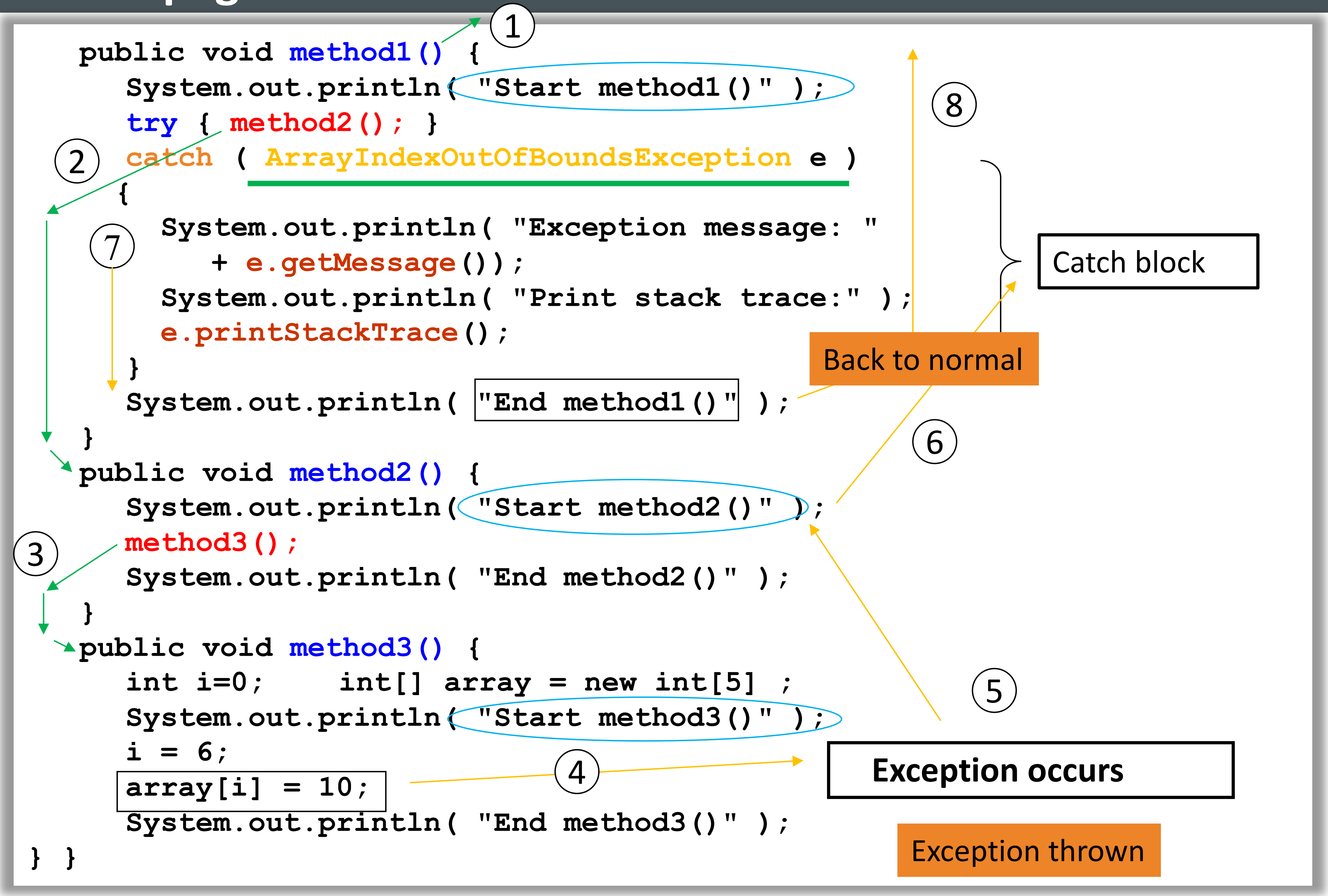
Exception Propagation

- **Exception propagation** - If an exception is thrown and not caught by the handlers after the try block where it occurs, control is then **transferred to the method** that invoked the method that threw the exception.



```
public class ExPropagation {
   public static void main( String[] args ) {
     System.out.println( "Start program execution" );
     ExPropagation exp = new ExPropagation();
     exp.method1();
     System.out.println( "End of program execution" );
   }
```

Exception Propagation



```
    public void method1() {
       System.out.println( "Start method1()" );
       try { method2(); }
       catch ( ArrayIndexOutOfBoundsException e )
       {
          System.out.println( "Exception message: "
             + e.getMessage());
          System.out.println( "Print stack trace:" );
          e.printStackTrace();
       }
       System.out.println( "End method1()" );
    }
    public void method2() {
       System.out.println( "Start method2()" );
       method3();
       System.out.println( "End method2()" );
    }
    public void method3() {
       int i=0;    int[] array = new int[5] ;
       System.out.println( "Start method3()" );
       i = 6;
       array[i] = 10;
       System.out.println( "End method3()" );
} }
```

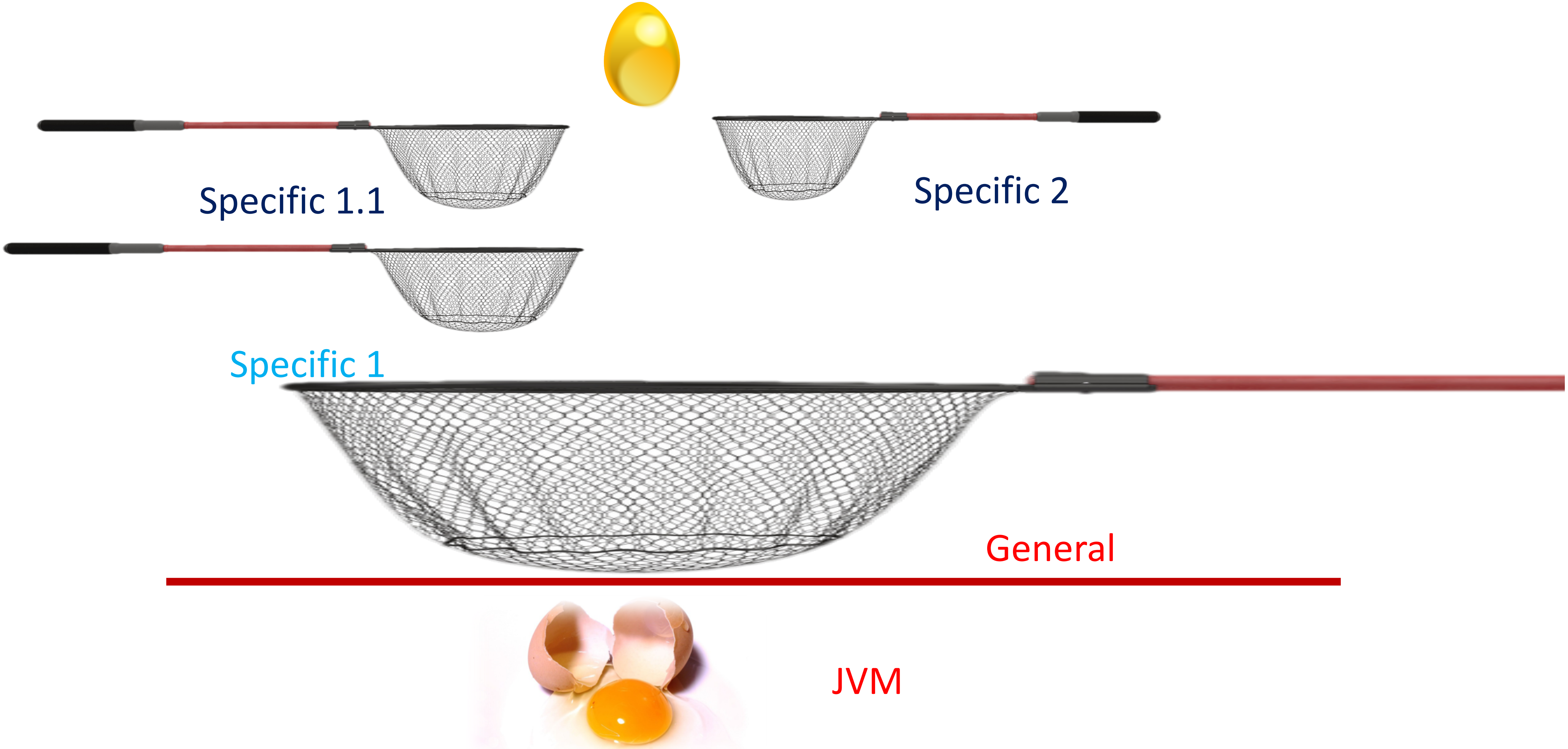
Exception Propagation

```
Program Output
Start program execution
Start method1()
Start method2()
Start method3()
Exception message: 6
Print stack trace:
Java.lang.ArrayIndexOutOfBoundsException: 6
at ExPropagation.method3(ExPropagation.java:31)
at ExPropagation.method2(ExPropagation.java:22)
at ExPropagation.method1(ExPropagation.java:11)
at ExPropagation.main(ExPropagation.java:5)
End method1()
End of program execution
```

Multiple Throws and Catches

- Methods can throw **more than one exception**.
- More than one **catch** block (handler) can be made available for catching exceptions.
- The **catch** blocks immediately following the try block are searched **in sequence** for matching the exception type being thrown previously.
  - Only the **first catch** block that matches the exception type will be executed.
- Specific exceptions are derived from classes of more general types (see **class inheritance hierarchy**).
  - Thus, specific exceptions can be caught by both general and specific exception types (IS-A relationship).
- **Recommendation**: First put the catch blocks for the more **specific**, then the derived exceptions, and then the more **general** ones near the end.

Multiple Throws and Catches



Multiple Throws and Catches

```
import java.util.Scanner ;
public class AverageMarksV5 {
   public static void main( String[] args ) {
     double average ;
     try {
        average = computeAvgMarks() ;
        System.out.println( "Average marks = " + average );
     }
     catch ( ArithmeticException e ) {
        System.out.println( e.getMessage() );
        System.exit(0);
     }
     catch ( Exception e ) {
        System.out.println( e.getMessage() );
     }
     finally {  // optional
        System.out.println( "End of program execution!" );
     }
   }
```

More specific one

More general

Multiple Throws and Catches

```
  public static double computeAvgMarks()
     throws ArithmeticException
  {
    int      i , numOfStudents ;
    double   totalMarks = 0 ;
    double   avgMarks   = 0 ;
    Scanner sc         = new Scanner( System.in );
    System.out.print( "Enter number of students: " );
    numOfStudents = sc.nextInt();
    if ( numOfStudents <= 0 )
      throw new ArithmeticException(
        "Error: no of students must not equal to 0!" );
    System.out.print( "Enter student marks: " );
    for ( i = 0 ; i < numOfStudents ; i++ )
      totalMarks += sc.nextDouble() ;
    avgMarks = totalMarks / (double) numOfStudents ;
    return avgMarks ;
  }
}
```

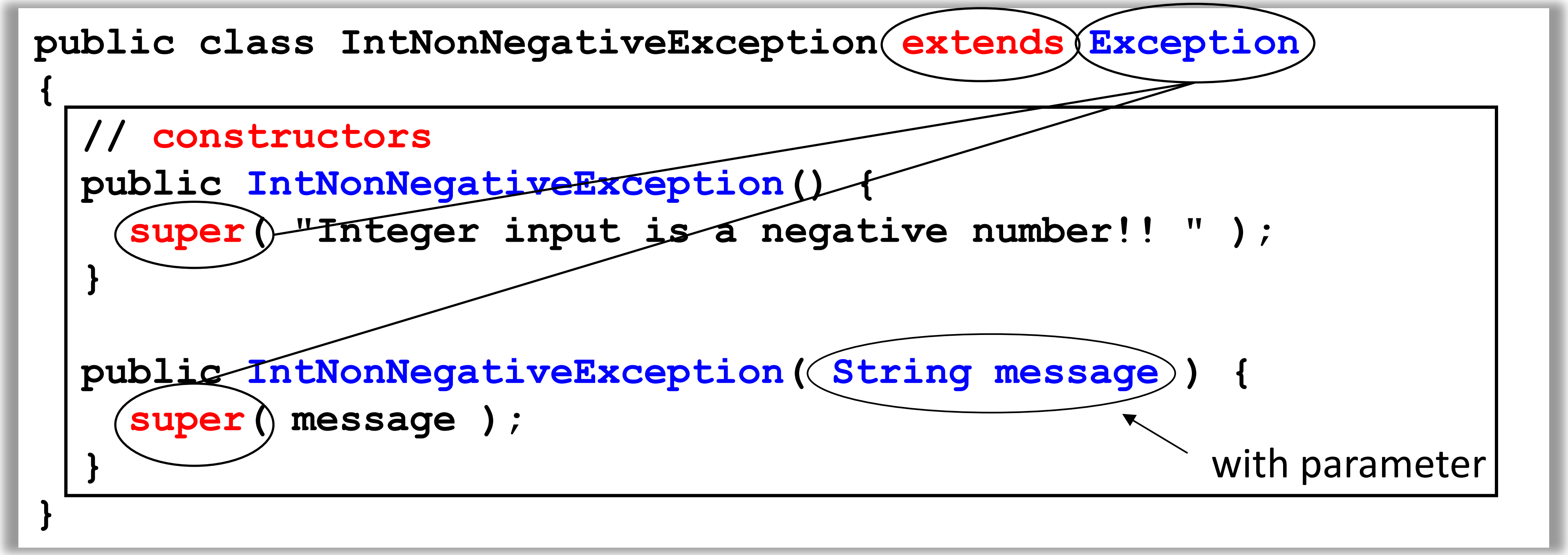
Multiple Throws and Catches

```
Program Output
Enter number of students: 5
Enter student marks: 70 80 90 60 50
Average marks = 70
End of program execution!

Enter number of students: 0
Error: no of students must not equal to 0!
```

Topic 5: Creating Your Own Exception Classes

Creating Your Own Exception Classes



```
public class IntNonNegativeException extends Exception
{
  // constructors
  public IntNonNegativeException() {
    super( "Integer input is a negative number!! " );
  }

  public IntNonNegativeException( String message ) {
    super( message );
  }
}
```

with parameter

- You can define **<u>your own exception class</u>** and use it in your throw statement (in your own code).
- Must be derived (inherited) from an existing **exception** class.
- Usually, we define **<u>only</u>** the **constructors** and call the default constructor using **super**.

Example

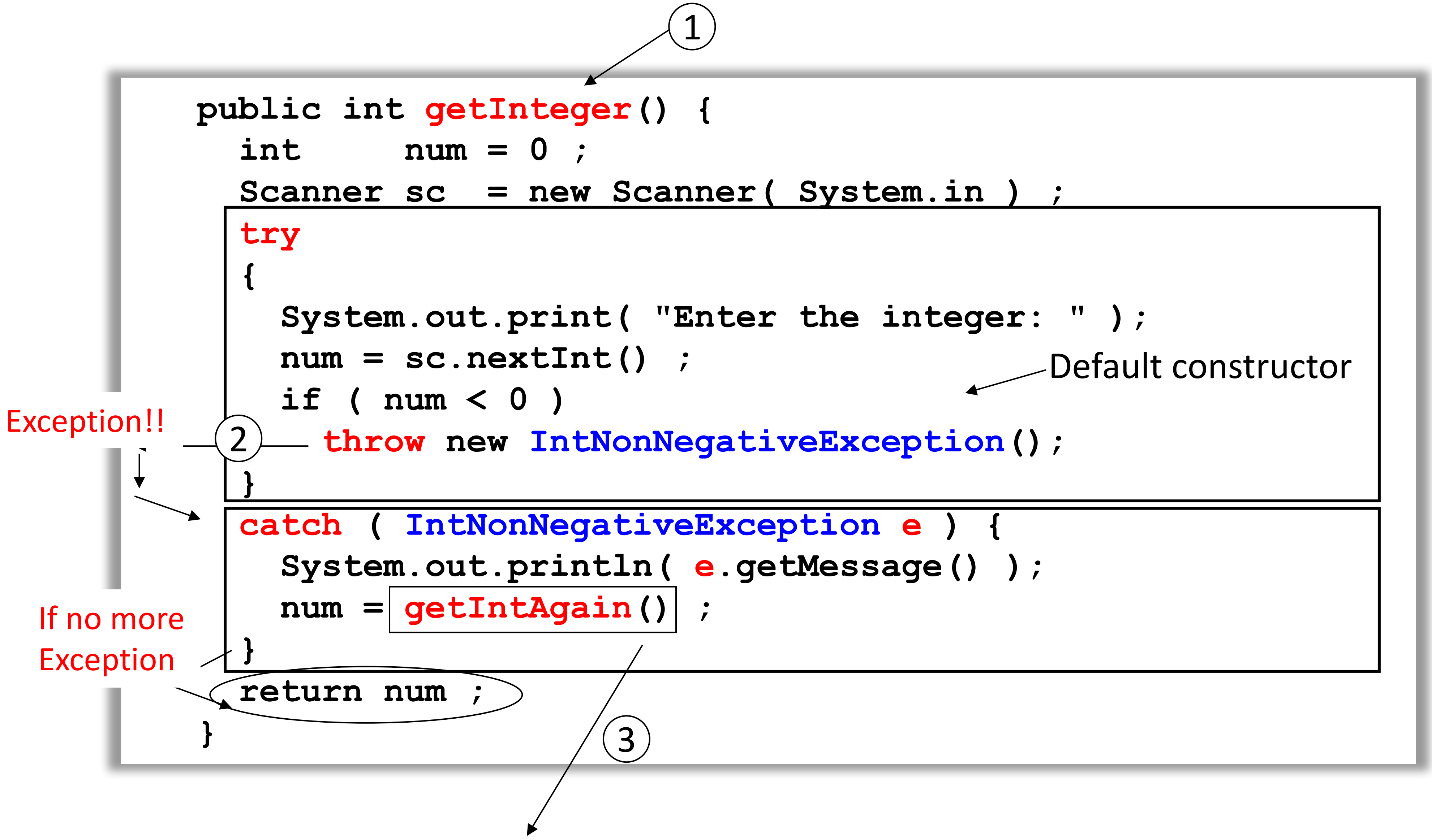
```
import java.util.Scanner ;
public class IntNonNegativeExceptionApp
{
  public static void main( String[] args ) {
    IntNonNegativeExceptionApp sumEx
                = new IntNonNegativeExceptionApp() ;
    int     inputNum ;
    int     sum = 0 ;
    Scanner sc  = new Scanner( System.in );

    System.out.print( "Enter total no. of integers: " );
    int total = sc.nextInt();

    for ( int i = 0 ; i < total ; i++ ) {
      inputNum = sumEx.getInteger();
      sum += inputNum ;
    }
    System.out.println( "The sum of integers: " + sum );
  }
```

①

Example



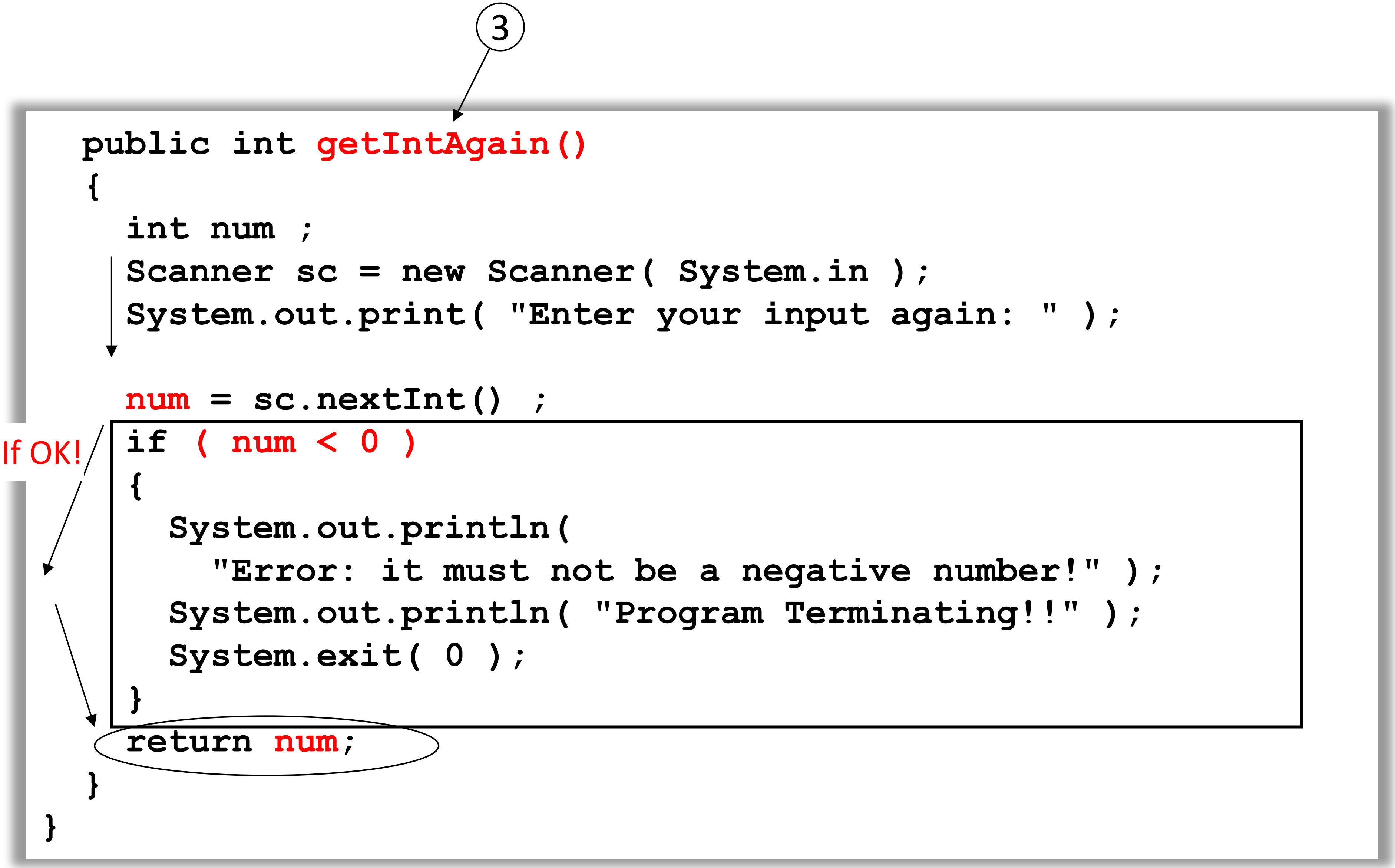
```
public int getInteger() {
  int      num = 0 ;
  Scanner sc  = new Scanner( System.in ) ;
  try
  {
    System.out.print( "Enter the integer: " );
    num = sc.nextInt() ;
    if ( num < 0 )
      throw new IntNonNegativeException();
  }
  catch ( IntNonNegativeException e ) {
    System.out.println( e.getMessage() );
    num = getIntAgain() ;
  }
  return num ;
}
```

Default constructor

Exception!!

If no more Exception

Example



```
  public int getIntAgain()
  {
    int num ;
    Scanner sc = new Scanner( System.in );
    System.out.print( "Enter your input again: " );

    num = sc.nextInt() ;
    if ( num < 0 )
    {
      System.out.println(
        "Error: it must not be a negative number!" );
      System.out.println( "Program Terminating!!" );
      System.exit( 0 );
    }
    return num;
  }
}
```

Example

```
Program Output
Enter total no. of input integers: 5
Enter the integer: 1
Enter the integer: 2
Enter the integer: 3
Enter the integer: 4
Enter the integer: 5
The sum of integers: 15

Enter total no. of input integers: 5
Enter the integer: 1
Enter the integer: 2
Enter the integer: 3
Enter the integer: -2
Integer input is a negative number!!
Enter your input again: 4
Enter the integer: 5
The sum of integers: 15
```

Exception occurs

Example

```
Program Output

Enter total no. of input integers: 4
Enter the integer: 1
Enter the integer: -2
Integer input is a negative number!!
Enter your input again: -4
Error: it must not be a negative number!
Program Terminating!!
```

Summary

Key points from this chapter:

- Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

- The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application and that is why we use exception handling.

- Suppose there are 10 statements in your program and an exception occurs at statement 5, rest of the code will not be executed, i.e. statement 6 to 10 will not run. However, if we perform exception handling, rest of the statement will be executed.