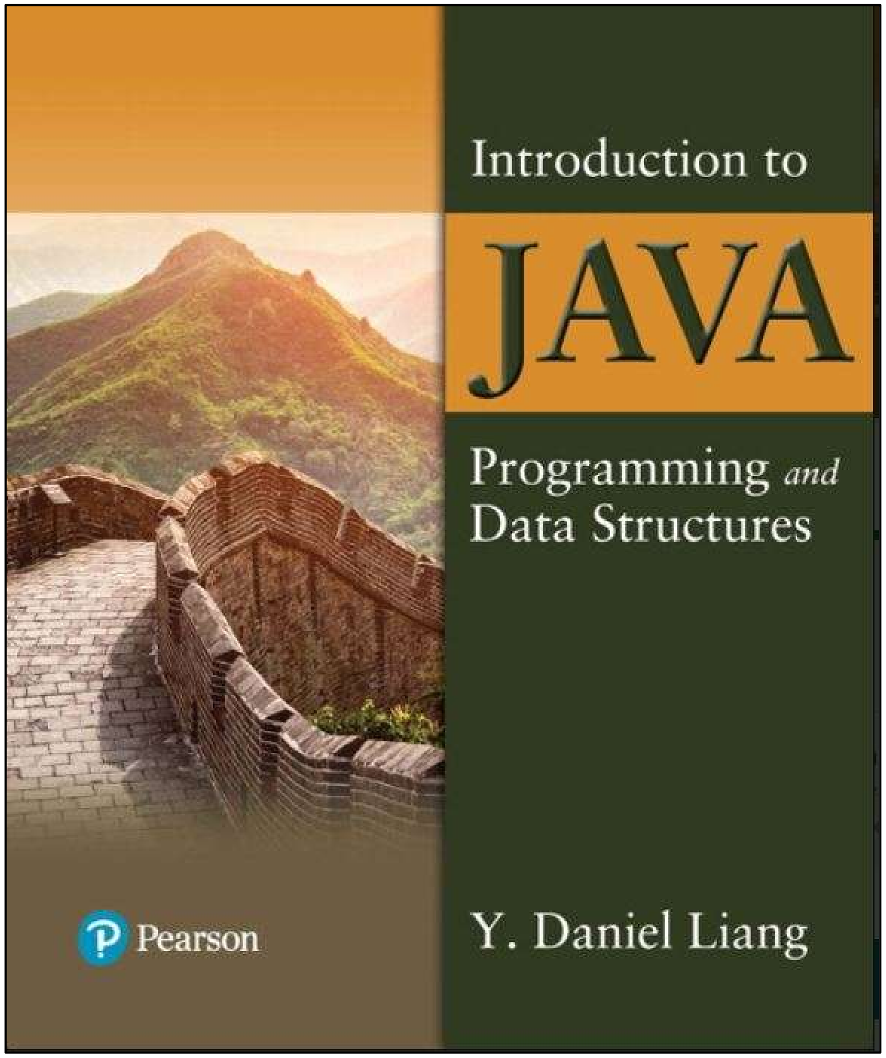
Introduction to Java Programming and Data Structures

Twelfth Edition



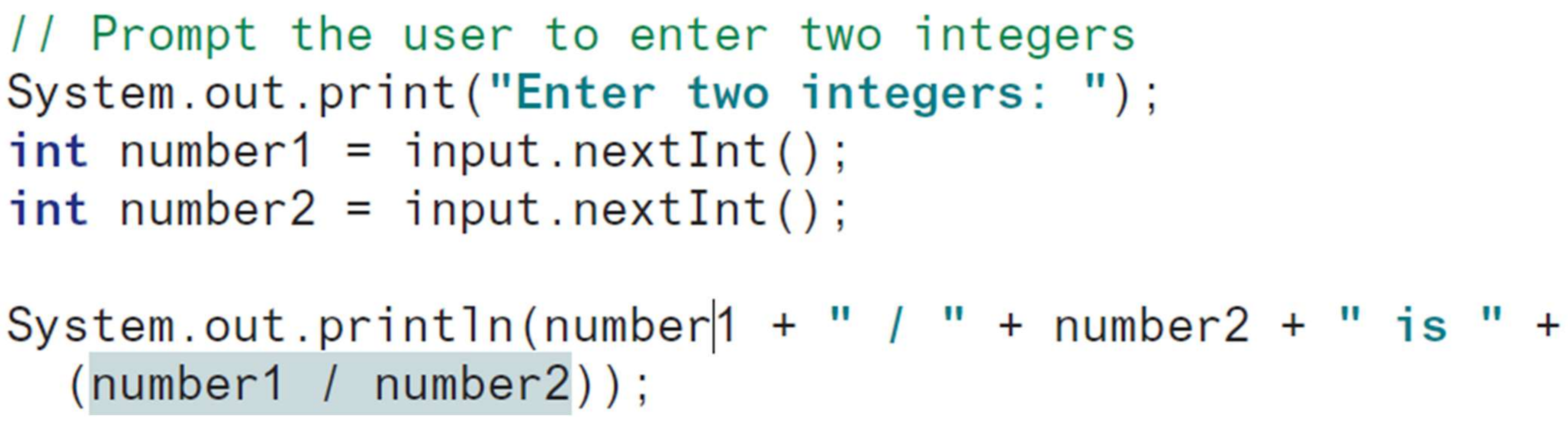
Chapter 12

Exception Handling and Text IO

Motivations

- You write a *small* method for a *big* Java program.

  If there is an (Unexpected?) error in your method, what will happen?



```java
// Prompt the user to enter two integers
System.out.print("Enter two integers: ");
int number1 = input.nextInt();
int number2 = input.nextInt();

System.out.println(number1 + " / " + number2 + " is " +
  (number1 / number2));
```

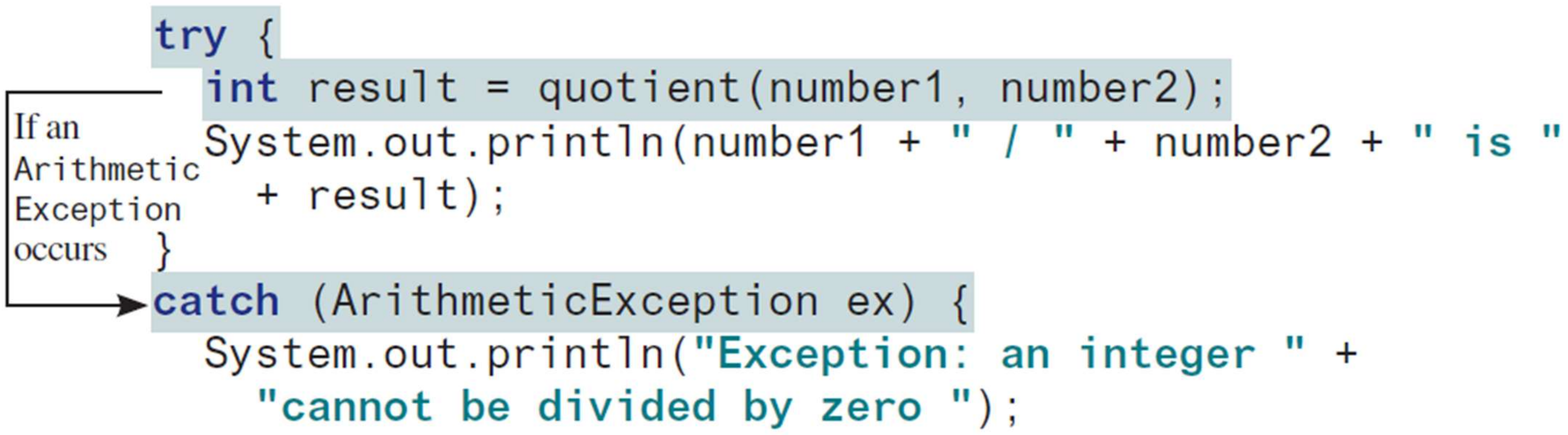
Motivations

When a program runs into a runtime error, the program terminates abnormally.

How can you handle the runtime error so that the program can continue to run or terminate gracefully?

This is the subject we will introduce in this chapter.

Exception-Handling



```
try {
  int result = quotient(number1, number2);
  System.out.println(number1 + " / " + number2 + " is "
    + result);
}
catch (ArithmeticException ex) {
  System.out.println("Exception: an integer " +
    "cannot be divided by zero ");
```

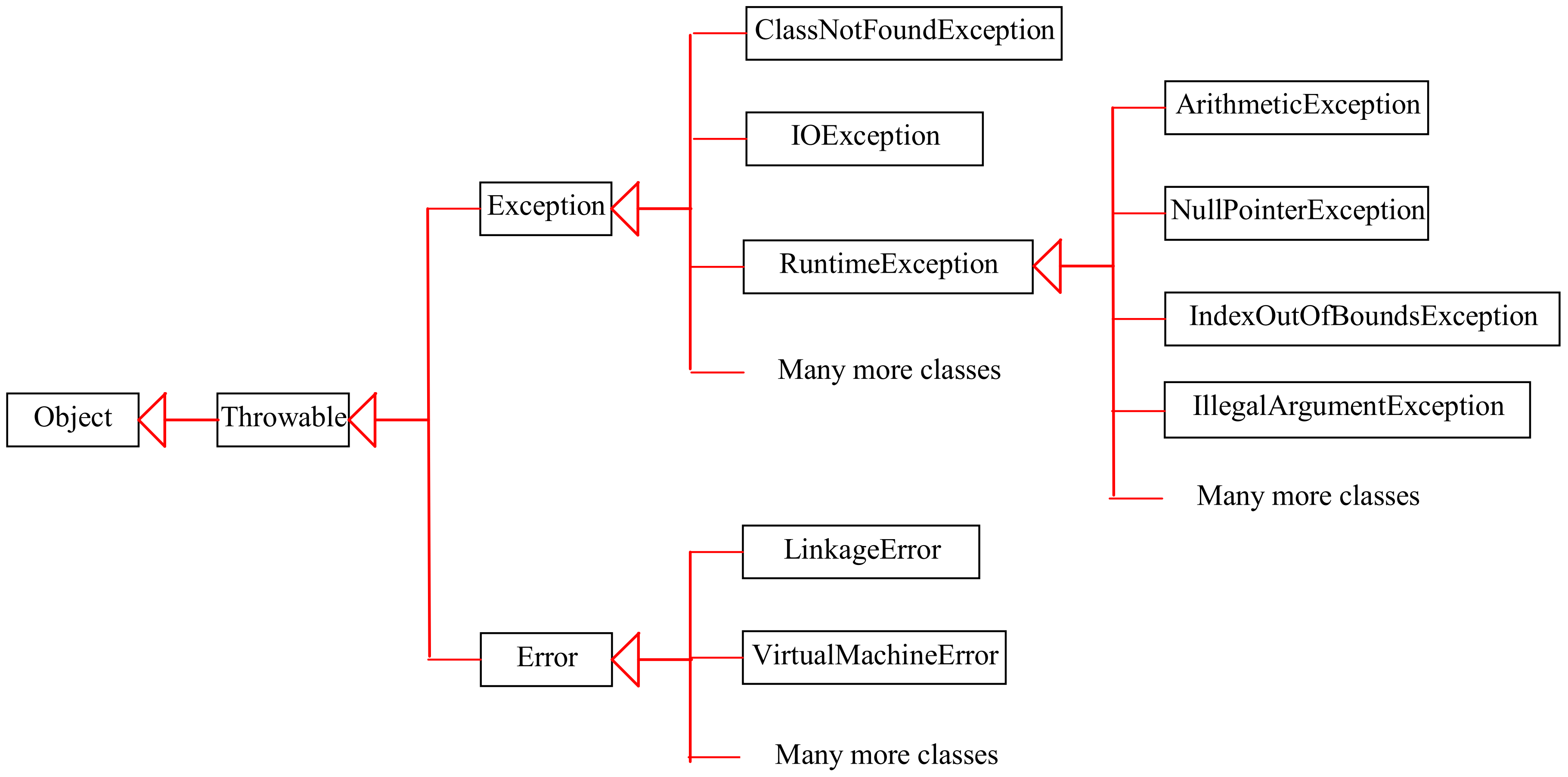
- The **try** block contains the code that is executed in normal circumstances.
- The exception is caught by the **catch** block. The code in the **catch** block is executed to *handle the exception*.

Exception Advantages

QuotientWithException

Now you see the **advantages** of using exception handling. It enables a method to throw an exception to its caller. Without this capability, a method must handle the exception or terminate the program.

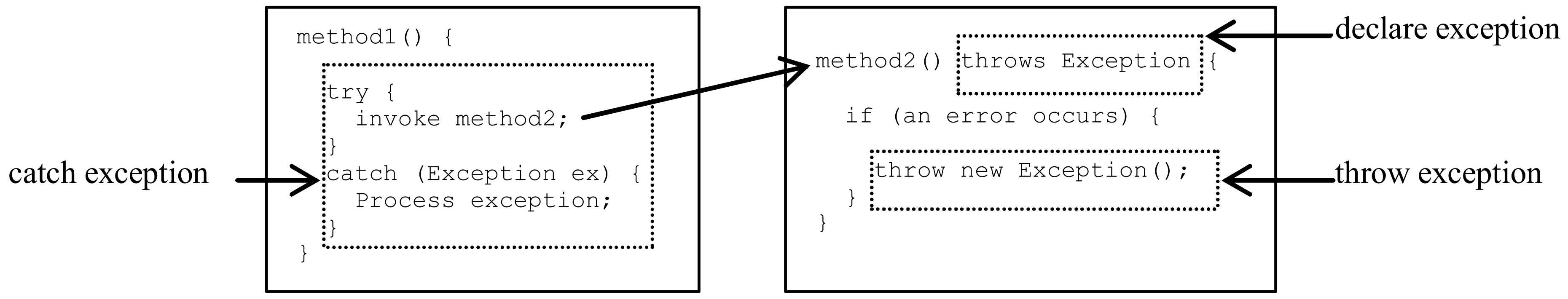
Exception Types



~~Checked Exceptions vs Unchecked Exceptions~~

RuntimeException, Error and their subclasses are known as **unchecked exceptions**. All other exceptions are known as **checked exceptions**, meaning that the compiler forces the programmer to check and deal with the exceptions.

Declaring, Throwing, and Catching Exceptions



```
method1() {
  try {
    invoke method2;
  }
  catch (Exception ex) {
    Process exception;
  }
}
```

```
method2() throws Exception {
  if (an error occurs) {
    throw new Exception();
  }
}
```

Declaring Exceptions

Every method must state the types of checked exceptions it might throw. This is known as **declaring exceptions**.

```
public void myMethod()
  throws IOException

public void myMethod()
  throws IOException, OtherException
```

Throwing Exceptions

When the program detects an error, the program can create an instance of an appropriate exception type and throw it. This is known as **throwing an exception**. Here is an example,

```
throw new TheException();
```

```
TheException ex = new TheException();
throw ex;
```

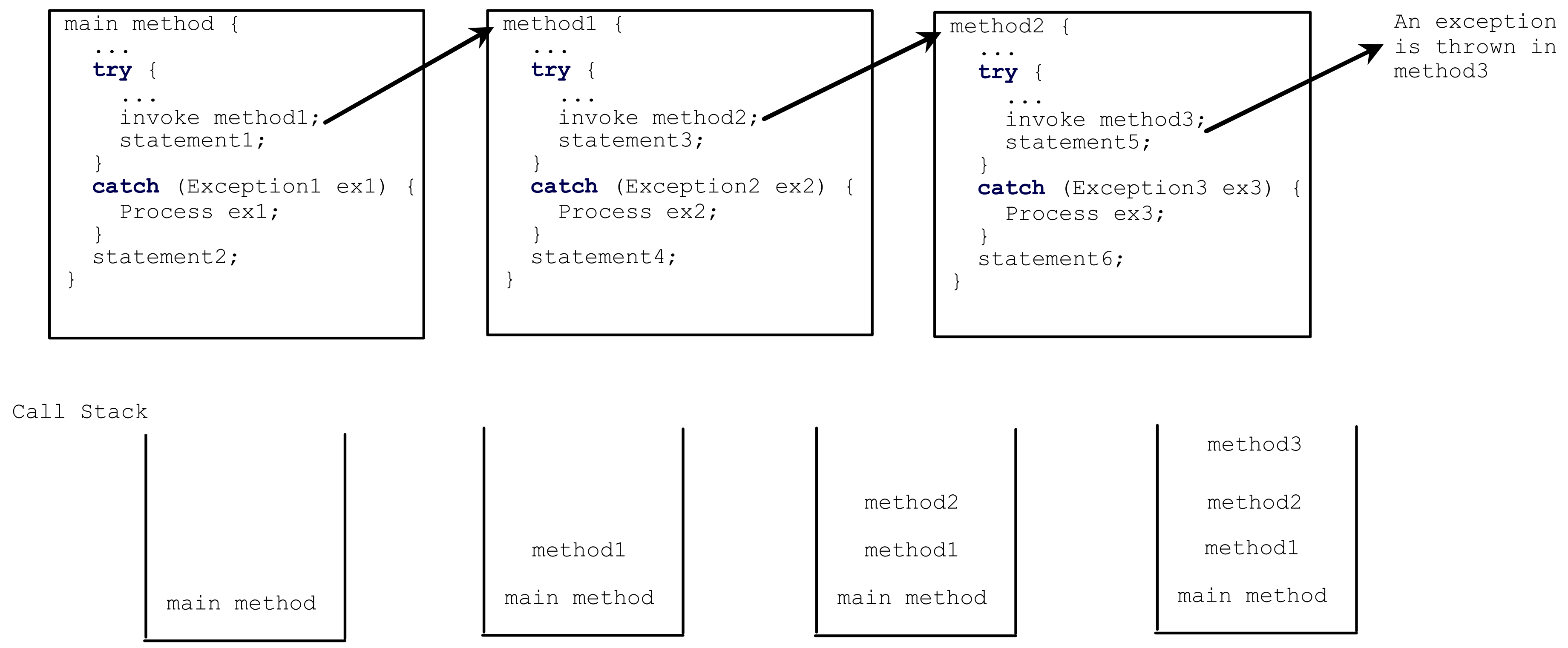
Throwing Exceptions Example

```
  /** Set a new radius */
public void setRadius(double newRadius)
    throws IllegalArgumentException {
  if (newRadius >= 0)
    radius = newRadius;
  else
    throw new IllegalArgumentException(
      "Radius cannot be negative");
}
```

Catching Exceptions (1 of 2)

```
try {
 statements; // Statements that may throw exceptions
}
catch (Exception1 exVar1) {
 handler for exception1;
}
catch (Exception2 exVar2) {
 handler for exception2;
}
...
catch (ExceptionN exVar3) {
 handler for exceptionN;
}
```

Catching Exceptions (2 of 2)



```
main method {
  ...
  try {
    ...
    invoke method1;
    statement1;
  }
  catch (Exception1 ex1) {
    Process ex1;
  }
  statement2;
}
```

```
method1 {
  ...
  try {
    ...
    invoke method2;
    statement3;
  }
  catch (Exception2 ex2) {
    Process ex2;
  }
  statement4;
}
```

```
method2 {
  ...
  try {
    ...
    invoke method3;
    statement5;
  }
  catch (Exception3 ex3) {
    Process ex3;
  }
  statement6;
}
```

The `finally` Clause

```
try {
  statements;
}
catch(TheException ex){
  handling ex;
}
finally {
  finalStatements;
}
```

Trace a Program Execution (1 of 11)

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Suppose no exceptions in the statements

Trace a Program Execution (2 of 11)

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The final block is always executed

Trace a Program Execution (3 of 11)

```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Next statement in the method is executed

Trace a Program Execution (4 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Suppose an exception of type Exception1 is thrown in statement2

Trace a Program Execution (5 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The exception is handled.

Trace a Program Execution (6 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

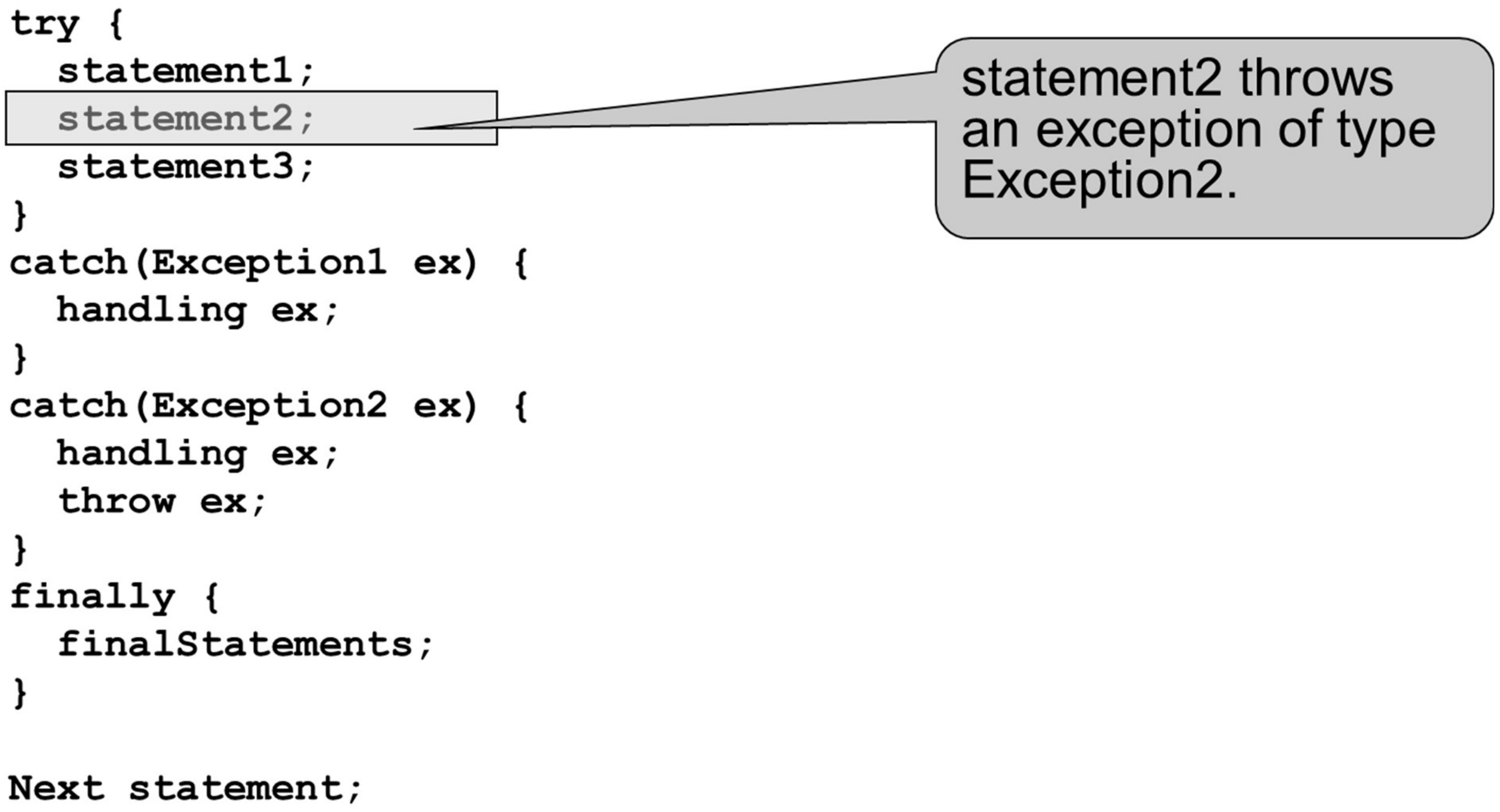
The final block is always executed.

Trace a Program Execution (7 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

The next statement in the method is now executed.

Trace a Program Execution (8 of 11)



```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

statement2 throws an exception of type Exception2.

Trace a Program Execution (9 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Handling exception

Trace a Program Execution (10 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Execute the final block

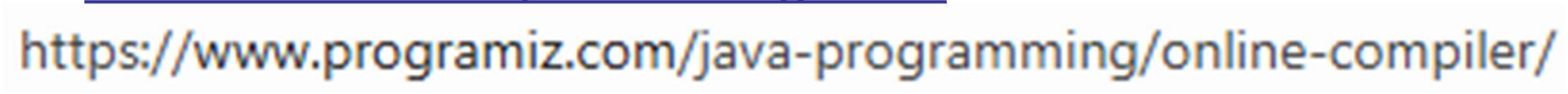
Trace a Program Execution (11 of 11)

```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

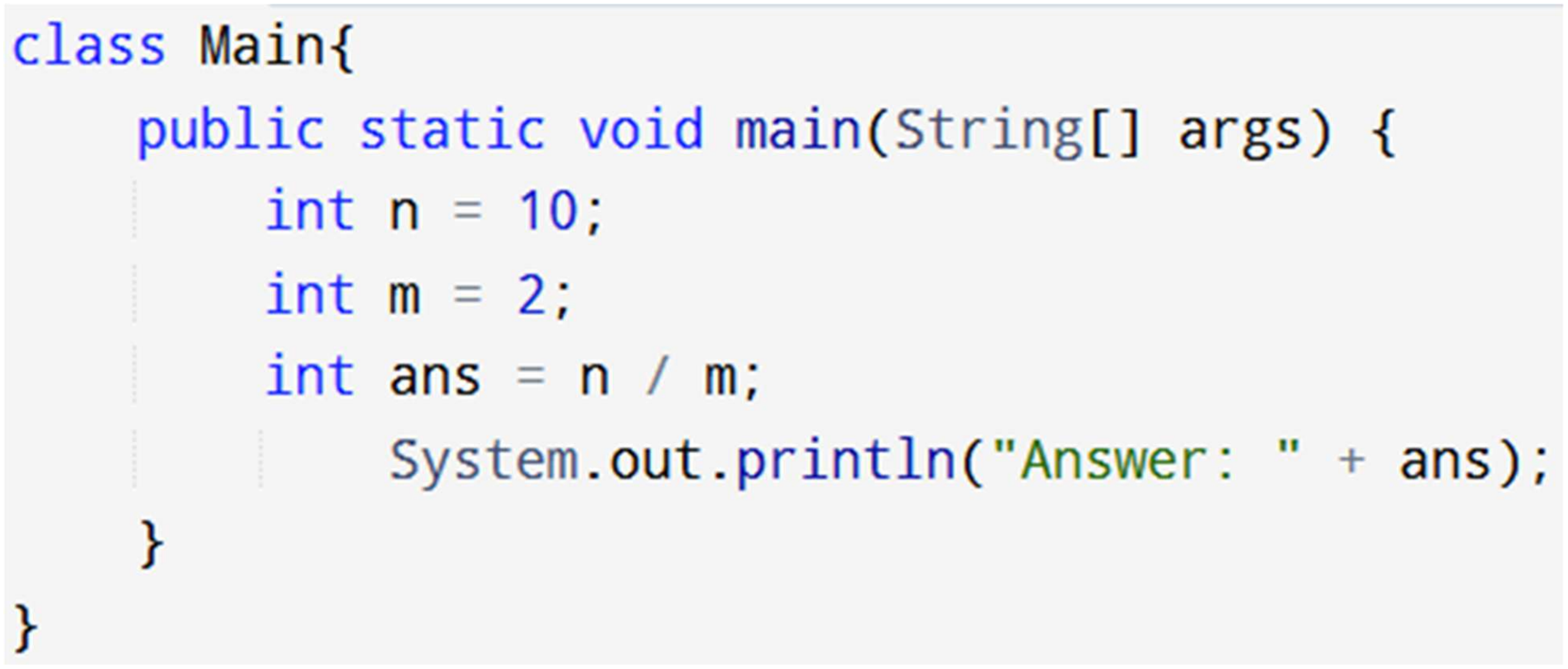
Rethrow the exception and control is transferred to the caller

Practice

- [Online Java Compiler – Programiz]


https://www.programiz.com/java-programming/online-compiler/

- Try this program.

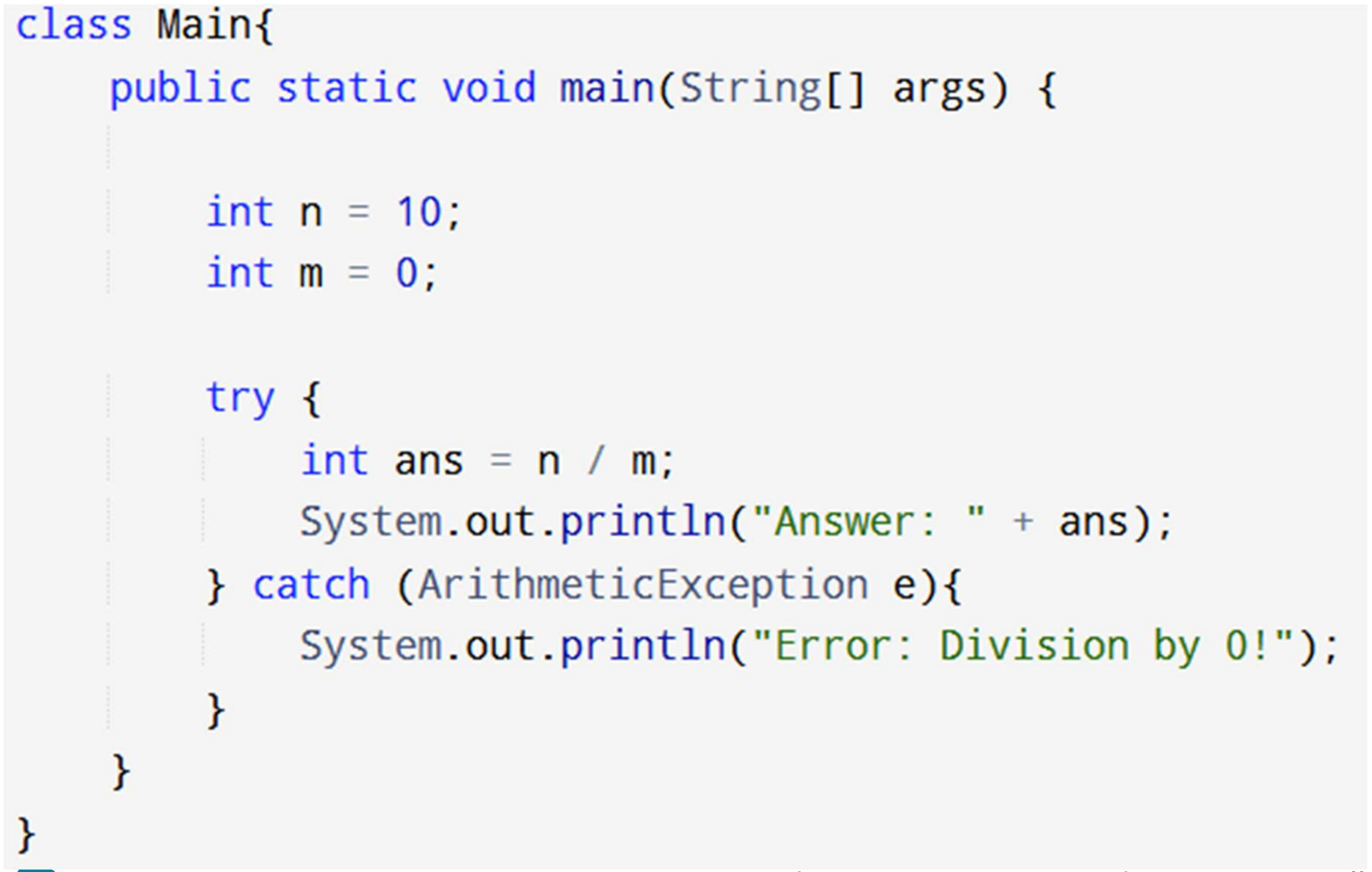


```java
class Main{
    public static void main(String[] args) {
        int n = 10;
        int m = 2;
        int ans = n / m;
            System.out.println("Answer: " + ans);
    }
}
```

- Change m from 2 to 0, run it again.

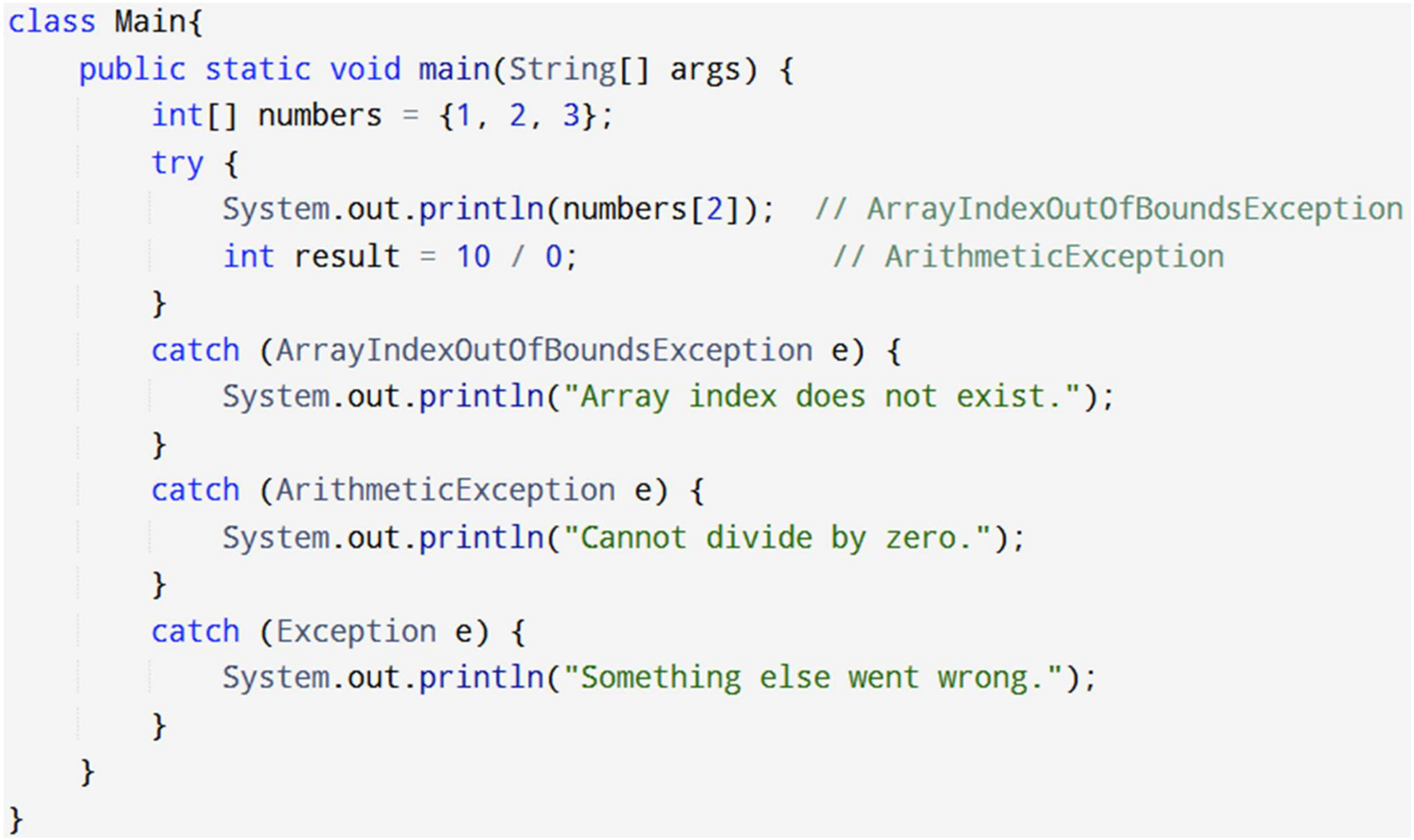
- If your program is a small/tiny part of a big program…

- Change to this:



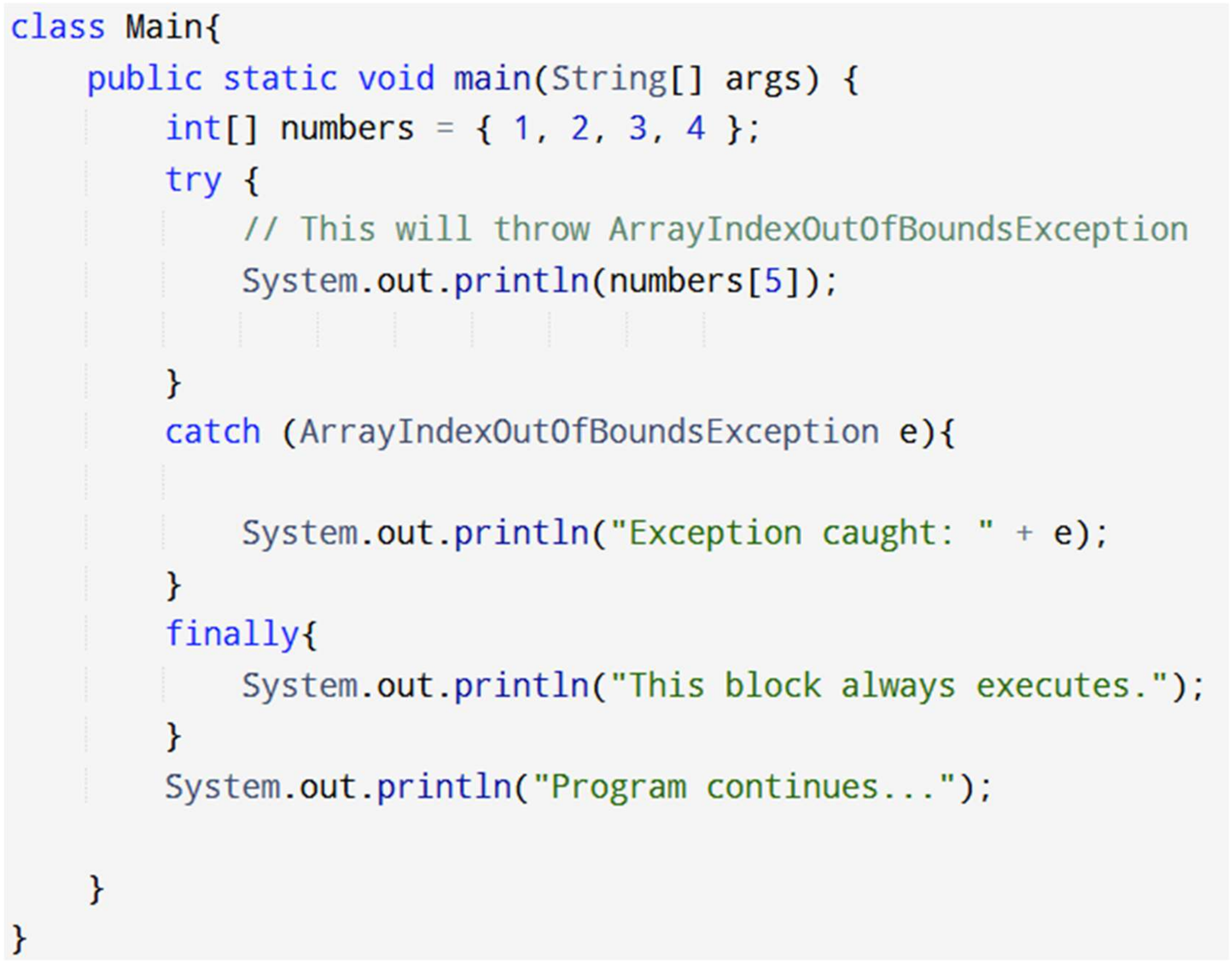
```java
class Main{
    public static void main(String[] args) {

        int n = 10;
        int m = 0;

        try {
            int ans = n / m;
            System.out.println("Answer: " + ans);
        } catch (ArithmeticException e){
            System.out.println("Error: Division by 0!");
        }
    }
}
```

Multiple catch



```java
class Main{
    public static void main(String[] args) {
        int[] numbers = {1, 2, 3};
        try {
            System.out.println(numbers[2]);  // ArrayIndexOutOfBoundsException
            int result = 10 / 0;              // ArithmeticException
        }
        catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Array index does not exist.");
        }
        catch (ArithmeticException e) {
            System.out.println("Cannot divide by zero.");
        }
        catch (Exception e) {
            System.out.println("Something else went wrong.");
        }
    }
}
```

Practice try/catch/finally



```java
class Main{
    public static void main(String[] args) {
        int[] numbers = { 1, 2, 3, 4 };
        try {
            // This will throw ArrayIndexOutOfBoundsException
            System.out.println(numbers[5]);

        }
        catch (ArrayIndexOutOfBoundsException e){

            System.out.println("Exception caught: " + e);
        }
        finally{
            System.out.println("This block always executes.");
        }
        System.out.println("Program continues...");

    }
}
```

- Next Monday, Test #2