COMP 110-001
Exception Handling

Yi Hong

June 10, 2015

Announcement

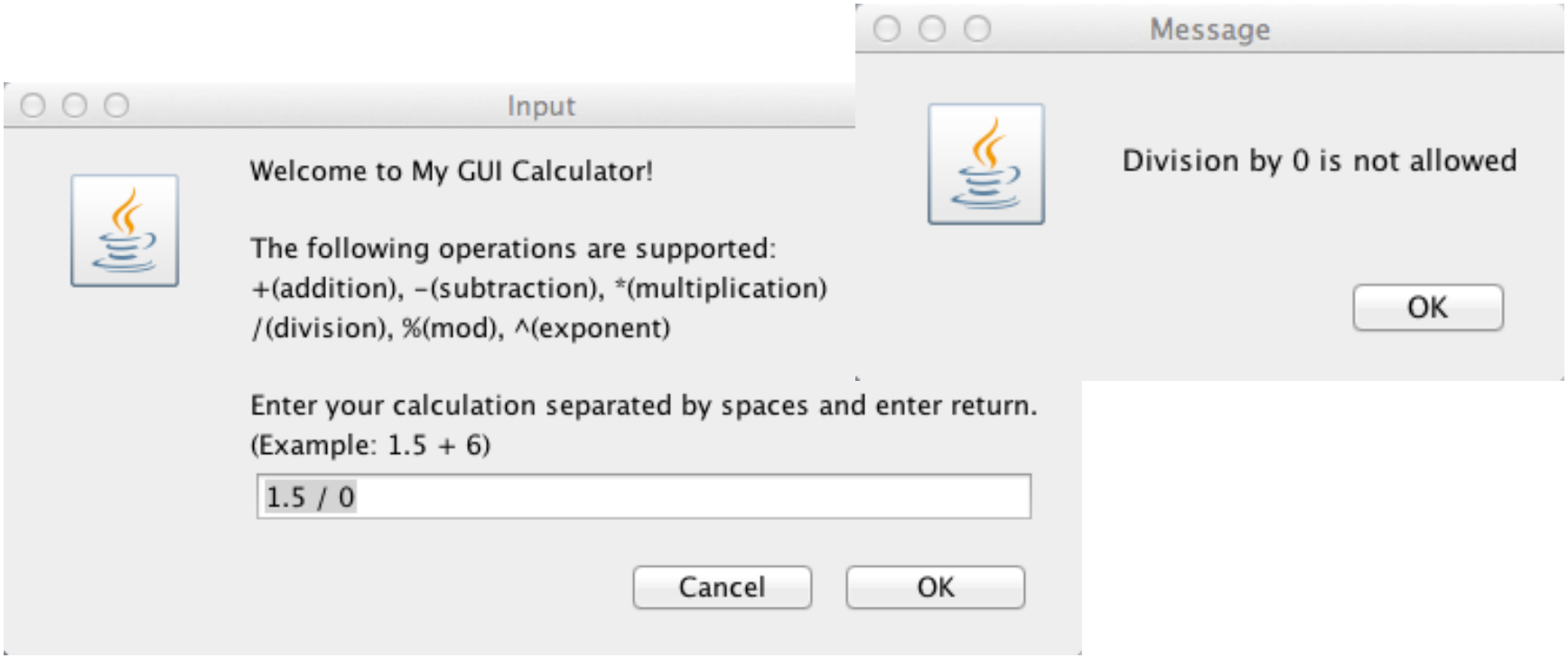
- Lab 7 is due today

Today

- Exception Handling
  - Important in developing Java code. But not a major focus of this course

Recall Homework 2

- **Homework 2: GUI Calculator**



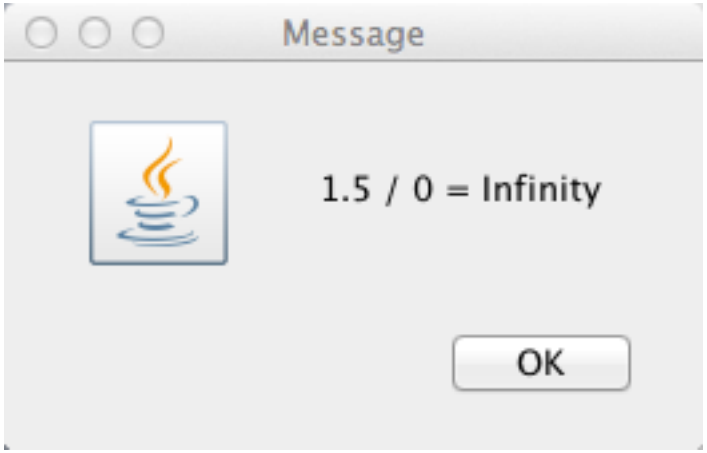
If the user try to divide by 0, prints out a message

Recall Homework 2

- **Homework 2: GUI Calculator**
  - Each of you used an if-else statement to test whether it is a division and the second operand is 0
  - If it is divided by 0, did you still do the division after you print out the message?

Recall Homework 2

- If you choose not to do, you have handled this case by skipping the result calculation part
- If you still calculates the result, you will probably get the output like this:



for floating-point number

If two numbers are integers, the program terminated due to the error.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Calculator.main(Calculator.java:63)
```

What Is The Right Thing To Do…

- When your code detects a problem?
- In program 2, we printed out a message to indicate a problem. And may choose to skip result calculation
- Not so many problems for a small program. We have control of everything involved
- But things quickly become messy when we want to write something slightly bigger

What If….

- What if you are writing some classes for others to use…
- What do you plan to do when your code detects some problem?

- Do you print out a message?
  - What if the program that uses your class runs in graphical mode?
  - Does the program really want some “uncontrolled” print-outs?

- Do you just let resulting errors terminate the program?
  - Sounds like a terrible idea in most cases
  - But if your class should do something and it is not performed properly, how to inform the program that uses the class?
  - E.g., a method in your class is called and is supposed to return some value. When your code sees error, should it still return any value?
    - If yes, what value?

What If….

- You are using someone's class for your program.
- E.g., you use the classes provided by Java to read from or write to a file.

- If some problems happens in reading / writing (file not found, cannot read/write), how does your program get notified?

The Need of a Formal Mechanism

- A formal mechanism is needed to handle "problems"
- "Problems" in one class should be reported and handled differently in different programs.
- This mechanism is different from return values in method-calling

Try-Throw-Catch

- In Java, the mechanism is called “Exception Handling”
  - Try to execute some actions
  - **Throw an exception**: report a problem and asks for some code to handle it properly
  - **Catch an exception**: a piece of code dedicated to handle one or more specific types of problem

Another Implementation Using Exception Handling

Try block

Catch block

```
try
{
    if(secondOperand == 0)
        throw new Exception("Division by 0 is not allowed");
    caluResult = firstOperand / secondOperand;
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    System.exit(0);
}
```

An exception's getMessage method returns a description of the exception

- A try bock detects an exception
- A throw statement throws an exception
- A catch block deals with a particular exception

More About Exception

- If an exception occurs within a try block, the rest of the block is ignored
- If no exception occurs within a try block, the catch blocks are ignored
- An exception is an object of the class Exception

Handling Exceptions

- Syntax for the try and catch statements

```
try
{
    Code_To_Try
    Possibly_Throw_An_Exception
    More_Code
}
catch (Exception_Class_NameCatch_Block_Parameter)
{
    Process_Exception_Of_Type_Exception_Class_Name
}
Possibly_Other_Catch_Blocks
```

- Syntax for the throw statement

```
throw new Exception_Class_Name(Possibly_Some_Arguments);
```

Another Example

```
1  import java.util.Scanner;
2
3  public class ExceptionDemo {
4    public static void main(String[] args) {
5      Scanner scanner = new Scanner(System.in);
6      System.out.print("Enter an integer: ");
7      int number = scanner.nextInt();
8
9      // Display the result
10     System.out.println(
11       "The number entered is " + number);
12   }
13 }
```

If an exception occurs on this line, the rest of the lines in the method are skipped and the program is terminated.

Terminated.

Another Example

```
1  import java.util.*;
2
3  public class HandleExceptionDemo {
4    public static void main(String[] args) {
5      Scanner scanner = new Scanner(System.in);
6      boolean continueInput = true;
7
8      do {
9        try {
10         System.out.print("Enter an integer: ");
11         int number = scanner.nextInt();
12
13         // Display the result
14         System.out.println(
15           "The number entered is " + number);
16
17         continueInput = false;
18       }
19       catch (InputMismatchException ex) {
20         System.out.println("Try again. (" +
21           "Incorrect input: an integer is required)");
22         scanner.nextLine(); // discard input
23       }
24     } while (continueInput);
25   }
```

If an exception occurs on this line, the rest of lines in the try block are skipped and the control is transferred to the catch block.

Predefined Exception Classes

- Java provides several exception classes
  - The names are designed to be self-explanatory
    - E.g.: BadStringOperationException,
      ClassNotFoundException,
      IOException,
      NoSuchMethodException,
      InputMismatchException
  - Use the try and catch statements

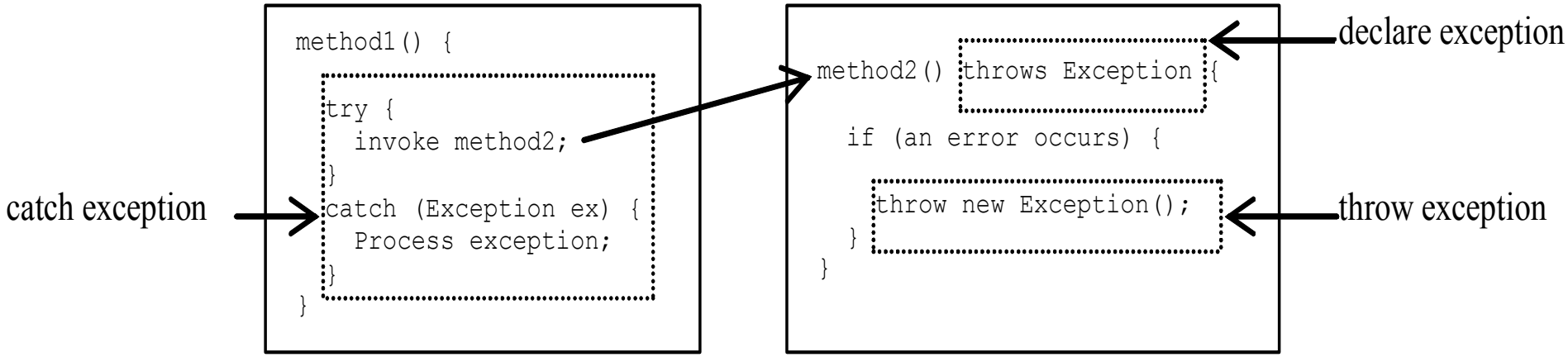
An Example

```
SampleClass object = new SampleClass();
try
{
    <Possibly some code>
    object.doStuff(); //may throw IOException
    <Possibly some more code>
}
catch(IOException e)
{
    <Code to deal with the exception, probably including the
     following:>
    System.out.println(e.getMessage());
}
```

- If you think that continuing with program execution is infeasible after the exception occurs, use System.exit(0) to end the program in the catch block

Declaring Exceptions

- When we want to delay handling of an exception
- A method might not catch an exception that its code throws



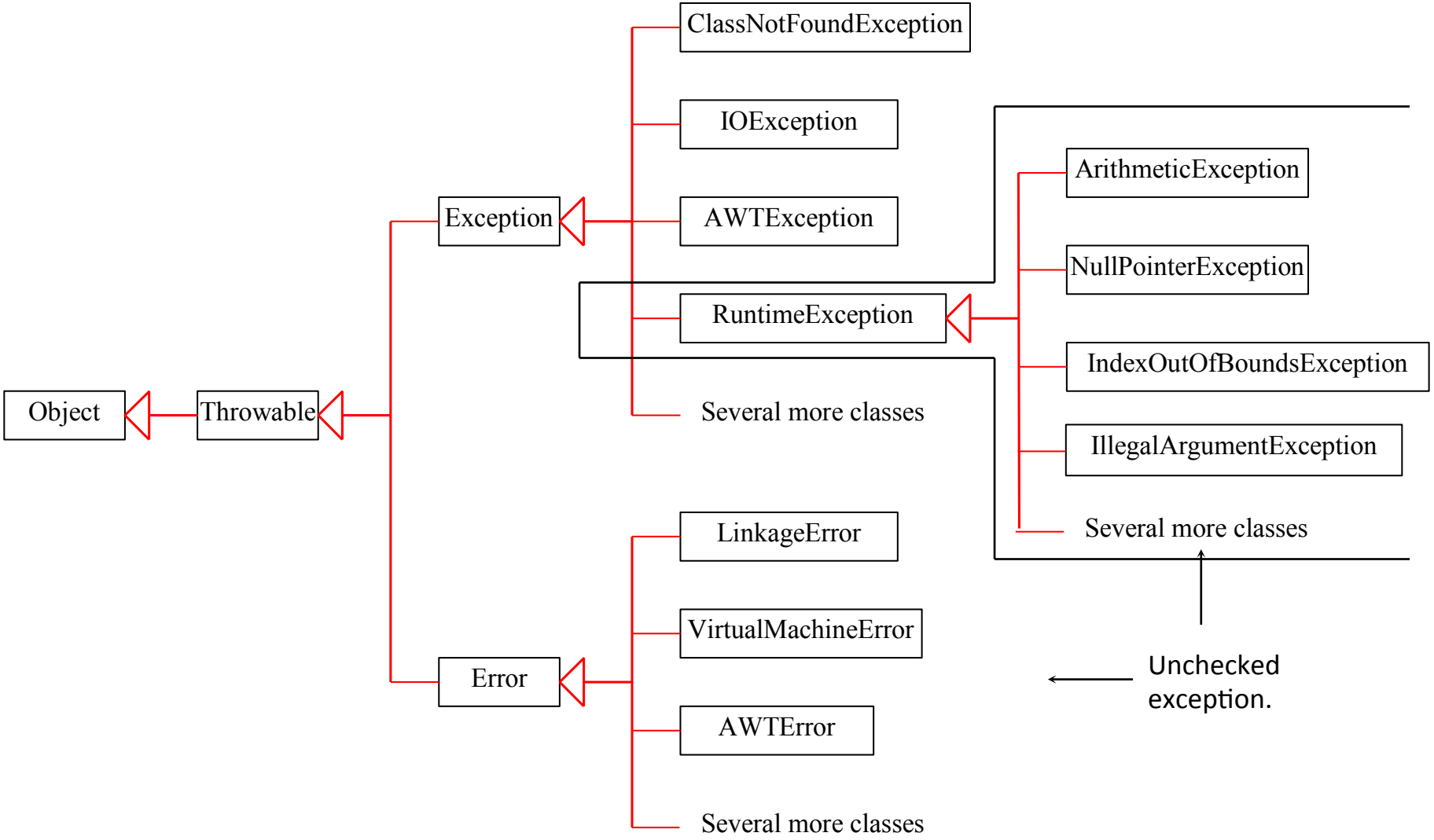
Throwing Exceptions Example

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

Step 1: add throws clause, "throws ExceptionType", in the method's heading

Step 2: when problem occurs, use a throw statement throws an exception, "throw new ExceptionType(….); "

The Java Exception Hierarchy



Checked Exceptions vs. Unchecked Exceptions

- RuntimeException, Error and their subclasses are known as *unchecked exceptions*
  - no need to be caught or declared in a throws clause of a method's heading
- All other exceptions are known as *checked exceptions*
  - must be either caught or declared in a throws clause

Unchecked Exceptions

- In most cases, unchecked exceptions reflect programming logic errors that are not recoverable
  - E.g., a NullPointerException is thrown if you access an object through a reference variable before an object is assigned to it
  - an ArrayIndexOutOfBoundsException is thrown if you access an element outside the bounds of the array

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at HandleExceptionDemo.main(HandleExceptionDemo.java:12)
```

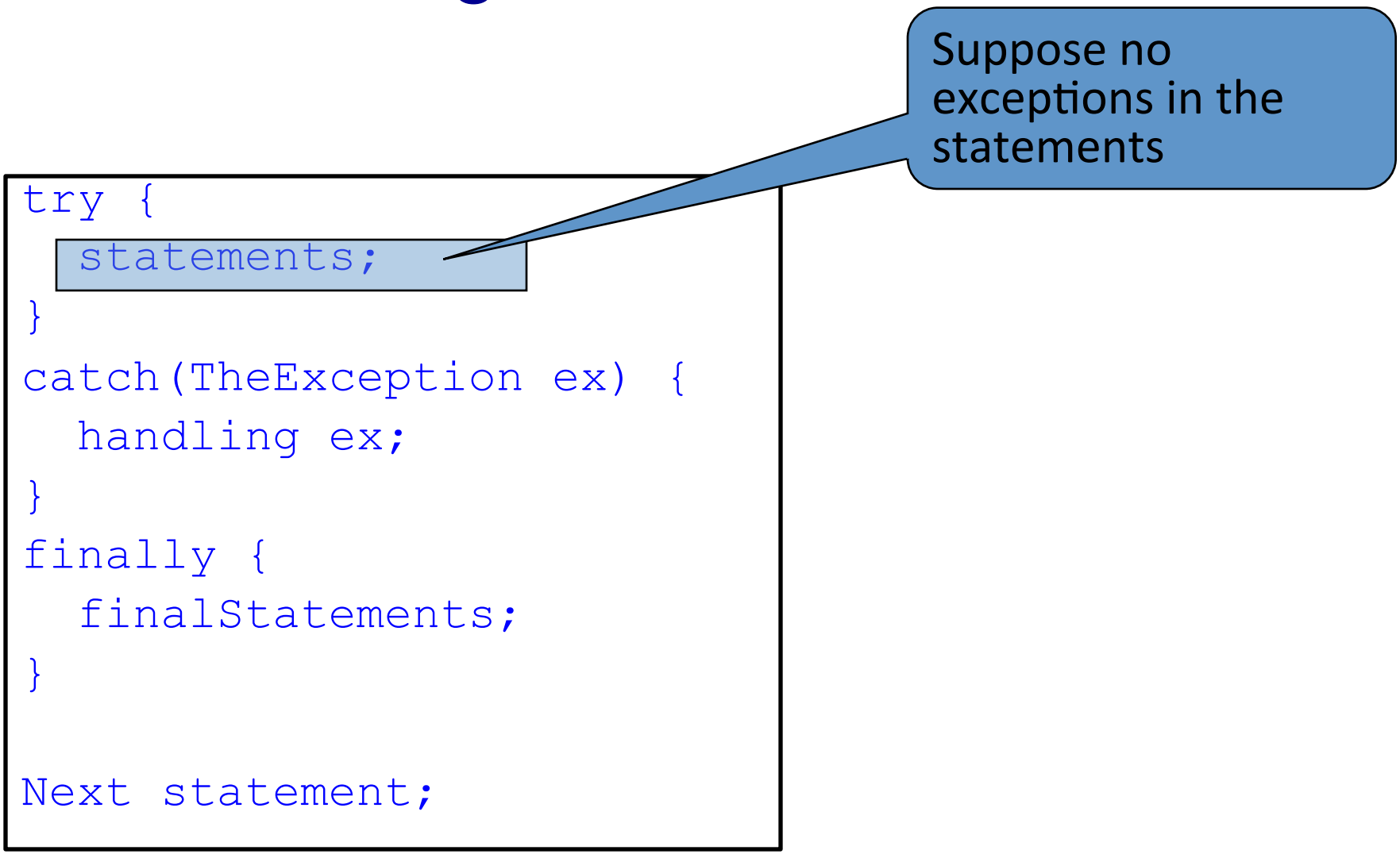
- Logic errors that should be corrected in the program, Java does not mandate you to write code to catch unchecked exceptions

The `finally` Bolck

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

- A finally block always executes
- Put cleanup code in a finally block, e.g., closing a file

Trace a Program Execution



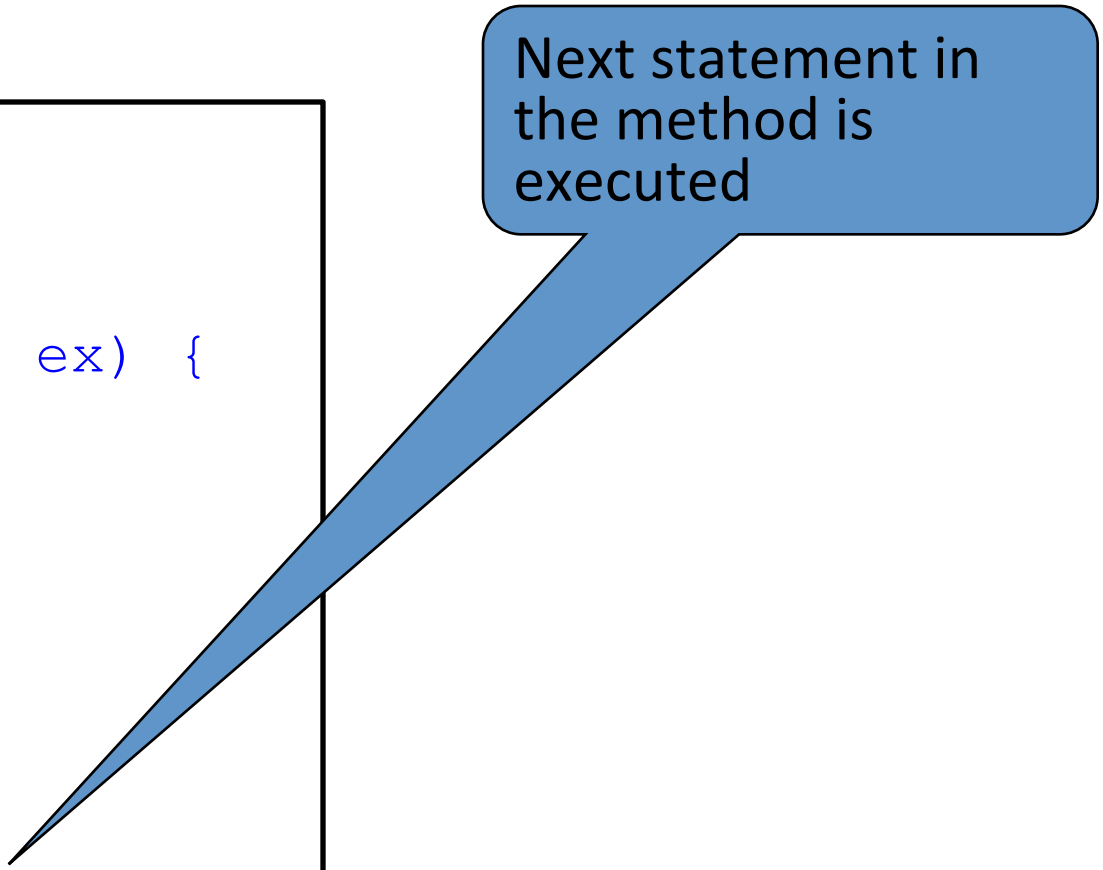
Trace a Program Execution

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

Next statement;
```

The final block is always executed

Trace a Program Execution

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

Next statement;
```



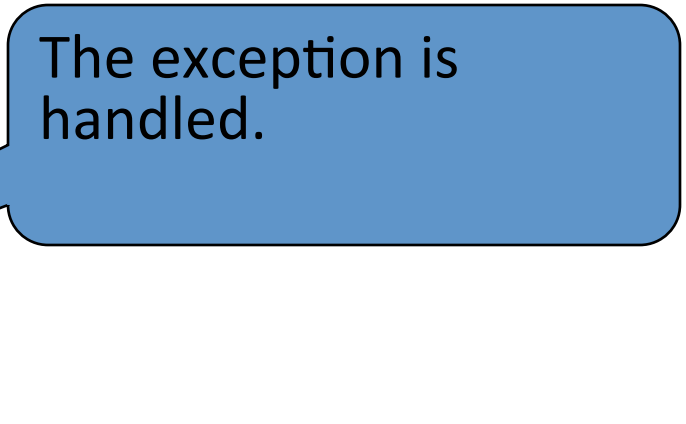
Trace a Program Execution

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

Next statement;
```

Suppose an exception of type Exception1 is thrown in statement2

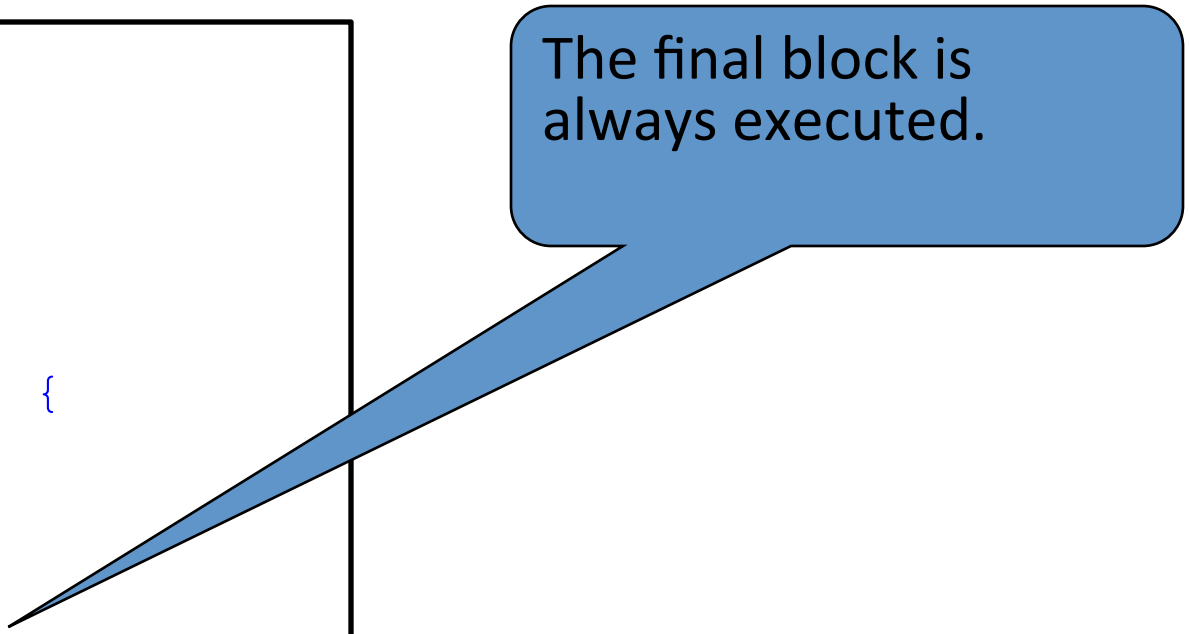
Trace a Program Execution

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

Next statement;
```



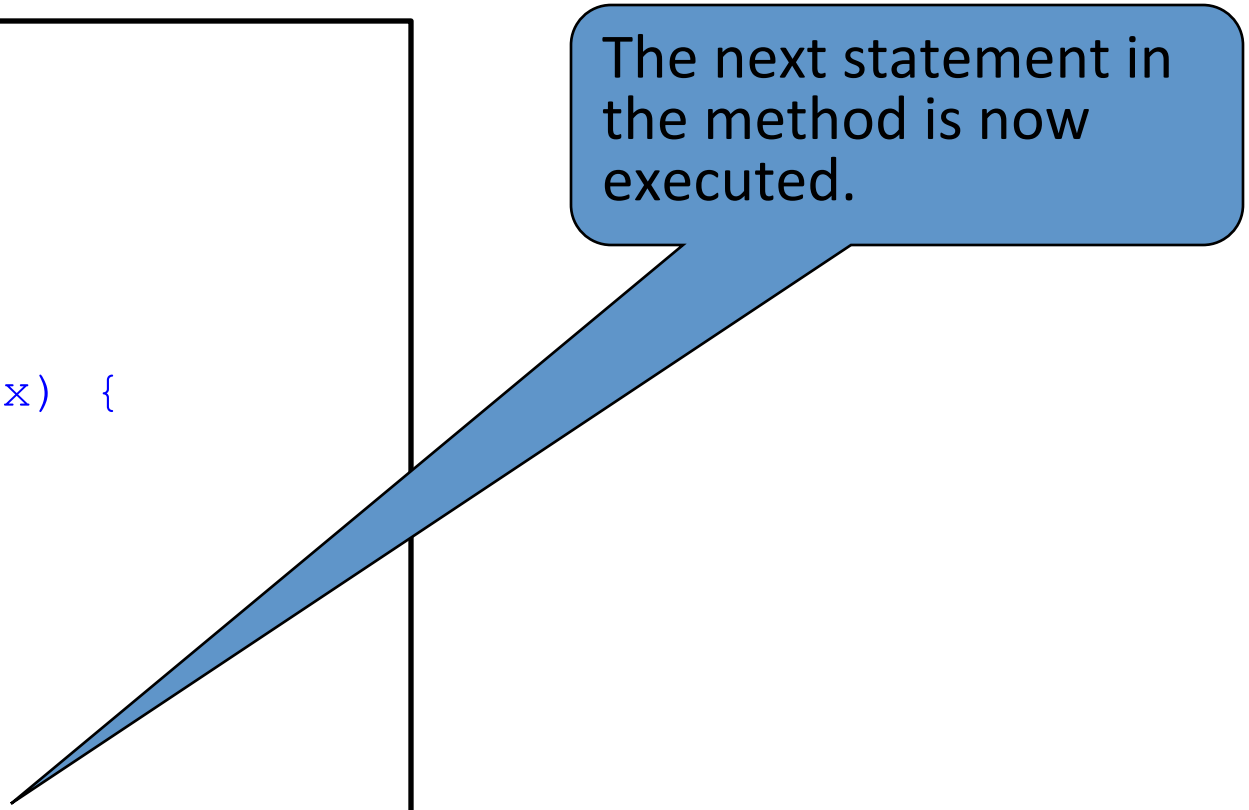
Trace a Program Execution

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

Next statement;
```



Trace a Program Execution

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

Next statement;
```



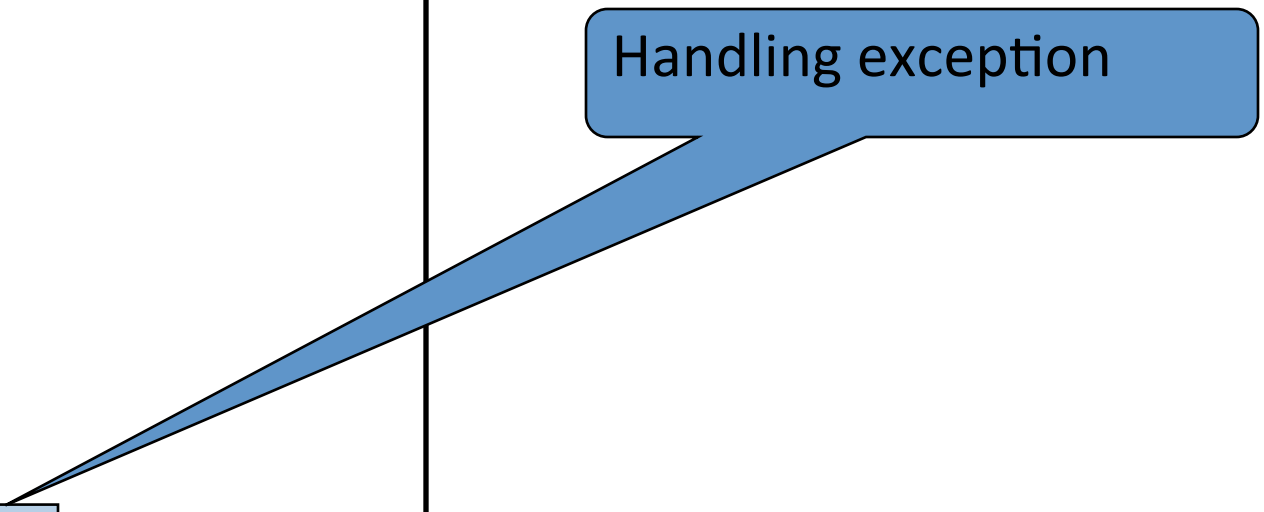
Trace a Program Execution

```
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.

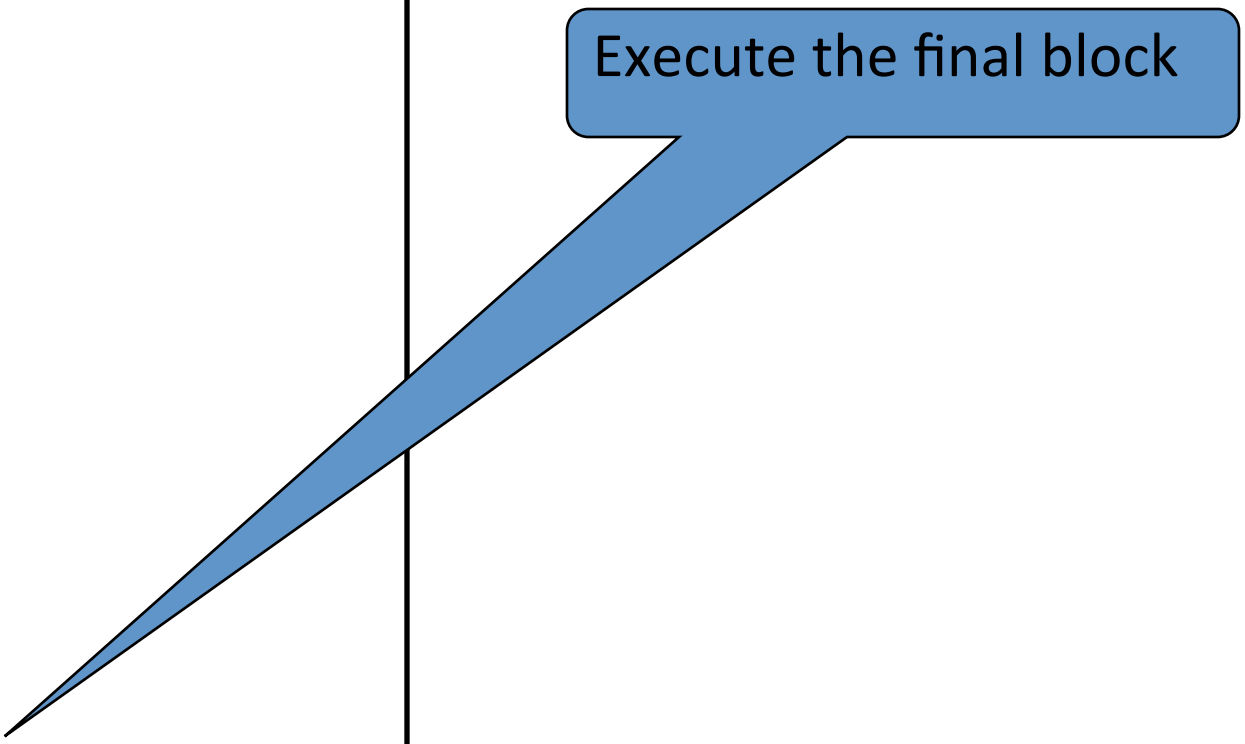
Trace a Program Execution

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

Next statement;
```



Trace a Program Execution

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

Next statement;
```



Trace a Program Execution

```
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

Next Class

- Streams and File I/O