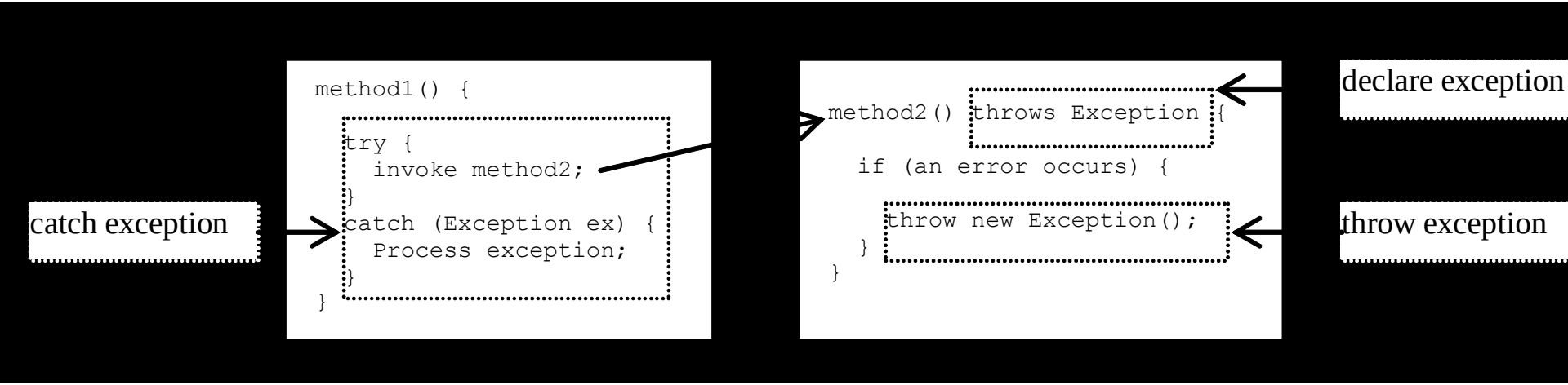
Exception handling

Try block and catch block

```
statement1;
try {
      statement2;
      statement3;
 }
catch (ArithmeticException ex) {
     statement4;
}
statement5;
```

Declaring, Throwing, and Catching Exceptions



Cohort Session 2: ModProg

Write a program that performs modular operation of two integer values, for example x % y. Prompt the user to enter two integer values. Catch an exception thrown if modular by zero. Also, catch all other exceptions using Exception. Your program should be in a loop until not 'y' is entered.

Cohort Session 2: TestScores

Write a program that checks the scores stored in an array. The method, TestScores, will throw IllegalArgumentException("Element: " + i + " Score: " + s[i]) if the values in array *s* are less than 0 or bigger than 100. Use try … catch block in main() to handle this exception.

Exception Handling with finally block

```
statement1;
try {
      statement2;
      statement3;
 }
catch (ArithmeticException ex) {
     statement4;
}
finally {
    statement5;
}
statement6;
```

Scenarios to previous slide

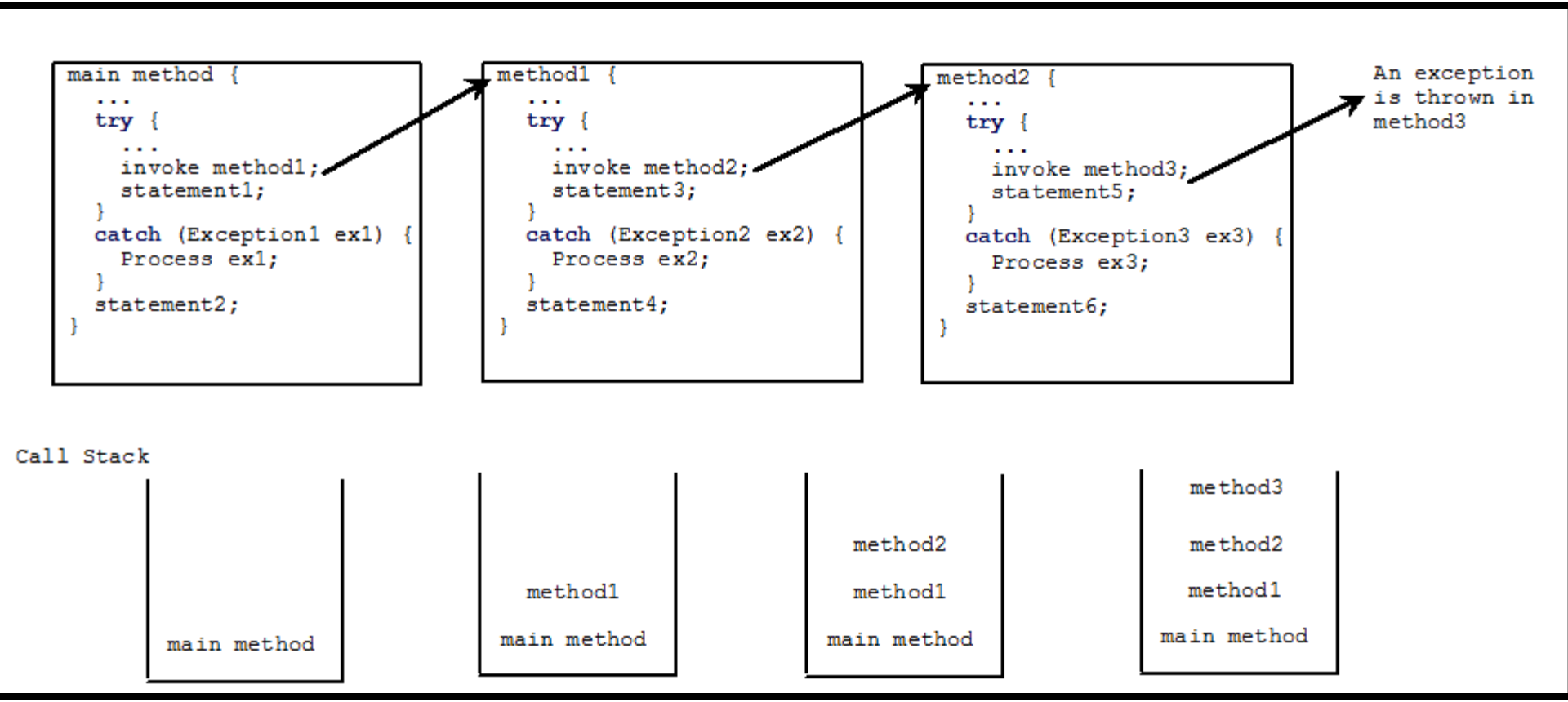
- The code in the **finally** block is executed under all circumstances, regardless of whether an exception occurs in the try block or is caught. For example, **statement 5**
- If no exception arises in the **try** block, **statement1, statement2, statement3, statement5 and statement6** are executed.
- If a **statement 2** causes an exception in the **try** block that is caught in a **catch** block, the rest of the statements in the try block are skipped, the **catch** block is executed, **finally** block is executed, and the next statement is executed. For example, **statement1, statement2, statement4, statement5 and statement6** are executed.
- If **statement2** causes an exception that is not caught in any **catch** block, the other statements in the **try** block are skipped, the **finally** block is executed (**statement 5**), and the exception is passed to the caller of this method. **The program terminates and**

Example1: ChainedExceptions

```
public class ChainedExceptions {

    public static void main(String[] args) {
        try {
            method1();  }
        catch (Exception ex) {   // ex.printStackTrace();
            System.out.println(ex);
       } }
    public static void method1() throws Exception {
        try {
            method2();  }
      catch (Exception ex) {
            ex.printStackTrace();
            System.out.println(ex);
            throw new Exception("More info from method1", ex);
     } }
     public static void method2() throws Exception {
            throw new Exception("Info from method2");
     } }
```

Catching Exceptions



Suppose the main method invokes method1, method1 invokes method2, method2 invokes method3, and method3 throws an exception. Consider scenarios in the next slide.

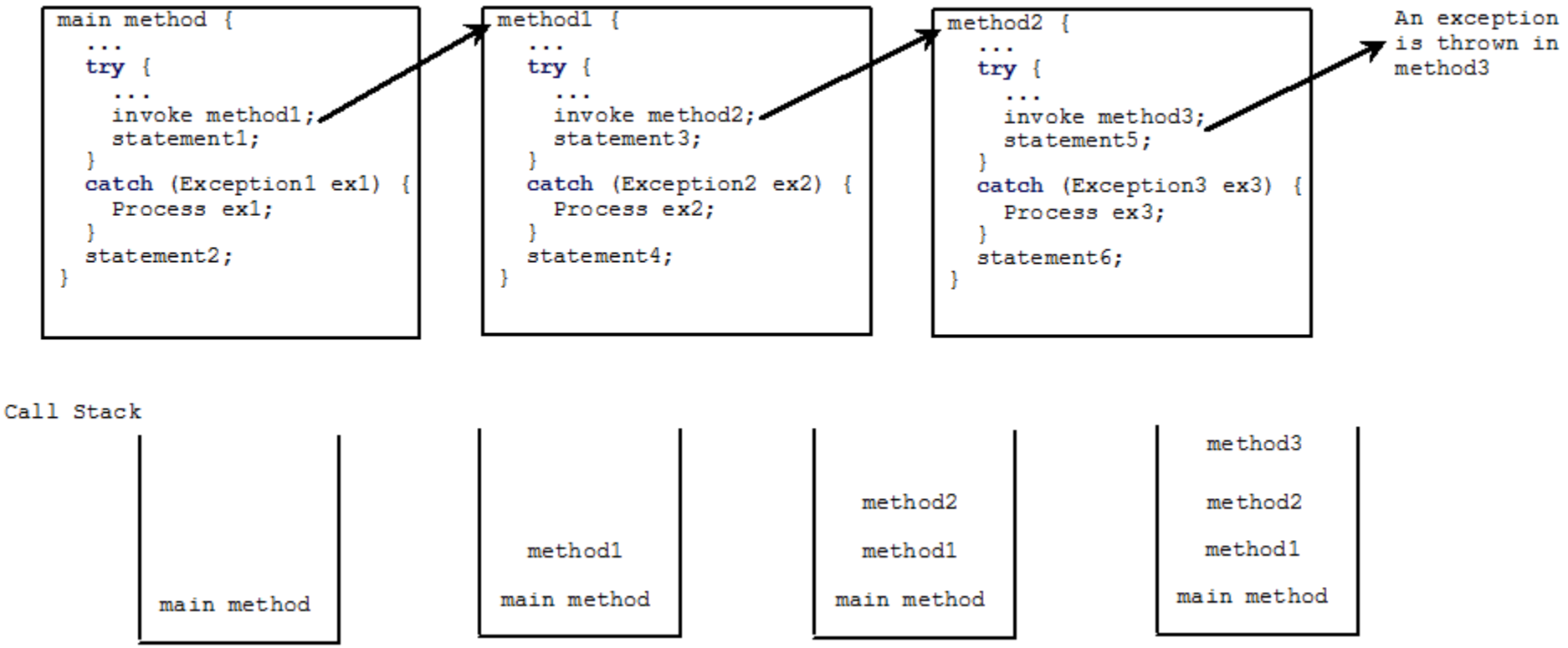
Scenarios to previous slide

- If the exception type is **Exception3**, it is caught by the **catch** block for handling exception **ex3** in **method2**. **statement5** is skipped, and **statement6** is executed …
- If the exception type is **Exception2**, **method2** is aborted, the control is returned to **method1**, and the exception is caught by the **catch** block for handling exception **ex2** in **method1**. **statement3** is skipped, and **statement4** is executed …
- If the exception type is **Exception1**, **method1** is aborted, the control is returned to **main** method, and the exception is caught by the **catch** block for handling exception **ex1** in **main** method. **statement1** is skipped, and **statement2** is executed.
- If the exception type is not caught in **method2, method1, or main**, the program terminates, **statement1 and statement2** are not executed. Error messages are printed on console.

Example2: Catching Exceptions

```
import java.util.*;
public class CatchingExceptionsv2 {
public static void main(String[] args) {
    try {
        method1();
        System.out.println("Statement1"); }
    catch (IndexOutOfBoundsException ex) {
     // ex.printStackTrace();
        System.out.println(ex); }
        System.out.println("Statement2");  }
public static void method1() throws ArithmeticException {
      try {
           method2();
          System.out.println("Statement3");  }
      catch (ArithmeticException ex) {
           System.out.println(ex);  }
           System.out.println("Statement4"); }
public static void method2() throws IllegalArgumentException  {
           throw new IllegalArgumentException("Info from method2"); } }
```

Cohort Session 3: Catching Exceptions



Implement the diagram above. Then execute the following scenarios:

(1) An exception is thrown in method3 and is caught in method1. Print the statements that are executed.

(2) An exception is thrown in method3 and is not caught in method2, method1, and method. Print the statements that are executed.

Example3: Custom Exception

```
import java.util.*;
public class MyInputMismatch extends Exception {
   public MyInputMismatch(String message)  {
      System.out.println("MyInputMismatch constructor executed"); }
    public static void main(String [] args) {
      Scanner input = new Scanner(System.in);
      boolean continueInput = true;
      do {
         try {
           System.out.println("Enter an integer: ");
           int number = input.nextInt();
           if (number  < 0)
                throw new MyInputMismatch("Value less than zero");
           System.out.println("The number entered is " + number);
           continueInput=false;  }
        catch (MyInputMismatch ex) {
            System.out.println(ex);
            System.out.println("Try again. An integer is required.");
            input.nextLine(); //clear the input  }
   }  while (continueInput);
   System.out.println("Execution continues here"); }  }
```