

**Experiment No. 08**  
**Experiment Name: Exception Handling**

*Course title: Programming Language II(Java) Lab*  
*Course code:*  
*Spring 2025*

**Date of Submission:**



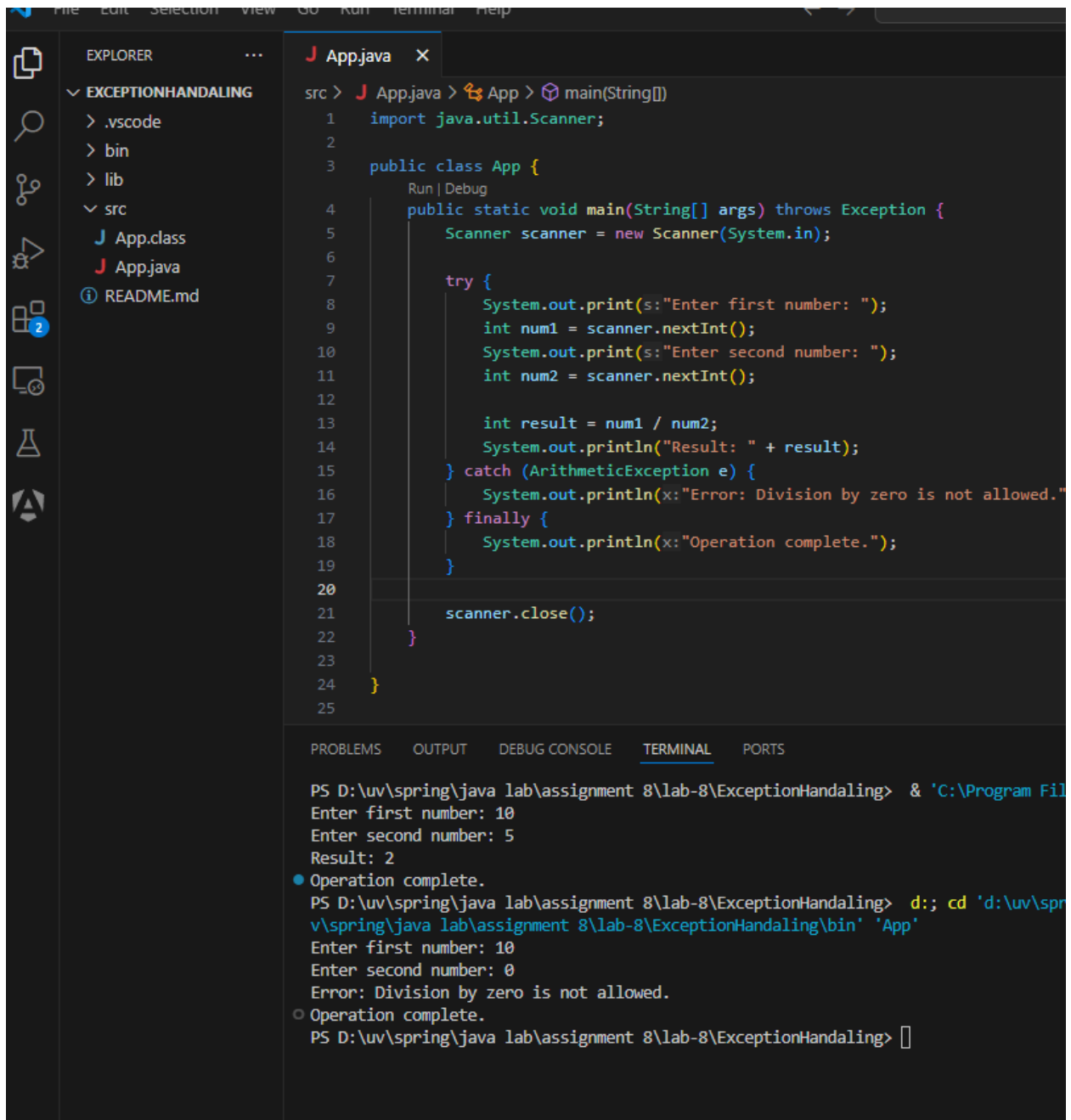
**Submitted to-**

**Md. Rafsan Jani**  
*Assistant Professor*  
*Department of Computer Science and Engineering*

Sl	Class Roll	Name
01	2023000010034	Md Arafat Rahman

## 1. Division with Exception Handling:

Here's how you can write a program to handle an `ArithmeticException` when dividing two integers:



The screenshot displays the Visual Studio Code editor with a Java file named `App.java` open. The Explorer sidebar on the left shows the project structure, including a `src` folder containing `App.class` and `App.java`. The main editor window shows the following Java code:

```
src > J App.java > App > main(String[])
1  import java.util.Scanner;
2
3  public class App {
4      public static void main(String[] args) throws Exception {
5          Scanner scanner = new Scanner(System.in);
6
7          try {
8              System.out.print(s:"Enter first number: ");
9              int num1 = scanner.nextInt();
10             System.out.print(s:"Enter second number: ");
11             int num2 = scanner.nextInt();
12
13             int result = num1 / num2;
14             System.out.println("Result: " + result);
15         } catch (ArithmeticException e) {
16             System.out.println(x:"Error: Division by zero is not allowed.");
17         } finally {
18             System.out.println(x:"Operation complete.");
19         }
20
21         scanner.close();
22     }
23
24 }
25
```

Below the code editor, the TERMINAL panel shows the execution of the program. It prompts for two numbers, calculates the result, and handles a division-by-zero error gracefully.

```
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling> & 'C:\Program Files\Java\jdk-11.0.10\bin\java.exe' -cp 'D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling\bin' App
Enter first number: 10
Enter second number: 5
Result: 2
● Operation complete.
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling> d:; cd 'd:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling\bin' 'App'
Enter first number: 10
Enter second number: 0
Error: Division by zero is not allowed.
○ Operation complete.
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling>
```

## 2. Handling Array and NullPointerExceptions:

This program demonstrates handling multiple exceptions, such as `ArrayIndexOutOfBoundsException` and `NullPointerException`.

The screenshot displays a VS Code editor with a Java file named `App.java` in the `src` directory. The code defines a `main` method that takes a `String[] args` and throws `Exception`. It initializes an array `int[] array = { 10, 20, 30, 40, 50 };` and a `String str = null;`. A `Scanner` is created for `System.in`. The `try` block prompts the user to enter an array index. If the index is out of bounds, it catches `ArrayIndexOutOfBoundsException` and prints an error message. If a `NullPointerException` occurs (due to accessing `str.length()`), it catches that exception and prints an error message. The `finally` block prints "Operation complete." and the `scanner` is closed.

The terminal output shows the program's execution for three different inputs:

- Input: 1  
Output: Value at index: 20  
Error: Attempted to access a null object!  
Operation complete.
- Input: 2  
Output: Value at index: 30  
Error: Attempted to access a null object!  
Operation complete.
- Input: 5  
Output: Error: Index is out of bounds!  
Operation complete.

```
src > J App.java > App > main(String[])
1  import java.util.Scanner;
2
3  public class App {
4      public static void main(String[] args) throws Exception {
5          int[] array = { 10, 20, 30, 40, 50 };
6          String str = null;
7          Scanner scanner = new Scanner(System.in);
8          try {
9              System.out.print(s:"Enter array index: ");
10             int index = scanner.nextInt();
11             System.out.println("Value at index: " + array[index]);
12
13             // Simulating a NullPointerException
14             System.out.println("String length: " + str.length());
15         } catch (ArrayIndexOutOfBoundsException e) {
16             System.out.println(x:"Error: Index is out of bounds!");
17         } catch (NullPointerException e) {
18             System.out.println(x:"Error: Attempted to access a null object!");
19         } finally {
20             System.out.println(x:"Operation complete.");
21         }
22         scanner.close();
23     }
24 }
25
26
27 // Scanner scanner = new Scanner(Svstem.in);
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

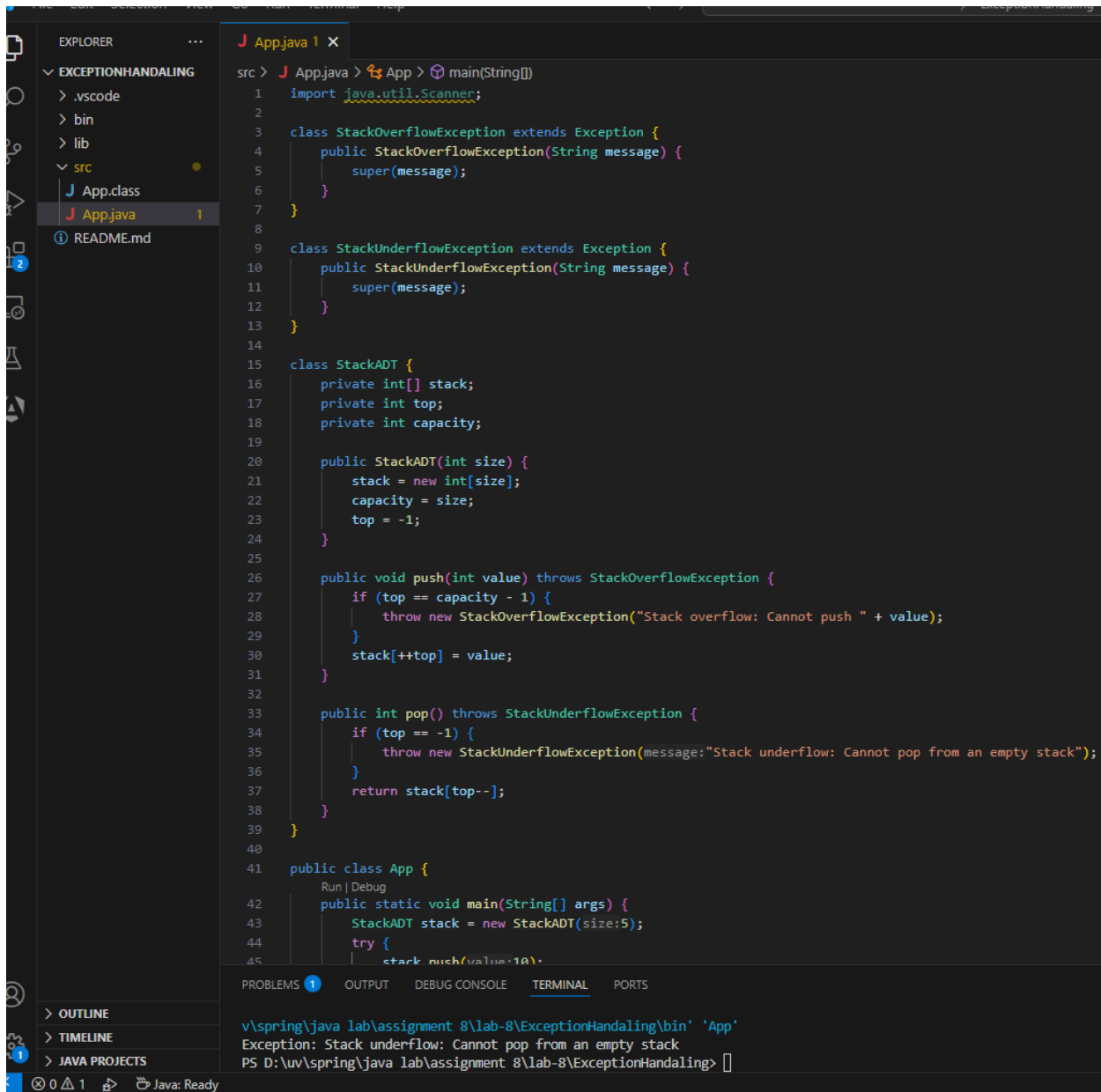
v\spring\java lab\assignment 8\lab-8\ExceptionHandling\bin' 'App'

- Enter array index: 1  
Value at index: 20  
Error: Attempted to access a null object!  
Operation complete.  
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling> d;; cd 'd:\uv\spring\j  
& 'C:\Program Files\Java\jdk-23.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessag  
ab-8\ExceptionHandling\bin' 'App'
- Enter array index: 2  
Value at index: 30  
Error: Attempted to access a null object!  
Operation complete.  
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling> d;; cd 'd:\uv\spring\j  
o & 'C:\Program Files\Java\jdk-23.0.1\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessag  
ab-8\ExceptionHandling\bin' 'App'
- Enter array index: 5  
Error: Index is out of bounds!  
Operation complete.  
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling>

> OUTLINE  
> TIMELINE  
> JAVA PROJECTS

### 3. Custom Exceptions for Stack Overflow and Underflow:

Here's how you can define custom exceptions and use them in a stack implementation:



The screenshot shows a VS Code editor with a Java project. The Explorer panel on the left shows the project structure with folders .vscode, bin, lib, and src. The src folder contains App.class, App.java, and README.md. The App.java file is open in the editor, showing the following code:

```
src > J App.java > App > main(String[])
1  import java.util.Scanner;
2
3  class StackOverflowException extends Exception {
4      public StackOverflowException(String message) {
5          super(message);
6      }
7  }
8
9  class StackUnderflowException extends Exception {
10     public StackUnderflowException(String message) {
11         super(message);
12     }
13 }
14
15 class StackADT {
16     private int[] stack;
17     private int top;
18     private int capacity;
19
20     public StackADT(int size) {
21         stack = new int[size];
22         capacity = size;
23         top = -1;
24     }
25
26     public void push(int value) throws StackOverflowException {
27         if (top == capacity - 1) {
28             throw new StackOverflowException("Stack overflow: Cannot push " + value);
29         }
30         stack[++top] = value;
31     }
32
33     public int pop() throws StackUnderflowException {
34         if (top == -1) {
35             throw new StackUnderflowException(message:"Stack underflow: Cannot pop from an empty stack");
36         }
37         return stack[top--];
38     }
39 }
40
41 public class App {
42     Run | Debug
43     public static void main(String[] args) {
44         StackADT stack = new StackADT(size:5);
45         try {
46             stack.push(value:10);
47         } catch (Exception e) {
48             e.printStackTrace();
49         }
50     }
51 }
```

The bottom panel shows the TERMINAL output:

```
v:\spring\java lab\assignment 8\lab-8\ExceptionHandling\bin' 'App'
Exception: Stack underflow: Cannot pop from an empty stack
PS D:\uv\spring\java lab\assignment 8\lab-8\ExceptionHandling> []
```