

Name	Udhayarajan J
Class	MCA - I
Sub	Java Programming
Type	Experiment Practical Program
Date	2025-11-12
Roll No	2202561

## Experiment 2: Introduction of arrays and different operations on arrays

Design an application by using arrays

### a. Finding the Largest/Smallest Element

The screenshot shows an IDE with a Java file named `findLargestAndSmallestElement.java`. The code defines a class `FindMinMax` with a `main` method. It initializes an array `arr` with values `{ 12, 45, 9, 78, 23, 5 }`. It then iterates through the array to find the maximum and minimum values. The output in the terminal window is:

```
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java findLargestAndSmallestElement.java
Largest element: 78
Smallest element: 5
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

```
1 class FindMinMax {
2     public static void main(String[] args) {
3         int[] arr = { 12, 45, 9, 78, 23, 5 };
4         int max = arr[0];
5         int min = arr[0];
6         for (int i = 1; i < arr.length; i++) {
7             if (arr[i] > max) {
8                 max = arr[i];
9             }
10            if (arr[i] < min) {
11                min = arr[i];
12            }
13        }
14        System.out.println("Largest element: " + max);
15        System.out.println("Smallest element: " + min);
16    }
17 }
```

### b. Reversing an Array

The screenshot shows an IDE with a Java file named `ReverseArray.java`. The code defines a class `ReverseArray` with a `main` method. It initializes an array `original` with values `{ 1, 2, 3, 4, 5 }` and creates a new array `reversed` of the same length. It then iterates through the `original` array to populate the `reversed` array in reverse order. The output in the terminal window is:

```
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java ReverseArray.java
Original array: [1, 2, 3, 4, 5]
Reversed array: [5, 4, 3, 2, 1]
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

```
1 public class ReverseArray {
2     public static void main(String[] args) {
3         int[] original = { 1, 2, 3, 4, 5 };
4         int[] reversed = new int[original.length];
5         for (int i = 0; i < original.length; i++) {
6             reversed[i] = original[original.length - 1 - i];
7         }
8         System.out.println("Original array: " + java.util.Arrays.toString(original));
9         System.out.println("Reversed array: " + java.util.Arrays.toString(reversed));
10    }
11 }
```

### c. Sum of all elements in a 2D Array



The screenshot shows an IDE with the file `SumOf2DArrayElements.java` open. The code defines a `main` method that initializes a 2D array `numbers` with the following values:

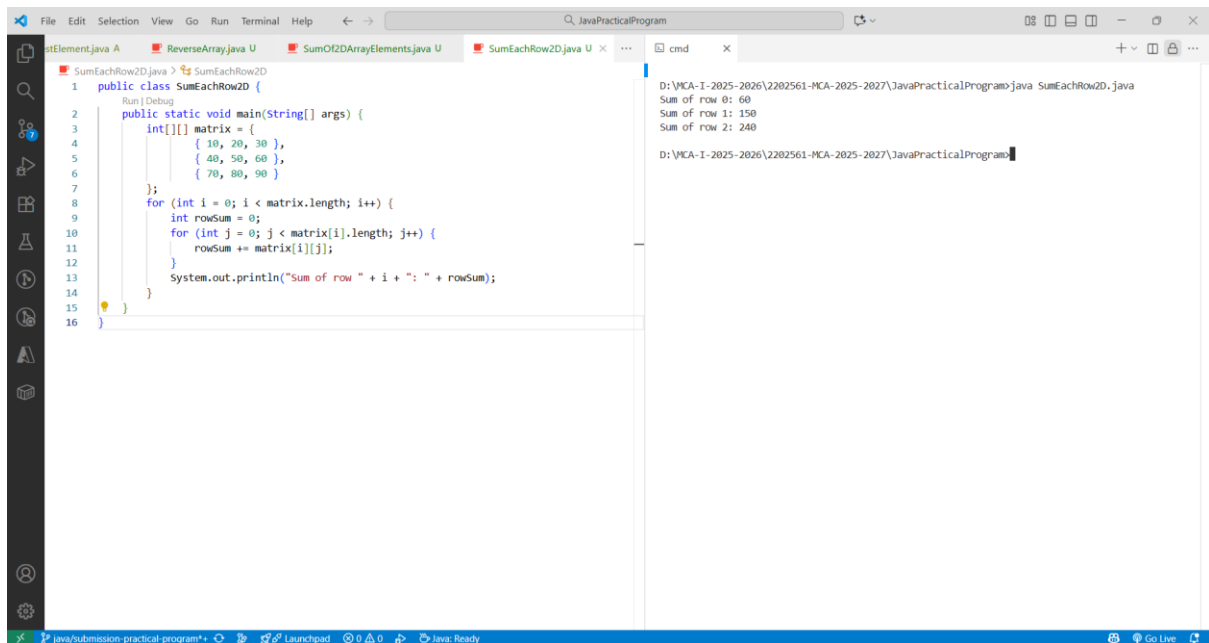
```
int[][] numbers = {
    { 10, 20, 30 },
    { 40, 50, 60 }
};
```

The code then calculates the sum of all elements in the array using nested loops and prints the result: "Sum of all elements: 210". The terminal output on the right confirms this result.

```
public class SumOf2DArrayElements {
    public static void main(String[] args) {
        int[][] numbers = {
            { 10, 20, 30 },
            { 40, 50, 60 }
        };
        int sum = 0;
        for (int i = 0; i < numbers.length; i++) {
            for (int j = 0; j < numbers[i].length; j++) {
                sum += numbers[i][j];
            }
        }
        System.out.println("Sum of all elements: " + sum);
    }
}
```

```
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java SumOf2DArrayElements.java
Sum of all elements: 210
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

### d. Sum of elements in each row of a 2D array



The screenshot shows an IDE with the file `SumEachRow2D.java` open. The code defines a `main` method that initializes a 2D array `matrix` with the following values:

```
int[][] matrix = {
    { 10, 20, 30 },
    { 40, 50, 60 },
    { 70, 80, 90 }
};
```

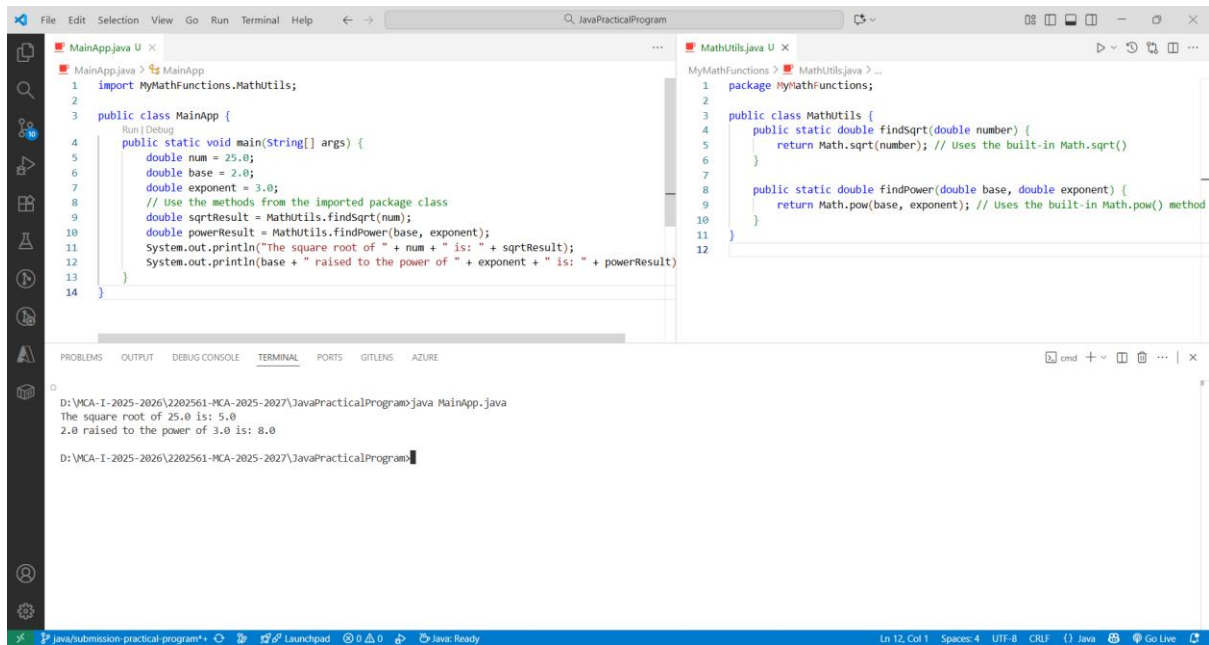
The code then calculates the sum of elements in each row using nested loops and prints the results: "Sum of row 0: 60", "Sum of row 1: 150", and "Sum of row 2: 240". The terminal output on the right confirms these results.

```
public class SumEachRow2D {
    public static void main(String[] args) {
        int[][] matrix = {
            { 10, 20, 30 },
            { 40, 50, 60 },
            { 70, 80, 90 }
        };
        for (int i = 0; i < matrix.length; i++) {
            int rowSum = 0;
            for (int j = 0; j < matrix[i].length; j++) {
                rowSum += matrix[i][j];
            }
            System.out.println("Sum of row " + i + ": " + rowSum);
        }
    }
}
```

```
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java SumEachRow2D.java
Sum of row 0: 60
Sum of row 1: 150
Sum of row 2: 240
D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

### 3: Package

Write a program for Implementation of package to create class and methods in mynmathfunction package



```
File Edit Selection View Go Run Terminal Help JavaPracticalProgram
```

```
MainApp.java > MainApp
1 import MyMathFunctions.MathUtils;
2
3 public class MainApp {
4     public static void main(String[] args) {
5         double num = 25.0;
6         double base = 2.0;
7         double exponent = 3.0;
8         // Use the methods from the imported package class
9         double sqrtResult = MathUtils.findSqrt(num);
10        double powerResult = MathUtils.findPower(base, exponent);
11        System.out.println("The square root of " + num + " is: " + sqrtResult);
12        System.out.println(base + " raised to the power of " + exponent + " is: " + powerResult);
13    }
14 }
```

```
MathUtils.java > MathUtils.java > ...
1 package MyMathFunctions;
2
3 public class MathUtils {
4     public static double findSqrt(double number) {
5         return Math.sqrt(number); // Uses the built-in Math.sqrt()
6     }
7
8     public static double findPower(double base, double exponent) {
9         return Math.pow(base, exponent); // Uses the built-in Math.pow() method
10    }
11 }
12
```

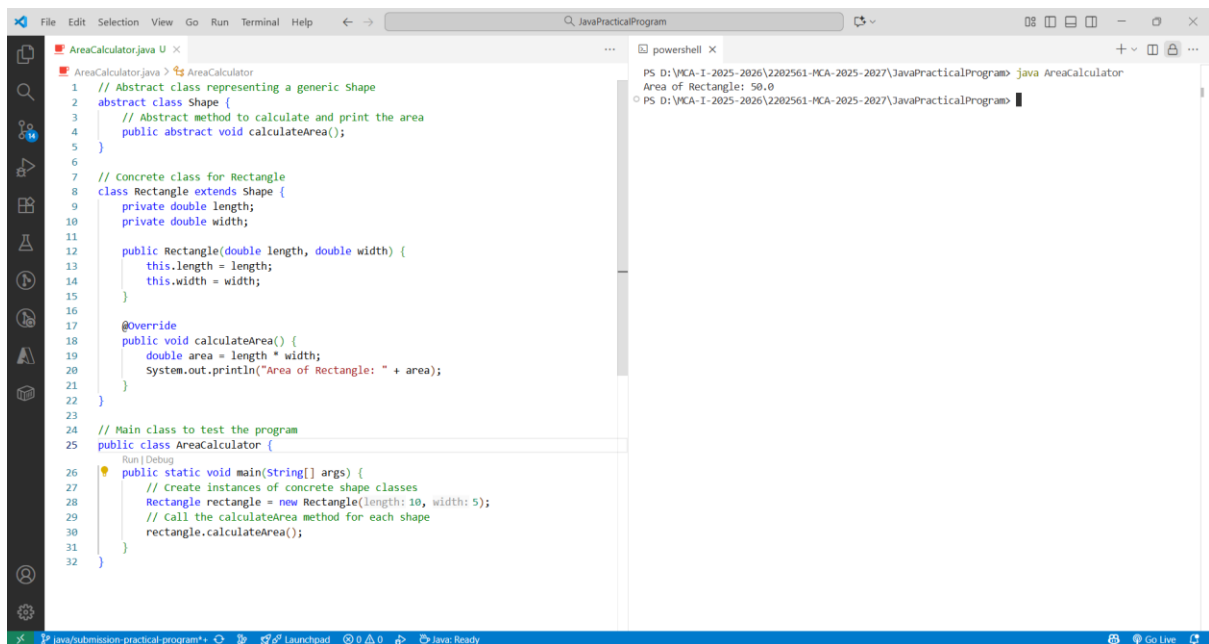
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS GIT LENS AZURE
```

```
D:\VKA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java MainApp.java
The square root of 25.0 is: 5.0
2.0 raised to the power of 3.0 is: 8.0

D:\VKA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

Ln 12, Col 1 Spaces: 4 UTF-8 CRLF Java Go Live

### b. abstract class Experiment for calculating area



```
File Edit Selection View Go Run Terminal Help JavaPracticalProgram
```

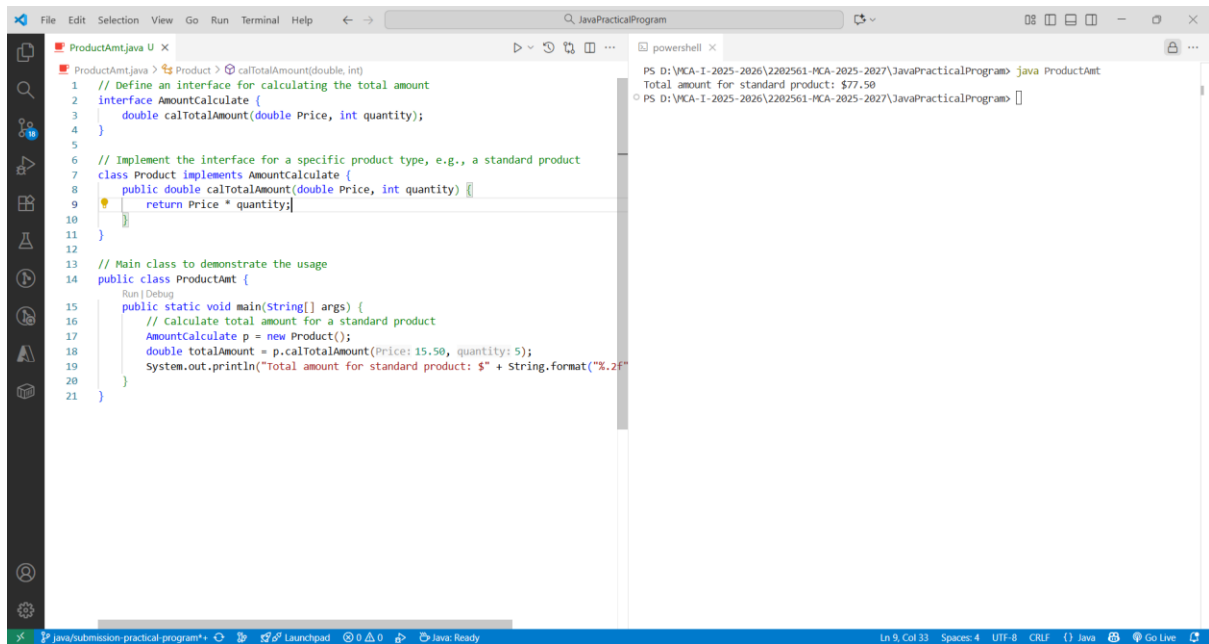
```
AreaCalculator.java > AreaCalculator
1 // Abstract class representing a generic Shape
2 abstract class Shape {
3     // Abstract method to calculate and print the area
4     public abstract void calculateArea();
5 }
6
7 // Concrete class for Rectangle
8 class Rectangle extends Shape {
9     private double length;
10    private double width;
11
12    public Rectangle(double length, double width) {
13        this.length = length;
14        this.width = width;
15    }
16
17    @Override
18    public void calculateArea() {
19        double area = length * width;
20        System.out.println("Area of Rectangle: " + area);
21    }
22 }
23
24 // Main class to test the program
25 public class AreaCalculator {
26     public static void main(String[] args) {
27         // Create instances of concrete shape classes
28         Rectangle rectangle = new Rectangle(length: 10, width: 5);
29         // Call the calculateArea method for each shape
30         rectangle.calculateArea();
31     }
32 }
```

```
PS D:\VKA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java AreaCalculator
Area of Rectangle: 50.0

PS D:\VKA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

Ln 12, Col 1 Spaces: 4 UTF-8 CRLF Java Go Live

### c. Interface program for calculating product amount



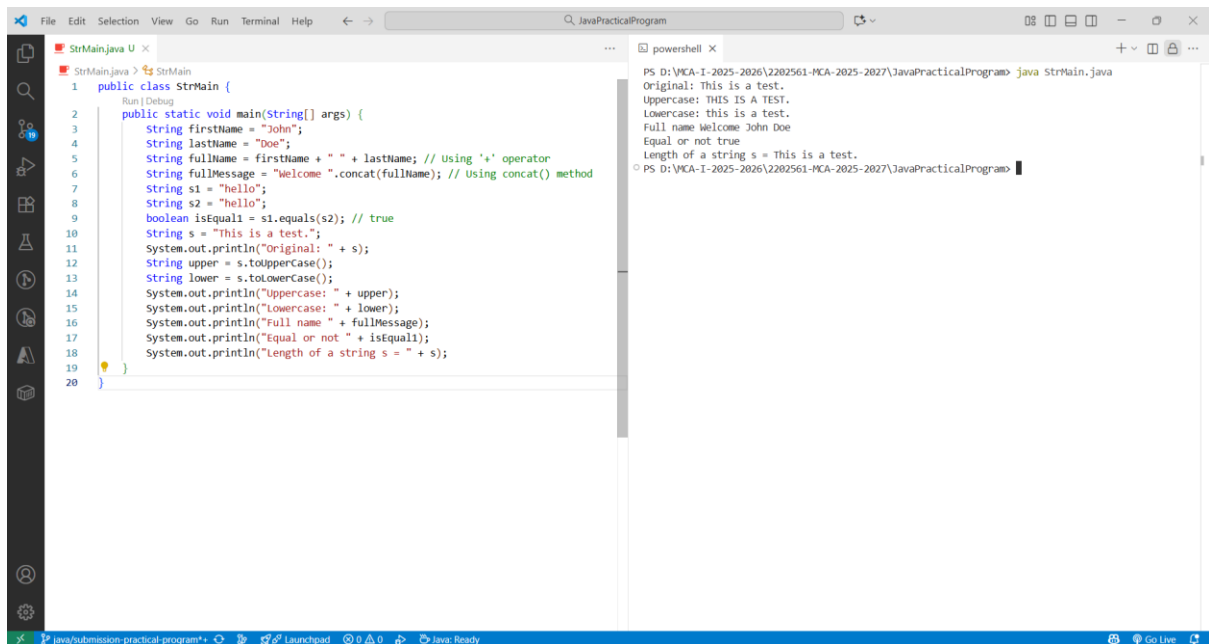
The screenshot shows an IDE with a Java file named `ProductAmt.java`. The code defines an interface `AmountCalculate` with a method `calTotalAmount(double Price, int quantity)`. A class `Product` implements this interface. A `main` method demonstrates the usage by creating a `Product` object and calling `calTotalAmount` with `Price: 15.50` and `quantity: 5`, resulting in a total amount of `$77.50`. The terminal output shows the command `java ProductAmt` and the corresponding output.

```
1 // Define an interface for calculating the total amount
2 interface AmountCalculate {
3     double calTotalAmount(double Price, int quantity);
4 }
5
6 // Implement the interface for a specific product type, e.g., a standard product
7 class Product implements AmountCalculate {
8     public double calTotalAmount(double Price, int quantity) {
9         return Price * quantity;
10    }
11 }
12
13 // Main class to demonstrate the usage
14 public class ProductAmt {
15     public static void main(String[] args) {
16         // Calculate total amount for a standard product
17         AmountCalculate p = new Product();
18         double totalAmount = p.calTotalAmount(Price: 15.50, quantity: 5);
19         System.out.println("Total amount for standard product: $" + String.format("%.2f", totalAmount));
20     }
21 }
```

```
PS D:\VICA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java ProductAmt
Total amount for standard product: $77.50
```

## 4: Introduction to string and its different operations

### Use of Different string methods

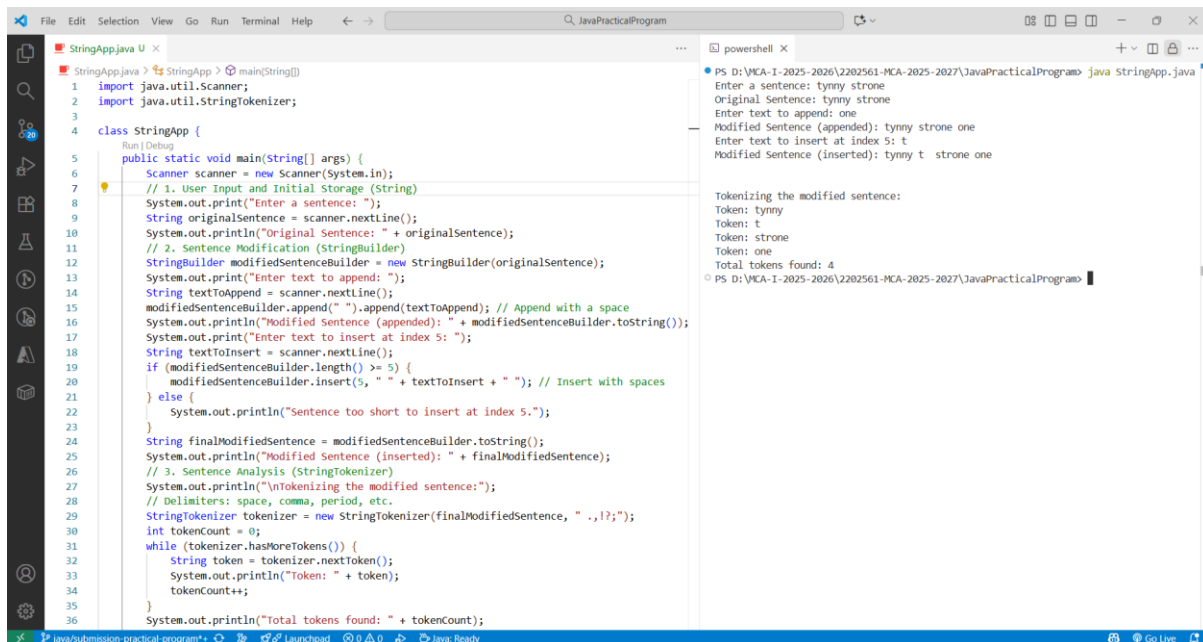


The screenshot shows an IDE with a Java file named `StrMain.java`. The code demonstrates various string operations including concatenation, comparison, and case conversion. The terminal output shows the command `java StrMain.java` and the corresponding output.

```
1 public class StrMain {
2     public static void main(String[] args) {
3         String firstName = "John";
4         String lastName = "Doe";
5         String fullName = firstName + " " + lastName; // Using '+' operator
6         String fullMessage = "Welcome ".concat(fullName); // Using concat() method
7         String s1 = "hello";
8         String s2 = "hello";
9         boolean isEqual = s1.equals(s2); // true
10        String s = "This is a test.";
11        System.out.println("Original: " + s);
12        String upper = s.toUpperCase();
13        String lower = s.toLowerCase();
14        System.out.println("Uppercase: " + upper);
15        System.out.println("Lowercase: " + lower);
16        System.out.println("Full name " + fullMessage);
17        System.out.println("Equal or not " + isEqual);
18        System.out.println("Length of a string s = " + s.length());
19    }
20 }
```

```
PS D:\VICA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java StrMain.java
Original: This is a test.
Uppercase: THIS IS A TEST.
Lowercase: this is a test.
Full name Welcome John Doe
Equal or not true
Length of a string s = This is a test.
PS D:\VICA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

## b. Design application using String, StringBuilder, StringTokenizer Experiment

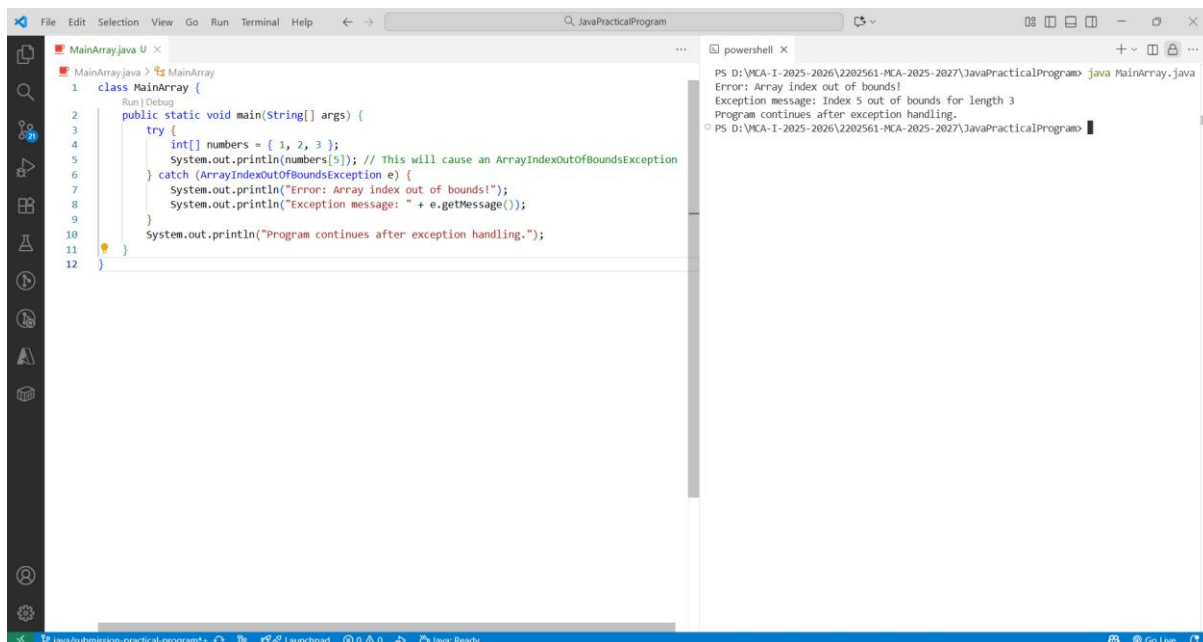


```
StringApp.java U X
StringApp.java > StringApp > main(String[])
1 import java.util.Scanner;
2 import java.util.StringTokenizer;
3
4 class StringApp {
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7         // 1. User Input and Initial Storage (String)
8         System.out.print("Enter a sentence: ");
9         String originalSentence = scanner.nextLine();
10        System.out.println("Original Sentence: " + originalSentence);
11        // 2. Sentence Modification (StringBuilder)
12        StringBuilder modifiedSentenceBuilder = new StringBuilder(originalSentence);
13        System.out.print("Enter text to append: ");
14        String textToAppend = scanner.nextLine();
15        modifiedSentenceBuilder.append(" ").append(textToAppend); // Append with a space
16        System.out.println("Modified Sentence (appended): " + modifiedSentenceBuilder.toString());
17        System.out.print("Enter text to insert at index 5: ");
18        String textToInsert = scanner.nextLine();
19        if (modifiedSentenceBuilder.length() >= 5) {
20            modifiedSentenceBuilder.insert(5, " " + textToInsert + " "); // Insert with spaces
21        } else {
22            System.out.println("Sentence too short to insert at index 5.");
23        }
24        String finalModifiedSentence = modifiedSentenceBuilder.toString();
25        System.out.println("Modified Sentence (inserted): " + finalModifiedSentence);
26        // 3. Sentence Analysis (StringTokenizer)
27        System.out.println("\nTokenizing the modified sentence:");
28        // Delimiters: space, comma, period, etc.
29        StringTokenizer tokenizer = new StringTokenizer(finalModifiedSentence, " ,!;");
30        int tokenCount = 0;
31        while (tokenizer.hasMoreTokens()) {
32            String token = tokenizer.nextToken();
33            System.out.println("Token: " + token);
34            tokenCount++;
35        }
36        System.out.println("Total tokens found: " + tokenCount);
37    }
38}
```

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java StringApp.java
Enter a sentence: tynny strone
Original Sentence: tynny strone
Enter text to append: one
Modified Sentence (appended): tynny strone one
Enter text to insert at index 5: t
Modified Sentence (inserted): tynny t strone one

Tokenizing the modified sentence:
Token: tynny
Token: t
Token: strone
Token: one
Total tokens found: 4
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

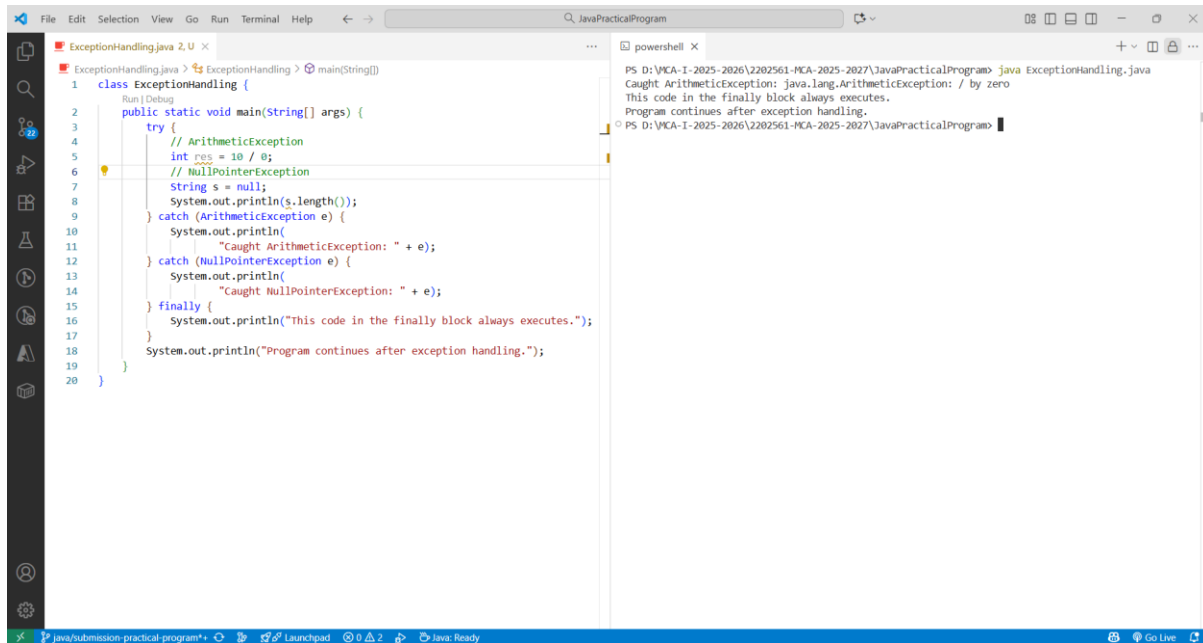
## 5: Exception Handling -Test any five of standard exception and user Defined Custom Exceptions in java



```
MainArray.java U X
MainArray.java > MainArray
1 class MainArray {
2     public static void main(String[] args) {
3         try {
4             int[] numbers = { 1, 2, 3 };
5             System.out.println(numbers[5]); // This will cause an ArrayIndexOutOfBoundsException
6         } catch (ArrayIndexOutOfBoundsException e) {
7             System.out.println("Error: Array index out of bounds!");
8             System.out.println("Exception message: " + e.getMessage());
9         }
10        System.out.println("Program continues after exception handling.");
11    }
12}
```

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java MainArray.java
Error: Array index out of bounds!
Exception message: Index 5 out of bounds for length 3
Program continues after exception handling.
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

### c. Try-catch-finally block with multiple catch statements.



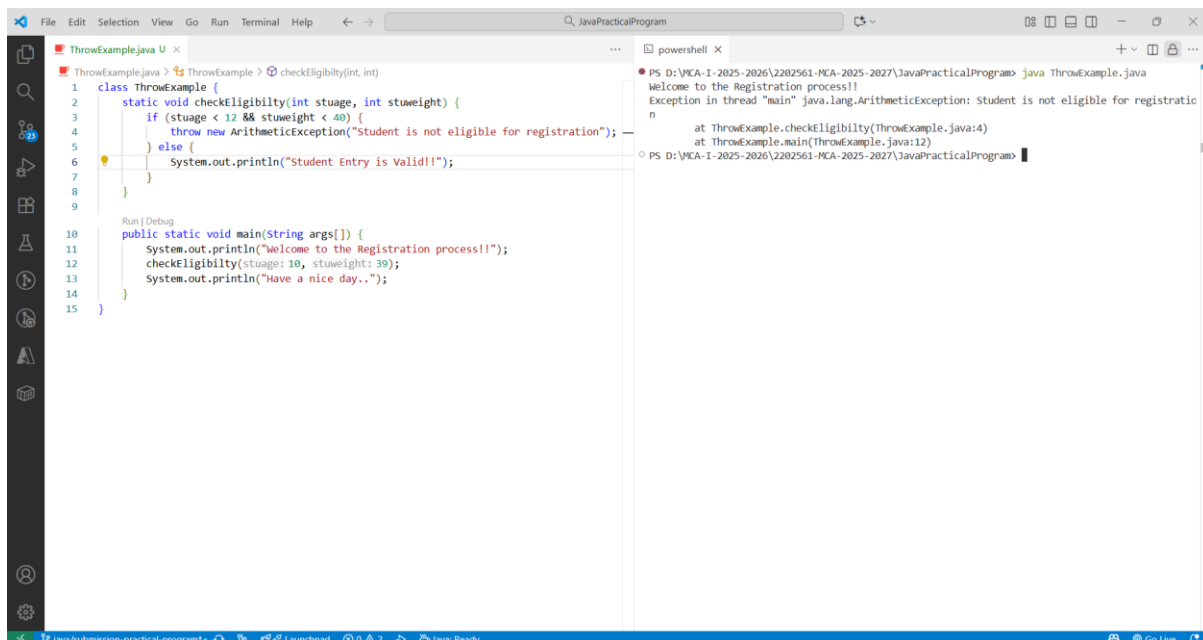
The screenshot shows an IDE with a Java file named `ExceptionHandling.java`. The code defines a class `ExceptionHandling` with a `main` method. The `main` method contains a try-catch-finally block. The try block calculates `10 / 0` and prints the length of a null string. The first catch block catches `ArithmeticException` and prints a message. The second catch block catches `NullPointerException` and prints a message. The finally block prints a message that always executes. The output window shows the execution results.

```
1 class ExceptionHandling {
2     public static void main(String[] args) {
3         try {
4             // ArithmeticException
5             int res = 10 / 0;
6             // NullPointerException
7             String s = null;
8             System.out.println(s.length());
9         } catch (ArithmeticException e) {
10            System.out.println(
11                "Caught ArithmeticException: " + e);
12        } catch (NullPointerException e) {
13            System.out.println(
14                "Caught NullPointerException: " + e);
15        } finally {
16            System.out.println("This code in the finally block always executes.");
17        }
18        System.out.println("Program continues after exception handling.");
19    }
20 }
```

Output:

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java ExceptionHandling.java
Caught ArithmeticException: java.lang.ArithmeticException: / by zero
This code in the finally block always executes.
Program continues after exception handling.
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

### d. User-defined exceptions



The screenshot shows an IDE with a Java file named `ThrowExample.java`. The code defines a class `ThrowExample` with a `checkEligibility` method that throws a `new ArithmeticException("Student is not eligible for registration")` if the student's weight is less than 40. The `main` method calls `checkEligibility` with `stuage: 10, stuweight: 30` and prints the result. The output window shows the execution results.

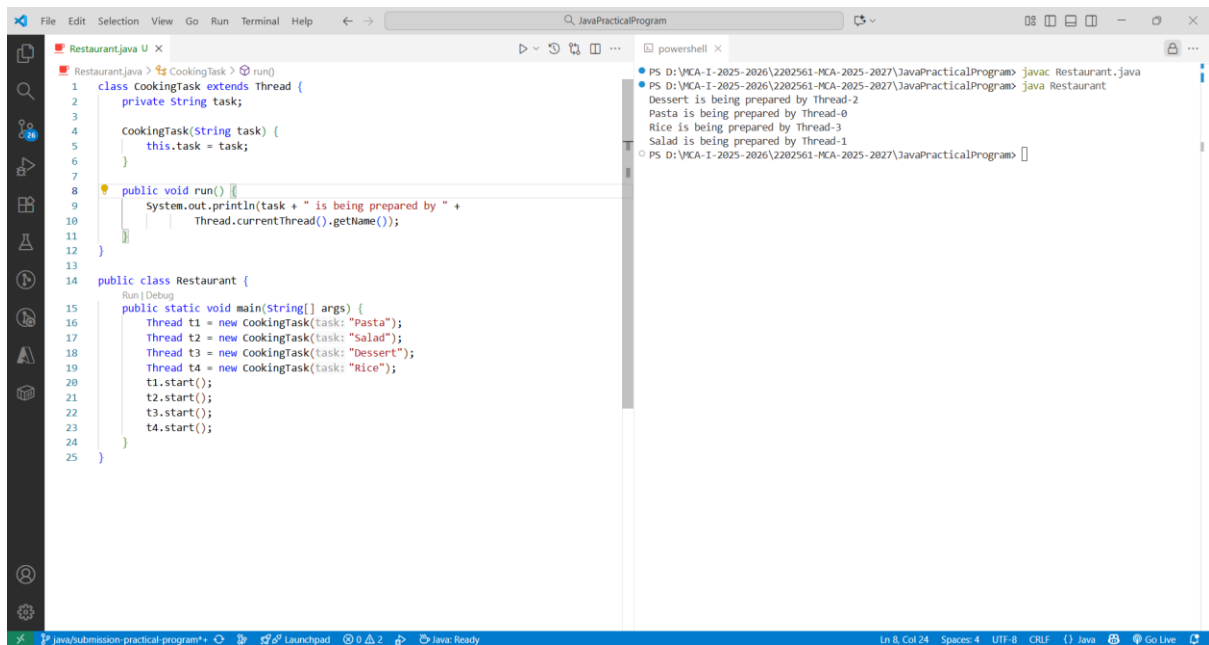
```
1 class ThrowExample {
2     static void checkEligibility(int stuage, int stuweight) {
3         if (stuage < 12 && stuweight < 40) {
4             throw new ArithmeticException("Student is not eligible for registration");
5         } else {
6             System.out.println("Student Entry is Valid!!");
7         }
8     }
9 }
10 public static void main(String args[]) {
11     System.out.println("Welcome to the Registration process!!");
12     checkEligibility(stuage: 10, stuweight: 30);
13     System.out.println("Have a nice day..");
14 }
15 }
```

Output:

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java ThrowExample.java
Welcome to the Registration process!!
Exception in thread "main" java.lang.ArithmeticException: Student is not eligible for registration
    at ThrowExample.checkEligibility(ThrowExample.java:4)
    at ThrowExample.main(ThrowExample.java:12)
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

## Experiment 6: Multithreading

### Implement a Thread - 1. Extending the Thread class



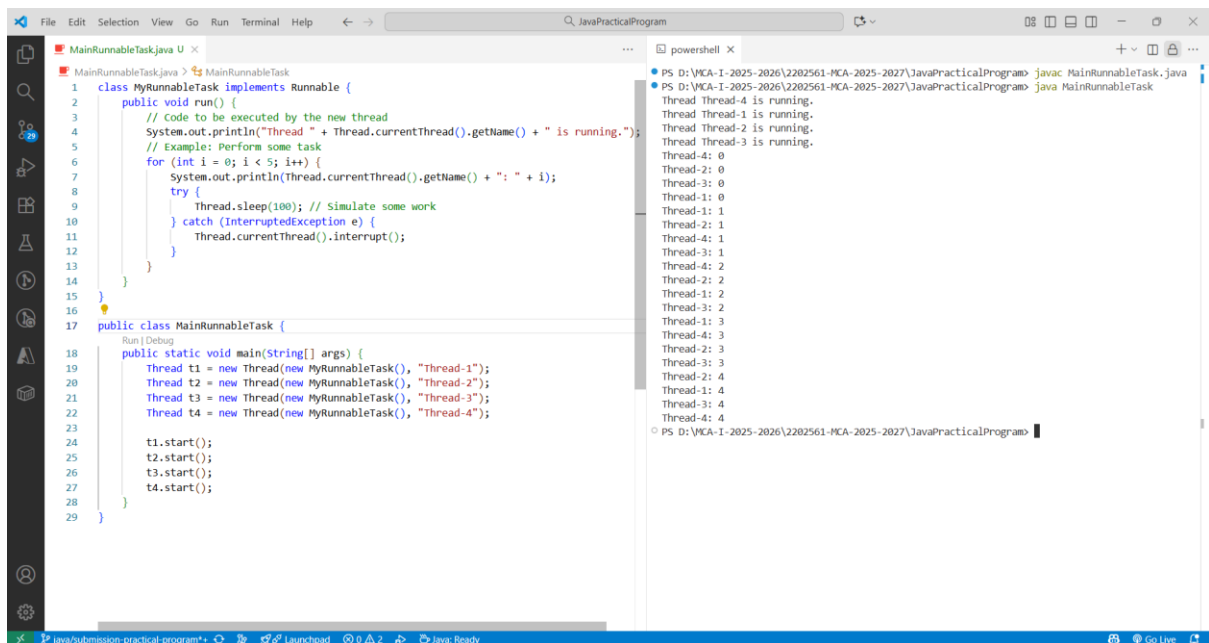
The screenshot shows an IDE with a Java file named `Restaurant.java`. The code defines a `CookingTask` class that extends `Thread` and implements the `run()` method. The `run()` method prints the task name and the current thread name. The `Restaurant` class has a `main` method that creates four `CookingTask` objects (Pasta, Salad, Dessert, Rice) and starts them. The terminal output shows the execution of the program, with each task being prepared by a different thread (Thread-0, Thread-1, Thread-2, Thread-3).

```
1 class CookingTask extends Thread {
2     private String task;
3
4     CookingTask(String task) {
5         this.task = task;
6     }
7
8     public void run() {
9         System.out.println(task + " is being prepared by " +
10             Thread.currentThread().getName());
11     }
12 }
13
14 public class Restaurant {
15     public static void main(String[] args) {
16         Thread t1 = new CookingTask(task: "Pasta");
17         Thread t2 = new CookingTask(task: "Salad");
18         Thread t3 = new CookingTask(task: "Dessert");
19         Thread t4 = new CookingTask(task: "Rice");
20         t1.start();
21         t2.start();
22         t3.start();
23         t4.start();
24     }
25 }
```

Terminal Output:

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> javac Restaurant.java
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java Restaurant
Dessert is being prepared by Thread-2
Pasta is being prepared by Thread-0
Rice is being prepared by Thread-3
Salad is being prepared by Thread-1
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

### b. Implement Thread using the Runnable Interface



The screenshot shows an IDE with a Java file named `MainRunnableTask.java`. The code defines a `MyRunnableTask` class that implements the `Runnable` interface and implements the `run()` method. The `run()` method prints the thread name and the current thread name, and then simulates some work by sleeping for 100ms. The `MainRunnableTask` class has a `main` method that creates four `Thread` objects (Thread-1, Thread-2, Thread-3, Thread-4) and starts them. The terminal output shows the execution of the program, with each thread running and printing its name and the current thread name.

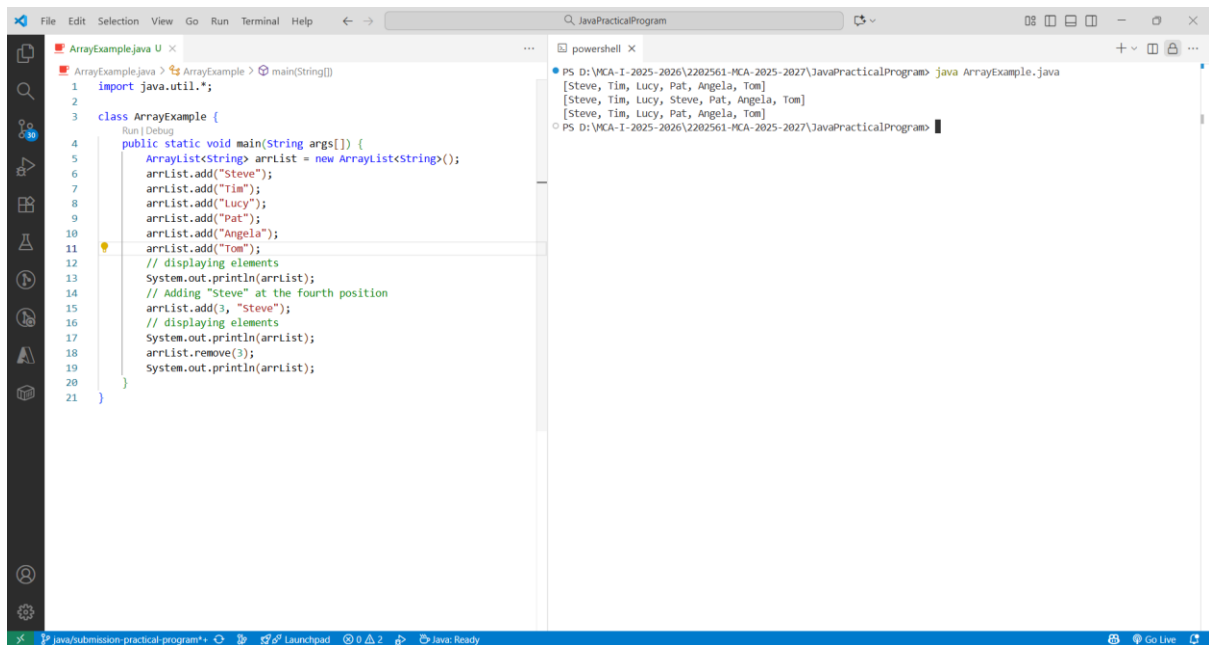
```
1 class MyRunnableTask implements Runnable {
2     public void run() {
3         // Code to be executed by the new thread
4         System.out.println("Thread " + Thread.currentThread().getName() + " is running.");
5         // Example: Perform some task
6         for (int i = 0; i < 5; i++) {
7             System.out.println(Thread.currentThread().getName() + ": " + i);
8             try {
9                 Thread.sleep(100); // Simulate some work
10            } catch (InterruptedException e) {
11                Thread.currentThread().interrupt();
12            }
13        }
14    }
15 }
16
17 public class MainRunnableTask {
18     public static void main(String[] args) {
19         Thread t1 = new Thread(new MyRunnableTask(), "Thread-1");
20         Thread t2 = new Thread(new MyRunnableTask(), "Thread-2");
21         Thread t3 = new Thread(new MyRunnableTask(), "Thread-3");
22         Thread t4 = new Thread(new MyRunnableTask(), "Thread-4");
23
24         t1.start();
25         t2.start();
26         t3.start();
27         t4.start();
28     }
29 }
```

Terminal Output:

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> javac MainRunnableTask.java
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java MainRunnableTask
Thread Thread-4 is running.
Thread Thread-1 is running.
Thread Thread-2 is running.
Thread Thread-3 is running.
Thread-4: 0
Thread-2: 0
Thread-3: 0
Thread-1: 0
Thread-1: 1
Thread-2: 1
Thread-4: 1
Thread-3: 1
Thread-4: 2
Thread-2: 2
Thread-1: 2
Thread-3: 2
Thread-1: 3
Thread-4: 3
Thread-2: 3
Thread-3: 3
Thread-2: 4
Thread-1: 4
Thread-3: 4
Thread-4: 4
Thread-4: 4
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

## Experiment 7: Collections

### Add and remove element using ArrayList

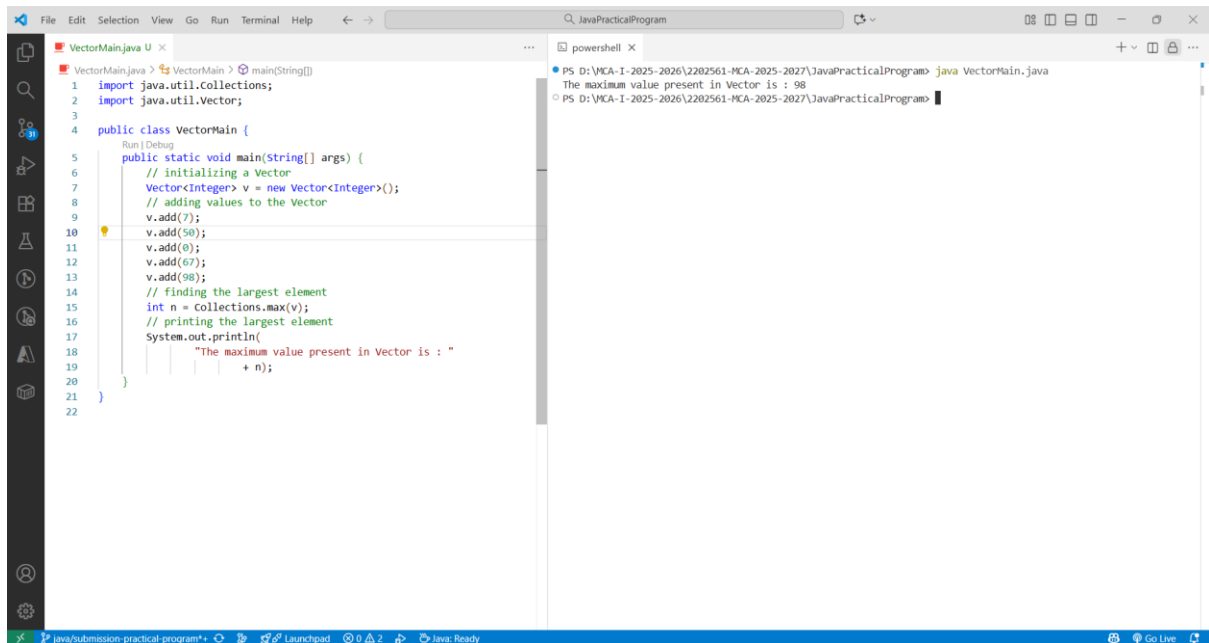


The screenshot shows an IDE with a Java file named `ArrayExample.java` and a PowerShell terminal window. The Java code defines a class `ArrayExample` with a `main` method that creates an `ArrayList`, adds several names, prints the list, adds "Steve" at index 3, prints the list again, removes the element at index 3, and prints the list a third time. The PowerShell terminal shows the output of running `java ArrayExample.java`, which displays the list at each step: `[Steve, Tim, Lucy, Pat, Angela, Tom]`, `[Steve, Tim, Lucy, Steve, Pat, Angela, Tom]`, and `[Steve, Tim, Lucy, Pat, Angela, Tom]`.

```
ArrayExample.java > ArrayExample > main(String[])
1 import java.util.*;
2
3 class ArrayExample {
4     public static void main(String args[]) {
5         ArrayList<String> arrList = new ArrayList<String>();
6         arrList.add("Steve");
7         arrList.add("Tim");
8         arrList.add("Lucy");
9         arrList.add("Pat");
10        arrList.add("Angela");
11        arrList.add("Tom");
12        // displaying elements
13        System.out.println(arrList);
14        // Adding "Steve" at the fourth position
15        arrList.add(3, "Steve");
16        // displaying elements
17        System.out.println(arrList);
18        arrList.remove(3);
19        System.out.println(arrList);
20    }
21 }
```

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java ArrayExample.java
[Steve, Tim, Lucy, Pat, Angela, Tom]
[Steve, Tim, Lucy, Steve, Pat, Angela, Tom]
[Steve, Tim, Lucy, Pat, Angela, Tom]
```

### b. To find maximum element vector using predefined method



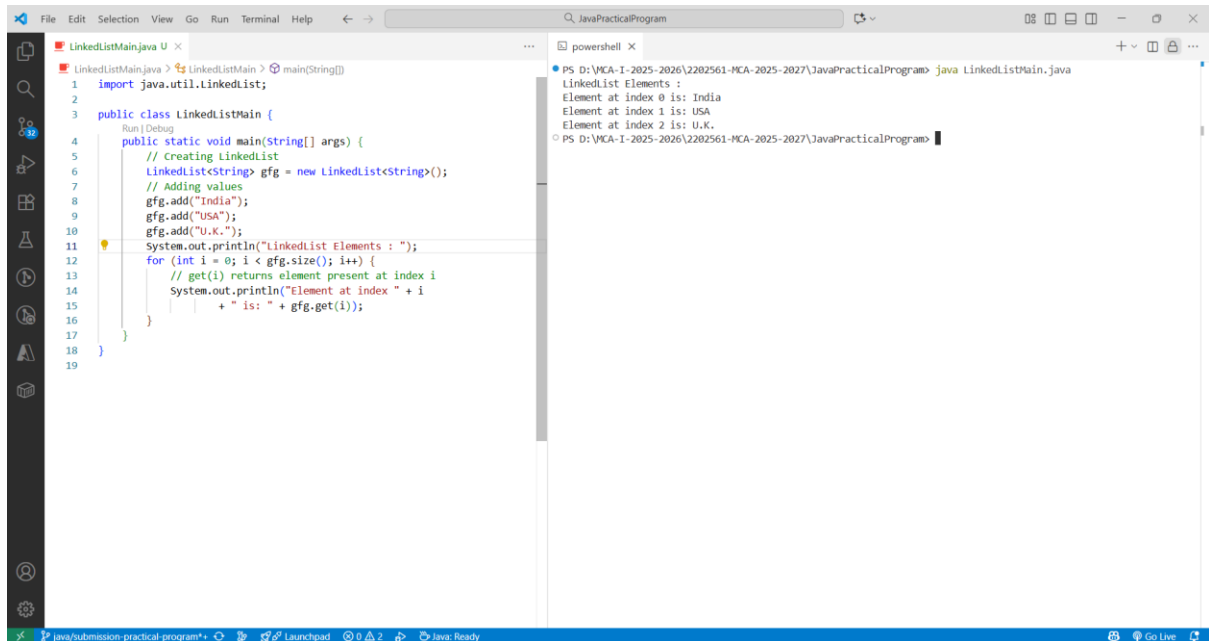
The screenshot shows an IDE with a Java file named `VectorMain.java` and a PowerShell terminal window. The Java code defines a class `VectorMain` with a `main` method that initializes a `Vector`, adds several integers, finds the maximum value using `Collections.max()`, and prints it. The PowerShell terminal shows the output of running `java VectorMain.java`, which displays: `The maximum value present in Vector is : 98`.

```
VectorMain.java > VectorMain > main(String[])
1 import java.util.Collections;
2 import java.util.Vector;
3
4 public class VectorMain {
5     public static void main(String[] args) {
6         // initializing a Vector
7         Vector<Integer> v = new Vector<Integer>();
8         // adding values to the Vector
9         v.add(7);
10        v.add(50);
11        v.add(0);
12        v.add(67);
13        v.add(98);
14        // finding the largest element
15        int n = Collections.max(v);
16        // printing the largest element
17        System.out.println(
18            "The maximum value present in Vector is : "
19            + n);
20    }
21 }
22 }
```

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java VectorMain.java
The maximum value present in Vector is : 98
```



### c. to get the elements of LinkedList



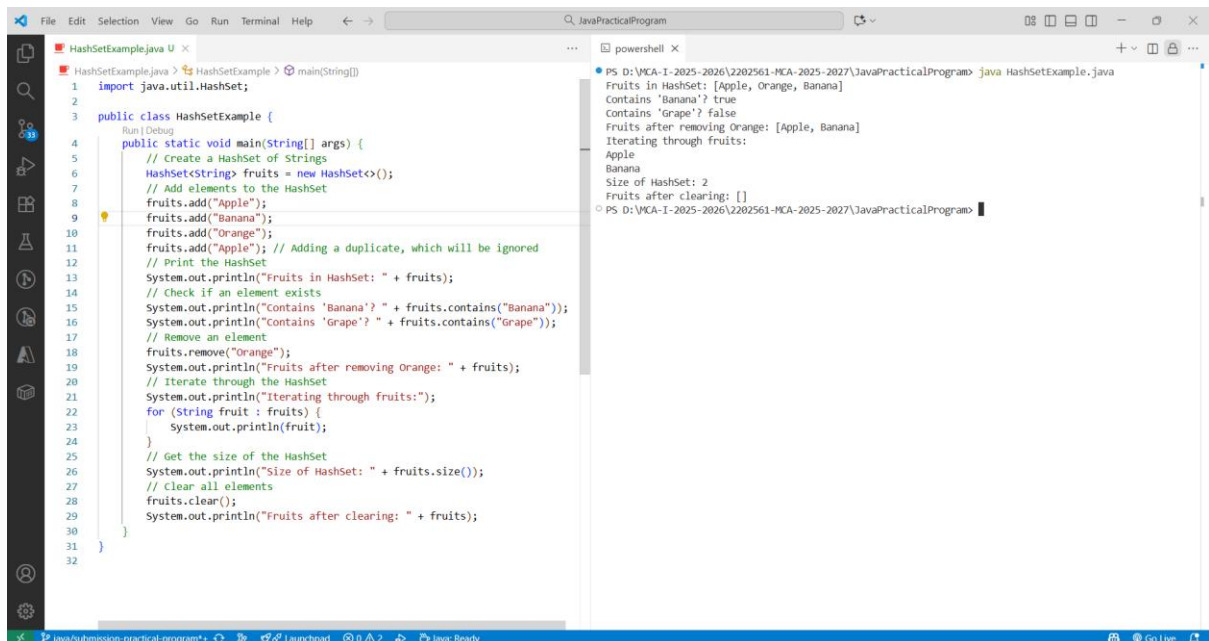
The screenshot shows an IDE with a Java file named `LinkedListMain.java` and a PowerShell terminal window. The Java code creates a `LinkedList` and adds the elements "India", "USA", and "U.K.". It then iterates through the list and prints each element with its index.

```
1 import java.util.LinkedList;
2
3 public class LinkedListMain {
4     public static void main(String[] args) {
5         // Creating LinkedList
6         LinkedList<String> gfg = new LinkedList<String>();
7         // Adding values
8         gfg.add("India");
9         gfg.add("USA");
10        gfg.add("U.K.");
11        System.out.println("LinkedList Elements : ");
12        for (int i = 0; i < gfg.size(); i++) {
13            // get(i) returns element present at index i
14            System.out.println("Element at index " + i
15                               + " is: " + gfg.get(i));
16        }
17    }
18 }
19
```

The PowerShell terminal output shows the execution of the program:

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java LinkedListMain.java
LinkedList Elements :
Element at index 0 is: India
Element at index 1 is: USA
Element at index 2 is: U.K.
```

### d. to get the elements of HashSet



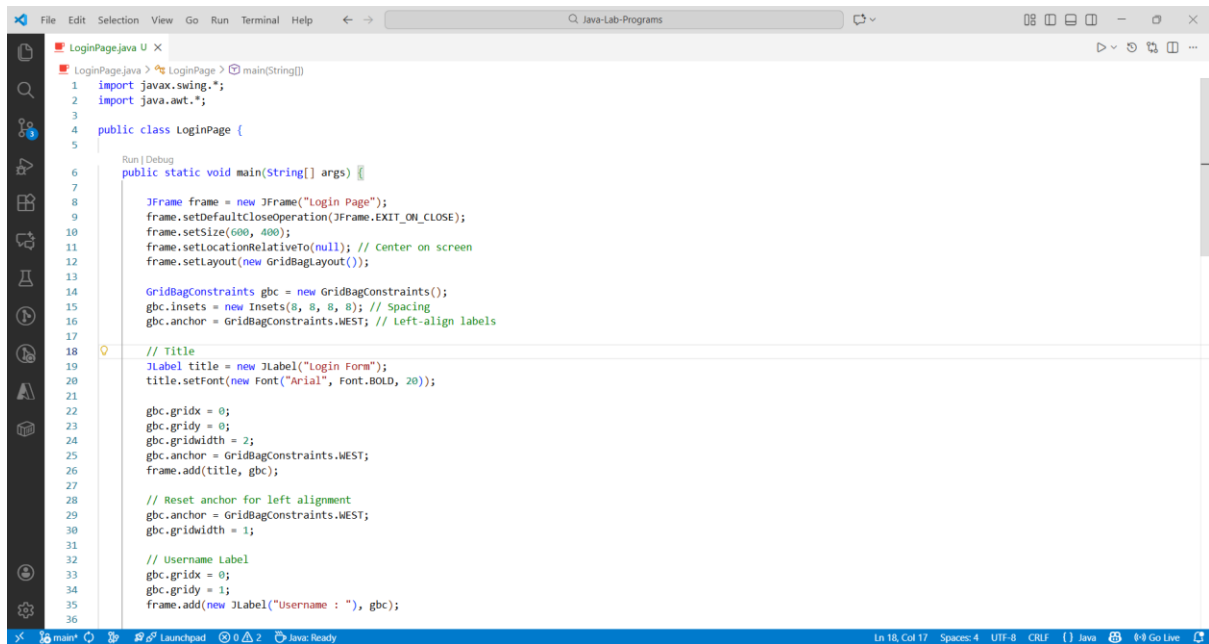
The screenshot shows an IDE with a Java file named `HashSetExample.java` and a PowerShell terminal window. The Java code creates a `HashSet` and adds the elements "Apple", "Banana", and "Orange". It then prints the set, checks for the presence of "Banana" and "Grape", removes "Orange", iterates through the remaining elements, prints the size, clears the set, and prints the elements after clearing.

```
1 import java.util.HashSet;
2
3 public class HashSetExample {
4     public static void main(String[] args) {
5         // Create a HashSet of Strings
6         HashSet<String> fruits = new HashSet<>();
7         // Add elements to the HashSet
8         fruits.add("Apple");
9         fruits.add("Banana");
10        fruits.add("Orange");
11        fruits.add("Apple"); // Adding a duplicate, which will be ignored
12        // Print the HashSet
13        System.out.println("Fruits in HashSet: " + fruits);
14        // Check if an element exists
15        System.out.println("Contains 'Banana'? " + fruits.contains("Banana"));
16        System.out.println("Contains 'Grape'? " + fruits.contains("Grape"));
17        // Remove an element
18        fruits.remove("Orange");
19        System.out.println("Fruits after removing Orange: " + fruits);
20        // Iterate through the HashSet
21        System.out.println("Iterating through fruits:");
22        for (String fruit : fruits) {
23            System.out.println(fruit);
24        }
25        // Get the size of the HashSet
26        System.out.println("Size of HashSet: " + fruits.size());
27        // Clear all elements
28        fruits.clear();
29        System.out.println("Fruits after clearing: " + fruits);
30    }
31 }
32
```

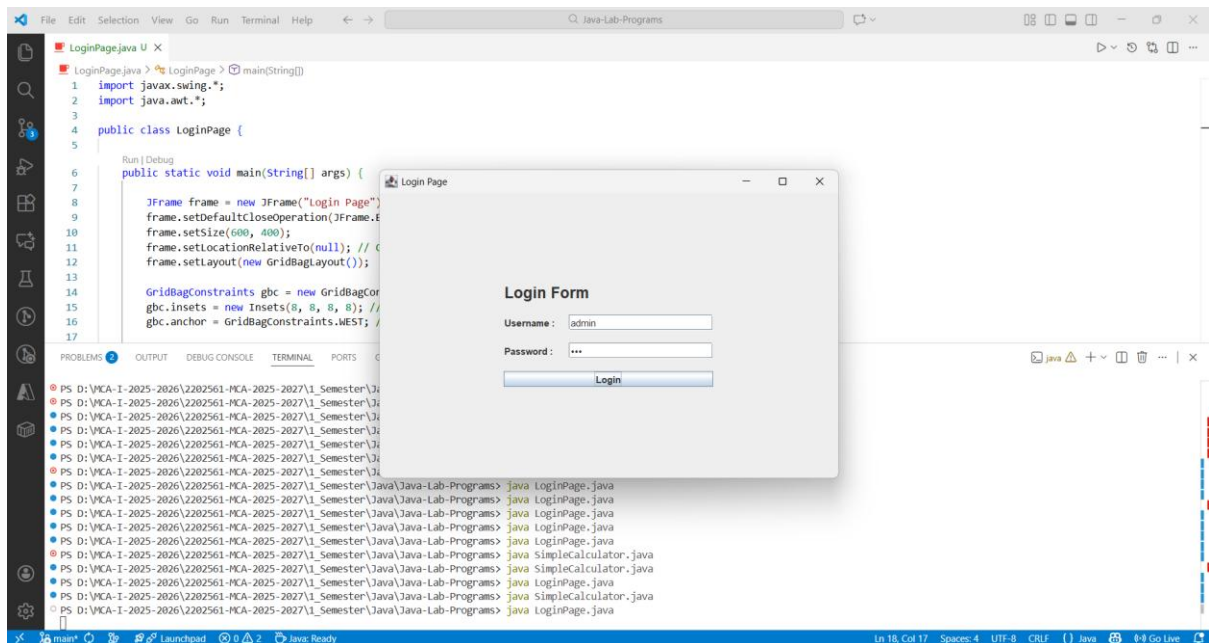
The PowerShell terminal output shows the execution of the program:

```
PS D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> java HashSetExample.java
Fruits in HashSet: [Apple, Orange, Banana]
Contains 'Banana'? true
Contains 'Grape'? false
Fruits after removing Orange: [Apple, Banana]
Iterating through fruits:
Apple
Banana
Size of HashSet: 2
Fruits after clearing: []
```

## Login form using SWING



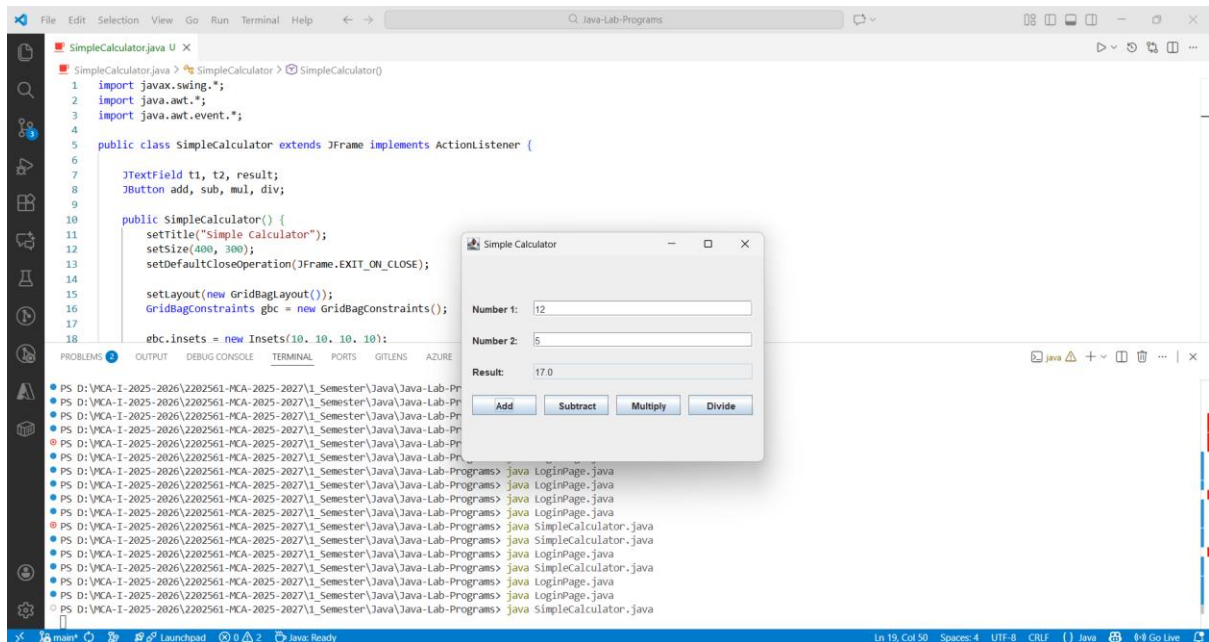
```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class LoginPage {
5
6     public static void main(String[] args) {
7
8         JFrame frame = new JFrame("Login Page");
9         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        frame.setSize(600, 400);
11        frame.setLocationRelativeTo(null); // Center on screen
12        frame.setLayout(new GridBagLayout());
13
14        GridBagConstraints gbc = new GridBagConstraints();
15        gbc.insets = new Insets(8, 8, 8, 8); // Spacing
16        gbc.anchor = GridBagConstraints.WEST; // Left-align labels
17
18        // Title
19        JLabel title = new JLabel("Login Form");
20        title.setFont(new Font("Arial", Font.BOLD, 20));
21
22        gbc.gridx = 0;
23        gbc.gridy = 0;
24        gbc.gridwidth = 2;
25        gbc.anchor = GridBagConstraints.WEST;
26        frame.add(title, gbc);
27
28        // Reset anchor for left alignment
29        gbc.anchor = GridBagConstraints.WEST;
30        gbc.gridwidth = 1;
31
32        // Username Label
33        gbc.gridx = 0;
34        gbc.gridy = 1;
35        frame.add(new JLabel("Username : ", gbc);
36    }
```



## Simple calculator using SWING



```
SimpleCalculator.java X
SimpleCalculator.java > SimpleCalculator
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class SimpleCalculator extends JFrame implements ActionListener {
6
7     JTextField t1, t2, result;
8     JButton add, sub, mul, div;
9
10    public SimpleCalculator() {
11        setTitle("Simple Calculator");
12        setSize(400, 300);
13        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14
15        setLayout(new GridBagLayout());
16        GridBagConstraints gbc = new GridBagConstraints();
17
18        gbc.insets = new Insets(10, 10, 10, 10);
19        gbc.fill = GridBagConstraints.HORIZONTAL;
20
21        // Number 1 label
22        gbc.gridx = 0;
23        gbc.gridy = 0;
24        add(new JLabel("Number 1:"), gbc);
25
26        // Number 1 text
27        t1 = new JTextField(15);
28        gbc.gridx = 1;
29        add(t1, gbc);
30
31        // Number 2 label
32        gbc.gridx = 0;
33        gbc.gridy = 1;
34        add(new JLabel("Number 2:"), gbc);
35
36        // Number 2 text
37        t2 = new JTextField(15);
```



```
SimpleCalculator.java X
SimpleCalculator.java > SimpleCalculator > SimpleCalculator()
1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 public class SimpleCalculator extends JFrame implements ActionListener {
6
7     JTextField t1, t2, result;
8     JButton add, sub, mul, div;
9
10    public SimpleCalculator() {
11        setTitle("Simple Calculator");
12        setSize(400, 300);
13        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14
15        setLayout(new GridBagLayout());
16        GridBagConstraints gbc = new GridBagConstraints();
17
18        gbc.insets = new Insets(10, 10, 10, 10);
```

Simple Calculator

Number 1: 12

Number 2: 5

Result: 17.0

Add Subtract Multiply Divide