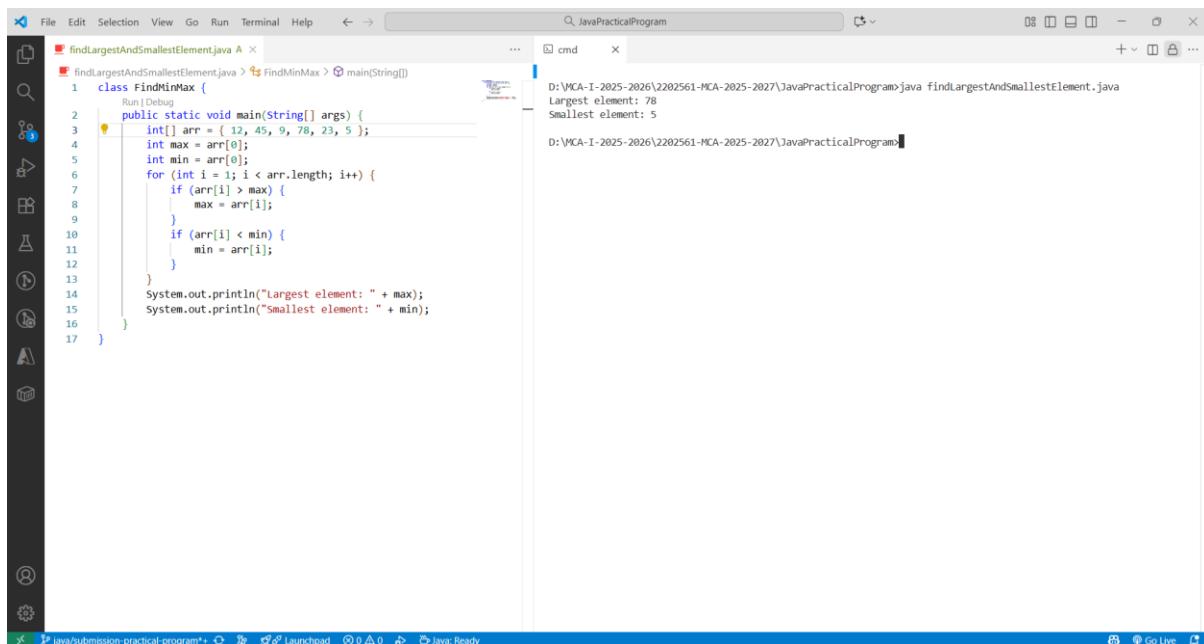


Name	Udhayarajan J
Class	MCA - I
Sub	Java Programming
Type	Experiment Practical Program
Date	2025-11-12
Roll No	2202561

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 a Java development environment with two tabs open: 'findLargestAndSmallestElement.java' and 'cmd'. The code in 'findLargestAndSmallestElement.java' is as follows:

```

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 }

```

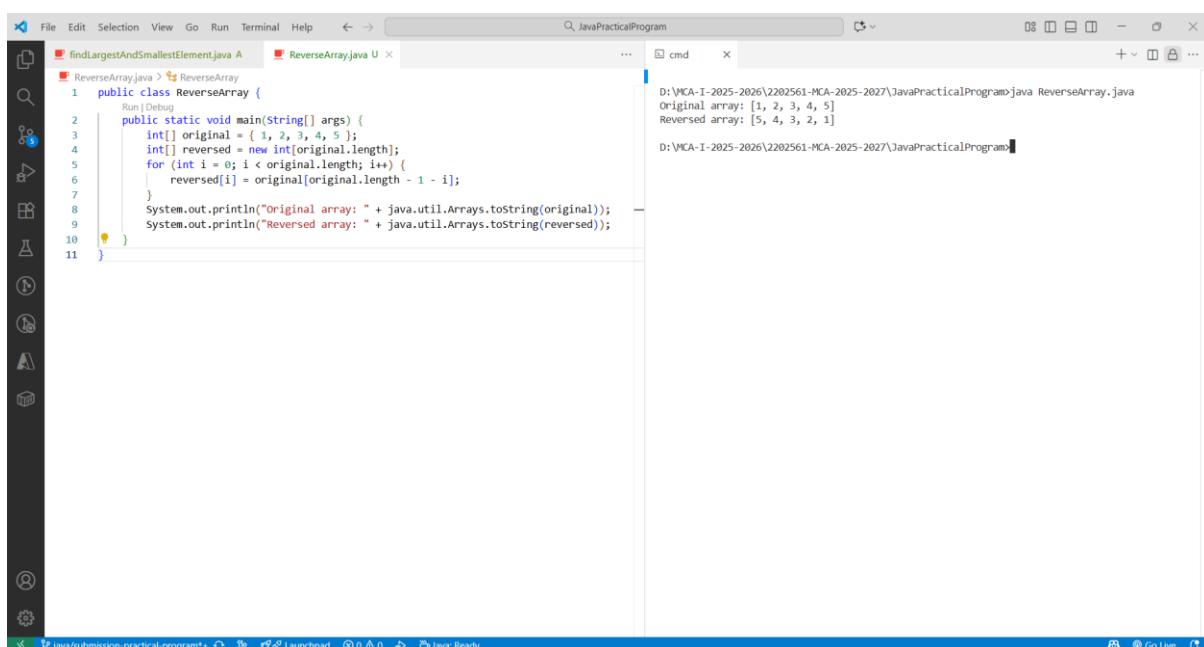
The 'cmd' tab shows the output of running the program with the command 'java findLargestAndSmallestElement.java'. The output is:

```

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

```

b. Reversing an Array



The screenshot shows a Java development environment with two tabs open: 'ReverseArray.java' and 'cmd'. The code in 'ReverseArray.java' is as follows:

```

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 }

```

The 'cmd' tab shows the output of running the program with the command 'java ReverseArray.java'. The output 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]

```

c. Sum of all elements in a 2D Array

The screenshot shows a Java IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** JavaPracticalProgram.
- Project Explorer:** Shows three files: findLargestAndSmallestElement.java, ReverseArray.java, and SumOf2DArrayElements.java.
- Code Editor:** Displays the code for `SumOf2DArrayElements`. The code initializes a 2D array with values [10, 20, 30] and [40, 50, 60]. It then iterates through the array to calculate the sum of all elements, which is printed as 210.
- Terminal:** cmd window showing the command `D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java SumOf2DArrayElements . java` and the output `Sum of all elements: 210`.
- Bottom Status Bar:** Shows the current file as `java/submission-practical program*`, Launchpad, Java Ready, and Go Live.

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

The screenshot shows a Java IDE interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Search Bar:** JavaPracticalProgram.
- Project Explorer:** Shows four files: findLargestAndSmallestElement.java, ReverseArray.java, SumOf2DArrayElements.java, and SumEachRow2D.java.
- Code Editor:** Displays the code for `SumEachRow2D`. The code initializes a 2D array with values [10, 20, 30], [40, 50, 60], and [70, 80, 90]. It then iterates through the array to calculate the sum of each row, printing the results as 60, 150, and 240 respectively.
- Terminal:** cmd window showing the command `D:\MCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>java SumEachRow2D . java` and the output `Sum of row 0: 60`, `Sum of row 1: 150`, `Sum of row 2: 240`.
- Bottom Status Bar:** Shows the current file as `java/submission-practical program*`, Launchpad, Java Ready, and Go Live.

3: Package

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

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

MathUtils.java
MyMathFunctions > 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 GITLENS AZURE
cmd + ⌘ ⌘ ⌘ Go Live
D:\VCA-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:\VCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

b. abstract class Experiment for calculating area

```
AreaCalculator.java
AreaCalculator > 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     Run | Debug
27     public static void main(String[] args) {
28         // Create instances of concrete shape classes
29         Rectangle rectangle = new Rectangle(length: 10, width: 5);
30         // Call the calculateArea method for each shape
31         rectangle.calculateArea();
32     }
}

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

c. Interface program for calculating product amount

The screenshot shows a Java development environment with the following details:

- ProductAmt.java** is the active file, containing Java code for an interface and its implementation.
- powershell** window shows the command `java ProductAmt` and its output: "Total amount for standard product: \$77.50".
- Bottom status bar indicates the file is 21 lines long, has 4 spaces per tab, and is in UTF-8 encoding.

```
ProductAmt.java U x
ProductAmt.java > Product > calTotalAmount(double)
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     Run | Debug
16     public static void main(String[] args) {
17         // calculate total amount for a standard product
18         AmountCalculate p = new Product();
19         double totalAmount = p.calTotalAmount(Price: 15.50, quantity: 5);
20         System.out.println("Total amount for standard product: $" + String.format("%.2f"
21     }

```

4: Introduction to string and its different operations

Use of Different string methods

The screenshot shows a Java development environment with the following details:

- StrMain.java** is the active file, demonstrating various string manipulation methods.
- powershell** window shows the command `java StrMain.java` and its output, including concatenation, equals(), toUpperCase(), toLowerCase(), and length() examples.
- Bottom status bar indicates the file is 20 lines long, has 4 spaces per tab, and is in UTF-8 encoding.

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

```

b. Design application using String, StringBuilder, StringTokenizer Experiment

The screenshot shows a Java development environment with a code editor and a terminal window.

Code Editor: The code editor displays `StringApp.java` which uses `Scanner` and `StringBuilder` to process user input and modify sentences. It includes sections for user input, sentence modification using `StringBuilder`, and sentence analysis using `StringTokenizer`.

```
StringApp.java
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("Tokenizing 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 }
```

Terminal Window: The terminal window shows the execution of `StringApp.java` and its output. It prompts for a sentence ("tynny strone"), prints the original sentence, appends "one", inserts "t" at index 5, prints the final modified sentence ("tynny t strone one"), and then tokenizes the sentence into tokens: "tynny", "t", "strone", and "one".

```
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
```

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

The screenshot shows a Java development environment with a code editor and a terminal window.

Code Editor: The code editor displays `MainArray.java` which contains a `main` method that attempts to access an array element at index 5, causing an `ArrayIndexOutOfBoundsException`. It catches this exception and prints the error message.

```
MainArray.java
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 }
```

Terminal Window: The terminal window shows the execution of `MainArray.java`. It prints the error message "Error: Array index out of bounds!" and the exception message "Exception message: Index 5 out of bounds for length 3". It then continues to print "Program continues after exception handling."

```
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.
```

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

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the following Java code:

```
ExceptionHandling.java 2, U
ExceptionHandling.java > ExceptionHandling > main(String[])
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 (ArithmaticException e) {
10            System.out.println(
11                "Caught ArithmaticException: " + 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    }
19 }
20 }
```

The terminal window shows the output of running the program:

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

d. User-defined exceptions

The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the following Java code:

```
ThrowExample.java 2, U
ThrowExample.java > ThrowExample > checkEligibility(int, int)
1 class ThrowExample {
2     static void checkEligibility(int stuage, int stuweight) {
3         if (stuage < 12 && stuweight < 40) {
4             throw new ArithmaticException("Student is not eligible for registration");
5         } else {
6             System.out.println("Student Entry is Valid!!!");
7         }
8     }
9
Run | Debug
10 public static void main(String args[]) {
11     System.out.println("Welcome to the Registration process!!!");
12     checkEligibility(stuage: 10, stuweight: 39);
13     System.out.println("Have a nice day..");
14 }
15 }
```

The terminal window shows the output of running the program:

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

Experiment 6: Multithreading

Implement a Thread - 1. Extending the Thread class

The screenshot shows a Java IDE interface with two panes. The left pane displays the source code for `Restaurant.java`. The right pane shows a terminal window titled "powershell" with the command "java Restaurant" running. The terminal output shows four threads (t1-t4) each printing a message indicating they are preparing a specific dish.

```
Restaurant.java U ×
Restaurant.java > CookingTask > run()
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 }
```

```
PS D:\VCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram> javac Restaurant.java
PS D:\VCA-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:\VCA-I-2025-2026\2202561-MCA-2025-2027\JavaPracticalProgram>
```

b. Implement Thread using the Runnable Interface

The screenshot shows a Java IDE interface with two panes. The left pane displays the source code for `MainRunnableTask.java`. The right pane shows a terminal window titled "powershell" with the command "java MainRunnableTask" running. The terminal output shows four threads (Thread-0 to Thread-3) each printing their name and a value from 0 to 4.

```
MainRunnableTask.java U ×
MainRunnableTask.java > MainRunnableTask
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     public class MainRunnableTask {
17         public static void main(String[] args) {
18             Thread t1 = new Thread(new MyRunnableTask(), "thread-1");
19             Thread t2 = new Thread(new MyRunnableTask(), "thread-2");
20             Thread t3 = new Thread(new MyRunnableTask(), "thread-3");
21             Thread t4 = new Thread(new MyRunnableTask(), "thread-4");
22
23             t1.start();
24             t2.start();
25             t3.start();
26             t4.start();
27         }
28     }
29 }
```

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

Experiment 7: Collections

Add and remove element using ArrayList

The screenshot shows a Java development environment with two panes. The left pane displays the code for `ArrayListExample.java`:

```
ArrayExample.java U X
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 }
```

The right pane shows the terminal output of running the program:

```
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 a Java development environment with two panes. The left pane displays the code for `VectorMain.java`:

```
VectorMain.java U X
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 }
```

The right pane shows the terminal output of running the program:

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

c. to get the elements of LinkedList

The screenshot shows a Java development environment with two panes. The left pane displays the code for `LinkedListMain.java`:

```
LinkedListMain.java U ...
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 }
```

The right pane shows the terminal output of running 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 a Java development environment with two panes. The left pane displays the code for `HashSetExample.java`:

```
HashSetExample.java U ...
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 }
```

The right pane shows the terminal output of running 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

The screenshot shows a Java code editor window with the following code:

```
LoginPage.java U X
LoginPage.java > LoginPage > main(String[])
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class LoginPage {
5
6     Run|Debug
7     public static void main(String[] args) {
8
9         JFrame frame = new JFrame("Login Page");
10        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        frame.setSize(600, 400);
12        frame.setLocationRelativeTo(null); // Center on screen
13        frame.setLayout(new GridBagLayout());
14
15        GridBagConstraints gbc = new GridBagConstraints();
16        gbc.insets = new Insets(8, 8, 8, 8); // Spacing
17        gbc.anchor = GridBagConstraints.WEST; // Left-align labels
18
19        // Title
20        JLabel title = new JLabel("Login Form");
21        title.setFont(new Font("Arial", Font.BOLD, 20));
22
23        gbc.gridx = 0;
24        gbc.gridy = 0;
25        gbc.gridwidth = 2;
26        gbc.anchor = GridBagConstraints.WEST;
27        frame.add(title, gbc);
28
29        // Reset anchor for left alignment
30        gbc.anchor = GridBagConstraints.WEST;
31        gbc.gridwidth = 1;
32
33        // Username Label
34        gbc.gridx = 0;
35        gbc.gridy = 1;
36        frame.add(new JLabel("Username : "), gbc);
37
38    }
39}
```

The code defines a `LoginPage` class with a `main` method. It creates a `JFrame` titled "Login Page" with a `GridBagLayout`. It adds a `JLabel` for the title and a `JLabel` for the username. The code uses `GridBagConstraints` to position the labels.

The screenshot shows a Java code editor window with the same `LoginPage.java` code as above. An overlay window titled "Login Page" is displayed, showing the UI elements defined in the code: a title label "Login Form" and a text input field for "Username" containing "admin".

Simple calculator using SWING

The screenshot shows a Java code editor window with the following code:

```
SimpleCalculator.java U 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);
```

The status bar at the bottom indicates "Ln 9, Col 1 Spaces: 4 UTF-8 CRLF {} Java Go Live".

The screenshot shows a Java code editor window with the same code as above, and a separate window titled "Simple Calculator" displaying the application's user interface. The application has three text fields labeled "Number 1", "Number 2", and "Result". Below these are four buttons: "Add", "Subtract", "Multiply", and "Divide".

The status bar at the bottom indicates "Ln 19, Col 50 Spaces: 4 UTF-8 CRLF {} Java Go Live".