List of Practical (With Solution)

Course: BCA

Subject Name and Code: Programming in Java Lab (BCAC BCAC0819)

Year/Semester: II/III

Objective: The objective of this course is that the students will understand fundamentals of programming such as variables, conditional and iterative execution, methods, etc. It will also provide the foundation of good programming skills by discussing key issues to the design of object oriented programming.

Experiment 1: (If Else)

Compulsory:

Program 1: WAP to print a message “Hello JAVA”.

```
public class HelloJavaProgram1 {
    public static void main(String[] args) {
        // Print the message "Hello JAVA"
        System.out.println("Hello JAVA");
    }
}
```

```
run:
Hello JAVA
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: Write a program to display whether a number is even or odd.

```
import java.util.Scanner;

public class EvenOddChecker {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a number
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Check if the number is even or odd
        if (number % 2 == 0) {
            System.out.println(number + " is even.");
        } else {
            System.out.println(number + " is odd.");
        }
```

```
        // Close the scanner
        scanner.close();
    }
}
```

Program 3 Write the following program using if else if ladder. Accept an hour from the user and output the following as indicated below. Include the last condition in the else section.

i) Hour greater than or equal to 0 and less than 12	"Good Morning"
ii) Hour greater than or equal to 12 and less than 18	"Good Afternoon"
iii) Hour greater than or equal to 18 and less than 24	"Good Evening"
iv) Any other input	"Time is out of range"

Solution

```
import java.util.Scanner;

public class TimeGreeting {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter an hour
        System.out.print("Enter the hour (0-23): ");
        int hour = scanner.nextInt();

        // Determine the appropriate greeting based on the hour
        if (hour >= 0 && hour < 12) {
            System.out.println("Good Morning");
        } else if (hour >= 12 && hour < 18) {
            System.out.println("Good Afternoon");
        } else if (hour >= 18 && hour < 24) {
            System.out.println("Good Evening");
        } else {
            System.out.println("Time is out of range");
        }

        // Close the scanner
        scanner.close();
    }
}
```

Additional Programs:

Program 4 : A student receives marks in three subjects. The program will calculate the total marks, average marks, and determine the grade based on the average:

Average >= 90: Grade A
Average >= 80 and < 90: Grade B
Average >= 70 and < 80: Grade C
Average >= 60 and < 70: Grade D
Average < 60: Grade

```
import java.util.Scanner;

public class StudentGradeCalculator {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter marks for three subjects
        System.out.print("Enter marks for Subject 1: ");
        double marks1 = scanner.nextDouble();

        System.out.print("Enter marks for Subject 2: ");
        double marks2 = scanner.nextDouble();

        System.out.print("Enter marks for Subject 3: ");
        double marks3 = scanner.nextDouble();

        // Calculate total marks and average
        double totalMarks = marks1 + marks2 + marks3;
        double averageMarks = totalMarks / 3;

        // Determine the grade based on the average marks
        char grade;
        if (averageMarks >= 90) {
            grade = 'A';
        } else if (averageMarks >= 80) {
            grade = 'B';
        } else if (averageMarks >= 70) {
            grade = 'C';
        } else if (averageMarks >= 60) {
            grade = 'D';
        } else {
            grade = 'F';
        }
```

```
        // Display the results
        System.out.println("Total Marks: " + totalMarks);
        System.out.println("Average Marks: " + averageMarks);
        System.out.println("Grade: " + grade);

        // Close the scanner
        scanner.close();
    }
}
```

Experiment 2: Nested If Else

Compulsory:

Program 1: : WAP to find maximum of three numbers

```
import java.util.Scanner;

public class MaxOfThreeNumbers {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter three numbers
        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();

        System.out.print("Enter the third number: ");
        double num3 = scanner.nextDouble();

        // Find the maximum of the three numbers
        double max;
        if (num1 >= num2 && num1 >= num3) {
            max = num1;
        } else if (num2 >= num1 && num2 >= num3) {
            max = num2;
        } else {
```

```
            max = num3;
        }

        // Display the result
        System.out.println("The maximum of the three numbers is: " + max);

        // Close the scanner
        scanner.close();
    }
}
```

// program that finds the maximum of three numbers using nested if-else statements:

```
import java.util.Scanner;

public class MaxOfThreeNumbersNestedIfElse {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter three numbers
        System.out.print("Enter the first number: ");
        double num1 = scanner.nextDouble();

        System.out.print("Enter the second number: ");
        double num2 = scanner.nextDouble();

        System.out.print("Enter the third number: ");
        double num3 = scanner.nextDouble();

        // Find the maximum of the three numbers using nested if-else
        double max;
        if (num1 >= num2) {
            // num1 is greater than or equal to num2
            if (num1 >= num3) {
                // num1 is also greater than or equal to num3
                max = num1;
            } else {
                // num1 is greater than or equal to num2 but less than num3
                max = num3;
            }
        } else {
            // num1 is less than num2
```

```
            if (num2 >= num3) {
                // num2 is greater than or equal to num3
                max = num2;
            } else {
                // num2 is less than num3
                max = num3;
            }
        }

        // Display the result
        System.out.println("The maximum of the three numbers is: " + max);

        // Close the scanner
        scanner.close();
    }
}
```

Program 2: A movie theatre has the following ticket pricing rules:
Children under 12 years old pay $5.
Seniors 65 years and older pay $7.
Regular adults (12-64 years old) pay $10.
Members get a $2 discount on all ticket prices.
We'll prompt the user to input their age and whether they have a membership card. Based on this input, the program will determine and print the ticket price.

Solution

```
import java.util.Scanner;

public class MovieTicketPrice {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get the age of the customer
        System.out.print("Enter your age: ");
        int age = scanner.nextInt();

        // Get the membership status of the customer
        System.out.print("Do you have a membership card? (yes/no): ");
        String membership = scanner.next();

        // Initialize the ticket price
        double ticketPrice = 0.0;

        // Determine the ticket price based on age
        if (age < 12) {
```

```
            ticketPrice = 5.0;
        } else if (age >= 65) {
            ticketPrice = 7.0;
        } else {
            ticketPrice = 10.0;
        }

        // Apply discount if the customer has a membership card
        if (membership.equalsIgnoreCase("yes")) {
            ticketPrice -= 2.0;
        }

        // Display the final ticket price
        System.out.println("Your ticket price is: $" + ticketPrice);

        // Close the scanner
        scanner.close();
    }
}
```

```
run:
Enter your age: 34
Do you have a membership card? (yes/no): yes
Your ticket price is: $8.0
BUILD SUCCESSFUL (total time: 5 seconds)
```

Program3: Create a program using switch case statement to identify the day of the week.

```
import java.util.Scanner;

public class DayOfWeek {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a number (1-7)
        System.out.print("Enter a number (1-7) to find the day of the week: ");
        int dayNumber = scanner.nextInt();

        // Identify the day of the week using a switch statement
        String dayName;
        switch (dayNumber) {
            case 1:
                dayName = "Monday";
```

```
            break;
        case 2:
            dayName = "Tuesday";
            break;
        case 3:
            dayName = "Wednesday";
            break;
        case 4:
            dayName = "Thursday";
            break;
        case 5:
            dayName = "Friday";
            break;
        case 6:
            dayName = "Saturday";
            break;
        case 7:
            dayName = "Sunday";
            break;
        default:
            dayName = "Invalid number. Please enter a number between 1 and 7.";
            break;
    }

    // Display the result
    System.out.println("Day of the week: " + dayName);

    // Close the scanner
    scanner.close();
  }
}
```

Additional Programs:

Program3: WAP to implement that performs basic arithmetic operations (addition, subtraction, multiplication, division) based on user input. use switch to select the arithmetic operation

```
import java.util.Scanner;

public class SimpleCalculator {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.println("Enter first number: ");
    double num1 = scanner.nextDouble();
```

```
        System.out.println("Enter second number: ");
        double num2 = scanner.nextDouble();

        System.out.println("Choose an operation: +, -, *, /");
        char operator = scanner.next().charAt(0);

        double result;

        switch (operator) {
            case '+':
                result = num1 + num2;
                break;
            case '-':
                result = num1 - num2;
                break;
            case '*':
                result = num1 + num2;
                break;
            case '/':
                if (num2 != 0) {
                    result = num1 / num2;
                } else {
                    System.out.println("Division by zero is not allowed.");
                    return;
                }
                break;
            default:
                System.out.println("Invalid operator");
                return;
        }

        System.out.println("The result is: " + result);
    }
}
```

Experiment 3: Simple and Nested Loop

Compulsory:

Program1 : Write a Program in Java to Calculate the Factorial of an Integer using a for loop

```
import java.util.Scanner;

public class FactorialCalculator3 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt user to enter a number
        System.out.print("Enter a number to calculate its factorial: ");
```

```
        int number = scanner.nextInt();

        // Calculate the factorial
        long factorial = 1;
        for (int i = 1; i <= number; i++) {
            factorial *= i;
        }

        // Print the factorial
        System.out.println("The factorial of " + number + " is " + factorial);

        // Close the scanner
        scanner.close();
    }
}
```

```
run:
Enter a number to calculate its factorial: 5
The factorial of 5 is 120
BUILD SUCCESSFUL (total time: 6 seconds)
```

Program 2: You are tasked with developing a program that simulates a login system. The user is given three attempts to enter the correct password (In integer format eg. 4545). If the user enters the correct password within three attempts, they are granted access. If the user fails to enter the correct password in three attempts, they are locked out.

```
import java.util.Scanner;

public class LoginSystem {
    public static void main(String[] args) {
        // Define the correct password
        final int CORRECT_PASSWORD = 4545;
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Variable to track the number of attempts
        int attempts = 0;
        boolean accessGranted = false;

        // Allow the user up to 3 attempts to enter the correct password
        while (attempts < 3) {
            // Prompt the user to enter the password
            System.out.print("Enter password: ");
            int enteredPassword = scanner.nextInt();

            // Check if the entered password is correct
            if (enteredPassword == CORRECT_PASSWORD) {
```

```
            accessGranted = true;
            break;
        } else {
            // Increment the number of attempts
            attempts++;
            System.out.println("Incorrect password. You have " + (3 - attempts) + " attempts
left.");
        }
    }

    // Display the result based on whether access was granted
    if (accessGranted) {
        System.out.println("Access granted.");
    } else {
        System.out.println("Access denied. You have been locked out.");
    }

    // Close the scanner
    scanner.close();
  }
}
```

Additional Programs:

Program 3: WAP to display table of a number in a format n x i=m

```
import java.util.Scanner;

public class MultiplicationTable {
  public static void main(String[] args) {
    // Create a Scanner object to read input from the user
    Scanner scanner = new Scanner(System.in);

    // Prompt the user to enter a number
    System.out.print("Enter a number to display its multiplication table: ");
    int number = scanner.nextInt();

    // Display the multiplication table
    System.out.println("Multiplication table for " + number + ":");
    for (int i = 1; i <= 10; i++) {
        int result = number * i;
        System.out.println(number + " x " + i + " = " + result);
    }

    // Close the scanner
    scanner.close();
```

```
    }
}
```

Program 4: WAP to find sum of digits of a number

```
import java.util.Scanner;

public class SumOfDigits {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter a number
        System.out.print("Enter an integer: ");
        int number = scanner.nextInt();

        // Initialize sum to 0
        int sum = 0;

        // Make the number positive if it is negative
        number = Math.abs(number);

        // Calculate the sum of the digits
        while (number > 0) {
            // Extract the last digit of the number
            int digit = number % 10;
            // Add the digit to the sum
            sum += digit;
            // Remove the last digit from the number
            number /= 10;
        }

        // Display the result
        System.out.println("Sum of the digits: " + sum);

        // Close the scanner
        scanner.close();
    }
}
```

Program 5: WAP to implement type conversion in java.

```
public class TypeConversionDemo {
    public static void main(String[] args) {
        // Implicit Type Conversion (Widening Conversion)
        int intValue = 100;
        double doubleValue = intValue;  // int is automatically converted to double
```

```
        System.out.println("Implicit Conversion:");
        System.out.println("Integer value: " + intValue);
        System.out.println("Double value after implicit conversion: " + doubleValue);

        // Explicit Type Conversion (Narrowing Conversion)
        doubleValue = 123.456;
        intValue = (int) doubleValue;  // double is explicitly converted to int

        System.out.println("\nExplicit Conversion:");
        System.out.println("Double value: " + doubleValue);
        System.out.println("Integer value after explicit conversion: " + intValue);

        // Converting String to Numeric Types
        String numberString = "2024";
        int number = Integer.parseInt(numberString);  // String to int

        System.out.println("\nString to Numeric Conversion:");
        System.out.println("String value: " + numberString);
        System.out.println("Integer value after conversion: " + number);

        // Converting Numeric Types to String
        int anotherNumber = 1234;
        String anotherNumberString = Integer.toString(anotherNumber);  // int to String

        System.out.println("\nNumeric to String Conversion:");
        System.out.println("Integer value: " + anotherNumber);
        System.out.println("String value after conversion: " + anotherNumberString);
    }
}
```

Experiment 4: Patterns

Compulsory:

Program 1: Using For…..Loop display the following pattern :

```
*
**
***
****
*****
```

```
public class PatternDisplay {
    public static void main(String[] args) {
        // Number of rows for the pattern
        int rows = 5;

        // Loop to iterate through each row
```

```
        for (int i = 1; i <= rows; i++) {
            // Loop to print stars in each row
            for (int j = 1; j <= i; j++) {
                System.out.print("*");
            }
            // Move to the next line after printing stars in a row
            System.out.println();
        }
    }
}
```

Program 2: WAP to print the pattern

```
11111
12111
11311
11141
11115
```

```
public class PatternPrint {
    public static void main(String[] args) {
        // Number of rows and columns for the pattern
        int size = 5;

        // Loop to iterate through each row
        for (int i = 0; i < size; i++) {
            // Loop to print each column in the current row
            for (int j = 0; j < size; j++) {
                // Print '1' except for the diagonal and last column
                if (i == j) {
                    System.out.print((i + 1));  // Print 1-based index of the row
                } else {
                    System.out.print("1");
                }
            }
            // Move to the next line after printing all columns in a row
            System.out.println();
        }
    }
```

<u>Additional Programs:</u>

Program3: WAP to print the pattern

```
11111
12111
12311
12341
12345
```

```
public class PatternPrint {
    public static void main(String[] args) {
        // Number of rows and columns for the pattern
        int size = 5;

        // Loop to iterate through each row
        for (int i = 1; i <= size; i++) {
            // Loop to print each column in the current row
            for (int j = 1; j <= size; j++) {
                // Print numbers up to the current row index or 1
                if (j <= i) {
                    System.out.print(j);
                } else {
                    System.out.print("1");
                }
            }
            // Move to the next line after printing all columns in a row
            System.out.println();
        }
    }
}
```

Program 4: WAP to print the pattern

```
#####
$####
$$###
$$$##
$$$$#
```

```
public class PatternPrint {
    public static void main(String[] args) {
        // Number of rows for the pattern
        int rows = 4;

        // Loop to iterate through each row
        for (int i = 1; i <= rows; i++) {
            // Print '$' characters for each row
            for (int j = 1; j < i; j++) {
                System.out.print("$");
            }
            // Print '#' characters for each row
            for (int k = i; k < rows; k++) {
                System.out.print("#");
            }
            // Move to the next line after printing all characters in the row
```

```
            System.out.println();
        }
    }
}
```

Experiment 5: Array

Compulsory:

Program 1: WAP to read and print an array. Also find greatest and smallest element in array.

```
import java.util.Scanner;

public class ArrayOperations {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the number of elements in the array
        System.out.print("Enter the number of elements in the array: ");
        int size = scanner.nextInt();

        // Initialize the array with the specified size
        int[] array = new int[size];

        // Read the elements of the array from the user
        System.out.println("Enter " + size + " elements:");
        for (int i = 0; i < size; i++) {
            array[i] = scanner.nextInt();
        }

        // Print the elements of the array
        System.out.println("Array elements:");
        for (int i = 0; i < size; i++) {
            System.out.print(array[i] + " ");
        }
        System.out.println();

        // Initialize variables to find the smallest and largest elements
        int smallest = array[0];
        int largest = array[0];

        // Find the smallest and largest elements in the array
```

```
        for (int i = 1; i < size; i++) {
            if (array[i] < smallest) {
                smallest = array[i];
            }
            if (array[i] > largest) {
                largest = array[i];
            }
        }

        // Display the smallest and largest elements
        System.out.println("Smallest element in the array: " + smallest);
        System.out.println("Largest element in the array: " + largest);

        // Close the scanner
        scanner.close();
    }
}
```

Program 2 You're developing a simple scoring system for a cricket match. The match involves a single over (6 balls) and you're required to record the runs scored on each ball. The system should also calculate the total runs scored in the over and determine if the over included any dot balls (a ball where no runs are scored).

Task:

Create a 1-dimensional array to store the runs scored on each of the 6 balls in the over.
Calculate the total runs scored in the over.
Count the number of dot balls (balls where zero runs were scored).
Determine the highest run scored on a single ball.

Runs scored in the over (ball by ball):

Ball 1: 1 run
Ball 2: 4 runs
Ball 3: 0 runs (dot ball)
Ball 4: 6 runs
Ball 5: 2 runs
Ball 6: 0 runs (dot ball)

```
public class CricketScoringSystem {
    public static void main(String[] args) {
        // Step 1: Initialize the array with runs scored on each ball
        int[] runsPerBall = {1, 4, 0, 6, 2, 0};

        // Step 2: Calculate the total runs scored in the over
        int totalRuns = 0;
```

```
        for (int runs : runsPerBall) {
            totalRuns += runs;
        }

        // Step 3: Count the number of dot balls
        int dotBalls = 0;
        for (int runs : runsPerBall) {
            if (runs == 0) {
                dotBalls++;
            }
        }

        // Step 4: Determine the highest run scored on a single ball
        int highestRun = runsPerBall[0];
        for (int runs : runsPerBall) {
            if (runs > highestRun) {
                highestRun = runs;
            }
        }

        // Output the results
        System.out.println("Total runs scored in the over: " + totalRuns);
        System.out.println("Number of dot balls: " + dotBalls);
        System.out.println("Highest run scored on a single ball: " + highestRun);
    }
}
```

Additional Programs:

Program 3: Write a program to search an element in an array.

```
import java.util.Scanner;

public class ArraySearch {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
        Scanner scanner = new Scanner(System.in);

        // Prompt the user to enter the number of elements in the array
        System.out.print("Enter the number of elements in the array: ");
        int size = scanner.nextInt();

        // Initialize the array with the specified size
        int[] array = new int[size];
```

```
        // Read the elements of the array from the user
        System.out.println("Enter " + size + " elements:");
        for (int i = 0; i < size; i++) {
            array[i] = scanner.nextInt();
        }

        // Prompt the user to enter the element to search for
        System.out.print("Enter the element to search for: ");
        int searchElement = scanner.nextInt();

        // Search for the element in the array
        boolean found = false;
        int position = -1;
        for (int i = 0; i < size; i++) {
            if (array[i] == searchElement) {
                found = true;
                position = i;
                break;
            }
        }

        // Display the result
        if (found) {
            System.out.println("Element " + searchElement + " is present at index " + position +
".");
        } else {
            System.out.println("Element " + searchElement + " is not present in the array.");
        }

        // Close the scanner
        scanner.close();
    }
}
```

Program 4: Write a program to sort an array using bubble sort.

```
import java.util.Scanner;

public class BubbleSortExample {
    public static void main(String[] args) {
        // Create a Scanner object to read input from the user
```

```
    Scanner scanner = new Scanner(System.in);

    // Prompt the user to enter the number of elements in the array
    System.out.print("Enter the number of elements in the array: ");
    int size = scanner.nextInt();

    // Initialize the array with the specified size
    int[] array = new int[size];

    // Read the elements of the array from the user
    System.out.println("Enter " + size + " elements:");
    for (int i = 0; i < size; i++) {
        array[i] = scanner.nextInt();
    }

    // Perform bubble sort
    bubbleSort(array);

    // Print the sorted array
    System.out.println("Sorted array:");
    for (int i = 0; i < size; i++) {
        System.out.print(array[i] + " ");
    }
    System.out.println();

    // Close the scanner
    scanner.close();
}

// Method to perform bubble sort
public static void bubbleSort(int[] array) {
    int n = array.length;
    boolean swapped;
    // Loop through all array elements
    for (int i = 0; i < n - 1; i++) {
        swapped = false;
        // Compare each pair of adjacent elements
        for (int j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                // Swap if elements are in the wrong order
                int temp = array[j];
                array[j] = array[j + 1];
```

```
                array[j + 1] = temp;
                swapped = true;
            }
        }
        // If no elements were swapped, the array is sorted
        if (!swapped) {
            break;
        }
    }
  }
}
```

Program 5: WAP to read and print a matrix

```
import java.util.Scanner;

public class MatrixOperations {
   public static void main(String[] args) {
      // Create a Scanner object to read input from the user
      Scanner scanner = new Scanner(System.in);

      // Prompt the user to enter the number of rows and columns
      System.out.print("Enter the number of rows: ");
      int rows = scanner.nextInt();
      System.out.print("Enter the number of columns: ");
      int columns = scanner.nextInt();

      // Initialize the matrix with the specified rows and columns
      int[][] matrix = new int[rows][columns];

      // Read the elements of the matrix from the user
      System.out.println("Enter the elements of the matrix:");
      for (int i = 0; i < rows; i++) {
         for (int j = 0; j < columns; j++) {
            System.out.print("Element [" + i + "][" + j + "]: ");
            matrix[i][j] = scanner.nextInt();
         }
      }

      // Print the matrix
      System.out.println("Matrix:");
      for (int i = 0; i < rows; i++) {
         for (int j = 0; j < columns; j++) {
```

```
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println(); // Move to the next line after printing a row
    }

    // Close the scanner
    scanner.close();
  }
}
```

Program 6: WAP to find sum of two matrices

```
import java.util.Scanner;

public class MatrixSum {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // Prompt the user to enter the number of rows and columns
    System.out.print("Enter the number of rows: ");
    int rows = scanner.nextInt();
    System.out.print("Enter the number of columns: ");
    int columns = scanner.nextInt();

    // Initialize matrices with the specified rows and columns
    int[][] matrix1 = new int[rows][columns];
    int[][] matrix2 = new int[rows][columns];
    int[][] sumMatrix = new int[rows][columns];

    // Read elements of the first matrix
    System.out.println("Enter the elements of the first matrix:");
    readMatrix(scanner, matrix1, rows, columns);

    // Read elements of the second matrix
    System.out.println("Enter the elements of the second matrix:");
    readMatrix(scanner, matrix2, rows, columns);

    // Compute the sum of the two matrices
    for (int i = 0; i < rows; i++) {
      for (int j = 0; j < columns; j++) {
        sumMatrix[i][j] = matrix1[i][j] + matrix2[i][j];
```

```
            }
        }

        // Display the resulting matrix
        System.out.println("Sum of the two matrices:");
        printMatrix(sumMatrix, rows, columns);

        // Close the scanner
        scanner.close();
    }

    // Method to read matrix elements from the user
    public static void readMatrix(Scanner scanner, int[][] matrix, int rows, int columns) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                System.out.print("Element [" + i + "][" + j + "]: ");
                matrix[i][j] = scanner.nextInt();
            }
        }
    }

    // Method to print matrix
    public static void printMatrix(int[][] matrix, int rows, int columns) {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < columns; j++) {
                System.out.print(matrix[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Experiment 6: Strings

Compulsory:

Program1: Write a Java program that will accept command-line arguments and display the same.

```
public class CommandLineArgs {
    public static void main(String[] args) {
        // Check if there are any command-line arguments
        if (args.length == 0) {
            System.out.println("No command-line arguments provided.");
```

```
        } else {
            System.out.println("Command-line arguments provided:");
            // Loop through the args array and print each argument
            for (int i = 0; i < args.length; i++) {
                System.out.println("Argument " + (i + 1) + ": " + args[i]);
            }
        }
    }
}
```

Program2: You are developing a simple application to manage the player lineup for a football (soccer) match. The coach needs to maintain a list of the starting 11 players for the match. After finalizing the lineup, the coach wants to:

Display the names of all the players in the starting lineup.
Check if a specific player is in the starting lineup.
Update the lineup by replacing a player who got injured during the warm-up with a substitute.
Task:

Create a String[] array to store the names of the 11 starting players.
Implement a method to display all players in the lineup.
Implement a method to check if a specific player is in the starting lineup.
Implement a method to replace an injured player with a substitute.
Starting Lineup:

Player 1: John
Player 2: David
Player 3: Mike
Player 4: Chris
Player 5: Alex
Player 6: Ryan
Player 7: James
Player 8: Sam
Player 9: Robert
Player 10: Daniel
Player 11: Steve
Substitute:

Substitute Player: Luke (to replace injured player James)

```
import java.util.Arrays;

public class FootballLineupManager {
    // Step 1: Initialize the array with the starting lineup
    static String[] startingLineup = {
        "John", "David", "Mike", "Chris", "Alex",
        "Ryan", "James", "Sam", "Robert", "Daniel", "Steve"
```

```
    };

    public static void main(String[] args) {
        // Display the starting lineup
        displayLineup();

        // Check if a specific player (e.g., "James") is in the lineup
        String playerToCheck = "James";
        if (isPlayerInLineup(playerToCheck)) {
            System.out.println(playerToCheck + " is in the starting lineup.");
        } else {
            System.out.println(playerToCheck + " is not in the starting lineup.");
        }

        // Replace an injured player with a substitute
        String injuredPlayer = "James";
        String substitutePlayer = "Luke";
        replacePlayer(injuredPlayer, substitutePlayer);

        // Display the updated lineup
        displayLineup();
    }

    // Method to display all players in the lineup
    public static void displayLineup() {
        System.out.println("Starting lineup:");
        for (String player : startingLineup) {
            System.out.println(player);
        }
        System.out.println();
    }

    // Method to check if a specific player is in the lineup
    public static boolean isPlayerInLineup(String playerName) {
        return Arrays.asList(startingLineup).contains(playerName);
    }

    // Method to replace an injured player with a substitute
    public static void replacePlayer(String injuredPlayer, String substitutePlayer) {
        for (int i = 0; i < startingLineup.length; i++) {
            if (startingLineup[i].equals(injuredPlayer)) {
                startingLineup[i] = substitutePlayer;
                System.out.println(injuredPlayer + " has been replaced by " + substitutePlayer + ".");
                return;
            }
        }
        System.out.println(injuredPlayer + " is not in the lineup, so no replacement was made.");
    }
}
```

Additional Programs:

Program 3: Write a program to accept number through Command line and display its factorial.

```
public class FactorialCalculator {
    public static void main(String[] args) {
        // Check if exactly one argument is provided
        if (args.length != 1) {
            System.out.println("Usage: java FactorialCalculator <number>");
            return;
        }

        try {
            // Parse the argument as an integer
            int number = Integer.parseInt(args[0]);

            // Check if the number is non-negative
            if (number < 0) {
                System.out.println("Factorial is not defined for negative numbers.");
                return;
            }

            // Calculate factorial
            long factorial = calculateFactorial(number);

            // Display the result
            System.out.println("Factorial of " + number + " is " + factorial);
        } catch (NumberFormatException e) {
            System.out.println("Invalid input. Please enter a valid integer.");
        }
    }

    // Method to calculate factorial
    public static long calculateFactorial(int number) {
        long factorial = 1;
        for (int i = 1; i <= number; i++) {
            factorial *= i;
        }
        return factorial;
    }
```

```
}
```

Program 4: Write a program to extract a substring from a given string.

```
import java.util.Scanner;

public class SubstringExtractor {
   public static void main(String[] args) {
      // Create a Scanner object to read input from the user
      Scanner scanner = new Scanner(System.in);

      // Prompt the user to enter the original string
      System.out.print("Enter the original string: ");
      String originalString = scanner.nextLine();

      // Prompt the user to enter the start index for the substring
      System.out.print("Enter the start index of the substring: ");
      int startIndex = scanner.nextInt();

      // Prompt the user to enter the end index for the substring
      System.out.print("Enter the end index of the substring (exclusive): ");
      int endIndex = scanner.nextInt();

      // Validate indices
      if (startIndex < 0 || endIndex > originalString.length() || startIndex >= endIndex) {
         System.out.println("Invalid indices. Ensure that 0 <= startIndex < endIndex <=
string length.");
      } else {
         // Extract the substring
         String substring = originalString.substring(startIndex, endIndex);

         // Display the substring
         System.out.println("Extracted substring: " + substring);
      }

      // Close the scanner
      scanner.close();
   }
}
```

Experiment 7: Class

<u>Compulsory:</u>

Program1: WAP to create a Rectangle class. Define functions for
Reading length and breadth for rectangle
Calculate Area
Display length, breadth and area

```
import java.util.Scanner;

class Rectangle {
   private double length;
   private double breadth;

   // Method to read length and breadth
   public void readDimensions() {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the length of the rectangle: ");
      length = scanner.nextDouble();
      System.out.print("Enter the breadth of the rectangle: ");
      breadth = scanner.nextDouble();
   }

   // Method to calculate the area of the rectangle
   public double calculateArea() {
      return length * breadth;
   }

   // Method to display the length, breadth, and area
   public void display() {
      double area = calculateArea();
      System.out.println("Length: " + length);
      System.out.println("Breadth: " + breadth);
      System.out.println("Area: " + area);
   }
}

public class RectangleTest {
   public static void main(String[] args) {
      // Create a Rectangle object
      Rectangle rectangle = new Rectangle();

      // Read dimensions of the rectangle
      rectangle.readDimensions();

      // Display the dimensions and area of the rectangle
      rectangle.display();
   }
}
```

Additional Programs:

Program2: This program defines a Student class with attributes like student ID, name, age, and grade. It also provides methods to set and get these attributes, along with a method to display the student's details.

```
class Student {
    // Attributes of the Student class
    private int studentID;
    private String name;
    private int age;
    private char grade;

    // Constructor to initialize the student object
    public Student(int studentID, String name, int age, char grade) {
        this.studentID = studentID;
        this.name = name;
        this.age = age;
        this.grade = grade;
    }

    // Getter and Setter methods for studentID
    public int getStudentID() {
        return studentID;
    }

    public void setStudentID(int studentID) {
        this.studentID = studentID;
    }

    // Getter and Setter methods for name
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Getter and Setter methods for age
    public int getAge() {
        return age;
    }
```

```
    public void setAge(int age) {
        this.age = age;
    }

    // Getter and Setter methods for grade
    public char getGrade() {
        return grade;
    }

    public void setGrade(char grade) {
        this.grade = grade;
    }

    // Method to display student details
    public void displayStudentDetails() {
        System.out.println("Student ID: " + studentID);
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Grade: " + grade);
    }
}

public class StudentTest {
    public static void main(String[] args) {
        // Create a Student object using the constructor
        Student student = new Student(101, "John Doe", 20, 'A');

        // Display student details
        student.displayStudentDetails();

        // Modify some attributes
        student.setName("Jane Doe");
        student.setGrade('B');

        // Display updated student details
        System.out.println("\nUpdated Student Details:");
        student.displayStudentDetails();
    }
}
```

Program3: You're tasked with developing a basic banking application. The application should allow users to:

Open a new bank account.
Deposit money into their account.
Withdraw money from their account.
Check their account balance.

```
class BankAccount {
    private String accountHolderName;
    private String accountNumber;
    private double balance;

    // Constructor to open a new bank account
    public BankAccount(String accountHolderName, String accountNumber) {
        this.accountHolderName = accountHolderName;
        this.accountNumber = accountNumber;
        this.balance = 0.0; // Initial balance is 0
    }

    // Method to deposit money into the account
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Successfully deposited: $" + amount);
        } else {
            System.out.println("Deposit amount must be positive.");
        }
    }

    // Method to withdraw money from the account
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Successfully withdrew: $" + amount);
        } else if (amount > balance) {
            System.out.println("Insufficient balance. Withdrawal failed.");
        } else {
            System.out.println("Withdrawal amount must be positive.");
        }
    }

    // Method to check the account balance
    public double getBalance() {
        return balance;
    }

    // Method to display account details
    public void displayAccountDetails() {
        System.out.println("Account Holder: " + accountHolderName);
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Current Balance: $" + balance);
    }
}

public class BankApplication {
    public static void main(String[] args) {
```

```
        // Open a new bank account
        BankAccount account = new BankAccount("John Doe", "1234567890");
        account.displayAccountDetails();

        // Deposit money
        account.deposit(1000.00);
        account.displayAccountDetails();

        // Withdraw money
        account.withdraw(500.00);
        account.displayAccountDetails();

        // Attempt to withdraw more money than the balance
        account.withdraw(600.00);
        account.displayAccountDetails();

        // Check balance
        System.out.println("Final Balance: $" + account.getBalance());
    }
}
```

Experiment 8: Constructor

Compulsory:

Program 1: In this program, we'll design a Book class that demonstrates the use of constructors to initialize objects. The Book class will have attributes like title, author, and price. We will use different types of constructors:

Default Constructor: Initializes attributes with default values.
Parameterized Constructor: Initializes attributes with specific values.

```
class Book {
    // Attributes of the Book class
    private String title;
    private String author;
    private double price;

    // Default Constructor
    public Book() {
        this.title = "Unknown Title";
        this.author = "Unknown Author";
        this.price = 0.0;
    }
```

```
    // Parameterized Constructor
    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    // Getter methods
    public String getTitle() {
        return title;
    }

    public String getAuthor() {
        return author;
    }

    public double getPrice() {
        return price;
    }

    // Setter methods
    public void setTitle(String title) {
        this.title = title;
    }

    public void setAuthor(String author) {
        this.author = author;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    // Method to display book details
    public void displayBookDetails() {
        System.out.println("Title: " + title);
        System.out.println("Author: " + author);
        System.out.println("Price: $" + price);
    }
}

public class BookTest {
    public static void main(String[] args) {
        // Create a Book object using the default constructor
        Book defaultBook = new Book();
        System.out.println("Default Book Details:");
        defaultBook.displayBookDetails();

        // Create a Book object using the parameterized constructor
        Book specificBook = new Book("1984", "George Orwell", 15.99);
```

```
        System.out.println("\nSpecific Book Details:");
        specificBook.displayBookDetails();

        // Modify the specific book's attributes
        specificBook.setTitle("Animal Farm");
        specificBook.setPrice(12.99);

        // Display updated book details
        System.out.println("\nUpdated Specific Book Details:");
        specificBook.displayBookDetails();
    }
}
```

Additional Programs:

Program2: In this program, we'll create a Car class that makes use of constructors. The Car class will have attributes like brand, model, and year of manufacture. We'll include:

A default constructor to initialize attributes with default values.
A parameterized constructor to initialize attributes with specific values.

```
class Car {
    // Attributes of the Car class
    private String brand;
    private String model;
    private int year;

    // Default Constructor
    public Car() {
        this.brand = "Unknown";
        this.model = "Unknown";
        this.year = 0;
    }

    // Parameterized Constructor
    public Car(String brand, String model, int year) {
        this.brand = brand;
        this.model = model;
        this.year = year;
    }

    // Getter methods
```

```
    public String getBrand() {
        return brand;
    }

    public String getModel() {
        return model;
    }

    public int getYear() {
        return year;
    }

    // Setter methods
    public void setBrand(String brand) {
        this.brand = brand;
    }

    public void setModel(String model) {
        this.model = model;
    }

    public void setYear(int year) {
        this.year = year;
    }

    // Method to display car details
    public void displayCarDetails() {
        System.out.println("Brand: " + brand);
        System.out.println("Model: " + model);
        System.out.println("Year: " + year);
    }
}

public class CarTest {
    public static void main(String[] args) {
        // Create a Car object using the default constructor
        Car defaultCar = new Car();
        System.out.println("Default Car Details:");
        defaultCar.displayCarDetails();

        // Create a Car object using the parameterized constructor
        Car specificCar = new Car("Toyota", "Corolla", 2022);
        System.out.println("\nSpecific Car Details:");
        specificCar.displayCarDetails();

        // Modify the specific car's attributes
        specificCar.setBrand("Honda");
        specificCar.setModel("Civic");
        specificCar.setYear(2023);
```

```
        // Display updated car details
        System.out.println("\nUpdated Specific Car Details:");
        specificCar.displayCarDetails();
    }
}
```

Program3 :In this program, we'll design a Person class that makes use of both default and parameterized constructors. The Person class will have attributes for the person's name, age, and address. We'll provide methods to initialize these attributes, display the person's details, and modify the attributes.

```
class Person {
    // Attributes of the Person class
    private String name;
    private int age;
    private String address;

    // Default Constructor
    public Person() {
        this.name = "Unknown";
        this.age = 0;
        this.address = "Unknown";
    }

    // Parameterized Constructor
    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }

    // Getter and Setter methods for name
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // Getter and Setter methods for age
    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
```

```
    // Getter and Setter methods for address
    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    // Method to display person details
    public void displayPersonDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Address: " + address);
    }
}

public class PersonTest {
    public static void main(String[] args) {
        // Create a Person object using the default constructor
        Person defaultPerson = new Person();
        System.out.println("Default Person Details:");
        defaultPerson.displayPersonDetails();

        // Create a Person object using the parameterized constructor
        Person specificPerson = new Person("Alice Johnson", 30, "123 Main St, Springfield");
        System.out.println("\nSpecific Person Details:");
        specificPerson.displayPersonDetails();

        // Modify the specific person's attributes
        specificPerson.setName("Bob Smith");
        specificPerson.setAge(35);
        specificPerson.setAddress("456 Elm St, Springfield");

        // Display updated person details
        System.out.println("\nUpdated Person Details:");
        specificPerson.displayPersonDetails();
    }
}
```

Experiment 9: Static and This

Compulsory:

Program 1: Write a Java program that demonstrates the use of a static data member to count the number of objects created for a class.

Solution

```
class Counter {
   // Static variable to count the number of objects created
   private static int objectCount = 0;

   // Constructor
   public Counter() {
      // Increment the count each time a new object is created
      objectCount++;
   }

   // Static method to get the count of objects created
   public static int getObjectCount() {
      return objectCount;
   }
}

public class ObjectCountDemo {
   public static void main(String[] args) {
      // Creating objects of the Counter class
      Counter obj1 = new Counter();
      Counter obj2 = new Counter();
      Counter obj3 = new Counter();

      // Displaying the count of objects created
      System.out.println("Number of Counter objects created: " + Counter.getObjectCount());
   }
}
```

```
run:
Number of Counter objects created: 3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: WAP Java program that demonstrates the use of 'this' pointer in a class. We'll create a Person class where we use 'this' pointer to distinguish between instance variables and constructor parameters.

Solution

```
class Person {
   // Instance variables
```

```
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) {
        // Using 'this' to differentiate instance variables from parameters
        this.name = name; // Assign parameter to instance variable
        this.age = age;   // Assign parameter to instance variable
    }

    // Method to display person's details
    public void displayDetails() {
        System.out.println("Name: " + this.name); // Using 'this' is optional here
        System.out.println("Age: " + this.age);   // Using 'this' is optional here
    }
}

public class PersonDemo {
    public static void main(String[] args) {
        // Creating an instance of Person
        Person person1 = new Person("Alice", 30);
        Person person2 = new Person("Bob", 25);

        // Displaying details of each person
        person1.displayDetails();
        person2.displayDetails();
    }
}
```

```
run:
Name: Alice
Age: 30
Name: Bob
Age: 25
BUILD SUCCESSFUL (total time: 0 seconds)
```

Additional Programs:

Program 1: WAP that use ‘this’ keyword to return the current class instance

https://www.geeksforgeeks.org/this-reference-in-java/

Solution

```
// Java code for using 'this' keyword
// to return the current class instance
class Test {
	int a;
	int b;

	// Default constructor
	Test()
	{
		a = 10;
		b = 20;
	}

	// Method that returns current class instance
	Test get() { return this; }

	// Displaying value of variables a and b
	void display()
	{
		System.out.println("a = " + a + " b = " + b);
	}

	public static void main(String[] args)
	{
		Test object = new Test();
		object.get().display();
	}
}

run:
a = 10 b = 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: WAP that use ‘this’ keyword to invoke the current class method

Solution

```
// Java code for using this to invoke current
// class method
class Test {

	void display()
	{
		// calling function show()
		this.show();

		System.out.println("Inside display function");
```

```
	}

	void show()
	{
		System.out.println("Inside show function");
	}

	public static void main(String args[])
	{
		Test t1 = new Test();
		t1.display();
	}
}
```

```
run:
Inside show function
Inside display function
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment 10: Inheritance

Compulsory:

Program 1: WAP to implement single inheritance in Java

Solution

```
// WAP to implement single inheritance in Java
class Animal {
    // Method in the base class
    public void display() {
        System.out.println("This is an animal.");
    }
}

// Derived class
class Dog extends Animal {
    // Method in the derived class
    public void bark() {
        System.out.println("The dog barks.");
    }
}

public class SingleInheritanceDemo {
    public static void main(String[] args) {
        // Creating an instance of Dog
        Dog myDog = new Dog();

        // Calling methods from both the Dog and Animal classes
        myDog.display(); // Method from the base class
        myDog.bark();    // Method from the derived class
    }
}
```

```
run:
This is an animal.
The dog barks.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: Animal and Dog Classes

Solution

Problem Statement: You need to manage different types of animals and their characteristics. The base class will represent a general animal, while the derived class will represent a specific type of animal, in this case, a dog. The program will demonstrate how the dog inherits attributes and behaviors from the animal.

Solution

```
// Base class
class Animal {
    // Instance variables
    private String name;
    private int age;

    // Constructor
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    // Method to display animal details
    public void displayInfo() {
        System.out.println("Animal Name: " + name);
        System.out.println("Animal Age: " + age);
    }

    // Method to make a sound
    public void makeSound() {
        System.out.println("Animal sound");
    }
}

// Derived class
class Dog extends Animal {
    // Additional instance variable for Dog
    private String breed;

    // Constructor
    public Dog(String name, int age, String breed) {
        super(name, age); // Call the constructor of the base class
        this.breed = breed;
    }

    // Overriding the makeSound method
    @Override
    public void makeSound() {
        System.out.println("Bark! Bark!");
    }

    // Method to display dog details
    public void displayDogInfo() {
        displayInfo(); // Call the base class method
        System.out.println("Dog Breed: " + breed);
    }
}

public class SingleInheritanceDemo {
```

```
    public static void main(String[] args) {
        // Create an instance of Dog
        Dog myDog = new Dog("Buddy", 3, "Golden Retriever");

        // Display dog details
        myDog.displayDogInfo();

        // Call the overridden method
        myDog.makeSound();
    }
}
```

```
run:
Animal Name: Buddy
Animal Age: 3
Dog Breed: Golden Retriever
Bark! Bark!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Additional Programs:

Program 1:
Scenario: Banking System
Problem Statement:
You are developing a simple banking system to manage different types of accounts. The base class Account will represent a general bank account with basic attributes, while the SavingsAccount class will represent a specific type of account that has additional features such as interest rate and deposit interest.

Solution

```
// Base class
class Account {
    // Instance variables
    private String accountHolderName;
    private String accountNumber;
    private double balance;

    // Constructor
    public Account(String accountHolderName, String accountNumber, double initialBalance)
{
        this.accountHolderName = accountHolderName;
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }
```

```
    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Deposited: " + amount);
        } else {
            System.out.println("Deposit amount must be positive!");
        }
    }

    // Method to check balance
    public double getBalance() {
        return balance;
    }

    // Method to display account details
    public void displayAccountInfo() {
        System.out.println("\n\n Account Holder: " + accountHolderName);
        System.out.println("Account Number: " + accountNumber);
        System.out.println("Current Balance: " + balance);
    }
}

// Derived class
class SavingsAccount extends Account {
    // Additional instance variable for SavingsAccount
    private double interestRate;

    // Constructor
    public SavingsAccount(String accountHolderName, String accountNumber, double
initialBalance, double interestRate) {
        super(accountHolderName, accountNumber, initialBalance); // Call the base class
constructor
        this.interestRate = interestRate;
    }

    // Method to apply interest to the balance
    public void applyInterest() {
        double interest = getBalance() * interestRate / 100;
        deposit(interest); // Use the deposit method to add interest
        System.out.println("Interest Applied: " + interest);
    }

    // Method to display savings account details
    public void displaySavingsAccountInfo() {
        displayAccountInfo(); // Call the base class method
        System.out.println("Interest Rate: " + interestRate + "%");
    }
}
```

```
public class BankingSystemDemo {
    public static void main(String[] args) {
        // Create an instance of SavingsAccount
        SavingsAccount savingsAccount = new SavingsAccount("Alice Johnson", "SA123456",
1000.0, 5.0);

        // Display savings account details
        savingsAccount.displaySavingsAccountInfo();

        // Deposit money
        savingsAccount.deposit(500.0);

        // Apply interest
        savingsAccount.applyInterest();

        // Display updated account details
        savingsAccount.displaySavingsAccountInfo();
    }
}
```

```
run:


 Account Holder: Alice Johnson
Account Number: SA123456
Current Balance: 1000.0
Interest Rate: 5.0%
Deposited: 500.0
Deposited: 75.0
Interest Applied: 75.0


 Account Holder: Alice Johnson
Account Number: SA123456
Current Balance: 1575.0
Interest Rate: 5.0%
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: WAP to make use of super to call base class constructor

Solution

```
// Base class
class Vehicle {
    private String brand;
    private int year;
```

```
    // Constructor of the base class
    public Vehicle(String brand, int year) {
        this.brand = brand;
        this.year = year;
    }

    // Method to display vehicle details
    public void displayDetails() {
        System.out.println("Brand: " + brand);
        System.out.println("Year: " + year);
    }
}

// Derived class
class Car extends Vehicle {
    private String model;

    // Constructor of the derived class
    public Car(String brand, int year, String model) {
        // Calling the base class constructor
        super(brand, year);
        this.model = model;
    }

    // Method to display car details
    public void displayCarDetails() {
        displayDetails(); // Call method from the base class
        System.out.println("Model: " + model);
    }
}

public class SuperKeywordDemo {
    public static void main(String[] args) {
        // Creating an instance of Car
        Car myCar = new Car("Toyota", 2020, "Camry");

        // Displaying car details
        myCar.displayCarDetails();
    }
}

run:
Brand: Toyota
Year: 2020
Model: Camry
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment 11: Overloading and Overriding

Compulsory:

Program 1: WAP to implement method overloading in Java

Solution

```
// Overloading
// AP to implement method overloading in Java
class SimpleCalculator {

    // Method to add two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Overloaded method to add two doubles
    public double add(double a, double b) {
        return a + b;
    }

    // Overloaded method to add three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

}
public class overloading1 {
    public static void main(String[] args) {
        SimpleCalculator calculator = new SimpleCalculator();

        // Adding two integers
        int sumInt = calculator.add(10, 20);
        System.out.println("Sum of 10 and 20 (integers): " + sumInt);

        // Adding two doubles
        double sumDouble = calculator.add(10.5, 20.3);
        System.out.println("Sum of 10.5 and 20.3 (doubles): " + sumDouble);

        // Adding three integers
        int sumThreeInt = calculator.add(5, 10, 15);
        System.out.println("Sum of 5, 10, and 15 (three integers): " + sumThreeInt);
    }
}
```

```
run:
Sum of 10 and 20 (integers): 30
Sum of 10.5 and 20.3 (doubles): 30.8
Sum of 5, 10, and 15 (three integers): 30
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: WAP to implement method overriding in Java

Solution

```
// WAP to implement method overriding in Java
class Animal {
    // Method to be overridden
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

// Derived class
class Dog extends Animal {
    // Overriding the makeSound method
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class AnimalSoundDemo {
    public static void main(String[] args) {
        // Creating an instance of Animal
        Animal myAnimal = new Animal();
        myAnimal.makeSound(); // Calls the method from the Animal class

        // Creating an instance of Dog
        Animal myDog = new Dog();
        myDog.makeSound(); // Calls the overridden method in the Dog class
    }
}
```

```
run:
Animal makes a sound
Dog barks
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 3:

Scenario: Banking Application
Problem Statement:
You need to implement a banking system that allows customers to make deposits into their accounts using different methods. The program will demonstrate method overloading by providing multiple deposit methods that handle different parameter types.

Solution

```
/*
Scenario: Banking Application
Problem Statement:
You need to implement a banking system that allows customers to make
deposits into their accounts using different methods.
The program will demonstrate method overloading by providing
multiple deposit methods that handle different parameter types.

*/
class Bank {
    private double balance;

    // Constructor to initialize the balance
    public Bank(double initialBalance) {
        this.balance = initialBalance;
    }

    // Method to deposit an integer amount
    public void deposit(int amount) {
        balance += amount;
        System.out.println("Deposited: $" + amount);
    }

    // Method to deposit a double amount
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited: $" + amount);
    }

    // Method to deposit with a description
    public void deposit(double amount, String description) {
        balance += amount;
        System.out.println("Deposited: $" + amount + " - " + description);
    }

    // Method to get the current balance
    public double getBalance() {
        return balance;
    }
}

public class BankDemo1 {
    public static void main(String[] args) {
        // Create an instance of Bank with an initial balance
        Bank myBank = new Bank(1000.0);

        // Using the deposit method for an integer amount
        myBank.deposit(500);
```

```
        // Using the deposit method for a double amount
        myBank.deposit(250.75);

        // Using the deposit method with a description
        myBank.deposit(100.50, "Paycheck deposit");

        // Display the current balance
        System.out.println("Current Balance: $" + myBank.getBalance());
    }
}
```

```
run:
Deposited: $500
Deposited: $250.75
Deposited: $100.5 - Paycheck deposit
Current Balance: $1851.25
BUILD SUCCESSFUL (total time: 0 seconds)
```

Additional Programs:

Program 1: WAP to implement run time polymorphism in java.

Solution

```
// Base class
class Animal1 {
    public void makeSound() {
        System.out.println("Animal makes a sound");
    }
}

// Derived class
class Dog1 extends Animal1 {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class AnimalSoundDemo2 {
    public static void main(String[] args) {
        // Reference of Animal type pointing to Dog object
        Animal1 myAnimal1 = new Animal1();
        Animal1 myAnimal2 = new Dog1();
```

```
        // Calling the makeSound method
        myAnimal1.makeSound(); // This will call the Animal makeSound method
        myAnimal2.makeSound(); // This will call the Dog's makeSound method
    }
}

run:
Animal makes a sound
Dog barks
BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment 12:Abstract and Final

Compulsory:

Program 1: WAP to make use of abstract class in Java

Solution

```
// WAP to make use of abstract class in Java
abstract class Shape {
    // Abstract method
    public abstract double area();
}

// Derived class for Circle
class Circle extends Shape {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Implementing the area method
    @Override
    public double area() {
        return Math.PI * radius * radius; // Area of Circle: πr²
    }
}


public class ShapeAreaDemo {
    public static void main(String[] args) {
        // Creating a Circle object
        Shape myCircle = new Circle(5);
        System.out.println("Area of Circle: " + myCircle.area());
    }
}
```

```
run:
Area of Circle: 78.53981633974483
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2: WAP to make use of final variables in Java

Solution

```
class Constants {
    // Final variable
    public static final double PI = 3.14159;
```

```
    // Final instance variable
    private final String name;

    // Constructor
    public Constants(String name) {
        this.name = name; // Assigning value to final instance variable
    }

    // Method to display the constants
    public void display() {
        System.out.println("Name: " + name);
        System.out.println("Value of PI: " + PI);
    }
}

public class FinalVariablesDemo {
    public static void main(String[] args) {
        // Creating an instance of Constants
        Constants constants = new Constants("Mathematical Constants");

        // Displaying the constants
        constants.display();

        // Attempting to change the final variable (uncommenting the next line will cause a compile error)
        // constants.name = "New Name"; // This will result in an error
        // Constants.PI = 3.14; // This will also result in an error
    }
}
```

```
run:
Name: Mathematical Constants
Value of PI: 3.14159
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 3 :
Scenario: Shape Area and Perimeter Calculation
Problem Statement:
You need to implement a system that can calculate the area and perimeter of different shapes. The abstract class Shape will define the method signatures, and the derived classes Circle and Rectangle will provide their specific implementations.

Solution

```
        // Abstract class
```

```
abstract class Shape {
    // Abstract method to calculate area
    public abstract double calculateArea();

    // Abstract method to calculate perimeter
    public abstract double calculatePerimeter();
}

// Derived class Circle
class Circle extends Shape {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Implementing calculateArea method
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius; // Area of the circle
    }

    // Implementing calculatePerimeter method
    @Override
    public double calculatePerimeter() {
        return 2 * Math.PI * radius; // Circumference of the circle
    }
}

// Derived class Rectangle
class Rectangle extends Shape {
    private double length;
    private double width;

    // Constructor
    public Rectangle(double length, double width) {
        this.length = length;
        this.width = width;
    }

    // Implementing calculateArea method
```

```
    @Override
    public double calculateArea() {
        return length * width; // Area of the rectangle
    }

    // Implementing calculatePerimeter method
    @Override
    public double calculatePerimeter() {
        return 2 * (length + width); // Perimeter of the rectangle
    }
}

public class ShapeDemo {
    public static void main(String[] args) {
        // Create instances of Circle and Rectangle
        Shape myCircle = new Circle(5); // Circle with radius 5
        Shape myRectangle = new Rectangle(4, 6); // Rectangle with length 4 and width 6

        // Display area and perimeter for each shape
        System.out.println("Circle Area: " + myCircle.calculateArea());
        System.out.println("Circle Perimeter: " + myCircle.calculatePerimeter());

        System.out.println("Rectangle Area: " + myRectangle.calculateArea());
        System.out.println("Rectangle Perimeter: " + myRectangle.calculatePerimeter());
    }
}
```

```
run:
Circle Area: 78.53981633974483
Circle Perimeter: 31.41592653589793
Rectangle Area: 24.0
Rectangle Perimeter: 20.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Additional Programs:

Program 1:
Scenario: Banking System
Problem Statement:
You need to implement a banking system that supports different types of accounts. The abstract class Account will define the methods for basic account operations. The derived classes will provide specific implementations for handling deposits and withdrawals.

Solution

```
// Abstract class
abstract class Account {
    protected double balance;

    // Constructor
    public Account(double initialBalance) {
        this.balance = initialBalance;
    }

    // Abstract method to deposit money
    public abstract void deposit(double amount);

    // Abstract method to withdraw money
    public abstract void withdraw(double amount);

    // Method to check the balance
    public double getBalance() {
        return balance;
    }
}

// Derived class SavingsAccount
class SavingsAccount extends Account {
    private double interestRate;

    // Constructor
    public SavingsAccount(double initialBalance, double interestRate) {
        super(initialBalance);
        this.interestRate = interestRate;
    }

    // Implementing deposit method
    @Override
    public void deposit(double amount) {
        balance += amount;
        System.out.println("Deposited $" + amount + " to Savings Account.");
    }

    // Implementing withdraw method
    @Override
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
            System.out.println("Withdrew $" + amount + " from Savings Account.");
        } else {
            System.out.println("Insufficient funds in Savings Account.");
```

```
        }
    }

    // Method to add interest
    public void addInterest() {
        balance += balance * interestRate / 100;
        System.out.println("Interest added to Savings Account.");
    }
}

public class BankingSystemDemo {
    public static void main(String[] args) {
        // Create instances of SavingsAccount and CheckingAccount
        Account savings = new SavingsAccount(1000.00, 2.5); // $1000 balance, 2.5% interest
        // Perform operations on Savings Account
        savings.deposit(200);
        savings.withdraw(150);
        ((SavingsAccount) savings).addInterest(); // Cast to access addInterest method
        System.out.println("Savings Account Balance: $" + savings.getBalance());

        System.out.println(); // Just for better readability in output


    }
}
```

```
run:
Deposited $200.0 to Savings Account.
Withdrew $150.0 from Savings Account.
Interest added to Savings Account.
Savings Account Balance: $1076.25

BUILD SUCCESSFUL (total time: 0 seconds)
```

Experiment 13: Package and Interface

Compulsory:

Program 1: WAP to make use of interfaces in Java

Solution

```
// Define the interface
interface Animal {
    // Abstract methods
    void makeSound();
    void eat();
}

// Implementing the interface in the Dog class
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }

    @Override
    public void eat() {
        System.out.println("Dog eats dog food");
    }
}

// Implementing the interface in the Cat class
class Cat implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Cat meows");
    }

    @Override
    public void eat() {
        System.out.println("Cat eats cat food");
    }
}

public class InterfaceDemo {
    public static void main(String[] args) {
        // Creating instances of Dog and Cat
        Animal myDog = new Dog();
        Animal myCat = new Cat();

        // Calling methods on the Dog object
        myDog.makeSound();
        myDog.eat();
```

```
        // Calling methods on the Cat object
        myCat.makeSound();
        myCat.eat();
    }
}
```

```
run:
Dog barks
Dog eats dog food
Cat meows
Cat eats cat food
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2:

Scenario: Transportation System
Problem Statement:
You need to implement a transportation system that allows different types of vehicles to start and stop. The interface Transportable will define methods for these actions. The classes Car and Bicycle will provide specific implementations for starting and stopping.

Solution

```
// Interface
interface Transportable {
    void start();  // Method to start the vehicle
    void stop();   // Method to stop the vehicle
}

// Class Car implementing Transportable
class Car implements Transportable {
    private String model;

    // Constructor
    public Car(String model) {
        this.model = model;
    }

    // Implementing the start method
    @Override
    public void start() {
        System.out.println("The car " + model + " is starting.");
    }
```

```
    // Implementing the stop method
    @Override
    public void stop() {
        System.out.println("The car " + model + " has stopped.");
    }
}

// Class Bicycle implementing Transportable
class Bicycle implements Transportable {
    private String brand;

    // Constructor
    public Bicycle(String brand) {
        this.brand = brand;
    }

    // Implementing the start method
    @Override
    public void start() {
        System.out.println("The bicycle " + brand + " is now in motion.");
    }

    // Implementing the stop method
    @Override
    public void stop() {
        System.out.println("The bicycle " + brand + " has stopped.");
    }
}

public class TransportationDemo {
    public static void main(String[] args) {
        // Create instances of Car and Bicycle
        Transportable myCar = new Car("Toyota");
        Transportable myBicycle = new Bicycle("Trek");

        // Start and stop the car
        myCar.start();
        myCar.stop();

        System.out.println(); // Just for better readability in output
```

```
        // Start and stop the bicycle
        myBicycle.start();
        myBicycle.stop();
    }
}
```

```
run:
The car Toyota is starting.
The car Toyota has stopped.

The bicycle Trek is now in motion.
The bicycle Trek has stopped.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Additional Programs:

Program 1: WAP to implement multiple inheritance using interfaces in Java

Solution

```
// First interface
interface Animal {
   void makeSound();
}

// Second interface
interface Pet {
   void play();
}

// Class implementing both interfaces
class Dog implements Animal, Pet {
   @Override
   public void makeSound() {
      System.out.println("Dog barks");
   }

   @Override
   public void play() {
      System.out.println("Dog plays fetch");
   }
}
```

```
public class MultipleInheritanceDemo {
   public static void main(String[] args) {
      // Creating an instance of Dog
      Dog myDog = new Dog();

      // Calling methods from both interfaces
      myDog.makeSound();
      myDog.play();
   }
}
```

```
run:
Dog barks
Dog plays fetch
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 2 : WAP to make use of packages in Java

Solution

Step 1: Create the shapes Package

1. **Create a Directory Structure**: Create a directory named `shapes` to hold your package files.
2. **Create the `Circle.java` File**: Inside the `shapes` directory, create a file named `Circle.java`.

```
// File: shapes/Circle.java
package shapes;

public class Circle {
    private double radius;

    // Constructor
    public Circle(double radius) {
        this.radius = radius;
    }

    // Method to calculate the area
    public double area() {
        return Math.PI * radius * radius;
    }

    // Method to display the radius
    public void display() {
        System.out.println("Radius: " + radius);
    }
}
```

Step 2: Create Another Class to Use the `Circle` Class

3. **Create a Main Class**: In the same directory as the `shapes` directory, create a file named `Main.java`.

```
// File: Main.java
import shapes.Circle; // Importing the Circle class from the shapes package

public class Main {
    public static void main(String[] args) {
        // Creating an instance of Circle
        Circle myCircle = new Circle(5.0);

        // Displaying the radius and calculating the area
        myCircle.display();
        System.out.println("Area: " + myCircle.area());
    }
}
```

Step 3: Compile and Run the Program

1. **Compile**: Open a terminal or command prompt and navigate to the directory containing `shapes` and `Main.java`. Then, compile the classes using the following commands:

```
javac shapes/Circle.java
javac Main.java
```

2. **Run**: After compiling, run the `Main` class:

```
java Main
```

Expected Output

When you run the program, you should see the following output:

```
makefile
Copy code
Radius: 5.0
Area: 78.53981633974483
```

Experiment 14: Exception handling and Multithreading

Compulsory:

Program 1: WAP to implement Exception handling in Java.

Solution

```
import java.util.InputMismatchException;
import java.util.Scanner;

public class ExceptionHandlingExample {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        try {
            // User input for numerator and denominator
            System.out.print("Enter numerator: ");
            int numerator = scanner.nextInt();

            System.out.print("Enter denominator: ");
            int denominator = scanner.nextInt();

            // Call the divide method
            double result = numerator / denominator;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            // Handle division by zero
            System.out.println("Error: Cannot divide by zero.");
        } catch (InputMismatchException e) {
            // Handle invalid input (non-integer)
            System.out.println("Error: Please enter valid integers.");
        } catch (Exception e) {
            // Handle any other exceptions
            System.out.println("An unexpected error occurred: " + e.getMessage());
        } finally {
            // Close the scanner resource
            scanner.close();
            System.out.println("Scanner closed.");
        }
    }
}
```

```
run:
Enter numerator: 12
Enter denominator: 0
Error: Cannot divide by zero.
Scanner closed.
BUILD SUCCESSFUL (total time: 4 seconds)
```

```
run:
Enter numerator: 12
Enter denominator: 6
Result: 2.0
Scanner closed.
BUILD SUCCESSFUL (total time: 2 seconds)
```

Program 2: WAP to implement Multithreading in Java

Solution

```
class MyThread1 extends Thread {

   MyThread1(String nm)
   {
      super(nm);
   }
   public void run()
   {
      // Print statement when the thread is called
      for(int i=1;i<=100;i++)
         System.out.println("Thread : " + this.getName() + " i = " + i);
   }
}
public class ThreadDemo1
{
   public static void main(String[] args)
   {
      MyThread1 t1 = new MyThread1("THRD1");
      MyThread1 t2 = new MyThread1("THRD2");

      t1.start();
      t2.start();


   }
}
```

```
run:
Thread : THRD2 i = 1
Thread : THRD2 i = 2
Thread : THRD1 i = 1
Thread : THRD2 i = 3
Thread : THRD2 i = 4
Thread : THRD2 i = 5
Thread : THRD2 i = 6
Thread : THRD1 i = 2
Thread : THRD1 i = 3
Thread : THRD2 i = 7
```

```
Thread : THRD2 i = 8
Thread : THRD2 i = 9
Thread : THRD2 i = 10
Thread : THRD1 i = 4
Thread : THRD1 i = 5
Thread : THRD1 i = 6
Thread : THRD1 i = 7
Thread : THRD1 i = 8
Thread : THRD1 i = 9
Thread : THRD1 i = 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

Program 3 : Scenario: Student Name Management

Problem Statement:
You need to manage a list of student names in an array. The program will allow users to input an index to retrieve the corresponding student's name. If the index is out of bounds, the program will handle the exception gracefully.

Solution

```
import java.util.Scanner;

public class StudentNameManager {
    public static void main(String[] args) {
        // Array of student names
        String[] students = {"Alice", "Bob", "Charlie", "Diana", "Ethan"};

        Scanner scanner = new Scanner(System.in);

        while (true) {
            System.out.print("Enter the index  (0 to " + (students.length - 1) + "), or -1 to exit: ");
            int index = scanner.nextInt();

            // Exit condition
            if (index == -1) {
                System.out.println("Exiting the program.");
                break;
            }

            try {
                // Accessing the student name at the specified index
                String studentName = students[index];
                System.out.println("Student name at index " + index + ": " + studentName);
            } catch (ArrayIndexOutOfBoundsException e) {
                // Handling the exception
                System.out.println("Error: Index " + index + " is out of bounds. Please enter a valid
index.");
```

```
        }
    }

    scanner.close();
  }
}
```

run:
Enter the index (0 to 4), or -1 to exit: 6
Error: Index 6 is out of bounds. Please enter a valid index.
Enter the index (0 to 4), or -1 to exit: 2
Student name at index 2: Charlie
Enter the index (0 to 4), or -1 to exit: -1
Exiting the program.
BUILD SUCCESSFUL (total time: 13 seconds)

Additional Programs: Applets

Program 1: WAP to create and display an Applet in Java

Solution

Java Code for Applet

```
import java.applet.Applet;
import java.awt.*;

// Simple Applet to display a message
public class SimpleApplet extends Applet {
    // Initialization method
    public void init() {
        // Set the background color
        setBackground(Color.lightGray);
        // Set the foreground color
        setForeground(Color.blue);
    }

    // Paint method to display the message
    public void paint(Graphics g) {
        g.drawString("Welcome to Java Applet!", 20, 20);
    }
}
```

HTML Code to Run the Applet

To run this applet, you'll also need an HTML file that will load the applet. Create a file named `applet.html` with the following content:

```
<!DOCTYPE html>
<html>
<head>
    <title>Simple Applet Example</title>
</head>
```

```
<body>
    <h1>Java Applet Example</h1>
    <applet code="SimpleApplet.class" width="300" height="100">
        Your browser does not support Java applets.
    </applet>
</body>
</html>
```

Steps to Compile and Run the Applet

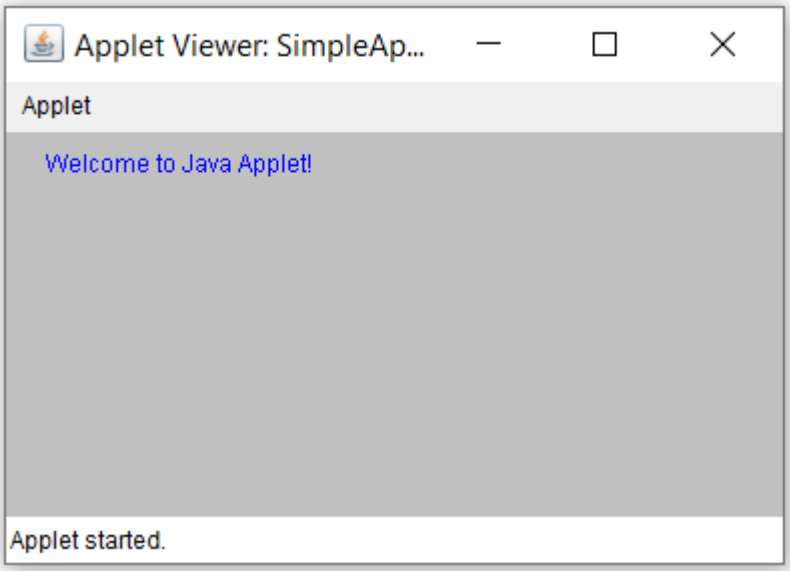
1. **Compile the Applet**:
    - Save the Java code in a file named `SimpleApplet.java`.
    - Open a terminal or command prompt and navigate to the directory where the file is located.
    - Compile the applet using the following command:

      ```
      javac SimpleApplet.java
      ```

2. **Run the Applet**:
    - You can use the applet viewer to run your applet. Use the following command:

      ```
      appletviewer applet.html
      ```

Output



Program 2: WAP to show Indian Flag on an Applet.

Solution

Java Code for Indian Flag Applet

```
import java.applet.Applet;
import java.awt.*;

// Applet to draw the Indian National Flag
public class IndianFlagApplet extends Applet {

    @Override
    public void init() {
        // Set the background color to white
        setBackground(Color.white);
    }

    @Override
    public void paint(Graphics g) {
        // Define the dimensions of the flag
        int width = 300;
        int height = 200;

        // Draw the saffron (top) rectangle
        g.setColor(new Color(255, 153, 51)); // Saffron color
        g.fillRect(0, 0, width, height / 3);

        // Draw the white (middle) rectangle
        g.setColor(Color.white);
        g.fillRect(0, height / 3, width, height / 3);

        // Draw the green (bottom) rectangle
        g.setColor(new Color(0, 128, 0)); // Green color
        g.fillRect(0, 2 * height / 3, width, height / 3);

        // Draw the Ashoka Chakra
        g.setColor(Color.blue);
        g.drawOval(width / 2 - 30, height / 3 - 15, 60, 30); // Outer
circle
        g.drawOval(width / 2 - 25, height / 3 - 10, 50, 20); // Inner
circle

        // Draw the spokes of the Ashoka Chakra
        for (int i = 0; i < 24; i++) {
            double angle = Math.toRadians(i * 15);
            int x1 = (int) (width / 2 + 25 * Math.cos(angle));
            int y1 = (int) (height / 3 - 10 + 15 * Math.sin(angle));
            int x2 = (int) (width / 2 + 30 * Math.cos(angle));
            int y2 = (int) (height / 3 - 10 + 30 * Math.sin(angle));
            g.drawLine(x1, y1, x2, y2);
        }
    }
}
```

HTML Code to Run the Applet

Create an HTML file named `applet.html` to display the applet:

```
html
Copy code
<!DOCTYPE html>
<html>
<head>
    <title>Indian Flag Applet</title>
</head>
<body>
```

```
    <h1>Indian National Flag</h1>
    <applet code="IndianFlagApplet.class" width="300" height="200">
        Your browser does not support Java applets.
    </applet>
</body>
</html>
```

Steps to Compile and Run the Applet

1. **Compile the Applet**:
   - Save the Java code in a file named `IndianFlagApplet.java`.
   - Open a terminal or command prompt and navigate to the directory where the file is located.
   - Compile the applet using the following command:

     ```
     javac IndianFlagApplet.java
     ```

2. **Run the Applet**:
   - You can use the applet viewer to run your applet. Use the following command:

     ```
     appletviewer applet.html
     ```

Output