/*1. Develop a JAVA program to add TWO matrices of suitable order N (The value of N should be read from
command line arguments). */

```
import java.util.Scanner;
public class MatrixAddition {

    public static void main(String[] args) {
        if (args.length != 1) {
            System.out.println("Please provide the value of N as a
command line argument.");
            return;
        }

        int N = Integer.parseInt(args[0]);
        int[][] matrix1 = new int[N][N];
        int[][] matrix2 = new int[N][N];
         Scanner scanner = new Scanner(System.in);
        // Populate matrix1 and matrix2 with random values for
demonstration
        // You can modify this part to input values from the user or
any other source
        // For simplicity, we use random values between 1 and 10



      /*  //Reading matrix using random
         for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                matrix1[i][j] = (int) (Math.random() * 10) + 1;
                matrix2[i][j] = (int) (Math.random() * 10) + 1;
                            }
        }*/



        // Reading the Matrix-1
         System.out.println("Enter the Elements of Matrix 1:");

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                //matrix1[i][j] = (int) (Math.random() * 10) + 1;
                //matrix2[i][j] = (int) (Math.random() * 10) + 1;
                matrix1[i][j]=scanner.nextInt();
            }
        }

         // Reading the Matrix-2
         System.out.println("Enter the Elements of Matrix 2:");
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                //matrix1[i][j] = (int) (Math.random() * 10) + 1;
                //matrix2[i][j] = (int) (Math.random() * 10) + 1;
                matrix2[i][j]=scanner.nextInt();
            }
        }
```

```
        // Perform matrix addition
        int[][] result = new int[N][N];
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }

        // Display the matrices and their sum
        System.out.println("Matrix 1:");
        printMatrix(matrix1);
        System.out.println("Matrix 2:");
        printMatrix(matrix2);
        System.out.println("Sum of the matrices:");
        printMatrix(result);
    }

    // Helper method to print a matrix
    private static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            for (int num : row) {
                System.out.print(num + "\t");
            }
            System.out.println();
        }
        System.out.println();
    }
}
/*
```

2. Develop a stack class to hold a maximum of 10 integers with suitable methods. Develop a JAVA main

method to illustrate Stack operations.

```
*/


import java.util.Scanner;
class Stack {
    private int[] elements;
    private int top;

    public Stack() {
        elements = new int[10];
        top = -1;
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == 9;
    }

    public void push(int element) {
```

```
        if (isFull()) {
            System.out.println("Stack is full. Cannot push more
elements.");
        } else {
            elements[++top] = element;
            System.out.println("Pushed: " + element);
        }
    }

    public void pop() {
        if (isEmpty()) {
            System.out.println("Stack is empty. Cannot pop elements.");
        } else {
            int poppedElement = elements[top--];
            System.out.println("Popped: " + poppedElement);
        }
    }

    public void printStack() {
        if (isEmpty()) {
            System.out.println("Stack is empty.");
        } else {
            System.out.print("Stack: ");
            for (int i = 0; i <= top; i++) {
                System.out.print(elements[i] + " ");
            }
            System.out.println();
        }
    }
}

public class Main {
    public static void main(String[] args) {
        Stack stack = new Stack();
        while(true)
        {
        System.out.println("Stack Operations");
         System.out.println("1. Push");
          System.out.println("2. Pop");
           System.out.println("3. Display");
           System.out.println("4. Exit");
            Scanner scanner = new Scanner(System.in);
             System.out.print("Enter your Choice: ");
        int choice = scanner.nextInt();

           switch(choice)
           {
               case 1:  System.out.println("Enter Number to push: ");
                         int num = scanner.nextInt();
                         stack.push(num);
                         break;
               case 2:
                         stack.pop();
                         break;
                case 3: stack.printStack();
                        break;
                        case 4:  System.exit(0);
                                break;
```

```
                default: System.out.println("Invalid choice ");
            }
        }
    }
}
/*
```

3. A class called Employee, which models an employee with an ID, name and salary, is designed as shown in
the following class diagram. The method raiseSalary (percent) increases the salary by the given
percentage. Develop the Employee class and suitable main method for demonstration.

```
*/


import java.util.Scanner;
public class Employee {
    private int empId;
    private String name;
    private double salary;

    public Employee(int empId, String name, double salary) {
        this.empId = empId;
        this.name = name;
        this.salary = salary;
    }

    public void raiseSalary(double percentage) {
        if (percentage > 0) {
            double raiseAmount = salary * (percentage / 100);
            salary += raiseAmount;
        }
    }

    public void displayInfo() {
        System.out.println("Employee ID: " + empId);
        System.out.println("Name: " + name);
        System.out.println("Salary: Rs." + String.format("%.2f",
salary));
    }

    public static void main(String[] args) {
        // Creating an Employee object
        Employee emp = new Employee(1, "Dr. Harish Kumar B T",
50000.0);
         Scanner scanner = new Scanner(System.in);
        // Displaying employee information before raise
        System.out.println("Employee information before raise:");
        emp.displayInfo();
          System.out.println("Enter the percentage of salary to
raise");
          int percentage = scanner.nextInt();
        // Raising salary by 10%
        emp.raiseSalary(percentage);
```

```
        // Displaying employee information after raise
        System.out.println("\nEmployee information after raise:");
        emp.displayInfo();
    }
}
/*
```

4. A class called MyPoint, which models a 2D point with x and y coordinates, is designed as follows:

--Two instance variables x (int) and y (int).

-- A default (or "no-arg") constructor that construct a point at the default location of (0, 0).

-- A overloaded constructor that constructs a point with the given x and y coordinates.

-- A method setXY() to set both x and y.

-- A method getXY() which returns the x and y in a 2-element int array.

-- A toString() method that returns a string description of the instance in the format "(x, y)".

-- A method called distance(int x, int y) that returns the distance from this point to another point at the

given (x, y) coordinates

-- An overloaded distance(MyPoint another) that returns the distance from this point to the given

MyPoint instance (called another)

-- Another overloaded distance() method that returns the distance from this point to the origin (0,0)

Develop the code for the class MyPoint. Also develop a JAVA program (called TestMyPoint) to test all the

methods defined in the class.

```
*/


public class MyPoint {
    private int x;
    private int y;

    // Default constructor
    public MyPoint() {
        this.x = 0;
        this.y = 0;
    }

    // Overloaded constructor
    public MyPoint(int x, int y) {
        this.x = x;
        this.y = y;
    }

    // Setters for x and y
    public void setXY(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

```
    // Getter for x and y
    public int[] getXY() {
        int[] coordinates = {x, y};
        return coordinates;
    }

    // Returns the string description of the instance in the format
"(x, y)"
    @Override
    public String toString() {
        return "(" + x + ", " + y + ")";
    }

    // Calculates distance from this point to another point (x, y)
    public double distance(int x, int y) {
        int xDiff = this.x - x;
        int yDiff = this.y - y;
        return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
    }

    // Calculates distance from this point to another MyPoint instance
    public double distance(MyPoint another) {
        int xDiff = this.x - another.x;
        int yDiff = this.y - another.y;
        return Math.sqrt(xDiff * xDiff + yDiff * yDiff);
    }

    // Calculates distance from this point to the origin (0, 0)
    public double distance() {
        return Math.sqrt(x * x + y * y);
    }

    public static void main(String[] args) {
        MyPoint point1 = new MyPoint(); // Default constructor (0,0)
        System.out.println("Point 1: " + point1);

        MyPoint point2 = new MyPoint(3, 4); // Overloaded constructor
(3,4)
        System.out.println("Point 2: " + point2);

        point1.setXY(5, 6); // Set coordinates using setXY() method
        System.out.println("Point 1 after setXY(): " + point1);

        int[] coordinates = point2.getXY(); // Get coordinates using
getXY() method
        System.out.println("Point 2 coordinates: (" + coordinates[0] +
", " + coordinates[1] + ")");

        System.out.println("Distance between Point 1 and (5, 6): " +
point1.distance(5, 6));
        System.out.println("Distance between Point 1 and Point 2: " +
point1.distance(point2));
        System.out.println("Distance from Point 1 to origin: " +
point1.distance());
    }
}
/*
```

5. Develop a JAVA program to create a class named shape. Create three sub classes namely: circle, triangle
and square, each class has two member functions named draw () and erase (). Demonstrate polymorphism concepts
by developing suitable methods, defining member data and main program.

```
*/
// Shape class (Superclass)
class Shape {
    public void draw() {
        System.out.println("Drawing a shape");
    }

    public void erase() {
        System.out.println("Erasing a shape");
    }
}

// Circle class (Subclass)
class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle");
    }

    @Override
    public void erase() {
        System.out.println("Erasing a circle");
    }
}

// Triangle class (Subclass)
class Triangle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a triangle");
    }

    @Override
    public void erase() {
        System.out.println("Erasing a triangle");
    }
}

// Square class (Subclass)
class Square extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a square");
    }

    @Override
    public void erase() {
        System.out.println("Erasing a square");
    }
}
```

```
// Main class
public class Main {
    public static void main(String[] args) {
        // Polymorphism: Creating objects of different subclasses using
the reference of the superclass
        Shape shape1 = new Circle();
        Shape shape2 = new Triangle();
        Shape shape3 = new Square();

        // Demonstrating polymorphic behavior
        shape1.draw(); // Calls draw() method of Circle class
        shape1.erase(); // Calls erase() method of Circle class

        shape2.draw(); // Calls draw() method of Triangle class
        shape2.erase(); // Calls erase() method of Triangle class

        shape3.draw(); // Calls draw() method of Square class
        shape3.erase(); // Calls erase() method of Square class
    }
}
```