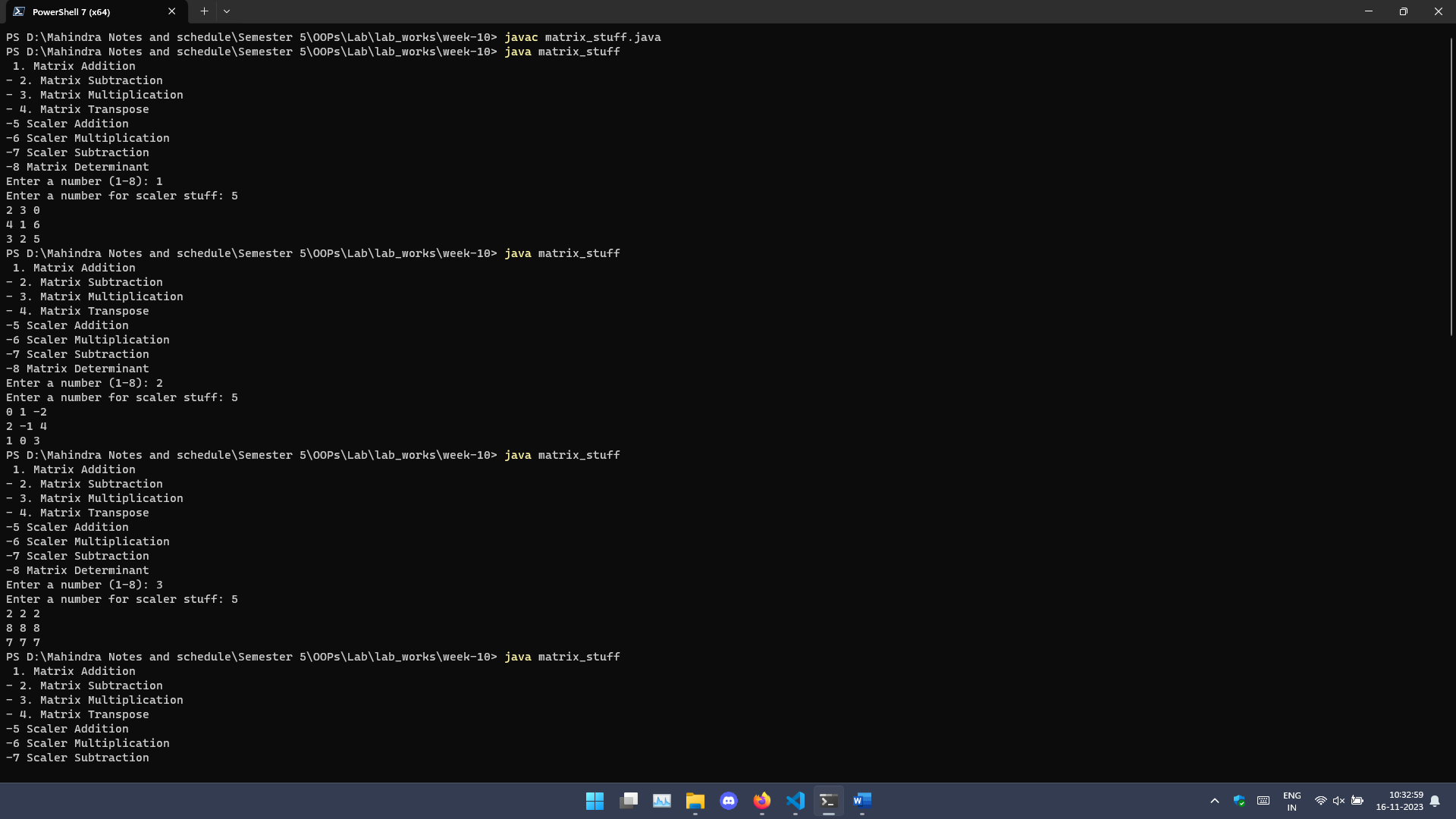
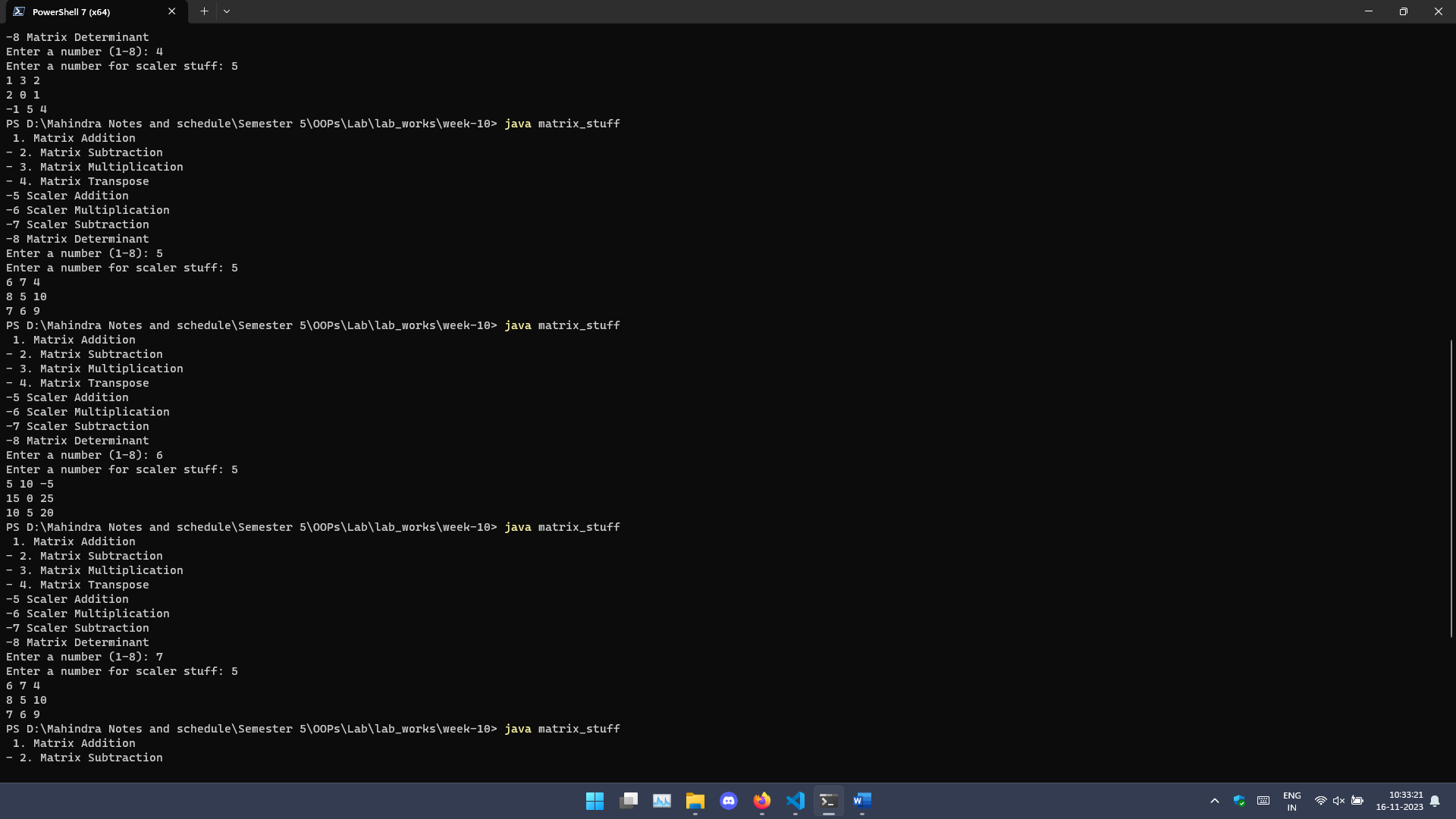
**Report OOP Matrices**

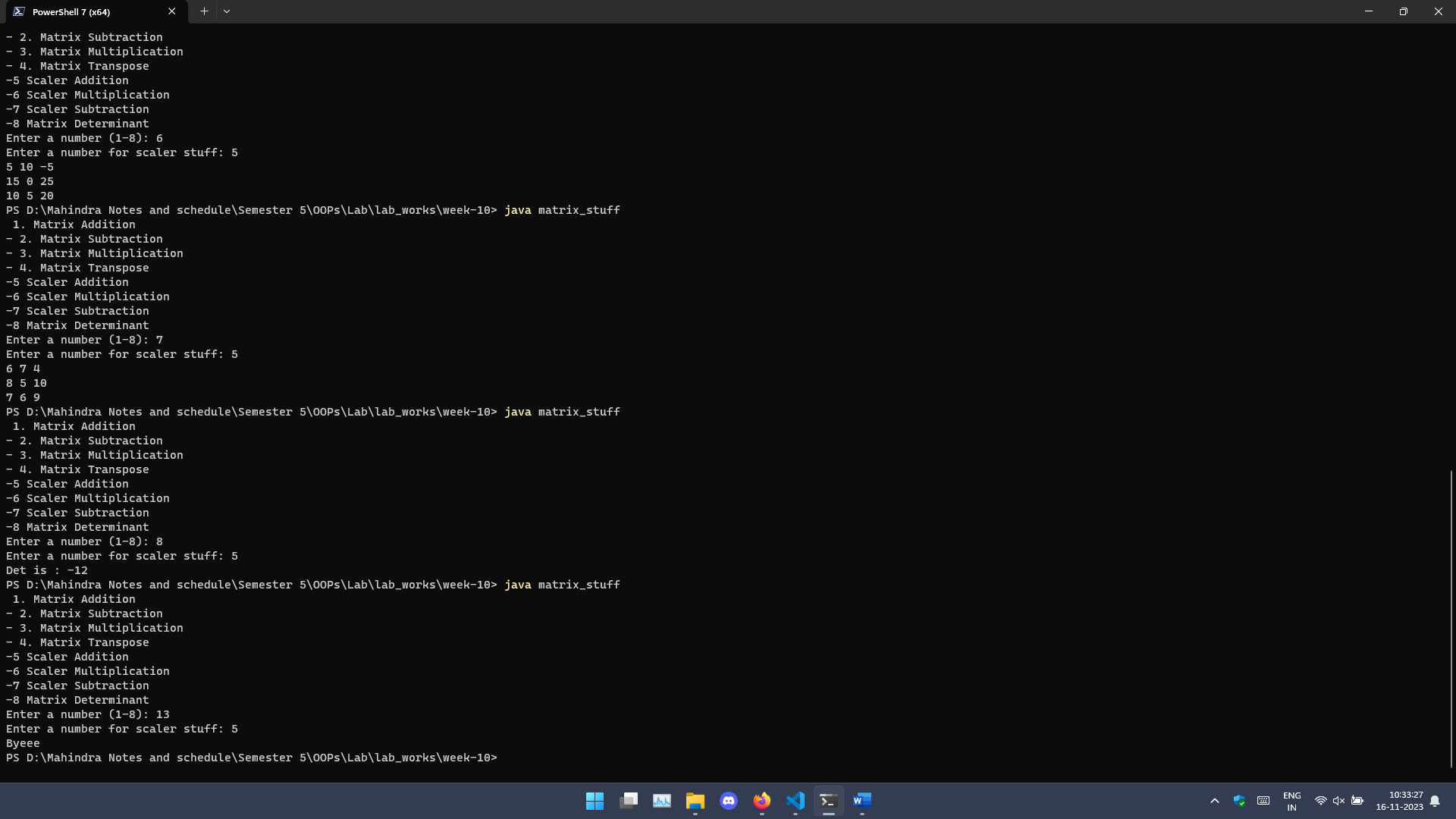
**Shashank K**

**SE21UCSE198**

**Output**

****

****

****

**Program:**

import java.util.**\***;

class Matrix {

    public int[][] subtraction(int *mat1*[][], int *mat2*[][]) {

        int[][] answer = new int[*mat1*.length][*mat1*[0].length];

        for (int i = 0; i < *mat2*.length; i++) {

            for (int j = 0; j < *mat2*[0].length; j++) {

                answer[i][j] = *mat1*[i][j] - *mat2*[i][j];

            }

        }

        return answer;

    }

    public int[][] addition(int *mat1*[][], int *mat2*[][]) {

        int[][] answer = new int[*mat1*.length][*mat1*[0].length];

        for (int i = 0; i < *mat2*.length; i++) {

            for (int j = 0; j < *mat2*[0].length; j++) {

                answer[i][j] = *mat1*[i][j] + *mat2*[i][j];

            }

        }

        return answer;

    }

    public int[][] scalar\_multiplication(int *mat1*[][], int *a*) {

        int[][] answer = new int[*mat1*.length][*mat1*[0].length];

        for (int i = 0; i < *mat1*.length; i++) {

            for (int j = 0; j < *mat1*[0].length; j++) {

                answer[i][j] = *mat1*[i][j] \* *a*;

            }

        }

        return answer;

    }

    public int[][] scalar\_addition(int *mat1*[][], int *a*) {

        int[][] answer = new int[*mat1*.length][*mat1*[0].length];

        for (int i = 0; i < *mat1*.length; i++) {

            for (int j = 0; j < *mat1*[0].length; j++) {

                answer[i][j] = *mat1*[i][j] + *a*;

            }

        }

        return answer;

    }

    public int[][] scalar\_subtraction(int *mat1*[][], int *a*) {

        int[][] answer = new int[*mat1*.length][*mat1*[0].length];

        for (int i = 0; i < *mat1*.length; i++) {

            for (int j = 0; j < *mat1*[0].length; j++) {

                answer[i][j] = *mat1*[i][j] - *a*;

            }

        }

        return answer;

    }

    public int[][] transposition(int *mat1*[][]) {

        int[][] answer = new int[*mat1*[0].length][*mat1*.length];

        for (int i = 0; i < *mat1*.length; i++) {

            for (int j = 0; j < *mat1*[0].length; j++) {

                answer[j][i] = *mat1*[i][j];

            }

        }

        return answer;

    }

    public int[][] multiplication(int *mat1*[][], int *mat2*[][]) {

        int[][] answer = new int[*mat1*.length][*mat2*[0].length];

        for (int i = 0; i < *mat1*.length; i++) {

            for (int j = 0; j < *mat2*[0].length; j++) {

                for (int k = 0; k < *mat2*.length; k++) {

                    answer[i][j] += *mat1*[i][k] \* *mat2*[k][j];

                }

            }

        }

        return answer;

    }

    public int determinant(int *mat*[][])

    {

        int n = *mat*.length;

        int num1, num2, det = 1, index,

                        total = 1; *// Initialize result*

*// temporary array for storing row*

        int[] temp = new int[n + 1];

*// loop for traversing the diagonal elements*

        for (int i = 0; i < n; i++) {

            index = i; *// initialize the index*

            while (index < n && *mat*[index][i] == 0 ) {

                index++;

            }

            if (index == n)

            {

                continue;

            }

            if (index != i) {

                for (int j = 0; j < n; j++) {

                    swap(*mat*, index, j, i, j);

                }

                det = (int)(det \* Math.pow(-1, index - i));

            }

*// storing the values of diagonal row elements*

            for (int j = 0; j < n; j++) {

                temp[j] = *mat*[i][j];

            }

            for (int j = i + 1; j < n; j++) {

                num1 = temp[i]; *// value of diagonal element*

                num2 = *mat*[j]

                          [i]; *// value of next row element*

                for (int k = 0; k < n; k++) {

*// multiplying to make the diagonal*

*// element and next row element equal*

*mat*[j][k] = (num1 \* *mat*[j][k])

                                - (num2 \* temp[k]);

                }

                total = total \* num1; *// Det(kA)=kDet(A);*

            }

        }

*// multiplying the diagonal elements to get*

*// determinant*

        for (int i = 0; i < n; i++) {

            det = det \* *mat*[i][i];

        }

        return (det / total); *// Det(kA)/k=Det(A);*

    }

    static int[][] swap(int[][] *arr*, int *i1*, int *j1*, int *i2*,int *j2*)

    {

        int temp = *arr*[*i1*][*j1*];

*arr*[*i1*][*j1*] = *arr*[*i2*][*j2*];

*arr*[*i2*][*j2*] = temp;

        return *arr*;

    }

    static void getCofactor(int *A*[][], int *temp*[][], int *p*, int *q*, int *n*)

{

    int i = 0, j = 0;

*// Looping for each element of the matrix*

    for (int row = 0; row < *n*; row++)

    {

        for (int col = 0; col < *n*; col++)

        {

*// Copying into temporary matrix only those element*

*// which are not in given row and column*

            if (row != *p* && col != *q*)

            {

*temp*[i][j++] = *A*[row][col];

*// Row is filled, so increase row index and*

*// reset col index*

                if (j == *n* - 1)

                {

                    j = 0;

                    i++;

                }

            }

        }

    }

}

public int[][] adjoint(int *A*[][])

{

    int N = *A*.length;

    int[][] adj = new int[N][N];

*// temp is used to store cofactors of A[][]*

    int sign = 1;

    int [][]temp = new int[N][N];

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < N; j++)

        {

*// Get cofactor of A[i][j]*

            getCofactor(*A*, temp, i, j, N);

*// sign of adj[j][i] positive if sum of row*

*// and column indexes is even.*

            sign = ((i + j) % 2 == 0)? 1: -1;

*// Interchanging rows and columns to get the*

*// transpose of the cofactor matrix*

            adj[j][i] = (sign)\*(**this**.determinant(temp));

        }

    }

    return adj;

}

public float[][] inverse(int *A*[][])

{

*// Find determinant of A[][]*

    int det = **this**.determinant(*A*);

    int N = *A*.length;

    float[][] inverse = new float[N][N];

    if (det == 0)

    {

        System.out.print("Singular matrix, can't find its inverse");

    }

*// Find adjoint*

    int [][]adj = **this**.adjoint(*A*);

    for (int i = 0; i < N; i++)

        for (int j = 0; j < N; j++)

            inverse[i][j] = adj[i][j]/(float)det;

    return inverse;

}

    public void print\_matrix(int[][] *answer*) {

        for (int i = 0; i < *answer*.length; i++) {

            for (int j = 0; j < *answer*[0].length; j++) {

                System.out.print(*answer*[i][j] + " ");

            }

            System.out.println("");

        }

    }

}

public class matrix\_stuff {

    public static void main(String[] *args*) {

        Matrix potato = new Matrix();

        int[][] a = { { 1,2, -1 },

        { 3, 0, 5 },

        { 2, 1, 4 }};

        int[][] b= { { 1, 1, 1 }, { 1, 1, 1 }, { 1, 1, 1 }};

        Scanner scanner = new Scanner(System.in);

        System.out.println(" 1. Matrix Addition\n- 2. Matrix Subtraction\n- 3. Matrix Multiplication\n- 4. Matrix Transpose\n-5 Scaler Addition\n-6 Scaler Multiplication\n-7 Scaler Subtraction\n-8 Matrix Determinant");

        System.out.print("Enter a number (1-8): ");

        int choice = scanner.nextInt();

        System.out.print("Enter a number for scaler stuff: ");

        int num = scanner.nextInt();

        scanner.close();

        switch (choice) {

            case 1:

                int[][] answer = potato.addition(a, b);

                potato.print\_matrix(answer);

                break;

            case 2:

                int[][] answer1 = potato.subtraction(a, b);

                potato.print\_matrix(answer1);

                break;

            case 3:

                int[][] answer2 = potato.multiplication(a, b);

                potato.print\_matrix(answer2);

                break;

            case 4:

                int[][] answer3 = potato.transposition(a);

                potato.print\_matrix(answer3);

                break;

            case 5:

*// Scanner scanner1 = new Scanner(System.in);*

*// System.out.print("Enter a number ");*

*// int choice1 = scanner1.nextInt();*

                int[][] answer4 = potato.scalar\_addition(a, num);

                potato.print\_matrix(answer4);

*// scanner1.close();*

                break;

            case 6:

*// System.out.println("Enter a number ");*

*// Scanner scanner2 = new Scanner(System.in);*

*// int choice2 = scanner2.nextInt();*

                int[][] answer5 = potato.scalar\_multiplication(a, num);

                potato.print\_matrix(answer5);

*// scanner2.close();*

                break;

            case 7:

*// Scanner scanner3 = new Scanner(System.in);*

*// System.out.print("Enter a number ");*

*//         int choice3 = scanner3.nextInt();*

                int[][] answer6= potato.scalar\_addition(a, num);

                potato.print\_matrix(answer6);

*// scanner3.close();*

                break;

            case 8:

                int det = potato.determinant(a);

                System.out.print("Det is : ");

                System.out.println(det);

                break;

            default:

            System.out.println("Byeee");

                            break;

        }

    }

}