Report

Nithish Raghav

SE21UCSE093

Output

```
[6, 8]
[10, 12]



Matrix Subtraction:
[-4, -4]
[-4, -4]


Scalar Multiplication:
[2, 4]
[6, 8]


Scalar Addition:
[3, 4]
[5, 6]


Scalar Subtraction:
[-1, 0]
[1, 2]


Matrix Multiplication:
[19, 22]
[43, 50]


Matrix Transposition:
[1, 3]
[2, 4]


Matrix Determinant: -2

Matrix Inversion:
[0, 0]
[0, 0]
```

Program

```
import java.util.Arrays;


public class MatrixOperations {

    public static int[][] addMatrices(int[][] matrix1, int[][] matrix2) {
        int rows = matrix1.length;
        int cols = matrix1[0].length;
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = matrix1[i][j] + matrix2[i][j];
            }
        }

        return result;
    }

    public static int[][] subtractMatrices(int[][] matrix1, int[][] matrix2) {
        int rows = matrix1.length;
        int cols = matrix1[0].length;
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = matrix1[i][j] - matrix2[i][j];
            }
        }

        return result;
    }

    public static int[][] scalarMultiply(int scalar, int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = scalar * matrix[i][j];
            }
        }

        return result;
    }

    public static int[][] scalarAddition(int scalar, int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
```

```
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = scalar + matrix[i][j];
            }
        }

        return result;
    }

    public static int[][] scalarSubtraction(int scalar, int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] result = new int[rows][cols];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[i][j] = matrix[i][j] - scalar;
            }
        }

        return result;
    }

    public static int[][] multiplyMatrices(int[][] matrix1, int[][] matrix2) {
        int rows1 = matrix1.length;
        int cols1 = matrix1[0].length;
        int rows2 = matrix2.length;
        int cols2 = matrix2[0].length;

        if (cols1 != rows2) {
            throw new IllegalArgumentException("Matrix dimensions are not valid for
multiplication");
        }

        int[][] result = new int[rows1][cols2];

        for (int i = 0; i < rows1; i++) {
            for (int j = 0; j < cols2; j++) {
                for (int k = 0; k < cols1; k++) {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }

        return result;
    }

    public static int[][] transposeMatrix(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] result = new int[cols][rows];
```

```
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                result[j][i] = matrix[i][j];
            }
        }

        return result;
    }

    public static int determinant(int[][] matrix) {
        int n = matrix.length;

        if (n != matrix[0].length) {
            throw new IllegalArgumentException("Matrix is not square");
        }

        if (n == 1) {
            return matrix[0][0];
        }

        if (n == 2) {
            return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0];
        }

        int det = 0;

        for (int i = 0; i < n; i++) {
            det += (i % 2 == 0 ? 1 : -1) * matrix[0][i] *
determinant(getSubMatrix(matrix, 0, i));
        }

        return det;
    }

    public static int[][] inverseMatrix(int[][] matrix) {
        int n = matrix.length;

        if (n != matrix[0].length) {
            throw new IllegalArgumentException("Matrix is not square");
        }

        int det = determinant(matrix);

        if (det == 0) {
            throw new IllegalArgumentException("Matrix is singular, cannot find
inverse");
        }

        int[][] adjugate = new int[n][n];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                adjugate[i][j] = (int) Math.pow(-1, i + j) *
determinant(getSubMatrix(matrix, i, j));
```

```
            }
        }

        int[][] inverse = scalarMultiply(1 / det, transposeMatrix(adjugate));

        return inverse;
    }

    private static int[][] getSubMatrix(int[][] matrix, int rowToRemove, int
colToRemove) {
        int n = matrix.length;
        int[][] subMatrix = new int[n - 1][n - 1];

        int newRow = 0;
        int newCol;

        for (int i = 0; i < n; i++) {
            if (i != rowToRemove) {
                newCol = 0;
                for (int j = 0; j < n; j++) {
                    if (j != colToRemove) {
                        subMatrix[newRow][newCol] = matrix[i][j];
                        newCol++;
                    }
                }
                newRow++;
            }
        }

        return subMatrix;
    }

    public static void main(String[] args) {
        int[][] matrix1 = {{1, 2}, {3, 4}};
        int[][] matrix2 = {{5, 6}, {7, 8}};
        int scalar = 2;

        // Matrix Addition
        int[][] resultAddition = addMatrices(matrix1, matrix2);
        System.out.println("Matrix Addition:");
        printMatrix(resultAddition);

        // Matrix Subtraction
        int[][] resultSubtraction = subtractMatrices(matrix1, matrix2);
        System.out.println("\nMatrix Subtraction:");
        printMatrix(resultSubtraction);

        // Scalar Multiplication
        int[][] resultScalarMultiply = scalarMultiply(scalar, matrix1);
        System.out.println("\nScalar Multiplication:");
        printMatrix(resultScalarMultiply);

        // Scalar Addition
        int[][] resultScalarAddition = scalarAddition(scalar, matrix1);
```

```
        System.out.println("\nScalar Addition:");
        printMatrix(resultScalarAddition);

        // Scalar Subtraction
        int[][] resultScalarSubtraction = scalarSubtraction(scalar, matrix1);
        System.out.println("\nScalar Subtraction:");
        printMatrix(resultScalarSubtraction);

        // Matrix Multiplication
        int[][] resultMultiplication = multiplyMatrices(matrix1, matrix2);
        System.out.println("\nMatrix Multiplication:");
        printMatrix(resultMultiplication);

        // Matrix Transposition
        int[][] resultTranspose = transposeMatrix(matrix1);
        System.out.println("\nMatrix Transposition:");
        printMatrix(resultTranspose);

        // Matrix Determinant
        int determinantValue = determinant(matrix1);
        System.out.println("\nMatrix Determinant: " + determinantValue);

        // Matrix Inversion
        try {
            int[][] resultInverse = inverseMatrix(matrix1);
            System.out.println("\nMatrix Inversion:");
            printMatrix(resultInverse);
        } catch (IllegalArgumentException e) {
            System.out.println("\nMatrix Inversion not possible: " + e.getMessage());
        }
    }

    private static void printMatrix(int[][] matrix) {
        for (int[] row : matrix) {
            System.out.println(Arrays.toString(row));
        }
        System.out.println();
    }
}
```