**Name:Nurul Kawsar**

**ID: 24070144**

**1. Print a 2D Array**

**Explanation:**

A 2D array is a collection of rows and columns. To print all elements, we use nested loops:

The outer loop iterates over rows.

The inner loop iterates over columns within each row.

**Code:**

public class Print2DArray {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

System.out.println("2D Array Elements:");

for (int i = 0; i < arr.length; i++) { // Loop through rows

for (int j = 0; j < arr[i].length; j++) { // Loop through columns

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

}

System.out.println();

}

}

}

**Output**:

2D Array Elements:

1 2 3

4 5 6

7 8 9

---

**2. Sum of Elements in a 2D Array**

**Explanation:**

We iterate through the 2D array and maintain a variable sum to accumulate all values.

Code:

public class SumOfElements {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int sum = 0;

for (int[] row : arr) { // Iterate over rows

for (int num : row) { // Iterate over elements in a row

sum += num;

}

}

System.out.println("Sum of all elements: " + sum);

}

}

**Output:**

Sum of all elements: 45

---

**3. Find Maximum and Minimum Elements**

**Explanation:**

Initialize max and min with the first element of the matrix.

Traverse the matrix and update max if a larger element is found.

Update min if a smaller element is found.

**Code:**

public class FindMaxMin {

public static void main(String[] args) {

int[][] arr = {

{10, 2, 3},

{4, 50, 6},

{7, 8, 90}

};

int max = arr[0][0], min = arr[0][0];

for (int[] row : arr) {

for (int num : row) {

if (num > max) max = num;

if (num < min) min = num;

}

}

System.out.println("Maximum: " + max);

System.out.println("Minimum: " + min);

}

}

**Output**:

Maximum: 90

Minimum: 2

---

**4. Row-wise and Column-wise Sum**

**Explanation:**

Iterate through rows and sum the values.

Iterate through columns and sum the values.

**Code:**

public class RowColSum {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

// Row-wise sum

System.out.println("Row-wise sum:");

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

int rowSum = 0;

for (int j = 0; j < arr[i].length; j++) {

rowSum += arr[i][j];

}

System.out.println("Row " + (i+1) + ": " + rowSum);

}

// Column-wise sum

System.out.println("Column-wise sum:");

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

int colSum = 0;

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

colSum += arr[i][j];

}

System.out.println("Column " + (j+1) + ": " + colSum);

}

}

}

**Output**:

Row-wise sum:

Row 1: 6

Row 2: 15

Row 3: 24

Column-wise sum:

Column 1: 12

Column 2: 15

Column 3: 18

---

**5. Transpose of a Matrix**

**Explanation**:

The transpose of a matrix swaps its rows and columns.

**Code**:

public class TransposeMatrix {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int rows = arr.length, cols = arr[0].length;

int[][] transposed = new int[cols][rows];

for (int i = 0; i < rows; i++) {

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

transposed[j][i] = arr[i][j];

}

}

System.out.println("Transpose:");

for (int[] row : transposed) {

for (int num : row) {

System.out.print(num + " ");

}

System.out.println();

}

}

}

**Output:**

Transpose:

1 4 7

2 5 8

3 6 9

---

**6. Matrix Addition**

**Explanation**:

Add corresponding elements of two matrices.

**Code:**

public class MatrixAddition {

public static void main(String[] args) {

int[][] A = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int[][] B = {

{9, 8, 7},

{6, 5, 4},

{3, 2, 1}

};

int[][] C = new int[3][3];

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

C[i][j] = A[i][j] + B[i][j];

}

}

System.out.println("Matrix Addition Result:");

for (int[] row : C) {

for (int num : row) {

System.out.print(num + " ");

}

System.out.println();

}

}

}

**Output:**

Matrix Addition Result:

10 10 10

10 10 10

10 10 10

---

**7. Matrix Multiplication**

**Explanation:**

Matrix multiplication follows the rule:

If A is an m × n matrix and B is an n × p matrix, the resultant matrix C will be m × p.

Each element in the resulting matrix is computed as:

C[i][j] = A[i][0] \* B[0][j] + A[i][1] \* B[1][j] + ... + A[i][n-1] \* B[n-1][j]

Outer loop: Traverses rows of A.

Middle loop: Traverses columns of B.

Inner loop: Computes the dot product of the row from A and column from B.

**Code**:

public class MatrixMultiplication {

public static void main(String[] args) {

int[][] A = {

{1, 2},

{3, 4}

};

int[][] B = {

{5, 6},

{7, 8}

};

int rowsA = A.length, colsA = A[0].length;

int rowsB = B.length, colsB = B[0].length;

// Condition: Number of columns in A should match the number of rows in B

if (colsA != rowsB) {

System.out.println("Matrix multiplication not possible.");

return;

}

int[][] C = new int[rowsA][colsB];

for (int i = 0; i < rowsA; i++) {

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

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

C[i][j] += A[i][k] \* B[k][j]; // Multiply and accumulate

}

}

}

System.out.println("Matrix Multiplication Result:");

for (int[] row : C) {

for (int num : row) {

System.out.print(num + " ");

}

System.out.println();

}

}

}

**Output**:

Matrix Multiplication Result:

19 22

43 50

---

**8. Search an Element in a 2D Array**

**Explanation:**

Iterate through each row and column.

If the target element is found, print its position (row and column index).

**Code**:

public class SearchElement {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int target = 5; // Element to find

boolean found = false;

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

for (int j = 0; j < arr[i].length; j++) {

if (arr[i][j] == target) {

System.out.println("Element found at: (" + i + ", " + j + ")");

found = true;

break;

}

}

if (found) break; // Exit loop early if found

}

if (!found) System.out.println("Element not found.");

}

}

**Output**:

Element found at: (1, 1)

---

**9. Check Symmetric Matrix**

**Explanation:**

A symmetric matrix is a square matrix where:

A[i][j] = A[j][i] \quad \forall \quad i, j

We traverse only the upper half (i.e., elements above the diagonal) and check if they match their transposed positions.

**Code**:

public class SymmetricMatrix {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{2, 5, 6},

{3, 6, 9}

};

boolean symmetric = true;

// Ensure square matrix

if (arr.length != arr[0].length) {

System.out.println("Not a square matrix, so it cannot be symmetric.");

return;

}

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

for (int j = i + 1; j < arr.length; j++) { // Compare only upper half

if (arr[i][j] != arr[j][i]) {

symmetric = false;

break;

}

}

if (!symmetric) break;

}

System.out.println("Matrix is " + (symmetric ? "Symmetric" : "Not Symmetric"));

}

}

**Output** (for the given matrix):

Matrix is Symmetric

---

**10. Diagonal Sum of a Square Matrix**

**Explanation**:

Primary diagonal elements are where row index = column index (arr[i][i]).

Secondary diagonal elements are where row index + column index = size - 1 (arr[i][n - i - 1]).

We traverse the matrix once, adding both diagonals simultaneously.

If the matrix has an odd number of rows/columns, the center element is counted twice, so we subtract it once.

**Code:**

public class DiagonalSum {

public static void main(String[] args) {

int[][] arr = {

{1, 2, 3},

{4, 5, 6},

{7, 8, 9}

};

int n = arr.length;

int primarySum = 0, secondarySum = 0;

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

primarySum += arr[i][i]; // Main diagonal

secondarySum += arr[i][n - i - 1]; // Secondary diagonal

}

// If the matrix has an odd size, subtract the middle element counted twice

if (n % 2 != 0) {

secondarySum -= arr[n / 2][n / 2];

}

System.out.println("Primary Diagonal Sum: " + primarySum);

System.out.println("Secondary Diagonal Sum: " + secondarySum);

}

}

**Output**:

Primary Diagonal Sum: 15

Secondary Diagonal Sum: 15

---