**Q1. Spiral Matrix II Given a positive integer n, generate an n x n matrix filled with elements from 1 to n2 in spiral order.**

class Solution {

    public int[][] generateMatrix(int n) {

        int[][] matrix = new int[n][n];

        int num = 1;

        int top = 0, bottom = n - 1, left = 0, right = n - 1;

        while (top <= bottom && left <= right) {

            for (int i = left; i <= right; i++) {

                matrix[top][i] = num++;

            }

            top++;

            for (int i = top; i <= bottom; i++) {

                matrix[i][right] = num++;

            }

            right--;

            if (top <= bottom) {

                for (int i = right; i >= left; i--) {

                    matrix[bottom][i] = num++;

                }

                bottom--;

            }

            if (left <= right) {

                for (int i = bottom; i >= top; i--) {

                    matrix[i][left] = num++;

                }

                left++;

            }

        }

        return matrix;

    }

}

**Q2. Richest Customer Wealth You are given an m x n integer grid accounts where accounts[i][j] is the amount of money the ith customer has in the jth bank. Return the wealth that the richest customer has. A customer's wealth is the amount of money they have in all their bank accounts. The richest customer is the customer that has the maximum wealth**

class Solution {

    public int maximumWealth(int[][] accounts) {

        int largest = accounts[0][0];

        for(int customer[] : accounts){

            int currentSum = 0;

            for(int bank : customer) currentSum += bank;

            largest = Math.max(largest, currentSum);

        }

        return largest;

    }

}

**Q3. Toeplitz Matrix Given an m x n matrix, return true if the matrix is Toeplitz. Otherwise, return false. A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.**

class Solution {

    public boolean isToeplitzMatrix(int[][] matrix) {

        for(int i=1;i<matrix.length;i++){

            for(int j=1;j<matrix[i].length;j++){

                  if(matrix[i][j]!=matrix[i-1][j-1]){

                    return false;

                }

            }

          }

              return true;

        }

    }

**Q4. Matrix Diagonal Sum Given a square matrix mat, return the sum of the matrix diagonals. Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.**

class Solution {

    public int diagonalSum(int[][] mat) {

        int sum =0;

        int len = mat.length;

        int mid = mat[len/2][len/2];

        for(int i=0;i<len;i++){

            sum+=mat[i][i];

            if(i!=len-i-1) sum+=mat[i][len-i-1];

        }

        return sum;

    }

}

**Q5. Count Negative Numbers in a Sorted Matrix Given a m x n matrix grid which is sorted in non-increasing order both row-wise and column-wise, return the number of negative numbers in grid.**

class Solution

{

    public int countNegatives(int[][] grid)

    {

        int countOfNeg = 0;

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

        {

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

            {

                if(grid[i][j]<0)

                {

                    countOfNeg += (grid[i].length-j); break;

                }

            }

        }

        return countOfNeg;

    }

}

**Q6. Transpose Matrix Given a 2D integer array matrix, return the transpose of matrix. The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.**

import java.util.ArrayList;

import java.util.List;

class Solution {

    public int[][] transpose(int[][] matrix) {

        int row = matrix.length;

        int col = matrix[0].length;

        // Create a new matrix to store the transposed elements

        int[][] ans = new int[col][row]; // New matrix with dimensions col x row

        // Traverse through the original matrix

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

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

                // Assign transposed elements to the new matrix

                ans[j][i] = matrix[i][j]; // Transpose by swapping row and column indices

            }

        }

        return ans; // Return the transposed matrix

    }

}

**Q7. Set Matrix Zeroes Given an m x n integer matrix matrix, if an element is 0, set its entire row and column to 0's.**

class Solution {

    public void setZeroes(int[][] matrix) {

        List<Integer> list = new ArrayList<>();

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

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

                if (matrix[i][j] == 0) {

                    list.add(i \* matrix[0].length + j);

                }

            }

        }

        for (int i = 0; i < list.size(); i++) {

            int index = list.get(i);

            int row = index / matrix[0].length;

            int col = index % matrix[0].length;

            for (int k = 0; k < matrix[0].length; k++) {

                matrix[row][k] = 0;

            }

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

                matrix[k][col] = 0;

            }

        }

    }

}

**Q8. Kth Smallest Element in a Sorted Matrix Given an n x n matrix where each of the rows and columns is sorted in ascending order, return the kth smallest element in the matrix. Note that it is the kth smallest element in the sorted order, not the kth distinct element. You must find a solution with a memory complexity better than O(n2).**

class Solution {

    public int kthSmallest(int[][] matrix, int k) {

        int[] arr=new int[matrix.length\*matrix.length];

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

        {

            for(int j=0;j<matrix.length;j++)

            {

                arr[(i\*matrix.length)+j]=matrix[i][j];

            }

        }

        Arrays.sort(arr);

        return arr[k-1];

    }

}