1. Spiral matrix(LEETCODE)

```
import java.util.ArrayList;
import java.util.List;

public class SpiralMatrix {

    public static List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> ans = new ArrayList<Integer>();
        int row = matrix.length;
        int col = matrix[0].length;

        int total = row * col;
        int count = 0;

        // initialise the indexes;
        int startRow = 0;
        int endCol = col - 1;
        int endRow = row - 1;
        int startCol = 0;

        while(count < total){

            //print startRow
            for(int idx = startCol; count < total && idx <= endCol; idx++){
```

```
        ans.add(matrix[startRow][idx]);
        count++;
    }
    startRow++;

    // print lastCol
    for(int idx = startRow; count < total && idx <= endRow; idx++){
        ans.add(matrix[idx][endCol]);
        count++;
    }
    endCol--;

    // print lastRow

    for(int idx = endCol; count < total && idx >= startCol; idx--){
        ans.add(matrix[endRow][idx]);
        count++;
    }
    endRow--;

    // print firstCol

    for(int idx = endRow; count < total && idx >= startRow; idx--){
        ans.add(matrix[idx][startCol]);
```

```
            count++;
          }
          startCol++;
        }

        return ans;
    }

        public static void main(String[] args) {
                // TODO Auto-generated method stub
                int matrix[][] = {{1,2,3},{4,5,6},{7,8,9}};
                System.out.println(spiralOrder(matrix));
        }

}
```

2. K-weakest-rows(LEETCODE)

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class KWeakestRow {
    public static int[] kWeakestRows(int[][] mat, int k) {
        int row = mat.length;
        int col = mat[0].length;
        int[] soldiersCount = new int[row];
        int[] result = new int[k];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                soldiersCount[i] += mat[i][j];
            }
        }

        Map<Integer, Integer> map = new HashMap<>();
        for (int i = 0; i < row; i++) {
            map.put(i, soldiersCount[i]);
        }
```

```
        List<Map.Entry<Integer, Integer>> list = new
ArrayList<>(map.entrySet());
        Collections.sort(list, Map.Entry.comparingByValue());

        for (int i = 0; i < k; i++) {
            result[i] = list.get(i).getKey();
        }

        return result;
    }
        public static void main(String[] args) {
                // TODO Auto-generated method stub
                int mat[][] =
                                {{1,1,0,0,0},
                                 {1,1,1,1,0},
                                 {1,0,0,0,0},
                                 {1,1,0,0,0},
                                 {1,1,1,1,1}};

                                int k = 3;
                                int [] arr = kWeakestRows(mat,k);

                                for(int i : arr) {
```

```
                    System.out.print(i + " ");
                        }
        }

}
```

3. Set-Matrix-zeroes(LEETCODE)

```
public class SetZeros {


    public static void setZeroes(int[][] matrix) {
    boolean fr = false,fc = false;
    for(int i = 0; i < matrix.length; i++) {
       for(int j = 0; j < matrix[0].length; j++) {
          if(matrix[i][j] == 0) {
             if(i == 0)
                fr = true;
             if(j == 0)
                fc = true;
             matrix[0][j] = 0;
             matrix[i][0] = 0;
          }
       }
    }
    for(int i = 1; i < matrix.length; i++) {
       for(int j = 1; j < matrix[0].length; j++) {
          if(matrix[i][0] == 0 || matrix[0][j] == 0) {
             matrix[i][j] = 0;
       }  }
    }
    if(fr) {
```

```
        for(int j = 0; j < matrix[0].length; j++) {
            matrix[0][j] = 0;
        }
    }
    if(fc) {
        for(int i = 0; i < matrix.length; i++) {
            matrix[i][0] = 0;
        }
    }

    for(int i[] : matrix) {
     for(int e : i) {
            System.out.print(e + " ");
     }
     System.out.println();
    }
  }

     public static void main(String[] args) {
            // TODO Auto-generated method stub
            int matrix[][] = {{1,1,1},{1,0,1},{1,1,1}};
            setZeroes(matrix);

     }}
```

4. Matrix diagonal sum(LEETCODE)

```
public class MatrixDiagonalSum {
        public static int diagonalSum(int[][] mat) {
      int sum = 0;
      for(int i =0 ;i < mat.length; i ++){
         for(int j = 0; j < mat[i].length; j++){
            if(i==j){
               sum += mat[i][j];
            }
            else if(i+j == mat.length-1){
               sum+= mat[i][j];
            }
         }
      }
      return sum;
   }
        public static void main(String[] args) {
               // TODO Auto-generated method stub
               int mat[][] = {{1,2,3},
                  {4,5,6},
                  {7,8,9}};

               System.out.println(diagonalSum(mat));
        }
```

```
}
```

5. Addition-of-two-square-matrix(geekforgeeks)

```
public class AdditionSquareMatrix {

    public static void Addition(int[][] matrixA, int[][] matrixB)
  {
    // code here
    for(int i=0; i<matrixA.length; i++){
      for(int j=0; j<matrixA[i].length;j++){
        matrixA[i][j]= matrixA[i][j] + matrixB[i][j];
      }
    }

    for(int i[] : matrixA) {
      for(int e : i) {
            System.out.print(e+" ");
      }
      System.out.println();
    }
  }

    public static void main(String[] args) {
            // TODO Auto-generated method stub
            int matrixA[][] = {{1, 2}, {3, 4}};
                    int matrixB[][] = {{4, 3}, {2, 1}};
```

```
Addition(matrixA, matrixB);
        }

}
```

6. Multiply-two-matrices(geekforgeeks)

```
public class MultiplyMatrix {
        public static void  Mutliply(int[][] matrixA, int[][] matrixB)
   {
      // code here
      int rowsA = matrixA.length;
      int colsA = matrixA[0].length;
      int colsB = matrixB[0].length;

      int[][] result = 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++) {
               result[i][j] += matrixA[i][k] * matrixB[k][j];
            }
         }
      }

      for (int i = 0; i < rowsA; i++) {
         for (int j = 0; j < colsB; j++) {
            matrixA[i][j] = result[i][j];
         }
      }
```

```
      for(int i[] : matrixA) {
       for(int e : i) {
              System.out.print(e+" ");
       }
       System.out.println();
      }
   }
       public static void main(String[] args) {
              // TODO Auto-generated method stub
              int matrixA [][] = {{1, 1, 1}, {1, 1, 1},
                     {1, 1, 1}};
              int matrixB [][] = {{1, 1, 1}, {1, 1, 1},
                     {1, 1, 1}};
              Mutliply(matrixA, matrixB);

       }

}
```