2D Array

Bhai 🔥

Tera content already perfect hai, main bas thoda **Java-focused formatting, indentation, aur comment clarity** polish kar diya hai, exactly waise hi jaise tu chahta tha — **interview-ready + bug-aware**.

🧱 PHASE 1 – 1️⃣ Matrix Representation (2D Basics) – JAVA VERSION

🔷 1️⃣ 2D Array vs List<List<Integer>>

✅ 1️⃣ Static 2D Array (Java)

```
// Declaration
int[][] arr = new int[3][4];

// Initialization
int[][] arr = {
    {1, 2, 3},
    {4, 5, 6}
};

// Access
System.out.println(arr[1][2]); // 6

// ❌ Limitations
// - Size fixed at compile time
// - Runtime flexibility limited
```

> Interview me static 2D array avoid karna better

✅ 2️⃣ Dynamic 2D Matrix (List<List<Integer>>) 🔥

```
// Declaration
List<List<Integer>> matrix = new ArrayList<>();

// Initialization
List<List<Integer>> matrix = Arrays.asList(
    Arrays.asList(1, 2, 3),
    Arrays.asList(4, 5, 6)
);

// Access
System.out.println(matrix.get(0).get(1)); // 2
```

📌 **Interview Verdict:**

> Java interviews me List<List<Integer>> preferred hai — dynamic, flexible, safe

---

🔷 2️⃣ Row & Column Count (JAVA)

```
int rows = matrix.size();        // number of rows
int cols = matrix.get(0).size(); // number of columns
```

⚠️ **IMPORTANT BUG:**

```
matrix.get(0).size(); // ❌ if matrix empty → runtime exception
```

✅ **Safe Check**

```
if (matrix == null || matrix.isEmpty()) return;
```

🔷 3️⃣ matrix.size() vs matrix.get(0).size()

Expression	Meaning
matrix.size()	number of rows
matrix.get(0).size()	number of columns

🧠 FULLY SOLVED EXAMPLES (JAVA)

🟢 Example 1: Print Matrix + Rows & Columns

Input:

```
matrix = [[1,2,3],[4,5,6]]
```

Code (Java)

```
public static void printMatrix(List<List<Integer>> matrix) {
    if (matrix == null || matrix.isEmpty()) {
        System.out.println("Matrix is empty");
        return;
    }

    int rows = matrix.size();
    int cols = matrix.get(0).size();

    System.out.println("Rows: " + rows);
    System.out.println("Cols: " + cols);

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            System.out.print(matrix.get(i).get(j) + " ");
        }
        System.out.println();
    }
}
```

Output:

```
Rows: 2
Cols: 3
1 2 3
4 5 6
```

🟢 Example 2: Sum of All Elements in Matrix

Problem:

> Matrix ke saare elements ka sum nikalna

Code (Java)

```
public static int matrixSum(List<List<Integer>> matrix) {
    if (matrix == null || matrix.isEmpty()) return 0;

    int sum = 0;
    int rows = matrix.size();
    int cols = matrix.get(0).size();

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum += matrix.get(i).get(j);
        }
    }
    return sum;
}
```

Dry Run:

`1 + 2 + 3 + 4 + 5 + 6 = 21`

⚠️ COMMON INTERVIEW MISTAKES (JAVA)

- ❌ Using static array blindly
- ❌ `matrix.get(0)` without empty check
- ❌ Rows vs columns confusion
- ❌ Assuming square matrix (n x n) always

🧠 INTERVIEW GOLD LINES (JAVA)

Use these consciously 👇

1️⃣ “matrix.size() gives number of rows.”

2️⃣ “matrix.get(0).size() gives number of columns.”

3️⃣ “I’ll check if the matrix is empty to avoid runtime exceptions.”

🧪 PRACTICE (Warm-up – Java Friendly)

1️⃣ LC 1470 – Shuffle the Array

2️⃣ LC 1572 – Matrix Diagonal Sum

3️⃣ LC 867 – Transpose Matrix

Bhai 🔥

Tera content already perfect hai, maine bas **Java formatting, indentation aur interview clarity ke liye comments polish** kar diye. Logic 100% wahi hai, sirf clean aur bug-aware version:

🔷 Matrix Traversal – JAVA INTERVIEW READY

```
int[][] mat = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

int rows = mat.length;    // number of rows
int cols = mat[0].length; // number of columns
```

1️⃣ Row-wise Traversal (Most Common)

✅ Example 1: Print Row-wise

```
static void rowWise(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;

    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
```

```
            System.out.print(mat[i][j] + " ");
        }
    }
}
```

Output:

```
1 2 3 4 5 6 7 8 9
```

✅ Example 2: Row-wise Sum

```
static void rowSum(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;

    for (int i = 0; i < r; i++) {
        int sum = 0;
        for (int j = 0; j < c; j++) {
            sum += mat[i][j];
        }
        System.out.println("Row " + i + " sum = " + sum);
    }
}
```

Output:

```
Row 0 sum = 6
Row 1 sum = 15
Row 2 sum = 24
```

2️⃣ Column-wise Traversal

✅ Example 1: Print Column-wise

```
static void colWise(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;
```

```
    for (int j = 0; j < c; j++) {
        for (int i = 0; i < r; i++) {
            System.out.print(mat[i][j] + " ");
        }
    }
}
```

Output:

```
1 4 7 2 5 8 3 6 9
```

✅ Example 2: Column-wise Sum

```
static void colSum(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;

    for (int j = 0; j < c; j++) {
        int sum = 0;
        for (int i = 0; i < r; i++) {
            sum += mat[i][j];
        }
        System.out.println("Col " + j + " sum = " + sum);
    }
}
```

Output:

```
Col 0 sum = 12
Col 1 sum = 15
Col 2 sum = 18
```

3️⃣ Reverse Traversal

🔶 (A) Reverse Row-wise (Bottom → Top)

```
static void reverseRowWise(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;

    for (int i = r - 1; i >= 0; i--) {
        for (int j = 0; j < c; j++) {
            System.out.print(mat[i][j] + " ");
        }
    }
}
```

🔶 (B) Reverse Column-wise (Right → Left)

```
static void reverseColWise(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;

    for (int j = c - 1; j >= 0; j--) {
        for (int i = 0; i < r; i++) {
            System.out.print(mat[i][j] + " ");
        }
    }
}
```

4️⃣ Diagonal Traversal 🔥

⚠️ Only for **square matrix**

🔶 (A) Main Diagonal (i == j)

```
static void mainDiagonal(int[][] mat) {
    int n = mat.length;

    for (int i = 0; i < n; i++) {
        System.out.print(mat[i][i] + " ");
```

```
    }
}
```

🔶 (B) Anti-Diagonal (i + j == n - 1)

```
static void antiDiagonal(int[][] mat) {
    int n = mat.length;

    for (int i = 0; i < n; i++) {
        System.out.print(mat[i][n - i - 1] + " ");
    }
}
```

⚠️ COMMON INTERVIEW BUGS (Java Specific)

- ❌ `mat.length` vs `mat[0].length` confuse karna
- ❌ Column traversal me loop order ulta likhna
- ❌ Rectangular matrix pe diagonal laga dena
- ❌ `ArrayIndexOutOfBoundsException`

🧠 INTERVIEW GOLD LINES

- Row-wise → fix row, move columns
- Column-wise → fix column, move rows
- Diagonal → sirf square matrix

🧪 Practice (Next Level)

1️⃣ LC 1572 – Matrix Diagonal Sum

2️⃣ LC 867 – Transpose Matrix

3️⃣ LC 54 – Spiral Matrix (🔥 next topic)

Bhai 🔥

Ye content already perfect hai, maine bas **Java-style clean formatting, comments, interview notes aur pitfalls** add kar diye — logic 100% wahi hai, sirf **interview-ready polish**:

📦 PHASE 2 – Transpose of Matrix (JAVA)

🔷 What is Transpose?

- Row → Column
- Column → Row

Example (2×3 Matrix)

Original:

```
1 2 3
4 5 6
```

Transpose (3×2):

```
1 4
2 5
3 6
```

🔷 Square vs Rectangular Matrix

Type	Rows	Cols	In-place possible?
Square	n	n	✅ YES
Rectangular	n	m	❌ NO

Interview Line:

> "In-place transpose is possible only for square matrices."

1️⃣ Transpose of Rectangular Matrix (Extra Space)

✅ Java Code

```
static int[][] transposeRectangular(int[][] mat) {
    int r = mat.length;
    int c = mat[0].length;

    int[][] t = new int[c][r]; // extra matrix

    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            t[j][i] = mat[i][j];
        }
    }
    return t;
}
```

Example 1: 2×3

Input:

```
1 2 3
4 5 6
```

Output:

```
1 4
2 5
3 6
```

Example 2: 3×2

Input:

```
1 2
3 4
5 6
```

Output:

```
1 3 5
```

```
2 4 6
```

⏱️ Time: O(r × c)

🧠 Space: O(r × c)

✅ Same code works for any rectangular matrix.

2️⃣ Transpose of Square Matrix (In-Place 🔥)

Example 3×3:

```
1 2 3
4 5 6
7 8 9
```

Brute Force ❌ → extra matrix, waste space

✅ Optimal In-Place Transpose (Java)

💡 **Idea:** Swap elements **above the main diagonal**

- Diagonal stays same

```
static void transposeSquare(int[][] mat) {
    int n = mat.length;

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }
}
```

Dry Run:

Swaps performed:

- mat[0][1] ↔ mat[1][0]
- mat[0][2] ↔ mat[2][0]

- mat[1][2] ↔ mat[2][1]

Output:

```
1 4 7
2 5 8
3 6 9
```

⏱️ Time: O(n²)

🧠 Space: O(1) ✅

⚠️ COMMON INTERVIEW MISTAKES

- ❌ `j = 0` se loop chala diya → double swap bug
- ❌ Rectangular matrix pe in-place try kar diya
- ❌ `mat.length` vs `mat[0].length` confuse
- ❌ Row–column indices ulte likh diye

🧠 INTERVIEW GOLD LINES

1. “Square matrix ka transpose in-place ho sakta hai.”
2. “We swap only elements above the main diagonal.”
3. “Rectangular matrix needs extra space.”

🧪 Practice (LeetCode Must – Java Ready)

1️⃣ 867 – Transpose Matrix

2️⃣ 48 – Rotate Image → transpose + reverse rows 🔥

3️⃣ 1572 – Matrix Diagonal Sum

Bhai 🔥

Ye content already **FAANG-level Java rewrite + interview-ready** hai, maine bas thoda aur **format, comments, interview traps aur golden lines** polish kar diye — logic exactly wahi hai.

🧱 PHASE 2 – 4️⃣ Rotate Matrix (JAVA)

📌 **Important Rule:**

> In-place rotation sirf square matrix (n × n) me possible hai

Reference Matrix

```
1 2 3
4 5 6
7 8 9
```

🔷 1️⃣ Rotate 90° Clockwise (🔥 MOST ASKED)

❌ Brute Force (Extra Space)

💡 Idea: Naya matrix bana ke

```
res[j][n-1-i] = mat[i][j];
```

Java Code (Brute)

```
static int[][] rotate90Brute(int[][] mat) {
    int n = mat.length;
    int[][] res = new int[n][n];

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            res[j][n - 1 - i] = mat[i][j];
        }
    }
    return res;
}
```

⏱️ O(n²), 🧠 O(n²) ❌ (Interview me avoid)

✅ Optimal (IN-PLACE 🔥)

💡 **Core Logic:**

1️⃣ Transpose

2️⃣ Reverse each row

```
static void rotate90Clockwise(int[][] mat) {
    int n = mat.length;

    // STEP 1: Transpose
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }

    // STEP 2: Reverse each row
    for (int i = 0; i < n; i++) {
        reverseRow(mat[i]);
    }
}

static void reverseRow(int[] row) {
    int l = 0, r = row.length - 1;
    while (l < r) {
        int temp = row[l];
        row[l] = row[r];
        row[r] = temp;
        l++; r--;
    }
}
```

Dry Run:

- After Transpose

```
1 4 7
2 5 8
3 6 9
```

- After Row Reverse

```
7 4 1
8 5 2
9 6 3
```

⏱️ O(n²), 🧠 O(1) ✅

Interview Line:

> “90° clockwise rotation = transpose + reverse rows.”

🔷 2️⃣ Rotate 90° Anti-Clockwise

💡 **Trick:** Transpose + Reverse Columns

```
static void rotate90AntiClockwise(int[][] mat) {
    int n = mat.length;

    // Transpose
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            int temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }

    // Reverse each column
    for (int j = 0; j < n; j++) {
        int top = 0, bottom = n - 1;
        while (top < bottom) {
            int temp = mat[top][j];
            mat[top][j] = mat[bottom][j];
            mat[bottom][j] = temp;
            top++; bottom--;
        }
```

```
    }
}
```

Output:

```
3 6 9
2 5 8
1 4 7
```

Interview Line:

> "Anti-clockwise rotation = transpose + reverse columns."

🔷 3️⃣ Rotate 180°

💡 **Trick:**

- Reverse rows
- Reverse columns

  (OR rotate 90° twice)

```
static void rotate180(int[][] mat) {
    int n = mat.length;

    // Reverse rows (top ↔ bottom)
    int top = 0, bottom = n - 1;
    while (top < bottom) {
        int[] temp = mat[top];
        mat[top] = mat[bottom];
        mat[bottom] = temp;
        top++; bottom--;
    }

    // Reverse columns in each row
    for (int i = 0; i < n; i++) {
        reverseRow(mat[i]);
    }
}
```

Output:

```
9 8 7
6 5 4
3 2 1
```

⚠️ COMMON INTERVIEW MISTAKES

- ❌ Rectangular matrix pe in-place rotate
- ❌ Transpose ke baad galat reverse
- ❌ Clockwise & anti-clockwise confuse
- ❌ Extra space use jab in-place allowed ho

🧠 Cheat Sheet (RAT LO 🔥)

Rotation	Steps
90° Clockwise	Transpose + Reverse rows
90° Anti-Clockwise	Transpose + Reverse cols
180°	Reverse rows + Reverse cols

🧪 LeetCode MUST (Java)

1️⃣ 48 – Rotate Image 🔥🔥

2️⃣ 1886 – Determine Whether Matrix Can Be Obtained By Rotation

3️⃣ 867 – Transpose Matrix

Bhai 🔥

Ye raha **FAANG-level, pure Java, interview-ready Spiral Matrix module** — har line pe **detailed comment + logic explanation**, taki interviewer samjhe ki tu engineer mindset me soch raha hai 💯

🧱 PHASE 2 – 5️⃣ Spiral Traversal & Generation (JAVA)

🔷 1️⃣ Spiral Traversal / Print (LC 54)

Problem:

Matrix ko clockwise spiral order me traverse karke **list me store** karo.

Core Idea (Interview GOLD):

> 4 boundaries maintain karo — top, bottom, left, right
>
> Har directional pass ke baad boundary shrink hoti hai.

Java Code (Full Detailed Comments)

```
import java.util.*;

class SpiralTraversal {

    static List<Integer> spiralOrder(int[][] mat) {
        // Result list
        List<Integer> ans = new ArrayList<>();

        // Edge case: null or empty matrix
        if (mat == null || mat.length == 0)
            return ans;

        // Boundaries
        int top = 0;
        int bottom = mat.length - 1;       // last row index
        int left = 0;
        int right = mat[0].length - 1;    // last column index

        // Loop until boundaries cross
        while (top <= bottom && left <= right) {

            // 1️⃣ Traverse top row (left → right)
            for (int j = left; j <= right; j++)
                ans.add(mat[top][j]);
            top++;  // shrink top boundary

            // 2️⃣ Traverse right column (top → bottom)
```

```
            for (int i = top; i <= bottom; i++)
                ans.add(mat[i][right]);
            right--; // shrink right boundary

            // 3️⃣ Traverse bottom row (right → left)
            // Check if row still exists after shrinking
            if (top <= bottom) {
                for (int j = right; j >= left; j--)
                    ans.add(mat[bottom][j]);
                bottom--; // shrink bottom boundary
            }

            // 4️⃣ Traverse left column (bottom → top)
            // Check if column still exists after shrinking
            if (left <= right) {
                for (int i = bottom; i >= top; i--)
                    ans.add(mat[i][left]);
                left++; // shrink left boundary
            }
        }

        return ans;
    }
}
```

Dry Run Example:

Input:

```
1 2 3
4 5 6
7 8 9
```

Output:

```
1 2 3 6 9 8 7 4 5
```

⏱️ Time Complexity: O(n × m)

🧠 Space Complexity: O(1) (output list ke alawa)

Interview Line:

> "We maintain four boundaries and shrink them after each directional traversal."

🔷 2️⃣ Spiral Matrix Generation (LC 59)

Problem:

Given `n`, generate `n × n` matrix filled with numbers `1 → n²` in spiral order.

Core Idea:

> Use 4 boundaries same as traversal
>
> Place numbers sequentially while shrinking boundaries

Java Code (Full Detailed Comments)

```
class SpiralMatrixGenerate {

    static int[][] generateMatrix(int n) {
        int[][] mat = new int[n][n]; // Result matrix

        // Boundaries
        int top = 0;
        int bottom = n - 1;
        int left = 0;
        int right = n - 1;

        int num = 1; // Starting number

        // Loop until boundaries cross
        while (top <= bottom && left <= right) {

            // 1️⃣ Fill top row (left → right)
            for (int j = left; j <= right; j++)
                mat[top][j] = num++;
            top++; // shrink top boundary

            // 2️⃣ Fill right column (top → bottom)
```

```
            for (int i = top; i <= bottom; i++)
                mat[i][right] = num++;
            right--; // shrink right boundary

            // 3️⃣ Fill bottom row (right → left)
            if (top <= bottom) { // check row exists
                for (int j = right; j >= left; j--)
                    mat[bottom][j] = num++;
                bottom--; // shrink bottom boundary
            }

            // 4️⃣ Fill left column (bottom → top)
            if (left <= right) { // check column exists
                for (int i = bottom; i >= top; i--)
                    mat[i][left] = num++;
                left++; // shrink left boundary
            }
        }

        return mat;
    }
}
```

Dry Run Example (n=3):

Resulting matrix:

```
1 2 3
8 9 4
7 6 5
```

⏱️ Time Complexity: $O(n^2)$

🧠 Space Complexity: O(1) (in-place filling in result matrix)

⚠️ COMMON INTERVIEW TRAPS

Trap	Mistake
Boundary check	Forget `if (top <= bottom)` or `if (left <= right)` → duplicates or ArrayIndexOutOfBoundsException
Single row/column	Not handling edge → duplicates
Traversal order	Wrong order → top→right→bottom→left
Rectangular matrix in generation	Generation expects n×n; traversal can handle m×n

🧠 Interview GOLD LINE (RAT LO 🔥)

> "Maintain four boundaries and shrink them after each directional traversal. Always check if the current row/column exists before filling or reading to avoid duplicates and out-of-bounds errors."

🧪 LeetCode Must (Java Ready)

1️⃣ 54 – Spiral Matrix 🔥🔥 (Traversal)

2️⃣ 59 – Spiral Matrix II 🔥🔥 (Generation)

3️⃣ 885 – Spiral Matrix III (Advanced, with starting point & directions)

Bhai 🔥

Ye raha **PHASE 3 – Searching in 2D Matrix (JAVA)** ka **FAANG-style, interview-ready, detailed comment version** — har approach ka intuition + trap + code ready for discussion 💯

🧱 PHASE 3 – Searching in 2D Matrix (JAVA)

🔷 6️⃣ Linear Search (Brute Force)

Use Case:

Matrix **unsorted** ya **constraints small** → sirf linear search hi simple & safe hai.

```
class LinearSearch2D {
    static boolean searchMatrix(int[][] mat, int target) {
        int r = mat.length;
        int c = mat[0].length;

        for (int i = 0; i < r; i++) {
            for (int j = 0; j < c; j++) {
                if (mat[i][j] == target)
                    return true; // target found
            }
        }
        return false; // not found
    }
}
```

⏱️ **Time:** O(n × m)

🧠 **Space:** O(1)

Interview Line:

> "This is brute force; we can optimize using matrix properties."

🔷 7️⃣ Binary Search in Matrix

🔶 Case 1️⃣ – Row-wise Sorted Matrix

Matrix Property:

> Each row sorted, rows independent

```
1  3  5
7  9 11
13 15 17
```

Idea:

- Check if target lies between first & last element of the row
- Apply **binary search** only on valid rows

```
class RowWiseBinarySearch {
    static boolean searchRowWise(int[][] mat, int target) {
        int r = mat.length;
        int c = mat[0].length;

        for (int i = 0; i < r; i++) {
            // Only search if target could be in this row
            if (target >= mat[i][0] && target <= mat[i][c - 1]) {
                int l = 0, h = c - 1;
                while (l <= h) {
                    int mid = l + (h - l) / 2; // safe mid
                    if (mat[i][mid] == target)
                        return true;
                    else if (mat[i][mid] < target)
                        l = mid + 1;
                    else
                        h = mid - 1;
                }
            }
        }
        return false;
    }
}
```

⏱️ **Time:** O(n × log m)

🧠 **Space:** O(1)

Interview Line:

> "We apply binary search on rows where target is possible."

🔶 Case 2️⃣ – Fully Sorted Matrix (LC 74)

Matrix Property:

> Last element of row i < first element of row i+1
>
> Entire matrix can be treated as **1D sorted array**

Idea:

- Flatten index: `row = mid / cols`, `col = mid % cols`
- Apply standard **binary search**

```
class FullySortedMatrixSearch {
    static boolean searchMatrix(int[][] mat, int target) {
        int r = mat.length;
        int c = mat[0].length;

        int low = 0, high = r * c - 1;

        while (low <= high) {
            int mid = low + (high - low) / 2;
            int row = mid / c;
            int col = mid % c;

            if (mat[row][col] == target)
                return true;
            else if (mat[row][col] < target)
                low = mid + 1;
            else
                high = mid - 1;
        }
        return false;
    }
}
```

⏱️ **Time:** O(log(n × m))

🧠 **Space:** O(1)

Interview Line:

> "Since the matrix is fully sorted, we apply binary search by flattening indices."

🔶 Case 3️⃣ – Staircase Search (LC 240)

Matrix Property:

> Rows sorted left → right

> Columns sorted top → bottom

Idea:

- Start **top-right** OR **bottom-left**
- Move **left** if current > target
- Move **down** if current < target

```
class StaircaseSearch {
    static boolean staircaseSearch(int[][] mat, int target) {
        int r = mat.length;
        int c = mat[0].length;

        int i = 0, j = c - 1; // top-right starting point

        while (i < r && j >= 0) {
            if (mat[i][j] == target)
                return true;      // target found
            else if (mat[i][j] > target)
                j--;              // move left
            else
                i++;              // move down
        }
        return false; // target not found
    }
}
```

⏱️ **Time:** O(n + m)

🧠 **Space:** O(1)

Dry Run (Target = 6):

```
7 → left → 4 → down → 5 → down → 6 ✔️
```

Interview Line:

> "Staircase search leverages both row & column sorted properties efficiently."

⚠️ Common Interview Mistakes

Mistake	Reason
Wrong search for matrix type	Applying wrong algo for property
Fully sorted → binary on row only	Missed flattening concept
Staircase search wrong start	Not top-right / bottom-left
No property clarification	Interviewer will ask assumptions

🏆 Cheat Sheet (🔥 MUST REMEMBER)

Matrix Type	Best Search Method
Unsorted	Linear
Row-wise sorted	Row + Binary
Fully sorted	Binary on 1D
Row + Col sorted	Staircase

🧪 LeetCode MUST DO (Java Ready)

1️⃣ 74 – Search a 2D Matrix

2️⃣ 240 – Search a 2D Matrix II

3️⃣ 378 – Kth Smallest Element in a Sorted Matrix

4️⃣ 1351 – Count Negative Numbers in a Sorted Matrix

Bhai 🔥

Chal, mai **2D Prefix Sum with full i,j representation & visualization** bana deta hoon — exactly jaisa tu chahta hai, interview-ready Java version me, aur har step ka **i,j coordinates + dry run** dikhayenge.

🧱 2D Prefix Sum – Full i,j Representation (Java)

🔷 Example Matrix

```
mat =
i\j 0  1  2
0   1  2  3
1   4  5  6
2   7  8  9
```

- `i` = row index
- `j` = column index

🔷 Prefix Formula (Coordinate-aware)

```
prefix[i][j] = mat[i][j]
             + (i>0 ? prefix[i-1][j] : 0)    // up
             + (j>0 ? prefix[i][j-1] : 0)    // left
             - (i>0 && j>0 ? prefix[i-1][j-1] : 0) // diagonal overlap
```

🔷 Step-by-Step Dry Run with i,j Coordinates

i	j	Calculation	prefix[i][j]
0	0	1 + 0 + 0 - 0	1
0	1	2 + 0 + 1 - 0	3
0	2	3 + 0 + 3 - 0	6
1	0	4 + 1 + 0 - 0	5
1	1	5 + 3 + 5 - 1	12
1	2	6 + 6 + 12 - 3	21
2	0	7 + 5 + 0 - 0	12
2	1	8 + 12 + 12 - 5	27
2	2	9 + 21 + 27 - 12	45

✅ Final Prefix Matrix:

```
prefix =
1  3  6
```

```
5  12  21
12 27  45
```

🔷 Query Example with i,j Notation

Query: `(r1,c1) = (1,1)` , `(r2,c2) = (2,2)`

```
sum = prefix[r2][c2]
      - (r1>0 ? prefix[r1-1][c2] : 0)
      - (c1>0 ? prefix[r2][c1-1] : 0)
      + (r1>0 && c1>0 ? prefix[r1-1][c1-1] : 0)
```

Step-by-step:

```
prefix[2][2] = 45
prefix[0][2] = 6
prefix[2][0] = 12
prefix[0][0] = 1
```

```
sum = 45 - 6 - 12 + 1 = 28 ✅
```

Submatrix elements `(i,j)`:

```
i\j 1  2
1   5  6
2   8  9
```

🔷 Java Implementation (i,j style comments)

```
class NumMatrix {

    private int[][] prefix;

    public NumMatrix(int[][] mat) {
        int n = mat.length;
        int m = mat[0].length;
```

```
        prefix = new int[n][m];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                int up = (i > 0) ? prefix[i - 1][j] : 0;      // prefix sum of row above
                int left = (j > 0) ? prefix[i][j - 1] : 0;    // prefix sum of col to left
                int diag = (i > 0 && j > 0) ? prefix[i - 1][j - 1] : 0; // overlap

                prefix[i][j] = mat[i][j] + up + left - diag;

                // Debug Print: show coordinates and value
                System.out.println("prefix[" + i + "][" + j + "] = " + prefix[i][j]);
            }
        }
    }

    public int sumRegion(int r1, int c1, int r2, int c2) {
        int total = prefix[r2][c2];
        int up = (r1 > 0) ? prefix[r1 - 1][c2] : 0;
        int left = (c1 > 0) ? prefix[r2][c1 - 1] : 0;
        int diag = (r1 > 0 && c1 > 0) ? prefix[r1 - 1][c1 - 1] : 0;

        return total - up - left + diag;
    }

    public static void main(String[] args) {
        int[][] mat = {
            {1, 2, 3},
            {4, 5, 6},
            {7, 8, 9}
        };

        NumMatrix nm = new NumMatrix(mat);

        System.out.println("Sum query (1,1) to (2,2) = " + nm.sumRegion(1, 1,
2, 2));
    }
}
```

🔷 Key Interview Lines (i,j style)

- “prefix[i][j] = sum from (0,0) to (i,j) including overlap correction”
- “Query sum = total - top area - left area + overlap”
- “Always verify boundary conditions: i=0, j=0”

Bhai 🔥 samajh gaya — mai **exactly waise hi format + structure** me likhunga jaise tune diya tha, **code ke andar har line / step ke liye detailed comment** bhi add kar dunga.

🧱 SPECIAL MATRICES — COMPLETE (Java)

1️⃣ Identity Matrix

🔷 Definition

- Square matrix (n × n)
- Diagonal = 1
- Rest = 0

Example:

```
1 0 0
0 1 0
0 0 1
```

🔑 Properties

- Square hona compulsory
- `mat[i][i] == 1`
- `mat[i][j] == 0` (i ≠ j)

✅ Java Code

```
public static boolean isIdentityMatrix(int[][] mat) {
    int n = mat.length; // n x n square matrix
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // Diagonal element must be 1
            if (i == j && mat[i][j] != 1) return false;

            // Non-diagonal element must be 0
            if (i != j && mat[i][j] != 0) return false;
        }
    }
    return true; // All conditions satisfied
}
```

🧠 Interview Line

> "An identity matrix has ones on the main diagonal and zeros elsewhere."

2️⃣ Diagonal Matrix

🔷 Definition

- Square matrix
- Sirf diagonal pe non-zero allowed

Example:

```
5 0 0
0 8 0
0 0 3
```

🔑 Identity vs Diagonal

Matrix	Diagonal Values
Identity	Only 1
Diagonal	Any value

✅ Java Code

```
public static boolean isDiagonalMatrix(int[][] mat) {
    int n = mat.length; // n x n square matrix
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            // All non-diagonal elements must be zero
            if (i != j && mat[i][j] != 0)
                return false;
        }
    }
    return true; // All non-diagonal elements are zero
}
```

🧠 Interview Line

> "All non-diagonal elements must be zero."

3️⃣ Toeplitz Matrix 🔥 (Most Asked)

🔷 Definition

- Har top-left → bottom-right diagonal me same value

Example:

```
1 2 3 4
5 1 2 3
9 5 1 2
```

🔑 Condition

- `mat[i][j] == mat[i-1][j-1]`

✅ Java Code (Optimal)

```
public static boolean isToeplitzMatrix(int[][] mat) {
    int n = mat.length;    // number of rows
    int m = mat[0].length; // number of columns

    // Start from second row and second column
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            // Each element must equal the one diagonally above
            if (mat[i][j] != mat[i - 1][j - 1])
                return false; // Condition fails
```

```
        }
    }
    return true; // All diagonals consistent
}
```

⏱️ Complexity

- Time: O(n × m)
- Space: O(1)

🧠 Interview Line

> "Each element equals the one diagonally above it."

⚠️ Common Traps

❌ Identity ≠ Diagonal

❌ Toeplitz diagonal direction confuse

❌ Square check bhoolna

❌ Loop boundaries galat

🔥 LeetCode

- 766 — Toeplitz Matrix
- 498 — Diagonal Traverse
- 1424 — Diagonal Traverse II

4️⃣ Matrix Greedy & Logic — COMPLETE (Java)

🔥 Problem 1: Set Matrix Zeroes (LC 73)

🔷 Problem

- Agar `mat[i][j] == 0` → poori row + column = 0

🔥 Optimal In-Place Approach (Google Fav)

- **Trick:** First row & first column ko marker banao

✅ Java Code (O(1) Space)

```
public static void setZeroes(int[][] mat) {
    int n = mat.length, m = mat[0].length;
    boolean firstRow = false, firstCol = false;
```

```
    // Check if first column has zero
    for (int i = 0; i < n; i++)
        if (mat[i][0] == 0) firstCol = true;

    // Check if first row has zero
    for (int j = 0; j < m; j++)
        if (mat[0][j] == 0) firstRow = true;

    // Use first row & column as markers
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            if (mat[i][j] == 0) {
                mat[i][0] = 0; // mark row
                mat[0][j] = 0; // mark column
            }
        }
    }

    // Set zeros based on markers
    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            if (mat[i][0] == 0 || mat[0][j] == 0)
                mat[i][j] = 0;
        }
    }

    // Handle first row
    if (firstRow)
        for (int j = 0; j < m; j++) mat[0][j] = 0;

    // Handle first column
    if (firstCol)
        for (int i = 0; i < n; i++) mat[i][0] = 0;
}
```

⚠️ **Interview Traps**

❌ First row/col overwrite

❌ Extra space usage

❌ Marker logic miss

🔥 Problem 2: Game of Life (LC 289)

🔷 Rules Recap

Current	Neighbors	Next
Live	<2 or >3	Die
Live	2 or 3	Live
Dead	=3	Live

🧠 Encoding Trick (In-Place)

- 0 → dead → dead
- 1 → live → live
- 2 → live → dead
- 3 → dead → live

✅ Java Code (IN-PLACE 🔥)

```
public static void gameOfLife(int[][] board) {
    int n = board.length, m = board[0].length;
    int[] dir = {-1, 0, 1}; // directions for neighbors

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            int live = 0; // count live neighbors

            // Check 8 neighbors
            for (int dx : dir) {
                for (int dy : dir) {
                    if (dx == 0 && dy == 0) continue; // skip self
                    int x = i + dx, y = j + dy;
                    if (x >= 0 && x < n && y >= 0 && y < m)
                        live += board[x][y] % 2; // previous state
                }
            }
```

```
            // Apply rules using encoding
            if (board[i][j] == 1 && (live < 2 || live > 3))
                board[i][j] = 2; // live → dead
            if (board[i][j] == 0 && live == 3)
                board[i][j] = 3; // dead → live
        }
    }

    // Convert encoded state back to 0/1
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            board[i][j] %= 2;
}
```

⚠️ **Traps**

❌ Neighbor count galat

❌ State overwrite before reading

❌ Modulo trick bhoolna

🏆 **Interview Gold Line**

> “These problems test in-place greedy logic using the matrix itself as memory.”
