# AP Computer Science A

Java Programming Essentials [Ver.4.0]

## Unit 4: Data Collections

CHAPTER 17: 2D ARRAYS

DR. ERIC CHOU
IEEE SENIOR MEMBER

# AP Computer Science Curriculum

- 2D Array Creation and Access (T4.11)
- 2D Array Traversals (T4.12)

# Objectives:

- Motivation of Using 2D Arrays.

- Declare, Instantiate and initialization of a 2D array.

- 2D Array Processing I: 2D Traversal, 2D Max/Min, and 2D Shuffling.

- 2D Array Processing II: 2D Index Space, Column Major/Row Major, area copy, area move, and flip

# Objectives:

- Image Processing
- Symmetric 2D Array: k-graph, closest points
- Combination Number: Binomial Theorem
- N-D Arrays

# Overview

Lecture 1

# 2D arrays

Many applications have multidimensional structures:
- Matrix operations
- Collection of lists
- Board games (Chess, Checkers)
- Images (rows and columns of pixels)
- ...

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

# Applications

- 2D arrays are useful when data can be represented by a a grid of fixed dimensions

- Often used to represent tables, matrices, images, and game boards

- Examples of games include checkers, chess, tic-tac-toe, crosswords, and mazes
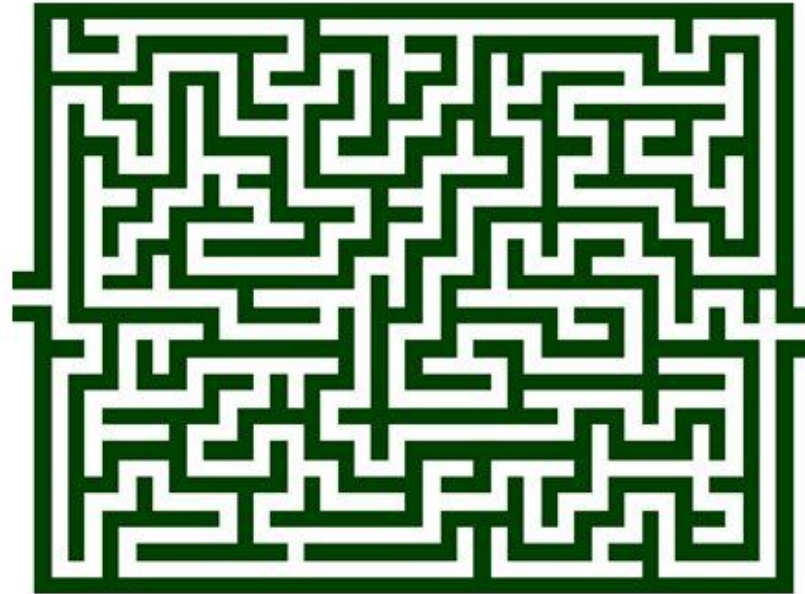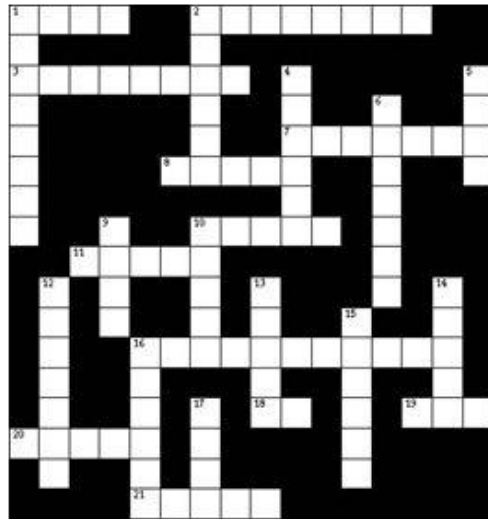
# Multidimensional Array

- Thus far, you have used one-dimensional arrays to model linear collections of elements.
- You can use a two-dimensional array to represent a matrix or a table.
- For example, the following table that describes the distances between the cities can be represented using a two-dimensional array.

Distance Table (in miles)

|          | Chicago | Boston | New York | Atlanta | Miami | Dallas | Houston |
|----------|---------|--------|----------|---------|-------|--------|---------|
| Chicago  | 0       | 983    | 787      | 714     | 1375  | 967    | 1087    |
| Boston   | 983     | 0      | 214      | 1102    | 1763  | 1723   | 1842    |
| New York | 787     | 214    | 0        | 888     | 1549  | 1548   | 1627    |
| Atlanta  | 714     | 1102   | 888      | 0       | 661   | 781    | 810     |
| Miami    | 1375    | 1763   | 1549     | 661     | 0     | 1426   | 1187    |
| Dallas   | 967     | 1723   | 1548     | 781     | 1426  | 0      | 239     |
| Houston  | 1087    | 1842   | 1627     | 810     | 1187  | 239    | 0       |

# Declare/Create Two-dimensional Arrays

```
// Declare array ref var
ElementType[][] refVar; /*or*/ ElementType refVar[][]; /*not preferred */

// Create array and assign its reference to variable
refVar = new ElementType[10][10];

// Combine declaration and creation in one statement
ElementType[][] refVar = new ElementType[10][10];

// Alternative syntax
ElementType refVar[][] = new ElementType[10][10]; /*not preferred */
```

# Declaring Variables of Two-dimensional Arrays and Creating Two-dimensional Arrays

```
int[][] matrix = new int[10][10];
 or
int matrix[][] = new int[10][10];
matrix[0][0] = 3;

for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 1000);

double[][] x;
```

# Two-dimensional Array Illustration

```
          [0] [1]  [2] [3]  [4]
     [0]  | 0 | 0 | 0 | 0 | 0 |
     [1]  | 0 | 0 | 0 | 0 | 0 |
     [2]  | 0 | 0 | 0 | 0 | 0 |
     [3]  | 0 | 0 | 0 | 0 | 0 |
     [4]  | 0 | 0 | 0 | 0 | 0 |

matrix = new int[5][5];

matrix.length?  5
matrix[0].length? 5
```

```
          [0] [1]  [2] [3]  [4]
     [0]  | 0 | 0 | 0 | 0 | 0 |
     [1]  | 0 | 0 | 0 | 0 | 0 |
     [2]  | 0 | 7 | 0 | 0 | 0 |
     [3]  | 0 | 0 | 0 | 0 | 0 |
     [4]  | 0 | 0 | 0 | 0 | 0 |

matrix[2][1] = 7;
```

```
          [0]   [1]   [2]
     [0]  | 1 |  2 |  3 |
     [1]  | 4 |  5 |  6 |
     [2]  | 7 |  8 |  9 |
     [3]  |10 | 11 | 12 |

int[][] array = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};

array.length?  4
array[0].length? 3
```

# Declaring, Creating, and Initializing Using Shorthand Notations

- You can also use an array initializer to declare, create and initialize a two-dimensional array. For example,
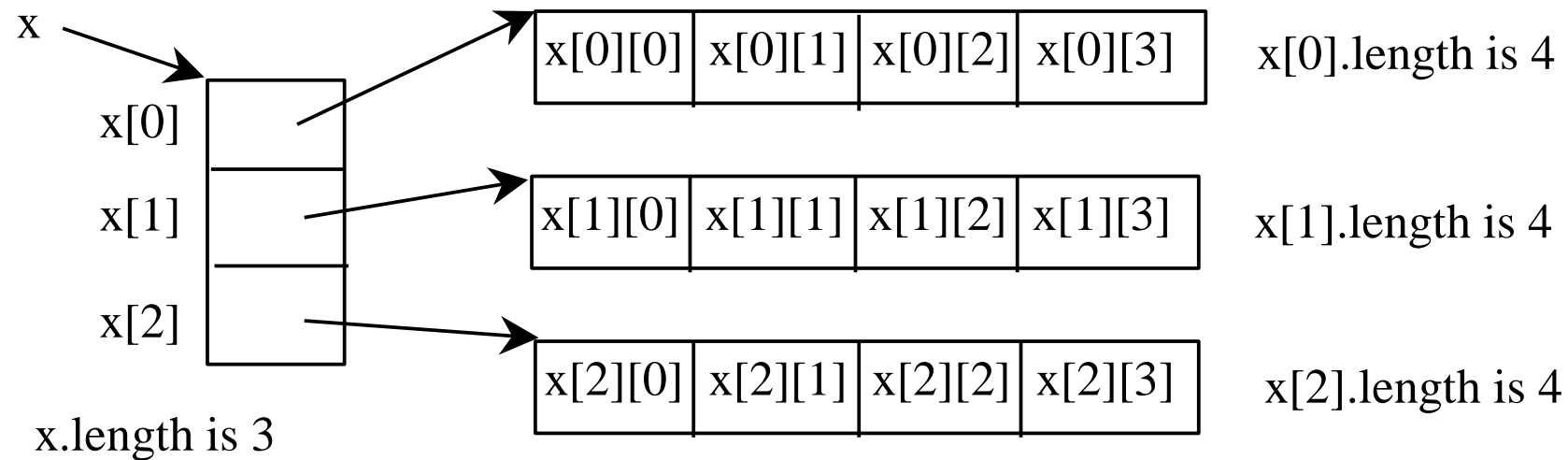
```
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

Same as

```
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

# Lengths of Two-dimensional Arrays

int[][] x = new int[3][4];

x

x[0]

x[1]

x[2]

x.length is 3

| x[0][0] | x[0][1] | x[0][2] | x[0][3] |
|---------|---------|---------|---------|

x[0].length is 4

| x[1][0] | x[1][1] | x[1][2] | x[1][3] |
|---------|---------|---------|---------|

x[1].length is 4

| x[2][0] | x[2][1] | x[2][2] | x[2][3] |
|---------|---------|---------|---------|

x[2].length is 4

# Lengths of Two-dimensional Arrays, cont.

```
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```

```
array.length
array[0].length
array[1].length
array[2].length
array[3].length
```

array[4].length

ArrayIndexOutOfBoundsException

# Ragged Arrays

- Each row in a two-dimensional array is itself an array. So, the rows can have different lengths. Such an array is known as *a ragged array*. For example,

```
int[][] matrix = {
  {1, 2, 3, 4, 5},
  {2, 3, 4, 5},
  {3, 4, 5},
  {4, 5},
  {5}
};
```

matrix.length is 5
matrix[0].length is 5
matrix[1].length is 4
matrix[2].length is 3
matrix[3].length is 2
matrix[4].length is 1

# Ragged Arrays, cont.

```
int[][] triangleArray = {
   {1, 2, 3, 4, 5},
   {2, 3, 4, 5},
   {3, 4, 5},
   {4, 5},
   {5}
};
```

| 1 | 2 | 3 | 4 | 5 |

| 2 | 3 | 4 | 5 |

| 3 | 4 | 5 |

| 4 | 5 |

| 5 |
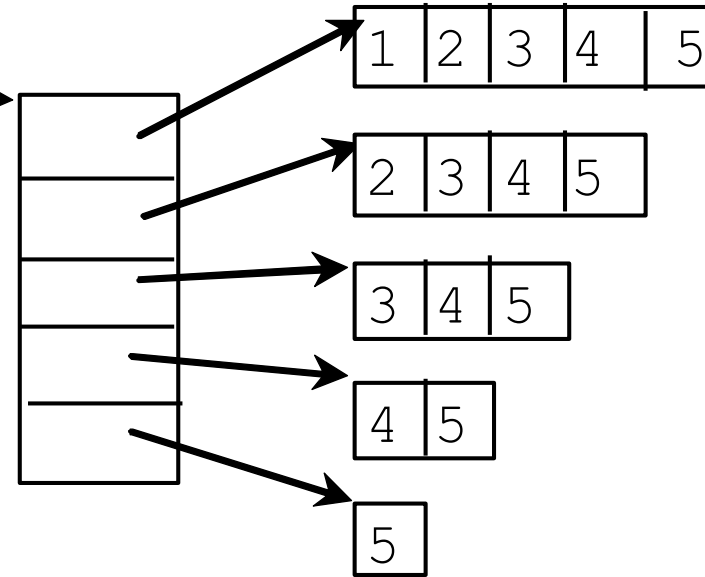
# 2D Array Processing I

Lecture 2

# Processing Two-Dimensional Arrays

1. (Initializing arrays with input values)

2. (Printing arrays)

3. (Summing all elements)

4. (Summing all elements by column)

5. (Which row has the largest sum)

6. (Finding the smallest index of the largest element)

7. (*Random shuffling*)

# Initializing arrays with input values

```java
Scanner input = new Scanner(System.in);
System.out.println("Enter " + matrix.length + " rows and " +
  matrix[0].length + " columns: ");
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column < matrix[row].length; column++) {
    matrix[row][column] = input.nextInt();
  }
}
```

# Initializing arrays with random values

```java
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column <
   matrix[row].length; column++) {
    matrix[row][column] = (int)(Math.random() *
    100);
  }
}
```

# Printing arrays

```java
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column <
    matrix[row].length; column++) {
    System.out.print(matrix[row][column] + " ");
  }
  System.out.println();
}
```

# Summing all elements

```java
int total = 0;
for (int row = 0; row < matrix.length; row++) {
  for (int column = 0; column <
   matrix[row].length; column++) {
    total += matrix[row][column];
  }
}
```

# Summing elements by column

```java
for (int column = 0; column < matrix[0].length;
    column++) {
  int total = 0;
  for (int row = 0; row < matrix.length; row++)
    total += matrix[row][column];
  System.out.println("Sum for column " + column + "
    is " + total);
}
```

# Which row has the largest sum

```java
int sum = 0;
int maxSum = Integer.MIN_VALUE; int maxRow = 0;
for (int i=0; i<matrix.length; i++){
  sum = 0;
  for (int j=0; j<matrix[i].length; j++){
     sum += matrix[i][j];
   }
  if (sum > maxSum) {maxSum = sum; maxRow = i; }
}
System.out.println("Row: "+maxRow+
               "has the largest sum="+maxSum);
```

# Finding the smallest index of the largest element

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

# Random shuffling

```java
for (int i = 0; i < matrix.length; i++) {
  for (int j = 0; j < matrix[i].length; j++) {
    int i1 = (int)(Math.random() * matrix.length);
    int j1 = (int)(Math.random() * matrix[i].length);
    // Swap matrix[i][j] with matrix[i1][j1]
    int temp = matrix[i][j];
    matrix[i][j] = matrix[i1][j1];
    matrix[i1][j1] = temp;
  }
}
```

# Demonstration Program

ArrayProcessing2D.java

Options

```
Matrix Print Out:
2 4 5 6 3
6 7 9 1 3
4 2 1 6 7
7 5 4 3 6
4 3 4 6 2

Sum of all elements: 110

Sum for column 0 is 23
Sum for column 1 is 21
Sum for column 2 is 23
Sum for column 3 is 22
Sum for column 4 is 21

Sum for row 0 is 20
Sum for row 1 is 26
Sum for row 2 is 20
Sum for row 3 is 25
Sum for row 4 is 19

Row: 1 has the largest sum=26
Smallest indeice of the largest element: matrix[1, 2] = 9
Matrix Print Out After Shuffling:
7 3 6 3 5
6 6 6 4 1
1 2 4 6 7
2 9 7 4 4
3 4 5 2 3
```

# 2D Array Processing II

Lecture 3

# Discrete Functional Model

- int[][] m = new int[5][5];
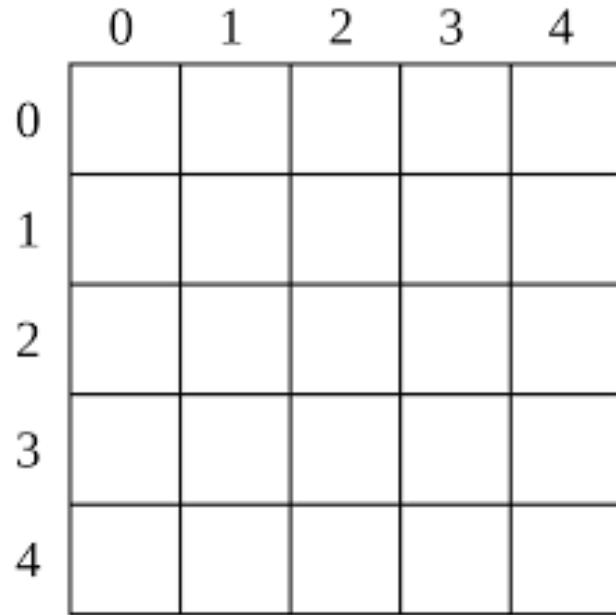


## 2-D Discrete Functional Model:
*f(x, y)* = m[i][j];
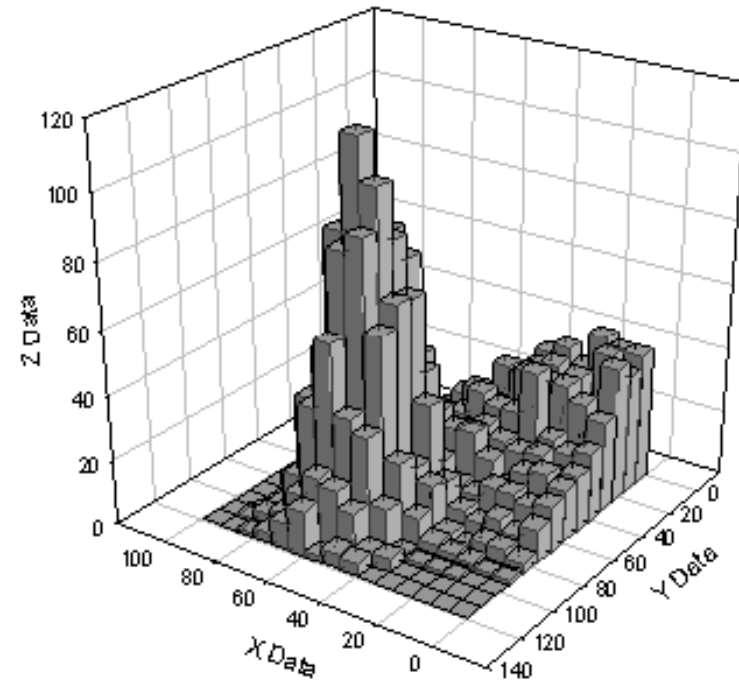
# Nested Loop

```
int i = 0;      // outer loop initial condition
/*  Before Outter Loop Processing */
while (i< m.length){
     int j = 0;              // inner loop initial condition
     /* Before Inner Loop Processing */
     while (j<m[0].length){
          /* cell processing */
          j++;
          // index update for inner loop
     } /* after inner loop processing (in-between rows(cols)) */
     i++;
          // index update of outer loop
}
/* after outer loop processing  (grand total )*/
```

# 9 x 9 multiplication table

```
int i= 1;
int sum = 0;
int[] rsum = new int[9+1];    // rsum[0] not used
while  (i<=9){
   int j = 0;
   rsum[i] = 0;    // can be omitted
   while  (j<= 9){
        rsum[i] += i * j;
        System.out.printf("%3d  ", i*j);
        j++;
   }
   System.out.println();
   sum += rsum[i];
   i++;
}
System.out.println(sum);
```

# Use (i, j) to create 2D index space

```
int i = 0;
/*  Before Outter Loop Processing */
while (i< m.length){
     int j = 0;
     /* Before Inner Loop Processing */
     while (j<m[0].length){
          /* cell processing */
          j++;
     }
     /*after inner loop processing (in-between rows(cols))*/
     i++;
}
/* after outer loop processing  (grand total )*/
```

# Column Major versus Row Major

- If the column index is used at outer loop, it is called a column major system.
- If the row index is used at the outer loop, it is called a row major system.

# 2-D Array Indexing for Traversal

**int row, col; int[][] m=new int[5][5];**
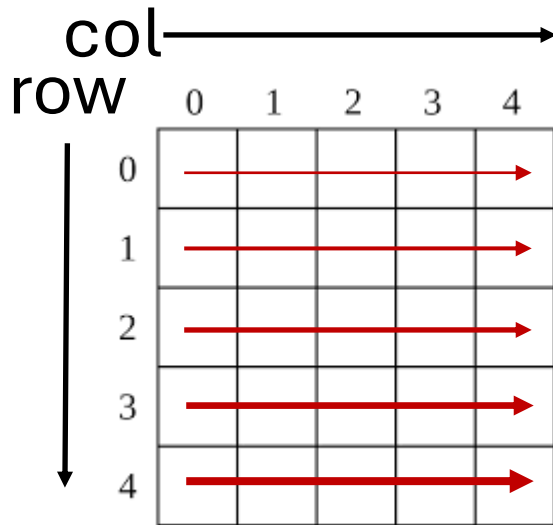


```
for(row=0; row<m.length; row++)
    for(col=0; col<m[0].length; col++)
        System.out.println(m[row][col]);
```
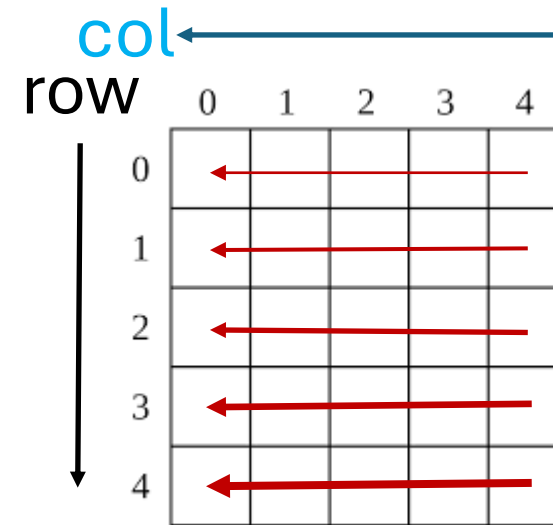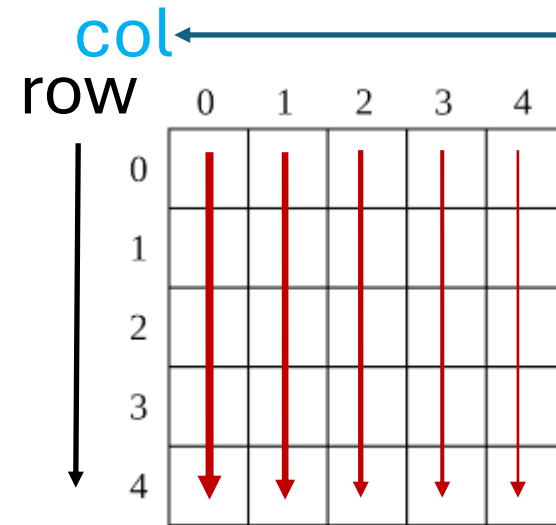
```
for(row=0; row<m.length; row++)
    for(col=m[0].length-1; col>=0; col--)
        System.out.println(m[row][col]);
```

# 2-D Array Indexing for Traversal
## int row, col; int[][] m=new int[5][5];
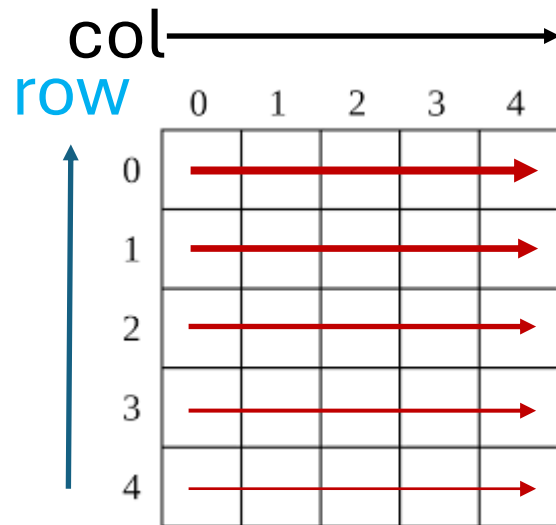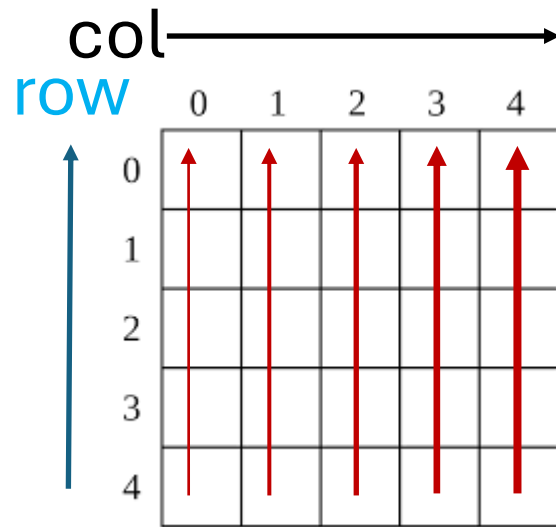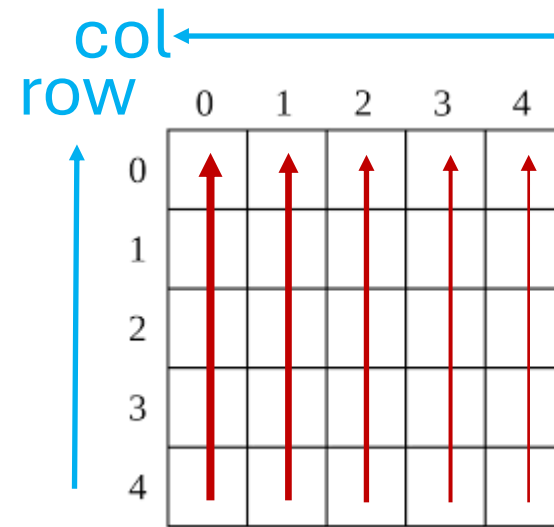


```
for(row=m.length-1; row>=0; row--)
    for(col=0; col<m[0].length; col++)
        System.out.println(m[row][col]);
```
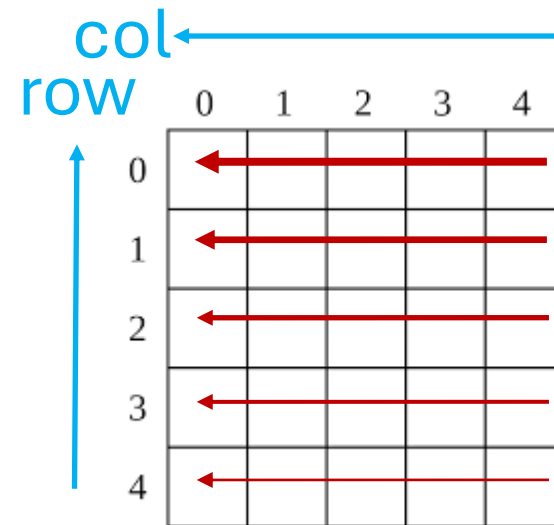
```
for(row=m.length-1; row>=0; row--)
    for(col=m[0].length-1; col>=0; col--)
        System.out.println(m[row][col]);
```

# 2-D Array Indexing for Traversal

**int row, col; int[][] m=new int[5][5];**

col ⟶

row

|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 |   |   |   |   |   |
| 1 |   |   |   |   |   |
| 2 |   |   |   |   |   |
| 3 |   |   |   |   |   |
| 4 |   |   |   |   |   |
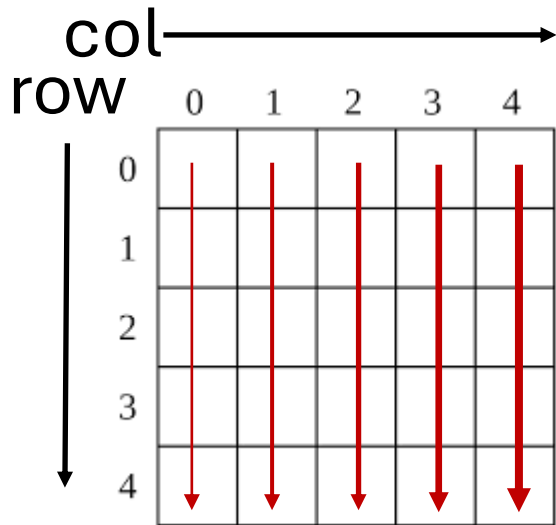
```
for(col=0; col<m[0].length; col++)
    for(row=0; row<m.length; row++)
        System.out.println(m[row][col]);
```

col ⟵

row

|   | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 |   |   |   |   |   |
| 1 |   |   |   |   |   |
| 2 |   |   |   |   |   |
| 3 |   |   |   |   |   |
| 4 |   |   |   |   |   |

```
for(col=m[0].length-1; col>=0; col--)
    for(row=0; row<m.length; row++)
        System.out.println(m[row][col]);
```

# 2-D Array Indexing for Traversal
## int row, col; int[][] m=new int[5][5];



```
for(col=0; col<m[0].length; col++)
    for(row=m.length-1; row>=0; row--)
        System.out.println(m[row][col]);
```

```
for(col=m[0].length-1; col>=0; col--)
    for(row=m.length-1; row>=0; row--)
        System.out.println(m[row][col]);
```
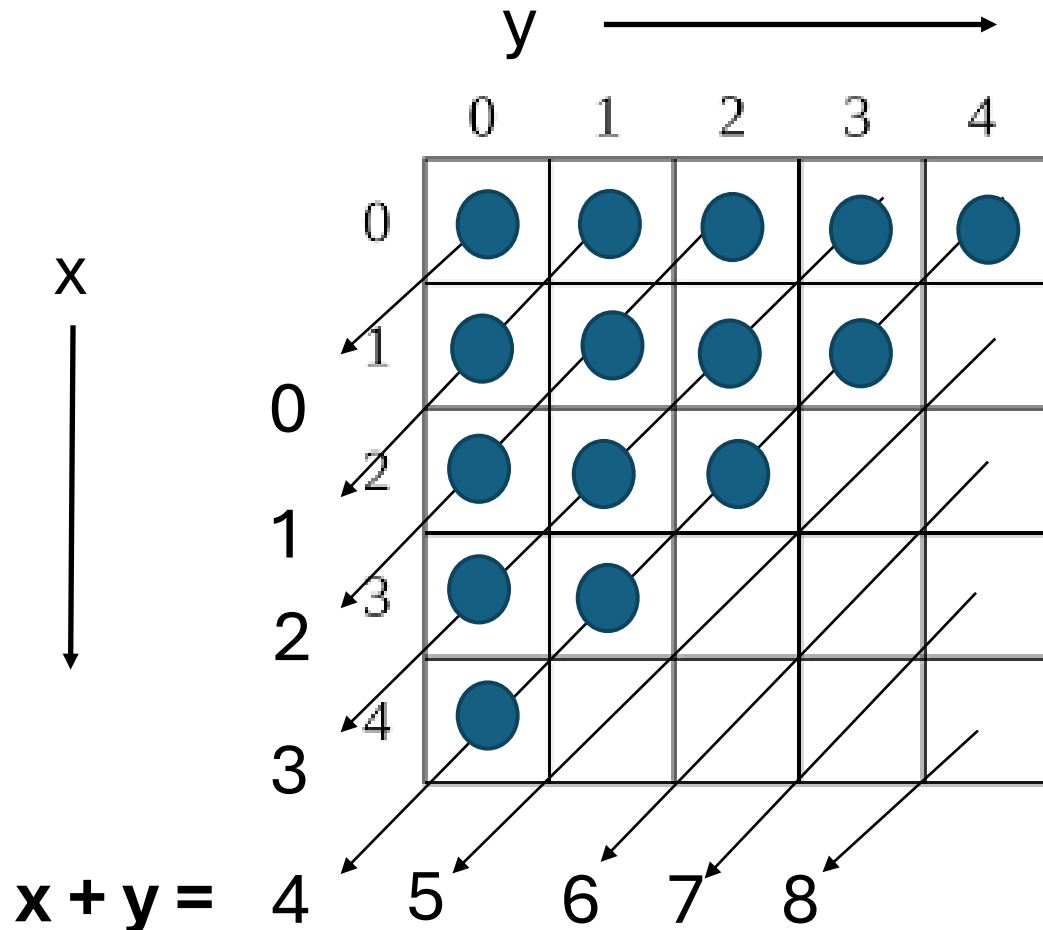
# Partial Array Traversal
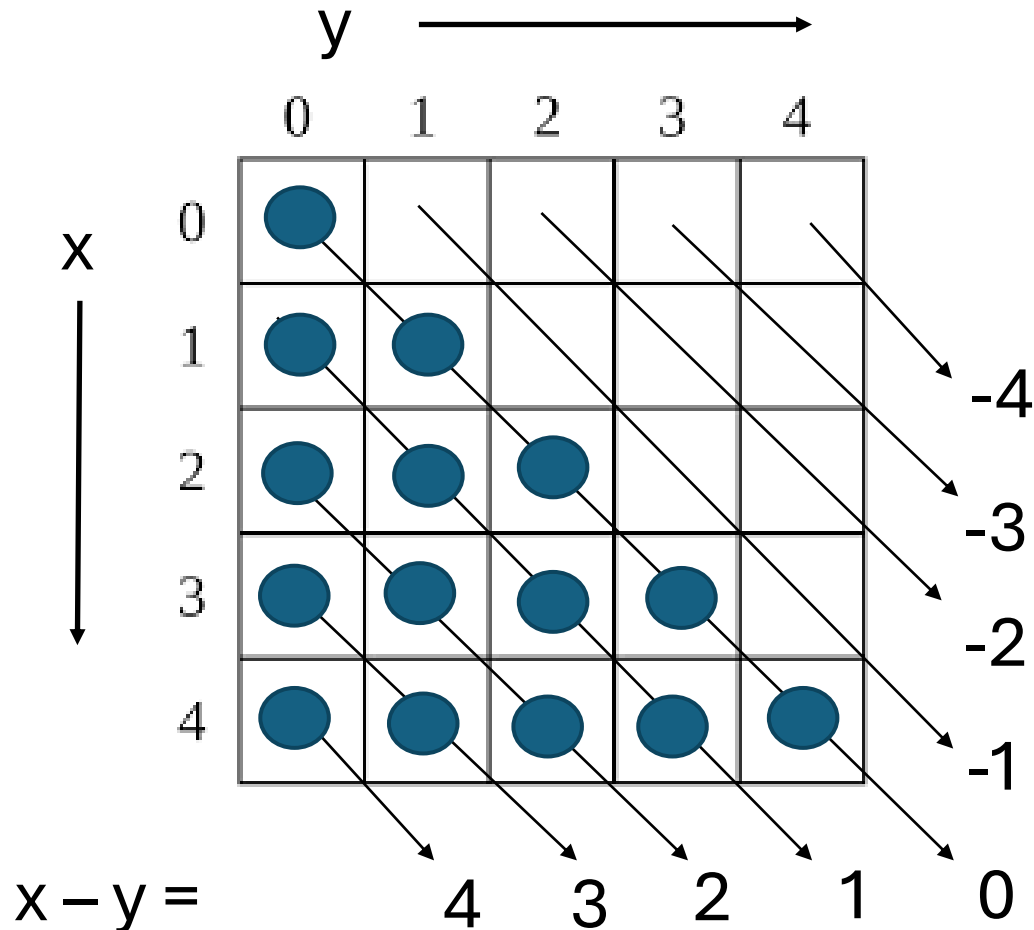


```
for (int I = 0; i<m.length; i++)
    for (int j =0; j< i +1;   j ++)
        { /* do something */}
```

Index:
  Stop Condition: j stop at j = i.
  **i – j = 0;**

# Partial Array Traversal



```
for (int i = 0; i<m.length; i++)
    for (int j =m.length-1-i; j>=0;  j --)
        { /* do something */}
```
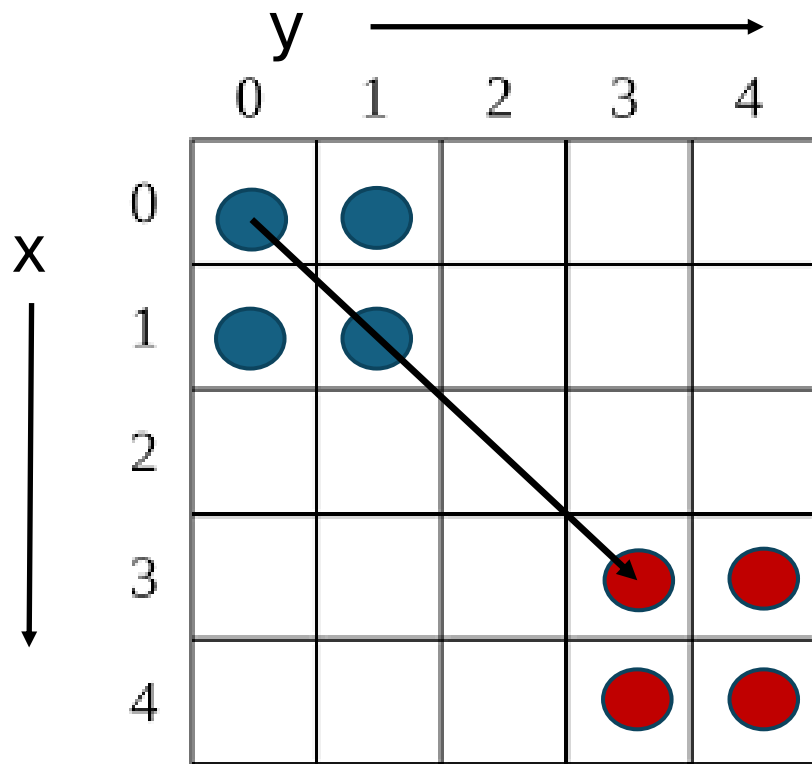
Index:

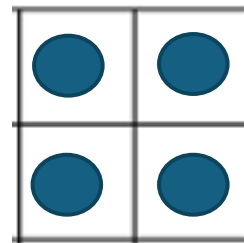  Start Condition: i + j = m.length-1;

  **i + j = 4;**

# Partial Array Traversal
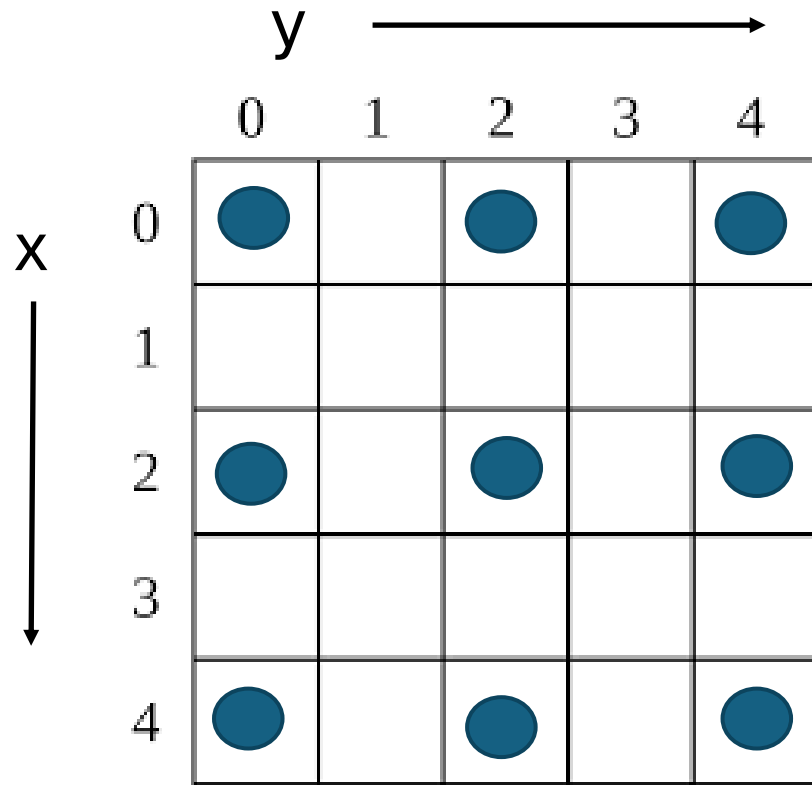


```
for (int i = 0; i<m.length; i+=2)
    for (int j =0; j<m[0].length;  j+=2)
        { /* do something */}
```
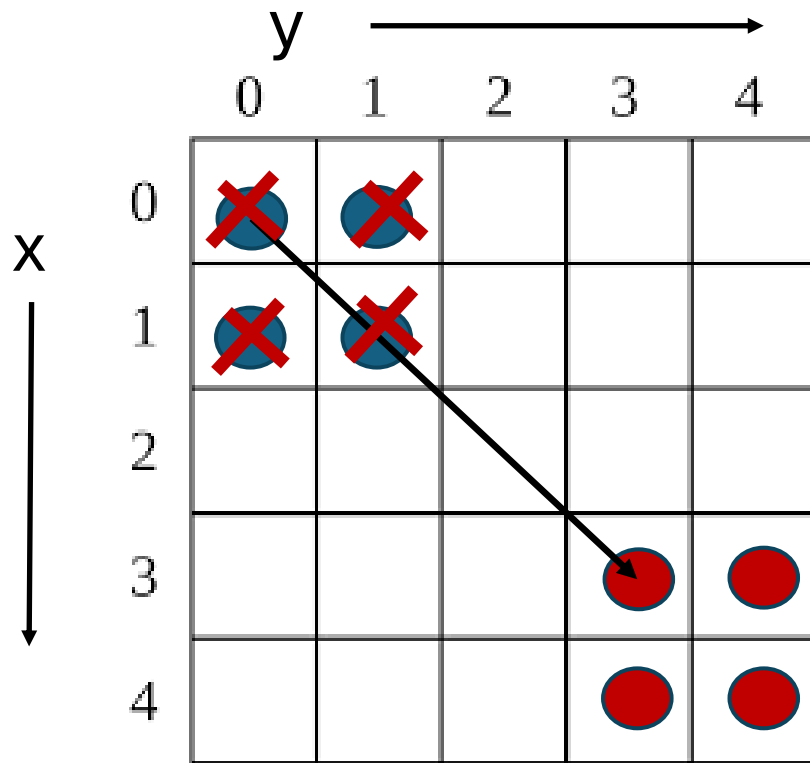
# Vector Operation (Area Copy)



```
for (int i = 0; i<2; i++)
    for (int j=0; j<2; j++)
        m[3+i][3+j] = m[0+i][0+j];
            /* 0 is not needed */
```
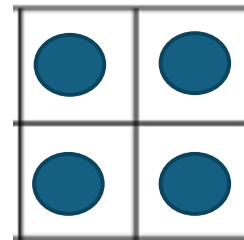
Area to be Copied: **2 x 2** block
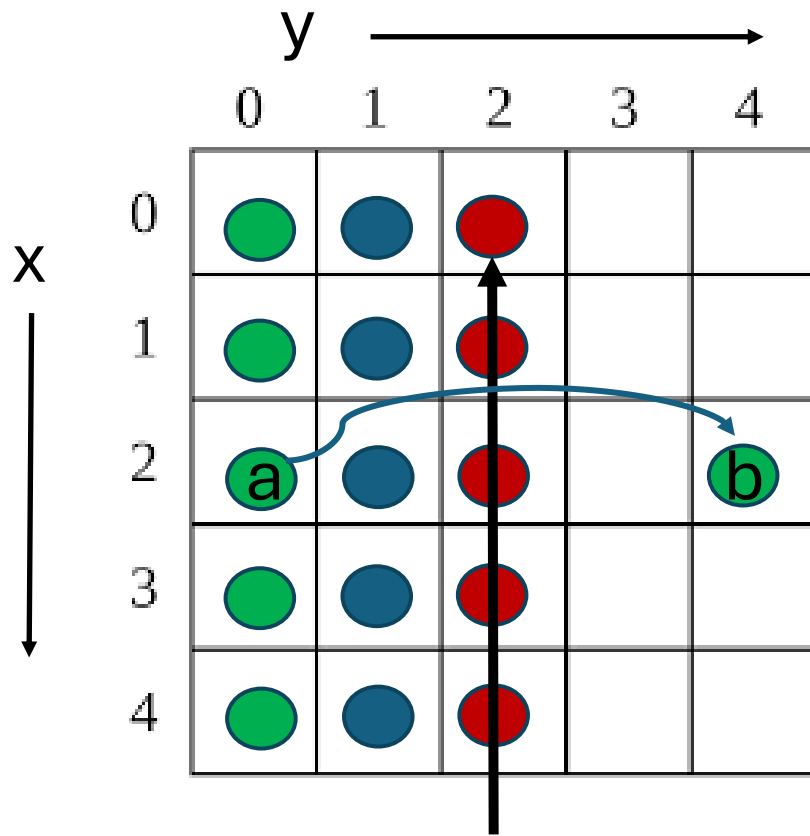
From (0, 0) to (3, 3)

# Vector Operation (Area Move)



```
for (int i = 0; i<2; i++)
    for (int j=0; j<2; j++)
        m[3+i][3+j] = m[0+i][0+j];
for (int i = 0; i<2; i++)
    for (int j=0; j<2; j++)
        m[0+i][0+j]=0;
```

Area to be Copied: **2 x 2** block

From (0, 0) to (3, 3)

# Flip



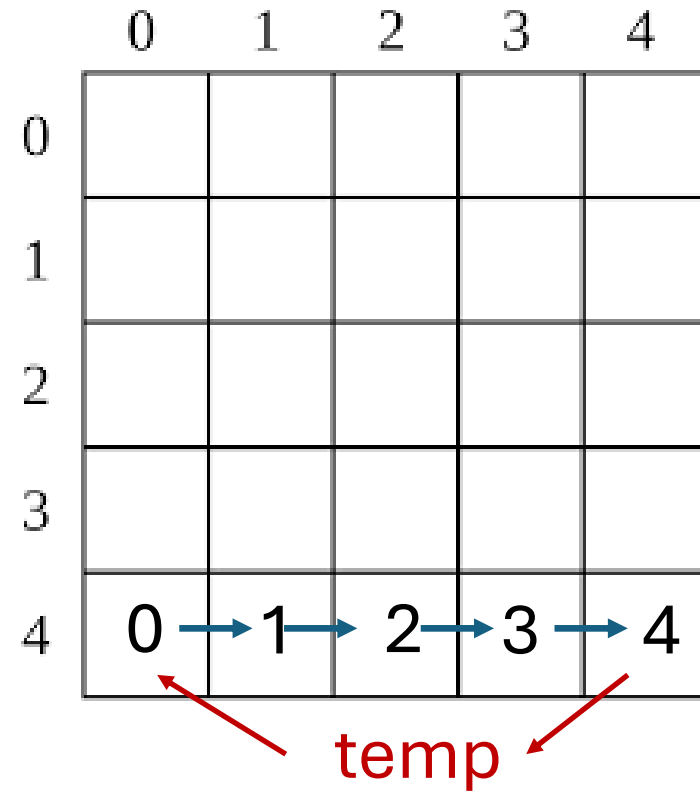## Symmetric Line for Flipping:

```
j = 2;     // j = m.length/2

(a + b)/2 = 2;
b == 4-a;     // b = m.length -a-1;

for (int i=0; i<m.length; i++){
    for (int j=0; j<m.length/2; j++){
        m[i][m.length-j-1] = m[i][j];
    }
}
```

# Area Shift

```
for (int i=0; i<m.length; i++){
    int temp = m[i][m.length-1];
    for (int j=m[0].length-2; j<=0; j--){
        m[i][j+1] = m[i][j];
    }
    m[i][0] =temp;
}
```



One Row after Shift: 4    0    1    2    3