

# Two-Dimensional Arrays



EECS2030: Advanced  
Object Oriented Programming  
Fall 2018

CHEN-WEI WANG

## 2-D Arrays: Motivating Example (1)

Consider a table of distances between seven cities:

	Chicago	Boston	New York	Atlanta	Miami	Dallas	Houston
Chicago	0	983	787	714	<b>1375</b>	967	1087
Boston	<b>983</b>	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	<b>1187</b>
Dallas	967	1723	1548	781	1426	0	239
Houston	1087	1842	1627	810	1187	239	0

As part of the program for an airline reservation system, the *distance of a trip* with *multiple stop-overs* is to be calculated in order to accumulate the milage of frequent flyers.

e.g., A trip {Boston, Chicago, Miami, Houston} takes  $983$  (B-to-C) +  $1375$  (C-to-M) +  $1187$  (M-to-H) =  $3545$  miles

**Question:** How do you manipulate such information in Java?

## 2-D Arrays: Motivating Example (2.1)

Here is a solution based on what we've learnt so far:

- Fix the “positions” of cities in the table as constants:

```
final int CHICAGO = 0;  
final int BOSTON = 1;  
final int MIAMI = 4;
```

- Represent each (horizontal) row using a one-dimensional array:

```
int[] fromChicago = {0, 983, 787, 714, 1375, 967, 1087}  
int[] fromBoston = {983, 0, 214, 1102, 1763, 1723, 1842}  
int[] fromMiami = {1375, 1763, 1549, 661, 0, 1426, 1187}
```

- Given an itinerary {Boston, Chicago, Miami, Houston}, choose the corresponding arrays in the right order:

```
int[] dist = fromBoston[CHICAGO]  
           + fromChicago[MIAMI]  
           + fromMiami[HUSTON];
```

## 2-D Arrays: Motivating Example (2.2)

What if cities of an itinerary are read from the user?

```
1 Scanner input = new Scanner(System.in);
2 System.out.println("How many cities?");
3 int howMany = input.nextInt(); input.nextLine();
4 String[] trip = new String[howMany];
5 /* Read cities in the trip from the user. */
6 for(int i = 0; i < howMany; i++) {
7     System.out.println("Enter a city:");
8     trip[i] = input.nextLine();
9 }
10 /* Add up source-to-destination distances. */
11 int dist = 0;
12 for(int i = 0; i < howMany - 1; i++) {
13     String src = trip[i];
14     String dst = trip[i + 1];
15     /* How to accumulate the distance between src and dst? */
16 }
```

## 2-D Arrays: Motivating Example (2.3)

Given a source and a destination, we need to *explicitly* select:

- The corresponding **source row** [e.g., fromBoston]
- The corresponding **destination index** [e.g., CHICAGO]

```
13 String src = trip[i];
14 String dst = trip[i + 1];
15 if(src.equals("Chicago")) {
16     if(dst.equals("Boston")) {dist += fromChicago[BOSTON];}
17     else if(dst.equals("New York")) {dist += fromChicago[NY];}
18     ...
19 }
20 else if(src.equals("Boston")) {
21     if(dst.equals("Chicago")) {dist += fromBoston[CHICAGO];}
22     else if(dst.equals("NEW YORK")) {dist += fromBoston[NY];}
23     ...
24 }
25 ...
```

- Drawback?

$7 \times (7 - 1)$  possibilities to program!

## 2-D Arrays: Initialization (1)

A **2D array** is really *an array of arrays*

	[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];
```

	[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}
};
```

## 2-D Arrays: Initialization (1)

A **2D array** may be initialized either at the time of declaration, or after declaration.

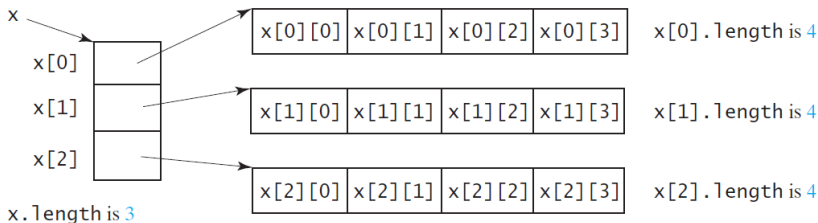
```
int[][] array = {  
    {1, 2, 3},  
    {4, 5, 6},  
    {7, 8, 9},  
    {10, 11, 12}  
};
```

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;
```

## 2-D Arrays: Lengths (1)

For a **2D array**, you may query about its *size*, or *sizes* of its component arrays.



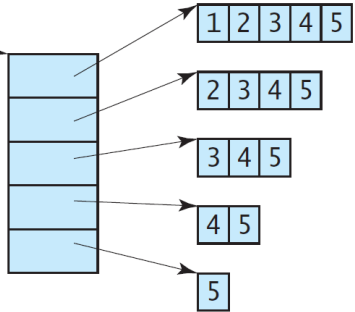


## 2-D Arrays: Lengths (2)

For a **2D array**, its components may have different *sizes*.

e.g.,

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



## 2-D Arrays: Assignments

For a **2D array**, access a slot via its *row* and *column*.

e.g.,

	[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;`

## Revisiting the Motivating Example

```
double[][] distances = {  
    {0, 983, 787, 714, 1375, 967, 1087},  
    {983, 0, 214, 1102, 1763, 1723, 1842},  
    {787, 214, 0, 888, 1549, 1548, 1627},  
    {714, 1102, 888, 0, 661, 781, 810},  
    {1375, 1763, 1549, 661, 0, 1426, 1187},  
    {967, 1723, 1548, 781, 1426, 0, 239},  
    {1087, 1842, 1627, 810, 1187, 239, 0},  
};
```

```
final int CHICAGO = 0;
```

```
final int BOSTON = 1;
```

```
...
```

```
final int HOUSTON = 6;
```

```
int MiamiToBoston = distances[MIAMI][BOSTON];
```

```
int BostonToNewYork = distances[BOSTON][NEWYORK];
```

```
int MiamiToNewYork = MiamiToBoston + BostonToNewYork;
```

## Two Dimensional Arrays: Example (1)

**Problem:** Given a 2D array `a` of integers, print out all its values: first row, second row, third row, and so on.

```
1 for(int row = 0; row < a.length; row ++) {  
2     System.out.print("Row" + row);  
3     for(int col = 0; col < a[row].length; col ++) {  
4         System.out.print(a[row][col]);  
5     }  
6     System.out.println(); }
```

- In **L1**, we write `a.length` so that it will print out exactly that many rows in the matrix.
- In **L3**, we write `a[row].length` so that it will print out according to how large the row `a[row]` is.

## Two Dimensional Arrays: Example (2)

**Problem:** Given a 2D array `a` of integers, calculate the average of its values.

```
int total = 0;
int numOfElements = 0;
for(int row = 0; row < a.length; row++) {
    for(int col = 0; col < a[row].length; col++) {
        total += a[row][col];
        numOfElements++;
    }
}
double average = ((double) total) / numOfElements;
System.out.println("Average is " + average);
```

- Why is the `numOfElements` counter necessary?
- Divide `total` by `a.length * a[0].length` instead?

## Two Dimensional Arrays: Example (3)

**Problem:** Given a 2D array `a` of integers, find out its *maximum* and *minimum* values.

```
int max = a[0][0];
int min = a[0][0];
for(int row = 0; row < a.length; row ++) {
    for(int col = 0; col < a[row].length; col ++) {
        if (a[row][col] > max) {
            max = a[row][col];
        }
        if (a[row][col] < min) {
            min = a[row][col];
        }
    }
}
System.out.println("Maximum is " + max);
System.out.println("Minimum is " + min);
```

## Two Dimensional Arrays: Example (4.1)

**Problem:** Given a 2D array `a` of integers, find out the row which has the *maximum* sum.

```
1  int maxRow = 0; int maxSum = 0;
2  for(int col=0; col < a[0].length; col++){maxSum += a[0][col];}
3  for(int row = 1; row < a.length; row++) {
4      int sum = 0;
5      for(int col = 0; col < a[row].length; col++) {
6          sum += a[row][col];
7      }
8      if (sum > maxSum) {
9          maxRow = row;
10         maxSum = sum;
11     }
12 }
13 System.out.print("Row at index " + maxRow);
14 System.out.println(" has the maximum sum " + maxSum);
```

**Q:** What if statement `int sum = 0;` at L4 is moved, outside the for-loop, between L2 and L3?

## Two Dimensional Arrays: Example (5)

**Problem:** Given a 2D array `a` of integers, determine if all elements are positive.

```
boolean allPos = true;
for(int row = 0; row < a.length; row++) {
    for(int col = 0; col < a[row].length; col++) {
        allPos = allPos && a[row][col] > 0;
    }
}
if (allPos) { /* print */ } else { /* print */ }
```

Alternatively (with *early exit*):

```
boolean allPos = true;
for(int row = 0; allPos && row < a.length; row++) {
    for(int col = 0; allPos && col < a[row].length; col++) {
        allPos = a[row][col] > 0;
    }
}
if (allPos) { /* print */ } else { /* print */ }
```



## Two Dimensional Arrays: Example (6.1)

**Problem:** Given a 2D array *a* of integers, determine if it is a *rectangle* (i.e., each row has the same number of columns).

```
if(a.length == 0) { /* empty array can't be a rectangle */ }
else { /* a.length > 0 */
    int assumedLength = a[0].length;
    boolean isRectangle = true;
    for(int row = 0; row < a.length; row++) {
        isRectangle =
            isRectangle && a[row].length == assumedLength;
    }
    if (isRectangle) { /* print */ } else { /* print */ }
}
```

**Exercise:** Change the above code so that it exits from the loop *as soon as* it is found that the 2-D array is not a rectangle.

## Two Dimensional Arrays: Example (6.2)

**Problem:** Given a 2D array `a` of integers, determine if it is a *square* (i.e., each row has the same number of columns, and that number is equal to the number of rows of the 2-D array).

```
if(a.length == 0) { /* empty array can't be a square */ }
else { /* a.length > 0 */
    int assumedLength = a.length;
    boolean isSquare = a[0].length == assumedLength;
    for(int row = 0; row < a.length; row++) {
        isSquare =
            isSquare && a[row].length == assumedLength;
    }
    if (isSquare) { /* print */ } else { /* print */ }
}
```

**Exercise:** Change the above code so that it exits from the loop *as soon as* it is found that the 2-D array is not a square.

## Two Dimensional Arrays: Example (7)

- Problem:** Given a 2D array `a` of integers, print out the *lower-left triangular* area of elements.

**Assumption:** The input 2D array is of a *square* shape.

```
for(int row = 0; row < a.length; row++) {  
    for(int col = 0; col <= row; col++) {  
        System.out.print(a[row][col]); }  
    System.out.println(); }
```

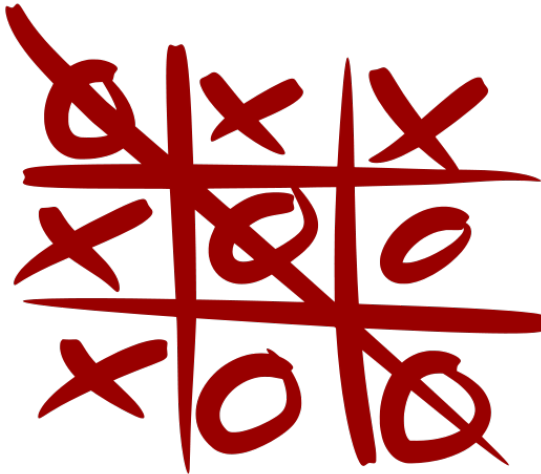
- Problem:** *upper-left triangular* area?

```
for(int row = 0; row < a.length; row++) {  
    for(int col = 0; col < a[row].length - row; col++) {  
        System.out.print(a[row][col]); }  
    System.out.println(); }
```

**Exercises:** *upper-right triangle? lower-right triangle?*

## 2-D Arrays: Example (8)

Consider the tic-tac-toe game:



# Index (1)

---

- 2-D Arrays: Motivating Example (1)
- 2-D Arrays: Motivating Example (2.1)
- 2-D Arrays: Motivating Example (2.2)
- 2-D Arrays: Motivating Example (2.3)
- 2-D Arrays: Initialization (1)
- 2-D Arrays: Initialization (1)
- 2-D Arrays: Lengths (1)
- 2-D Arrays: Lengths (2)
- 2-D Arrays: Assignments
- Revisiting the Motivating Example
- Two Dimensional Arrays: Example (1)
- Two Dimensional Arrays: Example (2)
- Two Dimensional Arrays: Example (3)
- Two Dimensional Arrays: Example (4.1)

## **Index (2)**

---

**Two Dimensional Arrays: Example (5)**

**Two Dimensional Arrays: Example (6.1)**

**Two Dimensional Arrays: Example (6.2)**

**Two Dimensional Arrays: Example (7)**

**2-D Arrays: Example (8)**