Unit 8: 2D Arrays
Introduction to 2D Arrays

Adapted from:

1) Building Java Programs: A Back to Basics Approach
by Stuart Reges and Marty Stepp

2) Runestone CSAwesome Curriculum

https://longbaonguyen.github.io

2D Arrays

We have only worked with one-dimensional arrays so far, which have a single row of elements.

But in the real world, data is often represented in a two-dimensional table with rows and columns.

Programming languages can also represent arrays this way with multiple dimensions.

A **two-dimensional (2D) array** has rows and columns. A 2D array in Java is actually an array of arrays.

A **row** has horizontal elements. A **column** has vertical elements. In the picture below there are 3 rows of lockers and 6 columns.

2D Arrays

A **two-dimensional (2D) array** has rows and columns.

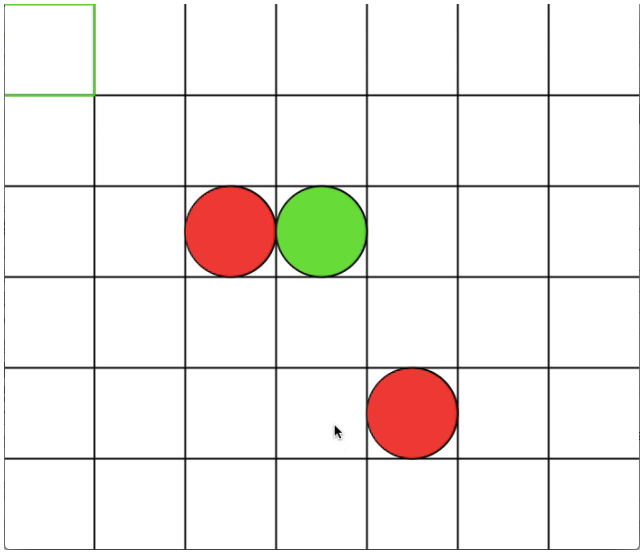
A **row** has horizontal elements. A **column** has vertical elements. In the picture below there are 3 rows of lockers and 6 columns.

Figure 1: Lockers in rows and columns

2D Arrays

Two dimensional arrays are especially useful when the data is naturally organized in rows and columns like in a spreadsheet, bingo, battleship, theater seats, classroom seats, connect-four game, or a picture.

One of our labs, we will write a program that can be later used to write Connect Four or Go.



2D Arrays

Many programming languages actually store two-dimensional array data in a one-dimensional array. The typical way to do this is to store all the data for the first row followed by all the data for the second row and so on. This is called **row-major** order.

Some languages store all the data for the first column followed by all the data for the second column and so on. This called **column-major** order.

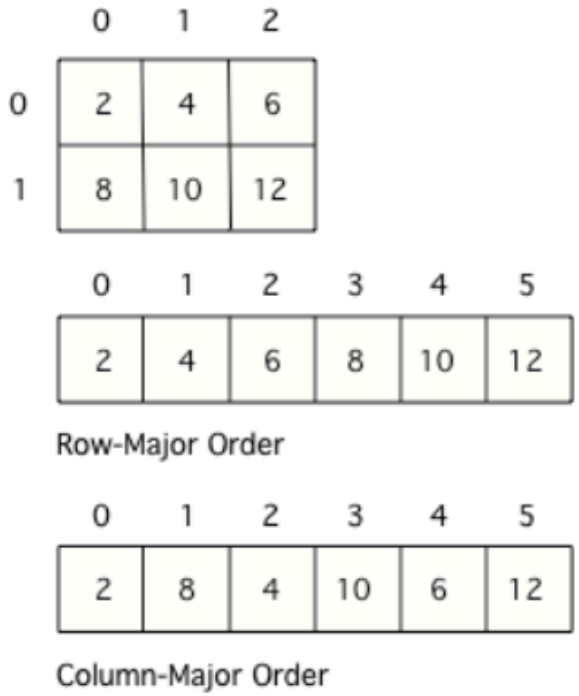


Figure 1: A 2D array stored in row-major order or column-major order as a 1D array.

Declare and Initialize

To declare **and** initialize a 2D array,

```
type[][] name = new type[row][col];
```

where row, col is the number of rows/columns. When arrays are created their contents are automatically initialized to 0 for numeric types, null for object references, and false for type boolean.

```
int[][] matrix = new int[3][4]; //3 rows, 4 columns
                                //initialized to 0.
```

0	0	0	0
0	0	0	0
0	0	0	0

2D Array

To explicitly put a value in an array, you can use assignment statements specifying the row and column of the entry.

```
int[][] matrix = new int[3][4]; //3 rows, 4 columns
                                //initialized to 0.
matrix[0][0] = 2;
matrix[1][2] = -6;
matrix[2][1] = 7;
```

2	0	0	0
0	0	-6	0
0	7	0	0

Initializer List

You can also initialize (set) the values for the array when you create it. In this case you don't need to specify the size of the array, it will be determined from the values you give. This is called using initializer list.

```
int[] array={1,4,3}; // 1D array initializer list.
// 2D array initializer list.
int[][] mat = {{3,4,5}, {6,7,8}};
// 2 rows, 3 columns
```

3	4	5
6	7	8

Declare and Initialize

Declaring and initializing 2D arrays.

```
int[][] table; //2D array of ints, null reference

double[][] matrix=new double[4][5];
// 4 rows, 5 columns
// initialized all to 0.0

String[][] strs=new String[2][5];
// strs reference 2x5 array of
// String objects. Each element is
// null

// Using initializer list.
String[][] seatingInfo = {{"Jamal", "Maria"},
                          {"Jake", "Suzy"}, {"Emma", "Luke"}};
```

Array of Arrays

A 2D array is implemented as an array of row arrays. Each row is a one-dimensional array of elements. Suppose that mat is the 2D array:

3	-4	1	2
6	0	8	1
-2	9	1	7

Then mat is an array of three arrays:

mat[0] is the one-dimensional array {3,-4,1,2}.

mat[1] is the one-dimensional array {6,0,8,1}.

mat[2] is the one-dimensional array {-2,9,1,7}.

mat.length is the number of rows.

Array of Arrays

3	-4	1	2
6	0	8	1
-2	9	1	7

1) mat.length is the number of rows. In this case, it equals 3 because there are three row-arrays in mat.

2) For each k, where 0<=k<mat.length, mat[k].length is the number of elements in that row, namely the number of columns. In this case, mat[k].length=4 for all k.

3) Java allows "jagged arrays" where each row array may have different lengths. **However, on the AP exam, assume all arrays are rectangular.**

Example

```
int[][] mat={{3,4,5},{1,2},{0,1,-3,5}};
```

```
mat[0] = {3,4,5}
mat[1] = {1,2}
mat[2] = {0,1,-3,5}
```

```
mat.length = 3
mat[0].length = 3
mat[1].length = 2
mat[2].length = 4
```

Traversing a 2D Array

Suppose that `mat` is a 2D array initialized with integers. Use nested for loop to print out the elements of the array. Traverse by row-major order.

```
int[][] mat = {{3,4,5},{1,2},{0,1,-3,5}};
for(int row = 0;i < mat.length;i++){
  for(int col = 0;j < mat[row].length;j++)
      System.out.print(mat[row][col]+ " ");
  System.out.println();
}
```

Output:

3 4 5

1 2

0 1 -3 5

For Each Traversal

Traverse an array by using a for each loop. For each loop, in general, are much easier to work with. If you are not modifying your 2D array, it is highly recommended that you use for each to avoid index errors.

```
int[][] mat = {{3,4,5},{1,2},{0,1,-3,5}};
for(int[] row: mat){
  for(int element: row)
      System.out.println(element + " ");
  System.out.println();
}
```

Output:

3 4 5

1 2

0 1 -3 5

Row-by-Row

Suppose the following method has been implemented which prints a 1D array.

```
// print out elements of array separated by spaces
public void printArray(int[] array)
{ /*implementation not shown*/ }
```

Use it to print out the 2D array `mat` by processing one row at a time(row-by-row).

```
for(int i = 0;i < mat.length; i++){
  printArray(mat[i]);   //mat[i] is row i of mat
  System.out.println();
}
```

2D Arrays of Objects

```
Point[][] pointMatrix;
```

Suppose that pointMatrix is initialized with Point objects. Change the x-coordinate of each Point to 1.

```
for(int row = 0;row < pointMatrix.length;row++)
  for(int col = 0;col < pointMatrix[0].length;col++)
      pointMatrix[row][col].setX(1);
```

Lab 1

Write the following methods.

`sum`: Write method `sum` which accepts a 2D array of integers and returns the sum of all of the elements. Use row-column traversal method. **Use a regular nested Loop.**

`rowSum`: `rowSum` accepts two parameters: a 2D array of integers and an integer `row`. `rowSum` returns the sum of the integers of elements in the row given by `row`.

`colSum`: `colSum` accepts two parameters: a 2D array of integers and an integer `col`. `colSum` returns the sum of the integers of elements in the column given by `col`.

`sum2`: This method is the same as `sum` above **but you must use `rowSum method` in your code. One loop.**

Lab 1

Write the following methods.

`largest` accepts a 2D array of integers and returns the largest value. Use row-column traversal method to examine each value. **Use a nested for each loop.**

`largestByRow` accepts two parameters: a 2D array of integers and an integer `row`. `largestByRow` returns the largest value in the row given by `row`.

`largest2` accepts a 2D array of integers and returns the largest value. **You must call `largestByRow`. One loop.**

Lab 1

`printTranspose`: Given 2D array of integers, print the transpose of the array. The transpose of a 2D array is the array whose rows are the columns of the original array. **Do not create a new array, instead, use for loops to traverse the original array.**

If `mat={{1,2,3},{4,5,6}}; printTranspose(mat)` will print:

1 4
2 5
3 6

Lab 2

A magic square is an N x N array of numbers such that

1. Every number from 1 through N^2 must appear exactly once.
2. Every row, column, major and minor diagonal must add up to the same total.

Example: N=4

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

Lab 2

Write the class `MagicSquare` with instance methods given in the next few slides. MagicSquare should have an instance 2D array variable `square`. MagicSquare should have a constructor that accepts a 2D array.

The methods `rowSum`, `colSum`, `diagSums` and `exactlyOnce` are intermediate methods to help you write the `isMagic` method, which determines whether a square is magic.

You must use the method headers indicated for each method.

Write a driver class with a main method to test your MagicSquare class.

Lab 2

```
public int rowSum(int row){…}
```

Returns the row sum indicated by `row`.

```
public int colSum(int col){…}
```

Returns the column sum indicated by `col`.

Lab 2

`public boolean diagSums(int sum){…}`

Returns whether both the major and minor diagonal sums are equal to `sum`. The major and minor diagonal are highlighted below.

16	3	2	**13**
5	**10**	**11**	8
9	**6**	**7**	12
4	15	14	**1**

Lab 2

`public boolean exactlyOnce(){…}`

Returns true if the numbers 1 to N^2 occurs exactly once in `square` and false otherwise. N is the number of rows(and columns) in `square`. Hint: Use a tally array discussed in the array algorithms lecture.

You must use the each of the above methods to write the following isMagic method.

`public boolean isMagic(){…}`

Returns true if `square` is magic and false otherwise.