AP Computer Science A

Mr. Folwell
nfolwell@schools.nyc.gov

Unit 8: 2D Arrays

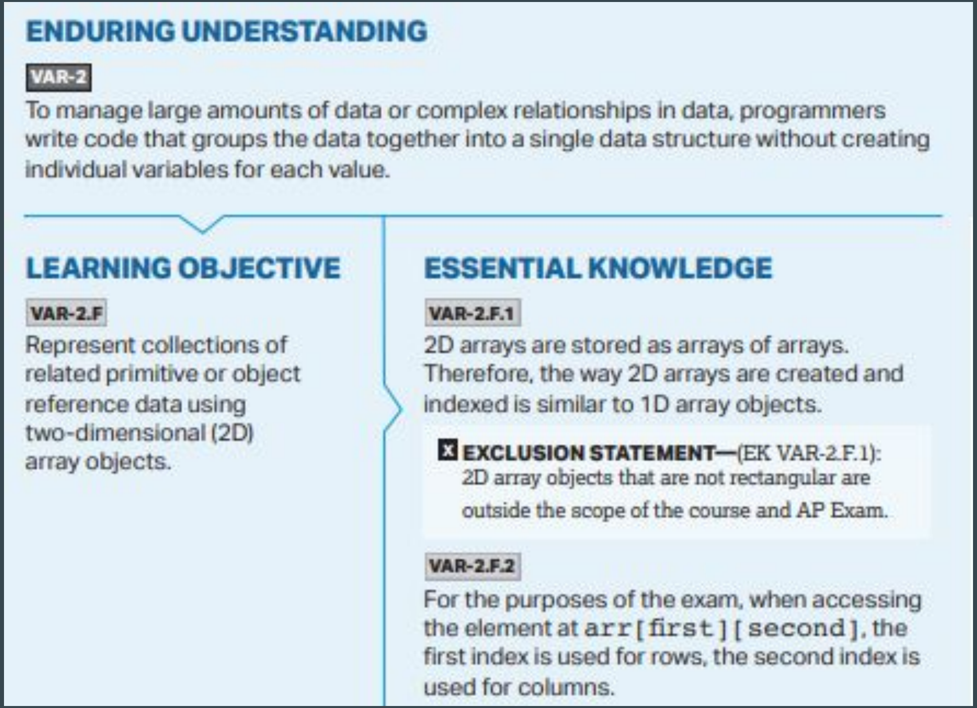
Intro to 2D Arrays

3/6/2023

UNIT 8 2D Array

CollegeBoard Standards

Unit 8 Topic 1



ENDURING UNDERSTANDING

VAR-2

To manage large amounts of data or complex relationships in data, programmers write code that groups the data together into a single data structure without creating individual variables for each value.

LEARNING OBJECTIVE

VAR-2.F

Represent collections of related primitive or object reference data using two-dimensional (2D) array objects.

ESSENTIAL KNOWLEDGE

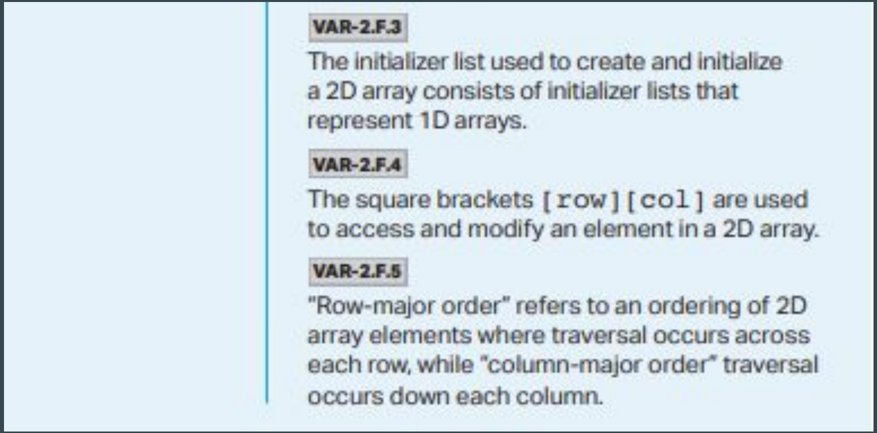
VAR-2.F.1

2D arrays are stored as arrays of arrays. Therefore, the way 2D arrays are created and indexed is similar to 1D array objects.

EXCLUSION STATEMENT—(EK VAR-2.F.1): 2D array objects that are not rectangular are outside the scope of the course and AP Exam.

VAR-2.F.2

For the purposes of the exam, when accessing the element at `arr[first][second]`, the first index is used for rows, the second index is used for columns.



VAR-2.F.3

The initializer list used to create and initialize a 2D array consists of initializer lists that represent 1D arrays.

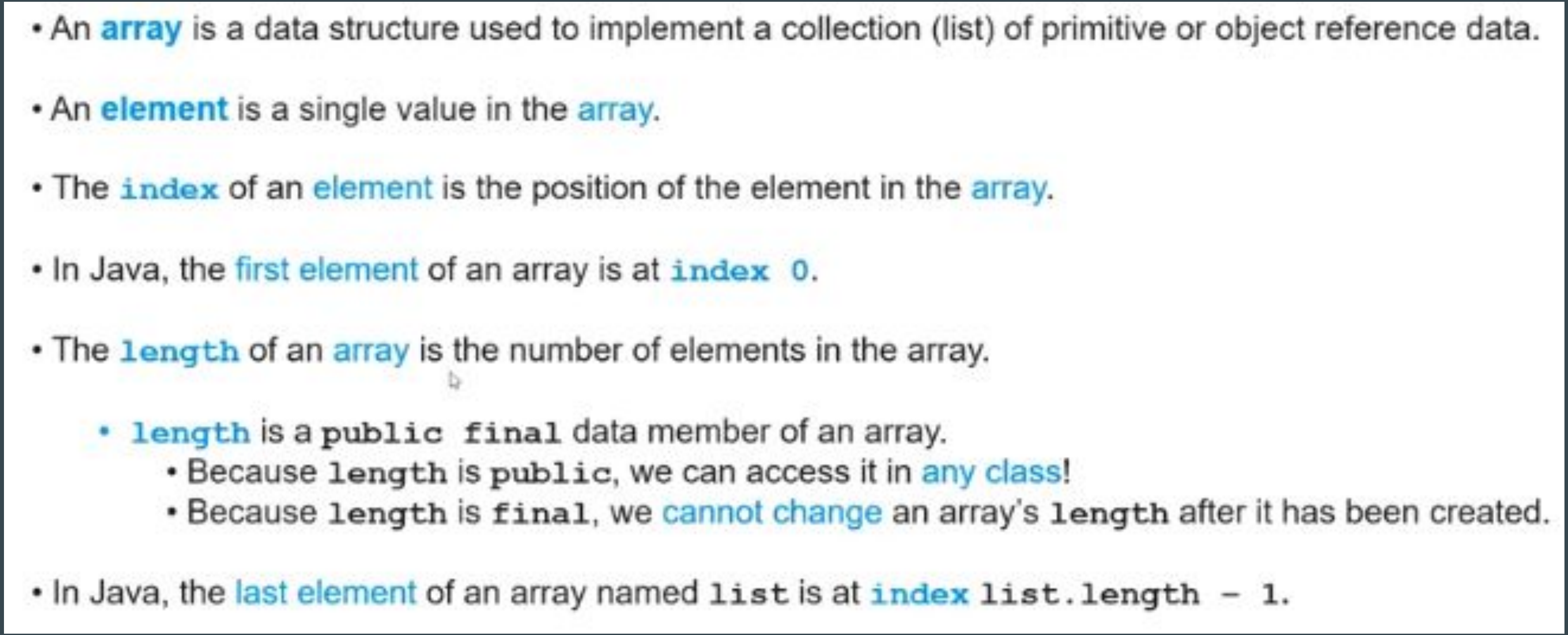
VAR-2.F.4

The square brackets `[row][col]` are used to access and modify an element in a 2D array.

VAR-2.F.5

"Row-major order" refers to an ordering of 2D array elements where traversal occurs across each row, while "column-major order" traversal occurs down each column.

Array Vocabulary REVIEW!

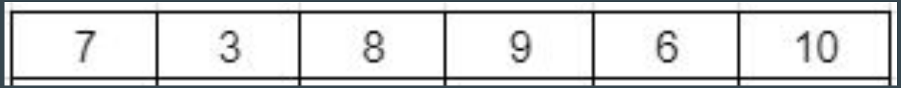


- An **array** is a data structure used to implement a collection (list) of primitive or object reference data.
- An **element** is a single value in the array.
- The **`index`** of an element is the position of the element in the array.
- In Java, the first element of an array is at **`index 0`**.
- The **`length`** of an array is the number of elements in the array.
  - **`length`** is a **`public final`** data member of an array.
    - Because **`length`** is **`public`**, we can access it in any class!
    - Because **`length`** is **`final`**, we cannot change an array's **`length`** after it has been created.
- In Java, the last element of an array named **`list`** is at **`index list.length - 1`**.

Source: AP Classroom 8.1 Video

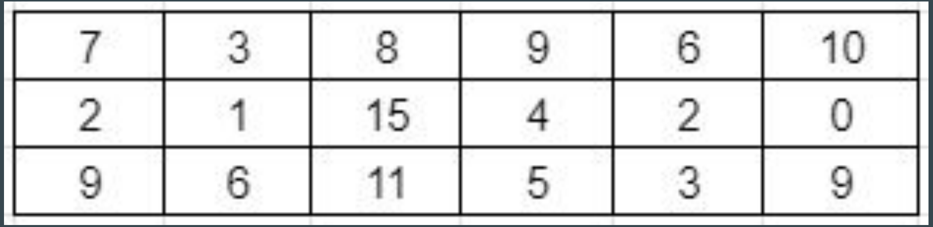
Do Now! 1D vs. 2D Arrays: Visually

- 1D array of random ints:



7	3	8	9	6	10

- 2D array of random ints:



7	3	8	9	6	10
2	1	15	4	2	0
9	6	11	5	3	9

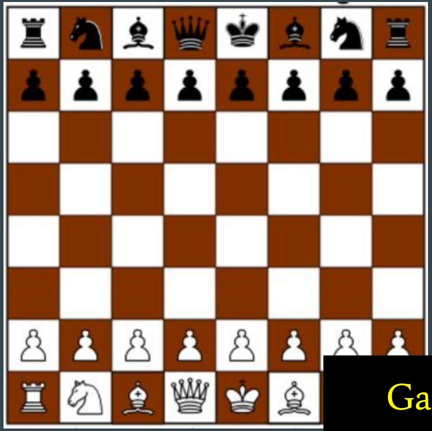
Discuss with your neighbor: **What are some possible applications of 2D arrays?**

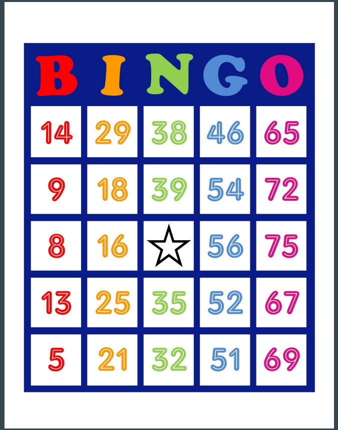
Challenge: What about 3D arrays (lists of lists of lists)?!?! :-O

2D Array Applications!

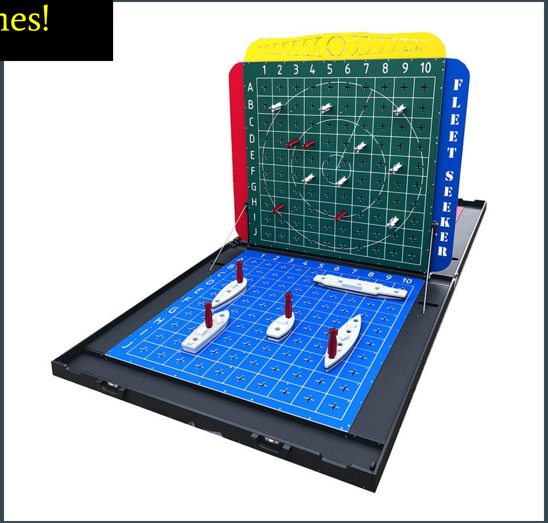
Lockers

Games!





B	I	N	G	O
14	29	38	46	65
9	18	39	54	72
8	16	☆	56	75
13	25	35	52	67
5	21	32	51	69

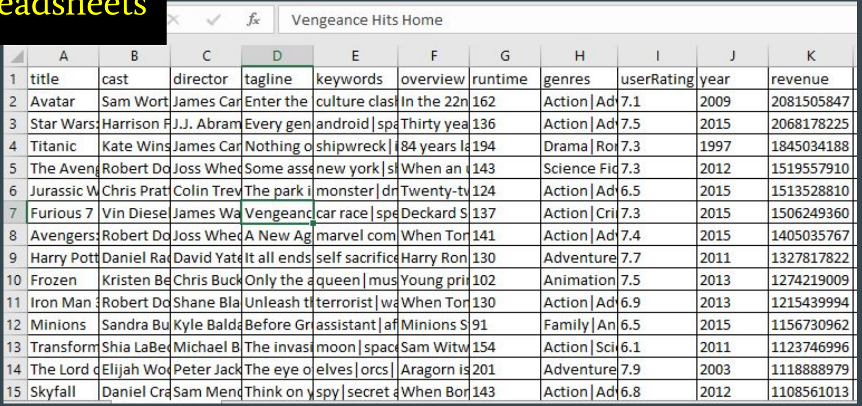


Mail boxes

2D Array Applications!

Theater seating

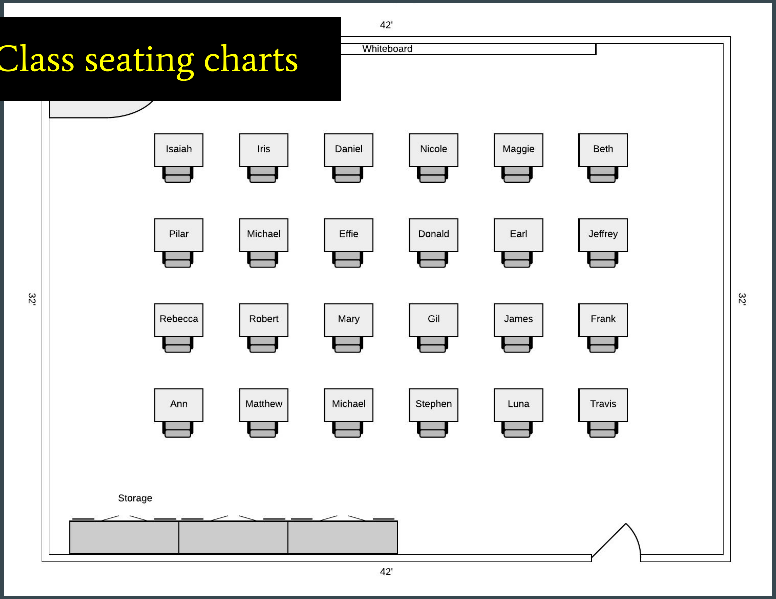
Spreadsheets



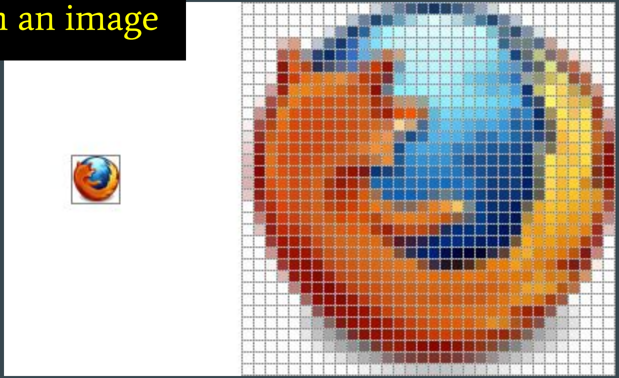
Vengeance Hits Home

	A	B	C	D	E	F	G	H	I	J	K
1	title	cast	director	tagline	keywords	overview	runtime	genres	userRating	year	revenue
2	Avatar	Sam Wort	James Car	Enter the	culture clas	In the 22n	162	Action|Ad	7.1	2009	2081505847
3	Star Wars:	Harrison F	J.J. Abram	Every gen	android|spa	Thirty yea	136	Action|Ad	7.5	2015	2068178225
4	Titanic	Kate Wins	James Car	Nothing o	shipwreck|i	84 years la	194	Drama|Ror	7.3	1997	1845034188
5	The Aveng	Robert Do	Joss Whed	Some asse	new york|sl	When an	143	Science Fic	7.3	2012	1519557910
6	Jurassic W	Chris Pratt	Colin Trev	The park i	monster|dr	Twenty-tw	124	Action|Ad	6.5	2015	1513528810
7	Furious 7	Vin Diesel	James Wa	Vengeanc	car race|spe	Deckard S	137	Action|Cri	7.3	2015	1506249360
8	Avengers:	Robert Do	Joss Whed	A New Ag	marvel com	When Tor	141	Action|Ad	7.4	2015	1405035767
9	Harry Pott	Daniel Rad	David Yate	It all ends	self sacrifice	Harry Ron	130	Adventure	7.7	2011	1327817822
10	Frozen	Kristen Be	Chris Buck	Only the a	queen|mus	Young prin	102	Animation	7.5	2013	1274219009
11	Iron Man 3	Robert Do	Shane Bla	Unleash th	terrorist|wa	When Tor	130	Action|Ad	6.9	2013	1215439994
12	Minions	Sandra Bu	Kyle Balda	Before Gr	assistant|af	Minions S	91	Family|An	6.5	2015	1156730962
13	Transform	Shia LaBe	Michael B	The invasi	moon|spac	Sam Witw	154	Action|Sci	6.1	2011	1123746996
14	The Lord o	Elijah Wo	Peter Jack	The eye o	elves|orcs|	Aragorn is	201	Adventure	7.9	2003	1118888979
15	Skyfall	Daniel Cra	Sam Mend	Think on y	spy|secret a	When Bor	143	Action|Ad	6.8	2012	1108561013

Class seating charts

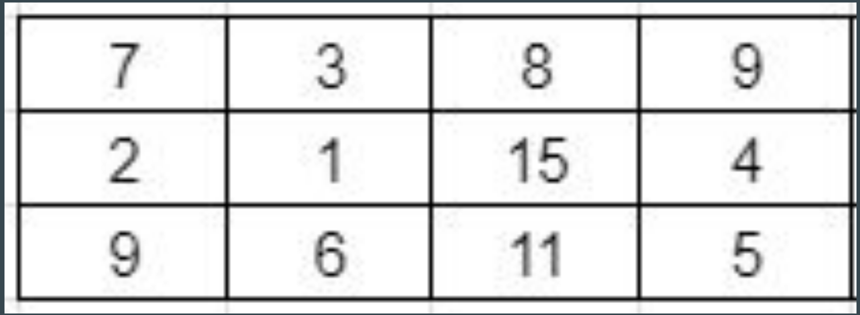


Pixels in an image



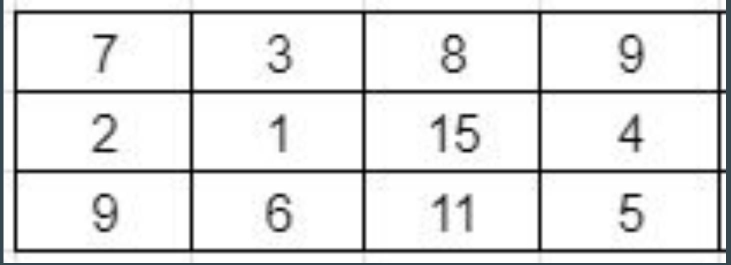
2D Array Creation

- As we learned, a 1D array holds elements that are primitive type or objects, and arrays *themselves* are objects
- A **2D array** is simply a 1D array where each element in the array is *itself* an array; i.e. **a 1D array of 1D arrays!**
- We think of 2D arrays in terms of “rows” and “columns,” just like spreadsheets, and we often *visualize* them as tables:



	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

Interview Time



7	3	8	9
2	1	15	4
9	6	11	5

As a 2D Array:
```
[ [ 7, 3, 8, 9 ],
  [ 2, 1, 15, 4 ],
  [ 9, 6, 11, 5 ] ]
```

As a 1D Array:
```
[ 7, 3, 8, 9, 2, 1, 15, 4, 9, 6, 11, 5 ]

     row1          row2          row3
```

You can represent any 2D array as a 1D array where you determine the rows and columns through your own logic. Why use 2D arrays at all?

2D Array Creation

- Like 1D arrays, 2D arrays hold elements of the same type (primitive or object type)
- Declaring and creating 2D arrays -- like 1D arrays, you must know how many elements you want in your 2D array ahead of time

1. Using the `new` keyword:

```
int[][] nums = new int[3][4];
```

Creates a 2D array with **3 rows** and **4 columns**

```
int[3][4]
```

Rows first, **Columns** second

Remember RC COLA!

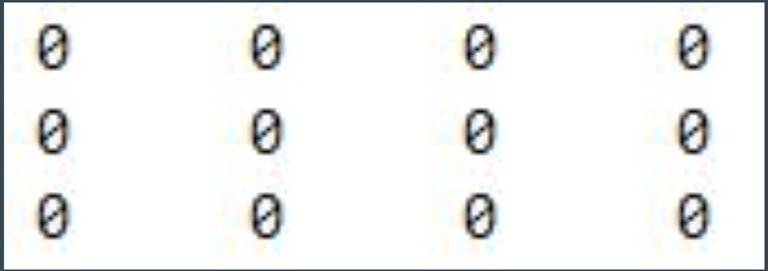
2D Array Creation

Using the `new` keyword:

```
int[][] nums = new int[3][4];
```

Creates a 2D array with **3 rows** and **4 columns**

Like 1D arrays, if you create a 2D array this way -- with *how many elements* there are, but not *what* those elements are -- then the values default to 0 until they get set:

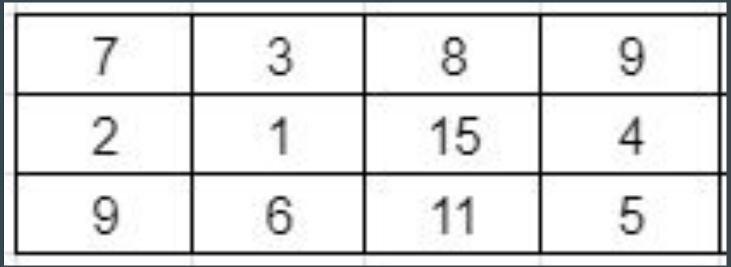


	Column 0	Column 1	Column 2	Column 3
Row 0	0	0	0	0
Row 1	0	0	0	0
Row 2	0	0	0	0

2D Array Creation

2. Like 1D arrays, we can use an “initializer list” to initialize a 2D array if we know what values we want in the 2D array at the time of its creation (it’s actually an “initializer list **of lists**”!)

   Example, let’s say we wanted to initialize a 2D array to contain these `ints`:



	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

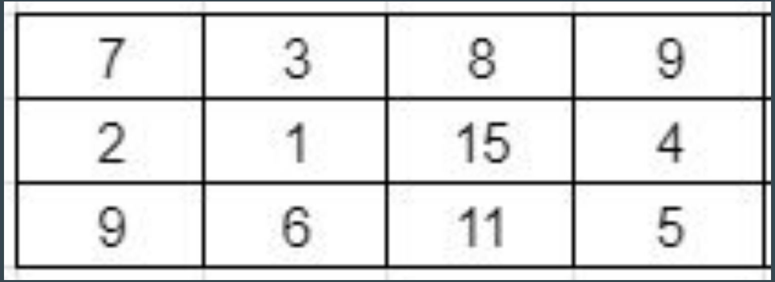
Like 1D arrays, when you do it this way, you don’t need to give the explicit size of each array, since Java determines it automatically

```
int[][] randNums = {{7, 3, 8, 9}, {2, 1, 15, 4}, {9, 6, 11, 5}};
```

Here you can see that each element of the `randNums` “outer” array is an individual array of `ints` (aka, the “inner” arrays), and that each “inner” array is a “**row.**” The individual elements of the *inner* array rows, when considered together, make up the “**columns**”.

2D Array Creation

When we type up a 2D array using initializer lists in our code, it is common practice to "write it the way we see it!":



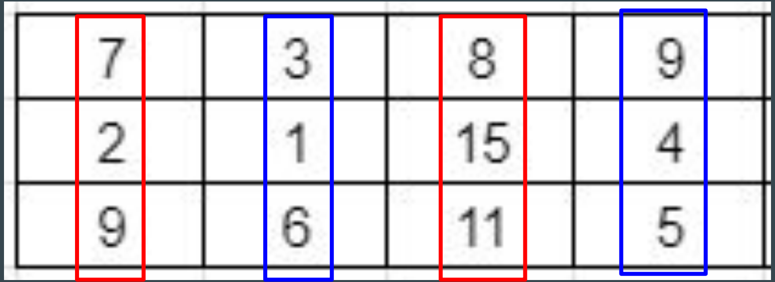
	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[][] randNums = {{7, 3, 8, 9},
                    {2, 1, 15, 4},
                    {9, 6, 11, 5}};
```

This is preferred since it makes clear the "shape" of the data: 3 rows, 4 columns!

2D Array Creation

When we type up a 2D array using initializer lists in our code, it is common practice to "write it the way we see it!":

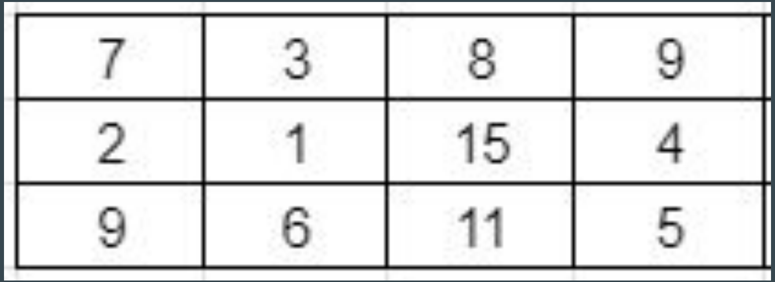


	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[][] randNums = {{7, 3, 8, 9},        Row 0
                    {2, 1, 15,  4},      Row 1
                    {9, 6, 11,  5}};     Row 2
```

This is preferred since it makes clear the "shape" of the data: 3 rows, 4 columns!

2D Array Creation



	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

You can also do it this way:

```
int[] row0nums = {7, 3, 8, 9};
int[] row1nums = {2, 1, 15, 4};
int[] row2nums = {9, 6, 11, 5};

int[][] randNums = {row0nums, row1nums, row2nums};
```

2D Array Creation

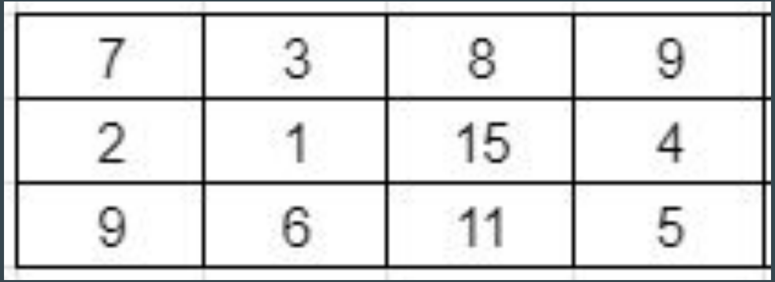
Because each element of the outer array (i.e. each “inner array” / “sub-array”) is a **row**, we call this organizational structure **row-major order**.

Note that *some* programming languages (not Java!) store each *column* of data as the sub-array elements of the outer array; this is called **column-major order**.

Also, we tend to think about 2D arrays like we think about spreadsheet tables: as **rectangular** in shape, where each “row” has the same length, i.e. the same number of elements in each inner array. Rectangular 2D arrays are the most common to use and see, and the only kind you need to know for this course and the AP Exam; however, it *is* possible in Java to have **jagged** 2D arrays where inner arrays have *different* lengths! (curious how that works? go look it up 😎)

2D Array Size? Length? What?

Does a 2D array have a `size()`? A `length`? Some other type of dimensions?



	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[][] randNums = {{7, 3, 8, 9},
                    {2, 1, 15, 4},
                    {9, 6, 11, 5}};
```

Remember: A 2D array is simply a 1D array that has 1D array as its elements!

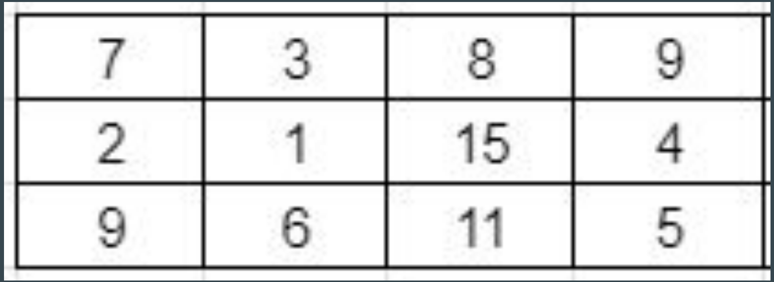
2D Array Size? Length? What?

	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[][] randNums = {{7, 3, 8, 9},
                    {2, 1, 15, 4},
                    {9, 6, 11, 5}};


System.out.println(randNums.length);
// prints 3 -- the number of inner array elements (ROWS)
```

`randNums` has a **`length`**, just like all other arrays -- *not* `size()`, which is for `ArrayList` -- and just like any other array, it's `length` is how many elements it contains! `randNums` contains **3** elements (which just happen to be *other* arrays)

How many elements does each row have (i.e. how many columns)?



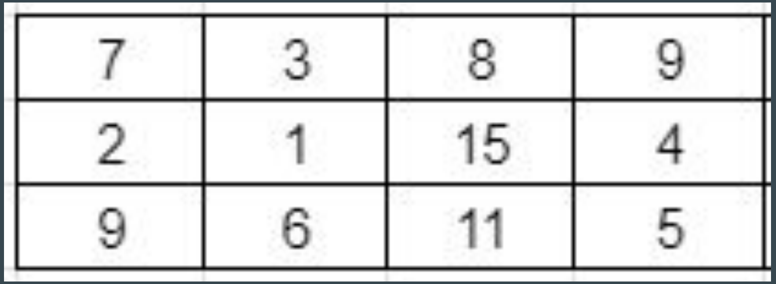
	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[][] randNums = {{7, 3, 8, 9},
                    {2, 1, 15, 4},
                    {9, 6, 11, 5}};



System.out.println(randNums[0].length);
```

We can check the `length` of *one of the inner arrays* to get the "column count" -- we could use any since we know it's rectangular and all have the same length, but by convention we use the length of "row 0" to get column count

```
// prints 4 -- the number of elements in an inner array (COLUMNS)
```

“Dimensions”



	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[][] randNums = {{7, 3, 8, 9},
                    {2, 1, 15, 4},
                    {9, 6, 11, 5}};
```

We sometimes refer to a 2D array’s “size” by its “dimensions.”

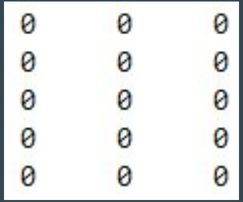
The 2D array above is a “**3 by 4**” array, or “**3 x 4**” (**Row by Column**!)

Quick Check!

```
int[][] ticketInfo = new int[5][3];
```

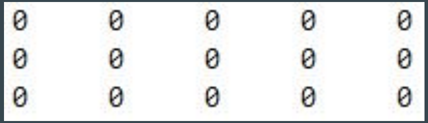
What will this 2D array “look like” if we printed it using rows and columns?

A.



```
0       0       0
0       0       0
0       0       0
0       0       0
0       0       0
```

B.



```
0       0       0       0       0
0       0       0       0       0
0       0       0       0       0
```

How many *total* elements in the entire 2D array?

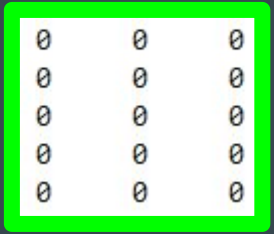
What is `ticketInfo.length`?

What is `ticketInfo[0].length`?

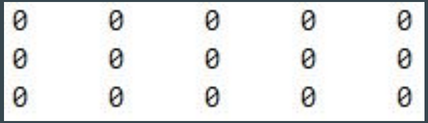
Quick Check!

```
int[][] ticketInfo = new int[5][3];
```

What will this 2D array “look like” if we printed it using rows and columns?

A. 

```
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
```

B. 

```
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
```

[5][3] → 5 Rows, 3 Columns → RC cola!

How many total elements? 5 x 3 = 15

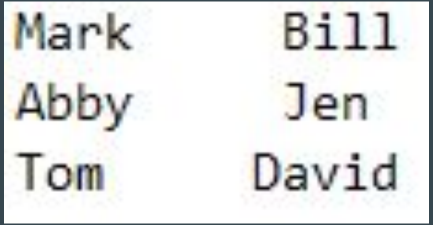
What is `ticketInfo.length`? 5 (has 5 inner arrays, or **rows**)

What is `ticketInfo[0].length`? 3 (each inner array has 5 elements, or **columns**)

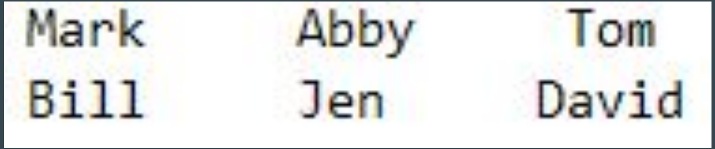
Quick Check!

```
String[][] names =

{{"Mark", "Abby", "Tom"}, {"Bill", "Jen", "David"}};
```

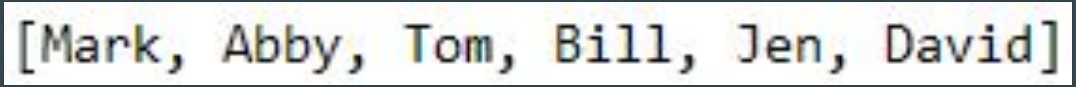
What will this 2D array "look like" if we printed it using rows and columns?

A. 

```
Mark        Bill
Abby        Jen
Tom         David
```

B. 

```
Mark        Abby        Tom
Bill        Jen         David
```

C. 

```
[Mark, Abby, Tom, Bill, Jen, David]
```

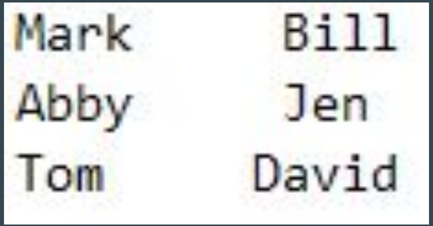
Quick Check!

```
String[][] names =
{{"Mark", "Abby", "Tom"}, {"Bill", "Jen", "David"}};
```

Row 0 Row 1

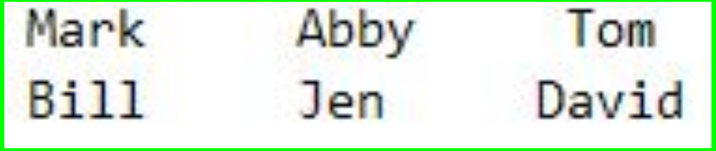
What will this 2D array "look like" if we printed it using rows and columns?

A.



```
Mark        Bill
Abby        Jen
Tom         David
```

[✓ B.]



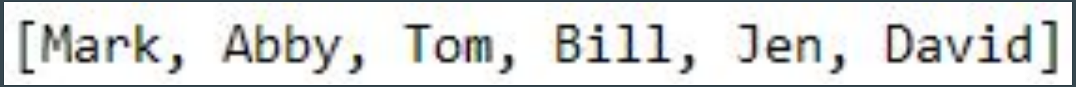
```
Mark        Abby        Tom
Bill        Jen         David
```

Row 0

Row 1

Java is a **row-major** language, so each inner array of the outer `names` array represents a ROW

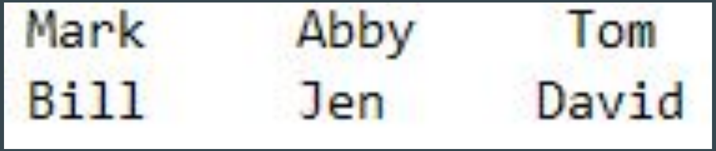
C.



```
[Mark, Abby, Tom, Bill, Jen, David]
```

Accessing/assigning/updating values

How could we have created this same 2D array using the new keyword?



	Column 0	Column 1	Column 2
Row 0	Mark	Abby	Tom
Row 1	Bill	Jen	David

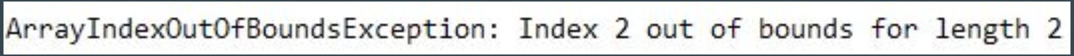
```
String[][] names = new String[2][3]; // 2 rows, 3 columns
names[0][0] = "Mark";   // row 0, column 0
names[0][1] = "Abby";   // row 0, column 1
names[0][2] = "Tom";    // row 0, column 2
names[1][0] = "Bill";   // row 1, column 0
names[1][1] = "Jen";    // row 1, column 1
names[1][2] = "David"; // row 1, column 2
```

[row] [column]

Don’t go out of bounds!

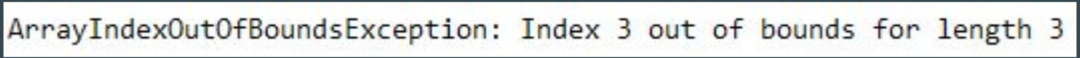
Be careful not to go out of bounds!

```
String[][] names = new String[2][3]; // 2 rows, 3 columns
names[0][0] = "Mark";   // row 0, column 0
names[0][1] = "Abby";   // row 0, column 1
names[0][2] = "Tom";    // row 0, column 2
names[1][0] = "Bill";   // row 1, column 0
names[1][1] = "Jen";    // row 1, column 1
names[1][2] = "David";  // row 1, column 2
names[2][0] = "Joe";    // row 2, column 0
```



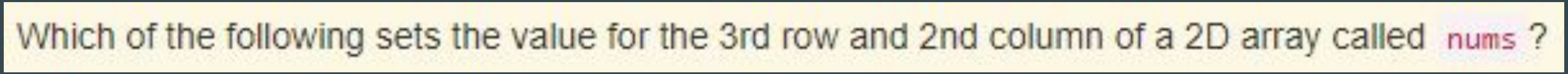
```
ArrayIndexOutOfBoundsException: Index 2 out of bounds for length 2
```

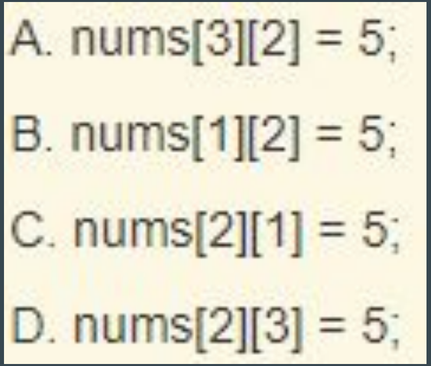
```
names[1][3] = "Jill";   // row 1, column 3
```



```
ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
```

Quick Check!


Which of the following sets the value for the 3rd row and 2nd column of a 2D array called `nums` ?

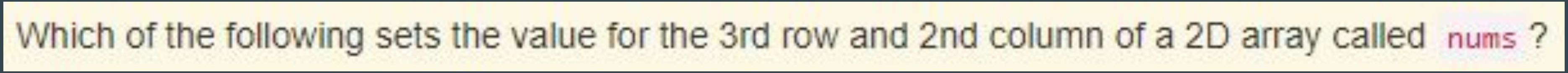

A. nums[3][2] = 5;

B. nums[1][2] = 5;

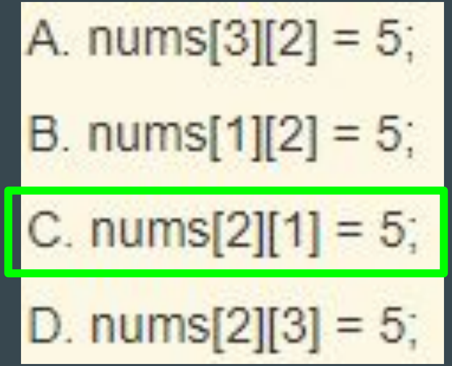
C. nums[2][1] = 5;

D. nums[2][3] = 5;

Quick Check!



Which of the following sets the value for the 3rd row and 2nd column of a 2D array called `nums` ?



A. nums[3][2] = 5;

B. nums[1][2] = 5;

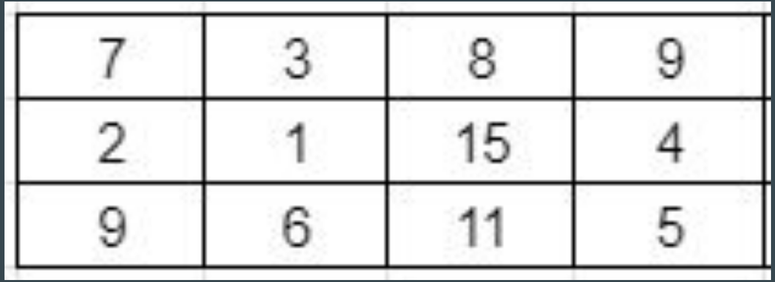
[✓ C. nums[2][1] = 5;]

D. nums[2][3] = 5;

`[row][column]`

Row first, column second (RC cola!)
The 3rd row would be index 2, and the 2nd column would be index 1 (since arrays are 0 indexed)

2D Array Creation



	Column 0	Column 1	Column 2	Column 3
Row 0	7	3	8	9
Row 1	2	1	15	4
Row 2	9	6	11	5

```
int[] row0nums = {7, 3, 8, 9};
int[] row1nums = {2, 1, 15, 4};
int[] row2nums = {9, 6, 11, 5};
```

You can assign entire rows using just the first index:

```
int[][] randNums = new int[3][4];
randNums[0] = row0nums;
randNums[1] = row1nums;
randNums[2] = row2nums;
randNums[2] = 5; // invalid! expecting an int[], not int
```

2D Array Declaration

- 2D arrays are simply 1D array objects that contain 1D array objects as elements
- So if you just declare a 2D array without initializing it immediately, the default value is `null` (like all objects):

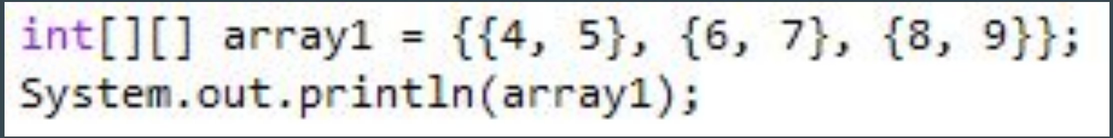
```
    int[][] scores;
```

(`scores` has the value `null` until you initialize it -- it does *NOT* default to something like [0][0])

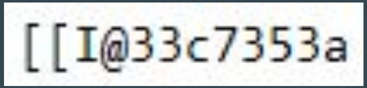
Printing 2D Arrays

So wait, how do you print a 2D array to “show” its rows and columns?

Just like 1D arrays, you can’t simply `println` a 2D array object:



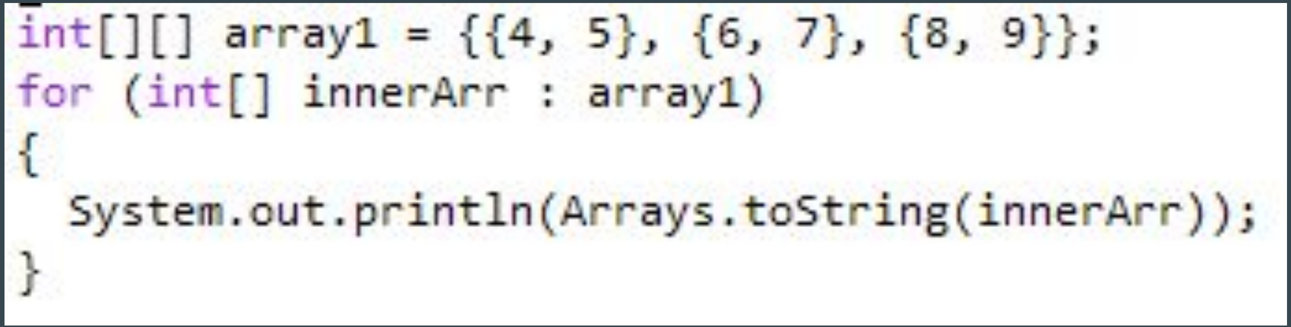
```
int[][] array1 = {{4, 5}, {6, 7}, {8, 9}};
System.out.println(array1);
```



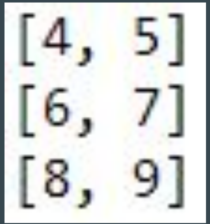
```
[[I@33c7353a
```

Printing 2D Arrays

Aahhh! Let's iterate over the outer array's elements (each a 1D array of ints, or type `int[]`) and use the `Arrays.toString()` method on each one!



```java
int[][] array1 = {{4, 5}, {6, 7}, {8, 9}};
for (int[] innerArr : array1)
{
  System.out.println(Arrays.toString(innerArr));
}
```



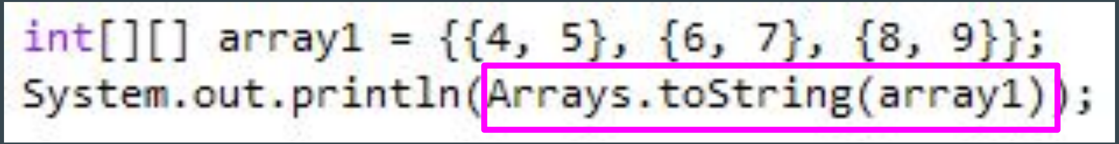
```
[4, 5]
[6, 7]
[8, 9]
```

😊

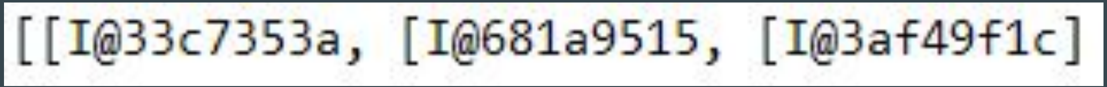
We could *also* use nested for loops to fully traverse every element -- full 2D array traversal is the topic of our next class!

Printing 2D Arrays

Oooooh! How about the `Arrays.toString()` method if we import `java.util.Arrays`? That worked for 1D arrays!



```
int[][] array1 = {{4, 5}, {6, 7}, {8, 9}};
System.out.println(Arrays.toString(array1));
```



```
[[I@33c7353a, [I@681a9515, [I@3af49f1c]
```

😫 😭

Although you can discern that the outer array contains 3 inner arrays, i.e. 3 rows!

Don’t forget: 2D arrays *are* arrays -- so they are objects!

```
int[][] array1 = {{4, 5}, {6, 7}, {8, 9}}; // 3 rows, 2 col

int[][] array2 = array1; // setting array2 to reference the
                            same array1 object -- both
                            references point to the same
                            2D array object in memory

array2[1][0] = 3;          // sets the 6 to a 3

System.out.println(array1[1][0]);   // prints 3!!!
```

Just another type!

```
public class Game
{
    private String[] names;
    private int[][] gameBoard;

    public Game(String[] playerNames, int boardSize)
    {
        names = playernames;
        gameBoard = new int[boardSize][boardSize];
    }
}
```