2-D Arrays

In Java – ICS3U

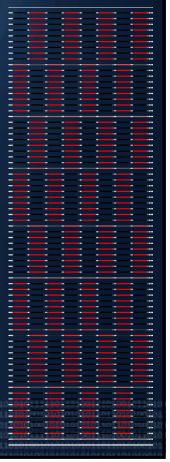
2-D Arrays

- A multi-dimensional array is often referred as
  - An array of arrays
  - Table
  - matrix

- A multi-dimensional array can be two-dimensional, three dimensional or higher.
- We will just focus on 2 D arrays.

2-D Arrays

- A two-dimensional array is basically a grid or matrix of items.
- It consists of a number of ROWS and COLUMNS.
- We could define the layout of a checkboard as a two-dimensional array.
- It would be a two-dimensional array declared to be 8 by 8.
- *don't be confused.



Remember

- ROW-COLUMN
- ROW-COLUMN
- ROW-COLUMN
- ROW-COLUMN
- ROW-COLUMN
- ROW-COLUMN
- ROW-COLUMN

2-D Arrays Declaration

- Declaration

```
int [ ] [ ] A;
```

- Initialization

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

- Declaration and Initialization Together

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

- These statements would create an array with 3 rows and 4 items in each row.

Filling an Array During Initialization

```
int [ ] [ ] A = {
                {6, 7, 4, 3},
                {0, 4, 7, 2},
                {4, 2, 1, 6}
              };
```

- If no initializer is provided for an array, then the array is created and automatically filled with the appropriate values depending on the base type of the array.
- For numbers the default is zero, false for Boolean, and null for objects.

Parts of the 2 D Array

```
int [ ] [ ] A = {
                {6, 7, 4, 3},
                {0, 4, 7, 2},
                {4, 2, 1, 6}
              };
```

- Name of Array: A
- Type of Array: Int
- Number of Rows: 3 (grid location 1 less)
- Number of Columns: 4 (grid location 1 less)
- Total Storage Locations: 12

6	7	4	3
0	4	7	2
4	2	1	6

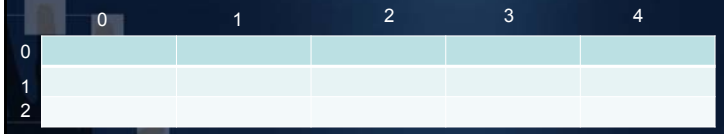
Access Elements

- Notice how the coordinates are 1 less then declared
- System.out.println(A[2][3]);

	0	1	2	3
0	6	7	4	3
1	0	4	7	2
2	4	2	1	6

Putting Values Individually

- int [ ] [ ] nums = new int [3] [5];
- nums[1][1]= 7
- nums[2][1]= 3
- nums[0][3]= 5
- nums[0][4]= 1
- nums[0][0]= nums[1][1]
- nums[2][4]= nums[0][3] * 2



- What's in the spaces we didn't fill in??
- nums[0][1]= num[1][1]*num[1][4] The answer is: 0 – why?

Processing Two-Dimensional Arrays

- Arrays are usually processed using the for statement.
  - To process all the items in a two-dimensional array, you will have to use a nested for statements (one for statement, inside another).
  - One for loop will change the row value and the other for loop will change the column value.

Example

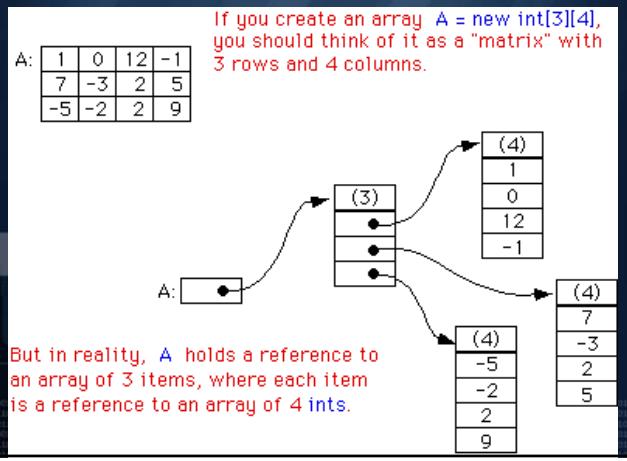
- To set all the values in the array equal to 0 you could use the following:

```
for (int row=0; row <3; row ++)    {
      for (int column=0; column < 4; column ++)  {
            A[row] [column]=0;
       }//end of 2nd for
} //end of 1st for.
```

Adding Up All Items in the Array

```
int sum = 0;
for (int row=0; row <3; row ++)
{
      for (int column=0; column < 4; column ++)
      {
            sum = sum + A[row] [column];
                  }
}
```

Advanced Concept – Pointers!



Caveats

- Don't let this confuse you – its okay for now to think of a array as a grid.
- But its not a rectangular grid. Or maybe think of it as its not exactly 2D in Java.
- This has a serious impact on length calls, and the fact that you can initialize arrays that are not perfect columns
  - as they are really a 1D array – being stored in the sequential array index.

Array.length

- Because of this pointer concept, make sure you understand what length your requesting!
- Lets examine this code.

```
public static void main(String[] args) {

int[][] myArray= new int[][] {
new int[] { 1, 2, 3 },
new int[] { 1, 2, 3, 4},
};
System.out.println(myArray.length); //2
System.out.println(myArray[0].length); //3
System.out.println(myArray[1].length); //4
}
```