Two-Dimensional Arrays

In JAVA an array can be declared and initialized using the shorthand assignment statement:

```
String[] people={"Aaron","Brandy","Ken","Greg","David","Jason"};
```

Creating a data structure like the one pictured on the right:

String[] people

index	Value
0	Aaron
1	Brandy
2	Ken
3	Greg
4	David
5	Jason

This makes it convenient to manage the list using structures like the "for loop" to iterate over the list.

And it would be trivial to create a second list, lets say, of family:

```
String[] family={"Steve","Carol","Kim","Nicole"};
```

But it is also possible to make a single array, which is itself an array of arrays, so that both our lists can be contained in one data structure:

```
String[][] people={
    {"Aaron","Brandy","Ken","Greg","David","Jason"},
    {"Steve","Carol","Kim","Nicole"}
};
```

Notice that the `String` array now has two sets of [] brackets, indicating that there are two-dimensions to this array!

Then, in the assignment portion of the statement there are actually two pairs of data sets inside curly braces {} separated by a comma, that sit within the outer curly braces {}.

This has the effect of creating two arrays within another array, which yields the data structure depicted to the right:

String[][] people

index	Value	
	index	Value
0	0	Aaron
	1	Brandy
	2	Ken
	3	Greg
	4	David
	5	Jason
	index	Value
1	0	Steve
	1	Carol
	2	Kim
	3	Nicole

String[][] people

Index	0	1	2	3	4	5
0	Aaron	Brandy	Ken	Greg	David	Jason
1	Steve	Carol	Kim	Nicole		

This may seem confusing, but another way to look at the same structure is pictured on the left.

The important difference between the two depictions is that the first is more technically accurate because each of the two lists inside the outer-list has a different number of elements in it, while the depiction on the above implies that the second list has the same number of elements as the first, which is not true.

Note: On the AP Test there will only be two-dimensional arrays with the same number of elements in each list. Therefore, you can use either visual representation to help you understand two-dimensional arrays!

Generally, two-dimensional arrays are made up of arrays that are of the same size.

For the array, `people`, the standard methods apply. For example the length of the people array is 2 because it has two elements, each of which is an array. Therefore `people[0]` is a `String` array with 6 elements and `people[1]` is a `String` array with 4 elements. So the statement:

```
System.out.println(people.length);
```

Generates the console output 2 because it contains two arrays!

Keep in mind that what we really have is an array of two arrays, so the length of the `people` array is the number of arrays it contains!

To find out how many elements are in the first array we have to specify which array within `people` we want so the statement:

`System.out.println(people[0].length);` outputs 6 to the console, and:

`System.out.println(people[1].length);` outputs 4 to the console.

What is the benefit of a two-dimensional array?

Just as we put elements into an array so that we can process them efficiently using loops, we can make use of loops to process each of the arrays that makes up `people`.

The code below sets up a nested loop to iterate over **all** the values `people`:

```java
public class Illustration {
 public static void main(String[] args) {
   String[][] people={
       {"Aaron","Brandy","Ken","Greg","David","Jason"},
       {"Steve","Carol","Kim","Nicole"}
   };
   for (int arrayIndex=0; arrayIndex<people.length; arrayIndex++) {
     for (int i=0; i<people[arrayIndex].length; i++) {
       System.out.print("people["+arrayIndex+"]["+i+"]=");
       System.out.println(people[arrayIndex][i]);
     }
   }
 }
}
```

Console Output:

```
people[0][0]=Aaron
people[0][1]=Brandy
people[0][2]=Ken
people[0][3]=Greg
people[0][4]=David
people[0][5]=Jason
people[1][0]=Steve
people[1][1]=Carol
people[1][2]=Kim
people[1][3]=Nicole
```

And with just a bit more tweaking the further advantages to such a data structure can be seen.

A new array is created that will be titles for each of the arrays that make up the `people` array. This array is **not** part of the people array, but works with it:

(New code is highlighted in yellow)

```java
public class Illustration {
 public static void main(String[] args) {
   String[][] people={
       {"Aaron","Brandy","Ken","Greg","David","Jason"},
       {"Steve","Carol","Kim","Nicole"}
   };
   String[] titles={"Friends","Family"};
   for (int arrayIndex=0; arrayIndex<people.length; arrayIndex++) {
     System.out.println(titles[arrayIndex]+":");
     for (int i=0; i<people[arrayIndex].length; i++) {
       System.out.println("["+i+"]="+people[arrayIndex][i]);
     }
   }
 }
}
```

Console Output:

```
Friends:
[0]=Aaron
[1]=Brandy
[2]=Ken
[3]=Greg
[4]=David
[5]=Jason
Family:
[0]=Steve
[1]=Carol
[2]=Kim
[3]=Nicole
```

And the real flexibility can be seen when it comes time to add a third category!

ASSIGNMENT PART A:

Modify the code above so that it includes a third and fourth category with at least three entries in each category.

PART II: FILLING A TWO-DIMENSIONAL ARRAY

The other method for declaring an array uses the keyword `new` with the assignment operator:
`String[][] rectArray=new String[3][5];`

The result of this statement is the creation of the variable `rectArray`, which is a rectangular array with 3 rows and 5 columns. Each element of the array is `null` because it is a `String` array.

In order to fill the array, first a “for loop” will iterate over each row using `rectArray.length` to know how many rows there are.

A nested “for loop” will iterate over each column in that row using `rectArray.length[0]` to know how many columns there are.

The program below creates a two-dimensional `String` array using `new`, then iterates over every element of the array, filling each element with a String that indicates the row and column of the element. Finally, the contents of the array are output to the console with labels for row and column:

```
01 public class Array2DNew {
02   public static void main(String[] args) {
03     String[][] rectArray=new String[3][5];
04     for(int row=0; row<rectArray.length; row++) {
05       for(int col=0; col<rectArray[0].length; col++) {
06         rectArray[row][col]="("+row+","+col+")";
07       }
08     }
09     System.out.print("\t");
10     for (int col=0; col<rectArray[0].length;col++) {
11       System.out.print("Col:"+col+"\t");
12     }
13     System.out.println();
14     for(int row=0; row<rectArray.length; row++) {
15       System.out.print("Row:"+row+"\t");
16       for(int col=0; col<rectArray[0].length; col++) {
17         System.out.print(rectArray[row][col]+"\t");
18       }
19       System.out.println();
20     }
21   }
22 }
```

Console Output:

```
        Col:0  Col:1  Col:2  Col:3  Col:4
Row:0   (0,0)  (0,1)  (0,2)  (0,3)  (0,4)
Row:1   (1,0)  (1,1)  (1,2)  (1,3)  (1,4)
Row:2   (2,0)  (2,1)  (2,2)  (2,3)  (2,4)
```

Step I: Initializing the Array

- **`Line 03`**: Declares the `rectArray` with the type `String[][]` using by assigning it the result of `new String[3][5]`. For now this array could be pictured:

`String[][] rectArray`

Index	0	1	2	3	4
0	null	null	null	null	null
1	null	null	null	null	null
2	null	null	null	null	null

Note: For AP Testing, two-dimensional arrays will always be rectangular with the number of rows first, accessed using `array.length`, and the columns second using `array[0].length`!
This is known as “ROW MAJOR” order.

- **`Line 04`**: Notice that the diagram above is a matrix. In order to process every element in the matrix, a variable must be created to iterate over the rows. The outer loop creates the `row` variable that iterates over all the rows that make up the matrix using `rectArray.length` to determine how many rows are in the matrix.
- **`Line 05`**: A nested loop is used to iterate over the columns using the `col` variable. The number of columns is determined using `rectArray[0].length`.
- **`Line 06`**: This line does all the work. It assigns a String value to an element of `rectArray` using `row` and `col` as the indexes. The value that gets assigned is a concatenated String made up of a left parenthesis, the row, a comma, the column, and a closing parenthesis: (row,column).

After Lines **`03-08`**: The array looks like this:

`String[][] rectArray`

Index	0	1	2	3	4
0	(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
1	(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
2	(2,0)	(2,1)	(2,2)	(2,3)	(2,4)

```
09    System.out.print("\t");
10    for (int col=0; col<rectArray[0].length;col++) {
11      System.out.print("Col:"+col+"\t");
12    }
13    System.out.println();
14    for(int row=0; row<rectArray.length; row++) {
15      System.out.print("Row:"+row+"\t");
16      for(int col=0; col<rectArray[0].length; col++) {
17        System.out.print(rectArray[row][col]+"\t");
18      }
19      System.out.println();
20    }
```

- `Lines 09-13`: Generate the column headings:
  - `Line 09`: Outputs a tab character to the console.
  - `Line 10`: Creates the `col` variable to iterate once for each column in `rectArray`.
  - `Line 11`: Outputs the letters "col:" followed by the value of `col` and then a tab character.
  - `Line 13`: Outputs a carriage-return to end the line.

Console Output from Lines `09-13`:

```
        Col:0  Col:1  Col:2  Col:3  Col:4
```

- `Lines 14-20`: These nested for loops output the row labels and the contents of the row from `rectArray` to the console:
  - `Line 14`: For loop iterates from 0 to 2 because `rectArray.length` has the value 3.
  - `Line 15`: Outputs the letters "row:" followed by the value of `row` and then a tab character.
  - `Line 16`: A nested loop is used to iterate up to `rectArray[0].length`.

Console Output from Lines `14-20`:

```
Row:0  (0,0)  (0,1)  (0,2)  (0,3)  (0,4)
Row:1  (1,0)  (1,1)  (1,2)  (1,3)  (1,4)
Row:2  (2,0)  (2,1)  (2,2)  (2,3)  (2,4)
```

  - `Line 17`: Outputs the value of `rectArray` using `row` and `col` as indexes.
  - `Line 19`: Outputs a carriage-return to end the line.

Assignment Part B:

Write the methods so the `main()` method below generates the console output like that shown below. Your methods should work with the method calls as written:

```
public class SumTables {
    public static void main(String[] args) {
        int rows=5, cols=5;
        int min=1, max=10;
        int[][] table1=fillTable(rows, cols, min, max);
        int[][] table2=fillTable(rows, cols, min, max);
        int[][] sumTable=sumTables(table1, table2);
        outputTable("Table 1", table1);
        outputTable("Table 2", table2);
        outputTable("Sum of Table 1 and Table 2", sumTable);
    }
}
```

```
Table 1
        1       2       3       4       5
1       6       4       5       5       8
2       5       3       5       5       2
3       7       7       8       6       1
4       10      9       3       2       2
5       4       1       3       6       1
Table 2
        1       2       3       4       5
1       8       9       6       6       7
2       7       10      6       9       9
3       10      6       4       7       5
4       10      3       1       10      1
5       4       5       7       4       4
Sum of Table 1 and Table 2
        1       2       3       4       5
1       14      13      11      11      15
2       12      13      11      14      11
3       17      13      12      13      6
4       20      12      4       12      3
5       8       6       10      10      5
```

The `fillTable()` method has four parameters, the first two are integers specifying the number of rows and columns in the array and the second two parameters specify the minimum and maximum values used to generate random values for each element of the int array returned by the method.

The `sumTables()` methods returns a two-dimensional int array where each element is the sum of the same row and column from the two parameter arrays.

Finally, the `outputTable()` method prints a table title followed by the values of the parameter `int` array with rows and columns numbered.