Lesson 22: Two-Dimensional Arrays

Consider the following array (3 rows, 2 columns) of numbers:

```
22   23
24   25
26   27
```

Let's declare our array as follows:

```
int a[][] = new int[3][2];
or
int[][] a = new int[3][2];
```

Subscript convention:

Notice that in both mathematics **and** computer science, designations for a two dimensional array (subscripted variable) conventionally have **rows first** and **columns second**. Just think of RC Cola, …RC (rows, columns).

Initializing a two-dimensional array:

Now let's initialize the *a* array, i.e. store values in the various positions. There are **three** ways to do this:

The first way:

```
int a[ ] [ ] = new int [3] [2]; //declaration
a[0] [0] = 22; //initialization from here on down
a[0] [1] = 23;
a[1] [0] = 24;
a[1] [1] = 25;
a[2] [0] = 26;
a[2] [1] = 27;
```

The second way:

```
int a[][] = {{22, 23},
             {24, 25},
             {26, 27} };
//Notice that declaration and initialization
// must both take place on the same line.
```

The third way:

```
int a[][]=new int[][]{{22, 23},
                      {24, 25},
                      {26, 27} };
```

How many rows and columns?

Determine the number of rows and columns in a two-dimensional array (sometimes

called a **matrix** or **subscripted variables**) as follows:

For the matrix above, *a.length* returns a value of 3…the numbers of rows.

For the matrix above,

a[0].length returns a value of 2…the number of columns in row 0
a[1].length returns a value of 2…the number of columns in row 1
a[2].length returns a value of 2…the number of columns in row 2

"Ragged" arrays:

The previous discussion seems redundant, since **all** rows have 2 columns. So we begin to wonder if it's possible for various rows to have **different** number of columns? Is it really possible to produce a "ragged" looking array with uneven rows? The answer is, "Yes," even though it's highly unusual and seldom used.

Suppose we want the following matrix structure:

```
X     X     X     X
X     X
X     X     X
```

Here's the code that would declare such an array:

```
int a[ ] [ ] = new int[3] [ ]; //array has 3 rows, unspecified number of columns
a[0] = new int[4]; //row 0 has 4 columns
a[1] = new int[2]; // row 1 has 2 columns
a[2] = new int[3]; // row 2 has 3 columns
```

Incidentally, the first line of code above (int a[ ] [ ] = new int[3] [ ];) could be equivalently replaced with the following; however, the former is preferred:

```
int a[ ] [ ] = new int[3] [0]; //3 rows, unspecified number of columns
```

While on the subject of working with a single row of a two-dimensional array, consider an array *a* of three rows declared as was done above. How would we pass a single row of this array to a method called *myMethod* in which each element would be initialized?

```
a[2] = new int[3]; //Row 2 has 3 columns. Before passing a[2] below, we must
//have specified the number of columns.
myMethod(a[2]); // Call the method and pass the row with index 2.
. . .
public void myMethod(int [] x) //Notice how we receive the row
{
        x[0] = 36; // Initialize the three columns.
        x[1] = 101;
        x[2] = -45;
}
```

Automatic initialization of arrays:

As with all one-dimensional numeric arrays, all elements of two-dimensional arrays are also automatically initialized to 0 until specific values are given.

```
int abc[][] = new int[20][30];
System.out.println(abc[5][22]); //0
```

Using the *Arrays* class:

In Lesson 19 (page 3), several methods of the *Arrays* class were discussed. Below we present their equivalents as used with two-dimensional arrays.

```
int a[][] = { {3, 9, 2, 1},
              {5, 7, 6, 0} };
int b[][] = { {0, 2, 8, 4},
              {3, 9, 2, 1} };

System.out.println(Arrays.equals(a, b)); //always false…won't compare entire two-
//dimensional arrays.
System.out.println(Arrays.equals(a[0],b[1])); //true, compares row 0 of a to row 1 of b.

Arrays.sort(a); //illegal (run-time exception), can't sort entire two-dimensional array.
Arrays.sort(a[0]); // sorts the 0 row of the a matrix. Row 0 is now {1, 2, 3, 9}

System.out.println(Arrays.binarySearch(a[0], 9)); //3, returns index of the 9 in row 0.
                                                  //This row must have been sorted first.
Arrays.fill(a, 22); //illegal, can't fill entire two-dimensional array
Arrays.fill(a[1], 22); //fills this row with 22 in each position
```