

Topic Notes: Two-Dimensional Arrays

Two-Dimensional Arrays

We can create arrays to hold objects of any type, either basic data types like `int` and `double`, or instances of objects.

Nothing stops us from defining arrays of arrays. To declare an array, each of whose elements is an array of `int`:

```
int[][] twoDArray;
```

While it is normally written without parentheses, we can think of the above declaration as defining `twoDArray` as having type `(int []) []`. Thus each element of `twoDArray` is an array of `ints`.

Despite the fact that Java will treat this as an array of arrays, we usually think about this as a two-dimensional array, with the elements arranged in a two-dimensional table so that `twoDArray[i][j]` can be seen as the element in the i th row and j th column. For example here is the layout for a two-dimensional array `a` with 6 rows (numbered 0 to 5) and 4 columns:

	0	1	2	3
0	<code>a[0][0]</code>	<code>a[0][1]</code>	<code>a[0][2]</code>	<code>a[0][3]</code>
1	<code>a[1][0]</code>	<code>a[1][1]</code>	<code>a[1][2]</code>	<code>a[1][3]</code>
2	<code>a[2][0]</code>	<code>a[2][1]</code>	<code>a[2][2]</code>	<code>a[2][3]</code>
3	<code>a[3][0]</code>	<code>a[3][1]</code>	<code>a[3][2]</code>	<code>a[3][3]</code>
4	<code>a[4][0]</code>	<code>a[4][1]</code>	<code>a[4][2]</code>	<code>a[4][3]</code>
5	<code>a[5][0]</code>	<code>a[5][1]</code>	<code>a[5][2]</code>	<code>a[5][3]</code>

Viewed in this way, our two-dimensional array is a grid, much like a map or a spreadsheet. This is a natural way to store things like tables of data or matrices.

We access elements of two-dimensional arrays in a manner similar to that used for one dimensional arrays, except that we must provide both the row and column to access an element, giving the row number first.

We create a two-dimensional array by providing the number of rows and columns. Thus we can create the two-dimensional array above by writing:

```
int[][] a = new int[6][4];
```

(Though as good programmers, you would define constants for the number of rows and the number of columns.)

A nested `for` loop is the most common way to access or update the elements of a two-dimensional array. One loop walks through the rows and the other walks through the columns. For example, if we wanted to assign a unique number to each cell of our two-dimensional array, we could do the following:

```
for (int row = 0; row < 6; row++) {
    for (int col = 0; col < 4; col++) {
        a[row][col] = 4*row + col + 1;
    }
}
```

This assigns the numbers 1 through 24 to the elements of array `a`. The array is filled by assigning values to the elements in the first row, then the second row, etc. and results in:

```
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16
17 18 19 20
21 22 23 24
```

And if we wanted to print the above, we can write a loop:

```
for (int row = 0; row < 6; row++) {
    for (int col = 0; col < 4; col++) {
        System.out.print(a[row][col] + " ");
    }
    System.out.println();
}
```

You could modify the above to be slightly more interesting by computing a multiplication table.

We could just as well process all the elements of column 0 first, then all of column 1, etc., by swapping the order of our loops:

```
for (int col = 0; col < 4; col++)
    for (int row = 0; row < 6; row++)
        ...
```

For the most part, it doesn't matter which order you choose, though for large arrays it is generally a good idea to traverse the array in the same order that your programming language will store the values in memory. For Java (and C, C++), the data is stored by rows, known as *row major* order. However, an two-dimensional array in FORTRAN is stored in *column major* order. You will almost certainly see this again if you go on and take courses like Computer Organization or Operating Systems.

Using Shorthand Initialization

Like with one-dimensional arrays, you can declare, construct, and initialize a two-dimensional array if you know all of the values initially:

```
int [][] matrix = { {1, 2},
                   {4, 5},
                   {7, 8} };
```

Two-Dimensional Matrices

One more bit of terminology here: if a two-dimensional array has the same number of rows and columns, it is called a *square array*.

A very common use of two-dimensional arrays is the representation of matrices. Starting in lab, we will look at an example of a class that represents two-dimensional square matrices and provides some basic operations on them.

Ragged Arrays

The arrays in the earlier examples here have the same number of columns in each row. This is not a requirement! Since two-dimensional arrays in Java are really just arrays of array, we can create any number of columns in a row that we want:

```
int [][] nums = new int[3][];
nums[0] = new int[1];
nums[1] = new int[2];
nums[2] = new int[3];
```

This is called a *ragged array*.

The array `nums` has 3 rows, with 1 column in the first, 2 columns in the second, and 3 columns in the third.