Two-Dimensional Arrays (Matrices)

The arrays you have worked with so far have been one-dimensional arrays. To review:

**double**[] array1 = **new** **double**[10];

**boolean**[] array2 = {true, true, false, true, false, false, false};

String[] array3 = {"dog", "cat", "eat", "pie"};

Foo[] array4 = **new** Foo[100];

**for**(**int** k = 0; k < array4.length; k++)

array4[k] = **new** Foo();

The square brackets [] indicate that you are creating an array of doubles, booleans, Strings, or Foos rather than a single double, boolean, String, or Foo. The array itself is an object and is instantiated with the new keyword, unless you use the short-cut { } notation. The indices of the array are always numbered with integers starting from zero. The public field length returns the number of cells in the array. The square brackets are used to access the contents of individual cells. Creating an array of objects is a two-step process: first create the array, then create the objects to fill the array. The same array cannot store objects of different types, unless the objects are subclasses of the declared type. The declared type might even be an abstract type (like Shape) or an interface.

A two-dimensional array is a *matrix*. In Java, a matrix is an array of arrays. For instance:

**double**[][] matrix = **new** **double**[2][3];

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | Matrix | |  |  |
|  |  |  |  |  |  |
|  |  |  | [0] | [1] | [2] |
| [0] |  |  |  |  |  |
| [1] |  |  |  |  |  |

This declaration creates a matrix named matrix with two rows and three columns. Of course the name of a two-dimensional array does not have to be matrix. Always indicate the rows first, then the columns.

To access any one cell you must specify both the row and the column. Some examples:

matrix[0][1] = 3.7;

**for**(**int** r = 0; r < matrix.length; r++) //assigns 1.0 to each

matrix[r][0] = 1.0; //cell in col 0.

**for**(**int** c = 0; c < matrix[0].length; c++) //assigns 2.0 to each

matrix[1][c] = 2.0; //cell in row 1

**for**(**int** r = 0; r < matrix.length; r++) //sets all values

**for**(**int** c = 0; c < matrix[0].length; c++) //in the matrix

matrix[r][c] = 0.0; //to 0.0

The number of rows is indicated by matrix.length. In the example above, matrix.length returns 2. The number of columns is matrix[0].length, which returns 3 in the example above. To fill one column, fix the column value and loop over the rows. To fill one row, fix the row value and loop over the columns. To fill the entire matrix, loop over both the rows and the columns.

Be careful! Always specify the row first and then the column. You can think of RC Cola, which stands for Royal Crown Cola everywhere else but in Computer Science it means **R**ow first, then **C**olumn.

Exercises

averageOfRows

matrix

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 20 | 19 | 20 | 10 | 20 | 18 | 20 | 18 | 20 | 20 |  |  |
| 12 | 20 | 19 | 20 | 19 | 17 | 20 | 19 | 19 | 20 |  |  |
| 19 | 15 | 20 | 0 | 19 | 18 | 20 | 20 | 19 | 16 |  |  |
| 15 | 19 | 18 | 20 | 20 | 16 | 20 | 0 | 19 | 18 |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |

averageOfCols

1. Given a matrix of integers, write code that calculates the average of each row and puts each average in the vertical array of doubles.
2. Given a matrix of integers, write code that calculates the average of each column and puts each average in the horizontal array of doubles.