Computer Programming II Instructor: Greg Shaw

# COP 3337

**Multi-Dimensional Arrays**

**I. Concepts**

In Java, a Two-Dimensional array is implemented as a One-Dimensional array of One-Dimensional arrays. I.e., a list where each element points to another list.

We can visualize a 2D array as a table, with rows and columns. I.e. an “*m by n*” 2D array can be seen as a table with *m* rows and *n* columns. But it’s really a 1D array of *m* elements where each element points to a 1D array of *n* elements.

For example, here is how we see a “3 by 5” 2D array:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 |  |  |  |  |  |
| 1 |  |  | x |  |  |
| 2 |  |  |  |  |  |
|  | 0 | 1 | 2 | 3 | 4 |

And here is what it really is:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 |

|  |  |
| --- | --- |
| 0 |  |
| 1 |  |
| 2 |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | x |  |  |
| 0 | 1 | 2 | 3 | 4 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 0 | 1 | 2 | 3 | 4 |

A Three-Dimensional array is a One-Dimensional array of Two-Dimensional arrays, etc. We can visualize it as a book, where each page contains a table.

1. **Declaring 2D Arrays**

*type* [][] *name* = **new** *type*[rows][cols] ;

1. *type* is the type of data stored in the array (any primitive type or class)
2. *name* is the name of the array object variable
3. *rows* is an integer expression indicating the number of rows
4. *cols* is an integer expression indicating the number of columns

Example**:** int [][] matrix = **new** int [5][4] ;

(creates a 2D array of 5 rows and 4 columns pointed to by *matrix*)

Naturally, the declaration of the object-variable and the creation of the object may be done separately:

int [][] matrix ;

.

.

.

matrix = **new** int [5][4] ;

1. **Accessing the Individual Elements of a 2D Array**

This is similar to accessing the elements of a 1D array, but each element requires two indices (aka: subscripts)

***name*[*row*][*col*]**

* + *name* is the name of the array object variable
  + *row* is an integer expression that tells you which row(i.e. the index of the element of the 1D array that points to the other arrays)
  + *col* is an integer expression that tells you which column(i.e. the index of the element in the array pointed to by *row*)
* The first subscript is *always* the “row” subscript

Example: to access the 3rd column of the 2nd row of the array *table*, use **table[1][2]** (since the first row and first column have index 0)

In the diagrams in **I.**, above, this is the element marked “x”

1. **The *length* Instance Variable**

Every array object has an instance variable called *length* which stores the size (i.e., number of elements) of the array

Since a 2D array is a 1D array of 1D arrays, each array has its own length

int table [][] = **new** int[4][5] ; // 4 rows by 5 columns

int numRows = table.length ; // numRows gets 4

int numCols = table[0].length ; // numCols gets 5

In the above example, the length of the first dimension is 4 and the length of each array “pointed to” is 5

1. **Alternate Notation for 2D Array Declarations**

As with 1D arrays, we may declare 2D arrays by specifying the initial values instead of the size

Java will infer the number of rows and columns from the initial values provided, as shown in this example:

// create a 3 by 4 array containing the ints 1 thru 12

int [][] table = {{1,2,3,4}, // 1st row (table[0])

{5,6,7,8}, // 2nd row (table[1])

{9,10,11,12}} ; // 3rd row (table[2])

table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 1 | 5 | 6 | 7 | 8 |
| 2 | 9 | 10 | 11 | 12 |
|  | 0 | 1 | 2 | 3 |

1. **Traversing a 2D Array**

To *traverse*, or “visit” each element of a 2D array, nested *for* statements are commonly used.

To traverse by rows, the *outer* loop variable is used as the row index and the *inner* loop variable as the column index.

E.g. suppose array *matrix* has been declared as shown here

int [][] matrix = **new** int[4][3] ; // 4 rows by 3 cols

The following code stores the ints 1 thru 12 in *matrix*, by rows, as shown below:

int count = 1 ;

// for each row...

for (int row = 0 ; row < matrix.length ; row++)

{

// visit each column...

for (int col = 0 ; col < matrix[0].length ; col++)

{

matrix[row][col] = count++ ;

}

}

matrix

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 1 | 4 | 5 | 6 |
| 2 | 7 | 8 | 9 |
| 3 | 10 | 11 | 12 |
|  | 0 | 1 | 2 |

To traverse by columns, the *outer* loop variable is used as the column index and the *inner* loop variable as the row index.

// store the ints 1 thru 12 in *matrix*, by columns

count = 1 ;

// for each column...

for (int col = 0 ; col < matrix[0].length ; col++)

{

// visit each row...

for (int row = 0 ; row < matrix.length ; row++)

{

matrix[row][col] = count++ ;

}

}

matrix

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 5 | 9 |
| 1 | 2 | 6 | 10 |
| 2 | 3 | 7 | 11 |
| 3 | 4 | 8 | 12 |
|  | 0 | 1 | 2 |

1. **Accessing An Entire Row of a 2D Array**

Consider the 2D array table, as initialized in **V.**, above, and the following statements:

// declare a 1D array object-variable pointing to the first

// row of *table*

int [] firstRow = table[0] ;

// now create a *deep* *copy* of the first row of *table*

int [] copy = Arrays.copyOf( table[0], table[0].length ) ;

The array object-variable *firstRow* contains a reference to the first row of table (i.e. it is a “pointer” to the array table[0]). So if these statements were executed:

firstRow[1] = 37 ;

System.out.println( table[0][1] ) ;

the output would be 37, because firstRow[1] and table[0][1] both refer to the same memory location.

However, this statement:

System.out.println( copy[1] ) ;

Will print the original value, 2, because the array *copy* is a *duplicate* of the array *table[0]* and was not modified

1. **Sample Programs**

See *VoteCounterTest.java* and *Irregular2D.java*, online