

Chapter 11

Two Dimensional Arrays

Java AP

Copyright Notice

Copyright © 2013 DigiPen (USA) Corp. and its owners. All Rights Reserved

No parts of this publication may be copied or distributed, transmitted, transcribed, stored in a retrieval system, or translated into any human or computer language without the express written permission of DigiPen (USA) Corp., 9931 Willows Road NE, Redmond, WA 98052

Trademarks

DigiPen® is a registered trademark of DigiPen (USA) Corp.

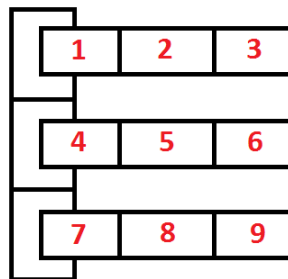
All other product names mentioned in this booklet are trademarks or registered trademarks of their respective companies and are hereby acknowledged.

In the arrays chapter we learned how to create a single dimensional array containing primitive types. Single dimensional arrays are a very useful data structure. However, what if we want to store data in a form of grid? This data can be anything from a simple mathematical matrix to the level data in a game. To accomplish this, we can use a 2 dimensional array also known as a matrix, and is constructed as an array of arrays.

For example:

1	2	3
4	5	6
7	8	9

The above 2D array contains 3 rows and 3 columns. In this case each row is an array of 3 elements, and each column is a position in the sub-arrays.



Creating a 2D Array

Just like creating a 1 dimensional array, we can use the **new** operator to create the memory for our 2D array. We need to follow the following convention:

type arrayName [][] = new type [number of rows] [number of columns];

Example

```
public static void main(String[] args)
{
    /* Creating a 3 rows by 5 columns array, 15 integers in total */
    int arrayOfInts[][] = new int [3][5];

    /* Creating a 12 rows by 5 columns array, 60 doubles in total */
    double arrayOfDoubles[][] = new double [12][5];
}
```

Example	<pre>public static void main(String[] args) { /* Creating a 3 rows by 4 columns array, 12 integers in total */ double points[][] = new double [3][4]; }</pre>																																				
Explanation	<p>We could diagram the array like this:</p> <div><p style="text-align: center;">points</p><table><tr><td></td><td>[0][0]</td><td>[0][1]</td><td>[0][2]</td><td>[0][3]</td><td></td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>[1][0]</td><td></td><td></td><td></td><td></td><td>[1][3]</td></tr><tr><td></td><td></td><td></td><td></td><td></td><td></td></tr><tr><td>[1][1]</td><td></td><td></td><td></td><td></td><td>[1][2]</td></tr><tr><td></td><td>[2][0]</td><td>[2][1]</td><td>[2][2]</td><td>[2][3]</td><td></td></tr></table></div> <p>Accessing the elements is done by using the bracket operators [] and by using the corresponding row and column number.</p> <p>Accessing the above green element is done as follows: points[1][2]</p>		[0][0]	[0][1]	[0][2]	[0][3]								[1][0]					[1][3]							[1][1]					[1][2]		[2][0]	[2][1]	[2][2]	[2][3]	
	[0][0]	[0][1]	[0][2]	[0][3]																																	
[1][0]					[1][3]																																
[1][1]					[1][2]																																
	[2][0]	[2][1]	[2][2]	[2][3]																																	

Initializing a 2D Array at Creation

Just like with a 1 dimensional array, we can create and initialize a 2D array at the same time.

Example	<pre>public static void main(String[] args) { /* Creating and initializing a 1 dimensional array */ int array1D[] = { 1, 2, 3, 4 }; /* Creating and initializing a 2 dimensional array */ double points[][] = { {1.0, 2.0, 3.0, 4.0}, {5.0, 6.0, 7.0, 8.0}, {9.0, 10.0, 11.0, 12.0} }; }</pre>
Explanation	<p>As you can see, initializing a 2 dimensional array is done by initializing each row as a 1 dimensional array.</p>

	points			
	1.0	2.0	3.0	4.0
	5.0	6.0	7.0	8.0
	9.0	10.0	11.0	12.0

It is legal to have differing number of elements in each array:

Example	<pre>public static void main(String[] args) { double points[][] = { {1.0, 2.0, 3.0}, {4.0, 5.0, 6.0, 7.0, 8.0}, {9.0, 10.0, 11.0, 12.0} }; }</pre>															
Explanation	<p>Each row in the above array has a different number of elements.</p> <p>Note: This is not acceptable in all languages</p> <p>points</p> <table><tr><td>1.0</td><td>2.0</td><td>3.0</td><td></td><td></td></tr><tr><td>4.0</td><td>5.0</td><td>6.0</td><td>7.0</td><td>8.0</td></tr><tr><td>9.0</td><td>10.0</td><td>11.0</td><td>12.0</td><td></td></tr></table>	1.0	2.0	3.0			4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0	
1.0	2.0	3.0														
4.0	5.0	6.0	7.0	8.0												
9.0	10.0	11.0	12.0													

Note: You must specify a value for every row and column you want (same rules as initializing a 1 dimensional array). No values can be skipped.

<code>int array1D[] = { 1, , , 4 };</code>	Illegal
<pre>double array2D[][] = { {1.0, , 3.0}, { , 6.0, } };</pre>	Illegal

Reading and Writing to a 2D Array

Again, accessing elements in a 2D array is done using the bracket operators []. The difference between a 1 dimensional array and a 2 dimensional array is that you need to access the specific row first then access the needed element in that row.

Example

```
public static void main(String[] args)
{
    /* Creating and initializing a 2 dimensional array */
    double points[][] = {
        {1.0, 2.0, 3.0, 4.0},
        {5.0, 6.0, 7.0, 8.0},
        {9.0, 10.0, 11.0, 12.0}
    };

    /* Reading from the array and displaying the value */
    System.out.println(points[1][2]);

    /* Writing to the array */
    points[1][2] = 17.0;
    System.out.println(points[1][2]);
}
```

Explanation

points[1][2] is the 3rd value stored in the second row. Initially it is equal to 7.0

points

	[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	1.0	2.0	3.0	4.0
[1][1]	5.0	6.0	7.0	8.0
[1][2]	9.0	10.0	11.0	12.0
	[2][0]	[2][1]	[2][2]	[2][3]

The following command: `"System.out.println(points[1][2]);"` is accessing the array and reading the value **7.0** and outputting it.

Then, `"points[1][2] = 17.0;"` is writing to the array. **7.0** will be replaced by **17.0**

points

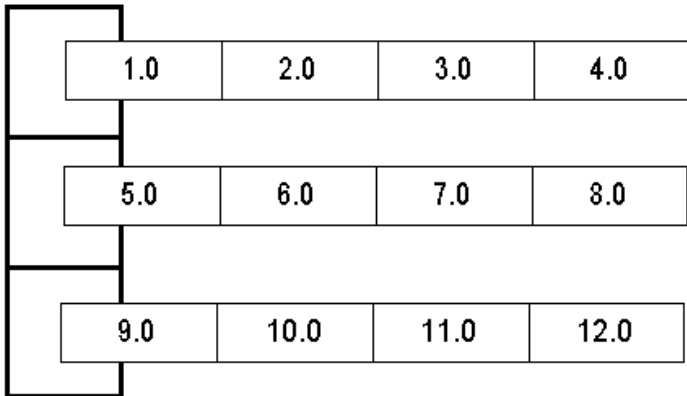
	[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	1.0	2.0	3.0	4.0
[1][1]	5.0	6.0	17.0	8.0
[1][2]	9.0	10.0	11.0	12.0
	[2][0]	[2][1]	[2][2]	[2][3]

A common mistake is using one index while accessing a 2D array.

Example	<pre>public static void main(String[] args) { /* Creating and initializing a 2 dimensional array */ double points[][] = { {1.0, 2.0, 3.0, 4.0}, {5.0, 6.0, 7.0, 8.0}, {9.0, 10.0, 11.0, 12.0} }; System.out.println(points[1]); }</pre>												
Output:	[D@2607c28c												
Explanation	<p>First, you need to ask yourself what is the type of "points[1]". "points[1]" is an array which forms the second row in our 2D array.</p> <div><p>points</p><table><tr><td>1.0</td><td>2.0</td><td>3.0</td><td>4.0</td></tr><tr><td>5.0</td><td>6.0</td><td>7.0</td><td>8.0</td></tr><tr><td>9.0</td><td>10.0</td><td>11.0</td><td>12.0</td></tr></table></div> <p>The following command: "System.out.println(points[1]);" will output the address in memory where the second row resides, which in our case is [D@2607c28c.</p> <p>Note: That location might be different every time you run the application.</p>	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	11.0	12.0
1.0	2.0	3.0	4.0										
5.0	6.0	7.0	8.0										
9.0	10.0	11.0	12.0										

The length property

What do we expect the length property to return?

Example	<pre>public static void main(String[] args) { /* Creating a 2 dimensional array */ int array2D[][] = new int [3][4]; System.out.println(array2D.length); System.out.println(array2D[0].length); }</pre>
Output:	<pre>3 4</pre>
Explanation	<p>Before we analyze the output, remember that the above array can be represented as follows:</p>  <p>Where, <code>array2D</code> is a 1 dimensional array that contains 3 elements. Each one of its elements is actually an array itself.</p> <p>The length property in <code>array2D</code> only keeps track of how many elements the vertical array (shown in the image above) contains which is 3.</p> <p>Of course, when outputting the length property of a specific row "<code>System.out.println(array2D[0].length);</code>", you are getting the number of elements found in that inner array. In our case, all of them are made out of 4 elements.</p>

Let's analyze the following code now:

Example	<pre>public static void main(String[] args) { short array2D[][] = new short[3][]; array2D[0] = new short[3]; array2D[1] = new short[5]; array2D[2] = new short[2]; System.out.println(array2D[0].length); System.out.println(array2D[1].length); System.out.println(array2D[2].length); }</pre>
Output:	<pre>3 5 2</pre>
Explanation	<p>Starting with: <code>short array2D[][] = new short[3][];</code> We are allowed to create a 2D array and only specify how many rows it contains. Later, we can create the inner arrays in each and every row.</p> <p>Note: "<code>short array2D[][] = new short[][];</code>" is illegal. You need to specify how many rows we have in a 2 dimensional array. The following error shows up if you don't specify.</p> <p><i>Exception in thread "main" java.lang.Error: Unresolved compilation problem: Variable must provide either dimension expressions or an array initializer</i></p> <p><i>at main.main(main.java:6)</i></p> <p>Second, we can start creating each row. As mentioned before, rows don't need to have the same amount of elements in them.</p> <pre>array2D[0] = new short[3]; //contains 3 elements array2D[1] = new short[5]; //contains 5 elements array2D[2] = new short[2]; // contains 2 elements</pre> <p>Note: If you leave one of the rows empty your application will crash with a null pointer exception when trying to access elements in that row!</p>

Looping Through a 2D Array

In order to loop through the elements of a 2D array we need a nested loop. One loop for the rows and the other for the columns.

Example	<pre> public static void main(String[] args) { /* Creating and initializing a 2 dimensional array */ double array2D[][] = { {1.0, 2.0, 3.0, 4.0}, {5.0, 6.0, 7.0, 8.0} }; /* Looping through the rows */ for(int r = 0; r < array2D.length; ++r) { /* looping through the elements in row r */ for(int c = 0; c < array2D[r].length; ++c) { System.out.print(array2D[r][c] + "\t"); } System.out.println(); } } </pre>
Output:	<pre> 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 </pre>

Using the for-each loop to go through a 2D array:

Example	<pre> public class main { public static void main(String[] args) { int array2D[][] = {{1,2,3},{4,5,6}}; for(int[] row : array2D) { for(int element : row) { System.out.print(element + " "); } System.out.println(); } } } </pre>
---------	---

Output:	1 2 3 4 5 6
Explanation:	<p>As mentioned many times previously, every row inside a 2D array is a 1D array.</p> <p>The first for-each loop is going through the rows and that is why the "row" variable should have the type "int []" which is a 1D array of integers.</p> <p>The second for-each loop is looping through each row that contains integers, hence the "int" type for the "element" variable.</p>

Passing a 2D Array to a Function

Just like a 1D array a 2D array is an object, so passing it as parameter means passing its object reference to the function. It assures that no copy of the 2D array is made. Additionally each row can be treated as a separate object since it is an array.

Example	<pre> public class main { public static void print2DArray(int array[][]) { //Looping through all of the rows for(int row = 0; row < array.length; ++row) { //Looping through all of the columns for(int col = 0; col < array[row].length; ++col) { System.out.print(array[row][col] + " "); } System.out.println(); } } public static void main(String[] args) { int[][] mat = {{1,2,3},{4,5,6}}; print2DArray(mat); } } </pre>
Output:	1 2 3 4 5 6

Since a reference of the array is actually passed to the function, changing any value in the sent array leads to changing in the original array.

Example

```
public class main
{
    public static void print2DArray(int array[][])
    {
        //Looping through all of the rows
        for(int row = 0; row < array.length; ++row)
        {
            //Looping through all of the columns
            for(int col = 0; col < array[row].length; ++col)
            {
                System.out.print(array[row][col] + " ");
            }
            System.out.println();
        }
    }

    public static void clear2DArray(int array[][])
    {
        //Looping through all of the rows
        for(int row = 0; row < array.length; ++row)
        {
            //Looping through all of the columns
            for(int col = 0; col < array[row].length; ++col)
            {
                //Setting the element to 0
                array[row][col] = 0;
            }
        }
    }

    public static void main(String[] args)
    {
        int mat[][] = {{1,2,3},{4,5,6}};
        System.out.println("Original Array:");
        print2DArray(mat);
        clear2DArray(mat);
        System.out.println("Array After Clearing:");
        print2DArray(mat);
    }
}
```

Output:

```
Original Array:
1 2 3
4 5 6
Array After Clearing:
0 0 0
0 0 0
```

As mentioned previously, a row in a 2D array is a 1D array so it is possible to pass each row separately to a function as a 1D array.

Example	<pre>public class main { public static void print1DArray(int array[]) { for(int i = 0; i < array.length; ++i) { System.out.print(array[i] + " "); } System.out.println(); } public static void main(String[] args) { int mat[][] = {{1,2,3},{4,5,6}}; for(int row = 0; row < mat.length; ++row) { print1DArray(mat[row]); } } }</pre>
Output:	<pre>1 2 3 4 5 6</pre>