CS620c

Introducing **Two dimensional (2D) arrays.**

Joe Duffin

Callan Building Room 2.108

Email: Joseph.Duffin@nuim.ie

Why do we need two dimensional arrays?

- You may want to store a collection of values that have a two dimensional layout like a table.

- This arrangement consisting of rows and columns of values is called a two-dimensional array or matrix.

- In java you create a two dimensional array by supplying the number of rows and columns.

```
int[][] counts = new int [7][3];
```

This is a combined array reference declaration and array memory slots creation. This results in the creation in a seven rows by three columns matrix of integer storage locations in our program.

How do we create a Two dimensional array?

In general an array declaration looks like this:

```
    type[][] arrayName = new type[ rows ][columns];
```

This determines the type of information in each slot and the number of slots. Once an array has been constructed, the number of slots it has does **not** change. You can also separate the **array reference declaration** from the memory block assignment

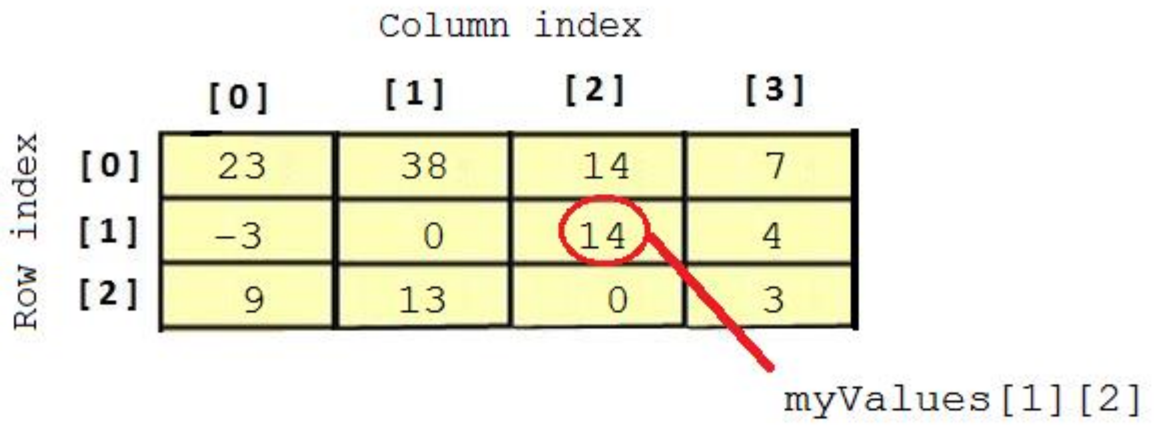
```
        type[][] arrayName;

        arrayName = new type[ rows][columns];
```

Array creation continued

You can declare, construct, and initialise the array all in one statement:

```
int[][] myValues= { {23,38,14,7},
                    {-3, 0,14,4},
                    { 9, 13,0,3},
                  };
```

This declares a **2D int array** which is named myValues (myValues is a link references to the created array. “Ultimately” the compiler constructs a 2D array of 3x4 memory slots and then puts the designated values into the slots.



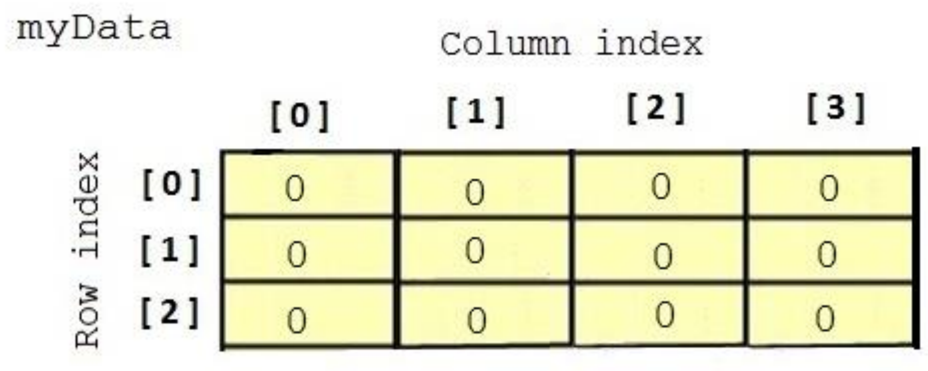
Also you can us this layout int[ ][ ] twoDimArray = { {1,2,3}, {4,5,6}, {7,8,9} };

Array creation continued

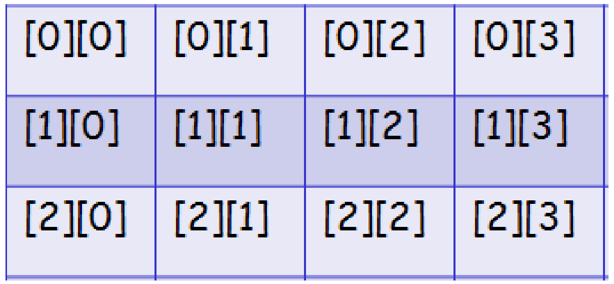
You can declare, construct and array without intialising it:

```
int[][] myData = new int[3][4];
```

This declares an **2D array of int** which is named myData.
The compiler constructs a 2D array of 3x4 memory slots and intialises each slot to have the value zero.


myData
Column index
Row index

	[0]	[1]	[2]	[3]
[0]	0	0	0	0
[1]	0	0	0	0
[2]	0	0	0	0

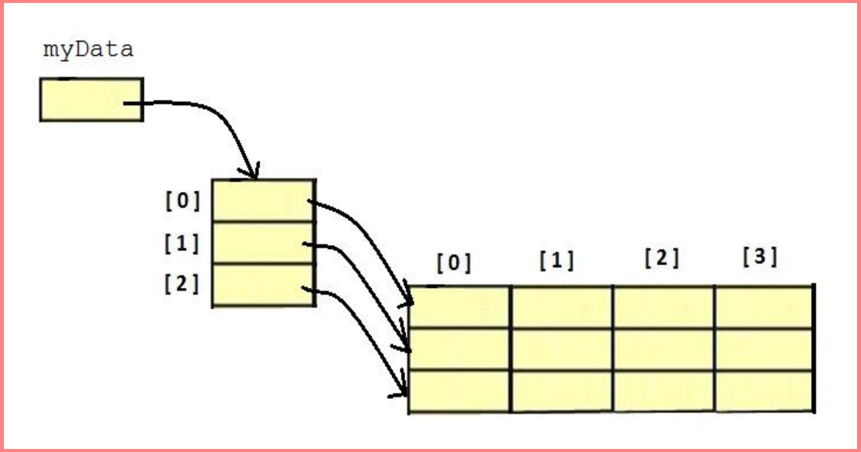


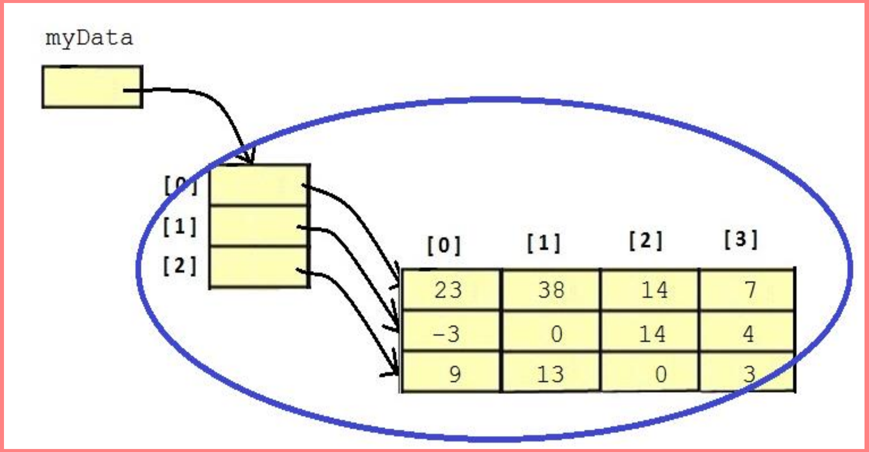
[0][0]	[0][1]	[0][2]	[0][3]
[1][0]	[1][1]	[1][2]	[1][3]
[2][0]	[2][1]	[2][2]	[2][3]

You need to specify at least the row (first index). If you leave the column blank you can create a non rectangular array. It is possible to create a triangular array. See Hortsman chapter 6 p. 288

What our 2 D looks like using memory box diagrams.

The declaration int [ ][ ] myData is for a reference variable **myData** which when assigned will contain a link reference to a (a) one dimensional array of references to (b) one dimensional arrays!!!



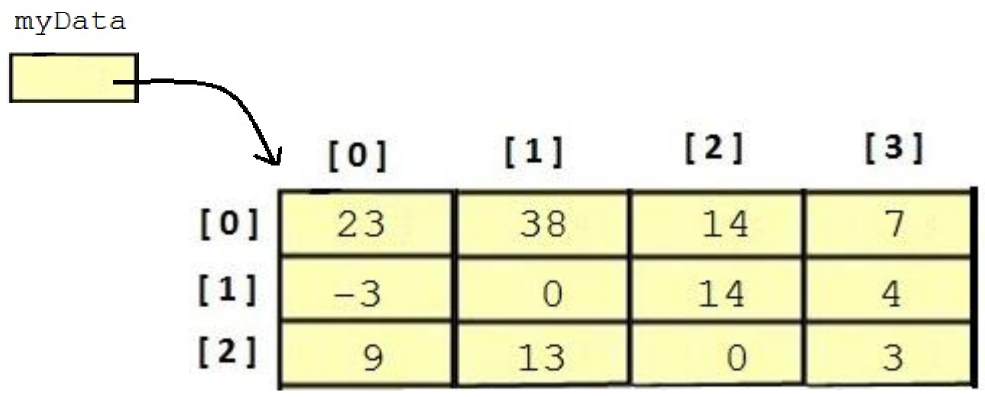


As a short hand just treat the 2D array myData as you would a 1D array as in the diagram on the right hand side. MyData is a link reference to a 2D array. (See last lessons notes on passing arrays to methods etc)

To fill the myData array you could do this element by element as below

```
int[][] myData = new int[3][4];

myData[ 0 ][ 0 ] = 23;
myData[ 0 ][ 1 ] = 38;
myData[ 0 ][ 2 ] = 14;
myData[ 0 ][ 3 ] = 7;
myData[ 1 ][ 0 ] = -3;
myData[ 1 ][ 1 ] = 0;
myData[ 1 ][ 2 ] = 14;
myData[ 1 ][ 3 ] = 4;
myData[ 2 ][ 0 ] = 9;
myData[ 2 ][ 1 ] = 13;
myData[ 2 ][ 2 ] = 0;
myData[ 2 ][ 3 ] = 3;
…..
```

We can consider the array as referenced below.



	[0]	[1]	[2]	[3]
[0]	23	38	14	7
[1]	-3	0	14	4
[2]	9	13	0	3

```
// But you really should use nested loops to access array
elements.
```

Using nested for loops to access the 2D array.

As with one dimensional arrays you can access the individual array slot (array elements) by using the counter variable of a loop. As there are now two dimensions you need two loops one nested inside the other.
The inner loop block { } will contain a statement which refers to the individual memory element of the 2D array and it does this by using both the its own inner loop index and outer loop index.

```
for (int row = 0; row < 3; row++){

     for (int column = 0; column < 4; column++){

       System.out.print(" " + myValues[row][column]);
      }

      System.out.println();
 }
```

Using nested for loops to print out indices as if they were array elements.

There is a convention in programming whereby programmers use the letter i and the letter j as the row and column indices for accessing 2D array elements.
This is not a rigid convention and is ok to use the identifiers (variable names) row and column.
The code below does not access an array but just prints out the loop counters (indices) as if they were array elements indices.

```
for (int row = 0; row < 3; row++){

     for (int column = 0; column < 4; column++){

          System.out.print(" ["+row+"]["+column+"] ");
      }

     System.out.println();
 }
```

Using 2D arrays built in length values.

As a result of the way arrays are implemented in Java, if the 2D array is a **rectangular** array you can use **myValues.length** as the number of rows and **myValues[0].length** as the number of columns. This assumes that all the rows are the same length so you can access the length for the first row **myValues[0].length** as it will be the same for the other rows which have equal number of elements.

```
for (int row = 0;row < myValues.length;row++){

    for (int column = 0;column < myValues[0].length;column++){

          System.out.print(" "+myValues[row][column]);
    }

    System.out.println();
}
```