# Object-Oriented Development (CIS1056-N) Worksheet 11: 2D Arrays

## Before You Start

Remember: You are not expected to complete the entire brief within the allotted two hours, but to make a start and continue outside of the class.

## Introduction

Your priority should be to make a start on the ICA. If you are yet to start, or not sure where to start, you should urgently speak to you tutor.

You can also use the practical time to follow along with the additional PlayingCard videos.

## 1. 2D Array Declaration and usage examples

To declare a 2d array is essentially the same as a 1d array, however you add another dimension. A 2d array is said to be an ***array of arrays***. Each row is a reference to a 1-dimensional array object - for anyone with experience of C or C++, the Java 2d array syntax is the very similar, the way that the arrays relate to memory is different.

The bracket nearest the type is the number of rows, the bracket furthest from the type is the columns.

**int[][] matrix1 = new int[2][3]; // 2 rows, 3 columns**

Java will initialise the array to 2 rows of 3 columns of 0s (zeros). To overwrite the values use 2 sets of brackets to index the row (nearest the array identifier) and the column (farthest from the array identifier).

**// populate:**

**//      r  c**

**//      o  o**

**//      w  l**

**matrix1[0][0] =   6;  // row 0, col 0**

**matrix1[0][1] =   4;  // row 0, col 1**

**matrix1[0][2] =  24;  // row 0, col 2**

**matrix1[1][0] =   1;  // row 1, col 0**

**matrix1[1][1] =  -9;  // row 1, col 1**

**matrix1[1][2] =   8;  // row 1, col 2**

With the array populate, we can visualise the array like a spreadsheet:

|  |  |  |  |
| --- | --- | --- | --- |
| *indices* | 0 | 1 | 2 |
| 0 | 6 | 4 | 23 |
| 1 | 1 | -9 | 8 |

If you know the values for the array, you may declare and initialise the array at the same time. Notice that each row of values is surrounded by curly-braces:

**int[][] matrix2 = {**

**{  4,  0 }, // Row 0**

**{  1, -9 }  // Row 1**

**};**

Here is an example of declaring a 2d array for Strings:

**String[][] string2dArray = new String[2][3];**

**string2dArray[0][0] = "Aardvark";**

**string2dArray[0][1] = "Azure";**

**string2dArray[1][0] = "Baa";**

**string2dArray[1][1] = "Baz";**

**for(String[] row : string2dArray) {**

**for(String col : row) {**

**System.out.print(col + " ");**

**}**

**System.out.println();**

**}**

Below is a method that you could use to display the contents of a 2d array containing integers and a message:

**private static void displayArray(String message, int[][] matrix) {**

**System.out.println("-".repeat(message.length()));**

**System.out.println(message);**

**System.out.println("-".repeat(message.length()));**

**for(int[] row : matrix) {**

**System.out.print("|");**

**for(int col : row) {**

**System.out.print(String.format("%3d |", col));**

**}**

**System.out.println();**

**}**

**}**

## 2. Matrices Example – Multiply by a Scalar

2D arrays are useful for representing matrices. This exercise will use the following maths tutorial about multiplying matrices: <https://www.mathsisfun.com/algebra/matrix-multiplying.html>

Create a Netbeans project and try out the following examples:

* Given the matrix represented by a 2d array:

**// Multiply a matrix by a single number (Scalar):**

**int[][] matrix2 = {**

**{  4,  0 },**

**{  1, -9 }**

**};**

**displayArray("Matrix 2 - Before Multiplication by a Scaler", matrix2);**

* We could use nested loops to multiply each value in the matrix by the scalar value:

**for(int[] row: matrix2) {**

**for(int colIdx = 0; colIdx < row.length; colIdx++) {**

**row[colIdx] \*= 2;**

**}**

**}**

**displayArray("Matrix 3 - After Multiplication by a Scaler", matrix2);**

1. Notice that the outer loop is a **for..in** style loop, whereas the inner loop is an *indexed loop* – **Why do you think that this is the case?**
2. Rewrite the inner-loop as a for..in type loop – does it still work?

## 3. Matrix Example – Matrix/Matrix multiplication

Given 2 matrices, a common operation is to multiply them to get a new matrix. Below we define 2 matrices, notice that the first is a 2 x 3 matrix, whereas the second is a 3 x 2 matrix. There must be the same number of columns in the first array as there are in rows in the second.

Read the following the tutorial for more information: <https://www.mathsisfun.com/algebra/matrix-multiplying.html>

Now let’s write the Java that multiplies the 2 matrices and populates 3rd matrix with the result.

**// Define our 2 matrices**

**int[][] matrix3 = {**

**{ 1, 2, 3 },**

**{ 4, 5, 6 }**

**};**

**int[][] matrix4 = {**

**{  7,  8 },**

**{  9, 10 },**

**{ 11, 12 }**

**};**

We can now compute the resulting matrix by multiplying the two matrices. Notice how it uses three loops. The two matrices must be compatible, where the column count of the first matrix must match the row count of the second matrix.

**// matrix3 column count must equal matrix4 row count**

**if(matrix3[0].length == matrix4.length) {**

**// Java will default the array values to 0**

**int[][] resultMatrix = new int[matrix3.length][matrix3.length];**

**for(int i = 0; i < matrix3.length; i++) {**

**for(int j = 0; j < matrix4[i].length; j++) {**

**for(int k = 0; k < matrix4.length; k++) {**

**resultMatrix[i][j] += matrix3[i][k] \* matrix4[k][j];**

**}**

**}**

**System.out.println();**

**}**

**displayArray("Matrix Multiplication: ", resultMatrix);**

**}**

1. On paper, work out how the value of each cell of the **resultMatrix** is computed.
2. What happens if you add another row to **matrix3**?
   1. Do you need to make any further changes for the algorithm to compute the dot product to work?
3. Try rewriting the multiplication algorithm by removing the innermost loop (the ***k*** loop) and hard-coding the calculation steps it would be used for.
   1. Can you think of any advantages for doing this?
   2. What are the disadvantages of doing this?

## 4. Matrix Example – Local shop example

Create a new Netbeans project and implement the local shop example from this site: https://www.mathsisfun.com/algebra/matrix-multiplying.html

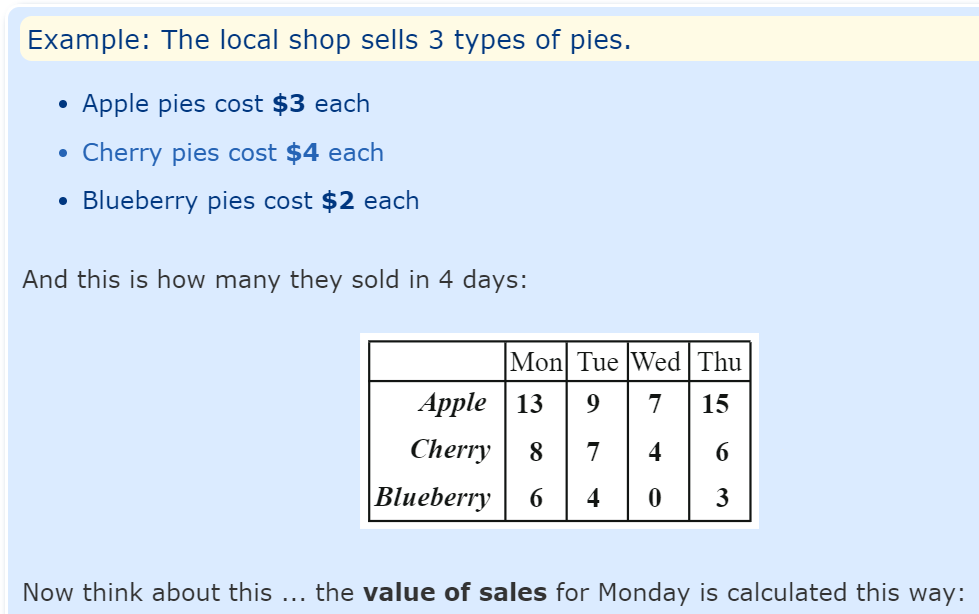


Figure 1: Screenshot from https://www.mathsisfun.com/algebra/matrix-multiplying.html

## 5. Jagged Arrays

In the previous examples, each individual array had rows with the same number of columns, i.e. they are rectangular like a spreadsheet. For some problems this is important (such as matrices), for other problems this is less important.

Arrays that do not have a uniform number columns for each row (or put another way – not rectangular) are known as “jagged arrays” (also known as “ragged arrays” or “irregular array”).

Here is an example of dynamically creating a “jagged array”:

**Random rand = new Random();**

**// Create a array of a random number of rows (min 1 row)**

**int[][] jaggedArray = new int[1 + rand.nextInt(10)][];**

**for(int rowIdx = 0; rowIdx < jaggedArray.length; rowIdx++) {**

**// Create a row of a random number of columns (min 1 col)**

**jaggedArray[rowIdx] = new int[1 + rand.nextInt(10)];**

**// Populate the column values**

**for(int colIdx = 0; colIdx < jaggedArray[rowIdx].length; colIdx++) {**

**jaggedArray[rowIdx][colIdx] = rand.nextInt(100);**

**}**

**}**

**displayArray("The contents of the \"jagged array\"", resultMatrix);**

Example Run:

**----------------------------------**

**The contents of the "jagged array"**

**----------------------------------**

**| 29 | 1 | 35 | 83 | 66 | 52 | 28 | 67 | 18 |**

**| 44 | 10 |**

**| 61 | 58 | 69 | 52 | 28 |**

**| 48 | 71 | 30 | 89 | 86 | 70 |**

**| 76 | 79 |**

**| 57 |**

**| 98 |**

**| 6 | 89 | 21 | 28 | 85 | 3 | 84 |**

**| 10 | 72 | 14 | 0 | 82 | 81 | 16 | 59 |**

**| 26** |

1. Create a new Netbeans project and implement the “jagged” array code above.
2. Modify the code so that the user is required to input the maximum values used for the number of rows, columns, and individual values stored in the array cells.
3. Read about “jagged arrays”: <https://en.wikipedia.org/wiki/Jagged_array>

## 6. Matrices revisited

Create a new Netbeans project. You will now create a new class called **Matrix** which will manage a 2D array of doubles.

The Matrix class will need:

* A constructor to specify the number of *rows* and *columns* – store these values as attributes of the class.
* An overloaded constructor that accepts a 2D array from which the matrix copies its data from.
* An overloaded constructor that accepts another **Matrix** as a single input and copies the data from it.
* A method to set a row/column in the **Matrix** with a specific value.
* A method called ‘**multiply**’ to multiply a matrix by a scalar value, it will return a new **Matrix** with the result.
* An overloaded method called ‘**multiply**’ to multiply 2 matrices together, it will return a new **Matrix** with the result.
* An overloaded **toString()** method to display its current state.

Remember, that to multiply two matrices together they must be compatible as described earlier.

Example use:

**Matrix m1 = new Matrix(3, 2);**

**Martix m2 = new Matrix(2, 3);**

**Martix m3 = new Matrix(new int[][] {{ 1, 2}, { 2, 3 }});**

**Martix copyOfm3 = new Matrix(m3);**

**// ... populate the two matrices ...**

m1.setValue(0, 0, 99); **// etc…**

**Matrix result1 = m1.multiply(2); // multiple by a scalar**

**Matrix result2 = m1.multiply(m2); // multiple by another matrix**

**System.out.println(result1); // print out the resultant matrix**

**System.out.println(result2); // print out the resultant matrix**

## Document History

Revision 0 (28-Nov-22): This is the initial version of the 2022/23 exercise.