2D Array Exercise

Objectives:

Practice using two dimensional arrays to input and output data in a tabular format.

A Simple 2D array example

```
//********************************************************************
//  TwoDArray.java       Author: Lewis/Loftus
//  Demonstrates the use of a two-dimensional array.
//********************************************************************
public class TwoDArray
{
   //-----------------------------------------------------------------
   //  Creates a 2D array of integers, fills it with increasing
   //  integer values, then prints them out.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int[][] table = new int[5][10];
      // Load the table with values
      for (int row=0; row < table.length; row++)
         for (int col=0; col < table[row].length; col++)
            table[row][col] = row * 10 + col;
      // Print the table
      for (int row=0; row < table.length; row++)
      {
         for (int col=0; col < table[row].length; col++)
            System.out.print (table[row][col] + "\t");
         System.out.println();
      }
   }
}
```

Run this program and observe what it does.

The output produced is shown below.

- Circle the entries for **table[0][5]** and **table[3][2]**

```
0     1     2     3     4     5     6     7     8     9
10    11    12    13    14    15    16    17    18    19
20    21    22    23    24    25    26    27    28    29
30    31    32    33    34    35    36    37    38    39
40    41    42    43    44    45    46    47    48    49
```

Rewrite to label output rows and columns.

- The output should now look EXACTLY like this:

```
#  |   0     1     2     3     4     5     6     7     8     9
---+-----------------------------------------------------------
0  |   0     1     2     3     4     5     6     7     8     9
1  |   10    11    12    13    14    15    16    17    18    19
2  |   20    21    22    23    24    25    26    27    28    29
3  |   30    31    32    33    34    35    36    37    38    39
4  |   40    41    42    43    44    45    46    47    48    49
```

Modify the dimensions of the array – make it 3 rows by 4 columns and run the program again. The output should look right *without having to change anything else in the program*. If necessary, adapt your program so that it works with any reasonable dimensions (note that there is a limit to how many columns can be displayed across on one line, so it is not expected to work well with large values for the number of columns).

A 2D array of `double`

Make a new version of your program that creates instead a 2D array of 5x5 values of type `double`, set to random values in the range 0....1 (use Math.random()).

Input values from a file

Make a new version of your program that inputs values from a file, stored in a tabular format. You can try this either with values of type double or of type int. Prepare a file with appropriate input, eg:

```
110    31     20     45
0      11     0      13
320    61     27     2
```

Hints:

- Use a Scanner to scan input from the file, as in Lab 12
- In the outer loop input a line from the file (the whole line, as a String). Set up a second Scanner to input from that line.
- In the inner loop use the second Scanner to input numbers (use nextInt() or nextDouble() as appropriate).
- We are assuming here that the input will be correctly formatted (i.e., you know exactly how many rows and columns it will have, and there are no errors or omissions in the file), so you can continue to use the loop structure as in the original program, without checking .hasNextInt(), etc.

Optional: A 2D array of boolean

Make a new version of your program that creates instead a 2D array of 5x5 values of type `boolean`.

1) Suppose indices represent people and that the value at row `i`, column `j` of a 2D array is `true` just in case `i` and `j` are friends and `false` otherwise. Use initializer list to instatiate and initialize your array to represent the following configuration:

```
# | 0  1  2  3  4
--+---------------
0 |    *     *  *
1 | *     *     *
2 |    *
3 | *           *
4 | *  *     *
```

(* means "friends")

2) Write some code to count how many pairs of friends are represented in the array. Note that each friendship pair appears twice in the array, so in the example above there are 6 pairs of friends).

3) Write a method to check whether two people have a common friend. For example, in the example above, 0 and 4 are both friends with 3 (so they have a common friend), whereas 1 and 2 have no common friends. The method should have three parameters: a 2D array of **`boolean`** representing the friendship relationships and two integers **`i`**, **`j`**. The method should return **`true`** if there is an integer **`k`** such that **`i`** is a friend of **`k`** and **`k`** is a friend of **`j`** and return **`false`** otherwise. Make this method **`static`** and test it from the **`main()`** method.