2D Array Problems

`printOutArray` prints out a two-dimensional array of integers

```
public static void printOutArray(int[][] array) {
    for(int r = 0; r < array.length; r++) {
        for(int c = 0; c < array[r].length; c++) {
            System.out.printf("%3d ", array[r][c]);
        } // end for
        System.out.println();
    } // end for
} // printOutArray
```

`createMultiplicationTables` constructs an array containing multiplication table for the given starting and ending integers

```
public static int[][] createMultiplicationTables(int startingAt, int goingTo) {

    int sizeOfRange = goingTo - startingAt + 1;
    int[][] retVal = new int[sizeOfRange][sizeOfRange];
    for(int r = 0; r < sizeOfRange; r++) {
        for(int c = 0; c < sizeOfRange; c++) {
            retVal[r][c] = (startingAt + r)*(startingAt + c);
        } // end for
    } // end for
    return retVal;
} // createMultiplicationTables
```

`findPairDistance` takes an array representing a map of positions and an element to search for. The operation requires that the searched for element will exist in the array exactly 2 times (this is the “pair”).

The function returns the distance between the of the pair in the map of positions. The distance is the Manhattan distance, i.e., the distance in terms of number of steps directly north, south, east, or west.
For example, given the 3 x 4 array map to the right:

```
aa.b
....
..b.
```

```
int dist1 = findPairDistance(map,'a'); // yields 1
```

because the 2nd ‘a’ is right next to the 1st ‘a’

```
int dist2 = findPairDistance(map,'b'); yields 3
```

because it takes 3 steps to get from the 1st ‘b’ to the 2nd ‘b’, by going: west, south, south

```
public static int findPairDistance(char[][] map, char pairToFind) {
// requires: pairToFind to be present in map exactly 2 times
    final int NOT_FOUND = -1;
    int firstRow = NOT_FOUND;
    int firstColumn = NOT_FOUND;
    for(int r = 0; r < map.length; r++) {
        for(int c = 0; c < map[r].length; c++) {
            char current = map[r][c];
            if(current == pairToFind) {
                if(firstRow == NOT_FOUND) {
                    firstRow = r;
                    firstColumn = c;
                } else {
                    int distance = Math.abs(firstRow - r) + Math.abs(firstColumn - c);
                    return distance;
                } // end if
            } // end if
        } // end for
    } // end for
    //should never get here
    return NOT_FOUND;
} // findPairDistance
```

Array Reference

hint – you might save this page off for later reference

```
// A 2D array in Java stores a whole table of information
// It has 2 indexes - a row and column
// You can make 2D arrays of any shape, similar to the way you
// make 1D arrays of any length

int[][] myIntArray = new int[50][7]; //50 x 7 array (50 rows, 7 columns)
String[][] myStringArray = new String[10][200]; //10 x 200 array

// assign and access elements like you expect
myIntArray[45][5] = 77;
myStringArray[6][157] = "hello";

System.out.println("value at index (45, 5) " + myIntArray[45][5]);
// note that both indexes start at 0
System.out.println("value at (9, 199) the largest indexes myStringArray: " +
myStringArray[9][199]);

// if you want to get the dimensions, the usual length will give you the first
// dimension
System.out.println(myIntArray.length); //prints 50 - i.e., the number of rows
//to get the second dimension, do it like this
System.out.println(myIntArray[0].length); //prints 7 - i.e., the number of columns
```