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
```

Map Problems

Java Maps are pretty much the same as Python Dictionaries.

`numberToLargestDivisor` constructs a map associating a number with its largest divisor. For example, `numberToLargestDivisior(10)` yields {2=1, 3=1, 4=2, 5=1, 6=3, 7=1, 8=4, 9=3, 10=5}

```
public static HashMap<Integer,Integer> numberToLargestDivisor(int maxNum) {
    HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
    for(int k = 2; k <= maxNum; k++) {
        int numberToFindDivisorOf = k;
        for(int j = numberToFindDivisorOf / 2; j > 0; j--) {
            if(numberToFindDivisorOf % j == 0) {
                map.put(numberToFindDivisorOf, j);
                break;
            } // end if
        } // end for
    } // end for
    return map;
} // numberToLargestDivisor
```

`doubleMap` takes a map and returns a new map of where both the keys and values are double. So far example, {A=a, BB=bb} yields {AA=aa, BBBB=bbbb}

```
public static HashMap<String,String> doubleMap(HashMap<String,String> originalMap) {
    HashMap<String, String> retVal = new HashMap<String, String>();
    for(String key : originalMap.keySet()) {
        retVal.put(key + key, originalMap.get(key) + originalMap.get(key));
    } // end for
    return retVal;
} // doubleMap
```

`findMostFrequentStartingLetter` takes an array of strings and returns the most frequent beginning letter. So for example, the strings {“ant”,”bug”,”aunt”} yield 'a'

```
public static char findMostFrequentStartingLetter(String[] strings) {
    HashMap<Character, Integer> letterToFreq = new HashMap<Character, Integer>();
    for(String current : strings) {
        char startingChar = current.charAt(0);
        if(!letterToFreq.containsKey(startingChar)) {
            letterToFreq.put(startingChar, 0);
        } // end if
        int currentFrequency = letterToFreq.get(startingChar) + 1;
        letterToFreq.put(startingChar, currentFrequency);
    } // end for

    //ok now find the most frequently appearing 1st character
    int highestFrequency = 0;
    char charWithHighest = '\0';
    for(char current : letterToFreq.keySet()) {
        if(letterToFreq.get(current) > highestFrequency) {
            highestFreq = letterToFreq.get(current);
            charWithHighest = current;
        } // end if
    } // end for
    return charWithHighest;
} // findMostFrequentStartingLetter
```

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
```

Map Reference

```
// A Java Map is like a dictionary in Python
// It associates a key with a particular value - but because this is
// java the key and the value both must be typed
//
// Format: HashMap<KeyType,ValueType> foo = new HashMap<KeyType,ValueType>();

HashMap<String, Integer> namesToWeight = new HashMap<String, Integer>();

//to add elements to the map, use put
namesToWeight.put("Buffalo", 160); // key = "Buffalo", value = 160
namesToWeight.put("Gretchen", 130);// key = "Gretchen", value = 130

//note that putting twice with the same key overwrites the original value in the map
namesToWeight.put("Buffalo", 165);

//to get elements out of the map, use get
System.out.println("Buffalo's weight is " + namesToWeight.get("Buffalo"));

//if you need to check if a particular key is in the map, use containsKey method
if(namesToWeight.containsKey("Steve")) {
      System.out.println("Steve is in the map!");
} // end if

//if you need iterate over all the keys in the map, use the keyset method
Set<String> keys = namesToWeight.keySet();
// Use the enhanced for loop to iterate over the entire set
for(String key : keys) {
      int value = namesToWeight.get(key);
      //do something for every key and value
} // end for
```