2D Array Problems

This code prints out a two dimensional array of integers.

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

This code 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[][] result = new int[sizeOfRange][sizeOfRange];
	for(int i = 0; i < sizeOfRange; i++) {
		for(int j = 0; j < sizeOfRange; j++) {
			result[i][j] = (startingAt + i)*(startingAt+j);
		}
	}
	return result;
}
```

This code takes an array representing a map of positions and an element to search for. 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). So for example, given the array:

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

findPairDistance(map,'a') yields 1 (because is right next to the other a)
findPairDistance(map,'b') yields 3
(because it takes 3 steps to get from the first b to the second = west south
south)
```

```
public static int findPairDistance(char[][] map, char pairToFind) {
	int NOT_FOUND = -1;
	int firstRow = NOT_FOUND;
	int firstColumn = NOT_FOUND;
	for(int i = 0; i < map.length; i++) {
		for(int j = 0; j < map[i].length; j++) {
			char current = map[i][j];
			if(current == pairToFind) {
				if(firstRow == NOT_FOUND) {
					firstRow = i;
					firstColumn = j;
				} else {
					int distance = Math.abs(firstRow - i) +
Math.abs(firstColumn - j);
					return distance;
				}
			}
		}
	}
	//should never get here
	return NOT_FOUND;
}
```

Map Problems

Maps are pretty much the same as Dictionaries in Python.

This function constructs a map associating a number with it's largest divisor. So 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 i = 2; i <= maxNum; i++) {
		int numberToFindDivisorOf = i;
		for(int j = numberToFindDivisorOf / 2; j > 0; j--) {
			if(numberToFindDivisorOf % j == 0) {
				map.put(numberToFindDivisorOf, j);
				break;
			}
		}
	}
	return map;
}
```

This function 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> result = new HashMap<String,String>();
	for(String key : originalMap.keySet()) {
		result.put(key + key, originalMap.get(key) + originalMap.get(key));
	}
	return result;
}
```

This function 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);
		}
		int currentFrequency = letterToFreq.get(startingChar) + 1;
		letterToFreq.put(startingChar, currentFrequency);
	}
	//ok now find the most frequent
	int highestFrequency = 0;
	char charWithHighest = '\0';
	for(char current : letterToFreq.keySet()) {
		if(letterToFreq.get(current) > highestFrequency) {
			highestFreq = letterToFreq.get(current);
			charWithHighest = current;
		}
	}
	return charWithHighest;
}
```

Array Intro

```
// 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.
int[][] myIntArray = new int[50][7]; //50x7 array
String[][] myStringArray = new String[10][200]; //10x200 array

// Set and get 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("largest indexes in 2d string array " + 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
//to get the second dimension, do it like this
System.out.println(myIntArray[0].length); //prints 7
// actually you could use any valid index instead of 0, but they'll
// all give you the same length
```

Map Intro

```
// A map is like a dictionary in Python
// It associates a key with a particular value - but because this is
// java they have to be typed
//
// Format: HashMap<KeyType,ValueType> foo = new HashMap<KeyType,ValueType>();
// e.g.
HashMap<String,Integer> namesToWeight = new HashMap<String,Integer>();

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

//note that putting twice with the same key overwrites the value
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
if(namesToWeight.containsKey("Steve")) {
      System.out.println("Steve is in the map!");
}

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