Lab 9 (October 9 or October 10)

Instructions: Complete the exercises below. Be sure to show your code to one of the lab TAs before you leave, so that you can receive credit for this lab. You must also upload a copy of all your source code (.java) files to the link on Blackboard by 11:59 PM on Tuesday, October 10.

1. Define a Java method with the following header:

```
public static void bubbleSort (double [ ] list)
```

This method applies *bubble sort* to arrange the elements of an array into ascending order. Note that this method does not return anything; it modifies the array parameter directly!

Bubble sort works by making several passes through an array. On each pass, successive neighboring pairs are compared. If a pair is in decreasing order (i.e., one element is larger than the following element), the values are swapped; otherwise, the values remain unchanged. The technique is called a bubble sort or sinking sort because the smaller values gradually "bubble" their way to the top and the larger values sink to the bottom. The algorithm can be described as follows:

```
boolean changed = true;
do {
    changed = false;
    for (int j = 0; j < list.length - 1; j++)
        if (list[j] > list[j + 1]) {
            // add code to swap list[j] with list[j + 1]
            changed = true;
        }
} while (changed);
```

Clearly, the list is in increasing order when the loop terminates. It is easy to show that the `do` loop executes at most (*list.length* - 1) times.

Write a Java program that creates a `double` array with the initial values {6.0, 4.4, 1.9, 2.9, 3.4, 2.9, 3.5} and calls `bubbleSort()` to re-order its elements. For clarity, add print statements to your `bubbleSort()` method to display the current arrangement of the array after every loop iteration.

2. Define a Java method named `valueCounter()` with the following header:

   ```
   public static void valueCounter (int [ ] values)
   ```

   This method takes an array of integers as its argument and prints the number of times that each unique value appears in the array. Your solution does not need to print the array contents in sorted order, but it ***MUST NOT*** print any values that do not exist in the array. For example, given the array

   ```
   [23, 12, 14, 12, 5]
   ```

   acceptable output would be:

   23 occurs 1 time(s).
   12 occurs 2 time(s).
   14 occurs 1 time(s).
   5 occurs 1 time(s).

   Note: You may assume that the array only contains integers in the range 0–100, inclusive.

   Finish this problem by writing a complete Java program that reads in (or generates via `Math.random()`) a series of integer values in the range 0–100, stores them in an array, and calls `valueCounter()` to determine and display the frequency of each unique value.

Grading Guidelines: This lab is graded on a scale of 0-3 points, assigned as follows:

0 points: Student is absent or does not appear to have completed any work for the lab

1 point: Student has written only one program, but it does not compile or run at all due to errors.

2 points: Student has written (or attempted to write) both programs, but only one compiles and runs without error.

3 points: Student has written both programs, and they both compile and run correctly, without any apparent errors.