This problem set is due Thursday, October 27 at 11:59pm, Korea Time. Note that the due date that you see on Blackboard is not accurate since it shows the time in EST. You should go by the due date in this handout.

- To solve each problem below, you will be implementing a class.
- Please carefully read and follow the directions exactly for each problem. Files and classes should be named exactly as directed in the problem (including capitalization!) as this will help with grading.
- You should create your programs using your preferred text-editor or the Eclipse text editor.
- Your programs should be formatted in a way thats readable. In other words, indent appropriately, use informative names for variables, etc. If you are uncertain about what is a readable style, see the examples from lectures and textbook as a starting point for a reasonable coding style.
- Your programs should compile and run without errors. Please do not submit programs that do not compile! Its better to submit partial implementation that compiles as opposed to complete implementation that does not compile. You may use either Eclipse or the command-line to compile and run your programs.
- Be sure to include your name and your email address as comments at the top of each file that you submit.

**What Java Features to Use**

For this assignment, you are not allowed to use more advanced features than what we have covered in Lecture 1 through Lecture 10 (Chapters 1 through 7 of our textbook).

**What to Submit**

Submit the following files on Blackboard. Please do not submit .class files or Eclipse-specific project files or any I did not ask for.

```
Arr.java
Arr2.java
```

**Problem 1 (80 points)**

Create a class named `Arr` in a file named `Arr.java` and introduce the following methods:

1. (7 pts) Define a method named `randomArray` with two formal parameters: one integer indicating the size of the array to be created and the other indicating the upper-limit for the range of random numbers to be generated. If the second number is 15, then it would mean that the random numbers will be in the range of 0 and 14 inclusive.
2. (3 pts) In your main call randomArray with two actual arguments: 100 as the size and 15 as the upper-limit, and store the returned array into a local variable of your choice in main.

3. (7 pts) Define a method, named arrSum with one formal parameter of type array of integers, which returns the sum of the elements in the array.

4. (2 pts) In your main print the average of the numbers in the array that you obtained in step 2 above. To compute the average, you must use arrSum that you defined earlier. Print the average to the standard output device, i.e., screen. When you generate output to the screen, add some annotation so that the output will be meaningful.

5. (6 pts) Define a method named contains with two formal parameters: one an array of integers and the other an integer. The method returns true if the second argument is contained in the first array argument; returns false otherwise.

6. (2 pts) In your main call contains with the array obtained in step 2 above and 8 as the second argument, and print the result to the standard output.

7. (7 pts) Define a method named contains2 with two formal parameters: one an array of integers and the other an integer. The method returns the index of the array where the first occurrence of the second argument is found, if found. If the second argument is not contained in the first array argument, it returns -1.

8. (2 pts) In your main call contains2 with the array obtained in step 2 above and 8 as the second argument, and print the result to the standard output device.

9. (10 pts) Define a method named countMultiplesOf with two parameters: one an array of integers and the other an integer. This method returns the count of the integers in the array that are multiples of the second parameter. For example, 8 is a multiple of 2, but not a multiple of 3, and zero is a multiple of any number. I suggest that you define and use an auxiliary function that tests if a number is a multiple of another and returns a boolean value.

10. (2 pts) In your main call countMultiplesOf with the array obtained in step 2 above and 8 as the second argument, and print the result to the standard output.

11. (20 pts) Define a method named buildHistogram with one parameter: an array of integers. It then returns an array of integers, which is a histogram. You will build the histogram as follows. If the array that you obtained in step 2 contains 5 occurrences of 6, then 5 will be stored as the value in the array index 6 of the histogram array. What should be the size of the histogram array? Well, you should find the largest element in the source array and use that information to decide the size of the histogram array. You should consider adding another method, say largest that finds the largest element of an array and returns.

12. (2 pts) In your main call buildHistogram with the array obtained in step 2 above. Store the histogram that is returned into a local variable of your choice in main.

13. (10 pts) Define a method named printHistogram with one formal parameter of type array of integers. Call this method from the main with the histogram obtained in the previous step and let this method print the histogram in the following format:

   i: <n>: <n of *'s for the index i of the array. n is the element in the index i>

So, if the array holds the following values:

```
+---+---+---+---+---+---+----+---+
| 5 | 1 | 4 | 6 | 0 | 2 | 10 | 6 |
+---+---+---+---+---+---+----+---+
 0 1 2 3 4 5 6 7
```
Then, the histogram printed would look like this:

```
0:  5:  *****
1:  1:  *
2:  4:  ****
3:  6:  *****
4:  0: 
5:  2:  **
6: 10:  **********
7:  6:  *****
```

Note that the distribution in this histogram will show you how good the random number generator is. That is, how random are those numbers in the array.

**Hints:** Try to solve this problem **incrementally.** That is, solve it one step at a time and make sure the current step works correctly before you move on to the next step. Using random numbers are convenient if you don’t want to generate arrays manually. However, testing your program with random numbers generated on the fly is very hard because you don’t know what numbers you will be getting. So, while you are developing your program, I suggest that you use a small array with known elements that you include in the array yourself. After you finish debugging your program, switch it to a random number array with the specified size before you hand it in. Of course, you should test your final version to your satisfaction before you actually hand it in. You will most likely want to write a function that prints the elements of an array and use it as a debugging tool as you develop your program.

Hand in your **Arr.java**.

### Problem 2 (30 points)

In a class called **Arr2** do the following:

1. (8 pts) Define a method called `isUnique` that accepts an array of integers as a parameter and returns `true` if all the numbers are unique (i.e., no duplicates) and `false` otherwise. For example, if the array `a1` stores `\[3, 8, 12, -48, 46, -3\]`, the call `isUnique(a1)` would return `true` and if the array `a2` stores `\[4, 7, 3, 9, 12, -47, 3\]`, the call `isUnique(a2)` would return `false` since the value `3` appears twice.

2. (2 pts) Add code in `main` to test your implementation of `isUnique` at least with the examples given in the previous step.

3. (8 pts) Define a method named `randomUniqueArray` with two formal parameters: one integer indicating the size of the array to be created and the other indicating the upper-limit for the range of random numbers to be generated. If the second number is 15, then it would mean that the random numbers will be in the range of 0 and 14 inclusive. However, the elements in the resulting array should all be unique. That is, there should be no duplicate elements. You are allowed to create only one array in the process.

4. (2 pts) Add code in `main` to test your implementation of `randomUniqueArray` with two actual arguments: 10 as the size and 15 as the upper-limit, and store the returned array into a local variable of your choice in `main` and print the elements of the array.

5. (8 pts) Define a method named `buildPassword` that takes no parameter and returns a valid password as a `String`. A password is valid if it passes all of the following rules:
   - A password must have at least eight characters but no more than 12.
   - A password must alternate between letters and digits starting with a letter.
   - The only letters allowed are vowels.
   - The only digits allowed are odd digits.

   The following examples are valid passwords: "a5a3e9o3", "e1o5u3a9e5u3". But, the following are not: "a5a3e9", "a5a3e8", "a5a3e6o3".
6. (2 pts) Add code in main to test your implementation of buildPassword by printing a password that was generated. Print at least three valid passwords.

Hand in your Arr2.java.