CSCI 1101 – 2017
Laboratory Report 7

Your task is to complete the assigned work using JGrasp or an IDE of your choosing. You may use your own computer, or one of the lab computers provided.

Your submission should be a **ZIP** file containing your source code files. You should submit your **ZIP file** on Brightspace:
http://dal.brightspace.com

Submission Deadlines (firm):
Monday Labs: due Wednesday by 12:00pm (noon)
Friday Labs: due Sunday by 12:00pm (noon)

NB:

- Try to submit this report *during* the lab so that your TA can check it for you before you submit!
- Attendance is mandatory in all labs, and will form part (10%) of your overall lab grade
- Acknowledge any help that you obtained from friends, TAs, the Learning Centre, etc., by adding comments to your code where appropriate. Obtaining help is fine, *so long as you acknowledge it!*
- Any students who cannot log on to the lab computers should speak to the Help Desk to set up their account.
- Textbooks, class handouts, and any other materials are welcomed and encouraged in all labs!
- Food and drink are not permitted in the computer labs
- Late labs are ***not*** accepted! It is known that computer errors, power outages, and network lag are 105% more likely to occur between 11:55-11:59am, in the moment they can do the most damage. Account for this, and give yourself the chance to make a timely submission!

Header Comments

Your code should now include header comments for **all** of your class (.java) files. The comment should include the lab/assignment number, the course (CSCI 1101), the name of your program and a short description of the entire class, the date, your name and Banner ID, and a declaration that matches the first page of this document (e.g., whether you received help). See the example below for what a header comment should include:

```
/*Lab1, Question 1 CSCI 1101
   Student.java holds information about a student at Dalhousie in CSCI1101 and
   their grades
   June 29, 2015
   John Smith B00112345
   This is entirely my own work. */

public class Student {
//rest of Code
```

If applicable, your demo class should then also have a similar header:

```
/*Lab1, Question 1  - demo class CSCI 1101
   StudentDemo.java is a demo program for the Student class. It creates student
   objects, and compares different students.
   June 29, 2015
   John Smith B00112345
   I received help with creating Student objects from my TA but the rest is my
   own work. */

public class StudentDemo {
//rest of Code
```

Exercise 1
(Please read Exercises 2 and 3 before you start Exercise 1, so that you have an idea of what you will need to know as you work through these simpler examples. This will let you ask better questions of your TA while you are still present in the lab!)

Write a program that reads in two set of positive Integers. One set goes into an ArrayList `int1` and second set goes into another ArrayList `int2.` Create a third arrayList of Integers that contains the shared Integers from `int1` and `int2` (i.e., the ones that are common to both ArrayLists). Order does not matter.

A sample dialog is shown below:

```
Enter the first set of integers on one line, end with -1
10 200 6 99 3 5 90 44 -1

Enter the second set of integers on one line, end with -1
200 56 34 3 5 87 44 5 1000 -1

Array List with shared Integers:
[200, 3, 5, 44]
```

Exercise 2: Write a program that reads words into two ArrayLists list1 and list2 and then creates a third ArrayList that contains words which are different between both list1 and list2 (i.e., that are not common to both ArrayLists). You may assume that the strings are entered by the user as words separated by a space on a single line and the end is signaled by the String “-1”. You can use the .next() method in class Scanner to read each word.

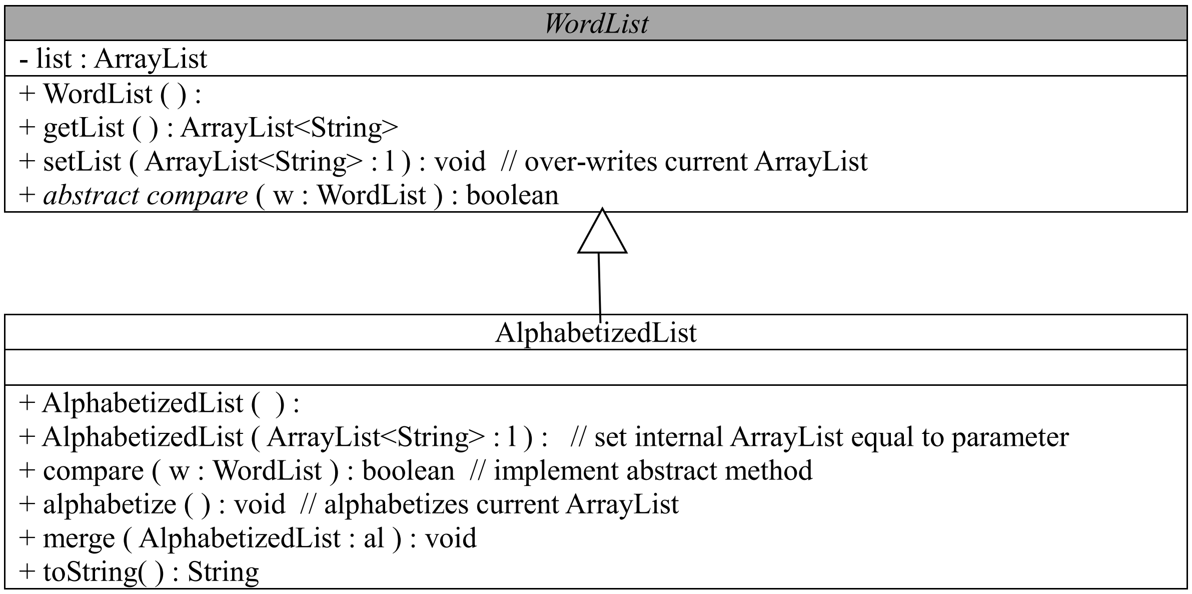
A sample dialog is shown below:

```
Enter words on one line, end with -1
java c pascal ada python java c++ -1

Enter words on one line, end with -1
c pascal lisp java lisp rust go -1

Array List with distinct strings:
[ada, python, c++, lisp, rust, go]
```

Exercise 3:
Implement the following UML diagram (remember that shading + italics here indicates an *abstract* class)



```
WordList
- list : ArrayList
+ WordList ( ) :
+ getList ( ) : ArrayList<String>
+ setList ( ArrayList<String> : l ) : void  // over-writes current ArrayList
+ abstract compare ( w : WordList ) : boolean

AlphabetizedList

+ AlphabetizedList (  ) :
+ AlphabetizedList ( ArrayList<String> : l ) :   // set internal ArrayList equal to parameter
+ compare ( w : WordList ) : boolean  // implement abstract method
+ alphabetize ( ) : void  // alphabetizes current ArrayList
+ merge ( AlphabetizedList : al ) : void
+ toString( ) : String
```

The basic idea here is to create a kind of "wrapper" class that will store an alphabetized list of words. Internally, the words are stored in an ArrayList<String>. The abstract class WordList defines several of the basic methods:

- the getList method should use a shallow copy to return the internal list
- the setList method should use a deep copy
- the compare method will be used for comparing two WordList objects

The class AlphabetizedList represents a list of words that has been sorted in alphabetical order (assume for this exercise that all words will be given using lower-case only). The underlying ArrayList for an AlphabetizedList should be in alphabetical order before it is used; this is an *invariant*. This class defines the following methods:

- there are two constructors, both of which should use calls to the super-class; the constructor with a single parameter should use a deep copy. After initialization, the internal list of words (if any) should be in alphabetical order
- the implemented compare method should always look at the first word in both WordLists, and return true if the first word for this AlphabetizedList comes (alphabetically) before the first word in the WordList parameter. So for instance, if
  ```
  list1 = ["apple", "pear", "plum"]
  list2 = ["aardvark", "llama", "koala"]
  list3 = ["zebra", "dog", "aardvark"]
  ```
  are the underlying ArrayLists for two WordList objects, then
  ```
  list1.compare(list2);
  ```
  should return false, since "apple" comes after "aardvark", but
  ```
  list2.compare(list3);
  ```
  should return true, since "aardvark" comes before "zebra".
- the alphabetize method actually ensures the underlying ArrayList is correctly ordered; after this method is run, the list of words is in alphabetical order. The order of two identical words does not matter.
- the merge method adds the elements of the passed parameter AlphabetizedList into the current AlphabetizedList

Include a small demo class that creates two new AlphabetizedList objects, populates them with words of your choosing (at least five in each), and merges them into a single AlphabetizedList. Use only lower-case letters in your input. Print the results after each AlphabetizedList is created as well as after the merge.

Exercise 3, Note 1:
In Java, alphabetical comparisons/sorting is simple. You can use the compareTo() method defined for String objects like so:

```
if(a.compareTo(b) < 0)
   System.out.println(a + " comes before " + b);
else if(a.compareTo(b) > 0)
   System.out.println(a + " comes after " + b);
else
   System.out.println(a + " is the same as " + b);
```

Exercise 3, Note 2:
The above is fine for comparing pairs of elements of an ArrayList<String>, but how should we ensure the entire list is sorted? This kind of sorting problem is common in Computer Science.

As a helpful step, consider a related question: say you are given a single ArrayList and asked to copy elements over individually to a second ArrayList, with the extra condition that each copied element should be placed in alphabetical order within the second ArrayList. How would you go about solving this?

Consider a simple example:

```
list1 = [Dog, Cat, Mop, Amp]
list2 = []
```

To begin with, the first ArrayList has four String elements and the second ArrayList is empty. When copying over elements, we will think about the *invariant* that the second ArrayList's elements should always be in alphabetical order.

The first element to be copied will be Dog, so where should this be placed in the second ArrayList in order to keep it in alphabetical order?

```
list2 = [Dog]
```

There really aren't any positions yet, so simply adding the Dog element means list2 is alphabetically sorted.

The next element to be copied will be Cat. This can be placed before or after Dog in the second list. By comparing the String 'Cat' to the String 'Dog', we can tell that only one of these positions will maintain our *invariant* that this list stays alphabetically sorted. Cat must come **before** Dog.

```
list2 = [Cat, Dog]
```

Next, we copy the Mop element. This could be placed in one of three positions: before Cat, between Cat and Dog, or after Dog.

```
list2 = [ ??, Cat, ?? , Dog, ?? ]
```

We have up to two comparisons to make. First, comparing Mop and Cat tells us that Mop comes **after** Cat. So we perform the second comparison as well, which tells us that Mop comes **after** Dog. By performing these (sequential) comparisons, we have eliminated the first two positions as options, and are left with placing Mop after Dog in order to maintain our *invariant* that the list stays alphabetically sorted.

```
list2 = [Cat, Dog, Mop]
```

Finally, we must copy the Amp element. This will require up to three comparisons (one with each sorted element already in list2) to let us decide between four positions.

```
list2 = [ ??, Cat, ??, Dog, ??, Mop, ?? ]
```

First, we compare Amp with Cat. This tells us that Amp comes **before** Cat, and no more comparisons are needed; Amp must come before Cat, so it cannot be placed in any of the places after.

```
list2 = [Amp, Cat, Dog, Mop]
```

By sequentially comparing elements, we are able to create a copy of an ArrayList<String> that is guaranteed to be in alphabetical (that is, sorted) order. A slight modification of this “sorting by insertion” approach will also work for the original question that asks for an alphabetically sorted ArrayList to be created by the `alphabetize()` method. You may adapt the “sorting by insertion” either by using a temporary ArrayList variable to help build the alphabetized list, or by thinking carefully about how to handle indices and working with only a single ArrayList. And of course, you may also use your own approach that does not rely on this example.