CSCI 2110 Data Structures and Algorithms
Assignment 3
Release Date: Oct 16th
Due Date Nov 1st
23h55 (5 minutes to midnight)

Introduction

This assignment will help you develop an understanding of ordered lists and basic recursion.

Marking Scheme

Exercise 1 /25

- Working code, Outputs included, Efficient, Good basic comments included: 25/25
- No comments or poor commenting: subtract up to 3 points
- Unnecessarily inefficient: subtract up to 4 points
- No outputs and/or not all test cases covered: subtract up to 2 points
- Code not working: subtract up to 14 points depending upon how many classes and methods are incorrect.
- Assignment submission and test case text files are not well labeled and organized: **subtract up to 2 points.**

Exercise 2 /12

- Working code, Outputs included, Efficient, Good basic comments included 12/12
- No comments or poor commenting: subtract up to 2 points
- Unnecessarily inefficient: subtract up to 1 point
- No outputs and/or not all test cases covered: subtract up to 2 points
- Code not working: subtract up to 5 points depending upon how many classes and methods are incorrect.
- Assignment submission and test case text files are not well labeled and organized: **subtract up to 2 points.**

Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.

Submission: All submissions are through Brightspace. Log on dal.ca/brightspace using your Dal NetId.

Java Versions: Please ensure your code runs with Java 11.

What to submit:
Submit one ZIP file containing all source code (files with .java suffixes) and a text documents containing sample outputs. For each exercise you will minimally have to submit a demo Class with a main method to be tested and your test code, sample outputs.

Your final submission should include the following files:*OrderedList.java, list1.txt ,list2.txt, OrderedListDemo.java, merged.txt, RecursionExercises.java, Exercise2out.txt*

You MUST SUBMIT .java files that are readable by your TAs. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers. Your code must compile. Failure to compile may result in a mark of 0.

You may add extra helper methods as needed as long as you adhere to the functionality and specifications of this document.

Exercise 0 (no marks): Ordered Lists

Examine the java file *OrderedList.java* and its methods. Create a test file with a main method, and practice the following:

- inserting items to a new OrderedList object
- use the enumerate method to print the list
- using the binarySearch method for items in the list and items not in the list
- using the first() method, the next() method, and how they relate to the cursor variable

You do not need to submit anything for this exercise.

Exercise 1: Two-finger Walking Algorithm

This exercise focuses on the two-finger walking algorithm for ordered lists. You will write a program called *OrderedListDemo*.java that can read a list of names from a file and construct an ordered list. You will be provided with the following starter files: *OrderedList.java, list1.txt ,list2.txt.* Examine these files and study the methods before proceeding.

Create a demo class called *OrderedListDemo.java*

- In the main method, read in the values from *list1.txt* and *list2.txt*
    - Your program should **accept input from a user** (specifying the names of **two** input files) and read files formatted like the one described (see also: list1.txt and list2.txt). Your program should read two text files, each containing a list of names, and construct two ordered lists.
    - Store the lists in two separate OrderedList objects called *names1* and *names2*

Input/Output

If the input text file is:

```
Shai
Tom
Jim
Aaron
Barbara
Beth
Fred
Jack
Jarred
Jill
Amar
Ralph
Jill
```

Hillary

It should create the following ordered list and display the contents of the list:

[Aaron, Amar, Barbara, Beth, Fred, Hillary, Jack, Jarred, Jill, Jim, Ralph, Shai, Tom]

Note: Although Jill is repeated in the input list, the ordered list does not have repeated items.

.
Finally, your program should demonstrate a method that returns a third list that is a merger of the two ordered lists. Use the two-finger walking algorithm discussed in class. Use the following method header:

- *public static <T extends Comparable<T>> OrderedList<T> merge(OrderedList<T> list1, OrderedList<T> list2)*

If list1 is:

Amar	Boris	Charlie	Dan	Fujian	Inder	Travis

And list2 is:

Alex	Ben	Betty	Charlie	Dan	Pei	Travis	Zola	Zulu

Your method should create and return the new merged ordered list:

Alex	Amar	Ben	Betty	Boris	Charlie	Dan	Fujian	Inder	Pei	Travis	Zola	Zulu

Student Generated Test Cases:

Test your program and print outputs from a sample run in the following format to a file called *merged.txt:*

Enter the first filename to read from: list1.txt
Enter the second filename to read from: list2.txt
The Ordered Lists contain the following entries:
List 1:
[Amar, Boris, Charlie, Dan, Fujian, Inder, Travis]
List 2:
[Alex, Ben, Betty, Charlie, Dan, Pei, Travis, Zola, Zulu]

A merged version of the two lists looks like this:
[Alex, Amar, Ben, Betty, Boris, Charlie, Dan, Fujian, Inder, Pei, Travis, Zola,
Zulu]

Exercise 2: Recursion

This exercise focuses on recursion. You will implement two static methods to solve two separate problems. All solutions must be recursive.

Create a class called *RecursionExercises.java*. This class will contain the static methods and the main method to run your code.
Create the following static methods:

- Write a recursive method call *squares* which accepts a single positive integer and calculates the sum of the squares of all the digits up to and including the integer. Use the following signature:
    - public static int squares(int n)
    - For example, squares(5) would return 55
- Write a recursive method called *reverseString* which accepts a string and an integer. The method returns the passed in string in reverse order starting from a specified index. Use the following header:
    - public static String reverseString(String n, int i)
    - For example, reverseString("Hello",0) would return "olleH"
    - reverseString("Hello",1) would return "lleH" since we start from index 1.

Input/Output
Input: No user input for this exercise.

Output:

- For the *squares* method call the method and print the following:
    - *The sum of the squares up to [n] is: [answer]*
    - Example: *The sum of the squares up to 5 is: 55*
- For the *reverseString* method call the method and print the following:
    - *Starting word is [word], reversed from index [i] is [answer]*
    - Example: *Starting word is Hello, reversed from index 2 is leH.*

Student Generated Test Cases:
Demonstrate that your recursive methods work by calling them in the main method of the *RecursionExercises.java* class.

- Use print statements for the demo, no scanner is needed.
- Show 2 different examples for *squares* and 2 examples for *reverseString* with different index values.

Test your program and print outputs from a sample run in the following format to a file called *Exercise2out.txt.*