CSIS 10B **Assignment 2**

Read: Chapters 5, 10 and 12

Choose and complete any 10 points from the problems below, which are all included in the download file on the website. Use BlueJ or Eclipse to complete the assignment, then export JAR file and upload to the server using your pass code. You may do more than 10 points of work but the max award will be 11 points.

Using Stacks, Queues, and Lists

1) **(4 points) Coding Bat Array Problems**

   Log in to coding bat and solve the following Array-2 problems (these problems all require loops to solve)
   countEvens, only14, shiftLeft, sum13
   Copy each of your solutions from Coding Bat into the CodingBat class in the Assignment2 project folder.

2) **(3 points)** A *palindrome* is a string of characters (a word, phrase, or sentence) that is the same regardless of whether you read it forward or backward—assuming that you ignore spaces, punctuation, and case. For example, *Race car* is a palindrome. So is *A man, a plan, a canal: Panama*. Write a Java program that uses an ArrayStack (a Stack based on an Array) and a LinkedQueue (both holding Character data) to test whether an input string is a palindrome.

   The program should read a whole line of text from the keyboard (use the Scanner nextLine method) into a String variable. Then, with a for loop, use the String charAt method to extract each letter one at a time from your string and if it is an alphabetical letter (i.e. not a space or punctuation mark), convert it to lowercase and add it to your Stack and Queue. Then with another loop use the basic stack and queue operations to determine if the string is a palindrome.

   Suggestions: 1) Use the Character.isLetter method to determine if a character is an alphabetical letter.
   2) you can either convert the entire string to lowercase or use the Character.toLowerCase method to convert each letter to lower case.

   For reference, here are the UML diagrams for Stack and Queue:

ArrayStack<T>
+ArrayStack<T> +push(newEntry: T): void +pop(): T +peek(): T +isEmpty(): boolean +clear(): void +toString(): String

LinkedQueue<T>
+LinkedQueue<T> +enqueue(newEntry: T): void +dequeue(): T +getFront(): T +isEmpty(): boolean +toString(): String

3) **(3 points)** The **Student100.txt** file in the download project folder has about 100 student records inside it. Write a program that reads the entire file into a LList<Student>. There are several ways you can read the file into the LList, but the recommended approach is to modify the Student.read method so that it returns a boolean using the exception handling technique we developed in Lab 4B Slide Show, and then put the read statement into the condition for a while loop so that when the end of file is reached, the loop automatically

terminates. Another option is to use the overloaded Student constructor that takes a Scanner parameter and builds the Student from data read from the file. This is demonstrated in the StudentReadFile program in the FileDemo project folder which is included in the download for assignment 2. If you come up with a different way to read the file, that's fine too as long as it works.

After you've successfully read the file, ask the user to input a cutoff GPA value. Go through and remove all of the Students who fall below the cutoff GPA, and print the resulting list of remaining students.

(Optional: 1 point) Lists are great because you can easily insert items anywhere. A reasonably simple way to loosely sort the students in your list by last initial would be to go through the list, remove all students whose last name begins with 'A', and insert them at the end of the list. Then repeat with students whose last name begins with 'B', 'C', 'D', etc. At the end of this process, all the A-name students will appear at the beginning, then B-name next, and so on, with Z-name students at the end. This would involve two nested loops. The outer loop would change the target letter, the inner loop would do the list processing. You can use a String to store the last name initials in the order in which you want to process them, such as

String targetLetter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

so the outer loop would just sample the next targetLetter based on its loop counter, for example,
char target = targetLetter.charAt(k) where k is the loop counter for the outer loop.

For reference, here are the UML diagrams for LList and Student

LList<T>
+LList<T>() +add(newEntry: T): void +add(newPosition: int, newEntry: T): boolean +remove(givenPosition: int): T +clear(): void +replace(givenPosition: int, newEntry: T): boolean +getEntry(givenPosition: int): T +contains(anEntry: T): boolean +getLength(): int +isEmpty(): boolean +toArray(): T[] +toString(): String

Student
+Student() +Student(first: String, last: String, ID: int, GPA: double) +Student(in: Scanner) +clone(): Student +equals(other: Object)boolean +toString() String +getFirstName(): String +getLastName(): String +getID(): int +getGPA(): double +setFirstName(first: String): void +setLastName(last: String): void +setID(ID: int): void +setGPA(GPA: double): void +read(in: Scanner): void

4) Do either a) for 3 points or b) for 4 points

Consider the following algorithm to sort the entries in a stack S1. First create two empty stacks, S2 and S3. At any given time, stack S2 will hold the entries in sorted order, with the smallest at the top of the stack. Move the top entry of S1 to S2. Pop and consider the top entry *t* of S1. Pop entries of stack S2 and push them onto stack S3 until you reach the correct place to put *t*. Then push *t* onto S2. Next move all the entries from S3 to S2.

a. Write an iterative implementation of this algorithm.

b. Consider the following revision of this algorithm. After moving the top entry of S1 to S2, compare the new top entry *t* of S1 with the top entry of S2 and the top entry of S3. Then either move entries from S2 to S3 or from S3 to S2 until you locate the correct position for *t*. Push *t* onto S2. Continue until S1 is empty. Finally, move any entries remaining in S3 to S2. Implement this revised algorithm.