CS145 – PROGRAMMING ASSIGNMENT #5

PART B

CARD LINKED LIST

OVERVIEW

This program primarily focuses on the implementation of a LinkedList version of the CardList that you did in part 1.

INSTRUCTIONS

Your deliverable will be to turn in three files. The files will be named `Card.java`, `CardArrayList.java` and finally `CardLinkedList.java`. *We won't need premium cards for this part of the assignment.*

For this assignment, any use of any built in List, Set or Map data structure will result in no credit. Note that other data structures are acceptable. I am aware that this assignment could be done quite simply by the implementation of an existing structure, the point of the assignment is to do it without the standard implementation to get a feel for how they work "under the hood".

Also as this is part B of the assignment, you really should only have to create new one file, the other two files should be copied over from the first part.

COLLECTABLE CARD

The primary goal of this assignment will be the implementation and use of a custom LinkedList.

We will be using the same `Card` class that we used for the first part. So refer to your notes that object implementation.

PROGRAM DESCRIPTION

In this assignment you will construct a LinkedList that can manage a list of Cards in the linked List approach that we have discussed. You will implement the various methods that are required in order to make the LinkedList function.

A sample program will be provided for you to test with, but you may want to alter it to check different functionality of your LinkedList.

YOUR INSTRUCTIONS

You will write the following classes, implementing the necessary methods as appropriate.

NECESSARY METHODS – CARD CLASS

The Card class will be exactly the same as in part A, so check your notes there for the implementation.

NECESSARY METHODS – CARD ARRAYLIST CLASS

The `CardArrayList` class will exactly the same as in part A, so check your requirements there. It is only used for speed testing/comparison in the main code.

NECESSARY METHODS – CARDLINKEDLIST CLASS

The `CardLinkedList` is the primary focus of this assignment. In it you will maintain a list of cards, allowing for all the methods listed below to manage a list of cards.

The basic idea of the `CardLinkedList` will be to keep track of an internal linked list of nodes containing cards as the data type.

Should the user try to add a value to the List, then you will need to add new nodes to the list in the appropriate place, maintaining the structure of the list and the size of the list at all times.

However, we want to be able to change the type of list in the future, so your `CardLinkedList` will need to implement the `CardList` interface in order to work. So while the `CardList` interface is provided for you, you will need to make sure that your program properly implements it.

REQUIRED METHODS

The linked list that you are creating must implement the same CardList interface that you used in the first part of this assignment, so you will need to implement all the necessary methods to make that work.

However in order to help you, below are some notes on some topics to help out.

PUBLIC CARDLINKEDLIST ()

In the `CardLinkedList ()` constructor should create an initial null pointing head pointer.

PUBLIC STRING TOSTRING ()

The `toString()` method should print the current values of the list being stored. It should surround the entire list inside [0 | |*size*] with commas between the values. A sample output might look like.

```
[0 | [2,3,4::55],[8,9,10::90],[4,5,6::82] | 3]
```

Note the 4 to show the current size. If the array is empty it should print out:

```
[0 | 0]
```

Note that this looks different than the ArrayList version.

PUBLIC VOID SORT ()

The `sort()` method should simply sort the array from smallest to largest. However I want you to implement your own version of the merge sort using a helper method. (See below)

This is actually easier with Linked Lists, since pulling from the front is easier. Think about the following pseudo-Code when trying to implement it.

PRIVATE VOID SORT (CARDLINKEDLIST X)

The `sort(X)` method should simply sort the array from smallest to largest using a merge sort implementation.

- Sort(CLL)
    - If the CLL is size 1 or less, be done.
    - Make two empty CardLinkedLists (left/right)
        - *Note, use YOUR linked list*
            - *CardLinkedList left;*

        - *CardLinkedList right.*
    - Find the size of ½ the input list current list as an int.
    - Use a for loop to fill the left with ½ the data, while removing the data from the input list.
        - Pull the data from the front of the list. *(remember, remove also returns)*
    - Use a while loop to fill the right with the remainder of the input list until it is empty.
        - Pull the data from the front of the list. *(remember, remove also returns)*
    - Sort the left
    - Sort the right.
    - Merge left + right into the original list.
- Merge
    - While (size of left) + (size of right) > 0
        - If the left is empty remove from right and add to main list.
        - If the right is empty remove from the left and add to main list
        - If the item in the front of left is smaller than the item in front of right, add from left
        - If none of the above, add from right.

Note that unlike the array version, you will destroy the original list when splitting the original list into left and right, but rebuild it in merge.

PUBLIC VOID SHUFFLE()

The `shuffle()` method should shuffle the array into a non-ordered arrangement. One way of doing this is picking two random numbers within the size of the list, and then swapping those two values. Then repeat this process a bunch of times. *(For example five times the number of elements in the list).*

PRIVATE VOID SWAP(INT A, INT B)

A good idea for your class is to create a private `swap(a,b)` method that will swap two cards around as necessary. Helpful for methods above *(shuffle)*.

Note that this method used ints as inputs. So you might just find a pointer to the a'th element, a pointer to the b'th element and then move the DATA from a into b and b into a. Don't try to rearrange the nodes. Just move the data.

PUBLIC VOID REVERSE()

You might want to use a Stack here. Move everything to the stack, then remove everything from the stack back to the list.

NOTES

Chapter 16 contains a number of ideas on how to implement an LinkedList of numbers, so do not be afraid to consult the chapter for hints/ideas.

The testing program is divided up into stages. Comment out the stages that you haven’t finished and work on things step by step.

STYLE GUIDELINES AND GRADING:

Part of your grade will come from appropriately utilizing object methods.

Your class may have other methods besides those specified, but any other methods you add should be private.

You should follow good general style guidelines such as: making fields private and avoiding unnecessary fields; declaring collection variables using interface types; appropriately using control structures like loops and if/else; properly using indentation, good variable names and types; and not having any lines of code longer than 100 characters.

Comment your code descriptively in your own words at the top of your class, each method, and on complex sections of your code. Comments should explain each method's behavior, parameters, return, and exceptions.