Data Structures and Algorithms Spring '23 section1 Dr. Hristescu

Lab3

Assigned: Thursday, February 2

Due: 10:00pm Monday, February 6

For this and all future labs:

Starting with this lab, guideline 9 from General guidelines to follow when developing code for this course will be strictly enforced. Menu cases should be handled by calling case specific methods that are passed the locally declared collection (in main) as a parameter. This means that all driver methods need to have as one of the parameters the collection. The collection has to be declared locally in main and passed to any driver method that needs to work with it.

Problem1:

Use the following code from the textbook:

- ListInterface for the ADT list
- ListReferenceBased //textbook code modified to work with redesigned Node (below)
- ListIndexOutOfBoundsException
- Node //textbook code modified to retain encapsulation

Using the provided code for ListReferenceBased, build your own class called **MyListReferenceBased** that implements the ListInterface and

- has **just one** data field called **head** (i.e. **remove** the data field **numItems** from MyListReferenceBased)
- has an additional **toString** method (you are NOT allowed to use the **get** or **find** method because it would be inefficient; instead traverse the linked structure and gather the information from the item data field of each node using the same frame as the size method)
- you cannot call find in the implementation of size

After incorporating these changes NO NOT make any other changes to MyListReferenceBased.

Note: **You cannot use the provided code AS IS, you need to adapt it to work with just the head data field. Make sure that every operation leaves the list object in a CONSISTENT state at ALL TIMES.**

Problem2:

Develop and test a menu-driven application program that works with input of type String with the following options. Note that option 6 and 7 are part of the application and **NOT** part of the functionality of the MyListReferenceBased class! You can assume unique input data. Optionally (not required for this lab) you can check for uniqueness of the Strings when inserting.

0. Exit program
1. Insert item into the list
2. Remove item from the list
3. Get item from the list
4. Clear the list
5. Display size and content of the list
6. Delete the smallest and largest item in the list
7. Reverse the list

A sample run of the program on this input data can be found here.

Important:

- Option 6 requires deleting the smallest and largest item(string) **in lexicographic order (**using **compareTo)**. You **have to** implement it using the following frame with the following 2 methods:
  - Design a **void returning** driver method called **findIndexLargeAndSmall** that has 2 parameters, the first one is the **collection** and the second one is an **array of primitive integer type with 2 entries**. The method should efficiently (**just one pass**) scan the collection and **efficiently** update the first array entry of the second parameter with the **index of the smallest item** in the collection (in lexicographic order) and the second array entry with the **index of the largest item**. If they do not exist, use -1 as the special value.
  - Design a **void returning** driver method called **displayAndDeleteLargeAndSmall** that first calls **findIndexLargeAndSmall** and uses the information in the updated (2-entry) array to delete the items in the specified entries in the collection. Make sure your method can handle a -1 value for the

updated array entries correctly and doesn't throw exceptions. Make sure the method also displays the **values** from the indices found that were deleted (see sample run).

- You can use the application frame from Lab2 and adapt it for this specific application.
- **displayAndDeleteLargeAndSmall** should **allocate the array** and **pass** it to **findIndexLargeAndSmall,** while **findIndexLargeAndSmall** should **only update the array passed as parameter.**

Problem3:

Investigate what the String class' **compareTo** method return value is. Use as input different characters, upper and lower case letters, digits, different characters on the keyboard, strings of various length and content. Ex: What info does the return value of **compareTo** provide when used on: "a" and "A", "t" and "T", "a" and "z", "a" and "1", "1" and "A", "1" and "9", "1" and "18", "21" and "81", "to" and "top", "%" and "0", "and **many others** that you come up with. Feel free to explore other ways to investigate the returned value.

Attach your:

- **sample runs and**
- **writeup with the findings as part of the lab**.

Problem4:

Lab3Conclusions: Summarize in a few sentences what you have learned from working on this lab.

ExtraCredit:

Add an *iterator* method that uses an inner class, called **LRBIterator,** that implements **Iterator**.

Submit: Follow the instructions to turn in the lab electronically using **Lab3** for the name of your directory.
Problem1: MyListReferenceBased.java
Problem2: code for application program and sample runs (including on provided input data)
Problem3: Code and sample run, writeup about the return value of compareTo.
Problem4: Conclusions