Project 3 – Lists Due April 18th, 2017

Project Description

Your goal is to implement code, documentation and testing for a variety of java lists including: arraylist, linkedlist, double linked list, JUnit tests, in-line comments and a driver class. Extra credit can be obtained by sorting the list structures in the following ways: sorted array list, sorted linked list and sorted doubly linked list. Skeleton code for the lists project is attached to project. Your project should be named cs1302.lists and you should store your source code on nike in the following directory src/test/java/cs1302/lists. You should include .java source files for: Driver.java, List.java, DoubleLinkedList.java, ArrayList.java and LinkedList.java. If you created the extra credit then the following java source files should be included: SortedList.java, SortedArrayList.java, SortedDoubleLinkedList.java and SortedLinkedList.java.

Project Tasks

Before submitting your project, you need to review the skeleton code and perform the following:

1. (10 points) Implement ArrayList.java from scratch according to the List interface that is provided. This list implementation will be backed by a _regular Java array_. You are not allowed to use any of the other data structures that are included with Java when implementing this class.

2. (20 points) Implement LinkedList.java from scratch according to the List interface that is provided. This list implementation will use a singly linked list. You are not allowed to use any of the other data structures that are included with Java when implementing this class.

3. (10 points) Implement DoubleLinkedList.java according to the List interface that is provided. This list implementation uses a doubly linked list. You are not allowed to use any of the other data structures that are included with Java when implementing this class. In order to make things easier, you may extend your LinkedList class when writing the code for DoubleLinkedList.

4. (10 points) Create JUnit tests for your list classes in the src/test/java/cs1302/lists directory. Do not cheat yourself by creating "soft" tests. These tests will be beneficial in helping you debug your code.

5. (10 points) Ensure that your code is properly documented using in-line comments as necessary. In general, you should describe in regular terms what your code is doing. Please note that you do not need to write JavaDoc comments for the methods that implement an interface as they will inherit the comments from the interface. However, if you create any new methods or classes then they will need to be properly documented using JavaDoc comments and tags.

6. (40 points) Experiment with your list implementations by creating a Driver class that compares the actual running times of each of the 3 list implementation when prepending, appending, and searching for 1000, 10000, 100000, and 1000000 random elements with each of the 3 lists. Create and generate a report that compares your results. Include in the report an analysis of the time complexity of your implementations. You should include the growth functions (i.e., t(n)) for the prepend, append, get,

remove, and search methods in each of your three implementations as well as the worst-case time complexity of each method expressed in Big-O notation. When providing the growth functions and time complexities, you will need to justify and explain why the functions and time complexities you provided work based on your code. Your code does not have to produce the report, but it does have to produce the statistics used in your report. You may submit the report summary portion of the project in one of the following formats: word or pdf. One Example of a table of stats that could be included in your pdf or word doc is the following table depicting the millisecond runtime for each operation in an Arraylist, followed by an evaluation of the millisecond run time stats.

Arraylist Records	prepend	append	get	remove	search
1000					
10000					
100000					
1000000					
10000000					

This table could be created for each of the 3 list structures and sorted versions of the lists.

7. Update the README in your project directory to contain the following information at the top of the file, updating it with your own information:

Extra Credit Project Tasks

You may earn extra credit for each of the tasks listed below:

1. (+5 points extra credit) Implement SortedArrayList.java from scratch according to the SortedList interface that is provided in the List class. This list implementation will use a _regular Java array_. You are not allowed to use any of the other data structures that are included with Java when implementing this class. In order to make things easier, you may extend your ArrayList class when writing the code for SortedArrayList. You must also include SortedArrayList in your report in order to receive this extra credit.

2. (+5 points extra credit) Implement SortedLinkedList.java from scratch according to the SortedList interface that is provided in the List class. This list implementation will use a singly linked list. You are not allowed to use any of the other data structures that are included with Java when implementing this class. In order to make things easier, you may extend your LinkedList class when writing the code for SortedLinkedList. You must also include SortedLinkedList in your report in order to receive this extra credit.

3. (+5 points extra credit) Implement SortedDoubleLinkedList.java from scratch according to the SortedList interface that is provided in the List class. This list implementation will use a doubly linked list. You are not allowed to use any of the other data structures that are included with Java when implementing this class. In order to make things easier, you may extend your DoubleLinkedList class

when writing the code for SortedDoubleLinkedList. You must also include SortedLinkedList in your report in order to receive this extra credit.

Linked Lists

In computer science, a linked list is a data structure consisting of a group of objects called nodes which together represent a sequence. Under the simplest form, each node is composed of a value and a reference (in other words, a link) to the next node in the sequence. We call this data structure a singly linked list. In general, the last node in the sequence should point, in its next reference, to the first node.

A slightly more complex variant adds some additional links. When each node is composed of a value and a reference to both the previous and next nodes in the sequence, we call the data structure a doubly linked list.

The way these data structures are usually implemented according to the following general guidelines:

1. Create a node class that includes a place to store the value of the element that is to be stored in the node as well as the appropriate number of reference variables to store the links (next for a singly linked list and both previous and next for a doubly linked list).

2. In the list class, have a reference variable called head that points to the first node in the sequence. When the size of the list is 0, this reference is null. When the first element is added to the list, a node object is created, the element is stored in the node, and the head is then set to reference that node.

3. Whenever an element is added to the list, a node object is created, the element is stored in the node, and the node is added to the sequence in the appropriate place by adjusting the links.

4. Whenever an element is removed from the list, the links of the nearby nodes are altered so that the node that contains the element is no longer in the sequence.

5. Searching is performed by traversing the links.

Resources

The UML class diagram has been attached to the project to provide an overview of the skeleton code objects. Note: the generic type parameter in the proposed Node<T> class may need to extend Comparable<T>. Development should be on nike because this is where your project will be run and tested. If any changes are made to the project description or skeleton code, they will be announced in class.

Directory Structure and Packages

All of the **non-test classes** for this project should be contained in the src/main/java/cs1302/lists directory. These classes are in the cs1302.lists package.

All of the **JUnit test classes** for this project should be contained in the src/test/java/cs1302/lists directory. These classes are also contained in the cs1302.lists package so that you do not need to do any imports to test your own code.

Submission Instructions

CRN 26245 TWR 9:30-10:45

submit project3 cs1302a

CRN 26311 TWR 3:30-4:45

submit project3 cs1302b

Questions: If you have any questions, please email them to Piazza or vlawson3@uga.edu