CMPT 270.3 Intersession 2016

Assignment #1

TTD of an OO implementation of a linked list.

In this assignment you are to use TDD to implement a linked list data structure class. You will write a java program lListTester.java and a class lList. The class should be developed by TDD methods. To do this you will need to take snapshots of your code with each new test to demonstrate how you went from a skeleton lLink class with no implementation to a full working implementation. The snippets of code can be cut and pasted into a single word doc. Each test you create will need to be documented by demonstrating the test code run, the implementation code generated and the test results (copy of the console output). The binary tree example used in class would look like the following for the very first test:

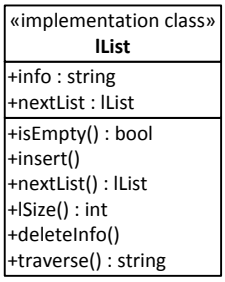
1. Create empty tree and test that it returns empty state when requested:

```
// test of empty tree reports not empty incorrectly
tempbTree testBtree = new tempbTree();
if (!(testBtree.isEmpty())) {
   // generate exception
   throw new Exception("Empty tree incorrectly reported not empty");
}
}

// tbtree function tested
public boolean isEmpty() {
      return treeEmpty;
   }
```

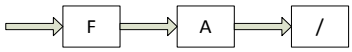
<remaining tests and their implementation would follow>
If the test results in some output, show that as well.

Here is the specification for the lLink class.

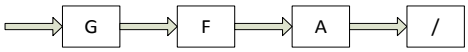


The behaviors are as follows:
isEmpty() return true if lList is empty, i.e. info is null and nextList is null.

insert(inInfo) returns nothing. Place inInfo into lList as a new lList at the beginning of the current lList. Given the following lList:



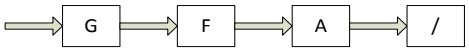
insert("G") would produce:



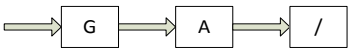
nextList() returns lList. If the current list is not empty, it returns a reference to the beginning of the remaining list. If the current list is empty it returns null.
If the current list references "G" then this.nextList() would return a reference to "F". If the current list is empty (/) then this.nextList() would return null.

lSize() returns an integer which counts the number of none-empty sublists in the current list + 1. If this references "G" then this.lSize() would return 3. If this references "F" then this.lSize() would return 2/ If this references / then this.lSize() would return 0.

deleteInfo(outInfo) returns a reference to the start of the list following the information removed if it exists. If the linked list is

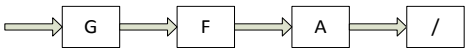


then deleteInfo("F") would result in the following linked list:



deleteInfo("Z") would change nothing.

traverse() returns a string which is the current natural order of the linked list as a comma separated set of strings. Given the linked list,



traverse() would return "G, F, A".

All methods should be recursive if possible (this will simplify implementation immensely). Make as much use of object oriented concepts as you can. If you must add extra methods or properties, explain why. Test extra methods as you would any given required methods.

Marking scheme:
10 TTD documentation of your approach to implementing your class methods.
10 Test application that incrementally builds the test set as you implement the class.
5 Pre and post conditions for each method as applicable.

I use a scale for marking where 1..10 means anything under 5 was not acceptable; 7 is expected; anything above 7 is superlative.
On a scale of 1..5 3 is average, 4 great, 5 is fantastic. You should never get a 2 or less.

HAVE FUN!