Lab 8: Singly-Linked Lists

Testing & Inserting After

Objective

This lab will complete the exploration of singly-linked lists. In this lab you will: 1) test your implementation of the constructor, append, and prepend, and 2) implement insertAfter, and search.

Test Singly-Linked List Implementation From Last Week

To test your Singly-Linked List implementation from last week:

1. Start with the code in your CSLibrary repo from last week.
2. Following in the directions given in class, create a lab-8 branch in your CSLibrary repo and check out the code in the lab-8 branch in IntelliJ.
3. Add the package cs.cscollections to the tests directory in your code.
4. Download the file SinglyLinkedListTests.zip from the lab assignment on Moodle.
5. Unzip the zip archive somewhere on your computer *outside of your CSLibrary project directory*. Copy the files from expanded archive into the cs.cscollections package you have created in the *tests* directory.
6. Run the tests in the new cs.cscollections package by right clicking on the cs.collections package and selecting the menu option: *Run tests in cs.cscollections*.
7. Commit the new code to your repo and push it back to the *lab-8 branch.*

Your implementation of the SinglyLinkedList class should pass the tests for creation, append, and prepend. It will fail the tests for insertAfter and search.

Note: *You will have to have the getLength and isEmpty methods implemented for any of the tests to complete successfully.*

Implement the insertAfter Method

Follow the process illustrated in lab lecture by the instructor to implement the insertAfter method. Start by implement two private utility methods with the following signatures:

```
private void insertAfter(Node<E> curNode, Node<E> newNode)

private Node<E> findNode(E target)
```

The insertAfter private method should follow the algorithm illustrated in Section 12.3 of your textbook. Note that unlike the *insertAfter* method in the *List* interface, which works for objects of the parameter type E, this version of insertAfter works on Node objects. The findNode method will require you to traverse the list using a *Node<E> reference* named *curNode*. The documentation for these two methods are given in Figures 1 and 2, respectively.

```
/**
     * This utility method inserts a newNode after the specified node,
     * curNode.
     *
     * Note 1:  It is required that newNode not be null.
     * Note 2:  curNode can only be null if the list is empty.  This
     *          indicates that first is not in the list, which
     *          violates the semantics of the method. No insertion is made.
     * Note 3:  This method does not increment size.  The instance variable
     *          size should always be incremented after calling this method.
     *
     * @param curNode The node in the list to insert the newNode after.
     * @param newNode The node to insert in the list.
     */
     private void insertAfter(Node<E> curNode, Node<E> newNode) { … }
```

Figure 1: Documentation for private insertAfter method

```
/**
     * The utility method findNode finds and returns a reference to the first
     * node in the list containing the data target.  If a node containing
     * target is not found, null is returned.  The search is made on equal
     * values for the data objects and not identity.
     *
     * Note: It is required that the type E implement the Comparable<T>
     *       interface.
     *
     * @param target The target object to search the list for the first
     *               occurrence.
     * @throws ClassCastException This exception is thrown if the class
     *                            instantiated for E does not implement the
     *                            Comparable<T> interface.
     */
     private Node<E> findNode(E target) { … }
```

Figure 2: Documentation for private findNode method

Implementation Notes for findNode:

Note 1: The findNode method can be done in 3 steps:

1. Set a search reference to the head of the list.
2. If the list is not empty, use a while loop to search the list for the specified element.

3. Return the search reference value

Note 2: Assuming you call your search reference *curNode*, the test to compare the value of the items is:

```
((Comparable<E>)(curNode.getData())).compareTo(target) != 0
```

Note 3: Note that you must check if *curNode* has run off the end of the list in the guard of a while loop before you make the comparison from Note 2.

Implement the public version of *insertAfter* using the utility methods insertAfter and findNode you have completed. Test your work by rerunning the SinglyLinkedListTests. Your SinglyLinkedList class should now pass the CreationTests, AppendTests, PrependTests, and InsertAfterTests. If it doesn't, find and correct the problem.

Finishing up

When you have your Singly-Link List class created and tested, commit your code to your local repo and then push your code back the *lab-8 branch* of your CSLibrary repo.

Bonus

If you have time, implement the search method. Hint: The findNode private method performs a basic search. Use it to implement the search method. The SinglyLinkedListTests contains tests for search. If you do the Bonus, make sure the new code is pushed back to the *lab-8* branch of your repo.