CS136 Lab Section 1

Logan Behnke and Joshua Pollock

18 April 2017

**Lab 9 – Doubly Linked List**

**Problem Statement:**

In this lab, we are to create a linked list that is capable of traversing both forward and backwards between the nodes. After creating this, we will also create a test for the Doubly Linked List class. We will need to verify that all tests past, that we are not repeating any code, and that we are not storing any data inefficiently.

**Constraints:**

1. DoublyLinkedList must be able to travel forwards and backwards between nodes
2. A test file must be created that tests the DoublyLinkedList class
3. The DoublyLinkedList class will implement List and extend AbstractSequenceList.
4. The tests we must implement are
   1. You must be able to retrieve an iterator object with listIterator(int)
   2. You must be able to add to an empty list using an iterator
   3. You must be able to read the first element of a list using an iterator
   4. You must be able to move forward and backwards through your list using an iterator
   5. You must be able to add arbitrary elements using an iterator
   6. You must be able to read arbitrary elements using an iterator
   7. You must be able to remove elements using an iterator
   8. You must be able to replace a value in the list using an iterator
   9. You must be able to determine if you are at the beginning or end of a list using an iterator
   10. You must be able to determine your current index using an iterator
   11. You must be able to retrieve iterator objects at a non-zero index from listIterator(int)
   12. You must be able to correctly retrieve the size of the list after adding and removing elements correctly
   13. You must be able to use addAll(Collection) to add every item in an existing collection.
   14. You must be able to read from the list by index
   15. You must be able to remove from the list by index
   16. You must be able to add to the list by index
   17. You must be able to replace an element by index
   18. Trying to iterate past the end of the list, or previous to the beginning should throw a NoSuchElementException

**Assumptions:**

* JUnit will not be needed to implement the tests
* A simple test class that can show list and iterator manipulation
* We can use the ListIterator from CS-136 lecture

**Features:**

* DoublyLinkedList
  + addFirst(Object t)
  + removeFirst()
  + getFirst()
  + addLast(Object t)
  + removeLast()
  + getLast()
  + getSize()
* Node
  + Node(Node previousNode,Object data,Node nextNode)
* DoublyLinkedListIterator implements ListIterator
  + Next()
  + hasNext()
  + previous()
  + hasPrevious()
  + add(Object t)
  + remove()
  + set(Object t)

**Planning:**

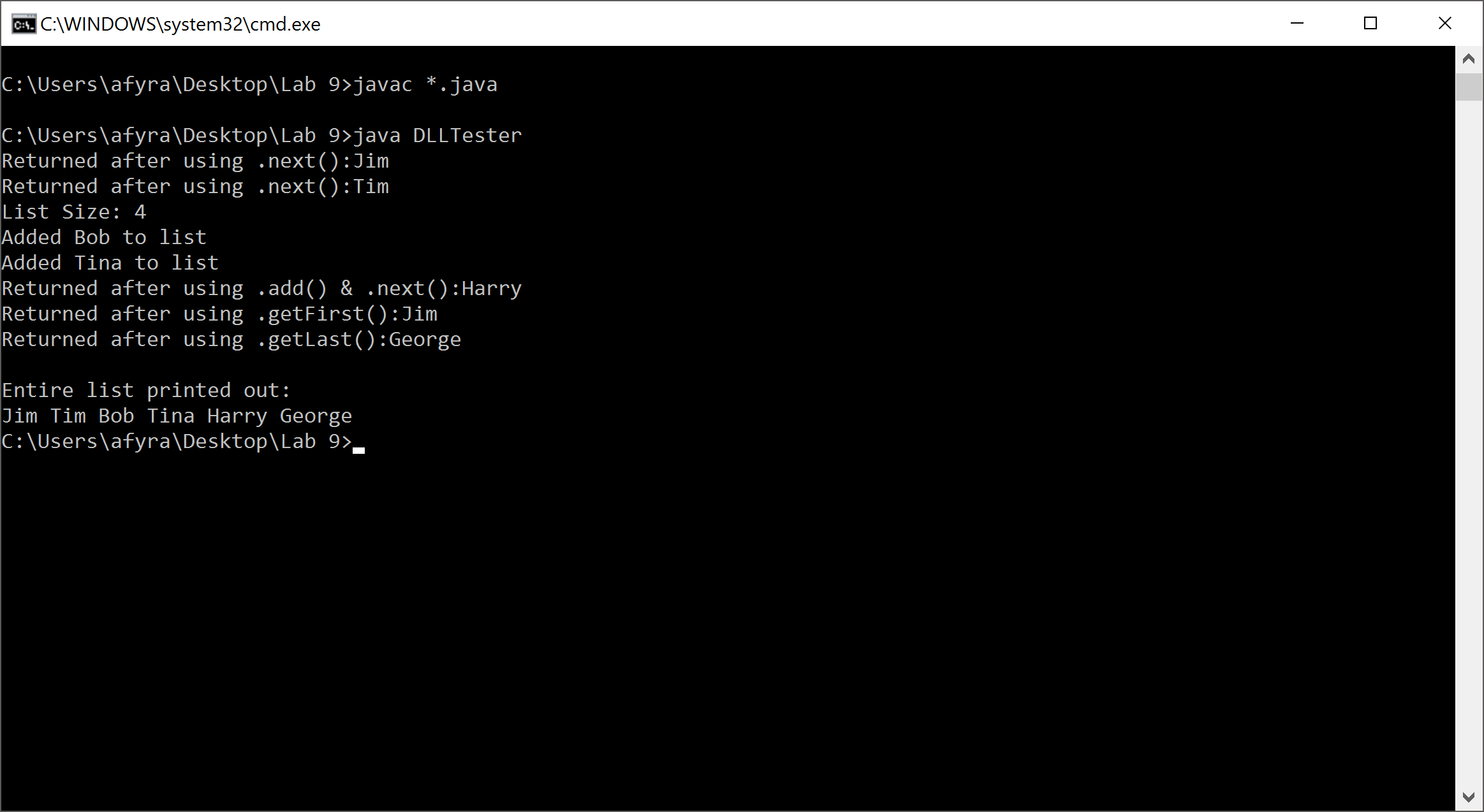
To start off this lab, we were provided a skeleton code. While the skeleton code was quite helpful for getting the DoublyLinkedList file set up, it did not provide much help in the way of filling in the methods for the Doubly Linked List. The next helpful thing we used was the LinkedList class file for CS136. This class contained almost everything we needed except the previous functionality.

With only the previous functionality needing to be implemented, we felt comfortable enough to begin writing code. To help us in this process we drew what the next and previous functions would do on paper.

**Implementation:**

­­­­ Implementing the code seemed quite easy at first. On the first lab session, we hammered through as much as we could and only have a few functions in the DoublyLinkedListIterator that still contained code stubs. A flaw we found with this is that when we started working on is again in the second lab session, the lack of comments had us confused on what all the code we had already implemented did. This took some time to go back and figure out what exactly was going on, but we did figure it out.   
  
 In the second lab session, a lot of work got done. We had implemented almost all the code but still had a few functions with stubs in it. The two main things that still needed to be edited was the previous and next functions. Collaborating with each other through Github, we finally implemented all the functions of the lab.

**Running Application:**



**Reflection & Refactor:**

This lab was easily one of the most difficult labs in the class so far. Even though it was a two-week lab, we had barely been able to get it done in time. There are many functions that still do not work properly and cannot be tested. There was not much to go off for this lab. The lab report did not have that much in terms of steps to guide us through the lab. The skeleton code that was provided did help to set up the file but after that it did not prove to be that helpful. The most helpful bit of code was the Linked List code from the 136 lecture. If we had more time we would like to finish troubleshooting each function that is not working properly. These functions are previous and remove functions. The rest of the functions seem to work fine as they are now.

One valuable thing we learned from this lab is to comment extensively. Without writing comments, we could not tell what we were doing when we came back to the lab. If we had proper comments, we would have been able to start right away and not focus on figuring out what the code does. Overall, we are not that satisfied with our code, as some of it does not function properly, but we do not see any way to fix and refactor it within the given time frame of the lab.