Lab 9 : Doubly Linked List

Clayton Williams Emalina Bidari

April 18th 2017

Tuesday 11:10 lab

**Problem Statement**

In this lab we will implement a linked list capable of traversing both forward and backwards between nodes. By utilizing an abstract base classes, we can define a fully featured Java list implementation by providing a handful of central methods. In addition we will continue to practice unit testing in which we write our linked list code to the requirements defined in our tests.

• write generic classes;

• write custom collections;

• utilize abstract classes for code reuse;

• work effectively with a partner using pair-programming;

• write an effective report that describes the students’ problem solving process.

**Planning**

In our planning we need to account that our linked list will be abstract. We’re given a skeleton of a doubly linked list with all the required methods that need to be tested. We also need to create a tester class to ensure that all our methods are working as intended.

Our methods that need to be implemented are as follows:

• Be able to retrieve an iterator object with listIterator(int)

• Be able to add to an empty list using an iterator

• Be able to read the first element of a list using an iterator

• Be able to move forward and backwards through your list using an iterator

• Be able to add arbitrary elements using an iterator

• Be able to read arbitrary elements using an iterator

• Be able to remove elements using an iterator

• Be able to replace a value in the list using an iterator

• Be able to determine if you are at the beginning or end of a list using an iterator

• Be able to determine your current index using an iterator

• Be able to retrieve iterator objects at a non-zero index from listIterator(int)

• Be able to correctly retrieve the size of the list after adding and removing elements correctly

• Be able to use addAll(Collection) to add every item in an existing collection.

• Be able to read from the list by index

• Be able to remove from the list by index

• Be able to add to the list by index

• Be able to replace an element by index

• Trying to iterate past the end of the list, or previous to the beginning should throw a NoSuchElementException

**Implementation and Testing**

Testing shows that the ‘add’ method and the ‘next’ method do not work as it throws a null pointer exception. Looking through the code that we had written, it would seem that this is because we forgot a case to deal with, but we cannot find one. The addition of the case dealing with adding a node if there is only one in the list does not deal with the NullPointerException, so we are stumped.

**Reflection**

What made this lab challenging was being able to visualize which node needed to be linked to each other. Drawing and visualizing what was happening with each node in methods such as add(), remove() made things much easier.