# Lab Report 02

Nathan Bickel

## Problem

The task was creating a linked list class that could store data of type double. Each node was to contain a double as well as two links, one link pointing to the node before it (or to null if it was the first element), and one link pointing to the node after it (or to null if it was the last element).

## Solution

There were a number of methods that I made.

- The constructor is simply the default constructor. I could have let the compiler create it but I wanted to be explicit.
- gotoNext() moves current one link forward (unless current is null).
- gotoPrev() moves current one link backward (unless current is null).
- reset() moves current to the front.
- gotoEnd() moves current to the end.
- hasMore() returns whether the current is not null, which signifies that no more data can be accessed (moving from either head to tail or tail to head)
- getCurrent() returns the data at current.
- setCurrent(double data) allows current to be changed to data determined by the method's parameter.
- add(double data) adds data to the end of the list. If the list is empty, head, tail, and current are set to a node containing the parameter data. Otherwise, the tail is the last element of the list, so once data is added, tail is temporarily the second to last element in the list. Thus, the method creates a node pointing back to the tail and forward to null with the parameter data. Then, it assigns tail's link forward to this new node. Finally, it adjusts tail so that it now refers to this new node.
- addAfterCurrent(double data) adds data after current. If current is null, the method does nothing. If current is the tail, the add method is called. Otherwise, a number of adjustments need to be made. First, a new node is created with the parameter data pointing forward to the element after current and back to current. Then, current is adjusted to point forward to the element. Finally, the element after the new element is adjusted to point back to the element.

- remove(double data) searches through the list from head to tail and removes the first instance found of the parameter data. For each element in the list, the method checks if the data equals the parameter data, and if it does, it removes it and stops running. It does this by going back an element and adjusting its forward pointer to skip over the element being removed, and then going forward an element from the one being removed and adjusting its backward pointer to skip over the element being removed.
- removeCurrent() removes the element and current and moves current one element forward (or one element backward if the element is the last one in the list). It does this in the same way remove works—adjusting both adjacent elements' pointers to skip over the element being removed.
- print() simply iterates through each node and prints its data.
- contains(double data) returns a boolean representing whether or not the list contains an element with the parameter data. It does this by iterating through each element until a match is found (or until the end of the list if one isn't found).

## Implementation Problems Encountered

I encountered several problems while writing the class. I initially added a check in the gotoNext() and gotoPrev() methods that wouldn't allow current to be moved if it would lead to a null reference. However, I realized this was throwing off the test code, and more importantly was unnecessary—current should be allowed to go to the end of the list if necessary, and this allows the programmer to use hasMore() to check if they are at the end of the list.

I also initially forgot to adjust both links in the add and remove methods—I adjusted the link going forward, but had to come back later and adjust the link going backward.

There were also several instances where I didn't initially consider the edge case of the element in question being the head or tail, so I had to go back in and adjust for those later.

Finally, I realized after encountering problems that I wasn't adjusting current to a different element when removing, so the node was still in the list and not eligible for garbage collection until current was manually moved away from the node to be removed.

## Lab Report Questions

1. When an object is unreachable because no reference points to it, it is considered eligible for garbage collection. The JVM will check for objects like this, and deallocate them in memory so that they are not needlessly taking up RAM.
2. The main advantage of a doubly linked list over a singly linked list is that one can progress through them in both directions. One can iterate through in both directions, and it is also possible to switch directions an arbitrary amount of times throughout a passthrough. This is not possible in a singly linked list—if one wants to go backward two elements, they must iterate through almost the entirety of the list including a reset to get to that element. However, the main advantage of a singly linked list over a doubly linked list is that less memory for each element is required. Instead of having two links, each element only has one, which saves a significant amount of memory if the list is long. Additionally, many methods are less complex in a singly linked list, because only one set of links need to be adjusted rather than two.