# Lab 12: Linked List: A tutorial

## Current Code

In lecture we demonstrated code for the implementation of linked data structures, such as LinkedList1, DLinkedList, and RLinkedList. We are going to write our own linked data structure. This structure requires two Classes:

**LinkedList -** The linked list class and the Class which is the element on the list. This is a public class, but it not executable (no main method);

**ListElement -** The element that is contained in the list. We are implementing this as a private class, which means that it is added at the bottom of the linked list class source code file, outside of the class braces.

We also establish a third class, to test our implementation: **ListDriver.**

Download the code bundle: **Lab12.zip** and install these files into an IntelliJ project in preparation for this lab. These files contains the bare bones class of the linked list Implementation (LinkedList and ListElement). The methods in blue are the ones we will implement in lab today.

| LinkedList |
|---|
| - label: String<br>- firstElement: ListElement |
| + LinkedList(String)<br>+ getLabel: String<br>+ getFirstElement: ListElement<br>+ setLabel( String )<br>+ setFirstElement( ListElement )<br>+ toString: String<br><br>+ isEmpty: boolean<br><br>+ addElement(String): boolean<br><br>+ isInList( String ):boolean<br>+ addElementAtHead(String): boolean<br><br>+ popElement: String<br>+ pushElement ( String )<br>+ removeElement(int): boolean<br>+ addElement(String, int): boolean<br>+ sortList(): boolean |

| ListElement |
|---|
| - itemText: String<br>- nextElement: ListElement |
| + ListElement(String)<br>+ getItemText: String<br>+ getNextElement: ListElement<br>+ setItemText( String )<br>+ setNextElement( ListElement )<br>+ toString: String<br><br>+ hasNext; boolean<br><br>+ compareTo( ListElement ): int |

Compile and run this code. You should see the following console display

```
Unknown Song
        If there's a bustling in the hedgerow, don't be alarmend now.
        It's just spring cleaning for the may queen.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.
```

The algorithm for the method **isEmpty** is very simple:

**Algorithm: isEmpty**

Input: none

1. If the first item is null
2. Return true
3. otherwise,
4. return false

The algorithm for the method **toString** shows the "recursive" nature of the data structure:

**Algorithm: toString**

Input: none

1. If the first item is null
2. Return "list is empty"
3. otherwise,
4. return the text of the first item plus the toString method from the nextItem of the first item.

You will see that we added a few formatting symbols to make the print out neat.

Now we will examine the algorithm for the **AddElement.** We will assume that this means to add the new element to the end of the list. This is not always the case, but for this exercise, we will make this assumption.

**Algorithm: addElement**

Input: string

1. Create a new ListElement with the string as text
2. If the list is empty, make the new node the firstElement of list
3. Otherwise, traverse the list to find the last element with next Element being null
4. When found, set the new node to this last node, return true
5. if not found return false

Step 5 of the algorithm was not applied. **How could it be applied? What are the circumstances that would generate a false (the last node not being found). How could this be avoided.**

## Implementing the remaining methods.

Now we will examine the algorithm for the i**sInList**. For this exercise, this method will return true if the String that is the parameter if the same as the string in the ListElement. Otherwise, it will return false.

**Algorithm: isInList**

Input: string

1. Starting with the firstElement, try each element to see if it equals the parameter string
2. If the parameter String is equal to the current list element string
3. return true;
4. If not equal string found, return false.

Implement this method. When complete, test it by uncommenting the code following **Test1**, which is

```
System.out.println("\n\n\"Performed by ????????\"  is in list:"+
            lyrics.isInList("Performed by ????????")+"\n\n");
```

You should see the following result. Notice that the result is “false”

```
Unknown Song
        If there's a bustling in the hedgerow, don't be alarmend now.
        It's just spring cleaning for the may queen.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.


"Performed by ????????"  is in list:false
```

---

Now we will examine the algorithm for the **length**. This method will the number of elements in the list.

**Algorithm: length**

Input: none

1. Initialize the counter to zero;
2. If the list is empty
3. Return counter
4. Otherwise
5. Traverse the list, increment the counter each time
6. return counter;

Implement this method. When complete, test it by uncommenting the code following **Test2**, which is

```
System.out.println("There are "+lyrics.length()+" items in the list "
      +lyrics.getLabel()+"\n\n");
```

You should see the following print out.

```
Unknown Song
        If there's a bustling in the hedgerow, don't be alarmend now.
        It's just spring cleaning for the may queen.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.


"Performed by ????????"  is in list:false


There are 4 items in the list Unknown Song
```

---

Now we will examine the algorithm for the **addElementAtHead.** For this exercise, this method adds the new element at the head of the list, or in place of the first element. The first element is made the next Item of the new first element. True is returned when successfully added. This method is also know as push, when dealing with stack.

**Algorithm: addElementAtHead**

Input: string

1. Create a new node with the new text
2. Assign the first Item the List the next Item of the new node.
3. Assign the new node as the first Item of the list
4. Return true.

Implement this method. When complete, test it by uncommenting the code following **Test3**, which is

```
System.out.println("\n\n\"Performed by ????????\"  is in list:"+
        lyrics.isInList("Performed by ????????"));
lyrics.addElementAtHead("Performed by ????????");
System.out.println("\n\n\"Performed by ????????\"  is in list:"+
        lyrics.isInList("Performed by ????????"));
System.out.println("\n"+lyrics.toString());
```

```
"Performed by ????????"  is in list:false
"Performed by ????????"  is in list:true

Unknown Song
        Performed by ????????
        If there's a bustling in the hedgerow, don't be alarmend now.
        It's just spring cleaning for the may queen.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.
```

You will see that we search for a line of text not yet in the list and it returns false. Then we add this at the head of the list, and then retest for it. It is now in the list. Then when we print out the list, we see that the new phrase is the list item in the list. If you have any question or if you code is not working correctly, ask your instructor. Many of the remaining exercises can depend on these methods.

---

Now we will examine the algorithm for the **pop.** This method is common terminology when using a stack. It refers to the process of getting the first element in the list and removing it from the list. This is usually to process the value the item holds. So, for this exercise, this method will return the text of the first Item on the list and remove that item from the list. Note: just because it returns a string, does not mean that string must be used. In essence, this method could be used to delete the first item in the list.

**Algorithm: pop**

Input: none

1. If the list is empty
2. Return null
3. Assign the text of first Item in the List to a holding variable.
4. Assign the nextItem of the firstItem as firstItem
5. Return the holding variable.

Implement this method. When complete, test it by uncommenting the code following **Test4**, which is

```
System.out.println("\"Performed by ????????\"  is in list:"
            +lyrics.isInList("Performed by ????????"));
lyrics.pop();
lyrics.addElementAtHead("Performed by LedZepplin");
System.out.println("\"Performed by ????????\"  is in list:"
            +lyrics.isInList("Performed by ????????")+"");
System.out.println("\"Performed by LedZepplin\"  is in list:"
            +lyrics.isInList("Performed by ????????")+"\n");
```

---

The result should be as below. From the last test, the phrase “Performed by ????????" should be in the list, then it is popped out and when tested, it is no longer in the list. Then the phrase “Performed by LedZepplin” is added to the head of the list and the whole list printed out.

```
"Performed by ????????"  is in list:true
"Performed by ????????"  is in list:false
"Performed by LedZepplin"  is in list:false

Unknown Song
        Performed by LedZepplin
        If there's a bustling in the hedgerow, don't be alarmend now.
        It's just spring cleaning for the may queen.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.
```

---

Now we will examine the algorithm for the method **remove.** This method will find the element in the indicated position and remove it from the list. Note: If you need to use the element before it is removed, you have to get it before calling this method.

**Algorithm: remove**

Input: index number

1. If the list is empty
2. Return false
3. if index is greater than the length of the list less 1
4. Return false
5. initialize a counter to 0
6. Assign the firstItem as the currentitem;
7. Traverse the list until you have the node that is located in the position one less than the index, incrementing the currentIndex by one each time
8. Retrieve the item that is at the currentIndex (This item is the one before the item you want to delete)
9. if there is a list item after the one to be deleted
10. assign the nextItem of the current node the nextItem of the curreentItems’s nextItem.
11. return true
12. Otherwise
13. assign the nextItem of the current node to null
14. return true

Implement this method. When complete, test it by uncommenting the code following **Test5**, which is

```
lyrics.remove(2);
System.out.println("\n"+lyrics.toString()+"\n\n");
```

You should see line 2: “It's just spring cleaning for the may queen.” Is no longer in the list.

```
Unknown Song
        Performed by LedZepplin
        If there's a bustling in the hedgerow, don't be alarmend now.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.
```

---

Now we will examine the algorithm for the method **addElement** at a specific location in the list.This method will find the position in the list required and will insert the new list item. All list items after the new insertion will be located one unit further from the head of the list.

**Algorithm: addElement**

Input: text and index number

1. If the list is empty and indexNumber is not 0
2. Return false
3. if index is greater than the length of the list less 1
4. Return false
5. Create a new listItem for the String
6. initialize a counter to 0
7. Assign the firstItem as the currentitem;
8. Traverse the list until you have the node that is located in the position one less than the index, incrementing the currentIndex by one each time
9. Retrieve the item that is at the currentIndex (This item is the one before the item you want to delete)
10. if there is a list item after the currentnode
11. assign the nextItem of the new node the nextItem of the currentnodes's nextItem.
12. assign the newNode to the nextItem of the currentNode
13. return true
14. Otherwise
15. assign the newItem to the nextItem of the current node
16. return true

Implement this method. When complete, test it by uncommenting the code following **Test6**, which is

```
lyrics.addElement("It's just spring cleaning for the may queen. ", 2);
System.out.println("\n"+lyrics.toString()+"\n\n");
```

You should see that the line has been inserted back to it's position in the list.

```
Unknown Song
        Performed by LedZepplin
        If there's a bustling in the hedgerow, don't be alarmend now.
        It's just spring cleaning for the may queen.
        Yes, there are two paths you can go by but in the long run.
        There's still time to change the road you're on.
```

**For extra credit ( 5 pointw),** write out the algorithm for and implement the following method:

```
public static boolean DeleteAllThatContain(String phrase)
```

This will require the use of a loop. Why do you think that is needed?