C:\Users\Rich\Documents\NetBeansProjects\Lab05\src\StinglyLinkedList.java

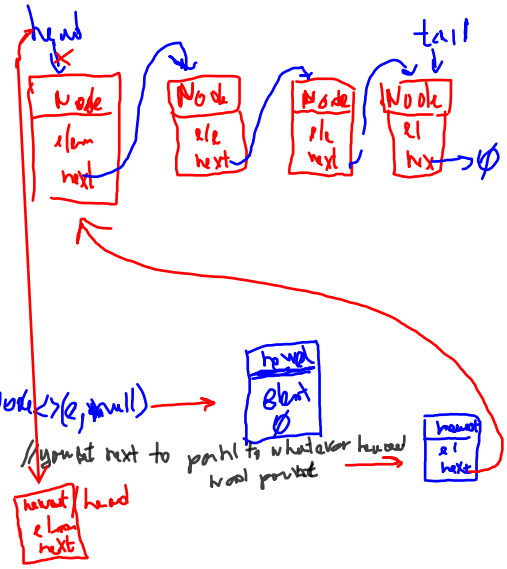
```
1
2 /**
3  * @author Rich
4  * @version 03/16/2017
5  * This class consist of method for creations of nodes, and methods on how to manipulate nodes
6  * data structure.
7  */
8 public class StinglyLinkedList<E> {
9    private static class Node<E>{
10       private E element;
11       private Node<E> next;
12
13       public Node(E e, Node<E> n)
14       {
15          element = e;
16          next = n;
17       }
18       /**
19        *
20        * @return  the element in the node.
21        */
22       public E getElement()
23       {
24          return element;
25       }
26       /**
27        *
28        * @return the pointer to the next node of the list.
29        */
30       public Node<E> getNext()
31       {
32          return next;
33       }
34       // setters
35       /**
36        *
37        * @param newNext set the pointer to point to the next item
38        */
39       public void setNext(Node<E> newNext)
40       {
41          next = newNext;
42       }
43
44    }
45    //  continuing of the StinglyLinkedList
46
47    private Node<E> head = null;    // point to 1st node of the list
48    private Node<E>  tail = null;   // pointer to the last node of the list
49    private int size = 0;
50
51
52    public StinglyLinkedList(){}   // construcst an initially empty list.
53
54    /**
55     *
56     * @return  how many nodes in the list
57     */
58    public int size()
59    {
```

so whenever you create a new node
you have use newNode.setNext(node u want it to point to)

head, tail, size = 0
0
cause it's a default constructor

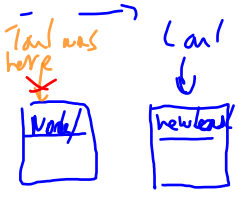
```
60      return size;
61    }
62   /**
63    *
64    * @return true if list is empty.
65    */
66   public boolean isEmpty()
67   {
68      return size==0;
69   }
70   /**
71    *
72    * @return  the first element in the first node
73    */
74   public E first()
75   {
76      if (isEmpty())
77         return null;
78      return head.getElement();
79   }
80   /**
81    *
82    * @return the last element in the list
83    */
84   public E last()
85   {
86      if(isEmpty()) return null;
87      return tail.getElement();
88   }
89   /**
90    *
91    * @param e generic element to be place as first element
92    */
93   public void addFirst(E e)
94   {
95      head = new Node(e,head);
96      if (size == 0)
97         tail = head;
98      size++;
99   }
100  /**
101   *
102   * @param e generic type element to set as last
103   */
104  public void addLast(E e)
105  {
106     Node<E> newest = new Node(e, null);
107     if(isEmpty())
108     {
109        head = newest;
110     }
111     else
112        tail.setNext(newest);
113
114     tail = newest;      // update tail,
115     size++;
116  }
117  /**
118   *
119   * @return the element in the 1st node removed
120   */
121  public E removeFirst()
```

unnecessary but if it was just head.getElement() then list is empty you'd have null.getElement() which is a null pointer exception



Alternative method

```
Node<E> newest = new Node<>(e, null)
newest.setNext(head)   // point next to point to whatever head was pointing
head = newest;
if (size == 0)
    tail = head
```

pictorally speaking advance tail to point to the newestLast

```
122   {
123     if (isEmpty())
124         return null;
125     E answer = head.getElement();
126     head = head.getNext();
127     size--;
128     if(size == 0)
129         tail = null;
130     return answer;
131   }
132
133   // my removeLast
134   /**
135    *
136    * @return  the removed item
137    */
138   public E removeLast()
139   {
140     if(isEmpty()) return null;
141
142     E answer = tail.getElement();
143
144     Node<E> current = head, previous = head;
145
146     while(current.getNext() != null)
147     {
148         previous = current; // the one before to last node/tail.
149         current = current.getNext();
150     }
151     // after exiting while loop current holds the memRef of tail, it is pointing to tail. current =tail
152     previous.setNext(null); // break the chain btw the one- before last and last node.
153     tail = previous;
154
155     return answer;
156   }
157   /**
158    *
159    * @return String format of the object
160    */
161   @Override
162   public String toString()
163   {
164      String listElements = "";
165
166     Node<E> current = head;
167     while(current != null)
168     {
169         listElements += current.getElement() +"-->";
170         current = current.getNext(); // update current to point to next node in the listt.
171     }
172     return listElements;
173   }
174  /**
175   *
176   * @param o object ref
177   * @return true if two linked list are equal
178   */
179  public boolean equals(Object o)
180  {
181     if(!(o instanceof StinglyLinkedList))
182         return false;
183     StinglyLinkedList l = (StinglyLinkedList) o;
```

```
184     if (size != l.size())
185         return false;
186     Node<E> sourceCurrentNodePtr = head; // current node pointer/Refvar for the "blueprint" list.
187     Node<E> targetCurrentNodePtr = l.head; // identifiers that points to current head of the list we're testing for equality.
188     while(sourceCurrentNodePtr != null)
189     {
190       if(!sourceCurrentNodePtr.getElement().equals(targetCurrentNodePtr.getElement()))
191           return false;
192       sourceCurrentNodePtr = sourceCurrentNodePtr.getNext(); // updtate memory pointer.; advancing current to next Node.
193       targetCurrentNodePtr = targetCurrentNodePtr.getNext();
194     } //end of while loop
195     return true;
196   }
197 }
```