

```
1  /*****
2  * Dalton Nofs
3  * Login ID: nofs5491
4  * CS-102, Summer 2017
5  * Programming Assignment 2
6  * LinkedList class: node object for linkedLists
7  *****/
8  public class LinkedList<T> implements ListInterface<T>
9  {
10     Node<T> head; // Head of the linked list
11
12     /*****
13     * Method: LinkedList()
14     * Purpose: default constructor for linkedList obj
15     *
16     * Parameters:          N/A
17     * Returns: void:       N/A
18     *****/
19     public LinkedList()
20     {
21         head = null;
22     }
23
24     /*****
25     * Method: isEmpty()
26     * Purpose: check to see if linkedList is empty
27     *
28     * Parameters:          N/A
29     * Returns: boolean:    if list is empty
30     *****/
31     public boolean isEmpty()
32     {
33         return (head == null);
34     }
35
36     /*****
37     * Method: size()
38     * Purpose: determine the size of linked list
39     *
40     * Parameters:          N/A
41     * Returns: int:        the size of the array
42     *****/
43     public int size()
44     {
45         Node<T> current = head; // counter node started at head
46         int counter = 0; // counter for size calc
47         while(current != null)
48         {
49             current = current.getNext(); // get next node
50             counter++;
51         }
52         return counter; // return the size of linkedList
53     }
54
55     /*****
56     * Method: get()
57     * Purpose: get object from linked list at index
58     *
59     * Parameters: int:      index
60     * Returns: T:          Object stored in index
61     *****/
62     public T get(int index) throws IndexOutOfBoundsException
63     {
64         Node<T> current = head; // set current to starting point
65         Node<T> previous = null; // holder for previous node
66         // walk array to find index
67         while((current != null) && (index != 0))
```

```
68     {
69         index--;
70         previous = current;
71         current = current.getNext();
72     }
73     // index is not in array
74     if (index != 0)
75         throw new IndexOutOfBoundsException();
76     return current.getData(); // return the data found
77 }
78
79 /*****
80  * Method: getNode()      !!! Private !!!
81  * Purpose: get Node from linked list at index
82  *
83  * Parameters: int:      index
84  * Returns: Node:      Node stored in index
85  *****/
86 private Node<T> getNode(int index) throws IndexOutOfBoundsException
87 {
88     Node<T> current = head; // set current to starting point
89     Node<T> previous = null; // holder for previous node
90     // walk array to find index
91     while((current != null) && (index != 0))
92     {
93         index--;
94         previous = current;
95         current = current.getNext();
96     }
97     // index is not in array
98     if (index != 0)
99         throw new IndexOutOfBoundsException();
100    return current; // return the node found
101 }
102
103 /*****
104  * Method: add()
105  * Purpose: add a object at specified index
106  *
107  * Notes: calls func that can throw indexoutboundsexception
108  *
109  * Parameters:
110  *             int:      index
111  *             Object:    Object to be placed
112  *
113  * Returns: void:      N/A
114  *****/
115 public void add(int index, T data)
116 {
117     // get the correct node
118     Node<T> current = this.getNode(index);
119
120     // create the new splice node
121     Node<T> splice = new Node<T>();
122     splice.setData(data);
123     splice.setNext(current);
124     if(current == null)
125         head = splice;
126     else
127         current.getPrevious().setNext(splice);
128 }
129
130 /*****
131  * Method: remove()
132  * Purpose: remove index postion and return object removed
133  *
134  * Notes: calls func that can throw indexoutboundsexception
135  */
```

```
136 * Parameters: int:          index          *
137 * Returns: Object:         Object removed  *
138 *****/
139 public T remove(int index) throws IndexOutOfBoundsException
140 {
141     Node<T> current = head; // current in the walk
142     Node<T> previous = null; // previous in the walk
143     // walk the node list
144     while((current != null) && (index != 0))
145     {
146         index--;
147         previous = current;
148         current = current.getNext();
149     }
150     if(current == null)
151         throw new IndexOutOfBoundsException();
152     if(previous == null)
153         head = current.getNext();
154     else
155         previous.setNext(current.getNext());
156     return (current.getData());
157 }
158
159
160 /*****
161 * Method: removeAll
162 *
163 * Purpose: removes all nodes from array
164 *
165 * Parameters:          N/A
166 * Returns: void:       N/A
167 *****/
168 public void removeAll() {head=null;}
169
170 /*****
171 * Method: addLast()
172 * Purpose: add a object to last index
173 *
174 * Notes: calls func that can throw indexoutboundsexception
175 *
176 * Parameters:
177 *           T:          Object to be placed
178 *
179 * Returns: void:       N/A
180 *****/
181 public void addLast(T data)
182 {
183     Node<T> target; // target Node
184     Node<T> insert = new Node<T>(); // New inserted node
185
186     // load data into node
187     insert.setData(data);
188     if(head!=null)
189     {
190         target = this.getNode(this.size()-1); // get the target index's node
191         target.setNext(insert); // set last to be the next node to append
192         insert.setPrevious(target); // set insert to the pre last
193     }
194     else
195     {
196         head = insert;
197         insert.setPrevious(head);
198     }
199 }
200 }
```