```java
/*************************************************************
* Dalton Nofs                                                *
* Login ID: nofs5491                                         *
* CS-102, Summer 2017                                        *
* Programming Assignment 2                                   *
* LinkedList class:  node object for linkedLists             *
*************************************************************/
public class LinkedList<T> implements ListInterface<T>
{
    Node<T> head; // Head of the linked list

    /*************************************************************
    * Method: LinkedList()                                       *
    * Purpose: default constructor for linkedList obj            *
    *                                                             *
    * Parameters:               N/A                              *
    * Returns: void:            N/A                              *
    *************************************************************/
    public LinkedList()
    {
        head = null;
    }

    /*************************************************************
    * Method: isEmpty()                                          *
    * Purpose: check to see if linkedList is empty               *
    *                                                             *
    * Parameters:               N/A                              *
    * Returns: boolean:         if list is empty                 *
    *************************************************************/
    public boolean isEmpty()
    {
        return (head == null);
    }

    /*************************************************************
     * Method: size()                                            *
     * Purpose: determine the size of linked list                *
     *                                                            *
     *  Parameters:         N/A                                   *
     *  Returns: int:         the size of the array               *
     *************************************************************/
    public int size()
    {
        Node<T> current = head; // counter node started at head
        int counter = 0;     // counter for size calc
        while(current != null)
        {
            current = current.getNext(); // get next node
            counter++;
        }
        return counter; // return the size of linkedList
    }

    /*************************************************************
    * Method: get()                                              *
    * Purpose: get object from linked list at index              *
    *                                                            *
    * Parameters: int:          index                            *
    * Returns: T:                Object stored in index          *
    *************************************************************/
    public T get(int index) throws IndexOutOfBoundsException
    {
        Node<T> current = head; // set current to starting point
        Node<T> previous = null; // holder for previous node
        // walk array to find index
        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     *               T:             Object to be placed            *
112     *                                                             *
113     * Returns: void:                N/A                            *
114     *************************************************************/
115     public void add(int index, T datum)
116     {
117         head = add(index,datum,head);
118     }
119
120     /*************************************************************
121     * Method: add()              *PRIVATE*                          *
122     * Purpose: add a object at specified index (recursive)       *
123     *                                                             *
124     * Notes: calls func that can throw indexoutboundsexception     *
125     *                                                               *
126     * Parameters:                                                 *
127     *               int:           index                          *
128     *               T:             Object to be placed            *
129     *               Node<T>:         current node                     *
130     *                                                             *
131     * Returns: Node<T>:          the current node                 *
132     *************************************************************/
133     private Node<T> add(int index, T datum, Node<T> current)
134     {
135         if(index == 0) // check for head
```

```
136         {
137             Node<T> splice = new Node<T>(); // create and fill splice
138             splice.setData(datum);
139             splice.setNext(current);
140             return(splice);
141         }
142         if( current == null ) // check to make sure we havent fallen off list
143             throw new IndexOutOfBoundsException();
144         current.setNext( add(index--,datum, current.getNext()) ); // build rest of list
145         return current;
146     }
147 
148     /*************************************************************
149     * Method: remove()                                           *
150     * Purpose: remove index postion and return object removed    *
151     *                                                            *
152     * Notes: calls func that can throw indexoutboundsexception   *
153     *                                                            *
154     * Parameters: int:           index                           *
155     * Returns: Object:           Object removed                  *
156     *************************************************************/
157     public T remove(int index) throws IndexOutOfBoundsException
158     {
159         Node<T> current = head; // current in the walk
160         Node<T> previous = null; // previous in the walk
161         // walk the node list
162         while((current != null) && (index != 0))
163         {
164             index--;
165             previous = current;
166             current = current.getNext();
167         }
168         if(current == null)
169             throw new IndexOutOfBoundsException();
170         if(previous == null)
171             head = current.getNext();
172         else
173             previous.setNext(current.getNext());
174         return (current.getData());
175 
176     }
177 
178     /*************************************************************
179     * Method: removeAll                                          *
180     *                                                            *
181     * Purpose: removes all nodes from array                      *
182     *                                                            *
183     * Parameters:              N/A                               *
184     * Returns: void:           N/A                               *
185     *************************************************************/
186     public void removeAll() {head=null;}
187 
188     /*************************************************************
189     * Method: addLast()                                          *
190     * Purpose: add a object to last index                        *
191     *                                                            *
192     * Notes: calls func that can throw indexoutboundsexception   *
193     *                                                            *
194     * Parameters:                                                *
195     *               T:            Object to be placed          *
196     *                                                            *
197     * Returns: void:                N/A                          *
198     *************************************************************/
199     public void addLast(T data)
200     {
201         Node<T> target;             // target Node
202         Node<T> insert = new Node<T>(); // New inserted node
203 
```

```
204         // load data into node
205         insert.setData(data);
206         if(head!=null)
207         {
208             target = this.getNode(this.size()-1); // get the target index's node
209             target.setNext(insert); // set last to be the next node to append
210             insert.setPrevious(target); // set insert to the pre last
211         }
212         else
213         {
214             head = insert;
215             insert.setPrevious(head);
216         }
217     }
218 }
```