```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))
        {
            index--;
            previous = current;
            current = current.getNext();
        }
        // index is not in array
        if (index != 0)
            throw new IndexOutOfBoundsException();
        return current.getData(); // return the data found
    }

    /**************************************************************
    * Method: getNode()    !!! Private !!!                           *
    * Purpose: get Node from linked list at index                  *
    *                                                               *
    * Parameters: int:           index                            *
    * Returns: Node:             Node stored in index              *
    ***************************************************************/
    private Node<T> getNode(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))
        {
            index--;
            previous = current;
            current = current.getNext();
        }
        // index is not in array
        if (index != 0)
            throw new IndexOutOfBoundsException();
        return current; // return the node found
    }

    /**************************************************************
    * Method: add()                                                 *
    * Purpose: add a object at specified index                     *
    *                                                             *
    * Notes: calls func that can throw indexoutboundsexception     *
    *                                                               *
    * Parameters:                                                  *
    *               int:           index                           *
    *               Object:         Object to be placed             *
    *                                                              *
    * Returns: void:                N/A                             *
    ***************************************************************/
    public void add(int index, T data)
    {
        // get the correct node
        Node<T> current = this.getNode(index);

        // create the new splice node
        Node<T> splice = new Node<T>();
        splice.setData(data);
        splice.setNext(current);
        if(current == null)
            head = splice;
        else
            current.getPrevious().setNext(splice);
    }

    /**************************************************************
    * Method: remove()                                               *
    * Purpose: remove index postion and return object removed    *
    *                                                             *
    * Notes: calls func that can throw indexoutboundsexception     *
    *                                                               *
```

```java
    * Parameters: int:           index                         *
    * Returns: Object:           Object removed                   *
    ***********************************************************/
    public T remove(int index) throws IndexOutOfBoundsException
    {
        Node<T> current = head; // current in the walk
        Node<T> previous = null; // previous in the walk
        // walk the node list
        while((current != null) && (index != 0))
        {
            index--;
            previous = current;
            current = current.getNext();
        }
        if(current == null)
            throw new IndexOutOfBoundsException();
        if(previous == null)
            head = current.getNext();
        else
            previous.setNext(current.getNext());
        return (current.getData());

    }

    /**********************************************************
    * Method: removeAll                                          *
    *                                                          *
    * Purpose: removes all nodes from array                     *
    *                                                           *
    * Parameters:              N/A                             *
    * Returns: void:           N/A                             *
    ***********************************************************/
    public void removeAll() {head=null;}

    /**********************************************************
    * Method: addLast()                                          *
    * Purpose: add a object to last index                    *
    *                                                        *
    * Notes: calls func that can throw indexoutboundsexception     *
    *                                                           *
    * Parameters:                                              *
    *               T:             Object to be placed           *
    *                                                          *
    * Returns: void:                N/A                         *
    ***********************************************************/
    public void addLast(T data)
    {
        Node<T> target;               // target Node
        Node<T> insert = new Node<T>(); // New inserted node

        // load data into node
        insert.setData(data);
        if(head!=null)
        {
            target = this.getNode(this.size()-1); // get the target index's node
            target.setNext(insert); // set last to be the next node to append
            insert.setPrevious(target); // set insert to the pre last
        }
        else
        {
            head = insert;
            insert.setPrevious(head);
        }
    }
}
```