```java
package hwk2;

/**
 * Linked List is a collection of data nodes.  All methods here relate to
 * how one can manipulate those nodes.
 *
 * @author Irene Yin
 * @version Oct.7.2019
 */
public class LinkedList
{
    private int length;          // number of nodes the linkList have in total
    private ListNode firstNode;  // pointer to first node

    public LinkedList()
    {
        length=0;
        firstNode=null;
    }

    /** insert new String at linked list's head
     *
     * @param newData the String to be inserted
     */
    public void insertAtHead(String newData)
    {
       ListNode newNode = new ListNode(newData);
        if (isEmpty())
        {
            firstNode=newNode;
        }
        else
        {
            newNode.next=firstNode;
            firstNode=newNode;
        }
        length++;
    }

    /** remove and return data at the head of the list
     *
     *  @return the String the deleted node contains.  Returns null if list empty.
     */
    public String removeHead()
    {
        if (!isEmpty()){
            String toReturn = firstNode.toString();
            firstNode = firstNode.next;
            length--;
            return toReturn;
        }
        else{ //if the linkList is empty
            return null;
        }

    }

    /** insert data at end of list
     *
     * @param newData new String to be inserted
     */
    public void insertAtTail(String newData)
    {
        ListNode newNode = new ListNode(newData);
        if (!isEmpty()){
            ListNode nextNode = firstNode;
            for(int i = 1; i < getLength(); i++){
                nextNode = nextNode.next;
            }
            nextNode.next = newNode;
        }
        else{// if linkList is empty
            firstNode = newNode;
        }
        length++;
    }

    /**
     * search for first occurrence of value and return index where found
     *
     * @param value string to search for
     * @return index where string occurs (first node is index 0).  Return -1 if
value not found.
     */
    public int indexOf(String value)
    {
        ListNode originalFirstNode = firstNode;
        int index = 0;
        if(!isEmpty()){
            while (!firstNode.data.equals(value)) {
                if (firstNode.next == null) {
                    return -1;
                }
                firstNode = firstNode.next;
                index++;
            }
            firstNode = originalFirstNode;
            return index;
        }
        else{
            return -1;
        }
    }


    /**
     *  @return return linked list as printable string
     */
    public String toString()
    {
       String toReturn="(";
       ListNode runner=firstNode;
       while (runner!=null)
       {
          toReturn = toReturn + runner;  //call node's toString automatically
          runner=runner.next;
          if (runner!=null)
          {
             toReturn = toReturn + ",";
          }
       }
       toReturn = toReturn + ")";
       return toReturn;
    }

    /**
     *
     * @return length of LL
     */
    public int getLength() {return length;}
    /**
     *
     * @return true if LL empty or false if not
     */
    public boolean isEmpty() {return getLength()==0;}
}




/**
 * JUnit test class.
 */
import hwk2.LinkedList;
import org.junit.*;
import org.junit.rules.Timeout;
import static org.junit.Assert.*;

public class linkedListTest
{
    @Rule // a test will fail if it takes longer than 1/10 of a second to run
    public Timeout timeout = Timeout.millis(100);

    @Test // the first listNode in the linkList should be removed with element
inside
    public void testRemoveHead(){
        LinkedList c = new LinkedList();
        c.insertAtHead("a");
        c.insertAtHead("b");
        assertEquals("(b,a)",c.toString());
        assertEquals("b",c.removeHead());
        assertEquals("(a)",c.toString());
        assertEquals(1,c.getLength());
    }

    @Test // removeHead method should do nothing if there's no listNode inside.
    public void testRemoveHead2(){
        LinkedList c = new LinkedList();
        assertEquals(null,c.removeHead());


    }

    @Test // there should be the listNode we assign at the end of the LinkList.
    public void testInsertAtTail(){
        LinkedList c = new LinkedList();
        c.insertAtHead("c");
        c.insertAtHead("b");
        c.insertAtHead("a");
        c.insertAtTail("d");

        assertEquals("(a,b,c,d)", c.toString());

    }

    @Test // there should be the listNode we assign at the end of the LinkList.
    public void testInsertAtTail2(){
        LinkedList c = new LinkedList();
        c.insertAtTail("a");

        assertEquals("(a)", c.toString());
        assertEquals(1,c.getLength());

    }

    @Test // there should be the listNode we assign at the end of the LinkList.
```

```java
    public void testInsertAtTail3(){
        LinkedList c = new LinkedList();
        c.insertAtHead("a");
        c.insertAtTail("b");

        assertEquals("(a,b)", c.toString());
        assertEquals(2,c.getLength());

    }

    @Test // should return the address of the listNode data we want.
    public void testIndexOf_Found(){
        LinkedList c = new LinkedList();
        c.insertAtHead("c");
        c.insertAtHead("b");
        c.insertAtHead("a");
        assertEquals(0,c.indexOf("a"));


    }

    @Test // should return the index of the listNode data we want. two same data
    public void testIndexOf_Found2(){
        LinkedList c = new LinkedList();
        c.insertAtHead("c");
        c.insertAtHead("b");
        c.insertAtHead("a");
        c.insertAtHead("a");
        assertEquals(2,c.indexOf("b"));


    }
    @Test // should return the index of the listNode data we want. two same data
    public void testIndexOf_Found3(){
        LinkedList c = new LinkedList();
        c.insertAtHead("c");
        c.insertAtHead("b");
        c.insertAtHead("a");
        c.insertAtHead("a");
        assertEquals(3,c.indexOf("c"));


    }

    @Test // should return -1 since there is no such data.
    public void testIndexOf_notFound(){
        LinkedList c = new LinkedList();
        c.insertAtHead("a");
//          c.insertAtHead("b");
//          c.insertAtHead("c");
        assertEquals(-1,c.indexOf("d"));

    }

    @Test // should return -1 since there is no such data.
    public void testIndexOf_emtpty(){
        LinkedList c = new LinkedList();
        assertEquals(-1,c.indexOf("d"));

    }
}
```