```
/**
 * Linked List is a collection of data nodes.  All methods here relate
to
 * how one can manipulate those nodes.
 *
 * @author Blair Hagen
 * @version 4-28-2016
 *
 * As a student at Union College, I am part of a community that values
intellectual effort, curiosity and discovery. I understand that in
order to truly claim my educational and academic achievements, I am
obligated to act with academic integrity. Therefore, I affirm that I
will carry out my academic endeavors with full academic honesty, and I
rely on my fellow students to do the same.
 */
public class LinkedList
{
    private int length;          // number of nodes
    private ListNode firstNode;  // pointer to first node

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

    /** insert new Event at linked list's head
     *
     * @param newData the Event to be inserted
     */
    public void insertAtHead(Event newData)
    {
     ListNode newnode = new ListNode(newData);
        if (getLength() == 0)
        {
            firstNode=newnode;
        }
        else
        {
            newnode.next=firstNode;
            firstNode=newnode;
        }
        length++;
    }
```

```java
/**
 * Removes the link node at the head of the list and returns it
 * If list is empty, returns null
 *
 * @return headOfList object at head of the list
 *           null if doesn't exist
 */
public Event removeHead() {
       if (firstNode == null)
       {
             return(null);
       }
       else
       {
             ListNode headOfList = firstNode;
             if (firstNode.next != null)
             {
                   firstNode = firstNode.next;
             }
             else
             {
                   firstNode = null;
             }
             length--;
             return(headOfList.data);
       }
}

/**
 * Insert link node at end of linked list
 *
 * @param newEvent
 *           The event for the added linked node to contain
 */
public void insertAtTail(Event newEvent) {
  ListNode runner = firstNode;
  ListNode newNode = new ListNode(newEvent);
  while(runner != null && runner.next != null)
  {
        runner = runner.next;
  }
  if (runner == null)
  {
```

```
            firstNode = newNode;
            length++;
       }
       else
       {
            runner.next = newNode;
            length++;
       }
     }

     /**
      * Searches for an event based on its event name and returns the
start time.
      *
      * @param eventName
      *           Name of event to be searched for
      * @return eventTime
      *           Start time of found event, -1 if event does not exist
      */
     public int search(String eventName) {
       ListNode runner = firstNode;

          while(runner != null && !
runner.data.getName().equals(eventName))
          {
             runner = runner.next;
          }
          if (runner == null)
          {
             return(-1);
          }
          else
          {
             return (runner.data.getStart());
          }



     }

     /** Turn entire chain into a string
      *
      *  @return return linked list as printable string
      */
```

```
    public String toString()
    {
     String toReturn="(";
     ListNode runner;
     runner=firstNode;
     while (runner!=null)
     {
            toReturn = toReturn + runner;  //call node's toString
automatically
            runner=runner.next;
            if (runner!=null)
            {
                   toReturn = toReturn + ",\n";
            }
     }
     toReturn = toReturn + ")";
     return toReturn;
    }

    /** getter for number of nodes in the linked list
     *
     * @return length of LL
     */
    public int getLength() {return length;}
}



/**
 * Tests new (and old?) LinkedList methods.
 *
 * @author Blair Hagen, Aaron Cass, and Chris Fernandes
 * @version 4/28/16
 */
public class LinkedListTester
{
    public static void main(String[] args)
    {
        Testing.setVerbose(true); // use false for less testing output

        Testing.startTests();

      testInsertAtHead();
```

```java
  testRemoveHead();

  testInsertAtTail();

  testSearch();

     Testing.finishTests();
}

private static void testInsertAtHead()
{
  Testing.testSection("insertAtHead");

     LinkedList list = new LinkedList();
     Event E1 = new Event("chess club", 2013, 2, 25, 1900, 1930);
     Event E2 = new Event("boy scouts", 2013, 2, 23, 900, 1000);

     list.insertAtHead(E1);

     Testing.assertEquals("Inserting at head of an empty list",
                          "(" + E1 + ")",
                          list.toString());

     Testing.assertEquals("Inserting at head of an empty list, "
                          + "length should change",
                          1,
                          list.getLength());

     list.insertAtHead(E2);

     Testing.assertEquals("Inserting at head of a 1-element list",
                          "(" + E2 + ",\n" + E1 + ")",
                          list.toString());

     Testing.assertEquals("Inserting at head of a 1-element list, "
                          + "length should change",
                          2,
                          list.getLength());
}

private static void testRemoveHead()
{
  Testing.testSection("removeHead");
```

```
        LinkedList list = new LinkedList();

        Testing.assertEquals("Removing head of empty list",
                             null,
                             list.removeHead());

        Event E1 = new Event("chess club", 2013, 2, 25, 1900, 1930);
        Event E2 = new Event("boy scouts", 2013, 2, 23, 900, 1000);
        list.insertAtHead(E1);
        list.insertAtHead(E2);
        Event result = list.removeHead();

        Testing.assertEquals("Removing head of list with two items",
                             "(" + E1 + ")",
                             list.toString());

        Testing.assertEquals("Length should change",
                             1,
                             list.getLength());

        Testing.assertEquals("Should return removed event",
                             E2,
                             result);

        LinkedList list2 = new LinkedList();
        list2.insertAtHead(E1);
        list2.removeHead();

        Testing.assertEquals("Removing head from list with one element",
    "()", list2.toString());

        Testing.assertEquals("Length should change", 0,
    list2.getLength());

      }

      private static void testInsertAtTail()
      {
        Testing.testSection("insertAtTail");

        LinkedList list = new LinkedList();

        Event E1 = new Event("chess club", 2013, 2, 25, 1900, 1930);
        Event E2 = new Event("boy scouts", 2013, 2, 23, 900, 1000);
        Event E3 = new Event("end event", 2013, 2, 23, 900, 1000);
        list.insertAtHead(E1);
        list.insertAtHead(E2);
        list.insertAtTail(E3);

        Testing.assertEquals("Adding event to end of list (called 'end
    event')",
                             "(" + E2 + ",\n" + E1 + ",\n" + E3 +
    ")",
                             list.toString());

        Testing.assertEquals("Length should change",
                             3,
                             list.getLength());

        LinkedList list2 = new LinkedList();
        list2.insertAtTail(E1);

        Testing.assertEquals("Inserting a tail in an empty list", "(" +
    E1 + ")" , list2.toString());


      }

      private static void testSearch()
      {
        Testing.testSection("search");

        LinkedList list = new LinkedList();

        Event E1 = new Event("chess club", 2013, 2, 25, 1900, 1930);
        Event E2 = new Event("boy scouts", 2013, 2, 23, 900, 1000);
        Event E3 = new Event("police academy", 2013, 2, 22, 700, 1000);

        list.insertAtHead(E1);
        list.insertAtHead(E2);
        list.insertAtHead(E3);

        Testing.assertEquals("Search for event in middle of list",
                             900,
                             list.search("boy scouts"));

        Testing.assertEquals("searching for event in the first position
    of list",
                                1900,
                                list.search("chess club"));
     Testing.assertEquals("searching for event in last position of
list" ,
                                700,
                                list.search("police academy"));

     Testing.assertEquals("Should return -1 if event doesnt exist",
                                -1,
                                list.search("Nonexistent"));

     LinkedList list2 = new LinkedList();

     Testing.assertEquals("Should return -1 if linked list is empty",
-1, list2.search("Event"));


    }
}
```

1. Worst case runtimes for three written methods

    removeHead has notation O(1).
    insertAtTail has notation O(n).
    search has notation O(n).

2. Running time comparison for an array with remove head

    The operation on an array would be much longer as an array must have all the elements moved to the left one.

    An array's removeHead O notation would be O(n), versus the O(1) of linkedNode removeHead.

3. Running time comparison for search method

    They are the same as both must step through every element in the array/linkedlist.

    They are both O(n).