OrderedList.java

```java
/* Vincent Chan
 * masc0264
 */

package data_structures;

import java.util.Iterator;
import java.util.NoSuchElementException;
import java.util.ConcurrentModificationException;

/*Ordered Linked List
 * This is an ordered linked list data structure.
 * It will organize data by ascending order.
 *
 * Supported Constructors:
 *    Default
 *
 * Supported Insertions:
 *    Insertion
 *
 * Supported Removals:
 *    Remove first
 *    Remove by object
 */

public class OrderedList<E> implements Iterable<E>{
  /*Functions Included
   * ====PUBLIC====
   * Constructors        //Ln. 57
   * insert(object)      //Ln. 63
   * remove(object)      //Ln. 94
   * pop()               //Ln. 116
   * peek()              //Ln. 127
   * contains(object)    //Ln. 133
   * find(object)        //Ln. 141
   * isEmpty()           //Ln. 151
   * size()              //Ln. 157
   * Iterator()          //Ln. 162
   */

  //Variable declarations
  Node<E> head, tail;
  int size;
  long modifyCtr;

  //Node class for the linked list
  class Node<T> {
    T data;
    Node<T> next;

    public Node(T obj) {
      data = obj;
      next = null;
    }
  } //End Node class

  //Constructors
  public OrderedList() {
    size = 0;
    modifyCtr = 0;
  } //End constructors

  //This will insert the object
```

```
64    //Ordered by ascending order defined by
65    //the comparable interface.
66    public void insert(E obj) {
67      Node<E> newNode = new Node(obj);
68      if(isEmpty()) head = tail = newNode;
69      else {
70        Node<E> previous = null;
71        Node<E> current = head;
72        while(current!=null) {
73          if(((Comparable<E>)obj).compareTo(current.data)<0)
74            if(previous==null) {
75              newNode.next = current;
76              head = newNode;
77              break;
78            }
79            else {
80              previous.next = newNode;
81              newNode.next = current;
82              break;
83            }
84          previous = current;
85          current = current.next;
86        }
87        if(current==null)
88          tail.next = tail = newNode;
89      }
90      size++;
91      modifyCtr++;
92    } //End insert
93
94      //This will remove the object if it is contained
95    //in the list. Null otherwise.
96    public E remove(E obj) {
97      Node<E> previous = null;
98      Node<E> current = head;
99      while(current != null)
100       if(((Comparable<E>)obj).compareTo(current.data)==0) {
101         if(previous != null)
102           previous.next = current.next;
103         else
104           head = current.next;
105         size--;
106         modifyCtr++;
107         return current.data;
108       }
109       else {
110         previous = current;
111         current = current.next;
112       }
113     return null;
114   } //End remove()
115
116   //This will pop out the first element
117   //in the list. Null if is empty.
118   public E pop() {
119     if(isEmpty()) return null;
120     E temp = head.data;
121     head = head.next;
122     size--;
123     modifyCtr++;
124     return temp;
125   } //End pop()
126
127   //This will return but not remove
128   //the first element of the list.
129   public E peek() {
130     return (isEmpty())? null : head.data;
131   } //End peek()
```

```
132
133   //Returns true if the list contains the element
134   public boolean contains(E obj) {
135     for(E current : this)
136       if(((Comparable<E>)obj).compareTo(current)==0)
137         return true;
138     return false;
139   } //End contains()
140
141   //This will find and return the object
142   //if it is contained in the list.
143   //Null otherwise.
144   public E find(E obj) {
145     for(E current : this)
146       if(((Comparable<E>)obj).compareTo(current)==0)
147         return obj;
148     return null;
149   } //End find()
150
151   //This will return true if the
152   //list contains nothing.
153   public boolean isEmpty() {
154     return size == 0;
155   } //End isEmpty()
156
157   //Returns the size of the list.
158   public int size() {
159     return size;
160   } //End size()
161
162   //This returns an iterator for iteration
163   public Iterator<E> iterator() {
164     return new IteratorHelper();
165   } //End iterator()
166
167   class IteratorHelper implements Iterator<E> {
168     private long lastMod;
169     Node<E> current;
170
171     public IteratorHelper() {
172       lastMod = modifyCtr;
173       current = head;
174     }
175
176     public boolean hasNext() {
177       return current != null;
178     }
179
180     public E next() {
181       if(lastMod != modifyCtr)
182         throw new ConcurrentModificationException();
183       if(!hasNext())
184         throw new NoSuchElementException();
185       E data = current.data;
186       current = current.next;
187       return data;
188     }
189
190     public void remove() {
191       throw new UnsupportedOperationException();
192     }
193   }
194 } //End OrderedList
```