

# OrderedList.java

```
1  /* Vincent Chan
2   * masc0264
3   */
4
5  package data_structures;
6
7  import java.util.Iterator;
8  import java.util.NoSuchElementException;
9  import java.util.ConcurrentModificationException;
10
11  /*Ordered Linked List
12   * This is an ordered linked list data structure.
13   * It will organize data by ascending order.
14   *
15   * Supported Constructors:
16   *   Default
17   *
18   * Supported Insertions:
19   *   Insertion
20   *
21   * Supported Removals:
22   *   Remove first
23   *   Remove by object
24   */
25
26  public class OrderedList<E> implements Iterable<E>{
27      /*Functions Included
28       * ====PUBLIC====
29       * Constructors           //Ln. 57
30       * insert(object)         //Ln. 63
31       * remove(object)         //Ln. 94
32       * pop()                  //Ln. 116
33       * peek()                 //Ln. 127
34       * contains(object)       //Ln. 133
35       * find(object)           //Ln. 141
36       * isEmpty()              //Ln. 151
37       * size()                 //Ln. 157
38       * Iterator()             //Ln. 162
39       */
40
41      //Variable declarations
42      Node<E> head, tail;
43      int size;
44      long modifyCtr;
45
46      //Node class for the linked list
47      class Node<T> {
48          T data;
49          Node<T> next;
50
51          public Node(T obj) {
52              data = obj;
53              next = null;
54          }
55      } //End Node class
56
57      //Constructors
58      public OrderedList() {
59          size = 0;
60          modifyCtr = 0;
61      } //End constructors
62
63      //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 //Returns true if the list contains the element
133 public boolean contains(E obj) {
134     for(E current : this)
135         if(((Comparable<E>)obj).compareTo(current)==0)
136             return true;
137     return false;
138 } //End contains()
139
140 //This will find and return the object
141 //if it is contained in the list.
142 //Null otherwise.
143 public E find(E obj) {
144     for(E current : this)
145         if(((Comparable<E>)obj).compareTo(current)==0)
146             return obj;
147     return null;
148 } //End find()
149
150 //This will return true if the
151 //list contains nothing.
152 public boolean isEmpty() {
153     return size == 0;
154 } //End isEmpty()
155
156 //Returns the size of the list.
157 public int size() {
158     return size;
159 } //End size()
160
161 //This returns an iterator for iteration
162 public Iterator<E> iterator() {
163     return new IteratorHelper();
164 } //End iterator()
165
166 class IteratorHelper implements Iterator<E> {
167     private long lastMod;
168     Node<E> current;
169
170     public IteratorHelper() {
171         lastMod = modifyCtr;
172         current = head;
173     }
174
175     public boolean hasNext() {
176         return current != null;
177     }
178
179     public E next() {
180         if(lastMod != modifyCtr)
181             throw new ConcurrentModificationException();
182         if(!hasNext())
183             throw new NoSuchElementException();
184         E data = current.data;
185         current = current.next;
186         return data;
187     }
188
189     public void remove() {
190         throw new UnsupportedOperationException();
191     }
192 }
193 } //End OrderedList

```