UnorderedList.java

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

package data_structures;

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

/*Unordered Linked List
 * This is an unordered linked list data structure.
 * It will manage data by adding data to the start
 * or end of the stack. It will not order the objects.
 *
 * Supported Constructors:
 *    Default
 *
 * Supported Insertions:
 *    Front insertion
 *    End insertion
 *
 * Supported Removals:
 *    Remove first
 *    Remove by object
 *    Remove by index
 */

public class UnorderedList<E> implements Iterable<E>{
  /* Functions Included
   * ====PUBLIC====
   * Constructor         //Ln. 58
   * addFirst(object)    //Ln. 64
   * addLast(object)     //Ln. 77
   * removeFirst()       //Ln. 88
   * find(object)        //Ln. 98
   * remove(object)      //Ln. 108
   * isEmpty()           //Ln. 129
   * size()              //Ln. 134
   * iterator()          //Ln. 139
   */

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

  class Node<T> {
    T data;
    Node<T> next;

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

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

```

```
64    //Add in front of the list
65    public void addFirst(E obj) {
66      Node<E> newNode = new Node(obj);
67      if(isEmpty())
68        head = tail = newNode;
69      else {
70        newNode.next = head;
71        head = newNode;
72      }
73      size++;
74      modifyCtr++;
75    } //End addFirst()
76
77    //Add to the back of the list.
78    public void addLast(E obj) {
79      Node<E> newNode = new Node(obj);
80      if(isEmpty())
81        head = tail = newNode;
82      else
83        tail = tail.next = newNode;
84      size++;
85      modifyCtr++;
86    } //End addLast()
87
88    //Remove the first element
89    public E removeFirst() {
90      if(isEmpty()) return null;
91      E temp = head.data;
92      head = head.next;
93      size--;
94      modifyCtr++;
95      return temp;
96    } //End removeFirst()
97
98    //Find the specified object
99    public E find(E obj) {
100     Node<E> current = head;
101     while(current != null)
102       if(((Comparable<E>)obj).compareTo(current.data)==0)
103         return obj;
104       else current = current.next;
105     return null;
106   } //End find()
107
108   //Remove the specified object
109   public E remove(E obj) {
110     Node<E> previous = null;
111     Node<E> current = head;
112     while(current != null)
113       if(((Comparable<E>)obj).compareTo(current.data)==0) {
114         if(previous != null)
115           previous.next = current.next;
116         else
117           head = current.next;
118         size--;
119         modifyCtr++;
120         return current.data;
121       }
122       else {
123         previous = current;
124         current = current.next;
125       }
126     return null;
127   } //End remove()
128
129   //Returns true if empty
130   public boolean isEmpty() {
131     return size == 0;
```

```
132     } //End isEmpty()
133
134     //Returns the size
135     public int size() {
136       return size;
137     } //End size()
138
139     //Returns an iterator for iteration
140     public Iterator<E> iterator() {
141       return new IteratorHelper();
142     } //End iterator()
143
144     class IteratorHelper implements Iterator<E> {
145       private long lastMod;
146       Node<E> current;
147
148       public IteratorHelper() {
149         lastMod = modifyCtr;
150         current = head;
151       }
152
153       public boolean hasNext() {
154         return current != null;
155       }
156
157       public E next() {
158         if(lastMod != modifyCtr)
159           throw new ConcurrentModificationException();
160         if(!hasNext())
161           throw new NoSuchElementException();
162         E data = current.data;
163         current = current.next;
164         return data;
165       }
166
167       public void remove() {
168         throw new UnsupportedOperationException();
169       }
170     }
171   } //End UnorderedList
```