```java
import CITS2200.Link;
import CITS2200.List;
import CITS2200.OutOfBounds;
import CITS2200.WindowLinked;

public class ListLinked implements List {

    /**
     * stores the reference of the head(before) Link
     */
    private Link head;
    /**
     * stores the reference of the tail(after) Link
     */
    private Link tail;

    /**
     * Constructor creates an empty linked list
     */
    public ListLinked() {
        tail = new Link(null, null);
        head = new Link(null, tail);
    }


    /**
     * checks if list is empty
     *
     * @return true if list is empty
     */
    @Override
    public boolean isEmpty() {
        return head.successor.equals(tail);
    }

    /**
     * Checks if windowLink is at the head(beginning) of the list
     *
     * @param windowLinked is the windowLink being checked
     * @return true if windowLink is at the head
     */
    @Override
    public boolean isBeforeFirst(WindowLinked windowLinked) {
        return windowLinked.link.equals(head);
    }

    /**
     * Checks if windowLink is at the tail(end) of the list
     *
     * @param windowLinked is the windowLink being checked
     * @return true if windowLink is at the tail
     */
    @Override
```

```
 54      public boolean isAfterLast(WindowLinked windowLinked)
    {
 55          return windowLinked.link.equals(tail);
 56      }
 57
 58      /**
 59       * Set the windowLink to the head of the list
 60       *
 61       * @param windowLinked the windowLink
 62       */
 63      @Override
 64      public void beforeFirst(WindowLinked windowLinked) {
 65          windowLinked.link = head;
 66      }
 67
 68      /**
 69       * Set the windowLink to the tail of the list
 70       *
 71       * @param windowLinked the windowLink
 72       */
 73      @Override
 74      public void afterLast(WindowLinked windowLinked) {
 75          windowLinked.link = tail;
 76      }
 77
 78      /**
 79       * Moves the windowLink to the next Link
 80       *
 81       * @param windowLinked the current windowLink
 82       * @throws OutOfBounds if calling next after list ends
 83       */
 84      @Override
 85      public void next(WindowLinked windowLinked) throws
    OutOfBounds {
 86          if (!isAfterLast(windowLinked)) {
 87              windowLinked.link = windowLinked.link.
    successor;
 88          } else {
 89              throw new OutOfBounds("Calling next after list
     ends");
 90          }
 91      }
 92
 93      /**
 94       * Moves the windowLink to the previous Link
 95       *
 96       * @param windowLinked is the current windowLink
 97       * @throws OutOfBounds if calling previous at the
    beginning of list
 98       */
 99      @Override
100      public void previous(WindowLinked windowLinked) throws
     OutOfBounds {
101          if (!isBeforeFirst(windowLinked)) {
102              Link current = head.successor;
```

```
103             Link previous = head;
104             while (current != windowLinked.link) {
105                 previous = current;
106                 current = current.successor;
107             }
108             windowLinked.link = previous;
109         } else throw new OutOfBounds("Calling previous
    before start of list");
110     }
111 
112     /**
113      * Inserts a new Link after the windowLink
114      *
115      * @param o            is the object to be inserted
116      * @param windowLinked is the windowLink after which
    the new link will be inserted
117      * @throws OutOfBounds if inserting after list ends
118      */
119     @Override
120     public void insertAfter(Object o, WindowLinked
    windowLinked) throws OutOfBounds {
121         if (!isAfterLast(windowLinked)) {
122             Link afterWindow = windowLinked.link.successor
    ;
123             if (windowLinked.link.successor.equals(tail))
    {
124                 windowLinked.link.successor = new Link(o,
    afterWindow);
125                 tail = windowLinked.link.successor.
    successor;
126                 if (isBeforeFirst(windowLinked)) {
127                     head.successor = windowLinked.link.
    successor;
128                 }
129             } else {
130                 windowLinked.link.successor = new Link(o,
    afterWindow);
131                 windowLinked.link.successor.successor =
    afterWindow.successor;
132             }
133         } else throw new OutOfBounds("Inserting after end
    of list");
134     }
135 
136     /**
137      * Inserts a new Link before the windowLink
138      *
139      * @param o            is the object to be inserted
140      * @param windowLinked is the windowLink before which
    the new link will be inserted
141      * @throws OutOfBounds if inserting before list starts
142      */
143     @Override
144     public void insertBefore(Object o, WindowLinked
    windowLinked) throws OutOfBounds {
```

```
145         if (!isBeforeFirst(windowLinked)) {
146             windowLinked.link.successor = new Link(
    windowLinked.link.item, windowLinked.link.successor);
147             if (isAfterLast(windowLinked)) {
148                 tail = windowLinked.link.successor;
149                 windowLinked.link.item = o;
150                 windowLinked.link = windowLinked.link.
    successor;
151             }
152         } else throw new OutOfBounds("Inserting before
    start of list");
153     }
154 
155     /**
156      * Examine the item in the windowLink
157      *
158      * @param windowLinked the windowLink to be examined
159      * @return the item
160      * @throws OutOfBounds if windowLink is at the head or
     the tail
161      */
162     @Override
163     public Object examine(WindowLinked windowLinked)
    throws OutOfBounds {
164         if (!isBeforeFirst(windowLinked) && !isAfterLast(
    windowLinked)) {
165             return windowLinked.link.item;
166         } else throw new OutOfBounds("The windowLink is
    not in the list");
167     }
168 
169     /**
170      * Replaces the item in the windowLink with the new
    Object o
171      *
172      * @param o            the new object to replace the
    current item
173      * @param windowLinked the windowLink whose item is to
     be replaced
174      * @return the object removed
175      * @throws OutOfBounds if the windowLink is before the
     start or after the end
176      */
177     @Override
178     public Object replace(Object o, WindowLinked
    windowLinked) throws OutOfBounds {
179         if (!isBeforeFirst(windowLinked) && !isAfterLast(
    windowLinked)) {
180             Object removedObject = windowLinked.link.item;
181             windowLinked.link.item = o;
182             return removedObject;
183         } else throw new OutOfBounds("The windowLink is
    not in the list");
184     }
185 
```

```
186     /**
187      * Removes the windowLink from the list and place window over next item
188      *
189      * @param windowLinked is the window to be removed
190      * @return the removed object stored in that window
191      * @throws OutOfBounds if the windowLink is before the start or after the end
192      */
193     @Override
194     public Object delete(WindowLinked windowLinked) throws OutOfBounds {
195         if (!isBeforeFirst(windowLinked) && !isAfterLast(windowLinked)) {
196             Object deletedObject = windowLinked.link.item;
197             Link nextLink = windowLinked.link.successor;
198             if (nextLink.equals(tail)) {
199                 windowLinked.link.item = nextLink.item;
200                 tail = windowLinked.link;
201             } else {
202                 windowLinked.link.item = nextLink.item;
203             }
204             return deletedObject;
205         } else throw new OutOfBounds("Attempting to delete sentinel Links");
206     }
207 }
208
```