```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
     */
```

```
49      @Override
50      public boolean isAfterLast(WindowLinked windowLinked) {
51          return windowLinked.link.equals(tail);
52      }
53
54      /**
55       * Set the windowlink to the head of the list
56       * @param windowLinked the windowlink
57       */
58      @Override
59      public void beforeFirst(WindowLinked windowLinked) {
60          windowLinked.link = head;
61      }
62
63      /**
64       * Set the windowLink to the tail of the list
65       * @param windowLinked the windowLink
66       */
67      @Override
68      public void afterLast(WindowLinked windowLinked) {
69          windowLinked.link = tail;
70      }
71
72      /**
73       * Moves the windowLink to the next Link
74       * @param windowLinked the current windowLink
75       * @throws OutOfBounds if calling next after list ends
76       */
77      @Override
78      public void next(WindowLinked windowLinked) throws
    OutOfBounds {
79          if(!isAfterLast(windowLinked)) {
80              windowLinked.link = windowLinked.link.successor
    ;
81          }
82          else {
83              throw new OutOfBounds("Calling next after list
    ends");
84          }
85      }
86
87      /**
88       * Moves the windowLink to the previous Link
89       * @param windowLinked is the current windowLink
90       * @throws OutOfBounds if calling previous at the
    beginning of list
91       */
92      @Override
93      public void previous(WindowLinked windowLinked) throws
    OutOfBounds {
```

```
 94         if(!isBeforeFirst(windowLinked)) {
 95             Link current = head.successor;
 96             Link previous = head;
 97             while(current!= windowLinked.link) {
 98                 previous = current;
 99                 current = current.successor;
100             }
101             windowLinked.link = previous;
102         }
103         else throw new OutOfBounds("Calling previous
    before start of list");
104     }
105 
106     /**
107      * Inserts a new Link after the windowLink
108      * @param o is the object to be inserted
109      * @param windowLinked is the windowLink after which
    the new link will be inserted
110      * @throws OutOfBounds if inserting after list ends
111      */
112     @Override
113     public void insertAfter(Object o, WindowLinked
    windowLinked) throws OutOfBounds {
114         if(!isAfterLast(windowLinked)) {
115             Link afterWindow = windowLinked.link.successor
    ;
116             windowLinked.link.successor = new Link(o,
    afterWindow);
117         }
118     }
119 
120     /**
121      * Inserts a new Link before the windowLink
122      * @param o is the object to be inserted
123      * @param windowLinked is the windowLink before which
    the new link will be inserted
124      * @throws OutOfBounds if inserting before list starts
125      */
126     @Override
127     public void insertBefore(Object o, WindowLinked
    windowLinked) throws OutOfBounds {
128         if(!isBeforeFirst(windowLinked)) {
129             windowLinked.link.successor = new Link(
    windowLinked.link.item, windowLinked.link.successor);
130             if (isAfterLast(windowLinked)) {
131                 tail = windowLinked.link.successor;
132                 windowLinked.link.item = o;
133                 windowLinked.link = windowLinked.link.
    successor;
134             }
```

```
135         }
136         else throw new OutOfBounds("Inserting before start
     of list");
137     }
138 
139     /**
140      * Examine the item in the windowLink
141      * @param windowLinked the windowLink to be examined
142      * @return the item
143      * @throws OutOfBounds if windowLink is at the head or
     the tail
144      */
145     @Override
146     public Object examine(WindowLinked windowLinked)
    throws OutOfBounds {
147         if(!isBeforeFirst(windowLinked) && !isAfterLast(
    windowLinked)) {
148             return windowLinked.link.item;
149         }
150         else throw new OutOfBounds("The windowLink is not
    in the list");
151     }
152 
153     /**
154      * Replaces the item in the windowLink with the new
    Object o
155      * @param o the new object to replace the current item
156      * @param windowLinked the windowLink whose item is to
     be replaced
157      * @return the object removed
158      * @throws OutOfBounds if the windowLink is before the
     start or after the end
159      */
160     @Override
161     public Object replace(Object o, WindowLinked
    windowLinked) throws OutOfBounds {
162         if(!isBeforeFirst(windowLinked) && !isAfterLast(
    windowLinked)) {
163             Object removedObject = windowLinked.link.item;
164             windowLinked.link.item = o;
165             return removedObject;
166         }
167         else throw new OutOfBounds("The windowLink is not
    in the list");
168     }
169 
170     /**
171      * Removes the windowLink from the list and place
    window over next item
172      * @param windowLinked is the window to be removed
```

```
173      * @return the removed object stored in that window
174      * @throws OutOfBounds if the windowLink is before the
     start or after the end
175      */
176     @Override
177     public Object delete(WindowLinked windowLinked) throws
     OutOfBounds {
178         if(!isBeforeFirst(windowLinked) && !isAfterLast(
    windowLinked)) {
179             Object deletedObject = windowLinked.link.item;
180             windowLinked.link.item = windowLinked.link.
    successor.item;
181             windowLinked.link.successor = windowLinked.
    link.successor.successor;
182             return deletedObject;
183         }
184         else throw new OutOfBounds("The windowLink is not
    in the list");
185     }
186 }
187
```