```
 1
 3 * Implementation of a Singly-Linked List.
 7
 8
 9import java.util.Iterator;
11
12
13
14public class DLinkedList<E>
15{
16    // Representation of the list nodes
17    private class Node
18    {
19        E data;          // the data value stored at the node
20        Node  next; // the successor of this node
21        Node previous;
22
23        // creates a node with the given data item and no successor
24        Node(E d)
25        {
26            data = d;
27            next = null;
28            previous = null;
29        }
30    }
31
32
33    /**
34     * The first node in the list.
35     * The last node in the list.
36     */
37    private Node head;
38    private Node last;
39    private int top;
40
41
42    // put comment in Javadoc style
43    public DLinkedList()
44    {
45        head = null;
46        last = null;
47    }
48
49
50
51    // put comment in Javadoc style
52    /**
53     * If the list is empty
```

```
54      * @return the data null;
55      */
56     public boolean isEmpty()
57     {
58         return head == null;
59     }
60 
61 
62     /**
63      * Adds the given element to the front of the list.
64      *
65      * @param data the element to add
66      */
67     public void addFirst(E item)
68     {
69         Node newNode = new Node(item);
70         newNode.data = item;
71         if(isEmpty()){
72             head = newNode;
73 
74         }else{
75             head.previous = newNode;
76         }
77         newNode.next = head;
78         this.head = newNode;
79     }
80 
81 
82     /**
83      * Returns the first element in the list.
84      *
85      * @return the first element in the list
86      * @throws NoSuchElementException when the list is empty
87      */
88     public E getFirst()
89     {
90         if (this.isEmpty()) {
91             throw new NoSuchElementException();
92         }
93 
94         return head.data;
95     }
96     /**
97      * Returns the last element in the list.
98      *
99      * @return the last element in the list.
100     * @throws NoSuchElementException when list is empty
101     */
```

```
102
103     public E getLast(){
104         if(this.isEmpty()){
105             throw new NoSuchElementException();
106         }else{
107             Node curr = head;
108             while(curr != null){
109                 curr = curr.next;
110             }
111             return curr.data;
112         }
113
114
115     }
116
117
118     // put comment in Javadoc style
119     /**
120      *Adds the given item to the end of the list
121      *
122      * @param item the element to add
123      */
124     public void addLast(E item)
125     {
126                 // special case
127         if (this.isEmpty()) {
128             this.addFirst(item);
129         }
130         else {
131             // find last node
132             Node curr = head;
133             while (curr.next != null) {
134                 curr = curr.next;
135             }
136
137             // attach the new node to the last node
138             Node node = new Node(item);
139             curr.next = node;
140         }
141     }
142
143     /**
144      * Add the given item at the given index
145      * @param index to the node that you are adding
146      * @param item added to the list
147      */
148     public void add(int index, E item)
149     {
```

```
150         Node newNode = new Node(item);
151         newNode = head;
152         for(int i = 0; i< index; i++){
153             head.next = newNode;
154             newNode.previous = head;
155             head = head.next;
156         }
157 
158 
159     }
160     /**
161      * returns the value at the given index
162      * @param index element that was called
163      * @return the element that you call
164      */
165     E get(int index){
166         if(this.isEmpty()){
167             throw new NoSuchElementException();
168         }
169         Node curr = head;
170         int count = 0;
171         while(count != index) {
172             curr = curr.next;
173             count++;
174         }
175 
176 
177     }
178     /**
179      * Replaces the node at the given index in the list
180      * @param index elements in the list
181      * @param items element that you set
182      * @return null
183      */
184 
185     E set(int index, E items){
186         Node curr = head;
187         int count = 0;
188         while(count != index) {
189             curr = curr.next;
190             count++;
191         }
192         E data = curr.data;
193         curr.data = items;
194 
195         return data;
196 
197     }
```

```
198     /**
199      * Determines if the list contains the given item
200      * @param items in the given list
201      * @return true
202      */
203     public boolean contains(E items){
204         Node curr = head.next;
205         while(curr != null) {
206             if(curr.data == items) {
207                 return true;
208             }
209             curr = curr.next;
210         }
211         return false;
212 
213     }
214     /**
215      * clears all elements in the list
216      * return the cleared list
217      */
218 
219     public void clear(){
220         if(isEmpty()) {
221             return;
222         }
223         Node curr = head;
224         Node prev;
225         while(curr != null){
226             prev = curr;
227             prev.data= null;
228             curr = curr.next;
229         }
230         last = head;
231 
232     }
233 
234     // put comment in Javadoc style
235     /**
236      * changing integers to string
237      * return string representation of the list
238      */
239     public String toString()
240     {
241         String str = "";
242         Node curr = head;
243 
244         // add each data item to the result string
245         while (curr != null) {
```

```
246                 str += curr.data + " ";
247                 curr = curr.next;
248         }
249 
250         // remove trailing space and enclose in [ ]
251         str = "[" + str.trim() + "]";
252 
253         return str;
254     }
255     /**
256      * reversing toString method
257      * @return the string
258      */
259     public String toStringRev(){
260         String str = "";
261         Node curr = last;
262         while(curr!= null){
263             str += curr.data + " ";
264             curr = curr.previous;
265         }
266         str = "[" + str.trim() + "]";
267         return str;
268     }
269     /**
270      * Goes through list one item at a time
271      * @return
272      * @return if hasNext is true or false
273      */
274 
275         public DLinkedList1(){
276             DLinkedList<E>.Node goal = head.next;
277 
278         }
279         public boolean hasNext(){
280             if(goal != head){
281                 return true;
282             }else{
283                 return false;
284             }
285 
286         }
287         public E next() {
288             if(hasNext()) {
289 
290                 E result = goal.data;
291                 goal = goal.next;
292                 return result;
293             }else {
```

```
294                 throw new NoSuchElementException();
295             }
296         }
297         public Iterator<E> Iterator() {
298             return new DLinkedList1();
299         }
300     /**
301      * Determines if the list contains the given item
302      * @param items visit the nodes as it searches for the item
303      * @return null
304      */
305 
306     public boolean conatinsIter(E items){
307         Iterator<E> itr = this.Iterator();
308         Node goal = head.next;
309         while(itr.hasNext()) {
310             if(itr.next() == items) {
311                 return true;
312             }
313         }
314         return false;
315 
316     }
317 
318     }
319 
320 }
321 
322 
323 
324 
325 
326 
327 
328 
329 
330 
331 
332 
333 
334 
335 
336 
337 
338 
```