```
 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
40
41    // put comment in Javadoc style
42    public DLinkedList()
43    {
44        Node dummy = new Node(null);
45        head = dummy;
46        last = dummy;
47
48    }
49
50
51
52    // put comment in Javadoc style
53    /**
```

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

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

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

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

```
246         String str = "";
247         Node curr = head.next;
248 
249         // add each data item to the result string
250         while (curr != null) {
251             str += curr.data + " ";
252             curr = curr.next;
253         }
254 
255         // remove trailing space and enclose in [ ]
256         str = "[" + str.trim() + "]";
257 
258         return str;
259     }
260     /**
261      * reversing toString method
262      * @return the string
263      */
264     public String toStringRev(){
265         String str = "";
266         Node curr = last;
267         while(curr!= head){
268             str += curr.data + " ";
269             curr = curr.previous;
270         }
271         str = "[" + str.trim() + "]";
272         return str;
273     }
274 
275     /**
276      * Determines if the list contains the given item
277      * @param items visit the nodes as it searches for the item
278      * @return null
279      */
280 
281     public boolean containsIter(E items) {
282         Iterator<E> itr = this.iterator();
283             Node curr = head.next;
284          while(itr.hasNext()) {
285             if(itr.next() == items) {
286                 return true;
287             }
288             }
289         return false;
290 
291     }
292 
293     /**
```

```
294      * Goes through list one item at a time
295      * @return
296      * @return if hasNext is true or false
297      */
298     private class DLinkedListIterator implements Iterator<E>
299     {
300         // memory
301 
302 
303         public DLinkedListIterator(){
304             Node curr = head.next;
305 
306         }
307         public boolean hasNext(){
308             DLinkedList<E>.Node curr= head.next;
309             if(curr != head){
310                 return true;
311             }else{
312                 return false;
313             }
314 
315         }
316         public E next() {
317             if(hasNext()) {
318 
319                 Node curr = head.next;
320                 E result = curr.data;
321                 curr = curr.next;
322                 return result;
323             }else {
324                 throw new NoSuchElementException();
325             }
326         }
327 
328 
329     }
330 
331     public Iterator<E> iterator() {
332         return new DLinkedListIterator();
333     }
334 
335 }
336 
337 
338 
339 
340 
341 
```

342
343
344
345
346
347
348
349
350
351
352
353