```java
package dataStructures;

public class DoublyLinkedList<E> implements List<E>
{

    static final long serialVersionUID = 0L;


    // Node at the head of the list.
    protected DListNode<E> head;

    // Node at the tail of the list.
    protected DListNode<E> tail;

    // Number of elements in the list.
    protected int currentSize;


    public DoublyLinkedList( )
    {
        head = null;
        tail = null;
        currentSize = 0;
    }


    // Returns true iff the list contains no elements.
    public boolean isEmpty( )
    {
        return currentSize == 0;
    }


    // Returns the number of elements in the list.
    public int size( )
    {
        return currentSize;
    }


    // Returns an iterator of the elements in the list (in proper
```

```
  sequence).
42      public Iterator<E> iterator( )
43      {
44          return new DoublyLLIterator<E>(head, tail);
45      }
46
47
48      // Returns the first element of the list.
49      public E getFirst( ) throws EmptyListException
50      {
51          if ( this.isEmpty() )
52              throw new EmptyListException();
53
54          return head.getElement();
55      }
56
57
58      // Returns the last element of the list.
59      public E getLast( ) throws EmptyListException
60      {
61          if ( this.isEmpty() )
62              throw new EmptyListException();
63
64          return tail.getElement();
65      }
66
67
68      // Returns the node at the specified position in the list.
69      // Pre-condition: position ranges from 0 to currentSize-1.
70      protected DListNode<E> getNode( int position )
71      {
72          DListNode<E> node;
73
74          if ( position <= ( currentSize - 1 ) / 2 )
75          {
76              node = head;
77              for ( int i = 0; i < position; i++ )
78                  node = node.getNext();
79          }
80          else
81          {
```

```
82              node = tail;
83              for ( int i = currentSize - 1; i > position; i-- )
84                  node = node.getPrevious();
85
86          }
87          return node;
88      }
89
90
91      // Returns the element at the specified position in the list.
92      // Range of valid positions: 0, ..., size()-1.
93      // If the specified position is 0, get corresponds to getFirst.
94      // If the specified position is size()-1, get corresponds to
    getLast.
95      public E get( int position ) throws InvalidPositionException
96      {
97          if ( position < 0 || position >= currentSize )
98              throw new InvalidPositionException();
99
100         return this.getNode(position).getElement();
101     }
102
103
104     // Returns the position of the first occurrence of the
    specified element
105     // in the list, if the list contains the element.
106     // Otherwise, returns -1.
107     public int find( E element )
108     {
109         DListNode<E> node = head;
110         int position = 0;
111         while ( node != null && !
    node.getElement().equals(element) )
112         {
113             node = node.getNext();
114             position++;
115         }
116         if ( node == null )
117             return -1;
118         else
119             return position;
```

```
120     }
121
122
123     // Inserts the specified element at the first position in the
    list.
124     public void addFirst( E element )
125     {
126         DListNode<E> newNode = new DListNode<E>(element, null,
    head);
127         if ( this.isEmpty() )
128             tail = newNode;
129         else
130             head.setPrevious(newNode);
131         head = newNode;
132         currentSize++;
133     }
134
135
136     // Inserts the specified element at the last position in the
    list.
137     public void addLast( E element )
138     {
139         DListNode<E> newNode = new DListNode<E>(element, tail,
    null);
140         if ( this.isEmpty() )
141             head = newNode;
142         else
143             tail.setNext(newNode);
144         tail = newNode;
145         currentSize++;
146     }
147
148
149     // Inserts the specified element at the specified position in
    the list.
150     // Pre-condition: position ranges from 1 to currentSize-1.
151     protected void addMiddle( int position, E element )
152     {
153         DListNode<E> prevNode = this.getNode(position - 1);
154         DListNode<E> nextNode = prevNode.getNext();
155         DListNode<E> newNode = new DListNode<E>(element, prevNode,
```

```
    nextNode);
156         prevNode.setNext(newNode);
157         nextNode.setPrevious(newNode);
158         currentSize++;
159     }
160 
161 
162     // Inserts the specified element at the specified position in
    the list.
163     // Range of valid positions: 0, ..., size().
164     // If the specified position is 0, add corresponds to addFirst.
165     // If the specified position is size(), add corresponds to
    addLast.
166     public void add( int position, E element ) throws
    InvalidPositionException
167     {
168         if ( position < 0 || position > currentSize )
169             throw new InvalidPositionException();
170 
171         if ( position == 0 )
172             this.addFirst(element);
173         else if ( position == currentSize )
174             this.addLast(element);
175         else
176             this.addMiddle(position, element);
177     }
178 
179 
180     // Removes the first node in the list.
181     // Pre-condition: the list is not empty.
182     protected void removeFirstNode( )
183     {
184         head = head.getNext();
185         if ( head == null )
186             tail = null;
187         else
188             head.setPrevious(null);
189         currentSize--;
190     }
191 
192 
```

```
193     // Removes and returns the element at the first position in the
    list.
194     public E removeFirst( ) throws EmptyListException
195     {
196         if ( this.isEmpty() )
197             throw new EmptyListException();
198 
199         E element = head.getElement();
200         this.removeFirstNode();
201         return element;
202     }
203 
204 
205     // Removes the last node in the list.
206     // Pre-condition: the list is not empty.
207     protected void removeLastNode( )
208     {
209         tail = tail.getPrevious();
210         if ( tail == null )
211             head = null;
212         else
213             tail.setNext(null);
214         currentSize--;
215     }
216 
217 
218     // Removes and returns the element at the last position in the
    list.
219     public E removeLast( ) throws EmptyListException
220     {
221         if ( this.isEmpty() )
222             throw new EmptyListException();
223 
224         E element = tail.getElement();
225         this.removeLastNode();
226         return element;
227     }
228 
229 
230     // Removes the specified node from the list.
231     // Pre-condition: the node is neither the head nor the tail of
```

```
   the list.
232     protected void removeMiddleNode( DListNode<E> node )
233     {
234         DListNode<E> prevNode = node.getPrevious();
235         DListNode<E> nextNode = node.getNext();
236         prevNode.setNext(nextNode);
237         nextNode.setPrevious(prevNode);
238         currentSize--;
239     }
240 
241 
242     // Removes and returns the element at the specified position in
   the list.
243     // Range of valid positions: 0, ..., size()-1.
244     // If the specified position is 0, remove corresponds to
   removeFirst.
245     // If the specified position is size()-1, remove corresponds to
   removeLast.
246     public E remove( int position ) throws InvalidPositionException
247     {
248         if ( position < 0 || position >= currentSize )
249             throw new InvalidPositionException();
250 
251         if ( position == 0 )
252             return this.removeFirst();
253         else if ( position == currentSize - 1 )
254             return this.removeLast();
255         else
256         {
257             DListNode<E> nodeToRemove = this.getNode(position);
258             this.removeMiddleNode(nodeToRemove);
259             return nodeToRemove.getElement();
260         }
261     }
262 
263 
264     // Returns the node with the first occurrence of the specified
   element
265     // in the list, if the list contains the element.
266     // Otherwise, returns null.
267     protected DListNode<E> findNode( E element )
```

```
268     {
269         DListNode<E> node = head;
270         while ( node != null && !
    node.getElement().equals(element) )
271             node = node.getNext();
272         return node;
273     }
274 
275 
276     // Removes the first occurrence of the specified element from
    the list
277     // and returns true, if the list contains the element.
278     // Otherwise, returns false.
279     public boolean remove( E element )
280     {
281         DListNode<E> node = this.findNode(element);
282         if ( node == null )
283             return false;
284         else
285         {
286             if ( node == head )
287                 this.removeFirstNode();
288             else if ( node == tail )
289                 this.removeLastNode();
290             else
291                 this.removeMiddleNode(node);
292             return true;
293         }
294     }
295 
296 
297     // Removes all of the elements from the specified list and
298     // inserts them at the end of the list (in proper sequence).
299     public void append( DoublyLinkedList<E> list )
300     {
301 
302         if(!list.isEmpty() && currentSize==0 )
303              head=list.head;
304 
305         if (!list.isEmpty()){
306              tail.setNext(list.head);
```

```
307             list.head.setPrevious(tail);
308         }
309             tail=list.tail;
310             currentSize=+list.currentSize;
311 
312         }
313 
314 
315 
316 }
317 
318 
319 
```