

Lecture 9: Jun. 19, 2015

Lecturer: Anwar Mamat

Disclaimer: *These notes may be distributed outside this class only with the permission of the Instructor.*

9.1 Doubly Linked List

Like a singly linked list, a doubly-linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Unlike a singly linked list, each node of the doubly singly list contains two fields that are references to the previous and to the next node in the sequence of nodes. The beginning and ending nodes' previous and next links, respectively, point to some kind of terminator, typically a sentinel node or null, to facilitate traversal of the list.

Listing 1: Doubly Linked List Node Class

```
1 class Node{
2     E data;
3     Node previous;
4     Node next;
5     Node(E item){
6         data = item;
7         next = null;
8         previous = null;
9     }
10    Node(){
11        data = null;
12        next = null;
13        previous = null;
14    }
15 }
```

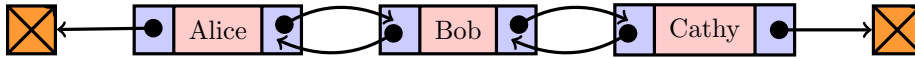
Usually Node class is nested inside the LinkedList class, and members of Node are private.

9.1.1 Create a simple linked list

Now, let us create a simple linked list.

```
1 Node<String> n1 = new Node(" Alice" );
2 Node<String> n2 = new Node("Bob" );
3 Node<String> n3 = new Node(" Cathy" );
4 n1.next = n2;
5 n2.previous = n1;
6 n2.next = n3;
7 n3.previous = n2;
```

This linked list represents this:



9.1.2 Display the Linked List

We can display all the linked list:

```

1 Node<String> current = first;
2 while(current != null){
3     System.out.println(current.data);
4     current = current.next;
5 }

```

We can also display all the linked list in reverse order:

```

1 Node<String> current = tail;
2 while(current != null){
3     System.out.println(current.data);
4     current = current.previous;
5 }

```

9.1.3 Insert a node

Now, let us insert a node between “Bob” and “Cathy”.

```

1 Node<String> n4 = new Node("Ethan");
2 n4.next = n2.next;
3 n4.previous = n2;
4 n2.next = n4;
5 n3.previous = n4;
6 //use "first" to reference the first node of the list.
7 Node<String> first = n1;

```

This linked list represents this:



9.1.4 Delete a node

In order to delete the node “Bob” reference by “current”, we can do this:

```

1 current.previous.next = current.next;
2 current.next.previous = current.previous;

```

No, we have:



9.2 Doubly Linked List Class

```

1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package doublylinkedlist;
6
7  import java.util.Iterator;
8  import java.util.ListIterator;
9  import java.util.NoSuchElementException;
10
11 /**
12  *
13  * @author anwar
14  */
15 public class DoublyLinkedList<E> implements Iterable<E>{
16     private int N; // number of nodes
17     private Node head; //sentinel before the first node
18     private Node tail; //sentinel after the last node;
19     DoublyLinkedList(){
20         head = new Node();
21         tail = new Node();
22         head.next = tail;
23         tail.previous = head;
24     }
25
26     @Override
27     public ListIterator<E> iterator() {
28         return new DoublyListIterator();
29     }
30
31     private class Node{
32         private E data;
33         private Node previous;
34         private Node next;
35         Node(E item){
36             data = item;
37             next = null;
38             previous = null;
39         }
40         Node(){
41             data = null;
42             next = null;
43             previous = null;
44         }

```

```

45     }
46     public int size(){return N;}
47     public boolean isEmpty(){ return N==0;}
48
49     public void insert(E item){
50         Node last = tail.previous;
51         Node t = new Node(item);
52         t.next = tail;
53         t.previous = last;
54         tail.previous = t;
55         last.next = t;
56         N++;
57     }
58
59     public String toString(){
60         StringBuilder s = new StringBuilder();
61         Node current = head.next;
62         while(current != tail){
63             s.append(current.data+",");
64             current = current.next;
65         }
66         return s.toString();
67     }
68
69     private class DoublyListIterator implements ListIterator<E>{
70         private int index = 0;
71         private Node current;
72         private Node lastAccessed;
73         DoublyListIterator(){
74             current = head.next;
75             lastAccessed = null;
76             index = 0;
77         }
78
79         @Override
80         public boolean hasNext() {
81             return index < N;
82         }
83
84         @Override
85         public E next() {
86             if(!hasNext()){
87                 throw new NoSuchElementException();
88             }
89             lastAccessed = current;
90             E item = current.data;
91             current = current.next;
92             index++;
93             return item;
94         }
95     }

```

```
96
97     @Override
98     public boolean hasPrevious() {
99         return index > 0;
100     }
101
102     @Override
103     public E previous() {
104         if (!hasPrevious()) {
105             throw new NoSuchElementException();
106         }
107         current = current.previous;
108         lastAccessed = current;
109         index--;
110         return current.data;
111     }
112
113     @Override
114     public int nextIndex() {
115         return index;
116     }
117
118     @Override
119     public int previousIndex() {
120         return index - 1;
121     }
122
123     @Override
124     public void remove() {
125         Node a = lastAccessed.previous;
126         Node b = lastAccessed.next;
127         a.next = b;
128         b.previous = a;
129         N--;
130         index--;
131         lastAccessed = null;
132     }
133
134     @Override
135     public void set(E e) {
136         throw new UnsupportedOperationException("Not supported yet.");
137     }
138
139     @Override
140     public void add(E e) {
141         Node b = new Node(e);
142         Node a = current.previous;
143         Node c = current;
144         a.next = b;
145         b.next = c;
146         c.previous = b;
```

```

147         b.previous = a;
148         index++;
149         N++;
150         lastAccessed = null;
151     }
152 }
153 }
154
155 /**
156  * @param args the command line arguments
157  */
158 public static void main(String[] args) {
159     DoublyLinkedList<Integer> dl = new DoublyLinkedList();
160     ListIterator<Integer> li;
161     for(int i = 2; i <= 6; i++){
162         dl.insert(i);
163     }
164     li = dl.iterator();
165     for(int i = 10; i <= 15; i++){
166         li.add(i);
167     }
168     //print using toString()
169     System.out.println(dl);
170     System.out.println("\n");
171     //print using foreach
172     for(Integer i: dl){
173         System.out.print(i+",");
174     }
175     System.out.println("\n");
176     //print using iterator
177     li = dl.iterator();
178     while(li.hasNext()){
179         int t = li.next();
180         System.out.print(t+",");
181     }
182     //print using iterator in reverse order
183     System.out.println("\n");
184     while(li.hasPrevious()){
185         int t = li.previous();
186         //if(t == 3)
187         System.out.print(t+",");
188         //if(t % 2 == 0) li.remove();
189     }
190     System.out.println("\n");
191 }
192 }

```