

Lecture 8:

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

8.1 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<E>{
2     E data;
3     Node previous;
4     Node next;
5     Node(E item){
6         data = item;
7     }
8 }

```

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

8.1.1 Create a simple linked list

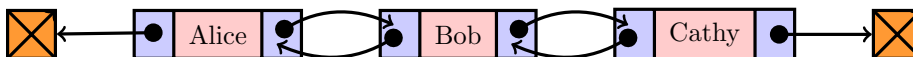
Now, let us create a simple linked list.

```

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:



8.1.2 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 }

```

8.1.3 Insert a node

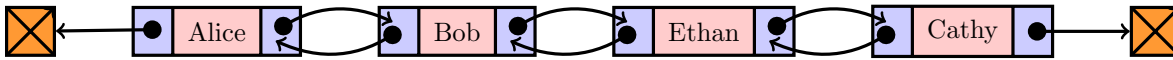
Now, let us insert a node between “Bob” and “Cathy”.

```

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:



8.1.4 Delete a node

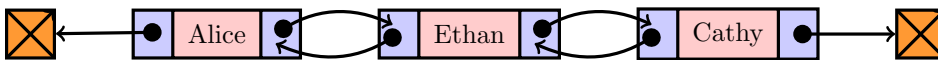
In order to delete the node “Bob” reference by “current”, we can do this:

```

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

```

No, we have:



8.2 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      }
41      public int size(){return N;}
42      public boolean isEmpty(){ return N==0;}
43
44      public void insert(E item){
45          Node last = tail.previous;
46          Node t = new Node(item);
47          t.next = tail;
48          t.previous = last;
49          tail.previous = t;
50          last.next = t;
51          N++;
52      }
53
54      public String toString(){
55          StringBuilder s = new StringBuilder();
56          Node current = head.next;
57          while(current != tail){
58              s.append(current.data+", ");
59              current = current.next;
60          }
61          return s.toString();
62      }
63
64      private class DoublyListIterator implements ListIterator<E>{
65          private int index = 0;
66          private Node current;
67          private Node lastAccessed;
68          DoublyListIterator(){
69              current = head.next;
70              lastAccessed = null;
71              index = 0;
72          }
73
74          @Override
75          public boolean hasNext() {
76              return index < N;

```

```
77     }
78
79     @Override
80     public E next() {
81         if(!hasNext()){
82             throw new NoSuchElementException();
83
84         }
85         lastAccessed = current;
86         E item = current.data;
87         current = current.next;
88         index++;
89         return item;
90     }
91
92     @Override
93     public boolean hasPrevious() {
94         return index > 0;
95     }
96
97     @Override
98     public E previous() {
99         if(!hasPrevious()){
100             throw new NoSuchElementException();
101         }
102         current = current.previous;
103         lastAccessed = current;
104         index--;
105         return current.data;
106     }
107
108     @Override
109     public int nextIndex() {
110         return index;
111     }
112
113     @Override
114     public int previousIndex() {
115         return index - 1;
116     }
117
118     @Override
119     public void remove() {
120         Node a = lastAccessed.previous;
121         Node b = lastAccessed.next;
122         a.next = b;
123         b.previous = a;
124         N--;
125         index--;
126         lastAccessed = null;
127     }
128
129     @Override
130     public void set(E e) {
131         throw new UnsupportedOperationException("Not_supported_yet.");
132     }
133
134     @Override
135     public void add(E e) {
136         Node b = new Node(e);
137         Node a = current.previous;
138         Node c = current;
139         a.next = b;
140         b.next = c;
141         c.previous = b;
142         b.previous = a;
143         index++;
144         N++;
```

```
145         lastAccessed = null;
146     }
147 }
148 }
149 }
150 /**
151  * @param args the command line arguments
152  */
153 public static void main(String[] args) {
154     DoublyLinkedList<Integer> dl = new DoublyLinkedList();
155     ListIterator<Integer> li;
156     for(int i = 2; i <= 6; i++){
157         dl.insert(i);
158     }
159     li = dl.iterator();
160     for(int i = 10; i <= 15; i++){
161         li.add(i);
162     }
163     //print using toString()
164     System.out.println(dl);
165     System.out.println("\n");
166     //print using foreach
167     for(Integer i: dl){
168         System.out.print(i+", ");
169     }
170     System.out.println("\n");
171     //print using iterator
172     li = dl.iterator();
173     while(li.hasNext()){
174         int t = li.next();
175         System.out.print(t+", ");
176     }
177     //print using iterator in reverse order
178     System.out.println("\n");
179     while(li.hasPrevious()){
180         int t = li.previous();
181         //if(t == 3)
182         System.out.print(t+", ");
183         //if(t % 2 ==0) li.remove();
184     }
185     System.out.println("\n");
186 }
187 }
```