CIS 2168 Data Structures **Fall 2014**

Lecture 5: Sep. 23 &25

Lecturer: Anwar Mamat

5.1 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

```
class Node{
        E data;
        Node previous;
        Node next;
        Node(E item){
            data = item;
            next = null;
            previous = null;
        }
        Node(){
            data = null;
            next = null;
            previous = null;
        }
    }
```

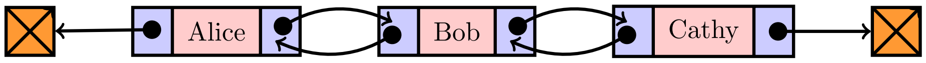
Usually Node class is nested inside the LinkedList class, and members of Node are private.

5.1.1 Create a simple linked list

Now, let us create a simple linked list.

```
Node<String> n1 = new Node("Alice");
Node<String> n2 = new Node("Bob");
Node<String> n3 = new Node("Cathy");
n1.next = n2;
n2.previous = n1;
n2.next = n3;
n3.previous = n2;
```

This linked list represents this:



5.1.2 Display the Linked List

We can display all the linked list:

```
Node<String> current = first;
while(current != null){
        System.out.println(current.data);
        current = current.next;
}
```

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

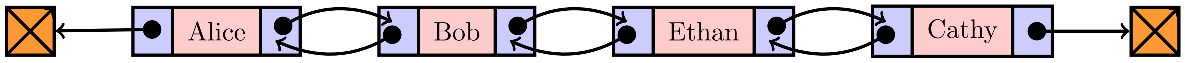
```
Node<String> current = tail;
while(current != null){
        System.out.println(current.data);
        current = current.previous;
}
```

5.1.3 Insert a node

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

```
Node<Stirng> n4 = new Node("Ethan");
n4.next = n2.next;
n4.previous = n2;
n2.next = n4;
n3.previous = n4;
//use "first" to reference the first node of the list.
Node<String> first = n1;
```

This linked list represents this:

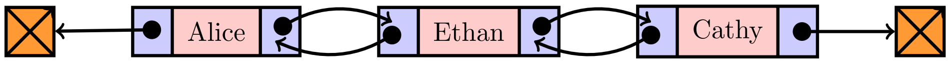


5.1.4 Delete a node

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

```
current.previous.next = current.next;
current.next.previous = current.previous;
```

No, we have:



5.2 Doubly Linked List Class

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package doublylinkedlist;

import java.util.Iterator;
import java.util.ListIterator;
import java.util.NoSuchElementException;

/**
 *
 * @author anwar
 */
public class DoublyLinkedList<E> implements Iterable<E>{
    private int N;  // number of nodes
    private Node head;  //sentinel before the first node
    private Node tail;  //sentinel aftet the last node;
    DoublyLinkedList(){
        head = new Node();
        tail = new Node();
        head.next = tail;
        tail.previous = head;
    }

    @Override
    public ListIterator<E> iterator() {
        return new DoublyListIterator();
    }

    private class Node{
        private E data;
        private Node previous;
        private Node next;
        Node(E item){
            data = item;
            next = null;
            previous = null;
        }
        Node(){
            data = null;
            next = null;
            previous = null;
        }
    }
    public int size(){return N;}
    public boolean isEmpty(){ return N==0;}

    public void insert(E item){
        Node last = tail.previous;
        Node t = new Node(item);
        t.next = tail;
        t.previous = last;
        tail.previous = t;
        last.next = t;
        N++;
    }

    public String toString(){
        StringBuilder s  = new StringBuilder();
        Node current = head.next;
        while(current != tail){
            s.append(current.data+",");
            current = current.next;
        }
        return s.toString();
    }

    private class DoublyListIterator implements ListIterator<E>{
        private int index = 0;
        private Node current;
        private Node lastAccessed;
        DoublyListIterator(){
            current = head.next;
            lastAccessed = null;
            index = 0;
        }

        @Override
        public boolean hasNext() {
            return index < N;
        }

        @Override
        public E next() {
            if(!hasNext()){
                throw new NoSuchElementException();

            }
            lastAccessed = current;
            E item = current.data;
            current = current.next;
            index++;
            return item;
        }

        @Override
        public boolean hasPrevious() {
            return index > 0;
        }

        @Override
        public E previous() {
            if(!hasPrevious()){
                throw new NoSuchElementException();
            }
            current = current.previous;
            lastAccessed = current;
            index--;
            return current.data;
        }

        @Override
        public int nextIndex() {
            return index;
        }

        @Override
        public int previousIndex() {
            return index - 1;
        }

        @Override
        public void remove() {
            Node a = lastAccessed.previous;
            Node b = lastAccessed.next;
            a.next = b;
            b.previous = a;
            N--;
            index--;
            lastAccessed = null;
        }

        @Override
        public void set(E e) {
            throw new UnsupportedOperationException("Not supported yet.");
        }

        @Override
        public void add(E e) {
            Node b = new Node(e);
            Node a = current.previous;
            Node c = current;
            a.next = b;
            b.next = c;
            c.previous = b;
```

```
                b.previous = a;
                index++;
                N++;
                lastAccessed = null;
            }

        }

        /**
         * @param args the command line arguments
         */
        public static void main(String[] args) {
            DoublyLinkedList<Integer> dl = new DoublyLinkedList();
            ListIterator<Integer> li;
            for(int i = 2; i <= 6; i++){
                dl.insert(i);
            }
            li = dl.iterator();
            for(int i = 10; i <= 15; i++){
                li.add(i);
            }
            //print using toString()
            System.out.println(dl);
            System.out.println("\n");
            //print using foreach
            for(Integer i: dl){
                System.out.print(i+",");
            }
            System.out.println("\n");
            //print using iterator
            li = dl.iterator();
            while(li.hasNext()){
                int t = li.next();
                System.out.print(t+",");
            }
            //print using iterator in reverse order
            System.out.println("\n");
            while(li.hasPrevious()){
                int t = li.previous();
                //if(t == 3)
                System.out.print(t+",");
                //if(t % 2 ==0) li.remove();
            }
            System.out.println("\n");
        }
}
```