CMSC 132, Object-Oriented Programming II **Summer 2018**

Lecture 6:

Lecturer: Anwar Mamat

6.1 Singly Linked List

A linked list is a data structure consisting of a group of nodes which together represent a sequence. Each node is composed of a data and a reference (in other words, a link) to the next node in the sequence. A Node class usually look like this:

Listing 1: Singly Linked List Node

```
1 class Node<E> {
2   public E data;
3   public Node<E> next;
4   Node(E item){
5     data = item;
6   }
7 }
```

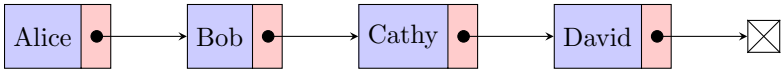
Usually Node class is nested inside the LinkedList class, and members of Node are private.

6.1.1 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 Node<String> n4 = new Node("David");
5 n1.next = n2;
6 n2.next = n3;
7 n3.next = n4;
```

This linked list represents this:



6.1.2 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 }
```

Here is the recursive version of the same code

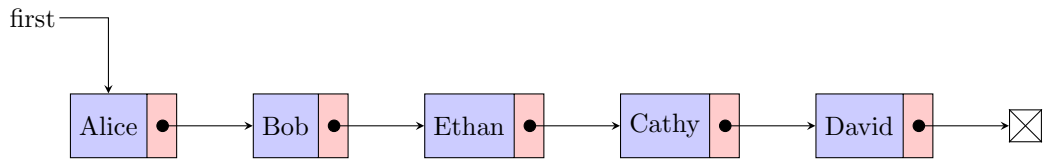
```
1 public void print(){
2         print(first);
3         System.out.println("");
4 }
5 private void print(Node r){
6         if(r == null) return;
7         System.out.print(r.data+",");
8         print(r.next);
9 }
```

6.1.3 Insert a node

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

```
1 Node<Stirng> n5 = new Node("Ethan");
2 n5.next = n2.next;
3 n2.next = n5;
4 //use "first" to reference the first node of the list.
5 Node<String> first = n1;
```

This linked list represents this:



6.1.4 Delete a node

6.1.4.1 Delete the first node

To delete the first node, we can simply move "first" to next node.

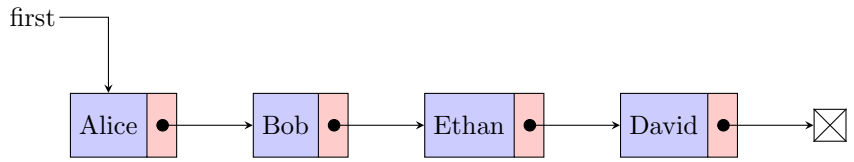
```
1 first = first.nex;
```

6.1.4.2 Delete other nodes

In order to delete a Node, we have to know the parent of the node. Assume "parent" references the node "Ethan", to delete the node "Cathy" reference by "current", we ca do this:

```
1 parent.next = current.next;
```

No, we have:



6.2 Linked List Class

```
1  /**
2   *  The Bag class represents a collection of generic items.
3   *  It supports insertion and iterating over the items in arbitrary order.
4   */
5
6  import java.util.ArrayList;
7  import java.util.Iterator;
8
9  public class LinkedBag<E extends Comparable<E>> implements Iterable<E>
10 {
11         protected int N;  //number of items in the bag
12         private Node<E> first;    //beginning of bag
13
14         // helper linked list class
15         private class Node<E> {
16             private E data;
17             private Node<E> next;
18             Node(E item){
19                 data = item;
20             }
21         }
22
23         /**
24         * Initializes an empty bag.
25         */
26         public LinkedBag() {
27             first = null;
28             N = 0;
29         }
30         /**
31         *   Returns an iterator that iterates through the items in the bag
32         * @return an iterator that iterates through the items in the bag
33         */
34         public Iterator<E> iterator() {
35             return new BagIterator(first);
36         }
37         /**
38         *   The iterator implementation
39         */
40         private class BagIterator implements Iterator<E> {
41             private Node<E> current = null;
42             public BagIterator(Node<E> first) {
43                 current = first;
44             }
45             public boolean hasNext()  { return current != null;}
46             public void remove()   { System.out.println("to be implemented.");  }
47             public E next() {
48                 if(!hasNext()) {return null;}
49                 E item = current.data;
50                 current = current.next;
51                 System.out.println("work");
52                 return item;
53             }
54
55         }
56
57         /**
58         * Adds the item to this bag.
59         * @param item the item to add to this bag
60         */
61         public void insert(E item) {
62             Node<E> oldfirst = first;
63             first = new Node<E>(item);
64             first.next = oldfirst;
65             N++;
```

```
66          }
67
68          /**
69          * Returns an item by index
70          * @param index is the item index
71          */
72          public E get(int index)
73          {
74              Node<E> current = first;
75              int i = 0;
76              while(current != null && i < index){
77                  current = current.next;
78                  i++;
79              }
80              if(current != null){
81                  return current.data;
82             }else{
83                  return null;
84             }
85          }
86          public boolean remove(E item){
87                  if(item == null) return false;
88                  if(isEmpty()) throw new NoSuchElementException();
89              Node<E> current = first;
90              Node<E> parent = first;
91              while(current != null){
92                  if(current.data.equals(item)){
93                          parent.next = current.next;
94                      return true;
95                  }
96                  parent = current;
97                  current = current.next;
98              }
99              return false;
100         }
101         /**
102     * Deletes an item recursively
103     * @param item is the item to be deleted
104     * @return true if item is deleted. false otherwise
105     */
106         public void remove_rec(E item){
107           first = remove_rec(first, item);
108         }
109
110         /**
111     * Deletes an item recursively
112     * @param item is the item to be deleted
113     * @param r is the starting node
114     * @return true if item is deleted. false otherwise
115     */
116
117         private Node<E> remove_rec(Node<E> r, E item){
118           if(r == null) return null;
119           if(r.data.equals(item)){
120             return r.next;
121           }
122           r.next = remove_rec(r.next, item);
123           return r;
124         }
125
126         /**
127         * Is this bag empty?
128         * @return true if this bag is empty; false otherwise
129         */
130         public boolean isEmpty() {
131             return first == null;
132         }
133
```

```
134         /**
135         * Returns the number of items in this bag.
136         * @return the number of items in this bag
137         */
138         public int size() {
139             return N;
140         }
141 
142         /**
143         *   if the bag contains a given item?
144         * @return true if bag contains the item. false otherwise
145         */
146         public boolean contains(E item)
147         {
148             Node<E> current = first;
149             while(current != null){
150                 if(current.data.equals(item)) return true;
151                 current = current.next;
152             }
153             return false;
154 
155         }
156  }
```

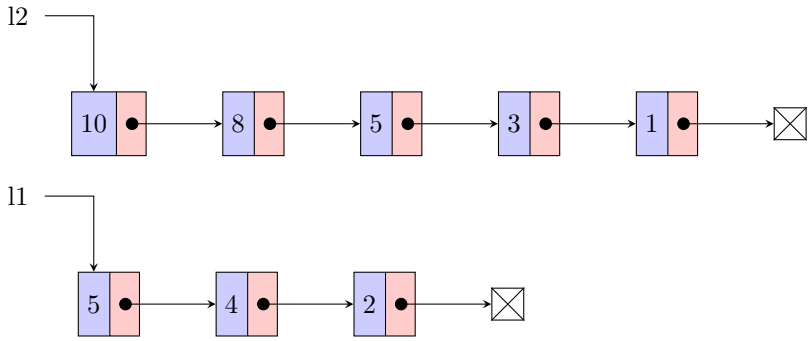
6.2.1 Test the Linked List

```
1  /**
2   * test Linked List Bag
3  */
4  public class LinkedBagUnitTest {
5      public static void main(String[] args){
6          LinkedBag<Integer> bag = new LinkedBag();
7          for(int i=1; i <= 3; i++){
8              bag.insert(i);
9          }
10         System.out.println("Size="+bag.size());
11         if(bag.contains(3)){
12             System.out.println("Bag␣contains␣3");
13         }else{
14             System.out.println("Not␣Found");
15         }
16         //print all items using iterator
17         for(Integer i:bag){
18             System.out.print(i + ",");
19         }
20         //print all items using get method, which is not efficient.
21         System.out.println("\n␣all␣items");
22         for(int i = 0; i < bag.size(); i++){
23             System.out.print(bag.get(i)+",");
24         }
25     }
26 }
```

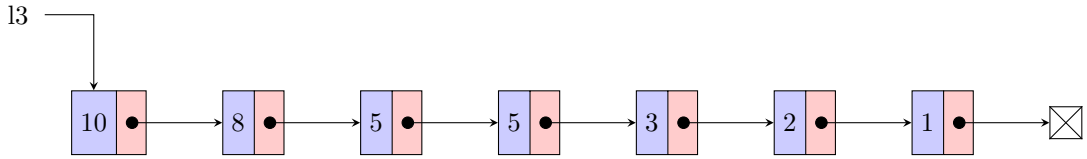
6.3 Code example

6.3.1 Merge two sorted linked list into one

We have two sorted linked lists *list*1 and *list*2.



We want to generate the list:



Here is the code that takes two lists as input, and merges them into one list. This function takes $O(n1+n2)$ time to merge two lists of size $n1$ and $n2$.

```
1  public Node merge(Node l1, Node l2){
2
3          //if one list is empty, return the other list
4          if(l1 == null){
5              return l2;
6          }
7          if(l2 == null){
8              return l1;
9          }
10         //if both lists are not empty
11         Node c1 = l1;
12         Node c2 = l2;
13         Node m = null;
14         //pick the larger node from l1 and l2.
15         if(c1.data > c2.data){
16             m = c1;
17             c1 = c1.next;
18         }else{
19             m = c2;
20             c2 = c2.next;
21         }
22         /**
23         *       walk through l1 and l2, every time pick the larger node.
24         *       comparison only occurs at the head of two lists.
25         */
26         Node c3 = m;
27         while(c1 != null && c2 != null){
28             if(c1.data > c2.data){
29                 c3.next = c1;
30                 c1 = c1.next;
31                 c3 = c3.next;
32             }else{
33                 c3.next = c2;
34                 c2 = c2.next;
35                 c3 = c3.next;
36             }
37
38         }
39
40         if(c1 != null){
```

```
41              c3.next = c1;
42          }else{
43              c3.next = c2;
44          }
45          return m;
46      }
```

The recursive version

```
1  public Node merge_rec(Node l1, Node l2){
2          if(l1 == null) return l2;
3          if(l2 == null) return l1;
4          if(l1.data > l2.data){
5            l1.next  = merge2(l1.next,l2);
6            return l1;
7          }else{
8            l2.next  = merge2(l1,l2.next);
9            return l2;
10         }
11 }
```