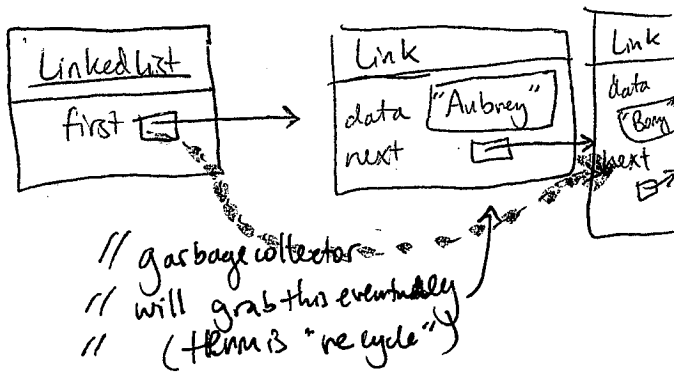
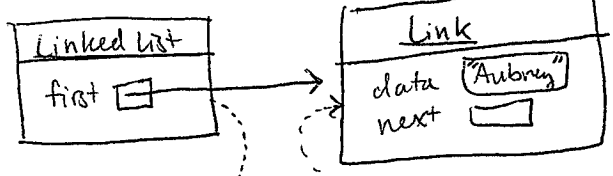
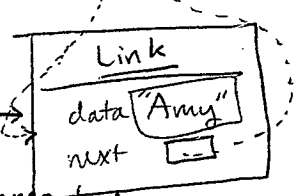
IMPLEMENTING LINKED LISTS

Keeping It Simple: Singly linked list, not all methods

```
1  import java.util.NoSuchElementException;
2
3  /**
4     A linked list is... a sequence of links with efficient element
5     insertion & removal. This class contains a subset of the methods of the
6     standard java.util.LinkedList class.
7  */
8  public class LinkedList
9  {
10    /**
11       constructs an empty linked list
12    */
13    public LinkedList()
14    {
15          first = null;
16    }
17
18    /**
19       Returns the first element in the linked list.
20       @return the first element in the linked list //return value notation
21    */
22    public Object getFirst() //grabs element from inside list (see next page!!)
23    {
24          if(first == null)
25                throw new NoSuchElementException(); //thrown if there are no elements in the linked list
26          return first.data;  //returns data, not link reference
27    }
28
29
30    /**
31       Removes the first element in the linked list.
32       @return the removed element
33    */
34    public Object removeFirst()
35    {
36          if(first == null)
37                throw new NoSuchElementException();
38          Object obj = first.data;
39          first = first.next;
40          return obj;
41    }
42
43    /**
44       Adds an element to the front of the linked list.
45       @param obj the object to add //notes the parameter
46    */
47    public void addFirst(Object obj) //makes a new head of the list
48    {
49          Link newLink = new Link();
50          newLink.data = obj;
51          newLink.next = first;
52          first = newLink;
53    }
54
55    /**
56       Returns an iterator for iterating through the list.
57       @return iterator for iterating through the list
58    */
59    public ListIterator listIterator()
60    {
61          return new linkedListIterator();
62    }
```





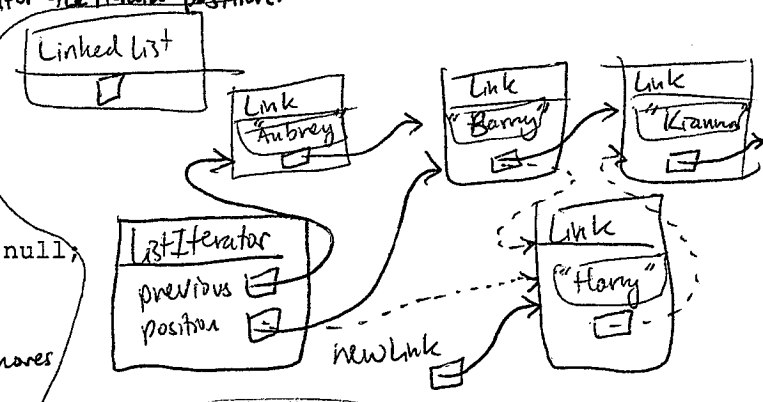


// inner class
// has 9 methods (standard)

// clients of LinkedList class don't know the iterator's name,
// just that it is implemented

→ // see private inner class

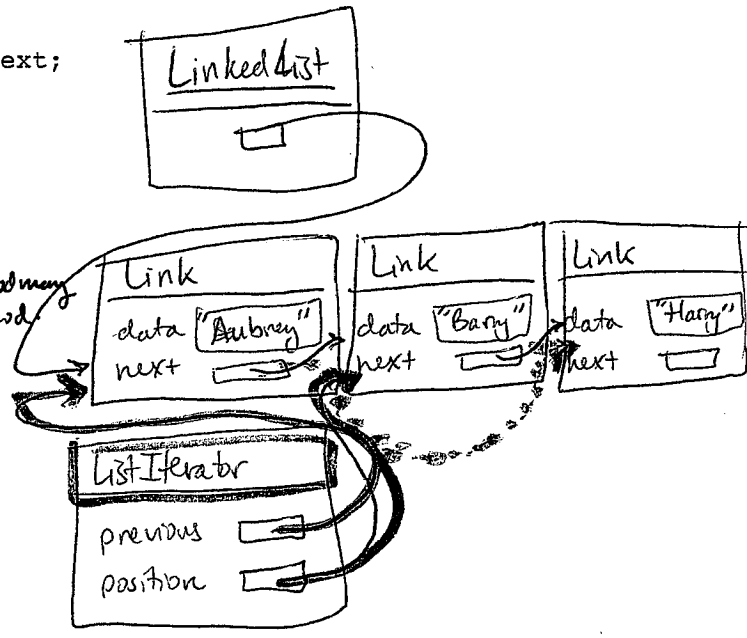
```
63
64  private Link first; // Stores the reference to the first element!! or nlll if empty list
65
66  private class Link    //private inner class
67  {
68        public Object data;   // these variables in the link are public because LinkedList class +
69        public Link next;     //iterator class use them so much!
70  }
71
72  private class LinkedListIterator()          // inner classes!!
73        implements ListIterator               // has current position + previous position → next page!
74  {                                           → previous page, made a new one + it's methods can be used
75        /**
76          Constructs an iterator that points to the front
77          of the linked list.
78        */
79        public LinkedListIterator()
80        {
81              position = null;
82              previous = null;
83        }   //see these variables initialized @ the end ↻
84
85        /**
86        Moves the iterator past the next element.
87          @ return the traversed element
88        */
89        public Object next()   //can only be called before iterator reaches the end of the list
90        {
91              if(!hasNext())
92                    throw new NoSuchElementException();
93              previous = position; //remember for remove
94
95              if(position == null)
96                    position = first;
97              else
98                    position = position.next;//advance to next
99
100             return position.data;
101       } //ex. if old position is null, then position has to be set to first (before
102                                                   setting 1st element)
103       /**
104           Tests whether there is an element after the iterator position
105           @ return true if there is an element after the iterator position.
106
107
108       */
109       public boolean hasNext()
110       {
111             if(position == null)
112                   return first != null;
113             else
114                   return position.next != null;
115       }
116
117       /**
118       Adds an element before the iterator position + moves
119       the iterator past the inserted element.
120         @ param obj the object to add
121       */
122       public void add(Object obj)
123       {
124             if(position == null)
125             {
126                   addFirst(obj); //sets the first element if it needs to
```



// most complex operation right here!
// set instance positions / references

see page 748

```
127                     position = first;
128               }
129               else
130               {
131                     Link newLink = newLink();
132                     newLink.data = obj;
133                     newLink.next = position.next;
134                     position.next = newLink;
135                     position = newLink;
136               }
137               previous = null;
138         }
139
140         /**
141           Removes the last traversed element. This method may
142           be called only after a call to the next() method.
143         */
144         public void remove()
145         {
146               if(position == first)
147               {
148                     removeFirst();
149                     position = first;
150               }
151               else
152               {
153                     if(previous == null)
154                           throw new IllegalStateException();
155                     previous.next = position.next;
156                     position = previous;
157               }
158               previous = null;        // remember, can't call remove() twice in a row
159         }
160
161         /**
162           Sets the last traversed element to a different value
163
164           @param obj the object to set
165         */
166         public void set(Object obj)   // can't call after add or remove, because the iterator's muts
167         {                             // are being tracked in the library
168               if(position == null)
169                     throw new NoSuchElementException();
170               position.data = obj;    // the restriction of calling set after add or remove
171         }                             // is not enforced in this code
172
173         private Link position;
174         private Link previous;
175   }
176 }
```



~~Most code~~

File ListIterator.java

```
1  import java.util.NoSuchElementException;
2
3  /**
4     A linked iterator... allows access to a position in a linked list. This interface
5     contains a subset of the methods of the standard java.util.ListIterator interface.
6     the methods for backward traversal are not included
7  */
8  public interface ListIterator
9  {
10    /**
11       moves iterator past the next element.
12       @return the traversed element
13    */
14    Object next();
15
16    /**
17       Tests whether there is an element after the iterator position
18       @return true if there is an element after the iterator position
19    */
20    boolean hasNext();
21
22    /**
23       Adds an element before the iterator position + moves the iterator past the inserted element
24       @param obj the object to add
25    */
26    void add(Object obj);
27
28    /**
29       Removes the last traversed element. This method may be called only
30       after a call to the next method
31    */
32    void remove();
33
34    /**
35       sets the last traversed element to a different value
36       @param obj the object to the set
37    */
38    void set(Object obj);
39 }
```

ADVANCED TOPIC

Link inner class has no need to access the outer class, + has no methods

```
Class LinkedList
{
   ...
   private static class Link
   {
      ...
   }
}
```

Static → indicates that the inner-class objects don't depend on the outer-class objects that generate them --- inside stuff can't access outside stuff. ⇒ efficient