### Linked Lists

Walls and Mirrors Chapter 5

```java
public class Node {
    private Object item;
    private Node next;
    public Node(int item) {...}
    public Node(int item, Node next) {...}
    // getters and setters
}
```

A linked list is a collection of Nodes:

```
<table>
<thead>
<tr>
<th>Item</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td></td>
</tr>
<tr>
<td>-3</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
</tr>
</tbody>
</table>
```

### The list interface

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get(index)</td>
<td>Returns the element at the given position</td>
</tr>
<tr>
<td>indexOf(object)</td>
<td>Returns the index of the first occurrence of the specified element</td>
</tr>
<tr>
<td>add(object)</td>
<td>Appends an element to the list</td>
</tr>
<tr>
<td>add(index, object)</td>
<td>Inserts given value at given index, shifting subsequent values right</td>
</tr>
<tr>
<td>remove(index)</td>
<td>Removes the element at the specified position (and returns it)</td>
</tr>
<tr>
<td>remove(object)</td>
<td>Removes the element that corresponds to the given object (and returns it)</td>
</tr>
<tr>
<td>size()</td>
<td>Returns the size of the list</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>Indicates if the list is empty</td>
</tr>
<tr>
<td>clear()</td>
<td>Removes all elements from the list</td>
</tr>
</tbody>
</table>

index is an int, and object is of type Object

### Implementing add

- **How do we add to a linked list?**
  - add at the end of the list
  - add at a given index

```
<table>
<thead>
<tr>
<th>Item</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>-3</td>
</tr>
<tr>
<td>-3</td>
<td>17</td>
</tr>
<tr>
<td>17</td>
<td>9</td>
</tr>
</tbody>
</table>
```

### The add method

```java
public boolean add(int index, Object item){
    if (index<0 || index>=size) {
        throw new IndexOutOfBoundsException("out of bounds");
    }
    if (index == 0) {
        head = new Node(item, head);
    } else {  // find predecessor of node to be removed
        Node curr = head;
        for (int i=0; i<index-1; i++){
            curr = curr.getNext();
        }
        curr.setNext(new Node (item, curr.getNext()));
    }
    size++;
}
```
Implementing remove

// Removes value at given index from list.
public void remove(int index) {
    //
    }  
    // How do we remove a node from a list?
    // Does it matter what the list's contents are before the remove?
    
    head = item
    size = 3

    item next
    item next
    item next

    Removing from a list

    Before removing element at index 1:
    
    After:
    
    head = item
    size = 2

    item next
    item next

    Removing from a list

    Before removing element at index 0:
    
    After:
    
    head = item
    size = 2

    item next
    item next

    List with a single element

    Before:
    
    After:
    
    head = item
    size = 0

    Data next
    element 0

    The remove method

    public void remove(int index) {
        if (index<0 || index >= size)
            throw new IndexOutOfBoundsException("List index out of bounds");
        if (index == 0) {
            // special case: removing first element
            head = head.getNext();
        } else {
            // removing from elsewhere in the list
            Node current = head;
            for (int i = 0; i < index - 1; i++) {
                current = current.getNext();
            }
            current.setNext(current.getNext().getNext());
        }
        size--;
    }

    The clear method

    How do you implement a method for removing all the elements from the list?
Linked lists recursively

- **Traversal:**
  - Write the first node of the list
  - Write the list minus its first node

- Let's code this!

Recursive linked list traversal

```java
private static void writeList(Node node) {
    //precondition: linked list is referenced by node
    //postcondition: list is displayed. list is unchanged
    if (node != null) {
        // write the first item
        System.out.println(node.getItem());
        // write the rest of the list
        writeList(node.getNext());
    }
}
```

Recursive backward traversal

- We had two ways for recursively traversing a string backwards:
  - Write the last character of the string
  - Write string minus its last character backward
  - Write string minus its first character backward
  - Write the first character of string

- Translated to our problem:
  - write the last node of the list
  - write the list minus its last node backward
  - write the list minus its first node backward
  - write the first node of the list

Which of these strategies is better for linked lists?

```java
private static void writeListBackward(Node node) {
    //precondition: linked list is referenced by node
    //postcondition: list is displayed. list is unchanged
    if (node != null) {
        // write the rest of the list
        writeListBackward(node.getNext());
        // write the first item
        System.out.println(node.getItem());
    }
}
```

Recursive linked list

- We can make the whole linked list data structure recursive!
Recursive add

```java
private Node addRecursive(Node node, Object item) {
    if (node == null) {
        node = new Node(item, node);
    } else {  // insert into the rest of the linked list
        node.setNext(addRecursive(node.getNext(), item));
    }
    return node;
}
```

```java
public void add(Object item) {
    head = addRecursive(head, item);
}
```

Recursive remove

```java
private Node removeRecursive(Node node, int index, int currentIndex) {
    if (index == currentIndex) {
        node = node.getNext();
    } else {
        node.setNext(removeRecursive(node.getNext(), index, currentIndex + 1));
    }
    return node;
}
```

```java
public void remove(int index) {
    head = removeRecursive(head, index, 0);
}
```

Linked list assignment

```
 Linked list with a dummy first node:

head = item
size = 2
item next
item next
item next
```

Variations

- Circular linked list
  ![Circular linked list](http://en.wikipedia.org/wiki/Linked_list)

- Doubly linked list
  ![Doubly linked list](http://en.wikipedia.org/wiki/Linked_list)

- What are the advantages and disadvantages of a doubly linked list?
  ![Images from](http://en.wikipedia.org/wiki/Linked_list)

Let's look at code for a doubly linked list
A digression on String vs StringBuffer

```java
public String toString() {
    StringBuffer result = new StringBuffer();
    result.append("(head - ");
    for (Node current = head; current != tail;
        current = current.getNext())
        result.append(current.getItem() + " - ");
    result.append("(tail)");
    return result.toString();
}
```

```
public String toString() {
    String result = ":[size = " + size + "]
    for (Node current = head; current != null;
        current = current.getNext())
        result += current.getItem().toString() + ";
    return result + "]";
}
```

Another digression – inner classes

- **Inner class**: defined inside another class
- If declared private it can’t be used by other classes
- The methods of the inner and outer classes have access to each other’s methods and instance variables, even if declared private.
- Makes the `DoublyLinkedList` class self-contained.