Linked Lists

Walls and Mirrors Chapter 5

Linked List Nodes

public class Node {
    private int item;
    private Node next;

    public Node(int item) { this(item, null); }

    public Node(int item, Node next) {
        setItem(item); setNext(next);
    }
}

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></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>null</td>
</tr>
</tbody>
</table>
```
## The list interface

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object get(index)</td>
<td>Returns the element at the given position</td>
</tr>
<tr>
<td>index 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>object remove(index)</td>
<td>Removes the element at the specified position (and returns it)</td>
</tr>
<tr>
<td>object remove(object)</td>
<td>Removes the element that corresponds to the given object (and returns it)</td>
</tr>
<tr>
<td>int size()</td>
<td>returns the size of the list</td>
</tr>
<tr>
<td>boolean 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

## Linked List version 1

```java
public class LinkedList {
    private Node head;
    private int size;

    public LinkedList() {
        head = null;
        size = 0;
    }

    // methods go here
}
```
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></td>
</tr>
<tr>
<td>-3</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>null</td>
</tr>
</tbody>
</table>

The add method

```java
public boolean add(int index, Object item) {
    if (index<0 || index>size) return false;
    if (index == 0) {
        head = new Node(item, head);
    } else {  // find predecessor of node
        Node curr = head;
        for (int i=0; i<index-1; i++){
            curr = curr.getNext();
        }
        curr.setNext(new Node(item, curr.getNext()));
    }
    size++;
    return true;
}
```
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?

Removing a node from a list

- Before removing element at index 1:

- After:
Removing first node from a list

- Before removing element at index 0:
  - head = -
  - size = 3

- After:
  - head = -
  - size = 2

List with a single element

- Before:
  - head = -
  - size = 1
  - data = 20
  - next = element 0

- After:
  - head = -
  - size = 0

- We must change the front field to store null instead of a node.
- Do we need a special case to handle this?
The remove method

```java
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 a linked 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 $s$
  - Write string $s$ minus its last character backward
    And
  - Write string $s$ minus its first character backward
  - Write the first character of string $s$

Translated to our problem:

- Write the last node of the list
- Write the list minus its last node backward
  And
- Write the list minus its first node backward
- Write the first node of the list

Which of these strategies is better for linked lists?
Recursive backward traversal

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());
    }
}

Variations

- Circular linked list
- Doubly linked list

What are the advantages and disadvantages of a doubly linked list?

images from: http://en.wikipedia.org/wiki/Linked_list
Doubly linked list with header node

- empty DLL:
  - header node points at itself

- non empty:
  - header node points at first and last

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.