Index of /~cs61b/lectures/09

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	

```
                     CS61b: Lecture 9
                    Monday, September 14
ANNOUNCMENTS:
-------------
Labs 11-24 will be run this week as scheduled.  Everyone not on the roster
  will be admitted on a space-available basis.

LISTNODE REVIEW
---------------
Last time we looked at the following class, which also had constructors
and a method to return the nth node in the list (not shown).

   class ListNode {
     public int item;
     public ListNode next;
   };

Let's try to write two more methods, search, which looks for an element
in the list and insertAt, which inserts an element into a given position.

PROBLEM 1: missing an invariant

      public boolean search (int element) {
        ListNode curr = this;
        while (curr != null) {
           if (curr.item = element) {  return true; }
        }
        return false;
      }

    Does this always work?  What happens in the following scenario?

       ListNode lst1 = new ListNode(1, new ListNode(2));
       lst1.next.next = lst1;
       lst1.search(4);

    Can we fix this problem?

PROBLEM 2: Modifying the front of the list.

    Now let's try to write insertAt:

       public void insertAt(int position, int element) {
         if (position == 1) { /* what to do here? */ }
         else if (position == 2) { next = new ListNode(element, next); }
         else { next.insertAt(position-1, element); }
       }
```

We can't insert (or delete) from the front of the list, because all we have is the "this" pointer to the first node.

PROBLEM 3: representing an empty list.

An empty list would be represented by null, e.g.,

```
ListNode anEmptyList = null;
```

What if we try to invoke any method on this?

```
anEmptyList.ptrTo(2);
anEmptyList.search(3);
```

Both would fail, because there is no object to receive the message.

SOLUTION: The List class

```
class List {
  private ListNode head;
  public List() { head = null; }
  public void insertFront (int obj) { head = new ListNode(obj, head); }
  // more methods go here
}
```

What does a List object look like?

```
  List lst1 = new List();            lst2.insertFront(2);
  List lst2 = new List();            lst2.insertFront(1);

       ----------         ----------   -----------   -----------
  lst1 ->| head:/ |    lst2 ->  | head: -|-->| item: 1  |   | item: 2  |
       ----------         ----------   | next: --|-->| next: /  |
                                       -----------   -----------
```

Invariant on the representation of a List: there are no cycles.
Adding to (or deleting from) the front is easy: modify the head pointer.
Empty lists are represented as null head pointers, still valid List objects.

MEMOIZATION

Note: The code for pj1 makes has an additional field in the List object

```
private int size;
```

This makes the "length()" operation an O(1) rather than O(n) operation.
An additional invariant is: size == the number of ListNodes in the list.

THE ITEM TYPE

Another difference between the code in lecture and in PJ1 was the item type.

```
class ListNode {
  Object item;   // rather than int
  ListNode next;
}
```

This also affects the insert operation and others, since they now take Objects.

What is the difference?   In Java, you can either build:

1) a list that holds a specific primitive type, e.g., a list of int, list of boolean, list of double, or
2) a list of Object, which can hold any mixture of Objects, e.g., a list

containing some Booleans, some Integers, some Accounts, and some Lists.

In lecture I used a list of int, because it makes the examples simpler.
In the pj and lab, we use a list of Object, because they are more general.

OTHER LINKED STRUCTURES

You should now be able to implement other linked structures in Java. E.g.,

A doubly-linked list:

```
class DoubleListNode {
  Object item;
  DoubleListNode previous;
  DoubleListNode next;
}
class DoublyLinked {
  DoubleListNode head;
}
```

A binary tree:

```
class TreeNode {
   Objet item;
   TreeNode left;
   TreeNode right;
}
class Tree {
   TreeNode root;
}
```