Linked Lists

How I spent my youth as a CS graduate student

Reading LDC Ch 14.10 - 14.14

Good Idea: Elements with Links

- Using only this one class, a linked structure of “nodes” is created
- One Person object contains a link to another Person object
- This second object contains a reference to a third Person, etc.
- This type of object is sometimes called self-referential
- This kind of relationship forms the basis of a linked list
- Has capacity limited only by memory in the computer
  - Is a dynamic structure because its size grows and shrinks as needed

Better Idea: Elements Without Links

- A flaw in the Person class is that the self-referential Person class must be designed so that it “knows” it may become a node in a linked list
- This violates the goal of separating the implementation details from the parts of the system that use the collection
- We better separate the details of the linked list structure from the elements that the list stores:
  - Define a separate node class LinearNode that serves to link the elements together

Arrays are limited in the sense they have a fixed size
- Resizing as needed must be done carefully and is not efficient
- A linked structure is the primary alternative to an array-based implementation of a collection

Initial implementation idea: a class Person as part of a community

```java
public class Person{
    private String name;
    private String address;
    private Person next;
    // and whatever else
}
```
Managing Linked Lists

- A node may be inserted or deleted at any location
  - at the front of the list,
  - among the interior nodes, or
  - at the end of the list

- There are a few basic techniques when managing nodes on the list, no matter what the list is used to store

- Special care must be taken when dealing with the first or last node in the list so that the reference to the entire list is maintained appropriately

Inserting a node at the front

Inserting a node in the middle

Deleting the first node in the list
Deleting an interior node

```
package javafoundations;
public class LinearNode<T> {
    private LinearNode<T> next;
    private T element;
    //Constructor: Creates an empty node
    public LinearNode() {
        next = null;
        element = null;
    }
    //Constructor: Creates node with element
    public LinearNode(T elem) {
        next = null;
        element = elem;
    }
    /**@return the node that follows this one */
    public LinearNode<T> getNext()
    { return next; }
    /** Sets the node that follows this one */
    public void setNext (LinearNode<T> node)
    { next = node; }
    /** @return element stored in this node */
    public T getElement()
    { return element; }
    /** Sets the element stored in this node */
    public void setElement (T elem)
    { element = elem; }
}
```
package javafoundations;
import javafoundations.exceptions.*;

public class LinkedStack<T> implements Stack<T> {
    private int count;
    private LinearNode<T> top;

    // Creates an empty stack.
    public LinkedStack() {
        count = 0;
        top = null;
    }

    // (more...)

    public T pop() throws EmptyCollectionException {
        if (count == 0)
            throw new EmptyCollectionException("Pop failed. Stack is empty.");
        T result = top.getElement();
        top = top.getNext();
        count--;
        return result;
    }

    // (more...)

    public String toString() {
        String result = "<top of stack>\n";
        LinearNode current = top;
        while (current != null) {
            result += current.getElement() + "\n";
            current = current.getNext();
        }
        return result + "<bottom of stack>";
    }

    // Try to implement the following.
    // public void push () { }
    // public T peek () throws EmptyCollectionException { }
    // public boolean isEmpty() { }
    // public int size() { }
}
Java’s LinkedList<E> “feels” like an array:

```java
public void add (int index, E element)
    indices change automatically!
    if index > size() throws IndexOutOfBoundsException
```
Methods for Setting and Removing Items

public E set (int index, E element)

public E remove (int index)

public E removeFirst ()

public E removeLast ()

L1.remove(1);
L1.set(2, "beer");
L1.set(1, L1.removeFirst());
L1.addFirst(L1.getLast());
L1.add(1, "butter");
System.out.println("new contents of L1: " + L1);

(new contents of L1: (what happened to the beer?)

BTW: Java’s Stack<E> implementation is using Java’s LinkedList<E>

public void push(E item)

public E pop() throws NoSuchElementException

gives StackOverflowException if stack is empty

public E element () // same as peek

List position
1    10
2    80
3    60
... ...
list.size()  5

Top of stack