CSC115 Lecture 5

LillAnne Jackson

Chapter 4: Data Abstraction: The Walls

Abstract Data Types & Java's Interface

- the IntegerList Interface
  - Implemented using an Array

Method Running Times

Introduction to the Linked List (Ch. 5)

Abstract Data Type

A high level description of

- a collection of data , and
- a set of operations on that data

Language Independent

Java Interface

A set of Java method signatures that describe

- a set of operations on some data

Example: IntegerList Interface

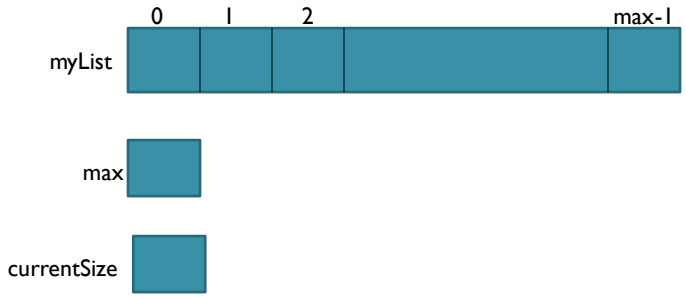
In the file : *IntegerList.java*

```
public interface IntegerList
{
  public void addFront (int x);
  public void addBack (int x);
  public int size();
  public int  get (int pos);
  public String toString();
}
```

Implementing Interfaces

- Choosing the data structure, depends on
  - Details of the Interface's operations
  - Context in which the operations will be used
- Implementation details will be hidden
  - A program (and thus, typical programmers) would only be able to access the data structure using the interface's operations

An Array-Based Implementation of the IntegerList



See Accompanying Code

Measuring Method Running Time

"Constant" Running time

- Methods like *size()*

```
public int size() {
      return currentSize;
}
```

Only a constant number of operations are done in the method

Measuring Method Running Time

"Linear" Running time

- Methods like *increaseMax()*

```
private void increaseMax (){
    //  make the array bigger(by 5)!!
    max += 5;

    // make a new array
    int[] newList = new int[max];
    // copy the previous array into the new array
    for (int i = 0; i < currentSize; i++)
        newList[i+1] = myList[i];

    // the new list becomes the list:
    myList = newList;
}
```

Goes through <u>all elements</u> in an array (complete list)
If the list is size *n*, the running time is a multiple of *n*

Test Yourself:

What is the running time for `get()`?

```
public int  get (int pos) {
      return myList[pos];
}
```

What is the running time for `addBack()`?

```
public void addBack (int x){
      if (currentSize >= max)
            increaseMax();
      myList[currentSize++] = x;
}
```

An Vector-Based Implementation of the IntegerList

From javadocs 7: http://docs.oracle.com/javase/7/docs/api/

The Vector class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

See Accompanying Code

2 implementations; 1 Interface

- Since the client code (the programs) would only use the methods of the interface, one implementation could be exchanged for another.

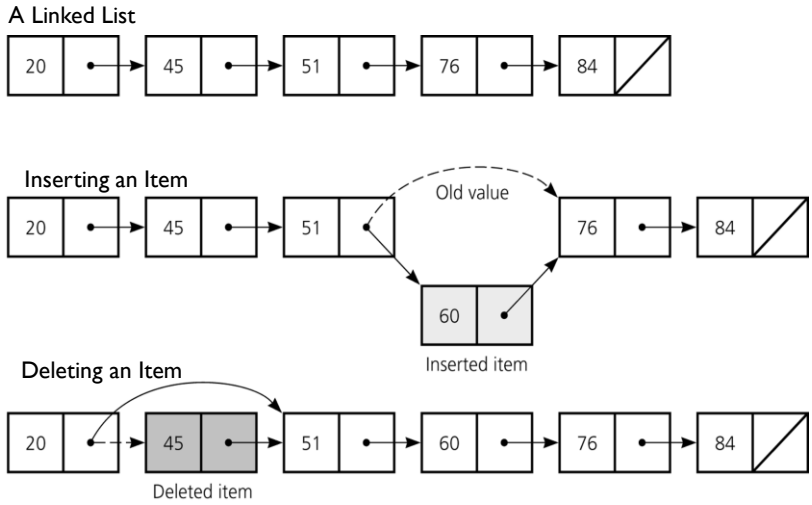
Linked List

- A reference based structure that has no pre-defined maximum or minimum size.
  - Is able to grow in size as needed
  - Does not require the shifting of items during insertions and deletions
- Can also be used to implement our IntegerList interface

Array Vs LinkedList

0	1	2	3	4
data1	data2	data3	data4	data5

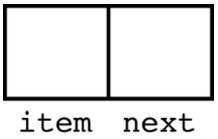
Linked List Basics



Reference-Based Linked Lists

A Linked list
- Contains *node*s that are linked to one another

A node
- Contains both data and a "link" to the next item
- Can be implemented as an object

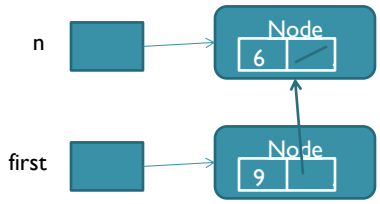



```
public class Node {
    private int item;
    private Node next;
    // constructors, accessors,
    // and mutators …
} // end class Node
```

Reference-Based Linked Lists

- Using the Node class

```
Node n = new Node (6);
Node first = new Node (9, n);
```



Reference-Based Linked Lists

- Data field `next` in the last node is set to `null`
- `head` reference variable
  - References the list's first node
  - Always exists even when the list is empty

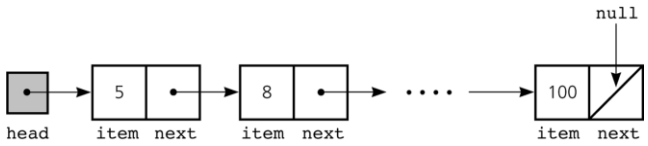


Figure 5-8

A ***head*** reference to a linked list

Reference-Based Linked Lists

- `head` reference variable can be assigned `null` without first using `new`
  - Following sequence results in a lost node

```
head = new Node(); // Don't really need to use new here
head = null; // since we lose the new Node object here
```

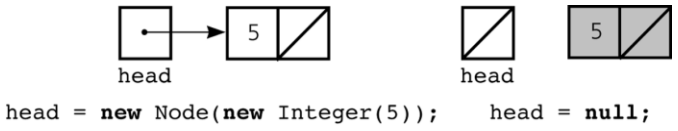


Figure 5-9

A lost node

Exercise: in preparation for next class

Re-Implement the IntegerList Interface as a Reference Based Linked List