Linked Lists

- 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
}
```

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 to accommodate the number of elements stored

Doubly Linked Lists

- A simple linked list is only one kind of linked structure
- In a doubly linked list, each node in the list stores both a reference to the next element and a reference to the previous one
- Java’s library implementation uses doubly linked lists

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
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.

```java
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;
    }
    // Returns 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;
    }
    // Returns the element stored in this node
    public T getElement() {
        return element;
    }
    // Sets the element stored in this node
    public void setElement(T elem) {
        element = elem;
    }
}
```

Forming a list with LinearNodes

// Three people we would like to link!
Person p1 = new Person("Stella", "106 Central Street");
Person p2 = new Person("Jean", "123 Sesame Street");
Person p3 = new Person("Christine", "4 Yawkey Way");

LinearNode<Person> front; // Refers to first node in list
LinearNode<Person> temp; // Refers to new node as it is being added to list

.front

Stella Jean Christine
Forming a list with LinearNodes

Counting LinearNodes in list

// Returns the number of nodes in list L.
// Recursion is elegant.
public static int getLength_rec(LinearNode<Person> L)
{
    if (L == null) return 0;
    else return 1 + getLength_rec(L.getNext());
}

Stack Implementation with Linked Lists

14.12 – javafoundations.LinkedStack

// Represents a linked implementation of a stack.
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;
    }
}
/** Removes the element at the top of this stack
 * @return a reference to it.
 * @throws an EmptyCollectionException if stack empty */
 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;
 }

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>";
 }

public void push () { }
public T peek () throws EmptyCollectionException { }
public boolean isEmpty () { }
public int size() { }

Java’s LinkedList<E> “feels” like an array:

Even though Java uses doubly linked list,
we will use the simplified picture
because it is simple and clear about accessibility

To use it you need to start your code by
import java.util.*;
To get a new list:
LinkedList<E> L = new LinkedList<E>();
LinkedList<String> L1 = new LinkedList<String>();
L1.add(0, "eggs");
L1.addFirst("milk");
L1.addLast("bread");
L1.add(2, "chicken");
System.out.println("contents of L1: " + L1);

Methods for Adding LinkedList Nodes

- public void add (int index, E element)
- indices change automatically!
- if index > size() throws IndexOutOfBoundsException

- public void addFirst (E element)
- public void addLast (E element)
- aka: public void add (E element)

- LinkedList<String> L2 = new LinkedList<String>();
for (int i = L1.size()-1; i >= 0; i--)
L2.add(L1.get(i));
System.out.println("contents of L2: " + L2);

What does L2 contain?

Methods for Getting List Items

- public E getFirst ()
- public E getLast ()
- public E get (int index)

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: Stack using Java’s LinkedList <E>

- The ADT LinkedList can be used to represent the items in a stack

- public void push(E item)
- public E pop () throws NoSuchElementException
- public E element () // same as peek

- List position
- Top of stack
- list.size()