Announcements

- Ryan Shelley
- Project 0 being graded
- Project 1 due week from Friday (strongly suggest finish by Monday)
- Programming Guidelines
- Office hours this week normal
- Office hours next week Thursday 5-7 moved to Tuesday 5-7
- Questions?
Linked Lists

LIST ADT

public interface List {
    public boolean insert(OurThing ot, int pos);
    public boolean delete(int pos);
    public int find(int key);
    public boolean isEmpty();
    public int length();
    public String toString();
}
Ordinary Linked List

- “Nodes” consisting of:
  - a data item
  - a reference (link) to the next Node
- “Head” special reference to beginning
- Classroom example
- Dynamically allocate memory
- Not necessarily contiguous in memory
- Complicated (requires thinking)
- Insert/delete more natural?
- Helps to draw pictures
public class Node {
    private OurThing thing;
    private Node next;

    public Node(OurThing ot) {
        thing = ot; next = null;
    }

    public Node() {
        thing = null; next = null;
    }

    public Node(OurThing ot, Node nd) {
        thing = ot; next = nd;
    }
}
public Node getNext() {
    return next;
}
public OurThing getOT() {
    return thing;
}

public void setNext(Node nd) {
    next = nd;
}
public void setOT(OurThing ot) {
    thing = ot;
}
public static void main(String args[]) {
    OurThing ot0, ot1, ot2, ot3;
    Node nd0, nd1, nd2, nd3;

    ot0 = new OurThing(WWW);  ot1 = new OurThing(XXX);
    ot2 = new OurThing(YYY);  ot3 = new OurThing(ZZZ);

    nd0 = new Node(ot0);      nd1 = new Node(ot1);
    nd2 = new Node(ot2);      nd3 = new Node(ot3);

    nd0.setNext(nd1);         nd1.setNext(nd2);
    nd2.setNext(nd3);

    for (Node nd = nd0; nd != null; nd = nd.getNext())
        System.out.println((nd.getOT()).toString());
}
public class LinkedList implements List {
    private Node head;
    // size?

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

    ...
Insert at beginning
public boolean delete(int ky) {
    //by key not pos
    Node prev = null;
    for (Node nd = head; nd != null; nd = nd.getNext()) {
        if (nd.getOT().getKey() == ky) {
            if (prev != null)
                prev.setNext(nd.getNext());
            else
                head = head.getNext();
            return true;
        }
        prev = nd;
    }
    return false; //size--;
public boolean delete(int pos) {return true;}

public int find(OurThing ot) {return 0;}

public boolean isEmpty() {
    if (head == null)
        return true;
    return false;
}

public int length() {return 0;}

Advantages & Disadvantages

- Relatively low overhead
- No extra “empty” space
- Insert/Delete no shifting
- More complicated to implement
- No “random access”
Variants

- Circular linked list - only one (tail?) pointer
- Doubly Linked List - Node with 3 parts
- Holding head & tail - more special cases
- Your project2 circular / doubly linked
Variants in Pictures
Most of remaining projects have something redundant
Write both versions
Select 5 (or more) different types of large data sets
Speculate which works faster/better
Time using System.currentTimeMillis()
Explain whether results were expected?
Did it actually work this way - why? why not?
Maybe also mention other factors you’d use to select