Announcements

• Exam 1 on Thursday
  – Topics: Hashing, Maps, Recursion, Linked Lists
  – Open Notes, Open book – you need to be organized

• DNA and apt-four due after exam

• Recitation on Friday
This week and next

• DNA and Exam
  – How do you study for the exam?
  – What do you study for the exam?
  – How do you work with your partner on DNA assignment?
  – When do you turn in DNA assignment?
Changing Linked Lists

```java
public class Node {
    String value;
    Node next;
    Node(String s, Node link) {
        value = s;
        next = link;
    }
}

["big","bad","cat"] : ["big","big","bad","bad","cat","cat"]
Node doubleUp(Node list){...}
```

- **Recursion to the rescue with nodes**
  - What list is it easy to `doubleUp`? Simple for other methods?
  - What about a one-node list?
  - Change one node (the one pointed to)
    - Recursion changes the rest

- **Iterative solution? Issues?**
Creating Linked List from Array

• ["one", "two", "three"] : ("one")->("two")->("three")

Node createFrom(ArrayList<String>){
    // missing code
}

• With iteration we need
  – A first node to return (pointer to it)
  – A current/last node to add on to (as we traverse array)
  – Alternative: add to front: easier, but array order ...

• With recursion we need current array element
  – Create helper method with auxiliary/extra parameter
Programming with Nodes/Linked Lists

• When adding or removing nodes
  – Be sure you alter a .next field: re-assign or call new
  – `list.next = new Node() OR tmp OR recursiveCall`

• Using iteration: keep pointer to first AND current
  – Allow iteration over list, but must keep pointer to front
  – Sometimes call new before looping to have a Node
    • e.g., invariant add to a .next field in loop
    • Return `temp.next` as needed

• Recursion is often simpler than iteration
  – Code mirrors structure of data!