Linked Data Structures

Linked lists
Recursive traversal
Reges & Stepp ch 16
References

We’ve seen references from the stack to the heap, and from the heap to the heap ...

Muppet [] ms;
ms = new Muppet[5];
ms[2] = new Muppet();
References can “link” objects

class Cell {
    String name;
    Cell next;
    public Cell( String name ) {
        this.name = name;
        this.next = null;
    }
    public void linkTo(Cell neighbor) {
        this.next = neighbor;
    }
    ...
}
We can build “linked lists” of objects
Traversing a List (with recursion)

public void printAll() {
    System.out.print(name + "->");
    if (next != null) {
        next.printAll();
    } else {
        System.out.println("[null]");
    }
}
Adding to the front ...

Cell c = new Cell("caboose");
for (int i=0; i < 3; ++i) {
    Cell head = new Cell( Integer.toString(i) );
    head.linkTo(c);
    c = head;
}

c.printAll();

What does it build? What does it print?
Cell c = new Cell("caboose");
for (int i=0; i < 3; ++i) {
    Cell head = new Cell( Integer.toString(i) );
    head.linkTo(c);
    c = head;
}
for (int i=0; i < 3; ++i) {
    Cell head = new Cell( Integer.toString(i) );
    head.linkTo(c);
    c = head;
}
c.printAll();
public void printAll() {
    System.out.print( name + "->");
    if (next != null) {
        next.printAll();
    } else {
        System.out.println("[null]");
    }
}

this

```
Cell

2

next

Cell

1

next

Cell

0

next

caboose

next
```
public void printAll() {
    System.out.print( name + "->");
    if (next != null) {
        next.printAll();
    } else {
        System.out.println("[null]");
    }
}
public void printAll() {
    System.out.print( name + "->");
    if (next != null) {
        next.printAll();
    } else {
        System.out.println("[null]");
    }
}
public void printAll() {
    System.out.println(name + "->");
    if (next != null) {
        next.printAll();
    } else {
        System.out.println("[null]");
    }
}
Linked Lists
versus Arrays and ArrayLists:
What are lists good for?
What are they bad at?

Consider an array or ArrayList of Muppet objects ...
Suppose I want to ...

Add a Muppet at the beginning of the sequence

Which is faster, array or list?

Add a Muppet at the end of the sequence

Which is faster, array or list?

Add a Muppet in alphabetical order by name

Which is faster, array or list?

Remove a Muppet from the middle of the sequence
Insertion at front of list

Faster or slower than array? Why?
First we have to *find* the end ... typically search the whole list. We say it is a linear time operation (proportional to list length), or $O(n)$ where $n$ is length of list. ( "big-oh of n" )
Delete from list

Cheap (constant time) once we find the element to delete ... but finding it still takes linear time.

Are arrays better or worse?  (*semi-trick question*)