Linked Structures

• Self-referential classes can be used to create linked data structures:

```java
class Node {
    private int data;
    private Node next;
    public Node(int d, Node n) {
        data = d;
        next = n;
    }
}
```

• `next` holds a reference to a `Node` object
• through the next reference, can link `Node` objects together:

![Linked Node Diagram]
Linked List

- Ordered Collection of data
- Need a single variable which is pointer to 1\textsuperscript{st} node on list
- Nodes are linked together in-order by following \texttt{next} references

```java
Node head;
head = new Node(25, null);
head.setNext(new Node(99, null));
```
Operations on a List

• All start at Node referred to by head reference, and traverse next references to access other nodes in the list

• Accessing the $i$th node is $O(n)$:
  – first access head Node, follow its next reference to access the $2^{nd}$ Node, follow its next reference to access the $3^{rd}$ Node, and so on
Insert at Head of List

```java
head = null;
for(i=0; i < 10; i++) {
    int val = reader.nextInt();
    tmp = new Node(val, null);
    tmp.setNext(head);
    head = tmp;
}
```

---

**i == 0:**

```
head
```

**i == 1:**

```
head
```
Resulting List of 10 nodes:

- head
  - data: 23
  - next
- data: 44
  - next
- data: 35
  - next
- data: 77
  - next
- data: 88
  - next
- data: 683
  - next
- data: 21
  - next
- data: 55
  - next
- data: 63
  - next
- data: 25
  - next
Traverse the List

tmp = head;       // start at the 1st node
while(tmp != null) {
    System.out.print(tmp.getData() + " ");
    tmp = tmp.getNext();  // make tmp ref to next node
}
// output:  23 44 35 77 88 683 21 55 63 25
Find Element In List

- Start at head Node, compare search value to data field
- traverse next refs until matching data field is found, or until no more list

```java
public Node find(Node head, int val) {
    Node tmp = head;
    while(tmp != null) {
        if(tmp.getData() == val)
            return tmp;
        tmp = tmp.getNext();
    }
    return null; // val not in list
}
```
Insert in the middle

Node new_node, tmp;

new_node = new Node(20, null);
tmp = head.getNext();  // lets just make tmp point  
// to some Node after head

// insert new_node after tmp
new_node.setNext(tmp.getNext());
tmp.setNext(new_node);