Java 2 Linked Structures

- Looking at a different data structure besides arrays

```
public class LinkedBag<T> implements Bag<T> {
        private ???
}
```

- Advantages of Array
    - Fast random access
    - efficient use of memory
    - Built into the language; a “common currency” for any data storage scheme

- Disadvantages of Array
    - Inefficient to insert or delete anywhere but the end; must shift left/right
    - Need to “resize” when full/sparse

```
public class LinkedBag<T> implements Bag<T> {
        private Node front;
        private int size;

        public LinkedBag() {
                front = null;
                size = 0;
        }

        public int size() {
                return size;
        }

        public boolean isEmpty() {
                return size == 0;
        }

        public boolean add(T element) {
                Node n = new Node(element);
                n.next = front;
                front = n;
                size++;
                return true;
        }

        public boolean contains(T element) {
                return locate(element);
        }

        public boolean remove(T element) {
                Node n = locate(element);

                if (n == null)      return false;

                size—;
                return true;
        }
```

```
    private class Node {
        private T element;
        private Node next;
        private Node prev;

        private Node(Object e) {
            element = e;
        }

        private Node(Object e, Node n) {
            element = e;
            next = n;
        }

        public int length(Node n) {
            Node p = n;
            int len = 0;
            while (p != null) { //A common traversal strategy to traverse the the chain of nodes//
            //We’re assuming the node chain is terminated by null//
                len++;
                p = p.next;
            }

            return len;
        }

        public boolean contains(Node n, Object target) {
            Node p = n;

            while (p != null) {
                if (p.element.equals(target)) {
                    return true;
                }
                p = p.next
            }
            return false;
        }
    }

    private class LinkedIterator implements Iterator<T> {
        private Node current = front;

        public boolean hasNext() {
            return current != null;
        }
    }
}
//
```

- Inserting

```
Node n = new Node(‘X’);

if (inserting a new first node) {
    n.next = front;
    front = n;
}
```

```
else {
        Node prev;
        //Find the right spot which is node right before
        n.next = prev.next;
        prev.next = n;
}
```

- Delete

```
if (deleting first node) {
        front = front.next;
}

else {
        Node prev;
        //find the right spot
        prev.next = prev.next.next;
}
//
                //First example of a recursive structure
        }
}
```

- Individual containers are explicitly linked together.
- Container must have reference to the element the node stores.
- Container must have a reference to the next node in the chain.

Memory

- Book b = new Book();
- int[] a = {2, 4, 6, 8, 10};
    - Two regions of memory involved:
        - Stack Memory
            - Stack memory gets consumed top to bottom
            - Used for things with names (methods, etc.)
            - b will be associated to the stack
        - Heap Memory
            - Heap memory gets consumed bottom to top
            - What b actually is can be found in the heap.
            - Every time something is instantiated it is allocated in heap memory
            - All allocation happen on the heap.
            - Garbage is memory that has been allocate on the heap, but cannot be accessed on the stack.

WHITEY BOARDY THINGY!!! Pretty Fly For a White Board

```
n = new Node(1, new Node(2));
n.next.next = new Node(3, null);
n = new Node(4, n.next);
n.next.next.next = n;
```

After the first two lines of code:

n > 1 | 2 > 2 | 3 > 3 | °

After the third line of code:

n > 4 | 2 > 2 | 3 > 3 | °

After the final line of code:

n > 4 | 2 > 2 | 3 > 3 | 4

- Advantage of nodes:

  - Given a reference to a node, efficient to insert or add before or after that node; no shifting required.
- Disadvantages
  - no random access
  - less efficient use of memory
  - not built in; nodes are user-created

PERFORMANCE

- add(T element)
  - both O(1) except ArrayBag is amortized cost.
- remove()
  - both O(N)
- contains()
  - both O(N)
- size()
  - both O(1)
- isEmpty()
  - both O(1)
- iterator
  - both O(1)

- **SET**
- add(T element)
  - both O(N)
- remove(T element)
  - both O(N)
- contains(T element)
  - both O(N)
- size()
  - both O(1)
- isEmpty()
  - both O(1)
- iterator()
  - both O(1)

- **SET Order Array**
- add(T element)
  - both O(N)
- remove(T element)
  - both O(N)
- contains(T element)
  - Array O(log N) Node O(N)
- size()
  - both O(1)
- isEmpty()
  - both O(1)
- iterator()
  - both O(1)