How to insert into a linked list

```
public class LinkedList
        extends AbstractCollection
        implements List
{
     private static class Node
     {
         // some constructors
         public Object   element;
         public Node next;
     }

    private int theSize;
    private Node beginMarker;
    private Node endMarker;

    //  • • •  Other stuff here
}
```

```
// Insert newNode after q
newNode.next = q.next;
q.next = newNode;

newNode.prev = q;
newNode.next.prev = newNode;
theSize++;
```

```
public void add( int idx, Object x ) {
        Node p = getNode( idx );
        Node newNode = new Node( x, p.prev, p );
        newNode.prev.next = newNode;
        p.prev = newNode;
        theSize++;
        modCount++;
}
```

How to delete & get from a linked list

```
// Delete node after q
q.next = q.next.next;

q.next.prev = q;
theSize-- ;
return q;
```

```
private Object remove( Node p )
{
        p.next.prev = p.prev;
        p.prev.next = p.next;
        theSize--;
        modCount++;
        return p.data;
}
```

```
p = beginMarker.next;
for( int i = 0; i < idx; i++ )
            p = p.next;
return p;
```

```
private Node getNode( int idx ) {
        Node p;
        if( idx < 0 || idx > size( ) )
            throw new IndexOutOfBoundsException( );
        if( idx < size( ) / 2 ) {
            p = beginMarker.next;
            for( int i = 0; i < idx; i++ )    p = p.next;
        } else {
            p = endMarker;
            for( int i = size( ); i > idx; i-- )  p = p.prev;
        }
        return p;
}
```

Stacks and Queues

```
public interface Stack
{
    public Object push( Object x );
    public Object pop( );
    public boolean isEmpty( );
}

public interface Queue
{
    public boolean isEmpty( );
    public void enqueue( Object x );
    public Object dequeue( );
}
```

How to search in a sorted list

```
public class BinarySearch // Fig 5.11, pg168
{
    public static final int NOT_FOUND = -1;
    public static int binarySearch
                ( Comparable [ ] a, Comparable x )
    {
        int low = 0;
        int high = a.length - 1;
        int mid;
        while( low <= high )
        {
            mid = ( low + high ) / 2;
            if( a[ mid ].compareTo( x ) < 0 )
                low = mid + 1;
            else if( a[ mid ].compareTo( x ) > 0 )
                high = mid - 1;
            else
                return mid;
        }
        return NOT_FOUND;     // NOT_FOUND = -1
    }
```

```
    // Test program
    public static void main( String [ ] args )
    {
        int SIZE = 8;
        Comparable [ ] a = new Integer [ SIZE ];
        for( int i = 0; i < SIZE; i++ )
            a[ i ] = new Integer( i * 2 );

        for( int i = 0; i < SIZE * 2; i++ )
            System.out.println( "Found " + i + " at " +
                  binarySearch( a, new Integer( i ) ) );
    }
}
```