How to insert into a linked list

```
public class LinkedList
       extends AbstractCollection
       implements List
{
    private static class Node
    { // p538
       // 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;
```

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

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

```
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;
}
```

```
package weiss.util;

public class LinkedList extends AbstractCollection
                 implements List // Fig 17.20-30, p554
{
   public LinkedList( ) { clear( ); }
   public LinkedList( Collection other ) {
      clear( );
      Iterator itr = other.iterator( );
      while( itr.hasNext( ) )
         add( itr.next( ) );
   }
   public int size( ) { return theSize; }
   public boolean contains( Object x ) {
      return findPos( x ) != NOT_FOUND;
   }
   private Node findPos( Object x ) {
      for( Node p = beginMarker.next;
                  p != endMarker; p = p.next )
         if( x == null ) {
            if( p.data == null ) return p;
         }
         else if( x.equals( p.data ) ) return p;
      return NOT_FOUND;
   }
   public boolean add( Object x ) {
      addLast( x );
      return true;
   }
   public void addFirst( Object x ) { add( 0, x ); }
   public void addLast( Object x ) { add( size( ), x ); }
```

```
   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++;
   }
   public Object getFirst( ) {
      if( isEmpty( ) )
         throw new NoSuchElementException( );
      return getNode( 0 ).data;
   }
   public Object getLast( ) {
      if( isEmpty( ) )
         throw new NoSuchElementException( );
      return getNode( size( ) - 1 ).data;
   }
   public Object get( int idx ) {return getNode( idx ).data;}
   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;
   }
```

```
public Object removeFirst( ) {
    if( isEmpty( ) ) throw new NoSuchElementException( );
    return remove( getNode( 0 ) );
}
public Object removeLast( ) {
    if( isEmpty( ) ) throw new NoSuchElementException( );
    return remove( getNode( size( ) - 1 ) );
}
public boolean remove( Object x ) {
    Node pos = findPos( x );
    if( pos == NOT_FOUND )   return false;
    else {
        remove( pos );
        return true;
    }
}
public Object remove( int idx ) { return remove( getNode( idx ) );}
private Object remove( Node p ) {
    p.next.prev = p.prev;
    p.prev.next = p.next;
    theSize--;
    modCount++;
    return p.data;
}
public void clear( ) {
    beginMarker = new Node( "BEGINMARKER", null, null );
    endMarker = new Node( "ENDMARKER", beginMarker, null );
    beginMarker.next = endMarker;
    theSize = 0;
    modCount++;
}
```

```
private class LinkedListIterator implements ListIterator
  {
    private Node current;
    private Node lastVisited = null;
    private boolean lastMoveWasPrev = false;
    private int expectedModCount = modCount;

    public LinkedListIterator( int idx ){current = getNode( idx );
    public boolean hasNext( ) {
      if( expectedModCount != modCount )
        throw new ConcurrentModificationException( );
      return current != endMarker;
    }
    public Object next( ) {
      if( !hasNext( ) ) throw new NoSuchElementException( );
      Object nextItem = current.data;
      lastVisited = current;
      current = current.next;
      lastMoveWasPrev = false;
      return nextItem;
    }
    public void remove( ){
      if( expectedModCount != modCount )
        throw new ConcurrentModificationException( );
      if( lastVisited == null ) throw new IllegalStateException( );
      LinkedList.this.remove( lastVisited );
      lastVisited = null;
      if( lastMoveWasPrev )
        current = current.next;
      expectedModCount++;
    }
```

```
    public boolean hasPrevious( )
    {
      if( expectedModCount != modCount )
        throw new ConcurrentModificationException( )
      return current != beginMarker.next;
    }

    public Object previous( )
    {
      if( expectedModCount != modCount )
        throw new ConcurrentModificationException( )
      if( !hasPrevious( ) )
        throw new NoSuchElementException( );

      current = current.prev;
      lastVisited = current;
      lastMoveWasPrev = true;
      return current.data;
    }
  }
```

Fig 17.30, page 562

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 ) ) );
   }
}
```

Stacks and Queues

```
public interface Stack
{ // Fig 6.21, p206
    public Object push( Object x );
    public Object pop( );
    public boolean isEmpty( );
}

public interface Queue
{ // Fig 6.23, p209
    public boolean isEmpty( );
    public void enqueue( Object x );
    public Object dequeue( );
}
```

Stacks & Queues – Implementations

```
public class Stack implements Serializable
{  // Fig 16.28, p532
   public Object push( Object x )
   {
      items.add( x );
      return x;
   }
   public Object pop( )
   {
      if( isEmpty( ) )
         throw new EmptyStackException( );
      return items.remove( items.size( ) - 1 );
   }
   public boolean isEmpty( )
      {  return size( ) == 0;  }

   private ArrayList items;
   // LinkedList????
}
```

```
public class ListQueue implements Queue
{  // Fig 16.25, p529
   public boolean isEmpty( )
   {  return front == null; }
   public void enqueue( Object x )
   {  if( isEmpty( ) )
         back = front = new ListNode( x );
      else              // Regular case
         back = back.next = new ListNode( x );
   }
   public Object dequeue( )
   {  if( isEmpty( ) )
         throw new UnderflowException( "" );
      Object returnValue = front.element;
      front = front.next;
      return returnValue;
   }

   private ListNode front;
   private ListNode back;
}
```

Stacks: Application 1

- Check balanced parentheses
  - (())()(()(()))
  - ((())()))()(()

```
While (expr.nextToken())
{
   if next token is “(“
            push “(“ on stack;
   else
            if stack is not empty
                     pop “(“ from stack;
            else report error;
}
If stack is not empty
            report error;
```

Stacks: Application 2

Evaluate Postfix Expressions

1 2 3 + *

= (1* (2 + 3))

4 1 2 2 3 * ^ + -1 * +

= ?

```
While (expr.nextToken())
{
   if next token is an operand
      push operand on stack;
   else if next token is an operator Op
      {
         pop Val1 from stack;
         pop Val2 from stack;
         compute Val1 Op Val2;
         push result on stack;
      }
   if stack has only one item
      pop value and return as Value of expr;
   else report error;
}
```

Stacks – Applications 3

- Convert Infix Expressions to Postfix

Recursion

- **Example 1:** Fibonacci Numbers
  1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

```
public static long fib(int n)
{
        if (n <= 1)
                return n;
        else
                return fib(n-1) + fib(n-2);
}
```

- **Example 2:** Towers of Hanoi