1 Boxes and Pointers

Draw a box and pointer diagram to represent the IntLists after each statement.

```java
IntList L = IntList.list(1, 2, 3, 4);
IntList M = L.tail.tail;
N = IntList.list(5, 6, 7);
N.tail.tail.tail = N;
L.tail.tail = N.tail.tail.tail.tail;
M.tail.tail = L;
```

See last page for solution

2 Shifting a Linked List

Implement the following methods to circularly shift an IntList to the left destructively and non-destructively.

```java
/** Destructively shifts the elements of the given IntList L to
 * the left by one position (e.g. if the original list is
 * (5, 4, 9, 1, 2, 3) then this method should return the list
 * (4, 9, 1, 2, 3, 5)). Returns the first node in the shifted list.
 * Don't use 'new'; modify the original IntList. */
public static IntList shiftListDestructive(IntList L) {
    if (L == null) {
        return null;
    }
    IntList cur = L;
    while (cur.tail != null) {
        cur = cur.tail;
    }
    cur.tail = L;
    IntList ret = L.tail;
    L.tail = null;
    return ret;
}

/** Non-destructively shifts the elements of the given IntList L
 * to the left by one position. Returns the first node in the shifted list.
 * Don't modify the original IntList. */
public static IntList shiftListNondestructive(IntList L) {
    if (L == null) {
        return null;
    }
    if (L.tail == null) {
        return new IntList(L.head, null);
    }
    IntList cur = L.tail;
    IntList ret = new IntList(cur.head, null);
    IntList L2 = ret;
    while (cur.tail != null) {
        cur = cur.tail;
    }
    return ret;
}
```
L2.tail = new IntList(cur.head, null);
L2 = L2.tail;
}
L2.tail = new IntList(L.head, null);
return ret;

3 Palindrome

Implement the following two methods which determine whether an IntList is a palindrome.

/** Non-destructively reverses an IntList L. 
* Do not modify the original IntList. */
public static IntList reverseNondestructive(IntList L) {
    IntList L2 = null;
    while (L != null) {
        L2 = new IntList(L.head, L2);
        L = L.tail;
    }
    return L2;
}

/** Returns whether the IntList L is a palindrome or not, 
* or if it reads the same backwards as forwards. Hint: you may 
* want to use reverseNondestructive. */
public static boolean isPalindrome(IntList L) {
    if (L == null) {
        return false;
    }
    IntList reversed = reverseNondestructive(L);
    while (L != null) {
        if (L.head != reversed.head) {
            return false;
        }
        L = L.tail;
        reversed = reversed.tail;
    }
    return true;
}
IntList \texttt{L} = \texttt{IntList.list (1, 2, 3, 4)};

\texttt{M = L.tail.tail};

\texttt{N = IntList.list (5, 6, 7)};

\texttt{N.tail.tail.tail = N};

\texttt{L.tail.tail = N.tail.tail.tail.tail};

\texttt{M.tail.tail = L};