Solution 3

Problem 1

// Recursive, Two-parameter solution
public MyList1<T> append(MyList1<T> L ,MyList1<T> M){
    MyList1<T> W= new MyList1<T>();
    W.value=L.value;
    if (L.next==null)
        W.next=M.next;
    else
        W.next=append(L.next,M);
    return W;
}

// Iterative, Two-parameter solution
public MyList1<T> append(MyList1<T> L ,MyList1<T> M){
    MyList1<T> W= new MyList1<T>();
    MyList1<T> PL = L.next;
    MyList1<T> PW = W;
    while (PL != null){
        PW.next = new MyList1<T>();
        PW.next.value = PL.value;
        PL = PL.next;
        PW = PW.next;
    }
    PW.next = M.next;
    return W;
}

// Recursive, One-parameter solution
public MyList1<T> append(MyList1<T> M){
    MyList1<T> W= new MyList1<T>();
    W.value=value;
    if (next==null)
        W.next=M.next;
    else
        W.next=next.append(M);
    return W;
}

// Iterative, One-parameter solution
public MyList1<T> append(MyList1<T> M){
    MyList1<T> W= new MyList1<T>();

MyList1<T> PL = next;
MyList1<T> PW = W;
while(PL != null){
    PW.next = new MyList1<T>();
    PW.next.value = PL.value;
    PL = PL.next;
    PW = PW.next;
}
PW.next = M.next;
return W;

B. To append M to the end of L and not having L altered, we need to make a copy from every
node in L. This depends on the length of L. When we have a copy of L we can easily make it to
point to the first of M, which means we don’t need to worry about the length of M.

Problem 2.

A. Within class MyList1<T>:

    public void SwapN(int N){
        MyList1<T> beforeNth = next;
        for(int i=0; i<N-1; i++){
            if(!(beforeNth.next!=null & & beforeNth.next.next!=null)){
                System.out.println("No way to swap Nth and the next");
                return;
            }
            beforeNth = beforeNth.next;
        }
        MyList1<T> Nth = beforeNth.next;
        beforeNth.next = Nth.next;
        Nth.next = Nth.next.next;
        beforeNth.next.next = Nth;
    }

B. Within class MyList2<T>:

    public void SwapN(int N){
        MyNode2<T> beforeNth = Nth(N-1);
        MyNode2<T> Nth = beforeNth.getNext();
        beforeNth.setNext(Nth.getNext());
        Nth.setNext(beforeNth.getNext().getNext());
beforeNth.getNext().setNext(Nth);
Nth.getNext().setPrev(Nth);
Nth.setPrev(beforeNth.getNext());
Nth.getPrev().setPrev(beforeNth);
}

Problem 3

A. L = [The quality mercy], M = [errand of mercy]
B. L = [The quality of], M = [errand of]
C. In MyList2 we are just keeping a reference to the first and the end of the list, while the other list related information (here next and prev in each node) is kept with the node itself. Therefore any alteration of node object is potential of making some other list to be changed unintentionally.

Honors

A. Within MyList2<T>:

    public void reverse(){
        backwards = true;
    }

B. For those type of methods within MyList2 that are present with their complement like (addFirst(T x) and addLast(T x), we can change them in this way for example:

    public void addFirst(T X) {
        if(backwards) addFirst(X, true);
        else addFirst(X, false);}

    public void addFirst(T X, boolean workBackward){
        if(workBackward) addLast(X, false);
        else /*!Old implementation of addFirst(T X)*/}

    public void addLast(T X) {
        if(backwards) addLast(X, true);
        else addLast(X, false);
    }

    public void addLast(T X, boolean workBackward ) {
        if(workBackward) addFirst(X, false);
        else /*!Old implementation of addLast(T X)*/
For the other type of methods like Nth, it would branch off to implement both case of regular and reversing list by replacing first with last and getNext() with getPrev() for the reverse case:

```java
public MyNode2<T> Nth(int N) {
    if (backwards) {
        MyNode2<T> W = last;
        int I = 0;
        while (I < N) {
            W = W.getPrev();
            I++;
        }
        return W;
    } else {
        MyNode2<T> W = first;
        int I = 0;
        while (I < N) {
            W = W.getNext();
            I++;
        }
        return W;
    }
}
```