1. public Object removeFirst() throws Exception

Step 1: Create a ListNode type object named cur, and assign head.next to cur. Cur now refers to head.next.

Step 2: Create a for loop that depends on this.size NOT being 0, because if the list size is 0, we can’t do anything. Minimum is 1, and we can assume no list with negative size will be passed in.

Step 3: Assign cur.next to head.next. This essentially bypasses the first node, as expected.

Step 4: If the array is empty, an exception is thrown.

null data1 data2 null

Dummy head

cur

Figure Non-empty singly linked list with dummy head implemented. Variable cur of type ListNode gets head.next.

null data1 data2 null

Dummy head

Cur cur.next

Figure Variable cur "moves up" in the linked list by the loop.

null data1 data2 null

Dummy head

Cur cur.next

Figure cur.next is then assigned to head.next, making cur's node (second node) the first node

1. public boolean contains(Object o)

Step 1: Create ListNode named tmp, with object o as the parameters.

Step 2: Create ListNode cur with head.next and prev with null.

Step 3: Create for loop that ends at (this.size-1).

Step 4: If tmp.data is equals to cur.data, we return true. Otherwise, return false.

1. public boolean remove(Object o)

Step1: Create nodes prev with this.head, cur with this.head.next.

Step2: when cur is NOT null (not at a non-node object), move down the nodes with prev=cur and cur=cur.next.

Step3: If cur.data matches object o from the parameters, change prev.next into cur.next, basically bypassing the current node cur is in, hence deleting it from the linked list. Return true.

Step 4: if cur.data and object o is null, return false.

1. public boolean removeAllCopies( Object o )

Step1: Create nodes tmp as listNode with parameters o, cur with head.next, and prev with this.head

Step2: Check WHILE this.head.next AND cur is NOT null to prevent null pointers

Step3: Check IF cur is NOT null in order to know when you’ve reached the end of the nodes

Step4: if cur.data == tmp.data, modify prev.next (essentially this.head.next) into cur.next, essentially bypassing the node that cur is currently pointing to.

Step5: Reduce the size

Step6: Move up the node with cur=cur.next .

Step7: return false otherwise.

1. public static MyLinkedList interleave(MyLinkedList A, MyLinkedList B)

Step1: Create LinkedList C

Step2: Create nodes nodeA for A.head, nodeB for B.head, curA for nodeA.next, curb for nodeB.next, prevC for C.head, and curC for C.head.next

Step3: WHILE curA.next OR curB.next is NOT null, in order to check for end of Nodes

Step4: curC gets curA, copying first node of linkedlist A to linkedlistC

Step5: curC.next gets curB, copying first node of linkedlist B to linkedlist C

Step6: curA gets curA.next and curB gets curB.next, in order to move them up their respective nodes

Step7: curC gets curA again, copying the second node of linkedlist A to linkedlistC

Step8: curC gets curB again, copying the second node of linked list B to linked list C.

1. public void add(int index, Object o)

Step1: Create prev and cur nodes with this.head and this.head.next respectively.

Step2: Check index number.

Step3: For loop step up the nodes until index number is achieved, using prev = cur then cur=cur.next

Step4: set prev.next as a new node with parameters o, and prev.next.next as cur when the index is inbetween two already used index, or null when index given is larger than linkedlist node size.

1. public Object get(int index)

Step 1: Check if index is < 0 or > this.size. If it is, throw IndexOutOfBoundsException.

Step2: For loop to step up the node with pre=cur and cur=cur.next, until given index is reached

Step3: return object cur

1. public Object remove(int index) throws IndexOutOfBoundsException

Step1: Check if index is < 0 or > this.size. If it is, throw IndexOutOfBoundsException.

Step 2: For loop to step up to the node with pre=cur and cur=cur.next, until given index is reached.

Step 3: Set pre.next as cur.next. This essentially bypasses the link between the previous node of the given index, and the node of the given index. The link essentially moves from cur (the current index) to cur.next (the next index). Cur will be lost forever (current index).

Step 4: Return cur

9. public boolean add(Object e)

Step 1: Create listNode cur with head.next and prev as null.

Step 2: if cur is null (meaning the only node is the dummy node), head.next gets a new ListNode as e as the parameters. Then increase this.size.

Step 3: if cur is NOT null (meaning it is a non-empty list), move upwards with prev=cur and cur=cur.next.

Step 4: if cur is now NULL, we know we have reached the end of the node. As sooner or later, the node that cur.next takes will point to NULL, as there are no nodes after it.

Step 5: Now we set prev.next as a new ListNode with parameter e.