LinkedList Add/Remove

```
public class LinkedListAddRemoveLastDemo {


   // 1. The LinkedList class
   static class LinkedList<E> {
      // Points to the first node in the list
      private Node<E> head;
      // Tracks the current number of elements
      private int currentSize;


      // 2. The Node<E> inner class
      private static class Node<E> {
         E data;      // the data stored in this node
         Node<E> next; // pointer to the next node


         public Node(E obj) {
            data = obj;
            next = null;
         }
      }


      // Constructor: empty list at start
      public LinkedList() {
         head = null;
         currentSize = 0;
      }


      // Check if the list is empty
```

```
public boolean isEmpty() {
   return (head == null);
}

// Return the current number of elements (O(1))
public int size() {
   return currentSize;
}

// ---------------------------------------------------------------
// Add a new element at the FRONT of the list
// ---------------------------------------------------------------
public void addFirst(E obj) {
   Node<E> newNode = new Node<>(obj);
   newNode.next = head; // new node points to the old head
   head = newNode;     // head is now the new node
   currentSize++;      // increment the count
}

// ---------------------------------------------------------------
// Remove the first element and return its data
// ---------------------------------------------------------------
public E removeFirst() {
   if (isEmpty()) {
      return null; // or throw an exception if preferred
   }
   E removedData = head.data;
   head = head.next; // move head to the next node
   currentSize--;
```

```
    return removedData;
}

// ---------------------------------------------------------------
// Add a new element at the END of the list
// ---------------------------------------------------------------
public void addLast(E obj) {
    Node<E> newNode = new Node<>(obj);

    // If list is empty, new node becomes the head
    if (isEmpty()) {
        head = newNode;
    } else {
        // Otherwise, find the last node
        Node<E> current = head;
        while (current.next != null) {
            current = current.next;
        }
        // Link the last node's 'next' to the new node
        current.next = newNode;
    }
    currentSize++;
}

// ---------------------------------------------------------------
// Remove the last element and return its data
// ---------------------------------------------------------------
public E removeLast() {
    if (isEmpty()) {
```

```
        return null; // or throw an exception if preferred
    }
    // If there's only one node, handle that separately
    if (head.next == null) {
        E data = head.data;
        head = null;
        currentSize--;
        return data;
    }

    // Otherwise, traverse to find the second-to-last node
    Node<E> current = head;
    Node<E> prev = null;
    while (current.next != null) {
        prev = current;
        current = current.next;
    }
    // current is now the LAST node, prev is the node before it
    E data = current.data;
    prev.next = null; // remove the last node
    currentSize--;
    return data;
}

// ------------------------------------------------------------------
// Print the elements in the list
// ------------------------------------------------------------------
public void printList() {
    Node<E> current = head;
```

```
        while (current != null) {
            System.out.print(current.data + " -> ");
            current = current.next;
        }
        System.out.println("null");
    }
}

// 3. A main method to test addLast/removeLast
public static void main(String[] args) {
    LinkedList<String> myList = new LinkedList<>();

    System.out.println("Is the list empty? " + myList.isEmpty());
    System.out.println("List size: " + myList.size());
    myList.printList();  // Should show "null"

    // Add elements at the end
    myList.addLast("A");
    myList.addLast("B");
    myList.addLast("C");
    System.out.println("\nAfter adding A, B, C at the END:");
    myList.printList();
    System.out.println("List size: " + myList.size());

    // Remove the last element
    String removedLast = myList.removeLast();
    System.out.println("\nRemoved last element: " + removedLast);
    myList.printList();
    System.out.println("List size: " + myList.size());
```

```
        // Add an element at the front
        myList.addFirst("X");
        System.out.println("\nAfter adding X at the FRONT:");
        myList.printList();
        System.out.println("List size: " + myList.size());

        // Remove the first element
        String removedFirst = myList.removeFirst();
        System.out.println("\nRemoved first element: " + removedFirst);
        myList.printList();
        System.out.println("List size: " + myList.size());

        // Remove the last element again
        removedLast = myList.removeLast();
        System.out.println("\nRemoved last element again: " + removedLast);
        myList.printList();
        System.out.println("List size: " + myList.size());

        System.out.println("\nIs the list empty? " + myList.isEmpty());
    }
}
```

Output

```
Is the list empty? true
List size: 0
null

After adding A, B, C at the END:
```

A -> B -> C -> null

List size: 3

Removed last element: C

A -> B -> null

List size: 2

After adding X at the FRONT:

X -> A -> B -> null

List size: 3

Removed first element: X

A -> B -> null

List size: 2

Removed last element again: B

A -> null

List size: 1

Is the list empty? false