1. develop an implementation of the equals method in the context of the SinglyLinkedList class.

public class ListNode {  
  
 int val;  
 ListNode next;  
  
 ListNode(int val) {  
 this.val = val;  
 }  
  
  
 public ListNode findSecondToLastNode(ListNode head) {  
 if (head == null || head.next == null) {  
 return null; // Empty list or list with only one node  
 }  
  
 ListNode current = head;  
 ListNode previous = head;  
  
 while (current.next != null) {  
 previous = current;  
 current = current.next;  
 }  
  
 return previous;  
 }  
}

1. Give an implementation of the size( ) method for the SingularlyLinkedList class, assuming that we did not maintain size as an instance variable.
2. public class SinglyLinkedListE {  
    private Node<E> head;  
    public int size() {  
    int count = 0;  
    Node<E> current = head;  
     
    while (current != null) {  
    count++;  
    current = current.next;  
    }  
     
    return count;  
    }  
     
    private static class Node<T> {  
    T data;  
    Node<T> next;  
     
    Node(T data) {  
    this.data = data;  
    this.next = null;  
    }  
    }  
   }

Implement a rotate( ) method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst( )), yet without creating any new node

public class SinglyLinkedList<T> {  
 private Node<T> head;  
 private Node<T> tail;  
 private int size;  
  
 // ... other methods and constructors ...  
  
 public void rotate() {  
 if (size <= 1) {  
 return; // No rotation needed  
 }  
  
 Node<T> newTail = head; // New tail node  
 head = head.next; // Move head to the next node  
 newTail.next = null; // Set the next of new tail to null  
  
 tail.next = newTail; // Make the new tail the next of the original tail  
 tail = newTail; // Update the tail reference  
  
 // The list has been rotated, and no new node has been created  
 }  
  
 // ... other methods ...  
  
 private static class Node<T> {  
 private T data;  
 private Node<T> next;  
  
 public Node(T data) {  
 this.data = data;  
 }  
 }  
}

1. Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.

public class SinglyLinkedList6 {  
 private Node<E> head;  
 private int size;  
  
 // ... other methods and constructors ...  
  
 public void reverse() {  
 Node<E> previous = null;  
 Node<E> current = head;  
 Node<E> next = null;  
  
 while (current != null) {  
 next = current.next;  
 current.next = previous;  
 previous = current;  
 current = next;  
 }  
  
 head = previous;  
 }  
  
 // ... other methods ...  
  
 private static class Node<T> {  
 private T data;  
 private Node<T> next;  
  
 public Node(T data) {  
 this.data = data;  
 }  
 }  
}  
}