public class SinglyLinkedList<E> {  
 //  
 public boolean equals(Object obj) {  
 if (this == obj) {  
 return true;  
 }  
  
 if (obj == null || getClass() != obj.getClass()) {  
 return false;  
 }  
  
 SinglyLinkedList<E> other = (SinglyLinkedList<E>) obj;  
  
 Node<E> currentThis = head;  
 Node<E> currentOther = other.head;  
  
 while (currentThis != null && currentOther != null) {  
 if (!currentThis.getElement().equals(currentOther.getElement())) {  
 return false;  
 }  
 currentThis = currentThis.getNext();  
 currentOther = currentOther.getNext();  
 }  
  
 return currentThis == null && currentOther == null;  
 }  
 //  
 public Node findSecondToLastNode(Node head) {  
 if (head == null || head.next == null) {  
 return null; // There are fewer than two nodes in the list  
 }  
  
 Node current = head;  
 Node previous = null;  
  
 while (current.next != null) {  
 previous = current;  
 current = current.next;  
 }  
  
 return previous;  
 }  
 //  
 private Node<E>head=null;  
 private Node<E>tail=null;  
 //private int size=0;  
 public SinglyLinkedList() {}  
 //  
 public int size() {  
 int count = 0;  
 Node current = head; // Assuming head is the reference to the first node  
  
 while (current != null) {  
 count++;  
 current = current.getNext(); // Assuming getNext() returns the reference to the next node  
 }  
  
 return count;  
 }  
 //-------------------------------------------------------------------------------------------  
 //public int size()  
 //{return size;}  
  
 //---------------------------------------------------------------------------------------------  
 public void rotate() {  
 if (head == null || head.next == null) {  
 return; // No rotation needed for empty or single-element list  
 }  
  
 Node<E> firstNode = head;  
 head = head.next;  
  
 Node<E> currentNode = head;  
 while (currentNode.next != null) {  
 currentNode = currentNode.next;  
 }  
 currentNode.next = firstNode;  
 firstNode.next = null;  
 }  
 //---------------------------------------------------------------------------------------------------  
 public SinglyLinkedList<E> concatenate(SinglyLinkedList<E> M) {  
 if (head == null) {  
 return M; // If L is empty, return M  
 }  
 if (M.head == null) {  
 return this; // If M is empty, return L  
 }  
  
 tail.next = M.head;  
 if (M.tail != null) {  
 tail = M.tail;//tail 1 is for L  
 }  
  
 return this; //L  
 }  
 //--------------------------------------------------------------------------------------------  
 //public class Main {  
 // public static void main(String[] args) {  
 // SinglyLinkedList<Integer> listL = new SinglyLinkedList<>();  
 // listL.addLast(1);  
 // listL.addLast(2);  
 // listL.addLast(3);  
 //  
 // SinglyLinkedList<Integer> listM = new SinglyLinkedList<>();  
 // listM.addLast(4);  
 // listM.addLast(5);  
 //  
 // listL.concatenate(listM);  
 //  
 // System.out.println("Concatenated list: " + listL);  
 // }  
 //}  
  
 //----------------------------------------------------------------------------------------------  
 public void reverse() {  
 if (head == null || head.next == null) {  
 return; // Nothing to reverse for empty or single-node list  
 }  
  
 Node<E> previous = null;  
 Node<E> current = head;  
  
while (current != null) {  
 Node<E> next = current.next;  
 current.next = previous;  
 previous = current;  
 current = next;  
 }  
  
head = previous;  
 }  
  
 //-----------------------------------------------------------------------------  
 public void addLast(E e)  
 {  
 Node<E> newest= new Node<>(e,null);  
 if (size()==0)  
 head=newest;  
 else  
 tail.setNext(newest);  
 tail=newest;  
 //size++;  
 }  
 public E removeFirst()  
 {  
 // if (isEmpty())return null;  
 E deleted= head.getElement();  
 head=head.getNext();  
 //size--;  
 if (size()==0)  
 tail=null;  
 return deleted;  
 }  
  
 private static class Node<E>{  
 E element;  
 Node<E>next;  
  
 public Node(E element, Node<E> next) {  
 this.element = element;  
 this.next = next;  
 }  
  
 public E getElement() {  
 return element;  
 }  
  
 public void setElement(E element) {  
 this.element = element;  
 }  
  
 public Node<E> getNext() {  
 return next;  
 }  
  
 public void setNext(Node<E> next) {  
 this.next = next;  
 }  
 }  
}