public class CircularlyLinkedList<E> implements Cloneable {  
 private Node<E>tail=null;  
 private int size=0;  
 public CircularlyLinkedList(){}  
 public boolean isEmpty(){  
 return size==0;}  
 //public int size(){  
 // return size;}  
  
 public E first(){  
 if (isEmpty())  
 return null;  
 return tail.next.element;  
  
 }  
 public E last(){  
 if (isEmpty())  
 return null;  
 return tail.element;  
  
 }  
 public void rotate(){  
 if (tail!=null)  
 tail=tail.next;  
 }  
 // public void addFirst(E elem){  
 // if (size==0)  
 // {tail=new Node<>(elem,null);  
 // tail.next=tail;  
 //}  
 //else {  
 // Node <E>newest=new Node<>(elem,tail.next);  
 //tail.next=newest;  
 //}  
 //size++;  
 //}  
 //1------------------------------------------------------------------------------------------------  
 public void addFirst(E e) {  
 if (size == 0) {  
 tail = new Node<>(e, null);  
 tail.next=tail; // link to itself circularly  
 } else {  
 tail.next=(new Node<>(e, tail.next));  
 tail = tail.next;  
 }  
 size++;  
 }  
 //2----------------------------------------------------------------  
 public int size() {  
 if (tail == null) {  
 return 0; // The list is empty  
 }  
  
 int count = 1; // Start with 1, as the tail node is not null  
  
 Node<E> current = tail.next;  
 while (current != tail) {  
 count++;  
 current = current.next;  
 }  
  
 return count;  
 }  
 //3----------------------------------------------------------------  
 public boolean equals(Object obj) {  
 if (this == obj) {  
 return true; // Same object reference  
 }  
  
 if (!(obj instanceof CircularlyLinkedList)) {  
 return false; // Different types, not equal  
 }  
  
 CircularlyLinkedList<?> otherList = (CircularlyLinkedList<?>) obj;  
  
 if (size() != otherList.size()) {  
 return false; // Different sizes, not equal  
 }  
  
 Node<E> thisCurrent = tail.next;  
 Node<?> otherCurrent = otherList.tail.next;  
  
 while (thisCurrent != tail) {  
 if (!thisCurrent.element.equals(otherCurrent.element)) {  
 return false; // Different elements, not equal  
 }  
  
 thisCurrent = thisCurrent.next;  
 otherCurrent = otherCurrent.next;  
 }  
  
 return true; // All elements are equal  
 }  
 //Q4-----------------------------------------------------------------  
 public static boolean sameSequence(CircularlyLinkedList<?> L, CircularlyLinkedList<?> M) {  
 if (L.isEmpty() && M.isEmpty()) {  
 return true; // Both lists are empty, same sequence  
 }  
  
 Node<?> currentL = L.tail.next;  
 Node<?> currentM = M.tail.next;  
  
 do {  
 if (!currentL.element.equals(currentM.element)) {  
 return false; // Different elements, not the same sequence  
 }  
  
 currentL = currentL.next;  
 currentM = currentM.next;  
 } while (currentL != L.tail.next && currentM != M.tail.next);  
  
 return true; // Same sequence  
 }  
 //Q5------------------------------------------------------------------  
 public static <E> CircularlyLinkedList<E>[] splitCircularList(CircularlyLinkedList<E> L) {  
 CircularlyLinkedList<E>[] result = new CircularlyLinkedList[2];  
  
 if (L.isEmpty()) {  
 result[0] = null;  
 result[1] = null;  
 return result;  
 }  
  
 Node<E> slow = L.tail.next;  
 Node<E> fast = L.tail.next;  
 Node<E> prev = L.tail;  
  
 while (fast != L.tail && fast.next != L.tail) {  
 fast = fast.next.next;  
 prev = slow;  
 slow = slow.next;  
 }  
  
 result[0] = new CircularlyLinkedList<>();  
 result[0].tail = prev;  
 result[0].tail.next=result[0].tail.next;  
  
 result[1] = new CircularlyLinkedList<>();  
 result[1].tail = L.tail;  
 result[1].tail.next=slow;  
  
 Node<E> headL1 = result[0].tail.next;  
 Node<E> headL2 = result[1].tail.next;  
  
 result[0].tail.next=headL1;  
 result[1].tail.next=headL2;  
  
 return result;  
 }  
 //Q6---------------------------------------------------------------------  
 @Override  
 public CircularlyLinkedList<E> clone() {  
 try {  
 CircularlyLinkedList<E> clonedList = (CircularlyLinkedList<E>) super.clone();  
  
 if (isEmpty()) {  
 clonedList.tail = null;  
 } else {  
 Node<E> current = tail.next;  
 Node<E> newTail = new Node<>(current.element, null);  
 clonedList.tail = newTail;  
  
 while (current != tail) {  
 Node<E> newNode = new Node<>(current.next.element, null);  
 newTail.next=newNode;  
 newTail = newNode;  
 current = current.next;  
 }  
  
 newTail.next=clonedList.tail;  
 }  
  
 return clonedList;  
 } catch (CloneNotSupportedException e) {  
 throw new InternalError(e);  
 }  
 }  
  
 //----------------------------------------------------------------------  
 public void addLast(E ppp){  
 addFirst(ppp);  
 tail=tail.next;  
 }  
 public E removeFirst(){  
 if (isEmpty())return null;  
 Node<E> n=tail.next;  
 if (n==tail)tail=null;  
 else tail.next=n.next;  
 size--;  
 return n.element;  
  
 }  
  
  
  
  
  
  
  
  
  
 private static class Node<E>  
 {  
 E element;  
 Node<E>next;  
  
 public Node(E element, Node<E> next) {  
 this.element = element;  
 this.next = next;  
 }  
  
 }  
  
}