public class DoublyLinkedList<E>{  
 // Q1  
 //The slowPtr moves forward by one node at a time  
 // the fastPtr moves forward by two nodes at a time  
 public Node findMiddleNode(Node header, Node trailer) {  
 Node slowPtr = header.getNext();  
 Node fastPtr = header.getNext();  
  
 while (fastPtr != trailer && fastPtr.getNext() != trailer) {  
 slowPtr = slowPtr.getNext();  
 fastPtr = fastPtr.getNext().getNext();  
 }  
  
 return slowPtr;  
 }  
 //Q2  
 public int size() {  
 int count = 0;  
 Node currentNode = header.getNext();  
  
 while (currentNode != trailer) {  
 count++;  
 currentNode = currentNode.getNext();  
 }  
  
 return count;  
 }  
 ///Q3  
 @Override  
 public boolean equals(Object obj) {  
 if (this == obj) {  
 return true;  
 }  
 if (obj == null || getClass() != obj.getClass()) {  
 return false;  
 }  
  
 DoublyLinkedList otherList = (DoublyLinkedList) obj;  
  
 if (this.size() != otherList.size()) {  
 return false;  
 }  
  
 Node currentNodeThis = this.header.getNext();  
 Node currentNodeOther = otherList.header.getNext();  
  
 while (currentNodeThis != this.trailer) {  
 if (!currentNodeThis.getElement().equals(currentNodeOther.getElement())) {  
 return false;  
 }  
  
 currentNodeThis = currentNodeThis.getNext();  
 currentNodeOther = currentNodeOther.getNext();  
 }  
  
 return true;  
 }  
 ///Q4  
 public DoublyLinkedList concatenateLists(DoublyLinkedList L, DoublyLinkedList M) {  
 DoublyLinkedList concatenatedList = new DoublyLinkedList();  
  
 // Connect the last node of L to the first node of M  
 Node lastNodeL = L.trailer.getPrev();  
 Node firstNodeM = M.header.getNext();  
 lastNodeL.setNext(firstNodeM);  
 firstNodeM.setPrev(lastNodeL);  
  
 // Update the trailer of L and the header of M in the concatenated list  
 concatenatedList.header = L.header;  
 concatenatedList.trailer = M.trailer;  
  
 return concatenatedList;  
 }  
 ///Q5  
// public class DoublyLinkedList<E> {  
// private Node sentinel;  
//  
// private class Node {  
// private E element;  
// private Node prev;  
// private Node next;  
//  
// public Node(E element, Node prev, Node next) {  
// this.element = element;  
// this.prev = prev;  
// this.next = next;  
// }  
// }  
//  
// public DoublyLinkedList() {  
// sentinel = new Node(null, null, null);  
// sentinel.next = sentinel;  
// sentinel.prev = sentinel;  
// }  
//  
// public void add(E element) {  
// Node newNode = new Node(element, sentinel.prev, sentinel);  
// sentinel.prev.next = newNode;  
// sentinel.prev = newNode;  
// }  
//  
// // Other methods...  
//  
// }  
 ///Q6  
 public class CircularDoublyLinkedList<E> {  
 private Node head;  
 private int size;  
  
 private class Node {  
 private E element;  
 private Node prev;  
 private Node next;  
  
 public Node(E element) {  
 this.element = element;  
 this.prev = null;  
 this.next = null;  
 }  
 }  
  
 public CircularDoublyLinkedList() {  
 head = null;  
 size = 0;  
 }  
  
 public void add(E element) {  
 Node newNode = new Node(element);  
  
 if (head == null) {  
 head = newNode;  
 head.prev = head;  
 head.next = head;  
 } else {  
 newNode.prev = head.prev;  
 newNode.next = head;  
 head.prev.next = newNode;  
 head.prev = newNode;  
 }  
  
 size++;  
 }  
  
 public void rotate() {  
 if (head != null) {  
 head = head.next;  
 }  
 }  
  
 public void rotateBackward() {  
 if (head != null) {  
 head = head.prev;  
 }  
 }  
  
 // Other methods...  
  
 }  
 //Q7  
 @SuppressWarnings("unchecked")  
 @Override  
 public DoublyLinkedList<E> clone() {  
 DoublyLinkedList<E> newList = new DoublyLinkedList<>();  
  
 Node<E> currentNode = header.getNext();  
 while (currentNode != null) {  
 newList.addLast(currentNode.element);  
 currentNode = currentNode.getNext();  
 }  
  
 return newList;  
 }  
 public void swap(Node<E> list,int i ,int j){  
 if (i==j)  
 return;  
 Node<E> CN=list;  
 Node<E> nodei=null;  
 Node<E> nodej=null;  
 int index=0;  
 while(CN!=null){  
 if(index==i)  
 nodei=CN;  
 else if (index==j)  
 nodej=CN;  
 CN=CN.next;  
 index++;  
 }  
 if (nodei!=null && nodej!=null){  
 E temp=nodei.getElement();  
 nodei.setElement(nodej.element);  
 nodej.setElement(temp);  
 }  
 }  
 private Node<E>header;  
 private Node<E>trailer;  
 private int size=0;  
 public DoublyLinkedList()  
 {  
 header=new Node<>(null,null,null);  
 trailer= new Node<>(null,header,null);  
 header.setNext(trailer);  
 }  
// public int size()  
// {return size;}  
 public boolean isEmpty(){  
 return size==0;}  
  
 public E first()  
 {if (isEmpty())return null;  
 return header.getNext().getElement();  
 }  
 public Node<E> FirstNode()  
 {if (isEmpty())return null;  
 return header.getNext();  
 }  
 public E last()  
 {if (isEmpty())return null;  
 return trailer.getPrev().getElement();  
 }  
 private void addBetween(E e,Node<E>p,Node<E>n)  
 {  
 Node<E>newest= new Node<>(e,p,n);  
 p.setNext(newest);  
 n.setPrev(newest);  
 size++;}  
 public void addFirst(E ee)  
 {  
 addBetween(ee,header,header.next);}  
 public void addLast(E ee)  
 {  
 addBetween(ee,trailer.prev,trailer);  
 }  
 private E remove(Node<E>x)  
 {Node<E>p=x.prev;  
 Node<E>n=x.next;  
 p.setNext(n);  
 n.setPrev(p);  
 size--;  
 return x.element;}  
 public E removeFirst()  
 {if (isEmpty())return null;  
 return remove(header.next);}  
 public E removeLast()  
 {if (isEmpty())return null;  
 return remove(trailer.prev);}  
 public String toString() {  
 StringBuilder sb = new StringBuilder();  
 Node<E> current = header.getNext();  
 while (current != trailer) {  
 sb.append(current.getElement()).append(" ");  
 current = current.getNext();  
 }  
 return sb.toString().trim();  
 }  
  
  
  
  
  
  
  
 private static class Node<E>  
 {  
 E element;  
 Node<E>prev;  
 Node<E>next;  
 public Node(E element, Node<E> prev, Node<E> next) {  
 this.element = element;  
 this.prev = prev;  
 this.next = next;  
 }  
  
 public E getElement() {  
 return element;  
 }  
  
 public void setElement(E element) {  
 this.element = element;  
 }  
  
 public Node<E> getPrev() {  
 return prev;  
 }  
  
 public void setPrev(Node<E> prev) {  
 this.prev = prev;  
 }  
  
 public Node<E> getNext() {  
 return next;  
 }  
  
 public void setNext(Node<E> next) {  
 this.next = next;  
 }}}