**Assignment**

1/

public void addFirst(E element) {  
 Node<E> newest = new Node<>(element);  
  
 if (isEmpty()) {  
 newest.setNext(newest);   
 tail = newest;   
 } else {  
 newest.setNext(tail.getNext());   
 tail.setNext(newest);   
 }  
  
 size++;   
 }

2\

public int size() {  
 if (isEmpty()) {  
 return 0;  
 }  
  
 int count = 1; // Start with 1 to account for the head node  
 Node<E> current = tail.getNext(); // Start from the head node  
  
 while (current != tail) {  
 count++;  
 current = current.getNext();  
 }  
  
 return count;  
 }

3\

public boolean equals(CircularlyLinkedList<E> otherList) {  
 if (size() != otherList.size()) {  
 return false;   
 }  
  
 if (isEmpty() && otherList.isEmpty()) {  
 return true;   
 }  
 Node<E> currentA = tail.getNext();   
 Node<E> currentB = otherList.tail.getNext();   
  
 while (currentA != tail && currentB != otherList.tail) {  
 if (!currentA.getElement().equals(currentB.getElement())) {  
 return false;   
 }  
 currentA = currentA.getNext();  
 currentB = currentB.getNext();  
 }  
 return currentA.getElement().equals(currentB.getElement());  
 }

4\

public boolean sameSequence(CircularlyLinkedList<E> L, CircularlyLinkedList<E> M) {  
 if (L.size() != M.size()) {  
 return false;   
 }  
  
 if (L.isEmpty() && M.isEmpty()) {  
 return true; }  
  
   
 E smallestElement = L.getHead().getElement();  
 Node<E> current = L.getHead().getNext();  
 while (current != L.getHead()) {  
 if (current.getElement().compareTo(smallestElement) < 0) {  
 smallestElement = current.getElement();  
 }  
 current = current.getNext();  
 }  
  
 current = M.getHead();  
 do {  
 if (!current.getElement().equals(smallestElement)) {  
 return false; }  
 current = current.getNext();  
 } while (current != M.getHead());  
  
 return true; }

5\

public void splitCircularLinkedList(CircularlyLinkedList<E> L) {  
 if (L.isEmpty() || L.size() % 2 != 0) {  
 throw new IllegalArgumentException("Invalid input: L must contain an even number of nodes.");  
 }  
 Node<E> slow = L.getHead();  
 Node<E> fast = L.getHead();  
 Node<E> prev = null;  
  
 while (fast.getNext() != L.getHead() && fast.getNext().getNext() != L.getHead()) {  
 fast = fast.getNext().getNext();  
 prev = slow;  
 slow = slow.getNext();  
 }  
  
 L.setTail(prev);  
  
 L2.setTail(slow.getNext());  
  
 L.getTail().setNext(L.getHead());  
  
 L2.getTail().setNext(slow.getNext());  
  
 L2.setTail(slow);  
   
 L.setTail(prev);  
 }

6\

public CircularlyLinkedList<E> clone() throws CloneNotSupportedException {  
 CircularlyLinkedList<E> newList = new CircularlyLinkedList<>();  
  
 if (isEmpty()) {  
 return newList; // Return an empty list  
 }  
  
 Node<E> current = tail.getNext(); // Start from the head node of the original list  
  
 do {  
 newList.addLast(current.getElement()); // Add the element to the new list  
 current = current.getNext();  
 } while (current != tail.getNext()); // Continue until we reach the tail of the original list  
  
 return newList;  
 }