Amal Al-Dubai / Software Engineering / Lab5

1. Consider the implementation of CircularlyLinkedList.addFirst, in Code Fragment 3.16. The else body at lines 39 and 40 of that method relies on a locally declared variable, newest. Redesign that clause to avoid use of any local variable.

```
else {
   tail.setNext(new Node<>(e, tail.getNext())); // Directly set the tail's next to the new node
}
```

2. Give an implementation of the size( ) method for the CircularlyLinkedList class, assuming that we did not maintain size as an instance variable.

```
public int size() {
   if (tail == null) {
      return 0; // The list is empty
   }

   int count = 1; // Start with one node (the node after tail)
   Node<E> current = tail.getNext(); // Start from the head (next node after tail)

   // Traverse the list until we circle back to the tail
   while (current != tail) {
      count++;
      current = current.getNext();
   }

   return count;
}
```

3.Implement the equals( ) method for the CircularlyLinkedList class, assuming that two lists are equal if they have the same sequence of elements, with corresponding elements currently at the front of the list.

```
public boolean equals(Object o) {
   if (this == o) {
      return true; // Same object reference
   }
   if (o == null || !(o instanceof CircularlyLinkedList<?>)) {
      return false; // Null or not the same type
   }

   CircularlyLinkedList<?> other = (CircularlyLinkedList<?>) o;

   // Check if sizes are different
   if (this.size() != other.size()) {
      return false;
   }

   if (this.isEmpty() && other.isEmpty()) {
      return true; // Both lists are empty
   }

   Node<E> currentA = this.tail.getNext(); // Head of this list
   Node<?> currentB = other.tail.getNext(); // Head of other list

   // Compare corresponding elements
   do {
      if (!currentA.getElement().equals(currentB.getElement())) {
         return false; // Mismatch found
      }
      currentA = currentA.getNext();
      currentB = currentB.getNext();
   } while (currentA != this.tail.getNext()); // Stop when we've completed the circle

   return true; // All elements matched
}
```

4.Suppose you are given two circularly linked lists, L and M. Describe an algorithm for telling if L and M store the same sequence of elements (but perhaps with different starting points).

```
public class CircularlyLinkedList<E> {

  private static class Node<E> {
    private E element;
    private Node<E> next;

    public Node(E element, Node<E> next) {
      this.element = element;
      this.next = next;
    }

    public E getElement() {
      return element;
    }

    public Node<E> getNext() {
      return next;
    }

    public void setNext(Node<E> next) {
      this.next = next;
    }
  }

  private Node<E> tail = null;
  private int size = 0;

  public int size() {
```

```
    return size;
}

public boolean isEmpty() {
    return size == 0;
}

public E first() {
    if (isEmpty()) return null;
    return tail.getNext().getElement();
}

public void rotate() {
    if (tail != null)
        tail = tail.getNext();
}

public void addLast(E e) {
    Node<E> newest = new Node<>(e, null);
    if (isEmpty()) {
        newest.setNext(newest);
    } else {
        newest.setNext(tail.getNext());
        tail.setNext(newest);
    }
    tail = newest;
    size++;
}

public void addFirst(E e) {
    Node<E> newest = new Node<>(e, null);
    if (isEmpty()) {
        newest.setNext(newest);
```

```
        tail = newest;
    } else {
        newest.setNext(tail.getNext());
        tail.setNext(newest);
    }
    size++;
}

public static <E> boolean areCircularListsEquivalent(CircularlyLinkedList<E> L, CircularlyLinkedList<E> M) {
    if (L.size() != M.size()) return false;
    if (L.isEmpty() && M.isEmpty()) return true;

    Node<E> headL = L.tail.getNext();
    Node<E> currentM = M.tail.getNext();

    do {
        if (headL.getElement().equals(currentM.getElement())) {
            Node<E> tempL = headL;
            Node<E> tempM = currentM;
            boolean isEqual = true;
            do {
                if (!tempL.getElement().equals(tempM.getElement())) {
                    isEqual = false;
                    break;
                }
                tempL = tempL.getNext();
                tempM = tempM.getNext();
            } while (tempL != headL);

            if (isEqual) return true;
        }
        currentM = currentM.getNext();
    } while (currentM != M.tail.getNext());
```

```
    return false;
}

public void printList() {
    if (isEmpty()) {
        System.out.println("List is empty.");
        return;
    }
    Node<E> current = tail.getNext();
    do {
        System.out.print(current.getElement() + " ");
        current = current.getNext();
    } while (current != tail.getNext());
    System.out.println();
}

public static void main(String[] args) {
    CircularlyLinkedList<Integer> L = new CircularlyLinkedList<>();
    CircularlyLinkedList<Integer> M = new CircularlyLinkedList<>();

    L.addLast(1);
    L.addLast(2);
    L.addLast(3);
    L.addLast(4);

    M.addLast(3);
    M.addLast(4);
    M.addLast(1);
    M.addLast(2);

    System.out.print("L: ");
    L.printList();
```

```
        System.out.print("M: ");
        M.printList();

        boolean areEquivalent = CircularlyLinkedList.areCircularListsEquivalent(L, M);
        System.out.println("Are L and M equivalent? " + areEquivalent);
    }
}
```

5.Given a circularly linked list L containing an even number of nodes, describe how to split L into two circularly linked lists of half the size.

```
public class CircularlyLinkedList<E> {

    private static class Node<E> {
        private E element;
        private Node<E> next;

        public Node(E element, Node<E> next) {
            this.element = element;
            this.next = next;
        }

        public E getElement() {
            return element;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> next) {
            this.next = next;
        }
    }

    private Node<E> tail = null;
    private int size = 0;

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void addLast(E e) {
        Node<E> newest = new Node<>(e, null);
        if (isEmpty()) {
            newest.setNext(newest);
        } else {
```

```
        newest.setNext(tail.getNext());
        tail.setNext(newest);
    }
    tail = newest;
    size++;
}

public void printList() {
    if (isEmpty()) {
        System.out.println("List is empty.");
        return;
    }
    Node<E> current = tail.getNext();
    do {
        System.out.print(current.getElement() + " ");
        current = current.getNext();
    } while (current != tail.getNext());
    System.out.println();
}

public CircularlyLinkedList<E>[] split() {
    if (size % 2 != 0) throw new IllegalStateException("List size must be even.");

    CircularlyLinkedList<E> L1 = new CircularlyLinkedList<>();
    CircularlyLinkedList<E> L2 = new CircularlyLinkedList<>();

    Node<E> current = tail.getNext();
    int halfSize = size / 2;

    for (int i = 0; i < halfSize; i++) {
        L1.addLast(current.getElement());
        current = current.getNext();
    }

    for (int i = 0; i < halfSize; i++) {
        L2.addLast(current.getElement());
        current = current.getNext();
    }

    return new CircularlyLinkedList[]{L1, L2};
}

public static void main(String[] args) {
    CircularlyLinkedList<Integer> L = new CircularlyLinkedList<>();

    L.addLast(1);
    L.addLast(2);
    L.addLast(3);
    L.addLast(4);

    System.out.print("Original List: ");
    L.printList();

    CircularlyLinkedList<Integer>[] result = L.split();
    CircularlyLinkedList<Integer> L1 = result[0];
    CircularlyLinkedList<Integer> L2 = result[1];
```

```
        System.out.print("L1: ");
        L1.printList();
        System.out.print("L2: ");
        L2.printList();
    }
}
```

6.Implement the clone( ) method for the CircularlyLinkedList class.

```
public class CircularlyLinkedList<E> {

    private static class Node<E> {
        private E element;
        private Node<E> next;

        public Node(E element, Node<E> next) {
            this.element = element;
            this.next = next;
        }

        public E getElement() {
            return element;
        }

        public Node<E> getNext() {
            return next;
        }

        public void setNext(Node<E> next) {
            this.next = next;
        }
    }

    private Node<E> tail = null;
    private int size = 0;

    public int size() {
        return size;
    }

    public boolean isEmpty() {
        return size == 0;
    }

    public void addLast(E e) {
        Node<E> newest = new Node<>(e, null);
        if (isEmpty()) {
            newest.setNext(newest);
        } else {
            newest.setNext(tail.getNext());
            tail.setNext(newest);
        }
        tail = newest;
        size++;
    }
```

```
    public void printList() {
        if (isEmpty()) {
            System.out.println("List is empty.");
            return;
        }
        Node<E> current = tail.getNext();
        do {
            System.out.print(current.getElement() + " ");
            current = current.getNext();
        } while (current != tail.getNext());
        System.out.println();
    }

    public CircularlyLinkedList<E> clone() {
        CircularlyLinkedList<E> cloneList = new CircularlyLinkedList<>();

        if (isEmpty()) {
            return cloneList; // Return an empty list if the original is empty
        }

        Node<E> current = tail.getNext();
        do {
            cloneList.addLast(current.getElement());
            current = current.getNext();
        } while (current != tail.getNext());

        return cloneList;
    }

    public static void main(String[] args) {
        CircularlyLinkedList<Integer> original = new CircularlyLinkedList<>();

        original.addLast(1);
        original.addLast(2);
        original.addLast(3);
        original.addLast(4);

        System.out.print("Original List: ");
        original.printList();

        CircularlyLinkedList<Integer> cloned = original.clone();
        System.out.print("Cloned List: ");
        cloned.printList();
    }
}
```