Data Structure Lab5 : Circularly Linked List 2022-2023

Topics: CircularlyLinkedList Implementation

1. Node Class

```
java
Copy code
public class Node<E> {
    E element; // Data stored in this node
    Node<E> next; // Reference to the next node in the list

    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;
    }
}
```

Data Structure Lab5 : Circularly Linked List 2022-2023

2. CircularlyLinkedList Class

```
java
Copy code
public class CircularlyLinkedList<E> {
    private Node<E> tail = null; // Tail node of the list
    private int size = 0; // Number of elements in the list

    public CircularlyLinkedList() { }

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

    // size
    public int size() {
        return size;
    }

    // first
    public E first() {
        if (isEmpty()) return null;
        return tail.getNext().getElement(); // Head is the node after the tail
    }

    // last
    public E last() {
        if (isEmpty()) return null;
        return tail.getElement();
    }

    // addFirst
    public void addFirst(E element) {
        Node<E> newNode = new Node<>(element, null);
        if (isEmpty()) {
            newNode.setNext(newNode); // Circular reference
            tail = newNode;
        } else {
            newNode.setNext(tail.getNext());
            tail.setNext(newNode);
        }
        size++;
    }

    // addLast
    public void addLast(E element) {
        addFirst(element); // Add to the front
        tail = tail.getNext(); // Update tail to the new last node
    }

    // removeFirst
    public E removeFirst() {
        if (isEmpty()) return null;

        Node<E> head = tail.getNext();
        if (head == tail) { // Single element case
            tail = null;
        } else {
            tail.setNext(head.getNext());
        }
        size--;
        return head.getElement();
    }

    // rotate
    public void rotate() {
        if (tail != null) {
            tail = tail.getNext(); // Tail points to what was previously the head
        }
    }
}
```

Data Structure Lab5 : Circularly Linked List 2022-2023

Homework

1. Redesign `addFirst` Without Local Variable

Here's a version of `addFirst` that avoids using the `newest` variable:

```
java
Copy code
public void addFirst(E element) {
    if (isEmpty()) {
        tail = new Node<>(element, null);
        tail.setNext(tail); // Circular reference
    } else {
        tail.setNext(new Node<>(element, tail.getNext()));
    }
    size++;
}
```

---

2. Implement `size()` Without Using `size` Variable

The `size()` method can traverse the circularly linked list to count nodes:

```
java
Copy code
public int sizeWithoutVariable() {
    if (isEmpty()) return 0;

    int count = 0;
    Node<E> current = tail.getNext(); // Start at the head
    do {
        count++;
        current = current.getNext();
    } while (current != tail.getNext()); // Stop when we return to the head

    return count;
}
```

Data Structure Lab5 : Circularly Linked List 2022-2023

3. Implement `equals()` for CircularlyLinkedList

Two circularly linked lists are equal if they have the same sequence of elements with corresponding elements at the front of the list.

```
java
Copy code
@Override
public boolean equals(Object o) {
    if (o == null || getClass() != o.getClass()) return false;
    CircularlyLinkedList<?> other = (CircularlyLinkedList<?>) o;

    if (size() != other.size()) return false;
    if (isEmpty() && other.isEmpty()) return true;

    Node<E> currentA = tail.getNext(); // Start at the head
    Node<?> currentB = other.tail.getNext();

    do {
        if (!currentA.getElement().equals(currentB.getElement())) return
false;
        currentA = currentA.getNext();
        currentB = currentB.getNext();
    } while (currentA != tail.getNext());

    return true;
}
```

Data Structure Lab5 : Circularly Linked List 2022-2023

4. Check if Two CircularlyLinkedLists Store the Same Sequence

Algorithm:

1. Ensure both lists have the same size.
2. Find if one list's sequence matches another's sequence from any starting point.
3. Rotate one list iteratively and compare.

```
java
Copy code
public boolean isSameSequence(CircularlyLinkedList<E> other) {
    if (size() != other.size()) return false;

    Node<E> startA = tail.getNext();
    for (int i = 0; i < size; i++) {
        Node<E> currentA = startA;
        Node<E> currentB = other.tail.getNext();
        boolean match = true;

        for (int j = 0; j < size; j++) {
            if (!currentA.getElement().equals(currentB.getElement())) {
                match = false;
                break;
            }
            currentA = currentA.getNext();
            currentB = currentB.getNext();
        }

        if (match) return true;
        startA = startA.getNext(); // Rotate start point
    }

    return false;
}
```

Data Structure Lab5 : Circularly Linked List 2022-2023

5. Split CircularlyLinkedList Into Two Halves

Algorithm:

1. Traverse the list to find the midpoint.
2. Split into two lists, setting tails appropriately.

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

    CircularlyLinkedList<E> firstHalf = new CircularlyLinkedList<>();
    CircularlyLinkedList<E> secondHalf = new CircularlyLinkedList<>();

    Node<E> current = tail.getNext(); // Start at the head
    for (int i = 0; i < size / 2; i++) {
        firstHalf.addLast(current.getElement());
        current = current.getNext();
    }

    for (int i = size / 2; i < size; i++) {
        secondHalf.addLast(current.getElement());
        current = current.getNext();
    }

    return new CircularlyLinkedList[]{firstHalf, secondHalf};
}
```

---

6. Implement `clone()` Method

```
java
Copy code
@Override
public CircularlyLinkedList<E> clone() {
    CircularlyLinkedList<E> cloned = new CircularlyLinkedList<>();
    if (!isEmpty()) {
        Node<E> current = tail.getNext(); // Start at the head
        do {
            cloned.addLast(current.getElement());
            current = current.getNext();
        } while (current != tail.getNext());
    }
    return cloned;
}
```