Topics

1. Implement Node Class
2. Implement CircularlyLinkedList Class
3. Implement Basic Methods of CircularlyLinkedList
    - isEmpty()
    - size()
    - first()
    - last()
    - addFirst()
    - addLast()
    - removeFirst()
    - rotate()

1. Implement Node Class

```
public class Node {
        int data;
       Node next;

    public Node(int data) {
      this.data = data;
       this.next = null;
  }
}
```

2. Implement CircularlyLinkedList Class

```
public class CircularlyLinkedList {
   Node head;

   public CircularlyLinkedList() {
      this.head = null;
   }
}
```

3. Implement Basic Methods of CircularlyLinkedList

```
public class CircularlyLinkedList {
    Node head;

    public CircularlyLinkedList() {
        this.head = null;
    }

    public boolean isEmpty() {
        return head == null;
    }

    public int size() {
        if (isEmpty()) {
            return 0;
        }
        int count = 1;
        Node current = head;
        while (current.next != head) {
            current = current.next;
            count++;
        }
        return count;
    }

    public int first() {
        if (isEmpty()) {
            throw new RuntimeException("List is empty");
        }
        return head.data;
    }

    public int last() {
        if (isEmpty()) {
            throw new RuntimeException("List is empty");
        }
        Node current = head;
        while (current.next != head) {
            current = current.next;
        }
        return current.data;
    }
```

```
    public void addFirst(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = newNode;
            newNode.next = newNode; // For circularly linked list
        } else {
            Node current = head;
            while (current.next != head) {
                current = current.next;
            }
            current.next = newNode;
            newNode.next = head;
            head = newNode;
        }
    }

    public void addLast(int data) {
        Node newNode = new Node(data);
        if (isEmpty()) {
            head = newNode;
            newNode.next = newNode; // For circularly linked list
        } else {
            Node current = head;
            while (current.next != head) {
                current = current.next;
            }
            current.next = newNode;
            newNode.next = head;
        }
    }

    public int removeFirst() {
        if (isEmpty()) {
            throw new RuntimeException("List is empty");
        }
        int removedData = head.data;
        if (head.next == head) { // For removing the only node in the list
            head = null;
        } else {
            Node current = head;
            while (current.next != head) {
                current = current.next;
            }
            current.next = head.next;
```

```
            head = head.next;
        }
        return removedData;
    }

    public void rotate() {
        if (isEmpty()) {
            throw new RuntimeException("List is empty");
        }
        head = head.next;
    }
}
```

Homework

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.

```
public void addFirst(int data) {
    Node newest = new Node(data);
    if (isEmpty()) {
        head = newest;
        head.next = head; // the list is circular
    } else {
        newest.next = head;
        head = newest;
    }
}
```

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 (isEmpty()) {
        return 0;
    }
    int count = 1; // start counting from the head
    Node current = head;
    while (current.next != head) {
        count++;
        current = current.next;
    }
    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 (o == this) {
        return true;
    }
    if (!(o instanceof CircularlyLinkedList)) {
        return false;
    }
    CircularlyLinkedList other = (CircularlyLinkedList) o;
    if (size() != other.size()) {
        return false;
    }
    Node current1 = head;
    Node current2 = other.head;
    while (current1 != head) {
        if (current1.data != current2.data) {
            return false;
        }
        current1 = current1.next;
        current2 = current2.next;
    }
    return true;
}
```

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). To compare two circularly linked lists, L and M, we can iterate through their elements simultaneously and compare corresponding elements. If we find any pair of elements that do not match, we return false. If we can successfully iterate through all elements of both lists without finding any mismatched elements, we return true.

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.

To split a circularly linked list L with an even number of nodes into two circularly linked lists of half the size, we can use a two-pointer approach. One pointer moves twice as fast as the other. When the fast pointer reaches the end of the list, the slow pointer will be at the midpoint. At this point, we can break the list into two halves by changing the next pointer of the slow pointer to the head of the list.

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

```
public CircularlyLinkedList clone() {
    if (isEmpty()) {
        return new CircularlyLinkedList();
    }
    CircularlyLinkedList clone = new CircularlyLinkedList();
    Node current = head;
    do {
        clone.addLast(current.data);
        current = current.next;
    } while (current != head);
    return clone;
}
```