Data Structure Lab5 : Circularly Linked List 2022-2023

Topics

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

```
// تعريف Node Class
public class Node<E> {
    private E element; // العنصر المخزن في العقدة
    private Node<E> next; // الإشارة إلى العقدة التالية

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

    public E getElement() {
        return element;
    }

    public void setElement(E element) {
        this.element = element;
    }

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

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

// تعريف CircularlyLinkedList Class
public class CircularlyLinkedList<E> {
    private Node<E> tail = null; // الإشارة إلى العنصر الأخير
    private int size = 0; // عدد العقد في القائمة

    // منشئ فارغ
    public CircularlyLinkedList() {}
```

Data Structure Lab5 : Circularly Linked List 2022-2023

```
// التحقق إذا كانت القائمة فارغة
public boolean isEmpty() {
    return size == 0;
}

// إعادة عدد العناصر
public int size() {
    return size;
}

// إرجاع العنصر الأول
public E first() {
    if (isEmpty()) return null; // إذا كانت القائمة فارغة
    return tail.getNext().getElement(); // العنصر الأول بعد tail
}

// إرجاع العنصر الأخير
public E last() {
    if (isEmpty()) return null; // إذا كانت القائمة فارغة
    return tail.getElement(); // العنصر المخزن في tail
}

// إضافة عنصر في بداية القائمة
public void addFirst(E element) {
    if (isEmpty()) { // إذا كانت القائمة فارغة
        tail = new Node<>(element, null);
        tail.setNext(tail); // الإشارة إلى نفسها لجعلها دائرية
    } else {
        Node<E> newNode = new Node<>(element, tail.getNext()); // إنشاء عقدة جديدة
        tail.setNext(newNode); // ربط tail بالعقدة الجديدة
    }
    size++; // زيادة عدد العقد
}

// إضافة عنصر في نهاية القائمة
public void addLast(E element) {
    addFirst(element); // أضف العنصر في البداية
    tail = tail.getNext(); // قم بتحديث tail ليُشير إلى العقدة الجديدة
}

// إزالة العنصر الأول
public E removeFirst() {
    if (isEmpty()) return null; // إذا كانت القائمة فارغة

    Node<E> head = tail.getNext(); // العقدة الأولى
    if (head == tail) { // إذا كانت هناك عقدة واحدة فقط
        tail = null; // القائمة تصبح فارغة
    } else {
        tail.setNext(head.getNext()); // تجاوز العقدة الأولى
```

Data Structure Lab5 : Circularly Linked List 2022-2023

```
    }
    size--; // تقليل عدد العقد
    return head.getElement(); // إعادة العنصر الذي تمت إزالته
}

// تدوير القائمة
public void rotate() {
    if (tail != null) { // إذا لم تكن القائمة فارغة
        tail = tail.getNext(); // اجعل العنصر الأول هو العنصر الأخير
    }
}

// (طباعة عناصر القائمة (للمساعدة في التحقق
public void printList() {
    if (isEmpty()) {
        System.out.println("The list is empty.");
        return;
    }

    Node<E> current = tail.getNext(); // البداية من العنصر الأول
    do {
        System.out.print(current.getElement() + " -> ");
        current = current.getNext();
    } while (current != tail.getNext()); // الاستمرار حتى نعود إلى البداية
    System.out.println("(back to start)");
}

// اختبار القائمة
public static void main(String[] args) {
    CircularlyLinkedList<Integer> list = new CircularlyLinkedList<>();

    // إضافة عناصر
    list.addLast(10);
    list.addLast(20);
    list.addLast(30);
    list.printList(); // 10 -> 20 -> 30 -> (back to start)

    // إضافة عنصر في البداية
    list.addFirst(5);
    list.printList(); // 5 -> 10 -> 20 -> 30 -> (back to start)

    // إزالة العنصر الأول
    System.out.println("Removed: " + list.removeFirst()); // Removed: 5
    list.printList(); // 10 -> 20 -> 30 -> (back to start)

    // تدوير القائمة
    list.rotate();
    list.printList(); // 20 -> 30 -> 10 -> (back to start)

    // الحصول على العنصر الأول والأخير
```

Data Structure Lab5 : Circularly Linked List 2022-2023

```
        System.out.println("First: " + list.first()); // First: 20
        System.out.println("Last: " + list.last()); // Last: 10
    }
}
```

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(E element) {
    if (tail == null) { // إذا كانت القائمة فارغة
        tail = new Node<>(element, null);
        tail.setNext(tail); // الإشارة إلى نفسها لجعلها دائرية
    } else {
        // إنشاء العقدة الجديدة مباشرةً وربطها بالبداية
        tail.setNext(new Node<>(element, tail.getNext()));
    }
}
```

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; // القائمة فارغة
    int count = 1; // البداية من العقدة الأولى
    Node<E> current = tail.getNext(); // نبدأ من العقدة الأولى
    while (current != tail) { // نستمر حتى نعود إلى العقدة الأخيرة
        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.**

```
@Override
public boolean equals(Object o) {
    if (o == null || !(o instanceof CircularlyLinkedList)) return false;

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

    // تحقق من الحجم أولاً
    if (this.size() != other.size()) return false;

    Node<E> currentA = this.tail.getNext(); // العقدة الأولى للقائمة الحالية
    Node<?> currentB = other.tail.getNext(); // العقدة الأولى للقائمة الأخرى
```

Data Structure Lab5 : Circularly Linked List 2022-2023

```
    // قارن كل العناصر
    for (int i = 0; i < this.size(); i++) {
        if (!currentA.getElement().equals(currentB.getElement())) return false;
        currentA = currentA.getNext();
        currentB = currentB.getNext();
    }
    return true; // جميع العناصر متطابقة
}
```

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 boolean isSameSequence(CircularlyLinkedList<E> other) {
    if (this.size() != other.size()) return false;

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

        boolean match = true;
        for (int j = 0; j < this.size(); j++) {
            if (!currentA.getElement().equals(currentB.getElement())) {
                match = false;
                break;
            }
            currentA = currentA.getNext();
            currentB = currentB.getNext();
        }
        if (match) return true;
        startA = startA.getNext(); // غيّر نقطة البداية
    }
    return false;
}
```

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

    CircularlyLinkedList<E> list1 = new CircularlyLinkedList<>();
    CircularlyLinkedList<E> list2 = new CircularlyLinkedList<>();

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

    for (int i = 0; i < this.size(); i++) {
        if (i < mid) {
            list1.addLast(current.getElement());
        } else {
            list2.addLast(current.getElement());
```

Data Structure Lab5 : Circularly Linked List 2022-2023

```
            }
            current = current.getNext();
        }

        return new CircularlyLinkedList[]{list1, list2};
    }
```

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

```
    @Override
    public CircularlyLinkedList<E> clone() {
        CircularlyLinkedList<E> cloned = new CircularlyLinkedList<>();
        if (this.tail == null) return cloned;

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

        return cloned;
    }
```