Lab 04

Circular Linked List

Objective:

In this lab session we will enhance singly-linked lists with another feature: we'll make them circular. And, as was the case in the previous lab session, we will show how to implement creation, insertion, and removal of nodes.

Activity Outcomes:

This lab teaches you the following topics:

- Creation of circular linked list
- Insertion in circular linked list
- Deletion from circular linked list
- Traversal of all nodes

1) Useful Concepts

A circular singly linked list is a singly linked list which has the last element linked to the first element in the list. Being circular it really has no ends; then we'll use only one pointer pNode to indicate one element in the list - the newest element. Figure 3.1 show a model of such a list. pNode

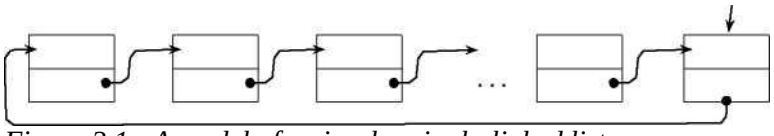

Figure 3.1.: A model of a circular singly-linked list.

```
public class Node {

    int value;
    Node nextNode;

    public Node(int value) {
        this.value = value;
         this.nextNode = null;
    }
}
```

Activity 1:

Creation of Circular linked list

create the first and last nodes in the circular linked list (`CircularLinkedList.java`)

```
private Node head = null;
private Node tail = null;
```

Inserting Elements

```
  public void addNode(int value) {
            Node newNode = new Node(value);

            if (head == null) {
                head = newNode;
            } else {
                tail.nextNode = newNode;
            }

            tail = newNode;
            tail.nextNode = head;
        }//end addNode
```

Activity 2:

Accessing nodes of Circular linked list

```
public Node searchKey(int givenKey) {
             Node currentNode = head;
             if (currentNode != null) {
                 do {
                     if (currentNode.value == givenKey) {
                         // key found at address p
                         return currentNode;
                     }
                     currentNode = currentNode.nextNode;
```

```
            } while (currentNode != head);
        }
        return null; // not found
    }//end
```

Output

Return the index of Node if the required element is found.

Activity 3:

Insertion in circular linked list

- ***Inserting Before*** *a node with key givenKey*

There are two steps to execute:

1. *Find the node with key givenKey:*
2. *Insert the node pointed to by p, and adjust links:*

```
public void insertBefore(int givenKey,int newValue)
    {
        Node newNode , currentNode, nextNode;

        nextNode = null; // initialize
        currentNode = head;
        newNode = new Node(newValue);

        do {
            nextNode = currentNode;
            currentNode = currentNode.nextNode;
            if ( currentNode.value == givenKey ) break;
            } while ( currentNode != head );

        if ( currentNode.value == givenKey ) {
          // node with key givenKey has address q
            nextNode.nextNode = newNode;
          newNode.nextNode = currentNode;
          }
    }//end insertBefore
```

- ***Insertion after*** *a node with key Again, there are two steps to execute:*

```
public void insertAfter(int givenKey,int newValue)
        {
            //TASK: COMPLETE THE CODE


        }//end insertAfter
```

Activity 4:

Deleting a node from circular linked list

Again there are two steps to take:

Find the node with key givenKey:

Delete the node pointed to by currentNode. If that node is head then we adjust head to point to its previous.

```
public void deleteNode(int givenKey)
        {

            Node  currentNode, nextNode;

            nextNode = null; // initialize
            currentNode = head;


            do {
                 nextNode = currentNode;
                 currentNode = currentNode.nextNode;
                 if ( currentNode.value == givenKey ) break;
                 } while ( currentNode != head );

            if ( currentNode.value == givenKey )
            {
                 if ( currentNode == currentNode.nextNode )
                 {
```

```
                    System.out.println("List now empty");
                }
                else
                {
                  nextNode.nextNode = currentNode.nextNode;
                  if(currentNode == head)
                       head = nextNode;
                }//end else
            }//end if

        }//end deleteNode
```

Activity 5:

Complete deletion of Circular linked list

```
    public void deleteNodeAll()
       {

            Node  currentNode, nextNode;

            nextNode = null; // initialize
            currentNode = head;


            do {
                nextNode = currentNode;
                currentNode = currentNode.nextNode;

                 } while ( currentNode != head );

            head = null;

        }//end deleteNodeAll
```

Output

All the nodes of Circular Linked List will be deleted.

2) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab

Lab Task 1

Write a function that deletes all those nodes from a linked list which have even/odd numbered value in their data part.

Lab Task 2

Write a function that implements Josephus problem.

Lab Task 3

Write a function that deletes all even positioned nodes from a linked list. Last node should also be deleted if its position is even.