Data Structure and Algorithms (JAVA)



**5th Lab**

**Semester: Spring 2025**

**Software Engineering**

**Faculty of Information Technology, UCP Lahore, Pakistan**

Table of Contents

[Session 1: Common Operations of Singly Circular Linked List 3](#_Toc195649800)

[1.1 Node Structure 3](#_Toc195649801)

[1.2 Singly Circular Linked List Operations 4](#_Toc195649802)

[1.3 Display function in Singly Circular Linked List: 4](#_Toc195649803)

[Session 2: Common Operations on Doubly Circular Linked List 5](#_Toc195649804)

[2.1 Node Structure 5](#_Toc195649805)

[2.3 Display of Doubly Circular Linked List 6](#_Toc195649806)

[Session 3: Stack Implementation 7](#_Toc195649807)

[Session 4: Complexity Analysis 9](#_Toc195649808)

[Session 5: Scenario-Based Case Studies 10](#_Toc195649809)

[Case Study 1: Playlist Circular Navigation 10](#_Toc195649810)

[Case Study 2: Text Editor Undo Feature 10](#_Toc195649811)

[Conclusion 10](#_Toc195649812)

**Lab Manual: Data Structures and Algorithms using Java**

Objective

In this lab, students will implement circular linked lists and stacks using various approaches in Java. This includes developing an understanding of singly and doubly circular lists, and different methods for implementing stack operations. The focus will be on practical use cases and time complexity analysis.

## **Session 1: Common Operations of Singly Circular Linked List**

### **1.1 Node Structure**

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

### **1.2 Singly Circular Linked List Operations**

class LinkedList {  
 Node head = null;  
 Node tail = null;  
  
 public void add(int data) {  
 Node newNode = new Node(data);  
 if (head == null) {  
 head = newNode;  
 tail = newNode;  
 newNode.next = head;  
 } else {  
 tail.next = newNode;  
 tail = newNode;  
 tail.next = head;  
 }  
 }

### **1.3 Display function in Singly Circular Linked List:**

public void display() {  
 Node current = head;  
 if (head != null) {  
 do {  
 System.out.print(current.data + " ");  
 current = current.next;  
 } while (current != head);  
 }  
 System.out.println();  
 }  
}

### **Session 2: Common Operations on Doubly Circular Linked List**

### 2.1 Node Structure

class DNode {  
 int data;  
 DNode next, prev;  
 DNode(int data) {  
 this.data = data;  
 }  
}

2.2 Doubly Circular Linked List Operations

class LinkedList {  
 DNode head = null;  
 DNode tail = null;  
  
 public void add(int data) {  
 DNode newNode = new DNode(data);  
 if (head == null) {  
 head = tail = newNode;  
 head.next = head.prev = head;  
 } else {  
 tail.next = newNode;  
 newNode.prev = tail;  
 newNode.next = head;  
 head.prev = newNode;  
 tail = newNode;  
 }  
 }

### **2.3 Display of Doubly Circular Linked List**

public void display() {  
 DNode temp = head;  
 if (head != null) {  
 do {  
 System.out.print(temp.data + " ");  
 temp = temp.next;  
 } while (temp != head);  
 }  
 System.out.println();  
 }  
}

## **Session 3: Stack Implementation**

**3.1 Stack using Arrays**

class ArrayStack {  
 int[] stack;  
 int top;  
 int capacity;  
  
 ArrayStack(int size) {  
 stack = new int[size];  
 top = -1;  
 capacity = size;  
 }  
  
 public void push(int data) {  
 if (top == capacity - 1) {  
 System.out.println("Stack Overflow");  
 } else {  
 stack[++top] = data;  
 }  
 }  
  
 public void pop() {  
 if (top == -1) {  
 System.out.println("Stack Underflow");  
 } else {  
 top--;  
 }  
 }  
  
 public int peek() {  
 return stack[top];  
 }  
}

3.2 Stack using Linked List

class StackNode {  
 int data;  
 StackNode next;  
 StackNode(int data) {  
 this.data = data;  
 }  
}  
class LinkedListStack {  
 StackNode top;  
 public void push(int data) {  
 StackNode newNode = new StackNode(data);  
 newNode.next = top;  
 top = newNode;  
 }  
 public void pop() {  
 if (top == null) {  
 System.out.println("Stack Underflow");  
 } else {  
 top = top.next;  
 }  
 }  
 public int peek() {  
 return top.data;  
 }  
}

3.3 Stack using Java Collection Framework

import java.util.Stack;  
  
Stack<Integer> stack = new Stack<>();  
stack.push(10);  
stack.push(20);  
System.out.println(stack.pop());  
System.out.println(stack.peek());

## **Session 4: Complexity Analysis**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Method (Array / LL / JCF)** | **Time Complexity** |
| Push element | push(element) | O(1) |
| Pop element | pop() | O(1) |
| Peek top element | peek() | O(1) |
| Check if stack is empty | isEmpty() | O(1) |
| Traverse the stack | loop through stack | O(n) |
| Search element (JCF only) | search(element) | O(n) |
| Create stack (array) | constructor with size | O(1) |
| Create stack (linked list) | add nodes manually | O(1) per node |
| Create stack (JCF) | Stack<Type> s = new Stack<>(); | O(1) |
| Capacity check (array) | top == size - 1 | O(1) |

## **Session 5: Scenario-Based Case Studies**

### **Case Study 1: Playlist Circular Navigation**

**Scenario:**

A music app uses a circular linked list to store a playlist of songs. Users can move to the next or previous song and the list wraps around when reaching the end.

**Tasks:**

1. Add a new song to the playlist
2. Navigate to the next and previous song
3. Remove a song from the playlist
4. Display the current playlist in order

## **Case Study 2: Text Editor Undo Feature**

**Scenario:**

A text editor uses a stack to track user actions. Each action (insert/delete) is pushed onto the stack. Users can undo the most recent action.

**Tasks:**

1. Push a new action to the stack
2. Undo last action (pop from the stack)
3. Peek current state
4. Display all actions

### **Conclusion**

In this lab, students developed core understanding and implementation of circular linked lists and stacks using Java arrays, linked lists, and Java’s Collection Framework. Real-world scenarios were used to demonstrate practical applications of these data structures in music apps and text editors.