Data Structure and Algorithms (JAVA)



**4th Lab**

**Semester: Spring 2025**

**Software Engineering**

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

Table of Contents

[Session 1: Common Operations and Complexity Analysis 4](#_Toc195044699)

[1.1 Node Structure 4](#_Toc195044700)

[Creating a Node for Browser History Navigation System 4](#_Toc195044701)

[1.2 Doubly Linked List Operations 4](#_Toc195044702)

[Insert page at the end of the History 4](#_Toc195044703)

[Go back to the previous page and move forward to the next page 5](#_Toc195044704)

[Go back to the first page in the history and last visited page 5](#_Toc195044705)

[Remove specific page from the history 6](#_Toc195044706)

[Clear History 6](#_Toc195044707)

[Get the current page URL 6](#_Toc195044708)

[Session 2: Linked List Built-In Functions 7](#_Toc195044709)

[2.4Check for an Element 7](#_Toc195044710)

[2.5 Traverse the List 7](#_Toc195044711)

[Session 3: Complexity Analysis 8](#_Toc195044712)

[Session 4: Scenario-Based Case Studies 9](#_Toc195044713)

[Case Study 1: Multiplayer Game Turn Management 9](#_Toc195044714)

[Case Study 2: Shopping Cart with Undo/Redo Feature 9](#_Toc195044715)

[Conclusion 9](#_Toc195044716)

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

Objective

In this lab, students will gain hands-on experience with doubly linked lists, circular linked lists, and Java’s LinkedList class, focusing on their operations and time complexities. By the end of the lab, students will be able to:

• Implement basic operations on doubly and circular linked lists.  
• Compare the time complexities and performance of singly, doubly, and circular linked lists.  
• Analyze key linked list operations such as reversal, merging, and cycle detection.  
• Understand the practical applications and trade-offs of different linked list structures.

## **Session 1: Common Operations and Complexity Analysis**

### **1.1 Node Structure**

### **Creating a Node for Browser History Navigation System**

class Node {

String url;

Node next;

Node prev;

public Node(String url) {

this.url = url;

this.next = null;

this.prev = null;

}

}

### **1.2 Doubly Linked List Operations**

### **I**nsert page at the end of the History

public void insertAtEnd(String url) {

Node newNode = new Node(url);

// If the current page is not the last in history, clear the forward history

if (currentPage.next != null) {

currentPage.next = null; // Clear forward history when a new page is visited

}

// Link the new page

currentPage.next = newNode;

newNode.prev = currentPage;

currentPage = newNode; // Move currentPage to the new page

}

### **Go back to the previous page and move forward to the next page**

// Go back to the previous page

public void moveBack() {

if (currentPage.prev != null) {

currentPage = currentPage.prev;

System.out.println("Going back to: " + currentPage.url);

} else {

System.out.println("No previous page.");

}

}

// Go forward to the next page

public void moveForward() {

if (currentPage.next != null) {

currentPage = currentPage.next;

System.out.println("Going forward to: " + currentPage.url);

} else {

System.out.println("No forward page.");

}

}

### **Go back to the first page in the history and last visited page**

public void goToFirstPage() {

if (head != null) {

currentPage = head;

System.out.println("Going to the first page: " + currentPage.url);

} else {

System.out.println("No history available.");

}

}

public void goToLastPage() {

Node temp = head;

while (temp != null && temp.next != null) {

temp = temp.next;

}

if (temp != null) {

currentPage = temp;

System.out.println("Going to the last visited page: " + currentPage.url);

} else {

System.out.println("No history available.");

}

}

### Remove a specific page from the history

public void removePage(String url) {

Node temp = head;

while (temp != null) {

if (temp.url.equals(url)) {

if (temp.prev != null) {

temp.prev.next = temp.next;

}

if (temp.next != null) {

temp.next.prev = temp.prev;

}

if (temp == head) {

head = temp.next; // Move head if the first page is removed

}

System.out.println("Removed page: " + url);

return;

}

temp = temp.next;

}

System.out.println("Page not found in history.");

}

### **Clear History**

public void clearHistory() {

head = null;

currentPage = null;

System.out.println("All history cleared.");

}

### Get the current page URL

public void getCurrentPage() {

if (currentPage != null) {

System.out.println("Current page: " + currentPage.url);

} else {

System.out.println("No current page.");

}

}

## **Session 2: Linked List Built-In Functions**

**2.1** Create a Doubly Linked List

import java.util.LinkedList;

LinkedList<String> list = new LinkedList<>();

2.2 Add Elements

list.add("A"); // Add at the end

list.addFirst("Start"); // Add at the beginning

list.addLast("End"); // Add at the end (same as add())

list.add(1, "Middle"); // Add at specific index

2.3 Remove Elements

list.remove("A"); // Remove by value

list.remove(2); // Remove by index

list.removeFirst(); // Remove first element

list.removeLast(); // Remove last element

### 2.4Check for an Element

if (list.contains("Middle")) {

System.out.println("Element found!");

}

### 2.5 Traverse the List

for (String item : list) {

System.out.println(item);

}

2.6 Get or Set an Element

String val = list.get(1); // Get element at index 1

list.set(1, "Updated"); // Set element at index 1

## **Session 3: Complexity Analysis**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Method** | **Time Complexity** |
| Add at the end | add(element) | O(1) |
| Add at a specific index | add(index, element) | O(n) |
| Remove an element by value | remove(element) | O(n) |
| Remove at a specific index | remove(index) | O(n) |
| Check if list contains element | contains(element) | O(n) |
| Get element at a specific index | get(index) | O(n) |
| Set element at a specific index | set(index, element) | O(n) |
| Traverse the list | for-each loop or iterator() | O(n) |
| Add at the beginning | addFirst(element) | O(1) |
| Remove from the beginning | removeFirst() | O(1) |
| Remove from the end | removeLast() | O(1) |
| Peek first element | getFirst() | O(1) |
| Peek last element | getLast() | O(1) |

## **Session 4: Scenario-Based Case Studies**

### **Case Study 1: Game Level Navigation System**

**Scenario:**

In a game with multiple levels, each level is represented as a node in a doubly linked list. Players can move back and forth between levels, replay previous levels, or skip ahead.

**Tasks:**

1. Add a new level to the end.
2. Move to the next or previous level.
3. Remove a level from the sequence (e.g., deprecated or bonus level).
4. Search for a level by name.
5. Insert a new bonus level after a specific level.
6. Display levels from latest to earliest.

## **Case Study 2: Shopping Cart with Undo/Redo Feature**

**Scenario:**

A shopping app uses a doubly linked list to implement an undo/redo system for the user's cart. Each action (add/remove item) is stored, and users can undo or redo changes.

**Tasks:**

1. Add a product to the cart (append to the list).
2. Undo the last action (move back in the list).
3. Redo the undone action (move forward in the list).
4. Remove a specific product from the list.
5. Display current cart contents.
6. Search for a product in the cart history.

### **Conclusion**

In this lab, students explored the fundamental operations of doubly linked lists, including insertion, deletion, traversal, and searching. They gained hands-on experience in implementing bidirectional navigation with doubly linked lists. Through various real-world case studies, students also analyzed the practical advantages of these data structures in scenarios like media players, navigation systems, and task managers, while developing an understanding of their time complexities and memory management trade-offs.