# LinkedList Operations in Java - Lab 06

Course: CS 201 - Object-Oriented Programming

Test Report and Submission

Author: Suproteek Banerjee

Date: September 22, 2025

## Introduction

This report demonstrates the implementation of a menu-driven console application using Java's `LinkedList` class. The program provides the user with a simple interface to perform common operations such as adding, removing, searching, displaying elements, and checking the size of the list. The goal of this exercise is to practice working with Java collections, user input handling, and exception management.

## Problem Statement

We are required to design a text-based menu that allows the user to interact with a `LinkedList`. The program should:

- Provide at least five operations from the `LinkedList` class.
- Validate user input and handle exceptions gracefully.
- Allow continuous interaction until the user chooses to exit.

## Source Code

Below is the complete Java program implementing the required functionality:

```java
// Import necessary libraries
import java.util.LinkedList;
import java.util.Scanner;

public class LinkedListOperations {
    public static void main(String[] args) {
        // Create a LinkedList of Strings
        LinkedList<String> list = new LinkedList<>();
        Scanner scanner = new Scanner(System.in);
        int choice;

        // Menu loop runs until user chooses EXIT
        do {
            // Display menu
            System.out.println("\n===== LINKEDLIST MENU =====");
            System.out.println("1. Add an element");
            System.out.println("2. Remove an element");
            System.out.println("3. Search for an element");
            System.out.println("4. Display all elements");
            System.out.println("5. Get size of list");
            System.out.println("6. Check if list is empty");
            System.out.println("7. EXIT");
            System.out.print("Enter your choice: ");

            // Validate input
            while (!scanner.hasNextInt()) {
                System.out.println("Invalid input! Please enter a number.");
                scanner.next(); // discard invalid input
                System.out.print("Enter your choice: ");
            }
            choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            try {
                switch (choice) {
                    case 1: // Add
                        System.out.print("Enter element to add: ");
                        String element = scanner.nextLine();
                        list.add(element);
                        System.out.println("Element added successfully!");
                        break;

                    case 2: // Remove
                        if (list.isEmpty()) {
                            System.out.println("List is empty. Cannot remove."
                               );
                        } else {
                            System.out.print("Enter element to remove: ");
                            String remElement = scanner.nextLine();
                            if (list.remove(remElement)) {
                                System.out.println("Element removed
                                   successfully!");
                            } else {
                                System.out.println("Element not found in list.
                                   ");
                            }
                        }
                        break;

                    case 3: // Search
                        if (list.isEmpty()) {
                            System.out.println("List is empty. Nothing to
                               search.");
                        } else {
                            System.out.print("Enter element to search: ");
                            String searchElement = scanner.nextLine();
                            if (list.contains(searchElement)) {
                                System.out.println("Element found in the list!
                                   ");
                            } else {
                                System.out.println("Element not found.");
                            }
                        }
                        break;

                    case 4: // Display
                        System.out.println("Current List: " + list);
                        break;

                    case 5: // Size
                        System.out.println("Size of list: " + list.size());
                        break;

                    case 6: // Empty check
                        if (list.isEmpty()) {
                            System.out.println("The list is empty.");
                        } else {
                            System.out.println("The list is not empty.");
                        }
                        break;

                    case 7: // Exit
                        System.out.println("Exiting program...");
                        break;

                    default:
                        System.out.println("Invalid choice! Please try again."
                            );
                }
            } catch (Exception e) {
                // General error handling
                System.out.println("An error occurred: " + e.getMessage());
            }
        } while (choice != 7);

        scanner.close(); // Close scanner resource
    }
}
```

Listing 1: LinkedListOperations.java

## Sample Run

A sample execution of the program is shown below:

```
===== LINKEDLIST MENU =====
1. Add an element
2. Remove an element
3. Search for an element
4. Display all elements
5. Get size of list
6. Check if list is empty
7. EXIT
Enter your choice: 1
Enter element to add: Apple
Element added successfully!

Enter your choice: 4
Current List: [Apple]
```

```
Enter your choice: 3
Enter element to search: Banana
Element not found.
```

## Conclusion

This project helped demonstrate how Java's `LinkedList` class can be used in a real-world style interactive console program. The menu-driven approach ensures clarity for the user, while exception handling and validation make the program robust. The implemented solution meets the requirements of the assignment.