Day 22: Singly Linked List

Abhinav Yadav

"A singly linked list is a powerful tool, one node at a time."
— Anonymous

---

1 Introduction

A **Singly Linked List** is a linear data structure in which elements are stored in nodes. Each node contains two parts:

- **Data:** Stores the actual value.
- **Next:** Points to the next node in the sequence (or NULL if it's the last node).

It is used in scenarios where elements need to be dynamically added or removed. Unlike arrays, the size of a linked list can grow or shrink at runtime, which makes it a highly efficient choice for dynamic memory allocation.

2 Operations on Singly Linked List

The most common operations performed on a singly linked list are:

- **Insertion:** Insert an element at the beginning, end, or at a specific position.
- **Deletion:** Remove an element from the beginning, end, or at a specific position.
- **Traversal:** Visit and print the elements of the list.

3 Code

```
import java.util.Scanner;

// Define a Node class
class Node {
    int data;
    Node next;

    // Constructor to initialize the node
    Node(int data) {
```

```
        this.data = data;
        this.next = null;
    }
}

public class LinkedList {
    // Insert a node at the beginning
    public static Node insertAtBeginning(Node head, int value) {
        // Create a new node
        Node newNode = new Node(value);
        // Link the new node to the previous first node
        newNode.next = head;
        // Return the new head (new node)
        return newNode;
    }

    // Delete a node from the beginning
    public static Node deleteFromBeginning(Node head) {
        if (head == null) {
            System.out.println("List is empty.");
            return head;
        }

        // Move head to the next node
        Node temp = head;
        head = head.next;
        System.out.println("Deleted node from the beginning: " +
           temp.data);

        // Return the updated head
        return head;
    }

    // Print the list
    public static void printList(Node head) {
        if (head == null) {
            System.out.println("List is empty.");
            return;
        }

        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("NULL");
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
```

```
        Node head = null; // Initialize an empty list (head is
            null)

        // Insert nodes at the beginning
        System.out.print("Enter the value to insert at the
            beginning: ");
        int value = sc.nextInt();
        head = insertAtBeginning(head, value);

        System.out.print("Enter the value to insert at the
            beginning: ");
        value = sc.nextInt();
        head = insertAtBeginning(head, value);

        // Print the list after insertion
        System.out.println("List after insertion: ");
        printList(head);

        // Ask user if they want to delete from the beginning
        System.out.print("Do you want to delete a node from the
            beginning? (1 for Yes, 0 for No): ");
        int choice = sc.nextInt();
        if (choice == 1) {
            head = deleteFromBeginning(head);
        }

        // Print the list after deletion
        System.out.println("List after deletion: ");
        printList(head);

        sc.close();
    }
}
```

```
PS E:\25 days DSA\Day22>  & 'C:\Program Files\Java\jdk-20\bin\java.exe' '-XX:+ShowCo
Code\User\workspaceStorage\55d839f19c4fcf4b35227b1b13f04f45\redhat.java\jdt_ws\Day22_
Enter the value to insert at the beginning: 5
Enter the value to insert at the beginning: 3
List after insertion:
3 -> 5 -> NULL
Do you want to delete a node from the beginning? (1 for Yes, 0 for No): 1
Deleted node from the beginning: 3
List after deletion:
5 -> NULL
PS E:\25 days DSA\Day22>
```

Figure 1: Output

4 Conclusion

The singly linked list is a versatile and efficient data structure for dynamically managing data. It allows for flexible memory allocation and easy insertion and deletion operations. The operations demonstrated here, such as insertion and deletion at the beginning, are fundamental in many applications, such as implementing queues, stacks, and various algorithmic problems.