**Task Management System**

**Step 1: Understand Linked Lists**

Types of Linked Lists:

1. Singly Linked List:
   * Each node stores data and a reference to the next node.
   * It can only be traversed in one direction (from head to tail).
2. Doubly Linked List:
   * Each node stores data, a reference to the next node, and a reference to the previous node.
   * It can be traversed both forward and backward.

**Step 2: Setup**

class Task {

int taskId;

String taskName;

String status;

Task(int taskId, String taskName, String status) {

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

void display() {

System.out.println("ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

**Step 3: Implementation**

class Node {

    Task data;

    Node next;

    Node(Task data) {

        this.data = data;

        this.next = null;

    }

}

public class TaskManagement {

    Node head = null;

    public void addTask(Task task) {

        Node newNode = new Node(task);

        if (head == null) {

            head = newNode;

        } else {

            Node temp = head;

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = newNode;

        }

    }

    public void searchTask(int id) {

        Node temp = head;

        while (temp != null) {

            if (temp.data.taskId == id) {

                temp.data.display();

                return;

            }

            temp = temp.next;

        }

        System.out.println("Task is not found.");

    }

    public void displayTasks() {

        Node temp = head;

        while (temp != null) {

            temp.data.display();

            temp = temp.next;

        }

    }

    public void deleteTask(int id) {

        if (head == null) {

            System.out.println("There are no tasks to delete.");

            return;

        }

        if (head.data.taskId == id) {

            head = head.next;

            System.out.println("Task is deleted.");

            return;

        }

        Node prev = head;

        Node curr = head.next;

        while (curr != null) {

            if (curr.data.taskId == id) {

                prev.next = curr.next;

                System.out.println("Task is deleted.");

                return;

            }

            prev = curr;

            curr = curr.next;

        }

        System.out.println("Task is not found.");

    }

    public static void main(String[] args) {

        TaskManagement tm = new TaskManagement();

        tm.addTask(new Task(101, "Frontend Design", "Completed"));

        tm.addTask(new Task(102, "Backend Connectivity", "In Progress"));

        tm.addTask(new Task(103, "Testing", "Pending"));

        tm.displayTasks();

        tm.searchTask(102);

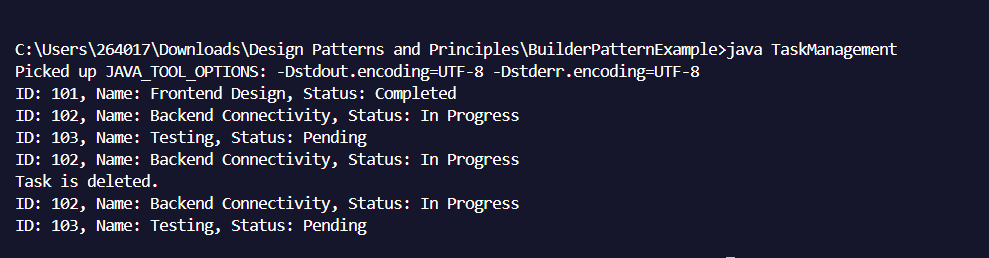
        tm.deleteTask(101);

        tm.displayTasks();

    }

}

**Output:**

****

**Step 4: Analysis**

| Operation | Time Complexity | Explanation |
| --- | --- | --- |
| Add | O(n) | Traverse till end and insert. |
| Search | O(n) | Check each node until its found. |
| Traverse | O(n) | Traverse all nodes one by one. |
| Delete | O(n) | Search and delete the node. |

**Advantages Of Linked Lists Over Arrays**

| Feature | Arrays | Linked Lists |
| --- | --- | --- |
| Size flexibility | Fixed size | Dynamic size |
| Insertion/Deletion | Costly | Efficient |
| Memory Usage | Might waste memory | Uses memory as needed |