**Module 2 - Data Structures and Algorithms**

**Exercise 5: Task Management System**

1. **Understand Linked Lists**

**➤ Singly Linked List:**

* Each node points to the next node.
* Traversal is one direction only.
* Lightweight, but no reverse traversal.

**➤ Doubly Linked List:**

* Each node has two references: next and prev.
* Can be traversed both ways.
* Requires more memory (extra pointer).

For this exercise, we focus on Singly Linked List.

**2,3). Setup and Implementation:**

**Code:**

class Task {

    int taskId;

    String taskName;

    String status;

    Task next;

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

        this.taskId = taskId;

        this.taskName = taskName;

        this.status = status;

        this.next = null;

    }

}

public class TaskManager {

    Task head = null;

    public void addTask(int id, String name, String status) {

        Task newTask = new Task(id, name, status);

        if (head == null) {

            head = newTask;

        } else {

            Task temp = head;

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = newTask;

        }

    }

    public void searchTask(int id) {

        Task temp = head;

        while (temp != null) {

            if (temp.taskId == id) {

                System.out.println(temp.taskId + " " + temp.taskName + " " + temp.status);

                return;

            }

            temp = temp.next;

        }

    }

    public void traverseTasks() {

        Task temp = head;

        while (temp != null) {

            System.out.println(temp.taskId + " " + temp.taskName + " " + temp.status);

            temp = temp.next;

        }

    }

    public void deleteTask(int id) {

        if (head == null) return;

        if (head.taskId == id) {

            head = head.next;

            return;

        }

        Task temp = head;

        while (temp.next != null) {

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

                temp.next = temp.next.next;

                return;

            }

            temp = temp.next;

        }

    }

    public static void main(String[] args) {

        TaskManager tm = new TaskManager();

        tm.addTask(1, "Design", "Pending");

        tm.addTask(2, "Code", "In Progress");

        tm.traverseTasks();

        System.out.println();

        tm.searchTask(2);

        tm.deleteTask(1);

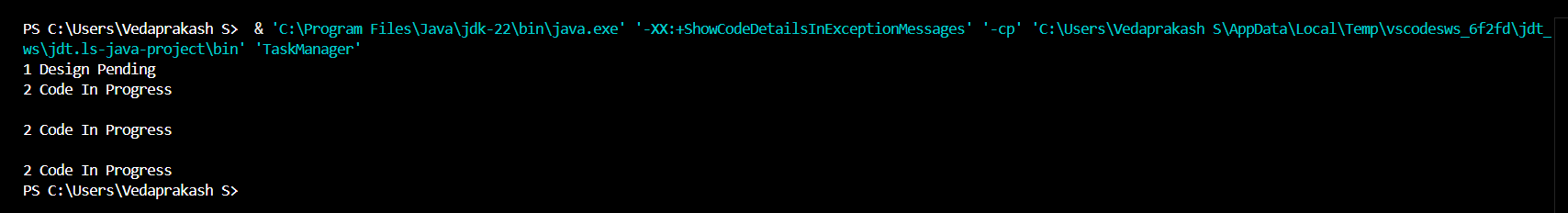
        System.out.println();

        tm.traverseTasks();

    }

}

**Output:**



**4). Analysis**

* **Add**: O(1) at head, O(n) at tail or position.
* **Search**: O(n)
* **Traverse**: O(n)
* **Delete**: O(1) if pointer to node given; otherwise O(n)

**Advantages over Arrays**

* Dynamic size
* Easier insertion/deletion
* No need to shift elements