**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

**Steps:**

1. **Understand Linked Lists:**
   * Explain the different types of linked lists (Singly Linked List, Doubly Linked List).
2. **Setup:**
   * Create a class **Task** with attributes like **taskId**, **taskName**, and **status**.
3. **Implementation:**
   * Implement a singly linked list to manage tasks.
   * Implement methods to **add**, **search**, **traverse**, and **delete** tasks in the linked list.
4. **Analysis:**
   * Analyze the time complexity of each operation.
   * Discuss the advantages of linked lists over arrays for dynamic data.

**1: Understanding Linked Lists**

**Types of Linked Lists:**

1. **Singly Linked List:**
   * Each node points to the next node.
   * Traversal is one-directional (from head to tail).
   * Uses less memory than doubly linked list.
2. **Doubly Linked List:**
   * Each node has two pointers: next and prev.
   * Traversal is possible in both directions.
   * Requires more memory and management.

**Code:-**

import java.util.Scanner;

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 void display() {

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

    }

}

public class TaskManagementSystem {

    private Task head = null;

    // Add task at end

    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;

        }

        System.out.println("Task added.");

    }

    // Search task by ID

    public Task searchTask(int id) {

        Task temp = head;

        while (temp != null) {

            if (temp.taskId == id) {

                return temp;

            }

            temp = temp.next;

        }

        return null;

    }

    // Traverse and display all tasks

    public void traverseTasks() {

        if (head == null) {

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

            return;

        }

        Task temp = head;

        while (temp != null) {

            temp.display();

            temp = temp.next;

        }

    }

    // Delete task by ID

    public void deleteTask(int id) {

        if (head == null) {

            System.out.println("No tasks to delete.");

            return;

        }

        if (head.taskId == id) {

            head = head.next;

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

            return;

        }

        Task prev = head;

        Task curr = head.next;

        while (curr != null) {

            if (curr.taskId == id) {

                prev.next = curr.next;

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

                return;

            }

            prev = curr;

            curr = curr.next;

        }

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

    }

    // Main method to test functionality

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        TaskManagementSystem system = new TaskManagementSystem();

        while (true) {

            System.out.println("\n1. Add Task\n2. Search Task\n3. View All Tasks\n4. Delete Task\n5. Exit");

            System.out.print("Choose an option: ");

            int choice = sc.nextInt();

            switch (choice) {

                case 1:

                    System.out.print("Enter Task ID: ");

                    int id = sc.nextInt();

                    sc.nextLine();

                    System.out.print("Enter Task Name: ");

                    String name = sc.nextLine();

                    System.out.print("Enter Status: ");

                    String status = sc.nextLine();

                    system.addTask(id, name, status);

                    break;

                case 2:

                    System.out.print("Enter Task ID to search: ");

                    int searchId = sc.nextInt();

                    Task found = system.searchTask(searchId);

                    if (found != null) {

                        found.display();

                    } else {

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

                    }

                    break;

                case 3:

                    system.traverseTasks();

                    break;

                case 4:

                    System.out.print("Enter Task ID to delete: ");

                    int delId = sc.nextInt();

                    system.deleteTask(delId);

                    break;

                case 5:

                    System.out.println("Exiting...");

                    sc.close();

                    return;

                default:

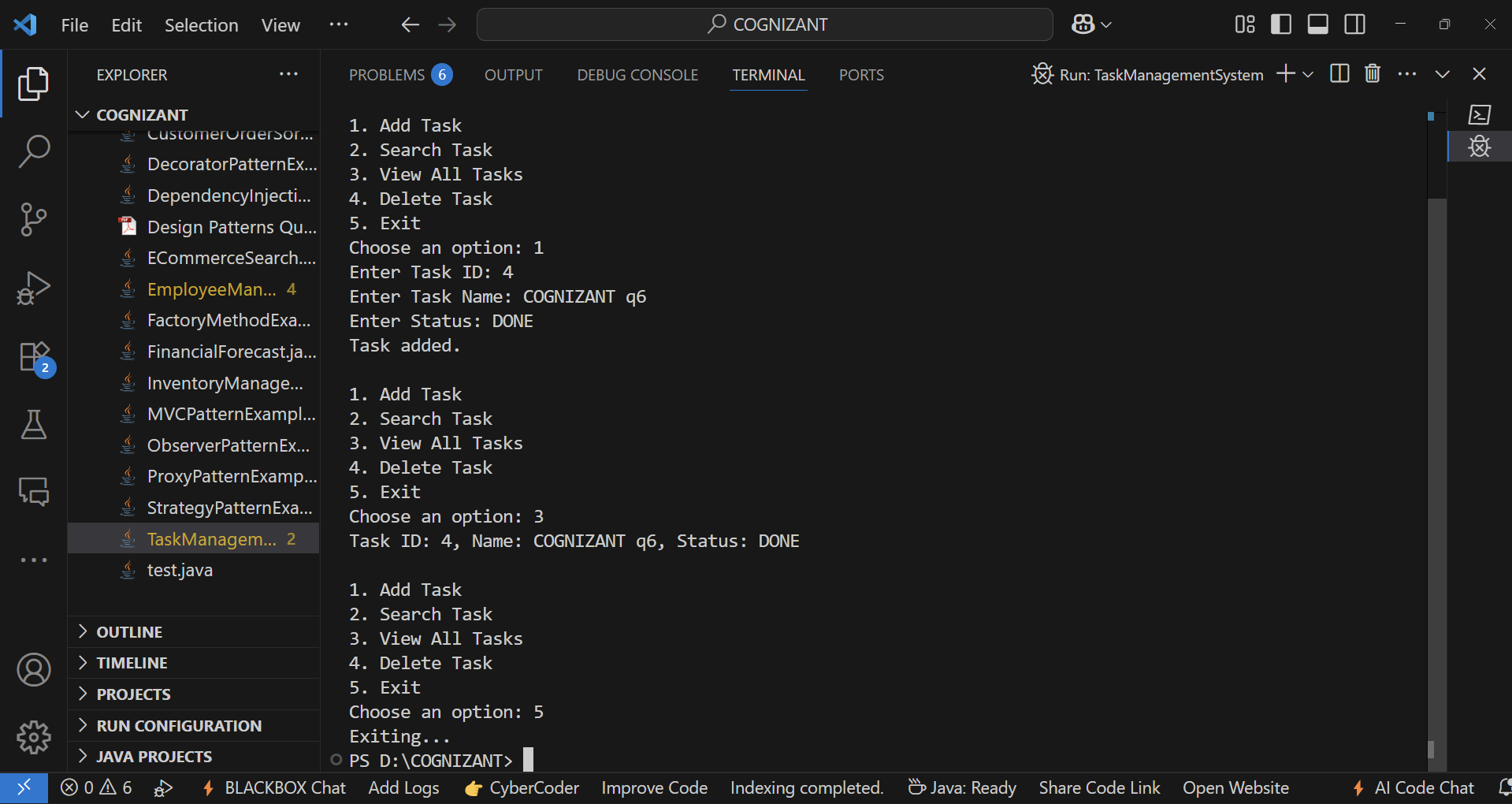
                    System.out.println("Invalid option.");

            }

        }

    }

}

**OUTPUT:-**

**4: Time Complexity Analysis**

| **Operation** | **Time Complexity** |
| --- | --- |
| Add Task | O(n) |
| Search Task | O(n) |
| Traverse Task | O(n) |
| Delete Task | O(n) |

n is the number of tasks.

**Advantages of Linked Lists over Arrays**

* **Dynamic size**: Can grow or shrink during execution without memory reallocation.
* **Efficient Insert/Delete**: No shifting required like in arrays.
* **Memory Usage**: Only memory for used elements is allocated (no fixed size needed).