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.

→ Step 1: Understand Linked Lists

Types of Linked Lists:

1. **Singly Linked List**
   - Each node points to the next node only.
   - Memory efficient and simple to implement.
2. **Doubly Linked List**
   - Each node has two pointers: one to the next node and one to the previous node.
   - Allows traversal in both directions but uses more memory.

Why Use Linked Lists?

- **Dynamic size**: Grows/shrinks as needed.
- **Efficient insertions/deletions** at the beginning or middle (no shifting).
- Ideal for applications like task managers where tasks can be frequently added or removed.

Step 2: Setup Task Class

```
public class Task {
  int taskId;
  String taskName;
```

```
    String status;
    public Task(int taskId, String taskName, String status) {
        this.taskId = taskId;
        this.taskName = taskName;
        this.status = status;
    }
    public String toString() {
        return "Task[ID=" + taskId + ", Name=" + taskName + ", Status=" + status + "]";
    }
}
```

Step 3: Implementation with Singly Linked List

Node Class

```
class Node {
    Task task;
    Node next;
    public Node(Task task) {
        this.task = task;
        this.next = null;
    }
}
```

Task Manager using Linked List

```
public class TaskManager {
    private Node head;
    // Add a task at the end
    public void addTask(Task task) {
        Node newNode = new Node(task);
        if (head == null) {
            head = newNode;
        } else {
            Node current = head;
            while (current.next != null)
```

```
        current = current.next;
      current.next = newNode;
    }
  }
  // Search a task by ID
  public Task searchTask(int taskId) {
    Node current = head;
    while (current != null) {
      if (current.task.taskId == taskId)
        return current.task;
      current = current.next;
    }
    return null;
  }
  // Traverse and display tasks
  public void displayTasks() {
    Node current = head;
    while (current != null) {
      System.out.println(current.task);
      current = current.next;
    }
  }
  // Delete a task by ID
  public void deleteTask(int taskId) {
    if (head == null) return;
    if (head.task.taskId == taskId) {
      head = head.next;
      return;
    }
    Node current = head;
    while (current.next != null && current.next.task.taskId != taskId)
```

```
            current = current.next;
        if (current.next != null)
            current.next = current.next.next;
    }
}
```

✅ **Main Method to Test**

```
public class Main {
    public static void main(String[] args) {
        TaskManager manager = new TaskManager();
        manager.addTask(new Task(1, "Design UI", "Pending"));
        manager.addTask(new Task(2, "Implement Backend", "In Progress"));
        manager.addTask(new Task(3, "Testing", "Pending"));
        System.out.println("All Tasks:");
        manager.displayTasks();
        System.out.println("\nSearch Task ID 2:");
        System.out.println(manager.searchTask(2));
        System.out.println("\nDelete Task ID 2:");
        manager.deleteTask(2);
        System.out.println("\nAll Tasks After Deletion:");
        manager.displayTasks();
    }
}
```

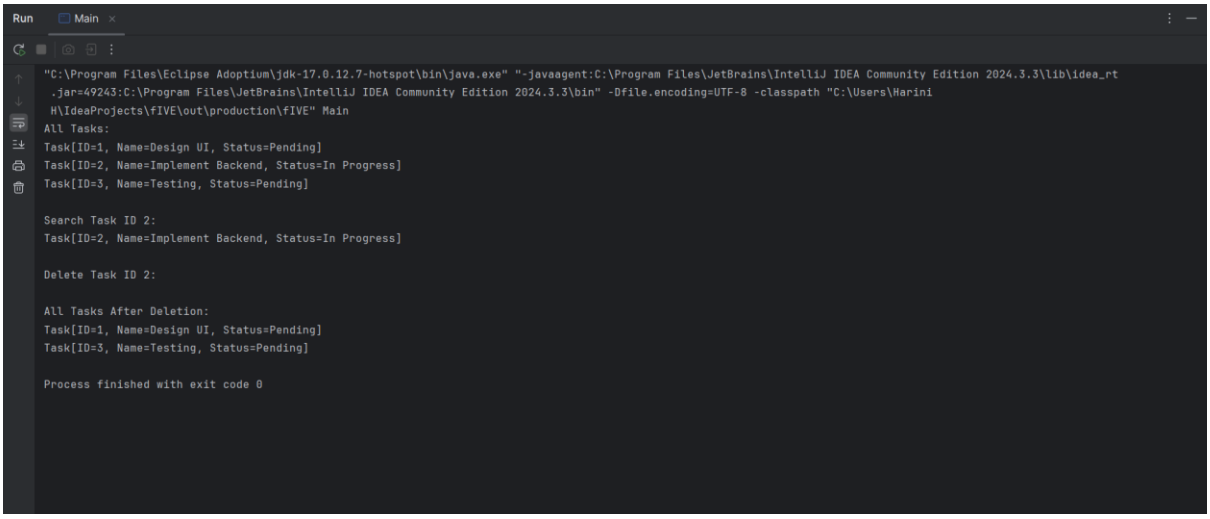
Step 4: Time Complexity Analysis

Operation	Time Complexity	Explanation
Add	O(n)	Traverse to end to add
Search	O(n)	Traverse through all nodes
Traverse	O(n)	Visit each node
Delete	O(n)	Need to find the node before deletion

Linked Lists vs Arrays

Feature	Arrays	Linked Lists
Size	Fixed	Dynamic
Insertion/Deletion	O(n) (shifting)	O(1) at head/tail
Search	O(n)	O(n)
Random Access	O(1)	O(n)
Memory	Compact	More (pointers used)

OUTPUT:




```
"C:\Program Files\Eclipse Adoptium\jdk-17.0.12.7-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.3\lib\idea_rt
 .jar=49243:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.3.3\bin" -Dfile.encoding=UTF-8 -classpath "C:\Users\Harini
 H\IdeaProjects\fIVE\out\production\fIVE" Main
All Tasks:
Task[ID=1, Name=Design UI, Status=Pending]
Task[ID=2, Name=Implement Backend, Status=In Progress]
Task[ID=3, Name=Testing, Status=Pending]

Search Task ID 2:
Task[ID=2, Name=Implement Backend, Status=In Progress]

Delete Task ID 2:

All Tasks After Deletion:
Task[ID=1, Name=Design UI, Status=Pending]
Task[ID=3, Name=Testing, Status=Pending]

Process finished with exit code 0
```