Exercise 5: Task Management System

**1. Understand Linked Lists**

**🔹 Singly Linked List:**

* **Each node contains:**
  + **data**
  + **a reference to the next node.**
* **Traverses in only one direction (forward).**

**🔹 Doubly Linked List:**

* **Each node has:**
  + **data**
  + **a reference to the next node**
  + **a reference to the previous node.**
* **Can traverse both forward and backward.**
* **Uses more memory than singly linked list.**

**For this exercise, we’re using a Singly Linked List.**

**2. Setup**

Create a class Task with:

* taskId (int)
* taskName (String)
* status (String)

**3. Implementation**

* Task.java

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;

}

@Override

public String toString() {

return taskId + " - " + taskName + " - " + status;

}

}

* TaskNode.java:

public class TaskNode {

Task data;

TaskNode next;

public TaskNode(Task data) {

this.data = data;

this.next = null;

}

}

* TaskManager.java

public class TaskManager {

TaskNode head = null;

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

} else {

TaskNode temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

public void deleteTask(int taskId) {

if (head == null) return;

if (head.data.taskId == taskId) {

head = head.next;

return;

}

TaskNode temp = head;

while (temp.next != null && temp.next.data.taskId != taskId) {

temp = temp.next;

}

if (temp.next != null) {

temp.next = temp.next.next;

}

}

public Task searchTask(int taskId) {

TaskNode temp = head;

while (temp != null) {

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

return temp.data;

}

temp = temp.next;

}

return null;

}

public void traverseTasks() {

TaskNode temp = head;

while (temp != null) {

System.out.println(temp.data);

temp = temp.next;

}

}

}

* TaskTest.java

public class TaskTest {

public static void main(String[] args) {

TaskManager manager = new TaskManager();

manager.addTask(new Task(1, "Write report", "Pending"));

manager.addTask(new Task(2, "Fix bug", "In Progress"));

manager.addTask(new Task(3, "Team meeting", "Completed"));

System.out.println("All Tasks:");

manager.traverseTasks();

System.out.println("\nSearch Task with ID 2:");

System.out.println(manager.searchTask(2));

System.out.println("\nDeleting Task with ID 1");

manager.deleteTask(1);

System.out.println("\nAll Tasks after Deletion:");

manager.traverseTasks();

}

}

4. Analysis

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

**Advantages of Linked Lists over Arrays**

| **Feature** | **Arrays** | **Linked Lists** |
| --- | --- | --- |
| Size | Fixed | Dynamic |
| Insert/Delete | Costly (O(n)) | Efficient (O(1)/O(n)) |
| Memory | Contiguous | Non-contiguous |

**Linked lists are perfect when:**

* You don’t know the size beforehand.
* You do frequent insertions and deletions.

**Output :**

