**Exercise 5: Task Management System**

Scenario: Developing a task management system where tasks need to be added, deleted, and traversed efficiently.

1. Understand Linked Lists:

* **Singly Linked List:** Each node points to the next node. Efficient for insertions and deletions at the head or tail.
* **Doubly Linked List:** Nodes point both to the next and previous nodes. Allows traversal in both directions, but uses more memory.

Linked lists are dynamic and do not require resizing like arrays. They are ideal for applications with frequent insertions and deletions.

1. Setup: Create a class Task with the following attributes:

* String taskId
* String taskName
* String status

1. Implementation:

Java Code:

class Task {

String taskId;

String taskName;

String status;

Task next;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

this.next = null;

}

public String toString() {

return taskId + " | " + taskName + " | " + status;

}

}

class TaskManager {

Task head = null;

void addTask(String 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;

}

}

void searchTask(String id) {

Task temp = head;

while (temp != null) {

if (temp.taskId.equals(id)) {

System.out.println(temp);

return;

}

temp = temp.next;

}

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

}

void deleteTask(String id) {

if (head == null) return;

if (head.taskId.equals(id)) {

head = head.next;

return;

}

Task temp = head;

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

temp = temp.next;

}

if (temp.next != null) temp.next = temp.next.next;

}

void displayTasks() {

Task temp = head;

while (temp != null) {

System.out.println(temp);

temp = temp.next;

}

}

public static void main(String[] args) {

TaskManager tm = new TaskManager();

tm.addTask("T001", "Design UI", "Pending");

tm.addTask("T002", "Develop Backend", "In Progress");

tm.addTask("T003", "Testing", "Pending");

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

tm.displayTasks();

System.out.println("\nSearch for T002:");

tm.searchTask("T002");

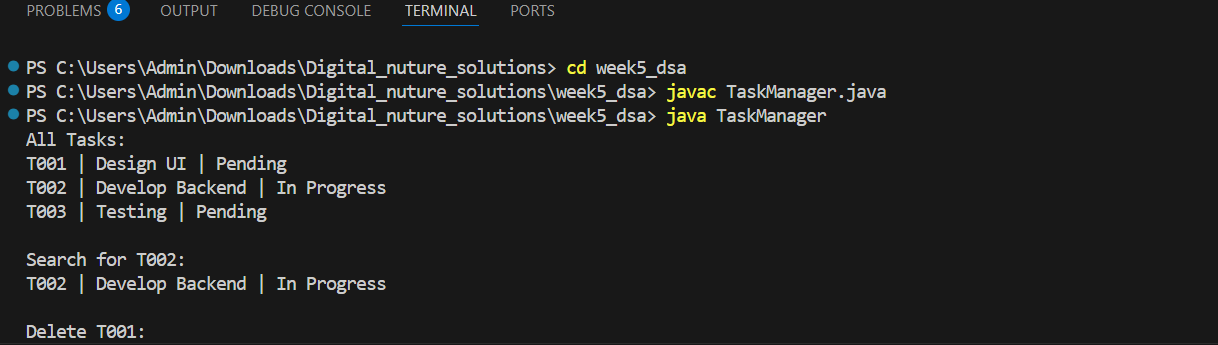
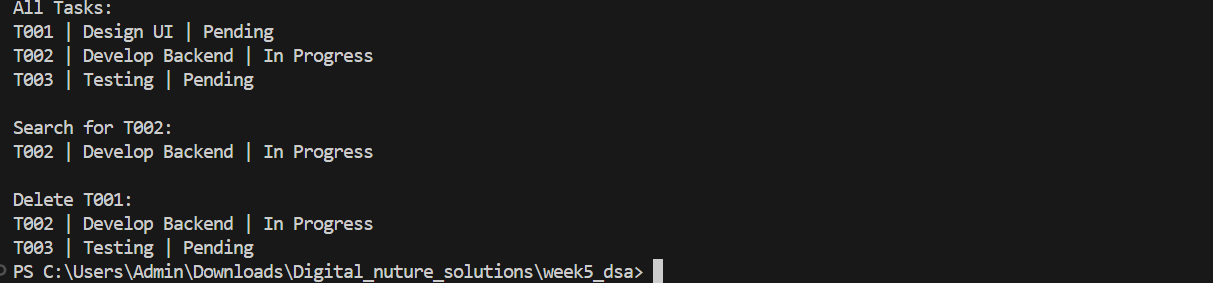
System.out.println("\nDelete T001:");

tm.deleteTask("T001");

tm.displayTasks();

}

}  
  
OUTPUT:



1. Analysis:

* **Add:** O(n) if added at the end.
* **Search:** O(n) – requires linear traversal.
* **Traverse:** O(n)
* **Delete:** O(n) – must find the node before the one to delete.

Advantages of Linked Lists:

* Dynamic memory allocation.
* Efficient insertions and deletions compared to arrays.

Use linked lists for dynamic datasets and when frequent additions/removals are required.