**Exercise 5: Task Management System**

**Understanding the Problem:**  
Tasks often need to be dynamically added and removed. Linked lists provide a flexible way to manage such dynamic data without the need to shift elements.

**Setup and Implementation:**  
A Task class is created with:

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

A singly linked list is used, with each task pointing to the next. Operations include add, delete, search, and traverse.

**Java Code:**

import java.util.\*;

class Task {

int taskId;

String taskName;

String status;

Task next;

Task(int id, String name, String stat) {

taskId = id;

taskName = name;

status = stat;

next = null;

}

}

public class TaskManagement {

static Task head = null;

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

while (true) {

System.out.println("1. Add 2. Search 3. View All 4. Delete 5. Exit");

int ch = sc.nextInt();

if (ch == 1) {

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

int id = sc.nextInt();

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

String name = sc.next();

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

String stat = sc.next();

Task t = new Task(id, name, stat);

if (head == null) head = t;

else {

Task temp = head;

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

temp.next = t;

}

} else if (ch == 2) {

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

int id = sc.nextInt();

Task temp = head;

while (temp != null) {

if (temp.taskId == id) {

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

}

temp = temp.next;

}

} else if (ch == 3) {

Task temp = head;

while (temp != null) {

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

temp = temp.next;

}

} else if (ch == 4) {

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

int id = sc.nextInt();

if (head != null && head.taskId == id) {

head = head.next;

} else {

Task prev = head;

Task temp = head.next;

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

prev = temp;

temp = temp.next;

}

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

}

} else break;

}

}

}

**Time Complexity Analysis:**

* Add (at end): O(n)
* Search: O(n)
* Traverse: O(n)
* Delete: O(n)

**Optimization Discussion:**  
Linked lists allow dynamic memory usage, avoiding the fixed-size limitation of arrays. Adding a tail pointer would make insertion O(1). Still, searching and deletion require linear time. If indexed access is required, ArrayList or HashMap could be more suitable.