Exercise 5: Task Management System

Linked List is a linear data structure where each element (node) points to the next.

Types:

Singly Linked List: Nodes have only a reference to the next node.

Doubly Linked List: Nodes have references to both next and previous nodes. Use Case: Linked lists are preferred when:

The number of elements changes frequently. Insertions/deletions are frequent and done in the middle.

Full Java 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 String toString() {

return "Task ID: " + taskId + ", Name: " + taskName + ", Status: " +

status;

}

}

public class TaskManagementSystem { static Task head = null;

static Scanner sc = new Scanner(System.in);

public static void addTask() {

System.out.print("Enter Task ID, Name, and Status: "); int id = sc.nextInt();

String name = sc.next(); String status = sc.next();

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.");

}

public static void searchTask() {

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("Found: " + temp); return;

}

temp = temp.next;

}

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

}

public static void traverseTasks() { System.out.println("All Tasks:"); Task temp = head;

while (temp != null) { System.out.println(temp); temp = temp.next;

}

}

public static void deleteTask() { System.out.print("Enter Task ID to delete: "); int id = sc.nextInt();

if (head == null) { System.out.println("List is empty."); return;

}

if (head.taskId == id) { head = head.next;

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

}

Task temp = head;

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

}

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

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

} else {

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

}

}

public static void main(String[] args) { while (true) {

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

int choice = sc.nextInt(); switch (choice) {

case 1: addTask(); break; case 2: searchTask(); break;

case 3: traverseTasks(); break; case 4: deleteTask(); break; case 5: return;

}

}

}

}

✅Analysis

Operation Time Complexity Description

Add O(n) Traverse to end of list to insert Search O(n) Traverse each node to find match Traverse O(n) Visit every node

Delete O(n) Find and remove matching task node

✅ Linked List vs Array

Feature Linked List Array

Memory Dynamic (grows/shrinks) Fixed size

Insertion/Deletion Fast (O(1) at head) Slow (O(n) with shifting) Access (by index) Slow (O(n)) Fast (O(1))

Use Case Frequent modifications Frequent direct access

✅ Sample Input

User selects options step-by-step:

1. Add Task

> Enter Task ID, Name, and Status:

1. WriteDocs Pending

1. Add Task

> Enter Task ID, Name, and Status:

1. CodeApp InProgress

1. Add Task

> Enter Task ID, Name, and Status:

1. ReviewCode Pending

3. Display All Tasks

✅ Sample Output

Task added. Task added. Task added.

All Tasks:

Task ID: 101, Name: WriteDocs, Status: Pending Task ID: 102, Name: CodeApp, Status: InProgress Task ID: 103, Name: ReviewCode, Status: Pending

✅ More Sample Actions

Searching a Task:

2. Search Task

> Enter Task ID to search: 102

Found: Task ID: 102, Name: CodeApp, Status: InProgress Deleting a Task:

4. Delete Task

> Enter Task ID to delete: 101 Task deleted.

Displaying Again After Deletion:

3. Display All Tasks

Task ID: 102, Name: CodeApp, Status: InProgress Task ID: 103, Name: ReviewCode, Status: Pending