**Data Structures Algorithms**

**EXERCISE 5: TASK MANAGEMENT SYSTEM**

**Source Code**

// Main.java

// Step 2: Task class

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 void display() {

System.out.println("ID: " + taskId + ", Name: " + taskName + ", Status: " + status);

}

}

// Node for linked list

class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

// Step 3: Singly linked list implementation

class TaskList {

private TaskNode head;

// Add task to end

public void addTask(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

} else {

TaskNode current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

System.out.println("Task added: " + task.taskName);

}

// Search by ID

public void searchTask(int taskId) {

TaskNode current = head;

while (current != null) {

if (current.task.taskId == taskId) {

System.out.println("Task found:");

current.task.display();

return;

}

current = current.next;

}

System.out.println("Task with ID " + taskId + " not found.");

}

// Traverse all tasks

public void displayAllTasks() {

if (head == null) {

System.out.println("Task list is empty.");

return;

}

System.out.println("Task List:");

TaskNode current = head;

while (current != null) {

current.task.display();

current = current.next;

}

}

// Delete task by ID

public void deleteTask(int taskId) {

if (head == null) {

System.out.println("List is empty.");

return;

}

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

head = head.next;

System.out.println("Task with ID " + taskId + " deleted.");

return;

}

TaskNode current = head;

while (current.next != null) {

if (current.next.task.taskId == taskId) {

current.next = current.next.next;

System.out.println("Task with ID " + taskId + " deleted.");

return;

}

current = current.next;

}

System.out.println("Task with ID " + taskId + " not found.");

}

}

// Step 4: Test class + Analysis

public class Main {

public static void main(String[] args) {

TaskList taskList = new TaskList();

// Add tasks

taskList.addTask(new Task(1, "Design UI", "Pending"));

taskList.addTask(new Task(2, "Develop Backend", "In Progress"));

taskList.addTask(new Task(3, "Test Application", "Pending"));

System.out.println("\n-- All Tasks --");

taskList.displayAllTasks();

System.out.println("\n-- Search Task with ID 2 --");

taskList.searchTask(2);

System.out.println("\n-- Delete Task with ID 1 --");

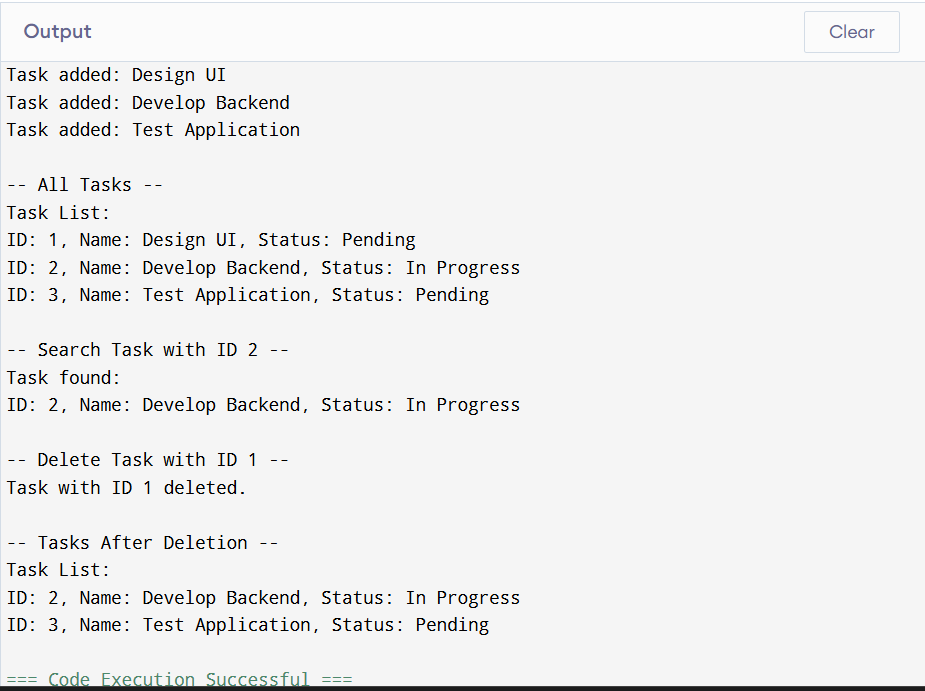
taskList.deleteTask(1);

System.out.println("\n-- Tasks After Deletion --");

taskList.displayAllTasks();

}

}

**OUTPUT**

**📚 Linked List Types1.Singly Linked List:** Each node points to the next. Simple and memory-efficient.**2.Doubly Linked List:** Nodes point to both next and previous. More flexible but needs more memory.⏱ **Time ComplexityAdd:** O(n) (at end), O(1) if inserting at head**Search:** O(n)**Delete:** O(n)**Traverse:** O(n)**🔍 Linked List vs ArrayPros:**1.Dynamic size2.Efficient insertions/deletions without shifting elements**Cons:**1.Slower access (no index-based random access)2.More memory due to pointer overhead