**Exercise 5: Task Management System**

**Scenario:**

You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

**1. Understand Linked Lists**

**Types of Linked Lists**:

* **Singly Linked List**: Each node points to the next node. It allows for efficient insertions and deletions but only supports traversal in one direction.
* **Doubly Linked List**: Each node points to both the next and previous nodes, allowing traversal in both directions and easier deletion and insertion operations.

**2. Setup:**

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 "Task ID: " + taskId + ", Name: " + taskName + ", Status: " + status;

}

}

**3. Implementation:**

public class TaskManagementSystem {

private class Node {

Task task;

Node next;

Node(Task task) {

this.task = task;

this.next = null;

}

}

private Node head;

// Add a task

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

}

// Search for a task by ID

public Task searchTask(int taskId) {

Node current = head;

while (current != null) {

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

return current.task;

}

current = current.next;

}

return null; // Task not found

}

// Traverse and print all tasks

public void traverseTasks() {

Node current = head;

while (current != null) {

System.out.println(current.task);

current = current.next;

}

}

// Delete a task by ID

public void deleteTask(int taskId) {

if (head == null) {

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

return;

}

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

head = head.next;

return;

}

Node current = head;

while (current.next != null && current.next.task.taskId != taskId) {

current = current.next;

}

if (current.next != null) {

current.next = current.next.next;

} else {

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

}

}

public static void main(String[] args) {

TaskManagementSystem tms = new TaskManagementSystem();

tms.addTask(new Task(1, "Complete Report", "Pending"));

tms.addTask(new Task(2, "Submit Assignment", "In Progress"));

tms.addTask(new Task(3, "Prepare Presentation", "Completed"));

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

tms.traverseTasks();

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

Task task = tms.searchTask(2);

System.out.println(task != null ? task : "Task not found");

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

tms.deleteTask(2);

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

tms.traverseTasks();

}

}

**4. Analysis**

**Time Complexity:**

* **Add:** O(n) (traverses to the end of the list).
* **Search**: O(n) (requires scanning through the list).
* **Traverse:** O(n) (needs to visit each node)**.**
* **Delete:** O(n) (requires finding the node and adjusting pointers).

**Advantages of Linked Lists over Arrays:**

* **Dynamic Size:** Linked lists can grow and shrink dynamically, making them more flexible for managing datasets with unpredictable sizes.
* **Efficient Insertions/Deletions:** Insertions and deletions are more efficient compared to arrays since they involve pointer adjustments rather than shifting elements.