**Exercise 5: Task Management System**

**Scenario:**

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

**Steps:**

1. **Understand Linked Lists:**
   * Explain the different types of linked lists (Singly Linked List, Doubly Linked List).
2. **Setup:**
   * Create a class **Task** with attributes like **taskId**, **taskName**, and **status**.
3. **Implementation:**
   * Implement a singly linked list to manage tasks.
   * Implement methods to **add**, **search**, **traverse**, and **delete** tasks in the linked list.
4. **Analysis:**
   * Analyze the time complexity of each operation.
   * Discuss the advantages of linked lists over arrays for dynamic data.

class Task {

int taskId;

String taskName;

String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

}

class Node {

Task data;

Node next;

Node(Task data) {

this.data = data;

this.next = null;

}

}

public class TaskManagementSystem {

private Node head;

public void addTask(Task task) {

Node newNode = new Node(task);

newNode.next = head;

head = newNode;

}

public Task searchTaskById(int taskId) {

Node current = head;

while (current != null) {

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

return current.data;

}

current = current.next;

}

return null;

}

public void displayTasks() {

Node current = head;

while (current != null) {

System.***out***.println(current.data.taskId + ", " + current.data.taskName + ", " + current.data.status);

current = current.next;

}

}

public void deleteTask(int taskId) {

if (head == null) {

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

return;

}

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

head = head.next;

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

return;

}

Node current = head;

while (current.next != null) {

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

current.next = current.next.next;

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

return;

}

current = current.next;

}

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

}

public static void main(String[] args) {

TaskManagementSystem system = new TaskManagementSystem();

system.addTask(new Task(1, "Task 1", "To do"));

system.addTask(new Task(2, "Task 2", "In progress"));

system.addTask(new Task(3, "Task 3", "Done"));

system.displayTasks();

Task foundTask = system.searchTaskById(2);

if (foundTask != null) {

System.***out***.println("Found task: " + foundTask.taskName);

}

system.deleteTask(2);

system.displayTasks();

}

}

OUTPUT:

3, Task 3, Done

2, Task 2, In progress

1, Task 1, To do

Found task: Task 2

Task deleted

3, Task 3, Done

1, Task 1, To do

1) The different types of linked lists (Singly Linked List, Doubly Linked List).

A linked list is a linear data structure where elements are not stored in contiguous memory locations. Instead, each element (node) contains data and a reference (link) to the next node in the sequence.

Types of Linked Lists

Singly Linked List: Each node has a reference to the next node.

Doubly Linked List: Each node has references to both the previous and next nodes.

2)Analysis

**Time complexity:**

addTask: O(1) as it adds a new node at the beginning.

searchTaskById: O(n) as we need to traverse the list in the worst case.

displayTasks: O(n) as we iterate through all nodes.

deleteTask: O(n) in the worst case when the task to be deleted is at the end.

**3)Advantages of linked lists over arrays for dynamic data:**

Dynamic size: Linked lists can grow and shrink as needed.

Efficient insertions and deletions: Elements can be inserted and deleted in O(1) time by modifying pointers.

No wasted space: Linked lists only allocate memory for nodes that are actually used.

However, linked lists have the disadvantage of not providing random access to elements, unlike arrays.