**Exercise – 5 Task Management System**

**Linked Lists:**

Linked Lists are sequential Data Structures, they are dynamic data structures used to store a sequence of elements. Unlike arrays, they do not require contiguous memory and allow efficient insertion and deletion of elements. There are several types of linked lists, each suited for different use cases:

**Singly Linked Lists:**

A Singly Linked List is a linear data structure where each node contains:

1. Data (value)

2. A pointer/reference to the next node

It can only be traversed in one direction, they use less memory per node than doubly linked lists. The insertion and deletion are efficient at the beginning.

**Doubly Linked List:**

A Doubly Linked List is a more advanced structure where each node contains:

1. Data (value)

2. A pointer to the next node

3. A pointer to the previous node

It can be traversed in both directions, easier to delete a node when give only a pointer to it.

More flexible to complex operations like bidirectional traversal, etc. It uses more memory and it is slightly more complex to implement and maintain.

**Task.java**

class Task {

int taskId;

String taskName;

String status;

Task next;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

this.next = null;

}

}

**Main.java**

import java.util.\*;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Task head = null;

boolean exit = false;

while (!exit) {

System.out.println("\nTask Management System");

System.out.println("1. Add Task");

System.out.println("2. Search Task");

System.out.println("3. Traverse Tasks");

System.out.println("4. Delete Task");

System.out.println("5. Exit");

System.out.print("Enter your choice: ");

int choice = sc.nextInt();

switch (choice) {

case 1:

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

int id = sc.nextInt();

sc.nextLine();

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

String name = sc.nextLine();

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

String status = sc.nextLine();

head = addTask(head, new Task(id, name, status));

break;

case 2:

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

int searchId = sc.nextInt();

Task found = searchTask(head, searchId);

if (found != null)

System.out.println("Task Found: " + found.taskId + " - " + found.taskName + " (" + found.status + ")");

else

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

break;

case 3:

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

traverseTasks(head);

break;

case 4:

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

int deleteId = sc.nextInt();

head = deleteTask(head, deleteId);

break;

case 5:

System.out.println("Exiting...");

exit = true;

break;

default:

System.out.println("Invalid choice!");

}

}

sc.close();

}

static Task addTask(Task head, Task newTask) {

if (head == null) {

head = newTask;

} else {

Task temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newTask;

}

System.out.println("Task added successfully.");

return head;

}

static Task searchTask(Task head, int id) {

Task temp = head;

while (temp != null) {

if (temp.taskId == id)

return temp;

temp = temp.next;

}

return null;

}

static void traverseTasks(Task head) {

if (head == null) {

System.out.println("No tasks to display.");

return;

}

Task temp = head;

while (temp != null) {

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

temp = temp.next;

}

}

static Task deleteTask(Task head, int id) {

if (head == null) {

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

return null;

}

if (head.taskId == id) {

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

return head.next;

}

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.");

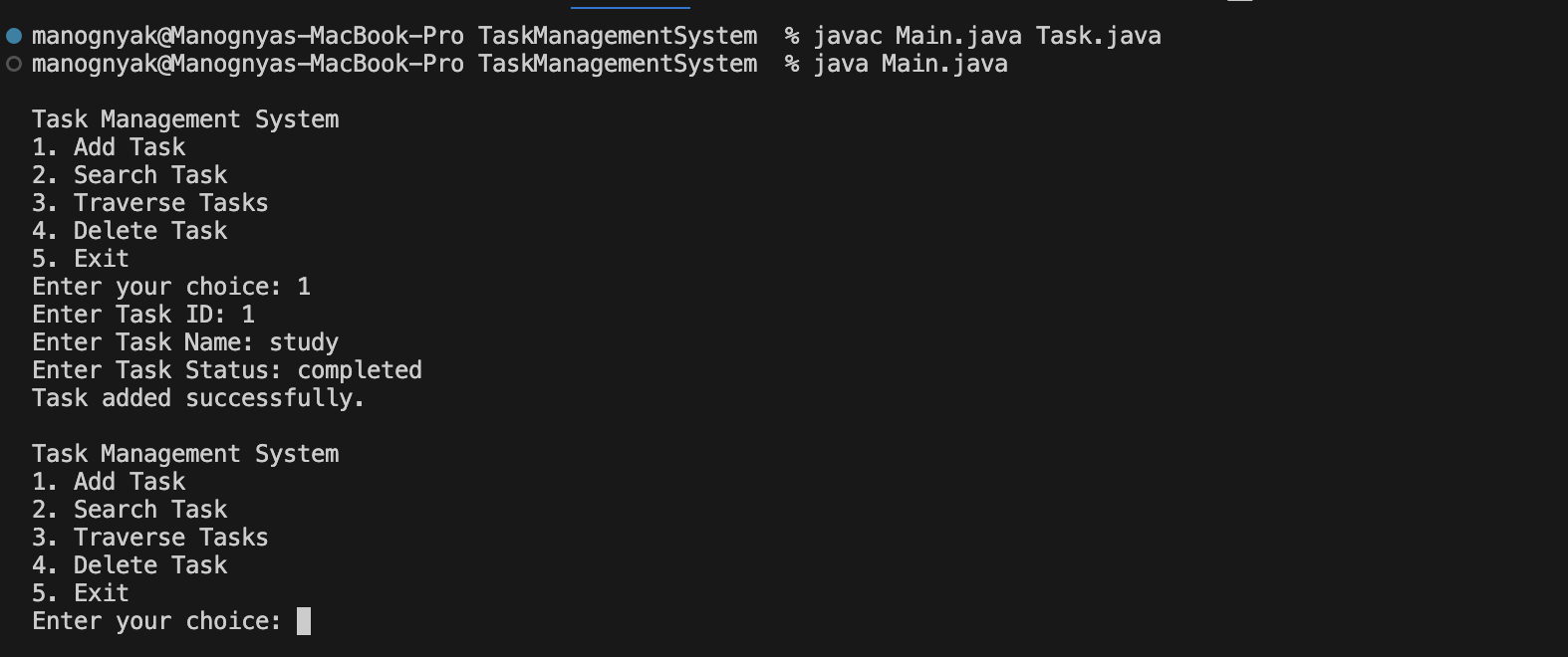
}

return head;

}

}

**Output**

****

**Time Complexity Analysis:**

Insert: O(n)

Delete: O(n)

Traverse: O(n)

Search: O(n)

**Advantages of Linked Lists over Arrays:**

1. Dynamic Size

2. Efficient Insertions/Deletions

3. No memory waste or overflow

4. Better for frequent modifications