**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.

**1. Understand Linked Lists**

**Singly Linked List**

A singly linked list is a type of linked list where each node points to the next node in the sequence and the last node points to null. It allows traversal in one direction, from the head node to the end.

**Doubly Linked List**

A doubly linked list is a type of linked list where each node has two pointers: one pointing to the next node and one pointing to the previous node. This allows traversal in both directions, from the head to the end and from the end to the head.

**2. Setup**

**Task Class**

The Task class will have attributes for taskId, taskName, and status. For managing tasks using a singly linked list, we'll also have a next pointer to point to the next task in the list.

**3. Implementation**

We will implement a singly linked list to manage tasks with methods to add, search, traverse, and delete tasks.

Task.java:

public 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;

    }

}

TaskMaanager.java:

import java.util.\*;

public class TaskManager{

    private Task head;

    public TaskManager() {

        this.head = null;

   }

    public void addTask(int taskId, String taskName, String status) {

        Task newTask = new Task(taskId, taskName, status);

        if(head==null)

        {

            head=newTask;

        }

        else{

            Task current=head;

            while(current.next!=null){

                current=current.next;

            }

            current.next=newTask;

        }

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

    }

    public Task searchTask(int taskId) {

        Task current = head;

        while (current != null) {

            if (current.taskId == taskId) {

                return current;

            }

            current = current.next;

        }

        return null;

    }

    public void deleteTask(int taskId) {

        Task current = head;

        Task previous = null;

        while (current != null) {

            if (current.taskId == taskId) {

                if (previous != null) {

                    previous.next = current.next;

                } else {

                    head = current.next;

                }

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

                return;

            }

            previous = current;

            current = current.next;

        }

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

    }

    public void traverseTasks() {

        Task current = head;

        while (current != null) {

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

            current = current.next;

        }

    }

    public static void main(String[] args) {

        TaskManager taskManager = new TaskManager();

        Scanner in=new Scanner(System.in);

        while(true)

        {

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

            System.out.println("1.ADD TASK ");

            System.out.println("2.SEARCH TASK");

            System.out.println("3.DELETE TASK");

            System.out.println("4.TRAVERSE TASK");

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

            System.out.println("enter choice: ");

            int ch = in.nextInt();

            in.nextLine();

            switch(ch)

            {

            case 1:

            {

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

                System.out.println("Enter task id to be added: ");

                int taskid=in.nextInt();

                in.nextLine();

                System.out.println("Enter task name to be added: ");

                String taskName=in.nextLine();

                System.out.println("Enter task status to be addded: ");

                String status=in.nextLine();

                taskManager.addTask(taskid,taskName,status);

                break;

            }

            case 2:

            {

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

                System.out.println("enter task id to search the task: ");

                int srch=in.nextInt();

                Task task = taskManager.searchTask(srch);

                if (task != null)

                {

                    System.out.println("Found task: " + task.taskName + " with status " + task.status);

                } else

                 {

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

                }

                break;

            }

            case 3:

            {

                System.out.println("3.Delete task");

                System.out.println("Enter taskId to delete task: ");

                int deltask=in.nextInt();

                taskManager.deleteTask(deltask);

                break;

            }

            case 4:

            {

                System.out.println("4.Traverse task");

                taskManager.traverseTasks();

            }

        }

        }

    }

}

**4. Analysis**

**Time Complexity of Operations**

* **Add Task**: O(1) - Adding a task to the head of the list takes constant time.
* **Search Task**: O(n) - In the worst case, you might have to traverse the entire list to find the task.
* **Delete Task**: O(n) - In the worst case, you might have to traverse the entire list to find and delete the task.
* **Traverse Tasks**: O(n) - Traversing the list requires visiting each node once.

**Advantages of Linked Lists over Arrays for Dynamic Data**

* **Dynamic Size**: Linked lists can easily grow and shrink in size by adding or removing nodes, whereas arrays have a fixed size.
* **Memory Allocation**: Linked lists use dynamic memory allocation, which can be more efficient when the number of elements is not known in advance.
* **Insertion and Deletion**: Insertion and deletion operations are more efficient in linked lists because they don't require shifting elements as in arrays.

This project showcases the implementation and management of tasks using a singly linked list in Java, highlighting the basic operations and their complexities.