**EXERCISE 5:**

**TASK MANAGEMENT SYSTEM**

**Singly Linked List:**

1. A singly linked list consists of nodes where each node contains data and a reference to the next node in the sequence.
2. The first node is called the head, and the last node points to null.
3. Operations such as insertion and deletion can be more efficient compared to arrays for certain scenarios (like when operations are performed at the beginning).

**Doubly Linked List:**

1. A doubly linked list consists of nodes where each node contains data, a reference to the next node, and a reference to the previous node.
2. This allows for traversal in both directions (forward and backward).
3. It provides more flexibility but requires more memory for the additional reference to the previous node.

**Add Task:**

Best Case: O(1) (if adding at the head)

Worst Case: O(n) (if adding at the end)

**Search Task:**

Best Case: O(1) (if the task is at the head)

Worst Case: O(n) (if the task is at the end or not found)

**Traverse Tasks:**

Best Case: O(n)

Worst Case: O(n)

**Delete Task:**

Best Case: O(1) (if the task is at the head)

Worst Case: O(n) (if the task is at the end or not found)

class Task {

private int taskId;

private String taskName;

private String status;

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

this.taskId = taskId;

this.taskName = taskName;

this.status = status;

}

public int getTaskId() { return taskId; }

public String getTaskName() { return taskName; }

public String getStatus() { return status; }

@Override

public String toString() {

return "Task{" +

"taskId=" + taskId +

", taskName='" + taskName + '\'' +

", status='" + status + '\'' +

'}';

}

}

class Node {

Task task;

Node next;

public Node(Task task) {

this.task = task;

this.next = null;

}

}

class TaskLinkedList {

private Node head;

public TaskLinkedList() {

this.head = null;

}

// Add a task

public void addTask(Task task) {

Node newNode = new Node(task);

if (head == null) {

head = newNode;

} else {

Node temp = head;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newNode;

}

}

// Search a task by ID

public Task searchTask(int taskId) {

Node temp = head;

while (temp != null) {

if (temp.task.getTaskId() == taskId) {

return temp.task;

}

temp = temp.next;

}

return null;

}

// Traverse all tasks

public void traverseTasks() {

Node temp = head;

while (temp != null) {

System.out.println(temp.task);

temp = temp.next;

}

}

// Delete a task by ID

public boolean deleteTask(int taskId) {

if (head == null) {

return false;

}

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

head = head.next;

return true;

}

Node temp = head;

while (temp.next != null) {

if (temp.next.task.getTaskId() == taskId) {

temp.next = temp.next.next;

return true;

}

temp = temp.next;

}

return false;

}

}

public class TaskManagementSystem {

public static void main(String[] args) {

TaskLinkedList taskList = new TaskLinkedList();

// Add tasks

taskList.addTask(new Task(1, "Design system", "In Progress"));

taskList.addTask(new Task(2, "Implement feature", "To Do"));

taskList.addTask(new Task(3, "Test application", "To Do"));

// Traverse tasks

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

taskList.traverseTasks();

// Search for a task

int searchId = 2;

Task task = taskList.searchTask(searchId);

if (task != null) {

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

} else {

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

}

// Delete a task

int deleteId = 1;

if (taskList.deleteTask(deleteId)) {

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

} else {

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

}

// Traverse tasks after deletion

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

taskList.traverseTasks();

}

}