**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).**

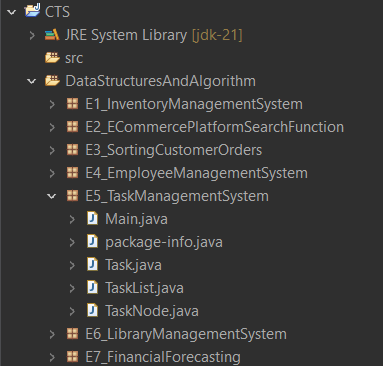
**Singly Linked List**

* Each node has: data and next
* Traverses one way (forward only)

**Doubly Linked List**

* Each node has: data, prev, and next
* Can traverse forward and backward

1. **Setup:**
   * **Create a class Task with attributes like taskId, taskName, and status.**

****

1. **Implementation:**
   * **Implement a singly linked list to manage tasks.**
   * **Implement methods to add, search, traverse, and delete tasks in the linked list.**

**Task.java**

package E5\_TaskManagementSystem;

public class Task {

int taskId;

String taskName;

String status;

public Task(int id, String name, String status) {

this.taskId = id;

this.taskName = name;

this.status = status;

}

public String toString() {

return taskId + " - " + taskName + " [" + status + "]";

}

}

**TaskNode.java**

package E5\_TaskManagementSystem;

public class TaskNode {

Task task;

TaskNode next;

public TaskNode(Task task) {

this.task = task;

this.next = null;

}

}

**TaskList.java**

package E5\_TaskManagementSystem;

public class TaskList {

private TaskNode head;

// Add task at the end

public void add(Task task) {

TaskNode newNode = new TaskNode(task);

if (head == null) {

head = newNode;

} else {

TaskNode temp = head;

while (temp.next != null) {

temp = temp.next;

}temp.next = newNode;

}

}

// Search by ID

public Task search(int id) {

TaskNode temp = head;

while (temp != null) {

if (temp.task.taskId == id) return temp.task;

temp = temp.next;

}

return null;

}

// Traverse all

public void showAll() {

TaskNode temp = head;

while (temp != null) {

System.***out***.println(temp.task);

temp = temp.next;

}

}

// Delete by ID

public void delete(int id) {

if (head == null) return;

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

head = head.next;

return;

}

TaskNode prev = head;

TaskNode curr = head.next;

while (curr != null) {

if (curr.task.taskId == id) {

prev.next = curr.next;

return;

}

prev = curr;

curr = curr.next;

}

}

}

**Main.java**

package E5\_TaskManagementSystem;

public class Main {

public static void main(String[] args) {

TaskList list = new TaskList();

list.add(new Task(1, "Design UI", "Pending"));

list.add(new Task(2, "Write Code", "In Progress"));

list.add(new Task(3, "Test App", "Pending"));

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

list.showAll();

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

Task found = list.search(2);

System.*out*.println(found != null ? found : "Not found");

System.*out*.println("\nDeleting Task ID 1");

list.delete(1);

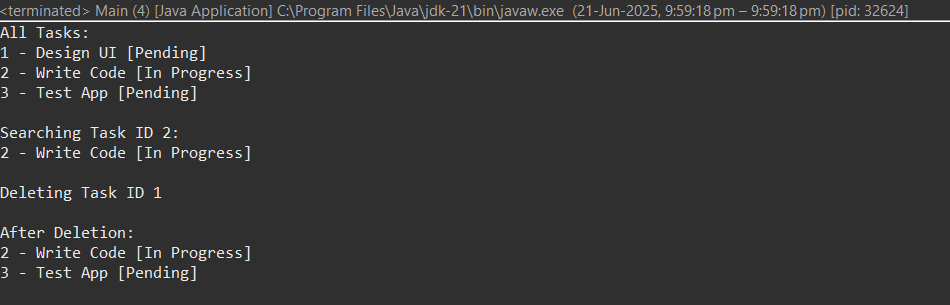
System.*out*.println("\nAfter Deletion:");

list.showAll();

}

}

**Output:**



1. **Analysis:**
   * **Analyze the time complexity of each operation.**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Time Complexity** | **Explanation** |
| **Add** | O(n) | Adds at end (can be O(1) with tail) |
| **Search** | O(n) | Check node by node |
| **Traverse** | O(n) | Visits each task |
| **Delete** | O(n) | Needs to find and update the next link |

* + **Discuss the advantages of linked lists over arrays for dynamic data.**

|  |  |
| --- | --- |
| **Linked List** | **Array** |
| Dynamic size (no resizing needed) | Fixed size (or needs manual resizing) |
| Easy insert/delete anywhere (O(1)) | Costly insert/delete (O(n)) |
| No wasted memory from extra capacity | May reserve unused space |