**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.

**Solution:**

**1. Understand Linked Lists**

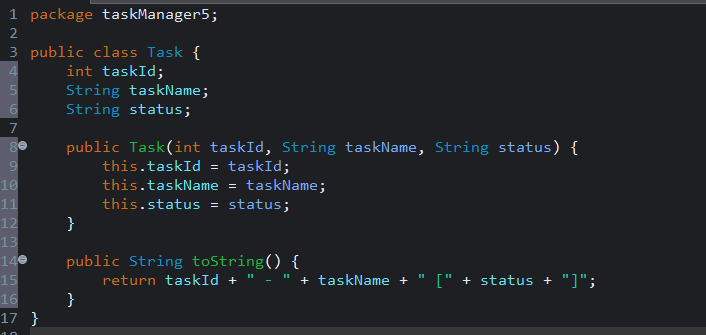
**Types of Linked Lists:**

| **Type** | **Description** |
| --- | --- |
| Singly Linked List | Each node has a data and a reference (next) to the next node. Traversal is only forward. |
| Doubly Linked List | Each node has data, a next reference, and a prev reference. Traversal can be both forward and backward. |

**Why Linked Lists?**

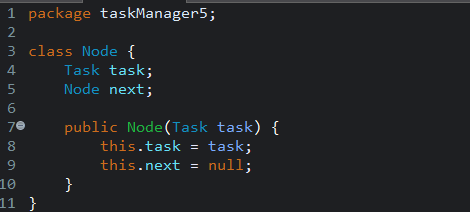
* Dynamic size: No need to predefine size like arrays.
* Efficient insertion/deletion: Especially at the beginning or middle.
* Arrays require shifting elements; linked lists just adjust pointers.

**2. Setup: Task Class**

****

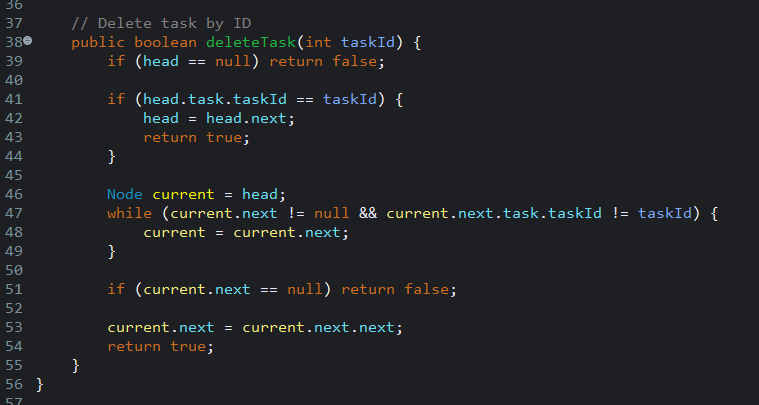
**3. Implementation: Singly Linked List**

**Node Class**

****

**Task Management System**

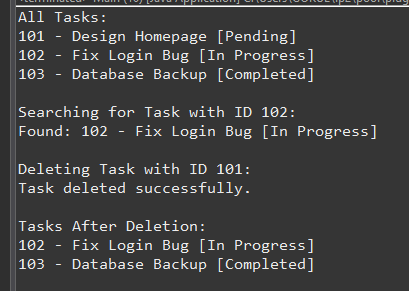
****

****

**Main Method to Test Task Management System**

****

**Output:**

****

**4. Analysis**

**Time Complexity**

| Operation | Time Complexity |
| --- | --- |
| Add (end) | O(n) |
| Search | O(n) |
| Traverse | O(n) |
| Delete | O(n) |

**Note: With an additional tail pointer or insertion at the head, adding can be optimized to O(1).**

**Advantages of Linked Lists over Arrays**

| Arrays | Linked Lists |
| --- | --- |
| Fixed size (need resizing) | Dynamic size (no preallocation) |
| Insert/delete is costly (O(n)) | Insert/delete is efficient (O(1) at head) |
| Fast random access (O(1)) | Sequential access (O(n)) |
| Memory overhead is minimal | Slight overhead due to next |