**Solution 5: Task Management System**

Scenario: You are developing a task management system where tasks need to be added, deleted, and traversed efficiently.

**>> Explain the different types of linked lists (Singly Linked List, Doubly Linked List).**

**Linked Lists** are a fundamental data structure used to store collections of elements. Each element (or node) contains data and a reference (or link) to the next node in the sequence.

* **Singly Linked List**:
  + **Description**: Each node points to the next node in the list. It allows traversal in one direction only.
  + **Advantages**:
    - **Dynamic Size**: Can grow and shrink in size more easily than arrays.
    - **Efficient Insertions/Deletions**: Insertions and deletions are more efficient compared to arrays if the position is known, as they don't require shifting of elements.
  + **Disadvantages**:
    - **Sequential Access**: Accessing elements requires traversal from the head node, which can be slower compared to arrays.
* **Doubly Linked List** (for reference):
  + **Description**: Each node points to both the next node and the previous node, allowing bidirectional traversal.
  + **Advantages**:
    - **Bidirectional Traversal**: Can be traversed in both forward and backward directions.
  + **Disadvantages**:
    - **Increased Memory**: Requires extra space for the previous node reference.

**Analysis**

**>> Analyze the time complexity of each operation.**

* **Add Task**:
  + **Time Complexity**: O(n) for adding at the end of the list (where n is the number of tasks). This is because you need to traverse the list to find the end.
  + **Space Complexity**: O(1) for each new task node.
* **Search Task**:
  + **Time Complexity**: O(n) in the worst case, as you may need to traverse the entire list to find a task with the given ID.
  + **Space Complexity**: O(1), as only a few variables are used for traversal.
* **Traverse Tasks**:
  + **Time Complexity**: O(n) because you must visit each node to collect all tasks.
  + **Space Complexity**: O(n) if you store tasks in a list for output.
* **Delete Task**:
  + **Time Complexity**: O(n) in the worst case, as you may need to traverse the list to find the node to delete.
  + **Space Complexity**: O(1), as only a few variables are used for traversal and node deletion.

**>> Advantages of Linked Lists Over Arrays for Dynamic Data**

1. **Dynamic Size**:
   * Linked lists dynamically allocate memory as tasks are added, so they can easily grow or shrink without needing to resize like arrays.
2. **Efficient Insertions and Deletions**:
   * Linked lists provide O(1) complexity for insertions and deletions if the node reference is known, making them more efficient for frequent changes compared to arrays, which require shifting elements and are O(n) for insertions and deletions.
3. **No Wasted Space**:
   * Linked lists use memory only for existing nodes, while arrays might allocate excess space to accommodate potential growth, which can be inefficient if the maximum size is overestimated.
4. **Flexibility**:
   * Linked lists are more flexible for scenarios where the number of tasks varies greatly or is unknown at the start. They handle frequent modifications more gracefully than arrays.

linked lists are suitable for applications where dynamic changes are frequent and size management is crucial, while arrays are better for applications requiring constant-time access and where the size is relatively static or changes infrequently.

**How to Run the code :**

* Run Main.java file