Quick Memorization Tips

1. **Singly Linked List (SLL)**
    - Each node has `data` and `next`.
    - Insert at end: Traverse till `next == null` then add.
    - Delete at position: Traverse to `pos-1`, change `next` pointer.
2. **Doubly Linked List (DLL)**
    - Each node has `data`, `prev`, and `next`.
    - Insert at end: Similar to SLL, but update `prev`.
    - Delete at position: Update both `next` and `prev`.
3. **Stack (LL Implementation)**
    - **Push**: Insert at the head.
    - **Pop**: Remove from the head.
    - **Peek**: Return `top` value.
    - **LIFO** (Last In, First Out).
4. **Queue (LL Implementation)**
    - **Enqueue**: Insert at `end`.
    - **Dequeue**: Remove from `front`.
    - **FIFO** (First In, First Out).

1. Singly Linked List (SLL)

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class SinglyLinkedList {
    Node head;

    void insertAtEnd(int data) {
        Node newNode = new Node(data);
        if (head == null) { head = newNode; return; }
        Node temp = head;
        while (temp.next != null) temp = temp.next;
        temp.next = newNode;
    }

    void deleteAtPosition(int pos) {
        if (head == null) return;
        if (pos == 0) { head = head.next; return; }
        Node temp = head;
        for (int i = 0; temp != null && i < pos - 1; i++) temp = temp.next;
        if (temp == null || temp.next == null) return;
        temp.next = temp.next.next;
    }

    void display() {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " -> "); temp = temp.next;
        }
        System.out.println("null");
    }
}
```

2. Doubly Linked List (DLL)

```
class DNode {
    int data;
    DNode prev, next;
    DNode(int data) {
        this.data = data;
        this.prev = null;
        this.next = null;
    }
}

class DoublyLinkedList {
    DNode head;

    void insertAtEnd(int data) {
        DNode newNode = new DNode(data);
        if (head == null) { head = newNode; return; }
        DNode temp = head;
        while (temp.next != null) temp = temp.next;
        temp.next = newNode;
        newNode.prev = temp;
    }

    void deleteAtPosition(int pos) {
        if (head == null) return;
        DNode temp = head;
        for (int i = 0; temp != null && i < pos; i++) temp = temp.next;
        if (temp == null) return;
        if (temp.prev != null) temp.prev.next = temp.next;
        if (temp.next != null) temp.next.prev = temp.prev;
        if (temp == head) head = temp.next;
    }

    void display() {
        DNode temp = head;
        while (temp != null) {
            System.out.print(temp.data + " <-> "); temp = temp.next;
        }
        System.out.println("null");
    }
}
```

3. Stack (Using Linked List)

```
class StackNode {
    int data;
    StackNode next;
    StackNode(int data) {
        this.data = data;
        this.next = null;
    }
}

class Stack {
    StackNode top;

    void push(int data) {
        StackNode newNode = new StackNode(data);
        newNode.next = top;
        top = newNode;
    }

    int pop() {
        if (top == null) return -1;
        int val = top.data;
        top = top.next;
        return val;
    }

    int peek() { return (top == null) ? -1 : top.data; }

    boolean isEmpty() { return top == null; }

    void display() {
        StackNode temp = top;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }
}
```

4. Queue (Using Linked List)

```
class QueueNode {
    int data;
    QueueNode next;
    QueueNode(int data) {
        this.data = data;
        this.next = null;
    }
}

class Queue {
    QueueNode front, rear;

    void enqueue(int data) {
        QueueNode newNode = new QueueNode(data);
        if (rear == null) { front = rear = newNode; return; }
        rear.next = newNode;
        rear = newNode;
    }

    int dequeue() {
        if (front == null) return -1;
        int val = front.data;
        front = front.next;
        if (front == null) rear = null;
        return val;
    }

    boolean isEmpty() { return front == null; }

    void display() {
        QueueNode temp = front;
        while (temp != null) {
            System.out.print(temp.data + " -> ");
            temp = temp.next;
        }
        System.out.println("null");
    }
}
```