**Approach:**

To find the second-to-last node in a singly linked list:

1. **Edge Case Handling:**
   * If the list is **empty** (head is null), return null.
   * If the list has **only one node**, return null since there is no second-to-last node.
2. **Iterate Through the List:**
   * Use a pointer (current) starting from head.
   * Stop **one node before** the last node (i.e., when current.next.next == null).
   * Return the current node.

**Algorithm (Pseudocode):**

Algorithm findSecondToLast(head)

1. If head is NULL OR head.next is NULL, return NULL (No second-to-last node)

2. Set current = head

3. While current.next.next is NOT NULL:

a. Move current to current.next

4. Return current (which is now the second-to-last node)

**Java Implementation:**

public class SinglyLinkedList<E> {

private static class Node<E> {

E element;

Node<E> next;

public Node(E element, Node<E> next) {

this.element = element;

this.next = next;

}

}

private Node<E> head = null;

private Node<E> tail = null;

private int size = 0;

public void addLast(E element) {

Node<E> newNode = new Node<>(element, null);

if (tail == null) {

head = tail = newNode;

} else {

tail.next = newNode;

tail = newNode;

}

size++;

}

public E findSecondToLast() {

if (head == null || head.next == null) {

return null; // No second-to-last node exists

}

Node<E> current = head;

while (current.next.next != null) { // Stop at second-to-last node

current = current.next;

}

return current.element;

}

public static void main(String[] args) {

SinglyLinkedList<Integer> list = new SinglyLinkedList<>();

list.addLast(10);

list.addLast(20);

list.addLast(30);

list.addLast(40);

System.out.println("Second-to-Last Node: " + list.findSecondToLast()); // Output: 30

}

}

**⏳ Time Complexity:**

* **O(n)** → We traverse the list once.
* **O(1) Space Complexity** → Only a few pointers are used.

**Edge Cases Tested:**

| **Case** | **Input List** | **Output** |
| --- | --- | --- |
| Empty List | [] | null |
| One Node | [10] | null |
| Two Nodes | [10, 20] | 10 |
| Multiple Nodes | [10, 20, 30, 40] | 30 |