class Node {

public int data;

public Node next;

public Node(int value) {

data = value;

next = null;

}

}

class SinglyLL {

public Node first;

public int count;

public SinglyLL() {

first = null;

count = 0;

}

public void insertLast(int no) {

Node newn = new Node(no);

if (first == null) {

first = newn;

} else {

Node temp = first;

while (temp.next != null) {

temp = temp.next;

}

temp.next = newn;

}

count++;

}

public void display() {

System.out.print("[ ");

Node temp = first;

while (temp != null) {

System.out.print(temp.data);

temp = temp.next;

if (temp != null) {

System.out.print(", ");

}

}

System.out.print(" ]");

}

public int findNthFromEnd(int n) {

Node mainPointer = first;

Node refPointer = first;

// Move refPointer n nodes ahead

for (int i = 0; i < n; i++) {

if (refPointer == null) {

System.out.println("List has fewer than " + n + " nodes.");

return -1; // Indicating that n is larger than the list size

}

refPointer = refPointer.next;

}

// Move both pointers until refPointer reaches the end

while (refPointer != null) {

mainPointer = mainPointer.next;

refPointer = refPointer.next;

}

// mainPointer now points to the nth node from the end

return mainPointer.data;

}

}

class Question5 {

public static void main(String[] args) {

// Test 1

SinglyLL obj = new SinglyLL();

obj.insertLast(10);

obj.insertLast(20);

obj.insertLast(30);

obj.insertLast(40);

obj.insertLast(50);

int n = 2; // Using the same variable n for both tests

// Display the input list and the value of n

System.out.print("Input: List = ");

obj.display();

System.out.println(", n = " + n);

int nthNode = obj.findNthFromEnd(n);

// Display the output

System.out.println("Output: " + nthNode);

// Test 2

obj = new SinglyLL(); // Reinitialize obj for the second test

obj.insertLast(5);

obj.insertLast(15);

obj.insertLast(25);

obj.insertLast(35);

// Using the same variable n, but now setting it to a different value

n = 4;

// Display the input list and the value of n

System.out.print("Input: List = ");

obj.display();

System.out.println(", n = " + n);

int nthNode2 = obj.findNthFromEnd(n);

// Display the output

System.out.println("Output: " + nthNode2);

}

}  
  
  
explanation  
The Node class in your code represents an individual element in a singly linked list, with data for storing the value and next for pointing to the next node. The SinglyLL class provides methods to insert nodes at the end (insertLast), display the linked list (display), and find the nth node from the end (findNthFromEnd). In findNthFromEnd, two pointers are used: refPointer is moved n nodes ahead, and then both refPointer and mainPointer are moved together until refPointer reaches the end. This makes mainPointer point to the nth node from the end of the list.  
  
flowchart  
 ┌───────────────────────────┐

│ Start │

└────────────┬──────────────┘

│

▼

┌───────────────────────────┐

│ Create SinglyLL Instance │

└────────────┬──────────────┘

│

▼

┌───────────────────────────┐

│ Insert Nodes │

└────────────┬──────────────┘

│

▼

┌───────────────────────────┐

│ Display List │

└────────────┬──────────────┘

│

▼

┌───────────────────────────┐

│ Find Nth Node From End │

└────────────┬──────────────┘

│

▼

┌───────────────────────────┐

│ Return Nth Node's Data │

└────────────┬──────────────┘

│

▼

┌───────────────────────────┐

│ End │

└───────────────────────────┘   
  
time and space complexity  
The insertLast method has a time complexity of O(n) because it might need to traverse the entire list to find the last node, with a space complexity of O(1) since it only uses a few variables. The display method runs with a time complexity of O(n) as it traverses all the nodes to print them, and its space complexity is O(1) due to constant additional space usage. The findNthFromEnd method operates with a time complexity of O(n) because it involves two passes over the list, and it has a space complexity of O(1) as it uses a fixed amount of extra space.  
  
output  
