**Merge two sorted linked lists.**explanation  
The provided Java code defines a simple implementation of a singly linked list (SinglyLL) with basic functionalities such as inserting nodes at the end, displaying the list, and merging two sorted linked lists. Each node is represented by the Node class, which contains an integer data field and a reference to the next node. The SinglyLL class includes methods to insert nodes, display the linked list, and merge two sorted linked lists into a single sorted list. In the main method, two test cases are executed, demonstrating the merging of two sorted lists and printing the input and output in a specified format.

**Time and Space Complexity**

The time complexity of the merge method is O(n+m)O(n + m)O(n+m), where nnn is the number of nodes in the first list and mmm is the number of nodes in the second list. This is because each node in both lists is visited once. The space complexity is O(n+m)O(n + m)O(n+m) as well, since a new linked list is created to store the merged results, which will contain all the nodes from both input lists.

Flowchart  
Start

↓

Initialize mergedList

↓

Set temp1 to first node of List1

Set temp2 to first node of List2

↓

While (temp1 is not null AND temp2 is not null)

↓

If (temp1.data <= temp2.data)

↓

Insert temp1.data into mergedList

Move temp1 to next node

Else

↓

Insert temp2.data into mergedList

Move temp2 to next node

↓

End While

↓

While (temp1 is not null)

↓

Insert temp1.data into mergedList

Move temp1 to next node

↓

End While

↓

While (temp2 is not null)

↓

Insert temp2.data into mergedList

Move temp2 to next node

↓

End While

↓

Return mergedList

↓

End  
  
program

class Node {

public int data;

public Node next;

public Node(int value) {

data = value;

next = null;

}

}

class SinglyLL {

public Node first;

public SinglyLL() {

first = null;

}

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;

}

}

public void display() {

Node temp = first;

System.out.print("[ ");

while (temp != null) {

System.out.print(temp.data);

temp = temp.next;

if (temp != null) {

System.out.print(", ");

}

}

System.out.print(" ]");

}

public SinglyLL merge(SinglyLL list2) {

SinglyLL mergedList = new SinglyLL();

Node temp1 = this.first;

Node temp2 = list2.first;

while (temp1 != null && temp2 != null) {

if (temp1.data <= temp2.data) {

mergedList.insertLast(temp1.data);

temp1 = temp1.next;

} else {

mergedList.insertLast(temp2.data);

temp2 = temp2.next;

}

}

while (temp1 != null) {

mergedList.insertLast(temp1.data);

temp1 = temp1.next;

}

while (temp2 != null) {

mergedList.insertLast(temp2.data);

temp2 = temp2.next;

}

return mergedList;

}

}

class Question4 {

public static void main(String[] args) {

// Test Case 1

SinglyLL list1 = new SinglyLL();

list1.insertLast(1);

list1.insertLast(3);

list1.insertLast(5);

SinglyLL list2 = new SinglyLL();

list2.insertLast(2);

list2.insertLast(4);

list2.insertLast(6);

System.out.print("Input: List1 = ");

list1.display();

System.out.print(", List2 = ");

list2.display();

SinglyLL mergedList1 = list1.merge(list2);

System.out.print("\nOutput: Merged List = ");

mergedList1.display();

System.out.println();

// Test Case 2

SinglyLL list3 = new SinglyLL();

list3.insertLast(10);

list3.insertLast(15);

list3.insertLast(20);

SinglyLL list4 = new SinglyLL();

list4.insertLast(12);

list4.insertLast(18);

list4.insertLast(25);

System.out.print("Input: List1 = ");

list3.display();

System.out.print(", List2 = ");

list4.display();

SinglyLL mergedList2 = list3.merge(list4);

System.out.print("\nOutput: Merged List = ");

mergedList2.display();

System.out.println();

}

}  
  
output  
  
