**1. Implement a singly linked list with basic operations: insert, delete, search.**

* **Test Case 1**:  
  Input: Insert 3 → Insert 7 → Insert 5 → Delete 7 → Search 5  
  Output: List = [3, 5], Found = True
* **Test Case 2**:  
  Input: Insert 9 → Insert 4 → Delete 4 → Search 10  
  Output: List = [9], Found = False

Program  
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("List=[ ");

Node temp = first;

while (temp != null) {

System.out.print(temp.data);

temp = temp.next;

if (temp != null) {

System.out.print(", ");

}

}

System.out.println(" ]");

}

public boolean search(int no) {

Node temp = first;

while (temp != null) {

if (temp.data == no) {

return true;

}

temp = temp.next;

}

return false;

}

public int count() {

return count;

}

public void delete(int no) {

Node temp = first;

Node prev = null;

while (temp != null) {

if (temp.data == no) {

if (prev == null) {

first = temp.next;

} else {

prev.next = temp.next;

}

count--;

return;

}

prev = temp;

temp = temp.next;

}

}

}

class Question1 {

public static void main(String[] args) {

SinglyLL obj = new SinglyLL();

boolean bRet = false;

obj.insertLast(9);

obj.insertLast(4);

// obj.insertLast(5);

obj.delete(4);

obj.display();

bRet = obj.search(10);

System.out.println("Found = " + bRet);

}

}  
  
flowchart  
 ┌───────────────────────────┐

│ Start │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ Create Node (InsertLast) │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ Traverse to Last Node │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ Add New Node at End │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ Display List │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ Search for Node │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ Delete Node │

└───────────┬───────────────┘

│

▼

┌───────────────────────────┐

│ End │

└───────────────────────────┘

Time and space comp.  
The insertLast method has a time complexity of O(n) since it may need to traverse the entire linked list to find the last node, while its space complexity is O(1) as it uses a constant amount of extra space. The display method has a time complexity of O(n) because it traverses all nodes to print them, with a space complexity of O(1) for constant space usage. The search method also has a time complexity of O(n) due to potentially traversing the whole list to find a node, and a space complexity of O(1) for constant extra space. Lastly, the delete method has a time complexity of O(n) for possibly needing to traverse the entire list to locate the node to delete, and a space complexity of O(1) due to its constant space requirement.  
  
