Delete given node in a Linked List : O(1) approach

Problem Statement: Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the head of the list instead, you will be given access to **the node to be deleted** directly. It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

Examples:

Example 1:

Input:

```
 Linked list: [1,4,2,3]

       Node = 2
```

Output:

```
Linked list: [1,4,3]
```

Explanation: Access is given to node 2. After deleting nodes, the linked list will be modified to [1,4,3].

Example 2:

Input:

```
 Linked list: [1,2,3,4]

       Node = 1
```

Output: Linked list: [2,3,4]

Explanation:

Access is given to node 1. After deleting nodes, the linked list will be modified to [2,3,4].

Solution:

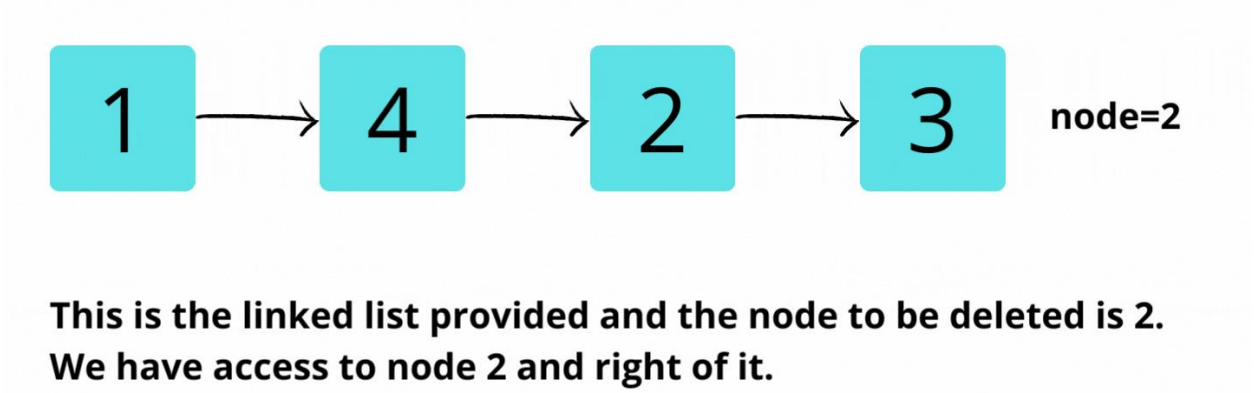
Disclaimer: *Don't jump directly to the solution, try it out yourself first.*

Approach:

We are given access to nodes that we have to delete from the linked list. So, whatever operation we want to do in the linked list, we can operate in the right part of the linked list from the node to be deleted.

The approach is to copy the next node's value in the deleted node. Then, link node to next of next node. This does not delete that node but indirectly it removes that node from the linked list.

Dry Run:



This is the linked list provided and the node to be deleted is 2.
We have access to node 2 and right of it.

```
import java.util.*;
class Node {
        int num;
        Node next;
        Node(int a) {
            num = a;
            next = null;
        }
}
class TUF{
```

```
//function to insert node at the end of the list
static Node insertNode(Node head,int val) {
    Node newNode = new Node(val);
    if(head == null) {
        head = newNode;
        return head;
    }
    Node temp = head;
    while(temp.next != null) temp = temp.next;
    temp.next = newNode;
    return head;
}
//function to get reference of the node to delete
static Node getNode(Node head,int val) {
    if(head==null)
    return null;
    while(head.num != val) head = head.next;

    return head;
}
//delete function as per the question
static void deleteNode(Node t) {
    if(t==null)
    return;
    t.num = t.next.num;
    t.next = t.next.next;
    return;
}
//printing the list function
static void printList(Node head) {
    if(head==null)
    return;
    while(head.next!=null ) {
        System.out.print(head.num+"->");
        head = head.next;
    }
    System.out.println(head.num);
}

public static void main(String args[]) {
    Node head = null;
    //inserting node
    head=insertNode(head,1);
    head=insertNode(head,4);
    head=insertNode(head,2);
    head=insertNode(head,3);
    //printing given list
    System.out.println("Given Linked List: ");
    printList(head);
    Node t = getNode(head,2);
    //delete node
```

```
    deleteNode(t);
    //list after deletion operation
    System.out.println("Linked List after deletion: ");
    printList(head);
}
}
```

Output:

Given Linked List:
1->4->2->3
Linked List after deletion:
1->4->3

Time Complexity: O(1)

***Reason*:** It is a two-step process that can be obtained in constant time.

Space Complexity: O(1)

***Reason*:** No extra data structure is used.

//solution for the problem

```
public void deleteNode(ListNode node) {
      node.val=node.next.val;
       node.next=node.next.next;
   }
```