Let's briefly review the structure definition of a node in the singly linked list.

|  |
| --- |
| // Definition for singly-linked list.  public class SinglyListNode {  int val;  SinglyListNode next;  SinglyListNode(int x) { val = x; }  } |

Based on this definition, we are going to give you the solution step by step.

**1. Initiate a new linked list: represent a linked list using the head node.**

|  |
| --- |
| class MyLinkedList {  private SinglyListNode head;  /\*\* Initialize your data structure here. \*/  public MyLinkedList() {  head = null;  }  } |

**2. Traverse the linked list to get element by index.**

|  |
| --- |
| /\*\* Helper function to return the index-th node in the linked list. \*/  private SinglyListNode getNode(int index) {  SinglyListNode cur = head;  for (int i = 0; i < index && cur != null; ++i) {  cur = cur.next;  }  return cur;  }  /\*\* Helper function to return the last node in the linked list. \*/  private SinglyListNode getTail() {  SinglyListNode cur = head;  while (cur != null && cur.next != null) {  cur = cur.next;  }  return cur;  }  /\*\* Get the value of the index-th node in the linked list. If the index is invalid, return -1. \*/  public int get(int index) {  SinglyListNode cur = getNode(index);  return cur == null ? -1 : cur.val;  } |

**3. Add a new node.**

|  |
| --- |
| /\*\* Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list. \*/  public void addAtHead(int val) {  SinglyListNode cur = new SinglyListNode(val);  cur.next = head;  head = cur;  return;  }  /\*\* Append a node of value val to the last element of the linked list. \*/  public void addAtTail(int val) {  if (head == null) {  addAtHead(val);  return;  }  SinglyListNode prev = getTail();  SinglyListNode cur = new SinglyListNode(val);  prev.next = cur;  }  /\*\* Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted. \*/  public void addAtIndex(int index, int val) {  if (index == 0) {  addAtHead(val);  return;  }  SinglyListNode prev = getNode(index - 1);  if (prev == null) {  return;  }  SinglyListNode cur = new SinglyListNode(val);  SinglyListNode next = prev.next;  cur.next = next;  prev.next = cur;  } |

It is worth noting that we have to get the (index - 1)th node or the last node before we add the new node (except adding at the head) and it takes O(N) time to get a node by index, where N is the length of the linked list. It is different from adding a new node after a given node.

**5. Delete a node.**

|  |
| --- |
| /\*\* Delete the index-th node in the linked list, if the index is valid. \*/  public void deleteAtIndex(int index) {  SinglyListNode cur = getNode(index);  if (cur == null) {  return;  }  SinglyListNode prev = getNode(index - 1);  SinglyListNode next = cur.next;  if (prev != null) {  prev.next = next;  } else {  // modify head when deleting the first node.  head = next;  }  } |

Similar to the add operation, it takes O(N) time to get the node by the index which is different from deleting a given node. However, even if we already get the node we want to delete, we still have to traverse to get its previous node.