Problem 707: Design Linked List

Problem Information

Difficulty: Medium
Acceptance Rate: 29.55%
Paid Only: No
Tags: Linked List, Design

Problem Description

Design your implementation of the linked list. You can choose to use a singly or doubly linked list. A node in a singly linked list should have two attributes: `val` and `next`. `val` is the value of the current node, and `next` is a pointer/reference to the next node. If you want to use the doubly linked list, you will need one more attribute `prev` to indicate the previous node in the linked list. Assume all nodes in the linked list are **0-indexed**.

Implement the `MyLinkedList` class:

* `MyLinkedList()` Initializes the `MyLinkedList` object. * `int get(int index)` Get the value of the `indexth` node in the linked list. If the index is invalid, return `-1`. * `void addAtHead(int val)` 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. * `void addAtTail(int val)` Append a node of value `val` as the last element of the linked list. * `void addAtIndex(int index, int val)` Add a node of value `val` before the `indexth` node in the linked list. If `index` equals the length of the linked list, the node will be appended to the end of the linked list. If `index` is greater than the length, the node **will not be inserted**. * `void deleteAtIndex(int index)` Delete the `indexth` node in the linked list, if the index is valid.

Example 1:

Input ["MyLinkedList", "addAtHead", "addAtTail", "addAtIndex", "get", "deleteAtIndex", "get"] [[], [1], [3], [1, 2], [1], [1], [1]] **Output** [null, null, null, null, 2, null, 3] **Explanation** MyLinkedList myLinkedList = new MyLinkedList(); myLinkedList.addAtHead(1); myLinkedList.addAtTail(3); myLinkedList.addAtIndex(1, 2); // linked list becomes 1->2->3 myLinkedList.get(1); // return 2 myLinkedList.deleteAtIndex(1); // now the linked list is 1->3 myLinkedList.get(1); // return 3

Constraints:

* `0 <= index, val <= 1000` * Please do not use the built-in LinkedList library. * At most `2000` calls will be made to `get`, `addAtHead`, `addAtTail`, `addAtIndex` and `deleteAtIndex`.

Code Snippets

C++:

```
class MyLinkedList {
public:
MyLinkedList() {

}

int get(int index) {

}

void addAtHead(int val) {

}

void addAtTail(int val) {

}

void addAtIndex(int index, int val) {

}

void deleteAtIndex(int index) {

}
};

/**
* Your MyLinkedList object will be instantiated and called as such:
* MyLinkedList* obj = new MyLinkedList();
* int param_1 = obj->get(index);
* obj->addAtHead(val);
```

```
* obj->addAtTail(val);
* obj->addAtIndex(index,val);
* obj->deleteAtIndex(index);
*/
```

Java:

```
class MyLinkedList {

public MyLinkedList() {

}

public int get(int index) {

}

public void addAtHead(int val) {

}

public void addAtTail(int val) {

}

public void addAtIndex(int index, int val) {

}

public void deleteAtIndex(int index) {

}
}

/**
* Your MyLinkedList object will be instantiated and called as such:
* MyLinkedList obj = new MyLinkedList();
* int param_1 = obj.get(index);
* obj.addAtHead(val);
* obj.addAtTail(val);
* obj.addAtIndex(index,val);
* obj.deleteAtIndex(index);
```

```
*/
```

Python3:

```
class MyLinkedList:

def __init__(self):


def get(self, index: int) -> int:


def addAtHead(self, val: int) -> None:


def addAtTail(self, val: int) -> None:


def addAtIndex(self, index: int, val: int) -> None:


def deleteAtIndex(self, index: int) -> None:



# Your MyLinkedList object will be instantiated and called as such:
# obj = MyLinkedList()
# param_1 = obj.get(index)
# obj.addAtHead(val)
# obj.addAtTail(val)
# obj.addAtIndex(index,val)
# obj.deleteAtIndex(index)
```