Technique: Implementation

Level: Easy

Implementing a linked list is pretty simple. In a singly linked list, the Node class will store the data and a pointer to the next node. In a doubly linked list, you will also have a pointer to the previous node. You should always confirm with the interviewer if they mean singly or doubly linked list. Most of the time, they mean singly linked.

In our implementation, we have used an integer as data. You can have any kind of data in a linked list node.

We also create a LinkedList class to keep track of the *head* and *tail*.

```
public class Node {
    Node next;
    int data;

    // Getters and Setters, you can skip writing these in an interview
    public Node(Node next, int data) {
        super();
        this.next = next;
        this.data = data;
    }

    public Node getNext() {
        return next;
    }

    public void setNext(Node next) {
        this.next = next;
    }

    public int getData() {
        return data;
    }

    public void setData(int data) {
        this.data = data;
    }
}

public class LinkedList {
    Node head;
    Node tail;
```

```
    // Getters and Setters, you can skip writing these in an interview
    public LinkedList(Node head, Node tail) {
        super();
        this.head = head;
        this.tail = tail;
    }

    public Node getHead() {
        return head;
    }

    public void setHead(Node head) {
        this.head = head;
    }

    public Node getTail() {
        return tail;
    }

    public void setTail(Node tail) {
        this.tail = tail;
    }

    /*
     * Get the nth element of the list.
     * Note: Here, the first node is 1. It's not zero based like
     *       in an array
     */
    public Node get(int n) {
        Node node = head;
        for (int i = 0; i < n-1; i++) { // move forward n-1 times
            if (node != null)
                node = node.getNext();
            else {
                throw new IndexOutOfBoundsException(
                            "No node at index " + Integer.toString(n));
            }
        }

        if (node == null) {
            throw new IndexOutOfBoundsException(
                            "No node at index " + Integer.toString(n));
        }

        return node;
    }
}
```

```
/*
 *  Get function - without using LinkedList class.
 *  Only difference here is that we pass head and tail as arguments.
 *
 *  Note: Be careful with this implementation (without a class).
 *
 *  While it works in read operations, it doesn't work well if you
 *  want to modify the list.
 *
 *  For example, if you need to change the head and tail pointers, simply
 *  reassigning them in this function call will not change the pointers. It will
 *  only change the local variables.
 *  We recommend making list modification functions a part of the
 *  LinkedList class, where the head and tail pointers are stored.
 */
public Node get(int n, Node head, Node tail) {
    Node node = head;
    for (int i = 0; i < n-1; i++) { // move forward n-1 times
        if (node != null)
            node = node.getNext();
        else {
            throw new IndexOutOfBoundsException(
                        "No node at index " + Integer.toString(n));
        }
    }
    return node;
}
```