Middle of the LinkedList (easy)

Problem Statement

Given the head of a **Singly LinkedList**, write a method to return the **middle node** of the LinkedList.

If the total number of nodes in the LinkedList is even, return the second middle node.

Example 1:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> null
Output: 3
```

Example 2:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> null
Output: 4
```

Example 3:

```
Input: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> null
Output: 4
```

Try it yourself

Try solving this question here:

Java | Python3 | JS | C++

```java
class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class MiddleOfLinkedList {

  public static ListNode findMiddle(ListNode head) {
    // TODO: Write your code here
    return head;
  }

  public static void main(String[] args) {
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    head.next.next.next = new ListNode(4);
    head.next.next.next.next = new ListNode(5);
    System.out.println("Middle Node: " + MiddleOfLinkedList.findMiddle(head).value);

    head.next.next.next.next.next = new ListNode(6);
    System.out.println("Middle Node: " + MiddleOfLinkedList.findMiddle(head).value);

  }
}
```

RUN SAVE RESET

Solution

One brute force strategy could be to first count the number of nodes in the LinkedList and then find the middle node in the second iteration. Can we do this in one iteration?

We can use the **Fast & Slow pointers** method such that the fast pointer is always twice the nodes ahead of the slow pointer. This way, when the fast pointer reaches the end of the LinkedList, the slow pointer will be pointing at the middle node.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```java
class ListNode {
  int value = 0;
  ListNode next;

  ListNode(int value) {
    this.value = value;
  }
}

class MiddleOfLinkedList {

  public static ListNode findMiddle(ListNode head) {
    ListNode slow = head;
    ListNode fast = head;
    while (fast != null && fast.next != null) {
      slow = slow.next;
      fast = fast.next.next;
    }

    return slow;
  }

  public static void main(String[] args) {
    ListNode head = new ListNode(1);
    head.next = new ListNode(2);
    head.next.next = new ListNode(3);
    head.next.next.next = new ListNode(4);
```

RUN SAVE RESET

Close

Output 2.438s

```
Middle Node: 3
Middle Node: 4
Middle Node: 4
```

Time complexity

The above algorithm will have a time complexity of $O(N)$ where 'N' is the number of nodes in the LinkedList.

Space complexity

The algorithm runs in constant space $O(1)$.