DATASTRUCTURE & ALGORITHMS

INTERVIEW LEETCODE PROBLEMS & SOLUTION

**PROBLEMS TO BE COVERED**

***DATASTRUCTURES:***

1. Linked Lists
2. Doubly Linked Lists
3. Stacks & Queues
4. Binary Search Trees
5. Hash Tables
6. Graphs

***ALGORITHMS:***

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Tree Traversal

* Breadth First Search
* Depth First Search - Pre Order
* Depth First Search - Post Order
* Depth First Search - In Order

***COURSE TAKEN FROM UDEMY:***

[Java Data Structures & Algorithms + LEETCODE Exercises (udemy.com)](https://dxc.udemy.com/course/data-structures-and-algorithms-java/learn/lecture/27815396#overview)

**Problem 1 🡪 Find Middle Node ( \*\* Interview Question)**

Implement a method called **findMiddleNode** that returns the middle node of the linked list.  
If the list has an even number of nodes, the method should return the second middle node.  
Method signature:

1. public Node findMiddleNode()

Example:

1. LinkedList myList = new LinkedList(1);
2. myList.append(2);
3. myList.append(3);
4. myList.append(4);
5. myList.append(5);
6. Node middleNode = myList.findMiddleNode();
7. System.out.println(middleNode.value); // Output: 3
9. myList.append(6);
10. middleNode = myList.findMiddleNode();
11. System.out.println(middleNode.value); // Output: 4

**Note:**

In this problem, you should use the slow and fast pointer technique (also known as Floyd's Tortoise and Hare algorithm) to find the middle element of the linked list efficiently.  
The key idea is to have two pointers, one that moves twice as fast as the other. By the time the fast pointer reaches the end of the linked list, the slow pointer will be at the middle.

DO NOT use the **length** attribute in your solution.



 