Project Title: Linked List Implementation with More Operations

Objective:
The objective of this project is to implement a singly linked list using a `Node` class and a `MyLinkedList` class. The `Node` class will represent individual elements (nodes) in the linked list, and the `MyLinkedList` class will provide various operations to manipulate and query the linked list. The solution also includes advanced methods such as finding the middle element, getting the nth node from the end, inserting at a specific position, inserting before an element, and deleting after a specific element.

Class Breakdown:

1. **Node Class:**
   - This class represents an individual node in the linked list.
   - **Attributes:**
     - `data` : Stores the value of the node (String in this case).
     - `next` : Points to the next node in the list (null for the last node).
   - **Constructor:**
     - Initializes the node with the given data and sets `next` to null.

```
class Node {
    String data;
    Node next;
    public Node(String data) {
        this.data = data;
        this.next = null;
    }
}
```

2. **MyLinkedList Class:**
   - This class manages the linked list using a single node reference (`head`) and provides several methods to manipulate the list.
   - **Attributes:**
     - `head` : The starting point of the linked list (points to the first node).
     - `size` : Keeps track of the number of elements in the list.
   - **Constructor:**
     - Initializes the `head` to null and the `size` to 0.

```
class MyLinkedList {
    private Node head;
```

```java
    private int size;

    public MyLinkedList() {
        head = null;
        size = 0;
    }
}
```

Methods in `MyLinkedList` :

1. **add(String s):**
   - Appends the given string `s` at the end of the list.
2. **addFirst(String s):**
   - Inserts the given string `s` at the beginning of the list.
3. **contains(String s):**
   - Checks if the list contains the given string `s` .
4. **getFirst():**
   - Returns the first element of the list.
5. **size():**
   - Returns the size (number of elements) of the list.
6. **remove():**
   - Removes and returns the first element of the list.
7. **removeLast():**
   - Removes and returns the last element of the list.
8. **get(int index):**
   - Returns the element at the specified index.
9. **clear():**
   - Clears all the elements in the list, resetting the list to empty.
10. **findMiddle():**

  - Returns the middle element of the list. Uses the slow and fast pointer technique to find the middle node.

11. **nthFromEnd(int n):**
    - Returns the nth node from the end of the list. It uses two pointers to find the nth node from the end efficiently.
12. **insertAtPosition(int index, String s):**
    - Inserts the given element `s` at the specified position `index` in the list. Shifts existing nodes accordingly.
13. **insertBefore(String target, String s):**
    - Inserts the element `s` before the first occurrence of the specified target element.
14. **deleteAfter(String target):**
    - Deletes the node that appears right after the first occurrence of the specified target element.