OOP Project: Singly Linked List Implementation

Project Overview

Objective:
Implement a custom singly linked list from scratch using object-oriented programming (OOP) principles. The project will support basic operations like adding, removing, searching, and retrieving elements, and it will provide a string representation of the list. This hands-on project is ideal for understanding how classes interact, how attributes encapsulate data, and how methods are defined with specific parameters and return types.

Languages Covered:

- **Java:** Demonstrates strict typing, access modifiers, and class structure.
- **Python:** Highlights dynamic typing, simplicity in class definition, and special methods like `__init__` and `__str__`.

Class Design

1. Class: Node

Purpose:
Represents a single element (or node) in the linked list.

Attributes:

- **data:**
  - *Java:* `String data`
  - *Python:* `data` (can be any type)
    Description: Stores the value of the node.
- **next:**
  - *Java:* `Node next`
  - *Python:* `next` (points to another Node or `None`)
    Description: Holds a reference to the next node in the list.

Methods:

- **Constructor:**
  - *Java:*

    ```
    public Node(String data)
    ```

    - **Arguments:**
      `data` – a `String` value.
    - **Return Type:**
      None (constructors do not return a value).
  - *Python:*

    ```
    def __init__(self, data):
    ```

    - **Arguments:**
      `data` – any value.
    - **Return Type:**
      None.

2. Class: MyLinkedList

Purpose:
Manages the linked list operations using `Node` instances.

Attributes:

- **head:**
  - *Java:* `private Node head;`
  - *Python:* `self.head` (initially `None`)
    Description: Points to the first node in the list.
- **tail:**
  - *Java:* `private Node tail;`
  - *Python:* `self.tail` (initially `None`)
    Description: Points to the last node in the list.
- **size:**

- *Java:* `private int size;`
- *Python:* `self._size` (initially `0`)
  Description: Tracks the number of elements in the list.

Methods:

1. **add / add(String s) (Java) / add(s) (Python)**
   - **Purpose:** Append an element to the end of the list.
   - **Arguments:**
     `s` – the element to add (a `String` in Java; in Python, it can be any type).
   - **Return Type:**
     Java: `void`
     Python: `None`
2. **addFirst / add_first(String s) (Java/Python)**
   - **Purpose:** Insert an element at the beginning of the list.
   - **Arguments:**
     `s` – the element to insert.
   - **Return Type:**
     Java: `void`
     Python: `None`
3. **contains / contains(String s) (Java/Python)**
   - **Purpose:** Check whether the list contains the specified element.
   - **Arguments:**
     `s` – the element to search for.
   - **Return Type:**
     Java: `boolean`
     Python: `bool`
4. **getFirst / get_first() (Java/Python)**
   - **Purpose:** Retrieve the first element in the list.
   - **Arguments:**
     None.
   - **Return Type:**
     Java: `String` (or generic type)
     Python: The data stored in the first node (or `None` if the list is empty).
5. **getLast / get_last() (Java/Python)**

- **Purpose:** Retrieve the last element in the list.
- **Arguments:**
  None.
- **Return Type:**
  Java: `String`
  Python: The data stored in the last node (or `None`).

6. **size() (Java/Python)**

   - **Purpose:** Return the number of elements in the list.
   - **Arguments:**
     None.
   - **Return Type:**
     Java: `int`
     Python: `int`

7. **remove() (Java/Python)**

   - **Purpose:** Remove and return the first element from the list.
   - **Arguments:**
     None.
   - **Return Type:**
     Java: `String`
     Python: The data of the removed node (or `None` if the list is empty).

8. **removeLast() (Java/Python)**

   - **Purpose:** Remove and return the last element from the list.
   - **Arguments:**
     None.
   - **Return Type:**
     Java: `String`
     Python: The data of the removed node.

9. **get(int index) / get(index) (Java/Python)**

   - **Purpose:** Retrieve the element at the specified index.
   - **Arguments:**
     `index` – an integer representing the position.
   - **Return Type:**
     Java: `String`
     Python: The data at that index (or `None` if out-of-bounds).

10. **clear() (Java/Python)**
    - **Purpose:** Remove all elements from the list.
    - **Arguments:**
      None.
    - **Return Type:**
      Java: `void`
      Python: `None`
11. **toString / str (Java/Python)**
    - **Purpose:** Provide a string representation of the list.
    - **Arguments:**
      None.
    - **Return Type:**
      Java: `String`
      Python: `str`
    - **Format:**
      For a non-empty list, each element is wrapped in square brackets (e.g., `[elem1] [elem2]`); if empty, it returns a message such as `"LinkedList is empty"`.