Project Overview: Doubly Linked List

Objective:
Implement a doubly linked list from scratch using OOP principles. In a doubly linked list, every node contains a reference to both the next and previous nodes, allowing bidirectional traversal. This project is designed to help both Java and Python students understand how classes, attributes, and methods work together in an OOP design.

Key OOP Concepts:

- **Classes & Objects:**
  Two main classes are used: a Node (to represent each element) and a DoublyLinkedList (to manage the list).
- **Encapsulation:**
  Internal attributes (like the head, tail, and size) are hidden from the user, who interacts with the list only through its public methods.
- **Abstraction:**
  Users need not worry about how the nodes are linked; they only use methods like `add`, `remove`, and `get`.
- **Modularity:**
  Separating the node functionality from the list management makes the code easier to maintain and extend.

Core Components:

Class: Node (or DoublyNode)

- **Attributes:**
  - **data:**
    Java: `String data`
    Python: `data` (can be any type)
    Description: Stores the element's value.
  - **prev:**
    Java: `Node prev`
    Python: `prev` (points to the previous node or `None`)
    Description: Reference to the previous node.
  - **next:**
    Java: `Node next`
    Python: `next` (points to the next node or `None`)
    Description: Reference to the next node.

- **Constructor:**
  Initializes the node with the provided data.

Class: DoublyLinkedList

- **Attributes:**
  - **head:**
    Points to the first node in the list.
  - **tail:**
    Points to the last node in the list.
  - **size:**
    Keeps track of the number of nodes.
- **Methods:**
  i. **add(String s)** / **add(s):**
  Append an element to the end of the list.
  Return Type: `void` (Java), `None` (Python)
  ii. **addFirst(String s)** / **add_first(s):**
  Insert an element at the beginning.
  Return Type: `void` / `None`
  iii. **contains(String s)** / **contains(s):**
  Check if the list contains the given element.
  Return Type: `boolean` / `bool`
  iv. **getFirst()** / **get_first():**
  Return the first element (or `null` / `None` if empty).
  Return Type: `String` / element type
  v. **getLast()** / **get_last():**
  Return the last element (or `null` / `None`).
  Return Type: `String` / element type
  vi. **size()** / **size():**
  Return the number of elements.
  Return Type: `int`
  vii. **remove()** / **remove():**
  Remove and return the first element.
  Return Type: `String` / element type
  viii. **removeLast()** / **remove_last():**
  Remove and return the last element.
  Return Type: `String` / element type
  ix. **get(int index)** / **get(index):**
  Retrieve the element at a specific index.
  Return Type: `String` / element type

x. **clear()** / **clear():**
Remove all elements from the list.
Return Type: `void` / `None`

xi. **toString()** / **str():**
Return a string representation of the list. For example, a non-empty list might be displayed as `[elem1]<->[elem2]<->[elem3]` while an empty list returns `"DoublyLinkedList is empty"`.