OOP Project: Steque Implementation

Project Overview

A **steque** is a hybrid data structure that combines the properties of a stack and a queue. It supports the following operations:

- **push:** Insert an element at the front (similar to a stack push).
- **pop:** Remove and return the element from the front (like a stack pop).
- **enqueue:** Insert an element at the end (like a queue enqueue).

This data structure is sometimes referred to as an output-restricted deque. The goal of this project is to implement a steque from scratch using object-oriented programming principles. Both Java and Python versions will demonstrate how to design the class with proper encapsulation, clear method definitions (including arguments and return types), and a test suite that validates every operation.

Key OOP Concepts Demonstrated

- **Classes and Objects:**
  The project revolves around a single `Steque` class (with an inner `Node` class) that encapsulates the data and behaviors of the steque.
- **Encapsulation:**
  The internal data (head, tail, and size) is kept private, and interaction with the data structure is performed through public methods.
- **Abstraction:**
  Users work with operations like `push`, `pop`, and `enqueue` without needing to know the underlying node management.
- **Modularity:**
  The implementation separates the node (element) logic from the steque management, making the code easier to maintain and extend.

Class Design

Class: Node

- **Attributes:**
  - **data:**
    - *Java:* `String data`
    - *Python:* `data` (can be any type)

Description: Stores the value.

- **next:**
  - *Java:* `Node next`
  - *Python:* `next` (points to the next node or `None`)
    Description: Reference to the next node in the steque.

- **Constructor:**
  Initializes the node with a given data element.

Class: Steque

- **Attributes:**
  - **head:**
    Points to the first node in the steque.
  - **tail:**
    Points to the last node in the steque.
  - **size:**
    An integer tracking the number of elements.

- **Methods:**

i. **push(s):**

- **Purpose:** Inserts an element at the front.
- **Arguments:** `s` (the element to add).
- **Return Type:** None.

ii. **pop():**

- **Purpose:** Removes and returns the element at the front.
- **Arguments:** None.
- **Return Type:** The removed element (or `null` / `None` if empty).

iii. **enqueue(s):**

- **Purpose:** Appends an element at the end.
- **Arguments:** `s` (the element to add).
- **Return Type:** None.

iv. **size():**

- **Purpose:** Returns the number of elements.

- **Return Type:** Integer.

v. **isEmpty():**

- **Purpose:** Checks if the steque is empty.
- **Return Type:** Boolean.

vi. **toString()** / **str():**

- **Purpose:** Returns a string representation of the steque. For example, non-empty steque elements can be shown as `[elem1][elem2][elem3]`. When empty, it might display `"Steque is empty"`.
- **Return Type:** String.