Project Title: Generic Deque Implementation

Overview

In this project, you will design and implement a generic double-ended queue (deque) data structure. A deque supports the addition and removal of elements at both the front and back ends

Objectives

- **Learn Generics:** Develop a type-safe implementation that can work with any data type.
- **Explore Data Structure Implementation:** Gain experience with both array-based and linked list-based approaches.
- **Handle Edge Cases:** Practice robust exception handling by dealing with invalid operations.
- **Analyze Performance:** Ensure that every operation runs in constant worst-case time.
- **Testing:** Write comprehensive unit tests to validate your implementation.

API Specification

Design your deque with the following operations (the method names and signatures can be adjusted to match the conventions of your chosen programming language):

- **Constructor:**
  - `Deque()`
    Constructs an empty deque.
- **State Check:**
  - `is_empty()`
    Returns whether the deque is empty.
- **Size:**
  - `size()`
    Returns the number of items in the deque.
- **Insertion:**

- `add_first(item)`
  Adds an item to the front of the deque.
- `add_last(item)`
  Adds an item to the back of the deque.

- **Removal:**

  - `remove_first()`
    Removes and returns the item from the front.
  - `remove_last()`
    Removes and returns the item from the back.

- **Iteration:**

  - `iterator()`
    Returns an iterator that yields items in order from front to back.

- **toString:**

  - `toString()`
    Returns a string that iterate all the elements separated by comma space.

- **Testing:**

  - A main testing function or a set of unit tests to demonstrate the correctness and performance of your deque implementation.

Corner Cases and Exception Handling

Ensure your implementation addresses the following cases by raising or throwing appropriate exceptions (adjust the exception types as needed for your language):

- **Null or Invalid Input:**

  - If `add_first()` or `add_last()` is called with a `null` (or invalid) argument, raise an `IllegalArgumentException` (or equivalent).

- **Empty Deque Removal:**

  - If `remove_first()` or `remove_last()` is invoked when the deque is empty, raise a `NoSuchElementException` (or equivalent).

- **Iterator Errors:**

  - The iterator's `next()` (or equivalent) method should raise a `NoSuchElementException` if

    there are no more items to return.
  - If an unsupported operation (like `remove()`) is attempted on the iterator, raise an `UnsupportedOperationException` (or equivalent).

Performance Requirements

- **Constant-Time Operations:**
  Every operation, including construction and iterator methods, must operate in constant worst-case time.
- **Efficient Iteration:**
  The iterator must provide each operation in constant worst-case time.

Evaluation Criteria

- **Correctness:**
  The implementation must adhere to the API and correctly handle all specified edge cases.
- **Efficiency:**
  All operations must run in constant worst-case time.
- **Code Quality:**
  The code should be clean, well-documented, and maintainable.