Project Description: Josephus Problem Using Circular LinkedList

Overview: This project implements a solution to the classic Josephus problem using a doubly circular linked list. In the Josephus problem, people (or nodes) are arranged in a circle and every k-th person is eliminated until only one person remains. The project uses a custom doubly circular linked list to simulate this process and determine the survivor.

Key Components:

1. **ListInterface:**
    - This interface defines a contract for list operations, including:
        - **clear()** – Empties the list.
        - **size()** – Returns the number of nodes in the list.
        - **toString()** – Provides a string representation of the list.
        - **contains(int value)** – Checks if a value exists in the list.
        - **isEmpty()** – Checks if the list is empty.
        - **getData()** – Returns the data from the “current” node.
        - **removeAt(int n)** – Removes (and returns) the node at the given index relative to the current node. The index may be positive or negative, and in case of an invalid index, it throws an exception.
2. **DoublyCircularLL (Doubly Circular Linked List):**
    - This class implements the `ListInterface` using a doubly circular linked list structure.
    - It maintains an inner `Node` class with data, and pointers to the next and previous nodes.
    - A pointer (often called “current”) is used to track a node in the list. The `removeAt` method uses this pointer to remove the node that is a certain number of steps away.
    - In addition to the required interface methods, helper methods (like list initialization) are added to simplify building the list.
3. **Josephus Class:**
    - This class contains the method **josephusDCLL(int size, int rotation)** (or its Python equivalent) that solves the problem.
    - **Parameters:**
        - `size` : The total number of nodes/people in the circle.
        - `rotation` : The elimination order (i.e. every k-th node is removed).

- **Algorithm:**
  - A doubly circular linked list is created and populated with values (typically 1 through `size`).
  - The algorithm repeatedly removes the node that is `(rotation - 1)` steps away from the current node. (For example, with `rotation = 2`, every second node is eliminated.)
  - This elimination continues until only one node remains. The last remaining node's value is returned as the survivor.

4. **Testing:**

   - The provided starter code includes a `main` method (or a main block in Python) that calls the `josephusDCLL` method with several test cases.
   - Additional test cases should be added to ensure robustness. For instance, testing trivial cases (like a list of size 1) or different rotation values (including rotation = 1 or larger skips).