Project Description: Implementing a Stack Using Two Queues

Overview

In this project, you will implement a stack data structure—commonly known for its Last-In-First-Out (LIFO) behavior—by using two queues. While stacks and queues have fundamentally different behaviors, this exercise will deepen your understanding of data structures and how one can be simulated using another.

Objectives

- **Understand the Stack and Queue Concepts:** Learn how a stack (LIFO) differs from a queue (FIFO) and how to reconcile these differences.
- **Data Structures Implementation:** Gain hands-on experience in implementing abstract data types using fundamental collections.
- **Exception Handling:** Properly manage error conditions by using Java's `EmptyStackException` when operations are invalid (e.g., popping or peeking from an empty stack).
- **Algorithmic Thinking:** Decide on an approach where either the push or the pop operation will require extra steps (transferring elements between queues) to simulate the stack behavior.

Functional Requirements

Your task is to create a `MyStack` class that supports the following operations:

1. **empty()**
    - **Description:** Returns `true` if the stack is empty; otherwise, returns `false` .
    - **Behavior:** Must correctly reflect the current state of the stack.
2. **peek()**
    - **Description:** Returns the element at the top of the stack without removing it.
    - **Exception:** Throws an `EmptyStackException` if the stack is empty.
3. **pop()**
    - **Description:** Removes and returns the top element of the stack.

   - **Exception:** Throws an `EmptyStackException` if the stack is empty.

4. **push(E item)**

   - **Description:** Adds an element to the top of the stack.
   - **Return Value:** Should return the element that was pushed.
   - **Note:** Although many implementations of push return `void`, your implementation must return the pushed element to satisfy the provided test cases.

5. **search(Object o)**

   - **Description:** Returns the 1-based position from the top of the stack where the object is located.
   - **Return Value:** If the object is not found, it should return `-1`.
   - **Behavior:** Must handle duplicate elements by returning the position of the top-most occurrence.

Implementation Requirements

- **Use Two Queues:** The `MyStack` class must use two queue instances (e.g., instances of `java.util.LinkedList` or `java.util.ArrayDeque`) to simulate the stack's behavior. You are not allowed to use any built-in stack implementations.
- **Operational Strategy:** You can choose one of the two common strategies:
  - **Making `push` Expensive:** Enqueue the new element and then rotate the elements so that the new element is always at the front.
  - **Making `pop` Expensive:** On a `pop` or `peek` operation, dequeue elements from one queue to the other until only one element remains (which is the top of the stack), then perform the operation.
- **Exception Handling:** Ensure that `peek()` and `pop()` throw an `EmptyStackException` if they are called on an empty stack.

By successfully completing this project, you will not only reinforce your understanding of stacks and queues but also improve your ability to manipulate and combine fundamental data structures in creative ways.

Happy coding!