Project Title: Custom Stack Implementation

Overview

The purpose of this project is to design and implement a Stack data structure that mimics the behavior of the typical Java `Stack` class but is built from scratch. The project is aimed at understanding fundamental data structure concepts such as LIFO (Last-In-First-Out) behavior, exception handling, and the differences between underlying storage mechanisms (dynamic arrays vs. linked lists). Two distinct implementations are provided:

- **ArrayList-based Implementation:** Uses a dynamic array (ArrayList) to manage the elements.
- **LinkedList-based Implementation:** Uses a singly linked list to manage the elements.

Both versions support the same set of operations and are fully tested with a series of test cases covering normal operations and edge cases, including exception scenarios.

Stack Methods Description

Each Stack implementation includes the following methods:

1. **Constructor: `Stack()`**
    - **Purpose:** Creates an empty stack.
    - **Details:** Initializes the underlying data structure (either an ArrayList or a linked list node pointer).
2. **`empty()`**
    - **Purpose:** Tests whether the stack is empty.
    - **Return:** `true` if there are no elements; `false` otherwise.
    - **Usage:** Useful for checking preconditions before performing operations like `pop()` or `peek()`.
3. **`peek()`**
    - **Purpose:** Looks at the object on the top of the stack without removing it.
    - **Return:** The element at the top of the stack.
    - **Exception Handling:** Throws an exception (e.g., `EmptyStackException` in Java or `IndexError` in Python) if the stack is empty.
4. **`pop()`**

- **Purpose:** Removes the object at the top of the stack and returns it.
- **Return:** The removed element.
- **Exception Handling:** Throws an exception if the stack is empty.

5. `push(String item)`
    - **Purpose:** Pushes an item onto the top of the stack.
    - **Return:** The item that was pushed.
    - **Details:** Adds a new element to the end (or head, depending on the implementation) of the underlying data structure.
6. `search(String o)`
    - **Purpose:** Returns the 1-based position of the object from the top of the stack.
    - **Return:** The position (with the top element being position 1) if found, or -1 if the element is not present.
    - **Details:** Iterates through the underlying structure starting from the top.

Implementation 1: ArrayList-based Stack (Student Created)

Design & Approach

- **Underlying Structure:** The stack is implemented using an `ArrayList<String>` (or a similar dynamic array structure in other languages).
- **Key Points:**
    - **Push Operation:** Simply appends the new element at the end of the ArrayList.
    - **Pop Operation:** Removes the last element (the top element) from the ArrayList.
    - **Search Operation:** Iterates backward from the end of the ArrayList to find the element, returning its 1-based position.