Project Title: Balanced Parentheses Checker – The Code-Breaker's Challenge

Overview

Imagine you're a secret code breaker working on a critical mission. You intercept mysterious messages where parentheses represent crucial commands. To decode these messages correctly, every opening parenthesis must have a matching closing one. Your challenge is to create a function, **hasBalancedParentheses**, that verifies if the sequence of parentheses in any given message is properly balanced. The solution must be independent of any specific programming language, allowing you to implement it in either Java or Python.

Story

In a high-stakes scenario, a secret agency intercepts an encrypted message. The message contains not only letters and numbers but also a series of parentheses that encapsulate hidden instructions. If these parentheses are not correctly balanced, the message is deemed corrupted, and the mission could be compromised. As an elite code breaker, your task is to ensure that every "(" has a matching ")" in the proper order. By doing so, you will guarantee that the secret instructions can be safely executed.

Project Objective

Develop a function **hasBalancedParentheses** that:

- Accepts a single string input containing a mix of characters.
- Ignores all non-parentheses characters.
- Returns **True** if the parentheses in the string are balanced.
- Returns **False** if they are not.

Constraints

- **Do Not Use Built-in Data Structures:** Avoid any built-in stack or similar libraries. You are allowed (and encouraged) to design your own version of a stack to manage the parentheses.
- **Language Agnostic:** The solution should be implementable in any programming language (Java, Python, etc.), so avoid language-specific features that cannot be ported.

Input/Output Format

- **Input:** A string that may include various characters, including the parentheses "(" and ")".
- **Output:** A boolean value:
    - **True** if every opening parenthesis has a corresponding closing parenthesis in the correct

    order.
  - **False** otherwise.

Test Case Explanation

Test Case:

- **Input:** `"((()())())"`
- **Expected Output:** `True`

Step-by-Step Explanation:

1. **Initialization:** Start with an empty stack (using your custom stack implementation).
2. **Processing Characters:**
   - Traverse each character in the string.
   - When an opening parenthesis "(" is encountered, push it onto the stack.
   - When a closing parenthesis ")" is encountered, check if the stack is not empty. If it is, then there is no corresponding opening parenthesis, and the function should return **False**.
   - Otherwise, pop the top element from the stack.
3. **Final Check:** After processing the entire string, check the stack:
   - If the stack is empty, all parentheses were matched correctly (balanced).
   - If the stack is not empty, some opening parentheses were left unmatched (imbalanced).

Since every "(" in `"((()())())"` is paired with a matching ")", the final output is **True**.