/\* 2. Check for balanced parentheses using a stack.

• Test Case 1:

Input: "({[()]})"

Output: Balanced

• Test Case 2:

Input: "([)]"

Output: Not Balanced \*/

import java.util.Scanner;

public class question2 {

private char[] stack;

private int top;

private int maxSize;

// Constructor to initialize the stack

public question2(int size) {

stack = new char[size];

maxSize = size;

top = -1;

}

// Push method

public void push(char c) {

if (top < maxSize - 1) {

stack[++top] = c;

}

}

// Pop method

public char pop() {

if (top >= 0) {

return stack[top--];

}

return '\0'; // Null character for an empty stack

}

// Check if the stack is empty

public boolean isEmpty() {

return top == -1;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Take input from user

System.out.print("Enter parentheses string: ");

String expression = scanner.nextLine();

// Check if the expression is balanced

boolean result = isBalanced(expression);

System.out.println("Input: \"" + expression + "\"");

System.out.println("Output: " + (result ? "Balanced" : "Not Balanced"));

}

public static boolean isBalanced(String expression) {

question2 stack = new question2(expression.length());

for (int i = 0; i < expression.length(); i++) {

char ch = expression.charAt(i);

// Push opening brackets onto stack

if (ch == '(' || ch == '{' || ch == '[') {

stack.push(ch);

}

// Check closing brackets

else if (ch == ')' || ch == '}' || ch == ']') {

if (stack.isEmpty()) {

return false;

}

char top = stack.pop();

if (!((top == '(' && ch == ')') || (top == '{' && ch == '}') || (top == '[' && ch == ']'))) {

return false;

}

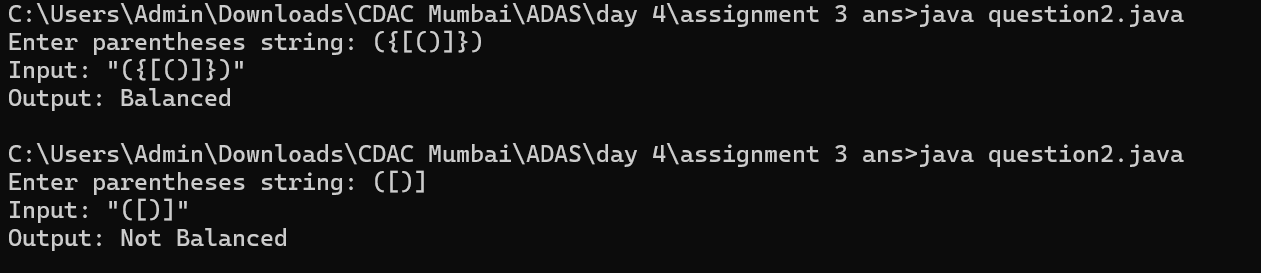
}

}

return stack.isEmpty();

}

}

  
  
  
explanation  
  
time and space complexity is 0(n)  
  
The code checks if a given string of parentheses is balanced using a custom stack implementation. It initializes an array-based stack to keep track of opening brackets. For each character in the input string, opening brackets are pushed onto the stack, while closing brackets are checked against the top of the stack to ensure they match. If a mismatch is found or if the stack is empty when a closing bracket is encountered, the string is declared unbalanced. After processing all characters, if the stack is empty, the string is balanced. The time complexity is O(n), as it processes each character once, and the space complexity is O(n), due to the stack storing up to n characters in the worst case