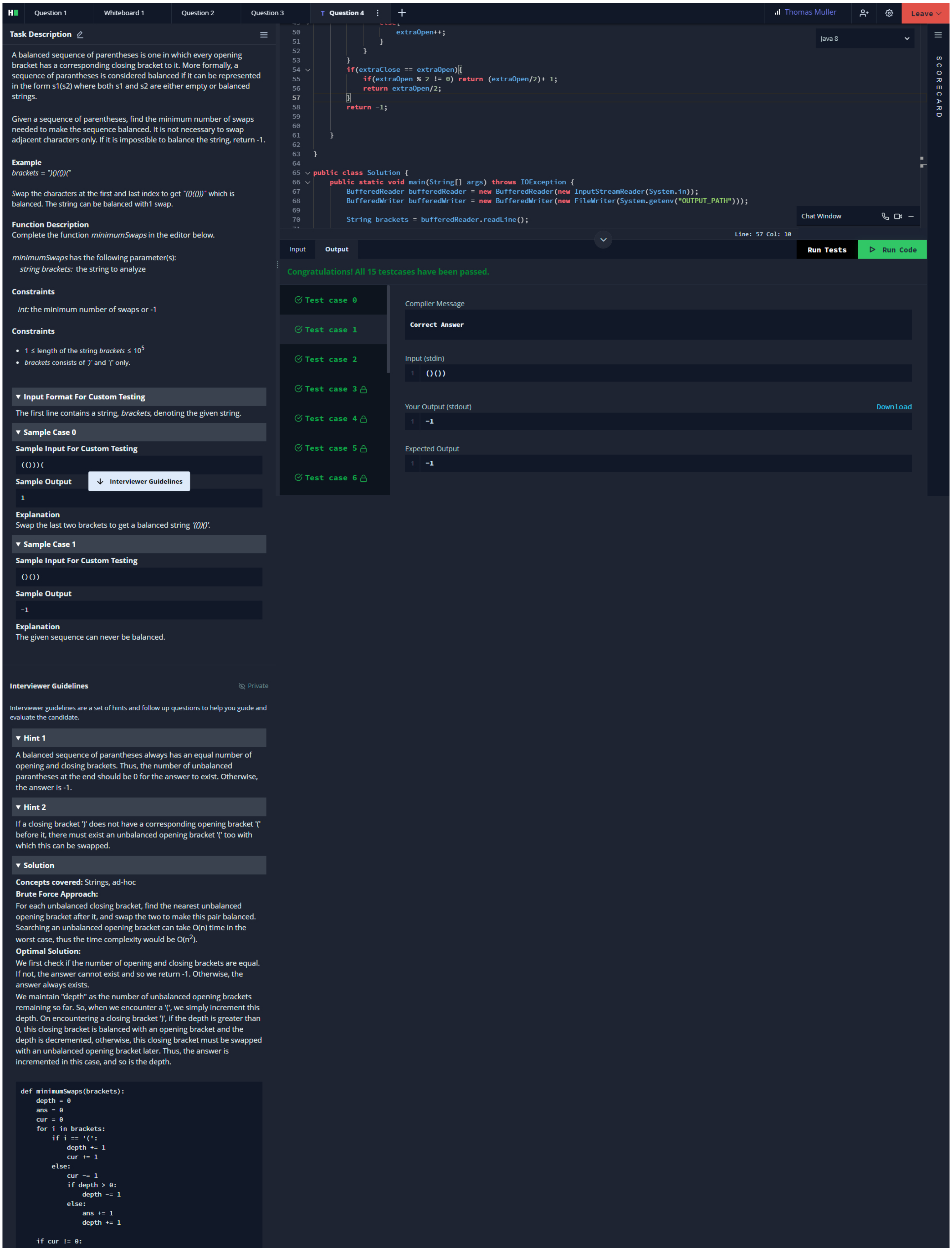

Task Description

A balanced sequence of parentheses is one in which every opening bracket has a corresponding closing bracket to it. More formally, a sequence of parentheses is considered balanced if it can be represented in the form s1(s2) where both s1 and s2 are either empty or balanced strings.

Given a sequence of parentheses, find the minimum number of swaps needed to make the sequence balanced. It is not necessary to swap adjacent characters only. If it is impossible to balance the string, return -1.

Example
brackets = ")()(()("

Swap the characters at the first and last index to get "(()(()))" which is balanced. The string can be balanced with1 swap.

Function Description
Complete the function *minimumSwaps* in the editor below.

minimumSwaps has the following parameter(s):
  string brackets: the string to analyze

Constraints

int: the minimum number of swaps or -1

Constraints

- 1 ≤ length of the string *brackets* ≤ 10^5
- *brackets* consists of ')' and '(' only.

▼ Input Format For Custom Testing

The first line contains a string, *brackets,* denoting the given string.

▼ Sample Case 0

Sample Input For Custom Testing

```
(()))(
```

Sample Output

```
1
```

Explanation
Swap the last two brackets to get a balanced string *'(())()'*.

▼ Sample Case 1

Sample Input For Custom Testing

```
()())
```

Sample Output

```
-1
```

Explanation
The given sequence can never be balanced.

Interviewer Guidelines

Interviewer guidelines are a set of hints and follow up questions to help you guide and evaluate the candidate.

▼ Hint 1

A balanced sequence of parantheses always has an equal number of opening and closing brackets. Thus, the number of unbalanced parantheses at the end should be 0 for the answer to exist. Otherwise, the answer is -1.

▼ Hint 2

If a closing bracket ')' does not have a corresponding opening bracket '(' before it, there must exist an unbalanced opening bracket '(' too with which this can be swapped.

▼ Solution

Concepts covered: Strings, ad-hoc
Brute Force Approach:
For each unbalanced closing bracket, find the nearest unbalanced opening bracket after it, and swap the two to make this pair balanced. Searching an unbalanced opening bracket can take O(n) time in the worst case, thus the time complexity would be O(n^2).
Optimal Solution:
We first check if the number of opening and closing brackets are equal. If not, the answer cannot exist and so we return -1. Otherwise, the answer always exists.
We maintain "depth" as the number of unbalanced opening brackets remaining so far. So, when we encounter a '(', we simply increment this depth. On encountering a closing bracket ')', if the depth is greater than 0, this closing bracket is balanced with an opening bracket and the depth is decremented, otherwise, this closing bracket must be swapped with an unbalanced opening bracket later. Thus, the answer is incremented in this case, and so is the depth.

```
def minimumSwaps(brackets):
    depth = 0
    ans = 0
    cur = 0
    for i in brackets:
        if i == '(':
            depth += 1
            cur += 1
        else:
            cur -= 1
            if depth > 0:
                depth -= 1
            else:
                ans += 1
                depth += 1

    if cur != 0:
```

```
                extraOpen++;
            }
        }
    }
    if(extraClose == extraOpen){
        if(extraOpen % 2 != 0) return (extraOpen/2)+ 1;
        return extraOpen/2;
    }
    return -1;


  }

}

public class Solution {
    public static void main(String[] args) throws IOException {
        BufferedReader bufferedReader = new BufferedReader(new InputStreamReader(System.in));
        BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter(System.getenv("OUTPUT_PATH")));

        String brackets = bufferedReader.readLine();
```

Line: 57 Col: 10

Congratulations! All 15 testcases have been passed.

Compiler Message

```
Correct Answer
```

Input (stdin)

```
()())
```

Your Output (stdout)

```
-1
```

Expected Output

```
-1
```

```
        return -1

    return ans
```

▼ **Complexity Analysis**

Time Complexity - O(n)
We iterate over the given sequence of brackets once, performing simple if-checks, thus the time complexity is O(n).
Space Complexity - O(1) - No extra space is required.
Since we only maintain three variables: to track the depth, answer and the total number of opening and closing brackets, no extra space is required, and thus auxilliary space required is of the order O(1).