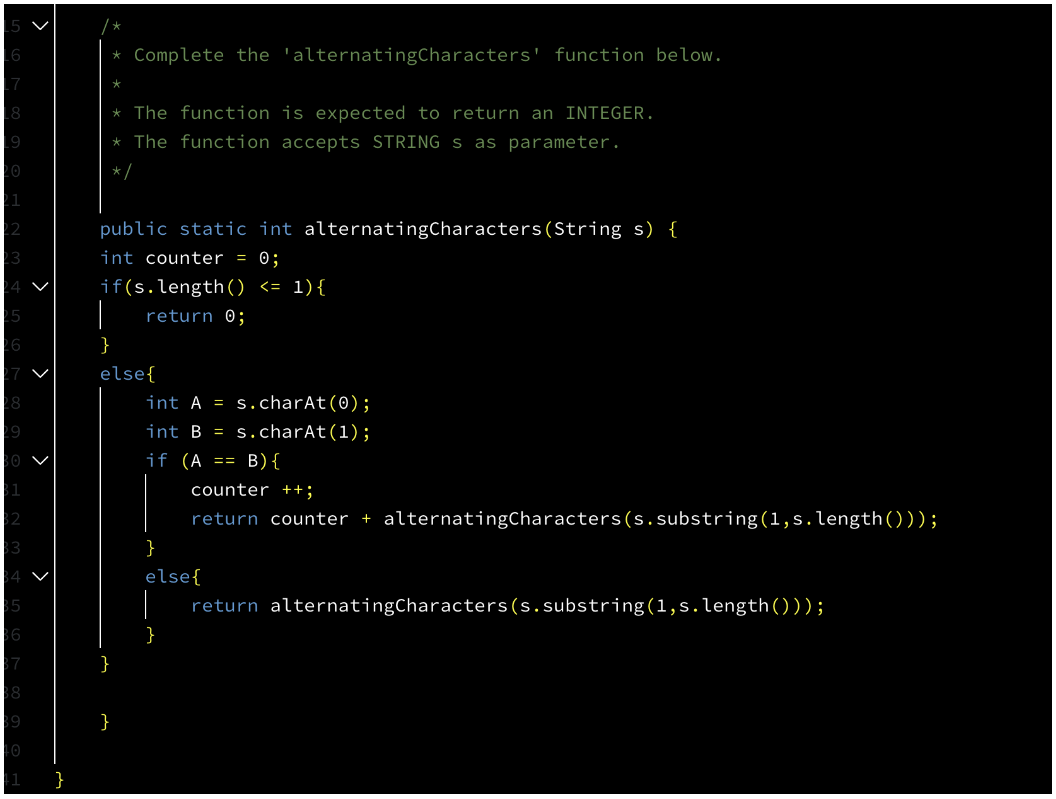
Alternating Characters

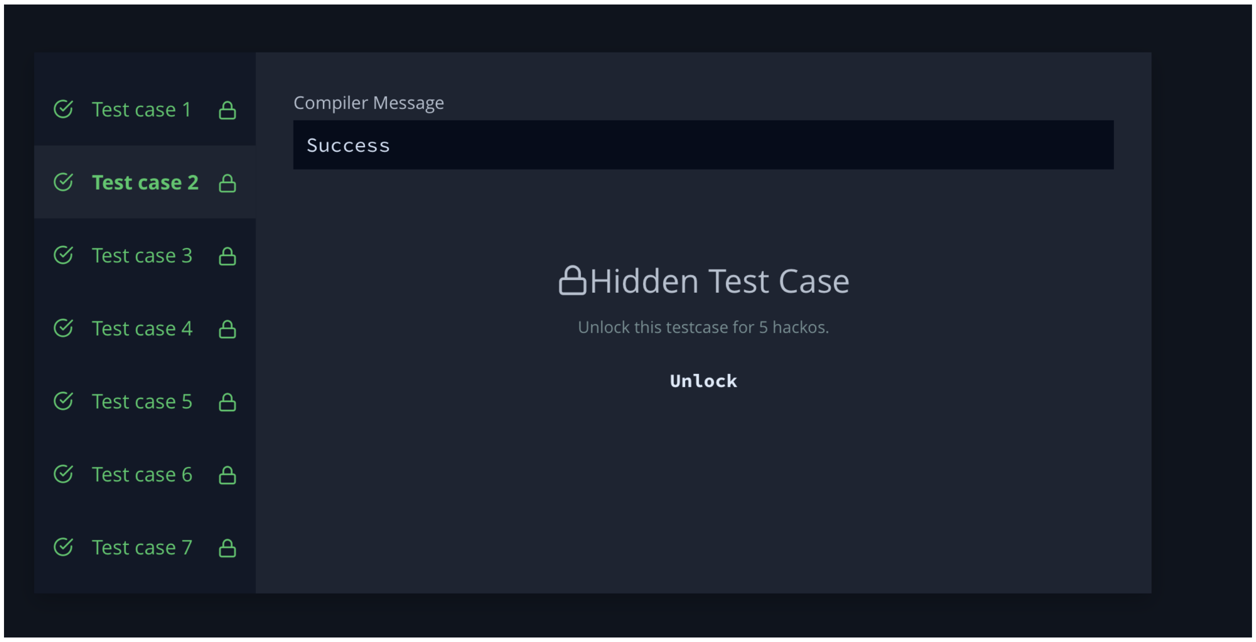


```
    /*
     * Complete the 'alternatingCharacters' function below.
     *
     * The function is expected to return an INTEGER.
     * The function accepts STRING s as parameter.
     */

    public static int alternatingCharacters(String s) {
    int counter = 0;
    if(s.length() <= 1){
        return 0;
    }
    else{
        int A = s.charAt(0);
        int B = s.charAt(1);
        if (A == B){
            counter ++;
            return counter + alternatingCharacters(s.substring(1,s.length()));
        }
        else{
            return alternatingCharacters(s.substring(1,s.length()));
        }
    }

    }

}
```



Compiler Message

```
Success
```

Hidden Test Case

Unlock this testcase for 5 hackos.

Time and Space complexity for Alternating Characters problem:
Time Complexity: Since the Alternating Characters is recursive, it calls back the number of string characters, ‘n’. Since the string has **n** amount of characters it will be called **n** amount of times therefore the time complexity for this code is O(n).
Space complexity: Since each recursive call calls the string n number of times, then the space being used each time is **n** number of times, each time there is one less character in n, however due to the recursive nature of the code it still use O(n) as memory, therefore the space complexity of O(n).

Recursive definition:
The base case for the code is if the number of strings is less than one than you return 0 since there are not sufficient numbers of strings to rotate them.
The recursive call itself happens when A == B, when two or more characters in a row are the same then a deletion is required, i.e there is a counter for deletions as well, as for each deletion that occurs the function recursively calls it back until there is one or less number of strings. It recursively calls back the characters with the deleted character until it satisfies the base case with only one character remaining, if A and B are different then no deletion is required and they move onto the next character. With both calls it always starts at the second character, i.e substring(1), if its the same deletion and increment of the counter when recursively calling it back, if not then recursively call it back without incrementing the counter and the deletion.

Footnote* When working on it i noticed that sometimes due to it being a recursive call, the grading times it out because it takes longer time to recursively call each character.