GTU Department of Computer Engineering
CSE 222/505 - Spring 2023
Homework #04 Report

Selim Aynigül
200104004004

Time Complexity Analysis

boolean checkIfValidUsername(String username) **Time Complexity:** O(n)

Description: The time complexity of this code is O(n), where n is the length of the input string username. Since the function is called recursively n times (once for each character in the string), the total time complexity is O(n).

boolean containsUserNameSpirit(String password1, String username) **Time Complexity:** O(n*m)

Description: The time complexity of this code is O(n*m), where n is the length of the input password1 and m is the length of the input username. Since the function uses nested loops with the length of the each input, the total time complexity is the multiplication of the lengths.

boolean isBalancedPassword(String password1) **Time Complexity:** O(n)

Description: The time complexity of this code is O(n), where n is the length of the input string password1. Since the function uses one for loop to iterate through the characters of the input n times the total time complexity is O(n).

boolean isPalindromePossible(String password1) **Time Complexity:** O(n!)

Description: The time complexity of this code is O(n!), where n is the length of the input string password1. Since we create all the possible permutations of given string in worst case the time complexity is O(n!).

boolean isExactDivision(int password2, int[] denominations) **Time Complexity:** O(2^n)

Description: The time complexity of this code is exponential, O(2^n), where n is the length of the denominations array. This is because the function isExactDivisionHelper calls itself recursively for each denomination that is less than or equal to the target.

Other Methods: Extra methods to check other validation conditions: In password1 class checkCharacters and checkBrackets methods have O(n) time complexity. They iterate through input strings with one for loop. In password1 and password2 classes we use checkSize methods with O(1) time complexity. They just have an if else statement to check size.

Class Structure

In my design we have 6 different classes.

Main and ***TestClass*** to test our methods. In main function we are creating our Officer objects with different input values and sending them to tryToEnter() method in the TestClass to check validation of the inputs. To do this TestClass calls static validation methods from username and password classes.

Officer class to create new officer objects with username, password1, and password2 values. (Constructor assigns these values as inputs.)

Username, ***Password1***, and ***Password2*** classes. Each of them has their own static methods to check different validation conditions.

If we go over all, in main method we create our officers with their username and passwords and send these objects as parameter to tryToEnter() method which belongs to TestClass class. In here tryToEnter() method calls all of the validation methods from username and password classes respectively and if all conditions are met then entrance attempt is considered successful.

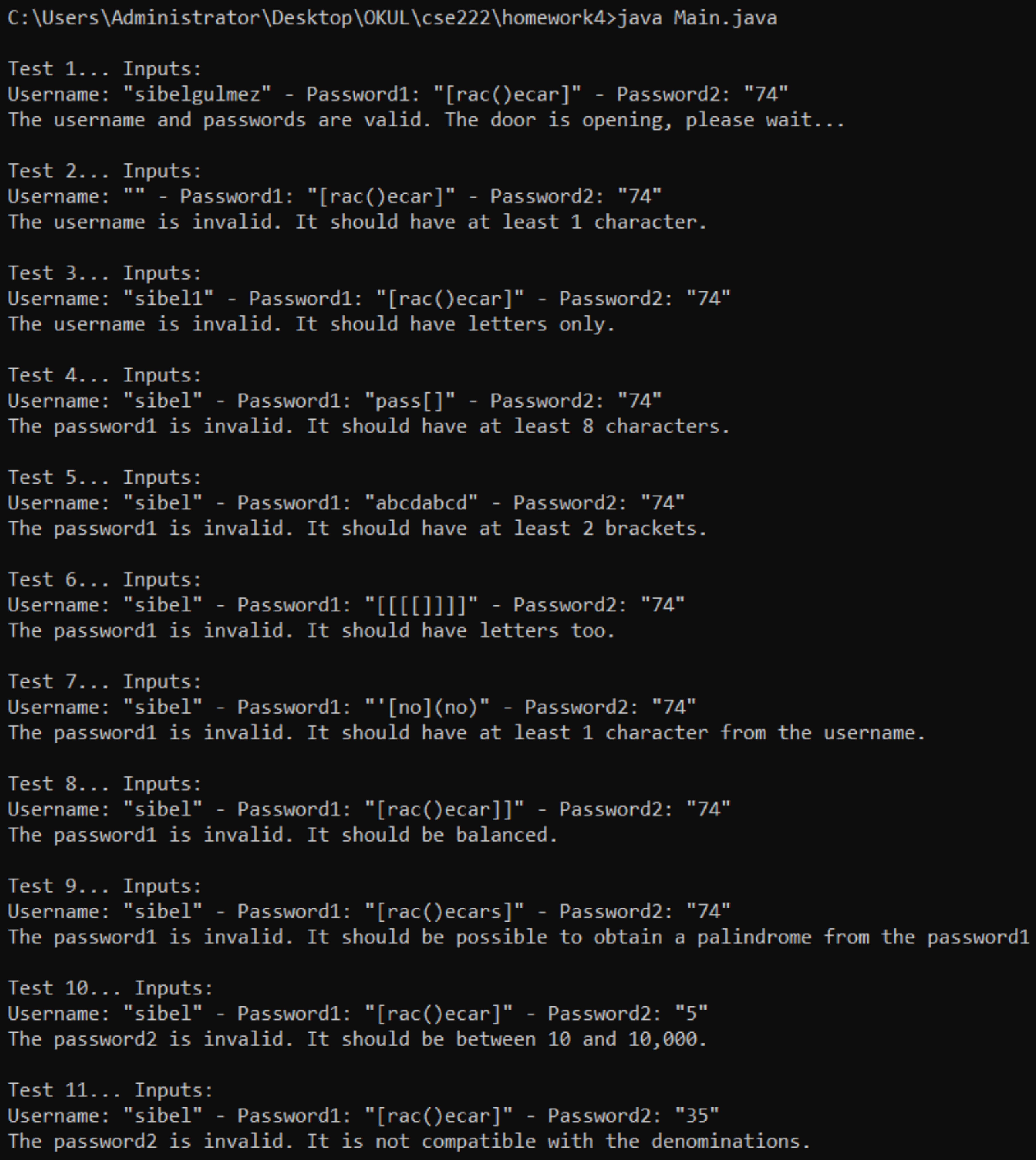
Running Command and Results

Compiling commands:

cd homework4
javac *.java
cd..

Running commands:

java homework4.Main



```
C:\Users\Administrator\Desktop\OKUL\cse222\homework4>java Main.java

Test 1... Inputs:
Username: "sibelgulmez" - Password1: "[rac()ecar]" - Password2: "74"
The username and passwords are valid. The door is opening, please wait...

Test 2... Inputs:
Username: "" - Password1: "[rac()ecar]" - Password2: "74"
The username is invalid. It should have at least 1 character.

Test 3... Inputs:
Username: "sibel1" - Password1: "[rac()ecar]" - Password2: "74"
The username is invalid. It should have letters only.

Test 4... Inputs:
Username: "sibel" - Password1: "pass[]" - Password2: "74"
The password1 is invalid. It should have at least 8 characters.

Test 5... Inputs:
Username: "sibel" - Password1: "abcdabcd" - Password2: "74"
The password1 is invalid. It should have at least 2 brackets.

Test 6... Inputs:
Username: "sibel" - Password1: "[[[[]]]]" - Password2: "74"
The password1 is invalid. It should have letters too.

Test 7... Inputs:
Username: "sibel" - Password1: "'[no](no)" - Password2: "74"
The password1 is invalid. It should have at least 1 character from the username.

Test 8... Inputs:
Username: "sibel" - Password1: "[rac()ecar]]" - Password2: "74"
The password1 is invalid. It should be balanced.

Test 9... Inputs:
Username: "sibel" - Password1: "[rac()ecars]" - Password2: "74"
The password1 is invalid. It should be possible to obtain a palindrome from the password1

Test 10... Inputs:
Username: "sibel" - Password1: "[rac()ecar]" - Password2: "5"
The password2 is invalid. It should be between 10 and 10,000.

Test 11... Inputs:
Username: "sibel" - Password1: "[rac()ecar]" - Password2: "35"
The password2 is invalid. It is not compatible with the denominations.
```