2023 Spring CSE222 Homework4 Report

In this assignment, I first created 3 different classes. These classes are Username, Password1, Passwor2. I created the desired functions in the homework under the relevant class. I created a test function to test the code in the Main class.

Functions:

checkIfValidUsername: This is a function that checks if all characters are letters, to check that the username is valid. This check is performed recursively, starting by creating a new Username object to handle the rest.

containsUserNameSpirit: To check the user name in the password, it is checked whether each character of the password is in the username. Stack data structure is used.

isBalancedPassword: When checking whether the password contains balanced parentheses, the removeLetters function is used, which removes letters and creates a string containing only parentheses. Then a stack is used to check for balanced parentheses.

isPalindromePossible: First, the removeBrackets function is used, which clears the parenthesis of the password. Then a recursive function is used that counts the frequency of each character and checks whether a palindrome can be formed.

isExactDivision: A recursive helper function is used to check if the second password is exactly divisible by the specified numbers. This function checks whether the password is divisible by the current number and then moves on to the next number. The function continues recursively until it checks whether the password is divisible by all numbers.

Running Command and Test

```
System.out.println("SCENARIO 1");
Username user1 = new Username("gokhan123");
Password1 pass1_1 = new Password1("{[aaddaa]}");
Password2 pass2_1 = new Password2(75);
```

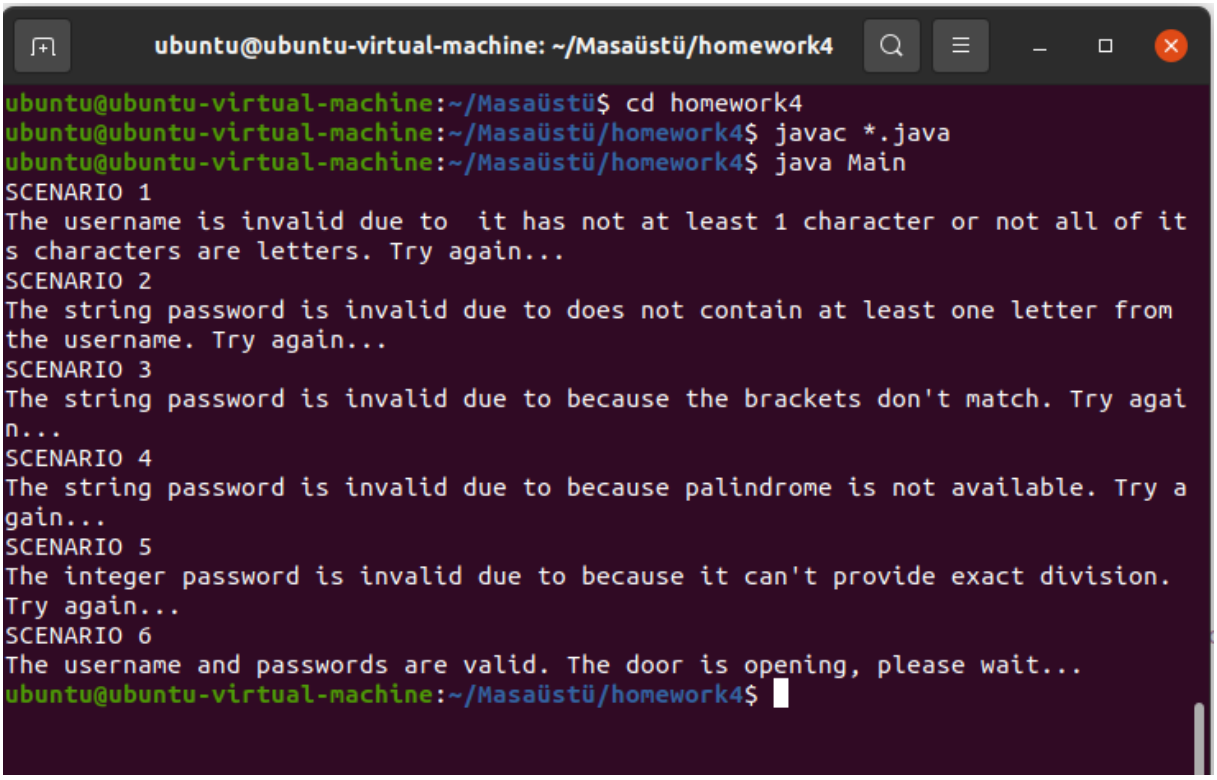
```
System.out.println("SCENARIO 2");
Username user2 = new Username("elif");
Password1 pass1_2 = new Password1("{[xy(xyxyx)]}");
Password2 pass2_2 = new Password2(56);
```

```
System.out.println("SCENARIO 3");
Username user3 = new Username("ahmet");
Password1 pass1_3 = new Password1("]{[abc(abc)abc]}");
Password2 pass2_3 = new Password2(88);
```

```
System.out.println("SCENARIO 4");
Username user4 = new Username("mehmet");
Password1 pass1_4 = new Password1("{[(wacaceacay)]}");
Password2 pass2_4 = new Password2(68);
```

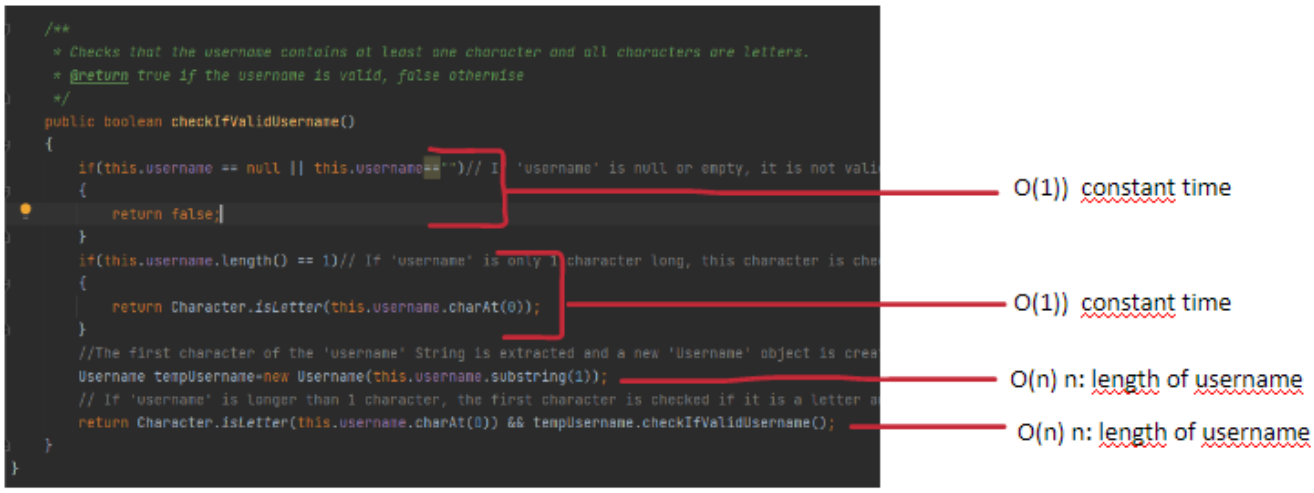
```
System.out.println("SCENARIO 5");
Username user5 = new Username("ayse");
Password1 pass1_5 = new Password1("{[xyy]x}");
Password2 pass2_5 = new Password2(35);
```

```
System.out.println("SCENARIO 6");
Username user6 = new Username("zeynep");
Password1 pass1_6 = new Password1("{[(ecarcar)]}");
Password2 pass2_6 = new Password2(75);
```


ubuntu@ubuntu-virtual-machine: ~/Masaüstü/homework4
ubuntu@ubuntu-virtual-machine:~/Masaüstü$ cd homework4
ubuntu@ubuntu-virtual-machine:~/Masaüstü/homework4$ javac *.java
ubuntu@ubuntu-virtual-machine:~/Masaüstü/homework4$ java Main
SCENARIO 1
The username is invalid due to it has not at least 1 character or not all of it
s characters are letters. Try again...
SCENARIO 2
The string password is invalid due to does not contain at least one letter from
the username. Try again...
SCENARIO 3
The string password is invalid due to because the brackets don't match. Try agai
n...
SCENARIO 4
The string password is invalid due to because palindrome is not available. Try a
gain...
SCENARIO 5
The integer password is invalid due to because it can't provide exact division.
Try again...
SCENARIO 6
The username and passwords are valid. The door is opening, please wait...
ubuntu@ubuntu-virtual-machine:~/Masaüstü/homework4$

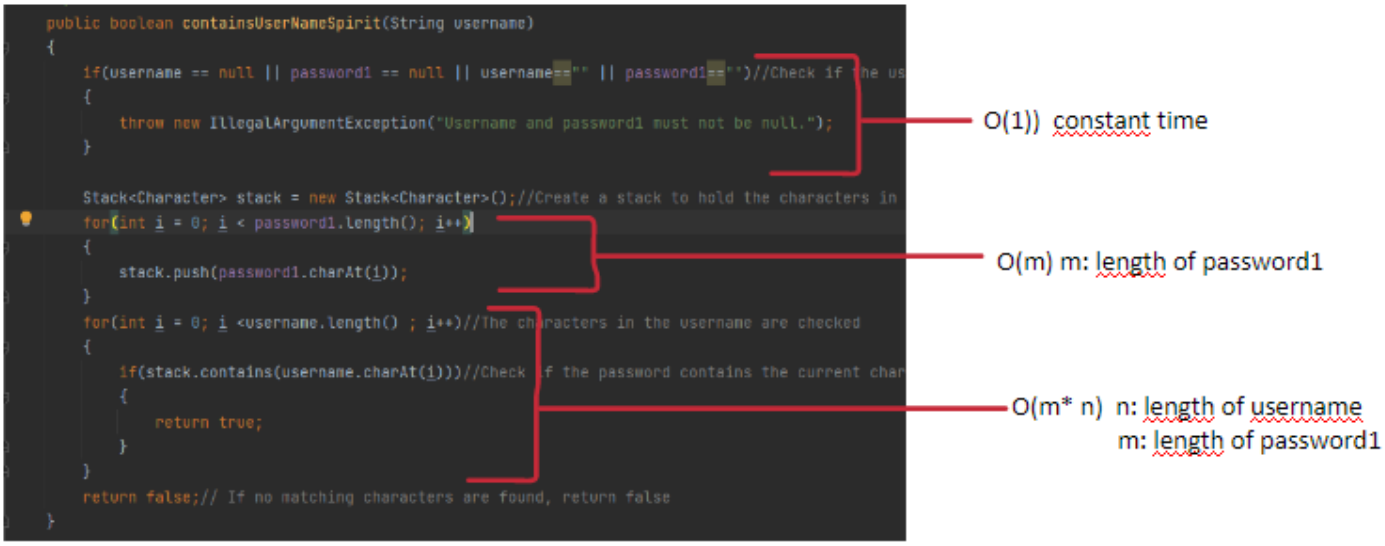
Time Complexity

1. [A Recursive Function] boolean checkIfValidUsername(String username):



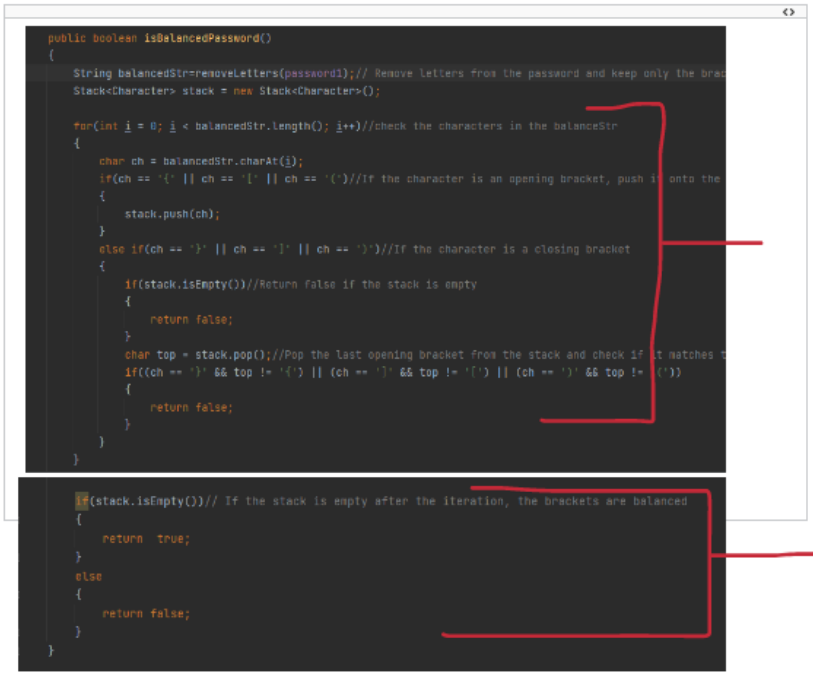
The total time complexity: O(1) + O(1) + O(n) + O(n) = O(2n) + O(2) = O(n)

2. [A Stack Function] boolean containsUserNameSpirit(String username, String password1):



Total time complexity: O(1) + O(m) + O(m*n) = O(m*n)

3. [A Stack Function] boolean isBalancedPassword(String password1):



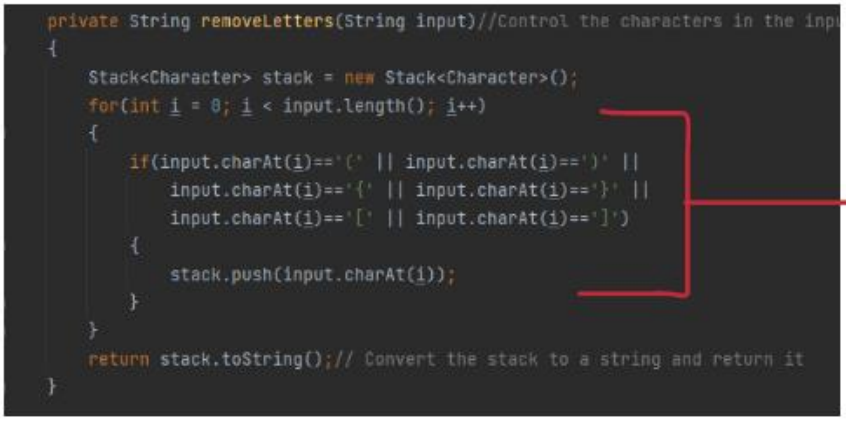
```
public boolean isBalancedPassword()
{
    String balancedStr=removeLetters(password1);// Remove letters from the password and keep only the brac
    Stack<Character> stack = new Stack<Character>();

    for(int i = 0; i < balancedStr.length(); i++)//check the characters in the balanceStr
    {
        char ch = balancedStr.charAt(i);
        if(ch == '{' || ch == '[' || ch == '('))//If the character is an opening bracket, push it onto the
        {
            stack.push(ch);
        }
        else if(ch == '}' || ch == ']' || ch == ')')//If the character is a closing bracket
        {
            if(stack.isEmpty())//Return false if the stack is empty
            {
                return false;
            }
            char top = stack.pop();//Pop the last opening bracket from the stack and check if it matches t
            if((ch == '}' && top != '{') || (ch == ']' && top != '[') || (ch == ')' && top != '('))
            {
                return false;
            }
        }
    }
```

O(n) n: length of password1
(The length of balancedStr can be up to the length of password1)


```
    if(stack.isEmpty())// If the stack is empty after the iteration, the brackets are balanced
    {
        return  true;
    }
    else
    {
        return false;
    }
}
```

O(1)) constant time

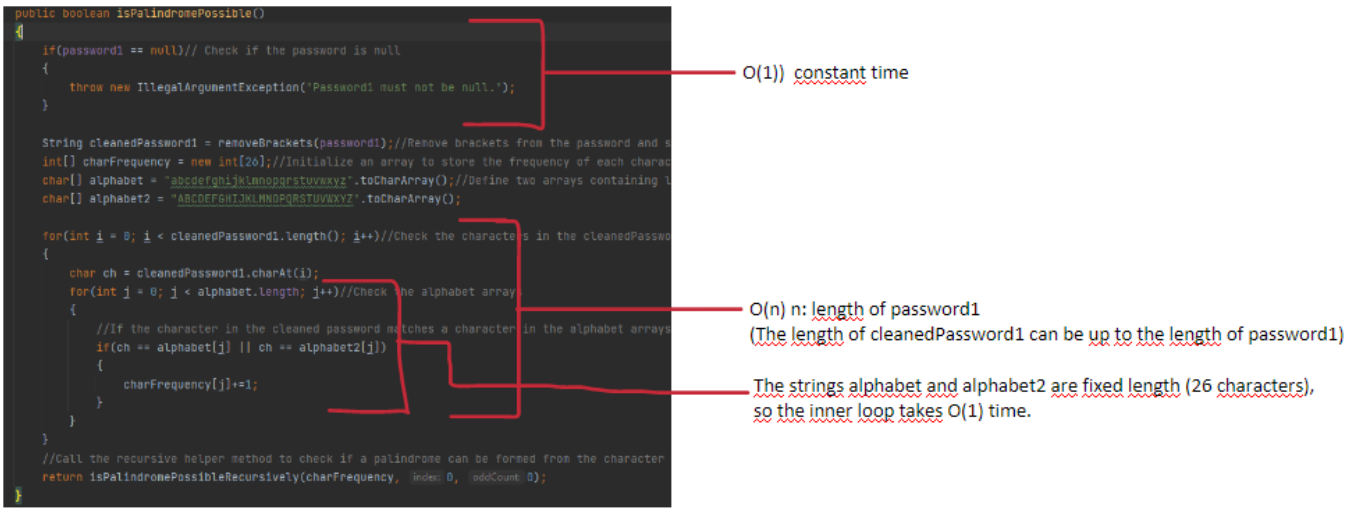


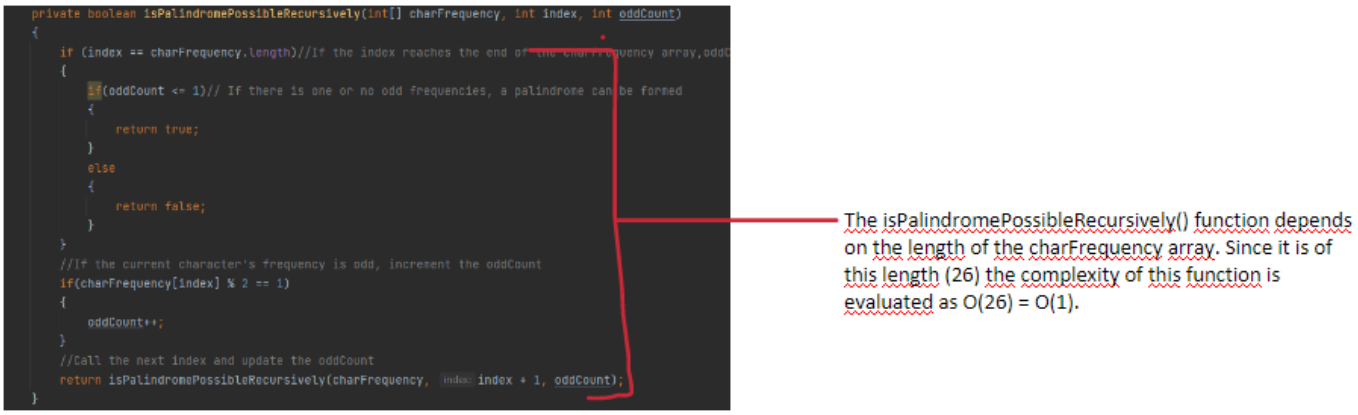
```
private String removeLetters(String input)//Control the characters in the inpu
{
    Stack<Character> stack = new Stack<Character>();
    for(int i = 0; i < input.length(); i++)
    {
        if(input.charAt(i)=='(' || input.charAt(i)==')' ||
            input.charAt(i)=='{' || input.charAt(i)=='}' ||
            input.charAt(i)=='[' || input.charAt(i)==']')
        {
            stack.push(input.charAt(i));
        }
    }
    return stack.toString();// Convert the stack to a string and return it
}
```

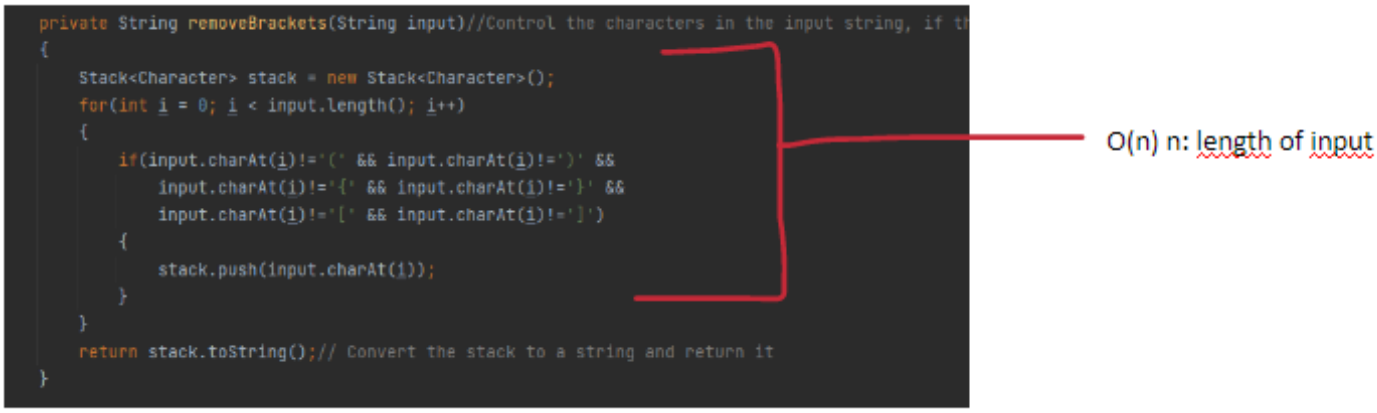
O(n) n: length of input

The total time complexity: O(n) + O(n) + O(1) = O(2n). Deleted constant coefficients => O(n)

4. [A Recursive Function] boolean isPalindromePossible(String password1):







Total time complexity: O(1) + O(n) + O(n) + O(1) = O(2n). Deleted constant coefficients => O(n)

5. [A Recursive Function] boolean isExactDivision(int password2, int [] denominations):

```java
public boolean isExactDivision(int[] denominations)
{
    return isExactDivisionHelper(password2, denominations, index: 0);
}
```

```java
private boolean isExactDivisionHelper(int password2, int[] denominations, int index)
{
    if(password2 == 0)
    {
        return true;
    }
    if(index >= denominations.length || password2 < 0)
    {
        return false;
    }
    // Check if the current denomination can be used to achieve exact division
    boolean divisonFlag = isExactDivisionHelper( password2: password2 - denominations[index]

    if(divisonFlag)
    {
        return true;
    }
    // Call to the next denomination
    return isExactDivisionHelper(password2, denominations, index: index + 1);
}
```

This code has time complexity depending on the size of the given array of denominations. The function completes only in two fixed cases, while in the other case it calls itself and the previous step. Therefore, the worst-case time complexity of the code is O(2^n) because a recursive call is made when the entire array of denominations is equal to password2.

Time complexity: O(2^n)