CSE 222/505 - Spring 2023
Homework 4 – Report
Erkut Dere – 1801042646

Time complexity analysis of each method:

```
public boolean checkIfValidUsername(String username)
```

This method's time complexity depends on the length of the username.(Let's say that length of the username is "n"). If the first character is character the method calls the same method with giving the rest of the string(n-1) recursively. That's why time complexity of this method must be O(n).
Also if string is null or empty string or string does not contains any character or string contains any number character; in these kind of situations this method works in O(1) time complexity.

```
public boolean containsUserNameSpirit(String username, String password)
```

This method's contains two consecutive for loop; first for loop spends time until reach to password length(Let's say that "n"). Second for loop reach at the end until index is equal to username's length(let's say that "m"). This two consecutive for loops decides this method's time complexity other constant operations takes constant time. That's why time complexity of this method is O(n+m).

```
public boolean isBalancedPassword(String Password1)
```

The first for loop iterates through password1 and pushes the brackets in stack(Let's say password1 length is "n"). That's why first loop's takes O(n) time. The while loop checks each bracket and if it founds it pair(closed bracket) in the stack it removes both of them, so stack size decreases by twos. So this loop takes O(n/2) time. Inside this loop remove operations from stack and other simple print statements takes O(1) time so it does not effect overall time complexity.
At the end if we look at this two loop first one takes O(n) time and the second one takes O(n/2) so we can say that this method's time complexity is equal to summation of this two loops time complexities that is equal to O(n).

```
public boolean isPalindromePossible(String password1)
```

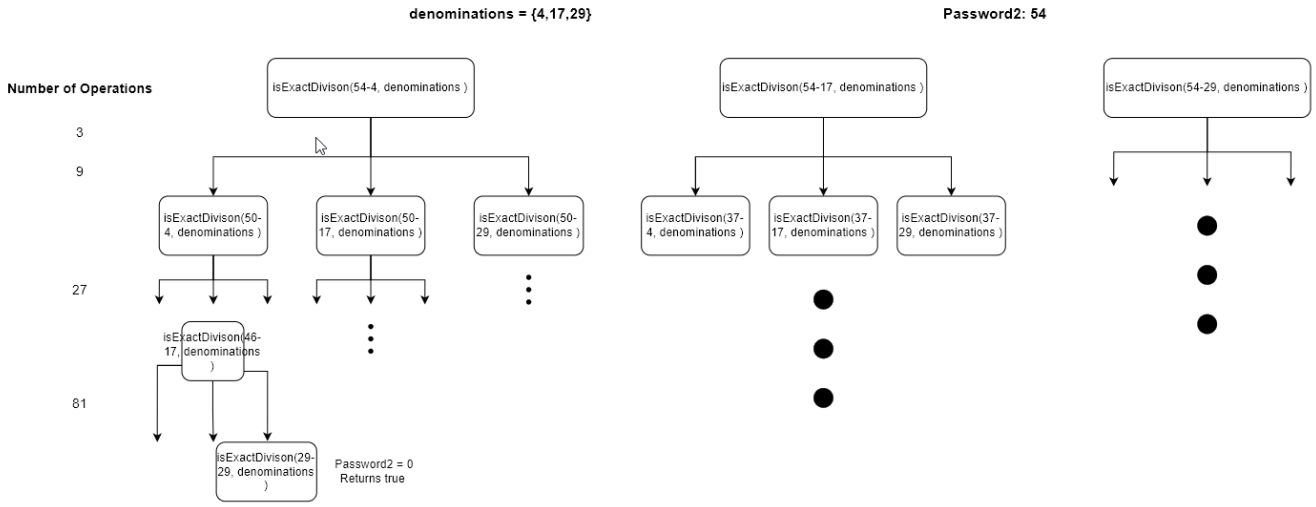
This method checks every character's presence in password1 by counting the number of them to check overall password is palindrome or not. It increments every character's

index by one in count array where it is fixed size. This takes O(1) time. After that it calls the rest of the string by calling the same function recursively. That's why this method's time complexity depends on the length of the password1. At the last recursive call if password1 equals to 0 the method calls checkOdd() method where it calculates the odd numbered characters in password1. But checkOdd method doesn't effect overall time complexity because it works with fixed size array that has length of 26(number of letters in English alphabet). So accessing this fixed array takes constant time O(1).

As a result we can say that this function's time complexity depends on the length of the password(If we take it as "n"). That is why time complexity of this method is O(n).

```
public boolean isExactDivison(int password2, int[] denominations)
```

This method have one for loop that iterates through depends on the length of the denominations array(let's call it m) and make a recursive call with subtracting password2 from each denominations array elements(password2 – denominations[j] in code) and calls it self again with this parameter.(isExactDivison(password2 – denominations[j],denominations)). If password2 equals zero recursive call returns true. If password2 is smaller than zero then it returns false.



So here is a default example, in this example the number of operations depends on the length of the denominations array and the depth of the operation number, in this example at the fourth row(81 number of operation part) the recursive call ends. So we don't know exactly how many times this recursive call takes in different denomination arrays because it has to calculate all the different scenarios that value of password2 will be. As a result if we don't know how many times this calculation goes deep, we can accept it as "m". And as in example it takes 3^4 time where 3 is the length of the denominations array length and 4 is the depth or steps(let's take it as "n") that will take

to calculate every password2 values.
In conclusion we can say that we can say that time complexity of this function is O(m^n).

Outputs and different test cases:

Here is my different test cases:

```
Username u = new Username(username:"sibelgulmez");
Password1 p = new Password1(p1:"[rac()ecar]");
Password2 p2 = new Password2(p2:74);

Username u2 = new Username(username:"\"");
Password1 p1_2 = new Password1(p1:"[rac()ecar]");
Password2 p2_2 = new Password2(p2:74);

Username u3 = new Username(username:"sibel1");
Password1 p1_3 = new Password1(p1:"[rac()ecar]");
Password2 p2_3 = new Password2(p2:74);

Username u4 = new Username(username:"sibel");
Password1 p1_4 = new Password1(p1:"pass[]");
Password2 p2_4 = new Password2(p2:74);

Username u5 = new Username(username:"sibel");
Password1 p1_5 = new Password1(p1:"abcdabcd");
Password2 p2_5 = new Password2(p2:74);

Username u6 = new Username(username:"sibel");
Password1 p1_6 = new Password1(p1:"[[[[]]]]");
Password2 p2_6 = new Password2(p2:74);

Username u7 = new Username(username:"sibel");
Password1 p1_7 = new Password1(p1:"[no](no)");
Password2 p2_7 = new Password2(p2:74);
```

```
Username u8 = new Username(username:"sibel");
Password1 p1_8 = new Password1(p1:"[rac()ecar]]");
Password2 p2_8 = new Password2(p2:74);

Username u9 = new Username(username:"sibel");
Password1 p1_9 = new Password1(p1:"[rac()ecars]");
Password2 p2_9 = new Password2(p2:74);

Username u10 = new Username(username:"sibel");
Password1 p1_10 = new Password1(p1:"[rac()ecar]");
Password2 p2_10 = new Password2(p2:5);

Username u11 = new Username(username:"sibel");
Password1 p1_11 = new Password1(p1:"[rac()ecar]");
Password2 p2_11 = new Password2(p2:35);
```

Output of these test cases:

```
erkut@DESKTOP-7JISF9F:~/1801042646$ javac Homework4.java
erkut@DESKTOP-7JISF9F:~/1801042646$ java Homework4
---Test1---
The username and passwords are valid. The door is opening, please wait..
---Test2---
The username is invalid. It should have at least 1 character.
---Test3---
The username is invalid. It should have letters only.
---Test4---
The password1 is invalid. It should have at least 8 characters.
---Test5---
The password1 is invalid. It should have at least 2 brackets.
---Test6---
The password1 is invalid. It should have letters too.
---Test7---
The password1 is invalid. It should have at least 1 character from the username.
---Test8---
The password1 is invalid. It should be balanced.
---Test9---
The password1 is invalid. It should be possible to obtain a palindrome from the password1.
---Test10---
The password2 is invalid. It should be between 10 and 10000.
---Test11---
The password1 is invalid. It should be possible to obtain a palindrome from the password1.
erkut@DESKTOP-7JISF9F:~/1801042646$
```

Design of the Homework:

Schema of the code:

```
public class Homework4{

        public static class Username{
                -Constructor
                //Methods
                *getUsername()
                * checkIfValidUsername()
        }

        public static class password1{
                -Constructor
                //Methods
                *getPassword1()
                *getLength()
                *isBracketPassword1()
                *isLetterContainPassword1()
                *removeBrackets()
                *containUserNameSpirit()
                *isBalancedPassword
                *checkOdd()
                *isPalindrome()
        }

        public static class Password2{
                -Constructor
                //Methods
                *getPassword2()
                *isExactDivison()
        }

        public static void main(){

        //Test objects
        //Calling test functions
        }
```

```
        public static void Test#(Username u, Password1 p1, Password2 p2){

        // Checking the validity of password1 and password2
        // Calling the five necessary function to check the validity of username
        password1, password2. If they are valid open the door otherwise print the
        necessary error message inside the functions or in here.
        }
//"#" means the number of Test function. They are implemented in a same way.

}//End of the class Homework4
```

Username, Password1 and Password2 has its own class and necessary functions. Each object uses its own class methods to get tested. In main I created different number of objects and send them to Test classes.