GTU DEPARTMENT of COMPUTER ENGINEERING

CSE222/505 – SPRING 2023

HOMEWORK 4 REPORT

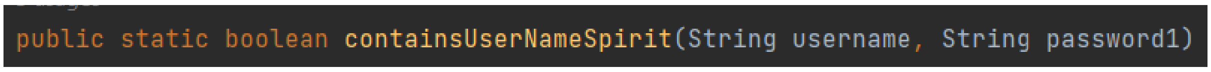
ÖMER FARUK ÇOLAKEL

200104004043

1. SYSTEM REQUIREMENTS and PROBLEM SOLUTION APPROACHES

Stack Methods:

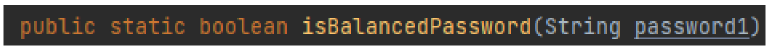
Part 2:



```
public static boolean containsUserNameSpirit(String username, String password1)
```

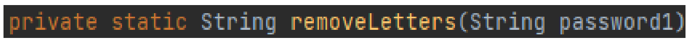
Requires username as a string and password1 as string. Checks username and password1's lengths. Pushes every character in username to a stack. Iterates through password1 string

Part 3:



```
public static boolean isBalancedPassword(String password1)
```

Requires password1. Checks length of password1. Removes letters using removeLetters method. If the new string that came from that method has less than 2 letters, returns false. Iterates through the string, if it finds an opening bracket pushes it to the stack. If it finds a closing bracket, checks if the stack is empty and if not, checks if the top bracket is the same type as the bracket at the top. If it's empty or brackets doesn't match returns false. At the end of the string if the stack is no empty, it means that there is an unnecessary opening bracket so it returns false. Returns true at the end.

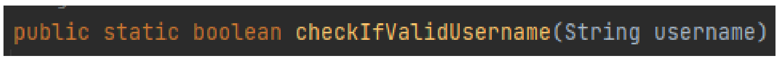


```
private static String removeLetters(String password1)
```

Requires password1 as string. Builds a string without letters.

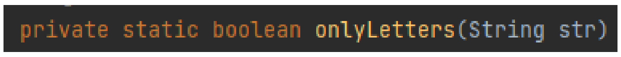
Recursive Methods:

Part1:



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

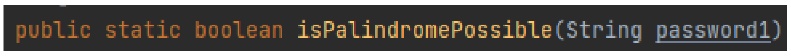
Requires username as string. Checks initial length which shouldn't be empty. Returns false if it is empty. Returns the return value of onlyLetters method.



```
private static boolean onlyLetters(String str)
```

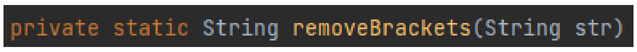
This is a helper recursive method. If string is empty, it means that there are no characters other than letters so returns true (initially string can't be empty but we checkes it in checkIfValidUsername method). Checks the character at index 0, if it isn't a letter returns false. If it is calls itself without the first letter.

Part 4:



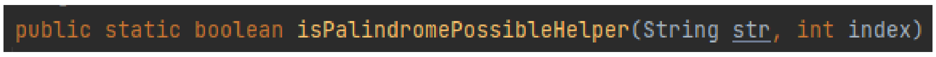
```
public static boolean isPalindromePossible(String password1)
```

Requires passwrod1 as string. Checks initial input. Password1 should have at least 8 characters. Removes brackets using removeBrackets method. Checks returned string since password1 should have at least 2 brackets. Then calls isPalindromePossibleHelper method.



```java
private static String removeBrackets(String str)
```

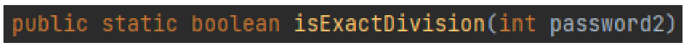
Removes brackets from the input string and returns the new string.



```java
public static boolean isPalindromePossibleHelper(String str, int index)
```

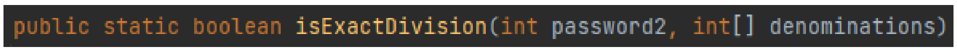
This is the recursive method. Requires a string and an integer. If the first and last characters are same calls itself with the remaining string. If they are not matching it doesn't return false immediately. Those characters might be in the middle of the string. Takes last character as pivot, increments index by 1 and calls itself. If index is not at the end, checks the character with given index. If that character and last character is same, builds a new string and calls itself with the index of 0. If it's at the end, there is a possibility of string having odd number of characters so one character doesn't have a match. If string has even number of characters returns false. If it is odd, ignores last character and calls itself without that character with index 0. Since new string is even, it can't ignore the second instance of non-matching character. If index finds a match without being at the end of the string, calls itself without the character in the index and last character, also resetting index value. After all of this if string is empty, returns true.

Part 5:



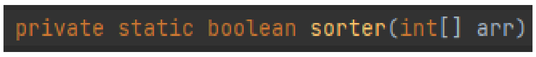
```java
public static boolean isExactDivision(int password2)
```

First isExactDivision requires a password. It calls second isExactDivision with the default array which was given in the PDF.



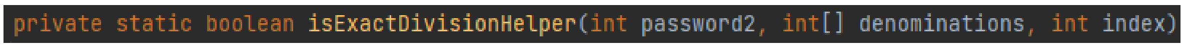
```java
public static boolean isExactDivision(int password2, int[] denominations)
```

Requires an integer and an integer array. Checks array and if it's empty calls first isExactDivision method. If password is smaller or greater than the range that was given in homework PDF, returns false. Calls sorter method to check and sort the array and then isExactDivisionHelper method.



```java
private static boolean sorter(int[] arr)
```

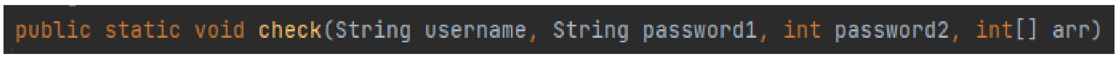
Sorts integer arrays in ascending order. If one of the denominators is less than 1, returns false.



```java
private static boolean isExactDivisionHelper(int password2, int[] denominations, int index)
```

This is the recursive method. If the denominator in the given index can divide it without remainder, returns true. If index is not at the end, increments it by 1 and calls itself so it can try with the new denominator. If the index is at the end, decrements password2 with greatest denominator, resets index and calls itself. If password is 0 in every try, return false.

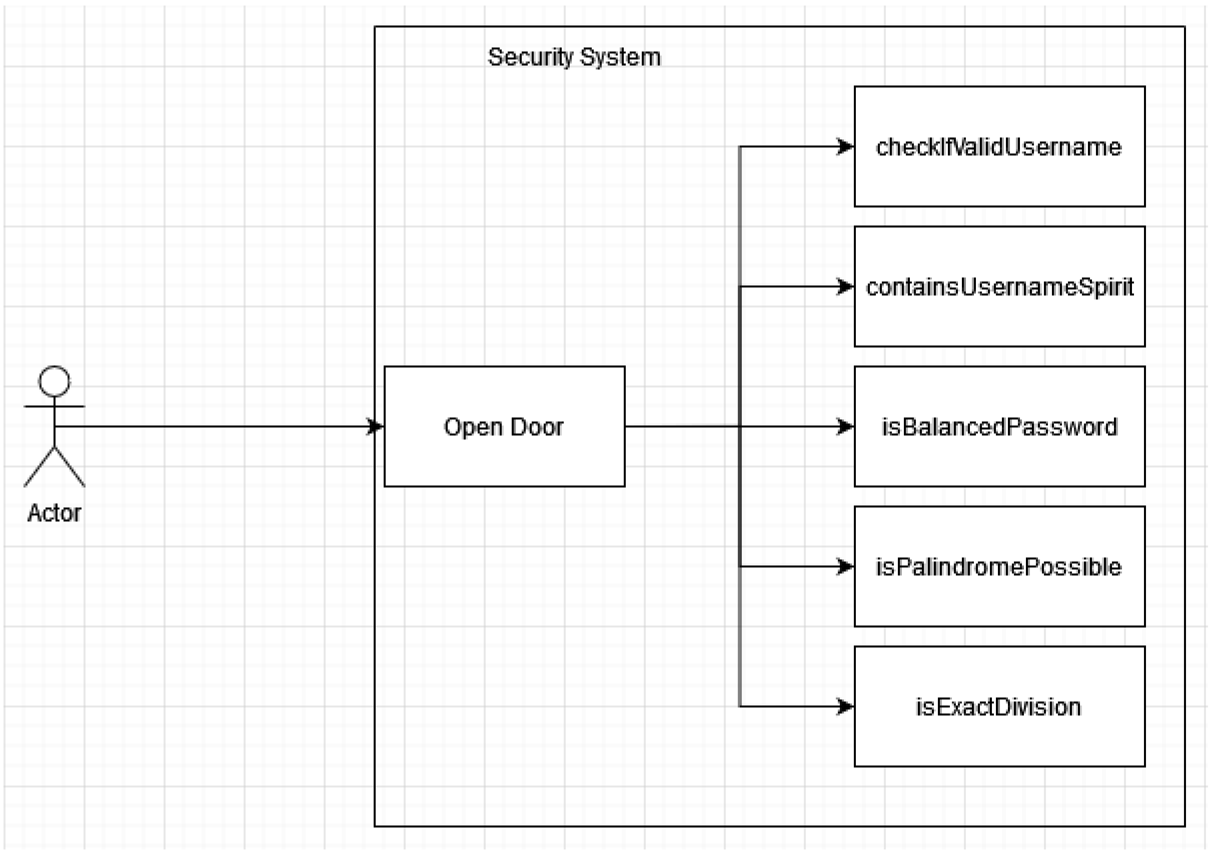
Main:



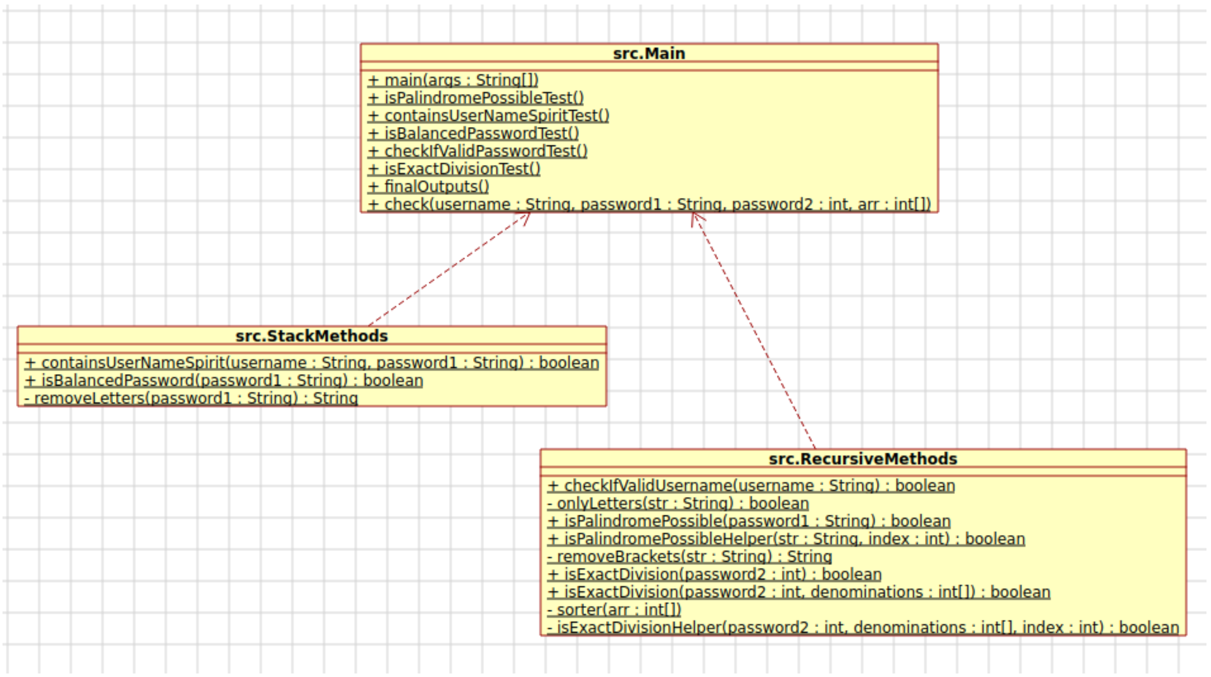
```java
public static void check(String username, String password1, int password2, int[] arr)
```

Requires username string, password1 integer, password2 integer and an integer array. Calls all recursive and stack methods with given parameters. Returns false if one of them is false and returns true if every one of them is true.

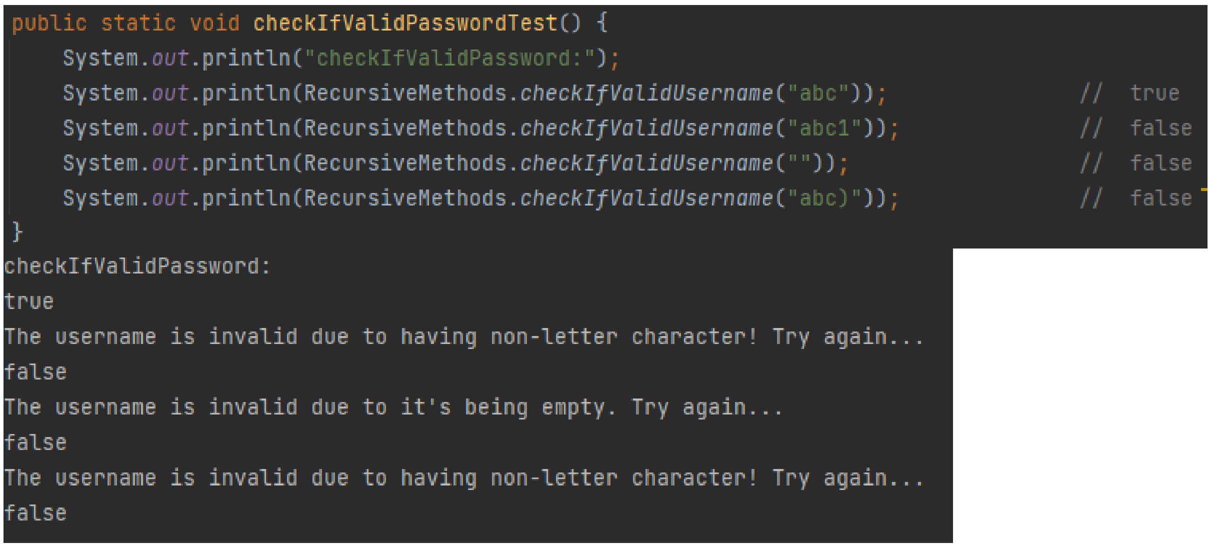
2. USE CASE DIAGRAM



3. CLASS DIAGRAM

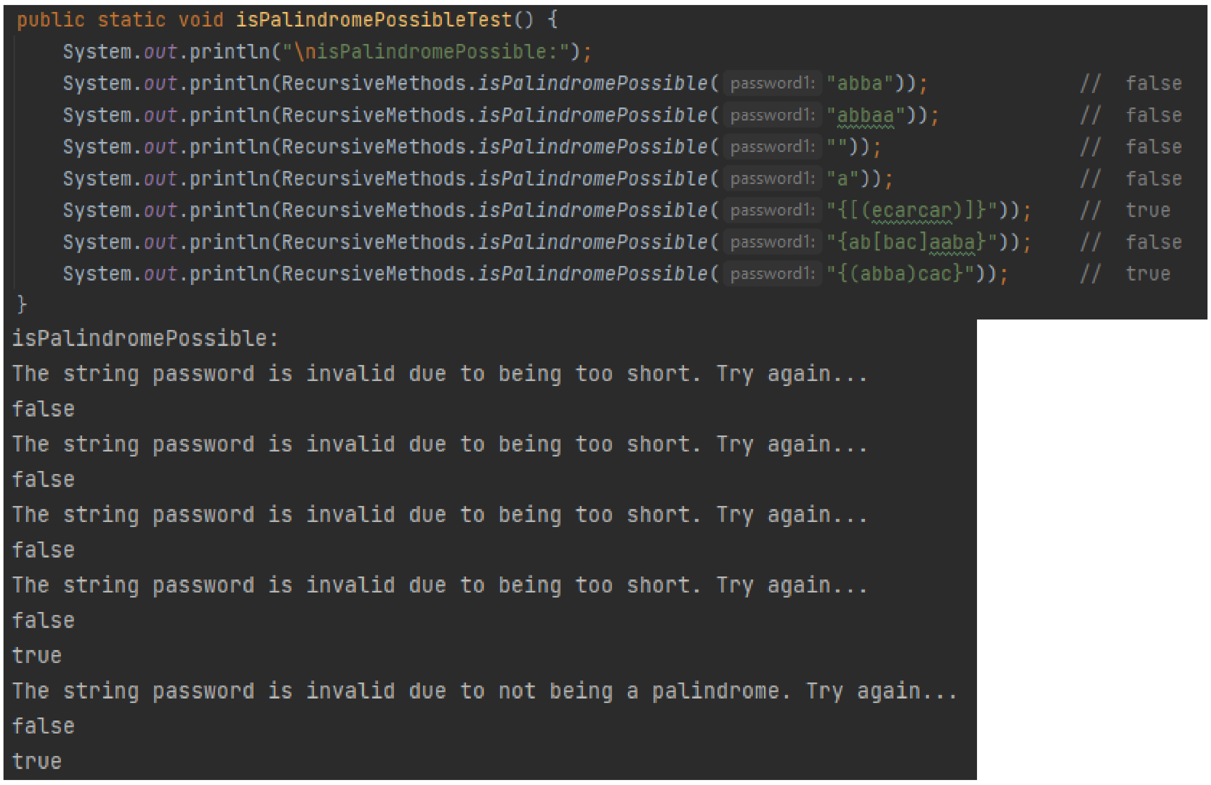

src.Main
+ main(args : String[])
+ isPalindromePossibleTest()
+ containsUserNameSpiritTest()
+ isBalancedPasswordTest()
+ checkIfValidPasswordTest()
+ isExactDivisionTest()
+ finalOutputs()
+ check(username : String, password1 : String, password2 : int, arr : int[])
src.StackMethods
+ containsUserNameSpirit(username : String, password1 : String) : boolean
+ isBalancedPassword(password1 : String) : boolean
- removeLetters(password1 : String) : String
src.RecursiveMethods
+ checkIfValidUsername(username : String) : boolean
- onlyLetters(str : String) : boolean
+ isPalindromePossible(password1 : String) : boolean
+ isPalindromePossibleHelper(str : String, index : int) : boolean
- removeBrackets(str : String) : String
+ isExactDivision(password2 : int) : boolean
+ isExactDivision(password2 : int, denominations : int[]) : boolean
- sorter(arr : int[])
- isExactDivisionHelper(password2 : int, denominations : int[], index : int) : boolean

4. TEST CASES and RESULTS



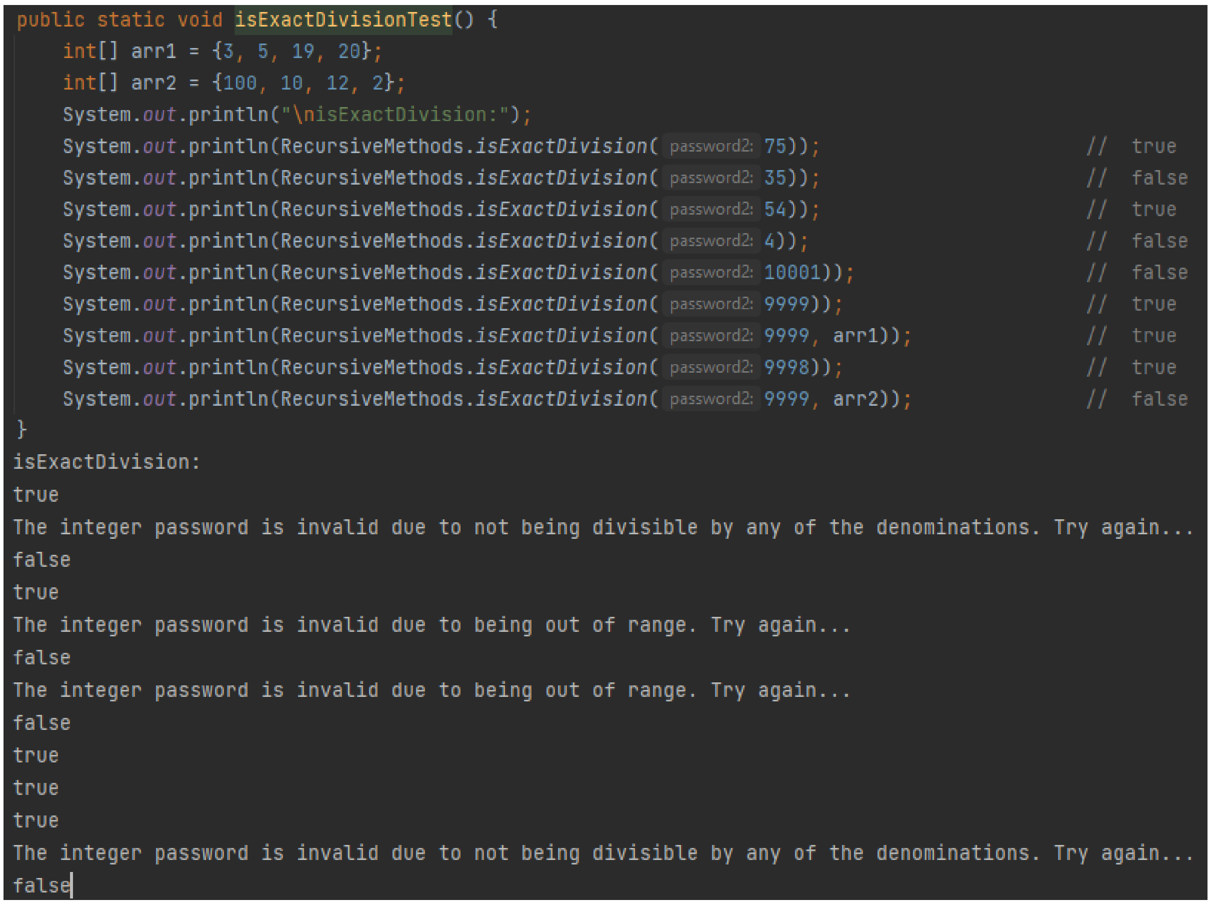
```
public static void checkIfValidPasswordTest() {
    System.out.println("checkIfValidPassword:");
    System.out.println(RecursiveMethods.checkIfValidUsername("abc"));            //  true
    System.out.println(RecursiveMethods.checkIfValidUsername("abc1"));           //  false
    System.out.println(RecursiveMethods.checkIfValidUsername(""));               //  false
    System.out.println(RecursiveMethods.checkIfValidUsername("abc)"));           //  false
}
```

```
checkIfValidPassword:
true
The username is invalid due to having non-letter character! Try again...
false
The username is invalid due to it's being empty. Try again...
false
The username is invalid due to having non-letter character! Try again...
false
```



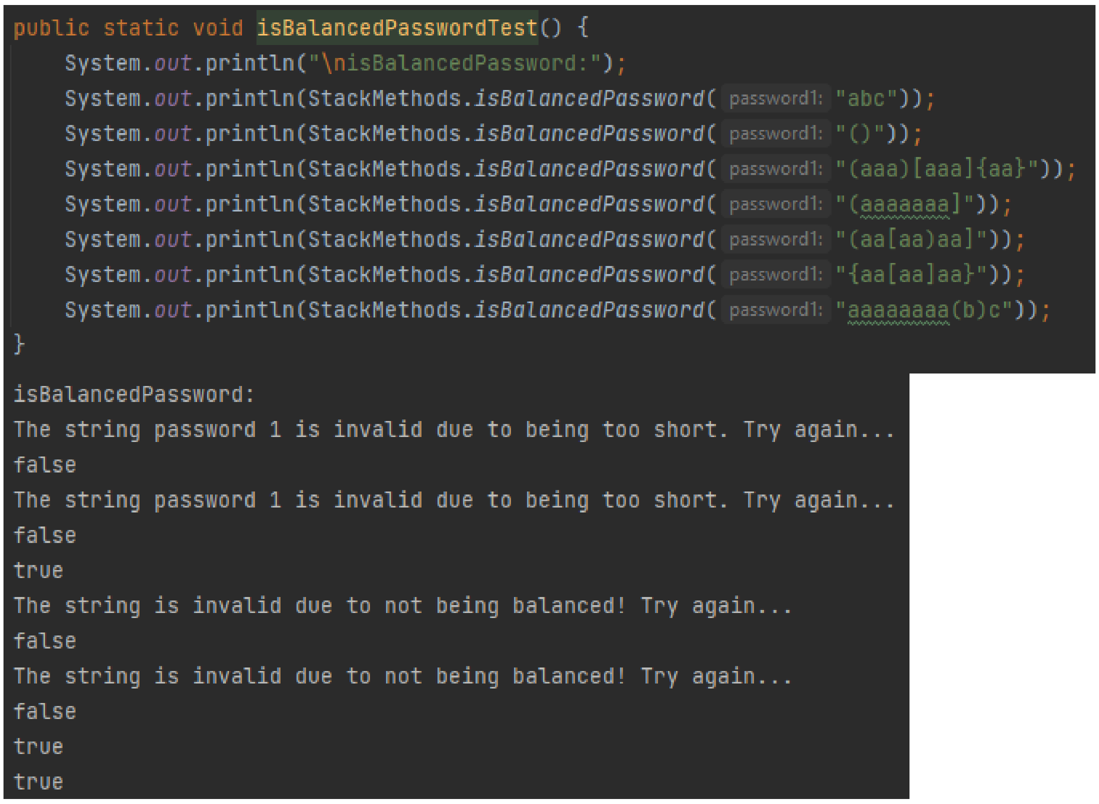
```
public static void isPalindromePossibleTest() {
    System.out.println("\nisPalindromePossible:");
    System.out.println(RecursiveMethods.isPalindromePossible( password1: "abba"));          //  false
    System.out.println(RecursiveMethods.isPalindromePossible( password1: "abbaa"));         //  false
    System.out.println(RecursiveMethods.isPalindromePossible( password1: ""));              //  false
    System.out.println(RecursiveMethods.isPalindromePossible( password1: "a"));             //  false
    System.out.println(RecursiveMethods.isPalindromePossible( password1: "{[(ecarcar)]}")); //  true
    System.out.println(RecursiveMethods.isPalindromePossible( password1: "{ab[bac]aaba}")); //  false
    System.out.println(RecursiveMethods.isPalindromePossible( password1: "{(abba)cac}"));   //  true
}
```

```
isPalindromePossible:
The string password is invalid due to being too short. Try again...
false
The string password is invalid due to being too short. Try again...
false
The string password is invalid due to being too short. Try again...
false
The string password is invalid due to being too short. Try again...
false
true
The string password is invalid due to not being a palindrome. Try again...
false
true
```



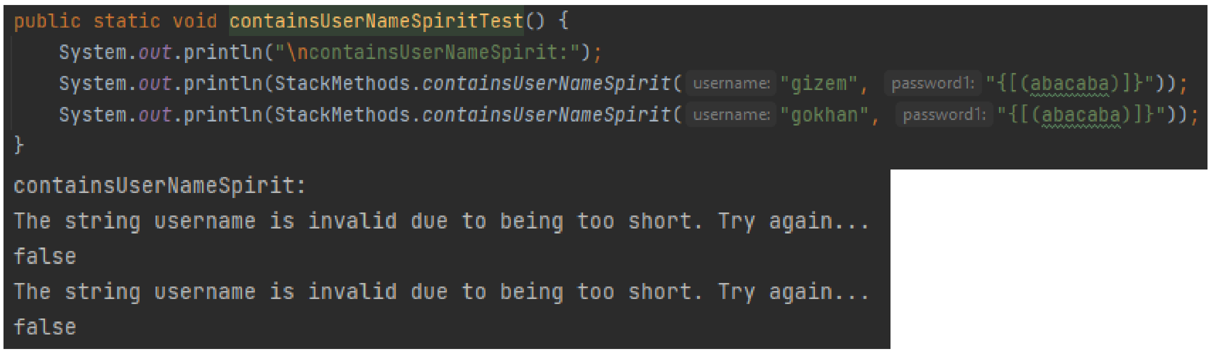
```
public static void isExactDivisionTest() {
    int[] arr1 = {3, 5, 19, 20};
    int[] arr2 = {100, 10, 12, 2};
    System.out.println("\nisExactDivision:");
    System.out.println(RecursiveMethods.isExactDivision( password2: 75));                    //  true
    System.out.println(RecursiveMethods.isExactDivision( password2: 35));                    //  false
    System.out.println(RecursiveMethods.isExactDivision( password2: 54));                    //  true
    System.out.println(RecursiveMethods.isExactDivision( password2: 4));                     //  false
    System.out.println(RecursiveMethods.isExactDivision( password2: 10001));                 //  false
    System.out.println(RecursiveMethods.isExactDivision( password2: 9999));                  //  true
    System.out.println(RecursiveMethods.isExactDivision( password2: 9999, arr1));            //  true
    System.out.println(RecursiveMethods.isExactDivision( password2: 9998));                  //  true
    System.out.println(RecursiveMethods.isExactDivision( password2: 9999, arr2));            //  false
}
```

```
isExactDivision:
true
The integer password is invalid due to not being divisible by any of the denominations. Try again...
false
true
The integer password is invalid due to being out of range. Try again...
false
The integer password is invalid due to being out of range. Try again...
false
true
true
true
The integer password is invalid due to not being divisible by any of the denominations. Try again...
false
```



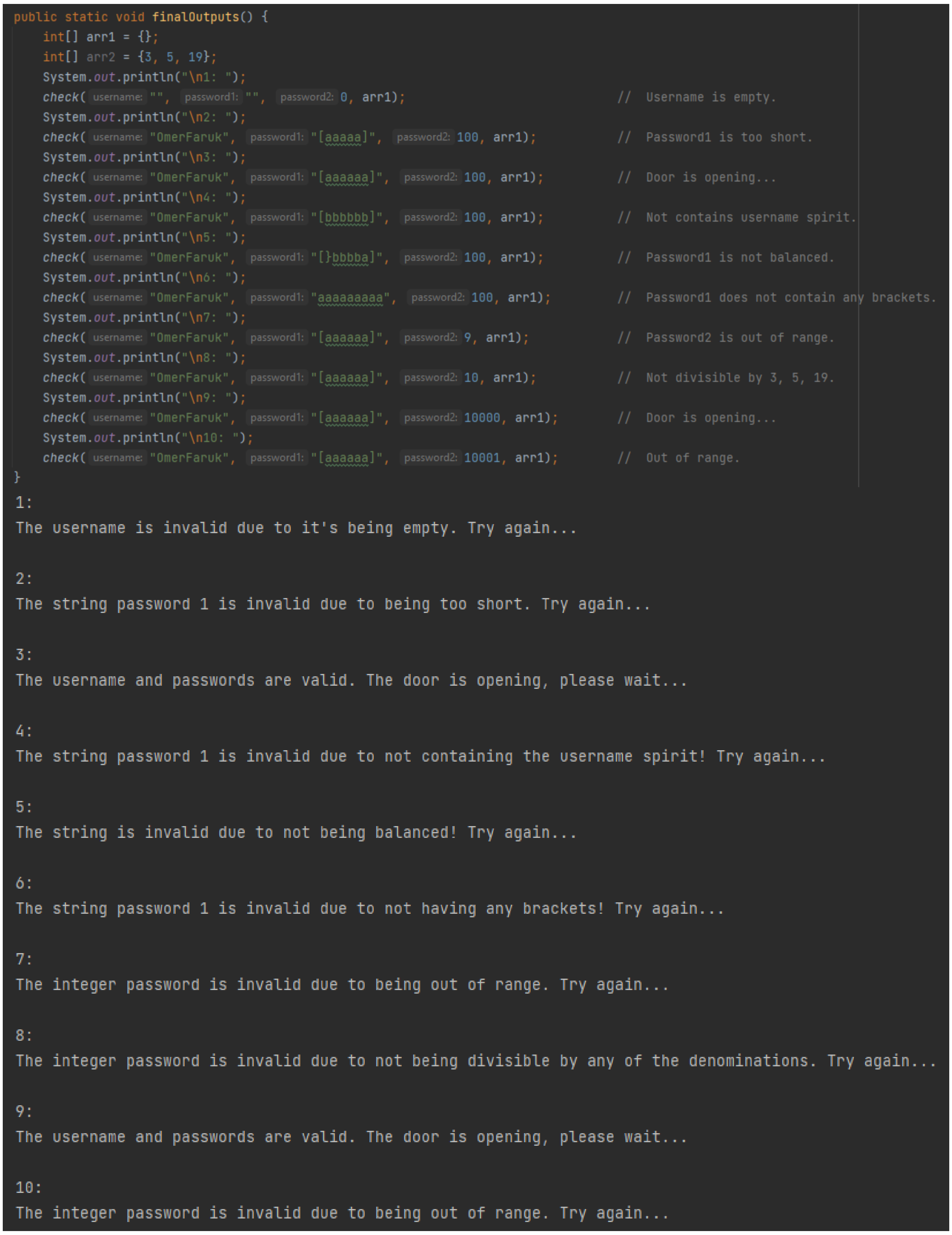
```
public static void isBalancedPasswordTest() {
    System.out.println("\nisBalancedPassword:");
    System.out.println(StackMethods.isBalancedPassword( password1: "abc"));
    System.out.println(StackMethods.isBalancedPassword( password1: "()"));
    System.out.println(StackMethods.isBalancedPassword( password1: "(aaa)[aaa]{aa}"));
    System.out.println(StackMethods.isBalancedPassword( password1: "(aaaaaaa]"));
    System.out.println(StackMethods.isBalancedPassword( password1: "(aa[aa)aa]"));
    System.out.println(StackMethods.isBalancedPassword( password1: "{aa[aa]aa}"));
    System.out.println(StackMethods.isBalancedPassword( password1: "aaaaaaaa(b)c"));
}
```

```
isBalancedPassword:
The string password 1 is invalid due to being too short. Try again...
false
The string password 1 is invalid due to being too short. Try again...
false
true
The string is invalid due to not being balanced! Try again...
false
The string is invalid due to not being balanced! Try again...
false
true
true
```



```
public static void containsUserNameSpiritTest() {
    System.out.println("\ncontainsUserNameSpirit:");
    System.out.println(StackMethods.containsUserNameSpirit( username: "gizem",  password1: "{[(abacaba)]}"));
    System.out.println(StackMethods.containsUserNameSpirit( username: "gokhan",  password1: "{[(abacaba)]}"));
}
```

```
containsUserNameSpirit:
The string username is invalid due to being too short. Try again...
false
The string username is invalid due to being too short. Try again...
false
```



```
public static void finalOutputs() {
    int[] arr1 = {};
    int[] arr2 = {3, 5, 19};
    System.out.println("\n1: ");
    check( username: "",  password1: "",  password2: 0, arr1);                         //  Username is empty.
    System.out.println("\n2: ");
    check( username: "OmerFaruk",  password1: "[aaaaa]",  password2: 100, arr1);           //  Password1 is too short.
    System.out.println("\n3: ");
    check( username: "OmerFaruk",  password1: "[aaaaaa]",  password2: 100, arr1);          //  Door is opening...
    System.out.println("\n4: ");
    check( username: "OmerFaruk",  password1: "[bbbbbb]",  password2: 100, arr1);          //  Not contains username spirit.
    System.out.println("\n5: ");
    check( username: "OmerFaruk",  password1: "[}bbbba]",  password2: 100, arr1);          //  Password1 is not balanced.
    System.out.println("\n6: ");
    check( username: "OmerFaruk",  password1: "aaaaaaaaa",  password2: 100, arr1);         //  Password1 does not contain any brackets.
    System.out.println("\n7: ");
    check( username: "OmerFaruk",  password1: "[aaaaaa]",  password2: 9, arr1);            //  Password2 is out of range.
    System.out.println("\n8: ");
    check( username: "OmerFaruk",  password1: "[aaaaaa]",  password2: 10, arr1);           //  Not divisible by 3, 5, 19.
    System.out.println("\n9: ");
    check( username: "OmerFaruk",  password1: "[aaaaaa]",  password2: 10000, arr1);        //  Door is opening...
    System.out.println("\n10: ");
    check( username: "OmerFaruk",  password1: "[aaaaaa]",  password2: 10001, arr1);        //  Out of range.
}
```

```
1:
The username is invalid due to it's being empty. Try again...

2:
The string password 1 is invalid due to being too short. Try again...

3:
The username and passwords are valid. The door is opening, please wait...

4:
The string password 1 is invalid due to not containing the username spirit! Try again...

5:
The string is invalid due to not being balanced! Try again...

6:
The string password 1 is invalid due to not having any brackets! Try again...

7:
The integer password is invalid due to being out of range. Try again...

8:
The integer password is invalid due to not being divisible by any of the denominations. Try again...

9:
The username and passwords are valid. The door is opening, please wait...

10:
The integer password is invalid due to being out of range. Try again...
```

5.TIME COMPLEXITIES

Part 1 (checkIfValidUsername): At worst case, username meets our requirements and it takes O(n) time to check whole string. N is the length of the string.

Part 2 (containsUsernameSpirit): There is a nested for loop that depends on both strings' lengths. That's why worst case scenario is O(n.m).

Part 3 (isBalancedPassword): removeLetters method is O(n). A while loop iterates through the string which makes it O(n). If we add those 2, we get O(2n) which is the same thing as O(n).

Part 4 (isPalindromePossible): This recursive will iterate through the password multiple times. Those iterations' worst case is O(n). Since O(n+n) is still O(n), whole function is O(n).

Part 5 (isExactDivision): If length of the array is n, sorting the array takes O(n^2) time. At worst case we will have to go through the array multiple times in linear time which makes the helper O(n). With other methods that isExactDivision uses, it is O(n^2).