GTU Department of Computer Engineering
CSE 222 - Spring 2023
Homework 4 Report

BEYZA ACAR

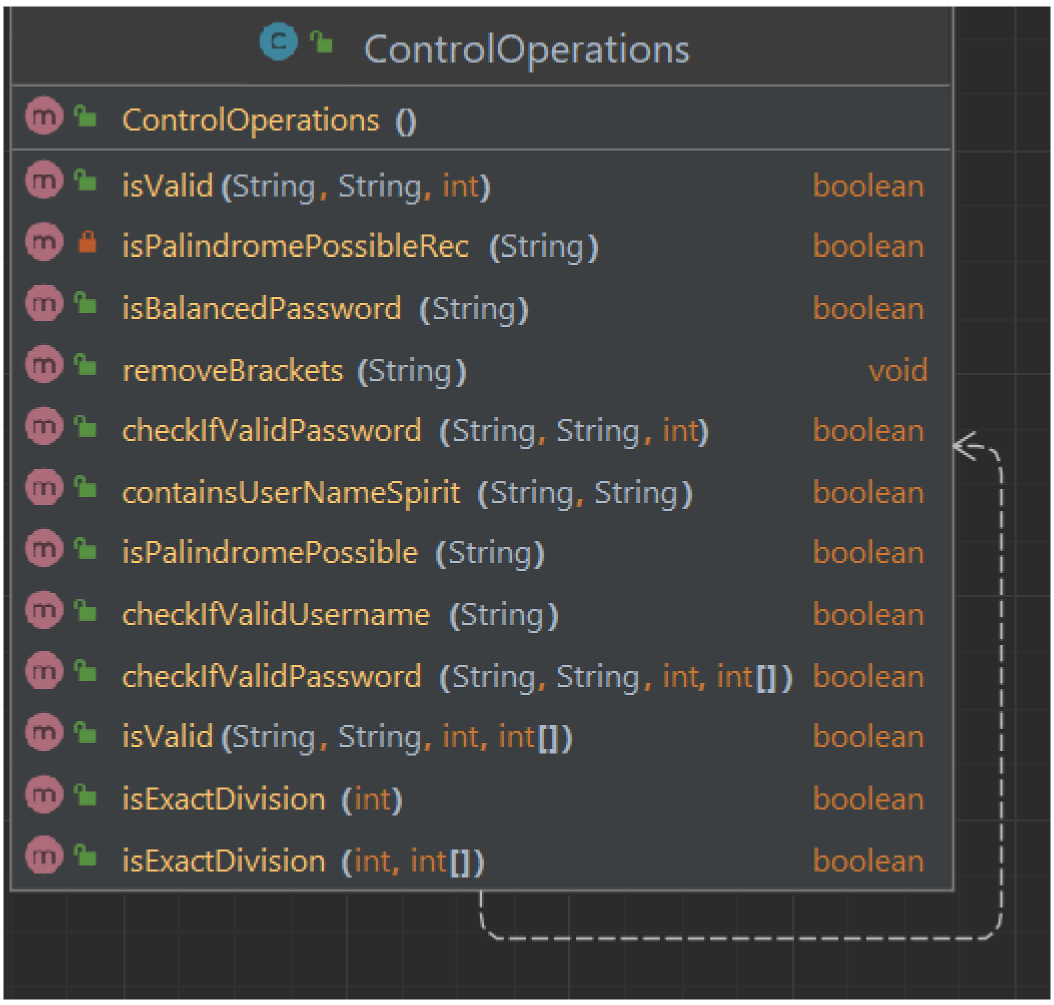
200104004065

20.04.2023

1- System Requirements

We are designing a security system for the museum in Topkapı Palace. The museum security system must provide an encrypted password for each museum officer. When the officers need to enter the warehouse where the most valuable items of the museum are stored, they need a personal password pair to type to the security system which is in front of the entrance of the warehouse. The personal password pair is specified and assigned to the officer. When an officer types their username and personal password pair, our system is responsible for decoding the passwords to check if they are valid or not. If the passwords are valid, the system opens the entrance door to let the officer get in. Accept that the passwords have already been given to the officers by the manager of the museum. Our security system does not need to generate the passwords of the officers.

2- UML Diagram



```
ControlOperations
ControlOperations ()
isValid (String, String, int)                              boolean
isPalindromePossibleRec (String)                           boolean
isBalancedPassword (String)                                boolean
removeBrackets (String)                                    void
checkIfValidPassword (String, String, int)                 boolean
containsUserNameSpirit (String, String)                    boolean
isPalindromePossible (String)                              boolean
checkIfValidUsername (String)                              boolean
checkIfValidPassword (String, String, int, int[])          boolean
isValid (String, String, int, int[])                       boolean
isExactDivision (int)                                      boolean
isExactDivision (int, int[])                               boolean
```

3-<u>Problem Solution Approach</u>

- For **<u>checkIfValidUsername(String username)</u>** recursive method :
    - Each time before the function is called again, It sends the given string one character short of the beginning.
    - Checks fort he first char in the string is a letter or not
    - If not it returns false, otherwise true.
    - The complexity of this method is ***<u>O(n).</u>***

- For **<u>boolean containsUserNameSpirit(String username, String password1):</u>**
    - Creats a stack.
    - Adds the character of password1 into this stack
    - compares this stack and username if they have a common character. If not, this method returns false.
    - The complexity of this method is ***<u>O(m*n)</u>*** which m = length of password1 and n is the length of username.

- For **<u>boolean isBalancedPassword(String password1):</u>**
    - If the length of the given password is less than 1, the function returns false.
    - A stack data structure is created.
    - Created a loop for iterate over password characters, if the read character is a open bracket, it is pushed into the stack, if it is a closing bracket, the stack is popped and compare if these two are equals. If yes, loop continues, if not method returns false.
    - The complexity of this method is ***<u>O(n)</u>*** which n is length of the password1

- For **<u>isPalindromePossible(String password1):</u>**
    - If the length of the given string is 0 returns true.
    - If the length of the given string is 1 returns false.

    - Creates a character named c and a string that contains characters of password1 except the first character (substring of password1)
    - Method checks if there is another character that is the same as the first character in the password1, if not password1 can not be a palindrome because first char in the password1 have not a pair, if yes method calls itself with the substring of it (that is created earlier as I've mentioned).
    - The complexity of this method is ***O(n)***

- For **boolean isExactDivision(int password2, int [] denominations) :**
    - Returns true if password2 is equal to 0.
    - Returns false if password2 is less then 0.
    - Creates a for loop to iterate over denominations and returns itself with the parameter of (password2 – i.th element of the denominations) to reduce the problem.
    - This function uses a brute force problem approach to check all denominations, if the password2 parameter is negative, this means dont use this denomination , instead, go back and try the other denomination till password2 is zero. If it is impossible to obtain zero from password2 with all the combination of denominators, this means this method should return false.
    - The complexity of this method is **O(m*n)** which m is the size of the denominations and n is the length of

- For **boolean isExactDivision(int password2) :**
    - It is the overloaded version of **boolean isExactDivision(int password2, int [] denominations)**
    - Default denominators are {4, 17, 29}

    - The complexity of this method is ***<u>O(m*n)</u>*** which m is length of denominations and n is directly proportional with password2 value.

- For **<u>String removeBrackets(String password1)</u>** :
    - Creates a char named c and a empty String named newPassword.
    - Creates a loop to iteate over the elements of password1 with char c it created
    - The loop iterates over each element and add it to the newPassword unless the element is not a parenthesis, in that case it skips to the next element.
    - The complexity of this method is ***<u>O(n)</u>*** which n is length of password1.
- For **<u>boolean checkIfValidPassword(String username, String password1, int password2, int [] denominations) :</u>**
    - If the length of the password1 is less than 8, tells why it returns false and returns false.
    - If password2 is less than 10 or greater than 1000, tells why it returns false and returns false.
    - Other than these, this method just call other methods to control if passwords are valid. The methods are :
        - **containsUserNameSpirit(username, password1)**
        - **isBalancedPassword(password1)**
        - **isPalindromePossible(password1)**
        - **isExactDivision(password2, denominations)**
    - It is created for the user’s ease of use.
    - The complexity of this method is ***<u>O(m*n)</u>*** which m = length of password1 and n is the length of username because of the containsUserNameSpirit(username, password1) method.

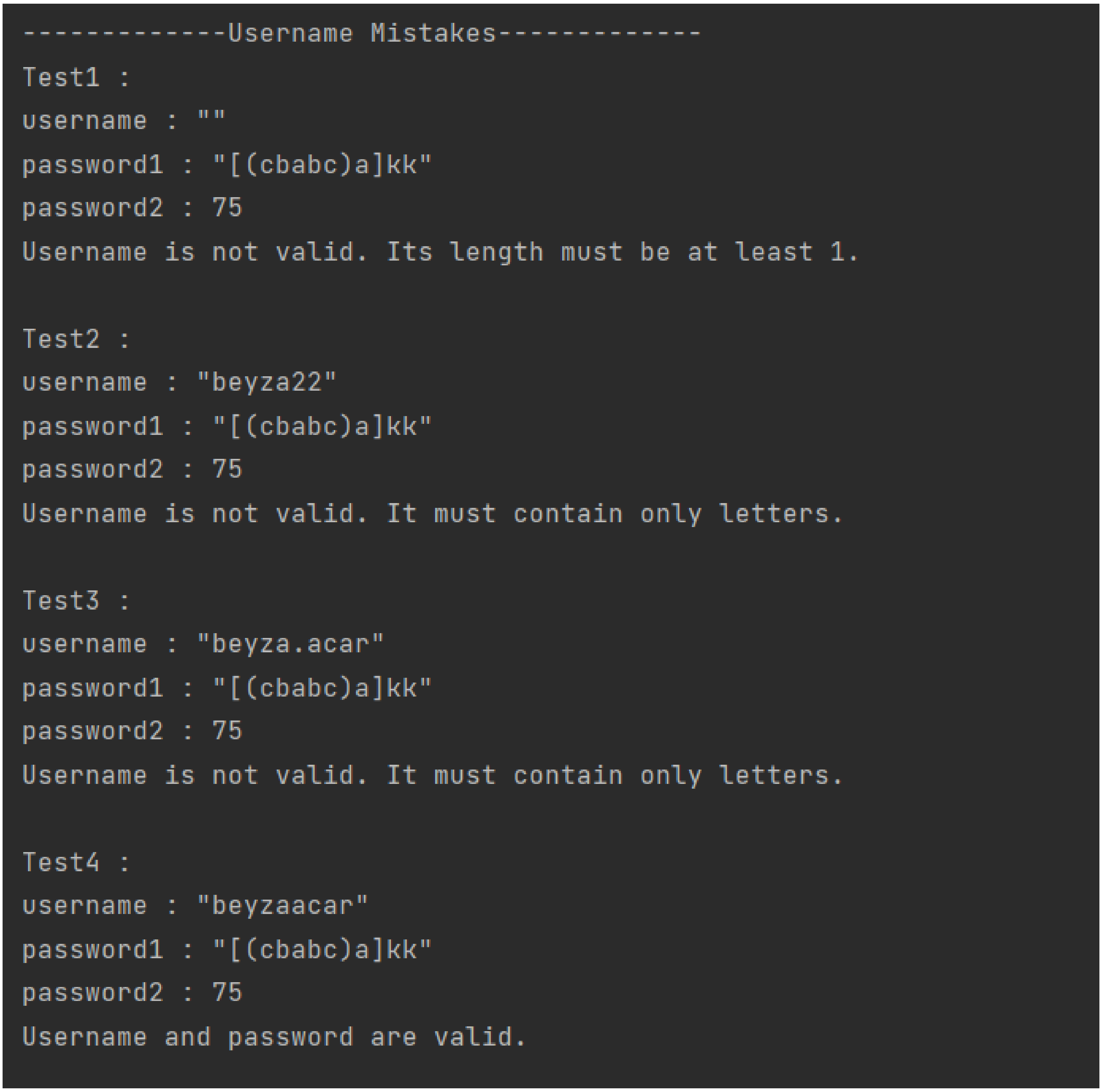
- For **<u>boolean checkIfValidPassword(String username, String password1, int password2) :</u>**
    - It is the overloaded version of **<u>boolean checkIfValidPassword(String username, String password1, int password2, int [] denominations) :</u>**
    - The only difference is it calls the overloaded **isExactDivision()**
    - The complexity of this method is ***<u>O(m*n)</u>*** which m = length of password1 and n is the length of username because of the containsUserNameSpirit(username, password1) method.

- For **<u>boolean isValid(String username, String password1, int password2, int [] denominations) :</u>**
    - This method just use other functions to control all the parameter at once. Created for user's ease of use.
    - Invoked methods : **checkIfValidUsername(username)** and **checkIfValidPassword(username, password1 , password2, denominations)**
    - The complexity of this method is ***<u>O(m*n)</u>*** which m = length of password1 and n is the length of username. Because of the For **boolean checkIfValidPassword(String username, String password1, int password2) method**

- For **<u>boolean isValid(String username, String password1, int password2) :</u>**
    - This is an overloaded version of **<u>boolean isValid(String username, String password1, int password2, int [] denominations)</u>** method.
    - Default denominators are : {4, 17, 29}
    - The complexity of this method is ***<u>O(m*n)</u>*** which m = length of password1 and n is the length of username. Because of the For **boolean checkIfValidPassword(String username, String password1, int password2) method**

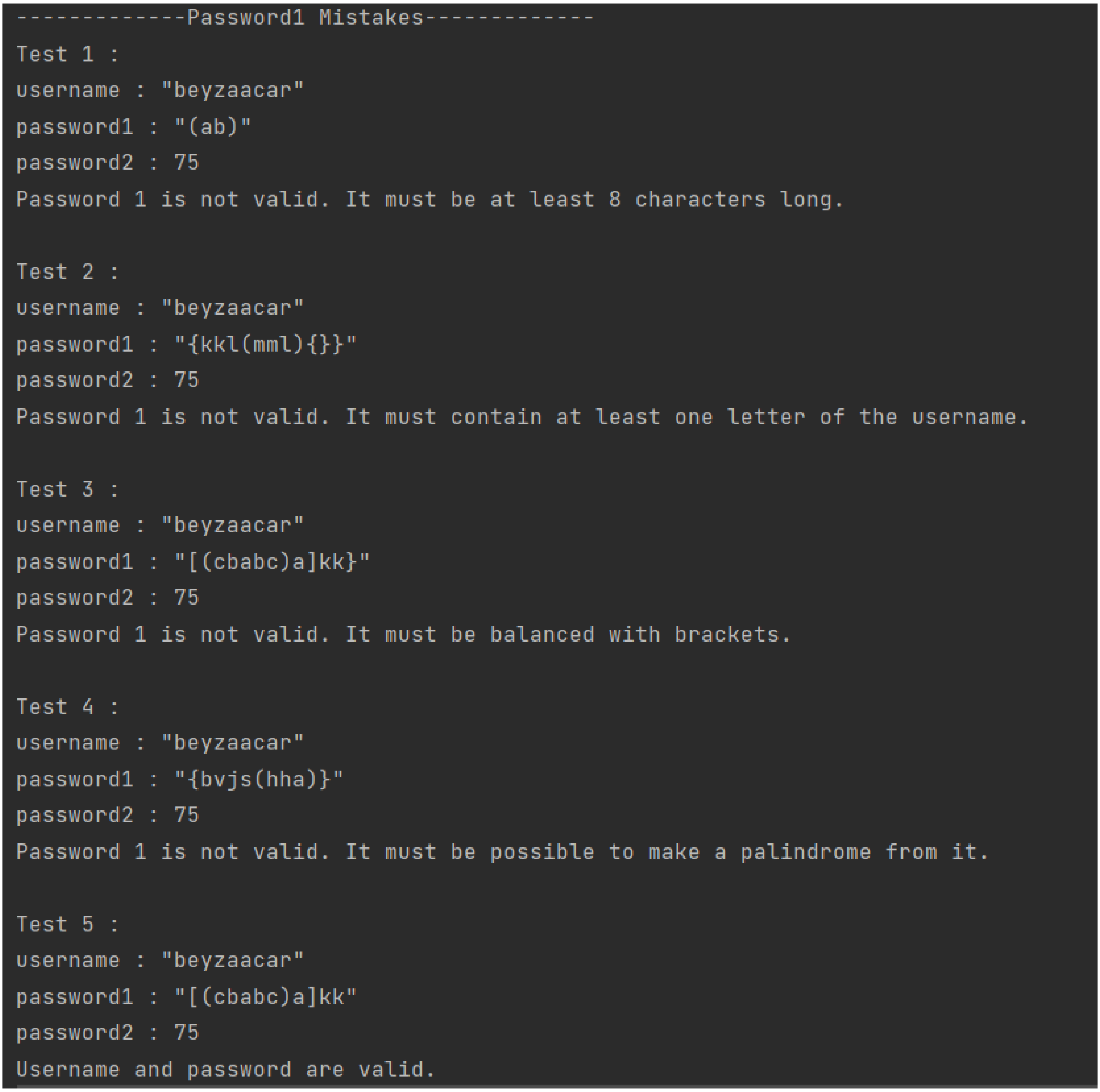
4-*Test Cases – Running and Results*

- ***Username mistakes***



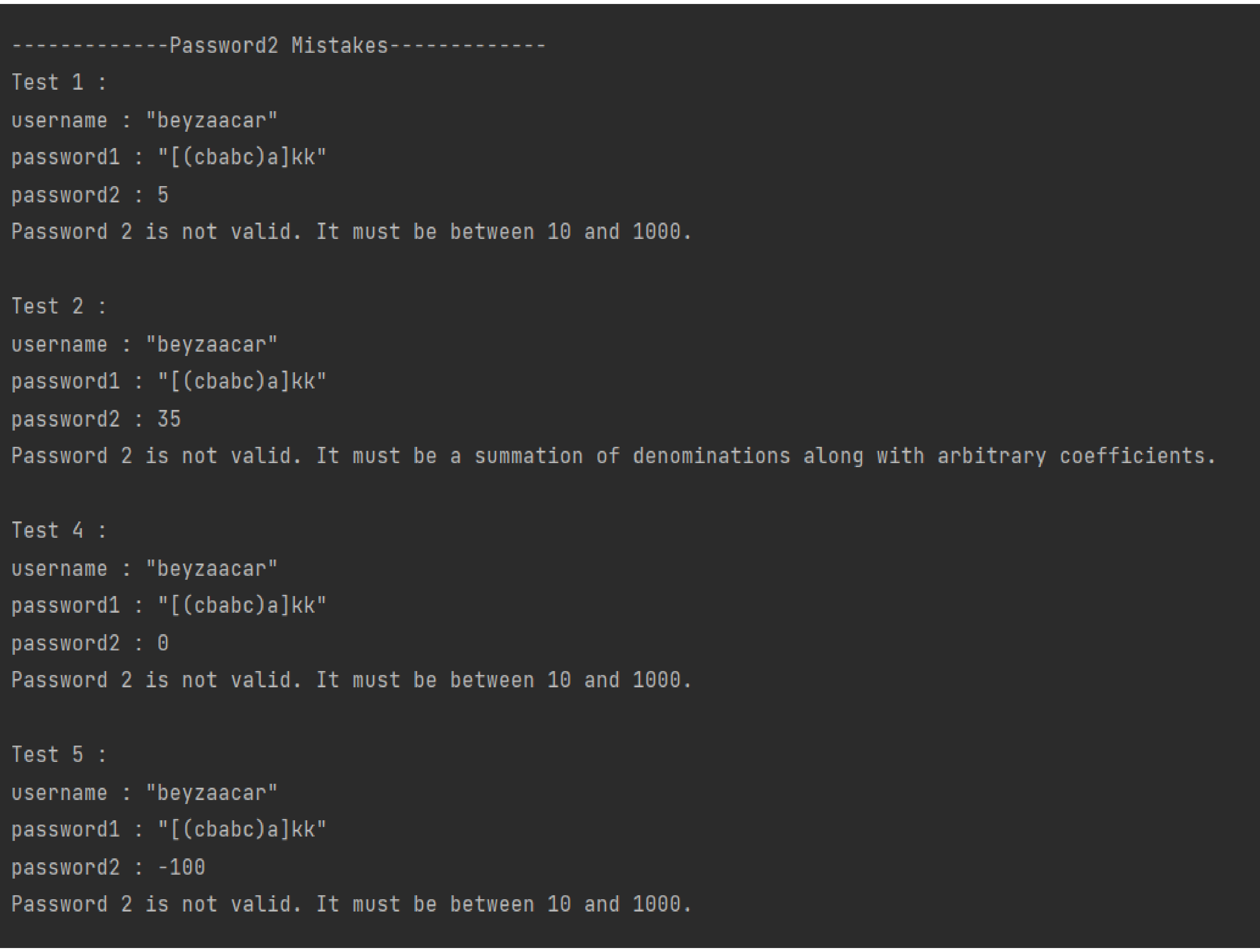
```
-------------Username Mistakes-------------
Test1 :
username : ""
password1 : "[(cbabc)a]kk"
password2 : 75
Username is not valid. Its length must be at least 1.

Test2 :
username : "beyza22"
password1 : "[(cbabc)a]kk"
password2 : 75
Username is not valid. It must contain only letters.

Test3 :
username : "beyza.acar"
password1 : "[(cbabc)a]kk"
password2 : 75
Username is not valid. It must contain only letters.

Test4 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 75
Username and password are valid.
```

- ***<u>Password1 mistakes</u>***



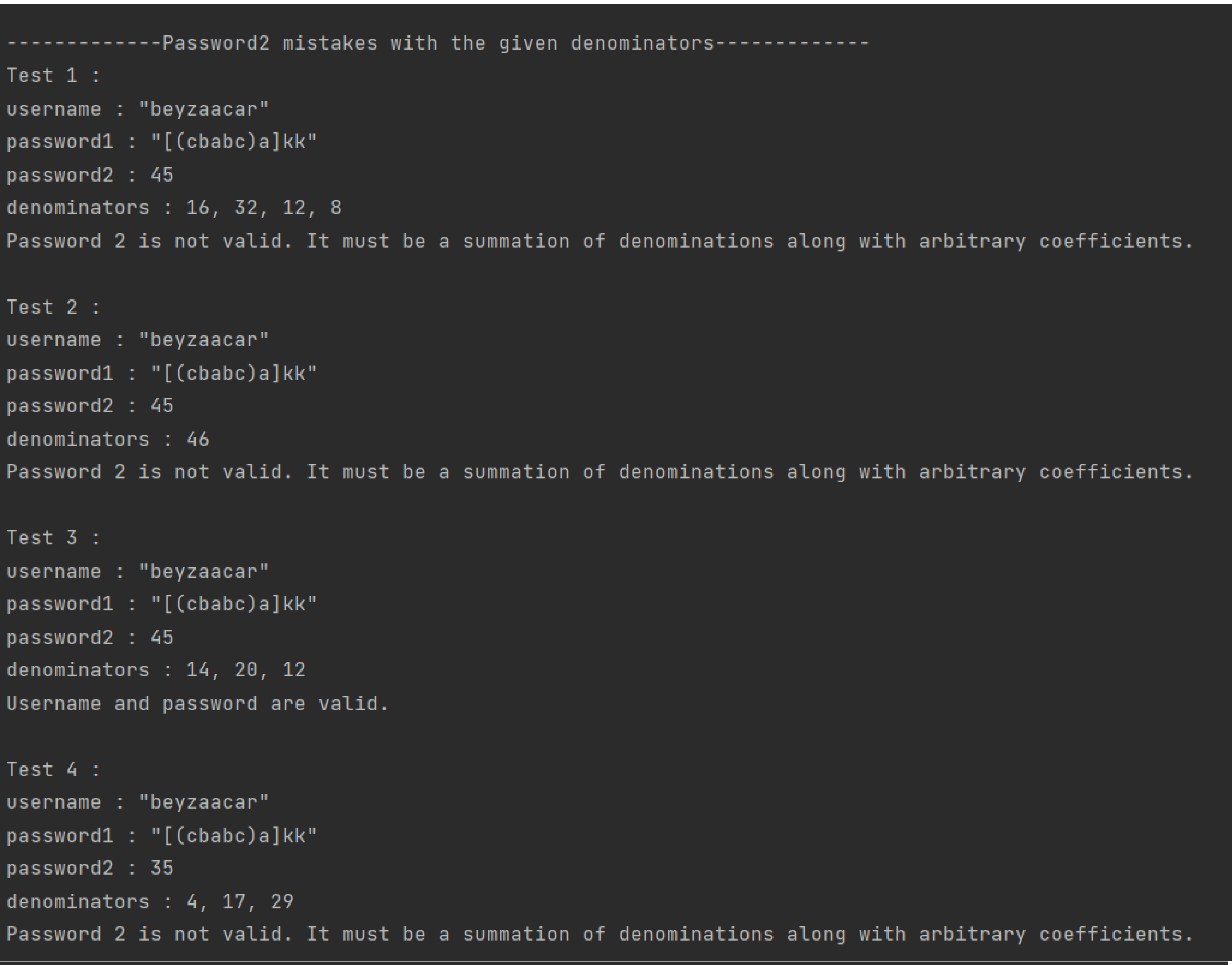
```
-------------Password1 Mistakes-------------
Test 1 :
username : "beyzaacar"
password1 : "(ab)"
password2 : 75
Password 1 is not valid. It must be at least 8 characters long.

Test 2 :
username : "beyzaacar"
password1 : "{kkl(mml){}}"
password2 : 75
Password 1 is not valid. It must contain at least one letter of the username.

Test 3 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk}"
password2 : 75
Password 1 is not valid. It must be balanced with brackets.

Test 4 :
username : "beyzaacar"
password1 : "{bvjs(hha)}"
password2 : 75
Password 1 is not valid. It must be possible to make a palindrome from it.

Test 5 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 75
Username and password are valid.
```

- ***Password2 mistakes***

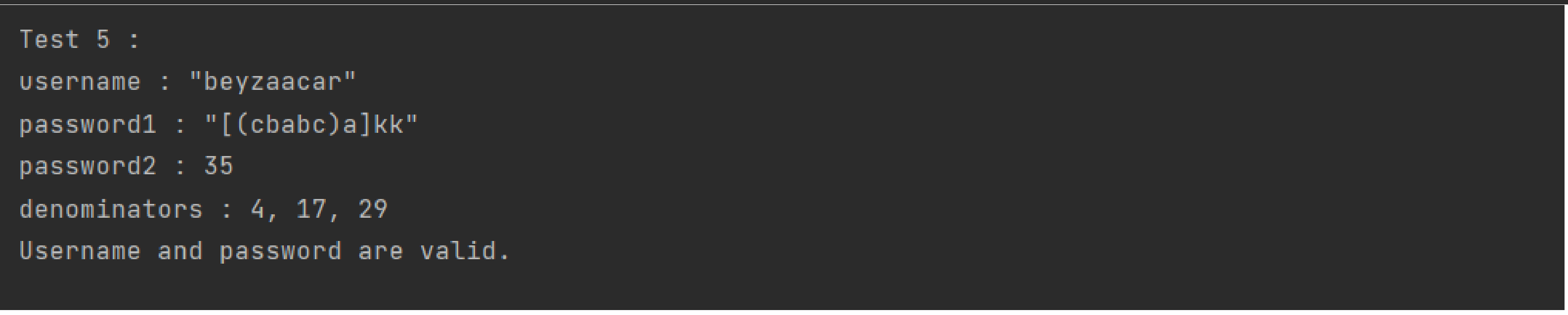


```
-------------Password2 Mistakes-------------
Test 1 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 5
Password 2 is not valid. It must be between 10 and 1000.

Test 2 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 35
Password 2 is not valid. It must be a summation of denominations along with arbitrary coefficients.

Test 4 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 0
Password 2 is not valid. It must be between 10 and 1000.

Test 5 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : -100
Password 2 is not valid. It must be between 10 and 1000.
```

- ***<u>Password2 mistakes with the given denominators</u>***



```
-------------Password2 mistakes with the given denominators-------------
Test 1 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 45
denominators : 16, 32, 12, 8
Password 2 is not valid. It must be a summation of denominations along with arbitrary coefficients.

Test 2 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 45
denominators : 46
Password 2 is not valid. It must be a summation of denominations along with arbitrary coefficients.

Test 3 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 45
denominators : 14, 20, 12
Username and password are valid.

Test 4 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 35
denominators : 4, 17, 29
Password 2 is not valid. It must be a summation of denominations along with arbitrary coefficients.
```



```
Test 5 :
username : "beyzaacar"
password1 : "[(cbabc)a]kk"
password2 : 35
denominators : 4, 17, 29
Username and password are valid.
```