2023

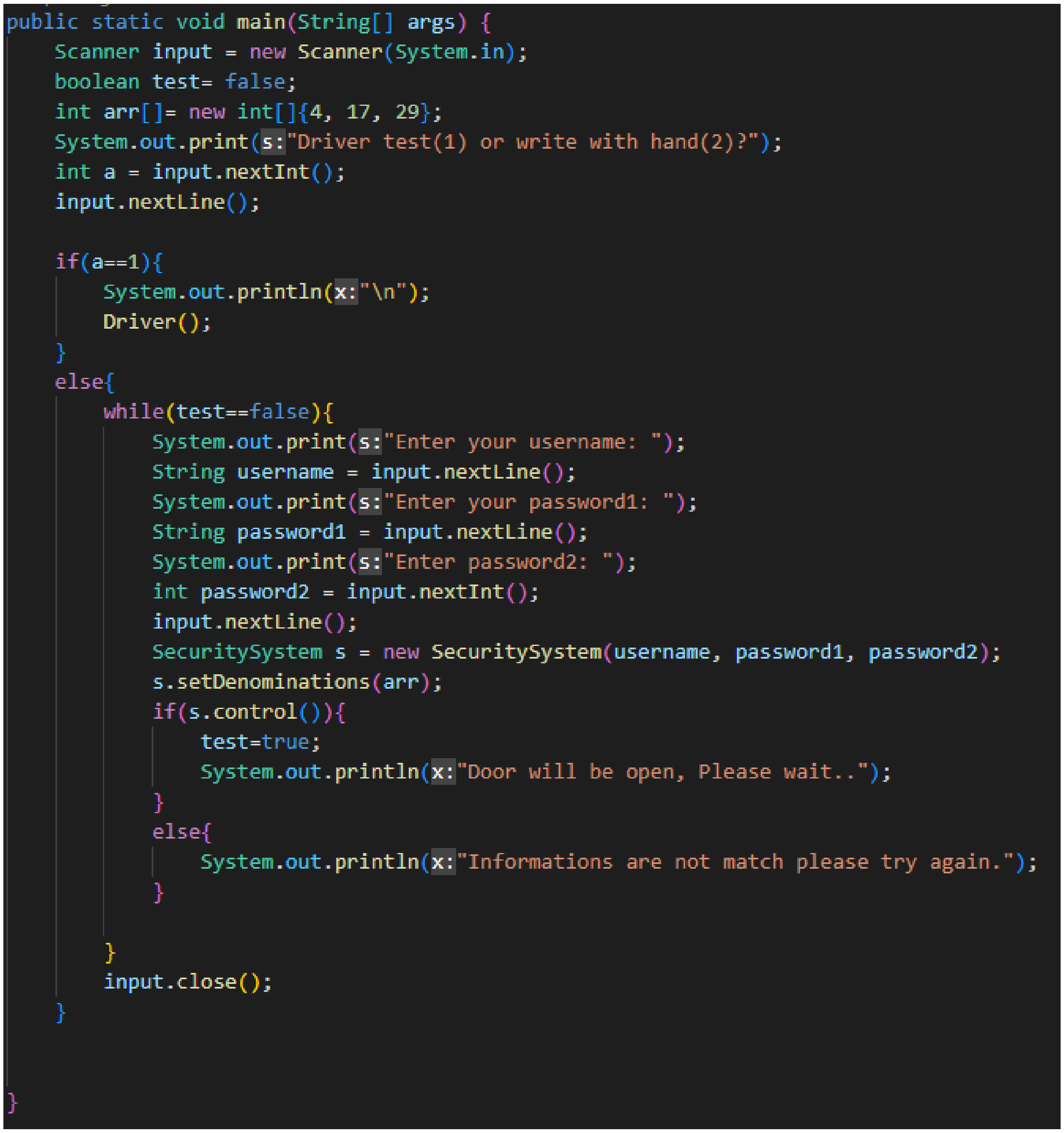
# CSE222 – HW4 Report

ELIFNUR KABALCI
1801042617

|

## FUNCTION EXPRESSIONS

1) Main



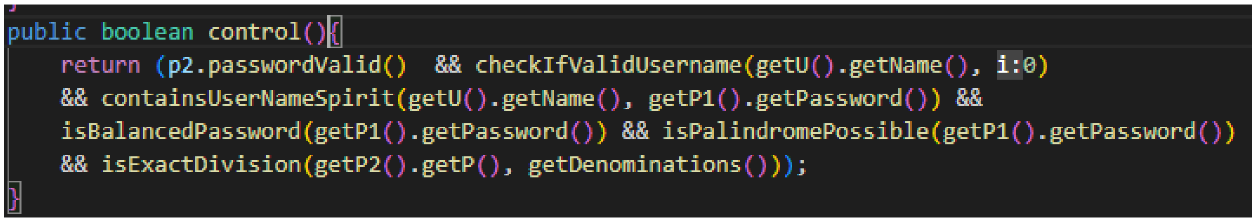
```java
public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    boolean test= false;
    int arr[]= new int[]{4, 17, 29};
    System.out.print(s:"Driver test(1) or write with hand(2)?");
    int a = input.nextInt();
    input.nextLine();

    if(a==1){
        System.out.println(x:"\n");
        Driver();
    }
    else{
        while(test==false){
            System.out.print(s:"Enter your username: ");
            String username = input.nextLine();
            System.out.print(s:"Enter your password1: ");
            String password1 = input.nextLine();
            System.out.print(s:"Enter password2: ");
            int password2 = input.nextInt();
            input.nextLine();
            SecuritySystem s = new SecuritySystem(username, password1, password2);
            s.setDenominations(arr);
            if(s.control()){
                test=true;
                System.out.println(x:"Door will be open, Please wait..");
            }
            else{
                System.out.println(x:"Informations are not match please try again.");
            }

        }
        input.close();
    }


}
```

If the user presses 1, the driver test is active, if he presses two, he can try the values himself. For second part, I get username, password1 and password2 from the user with input. I continue the loop until the user enters the correct information. The results of all the values checked in all functions are printed on the screen respectively.

I will describe the complexities of the functions in the description of each function.

2) General Design
I was going to design the usernames and passwords as elements of a class, but I decided that this was not efficient in some functions, and I set up a class structure for each element, creating elements from each class structure to call the relevant function from the functions defined in the Main class. I made each item with the design principles we were told.

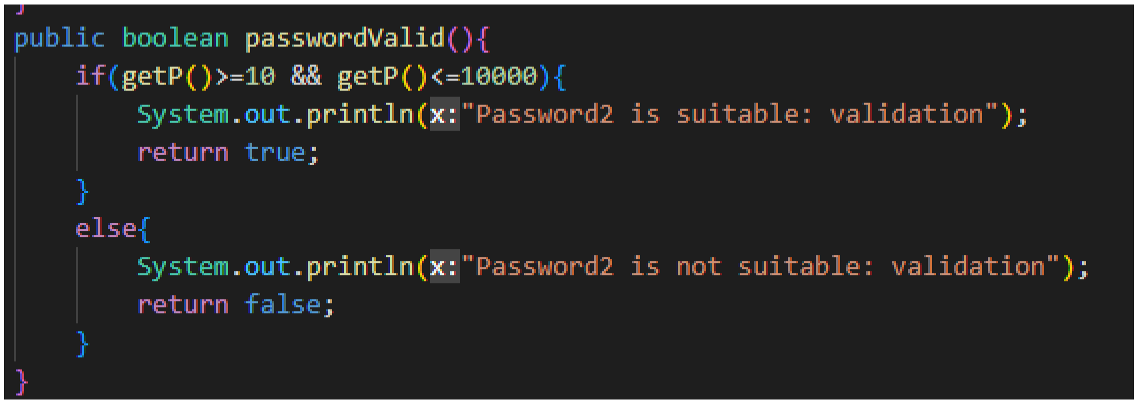
3) Control Function



```
public boolean control(){
    return (p2.passwordValid()  && checkIfValidUsername(getU().getName(), i:0)
    && containsUserNameSpirit(getU().getName(), getP1().getPassword()) &&
    isBalancedPassword(getP1().getPassword()) && isPalindromePossible(getP1().getPassword())
    && isExactDivision(getP2().getP(), getDenominations()));
}
```

All functions must return true before the security guard can open the doors. It is a test function written to call all functions at the same time and return the common value of all of them.

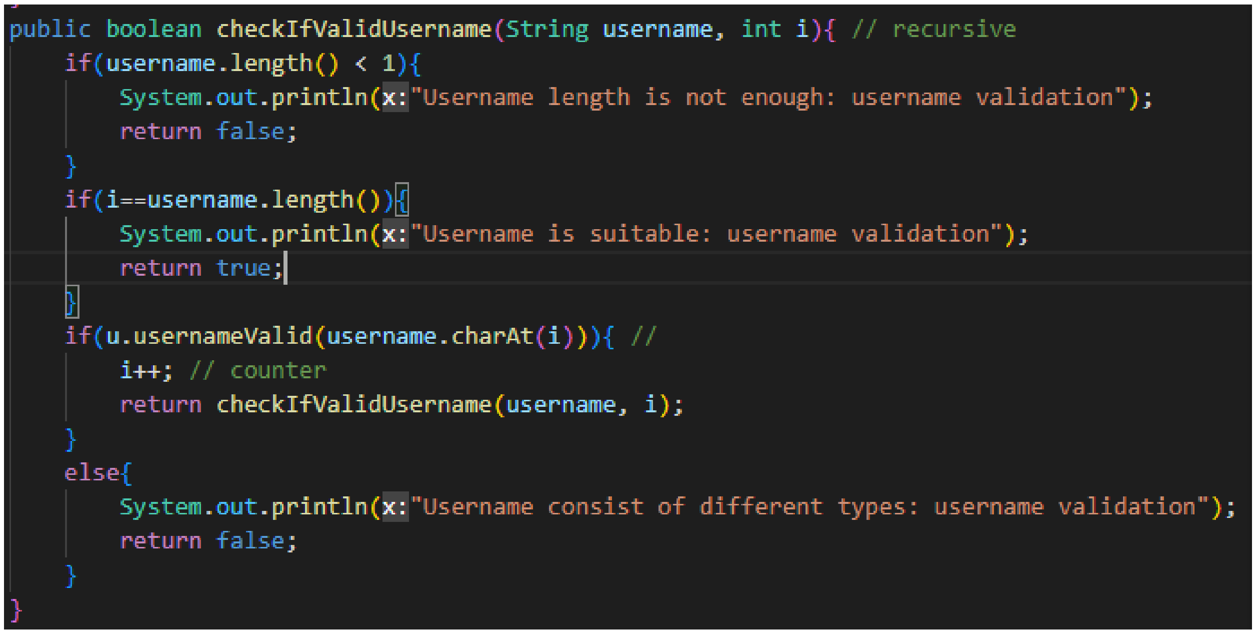
4) Password2 Validation



```
public boolean passwordValid(){
    if(getP()>=10 && getP()<=10000){
        System.out.println(x:"Password2 is suitable: validation");
        return true;
    }
    else{
        System.out.println(x:"Password2 is not suitable: validation");
        return false;
    }
}
```

Here I checked whether the 2nd password received between 10 and 10000 values and printed it out. where the complexity value is O(1). There is no loop or repetition.

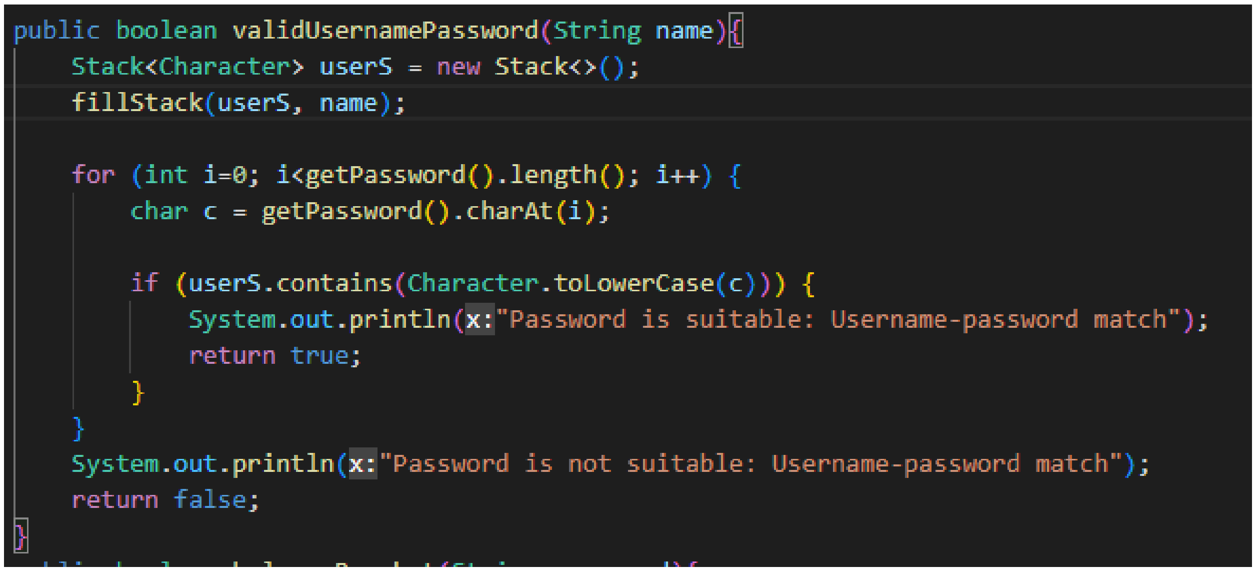
5) CheckIfValidUsername



```java
public boolean checkIfValidUsername(String username, int i){ // recursive
    if(username.length() < 1){
        System.out.println(x:"Username length is not enough: username validation");
        return false;
    }
    if(i==username.length()){
        System.out.println(x:"Username is suitable: username validation");
        return true;
    }
    if(u.usernameValid(username.charAt(i))){ //
        i++; // counter
        return checkIfValidUsername(username, i);
    }
    else{
        System.out.println(x:"Username consist of different types: username validation");
        return false;
    }
}
```

Recursive checking whether the entered username complies with the standards. The amount of time the function is repeated is the length of the username. So the complexity becomes O(n).

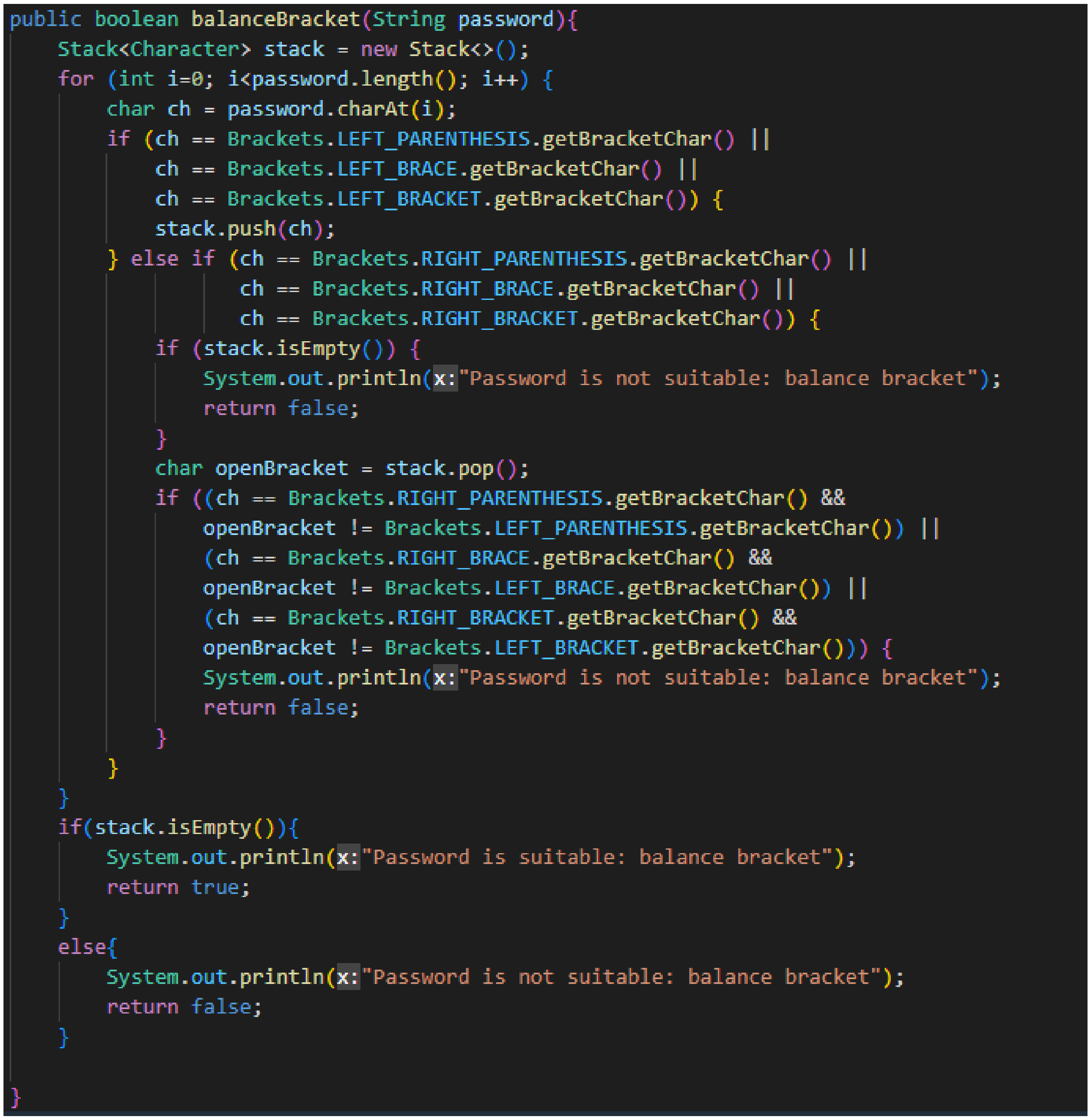
6) ContainUserNameSpirit



```
public boolean validUsernamePassword(String name){
    Stack<Character> userS = new Stack<>();
    fillStack(userS, name);

    for (int i=0; i<getPassword().length(); i++) {
        char c = getPassword().charAt(i);

        if (userS.contains(Character.toLowerCase(c))) {
            System.out.println(x:"Password is suitable: Username-password match");
            return true;
        }
    }
    System.out.println(x:"Password is not suitable: Username-password match");
    return false;
}
```

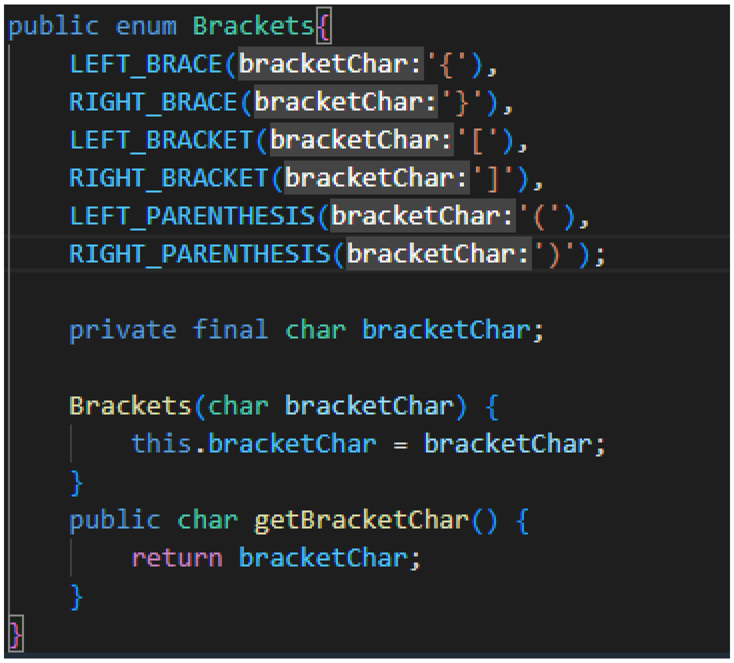
The function in the photo is located inside the password1 class, which is called by the ContainUserNameSpirit function in the main class, which performs the check. In this function, I aimed to examine the stack from the beginning by filling it in reverse. If the password contains a letter from the username, it returns true.
The reversofillstack function will return up to the length of the password, so complexity O(n). And because of the for loop, it becomes O(n). Since these expressions are not nested but side by side, our complexity value is O(n).

7) isBalancedPassword



```java
public boolean balanceBracket(String password){
    Stack<Character> stack = new Stack<>();
    for (int i=0; i<password.length(); i++) {
        char ch = password.charAt(i);
        if (ch == Brackets.LEFT_PARENTHESIS.getBracketChar() ||
            ch == Brackets.LEFT_BRACE.getBracketChar() ||
            ch == Brackets.LEFT_BRACKET.getBracketChar()) {
            stack.push(ch);
        } else if (ch == Brackets.RIGHT_PARENTHESIS.getBracketChar() ||
                   ch == Brackets.RIGHT_BRACE.getBracketChar() ||
                   ch == Brackets.RIGHT_BRACKET.getBracketChar()) {
            if (stack.isEmpty()) {
                System.out.println(x:"Password is not suitable: balance bracket");
                return false;
            }
            char openBracket = stack.pop();
            if ((ch == Brackets.RIGHT_PARENTHESIS.getBracketChar() &&
                openBracket != Brackets.LEFT_PARENTHESIS.getBracketChar()) ||
                (ch == Brackets.RIGHT_BRACE.getBracketChar() &&
                openBracket != Brackets.LEFT_BRACE.getBracketChar()) ||
                (ch == Brackets.RIGHT_BRACKET.getBracketChar() &&
                openBracket != Brackets.LEFT_BRACKET.getBracketChar())) {
                System.out.println(x:"Password is not suitable: balance bracket");
                return false;
            }
        }
    }
    if(stack.isEmpty()){
        System.out.println(x:"Password is suitable: balance bracket");
        return true;
    }
    else{
        System.out.println(x:"Password is not suitable: balance bracket");
        return false;
    }

}
```

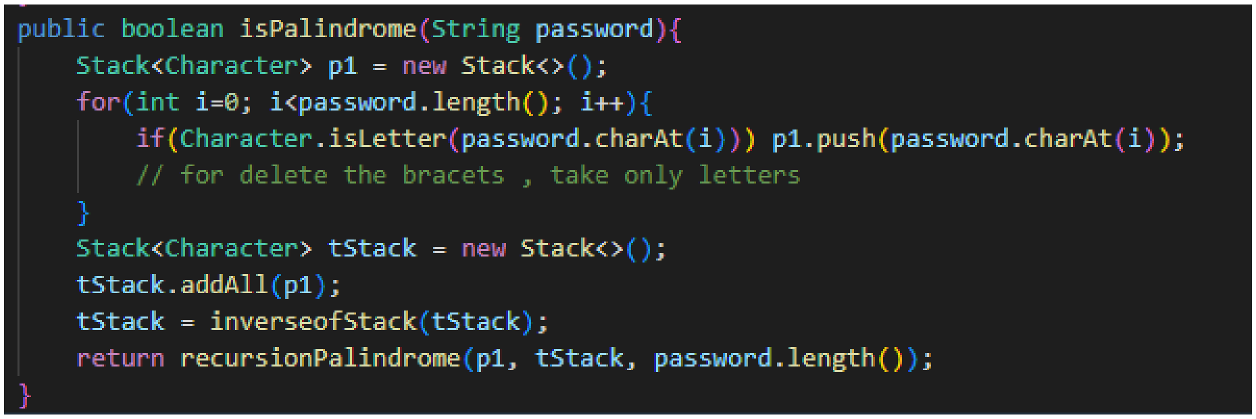
In this function, I looked to see if the parentheses were balanced, the complexity value O(n) due to the For loop.



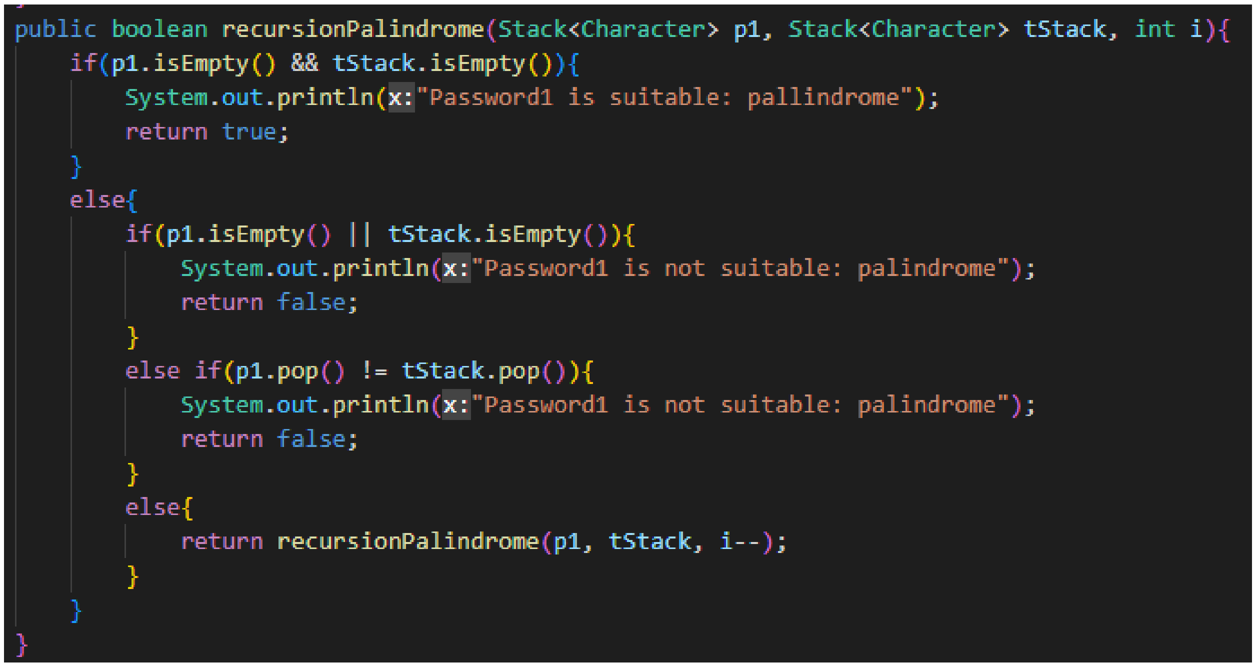
```java
public enum Brackets{
    LEFT_BRACE(bracketChar:'{'),
    RIGHT_BRACE(bracketChar:'}'),
    LEFT_BRACKET(bracketChar:'['),
    RIGHT_BRACKET(bracketChar:']'),
    LEFT_PARENTHESIS(bracketChar:'('),
    RIGHT_PARENTHESIS(bracketChar:')');

    private final char bracketChar;

    Brackets(char bracketChar) {
        this.bracketChar = bracketChar;
    }
    public char getBracketChar() {
        return bracketChar;
    }
}
```

I did not find it appropriate to use parentheses directly in the OOP design. So I created an enum where I defined all of the parentheses. That's how I called it in the function.

8) isPalindromePossible



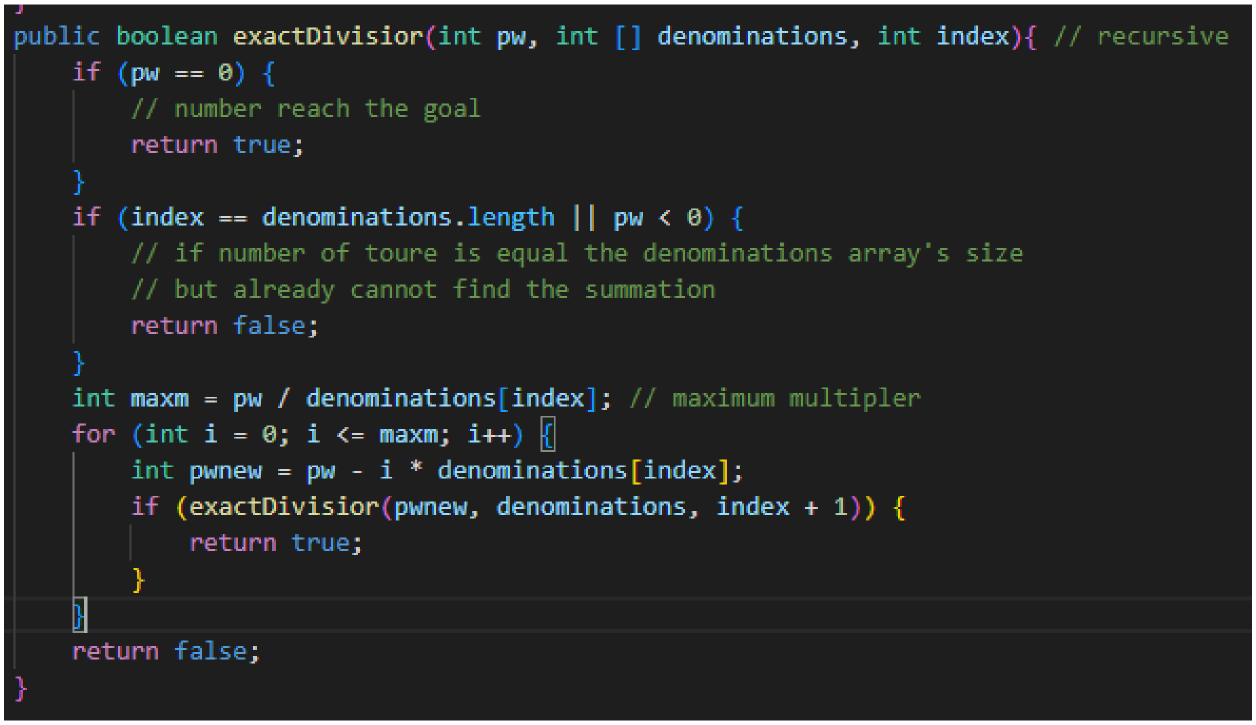
```java
public boolean isPalindrome(String password){
    Stack<Character> p1 = new Stack<>();
    for(int i=0; i<password.length(); i++){
        if(Character.isLetter(password.charAt(i))) p1.push(password.charAt(i));
        // for delete the bracets , take only letters
    }
    Stack<Character> tStack = new Stack<>();
    tStack.addAll(p1);
    tStack = inverseofStack(tStack);
    return recursionPalindrome(p1, tStack, password.length());
}
```



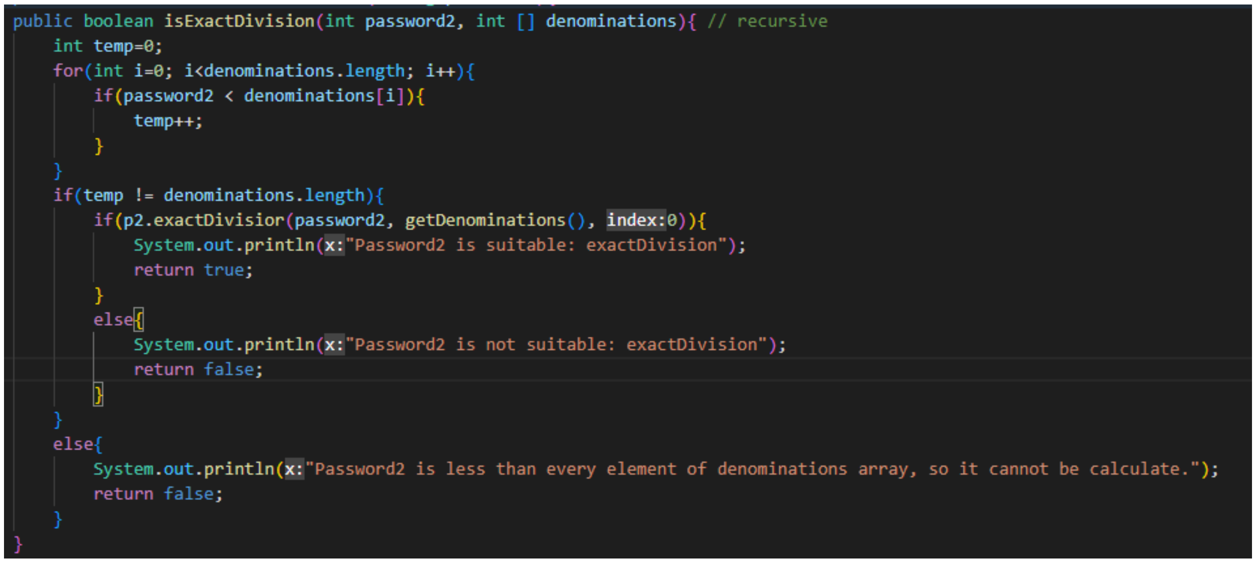
```java
public boolean recursionPalindrome(Stack<Character> p1, Stack<Character> tStack, int i){
    if(p1.isEmpty() && tStack.isEmpty()){
        System.out.println(x:"Password1 is suitable: pallindrome");
        return true;
    }
    else{
        if(p1.isEmpty() || tStack.isEmpty()){
            System.out.println(x:"Password1 is not suitable: palindrome");
            return false;
        }
        else if(p1.pop() != tStack.pop()){
            System.out.println(x:"Password1 is not suitable: palindrome");
            return false;
        }
        else{
            return recursionPalindrome(p1, tStack, i--);
        }
    }
}
```

The complexity of the ispallindrome function is O(n). because of the for loop. The recursion pallindrome function loops at most the size of the stack. So there is also the complexity value O(n).

9) isExactDivision



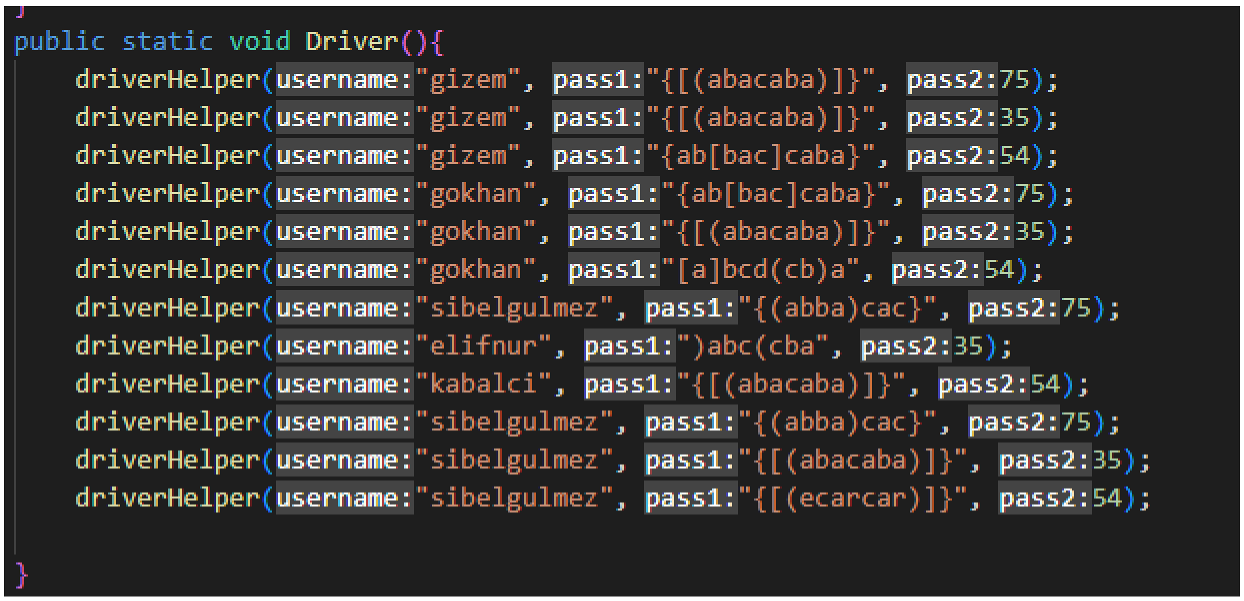
```java
public boolean exactDivision(int pw, int [] denominations, int index){ // recursive
    if (pw == 0) {
        // number reach the goal
        return true;
    }
    if (index == denominations.length || pw < 0) {
        // if number of toure is equal the denominations array's size
        // but already cannot find the summation
        return false;
    }
    int maxm = pw / denominations[index]; // maximum multipler
    for (int i = 0; i <= maxm; i++) {
        int pwnew = pw - i * denominations[index];
        if (exactDivision(pwnew, denominations, index + 1)) {
            return true;
        }
    }
    return false;
}
```

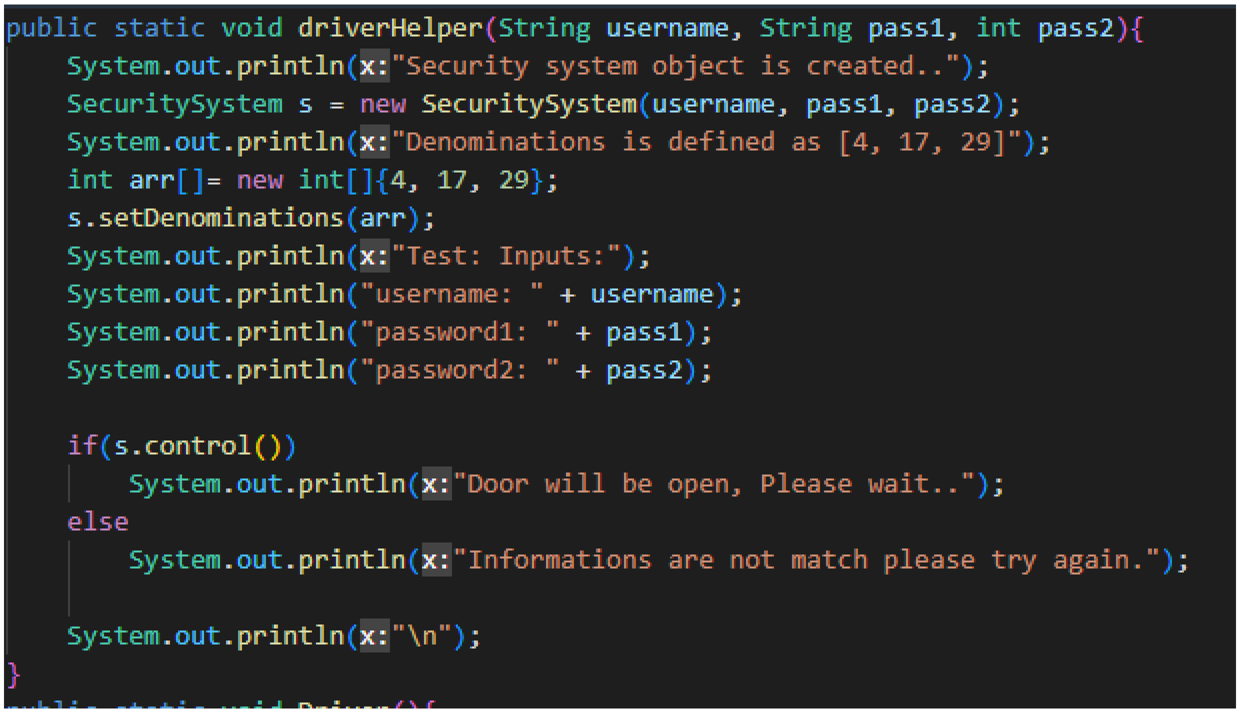

```java
public boolean isExactDivision(int password2, int [] denominations){ // recursive
    int temp=0;
    for(int i=0; i<denominations.length; i++){
        if(password2 < denominations[i]){
            temp++;
        }
    }
    if(temp != denominations.length){
        if(p2.exactDivision(password2, getDenominations(), index:0)){
            System.out.println(x:"Password2 is suitable: exactDivision");
            return true;
        }
        else{
            System.out.println(x:"Password2 is not suitable: exactDivision");
            return false;
        }
    }
    else{
        System.out.println(x:"Password2 is less than every element of denominations array, so it cannot be calculate.");
        return false;
    }
}
```

I defined the denominator array in Main and sent its values in the first definition. I checked to see if we could get password2 with these values. First of all, I did a check in the main class. In this check, if the value of password is less than all the elements in the array, it does not enter the calculation function. Thus, power is not wasted unnecessarily. While this check is being done, all the elements of the array are scanned. So the complexity value here is O(n).

When we enter the calculation function, it returns the maximum amount of the size of the stack sent. Hence the complexity value O(n).

10) Driver

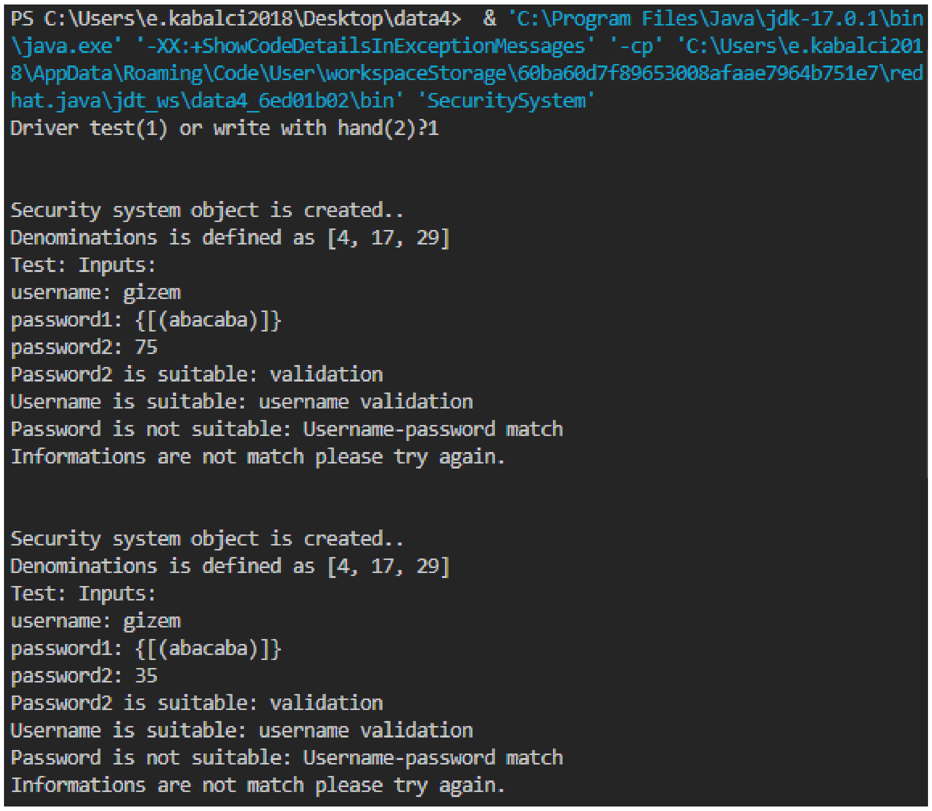

```java
public static void Driver(){
    driverHelper(username:"gizem", pass1:"{[(abacaba)]}", pass2:75);
    driverHelper(username:"gizem", pass1:"{[(abacaba)]}", pass2:35);
    driverHelper(username:"gizem", pass1:"{ab[bac]caba}", pass2:54);
    driverHelper(username:"gokhan", pass1:"{ab[bac]caba}", pass2:75);
    driverHelper(username:"gokhan", pass1:"{[(abacaba)]}", pass2:35);
    driverHelper(username:"gokhan", pass1:"[a]bcd(cb)a", pass2:54);
    driverHelper(username:"sibelgulmez", pass1:"{(abba)cac}", pass2:75);
    driverHelper(username:"elifnur", pass1:")abc(cba", pass2:35);
    driverHelper(username:"kabalci", pass1:"{[(abacaba)]}", pass2:54);
    driverHelper(username:"sibelgulmez", pass1:"{(abba)cac}", pass2:75);
    driverHelper(username:"sibelgulmez", pass1:"{[(abacaba)]}", pass2:35);
    driverHelper(username:"sibelgulmez", pass1:"{[(ecarcar)]}", pass2:54);

}
```



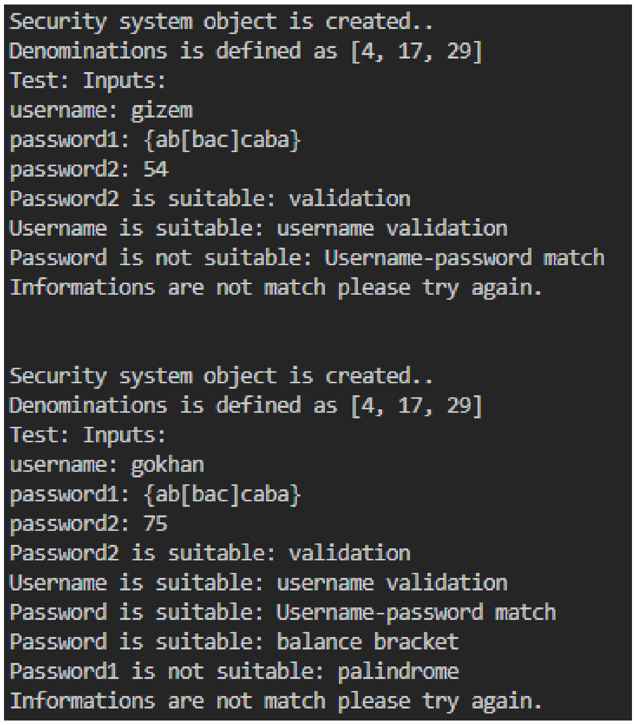
```
public static void driverHelper(String username, String pass1, int pass2){
    System.out.println(x:"Security system object is created..");
    SecuritySystem s = new SecuritySystem(username, pass1, pass2);
    System.out.println(x:"Denominations is defined as [4, 17, 29]");
    int arr[]= new int[]{4, 17, 29};
    s.setDenominations(arr);
    System.out.println(x:"Test: Inputs:");
    System.out.println("username: " + username);
    System.out.println("password1: " + pass1);
    System.out.println("password2: " + pass2);

    if(s.control())
        System.out.println(x:"Door will be open, Please wait..");
    else
        System.out.println(x:"Informations are not match please try again.");

    System.out.println(x:"\n");
}
```

A fixed function that is tested when manually entering data is not desired. The outputs are as follows:

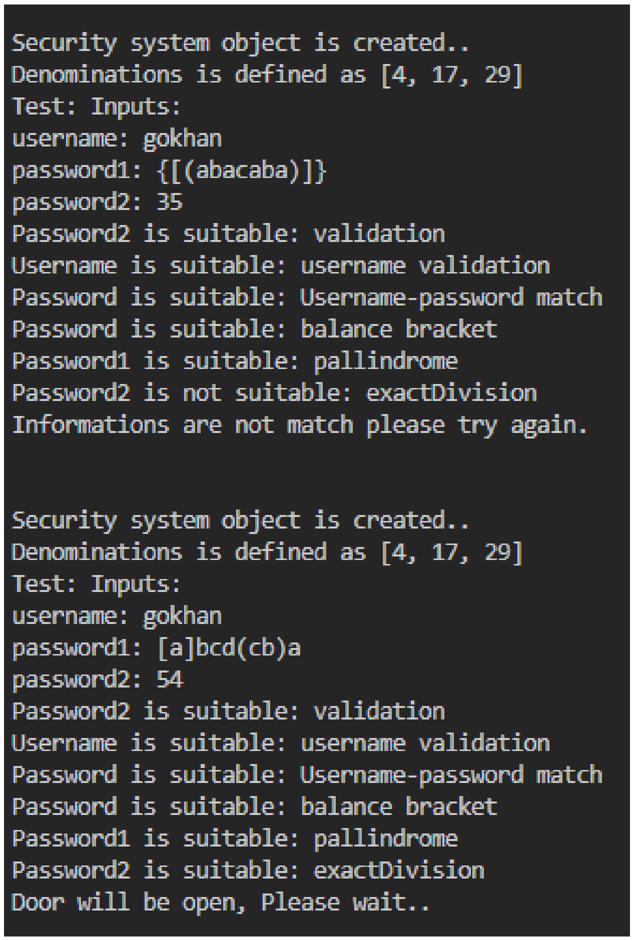
Driver Tests Result:



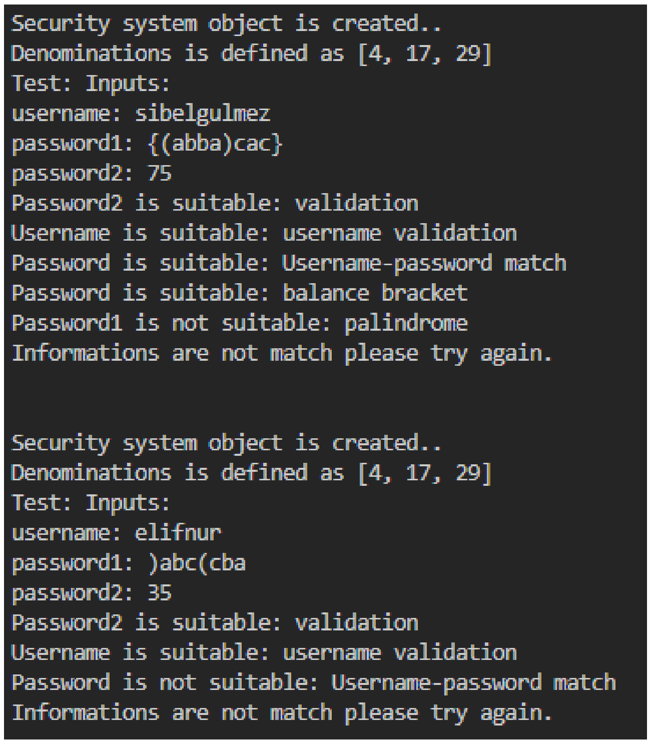
```
PS C:\Users\e.kabalci2018\Desktop\data4>  & 'C:\Program Files\Java\jdk-17.0.1\bin
\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\e.kabalci201
8\AppData\Roaming\Code\User\workspaceStorage\60ba60d7f89653008afaae7964b751e7\red
hat.java\jdt_ws\data4_6ed01b02\bin' 'SecuritySystem'
Driver test(1) or write with hand(2)?1


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: gizem
password1: {[(abacaba)]}
password2: 75
Password2 is suitable: validation
Username is suitable: username validation
Password is not suitable: Username-password match
Informations are not match please try again.


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: gizem
password1: {[(abacaba)]}
password2: 35
Password2 is suitable: validation
Username is suitable: username validation
Password is not suitable: Username-password match
Informations are not match please try again.
```



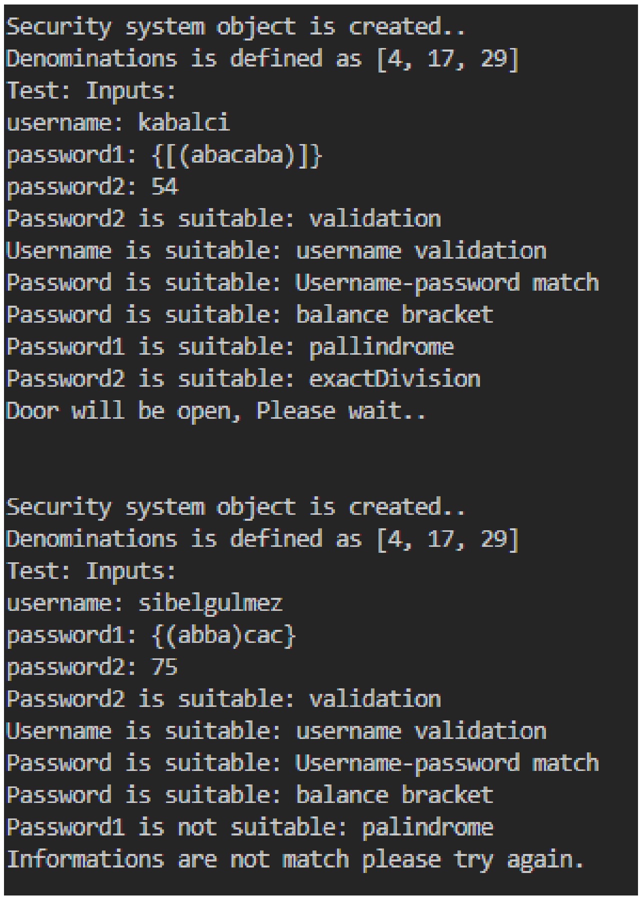
```
Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: gizem
password1: {ab[bac]caba}
password2: 54
Password2 is suitable: validation
Username is suitable: username validation
Password is not suitable: Username-password match
Informations are not match please try again.


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: gokhan
password1: {ab[bac]caba}
password2: 75
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is not suitable: palindrome
Informations are not match please try again.
```



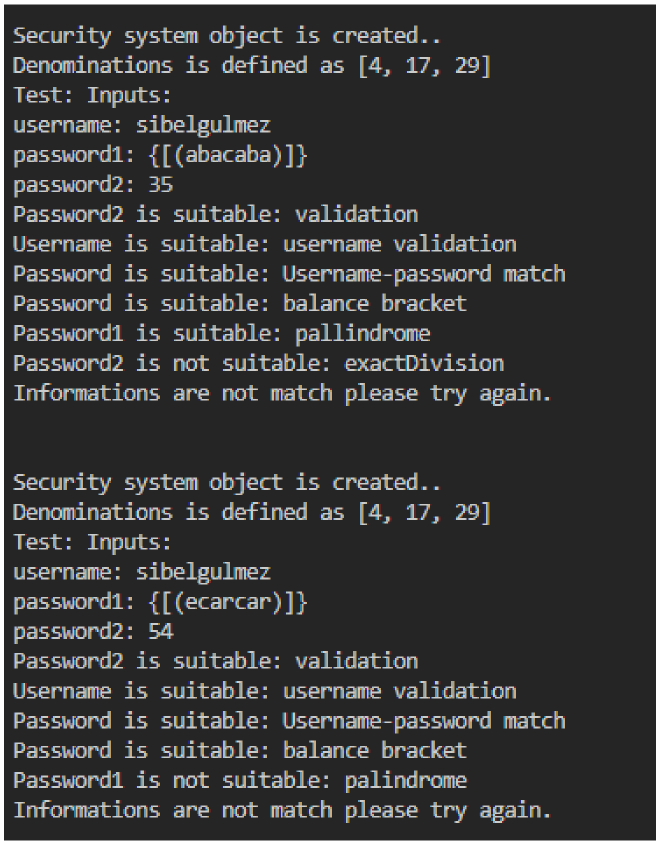
```
Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: gokhan
password1: {[(abacaba)]}
password2: 35
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is suitable: pallindrome
Password2 is not suitable: exactDivision
Informations are not match please try again.


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: gokhan
password1: [a]bcd(cb)a
password2: 54
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is suitable: pallindrome
Password2 is suitable: exactDivision
Door will be open, Please wait..
```



```
Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: sibelgulmez
password1: {(abba)cac}
password2: 75
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is not suitable: palindrome
Informations are not match please try again.


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: elifnur
password1: )abc(cba
password2: 35
Password2 is suitable: validation
Username is suitable: username validation
Password is not suitable: Username-password match
Informations are not match please try again.
```

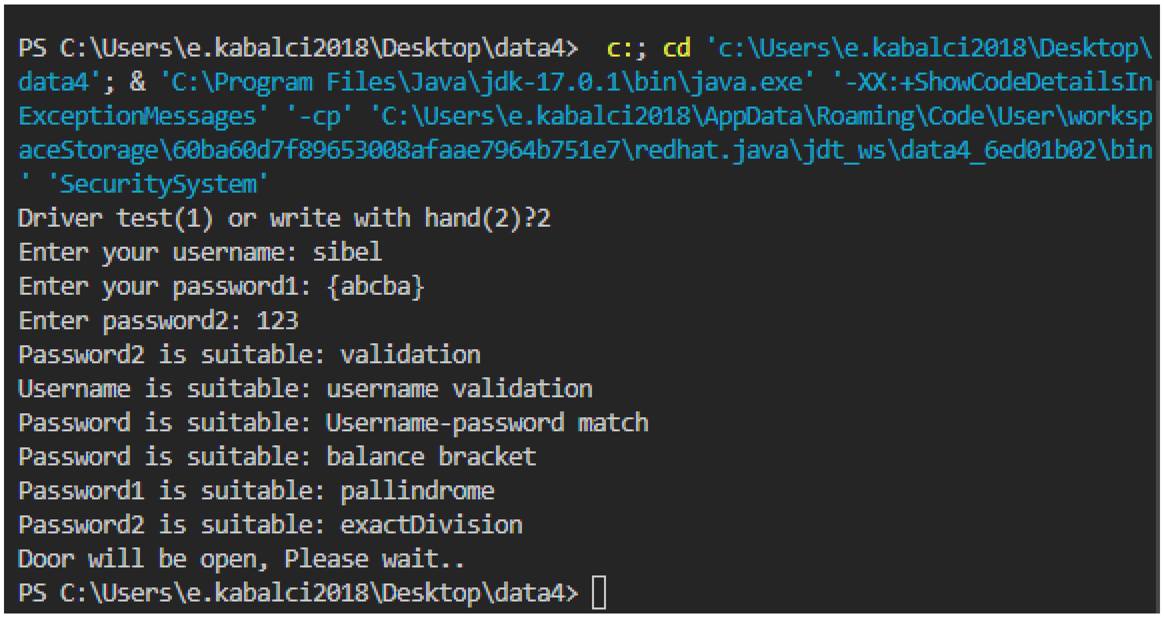


```
Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: kabalci
password1: {[(abacaba)]}
password2: 54
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is suitable: pallindrome
Password2 is suitable: exactDivision
Door will be open, Please wait..


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: sibelgulmez
password1: {(abba)cac}
password2: 75
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is not suitable: palindrome
Informations are not match please try again.
```

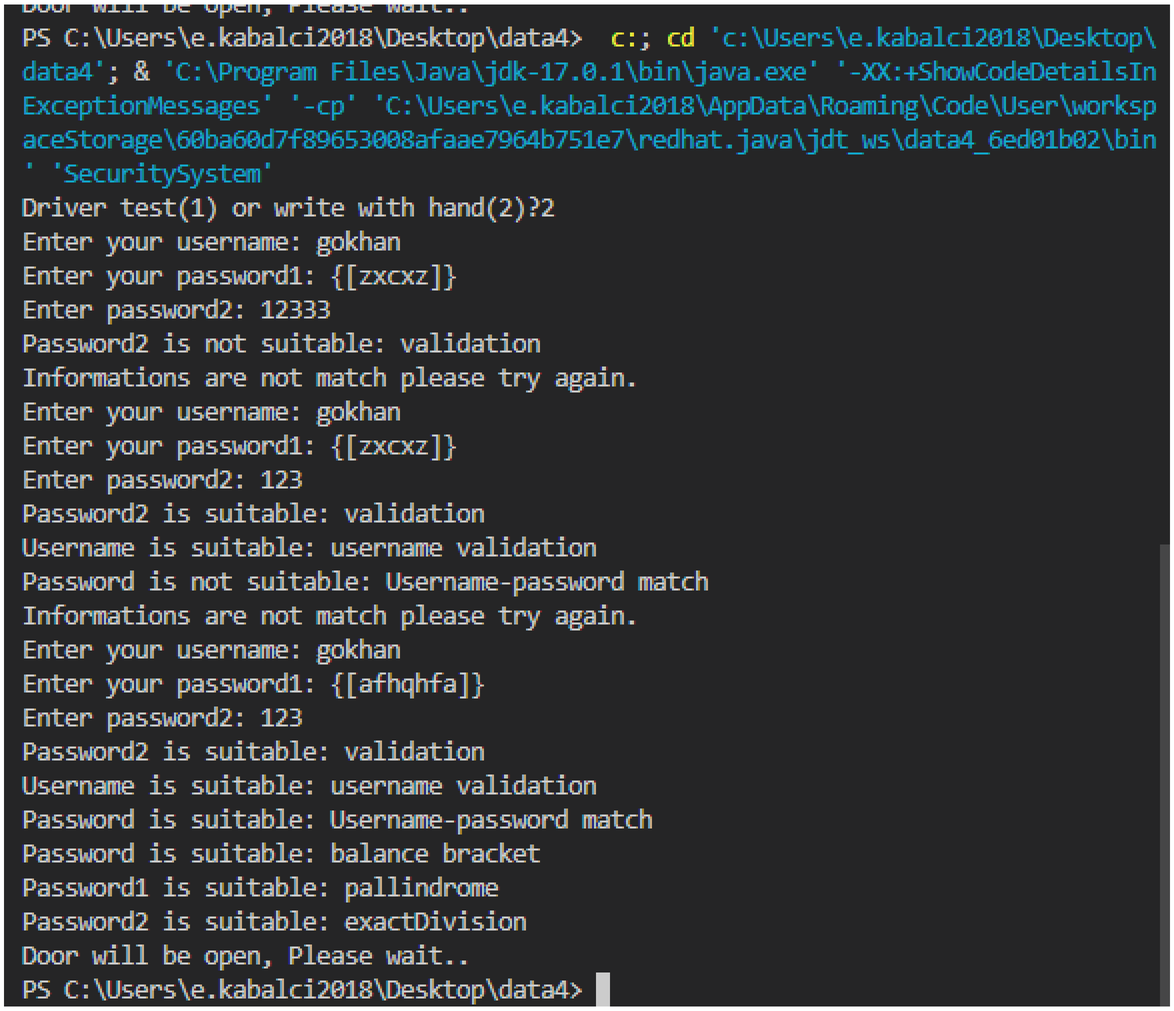


```
Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: sibelgulmez
password1: {[(abacaba)]}
password2: 35
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is suitable: pallindrome
Password2 is not suitable: exactDivision
Informations are not match please try again.


Security system object is created..
Denominations is defined as [4, 17, 29]
Test: Inputs:
username: sibelgulmez
password1: {[(ecarcar)]}
password2: 54
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is not suitable: palindrome
Informations are not match please try again.
```

Hand Written Test Results:



```
PS C:\Users\e.kabalci2018\Desktop\data4>  c:; cd 'c:\Users\e.kabalci2018\Desktop\
data4'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsIn
ExceptionMessages' '-cp' 'C:\Users\e.kabalci2018\AppData\Roaming\Code\User\worksp
aceStorage\60ba60d7f89653008afaae7964b751e7\redhat.java\jdt_ws\data4_6ed01b02\bin
' 'SecuritySystem'
Driver test(1) or write with hand(2)?2
Enter your username: sibel
Enter your password1: {abcba}
Enter password2: 123
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is suitable: pallindrome
Password2 is suitable: exactDivision
Door will be open, Please wait..
PS C:\Users\e.kabalci2018\Desktop\data4> 
```



```
PS C:\Users\e.kabalci2018\Desktop\data4>  c:; cd 'c:\Users\e.kabalci2018\Desktop\
data4'; & 'C:\Program Files\Java\jdk-17.0.1\bin\java.exe' '-XX:+ShowCodeDetailsIn
ExceptionMessages' '-cp' 'C:\Users\e.kabalci2018\AppData\Roaming\Code\User\worksp
aceStorage\60ba60d7f89653008afaae7964b751e7\redhat.java\jdt_ws\data4_6ed01b02\bin
' 'SecuritySystem'
Driver test(1) or write with hand(2)?2
Enter your username: gokhan
Enter your password1: {[zxcxz]}
Enter password2: 12333
Password2 is not suitable: validation
Informations are not match please try again.
Enter your username: gokhan
Enter your password1: {[zxcxz]}
Enter password2: 123
Password2 is suitable: validation
Username is suitable: username validation
Password is not suitable: Username-password match
Informations are not match please try again.
Enter your username: gokhan
Enter your password1: {[afhqhfa]}
Enter password2: 123
Password2 is suitable: validation
Username is suitable: username validation
Password is suitable: Username-password match
Password is suitable: balance bracket
Password1 is suitable: pallindrome
Password2 is suitable: exactDivision
Door will be open, Please wait..
PS C:\Users\e.kabalci2018\Desktop\data4>
```