**Process Documentation**

**Final Project Submission: Authentication – Week 7**

**Michael Mingrone**

**IT-145-R2919 Found in App Development 18EW2**

**12/16/18**

Contents

[Problem Statement/Scenario 3](#_Toc532398179)

[Overall Process 3](#_Toc532398180)

[Pseudocode 5](#_Toc532398181)

[Methods and Classes 10](#_Toc532398182)

[Error Documentation 10](#_Toc532398183)

[Solution Documentation 10](#_Toc532398184)

[References 12](#_Toc532398185)

# Problem Statement/Scenario

The problem, more so the task, was to design and implement a way for employees of a zoo to review information about their job specifically, based on their input. The system needed to verify that the user was even in the system, and that their password matched the one on file. Given the correct information, the system then needed to display the correct role information.

# Overall Process

This was not nearly as bad as I initially thought. We were expected to make a console application. In some ways, this was easier because I didn’t have to worry as much about loop management. However, it was harder because I needed to set up a GUI (something I hadn’t really done before).

I started planning in my head first. I needed a way to take username and password and get a returned file, basically. My first thought was to use a bunch of IF’s. Shortly after, I realized that all I was doing was using a (key,value) pair; the key being the username and the value being the role. I was still left with password though. Next day I had a *great* idea. Concatenation. I thought, instead of the username being unique to one person, won’t a combination of their username/password also be unique? The (key,value) data structure is commonly referred to as a dictionary.

I started with the classes, more specifically, roles. Creating the classes first I feel was tremendously helpful. It was at the start of this I realized I needed another dictionary: one that showed the appropriate text file given the role. In my constructor for this it only made sense to populate my dictionary with the proper ‘.txt’s. After this I made a class method that returns the **private** aforementioned dictionary entry given a role. This ‘getter’ method is necessary because we can’t access private members from outside the class.

Next, I took a break from classes and dove into creating the GUI. I know I needed 2 ‘screens’, so I created 2 panels. Like I said, creating the GUI was pretty straightforward. It was mostly just drag and drop, since I was using a jFrame.

The class for users came next. As I mentioned, I needed a dictionary, so I declared that first. Writing my constructor was easy. I only needed one file! On each line, I grabbed 3 things: the username, that user’s password represented as an MD5 hash, and finally, that user’s role. I added the first two together and made that a key in the dictionary. Then, I made the role a value. They way it ends up working, is if an incorrect key is used (wrong username or password), ‘null’ is the value returned. If valid, then that user’s role. My next method just returns the **private** dictionary member. Next, I don’t know why, this could’ve easily been in main, I put the ‘getMD5Hash method.

Next, I worked on main. Because most things occur outside main, main just calls them. One thing that was important, was the first thing I do in main is setup my user’s and role’s classes.as well as the login counter.

Lastly, one of the most important, I setup all the methods that pressing buttons triggers. I setup a quit method taking a Boolean because you can exit in two ways. I also setup two ways to login as pressing the button or pressing enter with the password box in focus logs you in. Here is where all my authentication and display logic lay. It was likely the most extensive method of the bunch, too. Lastly, I created a ‘show message’ method as previous coders told me I should never repeat code. I also, just for fun, play an alert sound on failure.

# Pseudocode

This was my old pseudocode, pre-gui stuff.

NEW

INITIALIZE – logonAttempts to 0

DO

GET – String username

GET – String password

CONVERT – password to a md5 hash (create a class for md5)

CHECK – the equality of the passwords to the given credentials file

IF – equal

DISPLAY – the correct ‘role’ file text

Create class of roles

PROMPT – Quit?

IF – yes

Break

ELSE –

INCREMENT – logonAttempts

WHILE – logonAttempts <= 3

Here is my new and improved pseudocode

Michael Mingrone (using notepad++)

Option 1: Authentication System

AUTHENTICATE:

INITIALIZE our classes, user and roles

INITIALIZE login attempts to 3 (much better as a code reader. This is intuitive to

count down and if we hit 0, quit)

CREATE GUI

CREATE quit method

DISPLAY an alert that you will be closing

END program

CREATE login method

IF credentials match

DISPLAY contents of the appropriate rolefile

RESET login attempts to 3

ELSE

DECREMENT loginAttempts

IF loginAttempts is more than 0

PLAY an alert sound

DISPLAY how many tries they have left

ELSE

DISPLAY an alert telling the user that they ran out

CALL quit method

END IF

END IF

BUTTON\_QUIT call the quit method

PASSWORD\_TEXTFIELD on enter press, call login

BUTTON\_LOGIN call login method

BUTTON\_SIGNOUT on press, return to login screen

END\_AUTHENTICATE

CLASS ROLE

DECLARE our dictionary (a dictionary is a set of key value pairs)

CALL CONSTRUCTOR (only happens once per instance)

LOAD proper directory where files are

GET all filenames in directory as string array

FOR each string in array

IF it ends in '.txt' and is not the 'users.txt' file

ADD string to valid list

END IF

END FOR

FOR each entry in valid list

DECLARE String Lines

READ and copy each line to Lines

ADD entry to dictionary

(The file name minus.txt is the key, Lines is now the value)

END FOR

END\_CONSTRUCTOR

GET ROLEINFO

when passed the 'role', return the answer by accessing the dictionary.

END

END\_CLASS

CLASS USER

DECLARE dictionary for use in authenticating us

CALL CONSTRUCTOR

LOAD the users file

WHILE a next line exists

DECLARE 2 strings, one for username/pass, one for role

READ line from file

SPLIT line by \t(tab) delimiters

ASSIGN first string to indexes 0 + 1 in resulting array

ASSIGN second string to role

ADD to dictionary (string1 is key, string2 is value)

ENDWHILE

END\_CONSTRUCTOR

GET ROLE (given user+pass)

RETURN the value in the dictionary corresponding to that key

END

GET MD5HASH (this could've been somewhere else the way I implemented it)

CONVERT our plain text password to one as an md5 hash

END

ENDCLASS

# Methods and Classes

Once again, I was told to never repeat code (I feel like I did with my try..catches though). When I first started out, the ‘quit’ method was directly attached to the quit button. As time progressed, and need for a second way to quit arose, I found myself making a separate method for quit, that is called from multiple places. My classes, I feel, were well utilized too. They read files at the start of each time of execute.

# Error Documentation

Honestly not too much troubled me, but here is what did:

File management – files were definitely the hardest hurdles for me to overcome, mainly because I hadn’t used try..catch blocks before. Oftentimes, I just listened to NetBeans while in development.

jTextFields – These were a pain for me and sucked up a lot of my time. Let me rewind, when first attempting this project, I tried to create everything programmatically. So it was creating them by text that got the best of me.

New roles – At first, I made the mistake of capitalizing my role file names. It was for this reason that I was blind to the easy solution to this.

# Solution Documentation

File management – File management turned out to be easy when I understood try..catch blocks. They work like this:

TRY

Do code

Do more code

Run into an error

Immediately jump to catch block and execute that code.

jTextFields – As I said, these gave me a headache. The solution was rather simple. I just didn’t do it. I switched to the GUI NetBeans provides.

New roles – This, at first, was painful to me. But, I only have myself to blame for all the trouble caused to me. After Mr. Darrow pointed it out, I saw that I wasn’t minding case sensitivity.

Other – I decided (for fun) to add a new style to this (“Java GUI: System Default Look And Feel”,April 2013). Along with this, I got a monumental amount of help from Stack Overflow (n.d.).

All in all, I think my project came out pretty darned good, considering it was supposed to be in the console.

NOTE: You’re free to use my project as an example, if you wish.

# References

Java GUI: System Default Look And Feel (April 2013) Retrieved from: https://www.youtube.com/watch?v=LY13j0ND-00

Stack Overflow (n.d.) Retrieved from: https://stackoverflow.com/