Final Project Zoo Authentication

Miguel Ovalles

Southern New Hampshire University

Final Project

Program selected: Option 1, Authentication System

Statement / Scenario:

The authentication system program will ask a for a user to input their credentials when prompted and then confirm the user’s credentials are correct. The credentials will be checked against the provided information for each role and user as well as the MD5 hash file that was provided. If all the information is true, then a message will be generated for the user that logged in. Each user will be allowed to log out by typing in the word ‘quit’. Users will only be allowed three login attempts before being prompted to exit the program and start over. I am looking to create two additional classes aside from the main class. One will store strings and input info and the other will be for combining blocks of print text for the user in different scenarios.

Overall Process:

For this project I began by writing the pseudocode. My original draft was a bit basic, mostly just stating what I wanted to accomplish in a general fashion. Along the lines of prompt, the user to login, check password, print text and so forth. As I reviewed it and thought about it more, I modified it to include more detailed descriptions. I then wrote the original program to match my pseudocode, which did not include multiple classes. This was before I had a better understanding of classes, methods and objects. I was able to get the program to work as one long code without the additional classes. This is where it got a bit complicated. I know I needed to add more than one class to the program as stated in the final project guidelines and rubric. I then went back and using google, YouTube and Stack Overflow I got a better understanding of how classes are meant to be used. I am certainly not an expert yet, but I believe I have a basic understanding of them now. After doing some research and learning or understanding classes a bit more I updated my Pseudocode to include the classes and some additional information. I also updated the layout of the code. I then went back to the code itself and slowly modified the code by creating two classes and then had different information stored under them. As I added to the classes, I updated the processing part of the code to call the information in the classes. I reviewed the code periodically to ensure I did not have redundant information and that the program ran successfully. I then modified it slowly and cleaned it up as I went along. I left some unused items in one of the classes because I was initially trying to get the program to review the role of the user after they successfully logged in but was not able to get this work correctly. I then had to go back and was able to get the program to look at the location of the files and search for the file with the role name and extension and read the file then print the text within the file. I took a bit of a newbie approach and worked a little backwards at some points of the project, but all in all it works. I had set up the program to have the classes at the end of the program and then moved them up in the program to make it easier to add and update as needed.

Methods and Classes:

Created class for prompts that will be called within the program and contains text that will be printed for the user. I created a second class for the initializing the user inputs. Also using the public class for the processing portion of the program.

Error and Solution Documentation:

My errors occurred mostly after the user logged in successfully. Once I got to this portion of the program, I was not able to get the program to behave correctly when reading in the file for that user’s role. I was able to get the text to print but it was not laying it out correctly. I could not figure out how to get it to work. I attempted make a few changes and then ended up scrapping that part and re-writing that part. I had to sit and think of another way to read in the file for the user role. I ended up having it look at the file name and extension. The program was able to access the file and print the data correctly. Aside from this the only other issue I faced was laying out the program. Spacing out the printed text and prompts for the user. As well as where to add in the additional class. Originally, I added the prompts class at the end of the program and then decided to move the classes up and leave the main class with the actual processing program afterwards. My main issue was really understanding when to use additional classes. I have a basic understanding of how they play a role and can be useful, but I still have some questions. As I progress and work with some programs I am hoping to understand it better. There are some insightful forums out there that should help me with this part. I have come to learn to work in small sections as we have learned in this course. I got a little ahead of myself by not fully incorporating all the requirements when beginning to work on the code. I ended up working a little backwards. Ultimately, I did get the program to work as I intended it to, but it took more time to go and revise the code and ensure that it was not duplicating any information. I did update the pseudocode for the final submission. I removed a text group that I had left in place in case I wanted to add any additional import functions. I removed that part since I ended up not adding any additional information. I also updated the location of the scanners for the user inputs under the main class.

Pseudocode

IMPORT scanner utility. // for user input

IMPORT security message digest utility. //for MD5

IMPORT file reader utility. // for reading files such as credentials.txt

NEW CLASS - userInputs

FOR

INITIALIZE Strings (filesPath, credentialsFile, and role files)

NEW CLASS – prompts

FOR

SET prompts for user in this class.

MAIN CLASS – zooAuthentication

(Remaining program will be part of this class)

CREATE Scanner input for userInput //for user inputs

CREATE Scanner input for userExit //to be used for logging out

INITIALIZE Integer for login attempts to 3.

INITIALIZE Boolean for login //default to false

WHILE true,

PROMPT user to enter username. (allow user to end login process by entering quit as username)

PRINT enter username.

OBTAIN username.

SET username to be converted to lower case characters.

CONFIRM that user did not enter ‘quit’ to stop login.

IF user entered quit,

END login process.

IF user did not enter quit,

PROMPT user to enter password.

PRINT enter password.

OBTAIN user password.

DO NOT set input to lower case characters.

EXECUTE MD5 hash section to confirm the user’s credentials.

CONFIRM index 0 and 1 of the credentials file match the user’s input.

IF user’s credentials are confirmed,

EXECUTE new scanner to access location of files and read the associated users role file.

PRINT text from the associated user’s role text file.

PROMPT ask the user if they would like to stay logged in.

ALLOW for user input for yes or no to stay logged in.

IF user enters yes, allow user to stay logged in.

IF user enters no, insert a prompt for the user that they are logged out.

INSERT break.

IF user’s credentials are not confirmed,

WHILE user credentials are incorrect and login attempts are greater than 0,

NOTIFY user that login credentials are incorrect.

PROMPT user to attempt to log in again.

SUBTRACT 1 from the login attempts integer total.

IF user reaches 0 login attempts left,

NOTIFY user that they have exhausted the 3 login attempts and must exit the program and begin again.

SAVE the program.

EXECUTE the program.

IF any errors are generated then

DEBUG the program.

IF no errors, save the program and exit.