IT145 – Final Project: Option 1 Zoo Authentication Program

Southern New Hampshire University

by:

Brandon Rickman

**Zoos Option 1: Authentication Program Process Documentation**

**Problem Statement**

Develop an authentication system that manages both authentication and authorization. The program requires you to check the username against a credentials file. Ensure that only correct passwords are used and lockout the unauthorized user if more than three attempts are made with wrong credentials. There are three levels of authorization zookeeper, veterinarian, and admin. The program will need to convert the user entered password into a MD5 Hash. This will be accomplished in a class separate from the main program. The program will parse the provided files for matching information entered by the user. If they match the program will open a file which symbolizes granting access into a computer system. If they DO NOT match the program will request another input of the correct credentials up to three times before exiting the program.

**Overall Process**

I first looked at this as a user input entry and file comparison problem. I wrote out all the code in main, only using the MD5 Hash as a separate class. I started by thinking of a while loop to continue prompting the user for input. Then I put a counter in place to only allow so many loops through the input request. I then worked on calling the MD5 Hash class and converting the user password to into a hash. Once this was working I started looking into how to read the credentials file and what it would take to break the file apart, line by line or string by string. I first thought string by string but as I progressed I realized it would be simpler to do line by line and string by string, while checking to see if the file had further elements in it. As I was able to get this to start working correctly, bringing in the information and printing it to the screen. I then set my sights on comparing the user input information, and password hash to each other. I first set these up as a dual comparison ( if one and the other were correct, etc ). I then implemented another class to take all the comparison out of main. This reduced my original code from over 150 lines to about 75 lines.

**Pseudocode**

IMPORT Scanner

IMPORT File Stream

CREATE counter for each attempt at access to system

CREATE input stream scanner

WHILE prompting user

PROMPT user to INPUT username

PROMPT user to INPUT password

COMPARE username and password against file credentials via MD5 Password Hash

IF INCORRECT username OR password

OUTPUT error message and attempt number out of total attempts

IF MAX (third) counter attempt

EXIT LOCK user input

PROMPT user for password reset

ELSE

PROMPT user input again

REDUCE counter by 1

IF CORRECT username AND password

OUTPUT that user entered correct credentials, and please standby while file is opened

OPEN ROLE file

OUTPUT authorized ROLE file

PROMPT user to LOGOUT

CLOSE file stream

OUTPUT Login PROMPT username and PASSWORD

**Pseudocode/Methods/Classes Changes**

My Pseudocode (pcode) changed in format and had some edits due to additional classes added, but the overall idea of my pcode remained the same. I added a loop around the entire main area of the code to check if input hand been complete before ending the program. I have seen this done in multiple programs and see it as a way to specifically be able to tell the program when to stop running. I added a pre-login prompt that is used to login or logout out of the system completely. This also adds a secondary level of security as it will not allow any entry other than “login” to access the login prompt. I moved the COMPARE password to another class method in HashCompare(). I also moved all the Credential file reading and parsing into the same class under a different method. This allowed for shorter code and placed all of the file read in one location. I then moved the read/parse/open correct Role file into the same class under another method. This allowed again for one location for all file streaming. I removed the CLOSE file stream as it was no longer necessary in main. I still prompted user for logout and if yes then sent them back to main screen where they could logout completely or re-login as another user.

**Error Documentation**

When first writing the code, I placed everything into main. This caused confusion in what part was acting up when an error would evolve. I ran into an issue with the MD5 Digest class in which it was throwing a NoSuchAlgorithmException. After a little bit of research and asking some other developers, I found this is common and that utilizing a try/catch around the hash password solved the solution. This was big because it was new to me and helps you to be able to write specific error messages, which allows you to know exactly where the problem is.

I had minor adjustments to the while loops, and to the attempt counter. At first, I was getting an infinite loop. I quickly fixed this by making sure to enter a way for the loop to finish or close. I then set out to work on the attempt counter. I had to change my attempt counter to 1 instead of 0. I didn’t understand why it would run 2 times only or then I would try and fix it and it would run over and over. I determined that by making the counter start at 1 and then also by breaking out of the loop when the max attempts were reached, I was able to stop this issue. I had a for loop at first but had to remove it due to the way I wanted the prompt to continue asking for input if the wrong information was entered, and the messages I wanted to print to the user.

Although I understood how to create a class and methods, I did not understand how to take my code (all written in Main) and break it apart to work with the new class I had created. I sat down with my brother-in-law who is a software engineer at Capitol One. He was able to show my quickly how each part of my code could be pulled out. I then took each segment out and created the specific method needed in the new class. Then came the fun part of making calls to that class in Main. However, before I could correct main and make the right calls in the right places I needed to make sure my new class was working properly with each method. This is the part I am most proud about: I wrote and test my own new class Test Bench calling my new class HashCompare(). After a few tweaks to ensure each entry was correct, I found that my new class was working properly. I felt so accomplished because I had successfully created, input the information, and tested my new class on my own Test Bench.

We had not learned arrays, but I needed a way to store multiple items into one variable and then call getMethods() from the new class to retrieve those items. From my past coding experience, a list or array would be the best option. I read through some of the material in lesson 7 of ZYBooks, which helped me understand the syntax for Java of an array, but could I make a method of an array? I search and found that yes indeed this would work. I created a method that would take in line by line of the Credentials file and then determine if the user name that was passed into the method matched any of those lines. If it did match then the full line would be read element by element, taking the first element in as the creds user name, the second element as the hashed password, and then finally I would scan the line using hasNext() until it was done, applying that element to the third part of the array. This ensured that the last item left in the variable was the Role that the user could have access. I could then use this later to call for the right Role file to print to the user.

After some tweaking of the Main to remove old lines and replace with method calls to the new class I was able to get my entire program working, completely like I wanted. Until it came to testing Main. Now I wanted to build a Test Bench for this but could not figure out how to do so with a new Main, so I gave up and tested the old way, MANUEL ENTRY. The good thing about this was I only had to be right on the entry when I wanted it to be write, otherwise I could misspell any word I wanted or type the wrong letters (which is harder than you think when you’re trying to type the right words). I quickly found out that my program, although working exactly as I had intended, was causing an issue. First, it would take in the information three times and kick you out. PERFECT. Second, it would take in the correct user name and password, printing out the correct Role file each time, and allow you continue down the Credentials file through each name. Then came the problem. What if Rosario Dawson logged in, but then Griffin Keyes needed to log in right after her without killing the program and starting it again, would it work? What if you typed in the wrong information two out of the three times but on the third time you got it right, would the system let you in? The answer to both questions was no, I soon found out.

To fix the previous problem I started adding lines in the Main to print what each element was returning. This way I could determine if it was call issue or a username issue. I determined that the array was staying the same from the previous entry or empty from how I set it up. This meant that the array was not clearing each time I started the log in process over (again without stopping the program). I tried clearing the array in Main but did not seem to work right or I was not placing it in the correct spot. I finally started looking at the method that created the array. I originally had set up the method to open the file stream and scanner in the class constructor. I did this if I needed to call it in other method. I quickly realized I did not need these two in any other method and it made no sense to place them there. I moved them to the method itself. This meant that each time the method was called it would open both the file stream, the scanner, and create the array. This stopped the issue, and I was able to log in with any user name in any order; I could enter the user name incorrect two times and correctly the third time. This solved the issue. This led to the final discovery.

While looking at the array method I realized I did not need to return the array itself because I had set up methods to get each part of the array separately. By changing the method to a void, I eliminated parts of the code that were redundant or unnecessary. Although this was small it was a big break in that I had start effectively understanding my own code, and Java itself.