

Mark Eilers  
CS 499: Computer Science Capstone  
4-2 Milestone Two: Enhancement Two: Algorithms and Data Structure  
July 26, 2024

#### 4-2 Milestone Two: Enhancement Two: Algorithms and Data Structure

For the second section of the ePortfolio, Algorithms and Data Structure, I have chosen to modify my zoo authentication application from my IT 145 Foundation in Application Development Course. The original artifact was created in September of 2022. In this updated version of the original assignment, I created a Java-based authentication system that was comprised of three main classes. There was the 'UserAuth' class, 'AuthHandler' class, and 'EncryptedCredentials' class. For this project, I basically started from the ground up to enhance the security measures of the assignment. The updated artifact demonstrates a basic implementation of user authentication by validating usernames and passwords, encrypting passwords using MD5 hashing, and providing role-based greetings to users upon successful login. This application was created using IntelliJ integrated development environment (IDE) specifically using Java as the coding language.

I selected this artifact for my ePortfolio because it exemplifies my skills in algorithms and data structures, particularly in the context of user authentication and password encryption. This project showcases several key components. For the algorithm portion of the project, the use of the MD5 hashing algorithm to securely encrypt user passwords demonstrates my understanding of cryptographic functions and their implementation in Java. For data structures, the use of HashMaps to store user credentials highlights my ability to effectively use data structures for efficient data retrieval and management. For the remaining portion of this assignment, I will be referring to the screenshots at the end of this document. Each screenshot is

labeled as a figure and will be referred to as such. Additionally, each screenshot provides a short description to identify what the image is showing.

For this assignment, I essentially started from the ground up. The original project lacked major components, and I wanted to create a more secure system. As a result, I created an entirely new authentication system. To start this enhancement, I began with an authorization user class. In **Figure 1** I imported the necessary classes for reading input and handling collections. I also initialized the public variable for predefined user access. Building on this, **Figure 2** shows the creation of the user credentials. This includes creating the username and password. Finally, in **Figure 3** a while loop statement was created to check user credentials. In this section of code, users are given a prompt to enter in their username and password for the system. If the credentials are valid, then users will be granted access to the program. If the credentials are wrong, the program increments the attempted count while informing users that their username and password do not match. Users are then prompted again. If wrong again, attempted count is incremented continuing the loop. Once the attempted count is no longer less than the max attempts, then users are presented with a maximum attempt statement asking them to try again later. The combination of **Figures 1 – Figure 3** showcase the creation and implementation of the user authentication class.

Once the user authentication class was completed, I then worked on the next class, AuthHandler. In this class credentials were checked. Once the credentials were correctly matched, then a welcoming statement as well as a defined role was presented to the user. If the credentials were not a match, then the user was informed of their failed login attempt. In **Figure 4** a method was created to validate the user's credentials. The use of HashMaps was implemented in this section to showcase my use of data structures. In **Figure 5 – Figure 6** the

user credentials were validated using switch and case statements. In each case, the username and password are validated for a match. Once a match is found the program displays a statement welcoming the user and informing them of their role within the computer system. If the credentials were not a match, then the user was also informed of their failed attempt.

In the final set of three screenshots, three additional classes were created to round out the final project. In **Figure 7** the credential class was created. This contained the setters and getters for the username, password, and encryption portions of the program. Finally, in **Figures 8 – Figures 9** code was written to transform the user's password into an encrypted password. This was done using MD5 Hashing to encrypt the passwords, adding an additional layer of security to the overall project. Overall, this project is a major improvement to the original artifact. Code was sectioned into different classes that would make it easier to maintain and scale for future use. Additionally, each class has appropriate comments and proper naming standards. This makes the code easier to manage, work with, scale, update, and overall use in a professional setting.

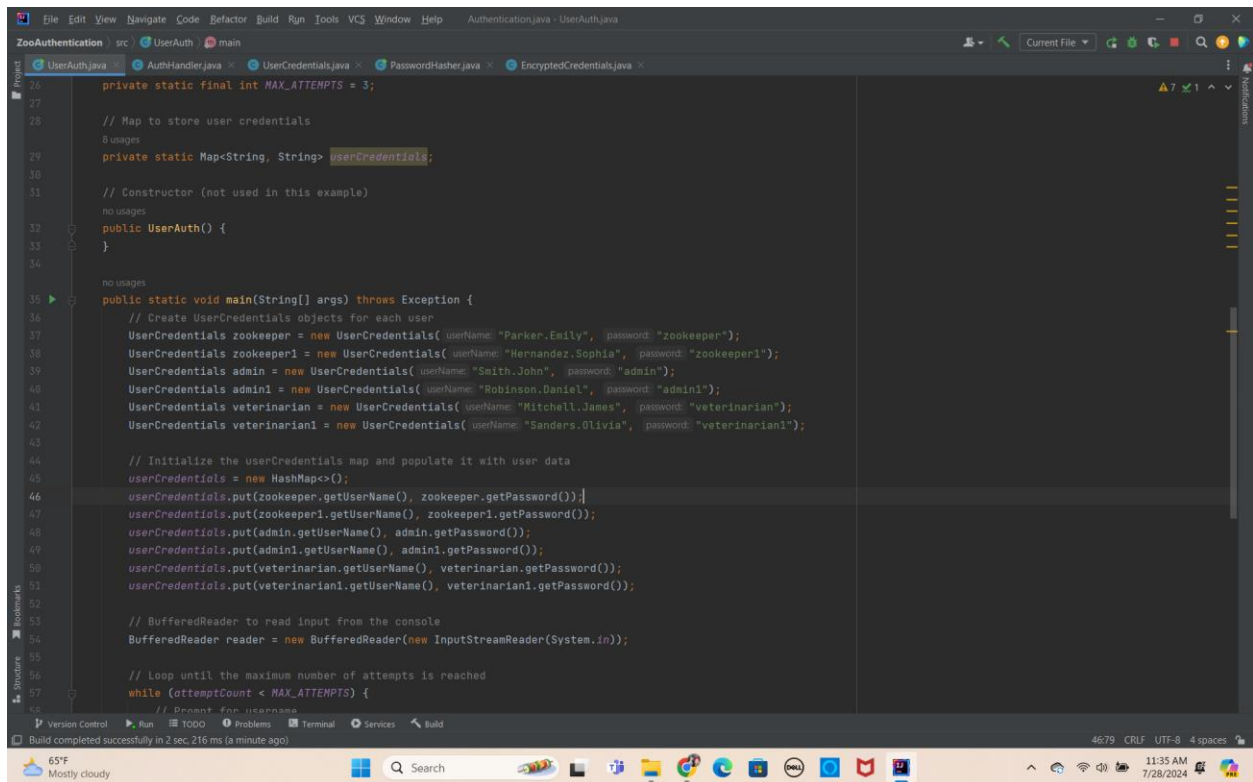
With the enhancements made to this artifact, I have successfully met the course outcomes I planned to achieve in Module One. These include demonstrating the ability to implement and improve algorithms and data structures in a real-world application. I currently have no updates to my outcome-coverage plans as the enhancement aligns with my initial goals. Enhancing and modifying this artifact was a valuable learning experience. During this process, I learned several important lessons. These include the importance of code readability, user experience, and exception handling.

While I would consider this project to be a success. I did run into some issues. Namely, one of the biggest issues to start was finding the IDE that worked for me. I initially started this work using eclipse. However, I quickly found that to be tedious and problematic for me. While I

can appreciate the features that eclipse offers, it is not a program that works best for me. As a result, I switched over to IntelliJ IDEA. Once I made the switch, I found that creating the project was easier to manage. This was due to the features that IntelliJ offers, and admittedly the overall look and feel of the IDE. Another issue I ran into was figuring out exactly how I wanted to plan this project. This artifact requires a lot of different classes. As a result, a lot of working parts are running to make the program function. I needed to find a way to plan out this project to effectively manage my time to meet the deadline. I found that pseudocode was beneficial. It allowed me to create an overall plan that I could work off. Pseudocode gave me the chance to get everything mapped out, so that I was not updating and changing major portions of already completed code. As a result, it allowed me to effectively manage my time. Overall, this process has enhanced my skills in writing clean, maintainable code and reinforced the importance of considering both functionality and user experience in software development.

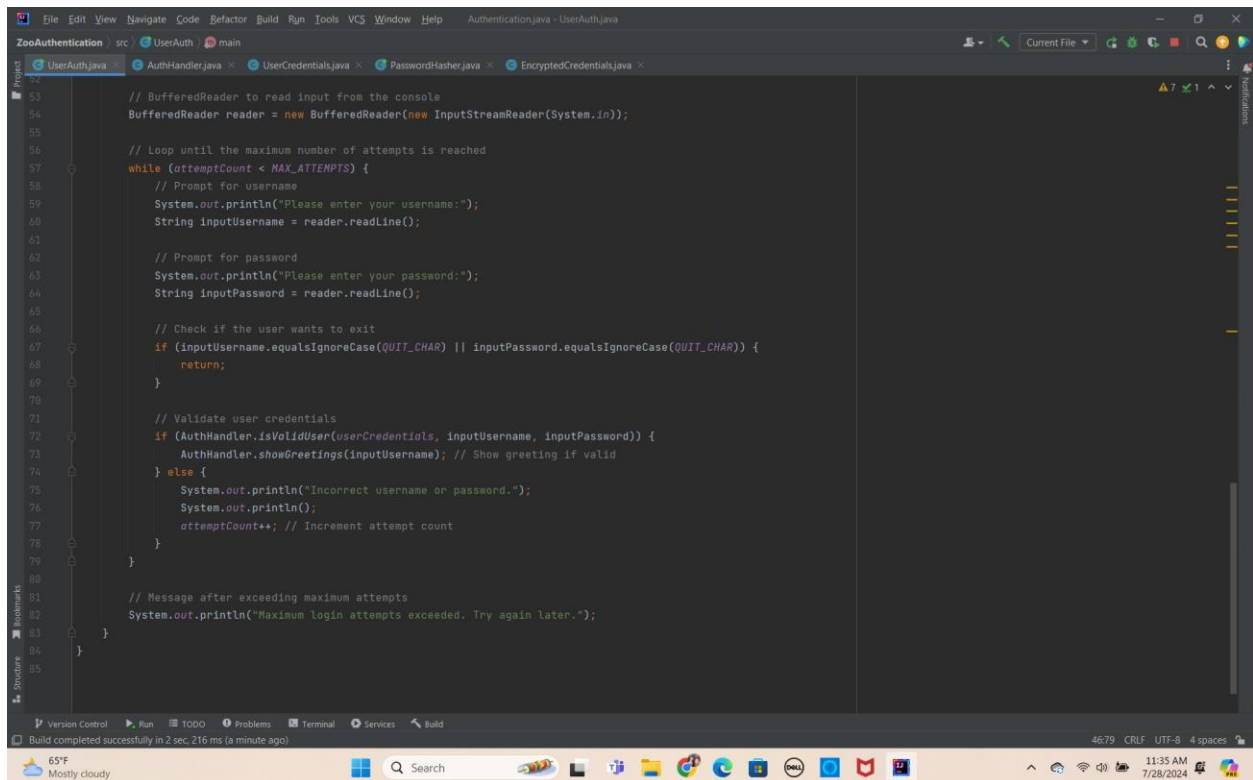
```
1 // Mark Eilers
2 // 4-2 Milestone Three: Enhancement Two: Algorithms and Data Structure
3 // CS 499: Senior Capstone
4 // July 23, 2024
5
6 // Import necessary classes for reading input and handling collections
7 import java.io.BufferedReader;
8 import java.io.InputStreamReader;
9 import java.util.HashMap;
10 import java.util.Map;
11
12 no usages
13 public class UserAuth {
14     // Constants for predefined usernames
15     no usages
16     public static final String ZOOKEEPER_USER = "Parker,Emily";
17     no usages
18     public static final String ADMIN_USER = "Smith,John";
19     no usages
20     public static final String VET_USER = "Mitchell,James";
21     no usages
22     public static final String ZOOKEEPER_USER_1 = "Hernandez,Sophia";
23     no usages
24     public static final String ADMIN_USER_1 = "Robinson,Daniel";
25     no usages
26     public static final String VET_USER_1 = "Sanders,Olivia";
27
28     // Constant for exit character
29     2 usages
30     private static final String QUIT_CHAR = "q";
31
32     // Variables for tracking login attempts
33     2 usages
34     private static int attemptCount = 0;
35     1 usage
36     private static final int MAX_ATTEMPTS = 3;
37
38 Version Control Run TODO Problems Terminal Services Build
39 Build completed successfully in 2 sec 216 ms (a minute ago)
40 65°F Mostly cloudy 46:79 CRUF UTF-8 4 spaces 11:35 AM 7/28/2024
```

**Figure 1:** This screenshot shows the beginning stages of the main function of the code. In this image, the necessary imports are shown as well as the creation of the public variables for the different users in the computer system. This screenshot begins the user authentication process.



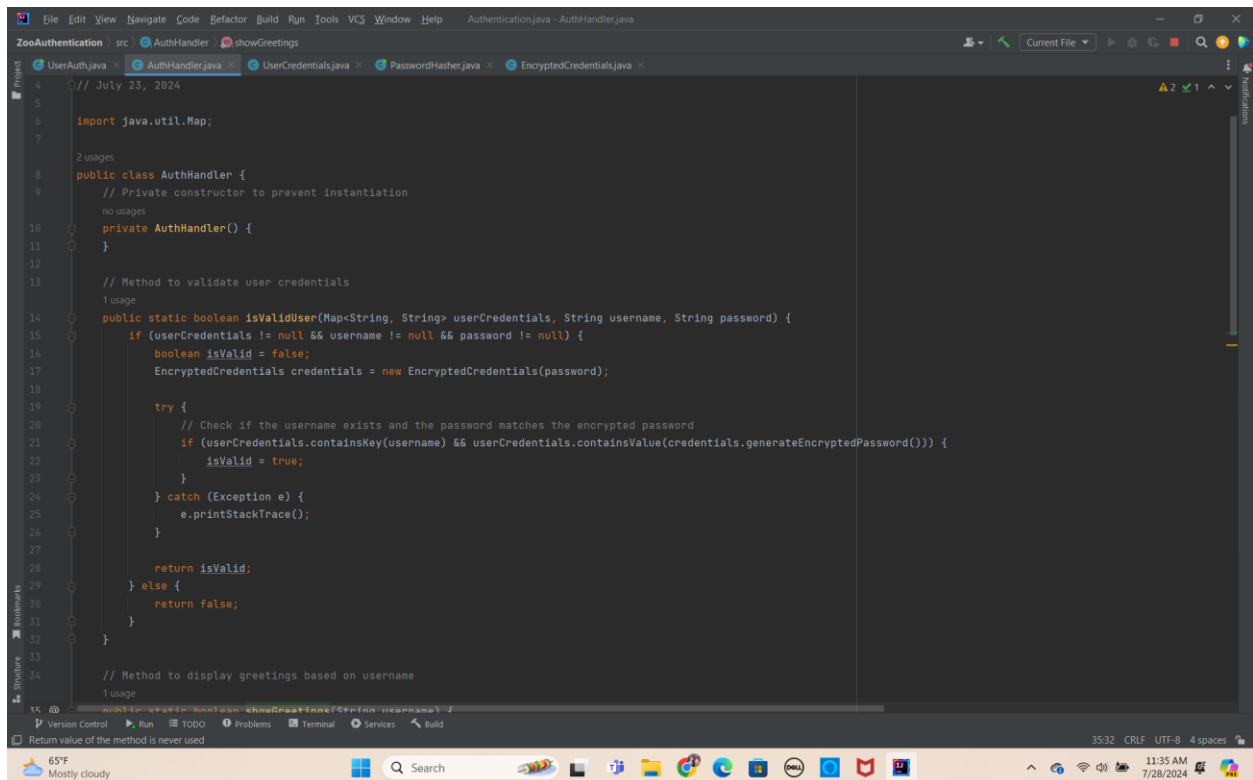
```
26 private static final int MAX_ATTEMPTS = 3;
27
28 // Map to store user credentials
29 // Map to store user credentials
30 private static Map<String, String> userCredentials;
31
32 // Constructor (not used in this example)
33 public UserAuth() {
34 }
35
36 // Create UserCredentials objects for each user
37 UserCredentials zookeeper = new UserCredentials( userName: "Parker.Emily", password: "zookeeper");
38 UserCredentials zookeeper1 = new UserCredentials( userName: "Hernandez.Sophia", password: "zookeeper1");
39 UserCredentials admin = new UserCredentials( userName: "Smith.John", password: "admin");
40 UserCredentials admin1 = new UserCredentials( userName: "Robinson.Daniel", password: "admin1");
41 UserCredentials veterinarian = new UserCredentials( userName: "Mitchell.James", password: "veterinarian");
42 UserCredentials veterinarian1 = new UserCredentials( userName: "Sanders.Olivia", password: "veterinarian1");
43
44 // Initialize the userCredentials map and populate it with user data
45 userCredentials = new HashMap<>();
46 userCredentials.put(zookeeper.getUserName(), zookeeper.getPassword());
47 userCredentials.put(zookeeper1.getUserName(), zookeeper1.getPassword());
48 userCredentials.put(admin.getUserName(), admin.getPassword());
49 userCredentials.put(admin1.getUserName(), admin1.getPassword());
50 userCredentials.put(veterinarian.getUserName(), veterinarian.getPassword());
51 userCredentials.put(veterinarian1.getUserName(), veterinarian1.getPassword());
52
53 // BufferedReader to read input from the console
54 BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
55
56 // Loop until the maximum number of attempts is reached
57 while (attemptCount < MAX_ATTEMPTS) {
58     // Prompt for username
59 }
```

**Figure 2:** This is continuation of the user authentication class creating user credentials.

The image is a screenshot of an IDE window titled 'ZooAuthentication'. The main editor shows a Java file named 'UserAuth.java'. The code implements a login loop. It starts by creating a 'BufferedReader' to read from 'System.in'. A 'while' loop with the condition 'attemptCount < MAX\_ATTEMPTS' handles the login process. Inside the loop, it prompts the user for a username and password using 'System.out.println' and 'reader.readLine()'. It then checks if the user wants to exit by comparing the input to 'QUIT\_CHAR'. If not, it calls 'AuthHandler.isValidUser' to validate the credentials. If valid, it shows a greeting. If invalid, it prints an error message and increments 'attemptCount'. After the loop, it prints a message about exceeding maximum attempts. The IDE interface includes a menu bar, a toolbar, a project explorer on the left, and a status bar at the bottom showing build information and system status.

```
53 // BufferedReader to read input from the console
54 BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
55
56 // Loop until the maximum number of attempts is reached
57 while (attemptCount < MAX_ATTEMPTS) {
58     // Prompt for username
59     System.out.println("Please enter your username:");
60     String inputUsername = reader.readLine();
61
62     // Prompt for password
63     System.out.println("Please enter your password:");
64     String inputPassword = reader.readLine();
65
66     // Check if the user wants to exit
67     if (inputUsername.equalsIgnoreCase(QUIT_CHAR) || inputPassword.equalsIgnoreCase(QUIT_CHAR)) {
68         return;
69     }
70
71     // Validate user credentials
72     if (AuthHandler.isValidUser(userCredentials, inputUsername, inputPassword)) {
73         AuthHandler.showGreetings(inputUsername); // Show greeting if valid
74     } else {
75         System.out.println("Incorrect username or password.");
76         System.out.println();
77         attemptCount++; // Increment attempt count
78     }
79 }
80
81 // Message after exceeding maximum attempts
82 System.out.println("Maximum login attempts exceeded. Try again later.");
83 }
84
85 }
```

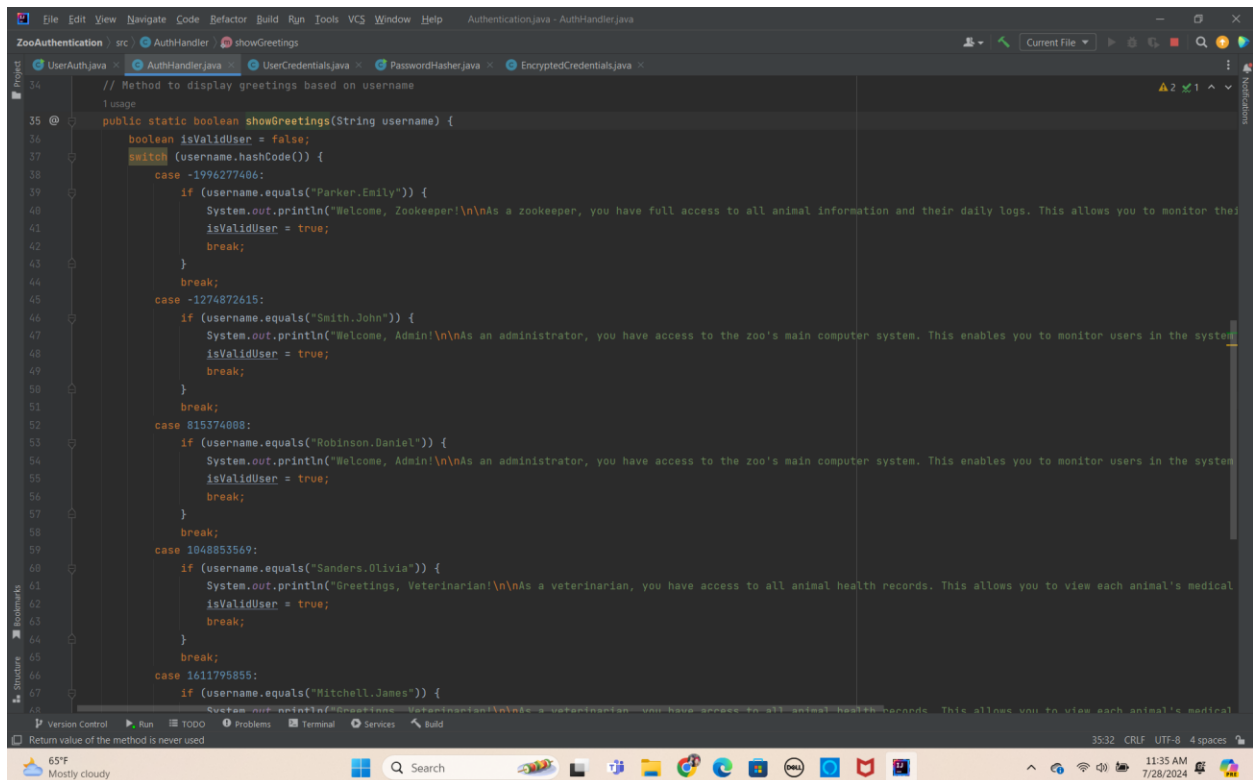
**Figure 3:** Screenshot shows a while loop statement to check for user credentials. In this section of code, users are given a prompt to enter in their username and password for the system. If the credentials are valid, then users will be granted access to the program. If the credentials are wrong, the program increments the attempted count while informing users that their username and password do not match. Users are then prompted again. If wrong again, attempted count is incremented continuing the loop. Once the attempted count is no longer less than the max attempts, then users are presented with a maximum attempt statement asking them to try again later.



```
4 // July 23, 2024
5
6 import java.util.Map;
7
8 2 usages
9 public class AuthHandler {
10     // Private constructor to prevent instantiation
11     no usages
12     private AuthHandler() {
13     }
14
15     // Method to validate user credentials
16     1 usage
17     public static boolean isValidUser(Map<String, String> userCredentials, String username, String password) {
18         if (userCredentials != null && username != null && password != null) {
19             boolean isValid = false;
20             EncryptedCredentials credentials = new EncryptedCredentials(password);
21
22             try {
23                 // Check if the username exists and the password matches the encrypted password
24                 if (userCredentials.containsKey(username) && userCredentials.containsValue(credentials.generateEncryptedPassword())) {
25                     isValid = true;
26                 }
27             } catch (Exception e) {
28                 e.printStackTrace();
29             }
30
31             return isValid;
32         } else {
33             return false;
34         }
35     }
36
37     // Method to display greetings based on username
38     1 usage
39     public static void showGreetings(String username) {
40     }
41 }
```

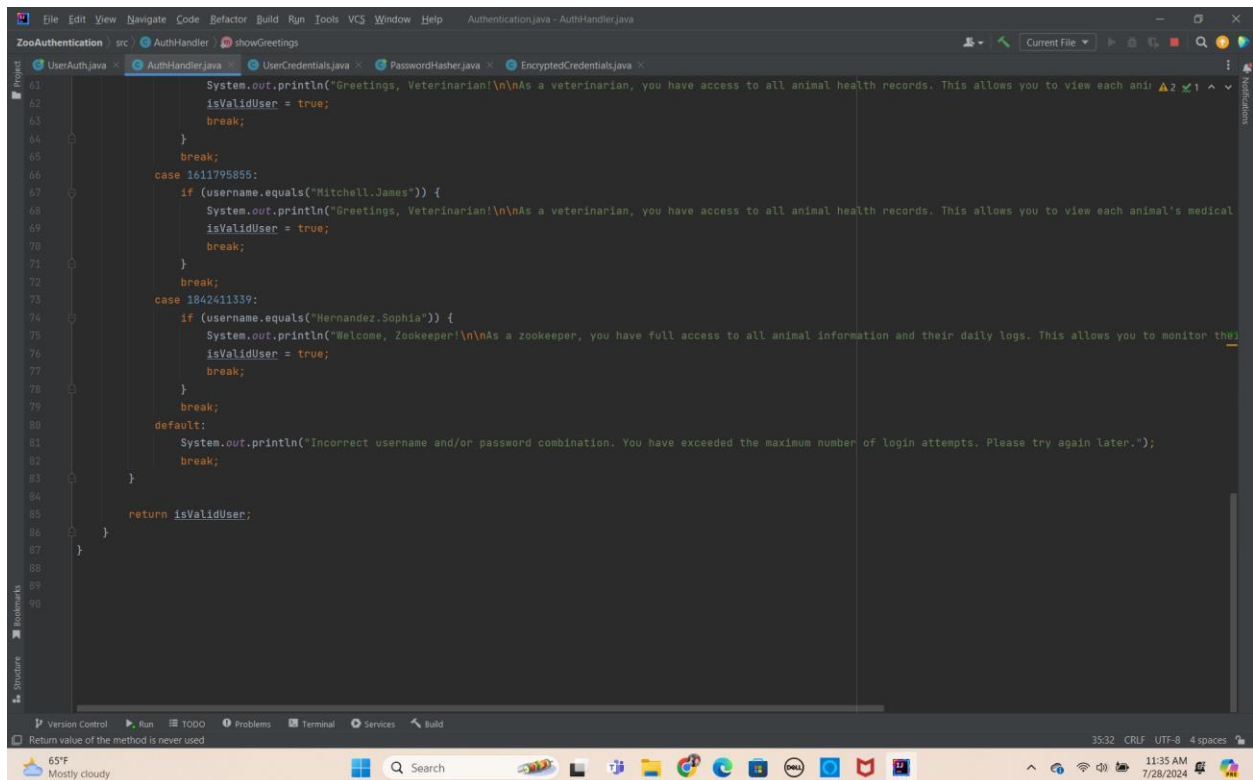
**Figure 4:** This is the creation of the second class, AuthHandler. In this screenshot, a method is created to validate user credentials.





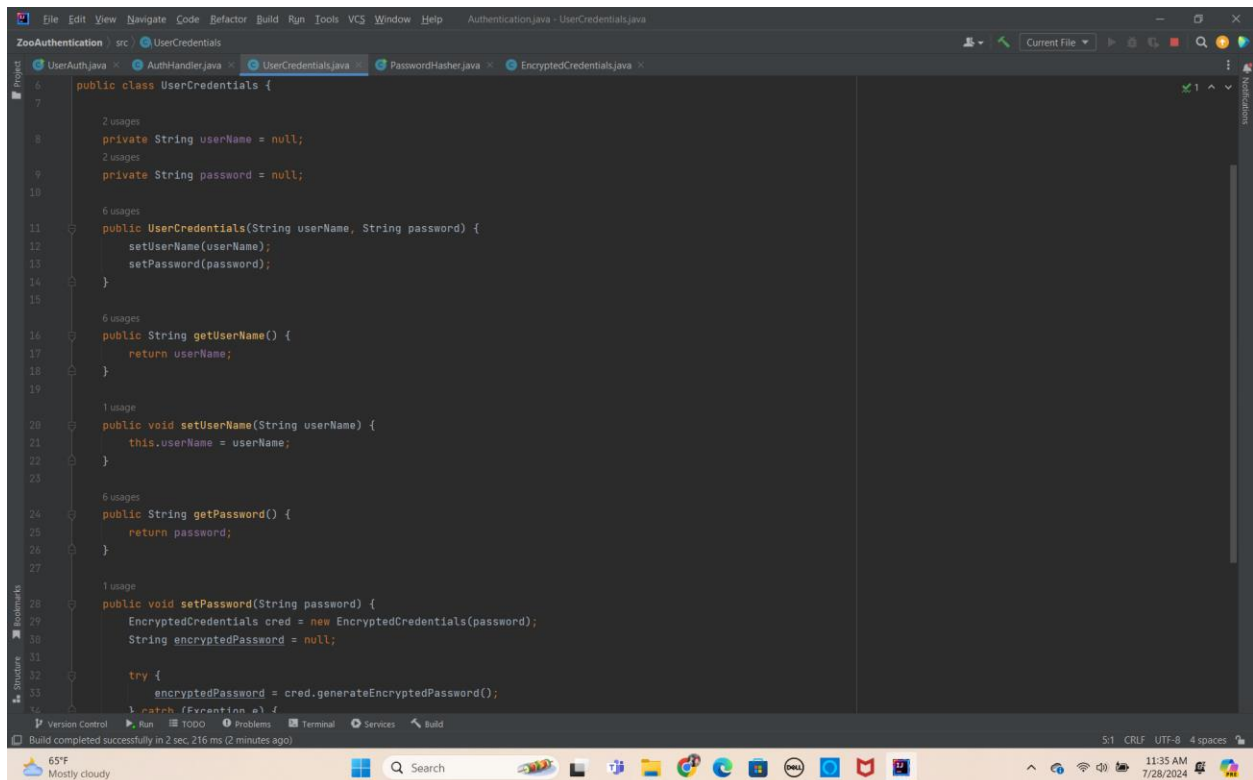
```
34 // Method to display greetings based on username
35 @ 1 usage
36 public static boolean showGreetings(String username) {
37     boolean isValidUser = false;
38     switch (username.hashCode()) {
39         case -1996277486:
40             if (username.equals("Parker.Emily")) {
41                 System.out.println("Welcome, Zookeeper!\n\nAs a zookeeper, you have full access to all animal information and their daily logs. This allows you to monitor thei
42                 isValidUser = true;
43                 break;
44             }
45             break;
46         case -1274872615:
47             if (username.equals("Smith.John")) {
48                 System.out.println("Welcome, Admin!\n\nAs an administrator, you have access to the zoo's main computer system. This enables you to monitor users in the system
49                 isValidUser = true;
50                 break;
51             }
52             break;
53         case 815374088:
54             if (username.equals("Robinson.Daniel")) {
55                 System.out.println("Welcome, Admin!\n\nAs an administrator, you have access to the zoo's main computer system. This enables you to monitor users in the system
56                 isValidUser = true;
57                 break;
58             }
59             break;
60         case 1048853569:
61             if (username.equals("Sanders.Olivia")) {
62                 System.out.println("Greetings, Veterinarian!\n\nAs a veterinarian, you have access to all animal health records. This allows you to view each animal's medical
63                 isValidUser = true;
64                 break;
65             }
66             break;
67         case 1611798855:
68             if (username.equals("Mitchell.James")) {
69                 System.out.println("Greetings, Veterinarian!\n\nAs a veterinarian, you have access to all animal health records. This allows you to view each animal's medical
70                 isValidUser = true;
71                 break;
72             }
73             break;
74     }
75     return isValidUser;
76 }
```

**Figure 5:** Once the credentials have been determined to be accurate, a welcome statement including the role of the individual is displaced. In this section, switch cases were used to check for the username. If the username was equal to a predetermined username and password from the user authentication class, then the program welcomed the user, as well as informing them of their role within the company.



```
61         System.out.println("Greetings, Veterinarian!\n\nAs a veterinarian, you have access to all animal health records. This allows you to view each ani\n62         isValidUser = true;\n63         break;\n64     }\n65     break;\n66     case 1611795855:\n67         if (username.equals("Mitchell.James")) {\n68             System.out.println("Greetings, Veterinarian!\n\nAs a veterinarian, you have access to all animal health records. This allows you to view each animal's medical\n69             isValidUser = true;\n70             break;\n71         }\n72         break;\n73     case 1842411339:\n74         if (username.equals("Hernandez.Sophia")) {\n75             System.out.println("Welcome, Zookeeper!\n\nAs a zookeeper, you have full access to all animal information and their daily logs. This allows you to monitor the\n76             isValidUser = true;\n77             break;\n78         }\n79         break;\n80     default:\n81         System.out.println("Incorrect username and/or password combination. You have exceeded the maximum number of login attempts. Please try again later.");\n82         break;\n83     }\n84\n85     return isValidUser;\n86 }\n87\n88\n89\n90
```

**Figure 6:** Screenshot is a continuation of Figure 5 with an added statement if none of the credentials are valid. If none of the credentials are valid, then a statement informing the incorection username and/or password is displayed.

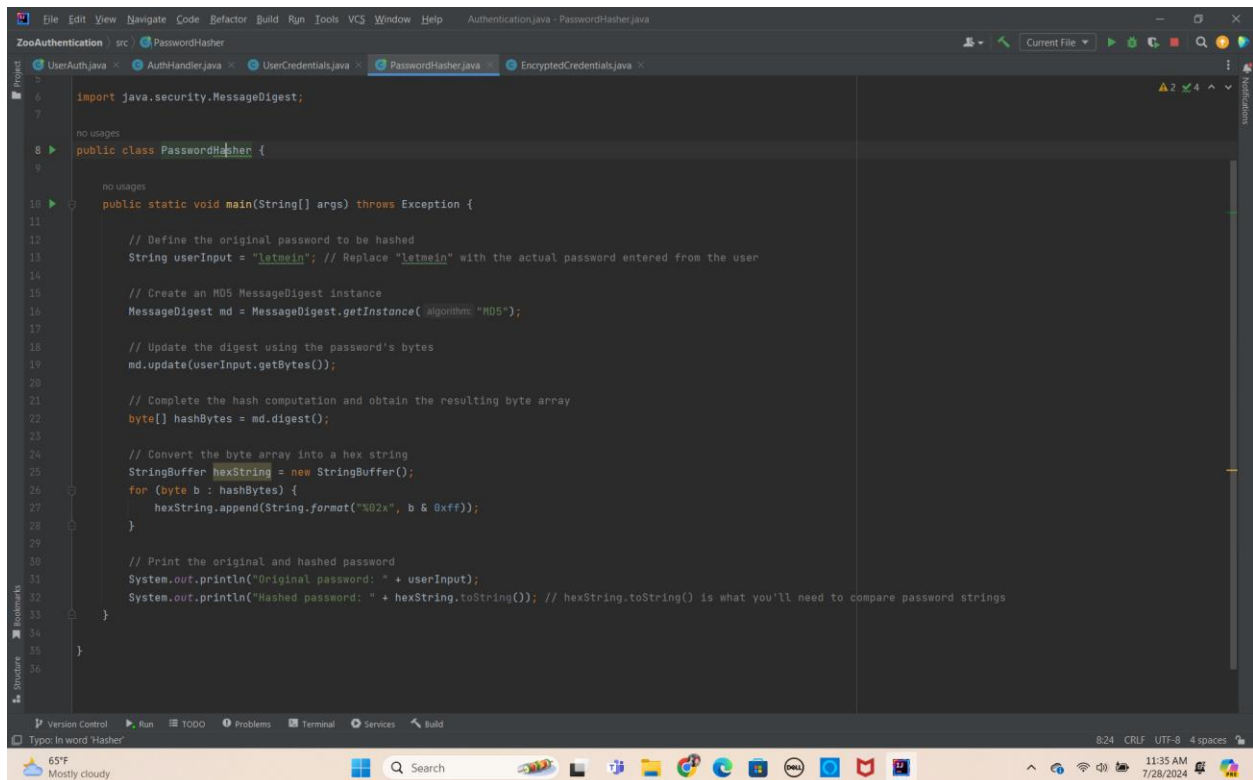


```
6 public class UserCredentials {
7
8     2 usages
9     private String userName = null;
10
11     2 usages
12     private String password = null;
13
14     6 usages
15     public UserCredentials(String userName, String password) {
16         setUsername(userName);
17         setPassword(password);
18     }
19
20     6 usages
21     public String getUserName() {
22         return userName;
23     }
24
25     1 usage
26     public void setUsername(String userName) {
27         this.userName = userName;
28     }
29
30     6 usages
31     public String getPassword() {
32         return password;
33     }
34
35     1 usage
36     public void setPassword(String password) {
37         EncryptedCredentials cred = new EncryptedCredentials(password);
38         String encryptedPassword = null;
39
40         try {
41             encryptedPassword = cred.generateEncryptedPassword();
42         } catch (Exception e) {
43             // Handle exception
44         }
45     }
46 }
```

Build completed successfully in 2 sec, 216 ms (2 minutes ago)

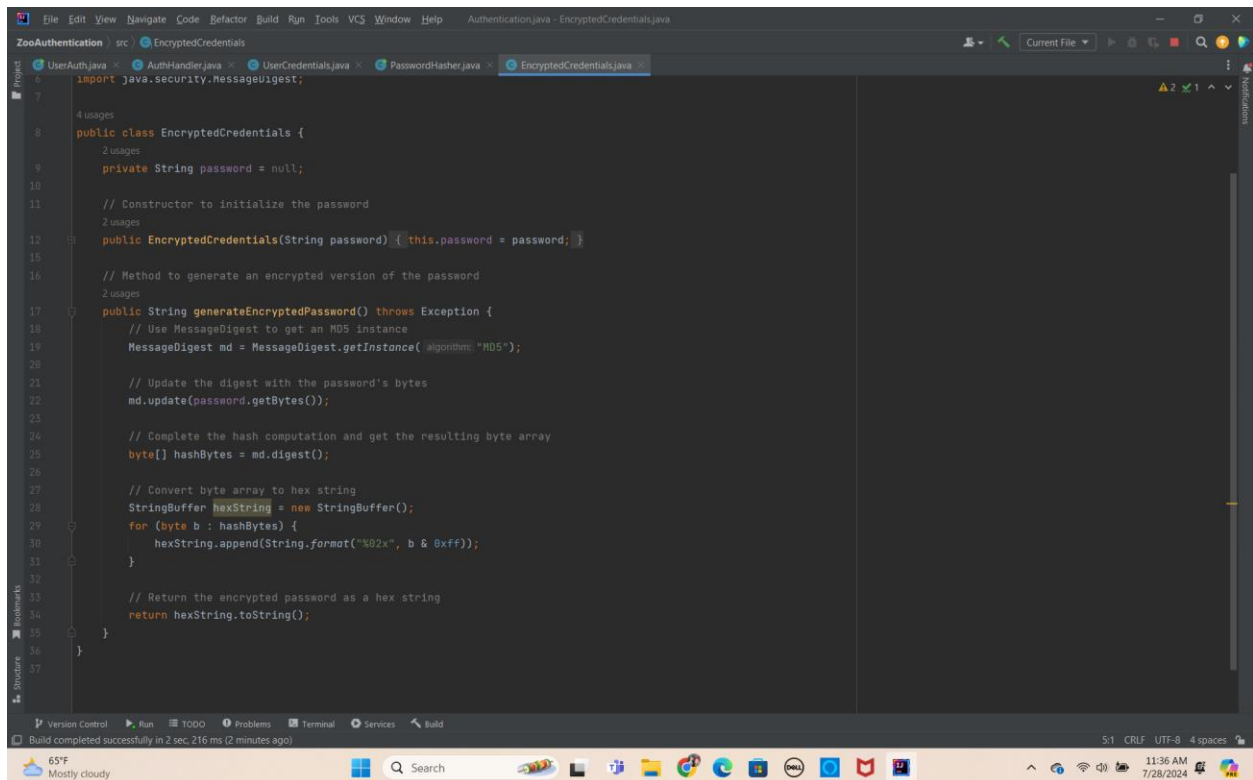
65°F Mostly cloudy 11:35 AM 7/28/2024

**Figure 7:** The screenshot shows the user credential class. In this class, the setters and getters with created to make the username and passwords for user credentials.



```
5 import java.security.MessageDigest;
6
7 no usages
8 public class PasswordHasher {
9
10 no usages
11 public static void main(String[] args) throws Exception {
12
13     // Define the original password to be hashed
14     String userInput = "letmein"; // Replace "letmein" with the actual password entered from the user
15
16     // Create an MD5 MessageDigest instance
17     MessageDigest md = MessageDigest.getInstance("MD5");
18
19     // Update the digest using the password's bytes
20     md.update(userInput.getBytes());
21
22     // Complete the hash computation and obtain the resulting byte array
23     byte[] hashBytes = md.digest();
24
25     // Convert the byte array into a hex string
26     StringBuffer hexString = new StringBuffer();
27     for (byte b : hashBytes) {
28         hexString.append(String.format("%02x", b & 0xff));
29     }
30
31     // Print the original and hashed password
32     System.out.println("Original password: " + userInput);
33     System.out.println("Hashed password: " + hexString.toString()); // hexString.toString() is what you'll need to compare password strings
34 }
35
36 }
```

**Figure 8:** This section of code shows the PasswordHasher class. In this class, to add to a layer of security, the original password a user creates it turned into a hashed password. This class is used to inform users in the end what their password was, as well as their new encrypted password.



```
6 import java.security.MessageDigest;
7
8 public class EncryptedCredentials {
9     private String password = null;
10
11     // Constructor to initialize the password
12     public EncryptedCredentials(String password) { this.password = password; }
13
14     // Method to generate an encrypted version of the password
15     public String generateEncryptedPassword() throws Exception {
16         // Use MessageDigest to get an MD5 instance
17         MessageDigest md = MessageDigest.getInstance("MD5");
18
19         // Update the digest with the password's bytes
20         md.update(password.getBytes());
21
22         // Complete the hash computation and get the resulting byte array
23         byte[] hashBytes = md.digest();
24
25         // Convert byte array to hex string
26         StringBuffer hexString = new StringBuffer();
27         for (byte b : hashBytes) {
28             hexString.append(String.format("%02x", b & 0xff));
29         }
30
31         // Return the encrypted password as a hex string
32         return hexString.toString();
33     }
34 }
```

**Figure 9:** The final screenshot shows the encrypted credentials code to encrypt user’s passwords. This process uses MD5 Hashing to encrypt user passwords, adding a layer of security to the overall program.