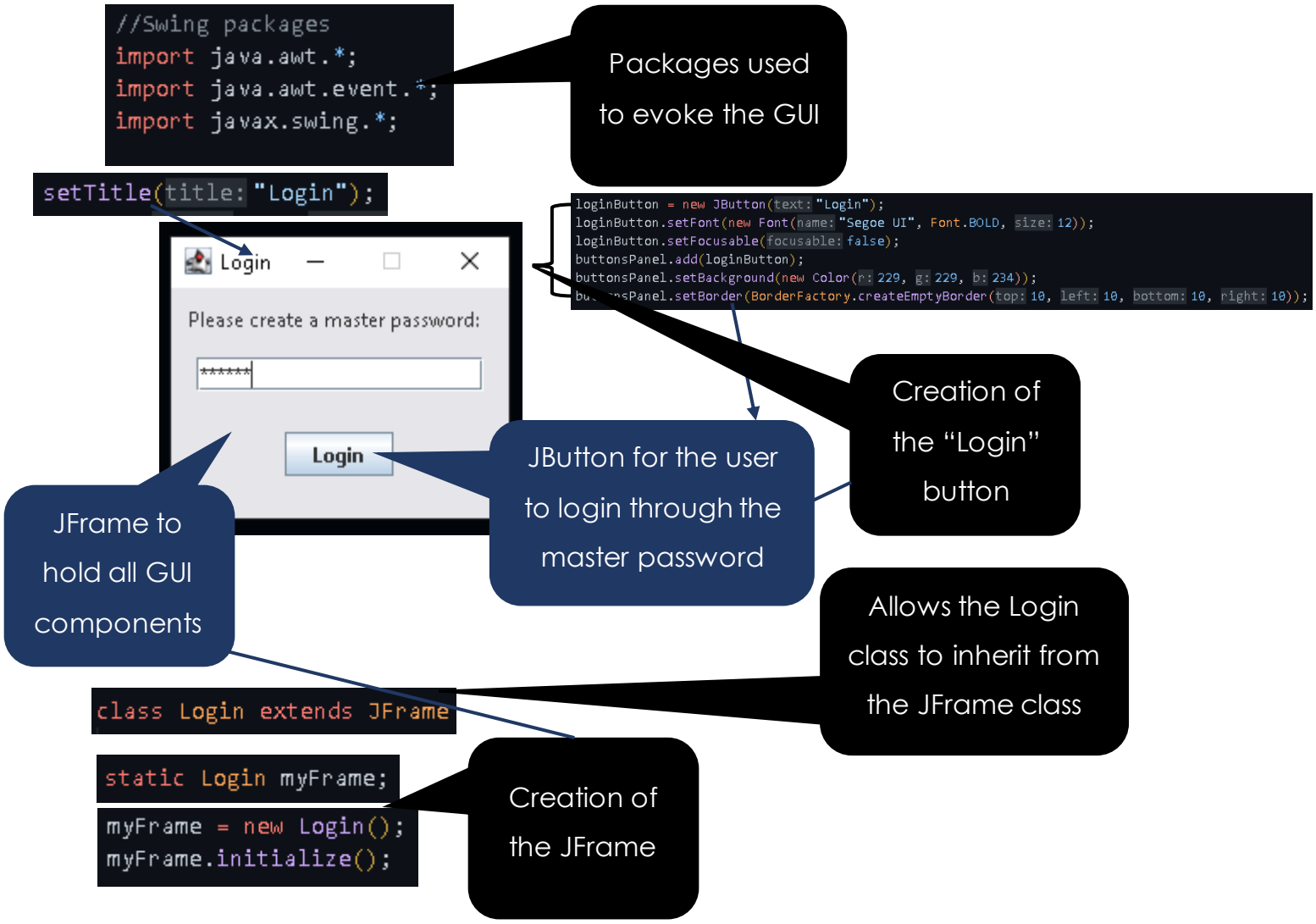
Development (716 words)

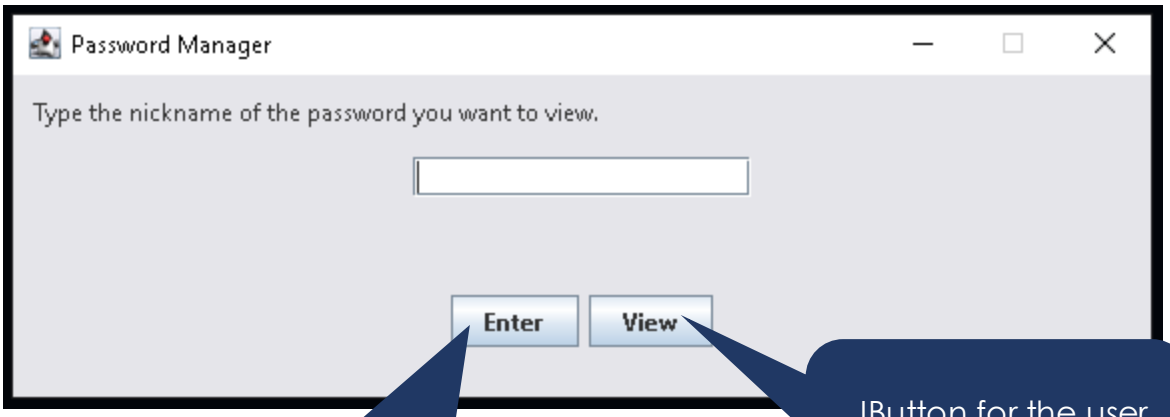
I used many techniques in solving the problem for the end-user. The password manager allows you to not have to recall all your passwords, and it is done in a secure way. The full source code for the solution is on Appendix B, but the table of contents below shows the specific techniques I used to complete the task.

Table of Contents

Java Swing Graphical User Interface:



Java Swing GUI is a powerful tool that allows the user to click buttons, type text, and do many other actions in window pop ups. Generally, it provides a great user experience (UX). I used this tool to make Franklin feel comfortable with an application that is like what he knows. Essentially, a GUI allows for an intuitive experience for the user.



JButton for the user to enter a new password.

JButton for the user to view a password.

```
JButton enter = new JButton(text: "Enter");
JButton view = new JButton(text: "View");
```

```
enter.setFont(new Font(name: "Segoe UI", Font.BOLD, size: 12));
enter.setFocusable(focusable: false);
enter.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        View newViewWindow = new View(readOnly: false, inputNickname: null);
    }
});

buttonsPanel.add(enter);
```

```
view.setFont(new Font(name: "Segoe UI", Font.BOLD, size: 12));
view.setFocusable(focusable: false);
view.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        if(nickname.getText().equals(anObject: ""))
            JOptionPane.showMessageDialog(frame, message: "Please enter a nickname to view."
                , title: "Error Message", JOptionPane.ERROR_MESSAGE);

        else {
            View newViewWindow = new View(readOnly: true, nickname.getText());
        }
    }});

buttonsPanel.add(view);
```

Hashing:

```
//Hashing package
import java.security.*;
import java.nio.charset.*;
```

Packages used to hash the master password

```
mp = hexDigest(password, digestName: "SHA-256");
```

A String variable, "mp," stores the hashed String returned from the SHA-256 hashing function.

```
ba7816bf8f01cfea414140de5dae2223b00361a396177a9cb410ff61f20015ad
```

The hashed master password in the Passwords.txt file

```
static String hexDigest(String str, String digestName) {
  try {
    MessageDigest md = MessageDigest.getInstance(digestName);
    byte[] digest = md.digest(str.getBytes(StandardCharsets.UTF_8));
    char[] hex = new char[digest.length * 2];
    for (int i = 0; i < digest.length; i++) {
      hex[2 * i] = "0123456789abcdef".charAt((digest[i] & 0xf0) >> 4);
      hex[2 * i + 1] = "0123456789abcdef".charAt(digest[i] & 0x0f);
    }

    return new String(hex);
  } catch (NoSuchAlgorithmException e) {
    throw new IllegalStateException(e);
  }
}
```

To satisfy Franklin's requirement for security, I chose to use a hashing algorithm for the master password because you cannot reverse a hash to obtain the original password. I knew I would be able to use hashing because I just needed to verify whether the user is authorized to use the password manager. I did not have to retrieve the master password. Also, hashing functions are very difficult to solve for the original String.

```
String verification = hexDigest(password, digestName: "SHA-256");

if(verification.equals(mp)) {
    myFrame.dispose();
    Manager newWindow = new Manager();
}
else {
    JOptionPane.showMessageDialog(myFrame, message: "Invalid Master Password. Please Try Again.",
        title: "Error Message", JOptionPane.INFORMATION_MESSAGE);
}
```

These lines of code are checking if the inputted master password matches with the master password in the text file.

Reading and Writing:

```
//Hashing package
import java.security.*;
```

Packages used to read and write to a text file

```
mp = ReadMasterPassword(FILEPATH);
```

The contents of the Passwords text file are read

```
static final String FILEPATH = "Passwords.txt";
```

```
static String ReadMasterPassword(String FilePath) {
    //if there is no file or the file is empty, then return the null string.
    String MasterPass = null;

    // attempt to read master password from file.
    try {
      BufferedReader in = new BufferedReader(new FileReader(FilePath));
      String s = in.readLine();
      MasterPass = s;
      in.close();
    }
    catch(java.io.FileNotFoundException e) { }
    catch(java.io.IOException t) { }

    return MasterPass;
  }
```

```
SaveMasterPassword(mp, FILEPATH);
```

The master password is written to the text file through the "SaveMasterPassword" behavior.

```
  private void SaveMasterPassword(String pwd, String FilePath) {
    try
    {
      PrintWriter out = new PrintWriter(new BufferedWriter (new FileWriter(FilePath)));

      out.println(pwd);
      out.flush();
      out.close();
    }
catch (IOException e) {}
  }
```

Reading and writing to a file allows the program to become non-volatile when storing data. This fact led me to use it when developing the product for Franklin.

```
SavePasswords(testList, Login.FILEPATH);
```

The user password information is written to the Passwords text file through the SavePasswords behavior

```
private static void SavePasswords(ArrayList<UserPassword> UPList, String FilePath) {
  try
  {
    PrintWriter out = new PrintWriter( new BufferedWriter (new FileWriter(FilePath)));

    //save the master password on the first line of the file.
    out.println(Login.mp);

    for (UserPassword p : UPList) {
      out.println(p.nickname + ";" + p.username + ";" + p.password + ";" + p.url);
    }
    out.flush();
    out.close();
  }
  catch (IOException e) { System.out.println(x: "Problem Saving Passwords"); }
}
```

Encryption and Decryption:

```
//Password encryption package (AES Encryption Algorithm)
import javax.crypto.*;
import javax.crypto.spec.SecretKeySpec;
```

Packages used to encrypt and decrypt Strings.

```
upobject.username = EncryptText(usernameField.getText(), ENC_KEY, algorithm: "AES");
upobject.password = EncryptText(passwordField.getText(), ENC_KEY, algorithm: "AES");
```

In these two lines, the username and password are encrypted through the "EncryptText" behavior to write to the Passwords text file.

```
private static String EncryptText(String plainText, String key, String algorithm) {
  String hex = null;
  try {
    Cipher c = Cipher.getInstance(algorithm);
    SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(StandardCharsets.UTF_8), algorithm);
    c.init(Cipher.ENCRYPT_MODE, secretKeySpec);

    byte[] encBytes = c.doFinal(plainText.getBytes(StandardCharsets.UTF_8));
    hex = DatatypeConverter.printHexBinary(encBytes);
  } catch (Exception ex) {
    System.out.println(x: "Error encrypting data");
    ex.printStackTrace();
  }
  return hex;
}
```

```
usernameField.setText(DecryptText(search.username, ENC_KEY, algorithm: "AES"));
passwordField.setText(DecryptText(search.password, ENC_KEY, algorithm: "AES"));
```

To satisfy Franklin's requirement for security, I used encryption and decryption for the username and passwords entered because I needed to display the username and passwords if the user want's to view them. So, I cannot use the hashing algorithm because I need to decrypt the passwords.

In these two lines, the username and password are decrypted through the "DecryptText" behavior to display the password information to the user.

```
private static String DecryptText(String cipherText, String key, String algorithm) {
  String decStr = null;
  try {
    Cipher c = Cipher.getInstance(algorithm);
    SecretKeySpec secretKeySpec = new SecretKeySpec(key.getBytes(java.nio.charset.StandardCharsets.UTF_8), algorithm);
    c.init(Cipher.DECRYPT_MODE, secretKeySpec);

    byte[] encBytes = DatatypeConverter.parseHexBinary(cipherText);
    decStr = new String(c.doFinal(encBytes), java.nio.charset.StandardCharsets.UTF_8);
  } catch (Exception ex) {
    System.out.println(x: "Error encrypting data");
    ex.printStackTrace();
  }
  return decStr;
}
```

Browser Launching:

```
// Browser Launching package
import java.net.URI;
```

Package used to launch the default web browser

```
visit.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {

    if(urlField.getText() == "") JOptionPane.showMessageDialog(frame, message: "Please provide a URL to browse.",
      title: "Error Message", JOptionPane.ERROR_MESSAGE);

    else {
      try {

        URI uri = new URI("https://" + urlField.getText());
        java.awt.Desktop.getDesktop().browse(uri);
      } catch(Exception f) {f.printStackTrace();}
    }
  }
});
```

These lines of code are executed in the event that the "Browse" button is pressed. It essentially opens the URL from user input into the system's default browser.

Since Franklin wanted a webpage to open through the application, I utilized the URI class from the java.net package to open the default web browser.

Error Management:

```
if(urlField.getText() == "") JOptionPane.showMessageDialog(frame, message: "Please provide a URL to browse.",
  title: "Error Message", JOptionPane.ERROR_MESSAGE);
```

This code sends an error message if the string in the URL text field is empty.

```
if(itemIndex == -1) {

  JOptionPane.showMessageDialog(frame, message: "The nickname you have entered is not in your passwords.",
   title: "Error Message", JOptionPane.ERROR_MESSAGE);

  frame.dispose();

  return;
}
```

This code sends an error message if the nickname entered when viewing is not in the Passwords text file.

In order for Franklin to learn the software easier, I utilized error messages to guide him through the software inputs.

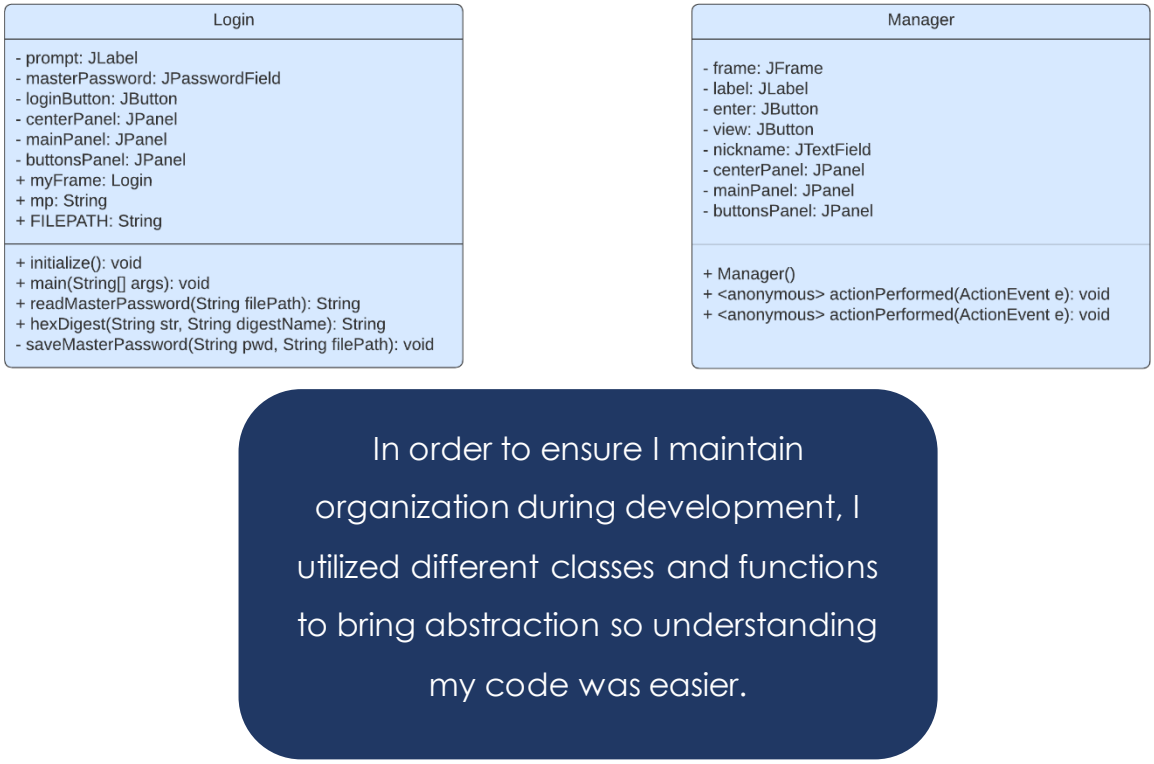
Event Handling:

```
loginButton.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        // Convert char[] to String.
        String password = new String(masterPassword.getPassword());

        String verification = hexDigest(password, digestName: "SHA-256");

        if(verification.equals(mp)) {
            myFrame.dispose();
            Manager newWindow = new Manager();
        }
        else {
            JOptionPane.showMessageDialog(myFrame, message: "Invalid Master Password. Please Try Again.",
                title: "Error Message", JOptionPane.INFORMATION_MESSAGE);
        }
    }
});
```

These lines of code are used to add functionality to the "Login" button.

Java events are called when certain actions occur in a program, such as clicking a button. To code the functions that are called when such things happen, I created "ActionListeners," and I passed them to the respective methods, such as "loginButton.addActionListener."

Modularity:



In order to ensure I maintain organization during development, I utilized different classes and functions to bring abstraction so understanding my code was easier.