CEN 4078 Secure Software Development

Project name: Programming Exercise 1

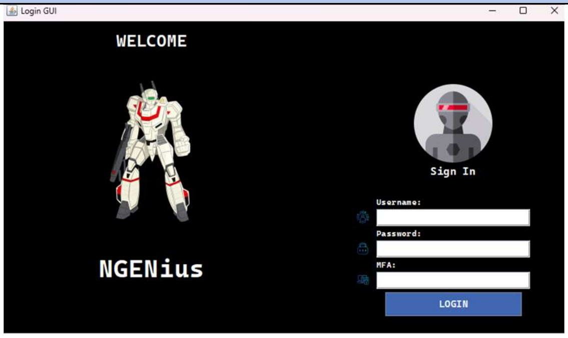
Module name: Login Module with Input Validation & Type Checking

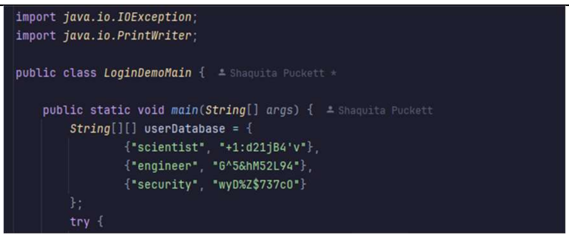
Developer: Shaquita Puckett

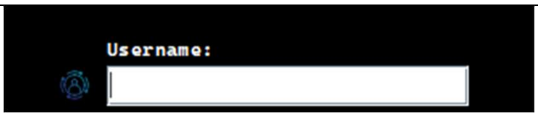
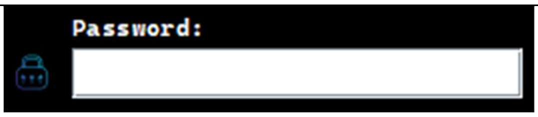
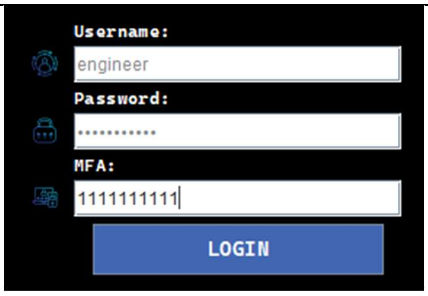
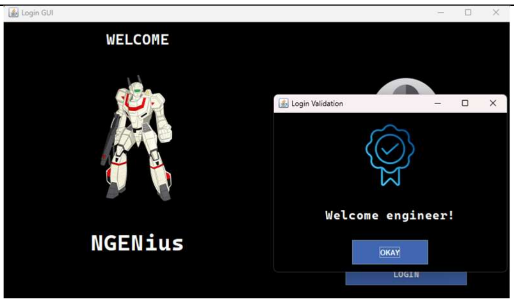
Course: CEN 4078 Secure Software Development

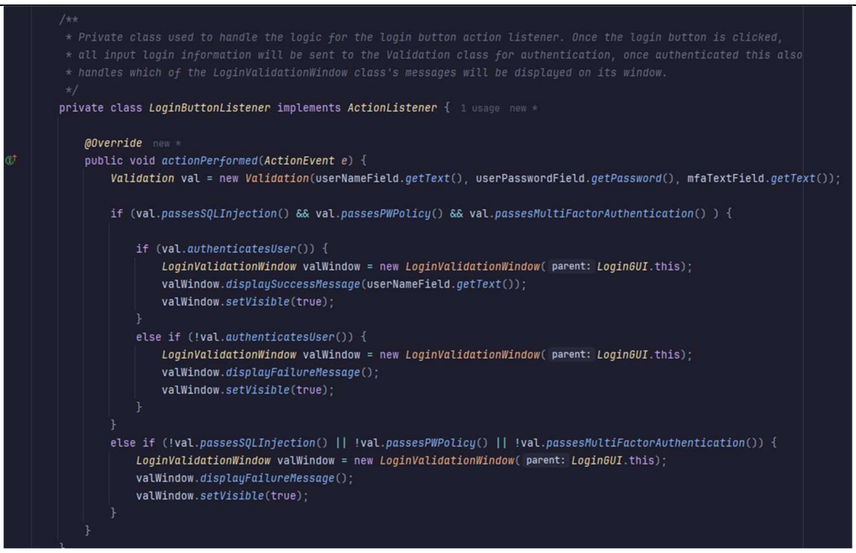
Date: 2/1/2026

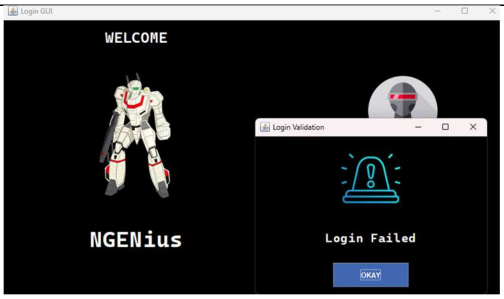
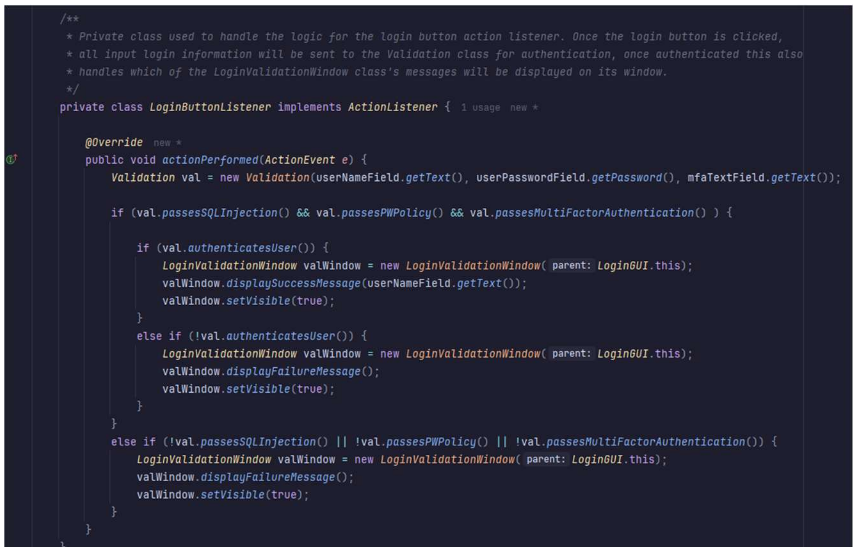
Table 1. Login Module Test report

Data	***Proof*** - ***may be copy of code snippet*** - ***may be screen captures*** - ***may be written proof but must have data not just prose***
Login Module	
Create an application to allow users to login	 Login GUI WELCOME NGENius Sign In Username: Password: MFA: LOGIN *The app is launched from the LoginDemoMain.java file.*
Create three users: (include the usernames)	*Usernames I used:* *-scientist* *-engineer* *-security*
Create three passwords: (include the passwords)	*Passwords I used:* *-scientist +1:d21jB4'v* *-engineer G^5&hM52L94* *-security wyD%Z$737cO*

Database: An array of the usernames and passwords	 `import java.io.IOException;`
	`import java.io.PrintWriter;`
	
	`public class LoginDemoMain {  Shaquita Puckett *`
	
	`    public static void main(String[] args) {  Shaquita Puckett`
	`        String[][] userDatabase = {`
	`                {"scientist", "+1:d21jB4'v"},`
	`                {"engineer", "G^5&hM52L94"},`
	`                {"security", "wyD%Z$737c0"}`
	`        };`
	`        try {`
	
	The database array was created in the LoginDemoMain.java file.

Prompt the user for a username	 *The GUI displays a field for inputting a username.*
Prompt the user for a password	 *The GUI displays a field or inputting a password.*
The password should not be visible	 Username: engineer Password: ••••••••••• MFA: 1111111111 LOGIN *Anything typed into the password field is displayed as gray dots instead of actual characters.*
If the login is successful, display a welcome message with the username in it	 Login GUI WELCOME NGENius Login Validation OKAY LOGIN Welcome engineer!

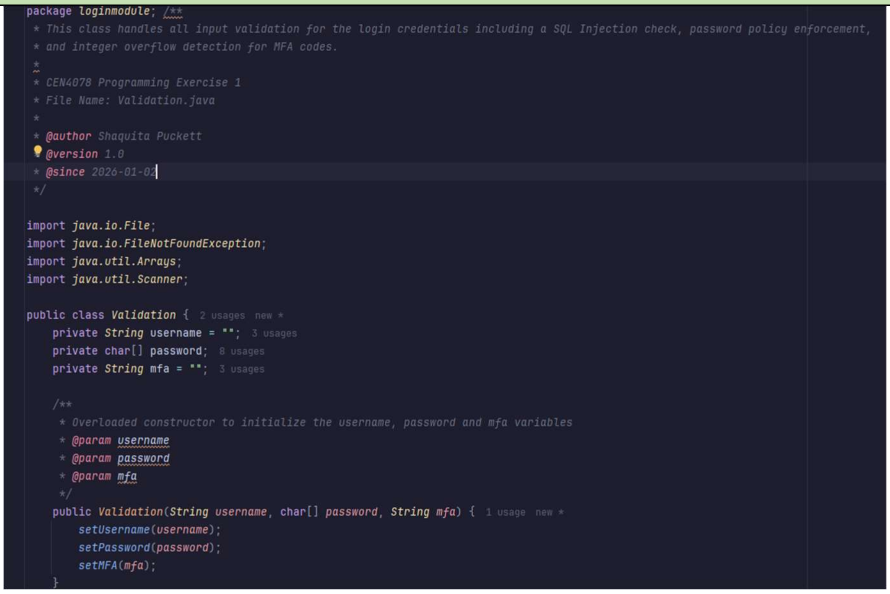
	
	```
	/**
	 * Private class used to handle the logic for the login button action listener. Once the login button is clicked,
	 * all input login information will be sent to the Validation class for authentication, once authenticated this also
	 * handles which of the LoginValidationWindow class's messages will be displayed on its window.
	 */
	private class LoginButtonListener implements ActionListener {  1 usage  new *

	    @Override  new *
	    public void actionPerformed(ActionEvent e) {
	        Validation val = new Validation(userNameField.getText(), userPasswordField.getPassword(), mfaTextField.getText());

	        if (val.passesSQLInjection() && val.passesPWPolicy() && val.passesMultiFactorAuthentication() ) {

	            if (val.authenticatesUser()) {
	                LoginValidationWindow valWindow = new LoginValidationWindow( parent: LoginGUI.this);
	                valWindow.displaySuccessMessage(userNameField.getText());
	                valWindow.setVisible(true);
	            }
	            else if (!val.authenticatesUser()) {
	                LoginValidationWindow valWindow = new LoginValidationWindow( parent: LoginGUI.this);
	                valWindow.displayFailureMessage();
	                valWindow.setVisible(true);
	            }
	        }
	        else if (!val.passesSQLInjection() || !val.passesPWPolicy() || !val.passesMultiFactorAuthentication()) {
	            LoginValidationWindow valWindow = new LoginValidationWindow( parent: LoginGUI.this);
	            valWindow.displayFailureMessage();
	            valWindow.setVisible(true);
	        }
	    }
	```
	 If the login is successfully validated with the Validation class, the LoginValidationWindow class popup displays with "Welcome + username!"

If the login is not successful, handle it in a "Secure Software Development" way.	 Login GUI
	WELCOME
	NGENius
	Login Validation
	Login Failed
	OKAY 
	```
	/**
	 * Private class used to handle the logic for the login button action listener. Once the login button is clicked,
	 * all input login information will be sent to the Validation class for authentication, once authenticated this also
	 * handles which of the LoginValidationWindow class's messages will be displayed on its window.
	 */
	private class LoginButtonListener implements ActionListener {  1 usage  new *

	    @Override  new *
	    public void actionPerformed(ActionEvent e) {
	        Validation val = new Validation(userNameField.getText(), userPasswordField.getPassword(), mfaTextField.getText());

	        if (val.passesSQLInjection() && val.passesPWPolicy() && val.passesMultiFactorAuthentication() ) {

	            if (val.authenticatesUser()) {
	                LoginValidationWindow valWindow = new LoginValidationWindow( parent: LoginGUI.this);
	                valWindow.displaySuccessMessage(userNameField.getText());
	                valWindow.setVisible(true);
	            }
	            else if (!val.authenticatesUser()) {
	                LoginValidationWindow valWindow = new LoginValidationWindow( parent: LoginGUI.this);
	                valWindow.displayFailureMessage();
	                valWindow.setVisible(true);
	            }
	        }
	        else if (!val.passesSQLInjection() || !val.passesPWPolicy() || !val.passesMultiFactorAuthentication()) {
	            LoginValidationWindow valWindow = new LoginValidationWindow( parent: LoginGUI.this);
	            valWindow.displayFailureMessage();
	            valWindow.setVisible(true);
	        }
	    }
	```

If the user information cannot be validated through the Validation class, the LoginValidationWindow class popup displays “Login Failed” message without revealing why the login failed.

Input Validation & Type Checking

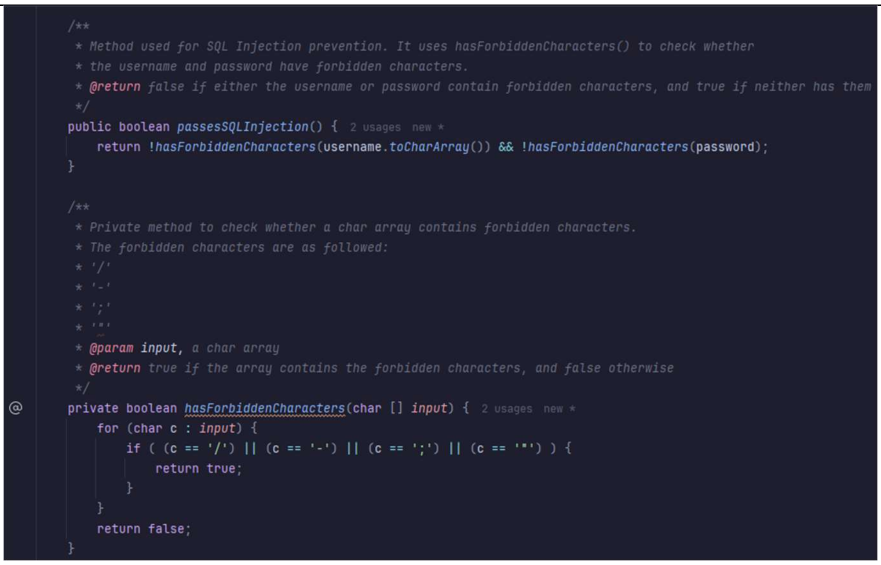
Create a validation class in your login module



```java
package loginmodule; /**
 * This class handles all input validation for the login credentials including a SQL Injection check, password policy enforcement,
 * and integer overflow detection for MFA codes.
 *
 * CEN4078 Programming Exercise 1
 * File Name: Validation.java
 *
 * @author Shaquita Puckett
 * @version 1.0
 * @since 2026-01-02
 */

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Scanner;

public class Validation {  2 usages  new *
    private String username = "";  3 usages
    private char[] password;  8 usages
    private String mfa = "";  3 usages

    /**
     * Overloaded constructor to initialize the username, password and mfa variables
     * @param username
     * @param password
     * @param mfa
     */
    public Validation(String username, char[] password, String mfa) {  1 usage  new *
        setUsername(username);
        setPassword(password);
        setMFA(mfa);
    }
```

This class handles all input validation for the login credentials including a SQL Injection check, password policy enforcement, and integer overflow detection for MFA codes.

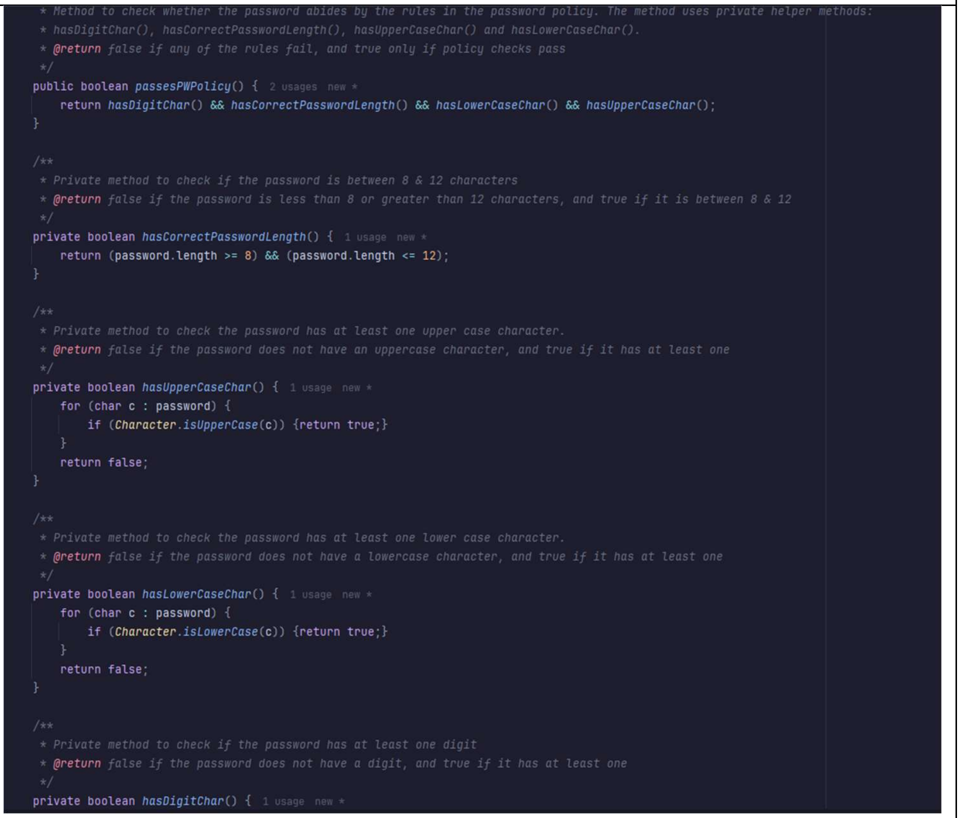
Create a SQL Injection validation method



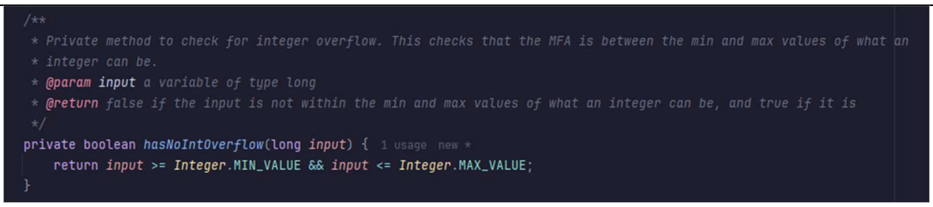
```java
    /**
     * Method used for SQL Injection prevention. It uses hasForbiddenCharacters() to check whether
     * the username and password have forbidden characters.
     * @return false if either the username or password contain forbidden characters, and true if neither has them
     */
    public boolean passesSQLInjection() {  2 usages  new *
        return !hasForbiddenCharacters(username.toCharArray()) && !hasForbiddenCharacters(password);
    }

    /**
     * Private method to check whether a char array contains forbidden characters.
     * The forbidden characters are as followed:
     * '/'
     * '-'
     * ';'
     * '"'
     * @param input, a char array
     * @return true if the array contains the forbidden characters, and false otherwise
     */
@   private boolean hasForbiddenCharacters(char [] input) {  2 usages  new *
        for (char c : input) {
            if ( (c == '/') || (c == '-') || (c == ';') || (c == '"') ) {
                return true;
            }
        }
        return false;
    }
```

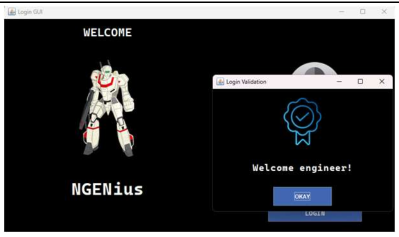
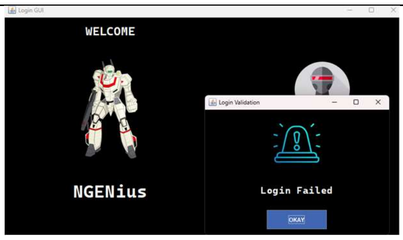
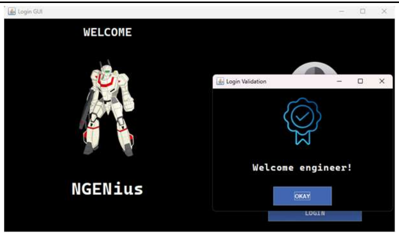
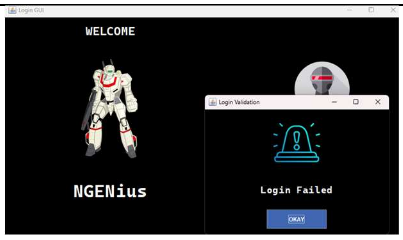
passesSQLInjection() *-> Method used for SQL Injection prevention. It uses* ***hasForbiddenCharacters()*** *to check whether the username and password have forbidden characters.*

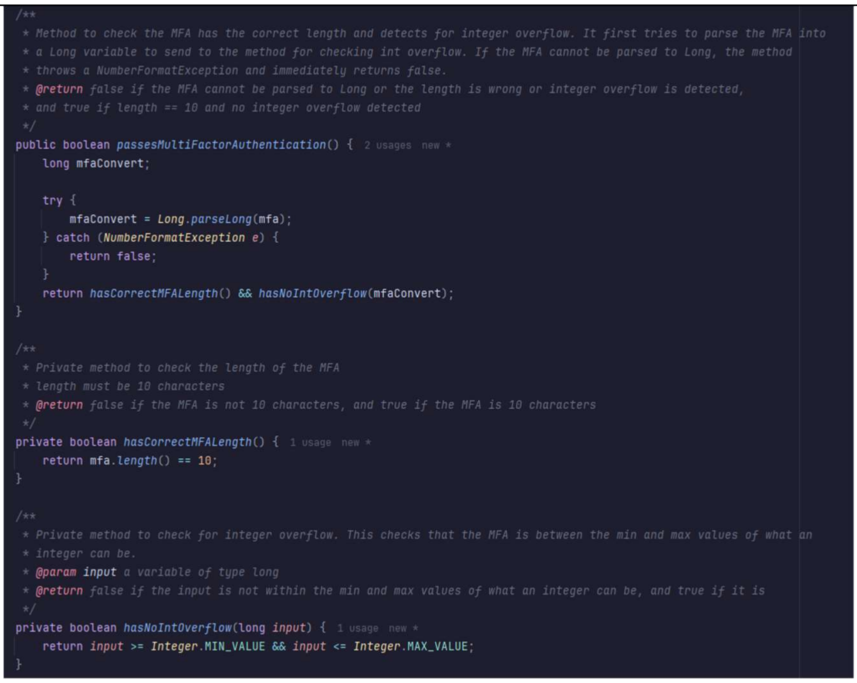
	hasForbiddenCharacters() *-> Private method to check whether a char array contains forbidden characters.* *The forbidden characters are as followed:* ** '/'* ** '-'* ** ';'* ** '"'* *@param input, a char array* *@return true if the array contains the forbidden characters, and false otherwise*

Create a Password Policy validation method	
	`* Method to check whether the password abides by the rules in the password policy. The method uses private helper methods:`
	`* hasDigitChar(), hasCorrectPasswordLength(), hasUpperCaseChar() and hasLowerCaseChar().`
	`* @return false if any of the rules fail, and true only if policy checks pass`
	`*/`
	`public boolean passesPWPolicy() {  2 usages  new *`
	`    return hasDigitChar() && hasCorrectPasswordLength() && hasLowerCaseChar() && hasUpperCaseChar();`
	`}`
	
	`/**`
	`* Private method to check if the password is between 8 & 12 characters`
	`* @return false if the password is less than 8 or greater than 12 characters, and true if it is between 8 & 12`
	`*/`
	`private boolean hasCorrectPasswordLength() {  1 usage  new *`
	`    return (password.length >= 8) && (password.length <= 12);`
	`}`
	
	`/**`
	`* Private method to check the password has at least one upper case character.`
	`* @return false if the password does not have an uppercase character, and true if it has at least one`
	`*/`
	`private boolean hasUpperCaseChar() {  1 usage  new *`
	`    for (char c : password) {`
	`        if (Character.isUpperCase(c)) {return true;}`
	`    }`
	`    return false;`
	`}`
	
	`/**`
	`* Private method to check the password has at least one lower case character.`
	`* @return false if the password does not have a lowercase character, and true if it has at least one`
	`*/`
	`private boolean hasLowerCaseChar() {  1 usage  new *`
	`    for (char c : password) {`
	`        if (Character.isLowerCase(c)) {return true;}`
	`    }`
	`    return false;`
	`}`
	
	`/**`
	`* Private method to check if the password has at least one digit`
	`* @return false if the password does not have a digit, and true if it has at least one`
	`*/`
	`private boolean hasDigitChar() {  1 usage  new *`
	
	passesPWPolicy() *-> Method to check whether the password abides by the rules in the password policy. The method uses private helper methods:****hasDigitChar(), hasCorrectPasswordLength(), hasUpperCaseChar()*** *and* ***hasLowerCaseChar().***
	@return false if any of the rules fail, and true only if policy checks pass
	
	hasDigitChar() *-> Private method to check if the password has at least one digit*
	@return false if the password does not have a digit, and true if it has at least one

	hasCorrectPasswordLength() *-> Private method to check if the password is between 8 & 12 characters* *@return false if the password is less than 8 or greater than 12 characters, and true if it is between 8 & 12* ***hasUpperCaseChar()*** *-> Private method to check the password has at least one upper case character.* *@return false if the password does not have an uppercase character, and true if it has at least one* ***hasLowerCaseChar()*** *-> Private method to check the password has at least one lower case character.* *@return false if the password does not have a lowercase character, and true if it has at least one*

Create an Integer Overflow validation method	
	`/**`
	` * Private method to check for integer overflow. This checks that the MFA is between the min and max values of what an`
	` * integer can be.`
	` * @param input a variable of type long`
	` * @return false if the input is not within the min and max values of what an integer can be, and true if it is`
	` */`
	`private boolean hasNoIntOverflow(long input) {  1 usage  new *`
	`    return input >= Integer.MIN_VALUE && input <= Integer.MAX_VALUE;`
	`}`
	
	hasNoIntOverflow() *-> Private method to check for integer overflow. This checks that the MFA is between the min and max values of what an integer can be.*
	@param input a variable of type long
	@return false if the input is not within the min and max values of what an integer can be, and true if it is

Validate the username with SQL Injection. Show pass and fail.	 Login GUI WELCOME NGENius Login Validation Welcome engineer! OKAY LOGIN Login GUI WELCOME NGENius Login Validation Login Failed OKAY *Pass/Fail shown by validation checking within the validation class and displaying on the popup window*
Validate the password with SQL Injection and Password Policy Show pass and fail.	 Login GUI WELCOME NGENius Login Validation Welcome engineer! OKAY LOGIN Login GUI WELCOME NGENius Login Validation Login Failed OKAY

	Pass/Fail shown by validation checking within the validation class and displaying on the popup window
Validate the MFA with Integer Overflow Show pass and fail.	 ***passesMultiFactorAuthentication*()** *-> Method to check the MFA has the correct length and detects for integer overflow. It first tries to parse the MFA into a Long variable to send to the method for checking int overflow. If the MFA cannot be parsed to Long, the method throws a NumberFormatException and immediately returns false.* *@return false if the MFA cannot be parsed to Long or the length is wrong or integer overflow is detected,and true if length == 10 and no integer overflow detected* ***hasCorrectMFALength*()** *-> Private method to check the length of the MFA. length must be 10 characters* *@return false if the MFA is not 10 characters, and true if the MFA is 10 characters* ***hasNoIntOverflow*(long *input*)** *-> Private method to check for integer overflow. This checks that the MFA is between the min and max values of what an integer can be.* *@param input a variable of type long* *@return false if the input is not within the min and max values of what an integer can be, and true if it is*
Summary	

```java
/**
 * Method to check the MFA has the correct length and detects for integer overflow. It first tries to parse the MFA into
 * a Long variable to send to the method for checking int overflow. If the MFA cannot be parsed to Long, the method
 * throws a NumberFormatException and immediately returns false.
 * @return false if the MFA cannot be parsed to Long or the length is wrong or integer overflow is detected,
 * and true if length == 10 and no integer overflow detected
 */
public boolean passesMultiFactorAuthentication() {  2 usages  new *
    long mfaConvert;

    try {
        mfaConvert = Long.parseLong(mfa);
    } catch (NumberFormatException e) {
        return false;
    }
    return hasCorrectMFALength() && hasNoIntOverflow(mfaConvert);
}

/**
 * Private method to check the length of the MFA
 * length must be 10 characters
 * @return false if the MFA is not 10 characters, and true if the MFA is 10 characters
 */
private boolean hasCorrectMFALength() {  1 usage  new *
    return mfa.length() == 10;
}

/**
 * Private method to check for integer overflow. This checks that the MFA is between the min and max values of what an
 * integer can be.
 * @param input a variable of type long
 * @return false if the input is not within the min and max values of what an integer can be, and true if it is
 */
private boolean hasNoIntOverflow(long input) {  1 usage  new *
    return input >= Integer.MIN_VALUE && input <= Integer.MAX_VALUE;
}
```

Did you meet all the requirements. If not, describe why?	Yes all requirements were met. The app creates a database of usernames and password and then validates username, password and MFA input by way of a Validation class.