Jordan Bassett & Lizbeth Garcia-Lopez

CS236 - Final Project (Last Version) Submission Box

03/19/2025

## Questions

“Test and revise the application to do the required tasks stated in the first three phases.”

Done. The program has been tested and revised to meet the requirements for the GUI, password creation logic, file handling, and game functionality. The concepts of polymorphism, inheritance, and interfaces were used, as required by the goals of the class.

“if you are going to use two screens, how would you do it? The new screen displays the remembered phrases entered from the first screen. Implement the application to do this task.”

The program uses two screens: the Password Game and Password Lab. The Password Lab allows users to add and modify phrases and common words, which are then displayed and accessible on the Password Game screen for password creation. This functionality was implemented through the use of plain text files (phrases.txt and common\_words.txt). Plain text wase chosen to allow users to easily edit and update the content outside the program. These files allow the data entered on the first screen is available on the second screen.

## Code Color Legend

Class Relationship ("is a" and/or "has a") - #006400

Declaration and Instantiation of Objects - #0000FF

Information to Disk - #8B4513

Class Methods that Create Strong Passwords - #FFFF00

GUI for Gameplay - #FF0000

GUI for String Selections Used for Password Creation - #800080

Second Window Implementation in Phase 4 - #008080

Word does not have required color options for some of the colors in its highlight tool (Brown, Dark Green). We adapted by using Word’s shading tool, which allows the use of a color picker.  
Also, not every category of was assigned a color (Second Window), so we picked one (Teal)  
When concepts fit under one or more color categories, it was impossible to use more than one color for highlighting given the above constraints that forced me to use shading (which goes by line, not by character). Therefore, we picked one or alternated. Where text colors overlapped, we were able to switch colors mid-code/sentence.

There is a large amount of overlap between the two required categories “Classes and Interfaces Managing Passwords and Scores” and “Classes Containing GUI Components.” We split up the classes into two the two categories as made sense given the nonsensical constraints and relied on the tedious highlighting system to point out the desired programming concepts.

## Classes and Interfaces Managing Passwords and Scores

### PasswordValidator.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordValidator.java

Author(s): Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

Validates password strength and provides feedback

Implements difficulty scaling, detects common words, and offers hints for improvement

Features:

- Evaluates password strength based on length, character variety, and common word usage

- Provides dynamic hints for password improvement

- Scales requirements by difficulty level

Dependencies:

- Extends: PasswordComponent.java

- Uses: PasswordFileManager.java, HintManager.java \*/

package passwordfactory;

import java.util.List;

// Validates password strength and provides feedback

public class PasswordValidator extends PasswordComponent

{

private HintManager hintManager;

private List<String> commonWords;

/\*PasswordValidator is a subclass of PasswordComponent and hasaHintManager for managing hints and PasswordFileManager for loading common words from a file  
It aligns with the UML\*/

private String difficulty;

private final int commonWordPenalty;

private final int moderateThreshold;

private final int strongThreshold;

/\*\*

\* Constructor for PasswordValidator.

\* @param fileManager The file manager to load common words.

\* @param difficulty The selected difficulty level (easy, medium, hard).

\*/

public PasswordValidator(PasswordFileManager fileManager, String difficulty)

{

this.hintManager = new HintManager();

this.commonWords = fileManager.loadCommonWords("common\_words.txt");

/\*PasswordValidator instantiates a HintManager object to manage dynamic hints and uses the PasswordFileManager to load common words for password evaluation. \*/

/\* loadCommonWords()` method retrieves common words from the `common\_words.txt` file. This file is read and used to evaluate the password for common word penalties (fits in two categories, so two colors) \*/

this.difficulty = difficulty.toLowerCase();

switch (this.difficulty)

{

case "easy":

commonWordPenalty = 2;

moderateThreshold = 3;

strongThreshold = 5;

break;

case "medium":

commonWordPenalty = 3;

moderateThreshold = 4;

strongThreshold = 5;

break;

case "hard":

commonWordPenalty = 4;

moderateThreshold = 4;

strongThreshold = 5;

break;

default:

commonWordPenalty = 2;

moderateThreshold = 3;

strongThreshold = 5;

}

}

/\*\*

\* Evaluates the strength of the password.

\* @param password The password to evaluate.

\* @return 0 = Weak, 1 = Moderate, 2 = Strong

\*/

/\* The checkStrength() method evaluates the password strength by checking its length, the presence of uppercase letters, numbers, and special characters, and looks for common words\*/

@Override

public int checkStrength(String password)

{

int score = 0;

hintManager.clearDynamicHints();

if (password.length() >= 8) {

score++;

}

else {

hintManager.addDynamicHint(new Hint("Try making it longer (8+ characters).", "Length"));

}

/\*Hint objects are instantiated and added to the list of dynamic hints, providing real-time feedback for password strength evaluation\*/

if (password.matches(".\*[A-Z].\*")) {

score++;

}

else {

hintManager.addDynamicHint(new Hint("Add at least one uppercase letter.", "Uppercase"));

}

/\*Hint objects are instantiated and added to the list of dynamic hints, providing real-time feedback for password strength evaluation\*/

if (password.matches(".\*[a-z].\*")) {

score++;

}

else {

hintManager.addDynamicHint(new Hint("Add at least one lowercase letter.", "Lowercase"));

}

/\*Hint objects are instantiated and added to the list of dynamic hints, providing real-time feedback for password strength evaluation\*/

if (password.matches(".\*\\d.\*")) {

score++;

}

else {

hintManager.addDynamicHint(new Hint("Add at least one number.", "Numbers"));

}

/\*Hint objects are instantiated and added to the list of dynamic hints, providing real-time feedback for password strength evaluation\*/

if (password.matches(".\*[!@#$%&\*].\*")) {

score++;

}

else {

hintManager.addDynamicHint(new Hint("Add at least one special character (!@#$%&\*).", "Special Characters"));

/\*Hint objects are instantiated and added to the list of dynamic hints, providing real-time feedback for password strength evaluation\*/

}

String matchedCommonWord = getMatchedCommonWord(password.toLowerCase(), commonWords);

if (matchedCommonWord != null)

{

hintManager.addDynamicHint(new Hint(matchedCommonWord + " is a common word! Try something more unique.", "Common Words"));

score -= commonWordPenalty;

if (difficulty.equals("hard")) {

return 0;

}

}

if (score >= strongThreshold) {

return 2;

}

else if (score >= moderateThreshold) {

return 1;

}

else {

return 0;

}

}

/\*\*

\* Provides feedback for improving a password.

\* @param password The password to analyze.

\* @return Feedback message.

\*/

@Override

public String giveFeedback(String password)

{

int strength = checkStrength(password);

if (strength == 2) {

return "Strong Password! Great job!";

}

if (hintManager.getDynamicHints().isEmpty()) {

return "Good! But there's room for improvement.";

}

StringBuilder feedback = new StringBuilder("Suggestions:\n");

for (Hint hint : hintManager.getDynamicHints()) {

feedback.append("- ").append(hint.getHint()).append("\n");

}

return feedback.toString();

}

/\*\*

\* Checks if the password contains any common word.

\* @param password The password to check.

\* @param commonWords List of common words.

\* @return The matched common word, or null if none found.

\*/

private String getMatchedCommonWord(String password, List<String> commonWords)

{

for (String word : commonWords)

{

if (password.contains(word)) {

return word;

}

}

return null;

}

/\*\*

\* Returns the HintManager instance.

\* @return The HintManager.

\*/

public HintManager getHintManager() {

return hintManager;

}

}

### PasswordComponent.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordComponent.java

Author(s): Lizbeth Garcia-Lopez & Jordan Bassett

Last Updated: March 16, 2025

Description:

This is an abstract superclass for password-related validation logic.

It enforces password validation rules while allowing subclasses to imlement

specific validation and feedback logic.

This class implements the PasswordChecker interface and is extended by PasswordValidator

Features:

- Implements the PasswordChecker interface.

- Defines abstract methods for evaluating password strength and generating feedback.

- Provides a shared utility method to check if a password contains common weak words.

Dependencies:

- Superclass for: PasswordValidator.java

- Implements: PasswordChecker.java \*/

package passwordfactory;

//Abstract superclass for password validation logic

public abstract class PasswordComponent implements PasswordChecker

/\* The PasswordComponent ‘is-a’ subclass of PasswordChecker and provides common functionality for its subclasses, such as checking for common weak words in a password\*/

{

/\*\*

Abstract method for checking password strength

Must be implemented by subclasses

@param password The password to evaluate.

@return Strength level as an integer:

0 = Weak

1 = Moderate

2 = Strong

\*/

@Override

public abstract int checkStrength(String password);

/\*\*

\* Abstract method for providing feedback about a password.

\* Must be implemented by subclasses

\* @param password The password to analyze

\* @return A feedback message with suggestions for improvement. \*/

@Override

public abstract String giveFeedback(String password);

/\*These abstract classes align with the UML\*/

/\*\*

\* Helper method to check if the given password contains a common weak word

\* @param password The password to check

\* @param commonWords array of common weak words

\* @return True if the password contains a common word, false otherwise

\*/

protected boolean containsCommonWord(String password, String[] commonWords)

{

for (String word : commonWords)

{

if (password.toLowerCase().contains(word.toLowerCase()))

{

return true; // Password contains a weak common word

}

}

return false;

}

}

### PasswordChecker.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordChecker.java

Author(s): Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

This interface defines methods for evaluating password strength and providing feedback

It is implemented by PasswordValidator to enforce password rules and provide user guidance.

Features:

- Defines a contract for password validation strategies

- Ensures all password evaluators provide strength checking and feedback methods.

Dependencies:

- Implemented by: PasswordValidator.java \*/

package passwordfactory;

// PasswordChecker interface defines methods for password validation and feedback.

// is implemented by PasswordValidator.

public interface PasswordChecker

/\* PasswordChecker interface is implemented by PasswordValidator, enforcing that all password validation strategies must provide methods for checking strength and providing feedback \*/

/\*The interface aligns with the UML\*/

{

/\*\*

\* Evaluates the strength of a given password

\* @param password The password to check

\* @return Strength level as an integer:

\* 0 = Weak

\* 1 = Moderate

\* 2 = Strong

\*/

int checkStrength(String password);

/\*\*

\* Provides feedback on how to improve password strength.

\* @param password The password to analyze.

\* @return A feedback message about password strength and recommendations.

\*/

String giveFeedback(String password);

}

### PasswordGame.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordGame.java

Author(s): Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

Manages the main gameplay loop for Password Factory. Handles user input, scoring,

timer management, and password validation during a timed round of the game.

Features:

- Runs a timed challenge to create strong passwords.

- Provides real-time hints based on password input.

- Displays helpful phrases and allows the user to add new phrases.

- Tracks and saves submitted passwords and their strength ratings.

Dependencies:

- Uses PasswordValidator for password evaluation.

- Uses PasswordFileManager for file operations (phrases and logging).

- Called from: GameSetupScreen.java

\*/

package passwordfactory;

import javafx.animation.KeyFrame;

import javafx.animation.Timeline;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.\*;

import javafx.stage.Stage;

import javafx.util.Duration;

import java.util.List;

import java.util.Random;

//Manages the main gameplay loop, user interactions, and password validation

public class PasswordGame

{

// Class variables

private String difficulty; //difficulty level selected (Easy, Medium, Hard)

private final Scene gameScene; //Main scene for the game window

private Stage gameStage; // Game window stage

private final int roundTime; //Total time for the round, based on difficulty

private int timeLeft; //Countdown timer for the current round

private int score = 0; //player score, based on password strength

private int passwordsSubmitted = 0; // Number of passwords submitted this round

// UI Labels for displaying game status

private final Label timerLabel; //Displays time left in the round

private final Label scoreLabel; //Displays current score

private Label resultLabel; //Displays strength feedback for the last password

private final Label hintLabel; //Displays hints for improving the password

private Timeline timer; // Timer for the countdown (JavaFX Timeline)

//User input and output components

private TextField passwordEntryField; //Field where user types passwords

private final ListView<String> passwordHistoryView; //Shows passwords submitted in this round

private ListView<String> phraseListView; //Displays helpful phrases to add to passwords

//Random number generator (currently unused)

//private Random random = new Random();

//Core components

private final PasswordChecker passwordChecker; //Password strength evaluator (PasswordValidator)

private PasswordFileManager fileManager; //Manages phrase/common word files and password logs

/\*\*

\* Constructor for PasswordGame.

\* Initializes the game UI and sets up game logic based on difficulty.

\* @param difficulty The selected difficulty level (Easy, Medium, Hard).

\*/

public PasswordGame(String difficulty)

{

this.difficulty = difficulty;

// Set round time based on difficulty level

switch (difficulty.toLowerCase())

{

case "easy":

roundTime = 60;

break;

case "medium":

roundTime = 45;

break;

case "hard":

roundTime = 30;

break;

default:

roundTime = 60;

}

this.timeLeft = roundTime;

//Initialize file manager and password validator

fileManager = new PasswordFileManager("phrases.txt", "password\_log.txt");

/\* uses PasswordFileManager for file handling ops

instantiates PasswordFileManager for handling file operations\*/

fileManager.initPasswordLog();

passwordChecker = new PasswordValidator(fileManager, difficulty);

/\* uses PasswordValidator to evaluate password strength during the game

instantiates PasswordValidator for password evaluation\*/

//instruction label at the top of the game screen

Label instructionLabel = new Label("Create as many strong passwords as you can in " + roundTime + " seconds!");

//Password input field with real-time hint updates

passwordEntryField = new TextField();

passwordEntryField.setPromptText("Enter your password");

passwordEntryField.setOnKeyReleased(e -> updateHints(passwordEntryField.getText()));

/\* This part of the GUI creates a TextField where users input their passwords. As they type, the system provides real-time hints on how to improve the password strength \*/

//Submit button for submitting passwords

Button submitButton = new Button("Submit Password");

submitButton.setOnAction(e -> submitPassword());

/\* button triggers password submission, calling the submitPassword(), which evaluates the password strength, updates the score, and displays feedback \*/

//labels for displaying hints, timer, score, and result feedback

hintLabel = new Label("Hints will appear here as you type.");

hintLabel.setWrapText(true);

hintLabel.setMaxWidth(400);

timerLabel = new Label("Time Left: " + timeLeft + "s");

/\*label shows the remaining time for the round, its a countdown for the player \*/

scoreLabel = new Label("Score: " + score);

/\*displays the current score, which updates based on the strength of the passwords submitted

resultLabel = new Label();

resultLabel.setWrapText(true);

resultLabel.setMaxWidth(400);

/\* displays feedback for the most recently submitted password strength and suggestions \*/

// shows submitted passwords for this round

passwordHistoryView = new ListView<>();

passwordHistoryView.setPrefWidth(400);

/\* ListView shows the passwords submitted during the round\*/

//shows 'helpful' phrases and allows adding new ones

List<String> phrases = fileManager.loadPhrases();

phraseListView = new ListView<>();

phraseListView.getItems().addAll(phrases);

phraseListView.setPrefWidth(200);

/\* ListView that shows phrases that can be selected and added to the password\*/

/\* user can select a phrase from the phraseListView, which is then appended to the password input field \*/

phraseListView.setOnMouseClicked(e -> appendPhraseToPassword());

/\*allows users to select phrases from phraseListView, which are then appended to the password input field to assist in creating stronger passwords\*/

TextField newPhraseField = new TextField();

newPhraseField.setPromptText("Enter new phrase");

Button addPhraseButton = new Button("Add Phrase");

addPhraseButton.setOnAction(e ->

{

boolean success = fileManager.addPhrase(newPhraseField.getText());

if (success)

{

phraseListView.getItems().add(newPhraseField.getText().trim());

resultLabel.setText("Phrase added successfully!");

newPhraseField.clear();

}

else {

resultLabel.setText("\u26A0 Enter a valid phrase!");

}

});

/\* user can enter a new phrase in the newPhraseField, which is then added to the list of phrases. The new phrase is passed into the game and available for password creation \*/

Button backButton = new Button("Back to Menu");

backButton.setOnAction(e ->

{

timer.stop();

fileManager.closeLog();

gameStage.close();

/\* back button, stop timer and closes file ops, closes the stage\*/

new App().start(new Stage());

/\*handles second window creation by transitioning between screens, returning to the main menu when the game ends\*/

});

//Left side: Helpful phrases and phrase adding UI

VBox leftSide = new VBox(10,

new Label("Helpful Phrases:"),

phraseListView,

newPhraseField,

addPhraseButton,

backButton);

leftSide.setPadding(new Insets(10));

leftSide.setStyle("-fx-alignment: top-center;");

//Center pane: Instructions, password entry, submit, feedback, and score

VBox centerPane = new VBox(10,

instructionLabel,

passwordEntryField,

submitButton,

hintLabel,

resultLabel,

timerLabel,

scoreLabel);

centerPane.setPadding(new Insets(20));

centerPane.setStyle("-fx-alignment: center;");

//Right side: Passwords submitted this round

VBox rightSide = new VBox(10,

new Label("Passwords This Round:"),

passwordHistoryView);

rightSide.setPadding(new Insets(10));

rightSide.setStyle("-fx-alignment: top-left;");

//Combine all sections into the main layout

HBox mainLayout = new HBox(30, leftSide, centerPane, rightSide);

mainLayout.setPadding(new Insets(10));

/\* Split the screen into 3 vertical boxes, and add the UI components to them. Alignment and insets were basically tweaking numbers until it looked ok\*/

gameScene = new Scene(mainLayout, 1100, 600);

/\*instantiates Scene for the game window\*/

}

// Displays the game window and starts the round timer.

public void show()

{

gameStage = new Stage();

gameStage.setTitle("Password Factory - Time Attack Mode");

gameStage.setScene(gameScene);

gameStage.setResizable(false);

gameStage.setWidth(1100);

gameStage.setHeight(600);

gameStage.show();

/\*Makes a game window\*/

startRoundTimer();

}

//Starts the countdown timer for the game round.

// Updates the timer label and ends the round when time runs out.

private void startRoundTimer()

{

timer = new Timeline(new KeyFrame(Duration.seconds(1), e ->

{

timeLeft--;

timerLabel.setText("Time Left: " + timeLeft + "s");

if (timeLeft <= 0)

{

timer.stop();

endRound();

}

}));

timer.setCycleCount(roundTime);

timer.play();

}

/\* Handles password submission, evaluates strength, updates score,

and logs the password with its feedback. \*/

private void submitPassword()

{

String password = passwordEntryField.getText();

if (password == null || password.isEmpty())

{

resultLabel.setText("\u26A0 Enter a password first!");

return;

}

password = password.replaceAll("\\s+", ""); // Strip spaces

/\* user inputs a password, and any spaces are stripped before the password is looked at. This makes sure that the password meets strength requirements \*/

int strength = passwordChecker.checkStrength(password);

/\*sends the input to passwordChecker to score it\*/

String feedback = passwordChecker.giveFeedback(password);

String strengthLabel;

switch (strength)

{

case 2:

strengthLabel = "Strong";

score += 10;

break;

case 1:

strengthLabel = "Moderate";

score += 5;

break;

default:

strengthLabel = "Weak";

score += 0;

}

passwordsSubmitted++;

resultLabel.setText("Password Strength: " + strengthLabel + "\n" + feedback);

scoreLabel.setText("Score: " + score);

String historyEntry = "Password: " + password + "\n"

+ "Strength: " + strengthLabel + "\n"

+ feedback;

passwordHistoryView.getItems().add(historyEntry);

fileManager.savePassword(password, strengthLabel);

/\*saves passwords and their strength using the PasswordFileManager\*/

passwordEntryField.clear();

hintLabel.setText("");

}

/\*\*

\* Updates hints in real time as the user types a password.

\* Provides tips on improving the password's strength.

\* @param password The password being entered.

\*/

private void updateHints(String password)

{

StringBuilder hints = new StringBuilder();

if (password.length() < 8) {

hints.append("Try making it longer. ");

}

if (!password.matches(".\*[A-Z].\*")) {

hints.append("Add an uppercase letter. ");

}

if (!password.matches(".\*[a-z].\*")) {

hints.append("Add a lowercase letter. ");

}

if (!password.matches(".\*\\d.\*")) {

hints.append("Add a number. ");

}

if (!password.matches(".\*[!@#$%&\*].\*")) {

hints.append("Add a special character. ");

}

if (hints.length() == 0) {

hints.append("Looks good so far!");

}

hintLabel.setText(hints.toString());

}

//ends the round, disables input, displays final score, and closes the log file

private void endRound()

{

passwordEntryField.setDisable(true);

resultLabel.setText("\uD83C\uDFC6 Time's Up! Final Score: " + score

+ " | Passwords Submitted: " + passwordsSubmitted);

timerLabel.setText("Round Over");

fileManager.closeLog();

}

/\*The PasswordGame class manages the GUI for gameplay, including the password input field, the submit button, score, timer, and the display of feedback and hints\*/

//appends a selected helpful phrase to the password input field

private void appendPhraseToPassword()

{

String selectedPhrase = phraseListView.getSelectionModel().getSelectedItem();

if (selectedPhrase != null)

{

String currentText = passwordEntryField.getText();

passwordEntryField.setText(currentText + selectedPhrase.replaceAll("\\s", ""));

}

}

/\*The selected phrase is appended to the passwordEntryField to help the player create a stronger password. Spaces in the phrase are removed before adding it to the password \*/

}

### PasswordFileManager.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordFileManager.java

Author(s): Lizbeth Garcia-Lopez & Jordan Bassett

Last Updated: March 16, 2025

Description:

Manages all file operations for Password Factory. This includes loading and saving helpful phrases,

common words, and password logs. Provides methods for reading from and writing to text files

used in the game and Password Lab.

Features:

- Loads and saves helpful phrases (phrases.txt)

- Loads and saves common words (common\_words.txt)

- Logs submitted passwords and their strength ratings to a log file (password\_log.txt).

- Validates and adds new phrases to the existing phrase file

Dependencies:

- Used by PasswordGame.java, PasswordLab.java, and PasswordValidator.java

The project requirement in Phase 3 states:

"Store each remembered phrase on disk using a database or a random-access file."

After careful consideration, we made a design decision to store helpful phrases

and common words in plain text files (phrases.txt and common\_words.txt). This allows

users to easily add or modify multiple phrases in bulk without needing to interact

directly with the application. We prioritized usability and accessibility,

aligning with Columbia Basin College’s Inclusiveness and Accessibility policies,

by ensuring the system supports user-friendly data management.

However, recognizing that password logs contain potentially sensitive information,

we chose to implement a RandomAccessFile (binary file) for passwordLogFile.

This approach ensures that generated passwords are not stored in plaintext and

demonstrates proper file handling and security measures.

The binary log prevents casual access to sensitive data while fulfilling the Phase 3

requirement of implementing random access file processing.

We applied critical thinking as outlined in CBC’s Institutional Learning Outcomes.

We used random access file handling where it made the most practical and secure sense,

ensuring we addressed file processing requirements while delivering a better user experience.

\*/

package passwordfactory;

import java.io.\*;

import java.util.ArrayList;

import java.util.List;

//Manages file operations for Password Factory.

// Loads and saves phrases, common words, and logs password submissions.

public class PasswordFileManager

{

private final String phrasesFile;

private final String passwordLogFile;

private PrintWriPrintWriter writerter logWriter;

/\* PasswordFileManager ‘has-a’ PrintWriter to handle writing data (password logs) to files, and it ‘uses’ a BufferedReader to read from files. This is a relationship that allows PasswordFileManager to interact with the file system for both saving and loading data\*/

/\*These classes align with the UML\*/

/\*\*

\* Constructor for PasswordFileManager.

\* @param phrasesFile Name of the helpful phrases file

\* @param passwordLogFile Name of the password log file

\*/

public PasswordFileManager(String phrasesFile, String passwordLogFile)

{

this.phrasesFile = phrasesFile;

this.passwordLogFile = passwordLogFile;

}

/\*\*

\* Loads helpful phrases from the specified file

\* @return List of helpful phrases

\*/

public List<String> loadPhrases()

{

List<String> phrases = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(phrasesFile)))

/\*instantiates object BufferedReader for reading from the files\*/

{

String line;

while ((line = reader.readLine()) != null)

{

if (!line.trim().isEmpty()) {

phrases.add(line.trim());

}

}

}

catch (IOException e) {

System.out.println("Error loading phrases: " + e.getMessage());

}

return phrases;

}

/\*\*

\* Saves the provided phrases list to the specified file.

\* @param phrases List of helpful phrases.

\* @param filename File to save phrases to.

\*/

public void savePhrases(List<String> phrases, String filename)

{

try (PrintWriter writer = new PrintWriter(new FileWriter(filename)))

/\*instantiates object PrintWriter for writing to the files\*/

{

for (String phrase : phrases) {

writer.println(phrase);

}

}

catch (IOException e) {

System.out.println("Error saving phrases: " + e.getMessage());

}

}

/\*\*

\* Loads common words from the specified file.

\* Prevents duplicate entries manually.

\* @param filename Name of the common words file.

\* @return List of common words.

\*/

public List<String> loadCommonWords(String filename)

{

List<String> words = new ArrayList<>();

try (BufferedReader reader = new BufferedReader(new FileReader(filename)))

/\*instantiates object BufferedReader for reading from the files\*/

{

String line;

while ((line = reader.readLine()) != null)

{

String word = line.trim().toLowerCase();

if (!word.isEmpty() && !words.contains(word)) // prevent duplicates

{

words.add(word);

}

}

}

catch (IOException e) {

System.out.println("Error loading common words: " + e.getMessage());

}

return words;

}

/\*\*

\* Saves the provided common words list to the specified file

\* @param words list of common words

\* @param filename File to save common words to

\*/

public void saveCommonWords(List<String> words, String filename)

{

try (PrintWriter writer = new PrintWriter(new FileWriter(filename)))

/\*instantiates object PrintWriter for writing to the files\*/

{

for (String word : words) {

writer.println(word);

}

}

catch (IOException e) {

System.out.println("Error saving common words: " + e.getMessage());

}

}

/\*\*

\* Initializes the password log file writer.

\* Opens the log file for appending password records.

\*/

public void initPasswordLog()

{

try {

logWriter = new PrintWriter(new FileWriter(passwordLogFile, true)); // append mode

}

catch (IOException e) {

System.out.println("Error initializing password log: " + e.getMessage());

}

}

/\*\*

\* Saves a generated password and its strength rating to the password log file.

\* Format: [password] - [strength]

\* Example: dragon123! - Strong

\* @param password The password submitted by the user.

\* @param strength The evaluated strength rating (Weak/Moderate/Strong).

\*/

public void savePassword(String password, String strength)

{

if (logWriter != null)

{

logWriter.println(password + " - " + strength);

logWriter.flush();

}

else {

System.out.println("Log writer not initialized.");

}

}

//closes the password log writer

public void closeLog()

{

if (logWriter != null){

logWriter.close();

}

}

/\*\*

\* Adds a new phrase to the phrase file if it's valid and not already present

\* @param phrase The phrase to add

\* @return True if the phrase was added successfully, false otherwise

\*/

public boolean addPhrase(String phrase)

{

if (phrase == null || phrase.trim().isEmpty()) {

return false;

}

List<String> phrases = loadPhrases();

if (phrases.contains(phrase.trim())) {

return false; //already exists

}

phrases.add(phrase.trim());

savePhrases(phrases, phrasesFile);

/\*saves phases to phrases.txt\*/

return true;

}

}

### PasswordValidatorTest.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordValidatorTest.java

Author(s): Lizbeth Garcia-Lopez & Jordan Bassett

Last Updated: March 16, 2025

Description:

Early test class for validating the PasswordValidator logic.

Runs a set of predefined passwords through the validator and prints

their strength ratings and feedback to the console.

Features:

- Tests PasswordValidator against a variety of sample passwords

- Displays strength ratings and improvement feedback

- Demonstrates different difficulty settings

Dependencies:

- Uses: PasswordValidator.java, PasswordFileManager.java

\*/

package passwordfactory;

//Early console-based test class for validating PasswordValidator.

//outputs strength evaluations and feedback for predefined passwords.

public class PasswordValidatorTest

{

public static void main(String[] args)

{

//Create file manager for test environment

PasswordFileManager fileManager = new PasswordFileManager("phrases.txt", "password\_log.txt");

/\*This class ‘has-a’ relationship to FileManager to manage file operations\*/

/\*This class instantiates PasswordFileManager for handling file operations\*/

/\*This test class doesn’t directly save data to disk. It uses PasswordFileManager to interact with the file system for managing phrases and logs, but it doesn't save or load anything itself\*/

//Create validator using file manager and set difficulty (easy/medium/hard)

String difficulty = "medium"; // Change this to test other difficulties

PasswordValidator validator = new PasswordValidator(fileManager, difficulty);

/\*This class ‘uses’ PasswordValidator to validate passwords \*/

/\*This class instantiates PasswordValidator for validating passwords\*/

// Array of test passwords to evaluate

String[] testPasswords = {

"123456", // Weak

"Password123!", // Moderate (common word)

"HelloWorld!", // Moderate

"Str0ngP@ss!", // Strong

"SupermanLives!1", // Moderate (common word!)

"qwerty1234" // Weak (common word)

};

System.out.println("Testing Passwords in '" + difficulty + "' Mode");

System.out.println("==========================================");

//loop through each password and print evaluation results

for (String pw : testPasswords)

{

System.out.println("Testing: " + pw);

int score = validator.checkStrength(pw);

String feedback = validator.giveFeedback(pw);

String strengthLabel;

switch (score)

{

case 2:

strengthLabel = "Strong";

break;

case 1:

strengthLabel = "Moderate";

break;

default:

strengthLabel = "Weak";

break;

}

System.out.println("Strength: " + strengthLabel + " (Score: " + score + ")");

System.out.println("Feedback:\n" + feedback);

System.out.println("------------------------------------------");

}

}

}

### Hint.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: Hint.java

Author(s): Lizbeth Garcia-Lopez & Jordan Bassett

Last Updated: March 16, 2025

Description:

A simple Plain Old Java Object (POJO) to hold hint data

Hint represents a suggestion or piece of feedback about a password

Features:

- Stores a feedback message and its category

- Provides methods to access hint data and display it as a string.

Dependencies:

- Used by HintManager.java to manage and display hints. \*/

package passwordfactory;

//simple POJO (Plain Old Java Object) to hold hint data.

public class Hint

/\*Hint class stores hint messages and their associated categories. It is ‘used by’ the HintManager class to manage and provide hints for password strengthening\*/

{

//Message that explains the hint or suggestion

private String message;

//category for the hint (e.g., "Length", "Complexity")

private String category;

/\*\*

\* Constructor for Hint.

\* @param message The feedback message.

\* @param category The category or type of hint.

\*/

public Hint(String message, String category)

{

this.message = message;

this.category = category;

}

/\*\*

\* Gets the hint message.

\* @return The feedback message.

\*/

public String getHint() {

return message;

}

/\*\*

\* Gets the hint category.

\* @return The category of the hint.

\*/

public String getCategory() {

return category;

}

/\*\*

\* Returns a string of the hint.

\* @return The category and message combined.

\*/

@Override

public String toString() {

return "[" + category + "] " + message;

}

}

### HintManager.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: HintManager.java

Author(s): Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

Manages both dynamic and generic hints that provide feedback during password creation

Dynamic hints are reset every round, while generic hints are always available

Features:

- Adds and retrieves dynamic hints during gameplay.

- Provides a list of generic hints that offer basic password advice.

- Clears dynamic hints between rounds.

- Can return hints as formatted strings for display.

Dependencies:

- Uses: Hint.java

- Called by: PasswordValidator.java and other password evaluation components. \*/

package passwordfactory;

import java.util.ArrayList;

import java.util.List;

//ArrayLists, as we add hints sequentially, and don’t need to search/remove specific ones often

public class HintManager

{

private final List<Hint> dynamicHints;

private final List<Hint> genericHints;

/\*HintManager **‘has-a’** list of Hint objects to store both dynamic and generic hints for password strength feedback\*/

// Constructor for HintManager.

public HintManager()

{

dynamicHints = new ArrayList<>();

genericHints = new ArrayList<>();

// Add generic hints at construction

genericHints.add(new Hint("Try making it longer.", "Length"));

genericHints.add(new Hint("Add an uppercase letter.", "Uppercase"));

genericHints.add(new Hint("Add a lowercase letter.", "Lowercase"));

genericHints.add(new Hint("Add a number.", "Numbers"));

genericHints.add(new Hint("Add a special character.", "Special Characters"));

}

/\*\*

\* Adds a dynamic hint during game

\* @param hint A hint generated based on the user's input.

\*/

public void addDynamicHint(Hint hint) {

dynamicHints.add(hint);

}

/\*\*

\* Returns the current list of dynamic hints

\* @return List of dynamic hints.

\*/

public List<Hint> getDynamicHints() {

return dynamicHints;

}

/\*\*

\* Returns the list of generic hints

\* @return List of generic hints

\*/

public List<Hint> getGenericHints() {

return genericHints;

}

//clears all dynamic hints between rounds.

public void clearDynamicHints() {

dynamicHints.clear();

}

/\*\*

\* Returns the current dynamic hints as formatted strings

\* @return List of dynamic hint messages.

\*/

public List<String> getDynamicHintMessages()

{

List<String> messages = new ArrayList<>();

for (Hint hint : dynamicHints) {

messages.add(hint.toString());

}

return messages;

}

/\*\*

\* Returns all generic hints as formatted strings

\* @return List of generic hint messages

\*/

public List<String> getGenericHintMessages()

{

List<String> messages = new ArrayList<>();

for (Hint hint : genericHints) {

messages.add(hint.toString());

}

return messages;

}

}

### PasswordFactoryConsole.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordFactoryConsole.java

Author(s): Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

Early development console application for testing password generation and validation

Allows users to select phrases, numbers, and special characters to assemble a password

Evaluates password strength using PasswordValidator and saves the result to a file

Runs on its own

Features:

- Console-based user input for selecting components of a password.

- Assembles a password from phrases, numbers, and symbols.

- Validates the password using PasswordValidator.

- Saves the password and its strength evaluation to a file (pwd\_output.txt).

Dependencies:

- Uses: PasswordValidator.java

- Writes output to: pwd\_output.txt \*/

package passwordfactory;

import java.io.FileWriter;

import java.io.IOException;

import java.io.PrintWriter;

import java.util.Scanner;

public class PasswordFactoryConsole

{

//Sample phrase options for the console version

private static final String[] PHRASES = {"sunshine", "rainbow", "monkey", "dragon", "coffee"};

//Special characters that can be added to the password

private static final String[] SPECIAL\_CHARS = {"!", "@", "#", "$", "%", "&", "\*"};

/\*\*

\* Main method for running the console application

\* Guides the user through creating a password and validates it

\* @param args Command-line arguments (not used)

\*/

public static void main(String[] args)

{

Scanner scanner = new Scanner(System.in);

StringBuilder passwordBuilder = new StringBuilder();

System.out.println("Welcome to Password Factory Console Edition!");

//Phrase selection

System.out.println("Choose a phrase:");

for (int i = 0; i < PHRASES.length; i++)

{

System.out.println((i + 1) + ". " + PHRASES[i]);

}

int phraseChoice = scanner.nextInt();

scanner.nextLine(); // consume newline

if (phraseChoice >= 1 && phraseChoice <= PHRASES.length)

{

passwordBuilder.append(PHRASES[phraseChoice - 1]);

}

else

{

System.out.println("Invalid choice. Exiting...");

return;

}

// Number input

System.out.println("Enter a number to add:");

String number = scanner.nextLine();

passwordBuilder.append(number);

//special character selection

System.out.println("Choose a special character:");

for (int i = 0; i < SPECIAL\_CHARS.length; i++)

{

System.out.println((i + 1) + ". " + SPECIAL\_CHARS[i]);

}

int charChoice = scanner.nextInt();

if (charChoice >= 1 && charChoice <= SPECIAL\_CHARS.length)

{

passwordBuilder.append(SPECIAL\_CHARS[charChoice - 1]);

}

else

{

System.out.println("Invalid choice. Exiting...");

return;

}

String finalPassword = passwordBuilder.toString();

//Validate the password

PasswordFileManager fileManager = new PasswordFileManager("phrases.txt", "password\_log.txt");

PasswordValidator validator = new PasswordValidator(fileManager, "easy");

/\*this instantiates PasswordFileManager to manage file operations and PasswordValidator to validate the strength of generated passwords\*/

int strength = validator.checkStrength(finalPassword);

String strengthLabel = (strength == 2) ? "Strong" : (strength == 1) ? "Moderate" : "Weak";

// Display result

System.out.println("Generated Password: " + finalPassword);

System.out.println("Password Strength: " + strengthLabel);

// Save the password and strength to a file

try (PrintWriter writer = new PrintWriter(new FileWriter("pwd\_output.txt", true)))

/\*writes the password and its strength to pwd\_output \*/

{

writer.println(finalPassword + " - " + strengthLabel);

System.out.println("Password saved to pwd\_output.txt");

}

catch (IOException e)

{

System.out.println("Error saving password to file: " + e.getMessage());

}

scanner.close();

}

}

## Classes Containing GUI Components

### App.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: App.java

Author(s): Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

This is the main entry point for the Password Factory game. It launches the JavaFX application,

initializes the main menu, and handles navigation to the Password Game, Password Lab, and About Screen.

Features:

- Displays the main menu with buttons for Play Game, Password Lab, and Exit

- Opens the GameSetupScreen for selecting difficulty

- Opens PasswordLab for password testing and file editing

- Opens AboutScreen to show system information

Dependencies:

- Calls: GameSetupScreen.java, PasswordLab.java, AboutScreen.java

- Launches the Password Factory application with JavaFX Application class \*/

package passwordfactory;

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.layout.\*;

import javafx.stage.Stage;

//Initializes and displays the main menu GUI.

public class App extends Application

{

//@param primaryStage The primary stage (window) for the program.

//@param primaryStage

@Override

public void start(Stage primaryStage)

{

//Title for main menu

Label titleLabel = new Label("Password Factory");

//Main menu buttons

Button playGameButton = new Button("Play Game");

Button passwordLabButton = new Button("Password Lab");

Button exitButton = new Button("Exit");

/\*buttons for playing the game, opening the password lab, and displaying the about screen

//play Game Button Action

playGameButton.setOnAction(e ->

/\* Switch to game screen \*/

{

System.out.println("Play Game button clicked"); //Debugging output

GameSetupScreen setupScreen = new GameSetupScreen(primaryStage);

setupScreen.show();

});

//Password Lab Button Action

passwordLabButton.setOnAction(e ->

/\* Switch to game screen \*

{

System.out.println("Password Lab button clicked"); //Debugging output

// Launch Password Lab

PasswordLab passwordLab = new PasswordLab();

/\*Instantiates a PasswordLab object to manage the password lab interface\*/

passwordLab.showLab();

// Close the main menu

primaryStage.close();

});

//Exit Button Action

exitButton.setOnAction(e ->

{

System.out.println("Exiting application..."); //debugging output

primaryStage.close();

});

//VBox for main menu buttons (Centered vertically)

VBox mainButtons = new VBox(15, titleLabel, playGameButton, passwordLabButton, exitButton);

mainButtons.setStyle("-fx-padding: 20px; -fx-alignment: center;");

//Small "About" button in the bottom-right corner

Button aboutButton = new Button("?");

aboutButton.setStyle("-fx-font-size: 10px; -fx-padding: 5px; -fx-background-radius: 50%;");

//About Button Action

aboutButton.setOnAction(e -> new AboutScreen().show());

//container to align the about button to the bottom-right

HBox aboutContainer = new HBox(aboutButton);

aboutContainer.setStyle("-fx-alignment: bottom-right; -fx-padding: 10px;");

//BorderPane layout for the main menu

BorderPane root = new BorderPane();

root.setCenter(mainButtons); // Main buttons in center

root.setBottom(aboutContainer); // About button in bottom-right

//Scene and stage setup

Scene scene = new Scene(root, 400, 300);

primaryStage.setTitle("Password Factory");

primaryStage.setScene(scene);

primaryStage.show();

}

//Main method to launch the JavaFX application.

//@param args Command-line arguments (not used).

public static void main(String[] args) {

launch(args); //starts the JavaFX app

}

}

### GameSetupScreen.java

/\* Lizbeth Garcia-Lopez & Jordan Bassett

Password Factory

CS236 - Advanced OOP (Java 2)

Final Project: Strong Password Creation Game for Kids

Updated March 16 2025

Desc: This class creates the game setup screen where players select a difficulty level

before starting the password challenge. It provides instructions and handles transitions

between the main menu and the actual game screen.

Features:

- Displays game instructions.

- Allows players to choose a difficulty level (Easy, Medium, Hard)

- Starts the game when the player clicks "Start Game"

- Provides a "Back to Main Menu" button to return to App.java

Navigation:

- Called from: App.java (when "Play Game" is clicked)

- Calls: PasswordGame.java (when "Start Game" is clicked) \*/

package passwordfactory;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.VBox;

import javafx.scene.text.Text;

import javafx.scene.text.TextFlow;

import javafx.stage.Modality;

import javafx.stage.Stage;

public class GameSetupScreen

{

private final Stage mainMenuStage; // stores reference to the main menu stage to close it

/\*\*

\* Constructor for GameSetupScreen

\* @param mainMenuStage Reference to the main menu stage so it can be closed when this screen opens

\*/

public GameSetupScreen(Stage mainMenuStage) {

this.mainMenuStage = mainMenuStage;

}

//Displays the setup screen where players select their difficulty level

public void show()

{

// Close the main menu when opening this setup screen

if (mainMenuStage != null) {

mainMenuStage.close();

}

// Create a new window/stage for the setup screen

Stage setupStage = new Stage();

//Prevent user from interacting with the main menu while the setup screen is open

setupStage.initModality(Modality.APPLICATION\_MODAL);

setupStage.setTitle("Game Setup");

// Debugging output

//System.out.println("Setup screen should be visible now");

// Instructions using TextFlow

TextFlow instructionsFlow = new TextFlow();

Text header = new Text("Welcome to Password Factory!\n\n");

header.setStyle("-fx-font-weight: bold; -fx-font-size: 16;");

Text line1 = new Text("• Enter passwords as fast as you can before time runs out.\n");

Text line2 = new Text("• Watch for hints to improve your password strength.\n");

Text line3 = new Text("• Earn points for stronger passwords. Weak ones won't score!\n");

Text line4 = new Text("• Avoid common words or risk losing points!\n\n");

Text difficultyHeader = new Text("Difficulty Levels:\n");

difficultyHeader.setStyle("-fx-font-weight: bold;");

Text easyDesc = new Text("Easy: More time. Basic password rules.\n");

Text mediumDesc = new Text("Medium: Less time. Common words are penalized harder.\n");

Text hardDesc = new Text("Hard: Tight time. Only complex passwords score!\n");

instructionsFlow.getChildren().addAll(

header, line1, line2, line3, line4,

difficultyHeader, easyDesc, mediumDesc, hardDesc

);

//Difficulty selection using radio buttons

ToggleGroup difficultyGroup = new ToggleGroup();

RadioButton easyButton = new RadioButton("Easy");

RadioButton mediumButton = new RadioButton("Medium");

RadioButton hardButton = new RadioButton("Hard");

/\* Radio buttons for choosing the difficulty level (Easy, Medium, Hard)\*/

easyButton.setToggleGroup(difficultyGroup);

mediumButton.setToggleGroup(difficultyGroup);

hardButton.setToggleGroup(difficultyGroup);

easyButton.setSelected(true); // Default selection is Easy

//Start Game button

Button startButton = new Button("Start Game");

startButton.setOnAction(e -> {

/\*Starts the game\*/

//find the selected difficulty

String selectedDifficulty = ((RadioButton) difficultyGroup.getSelectedToggle()).getText();

//close the setup window

setupStage.close();

//open the PasswordGame screen and send it selected difficulty

new PasswordGame(selectedDifficulty).show();

});

//Back to Main Menu button

Button backButton = new Button("Back to Main Menu");

backButton.setOnAction(e ->

//Back to Main Menu button

{

setupStage.close();

Stage mainStage = new Stage();

new App().start(mainStage);

});

//Layout: VBox with spacing and padding

VBox layout = new VBox(10,

instructionsFlow,

easyButton,

mediumButton,

hardButton,

startButton,

backButton

);

layout.setStyle("-fx-padding: 20px; -fx-alignment: center;");

//assign the scene to the stage and show the setup screen

setupStage.setScene(new Scene(layout, 500, 500));

setupStage.show();

}

}

### AboutScreen.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: AboutScreen.java

Author(s): Lizbeth Garcia-Lopez & Jordan Bassett

Last Updated: March 16, 2025

Description:

This class creates a simple "About" screen that provides information about the game

and displays system details, such as Java and JavaFX versions.

Features:

- Displays game title, version, and credits.

- Uses SystemInfo.java to retrieve Java and JavaFX version details.

- Provides a "Close" button to dismiss the window.

Dependencies:

- Called from: App.java (when the "About" button is clicked).

- Uses: SystemInfo.java (provided by Netbeans) \*/

package passwordfactory;

import javafx.scene.Scene;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.layout.VBox;

import javafx.stage.Modality;

import javafx.stage.Stage;

//Displays the About window with project and system information

public class AboutScreen

{

public void show()

{

//create a new Stage for the About window

Stage aboutStage = new Stage();

//Block interaction with other windows until closed

//entire application is locked until this stage closes

aboutStage.initModality(Modality.APPLICATION\_MODAL);

aboutStage.setTitle("About Password Factory");

//Labels for project info

Label projectLabel = new Label("Password Factory");

Label courseLabel = new Label("CS236 - Advanced OOP");

Label collegeLabel = new Label("Columbia Basin College");

//created By section

Label createdByLabel = new Label("Created By:");

Label authorsLabel = new Label("Lizbeth Garcia-Lopez & Jordan Bassett");

//System information (Java & JavaFX versions)

Label javaVersion = new Label("Java Version: " + SystemInfo.javaVersion());

Label javafxVersion = new Label("JavaFX Version: " + SystemInfo.javafxVersion());

/\*these are labels for displaying text\*/

//Close button to close the About window

Button closeButton = new Button("Close");

closeButton.setOnAction(e -> aboutStage.close());

//Close button to close the About window. That’s why we said so two lines ago

//Layout setup for the About window

VBox layout = new VBox(10,

projectLabel,

courseLabel,

collegeLabel,

createdByLabel,

authorsLabel,

javaVersion,

javafxVersion,

closeButton

);

layout.setStyle("-fx-padding: 20px; -fx-alignment: center;");

// Set the scene and display the About window

aboutStage.setScene(new Scene(layout, 400, 250));

aboutStage.show();

}

}

### PasswordLab.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: PasswordLab.java

Author(s): Lizbeth Garcia-Lopez & Jordan Bassett

Last Updated: March 16, 2025

Description:

Password Lab is the sandbox mode for Password Factory. It allows users to test

passwords and edit helpful phrases and common words. The Lab provides feedback

on password strength in real-time and saves user edits to the corresponding text files

Features:

- Tests passwords and provides feedback using PasswordValidator

- Allows adding, removing, and saving common words

- Allows adding, removing, and saving helpful phrases

- Provides an interface to return to the main menu

Dependencies:

- Uses: PasswordValidator.java, PasswordFileManager.java, App.java

\*/

package passwordfactory;

import javafx.geometry.Insets;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.layout.\*;

import javafx.stage.Stage;

import java.util.List;

import java.util.Random;

public class PasswordLab

{

private Stage labStage; //window for Password Lab

// Core Components

private final PasswordFileManager fileManager; //file operations (phrases, common words, logs)

private PasswordValidator validator; //Validates passwords and provides feedback

//UI Components - Lists

private ListView<String> commonWordsList; //List of common words (editable)

private ListView<String> phrasesList; //List of helpful phrases (editable)

//UI Components - Password Testing

private TextField passwordTestField; //Field for entering passwords to test

private Label testResultLabel; //shows the strength rating of the tested password

private Label feedbackLabel; //shows detailed feedback and suggestions

private Button generatePasswordButton; //Button to generate a secret password

/\*These have comments\*/

/\*\*

\* Constructor for PasswordLab.

\* Initializes the validator and file manager.

\*/

public PasswordLab()

{

//Initialize file manager and validator (Easy mode for testing)

fileManager = new PasswordFileManager("phrases.txt", "password\_log.bin");

validator = new PasswordValidator(fileManager, "easy");

}

/\*\*

\* Displays the Password Lab window.

\* Provides UI for testing passwords and editing common words and helpful phrases.

\*/

public void showLab()

{

labStage = new Stage();

labStage.setTitle("Password Lab");

// Password Tester Section

Label testHeader = new Label("Test a Password");

passwordTestField = new TextField();

passwordTestField.setPromptText("Enter a password to test...");

passwordTestField.setPrefWidth(200); // Set width for password field

Button testButton = new Button("Test Password");

testButton.setOnAction(e -> testPassword());

testResultLabel = new Label();

feedbackLabel = new Label();

feedbackLabel.setWrapText(true);

feedbackLabel.setMaxWidth(600);

/\*these are labels for displaying text, with options for width and wrapping\*/

// Box for testing passwords with a fixed width

VBox testBox = new VBox(10,

testHeader,

passwordTestField,

testButton,

testResultLabel,

feedbackLabel);

testBox.setPadding(new Insets(10));

testBox.setStyle("-fx-border-color: #aaa; -fx-border-radius: 5; -fx-padding: 10;");

//finding a maximum width for the testBox to prevent it from stretching too far

testBox.setMaxWidth(350); //adjustable

testBox.setPrefWidth(300); //Preferred width for the test box

// Common Words Section

Label commonWordsHeader = new Label("Common Words");

commonWordsList = new ListView<>();

commonWordsList.getItems().addAll(fileManager.loadCommonWords("common\_words.txt"));

/\*displays and allows users to add/remove phrases\*/

TextField addCommonWordField = new TextField();

addCommonWordField.setPromptText("Add common word...");

Button addCommonWordButton = new Button("Add");

addCommonWordButton.setOnAction(e -> {

String word = addCommonWordField.getText().trim().toLowerCase();

if (!word.isEmpty() && !commonWordsList.getItems().contains(word))

{

commonWordsList.getItems().add(word);

addCommonWordField.clear();

}

});

Button removeCommonWordButton = new Button("Remove Selected");

removeCommonWordButton.setOnAction(e -> {

String selected = commonWordsList.getSelectionModel().getSelectedItem();

if (selected != null) {

commonWordsList.getItems().remove(selected);

}

});

Button saveCommonWordsButton = new Button("Save Common Words");

saveCommonWordsButton.setOnAction(e -> saveCommonWords());

VBox commonWordsBox = new VBox(10,

commonWordsHeader,

commonWordsList,

addCommonWordField,

addCommonWordButton,

removeCommonWordButton,

saveCommonWordsButton);

commonWordsBox.setPadding(new Insets(10));

commonWordsBox.setStyle("-fx-border-color: #aaa; -fx-border-radius: 5; -fx-padding: 10;");

commonWordsBox.setPrefWidth(300);

// Helpful Phrases Section

Label phrasesHeader = new Label("Helpful Phrases");

phrasesList = new ListView<>();

phrasesList.getItems().addAll(fileManager.loadPhrases());

TextField addPhraseField = new TextField();

addPhraseField.setPromptText("Add helpful phrase...");

Button addPhraseButton = new Button("Add");

addPhraseButton.setOnAction(e -> {

String phrase = addPhraseField.getText().trim();

if (!phrase.isEmpty() && !phrasesList.getItems().contains(phrase))

{

phrasesList.getItems().add(phrase);

addPhraseField.clear();

}

});

Button removePhraseButton = new Button("Remove Selected");

removePhraseButton.setOnAction(e -> {

String selected = phrasesList.getSelectionModel().getSelectedItem();

if (selected != null)

{

phrasesList.getItems().remove(selected);

}

});

Button savePhrasesButton = new Button("Save Phrases");

savePhrasesButton.setOnAction(e -> savePhrases());

VBox phrasesBox = new VBox(10,

phrasesHeader,

phrasesList,

addPhraseField,

addPhraseButton,

removePhraseButton,

savePhrasesButton);

phrasesBox.setPadding(new Insets(10));

phrasesBox.setStyle("-fx-border-color: #aaa; -fx-border-radius: 5; -fx-padding: 10;");

phrasesBox.setPrefWidth(300);

// Back Button to return to main menu

Button backButton = new Button("Back to Menu");

backButton.setOnAction(e -> {

labStage.close();

new App().start(new Stage());

});

// Layout

HBox mainLayout = new HBox(20,

commonWordsBox,

testBox,

phrasesBox);

VBox rootLayout = new VBox(10,

mainLayout,

backButton);

rootLayout.setPadding(new Insets(10));

rootLayout.setStyle("-fx-alignment: center;");

Scene labScene = new Scene(rootLayout, 600, 600);

labStage.setScene(labScene);

labStage.setResizable(false);

labStage.show();

// Add the "Generate Password" Button

generatePasswordButton = new Button("Get a secret password");

generatePasswordButton.setOnAction(e -> generatePassword());

// Layout Adjustment for Button

VBox passwordBox = new VBox(10, generatePasswordButton, testBox);

rootLayout.getChildren().add(passwordBox);

}

// Automatically generate a password by selecting at least 3 phrases

private void generatePassword()

{

List<String> phrases = fileManager.loadPhrases();

// Select 3 random phrases

String phrase1 = phrases.get(new Random().nextInt(phrases.size()));

String phrase2 = phrases.get(new Random().nextInt(phrases.size()));

String phrase3 = phrases.get(new Random().nextInt(phrases.size()));

// Remove spaces from phrases and take the first word

String word1 = stripSpaces(phrase1.split(" ")[0]); // First word, spaces removed

String word2 = stripSpaces(phrase2.split(" ")[0]); // First word, spaces removed

String word3 = stripSpaces(phrase3.split(" ")[0]); // First word, spaces removed

// Combine them into a password and add some extra characters for complexity

String password = word1 + word2 + word3 + generateRandomCharacters();

// Ensure the password meets all requirements

password = ensurePasswordRules(password);

// Display the generated password

testResultLabel.setText("Generated Password: " + password);

// Set the generated password in the "Test a Password" field so user can test it immediately

passwordTestField.setText(password);

}

//Removes spaces from the given string

private String stripSpaces(String phrase) {

return phrase.replaceAll("\\s", ""); // Removes all spaces from the string

}

//generates random characters to add complexity to the password

private String generateRandomCharacters() {

Random rand = new Random();

String chars = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz!@#$%^&\*";

StringBuilder randomChars = new StringBuilder();

// Add 2 random characters to the password

for (int i = 0; i < 2; i++) {

randomChars.append(chars.charAt(rand.nextInt(chars.length())));

}

return randomChars.toString();

}

// Ensures the password meets the required password rules

private String ensurePasswordRules(String password) {

// Make sure we have an uppercase, lowercase, number, and special character

if (!password.matches(".\*[A-Z].\*")) {

password += "A"; // Add an uppercase letter if missing

}

if (!password.matches(".\*[a-z].\*")) {

password += "b"; // Add a lowercase letter if missing

}

if (!password.matches(".\*[0-9].\*")) {

password += "1"; // Add a number if missing

}

if (!password.matches(".\*[!@#$%^&\*].\*")) {

password += "!"; // Add a special character if missing

}

// Ensures the password meets the required password rules

// Make sure we have an uppercase, lowercase, number, and special character

return password;

}

// Tests the entered password for strength and feedback

private void testPassword()

{

String password = passwordTestField.getText().trim();

if (password.isEmpty())

{

testResultLabel.setText("Enter a password to test.");

feedbackLabel.setText("");

return;

}

int strength = validator.checkStrength(password);

String feedback = validator.giveFeedback(password);

String strengthLabel;

switch (strength)

{

case 2: strengthLabel = "Strong"; break;

case 1: strengthLabel = "Moderate"; break;

default: strengthLabel = "Weak"; break;

}

testResultLabel.setText("Strength: " + strengthLabel);

feedbackLabel.setText(feedback);

}

// Saves edits to the common words list back to common\_words.txt.

// Refreshes the validator to use the updated common words.

private void saveCommonWords()

{

List<String> words = commonWordsList.getItems();

fileManager.saveCommonWords(words, "common\_words.txt");

// Saves edits to the common words list back to common\_words.txt.

// Refresh validator with updated common words

validator = new PasswordValidator(fileManager, "easy");

testResultLabel.setText("Common words saved.");

}

// Saves edits to the helpful phrases list back to phrases.txt.

private void savePhrases()

{

List<String> phrases = phrasesList.getItems();

fileManager.savePhrases(phrases, "phrases.txt");

testResultLabel.setText("Phrases saved.");

}

// Saves edits to the helpful phrases list back to phrases.txt.

}

### SystemInfo.java

/\* Password Factory Project

CS236 - Advanced OOP (Java 2)

Class: SystemInfo.java

Author(s): Provided by NetBeans | Integrated by Jordan Bassett & Lizbeth Garcia-Lopez

Last Updated: March 16, 2025

Description:

Utility class to retrieve system properties for Java and JavaFX versions.

Used by AboutScreen to display system information.

Features:

- Retrieves the Java runtime version.

- Retrieves the JavaFX runtime version.

Dependencies:

- Used by: AboutScreen.java

\*/

package passwordfactory;

public class SystemInfo

{

/\*\*

\* Returns the current Java version.

\* @return Java version as a String.

\*/

public static String javaVersion() {

return System.getProperty("java.version");

}

/\*\*

\* Returns the current JavaFX version.

\* @return JavaFX version as a String.

\*/

public static String javafxVersion() {

return System.getProperty("javafx.version");

/\*utility methods to retrieve the current version of Java and JavaFX running on the system\*/

}

}

## Main Methods

“After each class that contains a main() method, paste the main() method at the end of the class”  
the main() methods in PasswordValidatorTest.java, PasswordFactoryConsole.java. and App.java are as far to the end of the files as possible.

**UML**

“Explain if your UML class diagrams align with the implementations.”

Yes, the UML class diagrams do align with the implementations in the code, as we were instructed to do in previous phases. Throughout the development process, we continuously updated the UML to match the changes made to the program. UML diagrams have effectively served as a **reliable blueprint** that helped visualize the relationships between classes

**“files on disk- must be submitted”**

**phrases.txt:** This file contains helpful phrases that can be used to aid users in creating stronger passwords. It is a plain text file to allow easy editing and manipulation by users outside of the program or it may be edited in the Password Lab.

**common\_words.txt**: This file stores common words that are considered weak for password creation, as researched online. This file is stored as a plain text file fore ease of modification by users and also can be edited in the password lab.

**password\_log.bin**: This file logs all generated passwords and their strength ratings. It is stored as a binary file using random access memory. Binary format was used to help protect the security of the stored data.

**Provide two screenshots of outputs that show results of a game-play.**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Provide two screenshots of outputs that show results of password created.**

A screenshot of a computer

AI-generated content may be incorrect.

