5/24/2023

# Assignment 4

Implement the next release of your term project. You will incorporate *saving and retrieving objects*, *lambdas*, and *streams*. You can substitute the use of JavaFX for one of these three concepts if you wish. Also, substitutions for any of these concepts by another advanced techniques are acceptable if you have already used them: please clear this with your facilitator.

The same instructions as in Assignment 3 apply to this completed Word document regarding the gray text, the 5 page limit, use of AI generation, appendices, JUnit tests, and a ReadMe file.

## 1 SUMMARY DESCRIPTION

One- or two-paragraph overall description of your proposed term project. Color red the parts changed from previous assignments, if any.

The Personal Health Information (PHI) app is a comprehensive solution designed to assist users in monitoring and managing their personal health. It allows users to create an account, securely log in, and track various health metrics such as BMI, blood pressure, and cholesterol levels. The app enables users to input and store their health data, providing a user-friendly interface for seamless data entry and updates. The app provides a user-friendly interface for adding and managing health data, making it easy for users to enter and update their information. It also offers the option to create custom health entries, where users can record specific health issues or concerns and add notes to track their progress.

With a strong emphasis on privacy and data security, the app ensures that each user has full control over their health information. Users can choose to keep their data private or share it with healthcare professionals or family members as needed. The app has the potential to not only prevent unwanted future health complications, but also act as a motivator to keep the health in track. Overall, the Health Tracker app provides a convenient and centralized platform for users to monitor and maintain their personal health records, empowering them to make informed decisions about their well-being.

## 2 ADDITIONAL REQUIREMENTS (FEATURES) IMPLEMENTED IN THIS RELEASE

Title and one or two sentences per requirement. Don’t repeat requirements implemented for prior assignments unless they are necessary to provide context—in which case, make it clear they are old.

### 2.1 Adding User (old from week 1)

The application shall allow users to register/login with their personal information, username, email, and password.

### 2.2 Filter health data by date (NEW)

The app shall be able to filter healthdata by date of record

### 2.3 Calculate average healthmetric (NEW)

The app shall be able calculate average healthmetric over time

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

I used one old requirement from week 1 as it is related to javafx. For the second one, I improved my prompt to help chatgpt to understand what a functional requirement is. The response was more accurate but not quite there. So I modified and simplified the response that follows the functional requirement rule and standard to best of my ability.

1. MY PROMPT SEQUENCE (ONLY)

[1]

This week I will implement JavaFX, streams, and lambdas in this iteration. Give me one functional requirement (2 sentences max) about this weeks iteration for the PHI project.

[2] A functional requirement should focus on what the app shall be able to do, not how it would do it, keeping that in mind, could you provide me a one sentence requirement that will be related to this week’s iteration topics?

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

[1] Functional Requirement: Implement a user interface in the PHI project that allows users to view their health data using JavaFX, streams, and lambdas, providing a streamlined and efficient way to access and analyze their personal health information.

[2] Functional Requirement: The PHI application shall provide users with the ability to filter and sort their health data based on various criteria using JavaFX, streams, and lambdas.

## 3 I/O SUPPORTING THE NEW REQUIREMENTS LISTED ABOVE

Provide examples of input / output generated by your implementation, showing the new features of your application.

### Input 1

User Login screen:  
A screenshot of a computer

Description automatically generated

### Output for 1

User created successfully:  
A screenshot of a computer

Description automatically generated

### Input 2

Health data entry screen:  
A screenshot of a computer

Description automatically generated

An example of data entry:  
A screenshot of a computer

Description automatically generated

### Output for 2

After inputting the values, the history page looks like this:  
The average values are calculated using lambda and stream functionalities.   
A screenshot of a computer

Description automatically generated

Here is the date filter working as expected. It is supposed to show data entry for the date selected.  
A screenshot of a computer

Description automatically generated

## 4 YOUR DIRECTORY

Show a screenshot of your directory. Include your “.dat” files (where objects are written). This should include JUnit tests—class-by-class, and method-by-method, except for trivial and inappropriate ones.

A picture containing text, screenshot, document, font

Description automatically generated

## 5 DESIGN

Supply a main use case, the class model, and the sequence diagram corresponding to the use case. These should be consistent. Indicate in red your class model where you applied object read, object write, streams and lambdas. Excellent assignments will typically include the use of Java FX (speak to your facilitator first if you wish to use alternative API’s) and event-driven programming.

Class uml:  
A screenshot of a computer

Description automatically generated with low confidence

Sequence Diagram:  
A picture containing text, diagram, parallel, plan

Description automatically generated

I used previous week’s plantuml text and added/edited the text to incorporate the new two classes HealthDataEntry, HistroryScreen and updated Main class.

### >>AI generation OR check: *I did not use AI generation here* \_NO\_(Collapse section before submitting.)

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

I improved the intent of the summary with a few edits. chatGPT’s output was too much like a marketing piece.

1. MY PROMPT SEQUENCE (ONLY)

[1]

Improve the following summary: In this strategy game, as Earth's Chief Scientist Alex, employed by the UN, you must save the Earth by managing funds and powers. You interact with governments, and through them, the public and companies, each with their own revenues, abilities to tax or be taxed, and cause or repair environmental damage. Before it is too late, you must halt the degradation of the Earth from CO2 emission and achieve global sustainability.

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

Embark on a thrilling strategy game as Alex, Earth's Chief Scientist employed by the United Nations, tasked with saving the planet by adeptly managing resources and influence. Engage with governments, the public, and corporations, each possessing unique revenue streams, taxation capabilities, and potential to either inflict or mitigate environmental damage. Race against time to curb Earth's degradation from CO2 emissions and secure global sustainability before it's too late.

## 6 CODE SNIPPETS

### 6.1 Code showing JavaFX

### Note: Only showing the code portion (without pre&post condition due to space issue. The main code should have all the pre-post conditions.

In the Main class:   
**private** **void** createLoginScene() {

// Create UI components for login scene

Label usernameLabel = **new** Label("Username:");

TextField usernameTextField = **new** TextField();

Label passwordLabel = **new** Label("Password:");

PasswordField passwordTextField = **new** PasswordField();

Button loginButton = **new** Button("Login");

Button createUserButton = **new** Button("Create User");

// Create layout container for login scene

GridPane root = **new** GridPane();

root.setAlignment(Pos.***CENTER***);

root.setHgap(10);

root.setVgap(5);

root.setPadding(**new** Insets(10));

// Add UI components to the grid pane

root.add(usernameLabel, 0, 0);

root.add(usernameTextField, 1, 0);

root.add(passwordLabel, 0, 1);

root.add(passwordTextField, 1, 1);

root.add(loginButton, 0, 2);

root.add(createUserButton, 1, 2);

// Create login scene

loginScene = **new** Scene(root, 400, 200);

// Handle create user button click event

createUserButton.setOnAction(event -> {

primaryStage.setScene(createUserScene);

});

// Handle login button click event (placeholder action)

loginButton.setOnAction(event -> {

// Placeholder action, you can add your own logic here

String username = usernameTextField.getText();

String password = passwordTextField.getText();

System.***out***.println("Logging in with username: " + username + ", password: " + password);

});

}

**private** **void** createCreateUserScene() {

// Create UI components for create user scene

Label firstNameLabel = **new** Label("First Name:");

TextField firstNameTextField = **new** TextField();

Label lastNameLabel = **new** Label("Last Name:");

TextField lastNameTextField = **new** TextField();

Label emailLabel = **new** Label("Email:");

TextField emailTextField = **new** TextField();

Label passwordLabel = **new** Label("Password:");

PasswordField passwordTextField = **new** PasswordField();

Label dateOfBirthLabel = **new** Label("Date of Birth:");

DatePicker dateOfBirthPicker = **new** DatePicker();

Label genderLabel = **new** Label("Gender:");

TextField genderTextField = **new** TextField();

Label phoneNumberLabel = **new** Label("Phone Number:");

TextField phoneNumberTextField = **new** TextField();

Button createUserButton = **new** Button("Create User");

// Create layout container for create user scene

VBox root = **new** VBox(5);

root.setAlignment(Pos.***CENTER***);

root.setPadding(**new** Insets(10));

root.getChildren().addAll(

firstNameLabel, firstNameTextField,

lastNameLabel, lastNameTextField,

emailLabel, emailTextField,

passwordLabel, passwordTextField,

dateOfBirthLabel, dateOfBirthPicker,

genderLabel, genderTextField,

phoneNumberLabel, phoneNumberTextField,

createUserButton

);

// Create create user scene

createUserScene = **new** Scene(root, 500, 500);

// Handle create user button click event

createUserButton.setOnAction(event -> {

// Retrieve user input

String firstName = firstNameTextField.getText();

String lastName = lastNameTextField.getText();

String email = emailTextField.getText();

String password = passwordTextField.getText();

LocalDate dateOfBirth = dateOfBirthPicker.getValue();

String gender = genderTextField.getText();

String phoneNumber = phoneNumberTextField.getText();

**try** {

// Validate email and password inputs using User class methods

User<HealthData<?>> user = **new** User<>(firstName, lastName, email, password, convertToDate(dateOfBirth), gender, phoneNumber);

healthDataEntry = **new** HealthDataEntry(primaryStage, user);

// Perform any additional operations with the created user object

// For example, save the user to a database

// Show a success message or provide feedback to the user

// Example: display a dialog

Alert alert = **new** Alert(Alert.AlertType.***INFORMATION***);

alert.setTitle("User Creation");

alert.setHeaderText(**null**);

alert.setContentText("User created successfully!");

alert.showAndWait();

// Switch to the health data entry scene

healthDataEntry.showHealthDataEntryScene();

} **catch** (IllegalArgumentException e) {

// Show an error message or provide feedback to the user

// Example: display an error dialog

Alert alert = **new** Alert(Alert.AlertType.***ERROR***);

alert.setTitle("Invalid Input");

alert.setHeaderText(**null**);

alert.setContentText(e.getMessage());

alert.showAndWait();

}

});

}

**public** **static** **void** main(String[] args) {

*launch*(args);

}

In the healthdataEntry screen:  
**public** **void** showHealthDataEntryScene() {

primaryStage.setScene(healthDataEntryScene);

primaryStage.setTitle("Health Data Entry");

primaryStage.show();

}

**private** **void** createHealthDataEntryScene() {

// Create UI components for health data entry scene

Label titleLabel = **new** Label("Health Data Entry");

Button bloodPressureButton = **new** Button("Blood Pressure");

Button cholesterolButton = **new** Button("Cholesterol");

Button bmiButton = **new** Button("BMI");

Button bloodSugarButton = **new** Button("Blood Sugar");

Button customHealthNoteButton = **new** Button("Custom Health Note");

Button historyButton = **new** Button("History");

// Create layout container for health data entry scene

VBox root = **new** VBox(10);

root.setAlignment(Pos.***CENTER***);

root.setPadding(**new** Insets(10));

root.getChildren().addAll(titleLabel, bloodPressureButton, cholesterolButton, bmiButton,

bloodSugarButton, customHealthNoteButton, historyButton);

// Create health data entry scene

healthDataEntryScene = **new** Scene(root, 400, 300);

// Handle blood pressure button click event

bloodPressureButton.setOnAction(event -> {

showBloodPressureScene();

});

// Handle cholesterol button click event

cholesterolButton.setOnAction(event -> {

showCholesterolScene();

});

// Handle BMI button click event

bmiButton.setOnAction(event -> {

showBMIScene();

});

// Handle blood sugar button click event

bloodSugarButton.setOnAction(event -> {

showBloodSugarScene();

});

// Handle custom health note button click event

customHealthNoteButton.setOnAction(event -> {

showCustomHealthNoteScene();

});

// Handle history button click event

historyButton.setOnAction(event -> {

showHistoryScreen();

});

}

And in the HistoryScreen class:  
**public** **void** display() {

Stage stage = **new** Stage();

stage.setTitle("Health Data History");

// Create columns for the table

TableColumn<HealthData<?>, String> nameColumn = **new** TableColumn<>("Name");

nameColumn.setCellValueFactory(param -> **new** SimpleStringProperty(param.getValue().getName()));

TableColumn<HealthData<?>, String> dateColumn = **new** TableColumn<>("Date");

dateColumn.setCellValueFactory(param -> **new** SimpleStringProperty(param.getValue().getDate().toString()));

TableColumn<HealthData<?>, String> metricColumn = **new** TableColumn<>("Metric");

metricColumn.setCellValueFactory(param -> **new** SimpleStringProperty(getMetricValue(param.getValue())));

metricColumn.setCellFactory(column -> {

**return** **new** TableCell<HealthData<?>, String>() {

@Override

**protected** **void** updateItem(String item, **boolean** empty) {

**super**.updateItem(item, empty);

**if** (empty || item == **null**) {

setText(**null**);

} **else** {

setText(item);

}

}

};

});

### 6.2 Code Showing *stream*() and Lambdas (separate int0 6.3 and 6.4 if you wish)

In the HistoryScreen class use of lambda function and Stream,

**private** **void** filterDataByDate(LocalDate selectedDate) {

**if** (selectedDate != **null**) {

List<HealthData<?>> filteredData = user.getHealthDataList().stream()

.filter(data -> data.getDate().toInstant().atZone(ZoneId.*systemDefault*()).toLocalDate().equals(selectedDate))

.collect(Collectors.*toList*());

data.setAll(filteredData);

calculateAverageMetrics(filteredData);

} **else** {

data.setAll(user.getHealthDataList());

calculateAverageMetrics(user.getHealthDataList());

}

}

Since this iteration had a lot of javafx work, I used javafx for the most part and used a little bit of lambdas, and stream() functionalities

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

The chatgpt response was once again very helpful to start off. I gathered my own idea and mixed them with the chatgpt response and as you you see in the code, it is quite different than what chatgpt response intended me to do. I added a back button in the historyscreen class which closes the current window and takes the user back to main screen. I rediesigned the wondows so it can fit more content. Later, I made sure the average health metrics cover all the important health metric that was missing in the chatgpt response. Also, re formatted the code to reshape the display contents, added missing codes as I needed to. In the main class, I made sure the User object is using generics which was missed by the chatgpt. Using of generics was important to make sure User class can take any type of healthdata the javafx can add on later.

1. MY PROMPT SEQUENCE (ONLY)

[1] Right now, I don't have database connectivity to store new user info. But this week, I want to be able to login as a user, then input healthdata information and submit. There should also be a history button inside the user screen. Once the user submit healthdata, everything should be displayed in the history page.

[2]

So, in the PHI app, I am required to implement lambdas and streams this week, I have added the following functional requirements for these two concepts,

2.2 Filter health data by date (NEW)

The app shall be able to filter healthdata by date of record

2.3 Calculate average healthmetric (NEW)

The app shall be able to calculate average healthmetric over time

[3] Can you give me codes for those?

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

[1]

To implement the functionality you described, you can follow these steps:

Create a new class, let's call it HealthDataEntryScreen, that represents the screen where users can input their health data.

In the createLoginScene() method of the Main class, add a handler to the login button that checks the username and password against a predefined set of credentials (e.g., using if-else statements or a switch case). If the credentials are valid, switch to the HealthDataEntryScreen.

Implement the HealthDataEntryScreen class to create the UI components for entering health data. This can include labels and input fields for various data points, such as date, weight, blood pressure, etc. Also, add a "Submit" button.

Add a handler to the "Submit" button in the HealthDataEntryScreen class. Inside the handler, retrieve the values entered by the user from the input fields, create a HealthData object (you may need to define this class), and add it to the user's health data list (you can access the current user using a reference or by passing it as a parameter).

Create another class, let's call it HistoryScreen, that represents the screen where the user's health data history will be displayed.

In the HealthDataEntryScreen, add a handler to the "History" button that switches to the HistoryScreen.

Implement the HistoryScreen class to display the user's health data history. This can be done by retrieving the user's health data list and displaying it in a table or a list view.

Update the Main class to hold references to the HealthDataEntryScreen and HistoryScreen instances and switch between them as needed.

By following these steps, you should be able to implement the desired functionality of logging in, entering health data, and displaying the history of health data for a user. Keep in mind that this is a high-level outline, and you'll need to fill in the details and handle edge cases as you implement the code.

[2]

2.2 Filter health data by date:

To filter health data by the date of the record, you can use the filter method of streams along with a lambda expression. Here's an example implementation:

java

Copy code

import java.util.List;

import java.util.stream.Collectors;

public class HealthDataFilter {

public List<HealthData> filterByDate(List<HealthData> healthDataList, LocalDate date) {

return healthDataList.stream()

.filter(data -> data.getDate().equals(date))

.collect(Collectors.toList());

}

}

In this example, HealthData represents the data structure for a single health record, and healthDataList is a list of such records. The filterByDate method takes the list and a LocalDate object representing the desired date. It uses the filter method to check if the record's date matches the given date and then collects the matching records into a new list using the collect method.

2.3 Calculate average health metric:

To calculate the average health metric over time, you can use the mapToDouble and average methods of streams. Here's an example implementation:

java

Copy code

import java.util.List;

public class HealthMetricCalculator {

public double calculateAverageHealthMetric(List<HealthData> healthDataList) {

return healthDataList.stream()

.mapToDouble(HealthData::getHealthMetric)

.average()

.orElse(0.0);

}

}

In this example, getHealthMetric is assumed to be a method in the HealthData class that returns the health metric value. The calculateAverageHealthMetric method takes a list of health records, uses mapToDouble to extract the health metric values as a DoubleStream, then calculates the average using the average method. If the list is empty or no average value is present, it returns a default value of 0.0.

You can integrate these implementations into your existing codebase according to your app's architecture and requirements. Let me know if you need any further clarification or assistance!

[3] package application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.stage.Stage;

import PHI.CommonHealthData;

import PHI.HealthData;

import PHI.User;

import java.util.Date;

public class HealthDataEntryScreen {

private Stage primaryStage;

private Scene healthDataEntryScene;

private User currentUser;

public HealthDataEntryScreen(Stage primaryStage, User currentUser) {

this.primaryStage = primaryStage;

this.currentUser = currentUser;

createHealthDataEntryScene();

}

private void createHealthDataEntryScene() {

// Create UI components for health data entry scene

Label nameLabel = new Label("Name:");

TextField nameTextField = new TextField();

Label metricLabel = new Label("Metric:");

TextField metricTextField = new TextField();

Label systolicBPLabel = new Label("Systolic BP:");

TextField systolicBPTextField = new TextField();

Label diastolicBPLabel = new Label("Diastolic BP:");

TextField diastolicBPTextField = new TextField();

Button submitButton = new Button("Submit");

Button backButton = new Button("Back");

// Create layout container for health data entry scene

GridPane root = new GridPane();

root.setAlignment(Pos.CENTER);

root.setHgap(10);

root.setVgap(5);

root.setPadding(new Insets(10));

// Add UI components to the grid pane

root.add(nameLabel, 0, 0);

root.add(nameTextField, 1, 0);

root.add(metricLabel, 0, 1);

root.add(metricTextField, 1, 1);

root.add(systolicBPLabel, 0, 2);

root.add(systolicBPTextField, 1, 2);

root.add(diastolicBPLabel, 0, 3);

root.add(diastolicBPTextField, 1, 3);

root.add(submitButton, 0, 4);

root.add(backButton, 1, 4);

// Create health data entry scene

healthDataEntryScene = new Scene(root, 500, 500);

// Handle submit button click event

submitButton.setOnAction(event -> {

// Retrieve health data input from the user

String name = nameTextField.getText();

String metric = metricTextField.getText();

int systolicBP = Integer.parseInt(systolicBPTextField.getText());

int diastolicBP = Integer.parseInt(diastolicBPTextField.getText());

// Create a CommonHealthData object

CommonHealthData healthData = new CommonHealthData(name, new Date(), metric, systolicBP, diastolicBP);

// Add the health data to the current user's health data list

currentUser.addHealthData(healthData);

// Show a success message or provide feedback to the user

// Example: display a dialog

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Health Data Submission");

alert.setHeaderText(null);

alert.setContentText("Health data submitted successfully!");

alert.showAndWait();

});

// Handle back button click event

backButton.setOnAction(event -> {

// Switch back to the login scene

primaryStage.setScene(loginScene);

});

}

public Scene getScene() {

return healthDataEntryScene;

}

}

package application;

import javafx.application.Application;

import javafx.scene.Scene;

import javafx.stage.Stage;

import PHI.User;

public class Main extends Application {

private Stage primaryStage;

private Scene loginScene;

private Scene createUserScene;

private HealthDataEntryScreen healthDataEntryScreen;

private HistoryScreen historyScreen;

private User currentUser;

@Override

public void start(Stage primaryStage) {

this.primaryStage = primaryStage;

createLoginScene();

createCreateUserScene();

// Set the initial scene to the login scene

primaryStage.setScene(loginScene);

primaryStage.setTitle("Login/Create User");

primaryStage.show();

}

private void createLoginScene() {

// ...

// Existing code for creating the login scene

// Handle login button click event (placeholder action)

loginButton.setOnAction(event -> {

// Placeholder action, you can add your own logic here

String username = usernameTextField.getText();

String password = passwordTextField.getText();

System.out.println("Logging in with username: " + username + ", password: " + password);

// Assuming login is successful and a user object is obtained

currentUser = new User(username, password);

// Create the health data entry screen and history screen

healthDataEntryScreen = new HealthDataEntryScreen(primaryStage, currentUser);

historyScreen = new HistoryScreen(primaryStage, currentUser);

// Switch to the health data entry scene

primaryStage.setScene(healthDataEntryScreen.getScene());

});

}

private void createCreateUserScene() {

// ...

// Existing code for creating the create user scene

// Handle create user button click event

createUserButton.setOnAction(event -> {

// Placeholder code for creating a new user

currentUser = new User(firstNameTextField.getText(), lastNameTextField.getText(), emailTextField.getText(), passwordTextField.getText(), null, null, null);

// Create the health data entry screen and history screen

healthDataEntryScreen = new HealthDataEntryScreen(primaryStage, currentUser);

historyScreen = new HistoryScreen(primaryStage, currentUser);

// Switch to the health data entry scene

primaryStage.setScene(healthDataEntryScreen.getScene());

});

}

public static void main(String[] args) {

launch(args);

}

}

package application;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import javafx.scene.control.DatePicker;

import java.time.LocalDate;

import java.time.ZoneId;

import java.util.Date;

import PHI.User;

public class Main extends Application {

private Stage primaryStage;

private Scene loginScene;

private Scene createUserScene;

private HealthDataEntryScreen healthDataEntryScreen;

private User currentUser;

@Override

public void start(Stage primaryStage) {

this.primaryStage = primaryStage;

createLoginScene();

createCreateUserScene();

// Set the initial scene to the login scene

primaryStage.setScene(loginScene);

primaryStage.setTitle("Login/Create User");

primaryStage.show();

}

private Date convertToDate(LocalDate localDate) {

return Date.from(localDate.atStartOfDay(ZoneId.systemDefault()).toInstant());

}

private void createLoginScene() {

// Create UI components for login scene

Label usernameLabel = new Label("Username:");

TextField usernameTextField = new TextField();

Label passwordLabel = new Label("Password:");

PasswordField passwordTextField = new PasswordField();

Button loginButton = new Button("Login");

Button createUserButton = new Button("Create User");

// Create layout container for login scene

GridPane root = new GridPane();

root.setAlignment(Pos.CENTER);

root.setHgap(10);

root.setVgap(5);

root.setPadding(new Insets(10));

// Add UI components to the grid pane

root.add(usernameLabel, 0, 0);

root.add(usernameTextField, 1, 0);

root.add(passwordLabel, 0, 1);

root.add(passwordTextField, 1, 1);

root.add(loginButton, 0, 2);

root.add(createUserButton, 1, 2);

// Create login scene

loginScene = new Scene(root, 400, 200);

// Handle create user button click event

createUserButton.setOnAction(event -> {

primaryStage.setScene(createUserScene);

});

// Handle login button click event

loginButton.setOnAction(event -> {

// Placeholder action, you can add your own logic here

String username = usernameTextField.getText();

String password = passwordTextField.getText();

// Perform authentication and validation logic here

// Replace the condition below with your actual authentication logic

if (username.equals("admin") && password.equals("password")) {

// Authentication successful, switch to the health data entry scene

currentUser = new User(username, password);

healthDataEntryScreen = new HealthDataEntryScreen(primaryStage, currentUser);

primaryStage.setScene(healthDataEntryScreen.getScene());

} else {

// Authentication failed, show an error message

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Authentication Failed");

alert.setHeaderText(null);

alert.setContentText("Invalid username or password.");

alert.showAndWait();

}

});

}

private void createCreateUserScene() {

// Create UI components for create user scene

Label firstNameLabel = new Label("First Name:");

TextField firstNameTextField = new TextField();

Label lastNameLabel = new Label("Last Name:");

TextField lastNameTextField = new TextField();

Label emailLabel = new Label("Email:");

TextField emailTextField = new TextField();

Label passwordLabel = new Label("Password:");

PasswordField passwordTextField = new PasswordField();

Label dateOfBirthLabel = new Label("Date of Birth:");

DatePicker dateOfBirthPicker = new DatePicker();

Label genderLabel = new Label("Gender:");

TextField genderTextField = new TextField();

Label phoneNumberLabel = new Label("Phone Number:");

TextField phoneNumberTextField = new TextField();

Button createUserButton = new Button("Create User");

// Create layout container for create user scene

VBox root = new VBox(5);

root.setAlignment(Pos.CENTER);

root.setPadding(new Insets(10));

root.getChildren().addAll(

firstNameLabel, firstNameTextField,

lastNameLabel, lastNameTextField,

emailLabel, emailTextField,

passwordLabel, passwordTextField,

dateOfBirthLabel, dateOfBirthPicker,

genderLabel, genderTextField,

phoneNumberLabel, phoneNumberTextField,

createUserButton

);

// Create create user scene

createUserScene = new Scene(root, 500, 500);

// Handle create user button click event

createUserButton.setOnAction(event -> {

// Retrieve user input

String firstName = firstNameTextField.getText();

String lastName = lastNameTextField.getText();

String email = emailTextField.getText();

String password = passwordTextField.getText();

LocalDate dateOfBirth = dateOfBirthPicker.getValue();

String gender = genderTextField.getText();

String phoneNumber = phoneNumberTextField.getText();

try {

// Validate email and password inputs using User class methods

User user = new User(firstName, lastName, email, password, convertToDate(dateOfBirth), gender, phoneNumber);

// Perform any additional operations with the created user object

// For example, save the user to a database

// Show a success message or provide feedback to the user

// Example: display a dialog

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("User Creation");

alert.setHeaderText(null);

alert.setContentText("User created successfully!");

alert.showAndWait();

// Switch back to the login scene

primaryStage.setScene(loginScene);

} catch (IllegalArgumentException e) {

// Show an error message or provide feedback to the user

// Example: display an error dialog

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Invalid Input");

alert.setHeaderText(null);

alert.setContentText(e.getMessage());

alert.showAndWait();

}

});

}

public static void main(String[] args) {

launch(args);

}

}

## 7 YOUR CODE

Unless your facilitator arranges another method, copy your Eclipse project to your file system, zip it, and attach it to your Blackboard response. Please contact your facilitator in advance if you require an exception.

Github link: <https://github.com/mahimcbu/CS622/tree/main/Assignments/Hw4/PhiFx>  
  


Readme file content:  
1. Please use the Main class's main method to run the program.

2. There are Junit test files for a few important classes and methods

3 I implemented JavaFX in this iteration. Follow this link in order to run JavaFX in eclipse.

https://openjfx.io/openjfx-docs/

4. I haven't implemented user log in yet as I am waiting for the database connectivity.

How to use the GUI:

5. click create user and input fields to create a user. Only email and password are validated as of now.

6. password must be 8 chars, minimum of one uppercase, number, special char.

Once I have database connectivity, everything else in the app should work properly.

Note: The main class creates the user, and user health info that will be user input later.

Note: The main program will change every week as we introduce more features to the project.

### >>AI generation OR check: *I did not use AI generation here* \_\_(Collapse section before submitting.)

1. PARAGRAPH DESCRIBING YOUR VALUE ADDED TO THE AI-GENERATED MATERIAL

As you can see the chatgpt gave me a good headstart but it is missing some information that I needed to fill up, also, it needed fix on where and how it’s validating the user login, healthdata creation.

1. MY PROMPT SEQUENCE (ONLY)

[1] Right now, I don't have database connectivity to store new user info. But this week, I want to be able to login as a user, then input healthdata information and submit. There should also be a history button inside the user screen. Once the user submit healthdata, everything should be displayed in the history page. Can you give me example codes?

1. CODE OR TEXT PRODUCED BY THE ABOVE PROMPT SEQUENCE

chatGPT X

package application;

import javafx.application.Application;

import javafx.geometry.Insets;

import javafx.geometry.Pos;

import javafx.scene.Scene;

import javafx.scene.control.Alert;

import javafx.scene.control.Button;

import javafx.scene.control.Label;

import javafx.scene.control.PasswordField;

import javafx.scene.control.TextField;

import javafx.scene.layout.GridPane;

import javafx.scene.layout.VBox;

import javafx.stage.Stage;

import javafx.scene.control.DatePicker;

import java.time.LocalDate;

import java.time.ZoneId;

import java.util.Date;

import PHI.User;

import PHI.HealthData;

import java.time.LocalDate;

import java.util.Date;

public class Main extends Application {

private Stage primaryStage;

private Scene loginScene;

private Scene createUserScene;

private Scene healthDataEntryScene;

private Scene historyScene;

private User currentUser; // Holds the currently logged-in user

@Override

public void start(Stage primaryStage) {

this.primaryStage = primaryStage;

createLoginScene();

createCreateUserScene();

createHealthDataEntryScene();

createHistoryScene();

// Set the initial scene to the login scene

primaryStage.setScene(loginScene);

primaryStage.setTitle("Login/Create User");

primaryStage.show();

}

private Date convertToDate(LocalDate localDate) {

return Date.from(localDate.atStartOfDay(ZoneId.systemDefault()).toInstant());

}

private void createLoginScene() {

// Create UI components for login scene

Label usernameLabel = new Label("Username:");

TextField usernameTextField = new TextField();

Label passwordLabel = new Label("Password:");

PasswordField passwordTextField = new PasswordField();

Button loginButton = new Button("Login");

Button createUserButton = new Button("Create User");

// Create layout container for login scene

GridPane root = new GridPane();

root.setAlignment(Pos.CENTER);

root.setHgap(10);

root.setVgap(5);

root.setPadding(new Insets(10));

// Add UI components to the grid pane

root.add(usernameLabel, 0, 0);

root.add(usernameTextField, 1, 0);

root.add(passwordLabel, 0, 1);

root.add(passwordTextField, 1, 1);

root.add(loginButton, 0, 2);

root.add(createUserButton, 1, 2);

// Create login scene

loginScene = new Scene(root, 400, 200);

// Handle create user button click event

createUserButton.setOnAction(event -> {

primaryStage.setScene(createUserScene);

});

// Handle login button click event

loginButton.setOnAction(event -> {

// Placeholder action, you can add your own logic here

String username = usernameTextField.getText();

String password = passwordTextField.getText();

// Placeholder validation, replace with your own user authentication logic

if (username.equals("admin") && password.equals("password")) {

currentUser = new User("John", "Doe", "johndoe@example.com", "password", new Date(), "Male", "1234567890");

primaryStage.setScene(healthDataEntryScene);

} else {

// Show an error message or provide feedback to the user

// Example: display an error dialog

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Invalid Credentials");

alert.setHeaderText(null);

alert.setContentText("Invalid username or password");

alert.showAndWait();

}

});

}

private void createCreateUserScene() {

// Create UI components for create user scene

// (omitted for brevity)

}

private void createHealthDataEntryScene() {

// Create UI components for health data entry scene

// (omitted for brevity)

// Create layout container for health data entry scene

// (omitted for brevity)

// Create health data entry scene

healthDataEntryScene = new Scene(root, 500, 500);

// Handle submit button click event

submitButton.setOnAction(event -> {

// Retrieve health data input from the user

// (omitted for brevity)

try {

// Create a HealthData object

HealthData healthData = new HealthData(date, weight, bloodPressure, ...);

// Add the health data to the current user's health data list

currentUser.addHealthData(healthData);

// Show a success message or provide feedback to the user

// Example: display a dialog

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Health Data Submission");

alert.setHeaderText(null);

alert.setContentText("Health data submitted successfully!");

alert.showAndWait();

// Switch to the history scene

primaryStage.setScene(historyScene);

} catch (IllegalArgumentException e) {

// Show an error message or provide feedback to the user

// Example: display an error dialog

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Invalid Input");

alert.setHeaderText(null);

alert.setContentText(e.getMessage());

alert.showAndWait();

}

});

}

private void createHistoryScene() {

// Create UI components for history scene

// (omitted for brevity)

// Create layout container for history scene

// (omitted for brevity)

// Create history scene

historyScene = new Scene(root, 800, 600);

// Handle back button click event

backButton.setOnAction(event -> {

primaryStage.setScene(healthDataEntryScene);

});

// Display the user's health data history in a table or list view

// (omitted for brevity)

}

public static void main(String[] args) {

launch(args);

}

}

## 8 Evaluation

