camera rental application:

The Camera Rental Application allows users to register, login, and rent cameras for a specified duration. Users can also manage their wallet balance by depositing and withdrawing money. The application provides a list of available cameras with their details, allowing users to choose and rent a camera based on their preferences.

Application flow:

Step 1: User Class

1. The User class is defined with variables username, password, and wallet balance.
2. The class has a constructor that initializes the username, password, and wallet balance.
3. Getter methods are provided to retrieve the username, password, and wallet balance.
4. The User class also includes methods to deposit and withdraw funds from the wallet balance.

Step 2: Camera Class

1. The Camera class is defined with variables camera ID, brand, model, rental price per day, and rent status.
2. This class has a constructor that initialize the camera ID, brand, model, and rental price per day. The rented status is set to false by default.
3. Getter methods are provided to retrieve the camera ID, brand, model, rental price per day, and rent status.

Step 3: Cam Class

1. The Cam class is the main class for the Camera Rental App.
2. The class includes a main method it uses a while loop.Inside the while loop, the user is prompted to either register, login, or exit the application.
3. If the user chooses to register, they are prompted to enter a username and password, and a new User object is created.
4. If the user chooses to login, they are prompted to enter their username and password, and the input is validated.
5. Once logged in, the user is presented with a menu to view cameras, rent a camera, list all cameras, manage their wallet balance, or logout.
6. Depending on the user's choice, different methods are handled the corresponding such as displaying cameras, renting a camera, managing wallet balance, etc.
7. The user can repeat these actions until they choose to logout, at which point they are returned to the login menu.

Top of Form

Sprints:

Sprint 1:

Create the User class with attributes like username, password, and wallet balance

Create the Camera class with attributes like cameraId, brand, model, and rental price per day

Implement the code for registering a user and logging in

Sprint 2:

Implement for code adding and removing cameras

Allow users to view all available cameras

Allow users to rent a camera if they have sufficient balance in their wallet if not show insufficient balance.

Sprint 3:

Implement wallet management functionalities like depositing and withdrawing money

Display the available wallet balance to the user

Enable users to manage their rented cameras

Implement the logout functionality

Algorithm:

Step 1. Create a class named "User" with private attributes: username (String), password (String), walletBalance (double).

- Include a constructor to initialize the username, password, and set the wallet balance to 0.

- Include getter methods for username, password, and walletBalance.

- Include methods to deposit and withdraw money from the wallet.

Step 2. Create a class named "Camera" with private attributes: cameraId (int), brand (String), model (String), rentalPricePerDay (double), rented (boolean).

- Include a constructor to initialize the cameraId, brand, model, rentalPricePerDay, and set rented to false.

- Include getter and setter methods for cameraId, brand, model, rentalPricePerDay, and rented.

Step 3. Create a class named "Cam" which serves as the main class.

- Create a Scanner object for user input and a List to store Camera objects.

- Declare a User object named "currentUser" to keep track of the currently logged-in user.

Step 4. Implement the main method to start the application.

- Create a while loop with a condition "running" to control the execution of the program.

- Inside the loop, check if a user is currently logged in (currentUser is null).

- If no user is logged in:

- Display the login menu options.

- Prompt the user for their choice and handle it accordingly.

- If the user chooses to register, call the registerUser() method.

- If the user chooses to log in, call the loginUser() method.

- If the user chooses to exit, set "running" to false to terminate the program.

- If a user is logged in:

- Display the user menu options.

- Prompt the user for their choice and handle it accordingly.

- If the user chooses to view cameras, call the displayCameras() method.

- If the user chooses to rent a camera, call the rentCamera() method.

- If the user chooses to view all cameras, call the listCameras() method.

- If the user chooses to manage their wallet, call the displayWalletBalance() method.

- If the user chooses to logout, call the logoutUser() method.

Step 5. Implement methods used in the main method.

- displayLoginMenu(): Display the login menu options.

- displayUserMenu(): Display the user menu options.

- displayCameras(): Display camera-related options.

- registerUser(): Prompt the user to enter username and password, and create a new User object.

- loginUser(): Prompt the user to enter username and password, and check if the credentials match.

- logoutUser(): Set the currentUser to null and display a logout message.

- listCameras(): Display the list of available cameras with their details.

- addCamera(): Prompt the user to enter camera details and add a new Camera object to the list.

- removeCamera(): Prompt the user to enter the camera ID and remove the corresponding Camera object from the list.

- rentCamera(): Prompt the user to enter the camera ID, check if the camera is available, and rent it if possible.

- displayWalletBalance(): Display wallet-related options.

- depositToWallet(): Prompt the user to enter the deposit amount and update the wallet balance accordingly.

- withdrawFromWallet(): Prompt the user to enter the withdrawal amount and update the wallet balance if sufficient funds are available.

- showAvailableBalance(): Display the current wallet balance.

Github link: