**Algorithm :**

1. Start the CameraRentalApp class.

2.Initialize the cameraList, MycameraList, scanner, cameraIdCounter, and wallet variables.

3.Create a list of User objects and add predefined user credentials.

4.Create a list of Camera objects and add predefined camera details.

5.Prompt the user to enter their username and password.

6.Validate the user's credentials by comparing them with the list of User objects.

7.If the credentials are invalid, display an error message and exit the program.

8.Display the main menu.

9.Loop indefinitely until the program is terminated.

1. Prompt the user to enter their choice.

11.Based on the user's choice, perform the following actions:

11.1. If the choice is 1, go to the "My Camera" submenu.

11.2. If the choice is 2, go to the "Rent a Camera" submenu.

11.3. If the choice is 3, go to the "View All Cameras" submenu.

11.4. If the choice is 4, go to the "My Wallet" submenu.

11.5. If the choice is 5, exit the program.

11.6. If the choice is invalid, display an error message.

12.Implement the "My Camera" submenu:

12.1. Display the submenu options.

12.2. Prompt the user to enter their choice.

12.3. Based on the user's choice, perform the following actions:

i. If the choice is 1, go to the "Add Camera" function.

ii. If the choice is 2, go to the "Remove Camera" function.

iii. If the choice is 3, go to the "View My Cameras" function.

iv. If the choice is 4, return to the main menu.

v. If the choice is invalid, display an error message.

13.Implement the "Add Camera" function:

13.1. Prompt the user to enter the camera brand, model, and rental amount.

13.2. Create a new Camera object with the entered details and add it to the cameraList.

13.3. Display a success message.

14.Implement the "Remove Camera" function:

14.1. Prompt the user to enter the camera ID to remove.

14.2. Iterate through the cameraList and remove the camera with the matching ID.

14.3. Display a success message if the camera is found and removed, or an error message if not found.

15.Implement the "View My Cameras" function:

15.1. Check if the MycameraList is empty.

15.2. If empty, display a message indicating no cameras.

15.3. If not empty, display the details of each camera in the MycameraList.

16Implement the "View All Cameras" submenu:

16.1. Check if the cameraList is empty.

16.2. If empty, display a message indicating no cameras.

16.3. If not empty, display the details of each camera in the cameraList.

17.Implement the "Rent a Camera" function:

17.1. Display the available cameras for rent.

17.2. Prompt the user to enter the camera ID they want to rent.

17.3. Check if the user has sufficient balance in their wallet.

17.4. If the balance is sufficient, set the camera as rented, add it to the MycameraList, and deduct the rental amount from the wallet balance.

17.5. Display a success message if the transaction is successful, or an error message if the balance is insufficient.

18.Implement the "My Wallet" submenu:

18.1. Display the current balance in the wallet.

18.2. Prompt the user if they want to deposit an amount to the wallet

1. Finally implement the “Exit” in the menu.

**Sprints:**

**Sprint1:**

1. Create the Camera class with its attributes (id, brand, model, rentalAmount, rented) and corresponding getters and setters.

2.Create the User class with username and password attributes, along with getters and setters.

1. Create the Wallet class with a balance attribute and methods to get and set the balance.
2. Implement the CameraRentalApp class with its main method.
3. Set up the initial list of cameras, users, and the wallet in the CameraRentalApp constructor.
4. Implement the login functionality in the CameraRentalApp's run method, allowing users to enter their credentials.

**Sprint2:**

1. Implement the displayMenu method in the CameraRentalApp class to show the main menu options.
2. Add the goMyCamera, goRentCamera, goViewAllCameras, and goMyWallet methods to the CameraRentalApp class as placeholders.
3. Implement the goMyWallet method to display the wallet balance and allow users to deposit money into their wallet.
4. Implement the goMyCamera method to display the sub-menu for managing the user's cameras.
5. Implement the goViewMyCameras method to show the list of cameras rented by the user.
6. Implement the goRemoveCamera method to remove a camera from the cameraList based on user input.
7. Implement the goAddCamera method to add a camera to the cameraList based on user input.
8. Implement the goViewAllCameras method to display the list of all available cameras.

**Sprint3:**

1. Implement the goRentCamera method to show the list of available cameras and allow users to rent a camera.
2. Add a list (MycameraList) to store the cameras rented by the user.
3. Update the goRentCamera method to handle the rental process, including checking the wallet balance, updating camera status, and deducting the rental amount from the wallet balance.
4. Implement the main method in the CameraRentalApp class to create an instance of the app and start the program.

**Note:** This breakdown of sprints is a general guideline and may vary based on your specific requirements and implementation approach.