**Project 2 : Camera Rental Application**

Project Agenda: Build a peer-to-peer camera rental application.

Scenario:

You have been hired by a company called rentmycam.io as a Full Stack Developer with the aim to

create a prototype of a camera rental application.

The tasks that need to be performed by you are:

• Create a specification document detailing the product’s capabilities, its appearance, and

user interactions.

o Document the application flow.

• Detail the number and duration of sprints required.

o Plan more than two sprints to complete the application.

• Implement Java concepts in the creation of this application.

o This includes data structures for sorting and searching.

• Include the following operations in the application:

o List a camera along with its brand, model, and per-day rental amount.

o Select a camera from a listing to rent.

o Add or view wallet amount.

o Navigate from the current execution context to the main context.

o Close the application.

• Create a welcome screen with the application name and developer details.

o Include options that the user can interact with to access various features of the

application

Application features:

• The user can add a new camera, along with its details, to the existing camera list.

• The application can display the list of cameras available for rent.

o The list will include details like the camera brand, model, and per-day rent amount.

• The user can select a camera to rent for a day.

• The user will only be allowed to rent the camera if there is sufficient balance in the user’s

wallet to fulfil the per-day rent amount. If not, a message is returned.

Insufficient wallet amount

• If there is sufficient balance, the amount will be deducted from the wallet.

• If the list of cameras is empty, an appropriate message is displayed: No Data Present at

This Moment.

• The user can add or view the balance in the wallet through a sub-menu.

• The user can deposit an amount to increase the wallet balance.

o A message is displayed to indicate the status of the operation.

• The application implements concepts like exceptions, collections, and sorting techniques for

source code optimization and increased performance.

**1. Application Overview**

The Camera Rental Application, named *Rentmycam.io*, allows users to browse, rent, and manage their wallets for camera rental. Users can add cameras to the system, view available cameras, rent cameras, and manage their wallet balance. The app facilitates adding, viewing, renting, and removing cameras. The user's balance is deducted when renting a camera.

**2. Product Capabilities**

**Main Features:**

* **Add Camera**: Admin (or the user who has the permissions) can add cameras by specifying the brand, model, and rental price.
* **View Cameras**: The list of available cameras will be displayed, with details such as brand, model, and rental cost per day.
* **Rent Camera**: The user can choose a camera from the available list and rent it by providing the model name. The user’s wallet will be charged the rent amount, and the camera will be removed from the list.
* **Manage Wallet**: The user can check the wallet balance and deposit money into the wallet.
* **Remove Camera**: Admin or the user can remove a camera from the system by specifying the model.

**User Interface Flow:**

* **Welcome Screen**: The first screen will greet the user with the app’s name and a list of options:
  + View available cameras
  + Add a camera
  + Rent a camera
  + Manage wallet
  + Remove camera
  + Exit
* **Display Cameras**: When the "View available cameras" option is selected, the user will see a list of cameras available to rent.
* **Add Camera**: On selecting this option, the user will enter details about the camera, and it will be added to the available cameras list.
* **Rent Camera**: The user will select the camera model they wish to rent. If they have sufficient balance, the camera will be rented, and the balance will be deducted.
* **Manage Wallet**: The user can choose to either view their wallet balance or deposit money into it.
* **Remove Camera**: The user will specify the model of the camera they wish to remove, and it will be deleted from the available list.

**3. User Interaction Flow**

1. **Welcome Screen**:
   * Displays the app name and the list of options.
   * User selects an option.
2. **View Cameras**:
   * If the list is empty, show "No cameras available."
   * If cameras are available, display the camera details (brand, model, rent per day).
3. **Add Camera**:
   * User enters the brand, model, and rent per day.
   * Camera is added to the list and the message "Camera added successfully" is shown.
4. **Rent Camera**:
   * User enters the model of the camera they want to rent.
   * The system checks if the user has enough funds in the wallet.
   * If funds are sufficient, the camera is rented, and the wallet is updated.
   * If insufficient funds, an error message is shown.
5. **Manage Wallet**:
   * User can choose between:
     + **View Balance**: Display the current wallet balance.
     + **Deposit Money**: User can input the deposit amount and add it to their wallet balance.
6. **Remove Camera**:
   * User enters the model of the camera to be removed.
   * If the camera exists in the list, it is removed and a confirmation message is shown.
7. **Exit**: The app exits, and the session ends.

**4. Application Flow**

plaintext

Copy code

--------------------------------------------

| Welcome to Rentmycam.io |

--------------------------------------------

| 1. View Available Cameras |

| 2. Add Camera |

| 3. Rent a Camera |

| 4. Manage Wallet |

| 5. Remove Camera |

| 6. Exit |

--------------------------------------------

User chooses option: 1

|-> Displays available cameras

|-> User chooses to rent a camera or manage wallet

|-> User is given the option to deposit money if balance is insufficient

|-> Option to exit when the user is done

After exit:

|-> Terminate the program

**5. Sprint Planning and Duration**

**Sprint 1 (Duration: 4 days)**:

* **Objective**: Basic Setup and Core Features Implementation.
* **Tasks**:
  + Setup project environment and basic structure (folders, packages, main class).
  + Implement Camera and User classes.
  + Implement add camera, view cameras, and manage wallet functionalities.
  + Create the welcome screen and main menu.

**Sprint 2 (Duration: 4 days)**:

* **Objective**: Add Rent and Remove Camera Features.
* **Tasks**:
  + Implement rent camera functionality.
  + Implement remove camera functionality.
  + Handle balance deduction and validation.
  + Add validations and error handling (insufficient balance, camera not found).
  + Create a feedback mechanism (confirmations on adding/removing cameras, renting success).

**Sprint 3 (Duration: 4 days)**:

* **Objective**: Finalization and Testing.
* **Tasks**:
  + Refine user interface and input prompts for clarity.
  + Test each feature with different test cases.
  + Perform integration testing to ensure proper functionality of features (e.g., wallet balance deduction, camera renting).
  + Code refactoring, adding comments, and preparing for deployment.

**Sprint 4 (Duration: 2 days)**:

* **Objective**: Review, Documentation, and User Feedback.
* **Tasks**:
  + Write documentation for the application (user guide, system architecture).
  + Conduct final user acceptance testing (UAT).
  + Collect feedback and make final adjustments.

**6. Java Concepts Implemented**

* **Data Structures**:
  + ArrayList is used to store and manage the cameras and the wallet balance.
  + Lists of camera models and rent amounts are used for rent management.
* **Control Structures**:
  + if conditions are used to check if the user has sufficient funds in the wallet and if a camera exists in the system.
  + Loops (for/while) to iterate through cameras and other collections.
* **Methods and Static Variables**:
  + Methods like addCamera(), viewCameras(), removeCamera(), and manageWallet() encapsulate specific behaviors and actions.
  + The User class uses a static variable (walletbalance) for shared data.
* **Input Validation**:
  + Ensuring the model exists before renting or removing it.
  + Checking if the wallet has sufficient balance before renting.

**package** miniproject2;

**public** **class** Camera {

String brand;

String model;

**double** rentperday;

**public** Camera(String brand,String model,**double** rentperday) {

**this**.brand=brand;

**this**.model=model;

**this**.rentperday=rentperday;

}

**public** **void** getbrand(String brand) {

**this**.brand=brand;

}

**public** String setbrand() {

**return** brand;

}

**public** **void** getmodel(String model) {

**this**.model=model;

}

**public** String setmodel() {

**return** model;

}

**public** **void** getrentperday(**double** rentperday) {

**this**.rentperday=rentperday;

}

**public** **double** setrentperday() {

**return** rentperday;

}

@Override

**public** String toString() {

**return** "[brand:"+brand+",model:"+model+",rentperday:"+rentperday+"]";

}

}

**package** miniproject2;

**public** **class** User {

String name;

**static** **double** *walletbalance*;

**public** User(String name,**double** walletbalance) {

**this**.name=name;

**this**.*walletbalance*=walletbalance;

}

**public** **static** **void** deposit(**double** amount) {

*walletbalance*+=amount;

System.***out***.println("enter amount to add wallet:"+*walletbalance*);

}

**public** **static** **double** getwalletbalance() {

**return** *walletbalance*;

}

**public** **static** **boolean** canRent(**double** amount) {

**return** *walletbalance* >= amount;

}

**public** **static** **void** deductBalance(**double** amount) {

*walletbalance* -= amount;

}

}

**package** miniproject2;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Scanner;

**public** **class** Camerarentalapp {

**static** List<Camera> *cameras* = **new** ArrayList<>();

**static** List<String> *camModel* = **new** ArrayList<>();

**static** List<Double> *camRent* = **new** ArrayList<>();

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

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("\nWelcome To Rentmycam.io");

**while**(**true**) {

System.***out***.println("\n1. View available cameras");

System.***out***.println("2. Add camera");

System.***out***.println("3. Rent a camera");

System.***out***.println("4. Manage my wallet");

System.***out***.println("5. Remove camera");

System.***out***.println("6. Exit");

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

System.***out***.print("Enter an option: ");

**int** option = sc.nextInt();

sc.nextLine();

**switch** (option) {

**case** 1:

*viewCameras*();

**break**;

**case** 2:

*addCamera*();

**break**;

**case** 3:

*rentCamera*();

**break**;

**case** 4:

*manageWallet*();

**break**;

**case** 5:

*removeCamera*();

**break**;

**case** 6:

System.***out***.println("Exiting application...");

**return**;

**default**:

System.***err***.println("Enter a valid option");

}

}

}

**public** **static** **void** viewCameras() {

**if** (*cameras*.isEmpty()) {

System.***out***.println("No cameras available.");

} **else** {

System.***out***.println("Available cameras:");

**for** (**int** i = 0; i < *cameras*.size(); i++) {

System.***out***.println(i+1 + ". " + *cameras*.get(i));

}

}

}

**public** **static** **void** addCamera() {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.print("Enter brand: ");

String brand = sc.nextLine();

System.***out***.print("Enter model: ");

String model = sc.nextLine();

System.***out***.print("Enter rent per day: ");

**double** rentperday = sc.nextDouble();

sc.nextLine();

Camera newCamera = **new** Camera(brand, model, rentperday);

*cameras*.add(newCamera);

System.***out***.println("Camera added successfully!");

*camModel*.add(model);

*camRent*.add(rentperday);

}

**public** **static** **void** rentCamera() {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println(*cameras*);

System.***out***.println("Enter the camera model to rent:");

String model=sc.nextLine();

**for**(String str: *camModel*) {

**if** (model.equalsIgnoreCase(str)) {

*cameras*.remove(*camModel*.indexOf(model));

User.*walletbalance* -= *camRent*.get(*camModel*.indexOf(model));

System.***out***.println("You have rented a camera");

System.***out***.println("Remaining wallet balance: $"+ User.*getwalletbalance*());

}

}

}

**public** **static** **void** removeCamera() {

**if** (*cameras*.isEmpty()) {

System.***out***.println("No cameras available.");

}**else** {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("Cameras List:");

System.***out***.println(*cameras*);

System.***out***.println("Enter the camera model to remove:");

String removeCamera = sc.nextLine();

**for**(String str: *camModel*) {

**if**(removeCamera.equals(str)){

*cameras*.remove(*camModel*.indexOf(removeCamera));

System.***out***.println("Camera has been removed successfully");

**break**;

}

}

}

}

**public** **static** **void** manageWallet() {

Scanner sc = **new** Scanner(System.***in***);

System.***out***.println("1. View wallet balance");

System.***out***.println("2. Deposit money");

System.***out***.print("Enter option: ");

**int** option = sc.nextInt();

sc.nextLine();

**switch** (option) {

**case** 1:

System.***out***.println("Current wallet balance: $" + User.*getwalletbalance*());

**break**;

**case** 2:

System.***out***.print("Enter deposit amount: ");

**double** depositAmount = sc.nextDouble();

sc.nextLine();

User.*deposit*(depositAmount);

**break**;

**default**:

System.***out***.println("Invalid option. Returning to main menu.");

}

}

}

**Output:**

Welcome To Rentmycam.io

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 1

No cameras available.

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 2

Enter brand: nokia

Enter model: ns200

Enter rent per day: 500

Camera added successfully!

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 2

Enter brand: sony

Enter model: vewin66

Enter rent per day: 700

Camera added successfully!

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 2

Enter brand: canonn

Enter model: ADF300

Enter rent per day: 1000

Camera added successfully!

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 1

Available cameras:

1. [brand:nokia,model:ns200,rentperday:500.0]

2. [brand:sony,model:vewin66,rentperday:700.0]

3. [brand:canonn,model:ADF300,rentperday:1000.0]

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 3

[[brand:nokia,model:ns200,rentperday:500.0], [brand:sony,model:vewin66,rentperday:700.0], [brand:canonn,model:ADF300,rentperday:1000.0]]

Enter the camera model to rent:

ns200

You have rented a camera

Remaining wallet balance: $-500.0

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 4

1. View wallet balance

2. Deposit money

Enter option: 1

Current wallet balance: $-500.0

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 4

1. View wallet balance

2. Deposit money

Enter option: 2

Enter deposit amount: 1000

enter amount to add wallet:500.0

1. View available cameras

2. Add camera

3. Rent a camera

4. Manage my wallet

5. Remove camera

6. Exit

====================

Enter an option: 4

1. View wallet balance

2. Deposit money

Enter option: 2

Enter deposit amount: 1500

enter amount to add wallet:2000.0