CAMERA RENTAL APPLICATION

---

1. INTRODUCTION

The Camera Rental Application is a console-based Java program designed to simplify the process of managing and renting cameras. The system allows an administrator to add or remove cameras, view the full inventory, rent available units, and manage wallet balance for rental transactions. This project demonstrates foundational object-oriented concepts in Java, along with data management using the TreeMap collection.

The application is structured into four classes—**Camera**, **Camera_Functions**, **Wallet**, and **Main**—each collaborating to provide core functionality through a menu-driven interface.

---

2. PROJECT OVERVIEW

This application enables basic camera rental operations. The administrator logs in using predefined credentials, after which the dashboard provides options for camera management, renting, viewing available units, and wallet interactions.

The system stores camera details in a TreeMap<Integer, Camera> structure, ensuring ordered and efficient access. Wallet operations (viewing and adding funds) are implemented via the standalone Wallet class. Camera rental automatically updates availability and deducts rental cost from the wallet if funds suffice.

This project reflects a modular approach, with each class encapsulating a focused responsibility.

---

3. APPLICATION FLOW

Below is the simplified operational flow consistent with your program's actual logic:

Login Stage

- User enters **username** and **password**
- Valid credentials: admin / admin123
- Upon success → navigates to **Main Menu**

Main Menu

1. **My Camera** → Add / Remove / View Cameras
2. **Rent Camera** → Displays available cameras → Rents selected unit
3. **View All Cameras** → Shows complete inventory
4. **My Wallet** → View or add balance
5. **Exit**

Each selection loops back to the main menu unless the user chooses Exit.

Flow Summary Diagram

START → Login

→ If correct → Main Menu

→ (1) Camera Management

→ (2) Rent Camera

→ (3) View All Cameras

→ (4) Wallet

→ (5) Exit → END

→ If incorrect → Terminate

---

4. SPRINT DETAILS

Although the real codebase is compact, the documentation preserves the academic sprint structure, updated to match actual features:

Sprint 1

- Design of class structure
- Creation of Camera class with attributes and accessors
- Implementation of camera storage using TreeMap
- Development of Camera_Functions class
- Adding: Add, Remove, View operations
- Table-format display for listings
- Wallet class implemented
- Wallet balance operations
- Integrated wallet with rental flow

Sprint 2

- Main class menu system
- Login validation
- Connecting menus to functional modules
- Final debugging and formatting
- Final debugging and formatting

---

5. CLASS SPECIFICATIONS

5.1 Camera Class

Attributes

- cameraId : int
- brand : String
- model : String
- price : float
- status : String → (“AVAILABLE” / “RENTED”)

Behaviors

- Getters & setters for all attributes

The class acts purely as a data carrier (POJO).

5.2 Camera_Functions Class

Responsible for camera management and rental logic.

Methods

1. manageCamera(TreeMap<Integer, Camera>)

Allows:

- Add Camera
- Remove Camera
- View All Cameras

2. viewAllCameras(TreeMap<Integer, Camera>)

Displays camera information in formatted table output.

3. rentCamera(TreeMap<Integer, Camera>, float walletBalance)

- Filters available cameras
- Allows the user to enter camera ID
- Checks if wallet has enough funds
- Deducts price and marks camera as RENTED

5.3 Wallet Class

Methods

1. manageWallet(float balance)

Displays menu:

- View Balance
- Add Money
- Back

Returns updated wallet balance.

Wallet logic is not tied to user accounts—only global balance.

5.4 Main Class

Responsibilities

- Displays welcome banner
- Authenticates login
- Loads predefined camera records
- Displays main menu with five options
- Calls appropriate functions based on user selection
- Maintains wallet balance throughout session

Important Characteristics

- Uses a single float walletBalance variable
- Uses TreeMap for sorted camera storage
- The system ends only when user chooses Exit

---

6. TASK SCHEDULING (SIMPLIFIED)

Task scheduling is aligned with the sprint breakdown:

- Designing class structure
- Implementing camera CRUD
- Integrating wallet
- Implementing rental workflow
- Menu integration and logical flow

No background tasks or threads exist in actual implementation.

---

7. APPLICATION FEATURES

✓ **Add Camera**

✓ **Remove Camera**

✓ **View All Cameras**

✓ **Rent Camera (with wallet deduction)**

✓ **Wallet balance management**

✓ **Preloaded camera inventory**

✓ **Login authentication**

✓ **Ordered camera storage using TreeMap**

❌ **Features Not Implemented (but removed from documentation responsibly)**

- Camera return
- Rental history
- User profiles
- Multiple rentals tracking

The final document reflects only implemented features.

---

8. APPLICATION FLOW CHART

Below is a text-form representation suited for academic reports, but accurately simplified to match your code:

Login

↓

Main Menu

├— My Camera

| ├— Add Camera

| ├— Remove Camera

| └— View Cameras

├— Rent Camera

| └— Wallet Check → Rent

├— View All Cameras

├— My Wallet

| ├— View Balance

| └— Add Money

└── Exit

---

9. USER MANUAL

Login

- Username → admin
- Password → admin123

Main Menu Options

1. My Camera

- Add Camera → Enter ID, brand, model, price
- Remove Camera → Enter ID
- View Cameras → Shows full camera list

2. Rent Camera

- Shows only *AVAILABLE* cameras
- Enter ID to rent
- If wallet ≥ price → camera becomes RENTED

3. View All Cameras

Displays entire camera list in a formatted table.

4. My Wallet

- View balance
- Add money

5. Exit

Terminates application.

---

10. SYSTEM REQUIREMENTS

Hardware

- Minimum 2GB RAM
- Basic CPU (any modern processor)

Software

- JDK 8 or above
- Any Java-compatible terminal or IDE
- Operating System: Windows / Mac / Linux

---

11. CONCEPTS USED

- **Object-Oriented Programming**
    - Encapsulation
    - Abstraction
    - Modular design
- **Java Collections**
    - TreeMap<Integer, Camera> for sorted storage
- **Control Structures**
    - Switch-case menus
    - Loops
- **Exception-safe Input Handling**
  (basic, as per project scope)
- **Static Methods**
    - Shared utilities (Camera_Functions, Wallet)

---

12. SAMPLE OUTPUT

(Reflecting exact format produced by your code.)

```
+----------------------------------------+
|     Welcome to Camera Rental App       |
+----------------------------------------+
Enter Username: admin
Enter Password: admin123


1. My Camera
2. Rent Camera
3. View All Cameras
4. My Wallet
5. Exit
Choose option:
```

Example table:

```
+----------+---------------+---------------+------------+------------+
| ID       | Brand         | Model         | Price      | Status     |
+----------+---------------+---------------+------------+------------+
| 12       | Some          | Another       | 100.00     | AVAILABLE  |
| 14       | NIKON         | DSLR-D700     | 500.00     | AVAILABLE  |
...
```

Wallet example:

Wallet Balance: ₹100.0

Rental example:

Enter Camera ID to rent: 12

✅ Rented Some Another

---

13. DEVELOPER INFORMATION

- **Developer Name** : Aman Pandey
- **Employee ID** : 2660292
- **Technology Used** : Java (OOPs)
- **Project Title** : Camera Rental Application

---

14. CONCLUSION

The Camera Rental Application demonstrates a functional console-based system designed using core Java concepts and structured modular programming. The integration of camera management, wallet operations, and rental workflow illustrates effective collaboration between classes, with a clear separation of concerns.

Using TreeMap for ordered data handling and static utility classes for core operations results in a clean and maintainable architecture. While simple in scale, the project successfully models real-world rental systems and provides a strong foundation for further expansion