PHASE-END PROJECT ONE

CAMERA RENTAL APPLICATION

PROJECT AGENDA: Build a peer-to-peer camera rental application

APPLICATION FEATURES:

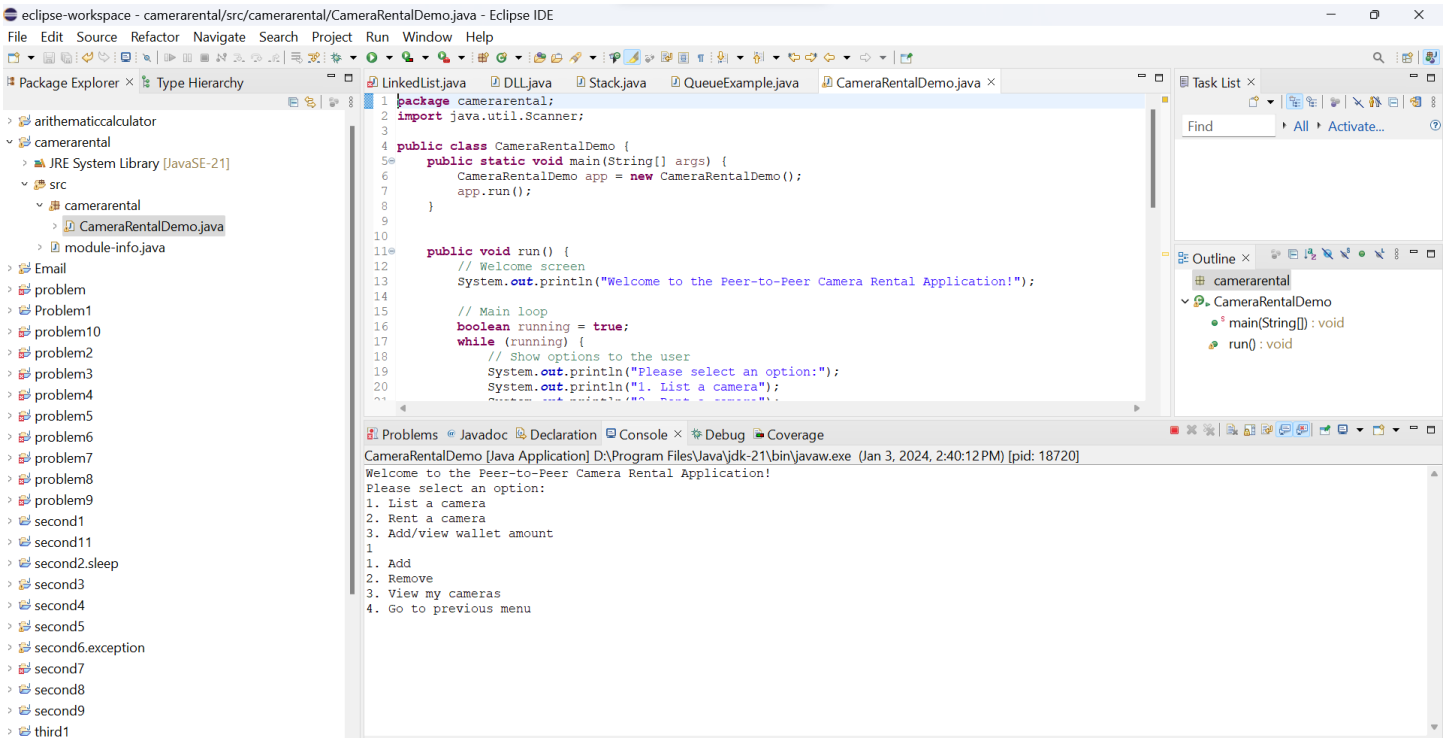
- The user can add a new camera, along with its details, to the existing camera list.
- The application can display the list of cameras available for rent.
- The list will include details like the camera brand, model, and per-day rent amount.
- The user can select a camera to rent for a day.
- The user 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

SOURCE CODE:

```
package camerarental;
import java.util.Scanner;

public class CameraRentalDemo {
      public static void main(String[] args) {
        CameraRentalDemo app = new CameraRentalDemo();
        app.run();
    }


    public void run() {
        // Welcome screen
        System.out.println("Welcome to the Peer-to-Peer Camera Rental Application!");

        // Main loop
        boolean running = true;
        while (running) {
            // Show options to the user
            System.out.println("Please select an option:");
            System.out.println("1. List a camera");
            System.out.println("2. Rent a camera");
            System.out.println("3. Add/view wallet amount");
            System.out.println("4. Exit");

            // Get user input
            Scanner scanner = new Scanner(System.in);
            int option = scanner.nextInt();

            switch (option) {
                case 1:

                    break;
                case 2:

                    break;
```

```
                case 3:

                    break;
                case 4:

                    running = false;
                    break;
                default:
                    System.out.println("Invalid option. Please try again.");
            }
        }

        System.out.println("Thank you for using the Peer-to-Peer Camera Rental
Application!");
    }


}
```

OUTPUT :



ALGORITHM:

1. **Renting a Camera (rent Camera):**

   Accepts a customer, a camera, start date, and end date. Checks if the camera is available and creates a rental if it is. Updates the camera's availability status.

2. **Returning a Camera (return Camera):**

   Marks a rental as returned and updates the camera's availability status.

3. **Generating Transaction (generate Transaction):**

   Creates a transaction for a rental with a total price and records the transaction details.

4. **Managing Cameras, Customers, Rentals, and Transactions:**

   Implement methods to add, remove, update, and retrieve information about cameras, customers, rentals, and transactions.

DESCRIPTION: Title: Building a Peer-to-Peer Camera Rental Application

1. **Introduction**

This document outlines the development of a peer-to-peer camera rental application. By using this application, users can easily rent or lend cameras to others within their network. The primary focus of this application is to enhance user experiences, encourage peer-to-peer sharing, and minimize costs associated with camera rental.

2. **Requirements and Objectives**

- Allow users to sign up, log in, and manage their accounts.
- Enable users to browse, filter, and search for available cameras.
- Facilitate peer-to-peer transactions between camera owners and renters.
- Implement a secure payment system using third-party APIs or cryptocurrencies.
- Offer real-time camera status updates, such as booking availability and current location.
- Ensure compliance with relevant laws and regulations.

3. **Architecture**

- Frontend: Develop a responsive web application using a modern framework like React.js or Angular.js. This will enable the application to work seamlessly on various devices, including smartphones, tablets, and desktop computers.
- Backend: Utilize a scalable serverless architecture with a Node.js backend. This approach allows for faster development, easier scaling, and more cost-effective operations.
- Database: Store user data, camera details, and transaction information in a secure and scalable cloud-based database like MongoDB or PostgreSQL.

- API: Integrate third-party APIs for authentication, location services, and secure payment systems.

4. **Implementation Plan**

- User Account Management: Develop user authentication and authorization functionalities using a secure framework like JWT. Implement user account creation, profile editing, and password recovery features.
- Camera Listing: Create a system for adding, editing, and removing camera listings. Enable users to upload high-quality images of their cameras. Implement filtering and search options based on camera specifications, price, and availability.
- Peer-to-Peer Transactions: Implement a transaction management system that allows users to rent or lend cameras to other users. Integrate secure payment methods using third-party APIs or cryptocurrencies.
- Real-time Updates: Develop a real-time camera status update system that displays booking availability and current location information.
- Compliance and Security: Ensure compliance with relevant laws and regulations, such as data privacy and GDPR. Implement robust security measures, including encryption, HTTPS, and secure server configurations.

5. **Testing and Deployment**

- Thoroughly test the application for bugs, security vulnerabilities, and performance issues.
- Deploy the application on a scalable cloud infrastructure like AWS or Google Cloud Platform.
- Monitor the application's performance and fix any issues that arise.

6. **Maintenance and Updates**

- Regularly update the application with new features, bug fixes, and performance improvements.
- Provide comprehensive support to users through email, live chat, or social media platforms.

7. **Conclusion**

By following the outlined architecture, development plan, and maintenance strategies,