Project Documentation: Hotel Reservation System

---

1. Project Overview

The Hotel Reservation System is a Java-based application designed to manage hotel room bookings, reservations, and customer profiles. This project employs the Singleton and Factory design patterns to ensure efficient and modular implementation. The system provides functionalities such as room management, reservation handling, and payment processing, all encapsulated within a GUI.

---

2. Design Patterns

Singleton Pattern

- **ReservationManager**:
    - **Purpose**: Centralized management of all reservations within the system.
    - **Justification**: Ensures only one instance exists to handle reservation operations, preventing data inconsistency.
    - **Implementation**: A static instance with synchronized access in multi-threaded environments.
- **PaymentProcessor**:
    - **Purpose**: Single point for managing payment activities.
    - **Justification**: Guarantees all transactions are processed consistently and securely.
    - **Implementation**: A single instance accessed globally for payment operations.

Factory Pattern

- **RoomFactory**:
    - **Purpose**: Creates room objects dynamically based on user input.
    - **Supported Types**: Standard, Deluxe, Suite.
    - **Justification**: Simplifies object creation, making the application scalable and modular.
- **CustomerFactory**:
    - **Purpose**: Creates customer profile objects dynamically.
    - **Supported Types**: Regular, VIP, Corporate.
    - **Justification**: Provides flexibility to add new customer types without modifying existing code.

---

3. Class Descriptions

ReservationManager (Singleton)

- **Responsibilities**:
    - Manage reservation creation, modification, and deletion.
    - Maintain a central list of all active reservations.
- **Key Methods**:
    - addReservation(): Adds a new reservation.
    - modifyReservation(): Modifies an existing reservation.
    - deleteReservation(): Deletes an existing reservation.

PaymentProcessor (Singleton)

- **Responsibilities**:
    - Handle payment processing and validation.
    - Ensure secure transaction management.
- **Key Methods**:
    - processPayment(): Validates and completes a payment.
    - refundPayment(): Handles payment refunds.

RoomFactory (Factory)

- **Responsibilities**:
    - Dynamically create room objects based on the specified type.
- **Key Methods**:
    - createRoom(String type): Returns an instance of a Room subclass (Standard, Deluxe, Suite).

CustomerFactory (Factory)

- **Responsibilities**:
    - Dynamically create customer profile objects.
- **Key Methods**:
    - createCustomer(String type): Returns an instance of a Customer subclass (Regular, VIP, Corporate).

Room (Abstract Class)

- **Subclasses**: StandardRoom, DeluxeRoom, SuiteRoom.
- **Responsibilities**:
    - Define common attributes like room type, rate, and capacity.
    - Implement specific attributes and behaviors in subclasses.

Customer (Abstract Class)

- **Subclasses**: RegularCustomer, VIPCustomer, CorporateCustomer.
- **Responsibilities**:
    - Define common attributes like name, contact information, and customer ID.
    - Implement specific behaviors and discounts in subclasses.

---

4. GUI Features

Main Dashboard

- Provides navigation to reservation management, room management, and customer profile management.

Reservation Form

- Fields for customer details and room preferences.
- Dropdown for room type (integrates with RoomFactory).

Room Selection

- Dynamically displays available rooms based on type.

Customer Profile Management

- Dropdown for customer type (integrates with CustomerFactory).
- Fields for entering customer details.

Payment Section

- Interfaces with PaymentProcessor for handling transactions.

---

5. Design Justifications

Singleton Pattern:

- Centralizes control for critical components like ReservationManager and PaymentProcessor.
- Ensures thread-safe, consistent state across the application.
- Avoids the overhead of creating multiple instances for globally used components.

Factory Pattern:

- Provides a clear and extensible way to manage object creation.
- Encapsulates the instantiation logic, adhering to the Open/Closed Principle.
- Simplifies the addition of new room and customer types.

---

6. Next Steps

- **Implementation**: Develop Java classes and integrate them with the GUI.
- **Testing**: Test Singleton and Factory implementations for concurrency and scalability.
- **Deployment**: Package the application using the build.xml script for distribution.