Hotel Reservation System

Documentation

Class Descriptions and Design Patterns

1. DatabaseConnection

- **Description:**
    - This class is responsible for managing a single connection to the database throughout the application's lifecycle.
- **Pattern Used:** Singleton
- **Justification:**
    - To ensure only one instance of the database connection exists, reducing resource usage and preventing potential conflicts from multiple connections. It uses lazy initialization for efficiency.

2. Guest

- **Description:**
    - Represents a guest with attributes like name, phone, email, address, city, nationality, and passport number.
    - Includes functionality to save guest data to the database and retrieve guest details by ID.
- **Pattern Used:** Builder
- **Justification:**
    - Simplifies the construction of Guest objects, especially when only some fields are required during instantiation. This avoids the need for multiple constructors and enhances code readability.

3. Room

- **Description:**
    - Represents a hotel room, including details like type, capacity, price, availability, and location.
    - Updates its availability status automatically when a reservation is made or canceled.
- **Pattern Used:** Observer
- **Justification:**
    - Ensures that room availability is updated in real-time when reservation statuses change, maintaining consistency between reservations and rooms without tight coupling.

4. Reservation

- **Description:**
    - Manages hotel room reservations, including guest ID, room number, check-in and check-out dates, and total price calculation.
    - Notifies observers (rooms) about changes in reservation status.
- **Pattern Used:** Observer
- **Justification:**
    - Allows the Reservation class to act as a subject that updates all observers (rooms) regarding changes in reservation status, promoting loose coupling and reusability.

5. Guests

- **Description:**
    - Manages a collection of Guest objects, providing methods to add, remove, and retrieve guests.
- **Pattern Used:** Singleton, Iterator
- **Justification:**
    - **Singleton:** Ensures that the Guests collection is managed by a single instance to avoid redundant data handling and duplication.
    - **Iterator:** Provides a standard way to traverse through the guest collection, encapsulating the iteration logic and improving flexibility.

6. Reservations

- **Description:**
    - Maintains a collection of Reservation objects with methods to add, remove, and retrieve reservations.
- **Pattern Used:** Singleton, Iterator
- **Justification:**
    - **Singleton:** Ensures that there is a centralized management system for all reservations, preventing inconsistencies.
    - **Iterator:** Offers a clean interface for iterating through the reservation collection, enhancing encapsulation and maintainability.

7. DeluxeRoom (Subclass of Room)

- **Description:**
  - A specialized type of room with predefined properties such as type and price.
- **Pattern Used:** Factory Method
- **Justification:**
  - Simplifies the creation of specific room types, allowing easy addition of new types without modifying existing code, adhering to the Open-Closed Principle.

Justification for Design Patterns

1. **Singleton:**
   - Ensures centralized control of shared resources (e.g., database connections, collections of guests and reservations).
   - Prevents unnecessary object creation, saving memory and processing time.
2. **Builder:**
   - Facilitates the creation of complex objects with optional parameters, improving flexibility and readability.
3. **Observer:**
   - Decouples the Reservation and Room classes, enabling real-time updates without creating tight dependencies.
4. **Iterator:**
   - Provides a consistent way to traverse through collections (Guests and Reservations), encapsulating iteration logic and adhering to the Single Responsibility Principle.
5. **Factory Method:**
   - Encapsulates the creation logic for room types, supporting easy extensibility and reducing the impact of changes in the system.