Code for the Hotel Management System

Write the object-oriented code to implement the design of the hotel management system problem.

We'll cover the following

- Hotel management system classes
  - Enumerations
  - Address and account
  - Person
  - Service
  - Invoice
  - Room booking
  - Bill transaction
  - Notification
  - Room, room key and room housekeeping
  - Search and catalog
  - Hotel and hotel branch
- Wrapping up

We've reviewed different aspects of the hotel management system and observed the attributes attached to the problem using various UML diagrams. Let's explore the more practical side of things, where we will work on implementing the hotel management system using multiple languages. This is usually the last step in an object-oriented design interview process.

We have chosen the following languages to write the skeleton code of the different classes present in the hotel management system:

- Java
- C#
- Python
- C++
- JavaScript

Hotel management system classes

In this section, we'll provide the skeleton code of the classes designed in the class diagram lesson.

> **Note:** For simplicity, we are not defining getter and setter functions. The reader can assume that all class attributes are private and accessed through their respective public getter methods and modified only through their public methods function.

Enumerations

First, we will define all the enumerations required in the hotel management system. According to the class diagram, there are six enumerations used in the system, i.e., `RoomStyle`, `RoomStatus`, `BookingStatus`, `AccountStatus`, `AccountType`, and `PaymentStatus`. The code to implement these enumerations is as follows:

> **Note:** JavaScript does not support enumerations, so we will be using the `Object.freeze()` method as an alternative that freezes an object and prevents further modifications.

```
// definition of enumerations used in hotel management system
enum RoomStyle {
    STANDARD,
    DELUXE,
    FAMILY_SUITE,
    BUSINESS_SUITE
}

enum RoomStatus {
    AVAILABLE,
    RESERVED,
    OCCUPIED,
    NOT_AVAILABLE,
    BEING_SERVICED,
    OTHER
}

enum BookingStatus {
    REQUESTED,
    PENDING,
    CONFIRMED,
    CANCELLED,
    ABANDONED
}

enum AccountStatus {
    ACTIVE,
    CLOSED,
    CANCELED,
    BLACKLISTED,
    BLOCKED
```

Enum definitions

Address and account

This section contains the `Address` and `Account` classes. Here, the `Address` class is used as a custom data type. The implementation of these classes is shown below:

```java
public class Address {
    private String streetAddress;
    private String city;
    private String state;
    private int zipCode;
    private String country;
}

public class Account {
    private String id;
    private String password;
    private AccountStatus status;

    public boolean resetPassword();
}
```

The Address and Account classes

Person

`Person` is an abstract class that represents the various people or actors that can interact with the system. There are four types of persons: `Guest`, `Receptionist`, `Server`, and `Housekeeper`. The implementation of the mentioned classes is shown below:

```java
public abstract class Person {
    private String name;
    private Address address;
    private String email;
    private String phone;
    private Account account;
}


public class Guest extends Person {
    private int totalRoomsCheckedIn;

    public List<RoomBooking> getBookings();
}

public class Receptionist extends Person {
    public List<Member> searchMember(String name);
    public boolean createBooking();
}

public class Housekeeper extends Person {
    public boolean assignToRoom();
}
```

Person and its child classes

Service

`Service` is an abstract class, and this section represents different charges, `Amenity`, `RoomService`, and `KitchenService`), against a booking. The code to implement these classes is shown below:

```
public abstract class Service {
    private Date issueAt;

    public boolean addInvoiceItem(Invoice invoice);
}

public class Amenity extends Service {
    private String name;
    private String description;
}

public class RoomService extends Service {
    private boolean isChargeable;
    private Date requestTime;
}

public class KitchenService extends Service {
    private String description;
}
```

Service and its derived classes

Invoice

This section contains information on the `Invoice` class to create a bill. The implementation of this class is represented below:

```
public class Invoice {
    private double amount;

    public boolean createBill();
}
```

The Invoice class

Room booking

`RoomBooking` is a class responsible for managing the bookings for a room. The implementation of this class is given below:

```
public class RoomBooking {
    private String reservationNumber;
    private Date startDate;
    private int durationInDays;
    private BookingStatus status;
    private Date checkin;
    private Date checkout;

    private int guestId;
    private Room room;
    private Invoice invoice;
    private List<Notification> notifications;

    public static RoomBooking fectchDetails(String reservationNumber);
}
```

The RoomBooking class

Bill transaction

After generating an invoice, a customer needs to pay the bill to confirm the booking of the room. A `BillTransaction` class is required to store the information of bill payment. Three ways to pay the bill are check transactions, cash transactions, and credit card transactions.

```
// BillTransaction is an abstract class
public abstract class BillTransaction {
    private Date creationDate;
    private double amount;
    private PaymentStatus status;

    public abstract void initiateTransaction();
}

class CheckTransaction extends BillTransaction {
    private String bankName;
    private String checkNumber;

    public void initiateTransaction() {
        // functionality
    }
}

class CreditCardTransaction extends BillTransaction {
    private String nameOnCard;
    private int zipcode;

    public void initiateTransaction() {
        // functionality
    }
}

class CashTransaction extends BillTransaction {
    private double cashTendered;

```

BillTransaction and its derived classes

Notification

The `Notification` class is another abstract class responsible for sending notifications, with the `SMSNotification` and `EmailNotification` classes as its child. The implementation of this class is given below:

```
// Notification is an abstract class
public abstract class Notification {
    private int notificationId;
    // The Date data type represents and deals with both date and time.
    private Date createdOn;
    private String content;

    public abstract void sendNotification(Person person);
}

class SMSNotification extends Notification {

    public void sendNotification(Person person) {
        // functionality
    }
}
```

```

class EmailNotification extends Notification {

    public void sendNotification(Person person) {
        // functionality
    }
}

```

Notification and its derived classes

Room, room key and room housekeeping

The Room class represents a room in the hotel. `RoomKey` is a class used to express the electronic key card and the `RoomHousekeeping` is a class used to keep track of all the housekeeping records for the rooms. The implementation of these classes is given below:

```
public class Room {
    private String roomNumber;
    private RoomStyle style;
    private RoomStatus status;
    private double bookingPrice;
    private boolean isSmoking;
    private List<RoomKey> keys;
    private List<RoomHousekeeping> housekeepingLog;

    public boolean isRoomAvailable();
    public boolean checkin();
    public boolean checkout();
}

public class RoomKey {
    private String keyId;
    private String barcode;
    private Date issuedAt;
    private boolean isActive;
    private boolean isMaster;

    public boolean assignRoom(Room room);
}

public class RoomHousekeeping
{
    private String description;
    private Date startDatetime;
    private int duration;
    private Housekeeper housekeeper;
```

The Room, RoomKey and RoomHousekeeping classes

Search and catalog

`Search` is an interface, and the `Catalog` class implements the search interface to help in the room search. The code to perform this functionality is presented below:

```java
public interface Search {
    public static List<Room> search(RoomStyle style, Date date, int duration);
}

public class Catalog implements Search {
    private List<Room> rooms;

    public List<Room> search(RoomStyle style, Date date, int duration);
}
```

The Search interface and Catalog class

Hotel and hotel branch

The `Hotel` class is the base class of the system that represents the hotel. The implementation of these classes is given below:

```java
public class HotelBranch {
  private String name;
  private Address location;

  public List<Room> getRooms();
}

public class Hotel {
  private String name;
  private List<HotelBranch> locations;

  public boolean addLocation(HotelBranch location);
}
```

The HotelBranch and Hotel classes

Wrapping up

We've explored the complete design of a hotel management system in this chapter. We've looked at how a basic hotel management system can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.

← Back

Activity Diagram for the Hotel Management System

☐ Complete

Next →

Getting Ready: The Amazon Online Shopping System