Code for the Restaurant Management System

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

We'll cover the following

- Restaurant management system classes
  - Enumerations and custom data type
  - Account and person
  - Table and table seat
  - Meal and meal item
  - Menu, menu section, and menu item
  - Order, kitchen, and reservation
  - Payment and bill
  - Notification
  - Seating chart, branch, and restaurant
- Wrapping up

We've covered different aspects of the restaurant management system and observed the attributes attached to the problem using various UML diagrams. Let us now explore the more practical side of things, where we will work on implementing the restaurant 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 RMS.

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

Restaurant management system classes

In this section, we will 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 method functions.

Enumerations and custom data type

The following code provides the definition of the enumeration and custom data type used in the restaurant management system.

`PaymentStatus` This enumeration keeps track of an order's payment status by the customer.

`TableStatus` This enumeration keeps track of the table's status.

`SeatType` This enumeration represents the type of seat for a customer.

`AccountStatus` This enumeration keeps track of an account's status.

`UserStatus` This enumeration keeps the customer's order status in check.

`ReservationStatus` This enumeration represents the reservation status of a table.

`Address` This custom datatype represents the address of the restaurant's branch or a person.

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

```
// Enumerations
enum PaymentStatus {
    Unpaid,
    Pending,
    Completed,
    Failed,
    Refunded,
    Partial,
    Cancelled,
    Abandoned,
    Settling,
    Settled
};
enum TableStatus {
    Free,
    Reserved,
    Booked,
    Other
};
enum SeatType {
    Regular,
    Kid,
    Accessible,
    Other
};
enum AccountStatus {
    Active,
    Closed,
    Canceled,
    Blacklisted,
    None
};
enum UserStatus {
    Active,
    Closed,
    Canceled,
    Blacklisted,
    None
};
enum ReservationStatus {
    Requested,
    Pending,
    Canceled,
    Confirmed,
    None
};
// Custom data type
struct Address {
    string street_address;
    string city;
    string state;
    string zip_code;
    string country;
};
```

Definition of enums and custom datatype

Account and person

The `Account` class represents a user account that has an ID, address, status, and a password that can be reset and is validated using the `resetPassword()` function.

The `Person` class stores the personal details of a person who can either be an `Employee` or a `Customer` of the restaurant. The derived class `Employee` is further extended by the following:

- `Receptionist` has a method named `createReservation()` to create reservations for customers
- `Manager` has a method named `addEmployee()` to hire new employees to the restaurant
- `Chef` has a method named `prepareMeal()` for preparing the order
- `Waiter` has a method named `pickOrder()` for taking new orders

The derived `Customer` class has a private member that stores the last visited date of the customer. The definitions of these classes are provided below:

```
public class Account {
    private int accountID;
    private String password;
    private Address address;
    private AccountStatus status;

    public boolean resetPassword() {
    }
}
public abstract class Person {
    private String name;
    private String email;
    private String phone;
}
public class Employee extends Person {
    private int employeeID;
    private Account account;
    private Person account_person;
}
public class Customer extends Person {
    private Date lastVisitedDate;
}
public class Receptionist extends Employee {
    public boolean createReservation() {
    }
}
public class Manager extends Employee {
}
```

Account, Person, and its derived classes

Table and table seat

The `Table` class has a unique ID. It stores the table status, table location, and maximum capacity. Reservations can be added to a table using the `addReservation()` method. It also maintains a list of all the seats it has, whom the `TableSeat` is identified by a seat number and has a type. The definitions of these classes are provided below:

```
public class Table {
    private int tableID;
    private TableStatus status;
    private int maxCapacity;
    private int locationIdentifier;
    private List<Reservation> reservations;

    public boolean isTableFree();
    public boolean addReservation();
    public static List<Table> search(int capacity, Date startTime);
}

public class TableSeat {
    private int seatNumber;
    public SeatType type;

    public boolean updateSeatType(SeatType type);
}
```

The Table and TableSeat classes

Meal and meal item

The `MealItem` class is identified by a unique ID and stores and updates the quantity of a menu item. It can be added to a `Meal` using the `addMealItem()` method, where the `Meal` class represents the meal ordered for a specific table. The definitions of these classes are provided below:

```
public class MealItem {
    private int mealItemId;
    private int quantity;
    private MenuItem menuItem;

    public boolean updateQuantity(int quantity);
}

public class Meal {
    private int mealId;
    private bool isFinished;
    private List<MealItem> mealItems;

    public boolean addMealItem(MealItem mealItem);
}
```

The Meal and MealItem classes

Menu, menu section, and menu item

The `Menu` class represents the menu of a restaurant. It is identified by a unique ID and maintains a list of `MenuSection`. Each `MenuSection` class is the representation of a menu's different sections. A `MenuSection` has a list of `MenuItem` on which an item's price can be updated. The definitions of these classes are provided below:

```
public class Menu {
    private int menuId;
    private String title;
    private String description;
    private bool isActive;
    private List<MenuSection> menuSections;

    public boolean addMenuSection(MenuSection menuSection);
    public boolean print();
}

public class MenuSection {
    private int menuSectionId;
    private String title;
    private String description;
    private List<MenuItem> menuItems;

    public boolean addMenuItem(MenuItem menuItem);
}

public class MenuItem {
    private int menuItemId;
    private String title;
    private String description;
    private double price;

    public boolean updatePrice(double price);
}
```

The Menu, MenuSection, and MenuItem classes

Order, kitchen, and reservation

An `Order` has a method to add and is assigned to a waiter and chef for a specific table. Meals can be updated and removed from the `Order` using the `addMeal()` and `removeMeal()` methods.

A `Kitchen` has a number of chefs where new chefs can be assigned using the `assignChef()` method.

The `Reservation` class represents the reservation of a table that stores the reservation time, people count, status, check in time, and customer information. It also maintains a list of notifications for the customer.

The definitions of these classes are provided below:

```
public class Order {
    private int OrderId;
    private OrderStatus status;
    private bool isFinished;
    private Meal[] meals;
    private Waiter waiter;
    private Waiter waiter;
    private Chef chef;

    public boolean addMeal(Meal meal);
    public boolean removeMeal(Meal meal);
}

public class Kitchen {
    private int kitchenId;
    private Chef[] chefs;

    public boolean assignChef();
}

public class Reservation {
    private int reservationId;
    private bool isWalkIn;
    private int peopleCount;
    private ReservationStatus status;
    private String roomNo;
    private Date checkInTime;
    private Customer customer;
    private Table[] tables;
    private List<Notification> notifications;
```

The Order, Kitchen, and Reservation classes

Payment and bill

The `Payment` class is an abstract class with the `Check` `Payment` and `Cash` classes as its child classes. This class has the `PaymentStatus` enum to keep track of the payment status. The `Bill` class represents the bill generated for a customer's order. The definitions of these classes are provided below:

```
// Payment is an abstract class
public abstract class Payment {
    private int paymentId;
    private PaymentStatus status;
    private double amount;
    private PaymentStatus status;

    public abstract void initiateTransaction();
}

public class Check extends Payment {
    private String bankName;
    private String checkNumber;

    public void initiateTransaction() {
        // functionality
    }
}

public class CreditCard extends Payment {
    private String cardHolderName;
    private int cardNo;

    public void initiateTransaction() {
        // functionality
    }
}
public class Cash extends Payment {
    public double cashTendered;
```

Payment and its derived classes

Notification

The `Notification` class is another abstract class responsible for sending notifications, with the `sendNotification` and `sendVerification` classes as its child. The implementation of this class is shown 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 send(Person person);
}

class SmsNotification extends Notification {
    private String phone;

    public void sendNotification(Person person) {
        // functionality
    }
}
class EmailNotification extends Notification {
    private String email;

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

Notification and its derived classes

Seating chart, branch, and restaurant

The `SeatingChart` class stores the seating plan for a `Branch` of the `Restaurant`, which can be printed. The `Branch` of a restaurant has a specific name, address, and kitchen. The `Restaurant` class maintains a list of all the branches under it. The definitions of these classes are provided below:

```
public class SeatingChart {
    private int seatingChartID;
    private byte[] seatingChartImage;

    public bool print();
}

public class Branch {
    private String name;
    private Address location;
    private Kitchen kitchen;
    private Menu menu;

    public Address addressOfBranch();
}

public class Restaurant {
    private String name;
    private List<Branch> branches;

    public boolean addBranch(Branch branch);
}
```

The SeatingChart, Branch, and Restaurant classes

Wrapping up

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