Code for the Car Rental System

Write object-oriented code to implement the design of the car rental system problem.

We'll cover the following

- Car rental system classes
  - Enumerations
  - Address, person, and driver
  - Account
  - Vehicle
  - Equipment
  - Service
  - Payment
  - Vehicle log and Vehicle reservation
  - Notification
  - Parking stall and fine
  - Search interface and vehicle catalog
  - Car rental system and car rental branch
- Wrapping up

We've reviewed different aspects of the car rental 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 car rental 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 car rental system:

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

Car rental 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 methods function.

Enumerations

First, we will define all the enumerations required in the car rental system. According to the class diagram, there are seven enumerations used in the system, i.e., `VehicleStatus`, `AccountStatus`, `ReservationStatus`, `PaymentStatus`, `VanType`, `CarType`, and `VehicleLogType`. The code to implement these enumerations is as follows:

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

Java | C# | Python | C++ | JavaScript

```
// definition of enumerations used in the car rental system
enum VehicleStatus {
  AVAILABLE,
  RESERVED,
  LOST,
  BEING_SERVICED
}

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

enum ReservationStatus {
  ACTIVE,
  PENDING,
  CONFIRMED,
  COMPLETED,
  CANCELED
}

enum PaymentStatus {
  UNPAID,
  PENDING,
  COMPLETED,
  CANCELED,
  REFUNDED
}
```

Enum definitions

Address, person, and driver

This section contains the `Address`, `Person`, and `Driver` classes, where the first two classes are used as a custom data type. The implementation of these classes is shown below:se classes can be found below:

Java | C# | Python | C++ | JavaScript

```
public class Address {
```

```
  private String streetAddress;
  private String city;
  private String state;
  private int zipCode;
  private String country;
}

public class Person {
  private String name;
  private Address address;
  private String email;
  private String phoneNumber;
}

public class Driver extends Person {
    private int driverId;
}
```

The Address, Person, and Driver classes

Account

`Account` is an abstract class that represents the various people or actors that can interact with the system. There are two types of accounts: receptionist and customer. The implementation of `Account` and its subclasses is shown below:

Java | C# | Python | C++ | JavaScript

```java
public abstract class Account extends Person {
    private String accountId;
    private String password;
    private AccountStatus status;

    public abstract boolean resetPassword();
}

public class Receptionist extends Account {
    private Date dateJoined;

    public List<Customer> searchCustomer(String name);
    public boolean addReservation();
    public boolean cancelReservation();
    public boolean resetPassword() {
        // definition
    }
}

public class Customer extends Account {
    private String licenseNumber;
    private Date licenseExpiry;

    public boolean addReservation();
    public boolean cancelReservation();
    public List<VehicleReservation> getReservations();
    public boolean resetPassword() {
        // definition
    }
}
```

Account and its derived classes

Vehicle

`Vehicle` will be another abstract class, which serves as a parent for four different types of vehicles: `Car`, `Van`, `Truck`, and `MotorCycle`. The definition of the `Vehicle` and its child classes is given below:

Java | C# | Python | C++ | JavaScript

```java
// Vehicle is an abstract class
public abstract class Vehicle {
  private String vehicleId;
  private String licenseNumber;
  private int passengerCapacity;
  private boolean hasSunroof;
  private VehicleStatus status;
  private String model;
  private int manufacturingYear;
  private int mileage;
  private List<VehicleLog> log;

  public boolean reserveVehicle();
  public boolean returnVehicle();
}

public class Car extends Vehicle {
  private CarType carType;
}

public class Van extends Vehicle {
  private VanType vanType;
}

public class Truck extends Vehicle {
  private TruckType truckType;
}

public class Motorcycle extends Vehicle {
  private MotorcycleType motorcycleType;
}
```

Vehicle and its child classes

Equipment

`Equipment` is an abstract class, and this section represents different equipment: `Navigation`, `ChildSeat`, and `SkiRack` added in the reservation. The code to implement these classes is shown below:

Java | C# | Python | C++ | JavaScript

```java
// Equipment is an abstract class
public abstract class Equipment {
    private int equipmentId;
    private int price;
}

public class Navigation extends Equipment {
}

public class ChildSeat extends Equipment {
}

public class SkiRack extends Equipment {
}
```

Equipment and its derived classes

Service

`Service` is an abstract class, and this section represents different services: `DriverService`, `RoadsideAssistance`, and `Wi-Fi` added to the reservation. The code to implement these classes is shown below:

Java | C# | Python | C++ | JavaScript

```java
// Service is an abstract class
public abstract class Service {
    private int serviceId;
    private int price;
}

public class DriverService extends Service {
    private int driverId;
}

public class RoadsideAssistance extends Service {
}

public class WiFi extends Service {
}
```

Service and its derived classes

Payment

The `Payment` class is another abstract class, with the `Cash` and `CreditCard` classes as its child. This takes in the `PaymentStatus` enum to keep track of the payment status. The definition of this class is provided below:

Java | C# | Python | C++ | JavaScript

```javascript
// Payment is an abstract class
class Payment {
    #amount;
    #timestamp;
    #status;

    // Data members
    constructor(amount, timestamp, status) {
        if (this.constructor === Payment) {
            throw new Error("Abstract classes can't be instantiated.");
        }
        else {
            this.#amount = amount;
            this.#timestamp = timestamp;
            this.#status = status; // Refers to the PaymentStatus enum
        }
    }

    makePayment() { }
}
```

```
}
class Cash extends Payment{
    makePayment(){
        // functionality
    }
}
class CreditCard extends Payment{
    #nameOnCard;
    #cardNumber;
    #billingAddress;
    #code;
```

Payment and its child classes

Vehicle log and Vehicle reservation

`VehicleLog` is a class responsible for keeping track of all the events related to a vehicle. `VehicleReservation` is a class responsible for managing the reservation of vehicles. The implementation of this class is given below:

Java | C# | Python | C++ | JavaScript

```java
public class VehicleLog {
  private int logId;
  private VehicleLogType logType;
  private String description;
  private Date creationDate;
}

public class VehicleReservation {
  private int reservationId;
  private String customerId;
  private String vehicleId;
  private Date creationDate;
  private ReservationStatus status;
  private Date dueDate;
  private Date returnDate;
  private String pickupLocation;
  private String returnLocation;

  private List<Equipment> equipments;
  private List<Service> services;

  public static VehicleReservation getReservationDetails();
  public boolean addEquipment();
  public boolean addService();
}
```

The VehicleLog and VehicleReservation 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 shown below:

Java | C# | Python | C++ | JavaScript

```java
// 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(Account account);
}

class SmsNotification extends Notification {

    public void sendNotification(Account account) {
        // functionality
    }
}

class EmailNotification extends Notification {

    public void sendNotification(Account account) {
        // functionality
    }
}
```

Notification and its derived classes

Parking stall and fine

`ParkingStall` is a class used to locate vehicles in the car rental branch while the `Fine` class represents the fine applied on payment. The implementation of these classes is given below:

Java | C# | Python | C++ | JavaScript

```java
public class ParkingStall {
  private int stallId;
  private String locationIdentifier;
}

public class Fine {
  private double amount;
  private String reason;
  public double calculateFine();
}
```

The Parking stall and fine classes

Search interface and vehicle catalog

`Search` is an interface and the `VehicleCatalog` class is used to implement the search interface to help in vehicle searching. The code to perform this function is presented below:

Java | C# | Python | C++ | JavaScript

```java
public interface Search {
  public List<Vehicle> searchByType(String type);
  public List<Vehicle> searchByModel(String model);
}

public class VehicleCatalog implements Search {
  private HashMap<String, List<Vehicle>> vehicleTypes;
  private HashMap<String, List<Vehicle>> vehicleModels;

  // to return all vehicles of the given type.
  public List<Vehicle> searchByType(String type) {
    // functionality
  }

  // to return all vehicles of the given model.
  public List<Vehicle> searchByModel(String model) {
    // functionality
  }
}
```

The Search interface and the VehicleCatalog class

Car rental system and car rental branch

The `CarRentalSystem` class is the base class of the system that is used to represent the whole car rental system (or the top-level classes of the system). `CarRentalBranch` represents the single branch of the system. The implementation of these classes is given below:

Java | C# | Python | C++ | JavaScript

```java
public class CarRentalBranch {
  private String name;
  private Address address;
  private List<ParkingStall> stalls;

  public Address getLocation();
}

public class CarRentalSystem {
  private String name;
  private List<CarRentalBranch> branches;

  public void addNewBranch(CarRentalBranch branch);

  // The CarRentalSystem is a singleton class that ensures it will have only one active instance at a tim
    private static CarRentalSystem system = null;

    // Created a static method to access the singleton instance of CarRentalSystem class
    public static CarRentalSystem getInstance() {
        if (system == null) {
            system = new CarRentalSystem();
        }
        return system;
    }
}
```

Wrapping up

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

Back

Activity Diagram for the Car Rental System

Next

Getting Ready: The ATM System

Mark as Completed