Code for the Parking Lot

Let's write the code for the classes that we have designed, in different languages in this lesson.

We'll cover the following

- Parking lot classes
    - Enumerations and custom data type
    - Parking spots
    - Vehicle
    - Account
    - Display board and parking rate
    - Entrance and exit
    - Parking ticket
    - Payment
    - Parking lot
- Wrapping up

We've gone over the different aspects of the parking lot 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 parking lot 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 parking lot system:

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

Parking lot classes

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

> **Note**: For simplicity, we aren't 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

First of all, we will define all the enumerations required in the parking lot. According to the class diagram, there are two enumerations used in the system i.e., `PaymentStatus` and `AccountStatus`. The code to implement these enumerations and custom data types 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.

```
// Enumeration
enum PaymentStatus {
  COMPLETED,
  FAILED,
  PENDING,
  UNPAID,
  REFUNDED
}

enum AccountStatus {
  ACTIVE,
  CLOSED,
  CANCELED,
  BLACKLISTED,
  NONE
}

// Custom Person data type class
public class Person {
  private String name;
  private String address;
  private String phone;
  private String email;
}

// Custom Address data type class
public class Address {
  private int zipCode;
  private String address;
  private String city;
```

Definition for the constants

Parking spots

The first section of the parking lot system that we will work on is the `ParkingSpot` class, which will act as a base class for four different types of parking spots: handicapped, compact, large, and motorcycle. This will have an instance of the `Vehicle` class. The definition of the `ParkingSpot` class and the classes being derived from it are given below:

```
// ParkingSpot is an abstract class
public abstract class ParkingSpot {
  private int id;
  private boolean isFree;
  private Vehicle vehicle;

  public boolean getIsFree();
  public abstract boolean assignVehicle(Vehicle vehicle);
  public boolean removeVehicle(){
    // definition
  }
}

public class Handicapped extends ParkingSpot {
  public boolean assignVehicle(Vehicle vehicle) {
    // definition
  }
}

public class Compact extends ParkingSpot {
  public boolean assignVehicle(Vehicle vehicle) {
    // definition
  }
}

public class Large extends ParkingSpot {
  public boolean assignVehicle(Vehicle vehicle) {
    // definition
  }
}
```

ParkingSpot and its derived classes

Vehicle

`Vehicle` will be another abstract class, which serves as a parent for four different types of vehicles: car, truck, van, and motor cycle. The definition of the `Vehicle` and its child classes are given below:

```
// Vehicle is an abstract class
public abstract class Vehicle {
  private int licenseNo;
  public abstract void assignTicket(ParkingTicket ticket);
}

public class Car extends Vehicle {
  public void assignTicket(ParkingTicket ticket) {
    // definition
  }
}

public class Van extends Vehicle {
  public void assignTicket(ParkingTicket ticket) {
    // definition
  }
}

public class Truck extends Vehicle {
  public void assignTicket(ParkingTicket ticket) {
    // definition
  }
}

public class MotorCycle extends Vehicle {
  public void assignTicket(ParkingTicket ticket) {
    // definition
  }
}
```

Vehicle and its child classes

Account

The `Account` class will be an abstract class, which will have the actors, `Admin` and `ParkingAttendant`, as child classes. The definition of these classes is given below:

```
public abstract class Account {
  // Data members
  private String userName;
  private String password;
  private Person person; // Refers to an instance of the Person class
  private AccountStatus status; // Refers to the AccountStatus enum

  public abstract boolean resetPassword();
}

public class Admin extends Account {
  // spot here refers to an instance of the ParkingSpot class
  public boolean addParkingSpot(ParkingSpot spot);
  // displayBoard here refers to an instance of the DisplayBoard class
  public boolean addDisplayBoard(DisplayBoard displayBoard);
  // entrance here refers to an instance of the Entrance class
  public boolean addEntrance(Entrance entrance);
  // exit here refers to an instance of the Exit class
  public boolean addExit(Exit exit);

  // Will implement the functionality in this class
  public boolean resetPassword() {
    // definition
  }
}

public class ParkingAttendant extends Account {
  public boolean processTicket(String ticketNumber);

  // Will implement the functionality in this class
```

Account and its child classes

Display board and parking rate

This section contains the `DisplayBoard` and `ParkingRate` classes that only have the composition class with the `ParkingLot` class. This relationship is highlighted in the `ParkingLot` class. The definition of these classes is given below:

```
public class DisplayBoard {
  // Data members
  private int id;
  private Map<String, List<ParkingSpot>> parkingSpots;

  // Constructor
  public DisplayBoard(int id) {
    this.id = id;
    this.parkingSpots = new HashMap<>();
  }

  // Member function
  public void addParkingSpot(String spotType, List<ParkingSpot> spots);
  public void showFreeSlot();
}

public class ParkingRate {
  // Data members
  private double hours;
  private double rate;

  // Member function
  public void calculate();
}
```

The DisplayBoard and ParkingRate classes

Entrance and exit

This section contains the `Entrance` and `Exit` classes, both of which are associated with the `ParkingTicket` class. The definition of the `Entrance` and `Exit` classes is given below:

```
public class Entrance {
  // Data members
  private int id;

  // Member function
  public ParkingTicket getTicket();
}

public class Exit {
  // Data members
  private int id;

  // Member function
  public void validateTicket(ParkingTicket ticket){
    // Perform validation logic for the parking ticket
    // Calculate parking charges, if necessary
    // Handle the exit process
  }
}
```

The Entrance and Exit classes

Parking ticket

The definition of the `ParkingTicket` class can be found below. This contains instancesof the `Vehicle`, `Payment`, `Entrance` and `Exit` classes:

```
public class ParkingTicket {
    private int ticketNo;
    private Date timestamp;
    private Date exit;
    private double amount;
    private boolean status;

    // Following are the instances of their respective classes
    private Vehicle vehicle;
    private Payment payment;
    private Entrance entrance;
    private Exit exitIns;
}
```

The ParkingTicket class

Payment

The `Payment` class is another abstract class, with the `Cash` and `CreditCard` classes as its child. This takes the `PaymentStatus` enumeration and the `dateTime` data type to keep track of the `payment status` and time. The definition of this class is given below

```java
// Payment is an abstract class
public abstract class Payment {
    private double amount;
    private PaymentStatus status;
    private Date timestamp;

    public abstract boolean initiateTransaction();
}

public class Cash extends Payment {
    public boolean initiateTransaction() {
        // definition
    }
}

public class CreditCard extends Payment {
    public boolean initiateTransaction() {
        // definition
    }
}
```

Payment and its derived classes

Parking lot

The final class of the parking lot system is the `ParkingLot` class which will be a Singleton class, meaning the entire system will only have one instance of this class. The definition of this class is given below:

```java
public class ParkingLot {
    private int id;
    private String name;
    private String address;
    private ParkingRate parkingRate;

    private HashMap<String, Entrance> entrance;
    private HashMap<String, Exit> exit;

    // Create a hashmap that identifies all currently generated tickets using their ticket number
    private HashMap<String, ParkingTicket> tickets;

    // The ParkingLot is a singleton class that ensures it will have only one active instance at a time
    // Both the Entrance and Exit classes use this class to create and close parking tickets
    private static ParkingLot parkingLot = null;

    // Created a private constructor to add a restriction (due to Singleton)
    private ParkingLot() {
        // Call the name, address and parking_rate
        // Create initial entrance and exit hashmaps respectively
    }

    // Created a static method to access the singleton instance of ParkingLot
    public static ParkingLot getInstance() {
        if (parkingLot == null) {
            parkingLot = new ParkingLot();
        }
        return parkingLot;
    }

```

The ParkingLot class

Wrapping up

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

← Back

Activity Diagram for the Parking Lot

☐ Complete

Next →

Getting Ready: Elevator System