Class Diagram for the Car Rental System

Learn to create a class diagram for the car rental system problem using the bottom-up approach.

We'll cover the following

Now, we'll create the class diagram for the car rental system on the basis of the given requirements. In the class diagram, we will first identify classes (concrete, abstract, or associated) and interfaces for the system. Then, we will determine the relationship between them, according to the requirements in the previous lesson.

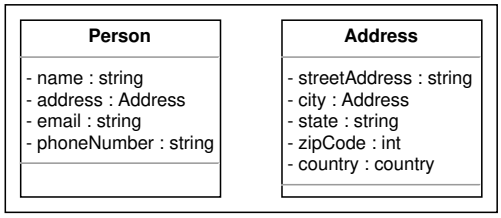
Components of a car rental system

As mentioned earlier, we'll design the car rental system using a bottom-up approach.

Address and person

The `Address` is a custom data type that is required to store any address. The `Address` contains attributes like a street address, city, state, etc. In the car rental system, this class will be used to specify the address of any person or a car rental location or branch. The `Person` class stores information related to a person like a name, email,

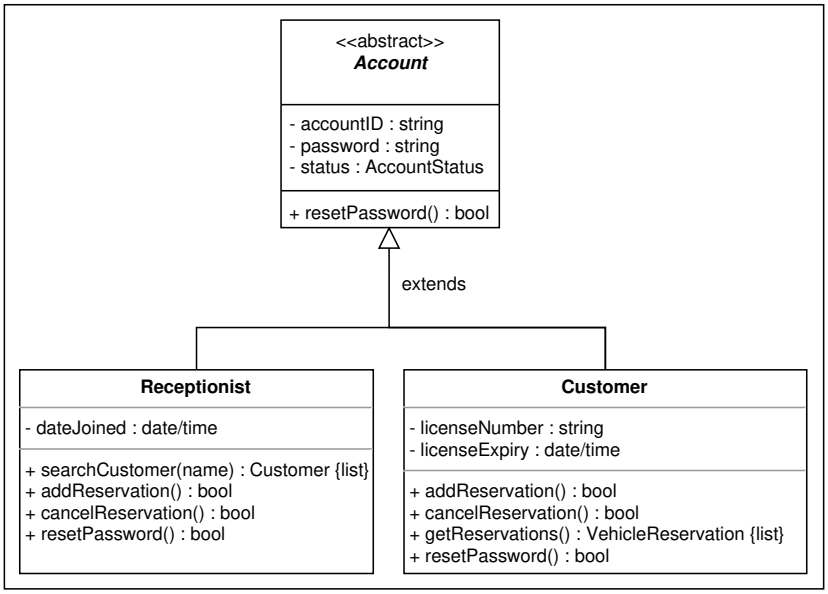
phone number, and address. In the `Person` class, there is an object of the `Address` type to specify the person's address. The class representation of `Address` and `Person` is given below:



The class diagram of the Address and Person classes

Account

`Account` is an abstract class that is used to store the account information of a person. This class has members like account ID, password, the status of an account, etc. There can be two types of accounts, i.e., customer and receptionist. The `Customer` class represents the customers who reserve the vehicle for themselves, while the `Receptionist` class represents the receptionist in the car rental system. Both of them can create any vehicle reservation and can cancel the reservation as well. The class representation of `Account` and its subclasses is given below:



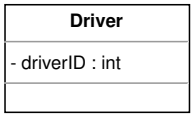
The class diagram of the Account class

R1: Car Rental System

R1: There can be two types of users in the car rental system, i.e., customers and receptionists.

Driver

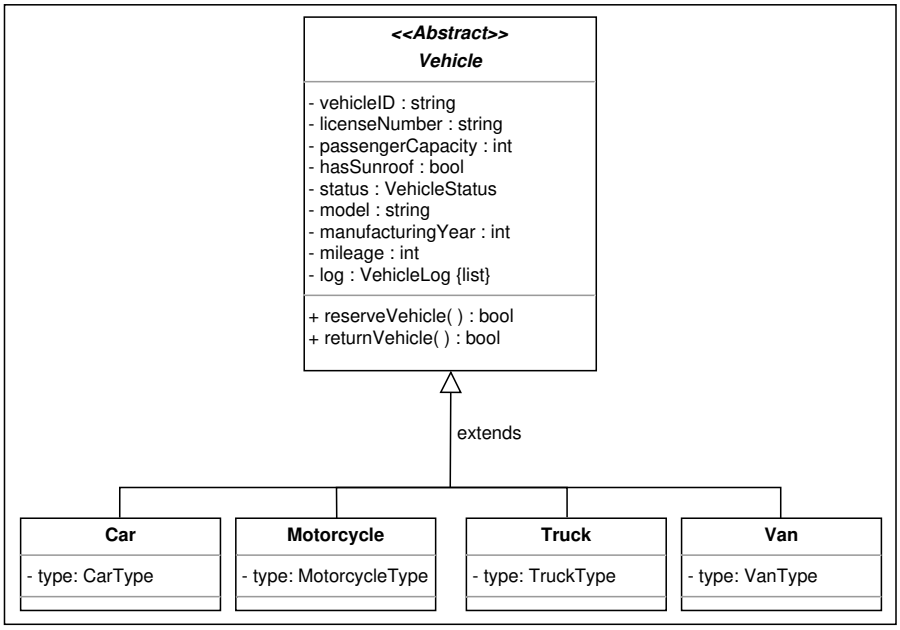
Since we are designing the car rental problem, we will have a `Driver` class. A customer can request an additional driver at the time of reservation. The class diagram is shown below:



The class diagram of the Driver class

Vehicle

Our car rental system should have a vehicle object according to the requirements. The vehicle can be of four types: a car, truck, van, and motorcycle. For this purpose, we'll create `Vehicle` as an abstract class and `Car`, `Truck`, `Van`, and `Motorcycle` as its subclasses, as shown in the figure below:



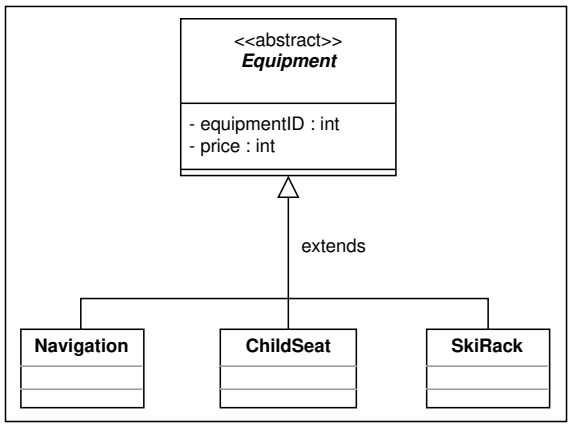
The class diagram of Vehicle and its derived classes

R2: Car Rental System

R2: The system should handle multiple types of vehicles. Initially, the system should cater to the following vehicles: cars, trucks, vans, and motorcycles.

Equipment

`Equipment` is an abstract class that stores information about different types of equipment that can be added to the reservation. For simplicity, we'll assume three types of equipment, i.e., navigation, child seat, and ski rack. The class diagram for `Equipment` and its subclasses is as follows:



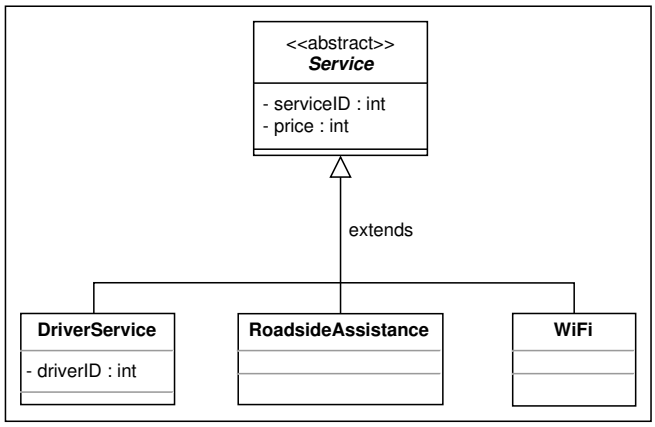
The class diagram of Equipment and its derived classes

R8: Car Rental System

R8: The system should allow the users to add equipment to the reservations like a ski rack, child seat, and navigation.

Service

`Service` is an abstract class that represents the services provided to the customers along with the vehicle. While reserving a vehicle, the customers can add a service to their reservation. Every service has its fixed cost. We have three types of services, i.e., driver, roadside assistance, and Wi-Fi. The UML diagram of `Service`, along with its subclasses `DriverService`, `RoadsideAssistance`, and `Wi-Fi`, is given below:



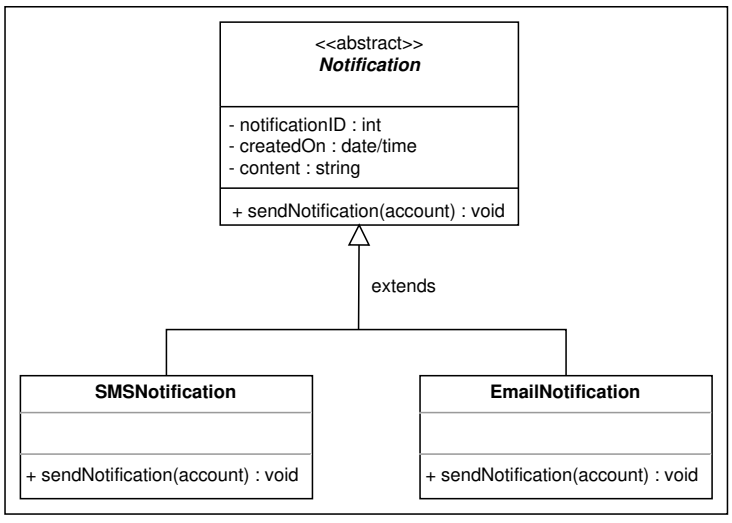
The class diagram of Service and its derived classes

R9: Car Rental System

R9: The system should allow the users to add services to the reservations like a driver, Wi-Fi, and roadside assistance.

Notification

`Notification` is an abstract class responsible for sending notifications to customers. Every notification has an ID, creation date, and content in it. The notification can either be an SMS notification or an email notification. The `SMSNotification` class requires the phone number of the customer to send a notification, while `EmailNotification` is sent to the email address of the customer. The relationship diagram of these classes is shown below:



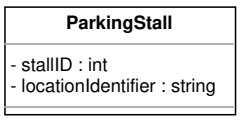
The class diagram of Notification and its derived classes

R10: Car Rental System

R10: The system should send a notification to the customer and generate a fine if the vehicle is not returned within the due date.

Parking stall

Each car rental location has parking stalls where the vehicles are parked. Each parking stall is identified by its ID and its location is specified by a location identifier. The representation of the `ParkingStall` class is shown below:



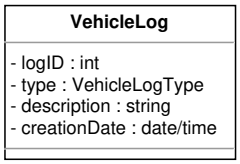
The class diagram of the ParkingStall class

R13: Car Rental System

R13: Every branch of the car rental system should have parking stalls to park the vehicles.

Vehicle log

`VehicleLog` is a class that is used to keep track of all the events related to a vehicle. Every vehicle log has its ID, log type, description, and creation date, as shown here:



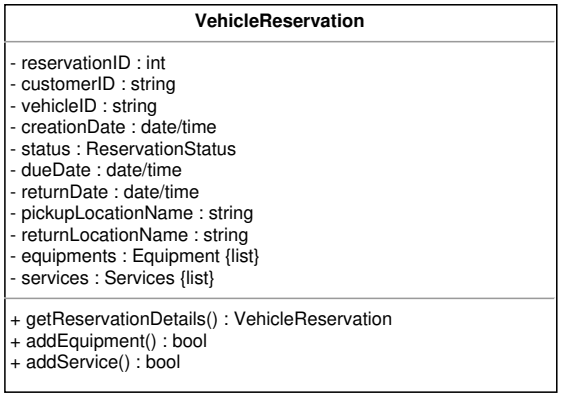
The class diagram of the VehicleLog class

R7: Car Rental System

R7: To keep track of all events related to the vehicle, the system should maintain a vehicle log.

Vehicle reservation

Vehicle reservation is one of the most important requirements of the car rental system. To fulfill this functionality, we have a `VehicleReservation` class. This class is responsible for managing the vehicle reservation status of vehicles. The customer can add any equipment or service at the time of reservation as well. The UML representation of the class is shown below:



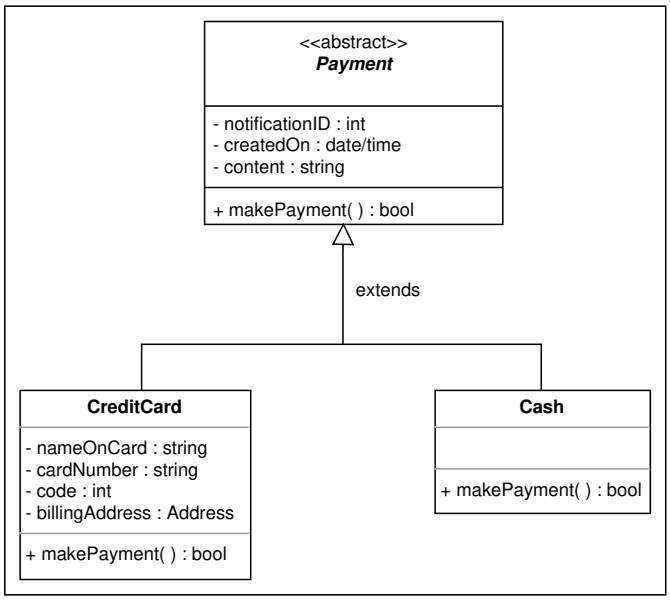
The class diagram of the VehicleReservation class

R4: Car Rental System

R4: The system should be able to keep a record of who reserved a particular vehicle and on which date the vehicle was issued.

Payment

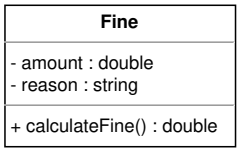
The `Payment` class will be an abstract class and will have two child classes: `CreditCard` and `Cash`. These represent the two payment methods in the car rental system. The representation of these classes is given below:



The class diagram of Payment and its child classes

Fine

The system needs the `Fine` class to calculate the fine on the vehicle reservation in case the customer returns the vehicle after the due date, the fuel in the vehicle is less than the limit value, or there is any damage to the vehicle. The representation of this class is given below:



The class diagram of the Fine class

R10: Car Rental System

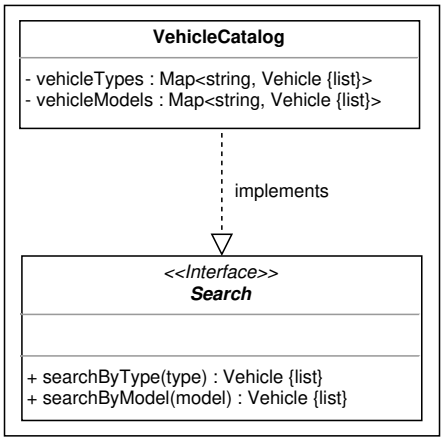
R10: The system should send a notification to the customer and generate a fine if the vehicle is not returned within the due date.

Search interface and vehicle inventory class

`Search` is one of the most important functionalities of the system. It is the interface that allows the user to search for any vehicle and return the list of vehicles upon searching by any of the following methods:

- Search car by its type
- Search car by its model

The `VehicleCatalog` is a class where the search function is implemented. In each catalog, the vehicles are sorted according to one of the given search techniques, i.e., either the vehicle type or model. The following UML diagram shows this relationship:



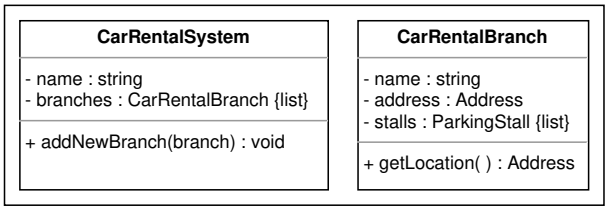
The class diagram of Search interface and VehicleCatalog class

R11: Car Rental System

R11: The system should allow the user to search vehicles by type, model, or seat capacity.

Car rental system and branch

`CarRentalSystem` is the main class of the car rental system and is the central part of the design. There can be multiple branches and locations of the car rental system. The `CarRentalBranch` class will represent each of these branches. The class representation is as follows:



The class diagram of CarRentalSystem and CarRentalBranch classes

R12: Car Rental System

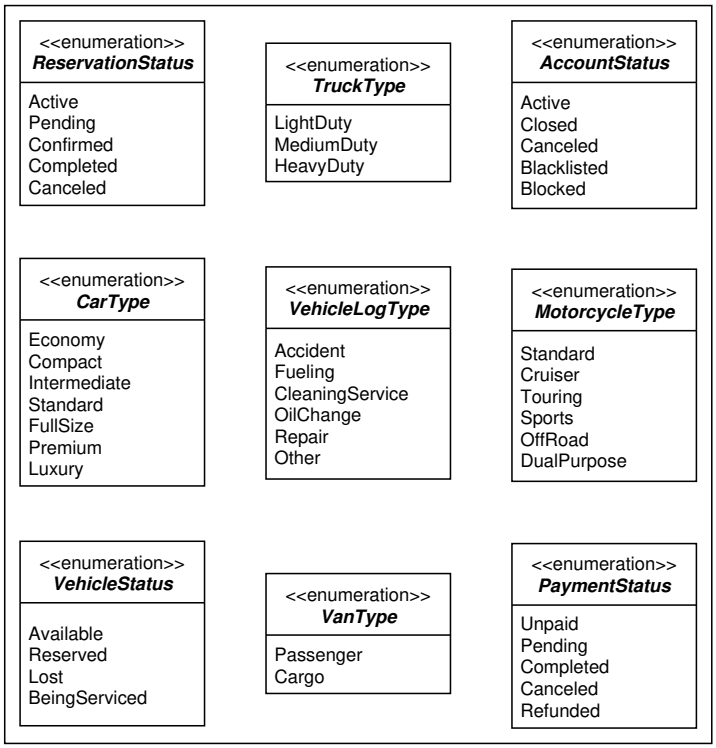
R12: A system should be able to manage the multiple branches of the car rental system.

Enumerations

The list of enumerations required in the car rental system is provided below:

- `VehicleStatus`: The vehicle status describes the status of the particular vehicle for the user, whether it is available, reserved, lost, or being serviced.
- `AccountStatus`: The account status tells about the user account status, i.e., active, closed, canceled, banned, or blocked.
- `ReservationStatus`: The reservation status tells about the reservation state of any vehicle, whether it is in an active state, pending state, confirmed state, completed state, or canceled state.
- `PaymentStatus`: The payment status checks if the customer's payment falls in any of the following stages: unpaid, pending, completed, canceled, or refunded.
- `VanType`: The van type specifies that the van can only be of two types, i.e, passenger or cargo.ifies that the van can only be of two types, i.e, passenger or cargo.
- `CarType`: The car type tells about the different types of cars, whether it is economy, compact, intermediate, standard, full size, premium, or luxury.
- `MotorcycleType`: Similar to the car type, the motorcycle type tells about the different types of motorcycles, whether it is standard, cruiser, touring, sports, off-road, or dual purpose.
- `TruckType`: The truck type specifies that the truck can be of three types, i.e, light-duty, medium-duty, or heavy-duty.
- `VehicleLogType`: The vehicle log type describes the type of a particular log of a vehicle, whether it is an accident, fueling, cleaning service, oil change, repair, or other.

These enumerations can be represented using the following class diagram:



R3: Car Rental System

R3: There can be multiple subtypes for vehicles. The car type can be economy, luxury, standard, and compact. The van type can be passenger or cargo type. Moreover, the motorcycle type can be cruiser, touring or sports. The truck type can be light, medium, or high-duty.

Relationship between the classes

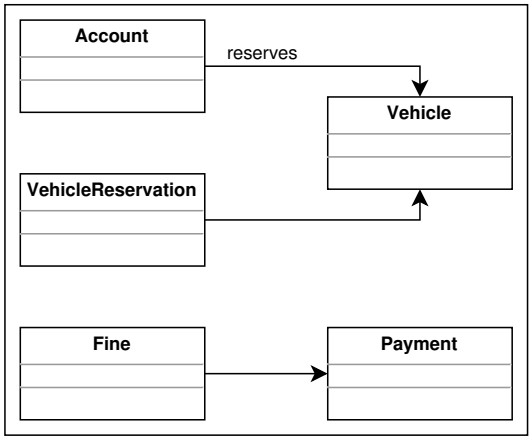
Now, we'll discuss the relationships between the classes we have defined above in our car rental system.

Association

The class diagram has the following association relationships.

One-way association

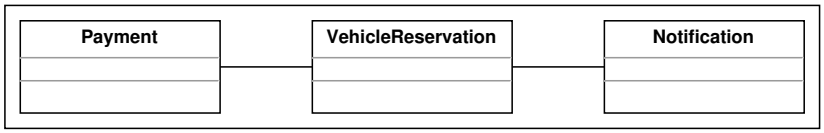
- Both the `Account` and `VehicleReservation` classes have a one-way association with the `Vehicle` class.
- The `Fine` class has a one-way association with `Payment`.



The one-way association relationship between classes

Two-way association

- The `VehicleReservation` class has a two-way association with `Payment` and `Notification`.

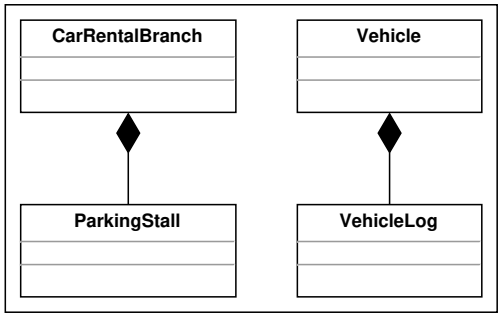


The two-way association relationship between classes

Composition

The class diagram has the following composition relationships:

- The `CarRentalBranch` class is composed of the `ParkingStall` class.
- The `Vehicle` class is composed of the `VehicleLog` class.

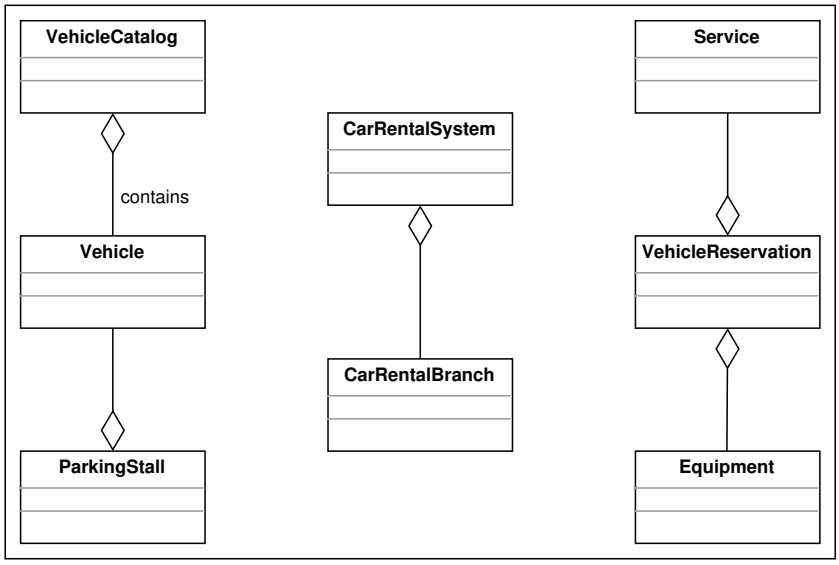


The composition relationship between classes

Aggregation

The following classes show an aggregation relationship:

- The `CarRentalSystem` class contains the `CarRentalBranch` class.
- Both the `ParkingStall` and `VehicleCatalog` classes consist of the `Vehicle` class.
- The `VehicleReservation` class has an aggregation relationship with the `Equipment` and `Service` classes.

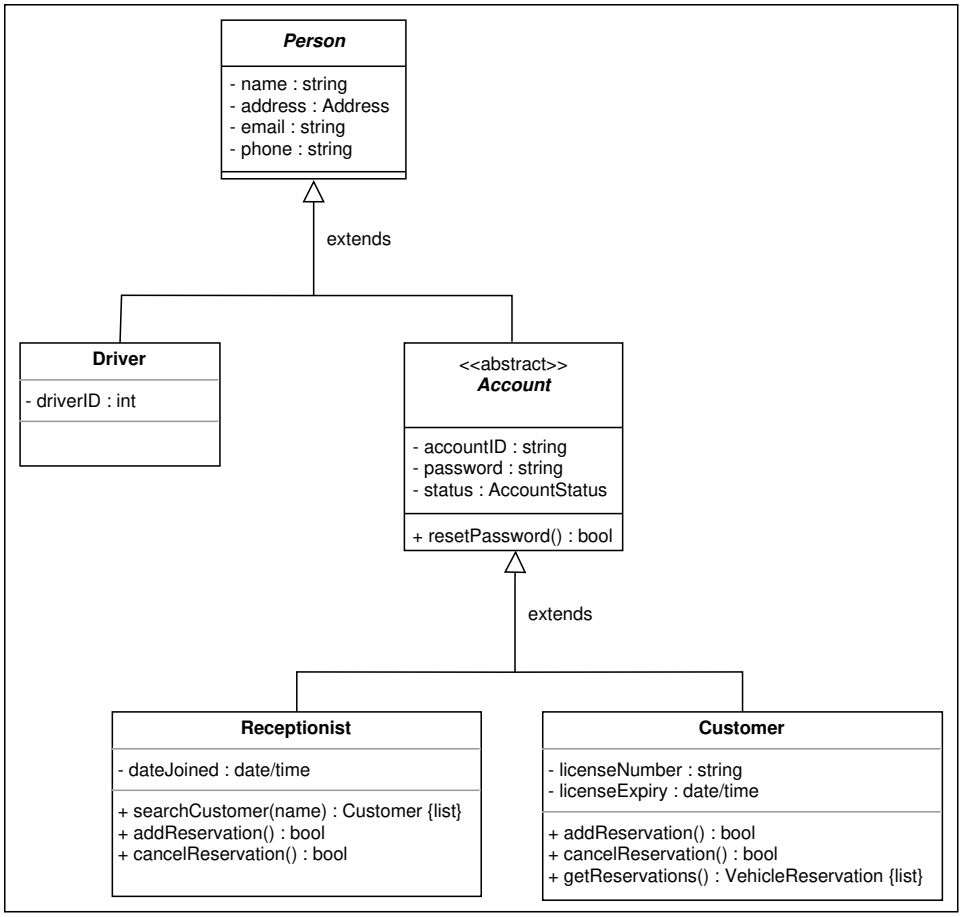


The aggregation relationship between classes

Inheritance

The following classes show an inheritance relationship:

- Both the `Receptionist` and `Customer` classes extend the `Account` class. Whereas, `Account` and `Driver` extend the `Person` class.



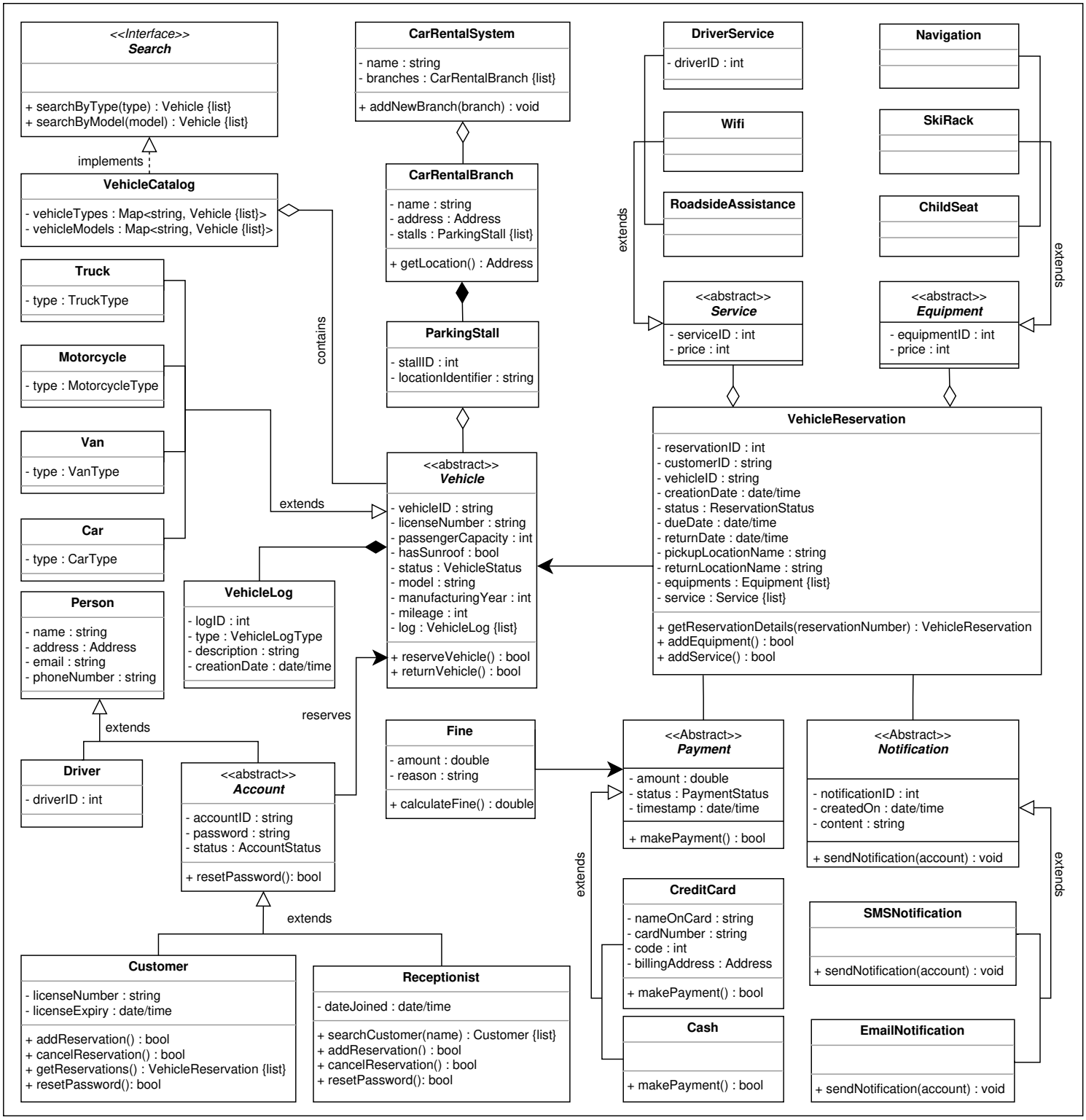
The class diagram of different users of our system

- The `Car`, `Truck`, `Van`, and `Motorcycle` classes extend the `Vehicle` class.
- The `Driver`, `RoadsideAssistance`, and `Wi-Fi` classes extend the `Service` class.
- The `Navigation`, `ChildSeat`, and `SkiRack` classes extend the `Equipment` class.
- Both the `SMSNotification` and `EmailNotification` classes extend the `Notification` class.
- Both the `Cash` and `CreditCard` classes extend the `Payment` class.
- The `VehicleCatalog` class implements a `Search` interface.

> **Note:** We've already discussed the inheritance relationship between classes in the component section above one by one.

Class diagram of the car rental system

Here's the class diagram of the car rental system:



The class diagram of the car rental system

Design pattern

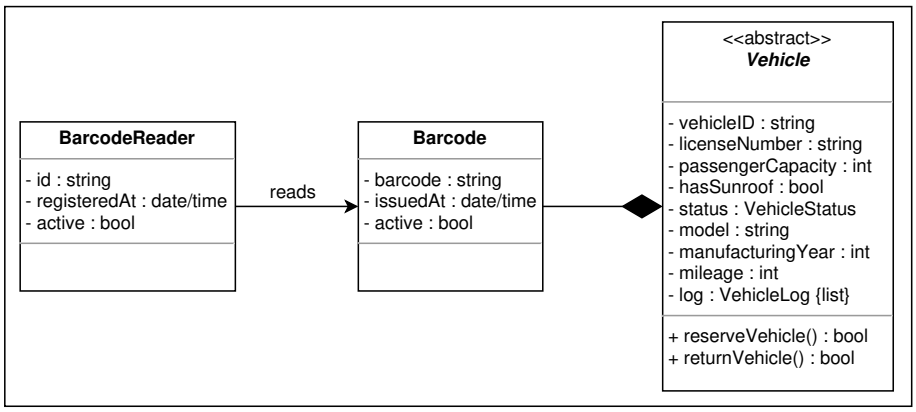
We can use the Decorator design pattern for our car rental system. We can design it using the following decorators:

- **Discount decorator:** It can be used to apply discounts to all types of vehicles in our car rental system.
- **Peak season decorator:** It can be used to increase the fare of all types of vehicles in our car rental system.
- **Damage fine decorator:** When the vehicle is returned, this decorator can help in calculating the fine due to car damage.
- **Partially filled fuel tank fine decorator:** When the vehicle is returned, this decorator can help calculate the fine due to the partially filled fuel tank.

Similarly, we can make several other decorators according to the system needs. These decorator fulfill the SRP and OCP design principles.

Additional requirements

The interviewer can introduce some additional requirements in the car rental system, or they can ask some follow-up questions. Let's see an example of additional requirements:
Barcode Scanner: Each vehicle should have a unique barcode associated with it, and the system should be able to scan the barcode of every vehicle. To fulfill this requirement, we have the class diagram as shown below:



The class diagram of barcode scanner functionality

We have completed the class diagram of the car rental system according to the requirements. Now let’s design the sequence diagram of the system in the next lesson.

Back

Use Case Diagram for ...

Next

Sequence Diagram for...

Mark as Completed