**Getting Ready: Parking Lot**

Let's make an object-oriented design for a multi-entrance and exit parking lot system.

**We'll cover the following**

* [Problem definition](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Problem-definition)
* [Expectations from the interviewee](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Expectations-from-the-interviewee)
  + [Payment flexibility](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Payment-flexibility)
  + [Parking spot type](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-spot-type)
  + [Vehicle types](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Vehicle-types)
  + [Pricing](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Pricing)
* [Design approach](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Design-approach)
* [Design pattern](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Design-pattern)

**Problem definition**

A **parking lot** is a designated area for parking vehicles and is a feature found in almost all popular venues such as shopping malls, sports stadiums, offices, etc. In a parking lot, there are a fixed number of parking spots available for different types of vehicles. Each of these spots is charged according to the time the vehicle has been parked in the parking lot. The parking time is tracked with a parking ticket issued to the vehicle at the entrance of the parking lot. Once the vehicle is ready to exit, it can either pay at the automated exit panel or to the parking agent at the exit using a card or cash payment method.

A screenshot of a parking lot

Description automatically generated

* How are customers able to pay at different exit points (i.e., either at the automated exit panel or to the parking agent) and by different methods (cash, credit, coupon)?
* If there are multiple floors in the parking lot, how will the system keep track of the customer having already paid on a particular floor rather than at the exit?

### Parking spot type

Another topic of discussion that an interviewer would expect you to be aware of is the different parking spot types—handicapped, compact, large, and motorcycle—regarding which you can ask the following questions:

* How will the parking capacity of each lot be considered?
* What happens when a lot becomes full?
* How can one keep track of the free parking spots on each floor if there are multiple floors in the parking lot?
* How will the division of the parking spots be carried out among the four different parking spot types in the lot?

### Vehicle types

Similar to the parking spot, an interviewer would also expect you to discuss the different vehicle types—car, truck, van, motorcycle—which can have the following set of questions:

* How will capacity be allocated for different vehicle types?
* If the parking spot of any vehicle type is booked, can a vehicle of another type park in the designated parking spot?

### Pricing

We touched upon the payment structure offered by the parking lot system. Now, the pricing model needs to be clarified from the interviewer, and therefore you may ask questions like these:

* How will pricing be handled? Should we accommodate having different rates for each hour? For example, customers will have to pay $4$4 for the first hour, $3.5$3.5 for the second and third hours, and $2.5$2.5for all the subsequent hours.
* Will the pricing be the same for the different vehicle types?

## Design approach

We are going to design this parking lot system using the bottom-up design approach. For this purpose, we will follow the steps below:

* Identify and design the smallest components first, like, the vehicle and parking spot types.
* Use these small components to design bigger components, for example, the payment system at the exit.
* Repeat the steps above until we design the whole system like the parking lot.

## Design pattern

It is always a good practice to discuss the design patterns that a parking lot system falls under, during the interview. Stating the design patterns will give the interviewer a positive impression and shows that the interviewee is well-versed in the advanced concepts of object-oriented design. The following design patterns can be used to design the parking lot system:

* Singleton design pattern
* Abstract Factory design pattern
* Factory design pattern

Let’s explore the requirements of the parking lot system in the next lesson.

Back

**Requirements for the Parking Lot Design**

Learn about all requirements of the parking lot problem.

**We'll cover the following**

* [Requirements collection](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Requirements-collection)

In this lesson, let’s list down the requirements of the parking lot system. This is a very crucial step since requirements define the scope of a problem, so getting them right from the interviewer and understanding them well will make the design of the rest of the system smooth and easy.

We’ll use the notational convention to identify each requirement with a unique label "Rn", where "R" is short for Requirement and "n" is a natural number.

A screenshot of a parking lot

Description automatically generated

A screenshot of a parking ticket

Description automatically generated

# Use Case Diagram for the Parking Lot

Learn how to define use cases and create the corresponding use case diagram for the parking lot system.

**We'll cover the following**

* [System](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#System)
* [Actors](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Actors)
  + [Primary actors](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Primary-actors)
  + [Secondary actors](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Secondary-actors)
* [Use Cases](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Use-Cases)
  + [Admin](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Admin)
  + [Customer](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Customer)
  + [Parking agent](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-agent)
  + [System](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#System)
* [Relationships](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Relationships)
  + [Generalization](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Generalization)
  + [Associations](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Associations)
  + [Include](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Include)
* [Use case diagram](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Use-case-diagram)

Let’s build the use case diagram of the parking lot system and understand the relationship between its different components.

First, let’s define the different elements of our parking lot, followed by the complete use case diagram of the system.

## System

Our system is a "parking lot."

## Actors

Here are the main actors of our parking lot system.

### Primary actors

* **Customer:**This actor can park the vehicle in the allocated parking space according to the vehicle type and pay for the parking before exit.
* **Parking agent:** The parking agent will assist the customer and perform all the tasks that a customer can do, such as paying the parking ticket on behalf of the customer.

### Secondary actors

* **Admin:** This can add, remove, or update a spot, agent, entry/exit panels, and view/update accounts.
* **System:** This is responsible for giving details of parking spot availability and assigning a parking spot to a vehicle.

## Use Cases

In this section, we will define the use cases for the parking lot. We have listed down the use cases according to their respective interactions with a particular actor.

**Note:**You will see some use cases occurring multiple times because they are shared among different actors in the system.

### Admin

* **Add spot:**To add a parking spot
* **Add agent:**To add a new agent
* **Add/modify rate:**To add/modify hourly rate
* **Add entry/exit panel:**To add and update exit/entry panel at each entry/exit
* **Update account:**To update account details and payment information
* **Login/Logout:**To login/logout to/from agent or admin account
* **View account:**To view account details like payment status or unpaid amount

### Customer

* **Take ticket:**To take a ticket at the entrance, that contains information regarding the vehicle and its entrance time
* **Scan ticket:**To scan the ticket at the exit and get the parking fee
* **Pay ticket:**To pay the parking fee at the exit panel via cash or a credit card
* **Cash:**To pay the parking fee via cash
* **Credit card:**To pay the parking fee via credit card
* **Park vehicle:**To park the vehicle at the assigned destination

### Parking agent

* **Update account:**To update account details and payment information
* **Login/Logout:**To log in/log out to/from the agent or admin account
* **View account:**To view account details like payment status or unpaid amount
* **Take ticket:**To take a ticket at the entrance, that contains information regarding the vehicle and its entrance time
* **Scan ticket:**To scan the ticket at the exit and get the parking fee
* **Pay ticket:**To pay the parking fee at the exit panel via cash or a credit card
* **Cash:**To pay the parking fee via cash
* **Credit card:**To pay the parking fee via credit card
* **Park vehicle:**To park the vehicle at the assigned destination

### System

* **Assigning parking spots to vehicles:**To check the vehicle type and associate a free spot according to it
* **Remove spot:**To remove a parking spot if it is not available for parking
* **Show full:** To display the status of the parking lot as full
* **Show available:** To show the details of available parking spots

## Relationships

This section describes the relationships between and among actors and their use cases.

### Generalization

* The “Parking agent” has a generalization relationship with the “Customer” since the parking agent can perform all those tasks that a customer can perform.
* “Cash” and “Credit card” use cases are used for payments. Hence, both have a generalization relationship with the “Pay ticket” use case.

### Associations

The table below shows the association relationship between actors and their use cases.

|  |
| --- |
|  |

A screenshot of a car service

Description automatically generated

A diagram of a parking system

Description automatically generated

In the next lesson, we will discuss the class diagram with a detailed explanation of all classes and their relationship with each other.

# Class Diagram for the Parking Lot

Learn to create a class diagram for the parking lot system using the bottom-up approach.

**We'll cover the following**

* [Components of a parking lot system](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Components-of-a-parking-lot-system)
  + [Vehicle](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Vehicle)
    - [Enumeration vs. abstract class](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Enumeration-vs.-abstract-class)
  + [Parking spot](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-spot)
  + [Account](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Account)
  + [Display board](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Display-board)
  + [Entrance and exit](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Entrance-and-exit)
  + [Parking ticket](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-ticket)
  + [Payment](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Payment)
  + [Parking rate](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-rate)
  + [Parking lot](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-lot)
  + [The enumerations and custom data types](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#The-enumerations-and-custom-data-types)
    - [Address](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Address)
  + [Person](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Person)
* [Relationship between the classes](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Relationship-between-the-classes)
  + [Association](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Association)
  + [Composition](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Composition)
  + [Inheritance](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Inheritance)
* [Class diagram of the parking lot system](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Class-diagram-of-the-parking-lot-system)
* [Design pattern](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Design-pattern)
* [Additional requirements](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Additional-requirements)

In this lesson, we will identify and design classes, abstract classes, and interfaces based on the requirements we have previously gathered from the interviewer in our parking lot system.

## Components of a parking lot system

As mentioned earlier, we should design the parking lot system using a bottom-up approach. Therefore, we will first identify and design the classes of the smaller components like vehicles and parking spots. Then, we will create the class of the entire parking lot system, including these smaller components.

### Vehicle

Our parking lot system should have a vehicle object according to the requirements. The vehicle can be a car, a truck, a van, and a motorcycle. There are two ways to represent a vehicle in our system:

* Enumeration
* Abstract class

#### Enumeration vs. abstract class

The**enumeration** class creates a user-defined data type that has the four vehicle types as values.

This approach is not proficient for object-oriented design because if we want to add one more vehicle type later in our system, then we would need to update the code in multiple places in our code, and this would violate the Open Closed principle of the SOLID design principle. This is because the Open Closed principle states that classes can be extended but not modified. Therefore, it is recommended not to use the enumeration data type as it is not a scalable approach.

**Note:** Using enums isn’t prohibited, but just not recommended. Later, we will use the PaymentStatusenum in our parking lot design as it won’t require further modifications.

An **abstract class** cannot instantiate the object and can only be used as a base class. The abstract class for Vehicle is the best approach. It allows us to create derived child classes for the Vehicle class. It can be extended easily in case the vehicle type changes in the future.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a parking ticket

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a parking lot

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer screen

Description automatically generated

A screenshot of a diagram

Description automatically generated

The following classes show an inheritance relationship:

* The Vehicle class includes Car, Truck, Van, and MotorCycle subclasses.
* The ParkingSpot class includes handicapped, compact, large, and motorcycle subclasses.
* The Payment class includes the Cash and CreditCard subclasses.

**Note:**We have already discussed the inheritance relationship between classes in the component section above.

## Class diagram of the parking lot system

Here is the complete class diagram for our parking lot system:

A diagram of parking lot

Description automatically generated

A screenshot of a diagram

Description automatically generated

A screenshot of a computer

Description automatically generated

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

Back

Use Case Diagram for the Parking Lot

**Sequence Diagram for the Parking Lot**

Create a sequence diagram for the card payment in the parking lot system and solve a challenge.

**We'll cover the following**

* [Card payment](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Card-payment)
* [Sequence challenge: Payment verification](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Sequence-challenge:-Payment-verification)

A sequence diagram is a great way to understand the interactions between different entities and objects in the system. There can be different sequence diagrams that we can create for our parking lot system. For the sake of this lesson, we will create sequence diagrams for the following two interactions:

* **Card payment:** This performs a payment using the card.
* **Sequence challenge:** This is for payment verification.

**Card payment**

The sequence diagram for the card payment should have the following actors and objects that will interact with each other:

* **Actor:** Customer
* **Object:**CardReader, Payment, and ExitPanel

Here are the steps in the card payment interaction:

1. The customer inserts the card into the card reader.
2. The card reader initiates a payment for the required parking fee.
3. The payment processes the payment and returns the payment status.
4. The card reader ejects the card.
5. If the payment is successful:
   1. The customer requests a receipt for the transaction.
   2. The exit panel prints a receipt for the customer.
6. If the payment is unsuccessful:
   1. The customer sees an error message for an unsuccessful Payment.

**Note:** The Payment object is created when a vehicle enters the parking lot.

Based on the order above, the sequence diagram of the card payment in a parking lot system is given below.

A screenshot of a computer program

Description automatically generated

## Sequence challenge: Payment verification

In this section, you will help us in completing a sequence diagram for the payment verification of a customer at the exit panel.

The skeleton below represents the payment verification sequence diagram. Here the payment status of the ticket is currently unpaid and the parking fee has to be calculated.

A diagram of a payment verification

Description automatically generated

# Activity Diagram for the Parking Lot

Create some activity diagrams for the parking lot problem.

**We'll cover the following**

* [Vehicle entering the parking lot](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Vehicle-entering-the-parking-lot)
  + [States](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#States)
  + [Actions](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Actions)
* [Activity challenge: The customer pays the parking ticket](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Activity-challenge:-The-customer-pays-the-parking-ticket)

An activity diagram is a great way to visualize the flow of messages from one activity to the other in the system. There can be different activity diagrams that we can create for our parking lot system. For this lesson, we will create activity diagrams for the following two activities:

* The vehicle entering the parking lot
* **Activity challenge:**Customer pays the parking ticket

## Vehicle entering the parking lot

The following are the states and actions that will be involved in this activity diagram.

### States

**Initial state:**The customer enters the parking lot.

**Final state:**There are two final states present in this activity diagram, shown below:

* The customer receives the parking ticket through the system.
* There is no parking slot vacant, so the customer is denied access to the parking lot.

### Actions

The customer arrives at the parking lot entrance and selects their vehicle type. They are assigned their dedicated parking spot according to their select vehicle type. The parking lot then informs us about the availability of that parking spot and allows access accordingly.

Based on the order above, the activity diagram of a vehicle entering a parking lot is given below.

**Note:** Here we assume that only a car can be parked in the handicap spot. Access is only available if the car has a disabled person card present.

A diagram of a parking ticket

Description automatically generated

## Activity challenge: The customer pays the parking ticket

You will help us create an activity diagram of a customer paying a parking ticket at the exit panel.

The skeleton of the activity diagram below represents that the customer has a valid parking ticket available.

A diagram of a vehicle

Description automatically generated

Notice that the actions in the diagram above are numbered from 1 to 11. The slots shown below represent the activities and the arrows represent the flow from one activity to the other. Can you rearrange the slots below in the correct order they should appear in the activity diagram above?

Based on the description above, can you fill in the missing slots with the correct order of actions in the activity diagram?

**Note:**If you get stuck, just click the “Show Solution” button for the correct answer.

Alternatively, you can also click the "Show complete diagram" button below to see the complete sequence diagram.

Show complete diagram

We've looked at some of the activity diagrams of our parking lot system. In the next lesson, we will present the code for our designed classes in some of the most popular languages.

# 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](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-lot-classes)
  + [Enumerations and custom data type](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Enumerations-and-custom-data-type)
  + [Parking spots](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-spots)
  + [Vehicle](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Vehicle)
  + [Account](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Account)
  + [Display board and parking rate](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Display-board-and-parking-rate)
  + [Entrance and exit](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Entrance-and-exit)
  + [Parking ticket](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-ticket)
  + [Payment](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Payment)
  + [Parking lot](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#Parking-lot)
* [Wrapping up](https://www.educative.io/order-confirmation/stripe/subscription-buy?payment_intent=pi_3O1FFeKhXp6R50hI1xakUX3g&payment_intent_client_secret=pi_3O1FFeKhXp6R50hI1xakUX3g_secret_EDla1wT2fgsm34I6up6bRJM7s&transaction_id=1d4a8dd5-516d-4906-aa95-9e9effdf447d#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.

Java

// 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;

private String state;

private String country;

}

### 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

}

}

public class Motorcycle extends ParkingSpot {

public boolean assignVehicle(Vehicle vehicle) {

// definition

}

}

### 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

}

}

### 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

public boolean resetPassword() {

// definition

}

}

### 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 List<Handicapped> handicappedSpot;

private List<Compact> compactSpot;

private List<Large> largeSpot;

private List<MotorCycle> motorCycleSpot;

// Member function

public void showFreeSlot();

}

public class ParkingRate {

// Data members

private double hours;

private double rate;

// Member function

public void calculate();

}

### Entrance and exit

This section contains the Entrance and Exit classes, both of which are associated with the ParkingTicketclass. 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

}

}

### 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;

}

### 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

// 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

}

}

### 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:

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, parking\_rate elements of the customer in the parking lot from the database

// 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;

}

public boolean addEntrance(Entrance entrance){}

public boolean addExit(Exit exit){}

// This function allows parking tickets to be available at multiple entrances

public ParkingTicket getParkingTicket(Vehicle vehicle) {}

public boolean isFull(ParkingSpot type){}

}

## 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.