

# Diagrama de classe para o estacionamento

Aprenda a criar um diagrama de classes para o sistema de estacionamento usando a abordagem ascendente.

## Abordaremos o seguinte

- Componentes de um sistema de estacionamento
  - Veículo
    - Enumeração vs. classe abstrata
  - Vaga de estacionamento
  - Conta
  - placa de exibição
  - entrada e saída
  - Bilhete de estacionamento
  - Pagamento
  - Taxa de estacionamento
  - Estacionamento
  - As enumerações e tipos de dados personalizados
    - Endereço
  - Pessoa



- Relação entre as classes
  - Associação
  - Composição
  - Herança
- Diagrama de classe do sistema de estacionamento
- Padrão de design
- Requisitos adicionais

Nesta lição, identificaremos e projetaremos classes, classes abstratas e interfaces com base nos requisitos que coletamos anteriormente do entrevistador em nosso sistema de estacionamento.

## Componentes de um sistema de estacionamento

Conforme mencionado anteriormente, devemos projetar o sistema de estacionamento usando uma abordagem de baixo para cima. Portanto, primeiro identificaremos e projetaremos as classes dos componentes menores, como veículos e vagas de estacionamento. Em seguida, criaremos a classe de todo o sistema de estacionamento, incluindo esses componentes menores.

### Veículo

Nosso sistema de estacionamento deve ter um objeto de veículo de acordo com os requisitos. O veículo pode ser um carro, um caminhão, uma van e uma motocicleta. Existem duas formas de representar um veículo em nosso sistema:

- Enumeração
- classe abstrata



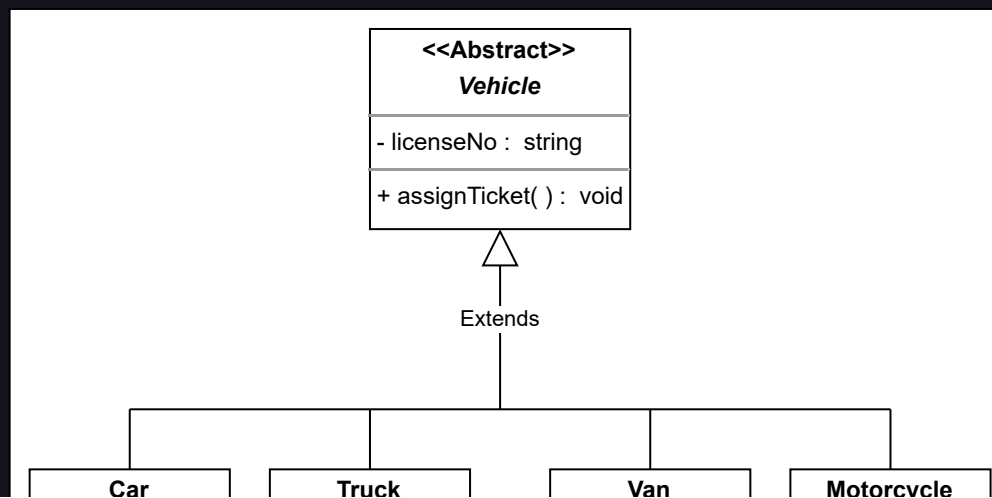
## Enumeração vs. classe abstrata

A classe **de enumeração** cria um tipo de dados definido pelo usuário que possui os quatro tipos de veículos como valores.

Essa abordagem não é proficiente para projeto orientado a objetos porque, se quisermos adicionar mais um tipo de veículo posteriormente em nosso sistema, precisaremos atualizar o código em vários lugares em nosso código, e isso violaria o princípio Aberto Fechado do Princípio de design SÓLIDO. Isso ocorre porque o princípio Aberto Fechado afirma que as classes podem ser estendidas, mas não modificadas. Portanto, é recomendável não usar o tipo de dados de enumeração, pois não é uma abordagem escalável.

**Nota:** O uso de enums não é proibido, mas apenas não é recomendado. Posteriormente, usaremos o `PaymentStatus` enum em nosso projeto de estacionamento, pois não exigirá mais modificações.

Uma **classe abstrata** não pode instanciar o objeto e só pode ser usada como uma classe base. A classe abstrata para `Vehicle` é a melhor abordagem. Ele nos permite criar classes filhas derivadas para a `Vehicle` classe. Pode ser estendido facilmente caso o tipo de veículo mude no futuro.



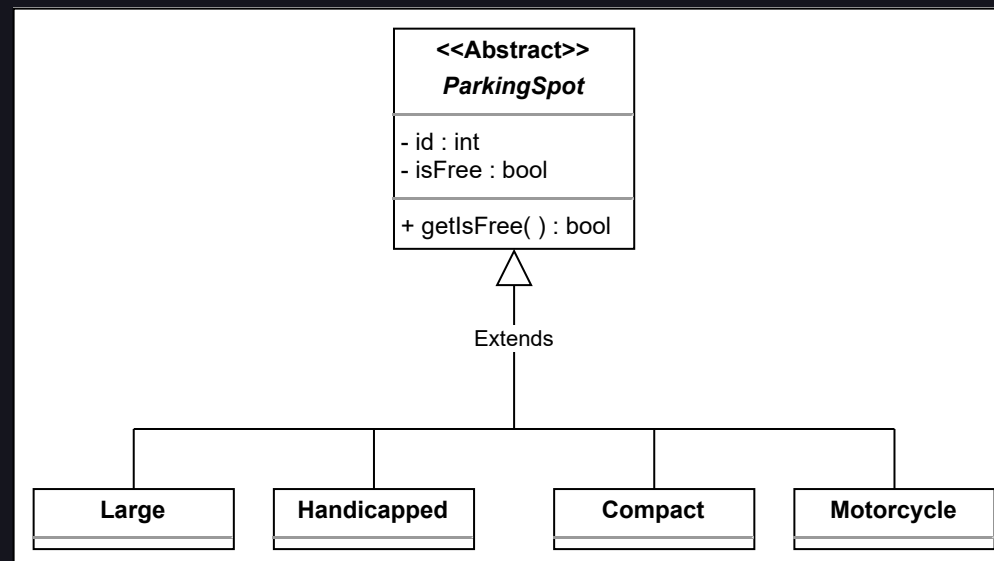


O diagrama de classes de Vehicle e suas classes derivadas

💡 Clique para ver o requisito relevante

## Vaga de estacionamento

Semelhante à `Vehicle` classe, o `ParkingSpot` também deve ser uma classe abstrata. Existem quatro tipos de vagas de estacionamento: deficientes, compactas, grandes e para motocicletas. Essas classes podem ser derivadas da classe abstrata vaga de estacionamento.



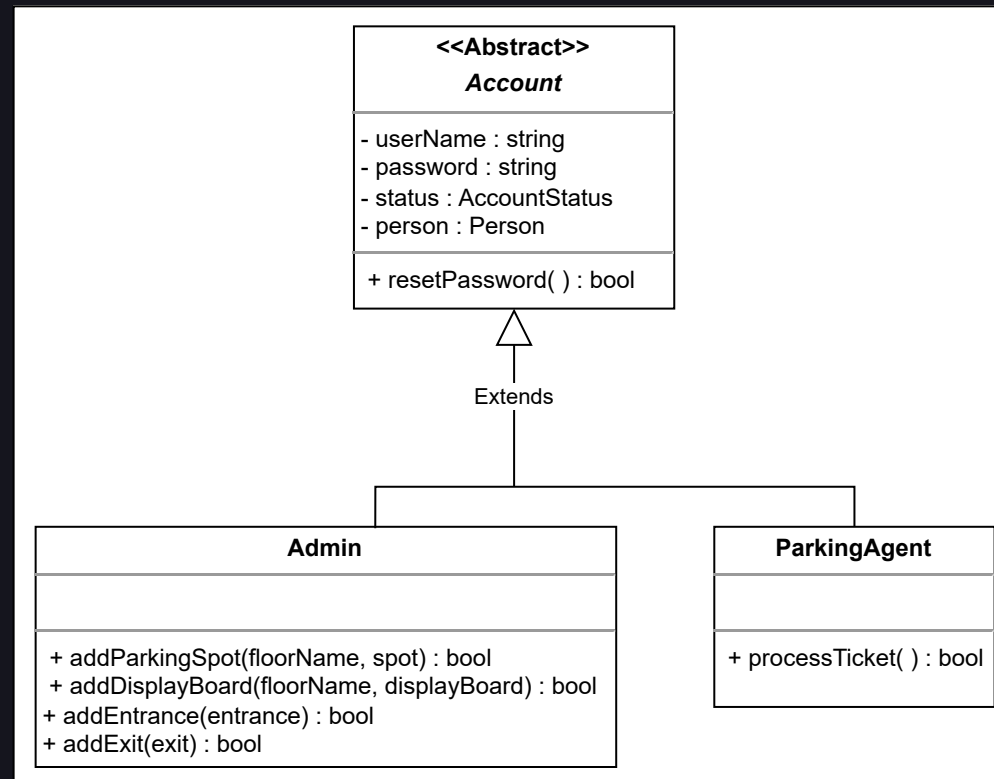
O diagrama de classe do ParkingSpot e suas classes derivadas

💡 Clique para ver o requisito relevante



## da conta

Semelhante às classes `Vehicle`, `ParkingSpot`, `Account` também deve ser uma classe abstrata. Existem duas classes filhas: `Admin`, `ParkingAgent`. Essas classes podem ser derivadas da classe abstrata da conta.



O diagrama de classes de Account e suas classes derivadas

## placa de exibição


Esta classe representa os tipos de vagas gratuitas e o número de vagas vazias.





DisplayBoard
- id : int - handicappedSpot : Handicapped {list} - compactSpot : Compact {list} - largeSpot : Large {list} - motorcycleSpot : Motorcycle {list}
+ showFreeSlot( ) : void

O diagrama de classes da classe DisplayBoard

 [Clique para ver o requisito relevante](#)

## Entrada e saída

A **Entrance** classe é responsável por devolver o ticket de estacionamento sempre que um veículo chegar. Ele contém o atributo ID, pois existem várias entradas para o estacionamento. Também tem o `getTicket()` método.

A **Exit** classe é responsável por validar a situação de pagamento do ticket de estacionamento antes de liberar o veículo para sair do estacionamento. Ele contém o atributo ID, pois existem várias saídas para o estacionamento. Também tem o `validateTicket()` método.

Entrance	Exit
- id : int	- id : int
+ getTicket( ) : ParkingTicket	+ validateTicket( ) : void

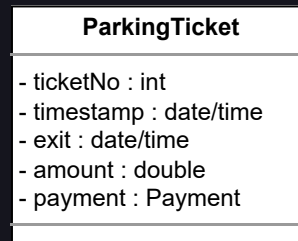
O diagrama de classes das classes Entrance e Exit



💡 Clique para ver o requisito relevante

## Bilhete de estacionamento

A `ParkingTicket` classe é uma das classes centrais do sistema. Ele acompanha os horários de entrada e saída dos veículos, o valor e a situação do pagamento.



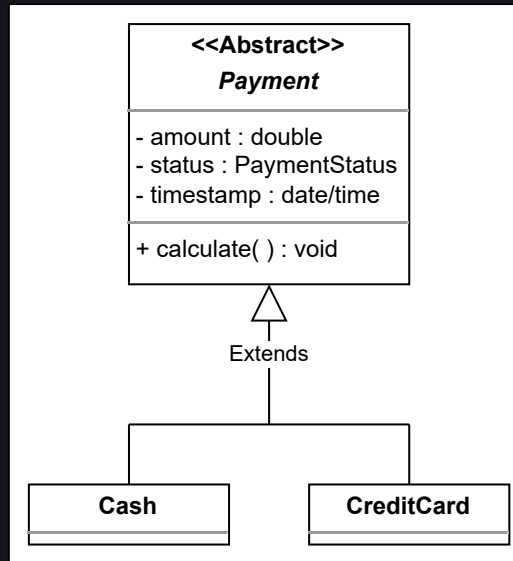
O diagrama de classes da classe ParkingTicket

💡 Clique para ver o requisito relevante

## de pagamento

A `Payment` classe será uma classe abstrata e terá duas classes filhas, `card` e `cash`, visto que são duas formas de pagamento do sistema de estacionamento.



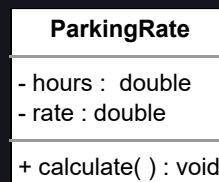


O diagrama de classes da classe Payment

💡 Clique para ver o requisito relevante

## Taxa de estacionamento

A **ParkingRate** classe é responsável por calcular o pagamento final com base no tempo gasto no estacionamento.



O diagrama de classes da classe ParkingRate

?

Tt

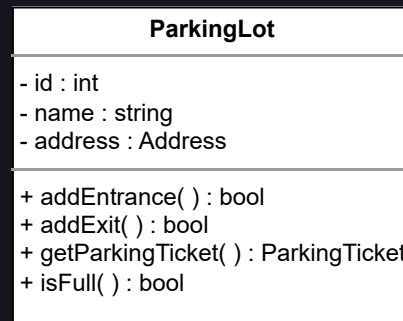
☀



💡 Clique para ver o requisito relevante

## Estacionamento

Agora, discutiremos o design de toda a `ParkingLot` classe do sistema. Este sistema de estacionamento é composto por objetos menores que já projetamos, como entrada/saída, vagas de estacionamento, taxas de estacionamento, etc.



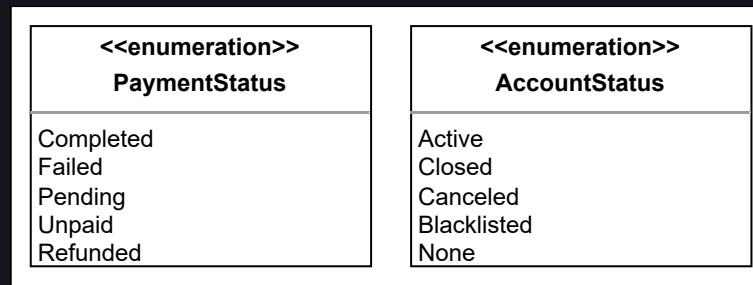
O diagrama de classe para a classe ParkingLot

## As enumerações e tipos de dados personalizados

O seguinte fornece uma visão geral das enumerações e tipos de dados personalizados usados neste problema:

- `PaymentStatus`: Precisamos criar uma enumeração para acompanhar o status do pagamento do ticket de estacionamento, seja ele pago, não pago, cancelado, reembolsado e assim por diante.
- `AccountStatus`: Precisamos criar uma enumeração para acompanhar o status da conta, se ela está ativa, cancelada, fechada e assim por diante.

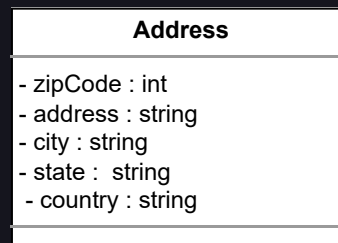




Enums no sistema de estacionamento

## Endereço

Também precisamos criar um tipo de dado personalizado, **Address**, que armazenará a localização do estacionamento.

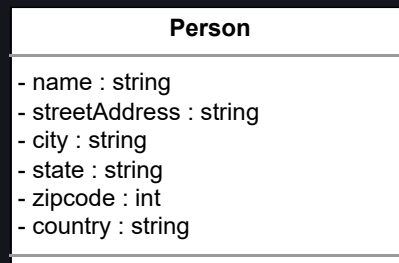


O diagrama de classes do tipo de dados personalizado Address

## Pessoa

A **Person** classe é usada para armazenar informações relacionadas a uma pessoa, como nome, endereço, país, etc.





O diagrama de classe do tipo de dados personalizado da classe Person

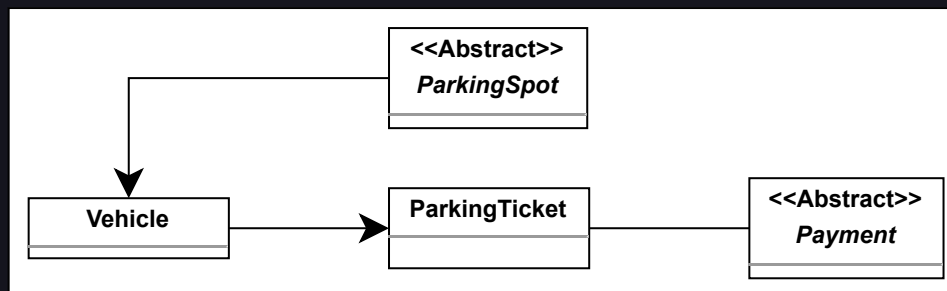
## Relação entre as classes

Agora, discutiremos os relacionamentos entre as classes que definimos acima em nosso sistema de estacionamento.

## Associação

O diagrama de classes tem os seguintes relacionamentos de associação:

- O **ParkingSpot** tem uma associação unidirecional com **Vehicle**.
- O **Vehicle** tem uma associação unidirecional com **ParkingTicket**.
- O **Payment** tem uma associação bidirecional com **ParkingTicket**.

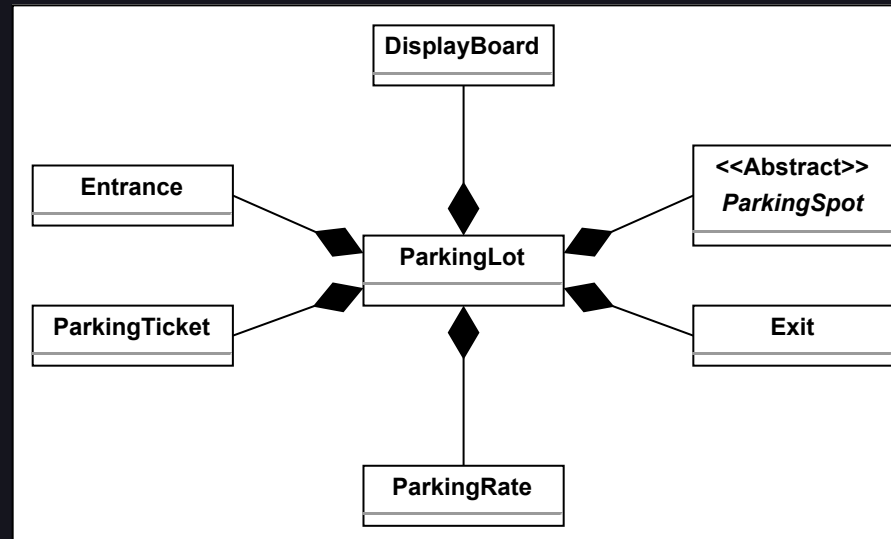


O relacionamento de associação entre as classes

## Composição

O diagrama de classes tem os seguintes relacionamentos de composição.

- A `ParkingLot` classe inclui `Entrance`, `Exit`, `ParkingRate`, `DisplayBoard`, `ParkingTicket` e `ParkingSpot`.



O relacionamento de composição entre classes

## Herança

As seguintes classes mostram um relacionamento de herança:

- A `Vehicle` classe inclui `Car`, `Truck`, `Vane` e `MotorCycles` subclasses.
- A `ParkingSpot` classe inclui `handicapped`, `compact`, `large` e `motorcycles` subclasses.
- A `Payment` classe inclui as subclasses `Cash` e `CreditCard`.





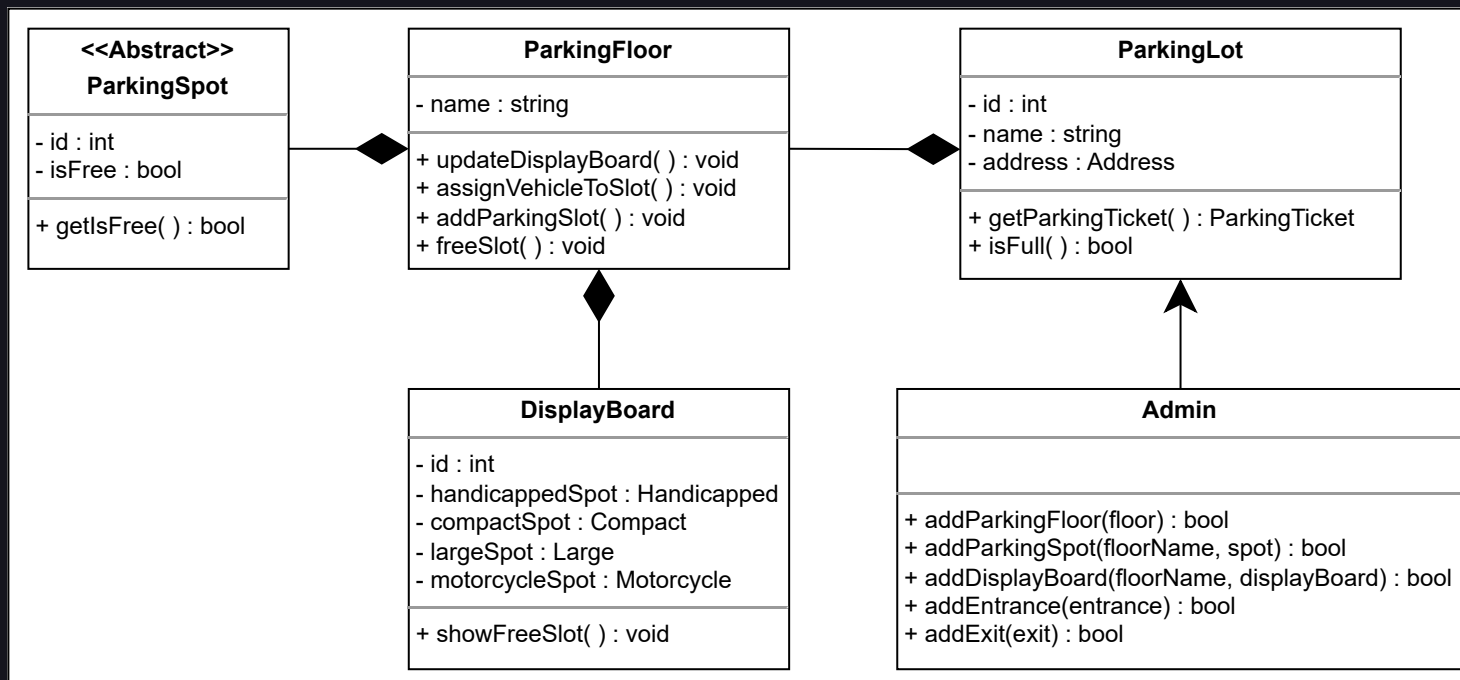
>

Este sistema de estacionamento também é composto por objetos menores que já projetamos, como entrada, saída, vagas de estacionamento, taxas de estacionamento, etc. Portanto, será uma boa prática usar o Abstract Factory e o padrão de design Factory para instanciar todos esses objetos.

## Requisitos adicionais

O entrevistador pode introduzir alguns requisitos adicionais no sistema de estacionamento ou pode fazer algumas perguntas complementares. Vejamos alguns exemplos de requisitos adicionais:

**Piso do estacionamento:** O estacionamento deve ter vários andares onde os clientes possam estacionar seus carros. O diagrama de classes fornecido abaixo mostra o relacionamento de `ParkingFloor` com outras classes:

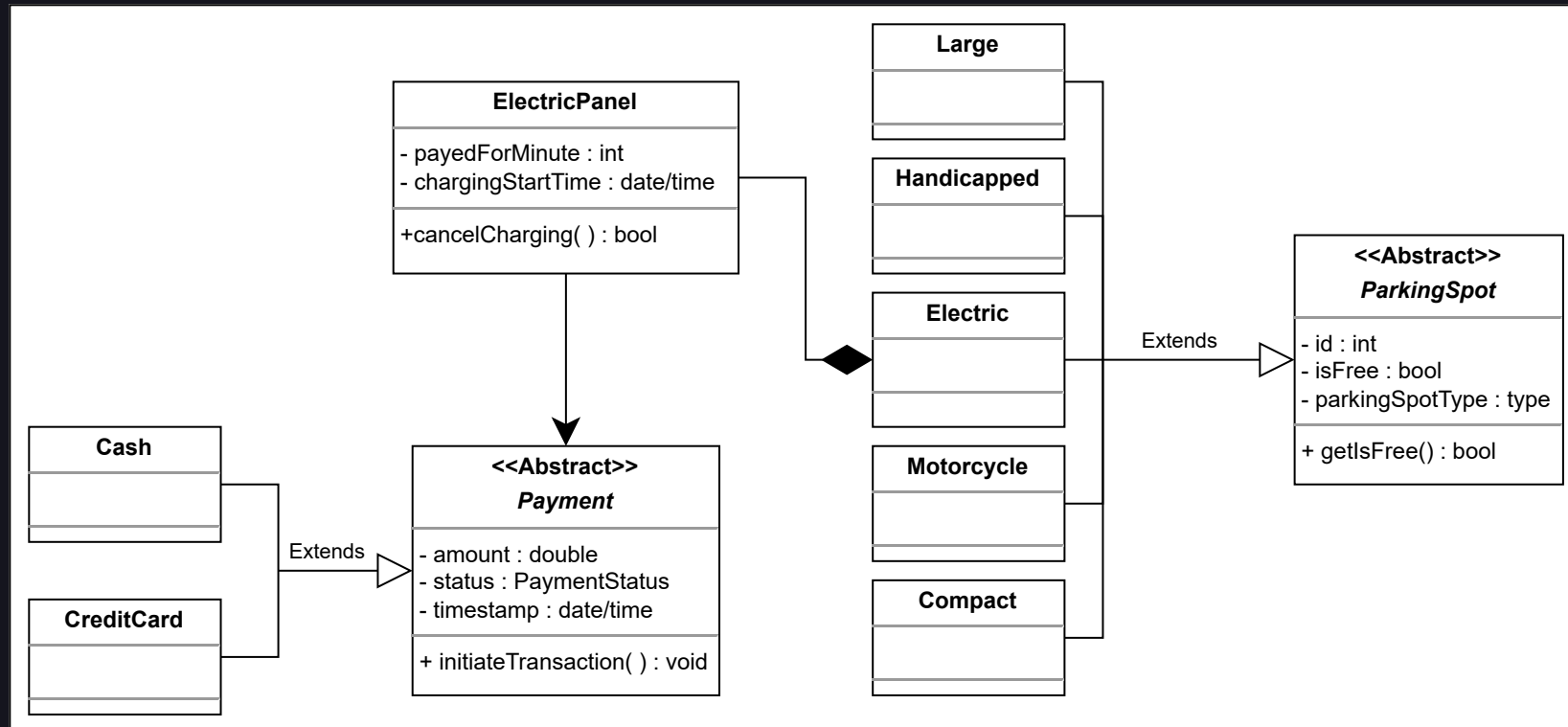


Relacionamento da classe `ParkingFloor` com outras classes



>

**Elétrico:** O estacionamento deve ter algumas vagas específicas para carros elétricos. Esses pontos devem ter um painel elétrico através do qual os clientes podem pagar e carregar seus veículos. O diagrama de classes fornecido abaixo mostra o relacionamento de `Electric` e `ElectricPanel` com outras classes:



Relacionamento da classe `Electric` e `ElectricPanel` com outras classes

Pergunta

Digamos que o entrevistador lhe peça que o estacionamento deva atribuir uma vaga mais próxima à

