Design a Parking Lot using OOP

A parking lot or car park is a designated open space designed for the purpose of parking vehicles. In regions where automobiles serve as a primary means of transportation, you can typically find parking lots in both urban and suburban areas. These parking areas are commonly seen at locations such as shopping centers, sports stadiums, large churches, and similar venues, often covering extensive areas.

System Requirements

1. The parking facility must incorporate a multi-level structure to accommodate customer vehicles.
2. There should be numerous entry and exit points distributed across the parking lot.
3. Customers should have the option to obtain a parking ticket upon entering and settle the parking fee when exiting via designated payment points.
4. Payment options should include both cash and credit cards for customer convenience.
5. The parking management system should display a message on the entrance panel and the ground floor parking display board when the capacity limit is reached.
6. Each parking level must feature a variety of parking spot types, including Compact, Large, Handicapped, Motorcycle, etc., catering to different vehicle sizes and needs.
7. The system should support diverse vehicle types, encompassing cars, trucks, vans, motorcycles, and more.
8. Each parking floor must be equipped with a display board that provides real-time information about the availability of parking spots for each spot type.
9. Fare Calculation: The system should calculate parking fees based on the duration of the vehicle's stay in the parking lot. Different rates may apply for various vehicle types and parking spot categories.
10. "Find My Car" feature that allows customers to locate their parked vehicles easily. Customers can enter their ticket number or use a mobile app to pinpoint the exact location of their car within the parking facility.
11. Membership Plans: Introduce membership plans for frequent visitors. Customers can opt for different membership tiers with benefits such as priority access.
12. Priority for People with Disabilities: Ensure priority parking spaces close to entrances are reserved exclusively for individuals with disabilities.

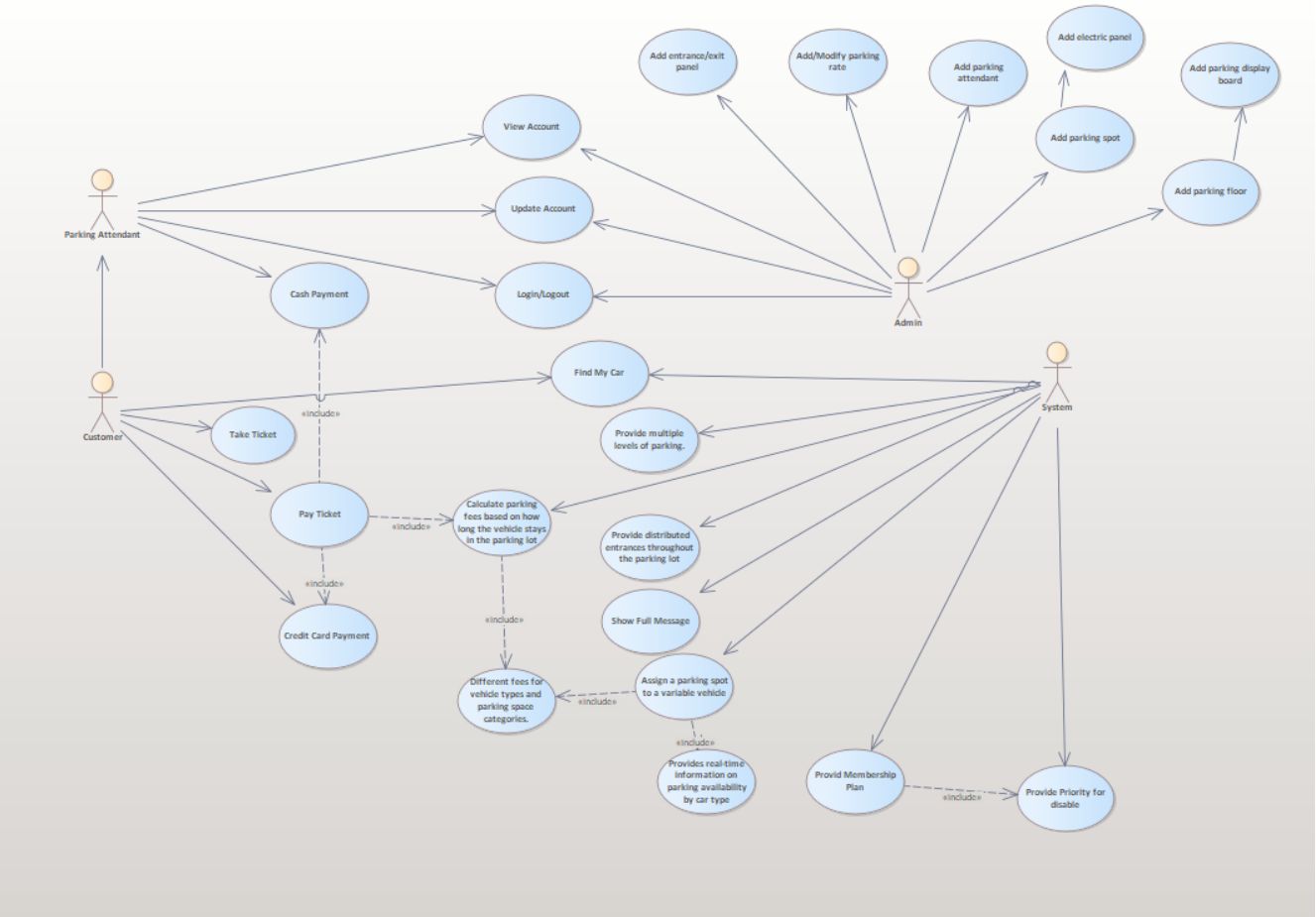
Use Case Diagram

Here are the main Actors in our system:

1. Admin: Mainly responsible for adding and modifying parking floors, parking spots, entrance, and exit panels, adding/removing parking attendants, etc.
2. Customer: All customers can get a parking ticket and pay for it.
3. Parking Attendant: Parking attendants can do all the activities on the customer's behalf, and can take cash for ticket payment.
4. System: To display messages on different info panels, as well as assigning and removing a vehicle from a parking spot.

Here are the top use cases for Parking Lot:

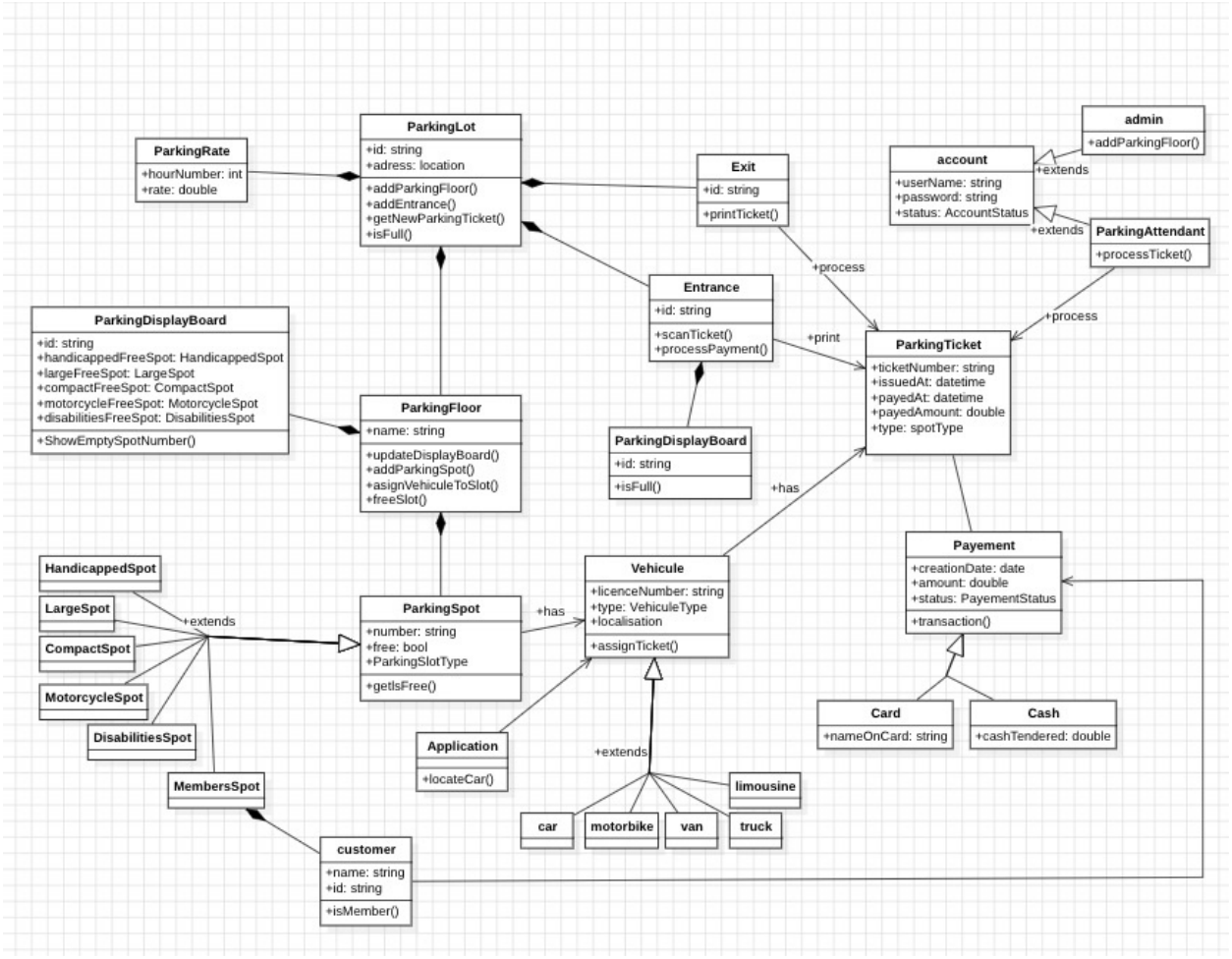
1. Add/Remove parking floor: To add, remove or modify a parking floor
2. from the system. Each floor can have its own display board to show free parking spots.
3. Add/Remove parking spot: To add, remove or modify a parking spot on a parking floor.
4. Add/Remove a parking attendant: To add or remove a parking attendant from the system.
5. Create/Modify Account: To create or modify a parking attendant's account
6. Take ticket: To provide customers with a new parking ticket when entering the parking lot.
7. Scan ticket: To scan a ticket to find out the total charge.
8. Calculate/Modify parking rate: To allow admin to add or modify the hourly parking rate.
9. Find My Car: To request one's car position on the parking lot


Add entrance/exit panel
Add/Modify parking rate
Add parking attendant
Add electric panel
Add parking display board
Add parking spot
Add parking floor
View Account
Update Account
Cash Payment
Login/Logout
Parking Attendant
Admin
Find My Car
System
Customer
Take Ticket
Provide multiple levels of parking.
«include»
Pay Ticket
Calculate parking fees based on how long the vehicle stays in the parking lot
Provide distributed entrances throughout the parking lot
Credit Card Payment
Show Full Message
Different fees for vehicle types and parking space categories.
Assign a parking spot to a variable vehicle
Provides real-time information on parking availability by car type
Provid Membership Plan
Provide Priority for disable

Class Diagram

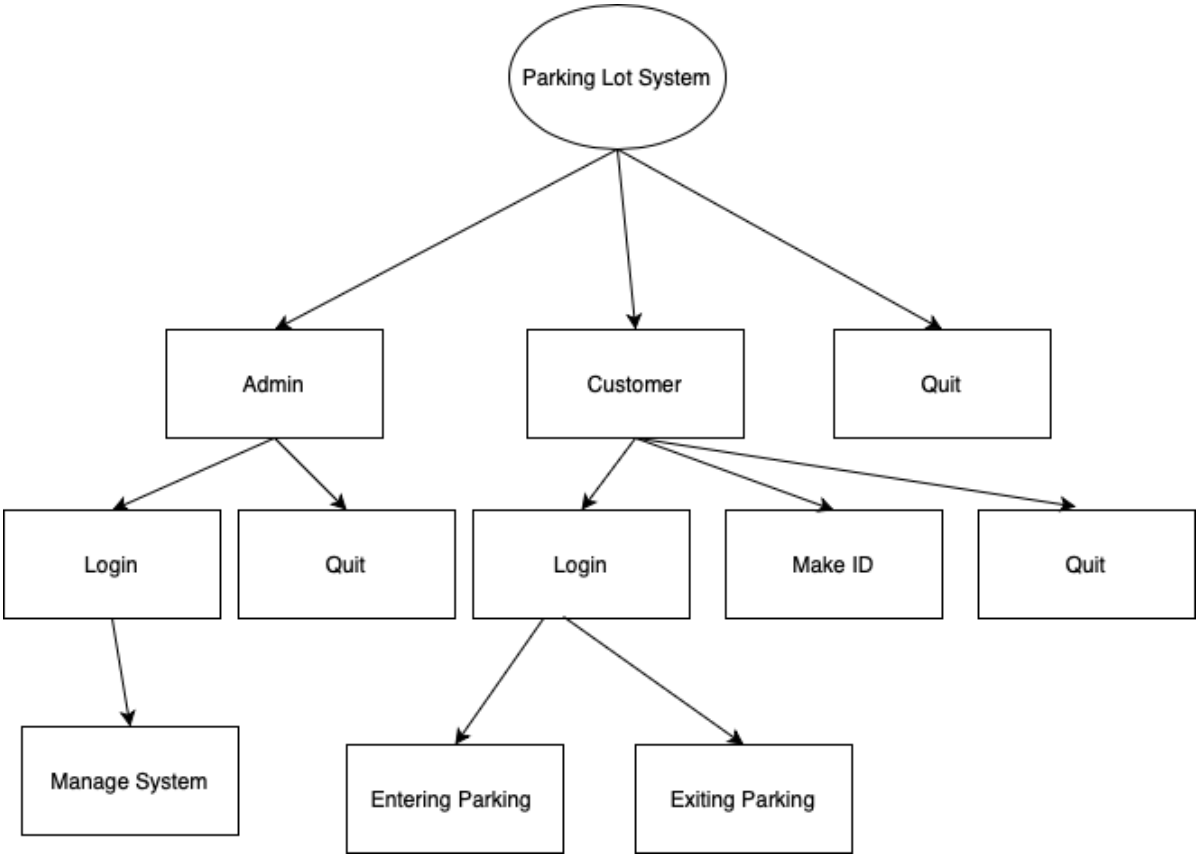
Here are the main classes of our Parking Lot System:

1. ParkingLot: The central part of the organization for which this software has been designed. It has attributes like 'id' to distinguish it from any other parking lots and 'Address' to define its location.
2. ParkingFloor: The parking lot will have many parking floors.
3. ParkingSpot: Each parking floor will have many parking spots. Our system will support different parking spots 1) Handicapped, 2) Compact, 3) Large, 4) Motorcycle
4. Account: We will have two types of accounts in the system: one for an Admin, and the other for a parking attendant.
5. Parking ticket: This class will encapsulate a parking ticket. Customers will take a ticket when they enter the parking lot.
6. Vehicle: Vehicles will be parked in the parking spots. Our system will support different types of vehicles 1) Car, 2) Truck, 3) Van and 4) Motorcycle. It will have an additional application for find my car function.
7. EntrancePanel and ExitPanel: EntrancePanel will print tickets, and ExitPanel will facilitate payment of the ticket fee.
8. Payment: This class will be responsible for making payments. The system will support credit card and cash transactions.
9. ParkingRate: This class will keep track of the hourly parking rates. It will specify a dollar amount for each hour.
10. ParkingDisplayBoard: Each parking floor will have a display board to show available parking spots for each spot type. This class will be responsible for displaying the latest availability of free parking spots to the customers.


ParkingRate
+hourNumber: int
+rate: double
ParkingLot
+id: string
+adress: location
+addParkingFloor()
+addEntrance()
+getNewParkingTicket()
+isFull()
Exit
+id: string
+printTicket()
admin
+addParkingFloor()
account
+userName: string
+password: string
+status: AccountStatus
+extends
+extends
ParkingAttendant
+processTicket()
+process
+process
Entrance
+id: string
+scanTicket()
+processPayment()
+print
ParkingTicket
+ticketNumber: string
+issuedAt: datetime
+payedAt: datetime
+payedAmount: double
+type: spotType
ParkingDisplayBoard
+id: string
+handicappedFreeSpot: HandicappedSpot
+largeFreeSpot: LargeSpot
+compactFreeSpot: CompactSpot
+motorcycleFreeSpot: MotorcycleSpot
+disabilitiesFreeSpot: DisabilitiesSpot
+ShowEmptySpotNumber()
ParkingFloor
+name: string
+updateDisplayBoard()
+addParkingSpot()
+asignVehiculeToSlot()
+freeSlot()
ParkingDisplayBoard
+id: string
+isFull()
+has
Payement
+creationDate: date
+amount: double
+status: PayementStatus
+transaction()
HandicappedSpot
LargeSpot
CompactSpot
MotorcycleSpot
DisabilitiesSpot
+extends
ParkingSpot
+number: string
+free: bool
+ParkingSlotType
+getIsFree()
+has
Vehicule
+licenceNumber: string
+type: VehiculeType
+localisation
+assignTicket()
Card
+nameOnCard: string
Cash
+cashTendered: double
Application
+locateCar()
+extends
limousine
car
motorbike
van
truck
MembersSpot
customer
+name: string
+id: string
+isMember()

Page Flow Diagram

A diagram that illustrates the relationship between pages in application



Code

Constants.py

Three enumerations were defined: VehicleType, ParkingSpotType, and UserRole.

```
from enum import Enum, IntEnum


class VehicleType(Enum):
    CAR = 1
    TRUCK = 2
    VAN = 3
    MOTORBIKE = 4

class ParkingSpotType(Enum):
    HANDICAPPED = 1
    COMPACT = 2
    LARGE = 3
    MOTORBIKE = 4


class UserRole(IntEnum):
    ADMIN = 1
    USER = 2
    NULL = 3
```

Account.py

This class takes three parameters in its constructor: id, password, and user_role, and contains a method called ComparePassword to compare the password.

```
from enum import Enum, IntEnum
from Constants import *

class Account:
    def __init__(self, id, password, user_role: UserRole):
        self.id = id
        self.password = password
        self.user_role = user_role

    def ComparePassword(self,password):
        return self.password == password
```

Admin.py

Represent an administrative user in a parking management system.

```
From Constants import *
from ParkingFloor import *
from Account import *
from ParkingLot import *


class Admin:
    _instance = None

    def __new__(cls, id, password):
        if cls._instance is None:
            cls._instance = super(Admin, cls).__new__(cls)
            cls._instance.initialized = False
        return cls._instance

    def __init__(self, id, password):
        if not self.initialized:
            self.__Account = Account(id, password, UserRole.ADMIN)
            self.initialized = True
            self.parkingLot = ParkingLot()

    def viewAdminPage(self):
        while True:
            print("Admin Menu:")
            print("1. Add Parking Floor")
            print("2. Display Parking Lot")
            print("3. Return to Main Menu")

            choice = input("Enter your choice (1/2/3): ")

            print("----------------------------------------")

            if choice == "1":
                self.addParkingFloor()
            elif choice == "2":
                self.displayParkingFloors()
            elif choice == "3":
                break
            else:
                print("Invalid choice. Please try again.")

    def checkAccount(self):
        print("---------------------------------------")
        inputId = input("Admin ID: ")
        inputPassword = input("Admin Password: ")
        print("---------------------------------------")

        if inputId == self.__Account.id and inputPassword == self.__Account.password:
            print("Successfully Logged In")
            return True
        else:
            return False

    def displayParkingFloors(self):
        if not self.parkingLot.parkingFloors:
            print("No parking floors available.")
        else:
            print("Parking Floors and Spots:")
            for floor_number, floor in enumerate(self.parkingLot.parkingFloors, start=1):
                floor.ShowAllParkingFloor()
                print()

    def addParkingFloor(self):
        try:
            spotCounts = {}
            spotCounts[ParkingSpotType.HANDICAPPED] = int(input("Enter the number of handicapped spots: "))
            spotCounts[ParkingSpotType.COMPACT] = int(input("Enter the number of compact spots: "))
            spotCounts[ParkingSpotType.LARGE] = int(input("Enter the number of large spots: "))
            spotCounts[ParkingSpotType.MOTORBIKE] = int(input("Enter the number of motorbike spots: "))

            if any(count < 0 for count in spotCounts.values()):
                print("Spot counts cannot be negative.")
                return

            floorNumber = len(self.parkingLot.parkingFloors) + 1
            floor = ParkingFloor(floorNumber)

            for spot_type, count in spotCounts.items():
                for _ in range(count):
                    floor.AddParkingSpot(spot_type)

            self.parkingLot.parkingFloors.append(floor)
            print(f"Added a new floor (Floor {floorNumber}) with the specified parking spots.")
        except ValueError:
            print("Invalid input. Please enter valid spot counts."
```

User.py

Represent a customer user in a parking management system.

```
from Account import Account
from Constants import *
from ParkingLot import ParkingLot
from ParkingSpot import ParkingSpot
import datetime

class User:
    def __init__(self,id,password,isDisable,carName,carType):
        self.parkingLot = ParkingLot()
        self.account = Account(id,password,UserRole.USER)
        self.car = Car(carName,carType)
        self.isDiable = isDisable

    def ComparePassword(self,password):
        return self.account.ComparePassword(password)

    def ShowUserPage(self):
        while True:
            print("User Menu:")
            print("1. Enter Parking Lot")
            print("2. Exit Parking Lot")
            print("3. Return to Main Menu")

            choice = input("Enter your choice (1/2/3): ")
            if choice == "1":
                self.EnterParkingLot()
            elif choice == "2":
                self.ExitParkingLot()
            elif choice == "3":
                break
            else:
                print("Invalid choice. Please try again.")

    def EnterParkingLot(self):
        for floor in self.parkingLot.parkingFloors:
            # Check for empty spots of the specified car type on the current floor
            if self.isDiable == True:
                for spot in floor.handicappedSpots():
                    if floor.handicappedSpots()[spot].isFree():
                        floor.handicappedSpots()[spot].assignVehicle( self.car )
                        self.car.SetEnterTime(datetime.datetime.now())
                        print(f"A { self.car.vehicleType } has entered the parking lot and parked in
Compact Spot {spot} on the {floor.floornumber()} floor")
                        self.car.SetSpotInfo(floor.floornumber(),spot,ParkingSpotType.HANDICAPPED)
                        self.car.__isParked = True
                        return

            elif self.car.vehicleType == VehicleType.CAR and len(floor.compactSpots()) > 0:
                # Assign the car to the first available compact spot
                for spot in floor.compactSpots():
                    if floor.compactSpots()[spot].isFree():
                        floor.compactSpots()[spot].assignVehicle( self.car )
                        self.car.SetEnterTime(datetime.datetime.now())
                        print(f"A { self.car.vehicleType } has entered the parking lot and parked in
Compact Spot {spot} on the {floor.floornumber()} floor")
                        self.car.SetSpotInfo(floor.floornumber(),spot,ParkingSpotType.COMPACT)
                        self.car.__isParked = True
                        return

            elif self.car.vehicleType  == VehicleType.TRUCK and len(floor.largeSpots()) > 0:
                # Assign the truck to the first available large spot
                for spot in floor.largeSpots():
                    if floor.largeSpots()[spot].isFree():
                        floor.largeSpots()[spot].assignVehicle( self.car )
                        self.car.SetEnterTime(datetime.datetime.now())
                        print(f"A { self.car.vehicleType } has entered the parking lot and parked in
Large Spot {spot} on the {floor.floornumber()} floor")
                        self.car.SetSpotInfo(floor.floornumber(),spot,ParkingSpotType.LARGE)
                        self.car.__isParked = True
```

```
                        return

            elif self.car.vehicleType  == VehicleType.VAN and len(floor.largeSpots()) > 0:
                # Assign the van to the first available large spot
                for spot in floor.largeSpots():
                    if floor.largeSpots()[spot].isFree():
                        floor.largeSpots()[spot].assignVehicle( self.car )
                        self.car.SetEnterTime(datetime.datetime.now())
                        print(f"A { self.car.vehicleType } has entered the parking lot and parked in
Large Spot {spot} on the {floor.floornumber()} floor")
                        self.car.SetSpotInfo(floor.floornumber(),spot,ParkingSpotType.LARGE)
                        self.car.__isParked = True
                        return

            elif self.car.vehicleType  == VehicleType.MOTORBIKE and len(floor.motorbikeSpots()) > 0:
                # Assign the motorbike to the first available large spot
                for spot in floor.motorbikeSpots():
                    if floor.motorbikeSpots()[spot].isFree():
                        floor.motorbikeSpots()[spot].assignVehicle( self.car.vehicleType )
                        self.car.SetEnterTime(datetime.datetime.now())
                        print(f"A { self.car.vehicleType } has entered the parking lot and parked in
Motorbike Spot {spot} on the {floor.floornumber()} floor")
                        self.car.SetSpotInfo(floor.floornumber(),spot,ParkingSpotType.MOTORBIKE)
                        return

    def ExitParkingLot(self):
        # if floorNum == 1 -> that means 0st array
        floor = self.parkingLot.parkingFloors[self.car.floorNum - 1]

        spot = None
        if self.car.spotType == ParkingSpotType.COMPACT:
            spot = floor.compactSpots()[self.car.spotNum]
        elif self.car.spotType == ParkingSpotType.LARGE:
            spot = floor.largeSpots()[self.car.spotNum]
        elif self.car.spotType == ParkingSpotType.MOTORBIKE:
            spot = floor.motorbikeSpots()[self.car.spotNum]
        elif self.car.spotType == ParkingSpotType.HANDICAPPED:
            spot = floor.handicappedSpots()[self.car.spotNum]

        self.car.SetExitTime(datetime.datetime.now())
        spot.removeVehicle()
        self.car.__isParked = False

        self.car.CalculateFee()


class Car:
    def __init__(self,name,vehicleType):
        self.__name = name
        self.vehicleType = vehicleType

        self.__enterTime = None
        self.__exitTime = None
        self.__isParked = False

        self.floorNum = None
        self.spotNum = None
        self.spotType = None

    def SetEnterTime(self, enterTime):
        self.__enterTime = enterTime

    def SetExitTime(self, exitTime):
        self.__exitTime = exitTime

    def FindMyCar(self):
        if(self.__isParked == False):
            print("Your Car not Parked")
            return

        else:
            print("Car Parked {} floor {} spot".format(self.floorNum,self.spotNum))

    def SetSpotInfo(self,floorNum,spotNum,spotType):
        self.floorNum = floorNum
        self.spotNum = spotNum
```

```
        self.spotType = spotType

        print("Car Parked {} floor {} spot".format(floorNum,spotNum))

    def CalculateFee(self):
        parking_duration = self.__exitTime - self.__enterTime
        hour_parked = parking_duration.total_seconds() / 3600
        if(self.vehicleType == 1):
            if(hour_parked <= 1):
                parking_fee = 4
            elif(hour_parked <= 3):
                parking_fee = 4 + 3.5*(hour_parked - 1)
            else:
                parking_fee = 4 + 3.5*2 + 2.5*(hour_parked - 3)

        elif(self.vehicleType == 2 or self.vehicleType == 3):
            if(hour_parked <= 1):
                parking_fee = 5
            elif(hour_parked <= 3):
                parking_fee = 5 + 4.5*(hour_parked - 1)
            else:
                parking_fee = 5 + 4.5*2 + 3*(hour_parked - 3)

        else:
            if(hour_parked <= 1):
                parking_fee = 3
            elif(hour_parked <= 3):
                parking_fee = 3 + 2.5*(hour_parked - 1)
            else:
                parking_fee = 3 + 2.5*2 + 2*(hour_parked - 3)

        print(f"You used parking spot for {hour_parked:.3f} hours and the Parking Fee is
${parking_fee}.")
```

Parkinglot.py

Class responsible for saving data regarding parking lot

```
from ParkingFloor import ParkingFloor
from Constants import *

class ParkingLot:
    _instance = None

    def __new__(cls):
        if cls._instance is None:
            cls._instance = super(ParkingLot, cls).__new__(cls)
            cls._instance.initialized = False
        return cls._instance

    def __init__(self):
        if not self.initialized:
            self.initialized = True
            self.parkingFloors = []
```

Parkingfloor.py

Class responsible for containing data regarding each floor separately

```
from ParkingSpot import *
from Constants import *


class ParkingFloor():
    def __init__(self, floorNumber):
        self.__spotCount = 0
        self.__floorNumber = floorNumber
        self.__handicappedSpots = {}
        self.__compactSpots = {}
        self.__largeSpots = {}
        self.__motorbikeSpots = {}


    def floornumber(self):
        return self.__floorNumber

    def handicappedSpots(self):
        return self.__handicappedSpots

    def compactSpots(self):
        return self.__compactSpots

    def largeSpots(self):
        return self.__largeSpots

    def motorbikeSpots(self):
        return self.__motorbikeSpots

    def AddParkingSpot(self, spotType: ParkingSpotType):
        self.__spotCount += 1
        # Determine the next spot number for the given spot type
        if spotType == ParkingSpotType.HANDICAPPED:
            next_spot_number = len(self.__handicappedSpots) + 1
            parkingSpot = HandicappedSpot(next_spot_number,self.__floorNumber)
            self.__handicappedSpots[next_spot_number] = parkingSpot
        elif spotType == ParkingSpotType.COMPACT:
            next_spot_number = len(self.__compactSpots) + 1
            parkingSpot = CompactSpot(next_spot_number,self.__floorNumber)
            self.__compactSpots[next_spot_number] = parkingSpot
        elif spotType == ParkingSpotType.LARGE:
            next_spot_number = len(self.__largeSpots) + 1
            parkingSpot = LargeSpot(next_spot_number,self.__floorNumber)
            self.__largeSpots[next_spot_number] = parkingSpot
        elif spotType == ParkingSpotType.MOTORBIKE:
            next_spot_number = len(self.__motorbikeSpots) + 1
            parkingSpot = MotorbikeSpot(next_spot_number,self.__floorNumber)
            self.__motorbikeSpots[next_spot_number] = parkingSpot


    def ShowAllParkingFloor(self):
        print(f"Floor {self.__floorNumber} - Parking Spots:")
        print(f"Total Spots: {self.__spotCount}")
        print(f"Handicapped Spots: {len(self.__handicappedSpots)}")
        for spot_number, spot in self.__handicappedSpots.items():
            print(f"  - Spot {spot_number}: {spot.isFree()}")

        print(f"Compact Spots: {len(self.__compactSpots)}")
        for spot_number, spot in self.__compactSpots.items():
            print(f"  - Spot {spot_number}: {spot.isFree()}")

        print(f"Large Spots: {len(self.__largeSpots)}")
        for spot_number, spot in self.__largeSpots.items():
            print(f"  - Spot {spot_number}: {spot.isFree()}")

        print(f"Motorbike Spots: {len(self.__motorbikeSpots)}")
        for spot_number, spot in self.__motorbikeSpots.items():
            print(f"  - Spot {spot_number}: {spot.isFree()}")

        print()
```

Parkingspot.py

Class containing information regarding each individual spot

```
from Constants import *
import datetime

class ParkingSpot:
    def __init__(self, spotNumber, parkingSpotType,floorNum):
        self.__free = True
        self.__vehicle = None
        self.__parkingSpotType = parkingSpotType
        self.floorNum = floorNum
        self.spotNum = spotNumber

    def isFree(self):
        return self.__free

    def assignVehicle(self, vehicle):
        self.__vehicle = vehicle
        self.__free = False
        self.__vehicle.parkedSpot = self
        self.__vehicle.enterTime = datetime.datetime.now()


    def removeVehicle(self):
        self.__vehicle.exitTime = datetime.datetime.now()
        self.__vehicle.parkedSpot = None
        self.__vehicle = None
        self.__free = True

    def ShowSpotInfo(self):
        print("Floor : {self.floorNum}, spotType = {self.__parkingSpotType},
spotNum : {self.spotNum}")

class HandicappedSpot(ParkingSpot):
    def __init__(self, number,floorNum):
        super().__init__(number, ParkingSpotType.HANDICAPPED,floorNum)

class CompactSpot(ParkingSpot):
    def __init__(self, number,floorNum):
        super().__init__(number, ParkingSpotType.COMPACT,floorNum)

class LargeSpot(ParkingSpot):
    def __init__(self, number,floorNum):
        super().__init__(number, ParkingSpotType.LARGE,floorNum)

class MotorbikeSpot(ParkingSpot):
    def __init__(self, number,floorNum):
        super().__init__(number, ParkingSpotType.MOTORBIKE,floorNum)
```

Output

An example of how code is working. A sample output

```
-----------------------------------------
 Select Your Mode
-----------------------------------------
 1. Admin
 2. Customer
 3. Quit
-----------------------------------------
Input : 1
-----------------------------------------
Admin ID: Admin
Admin Password: 123
-----------------------------------------
Successfully Logged In
```

```
Admin Menu:
1. Add Parking Floor
2. Display Parking Lot
3. Return to Main Menu
Enter your choice (1/2/3): 1
----------------------------------------
Enter the number of handicapped spots: 1
Enter the number of compact spots: 1
Enter the number of large spots: 1
Enter the number of motorbike spots: 1
Added a new floor (Floor 1) with the specified parking spots.
Admin Menu:
1. Add Parking Floor
2. Display Parking Lot
3. Return to Main Menu
Enter your choice (1/2/3): 2
----------------------------------------
Parking Floors and Spots:
Floor 1 - Parking Spots:
Total Spots: 4
Handicapped Spots: 1
  - Spot 1: True
Compact Spots: 1
  - Spot 1: True
Large Spots: 1
  - Spot 1: True
Motorbike Spots: 1
  - Spot 1: True
```

```
-----------------------------------------
 Select Your Mode
-----------------------------------------
 1. Admin
 2. Customer
 3. Quit
-----------------------------------------
Input : 2
User Account Menu:
1. Make ID
2. Log in
3. Quit
Enter your choice (1/2/3): 1
User ID: ID
User Password: 123
Are you didable(y/n): y
Choose your car type(1.Car/2.Truck/3.Van/4.Motorbike): 1
Write your car name: Car
```

```
Enter your choice (1/2/3): 2
User ID: ID
User Password: 123
Success to Login
User Menu:
1. Enter Parking Lot
2. Exit Parking Lot
3. Return to Main Menu
Enter your choice (1/2/3): 1
A VehicleType.CAR has entered the parking lot and parked in Compact Spot 1 on the 1 floor
Car Parked 1 floor 1 spot
User Menu:
1. Enter Parking Lot
2. Exit Parking Lot
3. Return to Main Menu
Enter your choice (1/2/3): 2
You used parking spot for 0.001 hours and the Parking Fee is $3.
```