Design a Parking Lot System

A parking lot or car park, also known as a car lot, is a cleared area intended for parking vehicles.

System Requirements

1. Support multiple floors.
2. Multiple Entry and exit points.
3. Customers Receive receipts at entry points and pay the fee at the exit points.
4. Each floor has many parking spots. Supports multiple types of parking spots (Compact, Regular, Large).
5. Support different types of vehicles. (Car, Truck, Van, MotorCycle).
6. Support per-hour parking fee model.

Main Actors:

1. **Admin:** Adding and modifying parking floors, spots, entrances, and exits.
2. **Customer:** Reserve and get a parking ticket.
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 assign and remove a vehicle from a parking spot.

Use Cases:

1. Add/Remove/Edit Parking Floor.
2. Add/Remove/Edit Parking Spot.
3. Add/Remove a Parking Attendant.
4. Take Ticket: Provide tickets to customers when entering the parking lot.

5. Scan Ticket: To scan a ticket to find out the total charge.
6. Payment: Cash/Credit Card.
7. Add/Modify Parking Rate.

Main Classes:

1. ParkingLot
2. ParkingFloor
3. ParkingSpot
4. Account
5. ParkingTicket
6. Vehicle
7. EntrancePanel and ExitPanel
8. Payment
9. ParkingRate
10. ParkingAttendant
11. CustomerInfoPortal
12. ElectricPanel

Implementation:

ENUMS and Constants:

```
public enum AccountStatus {
    ACTIVE,
    BLOCKED,
    INACTIVE
}
```

```
public enum ParkingSpotType {
    COMPACT,
    REGULAR,
    LARGE,
    ELECTRIC
}

public enum ParkingTicketStatus {
    ACTIVE,
    PAID,
    LOST
}

public enum VehicleType {
    CAR,
    ELECTRIC,
    MOTORCYCLE,
    VAN
}
```

Person and Address:

```
public class Address {
    private String streetAddress;
    private String city;
    private String state;
    private String zipCode;
    private String country;
}

public class Person {
    private String name;
    private Address address;
    private String email;
    private String phone;
}
```

Account, Admin and ParkingAttendent:

```
public abstract class Account {
    private String userName;
    private String password;
    private AccountStatus status;
```

```
    private Person person;
    protected ParkingLot parkingLot;

    public boolean resetPassword() {
        //logic
        return true;
    }
}

public class Admin extends Account{
    public boolean addParkingFloor(ParkingFloor parkingFloor) {
        parkingLot.addParkingFloor(parkingFloor);
        return true;
    }
    public boolean addParkingSpot(String floorName, ParkingSpot spot) {
        parkingLot.getParkingFloor(floorName).addParkingSpot(spot);
        return true;
    }

    public boolean addEntrancePanel(EntrancePanel entrancePanel) {
        parkingLot.addEntrancePanel(entrancePanel);
        return true;
    }
    public boolean addExitPanel(ExitPanel exitPanel) {
        parkingLot.addExitPanel(exitPanel);
        return true;
    }
}

public class ParkingAttendant extends Account{
    public boolean processTicket(String ticketNumber) {
        parkingLot.getTicket(ticketNumber).processTicket();
        parkingLot.removeActiveTicket(ticketNumber);
        return true;
    }
}
```

Vehicle and Types:

```
public abstract class Vehicle {
    private String licenseNumber;
    private final VehicleType type;
    private ParkingTicket ticket;

    public Vehicle(VehicleType type) {
        this.type = type;
    }
```

```
    public void assignTicket(ParkingTicket ticket) {
        this.ticket = ticket;
    }

    public String getLicenseNumber() {
        return licenseNumber;
    }

    public void setLicenseNumber(String licenseNumber) {
        this.licenseNumber = licenseNumber;
    }

    public VehicleType getType() {
        return type;
    }

    public ParkingTicket getTicket() {
        return ticket;
    }

    public void setTicket(ParkingTicket ticket) {
        this.ticket = ticket;
    }
}

public class Car extends Vehicle{

    public Car() {
        super(VehicleType.CAR);
    }
}

public class Electric extends Vehicle{
    public Electric() {
        super(VehicleType.ELECTRIC);
    }
}

public class MotorCycle extends Vehicle{

    public MotorCycle() {
        super(VehicleType.MOTORCYCLE);
    }
}

public class Van extends Vehicle{
    public Van() {
        super(VehicleType.VAN);
    }
}
```

ParkingSpot and Types:

```
public abstract class ParkingSpot {
    private String number;
    private boolean free;
    private Vehicle vehicle;
    private final ParkingSpotType type;

    public boolean IsFree() {
        return this.free;
    }

    public ParkingSpot(ParkingSpotType type) {
        this.type = type;
    }

    public boolean assignVehicle(Vehicle vehicle) {
        this.vehicle = vehicle;
        free = false;
        return true;
    }

    public boolean removeVehicle() {
        this.vehicle = null;
        free = true;
        return true;
    }

    public String getNumber() {
        return number;
    }

    public void setNumber(String number) {
        this.number = number;
    }

    public boolean isFree() {
        return free;
    }

    public void setFree(boolean free) {
        this.free = free;
    }

    public Vehicle getVehicle() {
        return vehicle;
    }
```

```
    public void setVehicle(Vehicle vehicle) {
        this.vehicle = vehicle;
    }

    public ParkingSpotType getType() {
        return type;
    }
}

public class RegularSpot extends ParkingSpot{
    public RegularSpot() {
        super(ParkingSpotType.REGULAR);
    }
}

public class LargeSpot extends ParkingSpot{
    public LargeSpot() {
        super(ParkingSpotType.LARGE);
    }
}

public class ElectricSpot extends ParkingSpot{
    public ElectricSpot() {
        super(ParkingSpotType.ELECTRIC);
    }
}

public class CompactSpot extends ParkingSpot{

    public CompactSpot() {
        super(ParkingSpotType.COMPACT);
    }
}
```

ParkingRate:

```
public class ParkingRate {
    private double rate = 20;

    public double getRate() {
        return rate;
    }

    public void setRate(double rate) {
        this.rate = rate;
```

```
        }
    }
```

ParkingFloor:

```
public class ParkingFloor {
    private String name;
    private HashMap<String, ElectricSpot> electricSpots;
    private HashMap<String, CompactSpot> compactSpots;
    private HashMap<String, RegularSpot> regularSpots;
    private HashMap<String, LargeSpot> largeSpots;


    public ParkingFloor(String name) {
        this.name = name;
    }

    public void addParkingSpot(ParkingSpot spot) {
        switch (spot.getType()) {
            case COMPACT:
                compactSpots.put(spot.getNumber(), (CompactSpot) spot);
                break;
            case LARGE:
                largeSpots.put(spot.getNumber(), (LargeSpot) spot);
                break;
            case REGULAR:
                regularSpots.put(spot.getNumber(), (RegularSpot) spot);
                break;
            case ELECTRIC:
                electricSpots.put(spot.getNumber(), (ElectricSpot) spot);
                break;
            default:
                System.out.println("Wrong parking spot type!");
        }
    }

    public void assignVehicleToSpot(Vehicle vehicle, ParkingSpot spot) {
        spot.assignVehicle(vehicle);
    }

    public void freeSpot(ParkingSpot spot) {
        spot.removeVehicle();
    }

    public void saveInDB() {
    }
```

```
    public boolean isFull() {
        return true;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public HashMap<String, ElectricSpot> getElectricSpots() {
        return electricSpots;
    }

    public void setElectricSpots(HashMap<String, ElectricSpot>
electricSpots) {
        this.electricSpots = electricSpots;
    }

    public HashMap<String, CompactSpot> getCompactSpots() {
        return compactSpots;
    }

    public void setCompactSpots(HashMap<String, CompactSpot>
compactSpots) {
        this.compactSpots = compactSpots;
    }

    public HashMap<String, RegularSpot> getRegularSpots() {
        return regularSpots;
    }

    public void setRegularSpots(HashMap<String, RegularSpot>
regularSpots) {
        this.regularSpots = regularSpots;
    }

    public HashMap<String, LargeSpot> getLargeSpots() {
        return largeSpots;
    }

    public void setLargeSpots(HashMap<String, LargeSpot> largeSpots) {
        this.largeSpots = largeSpots;
    }
}
```

ParkingTicket:

```
public class ParkingTicket {
    private ParkingTicketStatus parkingTicketStatus;
    private double cost;
    private Date createdAt;
    private Date releaseDate;
    private String ticketNumber;
    private ParkingRate parkingRate;

    public void saveInDB() {
    }

    public String getTicketNumber() {
        return ticketNumber;
    }

    public void processTicket() {
        this.releaseDate = new Date();
        this.cost = (releaseDate.getHours() -
createdAt.getHours())*parkingRate.getRate();
        this.updateInDB();
    }

    private void updateInDB() {
    }
}
```

Entrance and ExitPanels:

```
public class EntrancePanel {
    private String name;
    public void saveInDB() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

public class ExitPanel {
    private String name;
    public void saveInDB() {
    }
```

```
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

ParkingLot:

```
public class ParkingLot {
    private String name;
    private Address address;
    private ParkingRate parkingRate;

    private int compactSpotCount;
    private int largeSpotCount;
    private int regularSpotCount;
    private int electricSpotCount;
    private final int maxCompactCount = 100;
    private final int maxLargeCount = 100;
    private final int maxRegularSpotCount = 100;
    private final int maxElectricCount = 100;

    private HashMap<String, EntrancePanel> entrancePanels;
    private HashMap<String, ExitPanel> exitPanels;
    private HashMap<String, ParkingFloor> parkingFloors;
    private HashMap<String, ParkingTicket> activeTickets;

    private static ParkingLot parkingLot = null;

    private ParkingLot() {
        /* read data from database and assign variables*/
    }

    public static ParkingLot getInstance() {
        if (parkingLot == null) {
            parkingLot = new ParkingLot();
        }
        return parkingLot;
    }

    public synchronized ParkingTicket getNewParkingTicket(Vehicle
vehicle) throws ParkingFullException {
        if (this.isFull(vehicle.getType())) {
```

```
                throw new ParkingFullException();
            }
            ParkingTicket ticket = new ParkingTicket();
            vehicle.assignTicket(ticket);
            ticket.saveInDB();
            // if the ticket is successfully saved in the database, we can
    increment the parking spot count
            this.incrementSpotCount(vehicle.getType());
            this.activeTickets.put(ticket.getTicketNumber(), ticket);
            return ticket;
        }

        private void incrementSpotCount(VehicleType type) {
            if(type == VehicleType.VAN) {
                largeSpotCount++;
            } else if (type == VehicleType.MOTORCYCLE) {
                if(compactSpotCount < maxCompactCount)
                    compactSpotCount++;
                regularSpotCount++;
            } else if (type == VehicleType.CAR) {
                if(regularSpotCount < maxRegularSpotCount)
                    regularSpotCount++;
                largeSpotCount++;
            } else {
                electricSpotCount++;
            }

        }

        private boolean isFull(VehicleType type) {
            if(type == VehicleType.VAN) {
                return largeSpotCount>=maxLargeCount;
            }

            if (type == VehicleType.MOTORCYCLE) {
                return (compactSpotCount + regularSpotCount) >=
    (maxCompactCount + maxRegularSpotCount);
            }

            if (type == VehicleType.CAR) {
                return (regularSpotCount + largeSpotCount) >=
    (maxRegularSpotCount + maxLargeCount);
            }

            return electricSpotCount >= maxElectricCount;
        }

        public boolean isFull() {
            for (String key : parkingFloors.keySet()) {
                if (!parkingFloors.get(key).isFull()) {
                    return false;
```

```
                }
            }
            return true;
        }

        public void addParkingFloor(ParkingFloor floor) {
            floor.saveInDB();
            parkingFloors.put(floor.getName(), floor);
        }

        public void addEntrancePanel(EntrancePanel entrancePanel) {
            entrancePanel.saveInDB();
            entrancePanels.put(entrancePanel.getName(), entrancePanel);
        }

        public void addExitPanel(ExitPanel exitPanel) {
            exitPanel.saveInDB();
            exitPanels.put(exitPanel.getName(), exitPanel);
        }

        public ParkingFloor getParkingFloor(String name) {
            return parkingFloors.get(name);
        }

        public ParkingTicket getTicket(String ticketNumber) {
            return activeTickets.get(ticketNumber);
        }

        public void removeActiveTicket(String ticketNumber) {
            activeTickets.remove(ticketNumber);
        }
    }
```

So it’s just an overview we can further add the logic of assigning floors.

We can assign an entrance panel and exit panel in the ticket to show from where a car enters and exits respectively.

Also, we can assign a ParkingAttendent at the exit panel to process tickets.