Car Parking Summary

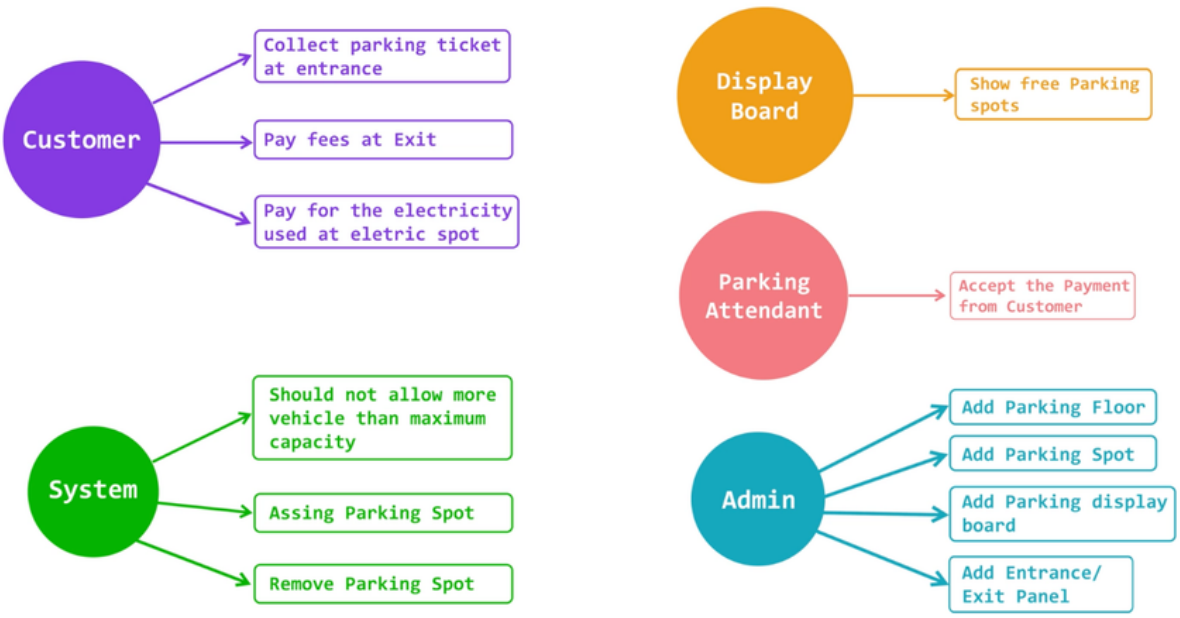
Introduction

Car Parking is a feature of all the big malls, hospitals, sports stadiums, mega-churches, and temples. So we need a car parking system to manage everything.

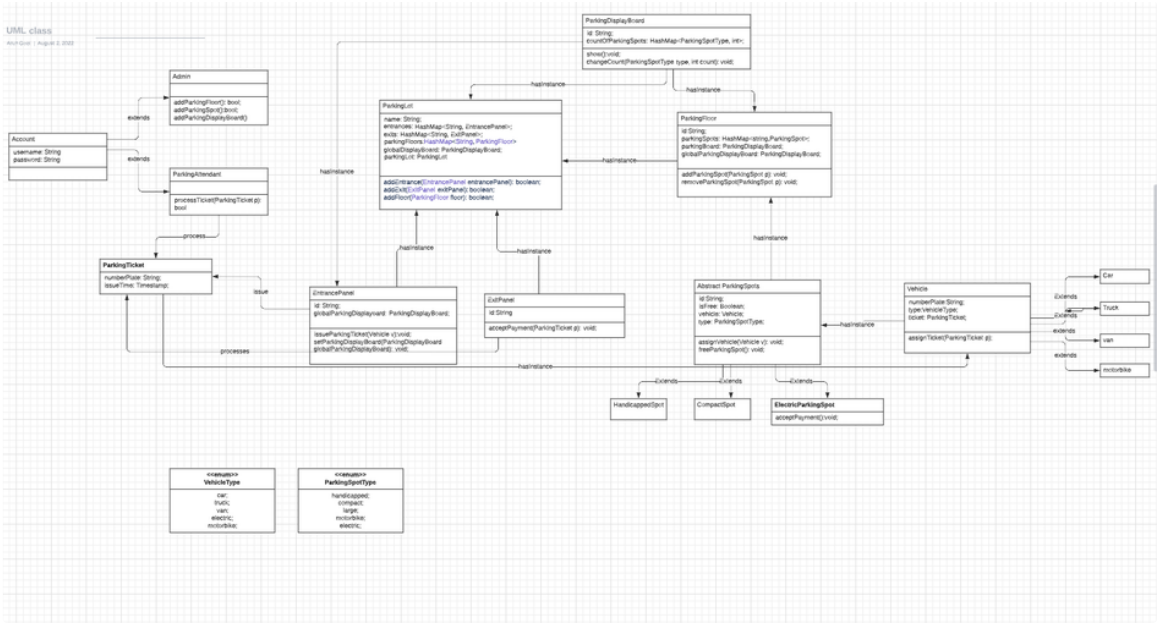
System Requirements

1. The parking lot should have multiple floors where customers can park their cars.
2. The parking lot should have multiple entry and exit points.
3. Customers can collect a parking ticket from the entry points and pay the parking fee at the exit points when they go out.
4. Customers can pay for the tickets at the automated exit panel or the parking attendant.
5. The system should not allow more vehicles than the maximum capacity of the parking lot. If the parking is complete, the system should be able to show a message on the entrance panel and on the parking display board on the ground floor.
6. Each parking floor will have many parking spots. The system should support multiple types of parking spots such as Compact, Large, Handicapped, Motorcycle, etc.
7. Each Parking Floor will have a display board showing the status of the parking spots.
8. The Parking lot should have some parking spots specified for electric cars. These spots should have an electric panel through which customers can pay and charge their vehicles.
9. The system should support parking for different types of vehicles like cars, trucks, vans, motorcycles, etc.
10. Each parking floor should have a display board showing any free parking spot for each spot type.
11. Admin can add parking floor, add parking spot, add parking display board to a floor, add entrance and exit panels.

Activity Diagram



Class Diagram



ENUMS:

```
public enum VehicleType{
  car, truck, van, electric, motorbike
}

public enum ParkingSpotType{
```

```
  handicapped, compact, large, motorbike,electric
}
```

Account:

```
public abstract class Account {
  private String userName;
  private String password;
}

public class Admin extends Account {
  public bool addParkingFloor(ParkingFloor floor);
  public bool addParkingSpot(String floorName, ParkingSpot spot);
  public bool addParkingDisplayBoard(String floorName, ParkingDisplayBoard displayBoard);

  public bool addEntrancePanel(EntrancePanel entrancePanel){ }
  public bool addExitPanel(ExitPanel exitPanel);
}

public class ParkingAttendant extends Account {
  public bool processTicket(ParkingTicket p);
}

```

PARKING LOT:

```
public class ParkingLot{
  private String name;
  private HashMap<String ,EntrancePanel > entrances;
  private HashMap<String, ExitPanel> exits;
  private HashMap<String, ParkingFloor> parkingFloors;
  private ParkingDisplayBoard globalDisplayBoard
  private static ParkingLot parkingLot = null;

  private ParkingLot(HashMap<String ,EntrancePanel > entrances,HashMap<String, ExitPanel> exit,HashMap<String, F
  private static getInstance(){
    if(parkingLot == null){
      parkingLot = new ParkingLot();
    }
    return parkingLot;
  }

  public boolean addEntrance(EntrancePanel entrancePanel){
    entrancePanel.setParkingDisplayBoard(this.globalDisplayBoard);
    entrances[entrancePanel.getId()]=entrancePanel;
  }
  public boolean addExit(ExitPanel exitPanel){}
  public boolean addFloor(ParkingFloor floor){}

}
```

PARKING TICKET:

```
public class ParkingTicket{
  private String numberPlate;
```

```
  private TimeStamp issueTime;
  public ParkingTicket( String number ){
    this.numberPlate= number;
    this.issueTime= Time.now();
  }
}
```

ENTRANCE PANEL:

```
public class EntrancePanel{
  private string id;
  private DisplayBoard globalParkingDisplayBoard;

  public void issueParkingTicket(Vehicle v){
     if( checkSpotAvailability(v) ){
       System.out.println("There is no availability");
     }
     else{
        v.assignTicket( new ParkingTicket(v.getNumberPlate()));
     }
     return;
  }

  public void setParkingDisplayBoard( ParingDisplayBoard globalParkingDisplayBoard ){
    this.globalParkingDisplayBoard= globalParkingDisplayBoard;
  }

}
```

EXIT PANEL:

```
public class ExitPanel{
  private String id;
  public void acceptPayment(ParkingTicket p){}
}
```

PARKING FLOOR:

```
public class ParkingFloor{
  private String id;
  private HashMap<String, ParkingSpot> parkingSpots;
  private ParkingDisplayBoard parkingBoard;
  private ParkingDisplayBoard globalParkingDisplayBoard;

  public ParkingFloor(HashMap<String, ParkingSpot> parkingSpots,ParkingDisplayBoard parkingBoard, ParkingDisplay

  public class addParkingSpot(ParkingSpot p){}
  public class removeParkingSpot(ParkingSpot p){}
}
```

PARKING SPOT:

```
public abstract class ParkingSpot{
```

```
  private String id;
  private boolean isFree;
  private Vehicle vehicle;
  private ParkingSpotType type;
  private ParkingFloor floor;

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

  public assignVehicle(Vehicle v){
    this.vehicle=v;
    this.isFree=0;
    floor.getparkingBoard().changeCount(type,-1);
    floor.getglobalParkingBoard().changeCount(type,-1);
  }

  public class freeParkingSpot(){
    this.isFree=1;
    vehicle=null;
    floor.getparkingBoard().changeCount(type,1); // short discussion here on how this could be an event to parki
    floor.getglobalParkingBoard().changeCount(type,1);
  }
}

public class HandicappedSpot extends ParkingSpot {
  super(handicapped); //short discussion on composition vs. inheritance
}

public class CompactSpot extends ParkingSpot {
  super(compact);
}

public class ElectricParkingSpot extends ParkingSpot {
  super(electric)
  public void acceptPayment(){}
}
```

Vehicle:

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

  public Vehicle(VehicleType type){
    this.type=type;
  }
  public void assignTicket(ParkingTicket ticket){
    this.ticket=ticket;
  }
}

public class Car extends Vehicle {
  public Car() {
    super(VehicleType.car);
  }
```

```
}

public class Truck extends Vehicle {
  public Truck() {
    super(VehicleType.truck);
  }
}
```

ParkingDisplayBoard:

```
public class ParkingDisplayBoard{
  private String id;
  private HashMap<ParkingSpotType , int> countsOfParkingSpots;

  public ParkingDisplayBoard(HashMap<ParkingSpotType , int> countsOfParkingSpots){}
  public void show(){}
  public changeCount( ParkingSpotType type, int change) { //short discussion on why we take change count and not
    countsOfParkingSpots[type]+= change;
  }
}
```