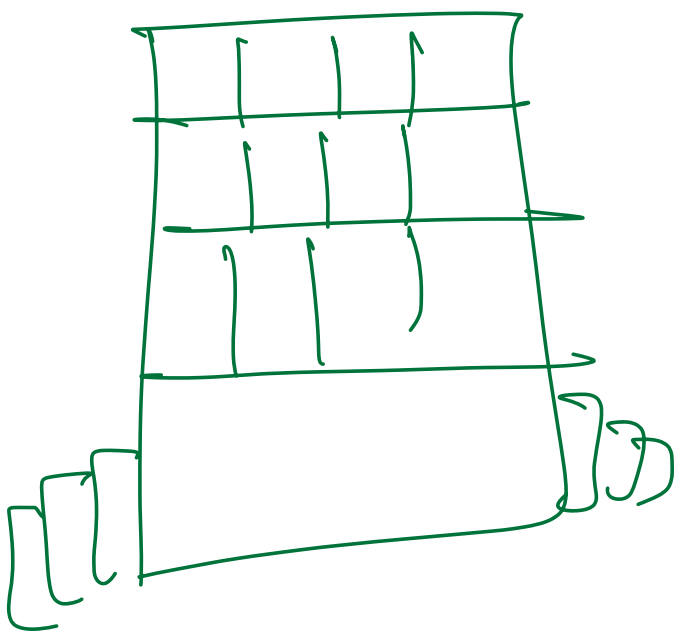
Design a Parking lot → VVI Popular Interview Q^n

① Overview
- Don't know
- Know → Practical

If you know about a system → what is parking lot
1. you briefly tell what you know
2. ask if you are thinking of correct system
3. follow up q^n
   - a.) entity v/s S/W System
   - b.) persistence or not → persist in memory
   - ⇒ c.) how input will be given → hard code in main file



② Requirement Gathering

→ Suggest ideas / features

→ Visualize

→ user journey

→ physical structure

→ Only closed parking lots → Can have multiple floors

↳ Open parking lot is one with 1 floor.

NOTE:

affects design

- a good ques is (1) or many
- ✗ not how many

Parking lot {

Vehicle

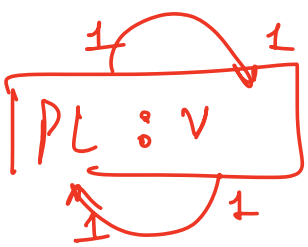
}

V/S

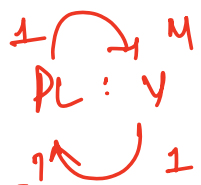
Parking lot {

List<Vehicle>

}

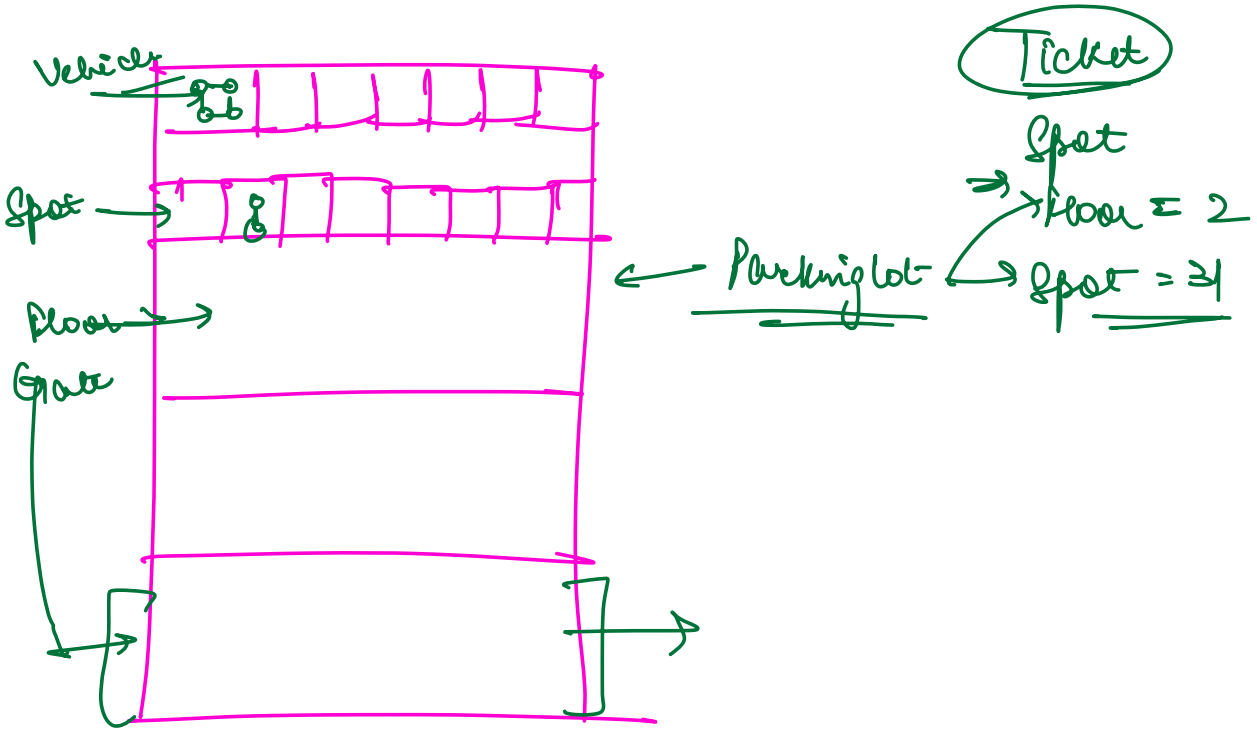


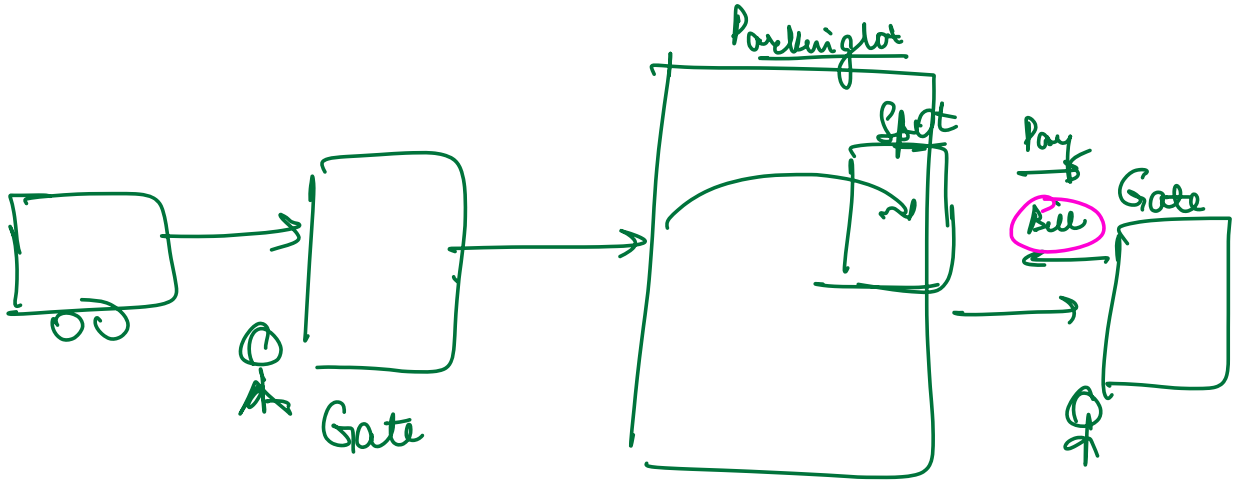
⇒ 1 : 1



1 : M

1. Multiple floors
2. Diff types of vehicles.
3. Multiple entry and exit gates for a parking lot
4. Parking lot Mgmt system should allow strategy to easily change how the fee is calculated. (type | time | type + time | dynamic)
5. Each floor will have multiple parking spots.
   - Each spot can be dedicated to a set of vehicles (CAR, SUV)
   - You aren't allowed to merge small → big
6. At the time of entry, a spot is assigned to the vehicle.
7. Spot is released at the time of exit.
8. We have to allow a parking lot to be flexible with how they assign a spot.
9. Human Operated Parking lot Mgmt System. (Clarification)
10. For today, login / logout etc is out of scope.





Dynamic

6 – 9 PM ⇒ Costly

9 – 11 AM ⇒ costly

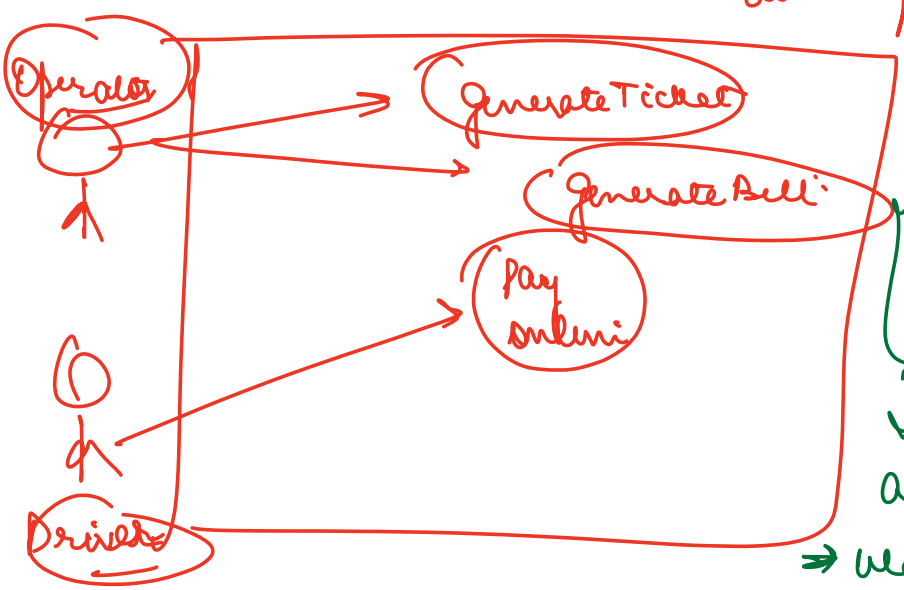
11 AM – 6 PM ⇒ cheap

(11) Our system support both online and offline payments.

(12) No pass system

→ Link is shared by operator and you pay there

→ We dont have to store any info about payments in our system



→ Just store 3rd party ref NO.

Adapter

→ We just have to store ref No.

Break till 10:25PM

→ Class Diagram

→ Start Scenario

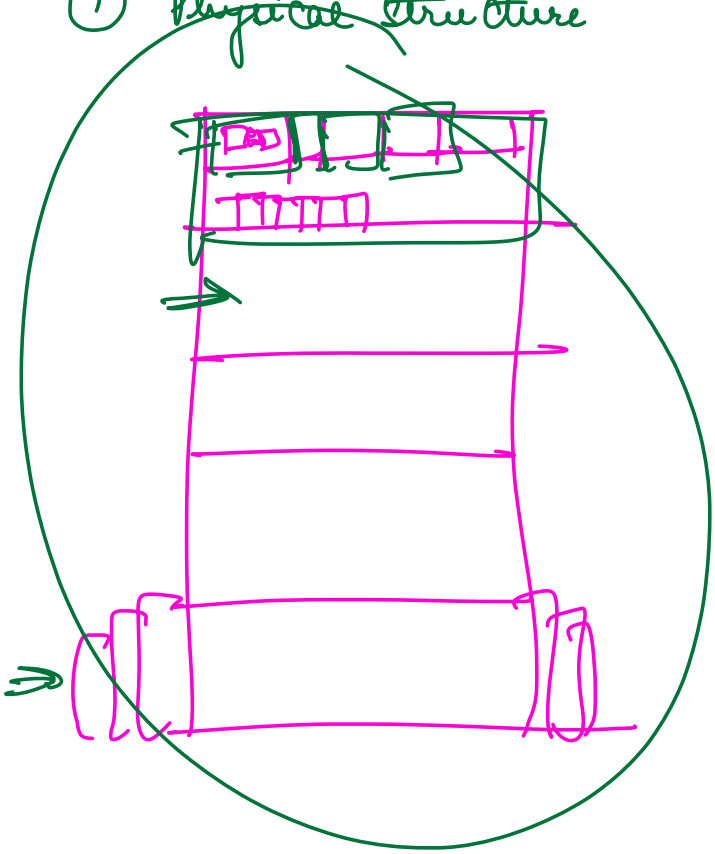
CLASS DIAGRAM

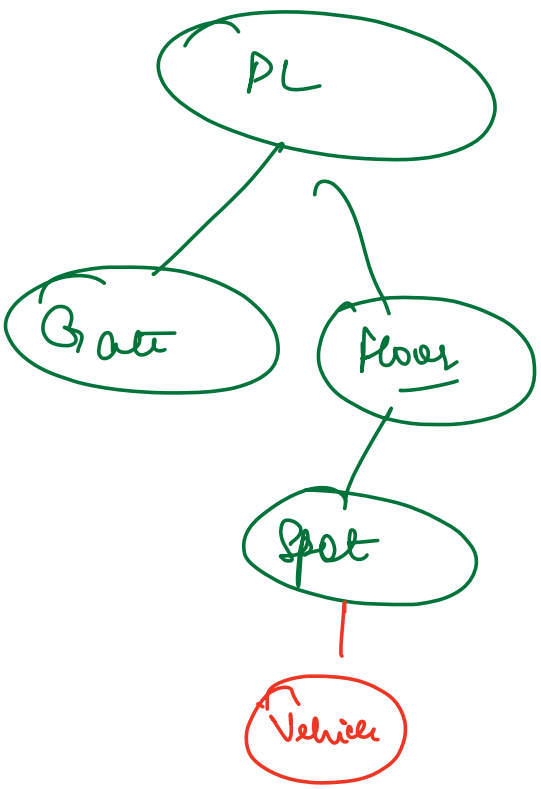
1. Visualization → Today
2. Nouns → Splitwise

Nouns is better when you already have req → MC Rounding

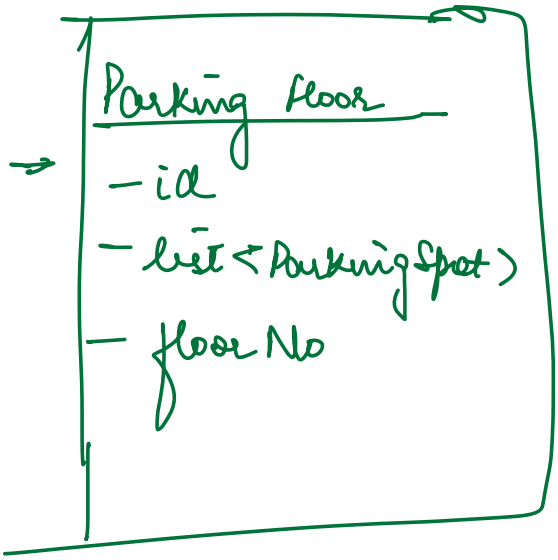
Visualization

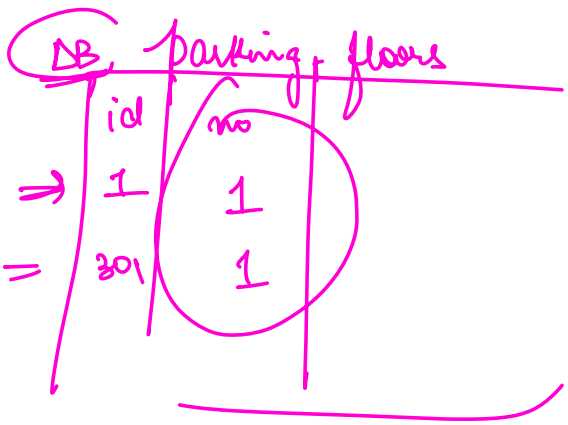
1. Physical structure





⇒ When we are building a s/w system, every entity will also have an Id attribute

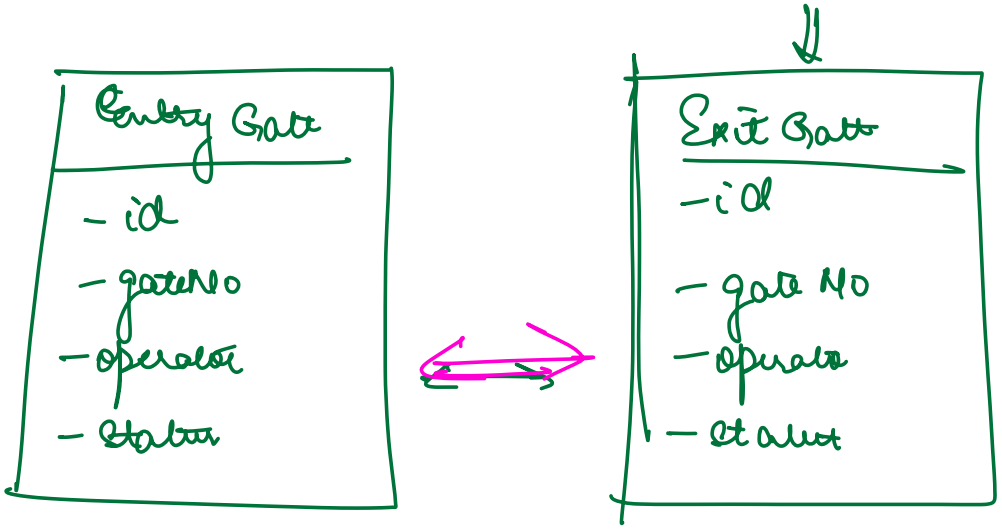




Entry Gate EntryGate ① Yes Sol

V/S

Gate ② No Sol

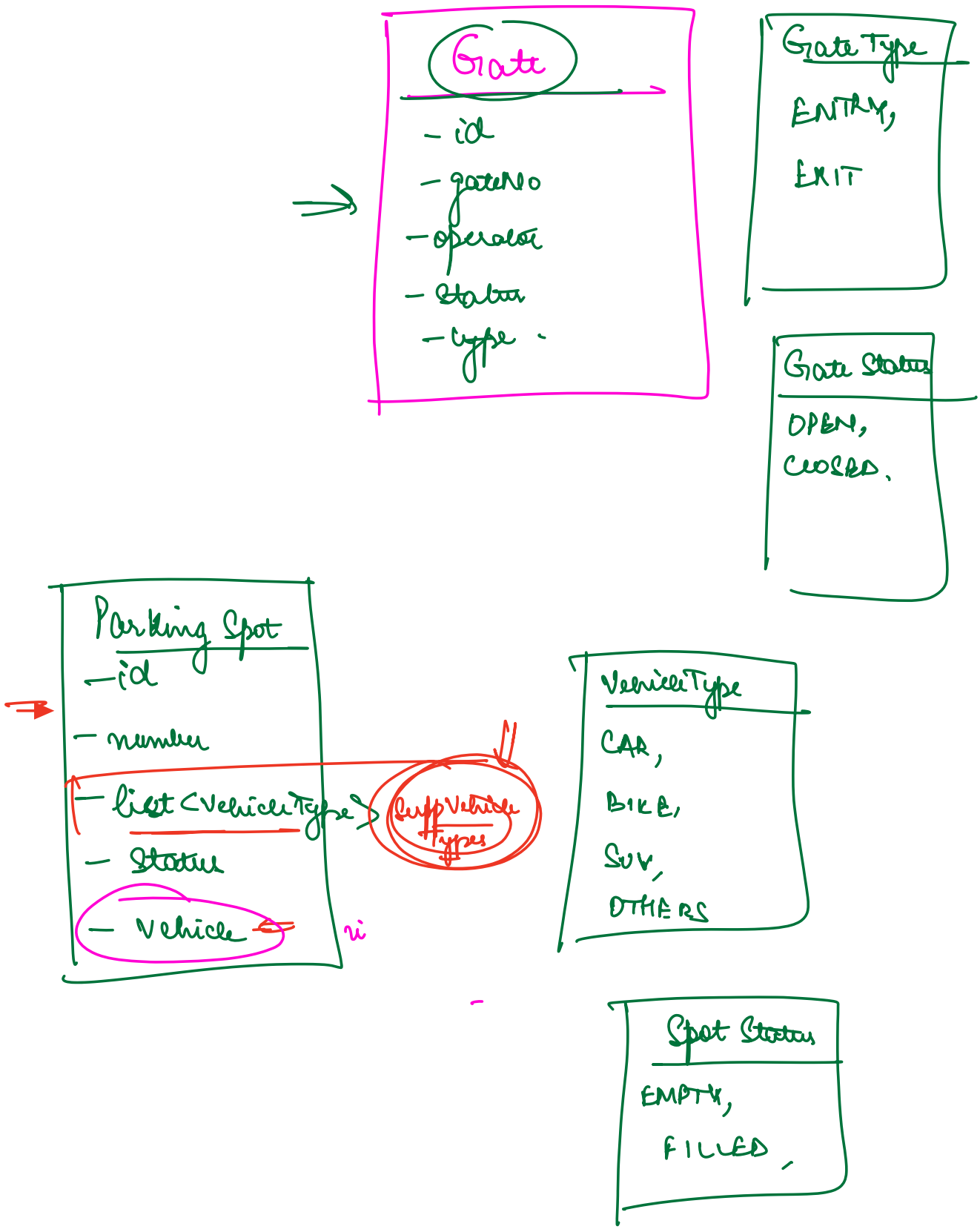


⇒ attrs will be same

Only diff will be methods

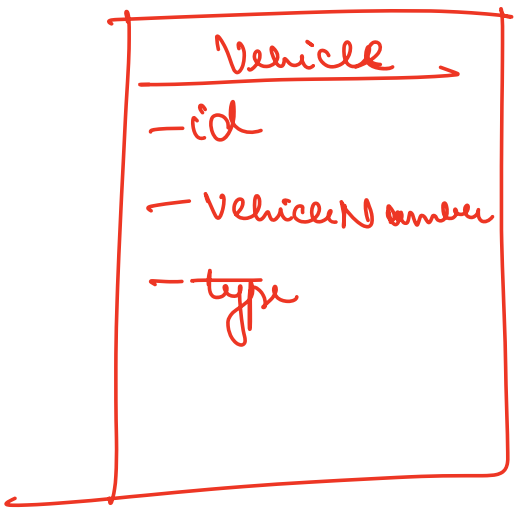
⇒ In a real system, functionality is implemented in Services

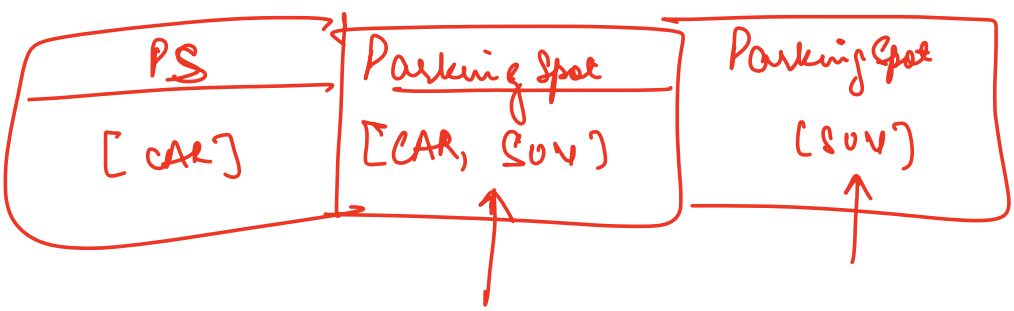
models are just data (only attributes)

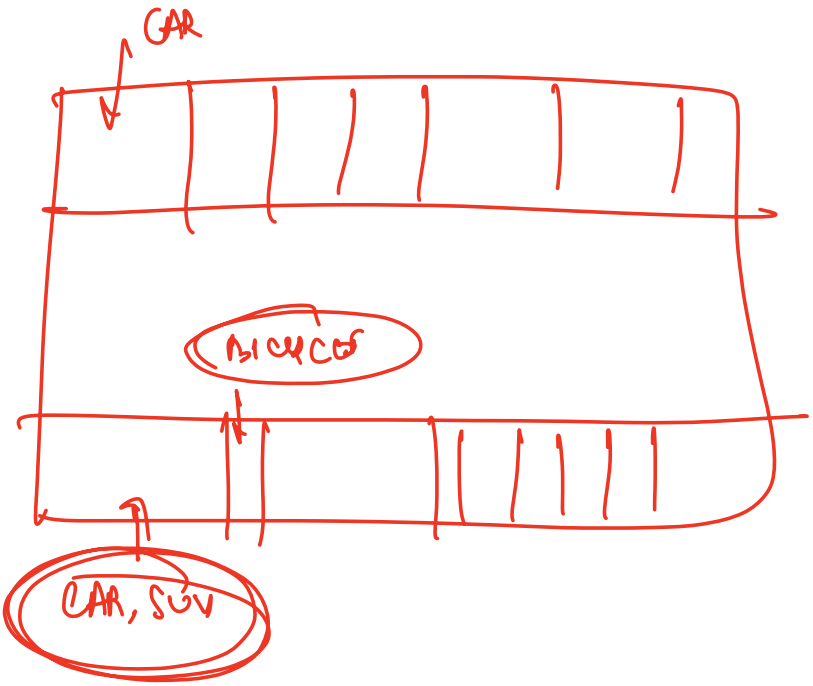

Gate
- id
- gateNo
- operator
- Status
- type
GateType
ENTRY,
EXIT
Gate Status
OPEN,
CLOSED.
Parking Spot
- id
- number
- list<VehicleType> suppVehicle Types
- Status
- Vehicle
VehicleType
CAR,
BIKE,
SUV,
OTHERS
Spot Status
EMPTY,
FILLED,

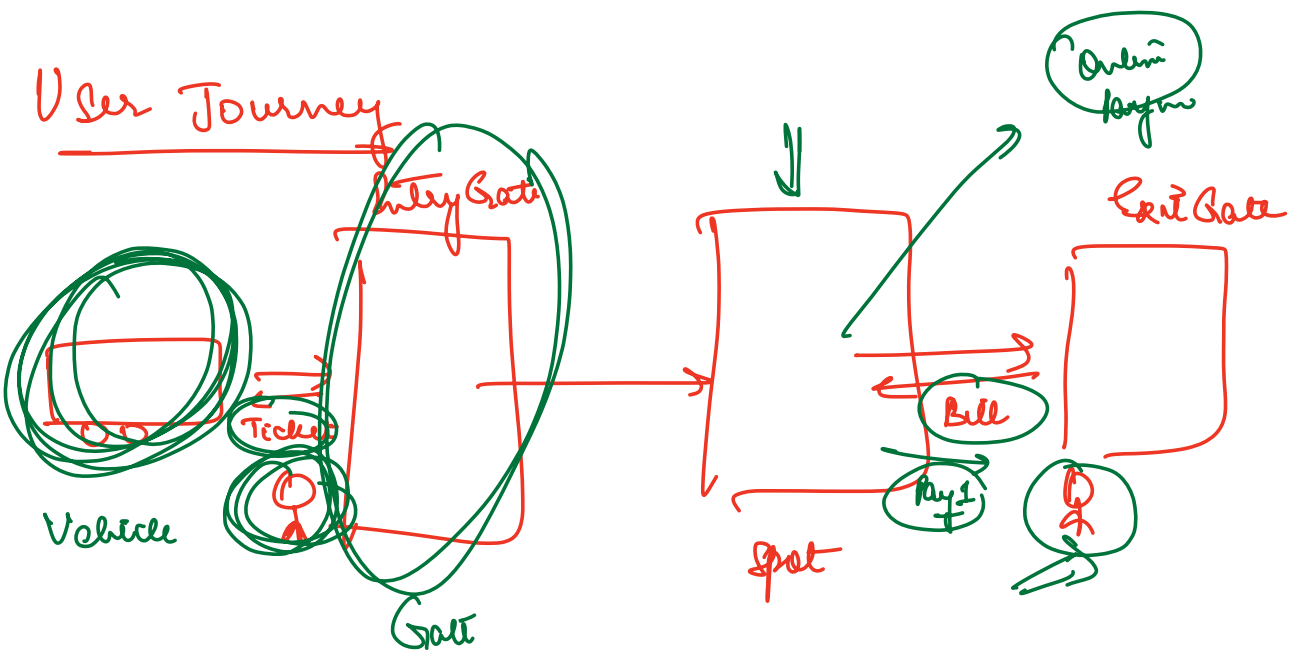
⇒ When we are doing Class diagram, classes rep assoc^n bw entities. Thus, classes must have the object of other class.

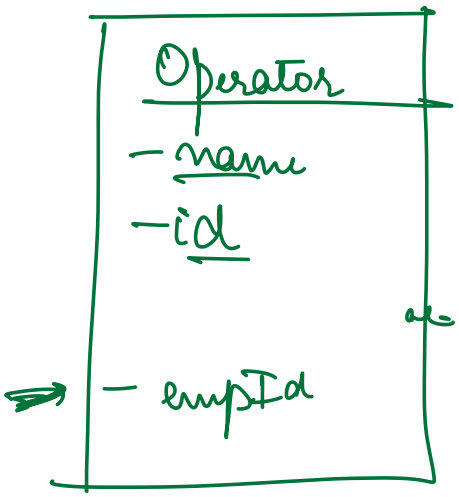
⇒ ORM automatically replaces that with id when storing in database.

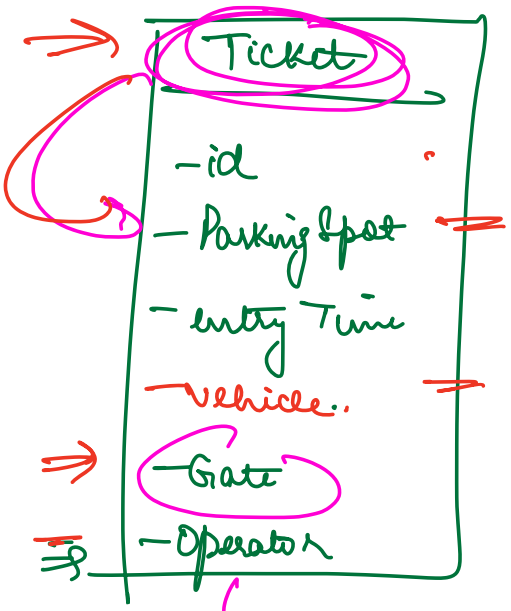





CAR
BICYCLE
CAR, SUV

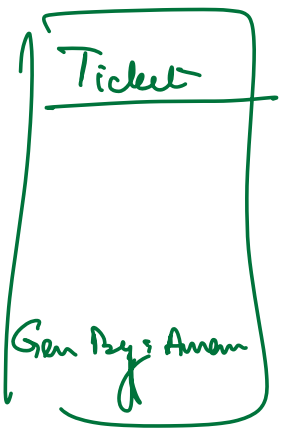

User Journey
Entry Gate
Ticket
Vehicle
Gate
Spot
Bill
Pay
Exit Gate

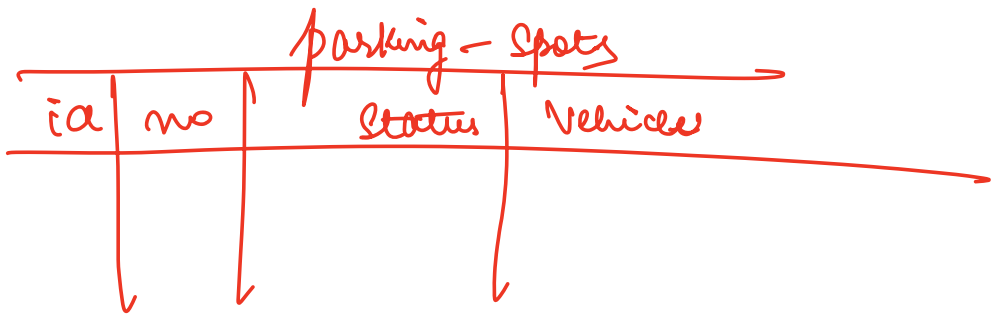




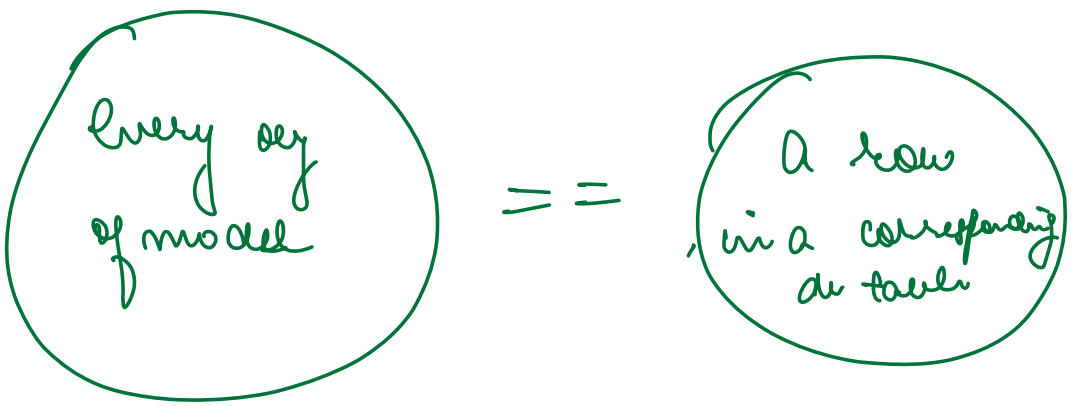
person who gen the ticket

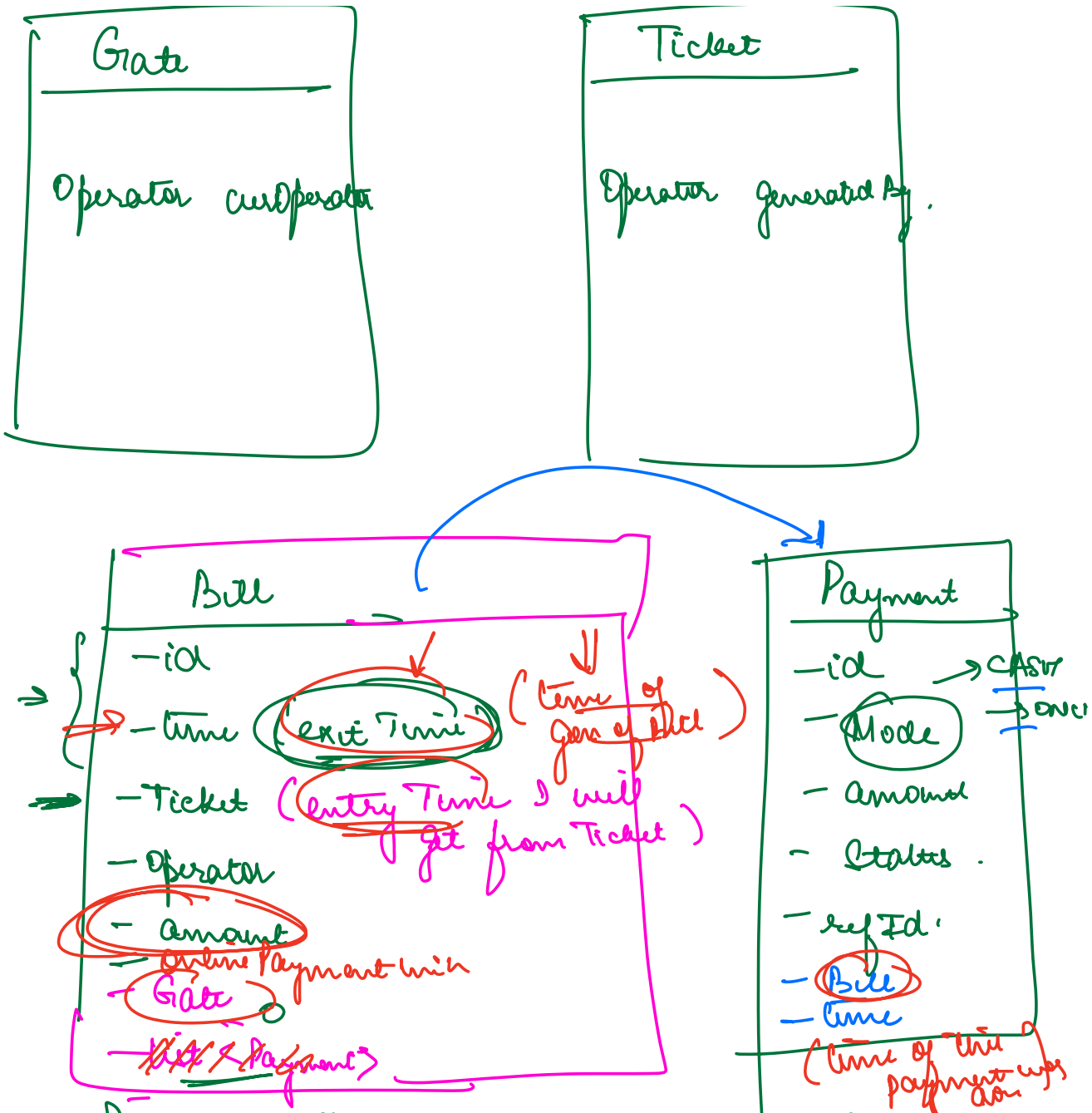
Gate → will have operator present there currently which may not be same as the one who gen it.






```
update ps
    set status = AV
    and vehicle = null
where id = 123
```





⇒ Payment should be a sep class in itself.

⇒ That class should have info like Mode, refId, Status.

⇒ Whenever you are associating payment with an entity, always associate a List<Payment>

↳ Failed Payment

↳ Partial Payment