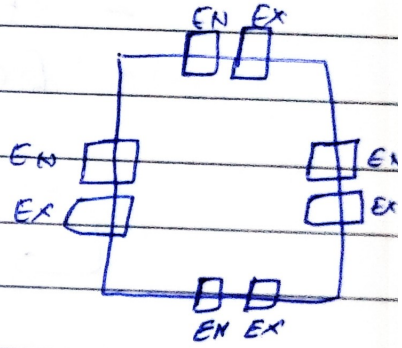
22 June 2023

Parking lot System Design

1. Clarify all the Requirements
2. Object oriented design skill like Object & Classes.
3. OOPS
4. links b/w Classes & Objects
5. Design Patterns

- Tackle Concurrency
- Problem Solving skills
- Simplify your problem & don't complicate it.



Requirements Collation

- Big Parking lots - 10K - 30K
- 4 entrances & 4 exits
- Ticket & parking spot allotted at entrance.
- Parking Spot should be nearest of the entrance.
- Limit / capacity - 30K
- Parking Spot - HC, Large, Compact, motorcycle.
- Hourly rate
- Cash & Credit Cards Accepted
- Monitoring System
- Your System should be installed at different parking lot like Shopping Mall, School, College parking lot with minimal Changes required in your System. You should just need to change the configuration and nothing major change required. - This is code re-usability.

- Don't discuss any unspecified requirements that is not necessary.

We design our parking lot System Using Design patterns.

There are 3 broad Categories of design patterns

- Creational Design patterns
- Structural Design patterns
- Behavioral Design patterns

→ Deals with how the objects are instantiated

→ Deals with how different objects and classes are composed in order to form larger structure.

→ Deals with the possibilities of the objects and how they interact with each other.

There are two different approach for designing a System.

→ Top-Down Approach
→ Bottom-up Approach.

Top Down Approach ⇒ high level design to it's Subcomponent and then again it's Subcomponent and so on & so forth

Bottom up Approach ⇒ we first design a smaller Sub-component & using that Subcomponent we design a bigger Sub-component & so on & so forth.

→ This is aligned with object oriented design. So prefer this.

→ Identify different Actors & Objects in our System.

1) Parking lot System
2) Entry / Exit terminals
   - ↳ Printer
   - ↳ Payment Processor
3) Parking Spot
4) Ticket
5) Database
6) Monitoring System

Note:- Here Vehicle & it's Subclasses like HC, Compact, Large, Motorcycle are not Actors as using the vehicle class didn't help us in designing Parking lot system. instead parking spot ha

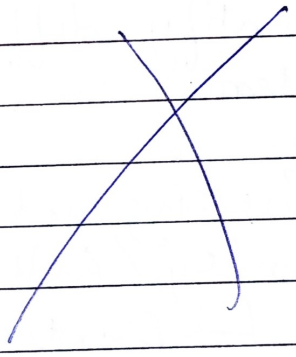
Now let's design different interfaces, classes & components in our parking lot system using bottom-up Approach.

(1) Parking Spot

- H C
- Compact
- Large
- Motorcycle

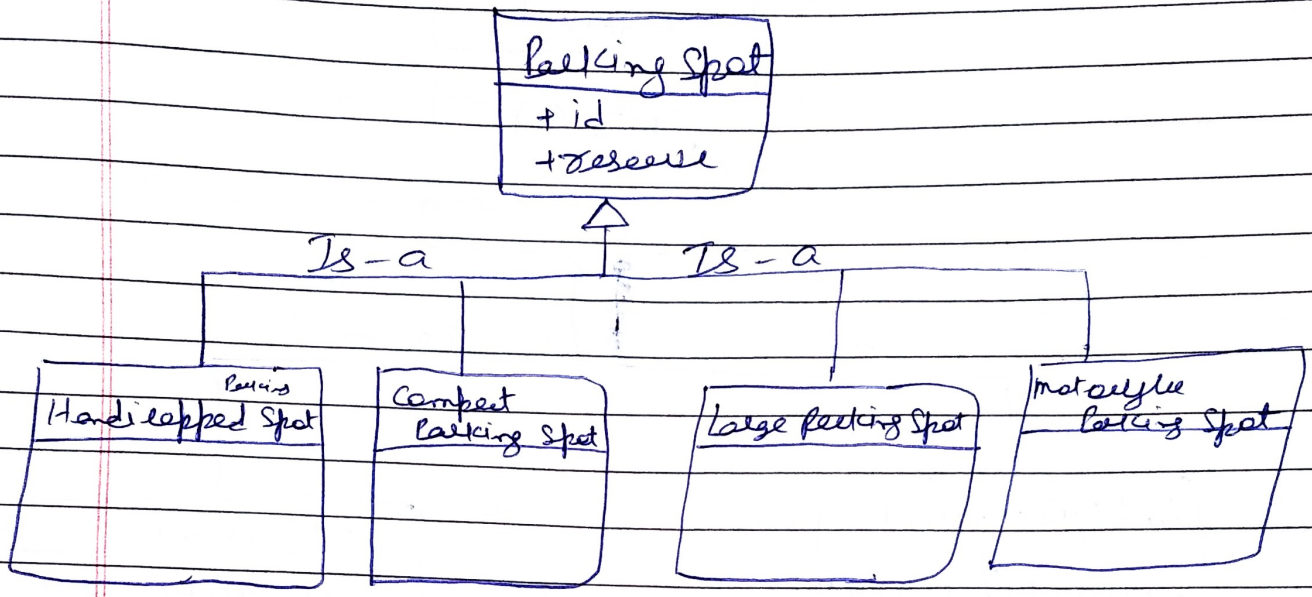
Method 1

```
enum ParkingSpot {
    handicapped,
    compact,
    large,
    motorcycle
}
```



This is a bad design b/c if you want to add/delete a parking spot then it will requires a lot of changes in the code which will violate the Open/close design which says that existing & well tested classes should not be modified while new features need to be built.

Second Approach



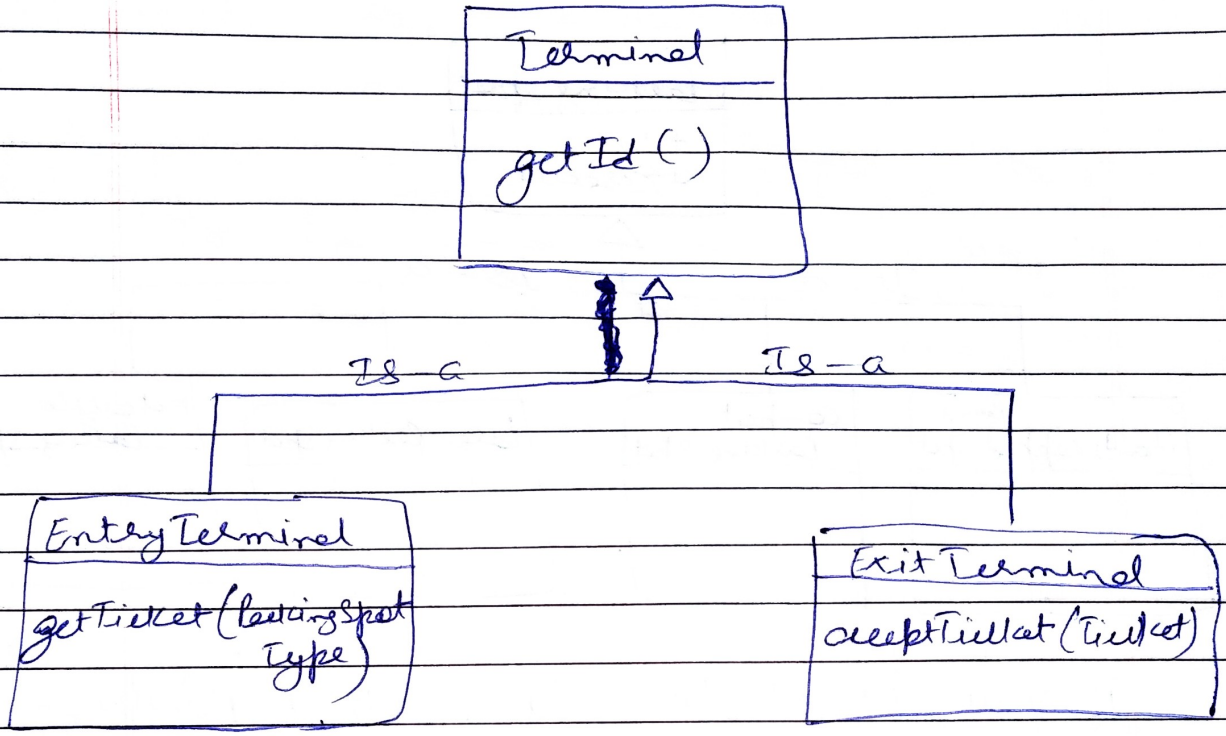
if you want to add another parking spot, then we just need to extend another type of parking spot with the parent parking spot class.

Note: Parking spot class will be an abstract class that means we cannot intantiate an object of a parking spot, we could instantiate an object of handicapped parking spot, a large or compact or motorcycle parking spot but not the parent parking class object.

(2) ParkingTicket

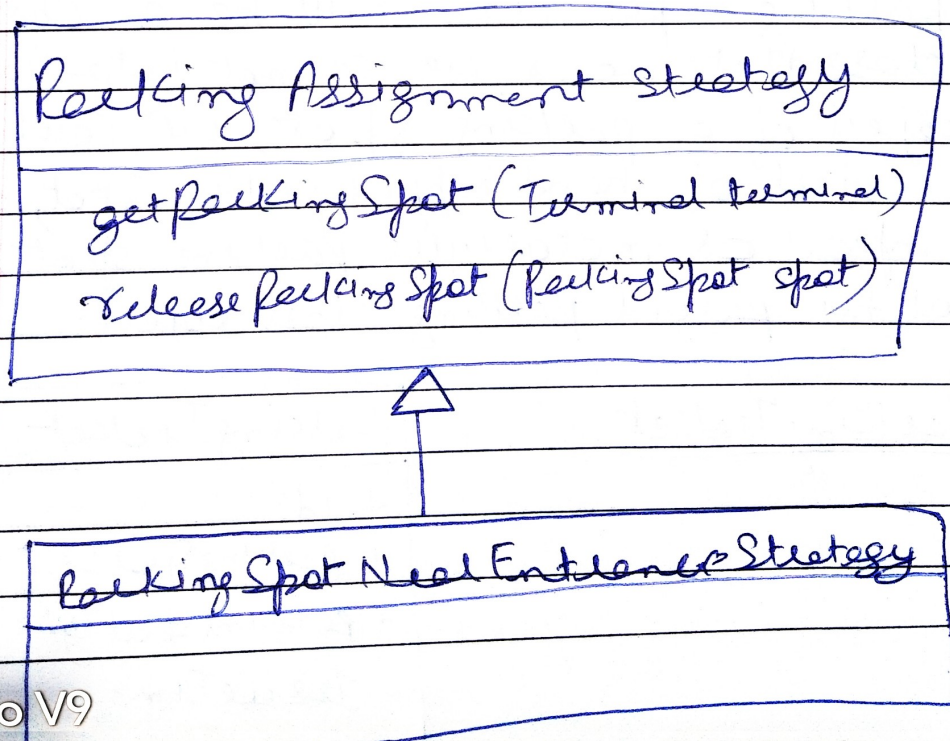
ParkingTicket
+ id
+ ParkingSpotId
+ parkingSpotType
+ IssueTime

(3) Entry / Exit Terminal



Note: These are either abstract classes or Interfaces which we we will be implement or by the actual class object

(4)



A big part of Interview will deal with assignment of the parking spot and then based on the requirements the design would change.

For ex.

(a) How will you assign a parking spot when you have single Entrance.

(b) How will you assign a parking spot when you have two or more entrances.

(c) How will you implement a parking lot when you have no such requirement of nearest parking spot.

(d) How ——— requirement of nearest parking spot from the entrance.

In our case we have 4 entrance & will try to return the nearest parking spot to the entrance from where the customer is entering the parking lot.

Using minHeaps.

# of minHeaps = # of Entrances

— 4 minHeaps
↳ will store the parking spot in order of distance from the entrance.

2 Sets

→ 1 Set – Available parking Spots
→ 1 Set – Reserved parking Spots

map < entrance_id, minHeap >

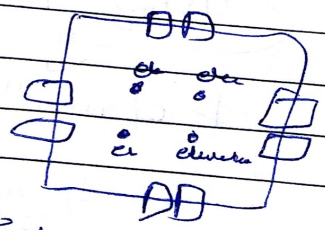
↳ We need to keep this mapping for each entrance.

So according to our parkingAssignmentStrategy we have a method getParkingSpot(TerminalVehicle)
So we get the terminal id (entrance_id) & from this id we fetch the minHeap from the map. & return that parking spot to the customer
Also we should remove that parking spot from available to Reserved Set & also we need to remove that parking spot from other minHeaps.

Completity → K log(n)

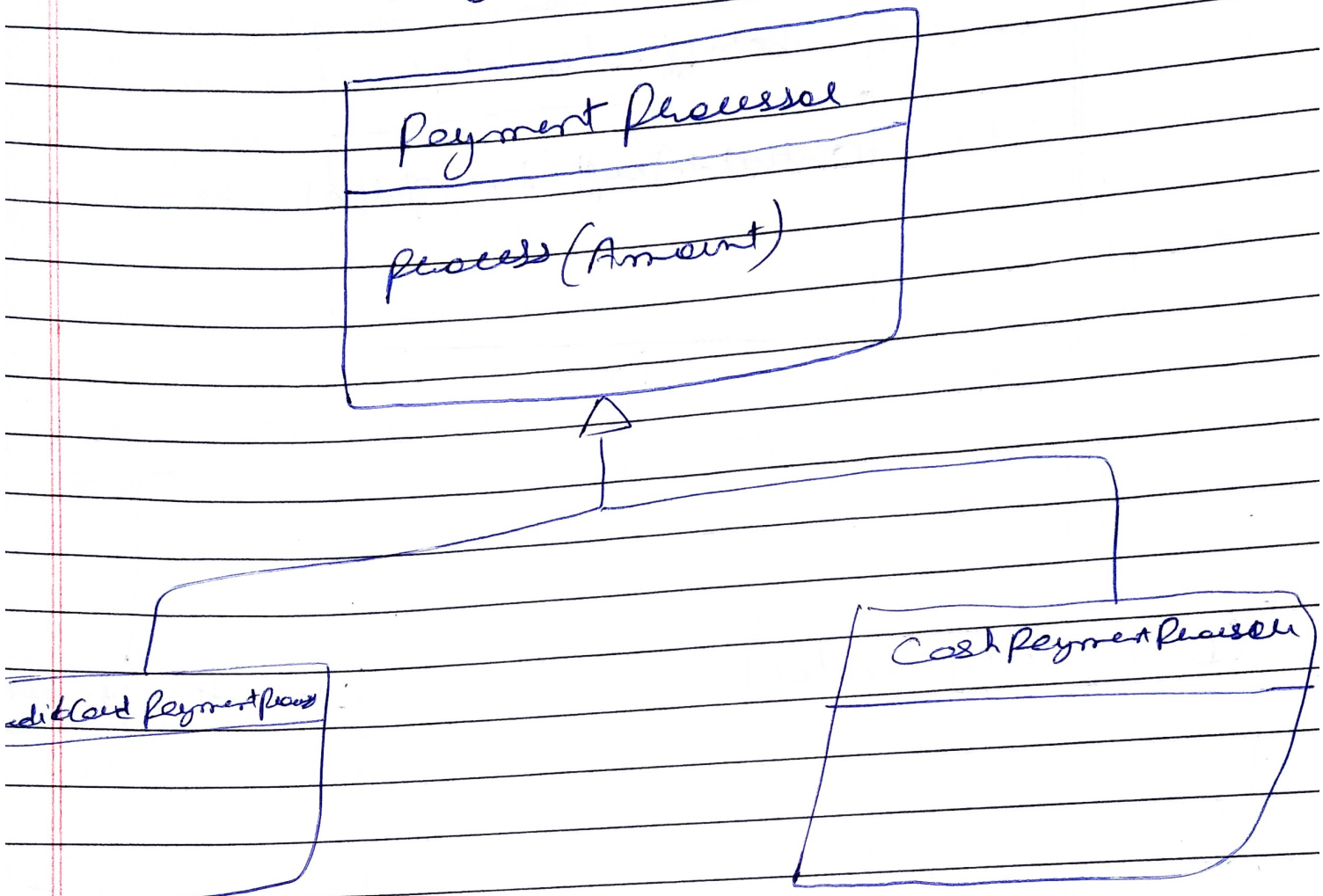
K → # of minHeaps
n → # of Parking Spots.

Question :- Let's there are 4 elevators also in our parking lot & requirement is return the parking spot which is near to the entrance and also near to the elevator.



Sol :- It will be a trade off b/w this two So we need to give a priority out of this two. I will use two minHeap one for entrance & one for elevator & based on priority I will give the parking spot.

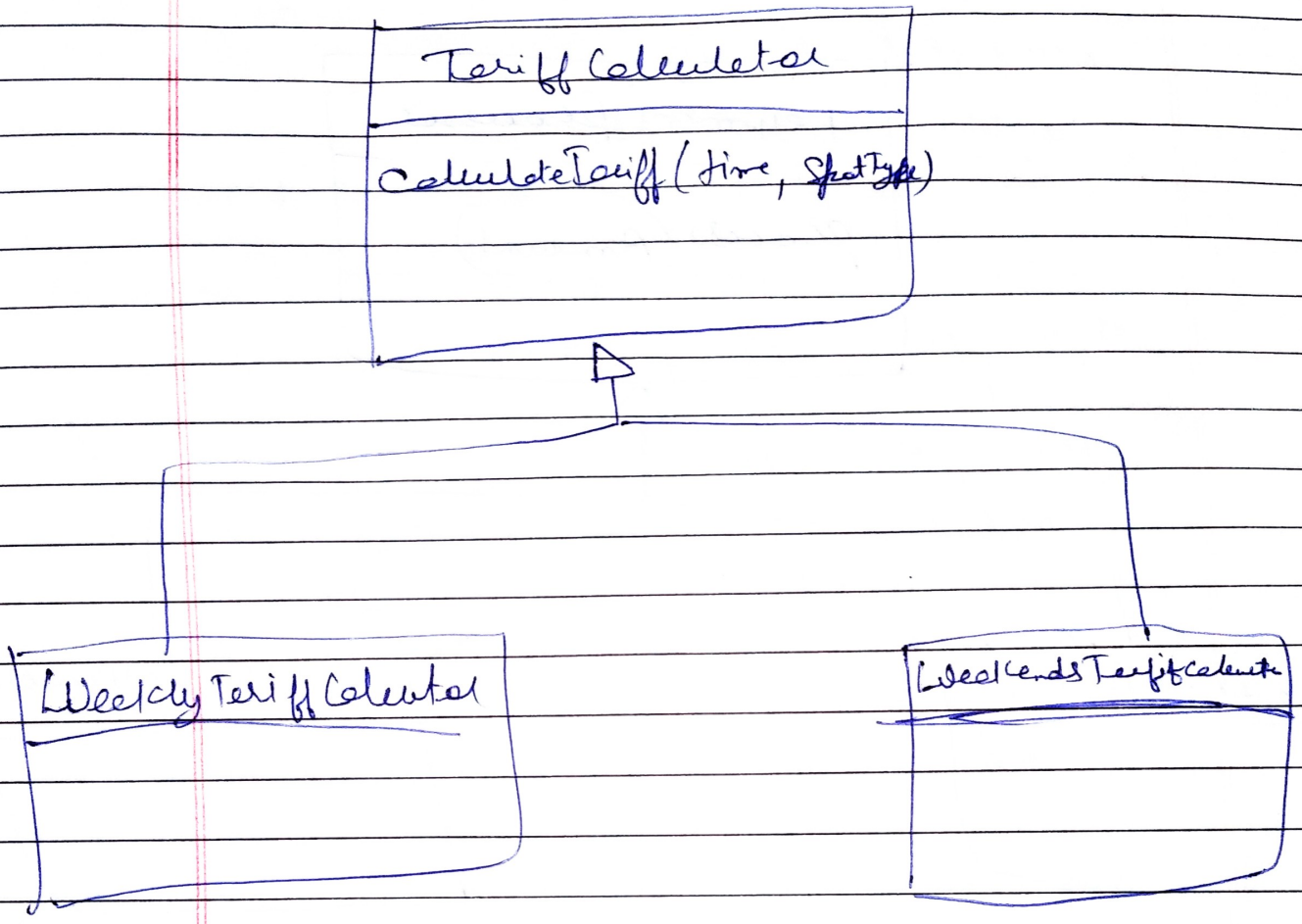
5 Payment Processor



So we will be using Strategy design pattern to implement this.

If in future we add applepay then we can easily add this using Strategy Design pattern.

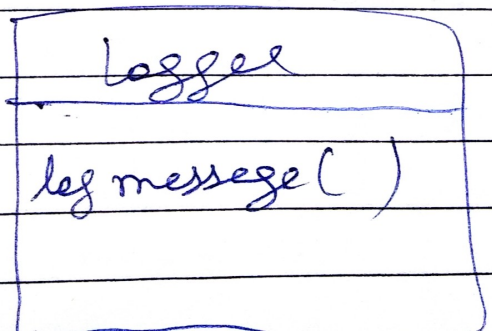
6 Tariff Calculator



7 Monitoring System

↳ It uses observer Design pattern

Monitoring system has a logger.



26 June 2022

LSD Intro

Since we are using bottom-up approach design
So first we will be discussing design of individual objects in the parking lot systems & now we will construct the whole parking lot systems using the small components.

So first of all parking lot System will have parking lot class which will be using singleton design pattern becoz there will be only one instance of parking lot System itself.

Now this parking lot System will be using the small components that we already discussed before like

- Entry/Exit Terminal
- Parking Spot
- Payment Processor
- -
- -
- -

& we will be using factory design pattern to instantiate all those objects.

Now the implementation of parking lot System class will take configuration object as input & this configuration object may have

Requirements of the

different value depending on the parking lot System. For ex. # of parking Spot, type of parking spot in it, etc - & then we can use abstract factor // factory design patterns to instantiate those objects & link them together.

So in the case in our parking lot System we will have a set of terminals that we will actually instantiate. i.e the set of entry & exit terminals the will instantiate, ~~etc~~. we will also instantiate a parking assignment strategy & will pass it in a configuration object & from the configuration object the strategy ~~instantiate~~ instance will actually retrieve all the parking spots along with the parking terminals and the distance of the parking spots from each entering terminal etc. We will also instantiate the different payment processors and we will pass those payment processors to the exit terminals bcoz this is the place where we would be processing the payment. we will also instantiate printers based on the from the Configuration. based on the printers in each ~~of~~ exit entry terminal & we will instantiate those objects & we will pass those objects to those terminal instance as well. So this is how overall System should be designed.

We can handle concurrency in the strategy class using synchronization mechanism like locks etc.