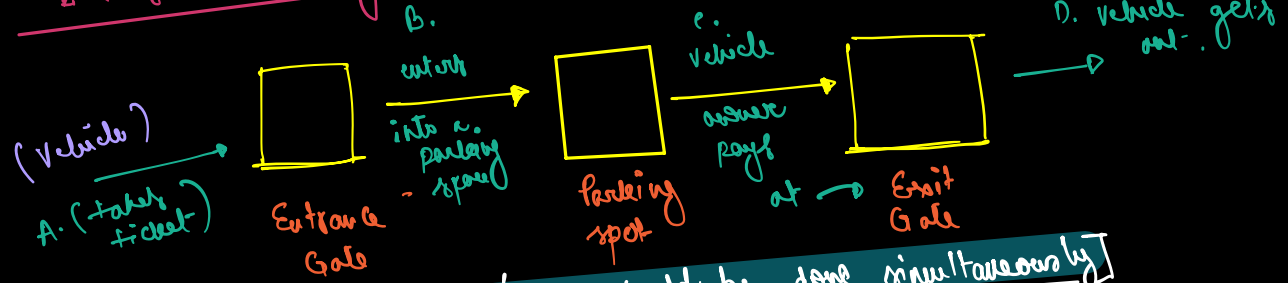
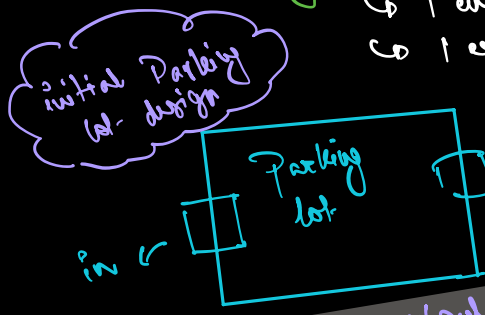
A. Rough functionality flow



(Step B & C should be done simultaneously)

B. Requirement clarification

1. How many entrances do we have?
   - 1 entrance (current scope)
   - 1 exit



[Note: Our code should be scalable enough to handle multiple entrance and exits]

2. How many different types of parking spot can there be? or in like based on Vehicles wheel
   - 2 wheeler (current scope)
   - 4 wheeler

[Note: It should be extensible for 3 wheeler and heavy vehicles like truck, bus etc.]

3. What are the payment strategy?
   - Hourly based or minute based?
   - mixture of both.
4. parking spot space should be marked.
5. How many floors? → 0

C. Object identification

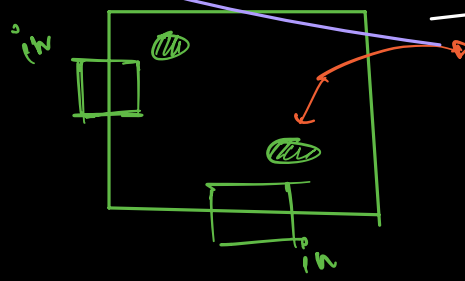
1. Vehicle
2. Ticket
3. Entrance Gate
4. Parking Spot
5. Exit Gate

D. Define high-level functionality and attributes of the above objects

1. Vehicle (Enum)
   - Vehicle No
   - Vehicle Type → 2 wheeler, 4 wheeler
2. Ticket
   - entry time
   - Vehicle
   - Parking Spot
3. Entrance Gate
   - find an empty parking spot
   - update parking spot space
   - Generate ticket
4. Parking spot
   - id
   - type [2 wheeler spot or 4 wheeler spot]
   - price
   - isEmpty
   - Vehicle
5. Exit Gate
   - cost calculation
   - payment
   - update parking spot space.

- there can be follow up questions from interviewer - to make it more challenging like for example ;

Entrance Gate
↳ find an empty parking spot space.

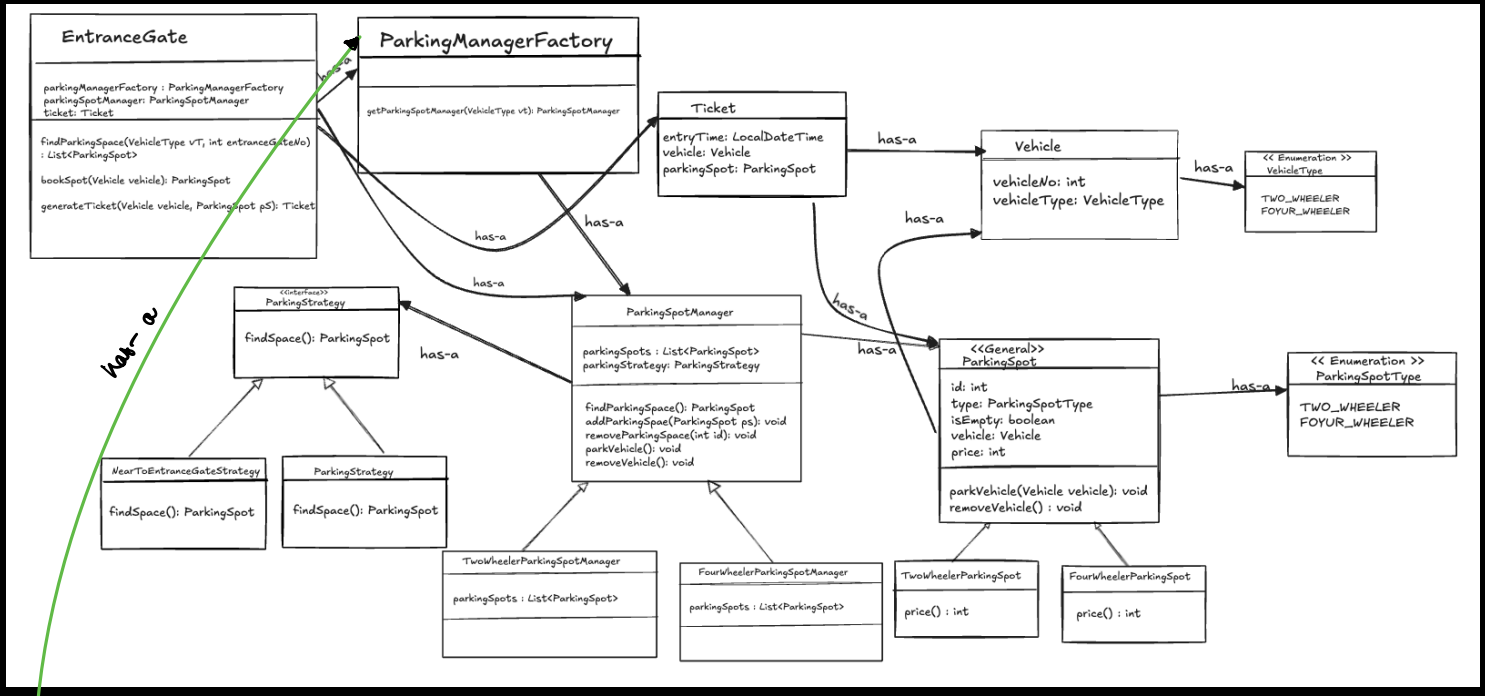


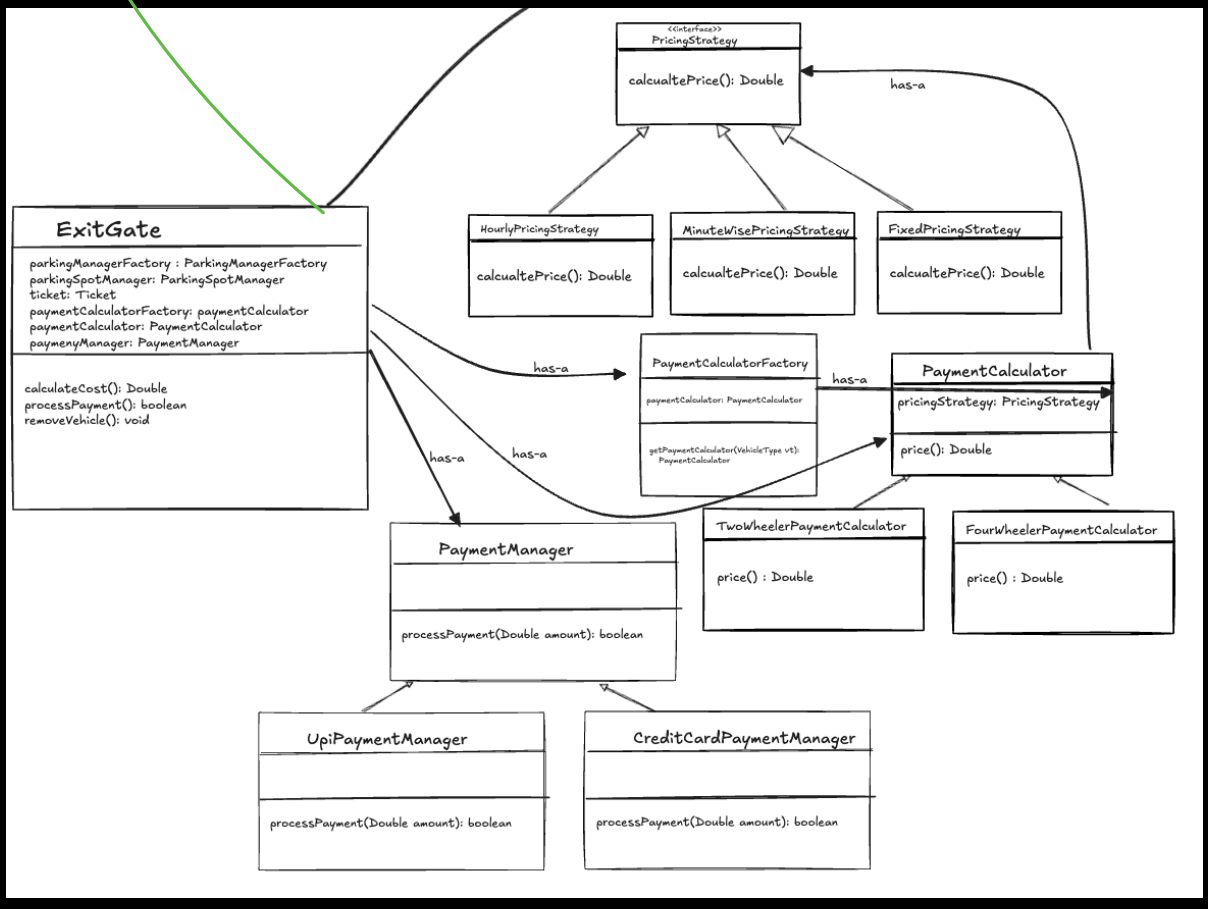
FollowUp Question

design in such a way so that we can get nearest parking spot space.

- Solution
  - A. Top-down (Start from entrance gate to all the way upto exit gate)
  - B. Bottom-Up (2 Approaches)
    - ↳ first develop parking space.
    - ↳ then develop the object which requires parking space and so on

- BottomUp
  - A. Draw class Diagram





Observations

- Whenever there are multiple behaviours possible there we are calling the strategy design patterns
- Whenever there are multiple objects that can be instantiated, there we are using factory design pattern