# APPROACH TO SOLVE LLD PROBLEMS

## AGENDA

- Interview Process:
  - Identify the requirements
  - Model the problem
  - Establish the classes and their relationships
  - Sequence and activity diagrams
  - Use design patterns
  - Code
- Design Approach:
  - Top-Down vs Bottom-Up Approach

## IDENTIFY THE REQUIREMENTS

- Gather system requirements
- What interviewer looks for?
  - How good are you at collecting requirements?
  - Are you able to effectively scope down the problem?
  - Can you produce a design based on your requirements within?

## MODEL THE PROBLEM

- Identify primary use cases and talk about it with interviewer.
- Think out loud.
- Interviewer asks you to sketch use case diagram.
- Revise different components – systems, actors and use cases.

## ESTABLISH CLASSES AND RELATIONSHIPS

- Example:
  - Parking lot: vehicles, parking spots, entrances, exits etc.
- Establish attributes and operations for each object.
- Nouns are objects
- Verbs are methods
- Map out the relationships between different objects that interact with each other. Justify about abstract classes or interfaces. Identify constraints of each class and point out how OOD concepts will help here.
- Sketch a high-level diagram for the interviewer so they can visualize what you explain to them.

## SEQUENCE AND ACTIVITY DIAGRAMS

- Sometimes asked by the interviewer to explain the sequence of events or the system flow of control of a certain activity.

## USE DESIGN PATTERNS

- Try to apply the best design patterns applicable to each problem and fully explain your solution using this.

## CODE

- Code classes and relationships.
- Code a high-level structure of the system.
- Maybe asked to code one of the classes at the implementation level.
- While coding, prioritize writing the abstract classes and interfaces, core objects and internal structure.
- Ensure to implement a code that follows the best practices in terms of maintainability.

## Design Approach

1. Identify possible use cases
   Example: Parking lot:
   parking toll by agent, showing parking slots available
2. Note any constraints. Ignore any attributes that don't align with requirements
3. We should keep the scalability of our system in mind.
   Example: Parking lot:
   one floor -> multiple floors

| | Top-Down | Bottom-Up |
|---|---|---|
| 1. | This approach constructs high level objects and then designs the smaller sub-components | This approach identifies the smallest components and use those components as a base to design bigger components |
| 2. | It's a backward-looking approach | It's a forward-looking approach |
| 3. | It's mainly used in structural programming | It's mainly used in object-oriented design |
| 4. | It allows for a high amount of data redundancy | It allows for a minimum data redundancy |