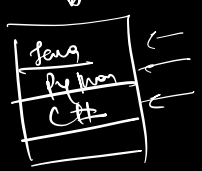
LLD → Introduction to LLD and OOP

TA Req

↓

LLD

↓



Agenda :-

i) what is LLD

ii) why is LLD important

iii) Structure of LLD

iv) Intro to OOP

↳ Basics OOP principles

LLD => low-level design

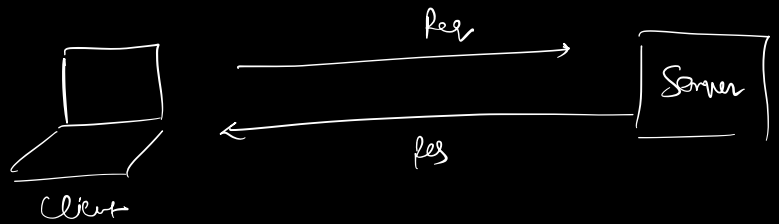
↓

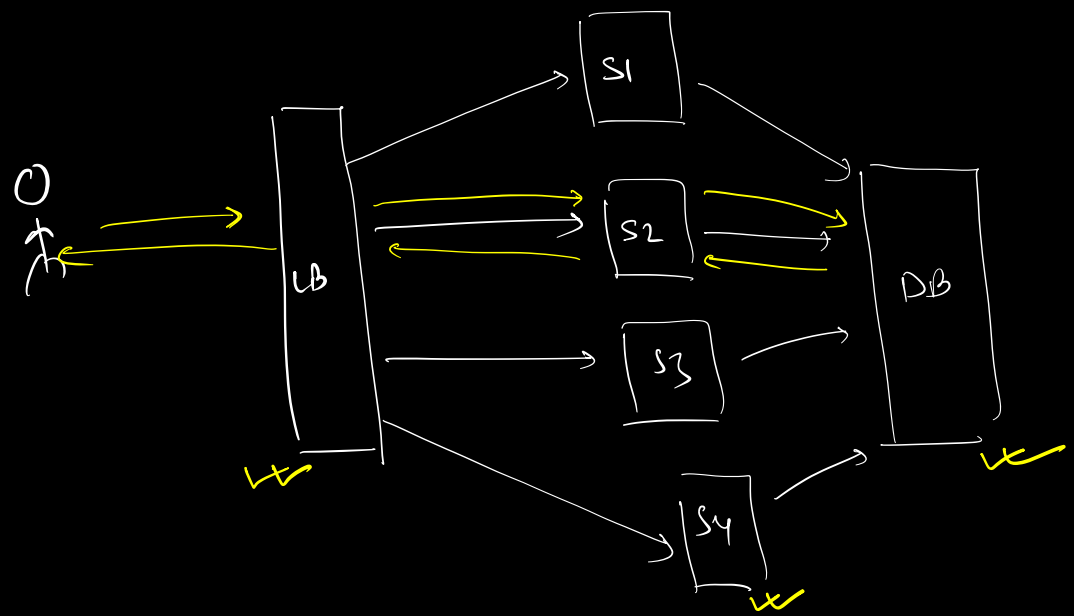
High level design

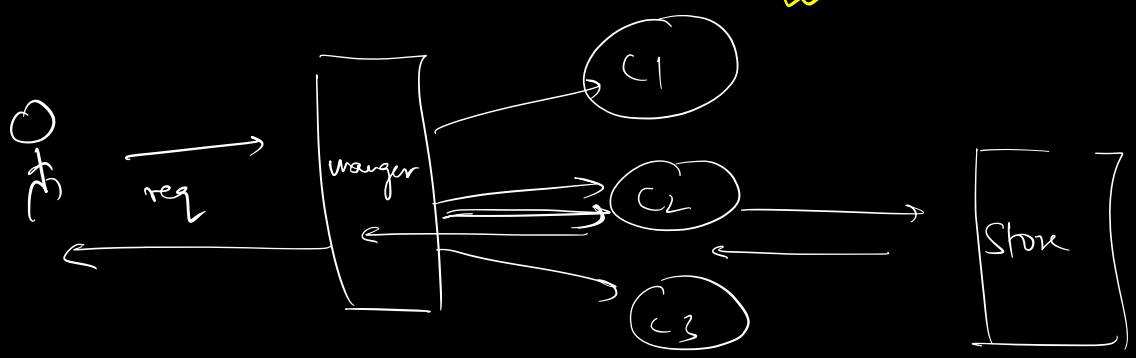
↳ * overview

* bird eye view

* without going into detail







S/w running is different for similar machines, giving them different characteristics.

HLD → design of different infrastructural layers that work together to serve an appn at desired efficiency

LLD => low level design

=> details of S/W system

=> How is a S/W system implemented

- what all packages will be there —
  classes / methods / libraries etc
- How are these things going to talk to each other

HLD → architectural

LLD → S/W impl changes

[Object Oriented Design]

- 86% < 3 hrs coding / day
- 12% of day goes into coding

40 LPA ⇒ 4.8 LPA

⇒ Code Review ⟶ Read code

⇒ Meetings and communications ⟶ gather requirements, gather issues, new feature

⇒ Testing ⟶ reading code

⇒ Debugging ⟶ reading code

⇒ Analyzing and reading code ⟶ gathering requirements and taking new feature

⇒ coding ⟶ coding

⇒ Documentation ⟶ reading code

LLD → makes your job easy

⇒ Code

i) Understandable [best documentation is a well written code]

ii) Easy Requirement gathering

iii) Extensible → adding new features.

iv) Maintainable → performance updations
bugs
library updates → Jenkins bot

Log 4 J

⇒ Why is LLD important for you :-

Top startups

SDE - 1 ⟶ LLD | Machine Coding

Cred, Scaler, Unacademy, Swiggy, GoTele, Flipkart

Big MNCs – Startups

SDE - 2+ ⟶ LLD | Machine round | GOSh

3yrs ⇒ Google ⇒ SDE 3+ { 5 YOE } L5

⇒ LLD module structure

1) OOPs (2 classes)
2) SOLID (1 class)
3) Design Patterns ⟶ (4 classes) (10)
4) UML Diagram ⟶ (1 class)

v) Schema Design ⟶ 2 classes

⇒ Case Studies (How to approach LLD problems)

Entities	Class	Assign
Games	Tic tac toe Snakes & Ladder	Chess Card Game
Real systems	Parking lot Book Myshow Mail Chimp Splitwise	Amazon / Flipkart _____ _____ _____
Eng problems	Distributed Cache	_____

Spring Boot ⟵

Assignment

i) MCQ ⟶ OOPs classes ⇒ Auto

ii) Diagram ⟶ { class diagram, UC diagram, Schema diag } ⟶ 3 peer review + 1 Scaler review

Draw.io embedded in Scaler ⟵

I model SOLID with defined approach ⟵

iii) Write code ⇒ problem statement ⇒ 3 peer review + 1 Scaler review

⇒ Intro to OOP

- Procedural Programming
- Object oriented programming

: Procedural Programming :-

- Procedures ⇒ old age name functions/methods

* We divide our code base into a bunch of procedures.
* every procedure may internally call other procedures
* Execution of program starts from a special procedure (main)

Procedure :- Set of instructions that might take some i/p values and perform some action based on them and may/may not return anything

- i/p values → Parameters
- return anything → return

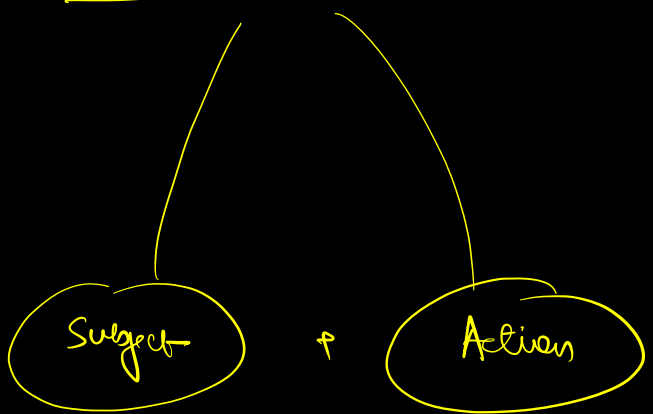
⇒ You are leading a team

⇒ Write down the tasks that needs to be done tomorrow

Notebook

* (A) will report to Mumbai HQ tomorrow
* (B) will solve issue X42
* (C) should finish task X
* (D) will give a demo
* E should give bT

Someone is doing something



⇒ Entities perform action ⇒ Real world

Procedural ⇒ Actions are happening on entities.

⇒ Procedures

```
action1() { }
action2() { }
action3() { }
```

⇒ struct Student {

name;

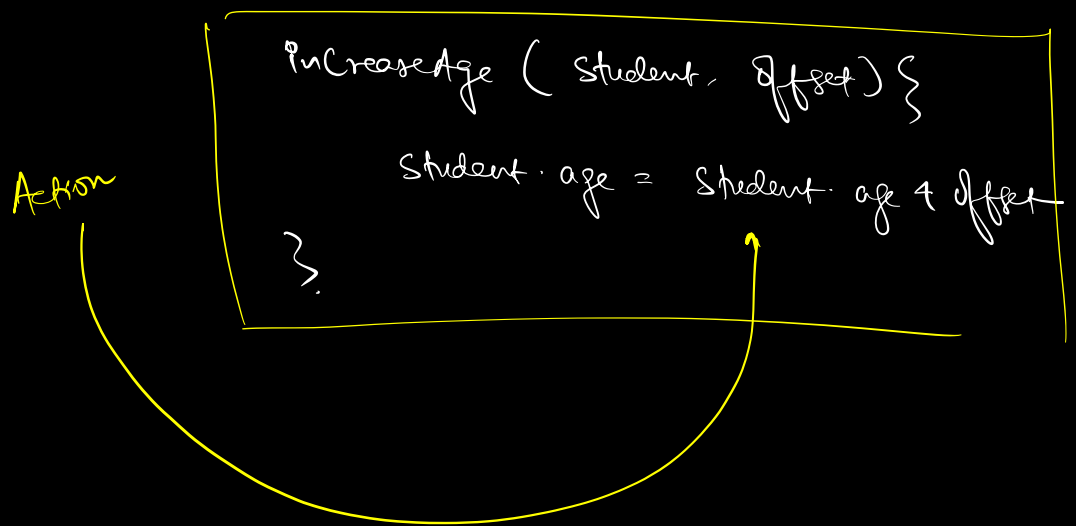
age;

gender;

}

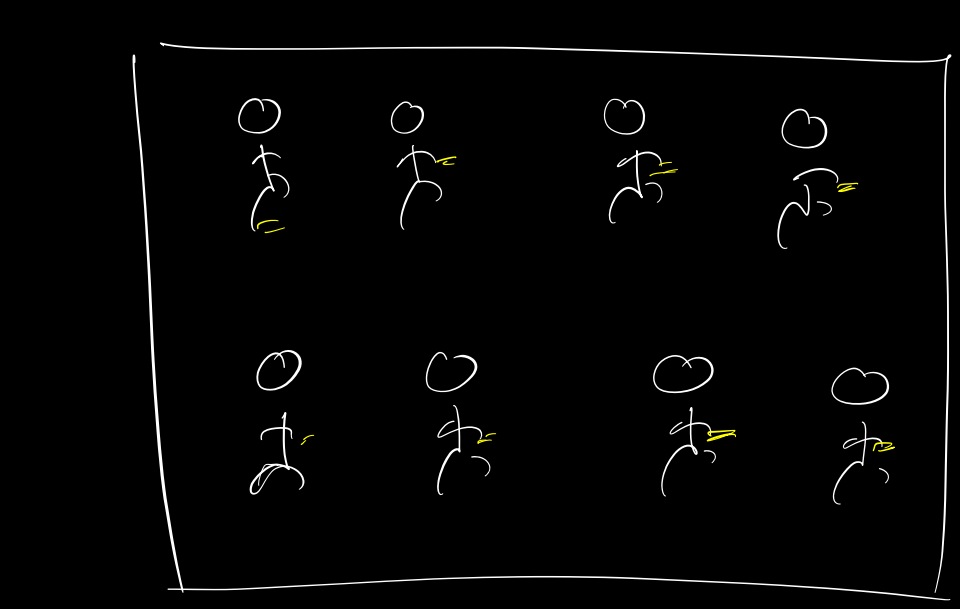
⇒ Structs are containers that contain diff data types together to represent something

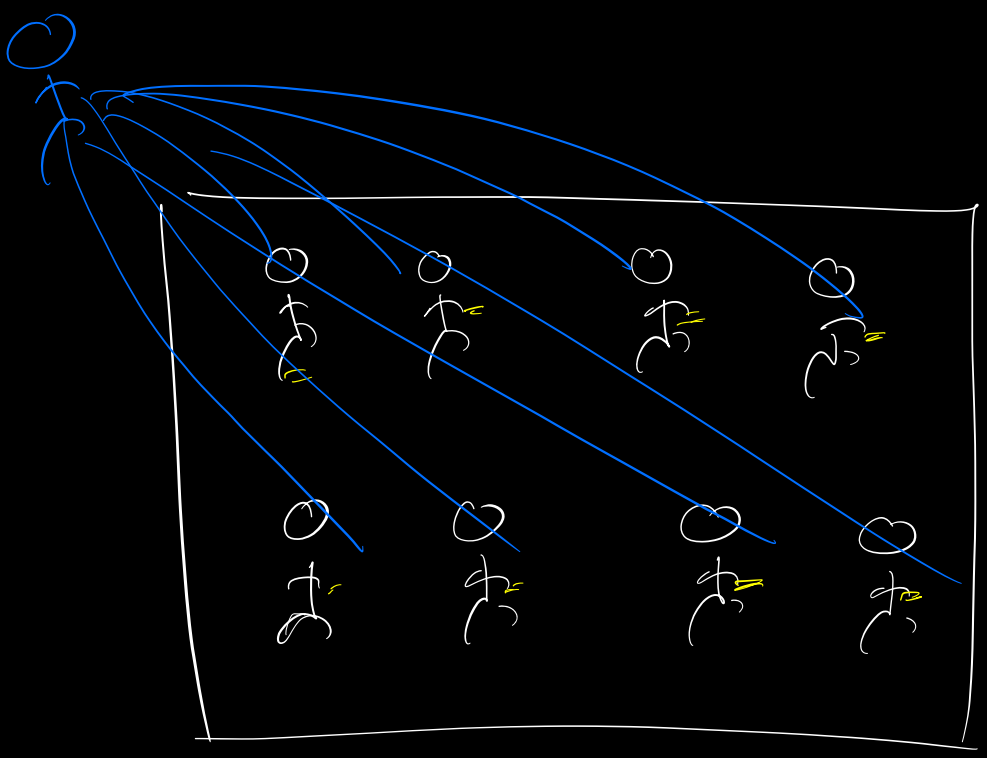
⇒ Like a class but has no behaviour only data points



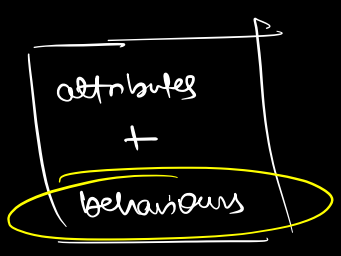
Real world ⇒ entities perform action

Procedural becomes difficult to understand as it does not replicate real world.





OOP → entity ⇒



entities control what behaviour can be performed on them

: CONS of procedural programming:-

i) Difficult to make sense in a complex system
ii) Understanding is diff
iii) diff to develop and debug
iv) lot of interdependencies
   └ Spaghetti code

: OBJECT ORIENTED PROG:-

Principles → Value System | fundamental

(i) Abstraction

Pillars

(i) Encapsulation
(ii) Inheritance
(iii) Polymorphism

=> Abstraction

=> foundation of OOP

=> Principle on which OOP is based

Abstraction => means making something abstract

↓

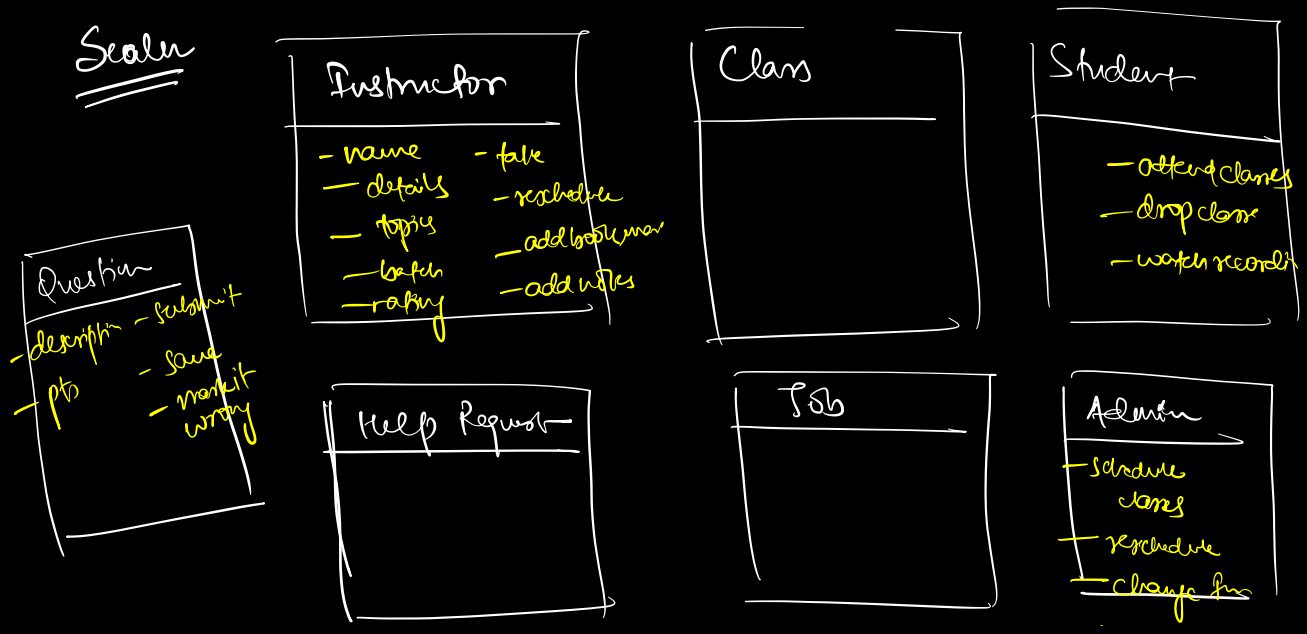
rep. something as an idea

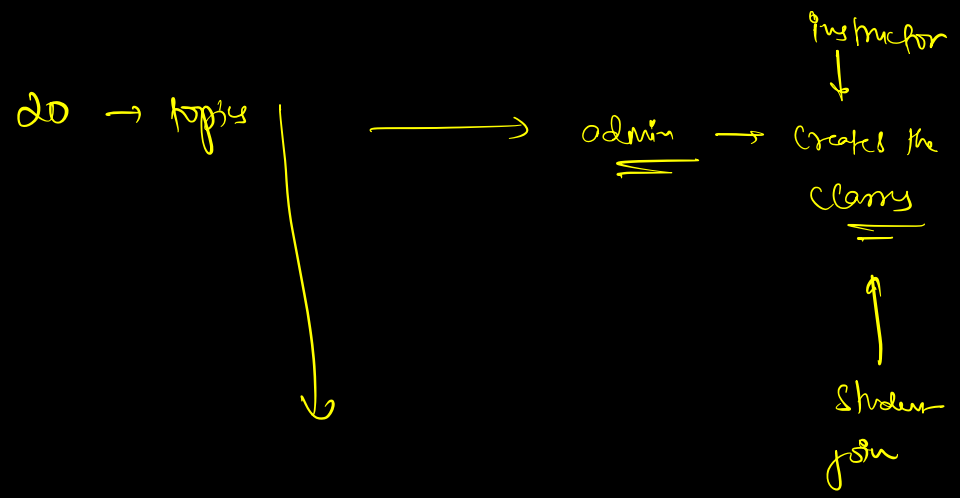
↳ Rep a complex software system in terms of ideas

↓

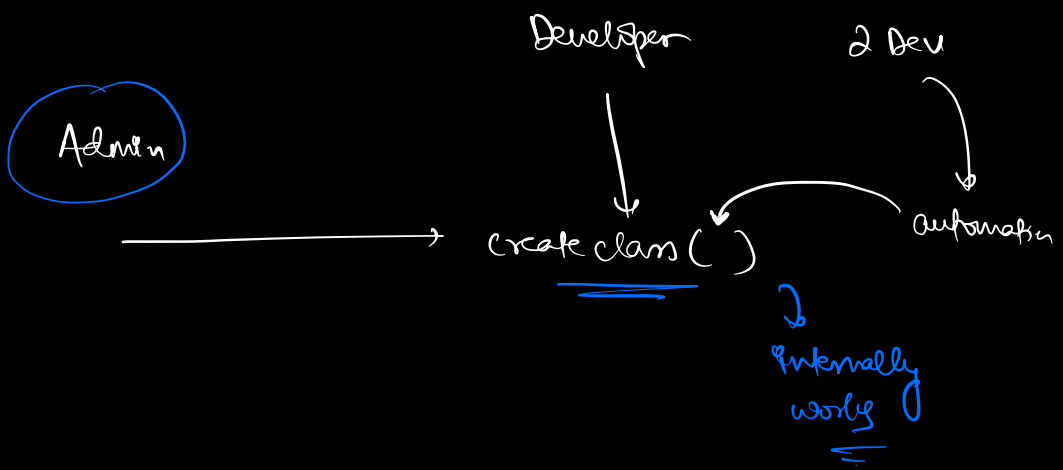
entities that would have some data attached to them, and they'll perform some actions.

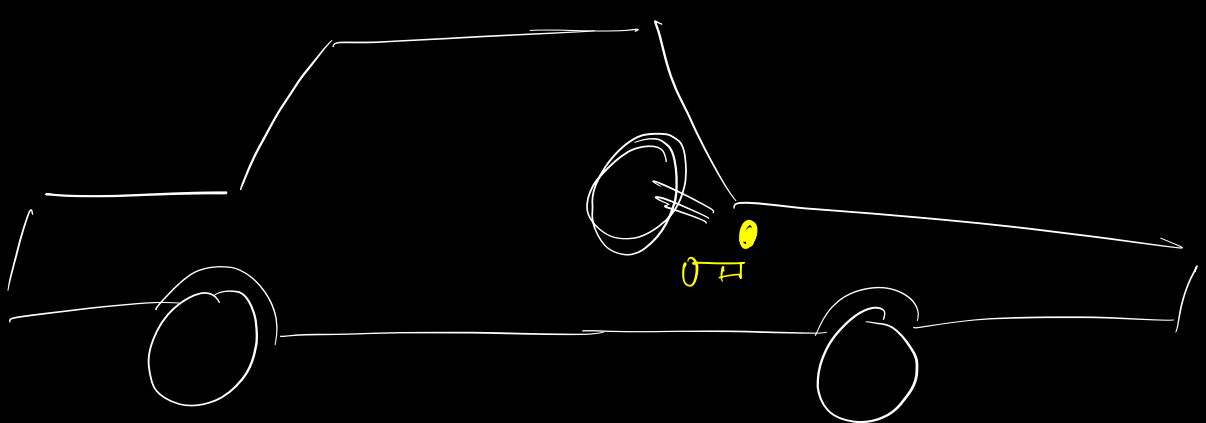
Scaler





=> [No need to be concerned about the internal working of an idea.]





Abstraction → (Target)

1) Rep. a complex system in terms of ideas
   - data
   - behaviour

2) As a client, we don't need to care about how a particular idea works internally, we only care about the exposed behaviour.

Pillars → next class