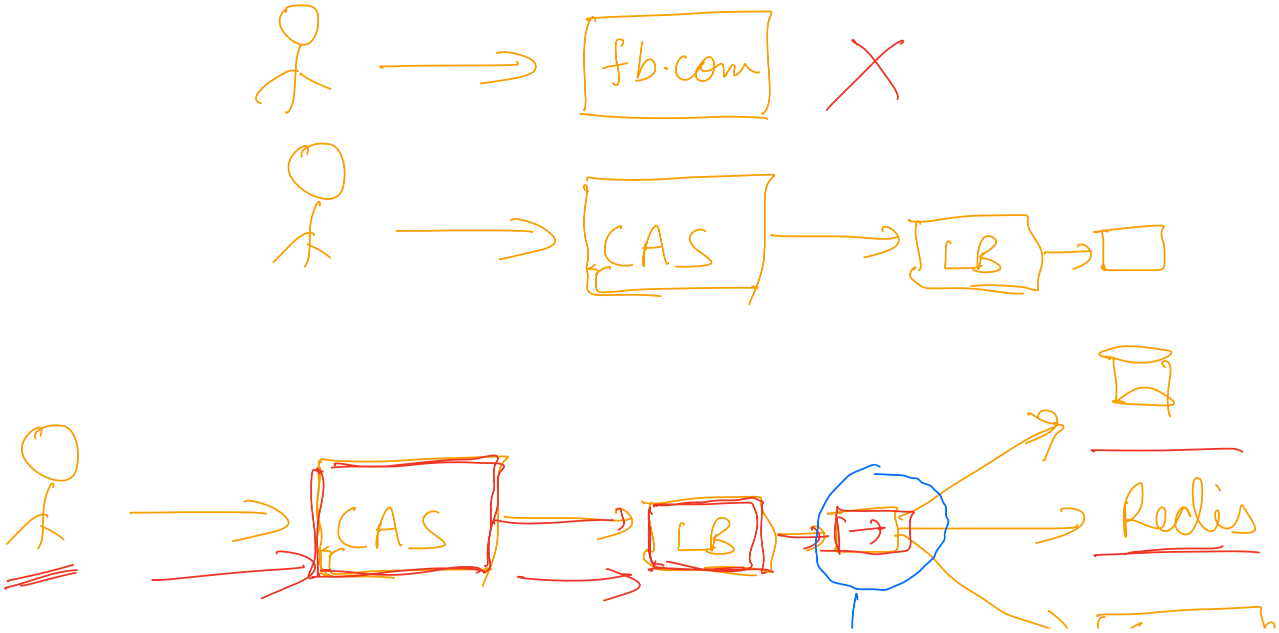
Class starts at 9:05 PM

Agenda → ~~Intro~~ to LLD
~~Why~~ is LLD important?
~~Structure~~ of LLD curriculum
Intro to OOP

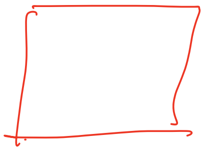
1) LLD → L → Low
L → Level
D → Design

HLD → High level design

DNS , Load balancer



LB →



↓
computer
that is running fb code

→ Internal services

HLD → Nothing but computers running these servers

LLD → details about the actual code that is running on these machines.

flyover

HLD → source, destination, cost
length

LLD → material, angle of flyover

⇒ According too research paper, an avg. soft. engg. spends around 15-20 (%) time in coding.

- → meeting → req. gathering
- → code-reviews → readability
- → logic building → extensible,
- → refactoring → maintainable

code

→

~~estensible~~ → if you want to add new features to the codebase with min. changes

~~maintable~~ → to keep the code running even when no development is required

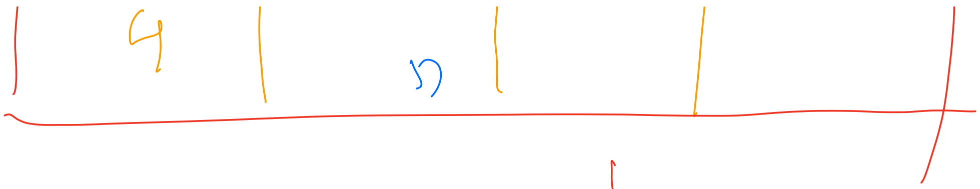
log 4J vulnerability →

Java eg. System.out. println ("Hello world");

② Why is LLD important ?

SDE → 1 → 0-2 yrs
2 → 2-5 yrs
3 → 5-10 yrs
4 → >10 yrs

SDE	Startups phonepe, curefit	MNC Adobe, DE shaw	FAANG
1	✗	✗	✗
2	✓	LLD	LLD rounds
3	"		



LLD interview

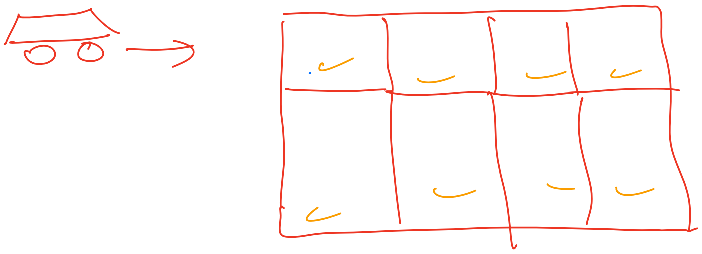
actal working soln is not expected

Machine coding

① expected to come up with actual working code

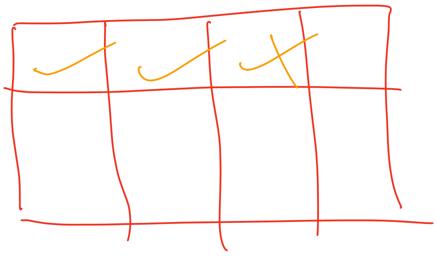
② show the output

③ actually code the soln on an IDE



Input for parking lot

- max. no. of slots ⇒ 8
- no. of cars ⇒ present no. of cars
- park functionality
- unpark functionity



max. slots → 8

park Car

unpark Car

③ structure of LLD curriclum.

- ↳ OOP → 4 classes
- ↳ SOLID
- ↳ UML Diagrams
- ↳ Design patterns
  - ↳ Creational
  - ↳ Structural
  - ↳ Behavioral
- ↳ Machine coding problems

game ↳ Tic tac toe → design & code

management system ↳ Parking lot → design & code

↳ Book my show → design & code ↳ concurrency

entity ↳ Design a Pen

Real world application ↳ Design splitwise ?

④ Intro to OOP — ~~10:17~~ 10:20

OOP → Object Oriented Programming

Programming paradigms

Procedural — C

Object Oriented — Java

Reactive — Java
functional — Scala, Haskell

A single language can have multiple programming paradigms.

Procedural programming →

```
func A (↓) {

}
```

(1) Procedure — block of code which can take an input & produce an output

(2) procedures — can call each other

(3) Usually, there is a main procedure from where the execution of the application starts

```
print (argument) {

}
```

1 natural way noun + verb → active

procedural way verb + noun → passive

2

```
struct Student {
    age;
    name;
    print(student) {
        print(student.name, student.age)
    }
}
```

not possible in procedural programming

3 A → B

easily access all the data of any struct

any procedure can modify any sort of data.

* Any thing in S/W system must be represented an entity with its attributes & its associated behaviour.

Summarize the issues in Procedural programming

① Related funtions can be present in diff. files. → print a student

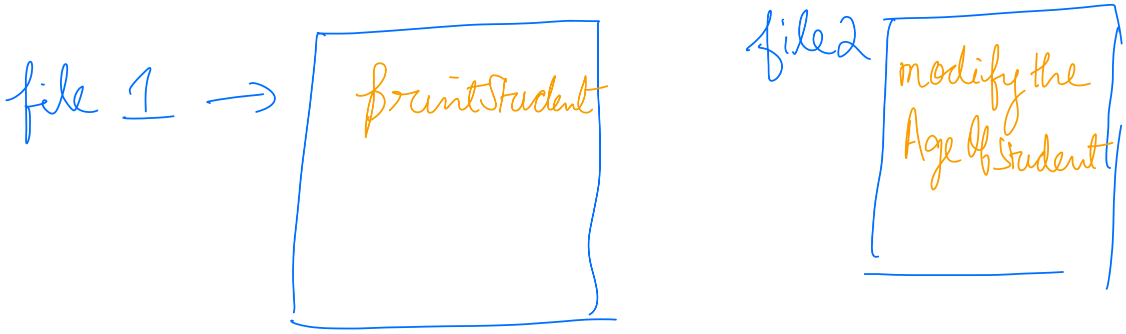
→ modify the name of the student

file 1 → printStudent()

file 2 → modifyTheAgeOfStudent()

② Difficult to debug the code

3 It can lead to spagetti-code, — related things in different files.



```
Student. printDetails (     ) {

}
```

```
verb printStudent(Student) {
```

noun

passive

}

active

Student.printStudent() {

noun verb

SOP("name");

}

Student.printStudent();

active

printStudent() {

verb

SOP("name");

}

→ Representing the idea in form of an entity → attributes → name, age

→ associated behaviour

↳ functions inside it

4 pillars of OOPs

↳ ① Abstraction → principle

② Encapsulation →
③ Polymorphism →
④ Inheritance →
} Pillars of OOP

principle → I'd be a good person

pillar → I wont tell a lie

Lets say we have to represent multiple students → $\underbrace{A1, A2, A3}$

```
class Student {
    int age;
    String name;
    String batch;


    void modifyBatch (String batch){
        this.batch = batch;

}}
```

Terms in OOP

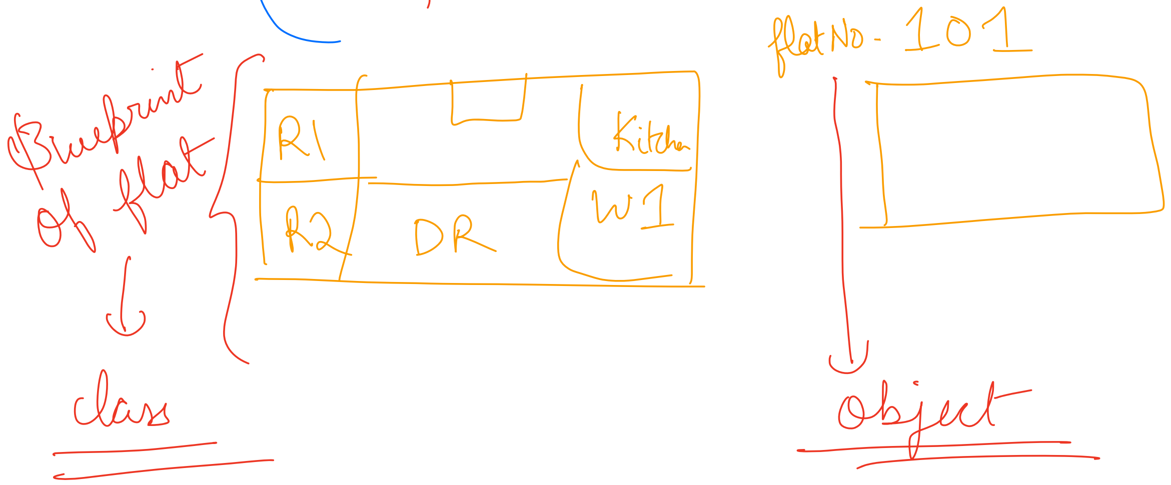
① Class → Blueprint of an idea

```
class Student {
    int age;
    String name;
    String batch;

    void modifyBatch(String batc
        this.batch = batch;
}}
```

arg.

object's batch value



⇒ All the flats are going to be created exactly similar to this structure.

② Object : creating an instance of the class

$$\underbrace{\text{Student}}_{\text{class name}}\ \underbrace{\text{s1}}_{\text{object name}} = \underline{\underline{\text{new}}}\ \underbrace{\text{Student}}_{\text{class name}}(\,);$$

s1. Name = "Ravi";
s1. age = 25;

⇒ class itself doesnt have any meaning
⇒ class doesn't take any memory

⇒ Object → takes its own memory
→ It has its own values for diff. attributes
→ No two objects will occupy the same memory.

Abstraction → ① Breaking down a complex S/W system into a set of entities and associated behaviour

② External users dont need to know the internal details of it.

eg. 1 → for a system like scalar -
entity → students, instructors,
Mentors

② Encapsulation

→ holds diff. medicines together

→ protects it from the outside environment

⇒ holding the data → attributes & its behaviour together and that is achieved by using classes

⇒ Encapsulation is the pillar which is helping us achieve abstraction.