Agenda

i) Intro to LLD

ii) Why is LLD important

iii) Structure of the module

iv) Introduction to OOPs

LLD ⇒ Low Level Design

↓

[how to implement something]

HLD ⇒ High Level Design

Requirement Doc → what to build → how to build → Architecture → impl → LLD

(Architecture: HLD)

DSA, LLD, HLD

↓

interviews

↑

day to day work

S/W ⇒ day to day jobs:-

* Meeting | Scrum → Req gathering | Planning
* PR review | Code review → Reading code
* Bug fix → refactoring code
* Testing → reading | ensuring E2E work | refactor
* Maintaining the work flow [Agile] → planning
* Documentation → reading code becomes easier
* Coffee | TT | Gossip ⇐
* writing code ⇐

⇒ SWE spends only 12% of working day, writing code

⇒ 40 LPA ⇒ 12% 40 ⇒ $\frac{12}{100} \times 40$ ⇒ 4.8L

88% { I) Requirement gathering
II) Readability of the code
III) Modularity of the code

12% [IV) writing the code

A

Code is answers

(a/b people will understand)

B

code is as streamlined as possible

documented

LLD

→ Readable / Understandable

→ Req. gathering

→ Extensible → easily add new features

~~scale~~

→ Maintainable → easy to keep the current system running

[ex ⇒ bug fix, patch updates, library upgrades]

ex ⇒ log4J

4 classes

configurations

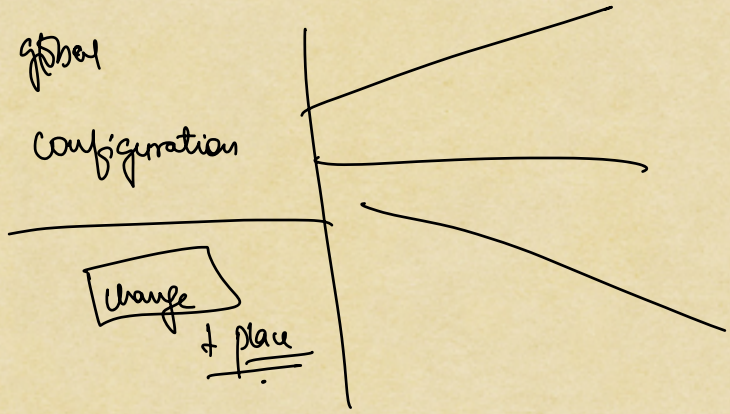
version

configurations

version

configurations

version



⇒ LLD for interviews :-

0-2 → SDE 1] 70% + 30% LLD

2-5 → SDE 2] 30% + 40% + 30% HLD
DSA LLD

5-10 → SDE 3] 20 + 30 + 50

10+ → Architect] 10 + 30 + 60

Theory	Design	Machine Coding
Service based companies, banks, consulting firms, accounting firms.	Oracle, MS, Atlassian, Google, Meta, Adobe, PayPal - -	Flipkart, Cred, Google, Myntra, Scaler, Intuit, Razorpay
ex ⇒ PwC, Deloitte, BoA, Sochen, EY etc -.	→ Single line problem statement.	→ Requirement document is given
→ theoretical & language / syntax based questions	→ req. gathering skills	→ need to write end to end working code with tests
	→ design the system	
	→ no coding	

⇒ no need to write code / design anything

⇒ 60 mins

→ Class names, tables names, class diagram, DB

→ design patterns required

following all LLD principles

→ 100 mins during a call.

⇒ LLD Curriculum :-

INTRO TO OOPS

Programming Paradigms :-

i) Procedural → C

ii) OOP → Java, Python, Js, C#

iii) Functional → Scala, Haskell

iv) Reactive → Java

Chat GPT

H/W read about them

⇒ Procedural Programming.

↓

Procedure :- set of instructions

+ function / methods

⇒ we organise our code into a bunch functions (procedures)

⇒ each procedure may internally call other procedures

⇒ execution of the program starts from a specific procedure ⇒ (main)

Procedure: set of instructions that work on a given set of data and may output some data

```
a(){
    b();
}
b(){
    c();
}
c(){
    printf(——)
}
main(){
    a();
}
```

PRL

→ Sandeep is teaching LLD.

~~LLD is being taught by Sandeep.~~

→ We are learning LLD

~~LLD is being learned by us.~~

IRL Subject performs a verb.

⇒ Problem in procedural programming

```
printStudent( String name, int age, String gender) {
        sout(name)
        sout(age)
        sout(gender)
}
```

```
struct Student {
    String name;
    int age;
    String gender;
}
```

* Structs don't have any methods

↓

print the details

```
printStudentDetails( Student StudentStruc) {
    sout(name)
    sout(age)
    sout(gender)
}
```

struct → performing action

action $\xrightarrow[\text{performed}]{}$ struct

OOPs

Object oriented system

↓

replicate the real world with object

Object ⇒ {attributes + methods}

↓ [properties] ↓ [actions]

```
Class Student {
        String name;
        int age;
        String gender;
        void printDetails(){
                sout(name, age, gender);
        }
}

Student s = new Student("A", 1, "M");
s.printDetails();
```

Procedural ⇒ action being performed on subject

Object ⇒ subject performs action.

↓

easy to replicate real world

⇒ OOPs ⇒

1 principle ⇒ Abstraction

3 pillars ⇒ Encapsulation, Inheritance, Polymorphism

Principle ⇒ fundamental / foundation

Pillars ⇒ support / hold the principle

⇒ We use our 3 pillars in OOPs to achieve Abstraction

⇒ Abstraction :- representing something in terms of our idea