22/08/24

# 4] Low Level Design - Basics.

Naman Bhalla

## LLD Basics :-

↳ Low Level Design

## Agenda

- → Intro
- → When will LLD start.
- → Structure of LLD classes.
- → What is LLD ?
- → Why should LLD learn ?
  - ↳ why interview ?
  - ↳ Job/career ?
- → Intro to OOP
  - → Prodecual vs OOP
  - → How OOP came into picture.
  - → Basic terminology of OOP
    - ↳ class
    - ↳ object
    - ↳ state
    - ↳ Inheritance
  - → pillars of OOP
    - ↳ Abstraction — not a pillar
    - ↳ Polymorphism
    - ↳ Encapsulation
    - ↳ Inheritance

    — learn this is not correct?

- Case Study

Java is used implementing in class.

- ↳ Companies wants Fellow experience in Java
- ↳ Also important for machine coding round.

Every other language → C++, C#, Python, JS, TS, Go lang.

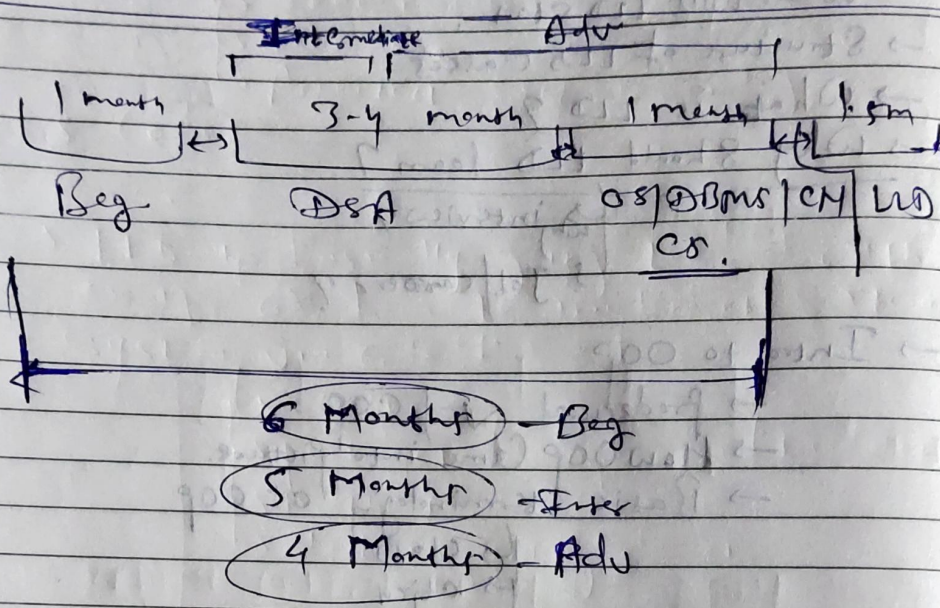
Bridge written material provide.

* Learn Java is Beneficial even know other stuffs.

Instructor :- Naman Bhalla. → scaler.

Ex. SWE @ Google, Curefit, Shiprocket.

Slack - @Naman
Linkedin - 
Phone - +91-9996203771




Structure of LLD curriculum :- 18-20 classes

2-2.5 hrs. 6-7 weeks

→ OOP (2 classes). 3 W
→ Design principles (solid, Code to interface) (1 class)
→ Design patterns (4 classes)
→ 11 Design patterns

5+2+4

→ UML Diagrams (1 class)
→ Schema Design (1 class)
→ Project Structure + Design (1 class)
→ Concurrency (Multithreading, Threads, IPC, Semaphores, Synchronized blocks)
→ Unit Testing, BDD

① Java
② Python.
→ very much used in Company.
Java 8+ / 11+

→ How to approach LLD problem
→ Case Studies

- Pen & entity
- Chess.
- TicTacToe
- Snakes & ladders. } Games
- BookMyShow
- Parking lot
- Mailchaing
- splitwise } Real System
- Distributed cache
- Message Queue } Generic Problems

20 classes

→ Every programming language has it's used case.

Singleton
- Multithreading
- Synchronization

Head First OOP
Refactoring. Fuse
Effective Java
Hacker News (News.ycombinator.com)

Books

* What is LLD? :-

LLD → Low Level Design | object oriented Design.

Design → Planning
- Share the plan with others
- Plan what to code
  - Split the work in diff. people.

Avg S.W. → Spends 12% of time day writing Code

1. Gathering Requirements
2. Clarify the Requirements
3. Code Diagram (UML Diagram)
   - Class Diagrams
   - Activity Diagrams
   - Usecase Diagrams
4. Design Doc
   - Team members
   - Managers
   - Architect
5. Code.

⟹ LLD is basically going to deal with, Design of how to implement Software System via Code.

- Class
- Packages (Components / Modules)
- methods
- Interact among classes.

→ persistance into DB
   - Schema of DB

LLD vs HLD ← Microservices

LLD → Design of Code

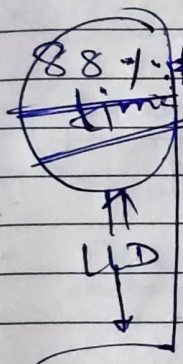
HLD → diff. In Programstructure / layer

* Why learn LLD?

12% time of day writing code.

⇒ Developer Spends time :-

→ Meeting ⇒ Gathered Req | Understand Req | Doubts
→ Analysis ⇒ Monitoring → Refactoring
→ Debugging ⇒ Read code
→ Emails & Scrums
→ Code Reviews ⇒ Reading code
→ Maintainence ⇒ Read previous code + Adapt
→ ~~Coffee~~
→ ~~Chai Sutta~~
→ ~~Scoter Assignments~~
→ Testing ⇒ Read code + Evaluate it against Cases.
→ Designing
→ Chaining | Refactoring code ⇐
→ Documentations.
→ Stack over Flow




88% Reading Code or Senior activities.
→ Doc
→ Code of others
↳ ReE
↳ Code Review
↳ Maintainence
↳ Debugging

LLD

⇒ Understanding about what code

⇒ LLD allows you to write code in such a way that makes what you do 88% of your time easier.

---

* How is LLD asked in Interviews ?

| | Startups | MNCs (FAANG) |
|---|---|---|
| SDE 1<br>0-2 YoE | 1 LLD / Machine Coding Round | — |
| SDE 2+<br>2-5 YoE | ✓ | ✓ |
| SDE 3<br>5-10 YoE | 2 DSA or 1 DSA<br>1 HLD<br>1 LLD | |

- There are 2 types of LLD interviews

(40-50 min) — (2-2.5 hrs) → set of requirements

LLD — Machine coding

Gather ✓
Clarify ✓
UML Design ✓
Schema Design ✓
⟹ Code (Working code)

Career

Engineering ladder

CTO | HOE

| Level | Skills |
|---|---|
| SDE | — |
| SDE 2 | — (API Design / Concurrency) |
| SDE 3 | — only when they demonstrate all skills of that position greater than 6-12 months |

# * Intro to OOP

## Paradigms of Prog. languages

Diff types
- → Functional
- → Procedural
- → object oriented

} most used.

- → Reactive
- → Aspect oriented

## Procedural :-

- method / function are called procedures in early days.
- Procedure ⟹ Group of instructions to take a particular actions.
- PPL are languages that run a program by running one procedure at a time and starting from a main procedure.
- And in this, program is nothing but, a different procedures running in a particular order to perform the desired actions.

⇓

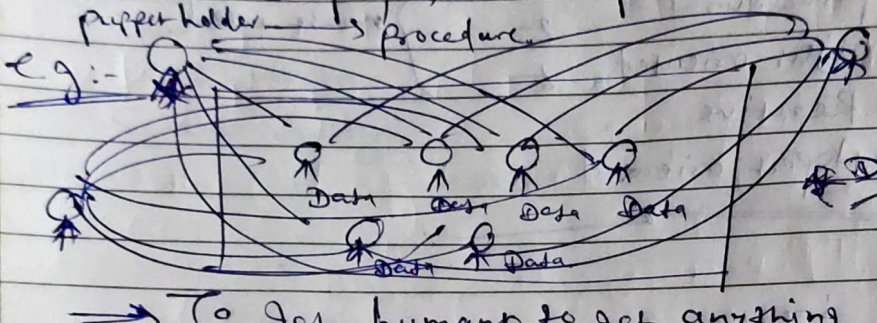
PROCEDURES (INPUT)
CODE THAT TAKE ACTIONS
OUTPUT

C
C++

⇒ The languages opposite to procedural programming languages ~~are~~ which are not support to any Func

## OOP | Non-OOP

→ Depending upon what & how data is supported they can say the language is OOP | Non OOP.




⇒ To get humans to get anything, PH has to take actions.

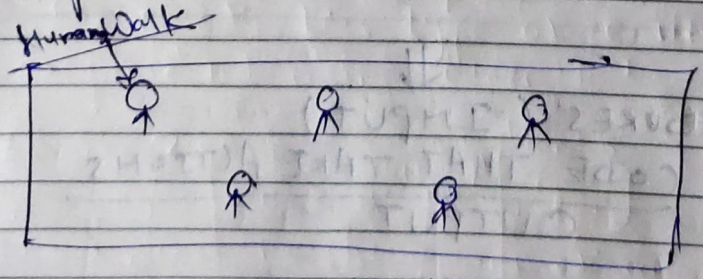
⇒ Structs Human{
name
age
~~leg~~ no. of legs
height
weight
}

Non-OOP

Walk Human( ){
}

eat Human( ){
}

→ In OOP language ⇒ Data items are fully independent entities.



# Procedural P != Functional P

## No-OOP

No actions

```
Struct Human {
    int age
    String Name
    int dob
}

humanEat (Human) {
}

humanDance (Human) {
}

humanWalk (Human) {
}

Human Shantanu = new Human()
walkHuman (Shantanu);
```

## OOP

eat | walk | dance

Full Fledged entity

behaviour

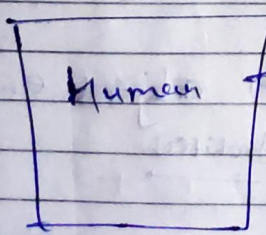
```
Class Human {
    int age;
    String Name;
    int dob;

    Eat () {
    }
    dance () {
    }
    walk () {
    }
}

Human Shantanu = new Human()
Shantanu.walk()
```

```
WalkHuman (List<Human> human)
{
    :
}
```

⟹ In OOP language: Data are Full Fledged entity.



OOP → Java, Python, JS (functi) → Go → No OOP (Fema...)

In Functional Prog. lang. Functions are also Full Fledged entity.
- we can create a variable functions
- we passed functions to functions.

* Key terms related to OOP:-
  - Class
  - Object
  - State
  - Member
  - Attributes

⇒ In OOP lang. each entity can perform action & also have their data.
So, every entity has their data & behaviour

Type of entity → Class

Zoo → class
- Animal
- Cage
- Zookeeper
- Tree
- Visitors
- Ticket

Each of this type of entity will be represented using Class

⇒ Class Zoo { class Animal [Walk() Sleep() Eat() Age name weight] class Cage }

Class ⇒ class is blueprint of an entity

Architect → Software Architect

Class, object, Fields, Attr, Methods, Members, State

↓ Covered.

real existance.

**Object** :- Object is a real instance of a class.

Animal Tiger = new Animal();

Tiger.age = 5;

age
name
everything

⇒ taking space in memory

Animal Dog = new Animal();

Dog.age = 2;

different Storage of object.

Loosely Coupled
High Cohesive classes
Use interface

* Util classes don't have any states.

Case study Adapter design

Paytm
14000 USD
Uber

Phone pe
API
Yes Bank
< Idea
FEDI
Transaction
RBI banned Yes Bank from online

⇒ class Stationary {

Stationary pencil = new Stationary();

⇒ **Attributes / Fields** :- Data values of a class.

⇒ **Methods** :- Behaviour of the class.

⇒ **State** :- State is the value of all the attributes of the object at a particular time.

Animal Tiger = new Animal();

Tiger.age = 12;

Tiger.name = AB;

Tiger.weight = 123;

age = 12;
name = AB;
weight = 123;
height = 0;

Tiger.age = 23

age = 23;

Members are all the attributes + methods of a class.

Mailchaining

Code design pattern

Refactoring guru

Stateless object are object which does not have any state.