Intro to LLD
and Intro to OOP

OOD

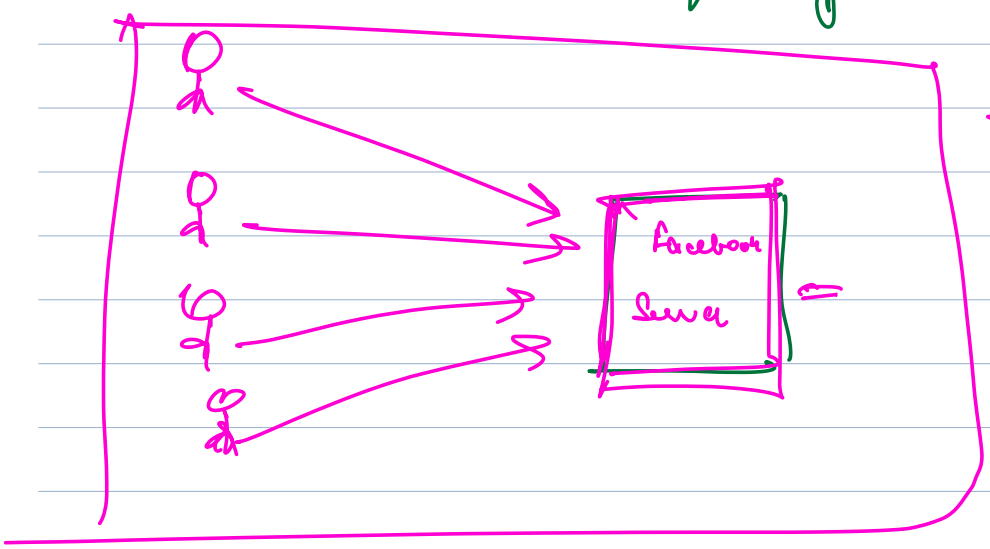
1. Intro to LLD (Object Oriented Design)
2. Why LLD is important
   - Interviews
   - Jobs.
3. ~~Structure of LLD module.~~
4. Intro to OOP

INTRO TO LLD

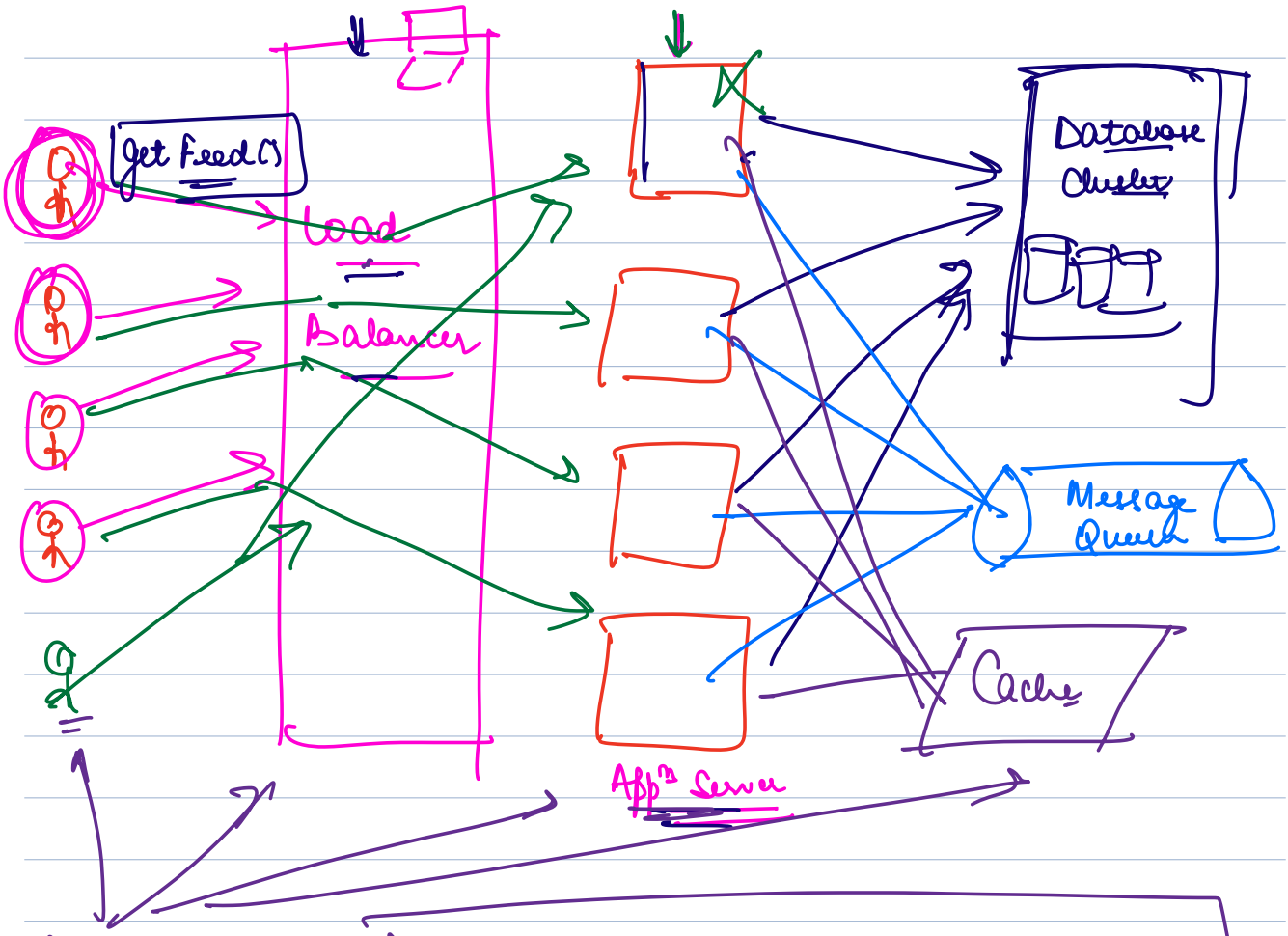
L → Low
L → level
D → Design.

High Level Design.
- ⇒ Overview
- ⇒ Not going into details
- Bird's eye view.

of the system

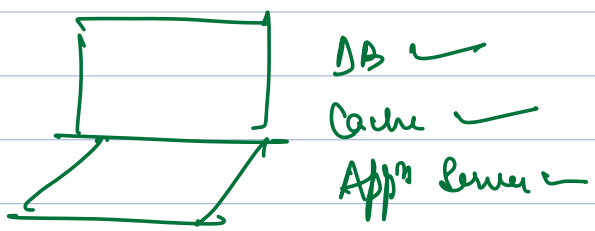


- Bottleneck
- Single Point of Failure



Infrastructure Layer

Design ⇒ Diff infra layers that work together to serve an app^n at scale



⇒ at the end what a comp works as is decided by the software that runs on the comp

LLD

↳ Details of code

↳ Implementation of S/w system

→ What are good ways to write code.

↳ Helps you write better quality code.

Why LLD is important

1. JOBS

8 hr

1 day

≃ (12%) of their day to day time actually writing code.

30 LPA

88 / 100 ✗ 33 LPA

Looks Good To Me

→ Giving KT → understand code
→ Meetings → Planning, Discuss Req
→ Designing ←
→ Code Review → understand code
→ Debugging ← understand code
→ Testing ←
→ Arguing with PM → Requirements

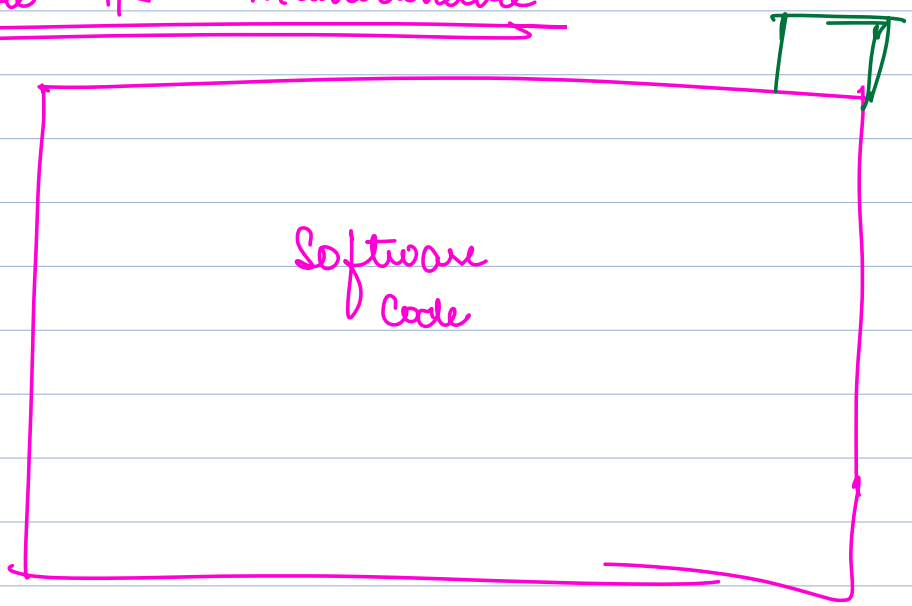
→ Coffee
→ Pair Prog
→ Scale Assign
→ To too small.

→ LLD is going to make what a s/w eng does over 85% of their time easy

LLD
1. Readable / Understandable
2. Req Gathering
3. Extensible
4. Maintainable.

Quality of output of work a SWE does.

Extensible V/S Maintainable



Extensible ⇒ easy to add new features. (NEW REQ)

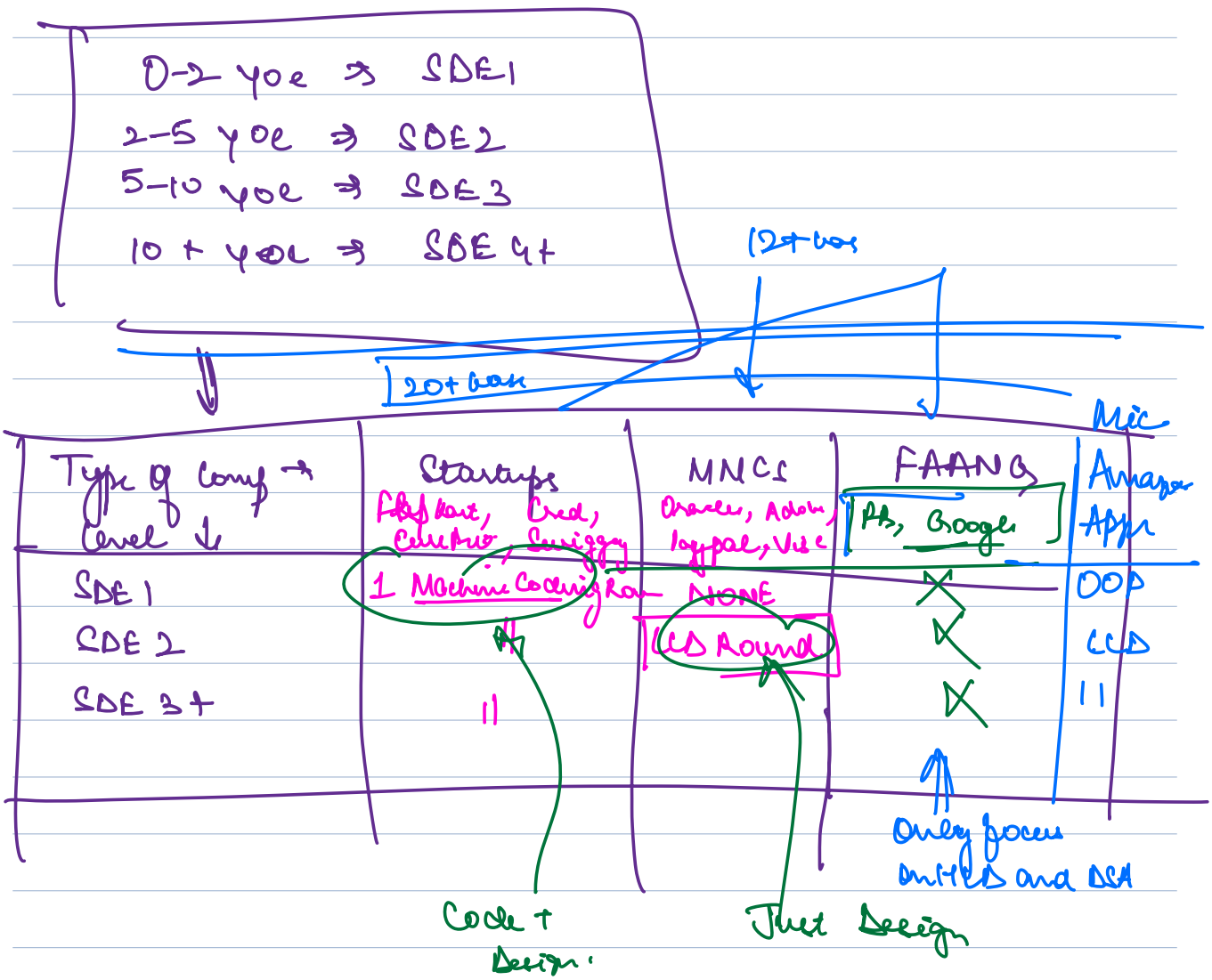
Maintainable → easy to keep running (NO NEW REQ)
→ Bugs
→ Updates to library

MacOS 12
MacOS 13 → Platform Upgrades.

2. INTERVIEWS

→ every startup is going to have atleast one LLD round.

0-2 yoe ⇒ SDE1
2-5 yoe ⇒ SDE2
5-10 yoe ⇒ SDE3
10+ yoe ⇒ SDE 4+



Structure of LLD Module ⇒ 2 Months

Theory

1. OOP ⇒ 4 classes.
2. SOLID Design Principles ⇒ 2 classes.
3. Design Patterns ⇒ 5 classes
   - ≈ 10 design pattern
   - → Singleton
   - → Factory
   - → Builder
   - → Observer
   - → Prototype
   - → Adapter
   - → Strategy
   - → Decorator
   - → Facade
   - → Flyweight
4. UML Diag ⇒ (1 class)
5. Interview Practice

≈ 2 classes design problem

a) How to approach design problems
b) Design TTT
c) Design Parking lot
d) Book My Show
e) Splitwise
f) Cache

- ⇒ API Design
- ⇒ Schema Desi.
- ⇒ MVC
- ⇒ ORM
- ⇒ Spring Boot

In class ⇒ Java

concepts ⇒ Python / Django

⇒ JS | TS / Express JS

INTRO TO OOP

Programming Paradigm

way to do something

↳ way to program

↳ way to structure code / write code.

- Procedural Programming → C, Pascal, JS
- Object Oriented Programming → Python, Java,
- Functional Programming → Java, JS
- Reactive Programming → Java, RxJava

Procedural Programming

↳ Old school name of function

↳ set of instructions to get something done

↳ we structure our code into a bunch of procedures.

→ each procedure may internally call other procedures

→ execution of program starts from a special procedure → main ()

```
acs{
    cc;
}
```

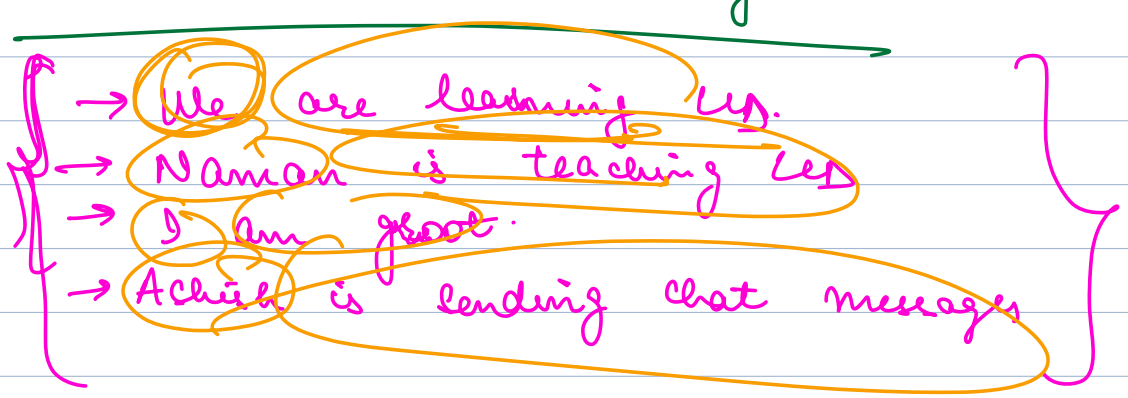
```
bcs{
    ac;
}
```

```
ccs{

}
```

```
main(){
    cc();
    ac;
}
```

Problems with Procedural Prog



→ Subject + Verb

Something . Someone.

print Student (String name, int age, String gender) {
sout (name)
sout (age)
sout (gender)
name = v . .
}

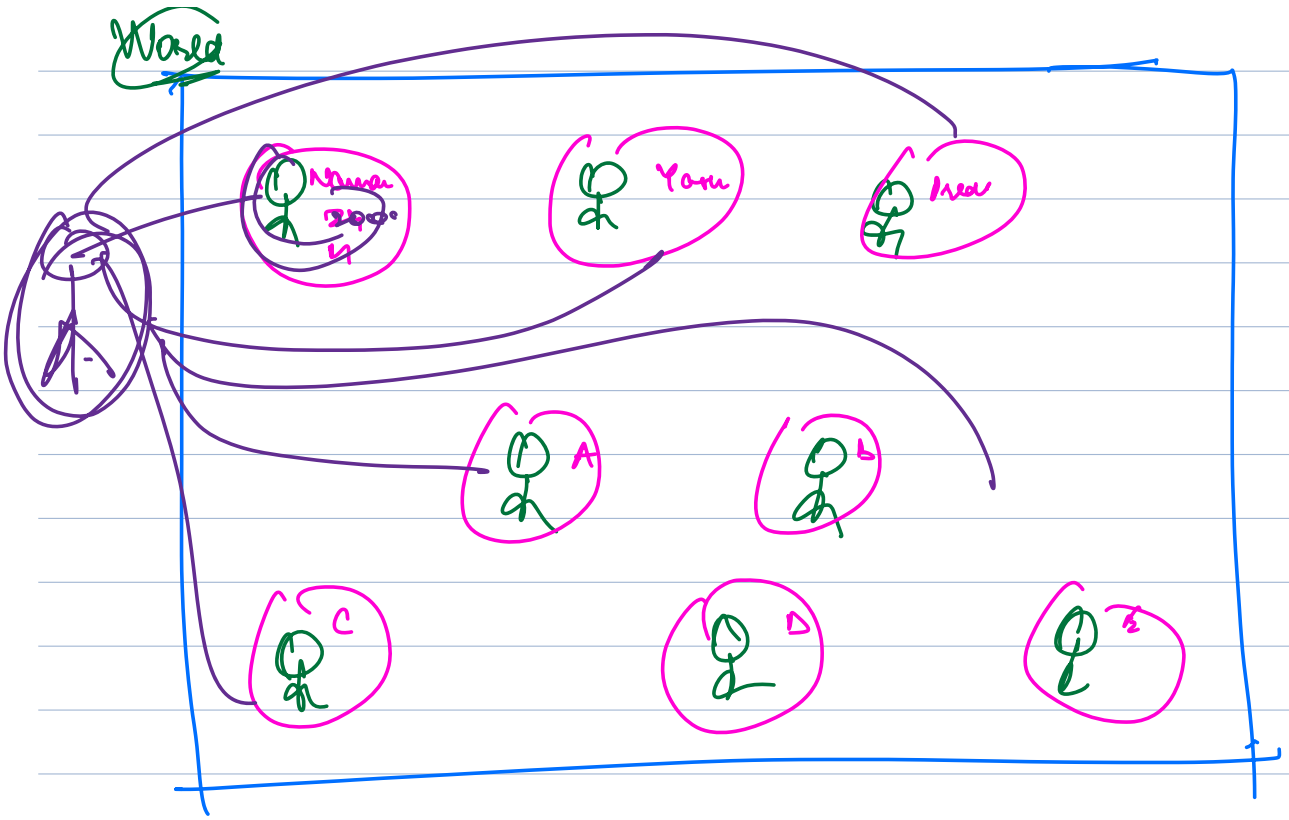
no = in Struct

Struct Student {
String name
int age
String gender
}

print Student (Student s) {
sout (s. name)
sout (s. age)
sout (s. gender)
}

→ has access to every attr of Student
→ it can modify any attr of Student

IRL ⇒ Someone is doing Something
Procedural ⇒ Something is happening on Someone.



Unix ⇒ 10000 ≤ of procedure

Student {

XYZ.C

dance Student (Shoe

→

}

print Student (Student)

⇒ Action is happening on entity

Student . print ()

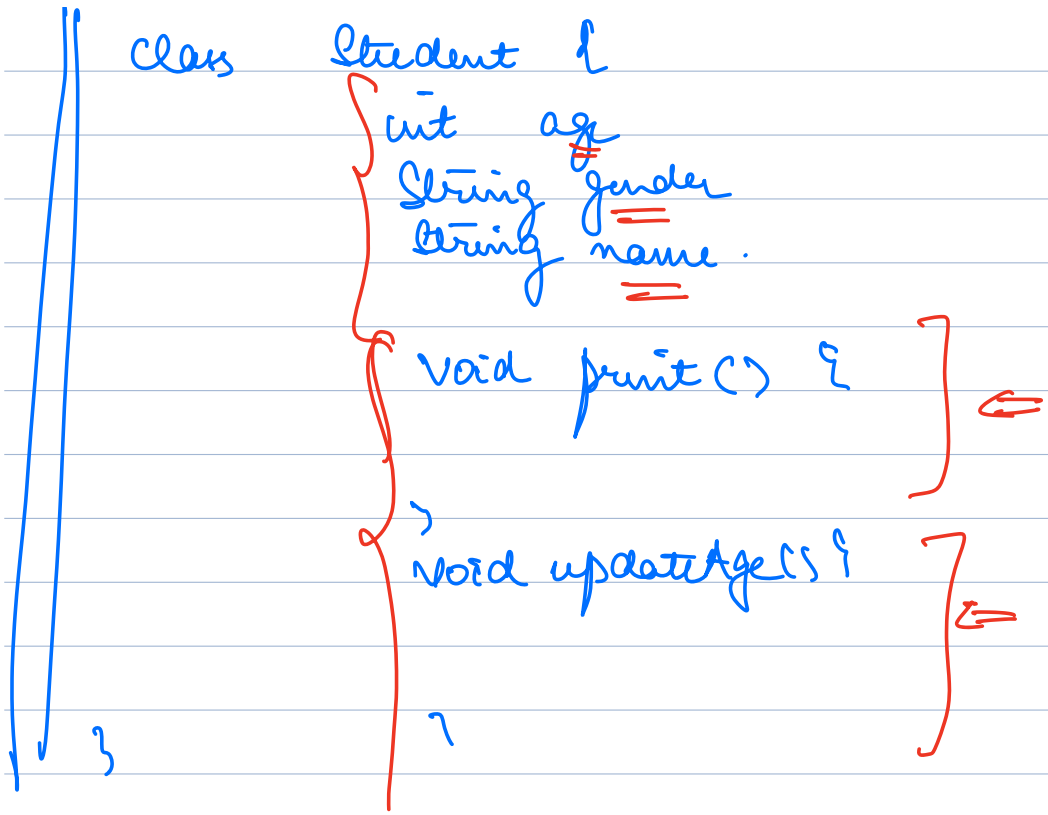
⇒ entity is performing a action

⇒ entities have no control on what happens to them.

OOP ⇒ Way to program where software system is structured around entities.

⇒ entities are the core construct
↳ actions have associated
↳ control what they want to do

⇒ In procedural prog it is more difficult to understand what and why is happening in a S/W system.
⇒ In OOP easy to understand.



OOP ⇒ becomes slow
⇒ Memory Overhead

put (Map , obj)

Map. put (obj.)

C

1 work ⇒ 2 days

Java

⇒ 1 day

=

OOP

→ entities are core construct in OOP

→ every entity has attr and behaviour that they perform

Functional Prog ⇒ Functions can be considered as entities

↓

doSomething(f() ⇒ ( ))

Head First Design Patterns