OOP - I

Agenda

- LLD v HLD
- why LLD
- → LLD module

- what is good software?
- Programming paradigms

- procedural
  - OOP

- OOP
  - Abstraction
  - Encapsulation

Low Level Design

↓

High

High level

- Overview
- bird's eye view
- not too much detail

CODE

Infra. layers

LLD - Code

- structure our code
- how files/classes talk to each other.
- how my code behaves

LLD → low level implementation
→ organisation of code
→ good software

Good software!

- Maintainable
- Scalable
- Extensible

Maintaible

- faster debugging
- easy to understand
- easy to change

- Modularity
- Reusability
- Testability
- Modifiability

SonarQube

Scalable

① Performance — handle the use
- 200 ms
- concurrency

~~② Business~~

1s → 2ms → 5ms →

200 ms

---

③ Extensibility

1000 lox 10000

Aliyun

LLD interviews

O-2 | SDE-1+ | Startups

CRED

Scaler

Locus

LLD

SDE-1

MNC - MoLLD

SDE-2

MAANG - LS - SDE3

3+

Design

LLD module

18 classes

1. OOP - 2
2. SOLID - 1 of 5
3. Design patterns - 4
4. UML - 1
5. Schema design - 2
6. Case studies

LLD

1. Design & implement
2. Design
3. Machine coding

LLD

HLD

Project Building

① entity	Class	Assign.	
② Games	pen	vending machine	
③ RL apps.	TT	chess	
	SQL		
	Parking Lot		11
	BMS		
	Mail chimp		
④ Eng.	Splitwise		
	distributed cache		

TA request

Programming paradigms

Java vs SQL

→ Set a=1

→ a = a+1

→ print a

SELECT *
FROM students;

get all users

series of steps → imperative

define how to → declarative

Set final value

Imp → C, C++, C#, Same

Declarative → SQL, LISP, COBOL

→ React

→ HTML | CSS

Imperative

① Procedural

② OOP

Alice - Bob

① Alice requests 50 tobe sent to Bob

② Bank deducts 50 from Alice

③ Bank adds 50 to bob

Object verb

data - behaviour

Procedural

- state & behaviours are separate

main

6:00 6:05

10:30 10:35

PP — state vs behaviour

Pro — easy

— modular

— decoupled state + behaviour

Con — state sharing - maintainable

— extensible

— security - state

State and behavior

OOP — Objects — state
behavior

Objects — attributes
methods

OOP — objects — Bank
Customer

↓

entities

Characteristics

OOP → Abstraction

implementation hiding

Input

night →

left →

Interface

steering

steering

Output

Car should go right

Car should

accel → gas pedal

go lf

car should go forward

1. Security
2. easy to use
3. separation of concerns

Abstraction

↳ Polymorphism
Inheritance
Encapsulation

Encapsulation

Capsule

① Holds everything together

② Protects my medicine from the environment

Encapsulation in OOP

① Single unit for data and behaviour

② Hides private or sensitive data

---

private - access modifier

Encapsulation - data/info hiding

Abstraction - imp. hiding

Class

blue print

Student - name
- age
- phone

Style

class → instance / objects