19 sept

HLD vs LLD

High Level Design - architecture level, infrastructure, choosing technologies

Low Level Design - Easily maintainable & extendable code

Classes - blueprint / template

Object - instances of it

Example :

Order - order id, status, add_item(...)

↳ properties + methods

↳ The class itself does not have data

OOPS - Programming approach

Encapsulation : • Put them together - putting properties & members in one capsule.

• Also access control - Private, protected, public

$$\begin{tikzcd}
\text{Private} \arrow[d] & \text{protected} \arrow[d] & \text{public} \arrow[d] \\
\text{Self} & \text{child class} & \text{All classes}
\end{tikzcd}$$

Inheritance :

• Reusable

• Restaurant ⟶ Fast food chain

Parent child

Abstraction : Ability to use features without looking at implementation

Polymorphism - Many forms
- different implementations of the same thing

$$\begin{tikzcd} & \text{Polymorphism} \arrow[dl] \arrow[dr] & \\ \text{Compile time} \arrow[d, Rightarrow] & & \text{Runtime} \arrow[d, Rightarrow] \\ \text{function overloading, Operator overloading} \arrow[d] & & \text{Virtual- override} \\ \text{Number of parameters + Type of parameter} & & \end{tikzcd}$$

function overloading
Operator overloading

Number of parameters
+
Type of parameter

eg: process payment () → Wallet / Card / gpay...
Type

Abstract class & ~~[illegible]~~ Interface

Place Order
MakeDelivery → BikeDelivery
→ CarDelivery

Some ^common^ properties will be there but how to do it is taken care separately
Eg: chess ⇒ move function
Implemented differently on each piece

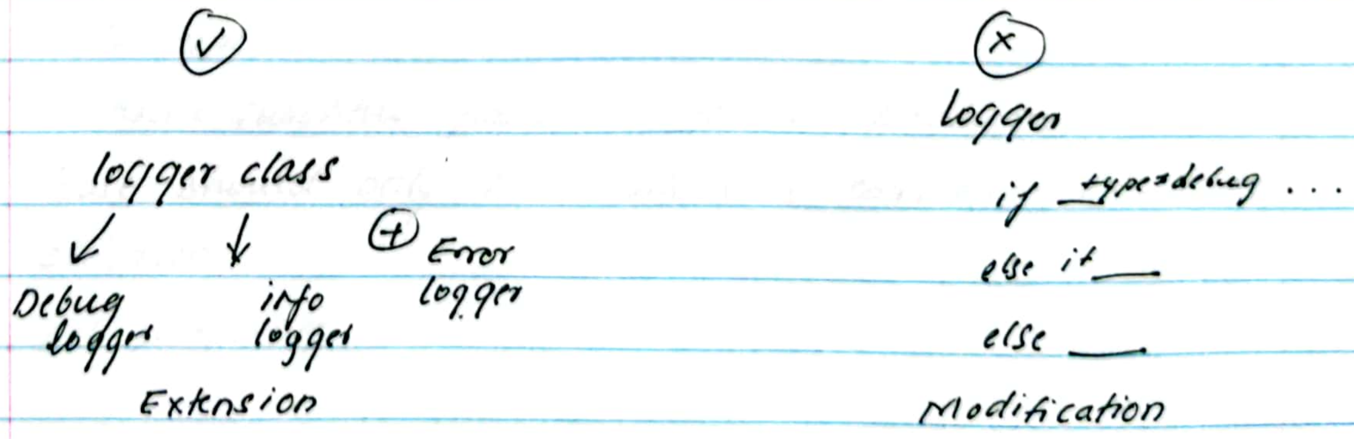
Abstract class → can be extended / inherited
→ It can also have defined function

Interface → can be implemented
→ Forces all methods to be implemented

<u>SOLID principles</u> – Guideline for designing

① Single Responsibility Principle
- One method does one thing
- Subjective

② Open / close principle
- Open for extension, closed for modification



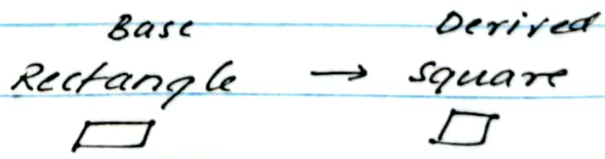
- If code is modified, need to test already existing code.

3. Liskov Substitution Principle - LSP

- Subclasses should be able to be used in place of parent class without causing issues.

  - Animal → Dog ; Dog object can be used as animal
  - Base class

Wherever we are using base class , we should be able to use desired class



```
change-width(x)
{ width = x
}
```

- Subtype
  should substitute parent without problems.
- Base should only have which is common for all its children.
  All properties.

$$\begin{tikzcd}
 & \text{User} \arrow[dl] \arrow[dr] & & \\
\text{Customer} & & \text{Partner - KYC} \arrow[dl] \arrow[dr] & \\
 & \text{Delivery} & & \text{owner}
\end{tikzcd}$$

④ Interface Segregation Principles

- No client should be forced to depend on interfaces they do not use.
- Encourages smaller - more focused interfaces.

⑤ Dependency Inversion Principles.

Decoupling - loose coupling

Notification manager ⇒ High level

```
{ if sms    ⇒ low level modules
  if wa
  if ....
}
```

$$\begin{tikzcd}
\text{Notification Manager} \arrow[r] & \text{INotification Sender} \arrow[d] \arrow[dr] \arrow[drr] \arrow[drrr] & & & \\
& \text{SMS} & \text{WA} & \text{Email} & \dots
\end{tikzcd}$$

High level
Abstraction layer between
low level