Lec-6 : Solid Principle Contd

Page No.
Date 19 05 25

Continuing LSP
-More detail description as LSP is the one who gets too much break or not able to follow

Guidelines / Principle
1. Signature Rule
2. Property Rule
3. Method Rule

Before starting let's understand what these words means-
a) Broad : Ancestor / Parent class ex: Animal
b) Narrow : Child class ex: Dog

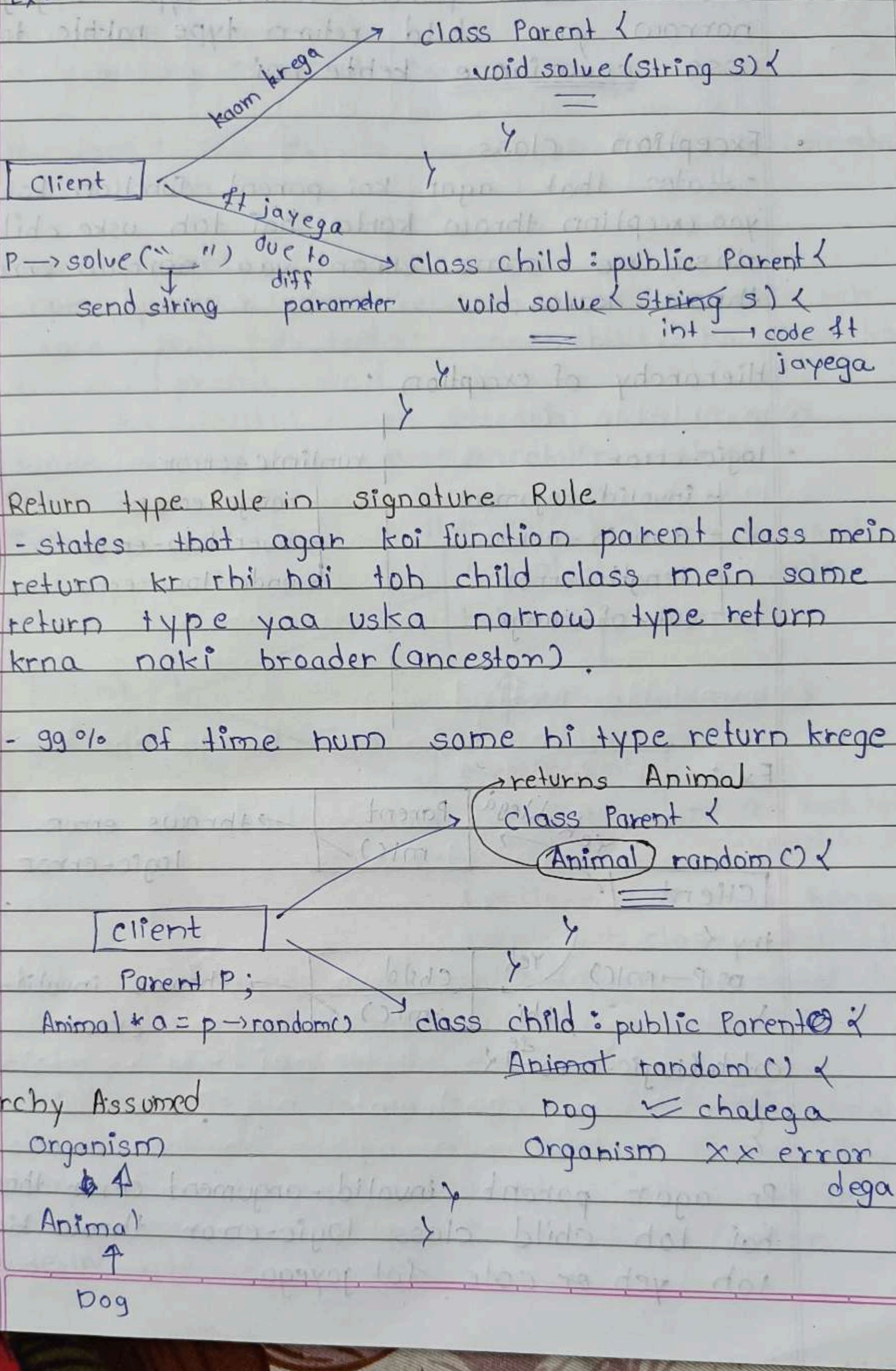
1] Signature Rule : this is method Argument Rule
- States that, agar koi function parent class mein argument leta hai toh child class mein bhi same argument lena yaa uska broader ~~ver~~ class

- Generally hum same hi argument lete hai & yeh rule CPP bhi khud compulsion krta hai even in Java

- Same argument kyu? kyuki client ko pata hai ki usse jab parent class se interact karna hai toh usse konse parameter bhejne hai agar humne internally parent ke jagah child class ko call kiya toh code fat jayega kyui different parameters.

Page No.
Date

Ex:



- Return type Rule in signature Rule
  - states that agar koi function parent class mein return kr rhi hai toh child class mein same return type yaa uska narrow type return krna naki broader (ancestor).

  - 99% of time hum same hi type return krege

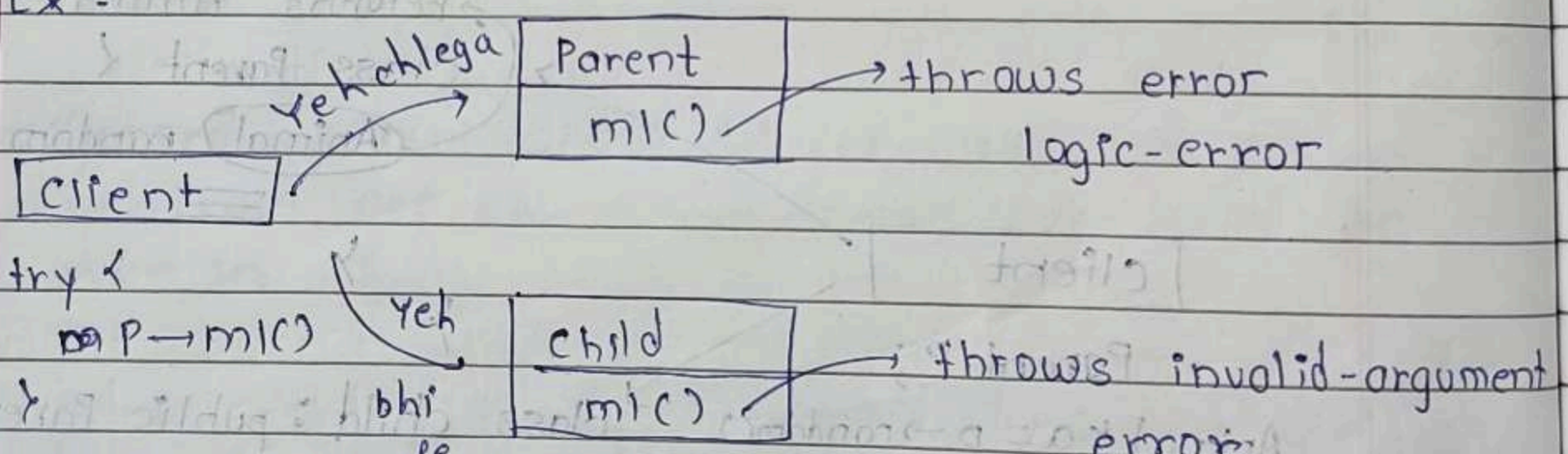
- Jab hum actual parent return type ke jagah narrow yaa child return type rakhte toh usse covariance kehte hai

- Exception class
  - states that agar koi parent function koi error yaa exception throw karta hai toh uske child class me isme error yaa narrow error throw karna.

Hierarchy of exception

- logic-error
  - invalid-argument
  - domain-error
  - length-error
  - out-of-range
- runtime-error
  - range-error
  - overflow-error
  - underflow-error

Ex :



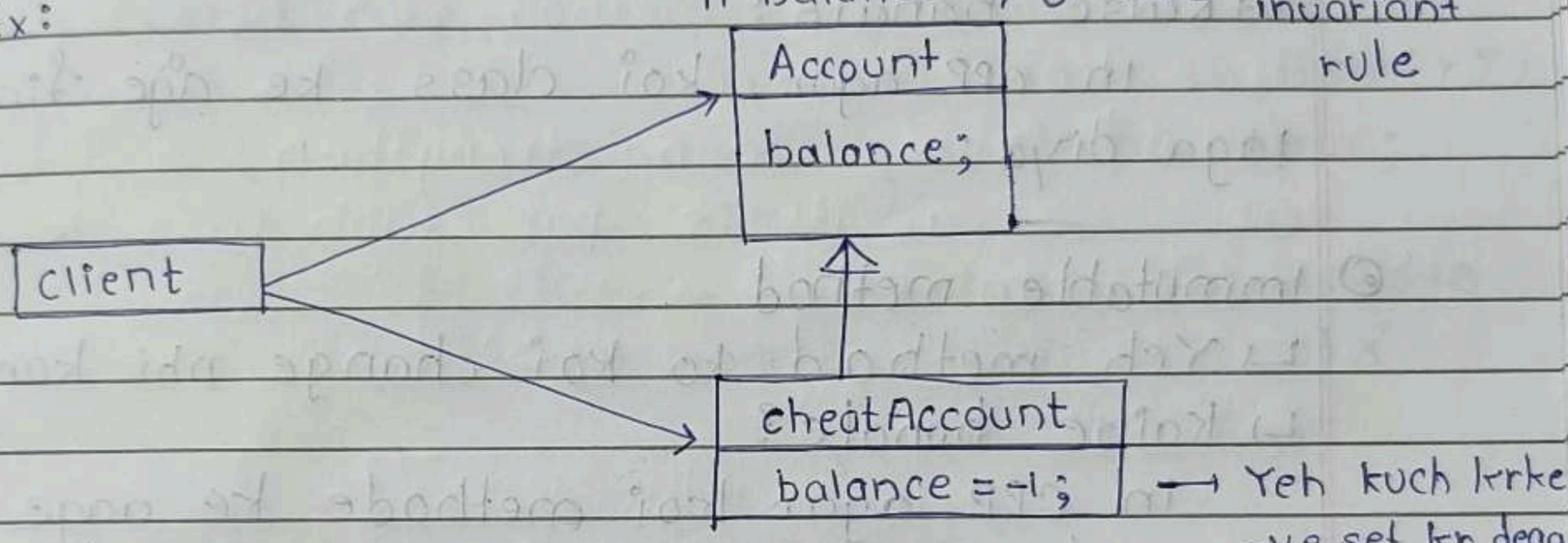
Pr agar parent invalid-argument error throw krta hai toh child class logic-error throw kiya toh yeh er code fat jayega

2. Propert Rule
   ↳ Class invariant
   ↳ history ~~Invariant~~ Constraint

- Invariant : Yeh ek rule hoga jo for class humesha true hona chahiye

* Class Invariant
-Hum yahapar class ke upar comments mein rule likhte hai jo humari responsibility ho class ko follow karana and child class bhi woh rule ko follow kare ek toh as it is yaa more strengthing par weak naa kare

Ex:



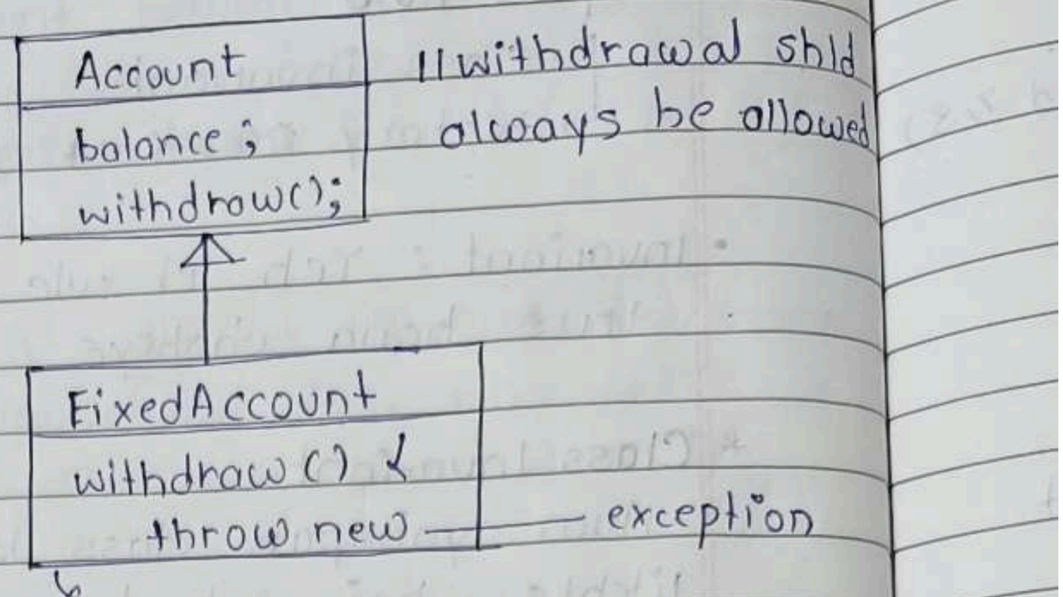
toh isse LSP break hoga cause yeh class substituatable nhi hoga for parent class

* History Constraint
-History constraint bhi hum comment krke batate above parent class and humari responsibility hoti hai ki ~~is~~ yeh property follow ho by child class . Yaha pr hum state /action define krte

* For codes check repo

Code mein acche se smjhega

Ex :



• Toh yeh liskov follow nhi krta toh yeh allowed nhi hai

⊙ Immutable class
↳ Yeh class ko koi inherit nhi kar skta naa hi koi change kar skta hai
↳ Kaise banate?
In CPP agar koi class ke age final keyword laga diya

⊙ Immutable method
↳ Yeh method ko koi change nhi kar skta
↳ Kaise banate?
In CPP agar koi methode ke aage final keyword laga diya

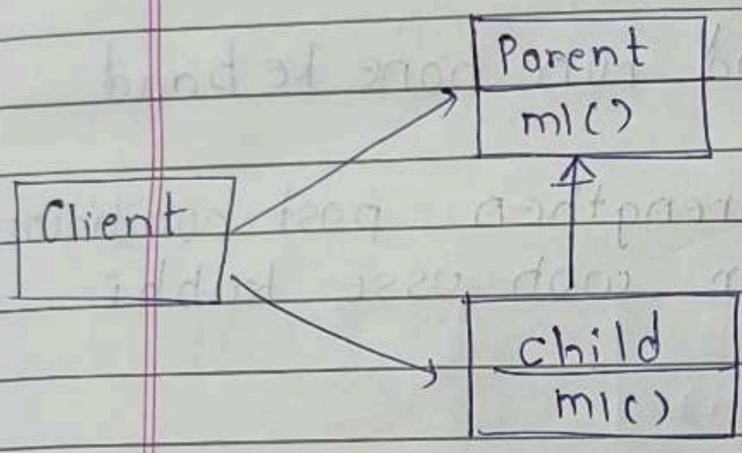
⊛ Agar parent class mein methods immutable ho and child class unhe mutable bana de toh bhi LSP break hoti hai - it is violation of history constraint in LSP

Page No.
Date

3. Method Rule
   ↳ Precondition
   ↳ Post Condition

1. Pre-condition
- Yeh bhi hum comment karke hi likhte hai & isse follow krna humari responsibility hai
- Yeh ek condition hai jab method run hone ke pehle follow hona chahiye and child class ne use ~~as~~ it is lena chahiye yaa weak krke use krna ~~as~~; strengthen allowd nhi ✗

Ex:



Pre:
// num >= 0 && num <= 5
void m1(int num) {
if (num < 0 || num > 5)
throw error;
}

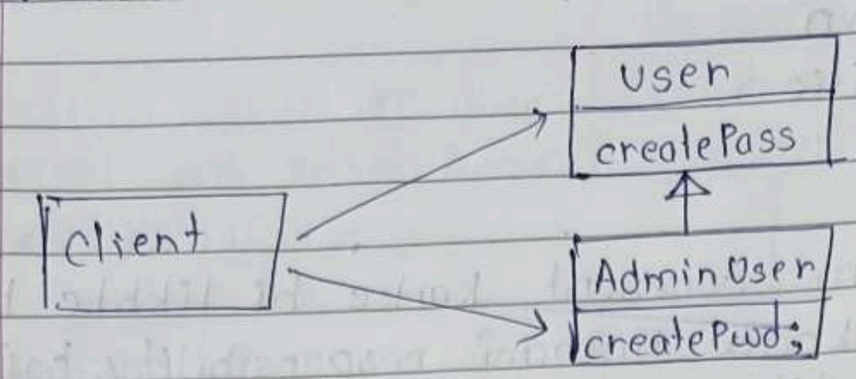
//Pre: num >= 0 && num <= 10
void m1(int num) {
}

* Client ko pata hai parent class 0-5 num accept krta hai toh ab hum usse parent de yaa child chli jayega kyoki child ne condition weak kr hai

* Agar light krta like num <= 3 toh error aata agar hum child ko call kre kyuki parent toh accept 5 tk kar raha tha... toh strong / strengthen nhi hoga...

Page No.
Date

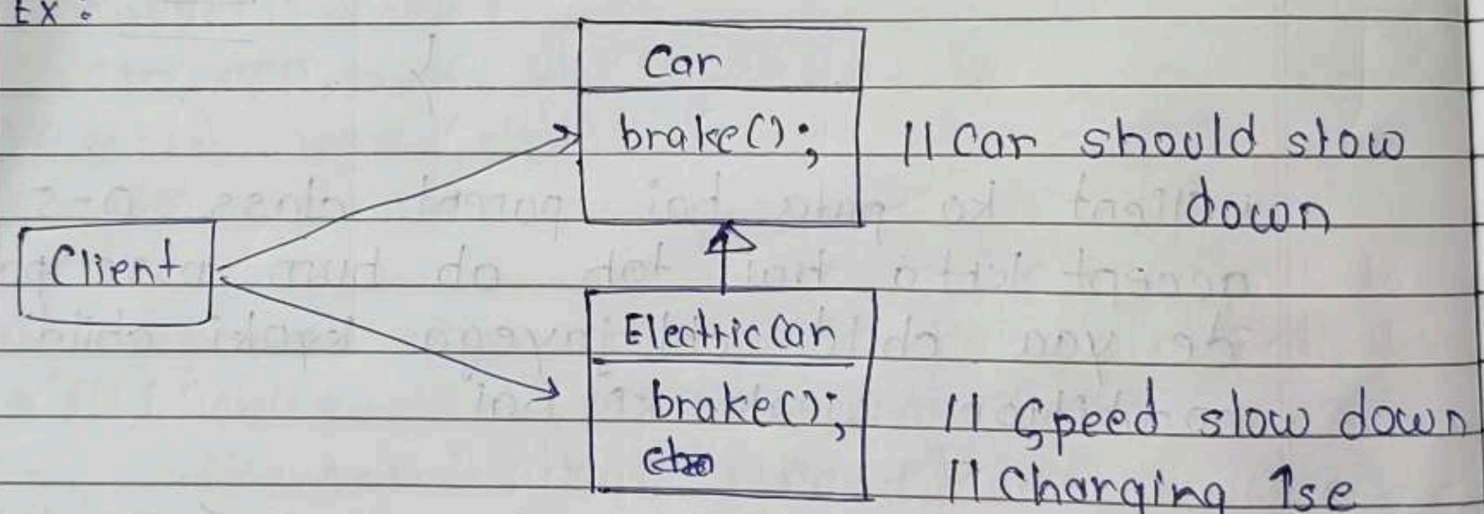
* Real-life example



Child class can be substituted in place of parent and will not throw error

2. Post Condition
- Yeh condition method run hone ke baad true honi chahiye
- Child class can strengthen post condition or use as it is par woh usse kabhi weak naa kare.

Ex:



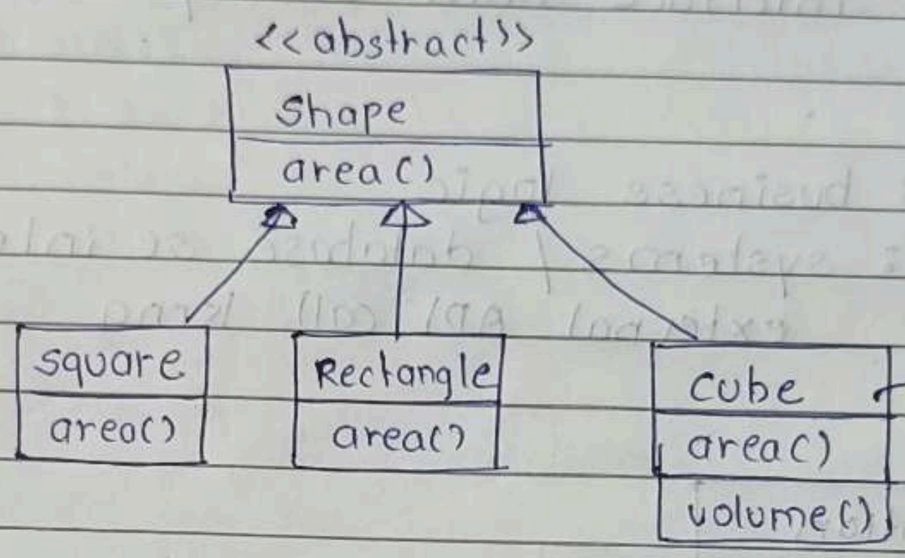
Yeh ElectricCar strengthen hi kar rha hai cause brake() dabne par speed kam ho rhi hai so it can replace parent, aisa nhi ho rha ki brake dabne ke baad speed badh rhi

4) I : Interface Segregation Principle

- Many client specific interface are better than one general purpose interface

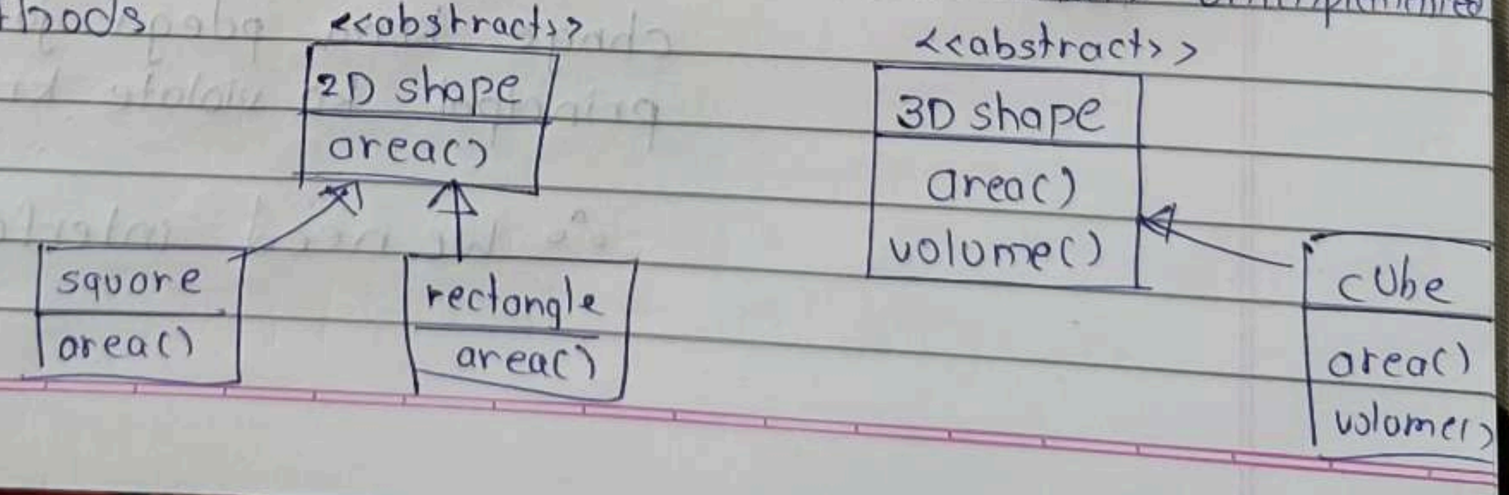
- Clients should not be forced to ~~use~~ implement methods they don't need.

Ex :



& we know ki har 3d object ka volume hota hai toh ab jaise shape add krege toh humme individually add karna padega yaa fir hum parent class mein add karte toh hume 2D object ke liye unwanted method volume ko override krna padega with exception (2nd stmt)

And yahi pe first stmt kaam aata hai ki multiple interface banao rather than unimplemented methods

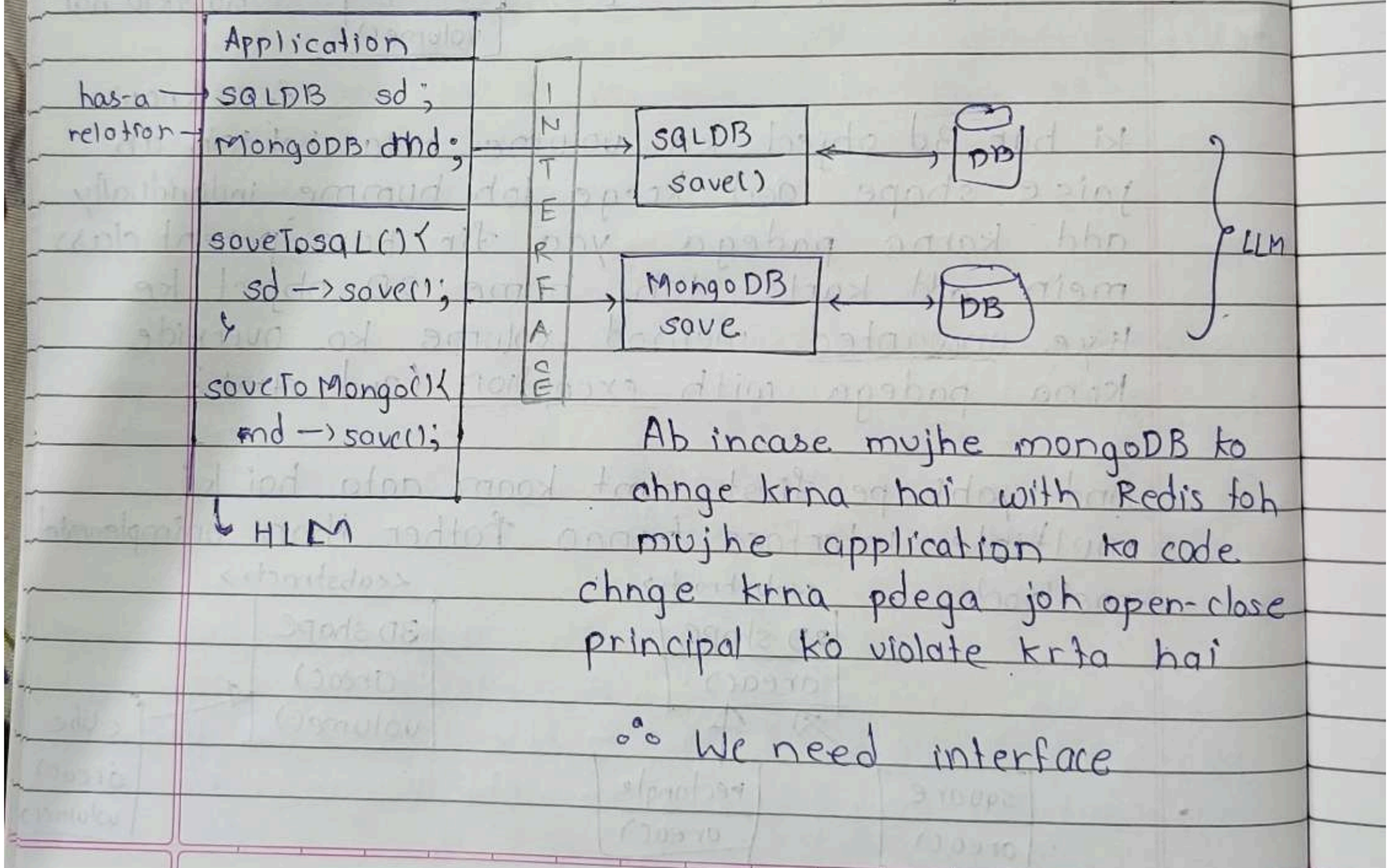


5) D : Dependency Inversion Principle

- High level module should not depend on low level module but rather should depend on abstraction.

Inshort, humara job HLM woh directly LLM se interact naa kare bich mein abstraction / ~~infere~~ interface hona chahiye.

- HLM : business logic
- LLM : systems / database se interact krna external API call krna

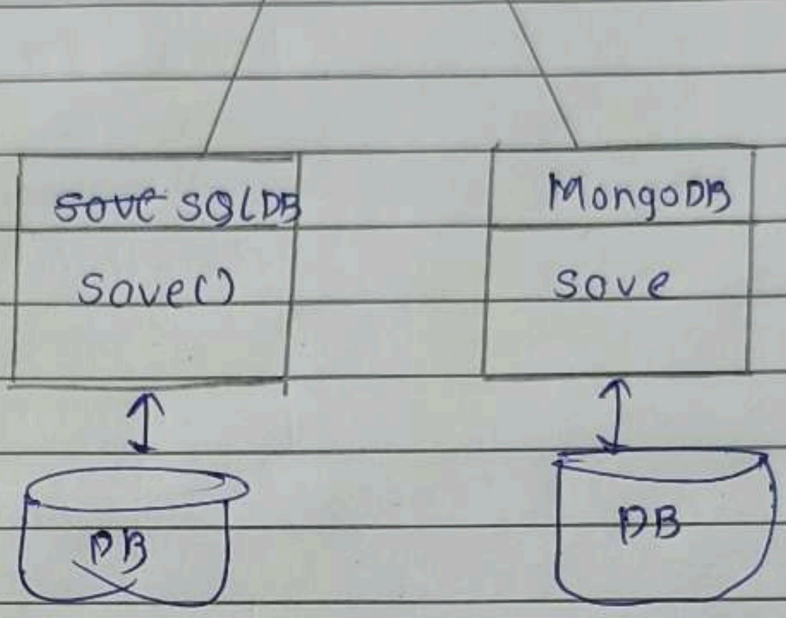


Ab incase mujhe mongoDB ko chnge krna hai with Redis toh mujhe application ka code chnge krna pdega joh open-close principal ko violate krta hai

∴ We need interface

Page No.
Date

Solution



Ab incase hume mongodb nikalna hai toh bi kuch farak nhi pdega humare application code ko & Persistance act krega as interface / abstraction jisse OCP yaa DIP violate na ho....