Solid Design Principles

✳ Problems in real-time Systems if we don't follow design Principles

① Less Maintainable → new features should be easily integrated.

② Less Readability → Due to Tight Coupling.

③ New Bugs !!

S O L I D

S: Single Responsibility Principle [SRP]

O: Open-Close Principle [OCP]

L: Liskov Substitution Principle [LSP]

I: Interface Segregation Principle [ISP]

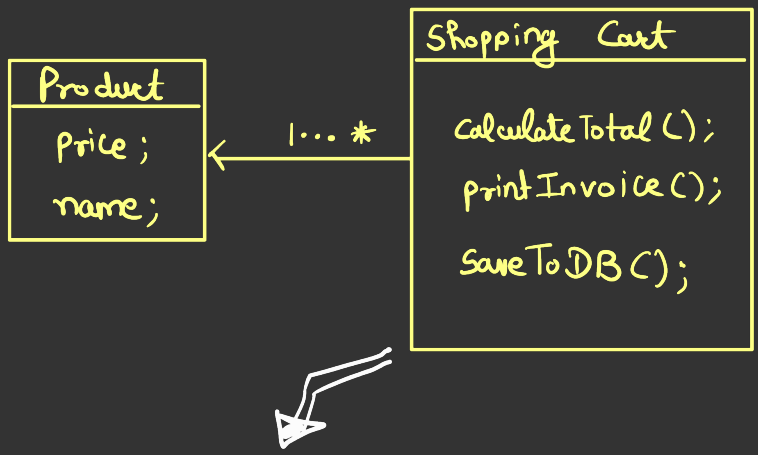
D: Dependency Inversion Principle [DIP]

✳ 5 : Single Responsibility Principle.

- A class should have only one reason to change.
- A class should do only one thing.

↳ Let's assume Shopping Cart use Case

Case1



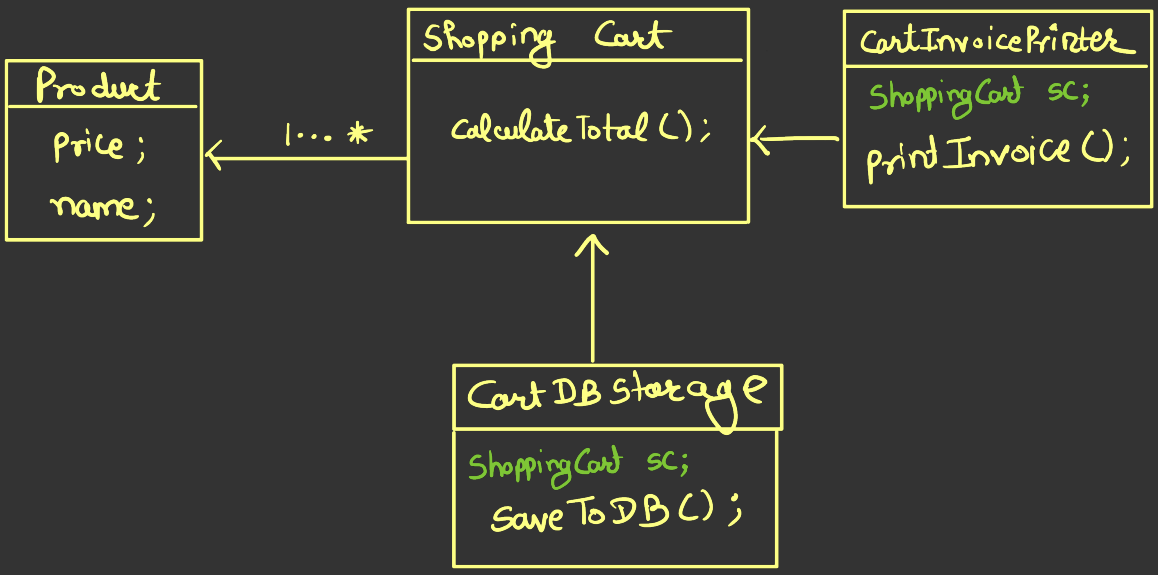
This is breaking single responsibility Principle

→ Shopping cart class needs to be changed, If

1. Change in DB pessist logic
2. Change in Invoice Printing Logic
3. Change in Total calculn Logic

So we have to follow SRP.

↳ We will use Composition to do so.



⊙ SRP doesn't mean One class can have only 1 method. A class can have any number of methods, its just that all method should be serving 1 particular responsibility

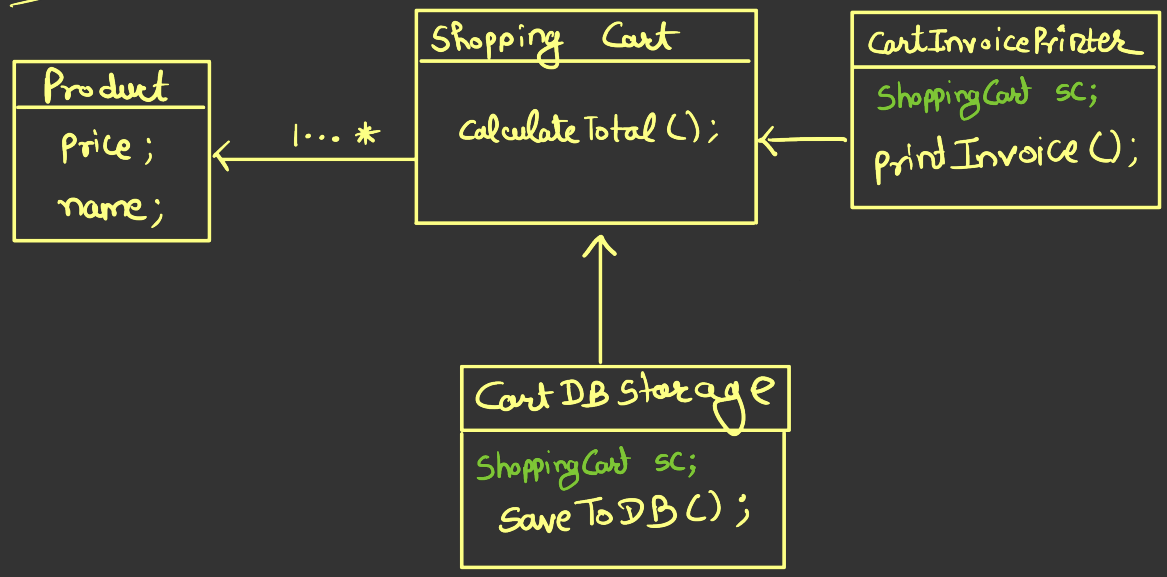
⊛ O : Open-Close Principle

↳ A class should be open for extension but closed for modification.

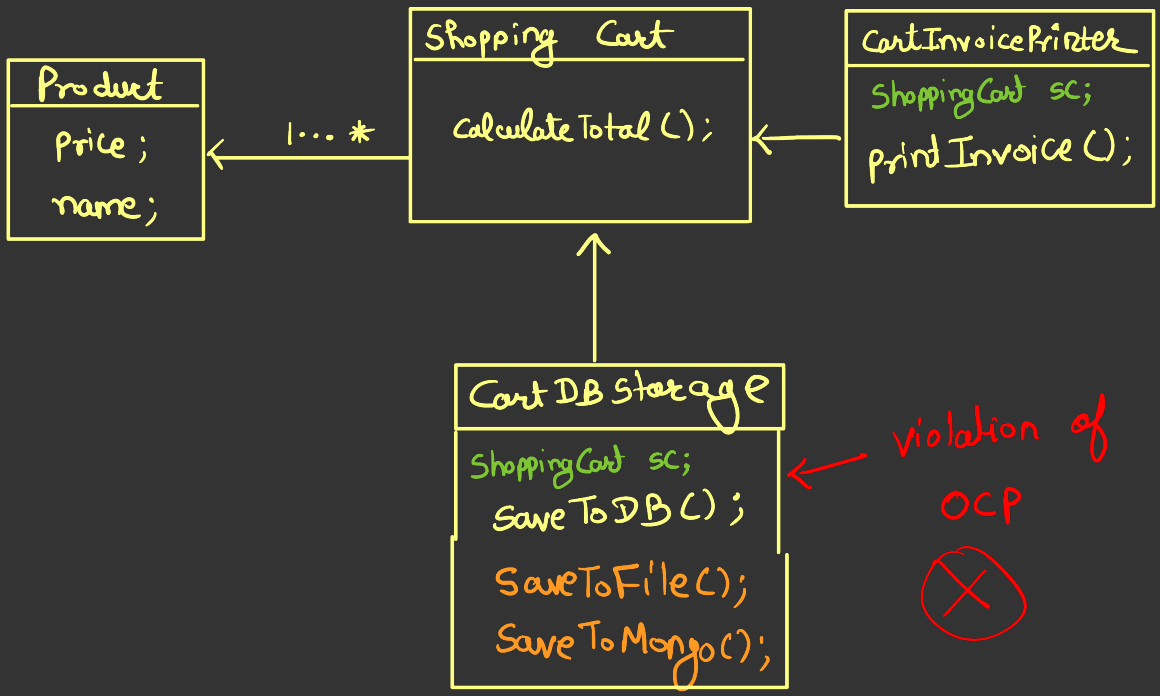
↳ pura code ko change karna

This means, naye feature ko implement karne ke liye koi purane methods me change nahi aana chaiye.

Diagram till now

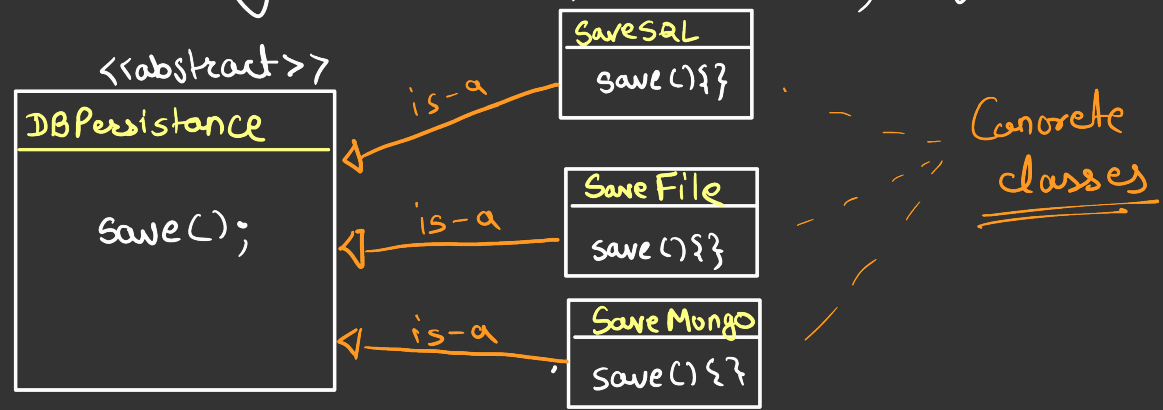


Now, I want to save in File and also MongoDB.



How to Implement OCP.

↳ using Abstraction, Inheritance, Polymorphism



↑ Interface → Contract bet^n APP & client.

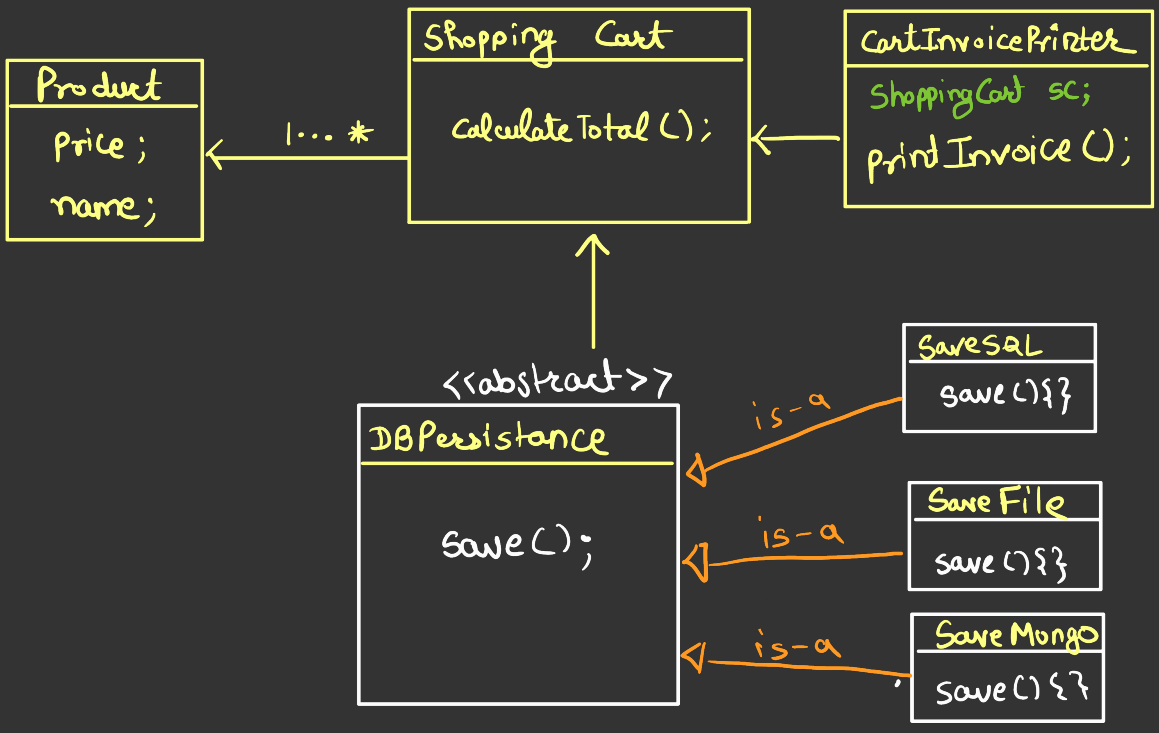
Point to Remember

Client

INTERFACE

Class

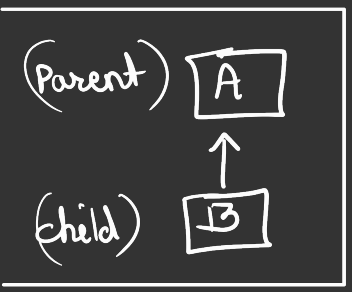
OCP Followed :-)



If in future, if we want Cassandra - Persistence we can add a new Concreate class.

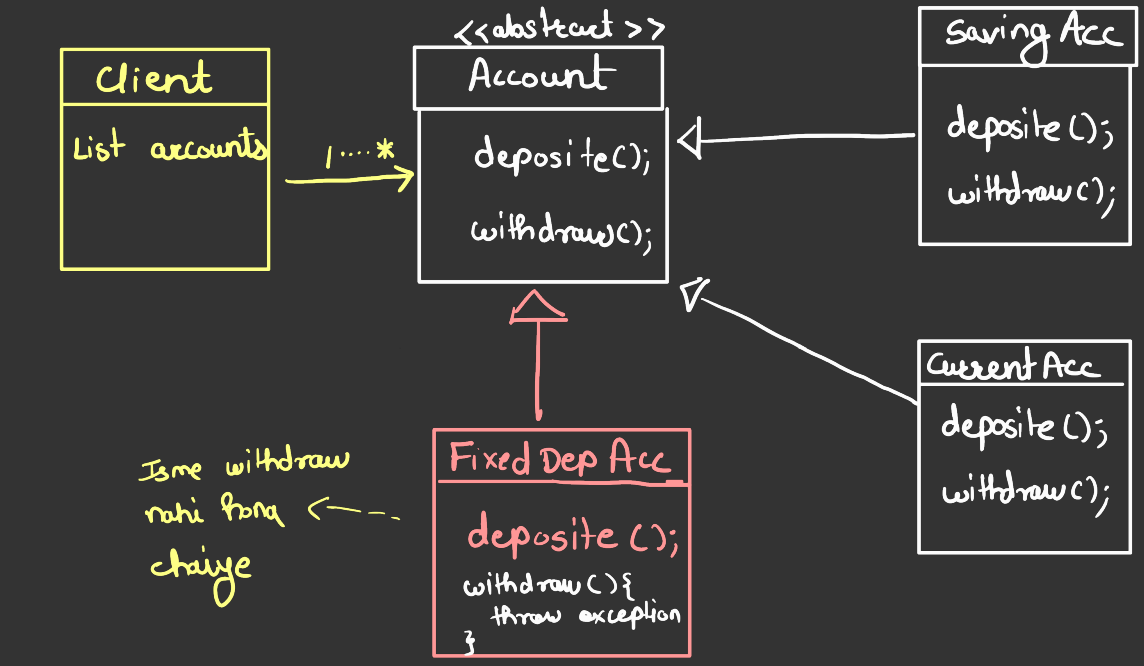
L: Liskov Substitution Principle (LSP)

↳ Sub classes should be substitutable for their Base classes.



Let's say a method want class A's object in argument & if we pass class B's object then also it should work absolutely fine.

Eg:-



now, If client calls withdraw() on FDAccount then exception will be thrown. Hence Violation of Liskov's Substitution Principle.

How to Implement Solution ?

① Bad Approach

Client ke code me change karo.

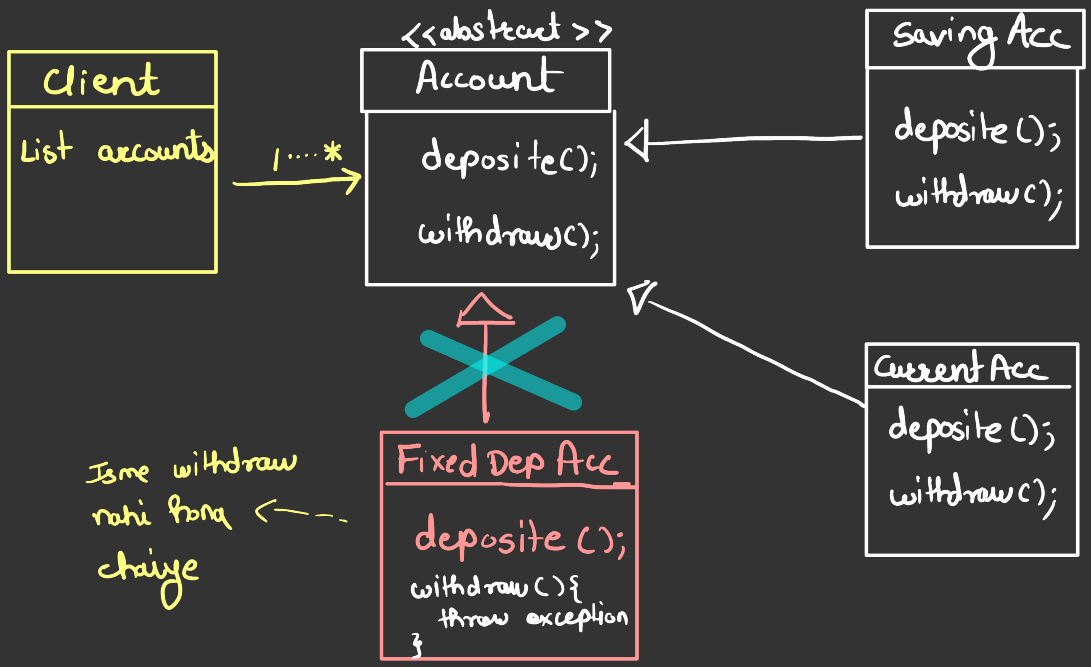
Sample code of client class →

```
for (acc : accounts) {
    acc.deposite();
    if (! acc == FDAcc){
        acc.withdraw();
    }
}
```

This is bad because now Client is tightly coupled with the mentioned types of accounts. Ideally client should only interact with Interface

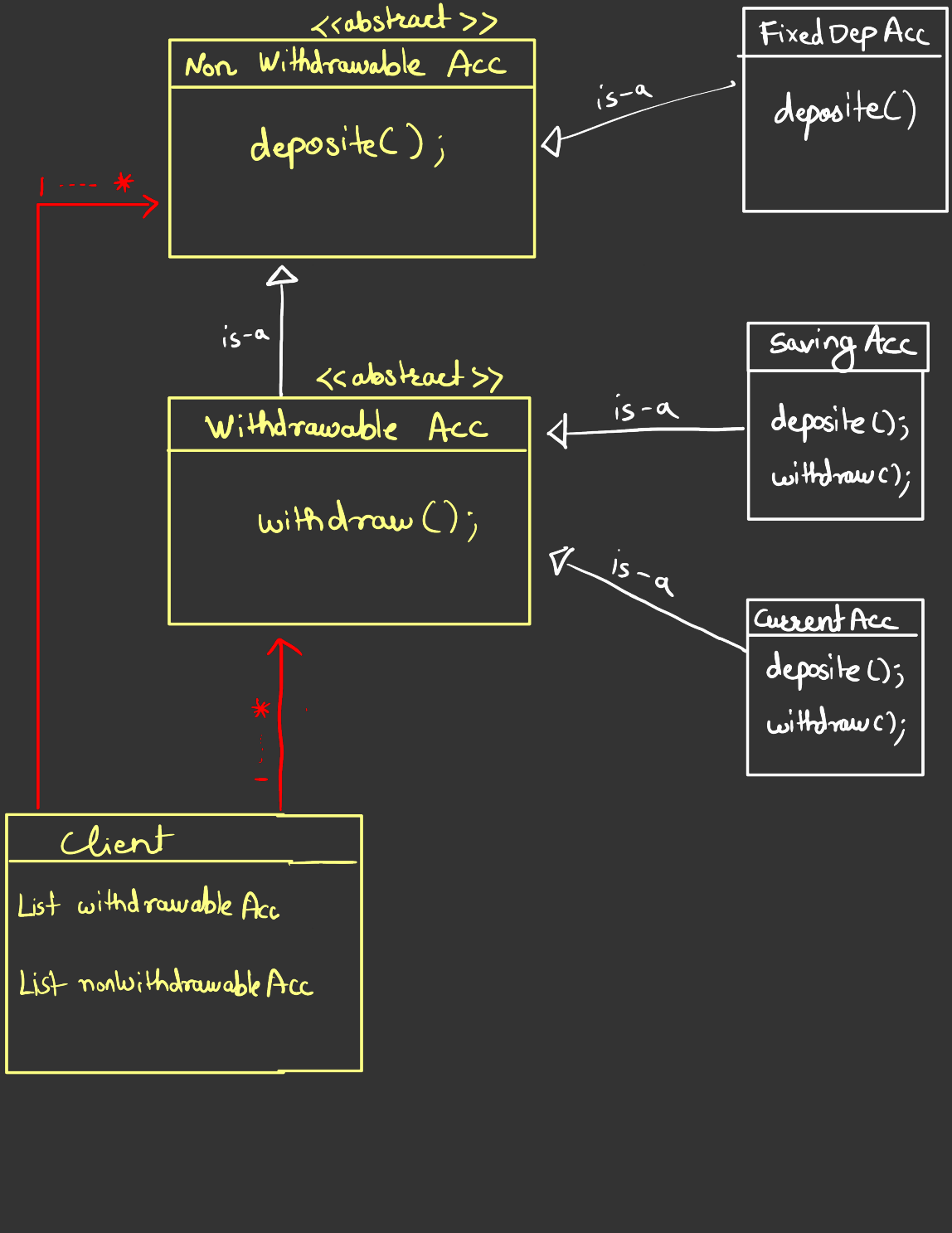
Tightly coupled hote hi, this will break OCP.

② Correct Approach



↳ Isme Account me 2 methods hai, but Fixed Dep Acc me sirf ek method hai, to Fixed Dep Acc ko Account class ka child banana thi nahi chaiye.

↳ Parent ke feature ko kam kar raha hai


<<abstract>>
Non Withdrawable Acc
deposite();
Fixed Dep Acc
deposite()
is-a
1
*
is-a
<<abstract>>
Withdrawable Acc
withdraw();
Saving Acc
deposite();
withdraw();
is-a
is-a
Current Acc
deposite();
withdraw();
*
1
Client
List withdrawable Acc
List nonWithdrawable Acc