May
20-05-25
Monday

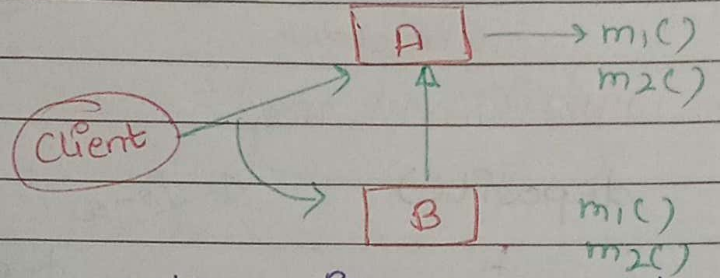
DAY-7
Lecture-5

Don't wish for it. ~~Work for it.~~

# SOLID Design Principles (Part-2)

## * Deep Dive : Liskov Substitution Principle (LSP)

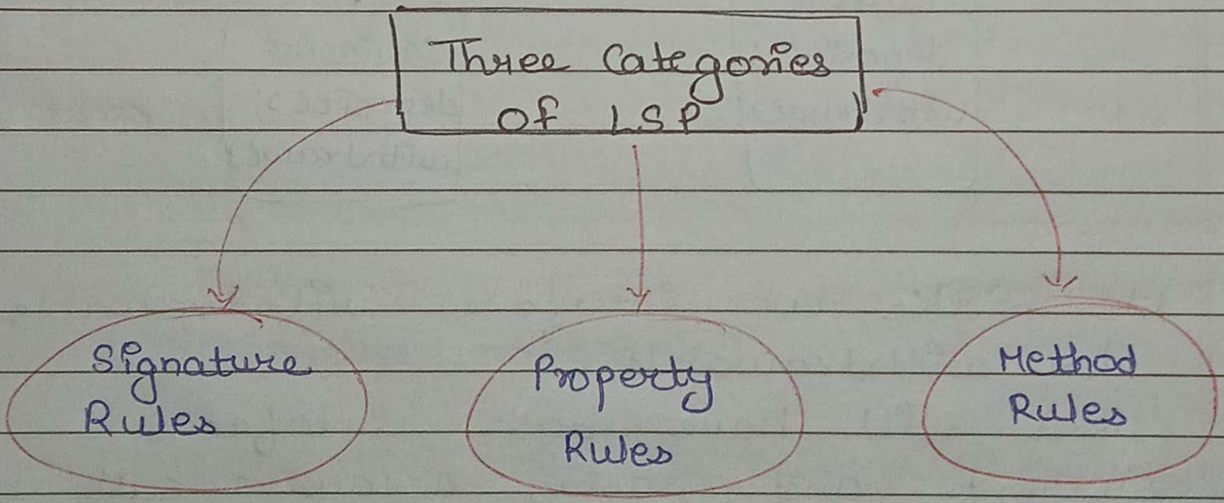
Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program



B should not only inherit & methods from A class. It should behave like A class

If a prog. works with a parent class, then it should also work correctly if we use a child class in place of parent class.

Guidelines of LSP



(i). Signature Rules :

In programming, method signature means :

- method name
- Input parameters
- Return type

When a child class overrides a method from parent class, it must follow the same signature ~~as~~ rules, so that the program doesn't break.

→ Method Argument Rule: The overriden method in a subclass must accept the same argument types as the parent, or wider.

Example:
```
Class A {
      void show (int x)
};
Class B : public A {
      void show (int x)
}

Class B: public A{
      void show (string x)
}
```
same

Not this

→ Return Type Rule: Child method must return the same type (or a subtype) as the parent method.
or narrow

Example:
```
ClassA {
    Animal getAnimal();
}

Class B : public A {
    Animal getAnimal() {
    or Dog
}
```

$$\begin{tikzcd} \boxed{\text{Organism}} \\ \boxed{\text{Animal}} \arrow[u] \\ \boxed{\text{Dog}} \arrow[u, "\text{is-a}"'] \end{tikzcd}$$

```
Animal *a = p→getAnimal()
```

Here in the reference we can Dog class }
but not organism class

because organism class is a parent class and we make a child class object

```
Class A{

    Animal random(){
           }

}

 Class  B: public A{

      Dog  random(){

      }

 }
```

Here if we child class object as a return type then this is known as covariance

→ Exception Rule: A child class should throw fewer or narrower exceptions but not additional or broader exception than the parent class.

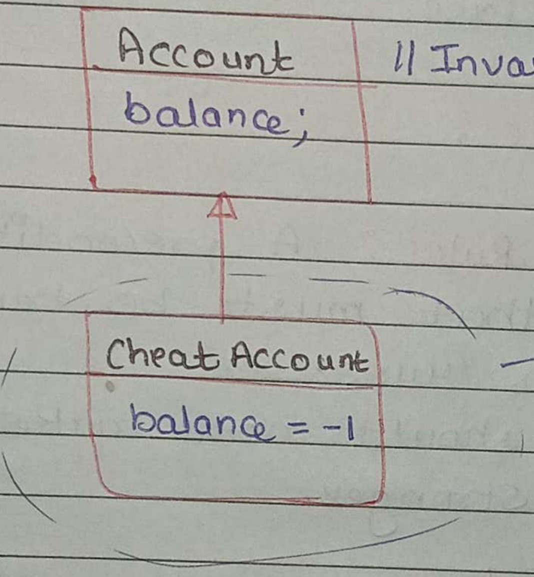
~~[illegible]~~ logic error
- invalid-argument
- domain-error
- length-error

runtime-error
- range-error
- overflow-error

If parent class is throwing logic-error then child can throw ~~[illegible]~~ invalid-argument, domain-error but it can't throw runtime error or range-error.

(ii) Property Rules: ~~[illegible]~~ Means that the child class should keep and respect the important things (property or behaviour) of the parent class.

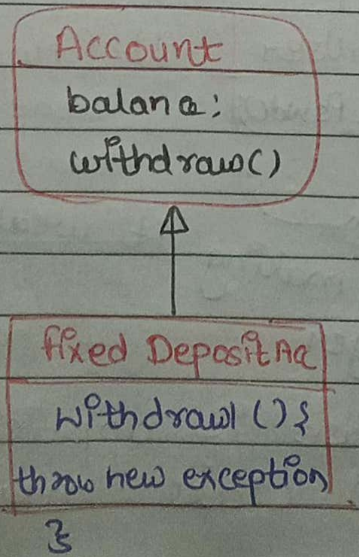
→ Class Invariant - It is a rule or condition that must always be true for an object before and after any public method is called.



// Invariant : balance should never be negative

→ This class break the rule it make the balance negative

→ History Constraint: The subclass must preserve the "history" or lifecycle behaviour of the parent.



// Withdrawal should always be used

} This child breaking the history by throwing an exception

Not inheritable

Immutable class → final } final keyword is used in C++ to make class and method immutable

Immutable methods → final

↓

No one can override this methods

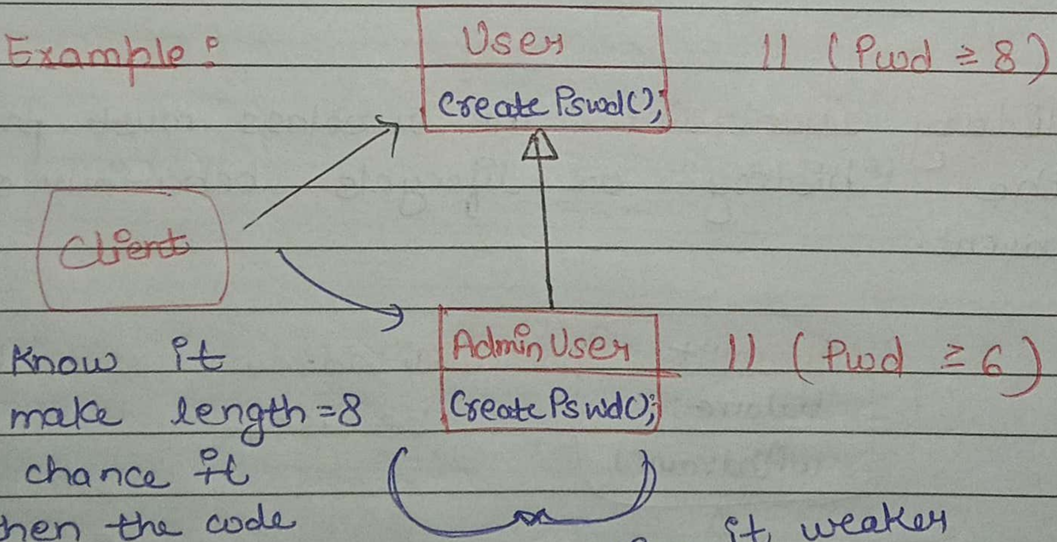
History constraint will break if child class change the immutable class to mutable class

(iii) Method Rule :

→ Precondition Rules : A precondition is something that must be true before the methods runs.

→ Child class should not make the precondition stronger.

Example :



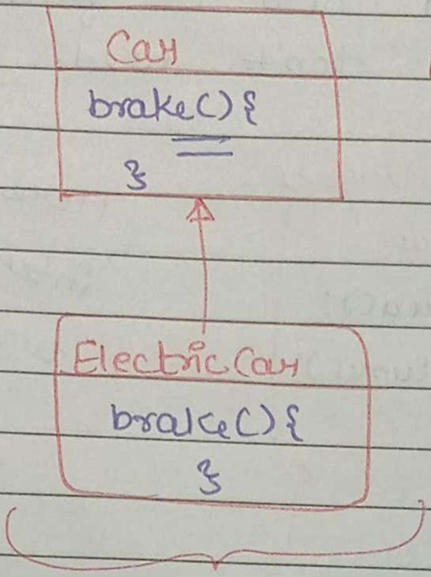
Client Know it should make length = 8 but by chance it pass 7 then the code won't break because child class allow that

Child Class making it weaker not stronger

Depend on use case if application demands pswd ≥ 8 then child class must follow pswd ≥ 8

→ Postcondition Rules: A postcondition is something that must be true after the method finishes running.

→ Child class should not make the postcondition weaker



// Car should slow down

// Speed ↓, Charging ↑

Here child class making it even stronger by increasing the charging.

## * Key Takeaways for LSP

i) Check Behaviour, Not Just code - Just because a child class doesn't give errors and the code compile doesn't mean it follows LSP.

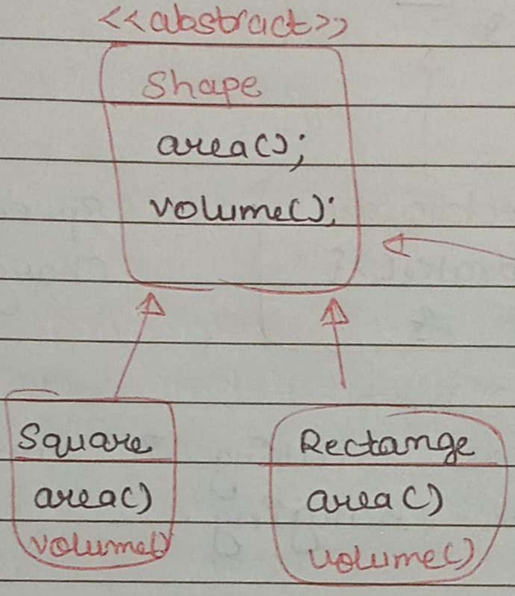
ii) Use Rules as a checklist - LSP has three main rules and you should remember that.

iii) How to stop LSP violations - If something breaks LSP, you might see -

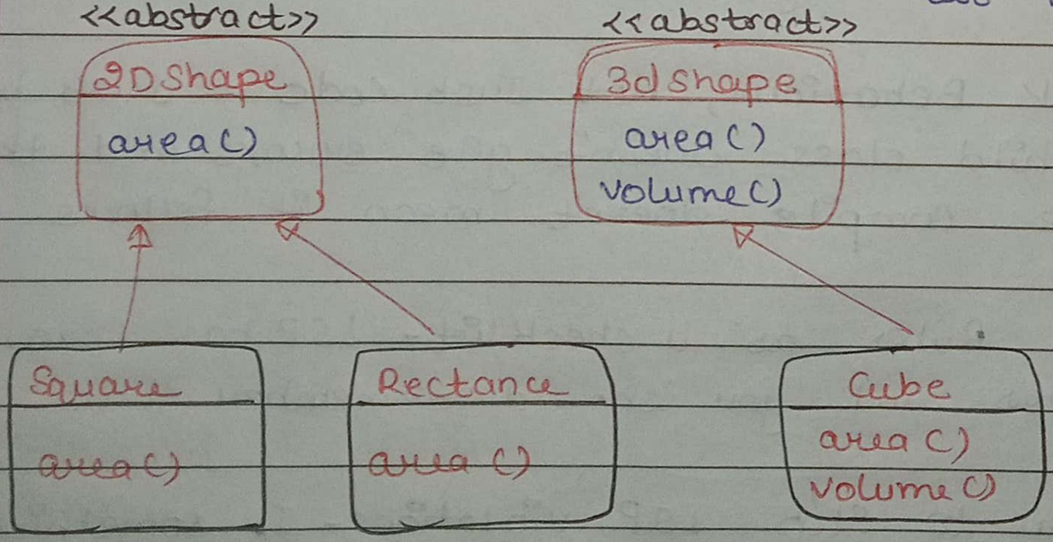
- Unexpected errors or exceptions.
- Wrong output values
- Broken rules in class (like -ve balance)

## * I: Interface Segregation Principle -

→ Many client specific are better than one general purpose interface.
→ Client should not be forced to implement methods they don't need.



Here there is one interface and it has two methods but square and rectangle don't need volume but they have to define it. cube only needs volumes so we can make two interface

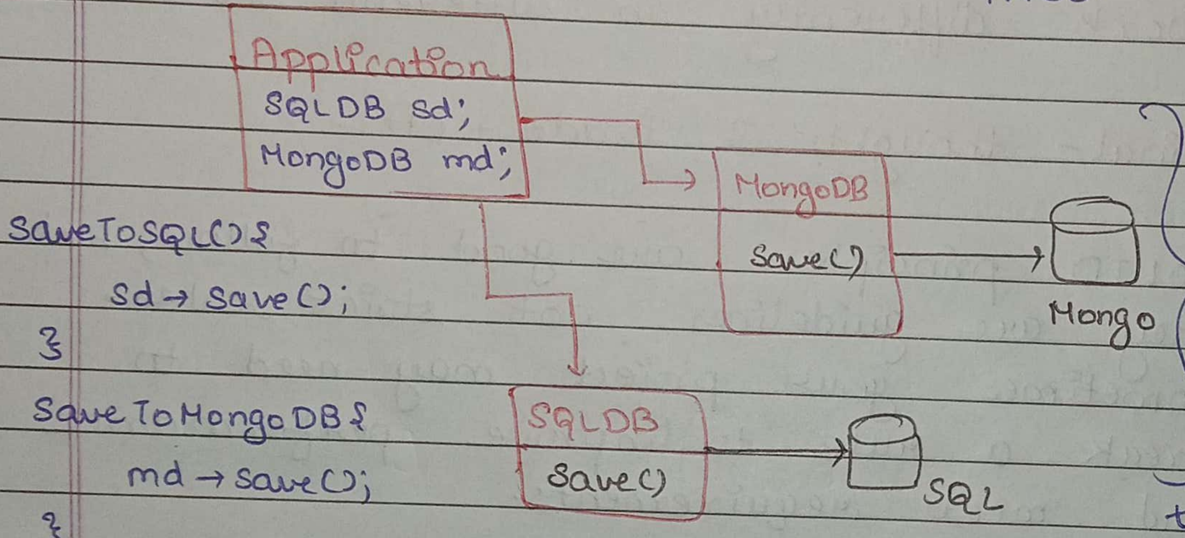


Now Here square and rectangle both have to call only area and cube can call methods from 3d shape

## * D : Dependency Inversion Principle

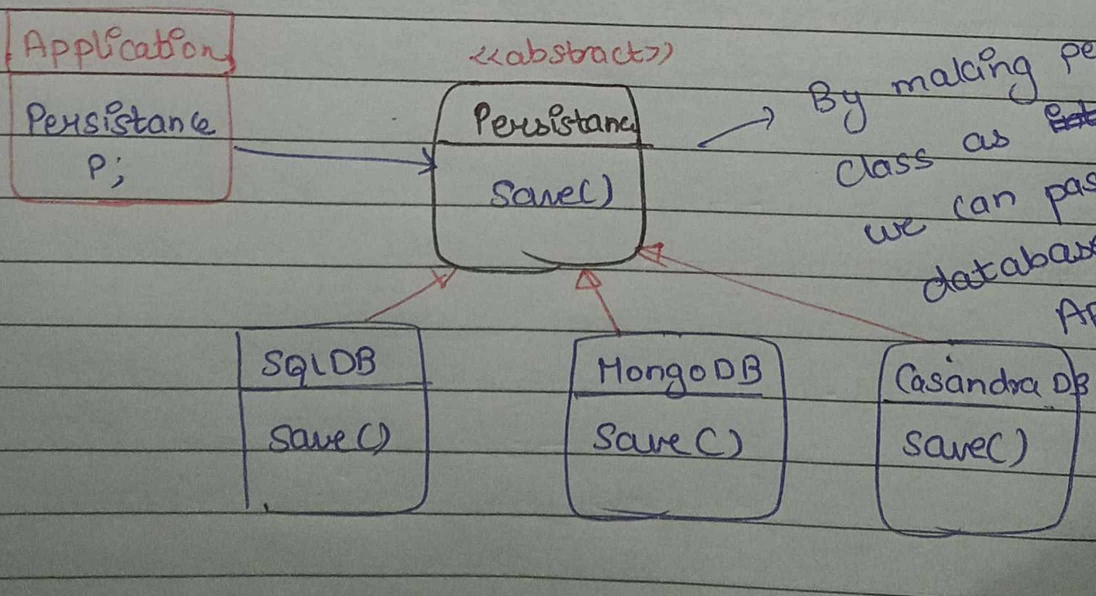
High-level modules should not depend on low-level modules. Both should depend on abstractions (interfaces).

| High-level module | Low-level module |
|---|---|
| ↓ | ↓ |
| Main logic or business logic | Helper classes like databases, file readers, APIs. |



```
saveToSQL(){
    sd → save();
}
saveToMongoDB{
    md → save();
}
```

Application is tightly coupled with Mongo and SQL and in future if we want to switch CasandraDB from Mongo then we must have to change App code



By making persistance ~~~~ interface class as we can pass any database to the Application without changing app code

• Real - World Analogy: CEO (high-level) doesn't directly talk to every developer. Instead, the CEO gives instruction to the Manager (interface) and managers handles the team and tells the developers ~~[illegible]~~ (low - level) what to do.

→ In real - life we can see human object is very complicated but human work differently in different situations.

* Final- thought & Trade - offs:

SOLID principles are good to follow, but they are guidelines, not strict rules. Sometimes your project may need to break a rule to improve performance and meet requirements.