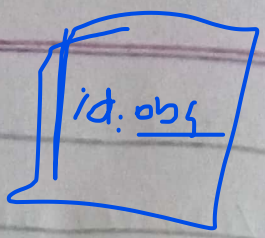
SDA

Observer Pattern:



Context:

In two way association when you compile one class the other has to be available, so if you reuse one class, you have to reuse other.

Problem:

How do you reduce interconnection between classes.

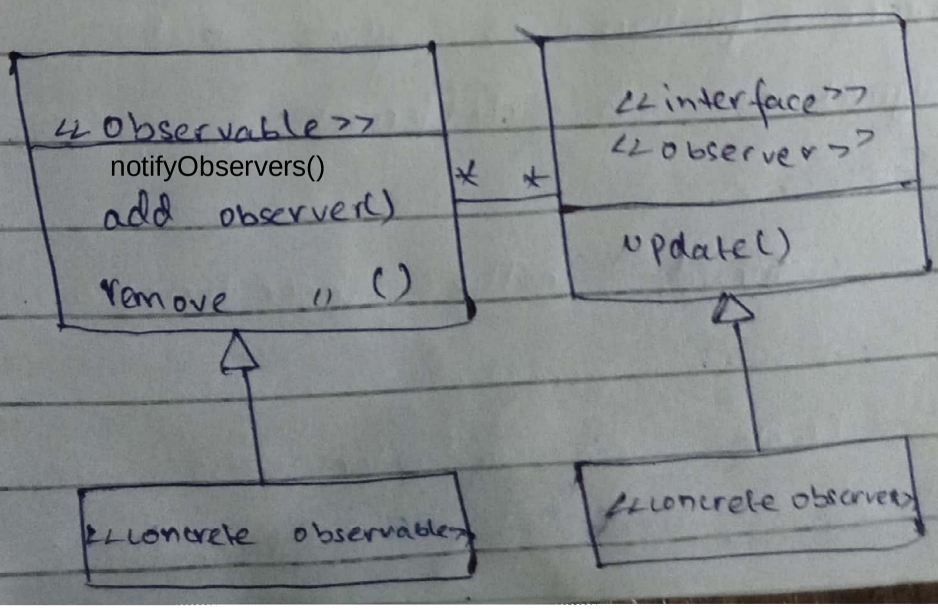
Forces:

Maximize, flexibility.

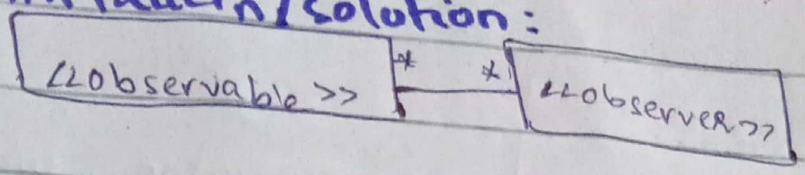
Solution:

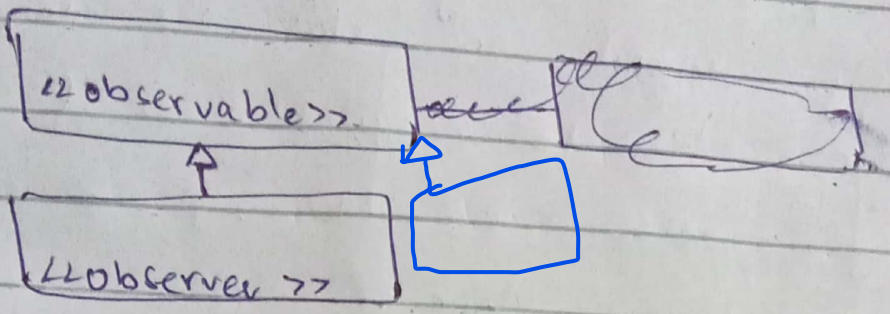
-> Abstract class observable, it add and remove observers.

-> an interface observer.



Anti Pattern / solution:

1- 

2- 

Adapter Pattern:

Context:

In inheritence hierarchy, you want to reuse existing class that do not have the same name or argument type.

Problem:

How to implement polymorphism, where funtion is same but signature is different.

Forces:

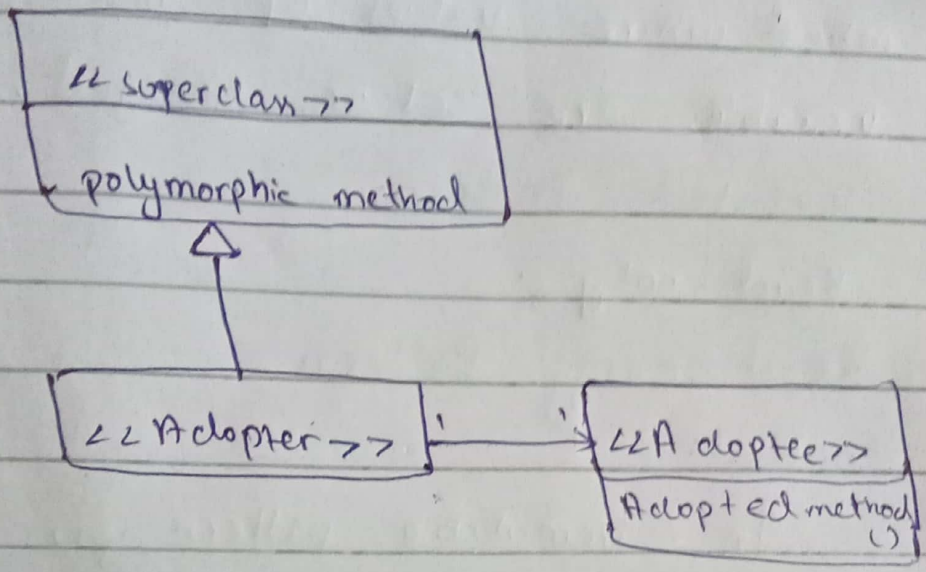
Avoid multiple inheritence.

Solution:

→ Create an adapter class which is called by adoptee class.

-> adopter can delegate the method to adoptee.

-> adoptee method may or may not have same name or adopter.



Related Pattern :

-> Facade

-> Read-only

-> Proxy

Immutable Pattern:

context:

immutable has a state that never changes.

problem:

How do you create a class whose interfaces are immutable?

Forces:

immutability must be enforced.

Solution:

A constructor is only placed where values of instance variable are set or changed.

Related Pattern:

Read-only.

- **Read-only Pattern:**

Context:

In immutable pattern, you sometimes wants to make changes to the instances.

Problem:

How do you create a situation where some ~~same~~ class see a class as read only while while others can make changes in it.

Forces:

Public, private, protected classes.

Solution:

Create mutable class, all

→ Classes created from this class can make changes to immutable class.

→ Create interface read-only, that has read-only operation of mutable.