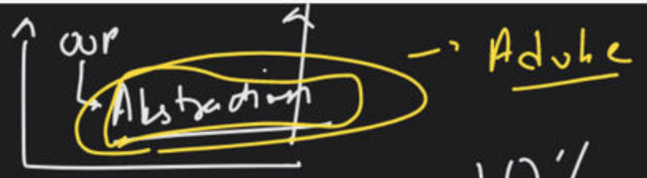
OOPs Design Patterns - Part II & Doubt Clearing Session

Course on OOPs Design Fundamentals

Love Babbar • Lesson 3 • May 29, 2022

Structural Design Pattern ?



10%

→ simplifies the structure by identifying the relationship.

types

→ Adapter → Proxy → facade

→ Bridge → Composite → Decorator

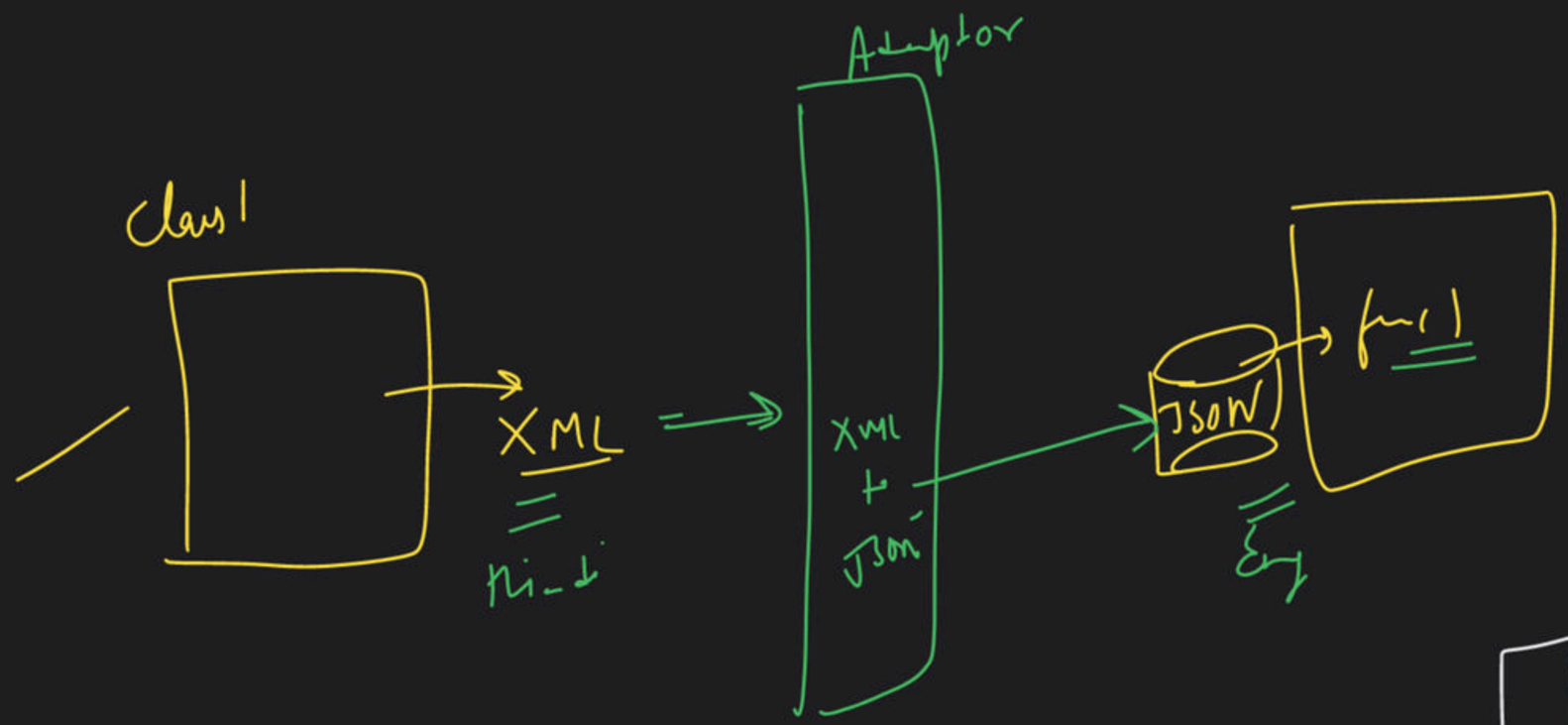
→ flyweight

→ Adapter DP:- [2 incompatible interfaces → Compatible]

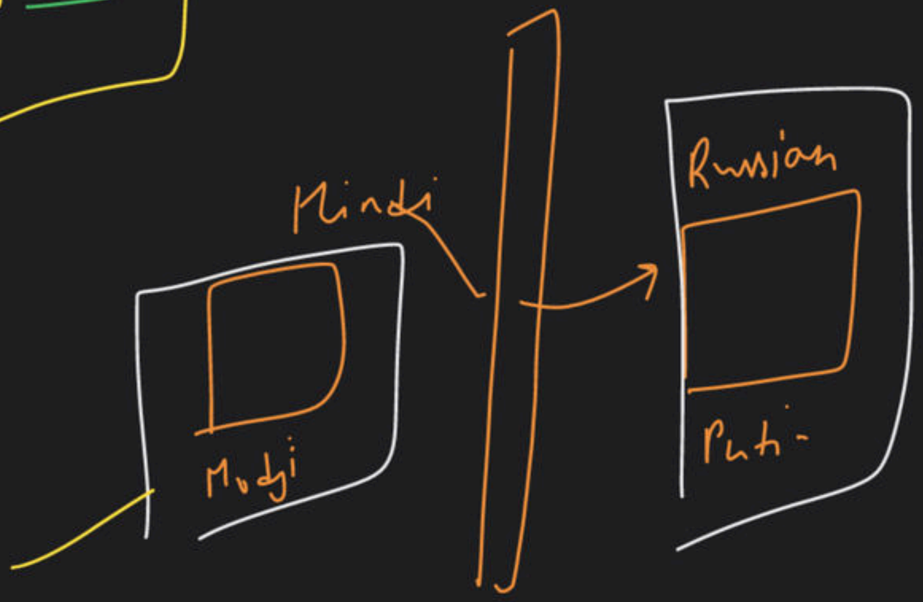
aka wrapper

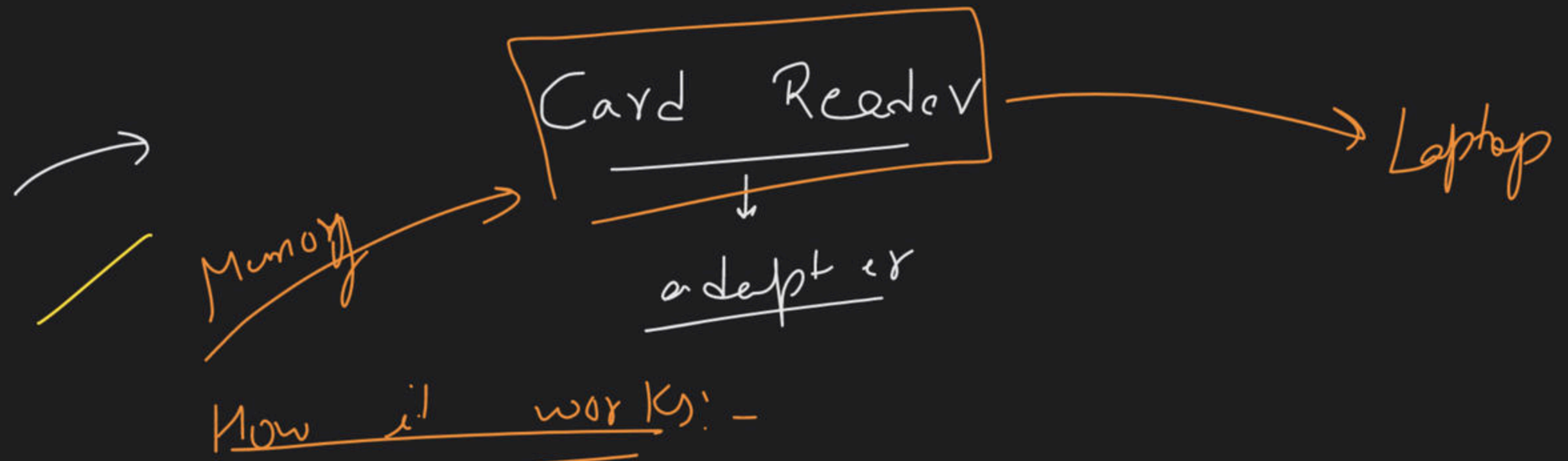
→

→ The AP converts the interface of a class into another interface that client expects:



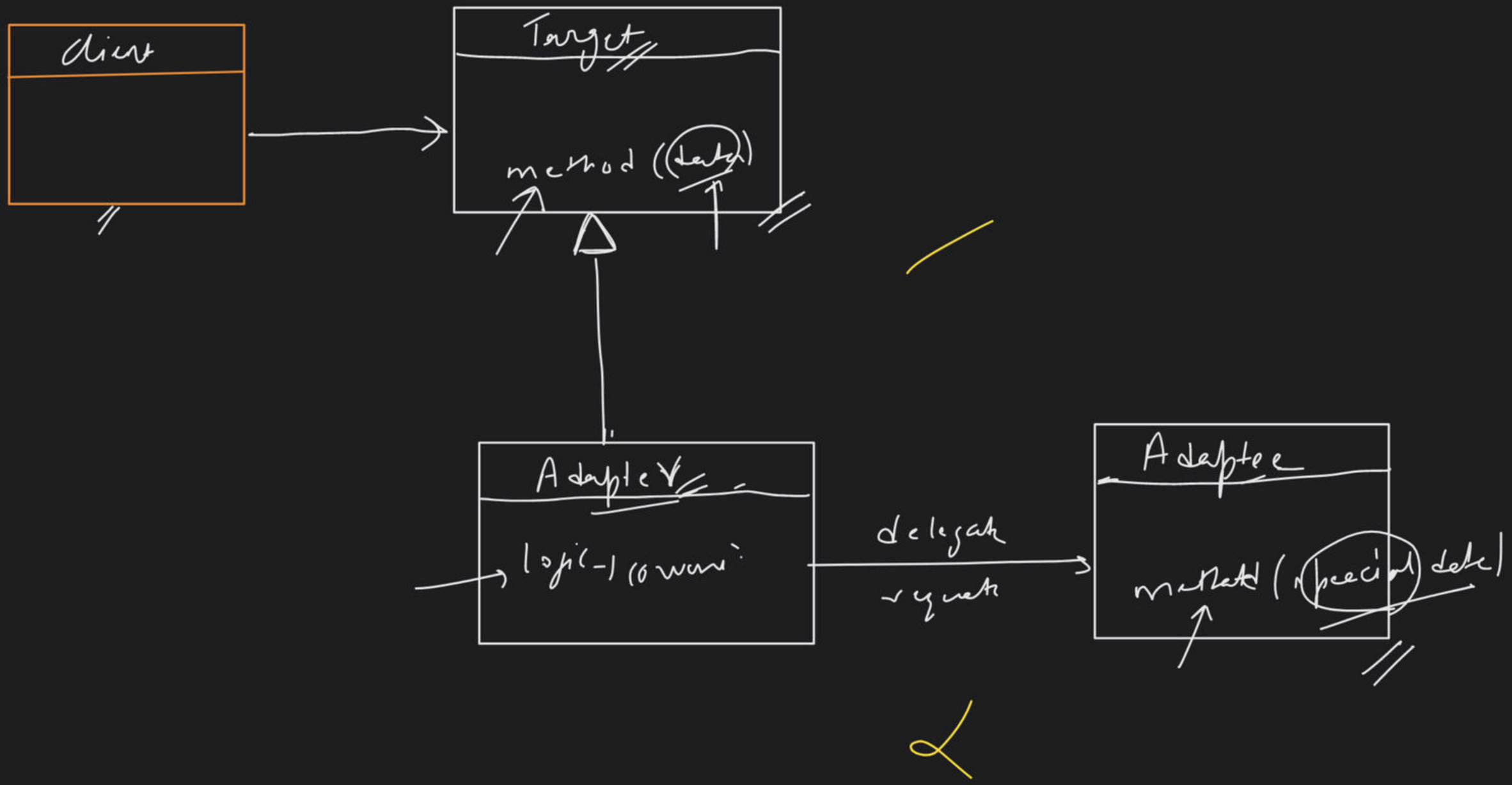
→ Ex -> ?

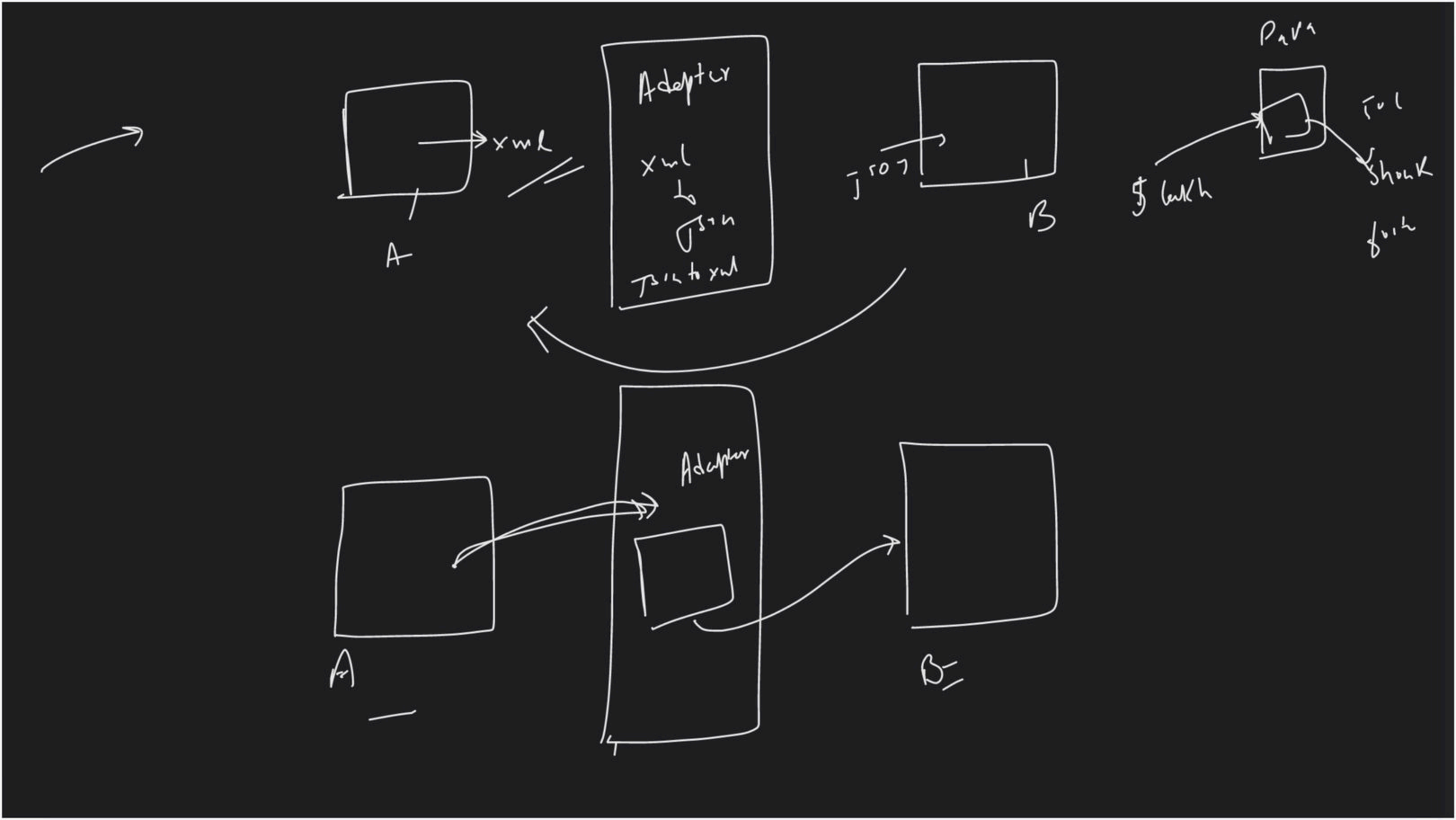


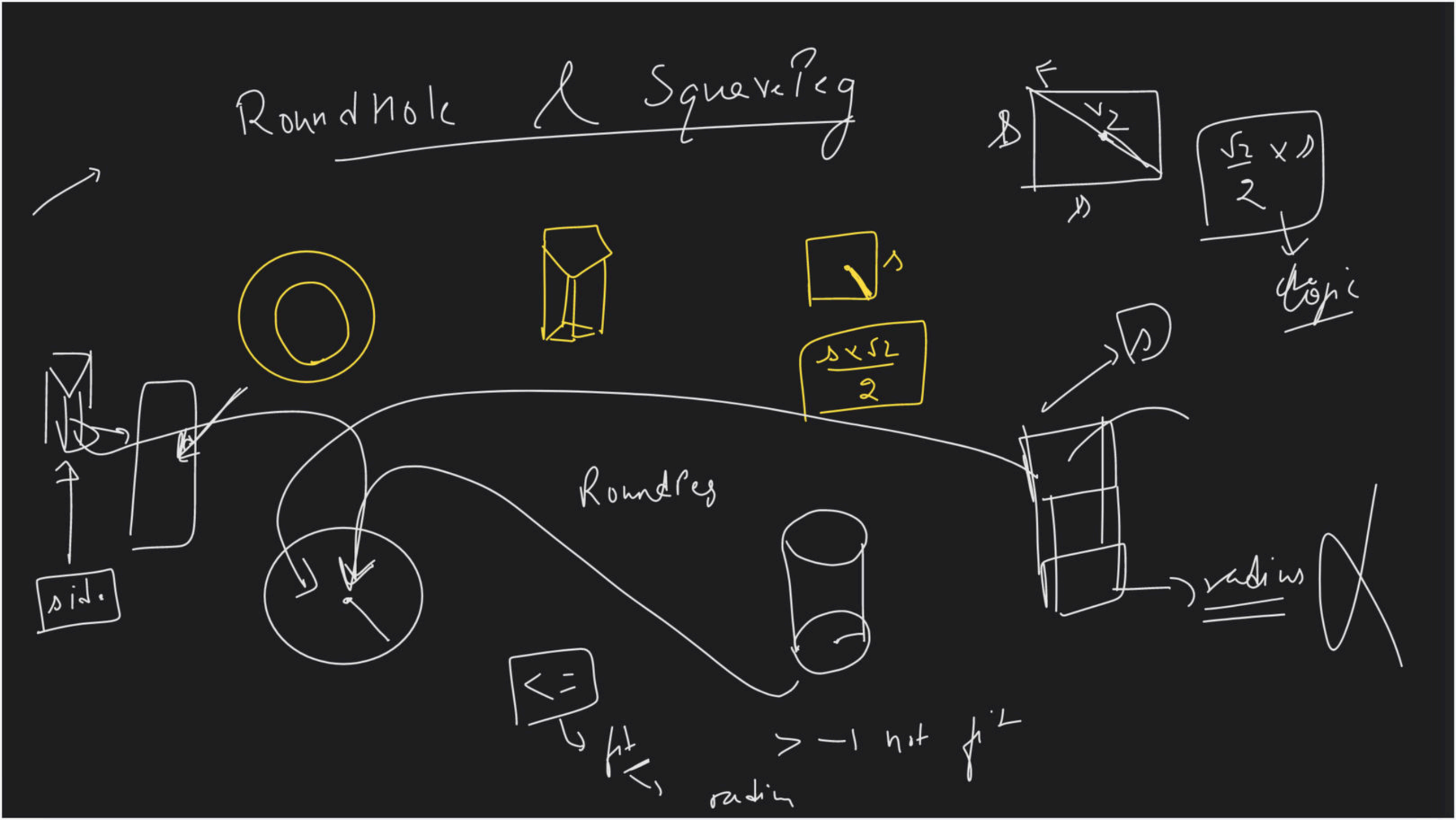


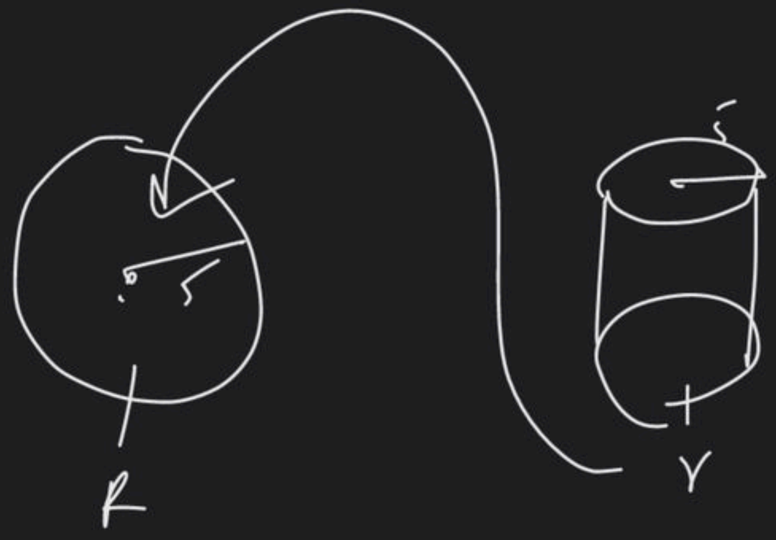
How it works:-

(1)


Client
Target
method ((data))
Adapter
logic-1 (0 words)
delegate
request
Adaptee
method1 ((special data))


Adapter
xml
A
B
Adapter
A
B

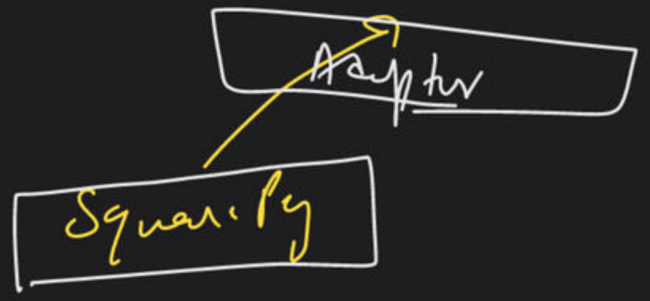

Round Hole & Square Peg
s
s
√2
(√2 × s) / 2
diagonal
s
s × √2 / 2
s
Round Peg
side
radius
<=
fit
>
not fit
radius

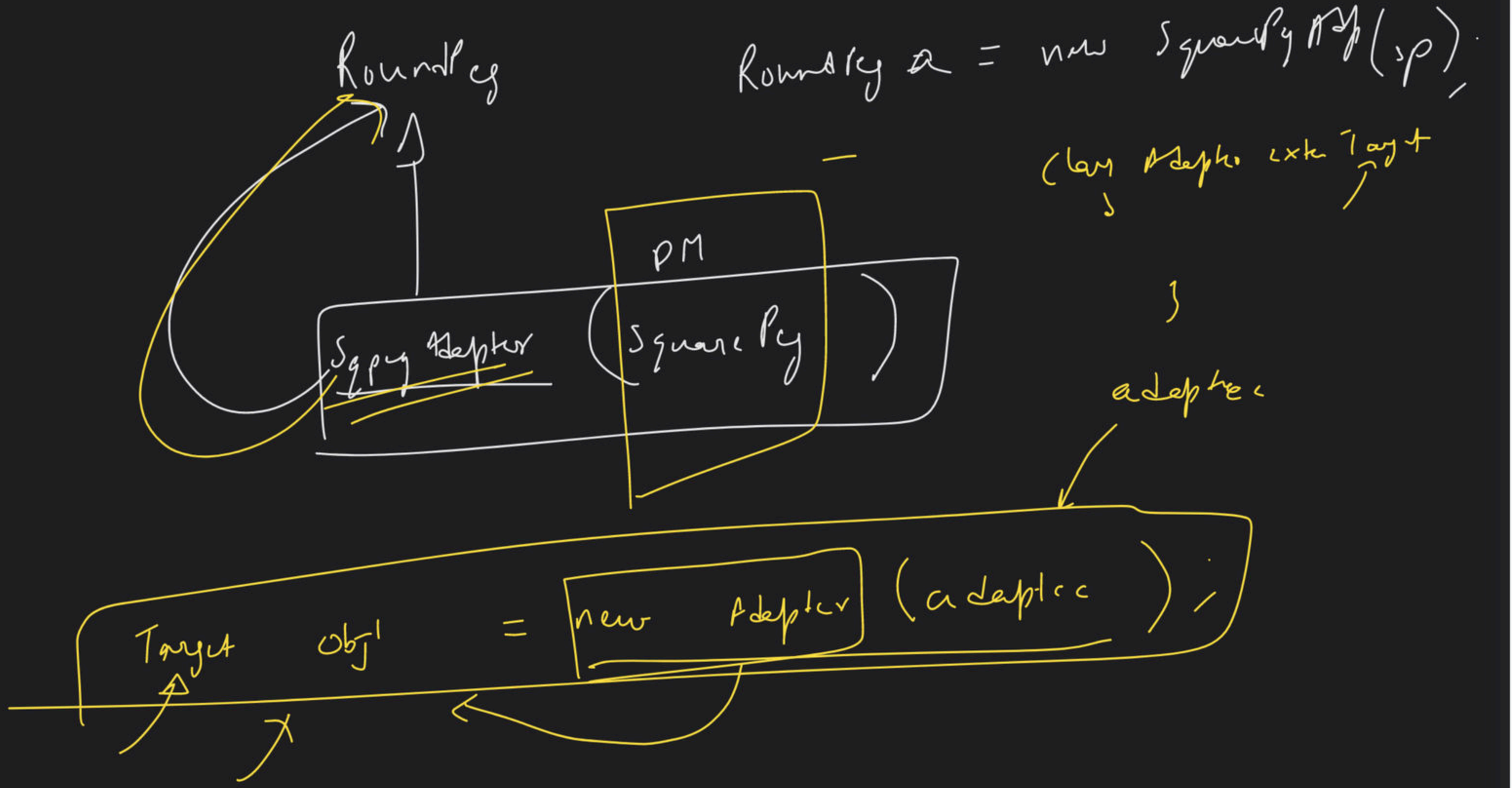


R >= r → fit

r > 2r → True

RoundHole → fits (RoundPeg)




RoundPeg
RoundPeg a = new SquarePegAdapter(sp);
PM
SquarePegAdapter
SquarePeg
(class Adapter extends Target
}
adaptee
Target obj = new Adapter (adaptee);

Ex:

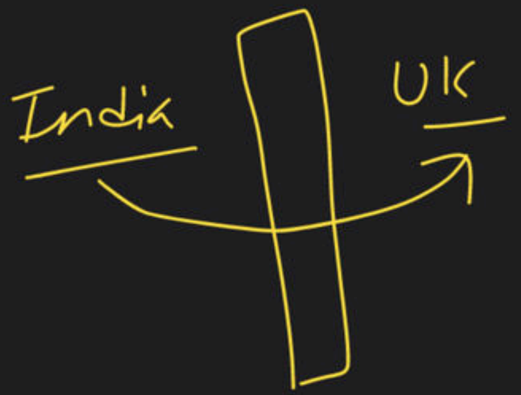
Media Player

↳

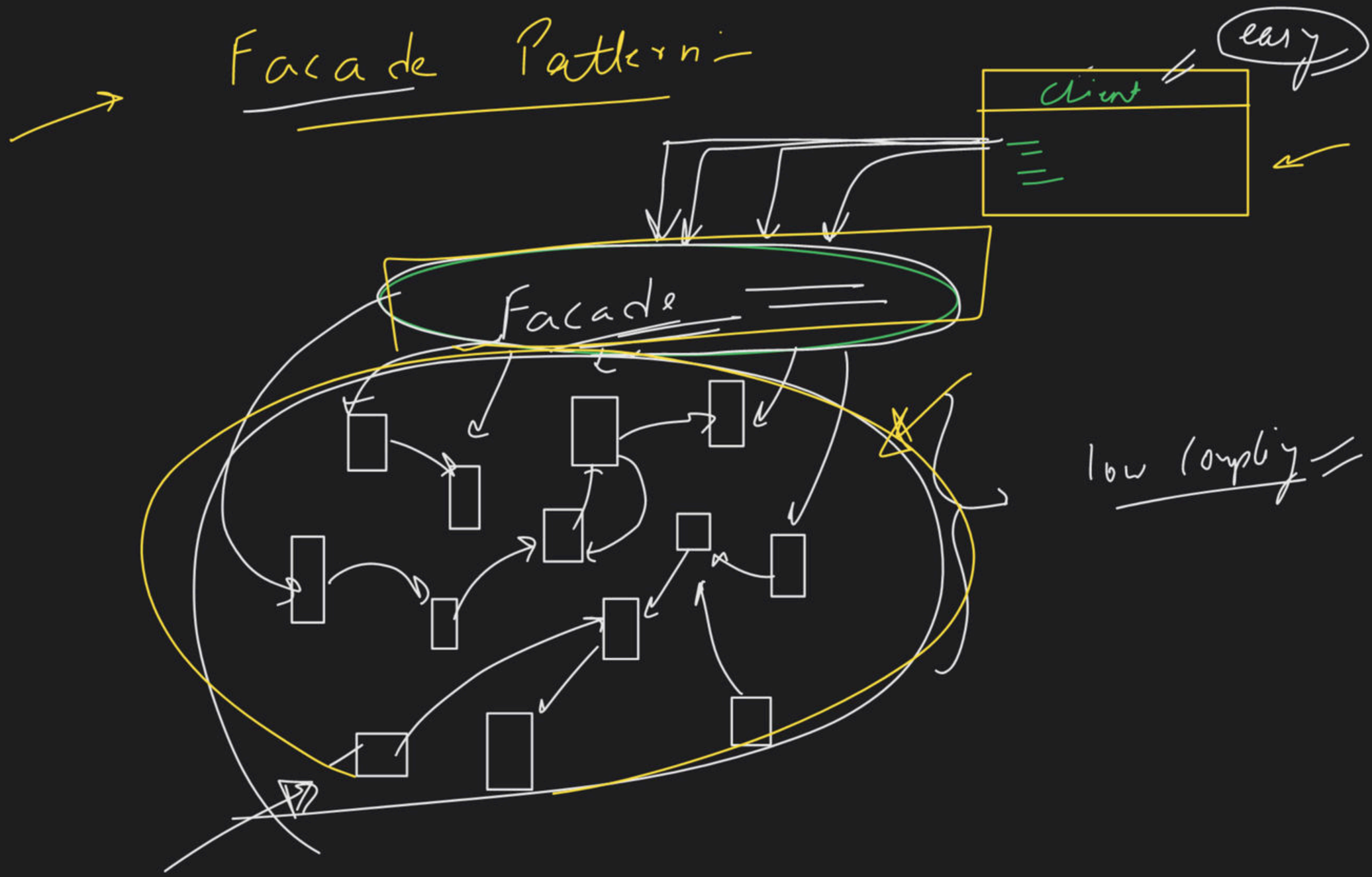
mp3

MyMedia

mp4



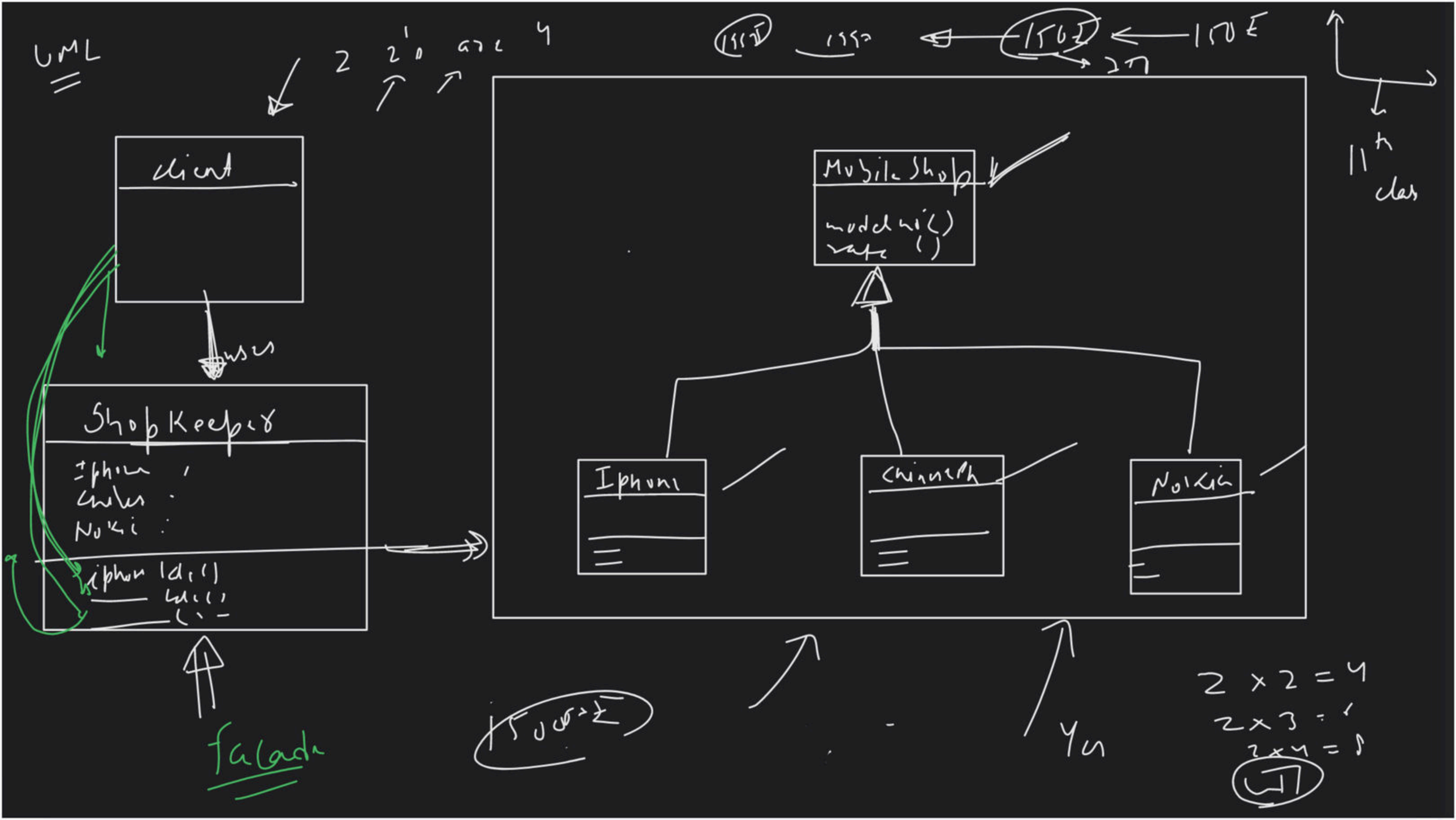
Facade Pattern



→ It provides a unified & simplified interface to a set of interfaces in a sub-system.
∴ it hides the complexity of sub system from client

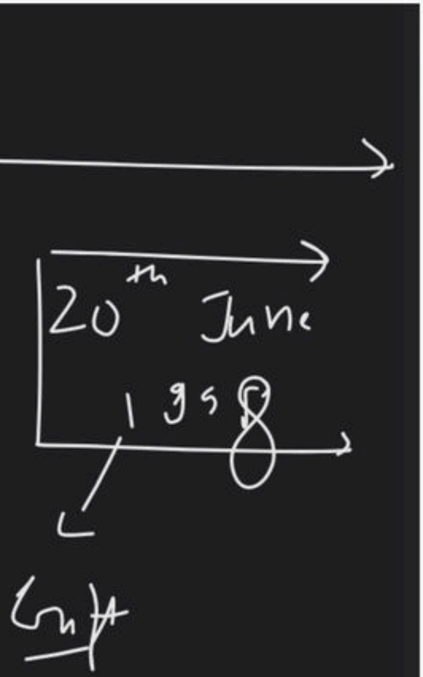
→ in other words, it describes a higher-level interface that make sub-system easy to use.

Adv:-
1. complex hidden from client
2. promotes loose coupling b/w sub-system & client


UML
client
MobileShop
Iphone
Chinese
Nokia
ShopKeeper
Facade

Proxy Pattern → ?

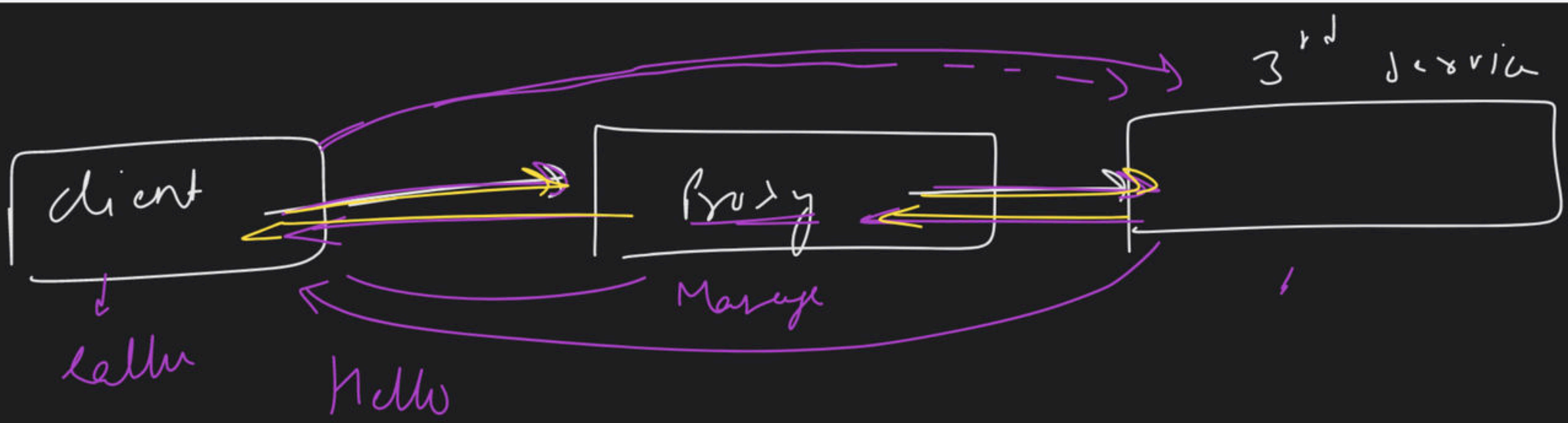
↳ proxy means an object representing other object



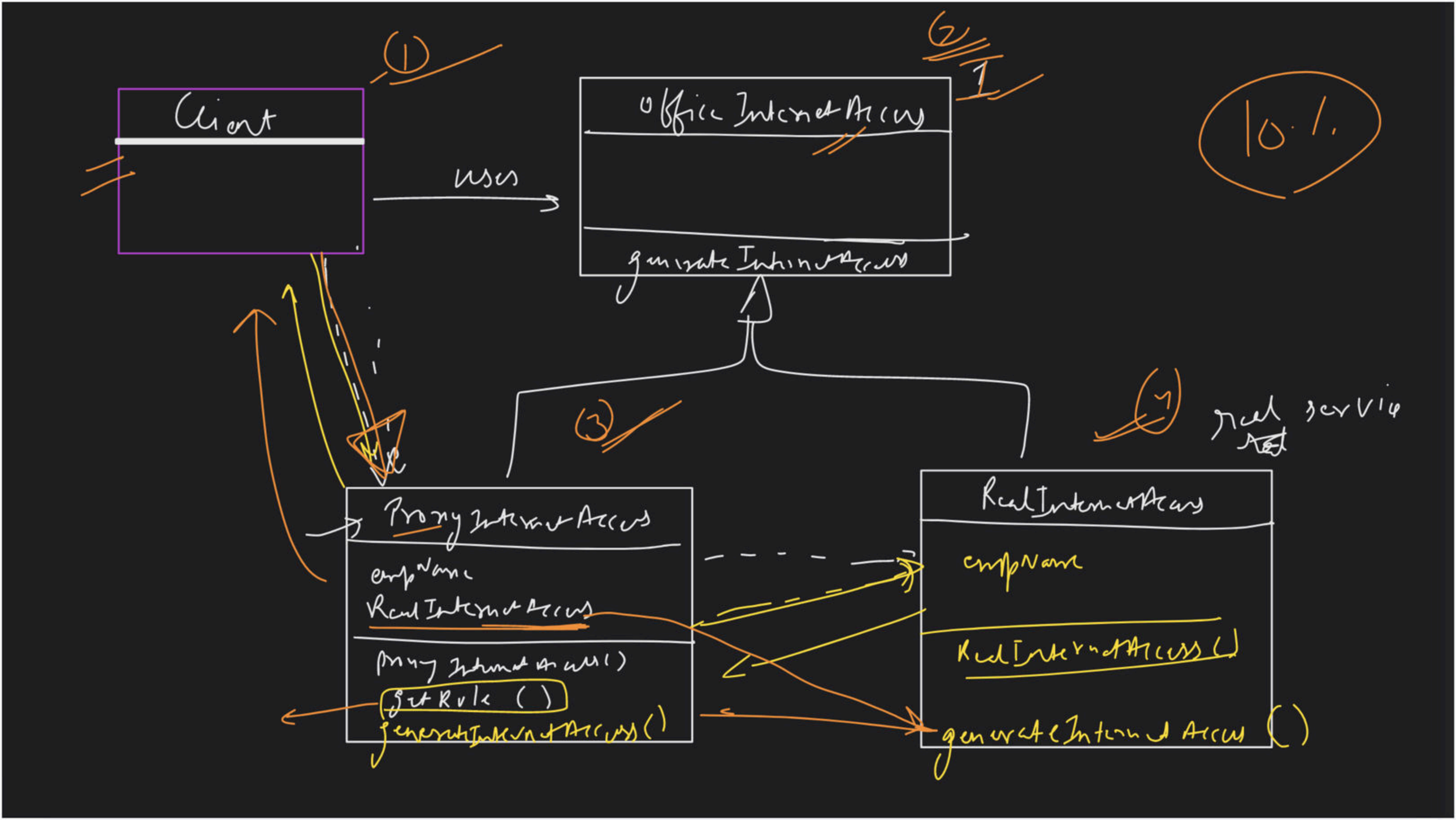
why do we need it ?

↳ protective proxy scenario:- [abc - ✓ xyz - ✗] ← tuition

↳ Smart proxy - [access object → locks]



→ example ?

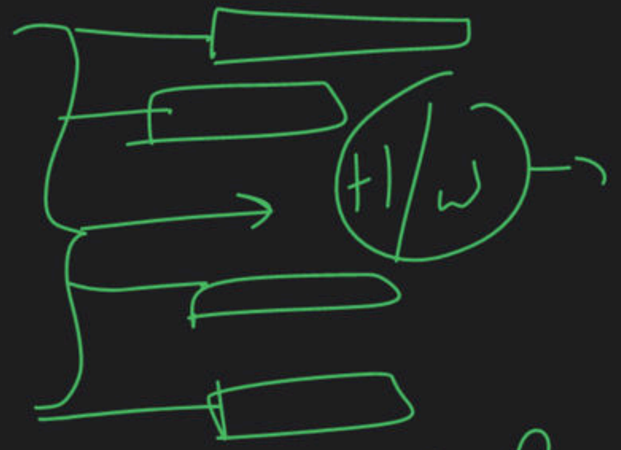

Client
uses
OfficeInternetAccess
generateInternetAccess
ProxyInternetAccess
empName
RealInternetAccess
getRule()
RealInternetAccess
empName
RealInternetAccess()
generateInternetAccess()

→ Adapter

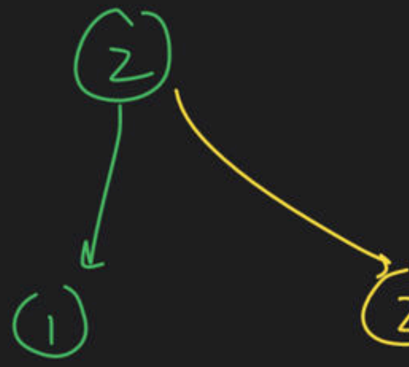
→ Facade

→ Proxy

→ Decorator



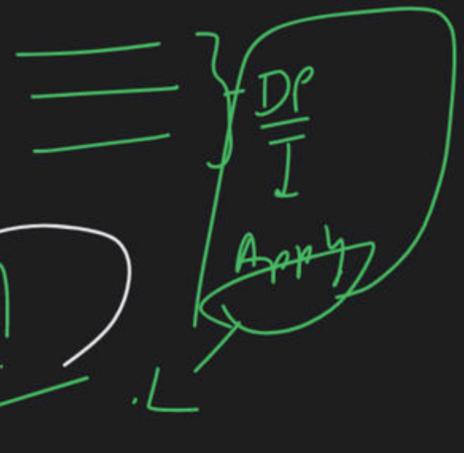
Differ

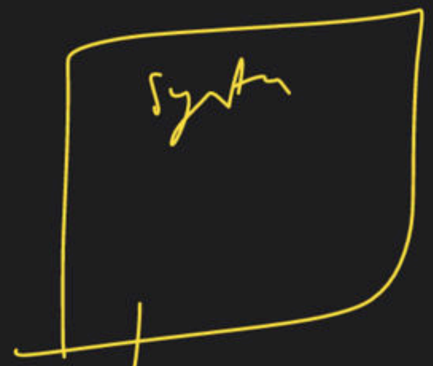


confusion

DP

argument





DP

DP →

Design Principles

↳ Single Resp

↳ Func Comp

↳ Hy –

Why ?

→

extra layers

Abstraction

Why ?

Know

Optimal

→ Behavioural Design Patterns

↳ it says:

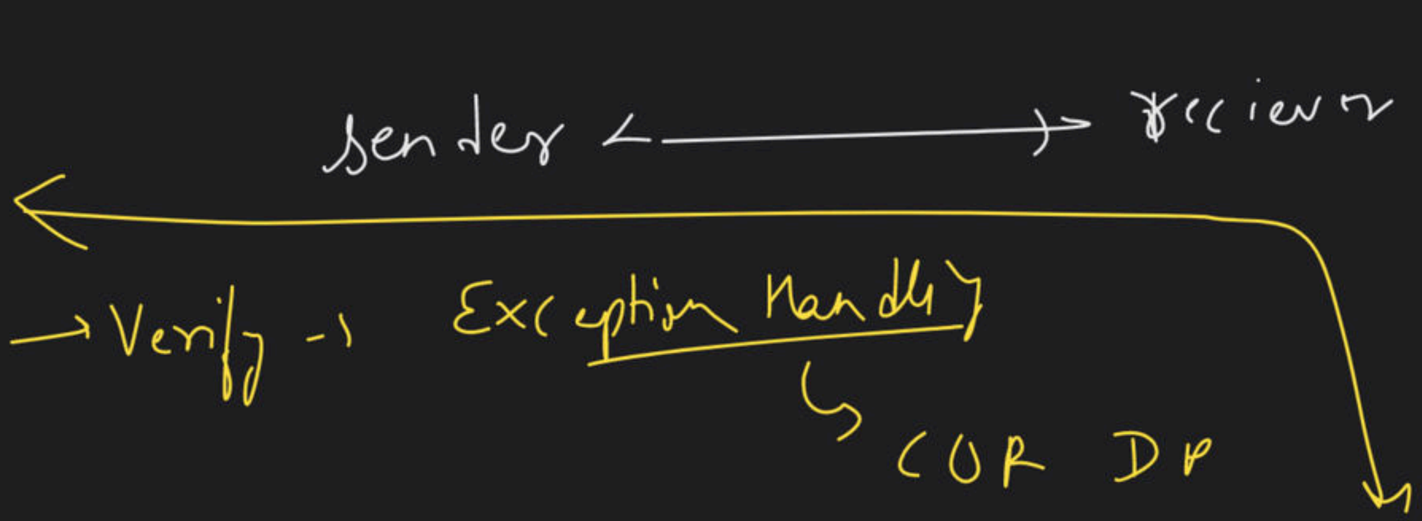
interaction b/w the objects should be in such a way that they can easily talk to each other & still be loosely coupled.

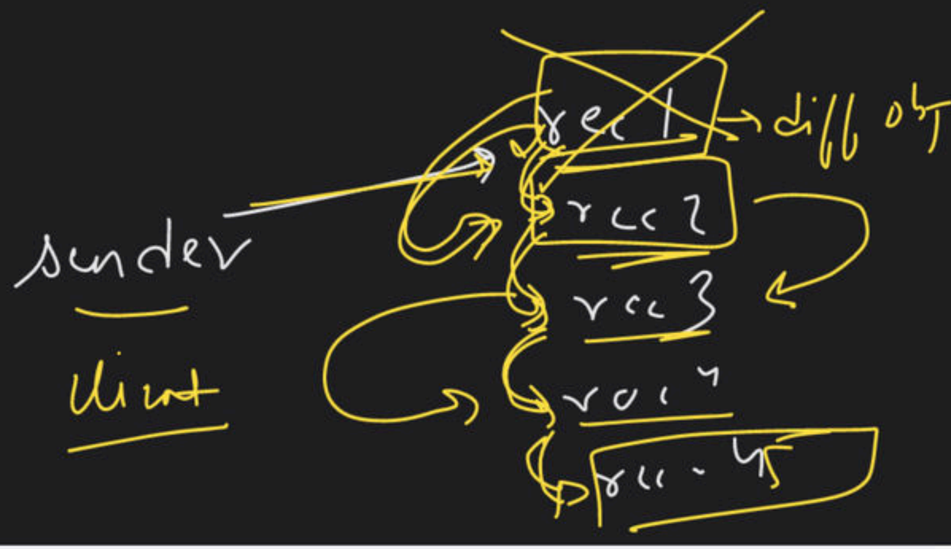
Types → 12 ← COR

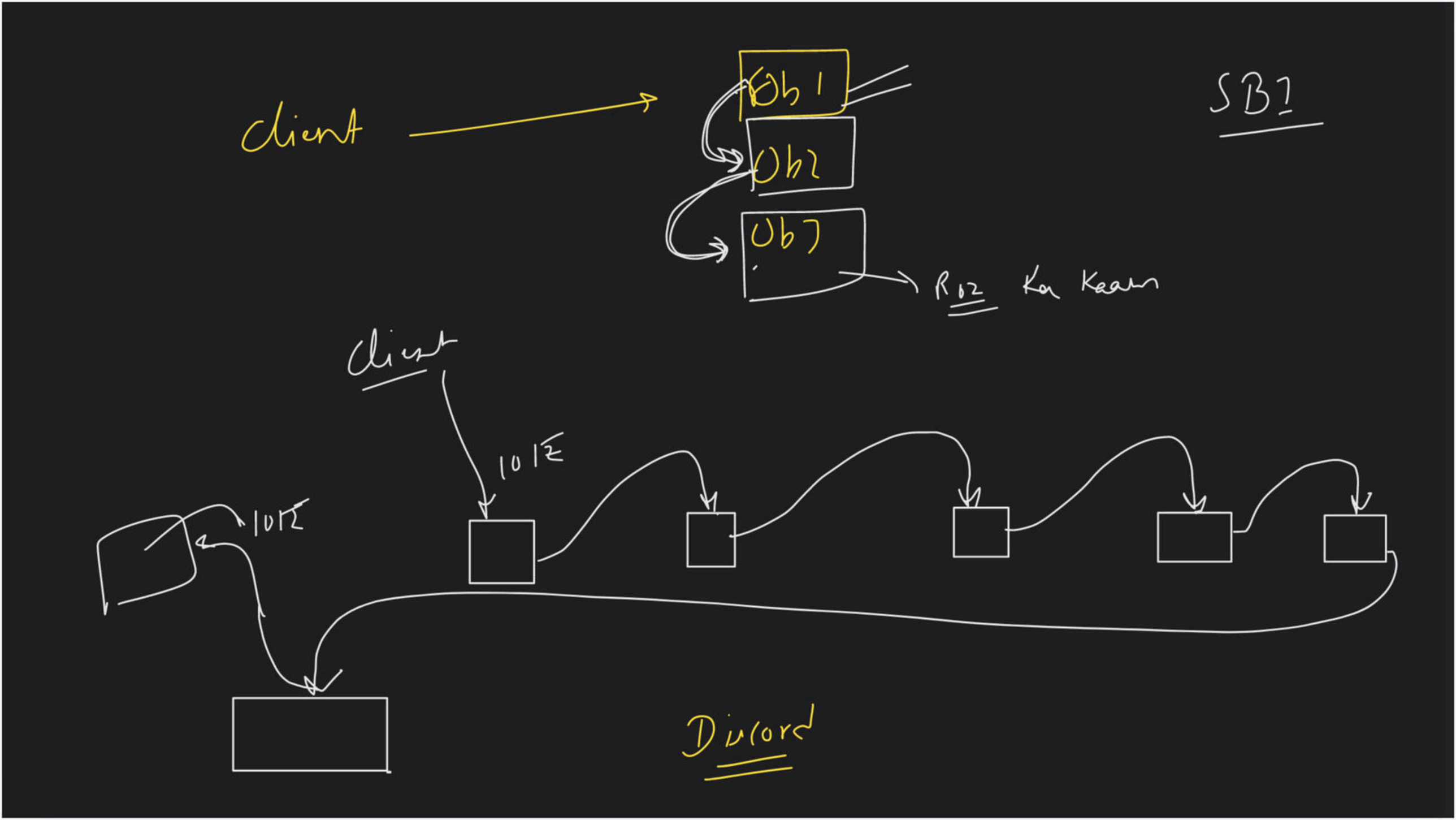
→ Chain of Responsibility Design Pattern (7th)

↳ "Avoid coupling the sender of a request to its reciever, by giving multiple objects a chance to handle requests."

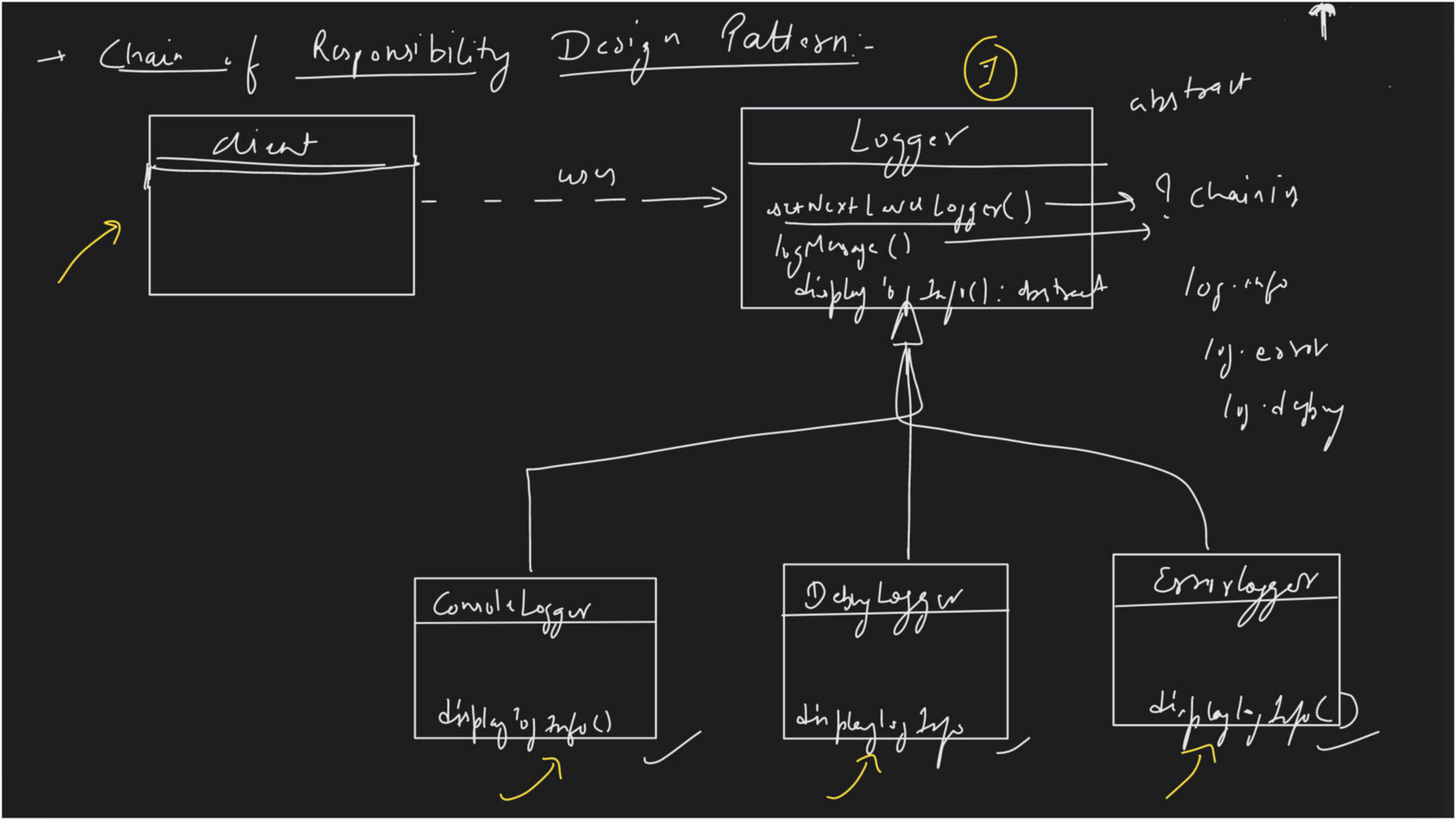
Broadcast comm↳



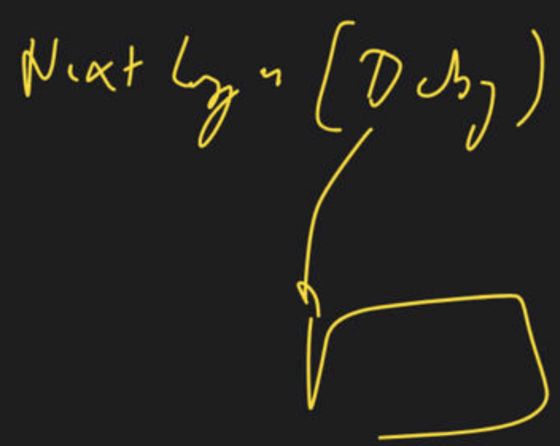



client
Obj
Obj
Obj
SB1
Req Ka Kaam
client
Req
Req
Discord

→ Chain of Responsibility Design Pattern:-



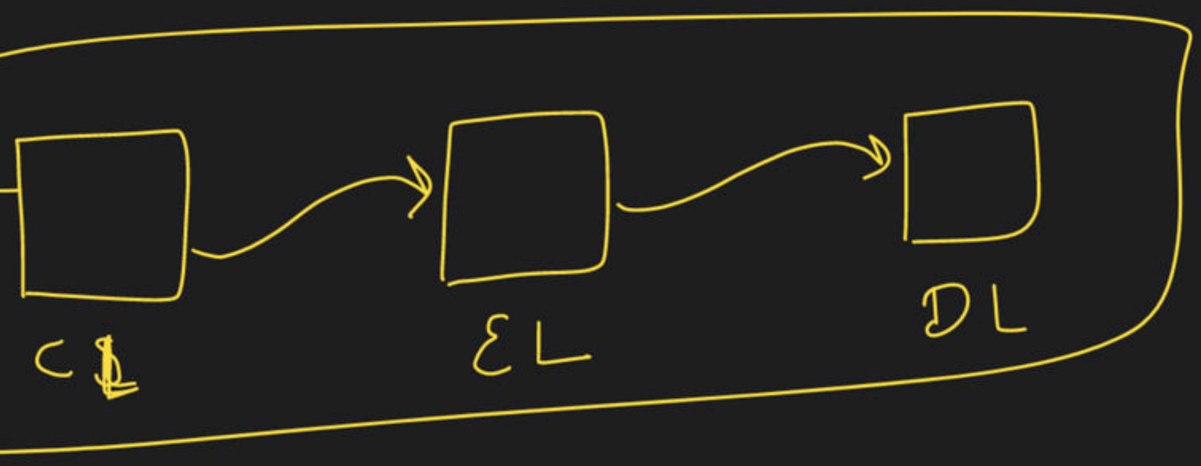
Console logger



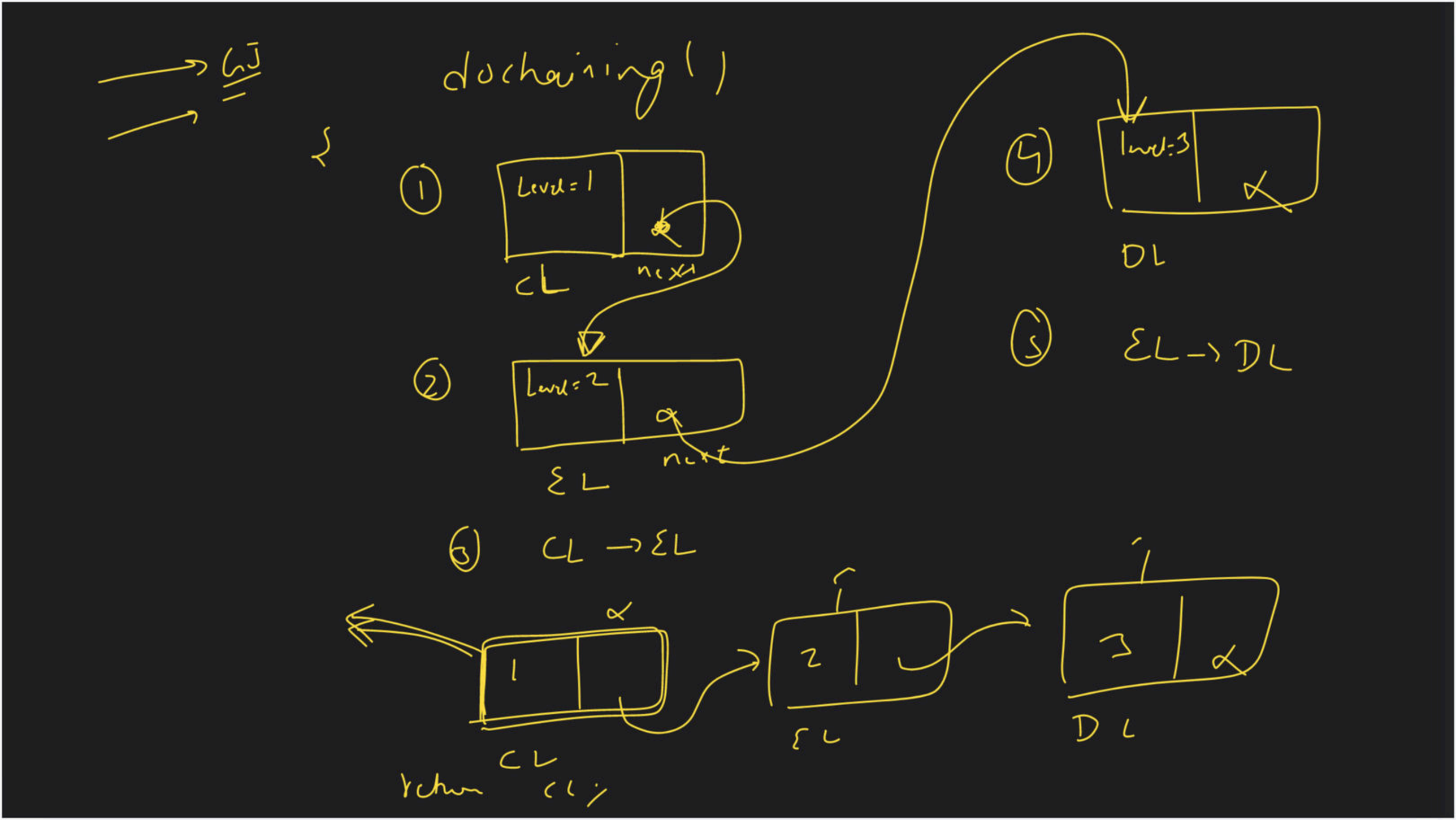
doChaining() → ?

{

return



}


docheining()
{
1
Level=1
next
CL
2
Level=2
next
EL
4
level=3
DL
5
EL → DL
6
CL → EL
CL
EL
DL
return cl;