OOPs Design Principles

Course on OOPs Design Fundamentals

Love Babbar • Lesson 1 • May 22, 2022

→ Clean code → ? → [easily readable by other developers]

→ Why ? → =

→ How → ?

by following a set of guidelines

↑ Design Principles

How → Clean code

→ meaningful variable names → function → class etc

→ Comments

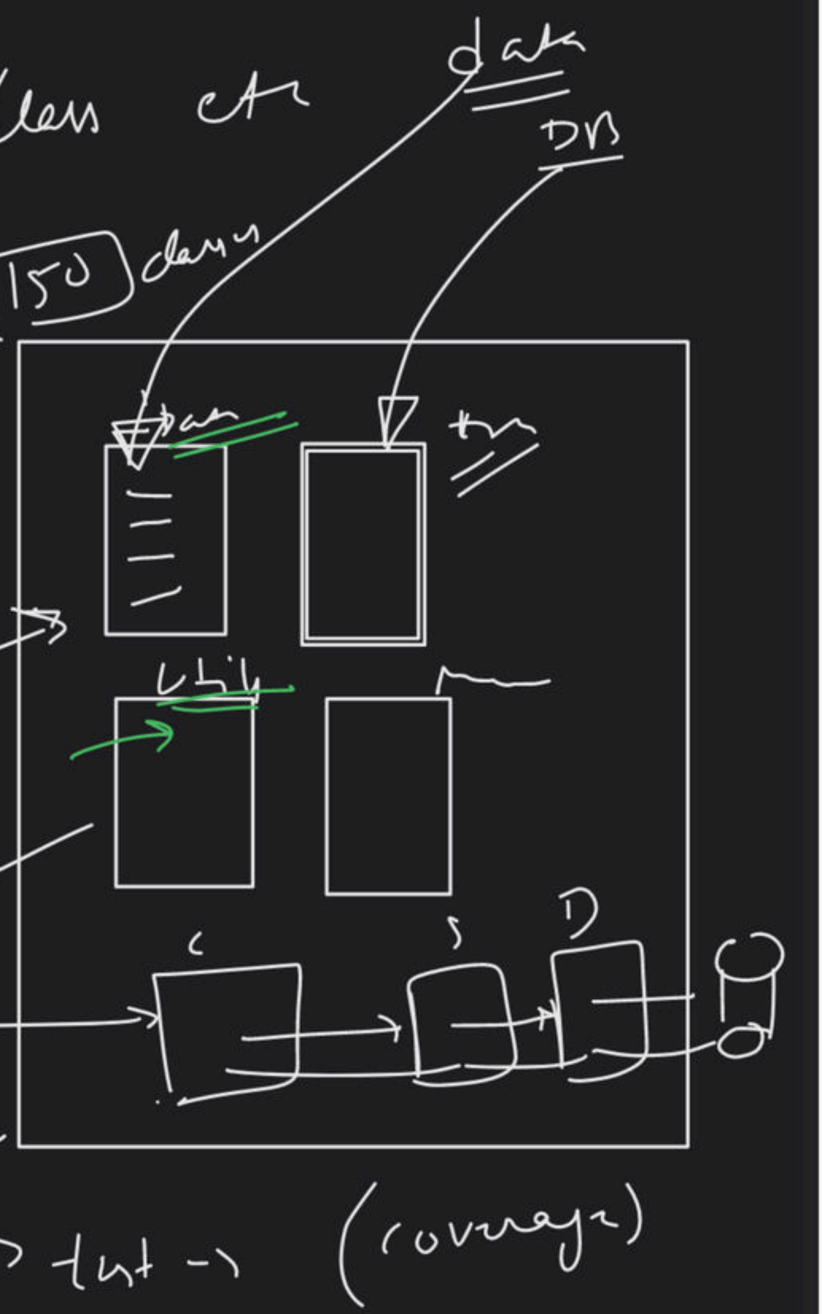
→ Modular → what?

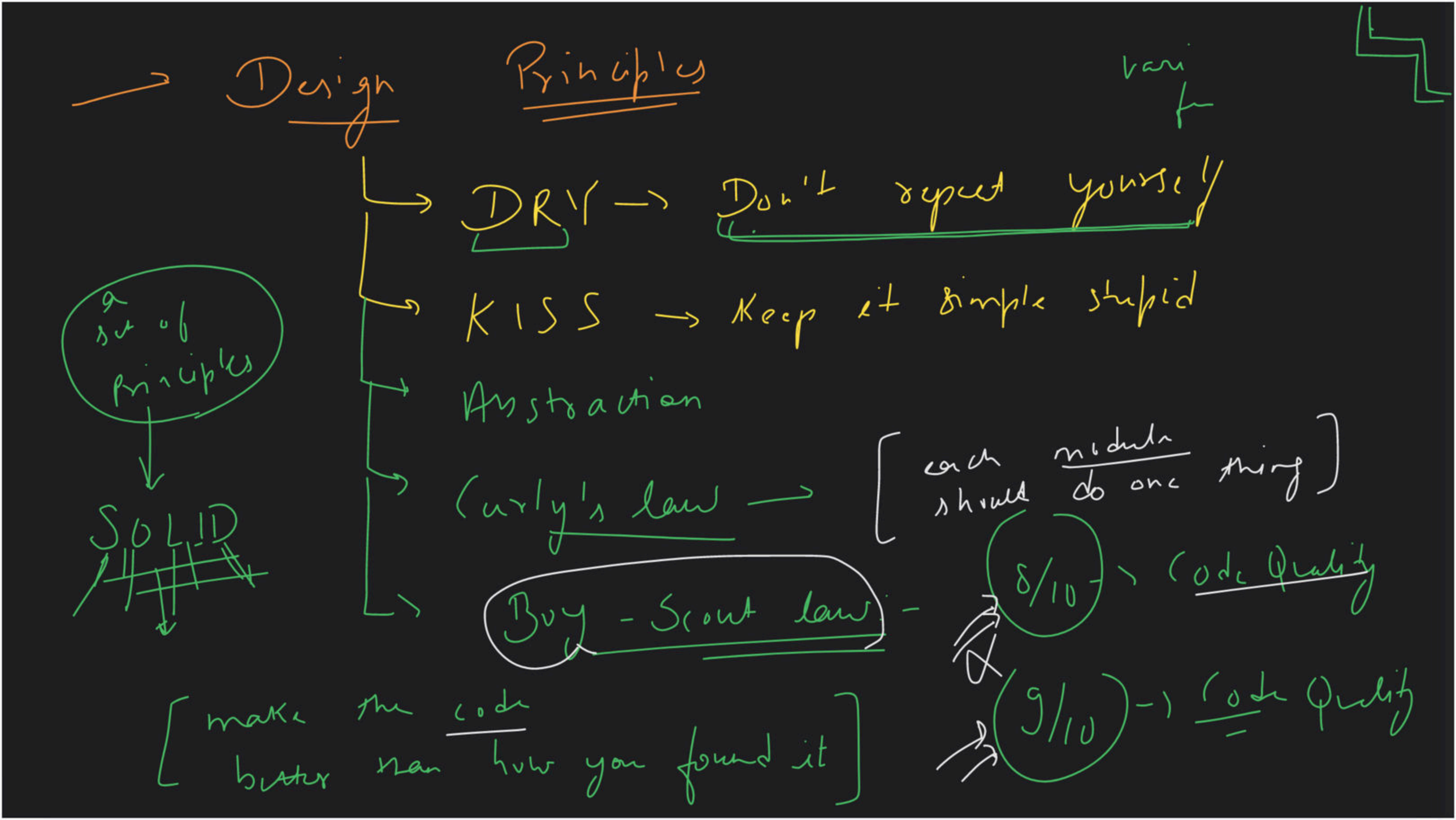
→ functions → size LOC

→ single objective

→ → short

→ structure → readable → simple



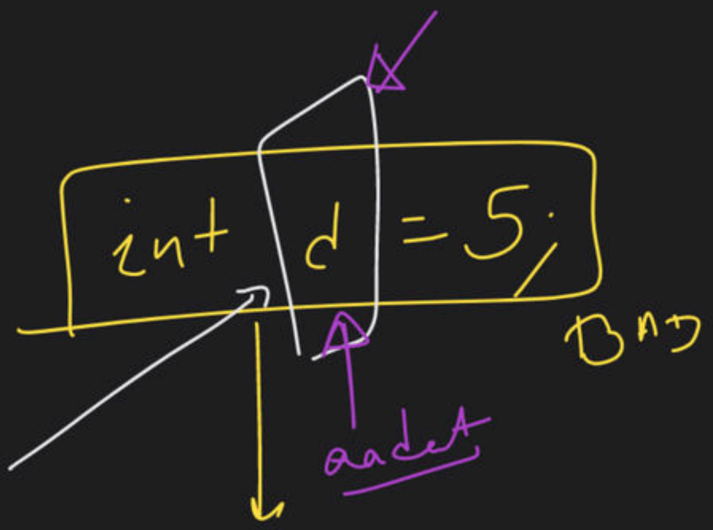

Design Principles
vari
DRY → Don't repeat yourself
KISS → Keep it simple stupid
Abstraction
Curly's law →
[each module should do one thing]
Boy - Scout law :-
[make the code better than how you found it]
a set of principles
SOLID
8/10 → Code Quality
9/10 → Code Quality

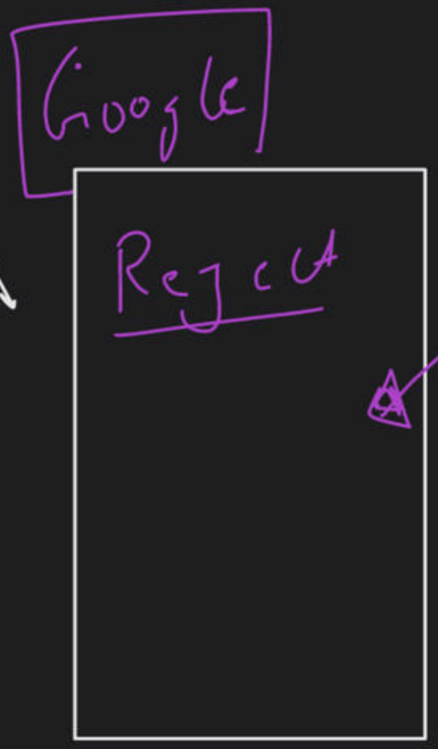
→ Naming → variables, func

for(int i=0; i<n; i++)

→ intention - revealing names

descriptive name





int getAnswer();

clear

int getSum();

clarity

function → meaningful intent-revealing description → getter/setter

"get" / "set"

getSum / setDays

Verb ?

↳ pre-defined → sort( )

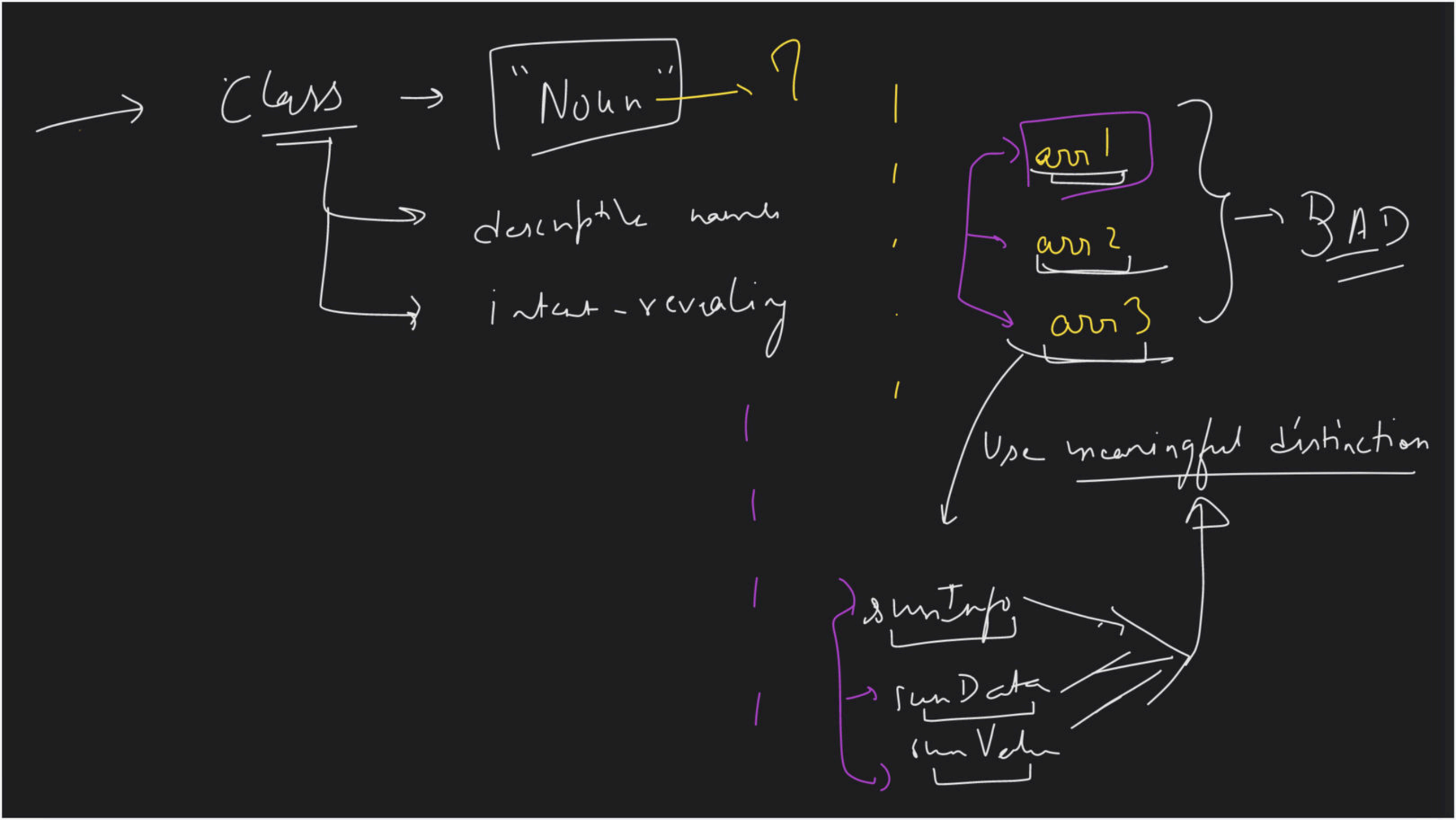
reverse( )

swap ( )

R. C Martin

Ward's principle →

Babbar Research


Class
"Noun"
?
descriptive names
intent-revealing
arr 1
arr 2
arr 3
BAD
Use meaningful distinction
sumInfo
sumData
sumValue

Good function -> ?

Gayandeep

proper name
- verb
- description
- intent-revealing

short → 20 LOC

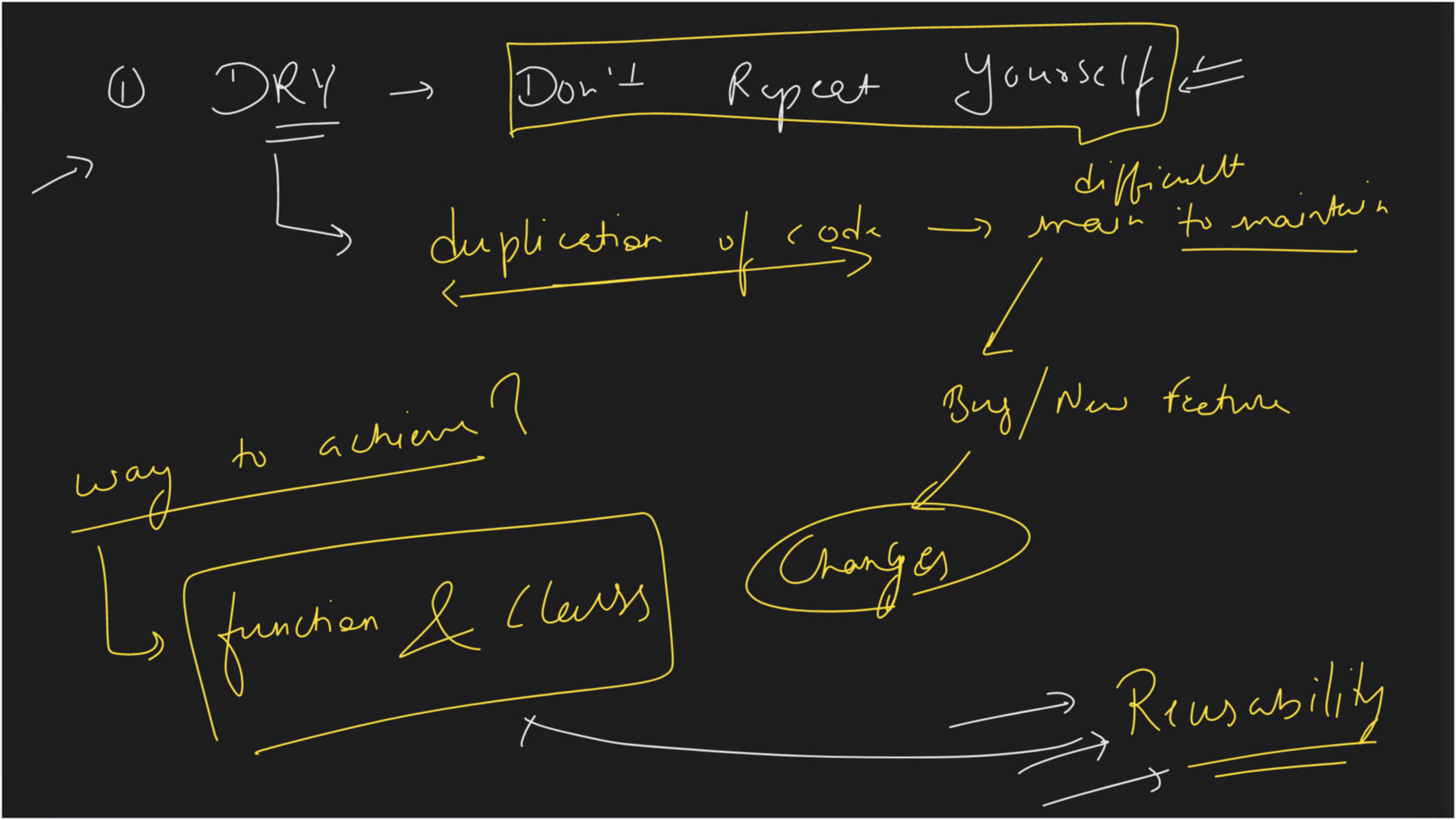
dedicated to single task

should have fewer arguments

check → P/W

Helper function

id -> 0, 1, 2, 3


① DRY → Don't Repeat Yourself
duplication of code → difficult main to maintain
Bug / New feature
Changes
way to achieve ?
function & class
Reusability

Observation :- DRY

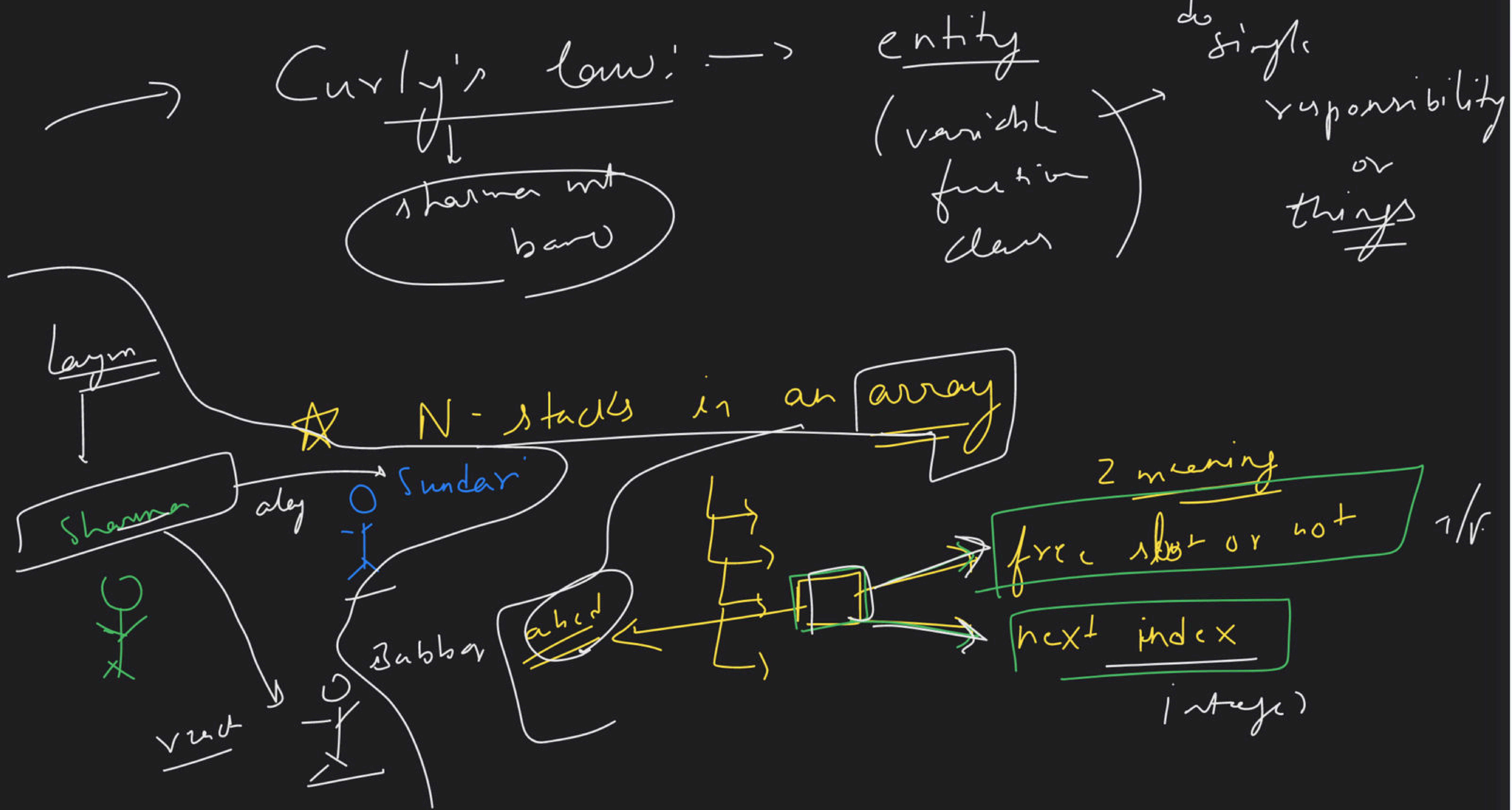
college() → row (11+_Madras);

college1() → Said ( );

college2() → row ( );

Law

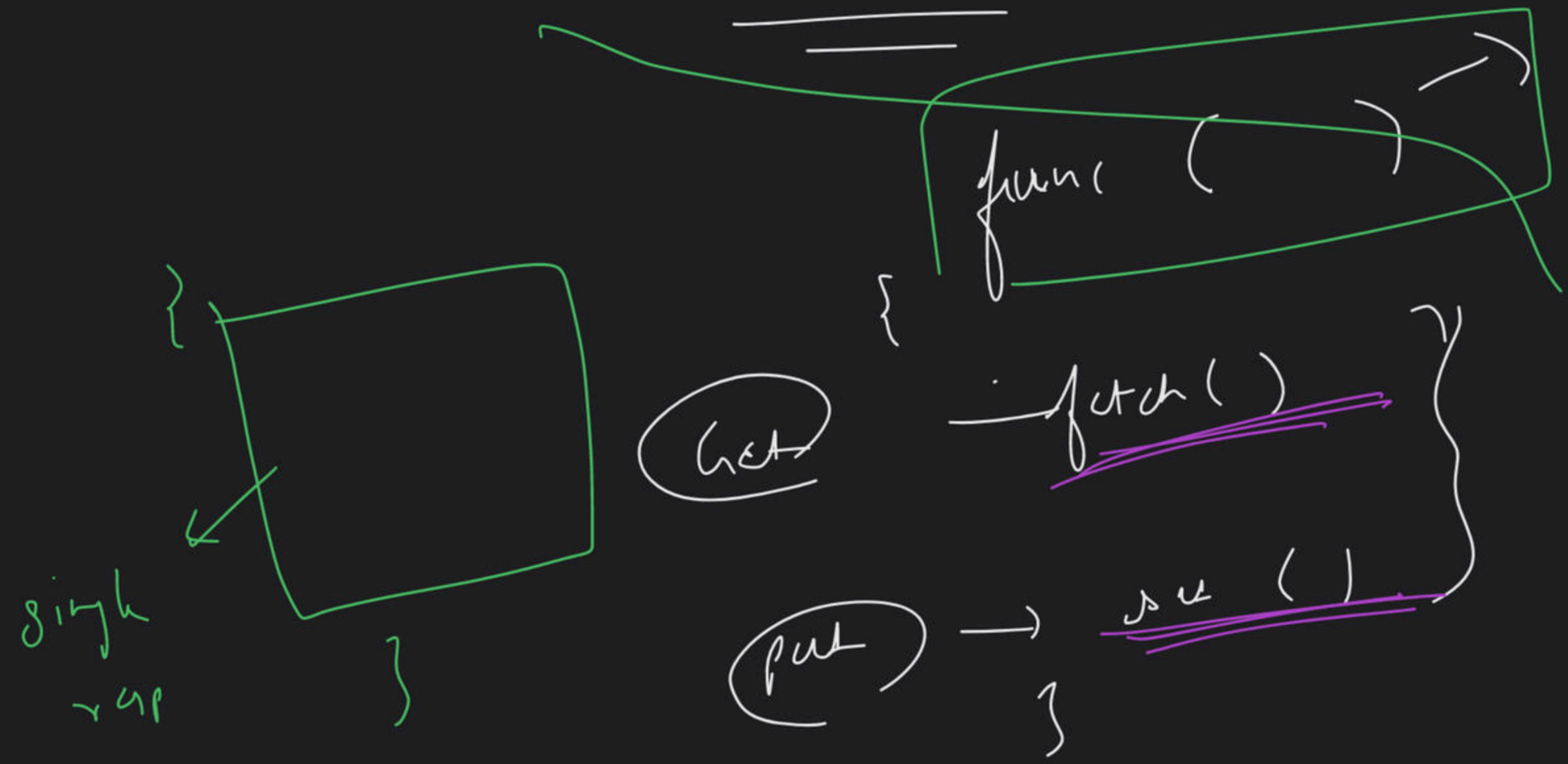
Boy-Scout Law



Curly's law:
entity
(variable
function
class)
do single
responsibility
or
things
Layers
N-stacks in an array
Sundari
Sharma
Babbar
abcd
2 meaning
free slot or not
next index
T/F
(integer)

→ Curly's law

↳ technical



→ KISS :- Keep it simple Stupid

what

R.C Martin

	a	b	c	d
Any	within p	—	—	—
Private	Yes	No	No	No
Default	Y	Y	No	No
Protected	Y	Y	Y	No
Public	Y	Y	Y	Y

explore

Easy default

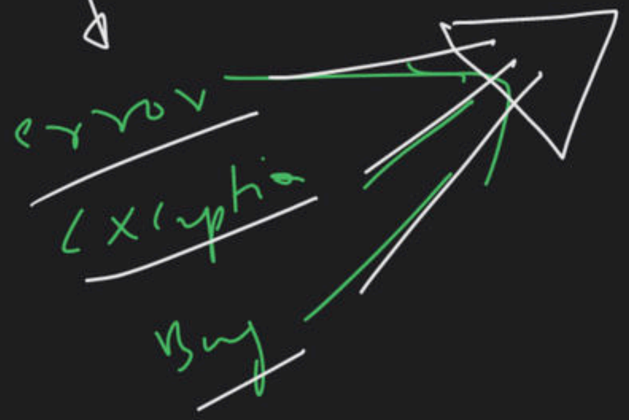
code

```
if (a)
    if (b)
        if (c)
            if (d)
                cout << "Public";
```

→ Benefits :-

Software Testing

H/w



what

Bugs

→ easy to read / understand

→ less time to code ← → delete

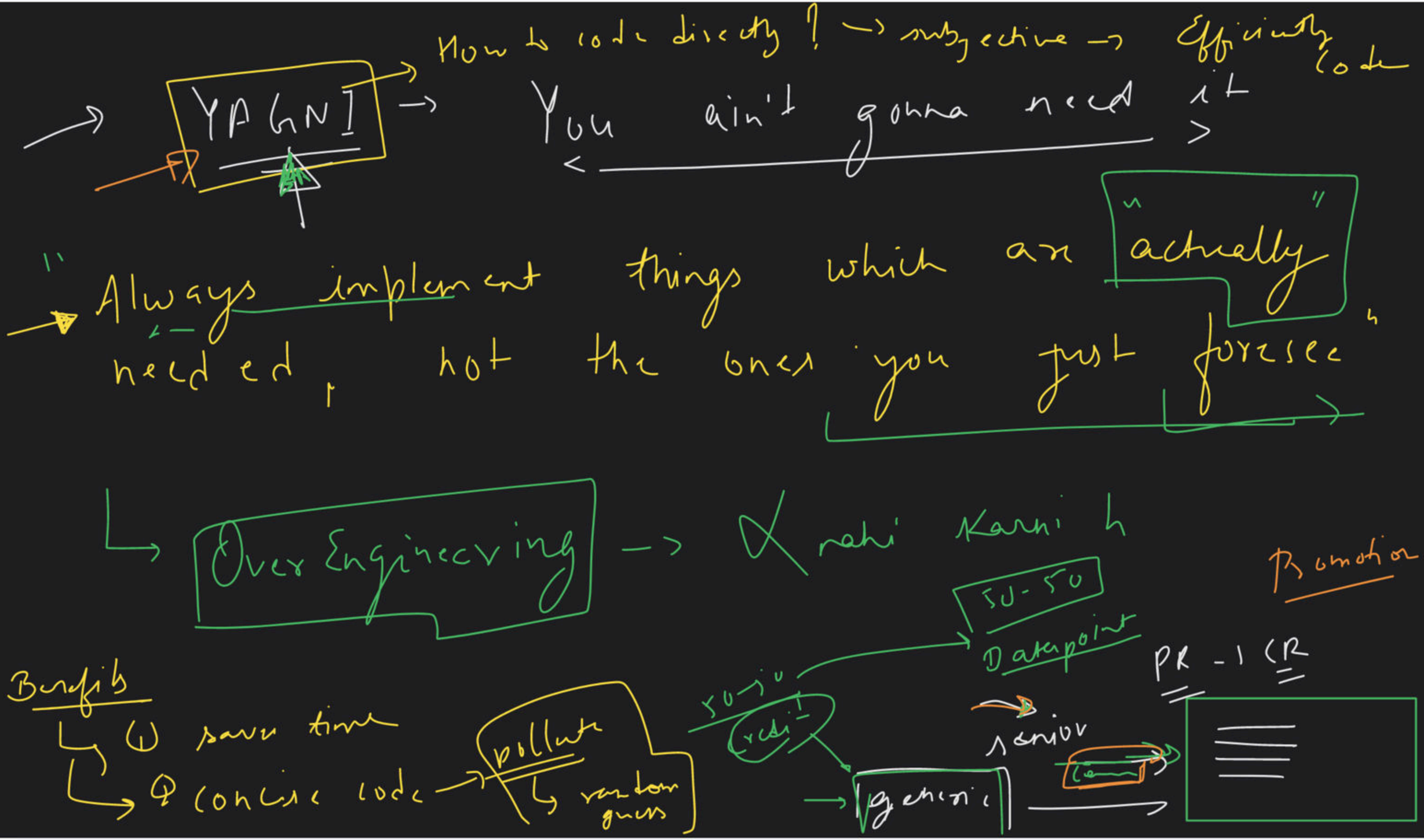
→ chances are less

debug / modify / update / protect

Unexpected behaviour

Why Bug was named Bug ?

HYT_short

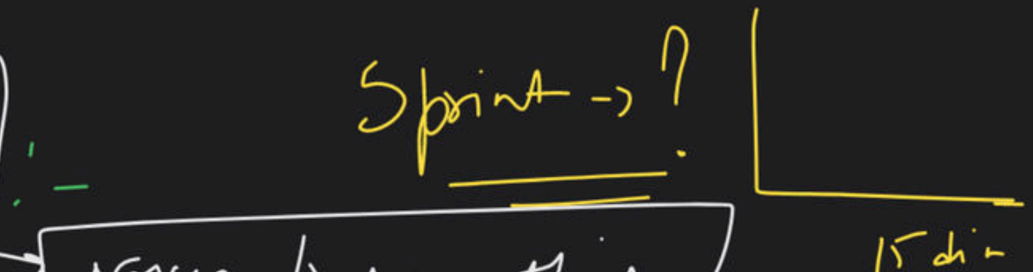

YAGNI
How to code directly ? → subjective → Efficiently (code
You ain't gonna need it
"Always implement things which are actually needed", not the ones you just foresee"
Over Engineering → Karni h
Benefits
① save time
② concise code → pollute
random guess
su-su
redis
generic
su-su
Datapoint
Promotion
PR - 1 CR
senior

Pre-Optimisation is the root of all Evils

Evil

Boy - Scout Law :-

Sprint -> ?



saves from this

→ The Code Quality tends to degrade with each change.

generally

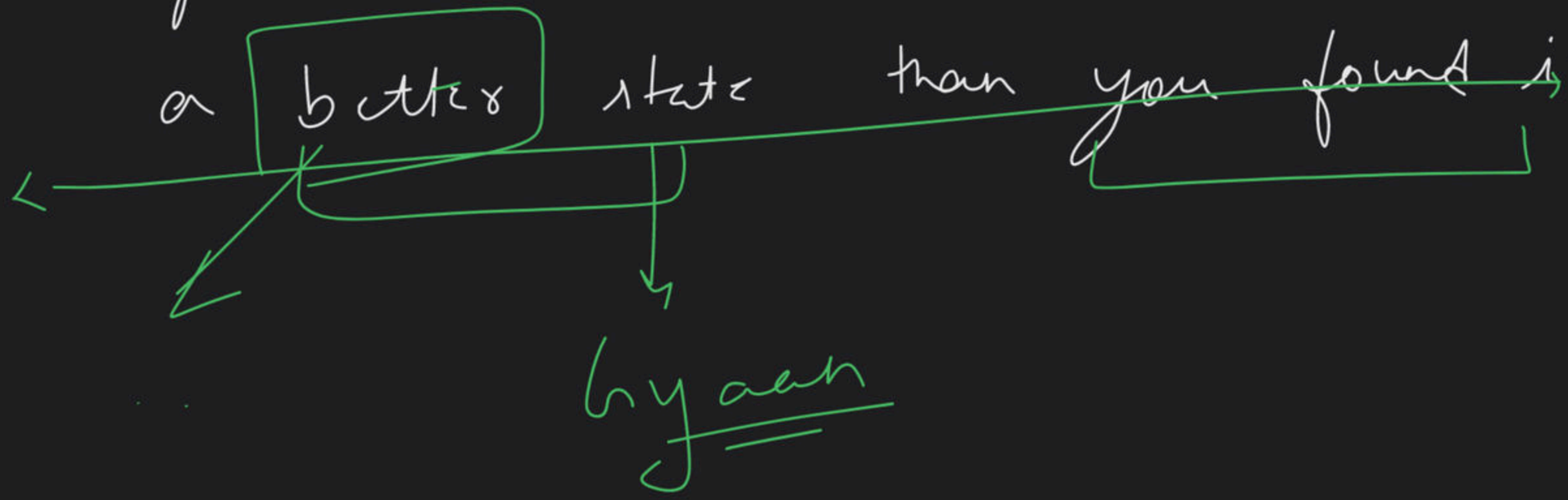
degrade → tech debt → generate

?

feature → API integration

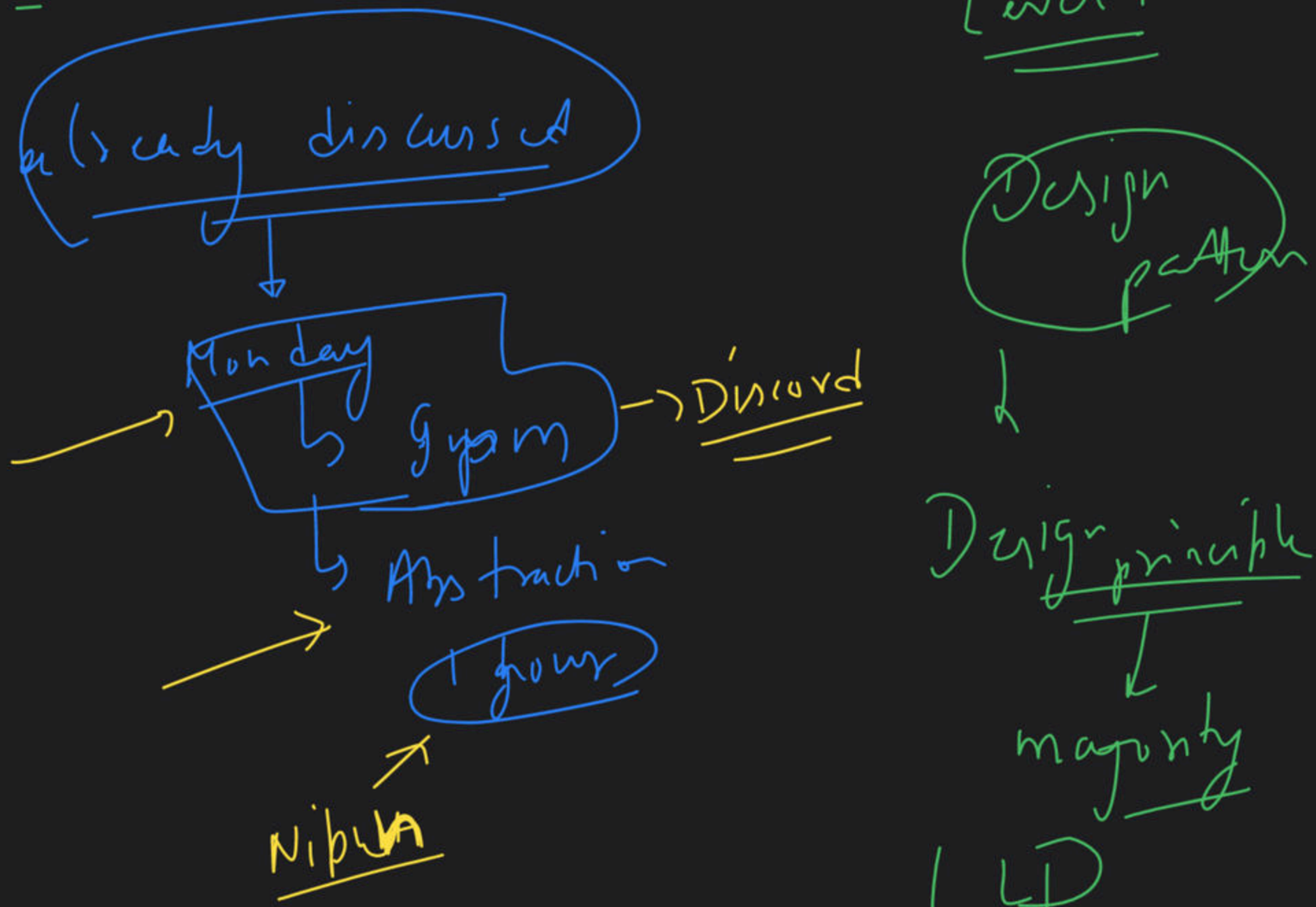
JUnit → tech debt → Backlog

→ "Always leave the code behind in a better state than you found it"



Gyaan

① Abstraction :-



Level 1

Design pattern

Design principle

majority

LLD

or

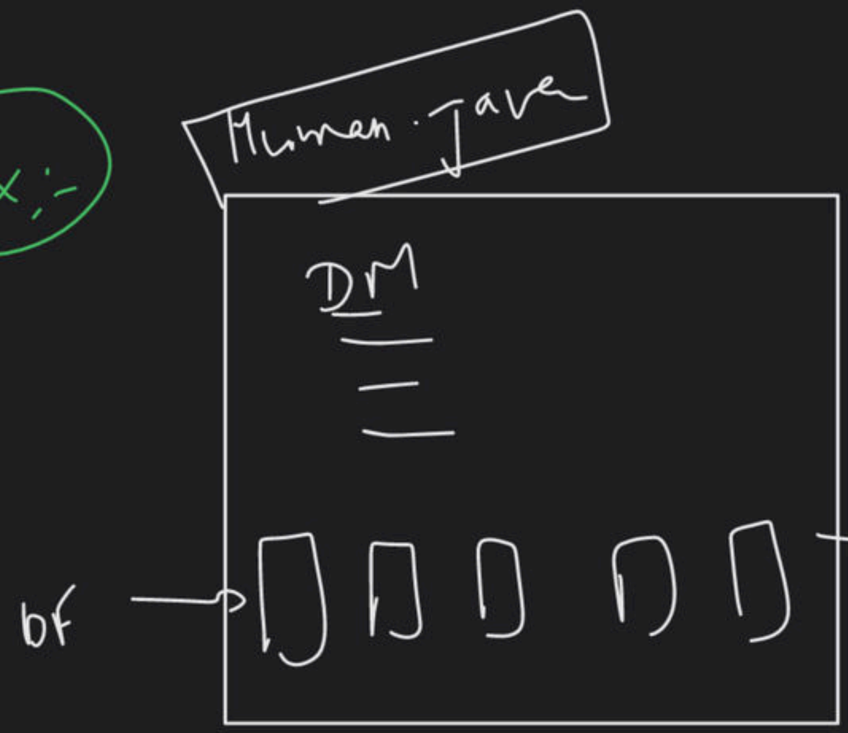
HLD

(2) Cohesion & Coupling - ?

Imp

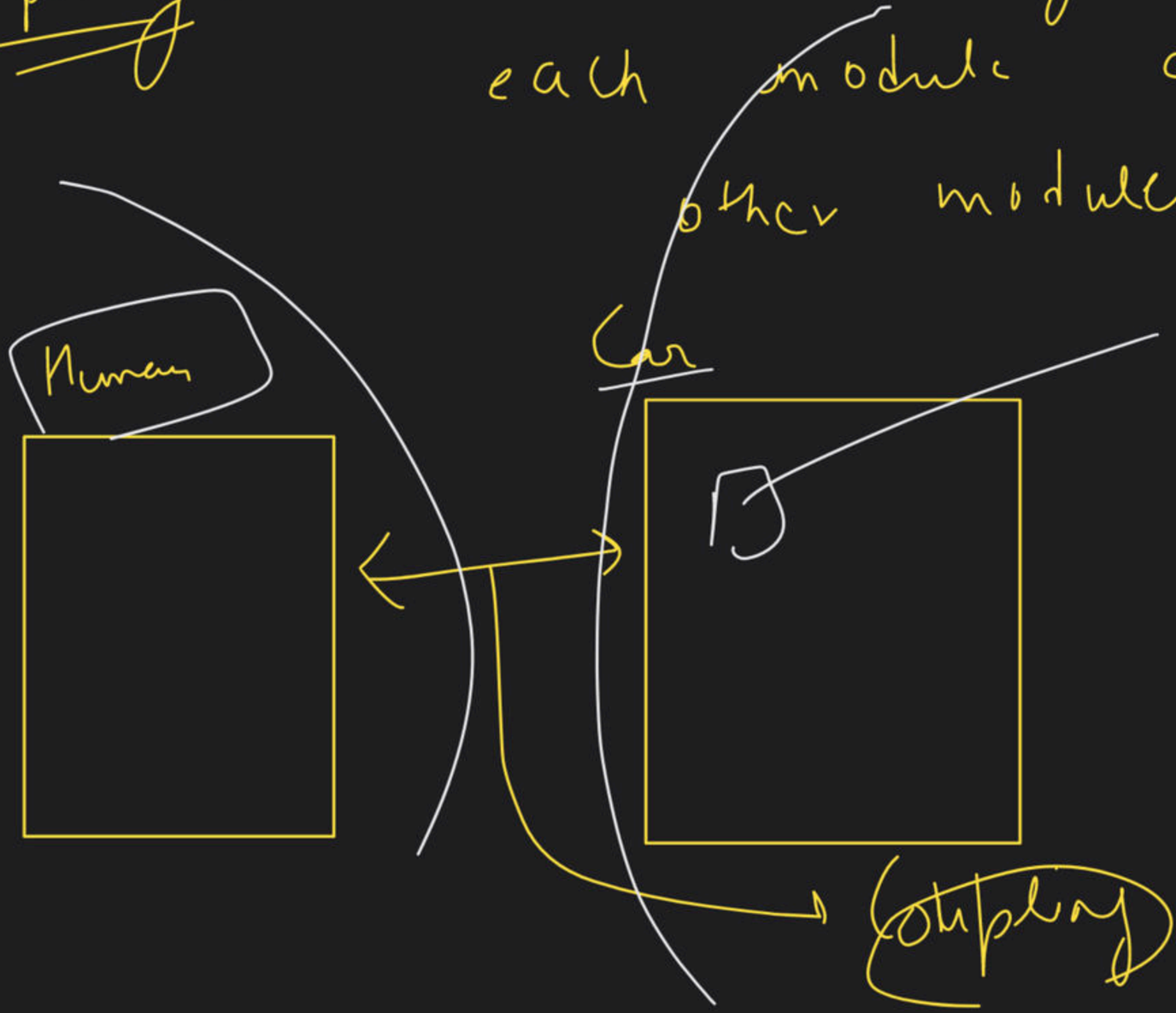
↳ **Cohesion:-** It is the degree of how strongly related & focused are the various responsibilities of a module.

Ex:-



Required → Maximum Cohesion

→ Coupling :- It is the degree to which each module depend on other modules



Required

↓

Low coupling

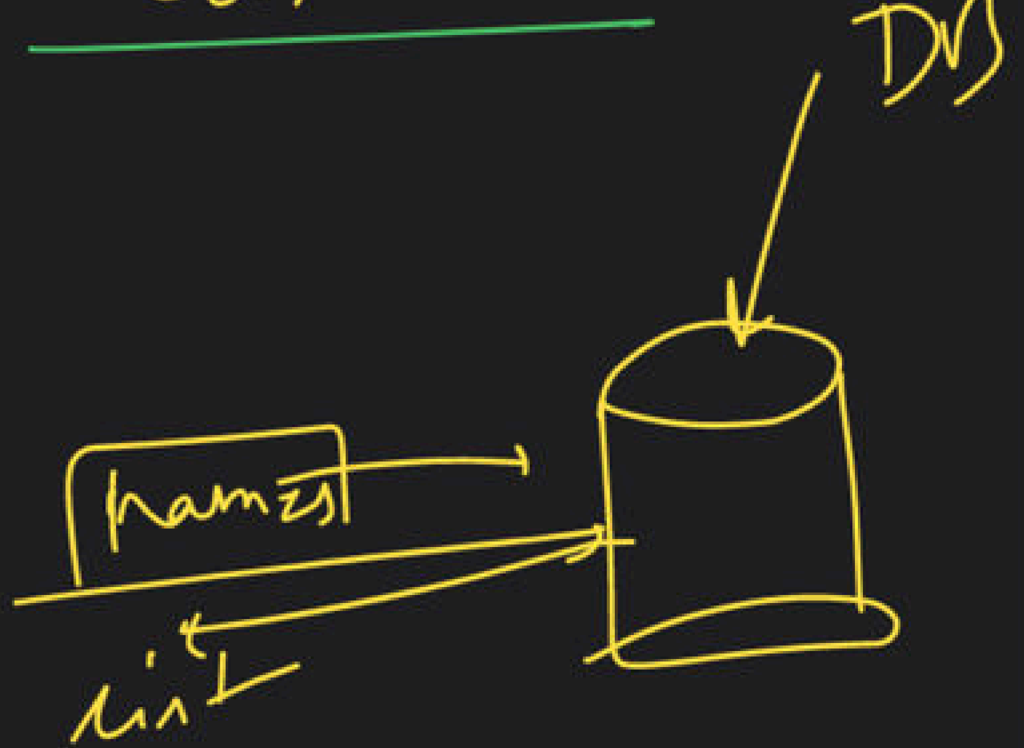
Command - Query Seperation: (CQS)

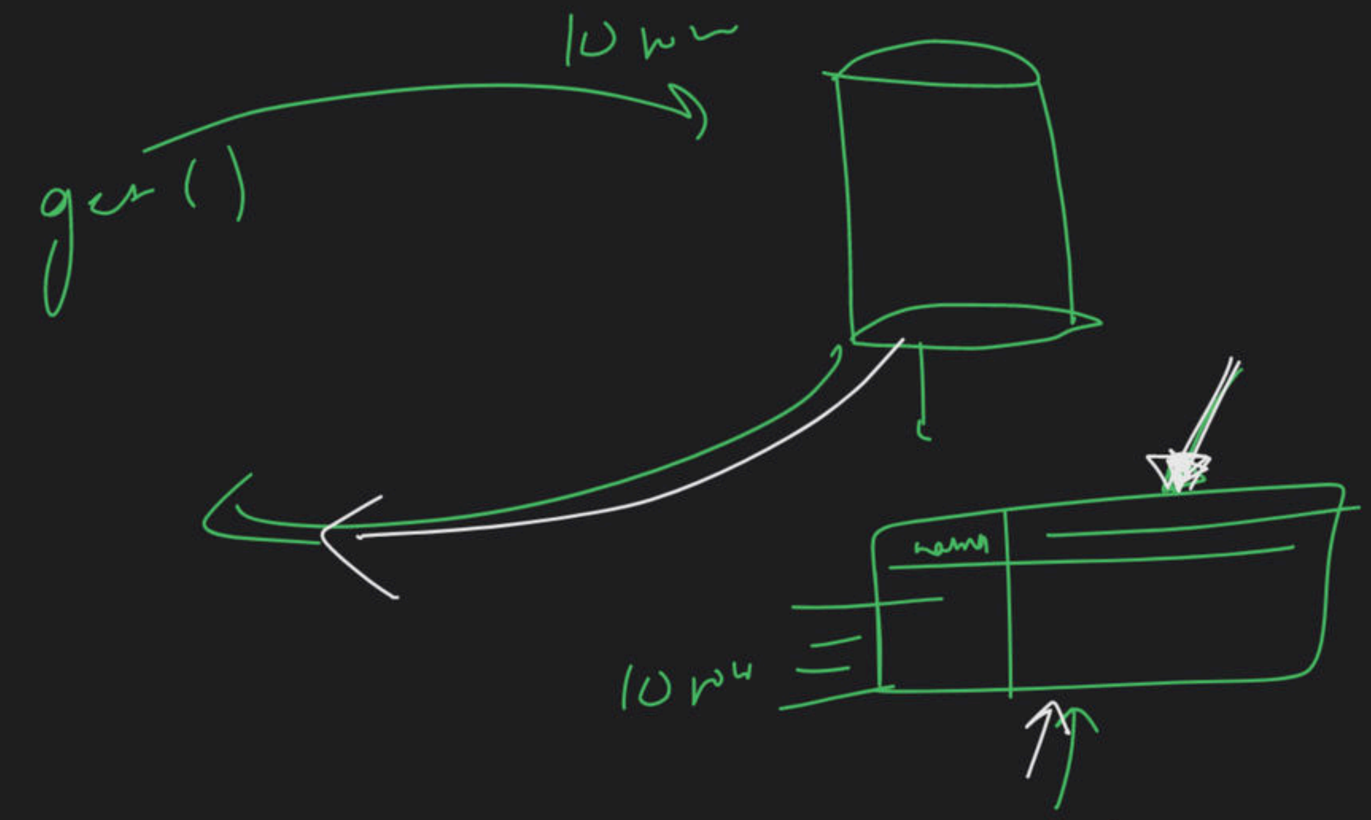
→ what? method ( ) → query / command → but not both

Query → returns a state w/o changing the state.

Command → changes the state but does not return any value

explore → CQS → exception → ?





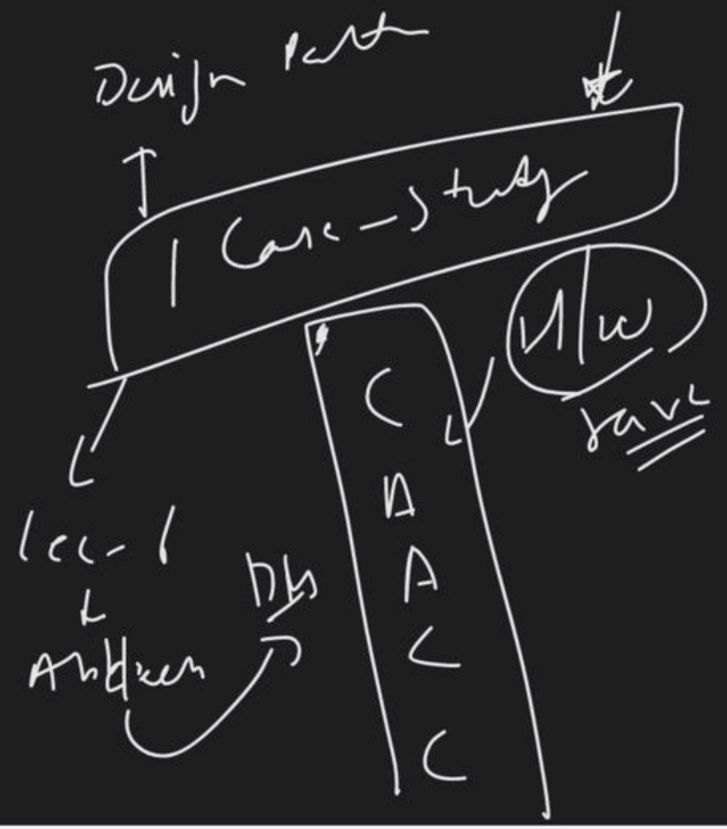
save()

get()

Overlap

Clean Code

Readability



→ SOLID → a set of principles

2 min Break

Single Responsibility Principle

(1)

```
public class Employee
{
    calculatePay();
    reportHours();
    save();
}
```

→ follow vs Not

No

(2)

```
public class Task
{
    downloadFile();
    parseFile();
    persistFile();
}
```

persistFile() → saveDB

follow u r No !=

→ Queue

↳ methods → follow or NIL

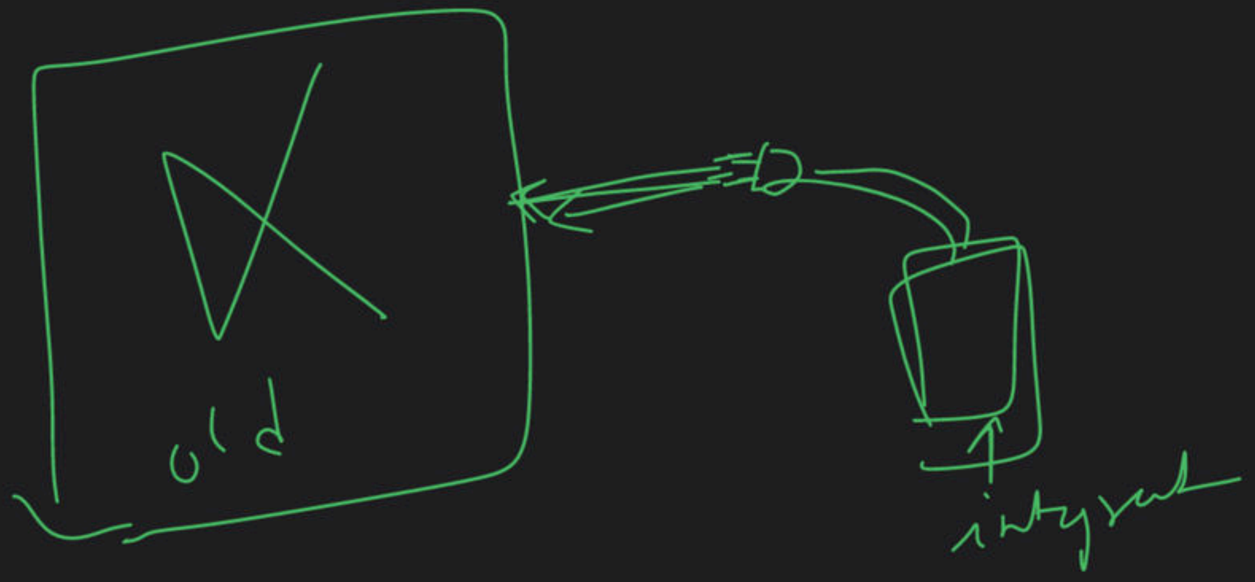
↳ Answer

Wrapper - I Wrapper

↳ Boxing

O(R) Open Closed Principle ?

"Open for extension but closed for modification"



Ex:-

```
public double Area (Object [] shapes)
{
    double area = 0
    for (var shape in Shapes)
    {
        if (shape is Rectangle)
        {
            area = ~~~;
        }
        else
        {
            //Circle
            area = ---
        }
    }
    return area;
}
```

YAGNI

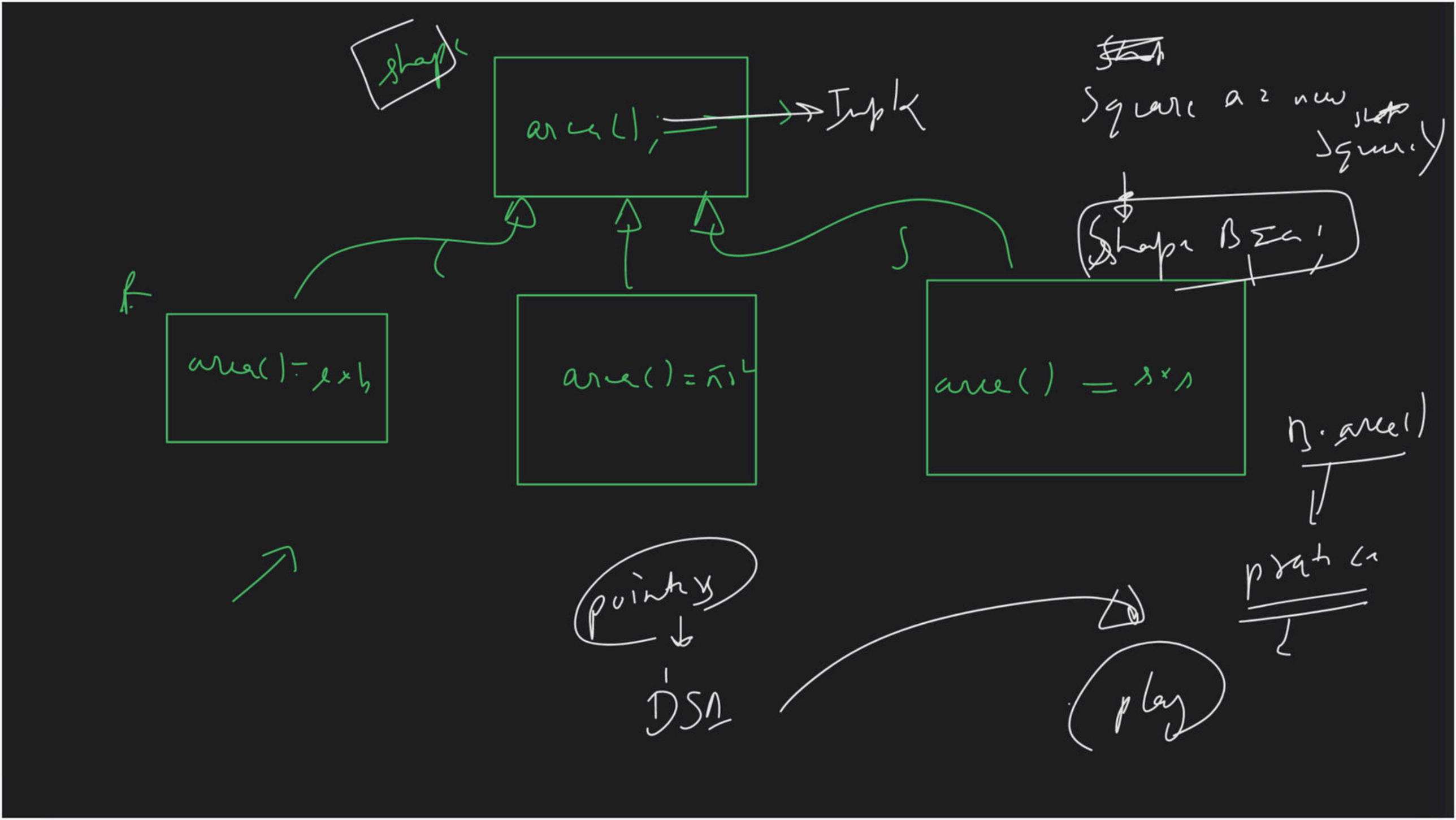
Issue- ?

not open for extensions

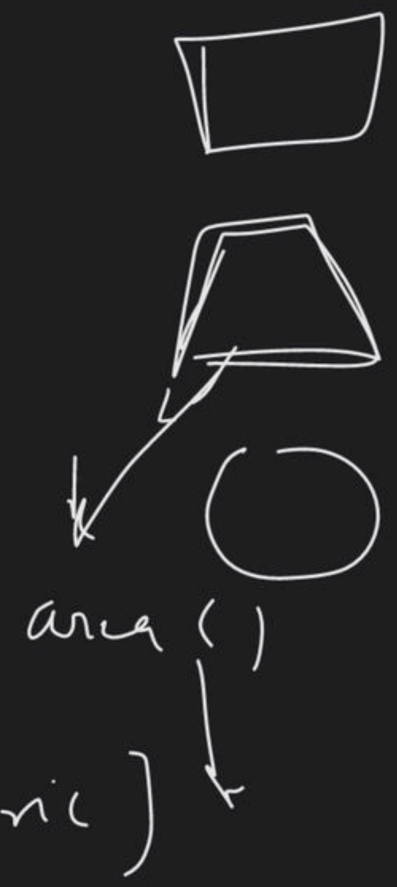
square

→ OOPS- gyaan

Generic


Shape
area();
Triple
Square a2 new
Square();
Shape Base
area() = l*b
area() = πr²
area() = s*s
pointer
DSA
play
p.area()
pratice

```
public double Area (Object[] shapes)
{
    double area=0;
    for (var shape in shapes)
    {
        area = shape.area();
    }
    return area;
}
```



→ LSP → Liskov Substitution Principle "?"

→ "Subtypes must be substitutable for their base types" }

class Rectangle → h x w

{ height = width

.getHeight();
setHeight();
getWidth();
setWidth();

}

class Square extends Rectangle

{

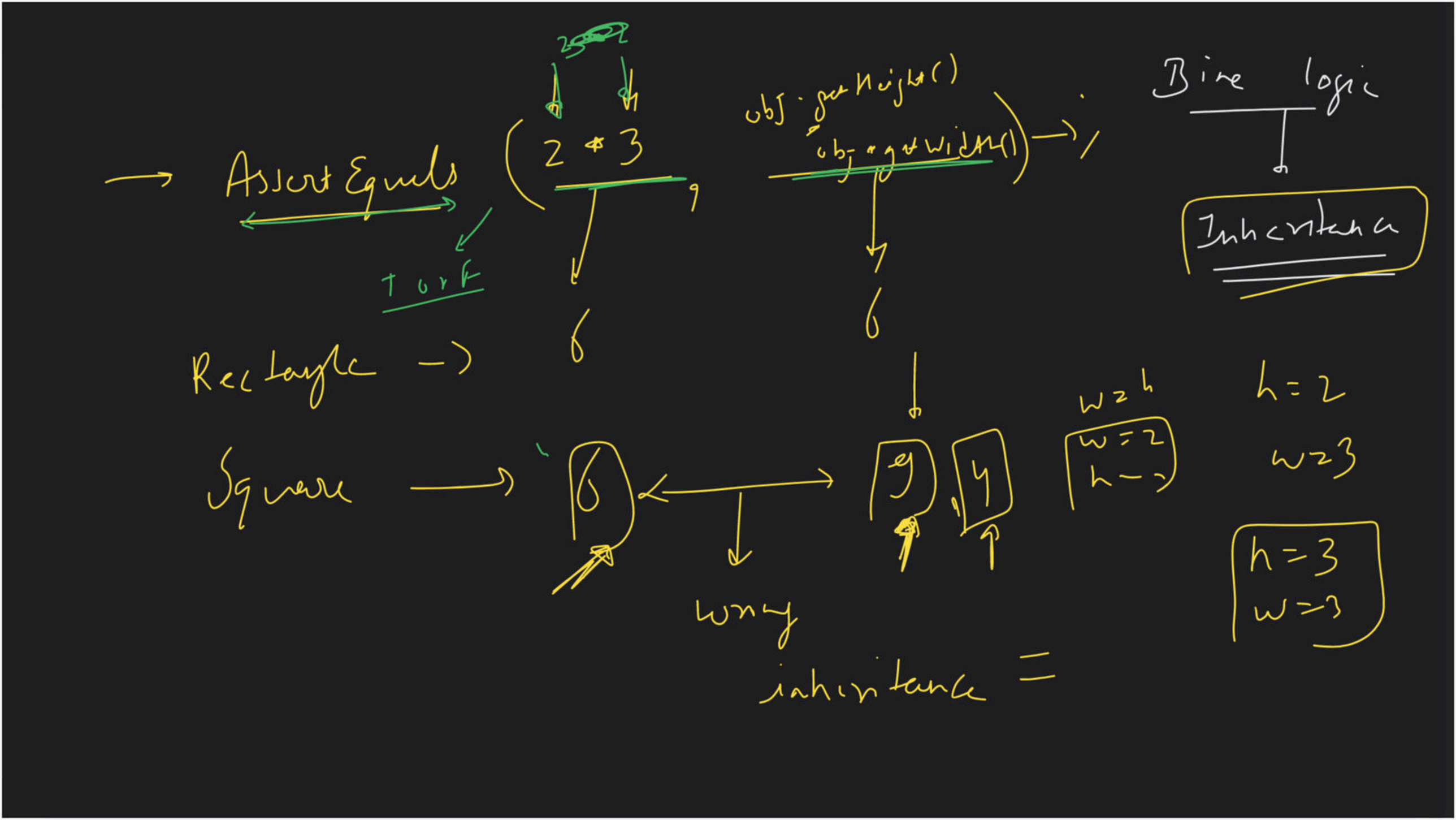
getHeight()
setHeight()
getWidth()
setWidth()
area

}

h = 3

Square
h = w
s

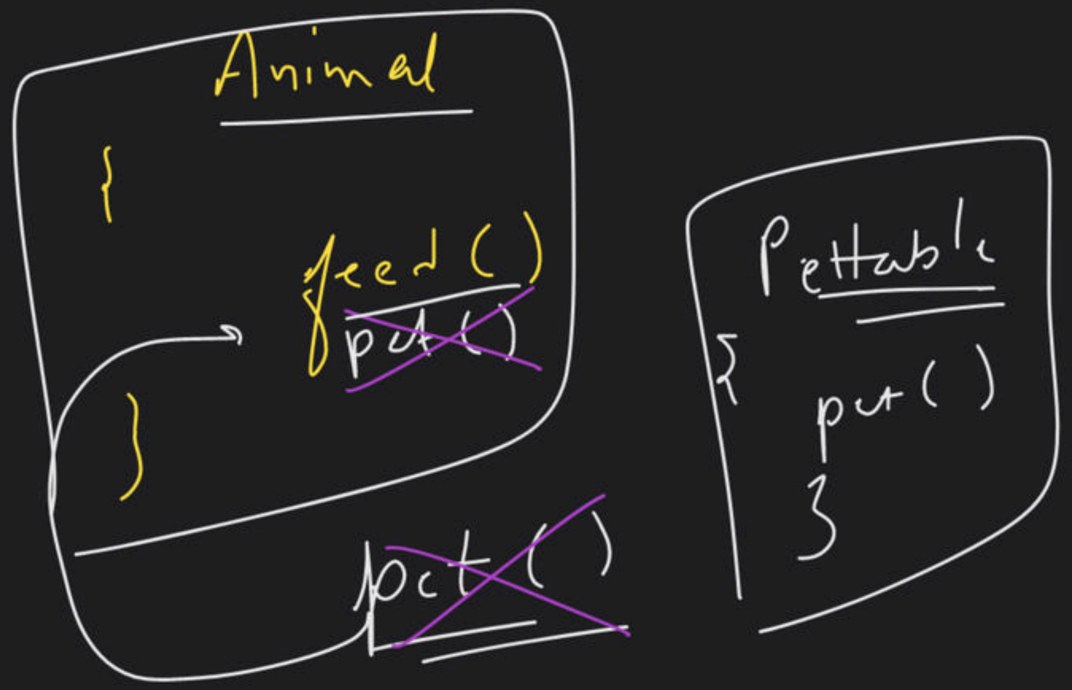
area h×w = 3×3 = 9


AssertEquals
2 * 3
obj.getHeight()
obj.getWidth()
Bine logic
Inheritance
TorF
Rectangle ->
Square
w=h
w=2
h=3
h=2
w=3
h=3
w=3
many
inheritance =

→ ISP → Interface Segregation Principle

Repeat

"Dependency of one class to another should be on smallest possible interface"



Dog implements Animal
{
feed()
put()
}

Lion implements Animal
{
feed()
put()
}

↳ clients should not be forced

to implement interface they don't use

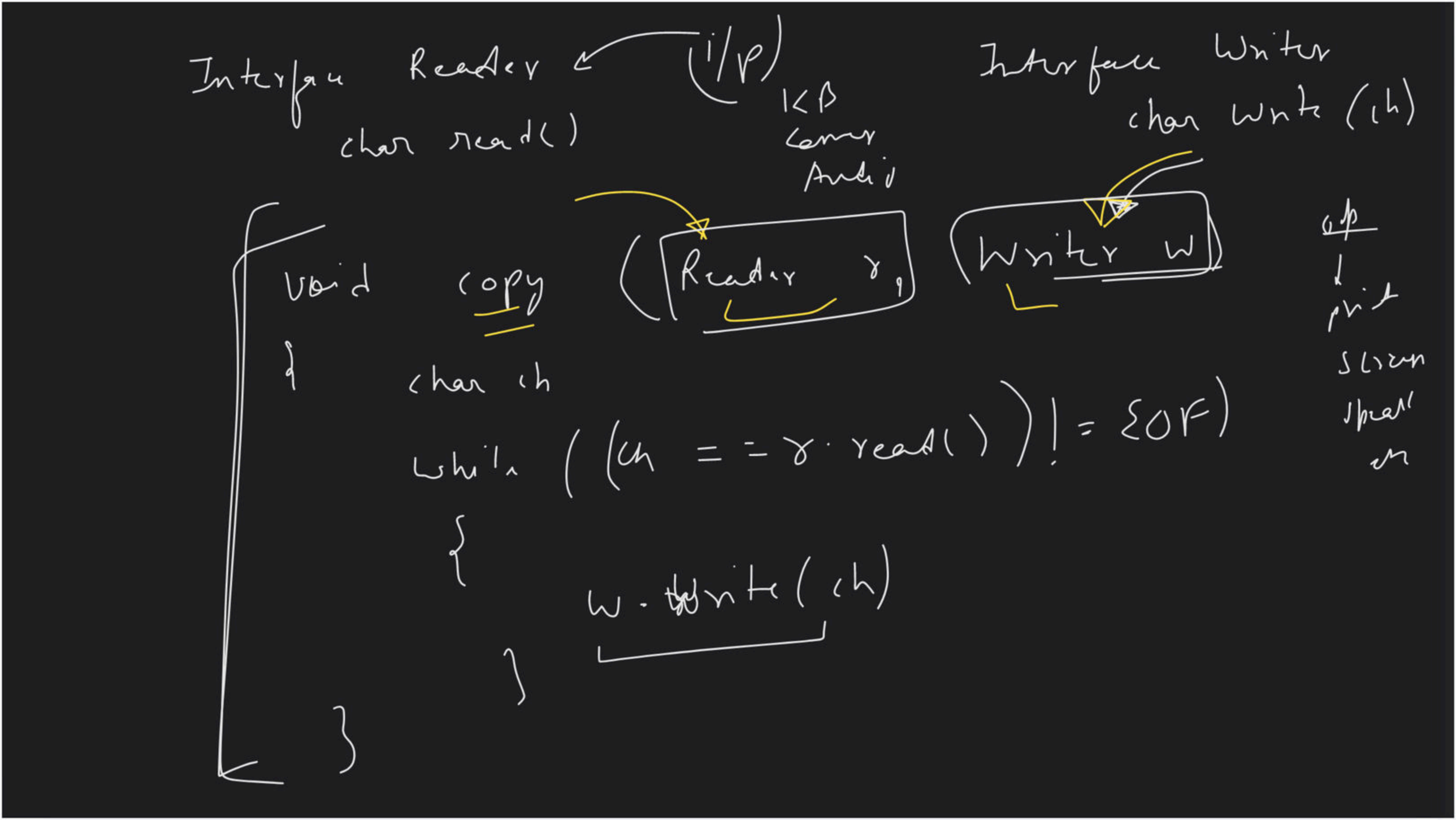
→ D -> Dependency Inversion Principle

"Depend on Abstraction (Interfaces) not on concrete classes"

Ex:-

```
void copy ( OutputDevice device )
{
    char ch;
    while ( ( ch = ReadKeyboard ( ) ) != EOF )
    {
        if ( device == printer )
            writePrinter ( ch ) ;
        else
            writeDisk ( ch );
    }
}
```

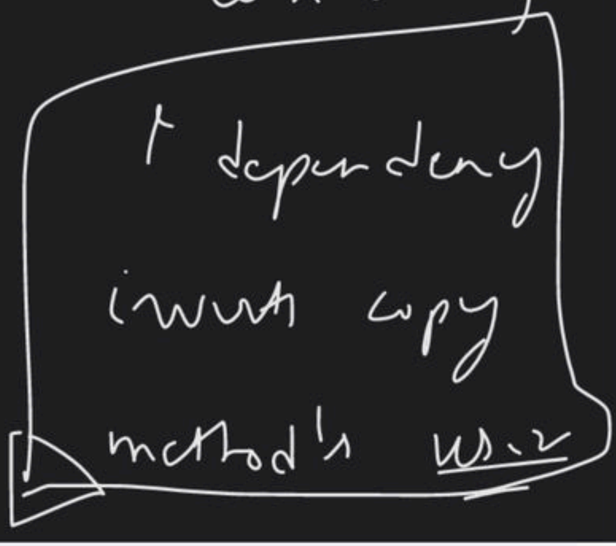
generic

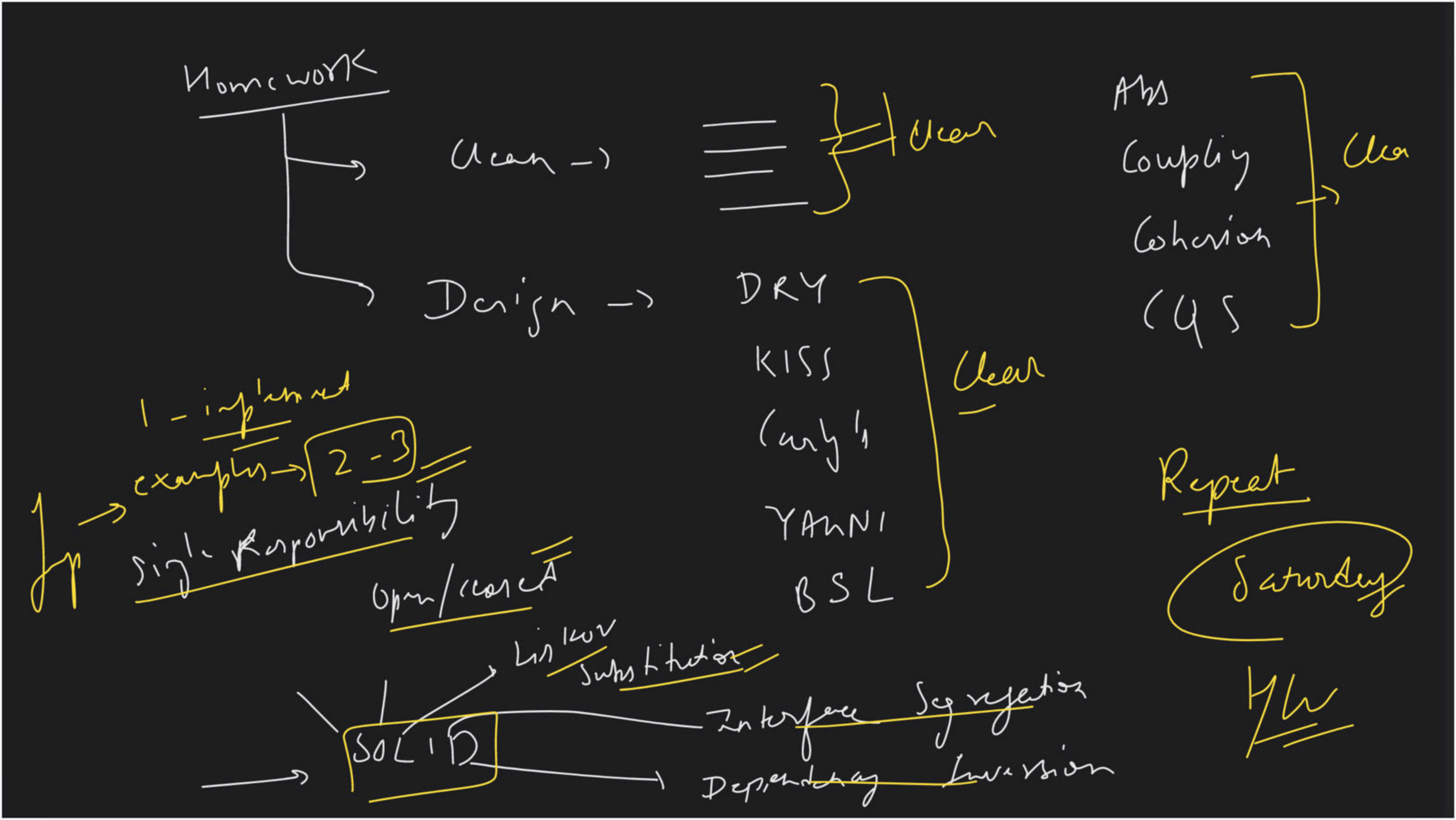

Interface Reader
char read()
i/p
KB
Console
Audio
Interface Writer
char write(ch)
void copy
Reader r
Writer w
o/p
Print
Screen
Speaker
char ch
while ((ch = = r.read()) != EOF)
{
w.write(ch)
}
}

→ Reader & writer both are dependency of 'copy' method

Ex -> 3 examples

↳ copy method → does not create writer obj




Homework
Clean ->
Clean
Design ->
DRY
KISS
(only 1
YAGNI
BSL
Clean
Abs
Coupling
Cohesion
(QS
Clean
1 - implement
examples -> 2 - 3
Single Responsibility
Open/closed
Liskov Substitution
SOLID
Interface Segregation
Dependency Inversion
Repeat
Saturday

Discord

Github

Google

LA