Let's learn Singleton & Builder Design Pattern

Special class

Love Babbar • Mar 21, 2024

xyz()

xyz(a)

xyz(a, b)

xyz(a, b, c)

Builder

Pattern

Engine

Tyres

Interior

BodyShell

Car → Sc

↳ Tyre

Car → For

↳ Tyre

Construct

S ④

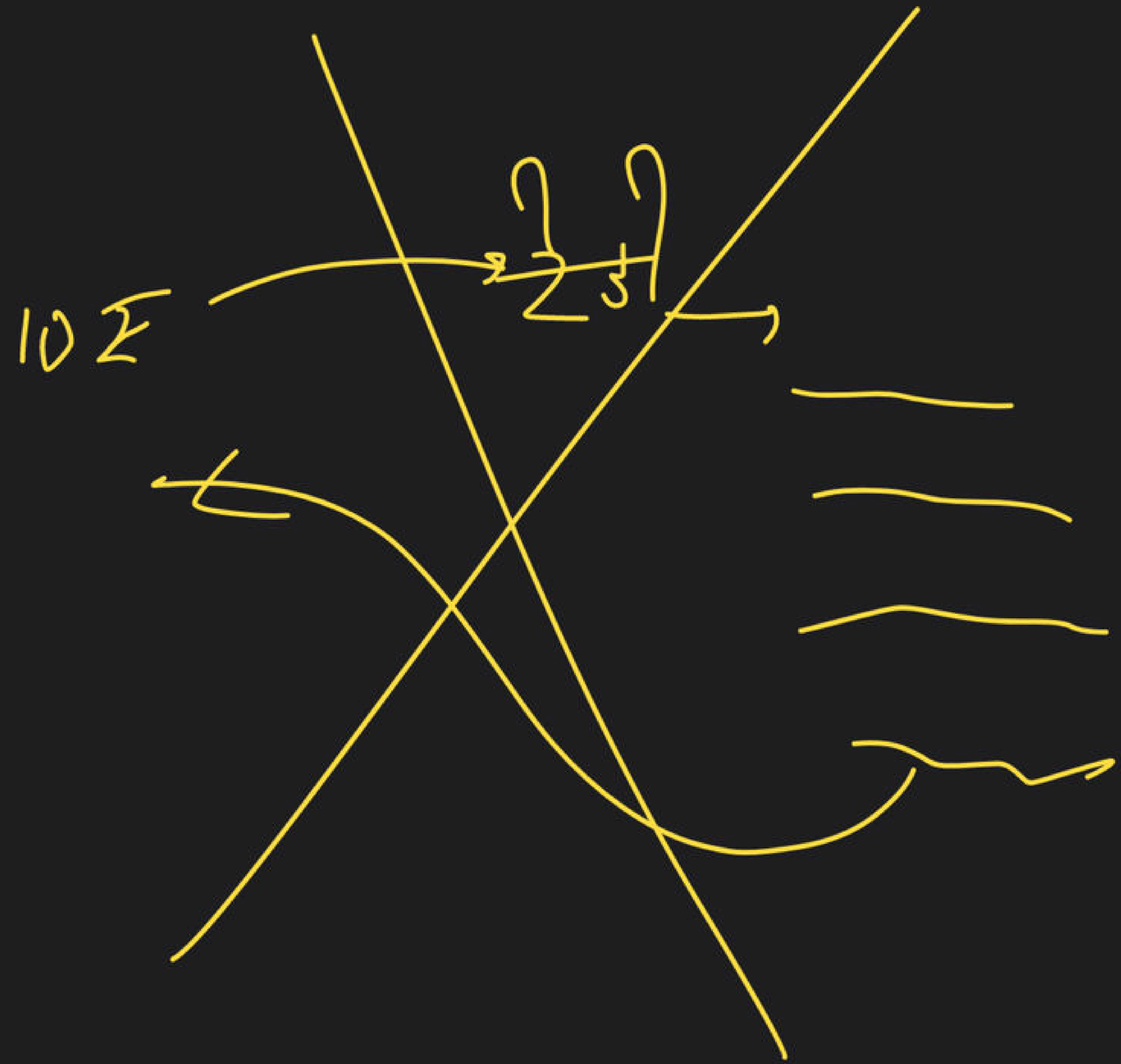
F ③ → TA

Builder Pattern:

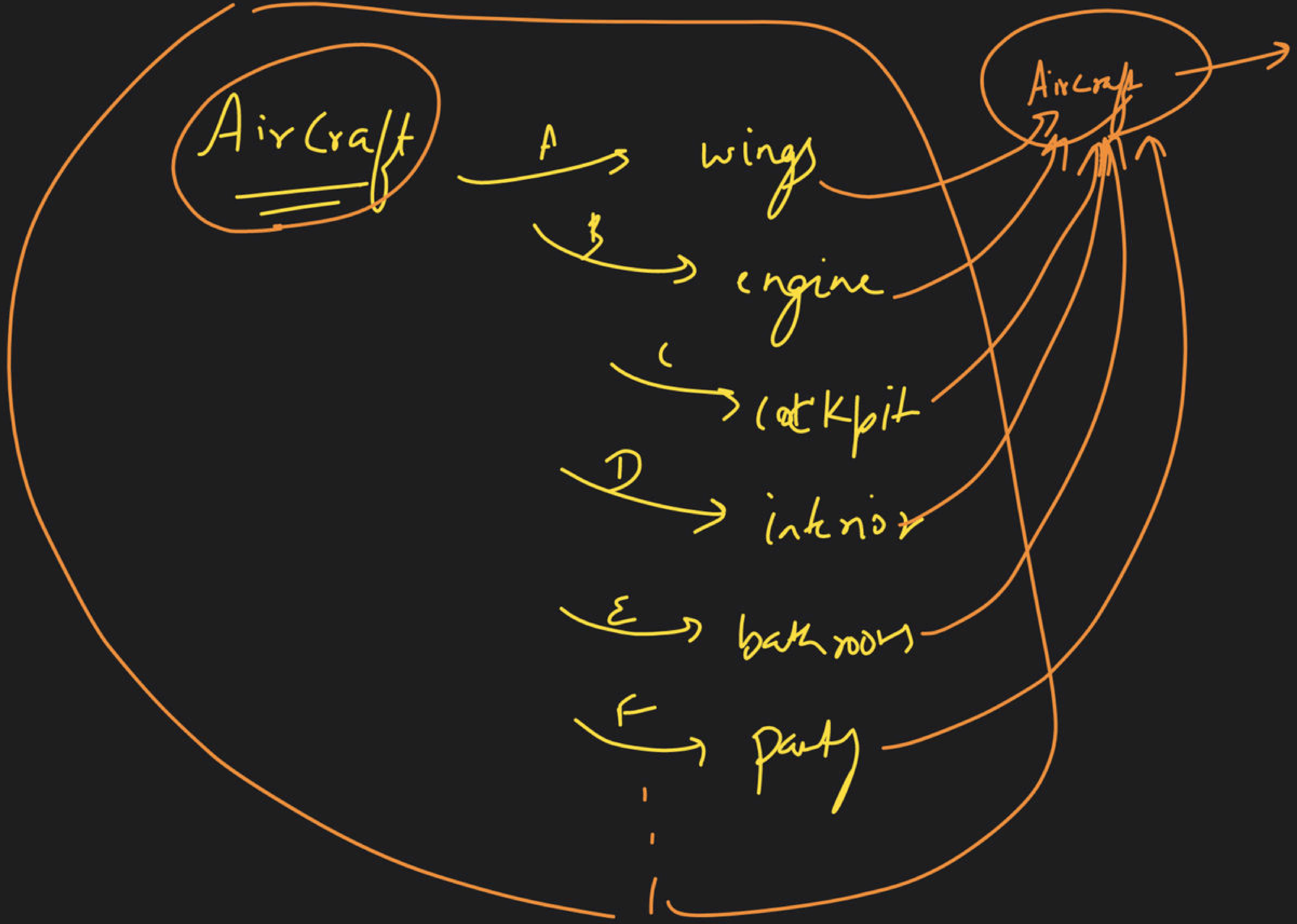
aggregate obj / Composite → Builder

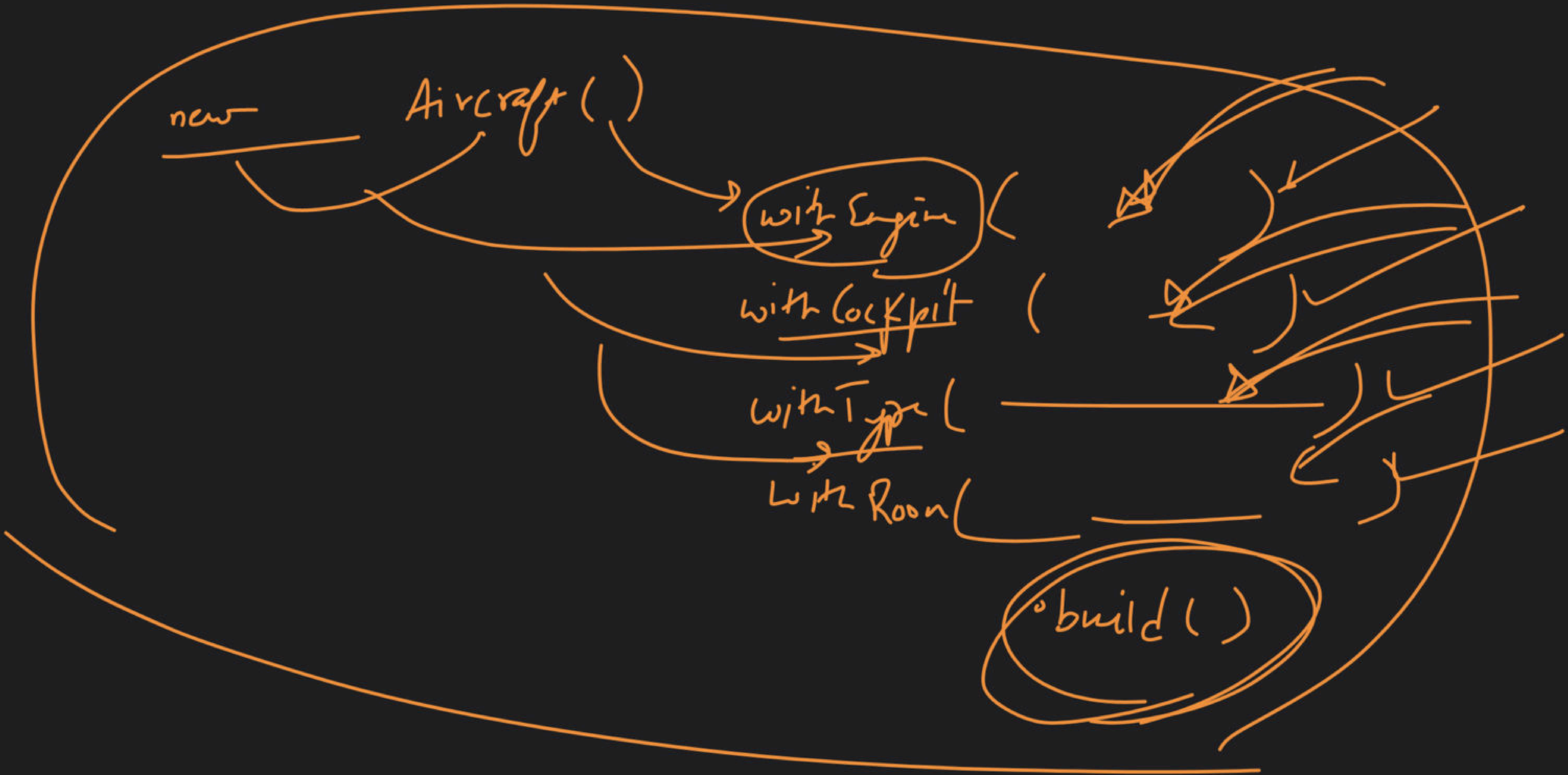
- create diff type of object → with same contruction process
- step by step
- hide → process of creating an object

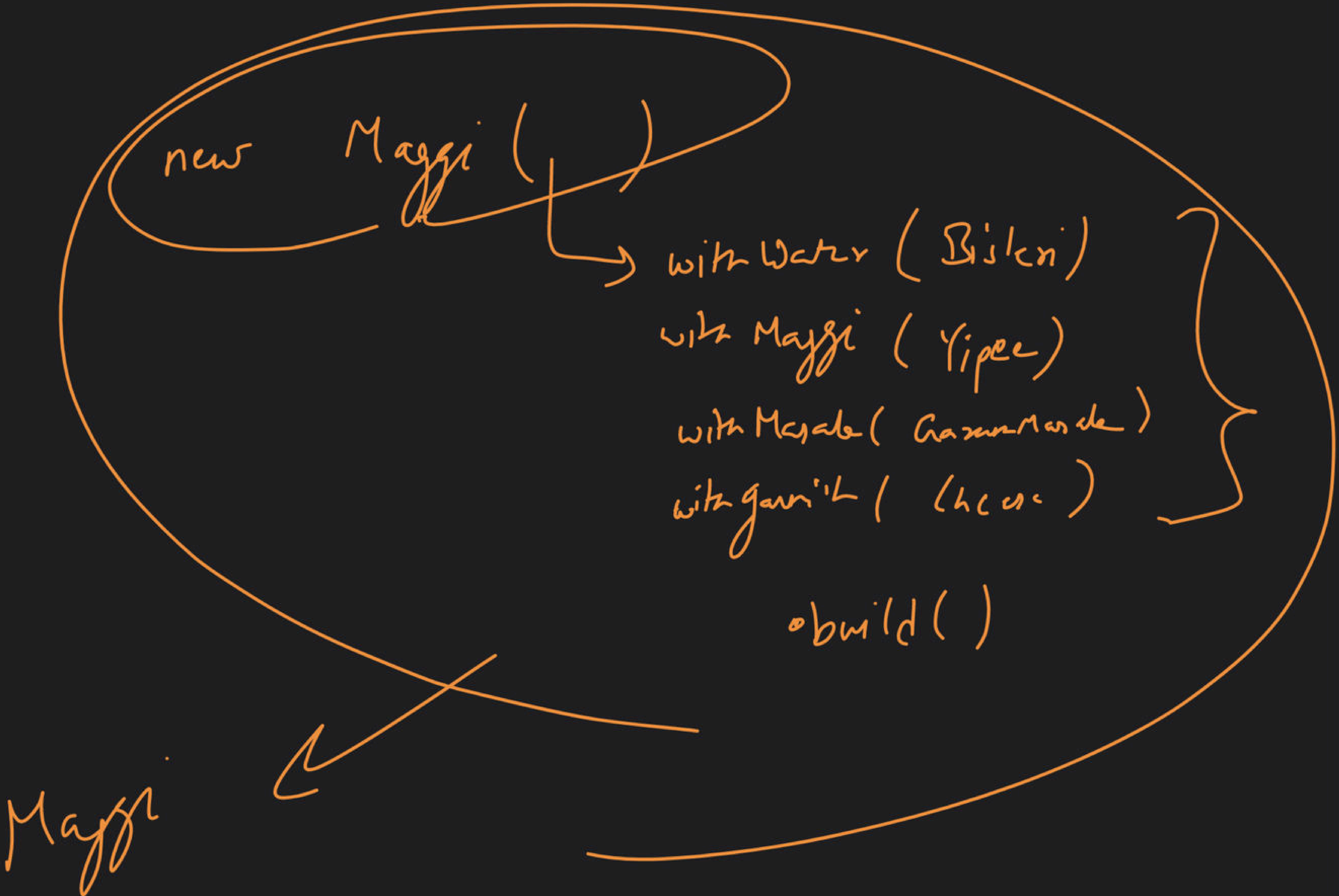
A


10 E
2
3

10 E
Builder
patter


AirCraft
A
wings
B
engine
C
cockpit
D
interior
E
bathroom
F
party
Aircraft


new Aircraft ( )
withEngine (
withCockpit (
withType (
withRoom (
.build ( )



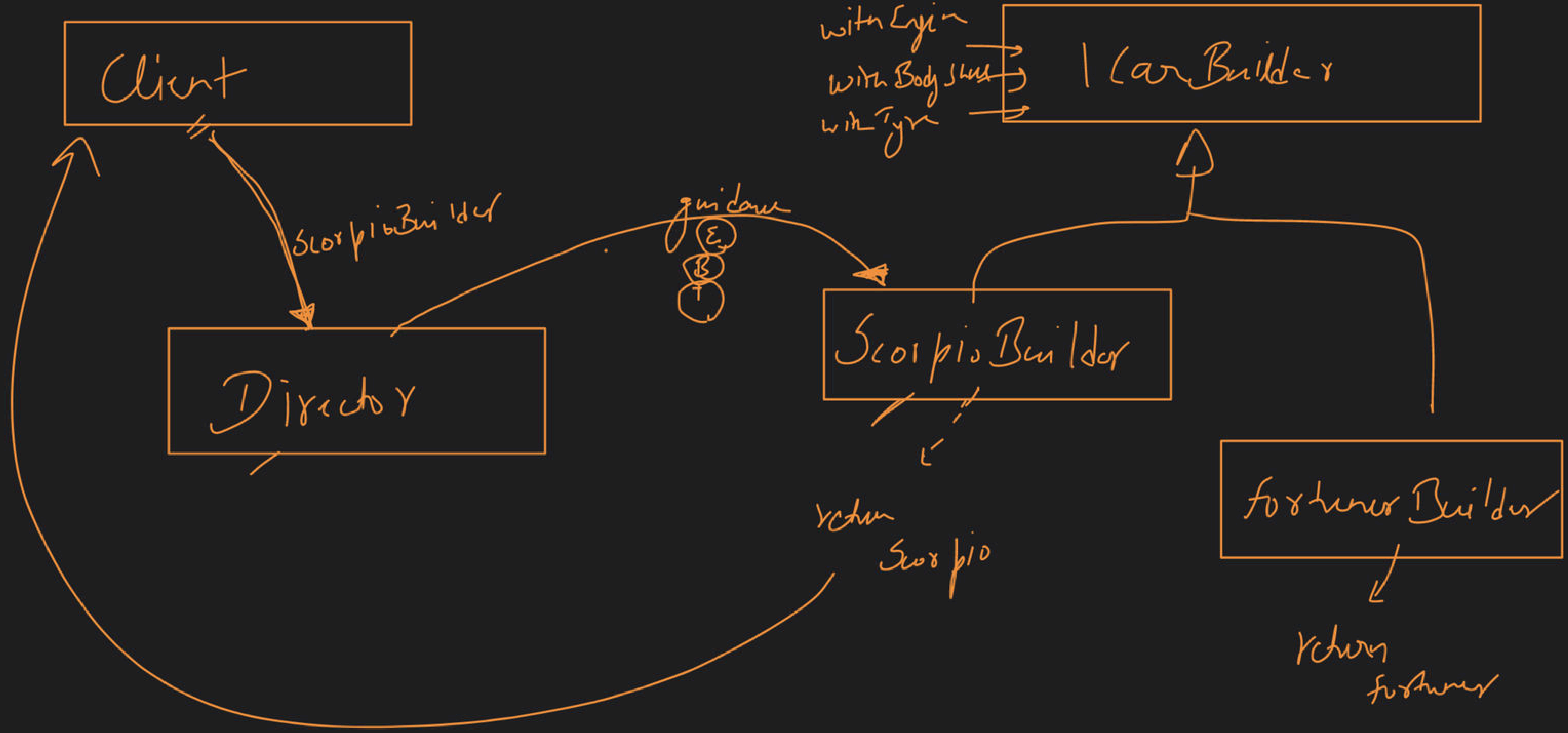
```
new Maggi ( )
    withWater ( Bisleri )
    withMaggi ( Yipee )
    withMasala( GaramMasala )
    withGarnish ( Cheese )

    .build( )
```

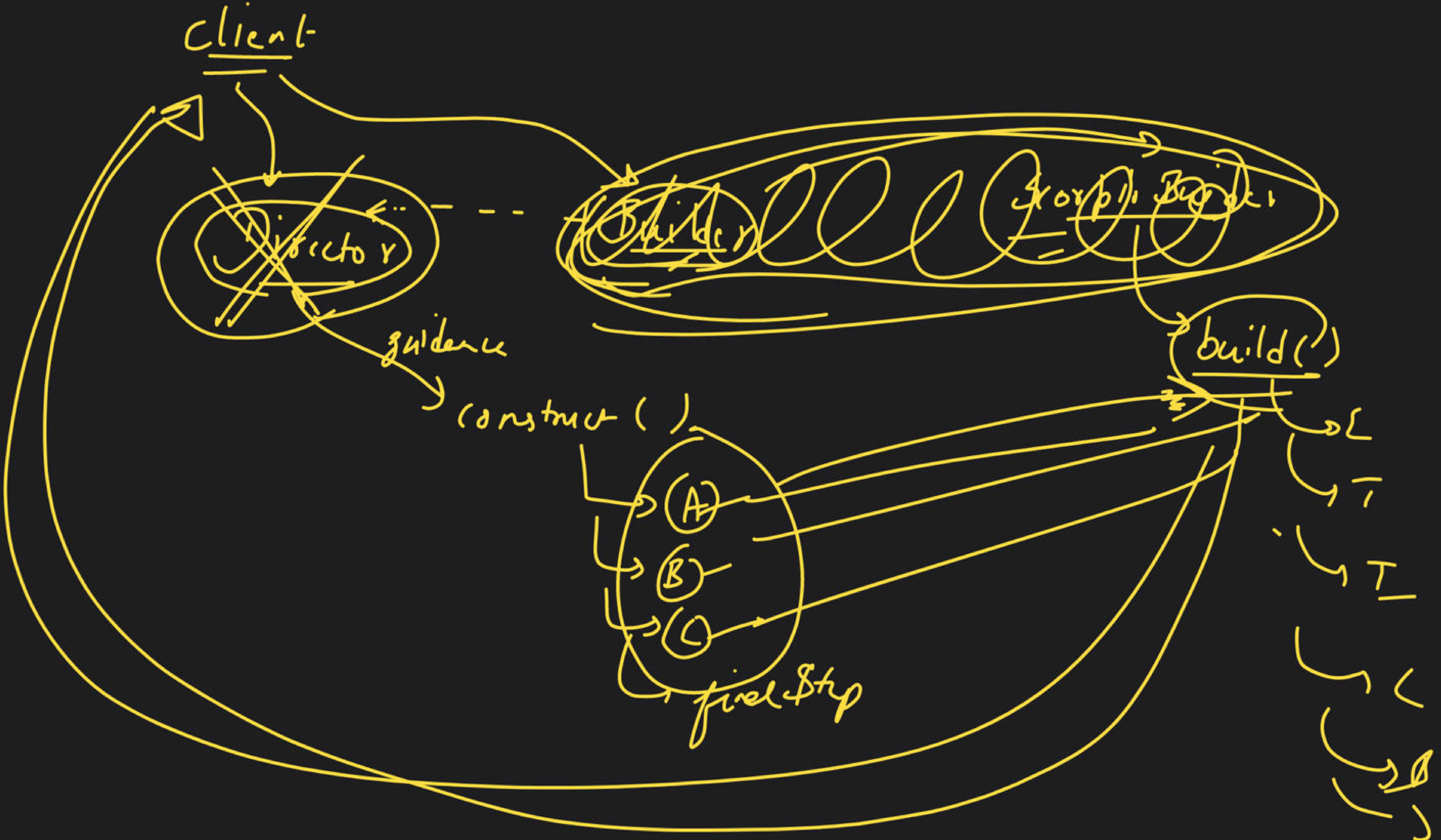
Maggi

Players → Director → directs construction of object

Builder → implementation

End → final Object → Builder


Client
Director
ScorpioBuilder
guidance
E
B
T
ScorpioBuilder
ICarBuilder
withEngin
withBodyshell
withTyre
fortunerBuilder
return Scorpio
return fortuner


Client
Director
Builder
ComputerBuilder
build()
guidance
construct()
A
B
C
finalStep

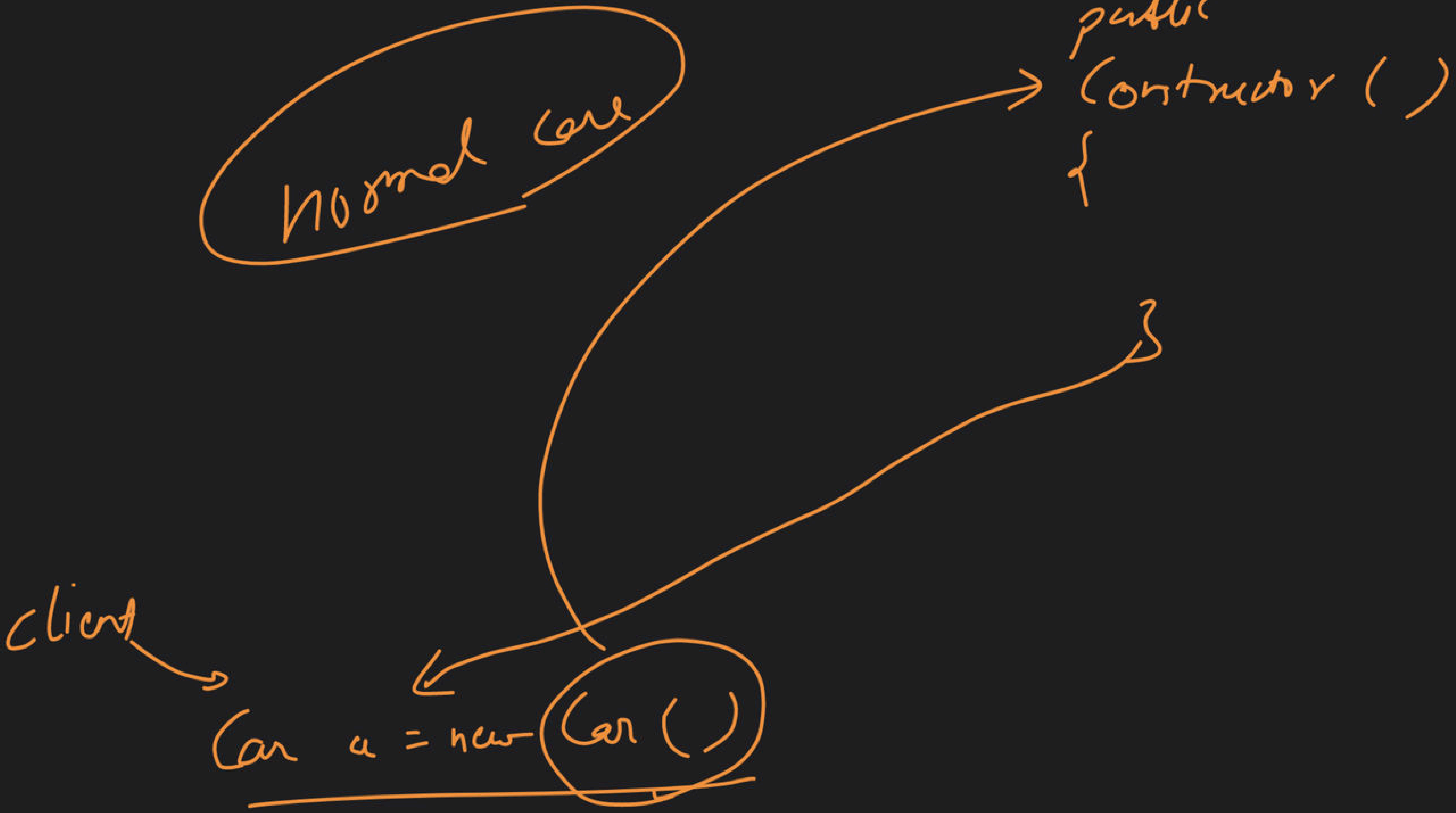
Builder

Abstract Factory

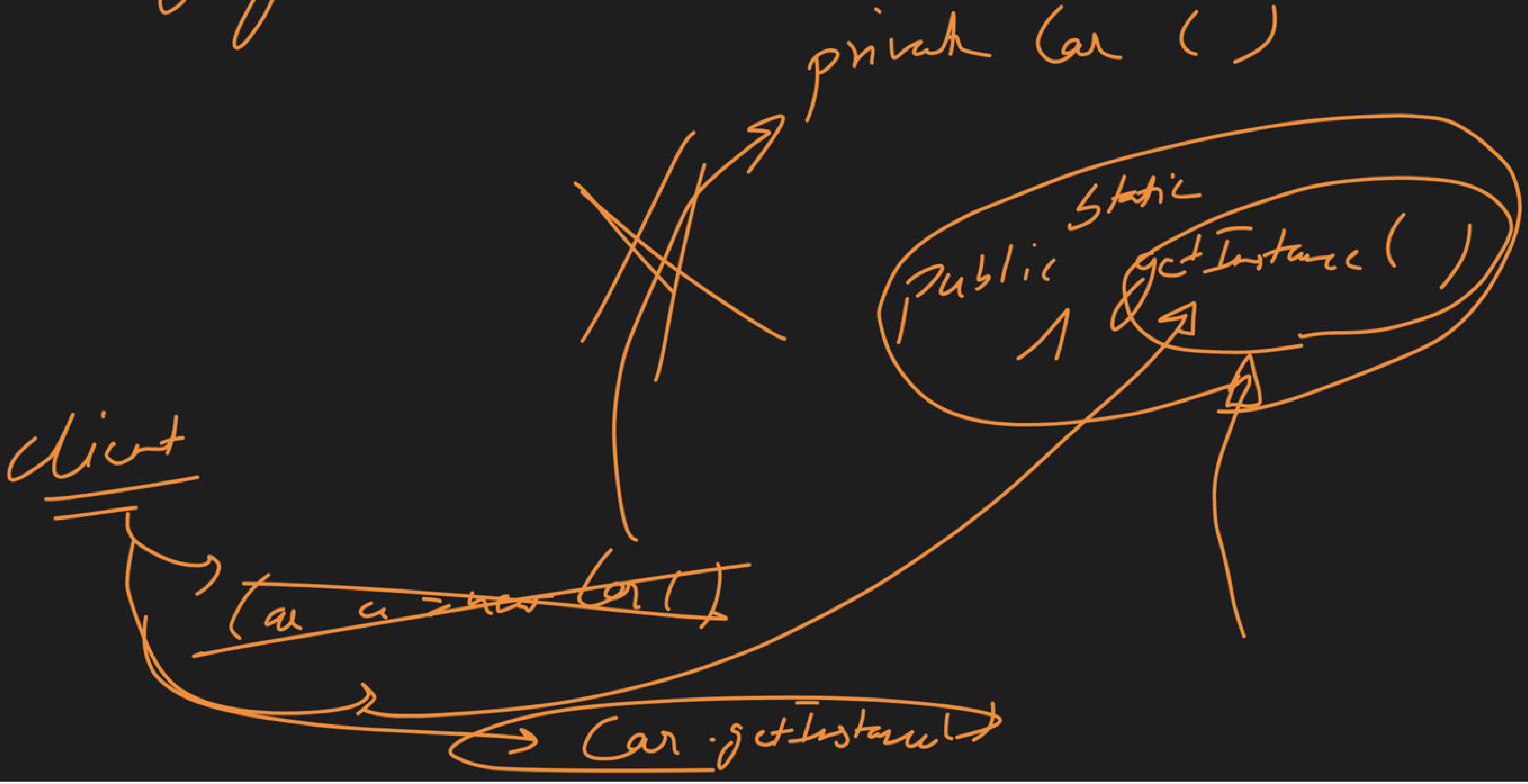
$$\begin{tikzcd}
{} \arrow[r] & \boxed{\text{Singleton}} \arrow[r] & \text{enforce} \arrow[r] & \boxed{\text{single obj}} & \text{(application lifetime)} \arrow[l, bend left]
\end{tikzcd}$$

How

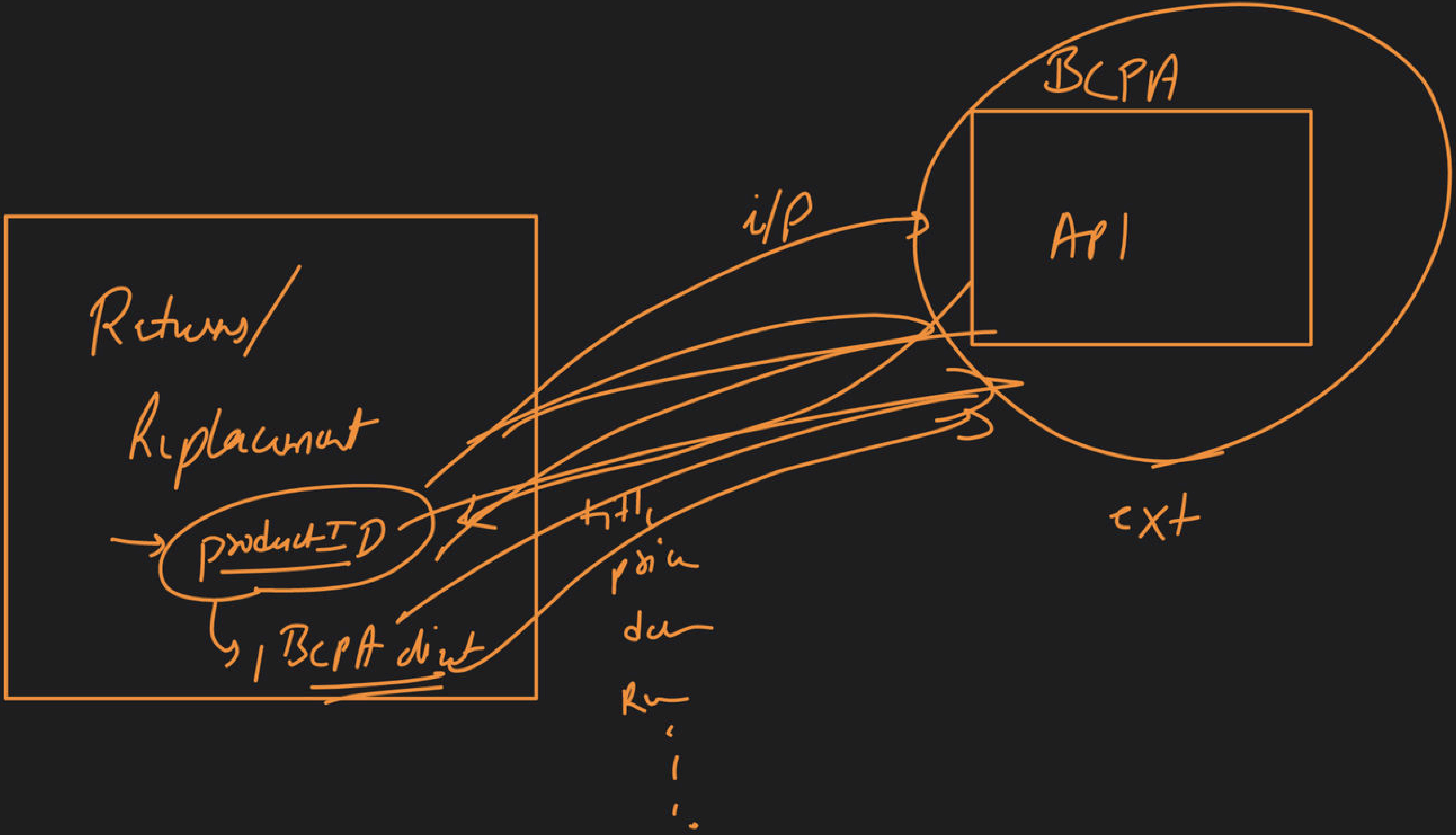
$$\begin{tikzcd}
\text{How} \arrow[r] \arrow[dr, bend right] & \text{private Constructor( )} \arrow[r] & \text{inside class data member} \\
 & \text{public getInstance( )} \arrow[u, bend left] &
\end{tikzcd}$$


Normal car
public
Constructor ( )
{
}
client
Car a = new Car ( )

singleton

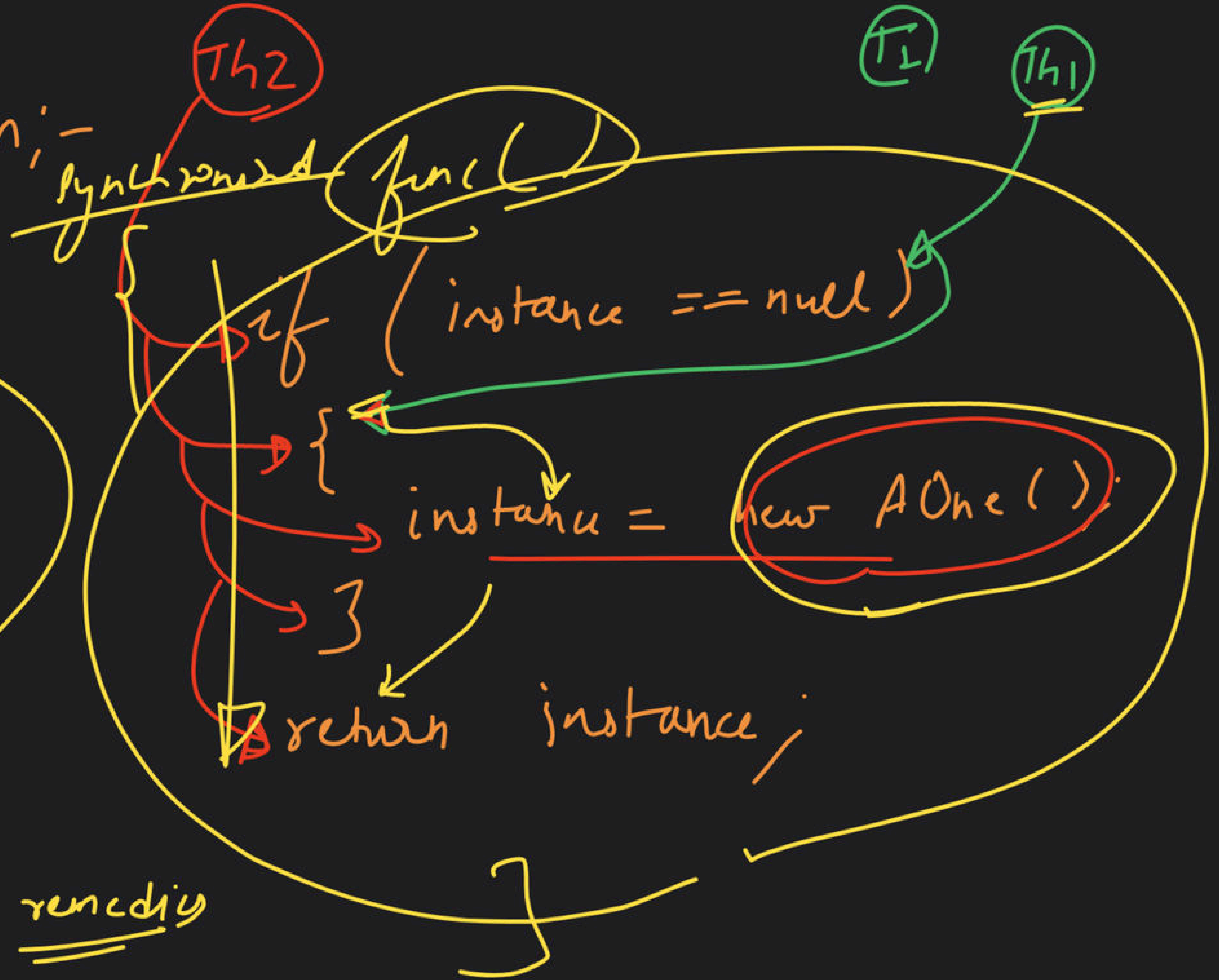

private Car ( )
static
public getInstance( )
client
~~Car a = new Car()~~
Car.getInstance()

fighter jet $\longrightarrow$ Aircraft One



Returns/
Replacement
productID
BCPA dict
i/p
BCPA
API
ext
title
price
desc
Rev

Singleton :-



```
~~synchronized~~ func()
{
    if (instance == null)
    {
        instance = new AOne();
    }
    return instance;
}
```

Kaise Bachu ?

Java → 2 remedies

lazy-initisation

Java →

Ⓐ synchronised → LOCK

expensive

wastage of resources

early initialisation

Ⓑ static-initialisation → performance

solves →

initialisation-on-demand

getInstance()

→ return Aone

~~public~~ private static Aone = new Aone()

dependency ✗

① synch → exp → locks

② static -init → perfome → westage of reson → obj, not needed

→ early-init

class {

class
{
INSTANCE = ...;
}

initialisation on demand

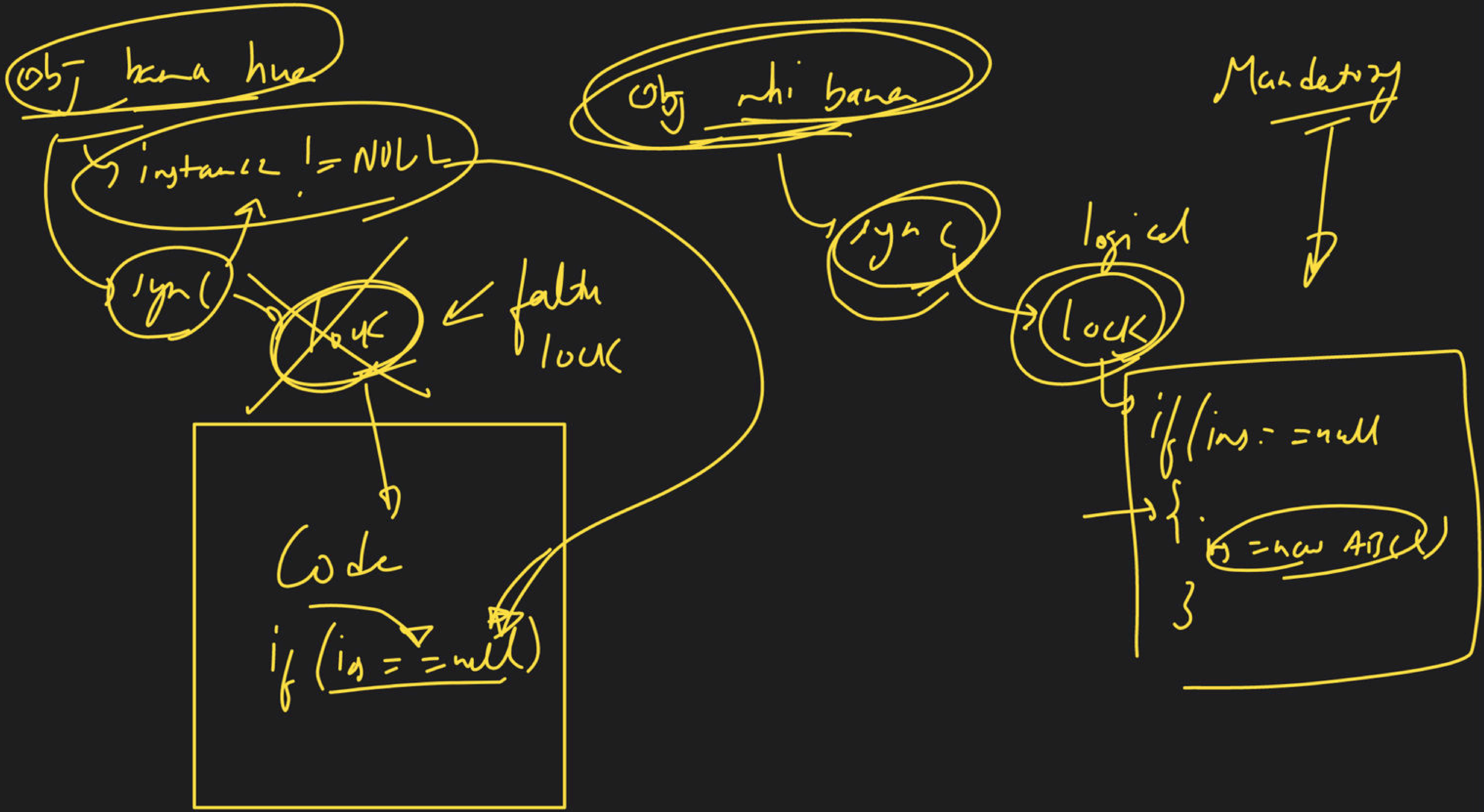
Holder.INSTANCE

Meyer's Approach

Singleton

→ java

C++



Obj bana hue
instance != NULL
sync
~~lock~~
falthu lock
Code
if (ins == null)
Obj nhi bana
sync
logical
lock
Mandatory
if (ins == null)
{
ins = new ABC()
}

Double-checked Locking :-

Obj nahi bana $\rightarrow$ Lock ✓

Obj pehle se bana hua h $\rightarrow$ ~~Lock~~ ✗

Java

→ volatile

~~2 things~~

2 synchronized

```
synchronized func()
{
    if (obj == NULL)
    {
        synch (  )
            if (obj == NULL)
            {;
```

C++

Lock locks.