AGENDA

1.> ADAPTER design pattern

2.> Facade design pattern

[STRUCTURAL DP]

start by 9:05 PM

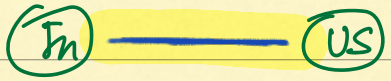
*] STRUCTURAL DP:

concerned with How to structure code base

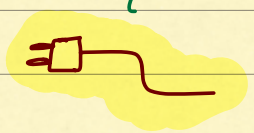
-> what attributes

-> How class communication takes place

what all classes should be present



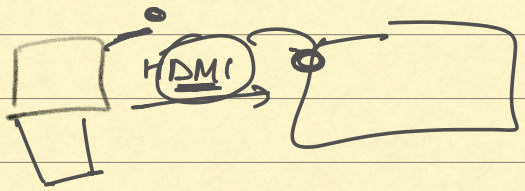
*] ADAPTER DP:



Adapter:

Power Adapter/ converter

different types of cable in countries for devices...



Mac

Eg: MacBook only supports c type → you need an connector to connect with MONITOR.

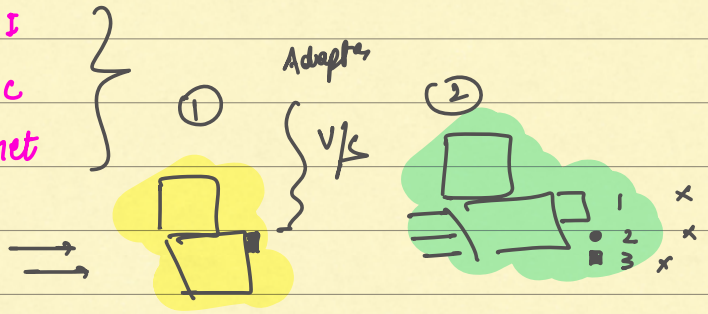
Adapter - Intermediary layer that connects two things.

↳ Similarly: Adapter dp solves a similar problem

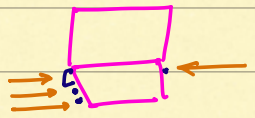
Example:

Apple wants to support all types of port in MAC.

- HDMI
- USB-C
- Ethernet



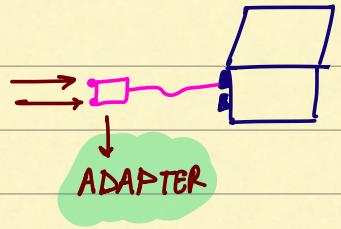
→ H/w needs to be designed + Need more code to support diff. ports.



↳

This becomes tough.

Instead just have 1 type of port & support via that



Problem Statement:

In code bases. Often we use 3rd Party libraries to support our code

Nuances:

1.) Library Integration can change
today — use AWS API
tomorrow: Google API

→ this would lead to HIGH development effort

2.> 3rd Party library is NOT available anymore.

→ Hence, we want to use New library...

D - D.I

Easy?

Phonepe → yesBank XX

MOST COMMON EX: PHONEPE

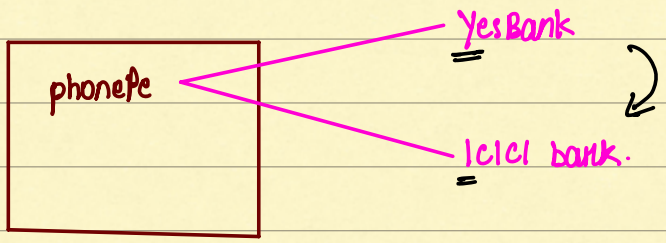
1 DAY

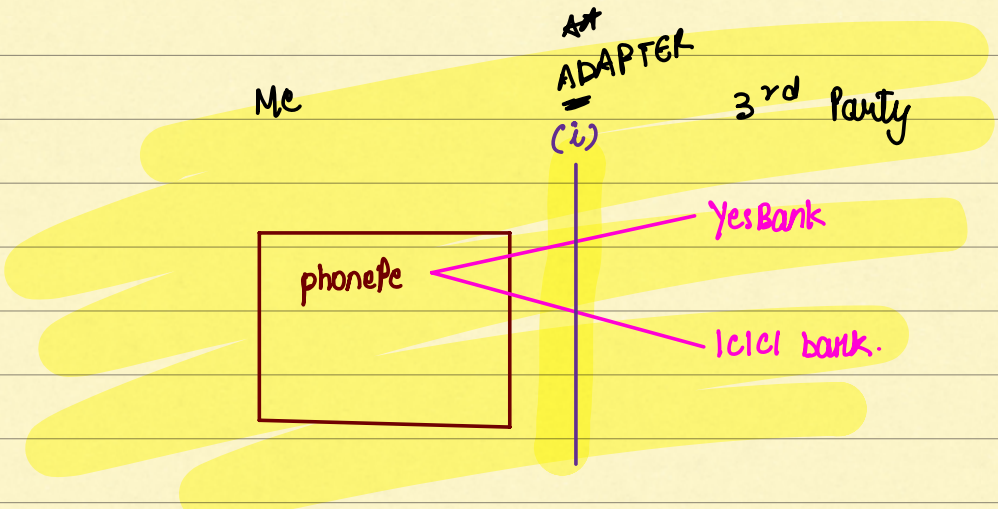
Interfaces

D

•> Initially dependent on yesBank

•> Moved to some other Bank





Here; phonepe depended on External API via → (i)

•> using direct API : was Problem [STRONG COUPLING]

Revising DI:

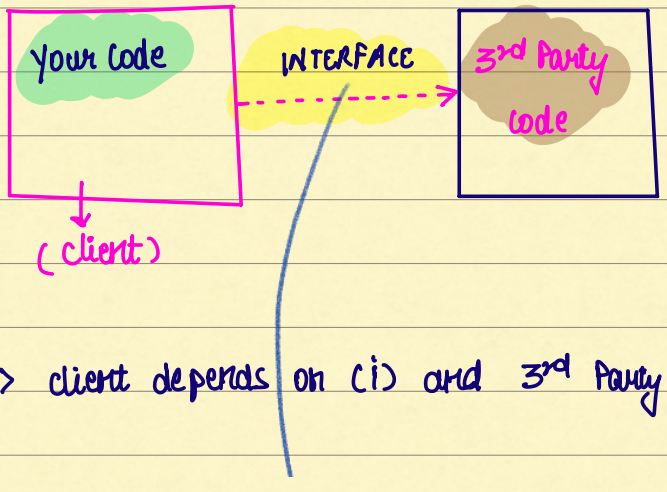
-> if code is integrated with any API directly:

Involves lot of tight coupling b/w your code & 3rd party API.

affecting Maintainability

SOLUTION:

use interfaces to solve. (D.I.)



-> client depends on (i) and 3rd Party API Implements (i)

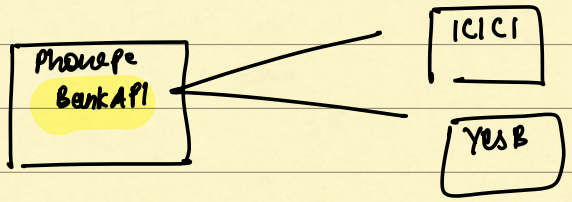
↓
ADAPTER

★] **STEPS - HOW TO USE ADP:**

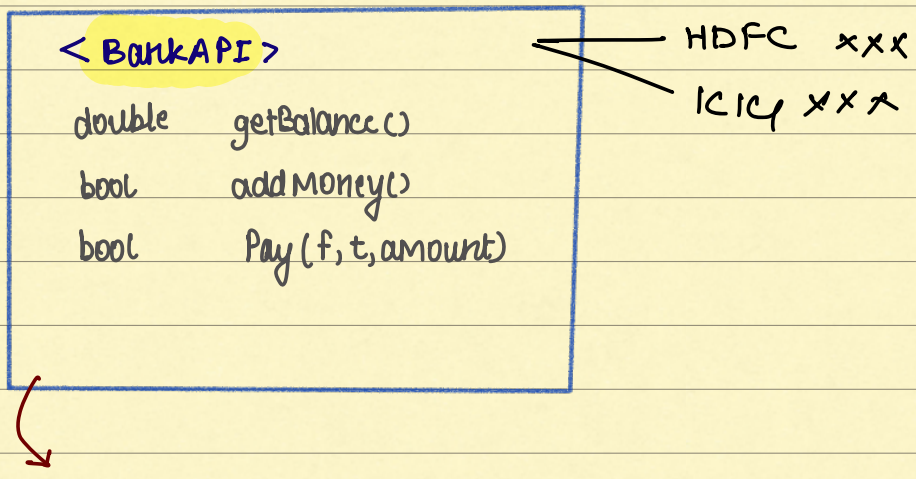
S1:> Always create interface; whenever connecting to 3rd Party

S2:> Add Methods → which you Need from 3rd Party

PhonePe **INTERFACE <BankAPI>** YesBank

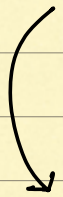


-> phonepe decides what methods added in interface



Phonepe's Interface.

•> Finally → Bank Implements the (i)



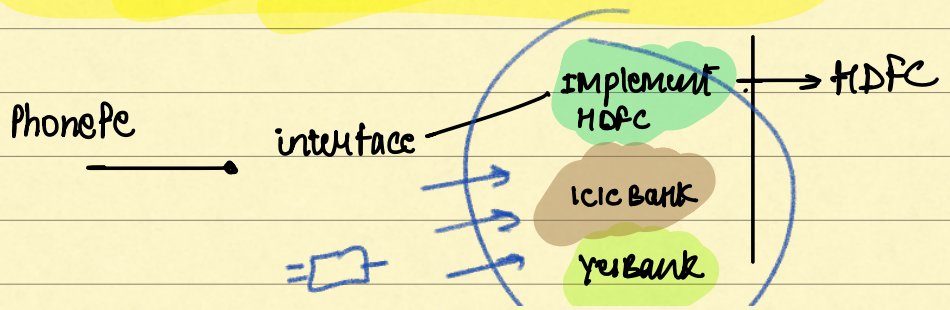
Q•> would Any bank Implement any interface ?

→ yesBank would just create their class & Ask client to Integrate.

YesBank API

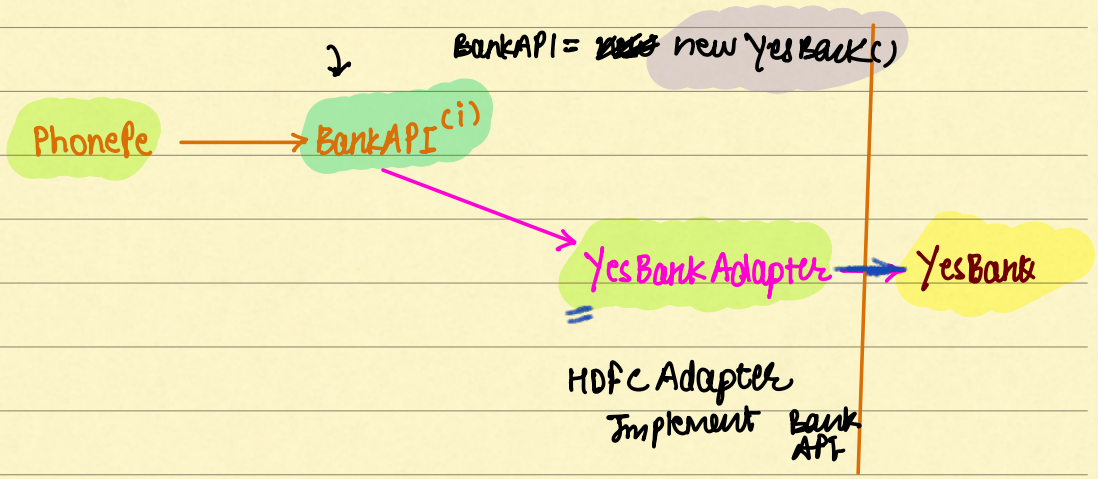
- fetchBalanceforAccount ()
- transfer (from, to, amt)

Methods.



S3.> As any Bank would NOT Implement Interface;

-> Have a class Implements our Interface ⊕ uses 3rd Party API.



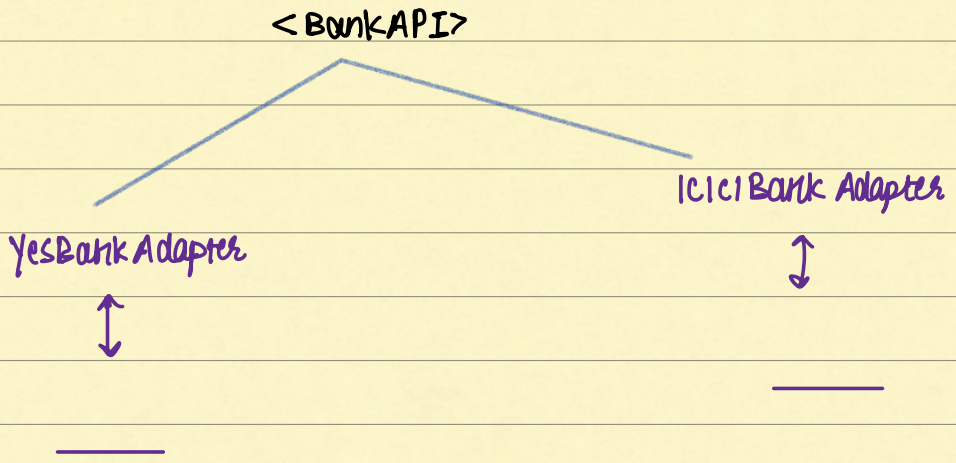
YesBankADAPter:

-> Adapter class
-> talks to YesBankAPI
-> Implements BankAPIi

OUR CLASSES	3rd PARTY
Phonepe / BankAPI /	YesBankAPI

*] ADVANTAGES

1.> switching to New class – No violation



*] REVISE

class PhonePe {

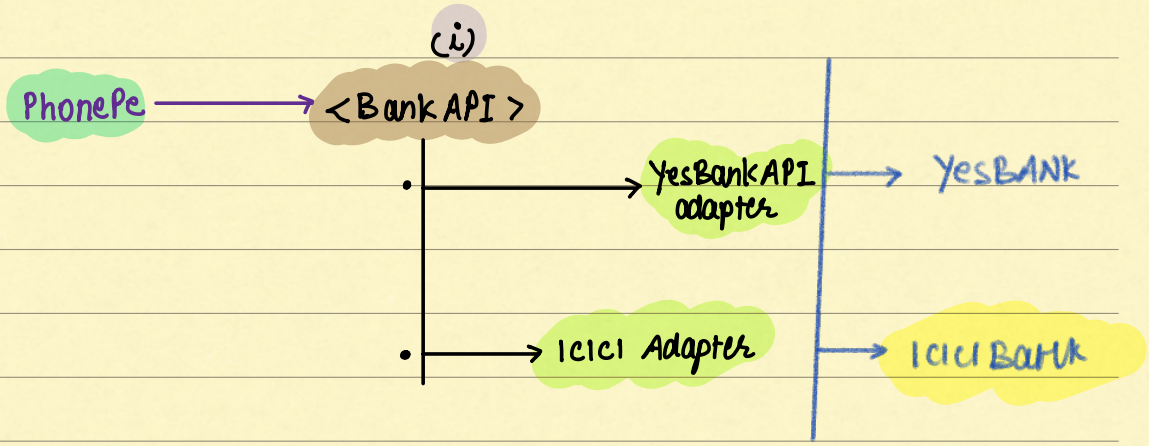
BankAPI api;

}

addMoney() {

api.add();

*] CLASS FLOW CHART:



Code walkthrough.

M.W. → calendly

functionalities:

① check free slots on calender → adding event to Google cal/...

② Add link to video conf. platform → video call...: zoom/ meet/ other

zoom/ GMeet/ ....

3rd Party APIs here:

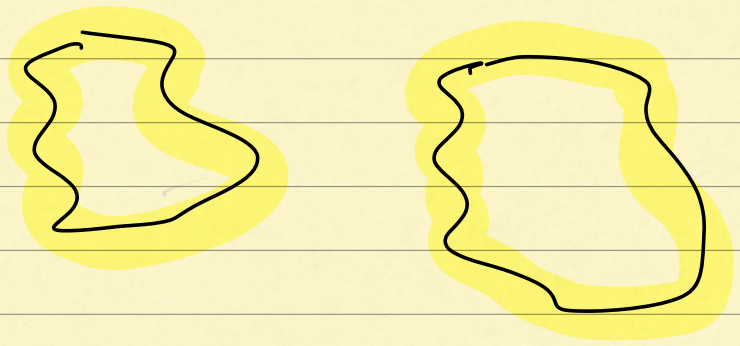
ZOOM API Adapter
Googlemeet

In calendly's code:

① video conf. Adapter
② calender API Adapter

eg: cal.com

Huddle video API Adapter



FACADE DESIGN PATTERN:

boundary / outside fence.

society :- outside area much more cleaner / simplified

→ Internally : things can be complicated...

Another Eg:

→ Banks provide relationship manager to some folks...

Relationship Manager : Makes life easy

Internally handles all complicated work.
FD creation / Requests

eg: celebrities

→ This is what FACADE DP

Instead of doing work by self, delegate

REAL CODE EG:

class Amazon

Eg: Amzn

Amazon

When order is placed → what happens?

a.> updateseller
b.> Payment
c.> logistics
d.> Notify user
:

CODE:

class Amazon {

- InventoryService is; — updates Inventory
- SellerService ss; — updates seller
- LogisticService ls;

OrderPlaced() {

// use all above services here.

.....

is.update()
ss.Notify() . . . valida-- 100 lines...
ls.createdelivery()

}

cancelOrder() {

.....

}

}

*) PROBLEM:

1.) Amazon class is very Bulky

2.) Doing All work

↳ Makes class very Big → breaks SRP...

BFSUtil

Facade DP:

if any method / class becomes very heavey OR doing lot of work

↳ Instead of doing Multiple things in single class : create Helper classes.

eg:

class OrderPlaceHelper {

- InventoryService is;
- SellerService ss;
- logisticService ls;
- analytics as;

orderPlaced() {

// same code as..

}

orderCancelled{}

Amazon {

orderPlaced() {

oph.orderPlaced();

}

orderCancel() {

och.CancelOrder();

}

}

In Amazon → just need OPH object.

```
Amazon {
    orderPlaced() {
        oph.orderPlaced();
    }
}
```

PS:

1.> Always can combine Multiple Patterns.
eg: Adapter ⊕ Facade

2.> Other similar design Pattern: Proxy Design Pattern

```
class phonePe {

    BankAPI api;

    addMoney() {
        api.addMoney(x,y,200)
    }
}
```

```
ICICIBank implements BankAPI {

    addMoney() {
    }
}
```

```
HDFCBank implements BankAPI {

    addMoney() {
    }
}
```