STRUCTURAL DESIGN PATTERNS

Agenda

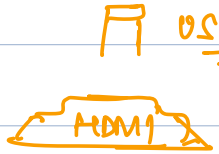
1. Adapter Design Pattern
2. Facade Design Pattern

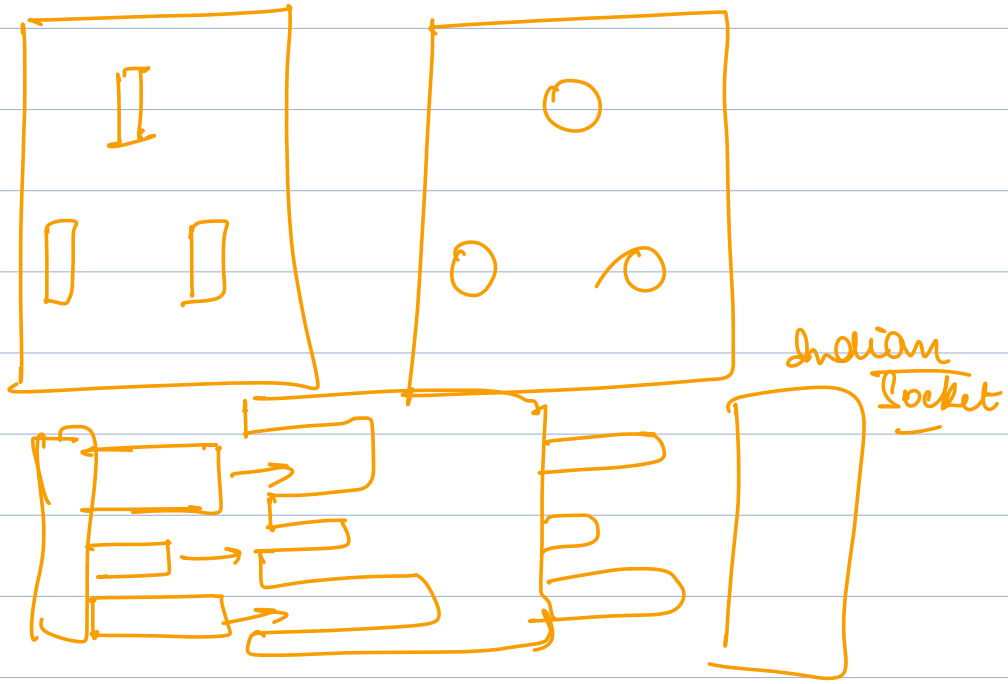
Structural Design Patterns

⇒ how to structure your codebase
- → what classes to be there
- → what attributes in every class
- → how diff classes talk to each other etc

I: Adapter Design Pattern

- → Power Adapter
- → Converter from one port to other (eg MacBook)





→ Adapter : intermediary layer that connects (transforms) one form to other

(HDMI → USB C)

(US Adapter → India Socket)

Assume apple wanted to have all types of ports

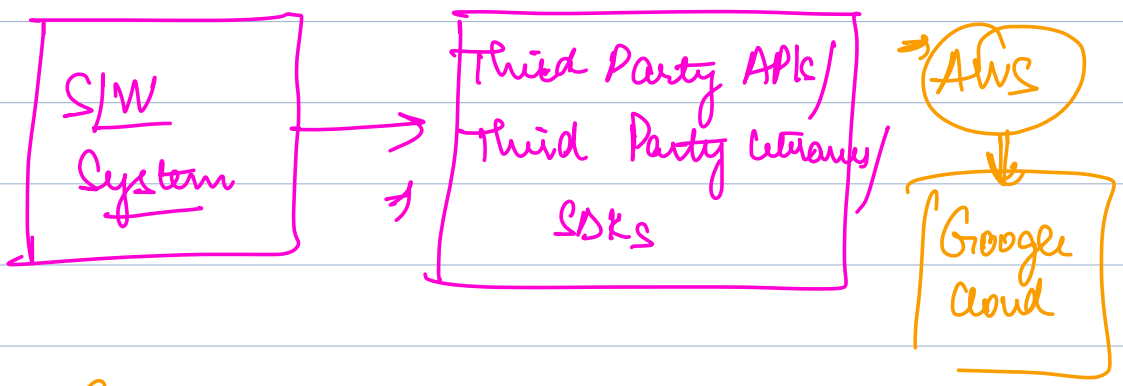
- → HDMI
- → USB-A
- → Ethernet

→ they would be required to write hardware code to talk to each of

these kind of ports.

⇒ ⇒ Currently, Apple developers only need to talk to one type of port ⇒ USB C. The work to handle other types of ports is done by adapter.

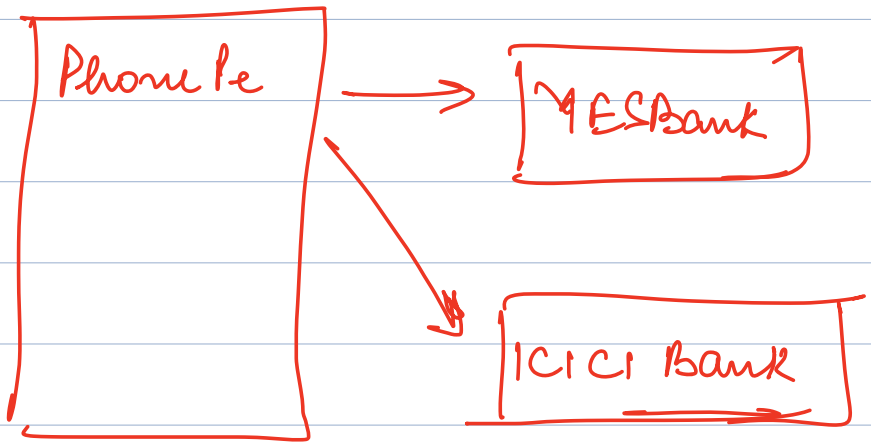
Adapter Design Pattern



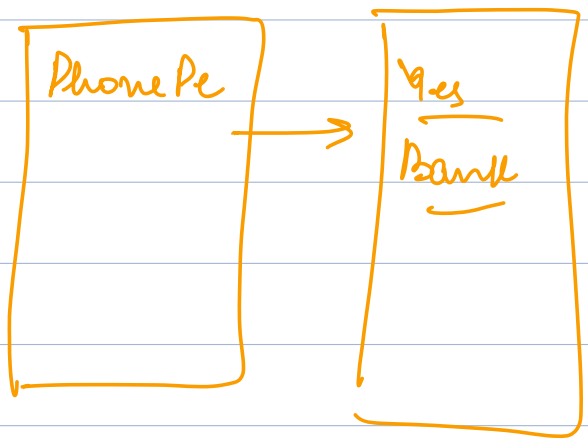
Problem Statement

1. I might want to change a third party provider in future
2. Third party API we were dependent on might get out of maintenance

We might want to migrate to another third party

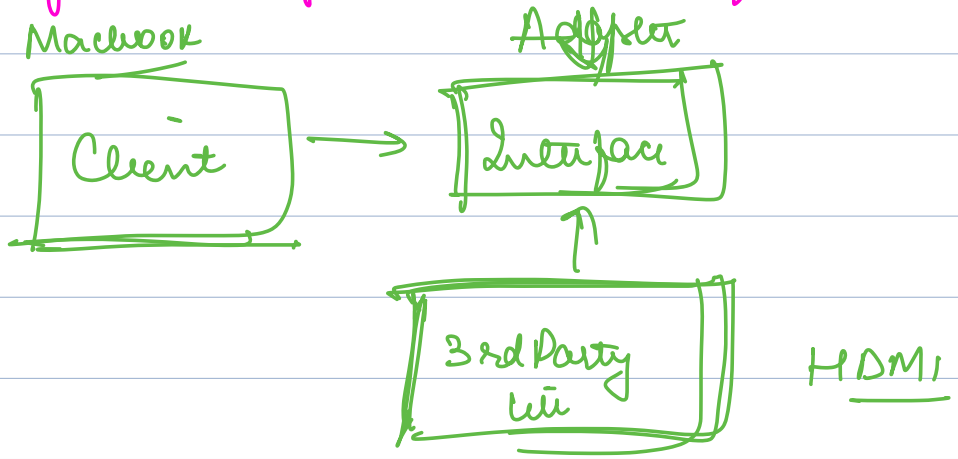


⇒ Adapter design pattern ensures that our codebase remains maintainable when we are talking to 3rd party APIs / SDK / Libs



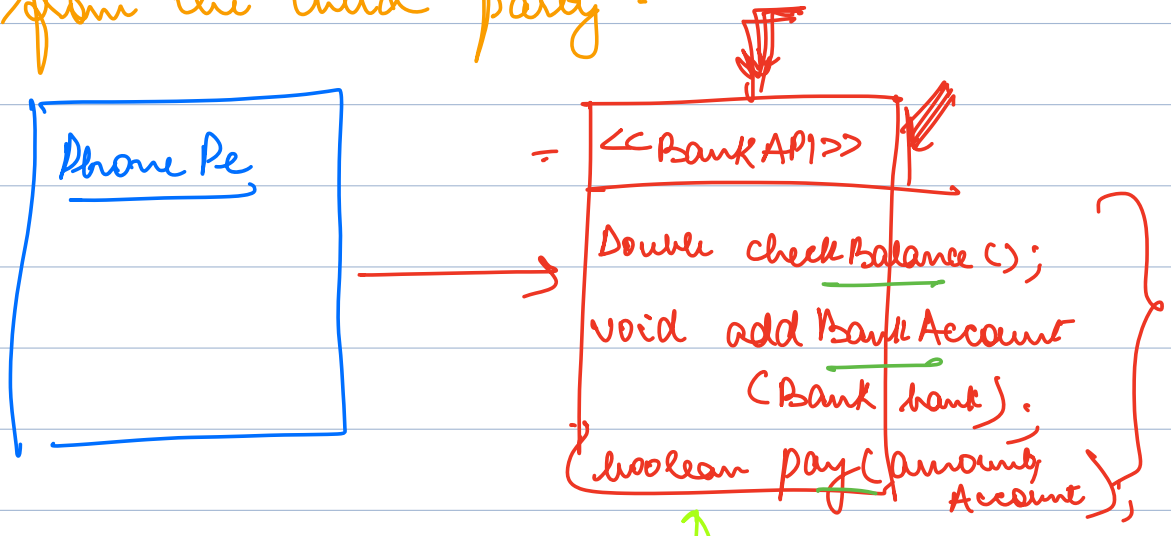
① If your codebase is directly talking to 3rd Party libraries, it involves a lot of tight coupling b/w your codebase and 3rd Party libraries. This can affect maintainability

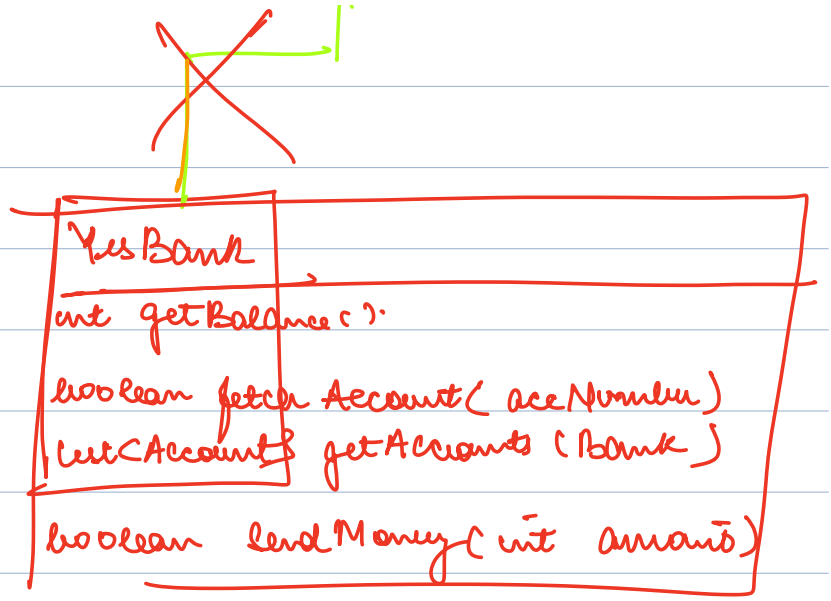
② Whenever you are connecting to a 3rd party API, never connect directly, always use an interface.



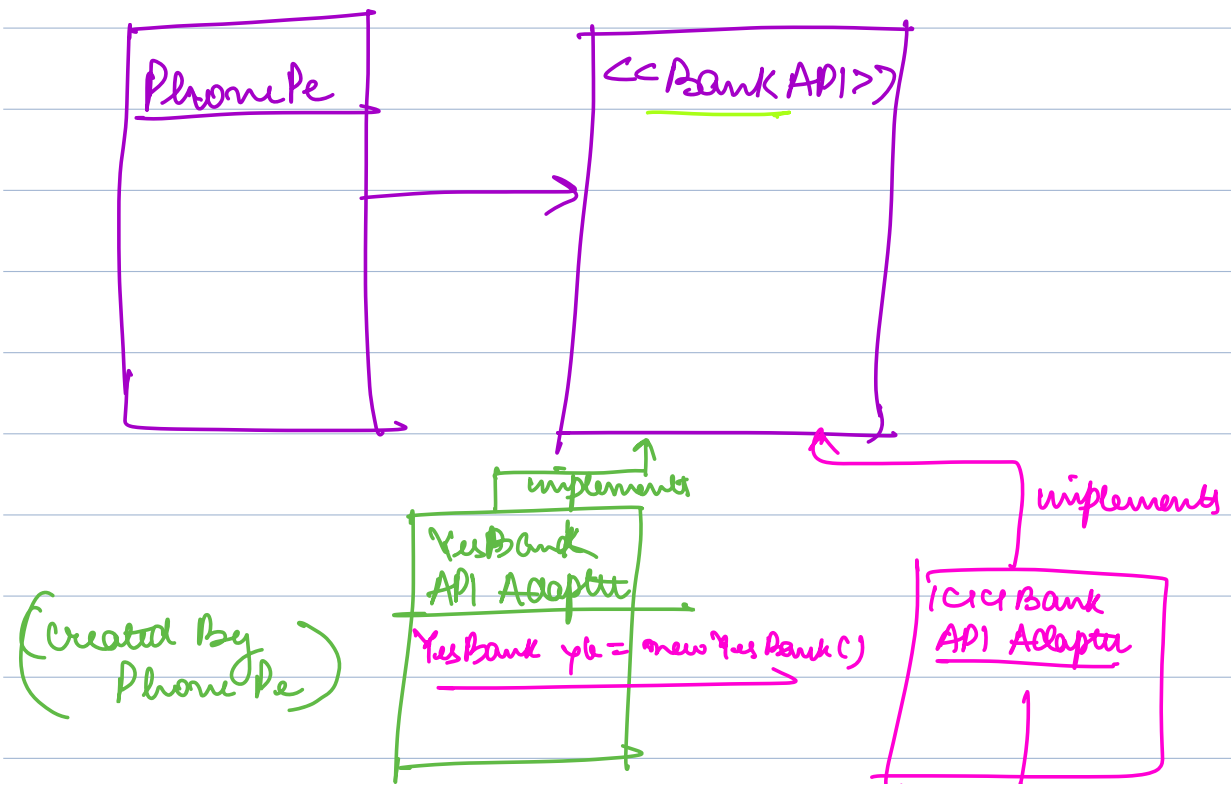
HOW TO USE ADAPTER

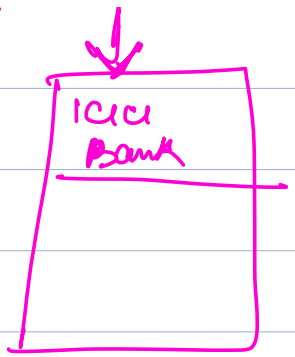
① Whenever you are connecting to any 3rd party API, create an interface with the methods that you need from the third party.





(2) As 3rd party will not implement this interface for us, we should create a wrapper class that implements our interface and uses 3rd party lib



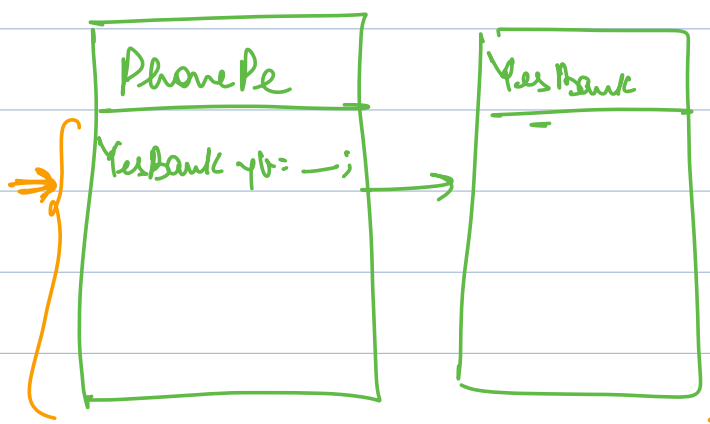


Adapter

When to consider using Adapter

Whenever talking to a 3rd Party API

Without Adapter



⇒ Violates

→ DIP

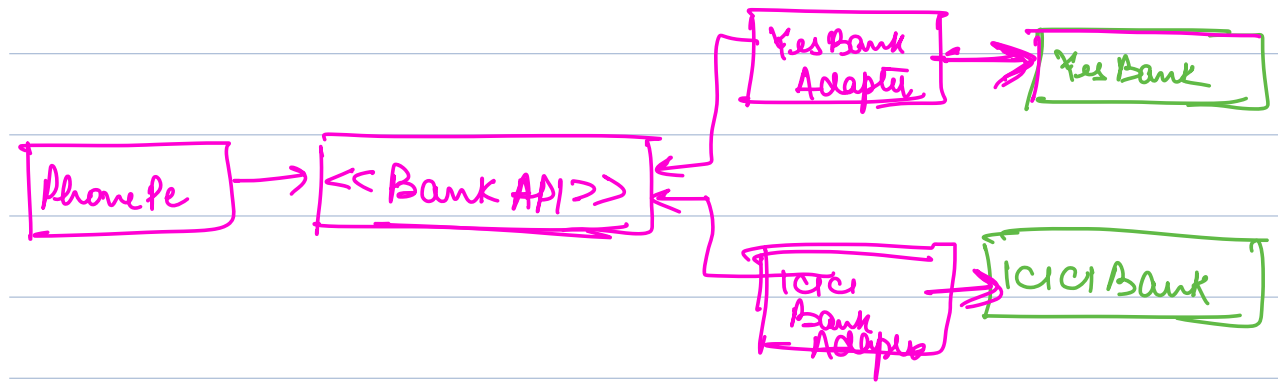
→ OCP

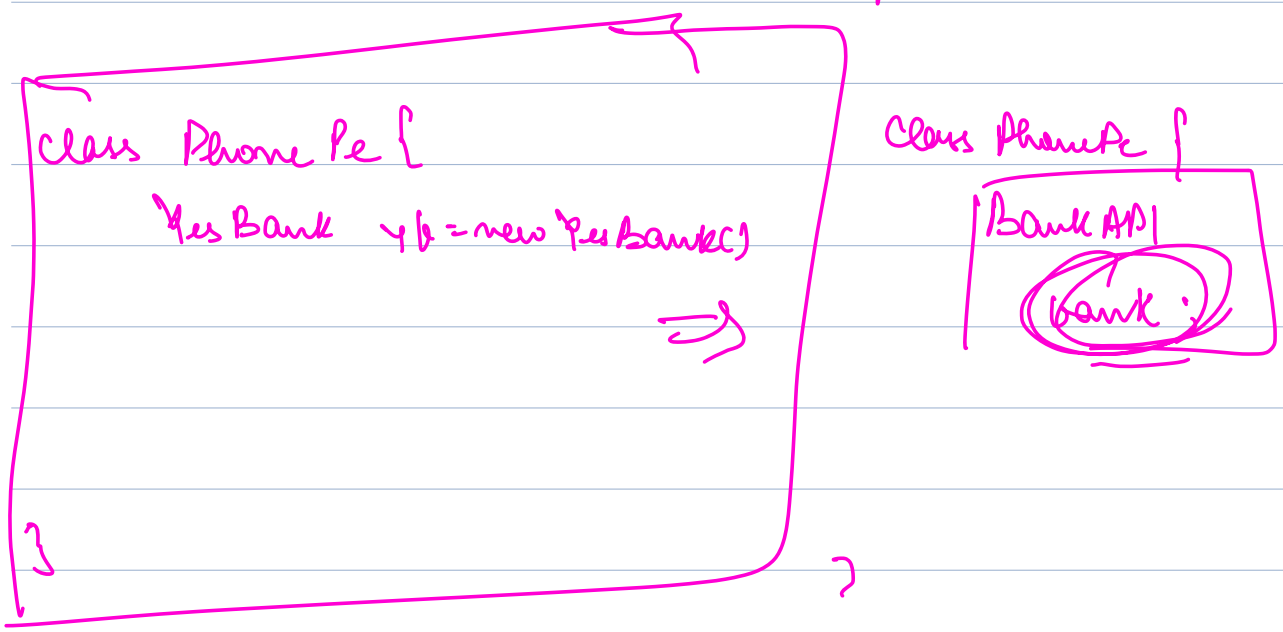
→ Problem Statement

→ a lot of changes in main code

→ Regression

With Adapter





Case Study

Calendly → application that allows users to create meeting scheduling links.

① See free slots on the calendar
② Automatically adds link to

a video conferencing platform

1:30 PM – 4:30 PM

Monday to Friday

Problem Statement

→ Identify what 3rd Party APIs will Calendly have to talk to

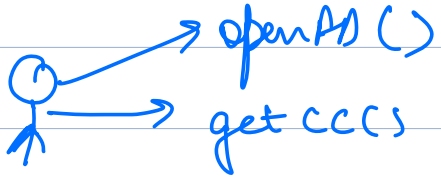
1. Zoom / Google Meet ⇒ To Create link
2. Google Cal / Microsoft Calendar ⇒ To see free slots.

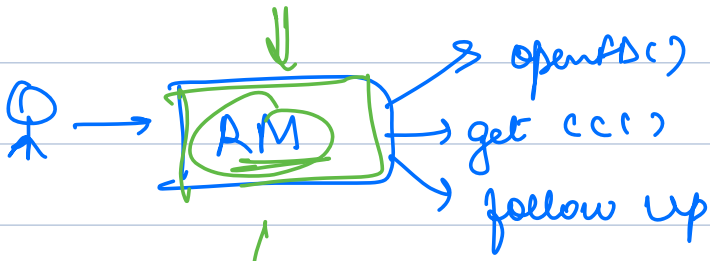
Cal.com ⇒ Open Source Calendly

Facade Design Pattern

↳ boundary / outside of a building

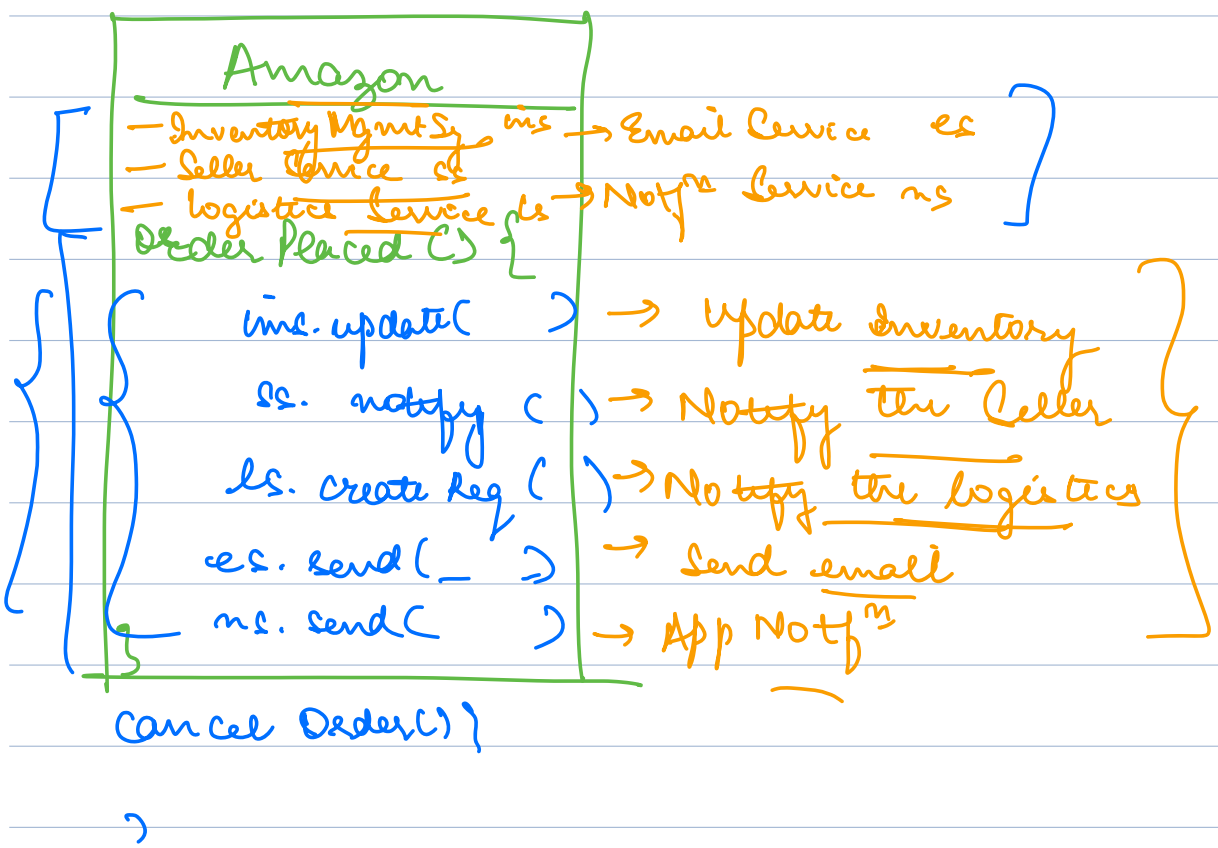
→ provides a cleaner / simplified view of a complex environment





↳ acting like a facade for me by simplifying the experience of a Bank

→ If RM was not there, I would have to do the work myself.

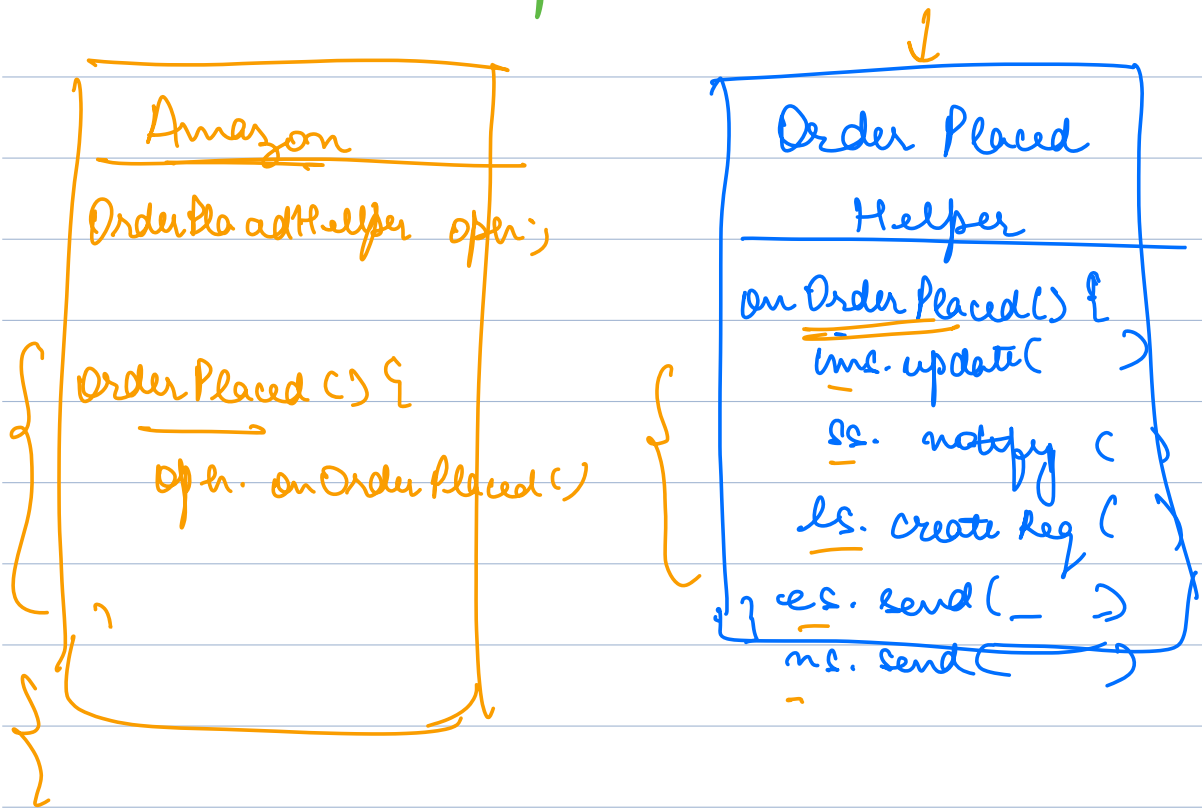


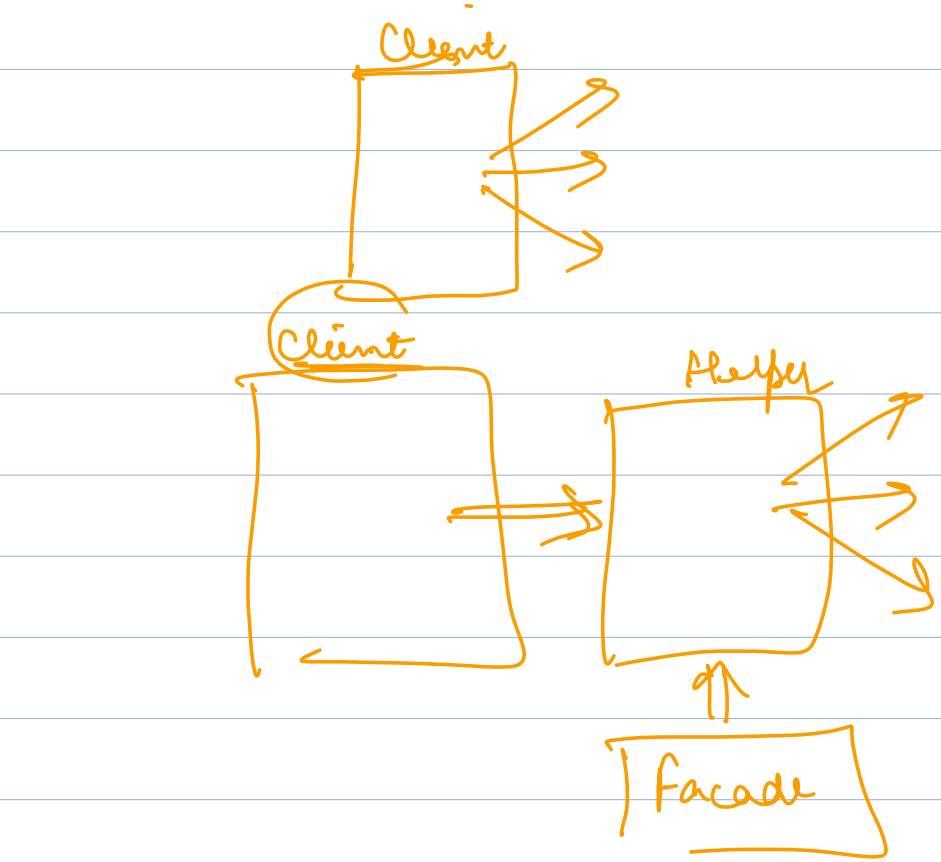
}

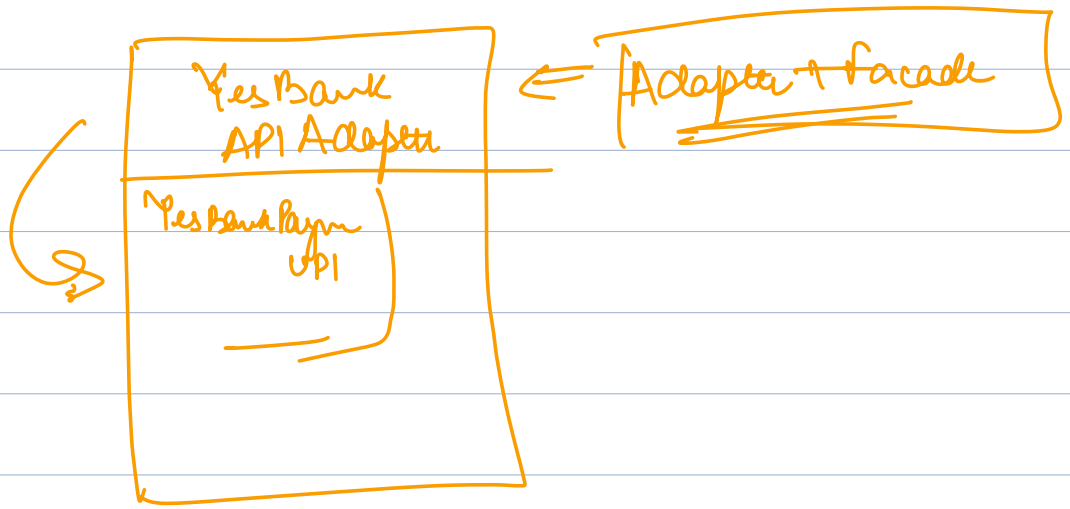
⇒ Currently all the work of what will happen when an order is placed is being done by Amazon class itself.

⇒ This might make Amazon class too big (linked with a lot of attributes)

Facade : whenever you find a method/class that is doing too much work, instead of doing the work within the class, create a helper to do that work




Client
Client
Helper
Facade


Yes Bank
API Adapter
Yes Bank Payment
API
Adapter + Facade

H/W

1. Video on facade <=
2. Read about Proxy and Bridge <=