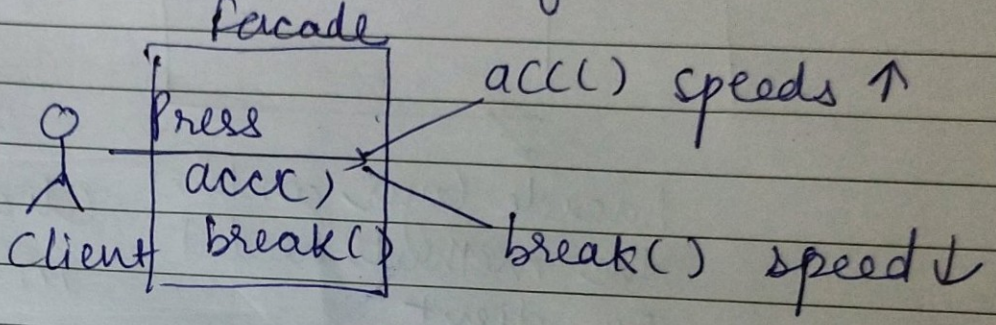
Facade Design Pattern

→ Simple
→ widely used.

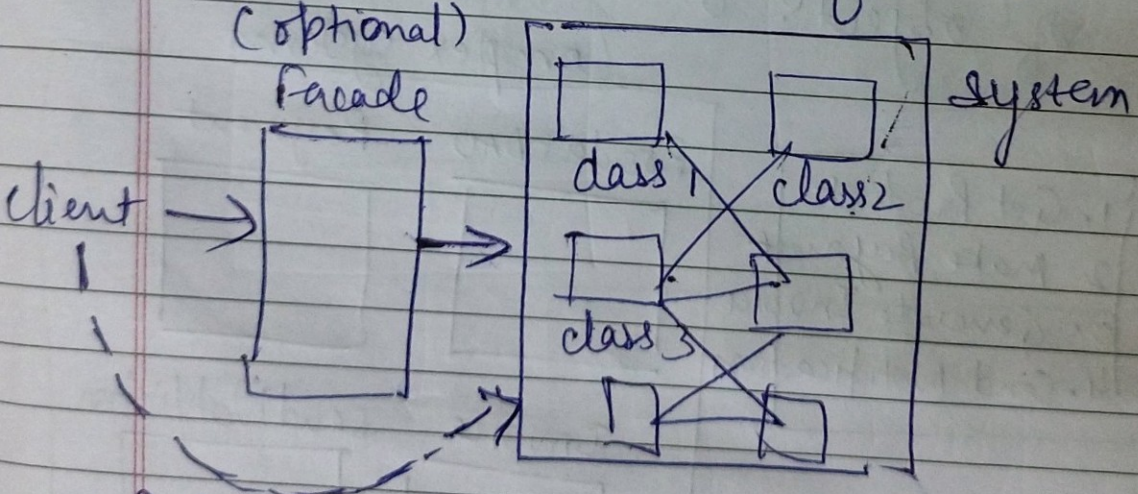
- Real World Usecases

Used :- When we have to hide the system Complexity from user then we use facade.

Ex: Car



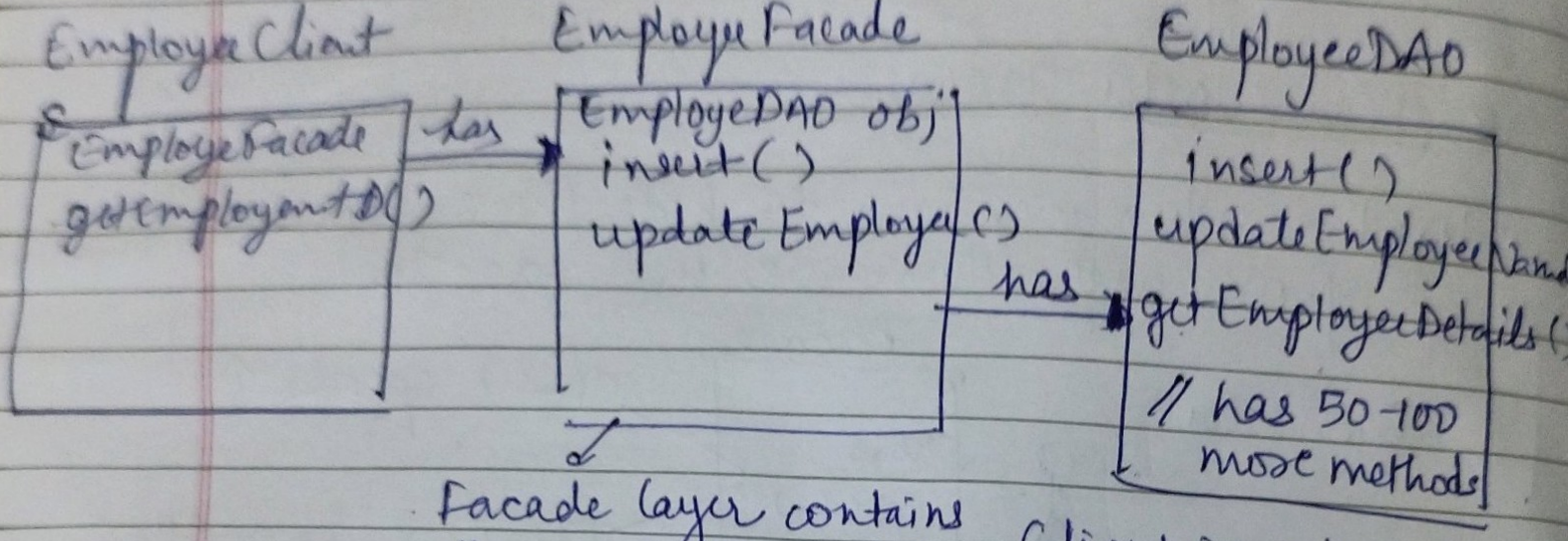
Here as a user only know the functionality but don't know the complexity. behind the acceleration and break function.



It's a client choice either to use facade or to use the classes directly.

Ex 2: DAO (Data Access Object)

Scenario 1:- Use Facade when you have to expose only certain methods to the client.



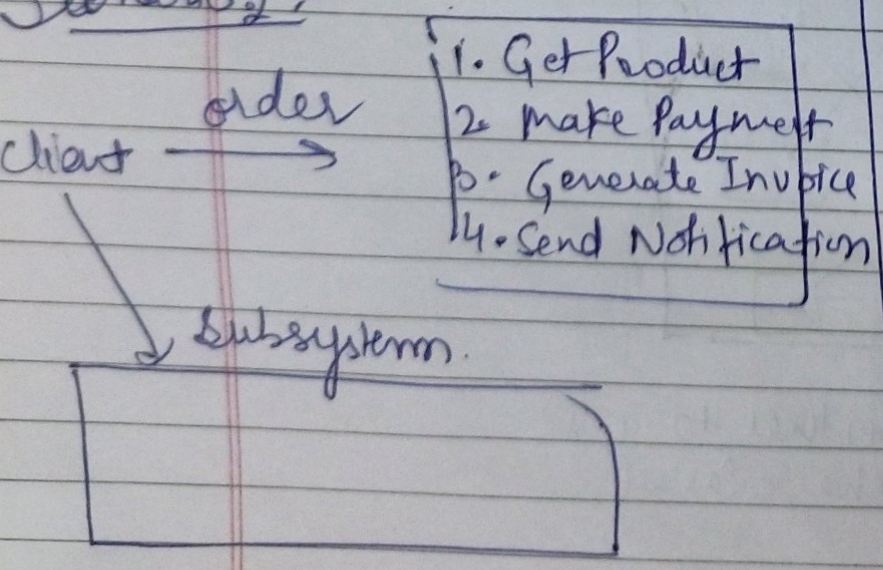
Facade layer contains the methods needed by client

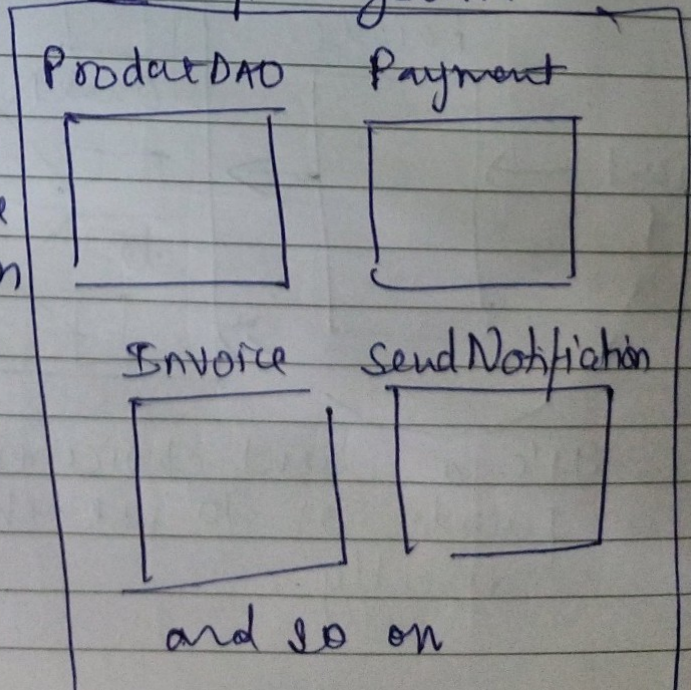
↓

Facade takes the responsibility of creation of object.

Client is only interested in insert and update method let say.

Scenario 2:-

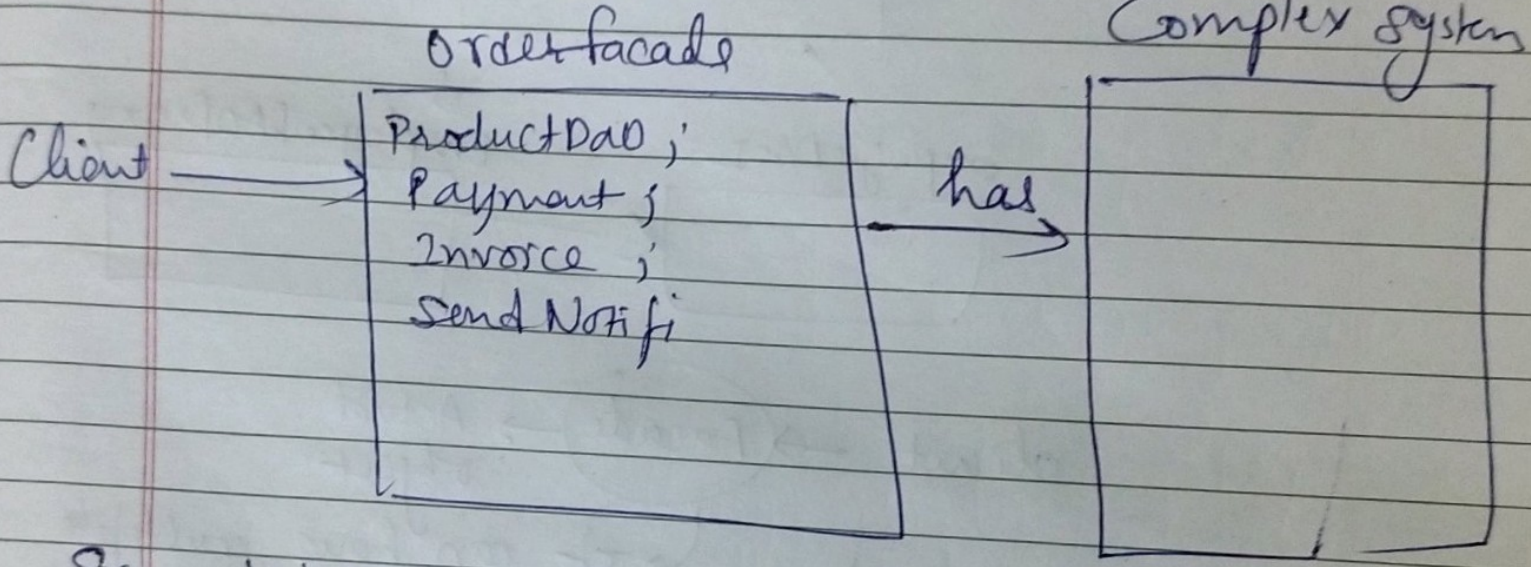




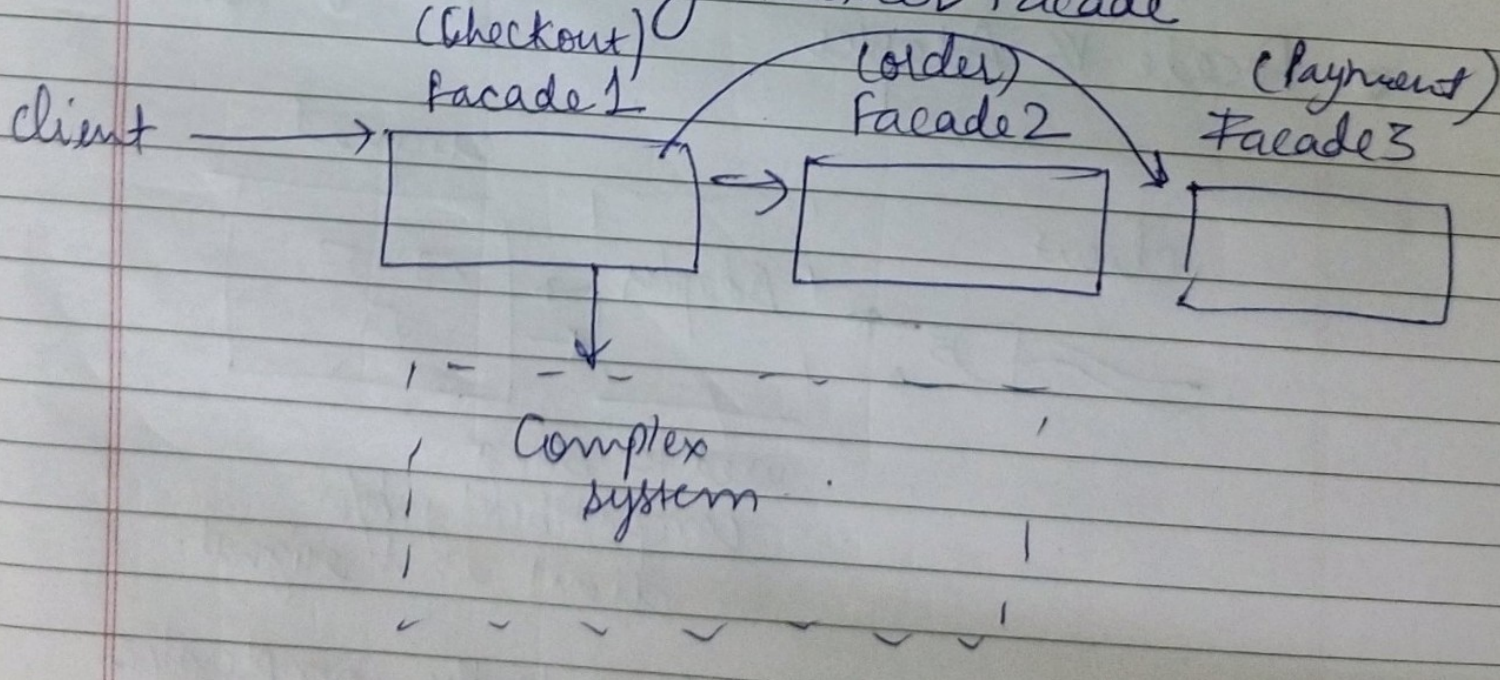
Problem: Any changes in the logic of these classes impacts client.

Ex) If the logic of order Creation changes then it will affect client (New step Added)

Solution: Bringing a facade layer.

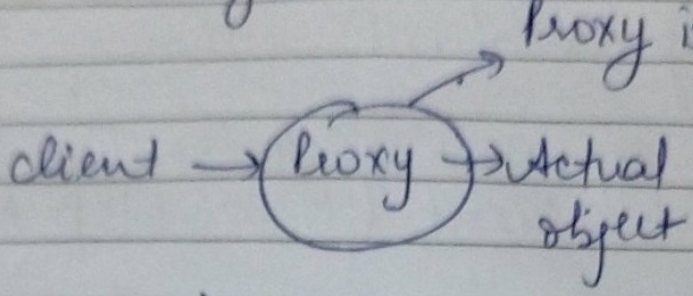


Scenario: Facade Using Another Facade



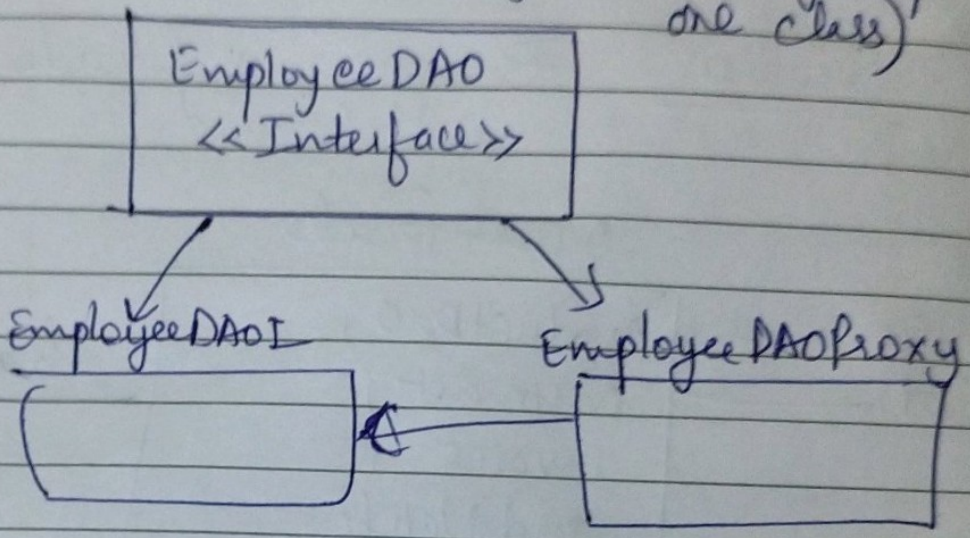
- Facade VS Proxy

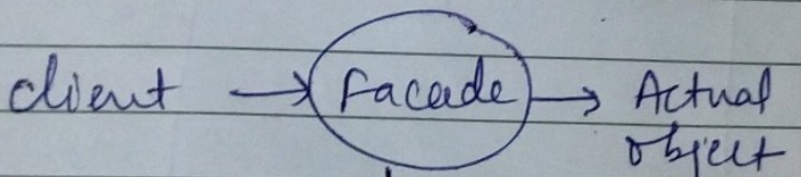
Proxy →



Proxy is same as object type

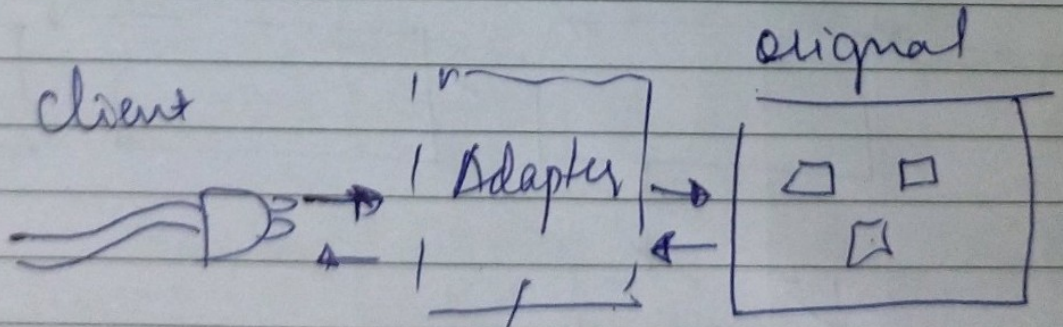
(Take cares of only a particular one class)





→ It can have multiple object of different type

- Facade VS Adapter



used to bring compatibility b/w client and original.

→ But Facade is used to hide the complexities