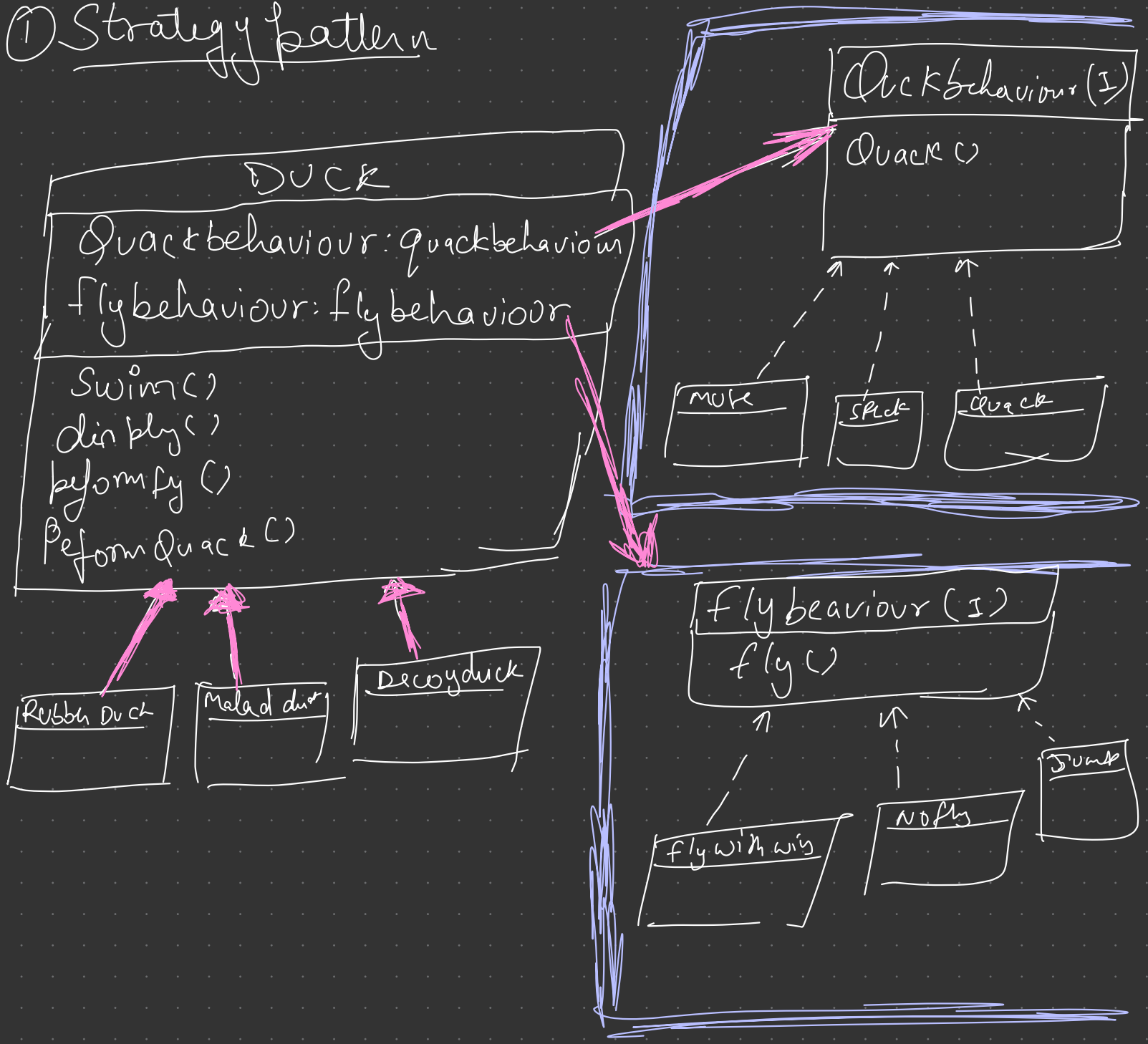
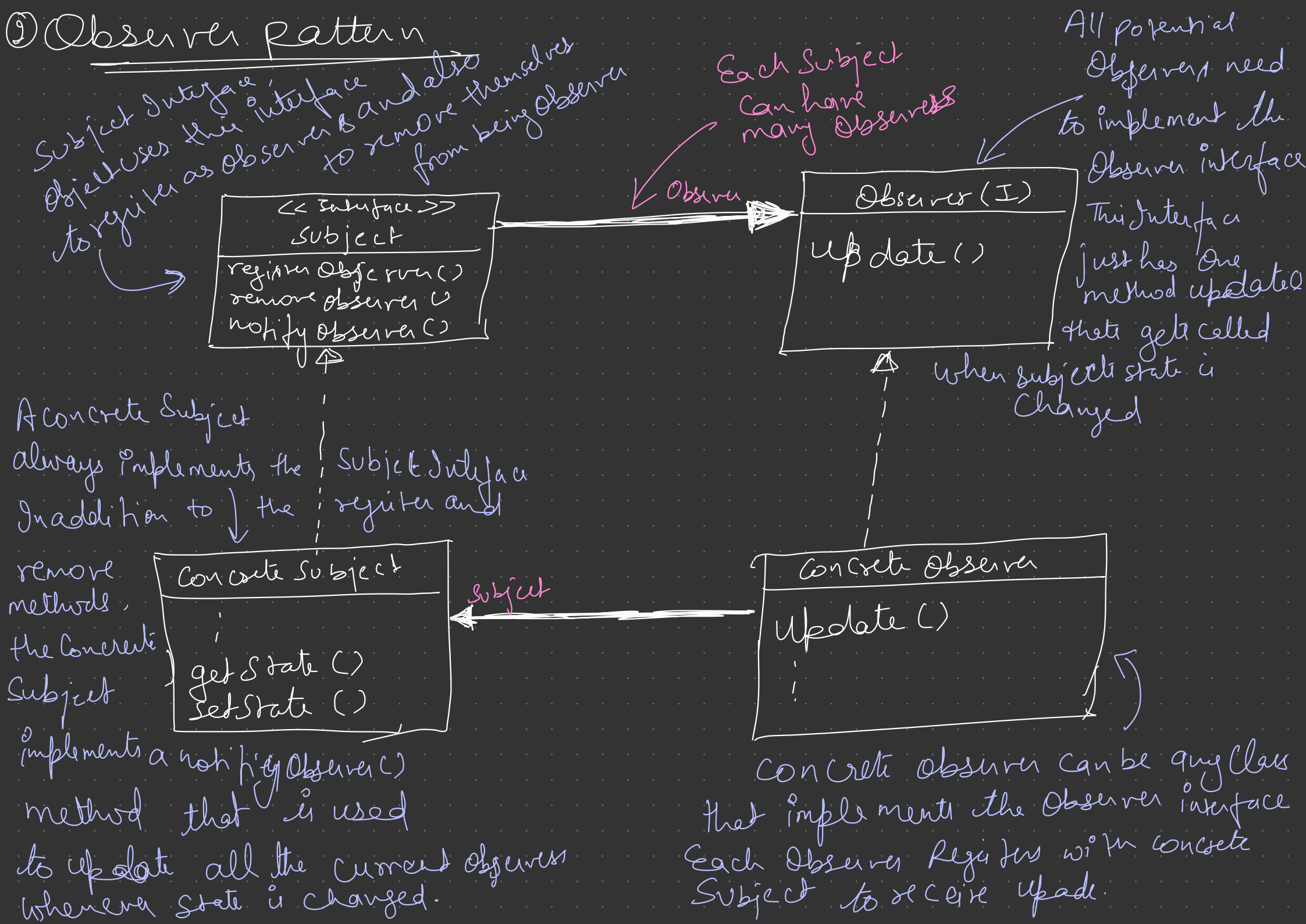
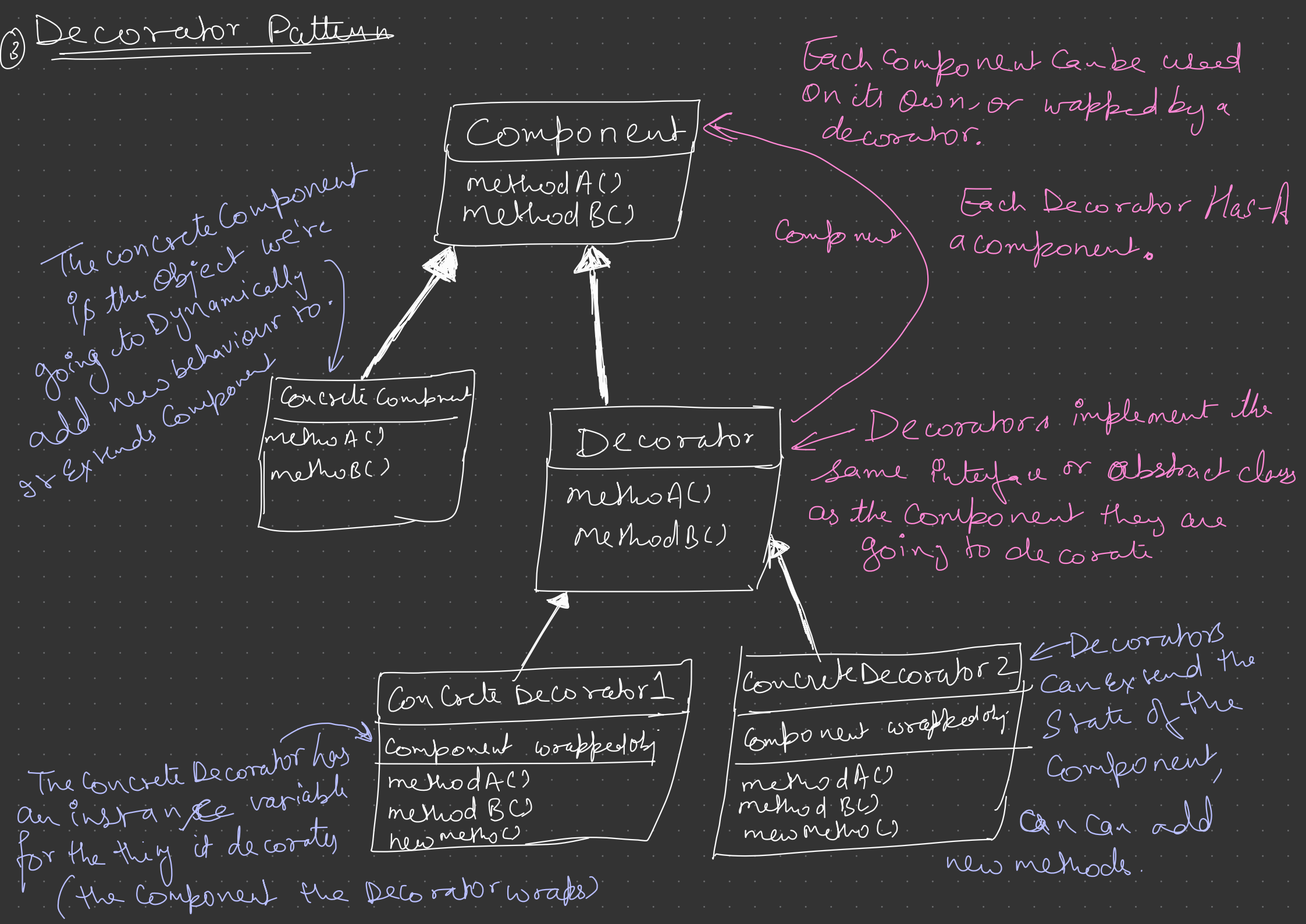
① Strategy pattern



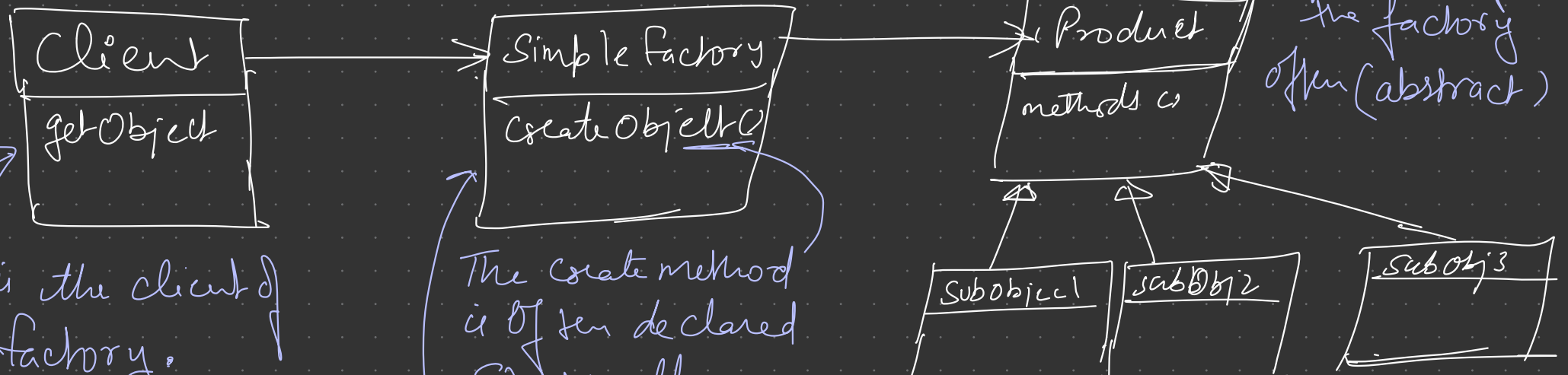
⑨ Observer Pattern



③ Decorator Pattern



① Simple factory



This is the main product of the factory often (abstract)

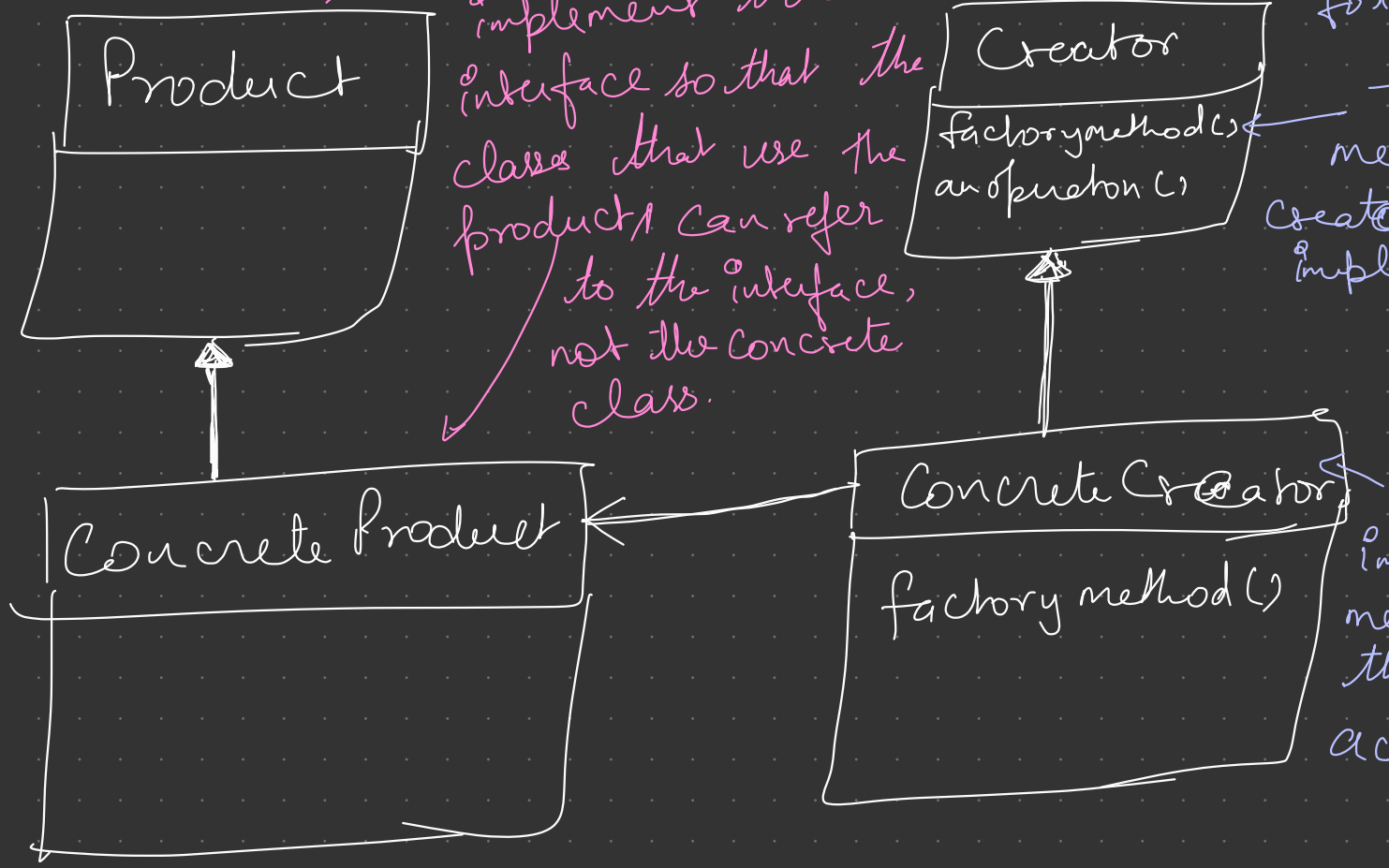
This is the client of the factory. factory will return the instance of Product.

The create method is often declared statically.

This is the factory where we create product; it should be the only part of the application that refers to concrete product.

These are the concrete products.

③ Factory Method pattern



All products must implement the same interface so that the classes that use the products can refer to the interface, not the concrete class.

The creator is a class that contains the implementation for all the methods to manipulate products, except for the factory method.

The abstract factory method() is what all creator subclass must implement.

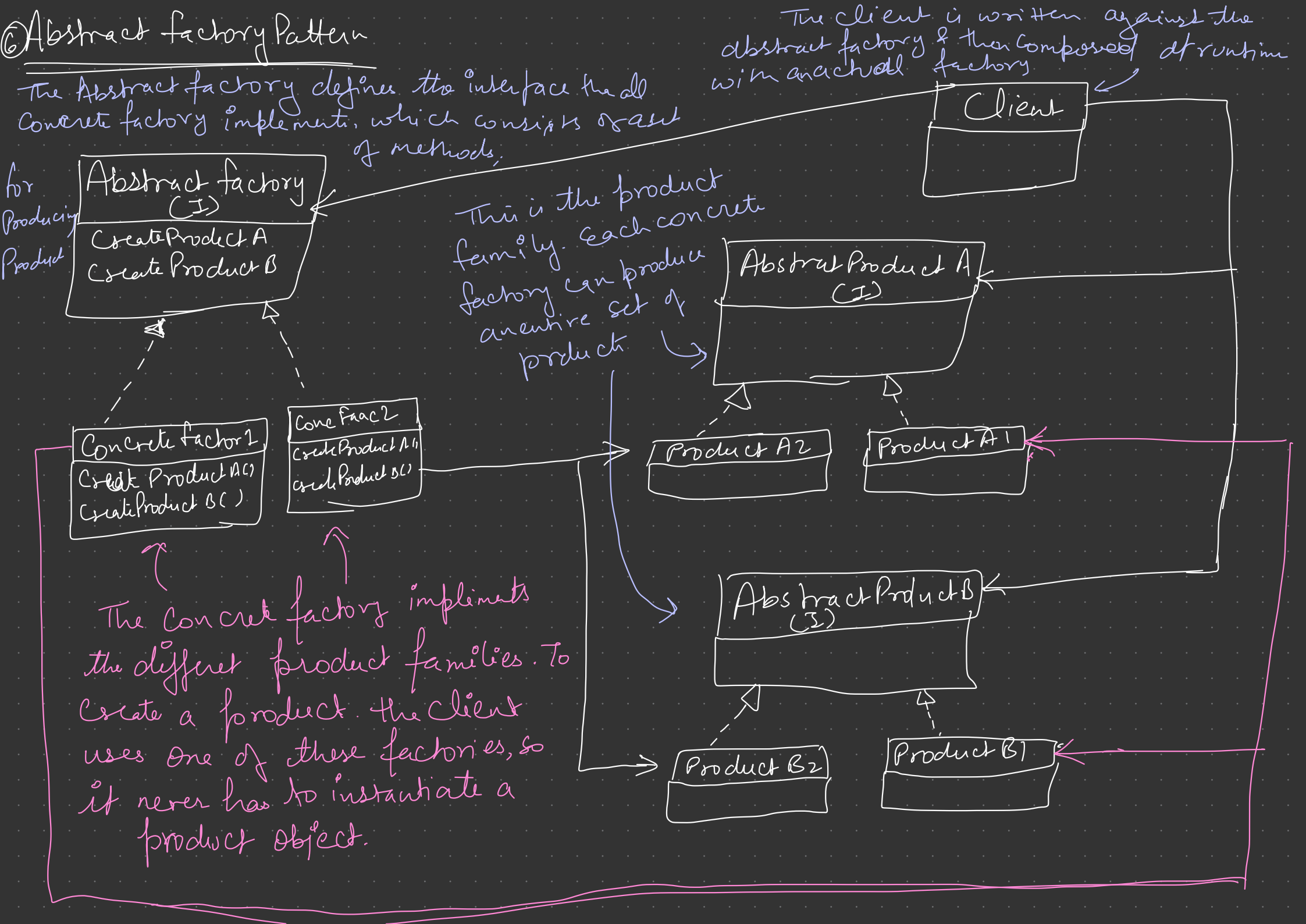
The concrete creator implements the factory method(), which is the method that actually produces products.

The Concrete Creator is responsible for creating one or more concrete products. It is the only class that has the knowledge of how to create Products.

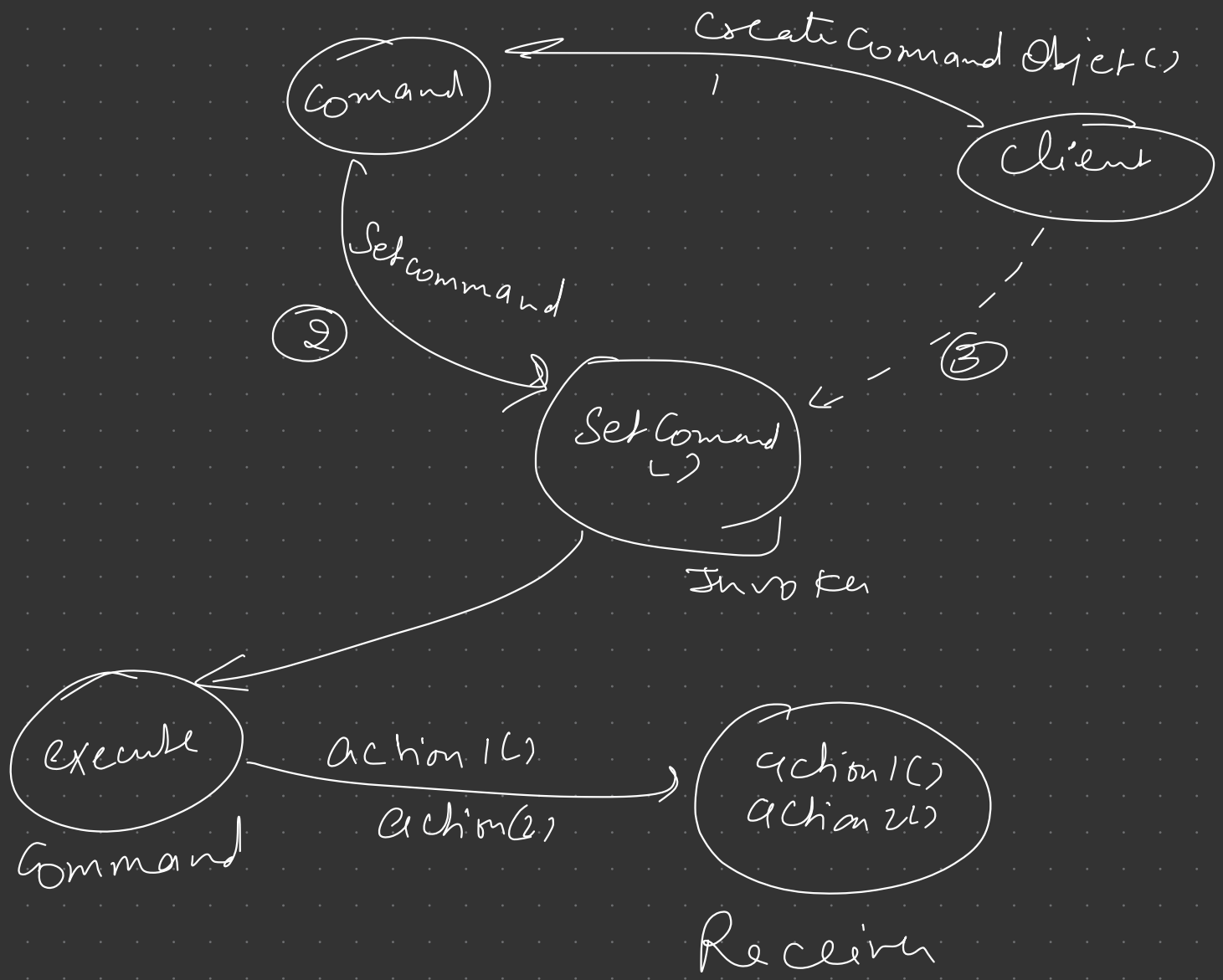
Abstract factory Pattern

The Abstract factory defines the interface that all Concrete factory implements, which consists of a set of methods.

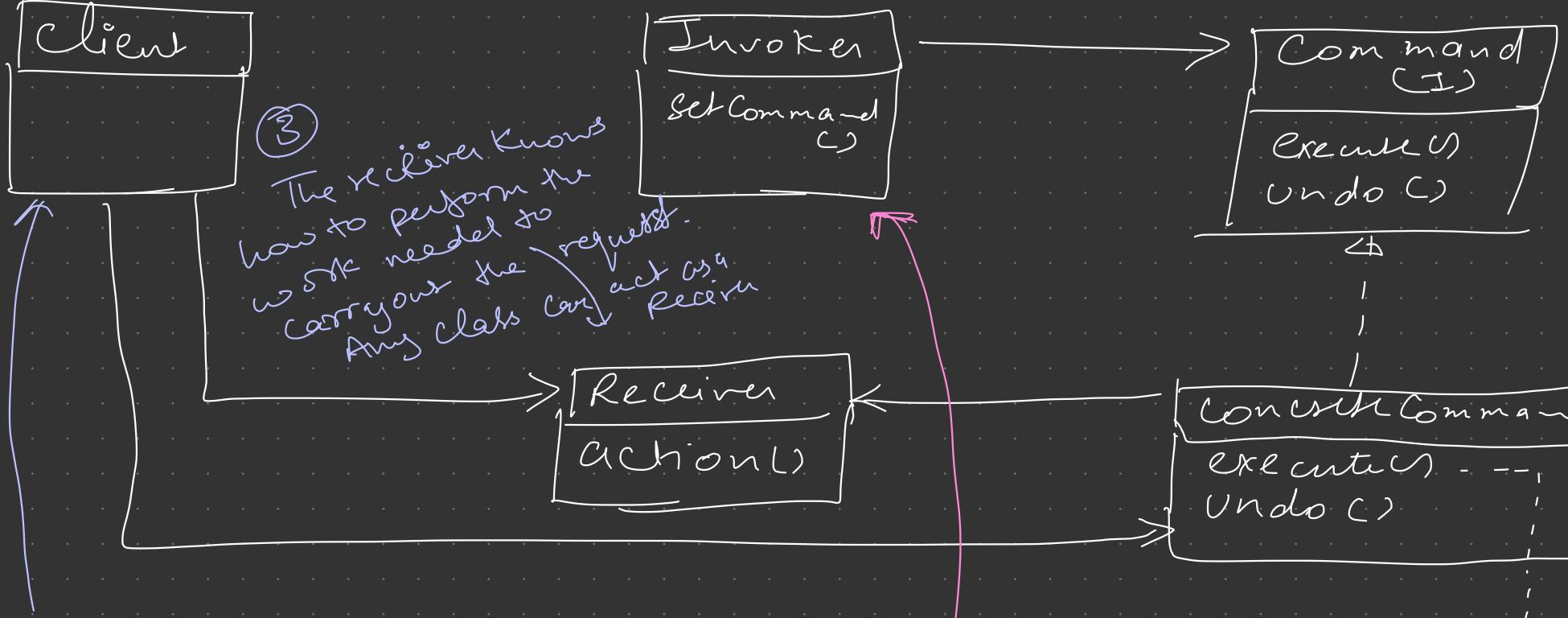
The client is written against the abstract factory & then composed at runtime with an actual factory



① Command pattern flow



① Command Pattern



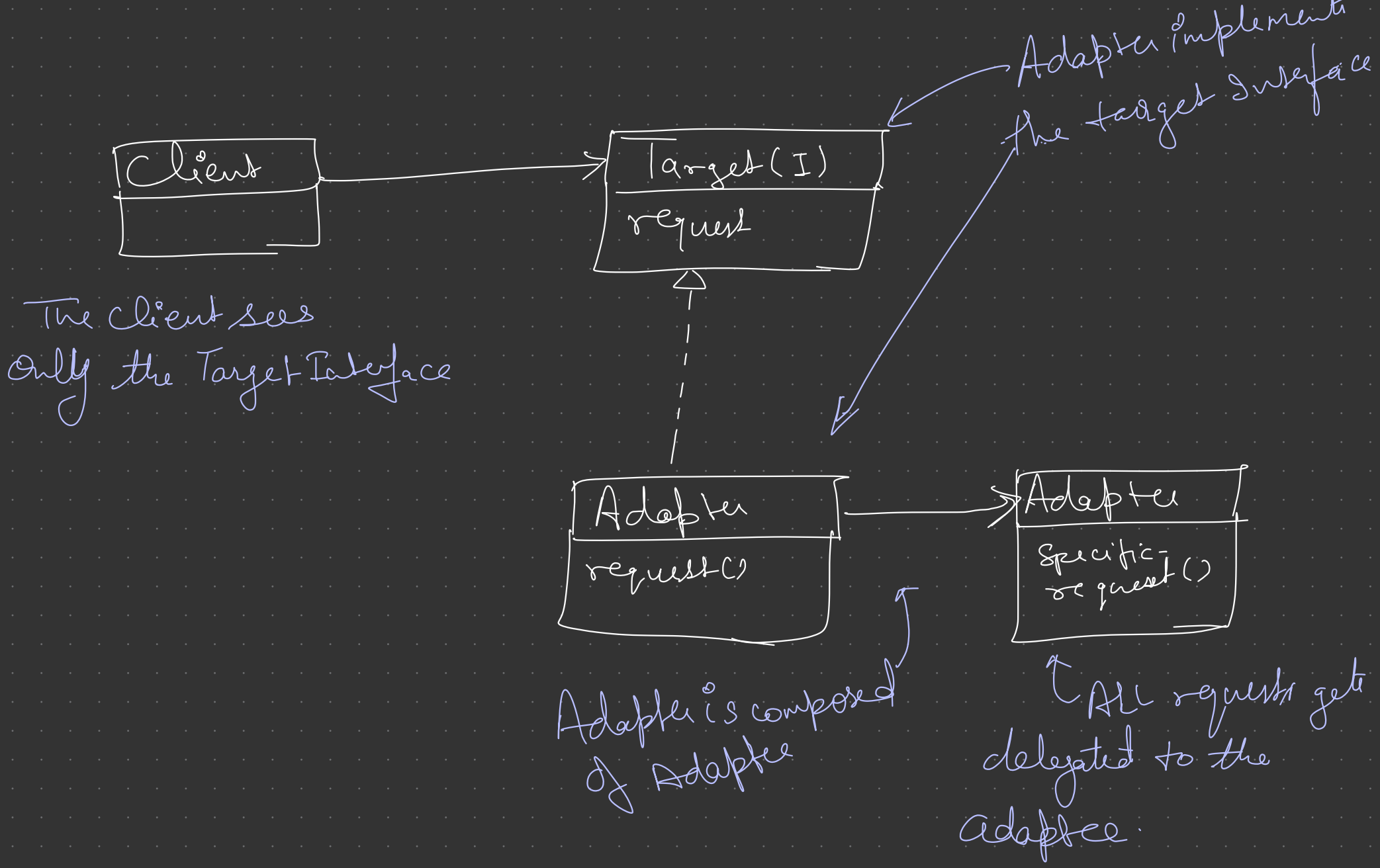
① Client is responsible for creating a ConcreteCommand and setting its Receiver

② The Invoker holds the command & at some point asks the command to carry out a request by calling its Execute() method

④ Command declares an interface for all commands.

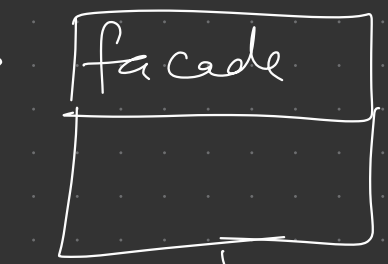
⑤ The Concrete Command defines a binding b/w an action and a Receiver.
The invoker makes request by calling Execute().

Adapter Pattern



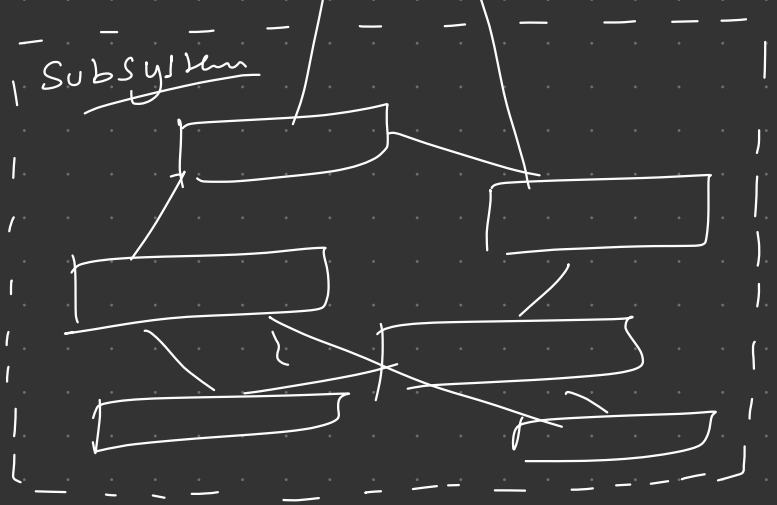
@Facade Pattern

Happy client whose job is just became easier because of facade



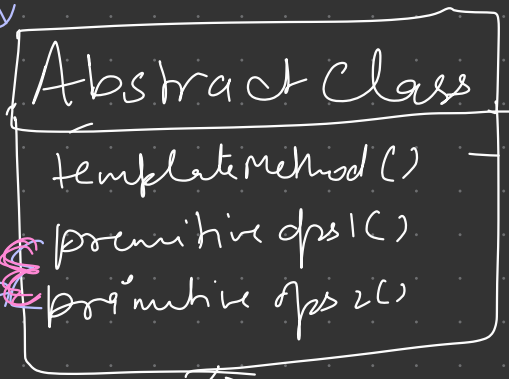
Unified interface that is easier to use

more complex Subsystem.

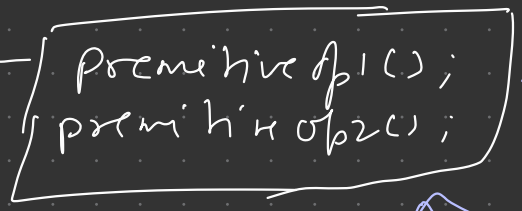


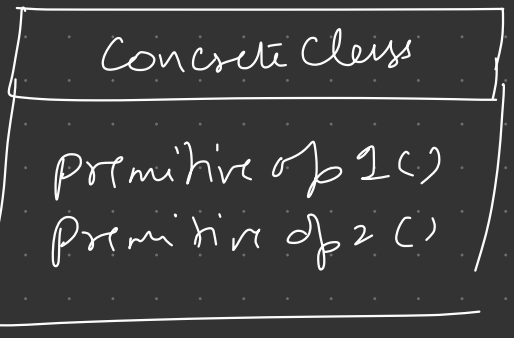
(10) # Template Method

The abstract class contains the template method ... And abstract versions of the opn used in the template method()



The template method makes use of the primitive operations to implement an algo. It is decoupled from the actual implementation of these operations



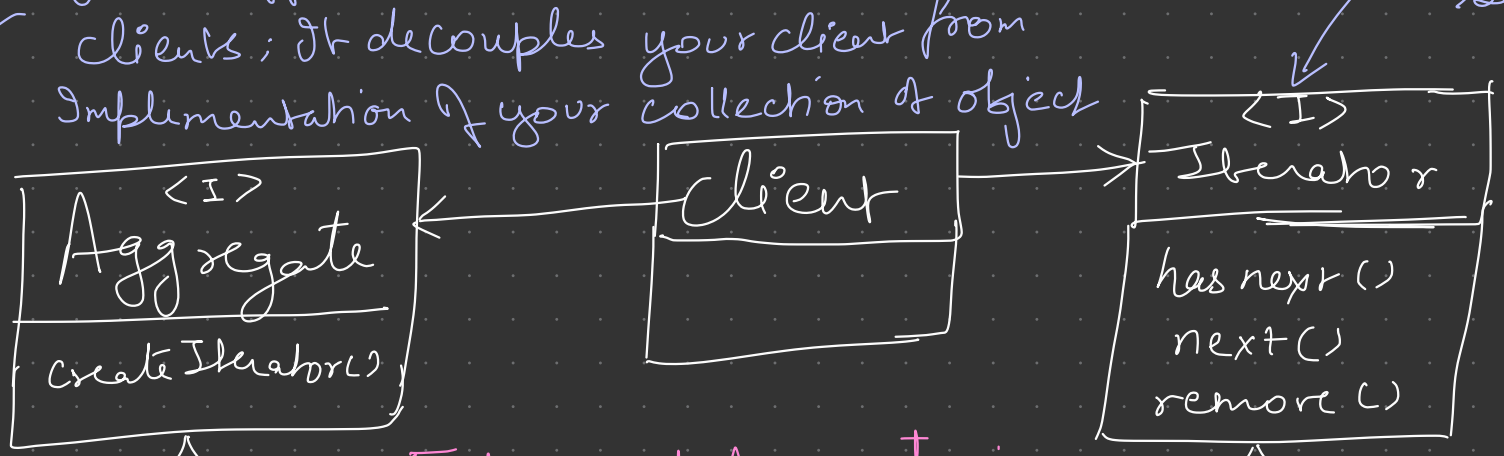


The concrete class implements the abstract operations, which are called when the templateMethod() needs them

There may be many concrete classes, each implementing the full set of operations required by the template method

(11) Iterator Pattern

Having a common Interface for your aggregates is handy for your clients; it decouples your client from implementation of your collection of object



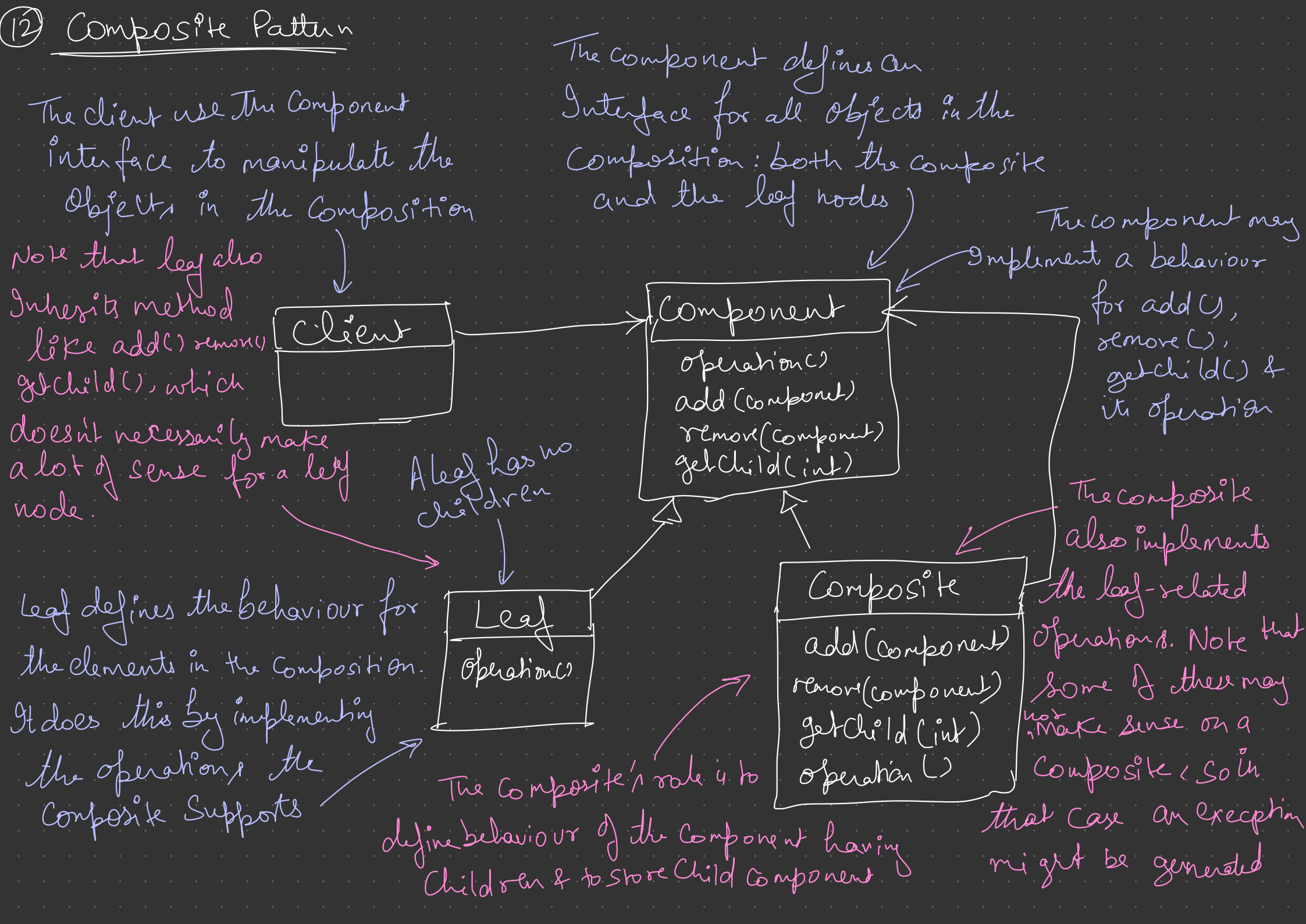
The Iterator Interface provides the interface that all Iterators must Implement, and a set of methods for traversing over elements of a collection.

Each concrete Aggregate is responsible for instantiating a concrete Iterator, that can Iterate over its collection of objects

The concrete Iterator is responsible for managing the current position of the iteration

The concrete Aggregate has a collection of objects and implements the method that returns an Iterator for its collection

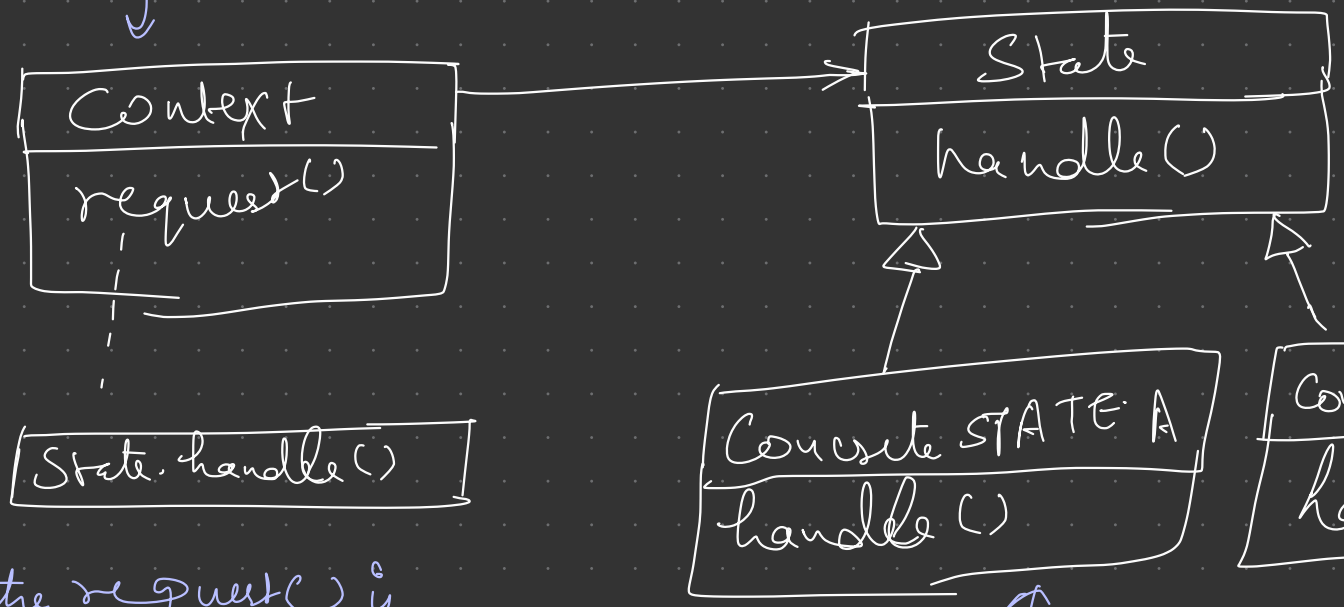
(12) Composite Pattern



(13) State Pattern

The context is the class that can have number of internal states.

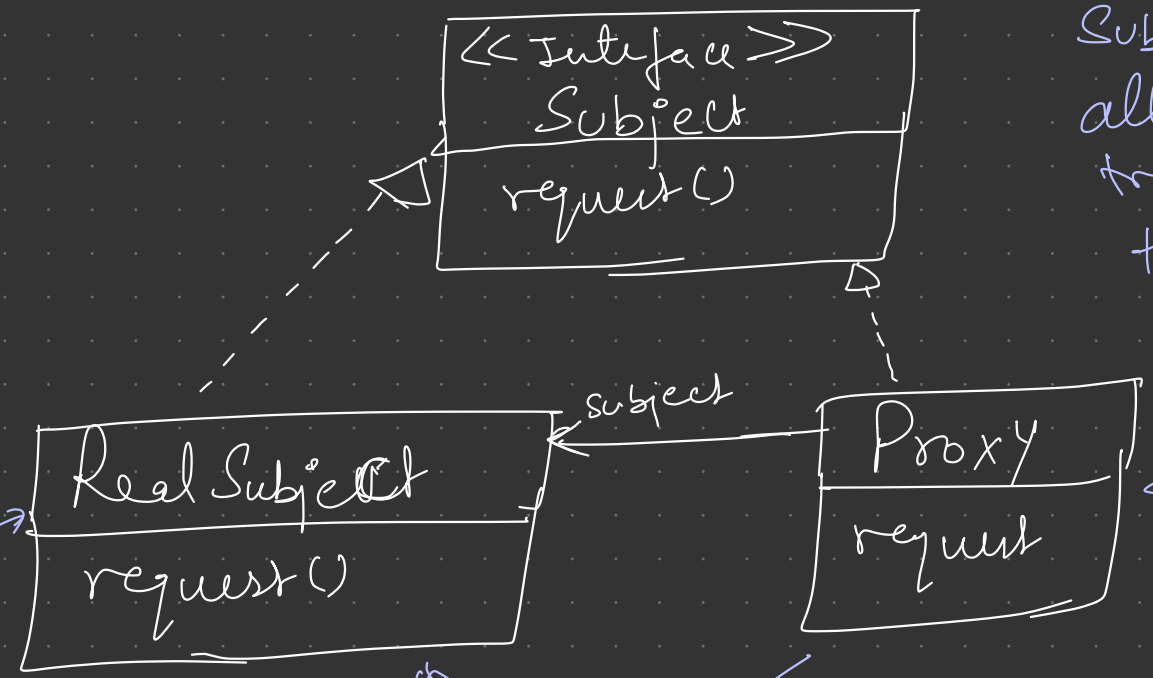
The State interface defines a common interface for all concrete States; The states all implement the same interface so they are interchangable



Whenever the request() is made on the Context it is delegated to the State to handle

Concrete States handle request from the Context. Each Concrete state Provides its own implementation for a request. In this way, when the Context changes state, its behaviour will change as well.

(14) Proxy Pattern



Both the proxy and the RealSubject implement the Subject interface. This allows any client to treat proxy just like the Real Subject

The RealSubject is usually the object that does most of the real work; the proxy controls the access to it

The proxy often instantiates or handles the creation of the RealSubject

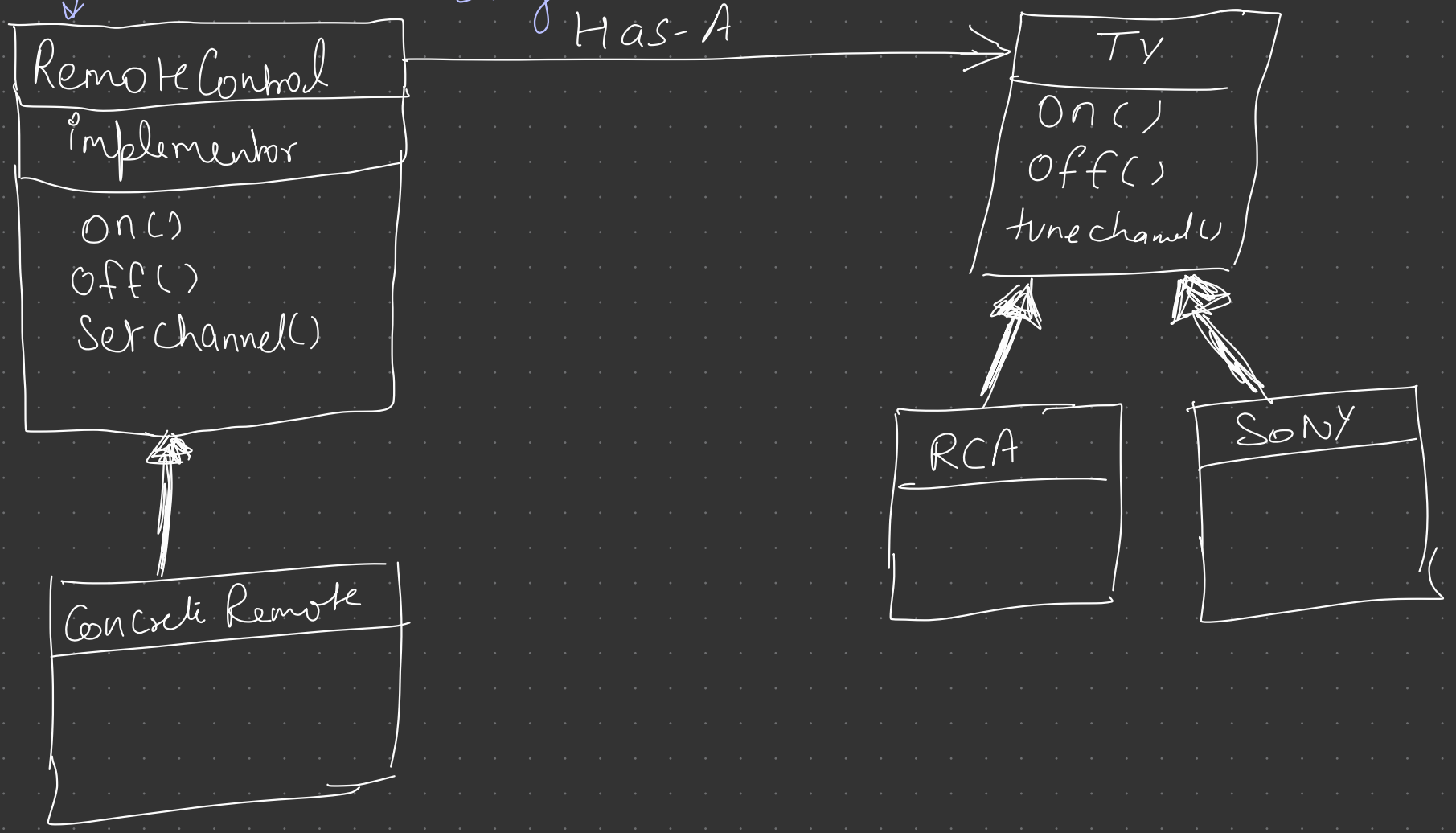
The proxy keeps a reference to the Subject, so it can forward requests to the Subject when necessary.

Bridge Pattern

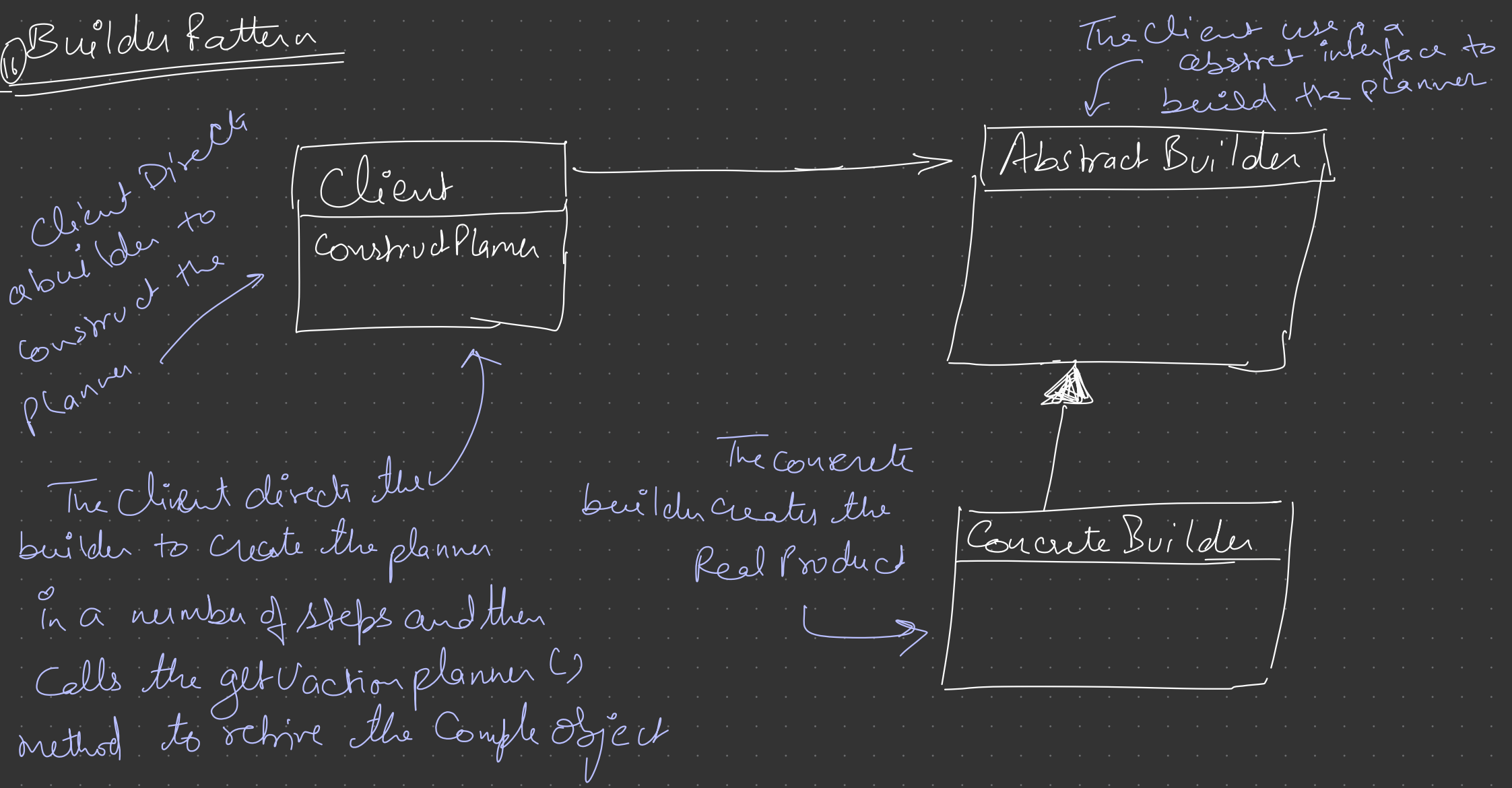
Abstraction class heirarchy

The relationship b/thwn the two is reffered to as Bridge

Implementor class heirarchy

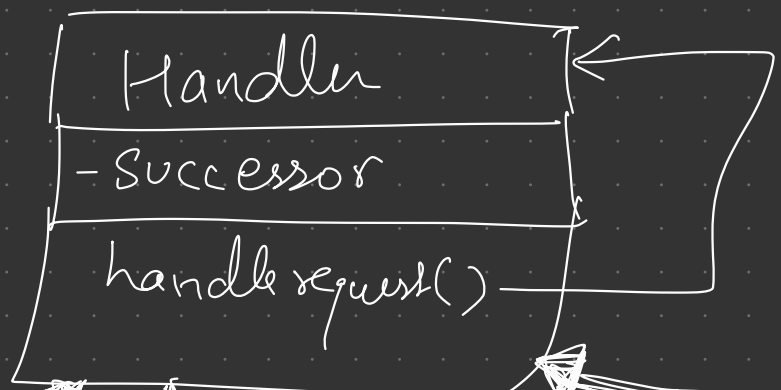


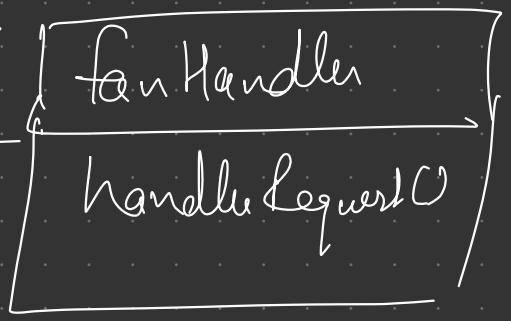
Builder Pattern



(17) Chain of Responsibility

Each object in the chain acts as a handler and has a successor object. If it can handle the request, it does otherwise it forwards the request to the successor





Each email is passed to first handler

Email is not handled if it falls off the end of the chain - although you can always implement a catch all handler