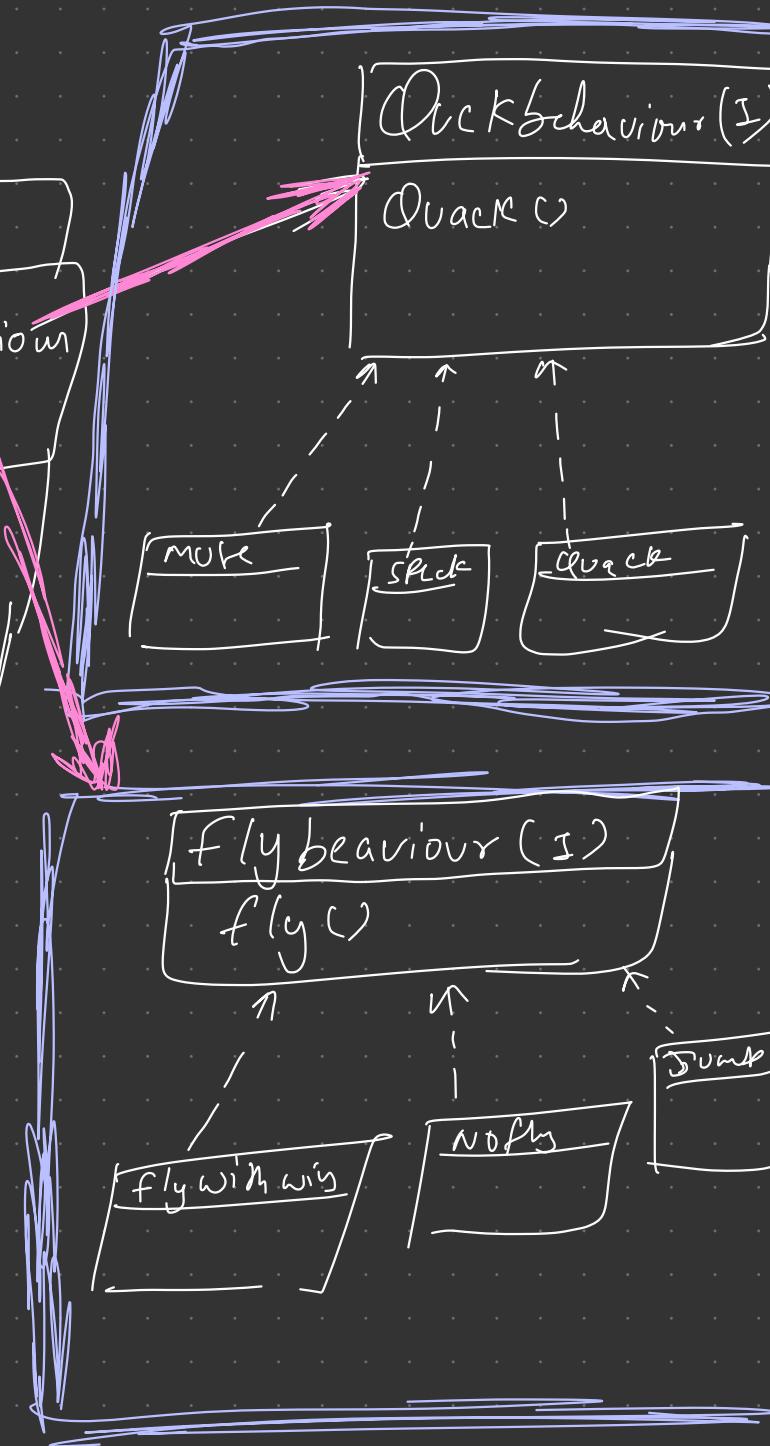
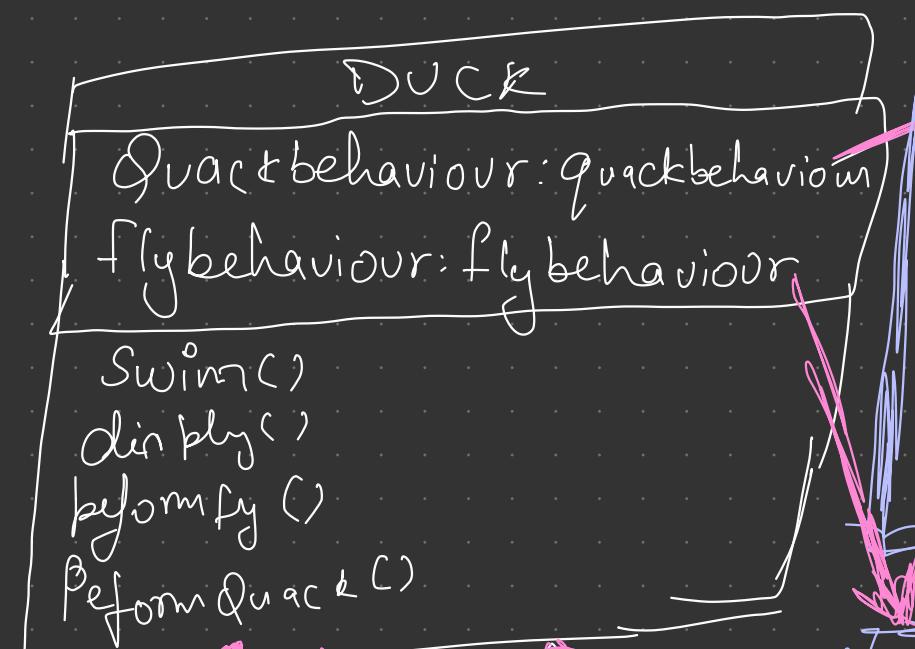


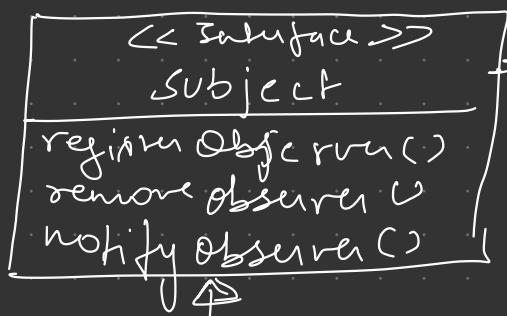


① Strategy pattern



⑨ Observer Pattern

Subject Interface
Object uses the interface
to register as observer and also
to remove themselves
from being observer

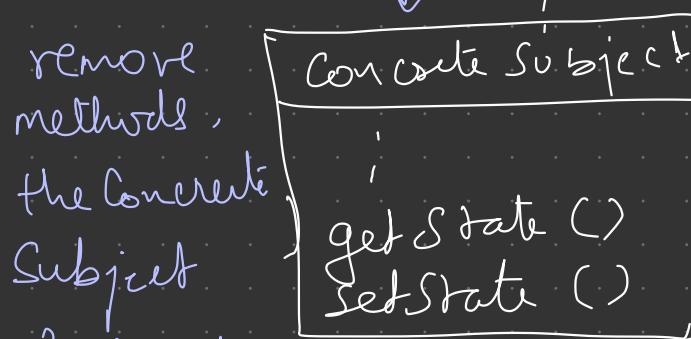


Each Subject
can have
many Observers



All potential
Observers need
to implement the
Observer interface
This interface
just has one
method update
that gets called
when subject state is
changed

A concrete Subject
always implements the Subject Interface
In addition to the register and



implements a notifyingObserver()
method that is used
to update all the current observers
whenever state is changed.



concrete observer can be any class
that implements the Observer interface
Each Observer registers with concrete
Subject to receive update.

③ Decorator Pattern

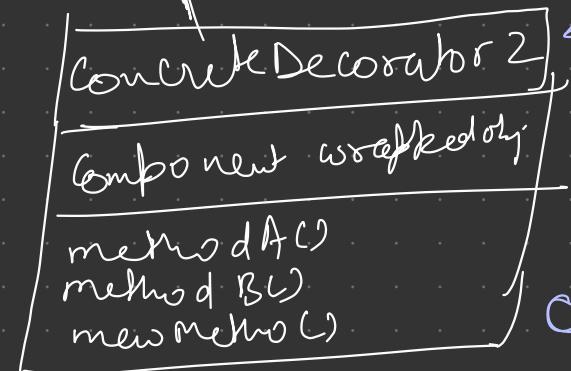
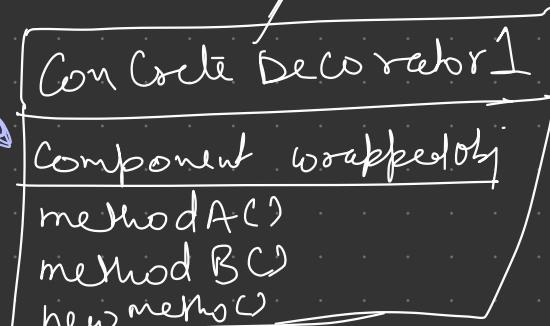
The concrete component is the object we're going to dynamically add new behaviour to.
It Extends Component



Each component can be used on its own, or wrapped by a decorator.

Each Decorator Has-A component.

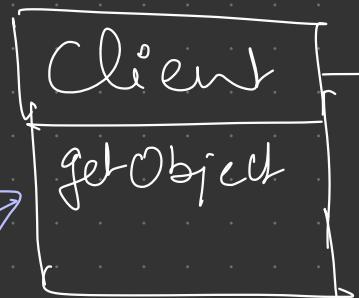
Decorators implement the same interface or abstract class as the component they are going to decorate.



Decorators can extend the state of the component, can add new methods.

The concrete decorator has an instance variable for the component it decorates
(the component the decorator works)

① Simple factory

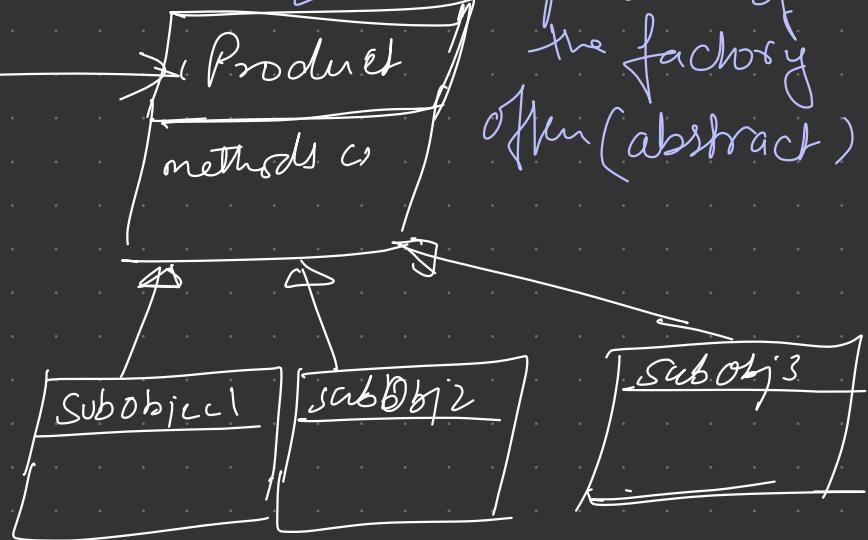


This is the client of the factory, factory will return the instance of Product.



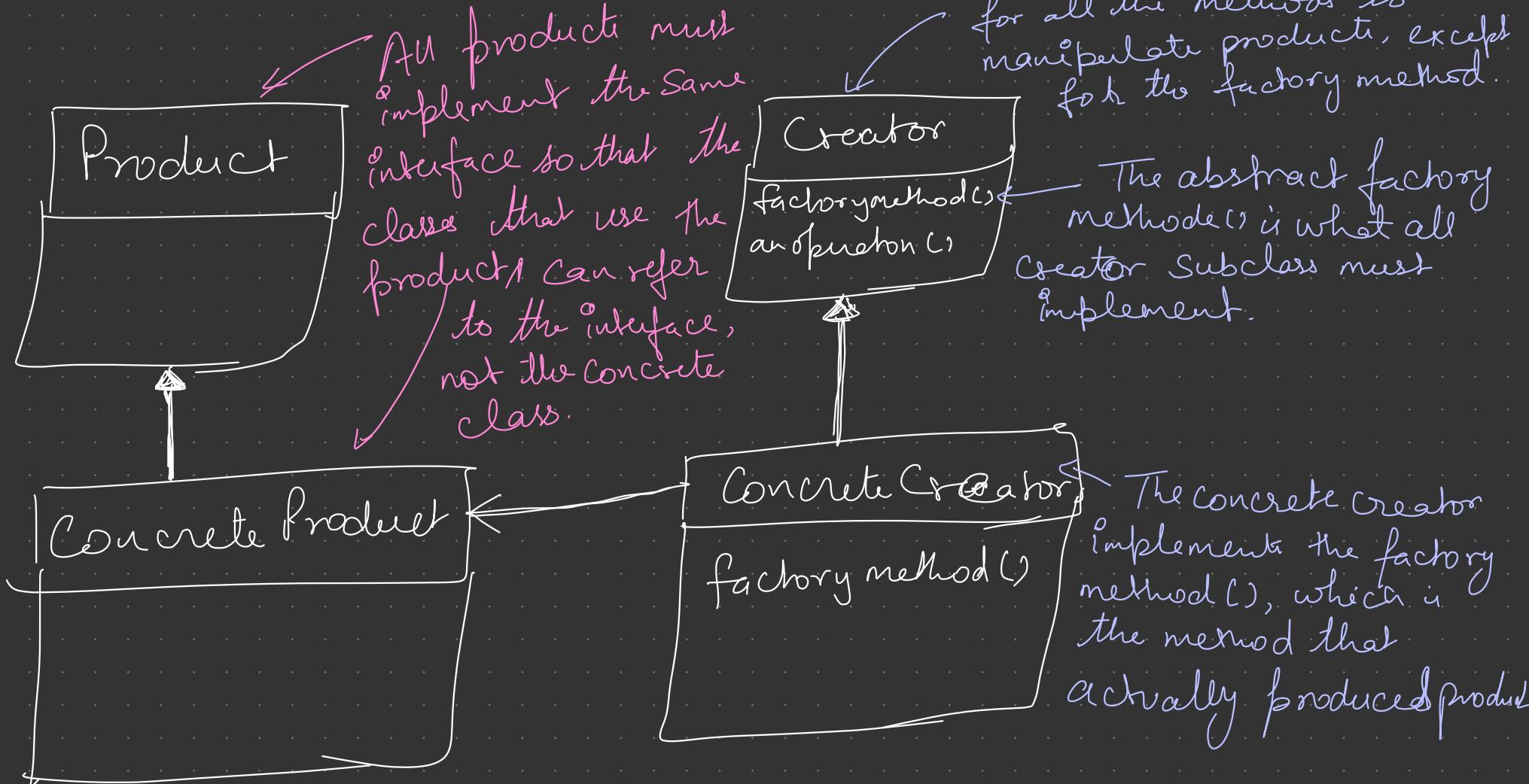
The Create method is often declared statically.

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



These are the concrete products.

⑤ Factory Method pattern



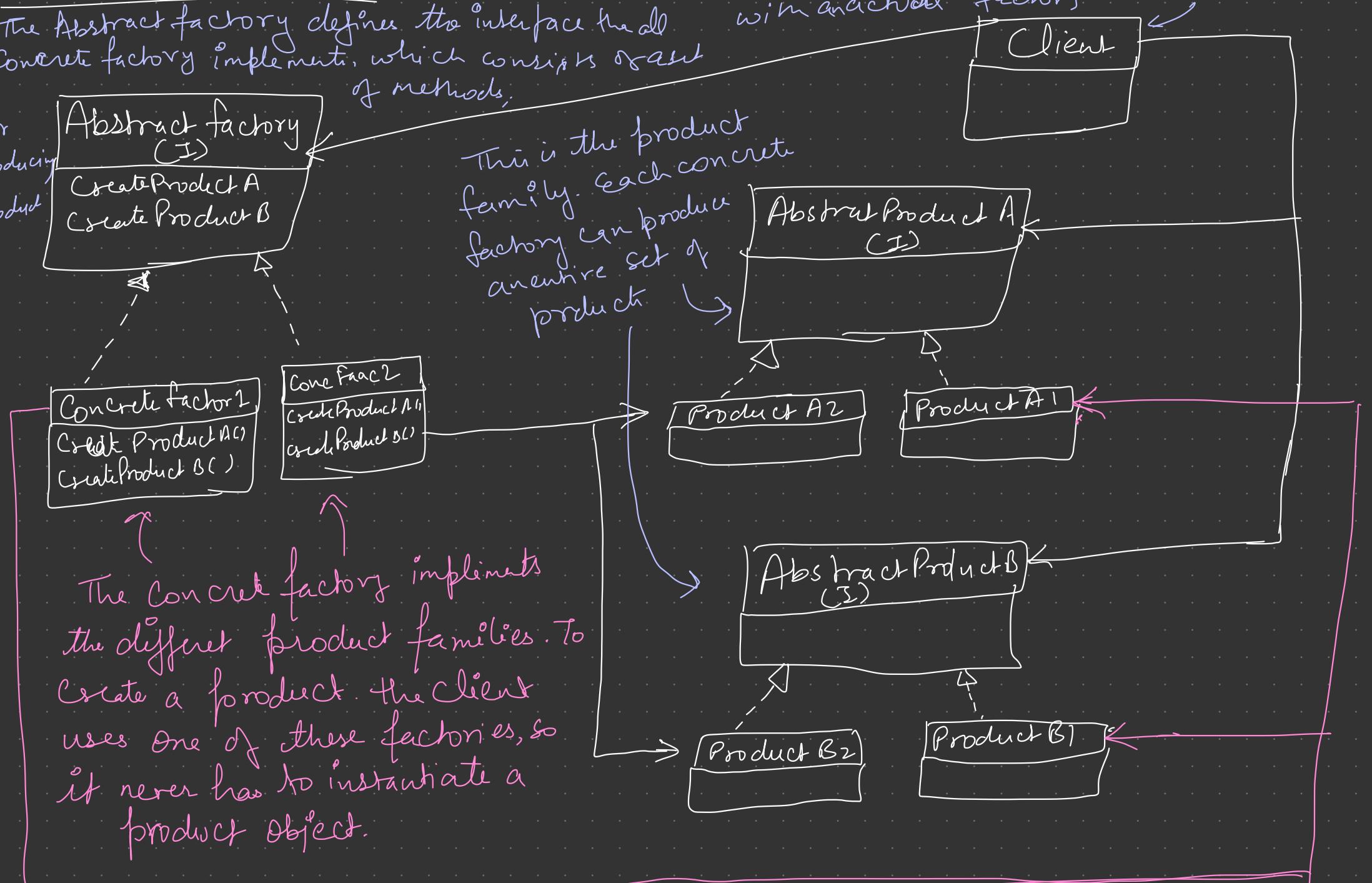
The Concrete Creator is responsible for creating one or more concrete products. It is the only class that has the knowledge of how to create products.

⑥ Abstract factory Pattern

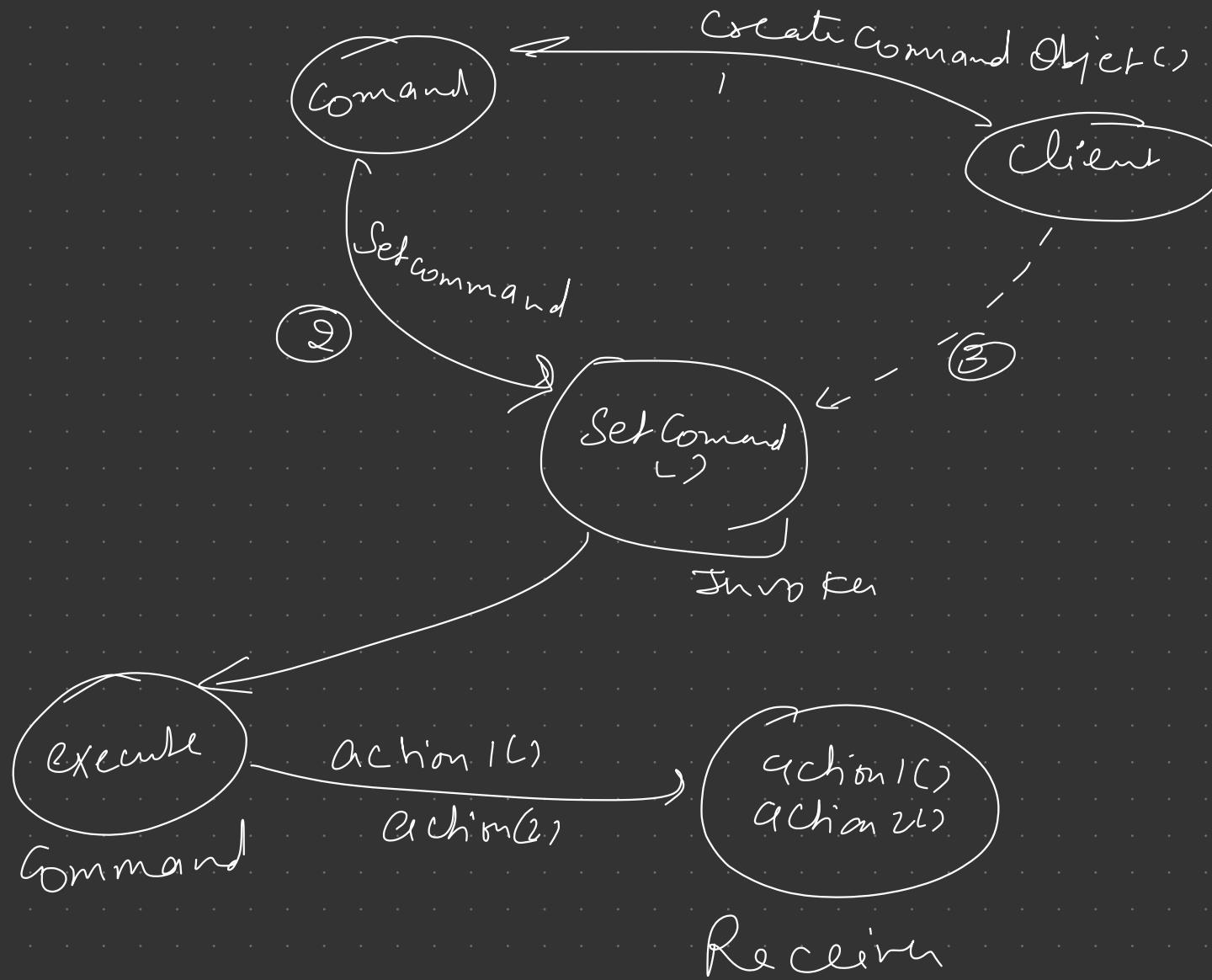
The Abstract factory defines the interface that all Concrete factory implements, which consists of a set of methods:



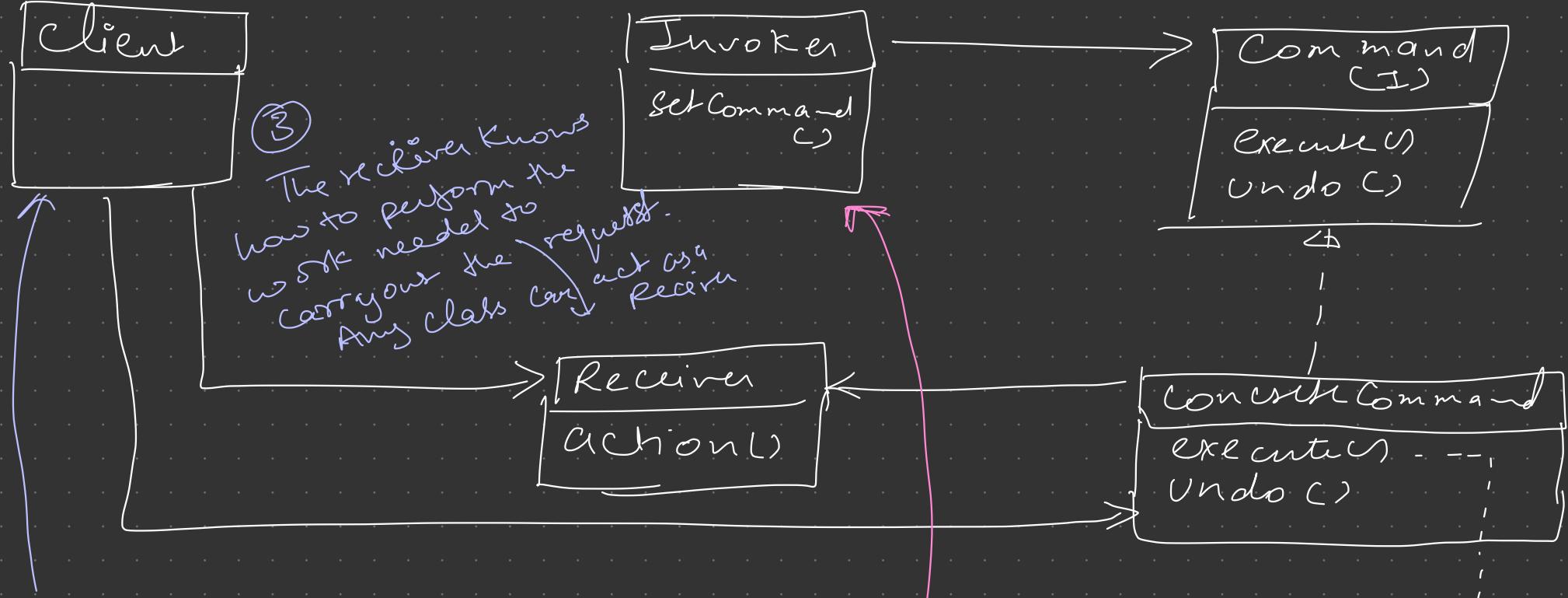
The client is written against the abstract factory & then composed at runtime with an actual factory.



① Command pattern flow

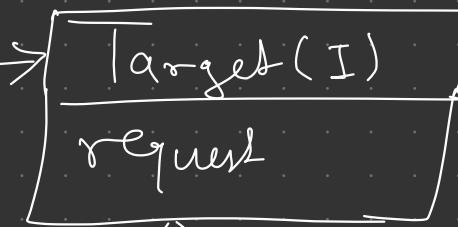


⑦ Command Pattern

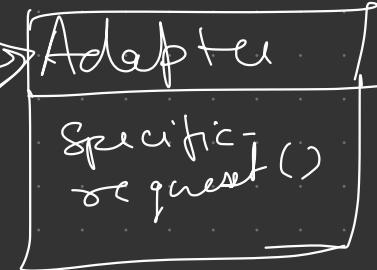


- ① Client is responsible for creating a ConcreteCommand and setting its Receiver.
Public void execute()
{
 receiver.action();
}
- ② The Invoker holds the command & at some point asks the command to carry out a request by calling its execute() method.
- ③ Command declares an interface for all commands.
- ④ The ConcreteCommand defines a binding b/w an action and a Receiver. The invoker makes request by calling execute().

Adapter Pattern



The Client sees
only the Target Interface



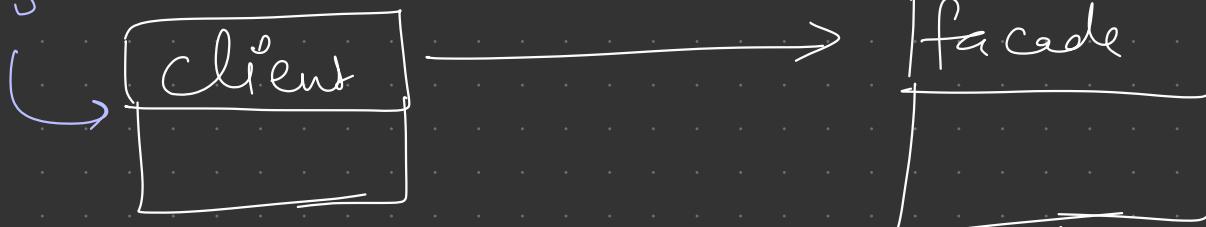
Adapter is composed
of Adaptee

All requests get
delegated to the
Adaptee.

Adapter implements
the Target's interface

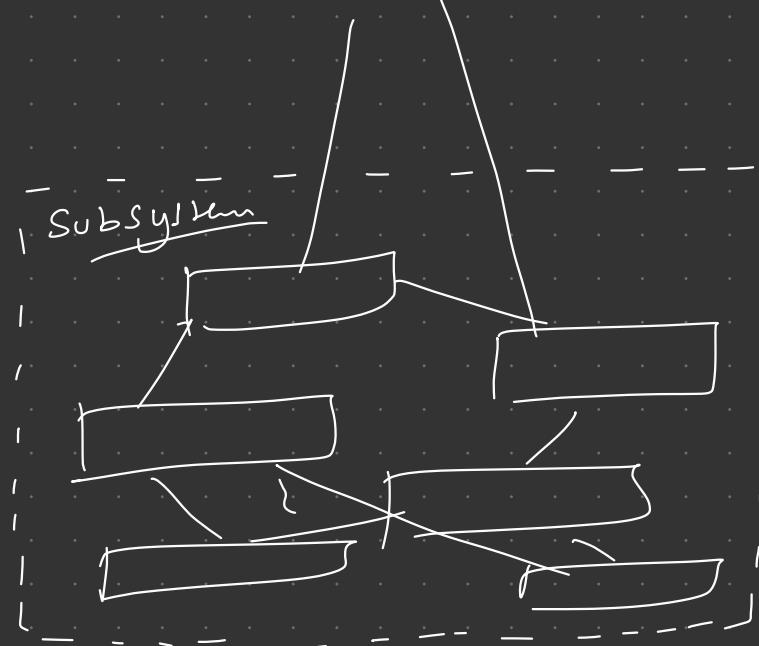
a) Facade Pattern

Happy client whose job is just became easier because of facade



Unified interface that is easier to use

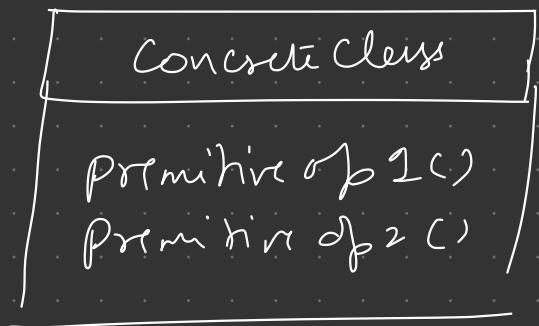
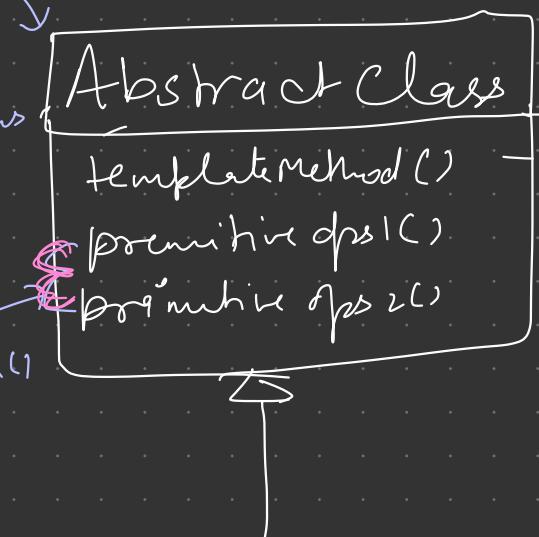
more complex Subsystem.



(10)

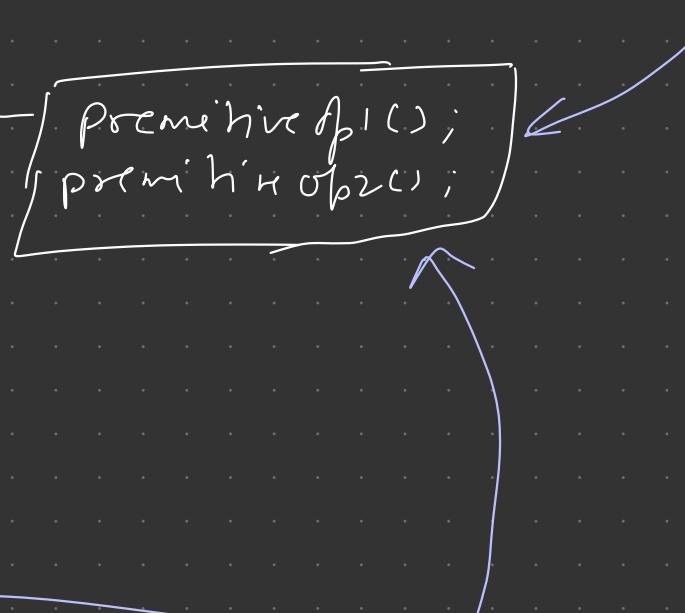
Template method

The abstract class
contains the template method
and
abstraction of the ops
used in the template
method.



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

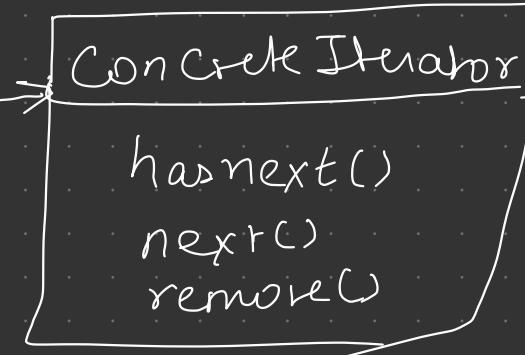
11 Iteration Pattern

Having a common Interface for your aggregates is handy for your clients; It decouples your client from Implementation of your collection of objects.

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

The client uses the Component interface to manipulate the objects in the composition.

Note that leaf also

inherits methods like add(), remove()

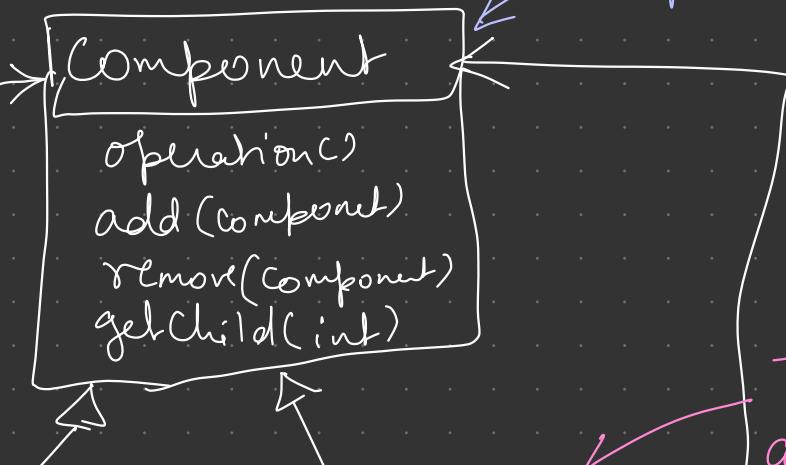
getchild(), which

doesn't necessarily make a lot of sense for a leaf node.



The component defines an interface for all objects in the composition: both the composite and the leaf nodes.

The component may implement a behavior for add(), remove(), getchild() & its operation.

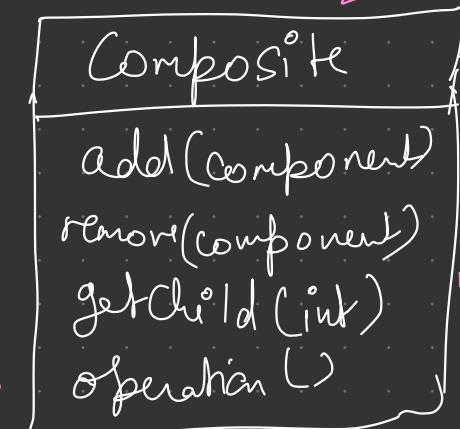


A leaf has no children

Leaf defines the behaviour for the elements in the composition.

It does this by implementing the operations the

Composite supports



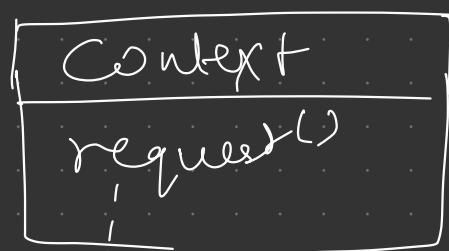
The composite also implements the leaf-related operations. Note that some of these may not make sense on a composite < so in that case an exception might be generated.

The composite's role is to define behaviour of the component having children & to store child component

(13)

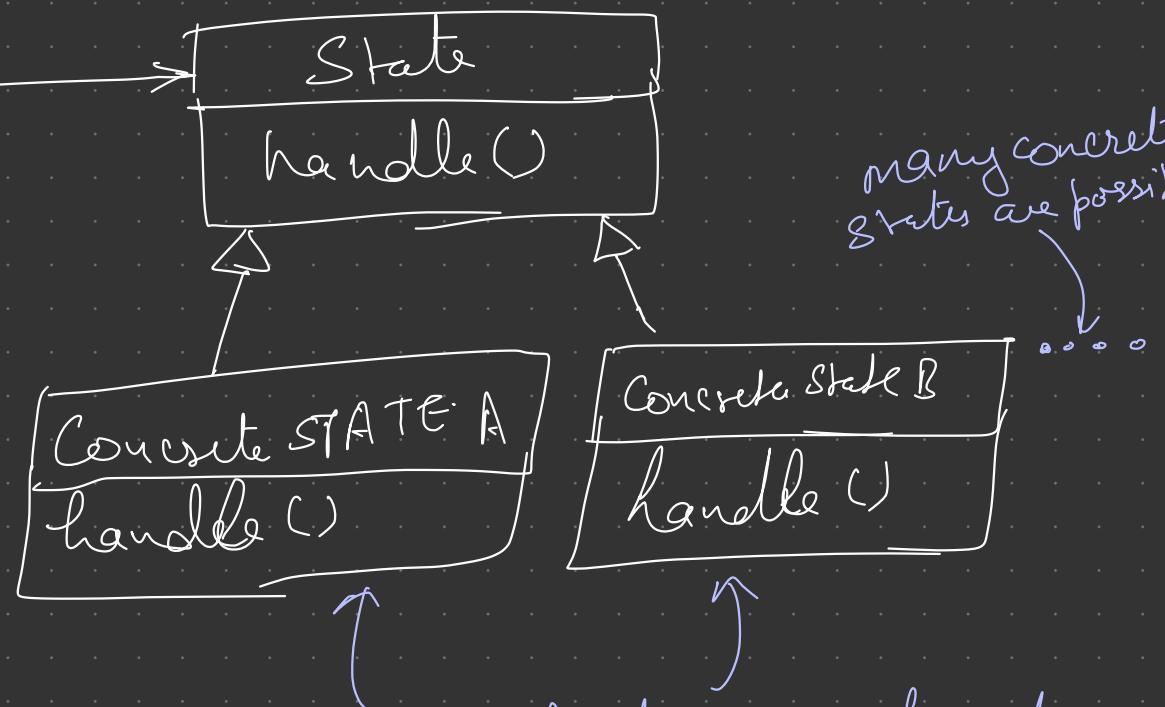
State Pattern

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



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

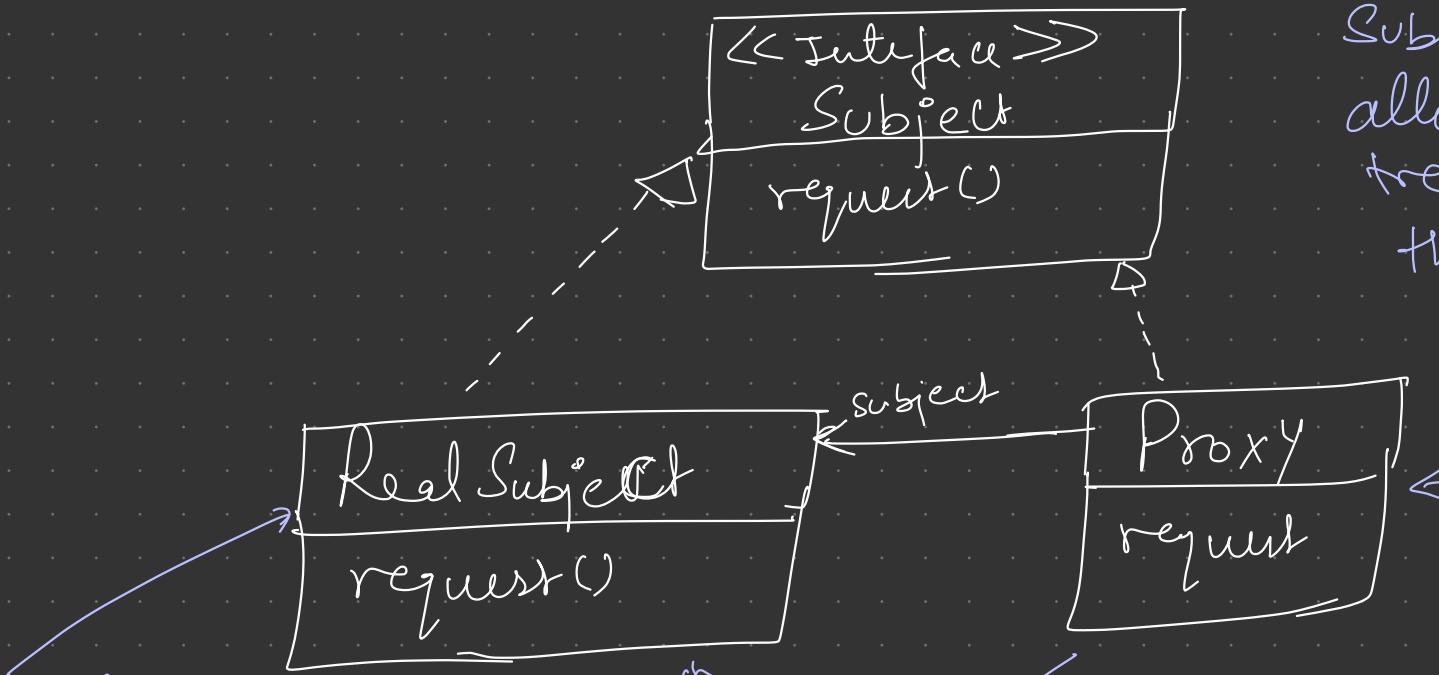
The State interface defines a common interface for all concrete States; The states all implement the same interface so they are interchangeable.



many concrete states are possible

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.

⑯ Proxy Pattern



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

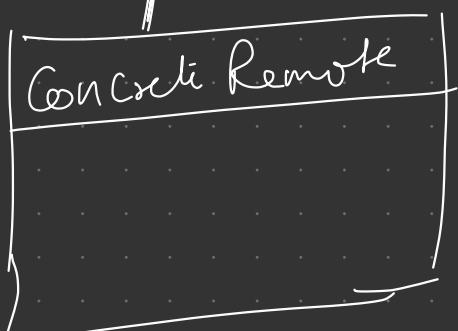
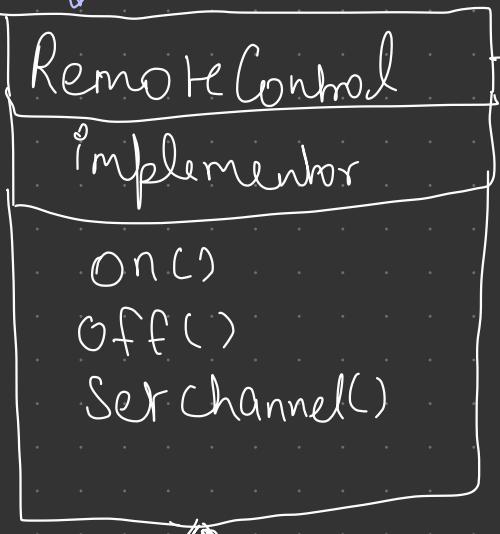
The proxy offers instantiates or handles the creation of the RealSubject

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

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

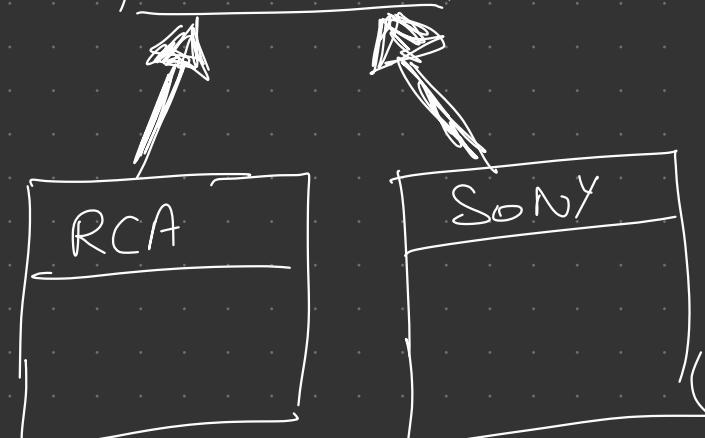
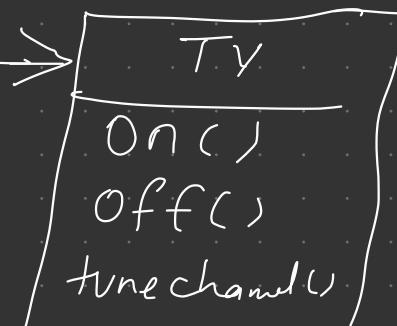
(B) Bridge Pattern

Abstraction class
Hierarchy

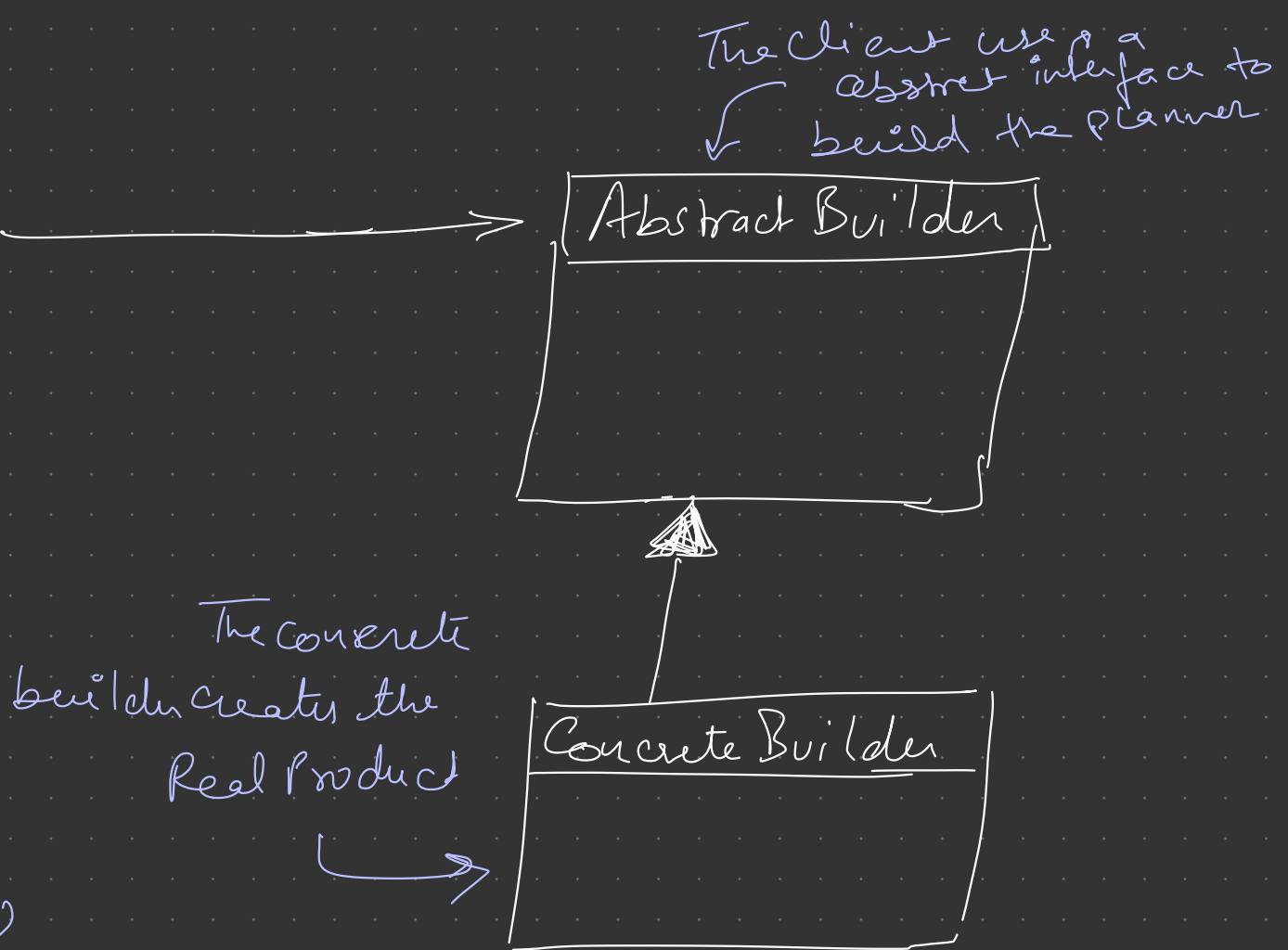
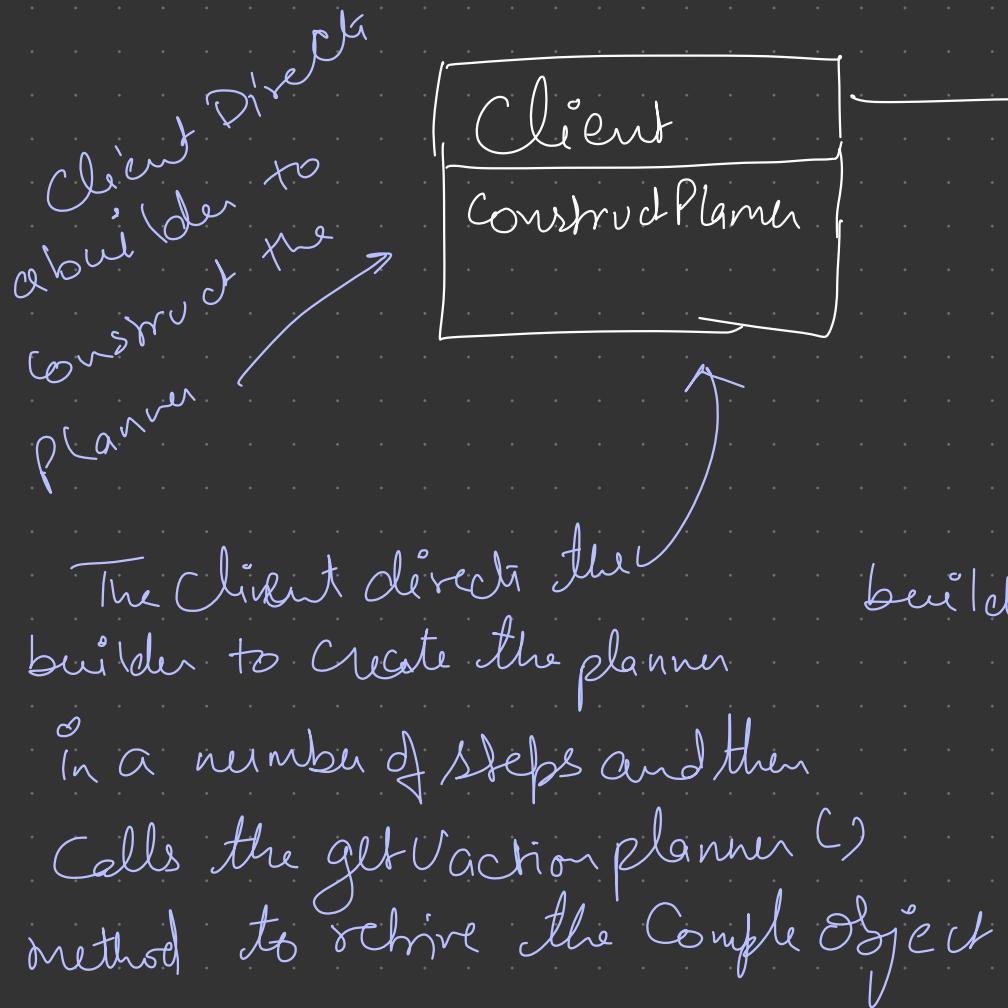


The relationship between the two is referred to as
Bridge

Has-A

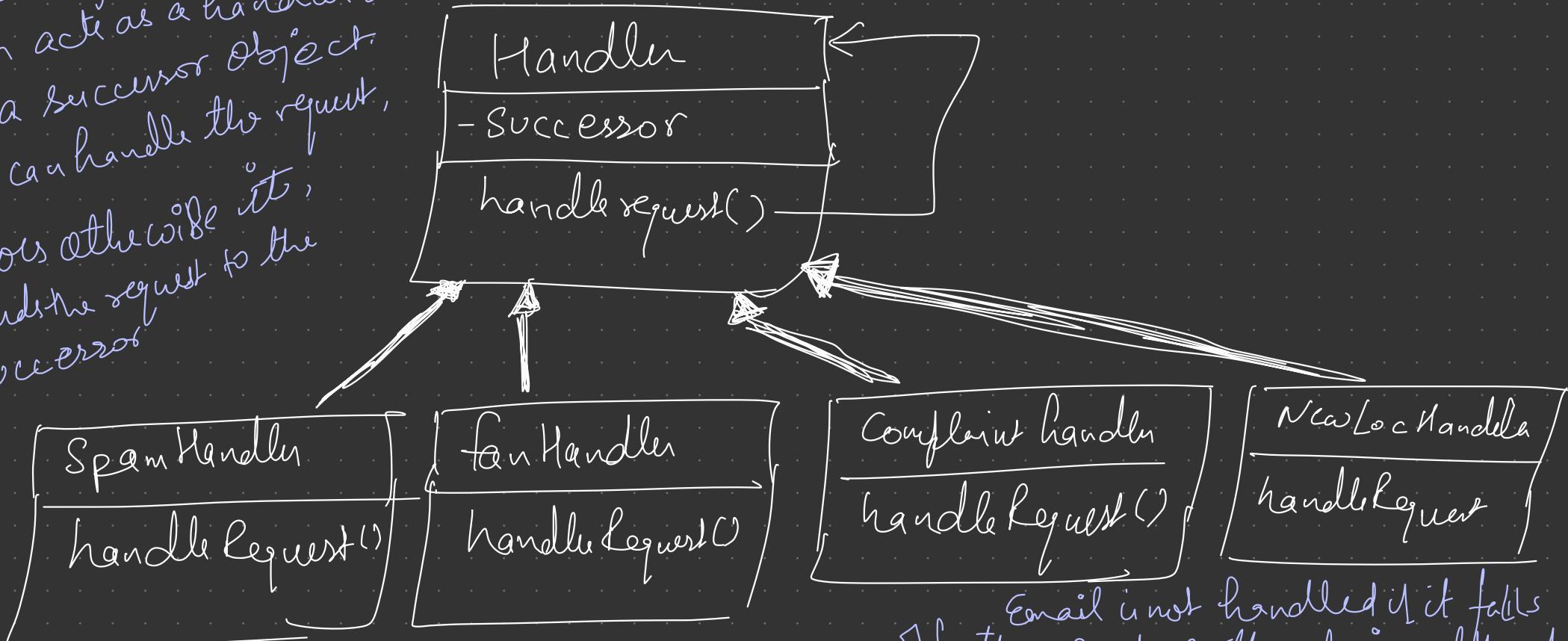


⑥ Builder pattern

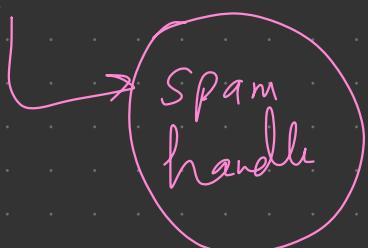


⑩ Chain of Responsibility

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

