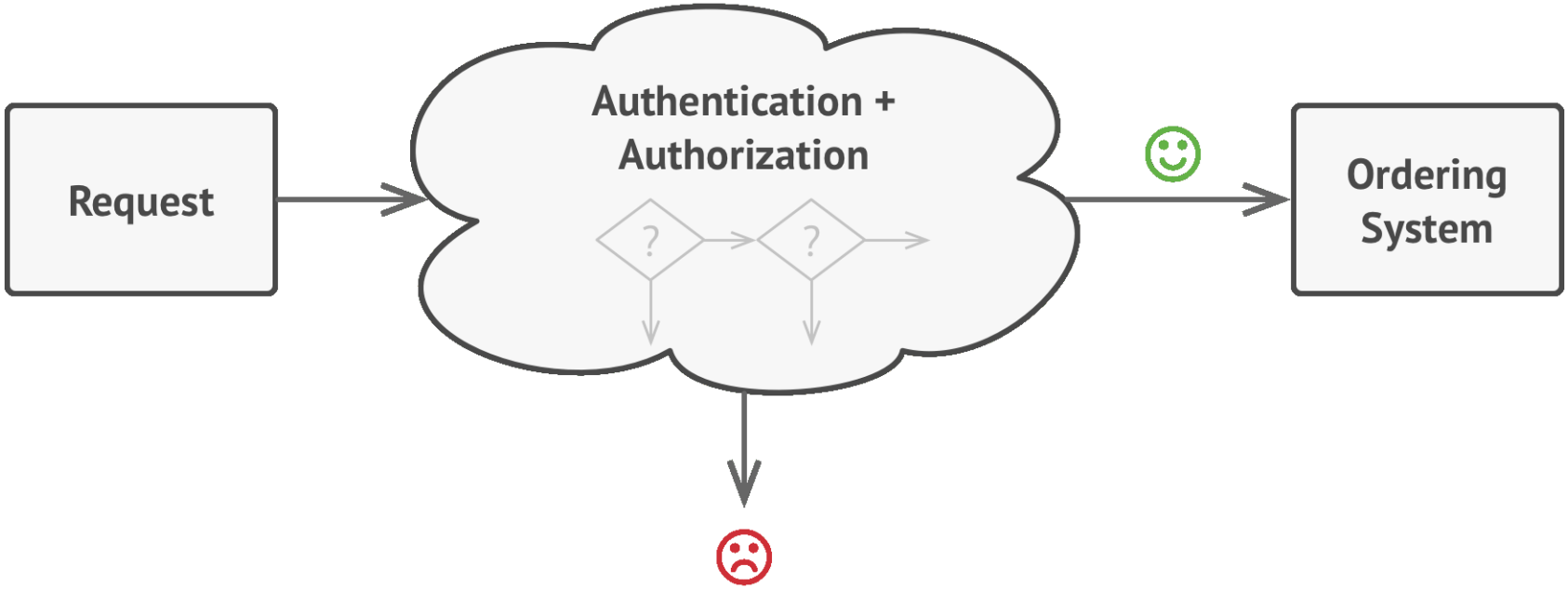
Chain of Responsibility

Also known as: CoR, Chain of Command

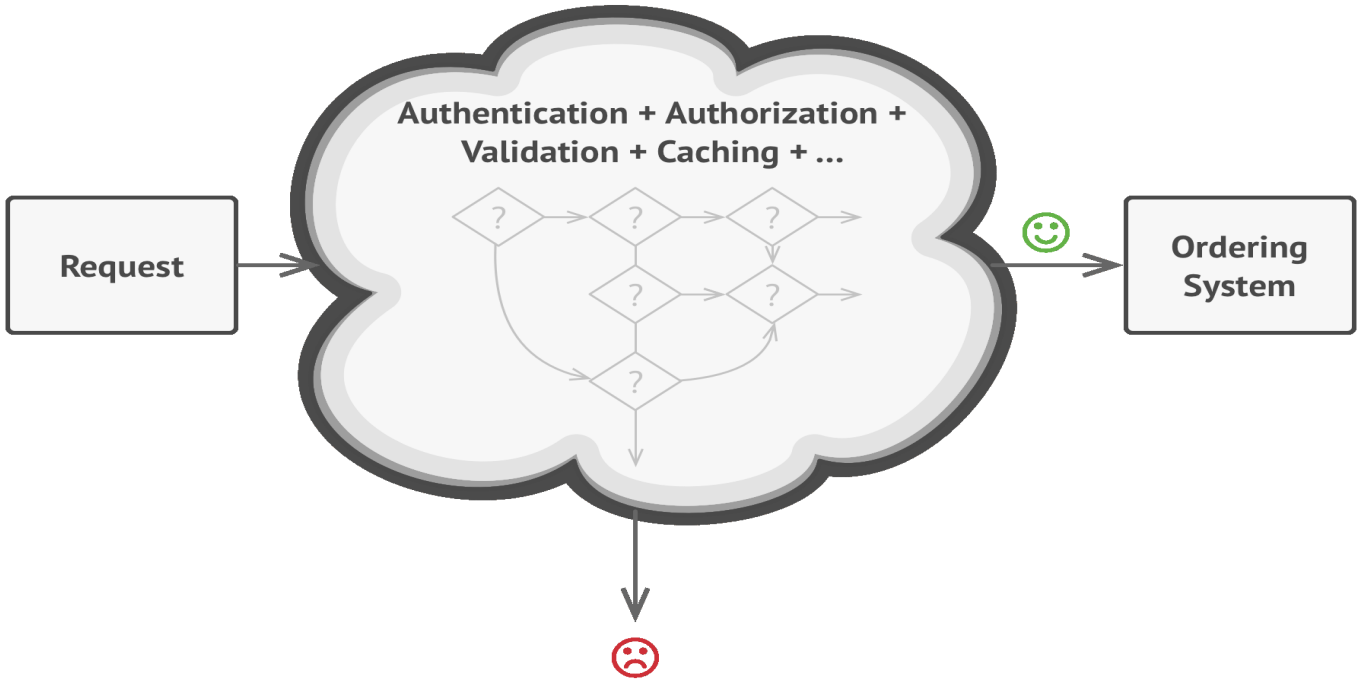
Chain of Responsibility is a behavioral design pattern that allows to pass requests along a chain of handlers. Upon receiving a request, each handler decides either to process the request or to pass it to the next handler in the chain.

Problem



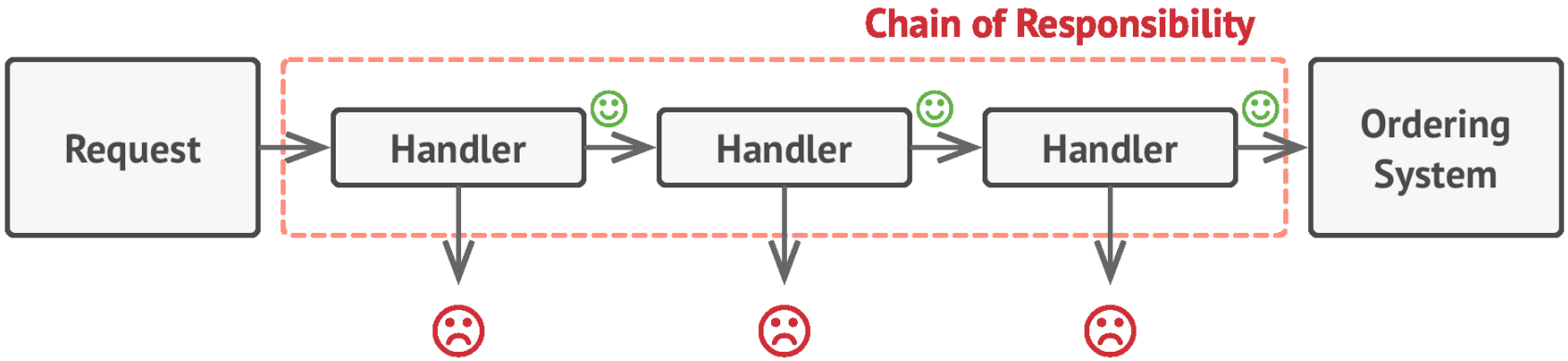
The request must pass a series of checks before the ordering system itself can handle it.

Contd.



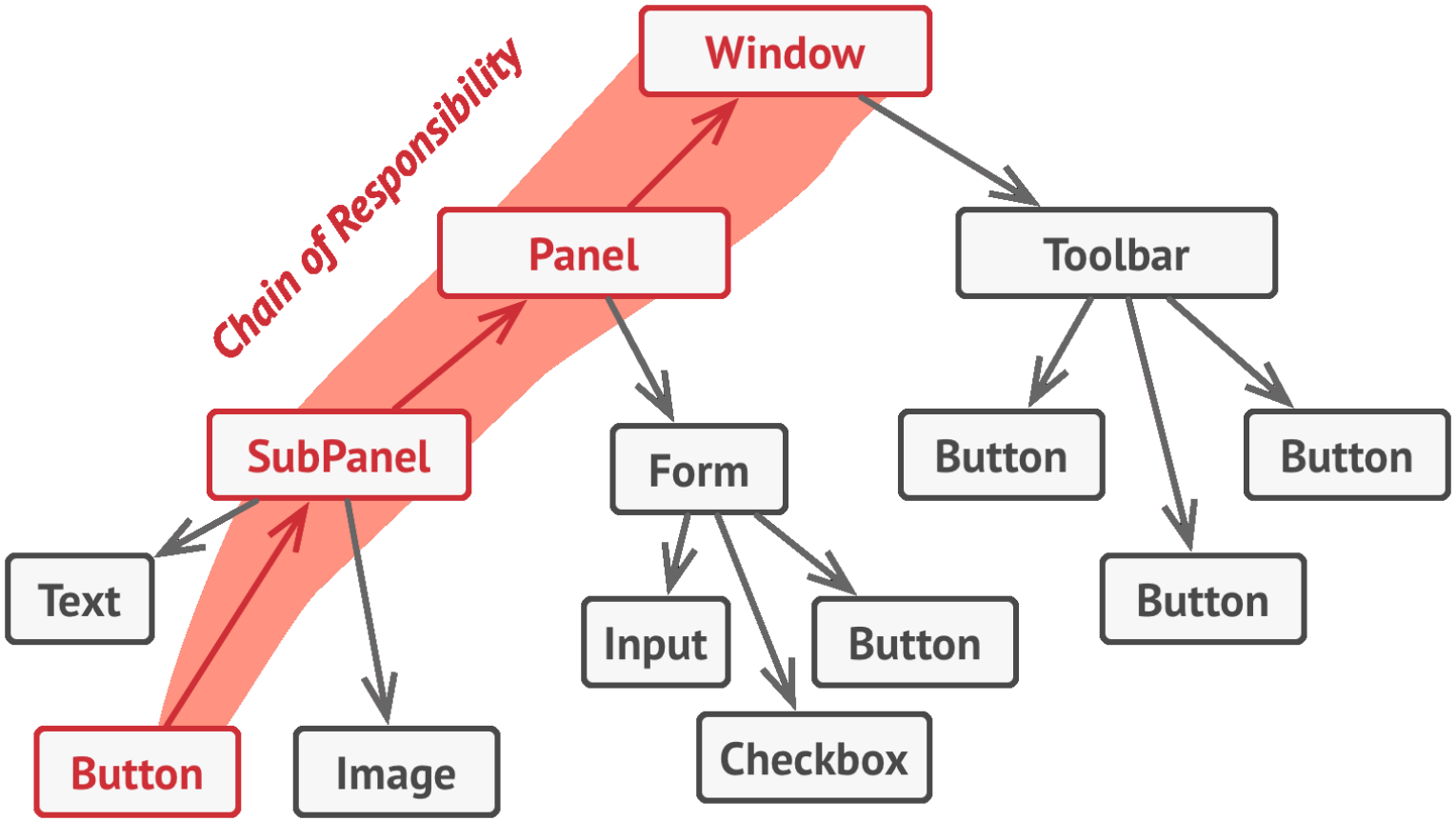
The bigger the code grew, the messier it became.

Solution



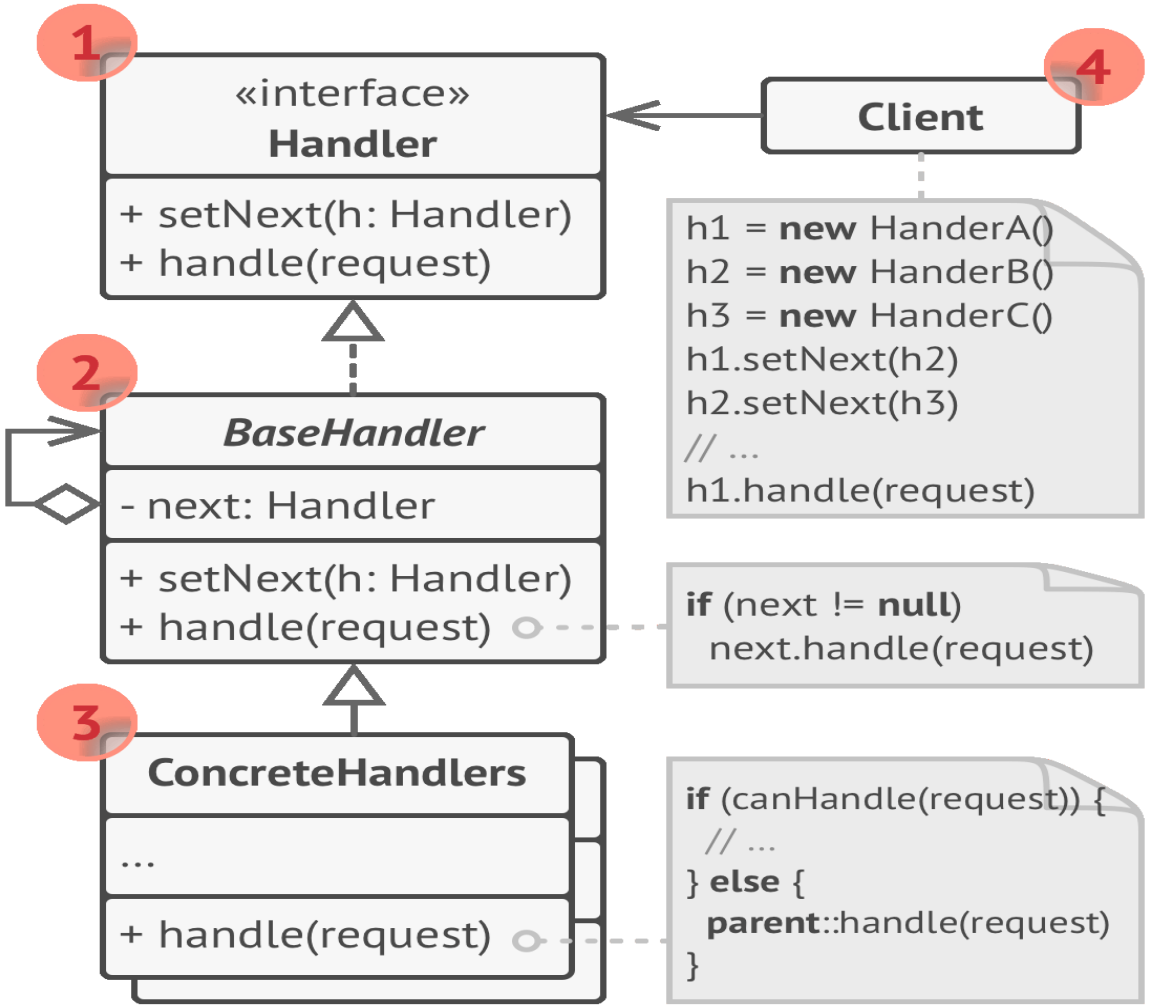
Handlers are lined up one by one, forming a chain.

Contd.



A chain can be formed from a branch of an object tree.

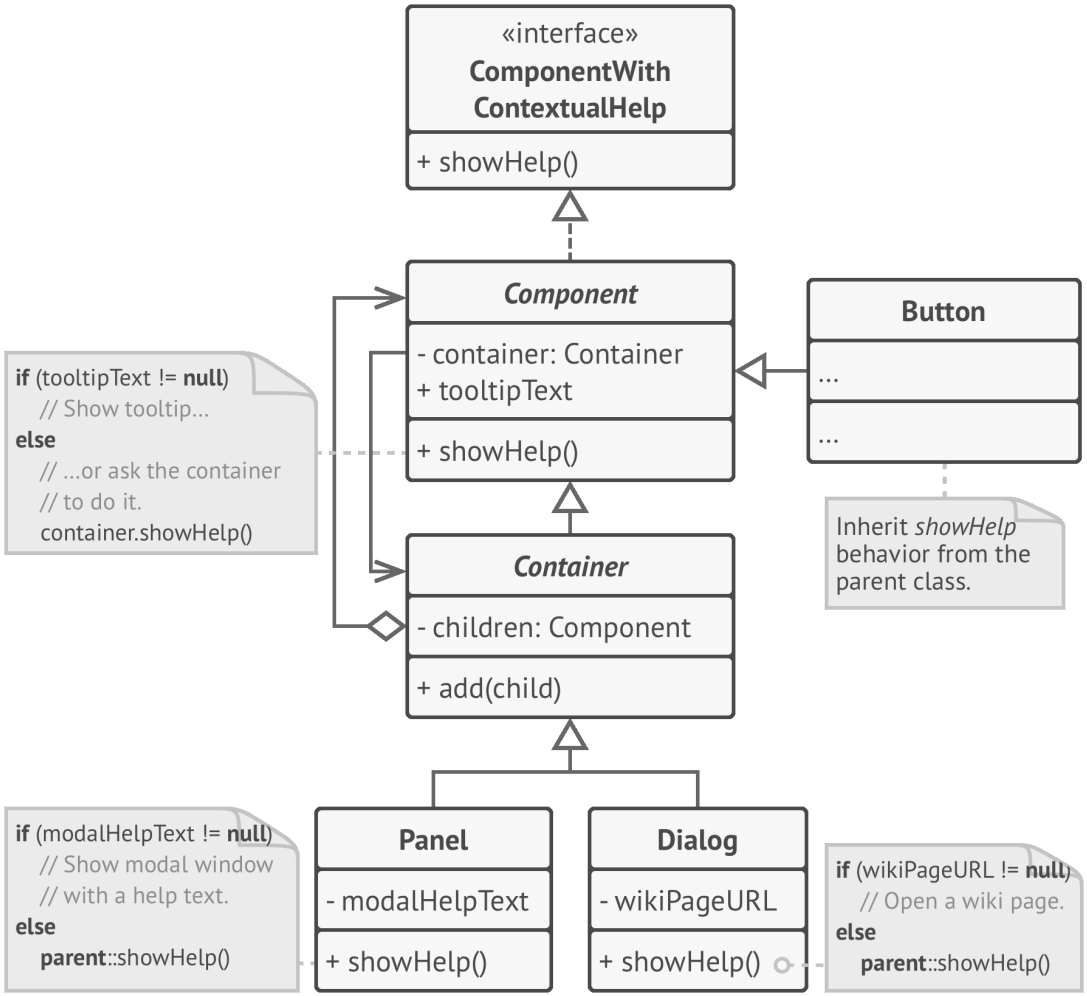
Structure


1
«interface»
Handler
+ setNext(h: Handler)
+ handle(request)
4
Client
h1 = new HanderA()
h2 = new HanderB()
h3 = new HanderC()
h1.setNext(h2)
h2.setNext(h3)
// ...
h1.handle(request)
2
BaseHandler
- next: Handler
+ setNext(h: Handler)
+ handle(request)
if (next != null)
next.handle(request)
3
ConcreteHandlers
...
+ handle(request)
if (canHandle(request)) {
// ...
} else {
parent::handle(request)
}

Contd.

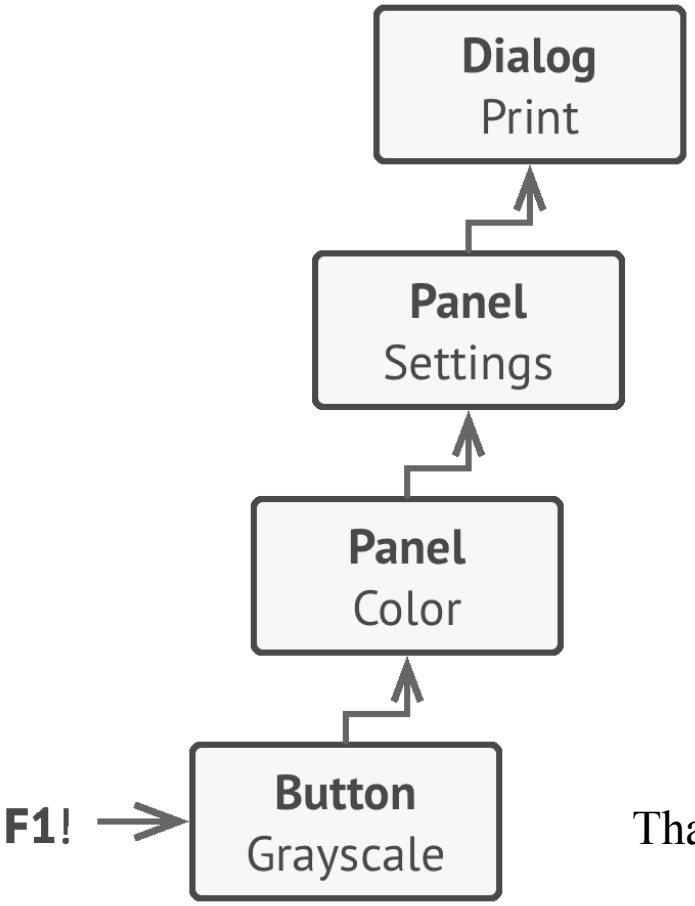
- The **Handler** declares the interface, common for all concrete handlers. It usually contains just a single method for handling requests, but sometimes it may also have another method for setting the next handler on the chain.
- The **Base Handler** is an optional class where we can put the boiler plate code that's common to all handler classes. Usually, this class defines a field for storing a reference to the next handler. Clients can build a chain by passing a handler to the constructor or setter of the previous handler. The class may also implement the default handling behavior: it can pass execution to the next handler after checking for its existence.
- **Concrete Handlers** contain the actual code for processing requests. Upon receiving a request, each handler must decide whether to process it and, additionally, whether to pass it along the chain. Handlers are usually self-contained and immutable, accepting all necessary data just once via the constructor.
- The **Client** may compose chains just once or compose them dynamically, depending on the application's logic. Note that a request can be sent to any handler in the chain—it doesn't have to be the first one.

Pseudo code



The GUI classes are built with the Composite pattern. Each element is linked to its container element. At any point, you can build a chain of elements that starts with the element itself and goes through all of its container elements.

Contd.



That's how a help request traverses GUI objects.

Applicability

- Use the Chain of Responsibility pattern when program is expected to process different kinds of requests in various ways, but the exact types of requests and their sequences are unknown beforehand.
- Use the pattern when it's essential to execute several handlers in a particular order.
- Use the CoR pattern when the set of handlers and their order are supposed to change at runtime.

Pros and Cons

- It can be controlled the order of request handling.
- *Single Responsibility Principle*. It can decouple classes that invoke operations from classes that perform operations.
- *Open/Closed Principle*. We can introduce new handlers into the app without breaking the existing client code.
- Some requests may end up unhandled.