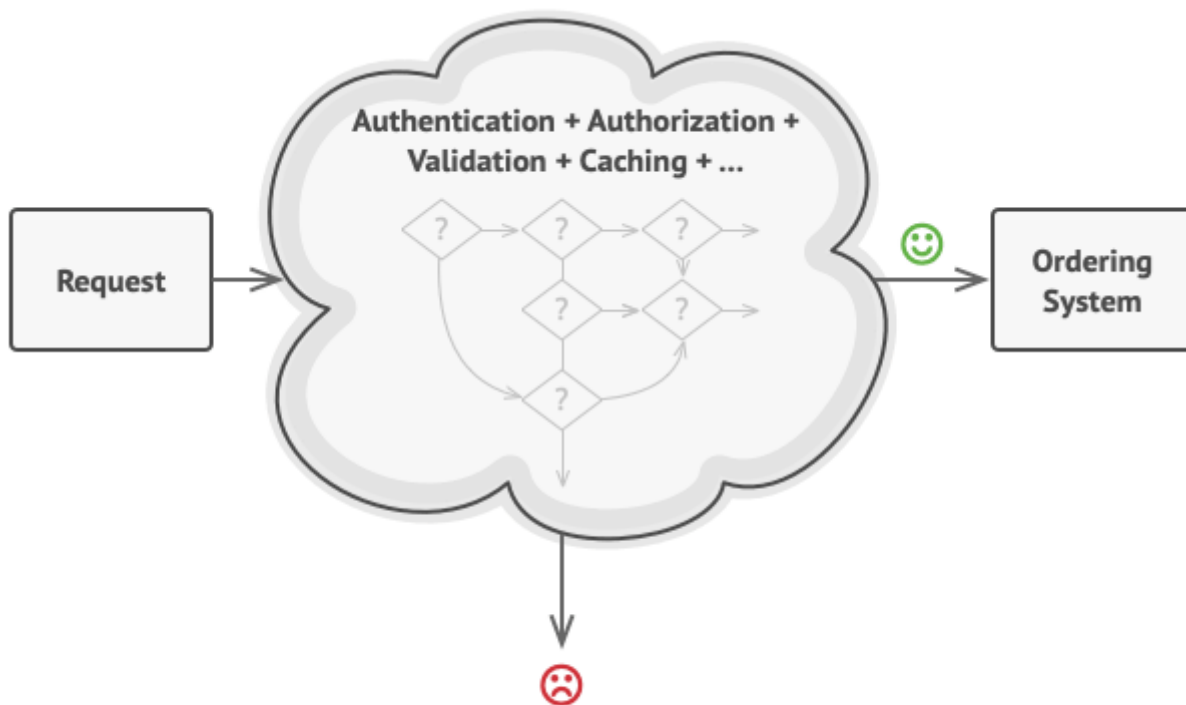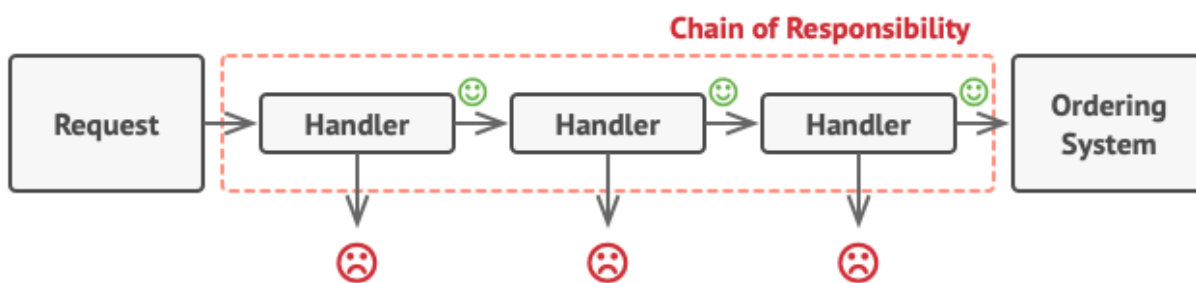# Intent

Chain of Responsibility is a behavioral design pattern that lets you 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 & Solution

The bigger the code grew, the messier it became.


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

Structure

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 you can put the boilerplate code that's common to all handler classes.

Usually, this class defines a field for storing a reference to the next handler. The 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.
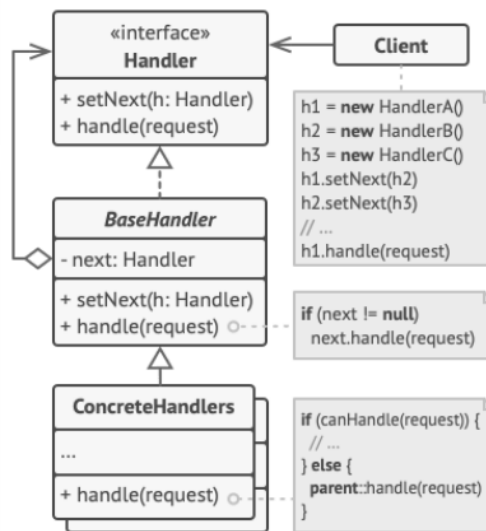
The **Client** may compose chains just once or compose them dynamically, depending on the application's logic. Note that a request can be sent to any handler in the chain—it doesn't have to be the first one.

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.

```
«interface»
Handler
+ setNext(h: Handler)
+ handle(request)
```

```
Client
```

```
h1 = new HandlerA()
h2 = new HandlerB()
h3 = new HandlerC()
h1.setNext(h2)
h2.setNext(h3)
// ...
h1.handle(request)
```

```
BaseHandler
- next: Handler
+ setNext(h: Handler)
+ handle(request) o----
```

```
if (next != null)
  next.handle(request)
```

```
ConcreteHandlers
...
+ handle(request) o----
```

```
if (canHandle(request)) {
  // ...
} else {
  parent::handle(request)
}
```

Applicability

- Use the Chain of Responsibility pattern when your 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.