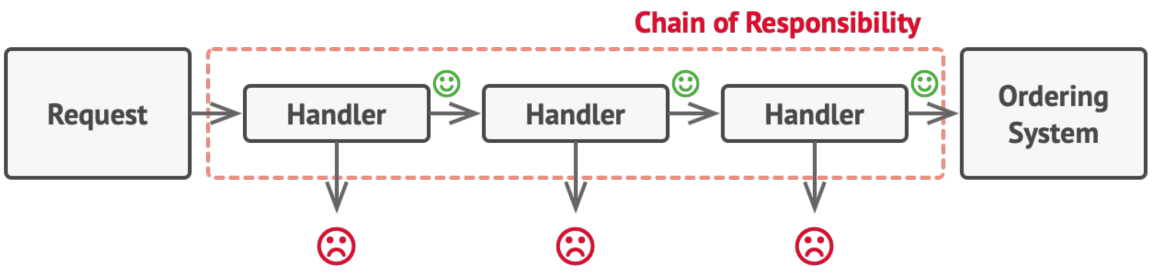
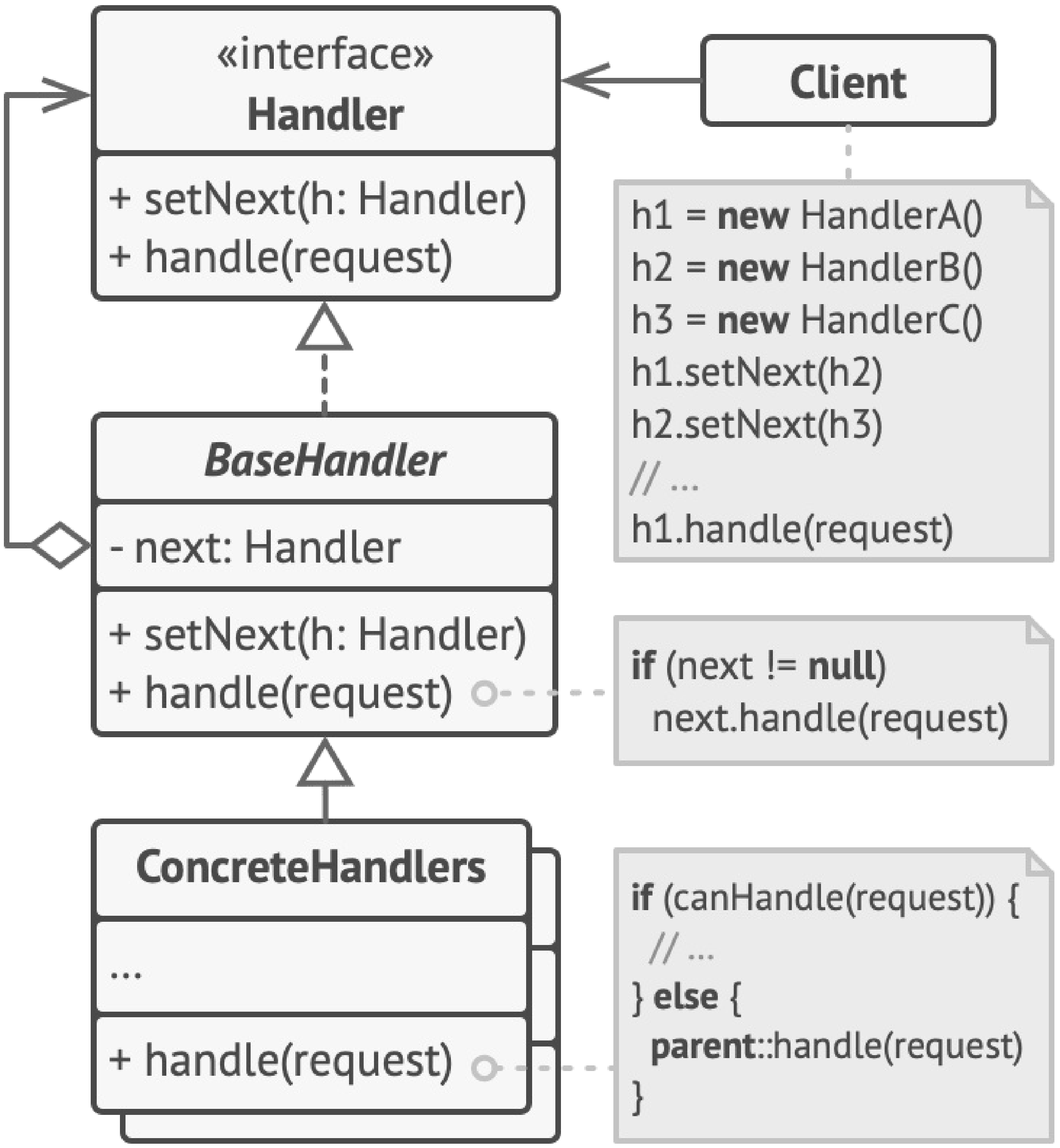
Chain Of Responsibility Design Pattern

- 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** : Imagine you have a system where you are passing requests to order something (food, products, etc).
  Now you want to add a handler to check if user is authenticated or not.
  After sometime, you add another check of the request coming to you is in JSON format or not.
  After sometime, you add another handler to check if the user is admin or not. If the user is admin, he/she can do things for which only an admin will have access.
  Now if you keep all this logic inside a single handler, you code becomes messy, large and is not reusable.
- **Solution** : Now, if you transform each functionality into it's own handler you end up with clean and reusable code.
  Also, each handler will have it's own functionality. If request format is correct then check if user is authenticated or not. If user is authenticated then check whether user is admin or not.



- **Chain of Responsibility** relies on transforming particular behaviors into stand-alone objects called handlers.
- Every handler has it's own responsibility and after completing that if can decide whether it needs to pass the request to next handler in the chain or not.

UML



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

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

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

Implementation Considerations

- Prefer defining handler as interface as it allows you to implement chain of responsibility without worrying about single inheritance rule of java
- Chain can be described by using XML or JSON as well so that you can add and remove handlers from chain without modifying code.

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 pattern when the set of handlers and their order are supposed to change at runtime.

Pros and Cons

Pros	Cons
You can control the order of request handling.	Some requests may end up unhandled.
Single Responsibility Principle. You can decouple classes that invoke operations from classes that perform operations.	
Open/Closed Principle. You can introduce new handlers into the app without breaking the existing client code.	