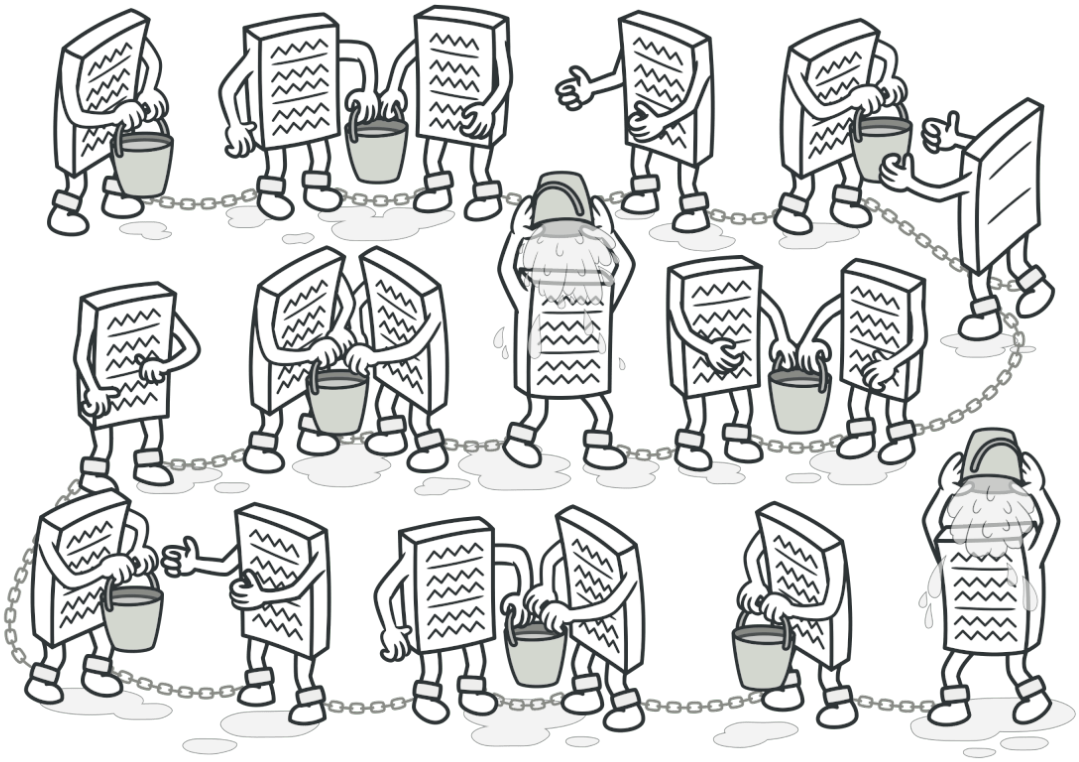
참고 자료

- https://refactoring.guru/design-patterns/chain-of-responsibility

Chain of Responsibility란?

- Behavioral object pattern
  - 그룹 간 어떻게 소통할 지에 관한 패턴
- 요청을 chain of handlers에게 전달하는 패턴입니다.

  이 체인에서 각 handler가 요청을 전달받으면 요청을 처리할 지, 다음에게 넘겨줄 지 결정을 합니다.



상황

- Online Ordering System

요구사항

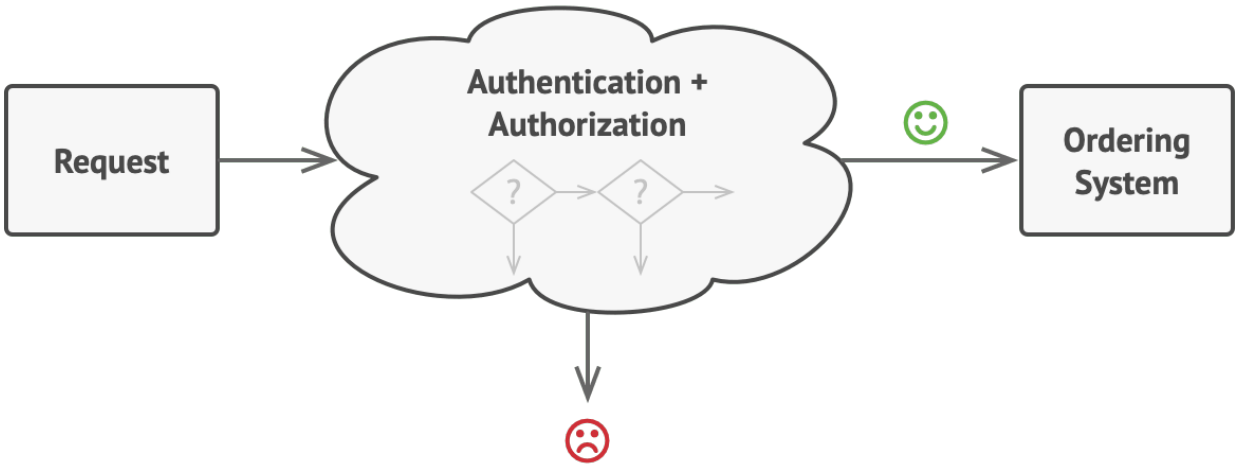
- 권한을 부여받은 유저만 주문을 생성할 수 있도록 시스템 접근을 제한하고 싶습니다.
- 또한, 관리자 권한이 있는 유저만 모든 주문에 대해 모든 정보를 볼 수 있도록 하고 싶습니다.

접근

- 이는 순차적으로 확인해야만 하는 사항임을 깨닫습니다.

  요청을 받을 때마다

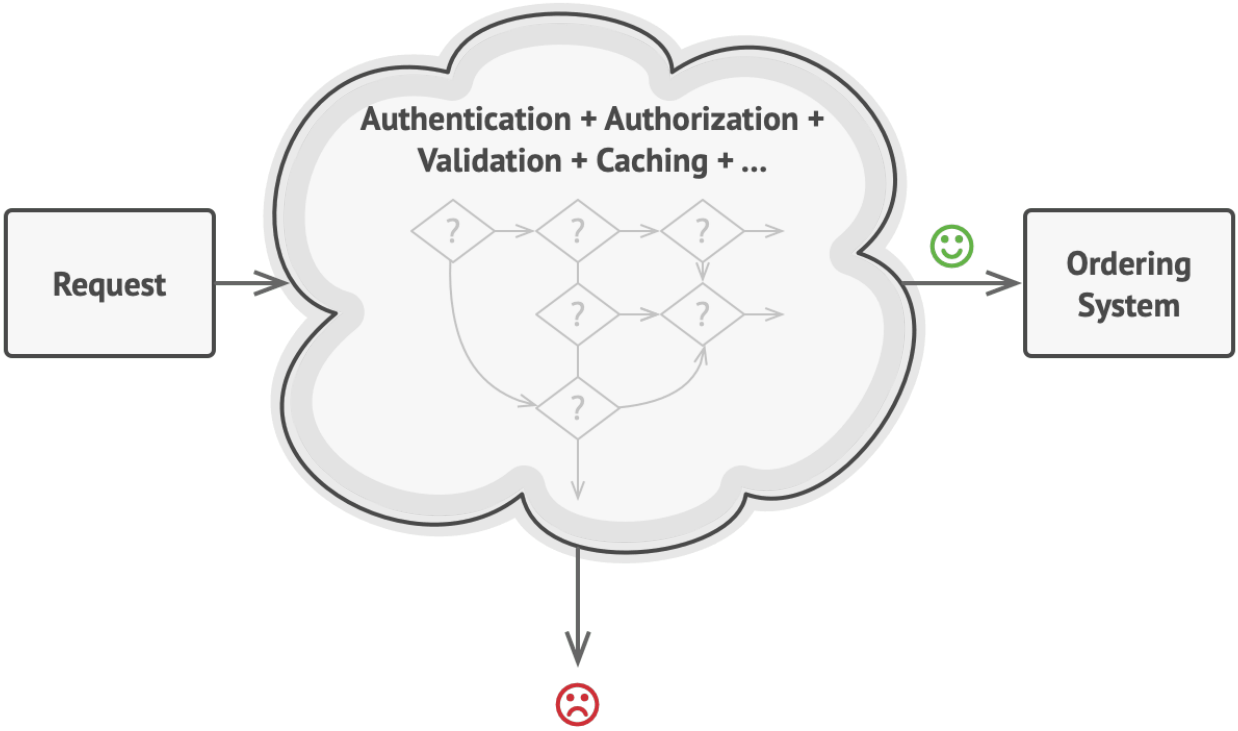
  1. credential을 확인한 뒤(authentication)
     - 여기서 만약에 credential이 잘못됐거나, authentication이 실패하면 절차를 진행할 수 없습니다.
  2. 그에 맞는 권한을 부여해야 합니다.(authorization)



문제의 발생 - 요구사항 추가

시간이 흘러 다음과 같은 요구사항을 구현해야 합니다.

- 보안상 문제로 주문 정보를 raw 데이터로 전달하는 것은 좋지 않기 때문에 validation 작업을 추가하는 요청
- burute force password cracking에 취약하여 동일 아이피로 반복하여 비밀번호 입력을 실패했는지 확인하는 작업
- 속도 개선을 위해 캐싱하는 작업



한 object에 이를 구현했으면 기능을 추가할 때마다 코드가 점점 더 커지고 다른 확인(check) 작업에 영향을 줄 수 있습니다. 다른 곳에 특정 기능을 추가하려 해도 재사용할 수 없어 중복이 발생할 수 밖에 없죠.

이로 인해 이 시스템은 점점더 이해하기 어려워지고 유지보수 비용이 증가하게 됩니다.

해법 - chain of handlers를 만들어라!

handler

- chain of handlers는 특정 behaviors를 handlers 객체에 전달하는 방식입니다.

  handler: stand-alone objects (자신 하나만으로 완전한 객체 또는 다른 객체와 연관없어도 완전한 객체?)
- 각 Check는 단일 method를 가진 하나의 클래스로 바꿀 수 있습니다.
- request에 관한 data는 해당 메서드의 argument로 전달되게 됩니다.

chain으로 적용

- 이러한 handler를 하나의 chain으로 연결하도록 합니다. 그리고 각 연결된 handler는 다음 핸들러 참조를 위한 필드를 지니게 합니다.

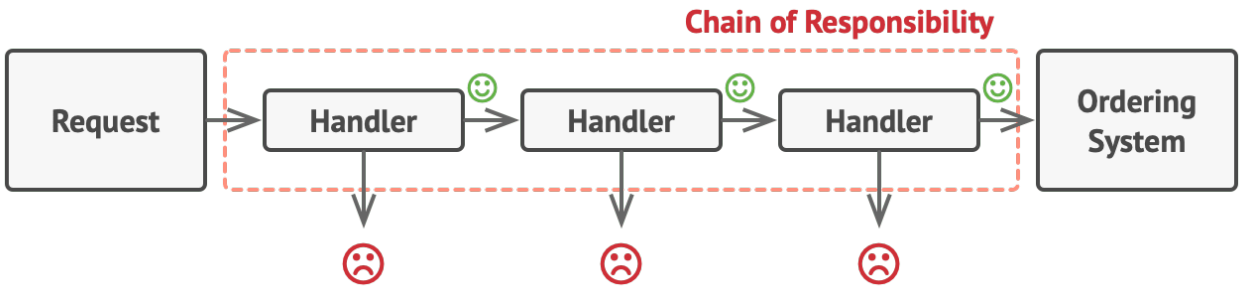
이렇게 함으로써 한 handler가 요청을 처리할 뿐만 아니라 해당 요청을 다음 handler에게 전달할 수 있게 되는 것이죠. 그래서 그 요청은 모든 handler가 요청을 처리할 수 있을 때까지 순회하게 됩니다.

- Point

핸들러는 요청을 처리할 지 말지 결정할 뿐만 아니라 효과적으로 처리를 중단할 수 있습니다.

적용

ordering system에서 validation, authentication 등등이 단일 메소드를 가진 handler가 될 수 있습니다. 그리고 handler의 check를 통과하게 되면 다음 handler로 요청을 전달하는 형태를 만들 수 있습니다.



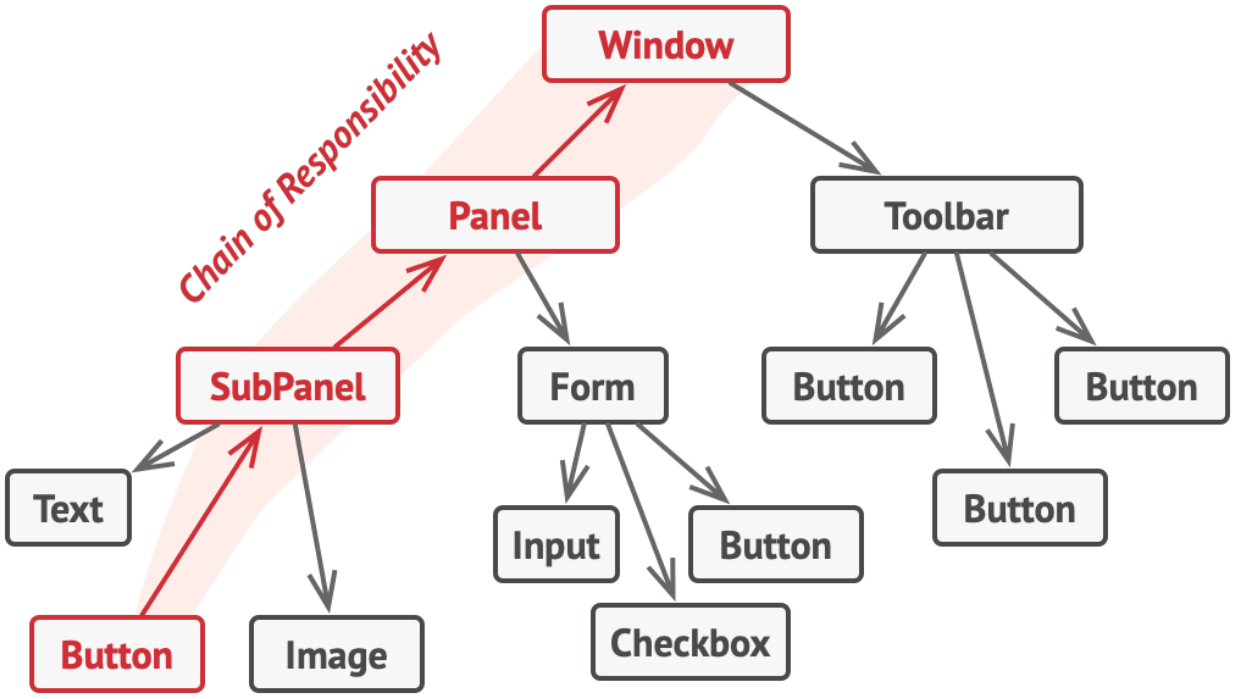
따라서 일련의 handler들을 check에 대한 responsibility를 지닌, chain이라 볼 수 있습니다.
(ChainOfResponsibility)

chain of handlers에 대한 또 다른 접근 - 단 하나의 handler만 요청을 처리하는 chain

위 예제는 하나라도 검증을 통과하지 못하면 request가 거부되는 형태입니다. 그런데 다른 측면에서는 아래와 같이 사용할 수 있습니다.

chain에서 요청을 받습니다. 그리고 만약 handler가 자신이 처리할 수 있으면 처리하고 끝냅니다. 하지만 자신이 요청을 처리할 수 없다면 다음 handler에게 넘기는 것이죠.

따라서 이 chain에서 request를 처리하는 handler는 오직 하나이거나 아예 없습니다. 이 접근은 GUI에서 흔합니다.

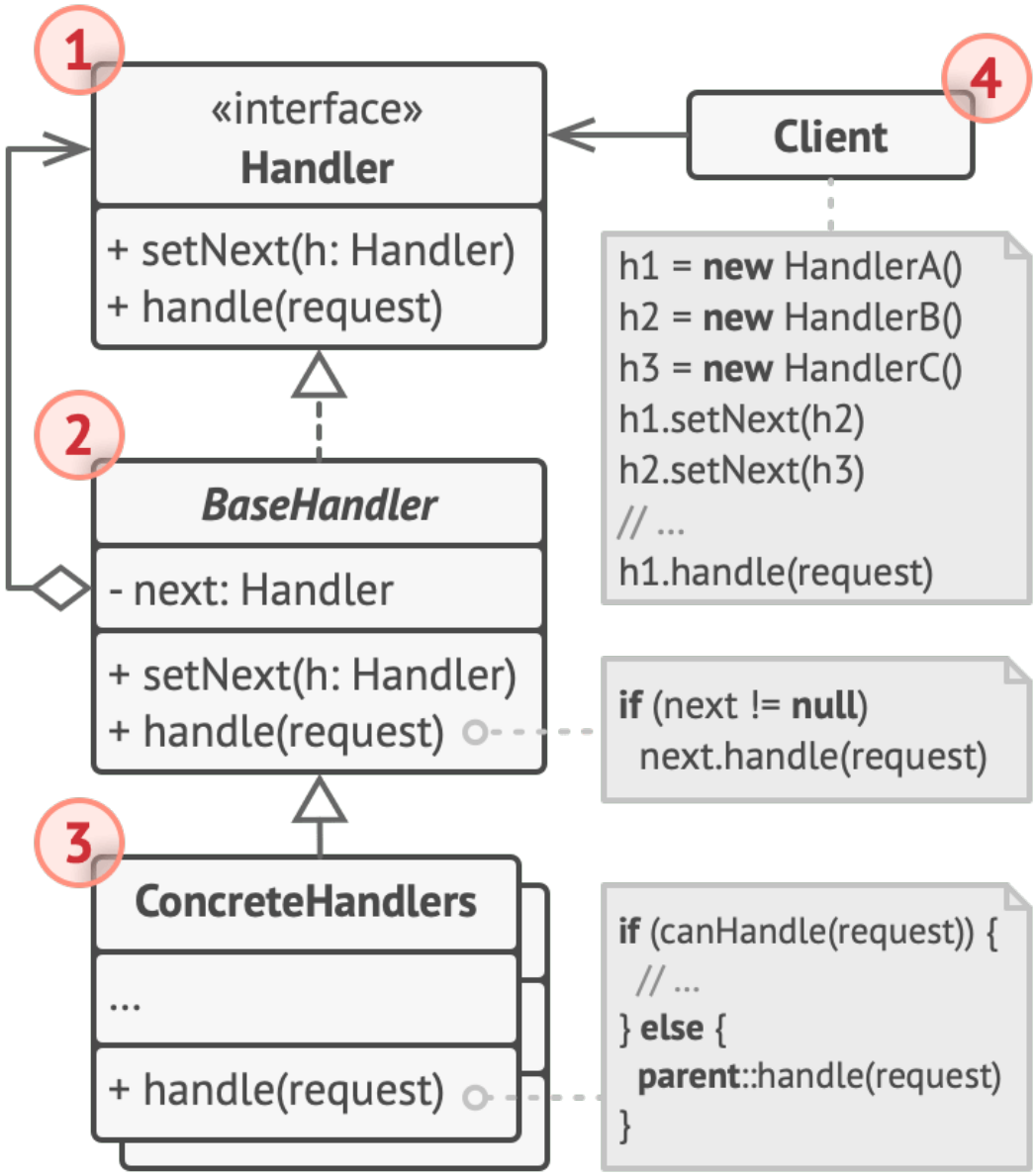


주의할 사항

handler는 동일한 인터페이스를 구현해야 합니다.

- 핸들러가 concrete 핸들러에 의존하면 다양한 핸들러 순서를 바꿀 수 없거나 클라이언트가 특정 핸들러에 의존되기 때문입니다.

구조



1. Handler

모든 Concrete handlers에 대한 Interface입니다.

이 interface는 request처리에 대한 single method만 가지고 있습니다.

하지만 때때로 chain에서 다음 핸들러 setting을 위해 handler 필드 값을 setting하는 메서드가 있을 수 있습니다.

2. BaseHandler

이 클래스는 옵션입니다. 모든 handler에 공통으로 적용하는 코드를 넣을 수 있습니다.

3.ConcreteHandlers

request를 처리하는 구체적인 로직이 담겨 있습니다.

요청을 받으면 각 핸들러는 요청을 처리할 지 말 지 반드시 결정해야 하며, 추가적으로 다음 handler에게 넘길 수 있습니다.

보통 contructor를 통해 필요한 데이터를 받음으로써 Handler는 self-contained하고 immutable하게 됩니다.

4.Client

클라이언트는 application logic에 따라 한 번만 chain을 구성하거나 동적으로 구성합니다.

주의할 사항은 request가 반드시 Chain의 첫번째 handler에 갈 필요는 없습니다. 중간 handler에게 갈 수 있죠.