Facade Pattern

결론 및 정리

정의

복잡한 서브시스템을 단순화된 인터페이스 (실제 interface 라기 보단, 유저와 상호작용할 수 있는 부분) 를 제공하는 단일 클래스로 감싸는 패턴.

다음과 같은 상황에서 보통 사용한다

1. 복잡한 서브시스템을 단순화: 여러 개의 클래스와 인터페이스로 구성된 복잡한 서브시스템을 가지고 있을 때, 클라이언트가 직접 서브시스템과 상호작용하기보다는 Facade 클래스를 통해 단순한 인터페이스로 상호작용할 수 있습니다.
2. 인터페이스의 일관성: 여러 개의 클래스와 인터페이스를 사용하는 서브시스템을 클라이언트에게 통일된 인터페이스로 제공하여 일관성을 유지할 수 있습니다.

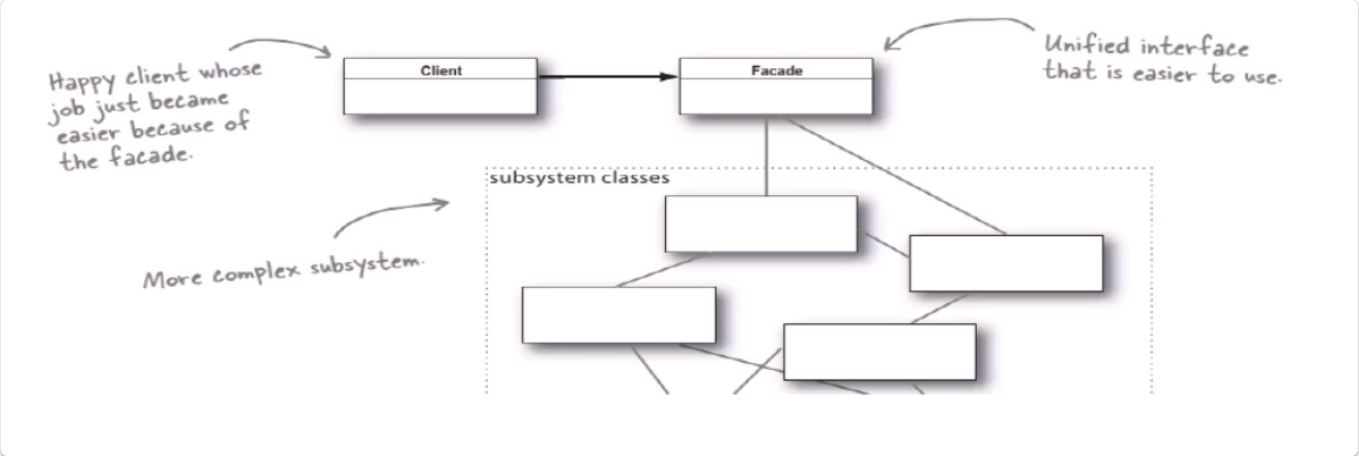
Facade 클래스: 서브시스템의 단순화된 인터페이스를 제공하는 클래스.

서브시스템 클래스: 실제 작업을 수행하는 여러 개의 클래스와 인터페이스로 구성된 서브시스템. Facade 클래스는 이러한 서브시스템 클래스를 호출하여 요청을 처리.

구현

맞닿아있는 부분을 만드는 것 자체가 정의인 만큼, 구현 자체도 굉장히 간단하다.

Facade Class 를 만든다. Facade 자체는 실제 동작 (알고리즘) 을 하지 않는다.
Facade Class 는 SubSystem Class 및 Interface 를 포함하며, 이들을 호출함으로써 작업을 진행한다.



Context & Problem & Merit

- 서브 시스템 등이 복잡하게 구성되어서, 그 복잡성을 낮추고자 할 때 사용한다.
- 사용성을 쉽게 하고, 복잡성을 낮춘다.
- 내부 내용이 바뀌더라도, 그에 상관없이 Facade Class 를 사용해줄 수 있다.

Design Principles

> *Design Principle 4 : The Pirnciple of Least Knowledge (Law of Demeter)*
> *객체 간의 상호작용을 제한하고 결합도를 낮추라는 원칙이다.*
>
> *연결을 줄여라. 최대한 Loose 하게 Coupling 될 수 있도록 만들라.*
> *많은 Dependency 를 가질수록, 해당 System 이 깨지기 쉽고, 유지하기 힘들다.*

이를 실현하기 위한 방법은 다음과 같다.

- 객체는 자신과 직접 관련된 객체와만 상호작용해야 한다.
- 객체는 자신이 가지고 있는 메서드를 통해서만 다른 객체와 상호작용해야 한다.
- 메서드 매개변수: 객체는 자신이 필요로 하는 정보를 메서드 매개변수로 전달받아야 한다. 객체는 매개변수로 전달받은 다른 객체의 메서드를 호출하여 상호작용할 수 있다.
- 객체는 다른 객체의 메서드를 호출한 결과를 반환받아 사용할 수 있다. 반환된 객체와는 직접적인 상호작용이 가능하다.

강의 들으며

내부가 굉장히 복잡하게 이루어져 있을 때, 단순하게 소통할 수 있는 Interface - 창구를 마련하여, 이를 사용하는 사람이 쉽게 사용할 수 있도록 하는 것.

Facade = Front of a building facing the public way or space (Architecture - 건축에서 의미하는 바.)

머 Sub System 이 있을 수도 있고~
Client 의 변경 없이, Facade Class 내부의 내용물만 바꿀 수 있다는 점이 굉장히 매력적이긴 하다!

Facade Interface with a few simple methods
Facade를 구현한 Class 내에 Subsystem class 와 method 들이 존재하고, Facade interface 의 method를 구현하기 위해 해당 subsystem 의 내용을 이용.

Subsystem Method 등도 간편하게 사용할 수 있도록 함.

Facade Class 를 아예 따로 만들어서, 복잡한 Class 정보를 다 안에 담아두고, 대문으로써 쓰는 게 더 일반적인 경우인 듯! (예시로서 제시된 게 이러한 경우.)\

Facade Pattern 이란 무엇이냐? Unified Interface for set of sub systems.
여기서 Interface 는 맞닿아 있는 부분, 상호작용의 지점이라는 뜻.

상호작용 할 수 있는 High Level 의 부분으로서 존재하며, Subsystem 등에 복잡하게 관여할 필요 없이 쉽게 해당 System (기능 묶음) 을 사용할 수 있도록 함.

> *개인적으로, 다양한 기능을 묶어 사용하는 Module 로 구성된 게 많은 환경에서, 가장 권장되는 구조가 아닐까 한다!*

Facade 는 다음과 같은 Design Principle 을 따른다.

> *Design Principle 4 : The Pirnciple of Least Knowledge (Law of Demeter)*
> *연결을 가능한 만큼 줄여라. 최대한 Loose 하게 Coupling 될 수 있도록 만들라.*
> *많은 Dependency 를 가질 수록, 해당 System 이 깨지기 쉽고, 유지하기 힘들고 그렇다. 이를 지킬 수 있는 Guideline 은 다음과 같다.*
> *Object 스스로만 호출하는 Method*
> *Object 를 Method의 parameter 로 호출할 수 있도록 하고.*
> *그렇게 하라.*

세부에 있는 것일수록 덜 연결되도록 하라.

Tight Coupling
Loose Coupling