**Design Pattern**

구체적인 구현을 실현하기 위한 사고방식, 해법.

https://en.wikipedia.org/wiki/Design_Patterns#Patterns_by_Type

Architecture, Modeling, DesignPattern

#### 기초 용어

SOLID : 객체 지향 프로그래밍 및 설계의 다섯 가지 기본 원칙 unified_modeling_language

- 상속(Inherit)와 위임(Delegation)
  - 상속은 Tight coupling, 위임은 Loose coupling.
  - 클래스 관계를 동적으로 바꾸 싶을때, 느슨한 연결(Loose Coupling)을 할 때는 위임. 그 반대의 경우 상속.
- 투과적 인터페이스(API)
  - 연쇄적인 상속으로 하위 클래스들에게 API(Method)들이 감추어지지 않고 보이는 것. 상속/구현을 통해 투영/투과.
  - Adapter Pattern
  - Decorator Pattern
  - Proxy Pattern
- Double Dispatch
  - 서로 상호적으로 한쌍을 이루어 Method Logic(Algorithm, Message)을 처리하는 관계
  - Visitor Pattern

#### 분류

기본적으로 Yuki Hiroshi에 의한 분류로 서술.

| **GoF에 따른 분류** | | |
|---|---|---|
| <hi yellow>Creational</hi> | <hi cyan>Structural</hi> | <hi pink>Behavioral</hi> |

#### 기초

- <hi pink>Iterator</hi>
- <hi cyan>Adapter</hi>

#### 하위 클래스에게 위임

상속과 관련된 디자인 패턴

- <hi pink>Template method</hi>
- <hi yellow>Factory method</hi>

#### 인스턴스 생성

- <hi yellow>Singleton</hi>
- <hi yellow>Prototype</hi>
- <hi yellow>Builder</hi>
- <hi yellow>Abstract factory</hi>

#### 분리해서 생각

복잡하게 뒤섞이기 쉬운 프로그램을 분리해서 생각

- <hi cyan>Bridge</hi>
- <hi pink>Strategy</hi>

#### 동일한 것으로 간주하기

언뜻 보기에 서로 다른 것을 **통일적으로 조작**할 수 있도록 하거나, **취급 방법을 바꾸지 않고** 기능을 추가하는 패턴과 위임에 대한 내용

- <hi cyan>Composite</hi>
- <hi cyan>Decorator</hi>

#### 구조를 돌아다니기

- <hi pink>Visitor</hi>
- <hi pink>Chain of responsibility</hi>

#### 단순화

클래스들이 복잡한 관계에 있을 때, 단순하게 만들기

- <hi cyan>Facade</hi>
- <hi pink>Mediator</hi>

#### 상태 관리

- <hi pink>Observer</hi>
- <hi pink>Memento</hi>
- <hi pink>State</hi>

#### 낭비 없애기

- <hi cyan>Flyweight</hi>
- <hi cyan>Proxy</hi>

**클래스로 표현**

명령, 문법규칙과 같은 이외의 것을 클래스로 표현

- <hi pink>Command</hi>
- <hi pink>Interpreter</hi>

From:
https://wiki.ledyx.xyz/ - **Ledyx WIKI**

Permanent link:
**https://wiki.ledyx.xyz/doku.php?id=design_pattern:design_pattern&rev=1517234391**

Last update: **2026/10/24 07:03**