Design Pattern

유튜브 주소: https://youtu.be/k_ncY3vLOWg

IT융합공학부 이준희

<목차>

1. 디자인 패턴(Design Pattern) 이란?

- 소프트웨어를 설계할 때 자주 발생하는 문제들에 대한 **"재사용 가능한 해결책**“
- 기존 환경 내에서 반복적으로 일어나는 문제들을 어떻게 풀어나갈 것인가에 대한 일종의 솔루션
- **Design Patterns:** 에릭 감마(Erich Gamma), 리차드 헬름(Richard Helm), 존 블리자이드(John Vlissides), 랄프 존슨(Ralph Johnson) **'GoF(Gang of Four) book**’, 1995년

Design Pattern이 주는 이점

- 개발자 간의 공통 언어 역할을 하여 개발자간의 **원활한 의사소통**을 가능하게 한다.
- 재사용 가능한 해결책을 사용하므로 **개발시간을 단축**을 가져올 수 있다.
- 공통적으로 자주 나타나는 문제에 대한 **재사용 가능한 해결책**
- 불필요한 논쟁 시간 줄임

Design Pattern의 종류

- **생성(creational) 패턴**: 객체의 생성과 관련된 패턴

  객체의 생성과 조합을 캡슐화해 특정 객체가 생성되거나 변경되어도 프로그램 구조에 영향을 크게 받지 않도록 유연성을 제공

- **구조(structural) 패턴**: 클래스나 객체들을 조합해 더 큰 구조로 만들 수 있게 해주는 패턴

  예를 들어 서로 다른 인터페이스를 지닌 2개의 객체를 묶어 단일 인터페이스를 제공하거나 객체들을 서로 묶어 새로운 기능을 제공하는 패턴

- **행위(behavioral) 패턴**: 객체나 클래스 사이의 알고리즘이나 책임 분배에 관련된 패턴

  한 객체가 혼자 수행할 수 없는 작업을 여러개의 객체로 어떻게 분배하는 지, 또 그렇게 하면서도 객체 사이의 결합도를 최소화하는 것에 중점을 둔다.

Design Pattern의 종류

표 4-1 GoF 디자인 패턴의 분류

	생성 패턴	구조 패턴	행위 패턴
패턴 이름	추상 팩토리(Abstract Factory) 빌더(Builder) 팩토리 메서드(Factory Method) 프로토타입(Prototype) 싱글턴(Singleton)	어댑터(Adapter) 브리지(Bridge) 컴퍼지트(Composite) 데커레이터(Decorator) 퍼사드(façade) 플라이웨이트(Flyweight) 프록시(Proxy)	책임 연쇄 (Chain of Responsibility) 커맨드(Command) 인터프리터(Interpreter) 이터레이터(Iterator) 미디에이터(Mediator) 메멘토(Memento) 옵서버(Observer) 스테이트(State) 스트래티지(Strategy) 템플릿 메서드(Template Method) 비지터(Visitor)

2. **싱글톤 패턴 (Singleton Pattern)** 이란?

- 디자인 패턴에서 **생성(creational) 패턴**의 일부

- 특정 클래스가 인스턴스화될 때 해당 클래스의 인스턴스가 **하나만 생성되도록 보장**하는 패턴

전역 변수를 사용하지 않고 객체를 하나만 생성하도록 하며, 생성된 객체를 어디에서든지 참조할 수 있도록 하는 패턴

- 즉, 싱글톤 패턴을 사용하면 특정 클래스의 객체가 **시스템 내에서 유일하게 존재**

Singleton Pattern의 목적

- **유일한 인스턴스 유지**

  해당 클래스의 인스턴스가 오직 하나만 존재하도록 보장하여,
  여러 곳에서 해당 객체를 공유하거나 조작할 수 있게 한다.

- **전역 접근 지원**

  어디서든 해당 객체에 접근할 수 있도록 하여
  중요한 리소스나 상태를 효율적으로 관리하고 사용할 수 있도록 한다.

Singleton Pattern의 종류

- Eager Initialization
- Lazy Initialization
- DCL(Double Checked Locking)
- Initialization on demand holder idiom

3. **SOLID (5가지 설계원칙)**

- ‘로버트 마틴’이 주창한 **객체 지향 프로그래밍의 다섯 가지 중요한 설계 원칙**

- 이 원칙들은 코드의 **유지보수성, 확장성 및 재사용성**을 향상시키고,
  객체 지향 소프트웨어 개발에서 더 좋은 설계를 지원하기 위해 개발

※ 좋은 소프트웨어란?

⇒1. 이해하기 쉽다. (가독성 측면)

2. 변경하기 용이 (유지보수 측면)

3. 확장이 용이 (확장성 측면)

SOLID의 종류

- SRP(단일 책임 원칙, Single Responsibility Principle)
- OCP(개방 폐쇄 원칙, Open Closed Principle)
- LSP(리스코프의 대입 원칙, Liskov Substitution Principle)
- ISP(인터페이스 분리 원칙, Interface Segregation Principle)
- DIP(의존성 역전 원칙, Dependency Inversion Principle)

4. OCP(개방 폐쇄 원칙, Open Closed Principle)

- 객체 지향 프로그래밍의 SOLID 원칙 중 하나

- 이 원칙은 소프트웨어 컴포넌트(클래스, 모듈, 함수 등)가 **확장에는 열려 있어야 하고, 변경에는 닫혀 있어야 함**을 나타낸다.

- 즉, OCP는 **기존의 코드를 변경하지 않으면서 새로운 기능을 추가할 수 있도록 설계하는 원칙**이다.

Q & A