디자인 패턴

프로그래밍 할때에 필요한 설계 지식과 코드의 구조들을 일정한 형태로 만들어 재 이용하기 편리하게 만든 일정한 패턴

1. 생성패턴

인스턴스를 만드는 절차를 추상화 하는 패턴

객체를 생성/합성 하는 방법이나 객체 표현 방법 과 시스템을 분리 해준다.

또한 이를 완전히 가려준다.

'무엇이생성?', '어떻게 생성?', '언제 생성?', '누가생성?' → 알수없음.

① Singleton

- 오직 한개의 인스턴스만을 갖도록 하며, 이에 대한 전역적인 접근을 허용.
- 특정 클래스의 인스턴스가 반드시 하나, 여러곳 사용 하는 경우 사용 (공유)
- 메모리 낭비 방지
- 단점: 상속 X / 강제적인 전역 상태 / 객체가 하나인 것을 보장 X
  - 상속, 다형성 X
  - 은닉화 ↓
  - ex) 멀티 쓰레드 → 그래이상의 객체 생성.

② Prototype

- 인스턴스들이 서로 다른 상태 값 또는 서로다른 조합으로 지속적으로나 간헐적으로 필요 할때, 나중에 인스턴스 생성을 위해 복제의 원본이 되어줄 원형 인스턴스를 정의
- Java의 clone() 메소드.
- 프로토 타입 패턴에 또다른 객체가 있을때 서로 공유 (얕은 복사)

③ Factory Method

- 객체를 생성하기 위해 인터페이스를 정의하지만 어떤 클래스의 인스턴스를 생성할지는 서브 클래가 결정
- 책임의 분리 가 필요. / 매개변수로 분리 처리

④ Builder

- 복잡한 객체를 생성하는 방법과 표현하는 방법을 정의하여 동일한 절차로 서로다른 표현이라도 이를 생성할수 있는 동일한 절차를 제공한다.
- 조립될 각 객체 들의 구체적인 클래스나 객체 들의 조립 방법이 서로 다르더라도 내부객체들이 어떻게 생성되고 조립되는지 제공 해야할 때.
- 생성될 객체 들로 최종적인 객체가 만들어지는 과정이 동일한 절차를 제공해야 하는 경우 빌더패턴을 사용.

⑤ Abstract Factory

- 상세화된 서브클래스를 정의하지 않고도 서로 관련성이 있거나 독립적인 여러 객체의 군을 생성하기 위한 인터페이스를 제공
- 여러 관련된 제품들이 하나의 군을 이루고, 여러 제품 군 중에서 하나를 선택하여 사용하고자 할 때
- 서브 클래싱은 해줄수 없이 많다 → 새로운 것이 추가되면 인터페이스에 메소드를 추가해주어야함

2. 구조패턴

더 큰 구조를 형성하기 위해 어떻게 클래스나 객체를 합성하는지에 관련된 패턴

① Adapter

적응대상 클래스에서 특성들을 상속받아 적응대상자에게 정의된 인터페이스를 마치 자신이 제공하는것처럼 하는 것.

- 기존클래스를 사용하고 싶은데 인터페이스가 맞지 않을 때
- 이미 만들어진 재사용 하고 싶은, 재사용 가능한 라이브러리를 수정할수 없을때.
- 이미 존재하는 여러 서브클래스를 사용해야 하는데, 이들이 인터페이스를 개조 하는것이 불가능할때

② Bridge

구현부에서 추상부를 분리하여, 이들이 독립적으로 다양성을 가질 수 있도록 하는 패턴

- 구상적 개념과 이에 대한 구현 사이의 지속적인 종속관계를 피하고 싶을때
- 추상적 개념과 구현 모두가 독립적으로 서브클래싱을 통해 확장되어야 할때

③ Composite

복잡한 전체의 계층을 표현하기 위해 객체들을 모아 트리구조로 구성하는 위한패턴

- 부분-전체의 객체 계통을 표현하고 싶을때
- 사용자가 객체의 합성으로 생긴 복합객체와 개개의 객체 사이의 차이를 알지 않고도 자기일을 할 수 있도록 만들고 싶은
- 사용자는 복합구조의 모든 객체를 똑같이 취급하게 된다.

④ Decorator

객체에 동적으로 새로운 책임을 추가할 수 있게 한다.

기능을 추가하려면, 서브클래스를 생성하는 것보다 융통성 있는 방법을 제공한다.

- 기본기능 + Decorator 클래스정의 → 추가기능의 조합을 설계

⑤ Facade

"건물의 정면" 이미, 어떤 소프트웨어의 다른 커다란 코드 부분에 대하여 간략화된 인터페이스를 제공

-퍼사드 객체는 복잡한 소프트웨어 라이브러리 안팎 코드에 의존성을 일으 강소 시켜주고 복잡한 소프트웨어를 사용하기 쉽게 간단한 인터페이스를 제공

⑥ Flyweight

어떤 클래스의 인스턴스 하나만 가지고 여러 개의 "가상 인스턴스"를 제공하고 싶을때 사용하는 패턴

new 연산자를 통한 메모리 낭비를 줄이는 방식

⑦ Proxy

실제 기능을 수행하는 Object 대신 가상의 Object를 사용하여 중간의 조증을 제어하는 패턴

어떤 클래스의 객체 생성이 오래걸리는 경우 그 일을 별도로 Proxy 클래스에서 처리하는 부분을 처리하고, 한 두 필요한 부분만 실제 클래스에서 객체를 생성하는 방식

3. 행동패턴

객체나 클래스 사이의 알고리즘이나 책임분배에 관련된 패턴

한 객체가 수행할 수 없는 작업을 여러개의 객체로 어떻게 분배하여, 객체 사이의 결합도 최소화에 중점

패턴 적용 → 클래스 vs 패턴 적용 → 객체

① Chain of Responsibility

요청기 대한 처리객체를 Chain화 → 결합도 ↓

Request → 처리X → 처리X → ···· 처리O.

여러개의 객체중 어떤것이 요구를 처리 할 수 있는지 사전에 알수 X 사용.

② Command

해당 요청이 따라야하는 기능들은 캡슐화한 객체에 저장하여 실행 할 수 있게 해주는 디자인

요청에 따라야하는 기능 추가, 변경/삭제 추가 경우 변경이 요청되는 클래스를 변경하지 않고 수정할때 매우 유용

③ Interpreter

언어 문법이나 표현을 평가 할 수있는 방법을 제공

특정 컨텍스트를 "해석" 하도록 지시하는 표현 인터페이스를 구현하는것

SQL 구문 분석, 기능처리 엔진

④ Iterator

반복자다의 의미, 어떠한 객체의 집합을 순서대로 열거를 처리할 수 있게 해주는 패턴

컨테이너 구현 방법은 노출 X, 그 집합체 안의 모든 항목 접근 가능하게 해주는 패턴

⑤ Observer

객체의 상태변화를 관찰하는 관찰자 객체를 생성하여 사용하는 패턴

객체 변화 감지 → 종속 객체 자동 변화 통지 → 연결 유지 : 일대 다의 의존성