6장. 스프링이 사랑한 디자인 패턴

요리	*객체 지향 프로그래밍(OOP)*
요리도구	4대 원칙(캡! 상추다)
요리도구 사용법	설계 원칙(SOLID)
레시피	디자인 패턴

[표 6-1] 요리와 OOP 비교

■ 디자인 패턴
- 실제 개발 현장에서 비즈니스 요구 사항을 프로그래밍으로 처리하며 만들어진 해결책 중 많은 이들이 인정한 방식을 정리한 것임
- 객체 지향 특성과 설계 원칙을 기반으로 구현됨

■ 스프링 프레임워크

- 자바 엔터프래즈 개발을 편리하게 해주는 오픈소스 경량급 애플리케이션 프레임워크

- 스프링은 객체 지향 특성과 설계 원칙을 극한까지 적용한 프레임워크
(가장 OOP 스럽다고 볼 수 있음)

- 디자인 패턴은 상속(extends), 인터페이스(interface/implements), 합성
(객체를 속성으로 사용)을 이용함

■ 어댑터 패턴(Adapter Pattern)

- 어댑터 직역 => 변환기(converter)

- 역할 : 서로 다른 두 인터페이스 사이 통신이 가능하게 하는 것

- 대표적 예시
ODBC, JDBC는 어댑터 패턴을 이용해 다양한 DB 시스템을 단일한 인터페이스로 조작할 수 있게 해줌

- SOLID 원칙 중 개방 폐쇄 원칙[OCP]을 활용한 설계 패턴

- 객체를 속성으로 만들어 참조하는 디자인 패턴

- 호출 당하는 쪽 메서드를 호출하는 쪽 코드에 대응하도록 중간에 변환기를 통해 호출

■ 프록시 패턴(Proxy Pattern)

- Proxy = 대리자, 대변인

- 실제 서비스 객체가 가진 메서드와 같은 이름의 메서드를 사용하는데,
  이를 위해 인터페이스를 사용함

  인터페이스를 사용하면 서비스 객체가 들어갈 자리에 대리자 객체를 대신 투입하여
  클라이언트 쪽에서는 실제 서비스 객체를 통해 메서드를 호출하고 반환 값을 받는지
  대리자(Proxy)를 통해 작업되는지 전혀 모르게 처리할 수 있음

- Proxy 패턴의 중요 포인트

  (1) Proxy는 실제 서비스와 같은 이름의 메서드를 구현함.
  인터페이스를 상속 받기 때문

  (2) Proxy는 실제 서비스에 대한 참조변수를 가짐 (합성)
  실제 서비스가 구현된 클래스의 인스턴스를 생성함

  (3) Proxy는 실제 서비스의 이름을 가진 가진 메서드를 호출하고
  값을 클라이언트에게 돌려줌

  (4) Proxy는 실제 서비스 메서드 호출 전후에 별도의 로직을 수행할 수도 있음

- 어렵지? 청와대 대변인 생각해보자.
  공식 석상에서 대리인은 본인 이야기를 하지 않고 입장을 대변해서 이야기 함
  Proxy 패턴도 마찬가지로 보면, 실제 서비스 메서드의 반환값을 그대로 반환함

- Proxy 패턴은 반환 값을 변화하는 것을 목적으로 하지 않고
  제어 흐름을 바꾸거나 다른 로직을 수행하기 위해 사용됨

- 한 줄 정리. “제어 흐름을 조정하기 위한 목적으로 중간에 대리자를 두는 패턴”

- SOLID의 OCP, DIP에 의해 설계된 패턴임.

■ 데코레이터 패턴(Decorator Pattern)

- Decorator = 장식자

- 원본에 장식을 더하는 패턴

- Proxy 패턴과 구현 방법은 동일하나,
  Proxy 패턴은 반환 값을 변화 없이 반환하는 반면
  Decorator 패턴은 반환 값에 장식을 덧 입힘

- 둘 차이를 한 번 더 정리하자면

  Proxy 패턴 : 제어 흐름 변경 혹은 별도 로직 처리를 목적으로 함
  : 클라이언트가 받는 반환 값을 특별한 경우가 아니라면 변경 안 함

  Decorator : 클라이언트가 받는 반환 값에 장식을 더함

- 중요 포인트

  (1) Decorator는 실제 서비스와 같은 이름의 메서드를 구현함.
  이때, 인터페이스를 상속 받아 사용함

  (2) Decorator는 실제 서비스에 대한 참조 변수를 가짐 (합성)
  인터페이스 타입으로 Service 인스턴스를 생성함

  (3) Decorator는 실제 서비스의 같은 이름을 가진 메서드를 호출하고,
  반환 값에 장식을 더해 클라이언트에 반환함

  (4) Decorator는 실제 서비스 메서드 호출 전 후 별도의 로직을 수행할 수 있음

- 요약. “메서드 호출의 반환 값에 변화를 주기 위해 중간에 장식자를 두는 패턴”

- SOLID 원칙 중, OCP와 DIP가 적용된 패턴

■ 싱글턴 패턴(Singleton Pattern)

- 인스턴스를 하나만 만들어 사용하기 위한 패턴

- 하나만 생성한 뒤 재사용

- 커넥션 풀, 디바이스 설정 객체 등 여러 인스턴스를 생성하면 자원낭비가 발생하며 예기치 못한 결과가 발생할 수 있음

- 싱글턴 패턴을 적용할 경우 의미상 두 개의 객체가 존재할 수 없으며 필요 요소는 다음과 같음

  (1) 생성자 제약: new를 생성할 수 없도록 생성자에 private 접근자 지정

  (2) 유일한 단일 객체 반환을 위한 정적 메서드(static method) 필요
  —> 직접적으로 생성자를 통해 생서하는 것이 아닌
  객체를 반환하는 정적 메서드를 이용

  (3) 유일한 단일 객체를 참조할 정적(static) 참조 변수 필요

- 인스턴스는 static에 전처리 된 후 Heap 영역의 참조 주소를 공유해 전체 코드에서 단 하나의 인스턴스만 생성해 활용할 수 있는 것임

- 단일 객체는 결국 공유 객체로 이용되기에 속성을 갖지 않는 것이 정석임 단일 객체가 속성을 갖게 되면 하나의 참조 변수가 변경한 단일 객체 속성이 다른 참조 변수에 영향을 미치기 때문임

- 고로 변화하는 값이 아닌 다른 단일 객체에 대한 참조를 단일 객체의 속성으로 갖는 것은 문제되지 않음

- Singleton Pattern 특징

  (1) private 생성자를 가짐

  (2) 단일 객체 참조 변수를 정적 속성으로 가짐

  (3) 단일 객체 참조 변수가 참조하는 단일 객체를 반환하는 getInstance() 정적 메서드를 가짐

  (4) 단일 객체는 쓰기 가능한 속성을 갖지 않는 것이 정석임

- 한 줄 요약. “클래스 인스턴스. 즉, 객체를 하나만 만들어 사용하는 패턴”

■ 템플릿 메서드 패턴(Template Method Pattern)

- 상위 클래스의 공통 로직을 수행하는 '템플릿 메서드'와
  구현을 강제하는 추상 메서드, 선택적 오버라이딩인 Hook(갈고리) 메서드로
  하위 클래스에서 메서드를 일정 패턴을 유지하며 재정의할 수 있는 패턴임

- 구성 요소 정리

  (1) 템플릿 메서드
  공통 로직을 수행하며, 로직 중 하위클래스에서 오버라이딩 당한
  추상 메서드 / 훅 메서드 호출

  (2) 템플릿 메서드에서 호출하는 추상 메서드
  하위 클래스가 반드시 오버라이딩 해야 함

  (3) 템플릿 메서드에서 호출하는 훅(Hook, 갈고리) 메서드
  하위 클래스가 선택적으로 오버라이딩 함

- 한 줄 요약.
  "상위 클래스의 견본 메서드에서 하위 클래스가 오버라이딩한 메서드를 호출하는 패턴"

- SOLID 중, DIP 활용

■ 팩터리 메서드 패턴(Factory Method Pattern)

- Factory = 공장, 객체 지향에서의 Factory는 객체를 생성함

- Factory Method
  객체 생성을 반환하는 메서드

  Factory Method Pattern
  하위 클래스에서 팩터리 메서드를 오버라이딩 해 객체를 반환하게 하는 것

- 요약. "오버라이드 된 메서드가 객체를 반환하는 패턴"

- SOLID 중, DIP가 활용되고 있음

■ 전략 패턴(Strategy Pattern)

- 구성 요소

  (1) 전략 메서드를 가진 전략 객체

  (2) 전략 객체를 사용하는 컨텍스트(전략 객체의 사용자/소비자)

  (3) 전략 객체를 생성해 컨텍스트에 주입하는 클라이언트
  (제 3자,전략 객체 공급자)

- 클라이언트는 다양한 전략 중 한 가지를 선택해 생성한 뒤 컨텍스트에 주입함

- 전략 패턴을 이용해 전략을 변경해가며 매개변수로 인스턴스를 넘겨 다양하게 컨텍스트(전략 객체의 사용자)를 실행할 수 있음

- 같은 문제의 해결책으로 상속을 이용하는 템플릿 메서드 패턴과 객체 주입을 통한 전략 패턴 중 선택하고 적용할 수 있음

- 자바의 단일 상속 속성상, 템플릿 메서드 패턴보다 전략 패턴이 더 많이 활용됨

- 요약. "클라이언트가 전략을 생성해 전략을 실행할 컨텍스트에 주입하는 패턴"

- SOLID의 OCP과 DIP가 적용됨

■ 템플릿 콜백 패턴(Template Callback Pattern - 견본/회신 패턴)

- 전략 패턴의 변형

- 스프링의 의존성 주임(DI, Dependency Injection)에서 사용되는
  특별한 형태의 전략 패턴

- 전략 패턴과 모든 것이 동일하나, 전략을 익명 내부 클래스로 정의해 사용함

- 내부 클래스와 익명 내부 클래스 [꼭 꼭 추가하기]

- 리팩토링 된 템플릿 콜백 패턴

  전략을 생성하는 코트가 컨텍스트(전략 사용자) 내부로 들어옴
  중요한 포인트는 중복되는 부분을 콘텍스트로 옮겨 코드의 중복성을 줄이는 것임

- 요약. “전략을 익명 내부 클래스로 구현한 전략 패턴”

- SOLID 중, OCP와 DIP가 적용된 설계 패턴