1️⃣

Section1. 객체 지향 설계와 스프링

객체 지향 특징

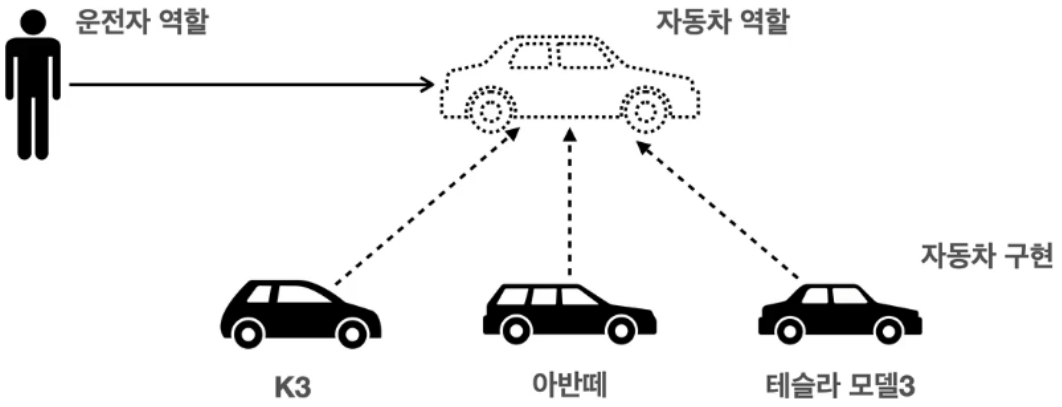
- 추상화
- 캡슐화
- 상속
- 다형성

객체 지향 프로그래밍의 개념

- 컴퓨터 프로그램을 여러 개의 독립된 단위, 즉 **'객체'** 들의 **모임**으로 파악하고자 하는 것
- 각각의 **객체**는 **메시지**를 주고받고, 데이터를 처리할 수 있다.**(협력)**
- 프로그램을 **유연**하고 **변경**이 용이하게 만들기 때문에 대규모 소프트웨어 개발에 사용된다.

다형성의 실세계 비유

- `역할`과 `구현`으로 세상을 구분



  - 운전자는 자동차의 역할(인터페이스)만 이해하고 있더라도 운전에 지장은 없다.
    - 자동차 세상을 무한히 확장 가능

→ 클라이언트에 영향을 주지 않고 새로운 기능을 제공할 수 있음

→ 따라서 새로운 자동차가 나온다고 하더라도 클라이언트는 익숙하게 사용 가능

역할과 구현을 분리

- 단순, 유연, 변경 ↑
- 장점
  - 클라이언트는 인터페이스만 알면 된다.
  - 내부 구조를 몰라도 되며, 구조가 변경되어도 영향을 받지 않는다.

다형성의 본질

- 클라이언트를 변경하지 않고, 서버의 구현 기능을 유연하게 변경할 수 있다.

한계

- 인터페이스 자체가 변하면 client, server 모두 큰 변경 발생

객체 지향 설계의 5가지 원칙

단일 책임 원칙 SRP(Single Responsibility Principle)

- 한 클래스는 하나의 책임만 가져야 한다
- 중요한 기준은 변경이다. 변경이 있을 때 파급 효과가 적으면 단일 책임의 원칙을 따른다.
  - 예) UI 변경, 객체의 생성과 사용을 분리

개방 폐쇄 원칙 OCP(Open/Closed Principle)

- 소프트웨어 요소는 확장에는 열려 있으나 변경에는 닫혀 있어야 한다.
- 다형성을 활용해보자

💡 문제점

- MemberService 클라이언트가 구현 클래스를 직접 선택
  - `MemberRepository m = new MemoryMemberRepository();` // 기존 코드
  - `MemberRepository m = new JdbcMemberRepository();` // 변경 코드
- 구현 객체를 변경하려면 클라이언트 코드를 변경해야 한다.
- 분명 다형성을 사용했지만 OCP 원칙을 지킬 수 없다.
- 이 문제를 해결하기 위해서는

  → 객체를 생성하고, 연관관계를 맺어주는 별도의 조립, 설정자가 필요하다.

리스코프 치환 원칙 LSP(Liskov Substitution Principle)

- 프로그램의 객체는 프로그램의 정확성을 깨뜨리지 않으면서 하위 타입의 인스턴스로 바꿀 수 있어야 한다.

인터페이스 분리 원칙 ISP(Interface Segregation Principle)

- 특정 client 를 위한 인터페이스 여러 개가 범용 인터페이스 하나보다 낫다.

의존관계 역전 원칙 DIP(Dependency Inversion Principle)

- 추상화에 의존해야지, 구체화에 의존하면 안된다.
- 구현체에 의존하게 되면 변경이 아주 어려워진다.
  - `MemberRepository m = new MemoryMemberRepository();` ← **DIP 위반**

정리

- 다형성 만으로는 OCP, DIP를 지킬 수 없다.

Spring

- 스프링은 다음 기술로 다형성 + OCP, DIP를 가능하게 지원
  - DI(Dependency Injection): 의존관계, 의존성 주입
  - DI 컨테이너 제공
- 클라이언트 코드의 변경 없이 기능 확장
- 쉽게 부품을 교체하듯이 개발