# 다형성과 설계 [Java]

| | |
|---|---|
| 소유자 | 종수 김 |
| 태그 | |

## [좋은 객체지향 프로그래밍이란?]

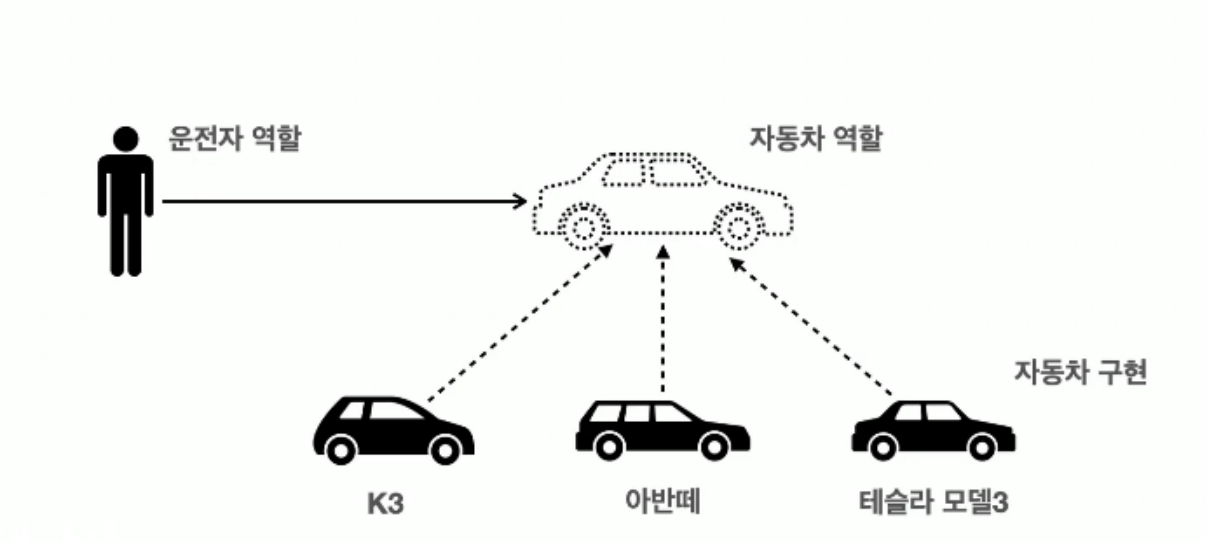
- 객체 지향의 특징
  - 추상화
  - 캡슐화
  - 상속
  - 다형성

객체 지향 프로그래밍이란, 프로그램을 '객체들의 모임'으로 설계하고 만드는 것.

- 객체는 메시지를 주고받고, 본인의 데이터를 처리
- 객체를 블럭을 조립하듯 조립하고,
  컴퓨터 부품을 갈아끼는 등, 객체를 조립함으로써 유연하게 변경하면서 개발
  - '다형성'

## [다형성의 실세계 비유]

'역할'과 '구현'으로 세상을 구분
역할 : 인터페이스
구현 : 인터페이스를 구현한 클래스




자동차라는 역할을 구현한 K3, Sonata등 구현 객체가 바뀐다고 운전을 못하진 않음.

- K3, Sonata라는 실제 모델(구현체)이 자동차라는 설계도(인터페이스)를 기준 만들어졌기에 유연해지고, 변경이 편해지고, 클라이언트가 내부 구조를 몰라도 되기에

[장점]

- 클라이언트는 대상의 역할(인터페이스)만 알면 됨.
- 클라이언트는 구현 대상의 내부 구조를 몰라도 됨.
- 클라이언트는 구현 대상의 내부 구조가 변경되더라도 인터페이스에 의존하므로 영향을 받지 않는다.
- 클라이언트는 구현 대상 자체를 변경해도 영향을 받지 않는다.

* 객체를 설계할 때 역할과 구현을 명확히 분리.
* 객체 설계시 역할(인터페이스)을 먼저 부여하고, 그 역할에 맞는 구현 클래스를 설계.

## [다형성의 본질]

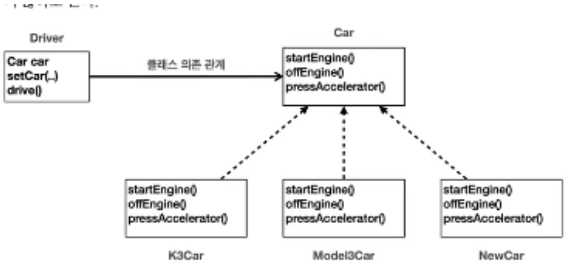
- 인터페이스를 구현한 객체 인스턴스를 **'실행 시점'**에 **'유연'**하게 변경
- 다형성의 본질을 이해하려면 **'협력'** 이라는 객체 사이의 관계에서 시작해야 함.
- **클라이언트를 변경하지 않고, 서버의 구현 기능을 유연하게 변경할 수 있다.**

**역할에 의존하므로, 역할(인터페이스) 자체가 변하면 클라이언트, 서버 모두에 큰 변경이 발생.**

> 정리
> 1. 다형성이 가장 중요!
> 2. 디자인 패턴 대부분은 다형성을 활용
> 3. 스프링 핵심, 제어의 역전(IoC), 의존관계 주입(DI)도 결국 다형성을 활용
> 4. 다형성을 잘 활용하면 구현을 편리하게 변경할 수 있음.

## [OCP (Open-Closed Principle)원칙]

- Open for extension
  - 새로운 기능의 추가나 변경 사항이 생겼을 때, 기존 코드는 확장할 수 있어야 한다.
- Closed for modification
  - 기존의 코드는 수정되지 않아야 한다.
- 확장에는 열려있고, 변경에는 닫혀있다




인터페이스에 의존하고 있으므로, 새로운 차량을 추가해도 Car 인터페이스를 구현해서 기능을 추가할 수 있다. (Open)

이 때, Car 인터페이스를 사용하는 메인 클라이언트 코드(Driver)는 변경하지 않아도 된다. (Closed)
* 물론 Main에선 변경이 발생하나, 이런 부분은 OCP를 지켜도 변경이 필요.
즉, 메인 클라이언트(Driver = 객체)는 변경이 없다는 것.

[참고]

- 전략 패턴(staragy pattern)
  - 알고리즘을 클라이언트 코드의 변경 없이 쉽게 교체할 수 있음.
  - Car 인터페이스가 바로 전략을 정의하는 인터페이스
  - 각각의 차량이 전략의 구체적인 구현
  - 위에서 Car를 구현한 객체가 전략 패턴을 사용한 것.