SOLID

CONTENTS

목차

CHAPTER 1 SOLID 이해를 위한, OOP 주요 내용

CHAPTER 2 SRP (Single Responsibility Principle)

CHAPTER 3 OCP (Open Closed Principle)

CHAPTER 4 LSP (Liskov Substitution Principle)

CHAPTER 5 ISP (Interface Segregation Principle)

CHAPTER 6 DIP (Dependency Inversion Principle)

OOP 주요 내용

SOLID 이해를 위한 OOP 주요 내용

객체지향프로그래밍

Object-Oriented Programming이란?

객체들로 복잡한 문제 -> 작은 문제로 분해

- 시스템의 복잡도를 줄이기 위한 분해(Decomposition)

작은 문제를 해결하기 위한 객체를 구성한다.
각 객체가 고유한 기능을 제공하고, 이러한 객체와 객체의 관계를 이용하여 프로그램을 완성한다.
이를 이용하여 객체의 재사용, 프로그램의 관리와 확장이 편리하게 된다.

좋은 소프트웨어 설계의 시작

모듈화

- 소프트웨어를 각 기능별로 분할, 설계 및 구현하는 기법
- 모듈화를 수행하면 복잡도가 감소되고, 변경과 구현이 용이하며 성능을 향상시킨다.
- 모듈간의 기능적 독립성을 보장한다.

결합도

- 모듈간의 상호 의존하는 정도, 연관관계.

응집도

- 하나의 모듈 안의 요소들이 서로 관련된 정도.

객체지향 4가지 기본원리

- **추상화(Abstraction)**

  중요하지 않는 자세한 사항은 감추고, 중요하고 필수적인 사항만 다룸으로써 복잡함을 관리할 수 있게 하는 개념

  중요여부의 판단은 업무나 관심사항에 따라 다르게 나타난다.

- **캡슐화(Encapsulation)**

  구현방법에 대한 자세한 사항은 블랙박스화 하여 드러내지 않고 외부로 노출된 인터페이스를 통해서만 사용할 수 있게 하는 개념

  인터페이스를 변경시키지 않는 한 사용자는 구현의 변경에 영향을 받지 않고 사용할 수 있고, 개발자는 내부 구조나 구현방법을 자유롭게 변경할 수 있다.

- **상속**

  클래스간의 관계를 계층구조화 하여 구체화와 일반화함.

  구체화 될수록 고유특징이 늘어나고, 일반화 될수록 더 많은 객체에 영향을 준다.

- **다형성**

  하나의 속성이나 행위가 여러 형태로 존재하는 것.

  Overriding 과 Overloading

객체지향 다섯가지 설계원칙 : SOLID 란?

- SOLID는 로버트 C. 마틴이 객체 지향 프로그래밍 및 설계의 다섯가지 기본원칙으로 제시한 것을 마이클 패더스가 알파벳 첫글자를 따서 소개한 것이다.

Single Responsibility Principle (SRP)	하나의 클래스는 하나의 책임만 가져야 한다.
Open/Closed Principle (OCP)	클래스는 확장에 대하여 열려 있어야 하고, 변경에 대해서는 닫혀 있어야 한다.
Liskov Substitution Principle (LSP)	기반 클래스의 메소드는 파생 클래스 객체의 상세를 알지 않고서도 사용될 수 있어야 한다.
Interface Segregation Principle (ISP)	클라이언트가 사용하지 않는 메소드에 의존하지 않아야 한다.
Dependency Inversion Principle (DIP)	추상화된 것은 구체적인 것에 의존하면 안 된다. (자주 변경되는 구체적인 것에 의존하지 말고 추상화된 것을 참조)

Github 에서 실습 소스코드 준비

소스코드 링크

https://github.com/mincoding1/SOLID

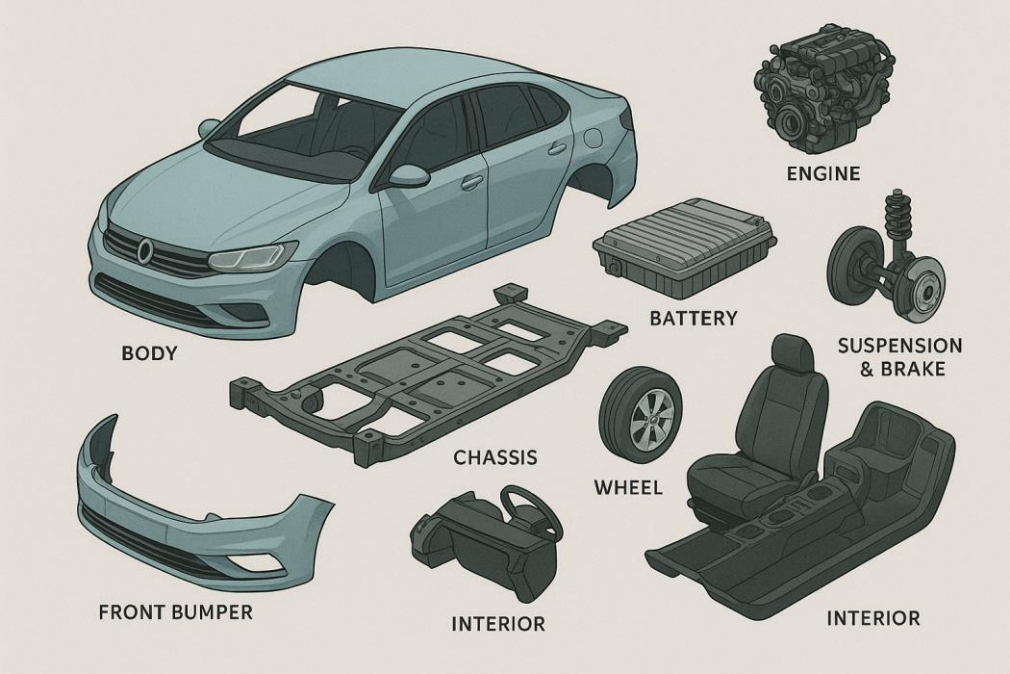
- Kata 출처 1 (Vehicle 컨셉) : https://github.com/bsferreira/solid
- Kata 출처 2 : https://github.com/mikeknep/SOLID

SRP

Single Responsibility Principle, 단일 책임 원칙

시스템에서 모듈의 책임

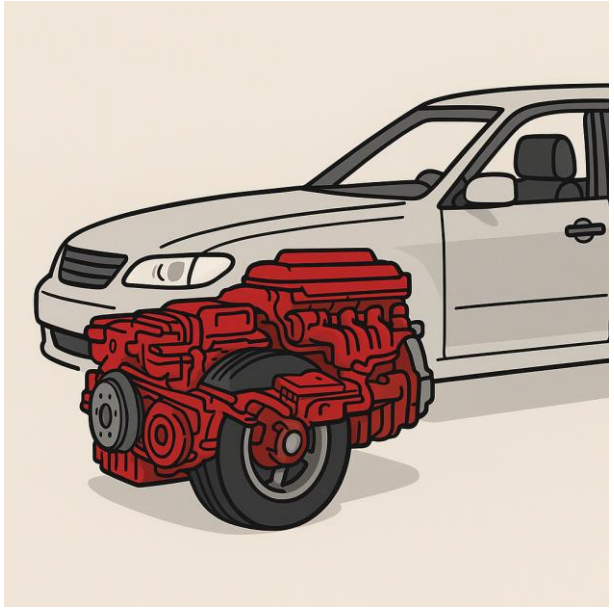
시스템은 **각자의 책임을 수행하는 모듈**로 구성되어 있다.



엔진은 동력을 생산하는 책임을 지닌다.
바퀴는 자동차의 실제 움직임을 발생시키는 책임을 지닌다.

책임이 섞여 있는 모듈

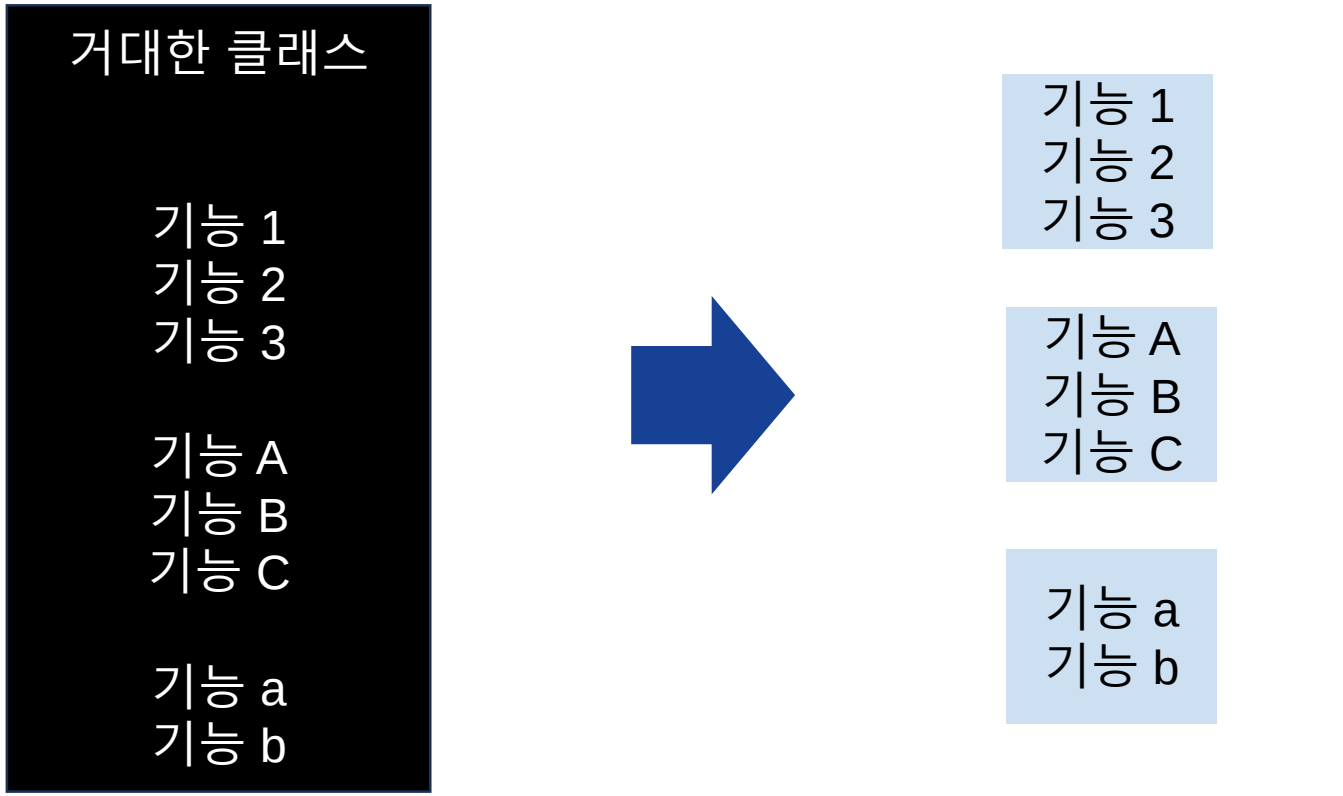
- 서로 다른 책임이 하나의 모듈에 섞여 있다면 유지보수에 문제점이 발생할 수 있다.
- 유지보수를 위해 **하나의 책임을 수행하는 모듈**로 분리를 해줘야 한다.



만약 엔진과 바퀴가 **하나의 모듈**로 되어있다면 어떤 문제가 생길 수 있을까?

SRP

거대한 클래스, 많은 일을 하고 있는 클래스를 분리한다



많은 기능들이 뒤엉켜서 한 곳에서 유지보수

독립적으로 유지보수

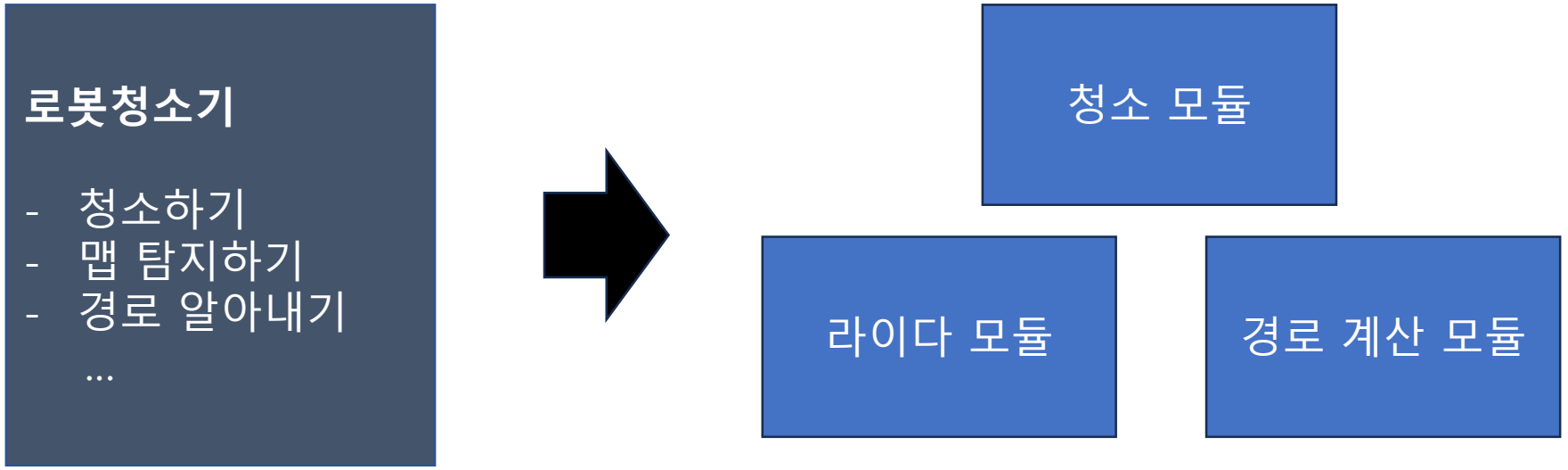
각자 변경되는 독립된 위치로 옮기기

로봇 청소기의 책임을 "청소 / 맵 탐지 / 경로 계산" 으로 구분 할 수 있다.

책임에 따라 변경을 다음과 같이 분류할 수 있게 된다.

- 청소에 관련된 요구사항 변경
- 맵 탐지에 관련된 요구사항 변경
- 경로 계산에 관련된 요구사항 변경

다른 책임과 관련된 변경에 영향 받지 않도록 코드를 분리해준다.



[도전] Vehicle 분리하기

Vehicle 클래스에서 Vehicle 의 책임이 아닌 부분을 분리한다.

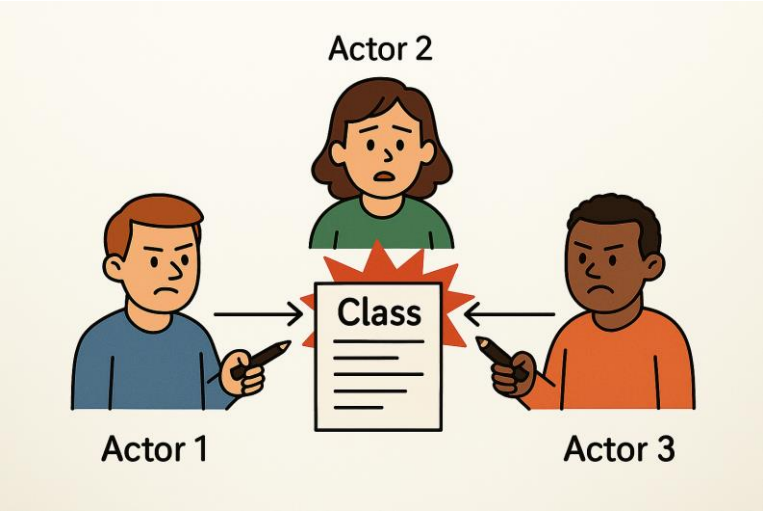
현재 Vehicle 클래스에 refuel 의 구체적인 로직이 들어가 있다 (주석으로 표기 되어 있음)
어떤 변경이 발생할 수 있을지 생각해보고 이에 맞게 SRP 를 적용해보자

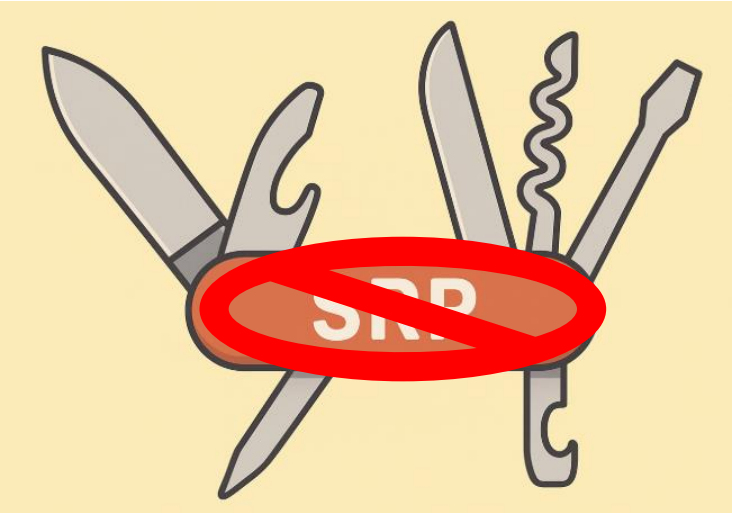
- 연료리필( )
- MAX량 확인( )
- 남은 연료 확인( )
- 연료 채우기( )
- 가속( )

[정리] SRP 정리

- **Single Responsibility Principle (SRP)**
  모든 클래스는 하나의 책임만 가지며, 클래스는 그 책임을 완전히 캡슐화해야 한다.
  클래스가 제공하는 모든 기능은 이 책임과 부합해야 한다.

- **로버트 C 마틴의 책임**
  로버트 C 마틴은 **하나의 책임**을 **하나의 변경하는 이유**라고 언급했다.





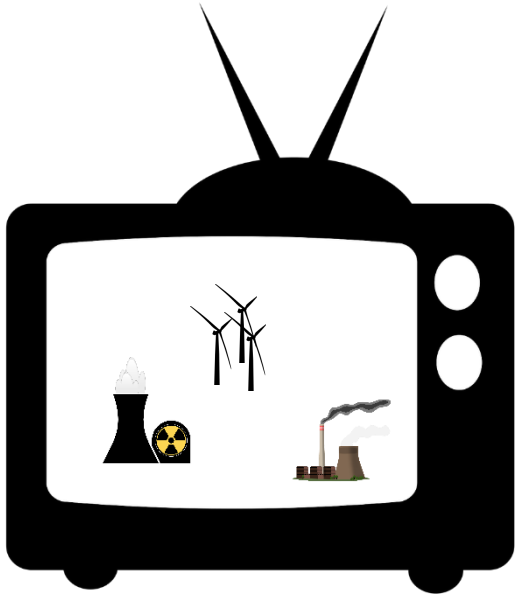
OCP

Open Closed Principle , 개방-폐쇄 원칙

열린마음 닫힌마음

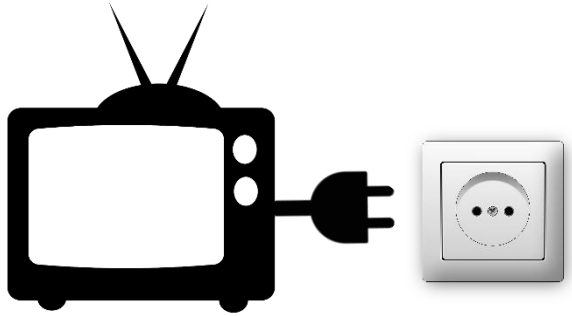
변경이 많은 곳을 분리

OCP는 변경이 많은 곳을 캡슐화를 통해 분리하여 새로운 기능을 추가하더라도 변경이 없는 구조로 만든다.

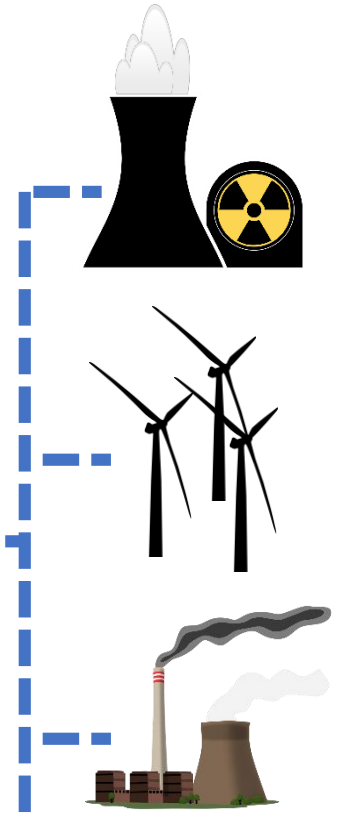


다양한 전기공급 이용
(TV 내부에 구현)

캡슐화



다른 클래스로 캡슐화

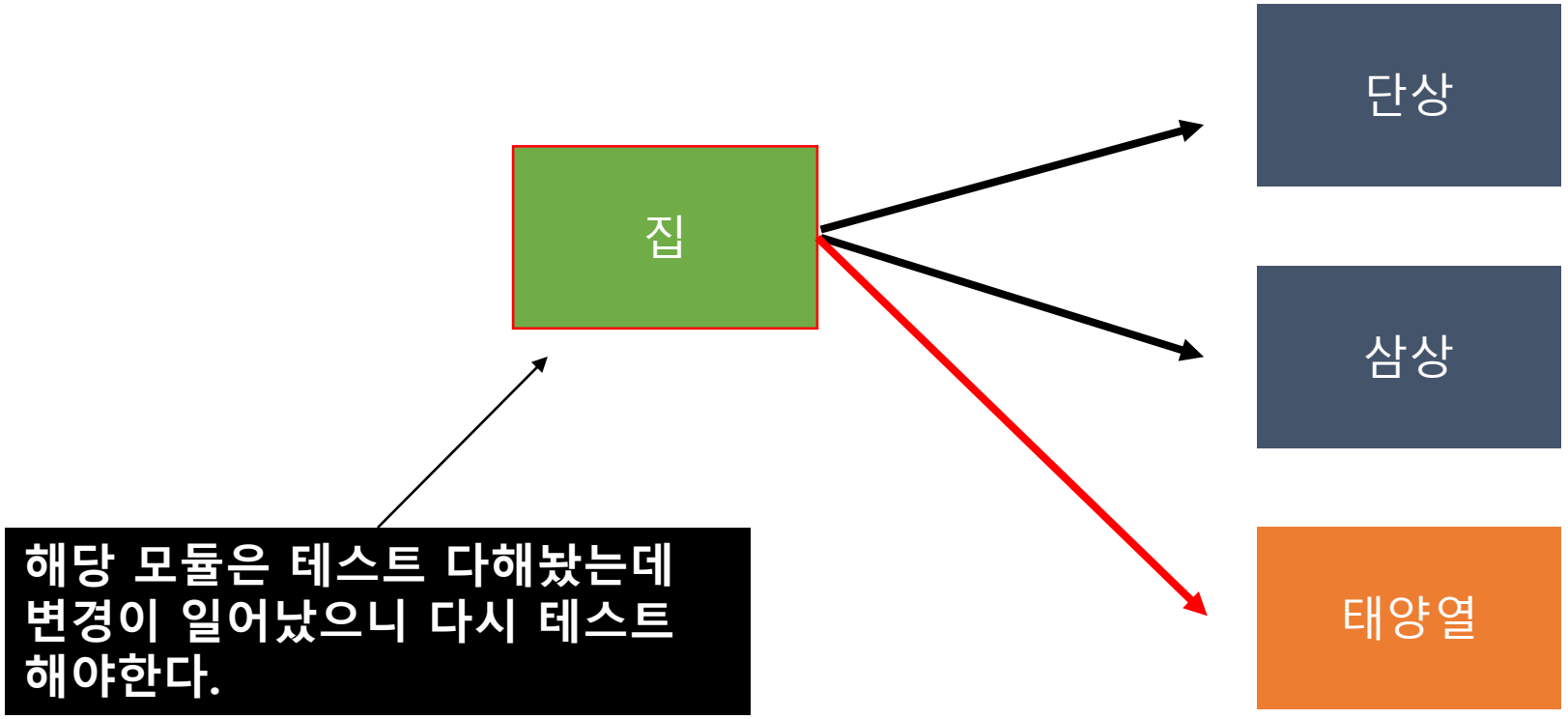


추가 확장

한 클래스가 다른 여러 클래스 의존하는 경우

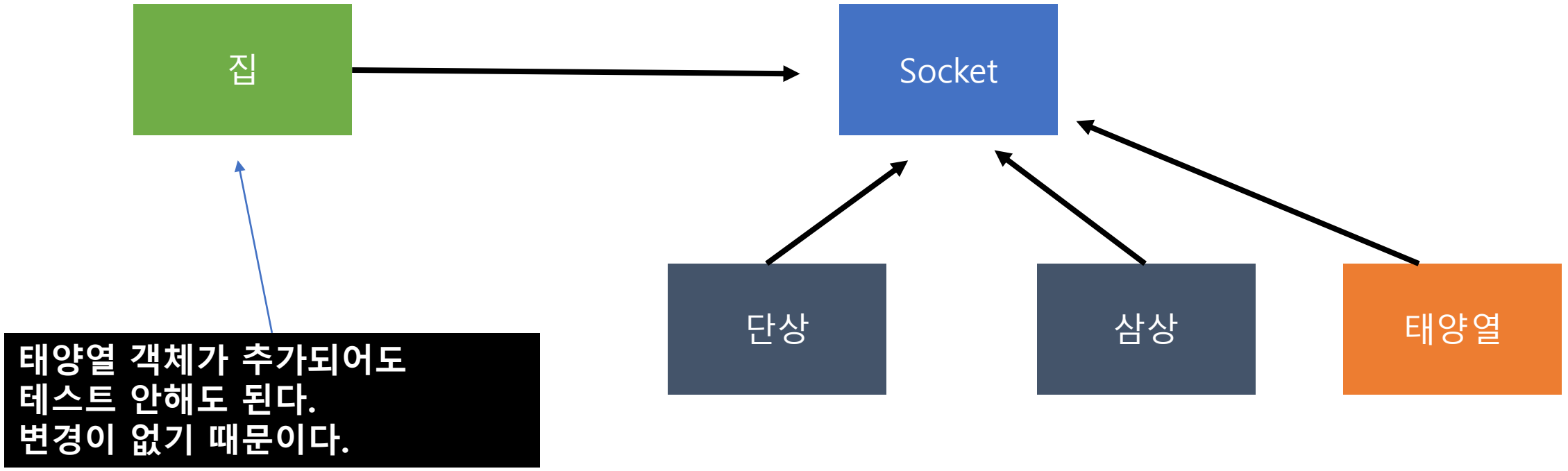
기능이 하나 더 추가되는 경우, 집 Class에 변경이 일어난다.

태양열 클래스 하나 더 추가시, '집' 객체는 변경이 일어남



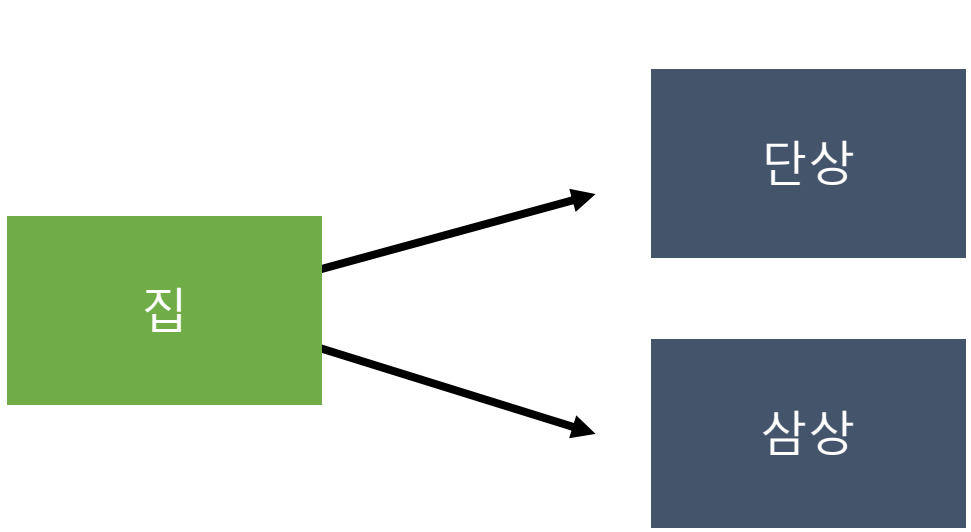
Interface를 추가한다.

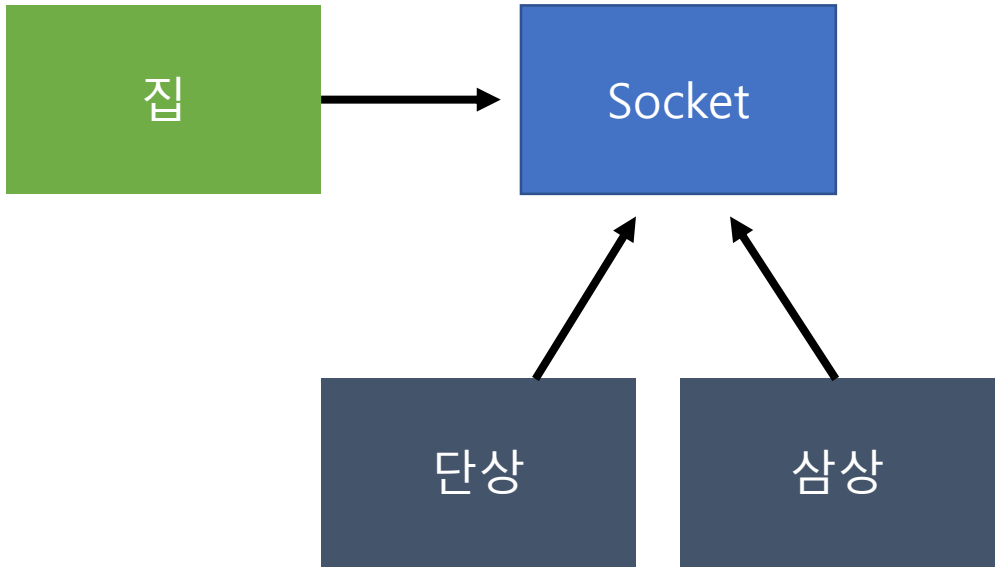
- Interface를 추가하면, 집 Class는 변경이 없다.
  태양열이 하나 더 추가되더라도, 집 Class는 변경이 없다.



[도전] 직접 구현해보기

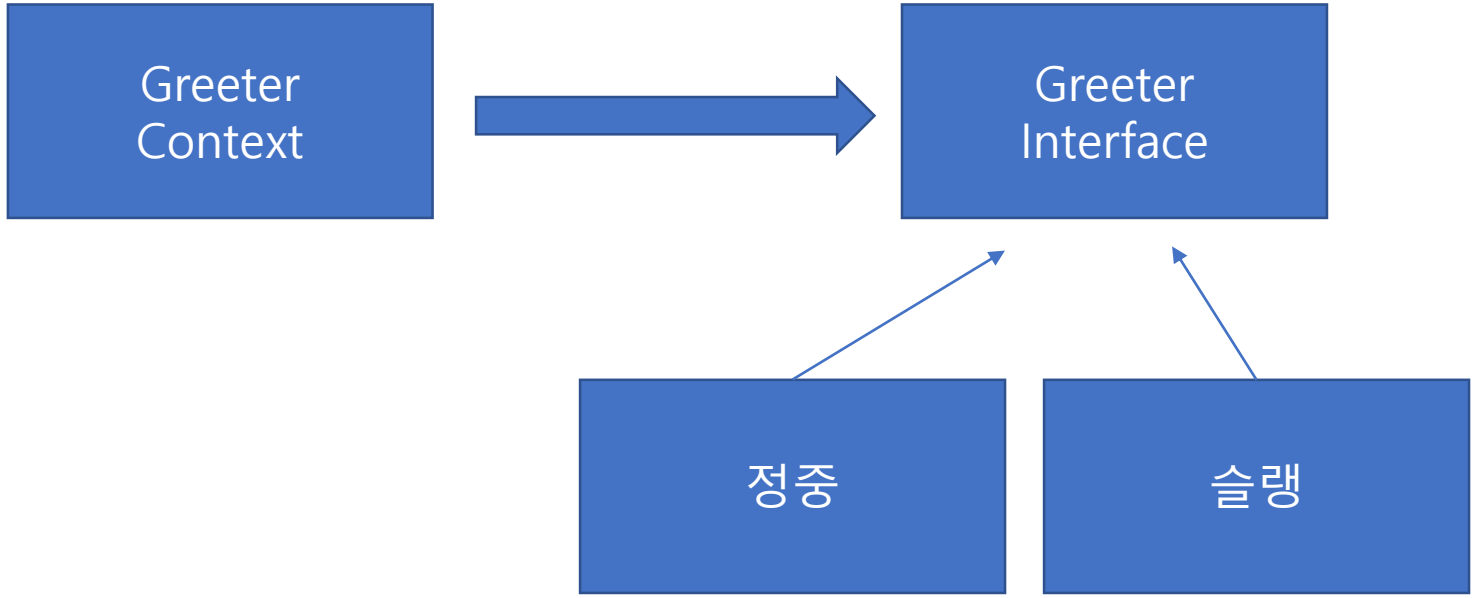
다음과 같이 Server Code를 구현하고,
이를 Test할 수 있는 Client Code도 구현해본다.





[도전] step1 : Greeter

새로운 인사 방식이 확장된다고 가정하고, 이를 OCP 설계에 맞도록 개선한다.



[도전] step2 : EventHandler

직접 해결해보자.

자동차는 Sport 모드 / Comport 모드 등이 존재

- 서스펜션 높이와 Power가 모드마다 달라짐

EventHandler 모듈의 변경을 최소화해보자.
그리고 Economy Driving Mode를 추가한다.