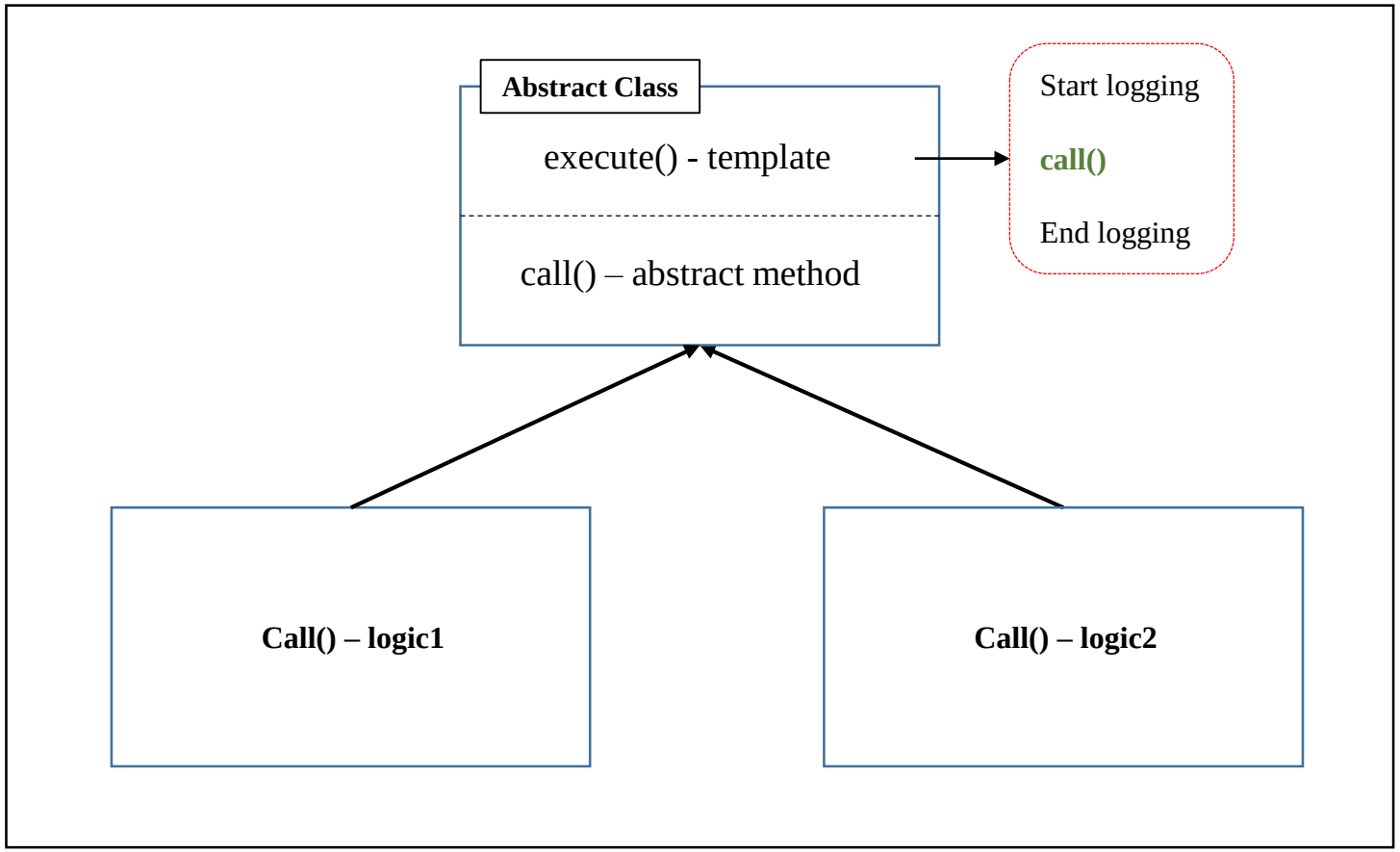
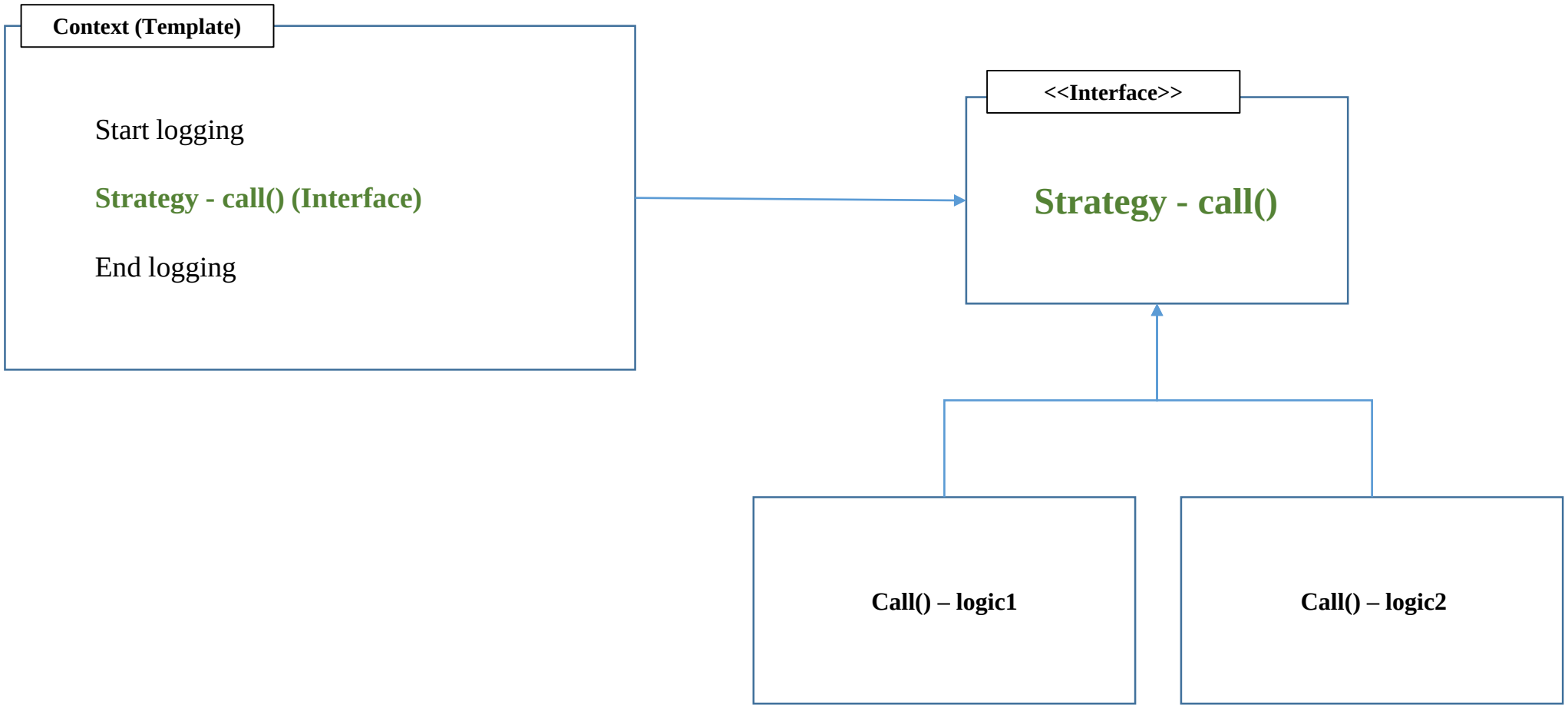
Template Method pattern

정리: 이태혁
(강의 김영한 스프링 advanced)

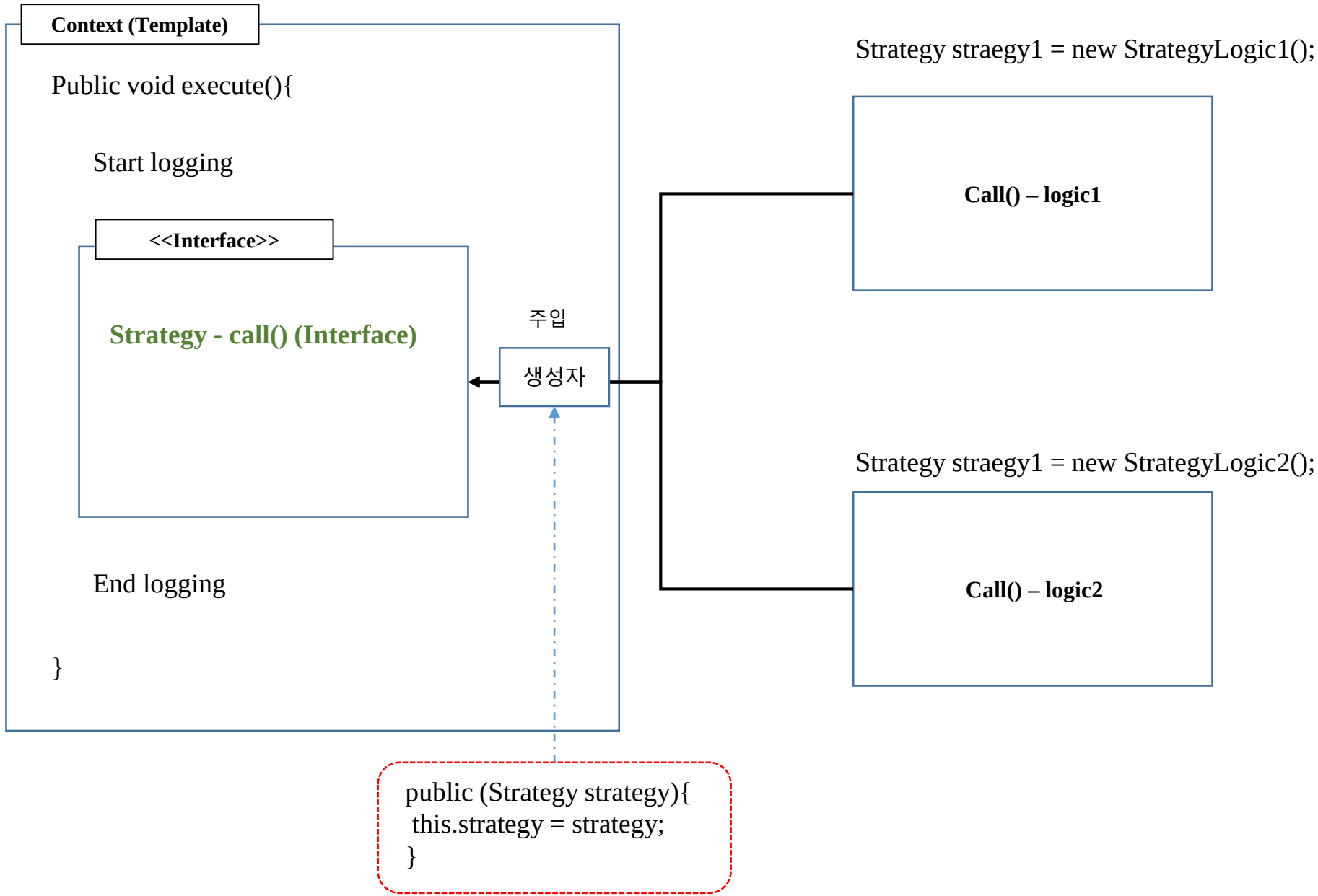


Abstract Class로 그냥 완성체를 하나 만들어서 사용 – 단점, class 상속

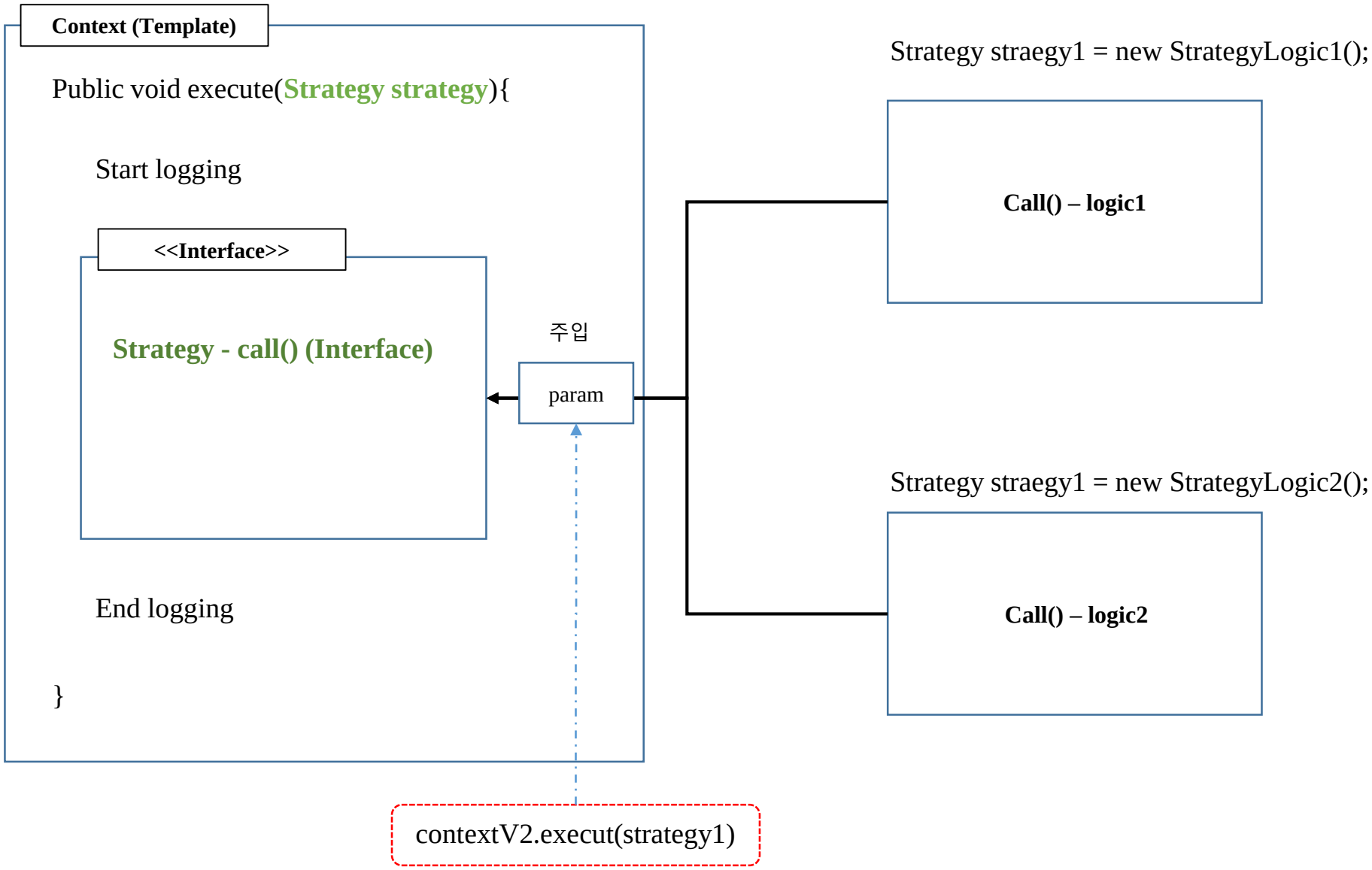
Strategy Pattern



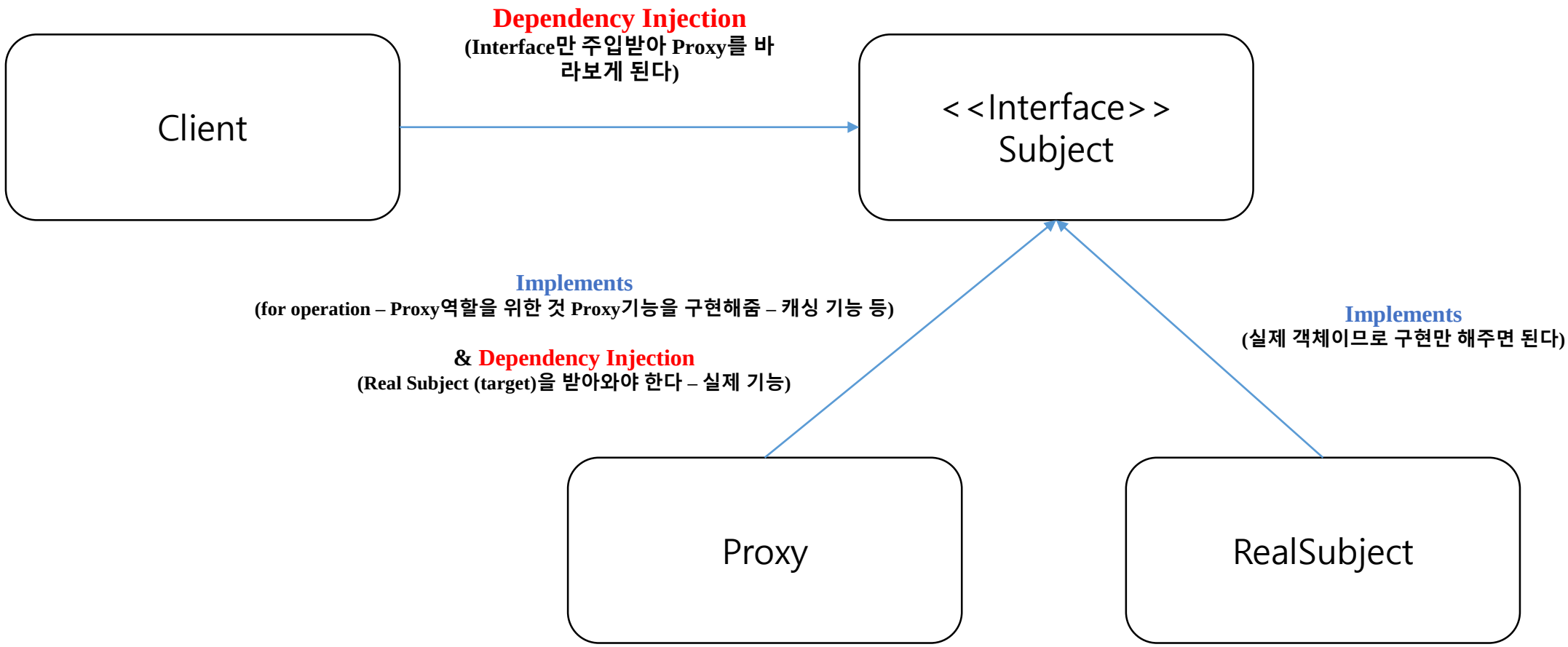
Strategy Pattern (ContextV1)



Strategy Pattern (ContextV2) – 템플릿 콜백 패턴

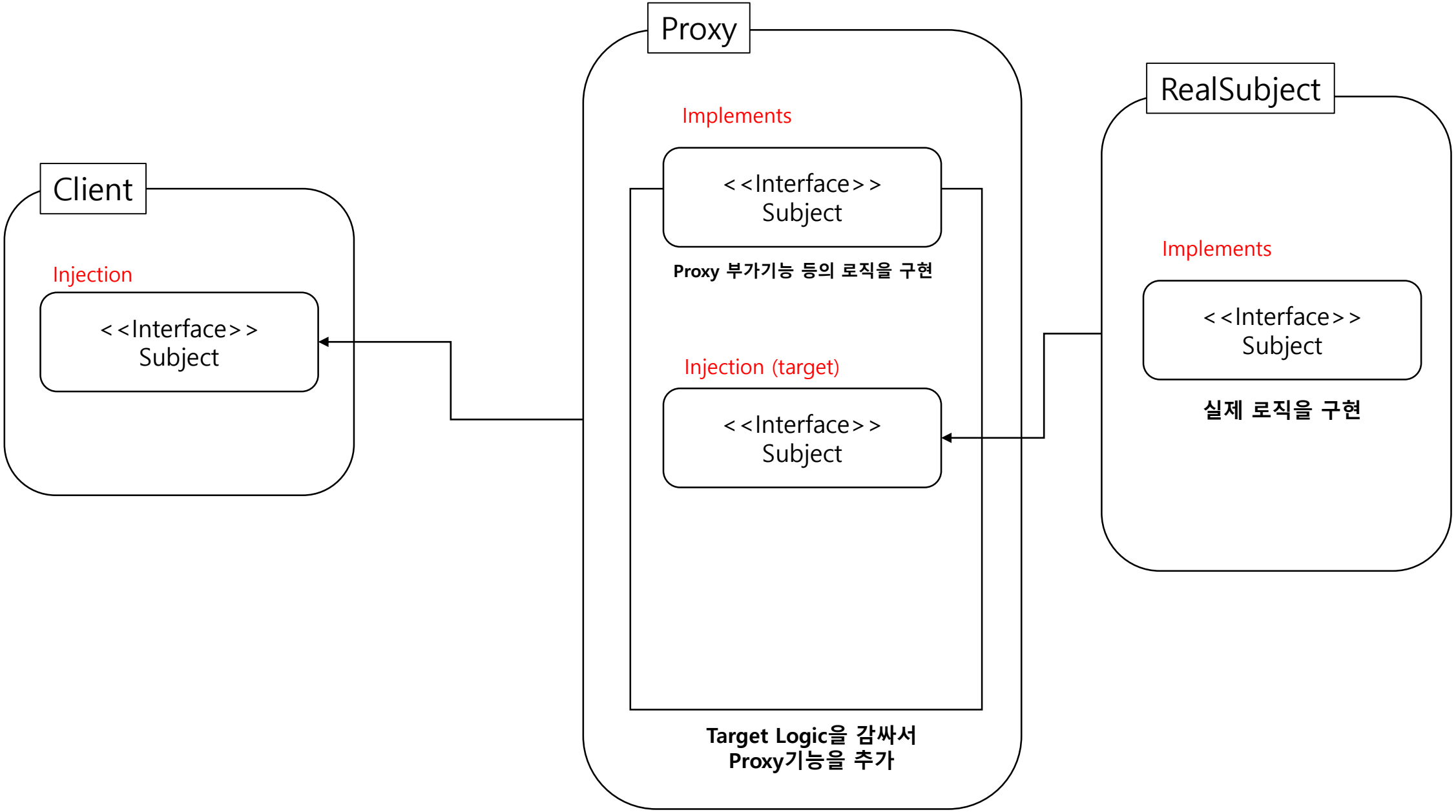


Proxy Pattern (Class 의존관계)

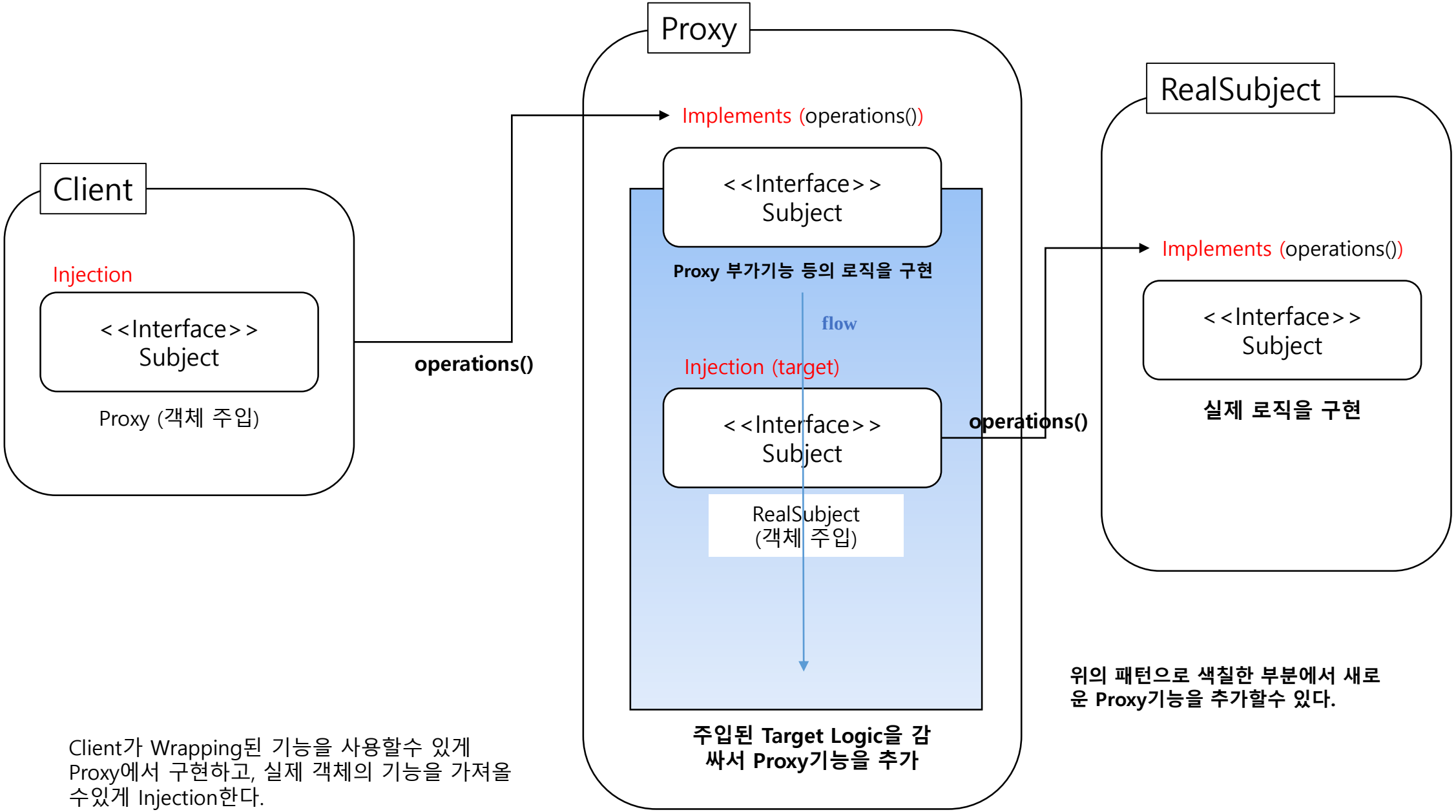


Client, Proxy, RealSubject 모두 한 인터페이스를 바라본다.

Proxy Pattern (실제 Injection 관계)



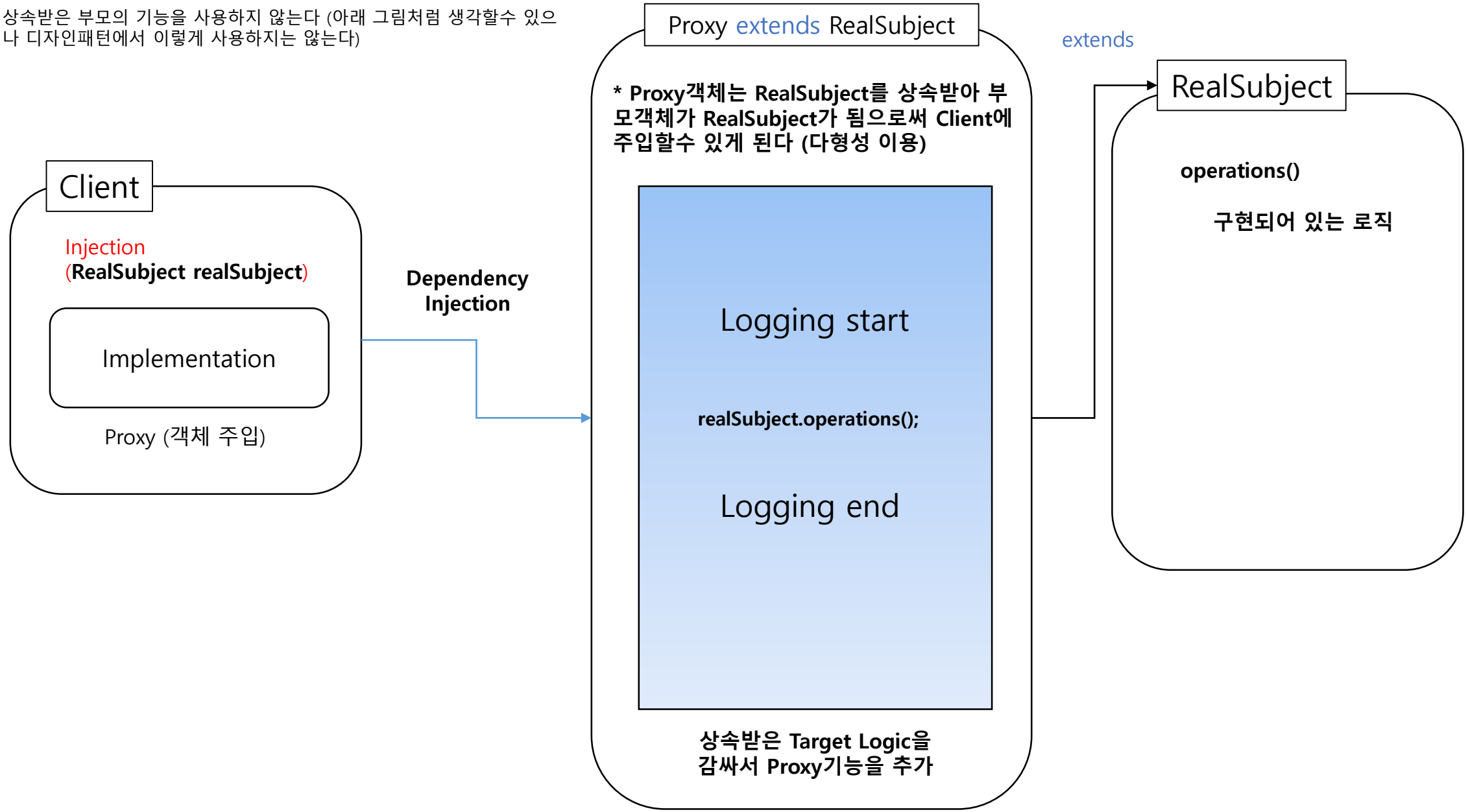
Proxy Pattern (실제 Runtime 동적관계) - 인터페이스가 있는 곳



Client가 Wrapping된 기능을 사용할수 있게 Proxy에서 구현하고, 실제 객체의 기능을 가져올 수있게 Injection한다.

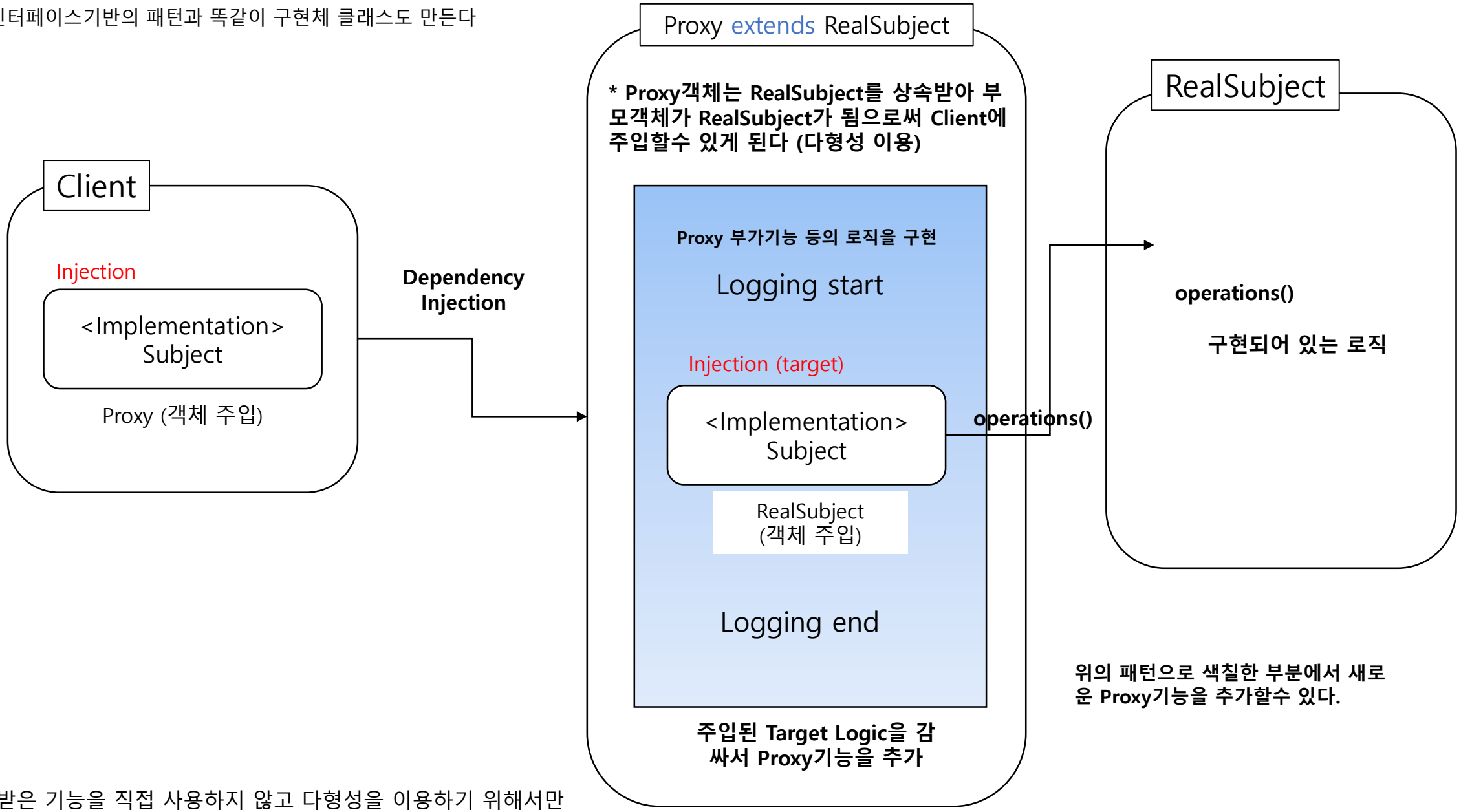
Proxy Pattern (실제 Runtime 동적관계) – 구체 클래스만 있는 곳

상속받은 부모의 기능을 사용하지 않는다 (아래 그림처럼 생각할수 있으나 디자인패턴에서 이렇게 사용하지는 않는다)



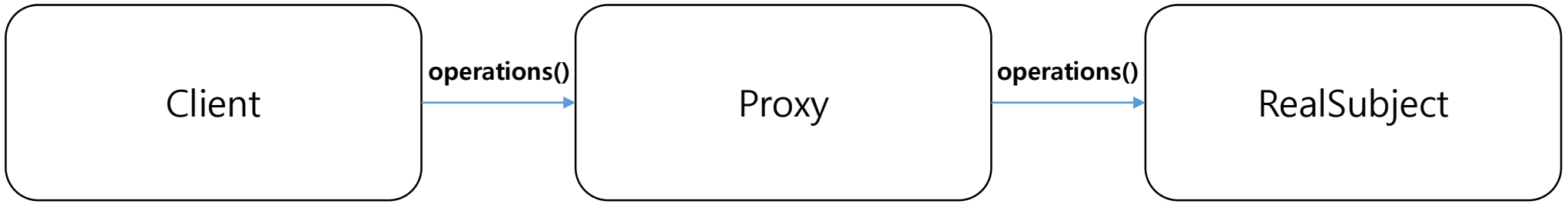
Proxy Pattern (실제 Runtime 동적관계) - 구체 클래스만 있는 곳

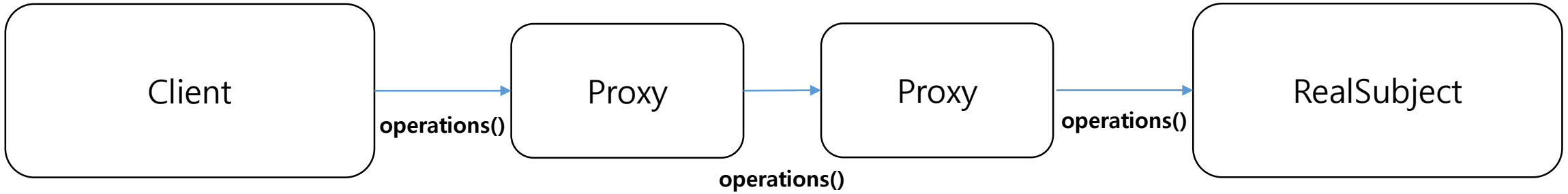
인터페이스기반의 패턴과 똑같이 구현체 클래스도 만든다



상속받은 기능을 직접 사용하지 않고 다형성을 이용하기 위해서만 상속받는다. 실제 target은 Injection받아서 사용한다 (패턴을 맞춘다)

Proxy Pattern (Runtime 의존관계) – 결론적으로 얘기하자면 아래와 같다.





Client, Target은 전혀 건들지않고 Proxy만 chain으로 계속 넣어서 꾸밀수 있다.