고급 객체지향 개발론

SOLID Principle

daumkakao
최윤상

OOP Design Principles

https://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/

SOLID 원칙

Single Responsibility Principle (SRP)

Open Closed Principle (OCP)

Liskov Substitution Principle (LSP)

Interface Segregation Principle (ISP)

Dependency Inversion Principle (DIP)

SINGLE RESPONSIBILITY PRINCIPLE

Just Because You Can, Doesn't Mean You Should

단일책임의 원칙(SRP)

“클래스는
단 한 개의 책임을
가져야 한다”

단일책임의 원칙(SRP)

"There should never be
more than one reason
for a class to change"

단일책임의 원칙(SRP)

단순하지만 다른 원리들의 **기초가 되는 원리**

변경시에도 **책임을 최상의 상태로 분배**

Shotgun surgery(산탄총 수술)

```
public class CodeViewer {
   // 표시할 소스코드를 set
   public void setCode(String code) {
     ……
   }
   // 소스코드를 화면에 표시
   public void display() {
     ……
   }
}
```

변경요구:
코드를 이쁘게 포매팅해서
표시해주셈.

```
public class CodeViewer {
   // 표시할 소스코드를 set
   public void setCode(String code) {
     ……
   }
   // 소스코드를 화면에 표시
   public void display() {
     ……
   }
}
```

```
public class CodeViewer {

  public void setCode(String Code) {
   ……
  }

  public void display() {
   String beautyCode = beautify(this.code);
   ……
  }
  private String beautify(String code){
    ……
    return beautyCode;
  }
}
```

```
public class CodeViewer {

  public void setCode(String Code) {
   ……
  }

  public void display() {
   String beautyCode = beautify(this.code);
   ……
  }
  private String beautify(String code){
    ……
    return beautyCode;
  }
}
```

변경요구:
주석은 빼고 보여주셈

```
public class CodeViewer {

  public void setCode(String Code)…

  public void display() {
   String filtered = filterComments(this.code);
   String beautyCode = beautify(filtered);
   ……
  }
  private String beautify(String code){
    ……
    return beautyCode;
  }
  private String filterComments(String code)…
}
```

```
public class CodeViewer {

   public

   public
    Strin
    Strin
     ……
   }
   }
   private String beautify(String code){
      ……
      return beautyCode;
   }
   private String filterComments(String code)…
}
```

추가 기능 요구에 따라 변경이 생기고 있음

이런 시점 즈음엔 '단일책임원칙'에 위배되었는지 모호한 시점.

그러나…

"포매팅할 때 indentation size 지정할 수 있게 해주세요"

"상황에 따라 javadoc은 남겨둬야 해요"

"code hightlight가 필요한데…"

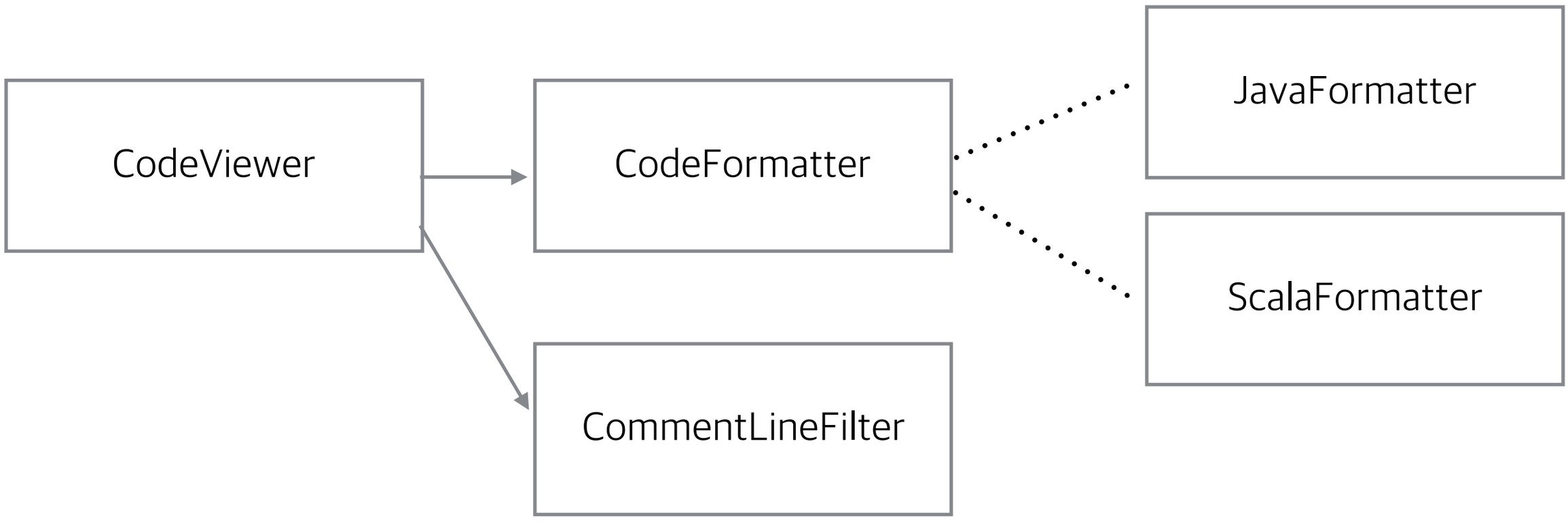
등의 요구사항이 생기면…

여러 원인에 의한 변경이 발생한다면

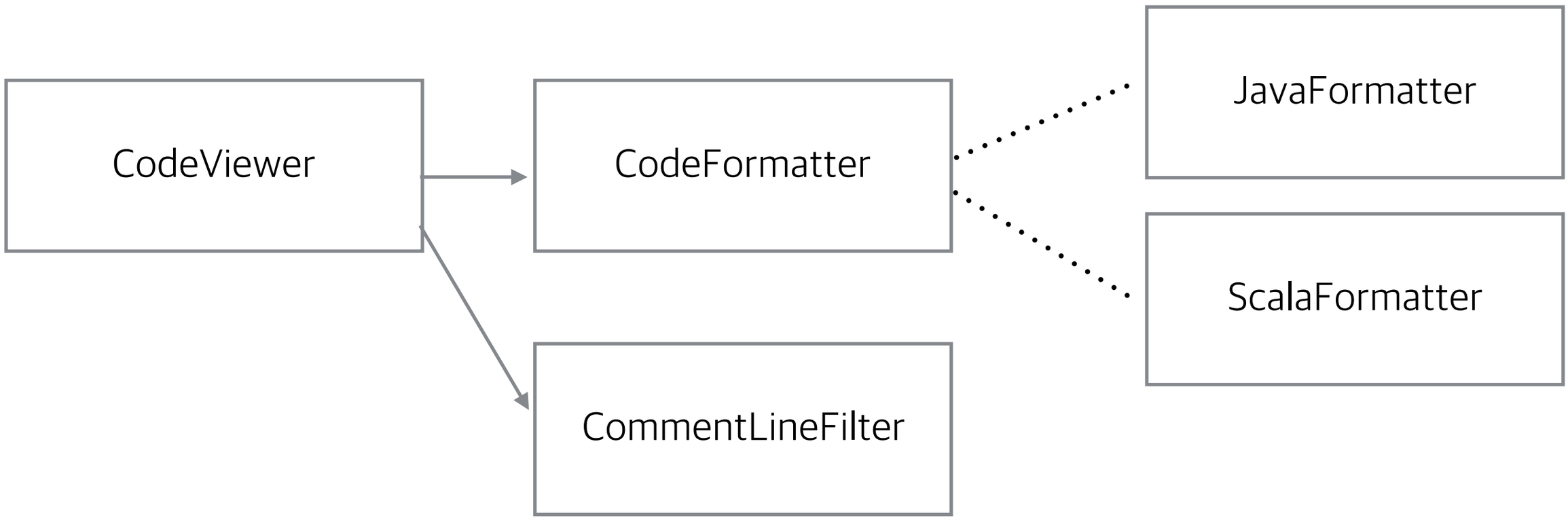
“단일책임원칙”에 위배되고 있는 것

=〉 책임의 재분배

Extract Class : 책임을 개별 클래스로 분할



Extract Class : 책임을 개별 클래스로 분할



SRP의 적용

클래스의 **이름**에 그 **책임**이 드러날 수 있게…

클래스가 **하나의 개념**을 나타낼수 있게…

책임을 분리하는 것만 SRP가 아님

때로는 책임을 병합해야 할때도..

OPEN CLOSED PRINCIPLE

Open Chest Surgery Is Not Needed When Putting On A Coat

개방 폐쇄의 원칙(OCP)

"**확장**에는 **열려** 있어야 하고
변경에는 **닫혀** 있어야 한다."

개방 폐쇄의 원칙(OCP)

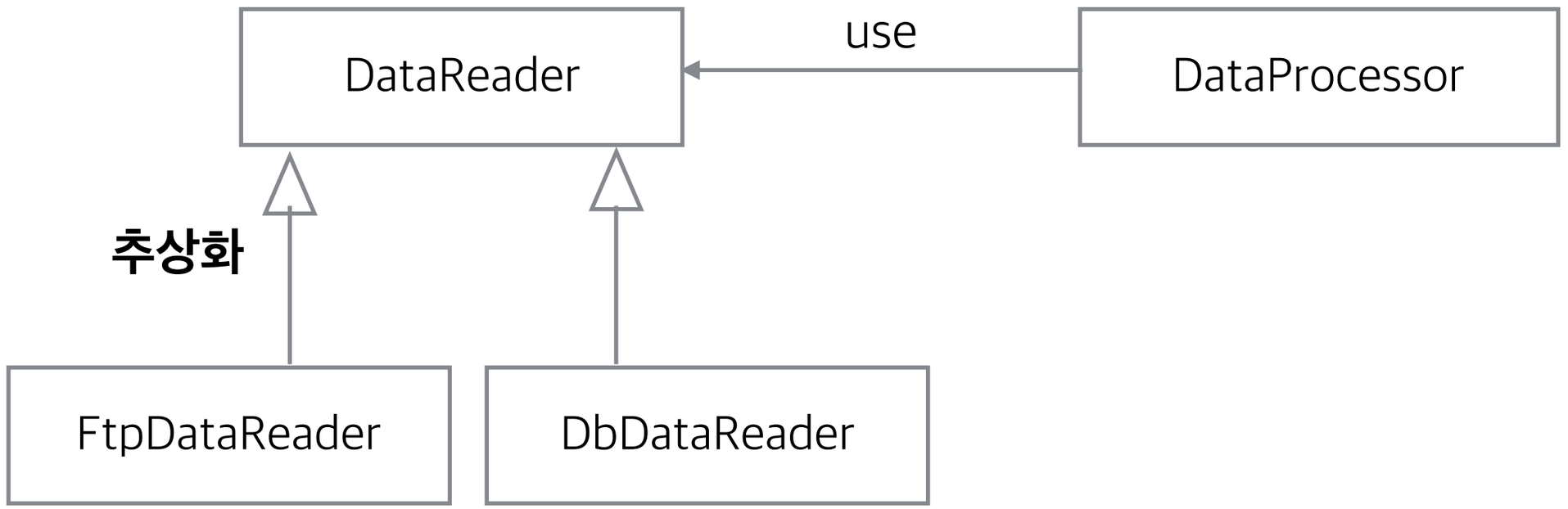
"You should
be able to extend a classes **behavior**,
without modifying it "

개방 폐쇄의 원칙(OCP)

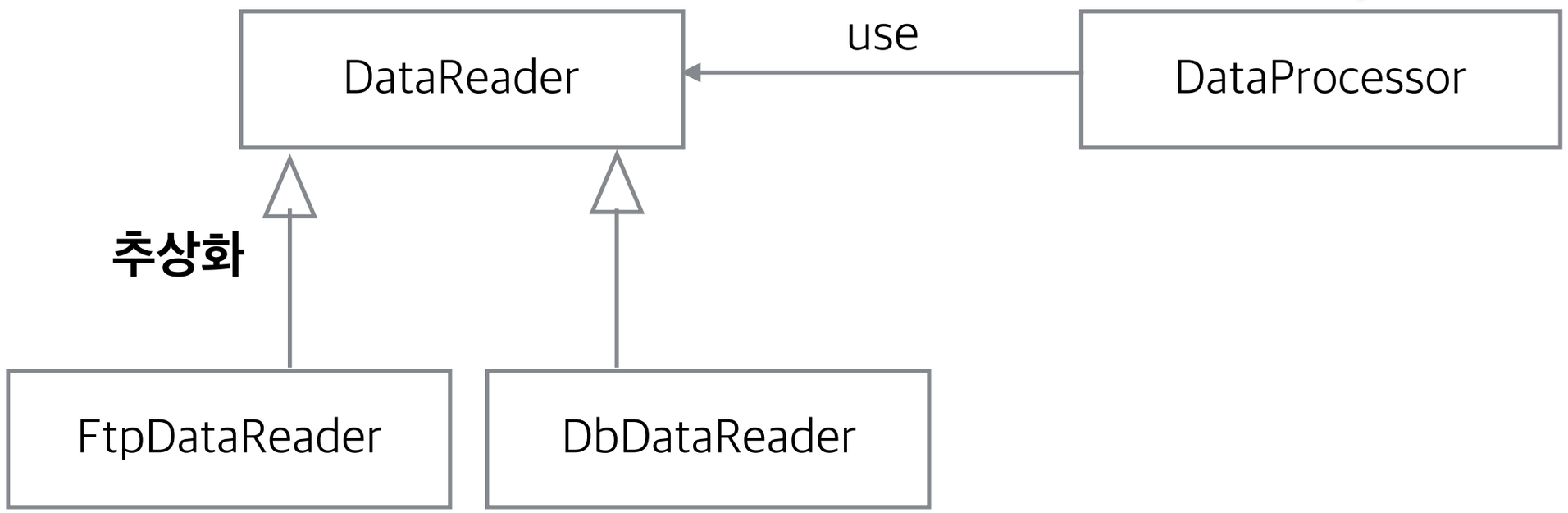
객체지향의 장점을 극대화하는 중요한 원리

변경 비용은 줄이고 **확장은 극대화**할 수 있게…

추상화와 **다형성**의 매커니즘을 활용


DataReader
use
DataProcessor
추상화
FtpDataReader
DbDataReader

데이터를 processing하는 방식을 변경하지 않고
다양한 data source를 처리하도록 확장할 수 있음
=>
reader의 확장에 대해서 열려있고
processor 수정에 대해서는 닫혀있음



How to OCP

1. **확장할 수 있는 부분**과
   그렇지 않은 부분을 **엄격히 구분**한다.
2. 그 두 부분이 만나는 지점에 **인터페이스를 정의**한다.
3. **인터페이스**에 의존하도록 **코드를 작성**

LISKOV SUBSTITUTION PRINCIPLE

If It Looks Like A Duck, Quacks Like A Duck, But Needs Batteries - You Probably Have The Wrong Abstraction

리스코프 치환 원칙(LSP)

"상위 타입의 객체를 하위 타입의 객체로 치환해도
상위 타입을 사용하는 프로그램은
정상적으로 동작해야 한다"

리스코프 치환 원칙(LSP)

"Functions
that use references to base classes
must **be able to**
use **objects of derived classes**
without knowing it."

리스코프 치환 원칙(LSP)

개방 폐쇄의 원칙을 받쳐주는 원칙

상속구조/다형성에 대한 원칙을 제공

계약(contract)에 의한 설계

LSP: 요구사항

하위형에서 메서드 인수의 **반공변성**

하위형에서 반환형의 **공변성**

하위형에서 발생하는 **예외의 공변성**

LSP: 공변성

공변성 / covariance

넓은 것에서 좁은 것으로의 변환

반공변성 / contravariance

좁은 것에서 넓은 것으로의 변환

LSP: 요구사항

하위형에서 선행조건은 강화될 수 없다

하위형에서 후행조건은 약화될 수 없다

하위형에서 상위형의 불변조건은 반드시 유지되어야 한다

직사각형–정사각형 문제

직사각형 setter의 사후조건(독립적인 너비/높이 변경)을
위반하는 케이스

getter만 갖는다면 LSP 위반은 발생하지 않음

LSP는
단순히 언어 구문상의 치환 문제가 아니라
“가변 객체의 치환성”의 개념

INTERFACE SEGREGATION PRINCIPLE

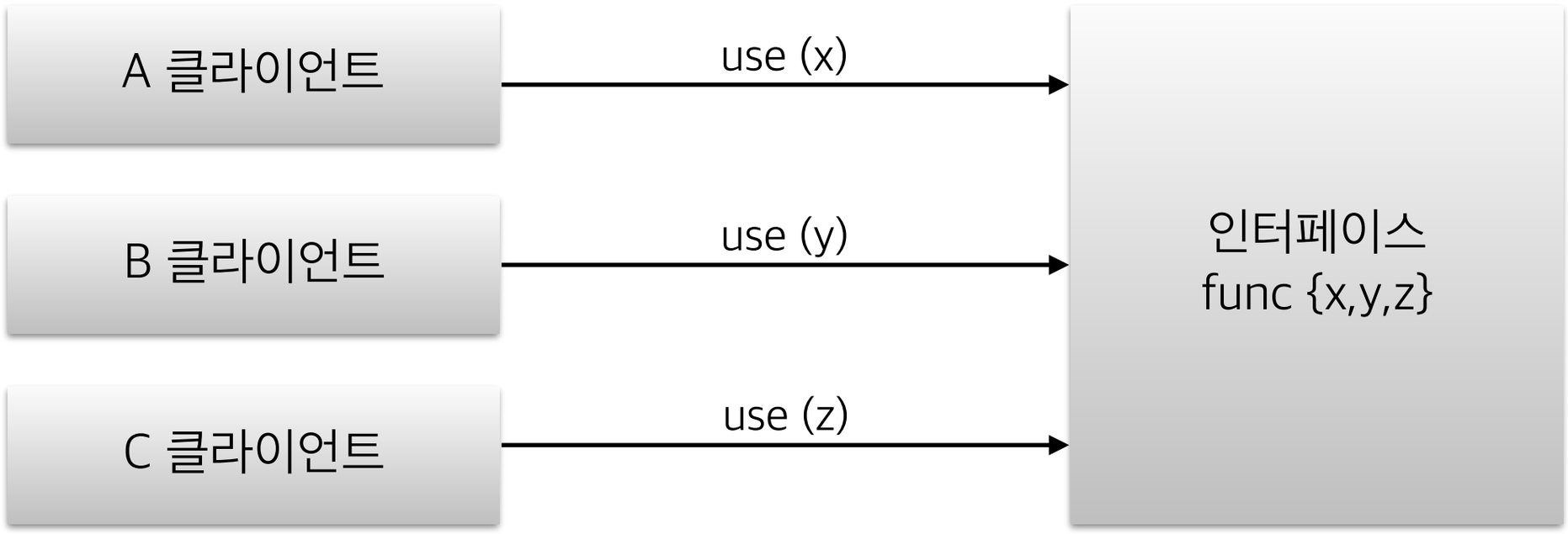
You Want Me To Plug This In, Where?

인터페이스 분리 원칙 (ISP)

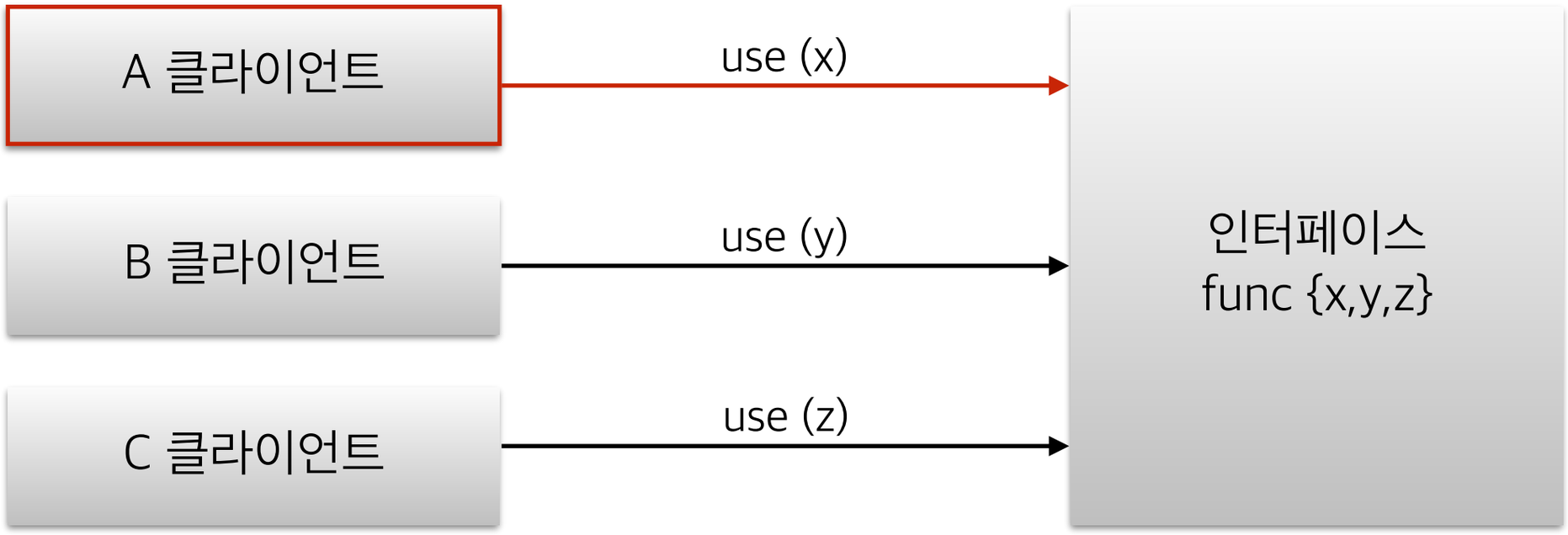
“인터페이스는 그 인터페이스를
사용하는 클라이언트를 기준으로
분리해야 한다”

인터페이스 분리 원칙 (ISP)

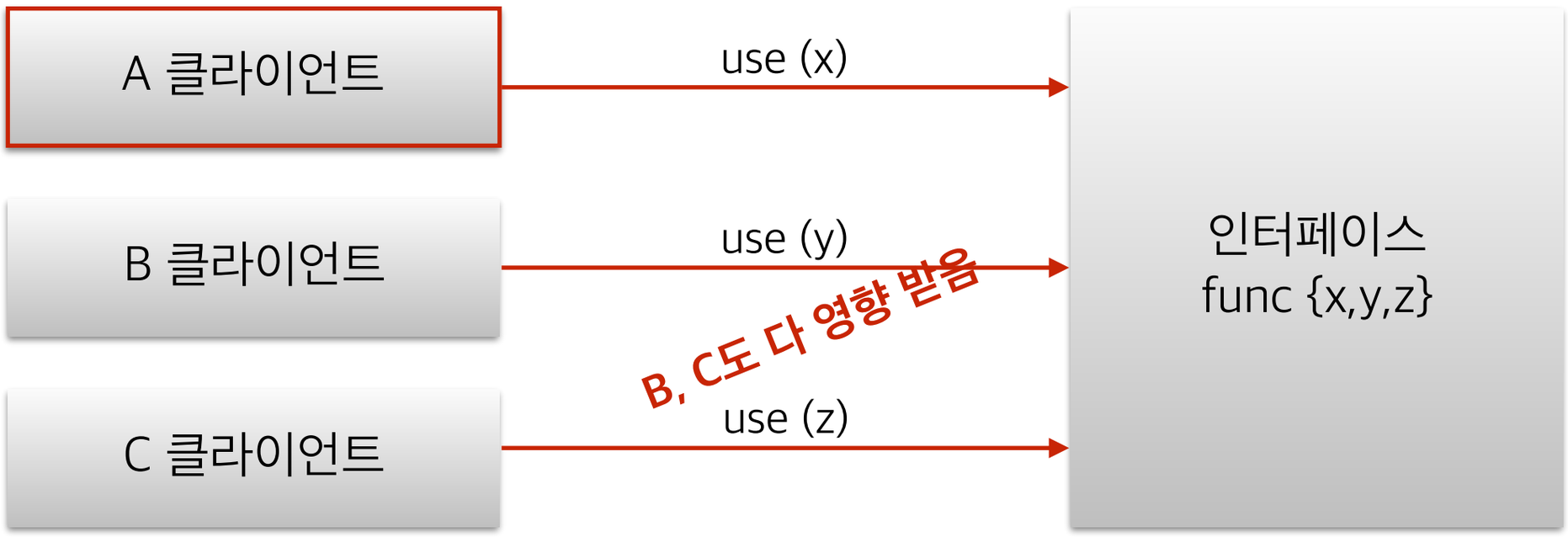
"Clients should not be forced to depend upon interfaces that they do not use"

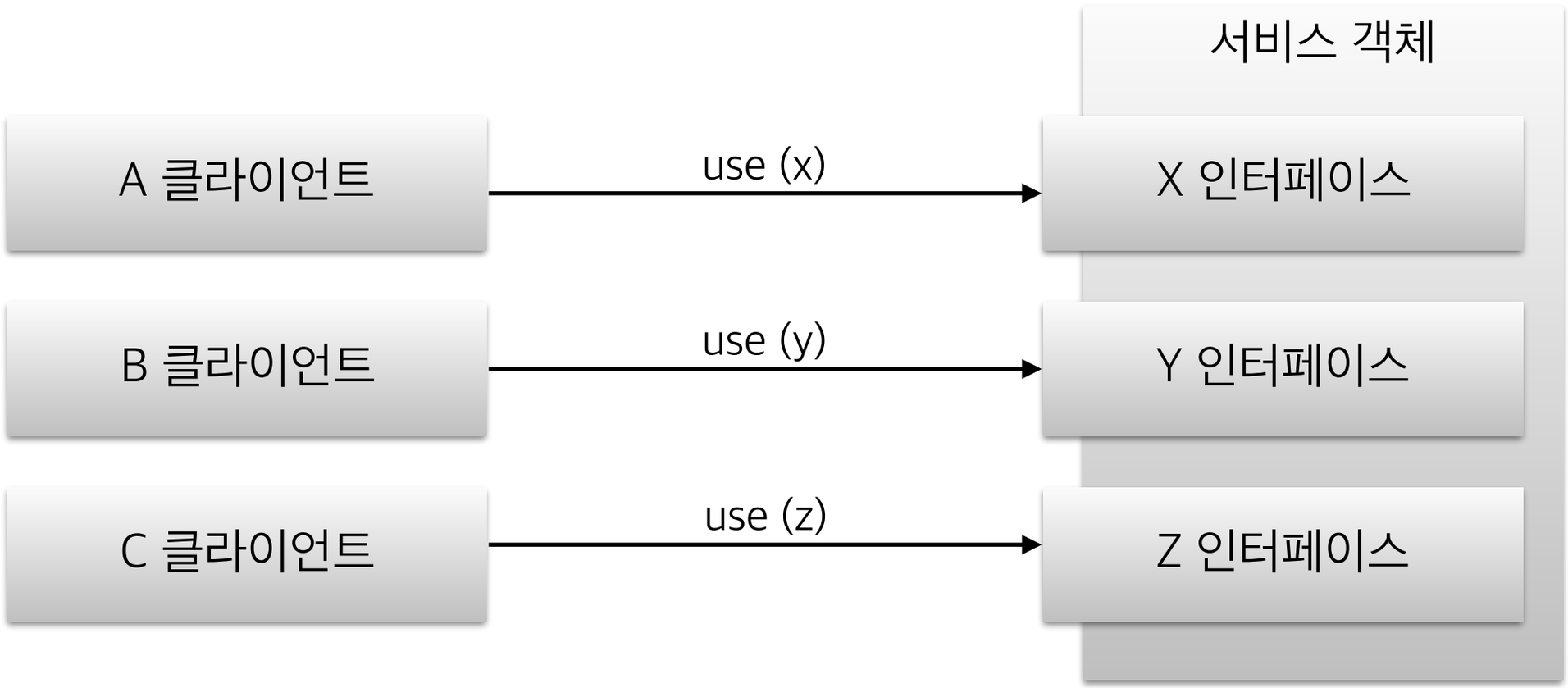

A 클라이언트
B 클라이언트
C 클라이언트
use (x)
use (y)
use (z)
인터페이스
func {x,y,z}

요구사항 변경으로 A가 변경됨
func x를 수정하기로 함



요구사항 변경으로 A가 변경됨
func x를 수정하기로 함




서비스 객체
A 클라이언트
use (x)
X 인터페이스
B 클라이언트
use (y)
Y 인터페이스
C 클라이언트
use (z)
Z 인터페이스

인터페이스 분리 원칙 (ISP)

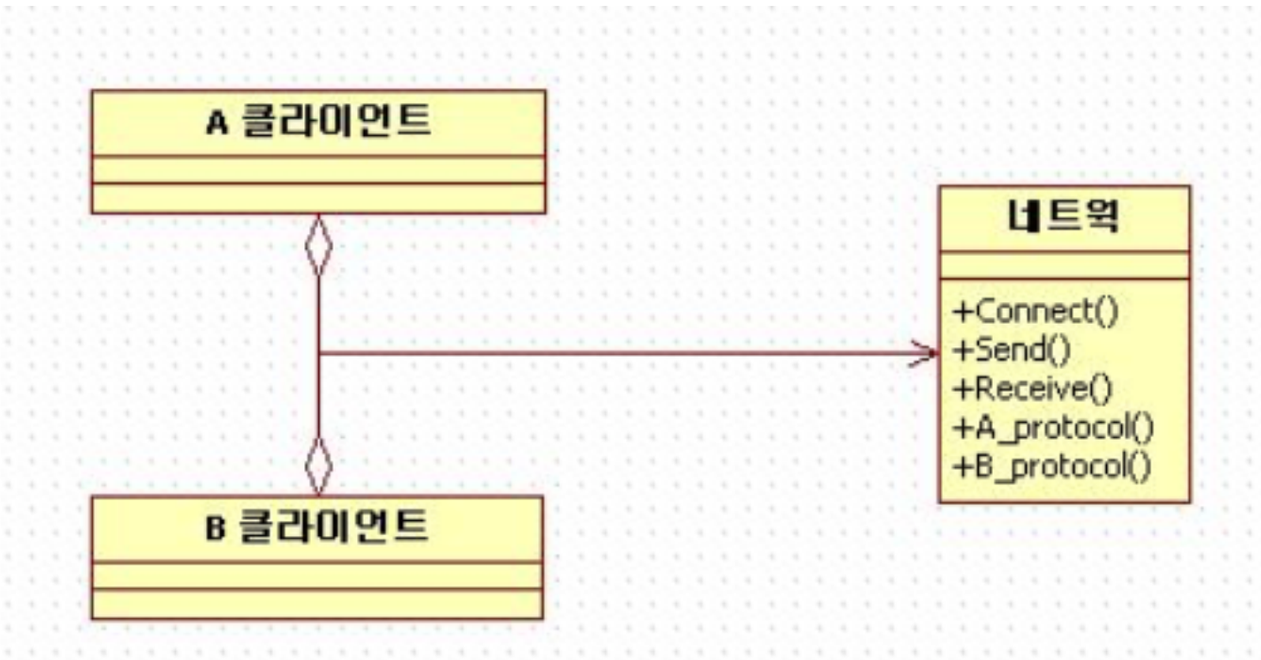
SRP(단일책임원칙)와 비슷

SRP는 클래스의 **단일책임**을 강조

ISP는 클라이언트 입장에서의
인터페이스의 **단일책임**을 강조

How to ISP

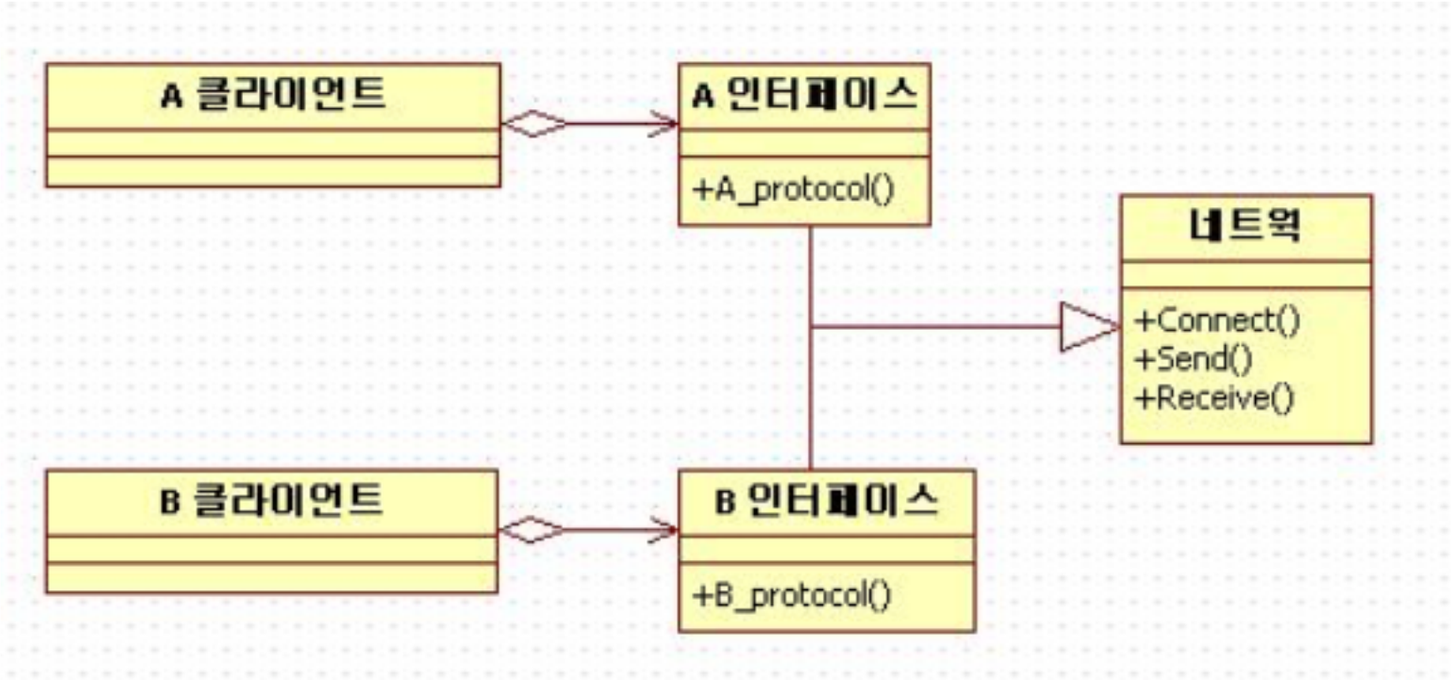
클래스 상속



http://extern.tistory.com/14

How to ISP

클래스 상속



http://extern.tistory.com/14

How to ISP

객체 인터페이스를 통한 분리 : delegate

```
public class SimpleTableDemo ... implements TableModelListener {
    ...
    public SimpleTableDemo() {
        ...
        table.getModel().addTableModelListener(this);
        ...
    }
    //인터페이스를 통해 노출할 기능을 구현합니다.
    public void tableChanged(TableModelEvent e) {
        int row = e.getFirstRow();
        int column = e.getColumn();
        TableModel model = (TableModel)e.getSource();
        String columnName = model.getColumnName(column);
        Object data = model.getValueAt(row, column);

        .../// Do something with the data...
    }
    ...
}
```

http://www.nextree.co.kr/p6960/

How to ISP

객체 인터페이스를 통한 분리 : delegate

e.g. Swing의 JTable

클라이언트에게 테이블에 대한 모든 기능을 제공하지만
Accessible, TableModleListener, ListSelectionListener
등의 인터페이스를 노출하여 서비스를 제공함

http://www.nextree.co.kr/p6960/

CQRS

Command Query Responsibility Segregation

"In an oversimplified manner, CQRS separates commands (that change the data) from the queries (that read the data)

CQRS는 (데이터를 변경하는) command부분을 (데이터를 읽는) query로부터 분리시킨다.

DEPENDENCY INVERSION PRINCIPLE

Would You Solder A Lamp Directly To The Electrical Wiring In A Wall?

의존 역전 원칙 (DIP)

“고수준 모듈은 저수준 모듈의 구현에
의존해서는 안된다.
저수준의 모듈이 고수준 모듈에서
정의한 추상 타입에 의존해야 한다”

Layering

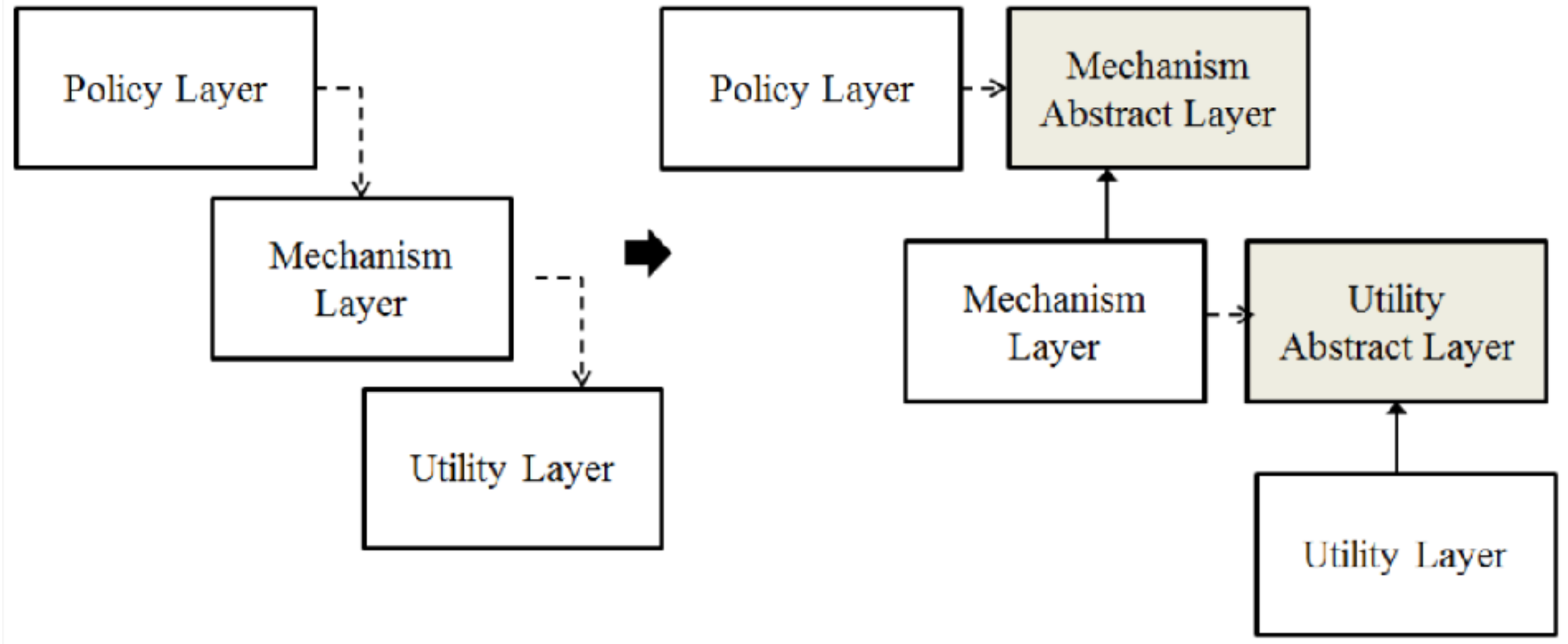

Policy Layer
Mechanism Layer
Utility Layer
Policy Layer
Mechanism Abstract Layer
Mechanism Layer
Utility Abstract Layer
Utility Layer

Figure 5: Naive Button/Lamp Model

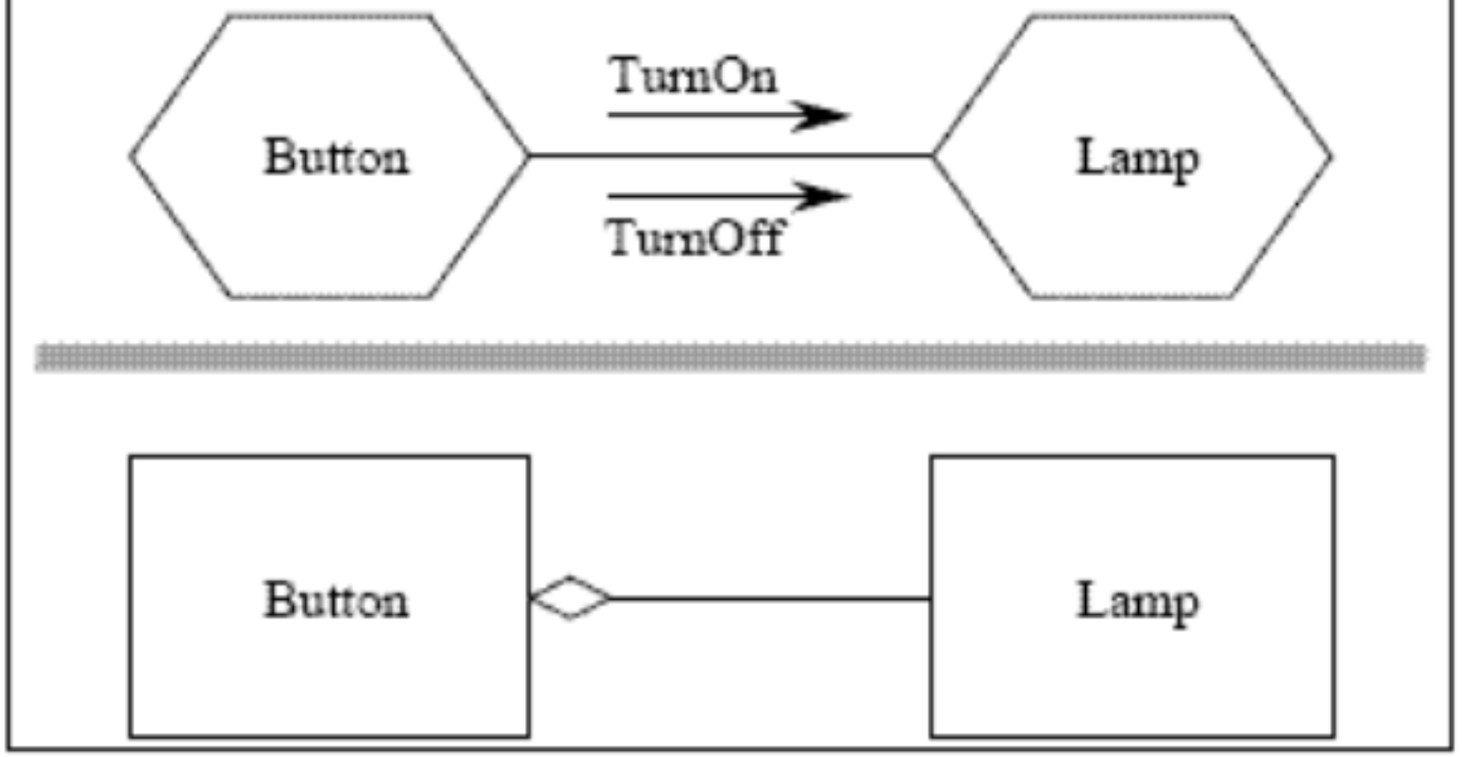


Figure 6: Inverted Button Model