KB금융그룹 | 국민의 평생 금융파트너

2025년 상반기 K-디지털 트레이닝

# Bridge - 기능 계층과 구현 계층을 나눈다

## [KB] IT's Your Life

KB 국민은행

# Bridge 패턴

## Bridge 패턴

- 기능의 클래스 계층과 구현의 클래스 계층을 연결

## 클래스 계층의 두가지 역할

- 기존 객체에 새로운 기능을 추가하고 싶을 때
  - 상속

```
Something
└ SomethingGood
```

```
Something
  └ SomethingGood
      └ SomethingBetter
```

- 상위 클래스는 기본 기능을 가짐
- 하위 클래스는 새로운 기능을 가짐

--> 기능의 클래스 계층

# Bridge 패턴

## 클래스 계층의 두가지 역할

### 새로운 구현을 추가하고 싶을 때

- Template Method 패턴

```
AbstractClass
└ ConcreteClass
```

```
AbstractClass
├ ConcreteClass
└ AnotherConcreteClass
```

- 상위 클래스는 추상 메서드로 인터페이스(API)를 규정
- 하위 클래스는 구상 메서드로 그 인터페이스(API)를 구현

→ 구현의 클래스 계층

# Bridge 패턴

## 클래스 계층의 혼재와 클래스 계층의 분리

- 하위 클래스를 만들고자 할 때 자신의 의도를 다음과 같이 확인
  - 기능을 추가하려고 하는가?
  - 기능을 구현하려고 하는가?

- 클래스 계층이 하나인 경우
  - 기능의 클래스 계층과 구현의 클래스 계층이 하나의 계층 구조 안에 혼재

→ 클래스 계층을 복잡하게 만들어 예측을 어렵게 함,
하위 클래스를 만들고자 할 때 클래스 계층 어디에 만들면 좋을지 고민하게 됨

- 기능의 클래스 계층과 구현의 클래스 계층을 두 개의 독립된 클래스 계층으로 분리

→ 두 계층을 연결하는 다리가 필요

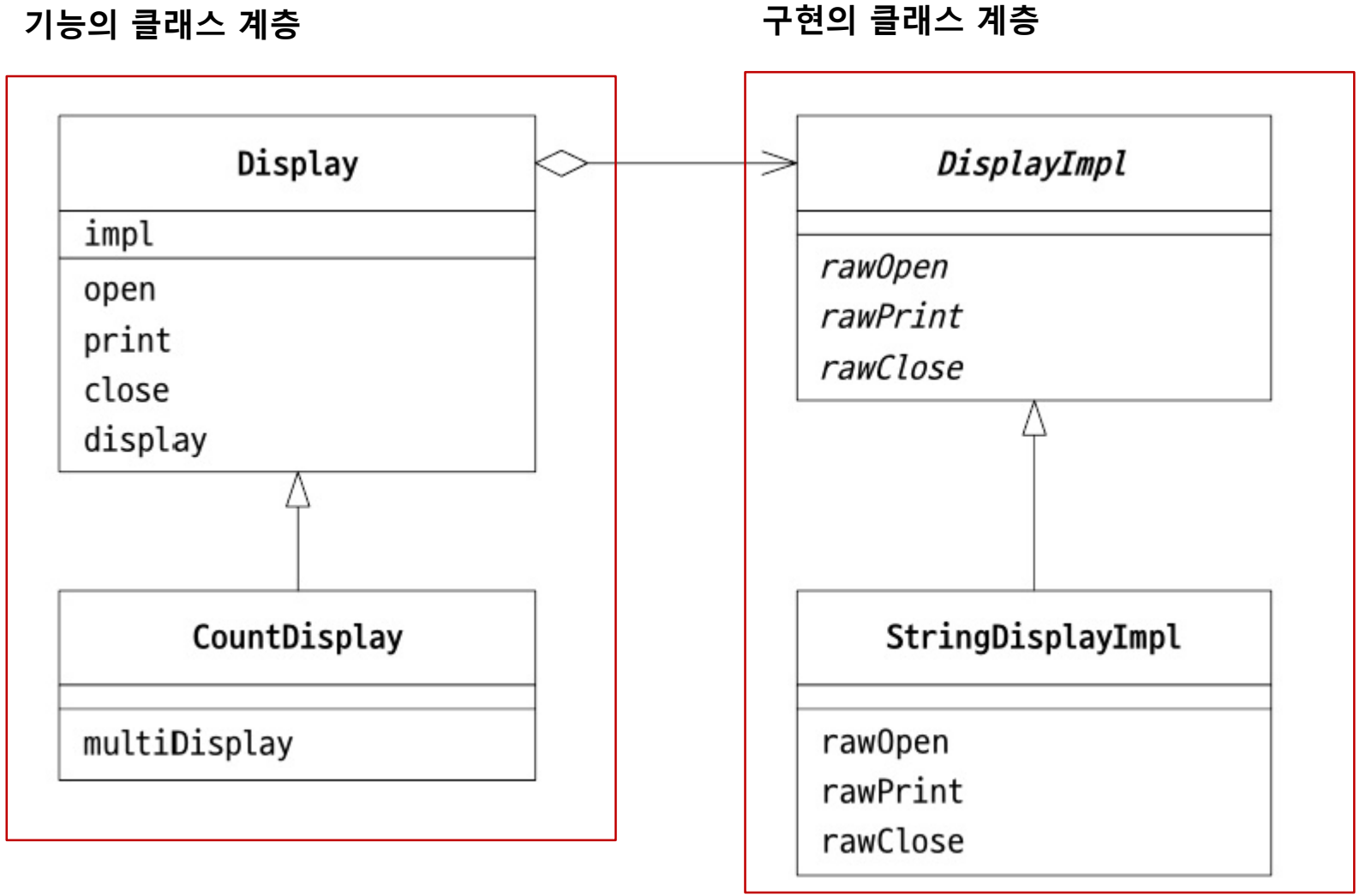
## 예제 프로그램

- 무엇인가를 표시하기 위한 프로그램

| 다리의 어느 쪽인가? | 이름 | 설명 |
|---|---|---|
| 기능의 클래스 계층 | Display | '표시한다' 클래스 |
| 기능의 클래스 계층 | CountDisplay | '지정 횟수만큼 표시한다' 기능을 추가한 클래스 |
| 구현의 클래스 계층 | DisplayImpl | '표시한다' 클래스 |
| 구현의 클래스 계층 | StringDisplayImpl | '문자열을 사용해서 표시한다' 클래스 |
| | Main | 동작 테스트용 클래스 |

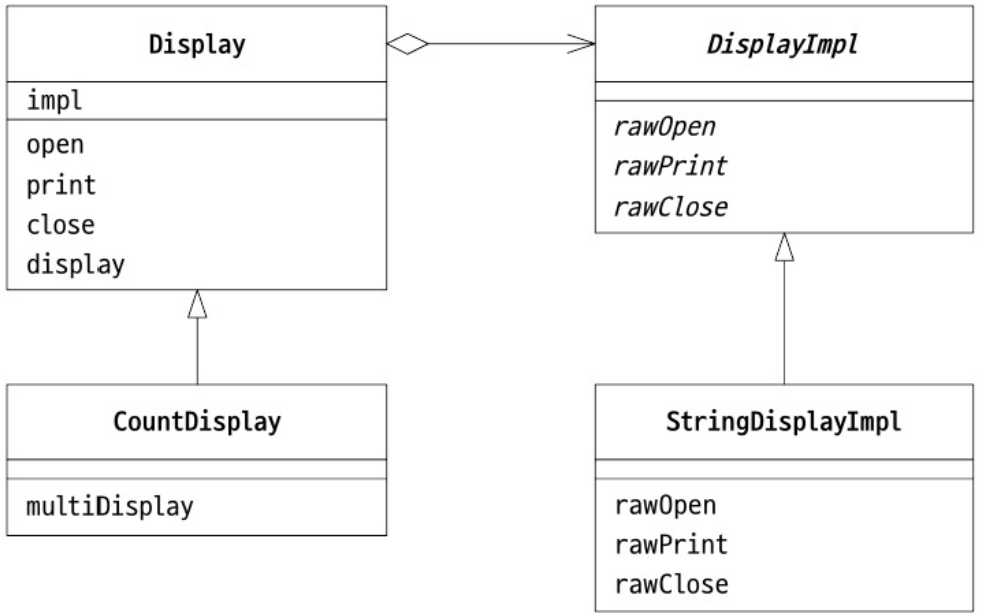
# Bridge 패턴

## 예제 프로그램 클래스 다이어그램



## 구현의 클래스 계층 Display 클래스

- 구현의 클래스 계층 최상위
- 추상 메서드
  - rawOpen
  - rawPrint
  - rawClose

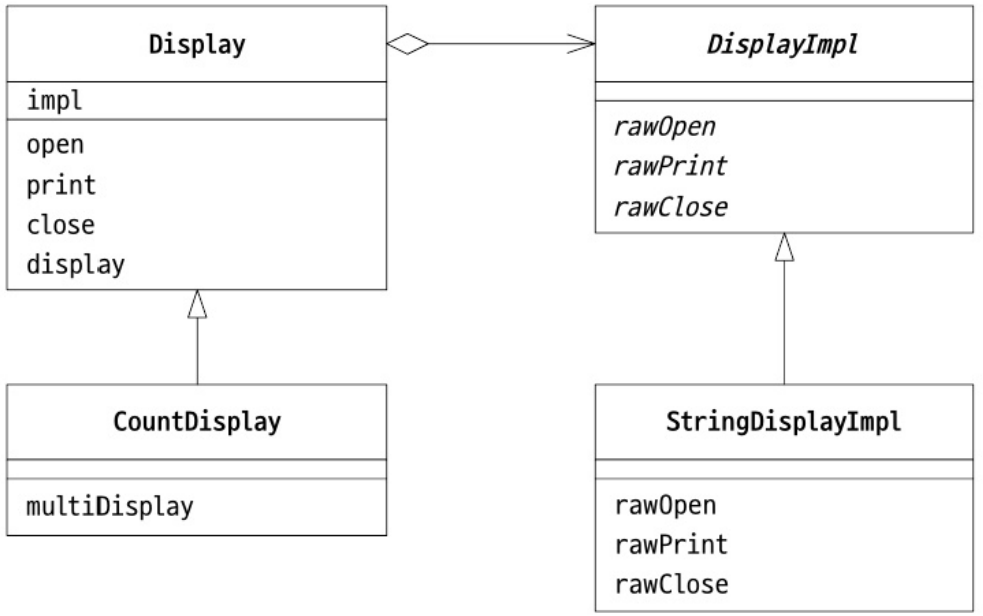


## DisplayImpl.java

```
public abstract class DisplayImpl {
    public abstract void rawOpen();
    public abstract void rawPrint();
    public abstract void rawClose();
}
```

# Bridge 패턴

- 구현의 클래스 계층: StringDisplayImpl
  - 진정한 구현 클래스
  - 실제 무엇인가를 표시



## StringDisplayImpl.java

```
public class StringDisplayImpl extends DisplayImpl {
    private String string;
    private int width;

    public StringDisplayImpl(String string) {
        this.string = string;
        this.width = string.length();
    }

    private void printLine() {
        System.out.print("+");
        for(int i = 0; i < width; i++) {
            System.out.print("-");
        }
        System.out.println("+");
    }
```

## StringDisplayImpl.java

```
    @Override
    public void rawOpen() {
        printLine();
    }

    @Override
    public void rawPrint() {
        System.out.println("¦" + string + "¦");
    }

    @Override
    public void rawClose() {
        printLine();
    }

}
```

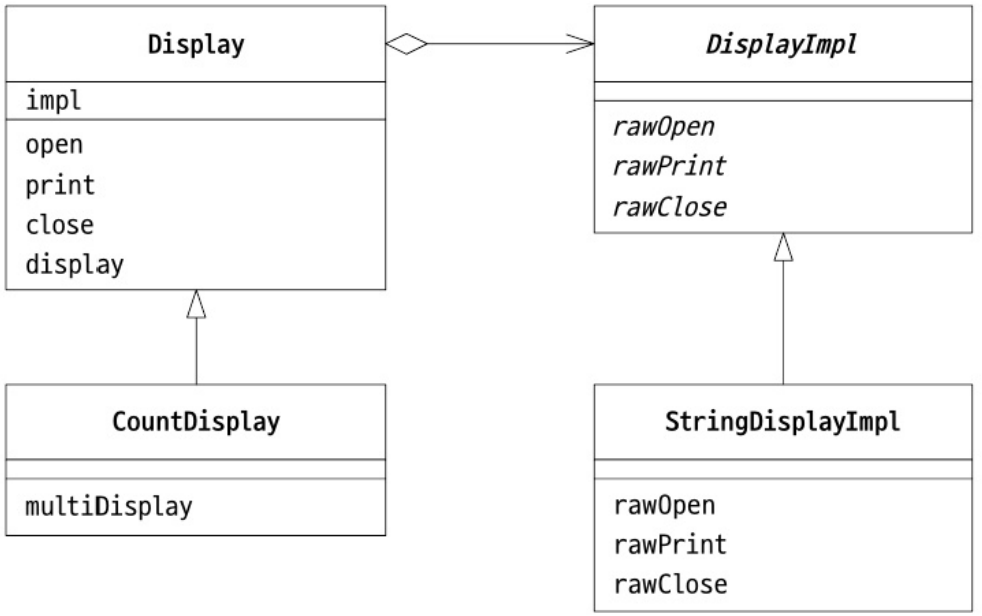
# Bridge 패턴

## 기능의 클래스 계층 Display 클래스

- 추상적인 무엇인가를 표시하는 것
- 기능의 클래스 계층에서 최상위 클래스

- impl 멤버
  - Display 클래스의 구현을 나타내는 인스턴스(위임)
- 메서드
  - open 표시의 전처리
  - print 표시 그 자체
  - close 표시의 후처리

→ 실제 처리는 impl을 통해서 이루어짐(위임)

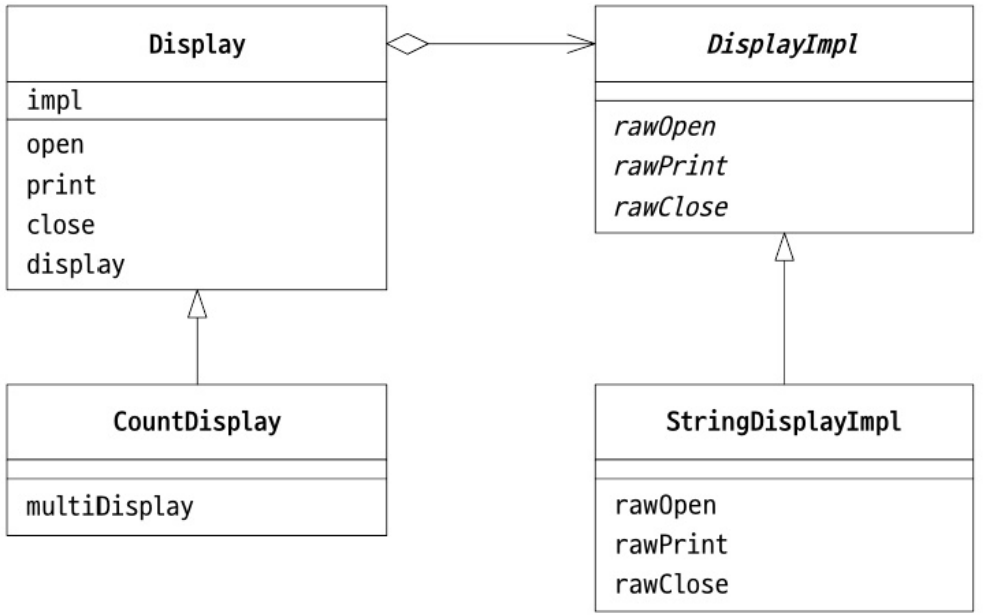


## Display.java – 기능의 클래스 계층

```
public class Display {
    private DisplayImpl impl;    // bridge 역할(위임)

    public Display(DisplayImpl impl) {
        this.impl = impl;
    }

    public void open() {
        impl.rawOpen();
    }

    public void print() {
        impl.rawPrint();
    }

    public void close() {
        impl.rawClose();
    }

    public final void display() {
        open();
        print();
        close();
    }
}
```

# Bridge 패턴

- 기능의 클래스 계층 CountDisplay 클래스
  - multiDisplay() 메서드로 새로운 기능 추가



## CountDisplay.java

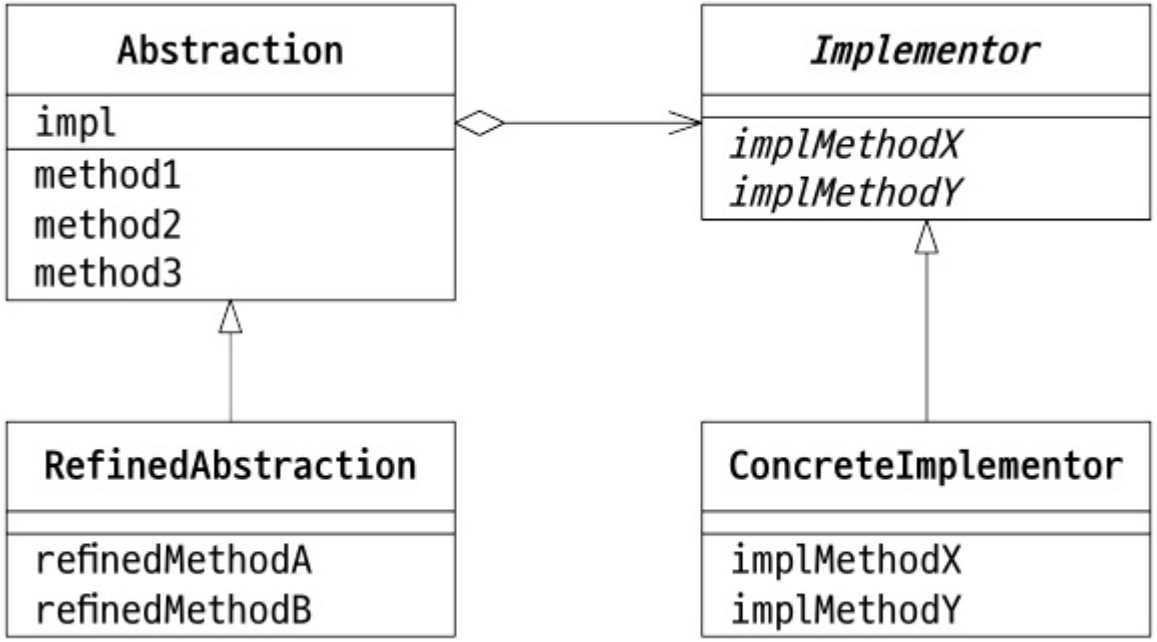
```
public class CountDisplay extends Display {
    public CountDisplay(DisplayImpl impl) {
        super(impl);
    }

    public void multiDisplay(int times) {
        open();
        for (int i = 0; i < times; i++) {
            print();
        }
        close();
    }
}
```

## Main.java

```
public class Main {
    public static void main(String[] args) {
        Display d1 = new Display(new StringDisplayImpl("Hello, Korea."));
        Display d2 = new CountDisplay(new StringDisplayImpl("Hello, World."));
        CountDisplay d3 = new CountDisplay(new StringDisplayImpl("Hello, Universe."));

        d1.display();
        d2.display();
        d3.display();

        d3.multiDisplay(5);

    }
}
```

```
+-------------+
|Hello, Korea.|
+-------------+
+-------------+
|Hello, World.|
+-------------+
+----------------+
|Hello, Universe.|
+----------------+
+----------------+
|Hello, Universe.|
|Hello, Universe.|
|Hello, Universe.|
|Hello, Universe.|
|Hello, Universe.|
+----------------+
```

## Bridge 다이어그램



# Bridge 패턴

## Bridge 패턴을 사용하는 이유

- 분리해 두면 확장이 편해진다
- 기능의 클래스 계층과 구현의 클래스 계층을 독립적으로 운영할 수 있음
- 기능 추가 → 형태(타입)의 확장
  - 기능의 클래스 계층에서 상속으로
- 구현 추가 → 구현의 확장(OCP)
  - 구현의 클래스 계층에서 구현 추가

- 상속은 강한 결합이고, 위임은 약한 결합