Command 패턴

2015004239 정성운

Command_Pattern

SeongWoon Jeong edited this page 3 minutes ago · 3 revisions

Software Engineering - CSE4006

Command Pattern

professor. 정소희

---

1. 개요

- Command Pattern에 대해서 이해한다.
  - Command Pattern : Command와 Concrete Command, Invoker와 Receiver로 구성된 패턴이다.

2. 과제 설명

- 주어진 소스코드의 TwoButtonController 클래스 설계의 취약점을 OCP 측면에서 설명하시오
  - OCP(Open Closed Principle) : 소프트웨어는 확장에는 열려있고, 수정에는 닫혀있어야 한다는 특성.
- Command Pattern을 활용해 TwoButtonController 클래스 설계의 취약점을 개선하는 방법을 설명하시오.

3. 사용언어 / 환경

- Java / Mac OS

[주어진 소스코드]

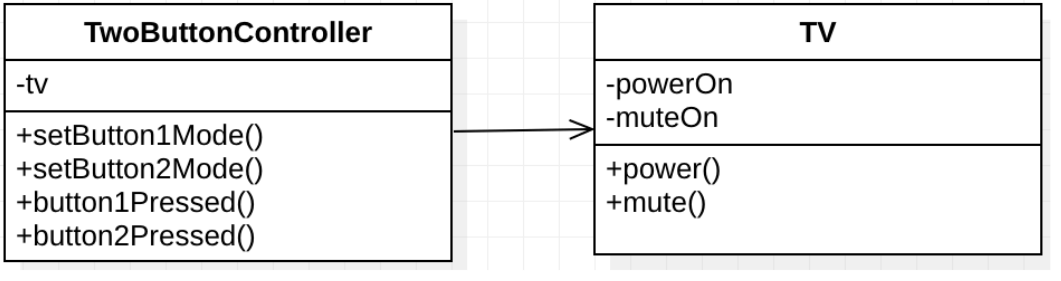
```java
package exercise02.command;

public class TwoButtonController {
    private TV tv;
    private Mode button1Mode;
    private Mode button2Mode;

    public TwoButtonController(TV tv) {
        this.tv = tv;
    }

    public void setButton1Mode(Mode button1Mode) {
        this.button1Mode = button1Mode;
    }

    public void setButton2Mode(Mode button2Mode) {
        this.button2Mode = button2Mode;
    }

    public void button1Pressed() {
        switch(button1Mode) {
        case POWER:
            tv.power(); break;
        case MUTE:
            tv.mute(); break;
        }
    }

    public void button2Pressed() {
        switch(button2Mode) {
        case POWER:
            tv.power(); break;
        case MUTE:
            tv.mute(); break;
        }
    }
}
```

```java
package exercise02.command;

public enum Mode {
    POWER, MUTE
}
```

```java
package exercise02.command;

public class TV {
    private boolean powerOn = false;
    private boolean muteOn = false;

    public void power() {
        powerOn = !powerOn;

        if(powerOn)
            System.out.println("Power On");
        else
            System.out.println("Power Off");
    }

    public void mute() {
        if(!powerOn)
            return;

        muteOn = !muteOn;

        if(muteOn)
            System.out.println("Mute On");
        else
            System.out.println("Mute Off");
```

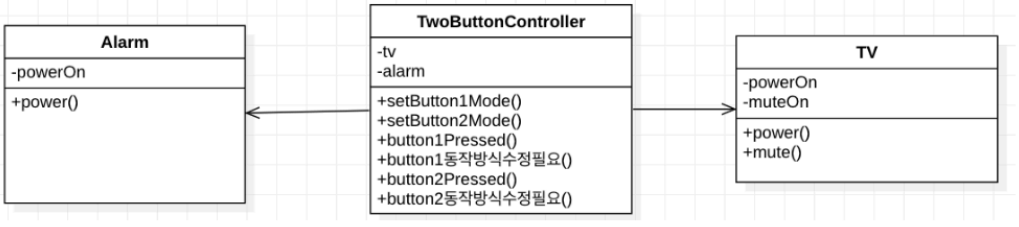
```java
package exercise02.command;

public class Client {
    public static void main(String[] args) {
        TV tv = new TV();
        TwoButtonController rc
            = new TwoButtonController(tv);
        rc.setButton1Mode(Mode.POWER);
        rc.setButton2Mode(Mode.MUTE);

        rc.button1Pressed() ; // Power On
        rc.button2Pressed() ; // Mute On
        rc.button1Pressed() ; // Power Off
        rc.button1Pressed() ; // Power On
        rc.button2Pressed() ; // Mute Off
        rc.button1Pressed() ; // Power Off
        rc.button2Pressed() ;
    }
}
```


TwoButtonController가 직접 TV 객체를 컨트롤하는 구조이다. (enum 클래스 생략)

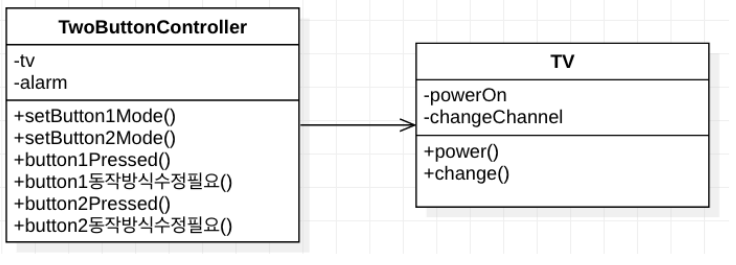


[주어진 소스코드의 취약점 분석]

- OCP의 측면에서 소프트웨어는 확장에 열려있고 변경에 닫혀있어야 하지만, 해당 설계는 확장에 닫혀있고 변경에 열려있다.
  - 만약, 새로운 Device를 추가하는 경우? TwoButtonController의 코드를 수정해야 한다.
  - Alarm이라는 Device가 추가되면, Button1Pressed()와 Button2Pressed()의 동작방식도 Alarm이라는 디바이스에 대해서도 동작할 수 있도록 변경해야 한다.



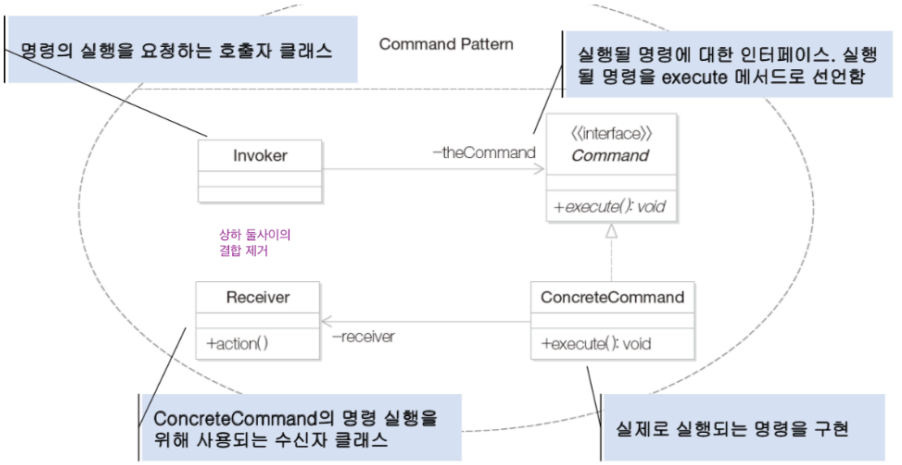
- 만약, 기존의 TV의 동작 방식을 변경하는 경우? TV의 코드와, TwoButtonController의 코드를 모두 변경해야 한다.
- TV의 동작방식이 Mute에서 ChannelChange로 변경되면, 해당 Controller의 코드 또한 변경되어야 한다.



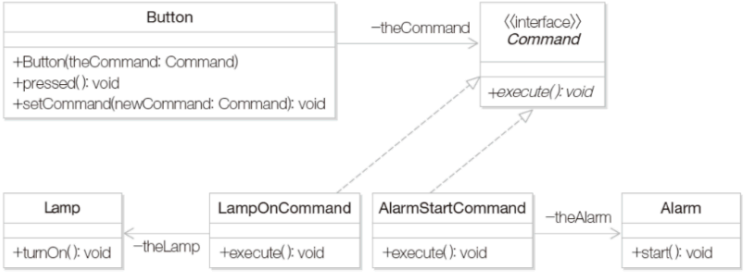
- 따라서 해당 코드는 OCP측면에서 취약한 코드이다.

[취약점 개선 - Command Pattern]

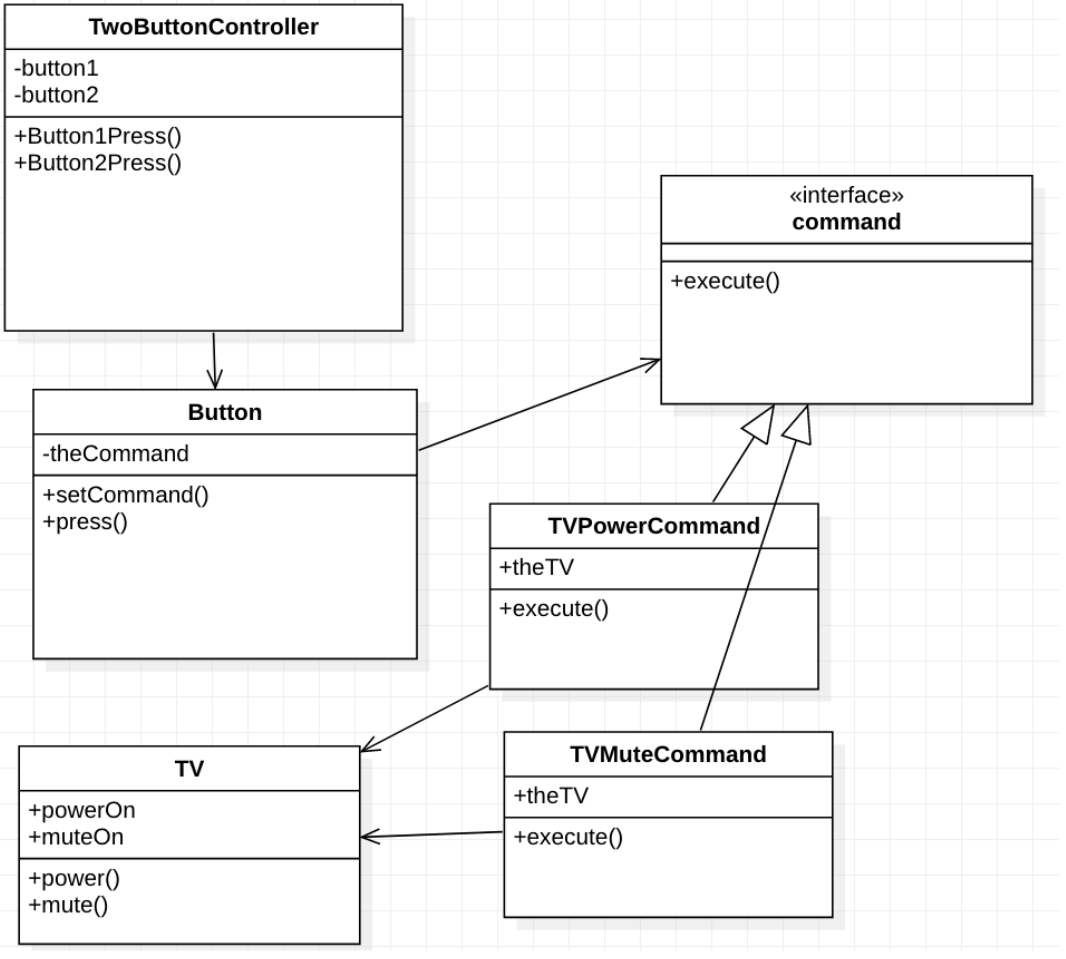
- Command Pattern은 아래의 4가지 요소로 구성된다.
  - Command Interface
  - Concrete Command Class
  - Invoker(호출자)
  - Receiver(피호출자)



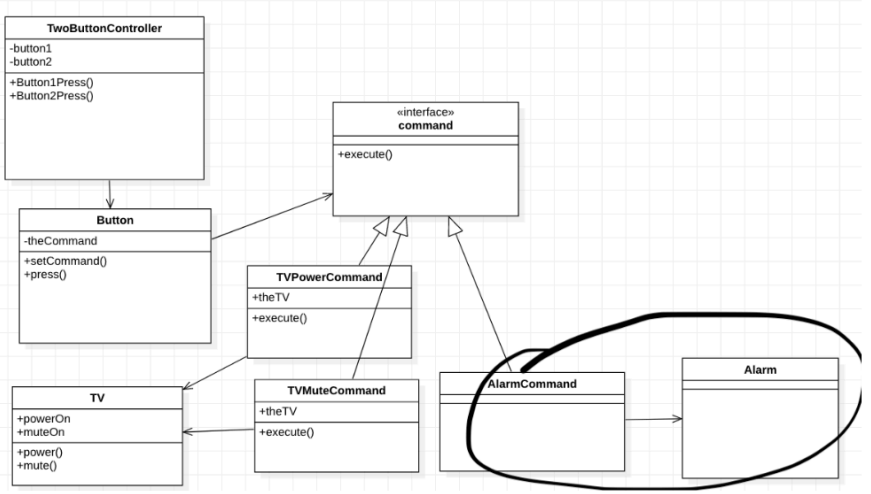
- 아래의 예는 4가지 요소를 적용한 적용 예시이다.
  - Command Interface : Command
  - Concrete Command Class : LampOnClass, AlarmOnClass
  - Invoker : Button
  - Receiver : Alarm, Lamp



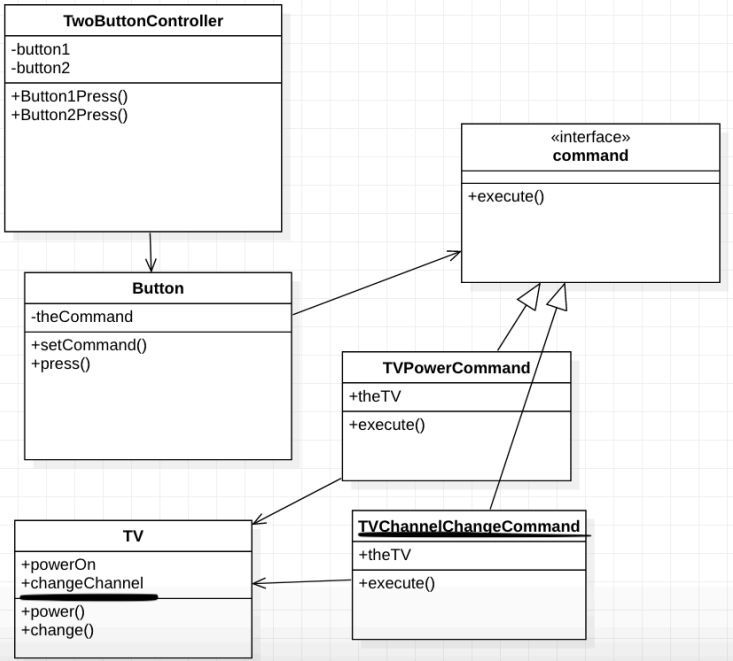
- 과제에서 주어진 소스코드를 Command Pattern을 적용한 결과이다. (인터페이스 구현 부분에서 정확하게는 실선이 아닌 점선이 맞음.)
    - Command Interface : Command
    - Concrete Command Class : TVPowerCommand, TVMuteCommand
    - Invoker : Button
    - Receiver : TV



- 해당 패턴을 적용한뒤에 얻을 수 있는 효과를 OCP의 측면에서 분석해보면,
    - 만약, 새로운 Deviece가 추가될 경우, Button과 TV, Command에 대한 수정 없이 Command의 추가와 Alarm의 추가만 있으면 된다.



- 만약, 기존의 TV의 동작방식을 변경하려면, 원래는 Controller까지 함께 바뀌어야 했으나, 이제는 Concrete Command만 변경해주면 된다.



[전체 소스코드]

[TwoButtonController.java] :

```
public class TwoButtonController {
        private Button button1;
        private Button button2;

        TwoButtonController(Button button1, Button button2
                this.button1 = button1;
                this.button2 = button2;
        }

        void Button1Press() {
                this.button1.pressed();
        }

        void Button2Press() {
                this.button2.pressed();
        }
}
```

두개의 버튼을 제어하는 컨트롤 클래스이다.

[Button.java] :

```
public class Button {
        private Command theCommand;

        public Button(Command theCommand) {
                setCommand(theCommand);
        }

        public void setCommand(Command newCommand) {
                this.theCommand = newCommand;
        }

        public void pressed() {
                theCommand.execute();
        }
}
```

컨트롤러에 의해 제어되는 버튼 클래스이다. Pressed()되면 세팅된 Command를 실행한다.

[Command.java] :

```
public interface Command {
        abstract public void execute();
}
```

Concrete Command구성을 위한 인터페이스이다.

[TVMuteCommand.java] :

```
public class TVMuteCommand implements Command {
        private TV theTV;
        public TVMuteCommand(TV theTV) {
                this.theTV = theTV;
        }
        public void execute() {
                theTV.mute();
        }
}
```

TV의 mute기능을 담당하는 Concrete Command이다.

[TVPowerCommand.java] :

```
public class TVPowerCommand implements Command {
        private TV theTV;
        public TVPowerCommand(TV theTV) {
                this.theTV = theTV;
        }
        public void execute() {
                theTV.power();
        }
}
```

TV의 power기능을 담당하는 Concrete Command이다.

[TV.java] :

```
public class TV {
        private boolean powerOn = false;
        private boolean muteOn = false;

        public void power() {
                powerOn = !powerOn;

                if (powerOn)
                        System.out.println("Power On");
                else
                        System.out.println("Power Off");
        }

        public void mute() {
                if (!powerOn)
                        return;

                muteOn = !muteOn;

                if(muteOn)
                        System.out.println("Mute On");
                else
                        System.out.println("Mute Off");
        }
}
```

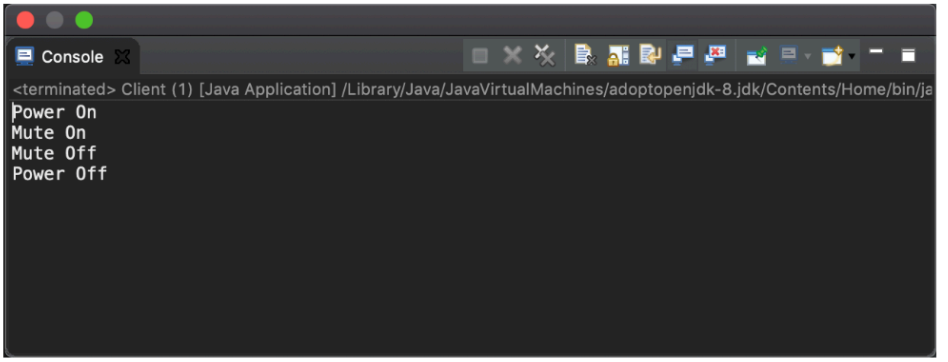
TV의 기능을 담당하는 기본 클래스이다.

[Client.java] :

```
public class Client {
        public static void main(String[] args) {
                TV tv = new TV();
                Command tvPowerCommand = new TVPowerComman
                Command tvMuteCommand = new TVMuteCommand(
                Button button1 = new Button(tvPowerCommand
                Button button2 = new Button(tvMuteCommand)
                TwoButtonController control = new TwoButto

                control.Button1Press();
                control.Button2Press();
                button1.setCommand(tvMuteCommand);
                button2.setCommand(tvPowerCommand);
                control.Button1Press();
                control.Button2Press();
        }
}
```

버튼 조작을 하는 클라이언트를 가정한 클래스이다.

[클라이언트 실행 결과 예시]



- Controller에 의해 Button조작을 하고, button1, button2의 동작을 바꾸어서 다시 제어해보았다.