커맨드 패턴
(Command Pattern)

이관우

kwlee@hansung.ac.kr

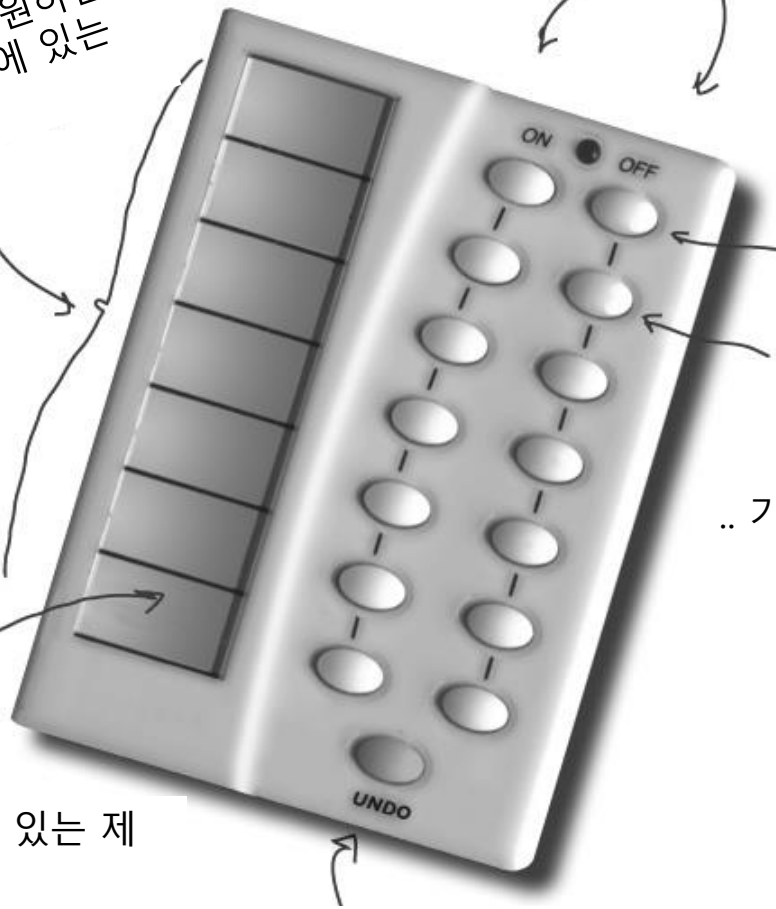
학습 목표

- 메소드 호출의 캡슐화가 필요한 상황을 이해한다.
- 메소드 호출을 캡슐화하는 커맨드 패턴을 이해한다.
- 커맨드 패턴으로 취소 기능을 구현해 본다.
- 커맨드 패턴의 활용 예를 알아본다.

홈 오토메이션 리모컨

일곱 개의 슬롯마다 각각 "ON" 버튼과 "OFF" 버튼이 있음

일곱가지 프로그래밍이 가능한 슬롯이 있고, 각 슬롯에 원하는 제품을 연결한 다음 옆에 있는 버튼을 가지고 조작



이 두 버튼으로 1번 슬롯에 연결된 가전제품을 제어

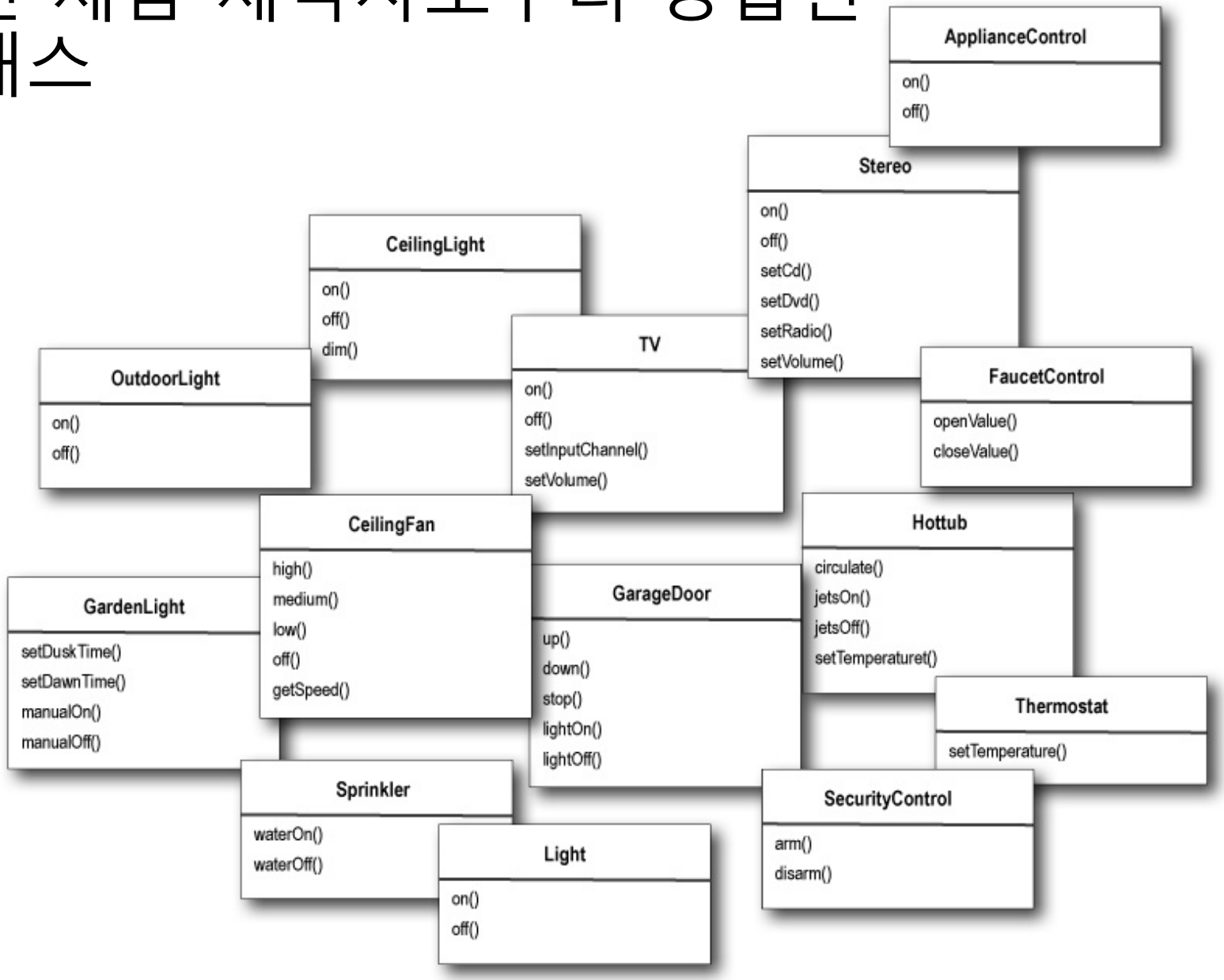
이 두 버튼으로 2번 슬롯에 연결된 가전제품을 제어

.. 기타 등등

여기에다가 제어할 수 있는 제품 이름을 작성

마지막으로 누른 버튼에 대한 명령을 취소

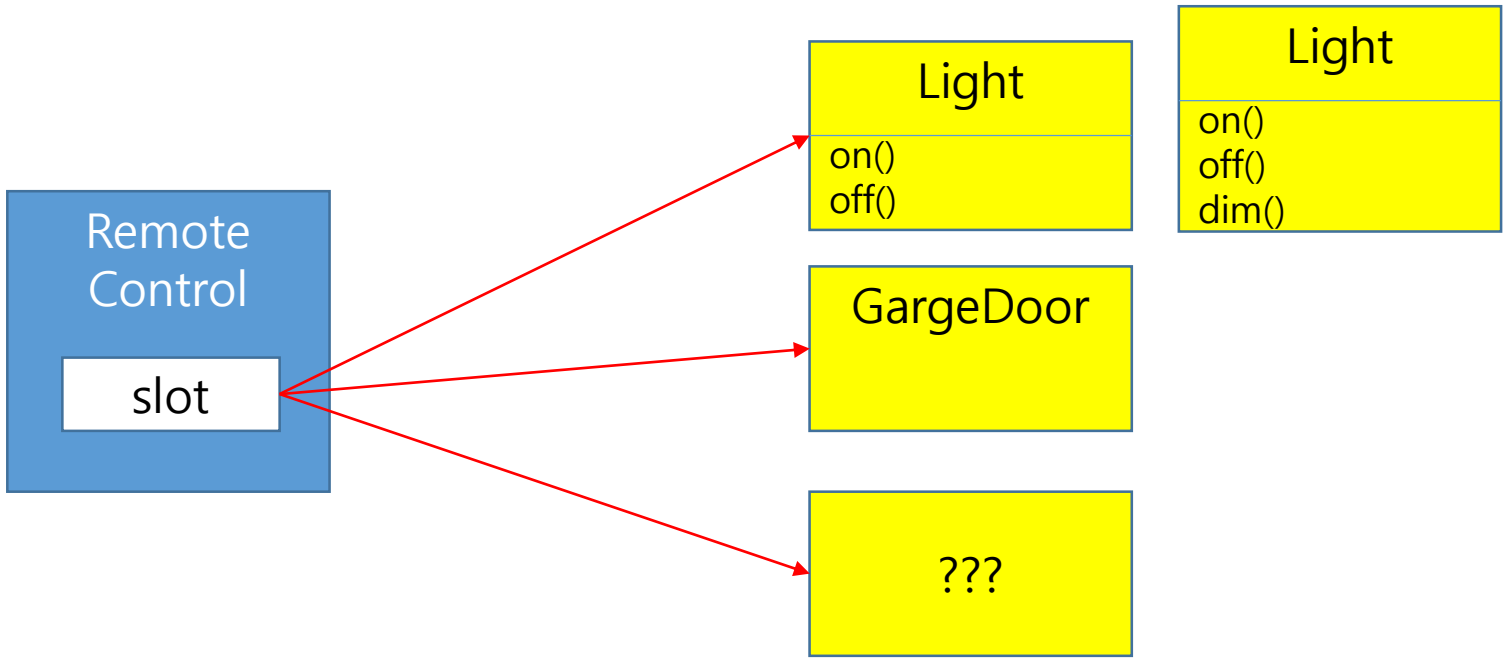
가전 제품 제작사로부터 공급된 클래스



디자인 고려사항

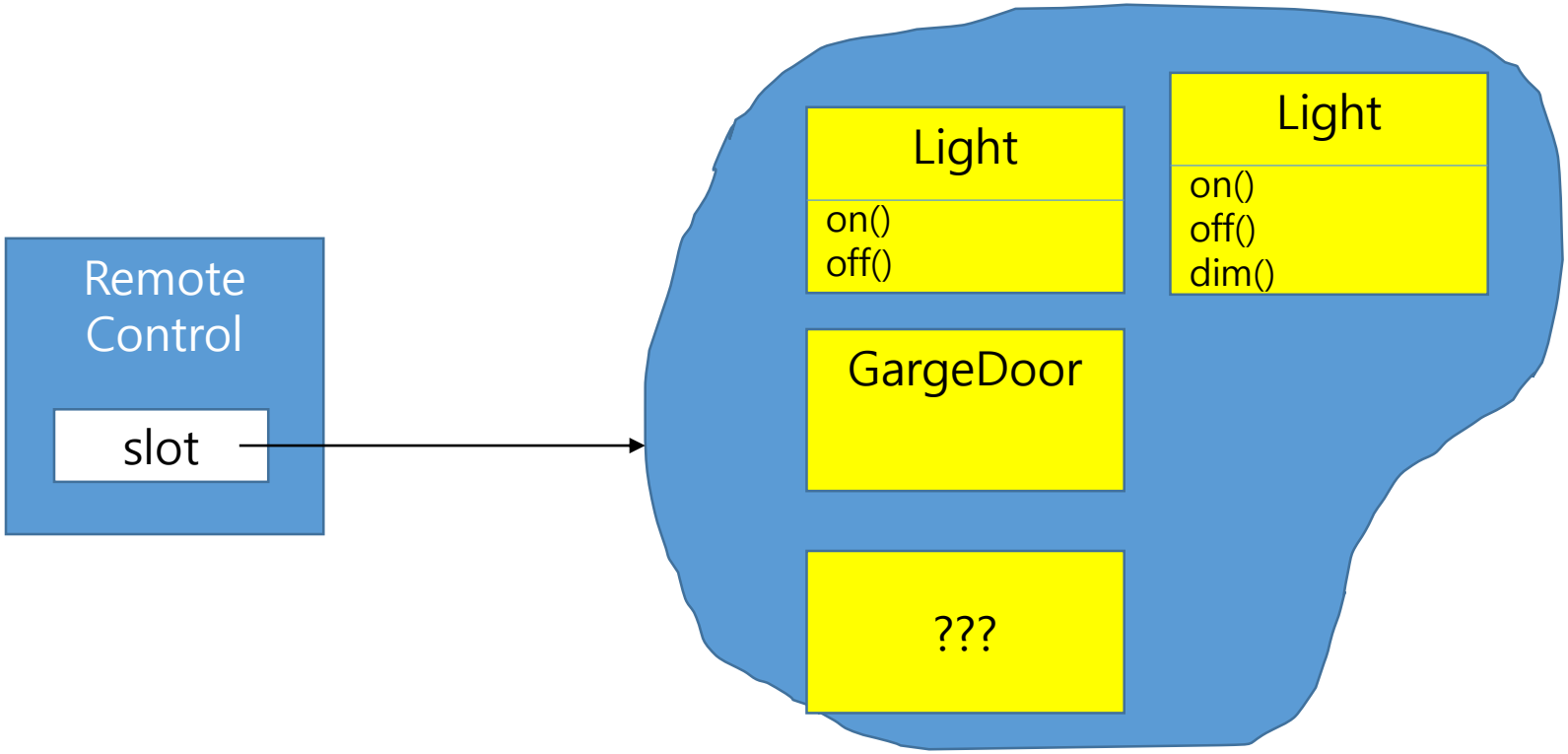
- 제어해야할 가전 제품 클래스의 인터페이스가 제작사별로 상이할수도 있음.

- 새로이 제어해야할 가전 제품 클래스가 추가 될 수 있음.



디자인 방향

- 리모컨의 특정한 슬롯에 다양한 제작사의 혹은 다양한 종류의 가전제품 클래스를 연결시키더라도 리모컨 클래스에 영향을 주지 않도록 하는 설계는?



호출 캡슐화

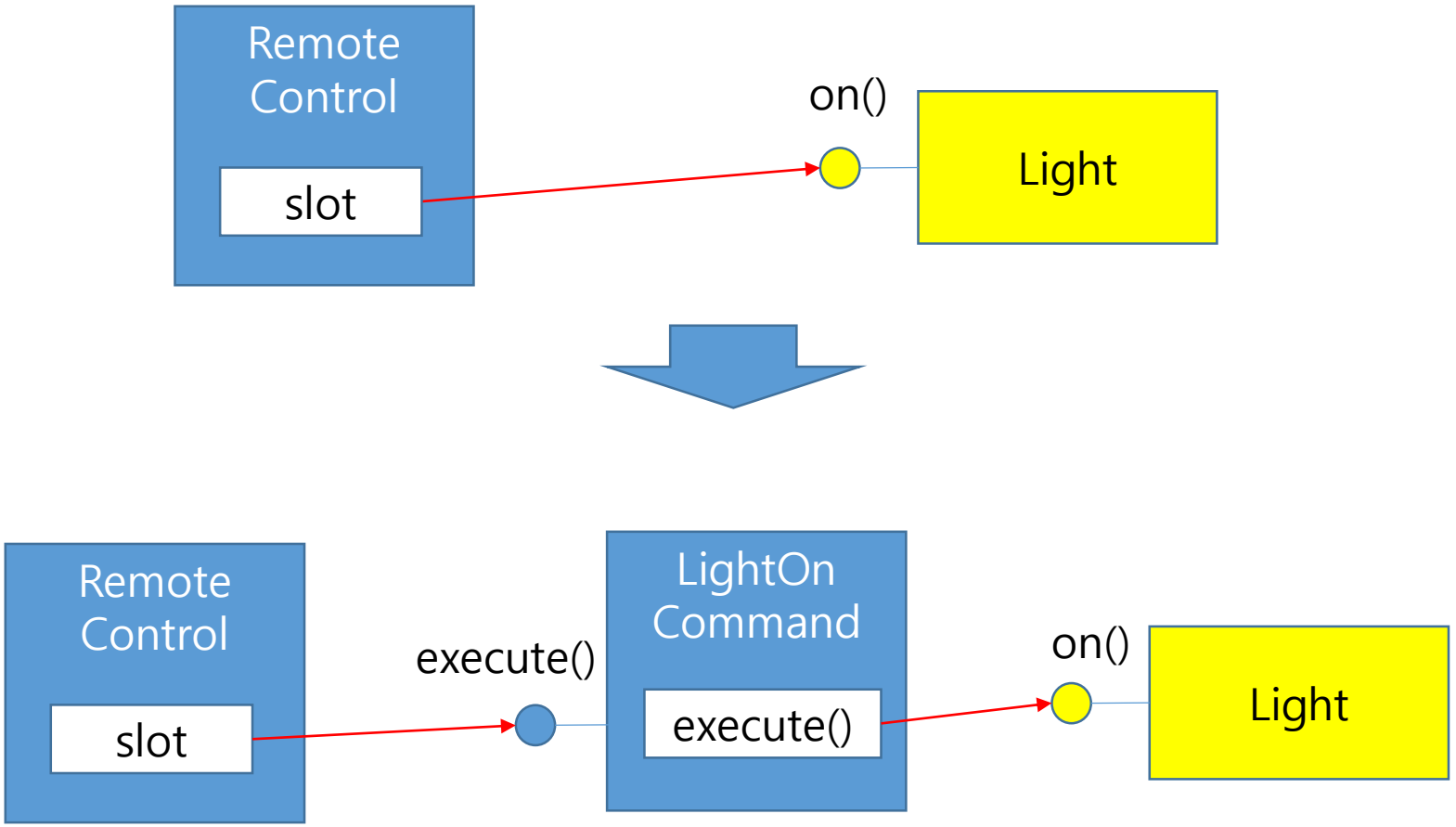
- Command 인터페이스
  - 각 가전제품의 메소드 호출을 캡슐화하는 공통 인터페이스

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

}
```

- 전등을 켜기 위한 Command 인터페이스 구현

```
public class LightOnCommand implements Command {
        Light light;
        public LightOnCommand(Light light) {
                this.light = light;
        }
        public void execute() {
                light.on();
        }
}
```

메소드 호출 캡슐화


Remote Control
slot
on()
Light
Remote Control
slot
execute()
LightOn Command
execute()
on()
Light

커맨트 객체 사용하기

- 슬롯이 하나 밖에 없는 리모콘이라고 가정

```
public class SimpleRemoteControl {
        Command slot;

        public SimpleRemoteControl() {}

        public void setCommand(Command command) {
                slot = command;
        }

        public void buttonWasPressed() {
                slot.execute();
        }
}
```

슬롯에 커맨트 객체 설정

이 메소드를 통해 리모콘 버튼의 기능을 바꿀수 있음

버튼이 눌러지면 호출됨

슬롯에 연결된 커맨트 객체의 execture()메소드 호출

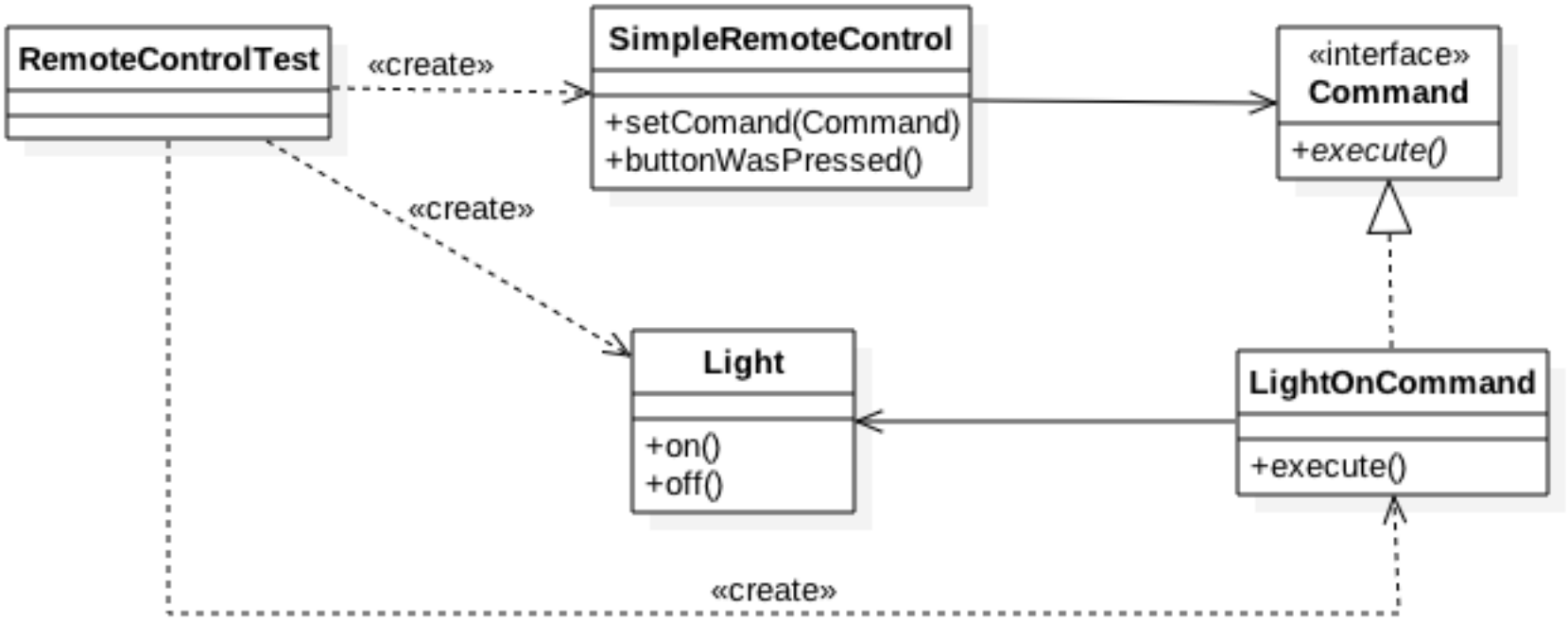
리모컨 사용을 위한 테스트 클래스

```
public class RemoteControlTest {
    public static void main(String[] args) {
        SimpleRemoteControl remote = new SimpleRemoteControl();
        Light light = new Light();
        LightOnCommand lightOn = new LightOnCommand(light);

        remote.setCommand(lightOn);
        remote.buttonWasPressed();
    }

}
```

예제 소스:

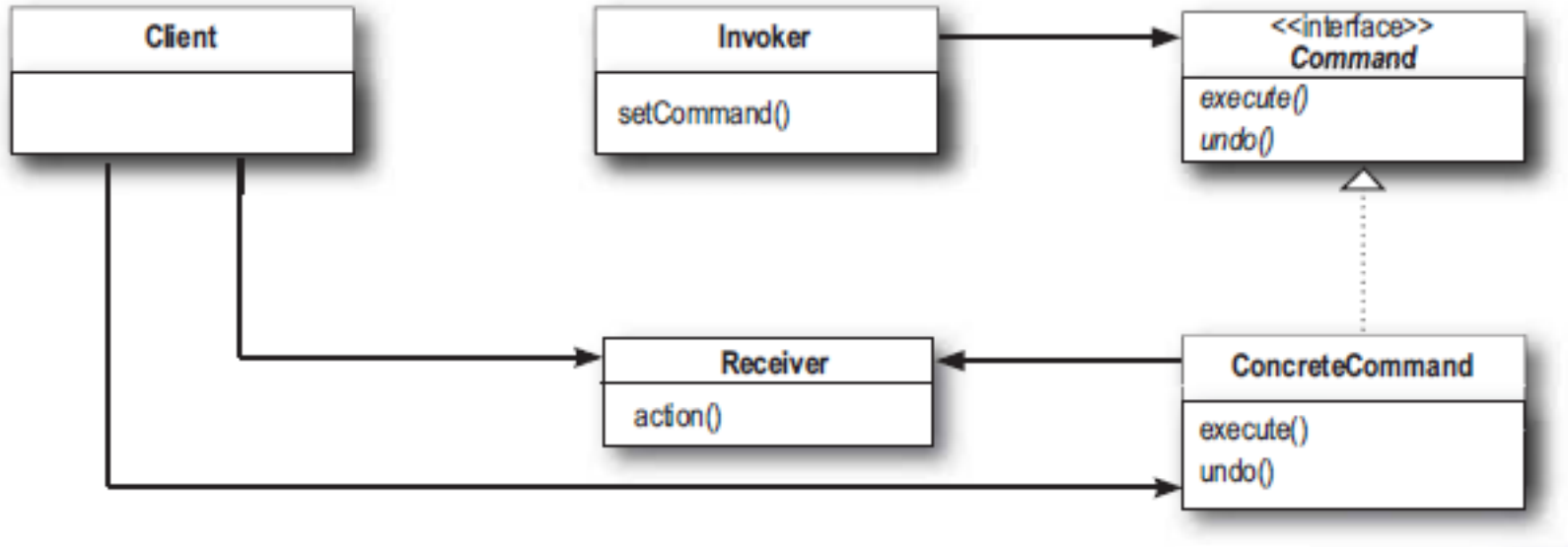
https://github.com/kwanulee/DesignPattern/tree/master/command/simpleremote

클래스 다이어그램


RemoteControlTest
«create»
SimpleRemoteControl
+setComand(Command)
+buttonWasPressed()
«interface»
Command
+execute()
«create»
Light
+on()
+off()
LightOnCommand
+execute()
«create»

커맨드 패턴

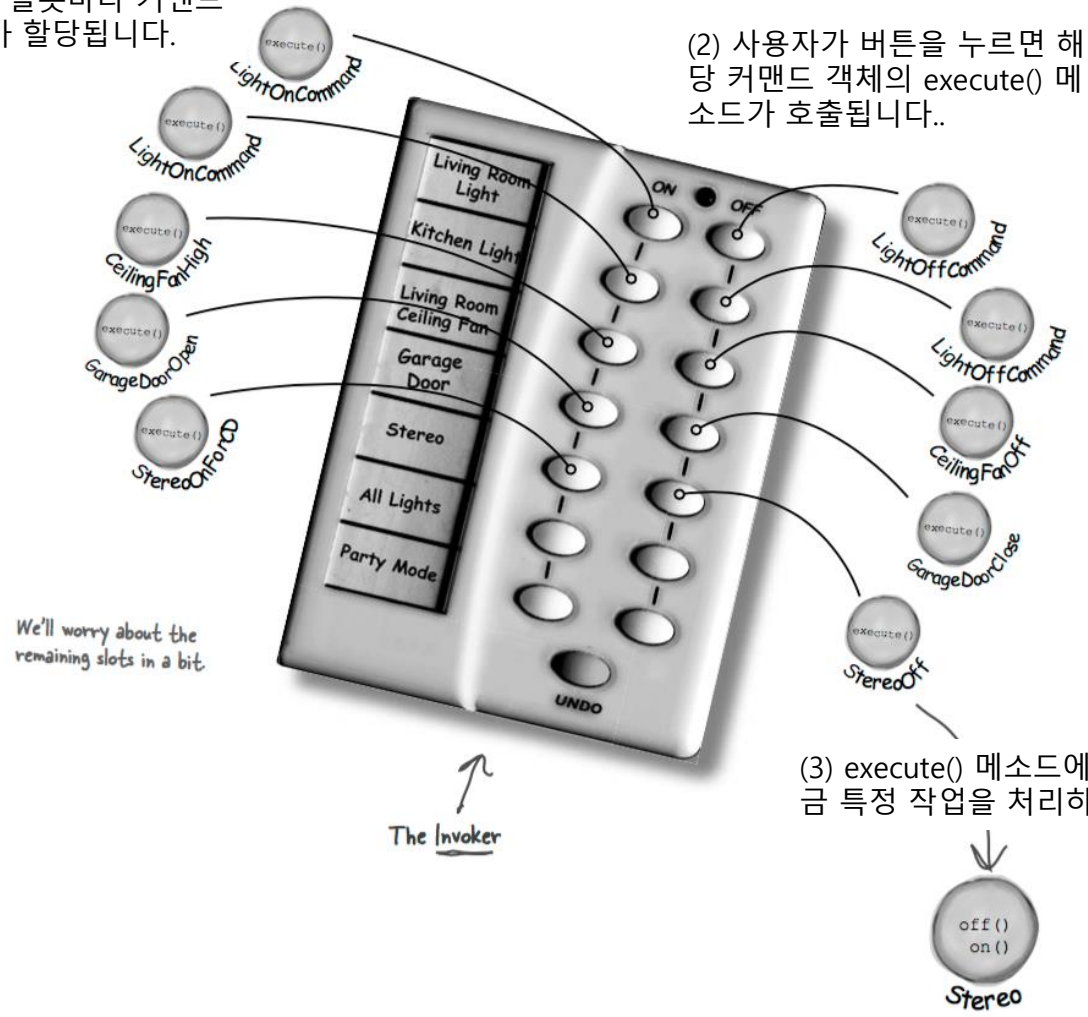
- 커맨트 패턴을 이용하면 메소드 호출을 커맨드 객체로 캡슐화할 수 있다.
- 매개변수를 써서 Invoker에 여러 가지 다른 커맨드 객체를 연결할 수 도 있습니다.



슬롯에 커맨트 객체 할당하기

(1) 각 슬롯마다 커맨드 객체가 할당됩니다.

(2) 사용자가 버튼을 누르면 해당 커맨드 객체의 execute() 메소드가 호출됩니다..



(3) execute() 메소드에서는 리시버로 하여금 특정 작업을 처리하도록 지시합니다.

리모컨 코드

```
public class RemoteControl {
    Command[] onCommands;
    Command[] offCommands;

    public RemoteControl() {
        onCommands = new Command[7];
        offCommands = new Command[7];

        Command noCommand = new NoCommand();
        for (int i = 0; i < 7; i++) {
                onCommands[i] = noCommand;
                offCommands[i] = noCommand;
        }
    }

    public void setCommand(int slot, Command onCommand, Command offCommand) {
        onCommands[slot] = onCommand;
        offCommands[slot] = offCommand;
    }
```

리모컨 코드

```
    public void onButtonWasPushed(int slot) {
        onCommands[slot].execute();
    }

    public void offButtonWasPushed(int slot) {
        offCommands[slot].execute();
    }

    public String toString() {
        StringBuffer stringBuff = new StringBuffer();
        stringBuff.append("\n------ Remote Control -------\n");
        for (int i = 0; i < onCommands.length; i++) {
            stringBuff.append("[slot " + i + "] " +
                onCommands[i].getClass().getName()
                + "    " +
                offCommands[i].getClass().getName() + "\n");
        }
        return stringBuff.toString();
    }
}
```

커맨트 클래스

```
public class LightOffCommand implements Command {
        Light light;

        public LightOffCommand(Light light) {
                this.light = light;
        }

        public void execute() {
                light.off();
        }
}
```

커맨트 클래스

```
public class StereoOnWithCDCommand implements Command {
        Stereo stereo;

        public StereoOnWithCDCommand(Stereo stereo) {
                this.stereo = stereo;
        }

        public void execute() {
                stereo.on();
                stereo.setCD();
                stereo.setVolume(11);
        }
}
```

리모컨 테스트

```
public class RemoteLoader {
    public static void main(String[] args) {
        RemoteControl remoteControl = new RemoteControl();

        Light livingRoomLight = new Light("Living Room");
        Light kitchenLight = new Light("Kitchen");
        …

        LightOnCommand livingRoomLightOn = new LightOnCommand(livingRoomLight);
        LightOffCommand livingRoomLightOff = new LightOffCommand(livingRoomLight);
        LightOnCommand kitchenLightOn =      new LightOnCommand(kitchenLight);
        LightOffCommand kitchenLightOff = new LightOffCommand(kitchenLight);
        …

        remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);
        remoteControl.setCommand(1, kitchenLightOn, kitchenLightOff);
        …

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);
        remoteControl.onButtonWasPushed(1);
        remoteControl.offButtonWasPushed(1);
        …
```

예제소스 https://github.com/kwanulee/DesignPattern/tree/master/command/remote

Undo 기능

- 작업 취소 기능 지원

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

```
public class LightOnCommand implements Command {
        Light light;
        public LightOnCommand(Light light) {
                this.light = light;
        }
        public void execute() {
                light.on();
        }
        public void undo() {
                light.off();
        }
}
```

Undo 기능

- 마지막으로 실행된 명령을 기록

```
public class RemoteControlWithUndo {
        Command[] onCommands;
        Command[] offCommands;
        Command undoCommand;
        …
        public void onButtonWasPushed(int slot) {
                onCommands[slot].execute();
                undoCommand = onCommands[slot];
        }

        public void offButtonWasPushed(int slot) {
                offCommands[slot].execute();
                undoCommand = offCommands[slot];
        }

        public void undoButtonWasPushed() {
                undoCommand.undo();
        }
```

Undo 버튼 테스트

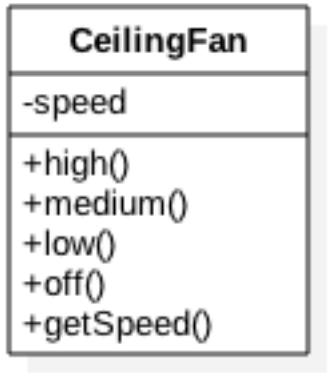
```
public class RemoteLoader {
    public static void main(String[] args) {
        RemoteControlWithUndo remoteControl = new RemoteControlWithUndo();

        Light livingRoomLight = new Light("Living Room");

        LightOnCommand livingRoomLightOn =
                                new LightOnCommand(livingRoomLight);
        LightOffCommand livingRoomLightOff =
                                new LightOffCommand(livingRoomLight);

        remoteControl.setCommand(0, livingRoomLightOn, livingRoomLightOff);

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);
        remoteControl.undoButtonWasPushed();

        remoteControl.offButtonWasPushed(0);
        remoteControl.onButtonWasPushed(0);
        remoteControl.undoButtonWasPushed();
```

예제소스 https://github.com/kwanulee/DesignPattern/tree/master/command/undo

작업 취소 기능을 구현할 때 상태를 사용하는 방법

- 선풍기의 속도를 상태로 관리

```
public class CeilingFan {
          public static final int HIGH = 3;
          public static final int MEDIUM = 2;
          public static final int LOW = 1;
          public static final int OFF = 0;
          String location;
          int speed;

          public CeilingFan(String location) {
                    this.location = location;
                    speed = OFF;
          }

          public void high() { speed = HIGH; }
          public void medium() {speed = MEDIUM; }
          public void low() { speed = LOW; }
          public void off() { speed = OFF; }
          public int getSpeed() { return speed; }
}
```



작업 취소 기능을 구현할 때 상태를 사용하는 방법

- 선풍기의 이전속도를 저장 후, undo 메소드 호출 시에 사용

```
public class CeilingFanHighCommand implements Command {
        CeilingFan ceilingFan;
        int prevSpeed;

        public CeilingFanHighCommand(CeilingFan ceilingFan) {
                this.ceilingFan = ceilingFan;
        }

        public void execute() {
                prevSpeed = ceilingFan.getSpeed();
                ceilingFan.high();
        }

        public void undo() {
                if (prevSpeed == CeilingFan.HIGH) {
                        ceilingFan.high();
                } else if (prevSpeed == CeilingFan.MEDIUM) {
                        ceilingFan.medium();
                …
        }
}
```

선풍기 테스트

```
public class RemoteLoader {
    public static void main(String[] args) {
        RemoteControlWithUndo remoteControl = new RemoteControlWithUndo();

        CeilingFan ceilingFan = new CeilingFan("Living Room");
        CeilingFanMediumCommand ceilingFanMedium =
                                new CeilingFanMediumCommand(ceilingFan);
        CeilingFanHighCommand ceilingFanHigh =
                                new CeilingFanHighCommand(ceilingFan);
        CeilingFanOffCommand ceilingFanOff =
                                new CeilingFanOffCommand(ceilingFan);

        remoteControl.setCommand(0, ceilingFanMedium, ceilingFanOff);
        remoteControl.setCommand(1, ceilingFanHigh, ceilingFanOff);

        remoteControl.onButtonWasPushed(0);
        remoteControl.offButtonWasPushed(0);
        remoteControl.undoButtonWasPushed();

        remoteControl.onButtonWasPushed(1);
        remoteControl.undoButtonWasPushed();
```

예제소스 https://github.com/kwanulee/DesignPattern/tree/master/command/undo

리모컨에 파티 모드를...

- 버튼 한 개만 누르면 전등이 어두워지면서 오디오, TV가 켜지고, DVD 모드로 변경되고, 욕조에 물이 채워지는 것까지 한꺼번에 처리하는 기능을 구현해 봅니다.

1 여러 가지 커맨드를 헌꺼번에 실행시킬 수 있는 MacroCommand

```
public class MacroCommand implements Command {
        Command[] commands;

        public MacroCommand(Command[] commands) {
                this.commands = commands;
        }

        public void execute() {
                for (int i = 0; i < commands.length; i++) {
                        commands[i].execute();
                }
        }
```

리모컨에 파티 모드를...

2 MacroCommand에 집어 넣을 일련의 커맨드들 생성

```
Light light = new Light("Living Room");
TV tv = new TV("Living Room");
Stereo stereo = new Stereo("Living Room");
Hottub hottub = new Hottub();

LightOnCommand lightOn = new LightOnCommand(light);
StereoOnCommand stereoOn = new StereoOnCommand(stereo);
TVOnCommand tvOn = new TVOnCommand(tv);
HottubOnCommand hottubOn = new HottubOnCommand(hottub);

LightOffCommand lightOff = new LightOffCommand(light);
StereoOffCommand stereoOff = new StereoOffCommand(stereo);
TVOffCommand tvOff = new TVOffCommand(tv);
HottubOffCommand hottubOff = new HottubOffCommand(hottub);
```

리모컨에 파티 모드를...

3 ON 커맨드와 OFF 커맨드를 위한 배열 준비하고, MacroCommand 객체 생성

```
Command[] partyOn = { lightOn, stereoOn, tvOn, hottubOn};
Command[] partyOff = { lightOff, stereoOff, tvOff, hottubOff};
MacroCommand partyOnMacro = new MacroCommand(partyOn);
MacroCommand partyOffMacro = new MacroCommand(partyOff);
```

4 MacroCommand 객체를 버튼에 할당

```
remoteControl.setCommand(0, partyOnMacro, partyOffMacro);
```

5 테스트

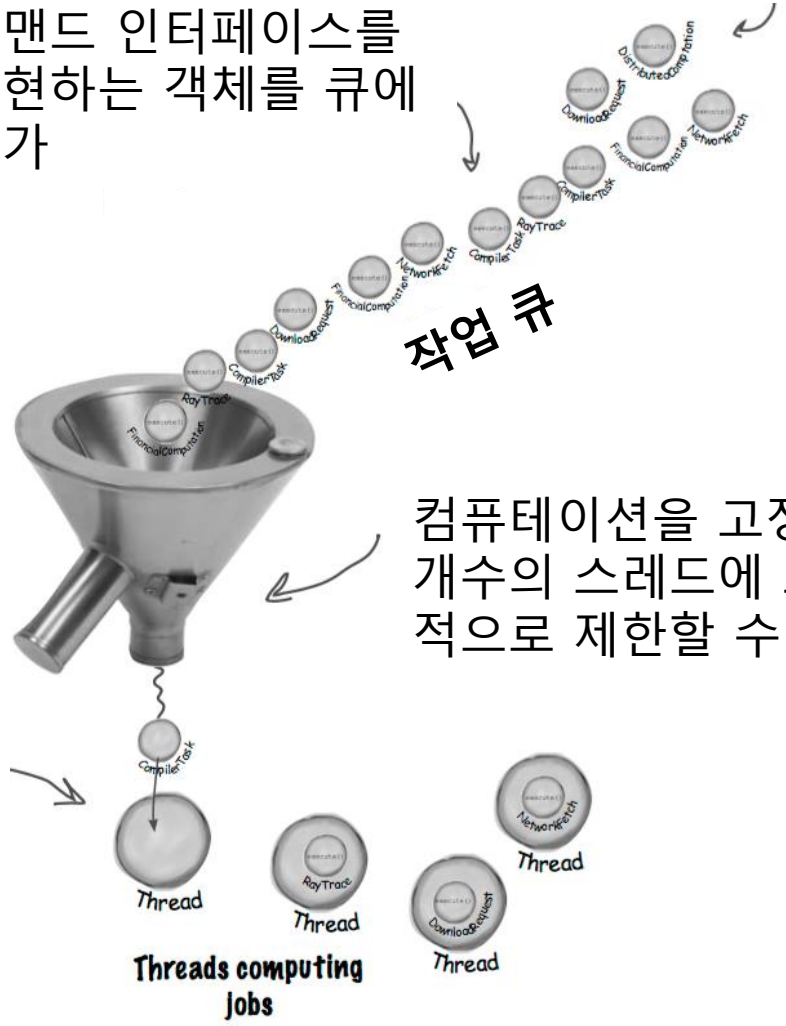
```
remoteControl.onButtonWasPushed(0);
remoteControl.offButtonWasPushed(0);
```

예제소스 https://github.com/kwanulee/DesignPattern/tree/master/command/party

커맨드 패턴 활용: 요청을 큐에 저장하기

커맨드

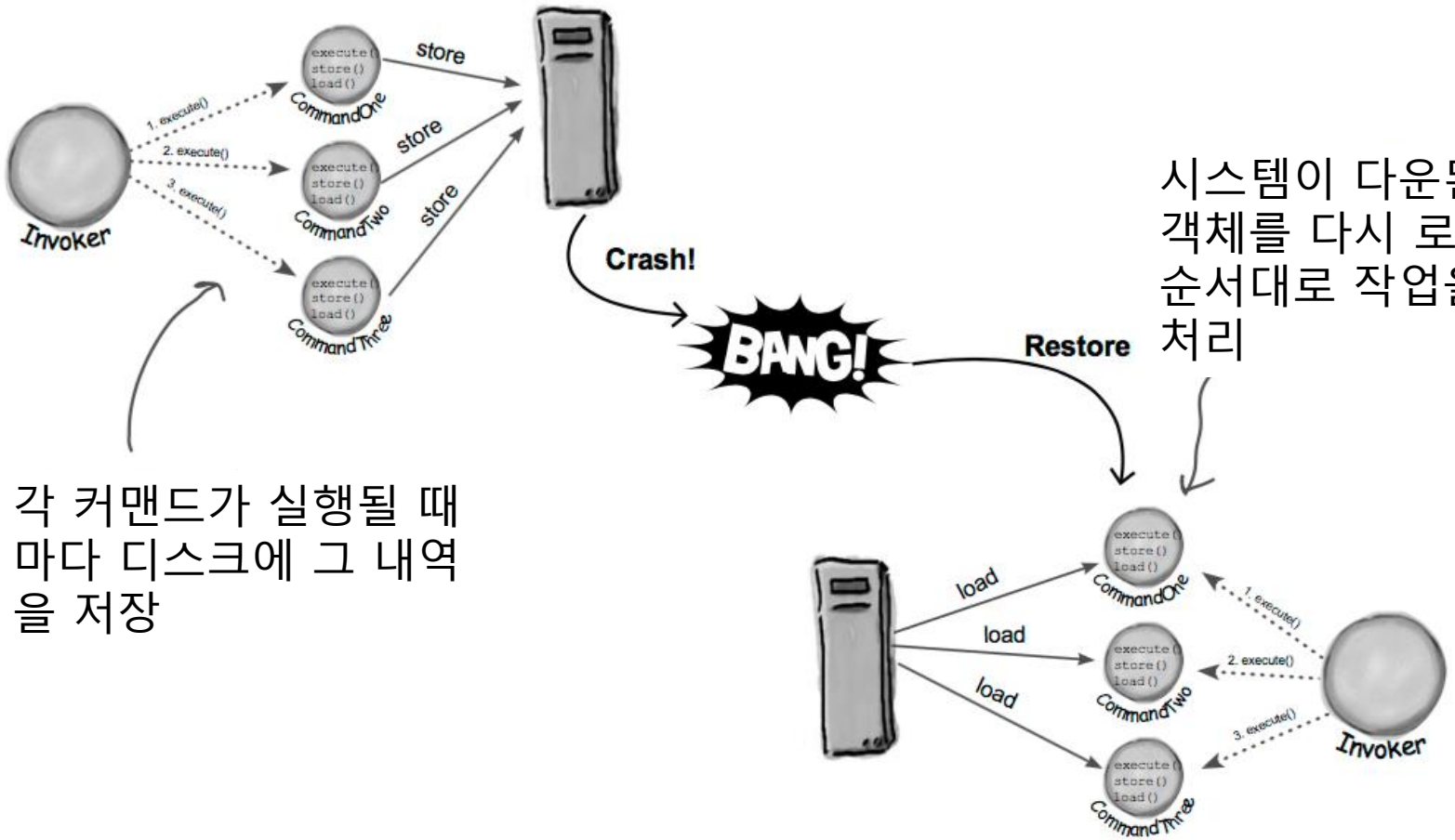
커맨드 인터페이스를 구현하는 객체를 큐에 추가



컴퓨테이션을 고정된 개수의 스레드에 효율적으로 제한할 수 있음

스레드에서는 큐로부터 커맨드를 하나씩 제거하면서 커맨드의 execute() 메소드를 호출

커맨드 패턴 활용: 요청을 로그에 기록



각 커맨드가 실행될 때 마다 디스크에 그 내역을 저장

시스템이 다운된 후에, 객체를 다시 로딩하고 순서대로 작업을 다시 처리

핵심 정리

- Command 패턴을 이용하면 요청을 하는 객체와 그 요청을 수행하는 객체를 분리할 수 있다.
- Command 객체는 Action 을 수행하는 Receiver 를 캡슐화 한다.
- Command 객체의 execute() 는 Receiver 의 Action 을 호출한다.
- Macro Command 는 여러 개의 Command 를 한꺼번에 호출할 수 있게 해주는 간단한 방법이다.
- Command 패턴을 이용하면 작업 취소(Undo) 기능을 지원할 수 있고, 로그나 트랜잭션 시스템 구현에도 활용된다.