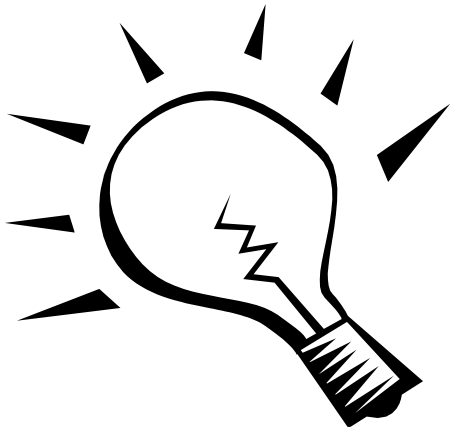
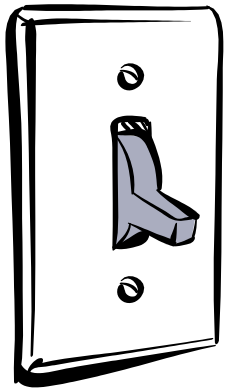
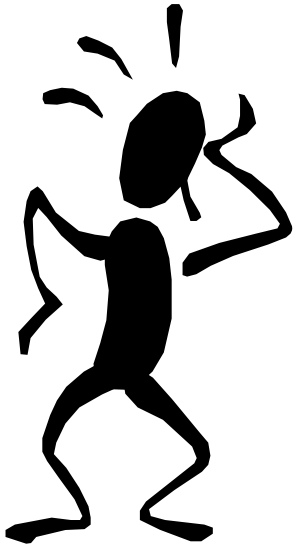
6. 커맨드 패턴

호출 캡슐화

거실의 전등은 어떻게 켜나요?

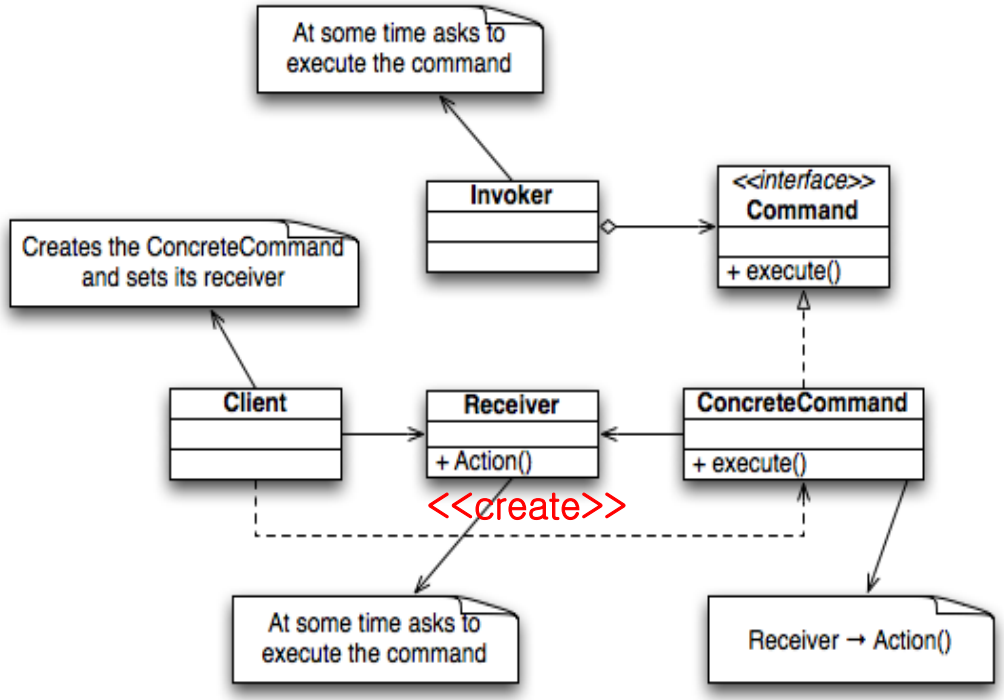






Command Pattern

- Encapsulate a request as an (command) object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations
- **문제점**: 요청할 오퍼레이션이나 요청 수신자에 대해 몰라도 객체에 대한 요청을 할 필요가 있다.
- **해결책**: Command 인터페이스를 사용하여 어떤 작업을 요청하는 쪽과 그 작업을 처리하는 쪽을 분리시켜라.

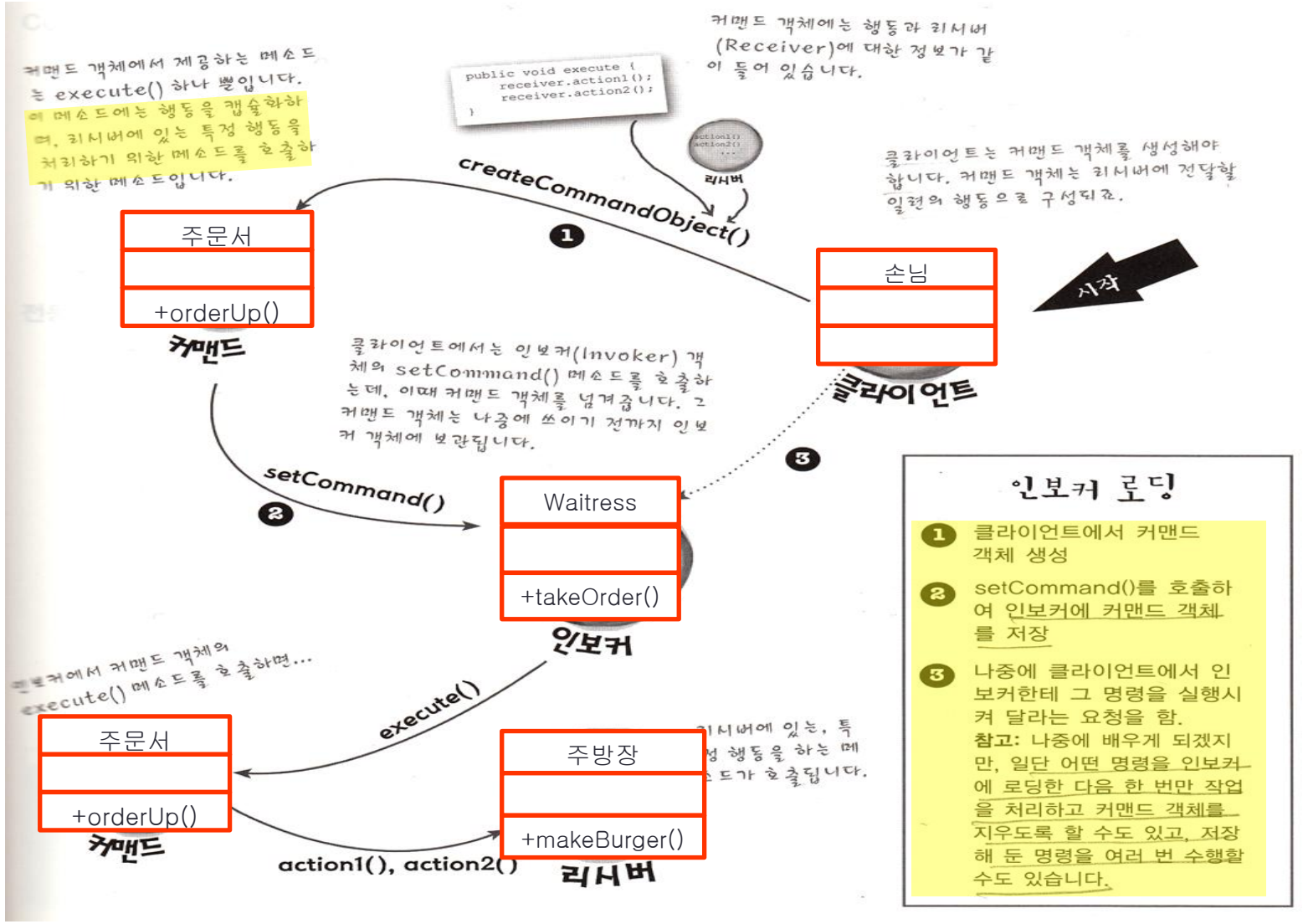


객체마을 식당에서는...

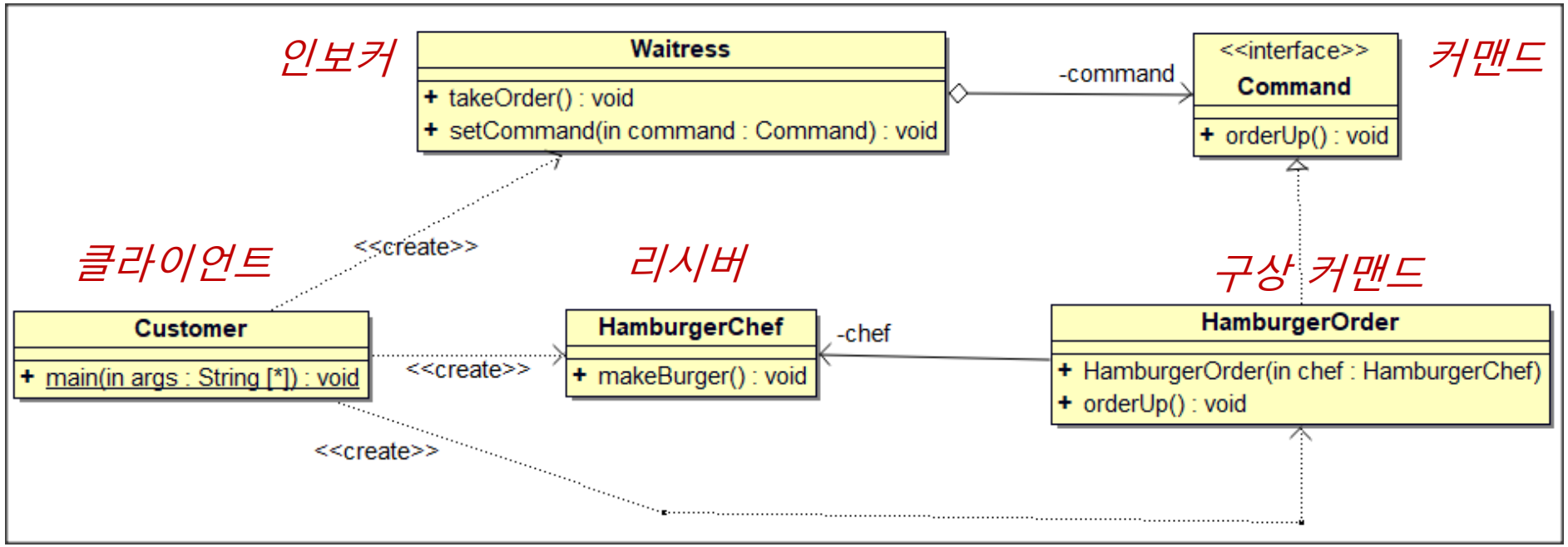

ORDER
1 고객이 웨이트리스한테 주문을 합니다.
2 웨이트리스는 주문을 받아서 카운터에 갖다 주고는 "주문 받아요!"라고 얘기합니다.
3 주방장이 주문대로 음식을 준비합니다.

객체와 메소드를 이용해서 표현

객체마을 식당과 커맨드 패턴



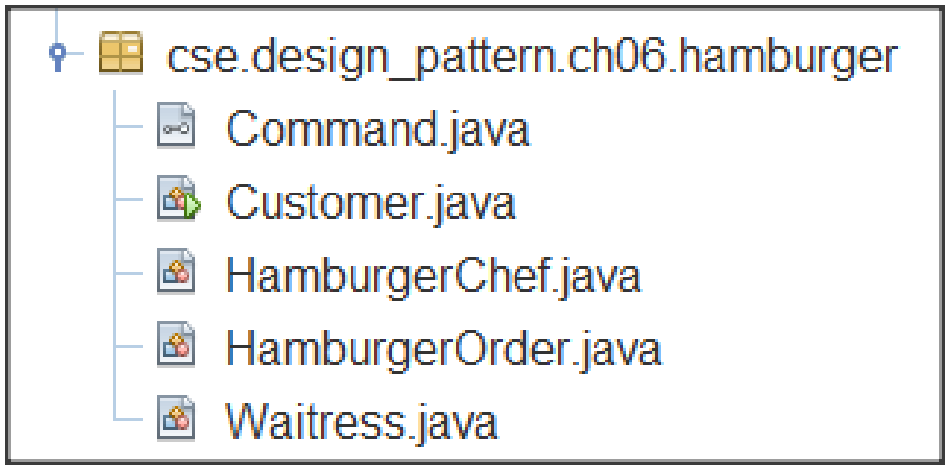
객체마을 식당 클래스 다이어그램



- 인보커인 웨이트리스는 무엇을 만드는지 구체적으로 몰라도 커맨드에 있는 orderUp() 메소드만 불러주면 리시버에서 알아서 일을 하도록 한다.

실습: hamburger

- 구성 클래스



Command.java

```
1  package cse.design_pattern.ch06.hamburger;
2
3  public interface Command {
4
5      void orderUp();
6
7  }
```

HamburgerOrder.java

- 구체적으로 햄버거 만드는 사람이 누구인지 알고 있어야만 함
  → 연관 관계로 표현: HamburgerOrder → HamburgerChef

```
2   package cse.design_pattern.ch06.hamburger;
3
4   public class HamburgerOrder implements Command {
5    private HamburgerChef chef;
6
7    public HamburgerOrder(HamburgerChef chef) {
8       this.chef = chef;
9    }
10
12   public void orderUp() {
13      System.out.println("햄버거 주문서가 주방에 전달됩니다.");
14      chef.makeBurger();
15   }
16
16  }
```

HamburgerChef.java

```java
package cse.design_pattern.ch06.hamburger;

public class HamburgerChef {

    public void makeBurger() {
        System.out.println("주방에서 햄버그를 만듭니다.");
    }

}
```

Waitress.java

- 인보커인 웨이트리스는 Command 인터페이스에 의존하여 코딩
- 리시버나 Concrete Command에 대해서 몰라도 됨

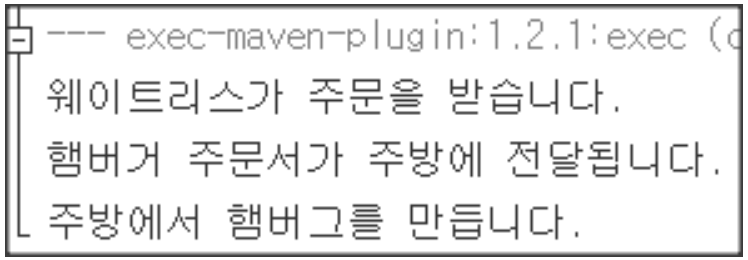
```
2   package cse.design_pattern.ch06.hamburger;
3
4   public class Waitress {
5    private Command command;
6
7    public void takeOrder() {
8      System.out.println("웨이트리스가 주문을 받습니다.");
9      command.orderUp();
10   }
11
12   public  void  setCommand(Command command) {
13     this.command = command;
14   }
15
16  }
```

Customer.java

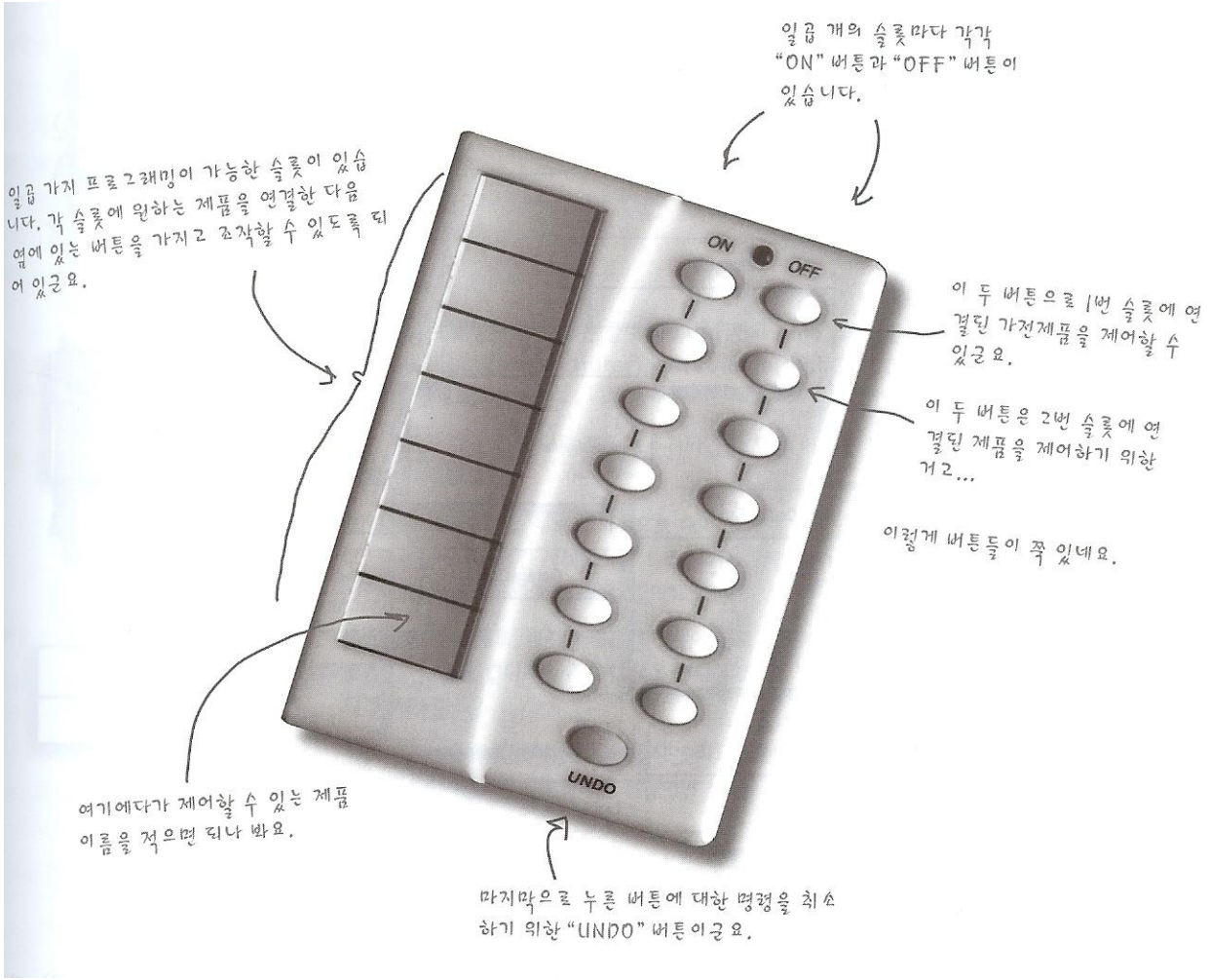
```
1   package cse.design_pattern.ch06.hamburger;
2
3   public class Customer {
4
5       public static void main(String[] args) {
6           Waitress waitress = new Waitress();
7
8           HamburgerChef chef = new HamburgerChef();
9           HamburgerOrder orderList = new HamburgerOrder(chef);
10
11          waitress.setCommand(orderList);
12          waitress.takeOrder();
13      }
14
15  }
```

햄버거를 만드는 주문서를 만들어 웨이트리스에게 알려준다

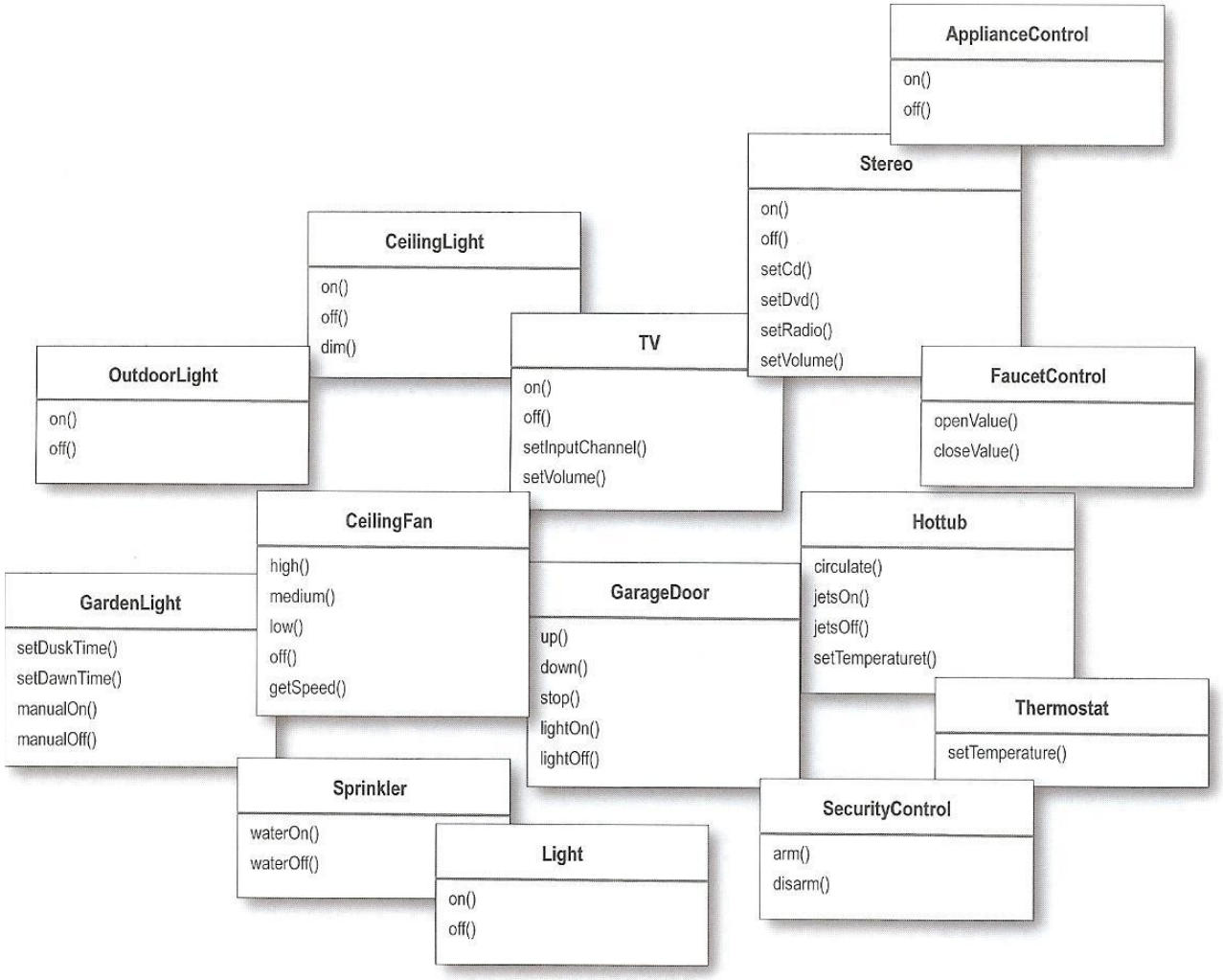
실행 결과


--- exec-maven-plugin:1.2.1:exec (
웨이트리스가 주문을 받습니다.
햄버거 주문서가 주방에 전달됩니다.
주방에서 햄버그를 만듭니다.

최종 목표: 홈오토메이션 리모컨



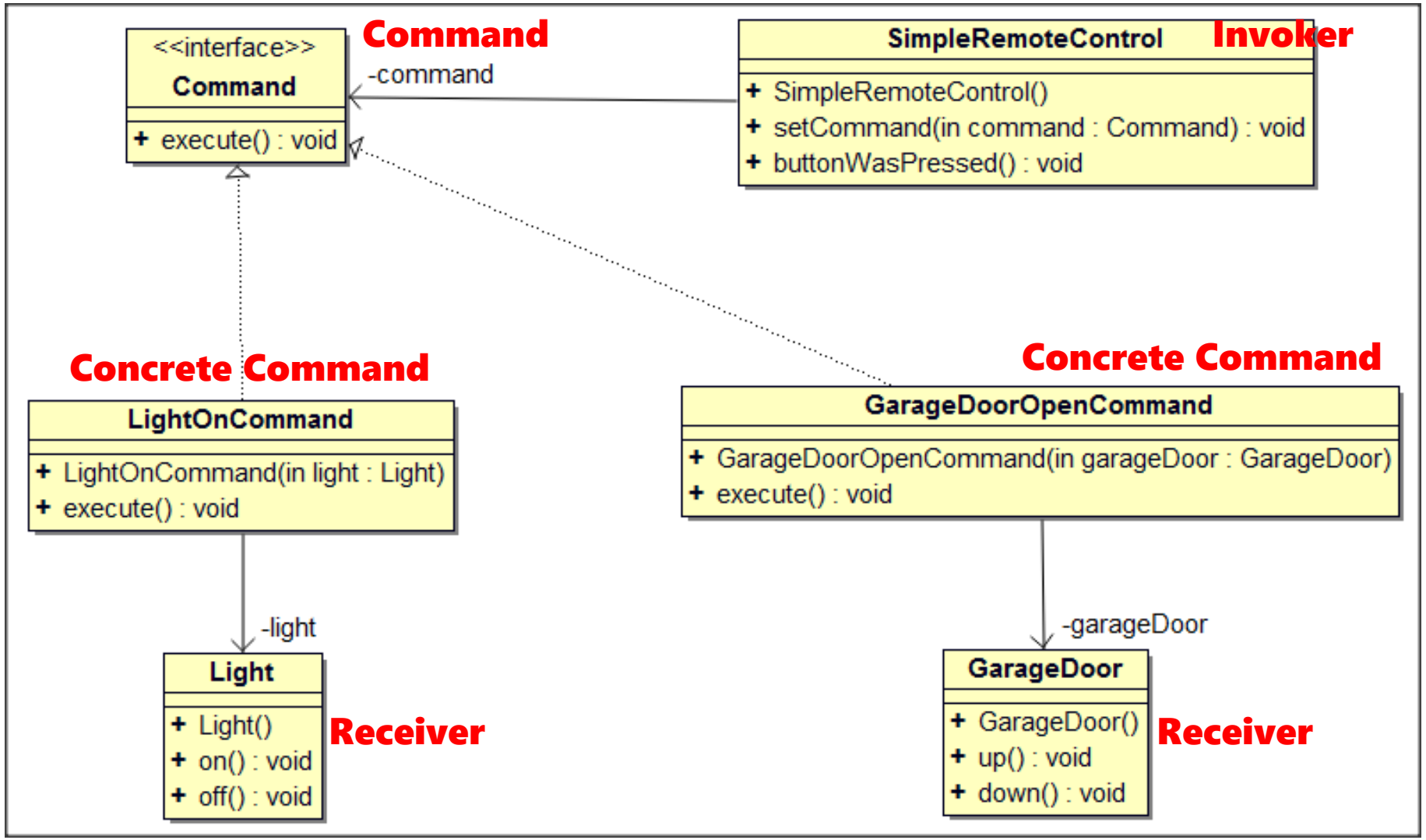
관련 클래스


ApplianceControl
on()
off()
Stereo
on()
off()
setCd()
setDvd()
setRadio()
setVolume()
CeilingLight
on()
off()
dim()
TV
on()
off()
setInputChannel()
setVolume()
OutdoorLight
on()
off()
FaucetControl
openValue()
closeValue()
CeilingFan
high()
medium()
low()
off()
getSpeed()
GardenLight
setDuskTime()
setDawnTime()
manualOn()
manualOff()
GarageDoor
up()
down()
stop()
lightOn()
lightOff()
Hottub
circulate()
jetsOn()
jetsOff()
setTemperaturet()
Thermostat
setTemperature()
Sprinkler
waterOn()
waterOff()
Light
on()
off()
SecurityControl
arm()
disarm()

실습: simple

- 전등과 차고 문을 **하나의 리모콘 버튼을 사용**하여 동작시킬 수 있는 리모콘을 설계하시오.
- **필요에 따라서** 리모콘의 버튼에 **전등을 동작시키는 명령**과 **차고 문을 동작시키는 명령을 설정**할 수 있어야 한다.

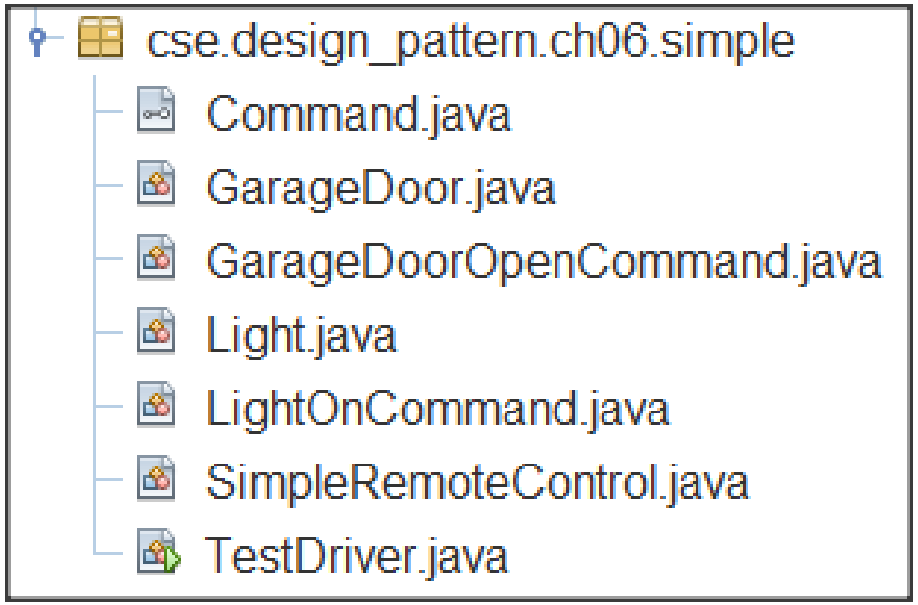
SimpleRemoteControl 클래스 다이어그램



Command Pattern에 따른 분류

- **Command (Command)**
  - declares an interface for executing an operation
- **ConcreteCommand (LightOnCommand, GarageDoorOpenCommand)**
  - defines a binding between a Receiver object and an action
  - implements Execute by invoking the corresponding operation(s) on Receiver
- **Client (RemoteControlTest)**
  - creates a ConcreteCommand object and sets its receiver
- **Invoker (SimpleRemoteControl)**
  - asks the command to carry out the request
- **Receiver (Light, GarageDoor)**
  - knows how to perform the operations associated with carrying out the request.

실습: simple


cse.design_pattern.ch06.simple
Command.java
GarageDoor.java
GarageDoorOpenCommand.java
Light.java
LightOnCommand.java
SimpleRemoteControl.java
TestDriver.java

Receiver 구현

- Light.java

```java
package cse.design_pattern.ch06.simple;

public class Light {

    public Light() {
    }

    public void on() {
        System.out.println("전등이 켜졌습니다.");
    }

    public void off() {
        System.out.println("전등이 꺼졌습니다.");
    }

}
```

GarageDoor.java

```java
package cse.design_pattern.ch06.simple;

public class GarageDoor {

    public GarageDoor() {
    }

    public void up() {
        System.out.println("차고 문이 열렸습니다..");
    }

    public void down() {
        System.out.println("차고 문이 닫혔습니다..");
    }

}
```

Command.java

```
1  package cse.design_pattern.ch06.simple;
2
3  public interface Command {
4
5      void execute();
6
7  }
```

LightOnCommand.java

```
1   package cse.design_pattern.ch06.simple;
2
3   public class LightOnCommand implements Command {
4
5       private Light light;
6
7       public LightOnCommand(Light light) {
8           this.light = light;
9       }
10
11      public void execute() {
12          light.on();
13      }
14
15  }
```

버튼을 누를 때마다 전등이 켜졌다 꺼졌다 하도록 수정해 봅시다.

GarageDoorOpenCommand.java

```
1   package cse.design_pattern.ch06.simple;
2
3   public class GarageDoorOpenCommand implements Command {
4
5     private GarageDoor garageDoor;
6
7     public GarageDoorOpenCommand(GarageDoor garageDoor) {
8       this.garageDoor = garageDoor;
9     }
10
11    public void execute() {
12      garageDoor.up();
13    }
14
15  }
```

버튼을 누를 때마다 차고 문이 열렸다 닫혔다 하도록 수정해 봅시다.

SimpleRemoteControl.java

```java
package cse.design_pattern.ch06.simple;

public class SimpleRemoteControl {

    private Command command;

    public SimpleRemoteControl() {
    }

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

    public void buttonWasPressed() {
        command.execute();
    }

}
```

TestDriver.java

```
6   package cse.design_pattern.ch06.simple;
7
8   /**...4 lines */
12  public class TestDriver {
13
14      /**...3 lines */
17      public static void main(String[] args) {
18          SimpleRemoteControl remote = new SimpleRemoteControl();
19
20          // Light 관련 명령 설정
21          Light light = new Light();
22          LightOnCommand lightOn = new LightOnCommand(light);
23          remote.setCommand(lightOn);
24          remote.buttonWasPressed();
25
26          // GarageDoor 관련 명령 설정
27          GarageDoor garageDoor = new GarageDoor();
28          GarageDoorOpenCommand garageOpen =
29                  new GarageDoorOpenCommand(garageDoor);
30          remote.setCommand(garageOpen);
31          remote.buttonWasPressed();
32      }
33
34  }
```

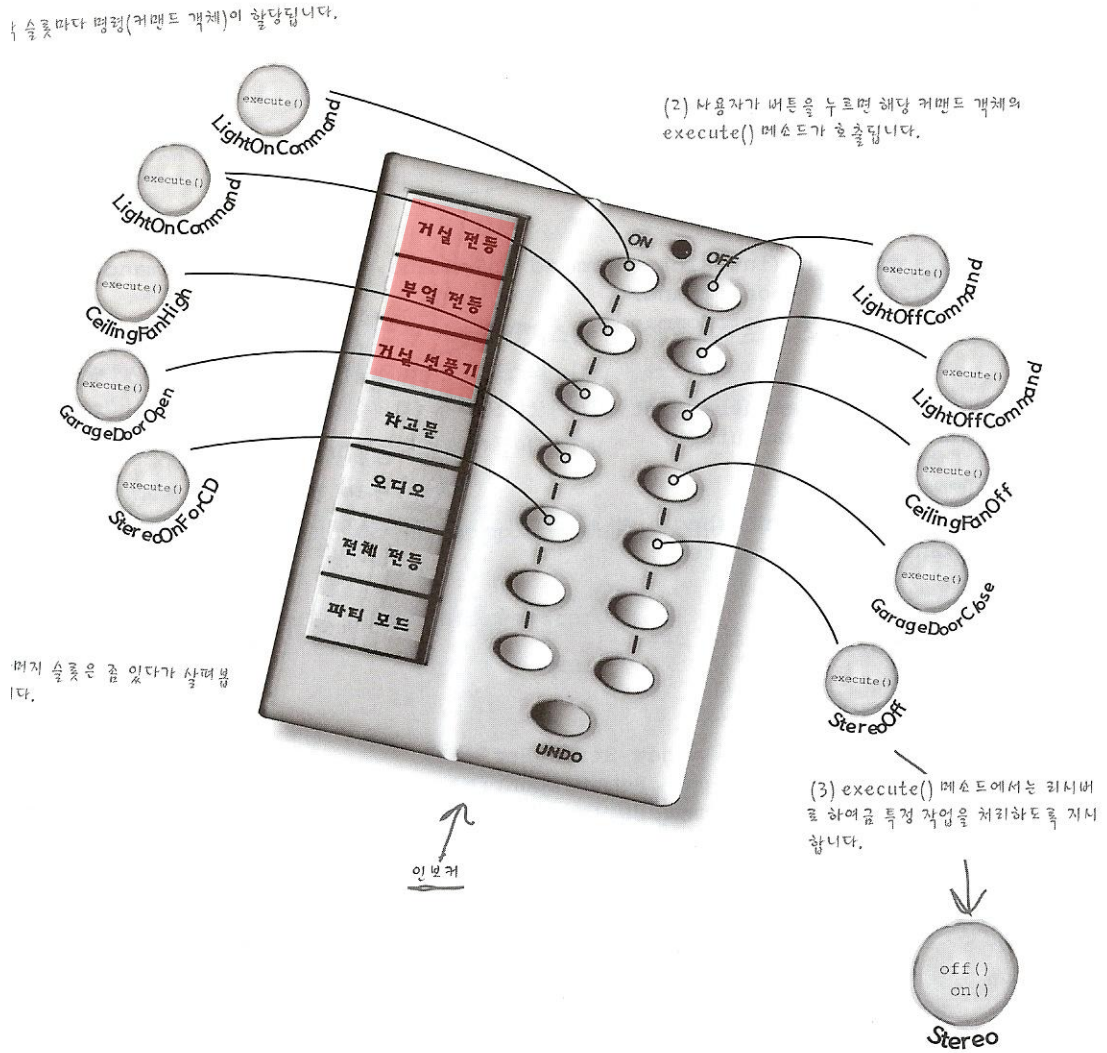
실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
전등이 켜졌습니다.
차고 문이 열렸습니다..
```

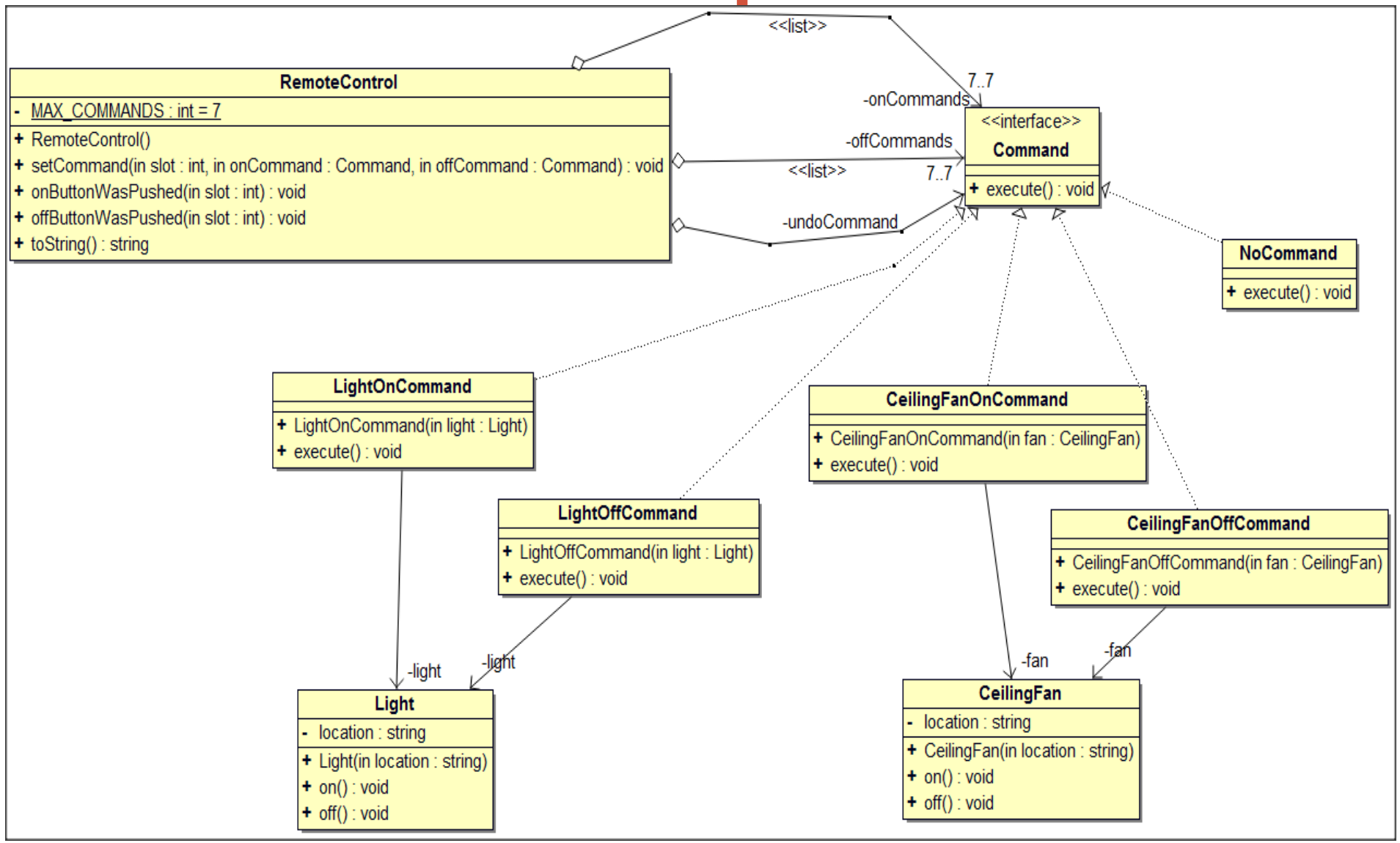
TODO: 토글 버튼 구현

- 리모콘 버튼에 하나의 명령이 설정된 상태에서 아래와 같이 동작하도록 프로그램을 변경하시오.
  - 전등이 켜진 경우 전등을 끈다
  - 전등이 꺼진 경우 전등을 켠다
  - 차고 문이 열린 경우 차고 문을 닫는다
  - 차고 문이 닫힌 경우 차고 문을 연다

슬롯에 명령 할당하기

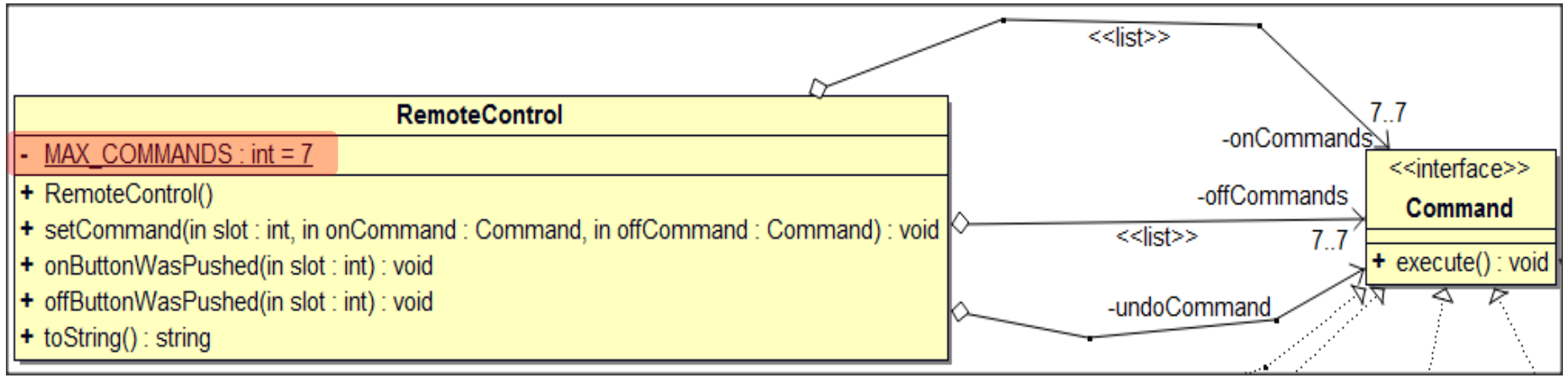

각 슬롯마다 명령(커맨드 객체)이 할당됩니다.
execute()
LightOnCommand
execute()
LightOnCommand
execute()
CeilingFanHigh
execute()
GarageDoorOpen
execute()
StereoOnForCD
거실 전등
부엌 전등
거실 선풍기
차고문
오디오
전체 전등
파티 모드
ON
OFF
UNDO
(2) 사용자가 버튼을 누르면 해당 커맨드 객체의 execute() 메소드가 호출됩니다.
execute()
LightOffCommand
execute()
LightOffCommand
execute()
CeilingFanOff
execute()
GarageDoorClose
execute()
StereoOff
(3) execute() 메소드에서는 리시버를 하여금 특정 작업을 처리하도록 지시합니다.
off()
on()
Stereo
인보커

클래스 다이어그램: multiple

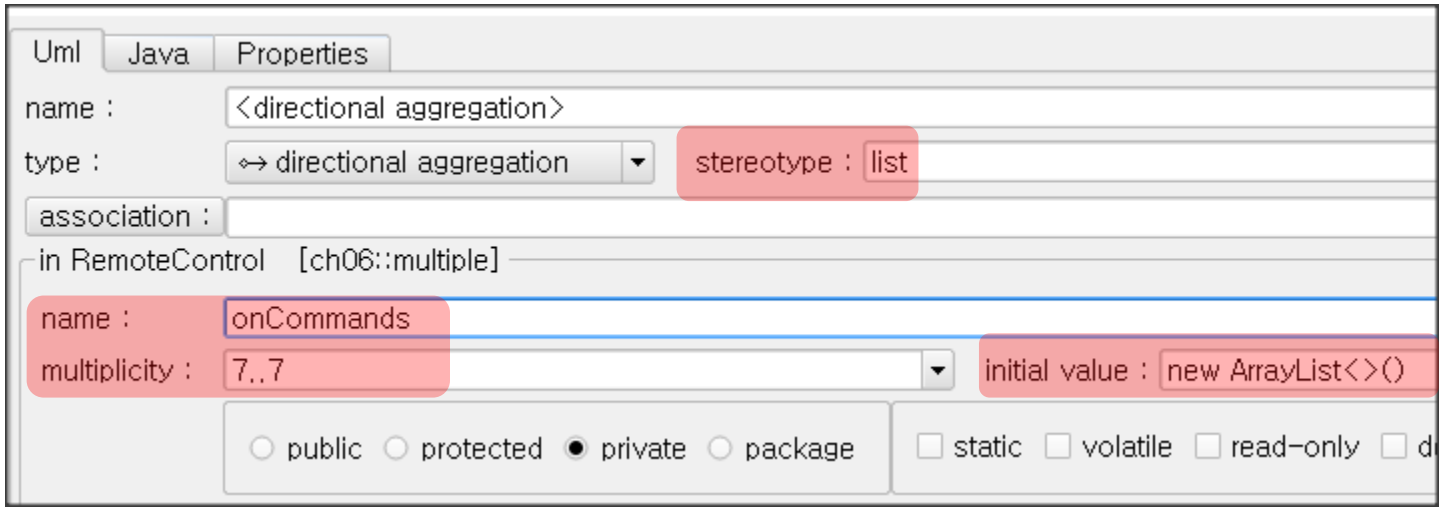


RemoteControl vs. Command

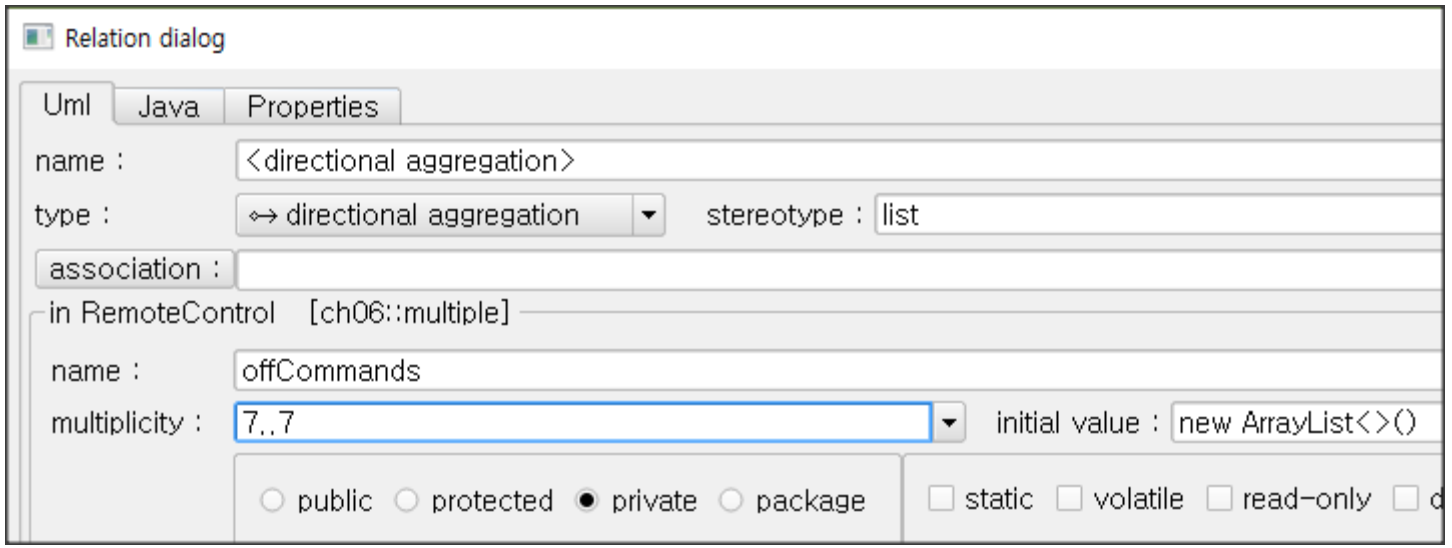
- Navigable aggregation 사용하여 연관 관계 표현



onCommands 집합 관계 설정



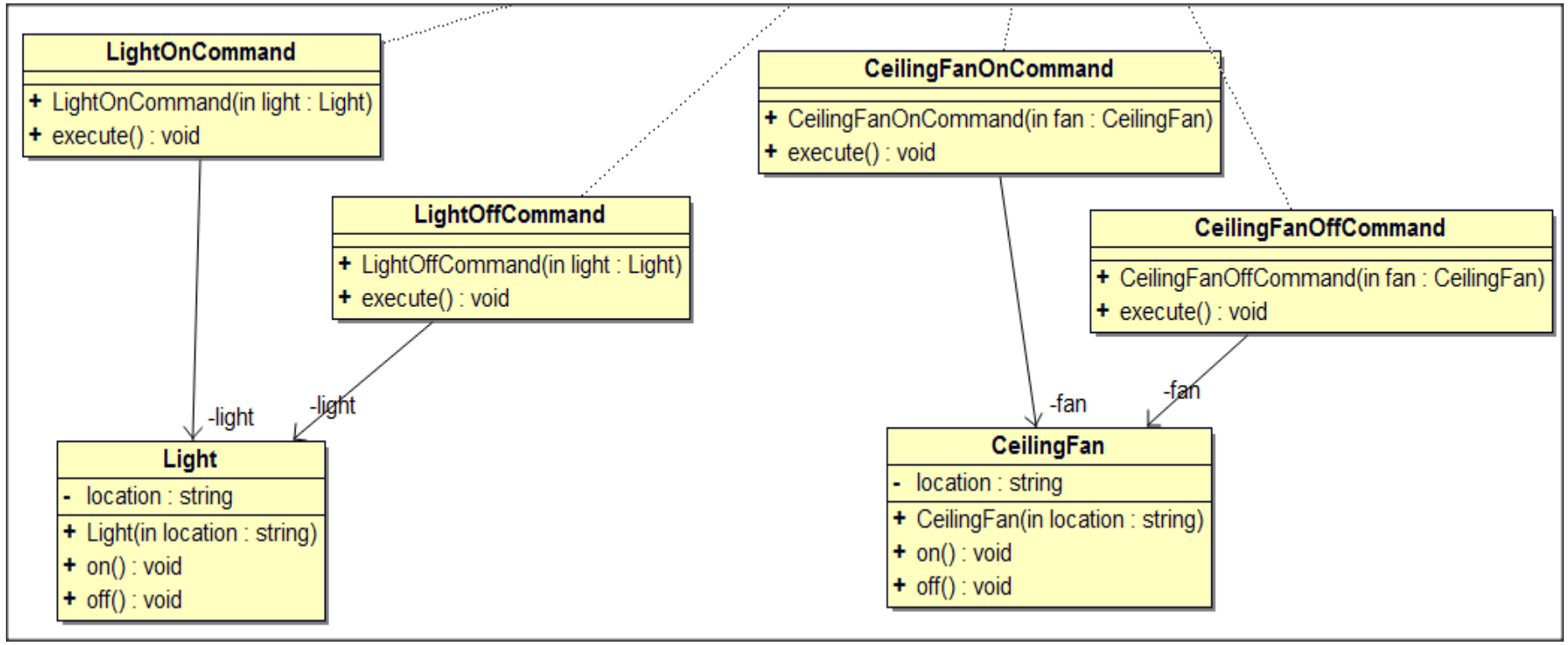
offCommands 집합 관계 설정



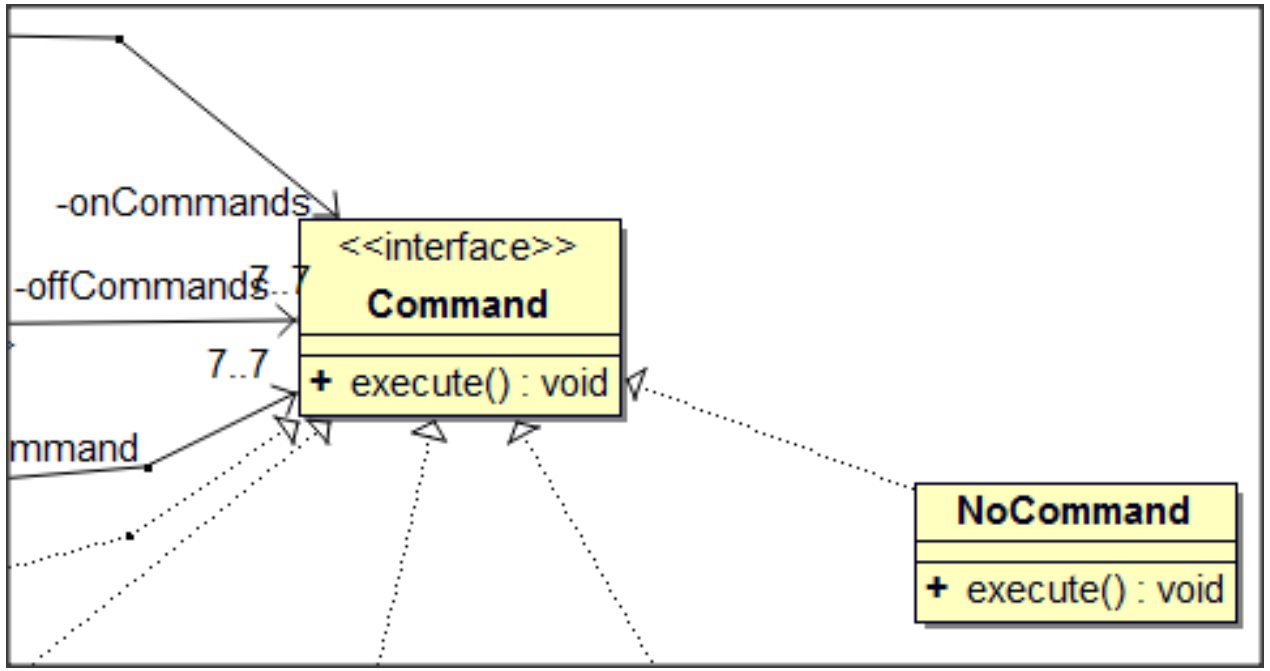
Result after substitution :

```
private List<Command> offCommands = new ArrayList<>();
```

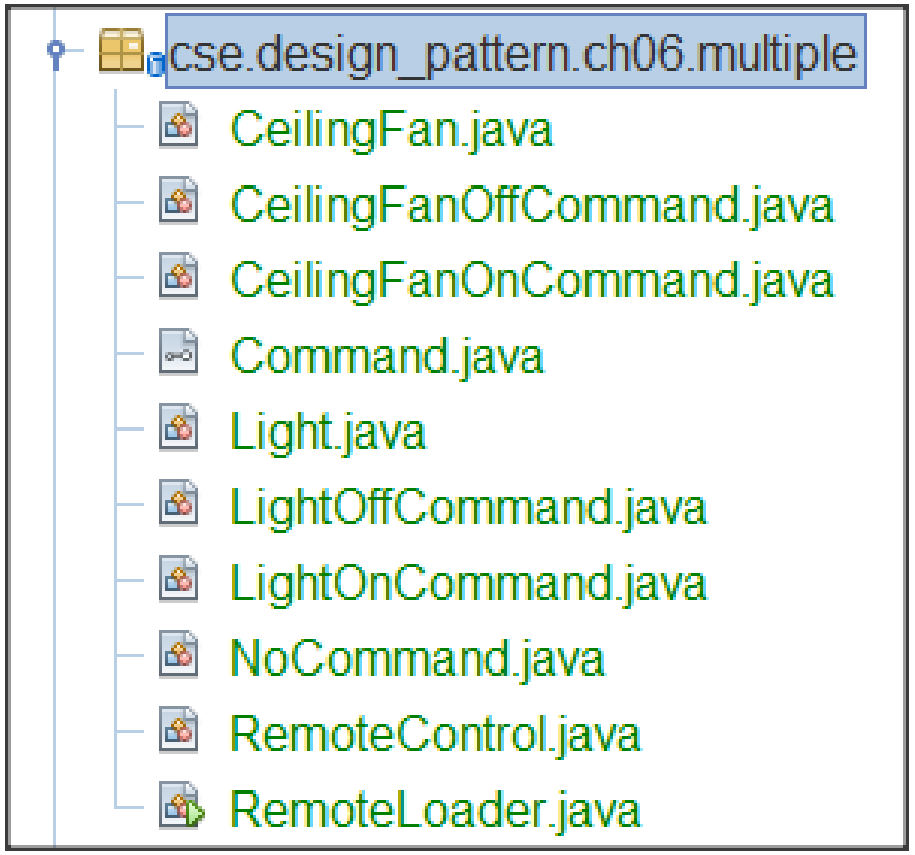
Light & CeilingFan 관련



Command & NoCommand



실습: multiple


cse.design_pattern.ch06.multiple
CeilingFan.java
CeilingFanOffCommand.java
CeilingFanOnCommand.java
Command.java
Light.java
LightOffCommand.java
LightOnCommand.java
NoCommand.java
RemoteControl.java
RemoteLoader.java

Light.java

```java
package cse.design_pattern.ch06.multiple;

public class Light {

    private String location;

    public Light(String location) {
        this.location = location;
    }

    public void on() {
        System.out.println(location + " 전등이 켜졌습니다.");
    }

    public void off() {
        System.out.println(location + " 전등이 꺼졌습니다.");
    }

}
```

CeilingFan.java

```java
package cse.design_pattern.ch06.multiple;

public class CeilingFan {

    private String location;

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

    public void on() {
        System.out.println(location + "  천장 선풍기가 켜졌습니다.");
    }

    public void off() {
        System.out.println(location + " 천장 선풍기가 꺼졌습니다.");
    }

}
```

Command.java & NoCommand.java

```java
package cse.design_pattern.ch06.multiple;

public interface Command {

    void execute();

}
```

```java
package cse.design_pattern.ch06.multiple;

public class NoCommand implements Command {

    public void execute() {
        System.out.println("아무 것도 실행되지 않습니다.");
    }

}
```

Light[On | Off]Command.java

```java
package cse.design_pattern.ch06.multiple;

public class LightOnCommand implements Command {

    private Light light;

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

    public void execute() {
        light.on();
    }

}
```

```java
package cse.design_pattern.ch06.multiple;

public class LightOffCommand implements Command {

    private Light light;

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

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

}
```

CeilingFan[On | Off]Command.java

```java
package cse.design_pattern.ch06.multiple;

public class CeilingFanOnCommand implements Command {

    private CeilingFan fan;

    public CeilingFanOnCommand(CeilingFan fan) {
        this.fan = fan;
    }

    public void execute() {
        fan.on();
    }

}
```

```java
package cse.design_pattern.ch06.multiple;

public class CeilingFanOffCommand implements Command {

    private CeilingFan fan;

    public CeilingFanOffCommand(CeilingFan fan) {
        this.fan = fan;
    }

    public void execute() {
        fan.off();
    }

}
```

RemoteControl.java

```
1   package cse.design_pattern.ch06.multiple;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   public class RemoteControl {
7
8       public static final  int MAX_COMMANDS = 7;
        private List<Command> onCommands = new ArrayList<>();
        private List<Command> offCommands = new ArrayList<>();
11      private Command undoCommand;
12
13      public RemoteControl() {
14          // 리모콘 버튼을 NoCommand 객체로 모두 초기화
15          Command noCommand = new NoCommand();
16          for (int i = 0; i < MAX_COMMANDS; i++) {
17              onCommands.add(i, noCommand);
18              offCommands.add(i, noCommand);
19          }
20      }
```

```
22      public void setCommand(int slot, Command onCommand, Command offCommand) {
23          onCommands.set(slot, onCommand);
24          offCommands.set(slot, offCommand);
25      }
26
27      public void onButtonWasPushed(int slot) {
28          onCommands.get(slot).execute();
29      }
30
31      public void offButtonWasPushed(int slot) {
32          offCommands.get(slot).execute();
33      }
34
        public String toString() {
36          StringBuilder buf = new StringBuilder();
37
38          buf.append("₩n------ Remote Control -------₩n");
39          for (int i = 0; i < onCommands.size(); i++) {
40              buf.append("[slot ").append(i).append("] ");
41              buf.append(onCommands.get(i).getClass().getName()).append("    ");
42              buf.append(offCommands.get(i).getClass().getName()).append("₩n");
43          }
44          return buf.toString();
45      }
46
47  }
```

RemoteLoader.java

```
6   package cse.design_pattern.ch06.multiple;
7
8   /**...4 lines */
12  public class RemoteLoader {
13
14      public static void main(String[] args) {
15          RemoteControl remote = new RemoteControl();
16
17          // 거실 전용 설정: 슬롯 0
18          Light livingRoomLight = new Light("거실");
19          LightOnCommand livingRoomLightOn = new LightOnCommand(livingRoomLight);
20          LightOffCommand livingRoomLightOff = new LightOffCommand(livingRoomLight);
21          remote.setCommand(0, livingRoomLightOn, livingRoomLightOff);
22
23          // 부엌 전등 설정: 슬롯 1
24          Light kitchenLight = new Light("주방");
25          LightOnCommand kitchenLightOn = new LightOnCommand(kitchenLight);
26          LightOffCommand kitchenLightOff = new LightOffCommand(kitchenLight);
27          remote.setCommand(1, kitchenLightOn, kitchenLightOff);
```

```
29          // 거실 선풍기 설정: 슬롯 2
30          CeilingFan fan = new CeilingFan("거실");
31          CeilingFanOnCommand fanOn = new CeilingFanOnCommand(fan);
32          CeilingFanOffCommand fanOff = new CeilingFanOffCommand(fan);
33          remote.setCommand(2, fanOn, fanOff);
34
35          System.out.println(remote);
36
37          for (int i=0; i<RemoteControl.MAX_COMMANDS; i++) {
38              System.out.println("[슬롯 " + i + "]");
39              remote.onButtonWasPushed(i);
40              remote.offButtonWasPushed(i);
41          }
42      }
43
44  }
```

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---

------ Remote Control -------
[slot 0] cse.design_pattern.ch06.multiple.LightOnCommand    cse.design_pattern.ch06.multiple.LightOffCommand
[slot 1] cse.design_pattern.ch06.multiple.LightOnCommand    cse.design_pattern.ch06.multiple.LightOffCommand
[slot 2] cse.design_pattern.ch06.multiple.CeilingFanOnCommand    cse.design_pattern.ch06.multiple.CeilingFanOffCommand
[slot 3] cse.design_pattern.ch06.multiple.NoCommand    cse.design_pattern.ch06.multiple.NoCommand
[slot 4] cse.design_pattern.ch06.multiple.NoCommand    cse.design_pattern.ch06.multiple.NoCommand
[slot 5] cse.design_pattern.ch06.multiple.NoCommand    cse.design_pattern.ch06.multiple.NoCommand
[slot 6] cse.design_pattern.ch06.multiple.NoCommand    cse.design_pattern.ch06.multiple.NoCommand

[슬롯 0]
거실 전등이 켜졌습니다.
거실 전등이 꺼졌습니다.
[슬롯 1]
주방 전등이 켜졌습니다.
주방 전등이 꺼졌습니다.
[슬롯 2]
거실  천장 선풍기가 켜졌습니다.
거실 천장 선풍기가 꺼졌습니다.
[슬롯 3]
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
[슬롯 4]
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
[슬롯 5]
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
[슬롯 6]
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
```

추가해 봅시다. (complete)

- UNDO 버튼
  - Command 인터페이스에 undo() 메소드 추가

- 전체 전등 (슬롯 5) & 파티 모드 (슬롯 6)
  - MacroCommand 클래스 정의하여 슬롯 5와 6에 사용

Command.java 수정

```
1  package cse.design_pattern.ch06.complete;
2
3  /**
4   * undo 기능 구현하기 위하여 undo() 메소드 추가
5   *
6   * @author Prof.Jong Min Lee (my_account AT
7   */
8  public interface Command {
9
10   void execute();
11   void undo();
12
13 }
```

LightOnCommand.java 수정

```
1   package cse.design_pattern.ch06.complete;
2
3   public class LightOnCommand implements Command {
4
5       private Light light;
6
7       public LightOnCommand(Light light) {
8           this.light = light;
9       }
10
11      public void execute() {
12          light.on();
13      }
14
15      /** undo 기능 구현 ...3 lines */
16      public void undo() {
19          light.off();
20      }
21
22  }
```

LightOffCommand.java 수정

```
1   package cse.design_pattern.ch06.complete;
2
3   public class LightOffCommand implements Command {
4
6       private Light light;
7
7       public LightOffCommand(Light light) {
8           this.light = light;
9       }
10
12      public void execute() {
12          light.off();
13      }
14
15      /** undo 기능 구현 ...3 lines */
19      public void undo() {
19          light.on();
20      }
21
22  }
```

CeilingFan[On|Off]Command.java

```java
package cse.design_pattern.ch06.complete;

public class CeilingFanOnCommand implements Command {

    private CeilingFan fan;

    public CeilingFanOnCommand(CeilingFan fan) {
        this.fan = fan;
    }

    public void execute() {
        fan.on();
    }

    /** undo 기능 구현 ...3 lines */
    public void undo() {
        fan.off();
    }

}
```

```java
package cse.design_pattern.ch06.complete;

public class CeilingFanOffCommand implements Command {

    private CeilingFan fan;

    public CeilingFanOffCommand(CeilingFan fan) {
        this.fan = fan;
    }

    public void execute() {
        fan.off();
    }

    /** undo 기능 구현 ...3 lines */
    public void undo() {
        fan.on();
    }

}
```

MacroCommand.java

```
6   package cse.design_pattern.ch06.complete;
7
8   /** 슬롯 5의 전체 전등 모드와 슬롯 6의 파티 모드를 구현하기 위한 매크로 명령어 ...5 lines */
13  public class MacroCommand implements Command {
14      private Command[] commands;
15
16      public MacroCommand(Command[] commands) {
17          this.commands = commands;
18      }
19
          public void execute() {
              for (int i=0; i<commands.length; i++) {
22                commands[i].execute();
23            }
24      }
25
          public void undo() {
              for (int i=0; i<commands.length; i++) {
28                commands[i].undo();
29            }
30      }
31
32  }
```

RemoteControl.java 수정

```
1   package cse.design_pattern.ch06.complete;
2
3   import java.util.ArrayList;
4   import java.util.List;
5
6   /** multiple 패키지의 기능에 UNDO 버튼, 전체 전등(슬롯 5)을 추가적으로 구현함 ...5 lines */
11  public class RemoteControl {
12
13      public static final  int MAX_COMMANDS = 7;
        private List<Command> onCommands = new ArrayList<>();
        private List<Command> offCommands = new ArrayList<>();
16      private Command undoCommand;
17
18      /** undo 기능 구현 위하여 undoCommand 초기화를 추가함 ...3 lines */
21      public RemoteControl() {
22          // 리모콘 버튼을 NoCommand 객체로 모두 초기화
23          Command noCommand = new NoCommand();
24          for (int i = 0; i < MAX_COMMANDS; i++) {
25              onCommands.add(i, noCommand);
26              offCommands.add(i, noCommand);
27          }
28
29          undoCommand = noCommand;
30      }
31
32      public void setCommand(int slot, Command onCommand, Command offCommand) {
33          onCommands.set(slot, onCommand);
34          offCommands.set(slot, offCommand);
35      }
```

```
37      public void onButtonWasPushed(int slot) {
38          onCommands.get(slot).execute();
39          undoCommand = onCommands.get(slot);
40      }
41
42      public void offButtonWasPushed(int slot) {
43          offCommands.get(slot).execute();
44          undoCommand = offCommands.get(slot);
45      }
46
47      /** undo 기능 구현 ...3 lines */
50      public void undoButtonWasPushed() {
51          undoCommand.undo();
52      }
53
        public String toString() {
55          StringBuilder buf = new StringBuilder();
56
57          buf.append("₩n------ Remote Control -------₩n");
58          for (int i = 0; i < onCommands.size(); i++) {
59              buf.append("[slot ").append(i).append("] ");
60              buf.append(onCommands.get(i).getClass().getName()).append("    ");
61              buf.append(offCommands.get(i).getClass().getName()).append("₩n");
62          }
63          return buf.toString();
64      }
65
66  }
```

RemoteLoader.java 수정

```
6   package cse.design_pattern.ch06.complete;
7
8   /**...4 lines */
12  public class RemoteLoader {
13
14      public static void main(String[] args) {
15          RemoteControl remote = new RemoteControl();
16
17          // 거실 전용 설정: 슬롯 0
18          Light livingRoomLight = new Light("거실");
19          LightOnCommand livingRoomLightOn = new LightOnCommand(livingRoomLight);
20          LightOffCommand livingRoomLightOff = new LightOffCommand(livingRoomLight);
21          remote.setCommand(0, livingRoomLightOn, livingRoomLightOff);
22
23          // 부엌 전등 설정: 슬롯 1
24          Light kitchenLight = new Light("주방");
25          LightOnCommand kitchenLightOn = new LightOnCommand(kitchenLight);
26          LightOffCommand kitchenLightOff = new LightOffCommand(kitchenLight);
27          remote.setCommand(1, kitchenLightOn, kitchenLightOff);
28
29          // 거실 선풍기 설정: 슬롯 2
30          CeilingFan fan = new CeilingFan("거실");
31          CeilingFanOnCommand fanOn = new CeilingFanOnCommand(fan);
32          CeilingFanOffCommand fanOff = new CeilingFanOffCommand(fan);
33          remote.setCommand(2, fanOn, fanOff);
```

```
35          // 전체 전등: 슬롯 5
36          Command[] temp1 = {livingRoomLightOn, kitchenLightOn};
37          Command[] temp2 = {livingRoomLightOff, kitchenLightOff};
38          MacroCommand allLightsOn = new MacroCommand(temp1);
39          MacroCommand allLightsOff = new MacroCommand(temp2);
40          remote.setCommand(5, allLightsOn, allLightsOff);
41
42          System.out.println(remote);
43
44          for (int i=0; i<RemoteControl.MAX_COMMANDS; i++) {
45              System.out.println("[슬롯 " + i + "]");
46              remote.onButtonWasPushed(i);
47              remote.offButtonWasPushed(i);
48              remote.undoButtonWasPushed();
49          }
50      }
51
52  }
```

실행 결과

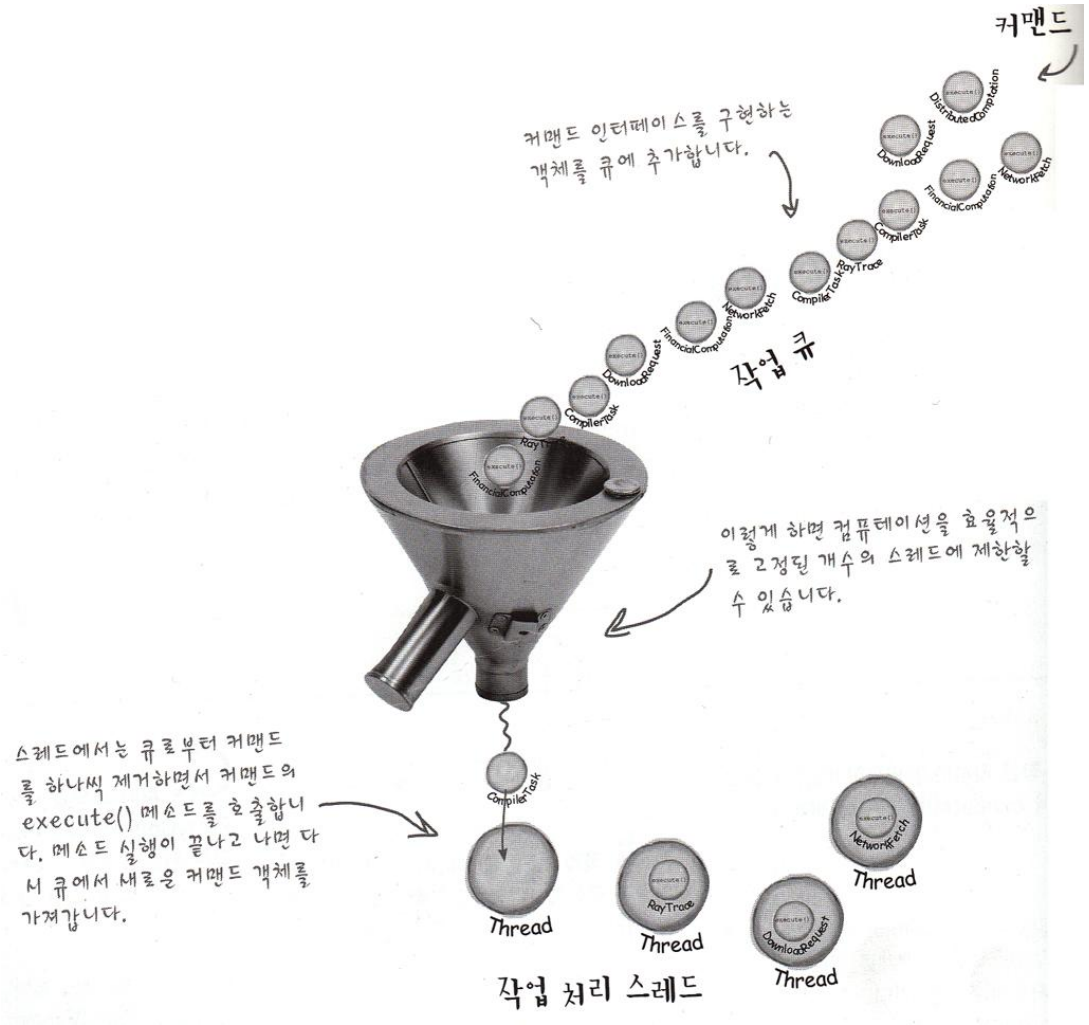
```
------ Remote Control -------
[slot 0] cse.design_pattern.ch06.complete.LightOnCommand    cse.design_pattern.ch06.complete.LightOffCommand
[slot 1] cse.design_pattern.ch06.complete.LightOnCommand    cse.design_pattern.ch06.complete.LightOffCommand
[slot 2] cse.design_pattern.ch06.complete.CeilingFanOnCommand    cse.design_pattern.ch06.complete.CeilingFanOffCommand
[slot 3] cse.design_pattern.ch06.complete.NoCommand    cse.design_pattern.ch06.complete.NoCommand
[slot 4] cse.design_pattern.ch06.complete.NoCommand    cse.design_pattern.ch06.complete.NoCommand
[slot 5] cse.design_pattern.ch06.complete.MacroCommand    cse.design_pattern.ch06.complete.MacroCommand
[slot 6] cse.design_pattern.ch06.complete.NoCommand    cse.design_pattern.ch06.complete.NoCommand

[슬롯 0]
거실 전등이 켜졌습니다.
거실 전등이 꺼졌습니다.
거실 전등이 켜졌습니다.
```

중략…

```
[슬롯 4]
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
[슬롯 5]
거실 전등이 켜졌습니다.
주방 전등이 켜졌습니다.
거실 전등이 꺼졌습니다.
주방 전등이 꺼졌습니다.
거실 전등이 켜졌습니다.
주방 전등이 켜졌습니다.
[슬롯 6]
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
아무 것도 실행되지 않습니다.
```

활용 1: Queueing Request

- Receiver와 일련의 액션을 묶어 Command 객체 형태로 작업 큐에 넣음.
- 작업 큐의 반대편에서는 몇 개의 스레드가 하나씩 Command 객체를 꺼내어 실행
- 스레드 개수에 따라서 작업 제어 가능



활용 2: Logging Requests

- Command 객체를 파일 형태로 저장
- 오류 발생 시 다시 복구하여 Command 객체를 순서대로 실행