REPORT

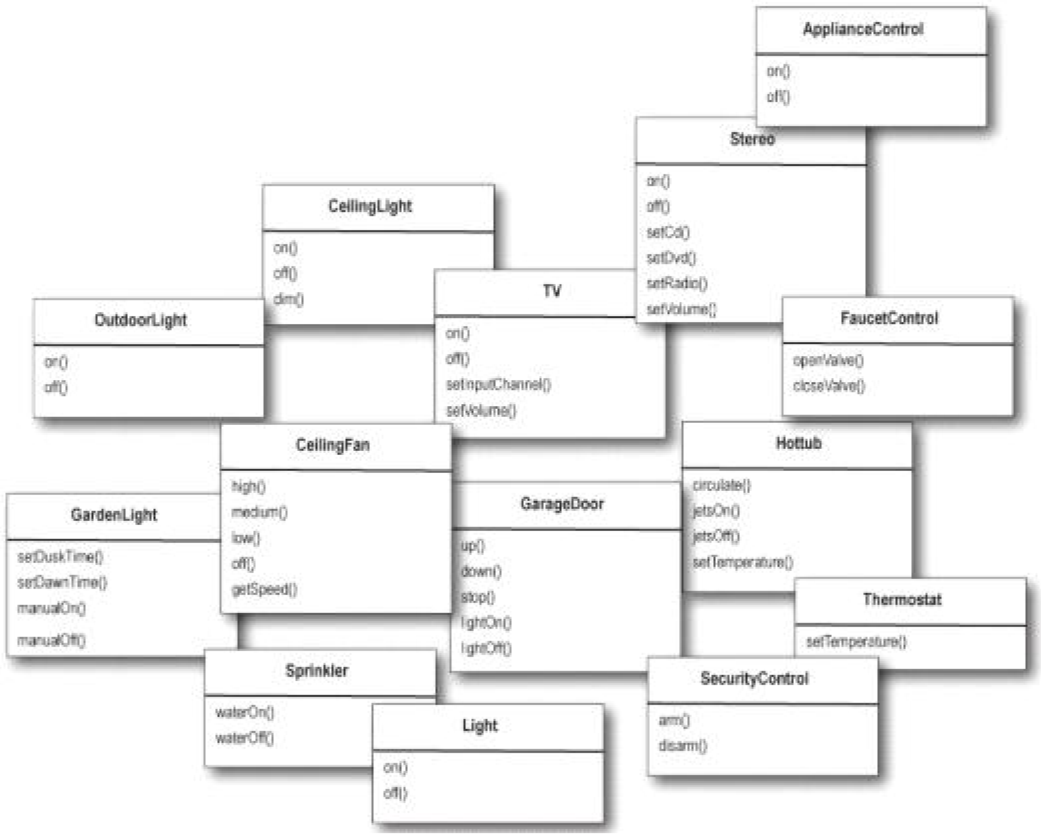
과 목 명	: Design Pattern
담담교수	: 박제호 교수님
소 속	: 소프트웨어학과
학 번	: 32151671
이 름	: 박민혁

단국대학교
Dankook University

Chapter 6 : Command Pattern

개요

- 내가 하는 동작을 encapsulation 시킨다.
- 리모컨 버튼의 기능을 어떻게 할 것인가?

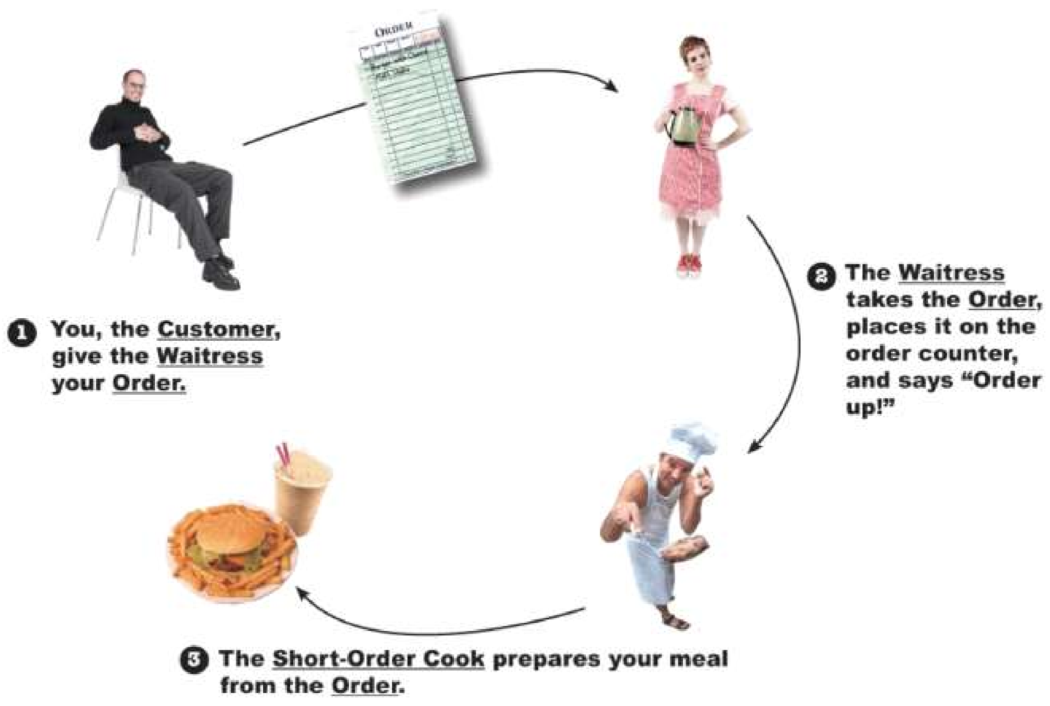


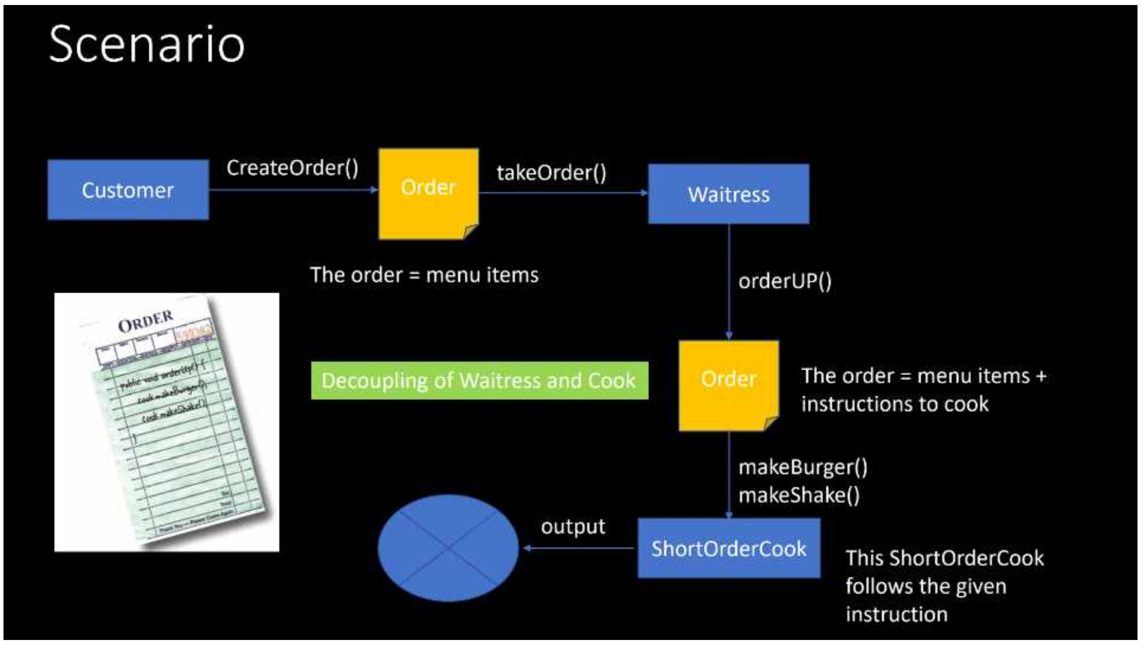
- 표준화 할 방법이 없다.

Requester -> remote control

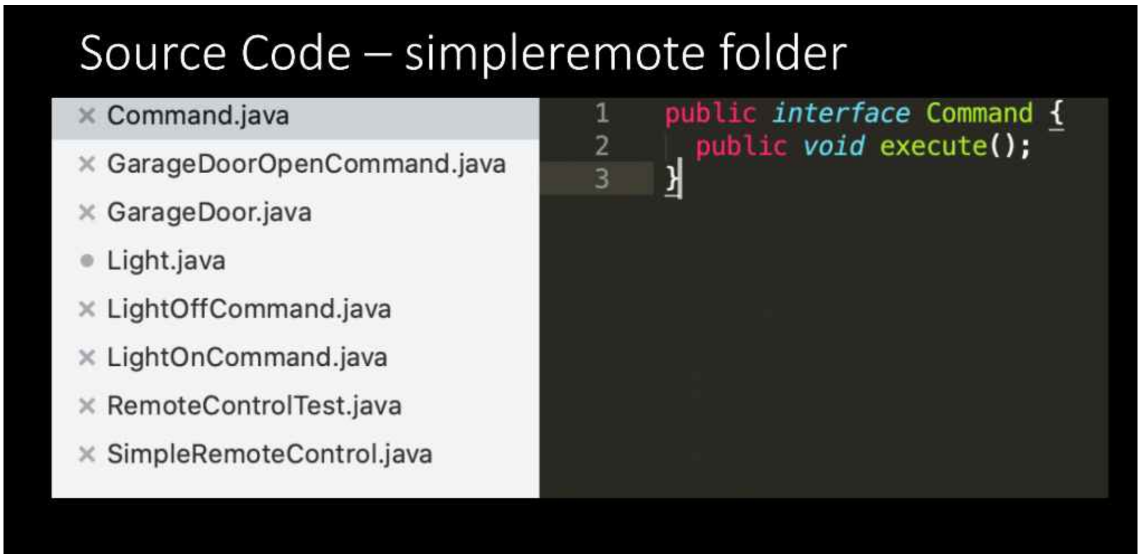
Object -> an instance of one of your vendor classes

책에서의 Example


ORDER
1 You, the Customer, give the Waitress your Order.
2 The Waitress takes the Order, places it on the order counter, and says "Order up!"
3 The Short-Order Cook prepares your meal from the Order.



- Order ; Waitress 와 ShortOrderCook 잘라버림
- Passing이 가능하다.
- Encapsulation
- Waitress의 내용을 알 필요가 없다.

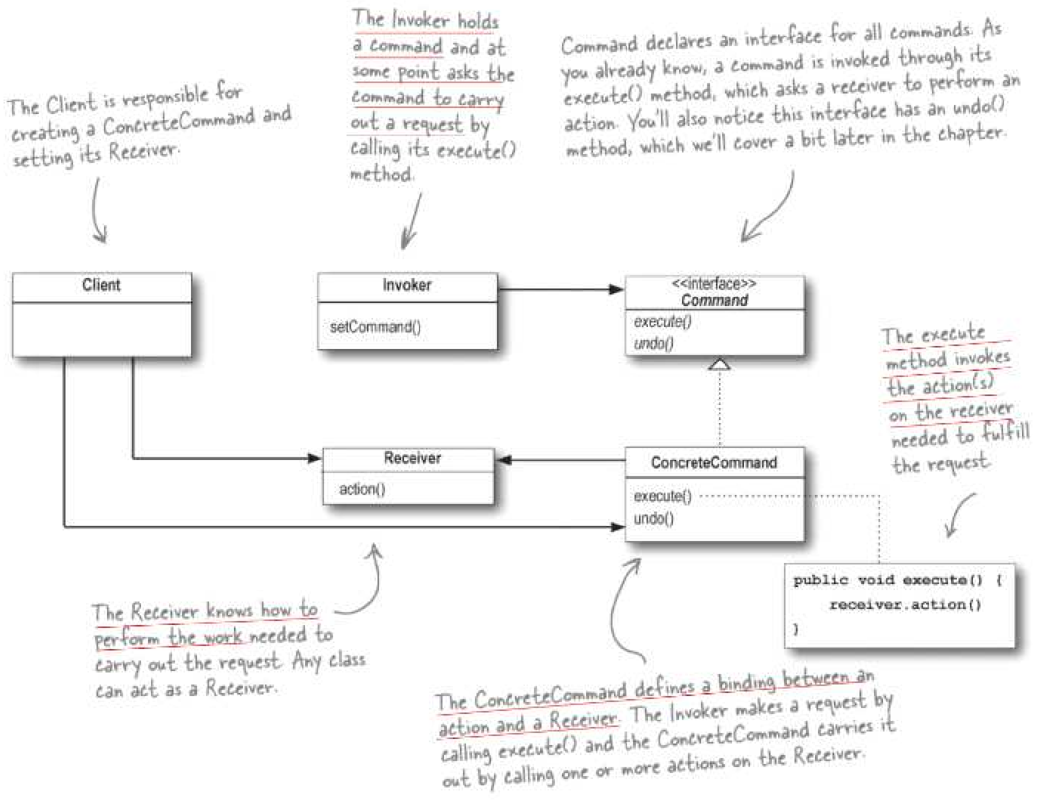


Source Code – simpleremote folder

- Command.java
- GarageDoorOpenCommand.java
- GarageDoor.java
- Light.java
- LightOffCommand.java
- LightOnCommand.java
- RemoteControlTest.java
- SimpleRemoteControl.java

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

- Command -> target이 누군지 알고 execute(실제 행동)를 한다.



The Client is responsible for creating a ConcreteCommand and setting its Receiver.

The Invoker holds a command and at some point asks the command to carry out a request by calling its execute() method.

Command declares an interface for all commands. As you already know, a command is invoked through its execute() method, which asks a receiver to perform an action. You'll also notice this interface has an undo() method, which we'll cover a bit later in the chapter.

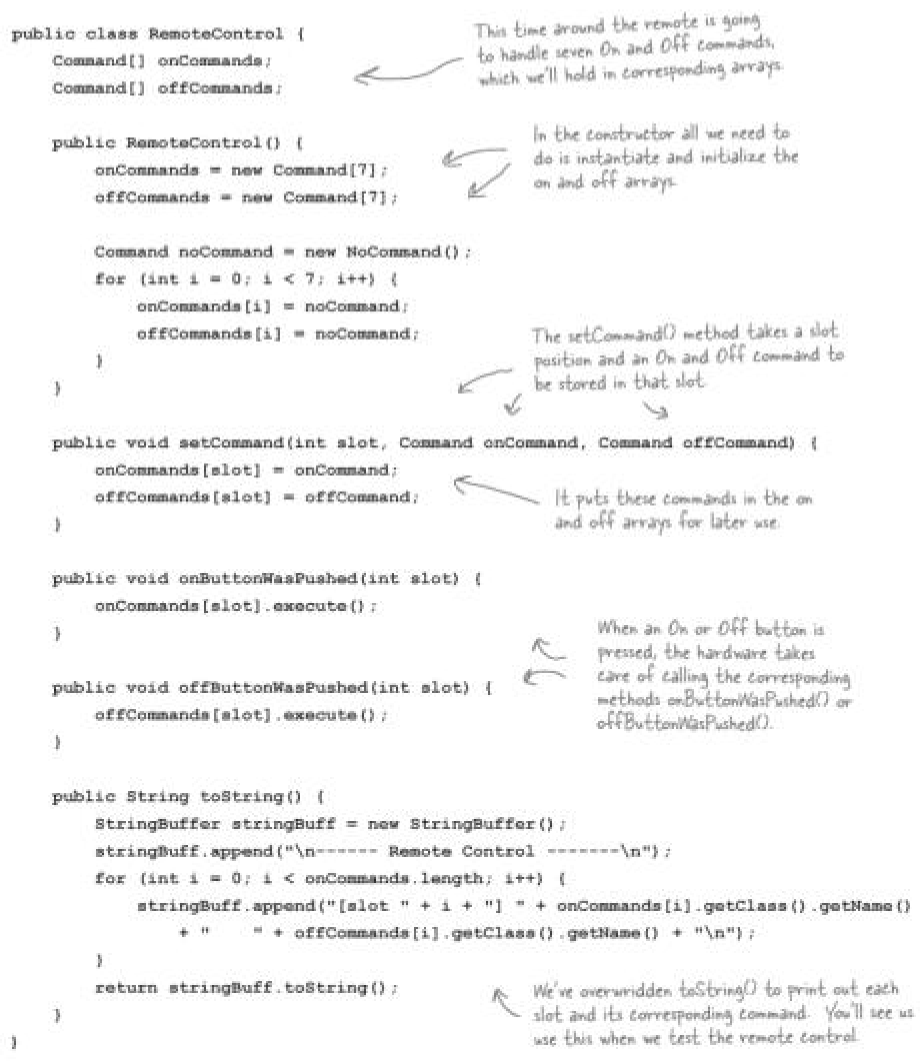
The execute method invokes the action(s) on the receiver needed to fulfill the request.

```java
public void execute() {
    receiver.action()
}
```

The Receiver knows how to perform the work needed to carry out the request. Any class can act as a Receiver.

The ConcreteCommand defines a binding between an action and a Receiver. The Invoker makes a request by calling execute() and the ConcreteCommand carries it out by calling one or more actions on the Receiver.

- Client -> 어떤 기기에 동작을 시킬 것인지 알고 있음
- Invoker -> 어떠어떠한 command가 있다라고 알고 있음



```
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;
    }

    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();
    }
}
```

This time around the remote is going to handle seven On and Off commands, which we'll hold in corresponding arrays.

In the constructor all we need to do is instantiate and initialize the on and off arrays.

The setCommand() method takes a slot position and an On and Off command to be stored in that slot.

It puts these commands in the on and off arrays for later use.

When an On or Off button is pressed, the hardware takes care of calling the corresponding methods onButtonWasPushed() or offButtonWasPushed().

We've overwridden toString() to print out each slot and its corresponding command. You'll see us use this when we test the remote control.

- 왼쪽 오른쪽 7개씩 배열 생성
- noCommand -> 아무 동작도 안하는 것
- setCommand -> 동작을 지시
- On / Off 버튼을 눌렀을 때 몇 번 버튼이 눌렀는지 알려줘야 함
- toString -> 객체의 상태의 출력

- High로 가기 전에 기억을 해서 undo 기능 수행