**소프트웨어 설계 패턴 (실습 5)**

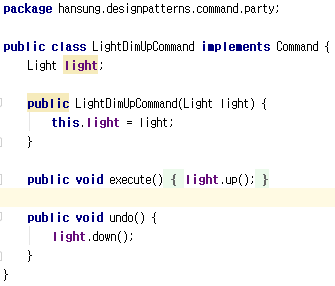
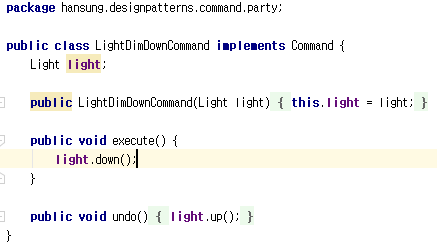
1494067 이찬희

이번 실습에서는 리모컨에 파티 모드를 구현한 예제 프로젝트인

<https://github.com/kwanulee/PatternExample/tree/master/command/party> 를 바탕으로 진행합니다.

1. 이 예제에서는 Light 클래스의 on(), off() 메소드를 호출하는 두 개의 커맨드 클래스 (LightOnCommand, LightOffCommand)가 포함되어 있습니다. 하지만, Light 클래스의 dim() 메소드를 호출하는 커맨트 클래스는 없습니다. dim() 메소드는 조명의 밝기 수준을 설정할 수 있는 메소드인데, 이를 사용하여 조명의 밝기를 한단계(예, 10%) 낮출수 있는 LightDimDownCommand와 밝기를 한단계(예, 10%) 높일 수 있는LightDimUpCommand클래스를 정의해 보세요. 그리고, 아래와 같이 동작하는 지를 확인해 보세요

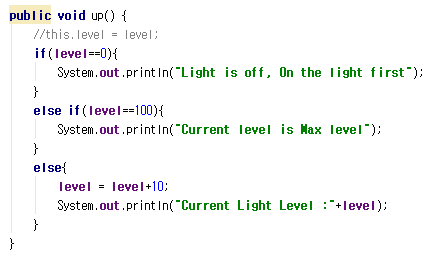
LightDimUpCommand, LightDimDownConnamd클래스를 생성하고 up()과 down()메소드를 정의해 줌 밝기를 올리는 게 up() 메소드이고 밝기를 내리는 게 down()메소드이다.

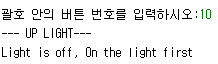
* + 두 커맨드는 Light가 켜진 상태에서만 동작해야 합니다. 만약 Light이 켜지지 않은 상태라면, 에러 메시지를 출력해야 합니다.
  + LightDimDownCommand의 수행에 의해서 밝기를 한단계 낮추었을 때, 밝기수준이 0가 된다면, 조명이 꺼진것입니다.
  + LightDimUpCommand의 수행에 의해서 높일 수 있는 밝기 수준은 최대 100%를 초과할 수 없습니다.
    - 위 3개 조건은 사진 1,2,3,4에 나와있습니다.
  + 두 커맨드 클래스의 undo메소드를 구현하여 이전 밝기 상태로 되돌아 갈 수 있어야 합니다.

->undo는 사진 5,6에 있습니다.

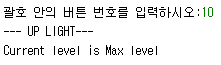
밑의 사진은 Light 클래스의 up() 메소드이다. Light가 0일 경우 라이트를 먼저 키라는 메시지를 출력하도록 하였고, 100 이상으로 넘어갈 수 없도록 하였다. 둘 다 아니면 10이 증가하도록 하였다.



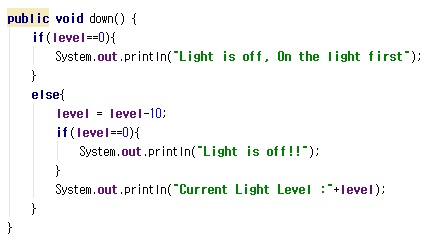
1. 라이트를 키지 않고 up()메소드 실행 시 나오는 결과



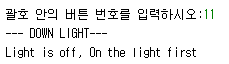
1. 라이트 밝기가 100이 되었을 때 넘어가지 않는 화면



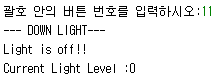
다음으로 down()메소드이다. 밝기가 0인 경우 라이트를 먼저 키라는 메시지를 출력하고 그게 아니라면 10씩 감소되게 구현하였다. 그리고 밝기가 0이 되면 Light is off와 같이 라이트가 꺼졌다는 메시지를 출력하도록 구현하였다.



1. 라이트를 키지 않고 down() 메소드 실행 결과

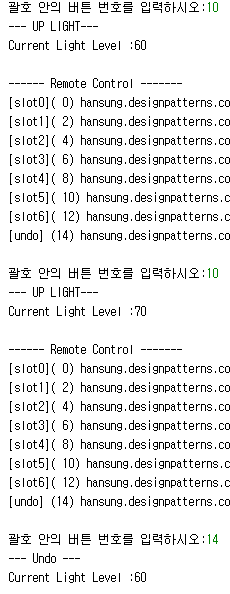


1. 라이트가 0이 되면 꺼지는 것을 보여주는 화면



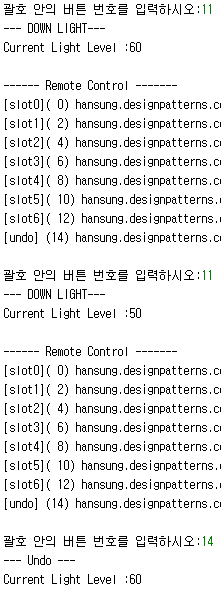
1. Up()메소드 실행 후 undo를 하는 화면

(밝기 60->70->60)



1. Down() 메소드 실행 후 undo를 하는 화면

(밝기 60->50->60)



1. 1번의 실습에서 추가된 커맨드 클래스로 인해 기존 코드의 어떤 부분을 수정하였는지를 분석해 보고, 커맨드 패턴의 장점 및 단점을 논의해 보세요.

->>추가한 것들

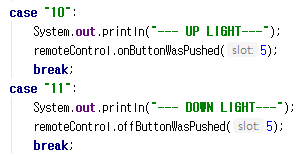
추가한 점은 LightDimUpCommand 클래스, LightDimDownCommand 클래스 2개를 추가하였고, 위 2개의 클래스를 추가함으로 인해 Light 클래스 안에 if문을 사용해 불이 꺼져 있을 때, 불의 밝기가 최대일 때, 밝기를 올리고 내릴 때를 구분하였다. Light 클래스 안의 Up()메소드와 down()메소드는 위 1번 문제에 사진이 있습니다.

그리고 RemoteLoader 클래스에



위 사진과 같이 LightDimUpCommand와 LightDimDownCommand 클래스 객체인 lightDimUp, lightDimDown 객체를 생성하였다.

그리고 switch case문에



다음과 같이 10,11번에 버튼을 추가하고

remoteControl.setCommand(5, lightDimUp, lightDimDown);

위와 같이 5번 슬롯에 추가해 준다.

추가하고자 하는 기능이 있을 때 Command 부분만 추가를 하면 간단히 추가할 수 있다. 실행을 직접 하지 않음으로 인해 상위 클래스가 하위 클래스의 코드 수정에 영향을 받지 않는다는 장점이 있다. 그렇기 때문에 확장도 용이하다.

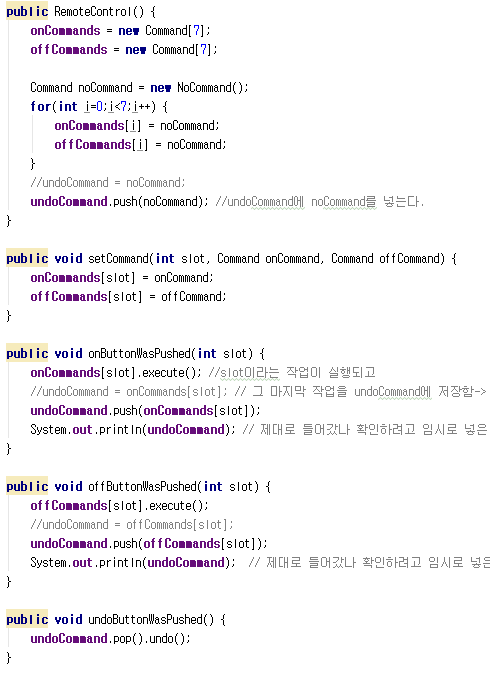
단점으로는 Command 인터페이스에 뭔가가 추가되고 삭제되었을 때에 이를 구현하는 모든 클래스들을 수정해 줘야 한다.

1. 현재 RemoteControl 클래스는 한번의 작업 취소(Undo)만 가능하다. 이를 수정하여 여러번의 작업 취소가 가능하도록 RemoteControl 클래스를 수정하고 이 기능을 테스트한 결과를 제출하세요.

[힌트] 실행한 커맨드를 스택에 집어넣고, 사용자가 UNDO 버튼을 누를때마다 스택 맨 위에 있는 항목을 꺼내서 undo() 메소드를 호출하면 됩니다.

->>설명 전에 전체 코드 사진을 첨부하겠습니다.

(맨 위에 스택 선언 부분 코드는 밑에 첨부)



첫번째로 아래 사진처럼 RemoteControl 클래스에서 스택을 사용할 수 있게 선언을 하였다.

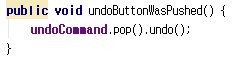


기존 코드는 undoCommand = onCommand[slot];과 같이 그저 변수에 전에 실행되었던 명령 한 가지만 들어가게 되어있었지만 스택을 선언하고 위의 코드를 아래 사진과 같이 수정하였다.



push명령어를 통해 명령을 하게 되면 그 명령을 스택에 하나씩 쌓아가는 구조이다. 이처럼 offCommands[slot]도 똑같이 구현하였다.

이런식으로 스택을 쌓아놓고undoButtonWasPushed() 메소드에서



다음과 같은 코드를 구현해 undo 버튼을 누를 때마다 pop()을 이용해 마지막에 들어간 명령부터 꺼내어지게 구현하였다.

마지막으로 결과 사진을 밑에 첨부하겠습니다.

결과 사진의 메뉴가 길어서 한번에 다 담지는 못해서

2장으로 나누어서 첨부하도록 하겠습니다.

순서는

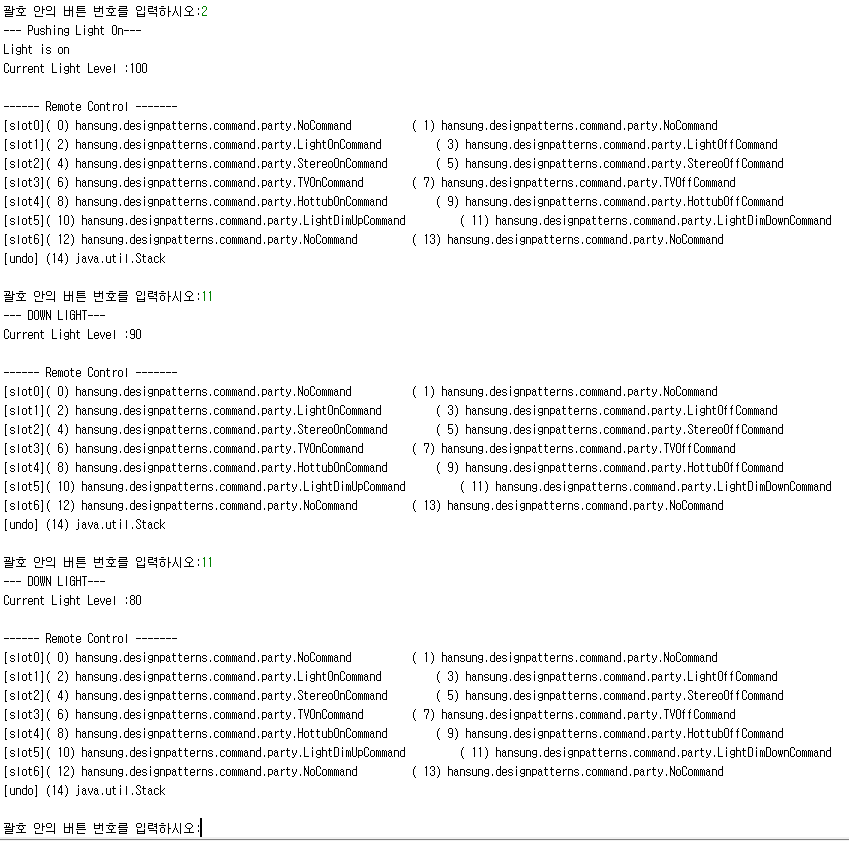
Light on->down->down->up->undo->undo 순서

100->90->80->90->80->90 순서

첫번째 결과화면

Light on->down->down

100->90->80 사진



두번째 결과화면

up->undo->undo

90->80->90 사진

