제주대 객체 지향 학기말 과제: 선풍기 제어 SW 설계하기

최범균(madvirus@mavirus.net)

과제

- 본 문서를 세심하게 읽고, 과제를 제출할 것
- 본 문서는 다음과 같이 구성됨
  - 만들 대상 설명 (선풍기 제어 SW)
  - 흔히 하는 잘못된 설계 소개
  - 과제 설명

1. 만들 대상: 선풍기 제어 SW

가상의 J전자 선풍기 제어 SW 만들기

- J전자는 한 가지 모델의 선풍기만 생산하다가, 다양한 시장을 공략하기 위해 다양한 종류의 선풍기를 생산할 계획을 세웠음
- 각 선풍기 모델마다 H/W 제어 API가 달라질 수 있음
  - 예, 현재 개발중인 모델 중에서는 회전 기능과 리모콘 기능이 없는 모델이 있음
- 이들 다양한 종류의 선풍기 모델을 제어하는데 사용할 선풍기 제어 SW를 만들고 싶음

선풍기 기능

- 바람 세기를 1-3 단계로 조절할 수 있다.
- 바람을 회전시키거나 고정시킬 수 있다.
- 선풍기를 켜거나 끌 수 있다.
- 타이머 기능이 있어서, 지정한 시간 뒤에 선풍기를 끌 수 있다.
- 리모콘으로 동일 기능을 실행할 수 있다.

선풍기 제어 SW가 감시/제어해야 할 장치

장치	설명
바람 발생용 모터	켜거나 끌 수 있고, 바람 세기를 조절할 수 있다.
회전용 모터	켜거나 끌 수 있다.
타이머	지정한 시간이 지나면 타이머로부터 신호를 받아 종료한다.
선풍기 겉면에 붙은 버튼의 상태	전원/풍속 제어버튼, 방향 조절 버튼, 타이머 시간 조절 버튼이 존재하고, 각 버튼을 누르면 알맞은 기능을 수행한다.
적외선 수신부	리몬콘에서 발사한 적외선을 수신해서, 수신한 신호에 따라 알맞은 기능을 수행한다.

HW에서 제공하는 API를 이용해서 장치와의 연동 처리를 함

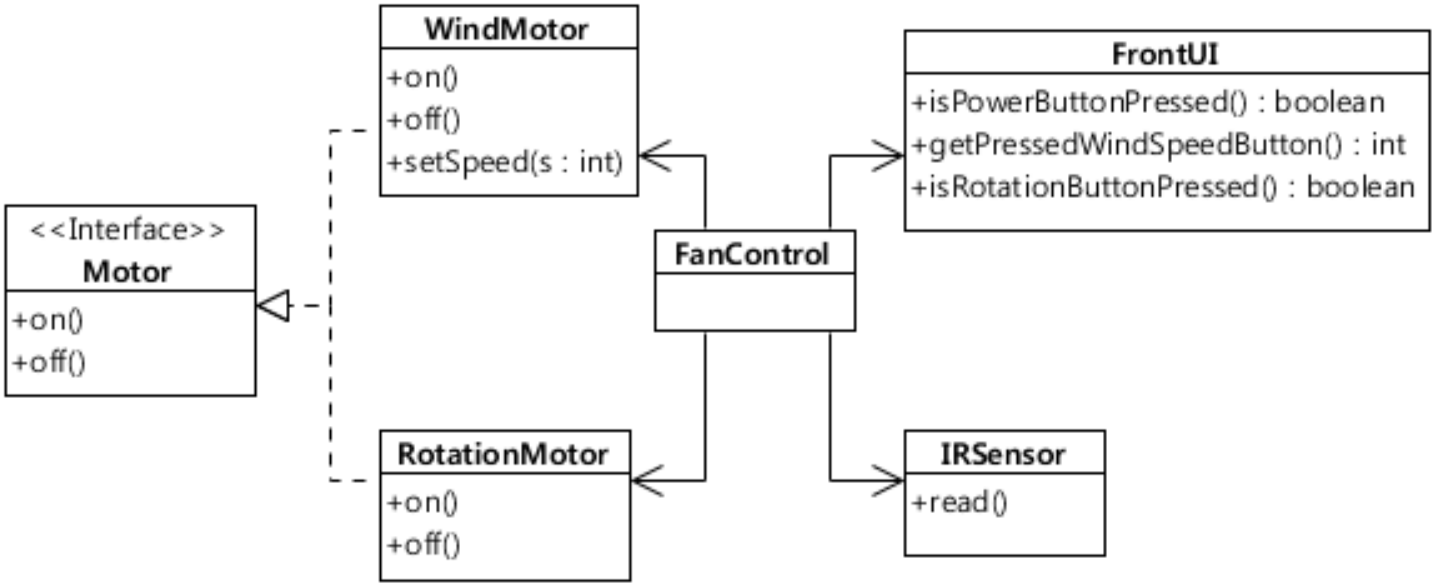
특정 선풍기 모델의 HW 관련 API 예제

```
package com.je.fan.modelA.api;

public abstract ModelAApi {
    public static enum WindSpeed {LEVEL0, LEVEL1, LEVEL2, LEVEL3}

    public abstract void setSpeed(WindSpeed speed);
    public abstract void turnOn();
    public abstract void turnOff();
    public abstract void startRotation();
    public abstract void stopRotation();

    public abstract boolean getPowerButtonStatus();
    public abstract WindSpeed getSpeedButtonStatus();
    public abstract boolean getRotationButtonStatus();

    public abstract boolean isTimeout();

    public static ModelAApi instance() { … }
}
```

*** 각 모델마다 제공되는 HW 제어 API 다르다고 가정**

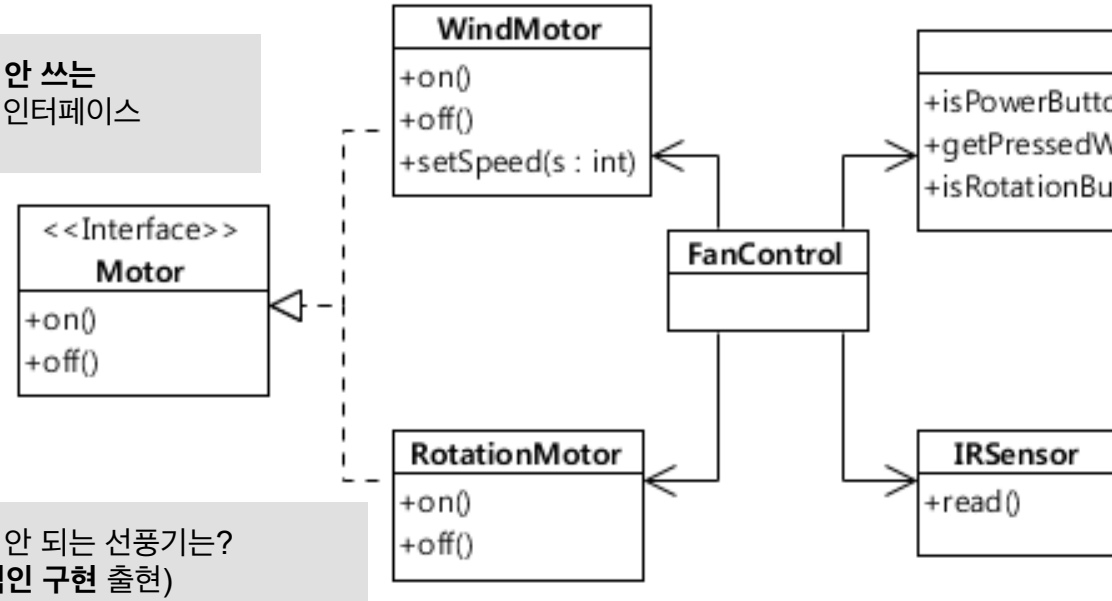
2. 최초의 잘못된 설계

선풍기 제어 SW:
최초의 설계는 대부분 이런식



문제점: HW에 종속됨, 구체적 구현에 의존, 불필요한 인터페이스

아무도 안 쓰는
Motor 인터페이스

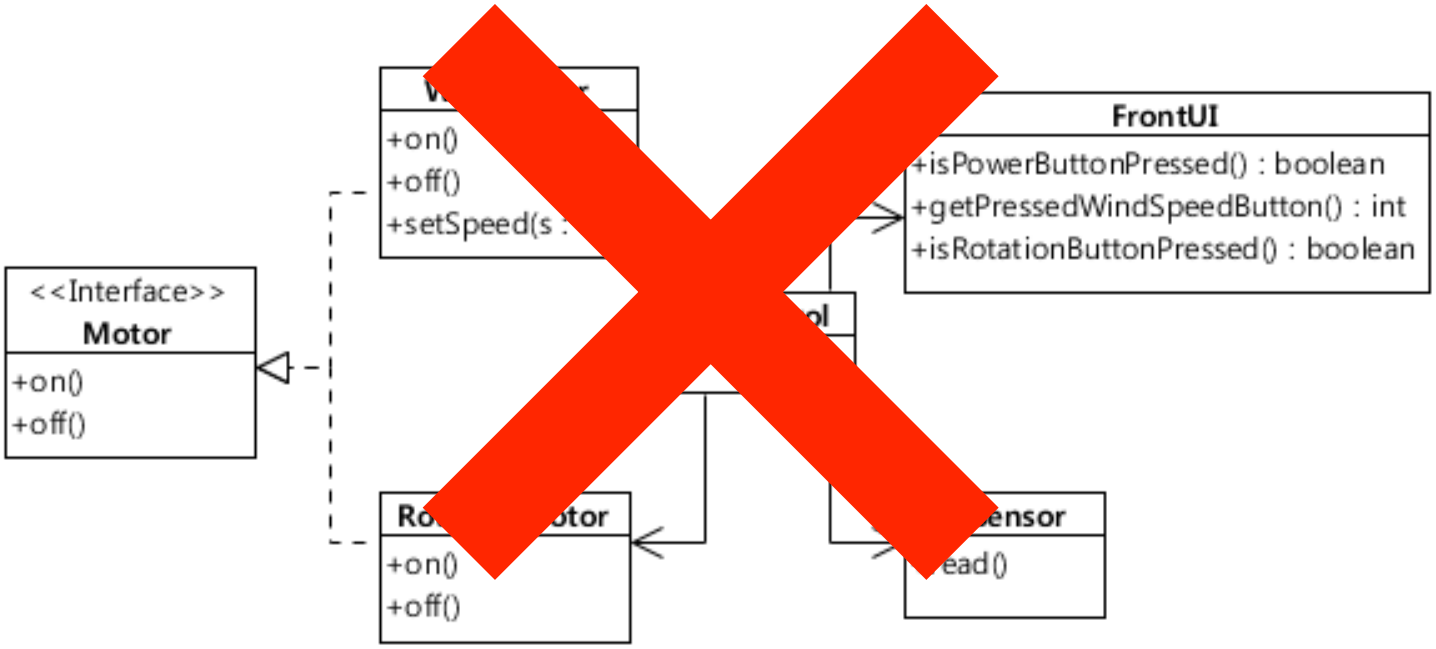


회전이 안 되는 선풍기는?
(**구체적인 구현** 출현)

리모콘이 없으면?
(**구체적인 구현** 출현)

```
public class RotationMotor {
    public void on() {
        // 제어SW에서 하드웨어 종속
        ModelAApi.instance().turnOnRotation();
    }
}
```

잘못된 설계의 원인: 문제의 본질이 없음



선풍기 SW의 본질은

“**바람을 발생시키고, 바람의 방향을 조절하는 것**이지”,

“모터를 돌리고, 적외선을 수신하는 것”이 아님!

잘못된 설계의 결과

- 선풍기 모델(즉, HW)마다 동일한/비슷한 모델을 새롭게 만들어야 주어야 하는 상황 발생!!
  - 즉, 선풍기 SW 모델의 중복 개발 문제

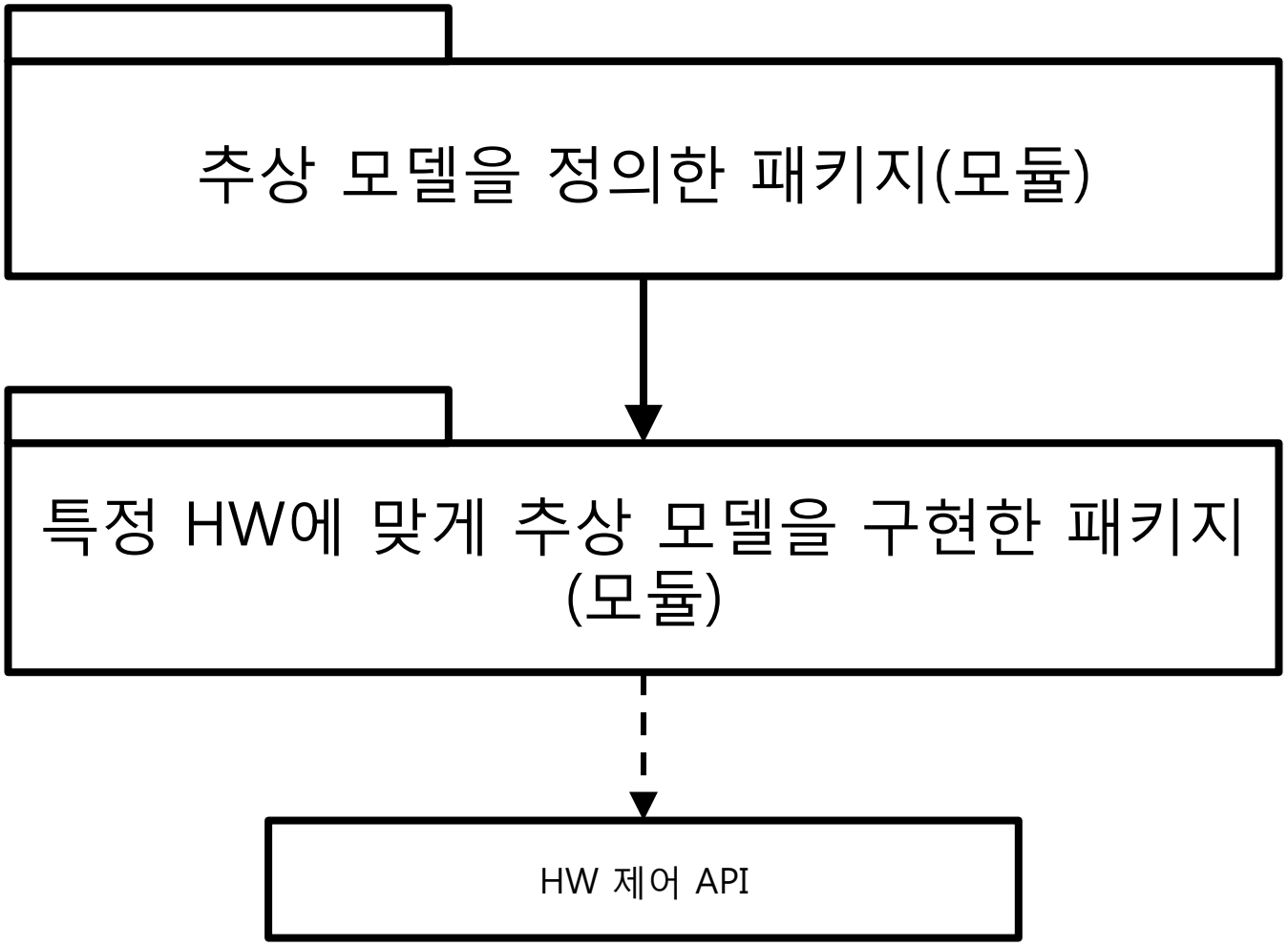
- 원하는 것은 **모든 선풍기에 맞는 공통된 모델을 만들고, 그 모델을 재사용**하는 것임!

3. 과제 안내

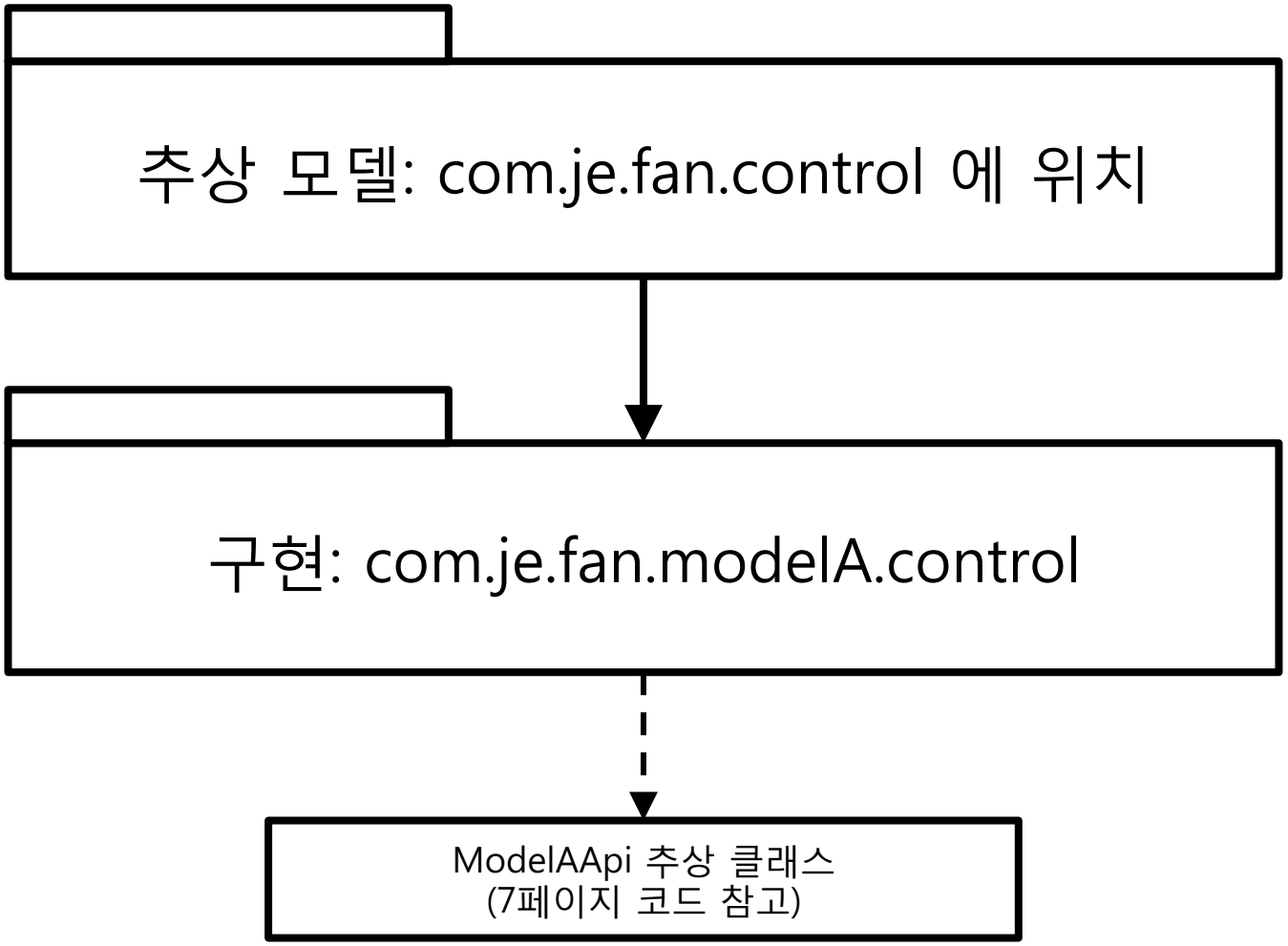
과제

- 각 선풍기 모델에 의존적이지 않은 선풍기 제어 모델 만들기!
- 힌트!
  - 이런 모델을 만드려면, 다음과 같이 구분해야 함
    - 선풍기 제어 본질을 표현하는 추상 모델 모듈
    - 추상 모델을 각 선풍기 모델(HW)에 알맞게 구현한 모듈
  - 추상 모델 모듈은
    - HW(모터, 센서, 버튼 등)에 의존을 가지면 안 됨
    - 소프트웨어의 본질을 모델
  - 선풍기 모델 별 구현 모듈
    - 하드웨어에 대한 의존은 구현 모듈이 가짐

모듈 구성 힌트(DIP를 사용할 것)



ModelA용 제어 SW 패키지 위치



Pollable 인터페이스

- ModelA 선풍기 HW는 Pollable 인터페이스를 implements 한 객체의 poll() 메서드를 0.1초 단위로 호출한다고 가정

```
// HW가 주기적(0.1초)으로 아래 코드 실행
for (Pollable pollable : pollableList)
    pollable.poll();
```

- Pollable 인터페이스 정의

```
package com.je.fan.modelA.api;

public interface Pollable {
    public void poll();
}
```

Pollable 인터페이스 구현

- 추상 모델을 구현한 클래스는 HW 정보를 읽어와야 할 경우 Pollable.poll() 메서드에서 읽어오도록 구현해야 함

```
// 예시

package com.je.fan.modelA.control;

import com.je.fan.control.UI;
// UI가 실제 도출될 추상 모델 중의 하나인 경우 (추상 클래스 또는 인터페이스)

public class ModelAUI extends UIimplements Pollable {
    private ModelAApi api;

    public void poll() {
        // 주기적으로 HW의 상태 정보를 읽어와 모델에 알맞은 기능 실행
        WindSpeed speed = api.getSpeedButtonStatus();
        … //
    }
    … // 필요한 의존 객체는 DI를 통해서 전달 받음
```

숙제 제출 관련

- 제출기한: **12월 13일 24시**까지!
- 결과물
  - 소스 코드 (이클립스 또는 인텔리J 프로젝트로 제출할 것)
- 소스 코드 제약
  - 각 객체는 DI를 통해서 조립할 것
  - ModelAApi를 상속한 가짜 구현 클래스를 만들것
    - ModelAApi.instance()는 가상의 구현 클래스의 인스턴스를 리턴할 것
    - 가상의 구현 클래스
  - 테스트를 위해 각 객체를 조립하고 실행하는 별도 클래스인 Main 클래스(main() 메서드) 제공할 것!

ModelAApi 가짜 구현 작성 예시 1/2

- HW UI 상태를 변경해주는 메서드 제공

```
public abstract class ModelAApi {
    … // 추상 메서드들 (7페이지 코드 참고)
    public ModelAApi instance() { return new FakeModelAApi(); }
}

public class FakeModelAApi extends ModelAApi {
    @Override
    public WindSpeed getSpeedButtonStatus() {
        return windSpeed;
    }
    public void setSpeedButtonStatus(WindSpeed speed) {
        this.speed = speed;
    }
    ...
}
```

ModelAApi 가짜 구현 작성 예시 2/2

- API 정의된 기능 실행시, 확인 위해 콘솔 출력 사용 제공

```
public abstract class ModelAApi {
    … // 추상 메서드들 (7페이지 코드 참고)
    public ModelAApi instance() { return new FakeModelAApi(); }
}

public class FakeModelAApi extends ModelAApi {
    @Override
    public void turnOn() {
        System.out.println(“선풍기 켬”);
    }
    ...
}
```

Main 클래스의 작성 예시

- 가짜 ModelAApi를 이용해서 상태 변경

```
public class Main {
    public static void main(String[] args) {
        FakeModelAApi api = (FakeModelAApi)ModelAApi.instance();

        // 모델 객체들 조립
        ModelAUI ui = new ModelAUI(api, ….); // DI로 의존 객체 전달
        …
        // 생성한 모델 객체를 Pollable 목록에 보관
        List<Pollable> pollables = new ArrayList<>();
        pollables.add(ui); // ModelAUI가 Pollable을 implement 했다면,
        …
        // 가짜 api로 HW 상태 변경
        api.setSpeedButtonStatus(WindSpeed.LEVEL3);
        for (Pollable p : pollables) p.poll(); // Pollable의 poll을 호출해서 변경된 HW 상태 모델에 반영
        // 반영 여부 확인 위해 모델 구현 클래스에서 System.out으로 출력된 메시지 확인

        // 동일한 방식으로 HW 상태 변경하고, 모델에 반영되는지 확인
    }

}
```