Chap 08_2. Advanced Class Features

학습목표

1. static keyword
2. final keyword
3. abstract class
4. interface

- Member Variable, Member Method 앞에 사용
- 일반적으로 멤버변수나 멤버메소드는 한 객체에서 의미가 있음
- static 키워드를 사용하면, 한 객체가 아닌 클래스 자체와 연관
- Class 변수, Class 메소드라고 불리운다.
- Class 변수, Class 메소드는 객체 생성 없이 사용한다.

static keyword – class 변수

- Class 로드시에 메모리에 생성되어진다.
- 전체 객체에서 공용으로 쓰이는 변수이다. (전역적인 성격)
- 만약, 다른 클래스에서 사용되어지면 *클래스이름.변수명* 으로 참조

```
public class Count {
    private int serialNumber;

    public static int counter = 0;

    public Count() {
        counter++;
        serialNumber = counter;
    }
}
```

```
public class TestCount {
    public static void main( String [] args ) {

        System.out.println( "Number of counter is "
                            + Count.counter );

        Count count1 = new Count();
        System.out.println( "Number of counter is "
                            + Count.counter );

        Count count2 = new Count();
        System.out.println( "Number of counter is “
                            + Count.counter );
    }
}
```

static keyword – static 메소드

- 객체생성 없이 호출될 수 이다.
- *클래스이름.메소드명()* 으로 호출
- static 메소드 안에서는 this, super, non-static 멤버들은 사용할 수 없다.
  (local 변수는 가능)

```
public class Count2 {

    private int serialNumber;
    private static int counter = 0;

    public static int getTotalCount()  {
        return counter;
    }

    public Count2()
    {
        counter++;
        serialNumber = counter;
    }
}
```

```
public class TestCount2 {
    public static void main( String [] args ) {

        System.out.println( "Number of counter is “
                            + Count2.getTotalCount() );

        Count2 count1 = new Count2();
        System.out.println( "Number of counter is "
                            + Count2.getTotalCount() );

        Count2 count2 = new Count2();
        System.out.println( "Number of counter is "
                            + Count2.getTotalCount() );
    }
}
```

- final class – 상속 못 하게 함
- final method – override 못 하게 함
- final variable – 상수

► final varialble

- 선언시 초기화 안 할 수도 있다. (blank final variable)
- 단, 선언시 초기화 안 한 경우 모든 생성자에서 초기화 해주어야 한다.
- 사용되기 전에 초기화가 되어 져야 한다. (local 변수와 동일)

- Radio, TV, MP3 세 개의 클래스를 살펴 본 결과, 공통 되는 멤버 변수와 멤버 메소드가 존재한다.
- 이 공통 되는 부분들을 상위 클래스로 작성 하면 어떨까?

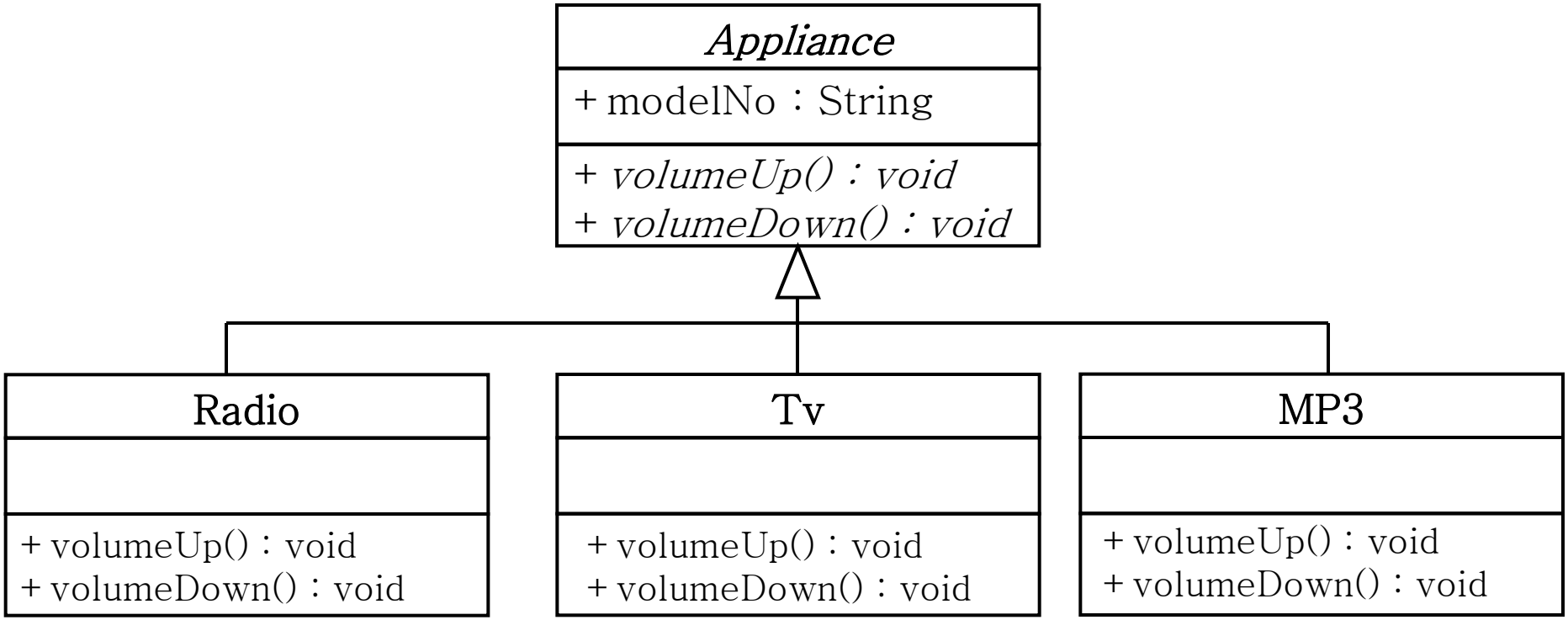
Super Class ?

Radio
+ modelNo : String
+ volumeUp() : void + volumeDown() : void

Tv
+ modelNo : String
+ volumeUp() : void + volumeDown() : void

MP3
+ modelNo : String
+ volumeUp() : void + volumeDown() : void

상속인 경우 문제점은?


Appliance
+ modelNo : String
+ volumeUp() : void
+ volumeDown() : void
Radio
+ volumeUp() : void
+ volumeDown() : void
Tv
+ volumeUp() : void
+ volumeDown() : void
MP3
+ volumeUp() : void
+ volumeDown() : void

```
abstract public class Appliance {
    private String modelNo;

    public Appliance() {
    }

    public Appliance( String modelNo ) {
        this.modelNo = modelNo;
    }

    public String getModelNo() {
        return modelNo;
    }

    public void setModelNo( String modelNo ) {
        this.modelNo = modelNo;
    }

    abstract public void volumeUp();

    abstract public void volumeDouwn();
}
```

```
public class Radio extends Appliance {
    public Radio( String modelNo ) {
        super( modelNo );
    }

    public void volumeUp() {
        System.out.println( "라디오 볼륨업" );
    }

    public void volumeDouwn() {
        System.out.println( "라디오 볼륨다운" );
    }
}
```

Abstract Class – 정리

- Abstract Class
  - 미완성 클래스 (abstract 키워드 사용)
  - abstract 메소드가 포함 된 클래스 → 반드시 abstract 클래스
  - 자체적으로 객체 생성 불가 → 반드시 상속 통하여 객체 생성
  - 일반적인 메소드, 멤버변수도 포함할 수 있다.
  - abstract 메소드가 없어도, abstract 클래스 선언 가능
  - 객체 생성은 안되나, Ref 변수로서는 가능하다.

    ex) Appliance app = new Radio();

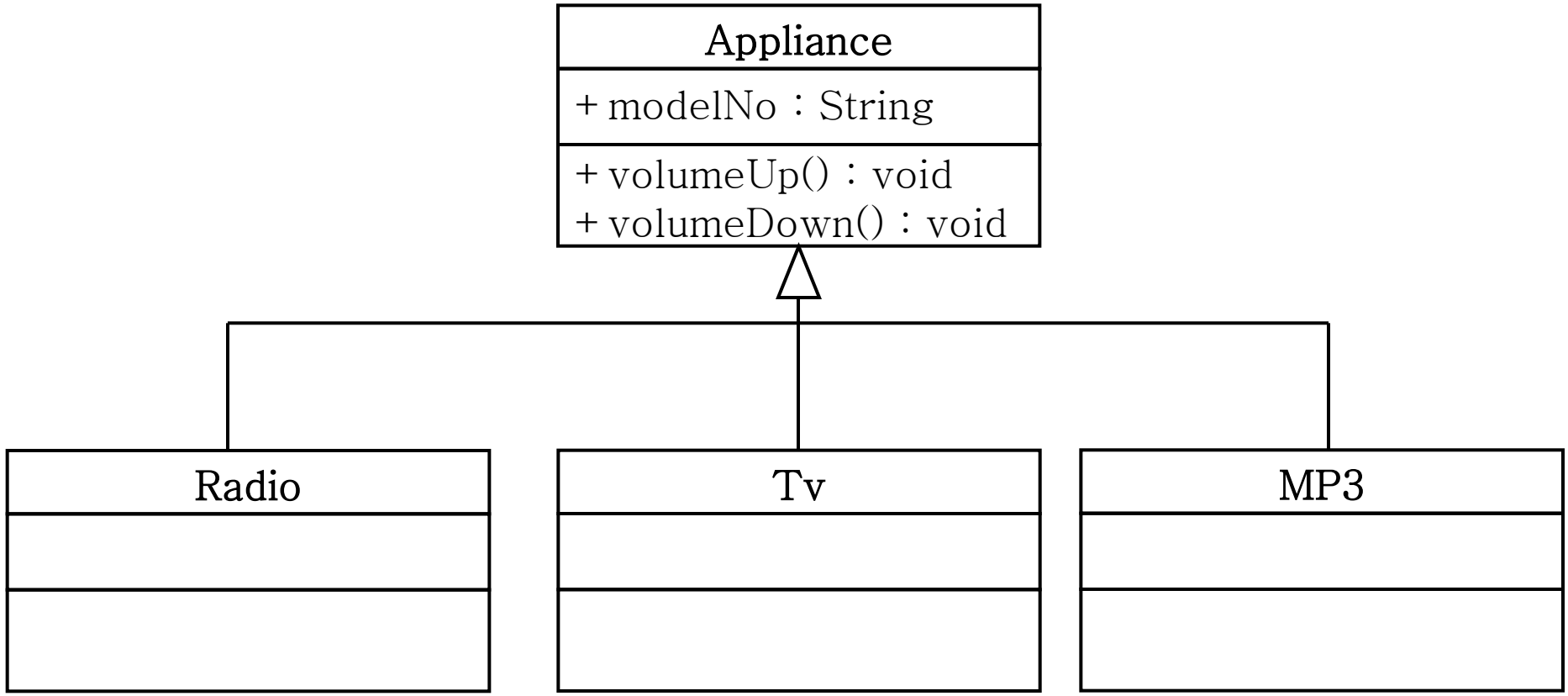
- Abstract Method 선언
  - { } (메소드 body)가 없는 메소드 , ';' 으로 끝나야 한다.

- Abstract Class 장점
  - 일관된 인터페이스 제공
  - 꼭 필요한 기능 강제함 (공통적이긴 하나, 자식클래스에서 특수화 되는 기능)

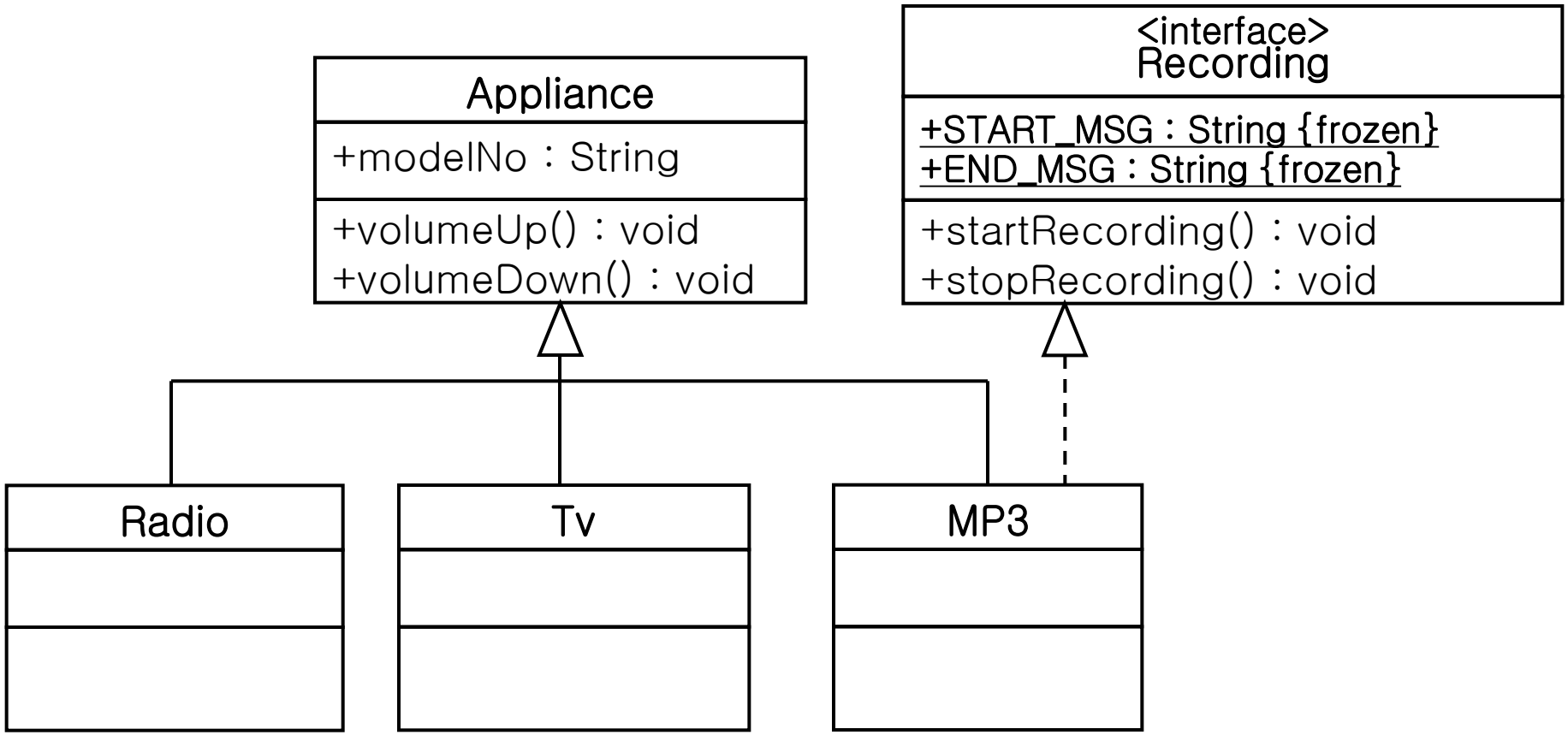
❑ 추가 요구 사항 도출

1) MP3 Player는 레코딩 기능이 있음

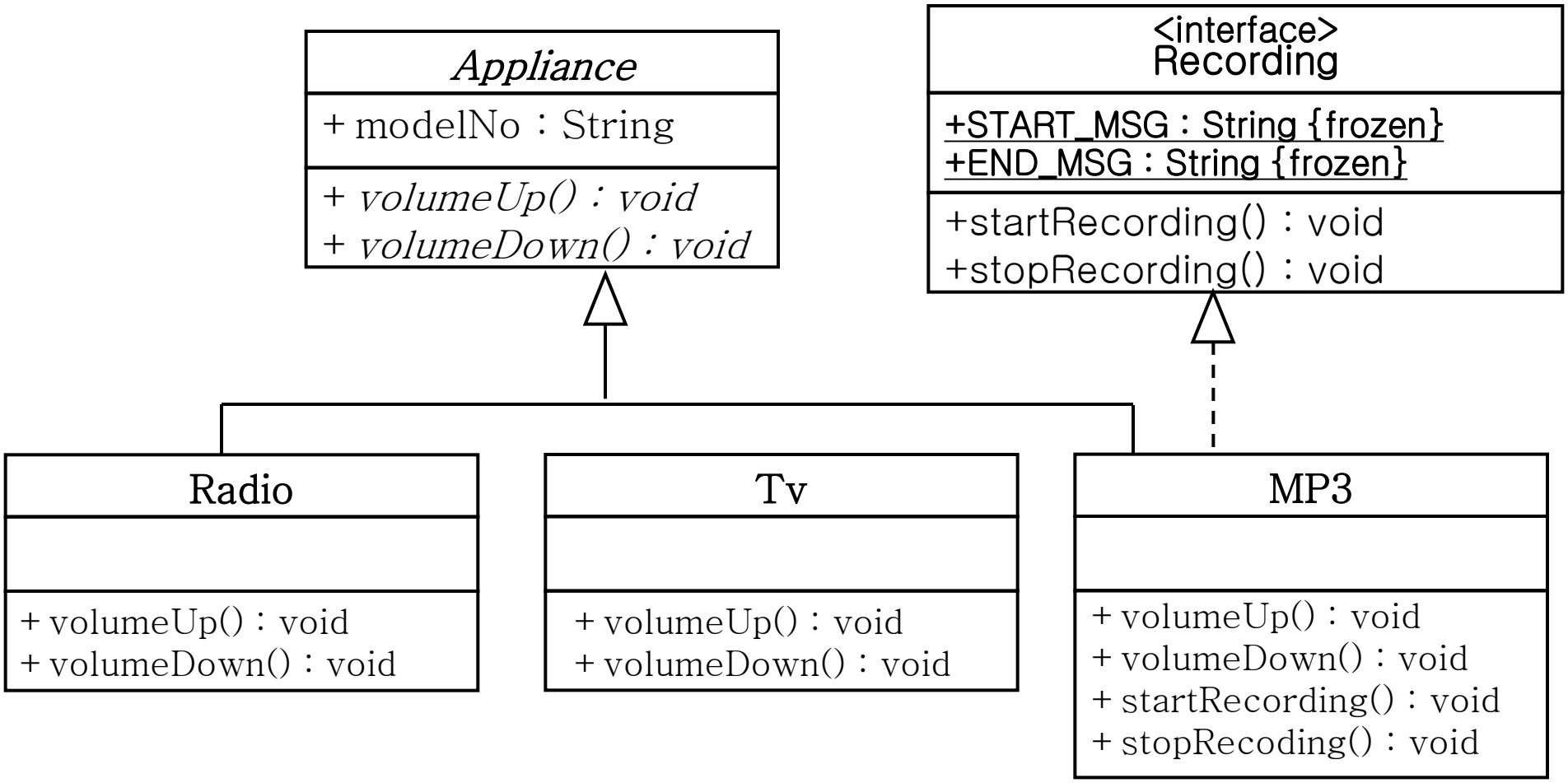
2) 추후 새로 개발되는 가전 제품에는 레코딩 기능 제공 예정



❑ 해결 : 레코딩에 관계 되는 메소드 형식만 모아 놓은 클래스 제공



❑ 해결 : 레코딩에 관계 되는 메소드 형식만 모아 놓은 클래스 제공



```
public interface Recording {

    public static final String START_MSG
                    = "Recording Started";
    String END_MSG
                    = "Recording Ended";

    public abstract void startRecord();
    void stopRecord();
}
```

```
public class MP3 extends Appliance
                    implements Recording {

    public MP3( String modelNo ) {
        super( modelNo );
    }

    public void volumeUp() {
        System.out.println( "MP3 볼륨업" );
    }

    public void volumeDouwn() {
        System.out.println( "MP3 볼륨다운" );
    }

    public void startRecord() {
        System.out.println( START_MSG );
    }

    public void stopRecord() {
        System.out.println( END_MSG );
    }
}
```

Interface – 정리

❑ Interface

- 모든 메소드가 abstract 메소드인 클래스
- 메소드는 묵시적으로 public abstract
- 멤버변수는 묵시적으로 public static final
- implements 키워드로 구현
- 다중 구현 가능 → 단일 상속 극복
- 객체 생성은 안되나, Ref 변수로서는 가능하다.

  ex) Recording app = new MP3();

❑ Interface의 장점

- 공통 기능상의 일관성 제공
- 공동 작업을 위한 인터페이스 제공