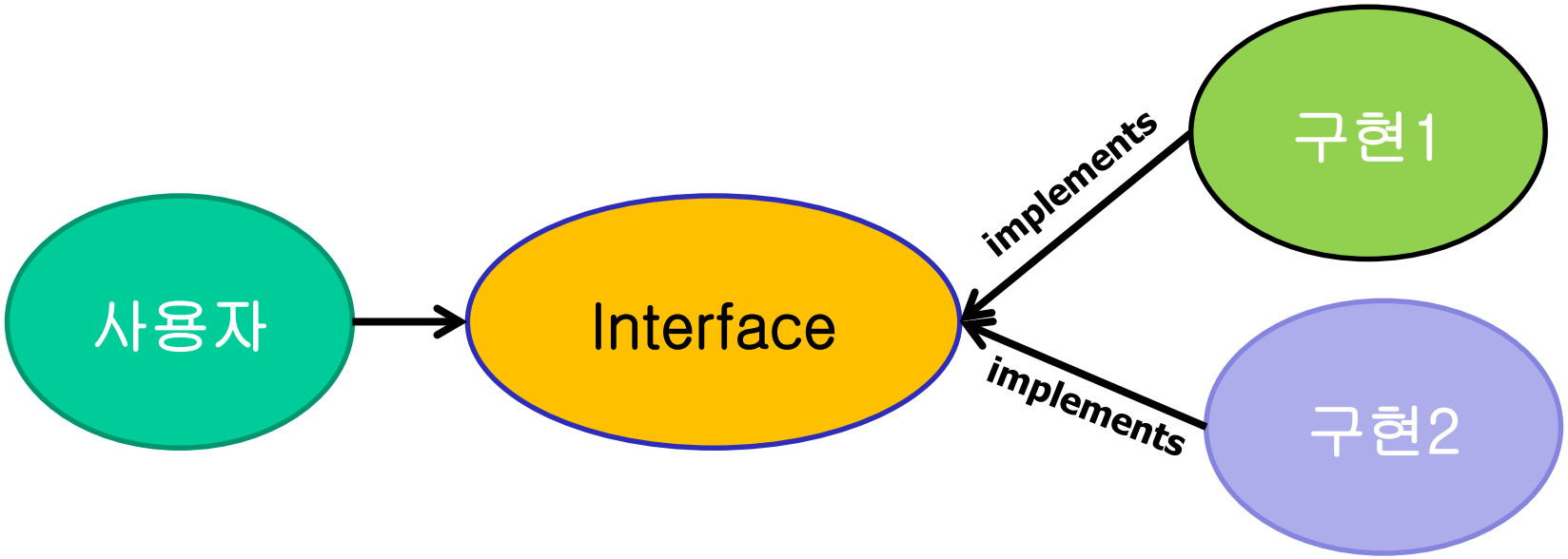
Interface

경복대학교 소프트웨어융합과
배희호 교수
010-2369-4112
031-570-9600
hhbae@kbu.ac.kr

Interface

- Interface 필요성



  - Interface를 상속 받아 Class를 정의하는 경우에는 Interface에서 정의된 모든 Abstract Method를 Override 해야 함
  - 그렇지 않은 경우 Compile Error를 발생 시킴

Interface

- 정의
  - Class 대신 Interface로 선언한 객체를 의미
  - 일종의 Abstract Class로 Abstract Class보다 추상화 정도가 높음

```
접근 지정자 interface 인터페이스 이름{
    상수;
    접근 지정자 추상 메소드 이름(인자들);
}
```

- 용도
  - 기능만을 상위에서 하위 Class로 강제하기 위해 사용
  - 다른 Sub Class 작성에 도움을 줄 목적임

Interface

- 특징
  - Object 생성이 불가능
  - Member로 Abstract Method, 일반 Method와 변수를 가질 수 있었던 “Abstract Class"와 달리, Interface는 Abstract Method와 상수 만을 구성 Member로 허용
  - 하위 Class를 이용하여 Member를 재사용
  - Type으로 사용 가능
  - 하위 Class에 강제성과 통일성을 줄 수 있음
  - 단일 상속의 단점을 보완할 수 있음
  - 하위 Class에서는 extends 대신 implements를 사용
  - 하위 Class에서는 다중 implements가 가능

Interface

- Interface 특징
  - Interface 역시 상속 관계에서 쓰이는데 이때 implements Keyword를 사용
  - 구현 받은 클래스는 반드시 상위 클래스의 추상 메소드를 “Overriding" 해야 함
  - 일반적으로 클래스를 상속 받을 때 1:1만 가능하였으나 Interface는 다중 상속 가능

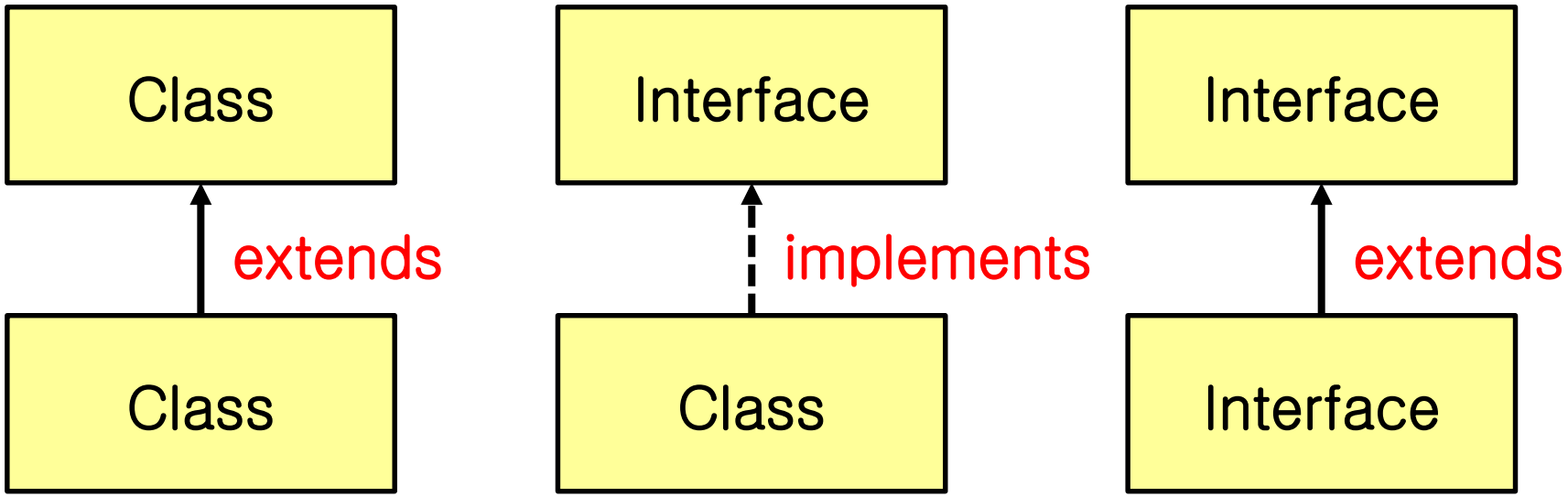
- Interface 장점
  - 개발 시간 단축
  - 표준화

Interface

- Interface 확장
  - Interface는 Class와 마찬가지로 확장해서 새로운 Interface를 선언할 수 있음
  - 확장되는 Interface를 Super Interface, 확장하는 Interface를 Sub Interface라고 함
  - Interface를 확장할 때에는 "extends"를 사용하지만, 이러한 Interface를 Class로 구현할 경우에는 "implements"를 사용

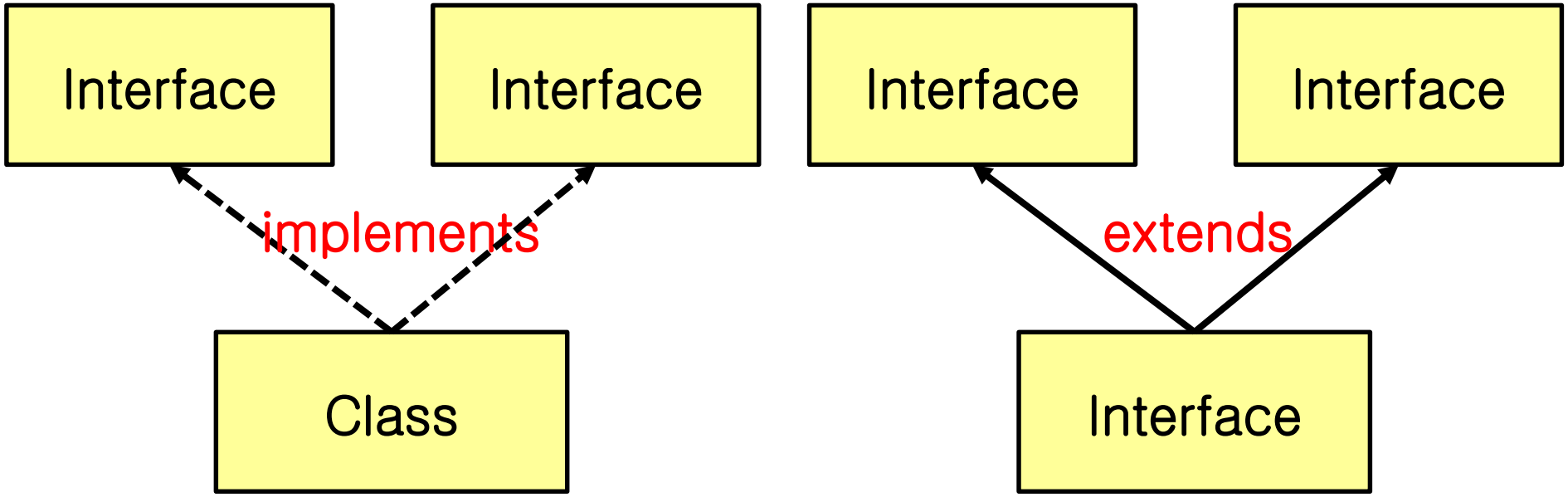
Interface

- Class와 Interface 관계



Interface

- Interface에 의한 다중 상속



Interface 구현

- implements Keyword 사용
- 여러 개의 Interface 동시 구현 가능
- 상속과 구현이 동시에 가능

```
interface MouseInterface {      // 인터페이스
    void mouseMove();
    void mouseClick();
}

public class MouseDriver implements MouseInterface {
    void mouseMove() { .... }      // 반드시 구현해야 함
    void mouseClick() { ... }

    // 추가적으로 다른 메소드를 작성할 수 있음
    int getStatus() { ... }
    int getButton() { ... }
}
```

Interface 구현

```
public interface Clock {
   public static final int ONEDAY = 24;          // 상수 필드 선언
   abstract public int getMinute();
   abstract public int getHour();
   abstract void setMinute(int i);
   abstract void setHour(int i);
}

public interface Car {
   int MAXIMUM_SPEED = 260;          // static final 생략
   int moveHandle(int degree);       // abstract public 생략
   int changeGear(int gear);         // abstract public 생략
}

new Clock();   // 오류.   인터페이스의 Object 생성 불가
new Car();     // 오류.   인터페이스의 Object 생성 불가

Clock clock;   // 인터페이스 Clock의 Reference 변수 선언 가능
Car car;       // 인터페이스 Clock의 Reference 변수 선언 가능
```

Interface 구현

- Interface의 다중 구현

```
interface USBMouse {
  void mouseMove();
  void mouseClick();
}

interface RollMouse {
  void roll();
}

public class MouseDriver implements RollMouse, USBMouse {
  void mouseMove() { .... }
  void mouseClick() { ... }
  void roll() { ... }

  // 추가적으로 다른 메소드를 작성할 수 있음
  int getStatus() { ... }
  int getButton() { ... }
}
```

Hello

- IHello.JAVA

```
interface IHello{
    void sayHello(String name);
}
```

```
interface IHello{
    public abstract void sayHello(String name);
}
```

Hello

- Hello.JAVA

```
class Hello implements IHello{
    //인터페이스의 추상 메서드 오버라이딩 :
                                    접근 지정자를 반드시 public으로 해야 함
     public void sayHello(String name){
          System.out.println(name+"씨 안녕하세요!");
     }
}
```

Hello

- Main.JAVA

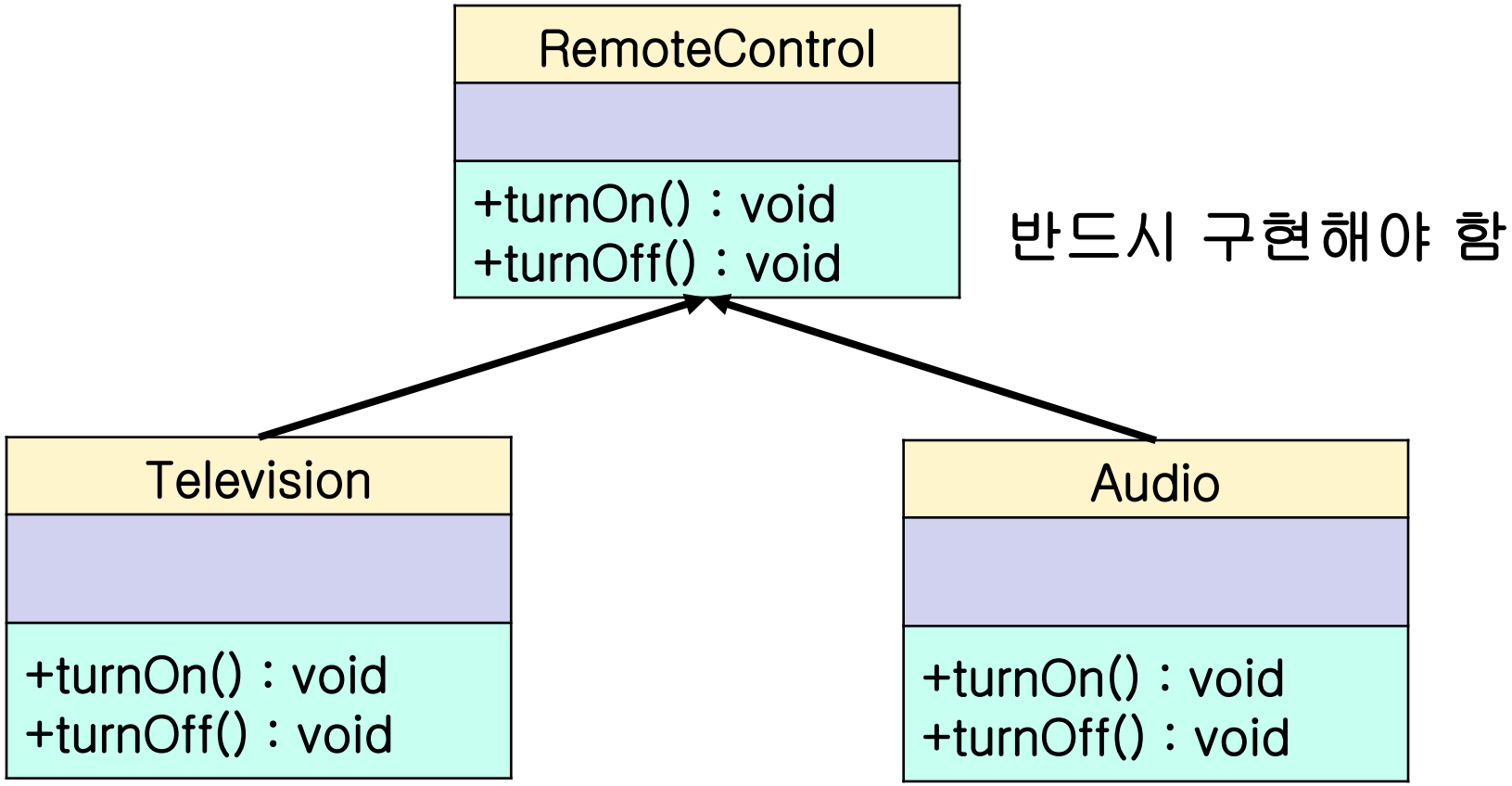
```
public static void main(String[] args) {
    Hello hello = new Hello();

    hello.sayHello("홍길동");
}
```

Remocon

- 만능 리모콘을 가지고 TV와 Audio의 전원을 제어해보자

Remocon(I)

- Class Diagram



Remocon(I)

- RemoteControl.JAVA

```
public interface RemoteControl {
    public void turnOn();
    public void turnOff();
}
```

Remocon(I)

- Television.JAVA

```
public class Television implements RemoteControl{

    @Override
    public void turnOn() {
        System.out.println("TV를 켜다");
    }

    @Override
    public void turnOff() {
        System.out.println("TV를 끄다");
    }
}
```

Remocon(I)

- Audio.JAVA

```
public class Audio implements RemoteControl{
    @Override
    public void turnOn() {
        System.out.println("Audio 전원을 켜다");
    }

    @Override
    public void turnOff() {
        System.out.println("Audio 전원을 끄다");
    }
}
```

Remocon(I)

- Main.JAVA

```
public static void main(String[] args) {
    Television myTV = new Television();
    Audio myAudio = new Audio();

    RemoteControl control = myTV;
    control.turnOn();
    control.turnOff();
    control = myAudio;
    control.turnOn();
    control.turnOff();
}
```

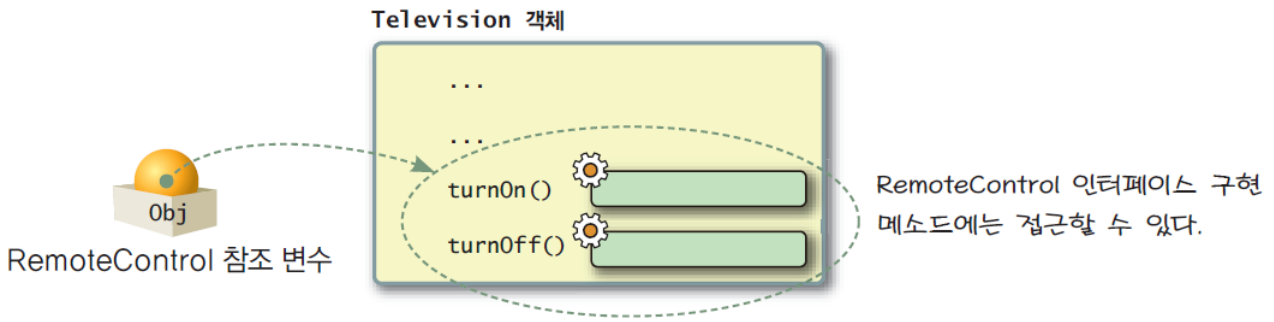
Remocon(I)

- Interface와 Type

```
RemoteControl obj = new Television();
obj.turnOn();
obj.turnOff();
```

Television 객체이지만 RemoteControl 인터페이스를 구현하기 때문에 RemoteControl 타입의 변수로 가리킬 수 있다.

obj를 통해서는 RemoteControl 인터페이스에 정의된 메소드 만을 호출할 수 있다.



Remocon(II)

- RemoteControl.JAVA

```
public interface RemoteControl {
    public static final int MAX_VOLUME = 10; // 최대 볼륨
    public static final int MIN_VOLUME = 0;   // 최소 볼륨

    // '추상 메소드' 선언
    // public abstract 생략 가능
    public abstract void turnOn();  // 켜기
    public abstract void turnOff(); // 끄기
    public abstract void setVolume(int volumn); // 볼륨 조절

    public abstract void setChannel(int channel);

    default void setMute(boolean mute) {
        if (mute) {
            System.out.println("무음 처리 합니다.");
        } else {
            System.out.println("무음 해제 합니다.");
        }
    }
```

Remocon(II)

- RemoteControl.JAVA

```
    default void setMute(boolean mute) {
        if (mute) {
            System.out.println("무음 처리 합니다.");
        } else {
            System.out.println("무음 해제 합니다.");
        }
    }

    // '정적' 메소드 구현
    // 허용한 이유 => 디폴트 메소드와는 달리 객체가 없어도
                                                인터페이스만으로 호출이 가능함 !
    static void changeBattery() {
        System.out.println("건전지를 교환합니다.");
    }
}
```

Remocon(II)

- Television.JAVA

```
public class Television implements RemoteControl{
    private int volume;
    private int channel;

    @Override
    public void turnOn() {
        System.out.println("TV를 켭니다.");
    }

    @Override
    public void setChannel(int channel) {
        this.channel = channel;
    }
```

Remocon(II)

- Television.JAVA

```
    @Override
    public void setVolume(int volume) {
        if(volume > RemoteControl.MAX_VOLUME) {
            this.volume = RemoteControl.MAX_VOLUME;
        }else if(volume < RemoteControl.MIN_VOLUME){
            this.volume = RemoteControl.MIN_VOLUME;
        }else{
            this.volume = volume;
        }
        System.out.println("현재 TV 볼륨 : " + this.volume);
    }

    @Override
    public void turnOff() {
        System.out.println("TV를 끕니다.");
    }
}
```

Remocon(II)

- Audio.JAVA

```
public class Audio implements RemoteControl{
    private int volume;

    @Override
    public void turnOn() {
        System.out.println("Audio를 켭니다.");
    }

    @Override
    public void turnOff() {
        System.out.println("Audio를 끕니다.");
    }
```

Remocon(II)

- Audio.JAVA

```
    @Override
    public void setVolume(int volume) {
        if(volume > RemoteControl.MAX_VOLUME) {
            this.volume = RemoteControl.MAX_VOLUME;
        }else if(volume < RemoteControl.MIN_VOLUME){
            this.volume = RemoteControl.MIN_VOLUME;
        }else{
            this.volume = volume;
        }
        System.out.println("현재 Audio 볼륨 : " + this.volume);
    }

    @Override
    public void setChannel(int channel) {
    }
}
```

Remocon(II)

- Lamp.JAVA

```
public class Lamp implements RemoteControl{
    @Override
    public void turnOn() {
        System.out.println("형광등을 켭니다");
    }

    @Override
    public void turnOff() {
        System.out.println("형광등을 끕니다");
    }

    @Override
    public void setVolume(int volumn) {
    }

    @Override
    public void setChannel(int channel) {
    }
}
```

Remocon(II)

- Main.JAVA

```
public static void main(String[] args) {
    RemoteControl tvRemocon = new Television();
    tvRemocon.turnOn();
    tvRemocon.setVolume(7);
    tvRemocon.setMute(true);
    RemoteControl.changeBattery();
    tvRemocon.turnOff();
    System.out.println();

    Audio audioRemocon = new Audio();
    audioRemocon.turnOn();
    audioRemocon.setVolume(7);
    audioRemocon.setMute(true);
    RemoteControl.changeBattery();
    audioRemocon.turnOff();
    System.out.println();
```

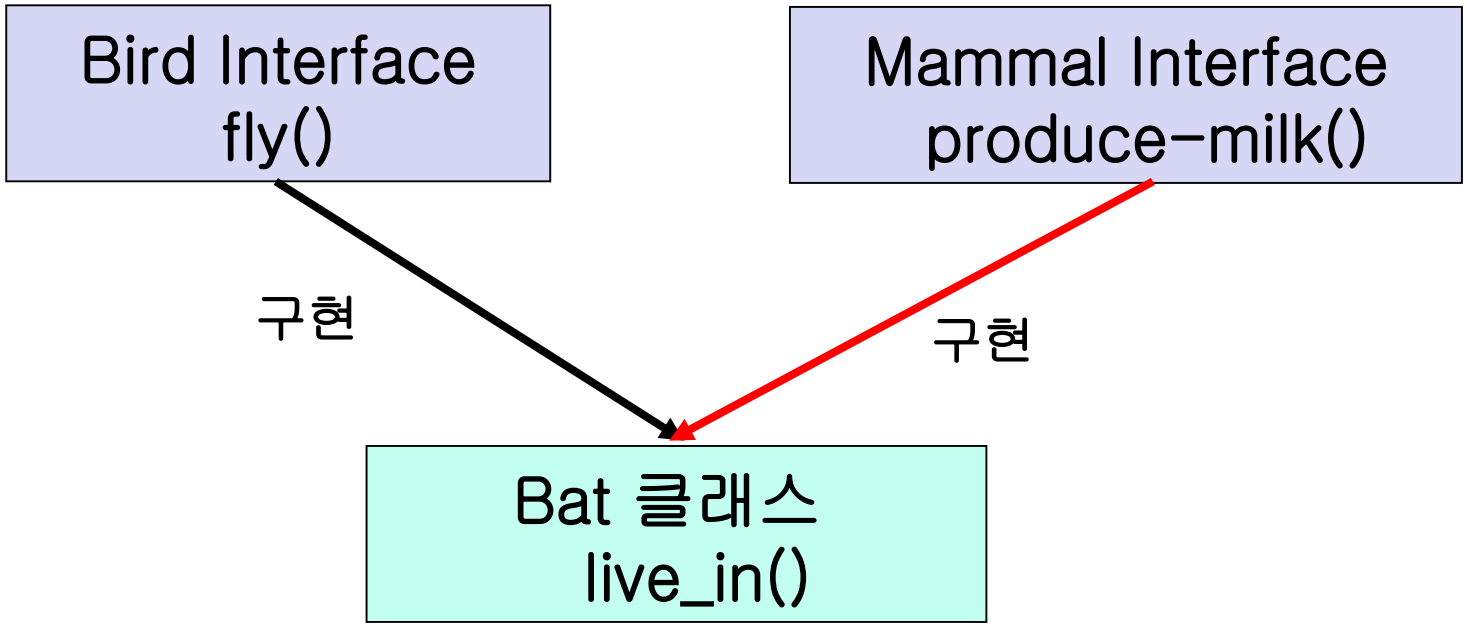
Remocon(II)

- Main.JAVA

```
    Lamp lamp = new Lamp();
    lamp.turnOn();
    lamp.turnOff();
}
```

Bat(박쥐)

- Bat(박쥐) 클래스는 Bird(새) Interface와 Mammal(포유 동물) Interface를 구현할 수 있음



운송수단

- 운송수단(자동차, 배, 비행기)의 속도를 나타내는 방법이 다양하다

차량번호 :
연료량 :
속도: Km/h

편명 :
속도: knot

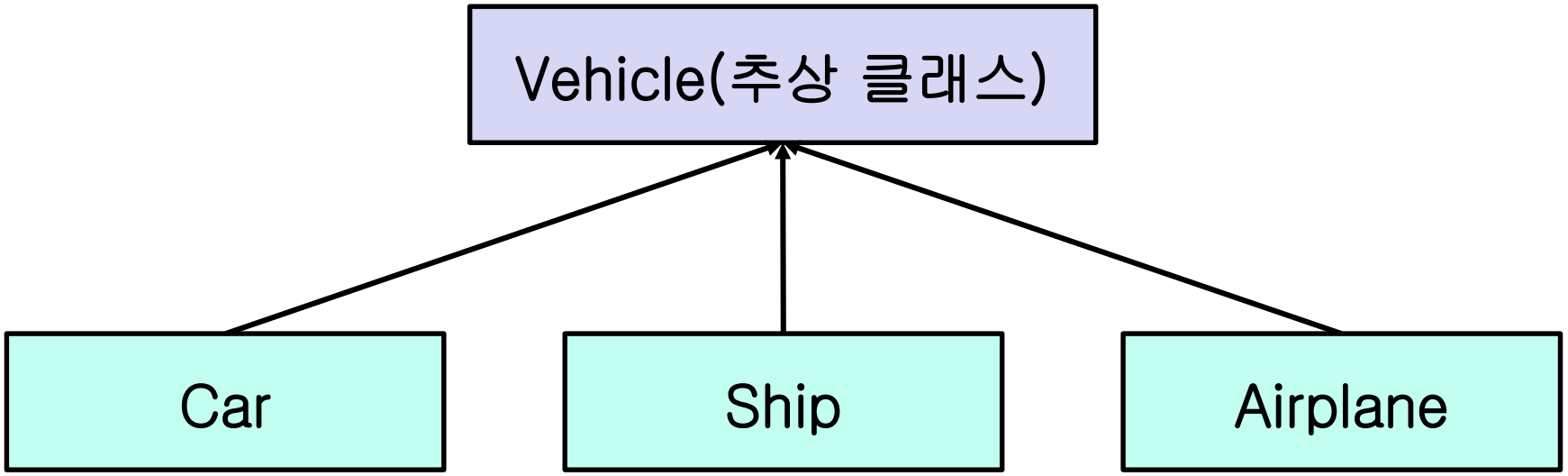
편명 :
속도: mile

운송수단(Abstract 클래스)

```
차량번호 85가3456, 연료양 20.5인 자동차가 만들어졌습니다.
속도를 60으로 변경했습니다.
비행기 번호가 KE 905인 비행기가 만들어졌습니다.
속도를 1500으로 변경했습니다.
배편 이름 퀸메리호인 배가 만들어졌습니다.
속도를 40으로 변경했습니다.
차량번호는 85가3456입니다.
연료 양은 20.5입니다.
속도는 60 Km/h 입니다.
비행기 번호는 KE 905입니다.
속도는 1500 Miles 입니다.
배 이름은 퀸메리호입니다.
속도는 40노트 입니다.
```

운송수단(Abstract 클래스)

- Class 구조도



운송수단(Abstract 클래스)

- Vehicle 추상 클래스

```
abstract public class Vehicle {
    private String name;
    protected int speed;

    public Vehicle(String name, int speed) {
        this.name = name;
        this.speed = speed;
    }

    public String getName() {
        return name;
    }
    public void setSpeed(int speed) {
        this.speed = speed;
        System.out.println("속도를 " + speed + "으로 변경했습니다.");
    }
    abstract void print();        //추상 메서드
}
```

운송수단(Abstract 클래스)

- Car 클래스

```
public class Car extends Vehicle {
    private double gas;

    public Car(String number, double gas, int speed) {
        super(number, speed);
        this.gas = gas;
        System.out.println("차량번호 " + number + ", 연료양 " + gas +
                                                    "인 자동차가 만들어졌습니다.");
    }

    @Override
    void print() {
        System.out.println("차량번호는 " + getName() + "입니다.");
        System.out.println("연료 양은 " + gas + "입니다.");
        System.out.println("속도는 " + speed + " Km/h 입니다.");
    }
}
```

운송수단(Abstract 클래스)

- Ship 클래스

```
public class Ship extends Vehicle {

    public Ship(String vessleName, int knot) {
        super(vessleName, knot);
        System.out.println("배편 이름 " + vessleName + "인 배가 만들어졌습니다.");
    }

    @Override
    public void print() { //추상 메서드 오버 라이딩
        System.out.println("배 이름은 " + getName() + "입니다.");
        System.out.println("속도는 " + speed + "노트 입니다.");
    }
}
```

운송수단(Abstract 클래스)

- Airplane 클래스

```
public class Airplane extends Vehicle {
    public Airplane(String flightNo, int mile) {
        super(flightNo, mile);
        System.out.println("비행기 번호가 " + flightNo +
                                                "인 비행기가 만들어졌습니다.");
    }

    @Override
    public void print() { //추상메서드 오버라이딩
        System.out.println("비행기 번호는 " + getName() + "입니다.");
        System.out.println("속도는 " + speed + " Miles 입니다.");
    }
}
```

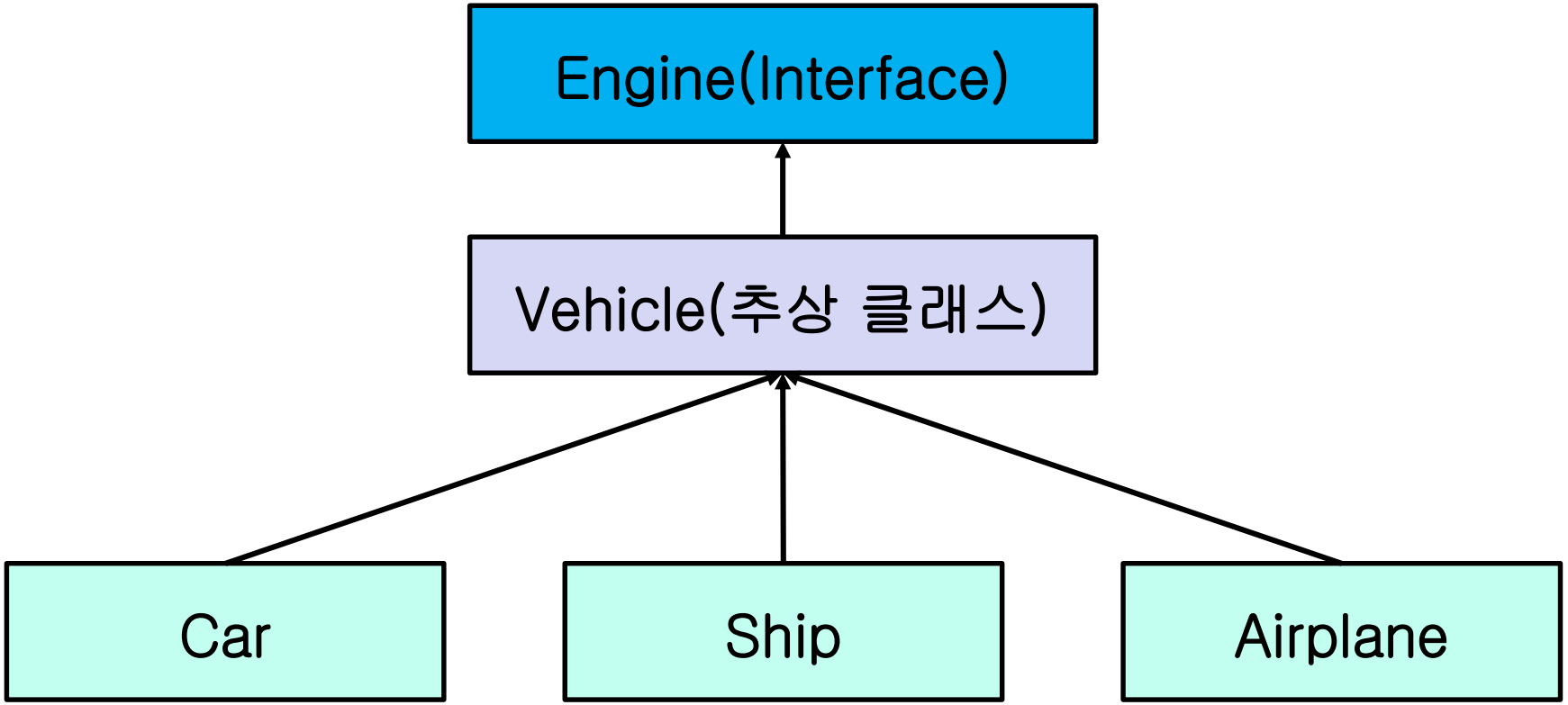
운송수단(Abstract 클래스)

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        Vehicle[] vehicles = new Vehicle[3];
        vehicles[0] = new Car("85가3456", 20.5, 0);
        vehicles[0].setSpeed(60);
        vehicles[1] = new Airplane("KE 905",232);
        vehicles[1].setSpeed(1500);
        vehicles[2] = new Ship("퀸메리호",12);
        vehicles[2].setSpeed(40);

        for(int i = 0; i < vehicles.length; i++){
            vehicles[i].print();
        }
    }
}
```

운송수단(Interface)

- Class 구조도



운송수단(Interface)

차량번호 85가3456, 연료양 20.5인 자동차가 만들어졌습니다.
속도를 60Km/h으로 변경했습니다.
비행기 번호가 KE 905인 비행기가 만들어졌습니다.
속도를 1500 Mile로 변경했습니다.
배편 이름 퀸메리호인 배가 만들어졌습니다.
속도를 40노트로 변경했습니다.
차량번호는 85가3456입니다.
연료 양은 20.5입니다.
속도는 60 Km/h 입니다.
비행기 번호는 KE 905입니다.
속도는 1500 Miles 입니다.
배 이름은 퀸메리호입니다.
속도는 40노트 입니다.

운송수단(Interface)

- Engine Interface (규칙)

```
public interface Engine {
    void print();          //추상 메소드
}
```

운송수단(Interface)

- Car 클래스

```
public class Vehicle implements Engine{
    private String name;
    protected int speed;

    public Vehicle(String name, int speed) {
        this.name = name;
        this.speed = speed;
    }

    public String getName() {
        return name;
    }
    public void setSpeed(int speed) {
        this.speed = speed;
    }
    @Override
    public void print() {
    }
}
```

운송수단(Interface)

- Car 클래스

```
public class Car extends Vehicle {
    private double gas;

    public Car(String number, double gas, int speed) {
        super(number, speed);
        this.gas = gas;
        System.out.println("차량번호 " + number + ", 연료양 " + gas +
                                                    "인 자동차가 만들어졌습니다.");
    }

    @Override
    public void setSpeed(int speed) {
        super.setSpeed(speed);
        System.out.println("속도를 " + speed + "Km/h으로 변경했습니다.");
    }
```

운송수단(Interface)

- Car 클래스

```
    @Override
    public void print() {
        System.out.println("차량번호는 " + getName() + "입니다.");
        System.out.println("연료 양은 " + gas + "입니다.");
        System.out.println("속도는 " + speed + " Km/h 입니다.");
    }
}
```

운송수단(Interface)

- Ship 클래스

```
public class Ship extends Vehicle {

    public Ship(String vessleName, int knot) {
        super(vessleName, knot);
        System.out.println("배편 이름 " + vessleName + "인 배가 만들어졌습니다.");
    }

    @Override
    public void setSpeed(int speed) {
        super.setSpeed(speed);
        System.out.println("속도를 " + speed + "노트로 변경했습니다.");
    }

    @Override
    public void print() { //추상메서드 오버라이딩
        System.out.println("배 이름은 " + getName() + "입니다.");
        System.out.println("속도는 " + speed + "노트 입니다.");
    }
}
```

운송수단(Interface)

- Airplane 클래스

```
public class Airplane extends Vehicle {

    public Airplane(String flightNo, int mile) {
        super(flightNo, mile);
        System.out.println("비행기 번호가 " + flightNo + "인 비행기가 만들어졌습니다.");
    }

    @Override
    public void setSpeed(int speed) {
        super.setSpeed(speed);
        System.out.println("속도를 " + speed + " Mile로 변경했습니다.");
    }

    @Override
    public void print() { //추상메서드 오버라이딩
        System.out.println("비행기 번호는 " + getName() + "입니다.");
        System.out.println("속도는 " + speed + " Miles 입니다.");
    }
}
```

SmartPhone

- 나의 SmartPhone은 MobilePhone의 특징인 음성 통화와 문자 Message를 주고 받을 수 있다
- MobilePhone은 MP3의 기능인 음악을 재생할 수 있다.
- MobilePhone은 PDA 기능인 계산과 일정 관리할 수 있다.

SmartPhone I

- Class 구조도

PDA(계산, 일정관리)

MobilePhone
(통화, 문자)

Mp3(음악)

Interface

Concrete Class

SmartPhone
(App 설치)

SmartPhone I

- MobilePhone Interface

```
public interface MobilePhone {
    public boolean sendCall();
    public boolean receiveCall();
    public boolean sendSMS();
    public boolean receiveSMS();
}
```

- MP3 Interface

```
public interface MP3 {
    public void play();
    public void stop();
}
```

SmartPhone I

- PDA Class

```
public class PDA {
    public int calculate(int x, int y) {
        return x + y;
    }

    public void scheduler() {
        System.out.println("일정 관리");
    }
}
```

SmartPhone I

- SmartPhone 클래스

```
public class SmartPhone extends PDA implements MobilePhone, MP3{
    @Override
    public void play() {
        System.out.println("음악 재생");
    }

    @Override
    public void stop() {
        System.out.println("재생 중지");
    }

    @Override
    public boolean sendCall() {
        System.out.println("전화 걸기");
        return true;
    }
```

SmartPhone I

- SmartPhone 클래스

```
    @Override
    public boolean receiveCall() {
        System.out.println("전화 받기");
        return true;
    }
    @Override
    public boolean sendSMS() {
        System.out.println("SMS 보내기");
        return true;
    }
    @Override
    public boolean receiveSMS() {
        System.out.println("SMS 받기");
        return true;
    }
    public void applicationManager() {
        System.out.println("어플리케이션 설치/삭제");
    }
}
```

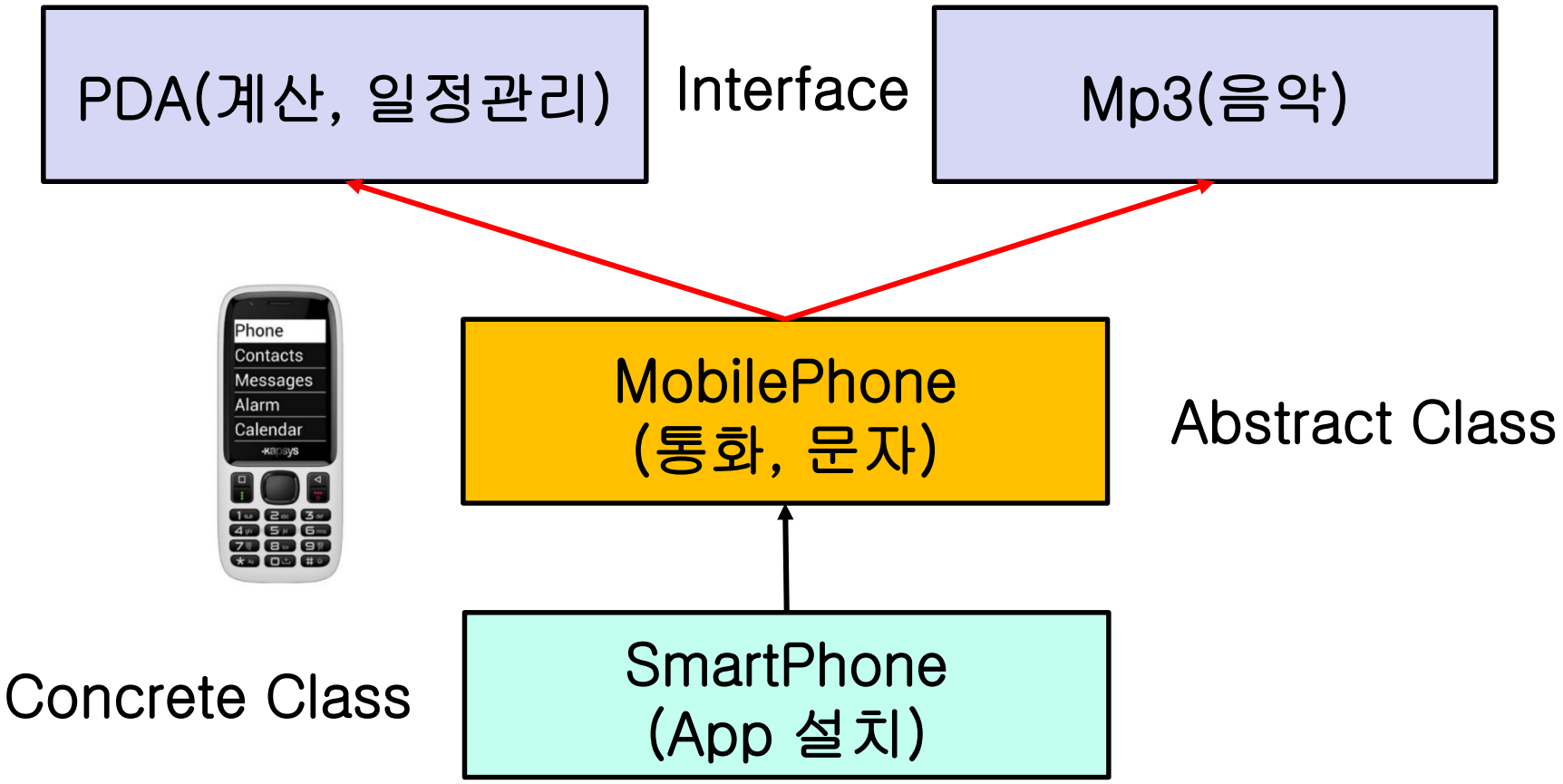
SmartPhone I

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        SmartPhone smartPhone = new SmartPhone();

        smartPhone.sendCall();
        smartPhone.play();
        System.out.println(smartPhone.calculate(3, 5));
        smartPhone.scheduler();
    }
}
```

SmartPhone II

- Class 구조도



SmartPhone II

- PDA 인터페이스

```
public interface PDA {
    public abstract void scheduler();
    public abstract int calculate(int x, int y);
}
```

SmartPhone II

- MP3 인터페이스

```
public interface MP3 {
    public void play();
    public void stop();
}
```

SmartPhone II

- MobilePhone 추상 클래스

```
public abstract class MobilePhone {
    abstract public boolean sendCall();

    public boolean receiveCall()  {
        System.out.println("전화 받기");
        return true;
    };

    abstract public boolean sendSMS();

    public boolean receiveSMS() {
        System.out.println("SMS 받기");
        return true;
    };
}
```

SmartPhone II

- SmartPhone 클래스

```
public class SmartPhone extends MobilePhone implements MP3, PDA {
    @Override
    public void play() {
        System.out.println("음악 재생");
    }

    @Override
    public void stop() {
        System.out.println("재생 중지");
    }

    @Override
    public boolean sendCall() {
        System.out.println("전화 걸기");
        return false;
    }
```

SmartPhone II

- SmartPhone 클래스

```
// MobilePhone의의 추상 메소드 구현
@Override
public boolean sendCall() {
    System.out.println("전화 걸기");
    return true;
}

@Override
public boolean sendSMS() {
    System.out.println("SMS 보내기");
    return true;
}
```

SmartPhone II

- SmartPhone 클래스

```
    @Override
    public int calculate(int x, int y) {
        return x + y;
    }

    public void scheduler() {
        System.out.println("일정 관리");
    }

    public void applicationManager() {
        System.out.println("어플리케이션 설치/삭제");
    }
}
```

SmartPhone II

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        SmartPhone myPhone = new SmartPhone();

        myPhone.sendCall();
        myPhone.play();
        System.out.println(myPhone.calculate(3, 5));
        myPhone.scheduler();
    }
}
```

VideoPlayer와 CDPlayer

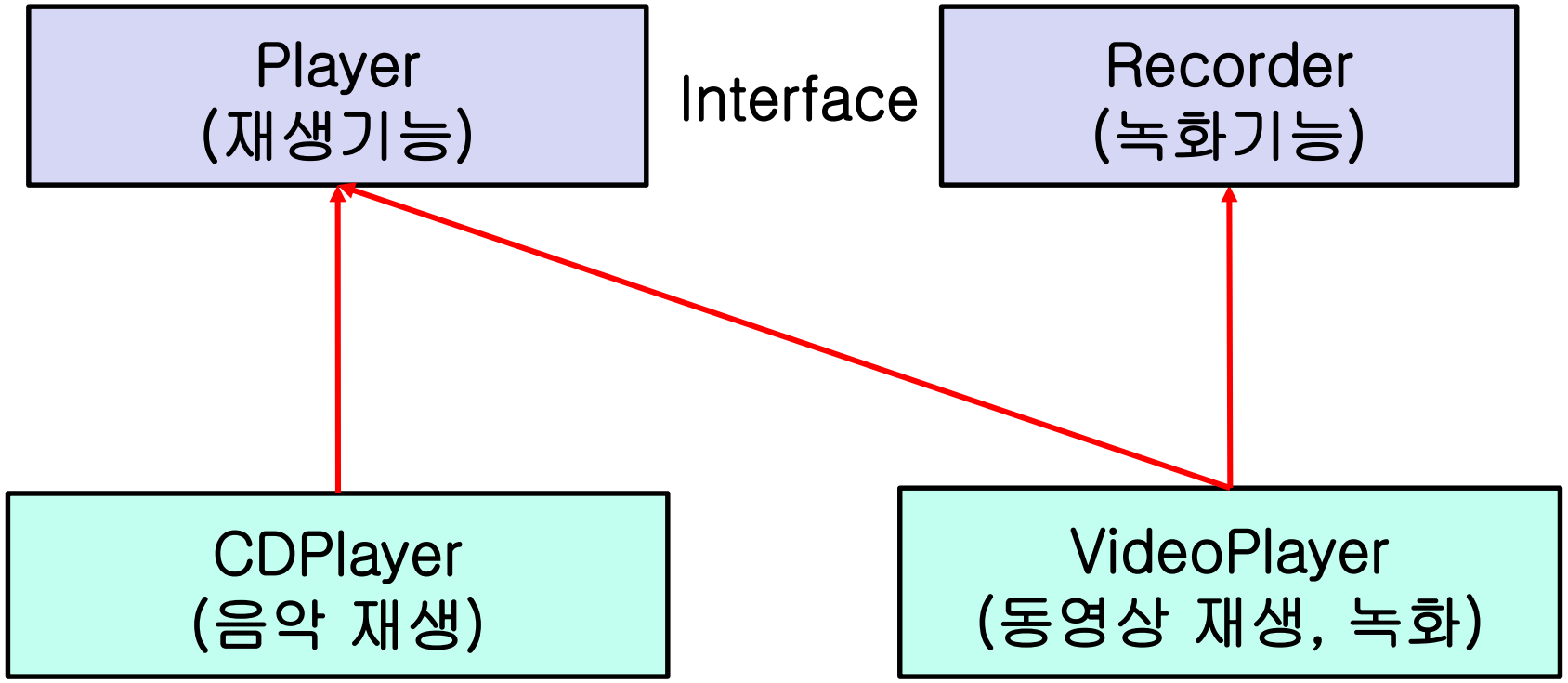
- 여러 학생분들은 알고 있나요 ?

VideoPlayer

CDPlayer

VideoPlayer와 CDPlayer

- Class로 구현해보자



VideoPlayer와 CDPlayer

- Player 인터페이스

```
public interface Player {
    public abstract void play();
    public abstract void stop();
    public abstract void info();
}
```

- Recorder 인터페이스

```
public interface Recorder {
    public abstract void record();
}
```

VideoPlayer와 CDPlayer

- CDPlayer 인터페이스

```
public class CDPlayer implements Player{
    private String maker;

    public CDPlayer(String maker) {
        this.maker = maker;
    }

    @Override
    public void play() {
        System.out.println("CD 재생시작!");
    }

    @Override
    public void stop() {
        System.out.println("CD 재생종료!");
    }
```

VideoPlayer와 CDPlayer

- CDPlayer 인터페이스

```
    @Override
    public void info() {
        System.out.println("제조회사 : " + maker);
    }

    public void cleaning() {
        System.out.println("헤드청소 완료!");
    }
}
```

VideoPlayer와 CDPlayer

- VideoPlayer 인터페이스

```
public class VideoPlayer implements Player, Recorder {
    private String maker;

    public VideoPlayer(String maker) {
        this.maker = maker;
    }

    @Override
    public void play() {
        System.out.println("비디오 재생시작!");
    }

    @Override
    public void stop() {
        System.out.println("비디오 재생종료!");
    }
```

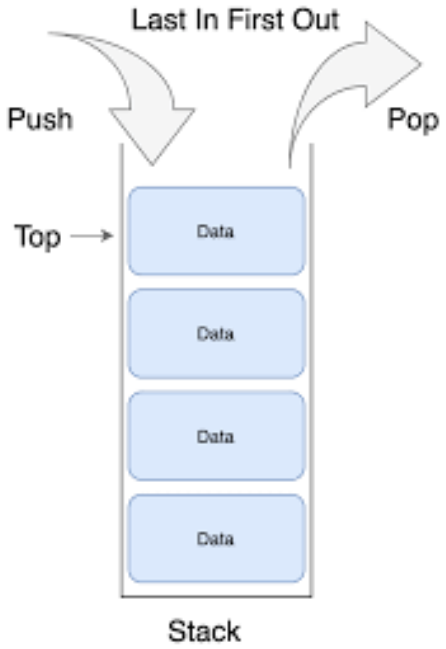
VideoPlayer와 CDPlayer

- VideoPlayer 인터페이스

```
    @Override
    public void record() {
        System.out.println("비디오 녹화시작!");
    }

    @Override
    public void info() {
        System.out.println("제조회사 : " + maker);
    }
}
```

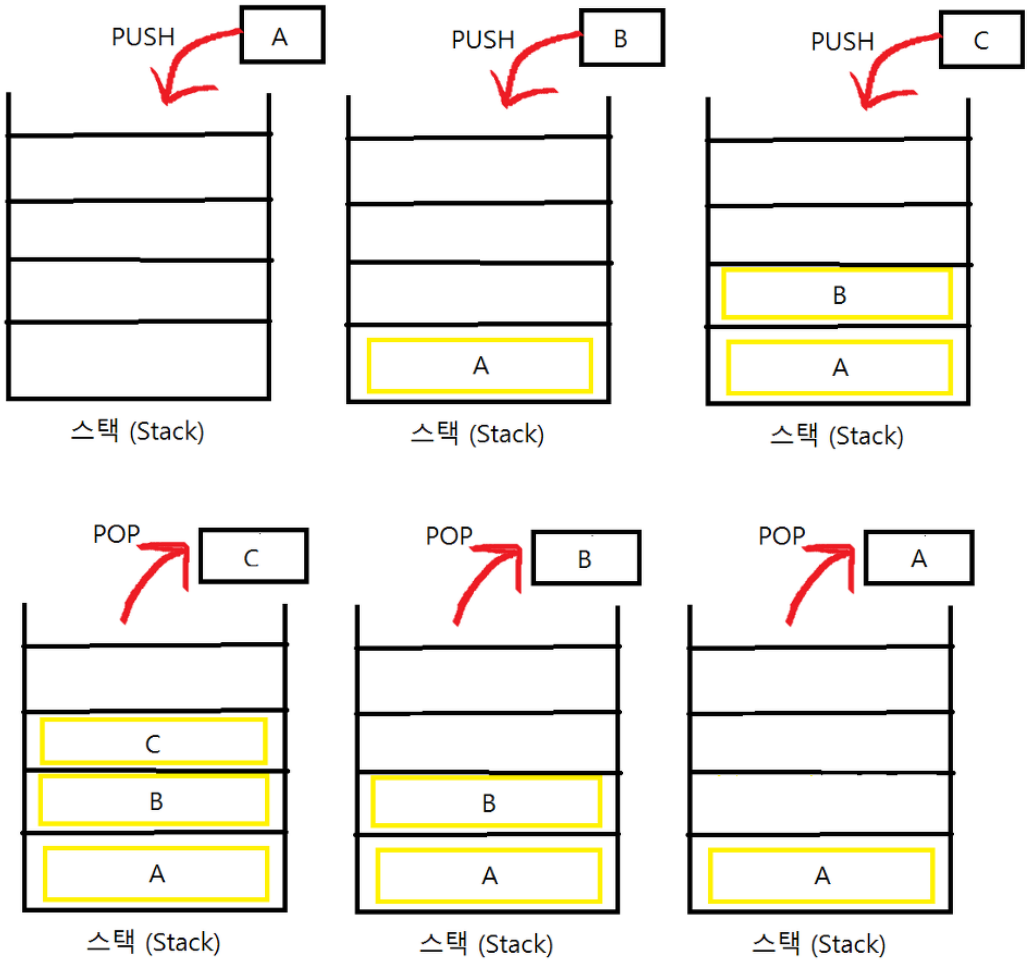
Stack 만들기

- Stack은 Data를 일시적으로 저장하기 위해 사용하는 자료 구조
- Data의 입력과 출력 순서는 후입선출(LIFO : Last In First Out)
- Programming 언어에서 함수를 호출하고 실행할 때 Program 내부에서는 Stack을 사용
  - push : Data를 넣는 작업
  - pop : Data를 꺼내는 작업
  - top : push, pop하는 위치



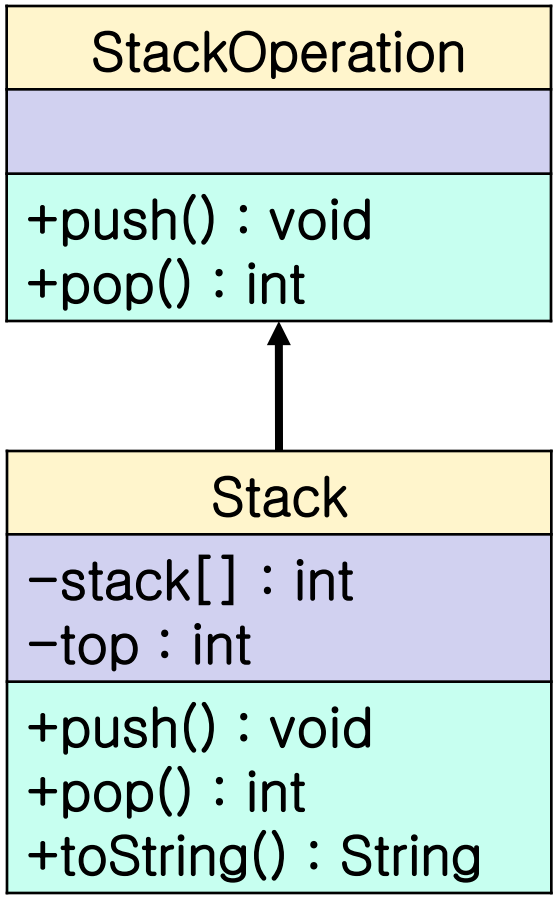
Stack 만들기

- Push, Pop



Stack 만들기

- Class Diagram



반드시 구현해야 함

Stack 만들기

- Interface StackOperation은 2개의 메소드 push()와 pop()을 선언하고, Stack 클래스는 Interface StackOperation을 상속 받아서 push()와 pop() 메소드를 오버라이딩 (Overriding) 함
- push()는 Stack에 정수형 Data를 입력하는 메소드, pop()은 Stack에 저장된 Data를 회수하는 메소드

Stack 만들기

- StackOperation 인터페이스

```
public interface StackOperation {
    public void push(int item);
    public int pop();
}
```

Stack 만들기

- Stack 클래스

```
public class Stack implements StackOperation{
    private int[] stack;
    private int top;

    public Stack(int size) {
        this.stack = new int[size];
        this.top = -1;
    }

    @Override
    public void push(int item) {
        if(top ==stack.length - 1)
            System.out.println("스택이 꽉 찼음");
        else
            stack[++top] = item;
    }
```

Stack 만들기

- Stack 클래스

```
    @Override
    public int pop() {
        if(top < 0) {
            System.out.println("스택이 비었음");
            return 0;
        } else
            return stack[top--];
    }

    public int size() {
        return stack.length;
    }
}
```

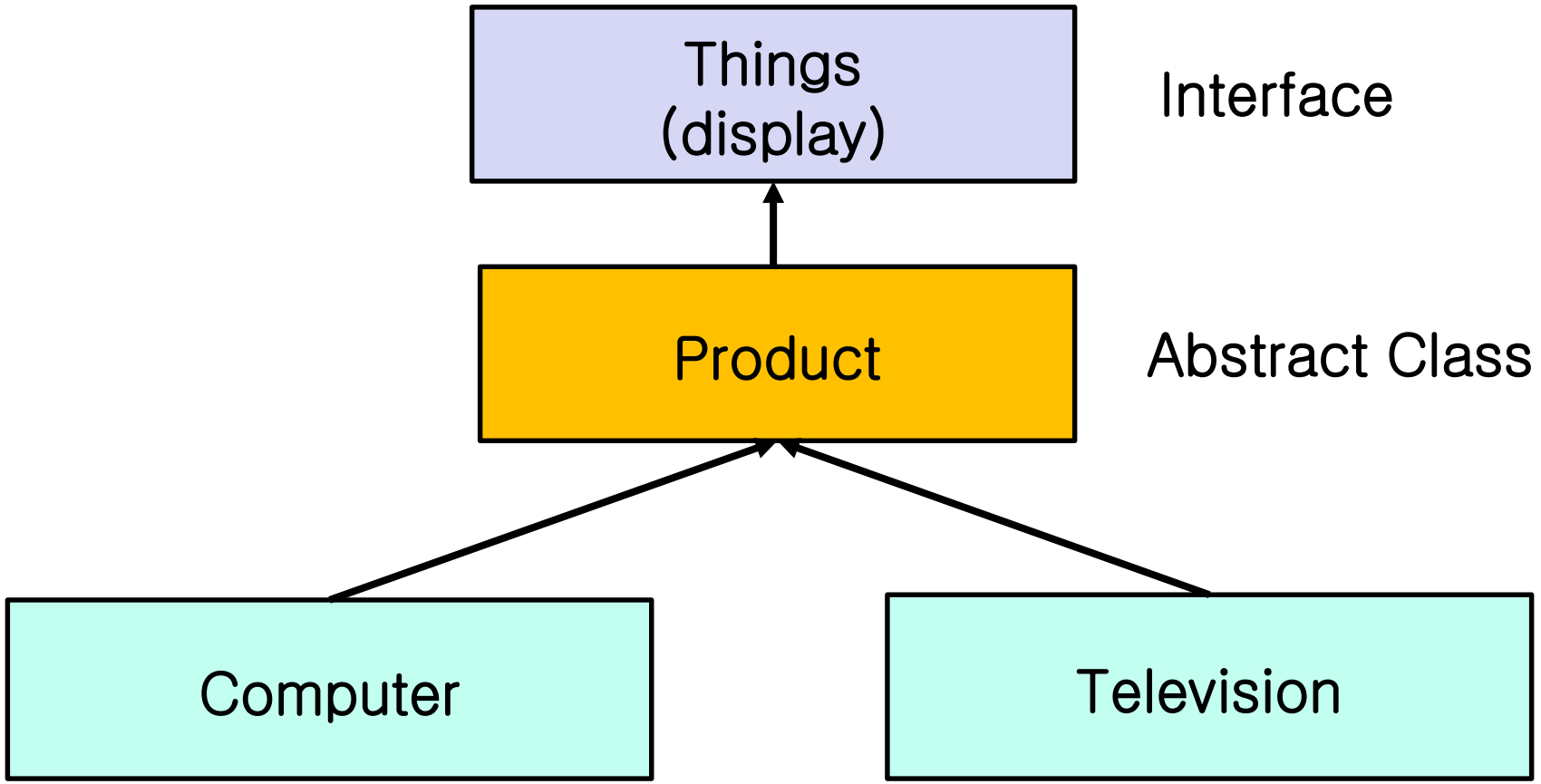
Stack 만들기

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        Random random = new Random();
        Stack stack = new Stack(10);

        System.out.print("스택 Push : ");
        for(int i = 0 ; i < stack.size() ; i++) {
            int value = random.nextInt(10);
            stack.push(value);
            System.out.print(value + "  ");
        }
        System.out.print("₩n스택 Pop  : ");
        for(int i = 0 ; i < stack.size(); i++)
            System.out.print(stack.pop() + "  ");
    }
}
```

Interface 예제(물건)

- 물건 Interface



Interface 예제(물건)

- Things Interface

```
public interface Things {
    public abstract void displayInfo();
}
```

Interface 예제(물건)

- Product 클래스

```
public abstract class Product implements Things {
    protected String company;
    protected  String name;
    protected int price;

    public Product() {
    }

    public Product(String company, String name, int price) {
        this.company = company;
        this.name = name;
        this.price = price;
    }
}
```

Interface 예제(물건)

- Television 클래스

```
public class Television extends Product {
    private int inchSize; // 화면 사이즈

    public Television(int inchSize) {
        super();
        this.inchSize = inchSize;
    }

    public Television(String company, String name, int price, int inchSize) {
        super(company, name, price);
        this.inchSize = inchSize;
    }

    @Override
    public void displayInfo() {
        System.out.println("TV size : " + inchSize + " | 제조사 : " + company +
                           " | 제품명 : " + name + " | 가격 : " + price);
    }
}
```

Interface 예제(물건)

- Computer 클래스

```
public class Computer extends Product {
    private int hddSize; // 하드디스크 사이즈
    private int memorySize; // 메모리 사이즈

    public Computer(int hddSize, int memorySize) {
        super();
        this.hddSize = hddSize;
        this.memorySize = memorySize;
    }

    public Computer(String company, String name, int price, int hddSize,
                                                        int memorySize) {
        super(company, name, price);
        this.hddSize = hddSize;
        this.memorySize = memorySize;
    }
```

Interface 예제(물건)

- Computer 클래스

```
    @Override
    public void displayInfo() {
        System.out.println("Computer 제조사 : " + company +
                " | 제품명 : " + name + " | 가격 : " + price +
                " | hddSize : " + hddSize + "GB | memorySize : " +
                memorySize + "GB");
    }
}
```

Interface 예제(물건)

- Main 클래스

```
public class Main {
    public static void main(String[] args) {
        Product[] products = {
                new Television("삼성", "파브", 2500000, 48),
                new Computer("삼성", "시리즈 9", 1800000, 500, 16)};

        for (int i = 0; i < products.length; i++){
            products[i].displayInfo();
        }
    }
}
```

Interface 예제(성적 1)

- Grade Interface

```
public interface Grade {
    int kor = 90;
    int eng = 80;
    int math = 70;
}

interface AddGrade extends Grade {
    int computer = 100;
    int math = 95;
}
```

Interface 예제(성적 1)

- Main.JAVA

```
public class Main implements AddGrade {
    public static void main(String[] args) {
        System.out.println("Kor = " + kor);
        System.out.println("Eng = " + eng);
        System.out.println("Computer = " + computer);
        System.out.println("Math = " + math);
        System.out.println("Grade_Math = " + Grade.math);
        System.out.println("AddGrade_Math = " + AddGrade.math);
    }
}
```

Interface 예제(성적 2)

- BaseGrade.JAVA

```
interface BaseGrade {
    int kor = 90;
    int eng = 80;
    int math = 70;

    void setComputer( int com );
    int getComputer();
}

abstract class addGrade implements BaseGrade {
    protected int com;

    public void setComputer( int com ) {
        this.com = com;
        System.out.println("setComputer method!!");
    }
}
```

Interface 예제(성적 2)

- Grade.JAVA

```
class Grade extends addGrade {
    public int getComputer() {
            System.out.println("getComputer method!!");
            return com;
    }
}
```

Interface 예제(성적 2)

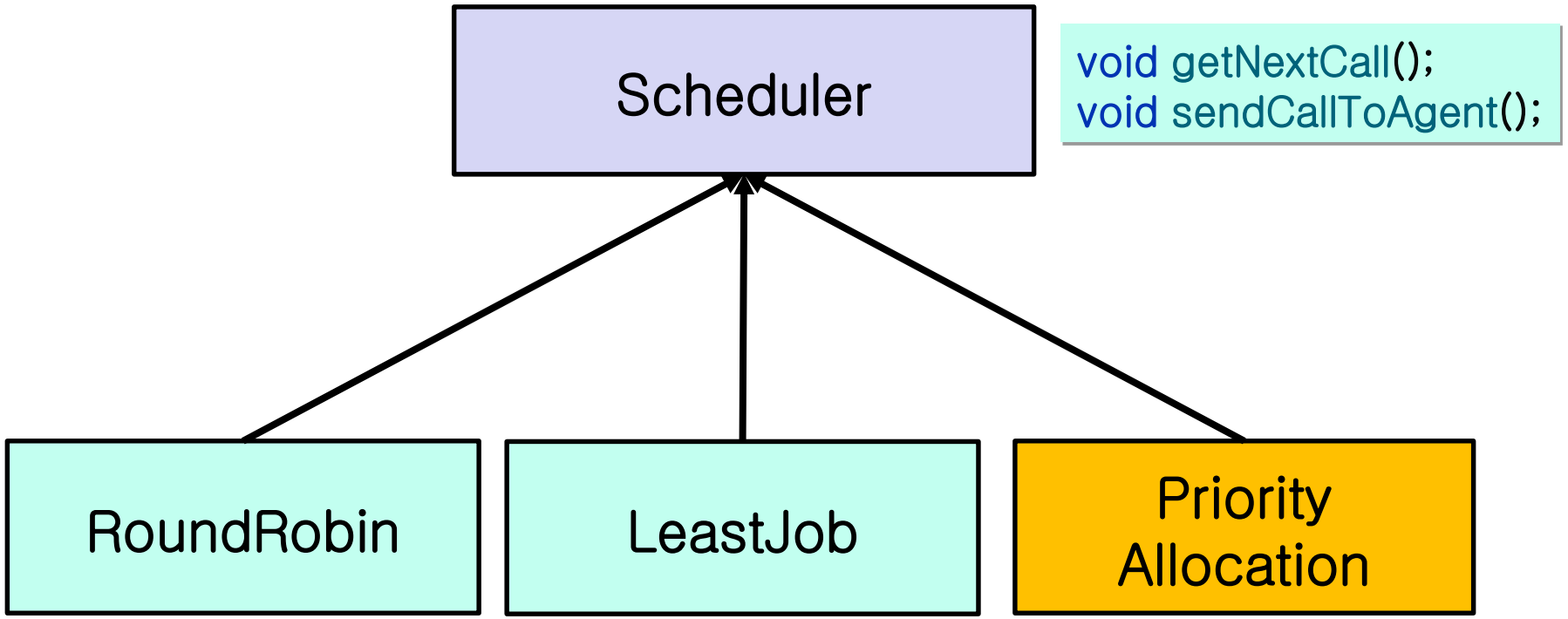
- Main.JAVA

```
public class Main {
    public static void main( String args[] ) {
            Grade gd = new Grade();
            gd.setComputer(100);
            int i = gd.getComputer();
                System.out.println("Main method getComputer = "+i);
    }
}
```

Service Center 상담원 연결

- Service Center에 전화가 오면 대기열에 저장된다. 상담원이 지정되기 전까지 대기상태가 된다.
- 각 전화를 상담원에게 배분하는 정책은 여러 방식으로 구현할 수 있다.
  - RoundRobin(순서대로 배분)
    - 모든 상담원이 동일한 상담 건수를 처리하도록 들어오는 전화 순서대로 배분
  - LeastJob(짧은 대기열 찾아 배분)
    - 고객 대기 시간을 줄이기 위해 상담을 하지 않는 상담원이나 가장 짧은 대기열을 보유한 상담원에게 배분
  - PriorityAllocation(우선순위에 따라 배분)
    - 고객 등급에 따라 등급이 높은 고객의 전화를 우선 가져와 업무 능력이 좋은 상담원에게 우선 배분

Service Center 상담원 연결



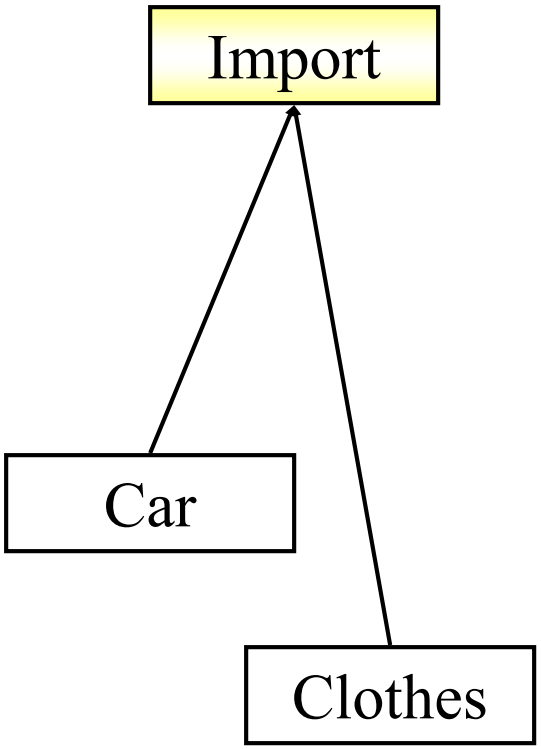
Interface 상속 실습

- 모든 차량은 차량 번호와 속도를 관리한다.
- 다음과 같은 Vehicle 인터페이스를 보고, Truck은 speed를 5씩 증감하고, SportsCar는 10씩 증감하도록 만들어보자

```
public interface Vehicle {
    void speedUp();
    void speedDown();
    abstract void stop();
}
```

  - 차량번호와 속도를 관리하는 Car 클래스를 만들어보자.
  - Truck 클래스와 SportsCar 클래스는 Vehicle 인터페이스와 Car 클래스를 활용하여 구현해보자

Interface Import




```
public interface Import {
        public abstract double calculate_tariff();
}
public class Car  implements Import {
        String make;
        String model;
        int year;
        int max_speed;
        int weight;
        float price;
        public abstract double calculate_tariff() {
                return price*0.1
        };
}
public class Clothes  implements Import {
        String make;
        String model;
        float price;
        public abstract double calculate_tariff() {
                return price*0.05
        };
}
```

Factory Method Pattern

- Factory Method Pattern은 Object 생성을 Sub Class에 위임하는 Design Pattern
- 즉, Object 생성 Code를 직접 작성하지 않고, “Factory Method"를 통해 Object를 생성하는 방식
- 이렇게 하면 Object 생성 과정이 Encapsulation화되어, Code가 유연하고 유지 보수하기 쉬워 짐

Factory Method Pattern

- 자동차 공장에서 자동차를 생산

```
interface Car {
    void drive();
}
```

```
class Sedan implements Car {
    public void drive() {
        System.out.println("세단을 운전합니다.");
    }
}
```

```
class SUV implements Car {
    public void drive() {
        System.out.println("SUV를 운전합니다.");
    }
}
```

Factory Method Pattern

```
abstract class CarFactory {
    abstract Car createCar();      // 팩토리 메소드(서브 클래스에서 구현)

    public void testDrive() {
        Car car = createCar();
        System.out.print("테스트 드라이브: ");
        car.drive();
    }
}
```

Factory Method Pattern

```
class SedanFactory extends CarFactory {
    @Override
    Car createCar() {
        return new Sedan();
    }
}
```

```
class SUVFactory extends CarFactory {
    @Override
    Car createCar() {
        return new SUV();
    }
}
```

Factory Method Pattern

```
public static void main(String[] args) {
        // 세단 공장에서 세단 생산
        CarFactory sedanFactory = new SedanFactory();
        sedanFactory.testDrive();  // 세단을 운전합니다.

        // SUV 공장에서 SUV 생산
        CarFactory suvFactory = new SUVFactory();
        suvFactory.testDrive();  // SUV를 운전합니다.
}
```

Factory Method Pattern

- Factory Method Pattern의 핵심 원칙
  - Object 생성을 Factory Method(createProduct())에서 담당
  - 상위 Class(Interface/Abstract Class)는 Object 생성 방식만 정의하고,실제 Object 생성은 하위 Class에서 담당
  - 새로운 Object 유형이 추가될 때, 기존 Code 수정 없이 확장 가능