

Basic of Java Programming

Java 프로그래밍 기초

추상클래스와 인터페이스
(Abstract & Interface)

학습목표



학/습/내/용

- 1 추상 메소드 /추상 클래스
- 2 interface



학/습/목/표

- 1 추상 메소드와 추상 클래스에 대하여 학습한다.
- 2 interface의 개념에 대하여 이해한다.

추상 메소드와 추상 클래스

- 추상 메소드(abstract method)

- 선언되어 있으나 구현되어 있지 않은 메소드, **abstract**로 선언

```
public abstract String getName();  
public abstract void setName(String s);
```

- 추상 메소드는 반드시 서브 클래스에서 오버라이딩하여 구현해야 함

- 추상 클래스(abstract class)의 2종류

1. 추상 메소드를 하나라도 가진 클래스

- 클래스 앞에 반드시 **abstract**라고 선언해야 함

2. 추상 메소드가 하나도 없지만 **abstract**로 선언된 클래스

2 가지 종류의 추상 클래스 사례

// 1. 추상 메소드를 포함하는 추상 클래스

```
abstract class Shape { // 추상 클래스 선언
    public Shape() { }
    public void paint() { draw(); }
    abstract public void draw(); // 추상 메소드
}
```

2 가지 종류의 추상 클래스 사례

// 2. 추상 메소드가 전혀 없는 추상 클래스

```
abstract class MyComponent { // 추상 클래스 선언
    String name;
    public void load(String name) {
        this.name = name;
    }
}
```

추상 클래스는 객체를 생성할 수 없다

```
abstract class Shape {
```

```
    ...
```

```
}
```

```
public class AbstractError {
```

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

```
        Shape shape;
```

```
        shape = new Shape(); // 컴파일 오류. 추상 클래스 Shape의 객체를  
        생성할 수 없다.
```

```
        ...
```

```
    }
```

```
}
```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
 Cannot instantiate the type Shape

at chap5.AbstractError.main(AbstractError.java:4)

추상 클래스의 상속

■ 추상 클래스의 상속 2 가지 경우(1)

1. 추상 클래스의 단순 상속

- 추상 클래스를 상속받아, 추상 메소드를 구현하지 않으면 추상 클래스 됨
- 서브 클래스도 abstract로 선언해야 함

```
abstract class Shape { // 추상 클래스
    public Shape() { }
    public void paint() { draw(); }
    abstract public void draw(); // 추상 메소드
}
abstract class Line extends Shape { // 추상 클래스.
                                   //draw()를 상속받기 때문
    public String toString() { return "Line"; }
}
```

추상 클래스의 상속

- 추상 클래스의 상속 2 가지 경우(2)

- 2. 추상 클래스 구현 상속

- 서브 클래스에서 슈퍼 클래스의 추상 메소드 구현(오버라이딩)
 - 서브 클래스는 추상 클래스 아님

```
abstract class Shape { // 추상 클래스
    public Shape() { }
    public void paint() { draw(); }
    abstract public void draw(); // 추상 메소드
}
class Line extends Shape { // 추상 클래스가 아님.
    @Override //draw()를 재정의(Override)하였음.
    public void draw()
    { System.out.println("Line을 그린다."); }
}
```


추상 클래스의 구현 및 활용 예

Line, Rect, Circle은 추상클래스 Shape를 상속받아 만든 서브 클래스들로서, draw()를 오버라이딩하여 구현한 사례입니다. 그러므로 Line, Rect, Circle은 추상 클래스가 아니며 이들의 인스턴스를 생성할 수 있습니다.



```
class Shape {  
    public void draw() {  
        System.out.println("Shape");  
    }  
}
```

추상 클래스로 수정

```
abstract class Shape {  
    public abstract void draw();  
}
```

```
class Line extends Shape {  
    @Override  
    public void draw() {  
        System.out.println("Line");  
    }  
}
```

```
class Rect extends Shape {  
    @Override  
    public void draw() {  
        System.out.println("Rect");  
    }  
}
```

```
class Circle extends Shape {  
    @Override  
    public void draw() {  
        System.out.println("Circle");  
    }  
}
```

draw()라고 하면 컴파일 오류가 발생.
추상 메소드 draw()를 구현하지 않았기 때문

추상 클래스의 용도

- 설계와 구현 분리
 - 슈퍼 클래스에서는 개념 정의
 - 서브 클래스마다 다른 구현이 필요한 메소드는 추상 메소드로 선언
 - 각 서브 클래스에서 구체적 행위 구현
 - 서브 클래스마다 목적에 맞게 추상 메소드를 다르게 구현
- 계층적 상속 관계를 갖는 클래스 구조를 만들 때

예제 1 : 추상 클래스의 구현 연습

- 다음 추상 클래스 Calculator를 상속받은 GoodCalc 클래스를 구현하시오.

```
abstract class Calculator {  
    public abstract int add(int a, int b);  
    public abstract int subtract(int a, int b);  
    public abstract double average(int[] a);  
}
```

예제 1정답

```
public class GoodCalc extends Calculator {  
    @Override  
    public int add(int a, int b) { // 추상 메소드 구현  
        return a + b;  
    }  
    @Override  
    public int subtract(int a, int b) { // 추상 메소드 구현  
        return a - b;  
    }  
    @Override  
    public double average(int[ ] a) { // 추상 메소드 구현  
        double sum = 0;  
        for (int i = 0; i < a.length; i++)  
            sum += a[ i ];  
        return sum/a.length;  
    }  
}
```

예제 1정답

```
public class CalcTestProg{  
    public static void main(String [ ] args) {  
        GoodCalc c = new GoodCalc();  
        System.out.println(c.add(2,3));  
        System.out.println(c.subtract(2,3));  
        System.out.println(c.average(new int [ ] { 2,3,4 }));  
    }  
}
```

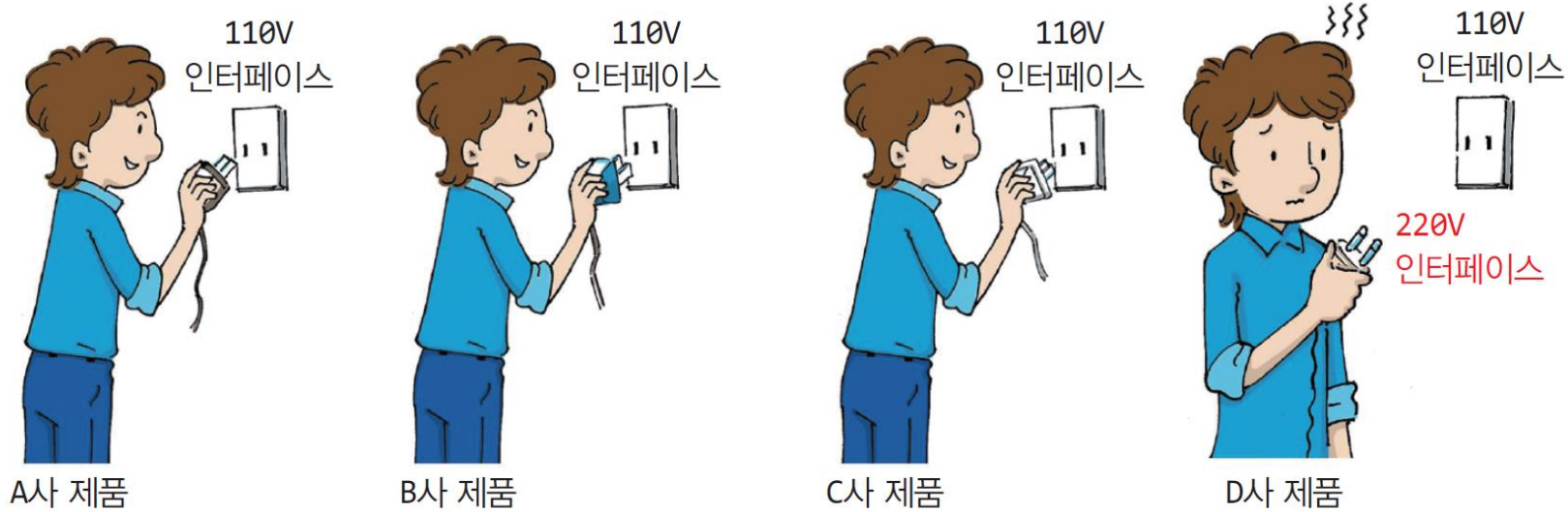
실행결과

5

-1

3.0

실세계의 인터페이스(interface)



정해진 규격(인터페이스)에
맞기만 하면 연결 가능.
각 회사마다 구현 방법은 다름

정해진
규격(인터페이스)에 맞지
않으면 연결 불가

- 인터페이스는 음식점의 메뉴판과 같은 것
 - 메뉴판을 보고 고객이 원하는 음식을 요청하게 되는데 메뉴판 자체가 음식을 주지는 않음.
 - 실제 음식은 주방이라는 곳에서 나오므로 메뉴판은 고객이 호출할 수 있는 서비스의 목록이라 할 수 있음.
 - 동일한 목적 하에 동일한 기능을 수행하게끔 강제하는 것
- JAVA에서 interface 란 클래스의 골격만 정의한 것.
- JAVA의 다형성을 극대화하여 개발코드의 수정을 줄이고 프로그램의 유지보수성을 높이기 위해 인터페이스를 사용

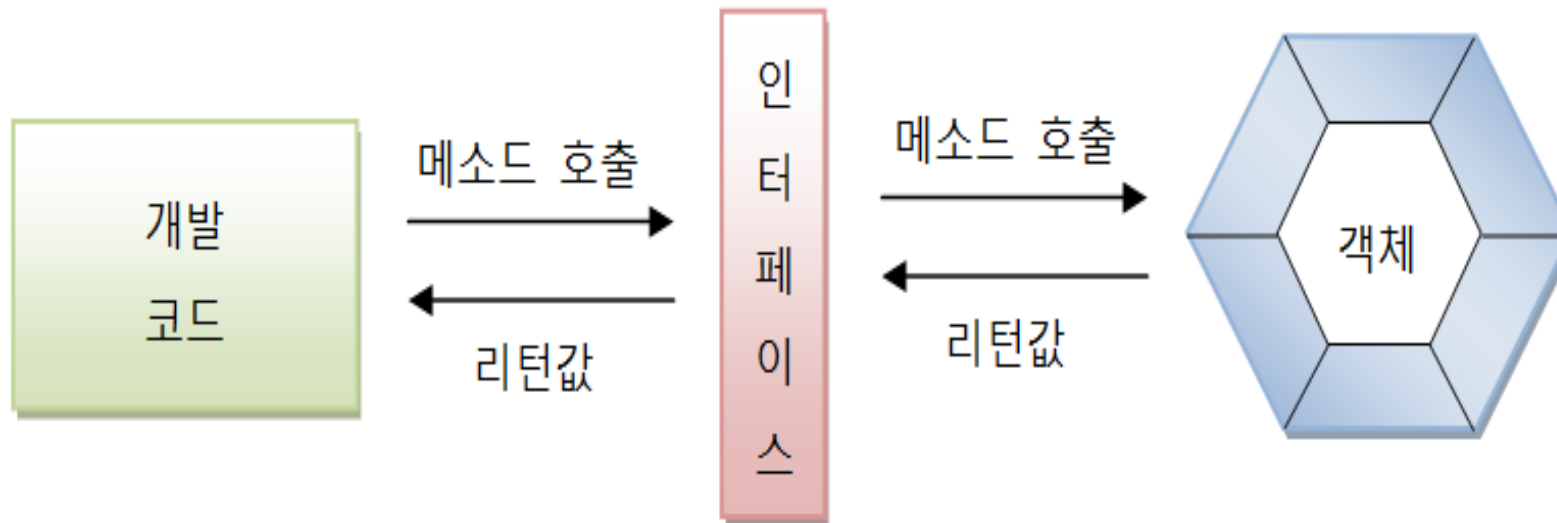
자바의 인터페이스(interface)

- 자바의 인터페이스(interface)
 - 클래스가 구현해야 할 메소드들이 선언되는 추상형
 - 인터페이스 선언
 - **interface** 키워드로 선언
 - Ex) public **interface** SerialDriver { ... }

```
[접근제한] interface [인터페이스명] {  
    상수;  
    접근_지정자 추상메서드(인자들);  
    default / static /private 메소드;  
}
```


■ 인터페이스란?

- 개발 코드와 객체가 서로 통신하는 접점 역할.
 - 개발 코드가 인터페이스의 메소드를 호출하면 인터페이스는 객체의 메소드를 호출 시킴.
 - 개발 코드는 객체의 내부 구조를 알 필요가 없고 **인터페이스의 메소드만 알고 있으면 OK**



자바의 인터페이스

- 자바 인터페이스에 대한 변화
 - Java 7까지
 - 인터페이스는 상수와 추상 메소드로만 구성
 - Java 8부터
 - 상수와 추상메소드 포함
 - **default** 메소드 포함 (Java 8)
 - **private** 메소드 포함 (Java 9)
 - **static** 메소드 포함 (Java 9)
- 여전히 인터페이스에는 필드(멤버 변수) 선언 불가

자바 인터페이스 사례

```
interface PhoneInterface { // 인터페이스 선언
    public static final int TIMEOUT = 10000; // 상수 필드 public static final 생략 가능
    public abstract void sendCall(); // 추상 메소드 public abstract 생략 가능
    public abstract void receiveCall(); // 추상 메소드 public abstract 생략 가능
    public default void printLogo() { // default 메소드 public 생략 가능
        System.out.println("** Phone **");
    }; // 디폴트 메소드
}
```

인터페이스의 구성 요소들의 특징

- 인터페이스의 구성 요소들(1)
 - 상수
 - public만 허용, **public static final** 은 생략 가능함.
 - 추상 메소드
 - **public abstract** 생략 가능
 - **default** 메소드
 - 인터페이스에 코드가 작성된 메소드
 - 인터페이스를 구현하는 클래스에 자동 상속
 - **public** 접근 지정만 허용. 생략 가능

인터페이스의 구성 요소들의 특징

- 인터페이스의 구성 요소들(2)

- **private 메소드**

- 인터페이스 내에 메소드 코드가 작성되어야 함
- 인터페이스 내에 있는 다른 메소드에 의해서만 호출 가능

- **static 메소드**

- public, private 모두 지정 가능. 생략하면 public

자바 인터페이스의 전체적인 특징

- 인터페이스의 객체 생성 불가



```
new PhoneInterface(); // 오류. 인터페이스 PhoneInterface 객체 생성 불가
```

- 인터페이스 타입의 레퍼런스 변수 선언 가능

```
PhoneInterface galaxy; // galaxy는 인터페이스에 대한 레퍼런스 변수
```

- 인터페이스 구현
 - 인터페이스를 상속받는 클래스는 인터페이스의 모든 추상 메소드 반드시 구현
- 다른 인터페이스 상속 가능
- 인터페이스의 다중 상속 가능

인터페이스 구현

- 인터페이스의 추상 메소드를 모두 구현하는 서브클래스 작성
 - **implements** 키워드 사용
 - 서브클래스에서는 여러 개의 인터페이스를 동시 구현 가능(다중상속)
- 인터페이스 구현 사례
 - PhoneInterface 인터페이스를 구현한 SamsungPhone 클래스

```
Class SamsungPhone implements PhoneInterface { // 인터페이스 구현
    // PhoneInterface의 모든 메소드 구현
    public void sendCall() { System.out.println("띠리리리링"); }
    public void receiveCall() { System.out.println("전화가 왔습니다."); }

    // 메소드 추가 작성
    public void flash() { System.out.println("전화기에 불이 켜졌습니다."); }
}
```

- SamsungPhone 클래스는 PhoneInterface의 default 메소드를 상속받음.

예제 2. 인터페이스 구현

(문제) PhoneInterface
인터페이스를 구현하고
flash() 메소드를
추가한 SamsungPhone
클래스를 작성하시오.

```
interface PhoneInterface { // 인터페이스 선언
    final int TIMEOUT = 10000; // 상수 필드 선언

    void sendCall(); // 추상 메소드
    void receiveCall(); // 추상 메소드

    default void printLogo() { // default 메소드
        System.out.println("** Phone **");
    }
}
```


예제 2. 인터페이스 구현

(문제) PhoneInterface
인터페이스를 구현하고
flash() 메소드를
추가한 SamsungPhone
클래스를 작성하시오.

```
class SamsungPhone implements
PhoneInterface { // 인터페이스 구현
    // PhoneInterface의 모든 추상 메소드 구현
    @Override
    public void sendCall() {
        System.out.println("띠리리리링");
    }
    @Override
    public void receiveCall() {
        System.out.println("전화가 왔습니다.");
    }

    // 메소드 추가 작성
    public void flash() { System.out.println("전화기에
불이 켜졌습니다."); }
}
```

예제 2. 인터페이스 구현

실행결과

**** Phone ****
띠리리리링
전화가 왔습니다.
전화기에 불이
켜졌습니다.

```
public class InterfaceEx {  
    public static void main(String[] args) {  
        SamsungPhone phone = new  
        SamsungPhone();  
        phone.printLogo();  
        phone.sendCall();  
        phone.receiveCall();  
        phone.flash();  
    }  
}
```

인터페이스간의 상속

- 인터페이스가 다른 인터페이스를 상속

✓ **extends** 키워드 이용

```
interface MobilePhoneInterface extends PhoneInterface {  
    void sendSMS();    // 새로운 추상 메소드 추가  
    void receiveSMS(); // 새로운 추상 메소드 추가  
}
```

인터페이스 상속

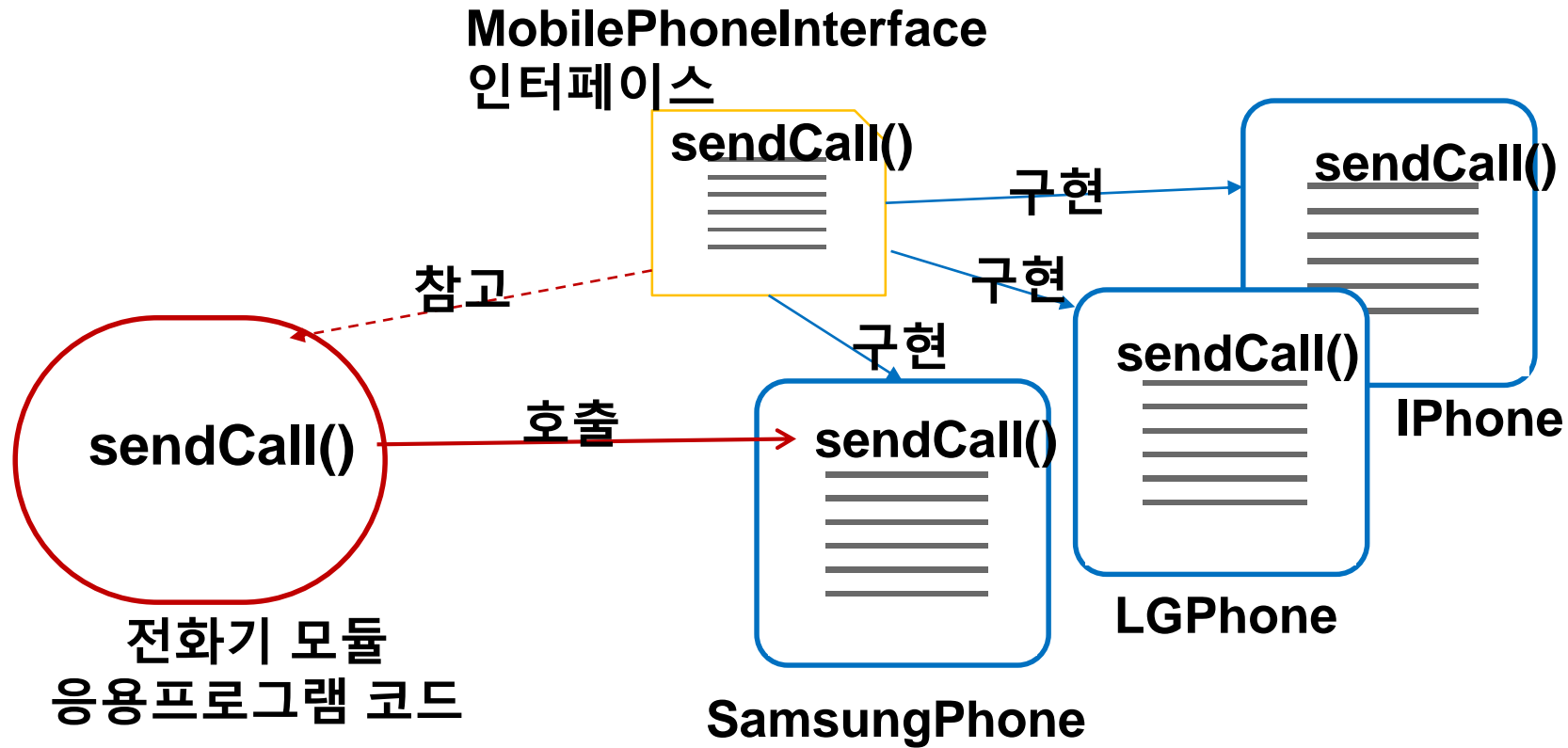
- 인터페이스가 다른 인터페이스 상속
 - 다중 인터페이스 상속

```
interface MP3Interface {  
    void play(); // 추상 메소드  
    void stop(); // 추상 메소드  
}
```

```
interface MusicPhoneInterface extends MobilePhoneInterface,  
                                     MP3Interface {  
    void playMP3RingTone(); // 새로운 추상 메소드 추가  
}
```

인터페이스의 목적

- ✓ 인터페이스는 스펙(명세서)을 주어 클래스들이 그 기능을 서로 다르게 구현할 수 있도록 하는 클래스의 규격 선언
- ✓ 클래스의 다형성을 실현하는 도구



다중 인터페이스 구현

- ✓ 클래스는 하나 이상의 인터페이스를 구현 가능
 - ✓ 다중 상속의 개념

```
interface AllInterface {  
    void recognizeSpeech(); // 음성 인식  
    void synthesizeSpeech(); // 음성 합성  
}
```

```
class AllPhone implements MobilePhoneInterface, AllInterface { // 인터페이스 구현
```

```
    // MobilePhoneInterface의 모든 메소드를 구현한다.
```

```
    public void sendCall() { ... }  
    public void receiveCall() { ... }  
    public void sendSMS() { ... }  
    public void receiveSMS() { ... }
```

```
    // AllInterface의 모든 메소드를 구현한다.
```

```
    public void recognizeSpeech() { ... } // 음성 인식  
    public void synthesizeSpeech() { ... } // 음성 합성
```

```
    // 추가적으로 다른 메소드를 작성할 수 있다.
```

```
    public int touch() { ... }
```

```
}
```

클래스에서 인터페이스의 메소드를
구현할 때
public을 생략하면 오류 발생

예제 3 : 인터페이스를 구현하고 동시에 클래스를 상속받는 사례

```
interface PhoneInterface { // 인터페이스 선언
    final int TIMEOUT = 10000; // 상수 필드 선언
    void sendCall(); // 추상 메소드
    void receiveCall(); // 추상 메소드
    default void printLogo() { // default 메소드
        System.out.println("** Phone **");
    }
}

interface MobilePhoneInterface extends PhoneInterface {
    void sendSMS();
    void receiveSMS();
}

interface MP3Interface { // 인터페이스 선언
    public void play();
    public void stop();
}

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

예제 3 : 인터페이스를 구현하고 동시에 클래스를 상속받는 사례

```
// SmartPhone 클래스는 PDA를 상속받고,  
// MobilePhoneInterface와 MP3Interface 인터페이스에 선언된  
추상 메소드를 모두 구현한다.  
class SmartPhone extends PDA implements  
MobilePhoneInterface, MP3Interface {  
    // MobilePhoneInterface의 추상 메소드 구현  
    @Override  
    public void sendCall() {  
        System.out.println("따르릉따르릉~~");  
    }  
    @Override  
    public void receiveCall() {  
        System.out.println("전화 왔어요.");  
    }  
}
```


예제 3 : 인터페이스를 구현하고 동시에 클래스를 상속받는 사례

```
@Override
public void sendSMS() {

    System.out.println("문자갑니다.");
}
@Override
public void receiveSMS() {

    System.out.println("문자왔어요.");
}
// MP3Interface의 추상 메소드 구현
@Override
public void play() {
    System.out.println("음악
연주합니다.");
}
```

```
@Override
public void stop() {
    System.out.println("음악
중단합니다.");
}
// 추가로 작성한 메소드
public void schedule() {
    System.out.println("일정
관리합니다.");
}
}
```

예제 3 : 인터페이스를 구현하고 동시에 클래스를 상속받는 사례

```
public class InterfaceEx {  
    public static void main(String [] args) {  
        SmartPhone phone = new  
            SmartPhone();  
  
        phone.printLogo();  
        phone.sendCall();  
        phone.play();  
        System.out.println("3과 5를 더하면 " +  
            phone.calculate(3,5));  
        phone.schedule();  
    }  
}
```

실행결과

```
** Phone **  
따르릉따르릉~~  
음악 연주합니다.  
3과 5를 더하면 8  
일정 관리합니다.
```

추상 클래스와 인터페이스 비교

- 유사점

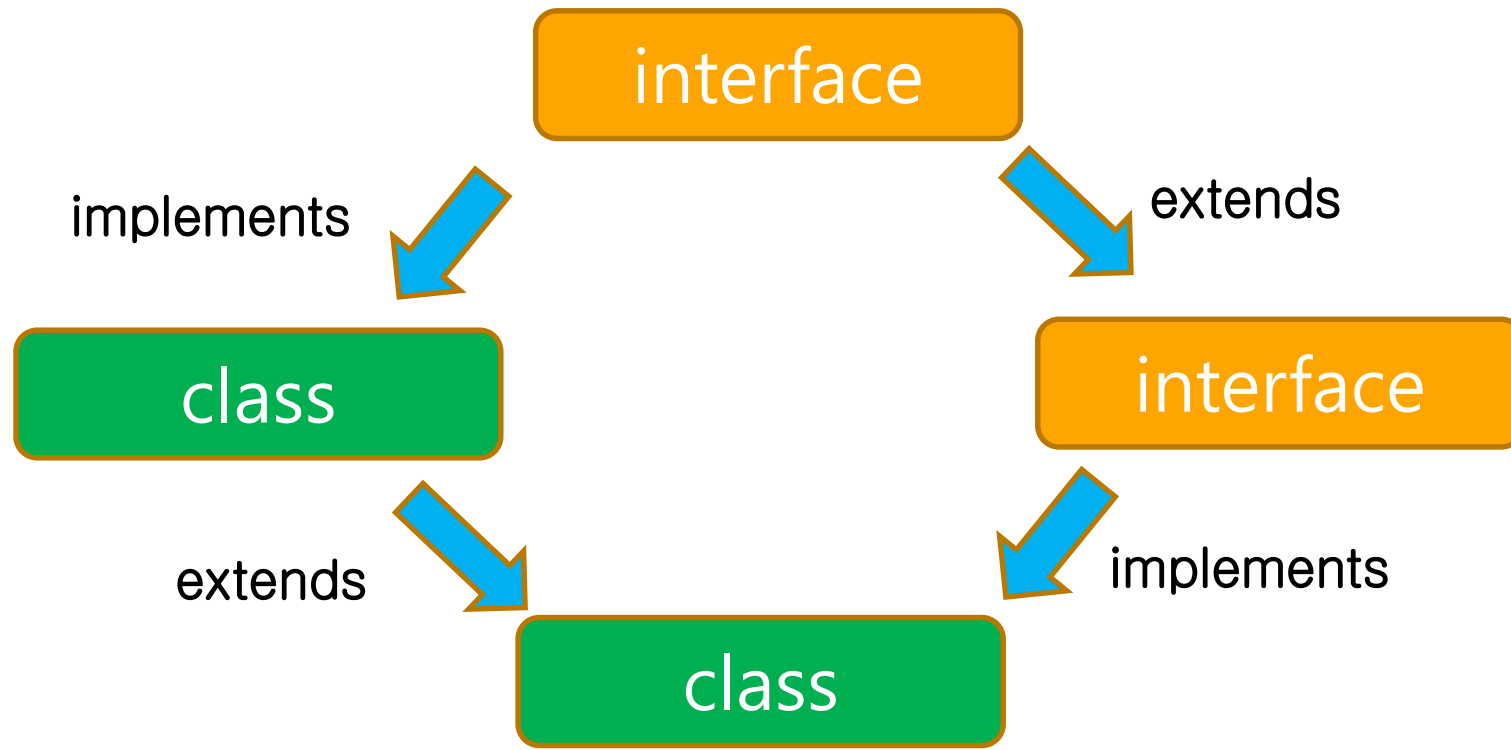
- 객체를 생성할 수 없고, 상속을 위한 슈퍼 클래스로만 사용
- 모든 추상 메서드는 객체 생성을 위한 서브 클래스에서 반드시 구현되어야 함.
- 업캐스팅(Upcasting) 이 가능함.
- 클래스의 다형성을 실현하기 위한 목적

추상 클래스와 인터페이스 비교

■ 다른 점

비교	목적	구성
추상 클래스	추상 클래스는 서브 클래스에서 필요로 하는 대부분의 기능을 구현하여 두고 서브 클래스가 상속받아 활용할 수 있도록 하되, 서브 클래스에서 구현할 수밖에 없는 기능만을 추상 메소드로 선언하여, 서브 클래스에서 구현하도록 하는 목적(다형성)	<ul style="list-style-type: none">• 추상 메소드와 일반 메소드 모두 포함• 상수, 변수 필드 모두 포함
인터페이스	인터페이스는 객체의 기능을 모두 공개한 표준화 문서와 같은 것으로, 개발자에게 인터페이스를 상속받는 클래스의 목적에 따라 인터페이스의 모든 추상 메소드를 만들도록 하는 목적(다형성)	<ul style="list-style-type: none">• 변수 필드(멤버 변수)는 포함하지 않음• 상수, 추상 메소드, 일반 메소드, default 메소드, static 메소드 모두 포함• protected 접근 지정 선언 불가• 다중 상속 지원

인터페이스와 클래스간의 상속



<인터페이스와 클래스간의 상속 다이어그램>

핵심정리



- 추상 메소드(abstract method)
 - 선언은 되어 있으나 구현되어 있지 않은 메소드
 - **abstract**로 선언
 - 추상 메소드는 반드시 서브 클래스에서 오버라이딩하여 구현해야 함
- 추상 클래스(abstract class)의 2종류
 1. 추상 메소드를 하나라도 가진 클래스
 - 클래스 앞에 반드시 **abstract**라고 선언해야 함
 2. 추상 메소드가 하나도 없지만 **abstract**로 선언된 클래스
- JAVA의 인터페이스(interface)
 - 클래스가 구현해야 할 메소드들이 선언되는 추상형