

## 7장 인터페이스와 특수 클래스(1)

# 목차

- 추상 클래스
- 인터페이스 기본
- 인터페이스 응용
- 인터페이스와 다형성
- 중첩 클래스와 중첩 인터페이스
- 익명 클래스

# 상속의 편리한 사례

class Student

말하기  
먹기  
걷기  
잠자기  
공부하기

class StudentWorker

말하기  
먹기  
걷기  
잠자기  
공부하기  
일하기

class Researcher

말하기  
먹기  
걷기  
잠자기  
연구하기

class Professor

말하기  
먹기  
걷기  
잠자기  
연구하기  
가르치기

상속이 없는 경우  
중복된 멤버를 가진  
4 개의 클래스

class Person

말하기  
먹기  
걷기  
잠자기

공통 기능을 Person  
클래스로 작성

상속

class Student

공부하기

class Researcher

연구하기

class StudentWorker

일하기

class Professor

가르치기

상속을 이용한  
경우 중복이 제거되고  
간결해진 클래스 구조

# 추상 메소드와 추상 클래스

## □ 추상 메소드(abstract method)

- 선언되어 있으나 구현되어 있지 않은 메소드, abstract로 선언

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

- 추상 메소드는 서브 클래스에서 오버라이딩하여 구현해야 함

## □ 추상 클래스(abstract class)의 2종류

1. 추상 메소드를 하나라도 가진 클래스
  - 클래스 앞에 반드시 abstract라고 선언해야 함
2. 추상 메소드가 하나도 없지만 abstract로 선언된 클래스

## 2 가지 종류의 추상 클래스 사례

### 추상 클래스(**abstract class**)의 2종류

1. 추상 메소드를 하나라도 가진 클래스  
클래스 앞에 반드시 **abstract**라고 선언해야 함

```
// 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. 추상 메소드가 하나도 없지만 **abstract**로 선언된 클래스

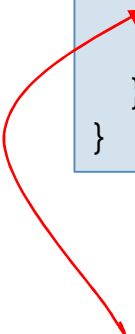
```
// 2. 추상 메소드 없는 추상 클래스

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

# 추상 클래스는 객체를 생성할 수 없다

추상 클래스에는 실행 코드가 없는 **미완성 상태인 추상 메소드가 있을 수 있기 때문에** 추상 클래스의 객체를 생성하는 코드에는 **컴파일 오류가 발생한다.**

```
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. 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"; }
}
```

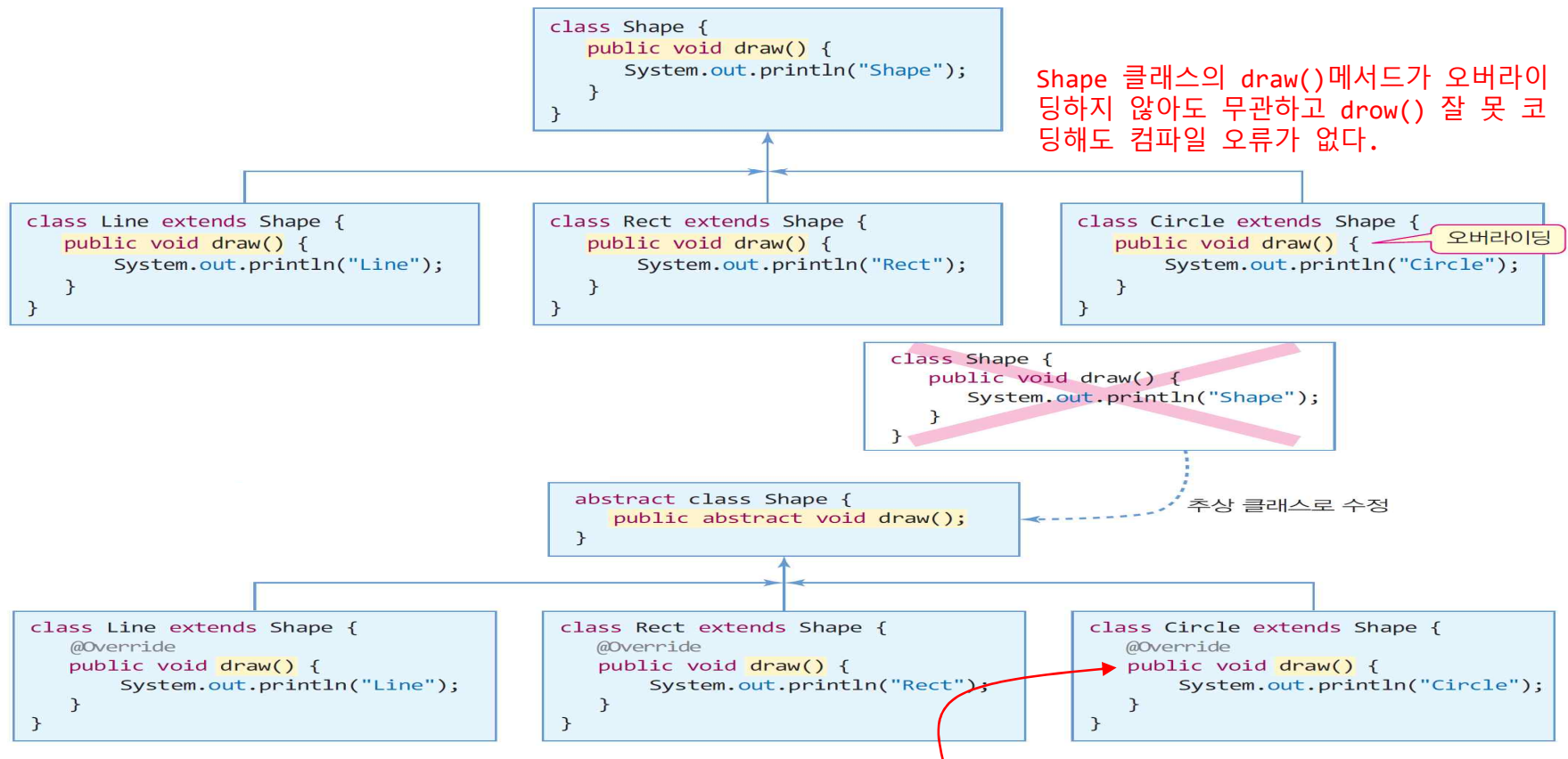
### 2. 추상 클래스 구현 상속

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

# 추상 클래스의 구현 및 과 목적

추상 클래스의 구현은 슈퍼 클래스에 선언된 모든 추상 메소드를 서브 클래스에서 오버라이딩하여 실행 가능한 코드로 구현하는 것을 말함.

추상 클래스의 목적은 추상 메소드를 통해 서브 클래스가 구현할 메소드를 명료하게 알려주는 인터페이스 역할을 하고, 서브 클래스는 추상 메소드를 목적에 맞게 구현하는 다형성을 실현 할 수 있다.



Shape 추상 메소드 draw()를 꼭 오버라이딩 해야 하고 draw()라고 하면 컴파일 오류가 발생.



# 추상 클래스의 용도

## □ 설계와 구현 분리

- ▣ 추상 클래스를 이용하면 응용 프로그램의 설계와 구현을 분리 할 수 있다.

- 추상 클래스에서 기본 방향을 잡아 놓고 서브 클래스에서 구현하면 구현 작업이 수월하다.(추상 클래스는 책의 목차로 비유하면 서브 클래스는 목차에 따라 작성된 책과 같다.)

- ▣ 슈퍼 클래스에서는 개념 정의

- 서브 클래스마다 다른 구현이 필요한 메소드는 추상 메소드로 선언

- ▣ 각 서브 클래스에서 구체적 행위 구현

- 서브 클래스마다 목적에 맞게 추상 메소드 다르게 구현

## □ 계층적 상속 관계를 갖는 클래스 구조를 만들 때

# 추상 클래스

## ■ 추상 메서드

- 메서드 본체를 완성하지 못한 메서드. 무엇을 할지는 선언할 수 있지만, 어떻게 할지는 정의할 수 없음
- 추상 메서드를 포함하는 클래스는 인스턴스를 생성 할 수 없으므로 추상 클래스 이다.(이유:미완성메서드)

## ■ 추상 클래스

- 보통 하나 이상의 추상 메서드를 포함하지만 없을 수도 있음
- 추상 클래스는 인스턴스를 생성 할 수 없고 오직 상속을 통한 자식 클래스를 구현한 후에 인스턴스를 생성 할수 있다.
- 주로 상속 계층에서 자식 멤버의 이름을 통일하기 위하여 사용
- 추상 클래스는 단독으로 사용될 수 없지만, 새로운 클래스를 작성하는데 밑바탕이 될 수 있다.

```
추상클래스 s = new 추상클래스(); // 추상 클래스는 인스턴스를 생성하지 못한다.
```

## ■ 추상 클래스의 구현 및 과 목적

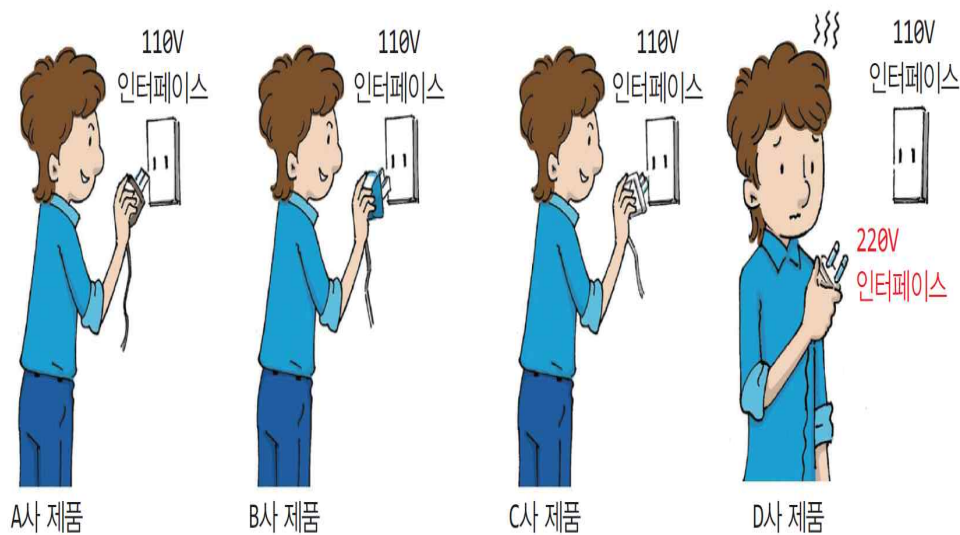
- 추상 클래스의 구현은 슈퍼 클래스에 선언된 모든 추상 메소드를 서브 클래스에서 오버라이딩하여 실행 가능한 코드로 구현하는 것을 말함.
- 추상 클래스의 목적은 추상 메소드를 통해 서브 클래스가 구현할 메소드를 명료하게 알려주는 인터페이스 역할을 하고, 서브 클래스는 추상 메소드를 목적에 맞게 구현하는 다형성을 실현 할 수 있다.

## ■ 추상 클래스의 용도

- 추상 클래스를 이용하면 응용 프로그램의 설계와 구현을 분리 할 수 있다.

# 인터페이스 기초

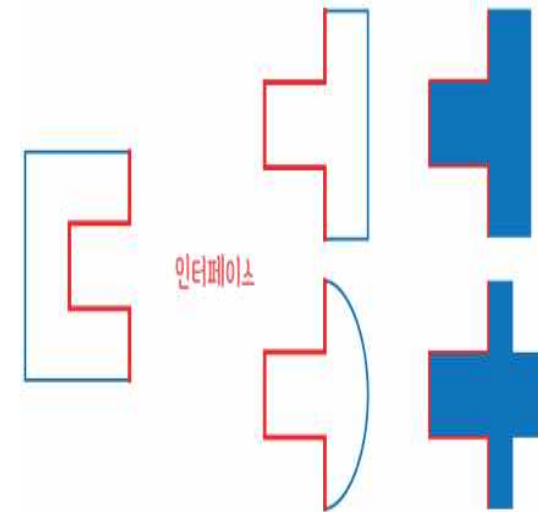
## ■ 인터페이스 의미



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

정해진 규격(인터페이스)에 맞지  
않으면 연결 불가

현실 세계의 인터페이스



인터페이스 : 규격, 표준화 문서

자바의 인터페이스

## ■ 인터페이스에 의한 장점

- 인터페이스만 준수하면 통합에 신경 쓰지 않고 다양한 형태로 새로운 클래스를 개발할 수 있다.
- 클래스의 다중 상속을 지원하지 않지만, 인터페이스로 다중 상속 효과를 간접적으로 얻을 수 있다

# 추상 클래스와 인터페이스 비교

## ■ 유사점

- ▣ 객체를 생성할 수 없고, 상속을 위한 슈퍼 클래스로만 사용
- ▣ 클래스의 다형성을 실현하기 위한 목적

## ■ 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"> <li>• 추상 메소드와 일반 메소드 모두 포함</li> <li>• 상수, 변수 필드 모두 포함</li> </ul>
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"> <li>• 변수 필드(멤버 변수)는 포함하지 않음</li> <li>• 상수, 추상 메소드, 일반 메소드, default 메소드, static 메소드 모두 포함</li> <li>• protected 접근 지정 선언 불가</li> <li>• 다중 상속 지원</li> </ul>

분류	인터페이스	추상 클래스
구현 메서드	포함 불가(단, 디폴트 메서드와 정적 메서드는 예외)	포함 가능
인스턴스 변수	포함 불가능	포함 가능
다중 상속	가능	불가능
디폴트 메서드	선언 가능	선언 불가능
생성자와 main( )	선언 불가능	선언 가능
상속에서의 부모	인터페이스	인터페이스, 추상 클래스
접근 범위	모든 멤버를 공개	추상 메서드를 최소한 자식에게 공개

# 자바의 인터페이스

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

### ▣ 상수

- public만 허용, public static final 생략

### ▣ 추상 메소드

- public abstract 생략 가능

### ▣ default 메소드

- 인터페이스에 코드가 작성된 메소드
- 인터페이스를 구현하는 클래스에 자동 상속
- public 접근 지정만 허용. 생략 가능

### ▣ 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 구현한 클래스 작성
  - ▣ implements 키워드 사용
  - ▣ 여러 개의 인터페이스 동시 구현 가능
- 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 메소드상속

# 예제 7-1 인터페이스 구현

PhoneInterface 인터페이스를 구현하고 flash() 메소드를 추가한 SamsungPhone 클래스를 작성하라.

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

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

# 인터페이스 상속

- 인터페이스가 다른 인터페이스 상속
  - ▣ extends 키워드 이용

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

- ▣ 다중 인터페이스 상속

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

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

# 다중 인터페이스 구현

클래스는 하나 이상의 인터페이스를 구현할 수 있음

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

```
class iPhone implements MobilePhoneInterface, AllInterface { // 인터페이스 구현  
    // MobilePhoneInterface의 모든 메소드를 구현한다.
```

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

클래스에서 인터페이스의 메소드를 구현할 때  
public을 생략하면 오류 발생

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

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

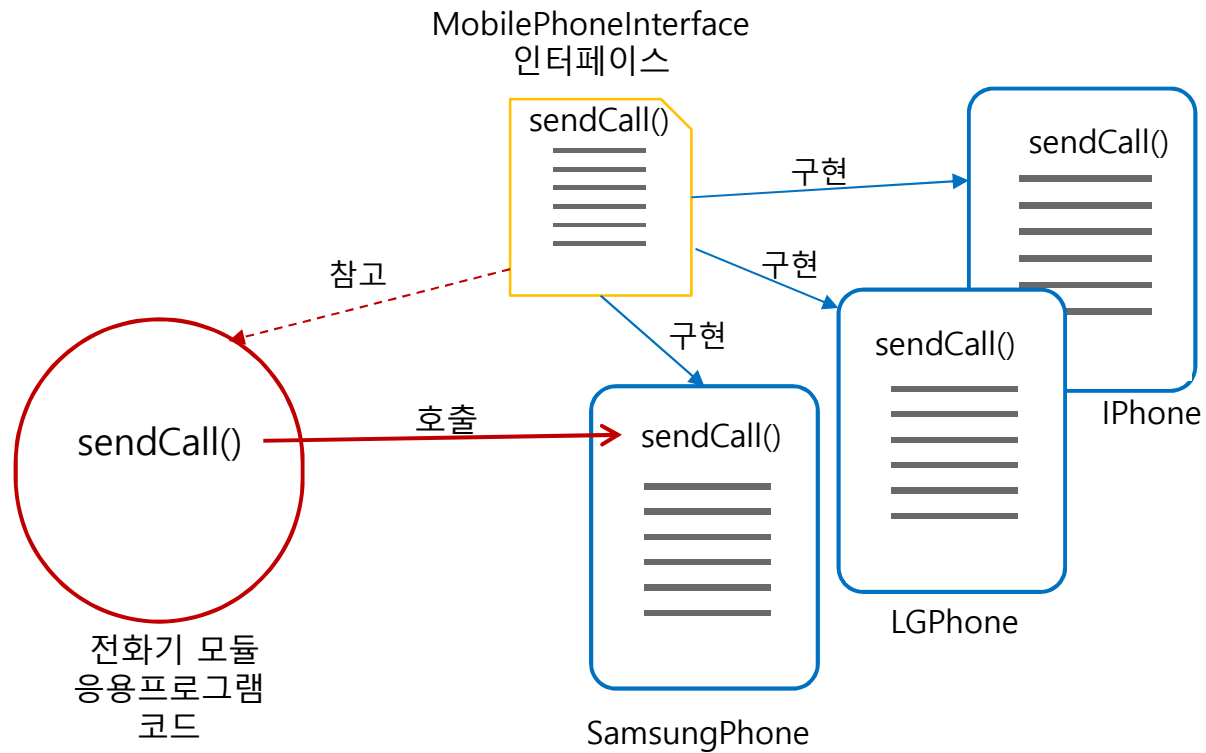
```
    // 추가적으로 다른 메소드를 작성할 수 있다.
```

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

```
}
```

# 인터페이스의 목적

인터페이스는 스펙을 주어 클래스들이 그 기능을 서로 다르게 구현할 수 있도록 하는 클래스의 규격 선언이며, 클래스의 다형성을 실현하는 도구이다



# [quiz7\_1\_식별자] 추상 클래스

다음 프로그램의 빈 곳을 채우고 프로그램의 중요 문장에 주석을 넣으시오.

```
week5
  JRE System Library [JavaSE-17]
  src
    w5_111
      quiz7_1_111.java
      module-info.java
```

<terminated> quiz7\_1\_111  
원을 그린다.  
원의 넓이는 28.3  
사각형을 그린다.  
사각형의 넓이는 12.0

```
package w5_111;
abstract class Shape {
    double pi = 3.14; //
    abstract void draw(); //
    public double findArea() { //
        return 0.0; }
}
```

```
class Circle extends Shape {
    int radius;
    public Circle(int radius) {
        ???; }
    public void draw() { //
        ??? ;
    }
    public double findArea() { //
        ??? ;
    }
}
```

```
class Rectangle extends Shape {
    int width, height;
    public Rectangle(int width, int height) {
        ???;
        ???;
    }
    public void draw() { //
        ???;
    }
    public double findArea() { //
        ???;
    }
}
```

```
public class quiz7_1_111 {
    public static void main(String[] args) {
        // Shape s = new Shape(); //

        Circle c = new Circle(3);
        c.draw();
        System.out.printf("원의 넓이는 %.1f\n", c.findArea());

        Rectangle r = new Rectangle(3, 4);
        r.draw();
        System.out.printf("사각형의 넓이는 %.1f\n", r.findArea());
    }
}
```

## [quiz7\_2\_식별자] 추상 클래스

다음 추상 클래스 Calculator를 상속받은 GoodCalc 클래스를 구현하시오.

```
package w5_111;
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);
}
```

//GoodCalc 클래스 구현

```
public class quiz7_2_111 {
    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 }));
    }
}
```

<terminated> quiz7\_2\_111

5

-1

3.0

# [quiz7\_3\_1\_식별자] 여러 개의 클래스 다루기(4장클래스)

다음은 사칙 계산기 프로그램이다. 프로그램의 빈 곳을 채우시오.

<terminated> quiz7\_3\_111 [Java Application] <terminated> quiz7\_3\_111 [Java Application]

두 정수와 연산자를 입력하시오>>5 9 +

14

두 정수와 연산자를 입력하시오>>6 5 \*

30

```
package w5_111;
import java.util.Scanner;

class Add {    private int a, b;
               public void setValue(int a, int b) {
                   ????.a = a;    ????.b = b;    }
               public int calculate() {
                   ???;    }
}
class Mul {    private int a, b;
               public void setValue(int a, int b) {
                   ????.a = a;    ????.b = b;    }
               public int calculate() {
                   ???;    }
}
class Sub {    private int a, b;
               public void setValue(int a, int b) {
                   ????.a = a;    ????.b = b;    }
               public int calculate() {
                   ???;    }
}
class Div {    private int a, b;
               public void setValue(int a, int b) {
                   ????.a = a;    ????.b = b;    }
               public int calculate() {
                   int res = 0;
                   try {
                       res = ???;
                   } catch (ArithmeticException e) {
                       System.out.print("0으로 나눌 수 없습니다.");
                   }
                   ???;
               }
}
}
```

```
public class quiz7_3_1_111 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("두 정수와 연산자를 입력하시오>>");
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        char operator = scanner.next().charAt(0); // 연산자를 문자로 변환
        switch (???) {
            case '+':
                Add add = ??? Add();
                add.???(a, b);
                System.out.println(????.calculate());
                break;
            case '-':
                Sub sub = ??? Sub();
                sub.??? (a, b);
                System.out.println(????.calculate());
                break;
            case '*':
                Mul mul = ??? Mul();
                mul.??? (a, b);
                System.out.println(????.calculate());
                break;
            case '/':
                Div div = ??? Div();
                div.??? (a, b);
                System.out.println(????.calculate());
                break;
            default:
                System.out.println("잘못된 연산자입니다.");
        }
    }
}
```



# [quiz7\_3\_2\_식별자] 상속, 추상 클래스, 동적 바인딩 활용

Quiz7\_3\_1\_식별자 프로그램에서 Add, Sub, Mul, Div 클래스에 공통된 필드와 메소드를 추출하여 추상 클래스 Calc 클래스로 작성하고 Calc를 상속받아 Add, Sub, Mul, Div 클래스를 작성하고 main()메소드에서 2개의 정수와 연산자를 입력 받은 후 Add, Sub, Mul, Div 중에서 이 연산을 처리 할 수 있는 객체를 생성하고 setValue()와 calculate()를 호출하여 그 결과 값을 화면에 출력하는 프로그램을 작성하시오.

=> 상속, 추상 클래스, 동적 바인딩 개념을 활용하시오.

```
<terminated> quiz7_3_2_111 (1) [Java Application]
```

```
두 정수와 연산자를 입력하시오>> 3 0 /  
계산할 수 없습니다.
```

```
package w5_2_111;  
import java.util.Scanner;
```

```
abstract class Calc { // 추상 클래스
```

```
    // 여러 문장
```

```
}
```

```
// 추상 클래스 Calc 클래스를 상속받아 Add, Sub, Mul, Div 클래스를 작성
```

```
public class quiz7_3_2_111 {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("두 정수와 연산자를 입력하시오>>");  
        int a = scanner.nextInt();  
        int b = scanner.nextInt();  
        char operator = scanner.next().charAt(0); // 연산자를 문자로 변환  
        Calc exp;  
        switch (operator) {  
            case '+':  
                exp = ??? ;  
                break;  
            case '-':  
                exp = ??? ;  
                break;  
            case '*':  
                exp = ??? ;  
                break;  
            case '/':  
                exp = ??? ;  
                break;  
            default:  
                System.out.println("잘못된 연산자입니다.");  
                return; //switch end  
        }  
        exp.??? (a, b); // 피연산자 a와 b 값을 객체에 저장  
        if (exp.??? Div && b == 0) // 0으로 나누는 경우  
            System.out.println("계산할 수 없습니다.");  
        else  
            System.out.println(????.calculate());  
    } //main end } //class end
```

# [quiz7\_4\_식별자] 인터페이스 활용

인터페이스를 구현하고 동시에 클래스를 상속받는 프로그램에서 빈 곳을 채워 프로그램을 완성하고 각 인터페이스와 클래스의 계층 구조를 그리고 주석이 있으면 주석을 넣으시오.

```
interface PhoneInterface { // ???
    final int TIMEOUT = 10000; // ???
    void sendCall(); // ???
    void receiveCall(); // ???
    default void printLogo() { // ???
        System.out.println("** Phone **");
    }
}

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

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

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

// ???
// ???
class SmartPhone extends PDA implements
MobilePhoneInterface, MP3Interface {
    // ???의 추상 메소드 구현
    @Override
    public void ???() {
        System.out.println("따르릉따르릉~~");
    }
    @Override
    public void ???() {
        System.out.println("전화 왔어요.");
    }
}
```

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

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

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

@Override
public void stop() {
    System.out.println("음악 중단합니다.");
}

// ???
public void schedule() {
    System.out.println("일정 관리합니다.");
}
}

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();
    }
}
```

클래스에서  
인터페이스의  
메소드를 구현할 때  
**public**을 생략하면  
오류 발생

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

# [quiz7\_5\_식별자]인터페이스 활용한 타입변환과 다형성

다음 프로그램에서 빈 곳을 채워 프로그램을 완성한 후 주석을 작성하고  
Main 메서드를 인터페이스를 활용한 타입 변환과 다형성을 고려하여 중복된 메서드를 간단하게 표현하시오.

```
package w6_111;
public interface Controllable {
    ??? void repair() { // 디폴트 메서드
        show("장비를 수리한다."); }
    ??? void reset() { // 정적 메서드
        System.out.println("장비를 초기화한다.");
    }
    private void show(String s) {
        System.out.println(s); }
    void turnOn(); //
    void turnOff(); //
}
```

```
package w6_111;
public class TV ??? Controllable {
    @Override //
    ??? void turnOn() { //
        ???; }
    @Override
    ??? void turnOff() { //
        ???; }
}
```

```
package w6_111;
public class Computer ??? Controllable {
    @Override //
    ??? void turnOn() { //
        ???; }
    @Override
    ??? void turnOff() { //
        ???; }
}
```

전자제품에 포함되어야 하는 제어부의 요구 조건

- 모든 전자제품에는 전원을 온·오프하는 기능이 있으며, 수리 및 공장 초기화를 할 수 있다.
- 전자제품 객체는 turnOn() 메서드, turnOff() 메서드로만 전원을 조절할 수 있어야 한다.
- 수리 및 공장 초기화 기능을 미리 구현해 놓아서 필요할 때 사용할 수 있어야 한다.
- 수리 기능은 자식 클래스에서 오버라이딩 할 수도 있다.

```
TV를 켜다.
TV를 끈다.
장비를 수리한다.
컴퓨터를 켜다.
컴퓨터를 끈다.
장비를 수리한다.
장비를 초기화한다.
```

```
package w6_111;
public interface RemoteControllable ??? Controllable {
    void remoteOn();
    void remoteOff(); }
}
```

```
package w6_111;
public class ControllableDemo {
    public static void main(String[] args) {
        TV tv = new TV();
        Computer com = new Computer();
        tv.turnOn(); tv.turnOff(); tv.repair();
        com.turnOn(); com.turnOff(); com.repair();
        Controllable.reset();
        // tv.reset();
        // com.reset();
    } }
}
```