

4부 상속

- 19장 인터페이스

최문환



19장 인터페이스

1. 인터페이스의 정의
2. 다중 상속을 위한 인터페이스
3. 인터페이스의 상속
4. 인터페이스의 속성



1. 인터페이스의 정의

```
접근_지정자 interface 인터페이스_이름{  
    상수;  
    접근_지정자 추상_메서드_이름(인자들);  
}
```

1. 인터페이스의 정의

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

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

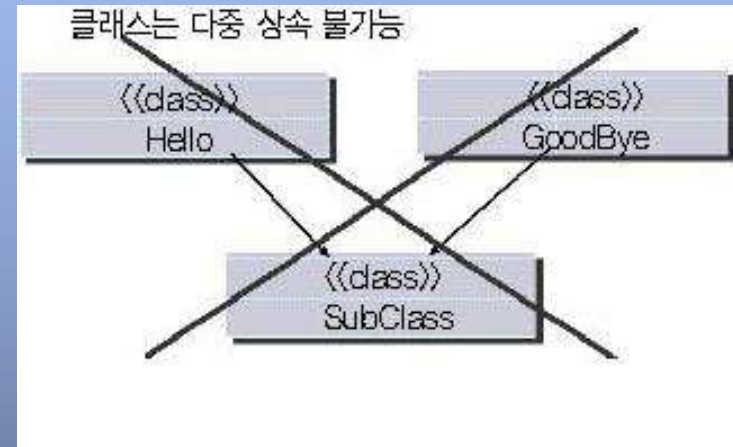
```
class Hello implements IHello{  
    //인터페이스의 추상 메서드 오버라이딩 : 접근 지정자를 반드시 public으로 해야 함  
    public void sayHello(String name){  
        System.out.println(name+"씨 안녕하세요!");  
    }  
}
```

<예제> 인터페이스 정의와 이를 구현한 클래스

```
001:interface IHello{
002: void sayHello(String name);
003:}
004:class Hello implements IHello{
005: public void sayHello(String name){
006: //void sayHello(String name){
007:     System.out.println(name+"씨 안녕하세요!");
008: }
009:}
010:class InterfaceTest01{
011: public static void main(String[] args) {
012:     Hello obj= new Hello();
013:     obj.sayHello(args[0]);//윤정
014: }
015:}
```

2. 다중 상속을 위한 인터페이스

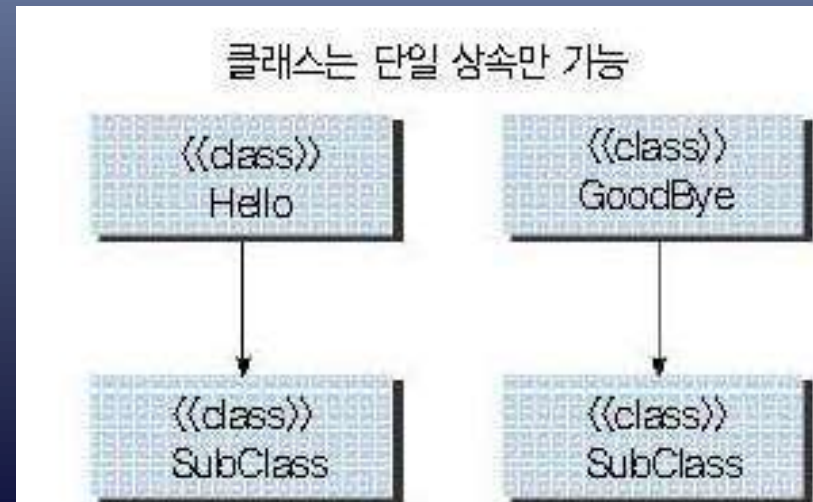
```
001:abstract class Hello{
002: public abstract void sayHello(String name);
003:}
004:abstract class GoodBye{
005: public abstract void sayGoodBye(String name);
006:}
007:
008:class SubClass extends GoodBye, Hello {
009: public void sayHello(String name){
010:     System.out.println(name+"씨 안녕하세요!");
011: }
012: public void sayGoodBye(String name){
013:     System.out.println(name+"씨 안녕히 가세요!");
014: }
015:}
```



```
016:class AbstractTest03{
017: public static void main(String[] args) {
018:     SubClass test= new SubClass();
019:     test.sayHello(args[0]);
020:     test.sayGoodBye(args[0]);
021: }
022:}
```

<예제> 클래스는 단일 상속만 가능

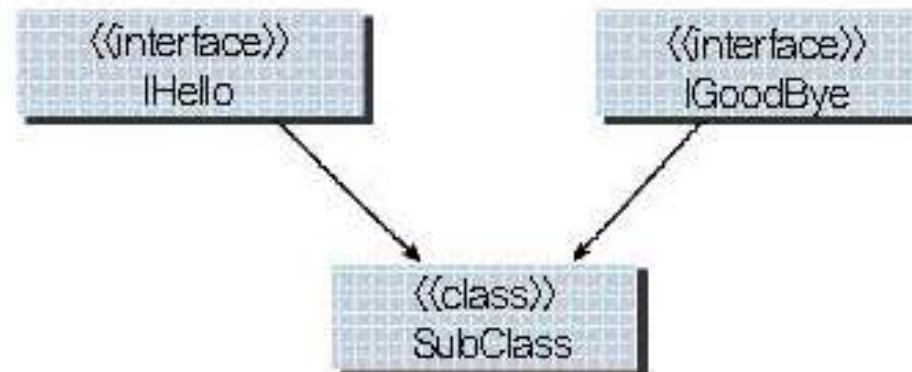
```
001:abstract class Hello{
002: public abstract void sayHello(String name);
003:}
004:abstract class GoodBye{
005: public abstract void sayGoodBye(String name);
006:}
007:
008:
009:class SubClass extends Hello{//추상 클래스 Hello를 상속 받았으므로
010: public void sayHello(String name){//추상 클래스 Hello 내의 추상 메서드 sayHello를 오버라이딩
011:   System.out.println(name+"씨 안녕하세요!");
012: }
013: public void sayGoodBye(String name){ //오버라이딩 한 것이 아님
014:   System.out.println(name+"씨 안녕히 가세요!");
015: }
016:}
017:class AbstractTest02{
018: public static void main(String[] args) {
019:   SubClass test= new SubClass();
020:   test.sayHello(args[0]);
021:   test.sayGoodBye(args[0]);
022: }
023:}
```



<예제> 인터페이스를 이용한 다중 상속

```
001: interface IHello{
002: public abstract void sayHello(String name);
003:}
004: interface IGoodBye{
005: public abstract void sayGoodBye(String name);
006:}
007: //두 인터페이스로부터 상속을 받는 클래스 설계
008: class SubClass implements IHello, IGoodBye{
009: public void sayHello(String name){
010:     System.out.println(name+"씨 안녕하세요!");
011: }
012: public void sayGoodBye(String name){
013:     System.out.println(name+"씨 안녕히 가세요!");
014: }
015:}
016: class AbstractTest05{
017: public static void main(String[] args)
018:     SubClass test= new SubClass();
019:     test.sayHello(args[0]);
020:     test.sayGoodBye(args[0]);
021: }
022:}
```

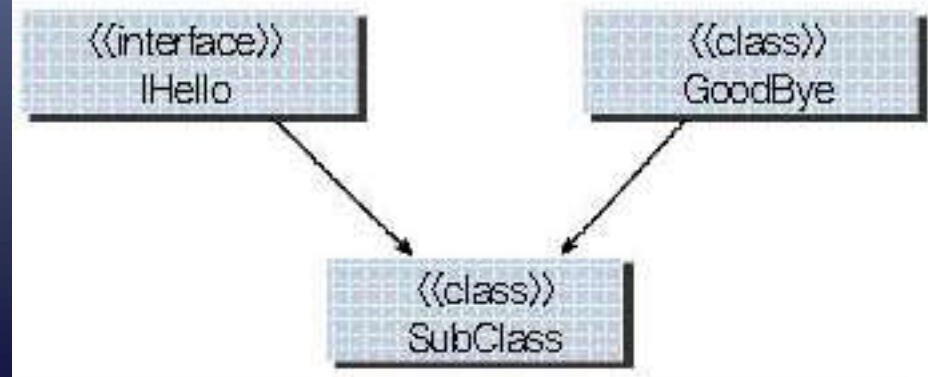
인터페이스의 계층 구조



<예제> 인터페이스와 클래스 혼합

```
001:interface IHello{
002: public abstract void sayHello(String name);
003:}
004:abstract class GoodBye{
005: public abstract void sayGoodBye(String name);
006:}
007:
008:class SubClass extends GoodBye implements IHello{
009: public void sayHello(String name){
010:   System.out.println(name+"씨 안녕하세요!");
011: }
012: public void sayGoodBye(String name){
013:   System.out.println(name+"씨 안녕히 가세요!");
014: }
015:}
016:class AbstractTest04{
017: public static void main(String[] args) {
018:   SubClass test= new SubClass();
019:   test.sayHello(args[0]);
020:   test.sayGoodBye(args[0]);
021: }
022:}
```

인터페이스와 클래스 혼합



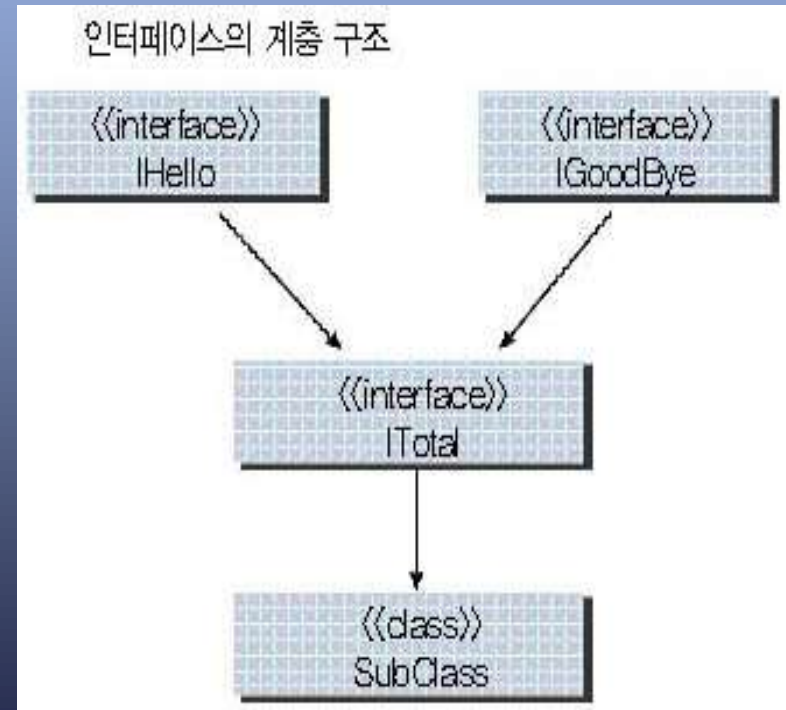
3. 인터페이스의 상속

인터페이스 선언시 필요에 따라 다른 인터페이스로부터 상속 받을 수 있습니다. 인터페이스들의 상속에도 extends 예약어를 사용합니다.

```
interface 인터페이스_이름 extends 인터페이스_이름[, 인터페이스_이름...]{  
    상수 선언  
    추상_메서드 선언  
}
```

<예제> 인터페이스의 상속

```
001:interface IHello{
002: public abstract void sayHello(String name);
003:}
004:interface IGoodBye{
005: public abstract void sayGoodBye(String name);
006:}
007:interface ITotal extends IHello, IGoodBye{
008: public abstract void greeting(String name);
009:}
010:
011:class SubClass implements ITotal{
012: public void sayHello(String name){
013:   System.out.println(name+"씨 안녕하세요!");
014: }
015: public void sayGoodBye(String name){
016:   System.out.println(name+"씨 안녕가세요!");
017: }
018: public void greeting(String name){
019:   System.out.println(name + ", 안녕!");
020: }
021: }
```



<예제> 인터페이스의 상속

```
023: class AbstractTest06{  
024:     public static void main(String[] args) {  
025:         SubClass test= new SubClass();  
026:         test.sayHello(args[0]);  
027:         test.sayGoodBye(args[0]);  
028:         test.greeting(args[0]);  
029:     }  
030: }
```

4. 인터페이스의 속성

```
001: //인터페이스(IColor)므로 다중 상속 가능
002: interface IColor{
003:     int RED=1;                //상수(public static final 로 인식)
004:     public static final int GREEN=2;    //상수
005:     public static final int BLUE=3;    //상수
006:     void setColor(int c);            //추상메서드 (public abstract 로 인식)
007:     public abstract int getColor();    //추상메서드
008: }
009: //클래스(AbsColor)이므로 다중 상속 불가능 단일 상속만,
010: abstract class AbsColor implements IColor{
011:     int color=GREEN;                //변수도 가질 수 있다.
012:     public void setColor(int c){    //구현된 메서드도 가질 수 있다.
013:         color=c;
014:     }
015: }
016: class SubClass extends AbsColor{
017:     public int getColor(){
018:         return color;
019:     }
020: }
```

4. 인터페이스의 속성

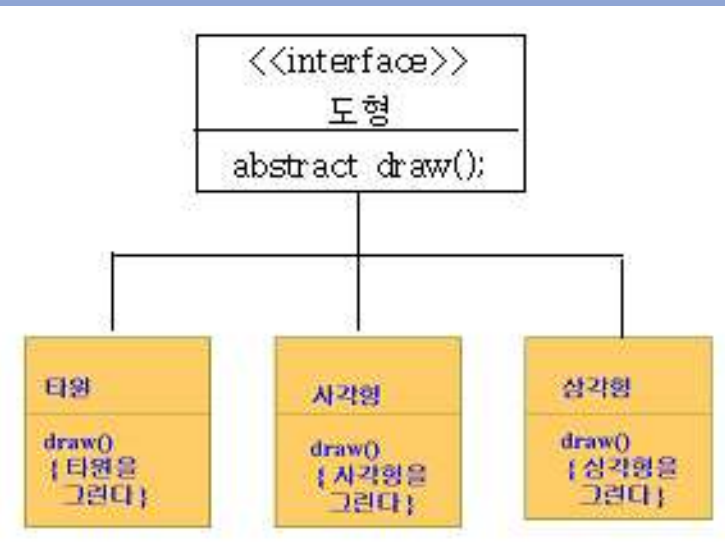
```
022: class AbstractTest07{  
023:     public static void main(String[] args) {  
024:         SubClass test= new SubClass();  
025:         test.setColor(IColor.RED);  
026:         System.out.println(test.getColor());  
027:     }  
028: }
```

추상 클래스와 인터페이스의 차이점

추상 클래스	인터페이스
클래스이다. (abstract class로 선언)	클래스가 아니다.(interface로 선언)
일반 메서드와 추상 메서드를 모두 가질 수 있다.	public abstract 추상 메서드만 가질 수 있다(jdk1.7). Jdk1.8 이후 부터는 static 메서드와 default 메서드를 가질 수 있다.
확장을 하며 extends로 서브 클래스로 정의	구현을 하며 implements로 서브 클래스 정의
단일 상속만 가능하다.	다중 상속도 가능하다.
변수와 상수를 모두 가질 수 있다.	public static final 형태의 상수만 가질 수 있다.
모든 추상 메서드는 객체 생성을 위한 서브 클래스에서 반드시 구현되어야 한다. 업캐스팅이 가능하다.	

<문제>

1. 다음과 같은 인터페이스와 클래스를 설계하십시오.



다음은 설계된 클래스로 프로그래밍한 것입니다.

```
public class Ex19_01 {
    public static void main(String[] args) {
        IShapeClass ref;
        ref = new Circ();
        ref.draw();

        ref = new Rect();
        ref.draw();

        ref = new Tria();
        ref.draw();
    }
}
```


<문제>

2. 다음 예제는 컴파일 에러가 발생합니다. 성공적으로 컴파일하기 위해서 어떤 부분을 수정해야 하는지 적합한 설명을 하시오.

```
1. public abstract class Test {  
2.     public abstract void methodA();  
3.  
4.     public abstract void methodB()  
5. {  
6.     System.out.println("Hello");  
7. }  
8. }
```