

# JAVA Programming

*Abstract class and interface*

BeomSeok Kim


Department of Computer Engineering  
KyungHee University  
passion0822@khu.ac.kr

# Contents



- Abstract classes and abstract methods
- Abstract classes and inheritance
- Interface
- Implementation of interface

# Abstract classes and abstract methods

- 다양한 프레임워크에 기반을 둔 프로그램을 개발하는 대규모 시스템에서 주로 사용
- 정해진 틀에 따라 프로그램을 개발하는 기본적인 방법
- 추상 클래스
  - ✓ 추상 메소드를 한 개 이상 포함하는 클래스
- 추상 메소드 
  - ✓ 메소드의 이름만 있고 실행 코드가 없는 메소드
    - 실행 코드 부분은 작성하지 않는다.



추상 클래스를 선언할때 키워드

추상 메소드를 선언할때 키워드

실행 코드 부분을 작성하지 않는다

[추상 클래스와 추상 메소드를 선언하는 방법]

```
abstract class [클래스명] {  
    --- abstract public void SetID(byte byID);  
}
```

# Abstract classes and abstract methods

- ✓ 추상 메소드의 구조적인 특징
  - abstract 키워드를 추가
  - 하나 이상의 추상 메소드가 존재
  - 추상 메소드 외에 구현된 일반 메소드나 변수도 존재
  - 구현 내용은 존재하지 않음
  - 서브 클래스를 만들어 추상 메소드를 오버라이딩 후 사용
  - **private 접근 한정자 사용불가**



[Bicycle 클래스를 추상 클래스로 변환해 선언한 예]

```
public abstract class AbstractBicycle {  
    // 멤버변수 선언  
    int id;  
    String brand;  
  
    // 추상 메소드 선언  
    abstract void prtInfo( );  
  
    // 일반 메소드 구현  
    public String getBrand( ) {  
        return brand;  
    }  
}
```

# Abstract classes and inheritance



## ■ 추상 클래스를 사용하는 목적

- ✓ 상속과 실체화 사이에서 발생하는 논리적인 문제를 해결하기 위함
  - 예) 4장에서 Bicycle 클래스와 서브 클래스인 MountainBike, RoadBike, Minivelo 등의 클래스 관계
  - Bicycle 클래스에 구현된 메소드 내용을 서브 클래스에 그대로 적용하기에는 현실적으로 많은 어려움이 존재함
  - 각 서브 클래스에서 필요한 메소드만 정의한 채 규격만 따르게 하는 것이 클래스를 설계하고 프로그램을 개발하는데 훨씬 도움이 되는 접근방법

여기서 잠깐




객체지향 언어에서 클래스를 이용해 객체를 만드는 일을 인스턴스화라고 한다. 클래스의 주된 용도는 인스턴스화지만 어떤 상황에서는 클래스의 인스턴스화를 막아야 할 때도 있다. 이때도 class 앞에 abstract 키워드를 붙이면 된다.

**인스턴스화를 금지하는  
abstract 키워드**

# Interface



## ■ 인터페이스란?

- ✓ 의미상으로 떨어져 있는 객체를 서로 연결해 주는 규격을 말함
  - 예) usb 규격 – 어떤 usb기기도 컴퓨터에 연결할 수 있는 인터페이스
- ✓ 인터페이스는 개념이나 구조적으로 추상 클래스와 유사함
- ✓ 특징
  - 구현된 메소드가 없다. 
  - 상수형 변수만 가능(일반적인 변수 사용 불가)
  - 모든 메소드는 메소드명과 매개변수, 반환 타입만 있는 추상 메소드로 정의
  - 메소드의 접근 한정자는 **항상 public** 이어야 함

[인터페이스를 선언하고 구현하는 방법]

```
public interface [인터페이스명] {  
    상수형 변수  
    abstract public [반환 타입] [메소드명] (매개변수);  
}  
  
public class [자식 클래스명] implements [인터페이스명] {  
    필드  
    메소드  
}
```


인터페이스 선언 키워드

서브 클래스에서 인터페이스 사용시 키워드

# Interface



## ■ 인터페이스의 구조적 특징 요약

- ✓ 인터페이스를 선언 시 `interface` 키워드를 사용
- ✓ 인터페이스명은 보통 `I`로 시작
- ✓ `public static final`로 선언된 변수에는 상수 가능 
  - `public static final`을 명시하지 않아도 자동으로 컴파일된다.
- ✓ 반드시 추상 메소드만 정의해야 함
- ✓ 인터페이스를 사용하려면 구현(implementation)이라는 용어를 사용
  - `Implements` 키워드를 사용

# Implementation of interface



- 자바에서는 다중상속을 지원하지 않음.
  - ✓ 다중상속은 프로그램을 설계할 때 많은 도움이 됨
  - ✓ 자바에서는 다중상속의 해결점을 상속과 인터페이스 구현을 동시에 이용 하여 해결한다.
  - ✓ 하나의 implements 키워드를 사용 후 각각의 인터페이스를 ','를 이용해 나열

[다중 인터페이스 구현방법]

```
public class [자식 클래스명] implements [인터페이스1], [인터페이스2], ..., [인터페이스n] {  
    필드  
    메소드  
}
```

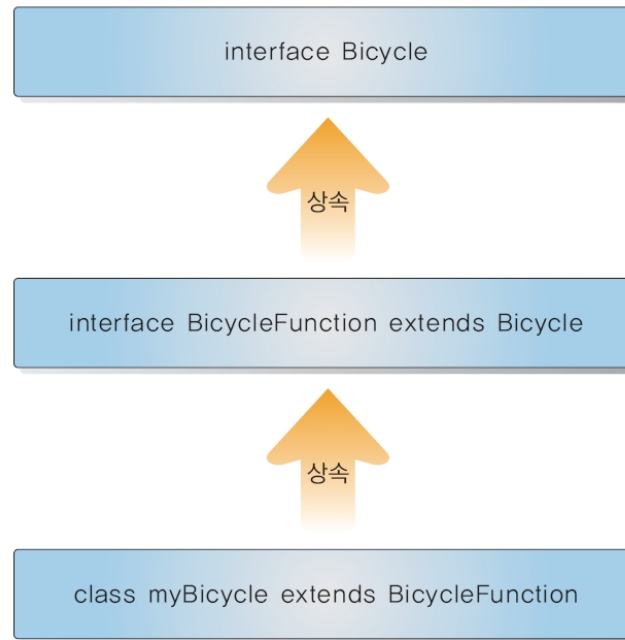


# Implementation of interface



## ■ 상속의 개념이 존재

- ✓ 슈퍼 인터페이스를 상속하는 서브 인터페이스를 정의할 수 있음
- ✓ 인터페이스의 계층구조가 용이
  - 더욱 정밀한 인터페이스와 클래스를 설계할 수 있음
- ✓ 인터페이스 상속은 슈퍼 인터페이스의 구현이 아닌 상속의 개념
  - 주의 : extends 키워드 사용

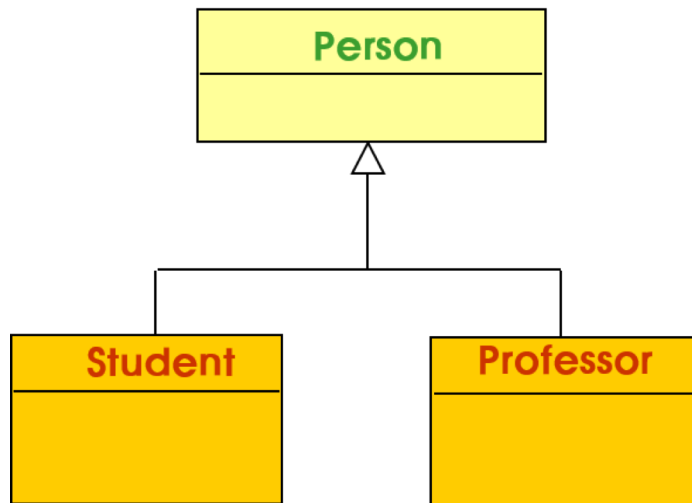


[그림 5-3] 인터페이스의 상속

# Implementation of interface

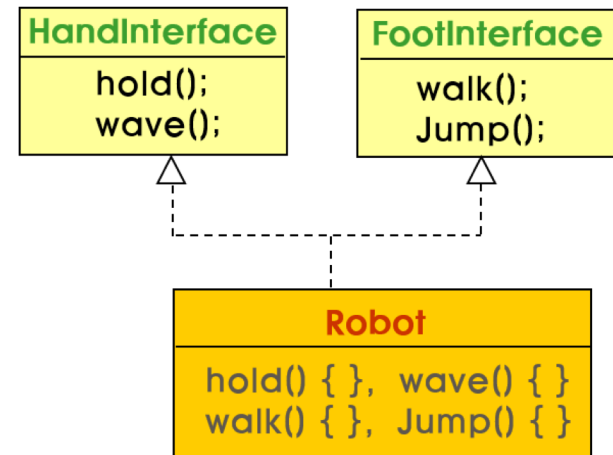
## ■ 단일 상속 (Single inheritance)

- ✓ 수퍼클래스가 하나
- ✓ 상속을 받는 서브클래스는 하나 또는 그 이상



## ■ 다중 상속 (Multiple inheritance)

- ✓ 수퍼 클래스가 여럿
- ✓ 복수의 인터페이스를 상속받는 하나의 인터페이스 선언 가능
- ✓ 복수의 인터페이스를 구현하는 하나의 클래스 선언 가능
- ✓ 복수의 클래스로부터 하나의 클래스로 상속 불가



# Implementation of interface



## ■ 공통점

- ✓ 둘 다 추상메소드를 포함
  - 자식클래스가 추상메소드를 구현

## ■ 차이점


- ✓ 추상클래스는 추상메소드가 아닌 메소드를 가질 수 있음
- ✓ 추상클래스는 단일 상속만 가능한 클래스이므로 인터페이스의 변수는 반드시 static final (상수)여야 함
- ✓ 인터페이스는 반드시 public 메소드만을 포함
- ✓ 추상클래스는 생성자를 정의할 수 있음

# Summary



## ■ 추상 클래스와 추상 메소드

### ✓ 추상 클래스

- 하나 이상의 추상 메소드를 포함하는 클래스 
- 변경될 가능성이 있는 부분을 추후에 상속으로 구현할 수 있도록 메커니즘을 제공
- 인스턴스를 만들 수 없음
- 반드시 상속으로 추상 메소드를 오버라이딩해 사용해야 함

## ■ 인터페이스

- 서로 떨어져 구현되는 객체를 서로 연결해 주는 규격을 기술해 놓은 클래스 형태
  - 예) usb 포트
- 일종의 규격을 제공하는 메커니즘
- 상속과 유사하긴 하나 implements 키워드를 사용
- 인터페이스에 선언된 모든 추상 메소드를 구현해 주어야 함
- 인터페이스는 여러 인터페이스를 동시에 구현 가능(다중상속)

**Thank You!**  
**Q&A**