

# JAVA Programming

*Modifier* 

BeomSeok Kim

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

# Contents

- Introduction to modifier
- 'static' modifier
- 'final' modifier

# Introduction to modifier

## ■ 변경자란?

- ✓ 클래스에 있는 메소드나 필드에 외부 클래스의 접근을 제한하는 **키워드**
  - default, public, private, protected
  - C++에서의 access specifier와 비슷한 개념

## ■ default 변경자

- ✓ 클래스 접근을 **동일 패키지**에 한정하려고 사용
- ✓ 동일 패키지에 있는 모든 클래스에 접근할 수 있는 접근 권한을 제공
- ✓ 사용법
  - 아무런 변경자도 적지 않는다.



[default 변경자 선언방법]

```
class MyBike { // 클래스 선언
    필드
    메소드
}
```

# Introduction to modifier

## ■ Default 변경자

[패키지 & 디렉터리]



[그림 6-1] default 변경자의 접근 권한

# Introduction to modifier

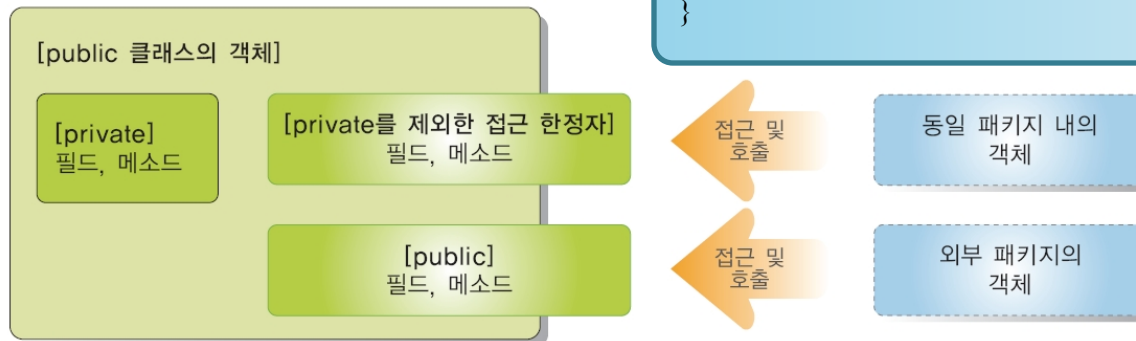
## ■ public 변경자

- ✓ 패키지 외부에서도 접근할 수 있도록 클래스를 개방하는 것
- ✓ 선언된 클래스나 필드, 메소드는 자유롭게 접근이 용이
  - 패키지, 외부 모두 접근 가능
- ✓ 캡슐화를 해치지 않는지 고려해야 함



[public 변경자 선언방법]

```
public class MyBike { // 클래스에 public 변경자 선언
    public int m_iSize; // 필드에 public 변경자 선언
    public int getSize() { // 메소드에 public 변경자 선언
        명령문
    }
}
```



[그림 6-2] public 변경자의 접근 권한


# Introduction to modifier

## ■ private 변경자

- ✓ 클래스의 필드나 메소드에서 외부 클래스의 접근을 차단하는 기능을 제공
- ✓ 클래스 내부 메소드를 제외하고는 접근을 허락하지 않음
- ✓ private로 선언된 메소드 또한 클래스 내부 메소드만 호출 가능

[private 변경자의 선언방법]

```
public class MyBike {  
    private int m_iSize; // private 변경자를 이용한 필드 선언  
    private int getSize() { // private 변경자를 이용한 메소드 선언  
        명령문  
    }  
}
```

- ✓ 객체지향 언어의 특징인 **캡슐화를 구현할 때 매우 유용** 
  - 외부 클래스에 내부의 구현방법이나 접근방법을 철저히 차단시켜 다른 클래스에 의존하지 않고 클래스를 생성할 수 있기 때문

# Introduction to modifier



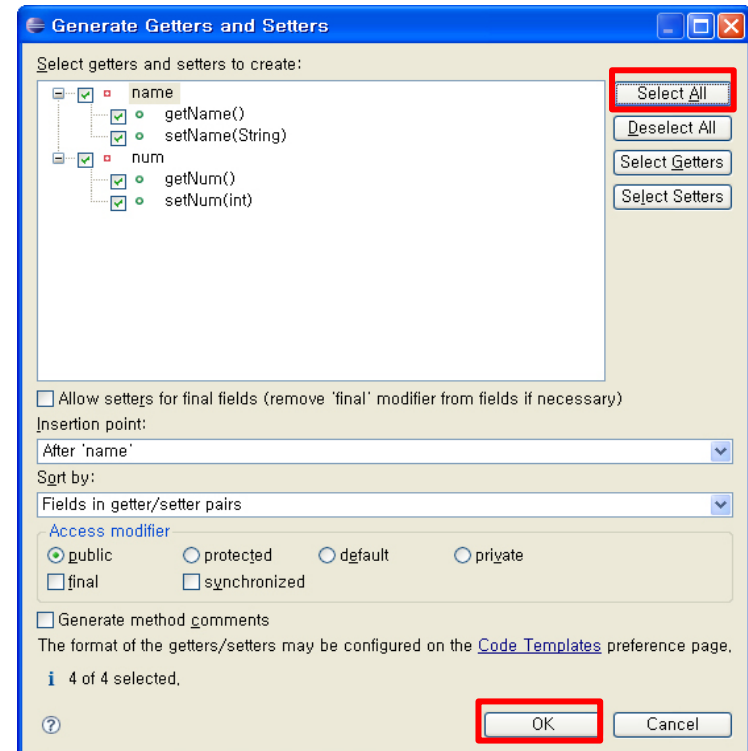
- private 변경자
  - ✓ 보통 멤버변수를 private로 선언
  - ✓ setter/getter 메소드
    - 해당 변수를 외부에서 변경하거나 참조할 수 있도록 만들어 제공한다.
    - getXxx( ), setXxx( ) 형태
      - get, set 접두어에 변수명의 첫 글자를 대문자로 하는 이름 규칙이 존재
    - getter, setter 메소드를 이클립스에서는 이를 자동으로 관리하는 기능을 제공.

# Introduction to modifier

## ■ private 변경자

Undo Typing	Ctrl+Z
Revert File	
Save	Ctrl+S
Open Declaration	F3
Open Type Hierarchy	F4
Open Call Hierarchy	Ctrl+Alt+H
Show in Breadcrumb	Alt+Shift+B
Quick Outline	Ctrl+O
Quick Type Hierarchy	Ctrl+T
Show In	Alt+Shift+W ▶
Cut	Ctrl+X
Copy	Ctrl+C
Copy Qualified Name	
Paste	Ctrl+V
Quick Fix	Ctrl+I
Source	Alt+Shift+S ▶
Refactor	Alt+Shift+T ▶
Local History	▶
References	▶
Declarations	▶
Run As	▶
Debug As	▶
Team	▶
Compare With	▶
Replace With	▶
Preferences...	

Toggle Comment	Ctrl+/ Ctrl+Shift+W
Remove Block Comment	
Generate Element Comment	Alt+Shift+J
Correct Indentation	Ctrl+I
Format	Ctrl+Shift+F
Add Import	Ctrl+Shift+M
Organize Imports	Ctrl+Shift+O
Sort Members...	
Clean Up...	
Override/Implement Methods...	
Generate Getters and Setters...	
Generate Delegate Methods...	
Generate hashCode() and equals()...	
Generate Constructor using Fields...	
Generate Constructors from Superclass...	
Externalize Strings...	





# Introduction to modifier

## ■ protected 변경자

- ✓ private, default, public 등 세 가지 변경자를 혼합한 성격
- ✓ 패키지 외부 클래스에서 접근하는 것은 차단
- ✓ 패키지 내부 클래스에서 접근하는 것은 허용
- ✓ 상속관계에서 접근하는 것은 허용

[protected 변경자 선언방법]

```
public class Bike {  
    protected int m_iSize; // protected 변경자를 이용한 필드 선언  
    protected int getSize() // protected 변경자를 이용한 메소드 선언  
    명령문  
}  
  
public class myBike extends Bike {  
    public setSize(int iSize) { // 슈퍼 클래스의 m_iSize를 자신의 필드처럼 호출  
        m_iSize = iSize;  
    }  
}
```

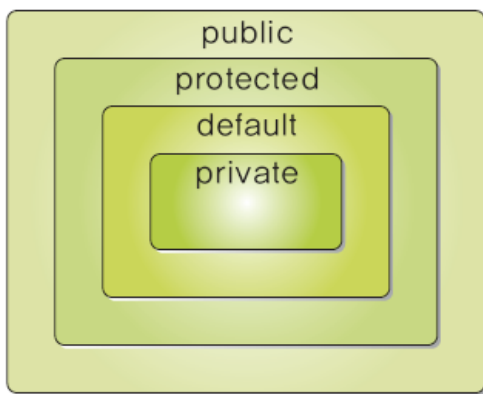
# Introduction to modifier



여기서 잠깐

변경자가 지정하는  
클래스 구성요소의  
접근 가능 범위

자바에서는 변경자를 이용해 클래스의 구성요소에 접근하는 것을 제어하는데, 변경자로 접근 가능한 범위는 그림과 같이 public → protected → default → private 순이다.



[그림 6-5] 변경자의 접근 제어 권한 포함관계

# Introduction to modifier



변경자	동일한 클래스	동일한 패키지	서브 클래스	모든 클래스
public	O	O	O	O
protected	O	O	O	
default	O	O		
private	O			

# 'static' modifier

## ■ 기본개념 및 사용방법

- ✓ 필드나 메소드의 속성을 변경
- ✓ 고정된 메모리 공간이 할당
  - static 영역이 별도의 공유 영역에서 관리
- ✓ 인스턴스를 생성하지 않고 클래스명만 이용해 호출
- ✓ static 메소드에서는 인스턴스 변수에 접근할 수 없다
  - 예) main 메소드는 static으로 선언되어 있기 때문  
동일 클래스에 있는 멤버변수라도 접근할 수 없다
- ✓ static 변경자 사용할 시 유의사항
  - 필드에 static 변경자를 붙일 때
    - 해당 필드가 클래스의 인스턴스와 공유하는 필드인지 확인
  - 메소드에 static을 붙일 때
    - 해당 메소드에서 인스턴스 변수에 접근하지 않는지 확인



[static 변경자 선언방법]

```
public class MyBike {  
    static public int m_iObjectCnt; // static 변경자를 필드에 선언  
    static public void IncreaseObject( ) { // static 변경자를 메소드에 선언  
        m_iObjectCnt++;  
    }  
}
```

# 'final' 변경자

## ■ 기본개념 및 사용방법



- ✓ 클래스나 필드, 메소드 등에 적용해 사용을 제한
- ✓ 클래스에 적용 시
  - 상속이 불가능 해짐
  - 이점 : 하위 클래스가 없음을 명시적으로 표현할 수 있음
- ✓ 메소드에 적용 시
  - 하위 클래스에서 오버라이딩할 수 없음
  - 이점 : 하위 클래스에서 특정 메소드의 기능을 바꿀 수 없도록 할 수 있음
- ✓ 필드에 적용 시
  - 상수형 변수처럼 변경할 수 없는 필드를 만듦
  - 이점 : 클래스에 있는 메소드 간에 일관적으로 변수의 값을 유지할 수 있음  
매개변수를 전달할 시, 실수로 전달받은 변수값을 바꾸는 오류를 최소화

### [final 변경자 선언방법]




```
final public class myBike { // 클래스에 final 선언
    final private double PI = 3.14; // 필드에 final 선언
    final public CalExtent(int iRadius) { // 메소드에 final 선언
        명령문
    }
}
```

# Summary



## ■ 변경자

- ✓ 외부 접근을 제한하는 기능을 제공
- ✓ default 
  - 선언할 때 아무런 변경자도 붙이지 않음
  - 보통, 동일한 패키지에 있는 클래스에 접근할 수 있는 권한을 제공
- ✓ private
  - 동일한 클래스에 접근하는 것을 원칙으로 함
- ✓ protected
  - 패키지 외부에서 접근은 허용하지 않음. 상속관계에서는 접근을 허용

## ■ Static

- ✓ 공유 메모리 영역에 할당되어 다른 클래스에서 공유할 수 있음
  - 필드가 클래스의 인스턴스가 공유하는 필드인지 확인해야 한다.
  - 메소드에서 인스턴스 변수를 접근하지 않는지 확인해야 한다.

## ■ Final

- ✓ 클래스 상속 불가, 메소드 오버라이딩 불가, 변수는 상수
  - 상수를 선언할 때 변수명을 대문자로 표기(관례)

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