



Field

경북대학교
소프트웨어융합과
배희호 교수



QUIZ

1. 객체는 속성 과 동작 을 가지고 있다.
2. 자동차가 객체라면 클래스는 설계도 이다.

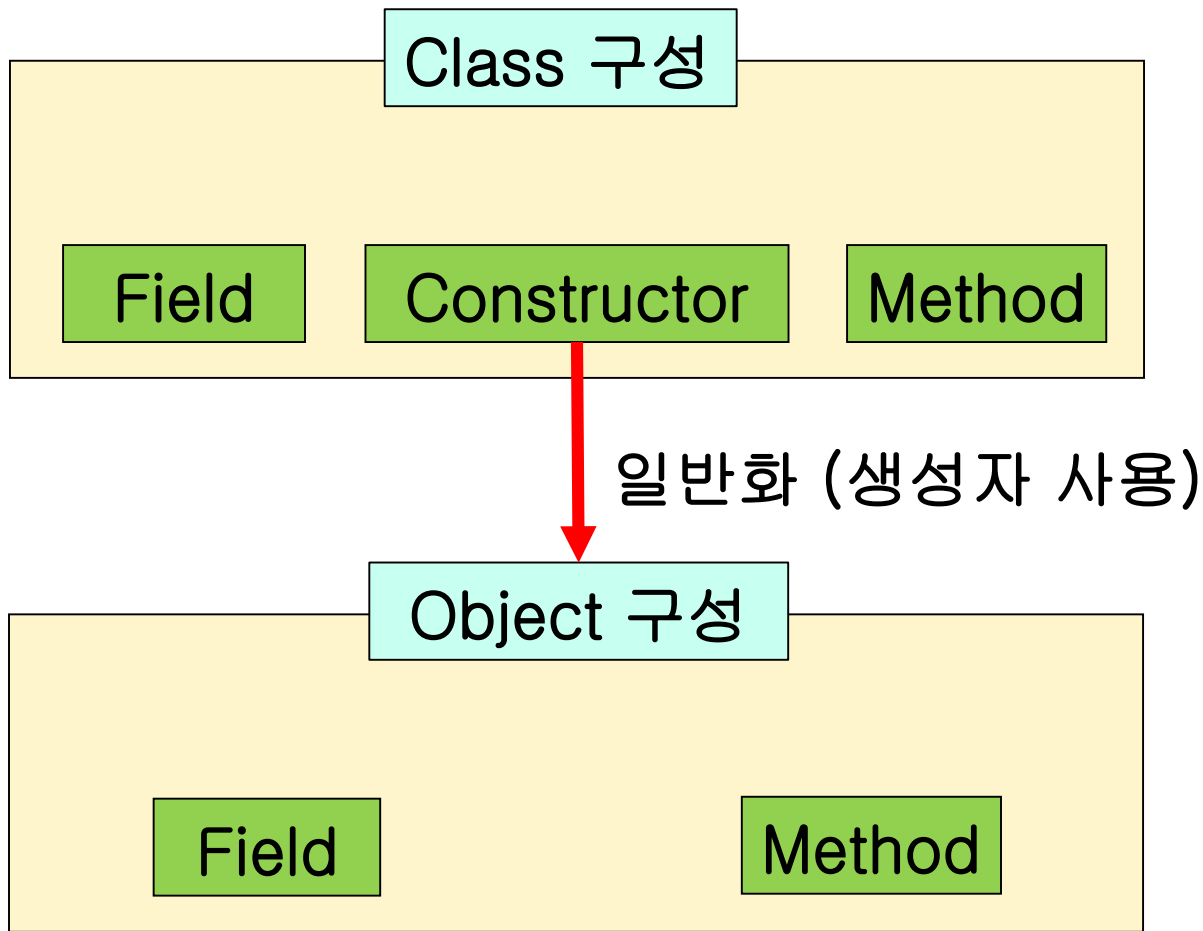


먼저 앞장에서
학습한 클래스와
객체의 개념을
복습해봅시다.





Class와 Object





Field



- Field는 Class 내부에서 선언된 변수로, Object의 Attribute (Data)를 저장하는 역할을 함
- Field는 Object가 생성될 때 Memory에 할당되며, Object의 Lifecycle과 함께 존재함



Field

■ Field 선언

- Field의 접근 지정자는 어떤 Class가 Field에 접근할 수 있는지를 표시
 - public : 이 Field는 모든 Class로부터 접근 가능
 - private : 해당 Class 내부에서만 접근 가능
- 객체 지향 캡슐화 원칙에 따라 Field는 **private로 선언하는 것이 좋음**

접근지정자;
private이나
public

필드의 타입

필드의 이름:
첫 단어만
소문자

public

int

speed;



Field



- Field는 정의된 위치에 상관없이 Class 안에서는 어느 곳에서도 사용 가능

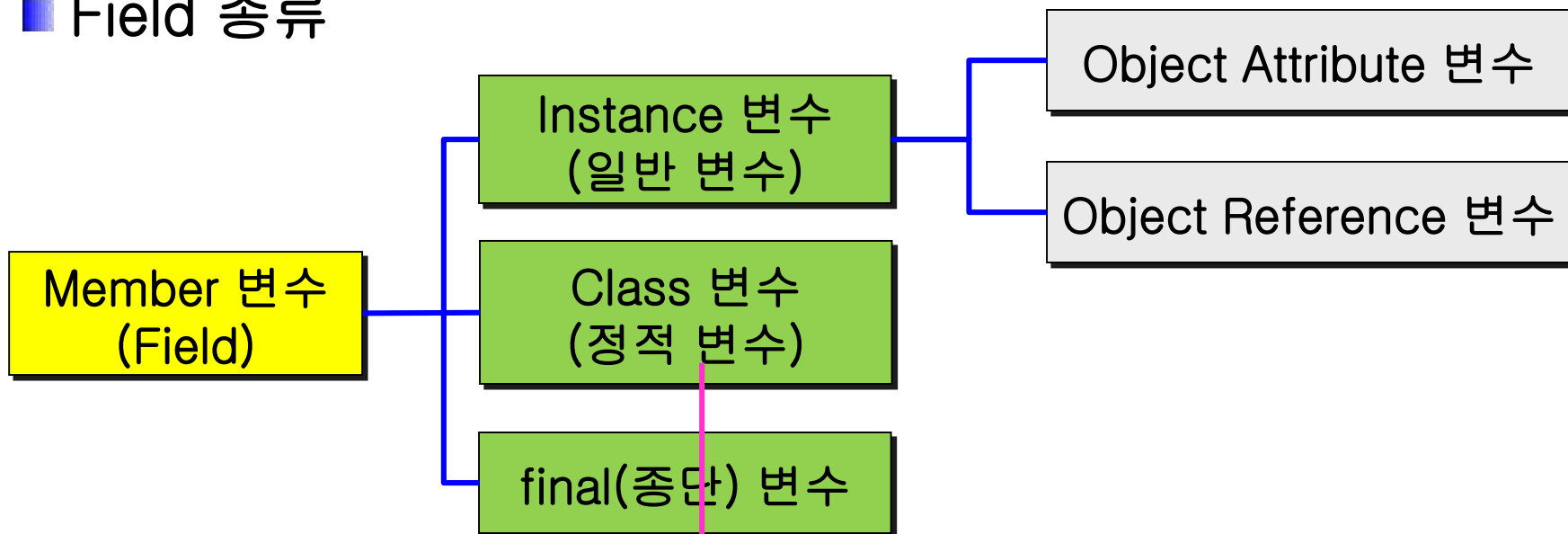
```
public class Date {  
    public void printDate() { // Member Method  
        System.out.println(year + "." + month + "." + day);  
    }  
  
    public int getDay( ) { // Member Method  
        return day;  
    }  
  
    private int year;  
    private String month;  
    private int day;  
}
```

} Member Field



Field

Field 종류



Field 선언

[~~public/private/protected~~] [static/final] 변수형 변수 이름;

- static과 final이 붙지 않은 변수 : Instance 변수 (Object Attribute 변수 또는 Object Reference 변수)



Field



■ Field의 종류

■ Instance 변수

- 각 Object마다 별도로 저장되는 변수
- Object가 생성될 때마다 새로운 Memory 공간이 할당됨
- Object의 상태를 저장하는 역할
- **static Keyword 없이 선언된 변수**

■ Static 변수 (정적 변수, Class 변수)

- **static Keyword를 사용하여 선언한 변수**
- Class에 속하는 변수로, 모든 Object가 공유
- Object가 아니라 Class 단위에서 Memory에 한 번만 할당됨
- 모든 Object가 같은 값을 공유하므로 공통된 속성(예: 전체 카운트, 설정 값 등)에 사용



Field



■ Field의 종류

■ Constant 변수 (상수 변수, final 변수)

- **final Keyword를 사용하여 선언한 변수**

- 값이 한 번 설정되면 변경할 수 없는 변수

- 보통 static final과 함께 사용하여 Class 상수로 선언

- Constant 변수는 대문자로 작성하는 것이 관례임



Instance 변수



- Instance 변수(일반 변수)
 - Object가 가질 수 있는 특성(Attribute)을 표현
 - Instance 변수는 Object가 생성될 때마다 할당되는 변수
 - Instance 변수는 Object의 이름을 사용하여 접근하게 됨
 - Instance 변수를 표현하는 값에 따라 Object Attribute 변수와 Object Reference 변수로 구분
 - Object Attribute 변수
 - Object가 가질 수 있는 Attribute 나타내는 값
 - 기본 Data Type의 값들로 구성
 - Object Reference 변수
 - Object를 지정하는 변수
 - JAVA에서는 기본 Data Type을 제외한 모든 요소들을 Object로 취급



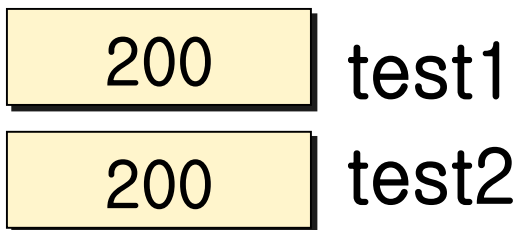
Instance 변수

■ Instance 변수(일반 변수)

- 사용자는 Object를 생성한 다음 그 Object에 접근하기 위해서는 Object Reference 변수를 통하여 그 Object의 Member들에 접근할 수 있음

■ Object Attribute 변수

```
int test1 = 200;  
int test2 = test1;
```



- Object Attribute 변수인 test1과 test2는 서로 다른 기억 장소에 저장된 Data를 말함
- 즉 test1의 값이 복사되어 test2의 값으로 전달



Instance 변수



- Instance 변수는 Class 내부에 위치함
- Instance 변수는 Object(Instance)가 생성될 때 생성 됨
- Instance 변수의 값을 읽어오거나 저장하려면 Instance를 먼저 생성해야 함
- Instance 별로 다른 값을 가질 수 있으므로, 각각의 Instance 마다 고유의 값을 가져야할 때는 Instance 변수로 선언
- Class 변수와의 차이점
 - Instance에 종속되어 Instance 생성시 마다 새로운 저장 공간을 할당. 즉 저장 공간이 공유되지 않음
 - Instance에 종속되기 때문에 꼭 Instance Object에서 호출해 주어야 함



Instance 변수



- Method 밖에서 선언된 변수
 - Member 변수는 같은 Class 안에 있는 모든 Member Method에서 사용할 수 있음
 - Member 변수는 0에 준하는 값으로 자동으로 초기화 됨

```
public class Test {  
    int z;           // instance 변수  
  
    public static void main(String [] args){  
        int x = 4;    // Local 변수  
        int y = 7;    // Local 변수  
        int z = 6;    // Local 변수  
    }  
}
```



Instance 변수

- Instance 변수는 각 Instance마다 독립적인 저장 공간을 갖음

```
public class Test {  
    int x = 3;        // Instance 변수  
    int y;            // instance 변수  
  
    public static void main(String[] args) {  
        Test temp = new Test( );  
        Test test = new Test( );  
  
        temp.y = 5;  
        System.out.println(temp.x + " + " + temp.y);  
        System.out.println(test.x + " + " + test.y);  
    }  
}
```

3 + 5
3 + 0



Instance 변수



■ Object Reference 변수와 Object Attribute 변수 예

```
class Box {  
    int width;    // Object Attribute 변수  
    int height;   // Object Attribute 변수  
    int depth;    // Object Attribute 변수  
}
```

```
class MyBox {  
    int vol;           // Object Attribute 변수  
    String boxname;    // Object Reference 변수 (문자열은 객체로 취급)  
    Box myBox;         // Object Reference 변수  
    myBbox = new Box();  
    Box yourBox = new Box(); // Object Reference 변수  
    .....  
}
```



Instance 변수



- Object Reference 변수와 Object Attribute 속성 변수 대입의 예

```
Box myBox = new Box();  
myBox.width = 100;  
myBox.height = myBox.width; // Object Attribute 변수의 대입  
myBox.depth = 30;
```

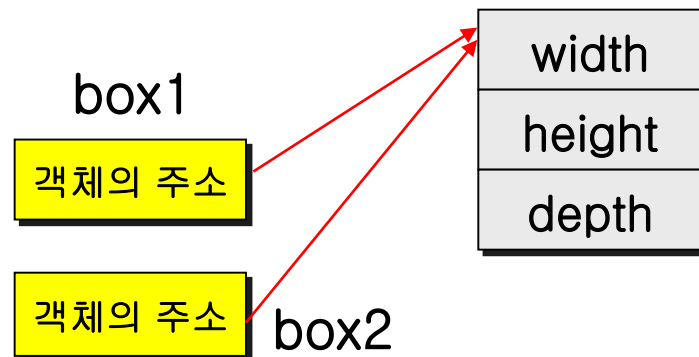
```
Box yourBox = myBox; // Object Reference 변수의 대입  
yourBox.depth = 20;
```




Instance 변수

■ Object Reference 변수

```
Box box1 = new Box();  
Box box2 = box1;
```



- Object Reference 변수인 box1과 box2는 그 값을 가지는 게 아니라 **그 Object의 주소를 가지고 있음**
- Object Reference 변수를 다른 Object Reference 변수에 배정하면 그 Object의 주소를 넘겨주게 되어 결국 같은 Object를 가르키게 됨
- 이 경우 box1 Object의 속성값이 변환되면 box2 Object의 값도 변환 됨



Class 변수(Global Variable)

- **static**을 사용하여 선언한 변수는 Class 변수
- Class 변수는 Class 선언 시에 할당되며 **그 Class의 모든 Object에 공유되는 변수**
- Class 변수는 전역 변수(Global Variable)처럼 동작
- Class 변수 선언 형식

static Data Type 변수 이름 = 초기값;

- Keyword static이 없으면, Instance 변수로 선언된 것
- Class 외부에서 접근하는 방법(**점 연산자 사용**)
 - Instance 변수와는 달리 Class 변수는 Class 이름을 통하여 접근

“클래스 이름.클래스 변수”





Class 변수(Global Variable)

- Class 변수의 용도

- Instance 변수(Object Reference, Object Attribute)는 Object가 생성될 때마다 각 Object에 변수들이 생성되지만, Class 변수는 Class로부터 생성된 Object들의 수와 상관없이 하나만 생성
- 한 Class로부터 생성된 모든 Object들은 Class 변수를 공유
- Class 변수를 이용하여 Object들 사이의 통신에 사용하거나 Object들의 공통 Attribute을 나타낼 수 있음



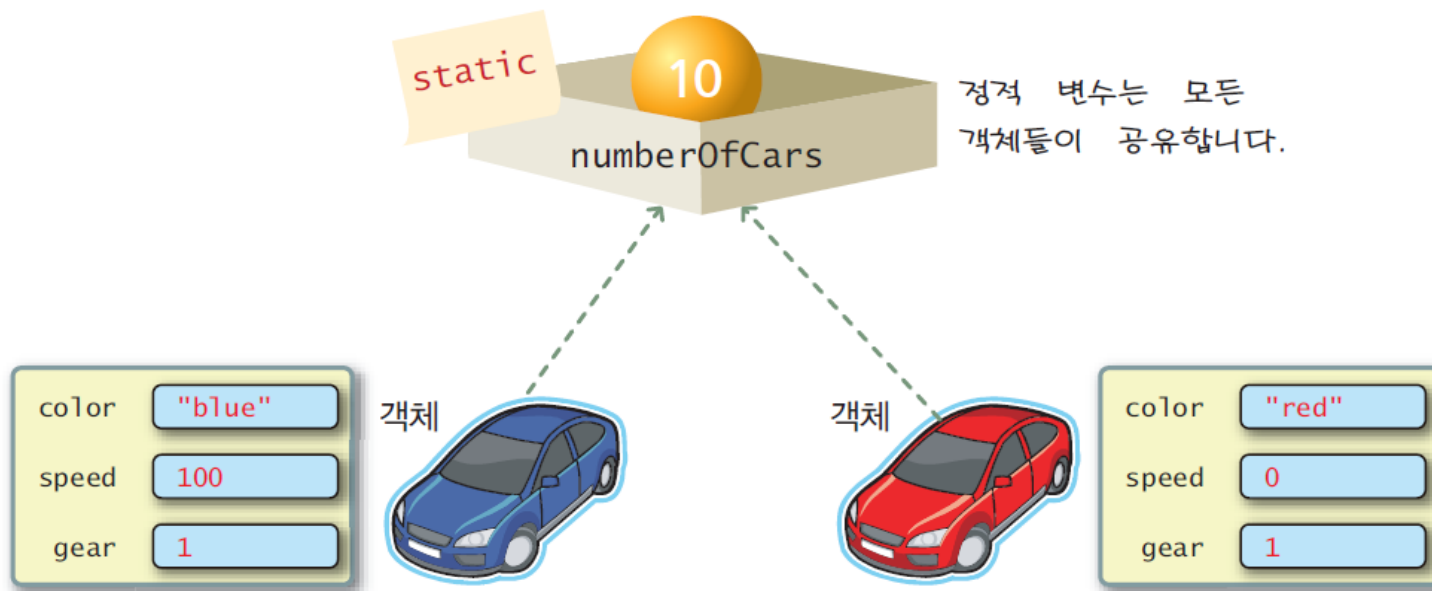
Class 변수(Global Variable)

■ Instance Variable

- Object마다 하나씩 생성되는 변수

■ Static Variable = Class 변수

- 하나의 Class에 하나만 존재
- 그 Class의 모든 Object들에 의하여 공유됨
- Class 변수를 정의할 때 맨 앞에 **static Keyword**를 붙임



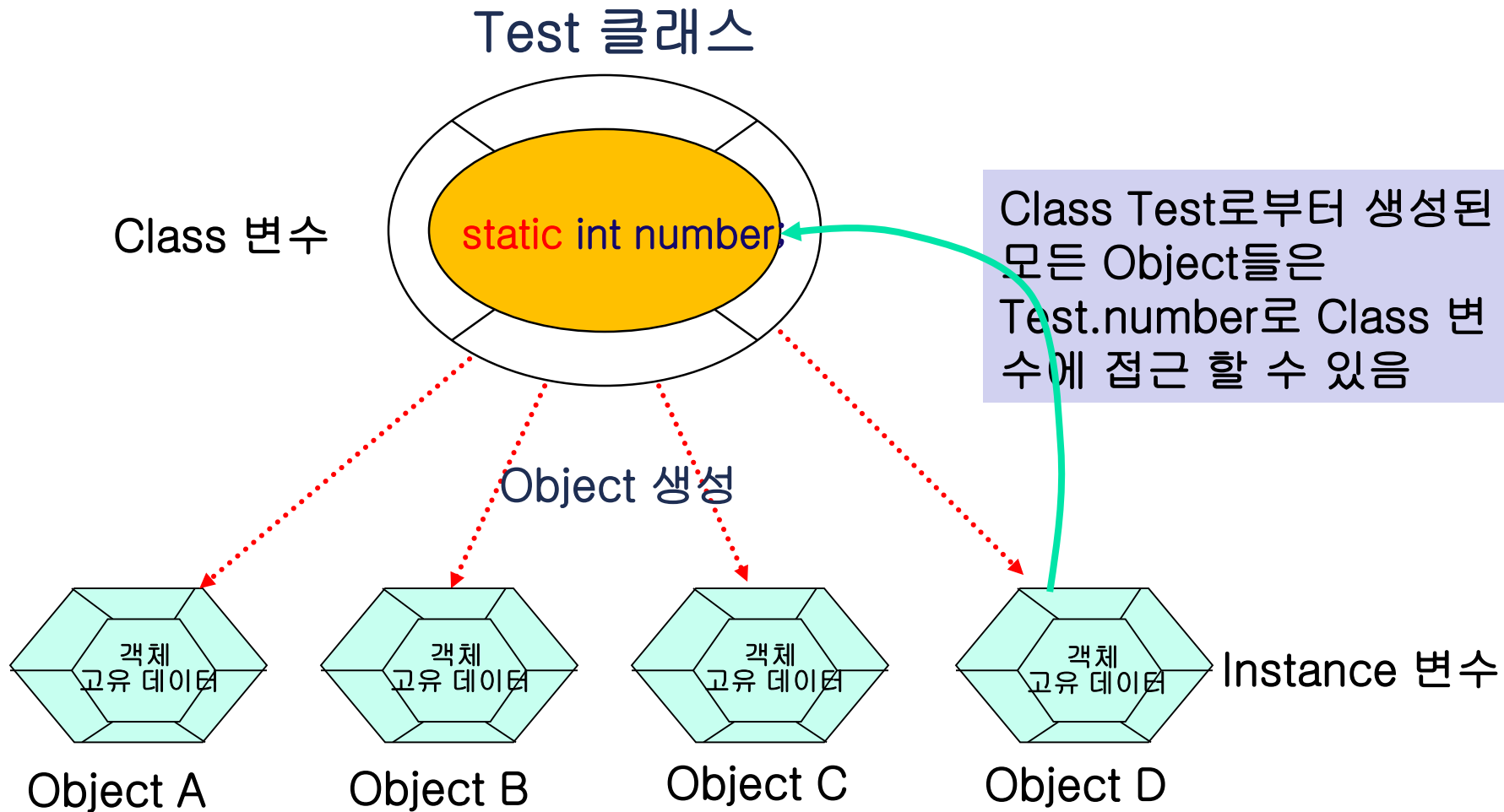


Class 변수(Global Variable)

- Class 변수(Static 변수)와 Instance 변수
 - Class 내에 정의되어 있는 Member는 Variable과 Method가 있고, 이는 다시 Instance 변수와 Instance Method 및 Class 변수와 Class Method로 나눔
 - 일반적으로 Class 내에 선언된 변수와 Method는 대부분이 Instance 변수와 Instance Method 임
 - Class에 대해 Instance를 생성할 때, **각 Instance가 독립적으로 이를 위한 Memory 공간을 확보하기 때문임**
 - Class 변수와 Class Method는 Class의 모든 Object(또는 Instance)가 공유하는 Global 변수와 Global Method를 말함
 - Class 변수 또는 Method는 기존의 Instance 변수와 Method 앞에 '**static**'이라는 Keyword를 명시해 주면 됨



Class 변수(Global Variable)





Class 변수(Global Variable)

- Program 실행 시 static으로 지정된 Class의 Member가 자동으로 Memory에 생성 됨
- Program 종료 시 소멸됨
- Object 생성과 무관함
(Object 생성 前인 Program 시작과 관련이 있음)
- 단 한번 실행됨
- static Member(변수) 접근은 “클래스 이름.멤버”로 접근 가능
- static 사용
 - 클래스 : inner 클래스에서 사용
 - Member 변수 : instance간 공유 목적으로 사용
 - Member 메소드 : Object 생성없이 접근할 목적으로 사용



Class 변수(Global Variable)

■ Class 변수와 Class Method의 선언

[접근권한] **static** 변수;
[접근권한] **static** 메소드;

■ Class 변수와 Class Method의 접근

클래스 이름.클래스 변수
클래스 이름.클래스 메소드()

■ Class(Static) 변수의 활용

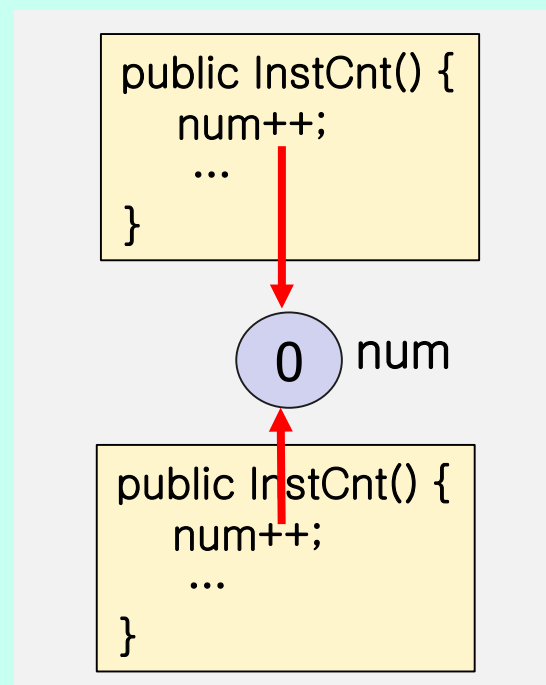
- 동일한 Class의 Instance 사이에서 Data 공유가 필요할 때 static 변수는 유용하게 활용
- Class 내부, 또는 외부에서 참조의 목적으로 선언된 변수는 **static final**로 선언



Class 변수(Global Variable)

- Object 생성과 상관없이 초기화되는 변수
- 하나만 선언되는 변수 (공유 변수)
- public으로 선언되면 누구나 어디서든 접근 가능!

```
public class Main {  
    public static void main(String[ ] args) {  
        Test num1 = new Test( );  
        Test num2 = new Test( );  
    }  
}  
  
class Test {  
    static int num = 0; // Class 변수  
  
    public InstCnt() { // Instance Method  
        num++;  
        System.out.println("객체 생성 : " + num);  
    }  
}
```





Class 변수(Global Variable)

- 어떠한 형태로 접근을 하건, 접근의 내용에는 차이가 없음
- 접근하는 위치에 따라서 접근의 형태가 달라질 수 있음

```
class Test {  
    static int num = 0;    // Class 변수  
  
    public void instCnt( ) {    // Instance Method  
        num++;  
    }  
}
```

```
public class Main {  
    public static void main(String[ ] args) {  
        Test test = new Test( );  
        test.num++;  
        Test.num++;  
        System.out.println("num = " + test.num);  
    }  
}
```

Object의 이름을 이용한 접근
클래스의 이름을 이용한 접근

num = 2



Class 변수(Global Variable)

- JVM은 실행 과정에서 필요한 Class 정보를 Memory에 Loading
- 바로 이 Loading 시점에서 static 변수가 초기화 됨

```
class Test {  
    static int num = 100;    // Class 변수  
  
    public void instCnt() {    // Instance Method  
        num++;  
        System.out.println("인스턴스 생성 : " + num);  
    }  
}
```

```
public class Main {  
    public static void main(String[ ] args) {  
        Test.num = 15;  
        System.out.println(Test.num);  
    }  
}
```

예제에서는 Object 생성이
진행되지 않았음.
즉, Object 생성과 static
변수와는 아무런 상관이
없음



Class 변수(Global Variable)

- static Method는 Instance가 만들어지지 않은 상태에서에서도 호출가능하지만, static Method 내에서 만들어지지도 않은 Instance 변수를 호출한다는 것은 불가능

```
class Test {  
    int var = 0;           // Instance 변수  
    static int num = 1;    // Class 변수  
  
    static int getVar() {  // Class Method  
        return var;       //var은 Instance 변수이므로 참조 불가능  
    }  
  
    public static void main(String[] args){  
        System.out.println(Test.getVar());  
    }  
}
```



Class 변수 예제 0

■ Class 변수 예

```
class Box {  
    int width;  
    int height;  
    int depth;  
    long idNum;  
    static long boxID = 0;    // 클래스 변수 선언  
  
    public Box() {  
        idNum = boxID++;  
        // 생성자가 수행될 때마다 클래스 변수의 값을 증가  
    }  
}
```

Box 클래스로부터 생성되는 모든 객체의 idNum 값은 객체가 생성되는 순서에 따라 유일한 값을 가지게 됨



Class 변수 예제 1

■ 다음 Program의 출력 결과는 ?

```
public class Main {  
    static int test = 100;    // Class 변수  
    int sample;              // Instance 변수  
  
    public static void main(String[] args) {  
        Main count = new Main();    // Instance 생성  
        count.test = 300;  
        count.sample = 500;  
        int test = 200;            // Local 변수  
  
        System.out.println("test = " + test);  
        System.out.println("test = " + Main.test);  
        System.out.println("sample = " + count.sample);  
    }  
}
```

test = 200
test = 300
sample = 500



Class 변수 예제 2

■ 다음 Program의 출력 결과는 ?

```
class Count {  
    private int number;           // Instance 변수  
    private static int counter = 0; // Class 변수  
  
    public static int getCount( ) { // Class Method  
        return counter;  
    }  
  
    public Count( ) {             // 생성자  
        counter++;  
        number = counter;  
    }  
}
```



Class 변수 예제 2



```
public class Main {  
    public static void main(String[ ] args) {  
        System.out.println("Number of counter is " + Count.getCount( ));  
        Count test = new Count( );  
        System.out.println("Number of counter is " + test.getCount( ));  
        Count test1 = new Count( );  
        System.out.println("Number of counter is " + test1.getCount( ));  
    }  
}
```

Number of counter is 0
Number of counter is 1
Number of counter is 2



Class 변수 예제 3



■ 다음 Program의 출력 결과는 ?

```
class Box {  
    private int width ;           // Instance 변수  
    private int height;          // Instance 변수  
    private int depth;           // Instance 변수  
    private long idNum;           // Instance 변수  
    static long boxID = 0;        // Class (static) 변수  
  
    public Box() {                //생성자  
        idNum = boxID++;  
        // 객체가 생성될 때마다 클래스 변수 값을 증가  
    }  
}
```



Class 변수 예제 3

```
public class Main {  
    public static void main(String[] args) {  
        Box box1 = new Box();  
        Box box2 = new Box();  
        Box box3 = new Box();  
  
        System.out.println("box1의 id번호 :"+ box1.idNum);  
        System.out.println("box2의 id번호 :"+ box2.idNum);  
        System.out.println("box3의 id번호 :"+ box3.idNum);  
        System.out.println("전체 박스의 개수:"+ Box.boxID);  
        //클래스 변수 boxID를 클래스 BOX로 바로 접근 가능  
    }  
}
```

box1의 id번호 :0
box2의 id번호 :1
box3의 id번호 :2
전체 박스의 개수:3



Class 변수 예제 4



■ 다음 Program의 출력 결과는 ?

```
public class Test {  
    static int sval = 123;           // Class 변수  
    int ival = 321;                 // Instance 변수  
  
    public static void main(String[] args) {  
        System.out.println(sval);  
        System.out.println(ival);   // 오류 (사용 불가능)  
  
        Test ex1 = new Test();      // Instance 생성  
        Test ex2 = new Test();  
  
        ex1.ival = 456;              // 공유되지 않음  
        System.out.println(ex1.ival);  
        System.out.println(ex2.ival);  
    }  
}
```



Class 변수 예제 5

```
class Circle {  
    static final double PI = 3.1415;    // Class (상수)  
    private double radius;              // Instance 변수  
  
    public Circle(double radius) {      // 생성자  
        this.radius = radius;  
    }  
  
    public void showPerimeter( ) {      // instance Method  
        double peri = 2 * PI * radius;  
        System.out.println("둘레 : " + peri);  
    }  
  
    public void showArea( ) {           // Instance Method  
        double area = PI * radius * radius;  
        System.out.println("넓이 : " + area);  
    }  
}
```

PI 값은 instance
별로 독립적으로
유지할 필요가
없음. 그리고 그
값의 변경도
불필요하다는
특성이 있음



Class 변수 예제 5



```
class Main {  
    public static void main(String[] args) {  
        Circle circle = new Circle(1.2);    // Instance 생성  
  
        circle.showPerimeter( );  
        circle.showArea( );  
    }  
}
```



Class 변수(Global Variable)

- Class 변수(static 변수) 장점
 - Memory를 효율적으로 사용할 수 있음
 - 생성할 때마다 Instance가 힙(heap)에 올라가는 것이 아니라 고정 Memory이므로 효율적
 - 속도가 빠름
 - Object를 생성하지 않고 사용하기 때문에 빠름



Class 변수(Global Variable)

- Class 변수(static 변수) 단점
 - 무분별한 static의 사용은 Memory Leak의 원인이 됨
 - Class 변수인 static 변수는 Class가 생성될 때 Memory를 할당 받고, Program 종료 시점에 반환되므로 사용하지 않고 있어도 Memory가 할당되어 있음
 - 반면에 Object 생성으로 만들어진 Instance는 참조되지 않으면 Garbage Collection에 의해 소멸되므로 Memory 낭비를 방지함
 - 많이 참조되는 static 변수는 Error 발생 시 Debugging이 힘들
 - 큰 Project에서 값이 자주 바뀌는 Object를 static으로 선언하게 되면 예상치 못한 Error가 생길 수 있음



final 변수

- 변하지 않는 상수 값을 가지는 변수
- Keyword final을 사용하여 선언하는 변수
- 종단 변수는 단 한번 초기화 될 수 있으며 그 이후에 그 값은 변경할 수 없음
- 종단 변수는 관례상 대문자를 사용

```
final int PI = 3.14;  
final int MAX = 1000;  
final int MIN;  
MIN = 0; // 단 한번 초기화 가능
```




Field



- Field의 초기화 방법
 - 명시적 초기화
 - 생성자 초기화
 - Instance Initializer Block(초기화 블록)
 - Static Initializer Block(정적 초기화 블록)



Field



■ 명시적 초기화(Explicit Initialization)

```
public class Classroom {  
    public static int capacity = 60; // 60으로 초기화  
    private boolean use = false;    // false로 초기화  
}
```

- 초기값이 있는 경우에 한 줄로 쓸 수 있으므로 간결함
- Field의 초기값이 지정되지 않았다면 Field는 default 값으로 초기화
 - int 형과 같은 숫자 : 0(zero)
 - 논리형 : false
 - 참조형 : null



Field



■ 초기값 대입

선언 예	설명
<pre>int i = 10; int j = 10;</pre>	int형 변수 i를 선언하고 10으로 초기화 int형 변수 j를 선언하고 10으로 초기화
<pre>int i = 10, j = 10;</pre>	같은 타입의 변수는 콤마(,)를 사용해서 함께 선언하거나 초기화 할 수 있음
<pre>int i = 10, long j = 0;</pre>	타입이 다른 변수는 함께 선언하거나 초기화 할 수 없음
<pre>int i = 10; int j = i;</pre>	변수 i에 저장된 값으로 변수 j를 초기화 변수 j는 i의 값인 10으로 초기화 됨
<pre>int j = i; int i = 10;</pre>	변수 i가 선언되기 전에 i를 사용할 수 없음



Field



■ Initialization Block

■ Class 초기화 Block : `static { }`

■ Class 변수의 복잡한 초기화에 사용되며, Class가 Loading될 때 실행

■ Instance 초기화 Block : `{ }`

■ 생성자에서 공통적으로 수행되는 작업에 사용되며 Instance가 생성될 때 마다(생성자보다 먼저) 실행

```
class InitBlock {  
    static { /* 클래스 초기화 블록 */ }  
    ...  
    { /* 인스턴스 초기화 블록 */ }  
    ...  
}
```



Field



■ Initialization Block

- 명시적 초기화가 1차적으로 고려되어야 하고, **명시적 초기화로 해결되지 않을 때 초기화 Block을 사용해야 함**
- 초기화 Block은 명시적 초기화만으로 처리할 수 없는 복잡한 초기화에 사용
- 초기화 Block은 내부적으로는 Method로 처리됨. 자동적으로 호출되는 초기화 Method 임
- 선언되는 영역도 Method처럼 Class 영역에 선언
- Class 초기화 Block은 Class 변수의 복잡한 초기화에 사용
- Class가 Loading되면 Class 변수가 만들어지고, 그 다음에 Class 초기화 Block이 자동적으로 호출되어 Class 변수에 대한 초기화를 수행
- Class 초기화 Block은 Class가 Loading될 때 단 한번만 실행 됨



Field



■ Initialization Block

- Instance 초기화 Block은 생성자처럼 Instance가 생성될 때 마다 자동적으로 호출됨
- Instance 초기화 Block 다음에 생성자가 실행됨
- Instance 변수의 복잡한 초기화는 생성자를 이용하면 되므로 사실 Instance 초기화 Block은 거의 사용되지 않음
- Instance 초기화 Block은 각 생성자에서 공통적으로 수행해야하는 작업을 따로 뽑아서 처리하는데 사용됨
- 각 생성자에서 수행될 공통 부분을 Instance 초기화 Block에 뽑아 넣으면 Code의 중복이 제거됨
- Instance 초기화 Block은 그냥 Block { }을 만들고 그 안에 Code를 적기만 하면 됨
- Class 초기화 Block은 Instance 초기화 Block에 static만 붙이면 됨



Field

- Class 변수 초기화 시점 : Class가 처음 Loading될 때 한번
- Instance 변수 초기화 시점 : Instance가 생성될 때 마다

```
class Test {  
    static int cv = 1;    // 클래스 변수  
    int iv = 1;           // 인스턴스 변수  
    static { cv = 2; }    // 클래스 초기화 블록  
    { iv = 2; }           // 인스턴스 초기화 블록  
    Test() {              // 생성자  
        iv = 3;  
    }  
}
```

Test test = new Test();

클래스 초기화			인스턴스 초기화			
기본값	명시적 초기화	클래스 초기화블록	기본값	명시적 초기화	인스턴스 초기화블록	생성자
cv 0	cv 1	cv 2	cv 2	cv 2	cv 2	cv 2
			iv 0	iv 1	iv 2	iv 3
1	2	3	4	5	6	7



Initialization Block 예제 1

```
class Product {
    static int count = 0;    // 명시적 초기화 (Class 변수)
    int serialNo;           // Instance 변수
    {                       // Instance 초기화 Block
        ++count;
        serialNo = 1000 + count;
    }
    product( ) { }          // 생성자
}

public class Test {
    public static void main(String[] args) {
        Product p1 = new Product();
        Product p2 = new Product();
        Product p3 = new Product();
        System.out.println("p1의 제품 번호는 " + p1.serialNo);
        System.out.println("p2의 제품 번호는 " + p2.serialNo);
        System.out.println("p3의 제품 번호는 " + p3.serialNo);
        System.out.println("생산된 제품의 수는 모두 " + Product.count + "개");
    }
}
```




Initialization Block 예제 2

```
class BlockTest {  
    static int[] arr = new int[10];           // 명시적 초기화 (Class 변수)  
    static {                                   // Class 초기화 Block  
        for (int i = 0; i < arr.length; i++)  
            arr[i] = (int) (Math.random() * 10) + 1;  
    }  
    {                                         // Instance 초기화 Block  
        for (int i = 0; i < arr.length; i++)  
            arr[i] = 100;  
    }  
  
    public static void main(String[] args) {  
        for (int i = 0; i < arr.length; i++)  
            System.out.println((i + 1) + ": " + arr[i]);  
        BlockTest test = new BlockTest();  
        for (int i = 0; i < test.arr.length; i++)  
            System.out.println((i + 1) + ": " + test.arr[i]);  
    }  
}
```



Local Variables

- Method 내에서 선언된 변수
- Method 매개 변수도 Local 변수의 일종
- 자신이 선언된 Method 내에서만 존재하고 Method 내부로부터의 접근만 가능
- Method가 호출될 때 Local 변수 Memory(Stack)가 할당되고, Method가 종료(반환)될 때 그 Memory는 삭제

```
class Box {  
    int width=0, length=0, height=0;  
    ...  
    public int getVolume()  
    {  
        int volume;  
        volume = width*length*height;  
        return volume;  
    }  
}
```

필드: 전체 클래스 안에서 사용가능

지역 변수

지역 변수 volume의 사용 범위



Local Variables

- Local 변수는 선언된 Method 안에서만 사용될 수 있음
- Local 변수를 초기화하지 않고 사용하면 오류
- Field는 Class 전체를 통하여 사용될 수 있으므로 Class 안의 모든 Method에서 사용 가능

```
class BugClass {  
    public int getSum() {  
        int sum;  
        for (int i = 0; i < 10; i++)  
            sum += i;  
        return sum;  
    }  
}
```

초기화되지 않은 지역변수를
사용하면 오류 발생

실행결과

Exception in thread "main" java.lang.Error: Unresolved compilation problems:
The local variable sum may not have been initialized



Local Variables

■ Local 변수 Scope

- Local 변수 선언 지점부터 변수가 선언된 Block 끝까지

```
void sumAndProduct(int n) { // 매개 변수도 Local 변수에 속함
    int sum = 0;           // Local 변수
    int product = 1;       // Local 변수

    for (int i = 1; i <= n; ++i) {
        sum += i;
        product *= i;
    }
    System.out.println("Sum of the first " + n + " positive integers is "
        + sum);
    System.out.println("Product of the first " + n + " positive integers is "
        + product);
}
```

매개 변수 n의 유효 범위는
Method 전체

변수 i는 loop 실행이 끝나자마자
Memory에서 제거



Local Variables



■ Local 변수 Scope

```
if (purchase > 200) {  
    double discount = .20 * purchase;  
    double discountPrice = purchase - discount;  
    tax = .05 * discountPrice;  
    total = discountPrice + tax;  
} else {  
    tax = .05 * purchase;  
    total = purchase + tax;  
}
```

discount와 discountPrice Local 변수의 유효 범위는 if 블록의 끝까지이고, else 블록 내에서는 두 변수에 접근할 수 없고 알지도 못함



Local Variables

■ Local 변수 Scope

- Local 변수의 scope내에는 같은 이름의 Local 변수가 있을 수 없음

```
Rectangle r = new Rectangle(5, 10, 20, 30);  
if (x >= 0) {  
    double r = Math.sqrt(x);  
    // Error-여기에서 r이라는 다른 지역 변수를 선언할 수 없음  
    ...  
}
```



Local Variables



■ Local 변수 Scope

- Scope가 겹치지 않는다면 같은 이름의 Local 변수를 사용할 수 있음

```
if (x >= 0) {  
    double r = Math.sqrt(x);  
    ...  
} // Scope of r ends here  
else {  
    Rectangle r = new Rectangle(5, 10, 20, 30);  
    // OK-여기에서 다른 r을 선언하는 것이 합법적  
    ...  
}
```



Local Variable



■ Local 변수 Scope

- Local 변수와 Field가 같은 이름을 가질 때는 Local 변수가 Field를 가림(shadow)

```
public class Coin {  
    private String name;  
    private double value; // Field (Instance 변수)  
  
    public double getExchangeValue(double exchangeRate) {  
        double value; // Local variable  
        ...  
        return value; // value가 아닌 value가 반환됨  
    }  
}
```




Local Variable



■ Local 변수 Scope

■ 가려진 Field에 접근하는 방법 - this 사용

```
class Coin {  
    private String name;        // Field (Instance 변수)  
    private double value;  
  
    public Coin(double value, String name) {  
        this.value = value;  
        this.name = name;  
    }  
}
```



Field와 Local Variable



■ Field

- Class안에서 선언되는 Member 변수, Instance 변수라고도 함
- Class 변수 : static이 붙은 변수
- Instance 변수 : static이 붙지 않은 변수

■ Local Variable

- Constructor나 Method 그리고 Block 안에서 선언되는 변수

■ Parameter

- Method 선언에서의 변수
- 일종의 Local Variable



Field와 Local Variable

- JAVA에서의 변수는 Class 변수, Instance 변수, Local 변수

```
public class test {
```

```
    int test;           // Instance 변수
    static int sample;  // Class 변수    //Member 변수
```

```
    void method() {
        int test;       // Local 변수 (우선 사용)
    }
}
```

- Instance 변수와 Local 변수는 이름이 같아도 됨
- 같은 변수의 이름인 test를 출력할 때 같은 구역에 Local 변수가 있다면 Local 변수가 우선 사용됨



Field와 Local Variable

- 변수의 종류를 결정 짓는 중요한 요소는 “변수의 선언 위치”

Class 영역

```
public class Classroom {  
    public static int capacity = 60; // Class 변수  
    private boolean use = false;    // Instance 변수  
  
    void start(int s) {              // 매개 변수  
        int t = 0;                  // Local 변수  
        .....  
    }  
}
```

Method 영역

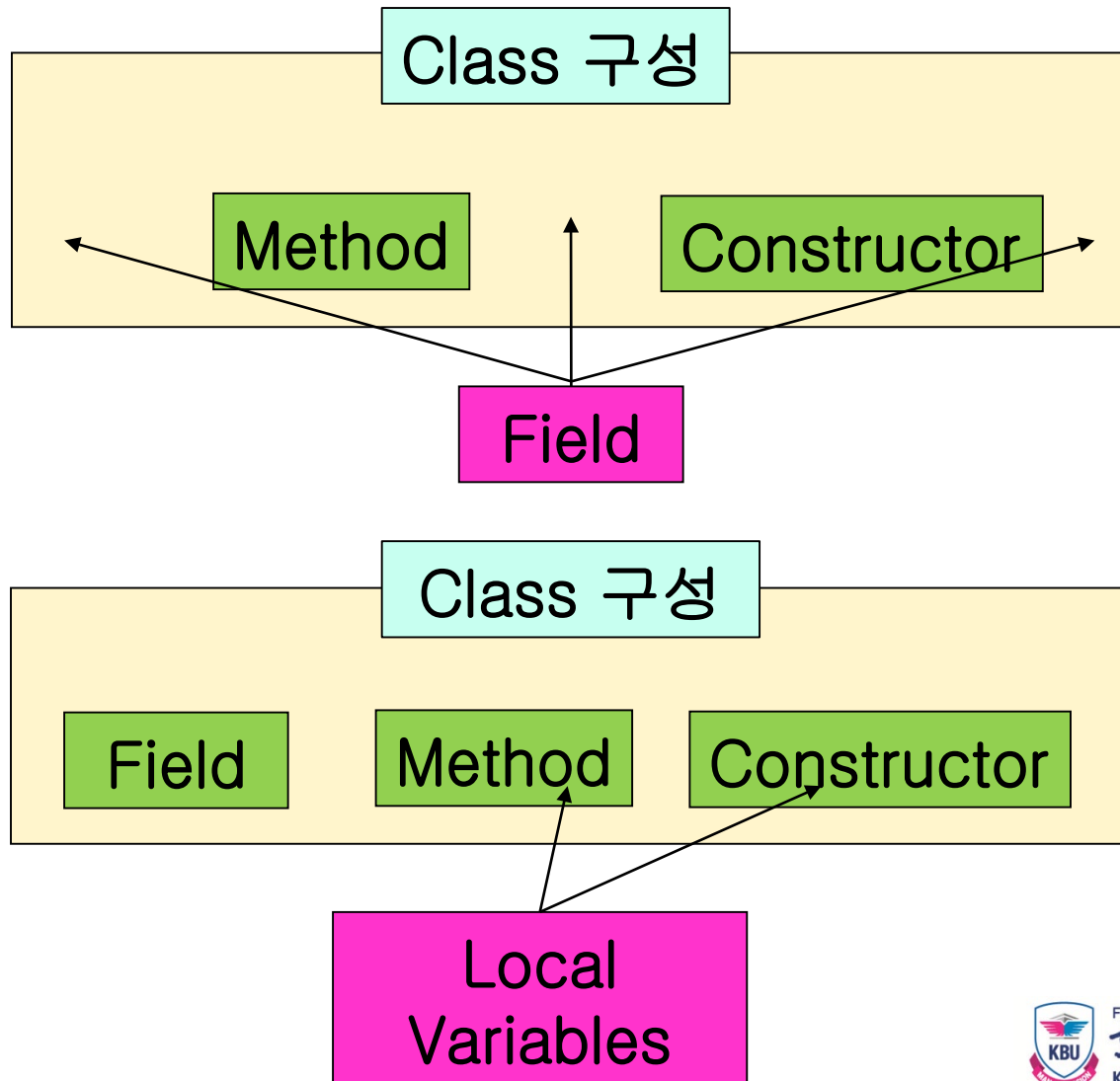
- Field는 Class 내부 중괄호 { }안에서 어디든지 선언이 가능하지만 Method와 Constructor의 요소 밖에만 선언 됨
- Local Variable는 Method와 생성자 중괄호 { }안에 선언이 되어 함





Field와 Local Variable

■ Field와 Local Variable의 선언 위치





Field와 Local Variable

- Fields (상태 변수)

- Local variables (지역 변수)

- Parameter (파라미터)

Method에 속하며
Method가 실행되는
동안에만 존재함

```
public class BankAccount {  
    private double balance;    // Instance 변수  
  
    public void deposit(double amount) {    // parameter  
        double newBalance = balance + amount;    // Local 변수  
        balance = newBalance;  
    }  
}
```



Field와 Local Variable



변수의 종류		선언 위치	정의(생성) 시기
Field (Member 변수)	Class 변수 (공유 변수) (정적 변수)	Class 영역(static)	Class가 Memory에 올라 갈 때
	Instance 변수	Class 영역	Instance가 생성 될 때 (heap 영역)
Local Variable		Class 영역 이외의 영역 (Method, 생성자, 초기화 Block 내부)	변수 선언문이 생성 되었을 때 (stack 영역)
Parameter		Method 선언 Header 부분	Method 실행 시 (stack 영역)

■ Class 변수 사용

- “클래스 이름.클래스 변수” 형식을 사용



변수의 유효 Scope



- 변수의 유효 범위는 선언된 변수가 동일한 의미를 갖는 영역을 의미
- 변수의 유효 범위는 변수에 따라 다음과 같이 구분함
 - Member 변수
 - Class 정의 시작 부분에 선언되며, 특정 Method에 속하지 않는 변수를 Member 변수라 함
 - 모든 Member 변수들은 Class 내의 모든 Method에서 사용 가능
 - 단 Method 내에서 동일한 이름의 변수가 선언 되었을 때는 접근 불가능 (**this**를 사용)



변수의 유효 Scope



- Method의 매개 변수(Parameters)와 Method Local 변수
 - Method Local 변수는 변수가 정의된 Method 내에서만 사용 가능
 - Method 매개 변수는 Method 호출 시 전달되는 변수로, Method 내에서만 유효한 Local Variable
 - main() Method는 String args[]라는 매개 변수를 가짐
 - main() Method 내에서만 사용할 수 있음



변수의 유효 Scope



■ Block Local Variable

- Method는 여러 개의 Block을 포함 할 수 있으며, Block 안에서 선언된 변수를 Block Local Variable라고 함
- Block Local Variable는 Block 안으로 들어갈 때 할당되고, Block을 빠져 나올 때 소멸됨
- 선언된 Block 안에서만 유효함
- Block의 경우, Local Variable의 이름은 비 Local 변수와 동일 해서는 안됨



변수의 유효 Scope



```
class Test {  
    static int s = 40;    // Class 변수  
    int tot = 20;        // Instance 변수  
  
    void f() {            // Instance Method  
        short s = 300;    // Local 변수  
        int tot = 100;    // Local 변수  
        System.out.println( "s=" + s + " tot=" + tot );  
    }  
    void g() {            // Instance Method  
        System.out.println( "s=" + s + " tot=" + tot );  
        {  
            int s = 500;    // Block Local 변수  
            System.out.println( "s=" + s + " tot=" + tot );  
        }  
        System.out.println( "s=" + s + " tot=" + tot );  
    }  
}
```



변수의 유효 Scope

```
public static void main( String args[] ) {  
    Test test = new Test();  
    test.f();  
    test.g();  
}  
}
```

s= 300	tot = 100
s= 40	tot = 20
s= 500	tot = 20
s= 40	tot = 20

Method f()내에서는 Member 변수와 동일한 이름의 변수가 선언되어 있으며, g()내에서 변수 s는 Block이 시작할 때 별도의 공간이 할당되어 Block이 종료되면 회수됨



final



- final의 의미는 최종적(변경 금지)이란 뜻을 가지고 있음
- final을 사용하는 이유
 - 보안과 설계 부분을 명확하게 하기 위함



final



■ 용도

■ Member 변수 - 상수

- final Field는 초기값이 저장되면 최종적인 값이 되어 Program 실행 도중에 수정을 할 수 없음
- 모든 Class에서 동일한 값을 공유해야 함
- 관습적으로 대문자로 상수 명을 지정

```
public static final TOTAL_NUM = 40 ;
```

■ Class - 상속 금지

- final Class의 모든 Method는 final로 명시되지 않아도 묵시적으로 final로 인식하게 됨

■ Object - 재 생성 금지

■ Member Method - Overriding 금지



final



■ final Class

■ 더 이상 Class 상속 불가능

```
final class FinalClass {
```

```
.....  
}
```


 class DerivedClass extends FinalClass { // 컴파일 오류

```
.....  
}
```

■ final Method

■ 더 이상 Overriding 불가능

```
public class SuperClass {  
    protected final int finalMethod() { ... }  
}
```

 class SubClass extends SuperClass {
 protected int finalMethod() { ... } // 컴파일 오류, 오버라이딩 할 수 없음
}



final



■ final Class

```
final class University{  
    String name;  
}
```

//상속 불가능

```
class Kyungbok extends University{  
  
}
```

- Class에 final을 사용하게 되면 그 Class는 최종 상태가 되어 더 이상 상속이 불가능
- final Class이어도 Field는 Setter() Method를 통하여 변경은 가능



final



■ final Method

```
class University{
    String name = "대학명";
    public final void print() {
        System.out.println("대학 : " + name + " 입니다.");
    }
}

class Kyungbok extends University{
    name = "경북대학교";
    public void print() {    // Method Overriding 불가능
    }
}
```

- Method에 final을 사용하게 되면 상속받은 Class에서 부모의 final Method를 재정의 할 수 없음
- 자신이 만든 Method를 변경할 수 없게끔 하고 싶을 때 사용되며 System의 Core부분에서 변경을 원치 않는 Method에 많이 구현되어 있음



final

- final Field, 상수 선언
 - 상수를 선언할 때 사용

```
class SharedClass {  
    public static final double PI = 3.14;  
}
```

- 상수 Field 선언 시에 초기 값을 지정하여야 함
- 상수 Field는 실행 중에 값을 변경할 수 없음

```
public class FinalFieldClass {  
    final int ROWS = 10; // 상수 정의, 이때 초기 값(10)을 반드시 설정  
  
    void f() {  
        int [] intArray = new int [ROWS]; // 상수 활용  
        ROWS = 30; // 컴파일 오류 발생, final 필드 값을 변경 불가  
    }  
}
```

오류



final



■ final 속성(Member 변수)

```
class FinalMember {  
    final int a=10;  
    public void setA(int a){  
        this.a=a;  
    }  
}  
  
public class Main{  
    public static void main(String[] args) {  
        FinalMember ft = new FinalMember( );  
        ft.setA(100);  
        System.out.println(ft.a);  
    }  
}
```



final



■ Overriding 불가능한 final Method

```
class FinalMethod{
    String str="Java ";

    // final 붙이면 Sub Class에서 Overriding이 불가
    public final void setStr(String s) {
        str=s;
        System.out.println(str);
    }
}

class FinalEx extends FinalMethod{
    int a=10;    // final 붙이면 밑에서 a값 대입 불가
    public void setA(int a) {
        this.a=a;
    }
}
```



■ final Object

```
class University{
    String name = "대학명";
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
}

public class Final {
    public static void main(String[] args) {
        final University university = new University();
        // university = new University(); //객체를 한번 생성했다면 재생성 불가능
        university.setName("경북대학교"); //클래스의 필드는 변경가능
    }
}
```



final



■ 더 이상의 상속을 허락하지 않는 클래스 설계

```
final class FinalClass{  
    String str="Java ";  
    public void setStr(String s){  
        str=s;  
        System.out.println(str);  
    }  
}
```

```
class FinalEx extends FinalClass{  
    int a=10;  
    public void setA(int a) {  
        this.a=a;  
    }  
    public void setStr(String s){  
        str+=s;  
        System.out.println(str);  
    }  
}
```

```
public class Main{  
    public static void main(String[] args) {  
        FinalEx fe= new FinalEx( );  
    }  
}
```



Access Control

- 다른 Class가 특정한 Field나 Method에 접근하는 것을 제어하는 것



Member 변수에 대한 접근 제어



Access Control

■ 종류

public : 모든 Class에서 접근이 가능

protected : 같은 Package(folder)에 있는 Class와 상속
관계의 Class들만 접근 가능

default : 같은 Package에 있는 Class들만
접근 가능

private : 같은 Class내에서만 접근 가능



Access Control



■ Member 변수의 접근 지정자

접근 지정자	동일 클래스	서브 클래스	동일 패키지	모든 클래스
private	○	X	X	X
package (생략된 경우)	○	X	○	X
protected	○	○	X	X
public	○	○	○	○

- 접근 지정자가 생략된 경우에는 동일 Package에서만 접근 가능



Access Control



- Member 변수의 접근 지정자

- public

- public으로 명시된 변수는 모든 Class로부터 접근 가능
 - 같은 Class, 자식 Class, 또는 같은 Package나 다른 Package 내에 있는 어떤 Class에서도 접근할 수 있음
 - 공통적으로 많이 쓰이는 변수들 앞에 이 public 예약어를 붙임

- protected

- 같은 Class, 자식 Class, 또는 같은 Package 내의 모든 Class에서만 접근 가능



Access Control



■ Member 변수의 접근 지정자

■ private

- 동일한 Class 내에서만 접근할 수 있음
- 이 Keyword는 Class가 내부적인 목적으로 변수를 사용할 때 쓰임
- 외부 Class에서는 전혀 사용할 수 없고 참조할 수 없어, 완벽한 캡슐화, 정보의 은폐를 가능하게 함

■ package(생략된 경우 : default)

- 접근 지정자를 생략한 경우 임
- 동일 Package의 Class에서만 접근 가능
- **protected 접근 지정자와는 달리 하위 Class, 즉 상속 받은 하위 Class가 다른 Package에 있으면 접근할 수 없음**



Access Control

■ 접근 제어의 종류

- Class 수준에서의 접근 제어

- Member 변수 수준에서의 접근 제어





Access Control



■ 사용법

- Class – public, default만 가능
- Member 변수 – 모든 접근 지정자 가능
- Member Method – 모든 접근 지정자 가능
- 생성자 – 모든 접근 지정자가 가능
- Local 변수에는 접근 지정자를 사용할 수 없음

```
public class Student {  
    public String name;  
    protected int grade;  
    private String address;  
    String tel;
```

```
    public Student() {  
    }
```

```
    private void method() {  
    }  
}
```



Access Control



■ Class 수준에서의 접근 제어

■ public

■ 다른 모든 Class가 사용할 수 있는 공용 Class

■ package

■ 수식자가 없으면 같은 Package안에 있는 Class들만이 사용

■ Member 변수 수준에서의 접근 제어

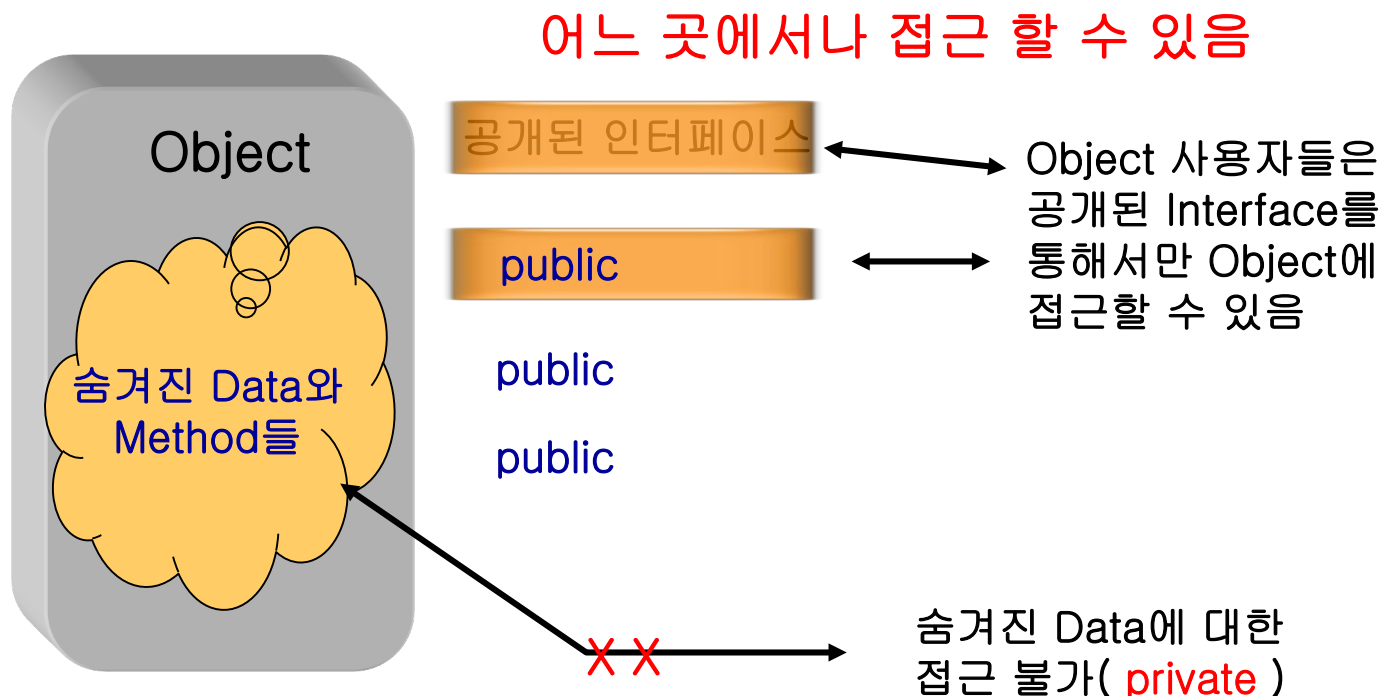
분류	접근 지정자	Class 내부	같은 Package 내의 Class	다른 모든 Class
전용 Member	private	○	X	X
Package Member	없음	○	○	X
공용 Member	public	○	○	○



Access Control

■ 접근 지정자 종류와 권한(Public)

- public으로 선언된 Instance 변수는 소속된 Class가 접근 가능 하면 항상 접근 가능하다는 것을 의미





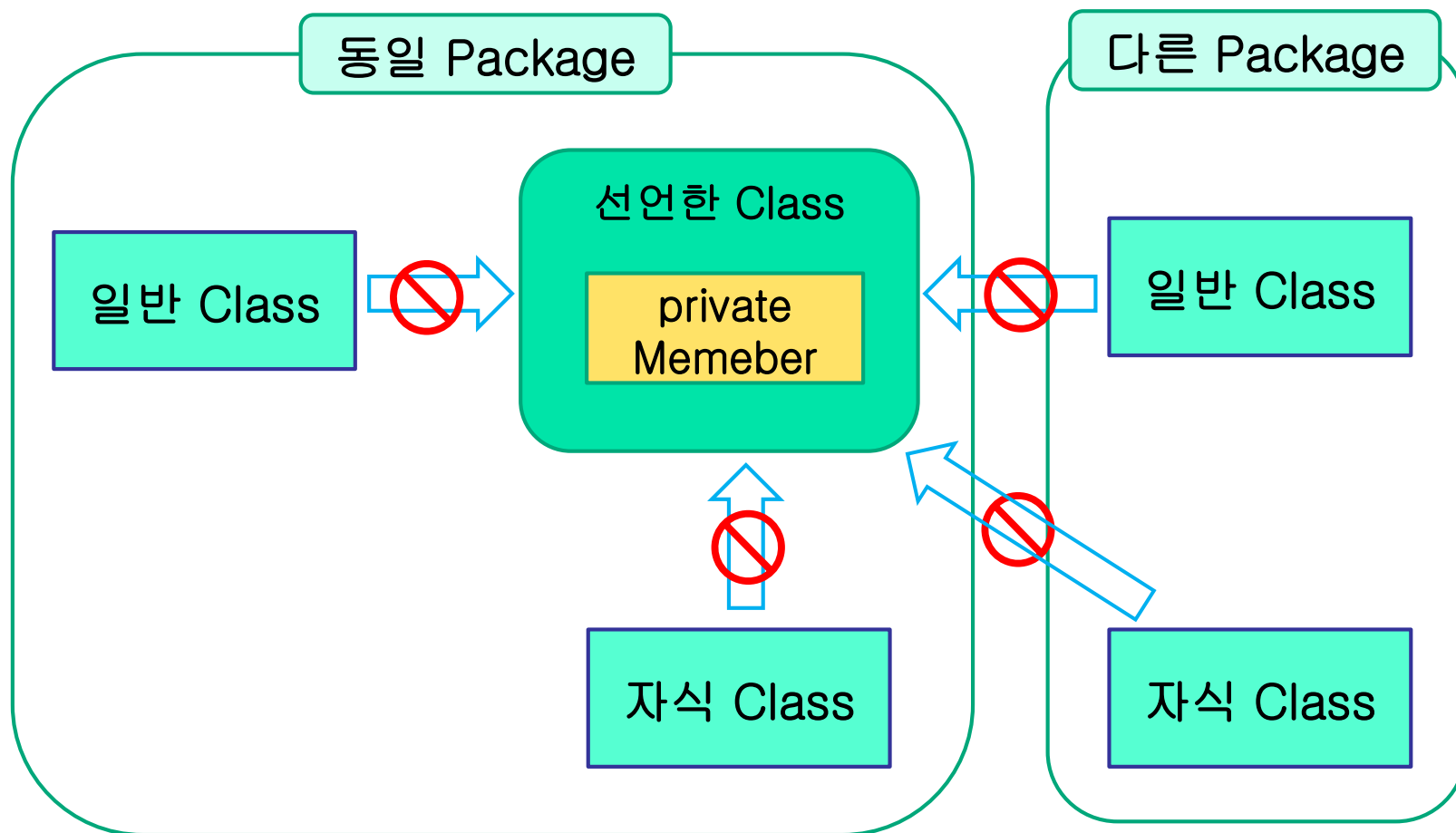
Access Control



- 접근 지정자 종류와 권한(private)
 - private로 선언된 Instance 변수는 소속된 Class에서만 사용
 - private로 선언된 변수는 그 Class 내부에서만 사용, 하위 Class에도 상속되지 않음
 - Class 외부에서 private로 선언된 Instance 변수에 접근하려면 Error가 발생
 - 외부 Class에서 이 변수에 직접 접근할 수 없음

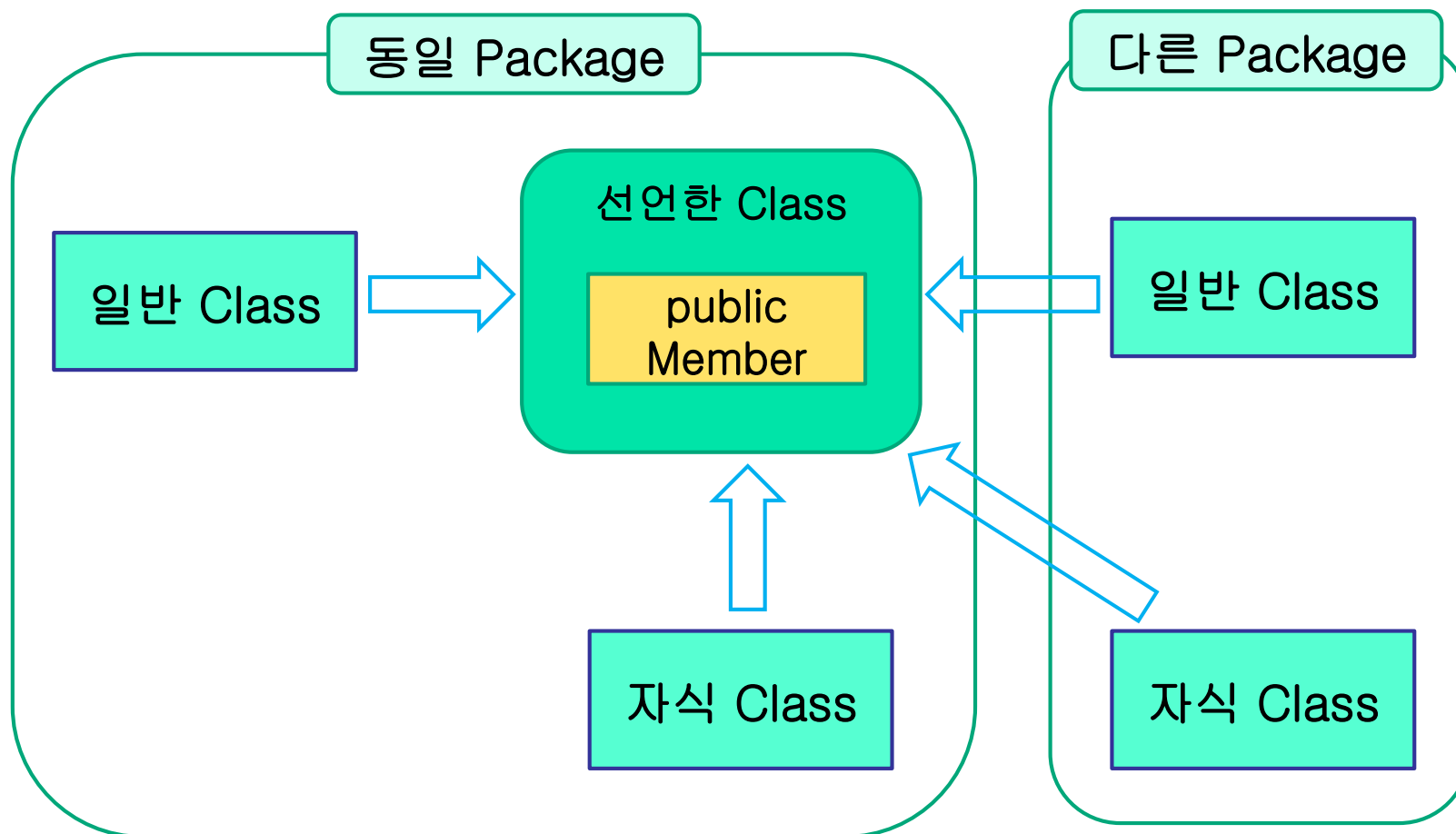


Access Control



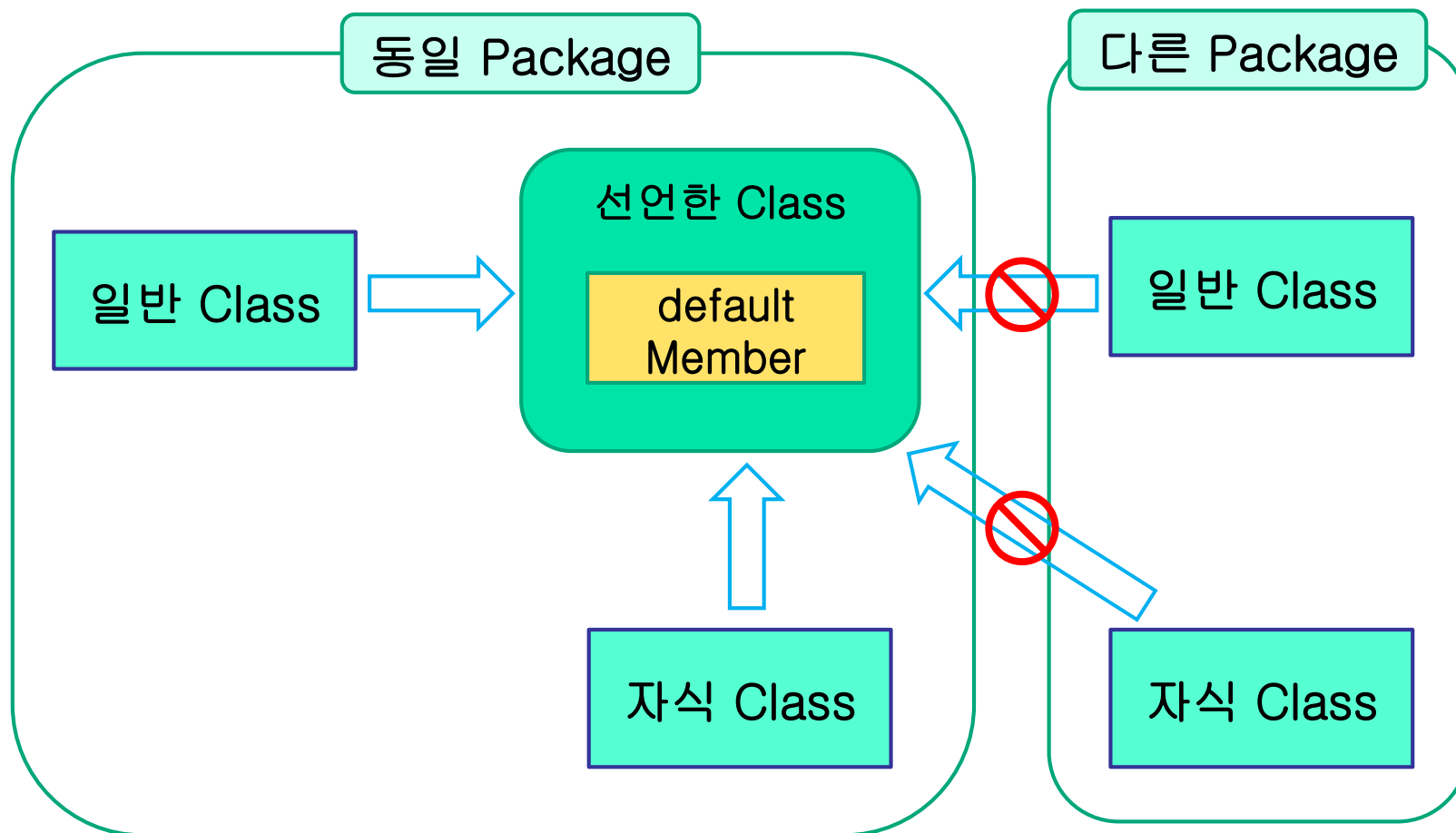


Access Control



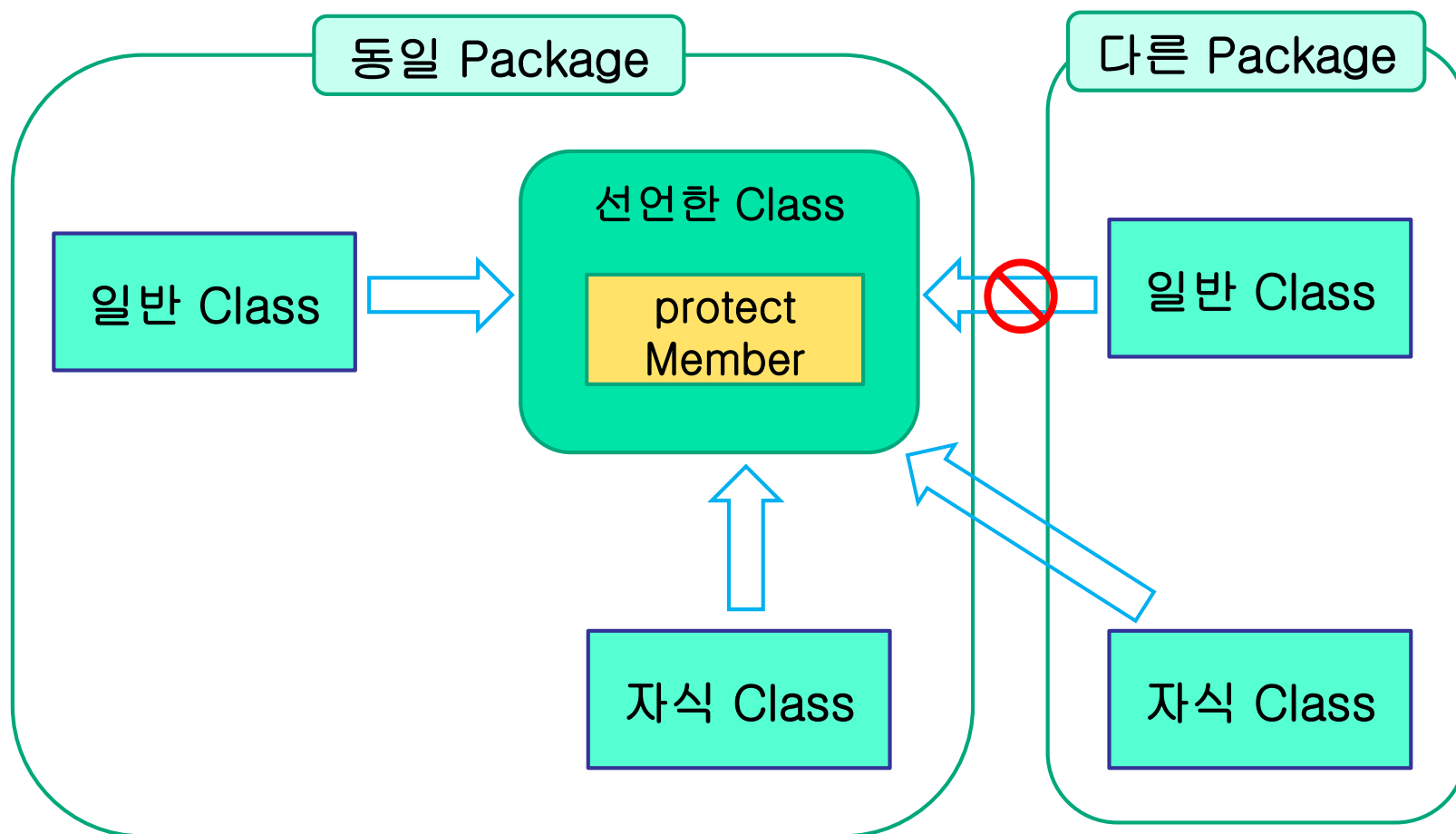


Access Control





Access Control





Access Control 예제

- “public”으로 선언된 Instance 변수는 소속된 Class가 접근 가능하면 항상 접근 가능

```
class Box {  
    public int width;  
    public int height;  
    public int depth;  
    public long idNum;  
    static long boxID = 0;  
    public Box() {                // 생성자  
        idNum = boxID++;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Box myBox = new Box();  
        myBox.width = 7;    // 접근 가능  
        myBox.depth = 20;  // 접근 가능  
        .....  
    }  
}
```



Access Control 예제

- “private”으로 선언된 Instance 변수는 소속된 Class 내에서만 사용 가능

```
class Box {  
    private int width;    // private로 선언 이 클래스 내부에서만 사용  
    private int height;  
    private int depth;  
    .....  
}  
  
public class Main {  
    public static void main(String args[]) {  
        Box myBox = new Box(10, 20, 30);  
        myBox.width = 7;    // 에러 발생  
        .....  
    }  
}
```



Access Control 예제

```
public class Empolyee {  
    public String name;           // public로 선언 (공용 Member)  
    int age;                      // package로 선언 (Package Member)  
    private int salary;          // private로 선언 (전용 Member)  
  
    public Empolyee(String name, int age, int salary) { // 생성자  
        this.name = name;  
        this.age = age;  
        this.salary = salary;  
    }  
  
    public String getName( ) {    //public로 선언  
        return(name);  
    }  
    private int getAge( ) {      // private로 선언  
        return(age);  
    }  
}
```



Access Control 예제



```
public int getSalary( ) {  
    return(salary);  
}  
  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Empolyee man = new Empolyee("홍길동", 0, 3000);  
        man.salary = 30000;        // 오류 : private 변수  
        man.age = 26;  
        int age = man.getAge( );    // 오류 : private 메소드  
        int salary = man.getSalary( );  
        System.out.printf("이름 : %s, 나이 : %d, 급여 : %s\\n",  
                           man.getName( ), man.age, salary);  
    }  
}
```




Access Control 예제

- “protected”로 선언된 Instance 변수는 소속 Class의 하위 Class와 소속 Class와 같은 Package의 Class에서만 사용 가능

```
class Box {  
    private int width;  
    private int height;  
    private int depth;  
    protected int count; //이 클래스와 이 클래스의 하위 클래스에서 사용  
    .....  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Box myBox = new Box();  
        myBox.count = 7; // 접근 가능  
        .....  
    }  
}
```

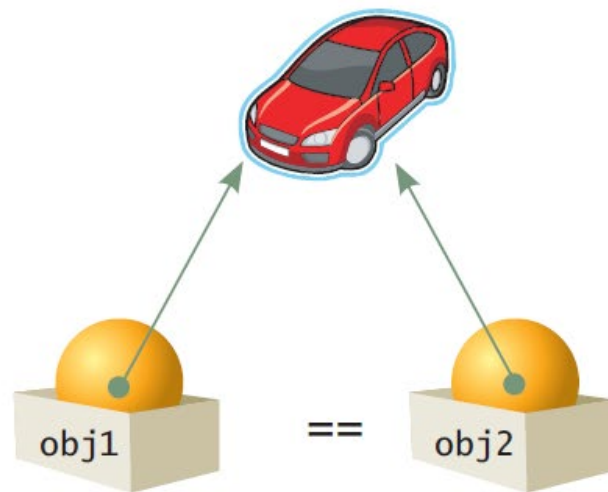


변수와 변수의 비교

■ “변수1 == 변수2”의 의미



기초형 변수의 경우
값이 같으면 true



참조형 변수의 경우
같은 객체를 가리키면 true

- 기초형 변수의 경우, 값이 일치하면 true 반환
- 참조형 변수의 경우, 객체의 내용이 같다는 의미가 아니라 두개의 변수가 같은 객체를 가리키고 있으면 true
- 내용이 같은지를 검사하려면 equals() 사용