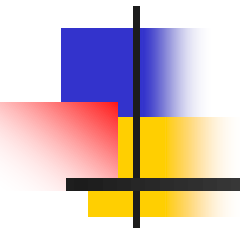


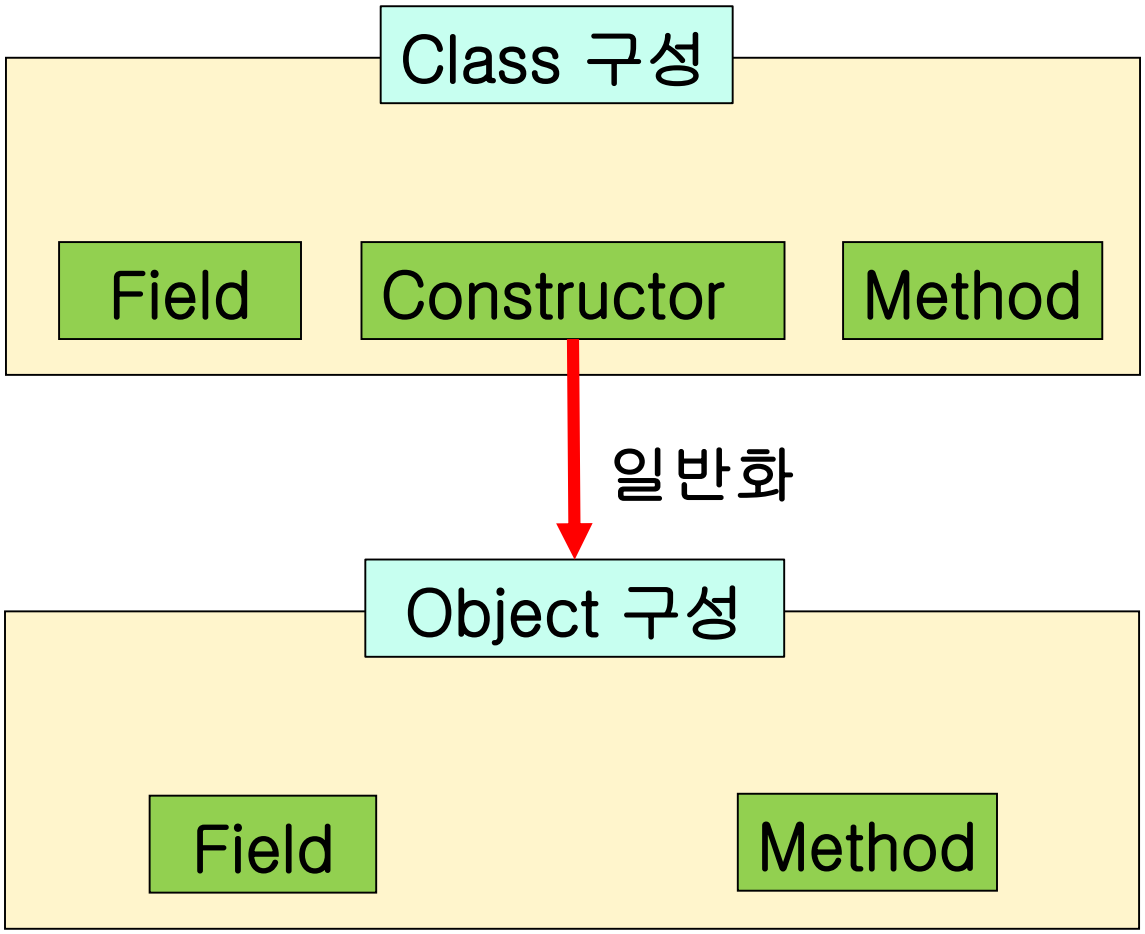
Field

경복대학교
소프트웨어융합과
배희호 교수

QUIZ

1. 객체는 ___속성___과 ___동작___을 가지고 있다.
2. 자동차가 객체라면 클래스는 ___설계도___이다.

Class와 Object



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

- 변수의 종류를 결정 짓는 중요한 요소는 "변수의 선언 위치"

클래스 영역

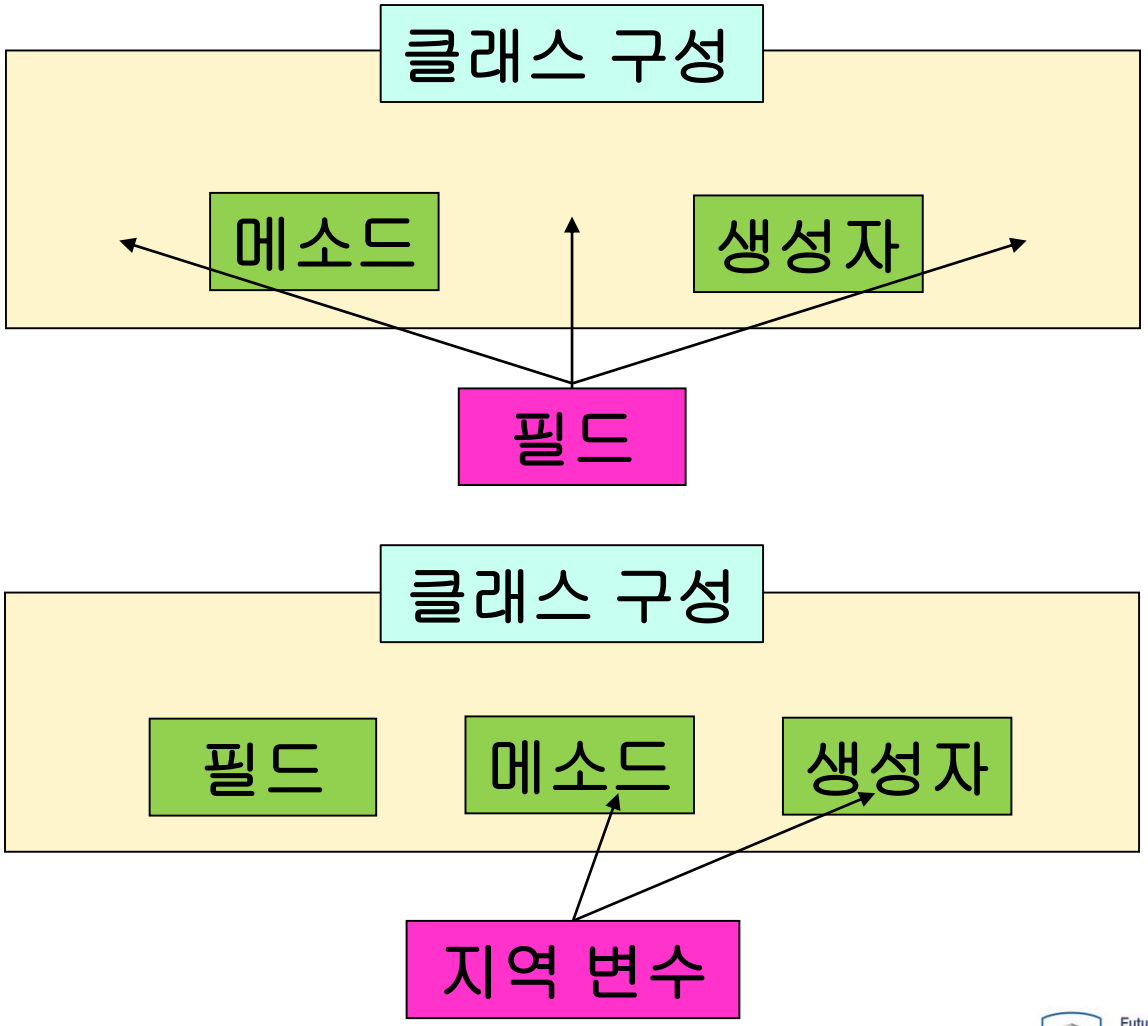
```
public class ClassRoom {
    public static int capacity = 60;   // 클래스 변수
    private boolean use = false;       // 인스턴스 변수
   void start(int s) {                 // 매개 변수
            int t = 0;                 // 지역 변수
              ……
   }
}
```

메소드 영역

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

Field와 Local Variable

- Field와 Local Variable의 선언 위치



Field와 Local Variable

- Fields (상태 변수)
- Local variables (지역 변수)
  - Parameter (파라미터)

메소드에 속하며
메소드가 실행되는
동안에만 존재함

```
public class BankAccount  {
    private double balance;        // Field

    public void deposit(double amount) {     // parameter
       double newBalance = balance + amount;   // 지역 변수
       balance = newBalance;
    }
}
```

Field와 Local Variable

- JAVA에서의 변수는 클래스 변수, 인스턴스 변수, 지역 변수

```
public class test {
        int test;                  // 인스턴스 변수
        static int sample;         // 클래스 변수      //멤버 변수

        void method() {
              int test;            // 지역 변수
        }
}
```

- 인스턴스 변수와 지역 변수는 변수 이름이 같아도 됨
- 같은 변수의 이름인 test를 출력할 때 같은 구역에 지역 변수가 있다면 지역 변수가 우선 사용

Field와 Local Variable

변수의 종류		선언 위치	정의(생성) 시기
Field (Member 변수)	Class 변수 (공유 변수) (정적 변수)	클래스 영역(static)	클래스가 메모리에 올라 갈 때
	Instance 변수	클래스 영역	Instance가 생성 될 때 (heap)
Local Variable		클래스 영역 이외의 영역 (메소드, 생성자, 초기화 블록 내부)	변수 선언문이 생성 되었을 때 (stack 영역)
Parameter		메소드 선언 헤더 부분	메소드 실행 시 (stack 영역)

- Class 변수 사용
  - “클래스 이름.클래스 변수” 형식을 사용

Field

- Field 선언
  - Field의 접근 지정자는 어떤 클래스가 Field에 접근할 수 있는지를 표시
    - public : 이 Field는 모든 클래스로부터 접근 가능
    - private : 클래스 내부에서만 접근 가능
    - 객체 지향 캡슐화 원칙에 따라 Field는 private로 선언하는 것이 좋음

Field

- 선언과 동시에 초기화 가능(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
  - 초기화 블록 (initialization block)
  - 생성자를 사용하는 방법 (추후 학습)

Field

초기값 대입

선언 예	설명
int i = 10; int j = 10;	int형 변수 i를 선언하고 10으로 초기화 한다 int형 변수 j를 선언하고 10으로 초기화 한다
int i = 10, j = 10;	같은 타입의 변수는 컴마(,)를 사용해서 함께 선언하거나 초기화 할 수 있다
int i = 10, long j = 0;	타입이 다른 변수는 함께 선언하거나 초기화 할 수 없다
int i = 10; int j = i;	변수 i에 저장된 값으로 변수 j를 초기화 한다 변수 j는 i의 값인 10으로 초기화 된다
int j = i; int i = 10;	변수 i가 선언되기 전에 i를 사용할 수 없다

Field

- Field는 정의된 위치에 상관없이 클래스 안에서는 어느 곳에서도 사용 가능

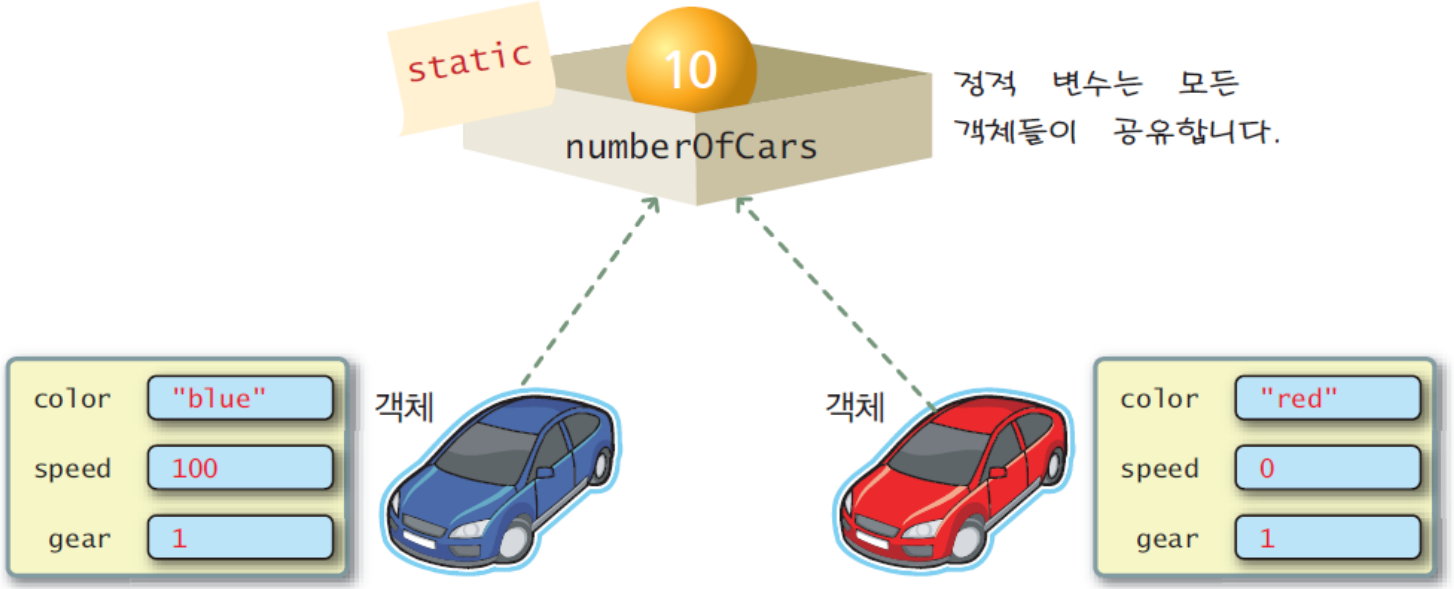
```
public class Date {
        public void printDate() {
            System.out.println(year + “.” + month + ”.” + day);
        }

        public int getDay( ) {
            return day;
        }

        private int year;
        private String month;
        private int day;
}
```

Class 변수(Global Variable)

- Instance Variable
  - 객체마다 하나씩 있는 변수
- Static Variable = Class 변수
  - 하나의 클래스에 하나만 존재
  - 그 클래스의 모든 객체들에 의하여 공유
  - Class 변수를 정의할 때 맨 앞에 static 키워드를 붙임



Class 변수(Global Variable)

- Class 변수(Static 변수)와 Instance 변수
  - 클래스 내에 정의되어 있는 Member는 Variable과 Method가 있고, 이는 다시 Instance 변수와 Instance 메소드 및 Class 변수와 Class 메소드로 나눔
  - 일반적으로 클래스 내에 선언된 변수와 메소드는 대부분이 Instance 변수와 Instance 메소드 임
  - 클래스에 대해 Instance를 생성할 때, 각 Instance가 독립적으로 이를 위한 Memory 공간을 확보하기 때문임
  - Class 변수와 Class 메소드는 클래스의 모든 Object(또는 Instance)가 공유하는 Global 변수와 Global 메소드를 말하며 Class 변수 또는 메소드는 기존의 Instance 변수와 메소드의 앞에 'static'이라는 Keyword를 명시해 주면 됨

Class 변수(Global Variable)

- Program 실행 시 static으로 지정된 클래스의 member가 자동으로 Memory에 생성 됨
- Program 종료 시 소멸됨
- 객체 생성과 무관함
  (객체 생성 前인 Program 시작과 관련이 있음)
- 단 한번 실행됨
- static 멤버(변수) 접근은 “클래스 이름.멤버”로 접근 가능

- static 사용
  - 클래스 : inner 클래스에서 사용
  - 멤버 변수 : instance간 공유 목적으로 사용
  - 멤버 메소드 : 객체 생성없이 접근할 목적으로 사용

Class 변수(Global Variable)

- Class 변수와 Class 메소드의 선언

```
[접근권한] static 변수;
[접근권한] static 메소드 ;
```

- Class 변수와 Class 메소드의 접근

```
클래스 이름.클래스 메소드()
클래스 메소드()
```

- Class(Static) 변수의 활용
  - 동일한 클래스의 Instance 사이에서 Data 공유가 필요할 때 static 변수는 유용하게 활용
  - 클래스 내부, 또는 외부에서 참조의 목적으로 선언된 변수는 static final로 선언

Class 변수(Global Variable)

```
class Circle {
    static final double PI = 3.1415;
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public void showPerimeter( ) {
        double peri = 2 * PI * radius;
        System.out.println("둘레 : " + peri);
    }

    public void showArea( ) {
        double area = PI * radius  * radius;
        System.out.println("넓이 : " + area);
    }
}
```

PI 값은 인스턴스 별로 독립적으로 유지할 필요가 없다. 그리고 그 값의 변경도 불필요하다는 특성이 있다

Class 변수(Global Variable)

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

Class 변수(Global Variable)

- JVM은 실행과정에서 필요한 클래스의 정보를 Memory에 로딩
- 바로 이 Loading 시점에서 static 변수가 초기화 됨

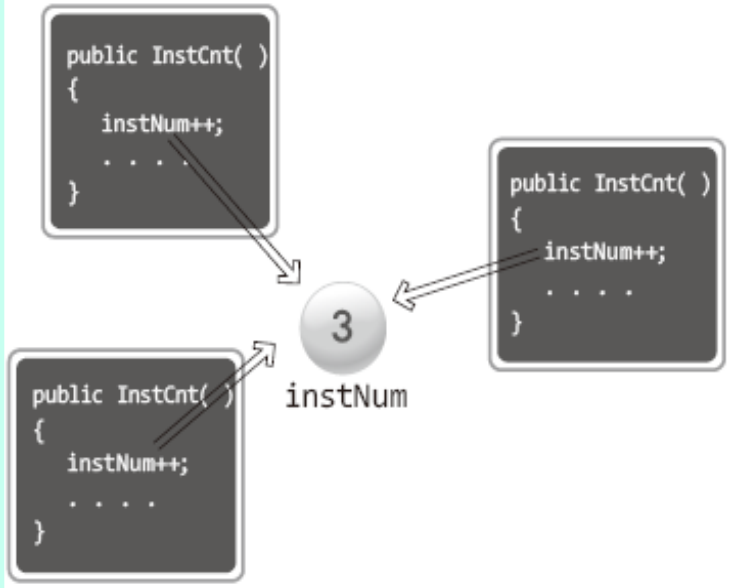
```
class InstCnt {
    static int instNum = 100;
    public InstCnt() {
        instNum++;
        System.out.println("인스턴스 생성 : " + instNum);
    }
}
public class StaticValNoInst {
    public static void main(String[ ] args) {
        InstCnt.instNum = 15;
        System.out.println(InstCnt.instNum);
    }
}
```

이 예제에서는 한번의 객체 생성이 진행되지 않았다. 즉, 객체 생성과 static 변수와는 아무런 상관이 없다

Class 변수(Global Variable)

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

```
public class ClassVar {
    public static void main(String[ ] args) {
        InstCnt cnt1 = new InstCnt( );
        InstCnt cnt2 = new InstCnt( );
        InstCnt cnt3 = new InstCnt( );
    }
}
class InstCnt {
    static int instNum = 0;
    public InstCnt() {
        instNum++;
        System.out.println("객체 생성 : " + instNum);
    }
}
```



Class 변수(Global Variable)

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

```
class AccessWay {
    static int num = 0;
    AccessWay( ) {
        incrCnt( );
    }
    public void incrCnt( ) { num++; }
}
public class ClassVarAccess {
    public static void main(String[ ] args) {
        AccessWay way = new AccessWay( );
        way.num++;
        AccessWay.num++;
        System.out.println("num = " + way.num);
    }
}
```

클래스 내부 접근 방법

객체의 이름을 이용한 접근 방법

클래스의 이름을 이용한 접근 방법

Class 변수(Global Variable)

- static 메소드는 instance가 만들어지지 않은 상태에서에서도 호출가능하지만, static 메소드 내에서 만들어지지도 않은 instance 메소드를 호출한다는 것은 불가능

```
class Test {
      int var1 = 0;
      static int var2 = 1;

      static int getVar() {
            return var1;      //var1은 인스턴스 변수이므로 참조 불가능
      }

      public static void main(String[] args){
            System.out.println(Test.getVar());          //error 발생
       }
}
```

Class 변수(Global Variable)

- 다음 Program의 출력 결과는 ?

```
public class Main {
    static int test = 100;     // 클래스 변수
    int sample;                // Instance 변수

    public static void main(String[] args) {
        Main count = new Main();
        count.test = 300;
        count.sample = 500;
        int test = 200;          // 지역 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(Global Variable)

- 다음 Program의 출력 결과는 ?

```
class Count {
    private int number;
    private static int counter = 0;

    public static int getCount( ) {
        return counter;
    }

    public Count( ) {
        counter++;
        number = counter;
    }
}
```

Class 변수(Global Variable)

```
public class CountTest {
     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( ));
     }
}
```

Class 변수(Global Variable)

- 다음 Program의 출력 결과는 ?

```
class Box {
      int width ;
      int height;
      int depth;
      long idNum;
      static long boxID = 0;   //클래스(static) 변수 boxID 선언 및 초기화

      public Box() {          //생성자
        idNum = boxID++;
                    // 객체가 생성될 때마다 클래스 변수 값을 증가
    }
 }
```

Class 변수(Global Variable)

```
public class Box_static_test {
    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(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
    - 객체를 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 객체 생성으로 만들어진 Instance는 참조되지 않으면 Garbage Collection에 의해 소멸되므로 Memory 낭비를 방지함
  - 많이 참조되는 static 변수는 Error 발생 시 Debugging이 힘듦
    - 큰 Project에서 값이 자주 바뀌는 객체를 static으로 선언하게 되면 예상치 못한 Error가 생길 수 있음

Instance 변수

- 메소드 밖에서 선언된 변수
  - Member 변수는 같은 class 안에 있는 모든 Member 메소드에서 사용할 수 있음
  - Member 변수는 0에 준하는 값으로 자동으로 초기화 됨

```
public class Test {
    int z;            // member 변수

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

Instance 변수

- Instance 변수는 class 내부에 위치함
- Instance 변수는 객체(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 객체에서 호출해 주어야 함

Instance 변수

```
public class Test {
        static int sval = 123;              // 클래스 변수
        int ival = 321;                     //  인스턴스 변수

        public static void main(String[] args) {
            System.out.println(sval);
         // System.out.println(ival);      // 오류 (사용 불가능)

            Test ex1 = new Test();     // 인스턴스 변수 생성
            Test ex2 = new Test();

            ex1.ival = 456;                         // 공유되지 않음
            System.out.println(ex1.ival);
            System.out.println(ex2.ival);
        }
}
```

Instance 변수

- Instance 변수는 각 Instance마다 독립적인 저장 공간을 갖음

```
public class Test {
    int x = 3;
    int y;

    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);
    }
}
```

Local Variables

- 메소드 내에서 선언된 변수
- 메소드의 매개 변수도 지역 변수의 일종
- 자신이 선언된 메소드 내에서만 존재하고 메소드 내부로부터의 접근만 가능
- 메소드가 호출될 때 지역 변수 Memeory(Stack)가 할당되고, 메소드가 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

- 지역 변수는 선언된 메소드 안에서만 사용될 수 있음
- 지역 변수를 초기화하지 않고 사용하면 오류
- Field는 클래스 전체를 통하여 사용될 수 있으므로 클래스 안의 모든 메소드에서 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

- 지역 변수 유효 범위(Scope)
  - 변수 선언 지점부터 변수가 선언된 Block의 끝까지

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

    for (int i= 1; i<= n; ++) {
        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
메소드 전체

변수 i는 루프 실행이 끝나자마자
메모리에서 제거

Local Variables

- 지역 변수 유효 범위(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 변수의 유효 범위는 if 블록의 끝까지이고, else 블록 내에서는 두 변수에 접근할 수 없고 알지도 못함

Local Variables

- 지역 변수 유효 범위(Scope)
  - 지역 변수의 scope내에는 같은 이름의 지역 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 없다

    . . .
}
```

Local Variables

- 지역 변수 유효 범위(Scope)
  - Scope가 겹치지 않는다면 같은 이름의 지역 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

- 지역 변수 유효 범위(Scope)
  - 지역 변수와 Field가 같은 이름을 가질 때는 지역 변수가 Field를 가림 (shadow)

```
public class Coin {
   private String name;
   private double value;   // Field

   public double getExchangeValue(double exchangeRate) {
      double value;   // Local variable
      . . .
      return value;    // value가 아닌 value가 반환됨
   }
}
```

Local Variable

- 가려진 Field에 접근하는 방법 – this 사용

```
class Coin {
   private String name;
   private double value;

   public Coin(double value, String name) {
          this.value = value;
          this.name = name;
   }
}
```

Initialization Block

- Class 초기화 블록 : static { }
  - Class 변수의 복잡한 초기화에 사용되며, 클래스가 Loading될 때 실행
- Instance 초기화 블록 : { }
  - 생성자에서 공통적으로 수행되는 작업에 사용되며 Instance가 생성될 때 마다(생성자보다 먼저) 실행

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

Initialization Block

- 초기화 블록은 명시적 초기화만으로 처리할 수 없는 복잡한 초기화에 사용
- 초기화할 때는 항상 명시적 초기화가 1차적으로 고려되어야 하고, 명시적 초기화로 해결되지 않을 때 초기화 블록을 사용해야 함
- 초기화 블록은 내부적으로는 메소드로 처리됨. 자동적으로 호출되는 초기화 메소드 임
- 선언되는 영역도 메소드처럼 클래스 영역에 선언
- 클래스 초기화 블록은 Class 변수의 복잡한 초기화에 사용
- 클래스가 Loading되면 Class 변수가 만들어지고, 그 다음에 클래스 초기화 블록이 자동적으로 호출되어 Class 변수에 대한 초기화를 수행
- 클래스 초기화 블록은 클래스가 Loading될 때 단 한번만 실행 됨

Initialization Block

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

Initialization Block

```
class StaticBlockTest {
        static int[] arr = new int[10];      // 명시적 초기화
        static {                                          //  클래스 초기화 블록
                for (int i = 0; i <arr.length; i++)
                        arr[i] = (int) (Math.random( ) * 10) +1;
        }
        {                                                  // 인스턴스 초기화 블록
                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]);
             StaticBlock test = new StaticBlock( );
             for (int i =0; i < test.arr.length; i++)
                  System.out.println((i +1) + ": " +test.arr[i]);
        }
}
```

변수의 초기화 시기와 순서

- Class 변수 초기화 시점 : 클래스가 처음 로딩될 때 단 한번
- Instance 변수 초기화 시점 : 인스턴스가 생성될 때 마다

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

```
InitTest it = new InitTest();
```

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

변수의 초기화 시기와 순서

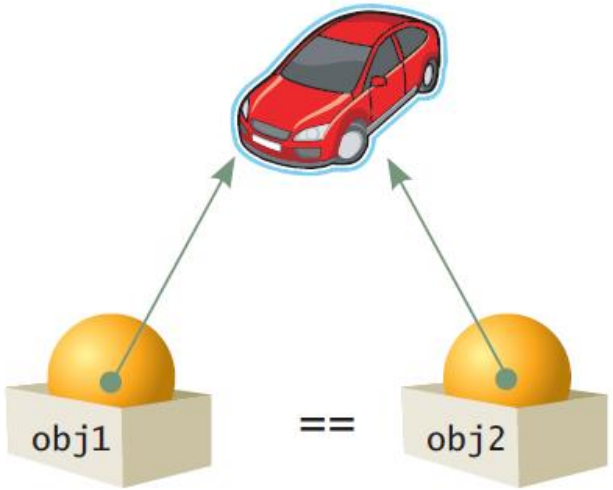
```
class Product {
        static int count = 0;       // 명시적 초기화
        int serialNo;
       {                            //  Instance 초기화 블록
              ++count;
              serialNo = 1000 + count;
        }
        product( ) { }              // 생성자
}

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");
     }
}
```

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

변수의 유효 Scope

- 변수의 유효 범위는 선언된 변수가 동일한 의미를 갖는 영역을 의미
- 변수의 유효 범위는 변수에 따라 다음과 같이 구분함
  - Member 변수
    - 클래스 정의의 시작 부분에 선언되며, 특정 메소드에 속하지 않는 변수를 멤버 변수라 함
    - 모든 멤버 변수들은 클래스 내의 모든 메소드에서 사용 가능
    - 단 메소드 내에서 동일한 이름의 변수가 선언 되었을 때는 접근 불가능

변수의 유효 Scope

- 메소드의 인자(arguments)와 메소드 지역 변수
  - 메소드 지역 변수(local variable)는 변수가 정의된 메소드 내에서만 사용 가능
  - 메소드 인자는 메소드 호출 시 전달되는 변수, 메소드 내에서만 유효
  - main() 메소드는 String args[]라는 매개 변수를 가짐
    - main() 메소드 내에서만 사용할 수 있음.

변수의 유효 Scope

- Block의 Local Variable
  - 메소드는 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, 지역 변수의 이름은 비 지역 변수와 동일해서는 안됨

변수의 유효 Scope

```
class  MyObj {
        static int s = 40;   // 클래스 변수 ( static variable )
        int tot = 20;        // 인스턴스 변수

        void f() {
             short s = 300;   // 지역 변수
             int tot = 100;    // 지역 변수
             System.out.println( "s=" + s + " tot="+tot);
        }
        void g() {
            System.out.println("s=" + s + " tot="+tot);
            {
                  int s= 500;    // s는 블록 지역 변수
                  System.out.println( "s=" + s + " tot=" + tot );
            }
            System.out.println("s=" + s + " tot=" + tot );
       }
```

변수의 유효 Scope

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

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

메소드 f()내에서는 멤버 변수와 동일한 이름의 변수가 선언되어 있으며, g()내에서 변수 s는 블록이 시작할 때 별도의 공간이 할당되어 블록이 종료되면 회수됨

final 클래스와 메소드

- final 클래스
  - 더 이상 클래스 상속 불가능

```
final class FinalClass {
.....
}
class DerivedClass extends FinalClass { // 컴파일 오류
.....
}
```

오류

- final 메소드
  - 더 이상 Overriding 불가능

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

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

오류

final Field

- 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