

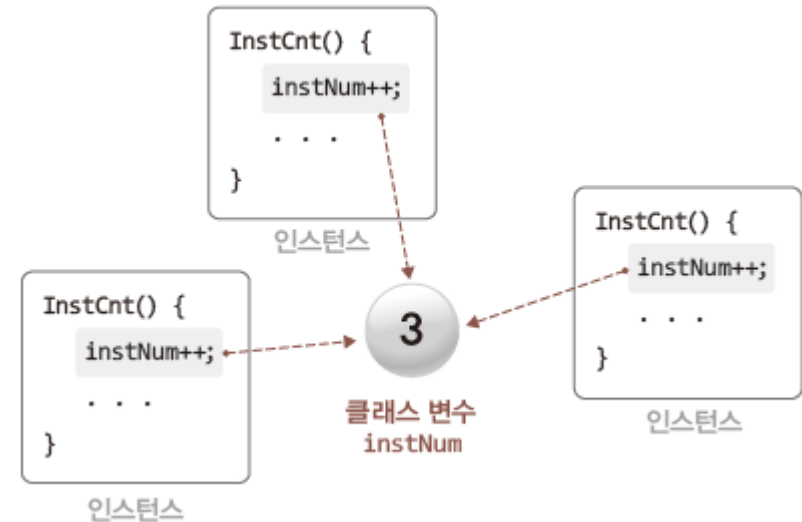
열혈 Java 프로그래밍

Chapter 10. 클래스 변수와 클래스 메소드

10-1. static 선언을 붙여서 선언하는 클래스 변수

선언된 클래스의 모든 인스턴스가 공유하는 클래스 변수

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



```
C:\JavaStudy>java ClassVar  
인스턴스 생성: 1  
인스턴스 생성: 2  
인스턴스 생성: 3  
  
C:\JavaStudy>
```

클래스 변수의 접근 방법

클래스 내부 접근

- static 변수가 선언된 클래스 내에서는 이름만으로 직접 접근 가능

클래스 외부 접근

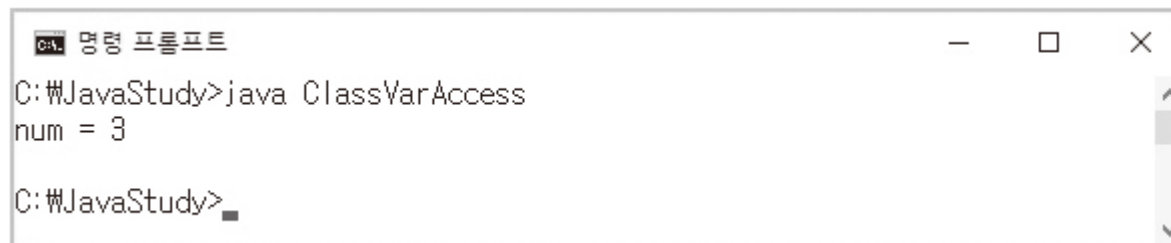
- private으로 선언되지 않으면 클래스 외부에서도 접근 가능
- 접근 수준 지시자가 허용하는 범위에서 접근 가능
- 클래스 또는 인스턴스의 이름을 통해 접근

클래스 변수 접근의 예

```
class AccessWay {
    static int num = 0;

    AccessWay() { incrCnt(); }
    void incrCnt() {
        num++; // 클래스 내부에서 이름을 통한 접근
    }
}

class ClassVarAccess {
    public static void main(String[] args) {
        AccessWay way = new AccessWay();
        way.num++; // 외부에서 인스턴스의 이름을 통한 접근
        AccessWay.num++; // 외부에서 클래스의 이름을 통한 접근
        System.out.println("num = " + AccessWay.num);
    }
}
```



```
명령 프롬프트
C:\JavaStudy>java ClassVarAccess
num = 3
C:\JavaStudy>
```

클래스 변수의 초기화 시점과 초기화 방법

```
class InstCnt {  
    static int instNum = 100;  
        클래스 변수의 적절한 초기화 위치  
    InstCnt() {  
        instNum++;  
        System.out.println("인스턴스 생성: " + instNum);  
    }  
}
```

클래스 변수는 생성자 기반 초기화 하면 안된다!

이 경우 인스턴스 생성시마다 값이 리셋!

```
class OnlyClassNoInstance {  
    public static void main(String[] args) {  
        InstCnt.instNum -= 15;    // 인스턴스 생성 없이 instNum에 접근  
        System.out.println(InstCnt.instNum);  
    }  
}
```

클래스 변수의 활용의 예

```
class Circle {  
    static final double PI = 3.1415;  
    private double radius;  
  
    Circle(double rad) {  
        radius = rad;  
    }  
    void showPerimeter() {  
        double peri = (radius * 2) * PI;  
        System.out.println("둘레: " + peri);  
    }  
    void showArea() {  
        double area = (radius * radius) * PI;  
        System.out.println("넓이: " + area);  
    }  
}
```

인스턴스 별로 가지고 있을 필요가 없는 변수

- 값의 참조가 목적인 변수
- 값의 공유가 목적인 변수

그리고 그 값이 외부에서도 참조하는 값이라면 public으로 선언한다.

10-2. static 선언을 붙여서 정의 하는 클래스 메소드

클래스 메소드의 정의와 호출

```
class NumberPrinter {  
    private int myNum = 0;  
    static void showInt(int n) { System.out.println(n); }  
    static void showDouble(double n) {System.out.println(n); }  
  
    void setMyNumber(int n) { myNum = n; }  
    void showMyNumber() { showInt(myNum); }  
}  
                        내부 접근
```

클래스 메소드의 성격 및 접근 방법이
클래스 변수와 동일하다.

```
class ClassMethod {  
    public static void main(String[] args) {  
외부 접근 NumberPrinter.showInt(20);  
        NumberPrinter np = new NumberPrinter();  
외부 접근 np.showDouble(3.15);  
        np.setMyNumber(75);  
        np.showMyNumber();  
    }  
}
```

클래스 메소드로 정의하는 것이 옳은 경우

```
class SimpleCalculator {
    static final double PI = 3.1415;

    static double add(double n1, double n2) {
        return n1 + n2;
    }
    static double min(double n1, double n2) {
        return n1 - n2;
    }
    static double calcCircleArea(double r) {
        return PI * r * r;
    }
    static double calcCirclePeri(double r) {
        return PI * (r * 2);
    }
}
```

단순 기능 제공이 목적인 메소드들, 인스턴스 변수와 관련 지을 이유가 없는 메소드들은 static으로 선언하는 것이 옳다.

클래스 메소드에서 인스턴스 변수에 접근이 가능할까?

```
class AAA {  
    int num = 0;  
    static void addNum(int n) {  
        num += n;  
    }  
}
```

논리적으로 이 문장이 유효할 수 있는지를 생각해보자.

10-3. System.out.println 그리고

```
public static void main()
```

System.out.println()에서 out과 println의 정체는?

```
java.lang.System.out.println(...);
```

| System은 java.lang 패키지에 묶여 있는 클래스의 이름

| 그러나 컴파일러가 다음 문장을 삽입해 주므로 java.lang을 생략할 수 있다.

```
| import java.lang.*;
```

```
System.out.println(...);
```

| out은 클래스 System의 이름을 통해 접근하므로,

| 이는 System 클래스의 클래스 변수 이름임을 유추할 수 있다.

```
System.out.println(...);
```

| println은 out이 참조하는 인스턴스의 메소드이다.

main 메소드가 public이고 static인 이유는?

```
public static void main(String[] args) {...}
```

| static인 이유! 인스턴스 생성과 관계없이 제일 먼저 호출되는 메소드이다.

```
public static void main(String[] args) {...}
```

| public인 이유! main 메소드의 호출 명령은 외부로부터 시작되는 명령이다.
| 단순히 일종의 약속으로 이해해도 괜찮다.

main 메소드를 어디에 위치시킬 것인가?

```
class Car {  
    void myCar() {  
        System.out.println("This is my car");  
    }  
}
```

```
public static void main(String[] args) {  
    Car c = new Car();  
    c.myCar();  
    Boat t = new Boat();  
    t.myBoat();  
}
```

```
}
```


Boat 클래스로 이동시킨다면 달라지는 것은?

```
class Boat {  
    void myBoat() {  
        System.out.println("This is my boat");  
    }  
}
```

10-4. 또 다른 용도의 static 선언

static 초기화 블록

```
class DateOfExecution {  
    static String date;    // 프로그램의 실행 날짜를 저장하기 위한 변수  
  
    public static void main(String[] args) {  
        System.out.println(date);  
    }  
}
```



```
static {  
    LocalDate nDate = LocalDate.now();  
    date = nDate.toString();  
}
```

인스턴스 생성과 관계 없이 static 변수가 메모리 공간에 할당될 때 실행이 된다.

static import 선언

```
System.out.println(Math.PI);  
                java.lang.Math.PI
```

```
System.out.println(PI);  
                import static java.lang.Math.PI;
```

The End

Chapter 10의 강의를 마칩니다.