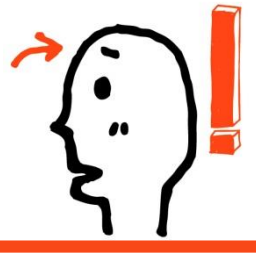
Android Java

9장 클래스 변수와 클래스 메소드

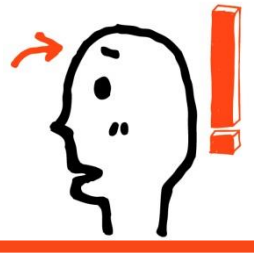
클래스 변수와 클래스 메소드



01 static 변수(클래스 변수)

02 static 메소드(클래스 메소드)

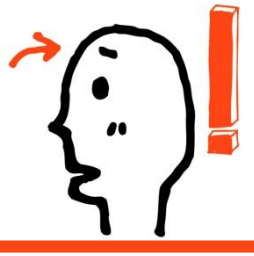
static 변수



static 변수

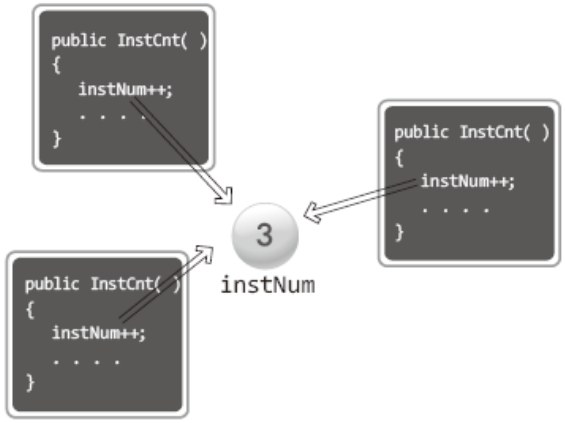
- 메모리 로딩 과정에서 static으로 선언된 자원들은
  인스턴스화 과정 없이 실행 가능한 형태로 메모리에 바로 생성
- static으로 선언된 모든 자원들은 클래스 로딩시에 생성되고,
  JVM이 종료되면 그 값은 메모리에서도 종료
- static 키워드가 없는 메소드를 사용하기 위해서는
  반드시 클래스를 인스턴스화 해야 메모리에 실행 가능한 상태가 됨
- new 키워드를 이용하여 객체를 생성하고 객체에 대한 접근을 해야 실행할 수 있음

static 변수



- 인스턴스의 생성과 상관없이 초기화되는 변수
- 하나만 선언되는 변수
- public으로 선언되면 누구나 어디서든 접근 가능!

```
class InstCnt
{
   static int instNum=0;
    public 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();
    }
}
```

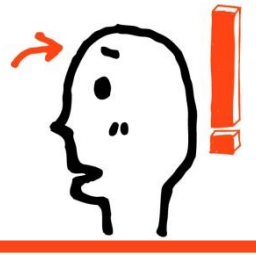


실행결과

```
인스턴스 생성 : 1
인스턴스 생성 : 2
인스턴스 생성 : 3
```

실행결과를 통해서 변수가 공유되고 있음을 확인할 수 있다.

static 변수의 접근방법



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

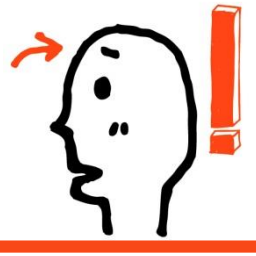
```
class AccessWay
{
    static int num=0;

    AccessWay()
    {
        incrCnt();
    }
    public void incrCnt() { num++; }
}

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

클래스 내부 접근방법

인스턴스의 이름을 이용한 접근방법

클래스의 이름을 이용한 접근방법

static 변수의 초기화 시점

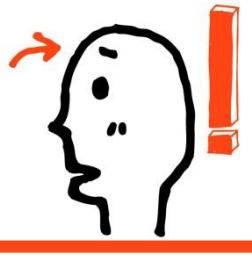


- JVM은 실행과정에서 필요한 클래스의 정보를 메모리에 로딩한다.
- 바로 이 Loading 시점에서 static 변수가 초기화 된다.

```
class InstCnt
{
    static int instNum=100;

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

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

static 변수의 활용 예



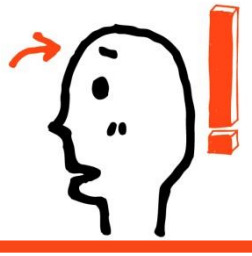
- 동일한 클래스의 인스턴스 사이에서의 데이터 공유가 필요할 때 static 변수는 유용하게 활용된다.
- 클래스 내부, 또는 외부에서 참조의 목적으로 선언된 변수는 static final로 선언한다.

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

    public Circle(double rad){ radius=rad; }
    public void showPerimeter()    // 둘레 출력
    {
        double peri=(radius*2)*PI;
        System.out.println("둘레 : "+peri);
    }
    public void showArea()     // 넓이 출력
    {
        double area=(radius*radius)*PI;
        System.out.println("넓이 : "+area);
    }
}
```

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

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

static 메소드의 정의와 호출



• static 메소드의 기본적인 특성과 접근방법은 static 변수와 동일하다.

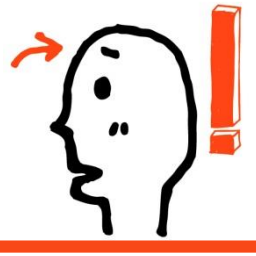
```
class NumberPrinter
{
    public static void showInt(int n){ System.out.println(n); }
    public static void showDouble(double n) { System.out.println(n); }
}

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

클래스의 이름을 통한 호출

인스턴스의 이름을 통한 호출

이 예제에서 보이듯이 인스턴스를 생성하지 않아도 static 메소드는 호출 가능하다.

static 메소드의 활용의 예



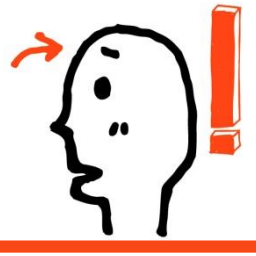
```
class SimpleMath
{
    public static final double PI=3.1415;
    public static double add(double n1, double n2) { return n1+n2; }
    public static double min(double n1, double n2) { return n1-n2; }
    public static double mul(double n1, double n2) { return n1*n2; }
}
```

```
class AreaMath
{
    public static double calCircleArea(double rad)
    {
        double result=SimpleMath.mul(rad, rad);
        result=SimpleMath.mul(result, SimpleMath.PI);
        return result;
    }
    public static double calRectangleArea(double width, double height)
    {
        return SimpleMath.mul(width, height);
    }
}
```

위 클래스들의 특징은 인스턴스의 생성이 의미가 없다는 것이다!

달리 말하면 위의 두 클래스에는 인스턴스 변수가 포함되어 있지 않다!

정적 멤버와 static



정적 초기화 블록

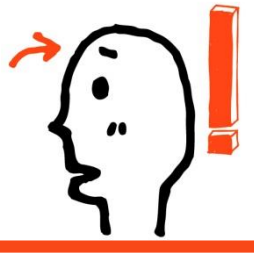
- 정적 필드의 복잡한 초기화 작업과 정적 메소드 호출 가능
- 클래스 내부에 여러 개가 선언되면 선언된 순서대로 실행

```
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.

정적 멤버와 static



정적 메소드와 정적 블록 작성시 주의할 점

- 객체가 없어도 실행 가능
- 블록 내부에 인스턴스 필드나 인스턴스 메소드 사용 불가
- 객체 자신의 참조인 this 사용 불가

EX) main()

```
//인스턴스 필드와 메소드
int field1;
void method1() { ... }

//정적 필드와 메소드
static int field2;
static void method2() { ... }
```

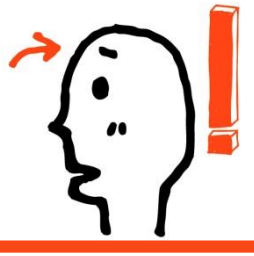
```
//정적 블록
static {
    field1 = 10;        (x)
    method1();          (x)
    field2 = 10;        (o)
    method2();          (o)
}

//정적 메소드
static void Method3 {
    this.field1 = 10;       (x)
    this.method1();         (x)
    field2 = 10;            (o)
    method2();              (o)
}
```

컴파일 에러

컴파일 에러

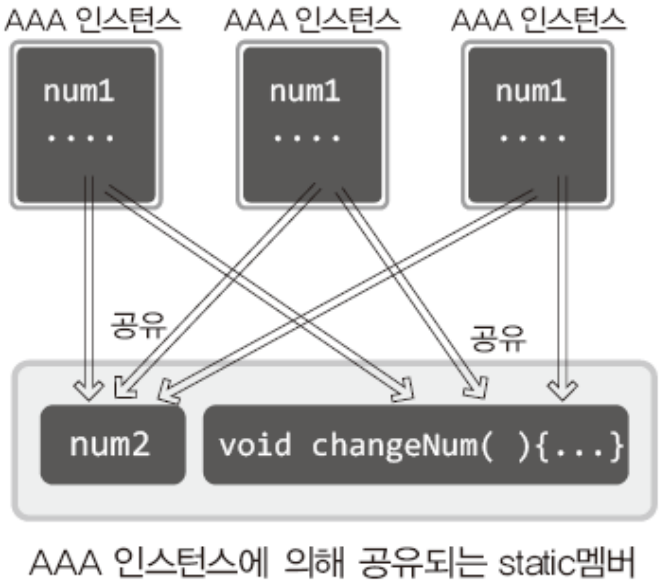
```
static void Method3() {
    ClassName obj = new ClassName();
    obj.field1 = 10;
    obj.method1();
}
```

static 메소드의 인스턴스 접근? 말이 안 된다!



• static 메소드는 인스턴스에 속하지 않기 때문에 인스턴스 멤버에 접근이 불가능하다! (객체가 생성되지 않은 상황에서도 static 메소드는 실행될 수 있기 때문에, 인스턴스 메소드와 필드 사용 불가)

```
class AAA
{
    int num1;
    static int num2;
    static void changeNum()
    {
        num1++;      // 문제 됨!
        num2++;      // 문제 안됨!
    }
    . . . .
}
```

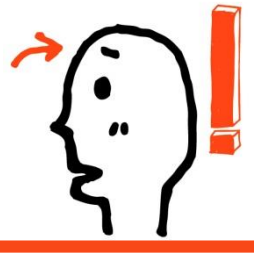


AAA 인스턴스에 의해 공유되는 static멤버

static 변수 num2의 증가를 어떤 인스턴스를 대상으로 진행해야 하겠는가?

때문에 이는 논리적으로 인식될 수 없는 코드이다.

Static의 활용



- static 멤버를 가진 클래스 사례
  - java.lang.Math 클래스
    - JDK와 함께 배포되는 java.lang.Math 클래스
    - 모든 필드와 메소드가 public static으로 선언
    - 다른 모든 클래스에서 사용할 수 있음

```
public class Math {
   public static int abs(int a);
   public static double cos(double a);
   public static int max(int a, int b);
   public static double random();
   ...
}
```

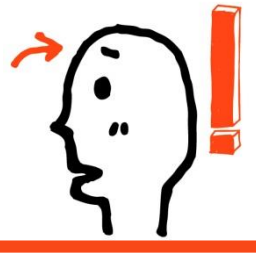
// 잘못된 사용법

```
Math m = new Math(); // Math() 생성자는 private
int n = m.abs(-5);
```

// 바른 사용법

```
int n = Math.abs(-5);
```

Static의 활용



- static import의 활용

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

java.lang.Math.PI

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

```
import static java.lang.Math.PI;
```

THANK YOU

실무에서 알아야 할 기술은 따로 있다! 자바를 다루는 기술