Chapter 04. 연산자(Operator)

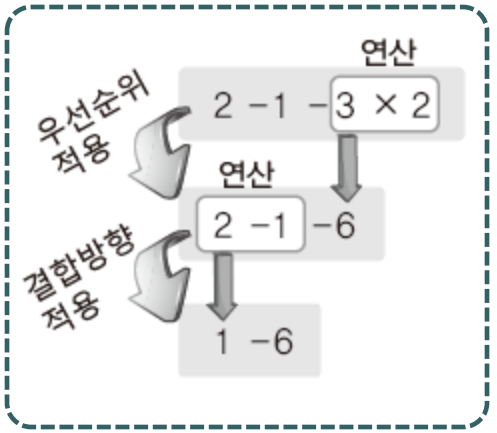
나는 정말 JAVA를 공부한 적이 없다고

04-1.
자바에서 제공하는 이항 연산자들

자바의 연산자와 연산의 과정

연산기호	결합방향	우선순위
[ ], .	➡	1(높음)
expr++, expr--	⬅	2
++expr, -- expr, +expr, -expr, ~, !, (type)	⬅	3
*, /, %	➡	4
+, -	➡	5
〈〈, 〉〉, 〉〉〉	➡	6
〈, 〉, 〈=, 〉=, instanceof	➡	7
==, !=	➡	8
&	➡	9
^	➡	10
|	➡	11
&&	➡	12
||	➡	13
? expr : expr	⬅	14
=, +=, -=, *=, /=, %=, &=, ^=, |=, 〈〈=, 〉〉=, 〉〉〉=	⬅	15(낮음)

연산의 과정



■ 대입 연산자(=)와 산술 연산자(+, -, *, /, %)

나 정말 JAVA를 공부할 때 없더라은

연산자	연산자의 기능	결합방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) val = 20;	⬅
+	두 피연산자의 값을 더한다. 예) val = 4 + 3;	➡
–	왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다. 예) val = 4 – 3;	➡
*	두 피연산자의 값을 곱한다. 예) val = 4 * 3;	➡
/	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다. 예) val = 7 / 3;	➡
%	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다. 예) val = 7 % 3	➡

■ 대입연산과 산술연산의 예


```
class ArithOp
{
    public static void main(String[] args)
    {
        int n1=7;
        int n2=3;

        int result=n1+n2;
        System.out.println("덧셈 결과 : " + result);

        result=n1-n2;
        System.out.println("뺄셈 결과 : " + result);
        System.out.println("곱셈 결과 : " + n1*n2);
        System.out.println("나눗셈 결과 : " + n1/n2);
        System.out.println("나머지 결과 : " + n1%n2);
    }
}
```

실행 결과

```
덧셈 결과 : 10
뺄셈 결과 : 4
곱셈 결과 : 21
나눗셈 결과 : 2
나머지 결과 : 1
```

■ 나눗셈 연산자와 나머지 연산자에 대한 보충

✓ / 연산자와 % 연산자의 연산방식

- 피연산자가 정수면 정수형 연산진행
- 피연산자가 실수면 실수형 연산진행, 단 % 연산자 제외!

```
class DivOpnd
{
    public static void main(String[] args)
    {
        System.out.println("정수형 나눗셈 : " + 7/3);
        System.out.println("실수형 나눗셈 : " + 7.0f/3.0f);
        System.out.println("형 변환 나눗셈 : " + (float)7/3);
    }
}
```

실행결과

```
정수형 나눗셈 : 2
실수형 나눗셈 : 2.3333333
형 변환 나눗셈 : 2.3333333
```

```
class AmpOpnd
{
    public static void main(String[] args)
    {
        System.out.println("정수형 나머지 : " + 7%3);
        System.out.println("실수형 나머지 : " + 7.2 % 2.0);
    }
}
```

실행결과

```
정수형 나머지 : 1
실수형 나머지 : 1.2000000000000002
```

복합대입 연산자

해석의 원칙은 동일

&=, ^=, |=, <<=, >>=, >>>=

```
class Comp
{
    public static void main(String[] args)
    {
        double e=3.1;
        e+=2.1;
        e*=2;
        int n=5;
        n*=2.2;
        System.out.println(e);
        System.out.println(n);
    }
}
```

실행결과

```
10.4
11
```

관계 연산자

나 정말 JAVA를 공부한 적이 없다구요

연산자	연산자의 기능	결합방향
<	예) n1 < n2 n1이 n2보다 작은가?	➡
>	예) n1 > n2 n1이 n2보다 큰가?	➡
<=	예) n1 <= n2 n1이 n2보다 같거나 작은가?	➡
>=	예) n1 >= n2 n1이 n2보다 같거나 큰가?	➡
==	예) n1 == n2 n1과 n2가 같은가?	➡
!=	예) n1 != n2 n1과 n2가 다른가?	➡

연산의 결과로 true or false 반환

관계연산의 예

```
class CmpOp
{
    public static void main(String[] args)
    {
        int A=10, B=20;

        if(true)
            System.out.println("참 입니다!");
        else
            System.out.println("거짓 입니다!");

        if(A>B)
            System.out.println("A가 더 크다!");
        else
            System.out.println("A가 더 크지 않다!");

        if(A!=B)
            System.out.println("A와 B는 다르다!");
        else
            System.out.println("A와 B는 같다!");
    }
}
```

실행결과

```
참 입니다!
A가 더 크지 않다!
A와 B는 다르다!
```

논리 연산자

연산의 결과로 true or false 반환

연산자	연산자의 기능	결합방향
&&	예) A && B A와 B 모두 true이면 연산결과는 true (논리 AND)	➡
||	예) A || B A와 B 둘 중 하나라도 true이면 연산결과는 true (논리 OR)	➡
!	예) !A 연산결과는 A가 true이면 false, A가 false이면 true (논리 NOT)	⬅

```
class LogicOp
{
    public static void main(String[] args)
    {
        int num1=10, num2=20;
        boolean result1=(num1==10 && num2==20);
        boolean result2=(num1<=12 || num2>=30);
        System.out.println("num1==10 그리고 num2==20 : " + result1);
        System.out.println("num1<=12 또는 num2>=30 : " + result2);
        if(!(num1==num2))
            System.out.println("num1과 num2는 같지 않다.");
        else
            System.out.println("num1과 num2는 같다.");
    }
}
```

실행 결과

```
num1==10 그리고 num2==20 : true
num1<=12 또는 num2>=30 : true
num1과 num2는 같지 않다.
```

논리 연산자와 SCE

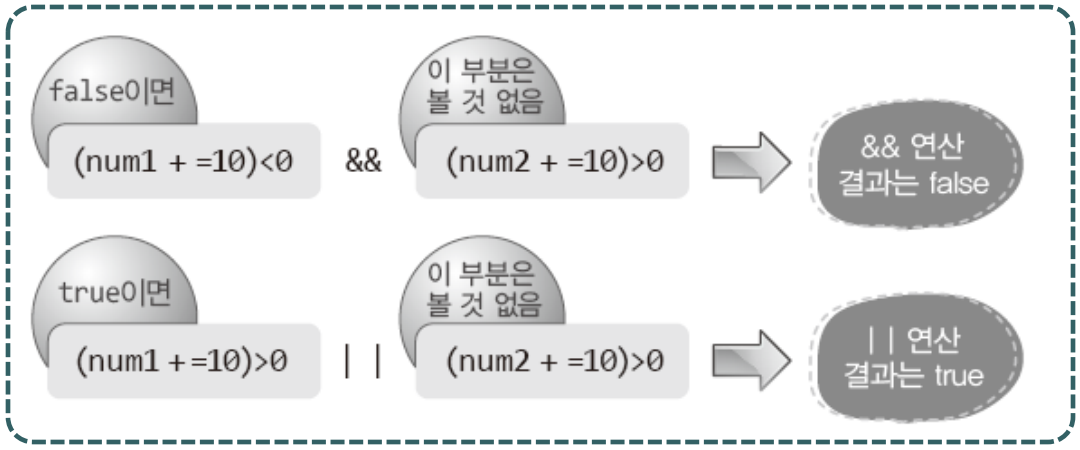
```
class SCE
{
    public static void main(String[] args)
    {
        int num1=0, num2=0;
        boolean result;

        result = (num1+=10)<0 && (num2+=10)>0;
        System.out.println("result="+result);
        System.out.println("num1="+num1+", num2="+num2);

        result = (num1+=10)>0 || (num2+=10)>0;
        System.out.println("result="+result);
        System.out.println("num1="+num1+", num2="+num2);
    }
}
```

실행 결과

```
result=false
num1=10, num2=0
result=true
num1=20, num2=0
```



Short-Circuit Evaluation

나는 정말 JAVA를 공부한 적이 없다고

04-2.
자바에서 제공하는 단항 연산자들

■ 부호 연산자로서의 +와 -

나는 정말 JAVA를 공부한 적이 없다구요

✓ 연산자의 기능

- 단항 연산자로서 –는 부호를 바꾸는 역할을 한다.
- 단항 연산자로서 +는 특별히 하는 일이 없다.

```
class UnaryAddMin
{
    public static void main(String[] args)
    {
        int n1 = 5;
        System.out.println(+n1);
        System.out.println(-n1);

        short n2 = 7;
        int n3 = +n2;
        int n4 = -n2;
        System.out.println(n3);
        System.out.println(n4);
    }
}
```

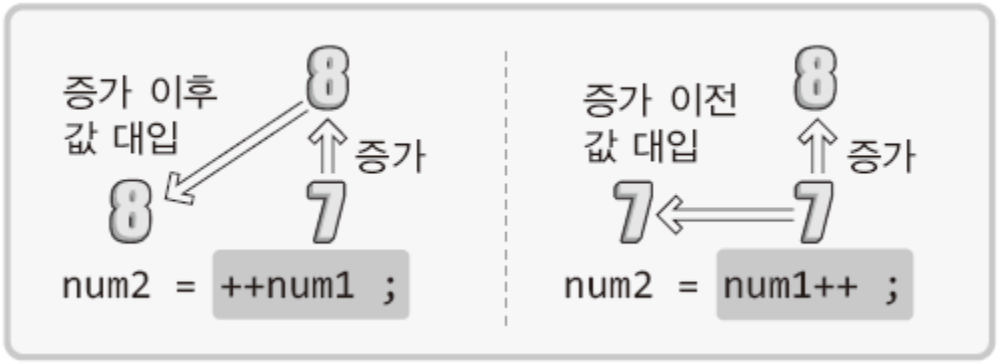
실행결과

```
5
-5
7
-7
```

■ 증가, 감소 연산자

연산자	연산자의 기능	결합방향
++ (prefix)	피연산자에 저장된 값을 1 증가 예) val = ++n;	←
-- (prefix)	피연산자에 저장된 값을 1 감소 예) val = --n;	←

연산자	연산자의 기능	결합방향
++ (postfix)	피연산자에 저장된 값을 1 증가 예) val = n++;	←
-- (postfix)	피연산자에 저장된 값을 1 감소 예) val = n--;	←



■ 증가 감소 연산의 예

```
class PrefixOp
{
    public static void main(String[] args)
    {
        int num1 = 7;
        int num2, num3;

        num2 = ++num1;
        num3 = --num1;

        System.out.println(num1);
        System.out.println(num2);
        System.out.println(num3);
    }
}
```

실행 결과

```
7
8
7
```

```
class PostfixOp
{
    public static void main(String[] args)
    {
        int num1 = 7;
        int num2, num3;

        num2 = num1++;
        num3 = num1--;

        System.out.println(num1);
        System.out.println(num2);
        System.out.println(num3);
    }
}
```

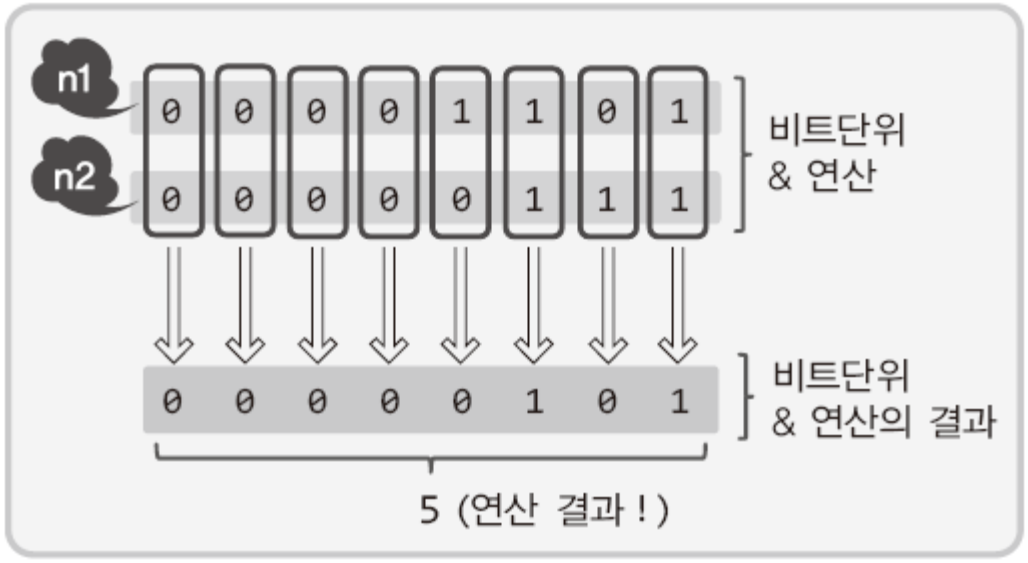
실행 결과

```
7
7
8
```

04-3.
비트와 관련이 있는 연산자들

비트 연산자

연산자	연산자의 기능	결합방향
&	비트단위로 AND 연산을 한다. 예) n1 & n2;	➡
|	비트단위로 OR 연산을 한다. 예) n1 | n2;	➡
^	비트단위로 XOR 연산을 한다. 예) n1 ^ n2;	➡
~	피연산자의 모든 비트를 반전시켜서 얻은 결과를 반환 예) ~n;	⬅



■ 비트연산 진리 표

비트 A	비트 B	비트 A & 비트 B
1	1	1
1	0	0
0	1	0
0	0	0

&(and)

비트 A	비트 B	비트 A | 비트 B
1	1	1
1	0	1
0	1	1
0	0	0

|(OR)

비트 A	비트 B	비트 A ^ 비트 B
1	1	0
1	0	1
0	1	1
0	0	0

^(XOR)

비트	~비트
1	0
0	1

~(NOT)

비트연산의 예

```
class BitOperator
{
    public static void main(String[] args)
    {
        int num1=5;     /* 00000000 00000000 00000000 00000101 */
        int num2=3;     /* 00000000 00000000 00000000 00000011 */
        int num3=-1;    /* 11111111 11111111 11111111 11111111 */

        System.out.println(num1 & num2);
        System.out.println(num1 | num2);
        System.out.println(num1 ^ num2);
        System.out.println(~num3);
    }
}
```

실행결과

```
1
7
6
0
```

■ 비트 쉬프트(Shift) 연산자

연산자	연산자의 기능	결합방향
〈〈	• 피연산자의 비트 열을 왼쪽으로 이동 • 이동에 따른 빈 공간은 0으로 채움 • 예) n 〈〈 2; → n의 비트 열을 두 칸 왼쪽으로 이동 시킨 결과 반환	➡
〉〉	• 피연산자의 비트 열을 오른쪽으로 이동 • 이동에 따른 빈 공간은 음수의 경우 1, 양수의 경우 0으로 채움 • 예) n 〉〉 2; → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	➡
〉〉〉	• 피연산자의 비트 열을 오른쪽으로 이동 • 이동에 따른 빈 공간은 0으로 채움 • 예) n 〉〉〉 2; → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	➡

✓ 비트연산의 특징

- 왼쪽으로의 비트 열 이동은 2의 배수의 곱
- 오른쪽으로의 비트 열 이동은 2의 배수의 나눗셈

- 정수 2 → 00000010 → 정수 2
- 2 〈〈 1 → 00000100 → 정수 4
- 2 〈〈 2 → 00001000 → 정수 8
- 2 〈〈 3 → 00010000 → 정수 16

■ 비트 쉬프트(Shift) 연산의 예

```
class BitShiftOp
{
    public static void main(String[] args)
    {
        System.out.println(2 << 1);    // 4 출력
        System.out.println(2 << 2);    // 8 출력
        System.out.println(2 << 3);    // 16 출력

        System.out.println(8 >> 1);    // 4 출력
        System.out.println(8 >> 2);    // 2 출력
        System.out.println(8 >> 3);    // 1 출력
        System.out.println(-8 >> 1);   // -4 출력
        System.out.println(-8 >> 2);   // -2 출력
        System.out.println(-8 >> 3);   // -1 출력

        System.out.println(-8 >>> 1);  // 2147483644 출력
    }
}
```