

# JAVA

## - 연산자

a + b

```
:      1      -> boolean  
:      2      ->  
:  ?  :
```

e.g.

->

->

-> n % 2 == 0

자바에서 제공하는 연산자

## 연산자(Operator)란?

### ▶ 연산자(Operator)

- 어떠한 기능을 수행하는 기호(+,-,\*,/ 등)

### ▶ 피연산자(Operand)

- 연산자의 작업 대상(변수,상수,리터럴,수식)

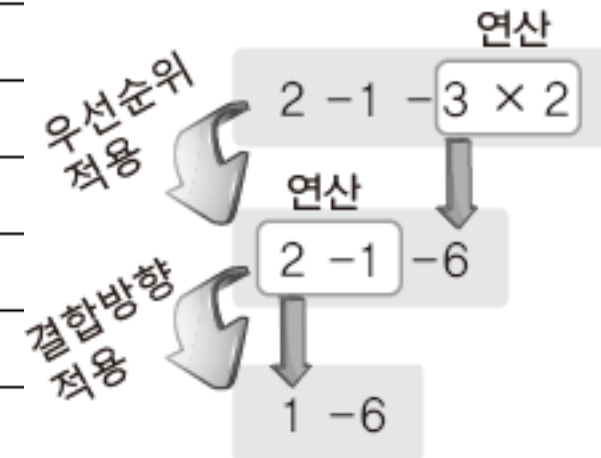
$$a + b$$

# 자바의 연산자와 연산의 과정

종 류	연산방향	연 산 자	우선순위
단항 연산자	←	++ -- + - ~ ! (타입)	높음
산술 연산자	→	* / % :	
	→	+ -	
	→	<< >> >>> :	
비교 연산자	→	< > <= >= instanceof	중간
	→	== !=	
논리 연산자	→	( )	
	→	&	
	→	^	
	→		
	→	&&	
삼항 연산자	→	?: if	낮음
대입 연산자	←	= *= /= %= += -= <<= >>= >>>= &= ^=  =	

- num  
-> num \* (-1)

boolean check = true;  
!check => false



## 연산자의 우선순위

```
class test {  
    :  
}
```

- 괄호의 우선순위가 제일 높다.
- 산술 > 비교 > 논리 > 대입
- 단항 > 이항 > 삼항
- 연산자의 연산 진행방향은 왼쪽에서 오른쪽( $\rightarrow$ )이다.  
단, 단항, 대입 연산자만 오른쪽에서 왼쪽( $\leftarrow$ )이다.

$((a + b) * c)$   
1. 2.

$\underline{7 + 1} - 5$   
1 2

$a = b = 3$

# 자바의 연산자와 연산의 과정

- 상식적으로 생각하라. 우리는 이미 다 알고 있다.

ex1)  $-x + 3$   
          1      2

단항 > 이항

ex2)  $x + 3 * y$   
          2      1

곱셈, 나눗셈 > 덧셈, 뺄셈

ex3)  $x + 3 > y - 2$   
          1      3      2

산술 > 비교

ex4)  $x > 3 \&\& x < 5$   
          1      3      2

비교 > 논리

ex5)  $\text{int result} = x + y * 3;$   
                  3      2      1

항상 대입은 맨 끝에

# 자바의 연산자와 연산의 과정

그러나 몇 가지 주의해야 할 것이 있다.

1.  $<<$ ,  $>>$ ,  $>>>$ 는 덧셈연산자보다 우선순위가 낮다.

ex5)  $x << 2 + 1$                        $x << (2 + 1)$  과 같다.

2.  $\|$ ,  $|$ (OR)는  $\&\&$ ,  $\&$ (AND)보다 우선순위가 낮다.

ex6)  $x < -1$  <sup>2</sup>  $\|$   $x > 3$  <sup>1</sup>  $\&\& x < 5$

$x < -1 \parallel (x > 3 \&\& x < 5)$  와 같다.

->  
-> , t/f

# 자바에서 제공하는 이항 연산자



# 대입 연산자(=)와 산술 연산자(+, -, \*, /, %)

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 얼게 되는 나머지를 반환한다. 예) val = 7 % 3	→

# 대입 연산자(=)와 산술 연산자(+, -, \*, /, %)

## 대입연산자

- 오른쪽 피 연산자의 값을 왼쪽 피 연산자에 저장한다.  
단, 왼쪽 피 연산자는 상수가 아니어야 한다.

```
int i = 0;
```

```
i = i + 3;
```

```
final int MAX = 3;
```

```
MAX = 10; // 에러
```

# 대입 연산과 산술 연산의 예

```
class ArithOp {  
    public static void main(String[] args) {  
        int n1 = 7;  
        int n2 = 3;  
  
        int result = n12 + n21; 가 final int n1 = 7;  
        System.out.println("덧셈 결과: " + result); -> 10  
  
        result = n12 - n21;  
        System.out.println("뺄셈 결과: " + result); -> 4  
  
        System.out.println("곱셈 결과: " + n1*n2);  
        System.out.println("나눗셈 결과: " + n1/n2);  
        System.out.println("나머지 결과: " + n1%n2);  
    }  
}
```

덧셈 결과 : 10

뺄셈 결과 : 4

곱셈 결과 : 21

-> 나눗셈 결과 : 2

나머지 결과 : 1

# 나눗셈 연산자와 나머지 연산자에 대한 보충

## / 연산자와 % 연산자의 연산 방식

→ 피 연산자가 정수면 정수형 연산 진행

→ 피 연산자가 실수면 실수형 연산 진행, 단% 연산자 제외!

- 나누기한 나머지를 반환한다.
- 홀수, 짝수 등 배수검사에 주로 사용.

`int share = 10 / 8;`

`10 % 8 → 2`

`int remain = 10 % 8;`

`10 % -8 → 2`

`-10 % 8 → -2`

`-10 % -8 → -2`

# 나눗셈 연산자와 나머지 연산자에 대한 보충

```
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);
    }
}
```

: int  
: double

$= 7f/3$   
 $= 7/3f$

(float) int / int  
float / int  
float / float => float

\*  
int < long < float < double

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

# 복합대입연산자

+

<code>a = a + b</code>	⇐ 동일 연산 ⇒	<code>a += b</code>
<code>a = a - b</code>	⇐ 동일 연산 ⇒	<code>a -= b</code>
<code>a = a * b</code>	⇐ 동일 연산 ⇒	<code>a *= b</code>
<code>a = a / b</code>	⇐ 동일 연산 ⇒	<code>a /= b</code>
<code>a = a % b</code>	⇐ 동일 연산 ⇒	<code>a %= b</code>

int sum = 0;

sum = sum + i;  
sum += i;

!

# 관계(비교)연산자

- 피 연산자를 같은 타입으로 변환한 후에 비교한다.

결과 값은 true 또는 false이다. boolean

- 기본형(boolean제외)과 참조형에 사용할 수 있으나  
참조형에는 **==**와 **!=**만 <sup>-></sup>사용할 수 있다.

수 식	연 산 결 과
$x > y$	x가 y보다 클 때 true, 그 외에는 false
$x < y$	x가 y보다 작을 때 true, 그 외에는 false
$x \geq y$	x가 y보다 크거나 같을 때 true, 그 외에는 false
$x \leq y$	x가 y보다 작거나 같을 때 true, 그 외에는 false
$x == y$	x와 y가 같을 때 true, 그 외에는 false
$x != y$	x와 y가 다를 때 true, 그 외에는 false

【표3-11】 비교연산자의 연산결과

# 관계(비교)연산자

## 연산의 결과로 true or false 반환

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?	→



# 관계(비교)연산자

char    ->    int  
'A' < 'B' → 65 < 66 → **true**

'0' == 0 → 48 == 0 → **false**

'A' != 65 → 65 != 65 → **false**

10.0d == 10.0f → 10.0d == 10.0d → **true**

0.1d == 0.1f → 0.1d == 0.1d → **true? false?**

double d = (double)0.1f;

System.out.println(d); // 0.100000000149011612

(float)0.1d == 0.1f → 0.1f == 0.1f → **true**

## 관계(비교) 연산의 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!");  
        }  
    }  
}
```

참 입니다!

A가 더 크지 않다!

A와 B는 다르다!

# 논리연산자

-피연산자가 반드시 boolean이어야 하며 연산결과도 boolean이다.

**&&가 || 보다 우선순위가 높다.** 같이 사용되는 경우 괄호를 사용하자

- ▶ OR연산자(||) : 피연산자 중 어느 한 쪽이 true이면 true이다.
- ▶ AND연산자(&&) : 피연산자 양 쪽 모두 true이면 true이다.

# 논리연산자

## 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)	←

x	y	x    y	x && y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

# 논리연산자

```
int i = 7;
```

```
i > 3 && i < 5
```

```
i > 3 || i < 0
```

x	y	x    y	x && y
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

```
char x = 'j';
```

```
x >= 'a' && x <= 'z'
```

```
(x >= 'a' && x <= 'z') || (x >= 'A' && x <= 'Z')
```

# 논리연산자

```
class LogicOp
{
    public static void main(String[] args)
    {
        int num1=10, num2=20;

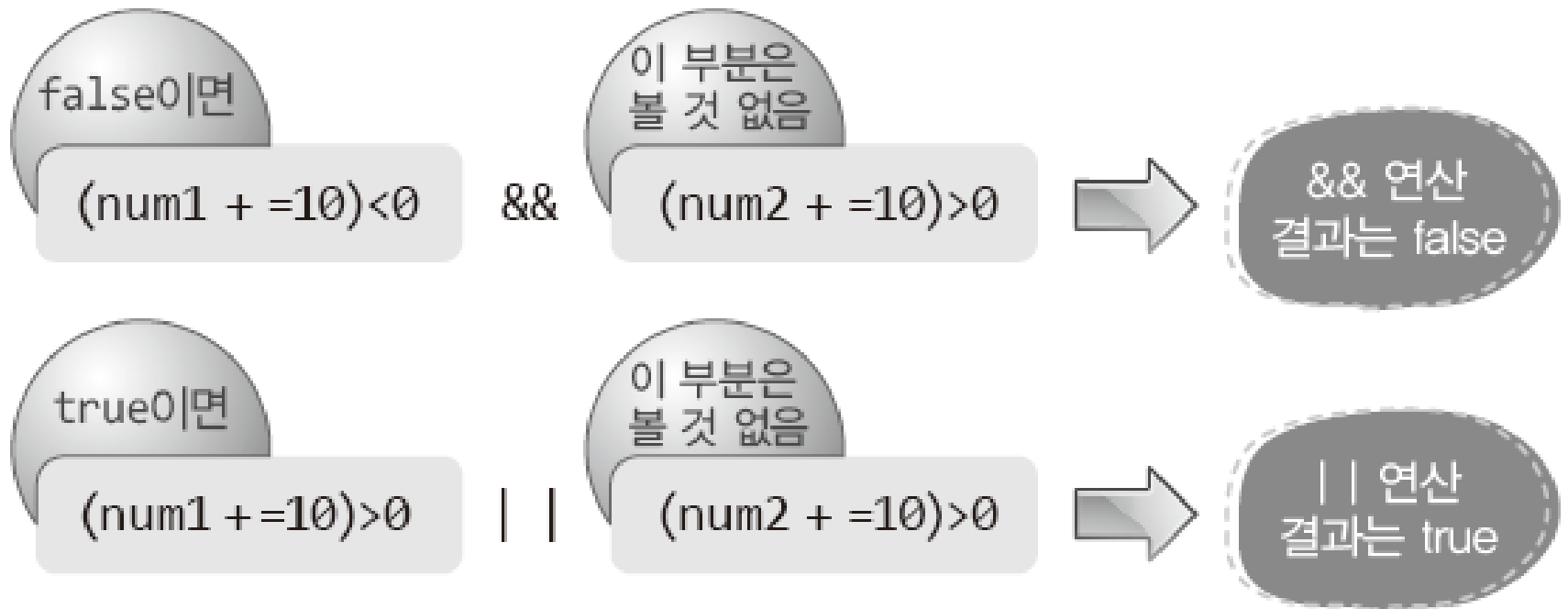
        boolean result1=(num1==10 && num2==20);
        boolean result2=(num1<=12 || num2>=30);

        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.");
    }
}
```

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

# 논리 연산자와 SCE



**Short-Circuit Evaluation**

# 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);
    }
}
```

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



# 이항연산자의 특징

**이항연산자는 연산을 수행하기 전에 피연산자의 타입을 일치시킨다.**

- int보다 크기가 작은 타입은 int로 변환한다.  
( byte, char, short → int )
- 피연산자 중 표현범위가 큰 타입으로 형변환 한다.

byte + short → int + int → int

char + int → int + int → int

float + int → float + float → float

long + float → float + float → float

float + double → double + double → double

## 이항연산자의 특징

```
byte a = 10;
```

```
byte b = 20;
```

```
byte c = a + b;
```

byte + byte → int + int → int

```
byte c = (byte)a + b; // 에러
```

```
byte c = (byte)(a + b); // OK
```

# 이항연산자의 특징

```
int a = 1000000; // 1,000,000
```

```
int b = 2000000; // 2,000,000
```

```
long c = a * b; // c는 2,000,000,000,000 ?  
              // c는 -1454759936 !!!
```

$\text{int} * \text{int} \rightarrow \text{int}$

```
long c = (long)a * b; // c는 2,000,000,000,000 !
```

$\text{long} * \text{int} \rightarrow \text{long} * \text{long} \rightarrow \text{long}$

## 이항연산자의 특징

`long a = 1000000 * 1000000; // a는 -727,379,968`

`long b = 1000000 * 1000000L; // b는 1,000,000,000,000`

`int c = 1000000 * 1000000 / 1000000; // c는 -727`

`int d = 1000000 / 1000000 * 1000000; // d는 1,000,000`

# 이항연산자의 특징

```
char c1 = 'a';
```

```
char c2 = c1 + 1; // 에러
```

```
char c2 = (char)(c1 + 1); // OK
```

```
char c2 = ++c1; // OK
```

```
int i = 'B' - 'A';
```

```
int i = '2' - '0';
```

문자	코드
...	...
0	48
1	49
2	50
...	...
A	65
B	66
C	67
...	...
a	97
b	98
c	99
...	...

## 이항연산자의 특징

```
float pi = 3.141592f;
```

```
float shortPi = (int)(pi * 1000) / 1000f;
```

```
(int)(3.141592f * 1000) / 1000f;
```

```
(int)(3141.592f) / 1000f;
```

```
3141 / 1000f;
```

```
3141.0f / 1000f
```

```
3.141f
```

## 이항연산자의 특징

\* Math.round() : 소수점 첫째자리에서 반올림한 값을 반환

```
float pi = 3.141592f;
```

```
float shortPi = Math.round(pi * 1000) / 1000f;
```

```
Math.round(3.141592f * 1000) / 1000f;
```

```
Math.round(3141.592f) / 1000f;
```

```
3142 / 1000f;
```

```
3142.0f / 1000f
```

```
3.142f
```

자바에서 제공하는 단항 연산자들



# 부호연산자로서의 +와 - 그리고 !

- 연산자의기능

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

- ▶ 부호연산자(+,-) : '+'는 피 연산자에 1을 곱하고  
'-'는 피 연산자에 -1을 곱한다.
- ▶ 논리부정연산자(!) : true는 false로, false는 true로  
피연산자가 boolean일 때만 사용가능

```
int i = -10;  
i = +i;  
i = -i;
```

```
boolean power = false;  
power = !power;  
power = !power;
```

## 부호연산자로서의 +와

```
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);  
  
    }  
}
```

5  
-5  
7  
-7

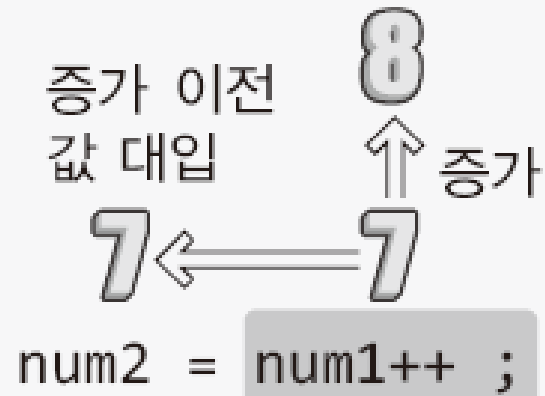
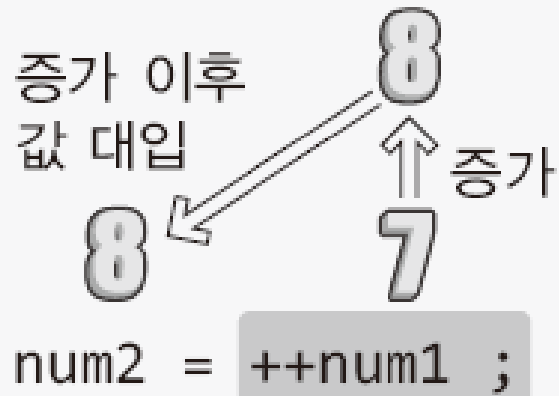
# 증가, 감소연산자

- ▶ 증가연산자(++): 피연산자의 값을 1 증가시킨다.
- ▶ 감소연산자(--): 피연산자의 값을 1 감소시킨다.

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, 감소연산자



전위형	<code>j = ++i;</code>	<code>++i;</code> <code>j = i;</code>	값이 참조되기 전에 증가시킨다.
후위형	<code>j = i++;</code>	<code>j = i;</code> <code>i++;</code>	값이 참조된 후에 증가시킨다.

## 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);  
    }  
}
```

7

8

7

## 증가 감소 연산의 예

```
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);  
    }  
}
```

7

7

8

# 문제

## 문제1.

int 형 변수 num1, num2, num3가 각각 10,20,30 으로 초기화 되어 있다.  
다음문장을 실행하면 각각 변수에는 어떠한 값이 저장되겠는가?

num1=num2=num3 ;

확인 하는 코드를 작성하고, 그 결과에 대해 설명.

## 문제2.

수학식  $\{(25 \times 5) + (36 - 4) - 72\} / 5$  의 계산결과를 출력하는 프로그램 작성.

## 문제3.

3+6, 3+6+9, 3+6+9+12 의 연산을 하는 프로그램 작성.

단. 덧셈 연산의 횟수 최소화

## 문제4.

$a = \{(25 + 5) + (36 / 4) - 72\} * 5$ ,  $b = \{(25 \times 5) + (36 - 4) - 71\} / 4$ ,  $c = (128 / 4) * 2$  일 때  
 $a > b > c$  가 참이면 true 아니면 false를 출력하는 프로그램 작성

# 정리합시다.

- 연산자의 종류
- 연산자의 우선 순위
- 연산의 결과
- 연산과정



## 예제

```
class OperatorEx1 {  
    public static void main(String args[]) {  
        int i=5;  
        i++;  
        // i=i+1;과 같은 의미이다. ++i;로 바꿔 써도 결과는 같다.  
        System.out.println(i);  
        i=5;  
        // 결과를 비교하기 위해 i값을 다시 5로 변경.  
        ++i;  
        System.out.println(i);  
    }  
}
```

## 예제

```
class OperatorEx2 {  
    public static void main(String args[]) {  
        int i=5;  
        int j=0;  
        j = i++;  
        System.out.println("j=i++; 실행 후, i=" + i + ", j="+ j);  
        i=5;  
        // 결과를 비교하기 위해, i와 j의 값을 다시 5와 0으로 변경  
        j=0;  
        j = ++i;  
        System.out.println("j=++i; 실행 후, i=" + i + ", j="+ j);  
    }  
}
```

## 예제

```
class OperatorEx3 {  
    public static void main(String args[]) {  
        int i=5, j=5;  
        System.out.println(i++);  
        System.out.println(++j);  
        System.out.println("i = " + i + ", j = " +j);  
    }  
}
```

## 예제

```
class OperatorEx4{  
  
    public static void main(String[] args) {  
        int i = -10;  
        i = +i;  
        System.out.println(i);  
        i=-10;  
        i = -i;  
        System.out.println(i);  
    }  
}
```

## 예제

```
class OperatorEx5 {  
  
    public static void main(String[] args)  
    {  
        byte b = 10;  
        System.out.println("b = " + b );  
        System.out.println("~b = " + ~b);  
        System.out.println("~b+1 = " + (~b+1));  
    }  
}
```

## 예제

```
class OperatorEx6
{
    public static void main(String[] args) {
        byte b = 10;
        // byte result = ~b;
        byte result =(byte)~b;

        System.out.println("b = " + b );
        System.out.println("~b = " + result );
    }
}
```

## 예제

```
class OperatorEx7 {  
  
    public static void main(String[] args) {  
        boolean power = false;  
        System.out.println(power);  
        power = !power;  
        // power의 값이 false에서 true로 바뀐다.  
        System.out.println(power);  
        power = !power;  
        // power의 값이 true에서 false로 바뀐다.  
        System.out.println(power);  
    }  
}
```

## 예제

```
class OperatorEx8 {  
  
    public static void main(String[] args)  
    {  
        byte a = 10;  
        byte b = 20;  
        byte c = a + b; // byte c = (byte)(a+b);  
        System.out.println(c);  
    }  
}
```



## 예제

```
class OperatorEx9
{
    public static void main(String[] args)
    {
        byte a = 10;
        byte b = 30;
        byte c = (byte)(a * b);
        System.out.println(c);
    }
}
```

## 예제

```
class OperatorEx10
{
    public static void main(String[] args)
    {
        int a = 1000000;           // 1,000,000    1백만
        int b = 2000000;           // 2,000,000    2백만
        long c = a * b;             // 2,000,000,000,000
        System.out.println(c);
    }
}
```

## 예제

```
class OperatorEx11
{
    public static void main(String[] args)
    {
        long a = 1000000 * 1000000;
        long b = 1000000 * 1000000L; // long형 리터럴
        System.out.println(a);
        System.out.println(b);
    }
}
```

## 예제

```
class OperatorEx12
{
    public static void main(String[] args)
    {
        int a = 1000000 * 100000 / 1000000;
        int b = 1000000 / 100000 * 1000000;
        System.out.println(a);
        System.out.println(b);
    }
}
```

## 예제

```
class OperatorEx13 {  
    public static void main(String[] args) {  
        char c1 = 'a'; // c1에는 문자 'a'의 코드 값인 97이 저장된다.  
        char c2 = c1; // c1에 저장되어 있는 값이 c2에 저장된다.  
        char c3 = ' '; // c3를 공백으로 초기화 한다.  
        int i = c1 + 1;    // 'a'+1 → 97+1 → 98  
        c3 = (char)(c1 + 1);  
        c2++;  
        c2++;  
        System.out.println("i=" + i);  
        System.out.println("c2=" + c2);  
        System.out.println("c3=" + c3);  
    }  
}
```

# 예제

```
class OperatorEx14 {  
    public static void main(String[] args) {  
        char c1 = 'a';  
        char c2 = c1+1  
        char c2 = 'a'+1;    // 라인 6 : 컴파일 에러 없음.  
        System.out.println(c2);  
    }  
}  
  
class OperatorEx14 {  
    public static void main(String[] args) {  
        char c1 = 'a';  
        char c2 = c1+1;    // 라인 5 : 컴파일 에러 발생!!!  
        char c2 = 'a'+1;    // 라인 6 : 컴파일 에러 없음.  
        System.out.println(c2);  
    }  
}
```

# 예제

```
class OperatorEx15 {  
    public static void main(String[] args) {  
        char c = 'a';  
        for(int i=0; i<26; i++) {    // 블록{} 안의 문장을 26번을 반복한다.  
            System.out.print(c++);    //'a'부터 26개의 문자를 출력한다.  
        }  
        System.out.println(); // 줄 바꿈을 한다.  
        c = 'A';  
        for(int i=0; i<26; i++) {    // 블록{} 안의 문장을 26번을 반복한다.  
            System.out.print(c++); //'A'부터 26개의 문자를 출력한다.  
        }  
        System.out.println();  
        c='0';  
        for(int i=0; i<10; i++) {  
            // 블록{} 안의 문장을 10번을 반복한다.  
            System.out.print(c++);    //'0'부터 10개의 문자를 출력한다.  
        }  
        System.out.println();  
    }  
}
```

## 예제

```
class OperatorEx16 {  
  
    public static void main(String[] args)  
    {  
        char lowerCase = 'a';  
        char upperCase = (char)(lowerCase - 32);  
        System.out.println(upperCase);  
    }  
}
```



## 예제

```
class OperatorEx17
{
    public static void main(String[] args)
    {
        float pi = 3.141592f;
        float shortPi = (int)(pi * 1000) / 1000f;

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

## 예제

```
class OperatorEx18
{
    public static void main(String[] args)
    {
        float pi = 3.141592f;
        float shortPi = Math.round(pi * 1000) / 1000f;

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

## 예제

```
class OperatorEx19
{
    public static void main(String[] args)
    {
        int share = 10 / 8;
        int remain = 10 % 8;
        System.out.println("10을 8로 나누면, ");
        System.out.println("몫은 "+share+"이고, 나머지는  
"+remain+"입니다.");
    }
}
```

## 예제

```
class OperatorEx20
{
    public static void main(String[] args)
    {
        // i가 1부터 10이 될 때까지, {}안의 문장을 반복 수행한다.
        for(int i=1; i <=10; i++) {
            if(i%3==0) {
                // i가 3으로 나누어 떨어지면, 3의 배수이므로 출력한다.
                System.out.println(i);
            }
        }
    }
}
```

## 예제

```
class OperatorEx21 {  
    public static void main(String[] args)  
    {  
        System.out.println(-10%8);  
        System.out.println(10%-8);  
        System.out.println(-10%-8);  
    }  
}
```

# 예제

```
class OperatorEx22 {  
    public static void main(String[] args) {  
        if(10 == 10.0f) {  
            System.out.println("10과 10.0f는 같다.");  
        }  
        if('0' != 0) {  
            System.out.println("'0'과 0은 같지 않다.");  
        }  
        if('A' == 65) {  
            System.out.println("₩"AW"는 65와 같다.");  
        }  
        int num = 5;  
        if( num > 0 && num < 9) {  
            System.out.println("5는 0보다 크고, 9보다는 작다.");  
        }  
    }  
}
```

# 예제

```
class OperatorEx23 {  
    public static void main(String[] args) {  
        float f = 0.1f;  
        double d = 0.1;  
        double d2 = (double)f;  
  
        System.out.println("10.0==10.0f ? "+(10.0==10.0f));  
        System.out.println("0.1==0.1f ? "+(0.1==0.1f));  
        System.out.println("f="+f);  
        System.out.println("d="+d);  
        System.out.println("d2="+d2);  
        System.out.println("d==f ? "+(d==f));  
        System.out.println("d==d2 ? "+(d==d2));  
        System.out.println("d2==f ? "+(d2==f));  
    }  
}
```

## 예제

```
class OperatorEx24 {  
    public static void main(String[] args) {  
        char x = 'j';  
  
        if((x>='a' && x <='z') || (x>='A' && x <='Z')) {  
            System.out.println("유효한 문자입니다.");  
        } else {  
            System.out.println("유효하지 않은 문자입니다.");  
        }  
    }  
}
```



## 예제

```
class OperatorEx25 {  
    public static void main(String[] args) {  
        int x = 3;  
        int y = 5;  
        System.out.println("x는 " + x + "이고, y는 " + y + "일 때, ");  
        System.out.println("x | y = " + (x|y));  
        System.out.println("x & y = " + (x&y));  
        System.out.println("x ^ y = " + (x^y));  
        System.out.println("true | false = " + (true|false));  
        System.out.println("true & false = " + (true&false));  
        System.out.println("true ^ false = " + (true^false));  
    }  
}
```

## 예제

```
class OperatorEx26 {  
  
    public static void main(String[] args) {  
        int x = 10;  
        int y = -10;  
  
        int absX = (x >= 0) ? x : -x;  
        int absY = (y >= 0) ? y : -y;  
  
        System.out.println("x= 10일 때, x의 절대값은 "+absX);  
        System.out.println("y=-10일 때, y의 절대값은 "+absY);  
    }  
}
```