

연산자

04-1.

자바에서 제공하는 이항 연산자들

이항 연산자들

연산기호	결합 방향	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)

우선순위

우선순위 적용

$$2 - 1 - 3 \times 2$$

결합 방향 적용

$$= 2 - 1 - 6$$

$$= 1 - 6$$

결합 방향은 우선순위가 같을 때 적용하는 기준.

대입 연산자와 산술 연산자

연산자	연산자의 기능	결합 방향
=	연산자 오른쪽에 있는 값을 연산자 왼쪽에 있는 변수에 대입한다. 예) <code>val = 20;</code>	←
+	두 피연산자의 값을 더한다. 예) <code>val = 4 + 3;</code>	→
-	왼쪽의 피연산자 값에서 오른쪽의 피연산자 값을 뺀다. 예) <code>val = 4 - 3;</code>	→
*	두 피연산자의 값을 곱한다. 예) <code>val = 4 * 3;</code>	→
/	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눈다. 예) <code>val = 7 / 3;</code>	→
%	왼쪽의 피연산자 값을 오른쪽의 피연산자 값으로 나눴을 때 얻게 되는 나머지를 반환한다. 예) <code>val = 7 % 3;</code>	→

대입 연산자와 산술 연산자의 예

◆ ArithOp.java

```
1. class ArithOp {  
2.     public static void main(String[] args) {  
3.         int num1 = 7;  
4.         int num2 = 3;  
5.  
6.         System.out.println("num1 + num2 = " + (num1 + num2));  
7.         System.out.println("num1 - num2 = " + (num1 - num2));  
8.         System.out.println("num1 * num2 = " + (num1 * num2));  
9.         System.out.println("num1 / num2 = " + (num1 / num2));  
10.        System.out.println("num1 % num2 = " + (num1 % num2));  
11.    }  
12. }
```

명령 프롬프트

```
C:\JavaStudy>java ArithOp  
num1 + num2 = 10  
num1 - num2 = 4  
num1 * num2 = 21  
num1 / num2 = 2  
num1 % num2 = 1  
C:\JavaStudy>
```

정수형 나눗셈과 실수형 나눗셈

```
int num1 = 7;
```

```
int num2 = 3;
```

```
System.out.println("num1 / num2 = " + (num1 / num2));
```

정수형 나눗셈 진행

```
System.out.println("num1 / num2 = " + (7.0 / 3.0));
```

실수형 나눗셈 진행

복합 대입 연산자

<code>a = a + b</code>	← 같다 →	<code>a += b</code>
<code>a = a - b</code>	← 같다 →	<code>a -= b</code>
<code>a = a * b</code>	← 같다 →	<code>a *= b</code>
<code>a = a / b</code>	← 같다 →	<code>a /= b</code>
<code>a = a % b</code>	← 같다 →	<code>a %= b</code>

ex1)

```
num = num + 5;  
→ num += 5;
```

ex2)

```
num = num * 3;  
→ num *= 3;
```


복합 대입 연산자 추가

$A \&= B$	\leftrightarrow	$A = A \& B$
$A \wedge= B$	\leftrightarrow	$A = A \wedge B$
$A \ll= B$	\leftrightarrow	$A = A \ll B$
$A \gg>= B$	\leftrightarrow	$A = A \gg B$

복합 대입 연산자 예제

◆ CompAssignOp.java

```
1. class CompAssignOp {
2.     public static void main(String[] args) {
3.         short num = 10;
4.         num = (short)(num + 77L);    // 형 변환 안하면 컴파일 오류 발생
5.         int rate = 3;
6.         rate = (int)(rate * 3.5);    // 형 변환 안하면 컴파일 오류 발생
7.         System.out.println(num);
8.         System.out.println(rate);
9.
10.        num = 10;
11.        num += 77L;    // 형 변환 필요하지 않다.
12.        rate = 3;
13.        rate *= 3.5;    // 형 변환 필요하지 않다.
14.        System.out.println(num);
15.        System.out.println(rate);
16.    }
17. }
```



```
C:\JavaStudy>java CompAssignOp
87
10
C:\JavaStudy>
```

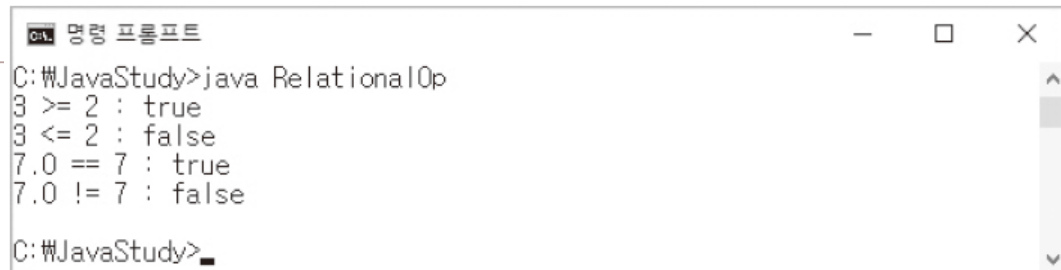
관계 연산자

연산자	연산자의 기능	결합 방향
<	예) $n1 < n2$ n1이 n2보다 작은가?	→
>	예) $n1 > n2$ n1이 n2보다 큰가?	→
<=	예) $n1 \leq n2$ n1이 n2보다 같거나 작은가?	→
>=	예) $n1 \geq n2$ n1이 n2보다 같거나 큰가?	→
==	예) $n1 == n2$ n1과 n2가 같은가?	→
!=	예) $n1 \neq n2$ n1과 n2가 다른가?	→

관계 연산자의 예제

◆ RelationalOp.java

```
1. class RelationalOp {  
2.     public static void main(String[] args) {  
3.         System.out.println("3 >= 2 : " + (3 >= 2));  
4.         System.out.println("3 <= 2 : " + (3 <= 2));  
5.         System.out.println("7.0 == 7 : " + (7.0 == 7));  
6.         System.out.println("7.0 != 7 : " + (7.0 != 7));  
7.     }  
8. }
```



```
C:\JavaStudy>java RelationalOp  
3 >= 2 : true  
3 <= 2 : false  
7.0 == 7 : true  
7.0 != 7 : false  
C:\JavaStudy>
```

논리 연산자

연산자	연산자의 기능	결합 방향
&&	예) A && B A와 B 모두 true이면 연산 결과는 true (논리 AND)	➡
	예) A B A와 B 둘 중 하나라도 true이면 연산 결과는 true (논리 OR)	➡
!	예) !A 연산 결과는 A가 true이면 false, A가 false이면 true (논리 NOT)	⬅

피연산자 1(OP1)	피연산자 2(OP2)	연산 결과(OP1 && OP2)
true	true	true
true	false	false
false	true	false
false	false	false


피연산자 1(OP1)	피연산자 2(OP2)	연산 결과(OP1 OP2)
true	true	true
true	false	true
false	true	true
false	false	false

피연산자(OP)	연산 결과(!OP)
true	false
false	true

논리 연산자 예제

◆ LogicalOp.java

```
1. class LogicalOp {
2.     public static void main(String[] args) {
3.         int num1 = 11;
4.         int num2 = 22;
5.         boolean result;
6.
7.         // 변수 num1에 저장된 값이 1과 100 사이의 수인가?
8.         result = (1 < num1) && (num1 < 100);
9.         System.out.println("1 초과 100 미만인가? " + result);
10.
11.        // 변수 num2에 저장된 값이 2 또는 3의 배수인가?
12.        result = ((num2 % 2) == 0) || ((num2 % 3) == 0);
13.        System.out.println("2 또는 3의 배수인가? " + result);
14.
15.        // 변수 num1이 0 인가?
16.        result = !(num1 != 0);
17.        System.out.println("0 인가? " + result);
18.    }
19. }
```

명령 프롬프트 창은 제목 '명령 프롬프트'를 가지고 있으며, 최소화, maximize, 닫기 버튼을 포함하고 있습니다. 창 내용은 다음과 같습니다:

```
C:\JavaStudy>java LogicalOp
1 초과 100 미만인가? true
2 또는 3의 배수인가? true
0 인가? false
C:\JavaStudy>_
```

논리 연산자 사용시 주의점 : SCE

```
result = ((num1 += 10) < 0) && ((num2 += 10) > 0);
```

```
result = ((num1 += 10) > 0) || ((num2 += 10) > 0);
```

num1과 num2의 값이 모두 증가할 수 있을까?

false 이면...

(num1 += 10) < 0

&&

이 부분 생략

(num2 += 10) > 0

true 이면...

(num1 += 10) > 0

||

이 부분 생략

(num2 += 10) > 0

SCE 동작을 확인하는 예제

◆ SCE.java

```
1. class SCE {  
2.     public static void main(String[] args) {  
3.         int num1 = 0;  
4.         int num2 = 0;  
5.         boolean result;  
6.  
7.         result = ((num1 += 10) < 0) && ((num2 += 10) > 0);  
8.         System.out.println("result = " + result);  
9.         System.out.println("num1 = " + num1);  
10.        System.out.println("num2 = " + num2 + '\n');    // '\n'은 개 행  
11.  
12.        result = ((num1 += 10) > 0) || ((num2 += 10) > 0);  
13.        System.out.println("result = " + result);  
14.        System.out.println("num1 = " + num1);  
15.        System.out.println("num2 = " + num2);  
16.    }  
17. }
```

C:\ 명령 프롬프트

```
C:\JavaStudy>java SCE  
result = false  
num1 = 10  
num2 = 0  
  
result = true  
num1 = 20  
num2 = 0  
  
C:\JavaStudy>
```


04-2.

자바에서 제공하는 단항 연산자들

부호 연산자

```
double e1 = 3.5;
```

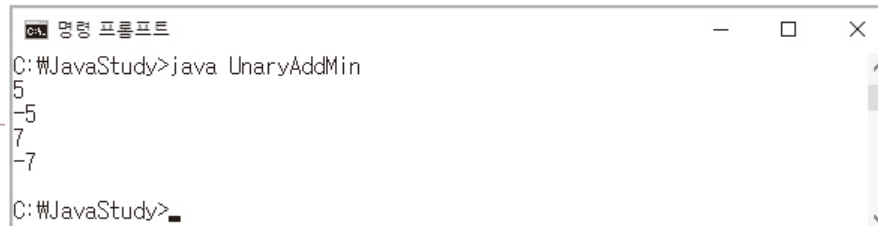
```
double e2 = -e1;    // e1에 저장되는 값은 -3.5
```

부호 연산자 -는 변수에 저장된 값의 부호를 바꾸어 반환한다.

부호 연산자 예제

◆ UnaryAddMin.java

```
1. class UnaryAddMin {
2.     public static void main(String[] args) {
3.         short num1 = 5;
4.         System.out.println(+num1);    // 결과적으로 불필요한 + 연산
5.         System.out.println(-num1);    // 부호를 바꿔서 얻은 결과를 출력
6.
7.         short num2 = 7;
8.         short num3 = (short)(+num2);  // 형 변환 하지 않으면 오류 발생
9.         short num4 = (short)(-num2);  // 형 변환 하지 않으면 오류 발생
10.        System.out.println(num3);
11.        System.out.println(num4);
12.    }
13. }
```



```
C:\JavaStudy>java UnaryAddMin
5
-5
7
-7
C:\JavaStudy>
```

증가 감소 연산자

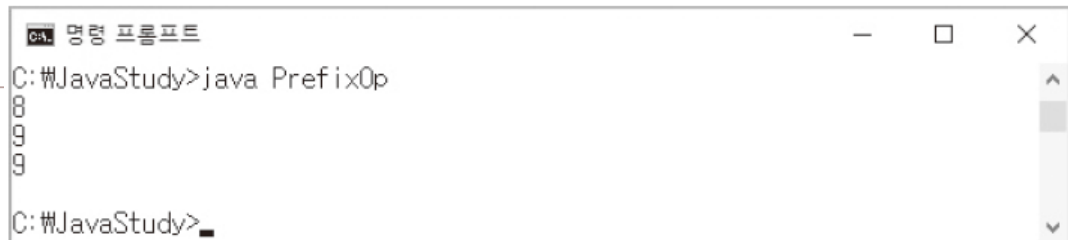
연산자	연산자의 기능	결합 방향
++ (prefix)	피연산자에 저장된 값을 1 증가 예) val = ++n;	←
-- (prefix)	피연산자에 저장된 값을 1 감소 예) val = --n;	←

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

Prefix 증가 감소 연산자 예제

◆ PrefixOp.java

```
1. class PrefixOp {  
2.     public static void main(String[] args) {  
3.         int num = 7;  
4.         System.out.println(++num);    // num의 값 하나 증가 후 출력  
5.         System.out.println(++num);    // num의 값 하나 증가 후 출력  
6.         System.out.println(num);  
7.     }  
8. }
```



```
명령 프롬프트  
C:\JavaStudy>java PrefixOp  
8  
9  
9  
C:\JavaStudy>
```

Postfix 증가 감소 연산자 예제

◆ PostfixOp.java

```
1. class PostfixOp {
2.     public static void main(String[] args) {
3.         int num = 5;
4.         System.out.print((num++) + " "); // 출력 후에 값이 증가
5.         System.out.print((num++) + " "); // 출력 후에 값이 증가
6.         System.out.print(num + "\n");
7.
8.         System.out.print((num--) + " "); // 출력 후에 값이 감소
9.         System.out.print((num--) + " "); // 출력 후에 값이 감소
10.        System.out.print(num);
11.    }
12. }
```

C:\ 명령 프롬프트

C:\JavaStudy>java PostfixOp

5 6 7

7 6 5

C:\JavaStudy>

04-3.

비트를 대상으로 하는 연산자들

비트 연산자의 이해

```
public static void main(String[] args) {  
    byte n1 = 13;  
    byte n2 = 7;  
    byte n3 = (byte)(n1 & n2);  
    System.out.println(n3);  
}
```

각각의 비트를 대상으로 연산을 진행, 그리고 각 비트를 대상으로
진행된 연산 결과를 묶어서 하나의 연산 결과 반환

연산자	연산자의 기능
&	비트 단위로 AND 연산을 한다. 예) n1 & n2;



비트 연산자

연산자	연산자의 기능	결합 방향
&	비트 단위로 AND 연산을 한다. 예) $n1 \& n2$;	
	비트 단위로 OR 연산을 한다. 예) $n1 n2$;	
^	비트 단위로 XOR 연산을 한다. 예) $n1 \wedge n2$;	
~	피연산자의 모든 비트를 반전시켜서 얻은 결과를 반환 예) $\sim n$;	

비트	~비트
1	0
0	1

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

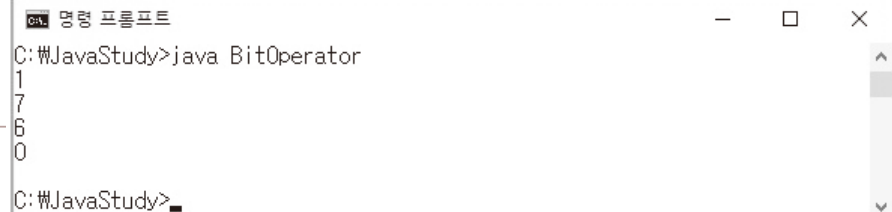
비트 A	비트 B	비트 A 비트 B
1	1	1
1	0	1
0	1	1
0	0	0

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

비트 연산자 예제

◆ BitOperator.java

```
1. class BitOperator {
2.     public static void main(String[] args) {
3.         byte n1 = 5;    // 00000101
4.         byte n2 = 3;    // 00000011
5.         byte n3 = -1;   // 11111111
6.
7.         byte r1 = (byte)(n1 & n2);    // 00000001
8.         byte r2 = (byte)(n1 | n2);    // 00000111
9.         byte r3 = (byte)(n1 ^ n2);    // 00000110
10.        byte r4 = (byte)(~n3);         // 00000000
11.
12.        System.out.println(r1);    // 00000001은 1
13.        System.out.println(r2);    // 00000111은 7
14.        System.out.println(r3);    // 00000110은 6
15.        System.out.println(r4);    // 00000000은 0
16.    }
17. }
```



```
C:\JavaStudy>java BitOperator
1
7
6
0
C:\JavaStudy>
```

비트 쉬프트 연산자

연산자	연산자의 기능	결합 방향
<<	<ul style="list-style-type: none">• 피연산자의 비트 열을 왼쪽으로 이동• 이동에 따른 빈 공간은 0으로 채움• 예) $n \ll 2$; → n의 비트 열을 두 칸 왼쪽으로 이동 시킨 결과 반환	➡
>>	<ul style="list-style-type: none">• 피연산자의 비트 열을 오른쪽으로 이동• 이동에 따른 빈 공간은 음수의 경우 1, 양수의 경우 0으로 채움• 예) $n \gg 2$; → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	➡
>>>	<ul style="list-style-type: none">• 피연산자의 비트 열을 오른쪽으로 이동• 이동에 따른 빈 공간은 0으로 채움• 예) $n \ggg 2$; → n의 비트 열을 두 칸 오른쪽으로 이동 시킨 결과 반환	➡


비트 쉬프트 연산자 예제

◆ BitShiftOp.java

```
1. class BitShiftOp {
2.     public static void main(String[] args) {
3.         byte num;
4.
5.         num = 2;    // 00000010
6.         System.out.print((byte)(num << 1) + " ");    // 00000100
7.         System.out.print((byte)(num << 2) + " ");    // 00001000
8.         System.out.print((byte)(num << 3) + " " + '\n'); // 00010000
9.
10.        num = 8;    // 00001000
11.        System.out.print((byte)(num >> 1) + " ");    // 00000100
12.        System.out.print((byte)(num >> 2) + " ");    // 00000010
13.        System.out.print((byte)(num >> 3) + " " + '\n'); // 00000001
14.
15.        num = -8;    // 11111000
16.        System.out.print((byte)(num >> 1) + " ");    // 11111100
17.        System.out.print((byte)(num >> 2) + " ");    // 11111110
18.        System.out.print((byte)(num >> 3) + " " + '\n'); // 11111111
19.    }
20. }
```

왼쪽으로의 쉬프트는 값의 2배 증가,

오른쪽으로의 쉬프트는 값을 2로 나눈 결과로 이어진다.



```
C:\JavaStudy>java BitShiftOp
4 8 16
4 2 1
-4 -2 -1
C:\JavaStudy>
```