

## ■ 연산이란?

- 데이터를 처리하여 결과를 산출하는 것

## ■ 연산자(Operations)

- 연산에 사용되는 표시나 기호(+, -, \*, /, %, =, ...)

## ■ 피연산자(Operand)

- 연산 대상이 되는 데이터(리터럴, 변수)

## ■ 연산식(Expressions)

- 연산자와 피연산자를 이용하여 연산의 과정을 기술한 것

## ■ 연산자 종류

연산자 종류	연산자	피연산자 수	산출값 타입	기능 설명
산술	+, -, *, /, %	이항	숫자	사칙연산 및 나머지 계산
부호	+, -	단항	숫자	음수와 양수의 부호
문자열	+	이항	문자열	두 문자열을 연결
대입	=, +=, -=, *=, /=, %= &=, ^=,  =, <<=, >>=, >>>=	이항	다양	우변의 값을 좌변의 변수에 대입
증감	++, --	단항	숫자	1 만큼 증가/감소
비교	==, !=, >, <, >=, <=, instanceof	이항	boolean	값의 비교
논리	!, &,  , &&,	단항 이항	boolean	논리적 NOT, AND, OR 연산
조건	(조건식) ? A : B	삼항	다양	조건식에 따라 A 또는 B 중 하나를 선택
비트	~, &,  , ^	단항 이항	숫자 bloolean	비트 NOT, AND, OR, XOR 연산
쉬프트	>>, <<, >>>	이항	숫자	비트를 좌측/우측으로 밀어서 이동

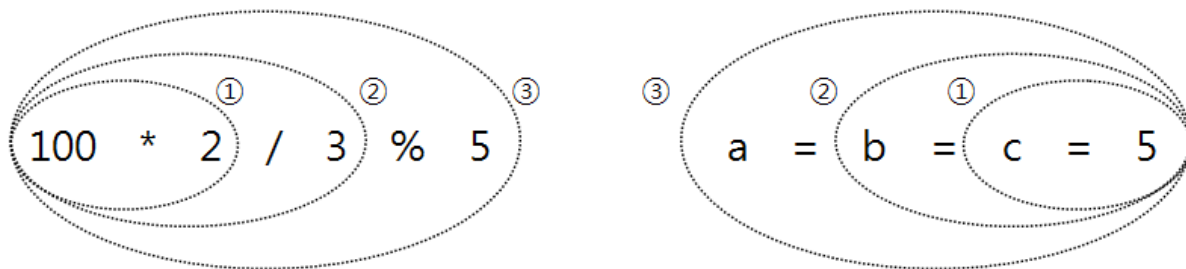
## ■ 연산 방향과 우선순위

- 연산자의 우선 순위에 따라 연산된다.

```
x > 0 && y < 0
```

- 동일한 우선 순위의 연산자는 연산의 방향 존재

\*, /, %는 같은 우선 순위를 갖고 있다. 이들 연산자는 연산 방향이 왼쪽에서 오른쪽으로 수행된다. 100 \* 2가 제일 먼저 연산되어 200이 산출되고, 그 다음 200 / 3이 연산되어 66이 산출된다. 그 다음으로 66 % 5가 연산되어 1이 나온다.



하지만 단항 연산자(++ , -- , ~ , !), 부호 연산자(+ , -), 대입 연산자(= , += , -= , ...)는 오른쪽에서 왼쪽(←)으로 연산된다. 예를 들어 다음 연산식을 보자.

## ■ 연산자 우선순위

연산자	연산 방향	우선 순위
증감(++), 부호(+, -), 비트(~), 논리(!)	←	<div>높음</div> <div>↑</div> <div>↓</div> <div>낮음</div>
산술(*, /, %)	→	
산술(+, -)	→	
쉬프트(<<, >>, >>>)	→	
비교(<, >, <=, >=, instanceof)	→	
비교(==, !=)	→	
논리(&)	→	
논리(^)	→	
논리( )	→	
논리(&&)	→	
논리(  )	→	
조건(?:)	→	
대입(=, +=, -=, *=, /=, %=, &=, ^=,  =, <<=, >>=, >>>=)	←	

## ■ 연산자 우선순위

- 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.

## ■ 연산자 우선순위

●  $-x + 3$  → 단항 > 이항

●  $x + 3 * y$  → 곱셈, 나눗셈 > 덧셈, 뺄셈

●  $x + 3 > y - 2$  → 산술 > 비교

●  $x > 3 \ \&\& \ x < 5$  → 비교 > 논리

● `int result = x + y * 3;` → 항상 대입은 맨 끝에

## ■ 단항연산자

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

```
int i = 5;
```

```
int j = 0;
```

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.

## ■ 단항연산자

- 부호연산자(+, -) : + 는 피연산자에 1을 곱하고  
- 는 피연산자에 -1을 곱한다.

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

```
i = +i;
```

```
i = -i;
```

- 논리부정연산자(!) : true는 false로, false는 true로  
피연산자가 boolean일 때만 사용가능

```
boolean power = false;
```

```
power = !power; // true
```

```
power = !power; // false
```



## ■ 단항연산자 사용

ch03.Operation1

```
int a = 10;

a = -a; // a의 값을 음수로
System.out.println(a);
System.out.println(a++); // a의 값 출력 후 증가
System.out.println(a); // 위의 증가된 a의 값 확인
System.out.println(--a); // a의 값 감소 후 출력

boolean b = true;
System.out.println(b);

b = !b; // b의 값 논리 부정
System.out.println(b);

char c1 = 'a';
char c2 = c1++;
char c3 = ++c1;
System.out.println(c2);
System.out.println(c3);
```

```
-10
-10
-9
-10
true
false
a
c
```

## ■ 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
...	...

## ■ 이항연산자 사용

ch03.Operation2

```
// 곱셈
long l = 10000L * 10000L;
System.out.println(l);

int a = 10;
int b = 5;
int result = ++a + b++;
System.out.println(result);
System.out.println(a);
System.out.println(b);
// 비교
int c = 10;
int d = 20;
boolean result2 = c > d;
System.out.println(result2);

result2 = c != d;
System.out.println(result2);

result2 = (c + 10) != d;
System.out.println(result2);
```

```
1000000000
16
11
6
false
true
false
```

## ■ 나머지 연산자 - %

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

```
int share = 10 / 8;
```

```
int remain = 10 % 8;
```



## ■ 비트 논리 연산자 - & | ^

- & (논리곱)

A	B	결과
0	0	0
0	1	0
1	0	0
1	1	1

- | (논리합)

A	B	결과
0	0	0
0	1	1
1	0	1
1	1	1

- ^ (배타적 논리합)

A	B	결과
0	0	0
0	1	1
1	0	1
1	1	0

## ■ 비트 이동 연산자 - <<, >>

- $2^n$ 으로 곱하거나 나눈 결과를 반환한다.
- 곱셈, 나눗셈보다 빠르다.

$x \ll n$  은  $x * 2^n$ 과 같다.

$x \gg n$  은  $x / 2^n$ 과 같다.

$8 \ll 2$  는  $8 * 2^2$ 과 같다.

$8 \gg 2$  는  $8 / 2^2$ 과 같다.

### ✓ 비트연산의 특징

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

- 정수 2 → 00000010 → 정수 2
- $2 \ll 1$  → 00000100 → 정수 4
- $2 \ll 2$  → 00001000 → 정수 8
- $2 \ll 3$  → 00010000 → 정수 16

## ■ 비트 이동 연산자 사용

ch03.Operation3

```
// 비트 논리 연산자
byte and1 = 1; byte and2 = 1;
System.out.println(and1 & and2);

and1 = 0;
System.out.println(and1 & and2);

byte or1 = 1; byte or2 = 0;
System.out.println(or1 | or2);

or1 = 0;
System.out.println(or1 | or2);

byte xor1 = 1; byte xor2 = 1;
System.out.println(xor1 ^ xor2);

xor1 = 0;
System.out.println(xor1 ^ xor2);

// 비트 이동 연산자
byte b = 10; // 00001010
           // 00101000
System.out.println(b << 2);

byte b2 = 10; // 00001010
             // 00000010
System.out.println(b2 >> 2);
```

1  
0  
1  
0  
0  
1  
40  
2

## ■ 비교연산자 - > < >= <= == !=

- 피연산자를 같은 타입으로 변환한 후에 비교한다.  
결과 값은 true 또는 false이다.
- 기본형(boolean제외)과 참조형에 사용할 수 있으나  
참조형에는 ==과 !=만 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

■ 비교연산자 - > < >= <= == !=

'A' < 'B' → 65 < 66 → true

'0' == 0 → 48 == 0 → false

'A' != 65 → 65 != 65 → false

## ■ 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.

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 사용

ch03.Operation4

```
boolean result;

int num1 = 0;
int num2 = 0;

/* Short-Circuit Evaluation */
result = num1++ < 0 && num2++ > 0;
System.out.println("result : " + result);
System.out.println("num1 : " + num1 + ", num2 : " + num2);

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

```
result : false
num1 : 1, num2 : 0
result : true
num1 : 2, num2 : 0
```



## ■ 논리연산자 사용

ch03.Operation4

```
int num3 = 0;
int num4 = 0;

result = num3++ < 0 & num4++ > 0;
System.out.println("result : " + result);
System.out.println("num3 : " + num3 + ", num4 : " + num4);

result = num3++ > 0 | num4++ > 0;
System.out.println("result : " + result);
System.out.println("num3 : " + num3 + ", num4 : " + num4);
```

```
result : false
num3 : 1, num4 : 1
result : true
num3 : 2, num4 : 2
```

## ■ 대입연산자 - = op=

- 오른쪽 피연산자의 값을 왼쪽 피연산자에 저장한다.  
단, 왼쪽 피연산자는 상수가 아니어야 한다.

```
int i = 0;
```

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

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

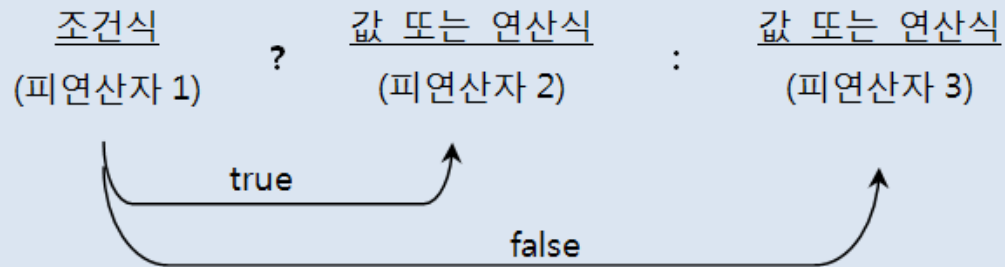
```
MAX = 10; // 에러
```

op=	=
i += 3;	i = i + 3;
i -= 3;	i = i - 3;
i *= 3;	i = i * 3;
i /= 3;	i = i / 3;
i %= 3;	i = i % 3;
i <<= 3;	i = i << 3;
i >>= 3;	i = i >> 3;
i >>>= 3;	i = i >>> 3;
i &= 3;	i = i & 3;
i ^= 3;	i = i ^ 3;
i  = 3;	i = i   3;
i *= 10 + j;	i = i * (10+j) ;

## ■ 삼항연산자 - ? :

### ● 세 개의 피연산자를 필요로 하는 연산자

- 조건식의 연산결과가 true이면 '식1' 결과 반환, false이면 '식2' 결과 반환  
(조건식) ? 식1 : 식2



```
int score = 95;  
char grade = (score > 90) ? 'A' : 'B'
```

=

```
int score = 95;  
char grade;  
if(score > 90) {  
    grade = 'A';  
} else {  
    grade = 'B';  
}
```

## ■ 삼항연산자 사용

ch03.Operation5

```
int score = 0;

System.out.print("숫자 입력 > ");

Scanner scan = new Scanner(System.in);
score = scan.nextInt();

char grade = score >= 90 ? 'A' : (score >= 80 ? 'B' : 'C');

System.out.println(grade);

scan.close();
```

숫자 입력 > 90  
A

## ■ 연습문제 (ch03.연습문제01)

### ● 결과와 같이 출력될 수 있도록 코드 작성하기

```
char first = 'A';  
char second = 'a';  
  
( ① ) result = first ( ② ) second;  
  
System.out.printf("A 가 a 보다 ASCII 값이 더 크다 => %s\n", result);
```

결과

```
A 가 a 보다 ASCII 값이 더 크다 => false
```

## ■ 연습문제 (ch03.연습문제02)

### ● 결과와 같이 출력될 수 있도록 코드 작성하기

```
int number = 77;  
  
int ten = number ( ① ) 10;  
int one = number ( ② ) 10;  
  
boolean isMatch = ten ( ③ ) one;  
  
System.out.printf("십의 자리와 일의 자리 숫자가 같다 : %s", isMatch);
```

#### 결과

십의 자리와 일의 자리 숫자가 같다 : true

## ■ 연습문제 (ch03.연습문제03)

### ● 제시된 num의 각 자리 숫자 합 구하기

```
int num = 12345;

int num1 = ( ① );
int num2 = ( ② );
int num3 = ( ③ );
int num4 = ( ④ );
int num5 = ( ⑤ );

int total = num1 + num2 + num3 + num4 + num5;
System.out.println("각 자리 숫자의 합 : " + total);
```

결과

각 자리 숫자의 합 : 15

## ■ 연습문제 (ch03.연습문제04)

- 입력된 값에 따라 짝수 또는 홀수 출력하기

```
int num = 10;
```

```
System.out.println( ① );
```

결과

짝수



## ■ 연습문제 (ch03.연습문제05)

### ● 대문자를 소문자로 변경하기

- 대문자 A의 ASCII 값 → 65

- 소문자 a의 ASCII 값 → 97

```
char ch = 'T';

char lowerCase = ( ① ) ? ( ② ) : ch;

System.out.println("입력된 문자 : " + ch);
System.out.println("소문자로 변경된 문자 : " + lowerCase);
```

### 결과

입력된 문자 : T

소문자로 변경된 문자 : t