

CH 3

연산자

피연산자

연산자의 종류

연산자의 우선 순위

연산자의 결합 규칙

증감 연산자

부호 연산자

형변환 연산자

자동 형변환

사칙 연산자

산술 변환

반올림 - Math.round()

나머지 연산자 %

비교 연산자

논리 연산자

논리 부정 연산자 !

조건 연산자 ?:

대입 연산자

복합 대입 연산자

연산자

- 연산을 수행하는 기호

피연산자

- 연산자의 연산 수행 대상



모든 연산자는 연산 결과를 반영한다

연산자의 종류

5

A 3 1 0 5 B

종류	연산자	설명
산술 연산자	+ - * / % << >>	사칙 연산과 나머지 연산(%)
비교 연산자	> < >= <= == !=	크고 작음과 같고 다른을 비교
논리 연산자	&& ! & ^ ~	'그리고(AND)'와 '또는(OR)'으로 조건을 연결
대입 연산자	=	우변의 값을 좌변에 저장
기 타	(type) ?: instanceof	형변환 연산자, 삼항 연산자, instanceof연산자

연산자의 우선 순위

- 하나의 식에 연산자가 둘 이상 있을 때 어떤 연산을 먼저 수행할지를 자동 결정
 - 괄호 사용하면 수동으로 결정

종 류	결합규칙	연산자	우선순위
단항 연산자	←	++ -- + - ~ ! (type)	높음
산술 연산자	→	* / %	↓
	→	+ -	
	→	<< >>	
비교 연산자	→	< > <= >= instanceof	
	→	== !=	
논리 연산자	→	&	
	→	^	
	→		
	→	&&	
	→		
삼항 연산자	→	? :	
대입 연산자	←	= += -= *= /= %= <<= >>= &= ^= =	낮음

▲ 표 3-4 연산자의 우선순위와 결합규칙

식	설명
$-x + 3$	단항 연산자가 이항 연산자보다 우선순위가 높다. 그래서 x 의 부호를 바꾼 다음 덧셈이 수행된다. 여기서 ' $-$ '는 뺄셈 연산자가 아니라 부호 연산자이다.
$x + 3 * y$	곱셈과 나눗셈이 덧셈과 뺄셈보다 우선순위가 높다. 그래서 ' $3 * y$ '가 먼저 계산된다.
$x + 3 > y - 2$	비교 연산자($>$)보다 산술 연산자 ' $+$ '와 ' $-$ '가 먼저 수행된다. 그래서 ' $x + 3$ '과 ' $y - 2$ '가 먼저 계산된 다음에 ' $>$ '가 수행된다.
$x > 3 \&& x < 5$	논리 연산자 ' $\&\&$ '보다 비교 연산자가 먼저 수행된다. 그래서 ' $x > 3$ '와 ' $x < 5$ '가 먼저 계산된 다음에 ' $\&\&$ '가 수행된다. 식의 의미는 ' x 가 3보다 크고 5보다 작다'이다.
<code>result = x + y * 3;</code>	대입 연산자는 연산자 중에서 제일 우선순위가 낮다. 그래서 우변의 최종 연산결과가 변수 <code>result</code> 에 저장된다.

- 1항 연산자가 2항 연산자보다 우선 순위가 높다
- 확실하지 않으면 괄호로 묶어서 계산하기 → 괄호가 가장 먼저 계산되기 때문

연산자의 결합 규칙

- 우선 순위가 같은 연산자가 있을 때, 어떤 것을 먼저 계산할 것인가?
- 대입과 단항 연산자를 제외하면 모두 왼쪽에서 오른쪽 으로
- 대입과 단항 연산자는 오른쪽부터

연산자의 우선순위와 결합법칙은
"세 가지만 기억하자"

1. 산술 $>$ 비교 $>$ 논리 $>$ 대입. 대입은 제일 마지막에 수행된다.
2. 단항(1) $>$ 이항(2) $>$ 삼항(3). 단항 연산자의 우선순위가 이항 연산자보다 높다.
3. 단항 연산자와 대입 연산자를 제외한 모든 연산의 진행방향은 왼쪽에서 오른쪽이다.

증감 연산자



증가 연산자(++) 피연산자의 값을 1 증가
감소 연산자(- -) 피연산자의 값을 1 감소

타입	설명	사용예
앞 전위형	값이 참조되기 전에 증가시킨다.	j = ++i;
뒤 후위형	값이 참조된 후에 증가시킨다.	j = i ++ ;

- 증감 연산자가 독립적으로 사용된 경우, 전위형과 후위형의 차이가 없다

증감 연산자가 포함된 식을 이해하기 어려울 때는 다음과 같이 증감 연산자를 따로 빼어내면 이해하기가 쉬워진다. 전위형의 경우 증감 연산자를 식의 이전으로,

(j = ~~++i;~~ // 전위형) →
 ~~++i;~~ // 증가 후에
 j = i; // 참조하여 대입

후위형의 경우 증감 연산자를 식의 이후로 빼어내면 된다.

j = i~~++~~; // 후위형 →
 j = i; // 참조하여 대입 후에
 i~~++~~; // 증가

```
int i = 5, j = 0;

// 후위형
//j = i++;
j = i;
i++;
System.out.println("j = i++; 실행 후, i=" + i + ", j=" + j);
// j = 5, i = 6

i = 5;
j = 0;
// 전위형
//j = ++i;
++i;
j = i;
System.out.println("j = ++i; 실행 후, i=" + i + ", j=" + j);
// j = 6, i = 6
```

부호 연산자

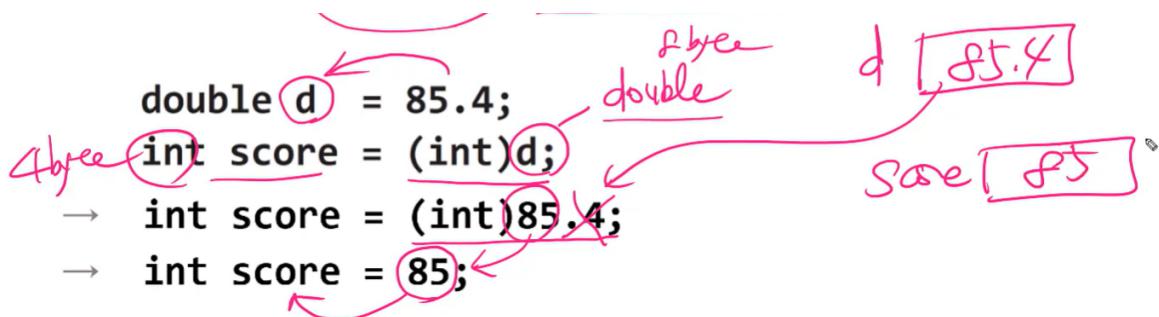
- $-$ 는 피연산자의 부호를 반대로 변경
- $+$ 는 아무 일도 하지 X (실제 사용 X)
- 둘 다 단항 연산자 (피연산자 1개)

형변환 연산자

- 변수, 상수의 타입을 다른 타입으로 변환하는 것



(타입) 피연산자



- 읽어 온 값인 85.4를 형 변환 한 것이기 때문에, 변수 d의 값은 변하지 않았다

변환	수식	결과
int \rightarrow char	(char)65	'A'
char \rightarrow int	(int)'A'	65
float \rightarrow int	(int)1.6f	1
int \rightarrow float	(float)10	10.0f

▲ 표 3-1 형변환의 다양한 예시

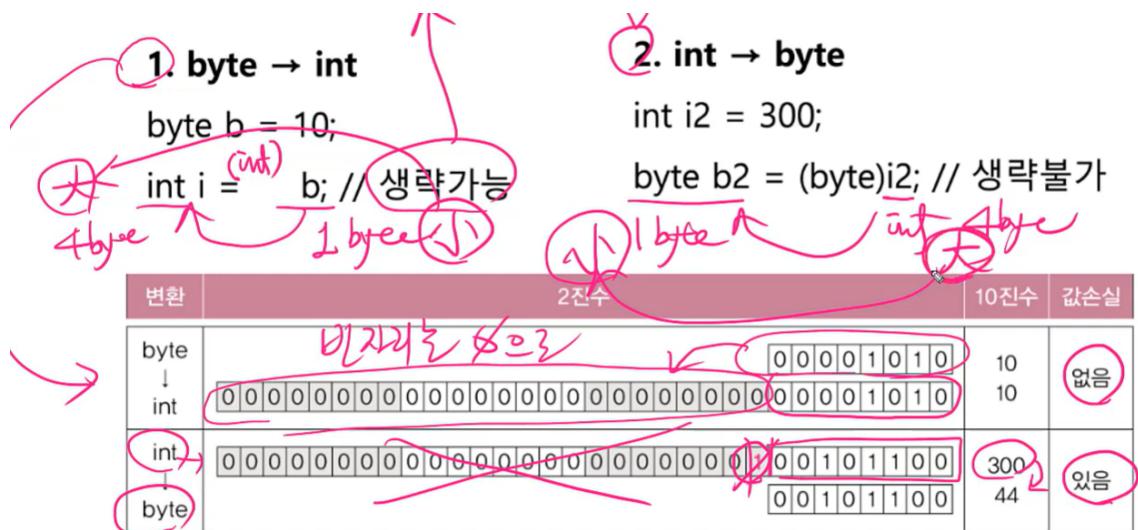
ASCII (아스키코드표)											
10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자	10진수	16진수	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	`
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	\$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(72	0x48	H	104	0x68	h
9	0x09	HT	41	0x29)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	₩	124	0x7C	
29	0x1D	GS	61	0x3D	=	93	0x5D]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	-	127	0x7F	DEL

자동 형변환

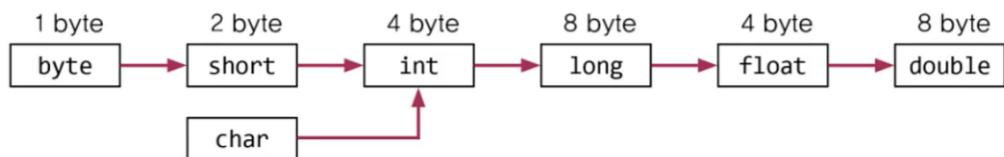
```
float f = 1234; // int 타입 값을 float 타입 변수에 저장 (형 변환 생략)  
// 작은 타입의 값을 큰 타입에 넣기 때문에 문제가 생기지 않는다
```

- 컴파일러가 자동으로 형 변환을 해 줘 형 변환을 생략한다

```
int i = 3.14f; // 에러
// 큰 타입의 값을 작은 타입에 넣으면 에러가 발생한다 => 값 손실 발생
int i = (int) 3.14f // 수동 형 변환 필요
```



그래서 표현범위가 좁은 타입에서 넓은 타입으로 형변환하는 경우에는 값 손실이 없으므로 두 타입 중에서 표현범위가 더 넓은 쪽으로 형변환된다.



byte b = 100; // OK. byte타입의 범위(-128 ~ 127)의 값의 대입
byte b = (byte)100; // OK. byte타입으로 형변환하여 대입

int i = 100;
byte b = i; // 에러 int타입을 byte의 타입에 대입
byte b = (byte)i; // OK. byte타입으로 형변환하여 대입

byte b = 1000; // 에러 byte타입의 범위(-128 ~ 127)를 벗어난 값의 대입
byte b = (byte)1000; // OK. 그러나 값 손실이 발생해서 변수 b에는 -24가 저장됨.

- 값 손실이 발생하는 경우에는 컴파일러가 자동 형 변환을 못 한다

사직 연삼자

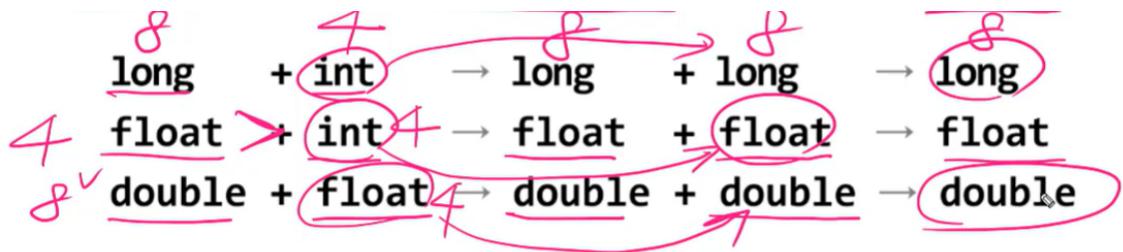
- + - * /

~~int~~ ~~int~~ ~~int~~ // 소수점 이하는 버려진다.

```
int    float      float    float      float
 10 / 4.0f  →  10.0f / 4.0f  →  2.5f
```

산술 변화

- 연산 전에 피연산자의 탑입을 일치시키는 것
 1. 두 피연산자의 탑입을 같게 일치시킨다. (큰 탑입으로 일치)



2. 피연산자의 타입이 int보다 작은 타입이면 int로 변환된다.

~~byte char, short~~

byte + short → **int + int** → **int**
~~char + short~~ → **int + int** → **int**

```
int a = 1_000_000;
int b = 2_000_000;

long c = a * b;
// 저장한 값을 long에 담으려고 해도 a*b가 이미 int이기 때문에 overflow 에러가 발생
// 해결하려면 a나 b를 long으로 형 변환 해 주어야 한다
long c = (long)a * b;
long c = a * (long)b;
```

long c = (long)a * b;
 → **long c = (long)1000000 * 2000000;**
 → **long c = 1000000L * 2000000;**
 → **long c = 1000000L * 2000000L;**
 → **long c = 200000000000L;**

반올림 - Math.round()

- 실수를 소수점 첫째 자리에서 반올림한 정수 반환

예제

3-11

```
class Ex3_11 {
    public static void main(String args[]) {
        double pi = 3.141592;
        double shortPi = Math.round(pi * 1000) / 1000.0;
        System.out.println(shortPi);
    }
}
```

Math.round(pi * 1000) / 1000.0 int int int
 → Math.round(3.141592 * 1000) / 1000.0 3142 / 1000 → 3
 → Math.round(3142.592) / 1000.0
 → 3142 / 1000.0 int double double double
 → 3.142 double 3142 / 1000.0 → 3142.0 / 1000.0 → 3.142

```
3     double pi = 3.141592;
4
5     System.out.println(pi);
6     System.out.println(pi*1000);
7     System.out.println(Math.round(pi*1000));
8     System.out.println(Math.round(pi*1000)/1000); // 3
9     System.out.println((double)Math.round(pi*1000)/1000); // 3.142
10
11    //     double shortPi = Math.round(pi * 1000) / 1000.0;
12    System.out.println(shortPi);
13
14 }
```

```
Problems @ Javadoc Declaration Console
<terminated> Ex3_11 [Java Application] C:\jdk1.8\bin\javaw.exe (2020. 1. 15. 오후 2:23:55)
3.141592
3141.592
3142
3
3.142
```

```

2  public static void main(String args[]) {
3      double pi = 3.141592; // 3.141을 얻으려면?
4
5      System.out.println(pi*1000);
6      System.out.println((int)(pi*1000));
7      System.out.println((int)(pi*1000)/1000);
8      System.out.println((int)(pi*1000)/1000.0);
9
10     //      double shortPi = Math.round(pi * 1000) / 1000.0;
11 //      System.out.println(shortPi);
12 }
13 }
```



Problems @ Javadoc Declaration Console

<terminated> Ex3_11 [Java Application] C:\jdk1.8\bin\javaw.exe (2020. 1. 15. 오후 2:27:17)

3141.592
3141
3
3.141

나머지 연산자 %

- 오른쪽 피연산자로 나누고 남은 나머지를 반환

비교 연산자

- 두 피연산자를 비교해서 true 또는 false를 반환

비교연산자	연산결과
>	좌변 값이 크면 , true 아니면 false
<	좌변 값이 작으면 , true 아니면 false
>=	좌변 값이 크거나 같으면 , true 아니면 false
<=	좌변 값이 작거나 같으면 , true 아니면 false

▲ 표 3-2 대소비교 연산자의 종류와 연산결과

비교연산자	연산결과
==	두 값이 같으면 , true 아니면 false
!=	두 값이 다르면 , true 아니면 false

▲ 표 3-3 등가비교 연산자의 종류와 연산결과

- 문자열 비교에는 == 대신 `equals()` 를 사용해야 한다

논리 연산자

- 조건식을 연결할 때 사용하는 연산자

|| (OR결합) 피연산자 중 어느 한 쪽이 true이면 true를 결과로 얻는다.

&& (AND결합) 피연산자 양쪽 모두 true이어야 true를 결과로 얻는다.

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

③ i는 2의 배수 또는 3의 배수지만 6의 배수는 아니다.

이전 조건에 6의 배수를 제외하는 조건이 더 불었다. 6의 배수가 아니어야 한다는 조건은 ' $i \% 6 \neq 0$ '이고, 이 조건을 '&&(AND)'로 연결해야 한다.

$(i \% 2 == 0 \text{ } || \text{ } i \% 3 == 0) \text{ } \&\& \text{ } i \% 6 \neq 0$

위의 식에 괄호를 사용한 이유는 '&&'가 '||'보다 우선순위가 높기 때문이다. 만일 괄호를 사용하지 않으면 '&&'를 먼저 연산한다. 다음의 두 식은 동일하다.

$i \% 2 == 0 \text{ } || \text{ } i \% 3 == 0 \text{ } \&\& \text{ } i \% 6 \neq 0$
 $i \% 2 == 0 \text{ } || \text{ } (i \% 3 == 0 \text{ } \&\& \text{ } i \% 6 \neq 0)$

- &&가 ||보다 우선 순위가 더 높다

④ 문자 ch는 숫자('0'~'9')이다.

사용자로부터 입력된 문자가 숫자('0'~'9')인지 확인하는 식은 다음과 같이 쓸 수 있다.

$'0' \leq ch \text{ } \&\& \text{ } ch \leq '9'$ ✓

유니코드에서 문자 '0'부터 '9'까지 연속적으로 배치되어 있기 때문에 가능한 식이다. 문자 '0'부터 '9'까지 유니코드는 10진수로 다음과 같다.

문자	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'
문자코드	48	49	50	51	52	53	54	55	56	57

⑤ 문자 ch는 대문자 또는 소문자이다.

④의 경우와 마찬가지로 문자 'a'부터 'z'까지, 그리고 'A'부터 'Z'까지도 연속적으로 배치되어 있으므로 문자 ch가 대문자 또는 소문자인지 확인하는 식은 다음과 같이 쓸 수 있다.

$('a' \leq ch \text{ } \&\& \text{ } ch \leq 'z') \text{ } || \text{ } ('A' \leq ch \text{ } \&\& \text{ } ch \leq 'Z')$

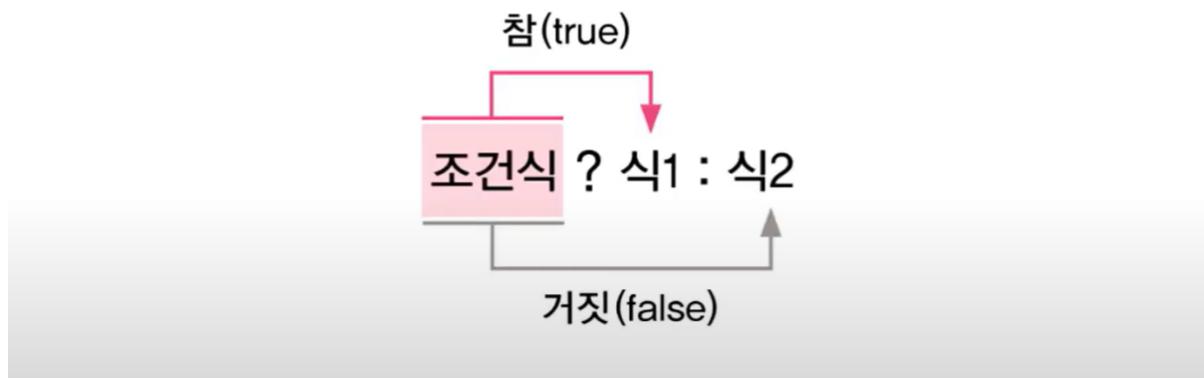
논리 부정 연산자 !

- true를 false로, false를 true로 바꾼다

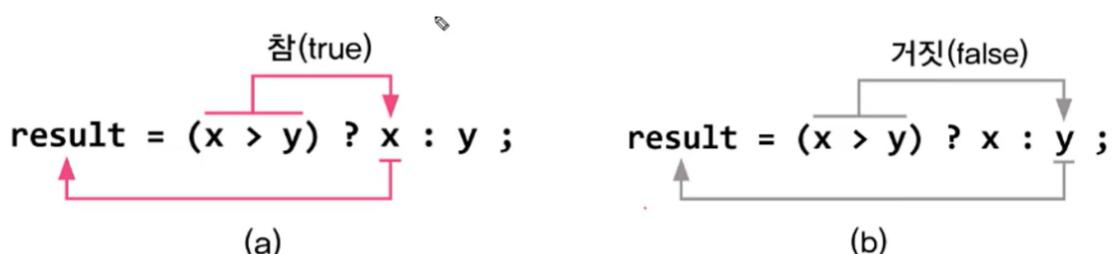
x	$!x$
true	\rightarrow false
false	\rightarrow true

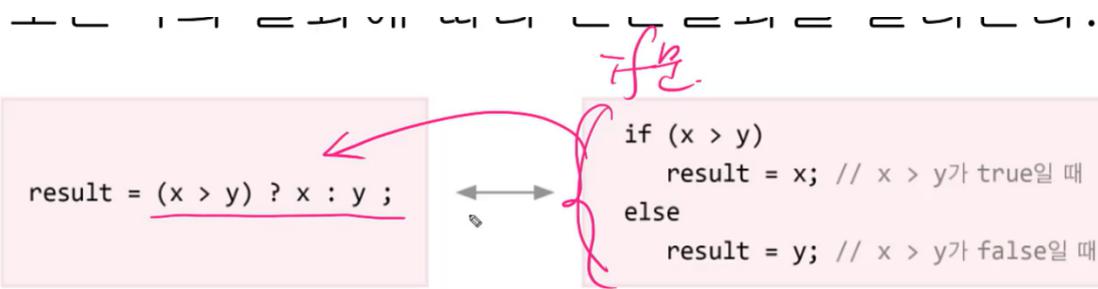
조건 연산자 ? :

- 조건식의 결과에 따라 연산 결과를 달리한다
- 삼항 연산자



result = 조건식 ? 식1 : 식2 ; // 괄호 생략 가능

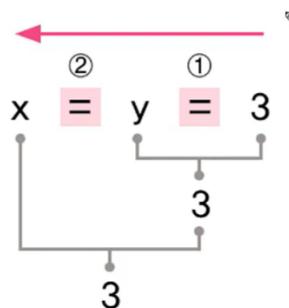




대입 연산자

- 오른쪽 피연산자를 왼쪽 피연산자에 저장 후 저장된 값을 반환

System.out.println(x = 3); 변수 x에 3이 저장되고
→ System.out.println(3); 연산결과인 3이 출력된다.



lvalue – 대입 연산자의 왼쪽 피연산자

rvalue – 대입 연산자의 오른쪽 피연산자

int i = 0; // 예러. lvalue가 값을 저장할 수 있는 공간이 아니다.
 3 = i + 3; // 예러. lvalue의 연산결과가 리터럴(i+3 → 0+3 → 3)
 i + 3 = i;

final int MAX = 3; // 변수 앞에 키워드 final을 붙이면 상수가 된다.
MAX = 10; // 예러. 상수(MAX)에 새로운 값을 저장할 수 없다.

복합 대입 연산자

- 대입 연산자와 다른 연산자를 하나로 축약

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 $\ = 3;$	$i = i \mid 3;$
i $*= 10 + j;$	$i = i * (10 + j);$

- 마지막에는 괄호를 꼭 치고 변환