

기본 입출력

■ 표준 입출력



기본 입출력

■ 화면에 데이터 출력

- `println()` : 내용을 출력한 후 행을 바꾼다.
- `print()` : 내용을 출력만 하고 행은 바꾸지 않는다.
- `printf()` : 포맷을 지정해서 출력한다.

■ `printf()` 형식

```
System.out.printf("포맷 명시자", 데이터, 데이터, ...);
```

```
int x = 5;
```

```
double pi = 3.14;
```

```
System.out.printf("x = %d and pi = %f\n", x, pi);
```

x 변수를 십진수 정수 포맷과 대응시킨다.

데이터 항목들

포맷 명시자

변수 pi를 십진수 실수 포맷과 대응시킨다.

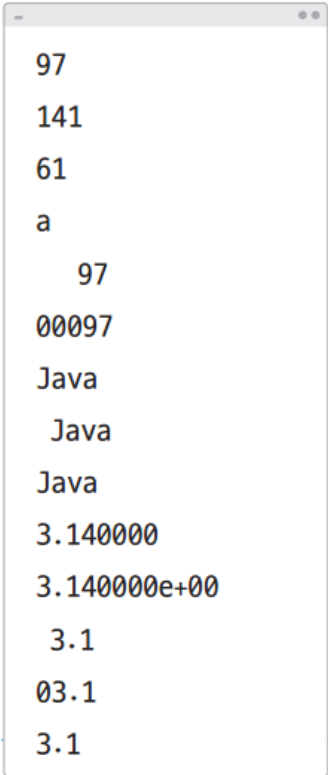


기본 입출력

■ 예제

● [sec04/PrintfDemo](#)

```
05      int i = 97;
06      String s = "Java";
07      double f = 3.14f;
08      System.out.printf("%d\n", i);
09      System.out.printf("%o\n", i);
10      System.out.printf("%x\n", i);
11      System.out.printf("%c\n", i);
12      System.out.printf("%5d\n", i);
13      System.out.printf("%05d\n", i);
14      System.out.printf("%s\n", s);
15      System.out.printf("%5s\n", s);
16      System.out.printf("%-5s\n", s);
17      System.out.printf("%f\n", f);
18      System.out.printf("%e\n", f);
19      System.out.printf("%4.1f\n", f);
20      System.out.printf("%04.1f\n", f);
21      System.out.printf("%-4.1f\n", f);
```



97
141
61
a
97
00097
Java
Java
Java
3.140000
3.140000e+00
3.1
03.1
3.1



기본 입출력

■ printf()의 포맷과 실행 결과

종류	데이터	포맷	실행 결과	설명
정수	97	%d	97	10진수
		%o	141	8진수
		%x	61	16진수
		%c	a	문자
		%5d	97	5자리. 빈자리는 공백 처리한다.
		%-5d	97	5자리. 빈자리는 공백 처리한다. 왼쪽 정렬
		%05d	00097	5자리. 빈자리는 0으로 채운다.
문자열	"java"	%s	"java"	문자열
		%5s	" java"	5자리. 빈자리는 공백 처리한다.
		%-5s	"java "	5자리. 빈자리는 공백 처리한다. 왼쪽 정렬
실수	3.14f	%f	3.140000	10진수 실수
		%e	3.140000e+00	지수
		%4.1f	3.1	4자리. 소수점 이하 1자리
		%04.1f	03.1	4자리. 소수점 이하 1자리. 빈자리 0
		%-4.1f	3.1	4자리. 소수점 이하 1자리. 왼쪽 정렬



기본 입출력

■ 키보드로 데이터 입력

- 프로그램의 첫 행에 다음을 추가해 Scanner 클래스의 경로 이름을 컴파일러에 알린다.

```
import java.util.Scanner;
```

- 키보드로 데이터를 입력 받기 위해 System.in 객체와 연결된 Scanner 객체를 생성한다.

```
Scanner in = new Scanner(System.in);
```

- Scanner 클래스가 제공하는 다양한 메서드를 이용해 키보드로 데이터를 입력 받는다.

```
int x = in.nextInt(); // 정수를 읽어 변수 x에 대입한다.
```



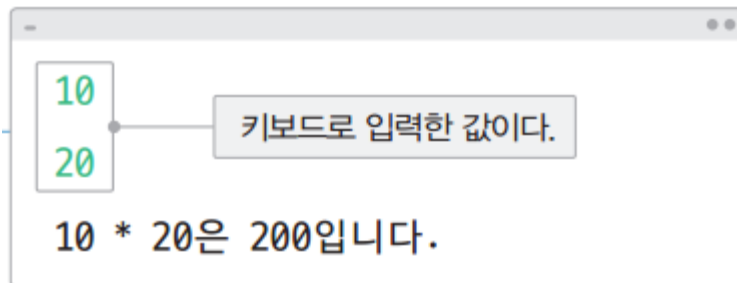
기본 입출력

■ 키보드로 데이터 입력

- Scanner 클래스가 제공하는 데이터 입력 메서드

- 예제 :

[sec04/ScannerDemo](#)



메서드	반환 타입
next()	String
nextByte()	byte
nextShort()	short
nextInt()	int
nextLong()	long
nextFloat()	float
nextDouble()	double
nextLine()	String

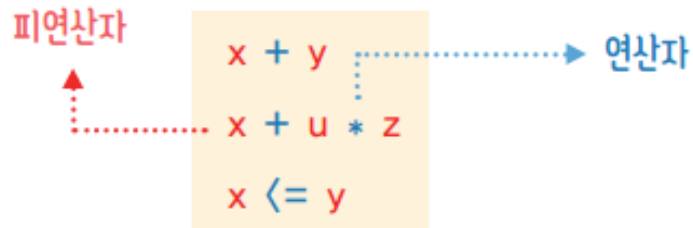
```
ScannerDemo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
import java.util.Scanner;

public class ScannerDemo {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int x = in.nextInt();
        int y = in.nextInt();
        System.out.printf("%d * %d은 %d입니다.\n", x, y, x * y);
    }
}
```



연산자

■ 연산자와 연산식의 의미



■ 자바 가상 머신은 기본적으로 32비트 단위로 계산

```
byte b1 = 1;  
byte b2 = 2;  
byte b3 = b1 + b2;    // 오류 발생
```



연산자

■ 종류

종류	연산자	설명	비고
증감	++, --	1만큼 증가 또는 감소한다.	단항
산술	+, -, *, /, %	사칙 연산과 모듈로 연산한다.	이항
시프트	>>, <<, >>>	비트를 좌우로 이동한다.	이항
부호	+, -	부호를 변환한다.	단항
비교	>, <, >=, <=, !=, instanceof	데이터 값을 비교하거나 데이터 타입을 비교한다.	이항
비트	&, , ~, ^	비트 단위의 AND, OR, NOT, XOR	단항, 이항
논리	&&, , !, ^	논리적 AND, OR, NOT, XOR	단항, 이항
조건	(expr) ? x : y	expr에 따라 x 또는 y로 값을 결정한다.	삼항
대입	=, +=, -=, *=, /=, &=, =, ^=, >>=, <<=, >>>=	오른쪽 값을 연산해 왼쪽에 대입한다.	이항



연산자

■ 산술 연산자

- 피연산자의 데이터 타입에 따라 결과 값이 다른데, 연산할 두 피연산자의 데이터 타입이 다르면 큰 범위의 타입으로 일치시킨 후 연산 수행
- 논리 타입을 제외한 기초 타입을 피연산자로 사용. 단, % 연산자는 정수 타입만 사용
- 덧셈 연산자는 문자열을 연결하는 데도 사용. 문자열과 덧셈을 하는 데이터는 먼저 문자열로 변환한 후 서로 연결

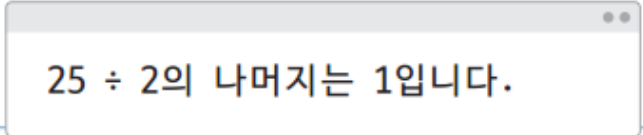
```
// 짝수와 홀수 여부 판단. a가 1이면 n은 홀수, 0이면 짝수
```

```
int a = n % 2;
```

```
// 3의 배수인지 확인, b가 0이면 n은 3의 배수
```

```
int b = n % 3;
```

- 예제 : [sec05/ArithmeticDemo](#)



25 ÷ 2의 나머지는 1입니다.



연산자

- 예제 : [sec05/ArithmeticDemo](#)

25 ÷ 2의 나머지는 1입니다.

```
ArithmeticDemo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
public class ArithmeticDemo {
    public static void main(String[] args) {
        int remainder = 25 % 2;
        System.out.println("25 ÷ 2의 나머지는 " + remainder + "입니다.");
    }
}
```

문제)

숫자를 입력하시오 : **24**

24 ÷ 2의 나머지는 0 입니다.



연산자

■ 비교 연산자

- 비교 연산자는 논리 타입을 제외한 기초 타입에만 사용할 수 있지만 ==와 !=는 모든 기초 타입에 사용
- 종류

연산자	사용 예	설명
==	$x == y$	x와 y는 같은가?
!=	$x != y$	x와 y가 다른가?
>	$x > y$	x는 y보다 큰가?
>=	$x >= y$	x는 y보다 크거나 같은가?
<	$x < y$	x는 y보다 작은가?
<=	$x <= y$	x는 y보다 작거나 같은가?



연산자

■ 논리 연산자

- 논리 연산자는 피연산자의 조건을 결합해서 true와 false를 조사하며, 논리 타입에만 사용
- 종류

a	b	!a	a && b	a b	a ^ b
false	false	true	false	false	false
false	true	true	false	true	true
true	false	false	false	true	true
true	true	false	true	true	false

- 쇼트서킷

조건식1 && 조건식2

조건식1이 false이면 조건식2의 진릿값과 상관없이 결과가 무조건 false가 된다. 따라서 조건식2의 진릿값을 조사할 필요가 없다.

조건식1 || 조건식2

조건식1이 true이면 조건식2의 진릿값과 상관없이 결과가 무조건 true가 된다. 따라서 조건식2의 진릿값을 조사할 필요가 없다.

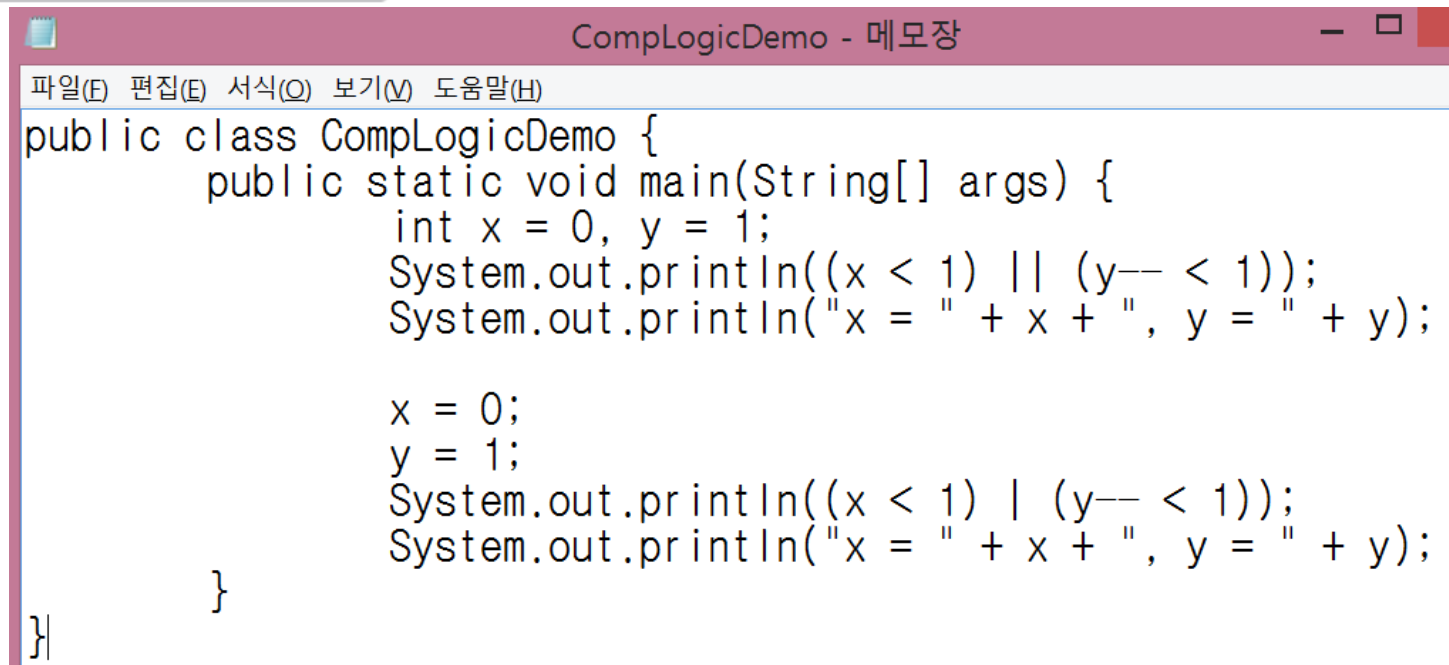


연산자

■ 논리 연산자

- 예제 : [sec05/CompLogicDemo](#)

```
true
x = 0, y = 1
true
x = 0, y = 0
```



```
CompLogicDemo - 메모장
파일(E) 편집(E) 서식(O) 보기(V) 도움말(H)

public class CompLogicDemo {
    public static void main(String[] args) {
        int x = 0, y = 1;
        System.out.println((x < 1) || (y-- < 1));
        System.out.println("x = " + x + ", y = " + y);

        x = 0;
        y = 1;
        System.out.println((x < 1) | (y-- < 1));
        System.out.println("x = " + x + ", y = " + y);
    }
}
```



연산자

■ 비트·시프트 연산자

- 비트 연산자와 시프트 연산자는 정수 타입에만 사용
- 비트 연산자의 종류

연산자	설명
&	두 비트가 모두 1일 때만 1이며, 나머지는 모두 0이다.
	두 비트가 모두 0일 때만 0이며, 나머지는 모두 1이다.
^	두 비트가 서로 다를 때는 1, 동일할 때는 0이다.
~	1을 0으로, 0을 1로 바꾼다.

- 예

0 1 0 1	0 1 0 1	0 1 0 1	
& 0 0 1 1	0 0 1 1	^ 0 0 1 1	~ 0 0 1 1
0 0 0 1	0 1 1 1	0 1 1 0	1 1 0 0



연산자

■ 비트·시프트 연산자

● 시프트 연산자의 종류

연산자	a 연산자 b일 경우 설명(예를 들어, a << b)
<<	a의 모든 비트를 왼쪽으로 b비트만큼 이동하며, 이동할 때마다 최하위 비트를 0으로 채운다. 곱셈 효과가 나타나기 때문에 산술적 왼쪽 시프트(Arithmetic Left Shift)라고 한다.
>>	a의 모든 비트를 오른쪽으로 b비트만큼 이동하며, 이동할 때마다 최상위 비트와 동일한 비트로 채운다. 나눗셈 효과가 나타나기 때문에 산술적 오른쪽 시프트(Arithmetic Right Shift)라고 한다.
>>>	a의 모든 비트를 오른쪽으로 b비트만큼 이동하며, 이동할 때마다 최상위 비트를 0으로 채운다. 산술적 효과가 없기 때문에 논리적 오른쪽 시프트(Logical Right Shift)라고 한다.

● 예

00000110
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
00000001

0b00000110 >> 2

오른쪽으로 2비트씩 이동
왼쪽 빈 2비트 공간을 00으로 채움

00000110
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
00001100

0b00000110 << 2

왼쪽으로 2비트씩 이동
오른쪽 빈 2비트 공간을 00으로 채움



연산자

■ 비트·시프트 연산자

- 예제 : [sec05/BitOperatorDemo](#)

```
03 public class BitOperatorDemo {
04     public static void main(String[] args) {
05         System.out.printf("%x\\n", 0b0101 & 0b0011);
06         System.out.printf("%x\\n", 0b0101 | 0b0011);
07         System.out.printf("%x\\n", 0b0101 ^ 0b0011);
08         System.out.printf("%x\\n", (byte) ~0b00000001);
09         System.out.printf("%x\\n", 0b0110 >> 2);
10         System.out.printf("%x\\n", 0b0110 << 2);
11
12         int i1 = -10;
13         int i2 = i1 >> 1;
14         int i3 = i1 >>> 1;
15         System.out.printf("%x -> %d\\n", i1, i1);
16         System.out.printf("%x -> %d\\n", i2, i2);
17         System.out.printf("%x -> %d\\n", i3, i3);
18     }
19 }
```

1

7

6

fe

1

18

ffffffff6 -> -10

fffffffb -> -5

7ffffffb -> 2147483643

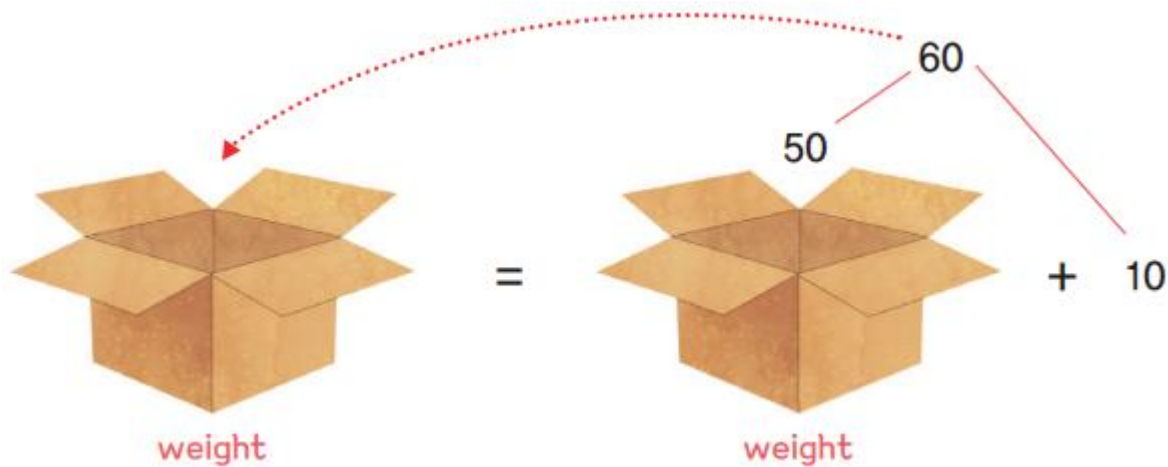


연산자

■ 대입 연산자

- 대입 연산자는 오른쪽에 있는 연산식의 결과 값을 왼쪽에 있는 변수에 대입
- 예

```
int weight = 50;  
weight = weight + 10;
```



연산자

■ 대입 연산자

- 복합 대입 연산자의 종류

- 예제 : [sec05/AssignmentDemo](#)

```
값 = 2
값 = 1
값 = 8
값 = 2
```

연산자	설명
$a += b$	$a = a + b$ 와 동일
$a -= b$	$a = a - b$ 와 동일
$a *= b$	$a = a * b$ 와 동일
$a /= b$	$a = a / b$ 와 동일
$a \% = b$	$a = a \% b$ 와 동일
$a \& = b$	$a = a \& b$ 와 동일
$a = b$	$a = a b$ 와 동일
$a \wedge = b$	$a = a \wedge b$ 와 동일
$a \gg = b$	$a = a \gg b$ 와 동일
$a \ll = b$	$a = a \ll b$ 와 동일

```
AssignmentDemo - 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
package sec05;

public class AssignmentDemo {
    public static void main(String[] args) {
        int value = 1;
        value += 1;
        System.out.println("값 = " + value);
        value -= 1;
        System.out.println("값 = " + value);
        value <<= 3;
        System.out.println("값 = " + value);
        value %= 3;
        System.out.println("값 = " + value);
    }
}
```



연산자

■ 부호·증감 연산자

- 숫자를 나타내는 기초 타입에 사용하며 피연산자의 부호를 그대로 유지하거나 반전
- 증감 연산자는 변수의 위치에 따라 의미가 다르다.
- 종류

연산자	설명
+	부호 유지
-	부호 반전

연산자	설명
++	++x 연산 전 x 값 증가(전위 증가)
	x++ 연산 후 x 값 증가(후위 증가)
--	--x 연산 전 x 값 감소(전위 감소)
	x-- 연산 후 x 값 감소(후위 감소)

- 예제 : [sec05/SignIncrement](#)

```
plusOne은 1입니다.  
minusOne은 -1입니다.  
x = 1, ++x = 2  
y = 1, y++ = 1  
x = 2, y = 2
```

```
SignIncrementDemo - 메모장  
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)  
public class SignIncrementDemo {  
    public static void main(String[] args) {  
        int plusOne = 1;  
        int minusOne = -plusOne;  
        System.out.println("plusOne은 " + plusOne + "입니다.");  
        System.out.println("minusOne은 " + minusOne + "입니다.");  
  
        int x = 1, y = 1;  
        System.out.println("x = " + x + ", ++x = " + ++x);  
        System.out.println("y = " + y + ", y++ = " + y++);  
        System.out.println("x = " + x + ", y = " + y);  
    }  
}
```

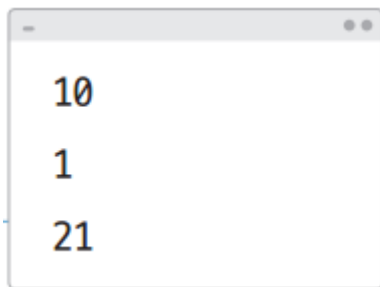
연산자

■ 조건 연산자

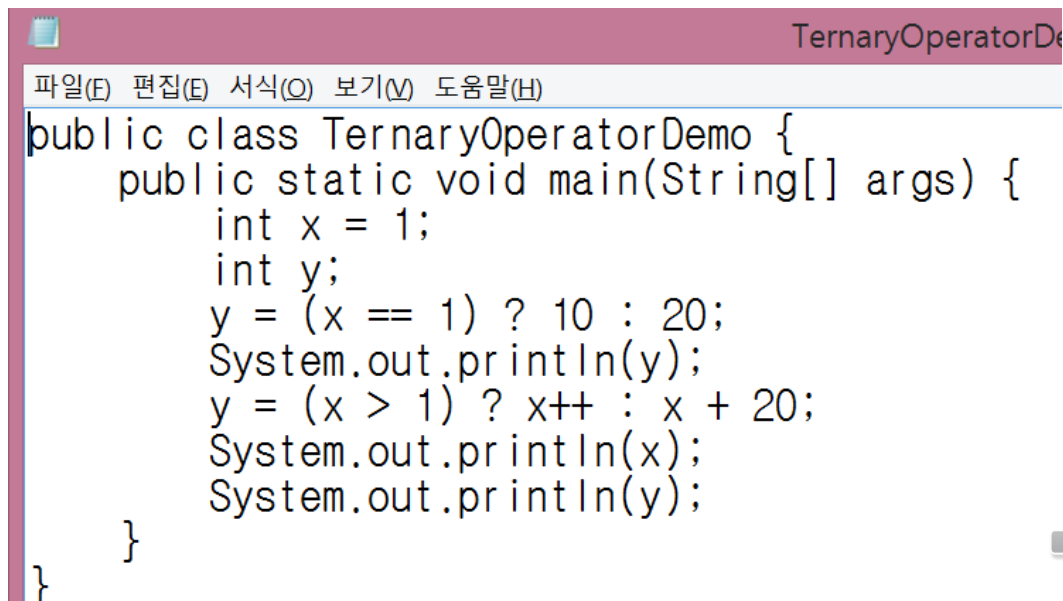
- 조건 연산자(?:)는 조건식이 true이면 결과 값은 연산식1의 값이 되고 false이면 결과 값은 연산식2의 값이 된다.

조건식 ? 연산식1 : 연산식2

- 조건 연산자도 쇼트서킷 로직을 이용하기 때문에 조건식에 따라 연산식1과 연산식2 중 하나만 실행
- 예제 : [sec05/TernaryOperatorDemo](#)



```
10
1
21
```



```
TernaryOperatorDe
파일(E) 편집(E) 서식(O) 보기(V) 도움말(H)
public class TernaryOperatorDemo {
    public static void main(String[] args) {
        int x = 1;
        int y;
        y = (x == 1) ? 10 : 20;
        System.out.println(y);
        y = (x > 1) ? x++ : x + 20;
        System.out.println(x);
        System.out.println(y);
    }
}
```



연산자

■ 우선순위

연산자	설명
[], .. (), ++, --	배열 접근, 객체 접근, 메서드 호출, 후위 증가, 후위 감소
+x, -x, ++x, --x, ~(비트), !(논리)	부호 +/-, 선위 증가, 선위 감소, 비트 부정, 논리 부정
(), new	타입 변환, 객체 생성
*, /, %	곱셈, 나눗셈, 모듈로
+, -	덧셈, 뺄셈
>>, <<, <<<	시프트
>, <, >=, <=, instanceof	비교
==, !=	동등 여부
&	비트 AND
^	비트 XOR
	비트 OR
&&	조건 AND
	조건 OR
?:	조건 연산
=, +=, -=, *=, /=, %=, &=, ^=, !=, <<=, >>=, >>>=	대입

$a + b * c$

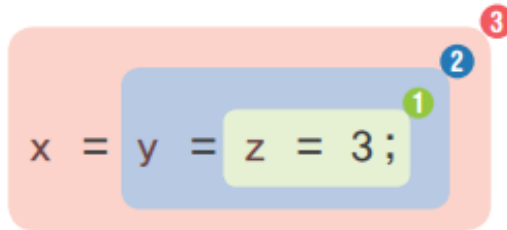
난 나중에

내가 먼저

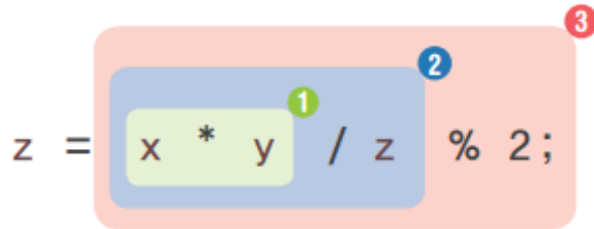


연산자

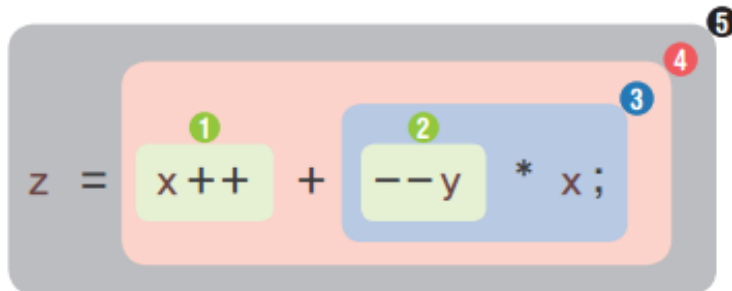
■ 결합 규칙



3을 z, y, x 순(오른쪽에서 왼쪽 순)으로 대입한다.



*, /, % 연산자는 우선순위가 모두 같으므로
왼쪽에서 오른쪽으로 순서대로 연산한다.
3*3/3%2를 연산하면 z에 1을 대입한다.



연산자의 우선순위에 따라 연산하면 ①은 3,
②는 2, ③은 2*4이므로 8, ④는 3+8이므로 11이다.
따라서 z에 11을 대입한다.

- 예제 : [sec05/OperatorPrecedenceDemo](#)

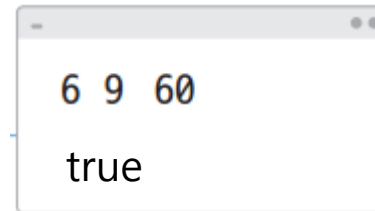
```
6 9 60
true
```



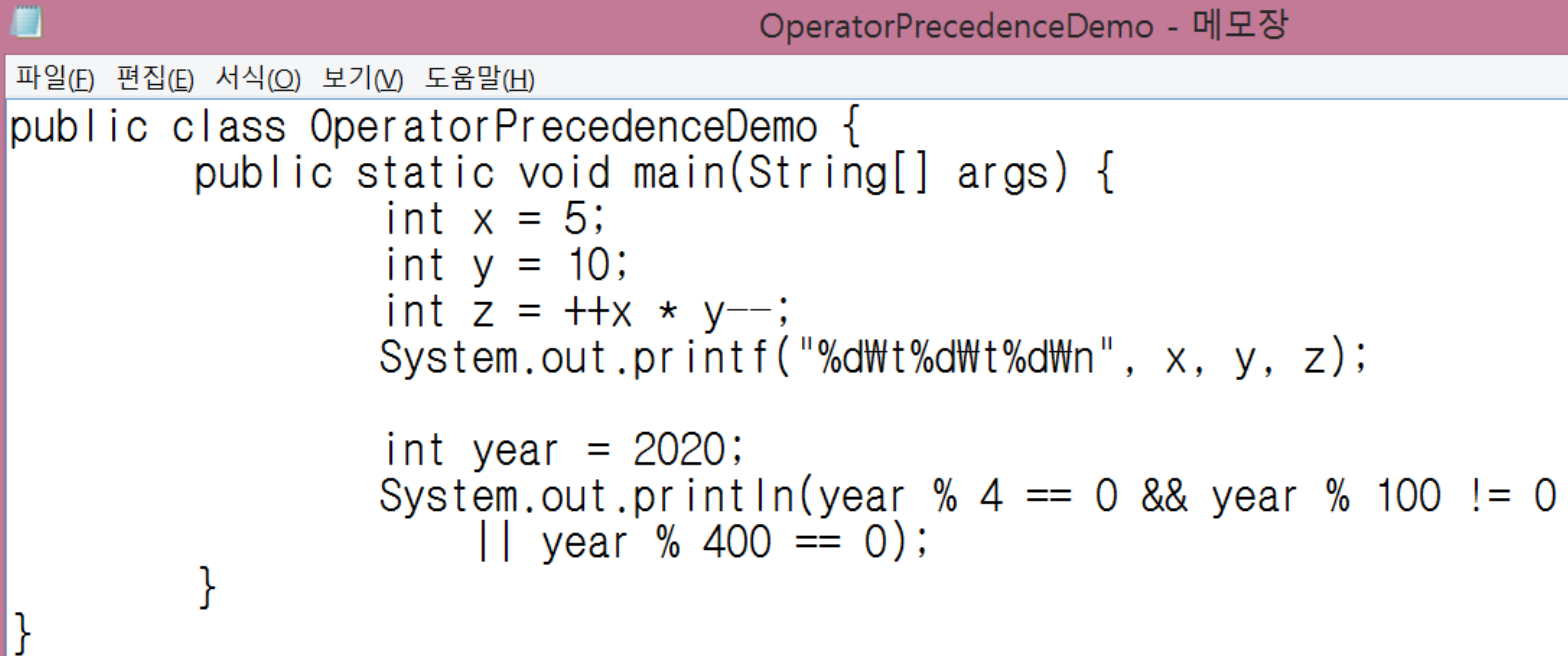
연산자

■ 결합 규칙

- 예제 : [sec05/OperatorPrecedenceDemo](#)



6 9 60
true



```
OperatorPrecedenceDemo - 메모장
파일(E) 편집(E) 서식(O) 보기(V) 도움말(H)
public class OperatorPrecedenceDemo {
    public static void main(String[] args) {
        int x = 5;
        int y = 10;
        int z = ++x * y--;
        System.out.printf("%dWt%dWt%dWn", x, y, z);

        int year = 2020;
        System.out.println(year % 4 == 0 && year % 100 != 0
            || year % 400 == 0);
    }
}
```

