

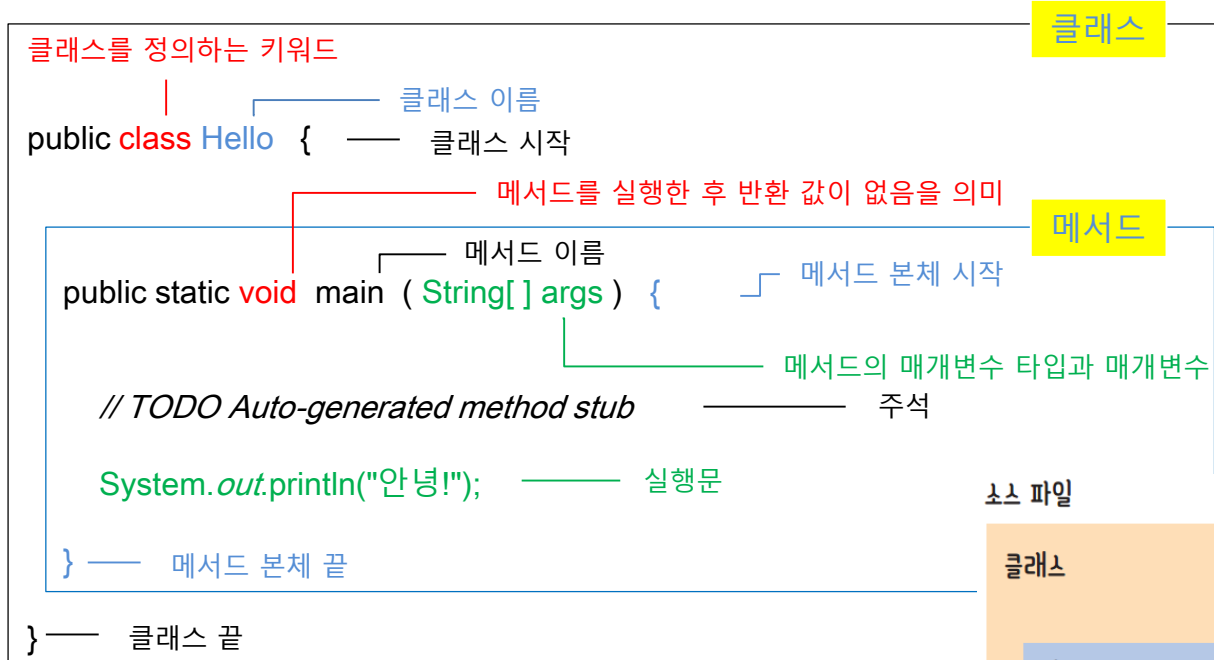
자바 프로그램의 구조와 기본 문법

예제 소스 코드는 파일과 연결되어 있습니다.

editplus(유료), notepad++(무료)와 같은 편집 도구를 미리
설치하여 PPT를 슬라이드 쇼로 진행할 때 소스 파일과
연결하여 보면 강의하실 때 편리합니다.

자바 프로그램 기본 구조

■ Hello 프로그램 구조



자바 프로그램 기본 구조

■ Hello 프로그램 구조

- 클래스 : 객체 지향 언어에서 프로그램을 개발하는 단위
- 메서드 : 수행할 작업을 나열한 코드의 모임
- 실행문 : 작업을 지시하는 변수 선언, 값 저장, 메서드 호출 등의 코드
- 주석문
 - 행 주석 : //
 - 범위 주석 : /* */
 - 문서 주석 : /** */

■ Hello 프로그램의 확장

- [sec01/Hello](#)

식별자

■ 규칙

- 문자, 언더바(_), \$로 시작해야 한다. 한글도 가능하며, 영문자는 대·소문자를 구분한다.
- +, - 등 연산자를 포함하면 안 된다.
- 자바 키워드를 사용하면 안 된다.
- 길이에 제한이 없다.

잘못된 식별자 : %5, a+b, 1b
올바른 식별자 : radius, \$a, _int

■ 자바 키워드

분류	키워드
데이터 타입	byte, char, short, int, long, float, double, boolean
접근 지정자	private, protected, public
제어문	if, else, for, while, do, break, continue, switch, case
클래스와 객체	class, interface, enum, extends, implements, new, this, super, instanceof, null
예외 처리	try, catch, finally, throw, throws
기타	abstract, assert, const, default, false, final, import, native, package, return, static, strictfp, synchronized, transient, true, void, volatile

식별자

■ 관례

- 변수와 메서드는 모두 소문자로 표기. 단, 복합 단어일 때는 두 번째 단어부터 단어의 첫 자만 대문자로 표기
- 클래스와 인터페이스는 첫 자만 대문자로 표기 하고 나머지는 소문자로 표기. 단, 복합 단어일 때는 두 번째 단어부터 단어의 첫 자만 대문자로 표기
- 상수는 전체를 대문자로 표기. 단, 복합 단어일 때는 단어를 언더바(_)로 연결

```
int thisYear;  
String currentPosition;  
boolean isEmpty;  
public int getYear( ) { }
```

```
public class HelloDemo { }  
public interface MyRunnable { }
```

```
final int NUMBER_ONE = 1;  
final double PI = 3.141592;
```

데이터 타입

■ 의미

- 값과 값을 다룰 수 있는 연산의 집합을 의미

■ 종류



데이터 타입

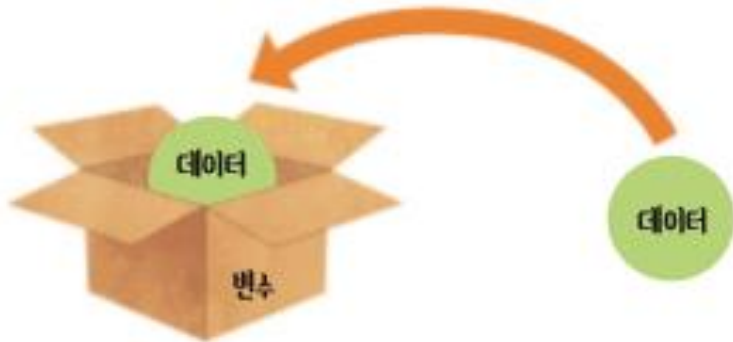
■ 기억 공간 크기 및 기본 값

분류	기초 타입	기억 공간 크기	기본 값	값의 범위
정수	byte	8비트	0	-128 ~ 127
	short	16비트	0	-32,768 ~ 32,767
	int	32비트	0	-2,147,483,648 ~ 2,147,483,647
	long	64비트	0L	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
문자	char	16비트	null	0('����') ~ 65,535('�FFFF')
실수	float	32비트	0.0f	약 $\pm 3.4 \times 10^{-38} \sim \pm 3.4 \times 10^{+38}$
	double	64비트	0.0d	약 $\pm 1.7 \times 10^{-308} \sim \pm 1.7 \times 10^{+308}$
논리	boolean	8비트	false	true와 false

변수

■ 의미

- 프로그램은 기억 공간에 데이터를 보관하고, 각 기억 공간을 변수Variable로 구분
- 변수는 데이터를 담는 상자와 같은 것으로 종류가 다양한데, 이를 구분하려고 데이터 타입을 사용



변수

■ 리터럴

- 프로그램 내부에서 값을 정의해 변수를 초기화할 수 있는데, 그 값을 리터럴

■ 정수

```
int fifteen = 15;           // 10진수
byte fifteen = 0b1111;      // 2진수 15
short fifteen = 017;        // 8진수 15
int fifteen = 0xF;          // 16진수 15
long lightSpeed = 300000L;  // L로 long 타입임을 명시
```

■ 실수

```
double half = 0.5;          // 일반 표기법
double half = 5E-1;          // 지수 표기법으로  $5 \times 10^{-1}$ 을 의미
float pi = 3.14159;          // 오류
float pi = 3.14159F;         // F는 float 타입임을 명시
double pi = 3.14159;
```

가수이다.

지수이다.

변수

■ 예제

- 코드 : [sec03/NumberTypeDemo](#)
- 실행 결과

```
소리가 1시간 동안 가는 거리 : 1224000m  
반지름이 10.0인 원의 넓이 : 314.0
```

변수

■ 문자

```
char c = 'A';           // 문자
char c = 65;            // 일종의 정수 타입이기 때문에 65 대입 가능
char c = '\u0041';      // 유니코드 값으로 대입
char c = "A";           // "A"는 문자가 아니라 문자열이므로 오류
```

■ 논리

```
boolean condition = true; // 논리 리터럴 true와 false 중 하나
```

■ 예제

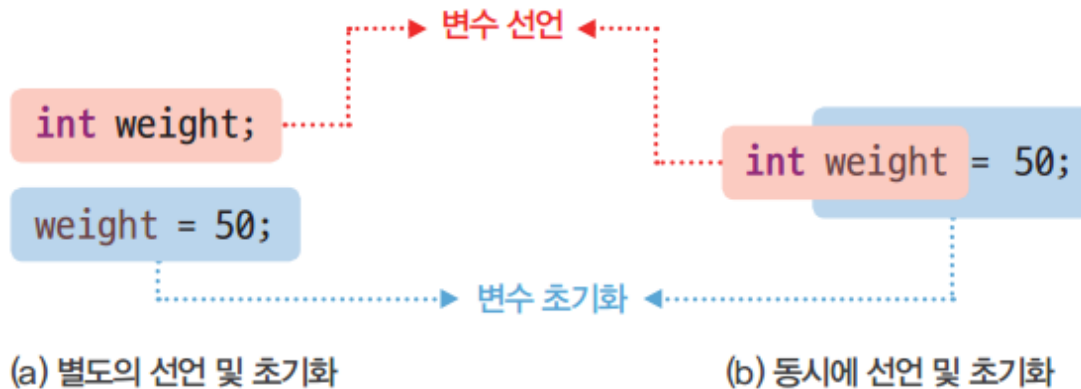
- 코드 [sec03/CharBoolDemo](#)
- 실행 결과

```
-
가
가
true가 아니면 false입니다.
```

변수

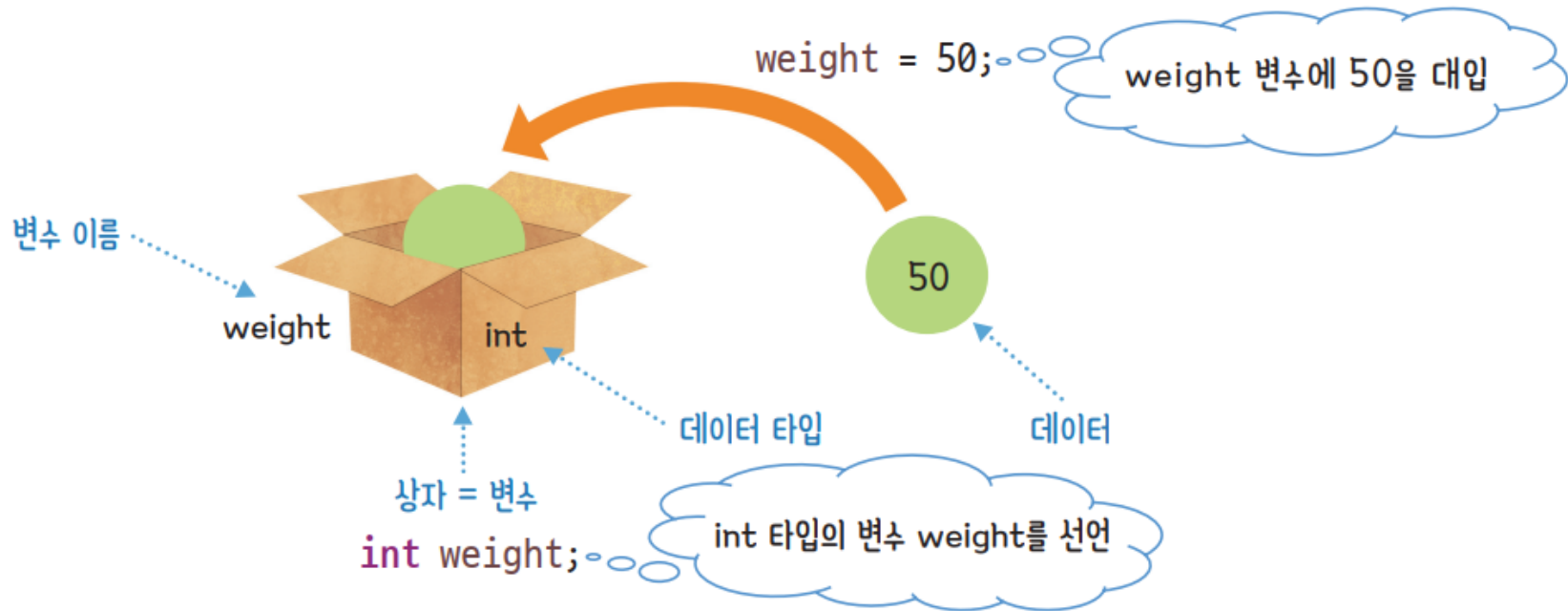
■ 변수 사용

```
int    weight;           // 정수 타입의 weight 변수 선언  
double x, y, z;          // 3개의 변수를 ,로 연결해 선언
```



변수

■ 변수 사용



변수

■ var 예약어

- 자바 10부터 지원
- 초깃값을 통하여 데이터 타입 추론 가능
- 식별자로 사용 가능

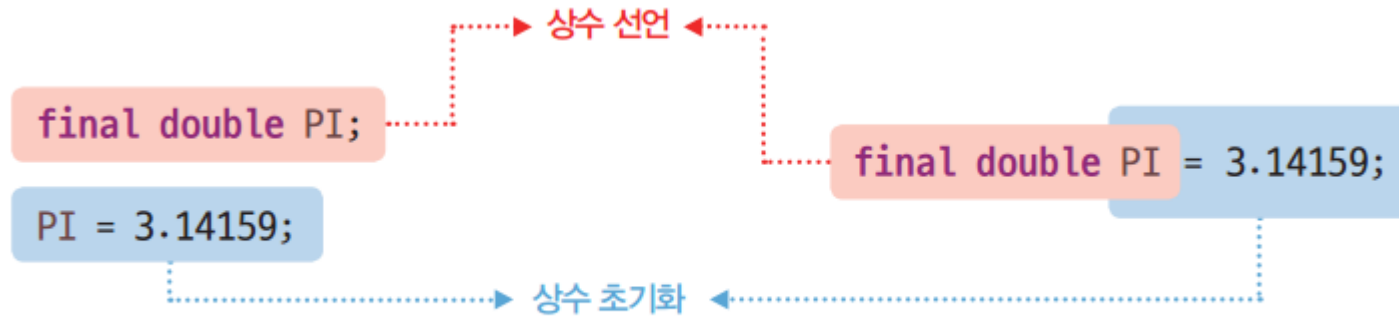
```
var number = 100;    // var은 정수를 나타낼 수 있는 int 타입으로 추론한다.  
var korean = "한국"; // var은 문자열을 나타낼 수 있는 String 타입으로 추론한다.  
var oops;           // 오류
```

- 예제 : [sec03/VarDemo](#)

변수

■ 상수

- 프로그램 실행 중 변경할 수 없는 데이터를 담는 변수
- 예를 들어 원주율 값(3.14159)이나 빛의 속도($3 \times 10^8 \text{m/s}$) 등
- 상수 이름은 변수와 구분하려고 모두 대문자로 표기
- 반드시 final 키워드로 지정



(a) 별도의 선언 및 초기화

(b) 동시에 선언 및 초기화

■ 상수와 리터럴

타입 변환

■ 자동 타입 변환

```
double d1 = 5 * 3.14; // 정수 5를 실수 5.0으로 자동 타입 변환
double d2 = 1;        // 정수 1을 실수 1.0으로 자동 타입 변환
```

■ 강제 타입 변환

```
// double의 3.14를 float로 형 변환해 f에 3.14F 저장
float f = (float)3.14;

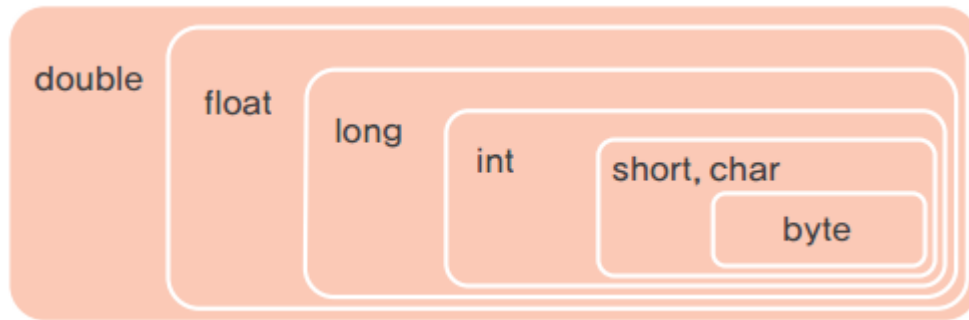
// int의 300을 byte로 형 변환하면 데이터 손실 발생
byte b = (byte)300;

// double의 3.14를 byte로 형 변환하면 데이터가 손실되고 3만 저장
byte x = (byte)3.14;

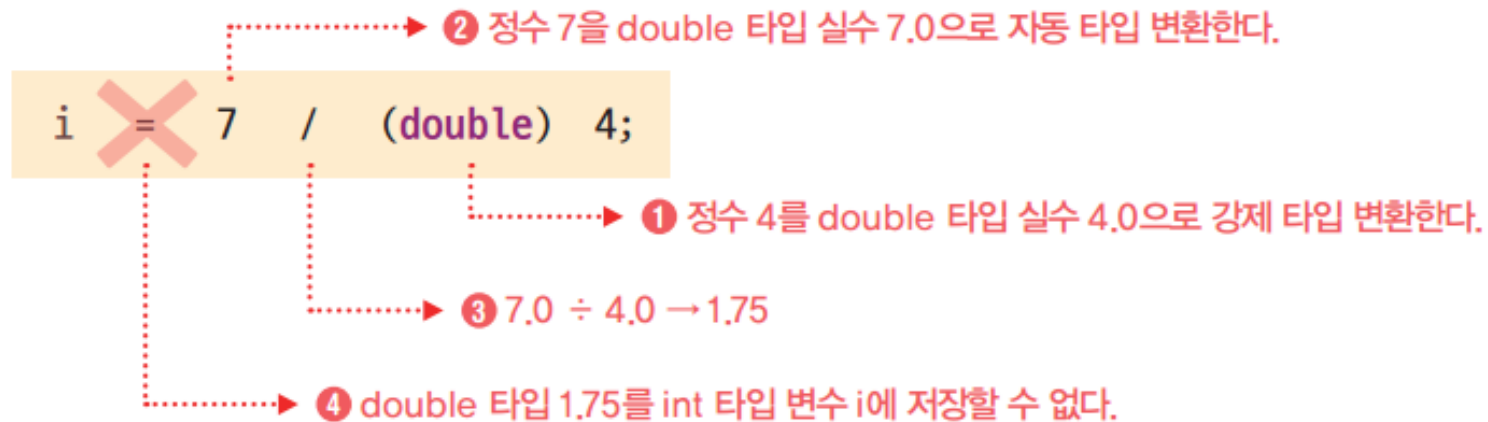
// float의 3.14를 double로 형 변환하면 데이터 손실 없이 저장
double d = (double)3.14f;
```


타입 변환

- 연산 중 필요하면 타입 범위가 넓은 방향으로 자동 타입 변환



- 예



타입 변환

■ 예제

- 코드 : [sec03/CastDemo](#)
- 실행 결과

```
1
```

```
1.0
```

```
1.75
```

```
byte 타입으로 변환할 수 없습니다.
```

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

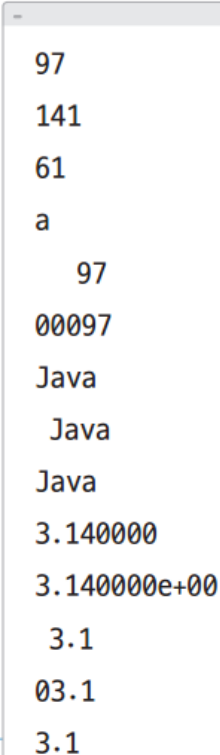
변수 pi를 십진수 실수 포맷과 대응시킨다.

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.
```

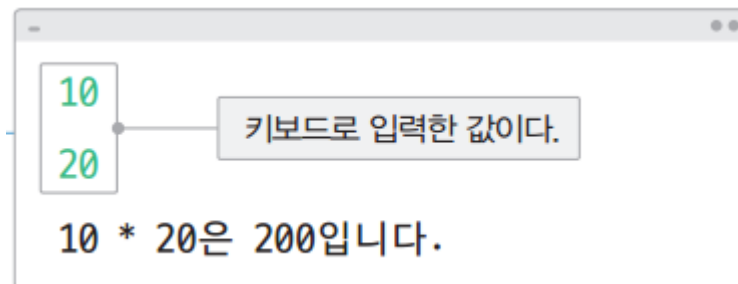
기본 입출력

■ 키보드로 데이터 입력

- Scanner 클래스가 제공하는 데이터 입력 메서드

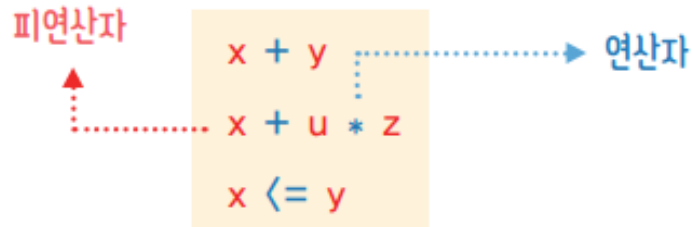
메서드	반환 타입
next()	String
nextByte()	byte
nextShort()	short
nextInt()	int
nextLong()	long
nextFloat()	float
nextDouble()	double
nextLine()	String

- 예제 :
[sec04/ScannerDemo](#)



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

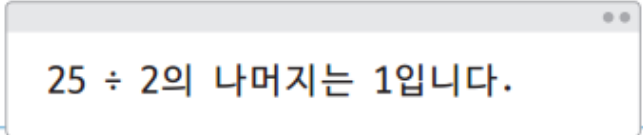
```
// 짝수와 홀수 여부 판단. a가 1이면 n은 홀수, 0이면 짝수
```

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

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

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

- 예제 : [sec05/ArithmeticDemo](#)



25 ÷ 2의 나머지는 1입니다.

연산자

■ 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
```

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
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
00000000110

0b00000110 >> 2

오른쪽으로 2비트씩 이동
왼쪽 빈 2비트 공간을 00으로 채움

00000110
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓
0000011000

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 }
```

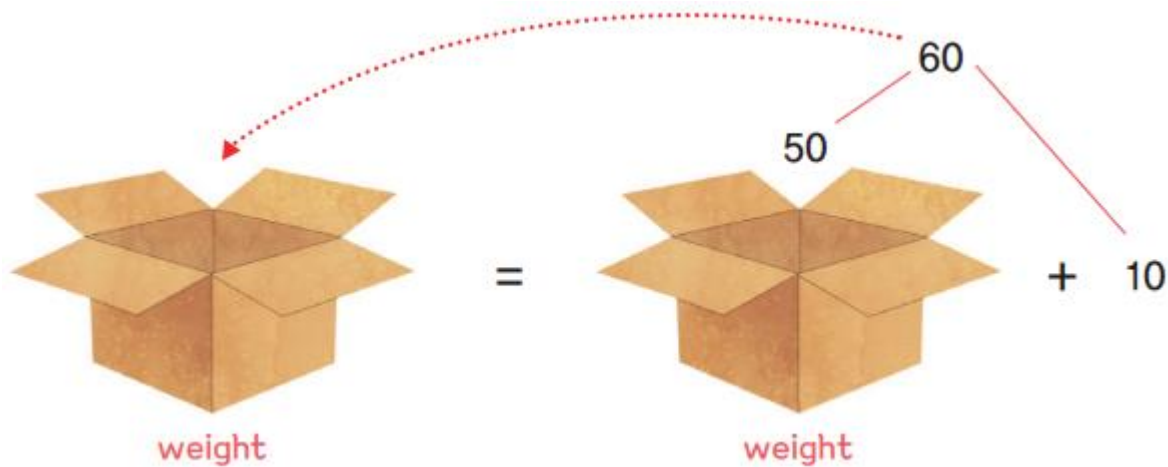
```
1
7
6
fe
1
18
ffffff6 -> -10
ffffffb -> -5
7ffffffb -> 2147483643
```

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

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/SignIncrementDemo](#)

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

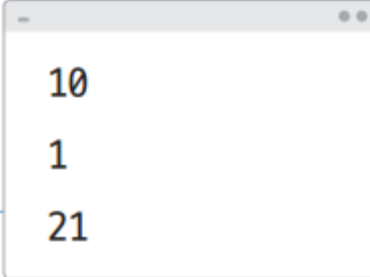
연산자

■ 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](#)

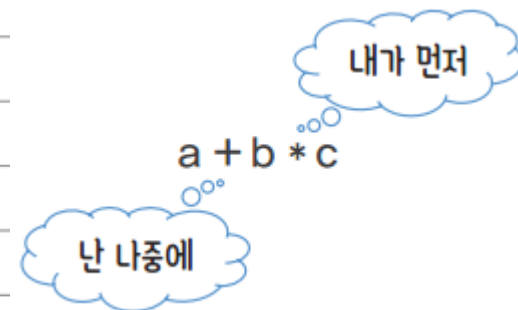


```
10
1
21
```

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



연산자

■ 결합 규칙

`x = y = z = 3;`

3을 z, y, x 순(오른쪽에서 왼쪽 순)으로 대입한다.

`z = x * y / z % 2;`

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

`z = x++ + --y * x;`

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
```