

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

클래스를 정의하는 키워드

```
public class Hello {
```

클래스 이름
클래스 시작

메서드를 실행한 후 반환 값이 없음 의미

```
public static void main (String[] args) {
```

메서드 이름
메서드 본체 시작

```
// TODO Auto-generated method stub
```

메서드의 매개변수 타입과 매개변수
주석

```
System.out.println("안녕!");
```

실행문

```
}
```

메서드 본체 끝

```
}
```

클래스 끝

클래스

메서드

소스 파일

클래스

메서드

실행문

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 표기하고 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
- 기본형(primitive type) - 8개, 실제 값을 저장
- 참조형(reference type) - 기본형을 제외한 나머지, 메모리 주소를 저장(4 byte => 4기가의 메모리를 다룰 수 있음),

■ 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

1비트 - 2진수 1자리

1바이트 - 8비트

데이터 타입

■ 기본형 데이터타입 정리(단위 바이트)

크기	1	2	4	8
정수형	byte	short	int	long
실수형			float	double
문자형		char (Unicode)		
논리형	boolean (false/true)			

byte b = 5 ;

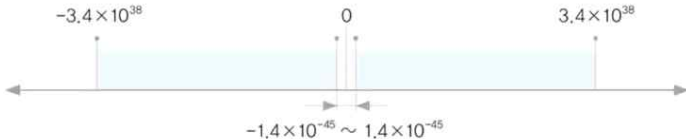
byte 값의 범위 $-2^7 \sim 2^7 - 1$



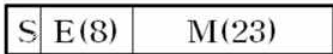
데이터 타입

■ float의 표현범위

자료형	저장 가능한 값의 범위	정밀도	크기	
			bit	byte
float	$1.4\text{E}-45 \sim 3.4\text{E}38$	7 자리	32	4
double	$4.9\text{E}-324 \sim 1.8\text{E}308$	15 자리	64	8



float $1+8+23=32$ bit = 4 byte

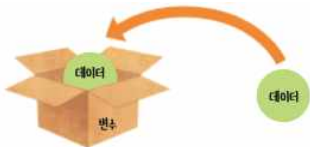


float 타입의 저장 형식 = 부호 + 지수 + 가수

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 => 자바의 변수는 하나의 변수에 다양한 타입의 값을 저장할 수가 없음



정보

PC가 모니터링되고 보호됩니다.

자세한 내용은 Windows 보안을 참조하세요.

장치 사양

장치 이름 DESKTOP-AHDSK3K
프로세서 Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz
설치된 RAM 16.0GB(15.0GB 사용 가능)
장치 ID FC9FA90F-3AEB-80F9-BE7B02969ED6
제품 ID 00325-96574-23195-AAOEM
시스템 종류 64비트 운영 체제, x64 기반 프로세서
변 및 타지 이 디스클레이에 사용할 수 있는 변 또는 타지식 일력 이 있습니다.

변수

■ 리터럴

- 프로그램 내부에서 값을 정의해 변수를 초기화할 수 있는데, 그 값을 리터럴
- 리터럴에 붙이는 접두사와 접미사

■ 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 타입
char c = '\u0041';      // 유니코드 값으로
char c = "A";           // "A"는 문자가 아
```

문자	코드	문자	코드	문자	코드
0	48	A	65	a	97
1	49	B	66	b	98
2	50	C	67	c	99
3	51	D	68	d	100
4	52	E	69	e	101
5	53
6	54	W	87	w	119
7	55	X	88	x	120
8	56	Y	89	y	121
9	57	Z	90	z	122

■ 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
```

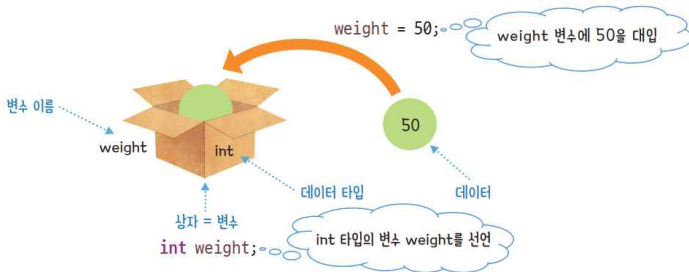


(a) 별도의 선언 및 초기화

(b) 동시에 선언 및 초기화

변수

■ 변수 사용



변수

■ var 예약어

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

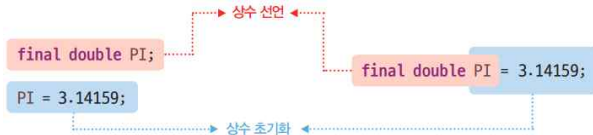
```
var number = 100;    // var은 정수를 나타낼 수 있는 int 타입으로 추론한다.  
var korean = "한국"; // var은 문자열을 나타낼 수 있는 String 타입으로 추론한다.  
var oops;           // 오류
```

- 예제 : [sec03/VarDemo](#)

변수

■ 상수(constant) : 한번만 값을 저장 가능한 변수

- 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

- 리터럴 : 그 자체로 값을 의미하는 것
- 상수와 리터럴은 기존의 상수와 같다 즉 같은 개념임
- 자바에서 상수를 한번만 값을 저장할 수 있는 변수라고 했기 때문에 구분하는 것임
- `int score = 100;` 여기서 100이 리터럴, `final int MAX = 200;` 200은 리터럴, MAX는 상수

변수

■ 변수와 리터럴의 타입 불일치

- 범위가 변수 > 리터럴 인 경우 **OK**

```
int i = 'A';           // int > char
long l = 123;          // long > int
double d = 3.14f ;     // double > float
```

- 범위가 변수 < 리터럴 인 경우 **Error**

```
int i = 10_000_000_000 // int 범위 20억 넘기 때문임
long l = 3.14f;        // long < float , 8byte > 4 byte 이지만 실수형이 저장범위가 더 넓음
float f = 3.14 ;       // float < double , 점미사 f 가 생략되면 double 임
```

- byte, short 변수에 int 리터럴 저장하는 경우 **OK**

byte b = 100; // byte 의 범위에 속하기 때문에 OK, 컴파일러가 (byte)100으로 자동형변환해줌.

byte b = 128 // -128 ~ 127 범위에 벗어나기 때문에 **Error**

```
int i = 100;
```

byte b = i ; // **Error**, 반드시 (byte)i 로 형변환을 해서 대입해야 함

byte b = 128 // -128 ~ 127 범위에 벗어나기 때문에 **Error**

타입 변환(형변환)

- 형변환 : 변수 또는 상수의 타입을 다른 타입으로 변환하는 것

- 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 : (타입)피연산자

```
// 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(형변환)

■ 자동 타입 변환

1. byte -> int

```
byte b = 10;
```

```
int i = b; // 생략가능
```

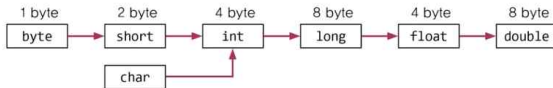
2. int -> byte

```
int i2 = 300;
```

```
byte b2 = (byte)i2; // 생략불가
```

[illegible]

- 표현범위가 좁은타입에서 넓은 타입으로 형변환하는 경우 값 손실이 없으므로, 두 타입 중에서 표현범위가 더 넓은 쪽으로 형변환



타입 변환(형변환)

■ 문자와 숫자간의 변환

- `3 + '0' => '3'`
- `'3' - '0' => 3`

ASCII 코드

문자

48

0

51

3

54

6

57

9

■ 문자열로의 변환

- `3 + "" => "3"`
- `'3' + "" => "3"`

■ 문자열을 숫자로 변환

- `"3" => 3`, `Integer.parseInt("3")` 의 결과가 3으로 변환됨.
- `"3.1" => 3.1`, `Double.parseDouble("3.1")` 의 결과가 double 리터럴 3.1 으로 변환됨.

■ 문자열을 문자로 변환

- `"3" => '3'`, `"3".charAt(0)` 의 결과가 '3' 으로 변환됨
- `"안녕하세요".charAt(1)` 의 결과는 ?

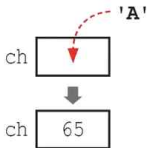
타입 변환(형변환)

■ 형변환 연산자

변 환	수 식	결 과
int → char	(char)65	'A'
char → int	(int)'A'	65
float → int	(int)1.6f	1
int → float	(float)10	10.0f

■ 유니코드 문자표

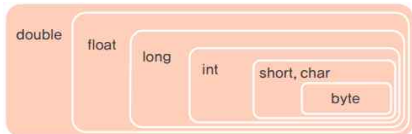
char ch = 'A' ;



문자	코드	문자	코드	문자	코드
0	48	A	65	a	97
1	49	B	66	b	98
2	50	C	67	c	99
3	51	D	68	d	100
4	52	E	69	e	101
5	53
6	54	W	87	w	119
7	55	X	88	x	120
8	56	Y	89	y	121
9	57	Z	90	z	122

타입 변환(형변환)

- 연산 중 필요하면 타입 범위가 넓은 방향으로 자동 타입 변환



- 예

`i = 7 / (double) 4;`

❌

- 정수 4를 double 타입 실수 4.0으로 강제 타입 변환한다.
- 정수 7을 double 타입 실수 7.0으로 자동 타입 변환한다.
- $7.0 \div 4.0 \rightarrow 1.75$
- double 타입 1.75를 int 타입 변수 i에 저장할 수 없다.

문자와 문자열

- `char c = 'A';`
- `char c = "A" ; // Error`
- `String s = "ABC";`
- `String`은 클래스이기 때문에 `String s = new String("ABC");` 와 같이 적어야 하나
워낙 자주 쓰여서 `String` class는 `String s = "ABC"` 와 같이 사용가능
: 참조변수 `s` 에 문자열의 주소가 저장되지만 변수 `s` 에 "ABC"가 저장된다고
생각해도 됨
- `char c = " ; // Error`
- `String s = "" ; // 빈 문자열`
- `String s = "A" + "B" ; // 문자열의 결합`
- `String s = "" + 7 ; ==> "" + "7" => 문자열의 결합 => "7"`
 - `"" + 7 + 7` 과 `7 + 7 + ""` 의 결과는 다름

타입 변환

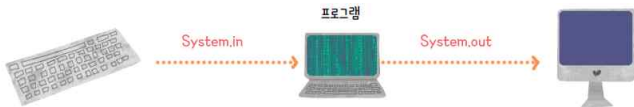
■ 예제

- 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("%dWn", i);
09      System.out.printf("%oWn", i);
10      System.out.printf("%xWn", i);
11      System.out.printf("%cWn", i);
12      System.out.printf("%5dWn", i);
13      System.out.printf("%05dWn", i);
14      System.out.printf("%sWn", s);
15      System.out.printf("%5sWn", s);
16      System.out.printf("%-5sWn", s);
17      System.out.printf("%fWn", f);
18      System.out.printf("%eWn", f);
19      System.out.printf("%4.1fWn", f);
20      System.out.printf("%04.1fWn", f);
21      System.out.printf("%-4.1fWn", 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; // 화면으로부터 데이터를 입력받는 기능을 제공하는 클래스
```

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

```
Scanner in = new Scanner(System.in); // System.in 은 화면입력을 의미
```

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

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

```
    * in.nextFloat() ; // 실수를 읽는다.
```

```
String input = in.nextLine(); // 화면에서 입력받은 내용을 input에 대입한다.
```

```
int num = Integer.parseInt(input); // 문자열(input)을 숫자(num)로 변환
```

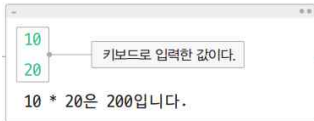
기본 입출력

■ 키보드로 데이터 입력

- Scanner 클래스가 제공하는 데이터 입력 메서드

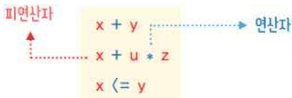
메서드	반환 타입
next()	String
nextByte()	byte
nextShort()	short
nextInt()	int
nextLong()	long
nextFloat()	float
nextDouble()	double
nextLine()	String

- 예제 : [sec04/ScannerDemo](#)



연산자

- 연산자: 연산을 수행하는 기호 (+, -, *, /), 피연산자 : 연산자의 연산 수행 대상
- 모든 연산자는 연산 결과를 반환한다. : () 는 연산자가 아니다.
- 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 다른면 큰 범위의 타입으로 일치시킨 후 연산 수행
- 산술변환 : 연산전에 피연산자의 타입을 일치시키는 것
 - 1) 두 연산자의 타입을 같게 일치시킨다.(값손실 최소화하기 위해 보다 큰 타입으로 일치)

```
long   + int   → long   + long   → long
float  + int   → float  + float  → float
double + float → double + double → double
```

- 2) 피연산자의 타입이 int보다 작은 타입이면 int로 변환된다.

```
byte + short → int + int → int
char + short → int + int → int
```

```
예> int  int  int
    10 / 4 = 2  // 소수점 이하는 버려짐
```

```
int  float  float  float  float
10 / 4.0f -> 10.0f / 4.0f -> 2.5f  // 값 손실을 막기 위해 float로 변환후 연산
```

연산자

■ 산술 연산자

- 산술변환 : 연산전에 피연산자의 타입을 일치시키는 것
 - 1) 두 연산자의 타입을 같게 일치시킨다.(값손실 최소화하기 위해 보다 큰 타입으로 일치)
 - 2) 피연산자의 타입이 int보다 작은 타입이면 int로 변환된다.

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

예> char 끼리의 연산은 int - int 로 바꾼 후 계산

$$'2' - '0' = 50 - 48 = 2$$

문자 '2'에서 문자 '0'을 빼주어 숫자 2로 변환되었다.

연산자

■ 산술 연산자

- 산술변환 : 연산전에 피연산자의 타입을 일치시키는 것
 - 1) 두 연산자의 타입을 같게 일치시킨다.(값손실 최소화하기 위해 보다 큰 타입으로 일치)
 - 2) 피연산자의 타입이 int보다 작은 타입이면 int로 변환된다.

```
예> int a = 1_000_000; // 백만  
    int b = 2_000_000; // 2백만
```

```
    long c = a * b ; // 두 정수의 곱인 c 값은?
```

연산자

■ 산술 연산자

- 나눗셈 또는 나머지 연산에서 좌측 연산자가 정수이고 우측 피연산자가 0일 경우 예외가 발생한다. 무한대의 값을 정수로 표현할 수 없기 때문임
 - NaN : Not a Number
 - infinity : 무한대

예> `int result = 5/0 ;` // ArithmeticException 발생

`5 / 0.0 --> infinity`

`infinity + 2 --> infinity`

`5 % 0.0 --> NaN`

`NaN + 2 --> NaN`

`boolean res = Double.isInfinity(변수) ;`

`boolean res = Double.isNaN(변수) ;`

나눗셈 연산의 결과가 infinity나 NaN인지 먼저 확인하고 다음 연산을 수행해야 한다.

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.

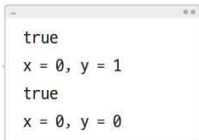
조건식1 || 조건식2

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

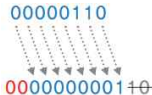
연산자

■ 비트·시프트 연산자

● 시프트 연산자의 종류

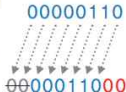
연산자	a 연산자 b일 경우 설명(예를 들어, a << b)
<<	a의 모든 비트를 왼쪽으로 b비트만큼 이동하며, 이동할 때마다 최하위 비트를 0으로 채운다. 곱셈 효과가 나타나기 때문에 산술적 왼쪽 시프트(Arithmetic Left Shift)라고 한다.
>>	a의 모든 비트를 오른쪽으로 b비트만큼 이동하며, 이동할 때마다 최상위 비트와 동일한 비트로 채운다. 나눗셈 효과가 나타나기 때문에 산술적 오른쪽 시프트(Arithmetic Right Shift)라고 한다.
>>>	a의 모든 비트를 오른쪽으로 b비트만큼 이동하며, 이동할 때마다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

● 예



0b00000110 >> 2

오른쪽으로 2비트씩 이동
왼쪽 빈 2비트 공간을 00으로 채움



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("%xWn", 0b0101 & 0b0011);
06         System.out.printf("%xWn", 0b0101 | 0b0011);
07         System.out.printf("%xWn", 0b0101 ^ 0b0011);
08         System.out.printf("%xWn", (byte) ~0b00000001);
09         System.out.printf("%xWn", 0b0110 >> 2);
10         System.out.printf("%xWn", 0b0110 << 2);
11
12         int i1 = -10;
13         int i2 = i1 >> 1;
14         int i3 = i1 >>> 1;
15         System.out.printf("%x -> %dWn", i1, i1);
16         System.out.printf("%x -> %dWn", i2, i2);
17         System.out.printf("%x -> %dWn", i3, i3);
18     }
19 }
```

1

7

6

fe

1

18

ffffff6 -> -10

ffffffb -> -5

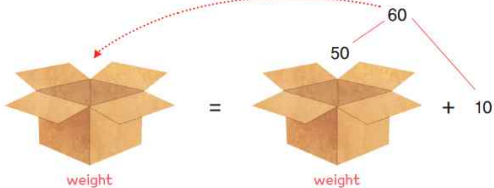
7ffffffb -> 2147483643

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
 - 전위형(값이 참조되기 전에 증감)과 후위형(값이 참조된 후에 증감)
- 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)

- 독립적으로 사용된 경우에는 전위형과 후위형의 차이가 없다.

```
++i;  
i++;
```

연산자

■ 증감연산자

- 예제 : [sec05/SignIncrementDemo](#)

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

- 증감연산자가 도입된 이유는 코드, 연산식을 간단히 하기 위해서였으나 이해하기 어려울 때는 증감연산자를 따로 떼어내면 이해하기가 쉬워진다. 각각 전위형과 후위형을 아래와 같이 떼어내서 보면 된다.

j= ++i; (전위형) ++i;
j=i;

J=i++; (후위형) j=i;
i++;

연산자

■ 조건 연산자

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

조건식 ? 연산식1 : 연산식2

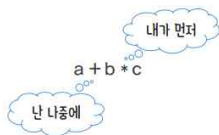
- 조건 연산자도 쇼트서킷 로직을 이용하기 때문에 조건식에 따라 연산식1과 연산식2 중 하나만 실행
- 예제 : [sec05/TernaryOperatorDemo](#)



연산자

- 우선순위 : 하나의 식(expression)에 연산자가 둘 이상 있을 때 어떤 연산을 먼저 수행할지를 자동 결정하는 것

연산자	설명
[] , () , ++ , --	배열 접근, 객체 접근, 메서드 호출, 후위 증가, 후위 감소
+x, -x, ++x, --x, ~(비트), !(논리)	부호 +/-, 순위 증가, 순위 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

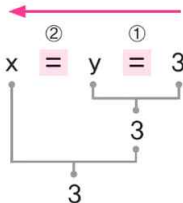
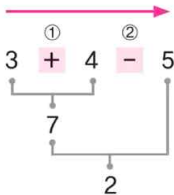
- **우선순위** : 하나의 식(expression)에 연산자가 둘 이상 있을 때 어떤 연산을 먼저 수행할지를 **자동 결정하는 것**

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 ' $x > 3$ '과 ' $x < 5$ '가 먼저 계산된 다음에 '&&'가 수행된다. 식의 의미는 ' x 가 3보다 크고 5보다 작다'이다.
$result = x + y * 3;$	대입 연산자는 연산자 중에서 제일 우선순위가 낮다. 그래서 우변의 최종 연산결과가 변수 $result$ 에 저장된다.

표 > 자바의 정석

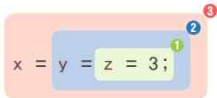
연산자

- 결합 규칙 : 우선순위가 같은 연산자가 있을 때, 어떤 것을 먼저 적용할지를 정하는 것
- 기본적으로는 대부분 왼쪽에서 오른쪽이고, 단항과 대입연산자만 오른쪽에서 왼쪽임



연산자

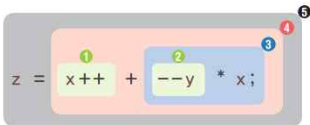
- 결합 규칙 : 우선순위가 같은 연산자가 있을 때, 어떤 것을 먼저 적용할지를 정하는 것



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

■ 연산자의 우선순위

- 산술 > 비교 > 논리 > 대입 의 순서, 대입이 제일 마지막에 수행된다.
- 단항(1) > 이항(2) > 삼항(3) , 단항 연산자의 우선순위가 이항 연산자보다 높다.

■ 결합규칙

- 단항연산자와 대입연산자를 제외한 모든 연산의 진행방향은 왼쪽에서 오른쪽이다.