

알 기 쉽 게 해 설 한

J A V A

9th edition

# 3장 변수, 자료형, 연산자

Section 1 식별자와 예약어

Section 2 변수

Section 3 자료형

Section 4 연산과 형 변환

Section 5 연산자와 수식

Section 6 연산자의 우선순위

Section 7 문자열 String





## ■ 학습목표

9<sup>th</sup> edition

- 식별자와 예약어를 알아보고 사용 관례에 관해 학습합니다.
- 변수의 의미와 변수명을 지정하는 방법에 관해 학습합니다.
- 자료형의 종류와 사용 방법을 알아봅니다.
- 연산의 의미와 서로 다른 자료형 간의 형 변환에 관해 학습합니다.
- 다양한 연산자들의 사용 방법에 관해 알아봅니다.
- 수식에서 사용된 연산자의 우선순위에 관해 학습합니다.
- 문자열과의 사용 방법에 관해 학습합니다.



### ● 자바에서 식별자

- 변수, 상수, 메소드, 배열, 문자열, 사용자가 정의하는 클래스나 메소드 등을 구분할 수 있는 이름을 의미

### ● 식별자의 사용 원칙

- 식별자는 문자, 숫자, 특수문자( \_, \$ )로 구성될 수 있다.
- 식별자의 첫 문자는 문자나 특수문자로 시작할 수 있다. 숫자는 사용할 수 없다.
- 예약어를 식별자로 사용할 수 없다.
- true, false, null은 식별자로 사용할 수 없다.
- 식별자는 길이에 제한을 두지 않는다.
- 같은 문자의 대소문자(Sum과 sum)는 서로 다른 식별자로 취급한다.



## ● 자바의 예약어

abstract	const	finally	interface	short	transient
assert	continue	float	long	static	try
boolean	default	for	native	strictfp*	void
break	do	goto	new	super	volatile
byte	double	if	package	switch	while
case	else	implements	private	synchronized	
catch	enum	import	protected	this	
char	extend	instanceof	public	throw	
class	final	int	return	throws	



### ● 실습예제 3.1 : 예약어를 변수로 사용?

예제 3.1 ReservedWordTest.java

```
01: public class ReservedWordTest {  
02:     public static void main(String[] args) {  
03:         int if = 20; ← 예약어를 변수로 사용  
04:         double for = 30.0; ←  
05:     }  
06: }
```

실행 결과

```
Exception in thread "main" java.lang.Error: Unresolved compilation problems:  
  Syntax error on token "if", invalid VariableDeclaratorId  
  Syntax error on token "for", invalid VariableDeclaratorId  
  Duplicate local variable $missing$  
  
at ReservedWordTest.main(ReservedWordTest.java:3)
```



### ● 자바에서 식별자의 사용은 강제적이지는 않지만 일반적인 관례가 있다

클래스 이름

JavaTest1, RuntimeErrorTest ← 바람직한 형태. 단어의 첫 글자는 대문자로 쓰는 것이 좋습니다.

applicationtest, sampletest ← 오류는 아니지만 관례에 어긋나고, 가독성이 떨어집니다.

메소드, 변수, 배열, 문자열의 이름

sum, sumAndSubstract, nameAddress ← 바람직한 형태. 단어의 첫글자는 소문자로 쓰는 것이 좋습니다.

NameAndAge, Productname ← 좋지 않은 형태. 클래스 이름과 혼동됩니다.

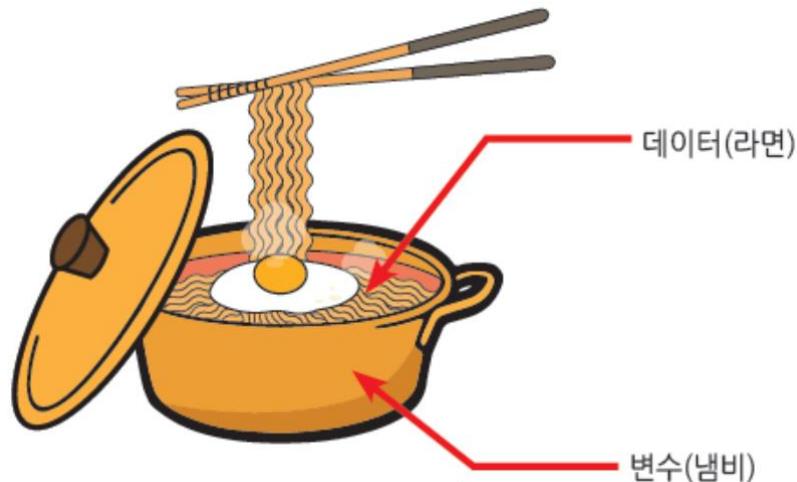
상수의 이름

PI, MAX\_NUMBER ← 상수는 모두 대문자로 쓰는 것이 관례입니다.

max, Max, Address ← 좋지 않은 형태. 다른 이름과 혼동됩니다.

## 2-1 변수의 의미

### ● 일상 생활에서의 변수는?

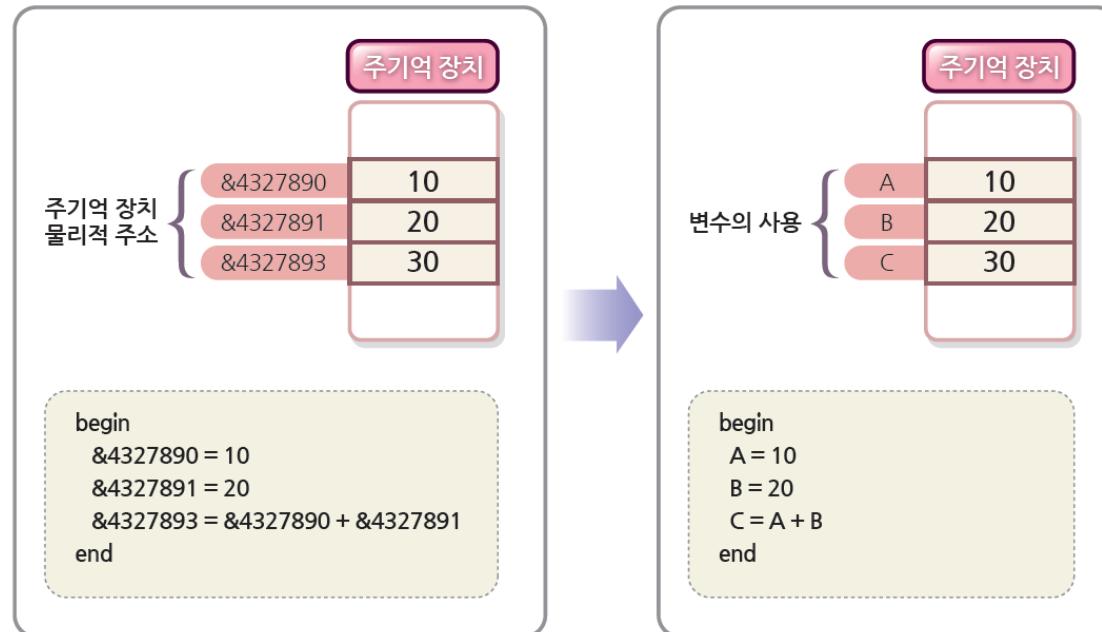


- 우리가 먹을 수 있는 것은? 라면인건가? 냄비인건가?
- 프로그램에서 처리하는 것은 라면, 라면을 저장하는 냄비가 바로 변수

## 2-1 변수의 의미

### ● 프로그래밍 언어에서 사용되는 변수

- “값(value)이 저장된 메모리의 위치에 주어진 이름”
- 변수에 값을 배정(assignment)할 때 “=”기호를 사용





### ● 변수명을 지정하는 규칙

- 변수명의 첫 글자는 반드시 영문자나 일부 특수 문자(\_,\$)로 시작한다.
- 변수명에는 숫자가 포함될 수 있다.
- 변수명에는 공백이 포함될 수 없다.
- 대소문자를 구분한다. 즉 Sum과 sum은 다른 변수명이다.
- 예약어를 변수명으로 사용할 수 없다.

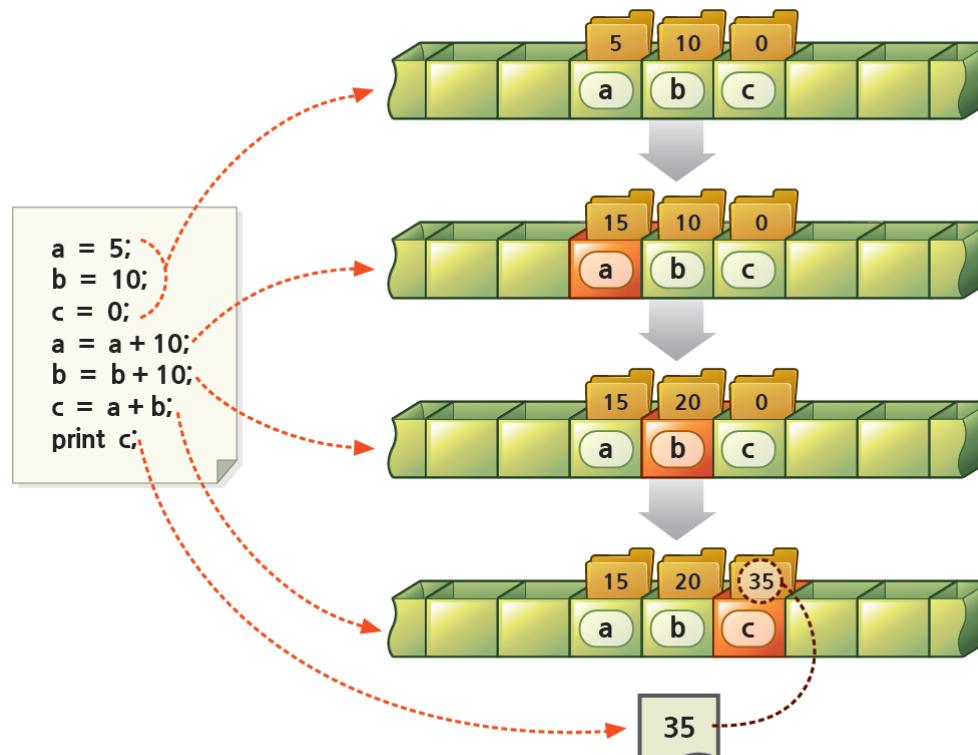
### ● 변수의 사전적 의미

- 변화하는 것

### ● 좋은 변수명?

- 의미를 가진 변수명

- 메모리에 저장된 변수의 값 변화

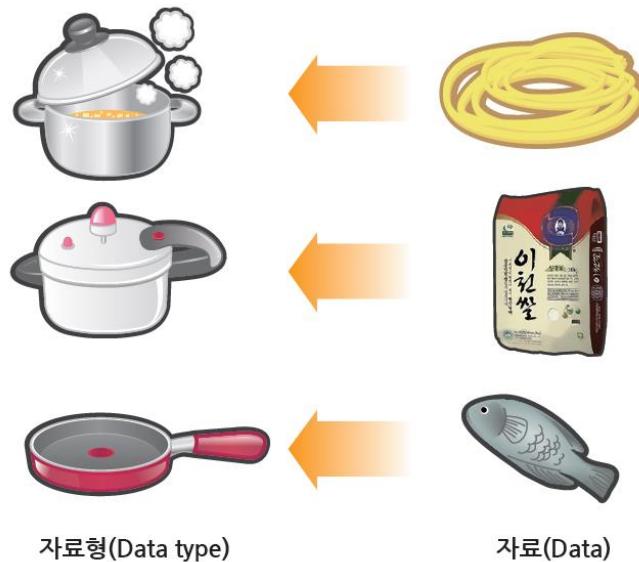


메모리에 저장된 변수값은 프로그램의 진행에 따라 변화됩니다.

## ● 자료형(Data Type)

- 변수가 가질 수 있는 값의 형태

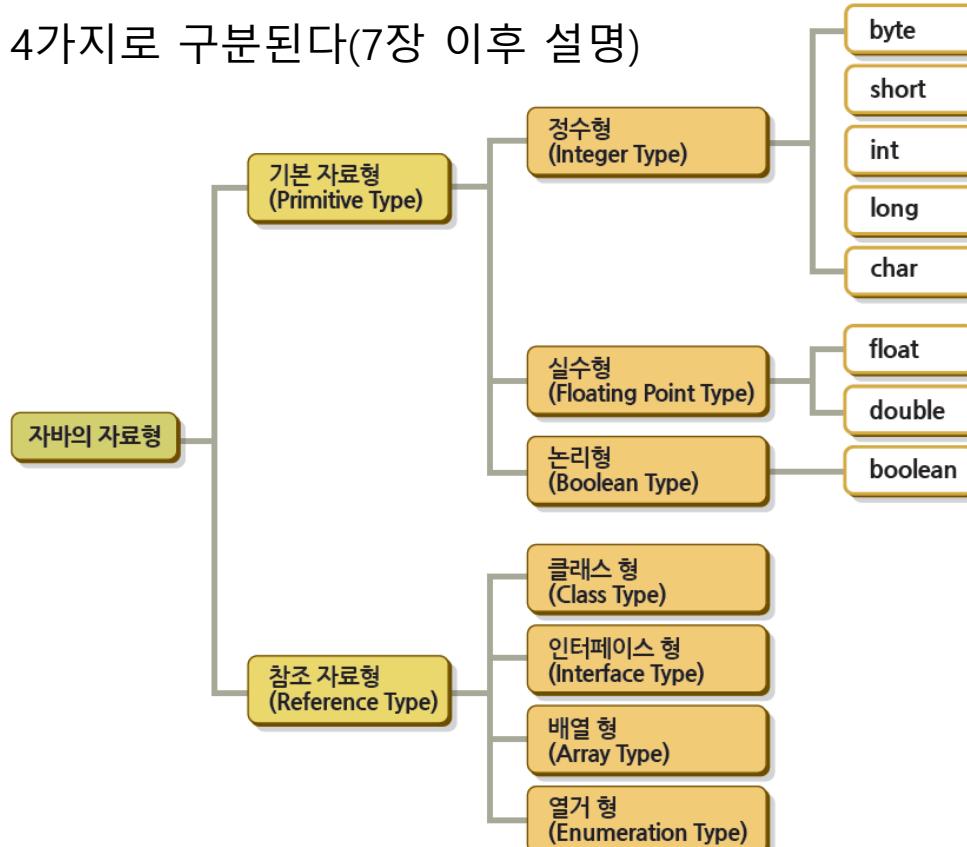
## ● 일상 생활에서의 자료형? 그릇마다 담을 수 있는 자료(데이터)가 다르다



## 3-1 기본 자료형과 참조 자료형

### ● 자바의 자료형은 크게

- 기본 자료형 8가지와
- 참조 자료형 4가지로 구분된다(7장 이후 설명)



## 3-1 기본 자료형과 참조 자료형

### ● 기본 자료형과 참조 자료형의 차이

- 기본 자료형 : 값을 가진다
- 참조 자료형 : 참조(주소)를 가진다

`int a = 10;`

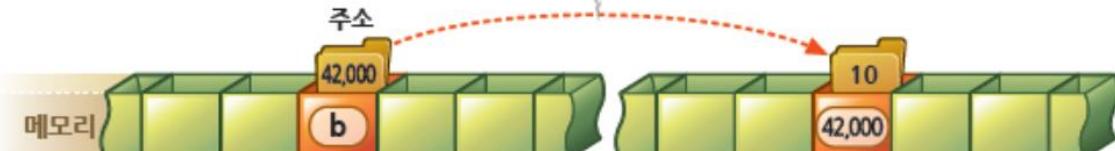
기본 자료형으로서 변수로 지정된 위치에 값이 저장되어 있습니다.



한 번의 접근으로 값을 가져올 수 있습니다.

`Integer b = new Integer(10);`

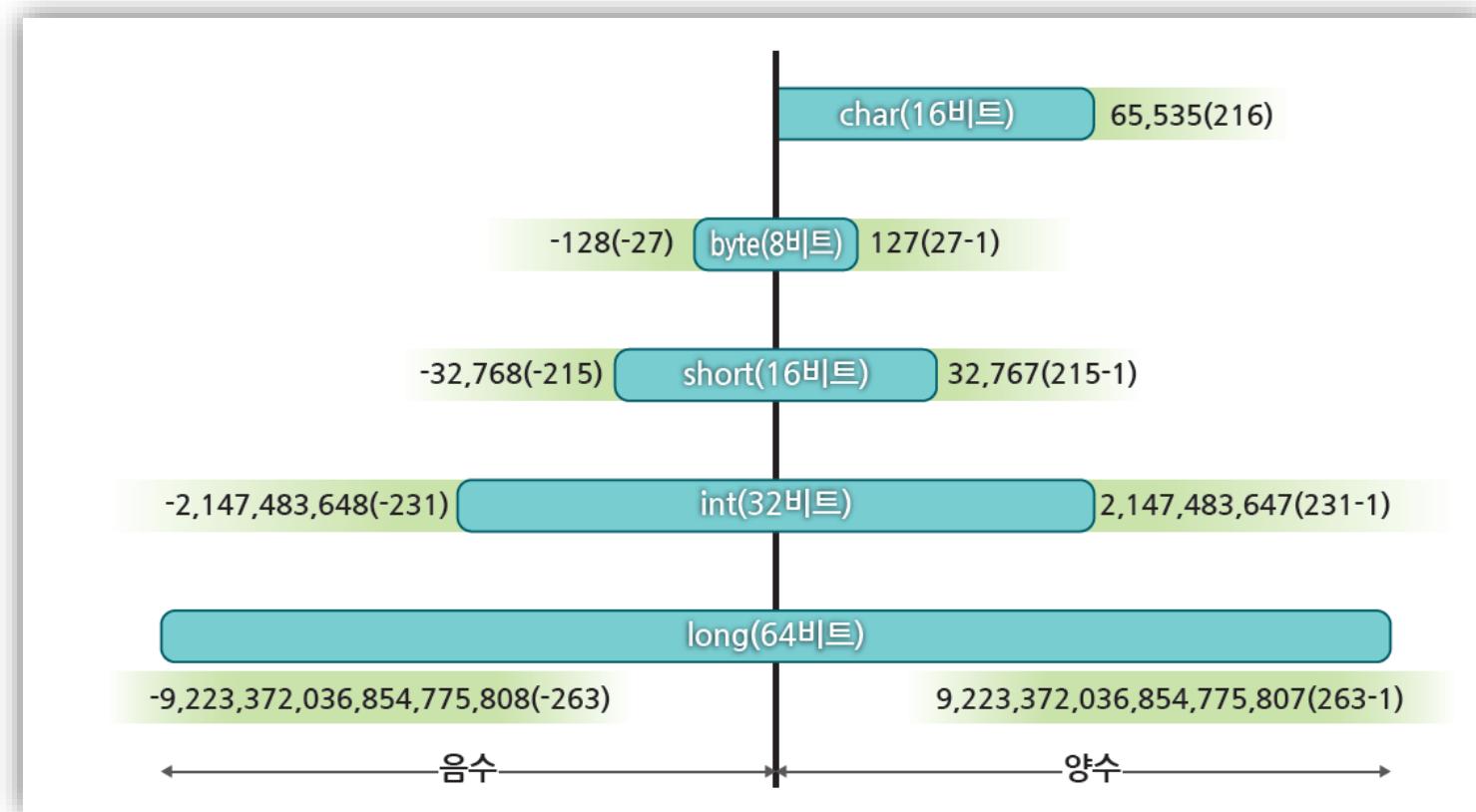
참조 자료형으로서 변수로 지정된 위치에는 실제 값이 있는 위치의 주소가 저장됩니다.



두 번의 접근으로 값을 가져올 수 있으므로 기본 자료형에 비해 효율성이 떨어질 수 있습니다.

## 3-2 정수형

- 자바의 정수형 : 5가지(byte, short, int, long, char) 제공



## ● 수치 정수형 : 4가지(byte, short, int, long) 예제 3.2

- 자바에서 정수는 묵시적으로 int로 취급

예제 3.2 NumericTypeTest.java

```
01: public class NumericTypeTest {  
02:     public static void main(String args[])  
03:     {  
04:         byte a = 127;           ← byte 형  
05:         System.out.println("127을 저장한 byte 값은: " + a);  
06:         short b = 32767;       ← short 형  
07:         System.out.println("32767을 저장한 short 값은 : " + b);  
08:         int c = 2147483647;    ← int 형  
09:         System.out.println("2147483647을 저장한 int 값은 : " + c);  
10:         long d = 9223372036854775807L;   ← long 형  
11:         System.out.println("9223372036854775807을 저장한 long 값은 : " + d);  
12:     }  
13: }
```

실행 결과

```
127을 저장한 byte 값은: 127  
32767을 저장한 short 값은 : 32767  
2147483647을 저장한 int 값은 : 2147483647  
9223372036854775807을 저장한 long 값은 : 9223372036854775807
```

## ● 예제 3.3 : 값의 범위를 벗어나면 오류 발생

### 예제 3.3

### ByteTestError.java

```
01: public class ByteTestError {  
02:     public static void main(String args[])  
03:     {  
04:         byte a = 128; ← 정수를 사용하면 int 형으로 취급한다. 127까지는  
                     자동으로 배정되지만, 범위를 벗어나면 오류 발생  
05:         System.out.println("128을 저장한 byte 값은: " + a);  
06:     }  
07:  
08: }
```

### 실행 결과

오류 발생

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
  Type mismatch: cannot convert from int to byte  
  
  at ByteTestError.main(ByteTestError.java:4)
```



## ● 예제 3.4 : 값의 범위를 벗어난 값을 강제 형 변환 하면 배정이 가능하다. 그러나 결과는?

예제 3.4

ByteTypeTest.java

```
01: public class ByteTypeTest {  
02:     public static void main(String args[])  
03:     {  
04:         byte a = (byte)128; ← 범위를 벗어났지만, byte  
05:         System.out.println("128을 저장한 byte 값은: " + a);  
06:         byte b = (byte)256; ← 형으로 강제 형 변환을 하여  
07:         System.out.println("256을 저장한 byte 값은: " + b);  
08:     }  
09: }
```

범위를 벗어났지만, byte  
형으로 강제 형 변환을 하여  
배정

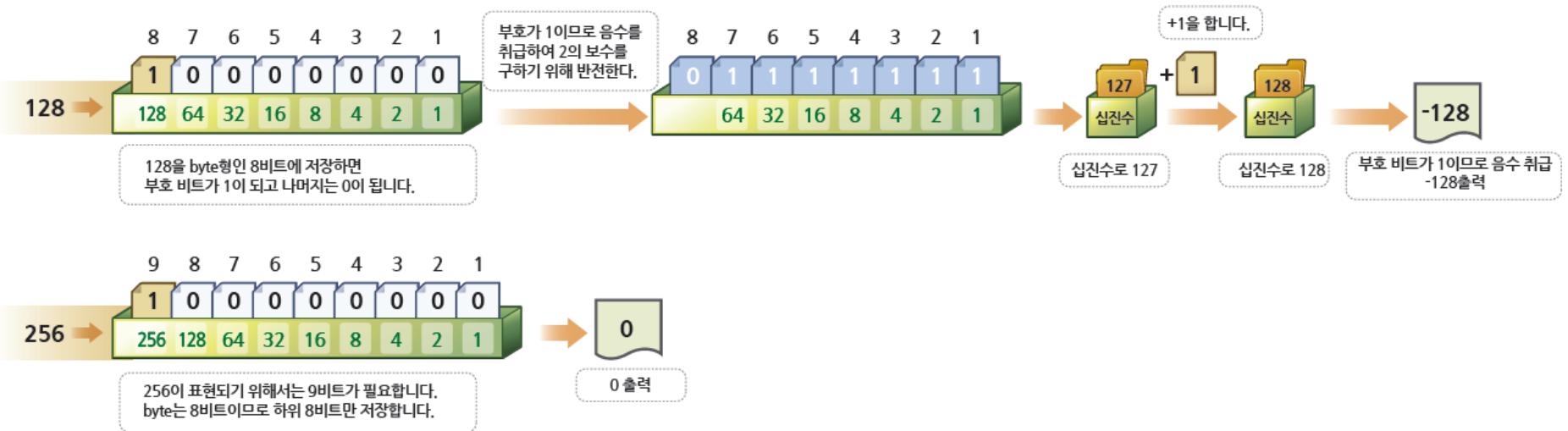
실행 결과

```
128을 저장한 byte 값은: -128  
256을 저장한 byte 값은: 0
```

## 3-2 정수형

- 예제 3.4 : 값의 범위를 벗어난 값을 강제 형 변환 하면 배정이 가능하다. 그러나 결과는?

- 이유는?





## ● 예제 3.5 : 8진수와 16진수 사용 가능. 정수 앞에 숫자 0과 0x를 사용하여 진법 표시

예제 3.5

LiteralTest.java

```
01: public class LiteralTest {  
02:     public static void main(String[] args) {  
03:         int a = 100;  
04:         int b = 0b1100100; ← 2진수로 초기화  
05:         int c = 0144; ← 8진수로 초기화  
06:         int d = 0x64; ← 16진수로 초기화  
07:         System.out.println("10진수 100 = " + a);  
08:         System.out.println("2진수 0b1100100 = " + b);  
09:         System.out.println("8진수 0144 = " + c);  
10:         System.out.println("16진수 0x64 = " + d);  
11:         System.out.println("2진수 0b111 = " + 0b111); ← 2진수로 출력  
12:         System.out.println("8진수 0777 = " + 0777); ← 8진수로 출력  
13:         System.out.println("16진수 0xffff = " + 0xffff); ← 16진수로 출력  
14:     }  
15: }
```

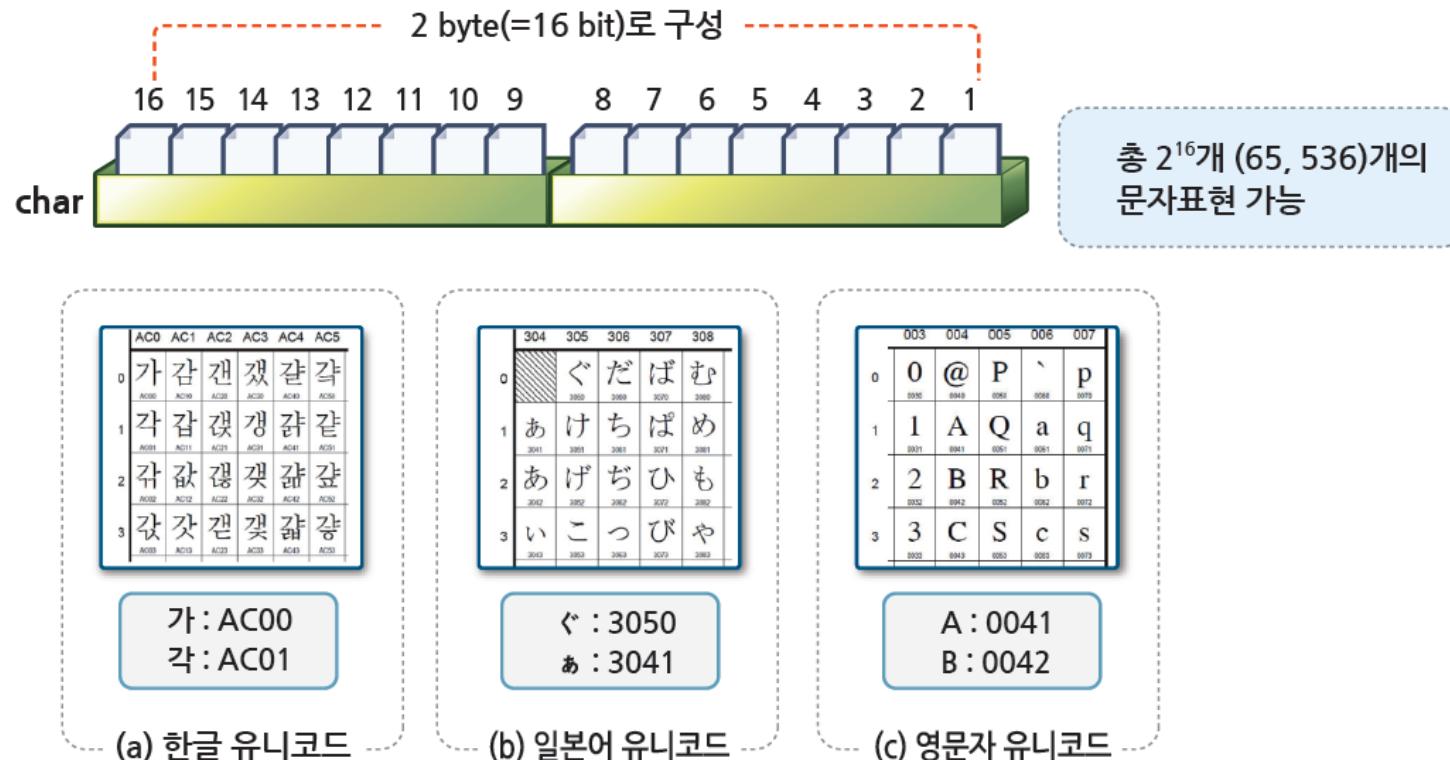
실행 결과

```
10진수 100 = 100  
2진수 0b1100100 = 100  
8진수 0144 = 100  
16진수 0x64 = 100  
2진수 0b111 = 7  
8진수 0777 = 511  
16진수 0xffff = 4095
```

## 3-2 정수형

### ● 문자 정수형 : 하나의 문자를 나타낼 수 있는 char형

- 자바의 문자는 16비트 유니코드로 구성





## ● 하나의 문자를 나타내기 위해 하나의 따옴표(' ')를 사용

- 대문자 'A'를 표시하는 방법 : 문자, 10진수, 8진수, 16진수, 유니코드로 표현 가능

```
char ch1 = 'A';  
char ch2 = 65; ← 10진수 값으로도 지정 가능  
char ch3 = 0101; ← 8진수 값으로도 지정 가능  
char ch4 = 0x41; ← 16진수 값으로도 지정 가능  
char ch5 = \u0041 ← 유니코드값으로도 지정 가능
```



## ● 문자형 예제 3.6

## 예제 3.6

## CharacterTypeTest.java

```
01: public class CharacterTypeTest {  
02:     public static void main(String args[])  
03:     {  
04:         char c1 = '한';   ←————— 한글을 저장  
05:         char c2 = '국';  
06:         char c3 = '韓'; ←————— 한자를 저장  
07:         char c4 = '國';  
08:         char c5 = 'K';   ←————— 영문자를 저장  
09:         char c6 = '\u004f';  
10:         char c7 = 'R';  
11:         char c8 = '\u0045';  
12:         char c9 = 'A';  
13:  
14:         System.out.println("출력 결과 : "+c1 + c2 + "(" + c3 + c4 + ")" + " = "  
15:             + c5 + c6 + c7 + c8 + c9 );  
16:     }
```

## 실행 결과

출력 결과 : 한국(韓國) = KOREA

## 3-2 정수형

## ● 특수 문자 표기 : 역슬래시 사용

특수 문자	표기 방법	유니코드 표기
Backspace	\b	\u0008
Tab	\t	\u0009
Linefeed	\n	\u000A
Formfeed	\f	\u000C
Carriage Return	\r	\u000D
Backslash	\\\	\u005C
Single Quote	\'	\u0027
Double Quote	\"	\u0022



### 3 자료형

## 3-2 정수형

9<sup>th</sup> edition

### ● 특수 문자 표기 : 예제 3.7

예제 3.7

CharacterSpecialType.java

```
01: public class CharacterSpecialType {  
02:     public static void main(String args[]) {  
03:           
04:             char ch1 = '\\"';           ←  
05:             char ch2 = '대';  
06:             char ch3 = '한';  
07:             char ch4 = '\\"';           ←  
08:             char ch5 = '\t';           ←  
09:             char ch6 = '민';  
10:             char ch7 = '국';  
11:             char ch8 = '\n';           ←  
12:             char ch9 = '만';  
13:             char ch10 = '세';  
14:             System.out.println("출력 결과 :\n" + ch1 + ch2 + ch3 + ch4 + ch5 +  
15:             ch6 + ch7 + ch8 + ch9 + ch10);  
16:     }
```

출력 결과

출력 결과:  
“대한” 민국  
만세

특수 문자를 저장



## ● 특수 문자 표기 : 예제 3.8

예제 3.8

CharacterNumericTest.java

```
01: public class CharacterNumericTest {  
02:     public static void main(String[] args) {  
03:         char a = 65;  
04:         char b = 0x41; ←———— 16진수 41에 해당하는 문자 배정  
05:         char c = 0101; ←———— 8진수 101에 해당하는 문자 배정  
06:         System.out.println(a);  
07:         System.out.println(b);  
08:         System.out.println(c);  
09:     }  
10: }
```

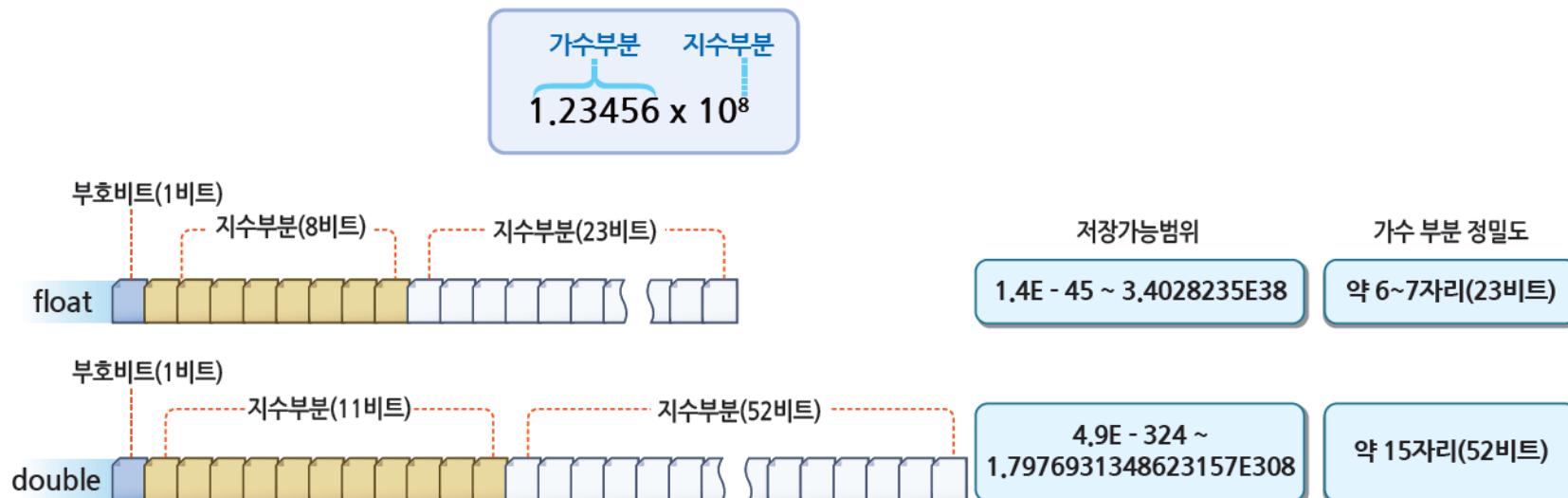
실행 결과

A  
A  
A

## 3-3 실수형

### ● 실수형

- 부호와 지수(exponential)부분, 가수(mantissa)부분으로 구성
- 저장할 수 있는 크기에 따라 float형과 double형으로 구분
- 묵시적(default) 데이터형은 double형





## 3-3 실수형

- 실수형 : float를 사용하려면 반드시 f를 붙여야 한다

```
float f = 3.14f; ← 오류 없음
```

```
float f = 3.14; ← 구문 오류 발생. 실수는 묵시적으로 double로 취급하므로 배정 불가
```

```
double d = 3.14; ← 오류 없음
```

```
double d = 3.14d; ← 오류 없음
```



## ● 예제 3.9

예제 3.9

FloatDoubleTest.java

```
01: public class FloatDoubleTest {  
02:     public static void main(String args[])  
03:     {  
04:         float a = 12345678901234567890.0f; // float형으로 유효 자리수가 많은 수를 배정  
05:         double b = 12345678901234567890.0; // double형으로 유효 자리수가 많은 수를 배정  
06:         System.out.println("float 변수 a의 값은 : " + a);  
07:         System.out.println("double 변수 b의 값은 : " + b);  
08:         float c = 1.0f / 3.0f; // float형으로 연산을 수행  
09:         double d = 1.0 / 3.0; // double형으로 연산을 수행  
10:         System.out.println("float 변수 c의 값은 : " + c);  
11:         System.out.println("double 변수 d의 값은 : " + d);  
12:     }  
13: }
```

## 실행 결과

```
float 변수 a의 값은 : 1.2345679E19  
double 변수 b의 값은 : 1.2345678901234567E19  
float 변수 c의 값은 : 0.33333334  
double 변수 d의 값은 : 0.3333333333333333  
3.4 논리형 Boolean type
```

## ● 논리형 변수 : 참(true) 또는 거짓(false)을 저장하는 변수

예제 3.10

BooleanTypeTest.java

```
01: public class BooleanTypeTest {  
02:     public static void main(String args[])  
03:     {  
04:         boolean a = true;           ← 논리형 변수에 true값을 저장  
05:         System.out.println("boolean 변수 a의 값은 : " + a);  
06:         boolean b = 10 > 20;        ← 논리형 변수에 (10 > 20)의 결과를 저장  
07:         System.out.println("boolean 변수 b의 값은 : " + b);  
08:         boolean c = a;           ← 논리형 변수에 논리형 변수값을 저장  
09:         System.out.println("boolean 변수 c의 값은 : " + c);  
10:     }  
11: }
```

실행 결과

```
boolean 변수 a의 값은 : true  
boolean 변수 b의 값은 : false  
boolean 변수 c의 값은 : true
```



- 상수 : 일반적으로 변하지 않는 값을 저장하는 변수
- 리터럴 : 값 자체를 의미
- 상수 선언의 예

```
final int MAX = 100;           ← 상수변수로 MAX를 선언하고 100으로 초기화  
final double PI;             ← 상수변수 PI를 선언  
PI = 3.14159;                ← PI 초기화  
PI = 3.14;                   ← 오류발생. 상수는 한번 값이 결정되면 변할 수 없음  
final boolean FLAG = true;    ← 상수변수로 boolean 형의 변수 FLAG를 선언하고 true 값으로 초기화
```



## 3-5 상수와 리터럴

## ● 리터럴 선언의 예

```
int num1 = 20;           ← 십진수 리터럴  
int num2 = 0b1001;       ← 2진수 리터럴  
int num3 = 070;          ← 8진수 리터럴  
int num4 = 0x20;          ← 16진수 리터럴  
double pi = 3.14159;      ← double 형 실수 리터럴  
double value = 0.12345E-25; ← double 형 실수 리터럴(지수승 표현)  
float ratio = 0.024f;       ← float 형 실수 리터럴  
char c = '김';            ← char 형 문자 리터럴  
String s = "대한민국";    ← String 형 문자열 리터럴  
boolean flag = true;        ← boolean 형 논리 리터럴
```



## ● 프로그램에서 상수를 사용하는 이유

- : 같은 리터럴이 여러 번 사용되는 경우의 효율성을 위해 사용

```
01: final int MAX = 100; // 상수 MAX를 100으로 선언. MAX의 값이 변경될 경우 상수 값만 변경  
02: int i = 1;  
03: double avg, sum=0.0;  
04: while(i <= MAX){ //  
05:     //....  
06: }  
07: avg = sum / MAX ; //  
08: System.out.println(MAX+"까지의 합은 : " + sum); //  
09: System.out.println(MAX+"까지의 평균은 : " + avg); //
```

상수 MAX 사용.  
상수를 사용하지 않은 경우  
값이 변경된 경우 MAX가  
사용된 모든 곳을 수정

## 3-6 형식 지정자를 사용한 출력 : printf()

## ● 출력문 : System.out.printf() 문을 제공

- printf() 출력문은 C언어의 출력문과 유사한 형식으로 사용

## [ 형식 ]

```
System.out.printf([형식제어문자],[변수 또는 리터럴]);
```

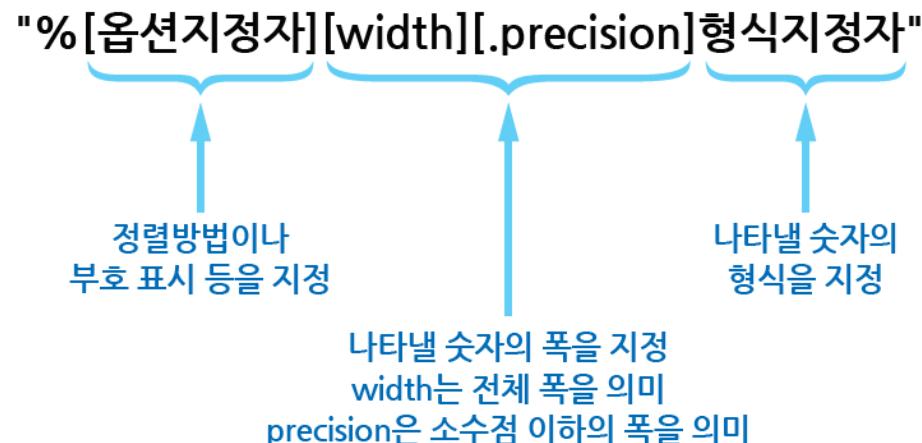


그림 3-11 형식 제어 문자의 구성

## 3-6 형식 지정자를 사용한 출력 : printf()

### ● 형식 지정자

형식 지정자	데이터 타입	출력 형식
%d	byte, short, int, long	부호가 있는 10진수 정수 출력
%X, %x	int	16진수를 출력(양수만 가능). 대문자 X의 경우 16진수 알파벳을 대문자로 표시
%o	int	양수의 8진수를 출력(양수만 가능).
%c	char	한 개의 문자 출력
%f	double	고정 소수점 실수 출력
%E, %e	double	부동 소수점 실수 출력. 대문자 E의 경우 지수 문자로 'E'를 사용
%G, %g	double	소수점 이하 자리수가 고정 또는 부동소수점으로 출력. 자리수가 짧은 것을 기준으로 선택함. 대문자 G의 경우 지수 문자로 'E'를 사용
%s	String	문자열 출력
%s	String	문자열 출력
%%		%출력

## 3-6 형식 지정자를 사용한 출력 : printf()

### ● 옵션 지정자

옵션 지정자	의미
-	폭이 지정된 경우 지정된 폭에서 좌측 정렬. 지정되지 않으면 묵시적으로 우측정렬
+	출력결과에 부호기호를 붙인다. 지정되지 않은 경우음수에만 부호가 붙는다
0	오른쪽 정렬인 경우 앞쪽을 모두 0으로 채우고, 지정되지 않은 경우 0으로 채우지 않음
#	형식지정자가 o(8진수)인 경우 앞에 0을 붙여 출력하고, x또는 X(16진수)인 경우 앞에 0x 또는 0X를 붙여 출력

- System.out.printf() 함수는 형식 제어 문자 부분에 일반 문자열을 혼용하여 사용 가능

```

System.out.printf("감사합니다\n");
System.out.printf("%s\n", "감사합니다");           ← 문자열 자체를 출력. 동일한 결과 출력
System.out.printf("%d번째 손님입니다\n", count);   ← 숫자(count 변수 값)와 같이 문자열 출력
System.out.printf("%d번째 손님께서 %s을(는) %d개 주문 하셨습니다\n", count, title, num2); ← 숫자와 문자열과 같이 출력
    
```



### 3 자료형

## 3-6 형식 지정자를 사용한 출력 : printf()

9<sup>th</sup> edition

### ▶ 저스팅 가이 츠려 . 챕터 03 : 11요

예제 3.11

TestInteger.java

```
01: public class TestInteger {  
02:     public static void main(String args[]) {  
03:         int u_price=-300; ← 정수 변수를 선언하고 초기화  
04:         int count = 9; ← %d를 사용하여 정수 리터럴을 출력  
05:  
06:         System.out.printf("%d원짜리 %d개를 사면 %d입니다\n",300,9,300*9); ←  
07:         System.out.printf("%10d원짜리 %+10d개를 사면 %10d입니다\n",u_price,  
08:             count, u_price*count); ← %10d를 사용하여 변수 값 출력. +를 붙이면 기호를 출력  
09:         System.out.printf("10진수 15를 16진수로 출력하면 : %x, 8진수로  
10:             출력하면 : %o\n", 15, 15); ← 리터럴(10진수) 15를 %x와 %o를 사용하여 16진수와 8진수로 출력  
11:     }
```

### 실행 결과

300원짜리 9개를 사면 2700입니다  
-300원짜리 +9개를 사면 -2700입니다  
10진수 15를 16진수로 출력하면 : f, 8진수로 출력하면 : 17  
음수값 -10을 %d로 출력하면 -10

## 3-6 형식 지정자를 사용한 출력 : printf()

- 형식 지정자와 옵션 지정자를 사용하여 정수 리터럴을 출력하는 예

("%d\n", 1234);

1	2	3	4
---	---	---	---

▶ %d만 사용

("%+d\n", 1234);

+	1	2	3	4
---	---	---	---	---

▶ +를 사용하여 부호를 지정

("%10d\n", 1234);

					1	2	3	4
--	--	--	--	--	---	---	---	---

▶ 출력 폭을 지정. 뮤시적으로 오른쪽 정렬

("%010d\n", 1234);

0	0	0	0	0	0	1	2	3	4
---	---	---	---	---	---	---	---	---	---

▶ 폭과 0을 지정. 오른쪽 정렬하고 0으로 채움

("%-10d\n", 1234);

1	2	3	4					
---	---	---	---	--	--	--	--	--

▶ 폭과 왼쪽 정렬을 지정

("%-+10d\n", 1234);

+	1	2	3	4				
---	---	---	---	---	--	--	--	--

▶ 폭과 왼쪽 정렬과 부호를 지정

("%+10d\n", 1234);

				+	1	2	3	4
--	--	--	--	---	---	---	---	---

▶ 폭과 부호를 지정

("%d\n", -1234);

-	1	2	3	4
---	---	---	---	---

▶ 음수를 출력

("%10d\n", -1234);

				-	1	2	3	4
--	--	--	--	---	---	---	---	---

▶ 폭을 지정하여 음수 출력

("%010d\n", -1234);

-	0	0	0	0	0	1	2	3	4
---	---	---	---	---	---	---	---	---	---

▶ 폭과 0을 지정. 왼쪽을 0으로 채움

("%#10o\n", 1234);

				0	2	3	2	2
--	--	--	--	---	---	---	---	---

▶ #을 지정하여 8진수 표시 출력

("%#10X\n", 1234);

				0	X	4	D	2
--	--	--	--	---	---	---	---	---

▶ #을 지정하여 16진수(0X) 표시 출력

("%#10x\n", 1234);

				0	X	4	d	2
--	--	--	--	---	---	---	---	---

▶ #을 지정하여 16진수(0x) 표시 출력

## 3-6 형식 지정자를 사용한 출력 : printf()

### ● 실수형 값의 출력 : 주로 %f, %e 사용

예제 3.12

TestDouble.java

```

01: public class TestDouble {
02:     public static void main(String[] args) {
03:         double r = 5.0;           ← 실수 변수를 선언하고 초기화
04:         double pi = 3.14159;    ←
05:
06:         System.out.printf("반지름이 %f인 원의 넓이는 %f입니다\n", 5.0,
07:                            3.14159*5*5); ← %f를 사용하여 실수를 출력
08:         System.out.printf("반지름이 %f인 원의 넓이는 %f입니다\n", r,
09:                            pi*r*r); ← %f를 사용하여 실수 변수 값을 출력
10:         System.out.printf("반지름이 %e인 원의 넓이는 %e입니다\n", r,
11:                            pi*r*r); ← %e를 사용하여 실수 변수 값을 출력          ← %e를 사용하여 123.456 출력
12:         System.out.printf("123.456을 %f로 : %f\n", 123.456); ←
13:         System.out.printf("123.456을 %e로 : %e\n", 123.456); ←
14:         System.out.printf("-123.456을 %E로 : %E\n", -123.456); ←
15:     }
16: }
```

실행 결과

```

반지름이 5.000000인 원의 넓이는 78.539750입니다
반지름이 5.000000인 원의 넓이는 78.539750입니다
반지름이 5.000000e+00인 원의 넓이는 7.853975e+01입니다
123.456을 %f로 : 123.456000
123.456을 %e로 : 1.234560e+02
-123.456을 %E로 : -1.234560E+02

```

## 3-6 형식 지정자를 사용한 출력 : printf()

- 형식 지정자와 옵션 지정자를 사용하여 실수 리터럴을 출력하는 예

("%10.6f%n",123.456); 1 | 2 | 3 | . | 4 | 5 | 6 | 0 | 0 | 0 ► %f 사용. 소수점 이하를 6자리로 지정

("%10.3f%n",123.456); 1 | 2 | 3 | . | 4 | 5 | 6 ► 소수점 이하를 3자리로 지정

("%10.1f%n",123.456); 1 | 2 | 3 | . | 5 ► 소수점 이하를 1자리로 지정. 반올림 수행

("%+10.3f%n",123.456); + | 1 | 2 | 3 | . | 4 | 5 | 6 ► 부호를 지정

(%"-10.3f%n",123.456); 1 | 2 | 3 | . | 4 | 5 | 6 | | | ► 왼쪽 정렬을 지정

(%"-+10.3f%n",123.456); + | 1 | 2 | 3 | . | 4 | 5 | 6 | | | ► 왼쪽 정렬과 부호를 지정

(%"0+10.3f%n",123.456); + | 0 | 0 | 1 | 2 | 3 | . | 4 | 5 | 6 ► 부호를 지정하고 왼쪽을 0으로 채움

(%"++10.3f%n",-123.456); - | 1 | 2 | 3 | . | 4 | 5 | 6 | | | ► 음수 실수 값을 부호를 지정하여 출력

(%"--10.3f%n",-123.456); - | 1 | 2 | 3 | . | 4 | 5 | 6 | | | ► 음수 실수 값을 부호와 왼쪽 정렬 지정하여 출력

(%"10.6e%n",123.456); 1 | . | 2 | 3 | 4 | 5 | 6 | 0 | e | + | 0 | 0 | 2 ► 지수 형태로 출력. 지정된 폭을 초과하는 경우 그대로 출력

(%"10.3e%n",123.456); 1 | . | 2 | 3 | 5 | e | + | 0 | 0 | 2 ► 지수 형태로 출력

(%"10.1e%n",123.456); 1 | . | 2 | e | + | 0 | 0 | 2 ► 지수 형태로 출력

(%"10.1e%n",-123.456); - | 1 | . | 2 | e | + | 0 | 0 | 2 ► 음수 실수를 지수 형태로 출력

(%"10.1e%n",0.0123456); 1 | . | 2 | e | - | 0 | 0 | 2 ► 지수 형태로 출력

## 3-6 형식 지정자를 사용한 출력 : printf()

- 문자와 문자열 값의 출력 : 주로 %c, %s 사용

예제 3.13

TestCharString.java

```

01: public class TestCharString {
02:     public static void main(String[] args) {
03:         char ch1 = 'A';           ← 문자형 변수를 선언하고 A로 초기화
04:         String name = "홍길동"; ← 문자열을 생성하기 위해 String 클래스의 객체 선언
05:
06:         System.out.printf("%c\n", 'A'); ← %c를 사용하여 문자 리터럴 출력
07:         System.out.printf("%s\n", "AB"); ← %s를 사용하여 문자열 리터럴을 출력
08:         System.out.printf("%s\n", "AB CCCCC");
09:         System.out.printf("%s씨 %s\n%s\n", name, "재미있는 자바언어", "열심히
10:                           하세요"); ← %s를 사용하여 문자열 객체와 문자열 리터럴을 출력
11:     }

```

실행 결과

A

AB

AB CCCCC

홍길동씨 재미있는 자바언어  
열심히 하세요



### 3 자료형

## 3-6 형식 지정자를 사용한 출력 : printf()

9<sup>th</sup> edition

- 형식 지정자와 옵션 지정자를 사용하여 문자와 문자열을 출력하는 예

("%10c\n",'a'); ▶ 폭을 지정하여 문자 a 출력

(%-10c\n,'a'); ▶ 폭과 왼쪽 정렬을 지정

(%10s\n,"ABCDE"); 

(%-10s\n,"ABCDE"); ▶ 폭과 왼쪽 정렬을 지정

## 4-1 연산과 자료형

- 자바는 목시적으로 정수 리터럴은 int 형, 실수 리터럴은 double 형으로 취급
- 연산을 수행할 때 두 개의 피연산자가 다른 형일 경우 자동으로 확대 형 변환을 수행

byte >> short/char >> int >> long >> float >> double

- 확대 형 변환의 순서 -

### [ 코드 ]

```
int avg = 9 / 2;           ← int = int / int 정수 연산이 수행되어 결과로 4가 저장됨  
double avg = 9 / 2        ← double = int / int 정수 연산이 수행된 결과 4가 double 변수에 저장될 때 4.0으로 변환되어 저장  
double avg = 9 / 2.0       ← double = int / double 앞쪽의 피연산자가 double로 변환되어 실수 연산이 수행됨.  
                           결과로 4.5가 double 변수에 저장됨  
double avg = 9.0 / 2       ← double = double / int 뒤쪽의 피연산자가 double로 변환되어 실수 연산이 수행됨.  
                           결과로 4.5가 double 변수에 저장됨  
double avg = 9.0 / 2.0     ← double = double / double 모든 연산이 실수 연산으로 수행되어 4.5가 저장됨  
int avg = 9.0 / 2.0        ← 구문 오류 발생됨. 연산은 double로 수행되어 결과도 double, double형 변수값을 정수로  
                           자동 변환 불가
```



## 4 연산과 형 변환

### 4-1 연산과 자료형

9<sup>th</sup> edition

예제 3.14

DatatypeOperation1.java

```
01: import java.util.Scanner;
02: public class DatatypeOperation1 {
03:     public static void main(String[] args) {
04:         Scanner stdin = new Scanner(System.in);
05:         System.out.print("첫 번째 정수를 입력 : ");
06:         int first = stdin.nextInt();           ← 첫 번째로 입력된 정수 저장
07:         System.out.print("두 번째 정수를 입력 : ");
08:         int second = stdin.nextInt();          ← 두 번째로 입력된 정수 저장
09:         double avg1 = (first+second) / 2 ;      ← 정수로 평균을 구하는 연산을 수행
10:         System.out.println("정수 연산 : 평균은("+first+"+"+second+")/2 =
11: " +avg1+ " 입니다");
12:         double avg2 = (first+second) / 2.0 ;    ← 실수로 평균을 구하는 연산을 수행
13:         System.out.println("실수 연산 : 평균은("+first+"+"+second+")/2.0 =
14: " +avg2+ " 입니다");
```

실행 결과

```
첫 번째 정수를 입력 : 3
두 번째 정수를 입력 : 6
정수 연산 : 평균은(3+6)/2 = 4.0 입니다
실수 연산 : 평균은(3+6)/2.0 = 4.5 입니다
```



#### ● 형 변환(Casting)

- 특정 자료형의 값을 다른 형태의 변수에 배정할 때 발생
- 확대(widening) 형 변환과 축소(narrowing) 형 변환으로 구분
- 확대 형 변환
  - 두 개의 타입이 같거나 치역(target type)이 정의역(source type)보다 더 넓어 값의 손실이 발생되지 않고 저장
  - 자동으로 형 변환이 발생한다
- 축소 형 변환
  - 확대 형 변환의 반대의 경우
  - 명시적이 혼 변화 구문을 사용

#### 【형 변환 구문】

(type) 식 또는 변수

#### 【 예 】

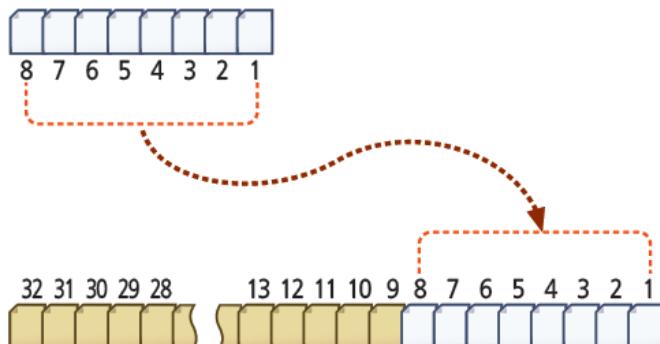
```
(double) 2 // 정수 2를 2.0으로 형 변환  
double avg1 = (double)(first+second) / 2 // first+second의 결과가 실수로 형 변환  
byte b = (byte) 700 // 정수 700이 바이트 형으로 형 변환
```

### 4-2 형 변환

**byte => int**

예) byte b = 120;  
int a = b;

byte형이 int형으로 변환되면  
값의 손실없이 하위 8비트에 저장됩니다.

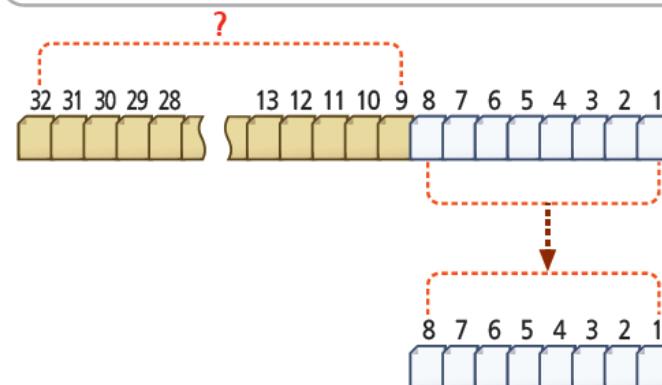


(a) 확대 형 변환

**int => byte**

예) int a = 259;  
byte b = (byte) a;

int형이 byte형으로 변환되면  
int형의 하위 8비트만 저장되므로 값의 손실이 발생됩니다



(b) 축소 형 변환



## 4 연산과 형 변환

### 4-2 형 변환

9<sup>th</sup> edition

예제 3.15

CastingTest1.java

```
01: public class CastingTest1 {  
02:     public static void main(String args[]){  
03:         byte b = 120;  
04:         int i = b; // byte형의 값을 int형의 변수에 저장  
05:         System.out.println("확대 형 변환 : "+i);  
06:         int j = 259;  
07:         double d = 259.428;  
08:         System.out.println("축소 형 변환 결과");  
09:         b = (byte) j; // int형의 값을 byte형으로 형 변환하여 저장(값의 손실 발생)  
10:         System.out.println("int 259를 byte로 : " + b);  
11:         i = (int) d; // double형의 값을 int형으로 형 변환하여 저장(값의 손실 발생)  
12:         System.out.println("double 259.428을 int로 : " + i);  
13:         b = (byte) d; // double형의 값을 byte형으로 형 변환하여 저장(값의 손실 발생)  
14:         System.out.println("double 259.428을 byte로 : " + b);  
15:     }  
16: }
```

실행 결과

```
확대 형 변환 : 120  
축소 형 변환 결과  
int 259를 byte로 : 3  
double 259.428을 int로 : 259  
double 259.428을 byte로 : 3
```



## 4 연산과 형 변환

### 4-2 형 변환

9<sup>th</sup> edition

#### 예제 3.16

#### CastingTest2.java

```
01: public class CastingTest2 {  
02:     public static void main(String args[])  
03:     {  
04:         int i = '1';           ← int형의 변수에 문자값을 배정  
05:         int j = 'A';           ← int 형의 변수에 문자값의 연산 결과를 배정  
06:         int k = '1' + 'A';    ← int형의 값은 char형으로 형 변환하여 출력  
07:         System.out.println("'1' = " + i);  
08:         System.out.println("'A' = " + j);  
09:         System.out.println("'1'+'A' = " + k);  
10:         int l = 66;  
11:         System.out.println("10진수 66에 해당되는 유니코드 문자는 " + (char)l +  
    " 입니다");           ← int형의 값을 char형으로 형 변환하여 출력  
12:     }  
13: }
```

#### 실행 결과

'1' = 49

'A' = 65

'1'+'A' = 114

10진수 66에 해당되는 유니코드 문자는 B 입니다



## ● 예제 3.14

예제 3.17

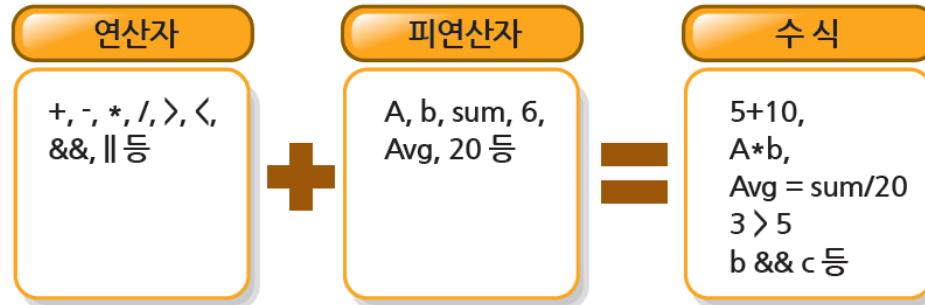
CastingTest3.java

```
01: public class CastingTest3 {  
02:     public static void main(String args[]) {  
03:         long l = 1234567890123456789L;      ← 19자리 정수를 long값으로 저장  
04:         float f = l;    ← 정수 long값을 float형으로 확대 형 변환(자동 변환 가능)  
05:         double d = l;   ← 정수 long값을 double형으로 확대 형 변환(자동 변환 가능)  
06:         System.out.println("원래의 long값 : "+l);  
07:         System.out.println("long >> float로 변환 : "+f);  
08:         System.out.println("long >> double로 변환 : "+d);  
09:         long fl = (long)f;    ← float값을 long값으로 명시적 축소 형 변환  
10:         long dl = (long)d;    ← double값을 long값으로 명시적 축소 형 변환  
11:         System.out.println("long >> float >> long으로 변환 : "+fl);  
12:         System.out.println("long >> double >> long으로 변환 : "+dl);  
13:     }  
14: }
```

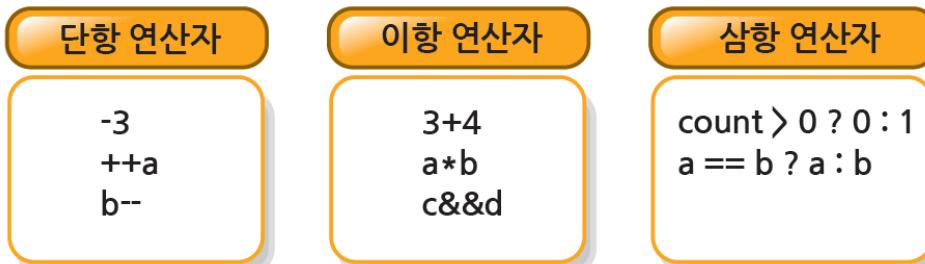
## 실행 결과

```
원래의 long값 : 1234567890123456789  
long >> float로 변환 : 1,23456794E18  
long >> double로 변환 : 1.23456789012345677E18  
long >> float >> long으로 변환 : 1234567939550609408  
long >> double >> long으로 변환 : 1234567890123456768
```

- 수식은 연산자와 피연산자로 구성



- 연산자의 사용 형태는 3가지





## 5 연산자와 수식

### 5-1 산술 연산자

9<sup>th</sup> edition

#### ● 자바의 산술 연산자

- 단항 연산자는 피연산자로 변수만 사용

연산자	사용법	설명	비고
+	op1+op2	op1과 op2를 더한다.	단항 및 이항
-	op-op2	op1에서 op2를 뺀다.	단항 및 이항
*	op1*op2	op1과 op2를 곱한다.	이항
/	op1/op2	op1을 op2로 나눈다.	이항
%	op1%op2	op1을 op2로 나눈 나머지를 구한다.	이항
++	var++ ++var	var 값 1 증가. var 값을 증가시키기 전에 평가 var 값 1 증가. var 값을 증가시킨 다음 평가	단항 단항
--	var-- --var	var 값 1 감소. var 값을 감소시키기 전에 평가 var 값 1 감소. var 값을 감소시킨 다음 평가	단항 단항



## 5 연산자와 수식

### 5-1 산술 연산자

9<sup>th</sup> edition

예제 3.18

ArithmeticOPTest1.java

```
01: public class ArithmeticOPTest1 {  
02:     public static void main(String args[])  
03:     {  
04:         int a=5, b=2 ;  
05:         int sum=a+b;  
06:         System.out.println("a+b=" + sum);  
07:         int sub=a-b;  
08:         System.out.println("a-b=" + sub);  
09:         int mul=a*b;  
10:         System.out.println("a*b=" + mul);  
11:         int div=a/b;  
12:         System.out.println("a/b=" + div);  
13:         int mod=a%b; ← 두 값을 나눈 나머지를 구하는 연산자  
14:         System.out.println("a%b=" + mod);  
15:         int c = ++a; ← 단항 연산자값을 증가시킨 후에 배정  
16:         System.out.println("a의 전위 증가 연산(prefix)="+c);  
17:         System.out.println("a 변수의 값 : "+a);  
18:         int d = b++; ← 단항 연산자값을 배정한 후에 1을 증가  
19:         System.out.println("b의 후위 증가 연산(postfix)="+d);  
20:         System.out.println("b 변수의 값 : "+b);  
21:     }  
22: }
```

실행 결과

```
a+b=7  
a-b=3  
a*b=10  
a/b=2  
a%b=1  
a의 전위 증가 연산(prefix)=6  
a 변수의 값 : 6  
b의 후위 증가 연산(postfix)=2  
b 변수의 값 : 3
```



## 5 연산자와 수식

### 5-1 산술 연산자

9<sup>th</sup> edition

#### ● 예제 3.16

예제 3.19

ArithmeticOPTest2.java

```
01: public class ArithmeticOPTest2 {  
02:     public static void main(String aa[]) {  
03:         byte a=127, b = 2;  
04:         byte c =(byte)(a*b); // int 연산 결과를 형 변환하여 byte 변수에 저장  
05:         System.out.println("a*b의 결과를 byte로 변환 출력 : " + c);  
06:         int d = a * b;  
07:         System.out.println("a*b의 결과를 int로 출력 : " + d);  
08:         int i = 1000000, j = 1000000; // int형으로 백만*백만을 수행하여 int형 변수에 저장  
09:         int k = i * j;  
10:         System.out.println("백만*백만의 결과를 int로 출력 : " + k);  
11:         long m = (long)(i*j); // int형으로 연산 수행 후에 long형으로 형 변환  
12:         System.out.println("곱셈(int 연산)의 결과를 long으로 변환한 후 출력 : " + m);  
13:         m = (long)i*j; // 변수를 long으로 형 변환한 후에 long 연산  
14:         System.out.println("i를 long 값으로 변환 후 곱셈(long 연산)하여 출력 : " + m);  
15:         m = (long)i*(long)j; // 각각을 long으로 형 변환한 다음 long 연산  
16:         System.out.println("백만을 long으로 변환 후 곱셈(long 연산) 결과를 출력 : " + m);  
17:  
18:     }  
19: }
```

#### 실행 결과

```
a*b의 결과를 byte로 변환 출력 : -2  
a*b의 결과를 int로 출력 : 254  
백만*백만의 결과를 int로 출력 : -727379968  
곱셈(int 연산)의 결과를 long으로 변환한 후 출력 : -727379968  
i를 long 값으로 변환 후 곱셈(long 연산)하여 출력 : 1000000000000  
백만을 long으로 변환 후 곱셈(long 연산) 결과를 출력 : 1000000000000
```



## 5 연산자와 수식

### 5-1 산술 연산자

9<sup>th</sup> edition

예제 3.20

ArithmeticOPTest3.java

```
01: public class ArithmeticOPTest3 {  
02:     public static void main(String aa[])  
03:     {  
04:         int a = 10;  
05:         System.out.println("(++a + ++a)의 결과는 : " + (++a + ++a));  
06:         a = 10;                                         ↑  
07:         System.out.println("(++a - ++a)의 결과는 : " + (++a - ++a));  
08:         a = 10;                                         ↑  
09:         System.out.println("(a++ + a++)의 결과는 : " + (a++ + a++));  
10:         a = 10;                                         ↑  
11:         System.out.println("(a++ - a++)의 결과는 : " + (a++ - a++));  
12:         a = 10;                                         ↑  
13:         System.out.println("(++a + a++)의 결과는 : " + (++a + a++));  
14:         a = 10;                                         ↑  
15:         System.out.println("(++a - a++)의 결과는 : " + (++a - a++));  
16:         a = 10;                                         ↑  
17:         System.out.println("(a++ + ++a)의 결과는 : " + (a++ + ++a));  
18:         a = 10;                                         ↑  
19:         System.out.println("(a++ - ++a)의 결과는 : " + (a++ - ++a));  
20:     }  
21: }
```

#### 실행 결과

(++a + ++a)의 결과는 : 23  
(++a - ++a)의 결과는 : -1  
(a++ + a++)의 결과는 : 21  
(a++ - a++)의 결과는 : -1  
(++a + a++)의 결과는 : 22  
(++a - a++)의 결과는 : 0  
(a++ + ++a)의 결과는 : 22  
(a++ - ++a)의 결과는 : -2



## 5 연산자와 수식

### 5-2 관계 및 논리 연산자

9<sup>th</sup> edition

#### ● 관계 연산자

- 두 개의 피연산자 값들을 비교하여 true 또는 false 값을 반환하는 연산자
- 선택문과 반복문의 조건식에 사용
- 피연산자가 서로 다른 형일 경우 자료형의 범위가 큰 쪽으로 자동 형 변환

연산자	사용법	설명
>	op1 > op2	op1이 op2보다 큰 경우
>=	op1 >= op2	op1이 op2보다 크거나 같은 경우
<	op1 < op2	op1이 op2보다 작은 경우
<=	op1 <= op2	op1이 op2보다 작거나 같은 경우
==	op1 == op2	op1과 op2가 같은 경우
!=	op1 != op2	op1과 op2가 같지 않은 경우
instanceof	op1 instanceof op2	op1이 op2의 인스턴스(객체)인 경우



## 5 연산자와 수식

### 5-2 관계 및 논리 연산자

9<sup>th</sup> edition

예제 3.21

RelationalOPTest.java

```
01: public class RelationalOPTest {  
02:     public static void main(String args[]) {  
03:         {  
04:             byte a = 20;  
05:             double d = 3.14;  
06:             boolean flag;  
07:             flag = a > d; // 관계 연산자 사용. 결과를 이진 변수에 저장  
08:             System.out.println("a가 d보다 큰가? " + flag);  
09:             flag = a == 20.0f; // byte형이 float형으로 자동 변환되어 비교  
10:             System.out.println("a가 20.0f와 같은가? " + flag);  
11:             flag = 10 != 10.0; // 정수 리터럴 10이 실수로 변환되어 비교  
12:             System.out.println("10이 10과 같지 않은가? " + flag);  
13:             flag = 10 <= 20; // 관계 연산자 사용  
14:             System.out.println("10이 20보다 작거나 같은가? " + flag);  
15:             System.out.println("10이 20보다 작은가? " + (10 < 20));  
16:             System.out.println("10이 20보다 크거나 같은가? " + (10 >= 20));  
17:         }  
18:     }
```

실행 결과

a가 d보다 큰가? true  
a가 20.0f와 같은가? true  
10이 10과 같지 않은가? false  
10이 20보다 작거나 같은가? true  
10이 20보다 작은가? true  
10이 20보다 크거나 같은가? false

관계 연산자를 출력문에  
직접 사용.

### 5-2 관계 및 논리 연산자

#### ● 논리 연산자

- 두 개의 피연산자의 값을 평가하여 true 또는 false 값을 반환
- 두 개의 피연산자가 반드시 true 또는 false 값을 가져야 한다
- 이항 논리 연산자 : &&(AND) , ||(OR)

x	y	$x  y$	$x \&\& y$
true	true	true	true
true	false	true	false
false	true	true	false
false	false	false	false

- 단항 논리 연산자 · ! (NOT)

x	$!x$
true	false
false	true



## 5 연산자와 수식

### 5-2 관계 및 논리 연산자

9<sup>th</sup> edition

#### ● 관계 연산자와 논리 연산자의 사용 예

```
boolean flag;  
flag = 30 < 20; ← 사용 가능  
flag = 30 && 20; ← 사용 불가능  
flag = 30 < 20 || 50 > 20; ← 사용 가능  
flag = 30 < 20 && 80; ← 사용 불가능  
flag = 20 + 30 < 20 * 2; ← 사용 가능  
flag = 20 + 30 || 20 * 2; ← 사용 불가능
```



## 5 연산자와 수식

### 5-2 관계 및 논리 연산자

9<sup>th</sup> edition

#### ● 관계 연산자와 논리 연산자의 사용 예

예제 3.22

LogicalOPTTest.java

```
01: public class LogicalOPTTest {  
02:     public static void main(String args[])  
03:     {  
04:         boolean a;  
05:         a = (20 > 10) || (30 > 40); ← OR 논리 연산자 수행  
06:         System.out.println("20이 10보다 크거나 또는(논리합 ||) 30이 40보다 큰가? " +  
07:             a);  
08:         a = (20 > 10) && (30 > 40); ← AND 논리 연산자 수행  
09:         System.out.println("20이 10보다 크고 그리고(논리곱 &&) 30이 40보다 큰가? " +  
10:             a);  
11:         a = ! true; ← NOT 논리 연산자 수행  
12:         System.out.println("ture의 !(not)은? " + a);  
13:         System.out.println("20이 10보다 크거나 또는(논리합 ||) 30이 40보다  
14:             큰가? " + ((20 > 10) || (30 > 40)));  
15:         System.out.println("20이 10보다 크고 그리고(논리곱 &&) 30이 40보다  
16:             큰가? " + ((20 > 10) && (30 > 40)));  
17:         System.out.println("ture의 !(not)은? " + (! true)); ← 출력문에 논리 연산자를 직접 사용  
18:     }  
19: }
```

실행 결과

```
20이 10보다 크거나 또는(논리합 ||) 30이 40보다 큰가? true  
20이 10보다 크고 그리고(논리곱 &&) 30이 40보다 큰가? false  
ture의 !(not)은? false  
20이 10보다 크거나 또는(논리합 ||) 30이 40보다 큰가? true  
20이 10보다 크고 그리고(논리곱 &&) 30이 40보다 큰가? false  
ture의 !(not)은? false
```

## 5-3 비트 연산자

- 비트 연산자는 2진수로 표현된 정수를 비트 단위로 취급하는 연산자
- 비트 연산자에는 비트 논리 연산자와 시프트 연산자가 있다
- 비트 논리 연산자

연산자	사용법	설명
&	5 & 7	5에 해당하는 101과 7에 해당하는 111을 비트 단위로 AND 한다
	5   7	5에 해당하는 101과 7에 해당하는 111을 비트 단위로 OR 한다
^	5 ^ 7	5에 해당하는 101과 7에 해당하는 111을 비트 단위로 XOR(exclusive OR) 한다
~	~5	5에 해당하는 101의 보수를 취한다(int를 32비트로 표현하므로 보수는 1111111111111111010)



## 5 연산자와 수식

### 5-3 비트 연산자

9<sup>th</sup> edition

#### ● 비트 논리 연산자의 진위표

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

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

x	y	x ^ y
true	true	false
true	false	true
false	true	true
false	false	false

x	$\sim x$
true	false
false	true



## 5 연산자와 수식

## 5-3 비트 연산자

9<sup>th</sup> edition

- `Integer.toBinaryString()` : 정수를 비트 단위로 출력하는 라이브러리 메소드

### 예제 3.23

BitOPTest1.java

```
01: public class BitOPTest1 {  
02:     public static void main(String args[])  
03:     {  
04:         int a = 14;  
05:         int b = 11; ← 14에 대한 2진 표현  
06:         System.out.println(" a = " +a+"("+Integer.toBinaryString(a)+")");  
07:         System.out.println(" b = " +b+"("+Integer.toBinaryString(b)+")");  
08:         System.out.println("a&b = " +(a&b)+"("+Integer.toBinaryString(a&b)+")");  
09:         System.out.println("a|b = " +(a|b)+"("+Integer.toBinaryString(a|b)+")");  
10:         System.out.println("a^b = " +(a^b)+"("+Integer.toBinaryString(a^b)+")");  
11:         System.out.println(" ~b = " +(~b)+"("+Integer.toBinaryString(~b)+")");  
12:     } ← 14와 11의 비트 단위의 AND 결과 출력  
13: } ← 14와 11의 비트 단위의 OR 결과 출력  
          ← 14와 11의 비트 단위의 XOR 결과 출력
```

a&b = 10(1010)  
a|b = 15(1111)  
a^b = 5(101)  
~b = -12(111111111111111111111111)

## 실행 결과

#### ----- 14에 대한 2진 표현

보수가 출력된다. 음수로 취급되어 2의 보수를 취하면 10진수로 -12가 출력



## 5-3 비트 연산자

## ● 시프트 연산자 : 비트 단위로 이동(왼쪽 또는 오른쪽)하는 연산자

- 정수를 좌우로 시프트하면 곱셈과 나눗셈의 결과를 얻을 수 있다

연산자	사용법	설명
«	a « n	정수 a를 비트 단위로 왼쪽으로 n비트 시프트한다. 비어 된 비트에는 0으로 채운다. 결과는 $a * 2^n$
»	a » n	정수 a를 비트 단위로 오른쪽으로 n비트 시프트한다. 비어 된 비트에는 시프트 전의 부호 비트를 채운다. 결과는 $a / 2^n$
»»	a »» n	정수 a를 비트 단위로 오른쪽으로 n비트 시프트한다. 비어 된 비트에는 0으로 채운다. 부호 비트를 고려하지 않기 때문에 a가 음수일 경우 시프트 결과는 양수로 나타낸다.



## 5 연산자와 수식

### 5-3 비트 연산자

9<sup>th</sup> edition

## ● 예제 3.21

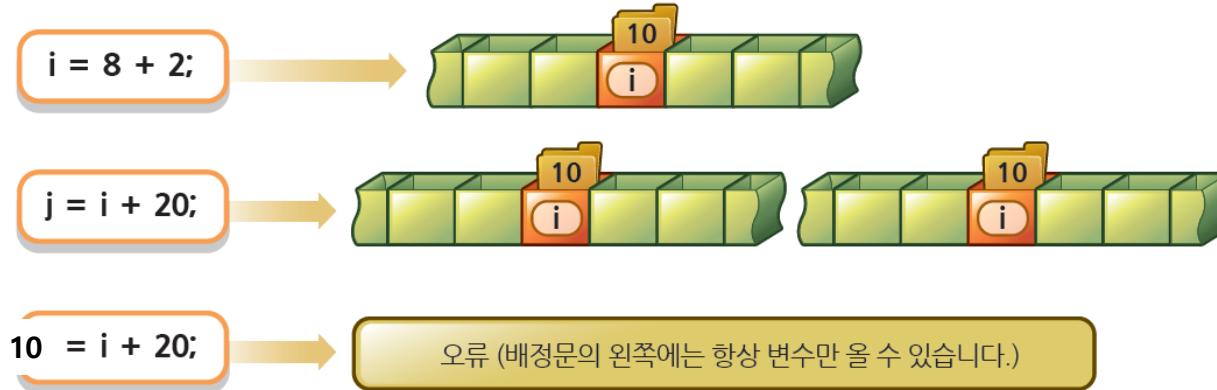
### 예제 3.24

ShiftOPTest1.java

```
01: import java.util.Scanner;
02: public class ShiftOPTest1 {
03:     public static void main(String args[])
04:     {
05:         Scanner stdin = new Scanner(System.in);
06:         System.out.print("두 개의 숫자를 입력 : ");
07:         int a = stdin.nextInt(); ← 키보드부터 두 개의 정수를 입력
08:         int b = stdin.nextInt(); ←
09:         System.out.println("    a = " +a+"("+Integer.toBinaryString(a)+")");
10:        System.out.println("    b = " +b+"("+Integer.toBinaryString(b)+")");
11:        System.out.println(" a<<2 = " +(a<<2)+"("+Integer.toBinaryString(a<<2)+")");
12:        System.out.println(" a>>2 = " +(a>>2)+"("+Integer.toBinaryString(a>>2)+")");
13:        System.out.println(" b<<2 = " +(b<<2)+"("+Integer.toBinaryString(b<<2)+")");
14:        System.out.println(" b>>2 = " +(b>>2)+"("+Integer.toBinaryString(b>>2)+")");
15:        System.out.println("b>>>2 = " +(b>>>2)+"("+Integer.toBinaryString(b>>>2)+")");
16:    } ← 왼쪽으로 2비트 시프트한 결과 출력
17: } ← 오른쪽으로 2비트 시프트한 결과 출력
          ← 오른쪽으로 2비트 시프트하면서 비 끝을 0으로 채운 결과 출력
```

## ● 자바의 배열 연산자 “=”

- 배열 연산자의 왼쪽에는 반드시 변수만 올 수 있다





## 5 연산자와 수식

### 5-4 배열 연산자와 단축 배열 연산자

9<sup>th</sup> edition

- 단축 배열 연산자 : 배열 연산자와 다른 연산자를 같이 사용할 수 있다

일반적 수식

단축 배열 연산자 수식

$j = j + 5;$  ----->  $j += 5;$

$j = j - 8;$  ----->  $j -= 8;$

$j = j * 10;$  ----->  $j *= 10;$

$j = j / 12;$  ----->  $j /= 12;$

## 5-4 배열 연산자와 단축 배열 연산자

### ● 단축 배열

연산자	사용법	의미
<code>+=</code>	<code>op1 += op2</code>	$op1 = op1 + op2$
<code>-=</code>	<code>op1 -= op2</code>	$op1 = op1 - op2$
<code>*=</code>	<code>op1 *= op2</code>	$op1 = op1 * op2$
<code>/=</code>	<code>op1 /= op2</code>	$op1 = op1 / op2$
<code>%=</code>	<code>op1 %= op2</code>	$op1 = op1 \% op2$
<code>&amp;=</code>	<code>op1 &amp;= op2</code>	$op1 = op1 \& op2$
<code> =</code>	<code>op1  = op2</code>	$op1 = op1   op2$
<code>^=</code>	<code>op1 ^= op2</code>	$op1 = op1 ^ op2$
<code>&lt;&lt;=</code>	<code>op1 &lt;&lt;= op2</code>	$op1 = op1 << op2$
<code>&gt;&gt;=</code>	<code>op1 &gt;&gt;= op2</code>	$op1 = op1 >> op2$
<code>&gt;&gt;&gt;=</code>	<code>op1 &gt;&gt;&gt;= op2</code>	$op1 = op1 >>> op2$



## 5 연산자와 수식

### 5-4 배열 연산자와 단축 배열 연산자

9<sup>th</sup> edition

#### 예제 3.25

#### ShiftOPTest1.java

```
01: public class AssignOPTest1 {  
02:     public static void main(String args[])  
03:     {  
04:         int a = 10;  
05:         System.out.println("a = " + a);  
06:         a += 4; // a = a+4를 수행  
07:         System.out.println("a += 4의 결과 " + a);  
08:         a %= 4; // a = a % 4를 수행. 14를 4로 나눈 나머지 2를 출력  
09:         System.out.println("a %= 4의 결과 " + a);  
10:         a <<= 4; // a = a << 4를 수행. 2를 왼쪽으로 4비트 시프트, 32출력  
11:         System.out.println("a <<= 4의 결과 " + a);  
12:         boolean b = false;  
13:         b &= a > 2; // b = b & a>2를 수행. false 출력  
14:         System.out.println("b &= a > 2의 결과 " + b);  
15:         b |= a > 2; // b = b | a>2를 수행. true 출력  
16:         System.out.println("b |= a > 2의 결과 " + b);  
17:     }  
18: }
```

#### 실행 결과

```
a = 10  
a += 4의 결과 14  
a %= 4의 결과 2  
a <<= 4의 결과 32  
b &= a > 2의 결과 false  
b |= a > 2의 결과 true
```



### 5-5 3항 연산자

- 3항 연산자 : 3개의 피연산자를 가진 3항 연산자 “ ?: ”제공

#### 【 형식 】 멤버 변수의 선언

수식1 ? 수식2 : 수식3

#### 【 코드 】

```
boolean flag;  
int a = 11;  
flag = a >= 10 ? true : false  
System.out.println(flag); ← true를 출력함  
  
int count = 10;  
System.out.println( count != 10 ? "10이 아님" : "10을 가짐"); ← "10을 가짐" 출력  
  
int b=-10;  
b = b >= 0 ? b * b : -b * -b;  
System.out.println(b); ← 100이 출력
```



## 5 연산자와 수식

### 5-5 3항 연산자

9<sup>th</sup> edition

예제 3.26

ShiftOPTest1.java

```
01: import java.util.Scanner;  
02:  
03: public class TernaryOPTest {  
04:     public static void main(String args[ ])  
05:     {  
06:         Scanner stdin = new Scanner(System.in);  
07:         System.out.print("한 개의 숫자를 입력 : ");  
08:         int a = stdin.nextInt();  
09:         boolean flag;  
10:         flag = (a % 2 == 0) ? true : false; ←----- 입력받은 숫자를 2로 나눈 나머지가  
11:         System.out.print(a +"이(가) 짝수입니까? : ");      0이면 true, 아니면 false를 반환  
12:         System.out.println(flag);  
13:     }  
14: }
```

실행 결과

한 개의 숫자를 입력 : 10  
10이(가) 짝수입니까? : true

- 동일한 수식에서 사용될 때 우선순위에 의해 수행 순서가 결정

우선순위	연산자	형식	이름
1	[]	a[b]	인덱스 연산자
	()	(a+b)*c	괄호
	.	a.b	멤버 접근 연산자
2	++	++a, a++	증가 연산자
	--	--a, a--	감소 연산자
	+	+a	단항 +연산자
	-	-a	단항 -연산자
	!	!a	NOT 연산자
	~	~a	비트 보수 연산자
3	new	new 클래스명	new 연산자
	()	(short)	캐스트 연산자
4	*	a * b	곱셈 연산자
	/	a / b	나눗셈 연산자
	%	a % b	나머지 연산자
5	+	a + b	+ 연산자
	-	a - b	- 연산자



## 6 연산자의 우선순위

9<sup>th</sup> edition

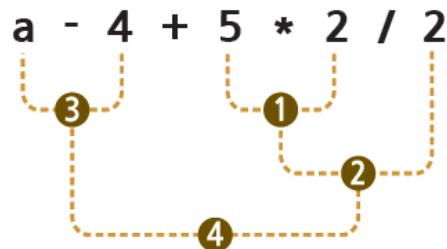
우선순위	연산자	형식	이름
6	«	a « b	시프트 연산자
	»	a » b	
	»»	a »» b	
7	<	a < b	관계 연산자
	>	a > b	
	≤	a ≤ b	
	≥	a ≥ b	
	instanceof	a instanceof b	instanceof 연산자
8	==	a == b	등가 연산자
	!=	a != b	
9	&	a & b	비트 AND 연산자
10	^	a ^ b	비트 XOR 연산자
11		a   b	비트 OR 연산자
12	&&	a && b	AND 연산자
13		a    b	OR 연산자
14	?:	a ? b : c	삼항 연산자
15	=	a = b	배정 연산자
	*=, /=...,	a *= b, a /= b, ...	단축 배정 연산자(11종)



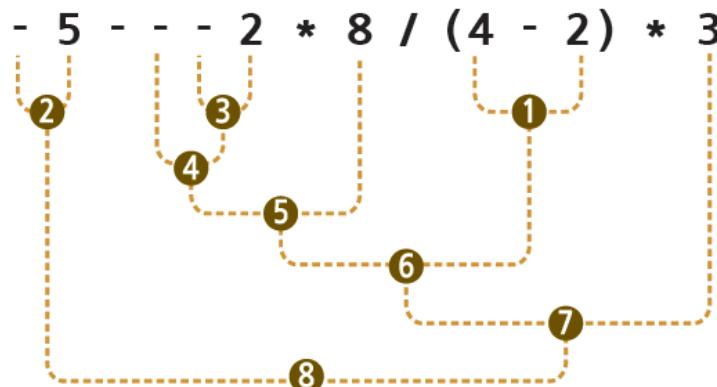
## 6 연산자의 우선순위

9<sup>th</sup> edition

다음은 수식에서의 우선순위를 그림으로 나타낸 것입니다.



일반적인 수식에서 우선순위 동일한 경우  
왼쪽부터(좌결합) 계산한다.



괄호는 최우선순위로 수행하고  
단항 연산자를 다음 순위로 수행한다.



a \*= b /= c += d -= 10;

배정 연산자나 단축 배정연산자가 하나의 수식에서 여러 개 사용될 경우 오른쪽부터 (우결합) 계산한다.

a ++ + + a

전위 증가 연산자나 후위 증가 연산자의 공통점은 모두 1을 증가시킨다는 점이다. 우선순위도 같다.  
다만 후위 증가 연산자는 변수의 값을 배정한 다음 1을 증가시키고,  
전위 증가 연산자는 1을 증가시켜 변수의 값을 배정한다는 점이 다르다.  
예를 들어, 변수 a의 값이 10일 경우 첫 번째 수식 a++에서 a에 배정된 값은 10이 되지만 수식의 평가가 끝나면 a는 11이 된다.  
즉, 두 번째 수식이 평가되기 전에 a 값은 이미 11이 된 상태이다.  
그러므로 이 수식이 평가가 종료되면 a의 값은 12가 되고,  
수식의 결과는 10+12가 되어 22가 된다.



## ● 자바의 문자열은 기본 자료형이 아닌 **String** 클래스로 구현되는 참조 자료형

- String 클래스는 12장에서 자세하게 기술(여기서는 문자열 리터럴을 사용하는 방법만 언급)

```
String s1 = "아! 대한민국";  
String s2 = "Korea";  
String str3;  
str3 = "처음 시작하는 자바";  
String s3 = s1 + s2;  
String s4 = "비어 있어야 비로소  
가득해지는 사랑";
```

문자열 변수를 선언하고 초기화

문자열 변수를 선언

선언된 변수에 값을 배정

다른 문자열을 합하여 새로운 문자열 생성

오류 발생. 문자열 중간에 줄을 바꿀 수 없다.



## 7 문자열

### 실행 결과

```
아! 대한민국 Korea
아! 대한민국
Korea
아! 대한민국 10002000리 금수강산
아! 대한민국 3000리 금수강산
3000리 금수강산
3천리 금수강산
3048리 금수강산
```

### 예제 3.27

#### StringTest1.java

```
01: public class StringTest1 {
02:     public static void main(String args[])
03:     {
04:         String str1 = "아! 대한민국 ";           ← 문자열 변수 선언 및 초기화
05:         String str2 = "Korea";                  ←
06:         System.out.println(str1 + str2);        ← 문자열 변수를 결합하여 출력
07:         System.out.println("아! 대한민국 \nKorea"); ← 문자열에 줄 바꿈(\n) 문자를 삽입
08:         int a = 1000;
09:         int b = 2000;                         ← 정수형 변수를 연결하여 출력
10:         System.out.println(str1 + a + b + "리 금수강산 "); ← 정수형 변수의 덧셈을 괄호로
11:         System.out.println(str1 + (a + b) + "리 금수강산 "); ← 우선 수행하여 출력
12:         System.out.println(a + b + "리 금수강산");      ← 정수형 변수의 덧셈이 먼저 나옴
13:         System.out.println('3' + "천리 금수강산");    ← '3'이라는 문자를 문자열과 연결
14:         System.out.println('3' + 2997 + "리 금수강산"); ← '3'이라는 문자를 숫자와 덧셈
15:     }
16: }
```



## 7 문자열

9<sup>th</sup> edition

```
System.out.println(str1 + a + b + "리 금수강산");
```



문자열과 숫자가 +로  
연결되면 숫자는 자동으로  
문자열로 전환된다.

```
System.out.println(str1 + (a + b) + "리 금수강산");
```



괄호에 의해 정수의 연산이  
먼저 수행되어 원하는  
문장이 완성된다.

```
System.out.println( a + b + "리 금수강산");
```



정수의 연산이 먼저  
수행된다.



## 7 문자열

9<sup>th</sup> edition

```
System.out.println('3' + "천리 금수강산");
```



3천리 금수강산

문자가 문자열과 덧셈으로  
연결되면 문자는 자동으로  
문자열로 변환되어  
연결된다.

```
System.out.println('3' + 2997 + "리 금수강산");
```



3048  
3048리 금수강산

문자가 정수와 덧셈으로  
연결되면 문자에 해당하는  
유니코드 값이(3은 51)  
정수에 더해진다.

## ● 식별자 identifier와 예약어 reserved word

- ① 자바에는 50개의 예약어가 있으며, 예약어는 식별자로 사용할 수 없습니다.
- ② 자바는 대소문자를 구분합니다.
- ③ 자바에서의 식별자 사용은 관례가 있으며, 관례에 따라 사용하는 것이 좋습니다.

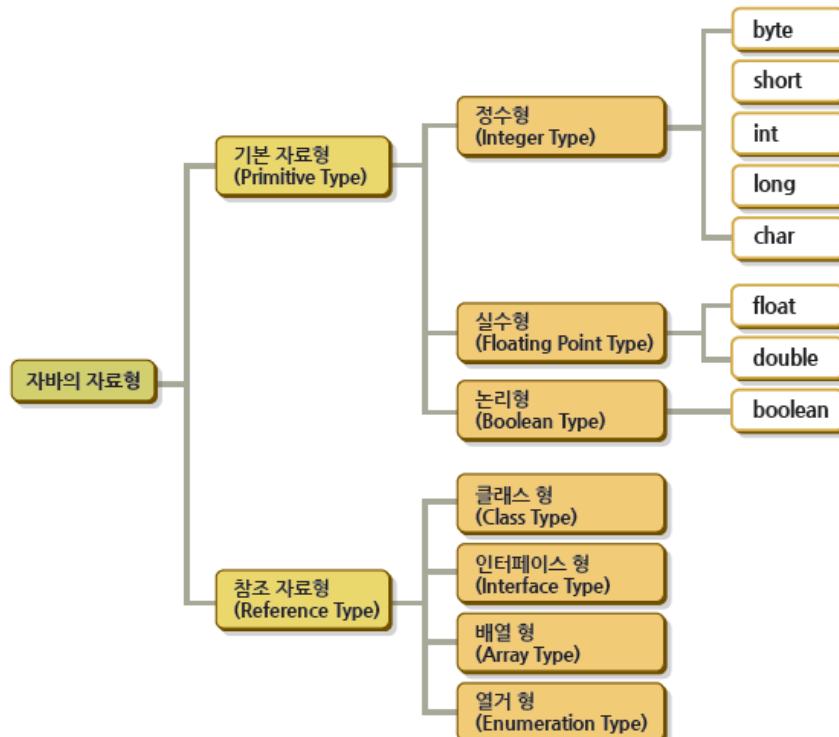
## ● 변수 Variable

- ① 변수는 값이 저장된 메모리에 주어진 이름입니다.
- ② 변수명은 다음과 같은 사용 규칙을 가지고 있습니다.

- 변수명의 첫 글자는 반드시 영문자나 일부 특수 문자(\_,\$)로 시작한다.
- 변수명에는 숫자가 포함될 수 있다.
- 변수명에는 공백이 포함될 수 없다.
- 대소문자를 구분한다. 즉 Sum과 sum은 다른 변수명이다.
- 예약어를 변수명으로 사용할 수 없다.

## ● 자료형 Data type

### ① 자바는 다음과 같은 자료형을 가지고 있습니다





# 학습 정리

9<sup>th</sup> edition

## ● 연산과 형 변환

- ① 자바는 다음과 같은 순서로 확대 형 변환이 이루어집니다.

byte >> short/char >> int >> long >> float >> double

- ② 자바의 축소 형 변환은 명시적인 형 변환 구문에 의해 수행되며, 축소 형 변환의 경우에는 데이터의 손실이 발생할 수 있습니다.

형 변환 구문 : (type) 식 또는 변수

## ● 연산자와 수식, 우선순위

- ① 자바에는 단항 연산자, 이항 연산자, 3항 연산자가 있습니다.

- ② 자바에는 다음과 같은 연산자(우선순위)가 있습니다.



# 학습 정리

9<sup>th</sup> edition

우선순위	연산자	형식	이름
1	[]	a[b]	인덱스 연산자
	()	(a+b)*c	괄호
	.	a.b	멤버 접근 연산자
2	++	++a, a++	증가 연산자
	--	--a, a--	감소 연산자
	+	+a	단항 +연산자
	-	-a	단항 -연산자
	!	!a	NOT 연산자
	~	~a	비트 보수 연산자
	new	new 클래스명	new 연산자
3	()	(short)	캐스트 연산자
	*	a * b	곱셈 연산자
	/	a / b	나눗셈 연산자
4	%	a % b	나머지 연산자
	+	a + b	+ 연산자
	-	a - b	- 연산자

# 학습 정리

9<sup>th</sup> edition

우선순위	연산자	형식	이름
6	<code>&lt;&lt;</code>	<code>a &lt;&lt; b</code>	시프트 연산자
	<code>&gt;&gt;</code>	<code>a &gt;&gt; b</code>	
	<code>&gt;&gt;&gt;</code>	<code>a &gt;&gt;&gt; b</code>	
7	<code>&lt;</code>	<code>a &lt; b</code>	관계 연산자
	<code>&gt;</code>	<code>a &gt; b</code>	
	<code>&lt;=</code>	<code>a &lt;= b</code>	
	<code>&gt;=</code>	<code>a &gt;= b</code>	
	<code>instanceof</code>	<code>a instanceof b</code>	instanceof 연산자
8	<code>==</code>	<code>a == b</code>	등가 연산자
	<code>!=</code>	<code>a != b</code>	
9	<code>&amp;</code>	<code>a &amp; b</code>	비트 AND 연산자
10	<code>^</code>	<code>a ^ b</code>	비트 XOR 연산자
11	<code> </code>	<code>a   b</code>	비트 OR 연산자
12	<code>&amp;&amp;</code>	<code>a &amp;&amp; b</code>	AND 연산자
13	<code>  </code>	<code>a    b</code>	OR 연산자
14	<code>?:</code>	<code>a ? b : c</code>	삼항 연산자
15	<code>=</code>	<code>a = b</code>	배정 연산자
	<code>*=, /=, ...</code>	<code>a *= b, a /= b, ...</code>	단축 배정 연산자(11종)



# 학습 정리

9<sup>th</sup> edition

## ● 문자열 String

- ① 자바에서 문자열은 **String** 클래스로 제공되며, 문자열 변수를 사용할 수 있습니다.
- ② 문자열 변수와 수치 변수가 + 연산자에 의해 결합되면, 수치 변수가 문자열로 자동 변환되어 연결됩니다.