

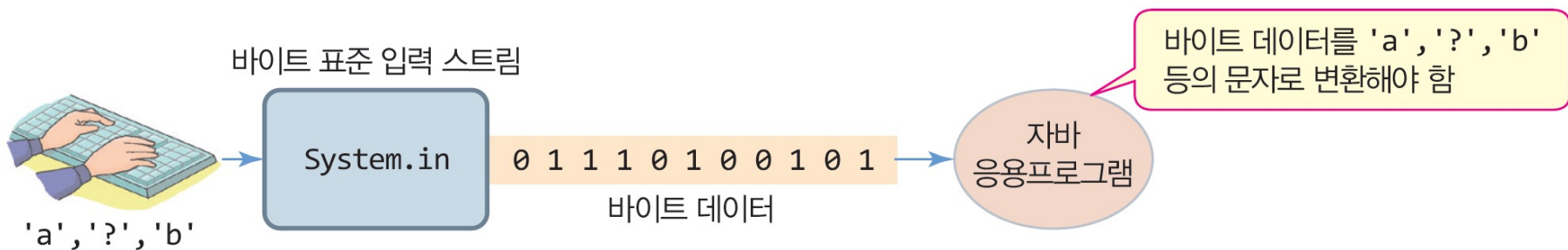
(INTERMEDIATE) JAVA PROGRAMMING

3. Java Basics: Chapter 2

JAVA: INTERACTIVE PROGRAMS

자바에서 키 입력

- System.in
 - 키보드로부터 직접 읽는 자바의 표준 입력 스트림
 - 키 값을 바이트(문자 아님)로 리턴
- System.in을 사용할 때 문제점
 - 키 값을 바이트 데이터로 넘겨주므로 응용프로그램이 문자 정보로 변환해야 함



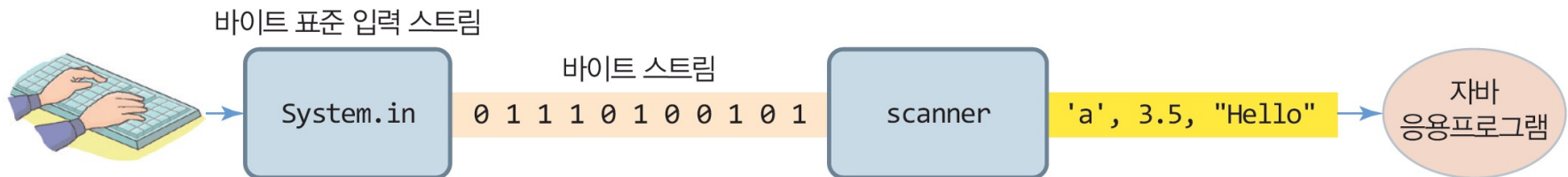
Scanner로 쉽게 키 입력

- Scanner 클래스

- System.in에게 키를 읽게 하고, 읽은 바이트를 문자, 정수, 실수, 불린, 문자열 등 다양한 타입으로 변환하여 리턴
 - java.util.Scanner 클래스

- 객체 생성

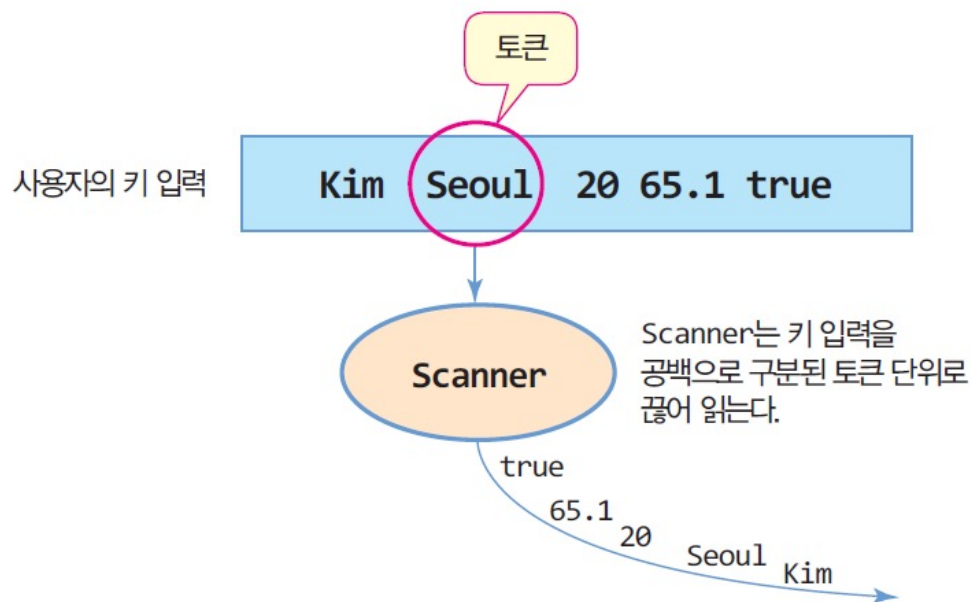
```
import java.util.Scanner; // import 문 필요
...
Scanner a = new Scanner(System.in); // Scanner 객체 생성
```



- System.in에게 키를 읽게 하고, 원하는 타입으로 변환하여 리턴

Scanner를 이용한 키 입력

- Scanner에서 키 입력 받기
 - Scanner는 입력되는 키 값을 공백으로 구분되는 아이템 단위로 읽음
 - 공백 문자 : '\t', '\f', '\r', '\n', ' '
- 개발자가 원하는 다양한 타입의 값으로 바꾸어 읽을 수 있음



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

```
String name = scanner.next();           // "Kim"
String city = scanner.next();           // "Seoul"
int age = scanner.nextInt();            // 20
double weight = scanner.nextDouble();   // 65.1
boolean single = scanner.nextBoolean(); // true
```

Scanner 주요 메소드

메소드	설명
<code>String next()</code>	다음 토큰을 문자열로 리턴
<code>byte nextByte()</code>	다음 토큰을 byte 타입으로 리턴
<code>short nextShort()</code>	다음 토큰을 short 타입으로 리턴
<code>int nextInt()</code>	다음 토큰을 int 타입으로 리턴
<code>long nextLong()</code>	다음 토큰을 long 타입으로 리턴
<code>float nextFloat()</code>	다음 토큰을 float 타입으로 리턴
<code>double nextDouble()</code>	다음 토큰을 double 타입으로 리턴
<code>boolean nextBoolean()</code>	다음 토큰을 boolean 타입으로 리턴
<code>String nextLine()</code>	'\n'을 포함하는 한 라인을 읽고 '\n'을 버린 나머지 문자열 리턴
<code>void close()</code>	Scanner의 사용 종료
<code>boolean hasNext()</code>	현재 입력된 토큰이 있으면 true, 아니면 입력 때까지 무한정 대기, 새로운 입력이 들어올 때 true 리턴. ctrl-z 키가 입력되면 입력 끝이므로 false 리턴

예제 2-4 : Scanner를 이용한 키 입력 연습

Scanner를 이용하여
이름, 도시, 나이, 체중,
독신 여부를 입력 받고
다시 출력하는
프로그램을 작성하라.

```
import java.util.Scanner;

public class ScannerEx {
    public static void main(String args[]) {
        System.out.println("이름, 도시, 나이, 체중, 독신 여부를 빈칸으로 분리하여 입력하세요");
        Scanner scanner = new Scanner(System.in);

        String name = scanner.next(); // 문자열 읽기
        System.out.print("이름은 " + name + ", ");

        String city = scanner.next(); // 문자열 읽기
        System.out.print("도시는 " + city + ", ");

        int age = scanner.nextInt(); // 정수 읽기
        System.out.print("나이는 " + age + "살, ");

        double weight = scanner.nextDouble(); // 실수 읽기
        System.out.print("체중은 " + weight + "kg, ");

        boolean single = scanner.nextBoolean(); // 논리값 읽기
        System.out.println("독신 여부는 " + single + "입니다.");

        scanner.close(); // scanner 닫기
    }
}
```

이름, 도시, 나이, 체중, 독신 여부를 빈칸으로 분리하여 입력하세요.

Kim Seoul 20 65.1 true

이름은 Kim, 도시는 Seoul, 나이는 20살, 체중은 65.1kg, 독신 여부는 true입니다.

Eclipse 단축키

- Ctrl+Shift+L: 단축키 설명 전부 보기
- Ctrl+Space : 자동 완성
- Ctrl+1 : Quick Fix
- Ctrl+i : 들여쓰기 자동 수정
- F3 : 메소드 위에서 누르면 해당 메소드 정의로 이동
- Ctrl+Alt+위(아래)화살표 : 라인복사
- Ctrl+Shift+O : 필요한 클래스 자동 import
- F11 : Run in debugging mode
- Ctrl+F11 : Run
- <https://dzone.com/articles/effective-eclipse-shortcut-key>
- <http://yeonicon.tistory.com/657>

Add 예제

- 사용자로부터 두 개의 정수를 받아서 더하는 문제

실행결과

```
첫 번째 숫자를 입력하시오: 10  
두 번째 숫자를 입력하시오: 20  
30
```

- 사용자로부터 숫자를 받을 수 있어야 한다!

Add 예제

Add2.java

```
01 // 사용자가 입력한 두 개의 숫자를 더해서 출력한다.
02 import java.util.Scanner; // Scanner 클래스 포함
03
04 public class Add2 {
05     // 메인 메소드에서부터 실행이 시작된다.
06     public static void main(String args[]) {
07
08         Scanner input = new Scanner(System.in); ← 사용자로부터 입력을 받기 위해
09         int x; // 첫 번째 숫자 저장                               Scanner 객체를 생성한다.
10         int y; // 두 번째 숫자 저장
11         int sum; // 합을 저장
12
13         System.out.print("첫 번째 숫자를 입력하십시오: "); // 입력 안내 출력
14         x = input.nextInt(); // 사용자로부터 첫 번째 숫자를 읽는다.
15
16         System.out.print("두 번째 숫자를 입력하십시오: "); // 입력 안내 출력
17         y = input.nextInt(); // 사용자로부터 두 번째 숫자를 읽는다.
18
19         sum = x + y; // 두 개의 숫자를 더한다.
20
21         System.out.println(sum); // 합을 출력한다.
22
23     } // 메인 메소드의 끝
24
25 } // Add 클래스의 끝
```

import 문장

```
import java.util.Scanner; // Scanner 클래스 포함
```

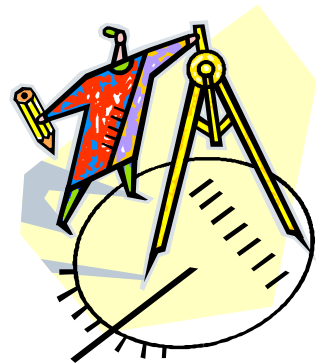
- Scanner 클래스를 포함시키는 문장
- Scanner는 자바 클래스 라이브러리(Java Class Library)의 일종
- Scanner는 입력을 받을 때 사용

중간 점검 문제

1. 사용자로부터 값을 입력받으려면 _____ 클래스를 사용하는 것이 편리하다.
2. Scanner 클래스에서 사용자로부터 정수를 입력받는 메소드의 이름은 _____이다.
3. 자바 API 문서에서 Scanner 클래스의 메소드 중에서 앞에 next가 붙은 메소드들을 조사하여 보자.

원의 면적 구하기

- 사용자로부터 원의 반지름을 입력받고 이 원의 면적을 구한 다음, 화면에 출력한다.



실행결과

반지름을 입력하시오: 5

78.5

원의 면적 구하기

CircleArea.java

```
01  import java.util.Scanner; // 프로그램은 스캐너 클래스를 사용한다.
02
03  public class CircleArea {
04      public static void main(String args[]) {
05
06          double radius; // 원의 반지름
07          double area;   // 원의 면적
08          Scanner input = new Scanner(System.in);
09          System.out.print("반지름을 입력하시오: "); // 입력 안내 출력
10          radius = input.nextDouble();
11          area = 3.14 * radius * radius;
12
13          System.out.println(area);
14      }
15
16  }
```

실수값을 입력받아서
radius에 저장한다.

JAVA: EXPRESSIONS

Expressions

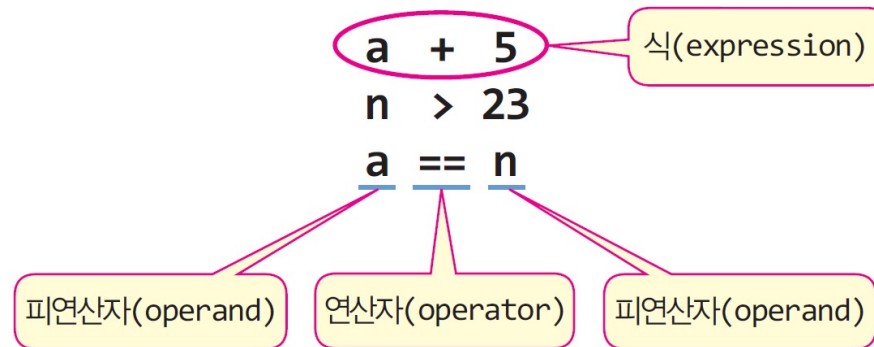
- An *expression* is a combination of one or more operators and operands
- *Arithmetic expressions* compute numeric results and make use of the arithmetic operators:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Remainder	%

- If either or both operands used by an arithmetic operator are floating point, then the result is a floating point

식과 연산자

- 연산 : 주어진 식을 계산하여 결과를 얻어내는 과정



연산의 종류	연산자	연산의 종류	연산자
증감	++ --	비트	& ^ ~
산술	+ - * / %	논리	&& ! ^
시프트	>> << >>>	조건	? :
비교	> < >= <= == !=	대입	= *= /= += -= &= ^= = <<= >>= >>>=

Division and Remainder

- If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

14 / 3 equals 4

8 / 12 equals 0

- The remainder operator (%) returns the remainder after dividing the second operand into the first

14 % 3 equals 2

8 % 12 equals 8

예제 2-5 : /와 % 산술 연산

초 단위의 정수를 입력받고, 몇 시간, 몇 분, 몇 초인지 출력하는 프로그램을 작성하라.

```
import java.util.Scanner;

public class ArithmeticOperator {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("정수를 입력하세요: ");
        int time = scanner.nextInt();    // 정수 입력
        int second = time % 60;          // 60으로 나눈 나머지는 초
        int minute = (time / 60) % 60;   // 60으로 나눈 몫을 다시 60으로 나눈 나머지는 분
        int hour = (time / 60) / 60;     // 60으로 나눈 몫을 다시 60으로 나눈 몫은 시간

        System.out.print(time + "초는 ");
        System.out.print(hour + "시간, ");
        System.out.print(minute + "분, ");
        System.out.println(second + "초입니다.");

        scanner.close();
    }
}
```

정수를 입력하세요:5000
5000초는 1시간, 23분, 20초입니다.

예제

- 문자열에 + 연산자는 결합(concatenation)을 수행한다.

```
public class StringOperator {  
  
    public static void main(String[] args) {  
        String s1 = "Hello";  
        String s2 = " World";  
        String s3 = s1 + s2;  
        System.out.println(s3);  
    }  
}
```




Operator Precedence

- Operators can be combined into complex expressions

```
result = total + count / max - offset;
```

- Operators have a well-defined precedence which determines the order in which they are evaluated
- Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation
- Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

연산자 우선순위

<div>높음</div>  <div>낮음</div>	++(postfix) --(postfix)
	+(양수 부호) -(음수 부호) ++(prefix) --(prefix) ~ !
	형 변환(type casting)
	* / %
	+(덧셈) -(뺄셈)
	<< >> >>>
	<> <= >= instanceof
	== !=
	& (비트 AND)
	^ (비트 XOR)
	(비트 OR)
	&& (논리 AND)
	(논리 OR)
	? : (조건)
	= += -= *= /= %= &= ^= = <<= >>= >>>=

주의!

- 같은 우선순위의 연산자
 - 왼쪽에서 오른쪽으로 처리
 - 예외) 오른쪽에서 왼쪽으로
 - 대입 연산자, --, ++, +, -(양수 음수 부호), !, 형 변환은 오른쪽에서 왼쪽으로 처리
- 괄호는 최우선순위
 - 괄호가 다시 괄호를 포함한 경우는 가장 안쪽의 괄호부터 먼저 처리

Operator Precedence

- What is the order of evaluation in the following expressions?

$$a + b + c + d + e$$

1 2 3 4

$$a + b * c - d / e$$

3 1 4 2

$$a / (b + c) - d \% e$$

2 1 4 3

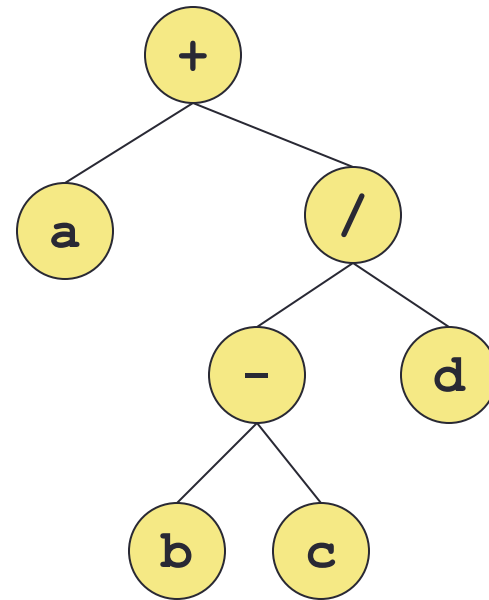
$$a / (b * (c + (d - e)))$$

4 3 2 1

Expression Trees

- The evaluation of a particular expression can be shown using an *expression tree*
- The operators lower in the tree have higher precedence for that expression

$a + (b - c) / d$



Assignment Revisited

- The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer = sum / 4 + MAX * lowest;
```

 4 1 3 2



Then the result is stored in the variable on the left hand side

Assignment Revisited

- The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the
original value of count

```
count = count + 1;
```



Then the result is stored back into count
(overwriting the original value)

단항 연산자

연산자	의미
$+x$	x 를 양수로 만든다.
$-x$	x 를 음수로 만든다.
$++x$	x 값을 먼저 증가한 후에 다른 연산에 사용한다. 이 수식의 값은 증가된 x 값이다.
$x++$	x 값을 먼저 사용한 후에, 증가한다. 이 수식의 값은 증가되지 않은 원래의 x 값이다.
$--x$	x 값을 먼저 감소한 후에 다른 연산에 사용한다. 이 수식의 값은 감소된 x 값이다.
$x--$	x 값을 먼저 사용한 후에, 감소한다. 이 수식의 값은 감소되지 않은 원래의 x 값이다.

Increment and Decrement

- The increment and decrement operators use only one operand
- The *increment operator* (**++**) adds one to its operand
- The *decrement operator* (**--**) subtracts one from its operand
- The statement

```
count++;
```

is functionally equivalent to

```
count = count + 1;
```

Increment and Decrement

- The increment and decrement operators can be applied in *postfix form*:

`count++`

- or *prefix form*:

`++count`

- When used as part of a larger expression, the two forms can have different effects
- Because of their subtleties, the increment and decrement operators should be used with care

Assignment Operators

- Often we perform an operation on a variable, and then store the result back into that variable
- Java provides *assignment operators* to simplify that process
- For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

- There are many assignment operators in Java, including the following:

<u>Operator</u>	<u>Example</u>	<u>Equivalent To</u>
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Assignment Operators

- The right hand side of an assignment operator can be a complex expression
- The entire right-hand expression is evaluated first, then the result is combined with the original variable
- Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```


예제 2-6 : 대입 연산자와 증감 연산자 사용

다음 코드의 실행 결과는 무엇인가?

```
public class AssignmentIncDecOperator {
    public static void main(String[] args) {
        int a=3, b=3, c=3;

        // 대입 연산자 사례
        a += 3;    // a=a+3 = 6
        b *= 3;    // b=b*3 = 9
        c %= 2;    // c=c%2 = 1
        System.out.println("a=" + a + ", b=" + b + ", c=" + c);

        int d=3;
        // 증감 연산자 사례
        a = d++; // a=3, d=4
        System.out.println("a=" + a + ", d=" + d);
        a = ++d; // d=5, a=5
        System.out.println("a=" + a + ", d=" + d);
        a = d--; // a=5, d=4
        System.out.println("a=" + a + ", d=" + d);
        a = --d; // d=3, a=3
        System.out.println("a=" + a + ", d=" + d);
    }
}
```

```
a=6, b=9, c=1
a=3, d=4
a=5, d=5
a=5, d=4
a=3, d=3
```

비교 연산과 논리 연산

• 비교 연산

- 두 피연산자를 비교하여 true 또는 false의 논리 값을 내는 연산

연산자	내용	예제	결과
$a < b$	a가 b보다 작으면 true	$3 < 5$	true
$a > b$	a가 b보다 크면 true	$3 > 5$	false
$a \leq b$	a가 b보다 작거나 같으면 true	$1 \leq 0$	false
$a \geq b$	a가 b보다 크거나 같으면 true	$10 \geq 10$	true
$a == b$	a가 b와 같으면 true	$1 == 3$	false
$a != b$	a가 b와 같지 않으면 true	$1 != 3$	true

• 논리 연산

- 논리 값으로 NOT, OR, AND 논리 연산. 논리 값을 내는 연산

연산자	내용	예제	결과
$!a$	a가 true이면 false, false이면 true	$!(3 < 5)$	false
$a \parallel b$	a와 b의 OR 연산. a와 b 모두 false인 경우에만 false	$(3 > 5) \parallel (1 == 1)$	true
$a \&\& b$	a와 b의 AND 연산. a와 b 모두 true인 경우에만 true	$(3 < 5) \&\& (1 == 1)$	true

비교 연산과 논리 연산의 복합 사례

```
// 나이(int age)가 20대인 경우  
(age >= 20) && (age < 30)
```

```
// 문자(char c)가 대문자인 경우  
(c >= 'A') && (c <= 'Z')
```

```
// (x,y)가 (0,0)과 (50,50)의 사각형 내에 있음  
(x>=0) && (y>=0) && (x<=50) && (y<=50)
```

```
20 <= age < 30    // 오류
```

예제 2-7 : 비교 연산자와 논리 연산자 사용하기

다음 소스의 실행 결과는 무엇인가?

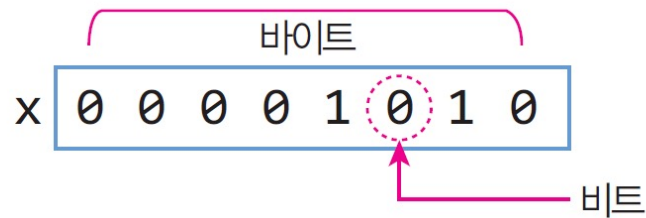
```
public class LogicalOperator {  
    public static void main (String[] args) {  
        // 비교 연산  
        System.out.println('a' > 'b');  
        System.out.println(3 >= 2);  
        System.out.println(-1 < 0);  
        System.out.println(3.45 <= 2);  
        System.out.println(3 == 2);  
        System.out.println(3 != 2);  
        System.out.println(!(3 != 2));  
  
        // 비교 연산과 논리 연산 복합  
        System.out.println((3 > 2) && (3 > 4));  
        System.out.println((3 != 2) || (-1 > 0));  
    }  
}
```

false
true
true
false
false
true
false
false
true

비트 연산

• 비트 개념

byte x = 10;



• 비트 연산

- 비트 논리 연산
 - 비트끼리 AND, OR, XOR, NOT 연산
- 비트 시프트 연산
 - 비트를 오른쪽이나 왼쪽으로 이동

비트 논리 연산

- 피 연산자의
각 비트들의
논리 연산

$$\begin{array}{r} 01101010 \\ \& 11001101 \\ \hline 01001000 \end{array}$$

모두 1이므로
결과는 1

둘 중 하나라도
0이면 결과는 0

$$\begin{array}{r} 01101010 \\ | 11001101 \\ \hline 11101111 \end{array}$$

모두 0이므로
결과는 0

둘 중 하나라도
1이면 결과는 1

$$\begin{array}{r} 01101010 \\ ^ 11001101 \\ \hline 10100111 \end{array}$$

두 비트가 같으므로
결과는 0

두 비트가 다르므로
결과는 1

$$\begin{array}{r} \sim 01101010 \\ \hline 10010101 \end{array}$$

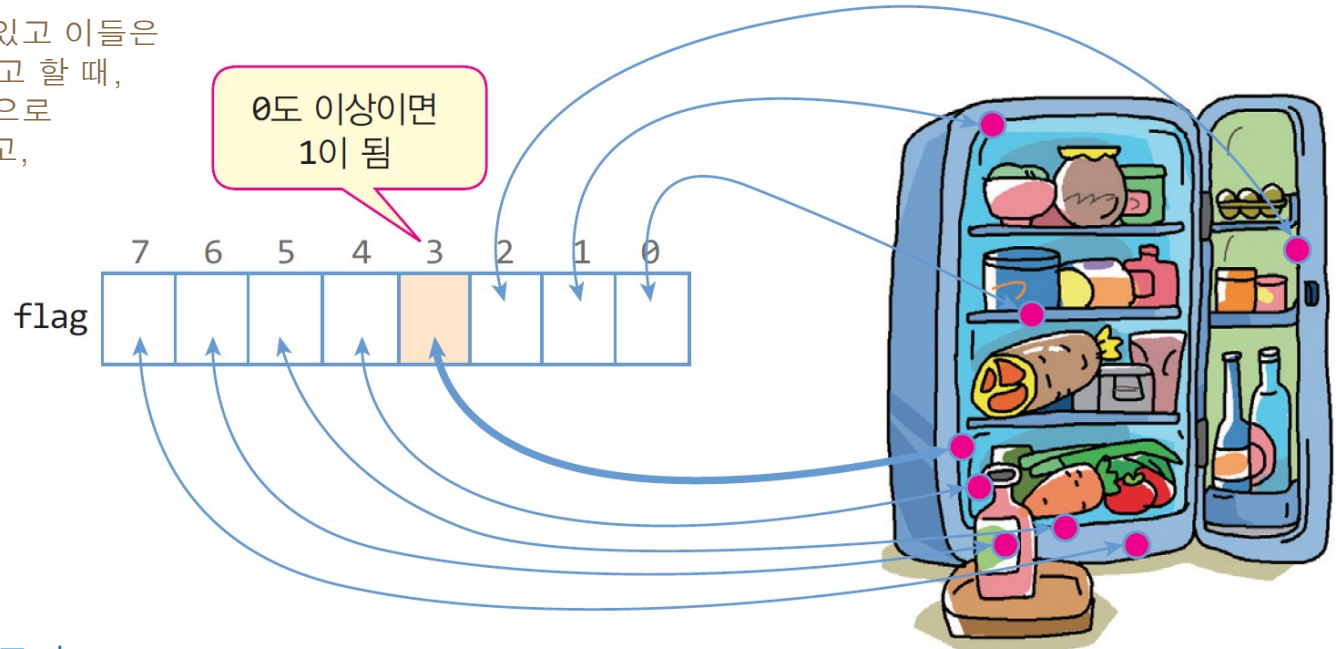
1은 0으로 변환

0은 1로 변환

연산자	별칭	내용
a & b	AND 연산	두 비트 모두 1이면 1, 그렇지 않으면 0
a b	OR 연산	두 비트 모두 0이면 0, 그렇지 않으면 1
a ^ b	XOR 연산	두 비트가 다르면 1, 같으면 0
~ a	NOT 연산	1을 0으로, 0을 1로 변환

비트 논리 연산 응용

냉장고에는 8개의 센서가 있고 이들은 flag 변수와 연결되어 있다고 할 때, 냉장고의 온도가 0도 이상으로 올라가면 비트 3이 1이 되고, 0도 이하이면 비트 3이 0을 유지한다.



문제) 현재 냉장고의 온도가 0도 이상인지 판단하는 코드를 작성하라.

```
byte flag = 0b00001010; // 각 비트는 8개의 센서 값을 가리킴
if((flag & 0b00001000) == 0)
    System.out.print("온도는 0도 이하");
else
    System.out.print("온도는 0도 이상");
```

온도는 0도 이상

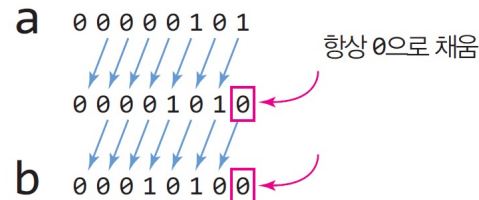
	x	x	x	x	Y	x	x	x
&	0	0	0	0	1	0	0	0
	0	0	0	0	Y	0	0	0

Y비트가 0 이면
& 결과는 0

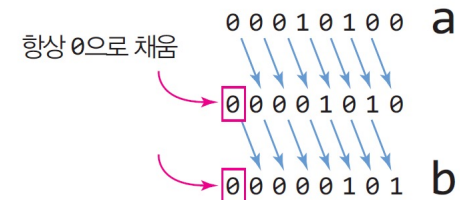
시프트 연산자의 사례

- 피 연산자의 비트들을 이동 연산

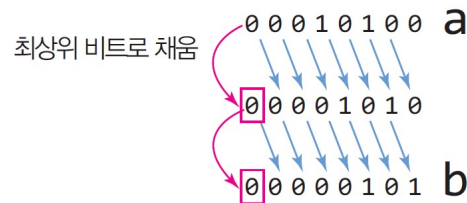
```
byte a = 5; // 5
byte b = (byte)(a << 2); // 20
```



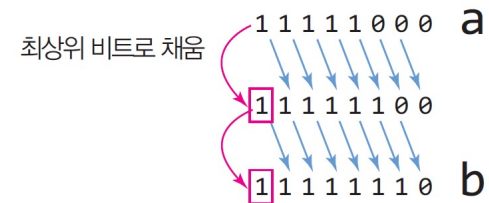
```
byte a = 20; // 20
byte b = (byte)(a >>> 2); // 5
```



```
byte a = 20; // 20
byte b = (byte)(a >> 2); // 5
```



```
byte a = (byte)0xf8; // -8
byte b = (byte)(a >> 2); // -2
```



시프트 연산자	내용
a >> b	a의 각 비트를 오른쪽으로 b번 시프트한다. 최상위 비트의 빈자리는 시프트 전의 최상위 비트로 다시 채운다. 산술적 오른쪽 시프트라고 한다.
a >>> b	a의 각 비트를 오른쪽으로 b번 시프트한다. 최상위 비트의 빈자리는 항상 0으로 채운다. 논리적 오른쪽 시프트라고 한다.
a << b	a의 각 비트를 왼쪽으로 b번 시프트한다. 최하위 비트의 빈자리는 항상 0으로 채운다. 산술적 왼쪽 시프트라고 한다.

예제 2-9 : 비트 논리 연산과 비트 시프트

연산

다음 소스의 실행 결과는 무엇인가?

```
public class BitOperator {
    public static void main(String[] args) {
        short a = (short)0x55ff;
        short b = (short)0x00ff;

        // 비트 논리 연산
        System.out.println("[비트 연산 결과]");
        System.out.printf("%04x\n", (short)(a & b)); // 비트 AND
        System.out.printf("%04x\n", (short)(a | b)); // 비트 OR
        System.out.printf("%04x\n", (short)(a ^ b)); // 비트 XOR
        System.out.printf("%04x\n", (short)(~a)); // 비트 NOT

        byte c = 20; // 0x14
        byte d = -8; // 0xf8

        // 비트 시프트 연산
        System.out.println("[시프트 연산 결과]");
        System.out.println(c <<2); // c를 2비트 왼쪽 시프트
        System.out.println(c >>2); // c를 2비트 오른쪽 시프트. 0 삽입
        System.out.println(d >>2); // d를 2비트 오른쪽 시프트. 1 삽입
        System.out.printf("%x\n", (d >>>2)); // d를 2비트 오른쪽 시프트. 0 삽입
    }
}
```

printf("%x\n", ...)는 결과 값을 16진수 형식으로 출력

[비트 연산 결과]
 00ff
 55ff
 5500
 aa00
 [시프트 연산 결과]
 80
 5
 -2
 3ffffffe