2장
자바 프로그래밍 기초

박숙영
blue@sookmyung.ac.kr

2장의 목표

1. 자바를 이용하여 콘솔 입출력하는 프로그램을 작성할 수 있나요?
2. 자바의 기본 자료형인 int, double,... 등을 사용하여서 프로그램을 작성할 수 있나요?
3. 자바의 각종 연산자를 사용할 수 있나요?
4. 반지름이 20cm 피자 2개와 30cm 피자 1개의 면적을 비교하는 프로그램을 작성할 수 있나요?

자바 프로그램 구성 요소

여러 차이점 : 한 프로젝트에 여러개의 main 존재 가능.
(class 여러 main이 여러 있어도 됨)

- 이번 장에서는 자바 프로그램을 구성하는 여러 가지 요소들을 살펴보자. 이어서 자바가 지원하는 여러 가지 자료형에 대하여 학습한다.

- Src

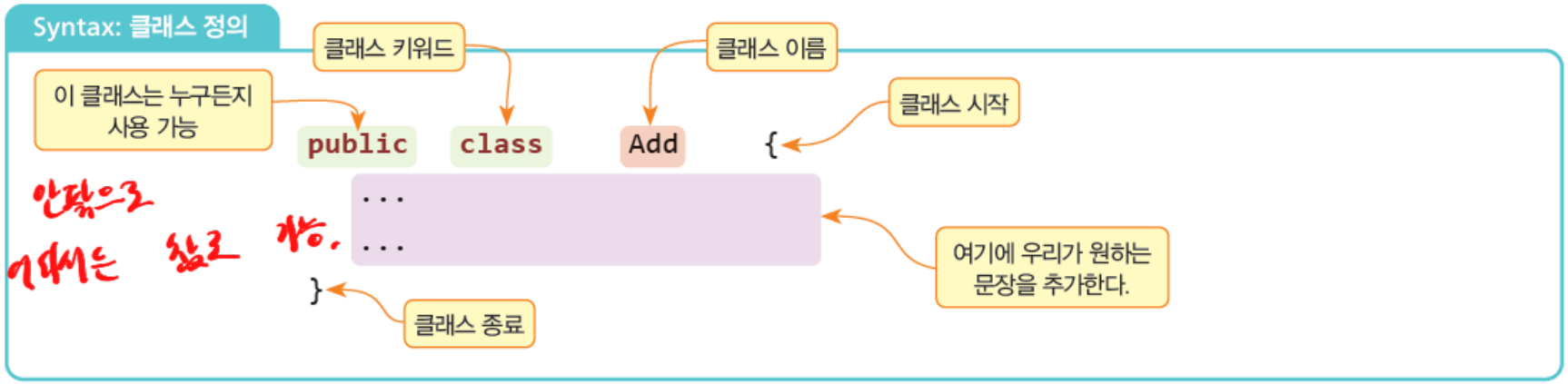
```
/* 덧셈 프로그램 */                                    // ①  주석 (남 똑같다.
                                                       (변수명)
public class Add                                       // ②  클래스 정의
                                                       이게 에서는 모두 main 선언할 때 바뀔수 있음.
{
    public static void main(String[] args) {           // ③  메소드 정의
        int x, y, sum;                                 // ④  변수 선언
        x = 100;                                       // ⑤  대입(할당) 연산
        y = 200;

        sum = x + y;
        System.out.println(sum);                       // ⑥  출력문
    }
}
```

- 실행 결과

```
300
```

클래스

- 클래스(class)는 자바와 같은 객체 지향 언어의 기본적인 빌딩 블록



- 자바에서 소스 파일 이름은 항상 public이 붙은 클래스의 이름과 동일하여야 한다.
  - 위의 소스 파일이름은 반드시 Add.java

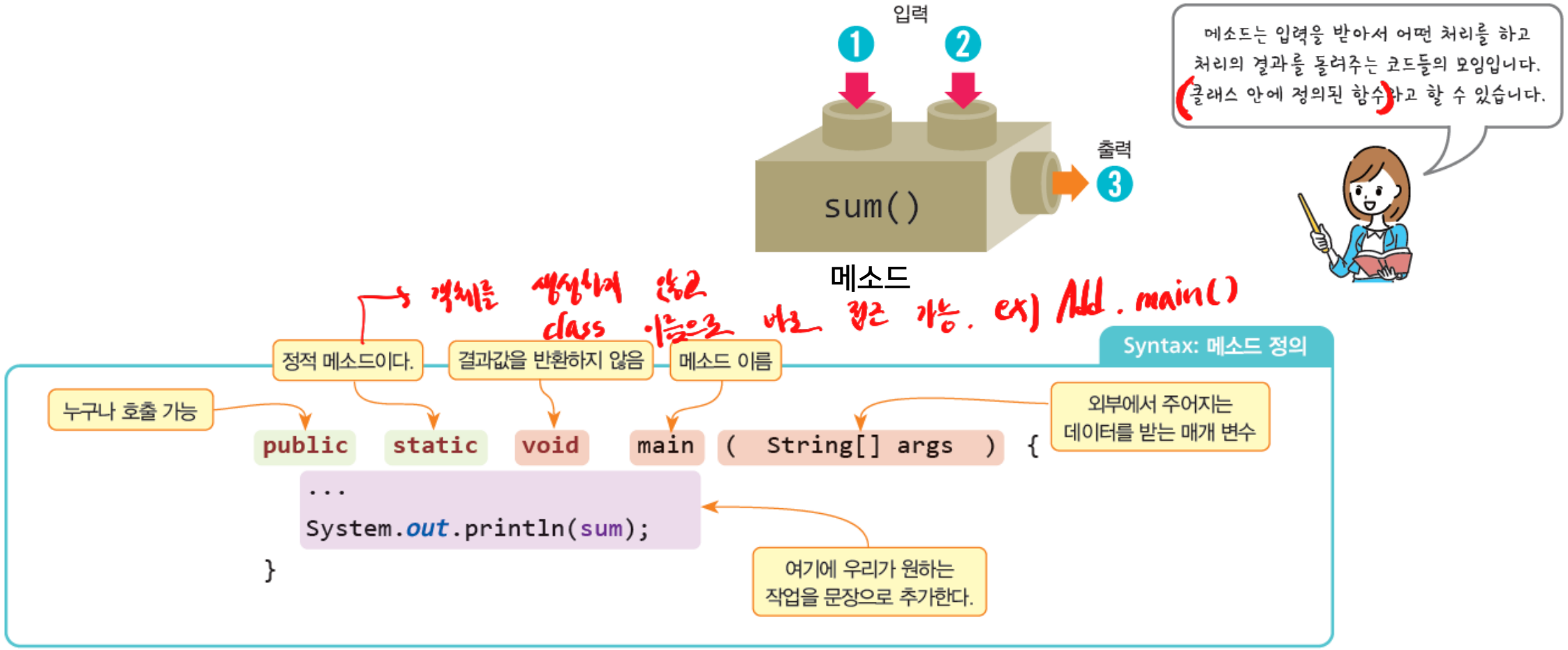
참고

자바에서는 소스 파일 이름과 클래스 이름이 상당한 관련이 있다. 일단 하나의 소스 파일 안에는 하나의 클래스만 있는 것이 바람직하다. 하지만 하나의 소스 파일에는 여러 개의 클래스가 들어 있을 수 있다.

- 소스 안에 public 클래스가 하나 있다면 반드시 소스 파일의 이름은 public 클래스의 이름과 일치하여야 한다. 다른 클래스는 public 클래스가 아니어야 한다.
- 만약 하나의 소스 파일 안에 public 클래스가 없다면, 소스 파일 안에 포함된 어떤 클래스의 이름으로 하여도 상관없다.
- 하나의 소스 파일 안에 public 클래스가 2개 이상 있으면 컴파일 오류가 발생한다.

메소드

- 메소드(method)는 특정한 작업을 수행하는 코드의 묶음

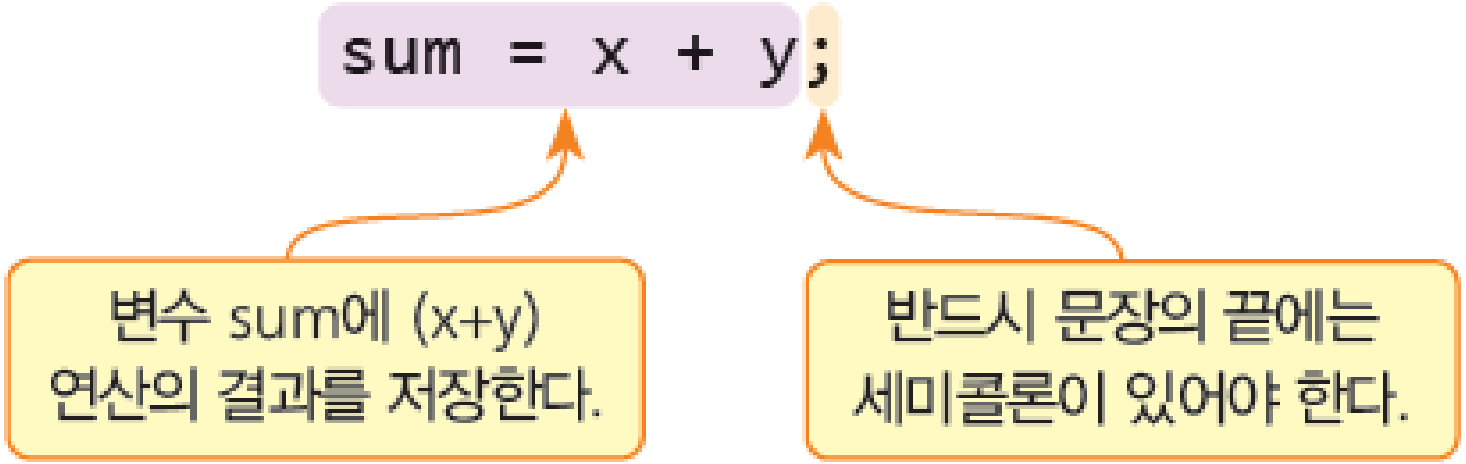


메소드 호출

```
public class Add2
{
    public static void main(String[] args) {
        ...
        z = addition(100, 200);
        ...
    }
    public static int addition(int x, int y) {
        int sum = x + y;
        return sum;
    }
}
```

문장

- 문장(statement)은 사용자가 컴퓨터에게 작업을 지시하는 단위
- 프로그램을 이루는 가장 기초적인 단위

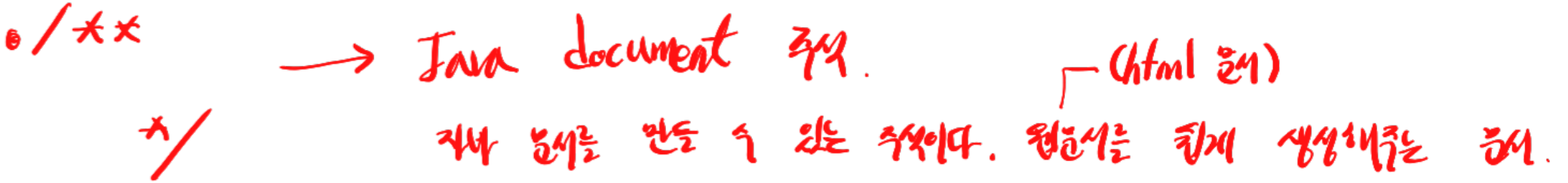


주석

- 주석(comment)은 소스 코드가 하는 일을 설명하는 글로서 프로그램의 실행 결과에 영향을 끼치지 않는다.

- /* text */
  - 주석의 시작과 끝을 /* 와 */로 표시한다. 여러 줄을 주석 처리할 때는 이 방법을 사용한다.

- // text
  - //에서 줄의 끝까지가 주석이다. 한 줄짜리 주석만 가능하다.



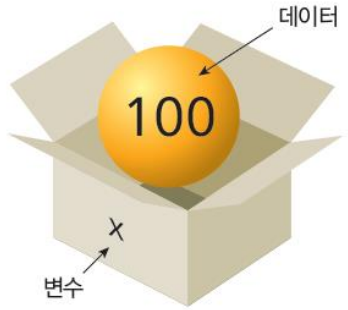
변수와 자료형

- 변수(variable)는 데이터를 담아두는 상자로 생각할 수 있다.

Syntax: 변수 정의

```
int x;
```

변수이름



식별자 만들기 (변수명 · method명, class 이름 ..)

- 알파벳 문자와 숫자, 밑줄 문자 _로 이루어진다. 한글 이름도 가능하다.
- 첫 번째 문자는 반드시 알파벳 또는 밑줄 문자 _이여야 한다. 숫자로 시작할 수 없다.
- '%', '&', '#'와 같은 특수 문자는 사용할 수 없다. 단 '$'와 '_'은 가능하다.
  → inner class의 존재 때문!
- 대문자와 소문자를 구별한다.
  - 따라서 변수 index와 Index, INDEX은 모두 서로 다른 변수이다. ==C
- 자바 언어 키워드(if, while, true, false, null,...)와 똑같은 이름은 허용되지 않는다.

자바 키워드

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

그림 2.2 자바에서의 키워드

식별자의 예

- 사용 가능 식별자

```
int     sum;
long    employee_id;          // '_' 사용가능
class   Sprite10 {  }         // 숫자 사용 가능
void    get_$() {  }          // '$' 문자 사용 가능
```

- 사용 불가능한 식별자

```
int     1stPrizeMoney;        // 첫 글자가 숫자
double  super;                // 키워드
int     #ofComputer;          // 허용되지 않는 기호
class   %_of_Money {    }     // 허용되지 않는 기호
```

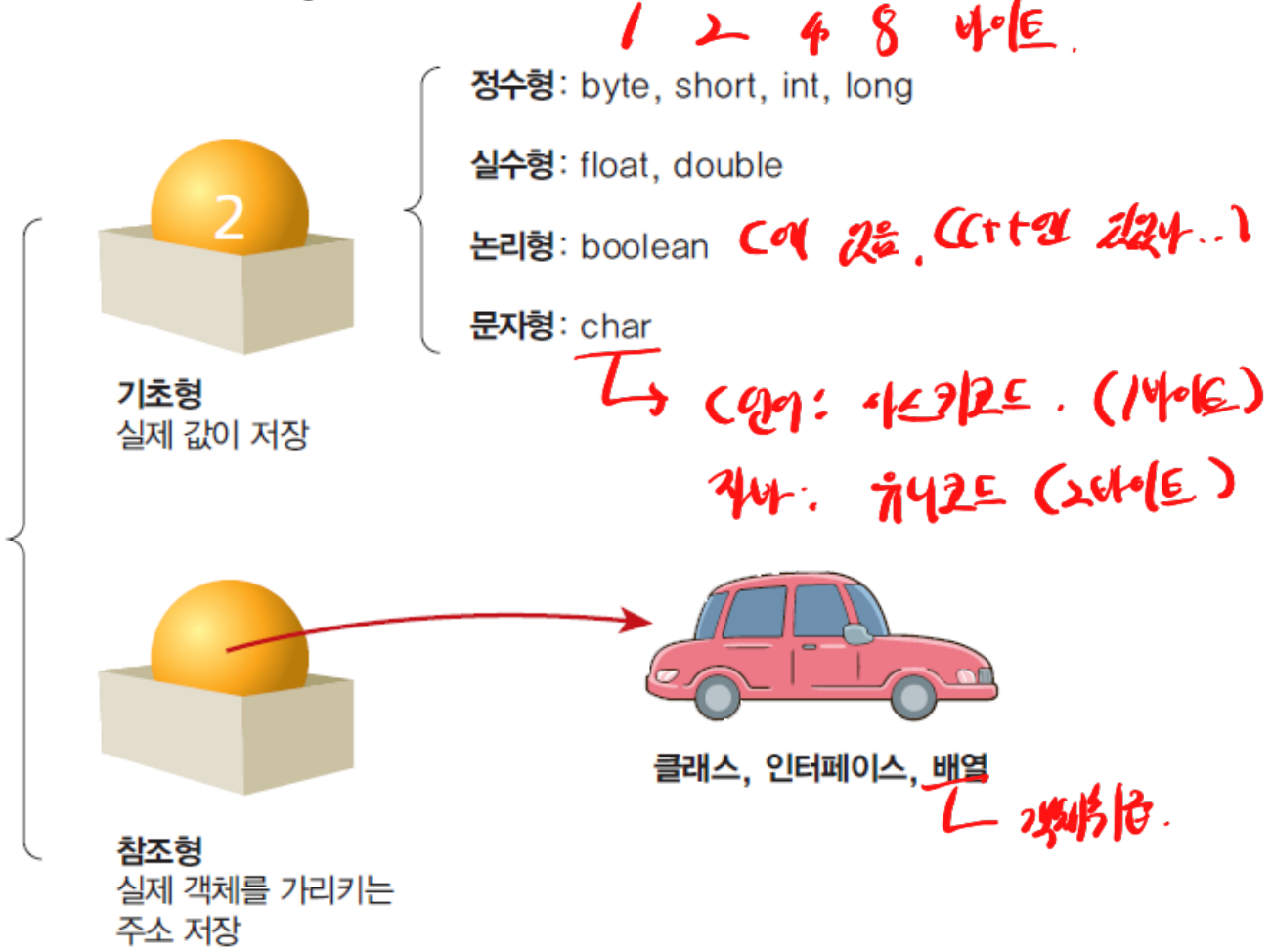
식별자의 관례

표 2.1 식별자의 관례

종류	사용 방법	예
클래스명	각 단어의 첫 글자는 대문자로 한다.	StaffMember, ItemProducer
변수명, 메소드명	첫 단어의 첫글자는 소문자로 시작하고 두 번째 단어부터는 단어의 첫 글자를 대문자로 한다.	width, payRate, acctNumber, getMonthDays(), fillRect(), = 카멜표기법.
상수	상수는 모든 글자를 대문자로 한다.	MAX_NUMBER 값은 변할 수 없음.

자료형

- 자료형(data type)은 변수에 저장되는 데이터의 타입을 의미



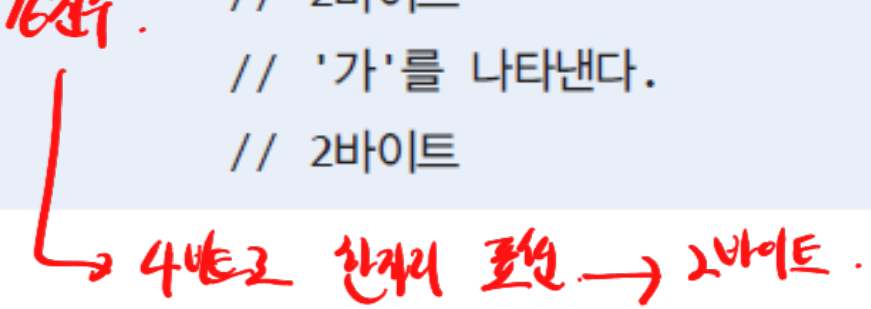
기초형

자료형	설명	크기(바이트)	범위
byte	부호있는 정수	1바이트	−128에서 127
short	부호있는 정수	2바이트	−32768에서 32767
int	부호있는 정수	4바이트	−2147483648에서 2147483647(20억 정도)
long	부호있는 정수	8바이트	−9223372036854775808에서 9223372036854775807
float	부동소수점형	4바이트	약 ± 3.40282347 × 10^{+38}(유효숫자 6-7개 정도)
double	부동소수점형	8바이트	약 ± 1.7976931 × 10^{+308}(유효숫자 15개 정도)
char	문자형	2바이트	\u0000에서 \uFFFF
boolean	논리형	1비트	true, false

문자형

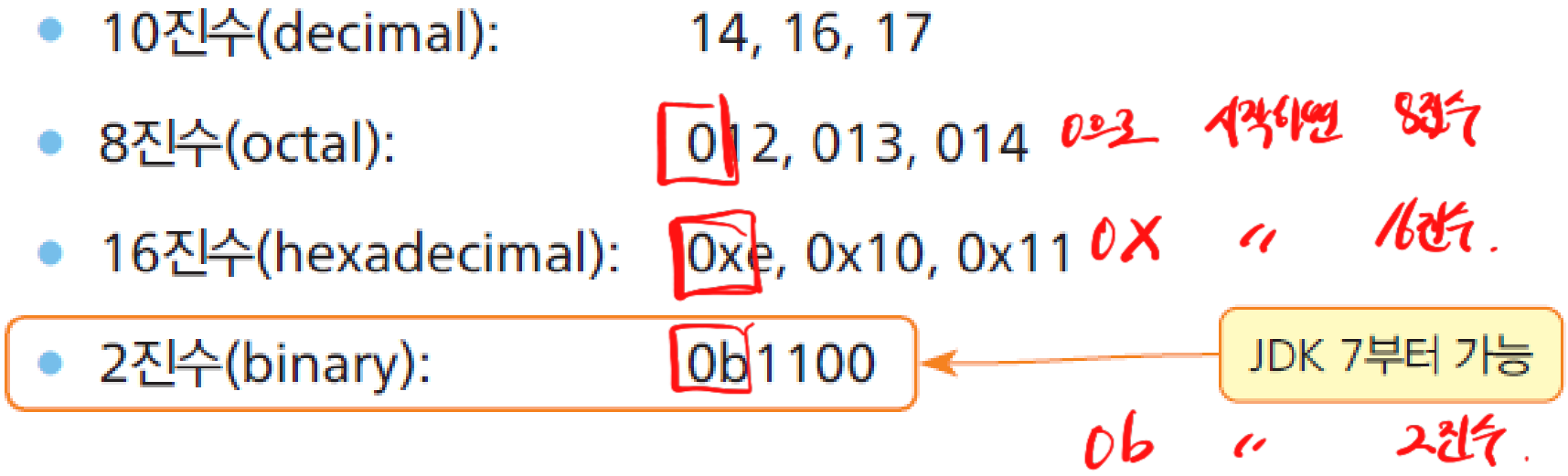
- 문자형인 char는 하나의 문자를 저장할 수 있다. 자바에서는 모든 문자를 2바이트의 유니코드(unicode)로 나타낸다

```
char ch1 = '가';          // 2바이트
char ch2 = '\uac00';      // '가'를 나타낸다.
char ch3 = 'a';           // 2바이트
```



리터럴

- 리터럴(literal)이란, x = 100;에서 100과 같이 소스 코드에 직접 쓰여 있는 값을 의미
- 리터럴에는 정수형, 부동소수점형, 문자형 등의 여러 가지 타입이 있다.



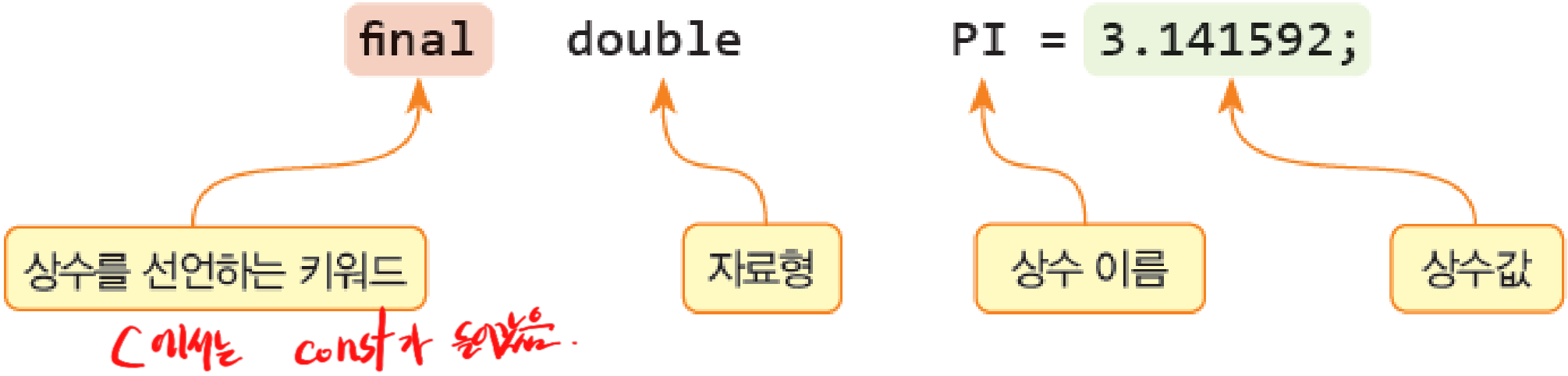
논리형 리터럴

- 논리형(boolean type)은 참과 거짓을 나타내는 데 사용
- 논리형은 true 또는 false만을 가질 수 있다.

```
boolean flag = true;
boolean x = 1 < 2;          // false가 저장된다.
```

상수

- 상수(constant)란 프로그램이 실행하는 동안 값이 변하지 않는 수 또는 변경 불가능한 수를 의미



변수 타입 추론

- Java 10부터는 var 키워드를 사용할 수 있다. 지역 변수의 타입을 자동으로 추론하는 것이 가능하다.

```
var  age = 22;                    // age는 int 타입으로 추론
var  name = "Kim";                // name은 String 타입으로 추론
```

var 키워드는 변수의 타입을 초기값으로부터 자동적으로 추론할 때 사용한다.

```
MAP<String,String>  map = new HashMap<String,String>();
```

```
var  map = new HashMap<String,String>();
```

map은 Map<String,String>타입으로 추론

컴파일러가 지역 변수 유형을 추론하기에 충분한 정보가 없으면 컴파일이 실패

```
var sum;
```

예제: 1광년 거리 계산하기

- src

```
public class Light {
        public static void main(String args[]) {
                final double LIGHT_SPEED = 3e5;
                double distance;

                distance = LIGHT_SPEED * 365 * 24 * 60 * 60;
                System.out.println("빛이 1년 동안 가는 거리 : " + distance + " km.");
        }
}
```

- 실행 결과

```
빛이 1년 동안 가는 거리 : 9.4608E12 km.
```

예제: 원의 면적 계산하기

- src

```
public class AreaTest {
        public static void main(String args[]) {
                final double PI = 3.141592;
                double radius, area;

                radius = 5.0;
                area = PI * radius * radius;
                System.out.println("반지름이 5인 원의 면적은 " + area);
        }
}
```

- 실행 결과

```
반지름이 5인 원의 면적은 78.5398
```

문자열

- 문자열(String)은 문자들의 모임이다.

```
String s1 = "Hello";
String s2 = "World!";
```

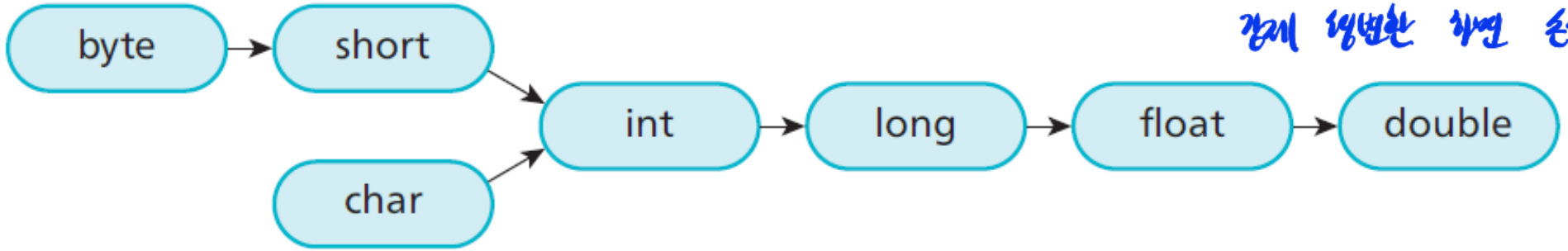
```
System.out.println(s1 + s2);          // "Hello World!"가 출력된다.
```

```
int age = 20;
System.out.println("내년이면 " + age + "살");   // "내년이면 20살"이 출력된다.
```

- System.out.println(1 + 2 + "3" + 4 + 5); 3345 .

형변환

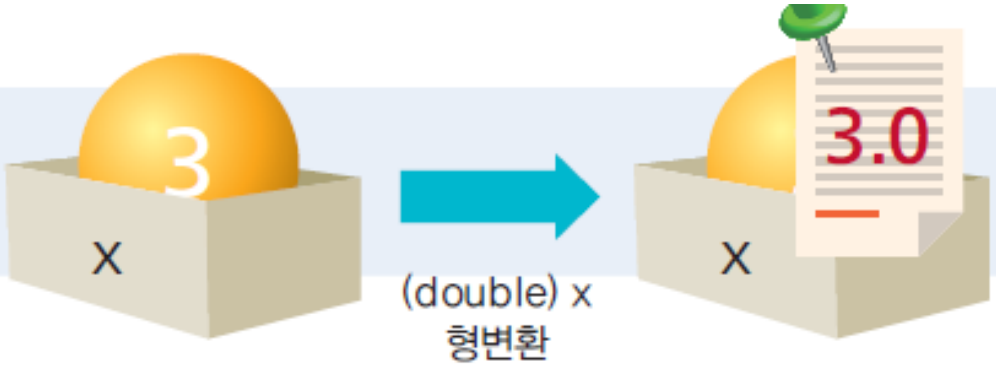
- 컴퓨터에서는 산술적인 연산을 하기 전에 피연산자의 타입을 통일하여야 한다.




```
double sum = 1.3 + 12;                    // 1.3 + 12.0으로 변환된다.
```

- 강제 형변환

```
int x = 3;
double y = (double) x;
```



예제: 형변환 실습하기

- Src

```
public class TypeConversion {
        public static void main(String args[]) {
                int i;
                double f;

                f = 1 / 5;
                System.out.println(f);

                f = (double) 1 / 5;
                System.out.println(f);

                i = (int) 1.7 + (int) 1.8
                System.out.println(i);
        }
}
```

- 실행결과

```
0.0
0.2
2
```

콘솔에서 입력 받기

- 콘솔에서 읽는 것은 System.in을 사용한다. System.in은 키보드에서 바이트를 읽어서 우리에게 전달한다.

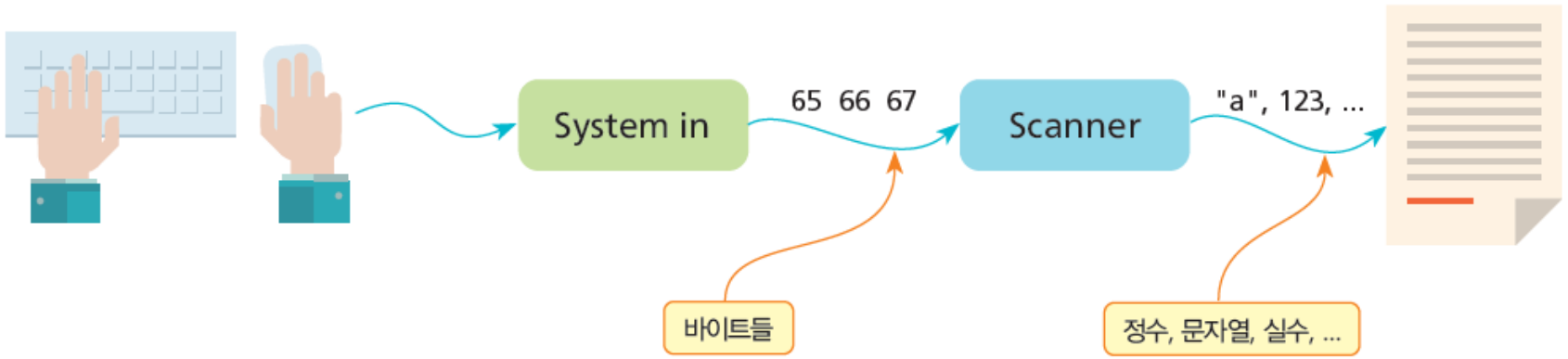
C의 #include 와 비슷.

```
import java.util.Scanner;                    // Scanner 클래스를 포함시킨다.
Scanner sc = new Scanner(System.in);         // Scanner 클래스의 객체를 생성한다.
```

Scanner 타입의 변수 선언

대문자. (키보드, 파일 등등 입력..)

Scanner 객체를 생성하고 System.in에 연결한다.



예제: 사용자로부터 두 수를 입력 받아서 더하기

다음과 같이 사용자로부터 정수 2개를 받아서 합을 계산하여 출력하는 프로그램을 작성해보자.

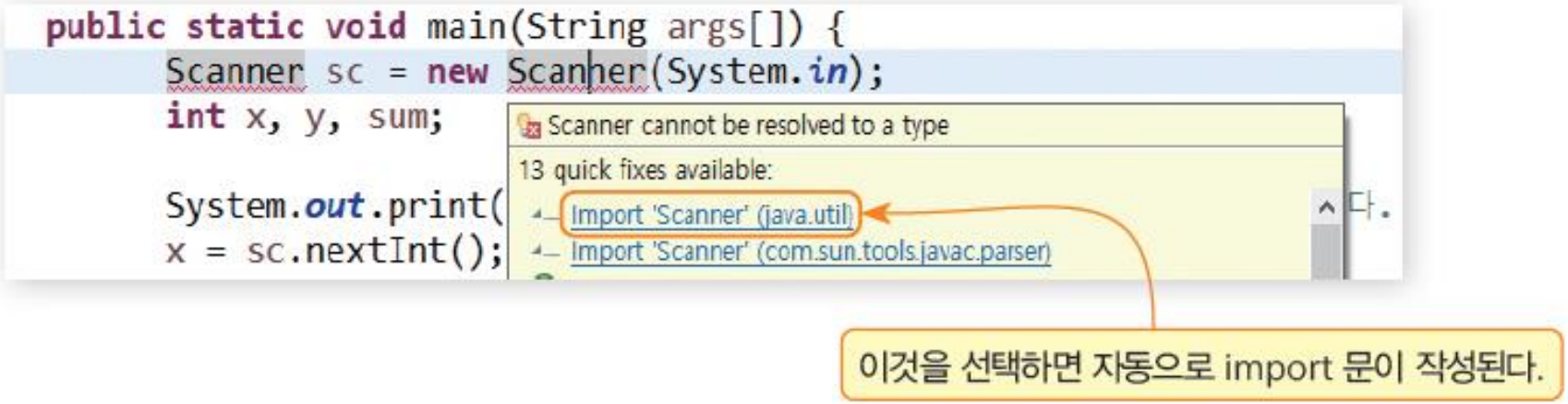
실행 결과

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

```
import java.util.Scanner;

public class Add2 {
	public static void main(String args[]) {
		Scanner sc = new Scanner(System.in);
		int x, y, sum;

		System.out.print("첫 번째 숫자를 입력하시오: "); // 줄을 바꾸지 않는다.
		x = sc.nextInt();
		 System.out.print("두 번째 숫자를 입력하시오: ");
		y = sc.nextInt();

		sum = x + y;
		System.out.println(sum); // 합을 출력한다.
	}
}
```

Import 문장

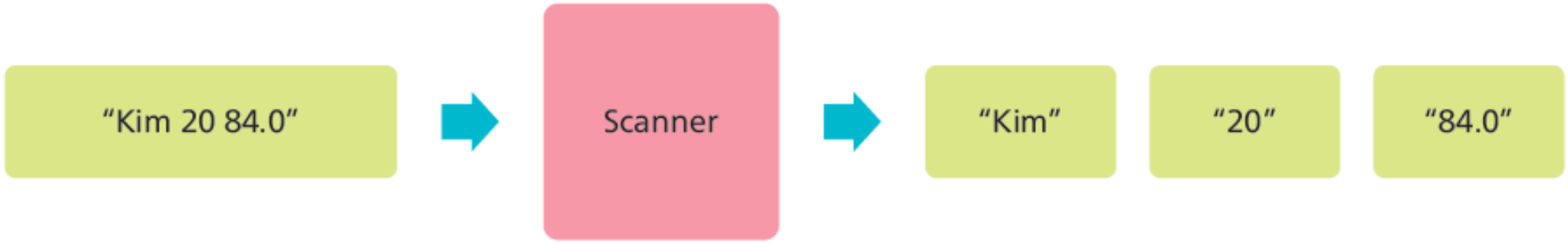
- 자바에서 모든 클래스는 사용하기 전에 import되어야 한다.
- 이클립스에서는 각종 import를 쉽게 할 수 있는 기능을 제공한다. 오류가 표시된 문장에 커서를 올리고 잠시 있으면, 해결책을 제공하는 Quick Fix 기능을 이용할 수 있다.



- 소스 파일 전체에 등장하는 모든 클래스를 import
  - Shift+Ctrl+O

Scanner 클래스

- Scanner 클래스는 키보드로부터 바이트 값을 받아서 분리자를 이용하여 각 바이트들을 토큰(token)으로 분리한다.
- 특별한 지정이 없으면 분리자는 공백문자(‘ ’, ‘₩n’, ‘₩t’)이다.



- String name = sc.next(); // 한 단어(토큰) "Kim"을 읽는다.
- int age = sc.nextInt(); // 문자열 "20"을 정수 20으로 변환하여 반환한다.
- double weight = sc.nextDouble(); // 문자열 "84.0"을 실수 84.0으로 변환하여 반환한다.
- String line = sc.nextLine(); // 문자열 "Kim 20 84.0"이 반환된다.

예제: 사용자로부터 이름과 나이를 받는 프로그램

사용자로부터 이름과 나이를 입력받아서 화면에 출력하는 프로그램을 작성하여 보자.

실행 결과

```
이름을 입력하시오: 홍길동
나이를 입력하시오: 24
홍길동님 안녕하세요! 24살이시네요.
```

```
import java.util.*;

public class InputString {
  public static void main(String[] args) {
    String name;
    int age;
    Scanner sc = new Scanner(System.in);

    System.out.print("이름을 입력하시오: ");
    name = sc.nextLine();
    System.out.print("나이를 입력하시오: ");
    age = sc.nextInt();

    System.out.println(name + "님 안녕하세요! "  + (age) + "살이시네요.");
  }
}
```

공백을 포함한 문자열을 통째로 입력.

수식과 연산자

- 수식은 피연산자와 연산자로 이루어진다.
- 연산자(operator)는 특정한 연산을 나타내는 기호를 의미
- 피연산자(operand)는 연산의 대상

연산자

표 2.2 자바 언어의 연산자들

우선순위 높음

연산자	우선순위	결합 규칙
단항(postfix)	++ --	오른쪽에서 왼쪽
단항(prefix)	++ -- + - ! ~ (형변환)	오른쪽에서 왼쪽
곱셈	* / %	왼쪽에서 오른쪽
덧셈	+ -	왼쪽에서 오른쪽
이동	〈〈 〉〉〉 〉〉	왼쪽에서 오른쪽
관계	〈 〉 〈= 〉=	왼쪽에서 오른쪽
동등	== !=	왼쪽에서 오른쪽
비트별 AND	&	왼쪽에서 오른쪽
비트별 XOR	^	왼쪽에서 오른쪽
비트별 OR	|	왼쪽에서 오른쪽
논리적 AND	&&	왼쪽에서 오른쪽
논리적 OR	||	왼쪽에서 오른쪽
조건	? :	오른쪽에서 왼쪽
대입	= += -= *= /= %=	오른쪽에서 왼쪽

우선순위 낮음

산술 연산

표 2.3 산술 연산자

연산자	기호	의미	예
덧셈	+	x와 y를 더한다	x+y
뺄셈	-	x에서 y를 뺀다.	x-y
곱셈	*	x와 y를 곱한다.	x*y
나눗셈	/	x를 y로 나눈다.	x/y
나머지	%	x를 y로 나눌 때의 나머지값	x%y

증감 연산자	차이점
++x	수식의 값은 증가된 x값이다.
x++	수식의 값은 증가되지 않은 원래의 x값이다.
--x	수식의 값은 감소된 x값이다.
x--	수식의 값은 감소되지 않은 원래의 x값이다.

예제: 나머지 연산자 예제

- 초 단위의 시간을 받아서 몇 분과 몇 초인지를 계산하여 출력하는 프로그램을 작성해보자.

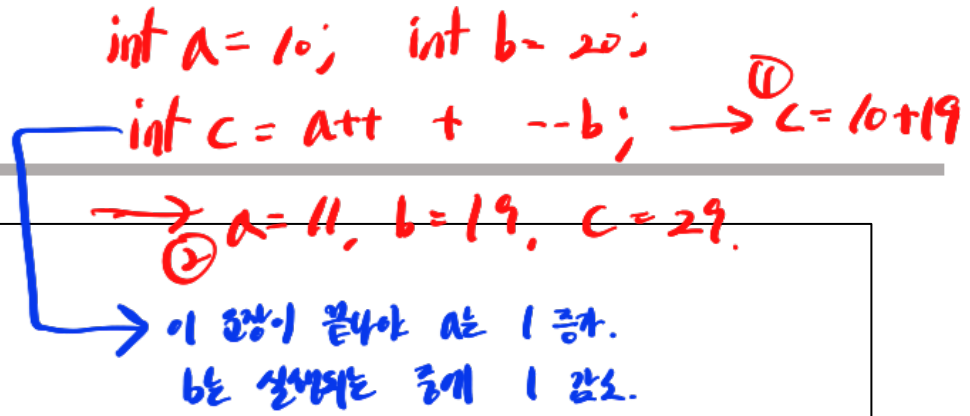
```
초를 입력하시오:310
310초는 5분 10초입니다.
```

```
import java.util.Scanner;

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

                System.out.print("초를 입력하시오:");
                int time = sc.nextInt();
                int sec = (time%60);            // 나머지 연산자를 이용한다.
                int min = (time/60);            // 정수 나눗셈을 이용한다.

                System.out.println(time+"초는 "+min+"분 "+sec +"초입니다.");
        }
}
```

복합 대입 연산자

복합 대입 연산자	의미
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

예제: 증감, 복합 대입 연산자 실습하기



- Src

```
public class IncOperator {
        public static void main(String[] args) {
                int x = 1, y = 1;

                int a = x++;                    // x의 값이 사용되기 전에 증가된다. a는 1가 된다.
                int b = ++y;                    // y의 값이 사용된 후에 증가된다. b는 2이 된다.
                System.out.println("a="+a+",  b="+b);

                int c = 100, d = 200;
                c += 10;                        // c = c + 10
                d /= 10;                        // d = d / 10
                System.out.println("c="+c+",  d="+d);
        }
}
```

- 실행 결과

```
a=1,  b=2
c=110,  d=20
```

관계 연산자

- 관계 연산자(relational operator)는 두 개의 피연산자를 비교하는 데 사용

표 2.4 관계 연산자

연산자 기호	의미	사용 예
==	x와 y가 같은가? 타입 다르면 다르다고 뜸?	x == y
!=	x와 y가 다른가?	x != y
〉	x가 y보다 큰가?	x 〉 y
〈	x가 y보다 작은가?	x 〈 y
〉=	x가 y보다 크거나 같은가?	x 〉= y
〈=	x가 y보다 작거나 같은가?	x 〈= y

논리 연산자

- 논리 연산자는 여러 개의 조건을 조합하여 참인지 거짓인지를 따질 때 사용한다.

연산자 기호	사용 예	의미
&&	x && y	AND 연산, x와 y가 모두 참이면 참, 그렇지 않으면 거짓
||	x || y	OR 연산, x나 y 중에서 하나만 참이면 참, 모두 거짓이면 거짓
!	!x	NOT 연산, x가 참이면 거짓, x가 거짓이면 참

예제: 관계/논리 연산자 실습하기

- Src

```
public class CompOperator {

    public static void main(String[] args) {

        System.out.print((3 == 4) + " ");
        System.out.print((3 != 4) + " ");
        System.out.print((3 > 4) + " ");
        System.out.print((4 > 3) + " ");

        System.out.print((3 == 3 && 4 == 7) + " ");        # 하나만 거짓이면 전체가 거짓
        System.out.print((3 == 3 || 4 == 7) + " ");        # 하나만 참이면 전체가 참

    }
}
```

↳ 관계연산자. 우선순위 ↑

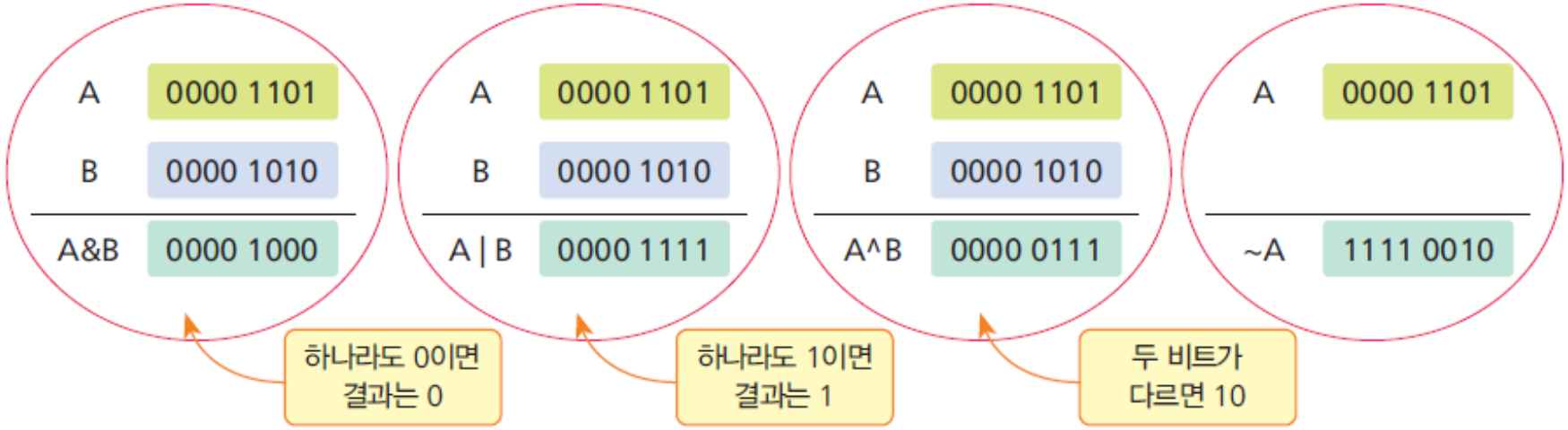
논리연산자 not > 관계연산자 우선순위 > 논리연산자 and 아

- 실행 결과

```
false true false true false true
```

비트 연산자

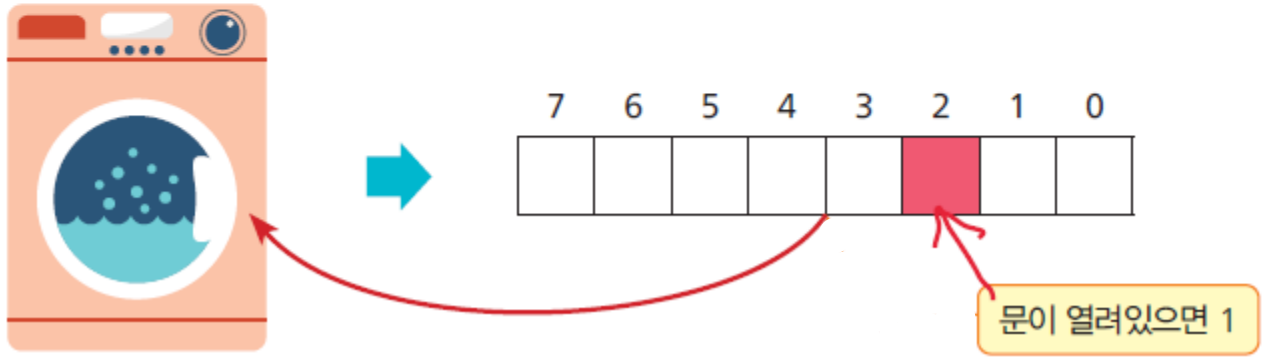
연산자	의미	예
~	비트 NOT	~(0x0FFF)은 0xF000이 된다.
&	비트 AND	(0x0FFF & 0xFFF0)은 0x0FF0이 된다.
^	비트 XOR	(0x0FFF ^ 0xFFF0)은 0xF00F이 된다.
|	비트 OR	(0x0FFF | 0xFFF0)은 0xFFFF이 된다.



예제: 비트 연산자 실습하기

비트 연산자 실습하기 예제 2-9

비트 연산은 어떤 경우에 사용될까? 프로그램과 하드웨어 칩 간의 통신에 사용된다. 예를 들어 세탁기 안에 있는 8개의 센서들의 값을 한 개의 바이트로 반환하는 하드웨어 칩이 있다고 하자. 이 바이트를 status라는 변수로 읽었다고 하자. 특정한 센서값이 1이 되었는지를 검사하는 용도로 사용된다. 예를 들어 세탁기의 문이 열려있으면 비트 2가 1이라고 하자. 비트 2가 0인지 1인지를 검사하는 코드를 작성해보자.



예제: 비트 연산자 실습하기

- Src

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

                byte status = 0b01101110;


                System.out.print( "문열림 상태=" + ((status & 0b00000100)!=0) );
        }
}
```

- 실행 결과

```
문열림 상태=true
```

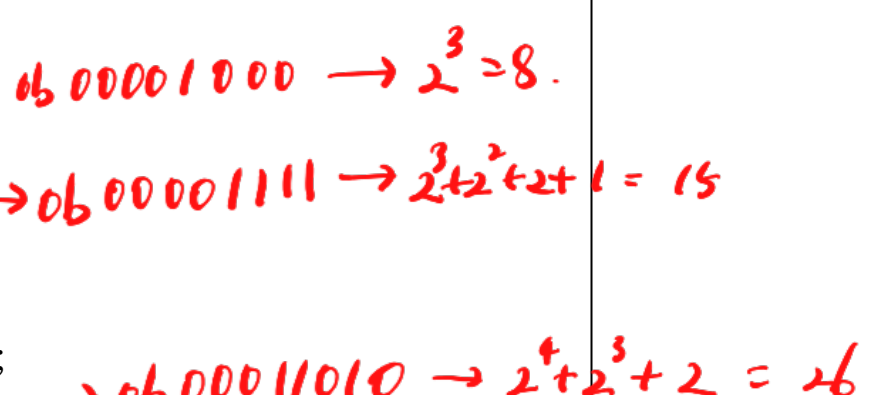
비트 이동 연산자

연산자	의미	예
〈〈	비트 왼쪽 이동	0xFFF 〈〈 4은 0xFFF0이 된다.
〉〉	비트 오른쪽 이동	0xFFFFFFF0 〉〉 4은 0xFFFFFFFF이 된다. 왼쪽 비트가 부호 비트로 채워진다.
〉〉〉	비트 오른쪽 이동 (unsigned)	왼쪽 비트가 부호 비트로 채워지지 않고 0으로 채워진다. 0xFFFFFFF0 〉〉〉 4은 0x0FFFFFFF이 된다.

예제: 비트 이동 연산자

- Src

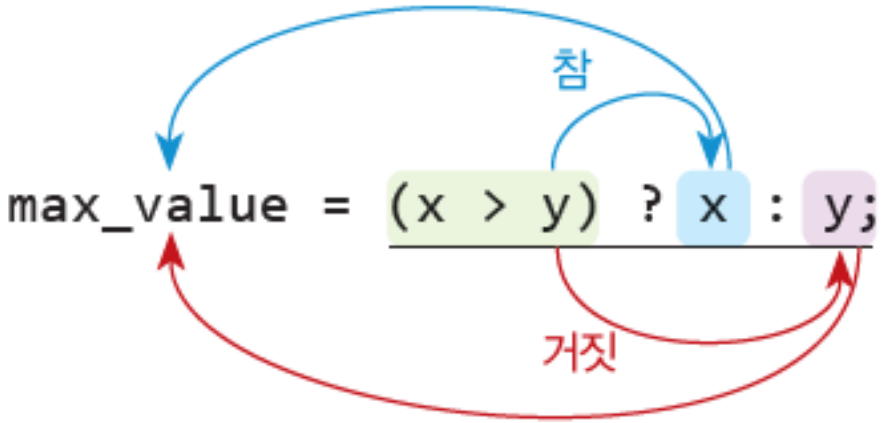
```
public class BitOperator2 {
	public static void main(String[] args) {

		int x = 0b00001101;		// 13
		int y = 0b00001010;		// 10

		System.out.print("x&y="+(x & y)+"    ");
		System.out.print("x|y="+(x | y)+"    ");
		System.out.print("x^y="+(x ^ y)+"    ");
		System.out.println("~x="+(~x)+"    ");
		System.out.print("x>>1="+(x>>1)+"    ");
		System.out.print("x<<1="+(x<<1)+"    ");
		System.out.println("x>>>1="+(x>>>1));
	}
}
```



- 실행 결과

```
x&y=8    x|y=15    x^y=7    ~x=-14
x>>1=6    x<<1=26    x>>>1=6
```

조건 연산자



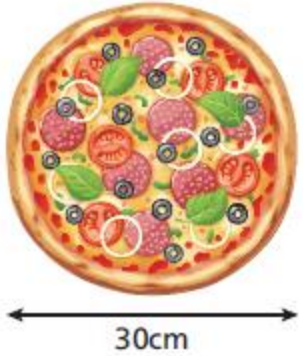

```
absolute_value = (x > 0) ? x: -x;      // 절대값 계산
max_value = (x > y) ? x: y;            // 최대값 계산
min_value = (x < y) ? x: y;            // 최소값 계산
```

조건 연산자 예제

예제 2-11 조건 연산자 사용하기

지름 20cm 피자 2개

지름 30cm 피자 1개



반지름이 20cm인 피자 2개와 30cm인 피자 1개의 면적을 비교해보자. 어떻게 주문하는 것이 유리한가?

실행 결과

```
20cm 피자 2개의 면적=2513.2736
30cm 피자 면적=2827.4328
30cm 피자 한 개를 주문하세요.
```

예제: 조건 연산자 예제

- Src

```
import java.util.*;

public class Pizza {
        public static void main(String[] args) {
                double area1 = 2 * 3.141592 * 20 * 20;
                double area2 = 3.141592 * 30 * 30;
                System.out.println("20cm 피자 면적=" + area1);
                System.out.println("30cm 피자 면적=" + area2);
                System.out.println((area1 > area2)? "20cm 두개": "30cm 한 개");
        }
}
```

- 실행 결과

```
20cm 피자 2개의 면적=2513.2736
30cm 피자 면적=2827.4328
30cm 피자 한 개를 주문하세요.
```