자바 프로그램 구조
자료형, 연산자, 키보드입력

514760
2022년 봄학기
3/8/2022
박경신

자바 프로그램 구조

```
/*
* 소스 파일 : Hello.java
*/
public class Hello {

  // main() 메소드에서 실행 시작
  public static void main(String[] args) {
     // "Hello World!" 문자열 화면 출력
      System.out.println("Hello World!");
  }
}
```

주석문

주석문

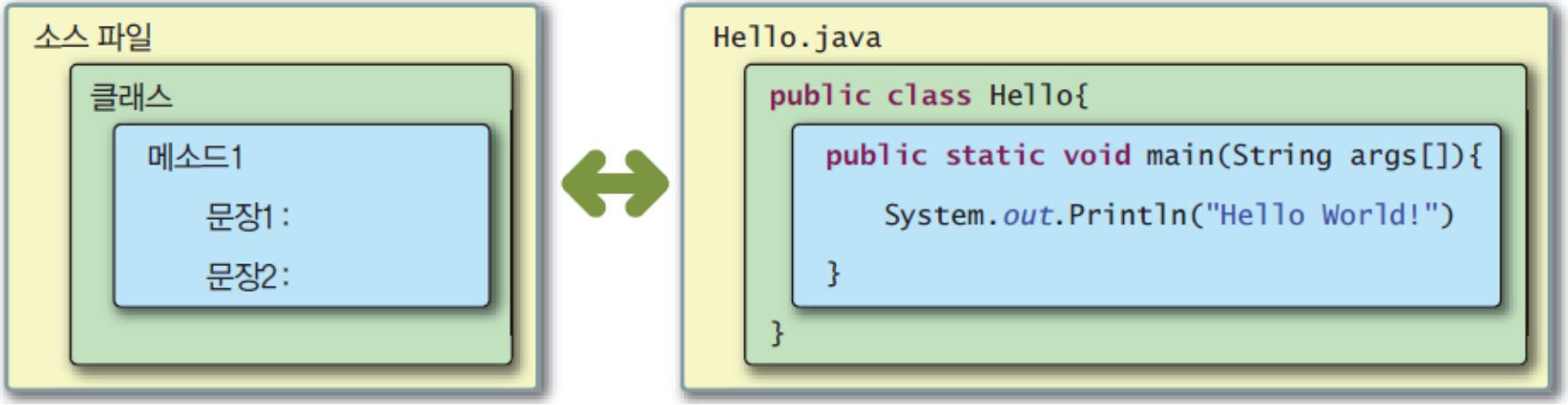
Hello 클래스

main 메소드

```
?
Hello World!
```

자바 프로그램의 구조

- 클래스 (class): 객체(object)를 만드는 설계도 (템플릿)
- 자바 프로그램은 기본적으로 클래스로 구성됨



- **public** 키워드는 Hello 클래스가 다른 클래스에서도 사용 가능함을 나타냄
- 하나의 클래스 안에는 여러 개의 메소드가 포함될 수 있음
- 하나의 메소드 안에는 여러 개의 문장이 포함될 수 있음

Comments

- 주석문 (Comments)
  - 프로그램에 대한 설명을 하기 위해 활용되는 코드의 일부로 컴파일 시에는 무시되고 사용되지 않음
- 세 가지 형태의 주석문
  - /* 설명 */
    - /*에서 */ 까지가 주석으로서 컴파일 시에 무시된다
    - 여러 줄에 걸쳐서 사용 가능
  - // 설명
    - //에서 줄의 끝까지 무시된다
  - /** 설명 */
    - /* 설명 */ 형태의 주석문이지만, 주로 선언문 앞에 사용되어 JDK에 포함된 Javadoc 프로그램을 이용해서 HTML문서를 만드는데 활용되는 주석문

Code Block

- 여러 명령문을 논리적으로 결합해야 할 때 중괄호 ({ })를 사용하여 명령문 그룹을 만들어 표현 - 이러한 명령문 그룹을 코드 블록(code block)이라고 함
- 코드 블록 안에는 변수를 선언할 수 있고, 다른 코드 블록을 포함할 수도 있음

```
public class Example {
    public static void main(String[] args) {
        int outer;
        {
            int inner;
            outer = 1;
            inner = 2;
        }
        outer = 5;
        //inner = 10;  // 오류
    }
}
```

내부 코드블록

main() 메소드 코드블록

Example 클래스 코드블록

Statement

- 문장 (Statement)은 사용자가 컴퓨터에게 작업을 지시하는 코드의 단위가 됨
- 문장들은 메소드 안에 들어 있거나 또는 클래스 내부에서 변수 등을 정의하는데 활용됨
- 보통 프로그램의 한 줄이 하나의 문장이 됨. 이때에는 문장의 끝은 항상 **세미콜론(;)**으로 끝남
- 때로는 { 문장 또는 문장들 }로 구성되는 블록 문장도 가능

Method

```
public static void main(String[] args) {
   // "Hello World!" 문자열 화면 출력
    System.out.println("Hello World!");
}
```

- public: 누구나 이용 가능
- static : 정적 메소드
- void : 반환값 (결과값) 없음
- main : 메소드 이름
- String[] args : 외부에서 주어지는 데이터를 받는 매개변수 (입력)

main()

- 자바 프로그램은 **main()** 메소드를 가지고 있는 클래스가 반드시 하나는 있어야 함
- **main()** 메소드에서 자바 프로그램의 실행이 시작됨

Identifier

- 식별자란?
  - 클래스, 변수, 상수, 메소드, 인터페이스 등에 붙이는 이름
- 식별자의 원칙
  - '@', '#', '!'와 같은 특수 문자, 공백 또는 탭은 식별자로 사용할 수 없으나 '_', '$'는 사용 가능
  - 유니코드 문자 사용 가능. 한글 사용 가능
  - 자바 언어의 키워드(keyword)는 식별자로 사용불가
  - 식별자의 첫 번째 문자로 숫자는 사용불가
  - ' _ ' 또는 '$'를 식별자 첫 번째 문자로 사용할 수 있으나 일반적으로 잘 사용하지 않음
  - 불린 리터럴 (true, false)과 널 리터럴(null)은 식별자로 사용불가
  - 영문 대소문자 구별 – e.g. Test와 test는 별개의 식별자
  - 길이 제한 없음
  - 관례적으로 낙타표기법 사용 – e.g. numberOfStudents

Identifier

- 사용 가능한 예

```
int name;
char student_ID;                    // '_' 사용 가능
void $func() { }                    // '$' 사용 가능
class Monster3 { }                  // 숫자 사용 가능
int whatsyournamemynameispark; // 길이 제한 없음
// 대소문자 구분. barChart와 barchart는 다름
int barChart; int barchart;
int 가격;                           // 한글 이름 사용 가능
```

- 잘못된 예

```
int 3Chapter;           // 숫자로 사용하였기 때문
class if { }            // if는 자바의 예약어임
char false;             // false는 사용 불가
void null() { }         // null 사용 불가
class %calc { }         // '%'는 특수문자
```

Keywords

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	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

식별자 이름 붙이는 관례

- 클래스(class) 이름

```
public class HelloWorld {}
class Vehicle {}
class AutoVendingMachine {}
```

  - 첫 번째 문자는 대문자로 시작
  - 여러 단어가 복합되어 있을 때는 각 단어의 첫 번째 문자만 대문자로 표시

- 변수(variable), 메소드(method) 이름

```
int iAge;                   // iAge의 i는 int의 i를 표시
boolean bIsSingle;          // bIsSingle의 처음 b는 boolean의 b를 표시
String strName;             // strName의 str은 String의 str을 표시
public int iGetAge() {}     // iGetAge의 i는 int의 i를 표시
```

  - 첫 단어 이후 각 단어의 첫 번째 문자는 대문자로 시작

식별자 이름 붙이는 관례

- 상수(constant) 이름

```
final static double PI = 3.141592;
```

  - 모든 문자를 대문자로 표시

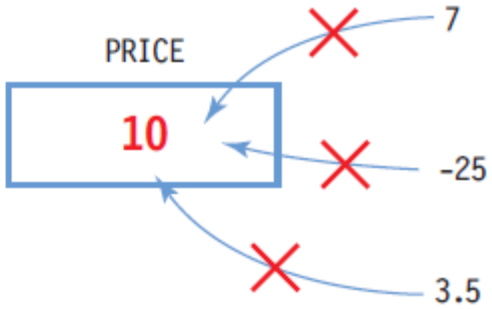
종류	사용 방법	예
클래스명	각 단어의 첫글자는 대문자	StaffMember, ItemProducer
변수명, 메소드명	소문자로 시작되어 2번째 단어의 첫글자는 대문자	width, height, payRate, accountNumber, getMonthDays()
상수	상수는 모든 글자를 대문자	MAX_NUMBER, PI

Constant

- 상수 (Constant)
  - **final** 키워드 사용
  - 변하지 않는 문자나 숫자 값. 값 변경 불가
  - 선언 시 초기값 지정

```
final static int PRICE = 10;
```

상수 선언 | 데이터 타입 | 상수 이름 | 초기화



- 상수 선언 사례

```
final double PI = 3.141592;
final int LENGTH = 20;
```

Variable

- 변수 (Variable)
  - 값을 임시 저장하기 위한 공간
    - 변수 값은 프로그램 수행 중 변경될 수 있음
  - 데이터 타입에서 정한 크기의 메모리 할당
  - 반드시 변수 선언과 값을 초기화 후 사용
- 변수 선언과 초기화
  - 자료형 (DataType)과 이름을 적어 변수를 선언

```
int radius; // 변수 선언
char c1, c2, c3; // 3 개의 변수를 한 번에 선언
double weight;
```

```
int radius = 10; // 변수 선언 및 초기화
char c1 = 'a', c2 = 'b', c3 = 'c'; // 선언과 동시에 초기값 지정
double weight = 75.56;
```

```
radius = 10 * 5; // 변수에 값 대입 (= 연산자 다음에 식)
c1 = 'r';
weight = weight + 5.0;
```

Variable

- 변수 선언과 초기화
  - 클래스 멤버 변수는 자동으로 초기화
  - 초기화시키지 않은 변수를 사용할 때 오류 발생

```
public class UninitializedVariable {
    double number; // 클래스 멤버 변수 자동으로 초기화
    public static void main(String[] args) {
        int num; // 지역 변수
        System.out.println("num = " + num);
    }
}
```

```
Error: variable num might not have been initialized
        System.out.println("num = " + num);
                                  ^
```

Data Type

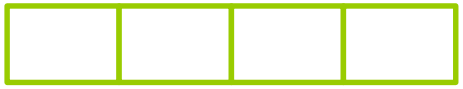
- 자바의 자료형 (Data Type)
  - 기초형 (Primitive Type)
    - boolean
    - char
    - byte
    - short
    - int
    - long
    - float
    - double
  - 참조형 (Reference Type)
    - 클래스 (class)
    - 인터페이스 (interface)
    - 배열 (array)
    - 문자열 (String)

Data Type	Default Value
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'₩u0000'
String (or any object)	null
boolean	false

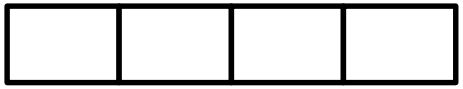
Primitive Data Types

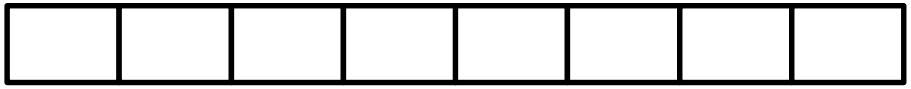
byte  *(1 Byte, -128 ~ 127)*

short  *(2 Bytes, -32768 ~ 32767)*

int  *(4 Bytes, $-2^{31} \sim 2^{31}-1$)*

long  *(8 Bytes, $-2^{63} \sim 2^{63}-1$)*

float  *(4 Bytes, -3.4E38~3.4E38)*

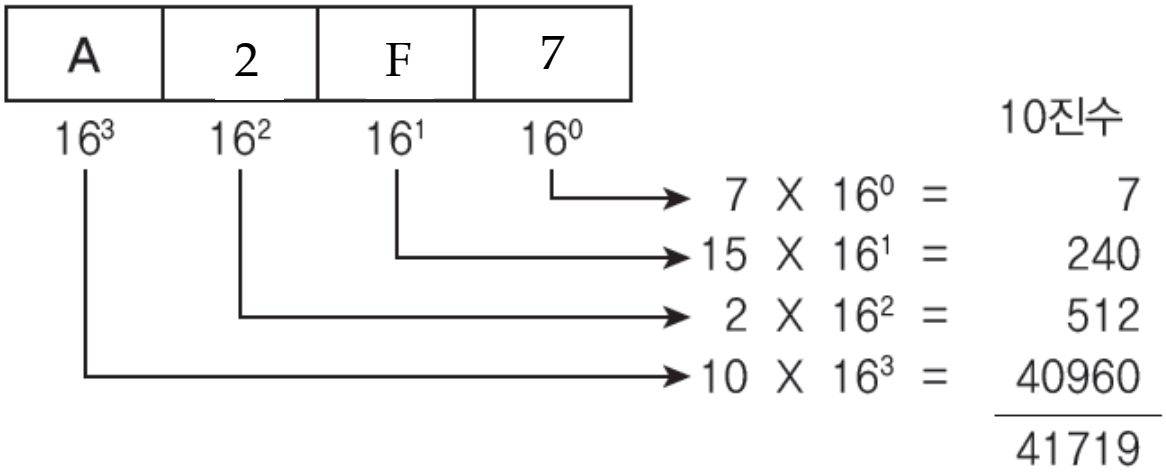
double  *(8 Bytes, -1.7E308~1.7E308)*

boolean  *(1 Byte, true or false)*

char  *(2 Bytes, Unicode)*

Integer Literals

- 10진수(decimal): 14, 16, 17
- 8진수(octal): 016, 018, 019
- 16진수(hexadecimal): 0xe, 0x10, 0x11
- 2진수(binary): 0b1100



Integer Literals

- 정수형 리터럴 : 정수 직접 표시
  - 8진수 : 0으로 시작하는 숫자는 모두 8진수
    - int n = 015; // 10진수로 13
  - 16진수 : 0x로 시작하는 숫자는 16진수
    - int n = 0x15; // 10진수로 21
  - 10진수 : 0으로 시작하지 않는 숫자는 10진수
    - 15, 3, 20, 55, 88
  - 모든 정수타입 리터럴은 int형으로 컴파일함
  - long 타입 리터럴은 숫자 뒤에 L 또는 l을 붙임
    - ex) 24L, 3578l

Floating-Point Literals

- 부동 소수점을 갖는 수 직접 표시
  - 소수점을 찍은 실수, 지수(exponent)식으로 표현한 실수
    - 12. 또는 12.0
    - .1234 또는 0.1234 또는 1234E-4
  - 숫자 뒤에 f(float)나 d(double)을 명시적으로 붙이기도 함
    - 0.1234 또는 0.1234D 또는 0.1234d → double 타입
    - 0.1234f 또는 0.1234F → float 타입
    - 1234D 또는 1234d → 1234.0과 같으며 double 타입
    - 1234F 또는 1234f → 1234.0과 같으며 float 타입
  - 실수 타입 리터럴은 double 타입으로 컴파일됨

Character Literals

- 문자는 유니코드 규격 중에서 UTF-16 사용
  - 단일 인용부호(‘’)로 문자 하나 표현
    - 'a', 'W', '가', '*', '3', '7'

```
char ch1 = '가';
char ch2 = ‘\uac00'; // '가'
```

  - ₩다음에 숫자는 8진수로서 0 ~ 337사이의 8진수만 가능
    - ₩102 -> 문자 ‘B’를 나타내는 8진수
    - ₩337 -> 문자 ‘β’를 나타내는 8진수
  - ₩u다음에 4자리 16진수, 2 바이트의 유니코드(Unicode)
    - ₩u0041 -> 문자 'A'의 유니코드(0041)
    - ₩uae00 -> 한글문자 '글'의 유니코드(ae00)
  - 특수 기호는 ₩로 시작

‘₩b’	백스페이스	‘₩t’	탭
‘₩n’	라인피드	‘₩f’	폼피드
‘₩r’	캐리지리턴	‘₩”’	이중 인용 부호
‘₩”’	단일 인용 부호	‘₩₩’	백슬래시

String Literals

- 문자열 리터럴
  - 이중 인용부호로 묶어서 표현
    - "Good", "Morning", "자바", "3.19", "26", "a"
  - 자바에서 문자열은 객체이므로 기본 타입이 아님
  - 자바에서 문자열**(String)**은 문자들의 모임이다. 예를 들어서 문자열 "Hello"는 H, e, l, l, o 등의 5개의 유니코드 문자로 구성됨
  - 문자열 리터럴은 String 객체로 생성됨
- **String 클래스**가 제공됨

```
String str1 = "Welcome";
String str2 = null;
System.out.println(str1);
```

Boolean Literals

- 논리 리터럴
  - true 또는 false 값 표시

```
boolean b = true;
boolean c = 10 > 0; // 10>0이 참이므로 c 값은 true
```

- 논리 타입과 정수타입 사이의 타입 변환 허용 안 됨

```
int i;
//if ((boolean)i) { } // 정수 i를 논리 타입으로 변환할 수 없음
                      // 컴파일 에러
```

  - (i == 1) 또는 (i != 0)과 같은 논리 연산을 사용해야 함

null

- null 리터럴
  - 어떠한 참조형(reference data type)의 값으로 사용 가능
    - int i = null; // 기본 데이터 타입에는 사용 불가
    - String str = null;

변수, 상수, 리터럴 사용하기

```
public class CircleArea {
  public static void main(String[] args) {
    final double PI = 3.141592; // 원주율을 상수로 선언
    double radius = 5.0; // 원의 반지름
    double circleArea = 0; // 원의 면적

    circleArea = radius*radius*PI; // 원의 면적 계산

    // 원의 면적을 화면에 출력한다.
    System.out.print("원의 면적 = ");
    System.out.println(circleArea);
  }
}
```

```
원의 면적 = 78.5398
```

Type Conversion

- 자동적인 형 변환(Implicit Type Conversion) 경우
  - 원래의 타입보다 큰 자료타입으로 바뀔 때

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

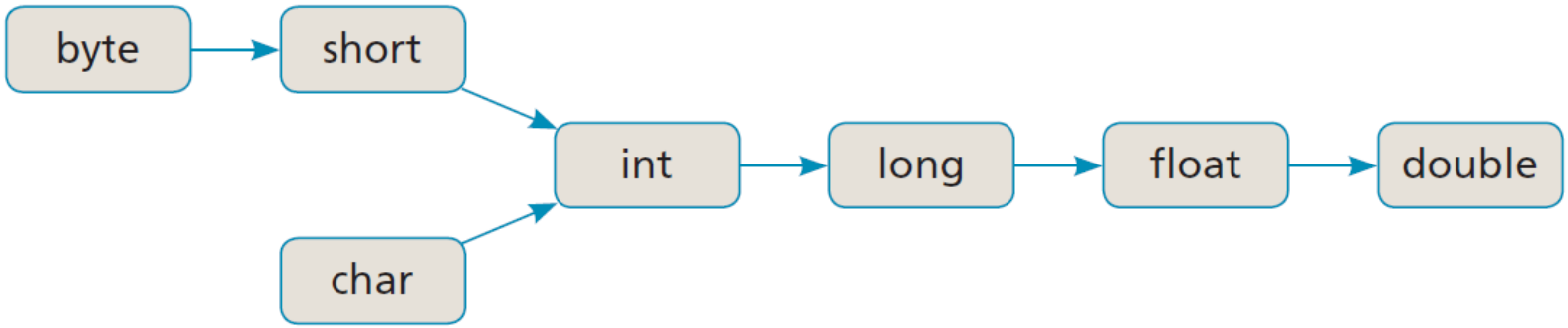
  - 원본 데이터 그대로 보존
- 자동적인 형 변환 사례

```
byte a;
int price;
price = a;
```


```
long var;
int n = 32555;
byte b = 25;
var = n; // int -> long 타입으로 자동 변환.
         // var 값은 32555
var = b; // byte -> long 타입으로 자동 변환
         // var 값은 25
```

Type Conversion

- 자동적인 형변환
  - 피연산자 중 하나가 double형이면 다른 피연산자도 double형으로 변환됨
  - 피연산자 중 하나가 float형이면 다른 피연산자도 float형으로 변환됨
  - 피연산자 중 하나가 long형이면 다른 피연산자도 long형으로 변환됨
  - 그렇지 않으면 모든 피연산자는 int형으로 변환됨



Type Conversion

- 강제 형 변환 (Explicit Type Conversion)
  - 개발자의 의도적으로 타입 변환
- 강제 형 변환 방법

```
byte a;
int price;
a = (byte) price;
```

바이트타입 변수

price 정수 값을 byte 타입으로 강제타입 변환

정수타입 변수

  - 강제 타입 변환 사례
    - 실수 타입이 정수 타입으로 강제 변환 시 소수점 아래가 버려짐
      - 데이터 손실

```
short var;
int n = 855638017; // n의 16진수 값은 0x33000001
var = (short) n; // int 타입에서 short 타입으로 강제 변환. var 값은 1
double d = 1.9;
int n = (int)d; // n은 1이 된다.
```

0x3300**0001** n 32비트

0x**0001** var 16비트

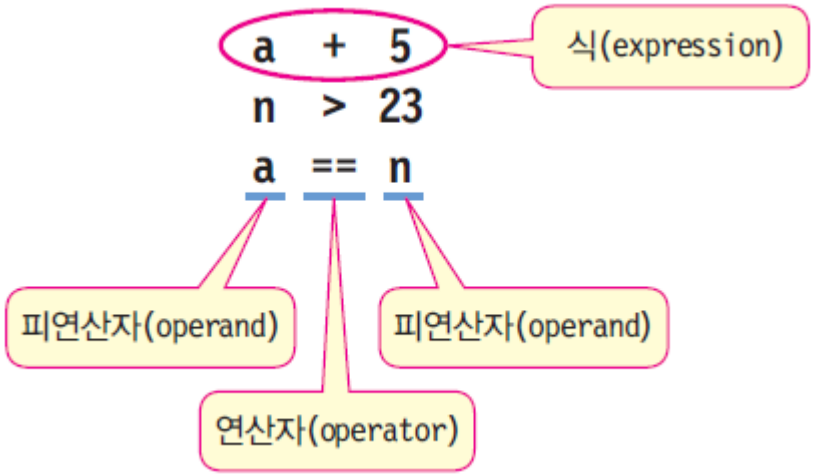
예제 : 형변환 사용

```
public class TypeConversion {
  public static void main(String[] args) {
    byte b = 127;
    int i = 100;
    System.out.println(b+i);
    System.out.println(10/4);
    System.out.println(10.0/4);
    System.out.println((char)0x12340041);
    System.out.println((byte)(b+i));
    System.out.println((int)2.9 + 1.8);
    System.out.println((int)(2.9 + 1.8));
    System.out.println((int)2.9 + (int)1.8);
  }
}
```

```
227
2
2.5
A
-29
3.8
4
3
```

식과 연산자

- 연산 - 주어진 식을 계산하여 결과를 얻어내는 과정



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

Operator Precedence

높음

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

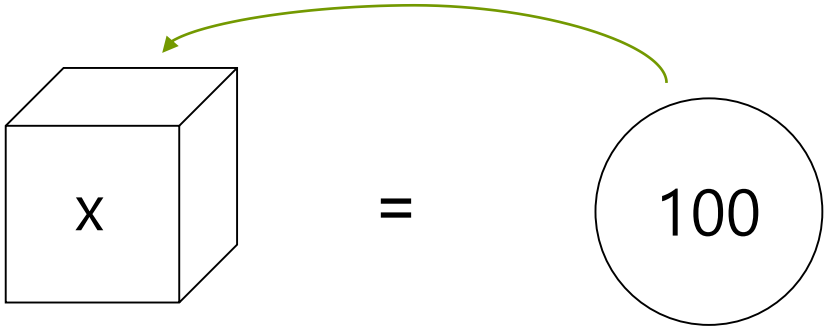
낮음

- 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

Assignment Operators

- 대입 연산자(=)는 왼쪽 변수에 오른쪽 수식의 값을 계산하여 저장
- 대입 연산자 == 할당 연산자 == 배정 연산자라고 함

```
x = 100; // 100을 변수 x에 대입
```



Arithmetic Operators

산술연산자	의미	예	결과값
+	더하기	25.5 + 3.6	29.1
-	빼기	3 - 5	-2
*	곱하기	2.5 * 4	10.0
/	나누기	5/2	2
%	나머지	5%2	1

- /와 % 연산자
  - 정수 연산에서, /은 나눗셈의 몫. %는 나눗셈의 나머지
  - %의 이용 사례 : 홀수 짝수 판별
    - int r = x % 2; // r이 1이면 x는 홀수

Unary Operators: Postfix & Prefix

연산자	내용
++x	x값을 먼저 증가한 후 다른 연산에 사용. 이 수식의 값은 증가된 x값
x++	x값을 먼저 사용한 후 증가. 이 수식의 값은 증가되지 않은 원래 x값
--x	x값을 먼저 감소한 후 다른 연산에 사용. 이 수식의 값은 감소된 x값
x--	x값을 먼저 사용한 후 감소. 이 수식의 값은 감소되지 않은 원래 x값

```
int i = 10;
System.out.println("i++=" + (i++)); // 10 출력 후 increments
System.out.println("i=" + i); // 11 출력

int j = 20;
System.out.println("++j=" + (++j)); // increments 한 후 21 출력
System.out.println("j=" + j); // 21 출력
```

Equality & Relational Operators

비교연산자	내용	예제	결과
a < b	a가 b보다 작으면 true 아니면 false	3 < 5	true
a > b	a가 b보다 크면 true 아니면 false	3 > 5	false
a <= b	a가 b보다 작거나 같으면 true 아니면 false	1 <= 0	false
a >= b	a가 b보다 크거나 같으면 true 아니면 false	10 >= 10	true
a == b	a가 b와 같으면 true 아니면 false	1 == 3	false
a != b	a가 b와 같지 않으면 true 아니면 false	1 != 3	true

Logical Operators

로직연산자	내용
a && b	AND연산. a와 b 모두 true 면 true, 그렇치 않으면 false
a || b	OR연산. a나 b 중 하나만 true 면 true, 모두 false이면 false
!a	NOT연산. a가 true 면 false, a가 false면 true
a ^ b	XOR연산. a와 b 중 정확히 하나만 true 면 true, 둘 다 true 또는 둘 다 false이면 false

예제 : 비교 연산자와 논리 연산자 사용

```
public class LogicalOperator {
   public static void main (String[] args) {
      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));
      System.out.println((3 > 2) && (3 > 4));
      System.out.println((3 != 2) || (-1 > 0));
      System.out.println((3 != 2) ^ (-1 > 0));
   }
}
```

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

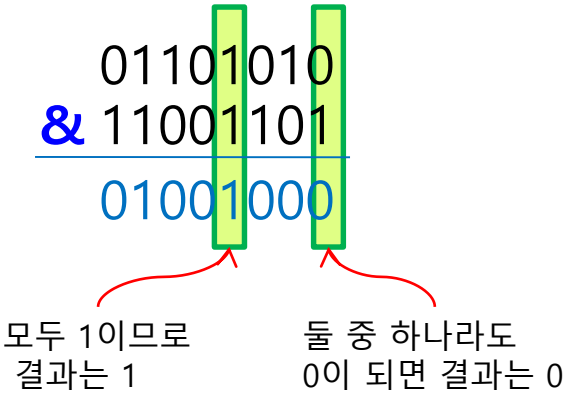
Bitwise Operators

- 피 연산자의 각 비트들을 대상으로 하는 연산

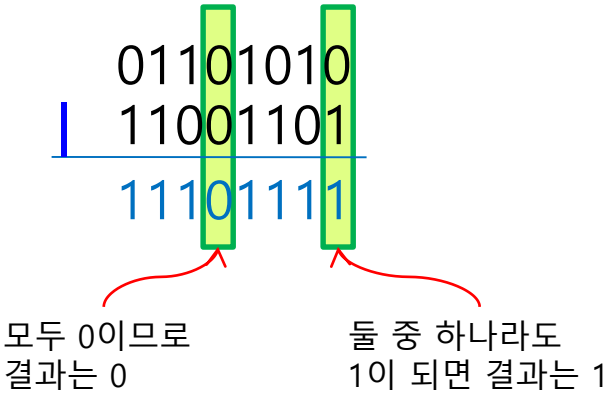
비트연산자	내용
a & b	a와 b의 각 비트들의 AND 연산. 두 비트 모두 1일 때만 1이 되며 나머지는 0
a | b	a와 b의 각 비트들의 OR 연산. 두 비트 모두 0일 때만 0이 되며 나머지는 1
a ^ b	a와 b의 각 비트들의 XOR 연산. 두 비트가 서로 다르면 1, 같으면 0
~ a	단항 연산자로서, a의 각 비트들에 NOT 연산. 1을 0으로, 0을 1로 변환

비트 연산자 사용 예시

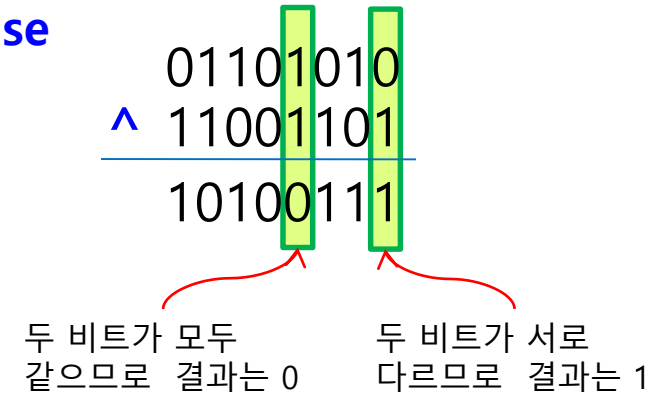
Bitwise AND



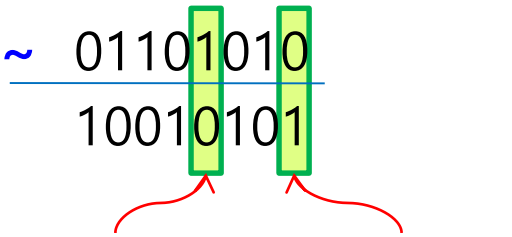
Bitwise OR



Bitwise XOR



Bitwise NOT

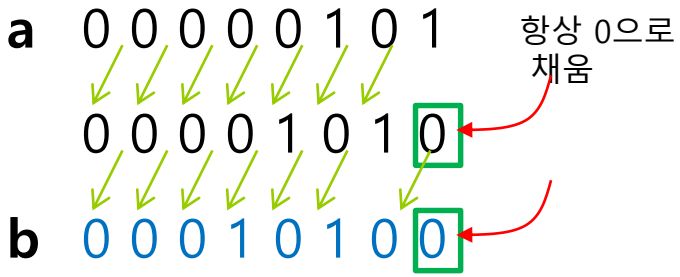


Bit Shift Operators

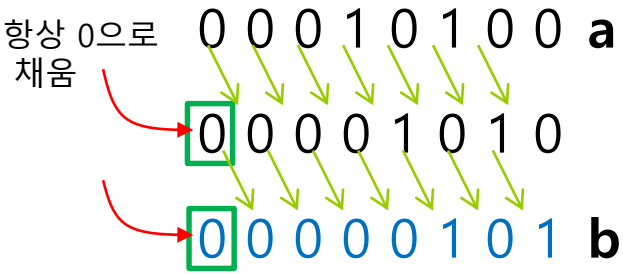
시프트 연산자	내용
a >> b	a의 각 비트를 오른쪽으로 b 번 시프트한다. 최상위 비트의 빈자리는 시프트 전의 최상위 비트로 다시 채운다. 산술적 오른쪽 시프트.
a >>> b	a의 각 비트를 오른쪽으로 b 번 시프트한다. 그리고 최상위 비트의 빈자리는 0으로 채운다. 논리적 오른쪽 시프트.
a << b	a의 각 비트를 왼쪽으로 b 번 시프트한다. 그리고 최하위 비트의 빈자리는 0으로 채운다. 산술적 왼쪽 시프트.

시프트 연산자 사용 예시

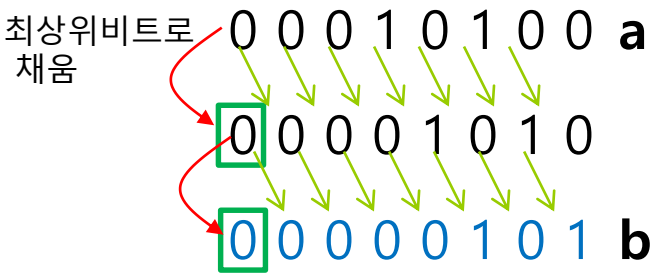
```
byte a = 5; // 5
byte b = (byte)(a << 2); // 20
```



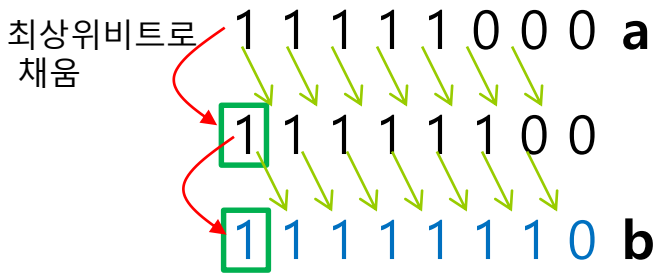

```
byte a = 20; // 20
byte b = (byte)(a >>> 2); // 5
```




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




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



예제 : 비트 연산자와 시프트 연산자

```
public class BitShiftOperator {
   public static void main (String[] args) {
      short a = (short)0x55ff;  // 0x000055ff
      short b = 0x00ff; // 0x000000ff
      // 비트 연산
      System.out.printf("%x\n", a & b);
      System.out.printf("%x\n", a | b);
      System.out.printf("%x\n", a ^ b);
      System.out.printf("%x\n", ~a);

      byte c = 20; // 0x14 (0000 0000 0000 0000 0000 0000 0001 0100)
      byte d = -8; // 0xf8 (1111 1111 1111 1111 1111 1111 1111 1000)
      // 시프트 연산
      System.out.printf("%x\n", c << 2); // c를 2비트 왼쪽 시프트
      System.out.printf("%x\n", c >> 2); // c를 2비트 오른쪽 시프트. 0 삽입
      System.out.printf("%x\n", d >> 2); // d를 2비트 오른쪽 시프트. 1 삽입
      System.out.printf("%x\n", d >>> 2); // d를 2비트 오른쪽 시프트. 0 삽입
   }
}
```

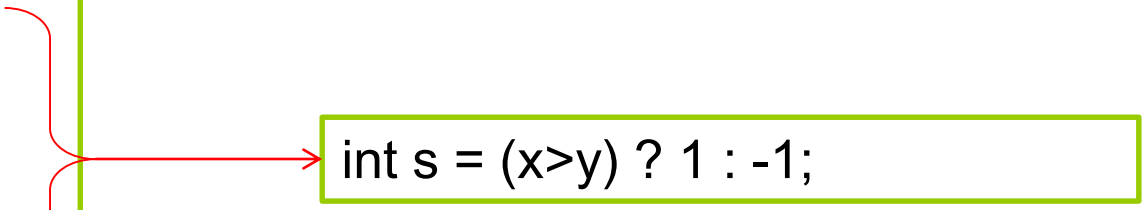
printf("%x₩n", …)는 결과 값을 16진수 형식으로 출력

```
ff
55ff
5500
ffffaa00


50
5
fffffffe
3ffffffe
```

Ternary Operator

- opr1?opr2:opr3
  - 세 개의 피연산자로 구성된 삼항(ternary) 연산자
    - opr1이 true이면, 연산식의 결과는 opr2, false이면 opr3
  - if-else을 간결하게 표현할 수 있음

```
int x = 5;
int y = 3;

int s;
if(x>y)
   s = 1;
else
   s = -1;
```



예제 : 조건 연산자 사용

```
public class TernaryOperator {
   public static void main (String[] args) {
      int a = 3, b = 5;

      System.out.println("두 수의 차는 " + ((a>b) ? (a-b) : (b-a)));
   }
}
```

```
두 수의 차는 2
```

System.out.println

- System.out.println 화면 출력
  - 표준 출력 스트림에 메시지 출력

```
System.out.println(a); // 문자 ? 화면 출력
System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
System.out.println(s); // 정수 s 값 화면 출력
```

  - 표준 출력 스트림 System.out의 println() 메소드 호출
  - println()은 여러 종류 데이터 타입 출력 가능
  - println()은 출력 후 다음 행으로 커서 이동
- System.out.printf()는 서식에 맞춰 값을 화면에 출력
  - 첫 번째 인자인 문자열이 서식
  - 서식 문자열에서 '%'로 시작하는 서식 지시어

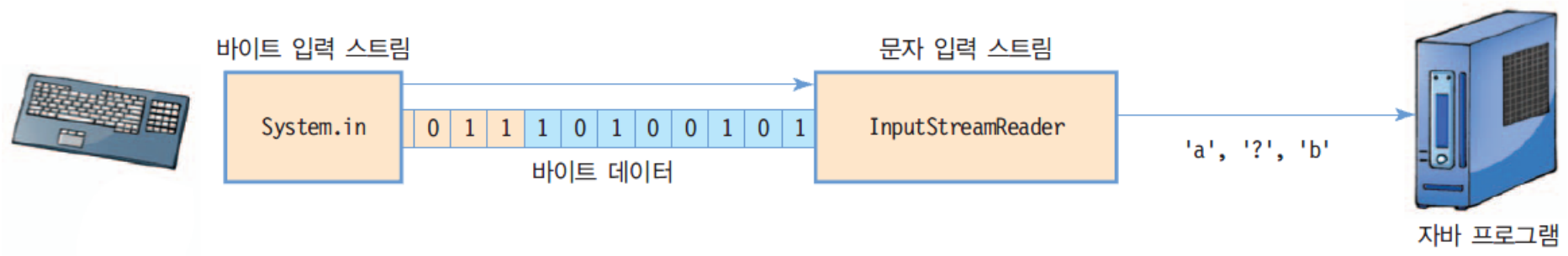
```
System.out.printf("byte min: %d max %d\n", Byte.MIN_VALUE,
Byte.MAX_VALUE); // 정수 출력
System.out.printf("circle area=%f\n", area); // 실수 출력
```

System.out.printf()

서식 지시어	설명
%%	% 글자를 화면에 출력
%n 또는 ₩n	줄바꿈 문자 출력
%d	인자로 전달되는 정수값을 출력 (byte, short, int, long)
%f	인자로 전달되는 실수값을 출력 (float, double)
%c	인자로 전달되는 문자를 출력(char)
%s	인자로 전달되는 내용을 문자열 형태로 변환해서 출력
%h	인자의 해시코드(hashcode)를 출력. 주로 참조값을 출력할 때 사용

System.in

- 자바에서 키 입력: System.in
  - 자바의 표준 입력 스트림
  - java.io 패키지의 InputStream 클래스
    - System.in은 바이트 스트림으로서 키 값을 바이트로 리턴
    - 문자로 변환하려면 InputStreamReader 클래스를 이용



  - 입력 동안 문제가 발생하면 IOException 발생
    - try-catch를 이용한 예외 처리 필요

사용자로부터 키보드 입력 받기

- Scanner 클래스를 사용

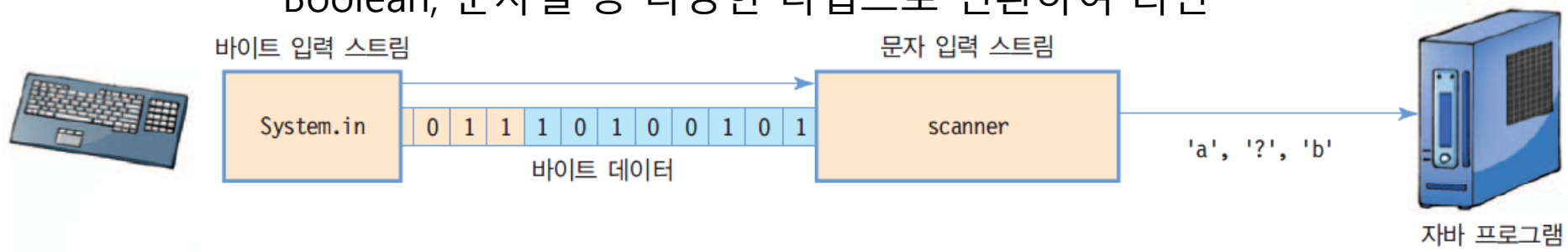
```
import java.util.*;                          // Scanner 클래스 포함

Scanner input = new Scanner(System.in);

System.out.print("문장을 입력하시오: ");
String line = input.nextLine();      // 한 줄을 읽는다.
```

Scanner

- Scanner 클래스를 이용한 키 입력
  - java.util.Scanner 클래스
  - Scanner 객체 생성

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

    - System.in에게 키를 읽게 하고 읽은 바이트를 문자, 정수, 실수, Boolean, 문자열 등 다양한 타입으로 변환하여 리턴

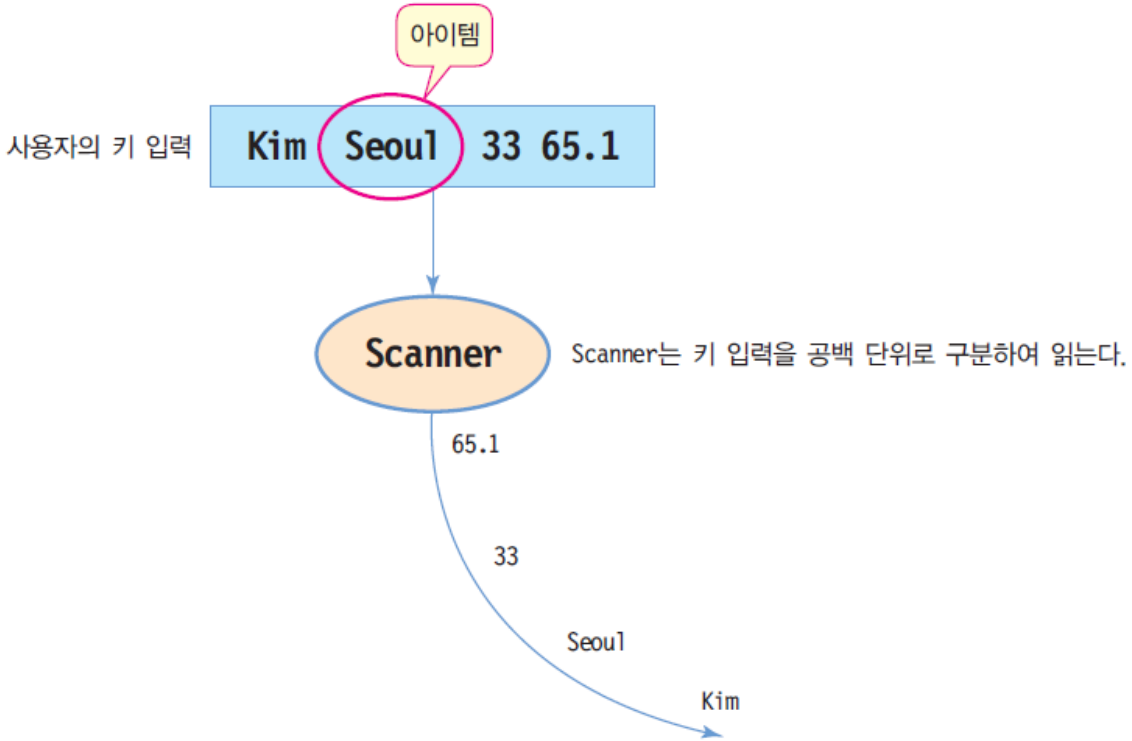



```
import java.util.Scanner;
```

  - import문 필요
    - 소스 맨 앞줄에 사용
  - Scanner에서 키 입력 받기
    - Scanner는 입력되는 키 값을 공백으로 구분되는 아이템 단위로 읽음
    - 공백 문자 :'₩t',' ₩f',' ₩r',' ',' ₩n'

Scanner 사용

```
Scanner scanner = new Scanner(System.in);
String name = scanner.next();                  // "Kim"
String addr = scanner.next();                  // "Seoul"
int age = scanner.nextInt();                   // 33
double weight = scanner.nextDouble();          // 65.1
```



Scanner 주요 메소드

생성자/메소드	설명
String next()	다음 아이템을 찾아 문자열로 반환
boolean nextBoolean()	다음 아이템을 찾아 boolean으로 변환하여 반환
byte nextByte()	다음 아이템을 찾아 byte로 변환하여 반환
double nextDouble()	다음 아이템을 찾아 double로 변환하여 반환
float nextFloat()	다음 아이템을 찾아 float로 변환하여 반환
int nextInt()	다음 아이템을 찾아 int로 변환하여 반환
long nextLong()	다음 아이템을 찾아 long으로 변환하여 반환
short nextShort()	다음 아이템을 찾아 short로 변환하여 반환
String nextLine()	한 라인 전체('₩n' 포함)를 문자열 타입으로 반환

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

예제 : Scanner를 이용한 키 입력 사용

```
import java.util.Scanner;

public class ScannerExam {
  public static void main (String args[]) {
    Scanner a = new Scanner(System.in);
    System.out.println("나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요");
    System.out.println("당신의 나이는 " + a.nextInt() + "살입니다.");
    System.out.println("당신의 체중은 " + a.nextDouble() +"kg입니다.");
    System.out.println("당신의 신장은 " + a.nextDouble()+ "cm입니다.");
  }
}
```

```
나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요
35 75 175
당신의 나이는 35살입니다.
당신의 체중은 75.0kg입니다.
당신의 신장은 175.0cm입니다.
```

Integer.parseInt()

- 기본형의 Wrapper 클래스는 문자열을 값으로 변환하는 메소드를 제공
  - 이러한 parseXXX() 형태로 구성
    - 예를 들어 정수로 구성된 문자열을 정수값으로 변환해주는 함수는 Integer.parseInt()
    - Short.parseShort(), Double.parseDouble(), Boolean.parseBoolean() 등이 있음

```
Scanner scanner = new Scanner(System.in);
String name = scanner.next();                  // “Kim"
String addr = scanner.next();                  // "Seoul"
String ageStr = scanner.next();                // "33"
String weightStr = scanner.next();             // "65.1"
int age = Integer.parseInt(ageStr);            // 33
double weight = Double.parseDouble(weightStr); // 65.1
```

예제 : Scanner를 이용한 키 입력

```
import java.util.Scanner;
public class ScannerExam {
  public static void main (String args[]) {
    Scanner a = new Scanner(System.in);
    System.out.println("나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요");
    System.out.println("당신의 나이는 " + Integer.parseInt(a.next()) + "살입니다.");
    System.out.println("당신의 체중은 " + Double.parseDouble(a.next()) +"kg입니다.");
    System.out.println("당신의 신장은 " + Double.parseDouble(a.next()) + "cm입니다.");
  }
}
```

```
나이, 체중, 신장을 빈칸으로 분리하여 순서대로 입력하세요
35 75 175
당신의 나이는 35살입니다.
당신의 체중은 75.0kg입니다.
당신의 신장은 175.0cm입니다.
```