

# 자바프로그램구조

# 2

자바 기본 프로그래밍

# 자바 프로그램 구조

2

```
/*  
 * Hello.java  
 */
```

```
public class Hello2 {
```

```
    public static int sum(int n, int m) {  
        return n + m;  
    }
```

메소드

```
    // main() 메소드에서 실행 시작
```

```
    public static void main(String[] args) {  
        int i = 20;  
        int s;  
        char a;
```

```
        s = sum(i, 10); // sum() 메소드 호출  
        a = '?';
```

```
        System.out.println(a); // 문자 '?' 화면 출력
```

```
        System.out.println("Hello2"); // "Hello2" 문자열 화면 출력
```

```
        System.out.println(s); // 정수 s 값 화면 출력
```

```
    }
```

```
}
```

클래스

```
?  
Hello2  
30
```

클래스 안에 포함되어  
있는 것을 사용하는 방법

메소드/메서드  
(Method)

# 자바 프로그램 구조 - 설명

3

## □ 클래스 만들기

- Hello2 이름의 클래스 선언

```
public class Hello2 {  
}
```

- class 키워드로 클래스 정의(4장 참고)
- public으로 선언하면 다른 클래스에서도 접근 가능
- 클래스 코드는 {} 내에 모두 작성

## □ main() 메소드

- public static void으로 선언되어야 함

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

- 자바 프로그램은 main() 메소드부터 실행 시작
- String[] args로 실행 인자를 전달 받음(3장 참고)

## □ 멤버 메소드

- 메소드 sum()

```
public static int sum(int n, int m) {  
    ...  
}
```

- 클래스에 속한 함수로서, 클래스 내에서만 선언

## □ 변수 선언

- 변수 타입과 변수 이름 선언

```
int i=20;  
int s;  
char a;
```

- 메소드 내에서 선언된 변수는 지역 변수

- 지역 변수는 메소드 실행이 끝나면 저장 공간 반환

## □ 메소드 호출

- sum() 메소드 호출

```
s = sum(i, 10); // 메소드 sum() 호출
```

- sum() 메소드의 호출 시 변수 i의 값과 정수 10을 전달
- sum() 메소드의 n, m에 각각 20, 10 값 전달
- sum() 메소드는 n과 m 값을 더한 30 리턴
- 변수 s는 정수 30을 전달받아 저장

# 자바 프로그램 구조 - 설명

4

## □ 주석문

- //을 만나면 한 라인으로 주석으로 처리
- /\* 와 \*/ 사이의 여러 행을 주석으로 처리

## □ 화면 출력

- 표준 출력 스트림에 메시지 출력

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

- 표준 출력 스트림 System.out의 println() 메소드 호출
- println()은 여러 타입의 데이터 출력 가능
- println()은 출력 후 다음 행으로 커서 이동

## □ 문장

- ;로 한 문장의 끝을 인식

```
int i=20;
b = '?';
s = sum(i, 20);
```

- 한 문장을 여러 줄에 작성해도 무방

```
b
= '?';
```

- 주석문 끝에는 ';'를 붙이지 않음

## □ 블록

- 블록은 { 로 시작하여 } 로 끝남

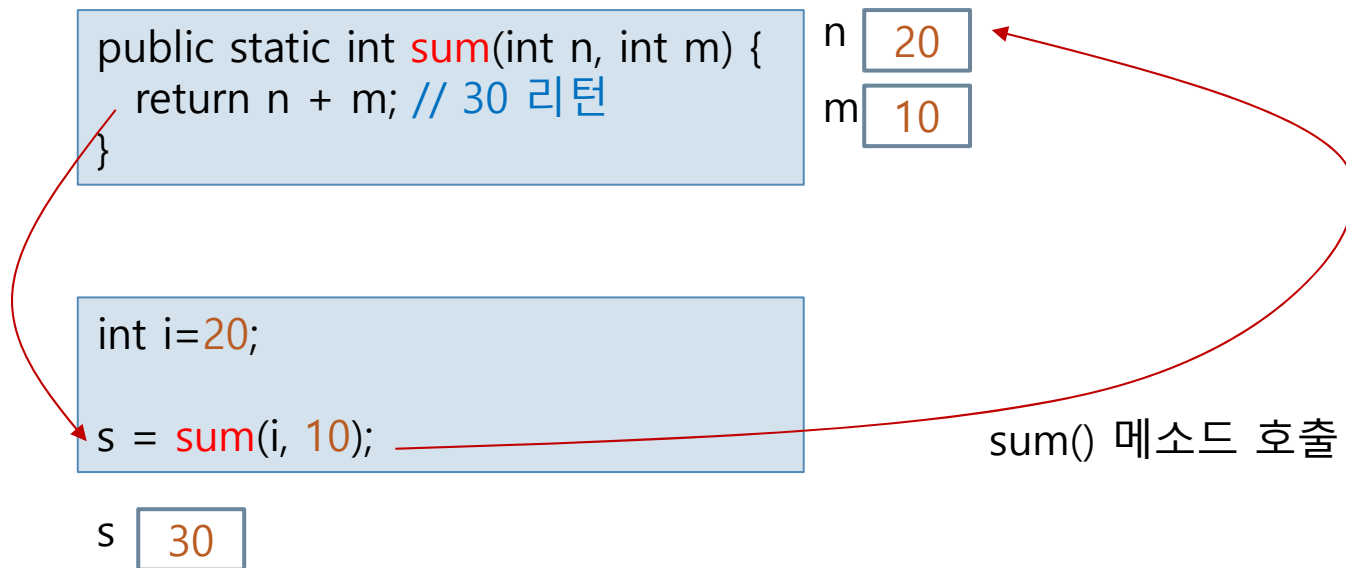
```
public class Hello2 {
    ...
} // Hello2 클래스 끝

public static void main(String[] args) {
    ...
} // main() 코드 끝
```

- 클래스와 메소드는 모두 블록으로 구성

# 자바 프로그램 구조 - 설명

5



# 식별자 (identifier)

6

## □ 식별자란?

- ▣ 클래스, 변수, 상수, 메소드 등에 붙이는 이름

## □ 식별자의 원칙

- ▣ '@', '#', '!'와 같은 특수 문자, 공백 또는 탭은 사용불가
- ▣ '\_', '\$'는 **사용 가능**, 일반적으로 잘 사용하지는 않음
- ▣ 유니코드 문자 사용 가능. 한글 사용 가능
- ▣ 자바 언어의 키워드는 식별자로 사용불가
- ▣ 식별자의 첫 번째 문자로 **숫자는 사용불가**
- ▣ 불린 리터럴 (true, false)과 널 리터럴(null)은 식별자로 사용불가
- ▣ 길이 제한 없음

## □ 대소문자 구별

- ▣ Test와 test는 별개의 식별자

# 식별자 이름 사례

7

## □ 사용 가능한 예

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

## □ 잘못된 예

```
int    3Chapter;             // 식별자의 첫문자로 숫자 사용 불가
class  if { }                 // 자바의 예약어 if 사용 불가
char   false;                 // false 사용 불가
void   null() { }             // null 사용 불가
class  %calc { }              // '%'는 특수문자
```

# 자바 키워드

8

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 관습

9

- 기본 : **헝그리안 이름 붙이기**

- **클래스 이름**

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

- 첫 번째 문자는 대문자로 시작

- 여러 단어가 복합될 때 각 단어의 첫 번째 문자만 대문자

- **변수, 메소드 이름**

```
int iAge;           // iAge의 i는 int의 i를 표시  
boolean blsSingle;  // blsSingle의 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

- **상수 이름**

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

- 모든 문자를 대문자로 표시

# 자료형과 변수

# 2

자바 기본 프로그래밍

# 자바의 데이터 타입

11

- 자바의 데이터 타입 => 마트 우유 용량
  - ▣ 기본 타입 : 8 개 => 우유, 과자, 빵, 과일, 옷, 약
    - boolean <- 참, 거짓
    - char <- 문자 1개 A
    - byte
    - short
    - int
    - long

} 정수 (음수, 0, 양수 )

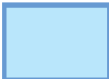
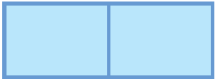
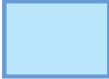
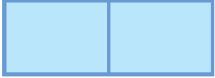
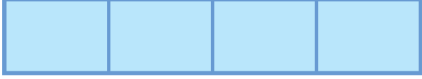



  - float <- 실수 1.2 8.235768
  - double <- 실수 1.2222222222
- ▣ 레퍼런스 타입 : 1 개이며 용도는 다음 3 가지
  - 클래스(class)에 대한 레퍼런스
  - 인터페이스(interface)에 대한 레퍼런스
  - 배열(array)에 대한 레퍼런스

# 자바의 기본 데이터 타입

12

## □ 특징

- 기본 데이터 타입의 크기가 정해져 있음
- 기본 데이터 타입의 크기는 CPU나 운영체제에 따라 변하지 않음

논리 타입	boolean		(1바이트, true 또는 false)
문자 타입	char		(2바이트, Unicode)
정수 타입	byte		(1바이트, -128~127)
	short		(2바이트, -32,768~32,767)
	int		(4바이트, $-2^{31} \sim 2^{31}-1$ )
	long		(8바이트, $-2^{63} \sim 2^{63}-1$ )
실수 타입	float		(4바이트, $-3.4\text{E}38 \sim 3.4\text{E}38$ )
	double		(8바이트, $-1.7\text{E}308 \sim 1.7\text{E}308$ )

# 자바 프로그램 구조

13

정수 5 저장  
0x00000010에 5저장해줘  
num이라는 메모리 공간에  
5를 저장해줘

**int** password

0x00000010에 저장된  
데이터를 가져와줘  
Num에 저장된 데이터  
가져와줘

**byte** num

**String** name

## 메모리 - 스택

0xffffffff	
	123456677
0x00000011	5
0x00000010	12
0x00000001	
0x00000000	10

# 변수와 선언

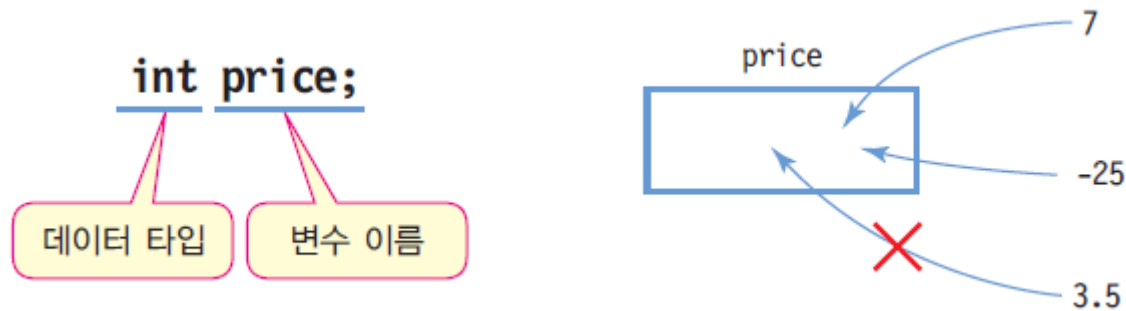
14

## □ 변수

- ▣ 프로그램 실행 중에 값을 **임시 저장하기 위한 공간**
  - 변수 값은 프로그램 수행 중 변경될 수 있음
- ▣ **데이터 타입에서 정한 크기의 메모리 할당**

## □ 변수 선언

- ▣ 변수의 타입 다음에 변수 이름을 적어 변수를 선언



# 변수 선언 사례

15

## □ 변수 선언 사례

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

## □ 변수 선언과 초기화

### ▣ 선언과 동시에 초기값 지정

```
int radius = 10;  
char c1 = 'a', c2 = 'b', c3 = 'c';  
double weight = 75.56;
```

## □ 변수에 값 대입

### ▣ 대입 연산자인 = 다음에 식(expression)

```
radius = 10 * 5;  
c1 = 'r';  
weight = weight + 5.0;
```

# 리터럴과 정수 타입 리터럴

16

## □ 리터럴(literal)

- 프로그램에서 직접 표현한 값
- 정수, 실수, 문자, 논리, 문자열 리터럴 있음

## □ 정수 타입 리터럴

- 8진수 : 0으로 시작
  - `int n = 015;` // 10진수로 13
- 16진수 : 0x로 시작
  - `int n = 0x15;` // 10진수로 21
- 10진수 : 0으로 시작하지 않는 숫자
  - 15, 3, 20, 55, 88
- 2진수 : 0b로 시작
  - `int n = 0b0101;` // 이진수 0101 -> 십진수 5
- 모든 정수 리터럴은 int 형으로 컴파일
- long 타입 리터럴은 숫자 뒤에 L 또는 l을 붙여 표시
  - ex) 24L, 3578l



# 실수 타입 리터럴

17

- 부동 소수점 실수 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 컴파일

# 문자 타입 리터럴

18

- 단일 인용부호(' ')로 문자 표현
  - 'a', 'W', '가', '\*', '3', '7'
- `\u`다음에 4자리 16진수로, 2 바이트의 유니코드(Unicode)
  - `\u0041` -> 문자 'A'의 유니코드(0041)
  - `\uae00` -> 한글문자 '글'의 유니코드(ae00)
- 특수 기호는 `\`로 시작 ➔ 이스케이프 문자

특수문자 리터럴 (이스케이프 시퀀스)	의미
<code>'\b'</code>	백스페이스(backspace)
<code>'\t'</code>	탭(tab)
<code>'\n'</code>	라인피드(line feed)
<code>'\f'</code>	폼피드(form feed)
<code>'\r'</code>	캐리지 리턴(carriage return)
<code>'\"'</code>	이중 인용부호(double quote)
<code>'\''</code>	단일 인용부호(single quote)
<code>'\\'</code>	백슬래시(backslash)

# 논리 타입 리터럴

19

## □ 논리 값 표시

- ▣ true 또는 false 뿐

```
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 오류
```

- ▣ (i==1) 또는 (i!=0)과 같은 명확한 논리 연산 사용해야 함

# Tip: 기본 타입 이외 리터럴

20

## □ null 리터럴

- 어떠한 레퍼런스 타입의 값으로도 사용 가능
  - ~~int n = null;~~ // 기본 데이터 타입에는 사용 불가
  - String str = null;

## □ 문자열 리터럴

- 이중 인용부호로 묶어서 표현
  - "Good", "Morning", "자바", "3.19", "26", "a"
- 자바에서 **문자열은 객체이므로 기본 타입 아님**
- 문자열 리터럴은 String 객체로 자동 처리

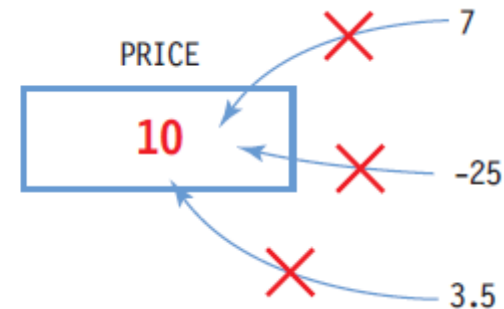
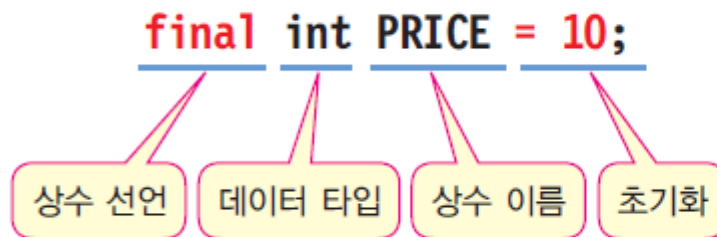
```
String str1 = "Welcome";  
String str2 = null;  
System.out.println(str1);
```

# 상수

21

## □ 상수 선언

- ▣ **final 키워드** 사용
- ▣ 선언 시 초기값 지정
- ▣ 실행 중 값 변경 불가



## □ 상수 선언 사례

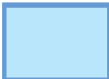
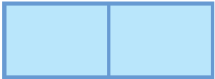
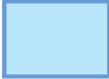
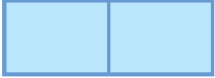
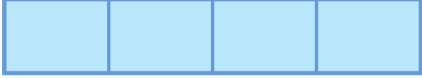



```
final double PI = 3.141592;  
final int LENGTH = 20;
```

# 자바의 기본 데이터 타입

22

## □ 특징

- 기본 데이터 타입의 크기가 정해져 있음
- 기본 데이터 타입의 크기는 CPU나 운영체제에 따라 변하지 않음

논리 타입	boolean		(1바이트, true 또는 false)
문자 타입	char		(2바이트, Unicode)
정수 타입	byte		(1바이트, -128~127)
	short		(2바이트, -32,768~32,767)
	int		(4바이트, $-2^{31} \sim 2^{31}-1$ )
	long		(8바이트, $-2^{63} \sim 2^{63}-1$ )
실수 타입	float		(4바이트, $-3.4\text{E}38 \sim 3.4\text{E}38$ )
	double		(8바이트, $-1.7\text{E}308 \sim 1.7\text{E}308$ )

# 타입 변환 - 자동 타입 변환

23

## □ 자동 타입 변환이 발생하는 경우

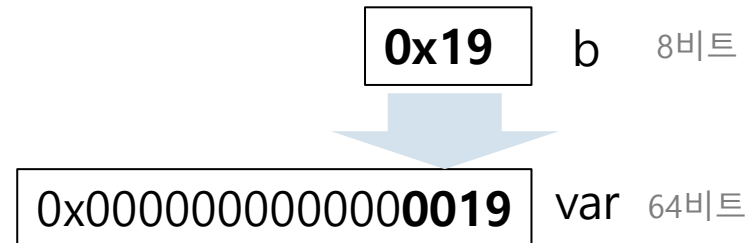
### ▣ 원래의 타입보다 큰 타입으로 바뀔 때

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

### ▣ 원본 값 보존

```
long var;  
byte b = 25; // 0x19  
var = b;
```

롱타입 변수      자동타입변환      바이트타입 변수

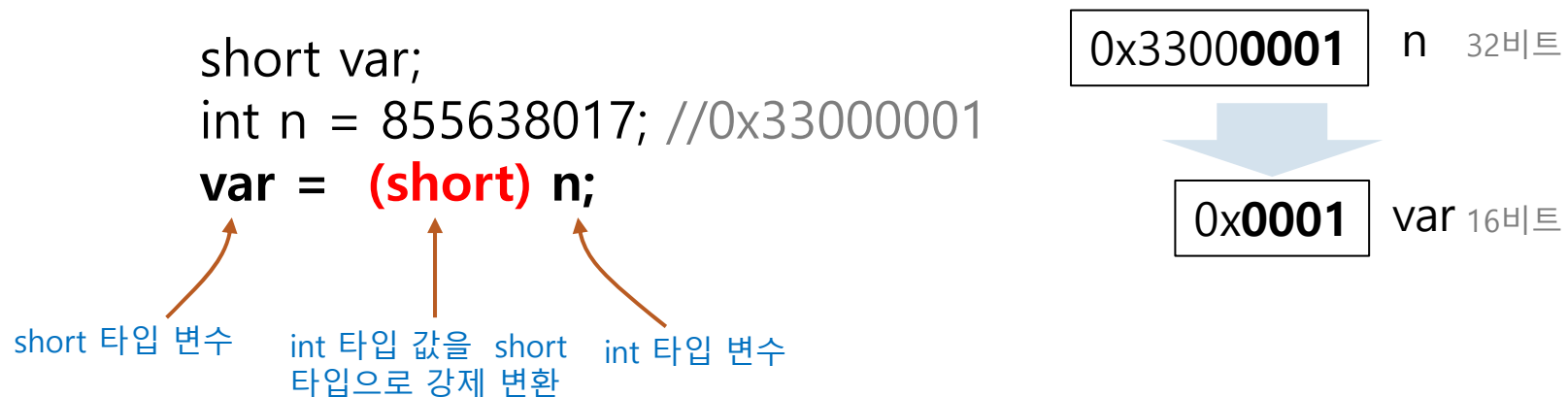


```
long var;  
int n = 32555;  
byte b = 25; // 0x19  
var = n; // int 타입에서 long 타입으로 자동 변환. var 값은 32555  
var = b; // byte 타입에서 long 타입으로 자동 변환. var 값은 25
```

# 강제 타입 변환 => 타입캐스팅

24

- 강제 타입 변환 : 개발자의 의도적으로 타입 변환
  - ▣ 개발자가 코드에 명시적으로 타입 변환 지정
    - 실수 타입이 정수 타입으로 강제 변환되면 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.
```





입력

2

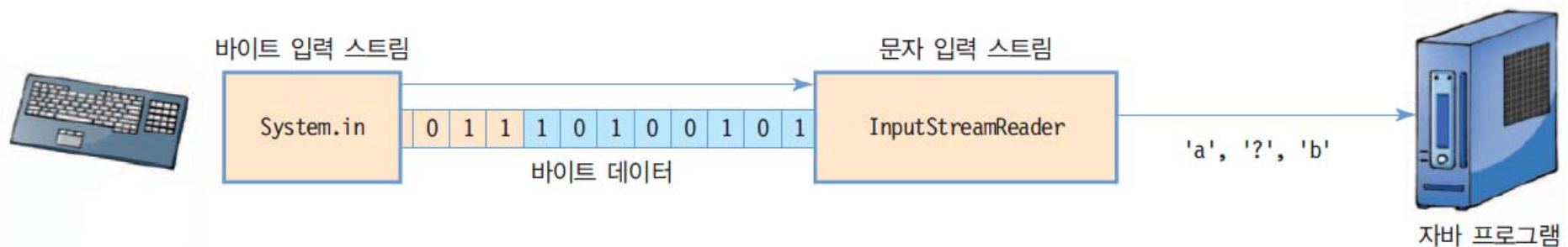
자바 기본 프로그래밍

# 자바에서 키 입력, InputStreamReader

26

- System.in
  - ▣ 키보드로부터 읽는 자바의 표준 입력 스트림
  - ▣ 읽은 키 값을 바이트(문자 아님)로 리턴
- 키보드로부터 문자 읽기 - InputStreamReader 클래스 이용
  - System.in에게 키를 읽게 하고, 읽은 바이트를 문자로 변환

```
InputStreamReader rd = new InputStreamReader(System.in);  
int a = rd.read(); // a는 키보드로부터 읽은 문자
```



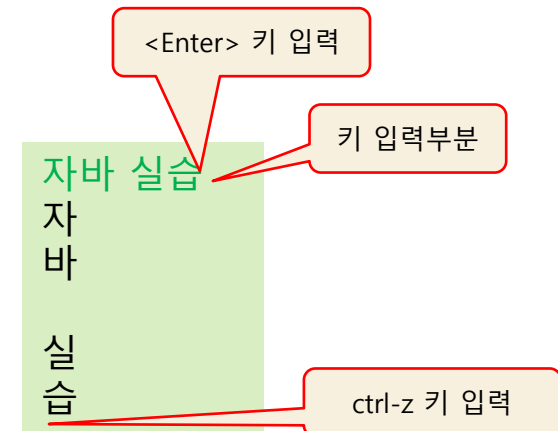
- ▣ 키 입력 동안 문제가 발생하면 IOException 발생
  - try-catch를 이용한 예외 처리 필요

# 예제 : 키보드로부터 문자 입력, 화면 출력

27

System.in을 InputStreamReader에 연결하여 사용자로부터 키 입력.  
입력 받은 문자를 화면에 출력하고, ctrl-z 키를 누르면 읽기 종료

```
import java.io.*;
public class InputExample {
    public static void main (String args[]) {
        InputStreamReader rd = new InputStreamReader(System.in);
        try {
            while (true) {
                int a = rd.read();
                if (a == -1) // ctrl-z가 입력되면 read()는 -1을 리턴
                    break;
                System.out.println((char)a); // 입력된 문자 출력
            }
        }
        catch (IOException e) {
            System.out.println("입력 에러 발생");
        }
    }
}
```



# Scanner를 이용한 키 입력 - 강추

28

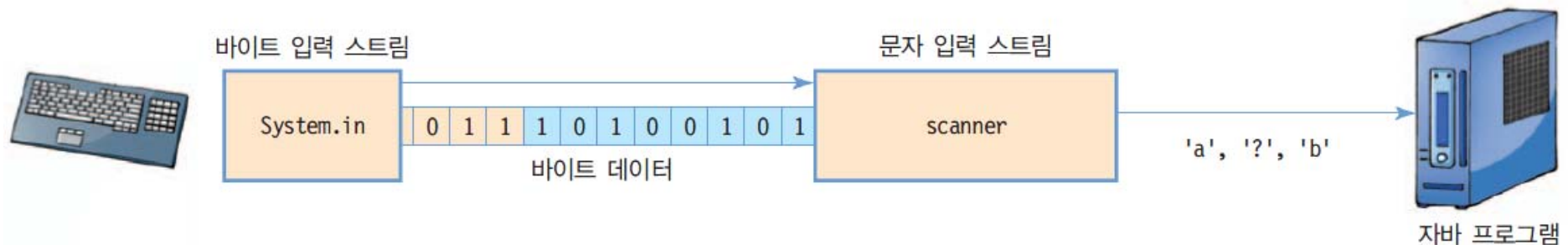
## □ Scanner 클래스

- ▣ InputStreamReader보다 쉬운 방법

- ▣ java.util.Scanner 클래스

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

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



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

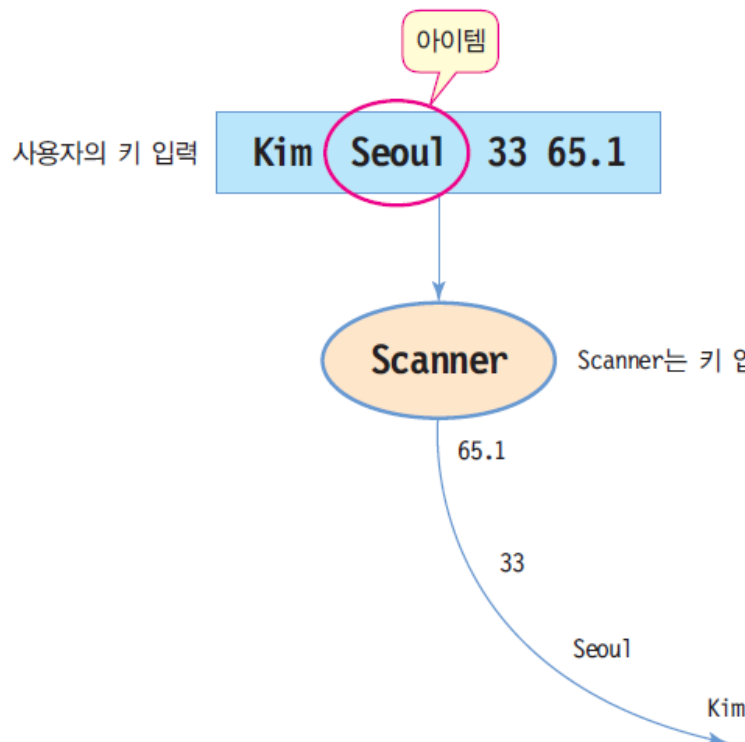
- ▣ import문 필요

- 소스 맨 앞줄에 사용

# Scanner를 이용한 키 입력

29

- Scanner에서 키 입력 받기
  - Scanner는 입력되는 키 값을 공백으로 구분되는 아이템 단위로 읽음
  - 공백 문자 : 'wt', 'wf', 'wr', ' ', 'wn'
- 개발자가 원하는 다양한 타입 값을 쉽게 읽을 수 있음



```
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
```

Scanner는 키 입력을 공백 단위로 구분하여 읽는다.

# Scanner 주요 메소드

30

메소드	설명
<code>String next()</code>	다음 아이템을 문자열 타입으로 리턴한다.
<code>byte nextByte()</code>	다음 아이템을 <code>byte</code> 타입으로 리턴한다.
<code>short nextShort()</code>	다음 아이템을 <code>short</code> 타입으로 리턴한다.
<code>int nextInt()</code>	다음 아이템을 <code>int</code> 타입으로 리턴한다.
<code>long nextLong()</code>	다음 아이템을 <code>long</code> 타입으로 리턴한다.
<code>float nextFloat()</code>	다음 아이템을 <code>float</code> 타입으로 리턴한다.
<code>double nextDouble()</code>	다음 아이템을 <code>double</code> 타입으로 리턴한다.
<code>String nextLine()</code>	한 라인 전체('\n'까지)를 읽어 문자열 타입('\n' 미포함)으로 리턴한다.





# 연산자

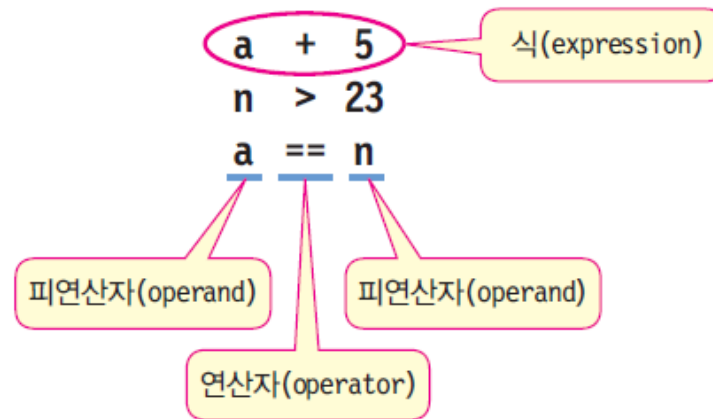
# 2

자바 기본 프로그래밍

# 식과 연산자

32

- 연산 : 주어진 식을 계산하여 결과를 얻어내는 과정




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



# 연산자 우선 순위

33

<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)
	= += -= *= /= %= &= ^=  = <<= >>= >>>=

- 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

# 산술 연산자

34

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

산술 연산자	의미	예	결과
+	더하기	<code>25.5 + 3.6</code>	29.1
-	빼기	<code>3 - 5</code>	-2
*	곱하기	<code>2.5 * 4.0</code>	10.0
/	나누기	<code>5/2</code>	2
%	나머지	<code>5%2</code>	1

# 비트 연산자

35

## □ 피 연산자의 각 비트들을 대상으로 하는 연산

비트 연산자	내용
$a \& b$	a와 b의 각 비트들의 AND 연산. 두 비트 모두 1일 때만 1이 되며 나머지는 0이 된다.
$a   b$	a와 b의 각 비트들의 OR 연산. 두 비트 모두 0일 때만 0이 되며 나머지는 1이 된다.
$a \wedge b$	a와 b의 각 비트들의 XOR 연산. 두 비트가 서로 다르면 1, 같으면 0이다.
$\sim a$	단항 연산자로서 a의 각 비트들에 NOT 연산. 1을 0으로, 0을 1로 변환한다.

# 비트 연산자의 사례

36

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

모두 1이므로  
결과는 1

둘 중 하나라도 0이 되면  
결과는 0

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

모두 0이므로  
결과는 0

둘 중 하나라도 1이 되면  
결과는 1

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

두 비트가  
모두 같으므로  
결과는 0

두 비트가  
서로 다르므로  
결과는 1

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

1은 0으로 변환

0은 1로 변환

# 시프트 연산자

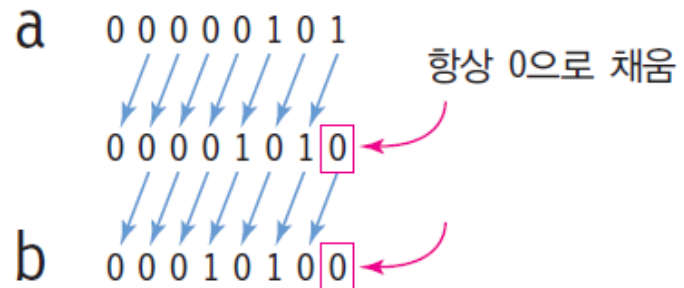
37

시프트 연산자	내용
$a \gg b$	a의 각 비트를 오른쪽으로 b번 시프트한다. 최상위 비트의 빈자리는 시프트 전의 최상위 비트로 다시 채운다. 산술적 오른쪽 시프트라고 한다.
$a \ggg b$	a의 각 비트를 오른쪽으로 b번 시프트한다. 그리고 최상위 비트의 빈자리는 0으로 채운다. 논리적 오른쪽 시프트라고 한다.
$a \ll b$	a의 각 비트를 왼쪽으로 b번 시프트한다. 그리고 최하위 비트의 빈자리는 0으로 채운다. 산술적 왼쪽 시프트라고 한다.

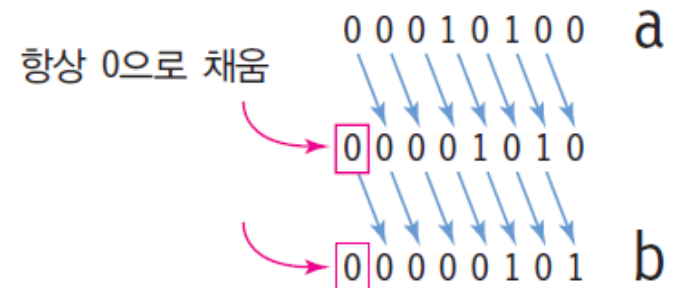
# 시프트 연산자의 사례

38

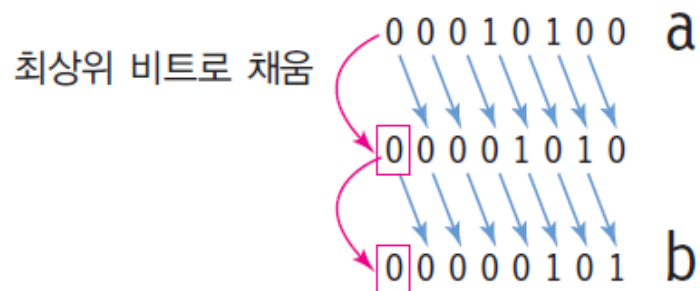
```
byte a = 5; // 5  
byte b = (byte)(a << 2); // 20
```



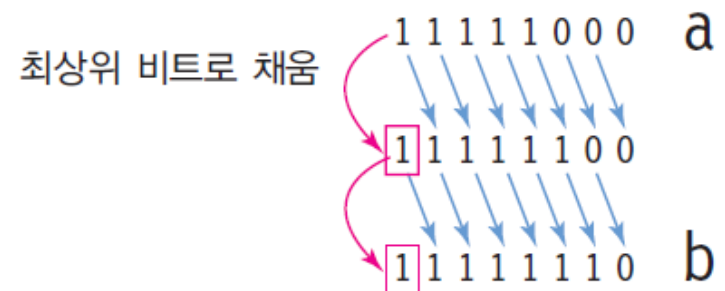
```
byte a = 20; // 20  
byte b = (byte)(a >> 2); // 5
```



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



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



# Tip: 산술적 시프트와 논리적 시프트

39

- 산술적 오른쪽 시프트
  - ▣  $>>$ 는 1비트 오른쪽으로 시프트 : 나누기 2의 결과
- 산술적 왼쪽 시프트
  - ▣  $<<$ 는 1비트 왼쪽 시프트 : 2로 곱하는 결과
  - ▣ 시프트 결과 음수(최상위 비트가 1)가 최상위 비트가 0인 양수가 되는 오버플로우 발생 가능 주의
- 논리적 오른쪽 시프트
  - ▣  $>>>$ 는 시프트 시 최상위 비트에 항상 0이 삽입
  - ▣ 나누기의 산술적 효과 없음
- byte, short, char 타입의 시프트 연산 시 주의 사항
  - ▣ int 타입으로 변환되어 연산, 원하지 않는 결과 발생 가능

# 비교연산자

40

비교 연산자	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
$a \leq b$	a가 b보다 작거나 같으면 true 아니면 false	$1 \leq 0$	false
$a \geq b$	a가 b보다 크거나 같으면 true 아니면 false	$10 \geq 10$	true
$a == b$	a가 b와 같으면 true 아니면 false	$1 == 3$	false
$a != b$	a가 b와 같지 않으면 true 아니면 false	$1 != 3$	true



# 논리 연산자

a	!a	예제
true	false	!(3 < 5)는 false
false	true	!(3 > 5)는 true

a	b	a ^ b	예제
true	true	false	(3 < 5) ^ (1 == 1)은 false
true	false	true	(3 < 5) ^ (1 == 2)은 true
false	true	true	(3 > 5) ^ (1 == 1)은 true
false	false	false	(3 > 5) ^ (1 == 2)은 false

a	b	a    b	예제
true	true	true	(3 < 5)    (1 == 1)은 true
true	false	true	(3 < 5)    (1 == 2)은 true
false	true	true	(3 > 5)    (1 == 1)은 true
false	false	false	(3 > 5)    (1 == 2)은 false

a	b	a && b	예제
true	true	true	(3 < 5) && (1 == 1)은 true
true	false	false	(3 < 5) && (1 == 2)은 false
false	true	false	(3 > 5) && (1 == 1)은 false
false	false	false	(3 > 5) && (1 == 2)은 false

# 대입 연산자, 증감 연산자

42

대입 연산자	내용
<code>a = b</code>	b의 값을 a에 대입
<code>a += b</code>	<code>a = a + b</code> 와 동일
<code>a -= b</code>	<code>a = a - b</code> 와 동일
<code>a *= b</code>	<code>a = a * b</code> 와 동일
<code>a /= b</code>	<code>a = a / b</code> 와 동일
<code>a %= b</code>	<code>a = a % b</code> 와 동일

대입 연산자	내용
<code>a &amp;= b</code>	<code>a = a &amp; b</code> 와 동일
<code>a ^= b</code>	<code>a = a ^ b</code> 와 동일
<code>a  = b</code>	<code>a = a   b</code> 와 동일
<code>a &lt;&lt;= b</code>	<code>a = a &lt;&lt; b</code> 와 동일
<code>a &gt;&gt;= b</code>	<code>a = a &gt;&gt; b</code> 와 동일
<code>a &gt;&gt;&gt;= b</code>	<code>a = a &gt;&gt;&gt; b</code> 와 동일

증감 연산자	내용
<code>a++</code>	a를 먼저 사용한 후에 1 증가
<code>a--</code>	a를 먼저 사용한 후에 1 감소
<code>++a</code>	a를 먼저 1 증가한 후에 사용
<code>--a</code>	a를 먼저 1 감소한 후에 사용

# 증감 연산자

43

## □ 증감 연산의 순서

### ▣ 연산자가 피연산자 뒤에 붙는 경우

```
int a, b = 4;  
a = b++;  
// 결과 a=4, b=5
```

### ▣ 연산자가 피연산자 앞에 붙는 경우

```
int a, b = 4;  
a = ++b;  
// 결과 a=5, b=5
```

# 조건 연산자 ?:

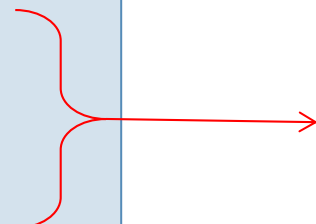
44

## □ opr1?opr2:opr3

- ▣ 세 개의 피연산자로 구성된 삼항(ternary) 연산자
  - opr1이 true이면, 연산식의 결과는 opr2, false이면 opr3
- ▣ if-else를 간결하게 표현할 수 있음

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

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



```
int s = (x>y)?1:-1;
```