

CH 2

변수

변수의 선언

1. 이유

2. 선언 방법

변수에 값 저장하기

1. 변수에 값 저장하기

2. 변수의 초기화

변수의 종류

변수의 값 읽어 오기

1. 변수의 값이 필요한 곳에 변수의 이름을 적는다

변수의 타입

1. 변수의 타입은 저장할 값의 타입에 의해 결정된다

2. 저장할 값의 타입과 일치하는 타입으로 변수를 선언한다

값의 타입

기본형

기본형과 참조형

기본형(Primitive type)

참조형(Reference type)

변수, 상수, 리터럴

변수(variable)

상수(constant)

리터럴(literal)

에러 메시지 보기

리터럴의 접두사와 접미사

변수와 리터럴의 타입 불일치

1. 범위가 '변수 > 리터럴' 인 경우, 가능하다.

2. 범위가 '변수 < 리터럴'인 경우, 에러가 발생한다.

3. byte, short 변수에 int 리터럴 저장 가능

문자와 문자열

두 변수의 값 교환하기

기본형의 종류와 크기

논리형

문자형

정수형

실수형

표현 범위

println()

단점

printf()
지시자

변수

- 하나의 값을 저장할 수 있는 메모리 공간

변수의 선언

1. 이유

- 값(data)을 저장할 공간을 마련하기 위해

2. 선언 방법

- 변수 타입 변수 이름 ;

예) int age = 15; ⇒ 정수 타입의 변수 age를 선언

변수에 값 저장하기

1. 변수에 값 저장하기

```
int age; // 정수(int) 타입의 변수 age를 선언
age = 25; // 변수 age에 25를 저장
// '='는 등호가 아니라 대입
int age = 25;
```

2. 변수의 초기화

- 변수에 처음으로 값을 저장하는 것

```
int x = 0; // 변수 x를 선언 후, 0으로 초기화
int y = 9; // 변수 y를 선언 후, 5로 초기화
int x = 0, y = 9; // 위의 두 줄을 한 줄로
```

변수의 종류

- 클래스 변수, 인스턴스 변수, 지역 변수



지역 변수는 읽기 전에 값을 꼭 초기화해야 한다 (0으로 자동 초기화 안 됨)

변수의 값 읽어 오기

1. 변수의 값이 필요한 곳에 변수의 이름을 적는다

```
int year = 0, age = 14;  
year = age + 2000;  
year = 14 + 2000;  
year = 2014;  
  
age = age + 1;           // 변수의 값을 1 증가시키는 방법  
age = 14 + 1;  
age = 15;  
  
System.out.println(age); // 변수 값 출력  
System.out.println(15); // 15가 화면에 출
```

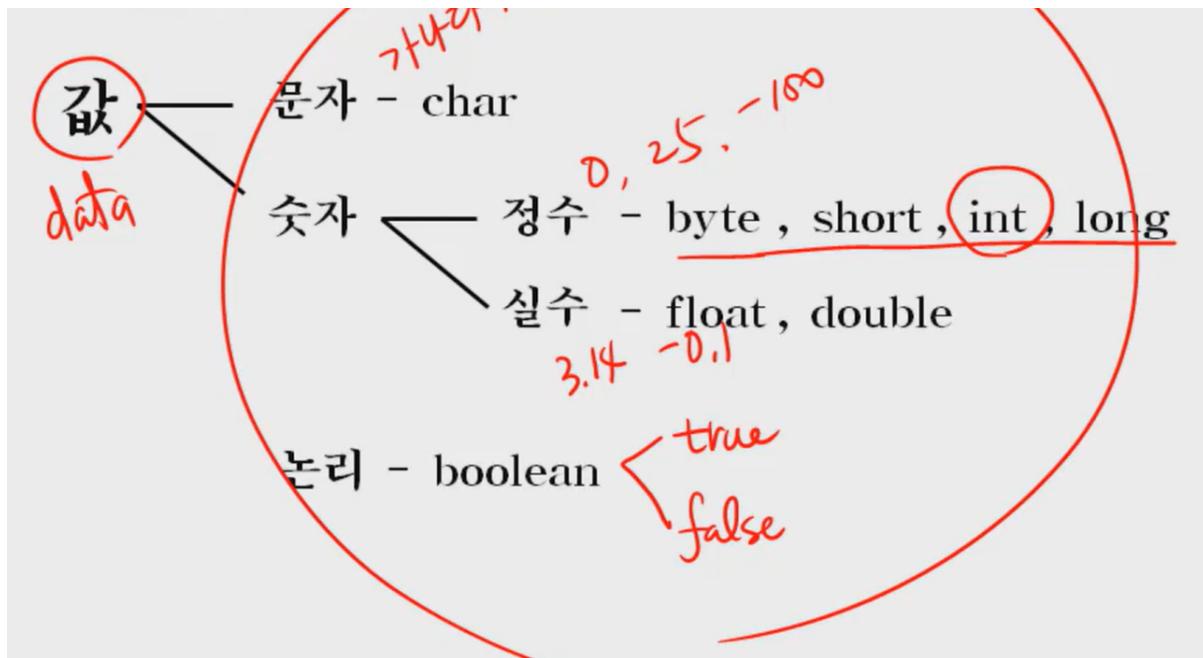
변수의 타입

1. 변수의 타입은 저장할 값의 타입에 의해 결정된다

2. 저장할 값의 타입과 일치하는 타입으로 변수를 선언한다

값의 타입

기본형



기본형과 참조형

기본형(Primitive type)

- 8개: boolean, char, byte, short, int, long, float, double
- 실제 값을 저장

참조형(Reference type)

- 기본형을 제외한 나머지: String, System
- 메모리 주소를 저장 (4byte or 8byte)
 - 4byte: 40억 → 32bit JVM
 - 8byte: 160경 → 64bit JVM

```
//참조형
Date today; // 참조형 변수 today 선언
// 객체 생
today = new Date(); // today에 객체의 주소를 저장
```

변수, 상수, 리터럴

변수(variable)

- 하나의 값을 저장하기 위한 공간 (변경 O)

상수(constant)

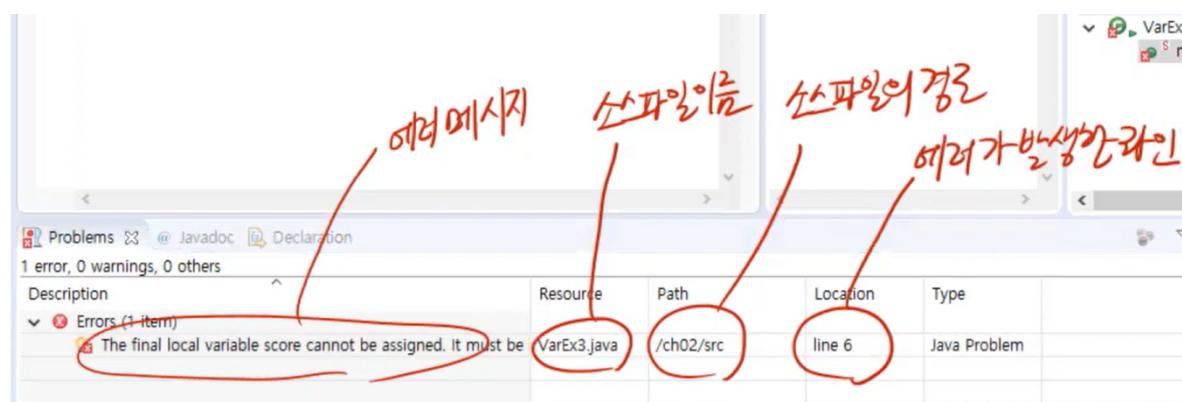
- 한 번만 값을 저장 가능한 변수 (변경 X)

리터럴(literal)

- 그 자체로 값을 의미하는 것
- 기존의 상수와 같은 개념

```
int score = 100;
score = 200;
final int MAX = 100; // Max는 상수
Max = 200; // 에러
char ch = 'A';
String str = "abc";
```

에러 메시지 보기



리터럴의 접두사와 접미사

종류	리터럴	접미사
논리형	false, true	없음
정수형	123, 0b(2진 접두사)0101, 077, 0xFF, 100L	L (long)
실수형	3.14, 3.0e8, 1.4f, 0x1.0p-1	f(float), d(생략 가능)
문자형	'A', '1', '\n' (개행 문자)	없음
문자열	"ABC", "123", "A"(문자열), "true"	없음

- 접미사는 대소문자 구별하지 않지만 소문자 l은 숫자 1과 헷갈리기 쉬우므로, 대문자 L을 사용

```

boolean power = true;
char ch = 'A';
String str = "ABC";
byte b = 127; // int 타입 byte: ~128 ~ 127까지 저장 가능

int i = 100; // 10진수
int oct = 0100; // 8진수
int hex = 0x100; // 16진수

long l = 10_000_000_000L; // 100억
// => Integer의 최대인 20억을 넘기 때문에 L 안 붙이면 에러
long l = 100; // L 생략 가능

float f = 3.14f; // 생략 불가
double d = 3.14d; // 생략 가능

```

```

(double) 10. -> 10.0
(double) .10 -> 0.10
(float) 10f -> 10.0f
(double) le3 -> 1000.0

```

변수와 리터럴의 타입 불일치

1. 범위가 '변수 > 리터럴' 인 경우, 가능하다.

- 변수: 그릇
- 리터럴: 물건

```

int i = 'A' (65) ; // int > char
long l = 123; // long > int
double d = 3.14f; // double > float

```

2. 범위가 '변수 < 리터럴'인 경우, 에러가 발생한다.

```

int i = 30_0000_0000; // int의 범위인 (-20억~20억) 벗어나서 에러
long l = 3.14f // long < float 에러
float f = 3.14; // float < double 에러

```

3. byte, short 변수에 int 리터럴 저장 가능

- 단, 변수 타입 범위 이내여야 한다

```
byte b = 100; (int) // byte의 범위 ~128 ~ 127에 속하므로 값이 더 작아도 저장 가능
byte b = 128 // 에러 발생
```

문자와 문자열

```
char ch = 'A';
char ch = 'AB'; // 에러
String s = "ABC" ; // 문자열

String s1 = "AB"; // 클래스
String s2 = new String("AB");

// 문자열은 원래 연속된 여러 문자를 의미하지만, 문자가 하나거나 빈 문자열이어도 괜찮다.

String s3 = "A";
String s4 = ""; // 빈 문자열
char ch = '' ; // 에러

String s5 = "A" + "B"; // 문자열 결합
String s5 = "AB"

"" + 7 = "7"; // 숫자 + 빈 문자열을 하면 문자열이 된다 (숫자 -> 문자열)
"" + 7 + 7 -> "7" + 7 -> "7" + "7" -> "77"
7 + 7 + "" -> 14 + "" -> "14" + "" -> "14"

문자열 + 다른 타입 -> 문자열
다른 타입 + 문자열 -> 문자열
```

두 변수의 값 교환하기

```
int x = 10, y = 20;
int tmp;

tmp = x;
x = y;
y = tmp;
```

1. x의 값을 tmp에 저장
2. y의 값을 x에 저장
3. tmp의 값을 y에 저장

기본형의 종류와 크기

논리형

- boolean
- true와 false 중 하나를 값으로 가지며, 조건식과 논리적 계산에 사용

문자형

- char
- 문자를 저장하는 데 사용되며, 변수당 하나의 문자를 저장 ↔ 여러 문자: string

정수형

- 주로 사용하는 것: int, long
- byte: 이진 데이터
- short: C언어와의 호환을 위해 추가 (잘 안 쓰임)
- 정수 값 저장하는 데 사용

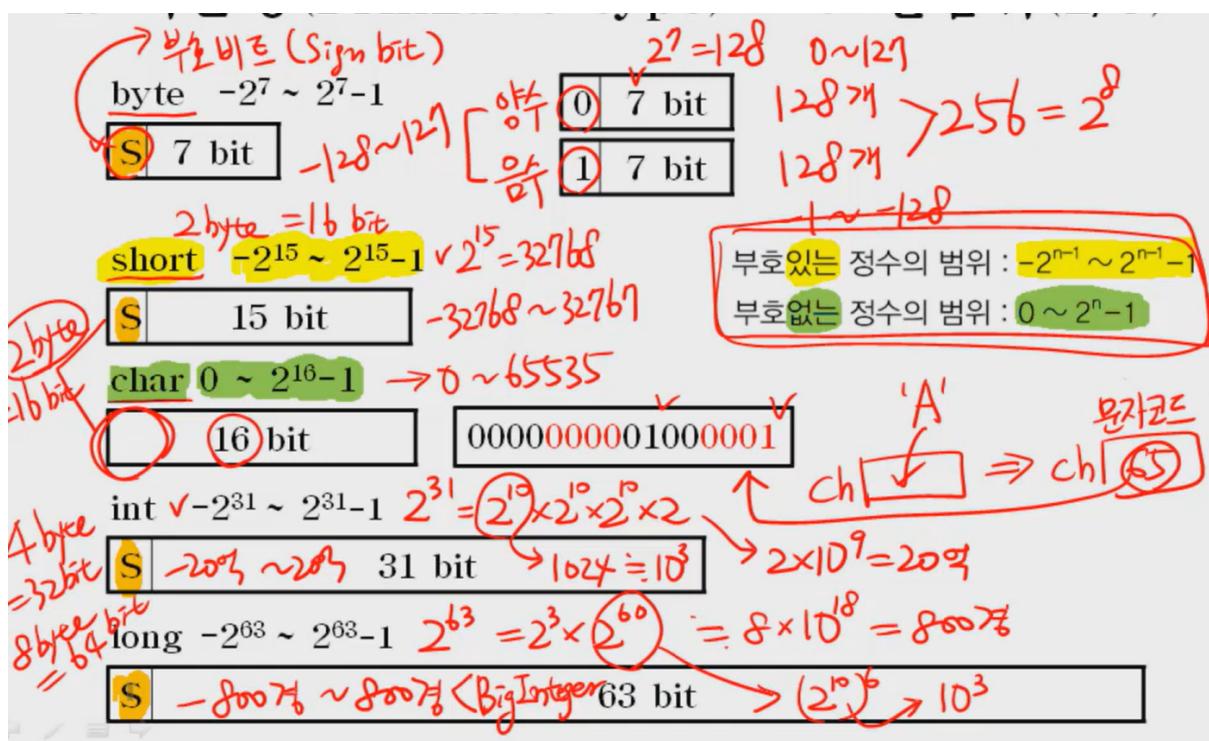
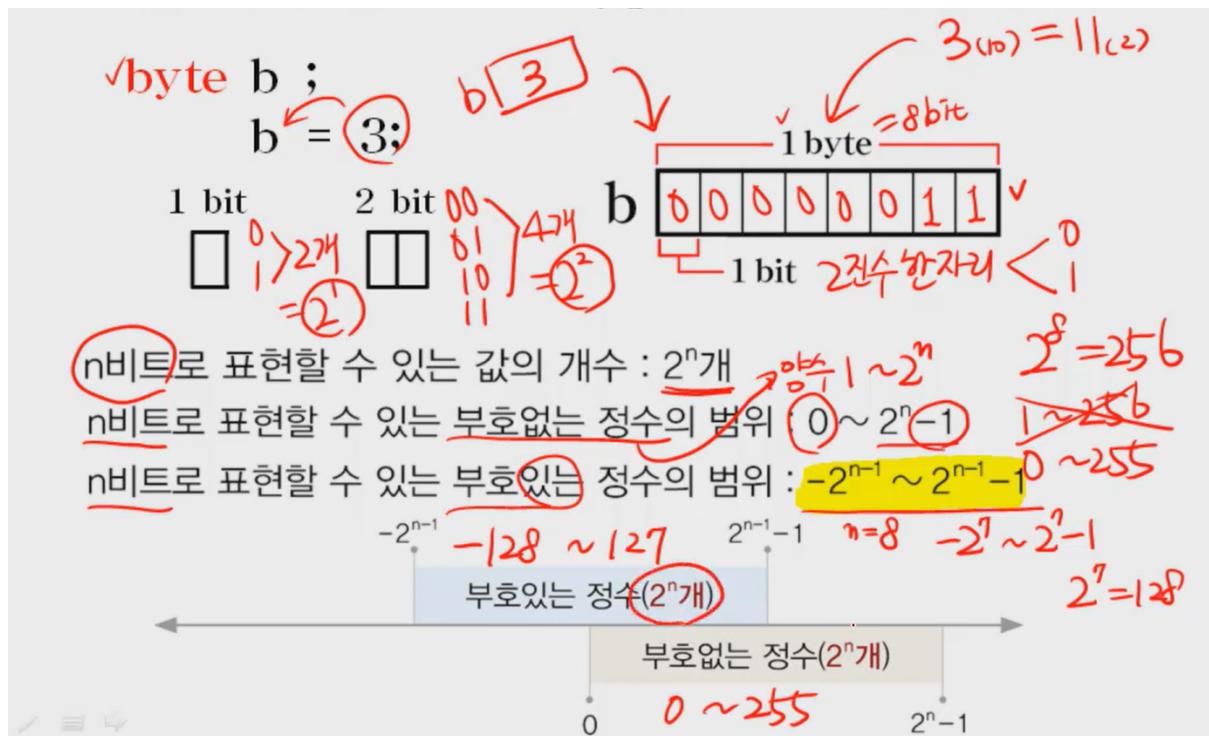
실수형

- float, double
- 실수 값 저장하는 데 사용

종류	크기 (byte)	1bit = 2진수 1자리	1byte = 8bit	2	4	8	浮
논리형	boolean	true false		무기고드 (2byte)			부동소수점 (floating point)
문자형		char					
정수형	byte		short 2	int 4	long 8		
실수형			float			double 16	

표현 범위

- 문자형과 정수형



- 실수

오차 있는 자리수

자료형	저장 가능한 값의 범위 (양수)	정밀도	크기
float	$1.4 \times 10^{-45} \sim 3.4 \times 10^{38}$	10^{38} 7 자리 $\times 2$	32 bit
double	$4.9 \times 10^{-324} \sim 1.8 \times 10^{308}$	10^{38} 15 자리	64 bit

값 범위: $-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$

부호: 자수: 가수: 0

float: $1+8+23=32$ bit = 4 byte

double: $1+11+52=64$ bit = 8 byte

$10^{-1} < 2^{24} < 10^8$

표현 할 수 없는 범위:

println()

단점

- 출력 형식 지정 불가
- 실수의 자릿수 조절 불가
 - 소수점 n자리만 출력하는 것이 불가능

```
System.out.println(10.0/3); // 3.3333333...
System.out.println(10/3); // 3
```

- 10진수로만 출력

- 8진수, 16진수 출력 불가

```
System.out.println(0x1A); // 26 (10진수)
```

printf()

- 형식화된 출력

- 출력 형식 지정 가능

```
// %. 지시자
System.out.printf("%.2f", 10.0/3); // 3.33 -> 소수점 둘째 자리까지만 출력하여라
System.out.printf("%d", 0x1A); // 26 -> 10진수
System.out.printf("%X", 0x1A); // 1A -> 16진수
```

지시자

지시자	설명
%b	불리언(boolean) 형식으로 출력
%d	10진(decimal) 정수의 형식으로 출력
%o	8진(octal) 정수의 형식으로 출력
%x, %X	16진(hexa-decimal) 정수의 형식으로 출력
%f	부동 소수점(floating-point)의 형식으로 출력
%e, %E	지수(exponent) 표현식의 형식으로 출력
%c	문자(character)로 출력
%s	문자열(string)로 출력

- JavaAPI → Formatter 들어가면 나머지 지시자 더 볼 수 있음
- 여러 개의 지시자를 한꺼번에 사용할 수도 있다
- `\n, %n`: 개행 문자 (줄바꿈)

```
System.out.printf("age:%d year:%d\n", 14, 2017); //age: 14 year:2017 출력
```

- printf는 줄바꿈을 자동으로 하지 않는다

1. 정수를 10진수, 8진수, 16진수로 출력

```
System.out.printf("%d", 15); // 15 10진수 (decimal)
System.out.printf("%o", 15); // 17 8진수 (octal)
System.out.printf("%x", 15); // f 16진수 (hexa)
System.out.printf("%s", Integer.toBinaryString(15)); // 1111 2진수 (2진 문자열로 변환)
```

2. 8진수(0)와 16진수(0x)에 접두사 붙이기

- 앞에 `#` 을 붙이면 접두사가 붙어서 출력된다

```
System.out.printf("%#o", 15); // 017
System.out.printf("%#x", 15); // 0xf
System.out.printf("%#X", 15); // 0XF
```

3. 실수 출력을 위한 지시자 %f

- 지수 형식: %e
- 간략한 형식: %g ⇒ %f와 %e 중 더 간략하게 표현할 수 있는 것을 골라서 출력

```
float f = 123.4567890f;
// float는 정밀도가 7자리기 때문에 앞 7자리까지만 값이 정확
// 여기서 87은 의미 없는 숫자
// 웬만하면 double을 사용하기
System.out.printf("%f", f); // 123.456787 소수점 아래 6자리
// e+02는 102
// 반올림했기 때문에 위와 값이 조금 다름
System.out.printf("%e", f); // 1.234568e+02 지수 형식

System.out.printf("%g", 123.456789); // 123.457 간략한 형식 (%f처럼)
System.out.printf("%g", 0.00000001); // 1.00000e-8 간략한 형식 (%e)
```

```
// 다섯 자리 출력
System.out.printf("[%5d]\n", 10); // [ 10 ] 앞에서부터 12345
// 왼쪽 정렬
System.out.printf("[-5d]\n", 10); // [10 ]
System.out.printf("[05d]\n", 10); // [00010]
// 자릿수보다 출력할 값이 더 크면 잘리지 않고 출력된다
System.out.printf("%5d\n", 1234567); // [1234567]
```

%전체자리.소수점아래자리 f

System.out.printf("d=814.10f\n", d); // 전체 14자리 중 소수점 아래 10자리



```
// 문자열 출력
System.out.printf("%s\n", url); // [www.codechobo.com]
System.out.printf("%20s\n", url); // [ www.codechobo.com ]
System.out.printf("%-20s\n", url); // [www.codechobo.com ]
System.out.printf("%.8s\n", url); // [www.code] 부분출력
```

