자료형

자료형-자료형의 개요

자료형-자료형의 개요

1 ☞ 자료형의 의미

- 저장할 수 있는 값의 형태를 지정
- Java 프로그램의 모든 변수/상수는 자료형 선언 후 사용가능

2

TIP

Java의 모든 변수/상수는 자료형이 먼저 선언되어야 함

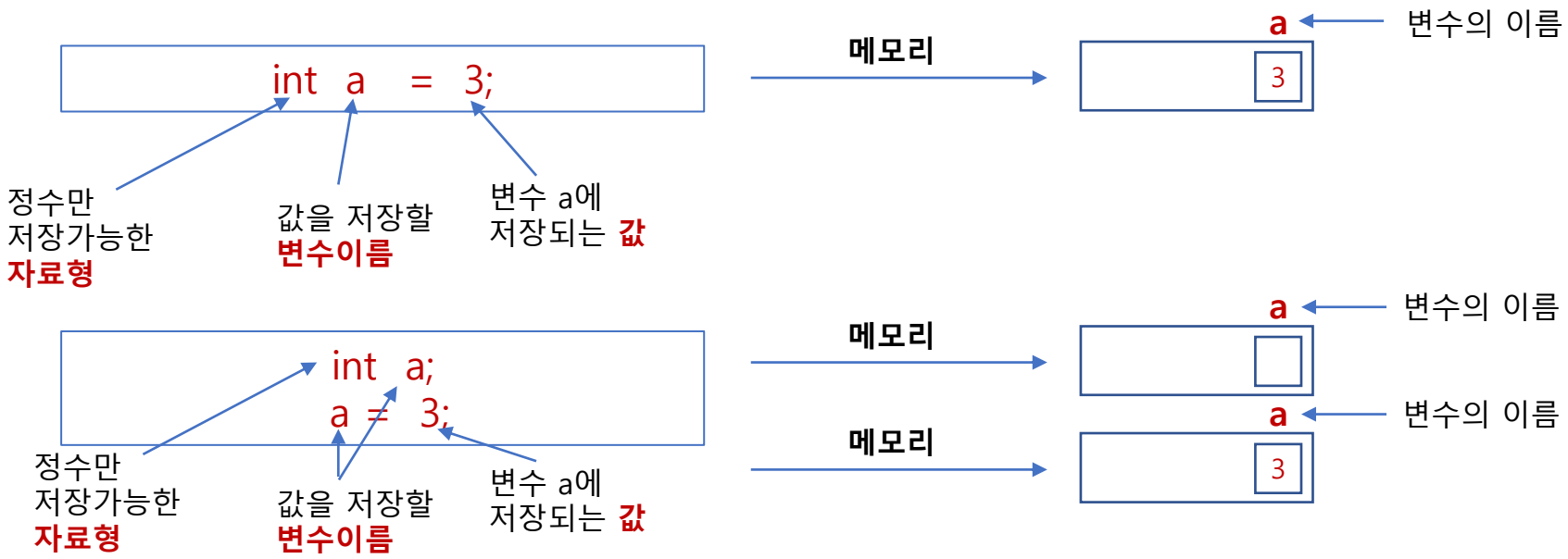
ex

int a = 3; //→(O)
b=5; //→(X)

String c;
c="안녕" //→(O)

☞ 자료형의 사용방법

3



자료형-자료형의 개요

1 ☞ 변수/상수/메서드 이름의 선정 규칙

- 변수/상수/메서드의 이름은 자유롭게 선택 가능
- 단, 선정규칙은 준수하여야 함

4

ex

```
boolean  aBcD;      (O) ← 가능하지만 권고위배
byte  가나다;       (O)
short  _abcd;       (O)
char  $ab_cd;       (O)
int  3abcd;         (X)
long  abcd3;        (O)
float  int;         (X)
double  main;       (O)
int  my Work;       (X)
String  ourClassNum;(O)
int ABC;            (O) ← 가능하지만 권고위배
```

5

ex

```
final double PI;   (O)
final int MY_DATA; (O)
final float myData;(O) ← 가능하지만 권고위배
```

2

상수이름 메서드이름

동일 동일

변수이름 선정 (필수사항)

- 영문대소문자 및 한글 사용 가능
- 특수문자는 두 가지만 표현 가능: '_', '$'
- 숫자 사용가능 단, 변수의 첫번째는 올 수 없음
- 자바에서 사용중인 키워드 사용 불가

3

변수이름 선정 (권고사항)

- 변수의 이름은 소문자로 시작
- 두개 이상의 단어가 결합된 경우 새 단어는 대문자로 시작

다름 → 상수이름

- 모든 문자를 대문자
- 두개 이상의 단어는 _로 연결

PI, MY_DATA

동일 → 메서드이름

자료형-자료형의 개요

☞ 변수의 생존기간

1

- 변수는 자신이 **선언**된 열린 중괄호('{')의 쌍이 되는 닫힌 중괄호('}') 안에서만 사용 가능

ex

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

  int value1 = 3;

  {
    int value2 =5;
    System.out.println(value1); //3
    System.out.println(value2); //5
  }

  System.out.println(value1);   //3
  //System.out.println(value2); //오류

}
```

2

3

TIP

실제 단순히 중괄호({ })만을 삽입하는 경우는 드물며
클래스/제어문에 포함된 { }내에서 변수를 선언하여 사용

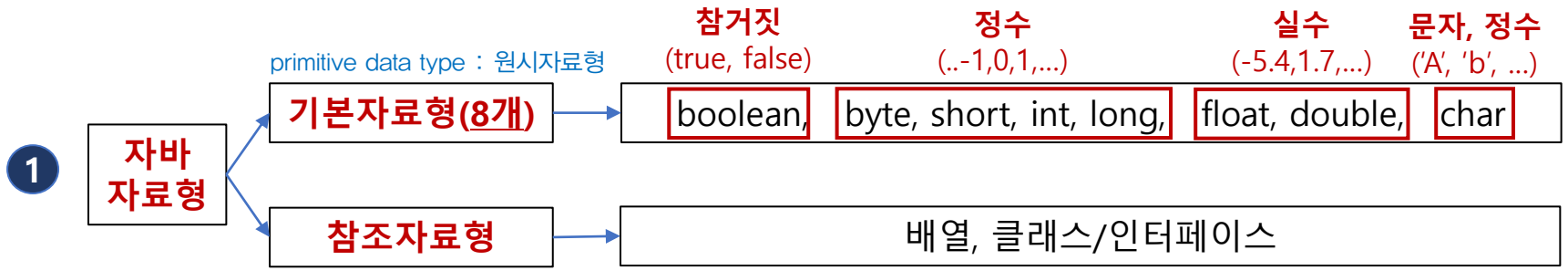
ex

```
int value1 = 3;
if(value1>1) {
  int value2 =5;
  System.out.println(value1); //3
  System.out.println(value2); //5
}

System.out.println(value1);   //3
//System.out.println(value2); //오류
```

자료형-자료형의 개요

☞ 자료형의 종류 (기본자료형 vs. 참조자료형)



2

TIP

참조자료형은 직접 정의할 수가 있어 무한개가 존재

3

TIP

기본자료형과 참조자료형의 차이점 #1 (필수가 아닌 권고사항)
- 기본자료형은 이름은 소문자로 시작 (ex. **i**nt, **d**ouble ...)
- 참조자료형은 이름은 대문자로 시작(권고) (ex. **S**tring, **S**ystem.)

자료형-자료형의 개요

☞ 자료형의 종류 (기본자료형 vs. 참조자료형)

- 메모리의 구조

1

class 영역 static 영역 final 영역 메서드 영역	Stack 영역	Heap 영역

2

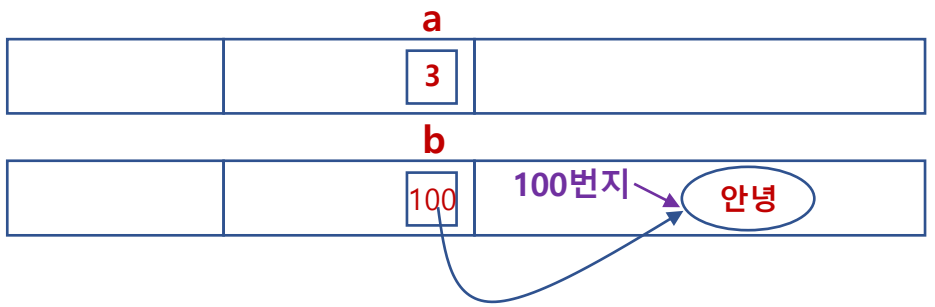
TIP

기본자료형과 참조자료형의 차이점 #2
- 기본자료형은 값을 stack 메모리에 저장
- 참조자료형은 값을 heap 메모리에 저장
  (heap 메모리는 직접접근 불가, stack에는 위치(번지)를 저장)

3

기본자료형 → int a = 3

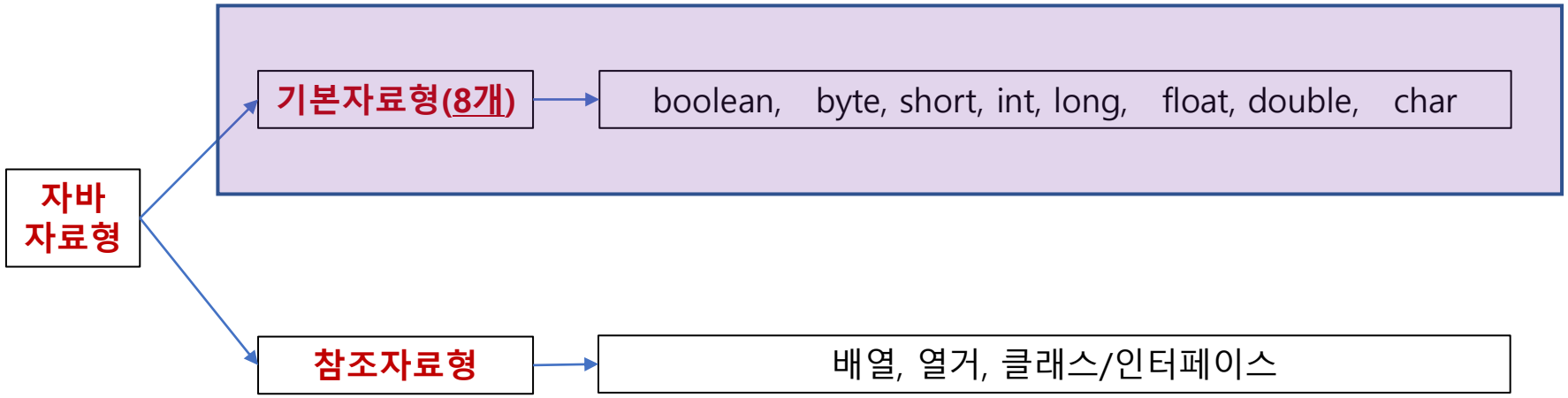
참조자료형 → String b = "안녕"



The End

자료형-기본자료형

자료형-기본자료형



자료형-기본자료형

☞ 8개의 기본자료형

1

자료형		자료크기	비고
부울대수	**boolean**	1 byte = 8 bit	true, false
정수	**byte**	1 byte = 8 bit	$-2^{7} \sim 2^{7}-1$
	short	2 byte = 16 bit	$-2^{15} \sim 2^{15}-1$
	int	4 byte = 32 bit	$-2^{31} \sim 2^{31}-1$
	long	8 byte = 64 bit	$-2^{63} \sim 2^{63}-1$
실수	**float**	4 byte = 32 bit	$\pm(1.40 \times 10^{-45} \sim 3.40 \times 10^{38})$
	double	8 byte = 64 bit	$\pm(4.94 \times 10^{-324} \sim 1.79 \times 10^{308})$
문자(정수)	**char**	2 byte = 16 bit	유니코드문자(0 ~ $2^{16}-1$)

2

TIP

n bit로 표현할 수 있는 정수의 개수 = 2^{n} 가지수

ex

- 2 bit → 2^{2}=4개
  00, 01, 10, 11
- 3 bit → 2^{3}=8개
  000, 001, 010, 011, 100, 101, 110, 111

Quiz.

어떻게 실수는 같은 크기로 넓은 범위를 저장할 수 있을까?

3

TIP

실수의 정밀도 (IEEE-754 표준)
- float은 소수점 대략 7자리 정도의 정밀도
- double은 소수점 대략 15자리 정도의 정밀도

4

ex

```
float f = 1.0000001f;
System.out.println(f); //→ 1.0000001
float f = 1.00000001f;
System.out.println(f); //→ 1.0

double d1 = 1.000000000000001;
System.out.println(d1); → 1.000000000000001
double d2 = 1.0000000000000001;
System.out.println(d2); → 1.0
```

자료형-기본자료형

☞ 8개의 기본자료형

- **부울대수 (boolean)**

1

참(true), 거짓(false) 값만 저장하는 자료형

ex

```
boolean a = true;
boolean b = false;
System.out.println(a); // true
System.out.println(b); // false
```

2

TIP

- boolean 자료형은 참(true)과 거짓(false)만 저장하여 실제로는 1bit로 가능
- 하지만 자료처리의 최소단위가 byte이기 때문에 1byte를 할당(상위 7bit는 사용하지 않음)

- **정수 (byte, short, int, long)**

3

음의 정수, 0, 양의 정수를 저장하는 자료형

ex

```
byte a = 10;
short b = -10;
int c = 100;
long d = -100L;
System.out.println(a); //10
System.out.println(b); //-10
System.out.println(c); //100
System.out.println(d); //-100
```

타입변환
(Type Casting)

4

- **실수(float, double)**

소수를 포함하는 실수를 저장하는 자료형

ex

```
float a = 1.2F;
double b = -1.5;
double c = 5;
System.out.println(a); // 1.2
System.out.println(b); // -1.5
System.out.println(c); // 5.0
```

출력시에는 자료형을 기준으로 출력됨

자료형-기본자료형

☞ 8개의 기본자료형

- **문자 (char)**

1

문자(정수)를 저장하는 자료형
문자를 저장하기 위해서는 작은따옴표 (' ') 사용
유니코드 값을 그대로 입력 가능 ('₩u+16진수코드')
정수값(10진수 또는 16진수 등 다양한 진법)의 직접 입력 가능

☞ 여기서 잠깐

2

자바에서는 다양한 진법의 정수값 표현을 지원함
(단, 각 진법마다 숫자의 표현방식이 다름)
10진수: 숫자
2진수: 0b + 숫자
8진수: 0 + 숫자
16진수: 0x + 숫자

ex int a=11; (10진수) → (10진수값) 11
int b=0b11; (2진수) → (10진수값) 3
int c=011; (8진수) → (10진수값) 9
int d=0x11; (16진수) → (10진수값) 17

3

ex

```
char a = 'A';
char b = '\u0042';
char c = '1';
char d = 97;
char e = 0xac00;
```

4

문자의 저장 방식

5

문자의 출력 방식

자료형-기본자료형

☞ 8개의 기본자료형

- **문자 (char)**

1

ex

```
char a = 'A';
char b = '\u0042';
char c = '1';
```

유니코드표		
10진수	16진수	문자
0	0x0000	NULL
…	…	…
48	0x0030	0
49	0x0031	1
50	0x0032	2
51	**0x0033**	**3**
…		
65	**0x0041**	**A**
66	0x0042	B
…	…	…
97	0x0061	a
98	0x0062	b
…	…	…
44032	**0xac00**	**가**
…	…	…

3

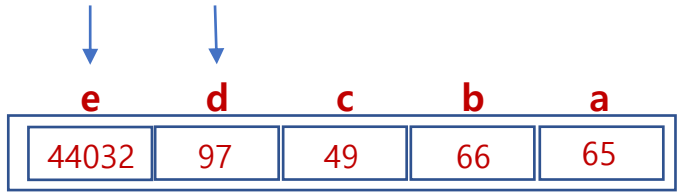
ex

```
System.out.println(a);// A
System.out.println(b);// B
System.out.println(c);// 1
System.out.println(d);// a
System.out.println(e);// 가
```

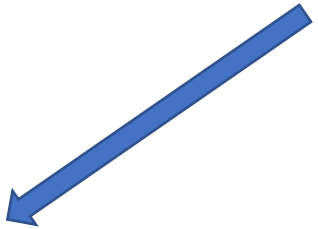
2

ex

```
char d = 97;
char e = 0xac00;
```



메모리



자료형-기본자료형

☞ 기본자료형 간의 타입변환

- **리터럴(literal) 타입**

2

자료형 없이 값으로 입력하는 경우 값의 형태에 따라 대표 자료형으로 자동 변환됨
정수값 → **int**로 저장
실수값 → **double**로 저장

4

예외 case
int 보다 작은 자료형인 byte/short에 대입되는 int 값은 각각 byte와 short 로 인식됨
(단, 해당 타입이 저장할 수 있는 범위 내 값인 경우)

TIP 1

JAVA는 등호(=)를 중심으로 좌우의 타입이 동일하여야 함

3

ex

```
int a = 3; // int 자료형 = int 자료형
double b = 5.8; // double 자료형 = double 자료형
float c = 3.2; //(X) float 자료형 = double 자료형
long d = 3; //(O) long 자료형 = long(int) 자료형

byte e = 5; //(O) byte 자료형 = byte 자료형
short f = 8; //(O) short 자료형 = short 자료형
```

해결책

- **타입변환(Type Casting)**

5

- 숫자를 저장하는 7개(boolean 제외)의 기본자료형 사이에 타입변환 가능
- **자동타입변환** 및 **수동타입변환**

boolean

6

숫자 저장 기본자료형간의 타입변환

byte	short	int	long
float	double		char

7

TIP

자료형의 크기(값의 범위) 순서
byte < short/char < int < long < float < double

자료형-기본자료형

☞ 기본자료형 간의 타입변환

TIP

4 JAVA는 등호(=)를 중심으로 좌우의 타입이 동일하여야 함

1 - **타입변환(Type Casting) 방법**

2 타입변환 대상 앞에 (자료형) 표기

ex

```
int a = (int)3.2;        //3
double b = (double)a;    //3.0
byte c = (byte)5.3;      //5
short d = (short)10;     //10
```

3 long 형의 경우 **숫자** 뒤에 L(l) 표기
float 형의 경우 **숫자** 뒤에 F(f) 표기

ex

```
long a = (long)10;       //10
long b = 10L;            //10
float c = (float)5.8;    //5.8
float d = 5.8F;          //5.8
```

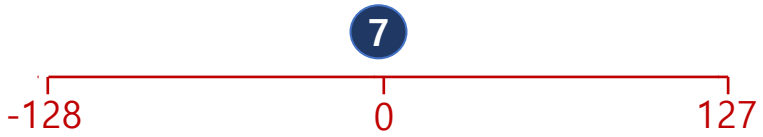
자료형-기본자료형

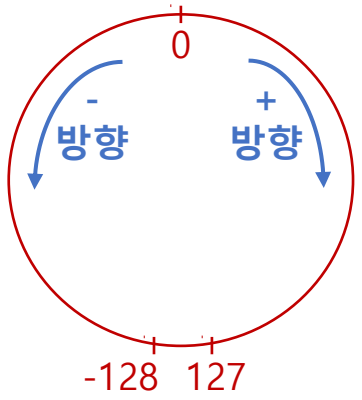
☞ 기본자료형 간의 타입변환

- **자동타입변환/수동타입변환**

TIP

자료형의 크기(값의 범위) 순서
byte < short/char < int < long < float < double





자동타입변환

1

- 값의 표현 범위가 **넓은** 쪽으로 저장되는 경우
- int 보다 작은 자료형(byte, short)에 정수를 입력하는 경우 (단, 값의 범위 값만 허용)
- 컴파일러가 자동으로 타입변환 수행 (즉, 타입변환 생략 가능)

3

ex

```
float a = 3;        //float 자료형 ← int 자료형
long b = 7;         //long 자료형 ← int 자료형
double c = 5.3F;    //double 자료형 ← float 자료형
```

4

ex

```
byte a = 3;         //byte 자료형 ← int 자료형
byte b = 128;(X)    //값 범위를 넘어 수동타입변환
short b = 7;        //short 자료형 ← int 자료형
```

5

TIP

CPU는 byte, short 데이터의 로딩 과정에서 int 값으로 읽어와 변환하여 저장 (하위비트) (연산시에도 최소자료형은 int)

수동타입변환

2

- 값의 표현 범위가 **좁은** 쪽으로 저장되는 경우
- 값의 손실이 발생할 수 있음
- 직접 표기하지 않으면 오류 발생

6

ex

```
int a = (int)3.5;           //3
float b = (float)7.5;       //7.5
byte c = (byte)128;         //-128
```

자료형-기본자료형

☞ 기본자료형 간의 타입변환

TIP

자료형의 크기(값의 범위) 순서
byte < short/char < int < long < float < double

- **기본 자료형간의 연산**

1

- 동일한 자료형끼리만 연산 가능
- 다른 자료형끼리 연산을 수행하는 경우 자동타입변환이 수행되어 연산

2 같은 타입끼리의 연산

byte 자료형 + byte 자료형 = int 자료형
short 자료형 + short 자료형 = int 자료형
int 자료형 + int 자료형 = int 자료형
long 자료형 + long 자료형 = long 자료형
float 자료형 + float 자료형 = float 자료형
double 자료형 + double 자료형 = double 자료형

3 다른 타입끼리의 연산

byte 자료형 + short 자료형 = int 자료형
byte 자료형 + int 자료형 = int 자료형
short 자료형 + long 자료형 = long 자료형
int 자료형 + float 자료형 = float 자료형
long 자료형 + float 자료형 = float 자료형
float 자료형 + double 자료형 = double 자료형

4 ex

```
int a = 3 + 5;              //8
int b = 8/5;                //1
float c = 3.0f + 5.0f;      //8.0
double d = 8.0/5.0;         //1.6
```

5 ex

```
double a = 5 + 3.5;         //8.5
int b = 5 + 3.5;            //오류
double c = 5/2.0;           //2.5
```

6 ex

```
byte a = 3;
short b = 5;
int c = a+b;                //8
double d = a+b;             //8.0
```

The End