

● 논리형(boolean): 논리형 변수에는 소문자 true와 false중 하나만 저장할 수 있으며 기본값(default)는 false다. 대답(yes/no), 스위치(on/off)등의 논리 구현에 주로 사용된다. 두 가지 값만을 표현하면 되므로 1 bit만으로 충분하지만, 자바에서는 데이터를 다루는 최소 단위가 byte이기 때문에, boolean의 크기가 가장 작은 1 byte이다.

● 문자형(char): 컴퓨터는 숫자만 알기에 문자를 문자의 유니코드(정수—2byte)로 변환하여 저장한다. 문자 리터럴 대신 문자의 유니코드를 직접 저장할 수도 있다. 어떤 문자의 유니코드를 알고 싶으면, char형 변수에 저장된 값을 정수형(int)으로 캐스팅 해 보면 된다. 문자형 타입의 크기는 2byte(16bit)이므로, 16자리의 2진수로 표현할 수 있는 정수의 개수인 (2^{16}) 65536개의 코드를 사용할 수 있으며, char형 변수는 이 범위 내의 코드 중 하나를 저장할 수 있다. 다만, 정수형과 달리 음수를 나타낼 필요가 없으므로 표현할 수 있는 값의 범위가 다르다. char타입에 저장되는 값인 유니코드는 모두 양수(0 포함)이므로, '0~65535'의 범위를 가지는 반면, 정수형인 'short'는 절반을 음수표현에 사용하므로 '32768~32767'을 범위로 갖는다.

→ 특수문자

특수 문자	문자 리터럴
tab	\t
backspace	\b (자바2.0)
form feed	\f
new line	\n
carriage return	\r
역슬래시(\)	\\
작은따옴표(')	\'
큰따옴표(“)	\"
유니코드(16진수)문자	\u유니코드 (예: char a='\u0041')

● 인코딩과 디코딩 (encoding & decoding): 문자를 코드로 변환하는 것을 문자 인코딩(encoding: 코드화하다, 암호화하다)이라고 하고, 코드를 문자로 변환하는 것을 문자 디코딩(decoding)이라고 한다. 문자를 저장할 때는 인코딩을 해서 숫자로 변환해서 저장하고, 저장된 문자를 읽어올 때는 디코딩을 해서 숫자를 원래의 문자로 되돌려야 한다. 만약 인코딩에 사용된 코드표와 디코딩에 사용된 코드표가 다르면 깨진 글자들로 바뀌어서 나타나기 때문에, 인코딩 설정이 옳아야 올바르게 디코딩된다.

- 아스키 (ASCII: American Standard Code for Information Interchange: 정보교환을 위한 미국 표준 코드): 128개($=2^7$)의 문자 집합(character set)을 제공하는 7 bit부호 - 기호와 숫자, 영대소문자로 이루어져 있다.

- 확장 아스키(Extended ASCII): 남은 1bit를 활용해서 문자를 추가로 정의한 것으로, 두 개의 문자코드로 한글을 표현하는 방법이 있다(조합형과 완성형). 현재는 확장 완성형이 사용된다 (CP 949: 확장 아스키의 일부 영역에 해당되는 두 문자코드를 조합하고, 완성형에 없는 잘 안 쓰이는 8822글자를 추가한 것) - 바로 한글 윈도우에서 사용하는 문자 인코딩.

- 유니코드 (Unicode): 전 세계의 모든 문자를 하나의 통일된 문자집합으로 표현. 유니코드에 포함시키고자 하는 문자들의 집합을 정의하였는데, 이것을 유니코드 문자 셋(character set)이라고 하고, 이 문자 셋에 번호를 붙인 것이 유니코드 인코딩이다. 자바에서는 인코딩 종류 중 하나인 UTF-16을 사용하는데, 이것은 모든 문자를 2 byte의 고정 크기로 표현한다. UTF-8은 하나의 문자를 1~4byte의 가변 크기로 표현. 두 인코딩 모두 처음 128문자가 아스키와 동일하다. 모든 문자의 크기가 동일한 UTF-16이 문자를 다루기에는 편하지만, 1 byte로 표현할 수 있는 영어와 숫자가 2 byte로 표현되므로 문서의 크기가 커진다는 단점이 있다. 반면 UTF-8은 영문과 숫자는 1 byte그리고 한글은 3 byte로 표현되기 때문에 문서의 크기가 작지만 문자의 크기가 가변적이므로 다루기 어렵다는 단점이 있다. 인터넷에서는 전송속도가 중요하므로, 문서의 크기가 작을 수록 유리하여 UTF-8인코딩으로 작성된 웹문서의 수가 많다.

● 정수형 (byte, short, int)

- 기본 자료형(default data type): int.
- 정수형의 표현 형식과 범위: 어떤 진법의 리터럴을 변수로 저장해도 실제로는 2진수로 바뀌어 저장된다. 정수형은 첫번째 비트로 S (부호 비트 - sign bit)를 사용하고, 나머지는 $n - 1$ bit 값을 사용하는데 사용한다. 그래서 n비트로 표현할 수 있는 값의 개수인 2^n 개에서, 절반인 0으로 시작하는 2^{n-1} 개의 값을 양수(0 포함)의 표현에 사용하고, 나머지 절반인 1로 시작하는 2^{n-1} 의 값은 음수의 표현에 사용된다. 그래서 정수형은 타입의 크기만 알면, 최대값과 최소값을 쉽게 계산할 수 있다.



n비트로 표현할 수 있는 정수의 개수: 2^n 개
 n비트로 표현할 수 있는 부호 있는 정수의 범위: $-2^{(n-1)} \sim 2^{(n-1)} - 1$

최대값에서 1을 빼는 이유는 범위에 0이 포함되기 때문



타입	저장 가능한 값의 범위	크기	
		bit	byte
byte	-128 ~ 127	8	1
short	- 32,768 ~ 32,767	16	2
int	- 2,147,483,648 ~ 2,147,483,647 (약 ± 20 억)	32	4
long	-9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807	64	8

- 선택기준: 변수에 저장하려는 정수값의 범위에 따라 선택 가능하지만, JVM의 피연산자 스택 (operand stack)이 피연산자를 4 byte로 저장하기 때문에 크기가 4 byte보다 작은 자료형(byte, short)의 값을 계산할 때는 4 byte로 변환하여 연산이 수행되므로 오히려 int를 사용하는 것이 효율적이다. 정수형 변수를 선언할 때는 int타입으로 하고, int의 범위인 약 ± 20 억을 넘어서는 수를 다뤄야 할 때는 long을 사용하면 된다. byte, short는 성능보다 저장공간을 절약하는 것이 더 중요할 때 사용하자. (long타입의 범위를 넘어서는 실수형 타입이나 BigInteger 클래스를 사용하기)
- 오버플로우(overflow): 타입이 표현할 수 있는 값의 범위를 넘어서는 것. 계수기에서처럼 정수형 타입이 표현할 수 있는 최대값에 1을 더하면 최소값이 되고, 최소값에서 1을 빼면 최대값이 된다.
- 부호 있는 정수의 오버플로우: 부호비트가 0에서 1이 될 때 오버플로우가 발생한다.

● 실수형 (float, double)

- 정밀도: 오차 없이 저장할 수 있는 10진수의 자리. 만약 7자리 이상의 정밀도가 필요하다면, 변수의 타입을 double로 해야 한다. 대부분 보다높은 정밀도가 필요해서 double 타입의 변수를 사용해서 실수형 값을 저장한다. 연산속도의 향상이나 메모리를 절약하려면 float을 선택하고, 더 큰 값의 범위라던가 더 높은 정밀도를 필요로 한다면 double을 선택해야 한다.

타입	정밀도	크기	
		bit	byte

float	7자리	32	4
double	15자리	64	8

- 같은 byte의 크기라도 실수가 정수보다 넓은 범위의 값을 저장할 수 있는 이유는 값을 저장하는 형식이 다르기 때문이다. 실수형은 값을 부동소수점수(floating-point)형태로, 즉, 실수를 $\pm M \times 2^E$ 와 같은 형태로 저장하기에 큰 범위의 값을 저장하는 것이 가능하다. 부동소수점수는 부호, 지수, 가수의 부분으로 나누어져 있다.

▶ 부호(S-sign: 1 bit, 0이면 양수, 1이면 음수)

▶ 지수(E-exponent: 8/11 byte): 부호 있는 정수, 범위는 float은 -127~128인데, 이 중에서 -127과 128은 숫자 아님 — NaN Not a Number이나 양의 무한대(POSITIVE_INFINITY), 음의 무한대(NEGATIVE_INFINITY)와 같이 특별한 값의 표현을 위해 예약되어 있어서 실제로 사용 가능한 범위는 -126~127이다. 그래서 지수의 최대값은 127이므로 표현할 수 있는 최대 값은 2^{127} (10진수로 10^{38}), double은 -1023~1024이다. 그러나 float의 최소값은 가수의 마지막 자리가 2^{-23} 이므로 지수의 최소값보다 2^{-23} 배나 더 작은 값, 약 10^{-45} 이다.

▶ 가수(M-mantissa: 23/52 byte): 실제 값을 저장하는 부분, 10진수로 7자리(float), 15자리(double)의 정밀도로 저장 가능하다.

→ 부동 소수점의 오차: 파이 같은 무한소수가 존재하고, 또한 10진수가 아닌 2진수로 저장하기 때문에 10진수로는 유한소수더라도 2진수로 변환할 때 무한소수가 되거나, 유한소수더라도 가수를 저장할 수 있는 자리수가 한정되어 있으므로 저장되지 못하고 버려지는 값들이 있으면 오차가 발생하는 경우도 있다.

→ 정규화: 2진수로 변환된 실수를 저장할 때 먼저 $1.xxx \times 2^x$ 의 형태로 변환하는 과정. 2진 실수는 항상 '1.'으로 시작하기 때문에, '1.'을 제외한 23자리의 2진수가 가수(mantissa)로 저장되고 그 이후는 잘려나간다. 지수는 기저법으로 저장되기 때문에 지수에 기저인 127을 더한 x가 2진수로 변환되어 저장된다 (기저법 = 특정값/기저를 더했다가 읽어올 때는 다시 빼는 형식). 이때 잘려나간 값들에 의해 발생할 수 있는 최대 오차는 약 2^{-23} 인데, 이 기댓값은 가수의 마지막 비트의 단위와 같다. 이는 10진수로 약 10^{-7} 이므로 float의 정밀도가 7자리라고 하는 것이다.

8

실수 데이터의 표현

▶ 10진수 -123.25를 4바이트 부동 소수점으로 표현

▶ 123.25를 2진수로 표현 : $1111011.01 = 1.11101101 \times 2^6$

▶ 부호비트 : 음수 → 1

▶ 지수부 : $127 + 6 = 133 = 1000\ 0101$

▶ 가수부 : 11101101

부호	지수부분	가수부분
31	23	22
1	1 1 0 0 0 0 1 0 1 1 1 1 0	1 1 0 1 0

<https://www.youtube.com/watch?v=SlkMOaKe7HQ>

- float 타입의 범위: $-3.4 \times 10^{38} \sim 3.4 \times 10^{38}$ 이지만 $-1.4 \times 10^{-45} \sim 1.4 \times 10^{-45}$ 의 범위(0은 제외)의 값은 표현할 수 없다. 실수형은 얼마나 큰 값을 표현할 수 있는가도 중요하고, 얼마나 0에 가깝게 표현할 수 있는지도 중요하다. printf메서드의 지시자 %f는 기본적으로 소수 점 이하 6자리까지만 출력하므로, 7번째 자리에서 반올림한다.

- 오버플로우: 실수형에서 오버플로우가 발생하면 변수의 값은 무한대(infinity)가 된다.

- 언더플로우(underflow): 실수형에서 양의 최소값보다 작은 값이 될 경우 변수의 값은 0이 된다.

- floatToIntBits(): float타입의 값을 int타입의 값으로 해석해서 변환한다 (16진수로 출력. 잘려나간 첫 번째 자리의 값이 1일 경우 반올림)

