Chapter 02. 변수

1. 변수란?

- 하나의 값을 저장할 수 있는 메모리 공간
- Data를 담아두는 Memory 공간

1-1. 변수의 초기화

- 변수에 처음으로 값을 저장하는 것.
- 이미 값이 초기화 되어 있는 변수에 다른 값을 넣는 것 또한 초기화한다고 함.

1-2. 변수의 종류

- 클래스변수**(Class variables)**
    - 클래스가 처음 호출될 때 시작하여 프로그램이 끝날 때 소멸
    - 자주 사용되고 변함 없는 자료일 경우 클래스 변수에 선언
- 인스턴스 변수**(Instance variables)**
    - 객체변수
    - 객체가 생성될 때 해당 객체를 참조하는 객체가 없을 때 소멸
- 지역변수**(Local variables)**
    - method 안에서만 유효한 변수
- 매개변수**(Parameters)**
    - method가 호출될 때 시작, method가 끝날 때 소멸

```
public class Variables{
  static int classVariables;   // 클래스 변수, 정적 변수

  int instanceVariables;       // 인스턴스 변수, 필드, 전역 변수

  public void method(int parameters){    // 매개변수, 파라미터
      int localVariables;                // 지역변수, local 변수
  }
}
```

1-2-1. 변수의 스코프

- 변수의 사용가능한 범위
- 변수가 선언된 블럭이 그 변수의 사용범위를 뜻한다.

관련문서 참고 : https://www.notion.so/scope-static-92aeb94fd1bc4bd1b00a42ee5fb0412c

1-3. 대입연산자의 의미

- '='의 의미
  - 오른쪽의 값을 왼쪽에 넣어준다는 의미
  - 수학적인 의미에서의 등호와는 쓰임이 다름

```
int x = 5;    // 5라는 값을 int형태의 변수 x에 저장한다(넣어준다)
```

1-4. 명명 규칙

💡 **< 필수 규칙 >**
1. 대소문자 구분, 길이에 제한 없음
2. 예약어 사용 불가
3. 특수문자는 오직 '$'와 '_'만 사용 가능
4. 숫자로 시작할 수 없음

💡 < 권장 규칙 > (관행)
1. 클래스 이름의 첫글자는 대문자로
   ⇒ 변수와 메서드 이름의 첫글자는 소문자
2. 여러 단어를 붙여서 사용할 경우 붙인 단어의 첫글자는 대문자
3. 상수의 이름은 모두 대문자로 사용. 단어의 구분은 '_'를 사용

2. 변수의 타입

- 변수의 타입은 크게 **기본형**과 **참조형**으로 나뉜다.

- **기본형은 변수의 실제 값을 저장**하는 반면에 **참조형은 어떤 값이 저장된 주소의 위치를 값으로 갖는다.**

2-1. 기본형(primitive type)

	1 byte	2 byte	4 byte	8 byte
논리형	boolean			
문자형		char		
정수형	byte	short	**int**	long
실수형			float	**double**

- boolean은 true와 false 두 가지 값만 표현할 수 있으면 되므로 가장 작은 크기인 1byte
- **char은 자바에서 유니코드(2byte 문자체계)를 사용하므로 2 byte.**
- byte는 크기가 1byte라서 byte.
- int(4byte)를 기준으로 짧아서 short(2byte), 길어서 long(8byte). (short ↔ long)
- **float는 실수값을 부동소수점(floating-point) 방식으로 저장하기 때문에 float**
- **double은 float보다 두 배의 크기(8byte)를 갖기 때문에 double.**

논리형 : boolean

- boolean은 true와 false 중 하나의 값을 가지며, 조건식 (0(no), 1(yes)), 스위치 (on/off)등의 논리구현에 사용된다
- 기본값은 false이다
- Java에서 true와 TRUE는 대소문자를 구문하므로 서로 다른 값이다
- boolean을 제외한 나머지 7개의 기본형은 서로 연산과 **변환(Casting)**이 가능하다

문자형 : char

- 문자를 저장하는데 사용되며, 변수 하나에 하나의 문자만 저장 가능
- 문자를 내부적으로 정수(유니코드)로 저장하므로 정수형 또는 실수형과 연산이 가능하다.
- 유니코드 체계를 사용하기 때문에 2byte를 차지한다. (a는 유니코드로 '\u0041')
  - 유니코드 문자 표현법은 '\u16진수'

정수형 : byte, short, int, long

- 정수를 저장하는데 사용되며, byte는 사진 데이터 같은 이진데이터를 다룰 때, short는 C언어와의 호환을 위해 사용된다.
- 정수형의 기본 자료형(defalut data type)은 int 이다.

- int는 CPU가 가장 효율적으로 처리할 수 있는 타입이기에 기본적으로 사용하지만, 효율적인 실행보다 메모리를 절약하려면 byte나 short를 사용하면 된다.
- long 타입의 변수를 선언한 후 값을 저장할 때는 리터럴에 접미사 'L'을 붙여야 한다.

> 💡 **16비트로 표현할 수 있는 정수의 개수 : 2^{16}개(65536개)**
>
> **short 타입의 표현범위 : -2^{15} ~ $2^{15}-1$ (-32768 ~ 32767)**
> **char 타입의 표현 범위 : 0 ~ $2^{16}-1$ (0 ~ 65535)**
>
> ⇒ char 타입은 유니코드 문자체계를 사용하기 때문에 0 이상의 수를 사용 (unsigned 자료형)

▼ 참고내용

> 💡 **Unsigned 자료형**
>
> **간단하게 말하면 unsigned 자료형은 양수만 저장하고**
> **signed 자료형은 음수를 포함하여 저장하는 자료형이다.**
>
> **char이 unsigned 자료형이고**
> **short가 signed 자료형이라고 생각하면 편함.**

C언어에서는 unsigned 자료형을 부호 없는 자료형이라고 표현하며 사실 자바에서는 이 기능이 빠져있다.

자세한 내용은 https://kirkim.github.io/java/2021/06/13/unsigned.html 참고

> 💡 **JVM의 피연산자 스택(operand stack)이 피연산자를 4byte 단위로 저장하기 때문에 크기가 4byte보다 작은 자료형 (byte, short)의 값을 계산할 때는 4 byte로 변환하여 연산이 수행된다.**

실수형 : float, double

- 실수를 저장하는데 사용된다.
- 실수형의 기본 자료형은 double이다.
- float 타입의 변수를 선언한 후 값을 저장할 때는 접미사 'f'를 붙여야 한다.
- float의 정밀도는 7, double의 정밀도는 15이다.

2-2. 저장 가능한 값의 범위

자료형	저장 가능한 값의 범위	크기	정밀도
boolean	false, true	1byte	
char	\u0000 ~ \uffff (0 ~ 66535)	2byte	
byte	-128 ~ 127 (-2^{8-1} ~ $2^{8-1}-1$)	1byte	
short	-32768 ~ 32767 (-2^{16-1} ~ $2^{16-1}-1$)	2byte	
int	-2147483648 ~ 2147483647 (-2^{32-1} ~ $2^{32-1}-1$, 약 +-20억)	4byte	
long	–2,147,483,648 ~ 2,147,483,647 (-2^{64-1} ~ $2^{64-1}-1$)	8byte	
float	1.175494351 E - 38 ~ 3.402823466 E + 38	4byte	7자리
double	1.7976931348623158 E + 308	8byte	15자리

💡 **n비트로 표현할 수 있는 정수의 개수** = 2^n개 (2^{n-1}개 + 2^{n-1}개)
n비트로 표현할 수 있는 부호있는 정수의 범위 = -2^{n-1} ~ $2^{n-1}-1$

- 참고
  - long 타입의 범위를 벗어나는 값을 다룰 때는, 실수형 타입이나 BigInteger 클래스를 사용.

오버플로우

💡 **컴퓨터에서 정수의 연산결과가 해당 타입의 허용범위를 초과할 때 발생하는 오류**

https://www.notion.so/045b7b192cc6474a9484afe4f945cf9c

2-3. 상수와 리터럴

상수(constant)란?

- 변수와 마찬가지로 '값을 저장할 수 있는 공간'이지만, 변수와 달리 **한번 값을 저장하면 다른 값으로 변경이 불가.**
- 상수의 선언은 변수와 동일하며, **변수의 타입 앞에 final**을 붙이지만 하면 된다
- **상수는 반드시 선언과 동시에 초기화** 되어야 하며, 그 이후부터는 상수의 값을 변경할 수 없다.
- 상수의 이름은 모두 대문자로 처리하는 것이 관례이며, 여러 단어로 이루어져 있는 경우 '_'로 구분한다.
- 보통 PI = 3.14 같은 변수를 지정할 때 사용한다. 이미 값이 사회적으로 약속된 변수를 집어넣을 때 사용하면 유용. (의미있는 이름)

리터럴(literal)

- 사실 **리터럴의 의미는 상수와 동일**하다. 하지만 프로그래밍에서 상수를 이미 '값을 한 번 저장하면 변경할 수 없는 저장공간'으로 정의를 해 두었기에 이를 구분하기 위해 상수를 다른 이름으로 불러야했다. 따라서 **상수 대신 리터럴이라는 용어를 사용한다.**

변수(variable) 하나의 값을 저장하기 위한 공간
상수(constant) 값을 한번만 저장할 수 있는 공간
리터럴(literal) 그 자체로 값을 의미하는 것.

리터럴의 타입과 접미사

종류	리터럴	접미사
논리형	false, true	없음
정수형	123, 0b0101, 077, 0xFF, 100L	**L**
실수형	3.14, 3.0e8, 1.4f, 0x1.0p-1	**f**, d
문자형	'A', '1', '\n'	없음
문자열	"ABC", "123", "A", "true"	없음

문자 리터럴와 문자열 리터럴

- 문자 리터럴은 'A'와 같이 작은 따옴표로 하나의 문자를 감싼 것을 말한다 (char)
- 문자열 리터럴은 "ABC"와 같이 쌍따옴표로 여러개의 문자를 감싼 것을 말한다 (String)

💡 **연산시 덧셈 연산자(+)는 피연산자가 모두 숫자일 때는 두 수를 더하지만, 피연산자 중 어느 한 쪽이 String 이면 나머지 한쪽을 먼저 String으로 변환한 다음 두 String을 결합한다.**

어렵게 말했지만 결국 연산시 좌우에 String이 존재할 경우 연산이 아닌 문자끼리 더해준다고 보면 된다.

2-4. 타입의 기본값

타입	기본값
boolean	false
byte	0
short	0
char	'\u0000'
int	0
long	0L
float	0.0f
double	0.0(d)
참조타입 (ex. String)	null

2-5. 이스케이프 문장

특수문자	리터럴
Tab	\t
Backspace	\b
Form feed	\f
new line	\n
carriage return	\r
역슬래시(\)	\\
홑따옴표	\'
겹따옴표	\"
유니코드 문자(16진수)	\u16진수(ex. char a = \u0041)

2-6. 형식화된 출력 - printf()

지시자를 이용한 포맷 형식

지시자	설명
%b	boolean 형식으로 출력
%d	10진(demical) 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)으로 출력

3. 형변환

💡 **형변환이란, 변수 또는 상수의 타입을 다른 타입으로 변환하는 것**

모든 변수와 리터럴에는 타입이 존재하는데, 이러한 타입들이 모두 일치하지는 않는다. 이렇게 서로 다른 타입의 변수나 리터럴들을 연산하기 위해서는 연산을 수행하기 전에 타입을 일치시켜 주어야 하는데 그러한 과정을 형변환이라고 한다

3-1. 형변환의 원칙

- 기본형(primitive type)에서 boolean을 제외한 나머지 타입들은 모두 형변환이 가능하다.
- 기본형과 참조형 간의 형변환은 불가능하다.
- 서로 다른 타입의 변수간의 연산은 형변환을 하는 것이 원칙이지만, 값의 범위가 작은 타입에서 큰 타입으로의 형변환은 생략이 가능하다.

3-2. 정수형간의 형변환

- 큰타입에서 작은타입으로 변환하는 경우, 작은 타입의 크기에 맞춰서 큰 타입의 나머지 부분이 잘려나간다. ⇒ 값 손실 발생 가능
- 작은 타입에서 큰 타입으로 변환하는 경우에는 문제 없음. 남는 메모리 칸은 0 또는 1로 채워진다.
  - 양수의 경우에는 0으로 채운다(일반적)
  - 음수의 경우에는 1로 채운다 ⇒ **2의 보수법**

💡 **Integer.toBinaryString(int i)**
= 10진 정수를 2진 정수로 변환한 문자열로 바꿔주는 메서드

3-3. 실수형간의 형변환

- 실수형도 정수형과 마찬가지로 작은 타입에서 큰 타입으로 변환하는 경우, 빈 공간을 0으로 채운다.
- **float ⇒ double**
  - 지수(E)는 float의 기저인 127을 뺴서 0을 만들고, 다시 double의 기저인 1023을 더해서 변환
  - **가수(M)은 float의 가수 23자리를 채우고 남은 자리를 0으로 채운다.**
- double ⇒ float
  - 지수(E)는 double의 기저인 1023을 뺴고, 다시 float의 기저인 127을 더해서 변환한다.
  - 가수(M)은 double의 가수 52자리 중 23자리만 저장되고 **나머지는 버려진다.**
- **다만, 형변환시 가수의 24번째 자리에서 반올림이 발생할 가능성이 있음.**
  - 24번째 자리의 값이 1이면, 반올림이 발생하여 23번째 자리의 값이 1 증가한다.
- **float 타입의 범위를 넘는 값을 float로 변환시키는 경우, '±무한대' 또는 '±0'을 결과로 얻는다.**

3-4. 정수형과 실수형 간의 형변환

정수형을 실수형으로 변환

- 실수형은 정수형보다 큰 저장범위를 갖기 때문에, 정수형을 실수형으로 변환하는데는 크게 문제가 없다.

> 💡 **실수형의 정밀도 제한으로 인한 오차 발생확률 존재 (int → float)**
>
> **int의 정밀도는 10자리, float의 정밀도는 7자리**
>
> **int ⇒ double로 형변환을 시도할 것**

실수형을 정수형으로 변환

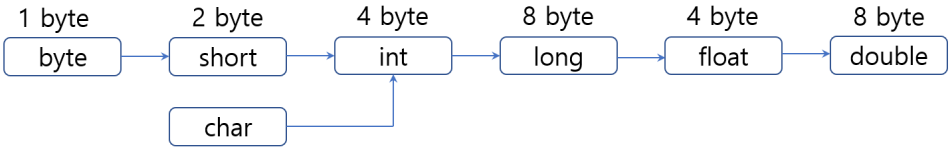
> 💡 **실수형을 정수형으로 변환하면, 실수형의 소수점 이하 값은 버려진다.**

- 만일 실수의 소수점을 버리고 남은 정수가 정수형의 저장범위를 넘는 경우에는 정수의 오버플로우가 발생한 결과를 얻는다.

3-5. 자동 형변환의 규칙

💡 **기존의 값을 최대한 보존할 수 있는 타입으로 자동 형변환한다**

표현범위가 좁은 타입에서 넓은 타입으로 형변환하는 경우에는 값 손실이 없으므로 두 타입 중에서 표현 범위가 더 넓은 쪽으로 형변환 한다.



자동 형변환이 가능한 방향

char과 short의 형변환 문제

💡 **char와 short**은 둘 다 2byte의 크기로 크기가 같지만, char의 범위는 unsigned한 범위로 0이상의 값을 지니고, short는 signed한 범위로 음수의 범위를 지닌다.
서로 범위가 달라서 어느 쪽으로의 형변환도 값 손실이 발생할 수 있으므로 자동 형변환이 수행될 수 없다.

char ⇒ $0 \sim 2^{16} - 1$ (0 ~ 65535)
short ⇒ $-2^{15} \sim 2^{15} - 1$ (-32768 ~ 32767)