22. 09 .14 - 변수

💡 **변수 (자료를 저장하기 위함)**

1. **메모리 공간 확보 = 변수 선언 (운영체제에 메모리 공간을 주세요)**
   - 사용하기 전에 반드시 미리 "**선언**" 해야 한다.
   - 변수 선언이란 컴파일러에게 어떤 변수를 사용하겠다고 미리 알리는 것
2. **타입 + 변수명 (인트 4바이트)**
   - 상자에도 라벨 붙이듯이 변수에 이름을 붙여야 다른 변수와 구분 가능
   - 변수가 선언되면 값은 아직 정의되지 않은 상태가 된다. 변수를 선언과 동시에 값을 넣으려면 변수 뒤에 = 를 붙이고 초기값을 넣는다
3. **구분자가 세미콜론(;) 이라서 줄 맞춤 필요 x**
4. **변수를 선언 함과 동시에 초기 값을 넣는다 = (assignment , 대입연산자)**
5. **입력한 값을 저장할 때도 필요하고 계산 도중에 중간 결과를 저장 할 때도 필요**

Data type Variable name ;

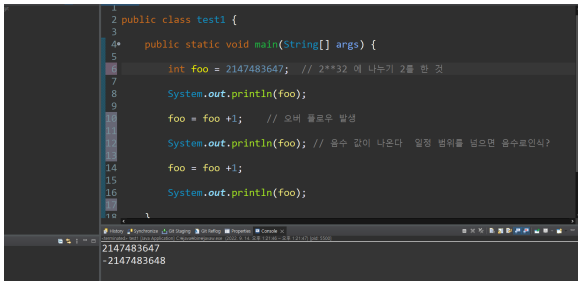
int foo(변수이름) ;

사용하기 전에 반드시 미리 선언하여야 한다.

선언을 하게 되면 컴파일러는 변수의 자료형에 맞는 기억 공간을 미리 확보한다.

💡 **오버플로우(Overflow) (타입이 표현할 수 있는 값의 범위를 넘어서는 것)**

담는 그릇 보다 큰 수가 들어오면 = 오버플로우 발생



바이트는 **파일 처리, 네트워크 통신**을 할 때 많이 사용 (그림을 비트를 바이트 형식으로 짤라서)

보통 int 를 사용한다

int 사이즈 4바이트

short 탄생 이유 = 기존에 c, c++ 짜놓은 코드들 까지 잡아먹을 라고 자바에서 short 를 만들었다

char (캐릭터) 2 바이트 → 문자는 아스키 코드로도 저장 가능 해서

자료형 (변수에 저장되는 데이터의 타입)

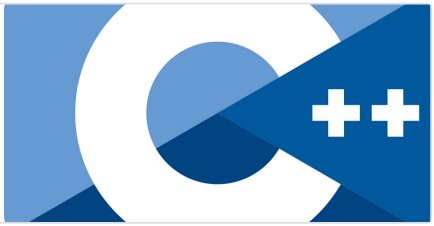
강의노트 12. 함수 호출방식(call-by-value, call-by-reference, call-by-assignment)

https://wayhome25.github.io/cs/2017/04/11/cs-13/

[C, C++] Call by value, Call by reference 쉽게 이해하기

(본 포스팅은 포인터와 관련이 깊습니다.) 함수의 호출 방법은 대표적으로 Call by value(값에 의한 호출) 와 Call by reference(참조에 의한 호출)가 있다. 함수 호출이란 말 그대로 정의된 함수를 호출하는 것으로 함수에 정의한 매개변수의 형태에 따라

https://kangworld.tistory.com/64



함수의 호출 방식

기본형

- 실제 값을 저장한다

참조형 (콜바이레퍼런스)

- 변수에는 변수의 값이 저장 되어 있다
- 정수형 , 실수형 , 문자형 , 논리형

- 참조형의 변수에서는 객체의 위치(참조 또는 주소) 가 들어있다. (값이 아니라 화살표가 저장되어 있다고 생각)
  어떤 값이 저장되어있는 주소(memory address)를 값으로 갖는다.
- 클래스, 배열, 인터페이스
- 인자 값의 자료형에 의존적이다
- 메모리 주소 값이 전달 된다
- 리스트는 리스트의 값이 넘어 가는 게 아니라 **리스트의 주소 값 자체**가 넘어간다
- 해당 객체의 주소값을 직접 넘기는 게 아닌 객체를 보는 또 다른 주소값을 만들어서 넘긴다

분류	변수의 타입	설명
숫자	**int** long	정수(integer)를 저장하기 위한 타입 (20억이 넘으면 long)
	Float **Double**	실수 (floating ~ point number)를 저장하기 위한 타입 (float는 오차 없이 7자리 , **double**은 15자리)
문자	char ' '	문자(character)를 저장하기 위한 타입 , 문자1개 ex) 'a'
	String " "	여러 문자(문자열,string)를 저장하기 위한 타입 ex) "betty"

String 은 객체이다. char배열과 String 클래스의 한 가지 중요한 차이가 있는데, String객체(문자열)는 읽을 수만 있을 뿐 내용을 변경할 수 없다는 것이다.

구분	데이터형	바이트 수	데이터 범위
정수	byte	1Byte	$-2^{7} \sim 2^{7}-1$ (-128 ~ 127)
	short	2Byte	$-2^{15} \sim 2^{15}-1$ (-32768 ~ 32767)
	int	4Byte	$-2^{31} \sim 2^{31}-1$
	long	8Byte	$-2^{63} \sim 2^{63}-1$
실수	float	4Byte	$-3.4 * 10^{38} \sim 3.4 * 10^{38}$
	double	8Byte	$-1.7 * 10^{308} \sim 1.7 * 10^{308}$
문자	char	2Byte	$0 \sim 2^{16}-1$ (Unicode 문자)
논리	boolean	-	true 또는 false

short 와 byte 는 메모리가 부족한 상황에서만 사용

기본형(Primitive type)

변수 종류에 따른 메모리 크기

종류 \ 크기 (byte)	1	2	4	8
논리형				
문자형		char		
정수형	byte	short	int	long
실수형			float	double

Boolean : 1bit or 1 byte JVM에 의존적

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

        short temp_1 = 2103 ; // 정수형 ,2byte

        char temp_2 = 'a';// 문자형 , 2byte

        byte temp_3 = 100 ; // byte , 1byte

        long temp_4  ; // 정수형, 8byte

        double temp_5 ; // 실수형, 8byte

        int temp_6 ; // 정수형, 4byte

        float temp_7 ; // 실수형, 4byte

        boolean temp_8 ; // 논리형, JVM 구현에 의존 1bit or 1b
    }
2147483647
-2147483648
```

Call-by-value vs Call-by-reference

```
def foo(a):
    a = a + 2

value = 1

# Call by value
# Primitive variables -> int, float, string, boolean
foo(value)
```

Call-by-value vs Call-by-reference

```
def bar(a):
    a.append(1)

value = [2, 3]

# Call by reference
# Reference variables -> Object -> List, Tuple, Dicitionary
bar(value)

print(value) # 2, 3, 1
```

상수

- 상수를 선언 하는 방법은 변수와 동일하며, 변수 타입 앞에 키워드 “**final**” 을 붙여주면 된다. *기호 상수 final 키워드는 변수에 값이 대입 되고 나면 변수의 값이 더 이상 변경되지 않는 다는 것을 의미한다
- 변수와 달리 **한번 값을 저장하면** 변수와 달리 **다른 값으로 변경 불가**!!.
- 상수는 **반드시 초기 값**이 있어야 한다
- 상수 이름은 모두 **대문자 ,** 여러 단어로 이루어져 있는 경우 ‘_’로 구분 한다

final

자바의 final은 자료형에 값을 단 한번만 설정할수 있게 강제하는 키워드이다. 즉, 값을 한번 설정하면 그 값을 다시 설정할 수 없다는 말이다.

```
public class Sample {
    public static void main(String[] args) {
        final int n = 123;  // final 로 설정하면 값을 바꿀수 없다.
        n = 456;  // 컴파일 에러 발생
    }
}
```

> `리터럴 (literal)`

프로그래밍에서는 상수를 “값을 한 번 저장하면 변경할 수 없는 저장 공간” 으로 정의 하였기 때문에 이와 구분하기 위해 상수를 다른 이름으로 불러야만 했다.

그래서 상수 대신 **리터럴 (literal)** 이라는 용어를 사용한다.

변수에 타입이 있는 것처럼 리터럴에도 타입이 있다.

변수의 타입은 저장될 “ 값의 타입(리터럴의 타입)”에 의해 결정 된다. ⇒ 리터럴에 타입이 없다면 변수의 타입도 필요 없을 것

`변수 (variable)` - 하나의 값을 저장하기 위한 공간

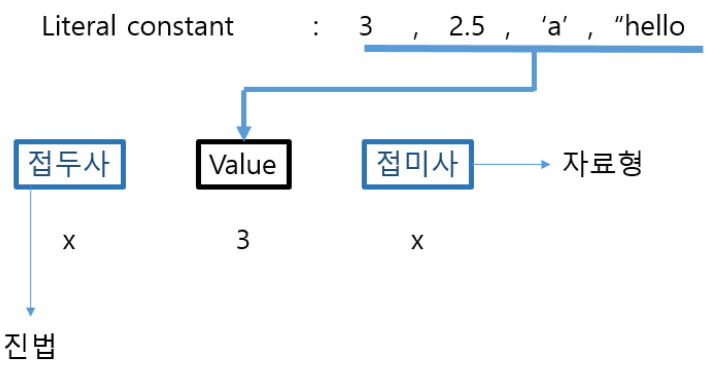
`상수 (constant)` - 값을 한번만 저장할 수 있는 공간

`리터럴 (literal)` - 그 자체로 값을 의미하는 것

> **정수형과 실수형에는 여러 타입이 존재 → 리터럴에 접미사를 붙여서 타입을 구분한다. 컴파일러는 “ _ “ 밑줄 기호를 무시한다.**

1. 접두사는 숫자에만 붙

는다



" **정수형** " 에서 접미사는 두 가지
1) 접미사 생략 시 -> **int 타입 (형) ⇒ 생략 가능**
2) 접미사 "L" 사용 시 -> long 타입 (형)

- 접두사는 진법을 나타낸다
- 아무것도 없으면 10진법

2. 접미사는 자료형을 나타낸다

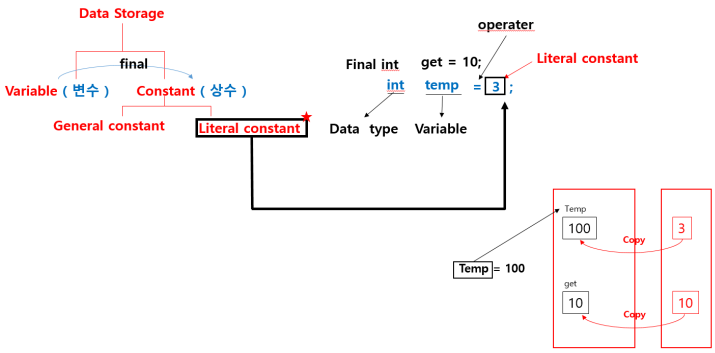
" **실수형** " 에서도 접미사는 두 가지
1) 접미사 생략 시 -> **double 타입 (형) ⇒ d 생략 가능**
2) 접미사 "f" 사용 시 -> float 타입 (형) ⇒ **f 생략 불가**

ex) float pi = 3.14f ; ⇒ float 는 반드시 리터럴 옆에 'f' 붙이기!

flaot betty = 12.3; ⇒ 12.3 은 double 형이므로 오류!!

float betty 12.3F ⇒ 가능

- double 은 float 보다 2배의 정밀도(precision)를 가진다
- float 는 메모리 용량이 제한된 장치에서만 사용
- **접미사 f 와 L 두 개는 꼭 기억!!**
- **작은 범위에서 큰 범위로 가는 건 에러 안 뜸 , 그 반대는 에러 뜬다!**




```
int     foo = 1L ;
long    bar = 1L;
float   pos = 1.2;
float   tel = 1.2f;
double  kal = 1.2;
```

‘문자 리터럴’ 과 “문자열 리터럴”

- ‘A’ 와 같이 작은 따옴표로 문자 하나를 감싼 것을 **‘*문자* 리터럴’** 이라고한다
- 두 문자 이상은 큰 따옴표로 감싸야 하며 **“*문자열* 리터럴”**이라고 한다 ⇒ 문자열은 “문자의 연속된 나열” 이라는 뜻이며 , String 이라고 한다.

원래 **String 은 클래스**이므로 아래와 같이 **객체를 생성**하는 **연산자 new**를 사용해야 하지만 특별이 이와 같은 표현도 허용한다.

→ String name = new String (”Java”); // String 객체를 생성 ← 원래는 이렇게 쓰는 게 맞다
→ String name = ”Java” ; // 위의 문장을 간단히

논리형

- 참과 거짓을 나타내는 데 사용된다.
- “`true`” 또는 “`false`” 만을 가질 수 있다. ⇒ 자바에서는 맨 앞 글자를 대문자로 사용하지 않고, 그대로 true , false 로 사용
- 자바에서는 정수 값을 논리형으로 사용 할 수 없다 ex) 파이썬에서 0은 False / 0이 아닌 값은 True

💡 ALU

```
import java.util.Scanner;

public class test1 {

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

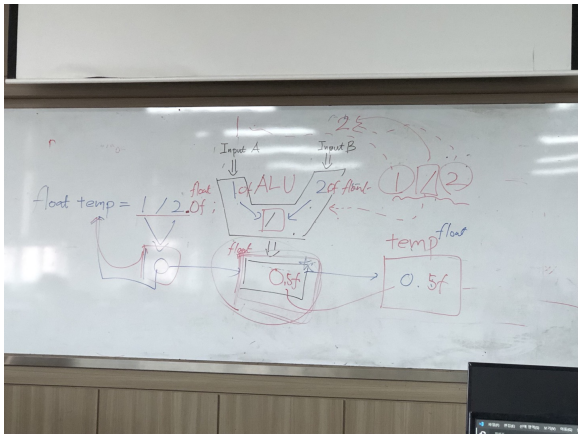
```
        Scanner scn = new Scanner(System.in);

        int firstValue = scn.nextInt();

        float result = firstValue / 3.0;

        System.out.println(firstValue);

    }
}
```



계산하여 결과 값을 저장하는 공간도 자료형을 가진다

수업 시간 연습 문제) 삼각형 면적 값 출력

```
import java.util.Scanner;

public class test1 {

  public static void main(String[] arg
s) {

    // 삼각형 밑변과 높이를 입력 받아 삼격형 면
적 값을 출력하는 프로그램을 작성
    Scanner scn = new Scanner(System.i
n);

    // 입력 값으로 넣기
        int firstValue = scn.nextInt();
        int secondValue = scn.nextInt
();

        // 결과값 = (밑면 x 높이) / 2
        float result = (firstValue*seco
ndValue)/2F ;

        // 출력
        System.out.println(result);
  }
```

```
import java.util.Scanner;

public class test1 {

    public static void main(String[] args) {

        // 삼각형 밑변과 높이를 입력 받아 삼격형 면적 값을 출력하는 프로그램을 작성
        Scanner scn = new Scanner(System.in);

        // 입력 값으로 넣기
        int firstValue = scn.nextInt();
        int secondValue = scn.nextInt();

        // 결과값 = (밑면 x 높이) / 2
        float result = (firstValue*secondValue)/2F ;

        // 출력
        System.out.println(result);
```

양에 정수 중 제일 큰 수에서 값을 더 더하면 `오버플로우`라고 한다

제일 낮은 수에서 더 낮게 하면 `언더플로우`가 발생한다

메모리를 딱 맞춰서 쓰면 `오버플로우` `언더플로우`가 발생할 수 있다.