**객체지향 프로그래밍** | 최윤정 교수님 | Summary 이서현 2276219  
[Chapter 02] week 2~3: 자바 기본 프로그래밍 (1) 기본 자료형과 연산자 (~35p)

+ [주석] 한 라인 주석: // 여러 행 주석: /\*\*/ 여러 줄 동시 선택해서 한 라인씩 주석처리: ctrl + /

**[클래스]** class 키워드로 클래스 선언. 클래스 코드는 {}안에 모두 작성. (ex. public class Hello {})

**[메소드]** C/C++에서의 함수. 호출해서 사용.  
- main() 메소드: 자바 프로그램은 main() 메소드에서부터 실행 시작. **public** **static** **void** main(String[] args) {}

**<출력>** System.***out***.println("");  
- 표준 출력 스트림 System.out의 println() 메소드를 호출. println()은 출력 후 다음 행으로 커서 이동. 여러 타입의 데이터 출력 가능.   
- print()는 엔터 없이 출력. printf()는 C와 사용법 동일, 문자열+변수값 동시 출력 위해 사용 (%d 등)

[변수] 프로그램 실행 중에 값을 임시 저장하기 위한 공간. 데이터 타입에서 정한 크기의 메모리를 할당한다.  
변수의 타입 다음에 변수 이름을 적어 변수를 선언. 메소드 내에서 선언된 변수는 지역 변수, 메소드 실행이 끝나면 자동 소멸.  
\*변수 초기화: 선언과 동시에 초기값 지정

[상수] final 키워드 사용해 상수 선언. 선언 시 초기값 지정, 실행 중 값 변경 불가! (ex. **final** **double** PI = 3.14;)

+ [식별자(identifier)] 클래스, 변수, 상수, 메소드 등에 붙이는 이름. 대소문자를 구별한다.  
<식별자 이름 규칙> 특수문자는 \_, $만 사용가능. 유니코드 문자나 한글 사용 가능(char이 2byte).  
식별자 첫 번째 문자로 숫자 사용불가. 자바 언어의 키워드(예약어)는 식별자로 사용불가 (true, false, null도 사용불가)

<자바 언어에서 식별자 이름 붙이는 관습>  
- 클래스: (첫 단어 포함) 각 단어 첫 번째 문자 대문자 (파스칼 표기법)  
- 변수, 메소드: 첫 단어 제외, 각 단어 첫 번째 문자 대문자 (카멜 표기법)  
- 상수: 모든 문자 대문자

[자바의 데이터 타입]

<논리 타입> boolean 1비트 (true or false)

<문자 타입> char 2바이트 Unicode (c언어에서는 1바이트였다)

<정수 타입> byte 1바이트 -128~127  
short 2바이트 -32768~32767  
int 4바이트 -2^31~2^31-1  
long 8바이트 -2^63~2^63-1 (c언어에서는 long은 4바이트, long long이 8바이트였다)

<실수 타입> float 4바이트 -3.4E38~3.4E38  
double 8바이트 -1.7E308~1.7E308

+ 문자열(String) " ". 많이 사용하는 변수 타입, 기본 타입이 아니어서 String 클래스로 문자열을 표현한다.

[리터럴(literal)] 프로그램에서 직접 표현한 값 (정수, 실수, 문자, 논리, 문자열 리터럴)

* 정수 리터럴: int 타입으로 컴파일. 10, 8, 16, 2진수 리터럴 존재.  
  (15: 10진수, 015: 8진수, 0x15: 16진수, 0b0101: 2진수) long 타입 리터럴은 숫자 뒤에 L또는 l을 붙여 표시한다.
* 실수 리터럴: double 타입으로 컴파일.   
  (12. 12.0 .1234 1234E-4) 숫자 뒤에 f나 d를 명시적으로 붙이기도 한다. (ex.0.1234f, .1234D. 서로 동일)
* 문자 리터럴: ' ' 홑따옴표로 문자 표현:  
  - 특수문자 리터럴은 백슬래시(\)로 시작. 탭: ' \t ', 단일 인용부호: ' \' ', 이중 인용부호: ' \" ', 백슬래시: ' \\ '  
  \u다음에 4자리 16진수가 오면 2바이트의 유니코드를 의미한다.
* 논리 리터럴: true, false 2개뿐. boolean 타입 변수에 넣거나 조건문에 이용.  
  - 자바에서는 1, 0을 참, 거짓으로 사용할 수 없다.  
  boolean a=true; //변수 초기화 boolean b = 10>0; //b는 true  
  boolean c = 1; //타입 불일치 오류. while(true){ } //무한 루프. while(1)로 사용하면 안 된다.
* null 리터럴: 레퍼런스에 대입해서 사용한다. (String str = null;)
* 문자열 리터럴: " "로 묶어 표현. 문자열이 섞인 '+' 연산은 문자열을 연결한다. (ex. "("+3+","+5+")" 🡪 "(3,5)"  
  문자열 리터럴은 String 객체로 자동 처리한다.

+ 팁: JDK7부터 숫자 사이 언더바 허용. 가독성 높임 (20\_000처럼. 자릿수 나누는 용도. 문자 앞뒤(소수점 앞뒤, L나F, 0x중간이나 끝)나 리터럴 끝에는 사용불가)

+ 팁: [var] java10부터 도입된 키워드. c++의 auto와 같은 기능. 지역 변수 선언에만 사용. 변수 타입 선언을 생략하고 초기값 지정해주면 컴파일러가 변수 타입을 추론하는 것. 변수 선언문에는 반드시 초기값을 지정해주어야 한다.

<Casting 캐스팅: 타입 변환>

[자동 타입 변환] 치환문이나 수식 내에서 타입이 일치하지 않을 때. (ex. long m = 25;)

[강제 타입 변환] 자동 타입 변환이 불가능한 경우 명시적으로 사용한다: 큰 타입이 작은 타입으로 변환될 때  
ex. double d = 1.9; int n = (int)d; //n=1;  
ex. System.***out***.println((**char**)0x12340041); //16진수인데 강제 타입 변환 결과 0x41, 10진수 65. A 출력.

<입력> Scanner input = **new** Scanner(System.***in***); //Scanner: 클래스명, input: 객체 이름, new: 메모리 할당해 생성

System.in: 키보드로부터 직접 받는 (바이트) 표준 입력 스트림. 키 값을 바이트로 리턴.

Scanner 클래스: System.in가 읽은 바이트 데이터를 문자, 정수, 실수, 불린, 문자열 등 다양한 타입으로 변환해서 리턴한다.

java.util.Scanner 클래스를 임포트해서 사용. 객체이름은 input이 많이 쓰인다.  
\* 스캐너 닫기 : \_\_.close();

Scanner는 키 입력을 공백으로 구분된 ‘토큰’ 단위로 끊어 읽는다.

(공백 문자: '\t', '\f', '\r', ' ', '\n')

<Scanner 주요 메소드> (\_\_: 변수이름)

* String next() : \_\_.next() 다음 토큰을 문자열로 리턴
* String nextLine() : \_\_.nextLine(), 공백 문자 기준으로 여러 번 읽을 때 사용, ‘\n’을 포함하는 한 라인을 읽고 ‘\n’을 버린 나머지 문자열을 리턴한다.
* int nextInt() : \_\_.nextInt(), 다음 토큰을 int 타입으로 리턴
* double nextDouble() : \_\_.nextDouble(), 다음 토큰을 double 타입으로 리턴
* boolean nextBoolean() : \_\_.nextBoolean(), 다음 토큰을 boolean 타입으로 리턴

nextLine은 엔터가 오기 전까지 받으나 next는 공백으로도 끊어진다. (ex. "김 철수" 입력 🡪 nextLine은 "김 철수", next는 "김")

문자열🡪정수🡪문자열 순서대로 입력할 경우, input.nextInt() 이후 엔터가 버퍼에 남아있게 되어 그 뒤의 input.next()나 input.nextLine()이 나왔을 때 버퍼에 남아있는 엔터를 받게 된다…!! 🡪의도적으로 엔터 먹어가는 함수 필요! String temp = input.nextLine(); 사용한다.

**if**(input.haxNextLine()) temp = input.nextLine(); temp 함수를 만들어 두고 여러 번 사용해도 된다.

저장 안하고 그냥 input.nextLine()해도된다