Java

Java SE - Standard Edition
Java EE - Enterprise Edition
Java ME - Micro Edition (사용안함)

설치

1. 컴파일러 다운로드

java.oracle.com

jdk-17_windows-x64_bin.exe

2. 설치

jdk-17_windows-x64_bin.exe 더블클릭

3. 환경변수

내 PC → 우클릭 → 속성 → 고급 시스템 설정 → 고급(탭) → 환경변수

① JDK의 위치

JAVA_HOME
C:\Program Files\Java\jdk-17

② 경로 설정

Path
C:\Program Files\Java\jdk-17\bin;
%JAVA_HOME%\bin;

카멜의 법칙

1. 클래스명의 첫 글자는 대문자로 시작하고 나머지는 소문자로 작성하고 단어가 바뀌면 다시 첫 글자를 다시 대문자로 시작한다.
2. 변수나 메소드의 첫 글자는 소문자로 시작하고 단어가 바뀌면 다시 첫 글자를 다시 대문자로 시작한다.
3. **상수는 전부 대문자로 지정한다 (상수 : 변하지 않는 값)**
4. Java의 파일명은 반드시 **클래스명과 동일**해야 한다.
5. 자바가상머신(JVM)이 가장 먼저 찾는 것은 **public static void main(String[] args)** 이다
   단, Servlet이 제일 먼저 찾는 것은 public void init()이다.
6. 문장의 끝에는 반드시 **;** 를 써야 한다.

[컴파일] **javac 파일명.java**
[실행] **java 파일명**

[예]
HelloTest.java
↓ 컴파일 (기계어+문법)
HelloTest.class
↓ 실행
HelloTest.exe를 만들지 않는다. (자바에서는)
자바가상머신(JVM)에 의해서 한 단계 한 단계씩(interpreter 방식) 실행한다.

bit

: 정보처리의 최소단위
: 0 또는 1

byte

: 1개의 영문자 또는 숫자를 의미한다
: 1byte = 8bit
: 한글 1자는 2byte (16bit)

상수 (Constant) C언어 : COST

: 변하지 않는 값
: 전부 **대문자**로 기술한다. :
숫자 상수 : 25, 1000, -78
문자 상수 : '1개 문자' (2byte) → **uni code**
"문자열" → ASCii 코드값이 **없다**
주소 전달

[예] 'A' '차'
[예]

	컴파일		10진수	ASCII
'A'	-------->	0100 0001	--------->	65
'B'	-------->	0100 0010	--------->	66
'C'	-------->	0100 0011	--------->	67
'a'	-------->	0110 0001	--------->	97
5	-------->	0000 0101	--------->	5
'5'	-------->	0011 0101	--------->	53

변수(variable) - 메모리 할당(instance)

: 변하는 값
: 데이타를 저장하는 곳

[형식]
자료형 변수명;

변수명 규칙

: 영문자, 숫자(0~9), _ , $ 를 섞어서 사용 가능
: 첫 글자는 소문자로 기술한다.
: 단어가 바뀌면 첫 글자를 대문자로 사용

자료형의 종류

한글은 자바에서 1자=2byte(기본)
Oracle : 3byte

1. 기본형

논리형 - boolean (1bit) - true or false
문자형 - char (2byte) → uni code 0 ~ 65535
정수형 - byte (1byte) → -128 ~ +127
- short (2byte) → -32768 ~ +32767
- int (4byte)
- long (8byte)

실수형 - float (4byte)
- double (8byte) - default

2. 객체형

String (문자열)

3. 배열

진수

① 2진수 :
② 8진수 :
③ 10진수 :
④ 16진수 :

[예] 234 → 10진수
0234 → 8진수 : 05 = 5
0x234 → 16진수 : 0xa = 숫자상수 10 (0x는 16진수를 나타내는 표시)

'a' + 3
"a" + 3
0xa + 3
a + 3

25 → int 정수
25l → long형 상수
25L

43.8 → double(기본)
43.8f → float형 상수
43.8F

① int a = 23L;
② float a = 43.8;
③ byte a = 300;
④ int a = 'B';
⑤ char a = 66;

short a=10;
short b=5;
short sum=a+b;
→ **char or byte or short 형으로 계산을 하면 결과는 int형으로 반환한다.**

연산자

1. 산술연산자

* /

+ -

% (나머지 연산자) - 반드시 양쪽의 항이 정수형이어야 한다.

2. 관계연산자

>	크다
<	작다
>=	크거나 같다
<=	작거나 같다
==	같다
!=	같지 않다

3. 논리연산자

조건이 2개 이상 존재할 때

&& (AND) : 모든 조건이 참일 때 성립

|| (OR) : 조건 중에서 하나만 참이어도 성립

& (AND)

| (OR)

x	y	x && y	x || y
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

4. 조건연산자

[형식] 조건 ? 참 : 거짓;

5. 증감연산자

++ (1씩 증가)

-- (1씩 감소)

독립적으로 혼자 쓰일 때는 선행, 후행연산을 따지지 않는다.

6. 대입연산자

=

+=

-=

*=

/=

%= 등

7. 부정연산자(NOT)

!

!true → false

!false → true

메소드()

① 명령어들의 집합
② 기본적으로 **public static void main(String[])** 제공
③ 인수는 반드시 따로 따로 선언한다.
④ 호출한 메소드는 반드시 호출한 곳으로 되돌아온다.
⑤ return(결과값, 반환값)은 반드시 1개뿐이다.
⑥ return이 없으면 void로 선언한다.

[형식]

1. 구현

returnType 메소드명(인수형 인수, 인수형 인수,...) **{**
    메소드body
}

2. 호출

객체명.메소드명(값1, 값2,...)
클래스명.메소드명(값1, 값2,...) → 메소드가 static 일 경우

기본형	Wrapper Class
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

AutoBoxing
unAutoBoxing

bit 연산자 - 0 or 1

1. 1의 보수 연산자(bit NOT)

~0→1

~1→0

8421코드

음수의 경우에는 8421코드가 적용되지 않는다

Java는 음수의 표현으로 2의보수(1의보수+1)를 사용한다

최상위 비트를 부호비트로 사용한다 (0:+, 1:-)

2. 논리연산자(bit)

		AND	OR	XOR
X	Y	X&Y	X|Y	X^Y
0	0			
0	1			
1	0			
1	1			

[문제] 20^7

0xca & 18

3. shift연산자(bit연산자) - 이동연산자

1) 왼쪽shift (<<)

: 왼쪽으로 1칸 이동시마다 *2씩 커진다

: 빈공간은 0으로 채운다

2) 오른쪽shift (>>)

: 오른쪽으로 1칸 이동시마다 /2씩 작아진다

: 빈공간은 부호비트으로 채운다

3) 오른쪽shift (>>>)

: 빈공간은 0으로 채운다 - 양수화

조건문

if

1. if(조건) 참 일 때;
   ↓ 거짓 일 때 (다음 문장 수행)

2. if(조건) 참 일 때;
   else 거짓 일 때;

3. if(조건) 참일때; → 다중 if문
   else if(조건) 참일때;
   else if(조건) 참일때;
   else if(조건) 참일때;
   else 거짓일때;

switch

- **실수형 타입은 쓸 수 없다**
- break 는 switch를 벗어나라

[형식]

```
switch(정수 or 문자 or 문자열 or 식) {
case 값1: break;
case 값2: break;
case 값3: break;
default :
}
```

반복문

1. for
2. while
3. do~while

1. for

[형식]

```
for(변수명=초기치; 조건치; 증감치){
        참일때
}
```

2. while

[형식]

```
while(조건){
   참일때
}
```

3. do~while

[형식]

```
do{
   참일때
}while(조건);
```

다중 for

1. for문안에 또 다른 for문이 존재
2. 서로의 변수명은 달라야 한다.
3. 겹쳐서도 안된다.

break

: **switch**, 반복문(**for, while, do~while**)를 벗어날 때
: 자신이 소속된 곳 1번만 벗어난다.
: 만약에 2개 이상 벗어나려면 라벨을 사용 한다

continue

: **반복문(for, while, do~while)**의 끝으로(**}**) 무조건 이동 :
자신이 소속된 끝으로(**}**) 무조건 이동
: 만약에 2개 이상 반복문 끝으로 이동하려면 라벨을 사용 한다

Array

1. 동일한 자료형의 모임
2. 연속적인 메모리 할당
3. 첨자가 0부터 시작
4. 한번 잡은 배열의 크기는 수정할 수 없다

1. 1차원 배열

[형식]

- 자료형[] 배열명 = {값1, 값2,...};
- 자료형[] 배열명;
  배열명 = new 자료형[개수];
- 자료형[] 배열명 = new int[]{값1, 값2,...}; -> int[ ] 개수를 넣으면 안된다.

ex)

1. int[] ar = {10, 20, 30};
2. int[] ar;
   ar = new int[3];
   ar[0] = 10;
   ar[1] = 20;
   ar[2] = 30;
3. int[] ar = new int[]{10, 20, 30};
4. int[] ar = new int[3]{10, 20, 30}; - X
5. int[] ar;
   ar = {10, 20, 30}; - X

2. 다차원 배열

- 생성할 때 행의 개수 꼭 써야 한다

2-1. 고정길이

- 자료형[][] 배열명 = { {값1, 값2,...}, {값3, 값4,...}, {...},...};
- 자료형[][] 배열명;
  배열명=new 자료형[개수][개수];

2-2. 가변길이

- 자료형[][] 배열명;
  배열명 = new 자료형[개수][];
  배열명[0] = new 자료형[개수];
  배열명[1] = new 자료형[개수];
  배열명[2] = new 자료형[개수];

객체지향언어 - OOP(Object Oriented Programming)

: 모의실험을 목적으로 사용
: 실제 사물의 속성(데이터)과 기능(메소드)을 정의하여 가상세계를 구현
: 모의실험을 통해 많은 시간과 노력을 절감
: 객체지향이론은 캡슐화, 상속, 추상화 개념
: 코드의 재사용이 높다
: 유지보수가 용이하다
: 캡슐화, 상속, 다형성의 특징

클래스

static 을 제외하고 모든 class는 new 로 생성해야한다.

객체를 정의한 것

[형식]

```
class  클래스명 {
    접근제한자 자료형 필드명;
    접근제한자 메소드() {
    }
    class Inner클래스명 {
    }
}
```

특징

1. 캡슐화 S O L I D
2. 상속성
3. 다형성

Overload 메소드 Overloading

하나의 클래스 안에서 **똑같은 이름**의 메소드가 2개 이상 존재 할 때
인수(매개변수) 형이 틀리거나
인수(매개변수) 개수가 틀린 경우

```
class AA {
    public void sub(int a){}        "하나의 class" 안에서 "같은 이름의 메
}                                   소드 함수"

class BB {
    public void sub(int a){}
    public void sub(String a){}          overload
    public void sub(char a){}
    public void sub(int a, int b){}
    public void sub(){}
    public int  sub(int a){}    overload X, 1번줄과 같은 함수.
}
```

생성자 (Constructor) 메소드

: 객체 초기화

1. 생성자명은 반드시 클래스명과 동일하다.
2. 자동호출 - 클래스를 메모리에 생성 시(new 할때)
3. returnType(결과형)이 없다.
4. 클래스 안에 생성자가 하나도 없을 시에는 자동으로 기본(default) 생성자 호출
   기본 생성자 - 인수가 없는 메소드

생성자 Overload

this

1. 생략가능
2. 자기 자신 클래스의 정보(reference-참조값)를 갖고 있다.

this()

1. Overload 된 다른 생성자를 호출할 때
2. 생성자에서 반드시 첫줄에 써야 한다.

varargs (Variable Argument)

: JDK 5.0에서 추가

: 통일된 인수의 자료형에 인수(매개변수)의 개수를 자유롭게 구현

: 내부적으로 배열화 작업으로 처리해 준다.

String

1. 문자열
2. **literal("") 생성 가능**
3. 문자열 **편집(수정)**을 할 수 없다.
4. 비교시 == 사용하면 reference(참조값)을 비교 한다
   equals() 사용하면 문자열의 내용을 비교한다.

※ 문자열 자체가 reference(참조값) 이다.
※ 똑같은 내용의 문자열 리터럴은 메모리에 1번만 생성된다.
   new는 할 때마다 메모리에 생성 된다

StringBuffer / StringBuilder

1. 문자열
2. 문자열 편집이 가능

3. append(), delete() 존재 (String class 에 없음)

StringTokenizer(java.util)클래스와 String클래스의 split()

1. 문자열을 분리할 때 사용
2. 분리된 문자열을 Token이라고 한다.
3. StringTokenizer는 비어있는 값은 무시하고 split()는 비어있는 값도 인식한다.

static - 클래스변수

1. 메모리 static 영역에 1번만 생성된다. → 초기화 1번만 수행 모든 객체가 공유한다. (공유변수)
2. static 메소드에서는 static 변수만 사용 가능 static 메소드에서는 this를 참조할 수 없다
3. static변수나 메소드는 호출시 클래스명으로 직접 호출 할 수 있다. 객체로도 호출이 가능하다
4. static{ } - 초기화 영역
   - 생성자보다도 먼저 수행한다.

import static

: 간단하게 static 상수 또는 메소드를 호출할 때 사용

상속 (inheritance)

: **is~a** 관계 ~이다
: 클래스의 재 구현

1. 상속받는 클래스는 상속해주는 클래스의 생성자와 private를 제외한 모든 것을 상속받는다.
2. Super class : 상속해 주는 클래스(부모)
   Sub class : 상속받는 클래스(자식)
3. 접근제한자 protected는 Sub class에서 접근이 가능하다
4. Sub class로 객체를 생성하면 Super class와 자신의 생성자를 모두 호출한다.
5. 다중상속을 할 수 없다.

[형식]
class Sub클래스명 **extends** Super클래스명{ }

Override 메소드

1. Super클래스와 Sub클래스에 똑같은 메소드가 존재
2. 모든 우선권은 Sub클래스가 갖는다.
3. Super, Sub 클래스의 접근제어자(Modifier)는 틀려도 되지만
   Super보다 Sub클래스의 접근제어자(Modifier)가 더 커야한다.

this 와 this()

1. this 는 자신의 클래스의 참조값을 갖고 있다
2. this() 는 Overload된 다른 생성자를 호출 할 수 있다.
3. this()는 생성자의 **첫줄**에 써야 한다

super 와 super()

1. super 는 부모클래스의 참조값을 갖고 있다.
2. super() 는 부모클래스의 생성자를 호출 할 수 있다.
3. super() 는 생성자의 **첫줄**에 써야 한다.

색

: 32bit (4byte)

: RGB (빛의 3요소)

R	G	B	
1byte	1byte	1byte	
0~255	0~255	0~255	
00~FF	00~FF	00~FF	
0000 0000	0000 0000	0000 0000	검정색
1111 1111	1111 1111	1111 1111	흰색

instanceof

: casting(형 변환)이 되는지 안 되는지를 판별

: 객체에 원하는 클래스 타입이 메모리 할당되었는지 안 되었는지를 확인

final (상수화)

1. final 변수는 값을 변경할 수 없다.
2. final 변수는 반드시 초기값을 주어야 한다.
   final 필드는 생성자에서 초기값을 주어야 한다
   static final 필드는 static 구역에서 초기값을 주어야 한다
3. final 변수는 대문자로만 기술
4. final 메소드는 Override를 할 수 없다.
5. final 클래스는 자식클래스를 가질 수 없다.- 상속이 안된다

package

1. 서로 관련이 있는 *.class 파일들의 모임
2. 맨 첫줄에 1번만 기술할 수 있다
3. 소문자로 기입
4. 자바가 제공하는 기본 패키지 **java.lang (default package)**이다

접근제한자 (Modifier)

	클래스	같은 패키지	다른 패키지	다른 패키지 자식 클래스
private	O	X	X	X
default	O	O	X	X
protected	O	O	X	O
public	O	O	O	O

※ default라고 직접 쓰는 것이 아니라
   public, protected, private 를 쓰지 않은 상태

※ protected는 다른 패키지에서 Sub 클래스라면 접근이 가능하다
   단 Sub 클래스로 생성해야만 한다
      Super클래스로 생성하면 접근이 안된다

enum (열거형)

: 자바의 열거형은 자료형(Data Type)을 의미한다
: 서로 관련 있는 상수들을 모아 놓은 것
: enum 상수들은 묵시적으로 static final status형으로 선언된다
: 먼저 자료형을 선언한 다음에 사용한다
: 대문자로 사용
: 열거된 순서에 따라 0부터 시작

Object

1. Java의 최상위 클래스
2. Java의 모든 클래스는 Object로 부터 상속받는다
3. extends Object라고 직접 쓰지 않아도 된다
4. Object에서는 == , equals() 가 모두 참조값(reference) 만으로 비교한다.
   단, String만이 equals()가 내용(문자열)을 비교한다.

추상클래스 - Sub class 제어

1. 추상화 작업
2. 메소드에 body { } 가 없는 메소드를 추상메소드라고 한다.
   추상메소드에는 **abstract** 라는 키워드를 써야 한다
   추상메소드는 {} body 없이 ;를 써야한다
3. 추상메소드가 있는 클래스는 반드시 추상클래스이어야 한다.
4. 추상메소드가 없는 추상클래스를 의미상의 추상클래스라고 한다.
   의미상의 추상클래스의 메소드는 모두 **빈body**로 되어 있다.
5. 추상클래스는 자신의 클래스로 메모리 생성을 할 수 없다
   => 생성하려면
   가. Sub Class를 이용 (반드시 Sub Class가 추상메소드를 Override 해야 한다)
   나. 메소드를 이용
6. 추상메소드는 반드시 Sub Class에서 Override 꼭 해 주어야 한다.
   Override를 안하면 Sub Class 마저도 abstract 가 되어야 한다.

주석

1. // 1줄 주석
2. /*
   2줄 이상 주석
   */

3. /**
   사용자 API 문서에서 주석을 작성할 때
   */

interface

- is~a 관계
- 틀

1. 표준명세서의 역활
2. 상수와 추상메소드만 존재
   public static final는 생략가능
   abstract는 생략가능
3. interface를 implements 한 클래스는 반드시 추상메소드를 Override(재구현)해주어야 한다.
4. Override(재구현) 할때 반드시 public를 붙여야 한다.
5. 다중상속이 가능
6. 상속과 같이 쓰일 때는 extends, implements 순서로 쓴다.

[형식]

```
interface 인터페이스명 {
    .....
}
class 클래스명 implements 인터페이스명{
    ...
}
```

[EX] 맞는 문장을 모두 고르시오

```
class A{}
interface InterA{}
```

1. class B extends A { }
2. class B implements InterA { }
3. class B implements A { }
4. class B extends InterA { }
5. interface InterB extends A { }
6. interface InterB implements InterA { }
7. interface InterB implements A { }
8. interface InterB extends InterA { }

중첩클래스

has~a 관계

다른 클래스 내부에 정의 되는 클래스를 중첩클래스(nested class)라고 한다.

중첩클래스는 독립적으로 오브젝트로 만들어질 수 있는 스태틱 클래스(static class)와 자신이 정의된 클래스의 오브젝트 안에서만 만들어질 수 있는 내부 클래스(inner class)로 구분된다.

내부클래스는 다시 범위에 따라 세 가지로 구분된다.

멤버필드처럼 오브젝트 레벨에 정의되는 멤버 내부 클래스(member inner class)와 메소드 레벨에서 로컬 변수를 선언하여 사용하듯 선언된 메소드 내에서만 사용 가능한 로컬 내부 클래스(local inner class), 그리고 이름을 갖지 않는 **익명 내부 클래스(anonymous inner class)**다.

Member Inner Class

안쪽에 있는 클래스는 바깥쪽 클래스의 모든 멤버에 접근 가능

하지만 바깥쪽 클래스는 안쪽의 클래스의 멤버에 접근 불가능

단 안쪽의 클래스로 객체를 선언하면 접근 가능하다.

Generic

: 제네릭은 동적으로 타입을 결정하지 않고 컴파일 시 타입이 결정되므로 보다 안전한 프로그래밍이 가능하다
: 실행 중에 타입 충돌 문제를 방지할 수 있다
: 프로그램 개발 시 타입 캐스팅 절차가 필요 없어지고 ClassCastException을 방지 할 수 있다
: 클래스 생성 시 타입을 명시해 준다.
: **<T> <E> <K> <V>** 4개의 문자로 표시한다.

<T> Type (데이터형)
<E> Element (요소, 항목)
<K> Key
<V> Value

Collection (java.util)

: **객체**를 담아주는 저장 창고
: 객체 타입에 상관없이 저장 가능
: 크기 조절 가능

Enumeration
Iterator

자바가 제공하는 어노테이션

1. @Override
2. @Deprecated
3. @SuppressWarnings(옵션)

옵션

1. all : 모든 경고를 억제
2. cast : 캐스트 연산자 관련 경고 억제
3. dep-ann : 사용하지 말아야 할 주석 관련 경고 억제
4. deprecation : 사용하지 말아야 할 메소드 관련 경고 억제
5. fallthrough : switch문에서의 break 누락 관련 경고 억제
6. finally : 반환하지 않는 finally 블럭 관련 경고 억제
7. null : null 분석 관련 경고 억제
8. rawtypes : 제네릭을 사용하는 클래스 매개변수가 불특정일 때의 경고 억제
9. unchecked : 검증되지 않은 연산자 관련 경고 억제
10. unused : 사용하지 않는 코드 관련 경고 억제

Comparable / Comparator 인터페이스

: 객체를 정렬하는데 필요한 메소드가 정의되어 있다
: 비교대상자가 더 크면(<) -1, 같다면(==) 0, 작으면(>) 1을 반환시켜줍니다.
: 이 값을 비교하여 정렬을 할 수 있다.

일반적인 int, char, double 같은 타입의 배열이라면 다음 함수들을 이용
Collections(또는 Arrays).sort()　　: 정렬
Collectoins(또는 Arrays).reverse()　　: 뒤집기

1. Comparable (java.lang)

- 기본 정렬기준을 구현하는데 사용
- 주로 Integer와 같은 warpper클래스와 String, Date, File과 같은 것
- 기본적으로 오름차순으로 정렬되도록 구현되어 있다
- public int compareTo(T o)

public final class Integer
extends Number
implements Comparable<Integer>

2. Comparator (java.util)

- 기본 정렬기준 외에 다른 기준으로 정렬하고자할 때 사용
- public int compare(T o1, T o2)
  public boolean equals(Object obj)

Collections.sort()

객체 정렬

예외처리

: 생각지도 않은 error가 발생하여 프로그램이 중도에 멈추는 것을 미리 예방하는 것

1. 예외처리의 최 상위 클래스는 Exception 이다
2. Exception이 여러 개 발생을 하면 한 번에 최 상위 클래스 Exception으로 처리하는 것이 편하다.
3. 자바가 제공하는 Override한 메소드에는 throws 사용해서는 안 된다.
4. 컴파일 Exception
   - RuntimeException 으로부터 상속을 받지 않기때문에
     반드시 Exception을 처리한다 => **try~catch, throws**

   실행 Exception
   - RuntimeException 으로부터 상속을 받으며
     Exception을 처리하지 말고 **비즈니스 로직**으로 처리한다.

※ 처리

1. try {
       error가 발생할 가능성이 있는 영역
   } catch( ){
       error가 발생하면 처리되는 부분
   }

2. try {
   }catch( ) {
   }catch( ) {
   }

3. try {
   } catch( ) {
   } finally {
       error가 있건 없건 무조건 실행하는 부분
   }

4. throw**s** - JVM에게 떠넘기는 것
5. throw - 사용자가 직접 Exception을 작성하여 발생

IO Stream

입출력 처리

* 단위

1. **byte 단위** 처리 (숫자, 영문자) - byte 스트림
   InputStream
   OutputStream
2. **문자(char-2byte) 단위** 처리(숫자, 영문자, 한글) - 문자 스트림
   Reader
   Writer

출력 시 파일 없으면 자동으로 파일 생성
입력 시 파일 없으면 Error (FileNotFoundException)

객체 직렬화

: 객체는 파일이나 네트워크로 전송이 안된다
: 객체를 byte[] 변환시켜서 전송해야한다
: Serializable

Application Serializable File

Java File