-------------------------------------------------------------160905----------------------------------------------------------

**자바 개발 환경 구축하기**

JDK 설치

1. 버전은 3가지를 가지고 있음
   1. JAVA SE 🡪 Standard Edition
   2. JAVA ME 🡪 모바일 2D 시절에 사용하던 플랫폼
   3. JAVA EE 🡪 Enterprize edition
2. <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
   1. JAVA SE에서 JDK 선택
   2. 라이선스 동의
   3. Java SE Development Kit 8u101 에서 운영체제에 맞는 파일을 선택하여 다운로드 받기
   4. 다음 ,다음 눌러서 기본폴더에 설치하기
   5. 중간에 설치 폴더 변경창이 발생하면 기본폴더를 그냥 사용한다.
3. Dos 창 실행
   1. Java 설치확인 (CMD에서 명령어 입력)
      1. Java
      2. Javac
         * 실행이 안될 경우
           + C:\Program Files\Java\jdk1.8.0\_101\bin 로 이동하여 경로명을 복사한다.
           + 시스템속성 -> 고급탭 -> 환경변수 -> 시스템변수에 Path 변수값을 선택하여 편집을 클릭
           + 변수 값을 클릭하여 맨 뒤에 ;을 추가한 후 C:\Program Files\Java\jdk1.8.0\_101\bin을 추가한다. 앞뒤 공백이 들어가지 않도록 주의한다.
      3. Java –version
4. Java는 사용자 생성 // javac는 컴퓨터 언어로 ‘컴파일러’

**Java API**

<http://www.oracle.com/technetwork/java/javase/documentation/index.html> -> JAVA SE Tech Document 확인가능 (all book -> API documentation)

실행을 위한 기본형식:

*class 이름 (첫글자는 대문자로 적는다)*

*{*

*public static void main(String[] args)*

*{*

*실행구문*

*}*

*}*

파일을 저장할 때는 클라스 이름을 대소문자까지 일치시키고 확장자 명은 java로 저장한다.

**컴파일하기**

DOS창 열기

파일을 저장한 폴더로 이동하기

**Javac 파일명.확장자** 로 실행한다.

정상 실행되는 경우 아무런 메시지가 출력되지 않는다.

오류 메시지가 출력되면 코드를 확인한다.

컴파일 후 Hello.class가 생성된다.

이후 게임 실행할 때는 확장명을 제외하고 파일명을 입력하여 실행한다. (java 파일명)

파일작성 규칙

1. 기본적으로 하나의 자바파일에 하나의 클래스 파일을 이용한다.
   * 만약 하나의 자바 파일로 여러 개의 클래스 파일을 만들 경우
     1. 일반적인 class 파일만 사용하는 경우에는 아무거나 사용가능
     2. Public이 있는 경우에는 반드시 해당 파일을 사용한다.
     3. Public은 반드시 1개만 사용이 가능하다.
     4. public은 반드시 메인 함수가 있는 쪽에만 사용이 가능하다.

주석방법

범위 주석 /\* \*/

한 줄 주석 //

**변수**

**변수 선언**

**형식:**

*Datatype 변수명;*

* + 기본형 (8개)
    1. int 변수명;

예) int a ; 🡸 선언

a = 10; 🡸 초기화

* + 1. int 변수명 = 초기화값 ; 🡸 선언과 초기화를 같이 진행가능

a = 20 ; 🡸 바꿀 수 있음

* + 1. int a; 🡸 선언

int b; 🡸 선언

int c; 🡸 선언

a = 10; 🡸 초기화

b = 20; 🡸 초기화

c = 30; 🡸 초기화

* + 1. int a, b, c ; 🡸 선언

a= 10; 🡸 초기화

b= 20; 🡸 초기화

c=30; 🡸 초기화

* + 1. int a=10, b=20, c=30; 🡸 타입이 같은 경우 함께 사용가능
    2. ~~int a =10, long 1=20;~~ 🡸 타입이 바뀌는 경우 함께 사용불가
  + 참조형 (무한대)

1. 출력명령어 *\*첫글자는 무조건 대문자로 입력한다.*

줄 그대로 출력: System.out.print (변수명 또는 “문자”)

줄을 바꿔서 출력: System.out.println()

형식화된 출력방법: System.out.printf( “형식문자“ , 값1, 값2… )

|  |  |
| --- | --- |
| 지시자 | 설명 |
| %b | 논리값 (boolean) |
| %d | 10진수 integer |
| %o | 8진수 |
| %x, %X | 16진수 |
| %f | 10진수 float |
| %e,%E | 지수형태표현 |
| %c | 문자 |
| %s | 문자열 |
| %n | 줄바꿈 |
| %g | 이하자리 생략 |

-------------------------------------------------------------160906----------------------------------------------------------

변수 = 값

X = Y 🡸 X는 Y가 가진 값을 가져간다는 뜻

Swap algorism (x와 y를 맞교환 하는 방법)

*int x = 10*

*int y = 20*

*int tmp*

*tmp = x*

*x = y*

*y = tmp*

명명 규칙 (변수, 클래스, 함수에 공통적으로 적용된다)

- 대소문자 구분되며 길이에 제한이 없음

- 예약어(키워드)는 사용하면 안 된다

- 숫자로 시작하면 안된다.

- 특수문자는 \_ 와 $만 된다.

- \*한글로도 만들 수는 있지만 사용은 하지 않는다.

클래스명의 묵시적인 규칙

- 첫문자를 반드시 대문자로 사용한다.

- 조합어를 사용하는 경우 두번째 문자도 대문자를 사용한다.

변수명의 묵시적인 규칙

- 소문자로 시작한다.

- 조합어를 사용하는 경우 두번째 문자도 대문자를 사용한다.

함수명의 묵시적인 규칙

- 소문자로 시작한다.

- 조합어를 사용하는 경우 두번째 문자도 대문자를 사용한다.

- 뒤에 반드시 ()가 따라온다

\*상수의 묵시적인 규칙

- 전부 대문자로 사용하며 두단어이상인 경우 \_ 로 구분한다.

기본형 8개

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | boolean | char | byte | short | | int | | long | float | double |
| 분류 | 논리형 | 문자형 | 정수형 | | | | | | 실수형 | |
| BYTE | 1 byte | 2 byte | 1 byte | 2 byte | 4 byte | | 8 byte | | 4 byte | 8 byte |
| BIT | 8 bit | 16 bit | 8 bit | 16 bit | 32 bit | | 64 bit | | 32 bit | 64 bit |
| 기본값  접미사 | false | ‘ ‘ 공백  ‘\u0000’ | 0 | 0 | 0l | | 0l | | 0.0  F/f | 0.0  [D/d] 생략가능 |

문자

- 한문자만을 사용한다.

표기방법: ‘ ‘

\*문자열은 기본형은 아니지만 기본형처럼 사용한다.

표기방법: “”

숫자

- 정수 : 2, 8, 10, 16

- 실수 : 소수점이 있는 정수

\*삼각함수는 공부할 것

기본형과 참조형의 차이점

기본형은 값을 저장하지만 참조형은 메모리의 주소값을 저장한다.

문자값은 유니코드를 통하여 숫자값으로 저장한다.

정수형의 경우 음수의 범위도 표현한다.

상수를 선언하는 방법: 상수의 값은 **literal** 이라고 한다.

- 변수 앞에 final을 붙여준다.

- 선언과 함께 초기화 해야 한다.

- 이후에는 변경이 되지 않는다.

리터럴에서 각 진수별로 표현하는 방법

|  |  |  |
| --- | --- | --- |
| 표현방법 | 진수 | 10진수 |
| 010 | 8진수 | 8 |
| 0x10 | 16진수 | 16 |
| 0b10 | 2진수 | 2 |

변수를 지정할 때는 타입까지 일치 시켜야지 같은 데이터로 인지한다.

타입이 일치하지 않아도 예외적으로 인정하는 경우가 있다. (변수의 허용 크기가 값보다 큰 경우)

문자열 String S = “A”

문자 Char C = ‘A’

String의 기본값은 null

\*공백없이 ””를 사용할 수 있으며 값이 없다는 뜻으로 사용한다. “ “ 을 줄 경우 공백이 있는 문자열로 인식한다.

Char의 기본값은 ‘ ‘

\*공백없이 ‘’ 붙여서 사용하는 것은 허용하지 않는다.

-------------------------------------------------------------160907----------------------------------------------------------

실수는 소수점 뒤 6자리까지 찍힌다.

%4.2f

. 앞은 정수,

. 뒤는 소수

C 언어에서는 %g를 사용하면 뒷자리에 0은 모두 제외가 된다.

JAVA에서는 C언어와 똑같이 작용되지는 않는다.

\*1e1f 🡸 1x 101x1

Scanner (입력) -> 콘솔 (키보드) 또는 파일

Scanner scanner = new Scanner (system.in) 🡸 키보드에서 입력하는 데이터를 받는다.

scanner.nextLine() 🡸 문자열로 받는다

scanner.nextInt() 🡸 숫자로 받는다

문자를 숫자로 변경하기

Integer.parseInt() 🡸 숫자로 된 데이터를 문자열로 받을 경우 정수 타입으로 바꾼다.

10진법의 소수를 2진법으로 변환하기

0.625 🡺 0.101

소수자리만 취해서 2를 곱해 준다.

0.625 x 2 = 1 . 25

1.25 x 2 = 0 . 5

0.5 x 2 = 1 . 0

0.101

0.101 🡺 1 x 1/2 + 1 x 1/4 + 1 x 1/8

10진수 10의 경우

Byte b =10;

**0**0001010 (2)

**맨앞에 0은 Sign bit로 0은 양수 1은 음수로 처리한다.**

음수로 변경할 때는 맨 앞에 1로 변경하면서 나머지 수는 전환을 하고 1을 더해준다.

Byte b =-10;

**0**0001010 (2)

11110101 (2) + 00000001 (2)

**1**1110110 (2)  = -10

**기본형 (Primitive Type)**

논리형 boolean

둘 중에 하나 인 경우에만 사용 (false 또는 True)

문자형 char

UNICODE 체계

|  |  |  |
| --- | --- | --- |
|  | 10진수 | 16진수 |
| ‘0’ ~ ‘9’ | 48 ~ 57 | 0030 ~ 0039 |
| ‘A’ ~ ‘Z’ | 65 ~ 90 | 0041 ~ 005A |
| ‘a’ ~ ‘z’ | 97 ~ 122 | 0061 ~ 007A |
| 한글 | 44032~55207 | AC00 ~ D7A3 |

println 또는 print 에서는

|  |  |
| --- | --- |
| 탭 | ＼t |
| Backspace | ＼b |
| Form feed | ＼f |
| 새로운 줄 | ＼n |
| Carriage return | ＼r |
| 역슬레쉬 | ＼＼ |
| 작은 따옴표 | ＼’ |
| 큰따옴표 | ＼” |
| 유니코드 문자 | ＼u유니코드 |

\*C언어는 ASCII 기반 JAVA는 UNICODE 기반임.

-------------------------------------------------------------160908----------------------------------------------------------

실수형 – float, double

float - 6번째자리까지 정밀하게 표시되며 이하는 반올림이 된다.

double - 15번째자리까지 정밀하게 표시되며 이하는 반올림이 된다.

기본형 형변환하기

7가지 타입만 형변화가 가능하다 (boolean의 경우 참과 거짓이기 때문에 필요없다)

크기 순서

byte < short < int < long < float < double

char => byte와 char, 그리고 short와 char는 무조건 강제형 변화만 가능하다.

자동형변환 : 변환하고자 하는 타입이 더 크기 때문에 자동으로 변환이 가능

강제형변환 : (타입) 🡺 캐스팅 연산자를 사용하여 명시적으로 변환한다.

\*작은 타입의 변수에 큰 타입의 데이터를 입력하는 경우에는 값의 범위와는 상관없이 항상 강제형변화를 사용해야 한다.

타입 변환 중 값의 손실이 한다.

BYTE 계산 중에 계산가능한 저장공간이 잘려나가며 그 값은 **2진법 기준으로 변경**된다. (기본 버림) 그래서 10진법에서의 값 편차는 크게 달라질 수 있다.

연산 중에 타입이 다른 경우 작은 타입이 큰 타입으로 자동으로 변환이 된다.

\*기본형과 참조형은 서로 형변환을 할 수 없다.

int보다 작은 byte와 short는 연산할 경우 int 타입으로 변경한다.

\*String type의 경우 기본형이 아니기 때문에 연산이 되지 않고 이어서 쓴다.

연습문제

2-1

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

2-2

변수: regNo

주민등록번호를 숫자로 저장할 데이터 코드는?

🡺 **13자리 데이터가 필요하므로 long 타입을 사용한다.**

**long regNo =8203301234567L**

2-3

|  |
| --- |
| **int i = 100**  **long l = 100L;**  **final float PI = 3.14f** |

- 리터럴: **100, 100L, 3.14f**

- 변수: **i, l**

- 키워드: **final, int, long, float**

- 상수: **PI**

2-4

기본형이 아닌 것은? **B**

a. int

b. Byte 🡨 참조형이다

c. double

d. boolean

2-5

출력결과 적기, 오류나면 “오류”로 표시

System.out.println( “1”+”2” ) 🡺 ( **12** )

System.out.println( true +”” ) 🡺 ( **true** )

System.out.println( ‘A’+’B’ ) 🡺 ( **131** )

System.out.println( ‘1’+2 ) 🡺 ( **51** )

System.out.println( ‘1’ + ‘2’ ) 🡺 ( **99** )

System.out.println( ‘J’+ “ava” ) 🡺 ( **Java** )

System.out.println( true +null ) 🡺 ( **오류** )

2-6 키워드가 아닌 것은? **a, b, c**

a. if

b. True

c. NULL

d. Class

e. System

2-7 변수의 이름으로 사용 가능한 것은? **a, d, e, g**

a. $ystem

b. channel#5

c. 7eleven

d. If

e. 자바

f. new 🡸 키워드

g. $MAX\_NUM 🡸 묵시적인 약속이기 때문에 사용은 가능하다

h. hello@com

2-8 참조형 변수와 같은 크기의 기본형은? **4byte**

**a. int**

b. long

c. short

**d. float**

e. double

2-9 형변환을 생략할 수 있는 것은? **d. e.**

byte b = 10;

char ch = ‘A’;

int i = 100;

long l = 1000L;

a. b = (byte)i;

b. ch = (char)b;

c. short s = (shrot)ch;

d. float f = (float)l;

e. I = (int)ch;

2-10 char 타입의 변수에 저장할 수 있는 정수 값의 범위는?

**0~65535**

2-11 변수 초기화가 잘못 된 것은? **a, b, c, d**

a. byte b = 256;

b. char c = ‘’;

c. char answer = ‘no’;

d. float f =3.14;

e. double d = 1.4e3f;

2-12 선언부로 알맞은 것은? a

a. public static void main (String [] args)

b. public static void main (String args[])

c. public static void main (String [] arv)

d. public static void main (String [] args)

e. static public void main (String [] args)

2-13 타입과 기본 값이 잘못 연결된 것은? **c, e, f**

a. boolean – false

b. char – ‘\u0000’

c. float – 0.0 🡨0.0f

d. int – 0

e. long – 0 🡨0l

f. String – “” 🡨 null

-------------------------------------------------------------160909----------------------------------------------------------

**연산자**

연산한 값을 처리하기 위해서는 변수를 활용해야 한다.

연산 우선순위

오른쪽에서 왼쪽으로 처리하는 연산자

단항 연산자: ++ -- + - ~ ! (type) 🡨 casting

대입 연산자: = += -= \*= /= %= <<= >>= &= ^= |=

\*자바에서 비교연산 시 같다는 ==을 사용하고 다름은 !=로 표시한다.

**증감연산자**

변수와 함께 사용하되 전위와 후위로 위치를 조정한다.

단독으로 사용할 경우 전위와 후위에 차이가 발생하지 않는다.

전위: ++i 값이 참조하기 전에 증가시키다.

후위: i++ 값이 참조된 후에 증가시킨다.

println은 매개변수를 이용하여 처리되기 때문에 ( ) 안에 단독으로 기입해도 전위와 후위가 적용된다.

byte a = 10;

byte b = 20;

byte c = a + b;

byte와 byte가 결합하면 int로 타입이 변경되어 오류가 발생한다. 이 때는 a+b 결과 값을 묶어서 int로 변경해야 한다. (int) (a + b)

int 로 연산할 때 int가 허용하는 범위를 넘을 경우, 결과 값에 오차가 발생된다.

이를 해소 하기 위하여 값의 일부를 long등 더 높은 범위를 포함하는 타입으로 변경할 경우 다른 값도타입일치를 통하여 함께 변경된다.

대소문자 변화는 유니코드를 활용하여 32를 더하거나 빼서 진행한다.

소수를 원하는 자리까지만 표현하고 이하 자리를 버리고 싶은 경우

1. 원하는 자리까지 정수화 한다.

2. casting을 이용하여 소수를 버린다.

3. 다시 실수(float 또는 double) 로 나눈다.

소수를 원하는 자리까지만 표현하되 이하 자리를 반올림하고 싶은 경우

1. 원하는 자리까지 정수화 한다.

2. casting을 이용하여 소수를 버린다.

3. 값에 0.5를 더하고 다시 실수(float 또는 double) 로 나눈다. (round 함수 처리 방식과 동일하다. round함수는 별도의 지정이 없을 경우 소수점 첫 번째 자리에서 반올림한다. )

**나머지 연산자 %**

맨 앞의 연산자의 부호를 따른다.

+ % + 🡺 +

- % - 🡺 -

+ % - 🡺 +

- % + 🡺 -

**대소비교 연산자 < > <= >=**

True 또는 false 값이 반환된다.

10 == 10.0f 🡸 casting이 일어난다.

0.1d 과 0.1f 는 정밀도 차이로 인하여 다른 값으로 인지한다.

따라서 0.1d 와 float(0.1d) 또한 다른 값으로 인지된다.

\*String의 경우 축약형 사용이 가능하다.

String s=”abc”;

🡺 축약형: String s=new String (“abc”);

또한 문자열을 비교할 때는 == 가 아닌 equals( )를 사용한다.

**논리연산자 && || !**

&& 그리고

|| 또는

! ~의 반대 (결과의 반대 값을 취한다. )

&&와 ||를 함께 사용하는 경우 연산의 우선순위에 따라서 &&가 먼저 연산된다.

효율적인 연산을 한다. (SCE Shortcut Circuit Evaluation)

|| 좌측에 true가 오면 무조건 true이다

&& 좌측에 false가 오면 무조건 false이다.

좌측 연산에 위 결과가 올 것이라고 예상되는 연산을 먼저 기재하면 더 빠른 연산이 가능하다.

**실제로 좌측연산으로 결과가 확정되면 이후 연산은 진행하지 않는다.**

논리 부정 연산자는 입력값을 반대로 전환한다.

-------------------------------------------------------------160912----------------------------------------------------------

**비트 연산자 & | ^ ~ << >>**

& 🡺 그리고

| 🡺 또는

^ 🡺 같으면 0 다르면 1

~ 🡺 1 이면 0, 0 이면 1 \*음수 값을 표현하는 식: ~b+1 (2의 보수를 구하는 식)

<< 🡺 쉬프트 연산자, 이진법기준 왼쪽으로 숫자가 이하 수만큼 이동한다. X <<N 은 X\*2n

>> 🡺 쉬프트 연산자, 이진법기준 오른쪽으로 숫자가 이하 수만큼 이동한다. X >>N 은 X/2n

쉬프트 연산자는 처리속도가 약간 빠르지만 사용처는 많지 않다.

**조건 연산자 : ?**

조건식 ? 식1 : 식2 🡺 삼항연산자, 조건식이 참이면 식1을 거짓이면 식2를 취한다.

중첩해서 사용하면 3개 이상을 사용할 수도 있다.

예)

X > 0 ? 1 : (X==0 ? 0 : -1 )

조건식 ? (조건식? 식a : 식b) : (조건식? 식c : 식d)

\*피연산자의 타입이 다른 경우 이항연산자를 이용하여 자동변환 한다.

**대입 연산자 = op=**

|  |  |
| --- | --- |
| **=** | **op=** |
| X = X+10; | X += 10; |
| X = X-10; | X -= 10; |
| X = X\*10; | X \*= 10; |
| X = X/10; | X /= 10; |
| X = X\*(10+Y) | X \*= 10+Y |

**문제 1**

조건연산자의 중첩을 이용해서 3개의 변수 중 가장 큰 숫자 구하기

A = 3 , B = 10, C = 7

**<풀이>**

|  |
| --- |
| **class OperatorPg130**  **{**  **public static void main (String [] arge)**  **{**  **int a = 3;**  **int b = 10;**  **int c = 7;**  **int max = 0;**  **max = a > b ? (a > c ? a : c) : ( b > c ? b : c);**  **System.out.printf("가장 큰 값은 = %d%n", max);**  **}**  **}** |

**문제 2-1**

int x=2, y=10, z=0

z= x++ \*2 + --y -5 +x \* (y%2)

z=

x=

y=

**<풀이>**

|  |
| --- |
| **z = 4 + 9 - 5 + ( 3 \* ( 1 ) )**  **= 13 - 5 + 3**  **= 11** |

**문제 2-2**

int i = 1;

i= I | 0xff

i =

**<풀이>**

|  |
| --- |
| **32비트로 변환할 경우**  **…..00000001**  **| …..11111111**  **11111111 🡺 255** |

**문제 2-3**

x=y=z=1

b = ( ( x-- > 0) || (++y !0) && (--z !=0) );

b=

x=

y=

z=

**<풀이>**

|  |
| --- |
| **b=true**  **x=0**  **y=1**  **z=1**  **앞선 연산이 true와 or 이므로 이하 연산은 하지 않는다.** |

-------------------------------------------------------------160913----------------------------------------------------------

이클립스 설치하기 (<http://www.eclipse.org/>)

1. 실행 후 Workspace 지정하기 (java 파일을 저장하던 곳으로 이용한다. )

2. Welcome 탭을 닫는다.

사용처

Android, java, jsp 등에 사용이 가능. C언어의 경우 전용툴이 있어서 잘 사용하지는 않지만 사용이 가능하다.

시작 방법

New -> Other -> Wizard 창에서 Java Project를 클릭

Project 이름을 기입하고 생성한다.

Project 생성 후 좌측에 Package Explorer 창에서 src를 우측 클릭하고 팝업창에서 New -> Class를 클릭한다.

Name 항목에 **Class 명을 입력**한다.

이하에 public static void main(String[] args)가 자동으로 생성하게 하는 옵션을 클릭한다.

프로그램을 실행하기 위해서는 Run 버튼을 클릭한다.

글꼴 변경

Windows -> Preferences -> General -> Appearance ->Colors and Fonts

클래스 이름도 java 생성할 때와 동일하게 맞춰야 한다.

이름을 변경할 때는 F2키를 사용한다.

**조건문**

\*if는 true 또는 false가 결과로 나오는 논리연산이어야 한다.

1. 실행 후 거짓이면 무시한다.

*형식 – if (조건식) { }*

2. 실행 후 거짓이면 else를 실행한다. else는 생략가능

*형식 – if (조건식) { } else { }*

3. 여러 개의 조건식을 다중으로 처리한다.

*형식 – if (조건식1) { } else if (조건식2) { } else if (조건식3) { } else{ }*

조건을 만족하는 경우 이하 else if는 실행되지 않는다.

4. 여러 개의 조건식을 중첩으로 처리한다. 🡸 경우에 따라 논리연산자로도 표현이 가능하다.

*형식 – if (조건식) { 1 또는 2 또는 3 } else{ 1 또는 2 또는 3 }*

\*단일실행문일 경우에는 { }을 생략할 수는 있지만, 다중일 경우에는 논리적으로 오류가 발생된다.

\*equals 와 equalsIgnoreCase의 차이는 대소문자를 구분하는 지 여부임.

**<참고사항>**

Scanner scan = new Scanner (System.in);

사용 시

Resource leak: 'scanner' is never closed 오류 메시지가 발생한다.

이때는

리소스가 새는 것을 막기 위해서 Scanner.close( ); 를 사용하여 scanner를 닫아 주어야 한다.

([리소스 누수 관련 출처 바로가기](http://translate.googleusercontent.com/translate_c?depth=1&hl=ko&prev=search&rurl=translate.google.co.kr&sl=en&u=http://help.eclipse.org/kepler/index.jsp%3Ftopic%3D/org.eclipse.jdt.doc.user/tasks/task-avoiding_resource_leaks.htm%26resultof%3D%2522avoiding%2522%2520%2522avoid%2522%2520%2522resource%2522%2520%2522resourc%2522%2520%2522leaks%2522%2520%2522leak%2522%2520&usg=ALkJrhjxqsfASnigQQ3raCbSqlloy6bJfg))

참고로 @SuppressWarnings 경고창을 무시할 수 있지만 권장하지 않는다.

|  |
| --- |
| **all : 모든 경고를 억제합니다.**  **boxing : boxing/unboxing 오퍼레이션과 관련된 경고를 억제합니다.**  **cast : 캐스트 오퍼레이션과 관련된 경고를 억제합니다.**  **dep-ann : 권장되지 않는 어노테이션과 관련된 경고를 억제합니다.**  **deprecation : 권장되지 않는 기능과 관련된 경고를 억제합니다.**  **fallthrough : switch 문에서 누락된 break 문과 관련된 경고를 억제합니다.**  **finally : 리턴되지 않는 마지막 블록과 관련된 경고를 억제합니다.**  **hiding : 변수를 숨기는 로컬과 관련된 경고를 억제합니다.**  **incomplete-switch : switch 문에서 누락된 항목과 관련된 경고를 억제합니다(enum case).**  **javadoc : javadoc 경고와 관련된 경고를 억제합니다.**  **nls : 비nls 문자열 리터럴과 관련된 경고를 억제합니다.**  **null : 널(null) 분석과 관련된 경고를 억제합니다.**  **rawtypes : 원시 유형 사용법과 관련된 경고를 억제합니다.**  **resource : 닫기 가능 유형의 자원 사용에 관련된 경고 억제**  **restriction : 올바르지 않거나 금지된 참조 사용법과 관련된 경고를 억제합니다.**  **serial : 직렬화 가능 클래스에 대한 누락된 serialVersionUID 필드와 관련된 경고를 억제합니다.**  **static-access : 잘못된 정적 액세스와 관련된 경고를 억제합니다.**  **static-method : static으로 선언될 수 있는 메소드와 관련된 경고를 억제합니다.**  **super : 수퍼 호출을 사용하지 않는 메소드 겹쳐쓰기와 관련된 경고를 억제합니다.**  **synthetic-access : 내부 클래스로부터의 최적화되지 않은 액세스와 관련된 경고를 억제합니다.**  **sync-override : 동기화된 메소드를 오버라이드하는 경우 누락된 동기화로 인한 경고 억제**  **unchecked : 미확인 오퍼레이션과 관련된 경고를 억제합니다.**  **unqualified-field-access : 규정되지 않은 필드 액세스와 관련된 경고를 억제합니다.**  **unused : 사용하지 않은 코드 및 불필요한 코드와 관련된 경고를 억제합니다.**  [**출처 바로가기**](http://www-01.ibm.com/support/knowledgecenter/SSQ2R2_9.1.0/org.eclipse.jdt.doc.user/tasks/task-suppress_warnings.htm?cp=SSQ2R2_9.1.0%2F24-6-3-8-2&lang=ko) |

-------------------------------------------------------------160919----------------------------------------------------------

**Switch 문**

단순비교 (등가 연산)만 가능

Switch (**조건식**)   
{ case 값 1 :

실행문 ;

Break;

case 값 2 :

실행문 ;

Break;

default : 실행문

}

default와 break는 생략가능

단 break가 생략되면 다음 break가 있거나 case가 모두 종료될 때까지 이하 실행문을 모두 실행한다.

조건식에서 값은 정수타입만 사용 가능하지만 버전 1.7이상에서만 문자열을 사용할 수 있다.

case 값은 변수사용불가. 리터럴만 허용된다.

여러 개의 case문이 하나의 실행문을 공유하는 형태로 구현도 가능하다.

|  |  |  |
| --- | --- | --- |
| case 값1:  case 값2:  case 값3:  실행문;  Break; | 또는 | case 값1: case 값2: case 값3:  실행문;  Break; |

\*난수만들기 함수

Math.random( )

기본값: **0.0 <= ? <1.0**

필요한 수만큼 연산을 해준다. 예를 들어서 1,2,3을 사용하는 경우

(int)(Math.random( )\*3) +1

0.0 <= ? < 1.0

0.0 <= ? < 3.0

0 <= ? < 3

1 <= ? < 4

Switch문 또한 중첩으로 사용이 가능하다.

문제 1

- 중첩 if문으로 만든다.

- 2에 배수 이면서 3에 배수인 수를 확인한다.

문제 2

- 논리 연산자로 바꿔서 한다.

문제 3

- 최소 공배수를 이용한다.

**반복문**

|  |  |
| --- | --- |
| 형태 | 예제 |
| For ( 변수; 조건식 ; 증감식 )  { 실행문  } | For ( int i = 0 ; i < 10 ; i+ = 2 )  {  } |

무한반복문을 표현하는 방법

1. for ( ; ; ) {실행문}

2. for ( ; true; ) {실행문}

For 문 내에서 초기화한 변수의 경우 블록 { } 외부에서 해당 변수는 사용할 수 없다.

단 외부에서 선언한 변수를 내부에서 초시화만 하여 사용한다면, 내,외부 모두에서 사용할 수 있다.

For 문안에 또 다른 For 문을 넣어서 중첩으로 할 수 있음

기본형이 아닌 참조형을 위해서는 for ( : ) 형태로 사용한다. 이후 배열에서 자세한 설명예정

문제4

직사각형에서 3개의 점이 주어 질 경우 나머지 한 개를 구하라

예) (5,5) // (5,7) // (7,5) // (?,?)

-------------------------------------------------------------160921----------------------------------------------------------

**While 문**

|  |
| --- |
| **변수 선언 및 초기화**  **while (*조건식*)**  **{**  ***실행문*;**  ***증감식*;**  **}** |

조건식은 true false를 받을 수 잇는 비교연산자 또는 논리연산자를 사용한다.

초기값과 증감식에 대한 부분을 임의로 추가해야 함

변수 초기화는 while문 상단에 표기

while 은 조건식이 없으면 오류가 발생한다. 무한반복문 사용은 true라고 표기한다.

형식이 다른 반복문도 중첩하여 사용이 가능하다.

실행문에 ; 만 입력하면 내용이 없는 빈 문장으로 인식한다.

For문에서 조건식 뒤에 { } 없이 ; 만을 넣게 되면 반복 자체가 실행되지 않는다.

아무것도 입력하지 않고 { } 만 붙이는 것도 빈 문장으로 인식한다.

While 문에서 조건식을 boolean 타입을 이용한 변수로 선언하면 제어가 가능하다.

While문 안에 scanner로 입력을 하게되면 무한반복으로 입력이 가능하다.

**do while 문**

while 문 앞에서 실행하여 무조건 1번은 실행된다.

while 이하 조건식이 충족하는 동안에만 계속 실행된다.

**break 문**

반복문에서 이하 실행문이 수행되면 해당 반복문을 종료할 수 있다.

**continue 문**

반복문에서 continue 이하 실행문을 생략하고 현재 반복문의 증감식으로 이동하여 다시 한번 조건식의 반복 여부를 판단하여 계속 실행하는 지 여부를 확인할 수 있다.

반복문

문제1 : 1 - 2 + 3 – 4 + 5 – 6 ….. +99 -100 결과는 -50

문제2 : 1/1 1/2 1/3 1/4 …. 1/9 1/10 결과는 1/1 =1 1/2 = 0.5 1/3 =0.33333

문제3 : 1/1 + 1/2 + 1/3 + …1/9 +1/10 결과는 2.928968

-------------------------------------------------------------160923----------------------------------------------------------

***유클리드 알고리즘***

주어진 두 수 중

나머지가 0이 나올 때까지 나눈다

큰수 % 작은수 == 0이 아닌 수1

작은 수 % 0이 아닌 수1 == 0 이 아닌 수2

0이 아닌 수1 % 0이 아닌 수2 == 0 이 아닌 수3

이런 식으로 나머지 값이 0이 되면 나눈 수가 최대 공약수가 된다.

-------------------------------------------------------------160926----------------------------------------------------------

**배열**

**1차원**

여러 개의 변수를 선언한 경우 Type이 같다면 아래와 같이 배열을 이용하여 변수를 생성할 수 있다.

일반 변수와 다르게 메모리에 연속적으로 선언이 된다.

형식 1.

int**[ ]** 배열명 🡸 변수선언

배열명 = new int **[변수크기]**  🡸 생성: 생성할 때 변수의 크기를 지정한다.

변수 사용할 경우 배열명[인덱스번호 ] 🡸 사용: 인덱스 번호는 0부터 시작한다.

형식 2. 선언과 생성을 한 개로 합쳐서 사용한다.

int**[ ]** 배열명 = new int **[변수크기 ]** 🡸 생성할 때 배열변수의 크기를 지정한다.

형식 3. 초기화까지 합치는 방법.

int**[ ]** 배열명 = **{10, 20, 30, 40, 50}** 🡸 초기화 값의 수에 따라서 인덱스 번호가 자동으로 지정된다.

형식 4. for문을 활용하는 방법 (배열명.length로 배열의 길이를 자동으로 가져올 수 있다.)

int**[ ]** 배열명 = new int **[ i ]**

for (int i=0; i < 배열명.**length**; i++)

{

배열명 [ i ] = ( i+1 ) \* 10 ; 🡸 10 단위로 초기화 하는 방법

}

배열을 생성하면 4byte 짜리 변수명을 가진 공간을 생성하며 주소값을 저장한다. (C언어에서 직접제어를 할 때 활용할 수 있다. JAVA에서는 사용하지 않는다. )

\*만약 배열에 변수를 초기화하지 않는 경우 해당 배열의 기본값을 자동으로 사용한다.

\*Arrays 객체사용

Arrays.toString (배열명);

현재 가지고 있는 배열의 값을 문자열 형태로 표시한다.

[ 값1, 값2, 값3….. ] 형태로 출력된다.

\*일반 배열명만을 출력시키면 내부주소값이 출력된다.

예외적으로 println에서 char타입의 배열은 그냥 출력이 된다.

**배열의 복사 - 3가지 방식을 사용한다.**

이유: 기존의 배열을 사용 중 공간이 부족할 때 배열자체를 임의로 늘릴 수 없다. 하여 더 큰 배열을 참조하게 강제하여 배열을 복사한다. .

1. for문을 사용

🡺 더 큰 배열을 생성한 후 for문을 이용하여 값을 복사한다.

2. arraycopy( ) 을 사용

🡺 System.arraycopy (기존배열명, 배열시작값, 새로운배열명, 배열시작값, 크기)

🡺 위치를 자유롭게 사용할 수 있다.

3. arrays.clone( ) 을 사용

🡺 차후에 설명

**Swap (교환)**

변수이기 때문에 직접적인 교환은 불가능하기 때문에 임시 저장공간을 생성하여 변경한다.

int tmp = 변수b[위치값2]

변수b[위치값2] = 변수a[위치값1]

변수a[위치값1] = tmp

-------------------------------------------------------------160927----------------------------------------------------------

불규칙 난수를 만들 때 배열을 활용한다.

(int)(Math.random \* 배열명.length)

규칙을 가지는 경우

{int)(Math.random \* 갯수 )+ 최소값

음수가 있어도 동일한 수식을 활용할 수 있다.

예) -3, -2, -1, 0, 1, 2, 3 이라면, {int)(Math.random \* 7)+ -3

배열 내 동일한 것이 몇 개인지 확인하는 방법에 활용하기

1. 동일한 배열을 한 개 더 생성한다.

2. 첫 번째 배열을 두 번째 배열의 인덱스로 활용한다. 예) 배열명2 [배열명1[인덱스]]

**String 배열**

문자열을 담을 수 있는 배열

참조형이므로 데이터를 직접 저장하지 않고 해당 데이터의 주소값을 저장한다.

참조형을 활용한 for 문은 아래와 같이 사용한다.

|  |
| --- |
| For (*변수정의* : *데이터*)  {  } |

\*변수정의 🡺 변수타입 변수명 : 데이터

-------------------------------------------------------------160928----------------------------------------------------------

**다차원**

**- 2차원**

행과 열의 형태로 표기한다. .

int**[ ] [ ]** 배열명 🡸 배열선언

배열명 = new int **[행크기] [열크기]** 🡸 생성: 생성할 때 변수의 크기를 지정한다.

배열명 [인덱스번호][ 인덱스번호] 🡸 사용 (초기화): 인덱스 번호는 0, 0 부터 시작한다.

초기화는 행 단위로 처리한다.

메모리는 행과 열을 별도의 위치에 저장한다.

배열명에는 첫번째 행의 주소값을, 각 행에는 첫번째 열의 주소값을 갖는다.

중첩 for문을 활용할 수 있다.

For ( int i=0; i<score.length; i++)

{

for (int j=0; j<score[i].length; j++)

}

아래와 같이 초기화를 포함하여 단축형으로 할 수 있음.

int[ ][ ] score = { {1,2,3},

{4,5,6},

{7,8,9},

{10,11,12},

{13,14,15}

};

**정방형배열과 비정방형배열 (가변배열)**

차이점은 열의 크기

정방형배열 - 열의 크기가 동일

예) int[ ] [ ] score = new int [5][3]

비정방배열 - 열의 크기가 다름

예) int[ ] [ ] score = new int [5][ ]

score [ 0 ] = new int [ 3 ]

score [ 1 ] = new int [ 1 ]

score [ 2 ] = new int [ 2 ]

score [ 3 ] = new int [ 4 ]

score [ 4 ] = new int [ 3 ]

각 열을 다르게 지정할 수 있으며, 실제로 메모리에서는 지정된 수 만큼만 지정된다.

1. 향상된 for문을 사용한다.

|  |  |  |
| --- | --- | --- |
| 10 | 11 | 12 |
| 20 | 21 |  |
| 30 | 31 | 32 |
| 40 | 41 |  |

**- 3차원 (다루지 않음)**

-------------------------------------------------------------160929----------------------------------------------------------

**정렬**

<선택정렬: 오름차순> : 비교구문만 변경하면 내림차순으로 변경할 수 있다.

|  |
| --- |
| int [ ] a = {69, 10, 30, 2, 16, 8, 31, 22 };  for (int i = 0 ; i < a.length; i++)  {  for (int j = i+1 ; j < a.length; j++)  {  if(a[i]>a[j])  {  int tmp = a [ i ];  a [ i ] = a [ j ];  a [ j ] = tmp;  }  }  } |

**문제1.**

1. 크기가 10인 배열 생성 : int [] score = new int[10];

2. 난수를 이용하여 1~100사이에 값을 초기화한다.

3. 3번째로 큰 값을 출력

**문제2.**

1. 23 45 12 34 65 23 89 61 26 11

2. 낮은 수가 1등이고 높은 숫자가 꼴등. 공동일 경우 다음 수를 비운다.

23(3) 45(7) 12(2) 34(6) 65(9) 23(3) 89(10) 61(8) 26(5) 11(1)

3. 처음 순서는 바뀌지 않는다.

2차원 배열의 사칙연산은 각각의 값의 같은 위치끼리 연산한다. 나누기는 되지 않는다.

문제 3.



위 모양으로 숫자 출력하기

0은 출력하지 않는다.

참조: x,y 값





**객체지향프로그래밍 I**

먼저 실행이 되도록 만든 후 최적화 작업을 진행한다.

프로그램에서 클래스에서 메인 선언은 특별한 경우가 아닌 경우 한 개만 사용한다.

이외의 클래스는 속성이나 기능을 가진다.

클래스 🡺 설계도

인스턴스 화 🡺 설계도로 제품으로 만드는 과정

객체 또는 인스턴스 🡺 완성된 제품

-------------------------------------------------------------161004----------------------------------------------------------

클래스를 여러 개로 작업하는 경우, 클래스 별로 각각의 작업파일로 작성하거나 한 개의 작업파일에 다 같이 작성할 수 있지만, 파일명은 반드시 여러 개의 클래스 중 main함수가 포함된 클래스의 이름으로 작성해야만 한다.

객체를 생성할 때는

1. 정의

2. 생성

3. 사용

의 3가지 단계를 거치게 된다.

Main이 포함된 클래스에는 파일이름에 특정 단어를 부치면 구분하기 쉬움. 현 교재에는 Test를 사용

정의된 클래스를 이용할 때는 참조형 변수를 선언한다. 이 때 값은 해당 클래스에 할당된 주소값을 가지고 온다.

형식1:

인스턴스의클래스명 참조형변수 = new 인스턴스의클래스명( ) ;

형식2;

인스턴스의클래스명 참조형변수;

참조형 변수 = new 인스턴스의클래스명( ) ;

인스턴스 내부에 변수나 메소드 등 해당 맴버를 참조하기 위해서는 소속을 명확히 한다.

참조형변수**.**멤버변수 = 값;

\*클래스에 정의된 메서드도 배열과 마찬가지로 초기화 하지 않으면 기본값을 가지게 된다.

같은 클래스로 생성한 참조변수a를 참조변수b와 동일하다고 입력하게 되면,

변수를 참조하는 것과는 다르게 해당 주소 값을 복사하기 때문에 기존의 값은 사용할 수 없게 된다. 이 때 참조변수가 없는 인스턴스들은 garbage collector에 의하여 자동으로 메모리에서 제거한다. \*수동제어도 가능하다.

객체도 배열형태로 생성이 가능하다.

인스턴스의클래스명[ ] 참조형변수 = new 인스턴스의클래스명[ 크기 ] ;

생성한 객체배열의 초기화를 기본형과 동일한 방법으로 하지만 저장되는 값은 주소값으로 저장이 된다.

인스턴스의클래스명[ ] 참조형변수 = {new 인스턴스의클래스명( ), new 인스턴스의클래스명( ), ….};

크기가 클 경우 for문을 활용할 수 있다.

사용할 때도 배열과 동일하게 [0], [1], [2] 순으로 사용한다.

-------------------------------------------------------------161005----------------------------------------------------------

인스턴스에 생성한 변수를 직접 참조하거나 setter/getter라는 메서드로 간접 참조할 수 있다.

getter/setter로 사용할 변수에 대한 제약을 넣을 수 있다. 일반적으로 조건문을 많이 사용한다.

변수앞에는 private을 붙인다.

**영역에 따른 변수의 종류**

변수

1. 멤버변수 인스턴스 변수 , 클래스 변수: 클래스 영역에 선언하며 자동초기화 됨

2. 지역변수 : 또다른 블록에 선언 (함수 안쪽에 선언), 명시적으로 초기화함

형식:

class

{

타입 변수명; 🡸 인스턴스 변수

제어자 타입 변수명; 🡸 클래스 변수

Void func ( )

{

타입 변수; 🡸 지역 변수

변수 = 초기값 🡸 지역 변수

}

}

클래스 변수는 제어자 static을 붙여서 선언하며 최초 선언 시 1개의 저장공간을 생성하여 다른 변수와공유가 된다. 굳이 생성단계를 거치지 않고 사용을 할 수 있도록 허용되어 있다.

**클래스명 변수** 형태로 사용이 가능하다.

반복되는 문장들을 묶어서 하나의 메서드로 작성하면 중복코드를 줄일 수 있다.

기본구조

리턴타입 메서드명 (매게변수….)

{

실행문;

Return문; 🡸 매서드를 종료할 때 사용하며 종료하면서 값을 특정할 경우 변수, 상수, 수식등을 추가

}

리턴타입에는 **기본형, 참조형, void**를 사용할 수 있는데 void는 리턴 받을 값이 없을 때 사용한다.

void 매서드 내에서 return은 종료의 의미를 갖고 별도의 값을 리턴하지 않는다.

일반 매서드에서 조건문을 입력할 때는 반드시 else까지 처리해야 한다.

자바에서 스택구조는 실행되는 순서대로 메모리에 할당이 되며 실행된 후 역순으로 해제한다.

**First In Last Out**

-------------------------------------------------------------161006----------------------------------------------------------

기본형 매개변수를 이용할 때는 변수의 값을 이용하는 데 이를 call by value라 한다.

이 때는 데이터를 복사하여 넘겨주기 때문에 원본 값이 변경되지 않는다.

참조형 매개변수를 이용할 때는 주소값 형태로 넘어오는 것을 call by reference라고 한다.

이 때는 주소값을 복사하는 것이기 때문에 원본 값도 변경이 된다.

반환타입이 참조형인 경우 주소값을 복사한다.

문제1

1. class를 2개 만든다. Cals, Calstest

2. 연산은 Cals에서만 한다.

3. class Cals 에 2개의 함수를 만든다.

a. 연산을 위한 함수 (사칙연산 수행)

- 반환타입: int

- 매개변수 : 3개

b. 출력함수

- 반환타입: void

- 매개변수: 1개

4. class CalsTest

a. 연산을 위한 값 입력: scanner를 이용해서 3개의 값 입력

b. class cals의 메서드를 이용해 연산 및 결과 출력

-------------------------------------------------------------161007----------------------------------------------------------

**재귀호출**

메서드가 매서드 자신을 다시 호출하는 방식. 주로 call by value로 사용하며 무한반복을 하는 것이 특징이다. 하여 종료시점을 필수로 입력해야 한다.

사용할 때 유의사항

1. 재귀변수에 사용할 매개변수를 설정

2. 재귀함수의 반환 값 설정

3. 종료조건을 설정 (주로 1에 설정한 매개변수를 사용한다)

클래스 매서드 : **static** void 매서드명() { }

인스턴스 매서드: void 매서드명() { }

클래스 매서드의 경우 생성 단계를 생략하고 소속을 명시하고 사용할 수 있다.

또한 동일한 클래스 내에서 클래스매서드를 사용하는 경우 소속명시 없이 사용도 가능하다. 클래스매서드의 경우 먼저 메모리에 등록이 되기 때문에 인스턴스 매서드와 인스턴스 변수는 사용할 수가 없다.

메서드의 이름이 동일하지만 ()안의 매개변수의 타입 또는 수량이 다르게 지정하여 오버로딩을 할 수 있다. 단 리턴타입은 오버로딩의 조건이 아님.

타입을 다르게 입력하더라도 캐스팅이 되어 조건의 만족하는 경우가 발생하기 때문에 오버로딩을 하기 위해서는 매개변수의 값을 입력할 때 타입을 맞췄을 경우에만 오버로딩으로 인정한다.

-------------------------------------------------------------161010----------------------------------------------------------

void 매서드명 (매개변수1, 매개변수2…..)

매개변수는 인자라고도 하는데 이 인자가 확정되어 있지 않고 변형이 되는 것을 가변인자라고 한다. JDK 1.5버전 이후부터 적용되었다. 함수 중 printf가 대표적인 예이다.

배열과 비슷한 방식으로 진행된다.

void 매서드명 (String format, 배열…..)

특정타입을 가지는 가변인수를 사용해도 된다. 배열형태로 사용하게 되기 때문에 퍼포먼스가 떨어지기 때문에 반드시 필요한 부분에만 사용하는 것을 권장한다.

delim; 구분자를 통칭한다.

… args ; 배열을 통칭한다.

Class

{

속성: 인스턴스 변수 / 클래스 변수 (static이 붙는다.)

메서드

생성자 : 멤버변수 중 인스턴스 변수를 초기화하는 역할을 하며 클래스의 이름과 동일하게 정의해야 한다. 또한 오버로딩이 가능하다.

초기화블럭

}

모든 클래스는 생성자를 1개 이상 포함하고 있다.

기본생성자를 포함한다.

동일한 클래스 안에서 생성자들 서로를 호출할 때는 this( )를 사용한다. 단 생성자 안에서 가장 첫 번째 줄에서만 사용할 수 있다.

**변수의 초기화**

멤버변수와 배열의 초기화는 선택적으로 진행하지만 (\*기본값이 자동으로 초기화된다) 지역변수는 반드시 초기화해야 한다.

명시적, 초기화, 생성자를 사용한다.

참조형의 기본값은 null이다.

초기화블럭 🡺 { } 또는 static { } 으로 표시한다.

-------------------------------------------------------------161011----------------------------------------------------------

결과

입력: (배열[3]에 입력한다.)

콜라 500 10 20

사이다 1000 20 30

맥주 2000 30 50

출력:

콜라 500 10 20

사이다 1000 20 30

맥주 2000 30 50

-------------------------------------------------------------161012----------------------------------------------------------

**객체지향프로그래밍 II**

개념:

- 상속

- 다형성

- 캡슐화 (은닉)

궁극적인 목적은?

코드의 최소화 및 재사용

**상속**

B클래스가 A클래스를 소속된 멤버 변수와 멤버 메서드를 상속하는 경우

형식:

class A

class B extends A

부모클래스는 여러 개의 자식 클래스를 가질 수 있고, 동일한 부모클래스를 가지더라도 자식클래스끼리는 어떠한 관계도 형성되지 않는다.

자바에서는 한 개의 자식은 두 개 이상의 부모클래스를 가질 수 없다.

**포함**

기존에 클래스 간에 상대 클래스를 선언한 후 멤버를 사용하게 될 경우 해당 클래스 간의 관계를 **포함**이라고 한다.

상속과의 차이점은 원하는 부분만 선언하여 사용이 가능하고 다중상속을 구현할 수 있다.

- 연관: 선택사항으로 사용되는 요소

- 집합: 필수 적으로 있어야 하는 요소

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 상속 | 포함: 연관 (단방향) | 의존 | 포함: 집합 | 구현 |
|  |  |  |  |  |

상속의 상속을 거듭하면 무거워지는 단점이 발생할 수 있다.

상속을 중복으로 사용하지 않고 한 클래스에서는 상속을 하고 다른 클래스에서 일부 필요한 부분만을 사용하는 포함과 상속을 같이 사용할 수가 있다.

자바에서 제공하는 최상위 클래스:

**Object** : 모든 클래스 (기존의 클래스 뿐 아니라 직접 만든 클래스까지 모두 포함)가 자동 상속한다.

-------------------------------------------------------------161013----------------------------------------------------------

**오버라이딩**

메서드에 대하여 적용되는 개념

조건:

1. 두 개 이상의 클래스가 **상속관계**에 있어야 한다.

2. 자식클래스에서 조상의 동일한 매서드를 재정의한다 (선언부는 같아야 한다, 즉)

- 매서드 이름은 반드시 같아야 한다.

- 매개변수가 같아야 한다.

- 반환타입이 같아야 한다.

3. 예외처리가 고려되어야 한다. (자세한 내용은 이후 예외처리에서 다룰 예정)

4. static 여부를 변경할 수 없다.

자식클래스는 상속을 통하여 두 개 이상의 동일한 맴버를 보유할 수 있게 된다.

**super**

this 와 동일한 방식으로 처리된다.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| super | 부모클래스명을 대체 | 자손클래스에서 사용 |
| this | 자신의 클래스명을 대체 | 멤버와 지역변수명이 동일할 때 구분을 위하여 사용 |

**Super()**

|  |  |  |
| --- | --- | --- |
| super( ) | 부모의 생성자 호출 | - 생성자 내에서만 사용  - 반드시 첫째줄에 코딩 |
| this( ) | 자신의 생성자간 호출 |

\*부모와 자식이 모두 기본생성자일 경우 컴파일로가 자동으로 super()를 처리하여 기본생성자를 모두 호출해 준다. 부모에 의하여 생성자가 지정되어 있을 경우에는 자식쪽에서 반드시 명시해줘야 한다.

**Package**

의미나 기능이 비슷한 클래스끼리 묶어놓는 것. 폴더를 생성하는 것과 같다.

최상단에 선언을 한다. 패키지명은 저장되는 경로명을 뜻하며 하위 디렉토리는 . 으로 구분하여 명시한다.

**형식: package** 패키지명;

패키지 경로는 묵시적으로 소문자를 사용한다.

**Import**

동일한 패키지 안에서 사용할 경우 클래스 사용에 제약이 없지만 패키지경로가 다른 클래스를 사용할 경우 사용한다.

**형식: Import** 패키지명**.**클래스명

\* 마크는 이하 class전체를 의미한다.

-------------------------------------------------------------161014----------------------------------------------------------

**제어자**

기본 형식에 부가적인 성질을 부여한다. 변수, 메서드, 생산자, 초기화블럭, 클래스에 붙일 수 있다.

- 접근제어자 : public, protected, default, private (서로 같이 사용이 불가능하다. )

- 기타제어자 : static, final, abstract……

public은 모든 패키지 안에서 공유 한다.

protected는 같은 패키지와 하위 패키지 까지만 공유할 수 있다.

default는 같은 패키지에서만 공유할 수 있다.

private 동일 클라스 내에서 맴버간의 참조만 가능하다

final이 붙은 클래스는 다른 클래스의 조상클래스가 되어 다른 클래스에 상속이 불가능하다.

변수에 사용하면 해당 변수가 상수로 바뀌면서 변경이 불가능하게 된다. 상수는 생성자를 이용하여 최초 1회만 변경할 수 있다.

abstract

추상매서드는 완성되지 않은 구현부가 없는 메서드로 추상 클래스가 있다면 1개 이상의 추상 메서드가 존재해야 한다.

private으로 지정된 맴버변수를 변경하기 위해서는 getter/setter를 사용한다.

getter/setter 형식

|  |  |
| --- | --- |
| private 타입 변수  pubic void set변수명(타입 변수명)  {조건값 return;  this.변수명 = 변수명} | pubic int get변수명( )  {  return 변수명;  } |
| 초기화 할 때 사용 | 변수를 가지고 올 때 사용 |

\*toString은 Object클래스에 있는 매서드로 사용할 경우 오버라이딩 된다.

생성자의 접근제어자는 보통 클래스의 접근제어자와 동일하게 작성한다. 하지만 예외적으로 private을생성자에 부여하는 경우도 있다. 예를 들면 싱글톤패턴

의미와 기능이 상충되는 조합은 함께 사용이 불가능하다.

**다형성**

오버로딩/ 오버라이딩 (메소드)

참조변수 (상속관계)

🡺 상위 타입과 하위타입 간의 참조를 위하여 사용한다.

🡺 하위 타입에서 상위타입을 참조할 때는 캐스팅을 해야 한다.

🡺 참조할 수 있는 객체의 개수가 달라진다.

자손클래스명 변수 = **(자손클래스명)**new 부모클래스명( )

\*이미 생성된 부모클래스는 자손클래스로 형변환이 불가능하다.

-------------------------------------------------------------161017----------------------------------------------------------

**instanceof연산자**

객체를 비교할 때 사용하는 연산자

형식: 참조변수 instanceof 참조형데이터타입(클래스명)

상속관계에 있는 클래스에서 매서드가 오버라이딩이 되는 경우 매개변수와는 다르게 부모나 자식으로 호출하는 것과는 상관없이 무조건 오버라이딩(자식에 있는 매서드)를 우선으로 처리한다

상속관계에서 중복되는 변수나 매서드가 없을 경우 부모나 자식 어느 쪽에서 호출하던지 같은 결과값을 얻을 수 있다.

상속관계에서 부모의 매서드를 참조하고 싶을 경우 자식에서 super를 사용한다.

콜백함수(자동호출)

**추상클래스**

추상클래스를 상속하는 경우 상속받은 자손클라스는 조상클래스 내부의 추상매서드를 모두 구현해 주어야 한다. 한 개라도 구현하지 않는 경우에는 자손도 역시 추상클래스로 지정해야 한다.

특징: 생성불가 / 상속해야 함 / 자식은 오버라이딩 / 일반적인 맴버 정의가능

반대로 말하면 부모에 추상매서드가 있다면 자손에서는 반드시 오버라이딩이 되어야 하므로 개발자가 자손클래스에서 해당 부분을 반드시 구현하라는 의미도 담겨 있다.

**interface**

추상클래스와 비슷한 특징이 있음

생성불가 / 상속해야 한다 / 자식은 오버라이딩 / 일반적인 맴버 정의 불가 / 제어자 생략가능

형식:

interface 인터페이스명

{

상수 또는 추상메서드 만 가능

}

제약사항: 일반적인 맴버 정의가 되지 않기 때문에 정의하지 않고 생략해도 같은 의미를 가진다.

상수: public static final

추상메서드: public abstract

-------------------------------------------------------------161018----------------------------------------------------------

인터페이스는 인터페이스간 다중 상속이 가능하다.

형식: interface C extends A, B

하지만 생성을 하기위해서는 class와 상속이 되어야만 한다.

단 이때는 상속키워드가 extends에서 implements로 변경된다.

형식: class B implements A, B, C

\*interface 를 상속받을 때는 다형성도 함께 적용된다.

\*interface는 다중으로 상속을 할 수 있다.

\*class에는 interface에 적용된 추상메서드를 오버라이딩해야 한다. (추상클래스를 만들면 하지 않아도 된다)

\*class와 interface를 동시에 적용할 수도 있다.

형식: class B extends A implements A

Interface를 class에 오버라이딩 할 때 implement에서 생략하고 사용한 제어자도 함께 사용을 해야 한다.

|  |  |
| --- | --- |
| 인터페이스 | 클래스 |
|  |  |

인터페이스를 사용하는 가장 큰 이유는 클래스가 다른 클래스를 사용할 때 대리인으로써의 역할을 부여 하기 위해서이다.

메소드에서 인터페이스가 반환타입이 되는 경우 해당 인터페이스를 상속하는 클래스를 반환하라는 의미가 된다.

인터페이스는 아무런 내용 없이 타입을 묶어 하나의 그룹으로 만들어 주는 역할을 한다.

클래스 내 그룹으로 묶었지만 하위 자손 클래스에서 공통적인 역할을 수행하는 경우 굳이 인터페이스를 사용할 필요없이 클래스 상속을 이용할 수 있지만, 인터페이스의 adapter 패턴을 활용하기 위해서 사용할 수 있다.

-------------------------------------------------------------161019----------------------------------------------------------

adapter 패턴이란?

인터페이스와 다수의 클래스가 상속되어 있는 중에 모든 상속받은 클래스들이 동일한 기능을 사용하는 경우에는 각 클래스 별로 재정의하는 것이 아니라 해당 동일한 기능을 구현하는 클래스를 하나 만들어 클래스들의 오버라이딩에 필요한 내용을 호출하여 사용하는 것

|  |  |
| --- | --- |
| 코딩 | 메모 인터페이스나 추상메소드는 기울임꼴로 적는다. |
| class Something  {  private int privateField  protected int protectField  public int publicField  int packageField  private String privateMethod(int a)  protected void protected(int a)  public void publicMethod(int a)  static void packageMethod(int a)  } | Something  - privateField: int  # protectField: int  + publicField: int  ~ packageField: int  - privateMethod (a: int): String  # protectedMethod(a: int): void  + publicMethod(a: int): void  ~ packageMethod(a: int): void |

|  |  |
| --- | --- |
| 매개변수, 맴버변수 | 생성 |
| use | create |

|  |  |
| --- | --- |
| 맴버를 사용하는 수를 관계에 포함한다. | |
| 1개만 사용하는 경우 | 다수의 배열로 사용하는 경우 |
| 1 | 0..\* |

Interface를 사용할 때 default라는 지시어로 구현해서 넘겨 줄 수 있다. 단, 기존 이름이 같을 경우 문제가 발생할 수 있다.

**내부클래스**

클래스 안에 클래스를 다시 정의하는 형식

class A

{

class B{ } 🡸 instance inner class

static class C { } 🡸 static inner class

void test { } 🡸 local inner class

{

class D { }

}

익명클래스 (1회용클래스)

}

-------------------------------------------------------------161020----------------------------------------------------------

**예외처리**

에러의 경우 프로그래머가 비정상종료를 막을 방법이 없는 것을 말하지만 예외는 예상 가능한 상황에 대하여 적절한 코드를 작성하여 비정상종료를 막는 방법이다.

예외처리 구문

Try { 문제소지가 있는 코드1; 문제소지가 있는 코드2; 문제소지가 있는 코드3; …. }

catch { 해결방법1 } catch { 해결방법2 } catch { 해결방법3 }…

에러타입

ArithmeticException 매개변수 🡨 산술과정에서 발생하는 오류사항

Try catch문에서 try 내부에 실행문 중에서 에러가 발생되면 try내부에 이하실행문은 실행하지 않는다.

Exception 으로 하면 모든 종류의 예외처리가 가능하다. 단 에러에 대한 확인 후 해결하는 것이 아니기때문에 정확한 처리라고 보기는 어렵다.

또한 Exception 타입은 가장 마지막에 작성해야 한다.

예외가 발생했을 때 요인들에 대한 메시지를 발생한다.

printStackTrace()

getMessage()

강제로 예외를 발생시킬 때는 throw구문을 사용한다.

try { throw new 예외처리명 } catch (예외처리명 변수) { 해결방법 }

메서드에 예외선언하기

메서드에 문제가 있음을 미리 선언하는 방법으로 호출하는 쪽에서 해결을 하고 쓰라고 하는 방법

모든 예외를 할 때는 Exception을 사용한다.

void test( ) throws 예외1, 예외2, 예외3………

{

문제 코드

}

finally 구문

try catch문을 실행할 때 필요에 따라서 사용. 예외발생여부와 상관없이 무조건 실행되며 항상 맨 마지막에 사용되어야 한다.

사용자정의 예외 만들기

class 예외처리명 extends Exception (또는 Runtime) 🡸 에러 메시지의 성질을 결정한다.

{

예외처리명 ( )

{

super(메시지); 🡸 출력할 에러 메시지를 작성한다.

}

}

시스템에서 작성한 예외만들기를 실행하도록 try catch문으로 정의한다.

try { throw new 예외처리명 ( ) } catch (예외처리명 변수) { 처리방법 }

-------------------------------------------------------------161021----------------------------------------------------------

문제

a~z 🡺 97 ~ 122

A~Z 🡺 65 ~ 90

알파벳이 아닌 다른 문자 일 때는 예외가 발생되도록 처리

**유용한 패키지**

Object : 모든 클래스의 최상위 조상 클래스

autoboxing : 기본형과 참조형간에 형변환을 하지 않아도 컴파일을 할 경우 자동으로 삽입해 준다.

Random 클래스 : math.random()과 동일한 기능이며 각 type별 기본형을 사용이 가능하다.

-------------------------------------------------------------161024----------------------------------------------------------

trim() : 단어의 앞 뒤 공백을 제거하는 함수

split() : 공백을 구분자로 자른다. (“ “) 🡺 공백 1칸을 기준으로 한다. (“ +“) 🡺 공백이 “1개보다 많다”를 의미한다.

hasnextInt() : 문서의 내용의 모든 줄을 스캔하며 줄단위로 스캔하여 true 값을 반환하다가 마지막줄이후에는 false 값을 반환한다. Int로 처리되어 있기 때문에 int형을 스캔할 수 있다.

Scanner (new file(“파일명.확장자”): 기입한 파일에 내용을 불러온다.

useDelimiter(“구분자”) 구분자로 구분되어 있는 값을 분리해서 사용한다.

-------------------------------------------------------------161025----------------------------------------------------------

예제

-------------------------------------------------------------161026----------------------------------------------------------

**날짜와 시간**

주로 Calendar를 사용하는 것을 권고한다.

Calendar 와 Date 클래스

1. 기본날짜 (현재 시간)

2. 임의의 날짜

API에서 기본제공하는 상수를 확인할 수 있다.

MONTH: 0 ~ 11 🡺 0이 1월 11이 12월

DAY\_OF\_WEEK: 1 ~ 7 🡺 1이 일요일 7이 토요일

getTimeInMillis() 🡺 1/1000초를 제공한다.

add 메서드

add (기준, 값) 음수의 경우 빼주고 양수의 경우 더해준다.

roll (기준, 값) 앞의 값에는 영향을 미치지 않고 해당 자리수만 연산한다.

-------------------------------------------------------------161027----------------------------------------------------------

과제제출

기획 PPT 자료와 코드제작

기한: 다음 주 금요일(11/4)까지 제출

-------------------------------------------------------------161028----------------------------------------------------------

사용하는 객체의 API를 먼저 확인해서 사용할 수 있는 부분을 먼저 체크한다.

**\*536pg 코드 분석해 보기**

**Format**

- DecimalFormat

- SimpleDateFormat

- ChoiceFormat

- MessageFormat

**java.time 패키지**

날짜와 시간을 표현하는 클래스, calendar와 다르게 날짜와 시간이 분리되어 있다.

**Collection Framework**

데이터를 효율적으로 관리하기 위한 자료구조로 변수를 이용하여 데이터를 저장하는 방식

- List : 순서대로 저장 / 중복허용

- Set : 순서없이 저장 / 중복허용안함

- Map : 키와 값이 한 쌍으로 저장 / 순서없이 저장 / 키는 중복허용하지 않지만 값은 중복허용.

Collection

많이 사용하는 매서드 boolean add, Iterator iterator(), int size()

List

배열의 성질을 띈다.

For문과 연결지어서 사용한다.

많이 사용하는 매서드 Object set (int index, Object element), void sort (Comparator c)

Map

내부적으로 인터페이스 구조를 가진다.

Object getKey() , Object getValue를 많이 사용한다.

공통으로 나타나는 특징:

주로 주소값을 받고 데이터 타입은 Object를 사용한다. 따라서 저장할때는 자동형변환이 되지만 읽어올 때는 캐스팅을 해 주어야 자기자신의 객체를 사용할 수 있다.

Collection은 크기가 자동으로 늘어난다.

읽고 쓰기

List와Set add() / get()

Map put() / get()

Arraylist에서 값을 넣을 때는 add를 사용한다. 이 경우 기존에 있는 자료가 뒤쪽으로 밀린디.

set을 이용하여 값을 덮어 씌울 수 있다.

문제 ArrayList

- 1~10까지 삽입

- 읽어서 합계 구하기

LinkedList

주소값과 값 2가지를 저장한다. 주소값만 알면 중간에 데이터를 넣을 수 있기 때문에 처리속도가 빠르다.

DoublyLinkedList

LinkedList의 단점을 보완한다. 이전데이터의 주소값도 저장한다.

DoublyCircularLinkedList

양끝을 연결해 준다.

데이터 사용방법에 따라서 골라서 사용한다.

문제 LinkedList

- 1~10까지 삽입

- 읽어서 짝수, 홀수 합계 구하기

**Stack 클래스**

F.I.L.O 구조로 구성된 클래스 (처음 들어온 데이터부터 마지막으로 사용)

empty() 🡸 데이터가 비었으면 true, 차 있으면 false를 반환한다.

**Queue 인터페이스**

F.I.F.O 구조로 구정된 클래스 (처음 들어온 데이터부터 순차적으로 사용)

isEmpty()🡸 데이터가 비었으면 true, 차 있으면 false를 반환한다.

생성할 때 다른 인터페이스를 사용한다. 주로 LinkedList를 사용한다.

-------------------------------------------------------------161031----------------------------------------------------------

빈 스탯에서 pop() 구문을 이용하여 데이터를 로딩하게 하면 오류메시지가 발생하게 된다.

**Iterator, Listlterator, Enumeration**

hasnext() 🡺 읽어 올 요소가 남아 있는지 확인한다.

Object next() 🡺 호출할 때 사용한다.

void remove() 🡺 next로 호출한 요소를 삭제한다. next 없이 단독으로 사용할 수 없다.

Iterator 의 경우 상속되므로 arrayList를 바로 사용할 수 있다.

단 map collection을 사용하는 경우 전환하는 메서드를 사용해야 한다.

entlyset().iterator() 🡺 key, value

keyset().iterator() 🡺 key

values().collection 🡺 value

Enumeration 에서는 Iterator와 동일하지만 hasMoreElements()와 nextElement()를 사용한다.

배열 복사의 사용법

Arrays.copyOf()

Arrays.copyOfRange()

기존 배열보다 크게 만들 경우 나머지는 기본값으로 채워진다.

fill() 을 사용하면 지정한 값으로 모든 값으로 채울 수 있다.

asList (Object a)배열를 list로 담아서 변환한다.

-------------------------------------------------------------161101----------------------------------------------------------

**Comparator와 Comparable**

인터페이스: 컬렉션 데이터의 정렬

Arrays.sort () 🡺 배열을 정렬할 때 사용

Collections.sort(List) 🡺 콜렉션을 정렬할 때 사용

Comparator 다양한 기준을 직접정의하여 정렬을 한다.

Comparable 기본 기준을 오버라이딩하여 정렬을 한다.

this.value의 위치에 따라서 정렬기준이 변경된다.

오름차순: this.value – 외부값

내림차순: 외부값 – this.value

A.compareTo(B) 🡺 오름차순 정렬하게 된다.

B.compareTo(A) 🡺 내림차순 정렬하게 된다. (A.compareTo(B) \* -1 로 처리할 수 있다. )

**HashSet**

중복과 저장순서를 유지를 지원하지 않는다.

문자열 값의 중복을 확인할 때 equals ()와 hashCode() 이 2개의 매서드를 반드시 오버라이딩 해야 한다.

-------------------------------------------------------------161102----------------------------------------------------------

Treeset

가장 왼쪽 하단은 최소값을, 가장 우측 하단은 최대값을 가진다.

데이터를 불러올 때는 최소값부터 시작하여 자동으로 오름차순 정렬이 된다.

set.headSet 🡺 기준값 미만을 구할 때 (기준값 이전까지만 조회하며 기준값은 제외된다.)

set.tailSet 🡺 기준값 이상을 구할 때 (기준값 이상 값을 조회하며 기준값은 포함된다.)

map.get(키값) 🡺 value 값을 제공해준다.

Map 방식을 Iterator한 방식으로 읽어 올때는

Set set = map.entryset();

Iterator it = set.iterator();

While(it.hasNext())

{

Map.Entry e = (Map.Entry)it.next();

**e.getkey() 🡨 사용할 수 있음**

**e.getValue() 🡨 사용할 수 있음**

}

-------------------------------------------------------------161103----------------------------------------------------------

**Properties**

setProperty / getProperty : 저장/로딩 하는 메서드

파일에 저장할 때는 A = B 형태로 저장되어 있어야 한다.

#은 주석처리된다.

파일 입력할 때 띄어쓰기는 하지 않고 붙여서 사용한다.

split는 scanner의 [useDelimiter](#useDelimiter) 와 동일한 역할을 한다.

-------------------------------------------------------------161104----------------------------------------------------------

과제

-------------------------------------------------------------161107----------------------------------------------------------

**지네릭스**

**장점:**

타입의 안정성을 제공한다.

타입체크와 형변환을 생략할 수 있으므로 코드가 간결해 진다.

class 클래스명 <T> 🡸 type을 정해 놓지 않은 경우 임시로 지정해 놓는다.

클래스명 🡨 원시타입이라고 지칭

<T> 🡨 타입변수 또는 타입 매개변수로 지칭

지네릭 클래스는 T의 클래스명 또는 T 클래스명 이라고 읽는다.

Type에 <String> 같이 특정 타입을 지정하는 경우 지정한 타입만을 사용한다는 의미임.

상속관계에 있을 경우 원시타입은 다형성 지원이 된다.

**제한된 지네릭 클래스**

타입을 제한 할 때 자손 관계에 있는 클래스만 사용하도록 제한할 수 있다.

**와일드 카드**

Static한 메서드 내에서 사용하는 경우 T를 사용할 수 없고 명확하게 작성해야 한다.

이를 피하기 위하여 와일드카드를 사용한다.

\*제네릭스 타입이 다른 것은 오버로딩이 안된다.

와일드카드:

<? extends T> 🡺 제네릭스 타입의 T와 자손까지는 인정한다는 의미

<? super T> 🡺 제네릭스 타입의 T와 조상까지는 인정한다는 의미

<? > 🡺 제한 없음 모든 제네릭스 타입이 가능 <? extends Object>를 의미

Collections.sort() 는 API에 아래와 같이 정의 되어 있다.

public static <T> void sort(List<T> list, Comparator<? super T> c)

하여 API에 정의된 내용에 따라서 조상까지 인정할 수 있다.

-------------------------------------------------------------161108----------------------------------------------------------

**지네릭 타입의 형변환**

다형성 적용을 위해서는 <? Extends Object> 형태로 기입하면 상속관계에 의한 다형성을 적용할 수 있다.

Object로 지정하여 사용하여도 무조건 형변환이 이루어 지지 않는다.

와일드카드로 사용하려는 타입이 인정하는 범위에 속하더라도 형변환이 불가능한 관계에서는 에러가 발생한다. 예를 들어 자손 타입으로 지정할 때 조상타입을 기준으로 와일드카드를 적용하는 경우, 와일드 카드에서는 모든 자속타입을 사용할 수 있지만, 자손끼리는 형 변환이 되지 않기 때문에 에러가 발생한다.

-------------------------------------------------------------161109----------------------------------------------------------

**쓰레드**

프로세스(Process) : 실행중인 프로그램

쓰레드(Thread): 프로세스 내에서 기능을 처리하는 것

Single Thread: 1 🡪 2 🡪 3 🡪 4 🡪 5 와 같이 순서대로 처리가 된다. 1이 끝나기 전에는 2를 진행할 수 없다.

Multi Thread: 실제로는 순서대로 처리하지만 각각의 일을 아주 짧은 시간씩 번갈아 가면서 처리하기 때문에 각각의 일을 동시에 처리할 수 있다. 프로그램을 사용하는 하드웨어의 성능에 따라서 동기화/교착상태 등의 문제가 발생할 수도 있다.

Thread구형방법:

1. Thread class를 상속받는다.

2. Runnable interface를 상속받는다.

\* run 메서드를 오버라이딩 해야 한다.

기본 정의는 다르지 않지만, 각 기능을 호출할 때 방법이 다르다. 이는 Runnable은 Thread와 다르게 인터페이스를 사용하여 직접 호출이 불가능하기 때문이다.

Runnable 인터페이스를 상속받은 경우에는 메서드를 사용할 때 Thread클래스의 생성자의 매개변수로 제공하여 Thread에 선언되어 있는 매서드를 사용해야 한다.

Thread 메소드의 실행방식은 run() 매서드를 호출하는 것이 아니라 **start() 매서드를 호출해서 사용해야 한다**.

실행을 요청하는 것은 순서대로 할 수 있지만 실제로 실행순서와 종료순서는 시스템이 결정한다.

Thread는 중복실행이 되지 않는다. 다시 실행하기 위해서는 Thread를 다시 생성한 후에만 가능하다.

run()으로 직접호출은 가능하지만 Single Thread로 호출이 되어 Multi Thread로써의 기능을 상실하게 된다.

main함수를 실행하는 것도 jvm인 시스템이 thread를 이용해서 실행하는 것이다. 이를 main Thread라고 한다. 다른 연관된 Thread들이 모두 종료될 때까지 대기하며 모든 Thread가 하나도 없어져야지만 프로그램이 종료된다.

-------------------------------------------------------------161110----------------------------------------------------------

Thread 간 작업전환을 context switching 이라고 한다.

**쓰레드의 우선순위**

모든 Thread는 실행한 Thread의 우선순위를 상속받는다. 우선순위는 1~10으로 숫자가 높을수록 우선순위가 높다. 일반적으로 main thread가 실행하는데 main thread는 default 값으로 5를 갖는다.

순위 값은 절대값이 아니라 상대값의 의미를 갖는다. 즉 우선순위가 높다, 낮다, 같다 등 비교 용도로 사용한다.

**Demon Thread**

Demon Thread는 일반 Thread가 죽을 때 함께 강제로 죽는다.

setDaemon(true) 로 호출한다.

**Thread 실행제어**

sleep(밀리세컨드) 🡺 thread 자신을 제어한다.

join(밀리세컨드) 🡺 상대 thread를 제어한다.

interrupt()

stop() 🡺 thread를 멈춘다.  
suspend() 🡺 thread를 일시적으로 멈춘다.

resume() 🡺 일시적으로 멈춘 thread를 재시작한다.

yield() 🡺 thread의 실행 순서를 제어한다. 본인의 순서를 다른 Thread에게 양보한다.

Thread 실행

suspend() 🡺 resume()

join() 또는 sleep() 🡺 time-out

join() 또는 sleep() 또는 wait 🡺 interrupt()

wait() 🡺 resume()

sleep()은 호출하는 위치에서의 Thread를 제어한다. 사용자 정의된 Thread 내부에 정의해야지만 원하는 결과를 얻을 수 있다.

interrupt() 와 interrupted()

interrupt()는 작업의 중지를 요청한다.

interrupted() 는 interrupt() 가 호출된 여부를 확인한다.

isInterrupted() 는 상태만 반환하고 interrupted() 상태반환 후 false로 초기화한다.

interrupt()를 join(), sleep(), wait()에 의하여 멈춰진 상태의 Thread를 실행대기상태로 만들 수 있다.

try catch문을 이용하여 자동예외발생 InterruptedException e

try catch문을 이용하여 자동으로 예외가 발생하는 경우 interrupted() 상태 또한 자동으로 false로 초기화 된다.

-------------------------------------------------------------161111----------------------------------------------------------

-------------------------------------------------------------161114----------------------------------------------------------

\*Deprecated란? 전에는 사용했지만 앞으로는 사용하지 않을 것을 권고한다는 의미

Deprecated된 resume 과 suspend는 각각 wait와 notify를 대신 사용하게 되는데 동기화된 메서드에서만 사용이 가능하다.

resume() 🡺 notify(): thread를 실행대기 상태로 만든다.

suspend() 🡺 wait(): thread를 일시정지 상태로 만든다.

\*notifyall(): 호출된 객체에서 block 되어진 전체 thread를 실행대기 상태로 만든다.

\*volatile로 동기화를 할 수 있음

yield(): suspend된 thread를 양보 처리하면 대기상태로 이동되기 때문에 이후의 Thread가 더 빠르게 실행상태로 들어갈 수 있다.

interrupt(): 실행대기 상태로 강제로 만들어 준다. 또한 예외처리도 가능하다. suspend()와 stop()은 interrupt()가 필요하다.

synchronized를 이용하여 메서드의 공통부분을 구현할 때 다른 thread가 간섭하지 못하도록 한다.

-------------------------------------------------------------161115----------------------------------------------------------

ConcurrentModificationException

동기화없이 코드가 구성되어서 여러 Thread를 동시에 실행하는 경우 Thread들 중 하나가 먼저 처리하여 다른 Thread가 그 처리를 미리 인지 하지 못하고 존재하지 않는 것을 처리하게 되는 문제가 발생하는 오류.

wait()와 notify()를 이용하여 synchronized 중인 메서드를 강제로 제어할 수 있다.

여러 개의 Thread를 동시에 사용하는 경우, 확률에 따라서 일정 Thread만 나오지 않는 경우가 발생되게 된다. 이를 기아현상(starvation)이라고 한다. 이 현상을 방지하기 위해서 notifyAll()을 사용한다. 단 이 경우 전체 Thread가 공지되게 되어 다시 전체 Thread가 경쟁을 하게 된다. (race condition)

Lock 과 Condition을 이용하여 동기화 하면 지정한 Thread를 제어할 수 있다.

-------------------------------------------------------------161116----------------------------------------------------------

Lock

Synchronized가 자동으로 진행되는 것과는 다르게 프로그래머가 수동으로 제어하는 방법이다.

종류:

- reentrantLock: 객체타입. 생성자로써 기능한다. void lock(), void unlock(), void isLocked()등의 메소드를 사용한다. lock()과 unlock()의 위치를 수동적으로 기입해야 한다. try-catch-finally 구문을 반드시 활용하면, unlock()을 finally에서 반드시 수행될 수 있다.

Condition

기아상태의 문제점을 해결하기 위해서 제공하는 솔루션. Waiting pool을 구분하여 별도로 대기 하도록 한다. 먼저 호출한 lock에서 newCondition()을 호출하여 사용한다.

**람다식**

함수형 언어로 자바에서는 람다식을 익명함수라고도 한다.

작성방법:

변수명을 사용하지 않고 매개변수의 타입도 생략할 수 있다.

선언부와 몸통{} 사이에 **->** 를 추가

매개변수가 1개 인 경우 ()를 생략한다. 단 타입도 함께 생략한다.

식으로 사용하는 경우 {}를 생략하고 ; 도 함께 생략한다.

{}란에 문장이 return문인 경우 생략불가

**함수형 인터페이스**

new Object() { 메서드; }

선언과 실행이 동시에 구현되어 있는 경우 Object는 빌려온 이름이며 익명클라스라고 한다.

함수형 인터페이스는 오직 하나의 추상메서드만 정의되어야 한다.

함수형 인터페이스의 경우 람다식으로 전환하여 매개변수 값으로 사용할 수 있다.

람다식은 object 타입으로 형변환을 할 수 없다.

-------------------------------------------------------------161117----------------------------------------------------------

외부 변수를 참조하는 람다식: 지역변수에서 처리될 때 상수(final)로 간주된다.

자주 쓰이는 형식의 람다식 메서드가 jave.util.function 에서 정의 되어 있어서 활용할 수 있다.

**자주쓰는 함수형인터페이스**

Java.lang.Runnable 🡺 void run() : 매개변수 없음, 반환값 없음

Supplier<T> 🡺 T get() : 매개변수 없음, 반환값 있음

Consumer<T> 🡺 void accept(T t) : 매개변수 있음, 반환값 없음

Function<T, R> 🡺 R apply(T t) : 일반적인 함수, 하나의 매개변수와 결과 반환

Predicate<T> 🡺 boolean test(T t): 조건식을 표현하는데 사용. 매개변수 하나, 반환타입은 boolean

매개변수의 개수가 2개인 함수 인터페이스는 Bi가 접두사로 붙는다.

BiConsumer<T, U> 🡺 void accept(T t, U u) : 두 개의 매개변수 있음, 반환값 없음

BiPredicate<T, U> 🡺 boolean test(T t, U u) : 조건식 표현. 매개변수 두개, boolean반환

BiFunction<T, U, R> 🡺 R apply(T t, U u) : 2개의 매개변수와 1개의 결과 반환

2개이상은 직접 정의필요

List, set, map등 콜랙션에 함수형 인터페이스가 포함이 되어 있다.

**stream**

향후 시간이 남으면 다시 다룰 예정

**람다식**

**입출력(Input / Output)**

입출력을 위해서는 일방으로 통행되기 때문에 데이터를 주고 받기 위해서는 2개의 통로가 있어야 한다.

한 개의 통로로 양방향을 모두 처리할 수는 없다.

read()와 write()가 기본

기본 read()는 1byte를 읽어온다. 읽어올 대상이 없으면 -1을 반환한다.

read(a,b,c) 로 사용할 수 있다. a위치에 b부터 c 길이만큼 읽는다.

flush()

ByteArrayInputStream / ByteArrayOutputStream

메모리에 입출력하는데 사용되는 스트림.

정확한 내용을 받기 위해서 읽어온 파일의 크기를 체크해서 넘기도록 한다.

--------------------------------------------------161118---------------------------------------------------------

**FileInputStream / FileOutputStream**

해당 구문은 메모리에 상주하므로 close() 하여 불필요한 메모리 소비를 막는다.

기본 경로에 있는 파일을 복사 할 수 있다. (경로를 지정할 수도 있다.)

두 구문을 사용하여 파일복사를 재연할 수 있다.

**바이트기반의 보조스트림**

FilterdInputStream / FilterOutputStream

BufferedInputStream / BufferedOutputStream

- BufferedInputStream (InputStream in, int size) 입력소스를 반드시 지정해 주어야 한다. int size를 지

정하지 않는 경우 8192byte를 기본으로 한다.

- flush: 버퍼에 남아 있는 데이터를 강제로 출력해 주는 역할을 한다.

- close: 사용한 자원을 모두 반환하도록 한다.

DataInputStream / DataOutputStream

DataOutputStream 을 사용한 파일은 DataInputStream 으로 쓰인 순서 그대로 읽어야 한다. 특히 여러개의 자료형을 사용한 경우 순서가 바뀌면 정확한 데이터를 읽어올 수 없다.

**문자기반의 스트림**

FileReader/FileWriter

바이트기반과 다르게 인코딩을 지원한다.

PipedReader / PipedWriter

쓰레드 간에 데이터를 주고 받을 때 사용한다.

StringReader / StringWriter

StringBuffer getBuffer() 🡨 데이터 저장에 버퍼가 이용되기 때문에 이 메서드를 이용하여 해당 버퍼 정보를 반환받을 수 있다.