Java

작성일시	@2023년 2월 20일 오후 6:55
강의 번호	허지훈 강사님
유형	강의
자료	
복습	☐

Java 교육

Java 컴파일 프로그램 사용

- Java 파일 저장 시 반드시 클래스 이름과 동일하게 지정
- CMD를 통한 JAVA 컴파일 : cmd - *.jAVA 파일 경로 이동 - javac *.java
- Class 파일 실행 : cmd - java 클래스이름

JVM (Java Virtual Machine) 구조

- **ClassLoader** : 모든 클래스는 참조 순간에 동적으로 JVM에 연결되며, 메모리에 로딩
- **Execution Engine** : Class 에 정의된 내용 실행
- **Runtime Data Areas** : 프로그램을 수행하기 위해 OS에서 할당 받은 메모리 공간
- **Class 영역** : 사용하는 클래스 파일의 바이트 코드가 로드 되는 곳 , static변수, 전역변수 등의 정보가 저장되면 JVM이 종료될 때까지 유지
- **Stack 영역** : 지역변수, 매개변수 등 함수의 호출부터 종료까지 유지
- **Heap 영역** : 참조형 변수들이 저장되는 영역

Java 특징

- Sun Microsystems 사에서 개발
- 객체지향언어
- 현실에 존재하는 사물과 개념을 소프트웨어적으로 구현하고, 그 구현된 객체들이 상호 작용하여 데이터를 처리하는 방식
- 코드를 객체화 하여 독립적으로 존재할 수 있도록 한다 = 코드의 재사용 목적
- OS 상관없이 실행
- 메모리 관리를 개발자가 하지 않는다.
- 동적이며 스레드를 지원한다.
- 네트워크 프로그래밍과 분산처리 지원

Java 기본 형식

- 프로젝트(Alt + Shift + N) > 패키지 > 클래스 > 메서드
- 기본 문법

```
public class 클래스이름 {
```

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

        System.out.println("Hello World!");

    }

}
```

- **소스코드 구조**

클래스명.java

```
/* 클래스 블록 */
public class 클래스명 {

    /* 메서드 블록 */
    [public|private|protected] [static] (리턴자료형|void) 메서드명1(입력자
료형 매개변수, ...) {
        명령문(statement);
        ...
    }

    /* 메서드 블록 */
    [public|private|protected] [static] (리턴자료형|void) 메서드명2(입력자
료형 매개변수, ...) {
        명령문(statement);
        ...
    }

    ...
}
```

- **주석**

=> 한줄주석 : // 내용 (Ctrl + /)

=>여러줄주석 : /* 내용 ^/ (블럭 처리 후 Ctrl + Shift + /)

- **Println / Print / Printf**

=> println : 출력 후 줄바꿈

=> printf() ==> C언어의 printf 사용법과 동일

=> printf("출력서식", 출력할 내용);

=> 출력 서식은 출력 서식 문자를 제외한 나머지 문자는 그대로 출력

=> 출력 서식 문자 : d(정수), f(실수), c(문자), s(문자열)

=> 출력 서식의 형식 : %[-][0][n][.m]서식문자

=> 출력할 전체 자리수가 지정된 경우 왼쪽에 맞춰 출력

=> 0 : 출력할 전체 자리수가 지정된 경우 왼쪽의 남는 자리에 0을 채워 출력

=> n : 출력할 전체 자리수

=> .m : 소수점 아래 자리수, 잘리는 자리에서 반올림시켜 출력, 원래 데이터 값은 변경되지 않는다.

※ int : 4Byte , char : 2Byte , String : 문자열(C언어에서의 포인터) , flot : 4Byte , Double : 8Byte

※ ++x : 전위증감연산자(선증가 후출력) , x++ : 후위증감연산자(선출력 후증가)

※ 변수 이름 첫글자 숫자 x, _ 특수문자 가능

- **C와 Java 비교**

<Java>

```
public class Test {
    public static void main(String [] args) {                                                    **※ 배열객체**
        int [] a = {1 , 2 , 3 , 4 , 5 , 6);
        int i = a.length - 1;
        while (i >= 0) {
            System.out.print(a[i]);
            i--;
        }
    }
}
```

<C>

```
#include <stdio.h>
void main(void)
{
int a[6] = {1 , 2 , 3 , 4 , 5 , 6};                                                          **※ 배열변수**
int I = sizeof(a)/sizeof(int)-1;                                                    // (4byte * 6) / 4byte -1 : 6 - 1
while (i >= 0) {
printf("%d",a[i]);
i--;
}
}
```

- **Java 프로그램 구성요소**

1. 예약어(reserved word) : 키워드

- 자료형관련, 기억관련, 제어관련, 기타
- int, char, static, if-else, for, while

2. 명칭(identifier) : 식별자

- 클래스명, 메소드명, 변수명, 배열명, 함수명
- (관례) 클래스 이름의 경우, 대문자로 시작
- 메소드나 필드, 변수의 이름은 소문자로 시작

3. 상수(constant)

- 정수상수, 실수상수, 문자상수, 문자열상수
- 상수 : 키워드 final 사용 / 상수 이름은 모두 대문자로 선언하는 것이 관례 / 이름이 둘 이상 단어로 연결할 경우 _ 사용이 관례 / 선언과 동시에 초기화 해주는 것이 관례

※ C에선 const 키워드 사용

4. 연산자(operator)

- 복합대입연산자의 경우 형변환 조건을 무시한다. 필요 x
- 논리연산자의 경우 SCE 오류 주의 (증감이 필요한 경우 증감을 우선적으로 수행한 후 논리연산자로 결과 출력할 것)

※ 대입연산자와 논리연산자를 같이 사용할 경우 각별히 주의

※ 증감연산자 (++ : 1증가, -- : 1감소)

- ++a : 변수 a에 저장된 값을 1증가 시키고 사용한다.
- a++ : 변수 a에 저장된 값을 사용하고 1증가 시킨다. 현재 문장을 실행하고 ";"을 만나서 문장이 종료되는 순간 실행된다.
- --a : 변수 a에 저장된 값을 1감소 시키고 사용한다.
- a-- : 변수 a에 저장된 값을 사용하고 1감소 시킨다. 현재 문장을 실행하고 ";"을 만나서 문장이 종료되는 순간 실행된다.

```
ex)

int a = 1, b, c;

b= a++;

※ 선언된 a=1 값이 b에 저장 후 ";"을 만난 시점에 1증가 후 다음 문장으로 넘어간다.

c= ++a;

※ 이전 문장에서 넘어온 값(2)에 1증가한 값을 c에 저장, 즉 c는 3이 된다
```

- **연산자의 종류**

1) 산술 연산자 : + - * / % ++ --

2) 관계 연산자 : > < >= <= == != instanceof(객체의 타입을 확인하는 연산자)

```
ex) 참조변수 instanceof 클래스명 : 참조변수가 클래스명으로 만들어진 객체인지 묻는다.

System.out.println("hello" instanceof String);

System.out.println(Integer.valueof(3) instanceof Integer);           ※ int 자료형의 Wrapper 클래스

System.out.println(Float.valueof(3.14f) instanceof Float);           ※ float 자료형의 Wrapper 클래스

System.out.println(Double.valueof(3.14) instanceof Double);          ※ Double 자료형의 Wrapper 클래스

System.out.println(Character.valueof('a') instanceof Character);     ※ char 자료형의 Wrapper 클래스
```

3) 논리 연산자 : && || !

4) 대입 연산자 : += -= *= /= %= >>= |= &=

- a = a + b ==> a += b 과 동일

5) 조건 연산자 : ? :

6) 비트 연산자 : & | ^ ~ << >> >>>

- NOT 연산공식 : ~a = -a-1
- ex) !5 : -5-1 --> -6

※ 오른쪽 쉬프트 연산자 : 오른쪽 1칸씩 이동할 때 마다 /2

※ 왼쪽 쉬프트 연산자 : 왼쪽 1칸씩 이동할 때 마다 *2

7) 기타 연산자 : sizeof() cast & *new delete

8) 삼항연산자 : 항이 3개인 연산자 , (조건)? 참일때 값 : 거짓일때 값

```
ex) int age = 17;

    system.out.println(age>19?"성인입니다" :"청소년입니다");
```

※ "=="를 사용해서 같은가 비교할 수 있는 데이터는 기본 자료형과 NULL(아무것도 없는 상태)만 가능하다.

※ 문자열 비교시 "equals()" 사용 (String 문자열 클래스 내부 메소드(기능) 사용)

- **length()**: 변수에 저장된 문자열을 구분하는 문자의 개수를 얻어온다.

=> System.out.println("입력한 문자열의 크기 : "+ str.length());

- **trim()**: 문자열 앞, 뒤와 불필요한 공백을 제거한다.

=> System.out.println("입력한 문자열에서 불필요한 공백을 제거한 문자열의 크기 : "+ str.trim().length());

- **toUpperCase()**: 영문자를 무조건 대문자로 변환한다.

=> System.out.println("무조건 대문자로 출력 :"+ str.toUpperCase());

- **toLowerCase()**: 영문자를 무조건 소문자로 변환한다.

=> System.out.println("무조건 소문자로 출력 :"+ str.toLowerCase());

- **charAt(index)**: 문자열에서 index 번째 문자 1문자를 얻어온다. 문자의 위치를 계산할 때 맨 앞의 문자 index가 0부터 시작한다.

=> System.out.println("3번째 문자 :" + str.charAt(2));

※ 단항 / 이항 / 삼항 연산자로 구분

- **연산자 우선순위**

=> 최우선연산자 → 단항연산자 → 산술연산자 → 쉬프트연산자 → 관계연산자 → 논리연산자 → 삼항연산자 → 대입연산자

5. 설명문(commnet) : 주석, 비실행문

- **자료형, 변수와 상수**

1. 자료형

1) 기본자료형

- 자료형 예약어(크기, byte)
- **정수형 byte(1), int(4), short(2), long(8)**

=> int : 4바이트, -2147483648 ~ 2147483647 사이 정수 기억

=> short : 2바이트, -32768 ~ 32767 사이 정수 기억

=> long : 8바이트, -2의 63승 ~ 2의 63승 -1 사이 정수 기억

- **실수형 float(4), double(8)**

=> float : 4바이트, 단정도 실수, 소수점 아래로 6자리 정도 표현

=> double : 8바이트, 배정도 실수, 소수점 아래로 16자리 정도 표현, float 보다 더 정밀한 연산 가능

- **문자형 char(2) <- UNICODE(16bit), 1문자 기억**
- **논리형 boolean(1) <- true, flase**

=> 논리값을 기억하는 변수나 논리값을 리턴하는 메소드의 이름은 "is"로 시작하게 하는 것이 관행이다.

2) 참조형(배열, 문자열) : int anArr[]; , String rStr;

자료형	데이터	크 기	표현 가능 범위
boolean	참과 거짓	1 바이트	true, false
char	문자	2 바이트	유니코드 문자
byte	정수	1 바이트	−128 ~ 127
short		2 바이트	−32,768 ~ 32,767
int		4 바이트	−2,147,483,648 ~ 2,147,483,647
long		8 바이트	−9,223,372,036,854,775,808 ~ 9,223,372,036,854,775,807
float	실수	4 바이트	$\pm(1.40\times10^{-45} \sim 3.40\times10^{38})$
double		8 바이트	$\pm(4.94\times10^{-324} \sim 1.79\times10^{308})$

자바에서 기본적으로 제공하는 자료형이라 하여 '기본 자료형(Primitive Data Type)'이라 한다.

3) 리터럴

- 표현하려는 방식에 따라 L(LONG), F(float), D(Double)를 뒤에 붙여준다.
- 이스케이프 시퀀스(특수문자)

=> \" , \'" , \n

- **자료형변환**

=> 명시적 자료형

```
ex) doubble pi = 3.1415;

    int wholeNumber = (int)pi;

    char c = (char)a;
```

※ 명시적 형 변환 예시

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

      int num1 = 31111;                                          ※ 2의 15승 ~ 2의 15승 -1 범위 (-32768 ~ 32767)

      int num2 = 31111;

      short num3 = (short)(num1 + num2);

      System.out.println(num1 + "+" + num2 + "=" + num3);
```

※ 자동 형변환(묵시적 형변환) : 자료형의 크기가 서로 다른 자료의 연산 결과는 큰 자료형으로 자동 변환된다.

```
ex) float b = a;
```

- **아스키코드값**

=> 컴퓨터는 문자를 저장할 수 없다. 따라서 문자마다 고유한 숫자값을 부여했는데 그 숫자가 바로 아스키코드값

- **제어문**

1) if

- **if** (true or false) {

  조건 true 시 실행되는 영역 ---> if절

  }

※ if문에 속한 문장이 하나일 경우 중괄호 생략 가능

2) if ~ else

- if (true or false) {

  조건 true 시 실행되는 영역

  }

  조건 false 시 실행되는 영역

  }

※ if ~ else문과 유사한 성격의 조건 연산자

- 조건(true or false) ? true 시 반환 : false 시 반환

```
ex) big = (num1 > num2) ? num1 : num2;
```

3) switch 와 break

- if의 하위 호환 버전

```
switch(n) {
           case 1:          n이 1이면 여기서부터 실행
           case 2:
           case 3:
           default:         해당하는 case 없으면 여기서부터 실행
```

※ break를 만나는 순간 '가장 가까운 {}'를 벗어난다

```
ex) switch / Key
    switch (key) {
                case value:
                            key 와 value 가 일치할 경우 실행할 문자;
               [break;]
               [default:
                           key 와 일치하는 value 가 없을 경우 실행할 문장;
                break;
                   }
```

- **반복문**

1) for , while 그리고 do ~ while

- 횟수 : for문 사용

- 기간 : while , do ~ while문 사용
- 반복문의 3가지 필요 요소 : 변수 , 조건 , 증감

1-1) while

- while의 최초 진입 조건이 거짓이면 {} 블록을 한번도 실행하지 않는다. (do while의 경우 최초 진입 조건이 거짓이라도 {} 블록을 한번은 실행)

```
ex) int n = 0;

    while(n < 5) {

        결과내용;

        n++;

    }
```

1-2) for(변수 선언 ; 조건 ; 증감) { **※ 반복 횟수가 몇번인지 알 경우 사용**

결과 내용;

}

※ for문이 실행되는 원리

- 변수에 저장된 값으로 조건식을 실행해서 참이면 반복을 시작한다.
- {} 블록을 한 번 실행한 후 변수 값을 증감치 만큼 변경시키고 조건식을 실행해서 참이면 {} 블록을 반복하고, 거짓이면 {} 블록을 탈출한다.
- 변수 선언 전 반드시 초기화 후 사용
- 수식 계산 순서
- () 안의 수식 → 산술연산자(*, /, %) → 산술연산자(+, -) → 관계연산자 → 논리연산사 → 대입연산자
- 단항 연산자 → 이항연산자 → 삼항연산자

※ Java 와 Python for문 비교

JavaBreakAndContinue.java

```java
public class JavaBreakAndContinue {
    Run | Debug
    public static void main(String[] args) {
        for(int i = 0; i < 3; i++) {
            System.out.println(x:"--------------------");

            for(int j = 0; j < 3; j++) {
                System.out.print("[" + i + ", " + j + "] ");
            }

            System.out.print(c:'\n');
        }
    }
}
```

pythonBreakAndContinue.py

```python
for i in range(0, 3):
    print("------------------------------")
    for j in range(0, 3):
        print("[", i, ", ", j, "]", end=" ")
    print("")
```

- **continue**

=> 만나는 순간 조건 검사로 넘어간다, 전체적인 반복 횟수에는 영향을 미치지 않지만 키워드를 만남과 동시에 아래 내용은 생략한다

- **무한 루프**

=> for(;;) , while(true) , do ~ while(true)

- **반복문의 중첩**

```
for(;;) {
    for(;;) {
    }
}
for ;;) {
do {
   } while();
★ for문 안에 for문 , while문 안에 while문을 많이 사용
```

- **메소드의 이해와 정의**
- 기본 정의

```
접근제어자 반환타입 메소드이름(매개변수목록) {       ※ 선언부
          ※ 구현부
             }
```

=> **접근제어자** : 메소드에 접근할 수 있는 범위 명시

=> **변환타입(return type)** : 메소드가 모든 작업을 마치고 반환하는 데이터의 타입 명시

=> **메소드 이름** : 메소드를 호출하기 위한 이름 명시

=> **매개변수 목록(parameters)** : 메소드 호출 시에 전달되는 인수의 값을 저장할 변수들을 명시

=> **구현부** : 메소드의 고유 기능을 수행하는 명령문의 집합

(※ http://www.tcpschool.com/java/java_methodConstructor_method 참조)

※ 자바에서는 함수라는 개념은 존재하지 않음 , 파이썬에서는 둘 다 존재

- **함수** => 기능 상자(이름 , 코드 , 기능 3가지 요소로 구성)

1) 입력 x , 반환 x

```
ex)
        public static void no_in_no_out() {                                    ※ no_~ : 함수이름 , () : 입력 x
        System.out.println("입력 x 반환 x");
        }
- 결과값
입력 x 반환 x
```

2) 입력 o , 반환 x

```
      ex)
            public static void one_in_no_out(int num1) {          ※ (num1) : 매개변수, 파라미터 --> 함수가 정의되는 과정에서 바로 사용되는 변수
            System.out.println(num1 * num1);
            }
```

```
- 결과값

49
```

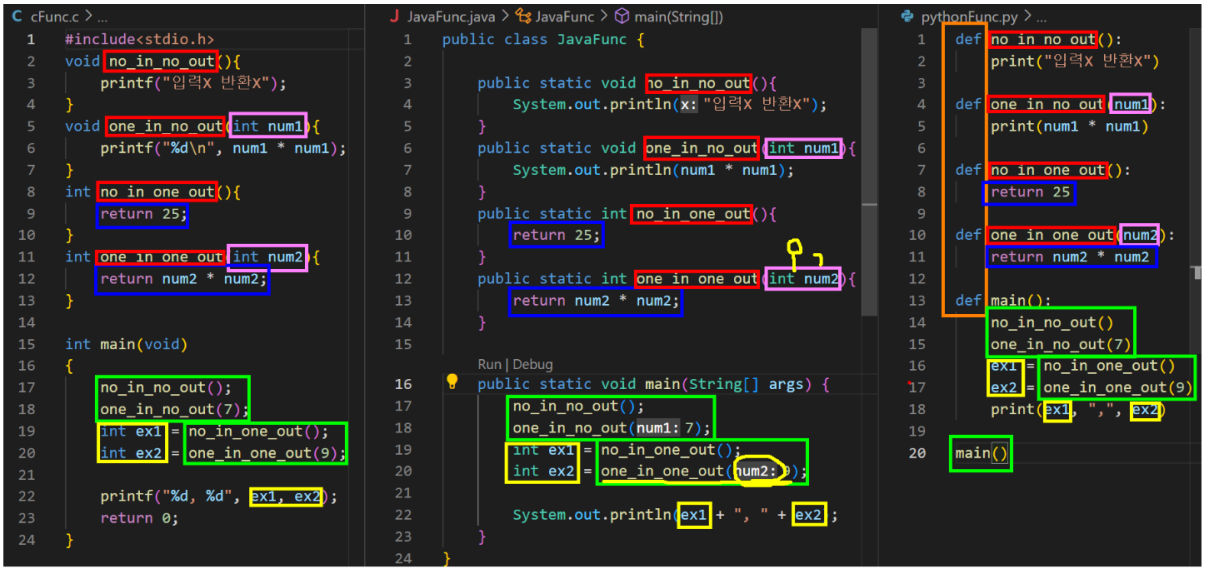
3) 입력 x , 반환 o

```
      ex)
              public static int no_in_one_out() {         ※ return 키워드를 만남과 동시에 값을 반환하고 호출 종료 (반환 / 함수 실행 종료 기능 수)

              return 25;                                  ※ return 뒤에 반환하려는 자료형을 명시해야 함
                                                          ※ 가능한 return(반환 값)은 통상적으로 하나만 사용하도록 함
              }

- 결과값

25
```

4) 입력 x , 반환 x

```
ex)
       public static int one_in_one_out(int num2) {

                return num2 * num2;

       }

       public static void main(String[] args) {              ※ void : 반환 x

                 no_in_no_out();

                 one_in_no_out(7);                           ※ (7) : 인수(인자) --> 호출하는 과정에서 전달하기 위해 사용

                 int ex1 = no_in_one_out();

                 int ex2 = one_in_one_out(9);

                 System.out.println(ex1 + "," + ex2);

                 }

          }

- 결과값

81
```

※ 언어(Java, Python, C) 별 함수의 종류



```c
// cFunc.c
#include<stdio.h>
void no_in_no_out(){
    printf("입력X 반환X");
}
void one_in_no_out(int num1){
    printf("%d\n", num1 * num1);
}
int no_in_one_out(){
    return 25;
}
int one_in_one_out(int num2){
    return num2 * num2;
}

int main(void)
{
    no_in_no_out();
    one_in_no_out(7);
    int ex1 = no_in_one_out();
    int ex2 = one_in_one_out(9);

    printf("%d, %d", ex1, ex2);
    return 0;
}
```

```java
// JavaFunc.java
public class JavaFunc {

    public static void no_in_no_out(){
        System.out.println(x: "입력X 반환X");
    }
    public static void one_in_no_out(int num1){
        System.out.println(num1 * num1);
    }
    public static int no_in_one_out(){
        return 25;
    }
    public static int one_in_one_out(int num2){
        return num2 * num2;
    }

    Run | Debug
    public static void main(String[] args) {
        no_in_no_out();
        one_in_no_out(num1: 7);
        int ex1 = no_in_one_out();
        int ex2 = one_in_one_out(num2: 9);

        System.out.println(ex1 + ", " + ex2);
    }
}
```

```python
# pythonFunc.py
def no_in_no_out():
    print("입력X 반환X")

def one_in_no_out(num1):
    print(num1 * num1)

def no_in_one_out():
    return 25

def one_in_one_out(num2):
    return num2 * num2

def main():
    no_in_no_out()
    one_in_no_out(7)
    ex1 = no_in_one_out()
    ex2 = one_in_one_out(9)
    print(ex1, ",", ex2)

main()
```

- **스코프 : 변수의 유효 범위**

=> **전역 스코프** : 변수가 '{}' 외부에서 선언되며, 프로그램의 모든 코드에서 엑세스 가능

=> **블록 스코프** : 변수가 '{}' 내부에서만 엑세스 가능

```
1) if(...) {

   int num = 5;

   .... 지역변수

   } num

2) public static void myFun(int num) {  매개변수 num

          ....

          }

3) for(int num = 1; num < 5; num++) {

          .....

          }

 ※ '{}' 속한 영역을 벗어나면 지역변수 소멸
```

- **메소드의 재귀호출**

=> 자신을 한번 더 호출하는 것 (종료됨과 동시에 가장 처음에 호출된 위치까지 나와야하는 구조)

=> 5! = 5 x 4 x 3 x 2 x 1 ==> n! = n x (n-1)! *** 수학적 측면**

```
ex)
    public static int factorial(int n) {

          if (n == 1)

                  return 1;

          else

                  return n * factorial(n-1);

          }
```

- **input 기능**

=> 문자열 입력 : 문자 1글자 입력시 키보드로 입력선언하는 방법

=> 선언 : String로 문자열 선언

=> char로 문자1글자 입력 받을 변수 선언

=> ex) **.charAt(자리수)

```
String temp;

Char op;

temp=in.next();

op=temp.charAt(0);
```

=> import java.util.Scanner 선언 -> Scanner sc(변수명) = new Scanner(System.in) -> 변수명(num) = sc.next입력받는 자료형(int, float 등);

- **입력메소드**

=> next() : 문자열을 입력 받는다. 띄어쓰기 전까지 입력 받는다.

=> nextInt() : 정수값을 입력 받는다.

=> nextFloat() : 실수형값으로 입력 받는다.

=> nextLine() : 문자열을 입력 받는다. 한줄 전체를 입력 받는다.

=> 문자열을 제외한 데이터 입력 후 nextLine() 메소드를 실행하기 위해 키보드 버퍼를 비우는 작업 수행

=> sc.nextLine();

- **클래스의 정의와 인스턴스 생성**

=> **클래스 = 데이터 + 기능**

=> 모든 코드는 클래스내 존재한다

=> 특정 클래스에 존재하는 함수, 변수를 활용하기위해 인스턴스(복사본) 클래스 생성

=> new 명령을 이용해 해당 클래스의 생성자함수 호출

=> 생성자함수 안에 출력하는 내용이 없을 경우 생략가능

=> 각 필요기능에 따라 사전에 규정된 패키지 내부에 있는 기능을 활용

=> 인스턴스 변수 , 메소드 = 멤버 변수, 멤버 메소드 --> 호출 시 직접 접근 = 클래스 소속 변수

=> 클래스의 명칭을 작성할 때 구현하려는 기능을 예측할 수 있는 이름으로 생성

=> 클래스로 만드는 모든 객체(변수)는 주소를 기억하는 참조형 변수이다.

=> 문자열(String)은 클래스화 되어있음

```
String str1 = "Happy"; ---> String str1 = new String("Happy");
```

=> 선언된 두개의 변수가 같은 메모리 주소를 참조하고 있는 경우 둘 중 하나의 변수의 참조값을 변경한다고 해도 나머지 하나의 참조값은 변경되지 않는다.

```
  ex)
       class Class_Exam {

             String bsw = "Happy";

       public static void main(String[] args) {

             Class_Exam aa = new Class_Exam();

             Class_Exam bb = new Class_Exam();
```

```
            System.out.println(aa.bsw + " , " + bb.bsw);

            aa.bsw = "good";

            System.out.println(aa.bsw + " , " + bb.bsw);
            }
      }
```

```
ex) 클래스 코드 예시

class BankAccount {
    int balance = 0;                          ※ 예금 잔액

    public int deposit(int amount) {
        balance += amount;
        return balance;
    }

    public int withdraw(int amount) {
        balance -= amount;
        return balance;
    }

    public int checkMyBalance() {
        System.out.println("잔액 : " + balance);
        return balance;
    }
}


public class BankAccountOO {
    public static void main(String[] args) {
        BankAccount yoon = new BankAccount();
        BankAccount park = new BankAccount();

        yoon.deposit(5000);
        park.deposit(3000);

        yoon.withdraw(2000);
        park.withdraw(2000);

        yoon.checkMyBalance();
        park.checkMyBalance();
    }
}
```

클래스, 인스턴스 관련 예제

```
class BankAccount {
    int balance = 0;    // 예금 잔액

    public int deposit(int amount) {
        balance += amount;
        return balance;
    }
    public int withdraw(int amount) {
        balance -= amount;
        return balance;
    }
    public int checkMyBalance() {
        System.out.println("잔액 : " + balance);
        return balance;
    }
}
```

```
class BankAccountOO {
    public static void main(String[] args) {
        // 두 개의 인스턴스 생성
        BankAccount yoon = new BankAccount();
        BankAccount park = new BankAccount();

        // 각 인스턴스를 대상으로 예금을 진행
        yoon.deposit(5000);
        park.deposit(3000);

        // 각 인스턴스를 대상으로 출금을 진행
        yoon.withdraw(2000);
        park.withdraw(2000);

        // 각 인스턴스를 대상으로 잔액을 조회
        yoon.checkMyBalance();
        park.checkMyBalance();
    }
}
```

```java
class BankAccount {
    int balance = 0;

    public int deposit(int amount) {
        balance += amount;
        return balance;
    }
    public int withdraw(int amount) {
        balance -= amount;
        return balance;
    }
    public int checkMyBalance() {
        System.out.println("잔액 : " + balance);
        return balance;
    }
}
```

```java
class DupRef {
    public static void main(String[] args) {
        BankAccount ref1 = new BankAccount();
        BankAccount ref2 = ref1;

        ref1.deposit(3000);
        ref2.deposit(2000);
        ref1.withdraw(400);
        ref2.withdraw(300);
        ref1.checkMyBalance();
        ref2.checkMyBalance();
    }
}
```

참조변수 관련 예제

=> 문자열이 최초로 만들어지면 메모리 어딘가에 문자열이 생성되고 문자열이 생성된 주소 값이 변수에 저장된다.

=> String str1 = "AAA"; ----> str1에는 "AAA"가 생성된 메모리의 주소가 저장된다.

=> 메모리에 같은 내용의 문자열이 있으면 다시 만들지 않고 기존에 있는 문자열의 주소값이 변수에 저장된다.

=> String str2 = "AAA"; ----> str2에는 str1에 저장된 "AAA"가 생성된 메모리의 주소가 저장된다.

=> if(str1 == str2) ----> "=="를 사용해서 비교했으므로 변수에 저장된 문자열 자체를 비교한 것이 아니고 변수에 저장된 주소를 비교하게 된다.

=> new라는 예약어를 사용해서 객체를 생성하게되면 메모리에 같은 내용 존재 유무 상관 없이 무조건 다시 만든다.

=> String str3 = new String("AAA");

=> 문자열을 비교하는 경우 반드시 equals() 메소드를 사용

=> ref = null; **※ ref가 참조하는 인스턴스와의 관계를 끊음**

=> 클래스 정의 시 구분에 필요한 고유정보 입력

- **String 클래스**

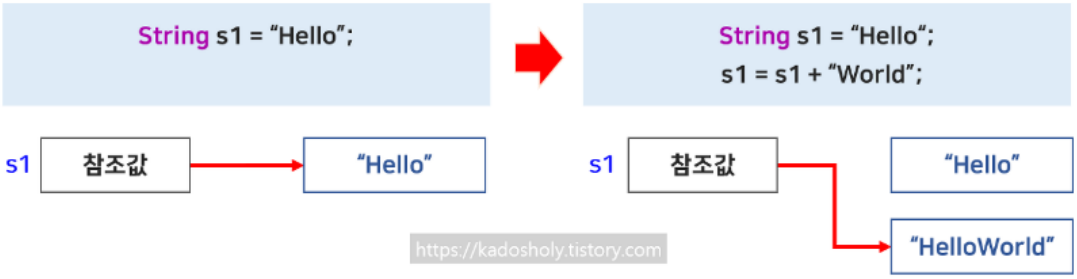
=> 자료형

=> java.lang 패키지에 포함되어 여러가지 정의된 기능 제공

String객체 생성에 따른 차이

※ String 객체는 내용을 수정할 수 없는 immutable 클래스

```
1. String s1 = "Hello";
2. s1 = s1 + "World";
```



String 참조변수에 새로운 문자열을 대입시킨경우

※ String Class 사용 예시

=> 선언된 **s1.(메소드) 형태**로 사용가능

```
String s1 = "ABC";
System.out.println(s1.length());

[실행결과]
3
```

- **생성자(Constru)**

=> 선언(생성) 시 클래스 이름으로 생성

=> 인스턴스를 생성할 때 인자값을 전달해주어야 함, 아닐 시 문법 오류

=> 클래스이름과 동일한 함수 : 생성자함수

=> 인스턴스 생성과 동시에 생성자 함수 호출

```
ex)

    Triangle(int base , int high)

    Triangle bs = new Triangle(50, 60);
```

=> 클래스 이름 생성 시 첫문자는 대문자, 두 단어 이상 묶어서 사용 시 단어의 시작하는 첫문자 대문자 (Camel Case 모델)

- **클래스 패스**

=> 클래스나 패키지를 찾을때 기준이 되는 경로 (:으로 구분된 디렉토리 및 파일 목록)

=> 경로지정방법

=> /export/home/username/java/classes와 같은 디렉토리

=> myclasses.zip과 같은 zip 파일

=> myclasses.jar와 같은 jar(자바 아카이브) 파일

=> 절대경로와 상대경로 사용

=> 사용 시 모든 경로를 표현해야하지만 'import' 를 통해 해당 클래스의 이름만 명시하고 사용할 수 있도록 함

- **정보은닉**

=> 접근 수준지시자

1) private int a;

```
ex) public void set(int x){                                    ※ 동일 패키지라도 접근 불가
                                                               ※ 접근을 위해 set, get 함수 사용
    a=x;                                                       ※ 클래스 내부에서만 접근 가능
```

2) default int b; (일반적인 선언)

※ 같은 패키지 + 같은 클래스

★private + 패키지

※ 선언하지 않은 경우 기본적으로 저장되는 영역

3) protected int c;

※ default 선언 + 다른패키지 상속가능 (extends)

```
public classs AAA {
    int num;
}

public class zzz extends AAA {
    public void init(int n){
      num = n;                    ※ 접근가능
  }
}
```

4) public d;

※ 공통적으로 사용

※ 동일 클래스내에 단독적으로 존재한다

- **캡슐화**

=> 우선순위를 무시하는 코드 작성으로 인한 문제 발생

=> 클래스는 그대로 유지하되 필요한 기능을 다양하게 사용할 수 있도록 새로운 인스턴스를 생성해서 사용하는 방법 추천

- **static 선언을 붙여서 선언하는 클래스 변수 (정적 변수)**

=> 전체 프로그램에 하나만 존재하는 변수

=> 클래스 내에서 이름만으로 접근 가능 or 외부에서 클래스 이름으로 접근 가능

※ 인스턴스와 전혀 관계 없기 때문에 클래스 이름으로 접근하는 것을 추천함

=> **클래스 변수는 생성자에서 초기화 하면 안됨** = 인스턴스 생성과 동시에 기존에 저장된 값이 초기화 선언된 값으로 바뀌기 때문에

=> 값의 참조 또는 공유가 목적인 변수를 선언할 때 사용

=> 선언된 변수나 함수 복사 불가

=> 서로 다른 메인에서 호출해서 사용하는 경우

=> 변수나 함수 사용 시 생성 불필요

★static 메소드는 static멤버와 로드되는 시점이 동일하기 때문에 새로운 인스턴스가 생성되기 전 상황에서도 사용할 수 있어야 한다

- **static 변수 사용**

=> 인스턴스에 따라서 변하지 않는 값이 필요한 경우

=> 인스턴스를 생성할 필요가 없는 값을 전역변수로 사용하려고 하는 경우

=> 변경 사항을 모든 인스턴스가 공유해야 하는 경우

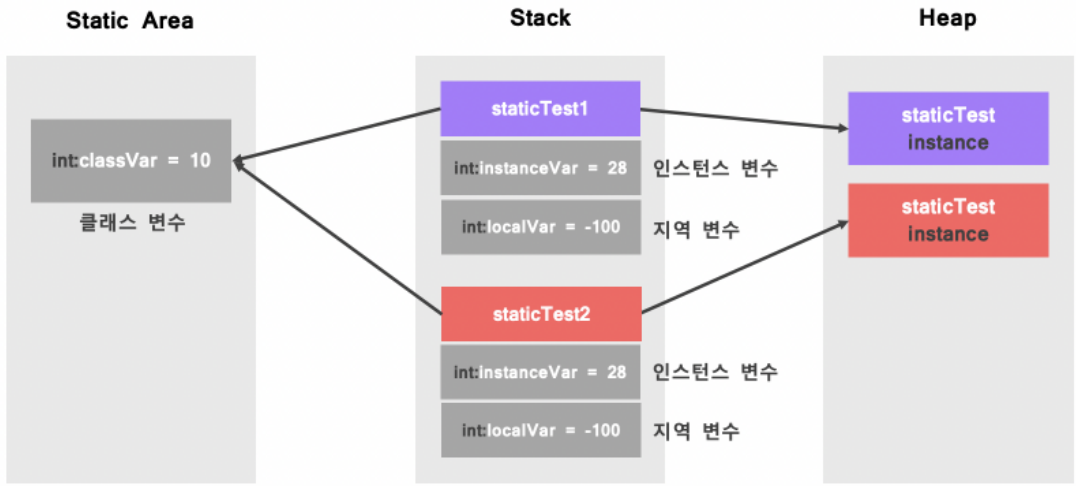
```
※ 클래스 변수 예시
```

```
public class Student {
  static int count = 0;           ※ 클래스(공유) 변수
  int id;                         ※ 인스턴스(멤버) 변수
  String name;                    ※ 생성자

  Student(String name) {
    count++;
    id = count;
    this.name = name;
  }
  public static void main(String[] args) {
    Student a = new Student("Tom");
    Student b = new Student("John");
    Student c = new Student("Kate");
    System.out.println("전체 학생수: " + Student.count);
  }
}

결과 값
전체 학생수: 3
```

※ 여러 변수의 선언 위치

변수의 종류	선언위치	생성시기(메모리 할당 시기)
클래스 변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수		인스턴스가 생성될 때
지역 변수	클래스 이외의 영역 (메서드, 생성자, 초기화 블럭)	변수 선언문이 수행 되었을 때

※ 클래스 / 인스턴스 / 지역변수가 메모리에 적재되는 위치



※ Java의 main 이 pubilc static인 이유

- 모든 자바 프로그램은 메인으로 시작해서 메인으로 끝나기 때문에 전체 코드에 1개만 존재해야함 --> static **(컴파일 되는 순간 정의 됨)**
- 외부에서 jvm이 접근하기 위해서 --> public 사용

```
 public       static                 void                  main          (String[] args)
접근제어자    정적함수     반환 없이 실행만 하겠다는 의미        연속적인 문자열 데이터가 들어가는 저장공간
```

- **배열 : 같은 자료형의 변수들이 나열된 묶음**

=> 타입[] 배열명; --> ex) int[] ar; **※ 선언**

=> 배열명 = new 타입[길이]; --> ar = new int[3] **※ 생성**

--> 타입[] 배열명 = new 타입[길이] **※ 선언과 생성을 동시에 하는 초기화**

```
public static void main(String[] args) {
    int[] num = new int[3];                      ※ 크기가 3인 배열 생성
    num[0] = 10;                                 ※ 0번 index에 값 할당
    num[1] = 15;                                 ※ 1번 index에 값 할당
    num[2] = 13;                                 ※ 2번 index에 값 할당
    for (int i = 0; i < num.length; i++) {
      System.out.println(num[i]);
    }
  }
```

=> 배열의 위치를 지정하는 첨자(index)는 0부터 시작하는 것에 주의하자

=> **배열의 초기화** : 정수형 0, 문자 공백(''), 실수형 0.0 , boolean는 false, 문자열로 만든 배열은 NULL로 자동으로 초기화 된다.

=> 배열명.length : 배열의 크기를 얻어온다.

=> Arrays.toString[배열명] : 배열의 요소를 문자열로 출력하기

```
※ 배열 예시

public static void main(String[] args) {
    String[] beer = {"Kloud", "Cass", "Asahi", "Guinness", "Heineken"};
      ※ 인덱스 번호 :   0  ,    1   ,   2   ,     3      ,     4
    System.out.println(beer[0]); // Kloud
    System.out.println(beer[1]); // Cass
    System.out.println(beer[2]); // Asahi
    System.out.println(beer[3]); // Guinness
    System.out.println(beer[4]); // Heineken
  }
}
```

- **오버로딩**

=> 하나의 클래스 안에 함수가 동일, 하지만 변수가 다름

=> 인수 넣는 수에 따라 결과값 출력

```
-. ex)
       class Over_Exam {

        int mux(int x) {

        return x;

        }

        int mux(int x, int y) {

        return x + y;

        }

        int mux(int x, int y, int z) {

        return x + y + z;

        }

        public static void main(String[] args) {

           Over_Exam r = new Over_Exam();

           int num1;

           num1 = r.mux(10);

           System.out.println(num1);

           num1 = r.mux(20, 30);

           System.out.println(num1);
```

```
            num1 = r.mux(50, 60, 70);

            System.out.println(num1);

        }
```

- 생성자 오버로딩

=> 생성되는 인스턴스의 유형 구분

- 키워드 this를 이용

=> 생성자 안에서 사용될 경우 입력 받은 인자를 받을 수 있는 메소드가 호출 된다

```
public mux(int n1, int n2)
    ....

public mux(int x)
    this(10, 7)

※ 여기서 this 가 가르키는 메소드는 mux(int n1, int n2) 이다
```

=> 변수

```
int num1 = 10;

public mux(int num1)

this.num1 = num1

※ this.num1 = int num1 = 10;
   num1 = public mux(int num1)
   즉, 최초 선언된 num1의 값 10을 입력받은 인자의 값 num1로 변경한다
```

- **인터페이스**

=> 모든 내용이 추상메소드로 구성

=> 상수 : 다른값으로 정의할 수 없는 고정된 값 (호출시 생성자 함수 불필요. 클래스 바로 호출)

※ 상수로 선언할때 이름 대문자 사용 (변수 정의 x, 사용 x)

=> implements : 인터페이스 상속

※ 인터페이스의 경우 상속받을 시 재정의할 내용이 없어도 반드시 문법으로 기재하여야함

※ 인터페이스는 다중상속 가능

- **인스턴스 생성불가 선언**

=> 변수 : static 으로 선언

=> 상수 : final static 선언 (상속불가)

- **업캐스팅**

```
RemoteControl rc = new Television();
rc.turnOn();
rc.turnOff();
rc.setVolume
```

- **다운캐스팅**

```
rc = new Audio();
Audio t = (Audio)rc;
```