08강. 객체지향 - 클래스

1. 객체란?

객체(object)는 조작 "적용 대상"이다. 더욱 상세하게 말한다면, 데이터와 그것을 처리하는 방법(메소드)를 하나의 캡슐로 표현한 것이다.

객체 지향(oo : object oriented)은 현실 세계에서의 작업의 개념을 소프트웨어에서 제공하는 설계 사상이다 객체 지향에서는 객체라는 물건끼리 대화를 하고 이에 따라 각각 행동 할 작업을 진행한다.

프로그램은 다양하게 원시 프로그램, 구조 프로그램, 객체 지향 프로그램으로 나뉜다.

>>원시적 인 프로그래밍(절차 지향 언어)

객체 지향 이전의 언어는 CPU의 동작을 프로그램에 적용 된다. 처음에는 CPU의 동작을 일일이 2진수로 기술하고 있으며 이것은 기계어라는 언어라고 부른다. 기계어의 이진수 혹은 16진수로 작성된 단어를 인간이 이해할 수 있는 단어 에 일대일로 번역 한 것이 어셈블리 언어라는 것이다. 그리고, 구문 자체를 인간이 이해할 수 있는 형태로 접근 한 고급 언어가 태어났다. FORTRAN, C, COBOL등이 그 대표적인 언어이다. 이것들은 모두 컴퓨터의 동작을 기술 한 절차 지향 언어라고 부른다.

절차 지향 언어에서는 컴퓨터의 제어 흐름에 주목하여 프로그램을 설계한다. 컴퓨터 제어의 흐름을 권유 작성하는 프로그램이며, 데이터는 단편적인 생성 / 소멸 / 변환을 반복 처리가 진행된다.

>>구조적 프로그래밍

절차 지향 프로그래밍은 구조적 프로그래밍 이라는 기술로 세련화 되었다. 구조적 프로그래밍에서는 데이터의 흐름 (데이터 흐름)에 주목하여 프로그램을 설계한다. 프로그램 코드는 입력 데이터를 변환하여 출력하는 형태와 같다. 이 때, 데이터 변환되어 소멸하는 것은 아니다. 어떤 데이터가 어디에서 들어와서 어떤 변화를 겪고, 어떤 데이터가 어디에서 출력되는지 생각하고 설계한다.

절차 지향 프로그램에서는 처리 및 데이터는 다른 것으로 분리되어 있다. 데이터는 실행시에 서브 루틴 인수의 형태로 프로그램에서 처리에 전달되어 처리에 의해 변환을 받고 다른 작업에 전달하고 있는 구조를 취한다.

>> 객체 지향

객체 지향은 데이터 흐름 중심주의를 더욱 발전시킨 것이다. 객체 지향 프로그램은 데이터와 행동을 하나로 묶어 클래스의 집합이라고 생각하고 설계한다. 처리 클래스에서 생성 된 객체들이 대화를 하는 것으로 실현한다.

원래는 실제 컴퓨터에 사상 하기 위한 모델링을 목적으로 개발되었다. 실제 작업의 대부분은 주관적으로 보면 버튼을 누르고 당기는, 목적지를 말하는 등의 물건을 상대하는 것이다. 객관적으로 보면 일련의 작업은 많은 물건들이 협조하여 실현하고 있다. 물건들은 다른 물건의 지시를 받고 자신의 역할에 따라 행동 결과를 보고 한다. 여기서 말하는 물건이 객체입니다.

객체 지향 프로그램에서는 처리와 데이터를 하나로 개체를 처리한다. 객체는 다른 객체와 메시지를 교환하여 행동한다. 객체 지향 컴파일러는 데이터 처리를 조합하여 객체를 생성하는 코드를 생성한다. 실행 환경은 객체를 생성하고 객체 간의 메시지 교환을 통해 프로그램이 작동한다.

>>객체 지향과 절차 지향의 차이

절차 지향은 데이터와 처리가 분리되어 있다. 객체 지향에서는 처리와 데이터를 조합 한 개체를 처리한다.

절차 지향 데이터 처리에 전달되어 변환을 받을 수 프로그램의 실행이며, 컴퓨터 제어의 흐름을 설명하는 것이었다. 객체 지향에서는 실행 시 데이터 처리가 하나 된 개체가 개체간에 메시지를 교환 할 수 프로그램을 실행한다. 개체는 컴퓨터 제어가 해당 객체의 처리에서 빠져도 메모리에 계속 존재 지속적으로 액세스 할 수 있다. 즉, 컴파일러와 실행 환경의 메모리 관리 등 구현 수준에서 절차 지향은 다르다 것이다 라는 것이다. 또한 절차 지향과 객체지향은 컴파일러와 실행 환경의 구현의 차이에 따라 설계 방법이 달라 분석 방법도 다르게 사용되며 개발 프로세스와 개발 방법이 다르다 .

객체 지향의 원류는 SIMULA (1966-)이며, 게다가 LOGO (1968-) Smalltalk (1972-) 등을들 수 있다. 객체 지향을 도입 한 SIMULA 과 LOGO 를 융합시킨 Smalltalk 는 XEROX 의 PARC 연구소에서

개발되었다. SIMULA 객체 지향을 세련화시키고 시장에 인식시킨 언어라고 한다. 실제로 현대적인 객체 지향 언어의 대부분은 Smalltalk 설립 한 객체 지향을 도입하고 있으며 SIMULA 와 C 를 융합 시켰다는 C ++는 시장에서 큰 성공을 거둔 언어이지만, 순수 객체 지향 언어가 아니라 절차 지향 언어의 하이브리드 언어로 간주한다.
Java 는 SIMULA, Smalltalk, C + + 등을 기반으로 구현된 언어이다 .

2. OOP 의 특징

>>객체 용어

객체 지향 프로그래밍뿐만 아니라 설계 기법, 모델링, 개발 프로세스 전반에 걸쳐 도입되는 포괄적인 개념이 되었다. 다음은 객체에 대한 용어이다.

객체 지향 분석 및 설계 OOA (Object Oriented Analysis) , OOD (Object Oriented Design): 객체 지향 개념을 도입 한 분석 · 설계 기술과 개발 방법론

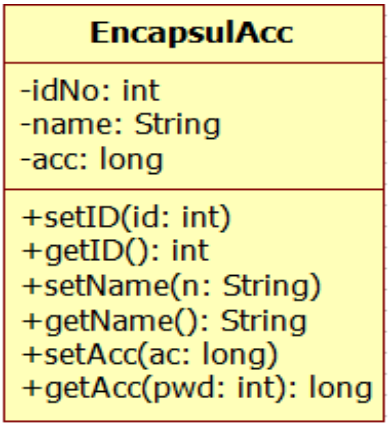
객체 지향 프로그래밍 OOP Object Oriented Programming : 객체 지향 개념을 도입 한 프로그래밍 기법이다. 객체 지향 프로그래밍에서는 종종 객체 지향 프로그래밍 언어가 사용되지만, 비객체 지향 프로그래밍 언어에서 객체 지향 프로그래밍을 할 수 있다.

객체 지향 관련 기술은 다음과 같다

UML (통합 모델링 언어) : 객체 지향 모델을 그래픽으로 표현하는 표기법. OMG 라는 단체에 의해

가. 캡슐화(Encapsulation) : 사용자 자료형인 UserDataType 으로 클래스를 만들어 데이터를 관리하는 멤버변수는 접근지정자인 private 로 은닉시키고 공개형인 public 으로 값을 전달 및 변경할 때는 void 형 setter 메소드로 리턴받을 때는 리턴형 getter 메소드의 원형을 만들어 구현하는 방법을 말한다. 자바는 Full Encapsulation 를 지향한다.

다음은 Full Encapsulation 을 구현한 EncapsulAcc 클래스이다. 멤버 변수인 idNo, name, acc 는 은닉된 private 로 지정하고 공개형 메서드인 getter & setter 가 값을 리턴 및 전달, 변경을 할 수 있다.



- private ~ default

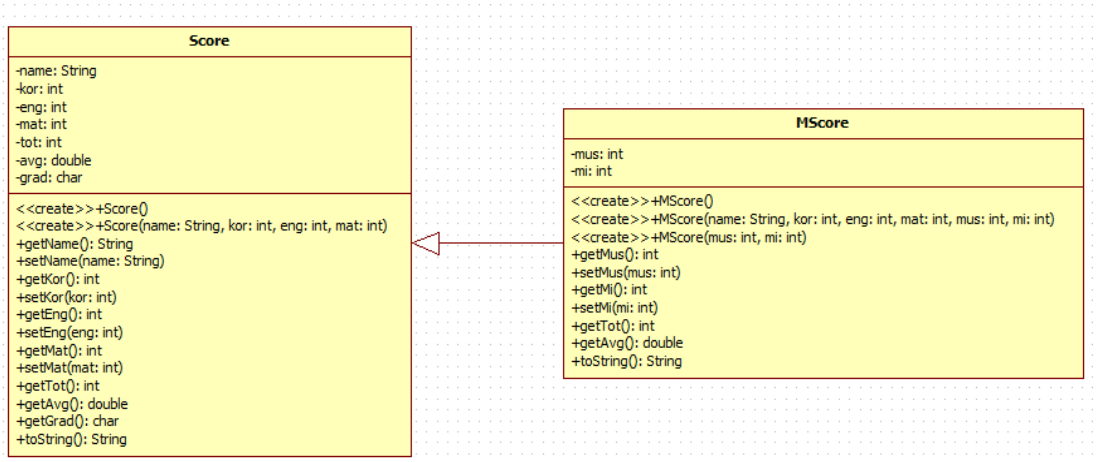
 # protected + public

캡슐화 된 코드는 다음과 같은 특징이 있어야 한다.

① 모든 사람이 액세스하는 방법을 알고 있어야 한다.
② 쉽게 구현에 관계없이 세부 사항을 사용할 수 있어야 한다.
③ 애플리케이션의 나머지 코드의 부작용이 없어야 한다.

캡슐화는 클래스 (인스턴스)를 하나의 개체로 독립성을 높이려는 것이다.
구체적으로 말하면, "멤버 변수"또는 "방법"에 대해 "액세스 한정자"를 적절하게 설정하고 클래스 (인스턴스)에 대한 액세스 제어한다. 따라서 클래스 (인스턴스) 액세스 (사용 방법)가 통일되어 독립성이 높아져, 그 결과 예상치 못한 버그를 방지하고, 클래스 (인스턴스)의 변경이나 재조합이 용이하도록 한다.

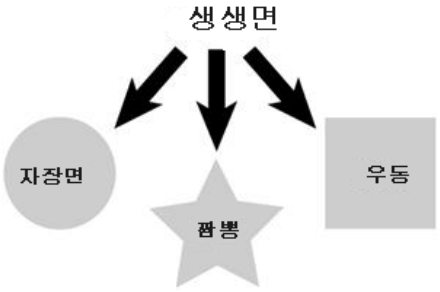
나. 상속 (**inheritance**): 상속은 클래스를 통해 기능을 확장하는 개념으로 상속은 객체가 몇 가지 / 다른 개체의 모든 속성을 취득하는 메커니즘이다. 이는 계층 분류의 개념을 지원한다. 상속 관계는 "is a"관계 즉 "부모 클래스 A'와'자식 클래스 B'가있을 경우 "B is A '(B 는 A 이다)이다. 따라서 B 를 A 로 취급도 가능하다.이것은 클래스 간의 연계를 보다 추상적(일반화)으로 하여 개체의 독립성을 높인다는 객체 지향 설계에서 가장 중요한 요소 중 하나를 실현하고 있다.



예를 들어 Score 클래스를 만들어 3 과목의 총점, 평균을 구하도록 설계 했다면 Score 클래스를 MScore 가 상속받아 2 과목만 추가하여 5 과목의 총점, 평균을 구하도록 확장하는 것이다. 이때 Score 클래스를 선조 클래스라고 하고 기능 추가된 클래스인 MScore 는 후손 클래스가 된다. 봉급 프로그램을 생각해 보자. 기본급을 계산하는 클래스를 설계 했다면 상속받아서 세금, 성과급, 보너스등을 추가해서 확장 클래스를 만들 수 있다.

다. 다형성 (**polymorphism**) : 다형성은 데이터 타입에 기반하여 다양하게 후손의 객체를 처리하는 것을 의미한다. 자바에서의 "다형성"은 "추상 클래스"및 "인터페이스"등을 이용하여 메소드 호출 방법을 공통화하고, "무시"하는 것으로 같은 메소드를 호출해도 실제 인스턴스마다 그 행동을 변화 시키려고하는 것이다. 예로 중국집의 음식을 생각할 수 있다.

먼저 면을 선조 클래스로 두고 선조의 면에 추가되는 메소드를 add()라고 선언하여 후손이 모두 add()라는 메소드를 통해 동일하게 면이 들어가지만 자장면, 짬뽕, 울면, 우동 등을 다양하게 만들어 내는 결과로 후손 클래스로 구현할 수 있다.



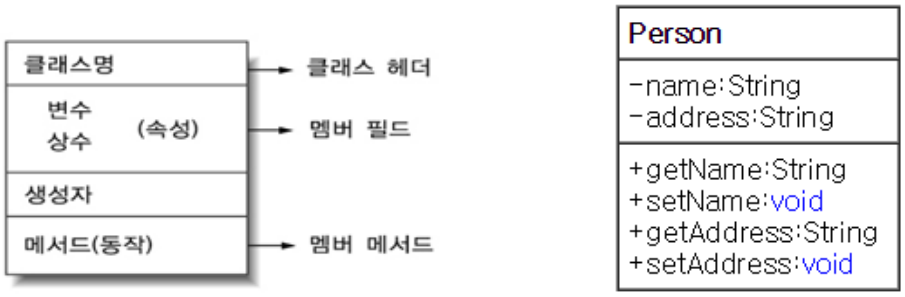
부모 클래스 (공통 클래스)의 형태로 클래스 연계 및 메서드 호출까지 실제 인스턴스를 의식하지 않고 할 수 있기 때문에 클래스의 독립성을 높일 수 있다.

3. class

>> class 란?

클래스란 '객체를 정의해놓은 것.' 으로 '객체의 설계도 또는 틀'이라고 정의할 수 있다. Java 프로그램은 개체(프로그래밍에서의 객체는 클래스에 정의된 내용대로 메모리에 생성된 것을 뜻한다.)의 집합이다. 런타임에 개체끼리 메시지를 보내면서 작동하지만, 객체는 다른 객체에 의해 클래스에서 만들어진다. 클래스는 개체의 설계도와 같은 것으로, 프로그래머는 독점적으로 클래스를 코딩 한다. 다음 클래스의 구조를 살펴 보자.

Person이란 클래스는 name,address라는 두 개의 속성과 속성에게 값을 전달 및 변경하는 setName,setAddress의 메소드, 속성값을 리턴하는 getName, getAddress의 메소드를 가지고 있다.



클래스의 구성 요소를 클래스의 **멤버**라고 클래스에서 만들어진 개체를 사용할 한 클래스의 멤버도 지속적으로 사용할 수 있다. 클래스의 멤버는 속성과 동작(기능, 메시지)로 이루어진다. 속성은 멤버 변수 , 필드를 말하고 메시지는 동작을 실행하는 메소드를 말한다.
클래스의 멤버의 뜻을 살펴 보자.

① 멤버 변수 (필드 또는 클래스의 속성)

개체의 상태를 유지하는 변수와 상수를 말한다. 이들을 초기화하여 클래스를 인스턴스화한다. 지금까지 다뤄 온 메소드에 정의 된 변수는 해당 메소드 내부에서만 유효 하지만 멤버 변수는 클래스의 인스턴스를 만든 객체가 지속하는 한 소멸되지 않는다. 덧붙여서, 지금까지 등장했던 변수와 같이 메소드 내에서 선언 변수를 "임시 변수", "지역 수"라고 하고 클래스에 선언된 멤버 변수는 클래스의 전역 변수라고 말한다.

② 메서드(멤버 메서드)

객체의 기능 / 역할을 기술한다. 객체를 조작하는 방법이라고 할 수 있다. 기본적으로 멤버 변수는 멤버 메소드를 통해 액세스 한다. 메소드에 기술하는 것은 컴퓨터 제어이며, 객체 지향이란 객체의 메소드를 통해 컴퓨터 제어를 지시하는 프로그래밍 스타일이다.

③ 생성자
생성자는 클래스의 멤버 변수를 초기화하여 메모리에 객체를 생성한다. 클래스로부터 객체를 만드는 과정을 클래스의 인스턴스화(instantiate)라고 하며, 어떤 클래스로부터 만들어진 객체를 그 클래스의 인스턴스(instance)라고 한다. 인스턴스화 할 때마다 생성자가 호출 되어 진다.

>> 클래스의 선언

클래스는 다음과 같이 선언해서 사용한다.

```
[한정자 class <클래스 이름> {

// 멤버 변수
        [한정자 <형식> <변수 이름> = 값;

// 생성자
        액세스 한정자 <클래스 이름> ([매개 변수]) {
              인스턴스화 처리(멤버변수 초기화)
}

// 메소드
[한정자 <반환 값 형> <메서드 이름> (메서드 인수) {
          실행할 명령
}
}
```

각 멤버에는 한정자를 지정할 수 있습니다. 아무것도 지정하지 않은 경우에도 기본 한정자를 선택한 것입니다. 한정자는 크게 두 가지로 분류됩니다. 액세스 한정자와 특수한 뜻을 가진 수식으로 나뉜다.

① 액세스 한정자

➔ 액세스 제어를 실현하는 규정 자. 패키지 또는 클래스 단위로 프로그램 코드의 다른 부분에서의 액세스 가능성을 기술한다.

`private` ... 자신이 포함 된 클래스 내에서만 액세스 한다

`protected` ... 같은 패키지 및 상속하는 서브 클래스에서만 액세스 한다

`public` ... 모든 위치에서 액세스 할 수 있다

`default` ... 액세스 한정자를 지정하지 않은 경우. 패키지 액세스라고도 불린다. 같은 패키지에 속하는 클래스에서만 액세스 할 수있다.

② 수식

➔ 액세스 한정자 이외의 수식은 다음과 같이 정의 된다.

`abstract` ... 추상 클래스 추상 메서드. 상속되어 구현 기술되어 처음 인스턴스화

가능해진다.

static ... 인스턴스화도 하지 않고 사용할 수 있는 클래스 고유의 멤버.

final ... 상속 못하고 재정의 할 수 없는 멤버. 코어 패키지의 클래스는 대부분 final.

strictfp ... FP 준엄. 부동 소수점 수가 CPU의 처리 계에 의존하지 않고 엄밀하게 같은 연산 결과가 된다.

transient ... 직렬화 (직렬화) 불가능하며 JVM 외부에 내보낼 수 없는 멤버에게 선언.

volatile ... 멀티 스레드에 의한 액세스시 반드시 마스터 복사본과 동기화를 구현

기존의 절차 적 언어는 객체 지향 언어에서 보면 메소드 내부에서만 명령을 구현했다면 Java는 기존의 클래스를 생성자를 통해 인스턴스화 하는 것으로, 그 클래스의 멤버 변수 와 메소드를 사용할 수 있다.

위의 한정자는 클래스, 멤버 변수, 메소드 ,생성자 앞에 선언할 수 있으며 그 용어나 기능이 어디에 선언되는 지에 따라 의미가 약간 달라진다.

클래스 앞에 선언되는 한정자는 다음과 같다.

abstract	추상 클래스로 상속되는 것을 명시한다. 이 클래스는 직접인스턴스화 할 수 없는 클래스이며 서브 클래스에서 구현하여 인스턴스화를 시킨다.
final	상속 받고 싶지 않은 것을 명시한다. 더 이상 확장 / 변경되고 싶지 않을 때 사용한다.
public	모든 클래스에서 참조 할 수 있다. 상속도 인스턴스화도 제한이 없다.
default	같은 패키지 내에서만 참조 할 수 있다

메소드 내에서 정의 된 변수의 범위는 해당 메소드내 밖에 못 미친다. 멤버 변수의 경우 다른 클래스에서 액세스 할 수 있기 때문에 그 액세스 제한을 규정 자로 실현할 수 있다.

다음은 멤버 변수의 한정자 있다.

public	액세스에 제한이 없다.
protected	같은 패키지 또는 그 서브 클래스에서만 액세스 할 수.
default	같은 패키지 내에서만 액세스 할 수.
private	같은 클래스에서만 액세스 할 수.
static	정적 변수. 보통의 변수는 인스턴스마다 다른 값을 유지하고 인스턴스 변수라고. 한편, 정적 변수는 인스턴스에 관계없이 공통의 메모리 공간을

	차지한다.
final	초기화 값 이외에 변경할 수 없다. 상수로 사용한다.
transient	객체의 직렬화(Serialization) 때 파일이나 데이터베이스 등의 지속적인 기억 영역에 저장하지 않는다.
volatile	스레드는 변수에 액세스하면 해당 스레드의 작업 복사본을 만들어 작업하고 개체 잠금 및 잠금 해제 시 마스터 복사본과 일치한다. 이 수정자가 붙여진 필드는 액세스 할 때마다 작업 복사본을 마스터 복사본과 일치시킨다.

메소드의 수식은 다음과 같다.

public	어디서든 호출.
protected	같은 패키지 서브 클래스에서만 호출한다.
default	같은 패키지 내에서 밖에 호출 할 수 없다.
private	같은 클래스에서만 호출 할 수 없다 (메시지가 도착하지 않는) 방법.
abstract	추상 메소드. 추상 메소드를 하나라도 가진 클래스는 추상 클래스입니다. 메소드 이름, 인수, 반환 값은 정의되어 있지만 블록의 처리 (구현)이 이루어지고 있지 않다.
static	정적 메서드입니다. 해당 메소드가 정의 된 클래스를 인스턴스화 하지 않고도 이용할 수 있다. 인스턴스에 상관없이 동일한 기능을 하는 메소드에 사용한다.
final	재정의 할 수 없는 메서드. 상속에 의해이 메소드의 기능 변경 / 확장 할 수 없다.
synchronized	멀티 스레드시 동기화 방법. 스레드 호출 된 시점에서 잠겨 있기 때문에 여러 프로세스에 의해 수행 될 수 없게 된다.
native	C / C + + 언어 등의 플랫폼 의존 코드에서 생성 된 DLL 등을 실체로 하는 메소드.

다음은 이름과 주소를 Person이라는 나만의 자료형인 userDataType인 Person클래스의 다이어그램을 코드로 옮긴 예이다.

```
public class Person {
private String name;
    private String address;

public String getName() {
        return name;
    }
```

```
    public void setName( String name_ ) {
        name = name_;
    }

    public String getAddress() {
        return address;
    }
    public void setAddress( String address_ ) {
        address = address_;
    }
}
```

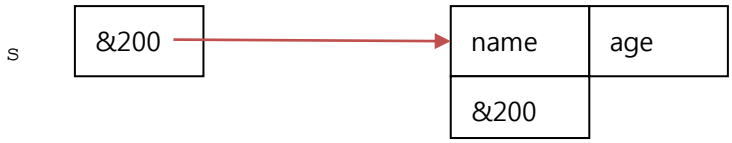
클래스의 인스턴스 (객체)를 만들려면 **new** 키워드를 사용한다.

```
클래스 명 변수 명 = new 클래스명 ( 생성자 매개 변수 ) ;
```

```
클래스 이름  인스턴스 변수명;
인스턴스 변수 명 = new 클래스 이름 (); - 인스턴스 생성 case 1
Ex) Person  s;
    S=new Person();

클래스 이름 인스턴스 변수 명 = new 클래스 이름 ();  -> 인스턴스 생성  case 2
 Ex) Person  s=new Person();

인스턴스 변수 이름. 메소드 이름 (); -> 메소드 호출
 Ex) s.setAddress("서울시");
인스턴스 변수 이름. 멤버 변수 이름; -> 변수 호출
```



인스턴스화를 s=new Person();으로 선언하게 되면 자유영역인 heap메모리에 Person클래스가 가진 멤버변수의 크기만큼 메모리 할당이 되어진다. 메모리 할당의 주소가 만일에 &200번지라면 대입연산자를 이용하여 s에 메모리 생성된 주소가 대입되어 참조관계가 된다. 참조란 상대방의 시작주소를 받아 그 주소를 통해 CRUD(Create, Read, Update, Delete)하는 관계를 말한다.

집에서 브라우저로 메일 서버의 메일을 확인할 때 메일서버의 주소를 http://www.000.000 으로 입력하게 되면 난 메일서버의 주소를 가지고 있고 그 주소로 인하여 메일을 참조하는 것처럼 클래스의 변수 또한 객체생성시 주소를 받아 다른 영역이지만 참조 관계를 맺을 수 있다.

>> 클래스의 멤버 변수와 멤버 메소드 게터와 세터 (getter & setter)

클래스의 안 메소드 밖에서 선언 된 변수는 멤버 변수(필드)이라고 하며 이 변수는 해당 클래스의 메서드는 제한 없이 볼 수 있다. 해당 클래스에 정의 된 메소드에서 변수를 공유 할 수 있다.

또한 다른 클래스에서 참조 할 수 있다. 또한 다른 클래스에서 접근하면 수식에 의해 액세스 제한을 할 수 있다. 즉, 한정자 액세스 제한은 클래스의 변수가 지정된 범위 이상으로 변경되지 않도록 보호 할 수 있다. 자바는 멤버 변수를 private 선언 해두면 다른 클래스에서 직접 참조 할 수 없게 되므로 클래스 개발자가 지정한 값 이외의 값으로 설정되는 것을 막을 수 있도록 한다.

이와 같이 다른 클래스에서 필드를 은폐하는 것은 캡슐화 라는 중요한 개념이 적용되며 다른 클래스에서 필드를 보호하는 것을 목적으로 한다. 멤버 변수는 런타임 제어가 해당 클래스에서 빗나가도 변수가 포함되어있는 개체와 함께 메모리에 계속 보관 된다. 범위에서 벗어나면 자동으로 메모리에서 드롭되는 지역 변수와는 크게 다른 점이다. 객체가 점유 한 메모리의 해제는 JavaVM의 garbage collector 에 의해 자동으로 실행되며 사용자가 명시 적으로 메모리를 조작하는 것은 있을 수 없다.

```
class FieldTest {
        int y;
        void method () {
                y *= 5;
                System.out.println ( "---- method () ----");
                System.out.println ( "method () y :"+ y );
        }
}
public class   MFieldTest {
        public static void main (String [] args) {
                FieldTest obj = new FieldTest ();   //   인스턴스화
                obj.y = 10;     // 필드에 액세스
                obj.method ();   // 메소드 호출
                System.out.println ( "---- main () ------");
                System.out.println ( "main () y :"+ obj.y );
        }
}
```

여기에서는 Field 클래스의 필드로 y 를 선언하고 있다. 이 변수는 Field 클래스에서 생성 된 객체 obj 가 존재하는 한 지속적으로 메모리에 유지 계속된다.

실행순서는 다음과 같다.

1. main ()실행
2. Field Test 인스턴스화 :y 암시 적 기본값은 0
3. y 에 값을 대입 :10
4. method ()실행 :y = y * 5
5. method ()에서 출력
6. main ()에서 출력

[실행결과]

---- method () ----

method () y :50

---- main () ------

main () y :50

다음은 접근 지정자를 이용하여 멤버 변수를 캡슐화로 구현해 보는 프로그램이다.

여기에 필드 변수를 직접 액세스 obj.y하고 있다. 캡슐화라는 관점에서는 필드는 private 선언 해 둔다 방법을 통해 액세스하는 편이 변수 값을 보호 할 수 있어 안전하다.

```
class Capsule {
	private int y;

	void setY(int y) {
		this.y = y;
	}

	int getY() {
		return y;
	}

	void method() {
		y *= 5;
```

```
                System.out.println("---- method () ----");
                System.out.println("method () y :" + y);
        }
}
public class CapsuleTest {
        public static void main(String[] args) {
                Capsule obj = new Capsule();
                obj.setY(10);
                obj.method();
                System.out.println("---- main () ------");
                System.out.println("main () y :" + obj.getY());
        }

}
```

[실행결과]

---- method () ----

method () y :50

---- main () ------

main () y :50

필드 y는 private 자격이 있기 때문에 다른 클래스에서 액세스를 할 수 없다. 다른 클래스에서 접근하기 위해서는 public의 메소드를 값전달 및 변경하기 위한 void형 setter와 값을 리턴하는 getter 메소드를 추가해야 한다. 다른 패키지에서 액세스를 허용하는 경우는, 이러한 메소드를 public 선언 한다. 값을 할당하는 방법을 세터(setter) 취득하는 메소드를 게터 (getter)라고 한다. 필드 name 세터는 setName () , 게터는 getName () 라고 명명한다

필드는 객체의 속성으로 메모리에 유지되는 변수이기 때문에 다른 개체에서 사용되기 위하여, 이상한 값이 설정된다는 위험이 따른다. 클래스 개발자의 책임으로 미리 이상한 값이 설정되지 않도록 논리를 짜 두는 것이 권장되지만, 이를 위해 필드를 private 선언 해 둔다. 즉, 액세스하려면 메소드를 통하도록 한다.

>> this 키워드

this란 현재 실행 중인 오브젝트를 지칭하는 연산자이다. 클래스 멤버에 default로 내장 되어 있어 각 멤버를 지칭 한다. 명시할 때는 this()라는 키워드를 이용해서 인스턴스를화 할 때 호출되는 생성자를 내부 호출 할 때와 메소드 내에서 지역변수와 멤버 변수를 구분할 때 this. 멤버로 명시 사용된다.

클래스의 멤버 변수를 10개를 선언하게 된다고 생각할 때 getter와 setter 메소드가 필요에 따라 각 10개씩 선언해서 사용하기 때문에 값전달 및 변경을 담당하는 setter 메소드의 매개 변수명을 일일이 명명하기가 불편하기 때문에 setter 메소드의 매개 변수 즉, 멤버변수에게 값을 전달하는 지역변수 setA(int a){} 처럼 int a의 변수명을 클래스의 멤버 변수명과 같이 사용하게 되면 변수명을 고민하지 않아도 된다. 단, 멤버 메서드 내에서 선언되는 변수는 지역변수가 우선순위를 갖게 되기 때문에 void seta(int a){ a =a;}로 선언하게 되면 외부에서 넘어오는 값이 멤버변수 a로 전달되지 않고 지역변수 a로 전달되는 형식을 취한다.

이때 멤버변수와 지역변수를 구분하기 위해 this.a= a;라고 this 키워드를 명시하게 된다.

```
public class Test01 {

    private int a;

    public int getA() {
            return a;
    }

    public void setA(int a) {
            this.a = a;  ----------→ 멤버 변수와 지역변수 a를 구분
    }
            public static void main(String[] args) {
              Test01 t1=new Test01();
              t1.setA(100);
              System.out.println(t1.getA());
    }

}
```

실행 결과는 100이 출력 된다. t1.setA(100);의 메소드가 호출되면 100이 int a의 지역변수에게 전달되고 this.a=a로 인해 멤버변수 a게 전달되어 100이 저장된다.
System.*out*.println(t1.getA());을 호출하게 되며 저장된 값을 리턴 받아 100이 된다.
만일 this.a의 this 키워드가 없다면 멤버 변수에게 값이 전달되지 않아 0이 출력 되어진다.

클래스도 배열을 선언해서 사용할 수 있다. Object배열이라고 한다

오브젝트 배열(Object Array)이란 선언된 클래스를 객체 생성해서 사용할 때 동적 할당한 배열의 주소를 참조 시켜 하나 이상의 객체 자료를 효율 적으로 관리 하는 것을 말하며 클래스 배열(Class array)이라고도 부른다.

```
MYJob[] m =new MYJob[]
                        {new MYJob(),
                         new MYJob(),
                         new MYJob()   };
```

->MyJob 클래스의 객체를 배열변수 m으로 선언한 형식이다.
