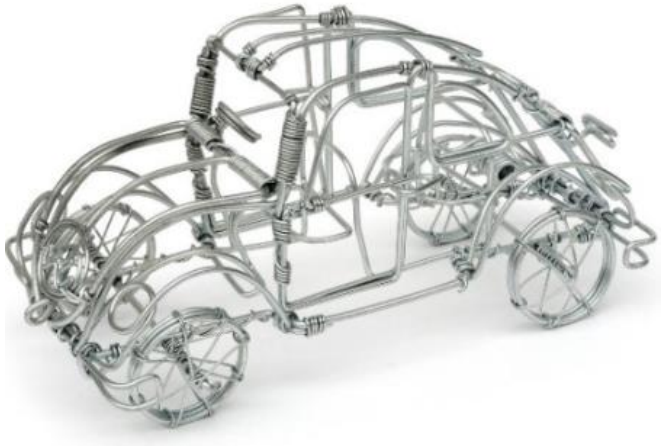
8강

JAVA_PROGRAMMING

OOP(Object Oriented Programing)

❖ 객체 지향 프로그래밍

- 현실 세계를 modeling하여 컴퓨터 프로그램으로 나타내는 개념
- 현실의 모든 사물을 컴퓨터로 표현하기 위한 프로그래밍 방식
- ex) 설계도에 맞추어 물건을 만들어 내는 방식

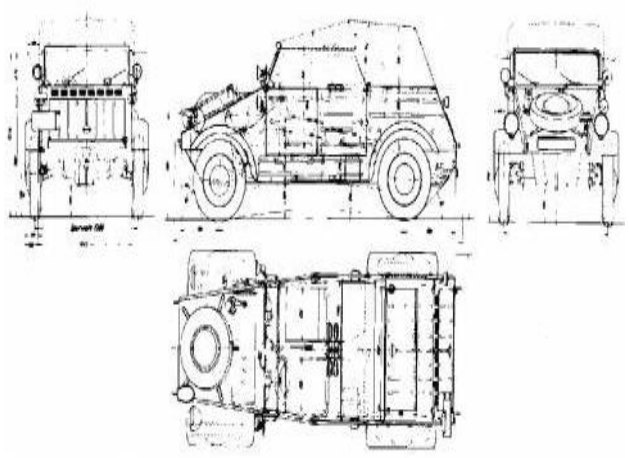


< 설계도 >

< 실체 >

OOP(Object Oriented Programing)

- ❖ 객체 지향 프로그래밍
  - 현실 세계를 modeling하여 컴퓨터 프로그램으로 나타내는 개념
  - 클래스(설계도)를 정의하여 컴퓨터 상에 인스턴스(실체)로 프로그래밍하는 것
  - ex) 시뮬레이션



< class >

< instance >

OOP 개념

❖ 객체 지향에서의 프로그래밍 개념

- 캡슐화(은닉화)
  - ✓ 관련 있는 속성(data)과 기능(method)를 하나(class)로 관리하는 것
  - ✓ class내부에서 처리되는 일은 외부에서 알 수 없음(사용 방법만 공개)
- 상속
  - ✓ 이전에 만들어 놓은 설계도에 기능을 추가하는 것
  - ✓ 코드의 재사용성 및 신뢰성 증가
- 추상화
  - ✓ 특정한 모양이 정해지지 않은 것
  - ✓ 추상적인 것을 여러 형태로 정의하여 사용이 가능한 특성
- 다형성
  - ✓ 하나의 형태가 여러 형태로 만들어 지고 사용될 수 있는 특성
  - ✓ Overloading, Overriding 등

Java Class 구성

- ✓ Member Field
- ✓ Member Method
- ✓ Constructor
- ✓ Access Modifiers
- ✓ Garbage Collection

class

- 객체지향 프로그램은 클래스를 기반으로 하여 구현
- 클래스
  - 객체를 정의해 놓은 것(설계도, 형틀 등)
- 객체
  - 클래스를 바탕으로 메모리에 실체를 구현한 것
- class의 구성 요소
  - 클래스는 객체가 가지는 속성과 기능이 들어있다.(멤버로 구성)
  - 속성은 변수, 기능은 메서드로 정의한다.

  - ex)

    TV라는 객체는 전원, 채널, 음량 등의 속성이 있다.

    TV라는 객체는 전원on/off , 채널변경, 음량변경 등의 기능이 있다.

    위의 속성과 기능을 멤버(변수, 메서드)로 정의한 것이 클래스다.

class member

- ❖ 멤버 필드
  - C++에서 이야기 하는 멤버 변수와 동일한 개념
  - 클래스 내부에 데이터를 저장할 수 있는 변수를 이야기한다.
  - ex)
    - 객체가 동작하면서 사용하게 되는 변수

- ❖ 멤버 메서드
  - C++에서 이야기하는 멤버 함수와 동일한 개념
  - 클래스 내부의 멤버필드를 사용하는 함수를 이야기 한다.
  - ex)
    - 객체가 동작하는 기능

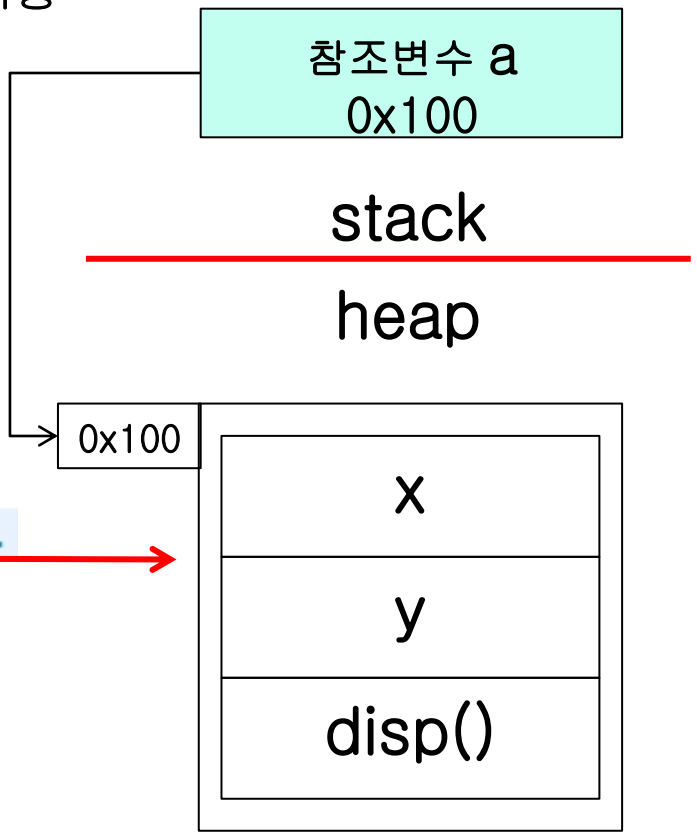
class

❖ 클래스의 구성

- class를 사용하여 객체 생성 (new 연산자)
- 객체내부의 멤버에 접근할 때는 참조연산자 .(점)을 사용

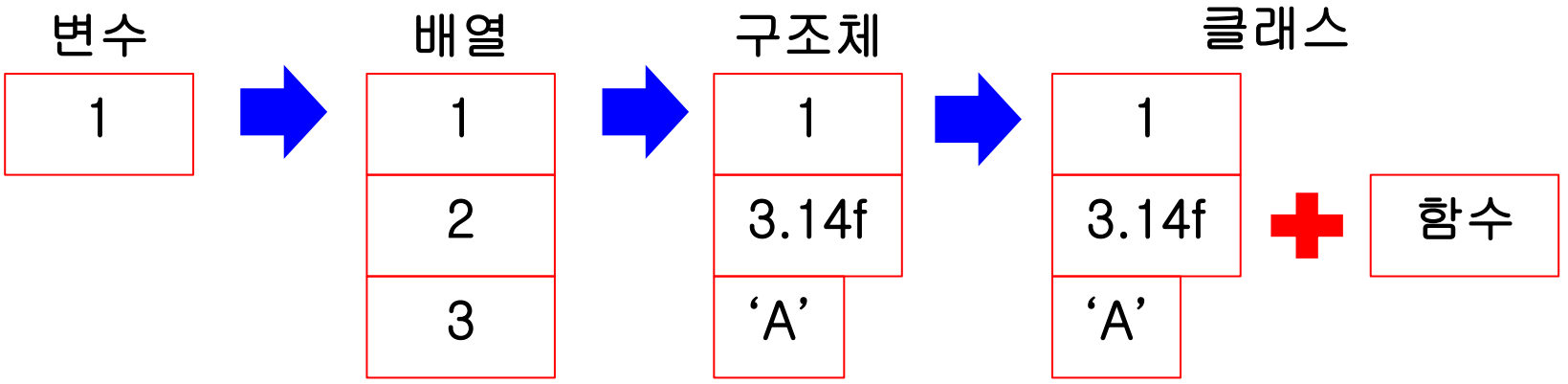
```
class A01{
    int x;  //멤버필드
    int y;  //멤버필드
    void disp(){    //멤버 메소드
        System.out.println("["+x+","+y+"]");
    }
}
public class test {
    public static void main(String []ar){
        A01 a = new A01(); //객체생성 A01()이 생성자다.
        a.disp(); //멤버 메소드 호출
        a.x = 10; //멤버필드 값 변경
        a.y = 20; //멤버필드 값 변경
        a.disp(); //멤버 메소드 호출

    }
}
```



class

- ❖ 변수
  - ▪ 하나의 데이터를 저장할 수 있는 공간
- ❖ 배열
  - ▪ 동일한 형태의 데이터를 저장할 수 있는 연속된 공간
- ❖ 구조체
  - ▪ 서로 다른 형태의 데이터를 저장할 수 있는 연속된 공간(사용자 정의 자료형)
- ❖ 클래스
  - ▪ 서로 다른 형태의 데이터를 저장하는 공간과 기능이 결합된 형태(구조체 + 함수)



class 예제

- TV 클래스를 정의
- 속성과 기능을 고려

TV의 속성(data)
전원
채널
볼륨
..
속성+n

+

TV의 기능(method)
전원버튼( )
채널변경( )
볼륨변경( )
..
기능+n

=

class 예제

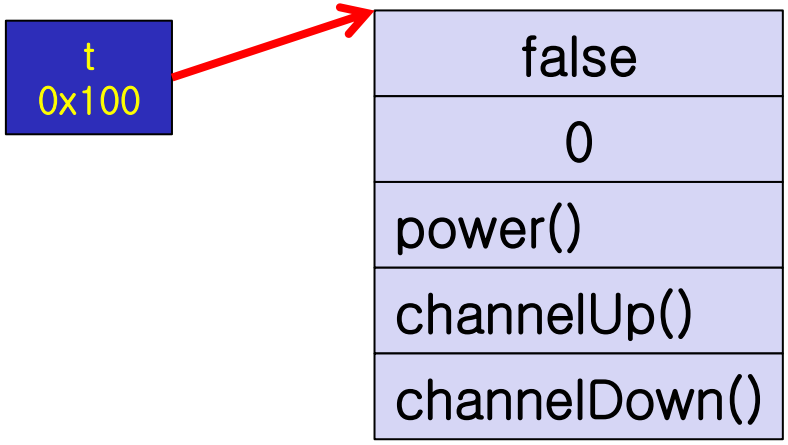
❖ class & instance

```
public static void main(String[] args){
    TV t;     //참조변수
    t = new TV();  //객체 생성
    t.power();     //기능 사용
    t.channelUp();
}
```

```
class TV{
    boolean power;    //전원
    int channel;      //채널

    void power() { power = !power; }
    void channelUp() { channel++; }
    void channelDown() { channel--; }
}
```

t = new TV();



class 예제(TV)

```
class TV{
    boolean power;//전원
    int channel;//채널
    int volume;//볼륨
    void power() { power = !power; }
    void channelUp() { channel++; }
    void channelDown() {channel--; }
    void volumeUp() { volume++; }
    void volumeDown() {volume--; }
    void disp(){
        if(power)                    System.out.print("전원 : 켜짐 ");
        else                         System.out.print("전원 : 꺼짐 ");
        System.out.print(" 채널 : "+channel);
        System.out.println(" 볼륨 : "+volume);
    }
}
```

```
public class test {
    public static void main(String[] args){
        TV t;
        t = new TV();
        t.disp();
        t.power();
        t.channelUp();
        t.volumeDown();
        t.disp();
    }
}
```

Quiz

❖ 앞서 보았던 TV를 클래스를 사용하는 리모컨을 직접 정의해 보기

- 리모컨의 기능과 속성을 생각하여 분류
  - ex) 속성 : 전원, 채널, 볼륨 등(이 때 TV를 참조해야 한다는 것을 이해)
    기능 : 전원, 채널변경, 볼륨변경 등
- 리모컨의 속성(값)은 멤버 필드(변수)로, 기능은 멤버 메서드(함수)로 정의
- 작성 후 main()에서 객체를 생성하여 기능 테스트(리모컨이 TV를 사용)

- 기타 형식 자유

  리모컨을 사용하여 TV의 기능을 동작시킨다는 것을 클래스로 작성하여 객체로 표현

변수(필드)

❖ 선언 위치에 따른 변수의 종류

```
class exam{
    int a;                  //인스턴스 변수
    static int b;           //클래스 변수
    void method(){
        int c = 10;         //지역변수
        System.out.println("a : "+a+", b : "+b+", c : "+c);
    }
}
```

변수의 종류	선언 위치	생성 시기
클래스 변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수		인스턴스 생성 시
지역 변수	메서드 영역	변수 선언문 수행 시

변수(필드)의 종류

❖ 선언 위치에 따른 변수의 종류

✓ **인스턴스 변수**(객체 생성 시 할당 – 참조변수가 없으면 소멸)

- 인스턴스가 생성 될 때 마다 만들어지는 변수
- 인스턴스 생성 후 참조 변수명으로 접근

✓ **클래스 변수**(클래스가 로딩될 때 할당 – 프로그램종료 시 소멸)

- 동일한 클래스의 인스턴스들이 공유하는 변수
- 인스턴스 생성 없이 클래스 명으로 접근 가능

✓ **지역 변수**(블록 안에서 생성 – 블록 끝나면 소멸)

- 메서드 내에 선언된 변수
- 블록 내부에서만 접근 가능

변수의 종류	선언 위치	생성 시기
클래스 변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수		인스턴스 생성 시
지역 변수	메서드 영역	변수 선언문 수행 시

member method

❖ instance method

- 인스턴스(객체) 생성 후 참조 변수명으로 접근하여 호출
- 인스턴스(객체)의 속성과 기능에 관련 된 메서드

❖ class method(static 메서드)

- 인스턴스(객체) 생성 없이 클래스명으로 접근하여 호출
- 멤버의 속성과 기능에 관련 없는 메서드(인스턴스 변수,함수 사용 불가)
- 인스턴스 변수를 사용하지 않는 메서드는 static 키워드 고려

instance method

- ❖ 예제
  - 카운트 기능을 가진 클래스

- ❖ 인스턴스 메서드
  - void init()
  - void increment()
  - int getCount()

- ➢ 인스턴스 생성 후 사용 가능
- ➢ 인스턴스 내부의 값을 사용하는 기능

```
class Counter {
    int count;  //인스턴스 변수

    void init() {
        count = 0;  //초기 값 지정
    }

    void increment() {
        count++;    //카운트 기능 메서드
    }

    int getCount() {
        return count;   //값 반환 메서드
    }
}
```

```
public class test {
    public static void main(String[] args) {
        Counter cnt = new Counter();    //객체생성
        cnt.init();                 //초기 값 지정
        cnt.increment();            //카운트 기능 사용
        System.out.println(cnt.getCount()); //카운트 정보 조회
    }
}
```

Constructor(생성자)

❖ 생성자 method

- 클래스명과 동일
- 멤버 필드 값 초기화
- 결과형 리턴 값이 없다.
- 생성자 오버로딩

★생성자 실습 기반코드★

```
class exam{
    int a;
    int b;
    void disp(){
        System.out.println("a : "+a+", b : "+b);
    }
}
public class test {
    public static void main(String[] args){
        exam ex1;        //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam();//디폴트 생성자 호출
        ex1.disp();      //ex1객체 내부의 disp()함수 호출
    }
}
```

- 객체 생성 시 반드시 하나의 생성자 호출

예제

❖ 생성자 실습테스트하기 기반 소스

```
class exam{
    int a;
    int b;
    void disp(){
        System.out.println("a : "+a+", b : "+b);
    }
}
public class test {
    public static void main(String[] args){
        exam ex1;          //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam();//디폴트 생성자 호출
        ex1.disp();        //ex1객체 내부의 disp()함수 호출
    }
}
```

생성자 예제

❖ 생성자가 단 하나도 정의되어 있지 않다면 디폴트 생성자 호출

```
class exam{
    int a, b;
    //exam(){}  //디폴프 생성자가 생략되어 있다.
}
public class test {
    public static void main(String[] args){
        exam ex1;                   //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam();           //디폴트 생성자 호출
        ex1.a = 10;
        ex1.b = 20;
        System.out.println("ex1.a : "+ex1.a+", ex1.b : "+ex1.b);
    }
}
```

생성자 예제

❖ 객체가 생성될 때 초기 값을 넣어줄 때 사용하면 편리

```
class exam{
    int a, b;
    //exam(){}  //디폴트 생성자
    exam(int x, int y){ //생성자 오버로딩
        a = x; b = y;
    }
}
public class test {
    public static void main(String[] args){
        exam ex1;                   //exam을 참조할 수 있는 참조변수 ex1생성
        //ex1 = new exam();         //디폴트 생성자 호출형식  에러!!!!
        ex1 = new exam(10,20);      //오버로딩 된 생성자 호출
        System.out.println("ex1.a : "+ex1.a+", ex1.b : "+ex1.b);
    }
}
```

예제

❖ 카운트 기능을 가진 클래스 수정

- init()메서드로 멤버필드의 값을 0으로 초기화 하고 있다.
- 생성자를 이용한다면 이러한 초기 값 지정을 신경 쓰지 않아도 된다.

```
class Counter {
    int count;  //인스턴스 변수

    void init() {
        count = 0;  //초기 값 지정
    }

    void increment() {
        count++;    //카운트 기능 메서드
    }

    int getCount() {
        return count;   //값 반환 메서드
    }
}
```

```
class Counter {
    int count;  //인스턴스 변수

    void Counter() {    //생성자
        count = 0;  //초기 값 지정
    }

    void increment() {
        count++;    //카운트 기능 메서드
    }

    int getCount() {
        return count;   //값 반환 메서드
    }
```

```
public class test {
    public static void main(String[] args) {
        Counter cnt = new Counter();    //객체생성
        //cnt.init();           //필요없음 생성자가 대신 함
        cnt.increment();        //카운트 기능 사용
        System.out.println(cnt.getCount()); //카운트 정보 조회
    }
}
```

예제

- 카운트 기능을 가진 클래스 수정
  - 생성자를 오버로딩하기
  - 무조건 카운트는 0부터 시작한다.
  - 임의의 수를 전달받아 생성자가 초기값을 지정할 수 있도록 수정

this

- this는 모든 클래스의 멤버 메서드의 0번째 참조매개변수 이다.
- 어떤 객체가 멤버 메서드를 호출했는지 참조하는 변수이다.
- 인스턴스 생성 시 this()생성자로 다른 생성자를 호출할 수도 있다.
  - 예제 (원치 않는 결과)

```
class exam{
    int a, b;
    exam(int a, int b){ //생성자 오버로딩
        a = a; b = b;
    }
    void disp(){
        System.out.println(a+b);
    }
}
public class test {
    public static void main(String[] args){
        exam ex1;                   //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam(10,20);  //오버로딩 된 생성자 호출
        ex1.disp();
    }
}
```

※이름이 중복 되므로 값이 멤버필드에 대입되지 않음

this 참조변수

❖ 멤버필드와 지역변수(매개변수)의 이름 중복을 해결할 수 있다.

```
class exam{
    int a, b;
    exam(int a, int b){ //생성자 오버로딩
        this.a = a; this.b = b;
    }
    void disp(){
        System.out.println(a+b);
    }
}
public class test {
    public static void main(String[] args){
        exam ex1;                   //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam(10,20);  //오버로딩 된 생성자 호출
        ex1.disp();
    }
}
```

this() 생성자

❖ this() 생성자의 활용

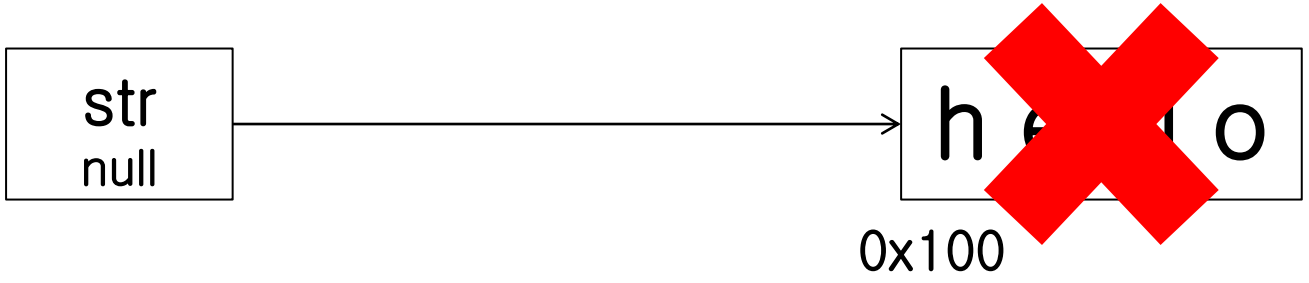
```
class exam{
    int a, b, c, d, e, f, g;
    exam(){ //생성자 오버로딩
        a = 1; b = 2; c = 3; d = 4; e = 5; f = 6; g = 7;
    }
    exam(int x){     //오버로딩한 생성자가 전달받은 값으로
        //this();    //멤버 필드 d의 값만을 초기화 하려는 경우
        d = x;       //다른 멤버필드의 값은 모두 0으로 초기화 된다.
    }
    //이런 경우 this();를 사용하면 먼저 전달인자가 없는 생성자가 호출 되고
    //그 다음 d에 x의 값을 대입하는 순서로 동작이 된다.
    //이 때 this();의 호출은 가장 먼저 실행 되어야만 한다.
    //주석을 풀고 실행해 보자.
}
public class test {
    public static void main(String[] args){
        exam ex1;              //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam(10); //오버로딩 된 생성자 호출
    }
}
```

Garbage Collection

- C/C++에서는 동적으로 메모리를 할당하는 경우 반드시 사용 후 초기화 필요
- Java에서는 heap에서 참조하지 못하는 메모리는 JVM이 자동으로 삭제
- heap영역의 관리를 JVM이 처리(Gabage Collector)

  - String str = new String("hello"); //heap영역에 메모리할당

  - str = null //참조 변수를 변경
  - 참조하지 못하는 공간을 garbage collector가 삭제



접근 제한자

❖ 클래스 내의 멤버에 접근을 제한하기 위한 예약어
- 클래스 내부로의 접근을 제어하고자 하는 목적으로 사용
- 객체를 가지고 접근을 하더라도 모두 개방적일 경우 원치 않는 값의 변경에 따른 에러 등의 문제를 해결하기 위해서 접근 제한자를 만들게 되었다.
- 어떠한 클래스이건 접근 제한자를 가진다.
- 클래스, 생성자, 메서드, 필드에 전부 적용(지역변수는 제외)
- private과 protected는 클래스와 생성자에는 잘 사용하지 않음을 권장
- 일반적으로 멤버 필드는 private , 멤버 메서드는 public으로 선언

✓ 접근 제한자

접근 제한자	접근 허용 범위
private	단일 클래스 내부
public	모든 영역에서 접근 허용
protected	단일 클래스, 동일파일, 동일폴더,상속관계만 접근 허용
package	단일 클래스, 동일 파일, 동일 폴더 내에서만 접근 허용

예제

❖ public, private, protected 접근 예제

```
class exam{
    public int pu;        //누구나 접근 가능
    protected int pro;    //단일 클래스 파일,폴더,클래스에서 접근 가능
    private int pr;       //단일 클래스에서만 접근 가능
    void method(){
        pu = 1; pr = 2; pro = 3;
    }
}
public class test {
    public static void main(String[] args){
        exam ex1 = new exam();
        ex1.pr = 10;      //ex1객체의 pr  접근 불가 (private)
                          //ex1객체의 private 멤버에 접근 에러!!
        ex1.pu = 20;      //ex1객체의 pu  접근 가능 (public)
        ex1.pro = 30;     //ex1객체의 pro 접근 가능 (protected)
    }
}
```

getter() / setter()

- private인 멤버필드에 접근하려면 그 값에 접근 할 수 있는 클래스 내부에 멤버 메서드를 정의해야 한다.
  - ✓ setter(멤버 필드 값을 세팅하는 메소드)
  - ✓ getter(멤버 필드 값을 얻어오는 메소드)

```
class A01{
    private int x;  //멤버필드
    private int y;  //멤버필드
    void Set_xy(int a,int b){
        x = a;
        y = b;

    }
    int Get_xy(){
        return x+y;
    }
}
```

```
public class test {
    public static void main(String []ar){
        A01 a = new A01(); //오버로딩 생성자 호출
        int ret;
        //a.y = 20; //에러가 난다.외부에서 a클래스내부로 접근개념
        System.out.println("결과는 "+a.Get_xy());//private 값 출력
        a.Set_xy(100, 200); //private 멤버 값 변경 함수 호출
        ret = a.Get_xy();   //private 멤버 값 얻어오는 함수 호출
        System.out.println("결과는 "+ret); //private 값 출력
    }
}
```

지정 예약어

- ❖ 필드에 대한 지정 예약어
- ❖ 접근 제한자 뒤에 오는 키워드
- ❖ static은 클래스 안의 멤버만 적용 가능

예약어	설명
static	동일한 클래스 모두 공유하는 필드(클래스명으로 접근 가능)
final	상수 필드(c언어의 const)
static final	static + final 혼합된 형태
transient	임시 메모리 필드(데이터 전송 시 사용)

지정 예약어

- 메서드에 대한 지정 예약어
- 접근 제한자 뒤에 오는 키워드
- 메서드가 사용하는 필드에 대한 제어

예약어	설명
static	static필드의 값을 처리.static필드 및 메서드만 포함가능. (클래스 명으로 접근 가능)
final	상속 시 오버라이딩을 하지 못하게 만드는 키워드
static final	static + final 혼합형태
abstract	내부에 정의가 없는 추상 메서드를 선언할 경우 사용
sysnchronized	특정 메서드에서 여러 스레드가 동작할 경우 중복 실행을 막기 위한 키워드 (동기화)
native	타 언어로 만들어진 코드를 Java내부에서 사용할 때

static 필드

- ❖ 동일한 클래스 또는 인스턴스가 공유하는 필드
- ❖ 클래스로딩 시 메모리에 생성
- ➢ 접근 제한자 static 자료형(클래스형) 필드명 [= 값];
- ➢ this 사용 불가
- ➢ 지역변수에는 사용불가

- ➢ 초기화 방법
  - ✓ 형식1 (클래스 내부에서 초기화)

```
static {
    내용...
}
```

  - ✓ 형식2 (직접 초기화)

```
private static int a = 100;
```

```java
class point{
    private static int x; //static 멤버 필드
    //private static int x = 10;    //직접 초기화 가능
    public int y;
    static{     //static 블럭에서 초기화
        x = 10;
    }
    point(){
        y = 20;
    }
    void disp(){
        System.out.println("x : "+x+", y : "+y);
    }
}
public class test {
    public static void main(String[] args){
        point pt = new point();
        pt.disp();
    }
}
```

static 메서드

- static 필드의 값을 처리하기 위한 메서드
- 클래스로딩 시 메모리에 생성

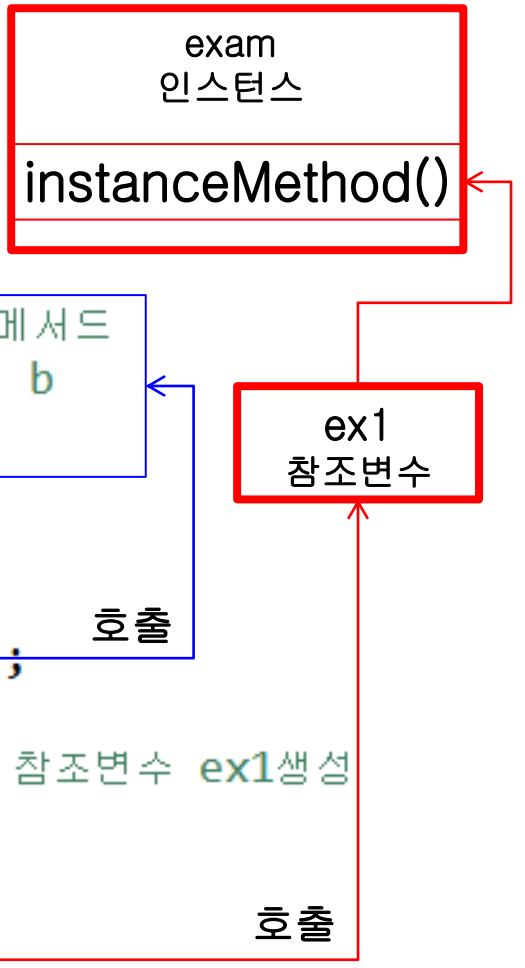
```
접근 제한자 static 반환 자료형 메서드명 (매개변수들) [throws 예외클래스]{
        내용...
}
```

- 클래스명 또는 객체명으로 접근이 가능하다.
- this가 없다.
- 일반 멤버 필드,멤버 메서드를 사용할 수 없다.
  (일반 멤버들은 프로그램 실행 후 생성이 안되었을 수도 있으므로)
- 인스턴스 멤버를 사용하지 않는 메서드는 static을 고려한다.

static(class) method

❖ 클래스(static) 메서드와 인스턴스 메서드

```
class exam{
    int a, b;
    int instanceMethod(){    //인스턴스 메서드
        return a + b;    //인스턴스 변수 a, b
    }
    static int classMethod(int a, int b){    //클래스 메서드
        return a + b;    //인스턴스 변수가 아닌 지역변수 a, b
    }
}
public class test {
    public static void main(String[] args){
        System.out.println(exam.classMethod(10, 20));
                                    //클래스 메서드 호출
        exam ex1;                   //exam을 참조할 수 있는 참조변수 ex1생성
        ex1 = new exam();           //디폴트 생성자 호출
        ex1.instanceMethod();       //인스턴스 메서드 호출
    }
}
```



final

- 멤버 필드에 지정 시
- 상수 변수를 위한 final 지정 예약어
- final 필드
  - 상수 값을 저장 하는 지정 예약어
  - 지역변수나 멤버 필드 모두 가능
- final 멤버 메서드
  - 오버라이딩 할 수 없는 메서드
- final 클래스
  - 상속할 수 없는 클래스
  - String, Math 클래스는 final클래스

```
final class fin {              //상속되지 못하는 클래스
    final int mark = 0;        //멤버필드 값을 변경 못함
    final int getCount() {     //오버라이딩 불가능
        final int a = 10;      //상수변수 10으로 고정
        mark += a++;           //멤버,지역변수 둘 다값 변경 못함
        return mark;           //값 반환 메서드
    }
}
```

static final

- static final
  - static의 특성과 final의 특성 모두를 가진다.

- 멤버 필드에 지정 시
  - 상수면서 클래스명으로 접근가능
  - 지역변수에는 static사용 불가능
- 멤버 메서드에 지정 시
  - 최종, 마지막 메서드라는 의미
  - 상속 시 오버라이드 되지 못하는 메서드(final)
- 클래스에 지정 시
  - 클래스 명으로 접근 가능
  - 상속 되지 못하는 클래스(확장 불가)

Quiz

- Person 클래스를 정의하시오.
  - ✓ 아래 멤버변수 선언, 생성자, getter, setter 등을 선언
    - *이름*
    - *나이*

- 만든 클래스를 이용하여 회원 5명의 정보를 저장하는 프로그램 작성
  - ✓ 배열을 사용하여 만든다.

    Ex) Person[] stud = new Person[5];

Quiz

- Student 클래스를 정의하시오.
  - ✓ 아래 멤버변수 선언, 생성자, getter, setter 등을 선언
    - *이름*
    - *나이*
    - *국어점수*
    - *영어점수*
    - *수학점수*
    - *총점*
    - *평균*

- 만든 클래스를 이용하여 학생 5명의 정보를 저장하는 프로그램 작성
  - ✓ 배열을 사용하여 만든다.

    Ex) Student[] stud = new Student[5];

Quiz

- Teacher 클래스를 정의하시오.
  - ✓ 아래 멤버변수 선언, 생성자, getter, setter 등을 선언
    - *이름*
    - *나이*
    - *담당 과목*

- 만든 클래스를 이용하여 3명의 정보를 저장하는 프로그램 작성
  - ✓ 배열을 사용하여 만든다.

    Ex) Teacher[] tc = new Teacher[3];

    .

    .

    .

    .

    .