
OOP
Prof. Youngchul Jung
주문식교육의 산실
영진전문대학

비 구조적 언어

작성하기 어렵다.
유지 보수도 어렵다!!!!

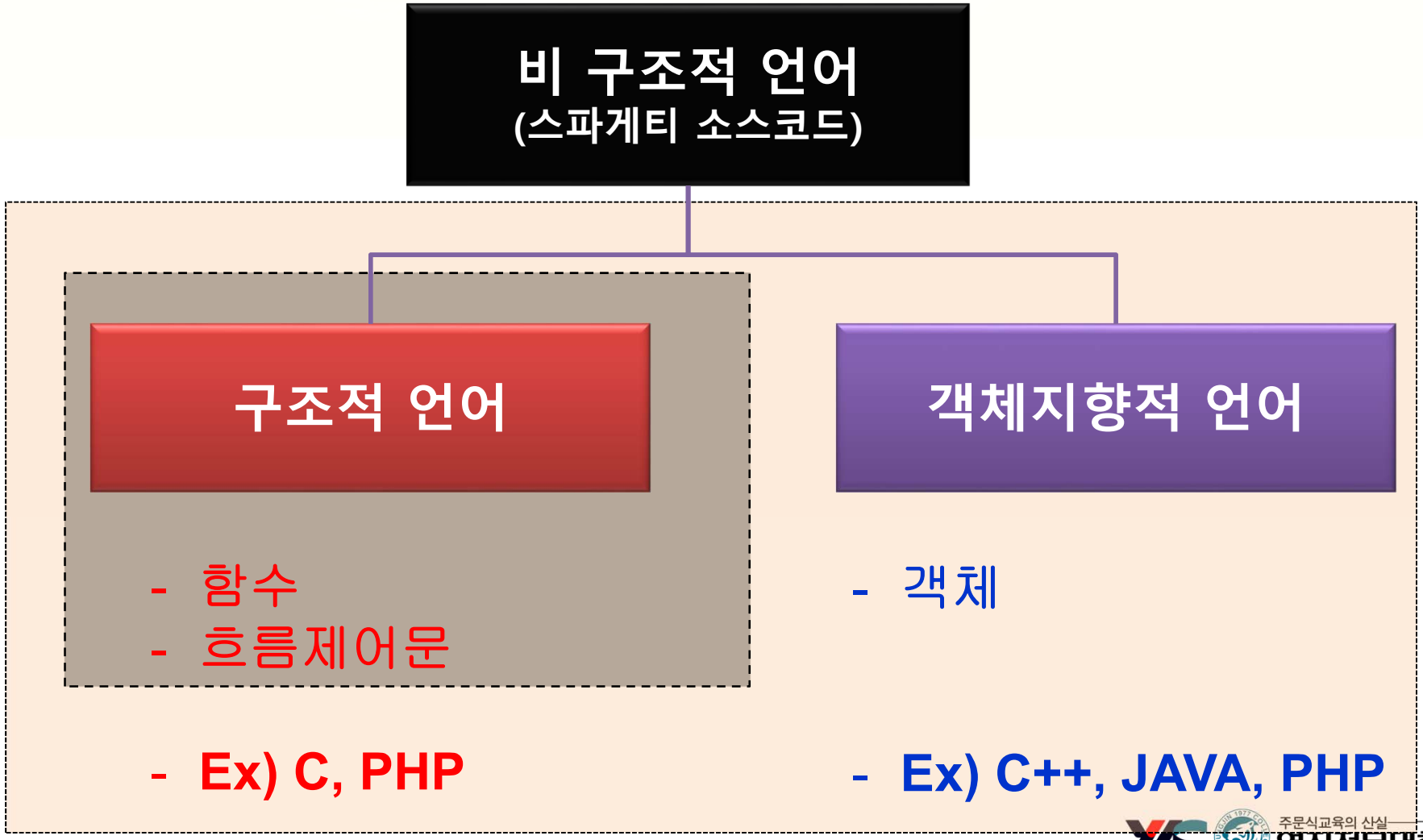
대안은?

수많은 분기문과
GOTO문으로 구성

Assembly language (어셀블리어)

주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

프로그래밍언어 분류: 구성체계



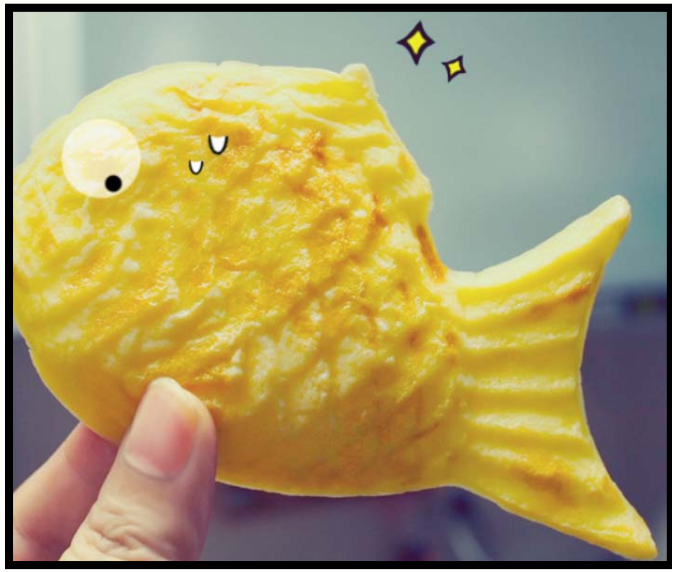
Object Oriented Programming (OOP) 구성요소

- 오브젝트 & 클래스 (Class & Object)
- 상속 (Inheritance)
- 제어자 (Modifier)
- 다형성 (Polymorphism)
- 추상클래스 (Abstract class)
- 인터페이스 (Interface)
- 예외처리 (Exception Handling)

OOP 구성요소: 클래스

- 클래스

- 오브젝트

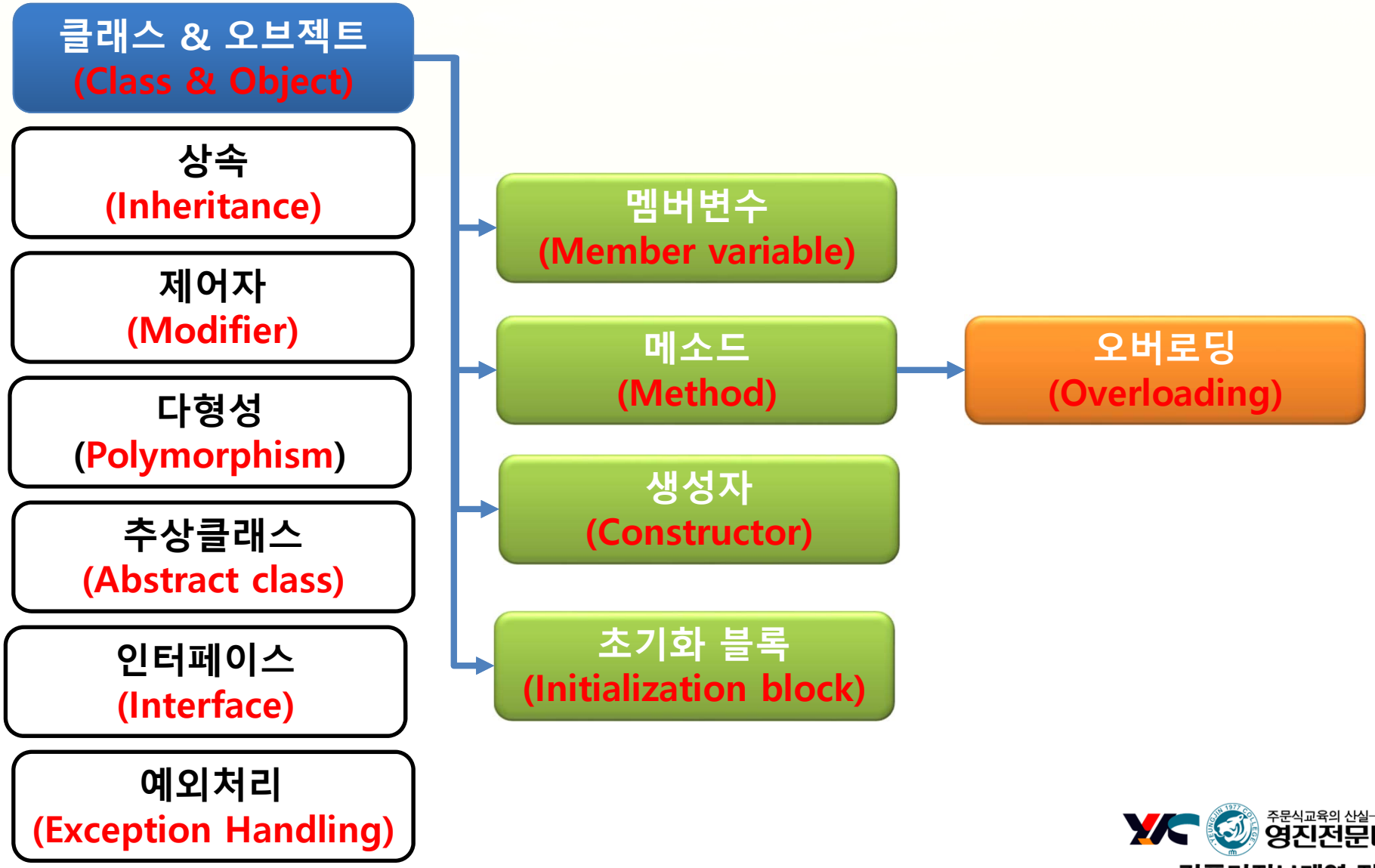


```
class FishCake {
        int age;

        void say_age() {
                System.out.println(age);
                }
}
```

```
FishCake Bungbung = new FishCake();
```

영진전문대학
컴퓨터정보계열 정영철

OOP 구성요소: 클래스 (1)



OOP 구성요소: 클래스 (2)

```
class MyFirstClass {                    //클래스 정의

    String   StrMyName;                 //멤버변수

    void printMyName()
    {
        System.out.println(StrMyName);  //메소드
    }
}
```

OOP 구성요소: 클래스 (3)

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

    MyFirstClass MyClass = new MyFirstClass();

      MyClass.StrMyName = "태연짱~";

      MyClass.printMyName();
      }

}
```

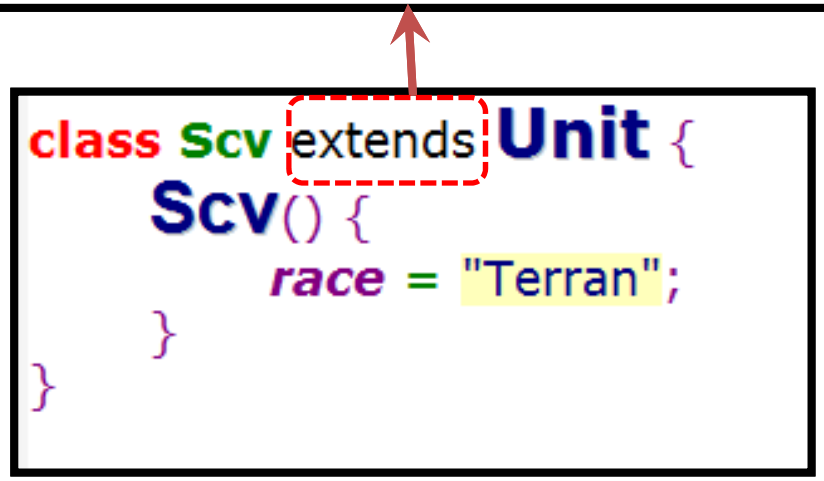
```
class MyFirstClass {

    String  StrMyName;

    void printMyName()
    {
        System.out.println(StrMyName);
    }
}
```

OOP 구성요소: 상속 (1)


클래스 & 오브젝트
(Class & Object)
상속
(Inheritance)
제어자
(Modifier)
다형성
(Polymorphism)
추상클래스
(Abstract class)
인터페이스
(Interface)
예외처리
(Exception Handling)
오버라이딩
(Overriding)

OOP 구성요소: 상속 (2)

```
class Unit {
        static int  unit_id;
        String          race;

        Unit () {
                ++unit_id;
        }

        public String toString() {
                String str = "My unit num: " + unit_id + "\nRace: " + race;
                return str;
        }
}
```



OOP 구성요소: 상속 (3)

```
class MainClass {

    public static void main(String[] args) {
        Scv scv_1 = new Scv();

        System.out.println(scv_1);
    }
}
```

```
My unit num: 1
Race: Terran
```

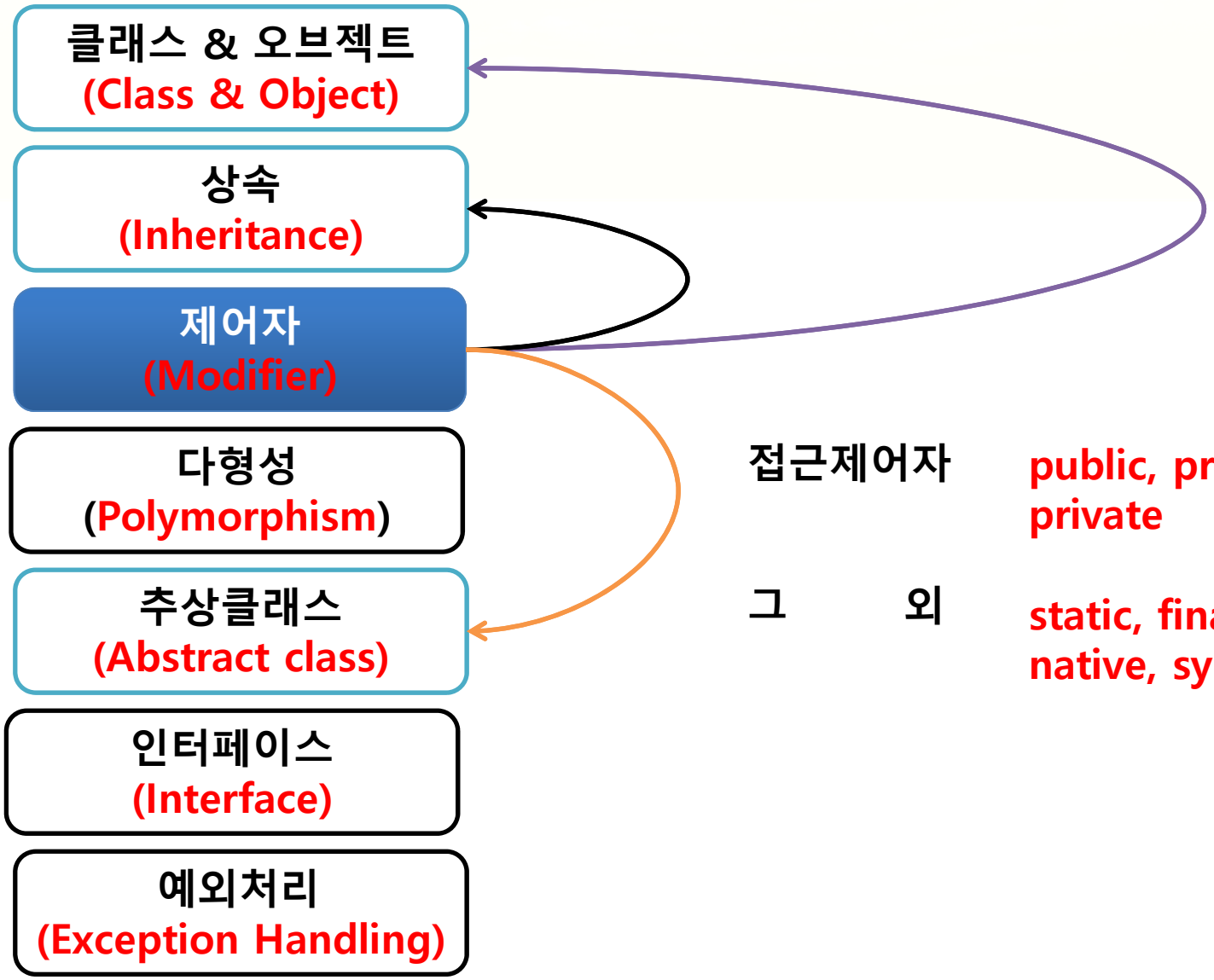
OOP 구성요소: 상속 (4)

```
class Unit {
        static int  unit_id;
        String          race;

        Unit () {
                ++unit_id;
        }

        public String toString() {
                String str = "My unit num: " + unit_id + "\nRace: " + race;
                return str;
        }
}
```

```
class Scv extends Unit {
        Scv() {
                race = "Terran";
        }
}
```

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

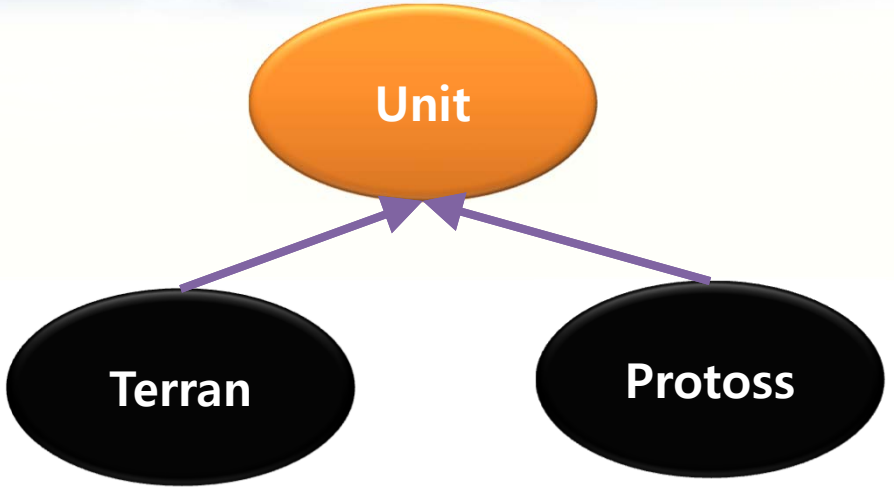
OOP 구성요소: 제어자



접근제어자	public, protected, default, private
그 외	static, final, abstract, native, synchronized

OOP 구성요소: 다형성

- 클래스 & 오브젝트 (Class & Object)
- 상속 (Inheritance)
- 제어자 (Modifier)
- **다형성 (Polymorphism)**
- 추상클래스 (Abstract class)
- 인터페이스 (Interface)
- 예외처리 (Exception Handling)



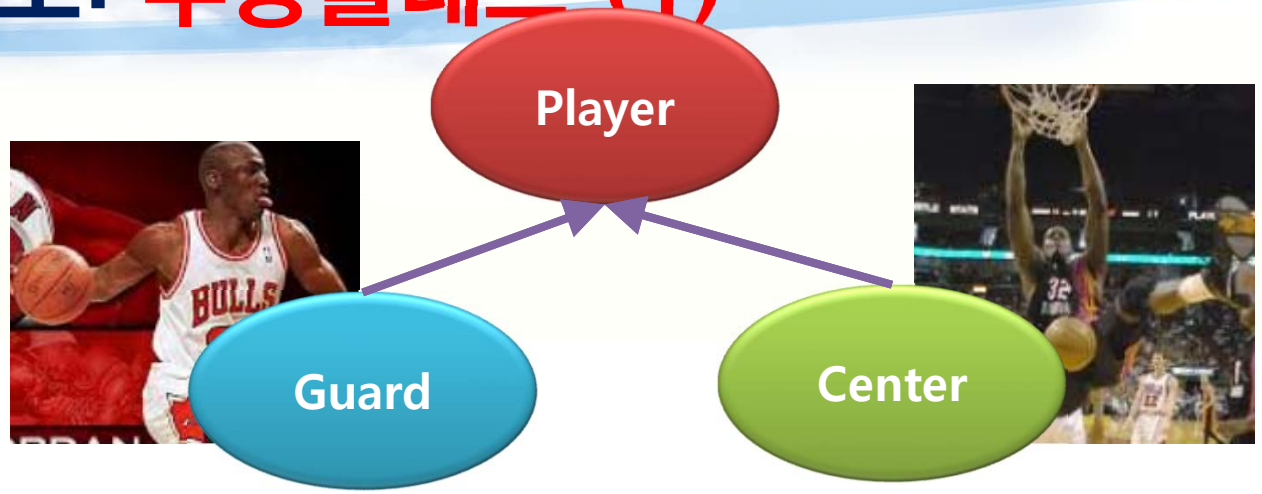

```
Terran myTerran = new Protoss();
```

```
Unit unit_t = new Terran();
Unit unit_p = new Protoss();
```

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

OOP 구성요소: 추상클래스 (1)

- 클래스 & 오브젝트 (Class & Object)
- 상속 (Inheritance)
- 제어자 (Modifier)
- 다형성 (Polymorphism)
- **추상클래스 (Abstract class)**
- 인터페이스 (Interface)
- 예외처리 (Exception Handling)




```
abstract class Player {
        abstract void shoot();
}

class Center extends Player {
        void shoot() {
                System.out.println("Dunk shoot! ! ");
        }
}

class Guard extends Player {
        void shoot() {
                System.out.println("3 Points shoot! ! ");
        }
}
```

주문식교육의 산실
영진전문대학
퓨터정보계열 정영철

OOP 구성요소: 추상클래스 (2)

```
class MainClass {
    public static void main(String[] args) {
        Guard    g_1 = new Guard();
        Center   c_1 = new Center();

        g_1.shoot();
        c_1.shoot();
    }
}
```

```
3 Points shoot!!
Dunk shoot!!
```

```
class MainClass {
    public static void main(String[] args) {
        Player   plyaer_list[] = new Player[2];

        plyaer_list[0] = new Center();
        plyaer_list[1] = new Guard();

        plyaer_list[0].shoot();
        plyaer_list[1].shoot();
    }
}
```

```
Dunk shoot!!
3 Points shoot!!
```

OOP 구성요소: 인터페이스

- 클래스 & 오브젝트 (Class & Object)
- 상속 (Inheritance)
- 제어자 (Modifier)
- 다형성 (Polymorphism)
- 추상클래스 (Abstract class)
- **인터페이스 (Interface)**
- 예외처리 (Exception Handling)

```
interface Attack {
    void general_attack();
}
```

```
interface Defense {
    void general_defense();
}
```

```
class SCV implements Attack, Defense {

    public void general_attack() {
        System.out.println("SCV general attack");
    }

    public void general_defense() {
        System.out.println("SCV general defense");
    }

}
```

OOP 구성요소: 예외처리

- 클래스 & 오브젝트 (Class & Object)
- 상속 (Inheritance)
- 제어자 (Modifier)
- 다형성 (Polymorphism)
- 추상클래스 (Abstract class)
- 인터페이스 (Interface)
- **예외처리 (Exception Handling)**

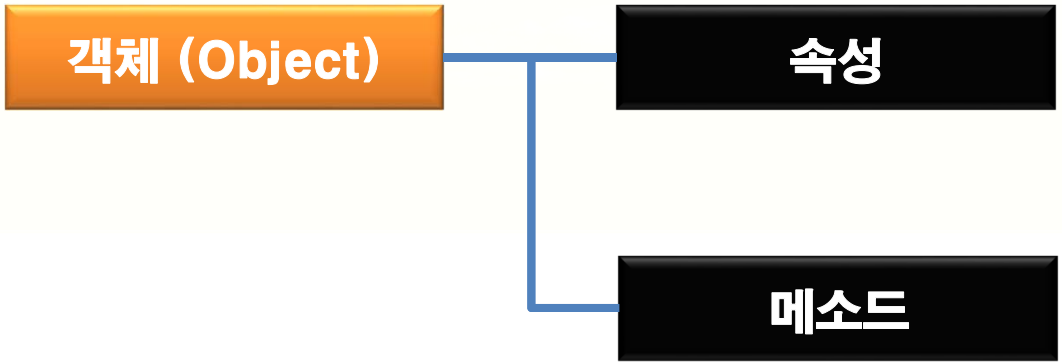
```
class MainClass {
    public static void main(String[] args) {

        try {

            System.out.println(1/0);

        } catch(ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

Object & Class

주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

What is an Object?



멤버변수

메소드

```
class MyFirstClass {                    //클래스 정의

    String  StrMyName;                                  //멤버변수

    void printMyName()
    {
        System.out.println(StrMyName);                  //메소드
    }
}
```

클래스와 객체

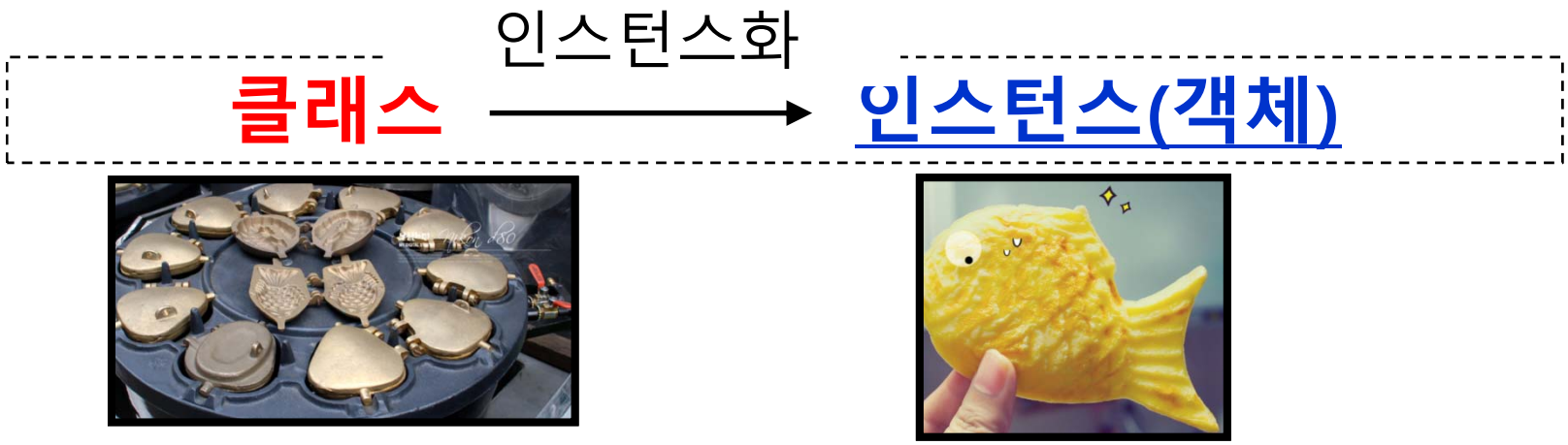
- 클래스의 정의 – 클래스란 객체를 정의해 놓은 것이다.
- 클래스의 용도 – 클래스는 객체를 생성하는데 사용된다.

- 객체의 정의 – 실제로 존재하는 것. 사물 또는 개념.

클래스	객체
제품 설계도	제품
TV설계도	TV
붕어빵기계	붕어빵

객체와 인스턴스

- **객체 ≒ 인스턴스**
  - 객체(object)는 인스턴스(instance)를 포함하는 일반적인 의미
- **인스턴스화(instantiate, 인스턴스化)**
  - 클래스로부터 인스턴스를 생성하는 것.



객체의 구성요소 – 속성과

- **객체는 속성과 기능으로 이루**
  - 객체는 속성과 기능의 집합이며 이
    의 멤버(member, 구성요소)라

- **속성은 멤버변수로, 기능은 메서드로 정의한다.**
  - 클래스를 정의할 때 객체의 속성은 변수로, 기능은 메서드로 정의한다

속성	크기, 길이, 높이, 색상, 볼륨, 채널 등
기능	켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 높이기 등

멤버변수 →
메서드 →

```
class Tv {

    String color;  // 색깔
    boolean power; // 전원상태on/off)
    int channel;   // 채널

    void power() { power = !power; } // 전원on/off
    void channelUp( channel++;)      // 채널 높이기
    void channelDown {channel--;}    // 채널 낮추기

}
```

인스턴스의 생성과 사

- **인스턴스의 생성방법**

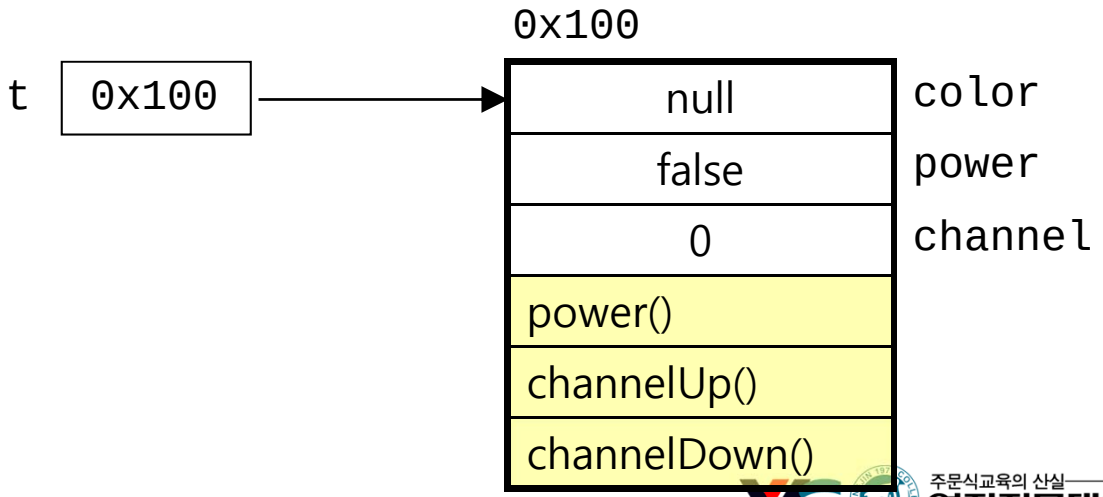
클래스명 참조변수명;

참조변수명 = new 클래스명();

```
class Tv {

    String color;  // 색깔
    boolean power; // 전원상태on/off)
    int channel;   // 채널

    void power() { power = !power; } // 전원on/off
    void channelUp( channel++;)      // 채널 높이기
    void channelDown {channel--;}    // 채널 낮추기

}
```

// 객체생성 후, 생성된 객체의
주소를 참조변수에 저장

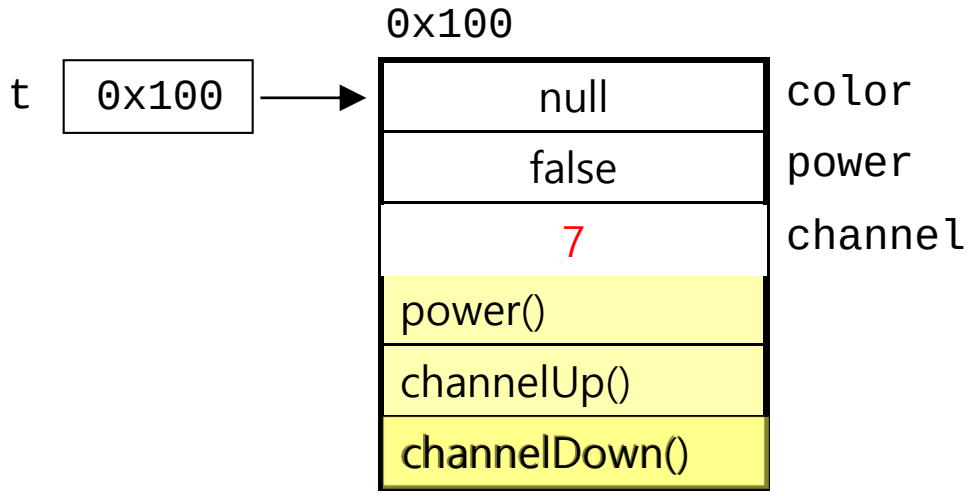
```
Tv t;
t = new Tv();
```



인스턴스의 생성과 사용(2/4)

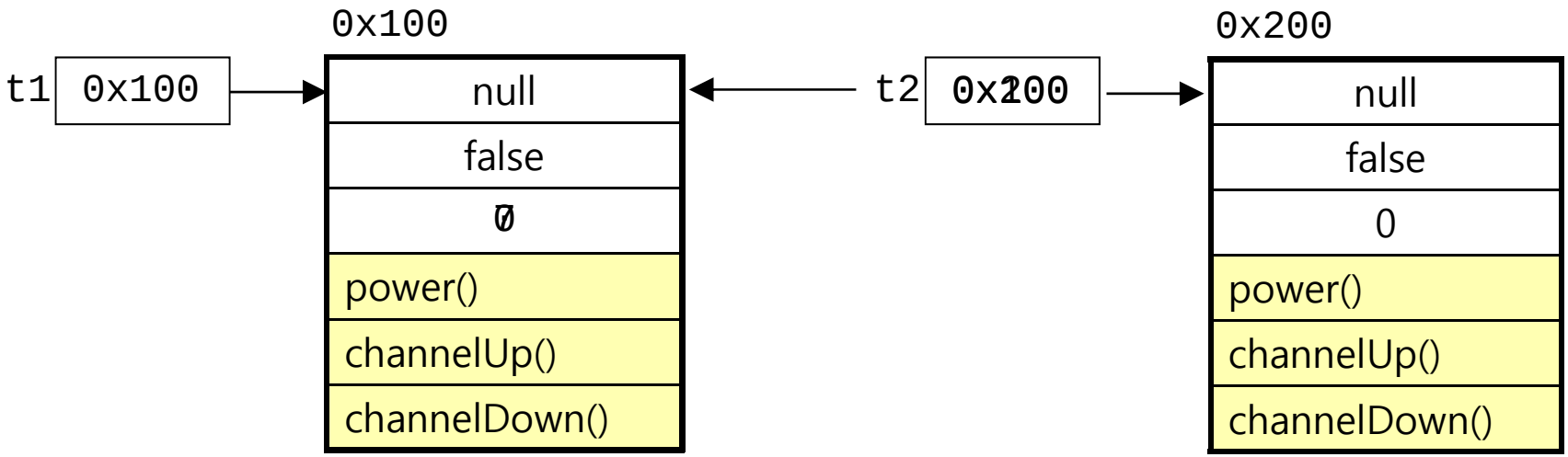
```
Tv t;
t = new Tv();
t.channel = 7;
t.channelDown();
System.out.println(t.channel);
```

```
class Tv {
    String color;  // 색깔
    boolean power; // 전원상태(on/off)
    int channel;   // 채널
    void power() { power = !power; } // 전원on/off
    void channelUp( channel++;)      // 채널 높이기
    void channelDown {channel--;}    // 채널 낮추기
}
```

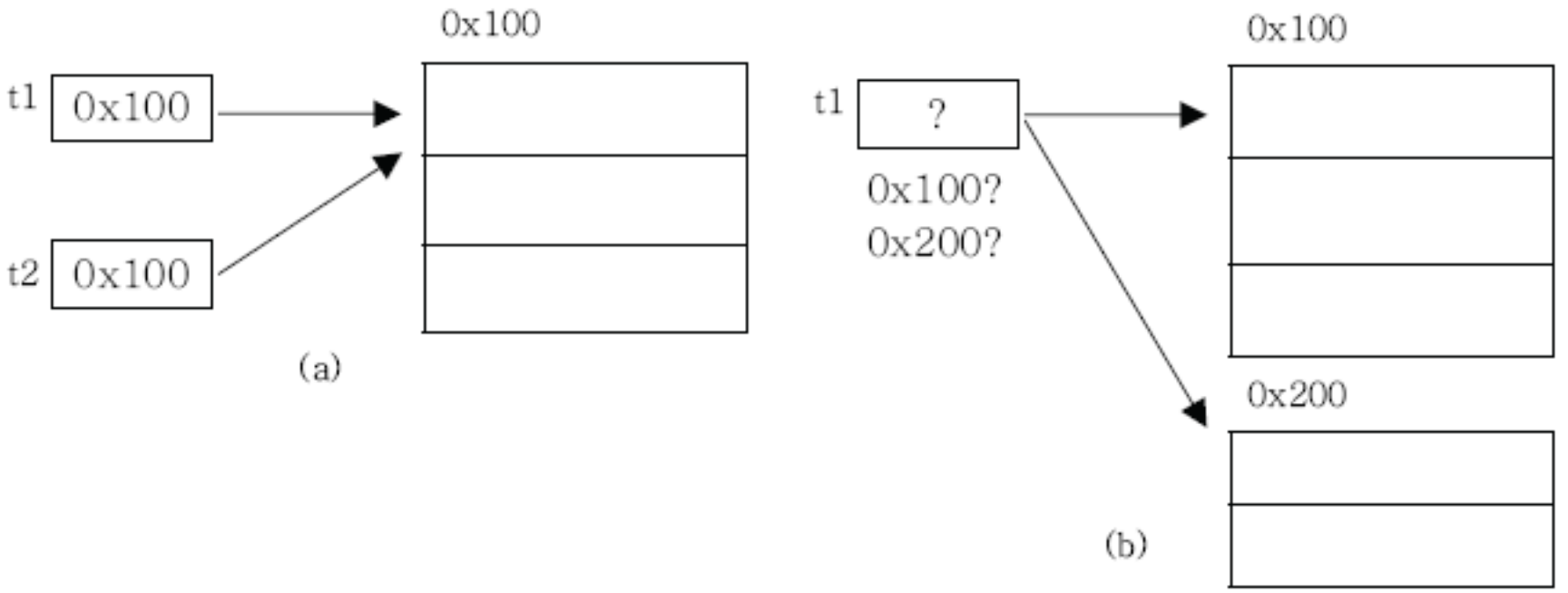


인스턴스의 생성과 사용(3/4)

```
Tv t1 = new Tv();
Tv t2 = new Tv();
t2 = t1;
t1.channel = 7;
System.out.println(t1.channel);
System.out.println(t2.channel);
```



인스턴스의 생성과 사용(4/4)



(a) 하나의 인스턴스를 여러 개의 참조변수가 가리키는 경우(가능)
(b) 여러 개의 인스턴스를 하나의 참조변수가 가리키는 경우(불가능)

[그림6-2] 참조변수와 인스턴스의 관계

연습문제: 학생성적 계산기 (1)

```
class StudentScore {
    String      studentName;

    int         korean;
    int         math;
    int         english;
    int         sum;

    double      average;

    void calculateSum()
    {
        sum = korean + math + english;
    }

    void calculateAverage()
    {
        average = sum/3.0d;
    }

    void printScore()
    {
        System.out.println(studentName + "\t" + korean + "\t" +  math + "\t" + english  + "\t" + sum + "\t" + average );
    }
} ? end StudentScore ?
```

//멤버변수

//메소드

//학생 성적 합계

//학생 성적 평균

//성적 출력하기

연습문제: 학생성적 계산기 (2)

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

		Scanner Scan = new Scanner(System.in);

		System.out.println("학생수를 입력하세요!");
		int rowNum = Scan.nextInt();

		StudentScore StdScoreList[] = new StudentScore[rowNum];

		for(int index = 0 ; index < StdScoreList.length ; index++)
		{
			StdScoreList[index] = new StudentScore();

			System.out.println("학생 이름을 입력하세요!");
			StdScoreList[index].studentName = Scan.next();

			System.out.println("국어 성적을 입력하세요!");
			StdScoreList[index].korean      = Scan.nextInt();

			System.out.println("영어 성적을 입력하세요!");
			StdScoreList[index].english     = Scan.nextInt();

			System.out.println("수학 성적을 입력하세요!");
			StdScoreList[index].math        = Scan.nextInt();

			StdScoreList[index].calculateSum();
			StdScoreList[index].calculateAverage();
		}
```

//성적입력 부분

연습문제: 학생성적 계산기 (3)

// 정렬 부분

```
        for(int base = 0 ; base < StdScoreList.length - 1 ; base++)
        {
            int          minKey = base;
            StudentScore TmpValue;

            for(int next = base + 1 ; next < StdScoreList.length ; next++)
            {
                if(StdScoreList[next].sum < StdScoreList[minKey].sum)
                    minKey = next;
            }

            TmpValue              = StdScoreList[base];
            StdScoreList[base]  = StdScoreList[minKey];
            StdScoreList[minKey] = TmpValue;
        }

        for(int index = 0 ; index < StdScoreList.length ; index++)
        {
            StdScoreList[index].printScore();
        }
    } ? end main ?
} ? end Test ?
```

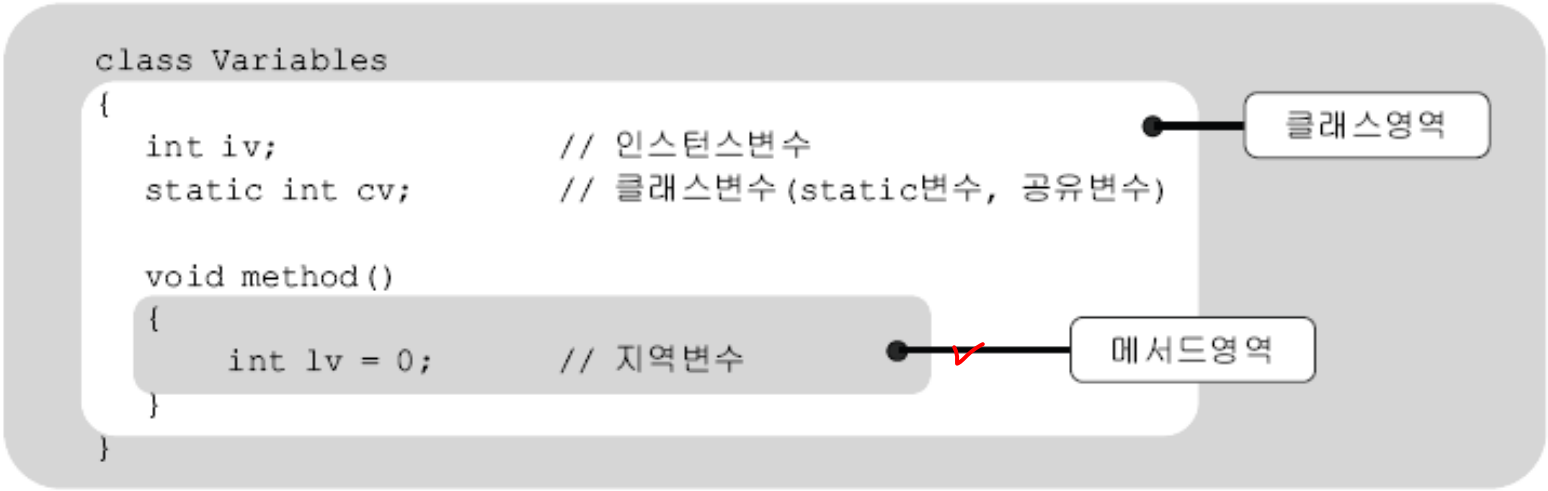
// 성적출력

멤버변수와 메서드

주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

선언위치에 따른 변수의 종류 (1)

"변수의 선언위치가 변수의 **종류**와 **범위(scope)**를 결정한다."



변수의 종류	선언위치	생성시기
클래스 변수	클래스 영역	클래스가 메모리에 올라갈 때
인스턴스 변수		인스턴스 생성시
지역변수	메서드 영역	변수 선언문 수행시

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

선언위치에 따른 변수의 종류 (2)

- **인스턴스 변수(instance member variable)**
  - 각 인스턴스의 개별적인 저장공간.
    - 인스턴스 마다 다른 값 저장가능
  - 인스턴스 생성 후, '참조변수.인스턴스변수명'으로 접근
  - 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지컬렉터에 의해 자동제거됨

- **클래스 변수(class member variable)**
  - 같은 클래스의 모든 인스턴스들이 공유하는 변수
  - 인스턴스 생성없이 '클래스이름.클래스변수명'으로 접근
  - 클래스가 로딩될 때 생성되고 프로그램이 종료될 때 소멸

- **지역변수(local variable)**
  - 메서드 내에 선언되며, 메서드의 종료와 함께 소멸
  - 조건문, 반복문의 블럭{} 내에 선언된 지역변수는 블럭을 벗어나면 소멸

클래스 변수 & 인스턴스 변수 생성시기 비교 (1)

```
class MyClass {

    // 인스턴스 멤버변수
    int         member_varialbe_1 = 18;

    // 클래스 멤버변수
    static int  member_varialbe_2 = 218;

}

class MainClass {
    public static void main( String[] args) {

        System.out.println(MyClass.member_varialbe_1);

        System.out.println(MyClass.member_varialbe_2);
    }
}
```

클래스 변수 & 인스턴스 변수 생성시기 비교 (2)

```
class MyClass {

    // 인스턴스 멤버변수
    int         member_varialbe_1 = 18;

    // 클래스 멤버변수
    static int  member_varialbe_2 = 218;

}

class MainClass {
    public static void main( String[] args) {

        //System.out.println(MyClass.member_varialbe_1);

        System.out.println(MyClass.member_varialbe_2);

        MyClass aObject = new MyClass();

        System.out.println(aObject.member_varialbe_1);
        System.out.println(aObject.member_varialbe_2);
    }
}
```

클래스 변수 & 인스턴스 변수 예제 (1)

```
class Card {
    String  kind;
    int     number;

    static  int width;
    static  int height;

    void printCardAttribute()
    {
        System.out.println(kind + "\t" + number + "\t width: " + width  + "\theight: " + height);
    }
}
```

// 인스턴스 변수

// 클래스 변수

클래스 변수 & 인스턴스 변수 예제 (2)

```
public class SimpleTest {

    public static void main(String[] args) {

        Card CardOne      = new Card();
        CardOne.kind      = "스페이스";
        CardOne.number    = 10;
        CardOne.width     = 80;
        CardOne.height    = 150;
        CardOne.printCardAttribute();


        Card CardTwo      = new Card();
        CardTwo.kind      = "하트";
        CardTwo.number    = 5;
        CardTwo.printCardAttribute();

        CardTwo.width     = 40;
        CardTwo.heig
        CardOne.print
    } ? end main ?
}
```

```
스페이스   10      width: 80      height: 150
하트       5       width: 80      height: 150
스페이스   10      width: 40      height: 20
```

메서드 (Method)

- **메서드란?**
  - 작업을 수행하기 위한 명령문의 집합
  - 어떤 값을 입력 받아서 처리하고 그 결과를 돌려준다.
  - **구조는 C언어의 함수와 동일!!**

- **메서드의 장점과 작성지침**
  - 반복적인 코드를 줄이고 코드의 관리가 용이하다.
  - 반복적으로 수행되는 여러 문장을 메서드로 작성한다
  - 하나의 메서드는 한 가지 기능만 수행하도록 작성하는 것이 좋다.

메서드 정의

- **메서드 정의 방법**
  - 클래스 영역에만 정의할 수 있음!!

```
리턴타입 메서드이름 (타입 변수명, 타입 변수명, ... )
{
        // 메서드 호출시 수행될 코드

}

int add(int a, int b)
{
        int result = a + b;
        return result;    // 호출한 메서드로 결과를 반환한다.

}

void power() {       // 반환값이 없는 경우 리턴타입 대신 void를 사용한다.
        power = !power;
}
```

선언부

구현부

선언부

구현부

클래스메서드(static메서드)와 인스턴스메서드

- **인스턴스 메서드**
  - **인스턴스 생성 후, '참조변수.메서드이름()'으로 호출**
  - 인스턴스 변수나 인스턴스 메서드와 관련된 작업을 하는 메서드
  - 메서드 내에서 인스턴스 변수 사용가능

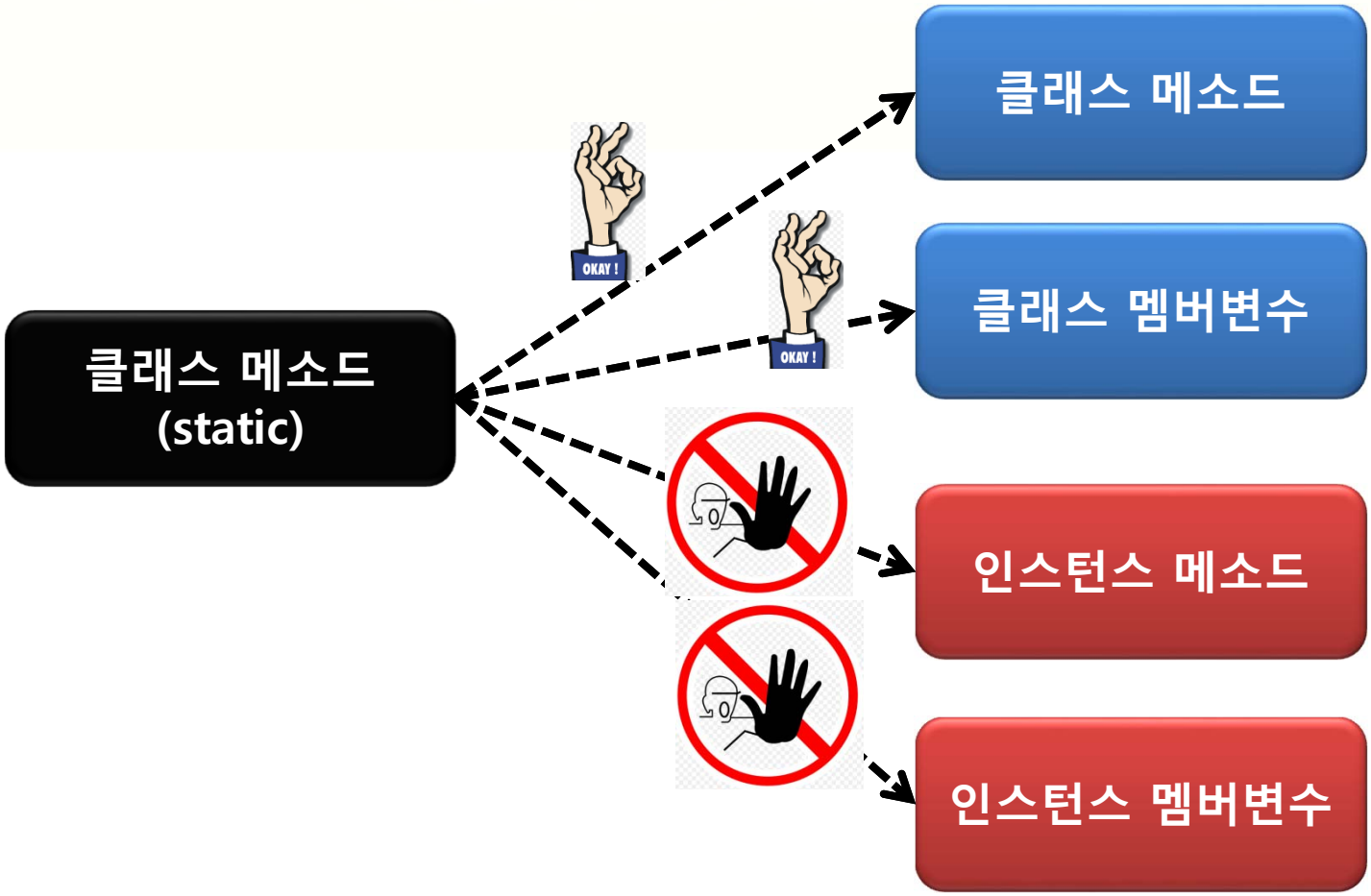
- **클래스 메서드(static메서드)**
  - **객체생성없이 '클래스이름.메서드이름()'으로 호출**
  - 인스턴스 변수나 인스턴스 메서드와 관련 없는 작업을 하는 메서드
  - 메서드 내에서 인스턴스 변수 사용불가
  - 메서드 내에서 인스턴스 변수를 사용하지 않는다면 static을 붙이는 것을 고려한다.

클래스 메서드, 인스턴스 메서드 예제

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

        Calculator MyCal = new Calculator();
        System.out.println(MyCal.add(3, 2));

        System.out.println(Calculator.multiply(2, 3));
    }
}

class Calculator {

    int  add(int x, int y)
    {
        return x + y;
    }

    static long multiply(int x, int y)
    {
        return (long)(x*y);
    }
}
```

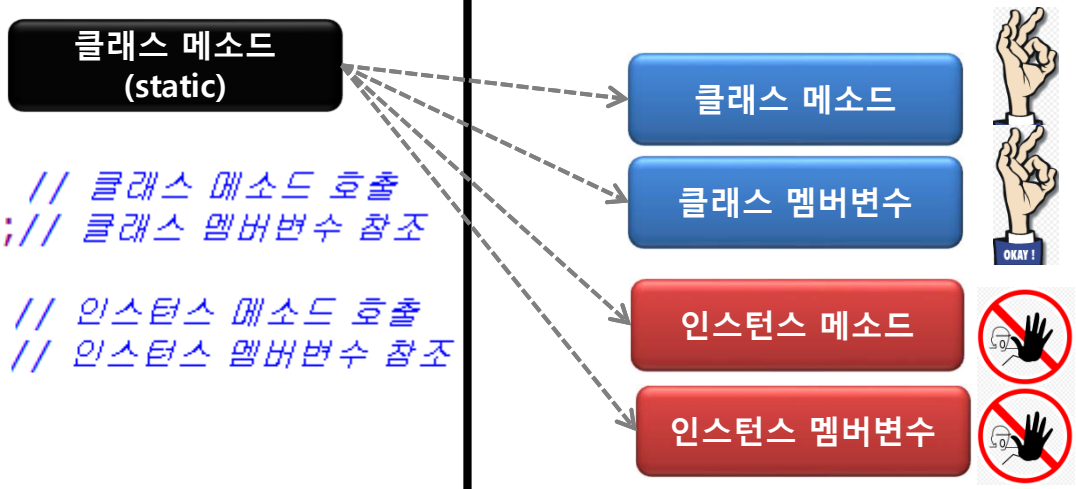
주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

멤버간의 참조와 호출 – 클래스 메서드 (1)

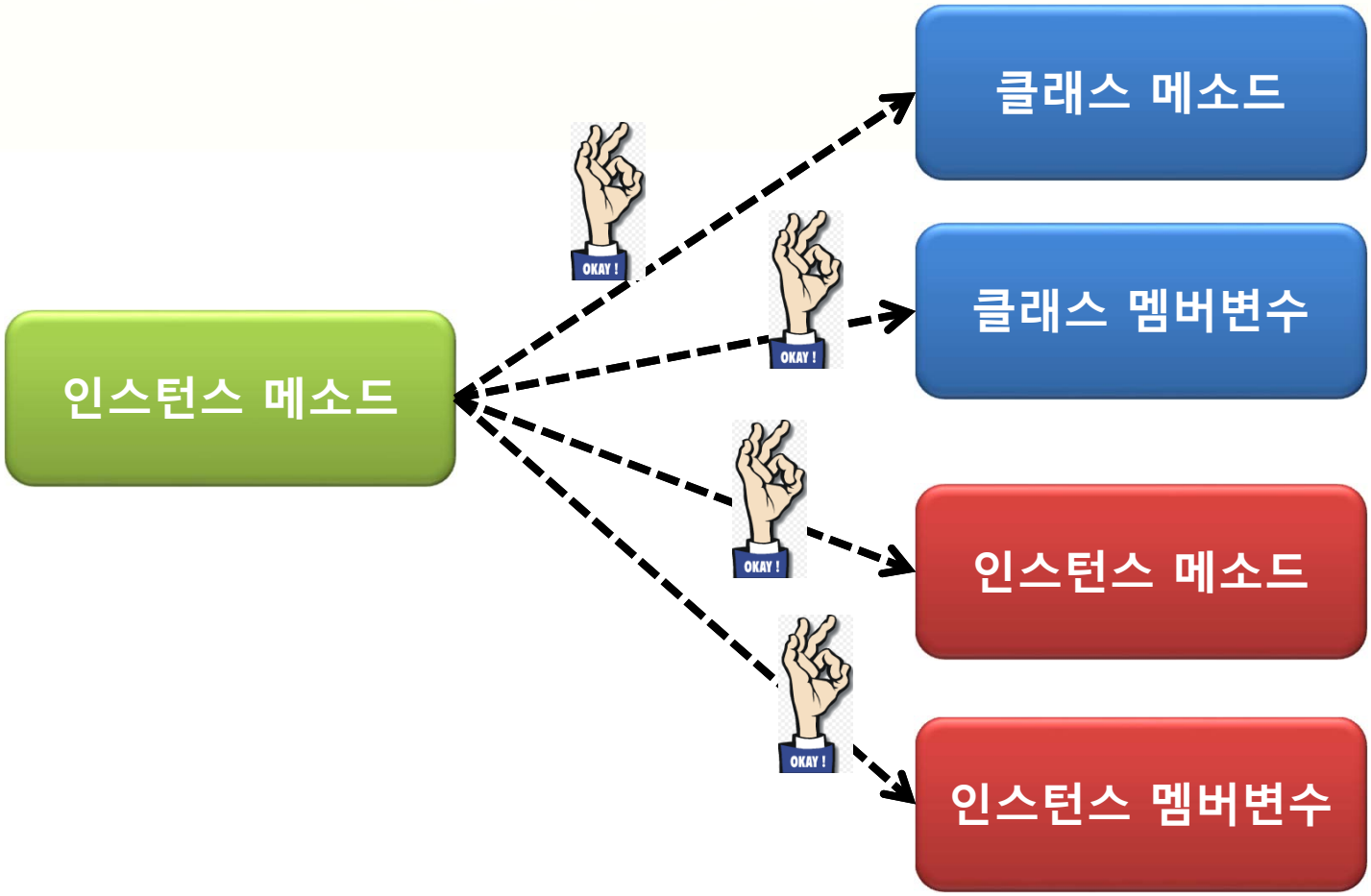


멤버간의 참조와 호출 – 클래스 메서드 (2)

```
class KukuDas {
    int    price   = 200;
    static int calorie = 100;

    void printPrice() {
        System.out.println(price);
    }

    static void printCalorie() {
        printName();                    // 클래스 메소드 호출
        System.out.println(calorie);// 클래스 멤버변수 참조

        printPrice();                   // 인스턴스 메소드 호출
        System.out.println(price);  // 인스턴스 멤버변수 참조
    }

    static void printName() {
        System.out.println("KukuDas");
    }
} ? end KukuDas ?
```

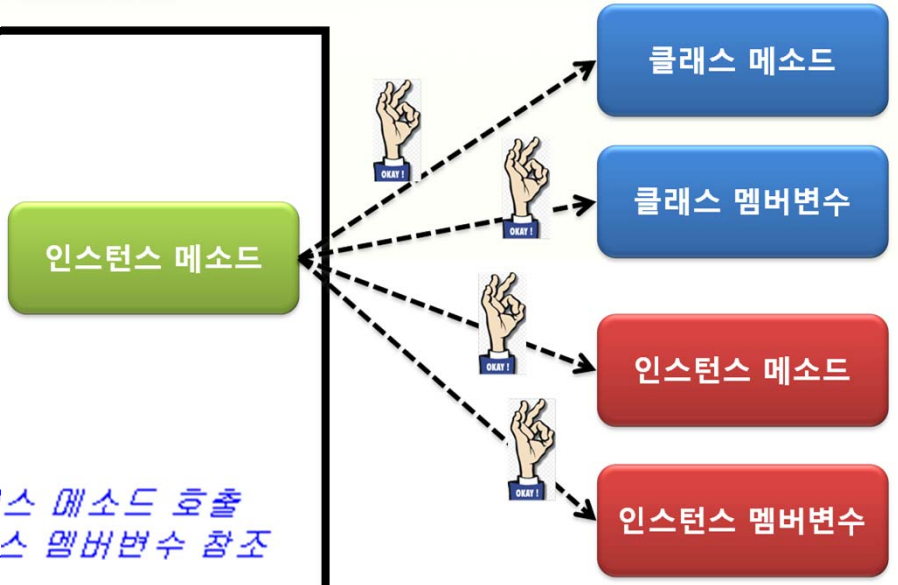


멤버간의 참조와 호출 – 인스턴스 메서드 (1)



멤버간의 참조와 호출 – 인스턴스 메서드 (2)

```
class KukuDas {
    int         price     = 200;
    static int  calorie   = 100;

    void printPrice() {
        System.out.println(price);
    }

    void printCalorie() {
        printName();                    // 클래스 메소드 호출
        System.out.println(calorie);// 클래스 멤버변수 참조

        printPrice();                   // 인스턴스 메소드 호출
        System.out.println(price);  // 인스턴스 멤버변수 참조
    }

    static void printName() {
        System.out.println("KukuDas");
    }
} ? end KukuDas ?
```



주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

기본형 매개변수와 참조형 매개변수

- **기본형(원시형) 매개변수 (Primitive)**
  - 변수의 값을 읽기만 할 수 있다.(read only)
  - boolean, char, byte, short, int, long, float, double
  - 실제 값을 저장

- **참조형 매개변수 (Reference)**
  - 변수의 값을 읽고 변경할 수 있다.(read & write)
  - 기본형을 제외한 나머지 (String, System 등)
  - 객체의 주소를 저장 (4byte, 0x00000000~0xffffffff)

기본형(Primitive) 매개변수 사용예제

```
public class SimpleTest {

    public static void main(String[] args) {

        Data d  = new Data();
        d.x     = 10;

        change(d.x);
        System.out.println("After change(): " + d.x);
    }

    static void change(int x)
    {
        x = 1000;

        System.out.println("change(): " + x);
    }
}

class Data {
    int     x;
}
```

Call by value
Call by reference

```
change(): 1000
After change(): 10
```

참조형(Reference) 매개변수 사용예제

```
public class SimpleTest {

    public static void main(String[] args) {

        Data d    = new Data();
        d.x       = 10;

        change(d);
        System.out.println("After change(): " + d.x);
    }

    static void change(Data d)
    {
        d.x = 1000;

        System.out.println("change(): " + d.x);
    }
}

class Data {
    int       x;
}
```

```
change(): 1000
After change(): 1000
```

오버로딩 (Overloading)

주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

메서드 오버로딩(Method overloading)

- **"하나의 클래스에 같은 이름의 메서드를 여러 개 정의하는 것을 메서드 오버로딩"**

* overload - vt. 과적하다. 부담을 많이 지우다.

- **오버로딩의 조건**
  - 메서드의 이름이 같아야 한다.
  - 매개변수의 개수 또는 타입이 달라야 한다.
  - **매개변수는 같고 리턴타입이 다른 경우는 오버로딩이 성립되지 않는다.**
    - 리턴타입은 오버로딩을 구현하는데 아무런 영향을 주지 못한다.

오버로딩 예제 (1)

```
class Calculator {

    int  add(int x, int y)
    {
        System.out.println("add(int x, int y) is invoked");
        return x + y;
    }

    int  add(int x, int y, int z)
    {
        System.out.println("add(int x, int y, int z) is invoked");
        return x + y + z;
    }

    long add(int x, long y)
    {
        System.out.println("add(int x, long y) is invoked");
        return x + y;
    }

    float add(int x, float y)
    {
        System.out.println("add(int x, float y) is invoked");
        return x + y;
    }

    float add(float x, float y)
    {
        System.out.println("add(float x, float y) is invoked");
        return x + y;
    }

    int add(int[] x)
    {
        System.out.println("add(int[] x) is invoked");

        int sum = 0;

        for(int i = 0 ; i < x.length ; i++)
            sum += x[i];

        return sum;
    }

}
```

오버로딩 예제 (2)

```
public class SimpleTest {

    public static void main(String[] args) {

        Calculator MyCal = new Calculator();

        MyCal.add(1, 2.0f);
        MyCal.add(1.0f, 2);
        MyCal.add(2, 4);
        MyCal.add(2, 8L);
        MyCal.add(3L, 8L);

        int[] intArray = {10, 20, 30};
        MyCal.add(intArray);


    }
}
```

오버로딩의 잘못된 예

- 매개변수의 이름이 다른 것은 오버로딩이 아니다.

```
[보기1]
int add(int a, int b) { return a+b; }
int add(int x, int y) { return x+y; }
```

- 리턴타입은 오버로딩의 성립조건이 아니다.

```
[보기2]
int add(int a, int b) { return a+b; }
long add(int a, int b) { return (long)(a + b); }
```

오버로딩의 장점

- **Look->** System.out.println메서드
  - 다양하게 오버로딩된 메서드를 제공함으로써 모든 변수를 출력할 수 있도록 설계

```
void println()
void println(boolean x)
void println(char x)
void println(char[] x)
void println(double x)
void println(float x)
void println(int x)
void println(long x)
void println(Object x)
void println(String x)
```

생성자 (Constructor)

주문식교육의 산실
영진전문대학
컴퓨터정보계열 정영철

생성자 (constructor)란?

- **인스턴스가 생성될 때마다 호출되는 '인스턴스 초기화 메서드'**
  - 인스턴스 변수의 초기화 또는 인스턴스 생성시 수행할 작업에 사용

- **모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.**

```
Card c = new Card();
```

1. 연산자 new에 의해서 메모리(heap)에 Card클래스의 인스턴스가 생성된다.
2. 생성자 Card()가 호출되어 수행된다.
3. 연산자 new의 결과로, 생성된 Card인스턴스의 주소가 반환되어 참조변수 c에 저장된다.

생성자 조건

- **생성자의 이름은 클래스의 이름과 같아야 한다.**

- **생성자는 리턴 값이 없다.**
  - 하지만 void를 생성자 선언문 앞에 쓰지 않는다.

```
class Calculator {
    Calculator()
    {
      // 객체 생성시 수행할 내용
    }
}
```

```
class Calculator{
      ...
      Calculator() {  // 매개변수가 없는 생성자.
          // 인스턴스 초기화 작업
      }

      Calculator(int x, float y) { // 매개변수가 있는 생성자
          // 인스턴스 초기화 작업
      }
}
```

생성자 예제 (1)

```
public class SimpleTest {

    public static void main(String[] args) {

        Calculator MyCal = new Calculator();

        MyCal.x = 2;
        MyCal.y = 3;

        System.out.println(MyCal.add());
    }
}

class Calculator {

    int x;
    int y;

    Calculator()
    {
    System.out.println("Constructor is invoked! ! ");
    }

    int add()
    {
        return x+y;
    }

}
```

생략가능

```
Constructor is invoked!!
5
```

생성자 예제 (2)

```
public class SimpleTest {

    public static void main(String[] args) {

        Calculator MyCalOne = new Calculator();
        System.out.println(MyCalOne.add());

        Calculator MyCalTwo = new Calculator(2, 3);
        System.out.println(MyCalTwo.add());
    }
}
```

```
class Calculator {

    int x;
    int y;

    Calculator()
    {
        x = 0;
        y = 0;
        System.out.println("Constructor() is invoked! ! ");
    }

    Calculator(int inputX, int inputY)
    {
        x = inputX;
        y = inputY;

    System.out.println("Constructor(int inputX, int inputY) is invoked! ! ");
    }

    int add()
    {
        return x+y;
    }

}
```

기본 생성자 (default constructor) (1)

"모든 클래스에는 반드시
하나 이상의 생성자가 있어야 한다."

- 클래스에 생성자가 하나도 없으면 컴파일러가 기본 생성자를 추가한다.
- 생성자가 하나라도 있으면 컴파일러는 기본 생성자를 추가하지 않는다!!!!

기본 생성자 (default constructor) (2)

```
class Calculator {



}
```

```
class Calculator {

    Calculator()
    {
      // 객체 생성시 수행할 내용
    }
}
```

다음 프로그램의 실행 결과는? ^_____^

```
public class SimpleTest {

    public static void main(String[] args) {

        Calculator MyCalOne = new Calculator();
        MyCalOne.x = 4;
        MyCalOne.y = 3;

        System.out.println(MyCalOne.add());


        Calculator MyCalTwo = new Calculator(2, 3);
        System.out.println(MyCalTwo.add());
    }
}


class Calculator {

    int x;
    int y;

    Calculator(int inputX, int inputY)
    {
        x = inputX;
        y = inputY;
    }

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

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

다음 프로그램 실행 결과 2탄~~~

```java
public class SimpleTest {

    public static void main(String[] args) {

        Car MyCarOne = new Car(2010, "BMW X5", 40);
        MyCarOne.printCarInfo();

        Car MyCarTwo = new Car(MyCarOne);
        MyCarTwo.printCarInfo();

        Car MyCarThree = new Car();
        System.out.println(MyCarTwo.printCarInfo());
    }
}
```

```java
class Car {

    int     year;
    String  name;
    int     mile;

    Car(int year, String name, int mile)
    {
        this.year   = year;
        this.name   = name;
        this.mile   = mile;
    }

    Car(Car SrcObj)
    {
        year    = SrcObj.year;
        name    = SrcObj.name;
        mile    = SrcObj.mile;
    }

    void printCarInfo()
    {
        System.out.println("Year: " + year + "\tName: " + name + "\tMile: " + mile);
    }
}
```

변수의 초기화

변수의 초기화

- 멤버변수(인스턴스 변수,클래스 변수)의 타입의 기본값으로 자동 초기화 되므로 생략할 수 있다.

자료형	기본값
boolean	false
char	'\u0000'
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d 또는 0.0
참조형 변수	null

- 지역변수는 사용전에 꼭!!! 초기화를 해수어야한다.

```
class InitTest {
      int x;            // 인스턴스변수
      int y = x;        // 인스턴스변수

      void method1() {
            int i;      // 지역변수
            int j = i;  // 컴파일 에러!!! 지역변수를 초기화하지 않고 사용했음.
      }
}
```

멤버변수의 초기화

- **멤버변수의 초기화 방법**
  - 명시적 초기화(Explicit initialization)

```
class Car {
     int door = 4;                // 기본형(primitive type) 변수의 초기화
     Engine e = new Engine();     // 참조형(reference type) 변수의 초기화

     //...
}
```

  - 생성자 (Constructor)

```
Car(String color, String gearType, int door){
    this.color = color;
    this.gearType = gearType;
    this.door = door;
}
```

  - 초기화 블록 (Initialization block)
    - 인스턴스 초기화 블록: { }
    - 클래스 초기화 블록 : static { }

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

초기화 블록 (initialization)

- **클래스 초기화 블록**
  - 클래스변수의 복잡한 초기화에 사용되며 클래스가 로딩될 때 실행된다.
- **인스턴스 초기화 블록**
  - 생성자에서 공통적으로 수행되는 작업에 사용되며 인스턴스가 생성될 때 마다 (생성자보다 먼저) 실행된다.

```
class InitBlock {
     static { /* 클래스 초기화블럭 입니다. */ }

     { /* 인스턴스 초기화블럭 입니다. */ }

     // ...
}
```

```
1  class StaticBlockTest {
2      static int[] arr = new int[10]; // 명시적 초기화
3
4      static { // 배열 arr을 1~10사이의 값으로 채운다.
5          for(int i=0;i<arr.length;i++) {
6              arr[i] = (int)(Math.random()*10) + 1;
7          }
8      }
9      //...
10 }
```

멤버변수의 초기화 시기와 순서

```
1  class InitTest {
2      static int cv = 1;   // 명시적 초기화
3      int iv = 1;          // 명시적 초기화
4
5      static {    cv = 2; }   // 클래스 초기화 블럭
6      {   iv = 2; }           // 인스턴스 초기화 블럭
7
8      InitTest() { // 생성자
9          iv = 3;
10     }
11 }
```

```
InitTest it = new
    InitTest();
```

클래스 초기화			인스턴스 초기화			
기본값	명시적 초기화	클래스 초기화블럭	기본값	명시적 초기화	인스턴스 초기화블럭	생성자
cv 0	cv 1	cv 2	cv 2 iv 0	cv 2 iv 1	cv 2 iv 2	cv 2 iv 3
1	2	3	4	5	6	7

주문식교육의 산실 영진전문대학
컴퓨터정보계열 정영철

초기화 블록…… 예제 : Instance

```
public class SimpleTest {

    public static void main(String[] args) {

        MobilePhone MyPhone1 = new MobilePhone();
        MobilePhone MyPhone2 = new MobilePhone();
        MobilePhone MyPhone3 = new MobilePhone();
        }
}

class MobilePhone {

    MobilePhone()
    {
        System.out.println("\"MobilePhone()\" constructor is invoked");
    }

    {System.out.println("\"Initialization block\" is invoked");}

    String    name;
    String    vendor;

    int   price;

    void printPhoneInfo()
    {
        System.out.println("Year: " + year + "\
    }
}
```

int yaer;
int mile;

```
"Initialization block" is invoked
"MobilePhone()" constructor is invoked
"Initialization block" is invoked
"MobilePhone()" constructor is invoked
"Initialization block" is invoked
"MobilePhone()" constructor is invoked
```

초기화 블록…… 예제 : Static

```
public class SimpleTest {

    public static void main(String[] args) {

        MobilePhone MyPhone1 = new MobilePhone();
        MobilePhone MyPhone2 = new MobilePhone();
        MobilePhone MyPhone3 = new MobilePhone();
        }
}

class MobilePhone {

    MobilePhone()
    {
        System.out.println("\"MobilePhone()\" constructor is invoked");
    }

    static {System.out.println("\"Initialization block\" is invoked");}

    String    name;
    String    vendor;

    int   price;

    void printPhoneInfo()
    {
        System.out.println("Year: " + year + "\tN
    }
}
```

```
"Initialization block" is invoked
"MobilePhone()" constructor is invoked
"MobilePhone()" constructor is invoked
"MobilePhone()" constructor is invoked
```