



# Object oriented Programming & Class

---

JAVA Programming  
강 성 관



# 객체지향 프로그래밍 이란 ?

---

## ■ 정의

- 처리하고자 하는 자료에 중점을 두고, 프로그램을 객체(object)라는 것으로 모델화하는 프로그래밍

## ■ 장점

- 소프트웨어의 확장성(extensibility) 향상
- 재사용(reusability)성 향상
- 프로그래머의 생산성(productivity) 증가
- 유지보수(maintainability) 비용 절감



# 객체지향 프로그래밍 개념(1)

---

## □ JAVA Programming Language(1)

- `main(String args[])` 함수에서 시작
- `class`와 `object`의 개념 기반으로 프로그래밍이 이루어짐.
- 이식성이 높다.
- 내부 포인터를 사용한다.
- 완벽한 객체지향 언어이다.
- 자바 가상 기계(JAVA Virtual Machine(JVM) ) 이용
  - 자바 컴파일러에 의해 만들어진 ‘자바 바이트코드(bytecode)’를 해석하고 실행하는 가상적 기계(CPU)’
  - 가비지 컬렉터(Garbage Collector) 모듈이 내장됨.
    - 사용이 끝난 객체를 감지해서 자동으로 삭제하는 일을 전담함.



# 객체지향 프로그래밍 개념(2)

---

## □ JAVA Programming Language(2)

- 플랫폼 독립성(Platform Independent)
  - 한번 만들어진 자바 프로그램은 운영체제나 CPU의 타입에 상관없이 프로그램이 동작됨.
  - JVM이 어떤 해당 플랫폼에 설치되면 자바로 작성된 프로그램에게 동일한 실행환경을 제공하기 때문.
- MultiThread Programming 이 가능
  - Thread
    - 프로그램 안에서 독립적으로 실행되는 작은 실행단위.
    - 하나의 프로그램 안에서 같이 실행되기 때문에, 메모리 공유가 가능하면서 프로그램이 보다 효율적으로 실행될 수 있도록 해줌.



# 객체지향 프로그래밍 개념(3)

---

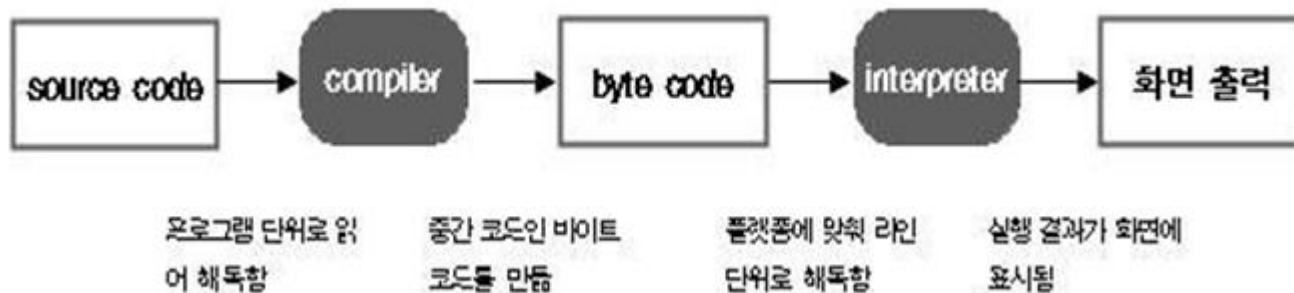
## □ JAVA Programming Language(3)

- MultiThread
  - (ex)인터넷에서 여러 개의 파일을 동시에 다운로드 받는 것.
  - CPU의 시분할 개념(Time Sharing)의 작동 방식에 근거함.
    - 시분할 개념이란 프로그램에 정해진 순서대로 단시간(약 100밀리초)씩 실행 시간을 주어 이를 되풀이 해서 일정 기간에 복수의 프로그램을 실행할 수 있는 시스템.
      - 시간을 세분화해서 사용하는 한편 동시에 복수의 일을 처리하는 것처럼 보이는 방식.

# 객체지향 프로그래밍 개념(4)

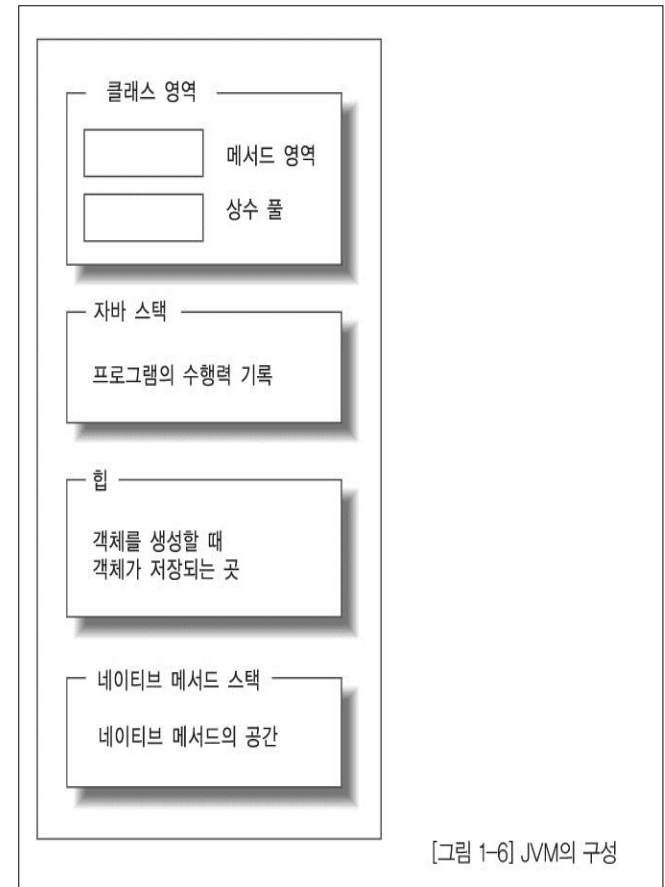
## □ JAVA Programming Language(3)

- 자바의 실행 과정
  - .java 파일 → 컴파일(javac) → .class 파일 → 인터프리터(java) → 실행 결과
- ✓ 자바 해석기(interpreter)
  - ✓ 자바 컴파일러에 의해서 생성된 바이트 코드를 자바가상머신에서 라인 별로 실행하도록 해주는 개발 도구.



# 객체지향 프로그래밍 개념(5)

- JVM (Java Virtual Machine)의 메모리
  - 스택 영역 (Runtime Stack) : 실행 시 사용하는 메모리 영역
  - 힙 영역 (Garbage Collection Heap) : 동적 메모리 할당 영역
  - 클래스 영역 (Constant & Code Segment) : 상수 데이터 및 static 데이터 할당 영역
  - 네이티브 메서드 스택영역 (Process Register) : 프로세서 실행 관련 메모리 할당 영역





# 객체지향 프로그래밍 개념(6)

---

□ `program = algorithm + data ;`

## □ 프로그래밍 기술(1)

### — Structured Programming

- 함수 단위로 프로그램을 개발
- 하향식(top-down) 설계
- 데이터보다는 알고리즘에 더 치중
- 순차, 선택, 반복의 3가지 구조로 구성가능
- **function, recursion**



# 객체지향 프로그래밍 개념(7)

## □ 프로그래밍 기술(2)

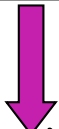
### - Object-oriented Programming(객체지향 프로그래밍)

- ◆ Object : 물건, 대상, 목적, 객체, 반대하다

- ◆ Structured Programming + new OOP

- ◆ **Object(객체) = Data(구성요소) + operation(행위)**

Abstraction  
(Modeling)



instantiation



**Class(클래스) = field(변수 및 상수) + method(메서드)**

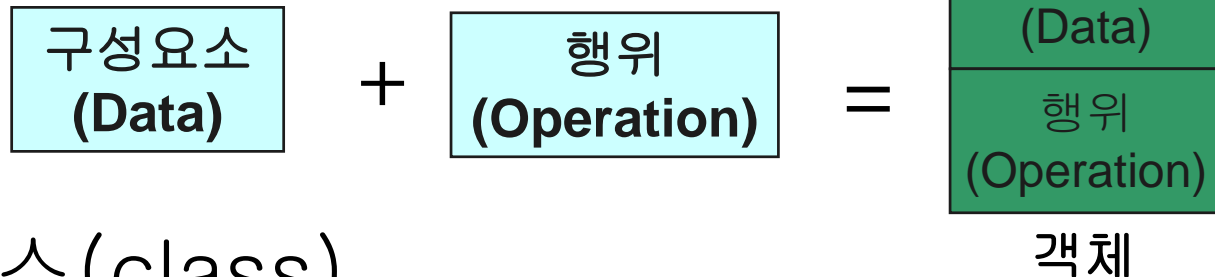
- 알고리즘보다는 데이터를 강조
- 해결해야 할 문제를 언어의 절차적 접근 방식에 끼워 맞추지 않고  
해결해야 할 문제의 특성에 맞게 데이터형 자체를 설계

❖ *class, object*

# 객체지향 프로그래밍 개념(8)

## ■ 객체(Object)

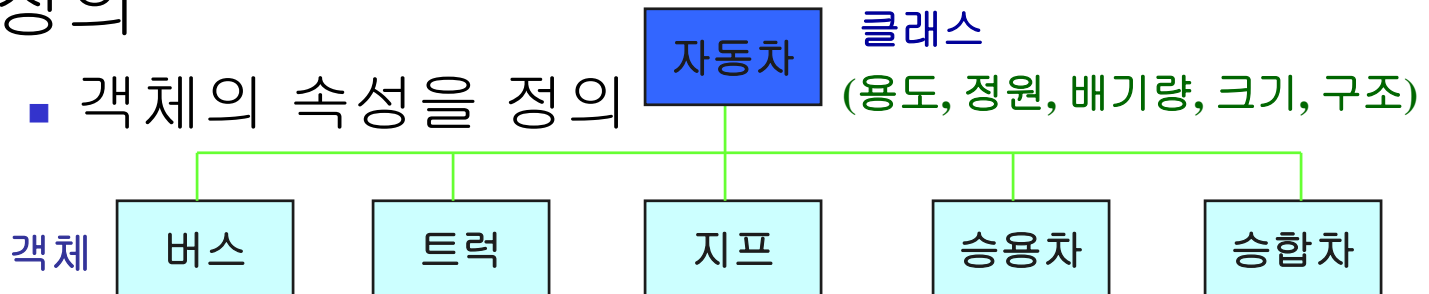
### ■ 정의



## ■ 클래스(class)

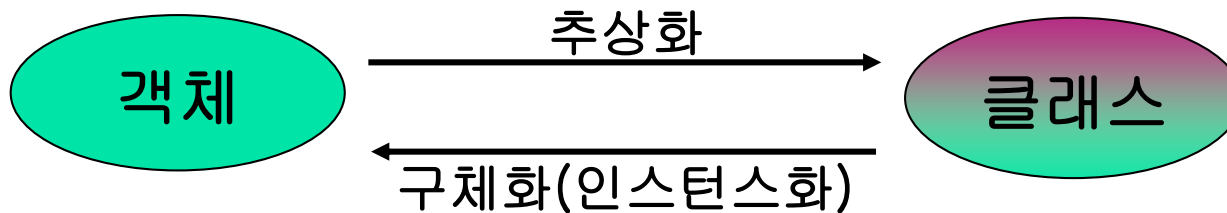
### ■ 정의

#### ■ 객체의 속성을 정의



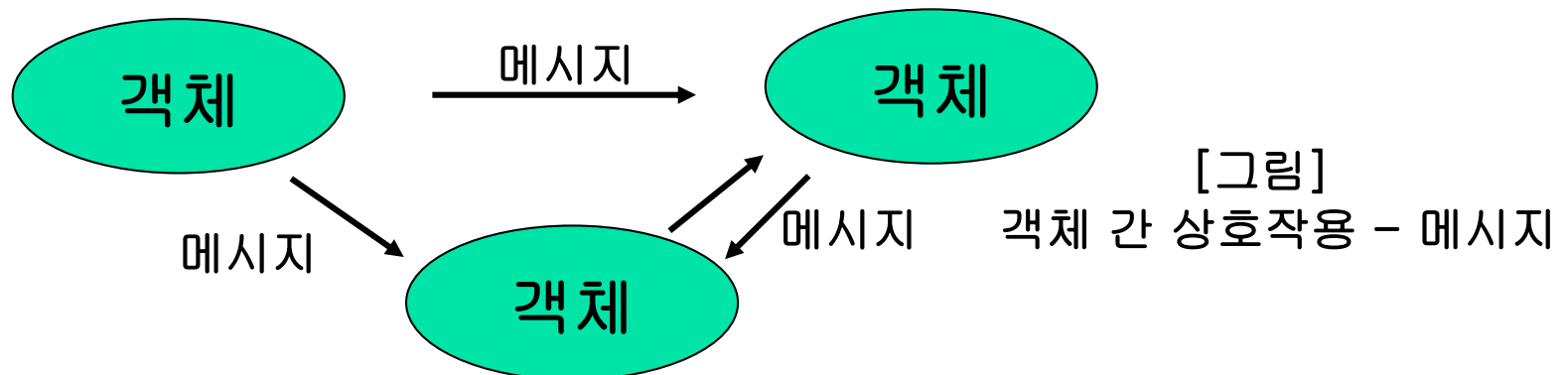
# 객체지향 프로그래밍 개념(9)

## ◆ 객체와 클래스간의 관계



- ✓ 객체는 클래스의 인스턴스(instance) : instantiation
- ✓ 객체는 클래스의 복사물이다.

## ◆ 객체와 객체간의 관계





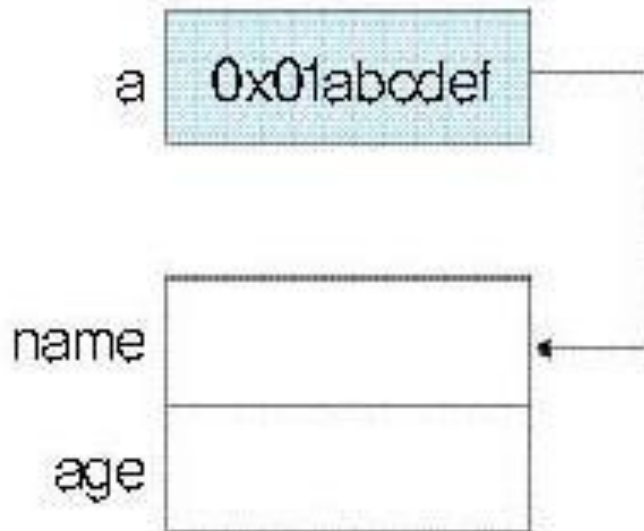
# 객체 생성

---

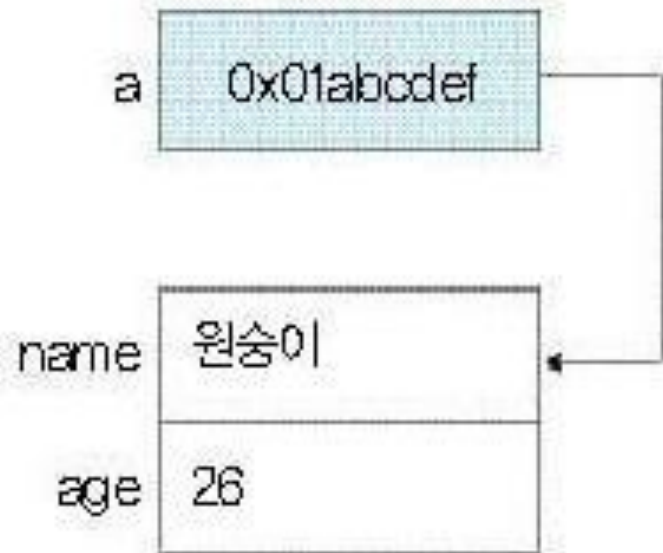
- 클래스는 붕어빵을 만드는 틀이고 객체는 붕어빵
  - 클래스는 동일한 모양의 객체를 여러 개 만들어 내기 위한 틀이기에 객체가 생성되어야만 실질적인 데이터를 저장할 수 있는 기억 공간이 메모리에 할당됨.
    - 이런 객체를 인스턴스, 실체라고도 명명함.
- 실체를 얻어내기 위한 기본적인 형틀이 바로 클래스.

# 레퍼런스 변수와 객체 생성(1)

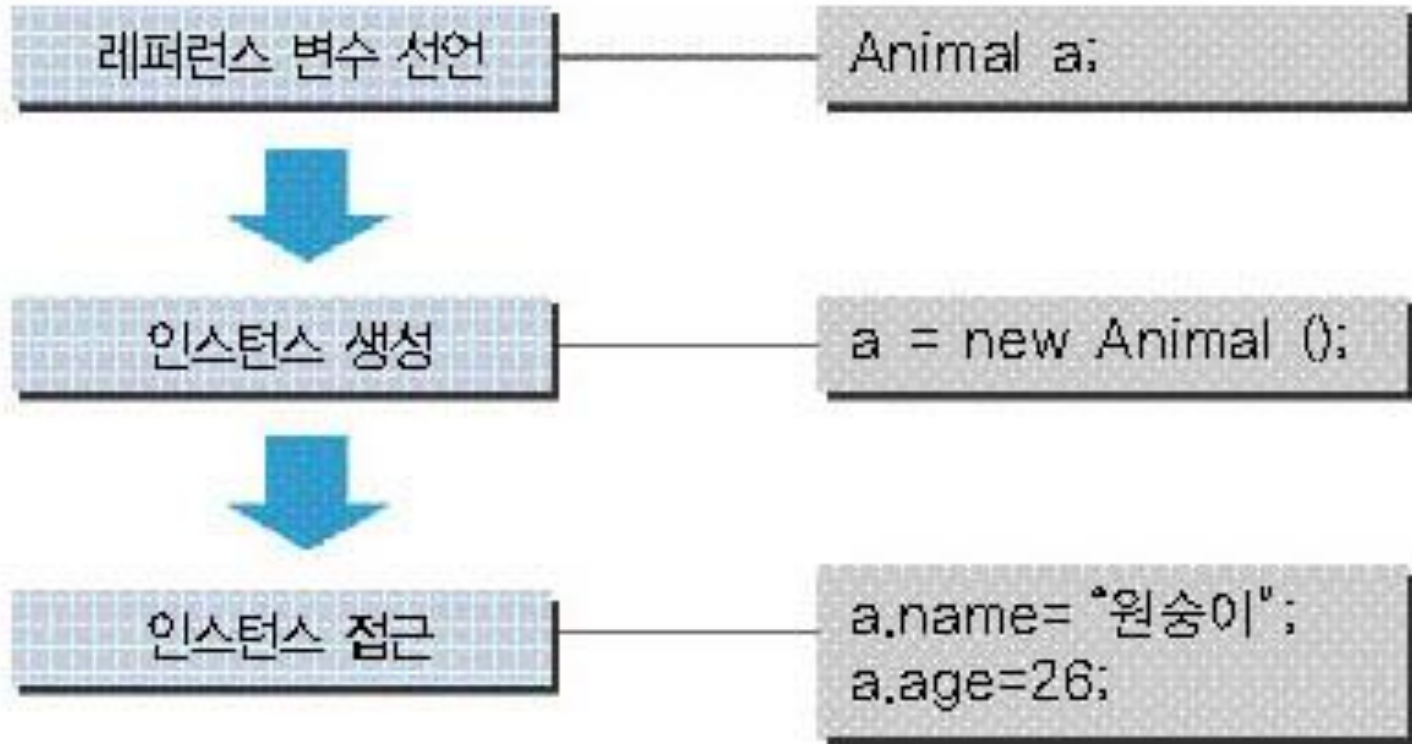
Animal a;  
a=new Animal();



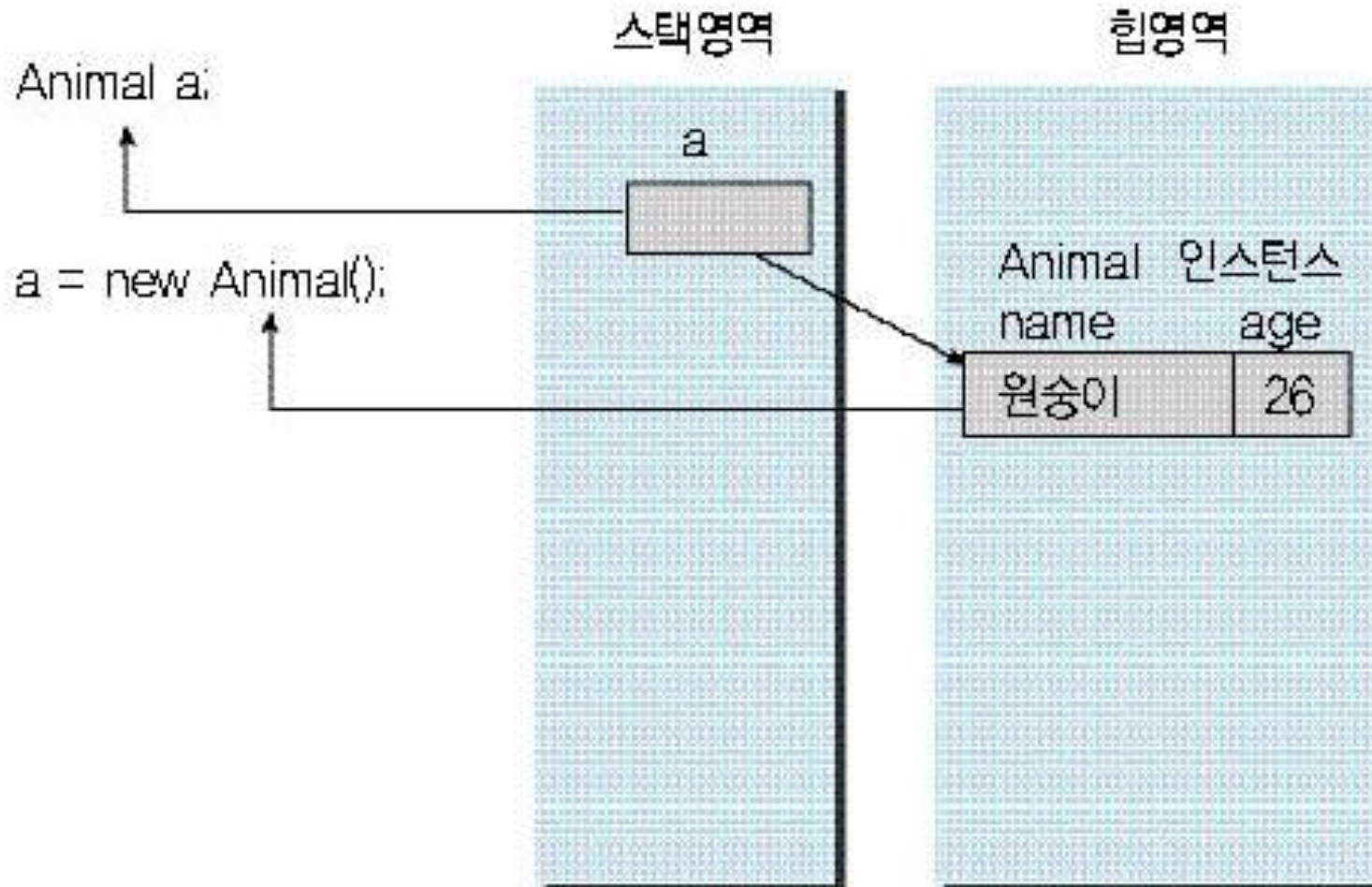
a.name ="원숭이";  
a.age=26;



## 레퍼런스 변수와 객체 생성(2)



# 레퍼런스 변수와 객체 생성(3)



# 객체지향 프로그래밍 개념(10)

## <클래스 정의>

클래스 사람

```
{  
    입;  
    키;  
    몸무게;  
  
    보다();  
    숨쉬다();  
    말하다();  
}
```

```
class Person  
{  
    mouth;  
    height;  
    weight;  
  
    see();  
    breathe();  
    talk();  
}
```

철 수	
입	
키	170
몸무게	
보다()	
숨쉬다()	
말하다()	

영 희	
입	
키	
몸무게	

```
public static void main(String ar[ ])  
{  
    Person chulsu=new Person();  
    Person younghee= new Person();  
  
    chulsu.height=170;  
    younghee.talk();  
}
```

## <객체 생성시 메모리 보기>

- ❖ 객체가 하나 더 늘 때마다 구성요소는 새로 메모리에 생성
  - ❖ 캡슐화되어서 객체별로 처리가 가능하기 때문

## <클래스와 객체>

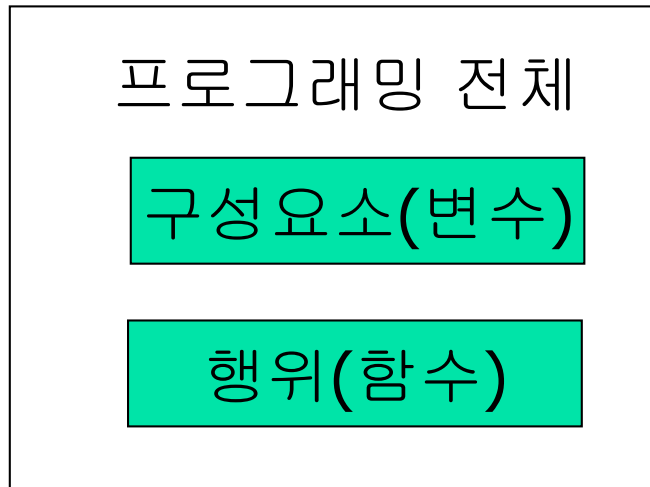


# 객체지향 프로그래밍 개념(11)

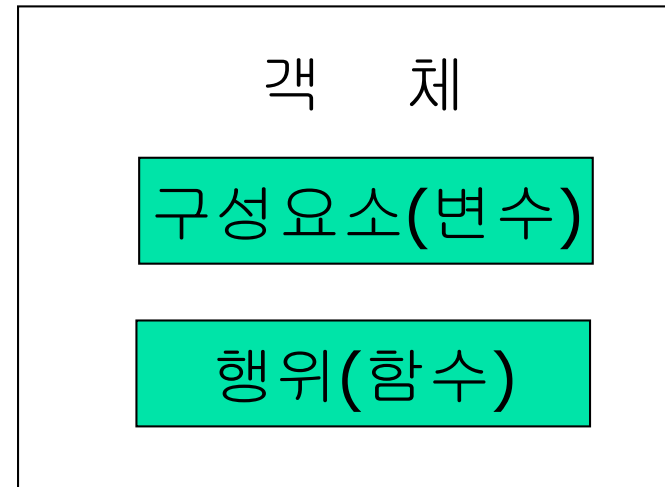
## ◆ 객체지향언어의 특성

### ■ 캡슐화(encapsulation)

- 객체(object)로서 캡슐화를 지원
  - 정보은닉(Information Hidding)
    - 객체 자신의 정보를 감춤
- 데이터와 그것을 조작하는 코드를 같이 묶는 구조
- 코드와 데이터는 비공개(private) 또는 공개(public)가능



<기존 프로그래밍>



<객체지향 프로그래밍>

# 객체지향 프로그래밍 개념(12)

## - 상속성(Inheritance)

- ◆ 하나의 객체가 다른 객체의 특성을 이어받을 수 있게 해 주는 과정
- ◆ 자식 클래스는 부모클래스의 모든 특성을 상속 받고, 자기 자신의 특성을 추가 시켜 정의 할 수 있다.
- 상위 클래스에 있는 것은 상속 받아쓰고 상위클래스에 없는 것은 새로 만들자

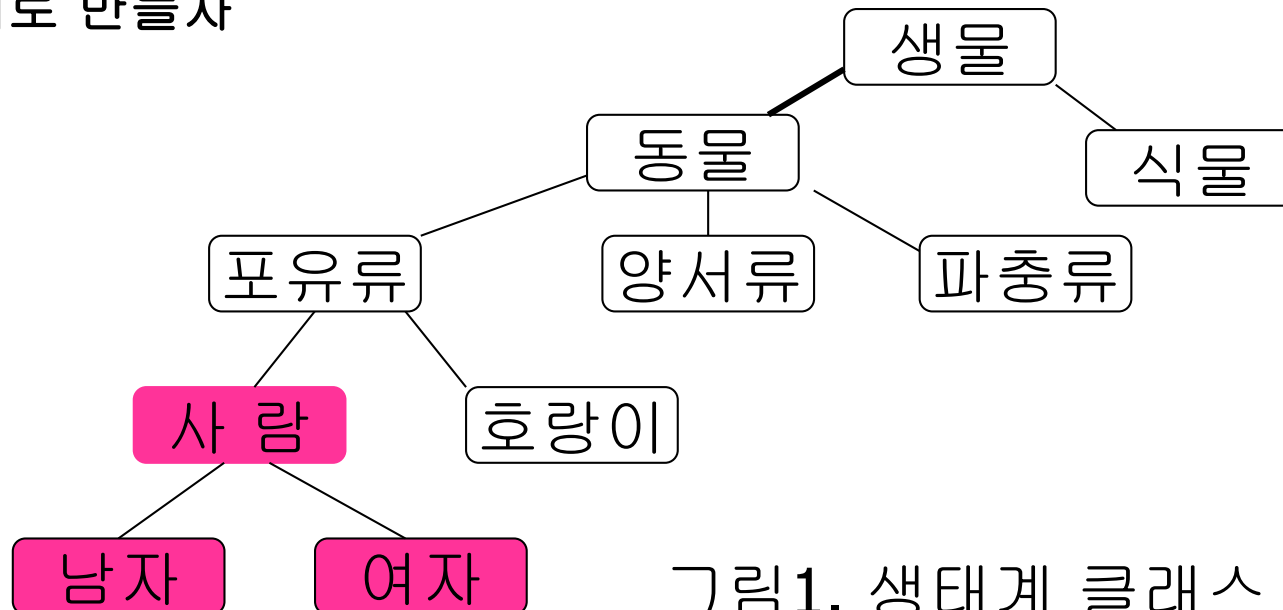
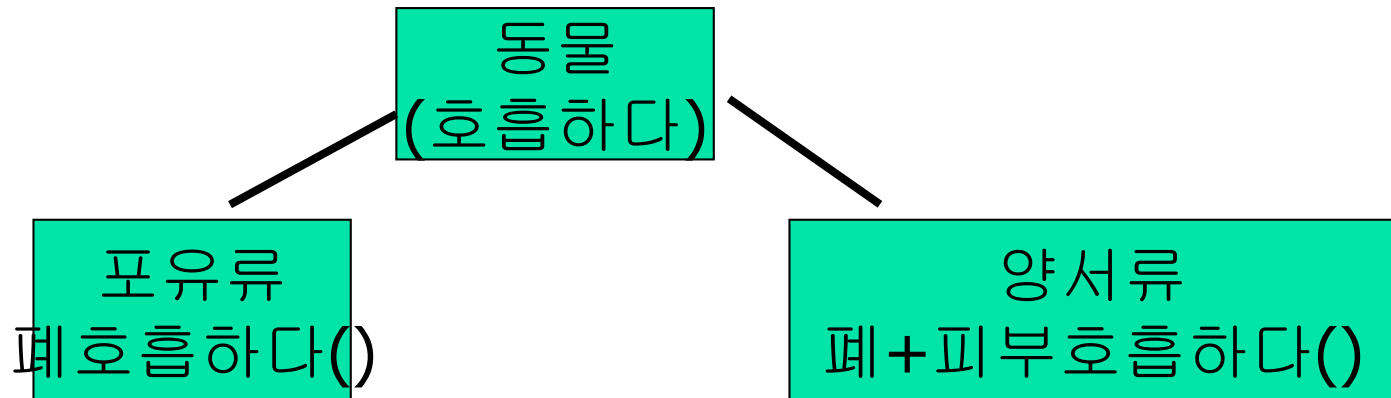


그림1. 생태계 클래스

# 객체지향 프로그래밍 개념(13)

## 다형성(polymorphism)

- ♦ 목적은 다르지만 연관성 있는 두 가지 이상의 용도로 하나의 이름을 사용할 수 있게 하는 성질
  - ➔ 하나의 인터페이스(Interface)에 여러 방법들
- 메서드 재정의(method redifinition)
  - method overloading(중복)
  - method overriding(재생 또는 치환)



# 클래스의 기본개념(1)

- 클래스 : 객체를 생성하는 도구
- 클래스 선언 형식

```
class 클래스명{  
    [private | public | protected | package] field 선언;  
    [private | public | protected | package] method 선언;  
}
```

- **field 선언**은 멤버변수의 선언으로 대부분 접근 지정자는 “**private**” 으로 한다.
- **method 선언**은 메서드의 선언을 말하는데 대부분 접근 지정자는 “**public**” 으로 한다.
- 세 액세스 권한을 나타내는 데에는 순서가 정해져 있지 않으며 생략 가능.
- 클래스 멤버 선언의 규칙
  - ◆ **field** 이름은 동일한 클래스 내에서 유일해야 한다.
  - ◆ **method**의 경우 중복가능
    - 메서드의 인자는 달라야 한다
    - 메서드 중복(method overloading)

# 클래스의 기본개념(2)

## 접근 지정자 - 멤버의 액세스 권한(1)

### - private

- 클래스 내에서만 액세스 됨
- 외부에서 접근 불가능
- 하나의 클래스 내에서만 사용 가능한 멤버를 지정할 때 사용하는 제한자

### - protected

- ◆ 클래스의 상속을 구현하는 과정에서 사용
- ◆ 파생된 클래스에 의해서는 제어가 가능
- ◆ 외부에서 접근 불가능
  - (동일 파일 + 동일 폴더 + 상속)에 있는 클래스에서만 접근 가능.

### - public

- 외부로부터 접근 가능
- 객체를 가진 모든 영역



# 클래스의 기본개념(3)

## □ 접근 지정자 - 멤버의 액세스 권한(2)

### - package

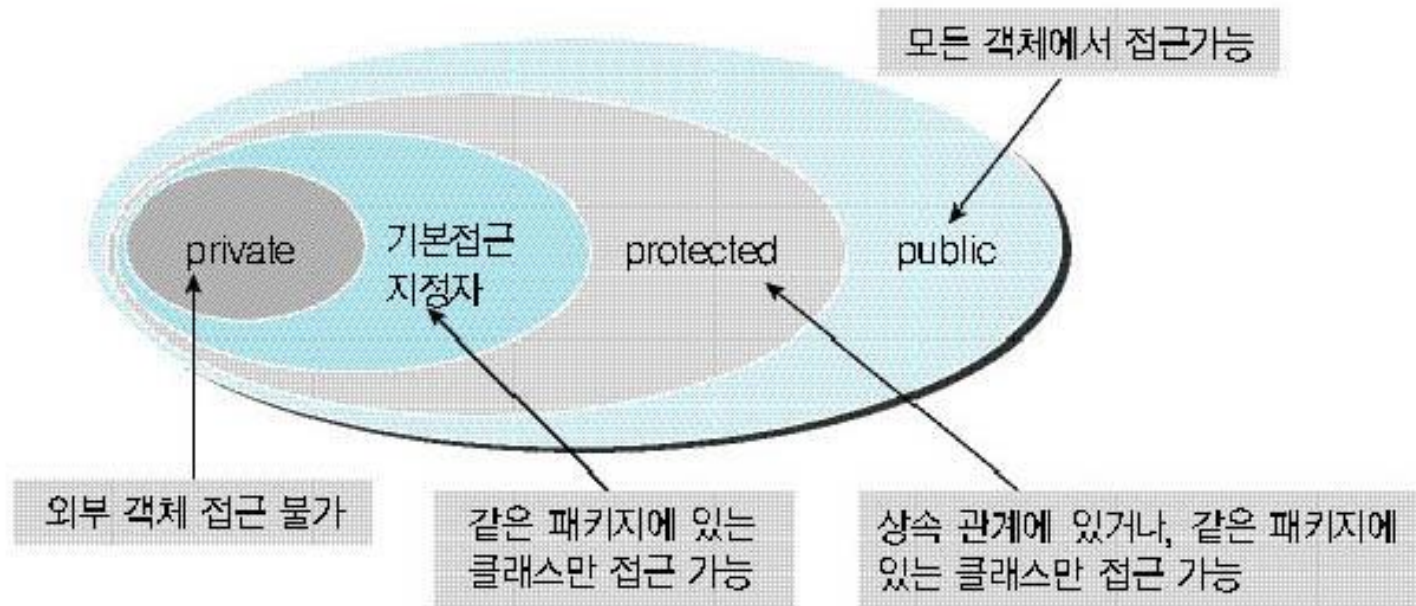
- 동일 파일 + 동일 폴더.
- 제한자를 사용하지 않았을 때를 나타냄.
- 자신의 클래스 내부와 동일한 패키지들 간에 그리고 동일한 파일 내부에서만 사용되도록 제한함.
- 동일한 말로 friendly 혹은 NONE 이라고도 함.

### ❖ 주의할 점

- ❖ 접근 제어 수식어(access control modifier)는 클래스의 멤버로 사용될 때에만 쓸 수 있는 예약어.
  - ❖ 어떤 메서드 안에 있는 지역 필드(local field)를 선언할 경우에 쓰면 에러가 발생함.

# 접근 지정자

접근 지정자	자신의 클래스	같은 패키지	하위 클래스	다른 패키지
private	O	X	X	X
생략(기본 접근 지정자)	O	O	O	X
protected	O	O	O	X
public	O	O	O	O



# 예제 프로그램

```
class MyClass{ public int a; }  
public class classExTwo {  
    public static void main(String args[]){  
        MyClass ob1 = new MyClass(); MyClass ob2 = new MyClass();  
        ob1.a = 10; ob2.a = 99;  
        System.out.printf("ob1.a = %d\n", ob1.a );  
        System.out.printf("ob2.a = %d\n", ob2.a );  
    }  
}
```

## □ **Myclass**클래스의 선언


- 정수형 변수 **a**를 **public**으로 선언
  - ◆ 클래스의 내부 및 외부에서 자유롭게 액세스가능

## □ **Myclass**클래스형의 객체 **ob1**과 **ob2**의 선언

- **public** 멤버의 접근
  - ◆ **ob1.a**      ◆ **ob2.a**

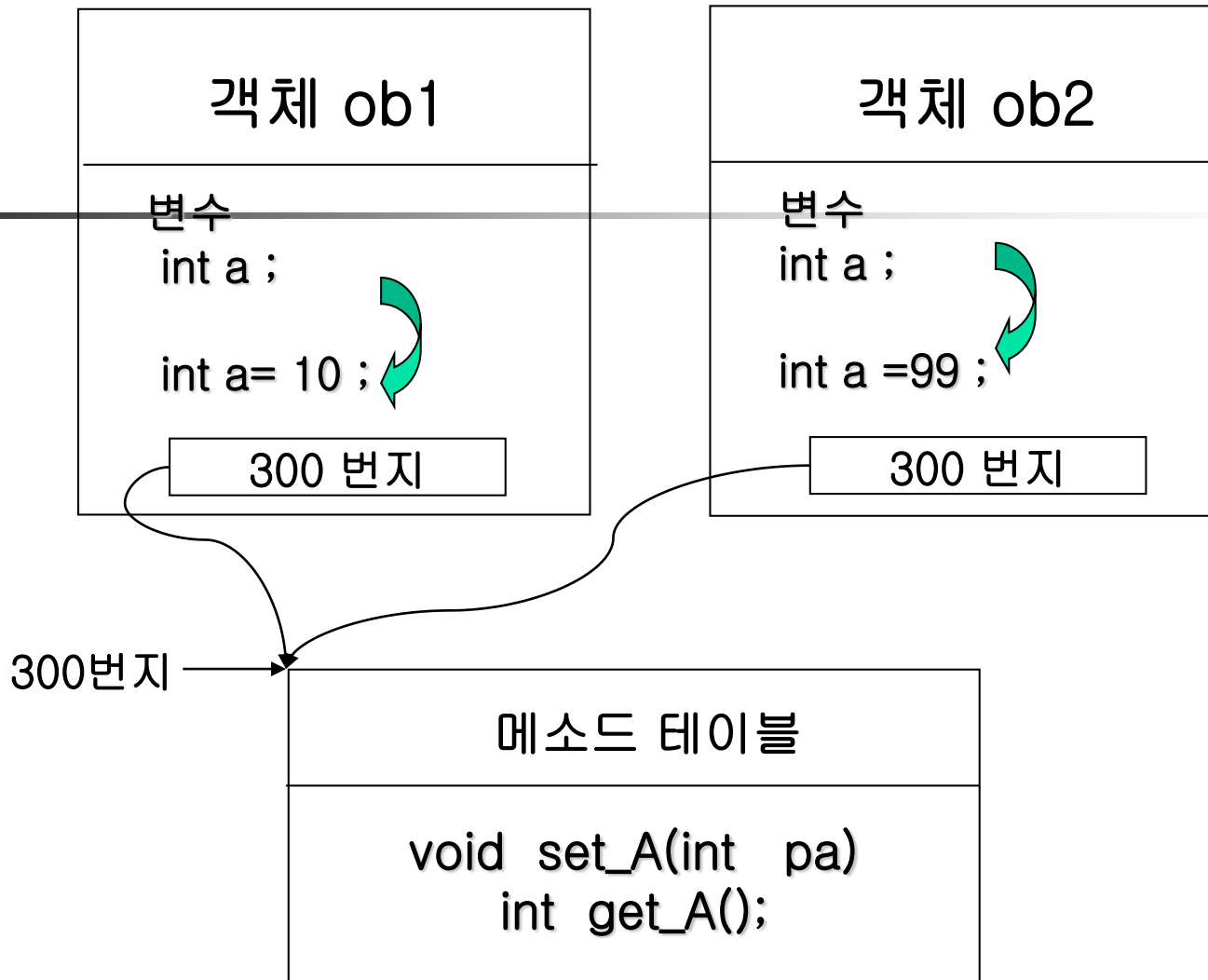


# 예제 프로그램



```
class MyClass{
    private int a ; //Myclass 의 비공개 부분
    public void setA(int pa){ a = pa ; }
    Public int getA(){ return a ;}
}

public class classExThree {
    public static void main(String args[]){
        MyClass  ob1 = new MyClass() ;
        MyClass  ob2 = new MyClass() ;
        ob1.setA(10) ; //ob1.a = 10; 구문은 Compile Error
        ob2.setA(20) ; //ob2.a = 99; 구문은 Compile Error
        System.out.printf("ob1.a = %d\\n", ob1.getA() ) ;
        System.out.printf("ob2.a = %d\\n", ob2.getA() ) ;
    }
}
```



< 각각 다른 메모리 공간에 객체가 들어간 모습 >