



05. 자바 객체 개념

1] 객체지향언어의 역사

- 1960년대 초반에 시작. 당시 미국의 컴퓨터 과학자 알란 케이(Alan Kay)는 Simula라는 프로그래밍 언어를 개발
- Smalltalk는 1970년대에 개발된 프로그래밍 언어로, OOP의 개념을 가장 완벽하게 구현한 언어로 평가
- C++은 1980년대에 개발된 프로그래밍 언어로, C 언어의 장점을 살리면서 OOP의 개념을 도입한 언어입니다.
- Java는 1990년대에 개발된 프로그래밍 언어로, 인터넷 환경에 최적화된 OOP 언어

2] 객체지향 기본원리

객체지향 기본원리

- 객체지향 프로그램은 프로시저(함수) 추상화 와 data 추상화를 묶은 것

원리	내용
추상화	현실세계를 그대로 표현하기보다 중요한 측면에 주목 , 상세내역을 없애 나가는 과정
상속성	Super Class가 갖는 성질을 Sub Class에 자동 으로 부여 PGM을 쉽게 확장할수 있게 만드는 수단
다형성	동일Interface 서로 다르게 응답할수 있는 특성 연관 Class 위한 일관된 매개체를 개발하는 수단 OverLoading : 동일한 이름의 Operation(Method) 사용 가능 OverRiding : Super Class의 Method를 sub Class에서 재정의
Capsule화	객체의 상세 내역을 객체외부에 철저히 숨기고 , 단순히 Message 만으로 객체와 상호작용 객체 내부 구조와 실체 분리 로 내부 반경이 PGM에 미치는 영향 최소화 유지보수 용이

3] 객체 지향 프로그래밍 특징

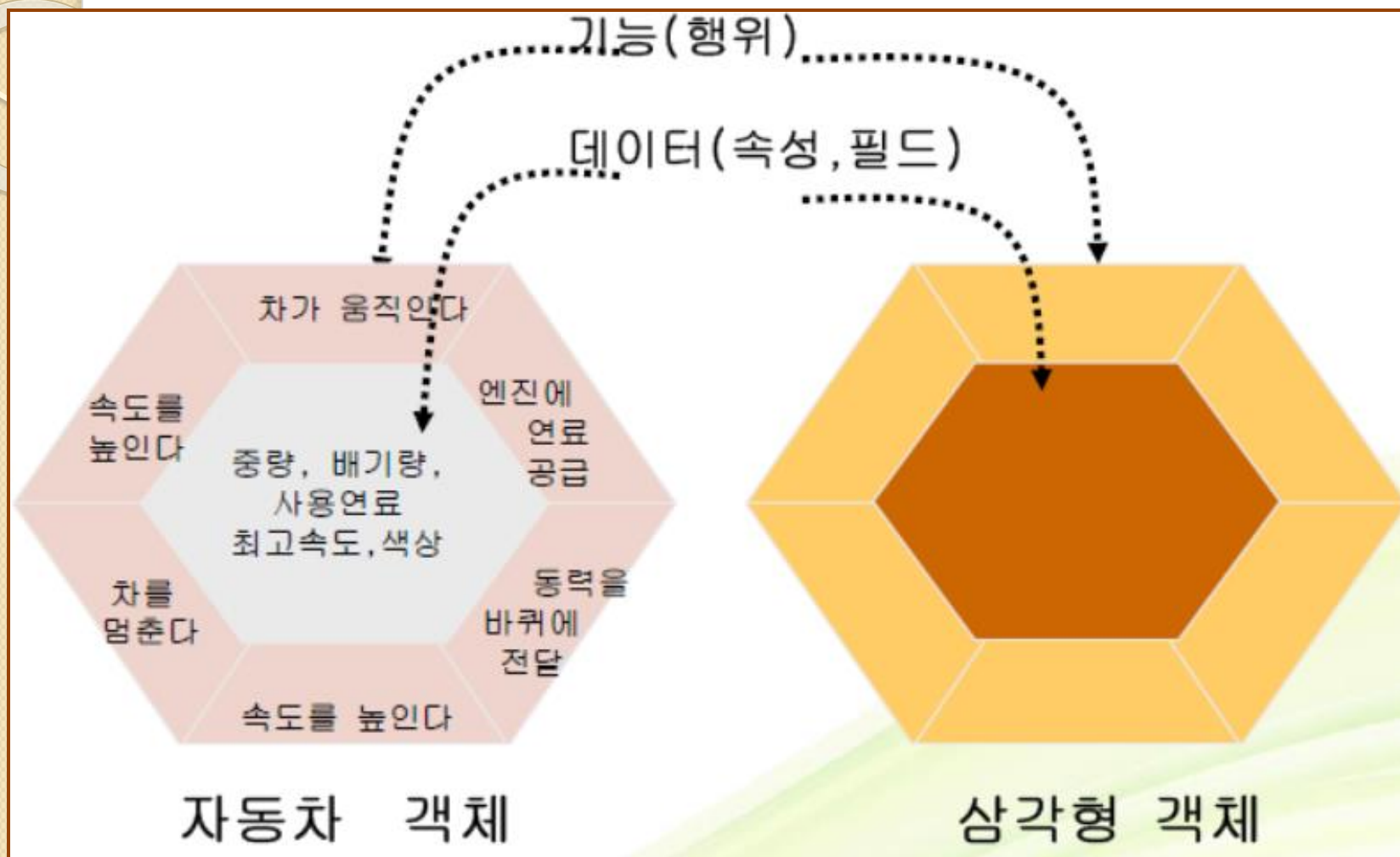
1. 주요요소

항목	특징
객체	현실 세계의 사물을 컴퓨터 프로그램에 구현한 단위 속성(attribute)과 메서드(method)로 구성
클래스	객체를 정의하는 청사진 객체의 속성과 메서드를 정의

2. 장점

항목	특징
모듈성	객체는 서로 독립적으로 구현되어 있기 때문에, 프로그램의 모듈성 향상
재사용성	기존 클래스를 재사용할 수 있기 때문에, 개발 시간을 단축하고, 코드의 품질을 향상
유지보수성	객체의 구조가 잘 정의되어 있기 때문에, 프로그램의 유지보수를 용이
확장성	새로운 기능을 추가하기 쉽기 때문에, 프로그램의 확장성

3] 객체(Object) 예시



4] 클래스(Class)

- 서로 공통되는 구조를 가지고 있는 객체를 모아 이 객체들이 가지고 있는 데이터 영역의 구조와 각각의 객체가 수행 할수 있는 메소드들을 정의한 객체를 의미.
- 일반화된 속성과 메소드로 객체를 기술한 것을 클래스
- 객체는 항상 클래스로부터 생성된다.
- 즉 클래스는 객체를 생성하는 형판(template)
- 클래스는 두개의 구성요소(member)인 자료구조(필드)와 연산(메소드) 로 구성
- 클래스로부터 생성된 객체를 instance라 한다.
객체 = instance
- 정보처리의 주체는 클래스가 아니라 객체.
객체지향 프로그래밍의 시작은 클래스의 생성.

5] 클래스 / 객체의 정의와 용도

1. 클래스의 정의 - 클래스란 객체를 정의해 놓은 설계도.
 - ▶ 클래스의 용도 - 클래스는 객체를 생성하는데 사용.
2. 객체의 정의 - 실제로 존재하는 것. 사물 또는 개념.
 - ▶ 객체의 용도 - 객체의 속성과 기능에 따라 다름.

클래스	객체
제품 설계도	제품
TV설계도	TV
붕어빵기계(틀의미)	붕어빵

1. 객체 개념

- ① 클래스로 정의된 데이터와 메서드의 집합.
- ② 객체는 클래스의 인스턴스

2. 인스턴스

- ① 인스턴스는 특정 클래스의 객체.
- ② 즉, 같은 클래스에서 생성된 객체는 모두 인스턴스 객체

3. 예시

```
public class Person {  
    private String name;  
    private int age;  
  
    public Person(String name, int age) {  
        this.name = name;  
        this.age = age;  
    }  
  
    public void sayHello() {  
        System.out.println("안녕하세요, 저는 " + name + "입니다.");  
    }  
}
```

```
Person person1 = new Person("홍길동", 30);  
Person person2 = new Person("김유신", 40);
```

person1과 person2는 모두 **Person** 클래스의 인스턴스.

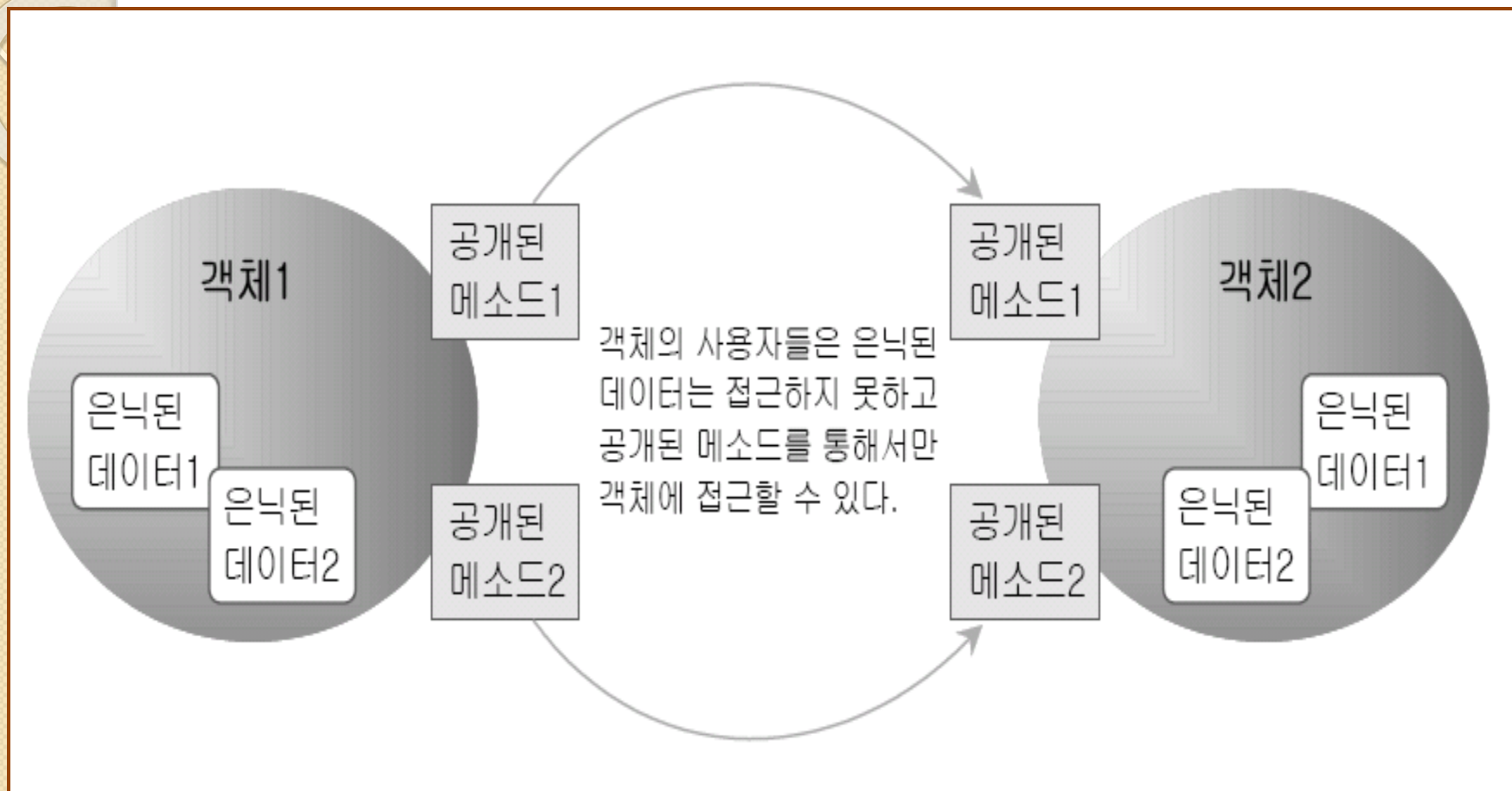
즉, person1과 person2는 모두 같은 클래스에서 정의된 데이터와 메서드를 가지고 있음

5] 객체의 구성요소 - 속성과 기능

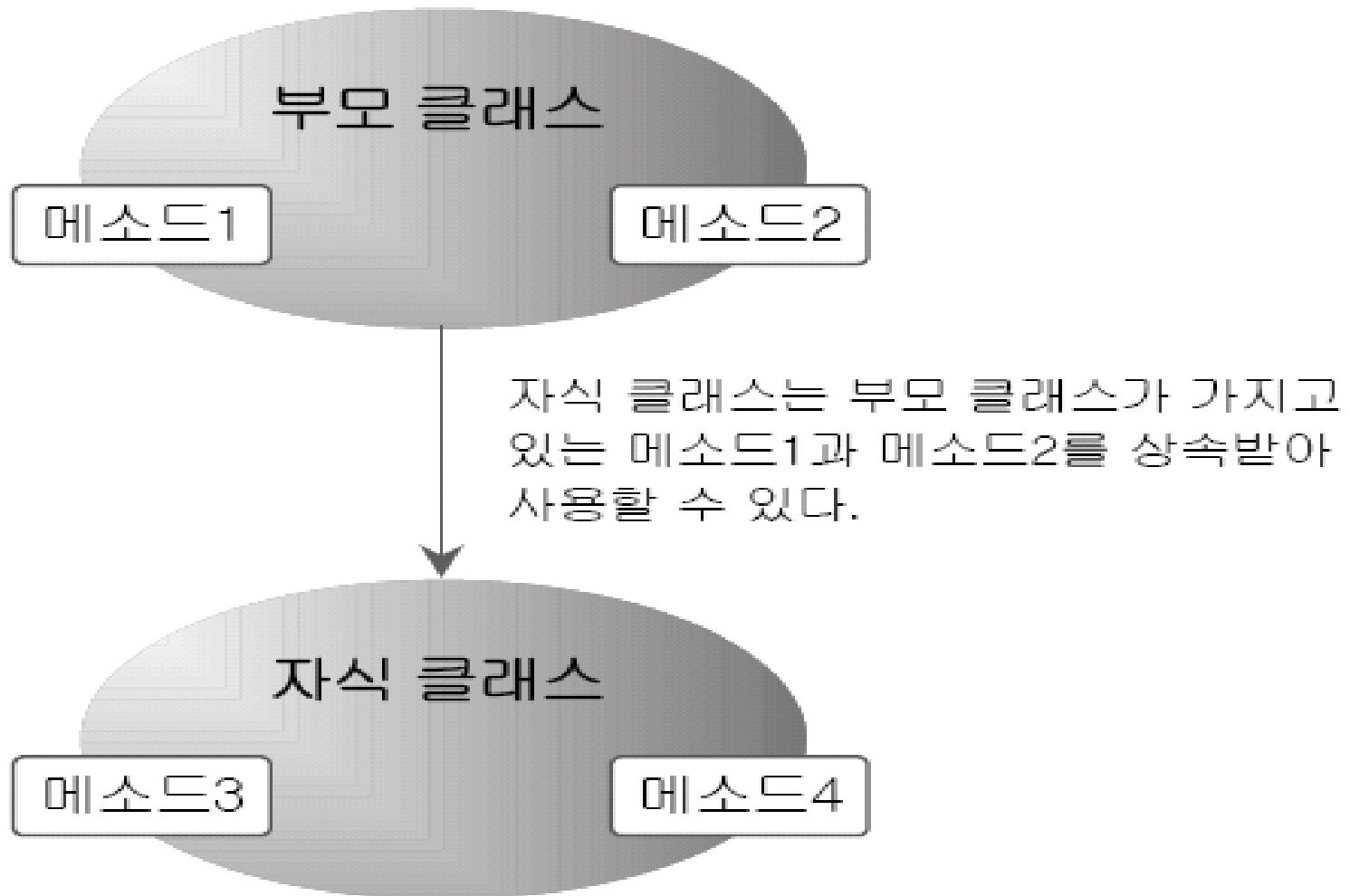
1. 객체는 속성과 기능으로 이루어져 있다.
 - 객체는 속성과 기능의 집합이며, 속성과 기능을 객체의 멤버(member, 구성요소)라고 한다
2. 속성은 변수로, 기능은 메서드로 정의한다.
 - 클래스를 정의할 때 객체의 속성은 변수로, 기능은 메서드로 정의

속성	크기, 길이, 높이, 색상, 볼륨, 채널 등	변수	<pre>class Tv { String color; // 색깔 boolean power; // 전원상태 (on/off) int channel; // 채널 }</pre>
기능	켜기, 끄기, 볼륨 높이기, 볼륨 낮추기, 채널 높이기 등	메서드	<pre>void power() { power = !power; } // 전원on/off void channelUp(channel++;) // 채널 높이기 void channelDown { channel--;} // 채널 낮추기</pre>

6] 클래스의 설계 - 캡슐화와 데이터 은닉



6] 클래스의 설계 - 다형성과 메소드 오버로딩



7] 클래스의 일반 구조

```
class SampleClass { // 클래스 헤더부분
    int a;
    int b; // 멤버 변수 부분
    int c;

    // 생성자
    public SampleClass() {
        // 생성자 부분. 이름이 클래스 명과 같다
        a = x;
        b = y;
        c = z;
    }
    public int sum() { // 메소드 부분
        int d;
        d = a + b + c;
    }
}
```

8] 클래스의 선언

```
[접근 제어자] class 클래스명  
    [extends 상위 클래스명]  
    [implements 인터페이스명] {  
        ..... //클래스 멤버 부분  
    }
```

- ▶ 접근제어자 : 다른 클래스가 이 클래스를 참조할 때의 제한 사항을 지정
- ▶ class : 클래스 정의를 시작하는 키워드
- ▶ 클래스명 : 정의될 클래스의 이름을 지정
- ▶ extends 와 implements : 클래스의 확장과 관련된 예약어

9] 접근 제어 자

▶ public :

해당 클래스의 필드와 메소드의 사용을 다른 모든 클래스에 허용 그리고 이들은 서브클래스로 상속.

▶ protected :

클래스의 멤버를 클래스 자신과 이 클래스로부터 상속받은 서브클래스에만 접근을 허용

▶ private :

해당 클래스만이 이 멤버를 사용할수 있다. 외부객체에서는 절대로 접근을 할 수 없다.

▶ default :

접근제어자를 명시하지 않은 경우의 디폴트 접근제어자 같은 패키지내의 클래스들은 public 권한을 갖고 접근가능

▶ final : 서브클래스를 가질수 없는 클래스

▶ abstract : 추상 메소드를 가지는 추상 클래스를 의미

10] 클래스와 객체 사용예시

1. new 연산자가 힙 영역에 메모리할당 할당 후 되돌려 주는 주소는 레퍼런스 변수에 저장

```
Car car01 = new Car( );
```

① 레퍼런스 변수 선언 ② 인스턴스 생성

2. 레퍼런스 변수는 객체에 대한 참조(주소)를 갖게 됨
3. new 연산자 다음에 클래스 명(Point)을 기술하면
 - 실질적인 좌표 값(x, y)을 저장할 수 있는 기억 공간이 생성.
4. 레퍼런스 변수의 역할
 - new에 의해서 할당되는 기억 공간은 힙 영역인데 이곳은 실질적인 값을 저장할 수 있지만, 힙 영역은 직접 접근할 수 없기에 따로 레퍼런스 변수를 두고 접근

10] 클래스와 객체 사용예시(계속)

1. 닷(.) 멤버 참조 연산자로 필드에 접근

```
Car car01 = new Car( );
```

① 레퍼런스 변수 선언 ② 인스턴스 생성



speed
direction

0
0

```
car02.speed = 10;  
car02.direction = 30;
```



car02

0x200

→ 0x200

speed
direction

10
30

11] 선언위치에 따른 변수의 종류

▶ 인스턴스변수(instance variable)

- 각 인스턴스 개별적인 저장공간. 인스턴스마다 다른 값 저장가능
- 인스턴스 생성 후, '참조변수.인스턴스변수명'으로 접근
- 인스턴스를 생성할 때 생성되고, 참조변수가 없을 때 가비지 컬렉터에 의해 자동 제거됨

▶ 클래스변수(class 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 블록{} 내에 선언된 지역변수는 블록을 벗어나면 소멸

11] 선언위치에 따른 변수의 종류 (계속)

Class Variables {

◦ static int staticVar; // 클래스변수

int instanceVar; // 인스턴스변수

Void method() {
int localVar; // 지역변수
}

}

변수의 종류	선언위치	생성시기
인스턴스변수	클래스 영역	인스턴스 생성 시
클래스변수		클래스가 메모리에 올라갈 때
지역변수	메소드 영역	해당 메소드가 호출될 때

II] 선언위치에 따른 변수의 종류(계속)

Class Variables {

◦ static int staticVar; // 클래스변수

int instanceVar; // 인스턴스변수

Void method() {
int localVar; // 지역변수
}

}

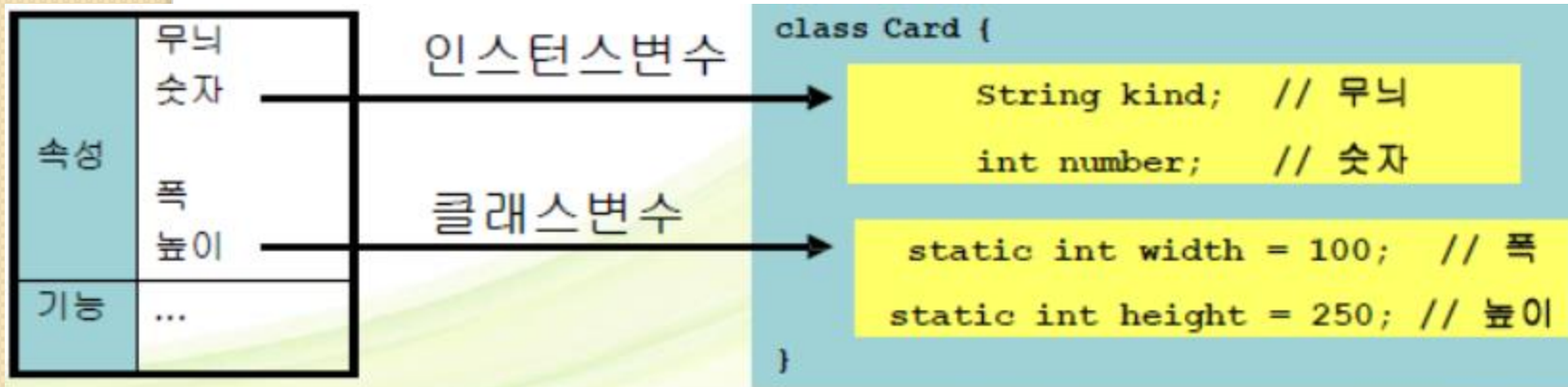
▶ 클래스변수(class variable)

- 동일 클래스 내의 모든 인스턴스들이 공유하는 변수
- 인스턴스 생성없이 '클래스이름.클래스변수명'으로 접근
- 생성과 소멸 : 클래스가 로딩될 때 생성, 프로그램 종료될 때 소멸

Variables.staticVar = 10; // 클래스이름.클래스변수명

12] 선언위치에 따른 변수의 종류-클래스변수와 인스턴스변수

- 1] 인스턴스변수 : 인스턴스가 생성될 때마다 생성, 인스턴스 마다 각기 다른 저장공간 생성 - 동적변수
- 2] 클래스변수 : 클래스가 로딩 될 때 한 번만 생성, 하나의 저장공간을 공유, 공통된 값을 갖는 경우 - 정적변수



12] 클래스와 객체 예시

1] CardTest.java



13] 멤버변수 선언

▶ 멤버 변수 선언

[접근제어자] [static/final] 데이터형 변수명;

☞ static : 클래스 변수, final : 종단 변수

☞ static과 final이 붙지 않은 변수 : 객체변수
(객체 속성변수, 객체참조변수)

(1) 객체 변수(객체 참조변수와 객체 속성변수)

☞ 객체 변수 : 객체가 가질 수 있는 특성을 표현

☞ 객체 속성 변수 : 객체가 가질 수 있는 속성을 나타내는 값으로서
기본 자료형의 값들로 구성

☞ 객체 참조 변수 : 객체를 지정하는 변수. 자바에서는 기본 자료형을
제외한 모든 요소들을 객체로 취급

☞ 사용자는 객체를 생성한 다음 그 객체에 접근 하기 위해서는 객체 참조
변수를 통하여 그 객체의 멤버들에 접근할 수 있다

13] 멤버변수

(1) 객체 참조변수와 객체 속성변수

```
class Box {  
    int width; // 객체 속성 변수 width  
    int height; // 객체 속성 변수 height  
    int depth; // 객체 속성 변수 depth  
}  
  
class MyBox {  
    int vol;           // 객체 속성 변수 vol  
    Box mybox1;        // 객체 참조 변수 mybox1  
    Box mybox2;        // 객체 참조 변수 mybox2  
    String boxname;    // 객체 참조 변수 boxname  
    // 자바에서 문자열은 객체로 취급한다  
    mybox1 = new Box();  
    mybox2 = new Box();  
    .....  
}
```

13] 멤버변수 예시

```
class A {  
    public int x = 10;  
    public int y = 20;  
    int add() {  
        return (x+y);  
    }  
}  
  
class B {  
    public static void main(String[] args) {  
        A a = new A();  
  
        System.out.println("x = " + a.x);  
        System.out.println("y = " + a.y);  
        System.out.println("Sum = " + a.add());  
    }  
}
```


[3] 멤버 변수 - 클래스 변수

클래스 Test

static int number;

객체 생성

클래스 Test로 부터
생성된 모든 객체들은
Test.number로 클래스
변수에 접근 할 수 있
다

객체 고유
데이터

객체 a

객체 고유
데이터

객체 b

객체 고유
데이터

객체 c

객체 고유
데이터

객체 d

13] 멤버 변수 - 클래스 변수

```
public class Static {  
    public int instance_var=100;  
    public static int static_var=100;  
    public void instance_method(){  
        System.out.println("instance_method() invoke...");  
    }  
    public static void static_method(){  
        System.out.println("static_method() invoke...");  
    }  
}
```

/*

static 제한자

1. 멤버변수, 메소드 앞에 부칠수있다.
2. static 제한자가 부터있는 변수나 메소드는 객체생성없이 사용가능
3. 클래스(정적,공용) 변수(메소드)

형식 - 멤버변수: public static int i;

- 멤버 메소드: public static int add(){}
*/

[3] 멤버 변수 - 클래스 변수 예시

```
class Static
```

```
static int num = 0; // 클래스 변수 선언
```

```
int a = 10; // 객체속성 변수
```

```
int b = 20; // 객체속성 변수
```

```
}
```

```
class StaticRun
```

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

```
    Static s1 = new Static();
```

```
    Static s2 = new Static();
```

```
    s1.num = 10;
```

```
    s1.a = 20;
```

```
    s1.b = 30;
```

```
    System.out.println("s1의 값 num="+s1.num+" a="+s1.a+" b="+s1.b);
```

```
    System.out.println("s2의 값 num="+s2.num+" a="+s2.a+" b="+s2.b);
```

```
    System.out.println("클래스 변수 num = " + Static.num );
```

```
}
```

```
}
```

13] 멤버 변수 - 종단(final) 변수

- ▶ 예약어 **final**을 사용하여 종단변수 지정
- ▶ 변할 수 없는 상수 값을 나타낸다
- ▶ 종단변수는 관례상 대문자로 표기한다.

```
final int MAX = 100;  
final int MIN = 1;
```

```
class Circle {  
    public static void main(String[] args) {  
        final float PI = 3.1415f; // 원주율  
        int r = 10;                // 원의 반지름  
        double area = PI * r * r;  
        double round = 2 * PI * r;  
        System.out.println("원의 넓이 = " + area);  
        System.out.println("원의 둘레 = " + round);  
    }  
}
```

13] 메서드-생성자(Constructor)란?

▶ 생성자란?

- 인스턴스가 생성될 때마다 호출되는 "인스턴스 초기화 메서드"
- 몇가지 조건을 제외하고는 메서드와 같다.
- 인스턴스 변수의 초기화 또는 인스턴스 생성시 수행할 작업에 사용
- 모든 클래스에는 반드시 하나 이상의 생성자가 있어야 한다.
- * 인스턴스 초기화 - 인스턴스 변수에 적절한 값을 저장하는 것.

```
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.

13] 생성자의 조건

▶ 생성자의 조건

- 생성자의 이름은 클래스의 이름과 같아야 한다.
- 생성자는 리턴값이 없다. (하지만 void를 쓰지 않는다.)

```
클래스이름 (타입 변수명, 타입 변수명, ... ) {  
    // 인스턴스 생성시 수행될 코드  
    // 주로 인스턴스 변수의 초기화 코드를 적는다.  
}
```

```
class Card {  
    ...  
    Card() { // 매개변수가 없는 생성자.  
        // 인스턴스 초기화 작업  
    }  
    Card(String kind, int number) { // 매개변수가 있는  
        생성자  
        // 인스턴스 초기화 작업  
    }  
}
```

13] 기본 생성자 (default constructor) I

▶ 기본 생성자란

- 매개변수가 없는 생성자
- 클래스에 생성자가 하나도 없으면 컴파일러가 기본 생성자를 추가.
- (생성자가 하나라도 있으면 컴파일러는 기본 생성자를 추가하지 않는다.)

// 컴파일러에 의해 추가된 Card클래스의 기본 생성자. 내용이 없다
클래스이름() {} Card() {}

```
public class Constructor {  
    int i; int j;  
    public Constructor(){ // 디폴트생성자  
        System.out.println("Constructor()"); this.i=100; this.j=100;  
    }  
    public Constructor(int i){ // 매개변수가 있는 생성자  
        System.out.println("Constructor(int i)"); this.i=i; this.j=100;  
    }  
    public Constructor(int i,int j){  
        System.out.println("Constructor(int i,int j)");  
        this.i=i; this.j=j;  
    }  
}
```

13] 기본 생성자 (default constructor)2

```
public class ConstructorMain {  
    public static void main(String[] args) { //case 1  
        Constructor constructor1 = new Constructor();  
        System.out.println("*****");  
        System.out.println("i="+constructor1.i);  
        System.out.println("j="+constructor1.j);  
        //case2  
        System.out.println("*****");  
        Constructor constructor2=new Constructor(900);  
        System.out.println("i="+constructor2.i);  
        System.out.println("j="+constructor2.j);  
        //case 3  
        System.out.println("*****");  
        Constructor constructor3=new Constructor(899, 898);  
        System.out.println("i="+constructor3.i);  
        System.out.println("j="+constructor3.j);  
        DefaultConstructor dc=new DefaultConstructor();  
    }  
}
```


13] 기본 생성자 (default constructor)3

```
class Box {  
    private int width; // 변수를 private로 선언하여 외부에서 접근을 막는다  
    private int height; // 정보의 은폐 제공  
    private int depth;  
    private int vol;  
    public Box(int a, int b, int c) { // 클래스의 이름과 동일하게 생성자함수 선언  
        width = a; height = b; depth = c;  
    }  
    public int volume() {  
        vol = width * height * depth;  
        return vol;  
    }  
}  
  
class BoxTestDemo {  
    public static void main(String args[]) {  
        int vol;  
        Box mybox1 = new Box(10, 20, 30);  
        vol = mybox1.volume();  
        System.out.println("mybox1 객체의 부피 : " + vol);  
    }  
}
```

13] 멤버변수의 초기화

▶ 멤버변수의 초기화 방법

1. 명시적 초기화(explicit initialization)

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

```
Car c = new Car();  
c.color = "white";  
c.gearType = "auto";  
c.door = 4;
```

→ Car c = new Car("white", "auto", 4);

2. 생성자(constructor)

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

3. 초기화 블록(initialization block)

- 인스턴스 초기화 블록 : { }

- 클래스 초기화 블록 : static { }

[3] 초기화 블록(initialization block)

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

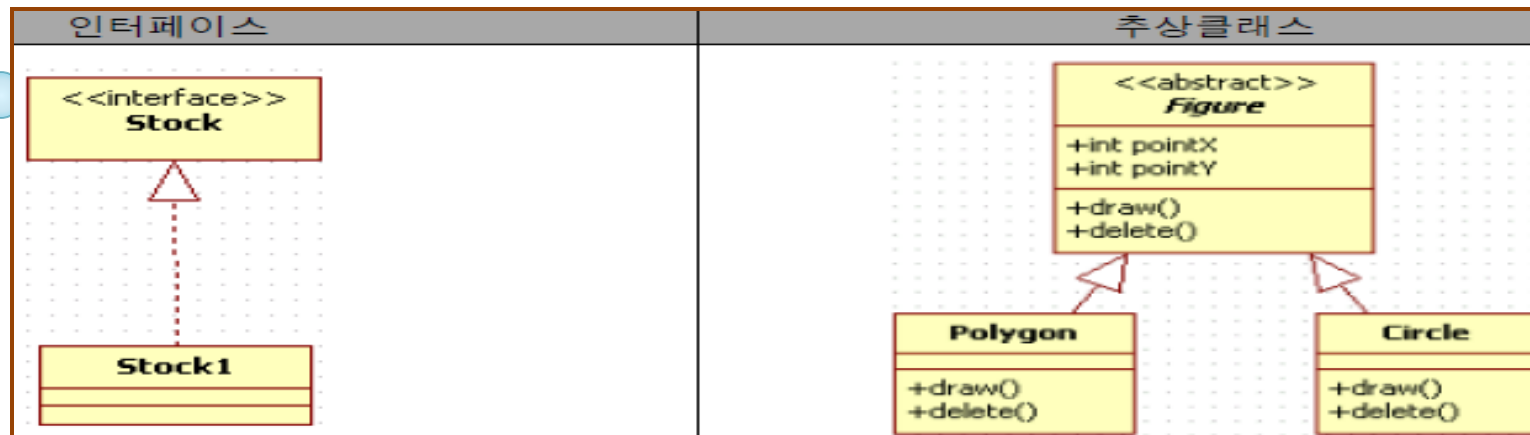
```
class InitBlock {  
    static { /* 클래스 초기화블록 입니다. */ }  
  
    { /* 인스턴스 초기화블록 입니다. */ }  
  
    // ...  
}
```

```
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 }
```

[4] 추상클래스 개념

1. 정의: 객체를 생성할 수 있는 공통 개념만을 표현하기 위한 인스턴스를 만들 수 없는 클래스
 - 하위 클래스를 만들기 위한 상위 클래스로서의 역할을 주목적으로 함
2. 특징:
 - 하나 이상의 추상 메소드 포함, 일반적인 메소드와 추상메소드 포함 가능, Abstract 클래스명으로 표기
3. 상속: 자바에서 단일 클래스에서만 상속 가능, 상속받을 때 extends 사용
 - 인터페이스와 추상클래스는 하위클래스에서 모든 추상 메소드를 구현하고 생성자를 갖지 못한다는 점에서는 동일하지만 추상클래스는 추상메소드 이외에 일반변수와 메소드를 갖는 점에서 차이점 있음

[4] 인터페이스와 추상클래스 도식 비교



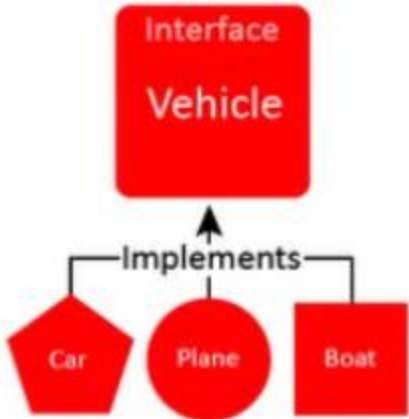
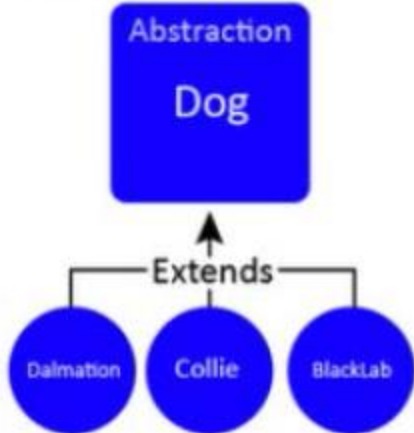
나. 인터페이스와 추상 클래스 구현방법(코딩) 비교

인터페이스	추상클래스
<pre> public interface Stock { public void sell(); public void buy(); } public class Stock1 implements Stock { public void sell(); public void buy(); } </pre>	<pre> public abstract class Figure { public Object int pointX; public Object int pointY; public void draw() { } public void delete() { } } public class Polygon extends Figure { public void draw() { } public void delete() { } } public class Circle extends Figure { public void draw() { } public void delete() { } } </pre>
Interface '이름' 으로 정의 Implements로 하위클래스에서 상속	abstract class '이름' 으로 정의 extends로 하위클래스에서 상속

[4] 인터페이스와 추상 클래스 사용 가이드라인

- ① 인터페이스: 추상 클래스와 달리 몸통을 갖춘 일반 메소드 또는 멤버변수를 구성원으로 가질 수 없음
오직 추상 메소드와 상수만을 멤버로 가짐
개발시간을 단축하고 표준화 가능,
서로 관련없는 클래스들의 관계형성 가능
독립적 프로그래밍 가능
- ② 추상클래스: 추상 클래스를 이용해 객체 생성 불가능,
추상 클래스를 상속한 클래스는 반드시 추상 메소드를 구현해야함
[추상 클래스는 인터페이스보다 느슨한 추상화를 제공하며
일반적인 클래스의 멤버를 가질 수 있음]

[4] 인터페이스와 추상클래스 사용 가이드라인

구분	인터페이스	추상클래스
개념도		
변수	상수(Static Final)만 가능	일반변수 가능
추상메소드	보유 가능	보유 가능
일반메소드	보유 불가능	보유 가능
다중상속	가능	불가능
장점	<ul style="list-style-type: none"> - 동시 개발로 개발기간 단축 - 표준화 가능 	<ul style="list-style-type: none"> - 재사용 부품을 이용하여 확장 가능 - 점진적 개발 용이