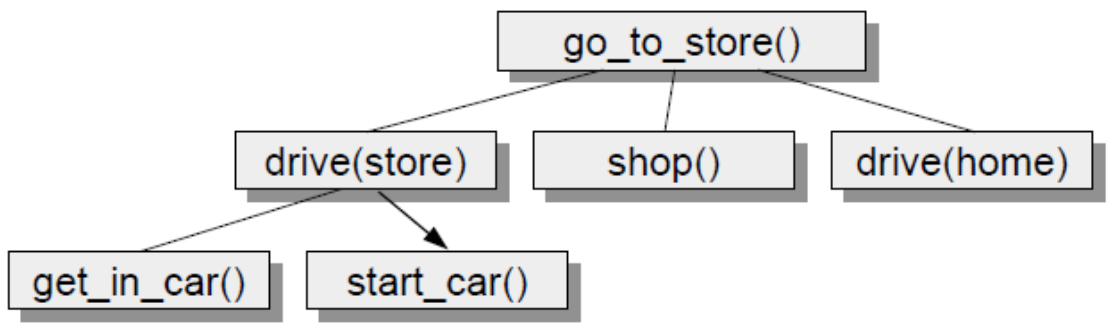
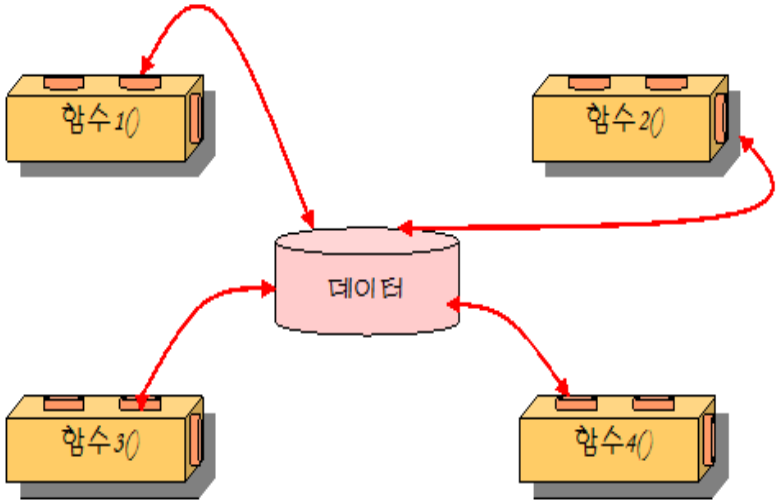
제 7 장
클래스와 객체 Part-1
클래스

절차 지향 vs. 객체 지향

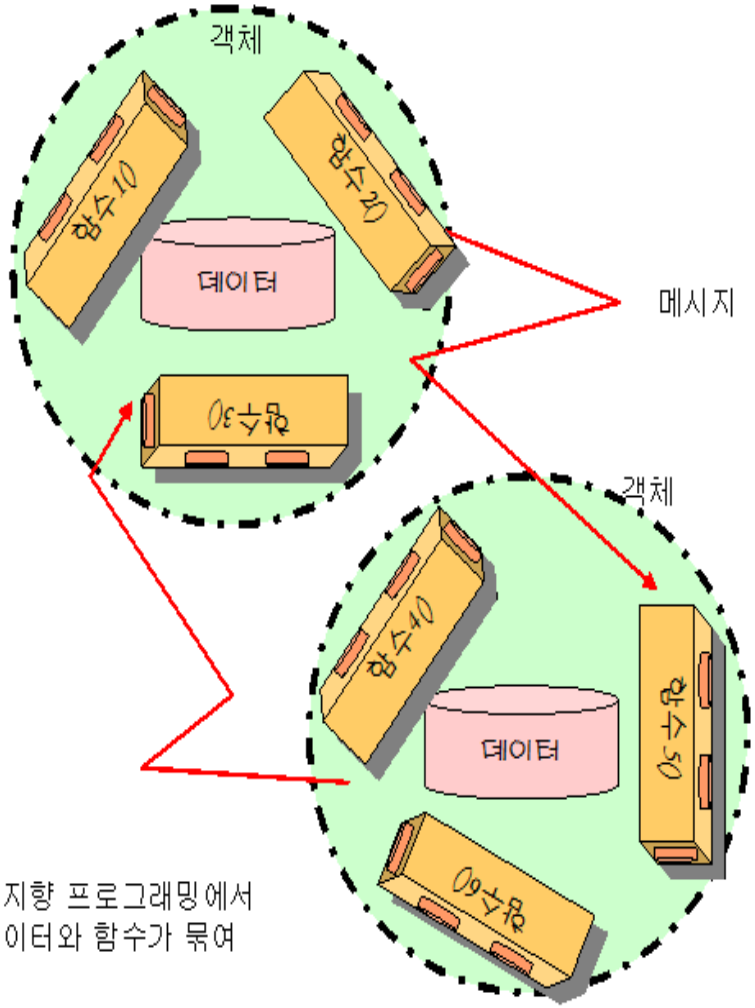
- 절차지향 프로그래밍(procedural programming)
  - 작업의 절차를 중심으로 프로그래밍



- 객체지향 프로그래밍(object-oriented programming)
  - 작업을 객체들간의 상호 소통으로 보고 프로그래밍
  - 신뢰성 있는 소프트웨어를 쉽게 작성할 수 있다
    - 오류가 해당 객체 내부로만 제한되므로
  - 코드를 재사용하기 쉽다 (reuse)
  - 유지보수(maintenamce)성이 좋다
    - 디버깅이 용이
    - 변경 및 수정이 용이

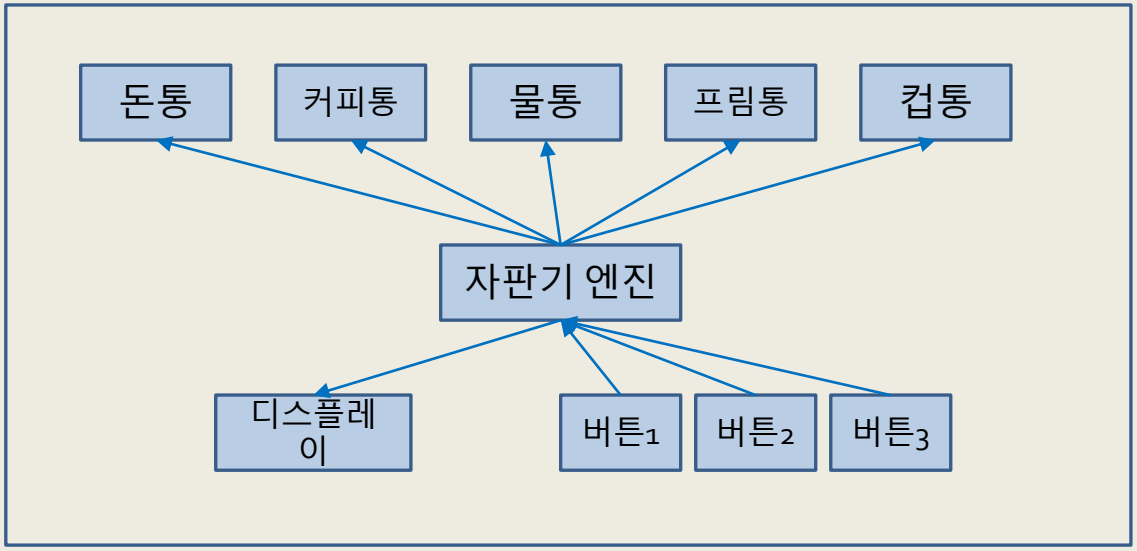


절차 지향 프로그래밍에서는 데이터와 함수가 묶여 있지 않다.

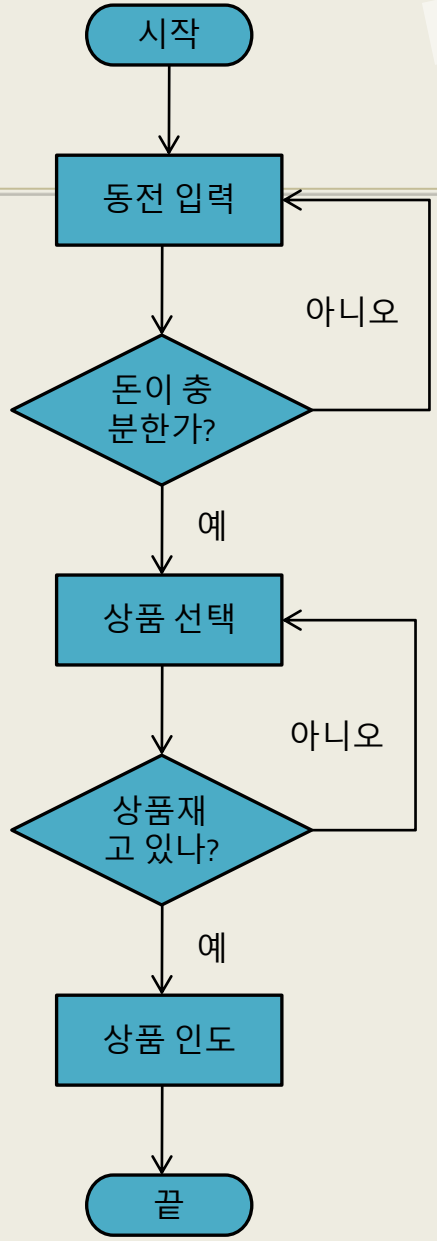


객체 지향 프로그래밍에서는 데이터와 함수가 묶여 있다.

커피 자판기



객체지향적 프로그래밍의 객체들의 상호 관련성



절차지향적 프로그래밍의 실행 절차

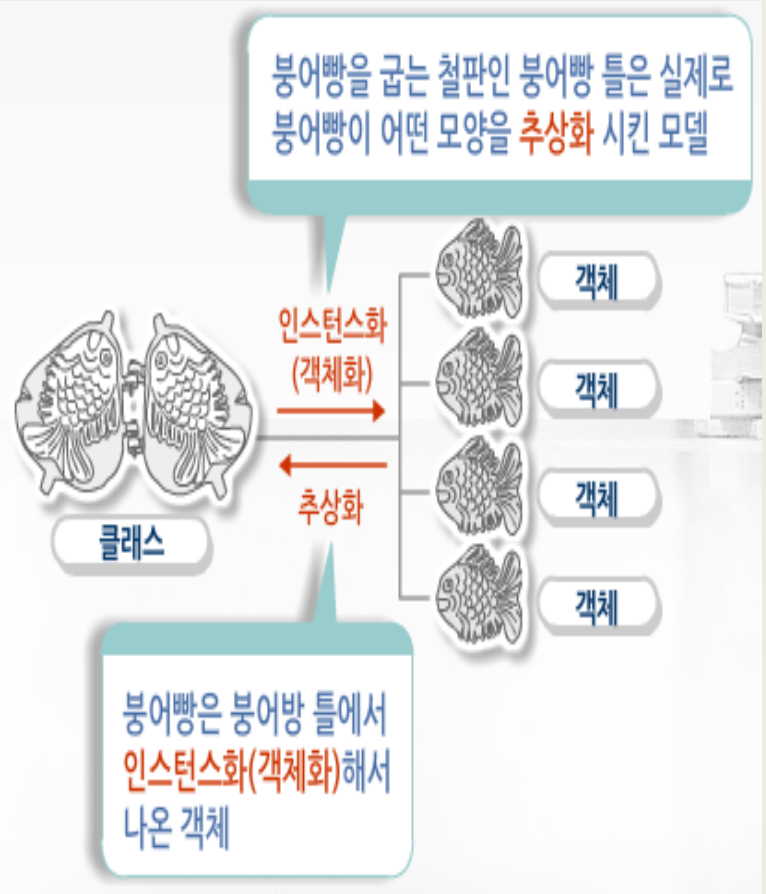
- 객체지향 프로그래밍 과정
  - **객체의 발견 및 분류**
    - **공통 특성의 객체들을 클래스화**
  - **작업을 위해 필요한 주요 필드와 메소드 선정**
    - **각 클래스에 대하여**
  - **작업의 실행 클래스 결정**
  - **작업의 순서를 객체간 메시지 전달 과정을 통해 수행**

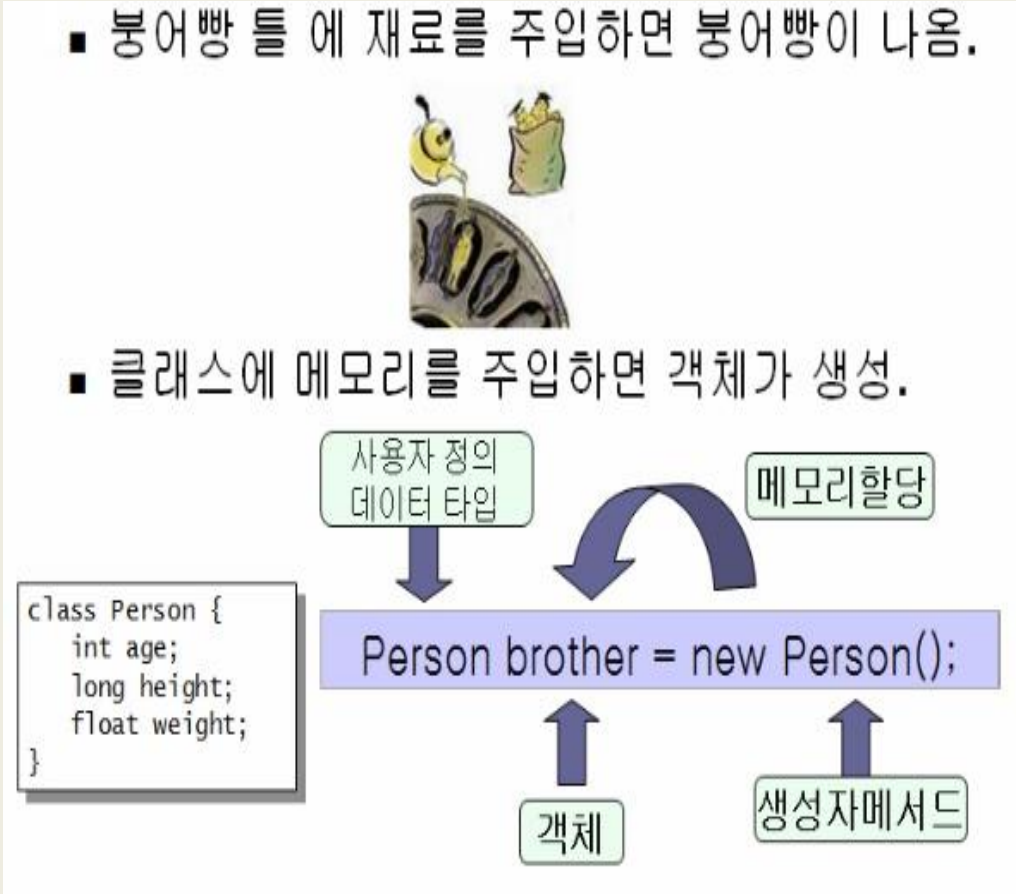
- 객체지향 프로그래밍의 장점
  - 코드 재사용이 용이
  - 업그레이드 및 디버깅이 용이
  - 신뢰성이 높은 소프트웨어 제작

- 자바 프로그램의 실행은 **객체**들을 기반으로 그들 간의 **메시지 통신**을 통해 이루어진다
- 그러므로 객체가 존재하여야 하며,
- 객체들은 객체를 생성(인스턴스화)하는 '틀(frame)'로부터 생성되며, 이 '틀'을 **클래스**라고 한다

필요한 객체들을 찾아서, 이들을 생성하는 클래스들을 구성하는 것이 자바 프로그램의 시작

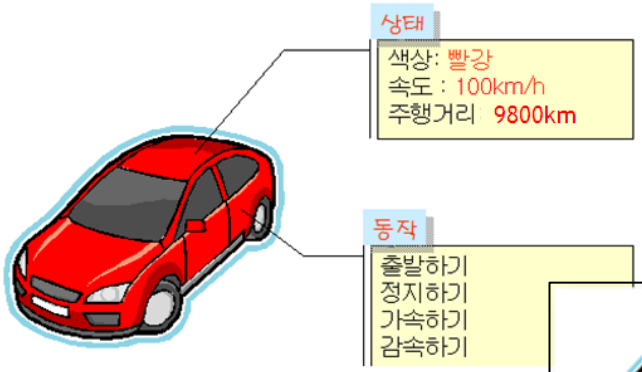
클래스


붕어빵을 굽는 철판인 붕어빵 틀은 실제로
붕어빵이 어떤 모양을 추상화 시킨 모델
객체
객체
객체
객체
인스턴스화
(객체화)
추상화
클래스
붕어빵은 붕어방 틀에서
인스턴스화(객체화)해서
나온 객체

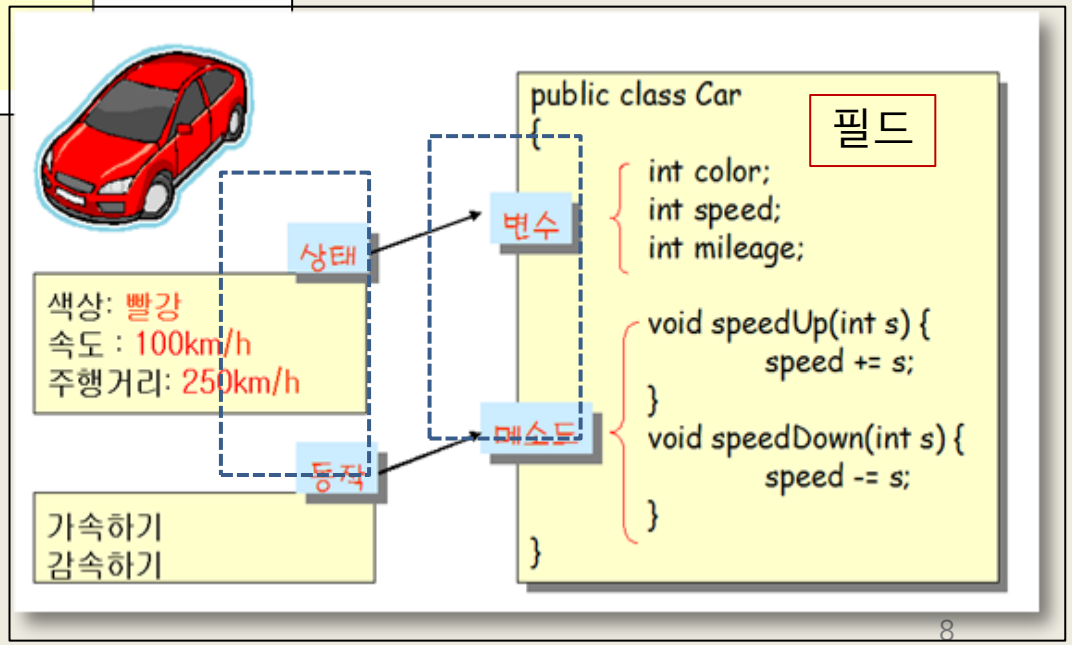

붕어빵 틀 에 재료를 주입하면 붕어빵이 나옴.
클래스에 메모리를 주입하면 객체가 생성.
사용자 정의
데이터 타입
메모리할당
class Person {
int age;
long height;
float weight;
}
Person brother = new Person();
객체
생성자메서드

객체

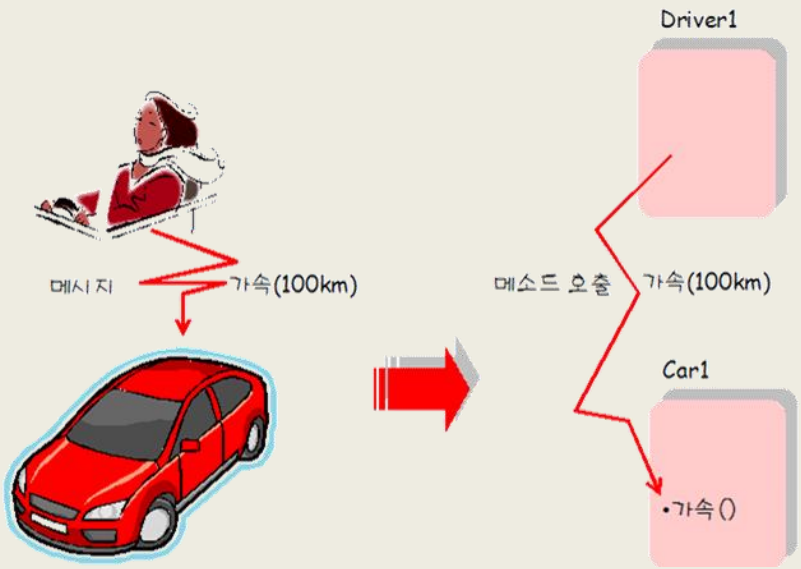
- 객체(object)는 상태와 동작을 가지고 있다.
- 객체의 상태(state)는 객체의 특징값(속성)이다.
- 객체의 동작(behavior) 또는 행동은 객체가 취할 수 있는 동작



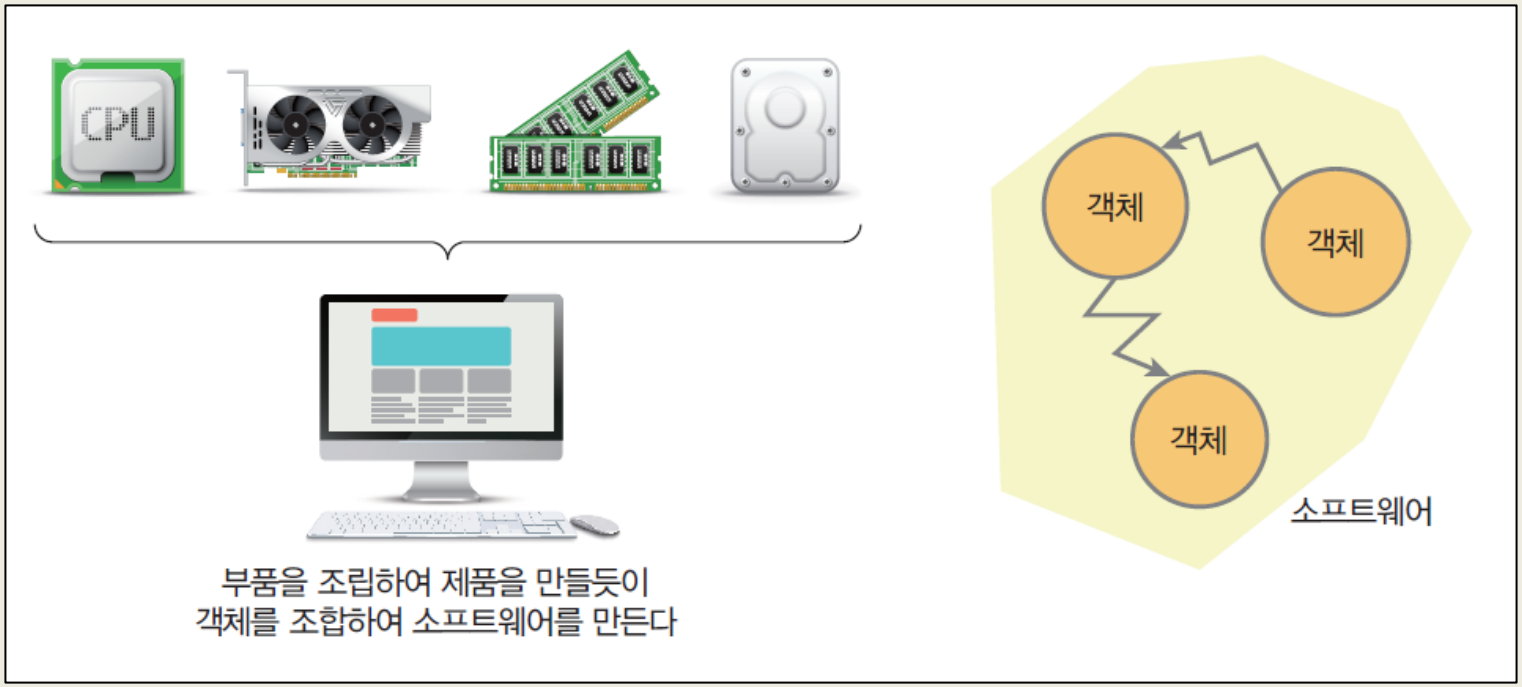
클래스



- 객체 간의 통신
  - 메시지 기반 : 클래스명, 메소드명, 파라메터, 필드 등으로 구성된다
  - 메시지 :
    - ObjectName.methodName(ParamList)
    - ObjectName.field
    - etc…



- 객체 지향으로 소프트웨어를 작성하는 것은 컴퓨터 하드웨어 부품을 구입하여서 컴퓨터를 조립하는 것과 비슷하다.
- Component-Based Software Development (CBSD)



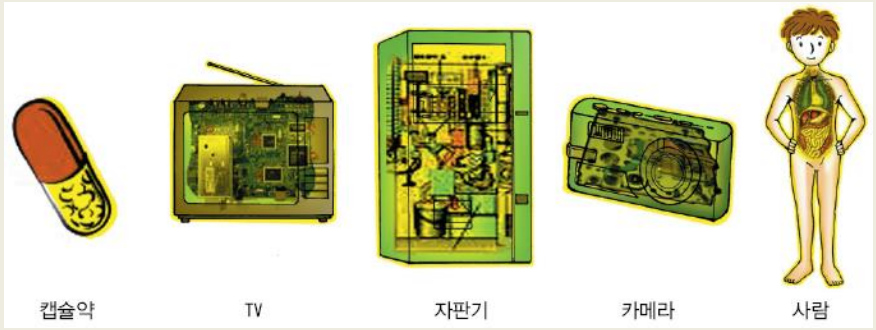
객체 지향 언어의 주요 특성

- 캡슐화 (encapsulation)
  - 정보은폐(Information Hiding)
- 추상화(abstraction)
- 다형성(polymorphism)
  - method overloading
  - method overriding
- 상속(inheritance)

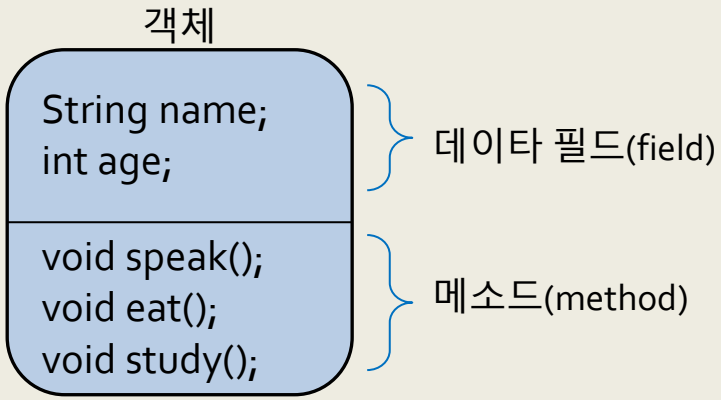
캡슐화(encapsulation)

- 캡슐화(encapsulation) – for **information hiding**(정보 은닉)
  - 메소드(함수)와 데이터를 클래스 내에 정의하고 구현
  - 외부에서는 공개된 메소드의 인터페이스만을 접근할 있음
    - 외부에서는 비공개 데이터에 직접 접근하거나 메소드의 구현 세부를 알 수 없다
  - 객체 내 데이터에 대한 보안, 보호, 외부 접근 제한 → 정보은닉

실세계의 캡슐화

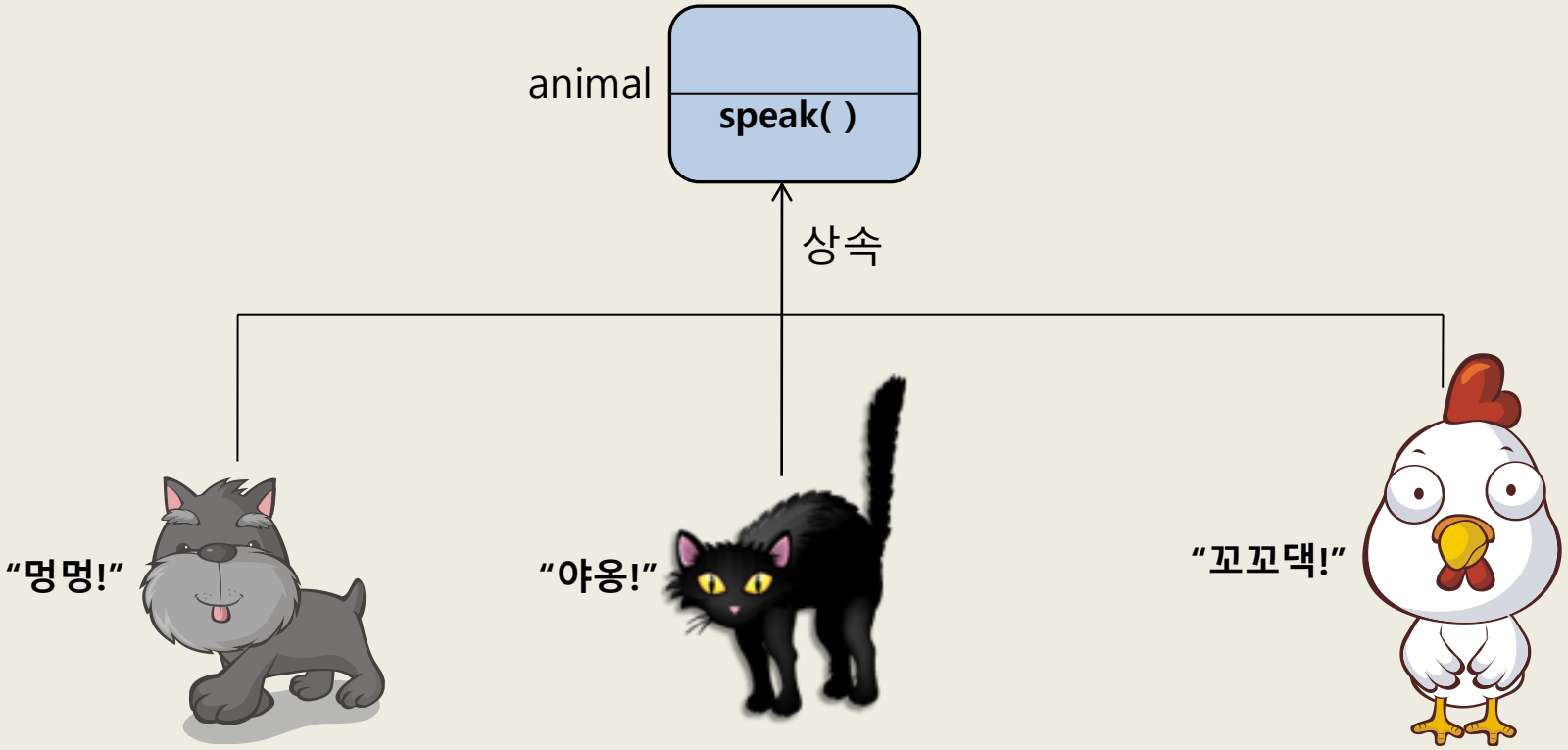


자바 객체의 캡슐화



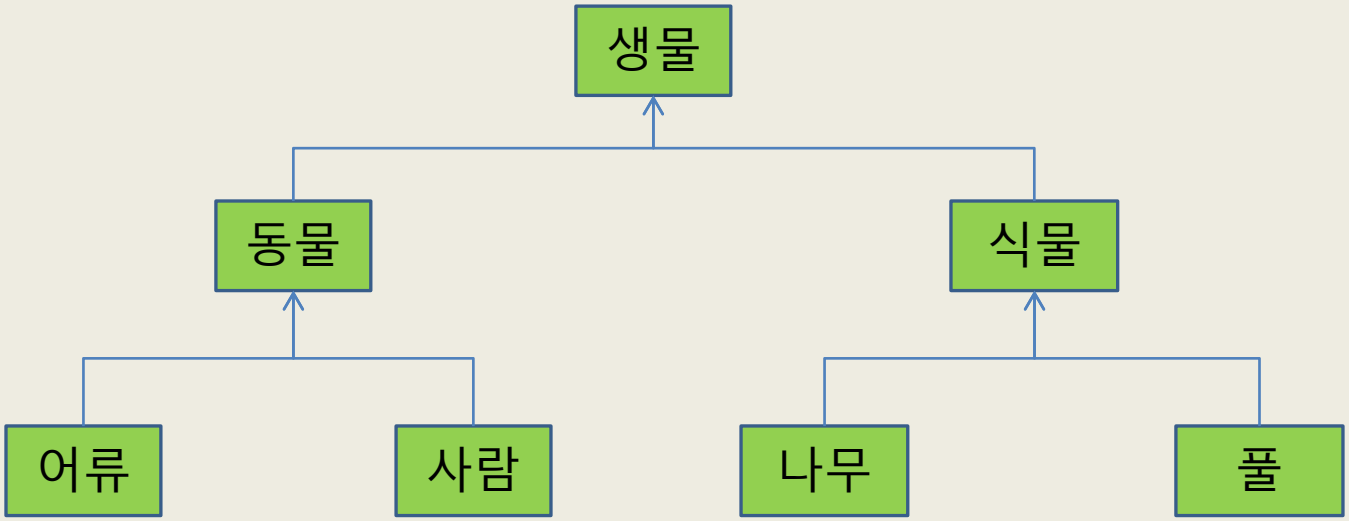
다형성(Polymorphism)

- 다형성(polymorphism)-overloading & overriding
  - 동일한 이름의 메시지 또는 메소드가 객체에 따라 다른 동작 가능
    - 동일한 이름으로 서로 다른 동작을 구현할 수 있다
  - 다형성은 메소드 오버로딩, 오버라이딩과 밀접한 관계가 있음



상속(inheritance)

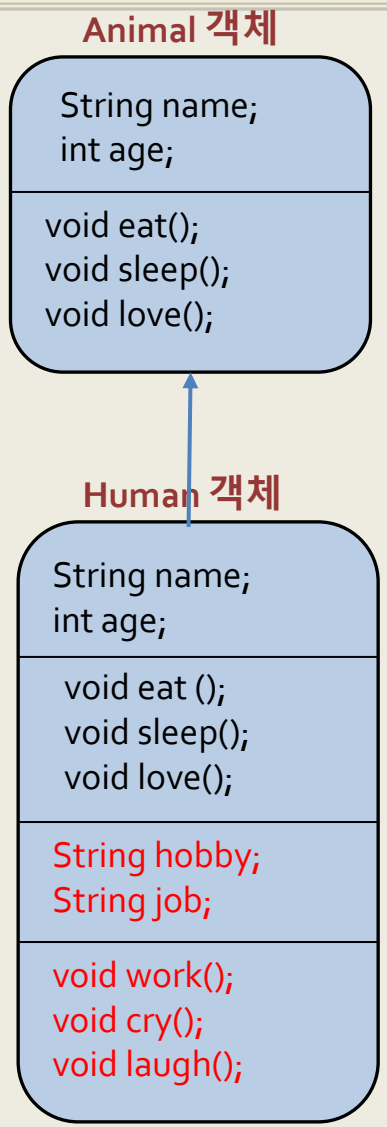
- 상속(inheritance): 이미 설계된 클래스(부모 클래스)의 특성을 그대로 받아서 새로운 클래스(자식 클래스)를 설계하는 기법
- 기존의 코드를 재활용하기 위한 기법



```
class Animal {
    String name;
    int age;
    void eat() {…}
    void sleep() {…}
    void love() {…}
}
```

상속

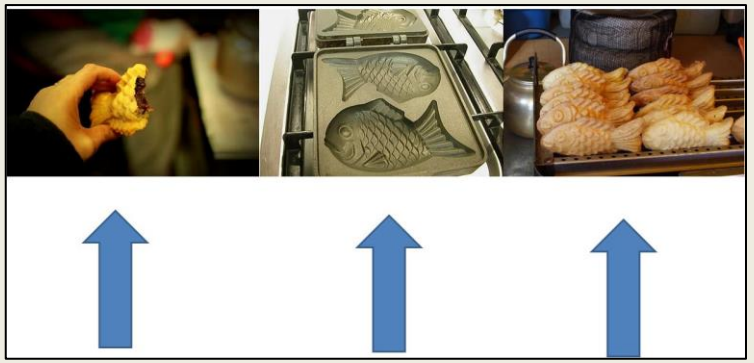
```
class Human extends Animal {
    String hobby;
    String job;
    void work() {…}
    void cry() {…}
    void laugh() {…}
}
```



- 상속 : 상위 클래스의 특성을 하위 클래스가 그대로 물려받음
  - 상위 클래스 : **수퍼** 클래스, 하위 클래스 : **서브** 클래스
  - 서브 클래스
    - 수퍼 클래스 코드 재사용
    - 새로운 특성 추가 가능
  - 다중 상속은 지원하지 않음
    - 인터페이스를 통해 다중 상속과 같은 효과를 얻을 수는 있다

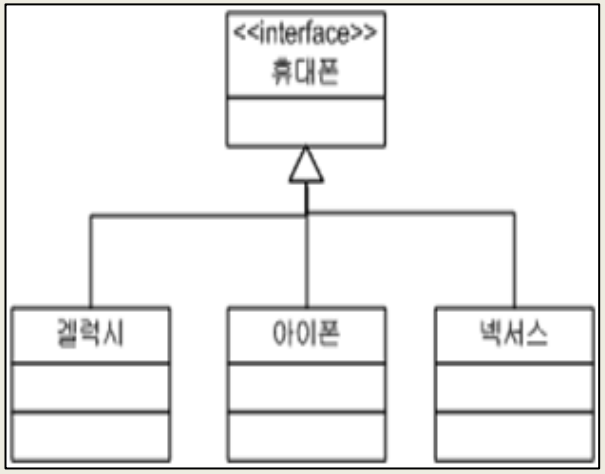
클래스와 객체

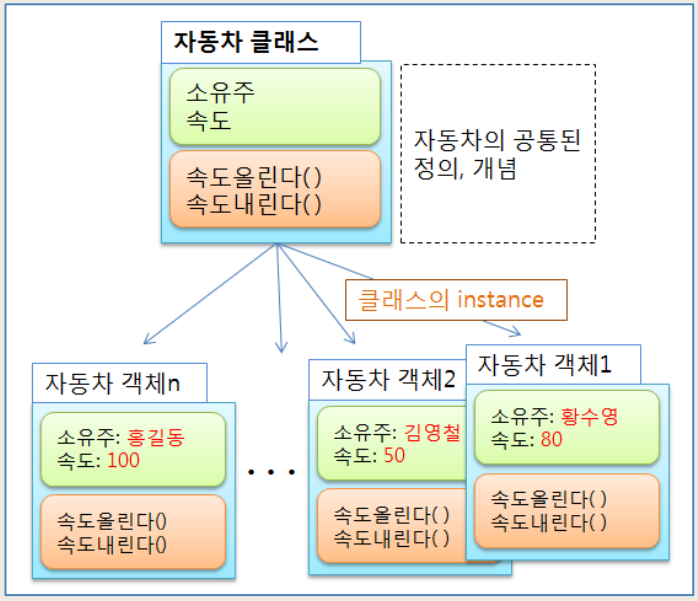
- 클래스 → 객체를 생성하는 frame
  - 객체의 공통된 특징을 기술한 것
  - 객체의 특성과 행위를 정의
- 객체 → 클래스로부터 생성
  - 메인 메모리를 점유
    - 인스턴스(instance)라고도 함
    - 보통 객체와 인스턴스는 같은 뜻으로 사용

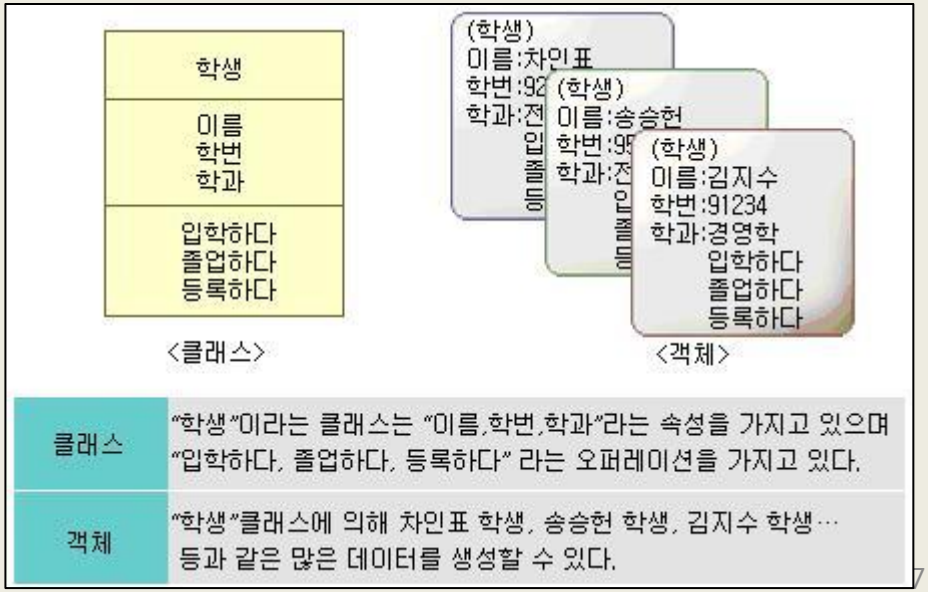


- 사례
  - 클래스: 소나타자동차, 객체: 출고된 실제 소나타 100대
  - 클래스: 벽시계, 객체: 우리집 벽에 걸린 벽시계들
  - 클래스: 책상, 객체: 우리가 사용중인 실제 책상들

- 세상의 모든 사물은 자신이 속한 클래스의 실체이다


<<interface>>
휴대폰
갤럭시
아이폰
넥서스


자동차 클래스
소유주
속도
속도올린다()
속도내린다()
자동차의 공통된 정의, 개념
클래스의 instance
자동차 객체n
소유주: 홍길동
속도: 100
속도올린다()
속도내린다()
자동차 객체2
소유주: 김영철
속도: 50
속도올린다()
속도내린다()
자동차 객체1
소유주: 황수영
속도: 80
속도올린다()
속도내린다()


학생
이름
학번
학과
입학하다
졸업하다
등록하다
<클래스>
(학생)
이름:김지수
학번:91234
학과:경영학
입학하다
졸업하다
등록하다
<객체>
클래스
"학생"이라는 클래스는 "이름,학번,학과"라는 속성을 가지고 있으며 "입학하다, 졸업하다, 등록하다" 라는 오퍼레이션을 가지고 있다.
객체
"학생"클래스에 의해 차인표 학생, 송승헌 학생, 김지수 학생… 등과 같은 많은 데이터를 생성할 수 있다.

클래스: 사람

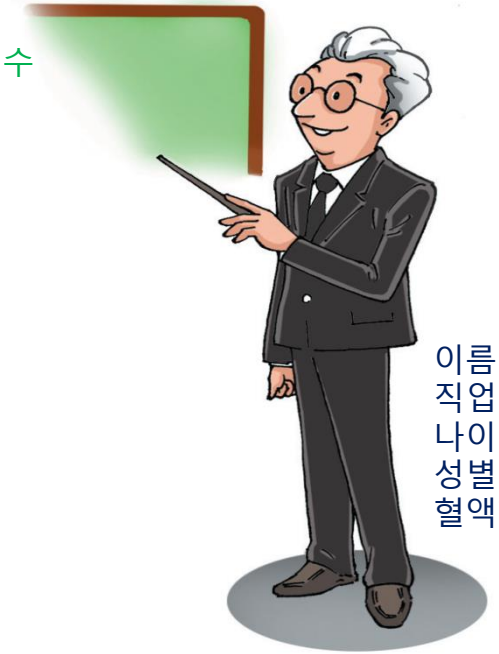
이름, 직업,나이, 성별, 혈액형
밥 먹기, 잠자기, 말하기, 걷기

이름 최승희
직업 의사
나이 45
성별 여
혈액형 A

객체 : 최승희

이름 이미녀
직업 골프선수
나이 28
성별 여
혈액형 O

객체 : 이미녀



이름 김미남
직업 교수
나이 47
성별 남
혈액형 AB

객체:김미남

클래스 선언

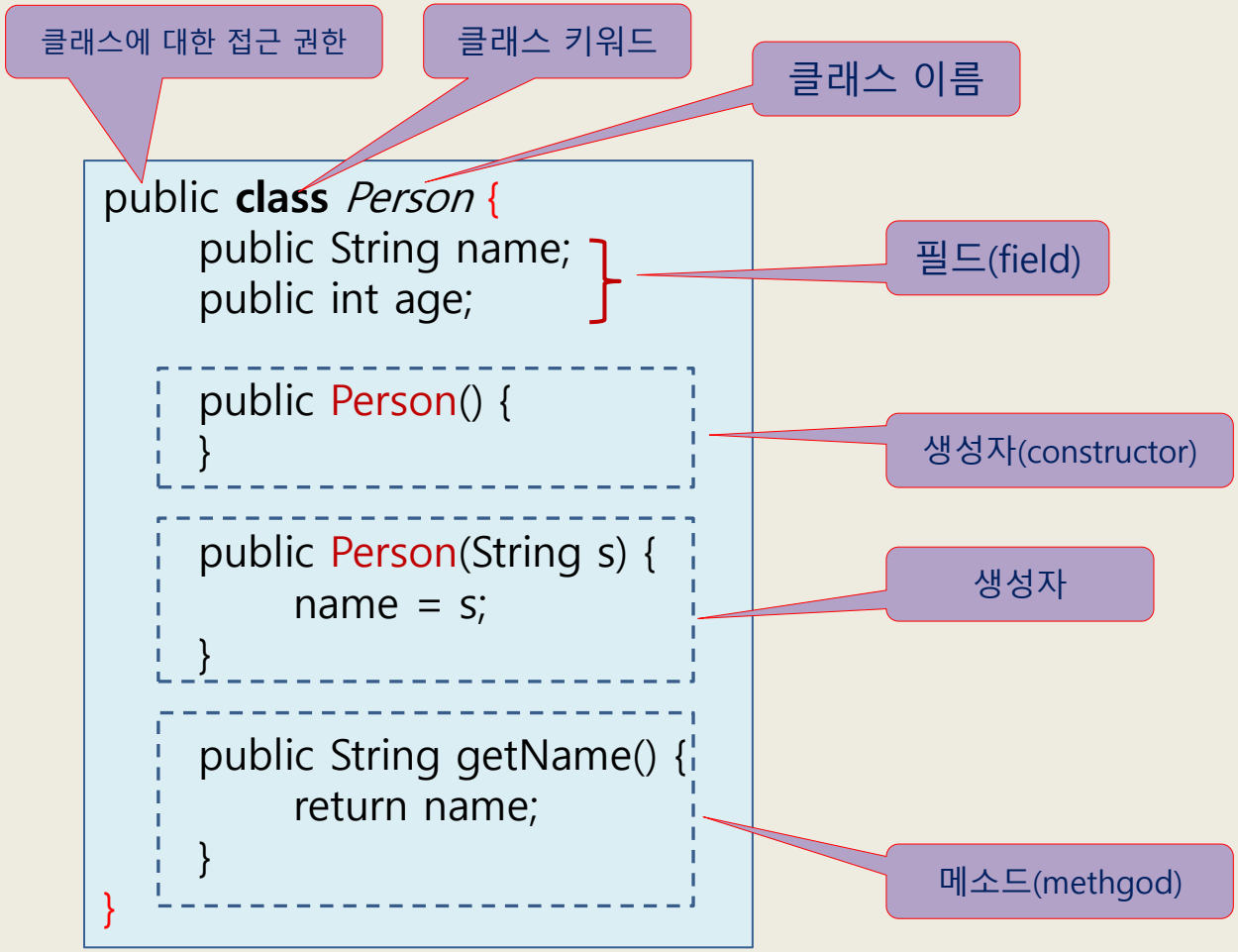
```
class 클래스이름 {
  // 필드(field)와 생성자(constructor) 선언
  // 메소드(method) 선언
     .....
}
```

클래스 헤더

클래스 몸체

```
접근지정자 class 클래스이름 extends 수퍼클래스이름 implements 인터페이스이름 {
  // 필드(field)와 생성자(constructor) 선언
  // 메소드(method) 선언
     ......
}
```

- 클래스의 구조
  - public을 선택적으로 포함하는 접근지정자
  - 영문 대문자로 시작하는 클래스이름
  - 수퍼클래스로부터 상속이 있을 경우 사용하는 extends 키워드와 해당 클래스의 수퍼클래스 이름 (하나만 허용)
  - 해당 클래스가 구현하고 있는 인터페이스 클래스의 이름과 implements 키워드 (하나 이상 가능하며 ‘,’를 사용하여 2개 이상 표현 가능)
  - 클래스 몸체의 시작과 끝을 알려주는 ‘{’와 ‘}’


클래스에 대한 접근 권한
클래스 키워드
클래스 이름
public class Person {
public String name;
public int age;
public Person() {
}
public Person(String s) {
name = s;
}
public String getName() {
return name;
}
}
필드(field)
생성자(constructor)
생성자
메소드(methgod)

- 클래스 접근 권한, *public*
  - 다른 모든 클래스들이 해당 클래스의 필드 및 메소드에 대해 접근이 가능함을 의미
- *class* Person
  - Person이라는 이름의 클래스 정의
  - class 다음에 클래스의 이름을 선언
  - 클래스는 '{ ' 로 시작하여 '}' 로 닫으며 이곳에 모든 필드와 메소드를 구현
- 필드(field) : cf) 지역변수, 매개변수
  - 클래스 내부의 모든 메소드들이 사용할 수 있는 변수
  - 멤버 변수 혹은 필드라고 함
  - 필드 앞에 붙은 접근 지정자 public은 이 필드가 다른 클래스에서 접근할 수 있도록 공개한다는 의미
- 생성자(constructor)
  - 클래스의 이름과 동일한 메소드
  - 클래스의 객체가 생성될 때 한번만 호출되는 메소드
- 메소드(method)
  - 메소드는 실행 가능한 함수이며 객체의 행위를 구현
  - 메소드 앞에 붙은 접근 지정자 public은 이 메소드가 다른 클래스에서 접근될 수 있도록 공개한다는 의미

클래스로부터 객체 생성

- 객체 생성
  - 객체는 **new** 키워드를 이용하여 생성
    - New는 **객체의 생성자 호출**
- 객체를 생성하는 두 단계
  - 객체에 대한 레퍼런스 변수 선언
  - 객체 생성

```
public static void main (String args[]) {
        Person aPerson;                    // 레퍼런스 변수 aPerson 선언
        aPerson = new Person("김미남");    // Person 객체 생성

        aPerson.age = 30;                  // 객체 멤버 접근
        int i = aPerson.age;               // 30
        String s = aPerson.getName();      // 객체 메소드 호출
}
```

객체 생성

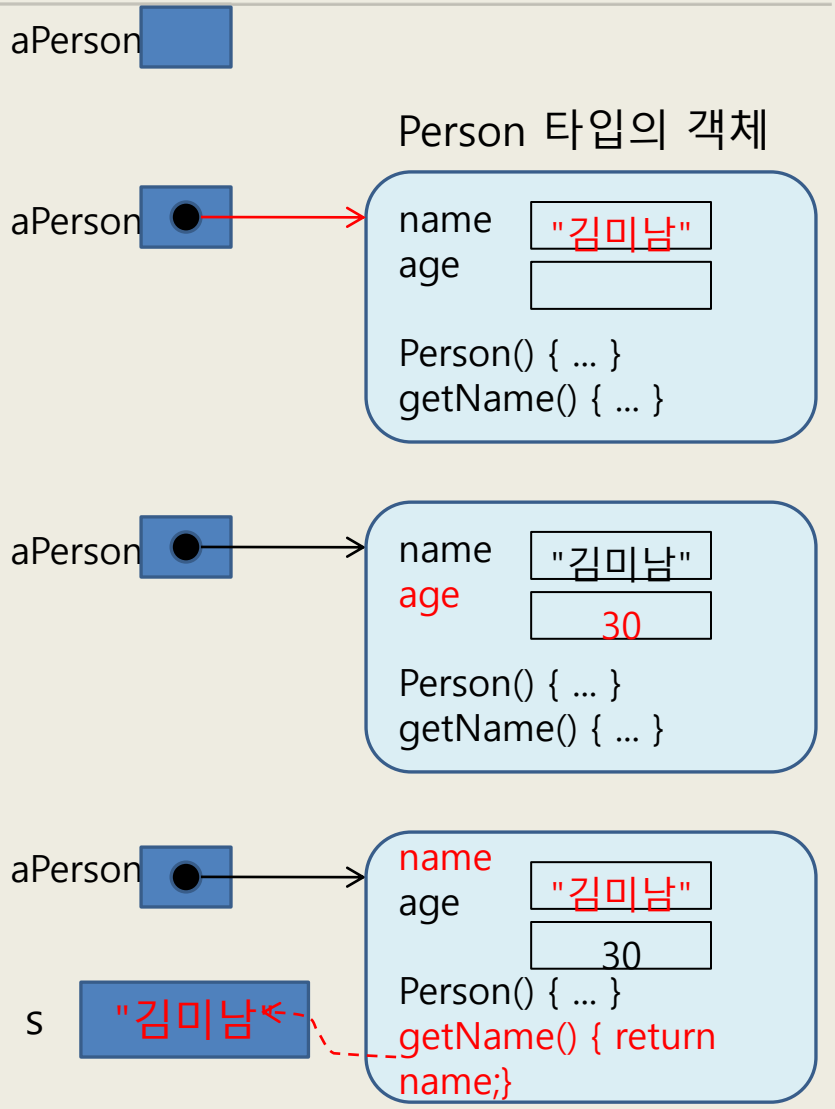
(1) Person aPerson;

(2) aPerson = new Person("김미남");

객체 사용

(3) aPerson.age = 30;

(4) String s = aPerson.getName();



객체의 멤버 접근

- 객체의 멤버 접근 : ' . ' 사용하여 접근

해당객체참조변수.멤버

```
public class ClassExample {
        public static void main (String args[]) {
                Person aPerson = new Person("홍길동");

                aPerson.age = 30;
                int i = aPerson.age;
                String s = aPerson.getName();
        }
}
```

객체의 필드에 값 대입

객체의 필드에서 값 읽기

객체의 메소드 호출

- **하나의 클래스로부터 여러 개의 객체 생성**

- **public class ClassExample {**
  - **public static void main (String args[]) {**
    - Person aPerson = **new Person("홍길동");**
    - **Person bPerson = new Person("손오공");**
    - **Person cPerson = new Person("관운장");**

    - aPerson.age = 30;
    - bPerson.age = 20;
    - cPerson.age = 60;

    - **int i = aPerson.age;**
    - String s = aPerson.getName();
  - }
- }

예제: 상품을 표현하는 클래스 Goods 만들기

상품 하나를 표현하는 클래스 Goods를 작성하라.

- 상품은 String 타입의 name, - int 타입의 price, numberOfStock, sold 등 네 개의 필드
- Goods 클래스 내에 main() 메소드를 작성하여 Goods 객체를 하나 생성하고 이 객체에 대한 레퍼런스 변수 명을 camer로 한다
- camera의 상품 이름(name 필드)을 "Nikon", 값(price)을 400000, 재고 갯수(numberOfStock)를 30, 팔린 개수(sold)를 50으로 설정하라.
- 설정된 이들 값을 화면에 출력하라.

```
public class Goods {
    String name;
    int price;
    int numberOfStock;
    int sold;

    public static void main(String[] args) {
        Goods camera = new Goods();

        camera.name = "Nikon";
        camera.price = 400000;
        camera.numberOfStock = 30;
        camera.sold = 50;

        System.out.println("상품 이름:" + camera.name);
        System.out.println("상품 가격:" + camera.price);
        System.out.println("재고 수량:" + camera.numberOfStock);
        System.out.println("팔린 수량:" + camera.sold);
    }
}
```

```
상품 이름:Nikon
상품 가격:400000
재고 수량:30
팔린 수량:50
```

예제 : 지수 클래스 MyExp 만들기

MyExp는 $base^{exp}$ 지수값을 구하는 클래스

- 두 개의 정수형 멤버 필드 base와 exp를 가진다. 2^3의 경우 base는 2이며, exp는 3이 된다. base와 exp는 양의 정수만을 가지는 것으로 가정한다.
- MyExp는 정수값을 리턴하는 getValue()라는 멤버 메소드를 제공한다. getValue()는 base와 exp 값으로부터 지수를 계산하여 정수 값으로 리턴한다. 예를 들어 MyExp객체의 base 필드가 2이고 exp가 3이라면 getValue()는 8을 리턴한다.

```
public class MyExp {
    int base;
    int exp;
    int getValue() {
        int res=1;
        for(int i=0; i<exp; i++)
            res = res * base;
        return res;
    }
    public static void main(String[] args) {
        MyExp number1 = new MyExp();
        number1.base = 2;
        number1.exp = 3;
        MyExp number2 = new MyExp();
        number2.base = 3;
        number2.exp = 4;

        System.out.println("2의 3승 = " + number1.getValue());
        System.out.println("3의 4승 = " + number2.getValue());
    }
}
```

```
2의 3승 = 8
3의 4승 = 81
```

객체 배열

- 객체들로 구성된 배열의 생성 과정

```
Person[] pa;
pa = new Person[10];
for (int i=0;i<pa.length;i++) {
   pa[i] = new Person();
   pa[i].age = 30 + i;
}
```

객체 배열을 위한 레퍼런스 선언

레퍼런스 배열 생성

객체 생성

객체 배열 사용

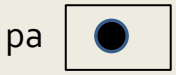
```
for (int i=0;i<pa.length;i++)
    System.out.print(pa[i].age+" ");
```

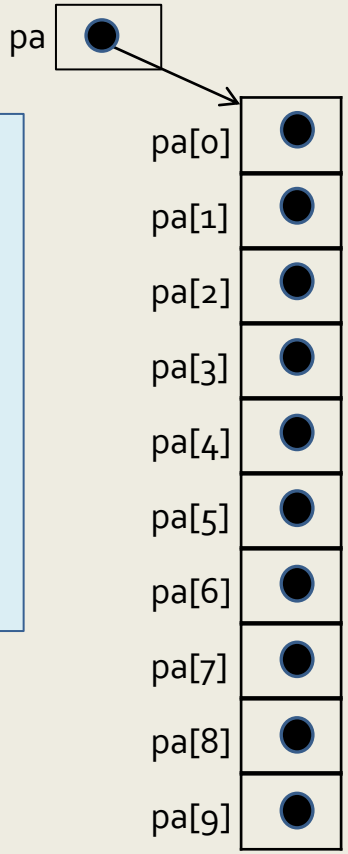
객체 배열 선언과 생성 예

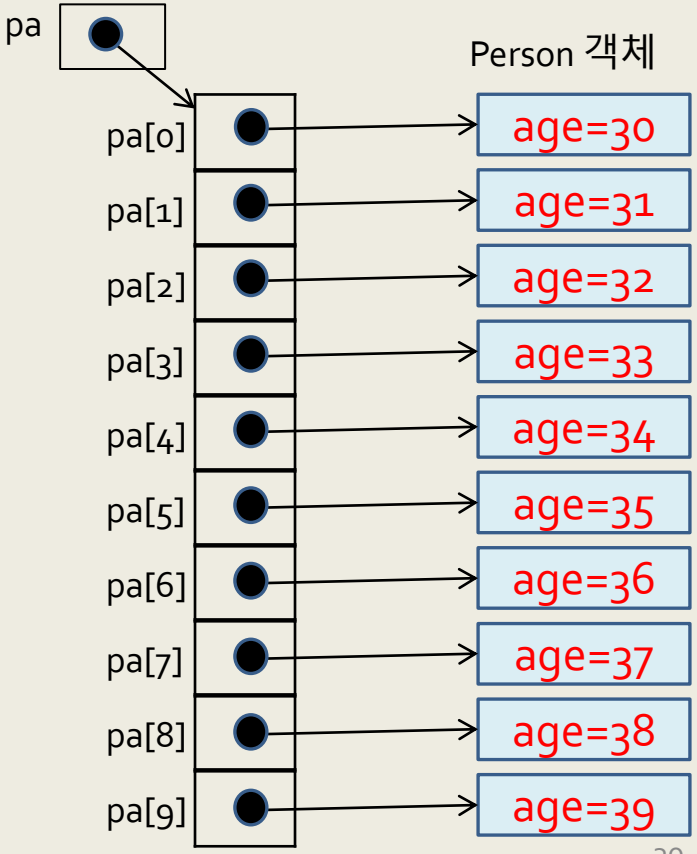
```
Person[] pa;
```

```
pa = new Person[10];
```

```
for (int i=0;i<pa.length;i++) {
    pa[i] = new Person();
    pa[i].age = 30 + i;
}
```








```
public static void main(String [] args) {
    Person[] pa;
    pa = new Person[10];
    for (int i=0;i<pa.length;i++) {
        pa[i] = new Person();
        pa[i].age = 30 + i;
    }

    for (int i=0;i<pa.length;i++)
        System.out.print(pa[i].age+" ");
}
```

```
30 31 32 33 34 35 36 37 38 39
```

예제 : 객체 배열 생성

java.util.Scanner 클래스를 이용하여 상품을 입력 받아 Goods 객체를 생성하고 이들을 Goods 객체 배열에 저장하라. 상품 즉 Goods 객체를 3개 입력 받으면 이들을 모두 화면에 출력하라.

```
import java.util.Scanner;

public class GoodsArray {
    public static void main(String[] args) {
        Goods[] goodsArray;
        goodsArray = new Goods[3];

         Scanner s = new Scanner(System.in);
        for(int i=0; i<goodsArray.length; i++) {
            String name = s.next();
            int price = s.nextInt();
            int n = s.nextInt();
            int sold = s.nextInt();
            goodsArray[i] = new Goods(name, price, n, sold);
        }

        for(int i=0; i<goodsArray.length; i++) {
            System.out.print(goodsArray[i].getName()+" ");
            System.out.print(goodsArray[i].getPrice()+" ");

    System.out.print(goodsArray[i].getNumberOfStock()+" ");
            System.out.println(goodsArray[i].getSold());
        }
    }
}
```

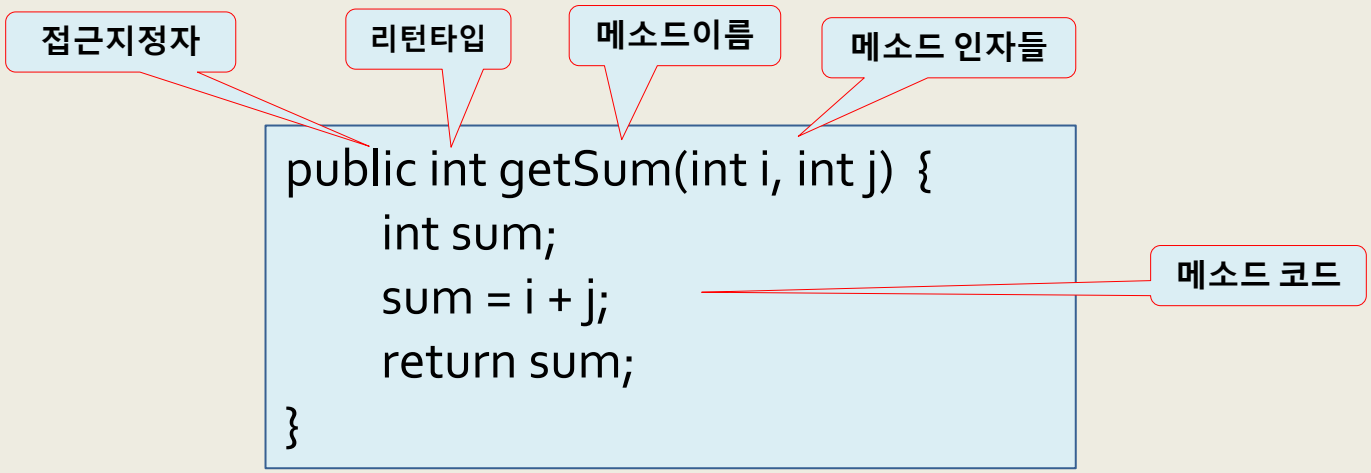
```
class Goods {
    private String name;
    private int price;
    private int numberOfStock;
    private int sold;

    Goods(String name, int price, int numberOfStack, int
sold) {
        this.name = name;
        this.price = price;
        this.numberOfStock = numberOfStock;
        this.sold = sold;
    }

    String getName() {return name;}
    int getPrice() {return price;}
    int getNumberOfStock() {return numberOfStock;}
    int getSold() {return sold;}
}
```

```
콜라 500 10 20
사이다 1000 20 30
맥주 2000 30 50
콜라 500 10 20
사이다 1000 20 30
맥주 2000 30 50
```

키 입력 부분

메소드(method)

- 메소드
  - 메소드는 객체가 실행할 수 있는 함수이며 함수 만드는 방법과 유사
  - 모든 메소드는 반드시 클래스 안에 있어야 함(캡슐화 원칙)
- 메소드 구성 형식
  - **접근 지정자**
    - public. private, protected, default(접근 지정자 생략된 경우)
  - **리턴 타입**
    - 메소드가 반환하는 결과값의 데이터 타입
  - **메소드명, 매개변수, 코드블럭, (예외 리스트, exception list)**



인자 전달 - call by value

- 메소드 호출 시 인자(argument) 전달 방식
  - 값에 의한 호출(call by value)

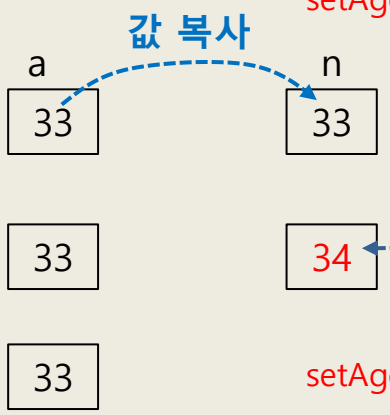
- 기본 데이터 타입의 값을 전달하는 경우
  - 값이 복사되어 전달
  - 메소드의 매개 변수의 값이 변경되어도 호출한 인자의 값은 변경되지 않음

- 객체 혹은 배열을 전달하는 경우
  - 객체나 배열의 레퍼런스 만이 전달됨
    - 객체 혹은 배열이 통째로 복사되어 전달되는 것이 아님
  - 메소드의 매개 변수와 호출한 인자가 객체 혹은 배열을 공유

call by value : 기본 데이터의 값 전달 사례

```
public class CallByValue {
    public static void main (String args[]) {
        Person aPerson = new Person("홍길동");
        int a = 33;


        aPerson.setAge(a);


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

33




```
public class Person {
    public String name;
    public int age;

    public Person(String s) {
        name = s;
    }
```

setAge()가 호출되면 매겨변수 n이 생성된다.

```
    public void setAge(int n) {
        age = n;
        n++;
    }
```

setAge()가 끝나면 n은 사라진다.

```
}
```

call by value : 객체 전달 사례

```
class MyInt {
    int val;
    MyInt(int i) {
        val = i;
    }
}
public class CallByValueObject {
    public static void main (String args[]) {
        Person aPerson = new Person("홍길동");
        MyInt a = new MyInt(33);

        aPerson.setAge(a);


        System.out.println(a.val);
    }
}
```

호출

```
public class Person {
    public String name;
    public int age;
    public Person(String s) {
        name = s;
    }

    public void setAge(MyInt i) {
        age = i.val;
        i.val++;
    }

}
```

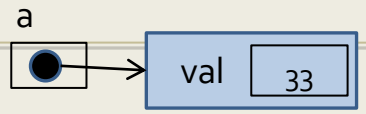
34

*** 객체가 복사되어 전달되는 것이 아님**
객체에 대한 레퍼런스 만이 복사되어 전달

(1)

```
MyInt a = new MyInt(33);
```

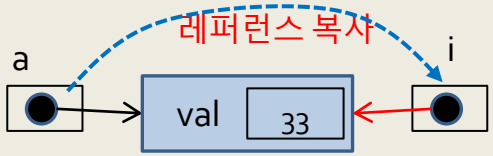
MyInt 타입의 객체 생성



(2)

```
aPerson.setAge(a);
```

레퍼런스 a의 복사본 전달
객체가 복사되는 것은 아님



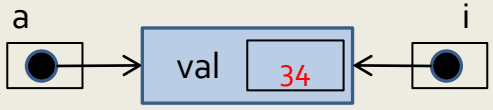
(3) setAge()호출됨

```
public void setAge(MyInt i)
```

매개변수로 레퍼런스 i가 생성
인자로 전달된 레퍼런스 a 값을 복사해 전달받음
i는 a의 객체를 가리키며 a와 i는 서로 동일한 객체를 공유

```
i.val++;
```

(4)

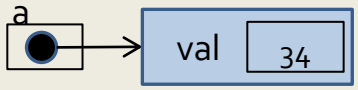


레퍼런스 i 가 가리키는 MyInt 객체의 val 값 1 증가

(5)

```
System.out.println(a.val);
```

34가 화면에 출력



setAge() 메소드가 끝나면 레퍼런스 i가 사라짐

예제 : 배열의 전달

char 배열을 메소드의 인자로 전달하여 배열 속의 공백(' ')문자를 ','로 대치하는 프로그램을 작성하라.

```
public class ArrayParameter {
   static void replaceSpace(char a[]) {
      for (int i = 0; i < a.length; i++)
         if (a[i] == ' ')
            a[i] = ',';
   }
   static void printCharArray(char a[]) {
      for (int i = 0; i < a.length; i++)
         System.out.print(a[i]);
      System.out.println();
   }
   public static void main (String args[]) {
      char c[] = {'T','h','i','s',' ','i','s',' ','a',' ','p','e','n','c','i','l','.'};
      printCharArray(c);
      replaceSpace(c);
      printCharArray(c);
   }
}
```

This is a pencil.
This,is,a,pencil.

메소드 오버로딩

- **메소드 오버로딩(Overloading)**
  - 자바는 메소드 시그너쳐가 서로 다른 메소드들을 구별할 수 있다
  - 한 클래스 내에 두 개 이상의 이름이 같은 메소드가 존재
    - 메소드 이름이 동일하여야 한다.
    - 메소드의 인자가 개수 서로 다르거나, 메소드의 인자 타입이 서로 달라야 한다.
    - 메소드의 이름이 같고 인자의 개수나 타입이 모두 같은데 메소드의 리턴 타입이 다르면 메소드 오버로딩이 성립되지 않으며 컴파일 오류가 발생한다.

```
// 메소드 오버로딩이 성공한 사례
class MethodOverloading {
    public int getSum(int i, int j) {
        return i + j;
    }
    public int getSum(int i, int j, int k) {
        return i + j + k;
    }
    public double getSum(double i, double j) {
        return i + j;
    }
}
```

```
// 메소드 오버로딩이 실패한 사례
class MethodOverloadingFail {
    public int getSum(int i, int j) {
        return i + j;
    }
    public double getSum(int i, int j) {
        return (double)(i + j);
    }
}
```

오버로딩된 메소드의 호출

```
public static void main (String args[]) {
   MethodSample a = new MethodSample();

   int i = a.getSum(1, 2);

   int j = a.getSum(1, 2, 3);

   double k = a.getSum(1.1, 2.2);

}
```

```
public class MethodSample {
   public int getSum(int i, int j) {
      return i + j;
   }

   public int getSum(int i, int j, int k) {
      return i + j + k;
   }

   public double getSum(double i,
double j) {
      return i + j;
   }
}
```