

# 6장. 클래스와 객체



*Visualstudio 2019*



# 객체에 대한 이해

- 세상 모든 것이 객체다.

컴퓨터, 책, 건물, 의자, 자동차, 캡슐 약, TV 등 실세계는 객체들의 집합이다.

컴퓨터 프로그램의 예를 들면 스타크래프트에 등장하는 각 캐릭터들, 테트리스 게임에 나오는 블록들, 한글 워드 프로그램의 메뉴나 버튼 모두 객체이다.

- 객체는 캡슐화된다.

다양한 실세계의 객체들이 자신만의 껍데기로 캡슐화 되어 있는데, TV에 케이스가 없다면 외부의 접촉이나 충격으로부터 보호할 수 없으며, 사람은 피부와 근육으로 혈관, 장기, 뇌등을 보호하고 있다.

- 객체의 일부 요소는 공개된다.

객체들이 서로 정보를 교환하고 통신하기 위해 일부 요소의 공개 노출이 필요하다.

TV의 On/Off 버튼, 밝기 조절, 채널 조절 버튼,  
음량 조절 버튼 등은 사용자의 리코컨등과 통신하기 위해 노출되어 있다.



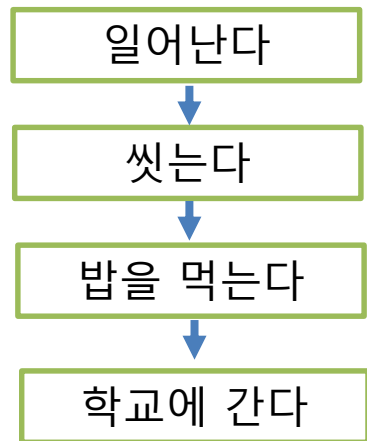
# 객체 지향 프로그래밍

## ■ 객체(Object)란?

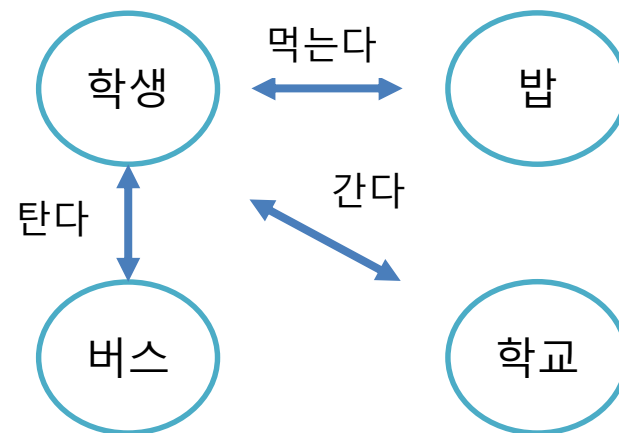
- 의사나 행위가 미치는 대상 -> 사전적 의미
- 구체적, 추상적 데이터 단위 (구체적-책상, 추상적-회사)

## ■ 객체지향 프로그래밍(Object Oriented Programming, OOP)

- 객체를 기반으로 하는 프로그래밍
- 먼저 객체를 만들고, 객체 사이에 일어나는 일을 구현함.



<절차지향 -C언어>



<객체지향 -C++,Java>



# 객체 지향 프로그래밍

## 절차지향 프로그래밍

```
작업 1();  
작업 2();  
작업 3();  
.  
.  
작업 99();  
작업 100();
```

작업(함수) 100개가 동  
등한 위치에서 나열되  
어 있다.

## 객체지향 프로그래밍

```
작업 1();  
작업 2();  
작업 3();  
.  
.  
작업 10();
```

```
작업 1();  
작업 2();  
작업 3();  
.  
.  
작업 10();
```

```
작업 1();  
작업 2();  
작업 3();  
.  
.  
작업 10();
```

연관있는 작업을 객체  
로 묶어서 처리하기때  
문에 보다 효율적으로  
관리할 수 있다.



# 구조체의 진화 -> 클래스

## 상태(State)

```
struct Dog{  
    char name[20];  
    int age;  
    double weight;  
    char color[10];  
    char type[20];  
};
```

(구조체)



객체  
(Object)

## 행위(Behaviour)

```
void bark();  
void eat();  
void run();  
void sleep();
```

(일반함수)

★ 절차지향적 접근 방법

## 클래스

```
class Dog{  
    char name[20];  
    int age;  
    double weight;  
    char color[10];  
    char type[20];  
  
    void bark();  
    void eat();  
    void run();  
    void sleep()  
};
```

★ 클래스는 구조체에 함수를 포함시킨 틀이다.



# 클래스의 구조

- 클래스란?

객체에 대한 속성과 기능을 코드로 구현 한 것

"클래스를 정의 한다"라고 하고, 객체에 대한 설계도 또는 청사진.

- 객체의 속성과 기능

- 객체의 특성(property), 속성(attribute) -> **멤버 변수**
- 객체가 하는 기능 -> **메서드(멤버 함수)**

학생 클래스

- 속성(멤버변수) : 이름, 나이, 학년, 사는 곳 등..
- 기능(함수) : 수강신청, 수업듣기, 시험 보기 등..



# 클래스 정의하기

- 클래스 정의하기

```
class 클래스 이름{  
    멤버 변수;  
    함수;  
}
```

```
class Dog{  
    string type;  
    string color;  
    int age;  
  
public:  
    void bark();  
    void dogInfo();  
    ...  
}
```

- 객체(인스턴스) 생성

```
클래스 이름 객체 변수
```

```
Dog dog
```



# 클래스의 사용

## ▪ Dog 클래스

```
class Dog {  
public:  
    string type;  
    string color;  
    int age;  
  
    void dogInfo() {  
        cout << "종류 : " << type << endl;  
        cout << "색깔 : " << color << endl;  
        cout << "나이 : " << age << endl;  
    }  
};
```

Dog dog; //dog 객체 변수(인스턴스)

```
dog.type = "푸들";  
dog.color = "brown";  
dog.age = 3;
```

```
dog.dogInfo();
```

```
/*cout << "종류 : " << dog.type << endl;  
cout << "색깔 : " << dog.color << endl;  
cout << "나이 : " << dog.age << endl;*/
```

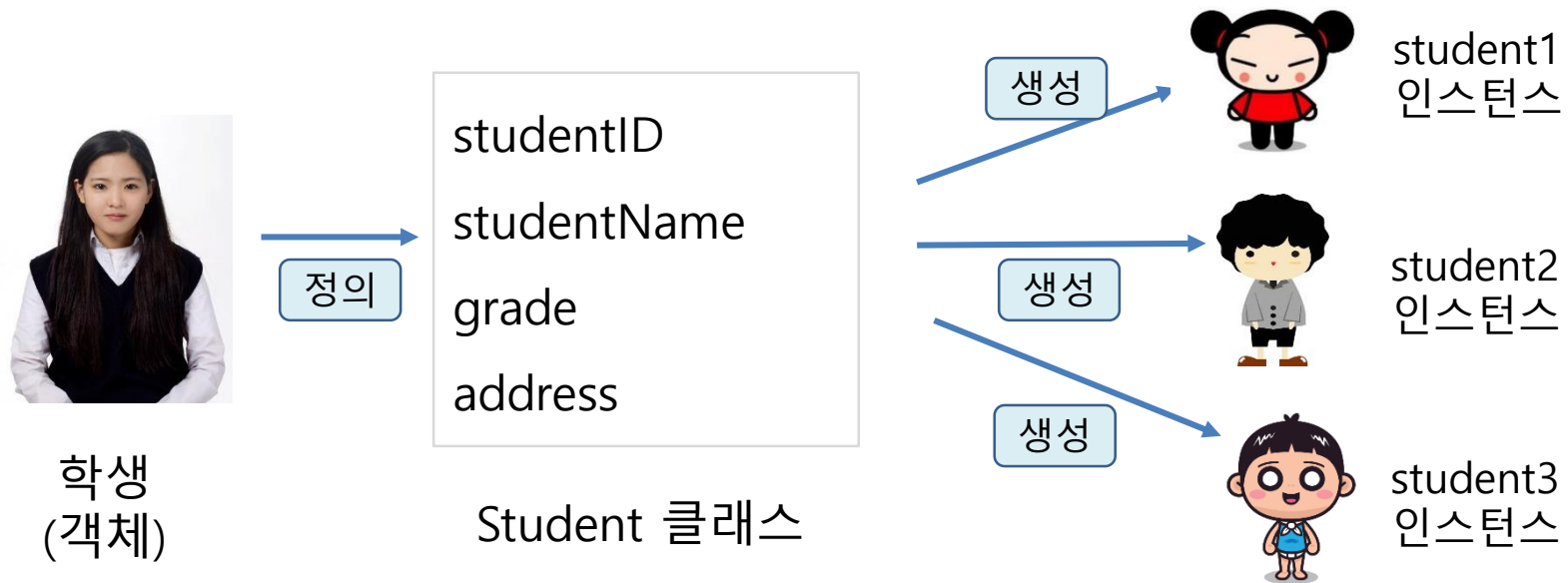




# 클래스와 인스턴스

## ■ 객체, 클래스, 인스턴스

- 객체 : '의사나 행위가 미치는 대상'
- 클래스 : 객체를 코드로 구현한 것
- 인스턴스 : 클래스가 메모리 공간에 생성된 상태.



# 클래스 선언과 구현부 분리

## ■ 클래스 선언과 구현부 분리

```
class Dog {  
public:  
    string type;  
    string color;  
    int age;  
  
    void dogInfo();  
    void bark();  
  
};  
  
void Dog::dogInfo() {  
    cout << "종류 : " << type << endl;  
    cout << "색깔 : " << color << endl;  
    cout << "나이 : " << age << endl;  
}  
  
void Dog::bark() {  
    cout << "멍~ 멍~" << endl;  
}
```

분리하는 이유-클래스의 재사용을 위해서...  
클래스를 사용하는 다른 C++파일에서는 컴파일 시 클래스 선언부만 필요하기 때문이다.

```
Dog dog;  
  
dog.type = "푸들";  
dog.color = "brown";  
dog.age = 3;  
  
dog.dogInfo();  
dog.bark();
```



# 생성자(Constructor)

## ■ 생성자(constructor)

```
class Dog {  
public:  
    string type;  
    string color;  
    int age;  
  
    Dog();    //생성자  
    void dogInfo();  
    void bark();  
  
};  
  
Dog::Dog() {  
    type = "진돗개";  
    color = "white";  
    age = 5;  
}
```

★ 생성자는 객체가 만들어질때 자동으로 호출되는 함수이다.

- 이름이 클래스와 동일하다.
- 생성자는 반환형이 없다.
- 생성자가 정의 되어 있지 않으면 컴파일러가 자동으로 기본 생성자(default constructor)를 제공한다.

```
Dog dog1;  
  
dog1.dogInfo();  
dog1.bark();  
  
cout << "=====" << endl;
```



# 생성자(Constructor)

## ■ 생성자(constructor) 오버 로딩

클래스에 생성자가 두 개 이상 제공되는 경우를 말한다.

이름은 같고, 매개 변수가 다른 생성자를 여러 개 만들수 있다.

```
Dog();    //생성자
Dog(string t, string c, int a); //생성자
void dogInfo();
void bark();
```

```
Dog::Dog(string t, string c, int a) {
    type = t;
    color = c;
    age = a;
}
```

```
Dog dog3 = Dog("치와와", "black", 2);

dog3.dogInfo();
dog3.bark();
```



# 소멸자(destructor)

## ▪ 소멸자(destructor)

- 객체가 소멸될 때 자동으로 호출되는 멤버 함수
- 메모리 해제 등의 폐기물 처리 역할을 한다.

```
Dog();    //생성자  
Dog(string t, string c, int a); //생성자  
~Dog();   //소멸자 - 생략 가능
```

```
void dogInfo();  
void bark();
```

```
Dog::~~Dog() {  
    cout << "Dog 소멸자 호출" << endl;  
}
```

```
종류 : 진돗개  
색깔 : white  
나이 : 5  
멍~ 멍~  
=====  
종류 : 푸들  
색깔 : brown  
나이 : 3  
멍~ 멍~  
=====  
종류 : 치와와  
색깔 : black  
나이 : 2  
멍~ 멍~  
Dog 소멸자 호출  
Dog 소멸자 호출  
Dog 소멸자 호출
```



# 정보 은닉(Information Hiding)

## ❖ 정보 은닉(Information Hiding)

- 접근 제어자 : 접근 권한 지정
- 변수에 대해서는 필요한 경우 **get()**, **set()** 메서드를 제공

| 접근 제어자           | 설 명                                       |
|------------------|---|
| <b>public</b>    | 외부 클래스 어디에서나 접근 할수 있다.                    |
| <b>private</b>   | 같은 클래스 내부 가능, 그 외 접근 불가                   |
| <b>protected</b> | 같은 패키지 내부와 상속 관계의 클래스에서만 접근(다른 패키지에서도 가능) |



# 정보 은닉(Information Hiding)

## ❖ 점수 클래스 만들기

```
//score.h
#include <iostream>
using namespace std;

class Score {
    int score; //기본 private
public:
    void showScore();
    void setScore(int num);
    int getScore();
};
```

score.cpp

```
#include "score.h"

void Score::showScore() {
    cout << "점수 = " << score << endl;
}

void Score::setScore(const int s) {
    score = s;
}

int Score::getScore() {
    return score;
}
```



# 정보 은닉(Information Hiding)

## ❖ 점수 클래스 만들기

```
#include "score.h"

int main() {
    Score score;

    score.setScore(85);
    //score.showScore();

    cout << "점수 : " << score.getScore() << endl;

    return 0;
}
```

score\_main.cpp





# 정보 은닉(Information Hiding)

## ❖ 은행 계좌 만들기

```
account.h

#include <iostream>
#include <string>
using namespace std;

class Account {
private:
    string ano;
    string owner;
    int balance;

public:
    void setAno(string ano);
    string getAno();
    void setOwner(string owner);
    string getOwner();
    void setBalance(int balance);
    int getBalance();
};
```

```
account.cpp

#include "account.h"

void Account::setAno(string a) {
    ano = a;
}

string Account::getAno() {
    return ano;
}

void Account::setOwner(string own) {
    owner = own;
}

string Account::getOwner() {
    return owner;
}

void Account::setBalance(int bal) {
    balance = bal;
}
```



# 정보 은닉(Information Hiding)

## ❖ 은행 계좌 만들기

```
Account a1 = Account();  
a1.setAno("10-1234");  
a1.setOwner("김실명");  
a1.setBalance(1000);
```

account\_main.h

```
cout << "계좌 번호 : " << a1.getAno() << endl;  
cout << "계좌주 : " << a1.getOwner() << endl;  
cout << "잔고 : " << a1.getBalance() << endl;
```



# 참조 자료형

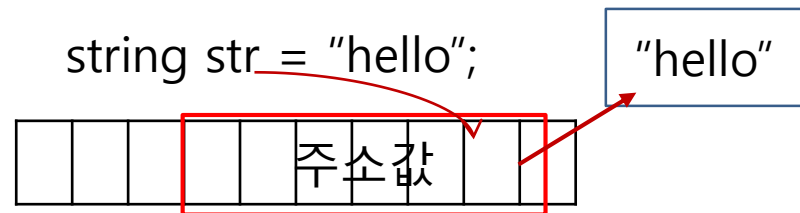
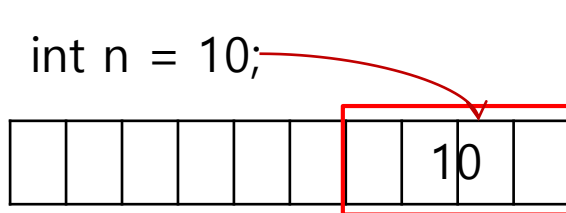
## ▪ 변수의 자료형

### 기본 자료형(Primitive)

C++ 언어에 이미 존재하고 있는 데이터 타입, 주로 간단한 데이터들이다.  
(int, double, bool, char 등)

### 객체 자료형(Object)

여러가지 데이터 타입으로 구성된 자료형(클래스)으로 기본 자료형에 비해 크기가 크다.(string, cout, vector 등)



# 클래스(객체) 참조

## ■ 클래스 간 참조

### Point 클래스

```
struct Point { //점  
    int x;  
    int y;  
}
```

### Circle 클래스

```
class Circle { //원  
    Point center; //중심점  
    int radius; //반지름  
}
```

Point 클래스(자료형)를 참조



# 클래스(객체) 참조

## ■ 점 클래스를 참조하는 원 클래스

점(0, 0) 생성자  
점(1, 2) 생성자  
점(3, 4) 생성자  
원(반지름 = 5) 생성자  
원(반지름 = 5) 생성자  
점(3, 4) 소멸자  
점(1, 2) 소멸자  
점(0, 0) 소멸자

shape.h

```
#pragma once
#include <iostream>
using namespace std;

struct Point { //멤버 변수의 기본 접근 지정자는 public 임
    int x, y;

    Point(int _x = 0, int _y = 0) {
        x = _x;
        y = _y;
        cout << "점(" << x << ", " << y << ") 생성자\n";
    }
    /*Point(int _x = 0, int _y = 0) : x(_x), y(_y) {
        cout << "점(" << x << ", " << y << ") 생성자\n";
    }*/

    ~Point() {
        cout << "점(" << x << ", " << y << ") 소멸자\n";
    }
};
```

# 클래스(객체) 참조

- 점 클래스를 참조하는 원 클래스

```
#include "shape.h"

int main() {

    //Point p;
    Point p = Point();

    //Point p2(1, 2);
    Point p2 = Point(1, 2);

    //Circle c(3, 4, 5);
    Circle c = Circle(3, 4, 5);

    return 0;
}
```

shape\_main.cpp



# this 포인터

## ▪ this 포인터

this는 클래스 멤버 함수에서 객체 자신(그 함수를 실행하는 현재 객체)의 메모리 상의 주소를 나타내는 포인터이다.

아래의 Point 클래스에서 **this->x** 는 멤버 변수이다.

멤버 변수와 생성자의 매개 변수가 같은 경우 사용한다.

```
class Point{
    int x, y;
public:
    Point(int x =0, int y = 0){
        this->x = x;
        this->y = y;
    }
};
```



# this 포인터

## ▪ person 클래스 만들기

```
class Person {  
private:  
    string name;  
    int age;  
  
public:  
    Person();  
    Person(string name, int age);  
    //매개변수와 멤버 변수의 이름이 같음  
  
    void setName(string name);  
    string getName();  
    void setAge(int age);  
    int getAge();  
};
```

```
Person::Person() : name("이름없음"), age(1) {}
```

```
Person::Person(string name, int age) {  
    this->name = name;  
    this->age = age;  
}
```

```
void Person::setName(string name) {  
    this->name = name;  
}
```

```
string Person::getName() {  
    return this->name;  
}
```

```
void Person::setAge(int age) {  
    this->age = age;  
}
```

```
int Person::getAge() {  
    return this->age;  
}
```



# this 포인터

## ▪ person 클래스 만들기

```
Person p;    //기본 생성자로 객체 생성
cout << "이름 : " << p.getName() << endl;
cout << "나이 : " << p.getAge() << endl;

//Person p1;
Person p1 = Person();
p1.setName("이강");
p1.setAge(30);

cout << "이름 : " << p1.getName() << endl;
cout << "나이 : " << p1.getAge() << endl;

Person p2 = Person("안산", 20);
cout << "이름 : " << p2.getName() << endl;
cout << "나이 : " << p2.getAge() << endl;
```



# 동적 메모리 할당과 해제

## ■ 동적 메모리 할당과 해제

- **new** 는 동적으로 메모리를 할당하여 주소를 반환하는 연산자이다.

```
int* ip = new int;
```

```
Person* p = new Person();
```

- 할당된 메모리의 사용

```
*ip = 10;          //ip가 가리키는 곳에 10을 복사
```

```
p->getName() //p가 가리키는 곳에 이름을 반환
```

- **delete**는 동적 할당된 메모리 블록을 시스템에 반납한다.

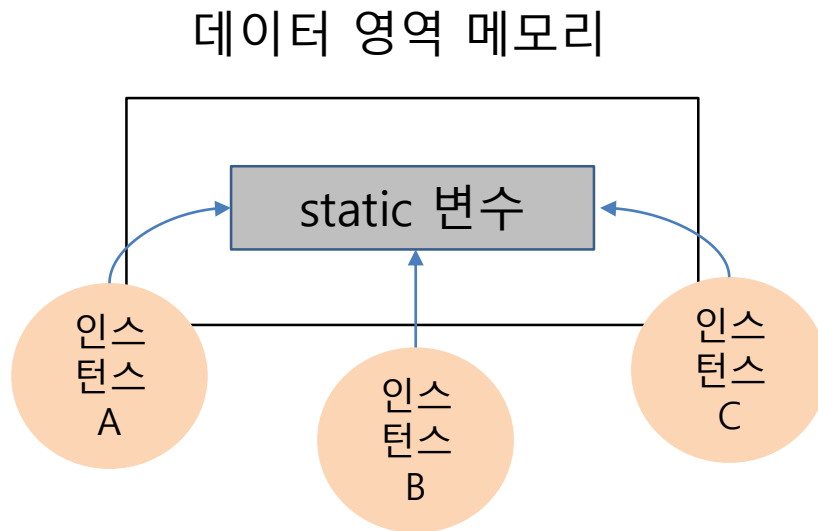
```
Person* p3 = new Person("산내들", 35);  
cout << "이름 : " << p3->getName() << endl;  
cout << "나이 : " << p3->getAge() << endl;  
  
delete p3;
```



# static 변수

## ▪ static 변수의 정의와 사용 방법

- 다른 멤버변수처럼 인스턴스가 생성될 때마다 새로 생성되는 변수가 아니다.
- 여러 개의 인스턴스가 같은 메모리의 값을 공유하기 위해 사용



Static 예약어

```
static int serialNum=1000;
```

# 학번, 카드 자동 부여

## ■ 학번 자동 부여하기

학번 : 101  
학번 : 102  
학번 : 103

```
class Student {  
    static int serialNum;  
    int studentId;  
  
public:  
    Student() {  
        serialNum++;  
        studentId = serialNum;  
    }  
  
    int getStudentId() {  
        return studentId;  
    }  
};
```



# 학번, 카드 자동 부여

## ■ 학번 자동 부여하기

전역공간에 변수 초기화

```
#include "student.h"
```

```
int Student::serialNum = 100;
```

```
int main() {
```

```
    Student jang = Student();
```

```
    cout << "학번 : " << jang.getStudentId() << endl;
```

```
    Student lee = Student();
```

```
    cout << "학번 : " << lee.getStudentId() << endl;
```

```
    Student han = Student();
```

```
    cout << "학번 : " << han.getStudentId() << endl;
```

```
    return 0;
```

```
}
```