



4. 클래스의 완성

- ❖ 정보은닉(자료은폐)과 캡슐화
- ❖ 생성자
- ❖ 소멸자
- ❖ 클래스와 배열
- ❖ 자기참조(this 포인터)
- ❖ friend 선언

Jong Hyuk Park

An abstract graphic on the left side of the slide. It features several translucent blue cubes of various sizes. Some cubes are stacked, while others are floating. A circular icon with a blue arrow pointing right is positioned to the left of the main title. The background has a faint grid pattern.

정보은닉과 캡슐화

Jong Hyuk Park

3장의 내용 정리



❖ 클래스에 대한 기본(7장까지 이어진다)

- 무엇인가를 구현하기에는 아직도 무리!

❖ 클래스의 등장 배경

- 현실 세계를 모델링

❖ 데이터 추상화 → 클래스화 → 객체화

❖ 접근 제어 : public , private, protected

4장의 핵심 내용



❖ 클래스 디자인 기본 원칙

- 캡슐화, 정보 은닉
- 캡슐화와 정보 은닉의 유용성

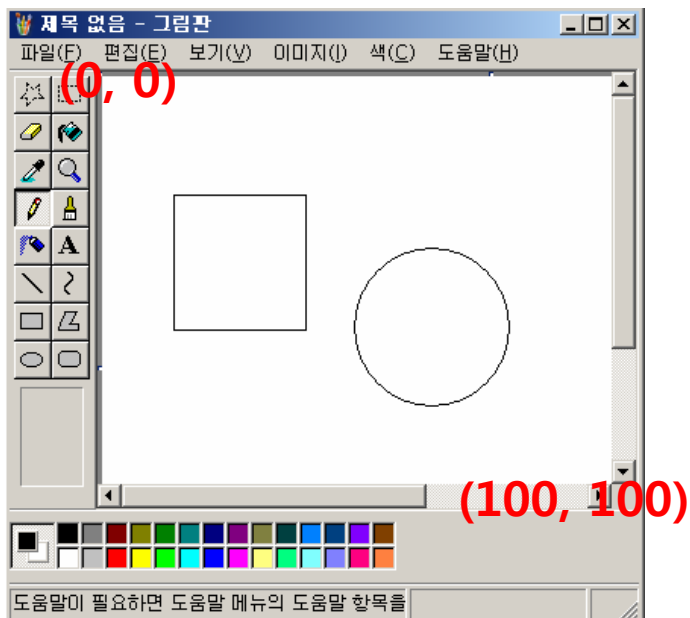
❖ 클래스 객체의 생성과 소멸

- 생성자, 소멸자
- 생성자, 소멸자의 유용성

정보은닉(Information Hiding) ➡

❖ 정보은닉의 필요성

- 프로그램의 안정적인 구현과 관련



```
class Point
{
    Public:
        int x;
        int y;
};
```

* Point → 마우스의 위치정보를 저장하기 위한 클래스

정보은닉(Information Hiding)

```
/*  
  InfoHiding1.cpp  
*/  
#include <iostream>  
using std::cout;  
using std::endl;  
using std::cin;
```

```
class Point
```

```
{  
  public:
```

```
    int x; // x좌표의 범위 : 0 ~ 100  
    int y; // y좌표의 범위 : 0 ~ 100
```

```
};
```

```
int main()
```

```
{
```

```
    int x, y;  
    cout << "좌표 입력 : ";  
    cin >> x >> y;
```

```
    Point p;
```

```
    p.x = x;
```

```
    p.y = y;
```

```
    cout << "입력 된 데이터를 이용해서 그림을 그림" << endl;  
    return 0;
```

```
}
```

실행후, 정상적인 값 입력: 10 20

비 정상값 입력: -20 120

→ 모두 정상적으로 그림을 그림

→ 문제: 객체 외부에서 직접
접근하는것

→ 간접접근으로 수정 ??

InfoHiding2.cpp

C++ Programming

정보은닉 예 - InfoHiding2 / 3.cpp



```
#include<iostream>
using std::cout;
using std::endl;
using std::cin;
```

```
class Point {
// default private
    int x; // x좌표의 범위 : 0 ~ 100
    int y; // y좌표의 범위 : 0 ~ 100
```

```
public:
    int GetX()    { return x; }
    int GetY()    { return y; }

    void SetX(int _x);
    void SetY(int _y);
};
```

- 정보는 은닉되었으나 간접적인 경로가 없음
- → 경로설정

```
void Point::SetX(int _x)
{
    if(_x<0 || _x>100)
    {
        cout<<"X좌표 입력 오류, 확인 요망"<<endl;
        return;
    }
    x=_x;
}
```

```
void Point::SetY(int _y)
{
    if(_y<0 || _y>100)
    {
        cout<<"Y좌표 입력 오류, 확인 요망"<<endl;
        return;
    }
    y=_y;
}
```

```
int main()
{
    int x, y;
    cout<<"좌표입력 : ";
    cin>>x>>y;

    Point p;
    p.SetX(x);
    p.SetY(y);

    cout<<"입력 된 데이터를 이용해서 그림을 그림"<<endl;
    return 0;
}
```

→ 실행

-10 -100

여전히 정상적인 값이 출력

정보은닉이 되었으나 여전히 문제

→ 데이터의 경계검사와
그에따른 적절한 처리 부분

캡슐화(Encapsulation)



❖ 캡슐화(Encapsulation)의 기본 개념

- 관련 있는 데이터와 함수를 하나로 묶는 것
- 예) Encapsulation1.cpp
Encapsulation2.cpp

❖ 캡슐화의 유용성

- 생각해봅시다. P 132, Y2K
 - 전역변수의 사용시, 변경 → 관련된 모든 함수의 수정이 이루어져야 함
- 하나의 클래스내의 변수를 조작하기 위한 함수는 다른 클래스에 두지 않는 것이 바람직!!

캡슐화 예 (1)



Encapsulation1.cpp

```
#include<iostream>
using std::cout;
using std::endl;
using std::cin;

class Point {
    int x; // x좌표의 범위 : 0 ~ 100
    int y; // y좌표의 범위 : 0 ~ 100
public:
    int GetX()    { return x; }
    int GetY()    { return y; }

    void SetX(int _x);
    void SetY(int _y);
};

void Point::SetX(int _x)
{
    if(_x<0 || _x>100) {
        cout<<"X좌표 입력 오류, 확인 요망"<<endl;
        return;
    }
    x=_x;
}

void Point::SetY(int _y)
{
    if(_y<0 || _y>100)
    {
        cout<<"Y좌표 입력 오류, 확인 요망"<<endl;
        return;
    }
    y=_y;
}
```

```
class PointShow {
public:
    void ShowData(Point p)
    {
        cout<<"x좌표: "<<p.GetX()<<endl;
        cout<<"y좌표: "<<p.GetY()<<endl;
    }
};

int main()
{
    int x, y;
    cout<<"좌표입력 : ";
    cin>>x>>y;

    Point p;
    p.SetX(x);
    p.SetY(y);

    PointShow show;
    show.ShowData(p);

    return 0;
}
```

캡슐화 예 (2)



Encapsulation2.cpp

```
#include <iostream>
using std::cout;
using std::endl;
using std::cin;

class Point {
    int x; // x좌표의 범위 : 0~100
    int y; // y좌표의 범위 : 0~100
public:
    int GetX(){ return x; }
    int GetY(){ return y; }

    void SetX(int _x);
    void SetY(int _y);

    void ShowData(); //캡슐화를 위해 추가된 함수.
};

void Point::SetX(int _x)
{
    if(_x<0 || _x>100) {
        cout<<"X좌표 입력 오류, 확인 요망"<<endl;
        return;
    }
    x=_x;
}

void Point::SetY(int _y)
{
    if(_y<0 || _y>100)
    {
        cout<<"Y좌표 입력 오류, 확인 요망"<<endl;
        return;
    }
    y=_y;
}
```

```
void Point::ShowData()
{
    cout<<"x좌표: "<<x<<endl;
    cout<<"y좌표: "<<y<<endl;
}

int main()
{
    int x, y;
    cout<<"좌표입력 : ";
    cin>>x>>y;

    Point p;
    p.SetX(x);
    p.SetY(y);
    p.ShowData();

    return 0;
}
```

정보은닉과 캡슐화



❖ 객체지향 프로그래밍의 객체 정보은닉, 캡슐화 지원

- 오류 검출 등 유지보수 용이

❖ 정보은닉(자료은폐, information hiding)

- 객체 내부의 자료를 객체 외부로부터 은폐하여 외부에서 접근금지
- 객체 내부 자료를 private으로 선언하여 실현
- 필요성 : 프로그램의 안정적 구현

❖ 캡슐화(Encapsulation)

- 관련된 자료와 함수를 하나의 단위로 그룹화



생성자

Jong Hyuk Park

생성자 (constructor)



❖ 생성자의 필요성

- 객체를 생성과 동시에 초기화하기 위해
- 객체는 생성과 동시에 초기화되는 것이 좋은 구조

❖ 생성자란?

- 객체 생성 시 반드시 한번 호출되는 함수
- 클래스와 같은 이름의 함수
- 리턴형이 없으며 리턴하지도 않음

생성자 (constructor)



```
class Square {  
    int side;  
public:  
    Square(); { side = 0; }    // 생성자  
    void set(int);  
    int area();  
};  
Square a;    // 객체 a 선언, 생성자 호출
```

```
class Square {  
    int side;  
public:  
    Square(int n); { side = n; } // 생성자  
    void set(int);  
    int area();  
};  
Square a(12); // 객체 a 선언, 생성자 호출
```

생성자와 소멸자



/* Person1.cpp */

class Person

{

char name[SIZE];

char phone[SIZE];

int age;

public:

void ShowData();

};

void Person::ShowData()

{

cout<<"name: "<<name<<endl;

cout<<"phone: "<<phone<<endl;

cout<<"age: "<<age<<endl;

}

int main()

{

Person p={"KIM", "013-113-1113", 22};

p.ShowData();

return 0;

}

Private 접근

오류의
원인은?

- C의 구조체 변수에서 사용
- C++ 본 예제에서는
직접접근 → 오류

C++ Programming

생성자와 소멸자



```
/*
   Person2.cpp
*/
class Person
{
    char name[SIZE];
    char phone[SIZE];
    int age;

public:
    void ShowData(){ ... 생략 ... }
    void SetData(char* _name, char* _phone, int _age);
};

void Person::SetData(char* _name, char* _phone, int _age)
{
    strcpy(name, _name);
    strcpy(phone, _phone);
    age=_age;
}

int main()
{
    Person p;
    p.SetData("KIM", "013-333-5555", 22);
    return 0;
}
```

name
phone
age
showDate()
setData()

메모리공간

생성과 동시에
초기화?

→ X

C++ Programming

생성자와 소멸자



❖ 객체의 생성 과정(Constructor1.cpp)

- 첫째. 메모리 할당
- 둘째. 생성자의 호출

```
class AAA
{
    int i, j;
public:
    AAA() //생성자
    {
        cout<<"생성자 호출"<<endl;
        i=10, j=20;
    }
    void ShowData()
    {
        cout<<i<<' '<<j<<endl;
    }
};
```

원하는 값으로
초기화할 수 있는가?

→ X

```
main()
{
    AAA a;
}
```

a

i=?	10
j=?	20
AAA()	
showDate()	

메모리공간

C++ Programming

생성자와 소멸자



❖ 생성자를 통한 인자의 전달

- 이전의 객체 생성 방법은 구조체 변수의 선언과 동일한 방법!
- Constructor2.cpp, Person3.cpp

```
class AAA
{
    int i, j;
public:
    AAA(int _i, int _j) //생성자
    {
        i=_i, j=_j;
    }
    void ShowData()
    {
        cout<<i<<' '<j<<endl;
    }
};
```

•객체생성 문법

AAA aaa (111, 222);

aaa

i=111
j=222
AAA()
showDate()

111, 222를 인자로 받을 수 있는 생성자 호출

객체의 이름은 aaa

AAA 클래스의 객체 생성

C++ Programming

생성자와 소멸자



❖ public 생성자, private 생성자

- public 생성자 : 어디서든 객체 생성 가능
- private 생성자 : 클래스 내부에서만 가능

❖ 객체 생성의 두 가지 형태

```
Person p = Person("KIM", "013-333-5555", 22);  
Person p("KIM", "013-333-5555", 22);
```

❖ 오해하기 쉬운 객체 생성

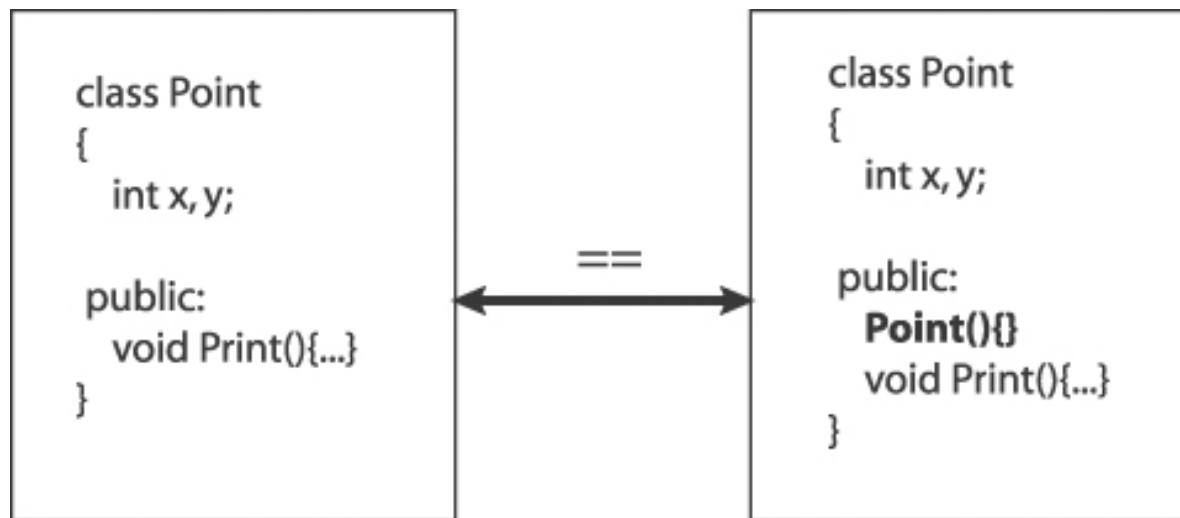
```
Person p("KIM", "013-333-5555", 22);    // 객체 생성  
Person p();                               // 객체 생성?
```

생성자와 소멸자



❖ 디폴트(default) 생성자

- 생성자가 하나도 정의되어 있지 않은 경우
- 자동으로 삽입이 되는 생성자
- 디폴트 생성자가 하는 일? 아무것도 없다.



C++ Programming

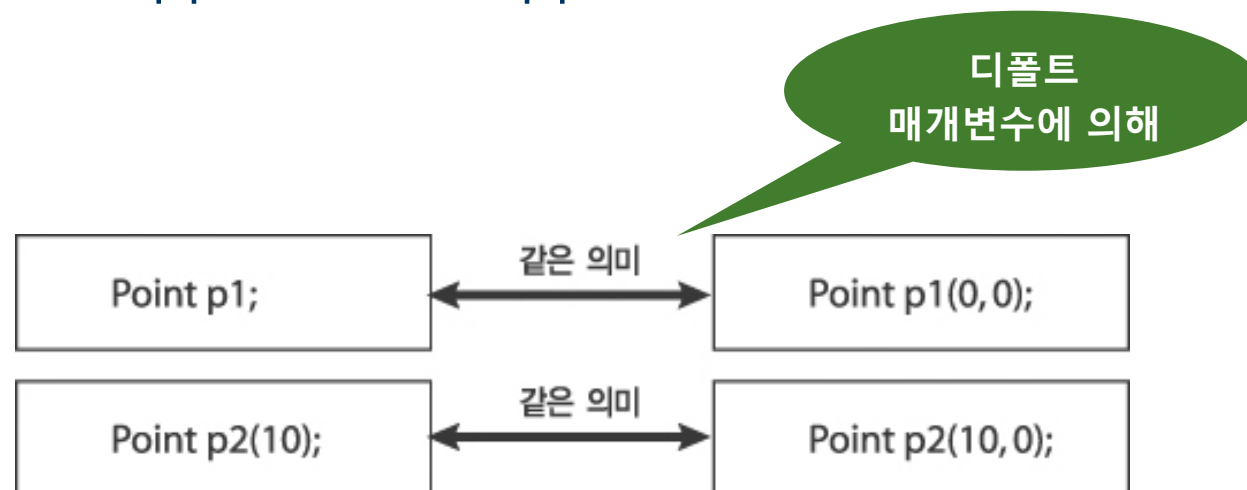
- Default1.cpp

생성자와 소멸자



❖ 생성자의 특징

- 생성자도 함수다!
- 따라서 함수 오버로딩이 가능하다
- 디폴트 매개 변수의 설정도 가능하다.
- Default2.cpp, Default3.cpp



생성자와 소멸자



❖ 객체가 소멸되는 시점은?

- 기본 자료형 변수, 구조체 변수가 소멸되는 시점과 동일하다.

❖ 함수 내에 선언된 객체

- 함수 호출이 끝나면 소멸된다.

❖ 전역적으로 선언된 객체

- 프로그램이 종료될 때 소멸된다.
- 이렇게 객체 생성할 일 (거의) 없다!

생성자와 소멸자



❖ 생성자와 동적 할당(Person4.cpp)

```
class Person
{
    char *name;
    char *phone;
    int age;
public:
    Person(char* _name, char* _phone, int _age);
    void ShowData(){ ... 생략 ... }
};

Person::Person(char* _name, char* _phone, int _age)
{
    name=new char[strlen(_name)+1];
    strcpy(name, _name);

    phone=new char[strlen(_phone)+1];
    strcpy(phone, _phone);
    age=_age;
}
```

동적 할당에 따른
메모리의 해제는?

생성자와 소멸자



❖ 메모리를 해제하는 멤버 함수의 제공

■ Person5.cpp

```
class Person
{
    char *name;
    char *phone;
    int age;
public:
    Person(char* _name, char* _phone, int _age);
    void DelMemory();
};

Person::Person(char* _name, char* _phone, int _age)
{
    ... 생략 ...
}

void Person::DelMemory()
{
    delete []name;
    delete []phone;
}
```

만족하는가?

•NO

객체의 소멸 직전에
동적할당을 명시적으로
해주어야함 → 부담.

생성자 예 (1)



```
#include <iostream>
using std::cout;
using std::endl;
class Square {
    int side;
public:
    Square(int n);    { side = n; } // 생성자
    void set(int);
    int area();
};
void Square::set(int x)
{
    side = x;
}
int Square::area()
{
    return side*side;
}
int main()
{
    Square a(12);
    cout << "area : " << a.area() << endl;
    return 0;
}
```

생성자 예 (2)



```
#include <iostream>
using std::cout;
using std::endl;
```

```
class Point {
    int x, y;
public:
    Point(int _x, int _y)
    {
        x=_x, y=_y;
    }
    void ShowData()
    {
        cout<<x<<' '<<y<<endl;
    }
};
```

```
int main()
{
    Point p1(10, 20); //10과 20을 인자로 받는 생성자 호출
    p1.ShowData();

    Point p2;    => 에러 !
    p2.ShowData();
    return 0;
}
```

생성자 예 (3)

- 생성자 오버로딩



```
#include<iostream>
using std::cout;
using std::endl;
```

```
class Point {
    int x, y;
public:
```

```
    Point()
```

```
    {
```

```
        x=y=0;
```

```
    }
```

```
    Point(int _x, int _y)
```

```
    {
```

```
        x=_x, y=_y;
```

```
    }
```

```
    void ShowData()
```

```
    {
```

```
        cout<<x<<' '<<y<<endl;
```

```
    }
```

```
};
```

```
int main()
```

```
{
```

```
    Point p1(10, 20); //10과 20을 인자로 받는 생성자 호출
    p1.ShowData();
```

```
    Point p2; //void 생성자 호출.
    p2.ShowData();
    return 0;
```

```
}
```

C++ Programming

생성자 예 (4)

- 생성자 디폴트 매개변수



```
#include<iostream>
using std::cout;
using std::endl;
```

```
class Point {
    int x, y;
public:
    Point(int _x=0, int _y=0)
    {
        x=_x, y=_y;
    }
    void ShowData()
    {
        cout<<x<<' '<<y<<endl;
    }
};
```

```
int main()
{
    Point p1(10, 20);
    p1.ShowData();

    Point p2;      // Point(0, 0)과 같다.
    p2.ShowData();

    Point p3(10);  // Point(10, 0)과 같다.
    p3.ShowData();

    return 0;
}
```

생성자 예 (4) - 동적할당



```
#include<iostream>
using std::cout;
using std::endl;
class Person {
    char *name;
    char *phone;
    int age;
public:
    Person(char* _name, char* _phone, int _age);
    void ShowData();
    void DelMemory();
};
Person::Person(char* _name, char* _phone, int _age)
{
    name=new char[strlen(_name)+1];
    strcpy(name, _name);
    phone=new char[strlen(_phone)+1];
    strcpy(phone, _phone);
    age=_age;
}
void Person::ShowData()
{
    cout<<"name: "<<name<<endl;
    cout<<"phone: "<<phone<<endl;
    cout<<"age: "<<age<<endl;
}
```

void Person::DelMemory()

```
{
    delete []name;
    delete []phone;
}

int main()
{
    Person p("KIM", "013-333-5555", 22);
    p.ShowData();

    p.DelMemory();

    return 0;
}
```

C++ Programming

A decorative graphic on the left side of the slide. It features several translucent blue cubes of various sizes, some stacked and some floating. A circular icon with a blue arrow pointing right is positioned to the left of the text.

소멸자

Jong Hyuk Park

소멸자의 특성과 필요성



- ❖ 소멸자는 생성자와 반대로 객체가 선언되어 사용되는 블록이 끝날 때(객체가 소멸될 때) 자동적으로 호출되는 함수
- ❖ 소멸자는 주로 객체가 생성될 때 동적으로 할당한 메모리를 객체의 소멸과 더불어 해제하고자 할 때 사용
- ❖ 소멸자는 클래스 이름 앞에 ~(tilde) 기호를 붙여서 정의
- ❖ 리턴하지 않으며, 리턴타입 없음
- ❖ 전달인자 항상 void
 - 오버로딩, 디폴트매개변수의 선언 불가능.
- ❖ 객체의 소멸 순서
 - 첫째: 소멸자 호출
 - 둘째: 메모리 반환(해제)
- ❖ Destructor.cpp Person6.cpp

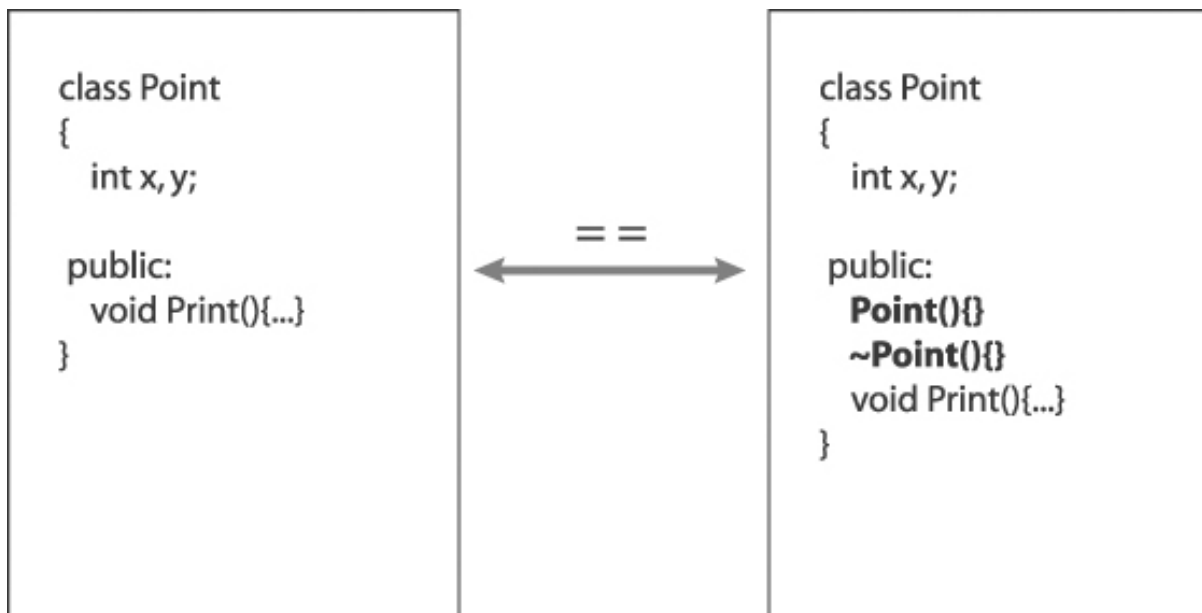
C++ Programming

생성자와 소멸자



❖ 디폴트(Default) 소멸자

- 객체의 소멸 순서를 만족시키기 위한 소멸자
- 명시적으로 소멸자 제공되지 않을 경우 자동 삽입
- 디폴트 생성자와 마찬가지로 하는 일 없다!



생성자와 소멸자



❖ 소멸자의 명시적 제공

- 생성자에서 메모리 동적 할당을 하는 경우
- Debugging 코드의 작성
 - Cout 등 객체소멸 시점 확인가능

소멸자 예 (1)

```
#include <iostream>
using std::cout;
using std::endl;

class Point {
    int xval, yval;
public:
    Point(int x, int y)    { xval = x; yval = y; }
    ~Point()              { cout << "Destructing..\n"; } // 소멸자
    showxy()              { cout << xval << " , " << yval << endl; }
};

int main()
{
    Point pt1(9,4);
    {
        Point pt2(11,88);           // 내부 블록 시작

        cout << "pt2 : ";
        pt2.showxy();
    }                               // 내부 블록 끝
    cout << "pt1 : ";
    pt1.showxy();

    return 0;
}
```

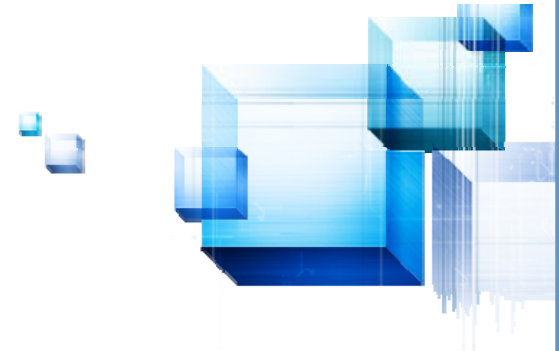


pt2 : 11 , 88
Destructing..
pt1 : 9 , 4
Destructing..

블록 내에서 객체 pt2는 블록이
끝날 때 소멸자를 호출하고, 객체
pt1은 주 프로그램의 시작에서 선
언되었으므로 주 프로그램이 끝
날 때 소멸자를 호출한다.

C++ Programming

소멸자 예 (2)



```
#include <iostream>
using std::cout;
using std::endl;

class Person {
    char *name;
    char *phone;
    int age;
public:
    Person(char* _name, char* _phone, int _age);
    ~Person();
    void ShowData();
};

Person::Person(char* _name, char* _phone, int _age)
{
    name=new char[strlen(_name)+1];
    strcpy(name, _name);

    phone=new char[strlen(_phone)+1];
    strcpy(phone, _phone);

    age=_age;
}
```

```
Person::~Person()
{
    delete []name;
    delete []phone;
}

void Person::ShowData()
{
    cout<<"name: "<<name<<endl;
    cout<<"phone: "<<phone<<endl;
    cout<<"age: "<<age<<endl;
}

int main()
{
    Person p("KIM", "013-333-5555", 22);
    p.ShowData();

    return 0;
}
```



클래스와 배열

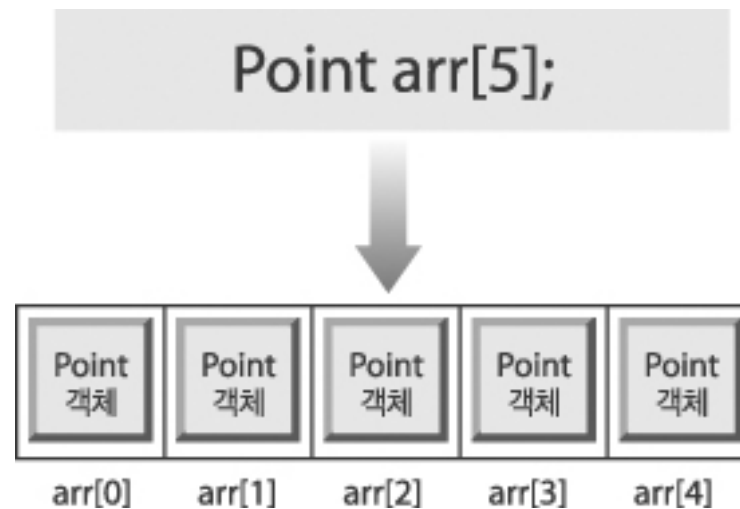
Jong Hyuk Park

클래스와 배열



❖ 객체 배열과 생성자

- 객체 배열-객체를 멤버로 지니는 배열
- 객체 배열은 기본적으로 void 생성자의 호출을 요구한다.
- PointArr1.cpp



객체 배열 예 (1)



```
#include<iostream>
using std::cout;
using std::endl;

class Point {
    int x;
    int y;
public:
    Point(){
        cout<<"Point() call!"<<endl;
        x=y=0;
    }
    Point(int _x, int _y){
        x=_x;
        y=_y;
    }

    int GetX()    { return x; }
    int GetY()    { return y; }

    void SetX(int _x){ x=_x; }
    void SetY(int _y){ y=_y; }
};
```

```
int main()
{
    Point arr[5];

    for(int i=0; i<5; i++)
    {
        arr[i].SetX(i*2);
        arr[i].SetY(i*3);
    }

    for(int j=0; j<5; j++)
    {
        cout<<"x: "<<arr[j].GetX()<<' ';
        cout<<"y: "<<arr[j].GetY()<<endl;
    }

    return 0;
}
```

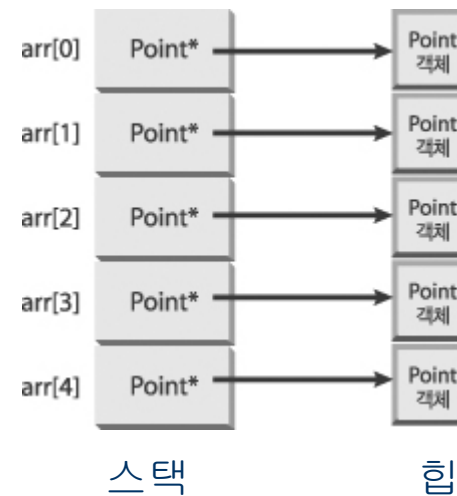
클래스와 배열



❖ 객체 포인터 배열

- 객체를 가리킬 수 있는 포인터를 멤버로 지니는 배열
- 객체의 동적 생성 방법(PointArr2.cpp)

```
Point* arr[5];
```



객체 배열 예 (2)



```
#include <iostream>
using std::cout;
using std::endl;
class Point {
    int x;
    int y;
public:
    Point(){
        cout << "Point() call!" << endl;
        x=y=0;
    }
    Point(int _x, int _y){
        x=_x;
        y=_y;
    }
    int GetX() { return x; }
    int GetY() { return y; }
    void SetX(int _x){ x=_x; }
    void SetY(int _y){ y=_y; }
};
int main()
{
    Point* arr[5];

    for(int i=0; i<5; i++)
    {
        arr[i]=new Point(i*2, i*3);
        // new에 의한 객체 동적 생성.
    }
}
```

```
for(int j=0; j<5; j++)
{
    cout << "x: " << arr[j]->GetX() << ' ';
    cout << "y: " << arr[j]->GetY() << endl;
}

for(int k=0; k<5; k++)
{
    delete arr[k]; // 힙에 저장된 객체 소멸.
}

return 0;
}
```




자기참조(this 포인터)

Jong Hyuk Park

자기참조(self-reference)



❖ 자기참조

- 클래스의 멤버함수는 자신을 호출한 객체를 가리키는 포인터를 명시
- `this`라는 키워드가 자기 자신 객체의 포인터를 가리킴

자기참조 예 (1)



```
#include <iostream>
using std::cout;
using std::endl;

class Person {
public:
    Person* GetThis(){
        return this; //this 포인터를 리턴.
    }
};

int main()
{
    cout<<"***** p1의 정보 *****"<<endl;
    Person *p1 = new Person();
    cout<<"포인터 p1: "<<p1<<endl;
    cout<<"p1의 this: "<<p1->GetThis()<<endl;

    cout<<"***** p2의 정보 *****"<<endl;
    Person *p2 = new Person();
    cout<<"포인터 p2: "<<p2<<endl;
    cout<<"p2의 this: "<<p2->GetThis()<<endl;

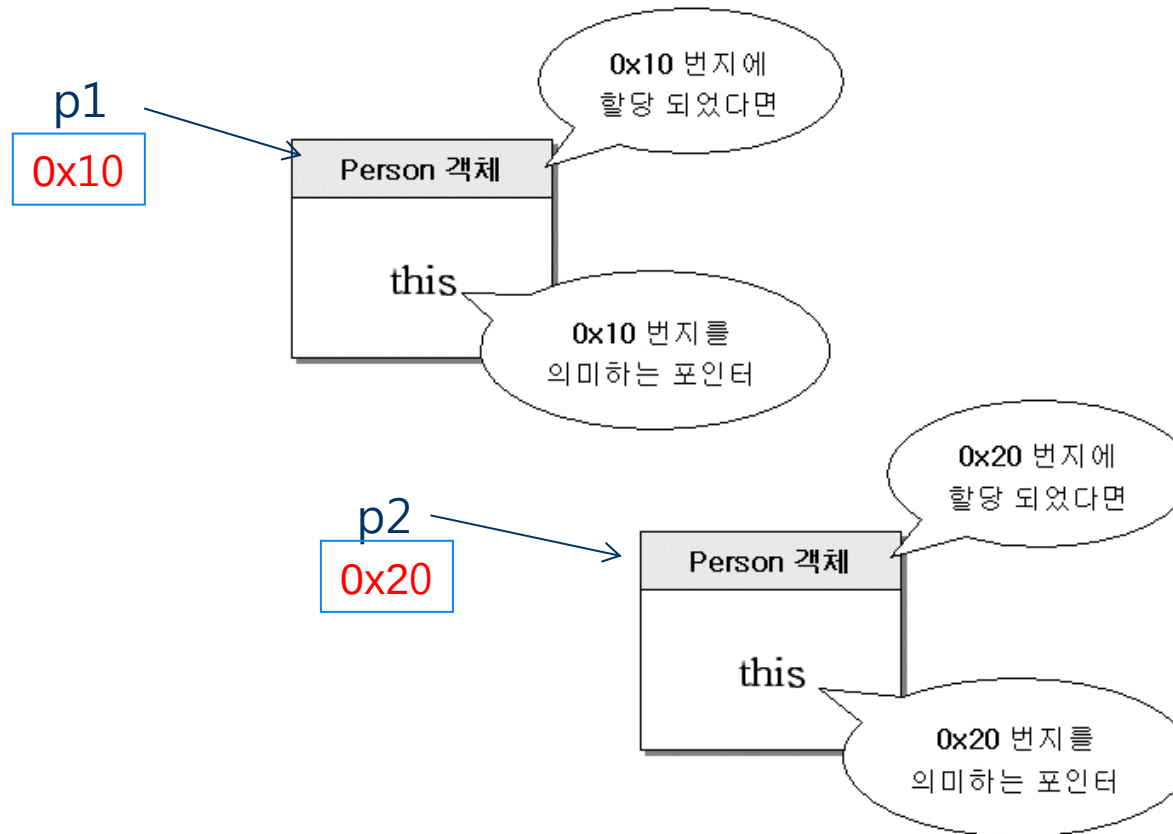
    return 0;
}
```

자기참조(self-reference)



❖ this 포인터의 의미

- this_under.cpp가 말해준다!



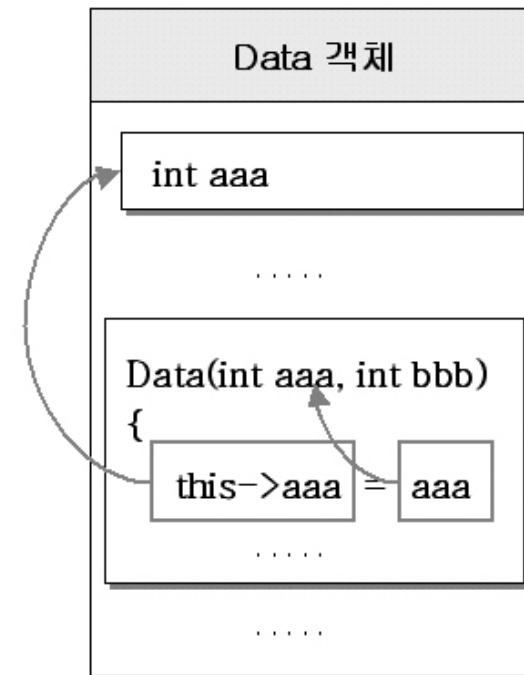
자기참조 예 (2)

```
#include <iostream>
using std::cout;
using std::endl;

class Data {
    int aaa;
    int bbb;
public :
    Data(int aaa, int bbb) {
        //aaa=aaa;
        this->aaa=aaa;

        //bbb=bbb;
        this->bbb=bbb;
    }
    void printAll() {
        cout<<aaa<<" "<<bbb<<endl;
    }
};

int main(void)
{
    Data d(100, 200);
    d.printAll();
    return 0;
}
```



자기참조 예 (3)



```
#include <iostream>
using std::cout;
using std::endl;

class Dog {
    Dog *body;
    char *name;
    Dog *tail;
public:
    Dog(char *s);
    char *dog_name();
    void make_tail(char *s);
    char *tail_name();
};

Dog::Dog(char *s)
{
    name = new char[strlen(s) + 1];
    strcpy(name, s);
    tail = body = 0;
}

char *Dog::dog_name()
{ return name; }
```

```
void Dog::make_tail(char *s)
{
    tail = new Dog(s); // tail 객체 생성
    tail->body = this; // 자기참조 포인터 지정
}

char *Dog::tail_name()
{
    return tail->dog_name();
}

int main()
{
    Dog dog("Happy");

    dog.make_tail("Merry");
    cout << "dog name : "
         << dog.dog_name() << endl;
    cout << "tail name : " << dog.tail_name()
         << endl;

    return 0;
}
```

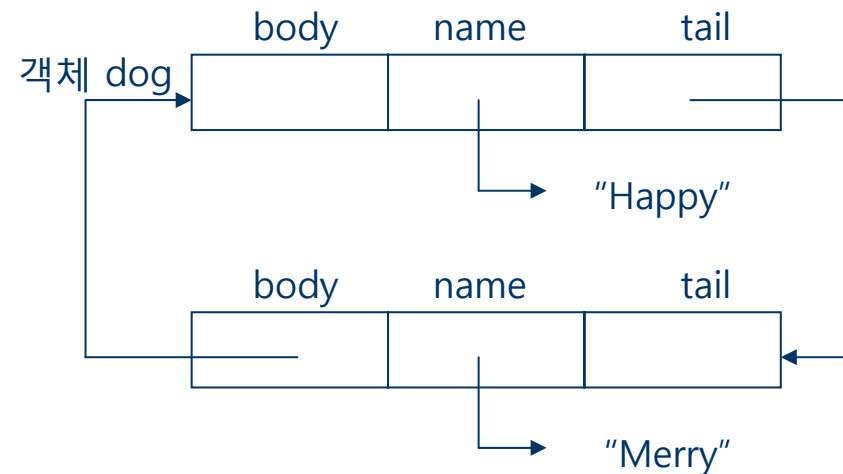
자기참조 예 (3)



dog name : Happy

tail name : Merry

전용부분에서 같은 클래스의 포인터로서 body, tail을 선언하였다. make_tail() 함수에서 새로 생성한 객체 tail의 body가 호출한 객체를 가리키도록 자기참조 포인터 this를 지정하였다.



A decorative graphic on the left side of the slide. It features several translucent blue cubes of various sizes, some overlapping. A circular button with a blue right-pointing arrow is positioned to the left of the text.

friend 선언

Jong Hyuk Park

프렌드 함수 (friend function) (1)



- ❖ C++ 언어의 클래스의 전용부분은 같은 객체의 멤버함수만이 접근 가능하다는 점에서 객체지향 프로그래밍에서의 자료의 은폐와 캡슐화를 지원
 - 전용부분 자료의 접근을 클래스의 내부에 국한시킴으로 클래스 내부의 수정이 클래스 외부 영역에 전혀 영향을 주지 못하기 때문에 프로그램의 유지보수가 용이
- ❖ 불가피하게 클래스의 멤버함수가 아닌 일반 함수 또는 다른 클래스의 함수가 클래스의 전용부분에 있는 자료를 사용해야만 하는 경우 **프렌드 함수** 사용
 - 프렌드 함수는 클래스의 멤버함수가 아닌 다른 멤버의 함수가 클래스의 전용부분의 자료에 접근하는 수단으로 이용

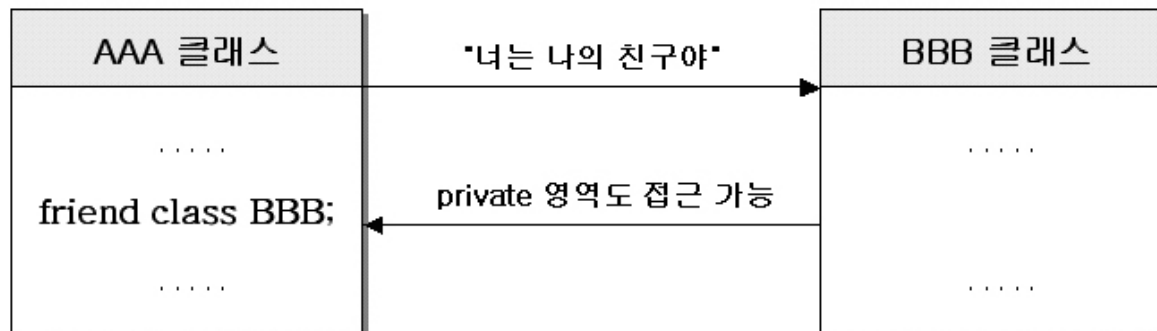
프렌드 함수 (2)



❖ 접근하는 클래스에 **friend 키워드**로 프렌드 함수 선언

- 프렌드 함수의 선언은 클래스의 전용부분이나 공용부분 어느 곳에 위치하든지 차이가 없음
- 전역함수에게 private 영역 접근 허용

```
class Ex {  
    .....  
    friend int func();  
    // 클래스 Ex의 프렌드 함수 func() 선언  
    .....  
};
```



friend 함수 예

friend1.cpp

```
#include <iostream>
using std::cout;
using std::endl;

class Counter {
private:
    int val;

public:
    Counter() {
        val=0;
    }
    void Print() const {
        cout<<val<<endl;
    }

    friend void SetX(Counter& c, int val);
    //friend 선언.
};

void SetX(Counter& c, int val) // 전역함수.
{
    c.val=val;
}
```



```
int main()
{
    Counter cnt;
    cnt.Print();

    SetX(cnt, 2002);
    cnt.Print();

    return 0;
}
```

C++ Programming

friend 클래스 예

- 클래스가 클래스를 friend로 선언

friend2.cpp

```
#include <iostream>
using std::cout;
using std::endl;

class AAA {
private:
    int data;
    friend class BBB;
    // class BBB를 friend로 선언함!
};

class BBB {
public:
    void SetData(AAA& aaa, int val){
        aaa.data=val;
        //class AAA의 private 영역 접근!
    }
};
```

```
int main()
{
    AAA aaa;
    BBB bbb;

    bbb.SetData(aaa, 10);

    return 0;
}
```

friend 선언



❖ friend 선언의 유용성

- 유용하지 않다!
- 정보 은닉에 위배되는 개념
- 연산자 오버로딩에서 유용하게 사용
- 그 전에는 사용하지 말자!

❖ friend 선언으로만 해결 가능한 문제

- 그런 것은 존재하지 않는다.
- 연산자 오버로딩에서는 예외!

과제 및 실습



HW #1. 연습문제 4-2: Time 클래스 프로그램

**** 모든 연습문제 풀어보기 ****

실습. 4-8절의 은행계좌 관리 프로그램에 다음을 추가하여라

- ❖ 고객의 전화번호 정보를 추가하여라. 전화번호는 다음과 같이 private으로 선언하라.
char *phone;
- ❖ (이름,전화번호)와 계좌번호로 잔액조회를 하는 Inquire() 오버로딩 함수를 작성하여라.
 - ❖ void Inquire(char *name, char *phone);
 - ❖ void Inquire(int id);



Q & A

Jong Hyuk Park