

## 8장 상속

- # 상속의 기본 개념
- # 상속 관련 문제 제기
- # base 클래스의 접근 제어와 protected 멤버
- # 상속 관계에서의 생성자와 소멸자
- # 함수 재정의(function overriding)
- # 디폴트 액세스 지정자와 구조체
- # derived 클래스로부터의 상속
- # 다중 상속
- # virtual base 클래스
- # derived 클래스의 디폴트 복사 생성자와 디폴트 대입 연산자
- # private 생성자의 사용

# 1. 상속의 기본 개념

## # 다음과 같은 문제를 위한 클래스 설계

### ■ 자동차

- 속성 : 색상, 배기량, 현재속도
- 메서드 : 가속하라, 멈춰라, 시동을켜라

### ■ 트럭

- 속성 : 색상, 배기량, 현재속도, 최대중량
- 메서드 : 가속하라, 멈춰라, 시동을켜라

### ■ 택시

- 속성 : 색상, 배기량, 현재속도, 요금, 주행거리
- 메서드 : 가속하라, 멈춰라, 시동을 켜라

## # 방법 1 : 자동차 클래스(CCar)로 트럭과 택시를 모두 표현

```
class CCar {  
속성 :  
    색상, 배기량, 현재속도, 최대중량, 요금, 주행거리;  
메서드 :  
    가속하라, 멈춰라, 시동을켜라;  
};
```

문제점 : 실제 객체가 트럭이라면  
요금과 주행거리는 불필요.  
실제 객체가 택시라면 최대중량 불필요

# 1. 상속의 기본 개념

## # 방법 2 : 자동차(CCar), 트럭(CTruck), 택시(CTaxi) 클래스 만들기

```
class CCar {
속성:
    색상, 배기량, 현재속도;
메서드:
    가속하라, 멈춰라, 시동을켜라;
};

class CTruck {
속성:
    색상, 배기량, 현재속도, 최대중량;
메서드:
    가속하라, 멈춰라, 시동을켜라;
};

class CTaxi {
속성:
    색상, 배기량, 현재속도, 요금, 주행거리;
메서드:
    가속하라, 멈춰라, 시동을켜라;
};
```

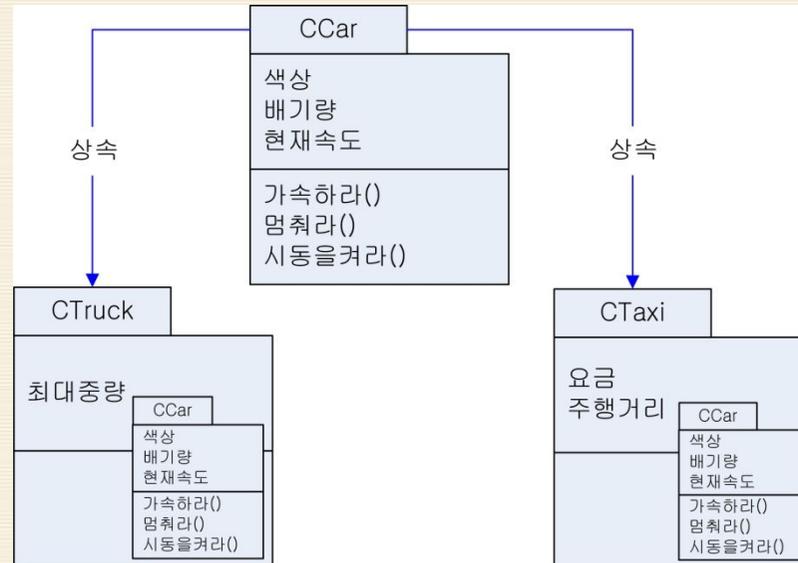
문제점 : 코드의 중복성  
CTruck의 대부분이 CCar와 동일  
CTaxi 역시 대부분이 CCar와 동일  
→ 작업의 비효율성

코드의 중복, 비효율성을 제거하는  
방법 → 상속

# 1. 상속의 기본 개념

## # 상속의 개념

- CTruck과 CTaxi 클래스 작성 시 CCar로부터 상속을 받으면 중복된 내용은 다시 언급할 필요가 없음



- 상속 기본 문법

```
class CTruck : public CCar {  
속성:  
    최대중량;  
메서드:  
};
```

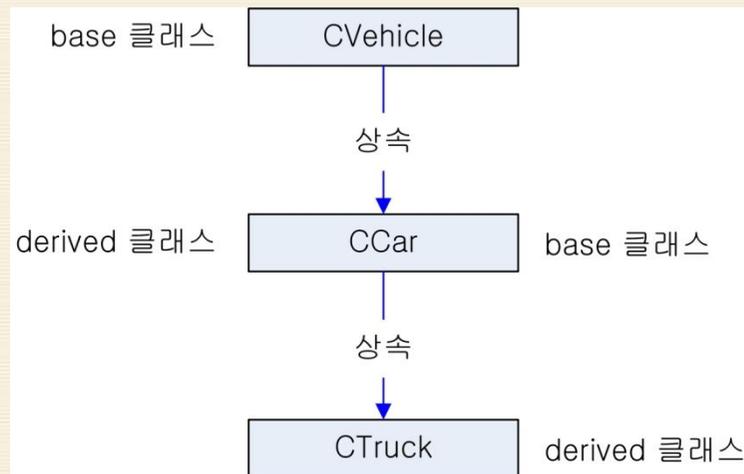
# 1. 상속의 기본 개념

## # base 클래스와 derived 클래스

- base 클래스 : CCar 클래스
- derived 클래스 : CTruck 클래스, CTaxi 클래스

## # base와 derived 클래스의 관계는 상대적인 개념

- CCar 클래스를 CVehicle 클래스로부터 상속받아 만든다면



## # 상속 관계가 자연스럽게 성립하는 경우

- is-a 관계 : 트럭은 자동차이다. 택시는 자동차이다. 사과는 과일이다.
- 그 외에 기존 클래스의 모든 특성을 상속받고자 할 때

## # 상속 : 코드 재활용을 위한 강력한 수단

## 2. 상속 관련 문제 제기

# 예 : 원을 나타내는 CCircle 클래스 작성

```
#define PI 3.14

class CCircle {
public :
    int x, y;      // 중심
    double Radius; // 반지름

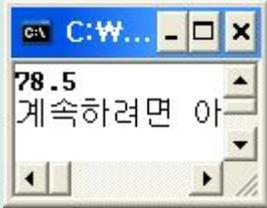
public :
    double GetArea() { return (PI * Radius * Radius); } // 면적
};

int main(void)
{
    CCircle Cir;
    Cir.x = 1; Cir.y = 1; Cir.Radius = 5;

    cout << Cir.GetArea() << endl;

    return 0;
}
```

편의상 모든 멤버는 **public**에 선언



## 2. 상속 관련 문제 제기

# 예 : 구를 나타내는 CSphere 클래스 작성

- 중심 (x, y, z), 반지름(Radius), 표면적 함수, 부피 함수 → CCircle 상속!

```
#define PI 3.14

class CCircle {
public :
    int x, y;           // 중심
    double Radius;     // 반지름
public :
    double GetArea() { return (PI * Radius * Radius); }           // 면적
};

class CSphere : public CCircle { // CCircle로부터 상속
public :
    int z;
public :
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};

int main(void)
{
    CSphere Sph;
    Sph.x = 1; Sph.y = 1; Sph.z = 1; Sph.Radius = 5; // Sph:x, y, Radius 상속

    cout << "구의 표면적 : " << Sph.GetArea() << endl; // Sph:GetArea 상속
    cout << "구의 부피   : " << Sph.GetVolume() << endl;
    return 0;
}
```

이 프로그램의 문제점은?



## 2. 상속 관련 문제 제기

### # 상속 관련 문제 제기

#### 1. 액세스 지정자

- CCircle의 멤버 변수들을 private 영역에 포함시킨다면 → 에러 발생
- 액세스 지정자의 종류 : public, private, protected

#### 2. 생성자와 소멸자

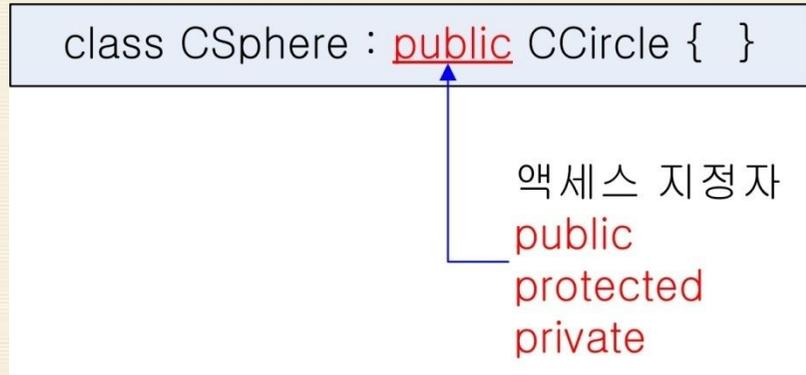
- base 클래스와 derived 클래스에 생성자와 소멸자가 필요하다면...
- 작성 방법, 호출 방법, 호출 순서

#### 3. CCircle과 CSphere 클래스의 GetArea 함수

- 함수 이름 및 기능은 동일하되 구현 방법은 차이가 남
  - ✓ CCircle :  $\pi * r * r$ ;
  - ✓ CSphere :  $4 * \pi * r * r$ ;
- 함수 오버라이딩!

### 3. base 클래스의 접근 제어와 protected 멤버

#### # 액세스 지정자의 위치 및 종류



#### # 액세스 지정자 public과 private에 대한 상속 권한 접근

base 멤버 \ 액세스 지정자	public		private	
	포함 영역	내부 접근	포함 영역	내부 접근
public	public	O	private	X
private	private	O	private	X

### 3. base 클래스의 접근 제어와 protected 멤버

#### # 다음 프로그램의 문제점은?

```
#define PI 3.14

class CCircle {
private :           // 멤버 변수를 private으로 선언
    int x, y;       // 중심
    double Radius;  // 반지름

public :
    double GetArea() { return (PI * Radius * Radius); }
};

class CSphere : public CCircle {
private :
    int z;

public :
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};

int main(void)
{
    return 0;
}
```

public 상속

base 클래스의 private 멤버로의  
직접 접근 허용하지 않음

### 3. base 클래스의 접근 제어와 protected 멤버

#### # protected 멤버

- 외부 접근 허용하지 않음 → private과 동일
- derived 클래스에서의 접근 허용 → public과 동일

#### # 액세스 지정자에 대한 상속 권한 접근

- protected 포함

base 멤버 \ 액세스 지정자	public		protected		private	
	포함 영역	내부 접근	포함 영역	내부 접근	포함 영역	내부 접근
public	public	O	protected	O	private	X
protected	protected	O	protected	O	private	X
private	private	O	private	O	private	X

#### # 주로 public 상속 사용!

### 3. base 클래스의 접근 제어와 protected 멤버

# 예 : CCircle의 멤버 변수들을 protected 멤버로 포함

```
#define PI 3.14

class CCircle {
protected :           // 멤버 변수를 protected로 선언
    int x, y;         // 중심
    double Radius;    // 반지름

public :
    double GetArea() { return (PI * Radius * Radius); }
};

class CSphere : public CCircle {
private :
    int z;

public :
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); };
};

int main(void)
{
    CCircle Cir;
    Cir.x = 5;
    return 0;
}
```

에러 : x는 protected 멤버 → 외부 접근은 허용되지 않음

## 4. 상속 관계에서의 생성자와 소멸자

### 예 : CSphere의 생성자 추가

```
#define PI 3.14

class CCircle {
protected :
    int x, y;           // 중심
    double Radius;     // 반지름

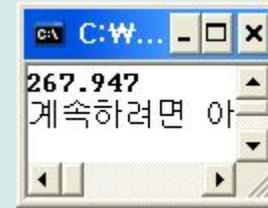
public :
    double GetArea() { return (PI * Radius * Radius); }
};

class CSphere : public CCircle {
private :
    int z;

public :
    CSphere(int a, int b, int c, double r)           // 생성자
        { x = a; y = b; z = c; Radius = r; }
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};

int main(void)
{
    CSphere Sph(1, 2, 3, 4);
    cout << Sph.GetVolume() << endl;

    return 0;
}
```

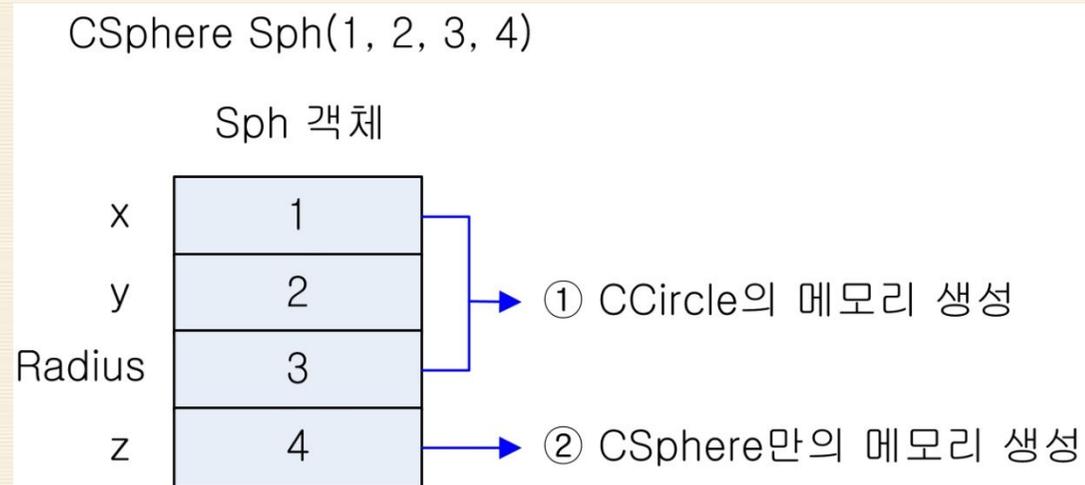


생성자 : base 클래스의 **protected**에 대한 접근 가능  
→ 문제 없이 수행됨  
→ **CCircle** 클래스에도 생성자를 추가한다면???

## 4. 상속 관계에서의 생성자와 소멸자

### ▣ derived 클래스 객체 생성 시 메모리 생성 순서

- base 클래스 부분 생성 → derived 클래스 부분 생성



### ▣ derived 클래스 객체 생성 시 생성자의 수행 순서

- CCircle 클래스 생성자 수행 → CSphere 클래스 생성자 수행
  - 앞의 예에서 CCircle 클래스의 생성자는? 디폴트 생성자!
  - CCircle 클래스의 생성자를 명시적으로 호출하는 방법???
- 멤버 초기화 구문

## 4. 상속 관계에서의 생성자와 소멸자

✦ 예 : base 클래스 생성자의 호출

```
#define PI 3.14

class CCircle {
protected :
    int x, y;           // 중심
    double Radius;     // 반지름

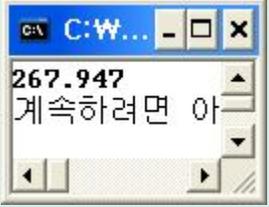
public :
    CCircle(int a, int b, double r) : x(a), y(b), Radius(r) { }
    double GetArea() { return (PI * Radius * Radius); }
};

class CSphere : public CCircle {
private :
    int z;

public :
    CSphere(int a, int b, int c, double r) : CCircle(a, b, r), z(c) { }
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};

int main(void)
{
    CSphere Sph(1, 2, 3, 4);
    cout << Sph.GetVolume() << endl;

    return 0;
}
```



base 클래스 생성자 호출

## 4. 상속 관계에서의 생성자와 소멸자

### # 고려 사항

- 다음과 같은 초기화 구문 사용 불가
  - `CSphere(int a, int b, int c, double r) : x(a), y(b), z(c), Radius(r) { }`
  - 아직 `x, y, Radius` 변수가 생성되기 전이므로 초기화 불가능
    - ✓ base 클래스의 멤버 변수는 반드시 해당(base) 클래스의 생성자로 초기화
- 앞의 예에서 다음과 같은 초기화는 가능할까?
  - `CSphere(int a, int b, int c, double r) { x = a; y = b; z = c; Radius = r; }`
  - 이 생성자가 수행되기 전에 `CCircle`의 생성자가 수행되어야만 됨
    - ✓ 이 경우 디폴트 생성자(매개변수가 없는 생성자)가 수행됨
      - `CCircle` 클래스에 디폴트 생성자가 존재하지 않으므로 수행 불가

## 4. 상속 관계에서의 생성자와 소멸자

# 상속 관계에서 소멸자의 호출 순서 : 생성자의 역순!

```
#define PI 3.14
```

```
class CCircle {  
protected :
```

```
    int x, y;           // 중심  
    double Radius;    // 반지름
```

```
public :
```

```
    CCircle(int a, int b, double r) : x(a), y(b), Radius(r) {  
        cout << "CCircle 생성자" << endl; }  
    ~CCircle() { cout << "CCircle 소멸자" << endl; }  
    double GetArea() { return (PI * Radius * Radius); }
```

```
};
```

```
class CSphere : public CCircle {
```

```
private :  
    int z;
```

```
public :
```

```
    CSphere(int a, int b, int c, double r) : CCircle(a, b, r), z(c) {  
        cout << "CSphere 생성자" << endl; }  
    ~CSphere() { cout << "CSphere 소멸자" << endl; }  
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
```

```
};
```

CSphere 객체 하나에  
대한 소멸자 호출 주목

```
int main(void)
```

```
{
```

```
    CSphere Sph(1, 1, 1, 1);
```

```
    cout << Sph.GetArea() << endl;
```

```
    cout << Sph.GetVolume() << endl;
```

```
    return 0;
```

```
}
```

```
C:\WWIND... - □ ×  
CCircle 생성자  
CSphere 생성자  
3.14  
4.18667  
CSphere 소멸자  
CCircle 소멸자  
계속하려면 아무 키
```

## 5. 함수 재정의

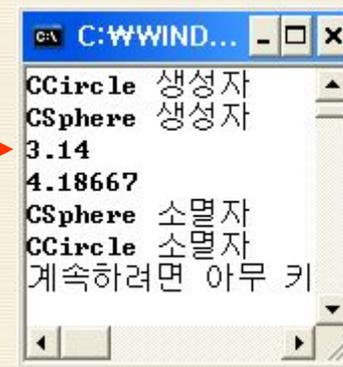
# 앞의 예에서 CSphere 객체의 표면적(GetArea)에 주목

- 구의 표면적 =  $4 * \pi * r * r = 12.56$
- 예에서는 원의 표면적 계산 함수 GetArea를 상속받아 사용했기 때문에  
→  $\pi * r * r = 3.14$ 가 나옴

```
int main(void)
{
    CSphere Sph(1, 1, 1, 1);

    cout << Sph.GetArea() << endl;
    cout << Sph.GetVolume() << endl;

    return 0;
}
```



- # 원의 면적, 구의 표면적을 구하는 함수 모두 GetArea로 하되 GetArea 함수를 호출하는 객체에 따라 자신의 함수가 수행되도록 하는 방법
- 함수 재정의(function overriding)

## 5. 함수 재정의

### ■ 함수 재정의를 통해 CSphere의 GetArea 함수 구현

```
// class CCircle 생략

class CSphere : public CCircle {
private :
    int z;

public :
    CSphere(int a, int b, int c, double r) : CCircle(a, b, r), z(c) {
        cout << "CSphere 생성자" << endl; }
    ~CSphere() { cout << "CSphere 소멸자" << endl; }
    double GetArea() { return (4 * PI * Radius * Radius); } // 함수 재정의
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};

int main(void)
{
    CSphere Sph(1, 1, 1, 1);

    cout << Sph.GetArea() << endl; // CSphere의 GetArea 함수 호출
    cout << Sph.GetVolume() << endl;

    return 0;
}
```



CCircle의 GetArea 함수는 상속되는가?  
CCircle 객체를 통해서만 호출 가능한가?

## 5. 함수 재정의

### # base 클래스(CCircle)의 GetArea 함수 호출

- CCircle 클래스의 GetArea 멤버 함수 역시 상속됨
- CSphere의 멤버 함수에서 CCircle의 GetArea를 호출하는 예
  - 다음 코드(CSphere의 멤버 함수 GetArea에서)의 의미는?
    - ✓ `double GetArea() { return (4 * GetArea()); }`
    - ✓ CSphere 멤버 함수 GetArea의 재귀호출 → 무한 반복 호출
  - CCircle의 GetArea를 호출하는 방법
    - ✓ `double GetArea() { return (4 * CCircle::GetArea()); }`
    - ✓ 범위 지정 연산자 :: 사용
- CSphere 클래스 외부에서 CSphere 객체를 통한 함수 호출
  - `CSphere sph;`
  - `Sph.CCircle::GetArea();` // CCircle의 GetArea 함수 호출, :: 사용

### # 만약 CSphere 클래스에 int x, y가 선언된다면

- CCircle의 x, y 역시 존재함
- `CCircle::x`, `CCircle::y` 와 같이 접근 가능

## 6. 디폴트 액세스 지정자와 구조체

### # 클래스와 구조체의 차이

- public, protected, private 영역 구분이 없을 경우의 디폴트 영역
  - 클래스 : private
  - 구조체 : public
- 상속 클래스 작성 시 derived 클래스의 디폴트 액세스 지정자
  - derived 클래스가 클래스인 경우 : private
  - derived 클래스가 구조체인 경우 : public
  - base 클래스가 클래스이냐 구조체이냐는 무관

```
struct CSphere : CCircle { // derived가 struct이므로 디폴트로 public 상속
private :
    int z;
public :
    CSphere(int a, int b, int c, double r) : CCircle(a, b, r), z(c) { }
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};
```

다음과 동일  
**struct CSphere : public CCircle**

## 6. 디폴트 액세스 지정자와 구조체

# 다음 프로그램의 문제점은?

```
#define PI 3.14

class CCircle {
protected :
    int x, y;          // 중심
    double Radius;    // 반지름

public :
    CCircle(int a, int b, double r) : x(a), y(b), Radius(r) { }
    double GetArea() { return (PI * Radius * Radius); }
};

class CSphere : CCircle { // derived가 class이므로 디폴트로 private 상속
private :
    int z;

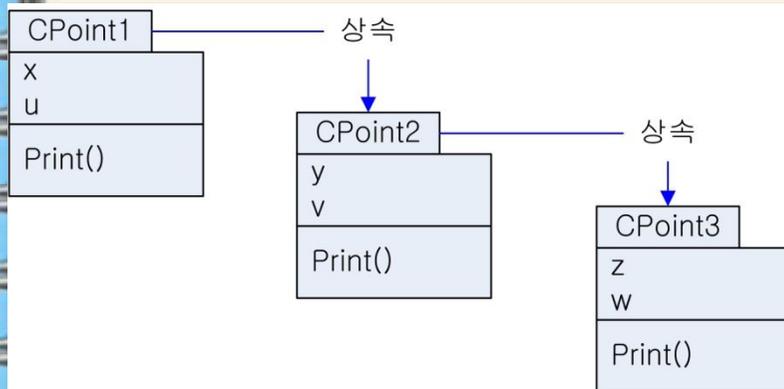
public :
    CSphere(int a, int b, int c, double r) : CCircle(a, b, r), z(c) { }
    double GetVolume() { return ((4.0/3.0) * PI * Radius * Radius * Radius); }
};

int main(void)
{
    CSphere Sph(1, 1, 1, 1);
    cout << Sph.GetArea() << endl;
    cout << Sph.GetVolume() << endl;
    return 0;
}
```

## 7. derived 클래스로부터의 상속

### # derived 클래스로부터의 상속 예

- 생성자와 소멸자의 호출 순서에 주의



```
class CPoint1 {
private :
    int x;

protected :
    int u;
```

```
public :
    CPoint1(int a) : x(a) { cout << "CPoint1 생성자" << endl; }
    ~CPoint1() { cout << "CPoint1 소멸자" << endl; }
    void Print() { cout << "CPoint1" << endl; }
};
```

```
class CPoint2 : public CPoint1 {
private :
    int y;
```

```
protected :
    int v;
```

```
public :
    CPoint2(int a, int b) : CPoint1(a), y(b)
        { cout << "CPoint2 생성자" << endl; }
    ~CPoint2() { cout << "CPoint2 소멸자" << endl; }
    void Print() { cout << "CPoint2" << endl; }
};
```

## 7. derived 클래스로부터의 상속

### # 코드 계속

```
class CPoint3 : public CPoint2 {
private :
    int z;

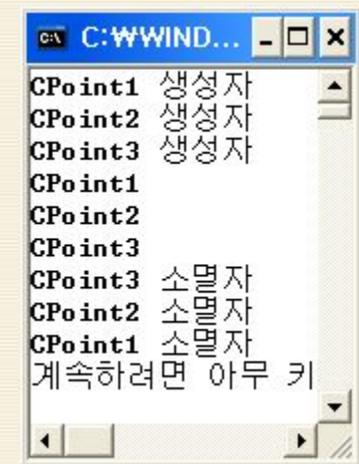
protected :
    int w;

public :
    CPoint3(int a, int b, int c) : CPoint2(a, b), z(c)
        { cout << "CPoint3 생성자" << endl; }
    ~CPoint3() { cout << "CPoint3 소멸자" << endl; }
    void Print() {
        CPoint1::Print();    // CPoint1의 Print 함수 호출
        CPoint2::Print();    // CPoint2의 Print 함수 호출
        cout << "CPoint3" << endl;
    }
};

int main(void)
{
    CPoint3 P3(1, 2, 3);
    P3.Print();

    return 0;
}
```

**Print 함수 재정의**  
**CPoint1, CPoint2의 Print 함수**  
**모두 상속됨**



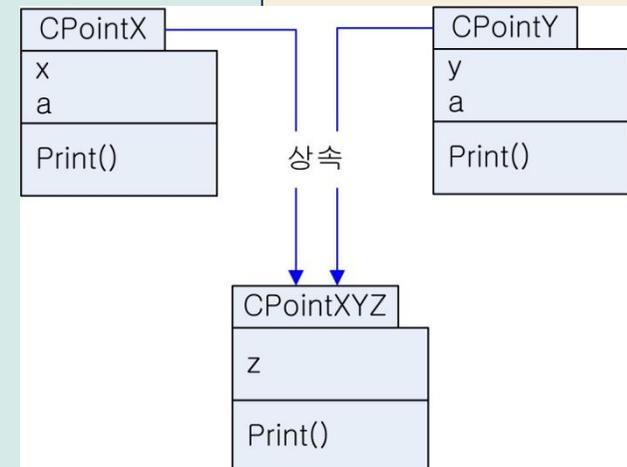
```
C:\> C:\WWIND...
CPoint1 생성자
CPoint2 생성자
CPoint3 생성자
CPoint1
CPoint2
CPoint3
CPoint3 소멸자
CPoint2 소멸자
CPoint1 소멸자
계속하려면 아무 키
```

## 8. 다중 상속

# 다중 상속 : 2개 이상의 클래스로부터 동시에 상속받는 경우

```
class CPointX {  
protected :  
    int x;  
    int a;  
  
public :  
    CPointX(int a) : x(a) { cout << "CPointX 생성자" << endl; }  
    ~CPointX() { cout << "CPointX 소멸자" << endl; }  
    void Print() { cout << "CPointX" << endl; }  
};
```

```
class CPointY {  
protected :  
    int y;  
    int a;  
  
public :  
    CPointY(int b) : y(b) { cout << "CPointY 생성자" << endl; }  
    ~CPointY() { cout << "CPointY 소멸자" << endl; }  
    void Print() { cout << "CPointY" << endl; }  
};
```



## 8. 다중 상속

### # 코드 계속

```
class CPointXYZ : public CPointX, public CPointY {    // 다중 상속
private :
    int z;

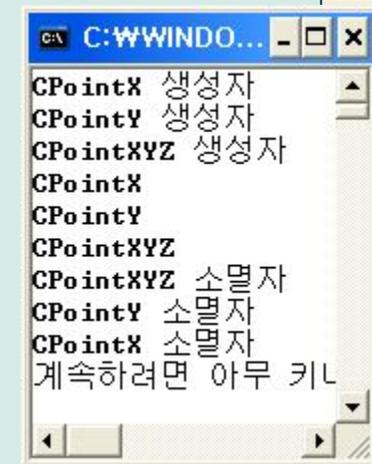
public :
    CPointXYZ(int a, int b, int c) : CPointX(a), CPointY(b), z(c)
        { cout << "CPointXYZ 생성자" << endl; }
    ~CPointXYZ() { cout << "CPointXYZ 소멸자" << endl; }
    void Print() {
        // cout << "a : " << a << endl; // 에러발생, 어떤 a?
        CPointX::Print(); // CPointX의 Print 함수 호출
        CPointY::Print(); // CPointY의 Print 함수 호출
        cout << "CPointXYZ" << endl;
    }
};

int main(void)
{
    CPointXYZ Pxyz(1, 2, 3);
    Pxyz.Print();

    return 0;
}
```

생성자 호출 순서 : 어느 쪽을 따를까?  
위 쪽(클래스 선언 시작 시 명시된 순서)

모호성 문제 발생  
CPointX::a, CPointY::a와 같이 사용



```
C:\WWINDO... - _ x
CPointX 생성자
CPointY 생성자
CPointXYZ 생성자
CPointX
CPointY
CPointXYZ
CPointXYZ 소멸자
CPointY 소멸자
CPointX 소멸자
계속하려면 아무 키
```

# 9. virtual base 클래스

## # 다중 상속의 예 및 고려 사항

```
class CPointX {
protected :
    int x;

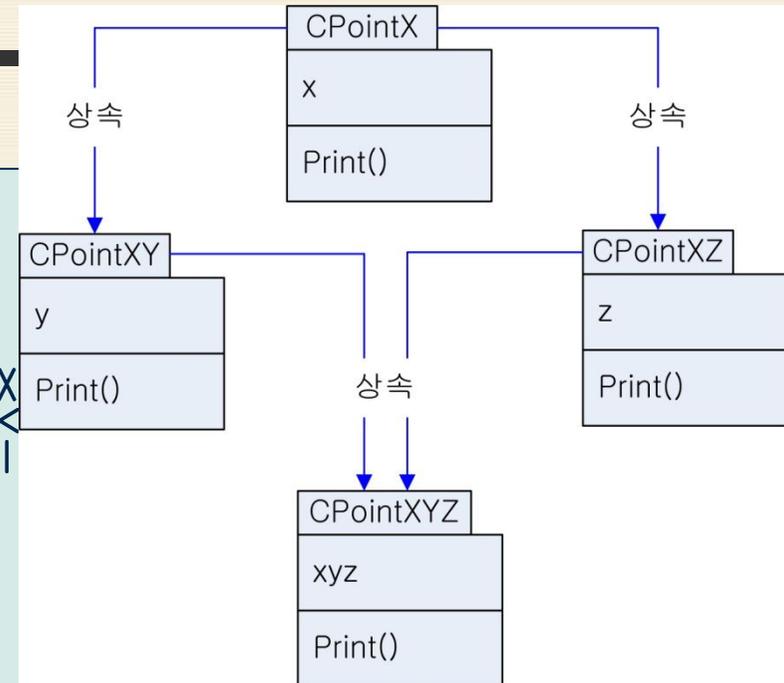
public :
    CPointX(int a) : x(a) { cout << "CPointX 생성자" << endl; }
    ~CPointX() { cout << "CPointX 소멸자" << endl; }
    void Print() { cout << "CPointX" << endl; }
};

class CPointXY : public CPointX {
protected :
    int y;

public :
    CPointXY(int a, int b) : CPointX(a), y(b)
        { cout << "CPointXY 생성자" << endl; }
    ~CPointXY() { cout << "CPointXY 소멸자" << endl; }
    void Print() { cout << "CPointXY" << endl; }
};

class CPointXZ : public CPointX {
protected :
    int z;

public :
    CPointXZ(int a, int c) : CPointX(a), z(c)
        { cout << "CPointXZ 생성자" << endl; }
    ~CPointXZ() { cout << "CPointXZ 소멸자" << endl; }
    void Print() { cout << "CPointXZ" << endl; }
};
```



# 9. virtual base 클래스

## # 코드 계속

```
class CPointXYZ : public CPointXY, public CPointXZ {
private :
    int xyz;

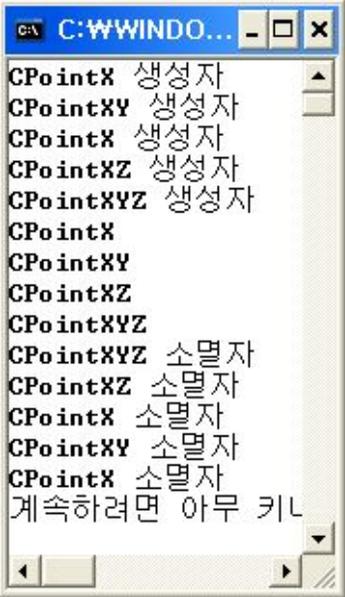
public :
    CPointXYZ(int a, int b, int c) : CPointXY(a, b), CPointXZ(a, c), xyz(0)
        { cout << "CPointXYZ 생성자" << endl; }
    ~CPointXYZ() { cout << "CPointXYZ 소멸자" << endl; }
    void Print() {
        // cout << "x : " << x << endl; // 에러발생, 어떤x?
        CPointX::Print(); // VC++ 6.0에서는 에러 발생
        CPointXY::Print();
        CPointXZ::Print();
        cout << "CPointXYZ" << endl;
    }
};
```

모호성 문제 발생

x 2개, y 1개, z 1개 존재  
CPointX로부터 단 한번만  
상속받는 방법은  
→ virtual base 클래스 지정

```
int main(void)
{
    CPointXYZ Pxyz(1, 2, 3);
    Pxyz.Print();

    return 0;
}
```



class CPointXY : virtual public CPointX { ... }  
class CPointXZ : virtual public CPointX { ... }  
수정 후 출력 결과를 비교해 보라. (디폴트 생성자 필수)

## 10. derived 클래스의 디폴트 복사 생성자와 디폴트 대입 연산자

### ⌘ 디폴트 복사 생성자와 디폴트 대입 연산자의 기본 동작 방식

- 멤버 단위 복사

```
class CPoint {  
private :  
    int x, y;  
  
public :  
    CPoint(const CPoint &Po) { x = Po.x; y = Po.y; } // 복사 생성자  
    CPoint &operator=(const CPoint &Po) // 대입 연산자  
        { x = Po.x; y = Po.y; return (*this); }  
};
```

### ⌘ derived 클래스의 디폴트 복사 생성자와 디폴트 대입 연산자

- 멤버 단위 복사?
- 다음과 같이 멤버 단위 복사를 수행하는 복사 생성자의 문제점은?

```
class CPoint3 : public CPoint {  
    CPoint3(const CPoint3 &Po) { x = Po.x; y = Po.y; z = Po.z; }  
}
```

x 접근 불가 : CPoint의 private 멤버임!

## 10. derived 클래스의 디폴트 복사 생성자와 디폴트 대입 연산자

### # derived 클래스의 디폴트 복사 생성자와 디폴트 대입 연산자의 동작

- 멤버 단위 복사
- base 클래스의 복사 생성자 및 대입 연산자 호출

```
class CPoint3 :public CPoint {
private :
    int z;

public :
    CPoint3(const CPoint3 &Po) : CPoint(Po) { z = Po.z; }
    CPoint3 &operator=(const CPoint3 &Po) {
        CPoint::operator=(Po); z = Po.z;
        return (*this);
    }
};
```

- 만약, base 클래스의 복사 생성자와 대입 연산자를 private 멤버로 명시적으로 작성하고, derived 클래스에서는 명시적으로 작성하지 않는다면?
  - base 객체와 derived 객체에 대한 복사 생성 및 대입 연산 불가

# 11. private 생성자의 사용

# 다음 프로그램의 문제점은?

```
class CPoint {
private :
    int x, y;
    CPoint(int a, int b) : x(a), y(b) { }    // 생성자, private 멤버임

public :
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};

int main(void)
{
    CPoint P1(3, 4); ← private 영역에 있는 생성자 호출 불가 → 에러 발생
    P1.Print();

    return 0;
}
```

# 생성자를 private 영역에 위치시키는 경우도 있을까?

# 11. private 생성자의 사용

## # private 생성자의 사용 예

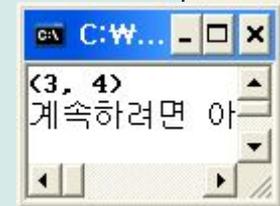
- 프로그램 전체적으로 유일한 객체 생성 및 사용

```
class CPoint {
private :
    int x, y;
    CPoint(int a, int b) : x(a), y(b) { }
    ~CPoint() { if (OnlyPoint != NULL) delete OnlyPoint; }
    static CPoint *OnlyPoint;      // 유일한 CPoint 객체를 가리킬 포인터

public :
    static CPoint *GetPoint() {      // OnlyPoint를 반환하는 함수
        if (OnlyPoint == NULL)      // 최초 수행 시 객체 생성
            OnlyPoint = new CPoint(3, 4);
        return OnlyPoint;
    }
    void Print() { cout << "(" << x << ", " << y << ")" << endl; }
};

CPoint *CPoint::OnlyPoint = NULL;  // 초기화, 아직 객체 생성 전

int main(void)
{
    CPoint::GetPoint()->Print();
    return 0;
}
```



디자인 패턴 : 반복적으로 발생하는 문제에 대한  
해법 연구

예 : **Singleton** 패턴 – 객체가 오직 하나만 존재  
→ 표준 C++과는 별개의 주제