



[이것이 C++ 이다]



Chapter 6

# 상속 기본

## 6장의 핵심 개념

상속, 재정의, 메서드 재정의

- **상속**  
: 객체단위로 코드를 재사용하거나 확장/개선하는 방법이다.
- **재정의**  
: 함수의 기존 정의를 새롭게 바꾸는(혹은 대체하는) 방법이다.
- **메서드 재정의**  
: Overriding은 보통 메서드 재정의이다.

- 추상화

인간용어이던.

폴리모피즘 (다형성)

# 상속이란?

상속은 객체 단위 코드를 재사용하는 방법이다.

- 상속을 통해 코드를 **재사용**하거나 **확장/개선**할 수 있다.
- 상속을 배우는 순간부터 클래스는 최소 2개 이상이 되고 이들 간의 **관계**를 이해하는 것이 매우 중요하다.
- 상속은 is-a, has-a 관계이다.
- 파생 클래스의 인스턴스가 생성될 때 기본 클래스 생성자도 호출된다.
- 파생 클래스는 기본 클래스의 멤버에 접근할 수 있다. 단, **private** 접근제어 지시자로 선언된 멤버는 접근할 수 없다.
- 파생 클래스 인스턴스를 통해 기본 클래스 메서드도 호출할 수 있다.

베이스 클래스      파생 클래스  
super                  sup

자식 클래스가 부모 클래스 멤버 기능!

# 기본 문법

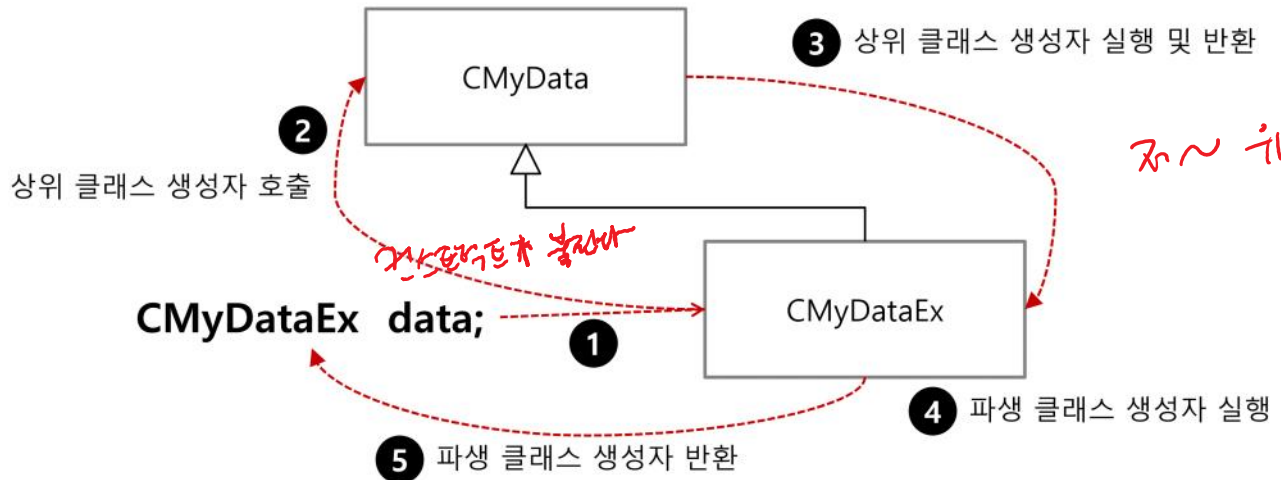
class 파생 클래스이름 : 접근제어 지시자 부모 클래스 이름

```
// 기본 클래스 혹은 부모 클래스
class CMyData
{
};

// 파생 클래스 혹은 자식 클래스
class CMyDataEx : public CMyData
{
};
```

# 기본 문법

상속 관계에서 생성자 호출 순서와 실행 순서는 다르다.



# 기본 문법

파생 클래스는 기본 클래스 멤버에 접근할 수 있다.

```
class CMyData
{
...
    int GetData() { return m_nData; }
    void SetData(int nParam) { m_nData = nParam; }
protected:
    // 파생 클래스만 접근 가능
    void PrintData() { cout << "CMyData::PrintData()" << endl; }
...
};

class CMyDataEx : public CMyData
{
...
    void TestFunc()
    {
        // 기본 형식 멤버에 접근
        PrintData();
        SetData(5);
    }
};
```

*Handwritten notes:*

- ~~private public~~ (next to CMyData)
- public (circled next to CMyDataEx)
- 이 자식도 가능.* (next to PrintData() in CMyData)
- 기본* (next to TestFunc() in CMyDataEx)
- 기본* (next to // 기본 형식 멤버에 접근)
- private* (underlined, next to SetData(5))

# 기본 문법

파생 클래스 인스턴스를 이용해 부모 클래스 메서드에 접근할 수 있다.

```
// 사용자
int _tmain(int argc, _TCHAR* argv[])
{
    CMyDataEx data;
    // 기본 클래스(CMyData) 멤버에 접근
    data.SetData(10);
    cout << data.GetData() << endl;
    // 파생 클래스(CMyDataEx) 멤버에 접근
    data.TestFunc();
    return 0;
}
```



# 메서드 재정의

메서드를 재정의하면 기존의 것이 '무시'된다.

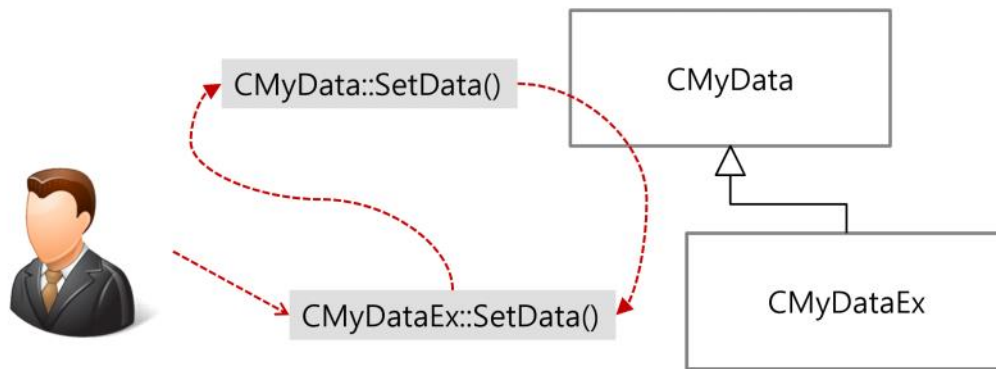
```
// 초기 제작자
class CMyData
{...
    void SetData(int nParam) { m_nData = nParam; }
...
};
// 후기 제작자
class CMyDataEx : public CMyData
{
public:
    // 파생 클래스에서 기본 클래스의 메서드를 재정의했다.
    void SetData(int nParam)
    {
        // 입력 데이터의 값을 보정하는 새로운 기능을 추가한다.
        if (nParam < 0)
            CMyData::SetData(0);
        if (nParam > 10)
            CMyData::SetData(10);
    }
};
```

Handwritten notes in red:

- A circle around the word "재정의" (redefinition) in the comment: "생략 가능한 것이" (something that can be omitted).
- A circle around the base class method call `CMyData::SetData(0);` with the note "이것도" (this too).
- A checkmark and note: "복합의 가치 불러서 생각해." (Think about the value of composition).

# 메서드 재정의

메서드를 재정의하면 기존의 것이 '무시'된다.

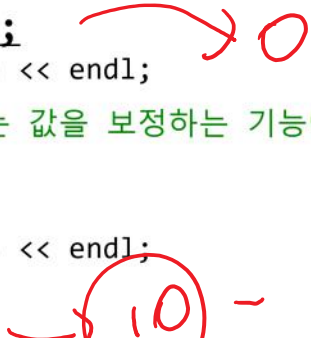


# 메서드 재정의

메서드를 재정의하면 기존의 것이 '무시'된다.

```
// 사용자 코드
int _tmain(int argc, _TCHAR* argv[])
{
    // 구형에서는 값을 보정하는 기능이 없다.
    CMyData a;
    a.SetData(-10);
    cout << a.GetData() << endl;

    // 새로 만든 신형에는 값을 보정하는 기능이 있다.
    CMyDataEx b;
    b.SetData(15);
    cout << b.GetData() << endl;
    return 0;
}
```



# 참조 형식과 실 형식

파생 형식을 기본 형식으로 참조하는 것은 자연스러운 일이다.  
단, 이때 어떤 메서드가 호출되는지 정확히 알고 있어야 한다.

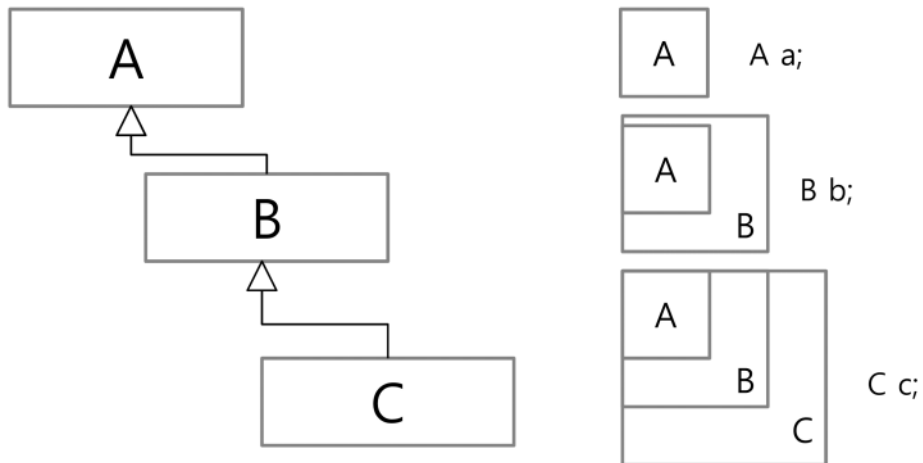
```
// 사용자 코드
int _tmain(int argc, _TCHAR* argv[])
{
    CMyDataEx a;
    CMyData &rData = a;

    rData.SetData(15);

    cout << rData.GetData() << endl;
    return 0;
}
```

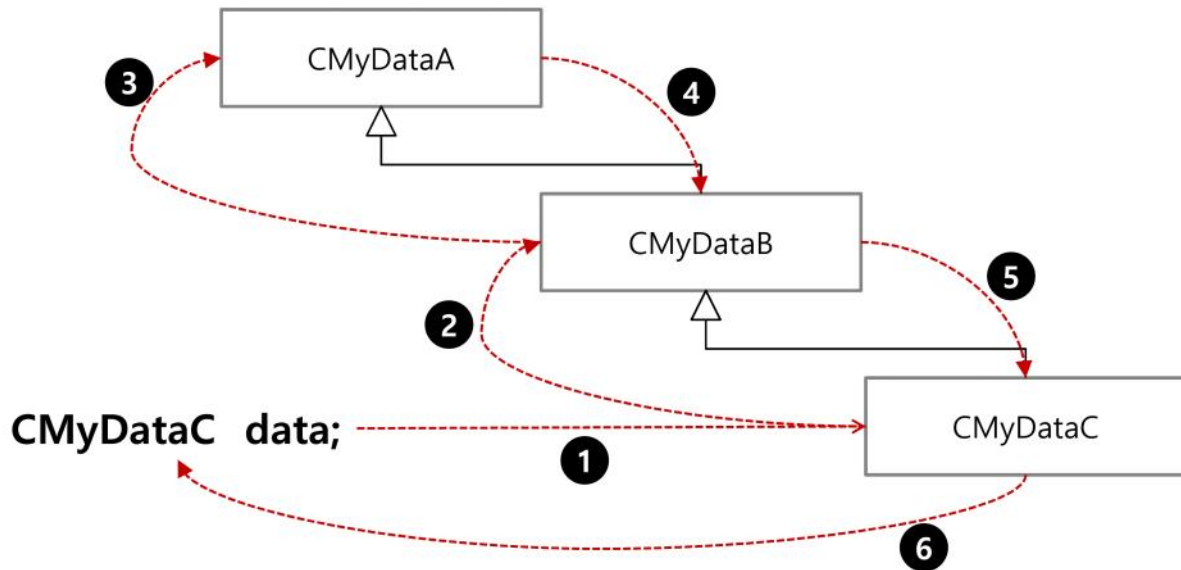
# 상속에서의 생성자와 소멸자

상속을 통해 클래스(코드) 덩치는 점점 커진다.



# 상속에서의 생성자와 소멸자

파생 클래스 생성자가 **가장 먼저 호출되지만 가장 나중에 실행**된다.  
재귀 호출은 아니지만 구조적으로 매우 흡사하다.



# 상속에서의 생성자와 소멸자

생성자에서는 객체 자신을 초기화하는 코드만 기술하라.

절대로 파생 클래스에서 부모 클래스 멤버를 초기화 하면 안 된다.

같은 맥락에서 파생 클래스 소멸자에서 부모 클래스 메모리를 직접 관리하면 안 된다.

```
class CMyDataA
{
...
protected:
    char *m_pszData;
};
class CMyDataB : public CMyDataA
{
...
};
class CMyDataC : public CMyDataB
{
public:
...
    ~CMyDataC() {
        // 파생 클래스에서 부모 클래스 멤버 메모리를 해제했다.
        delete m_pszData;
    }
};
```

# 생성자 선택

파생 클래스 생성자에서 호출하려는 부모 클래스 생성자를 선택할 수 있다.

```
class CMyData
{
public:
    CMyData() { cout << "CMyData()" << endl; }
    CMyData(int nParam) { cout << "CMyData(int)" << endl; }
    CMyData(double dParam) { cout << "CMyData(double)" << endl; }
};
class CMyDataEx : public CMyData
{
public:
    CMyDataEx() { cout << "CMyDataEx()" << endl; }
    CMyDataEx(int nParam) : CMyData(nParam)
    {
        cout << "CMyDataEx(int)" << endl;
    }
    CMyDataEx(double dParam) : CMyData()
    {
        cout << "CMyDataEx(double)" << endl;
    }
};
```

생성자

자식

default  
int

double

의미는? 선택해 호출하는지 말해?