22장. 상속과 포함

01_ 포함

02_ 상속

박 종 혁 교수

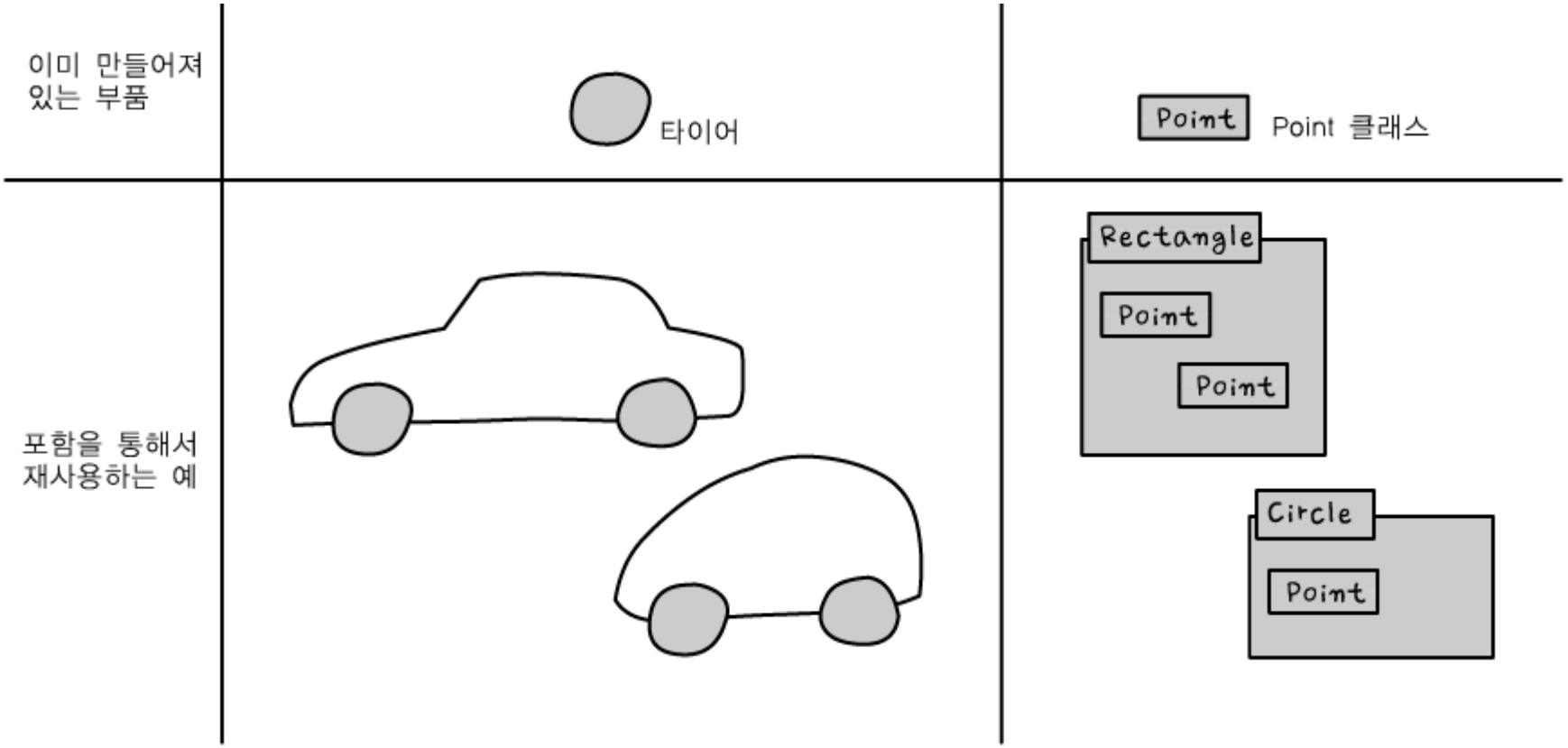
UCS Lab

Tel: 970-6702

Email: jhpark1@seoultech.ac.kr

포함을 사용한 재사용

- 포함을 사용한 재사용의 예



객체를 멤버 변수로 갖는 클래스(1)

- Point 클래스를 멤버 변수로 갖는 Rect 클래스의 예

```
// Rect.h
#include "Point.h"

class Rect
{
public:
          // 생성자
          Rect();

          // 각 점의 값 지정/얻기
          void SetTopLeft(const Point& topLeft);
          void SetBottomRight(const Point& bottomRight);
          void SetRect(int left, int top, int right, int bottom);
          Point GetTopLeft() const;
          Point GetBottomRight() const;
          void GetRect(int& left, int& top, int& right, int& bottom);

          // 넓이, 높이 계산
          int GetWidth() const;
          int GetHeight() const;

          // 내용 출력
          void Print() const;

protected:
          Point _topLeft;
          Point _bottomRight;
};
```

객체를 멤버 변수로 갖는 클래스(2)

- Point 클래스를 멤버 변수로 갖는 Rect 클래스의 예

```
// Rect.cpp
#include "rect.h"
#include <iostream>
using namespace std;

Rect::Rect()
{
}

void Rect::SetTopLeft(const Point& topLeft)
{
          _topLeft = topLeft;
}
void Rect::SetBottomRight(const Point& bottomRight)
{
          _bottomRight = bottomRight;
}
void Rect::SetRect(int left, int top, int right, int bottom)
{
          _topLeft.SetX(left);
          _topLeft.SetY(top);
          _bottomRight.SetX(right);
          _bottomRight.SetY(bottom);
}
Point Rect::GetTopLeft() const
{
          return _topLeft;
}
```

객체를 멤버 변수로 갖는 클래스(3)

- Point 클래스를 멤버 변수로 갖는 Rect 클래스의 예

```
// Rect.cpp 이어서
void Rect::GetRect(int& left, int& top, int& right, int& bottom)
{
        left = _topLeft.GetX();
        top = _topLeft.GetY();
        right = _bottomRight.GetX();
        bottom = _bottomRight.GetY();
}

int Rect::GetWidth() const
{
        return (_bottomRight.GetX() - _topLeft.GetX() + 1);
}

int Rect::GetHeight() const
{
        return (_bottomRight.GetY() - _topLeft.GetY() + 1);
}

void Rect::Print() const
{
        cout << "{L=" << _topLeft.GetX() << ", T=" << _topLeft.GetY();
        cout << ", R=" << _bottomRight.GetX() << ", B=" << _bottomRight.GetY() << "}\n";
}
```

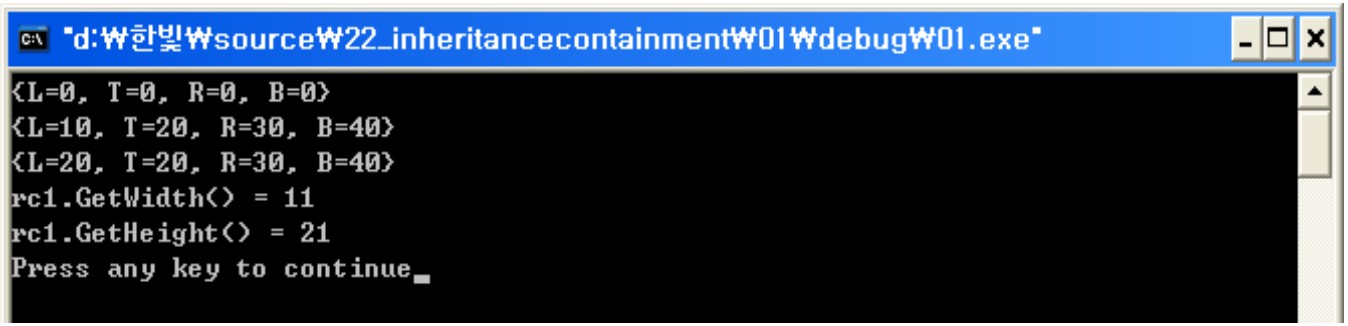
객체를 멤버 변수로 갖는 클래스(4)

- Point 클래스를 멤버 변수로 갖는 Rect 클래스의 예

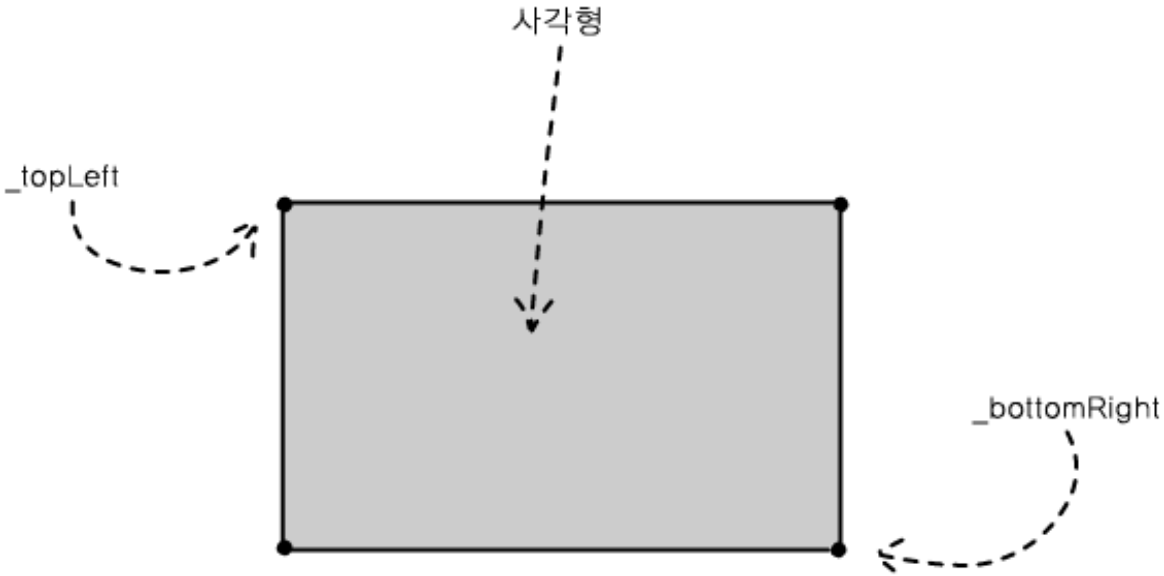
```
// Example.cpp
#include "Rect.h"
#include <iostream>
using namespace std;

int main()
{
          Rect rc1;                                    // Rect 객체 생성
          rc1.Print();                                 // 내용 출력

          rc1.SetRect( 10, 20, 30, 40);                // 값을 바꿔본다.
          rc1.Print();                                 // 내용 출력

          rc1.SetTopLeft( Point(20, 20));              // 또 값을 바꿔본다.
          rc1.Print();                                 // 내용 출력

          // 넓이, 높이 출력
          cout << "rc1.GetWidth() = " << rc1.GetWidth() << "\n";
          cout << "rc1.GetHeight() = " << rc1.GetHeight() << "\n";

          return 0;
}
```

객체를 멤버 변수로 갖는 클래스(5)

- 실행 결과



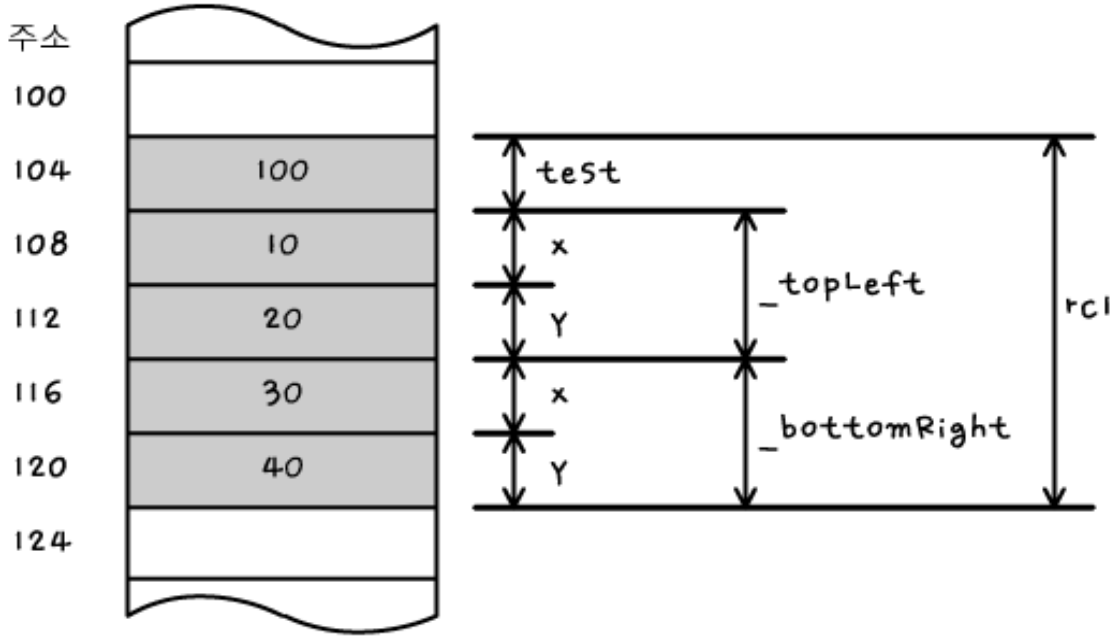
- Rect 클래스가 멤버 변수로 갖는 Point 객체 2개의 의미



객체를 포함한 경우의 메모리 구조

- 이해를 돕기 위해서 Rect 클래스에 int 타입의 멤버 변수를 추가해보고, 이 경우 Rect 객체의 메모리 구조를 살펴보자.

```
class Rect
{
// 중간 생략
protected:
          int test;
          Point _topLeft;
          Point _bottomRight;
};
```



생성자와 소멸자(1)

- 포함된 객체의 생성자를 직접 호출하는 예

```
// Rect.h
class Rect
{
public:
          // 생성자
          Rect();
          Rect(const Point& topLeft, const Point& bottomRight);
          Rect(int left, int top, int right, int bottom);
...
```

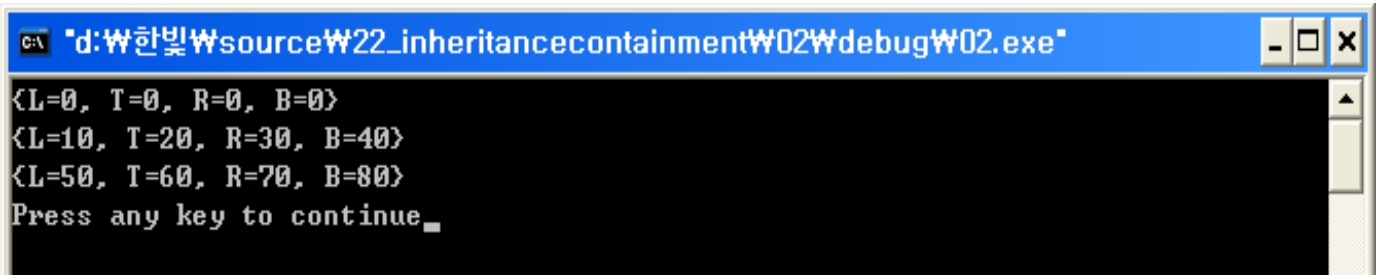
```
// Rect.cpp
Rect::Rect(const Point& topLeft, const Point& bottomRight)
: _topLeft( topLeft), _bottomRight( bottomRight)
{
}


Rect::Rect(int left, int top, int right, int bottom)
: _topLeft( left, top), _bottomRight( right, bottom)
{
}

...
```

생성자와 소멸자(2)

- 포함된 객체의 생성자를 직접 호출하는 예

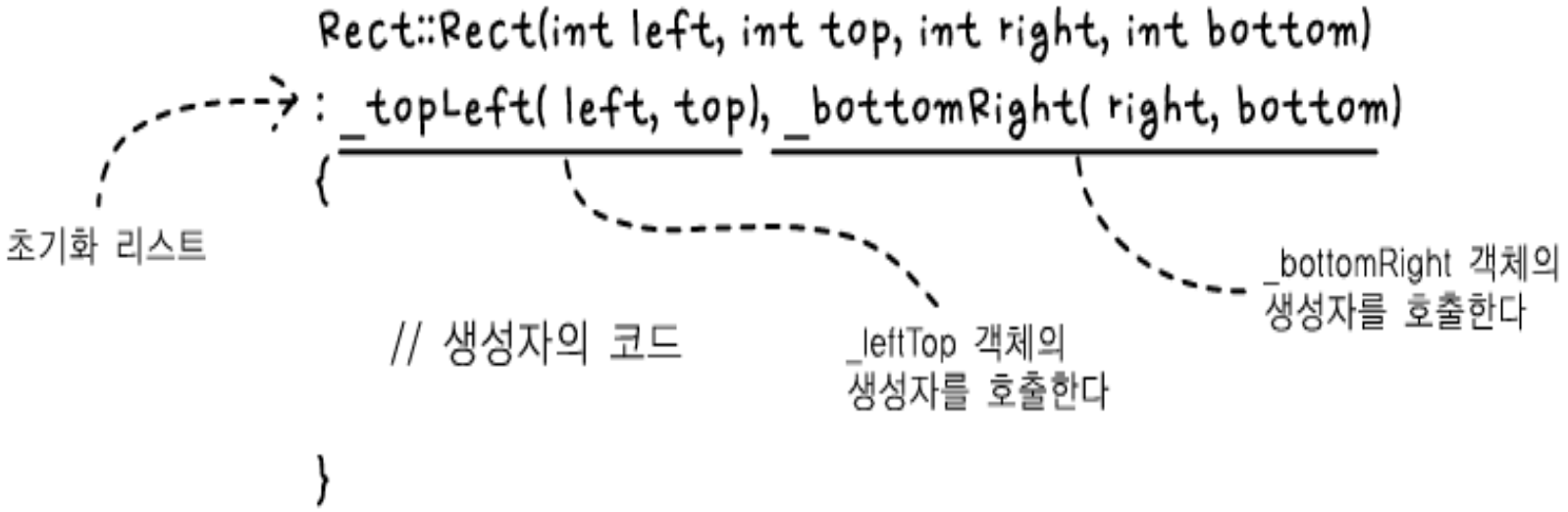
```
// Example.cpp
Rect rc1;
Rect rc2( Point(10, 20), Point(30, 40) );
Rect rc3( 50, 60, 70, 80);

// 내용 출력
rc1.Print();
rc2.Print();
rc3.Print();
```

- 실행 결과



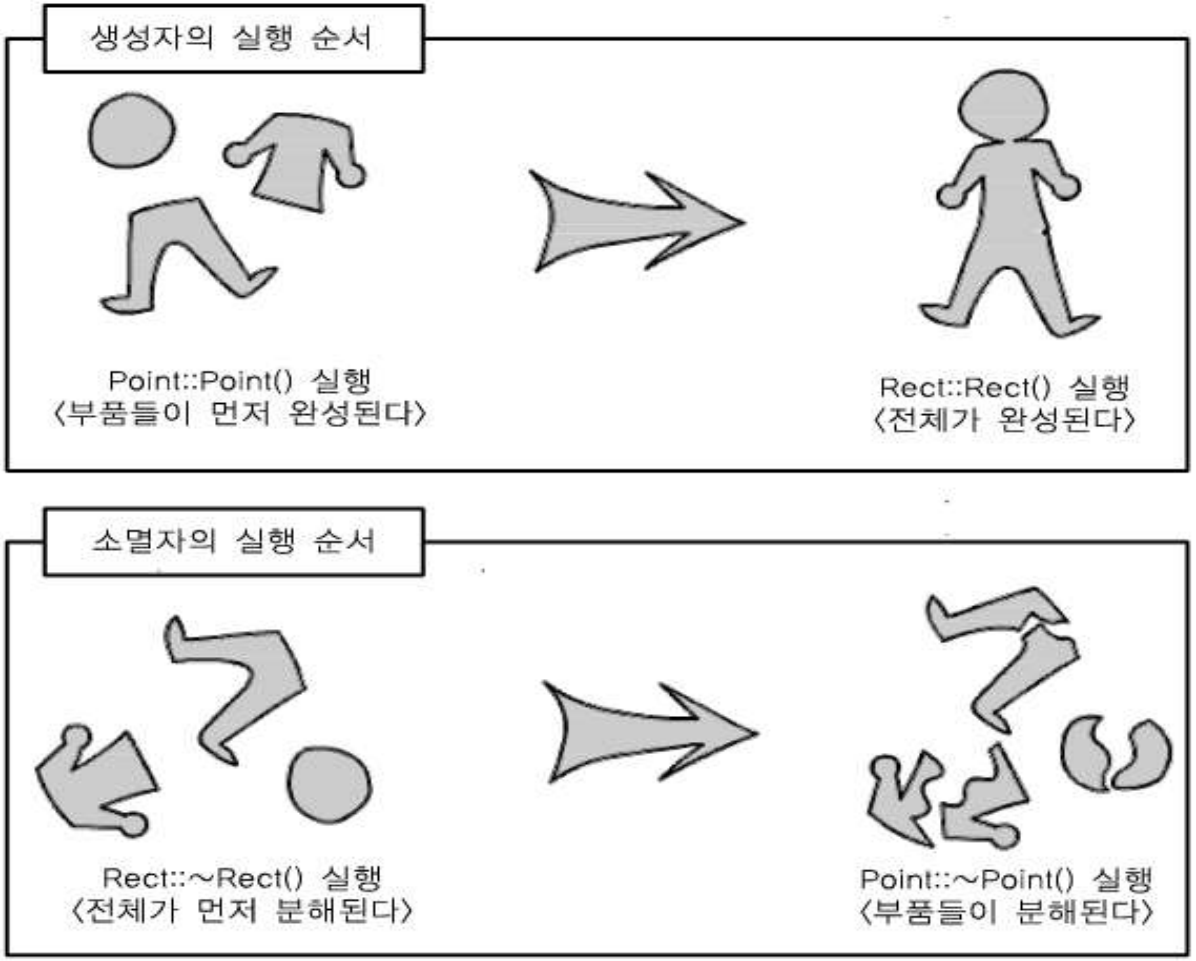
생성자와 소멸자(3)

- 초기화 리스트를 사용해서 멤버의 생성자 호출하기



생성자와 소멸자(7)

- Rect 클래스와 Point 클래스 각각의 생성자와 소멸자의 호출 순서



문서 저장 클래스(1)

- 상속을 공부하기 위해서 만든 문서 저장 클래스

```
// DocWriter.h
class DocWriter
{
public:
          DocWriter();
          DocWriter(const string& fileName, const string& content);
          ~DocWriter();

          // 파일 이름을 지정
          void SetFileName(const string& fileName);

          // 저장할 텍스트를 지정
          void SetContent(const string& content);

          // 파일에 텍스트를 저장시킨다.
          void Write();

protected:
          string _fileName;
          string _content;
};
```

문서 저장 클래스(2)

- 상속을 공부하기 위해서 만든 문서 저장 클래스

```
// DocWriter.cpp
DocWriter::DocWriter()
{
          // 파일이름과 텍스트를 디폴트로 지정시켜 놓는다.
          _fileName = "NoName.txt";
          _content = "There is no content.";
}

DocWriter::DocWriter(const string& fileName, const string& content)
{
          _fileName = fileName;
          _content = content;
}
DocWriter::~DocWriter()
{
}

// 파일 이름을 지정
void DocWriter::SetFileName(const string& fileName)
{
          _fileName = fileName;
}

// 저장할 텍스트를 지정
void DocWriter::SetContent(const string& content)
{
          _content = content;
}
```

문서 저장 클래스(3)

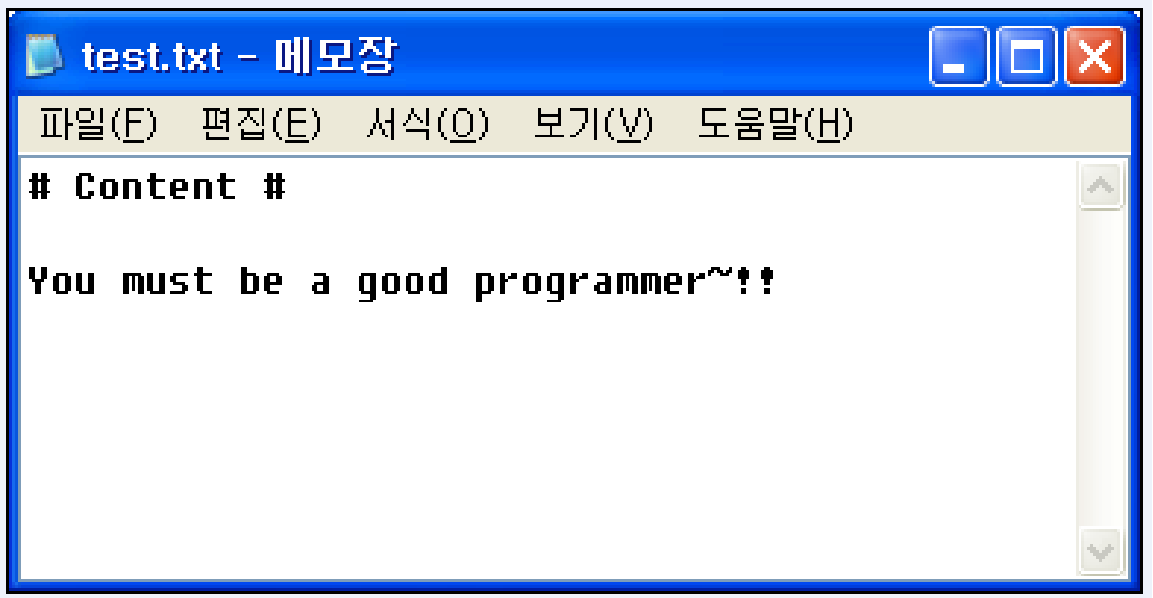
- 상속을 공부하기 위해서 만든 문서 저장 클래스

```
// DocWriter.cpp 이어서
// 파일에 텍스트를 저장시킨다.
void DocWriter::Write()
{
        // 파일을 연다.
        ofstream of( _fileName.c_str() );

        // 간단한 헤더를 출력한다.
        of << "# Content #\n\n";

        // 텍스트를 있는 그대로 저장한다.
        of << _content;
}
```

```
// Example.cpp
int main()
{
        DocWriter dw;
        dw.SetFileName( "test.txt" );
        dw.SetContent("You must be a good programmer~!!");
        dw.Write();

        return 0;
}
```

문서 저장 클래스(4)

- 결과로 생성된 test.txt 파일



HTML 저장 클래스(1)

- 문서 저장 클래스를 상속해서 만든 HTML 저장 클래스

```
// HTMLWriter.h
#include "DocWriter.h"

class HTMLWriter : public DocWriter
{
public:
          HTMLWriter();
          ~HTMLWriter();

          // 텍스트를 파일로 저장시킨다.
          void Write();

          // 폰트를 지정한다.
          void SetFont(const string& fontName, int fontSize, const string& fontColor);


protected:
          string    _fontName;
          int       _fontSize;
          string    _fontColor;
};
```

HTML 저장 클래스(2)

- 문서 저장 클래스를 상속해서 만든 HTML 저장 클래스

```
// HTMLWriter.cpp
HTMLWriter::HTMLWriter()
{
          // 디폴트 파일 이름만 바꾼다.
          _fileName = "NoName.html";

          // 디폴트 폰트를 지정한다.
          _fontName = "굴림";
          _fontSize = 3;
          _fontColor = "black";
}

HTMLWriter::~HTMLWriter()
{
}

// 폰트를 지정한다.
void HTMLWriter::SetFont(const string& fontName, int fontSize, const string&
fontColor)
{
          _fontName = fontName;
          _fontSize = fontSize;
          _fontColor = fontColor;
}
```

HTML 저장 클래스(3)

- 문서 저장 클래스를 상속해서 만든 HTML 저장 클래스

```
// HTMLWriter.cpp 이어서
// 파일에 텍스트를 저장시킨다.
void HTMLWriter::Write()
{
          // 파일을 연다.
          ofstream of( _fileName.c_str() );

          // HTML 헤더 부분을 저장한다.
          of << "<HTML><HEAD><TITLE>This document was generated by
HTMLWriter</TITLE></HEAD><BODY>";
          of << "<H1>Content</H1>";

          // 폰트 태그를 시작한다.
          of << "<Font name='" << _fontName << "' size='" << _fontSize << "' color='"
<< _fontColor << "'>";

          // 텍스트를  저장한다.
          of << _content;

          // 폰트 태그를 닫는다.
          of << "</FONT>";

          // HTML을 마무리 한다.
          of << "</BODY></HTML>";
}
```

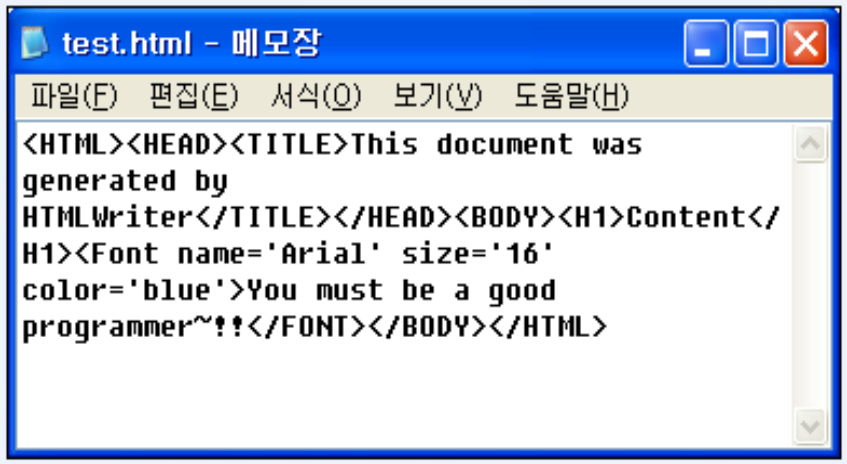
HTML 저장 클래스(4)

- 문서 저장 클래스를 상속해서 만든 HTML 저장 클래스

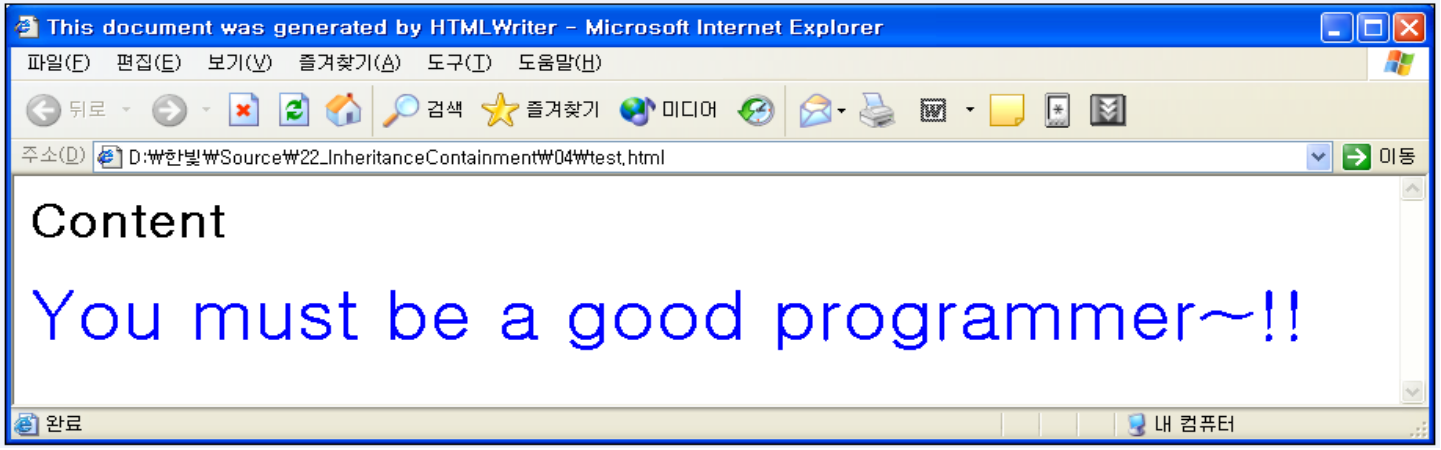
```
// Example.cpp
#include "HTMLWriter.h"

int main()
{
        HTMLWriter hw;
        hw.SetFileName( "test.html" );
        hw.SetContent("You must be a good programmer~!!");
        hw.SetFont("Arial", 16, "blue");
        hw.Write();

        return 0;
}
```

HTML 저장 클래스(5)

- 결과로 생성된 test.html 파일
  - 텍스트 편집기에서 연 경우

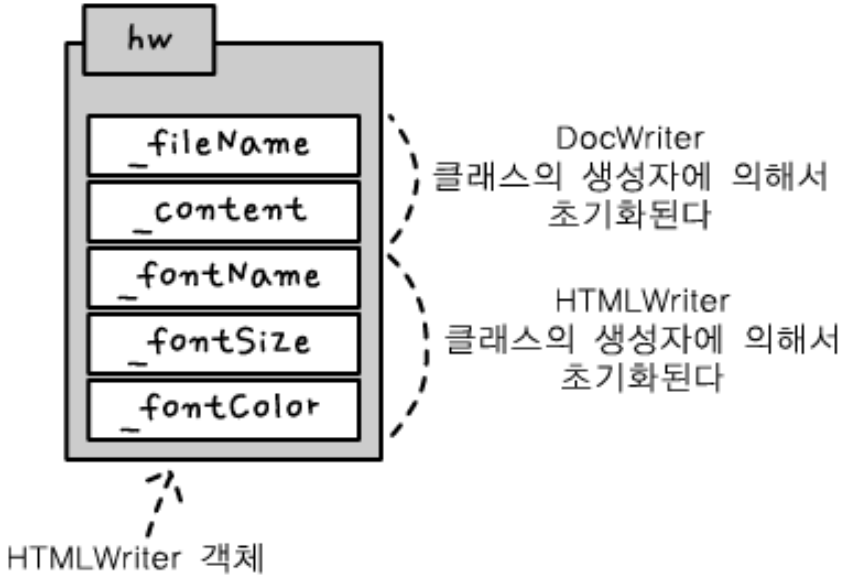


  - 웹 브라우저에서 연 경우



생성자와 소멸자

- 자식 객체가 생성될 때, 자식 클래스의 생성자 뿐만 아니라 부모 클래스의 생성자도 호출된다.



- 부모 클래스에 오버로드된 여러 생성자가 있다면 그 중에서 어떤 생성자가 호출될 지 결정할 수 있다. (다음 페이지 참조)

부모 클래스의 생성자(1)

- 자식 클래스의 생성자에서 부모 클래스의 생성자를 지정하는 예

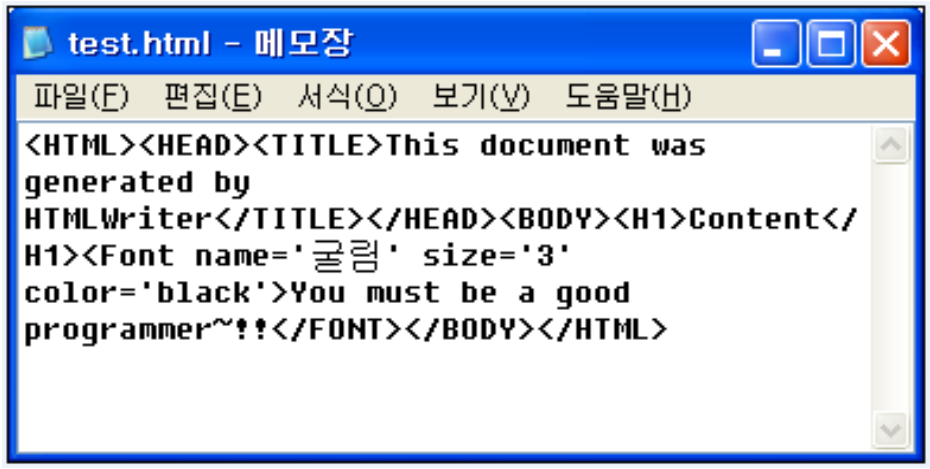
```
// HTMLWriter.h
class HTMLWriter : public DocWriter
{
public:
          HTMLWriter(void);
          HTMLWriter(const string& fileName, const string& content);
          ~HTMLWriter(void);
...
```

```
// HTMLWriter.cpp
HTMLWriter::HTMLWriter(const string& fileName, const string& content)
: DocWriter( fileName, content)          // 부모의 생성자 지정
{
          // 디폴트 폰트를 지정한다.
          _fontName = "굴림";
          _fontSize = 3;
          _fontColor = "black";
}
...
```

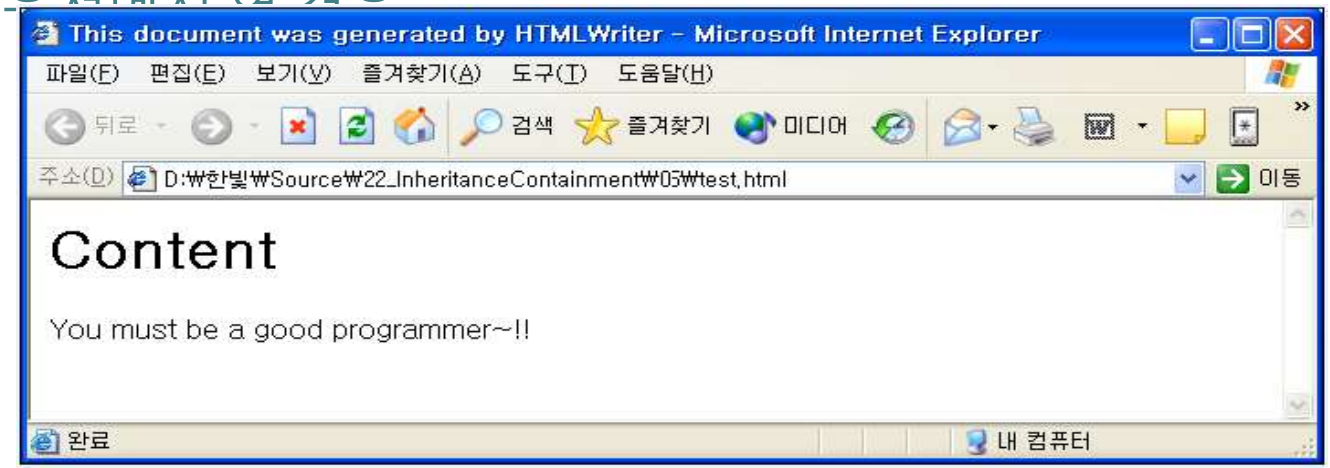
```
// Example.cpp
...
          HTMLWriter hw( "test.html", "You must be a good programmer~!!");
          hw.Write();
...
```

부모 클래스의 생성자(2)

- 결과로 생성된 test.html 파일
  - 텍스트 편집기에서 연 경우



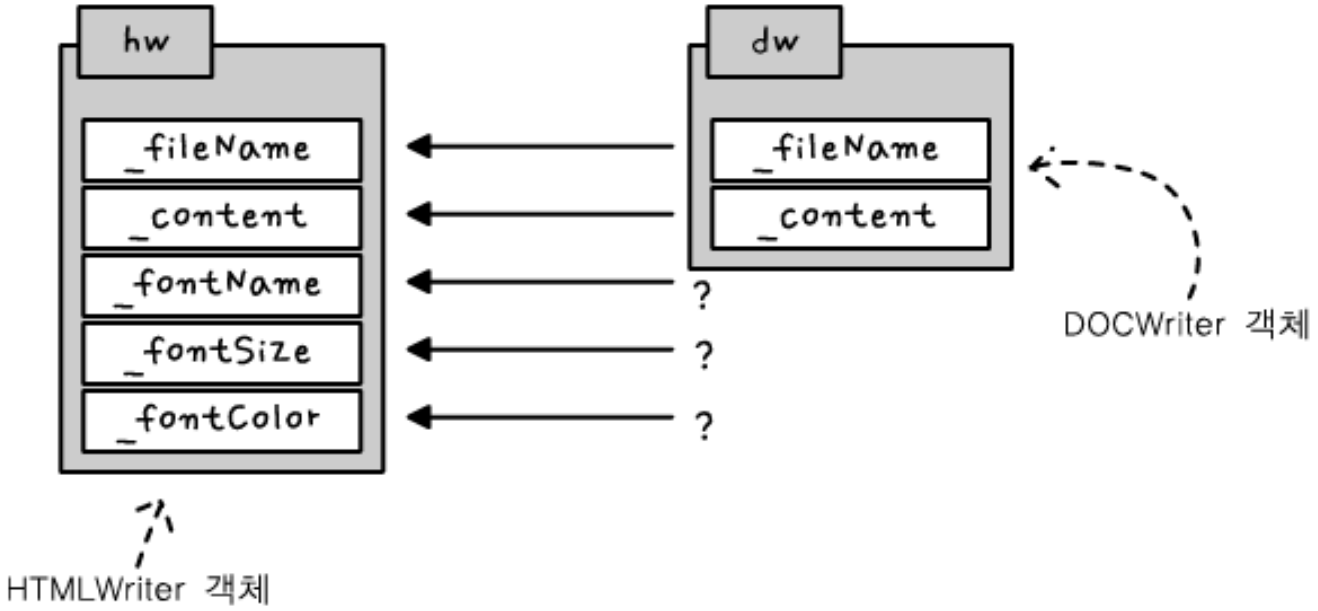
  - 웹 브라우저에서 연 경우



부모와 자식 객체 간의 대입(1)

- 부모 클래스의 객체를 자식 클래스의 객체에 대입할 수 없다.

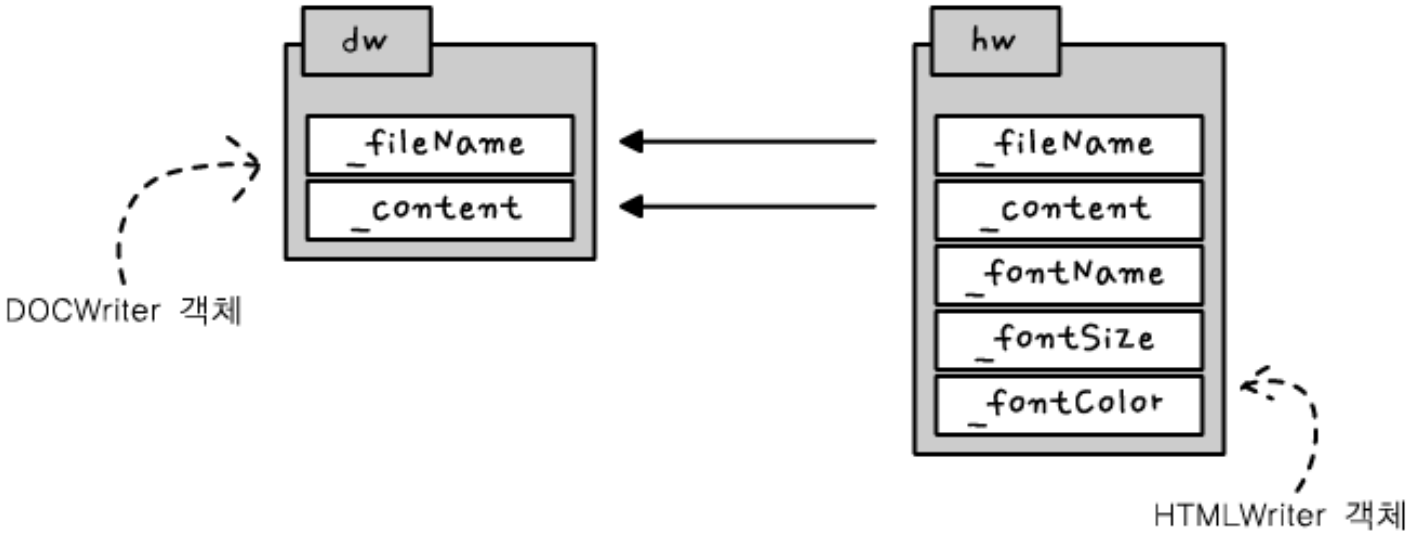
```
HTMLWriter hw;       // 자식 클래스의 객체 생성
DocWriter dw;        // 부모 클래스의 객체 생성

// 부모 클래스의 객체를 자식 클래스의 객체로 대입
hw = dw;             // Error!!
```



부모와 자식 객체 간의 대입(2)

- 자식 클래스의 객체를 부모 클래스의 객체에 대입할 수 있다.

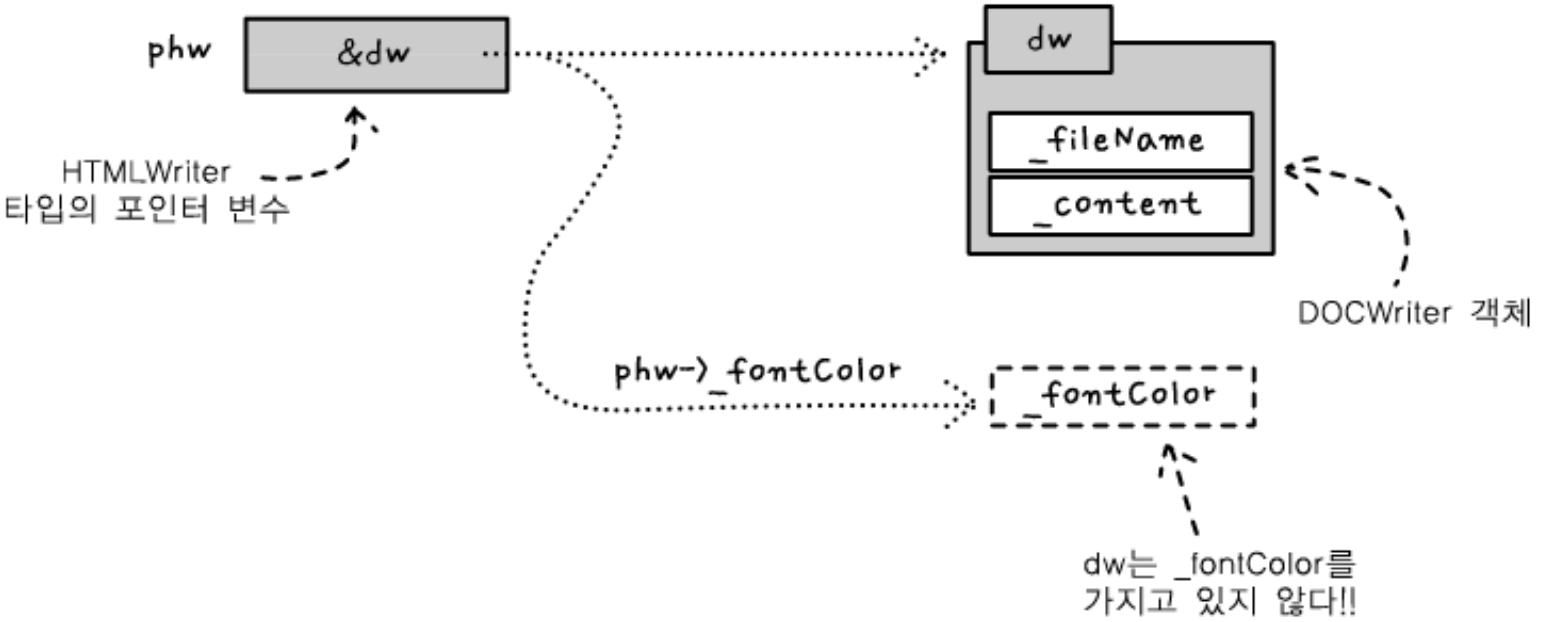
```
HTMLWriter hw;       // 자식 클래스의 객체 생성
DocWriter dw;        // 부모 클래스의 객체 생성

// 자식 클래스의 객체를 부모 클래스의 객체로 대입
dw = hw;             // OK
```



포인터, 레퍼런스의 형변환(1)

- 자식 클래스의 포인터로 부모 객체를 가리킬 수 없다.

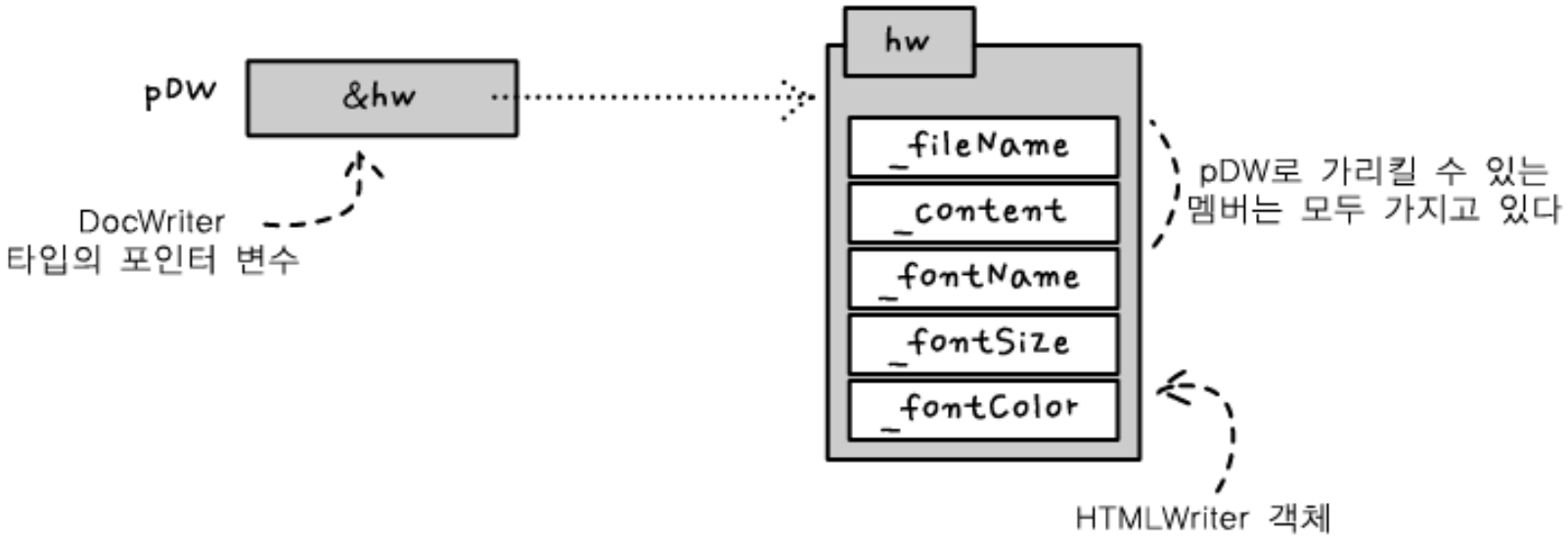
```
DocWriter dw;                    // 부모 클래스의 객체 생성

// 자식 클래스의 포인터 변수로 부모 객체를 가리킨다.
HTMLWriter* phw = &dw;           // Error!!
```



포인터, 레퍼런스의 형변환(2)

- 부모 클래스의 포인터로 자식 객체를 가리킬 수 있다.

```
HTMLWriter hw;                  // 자식 클래스의 객체 생성

// 부모 클래스의 포인터 변수로 자식 객체를 가리킨다.
DocWriter* pDW = &hw;           // OK
```



포인터, 레퍼런스의 형변환(3)

- 레퍼런스의 경우도 포인터와 동일한 규칙을 갖는다.
  - 자식 클래스의 레퍼런스로 부모 객체를 참조할 수 없다.

```
DocWriter dw;                    // 부모 클래스의 객체 생성

// 자식 클래스의 레퍼런스 변수로 부모 객체를 참조한다.
HTMLWriter& hw = dw;             // Error!!
```

  - 부모 클래스의 레퍼런스로 자식 객체를 참조할 수 있다.

```
HTMLWriter hw;                   // 자식 클래스의 객체 생성

// 부모 클래스의 레퍼런스 변수로 자식 객체를 참조한다.
DocWriter& dw = hw;              // OK
```

접근 제어(1)

- 상속과 관련해서 접근 제어 키워드를 실험해 보자.

```
class Parent
{
private:
          int priv;

protected:
          int prot;

public:
          int pub;
};

class Child : public Parent
{
public:
          void AccessParents()
          {
                    int n;
                    // 부모의 멤버에 접근을 시도
                    n = priv; // 실패
                    n = prot; // 성공
                    n = pub;  // 성공
          }
};
```

접근 제어(2)

- 접근 제어 키워드를 다시 정리해보자.

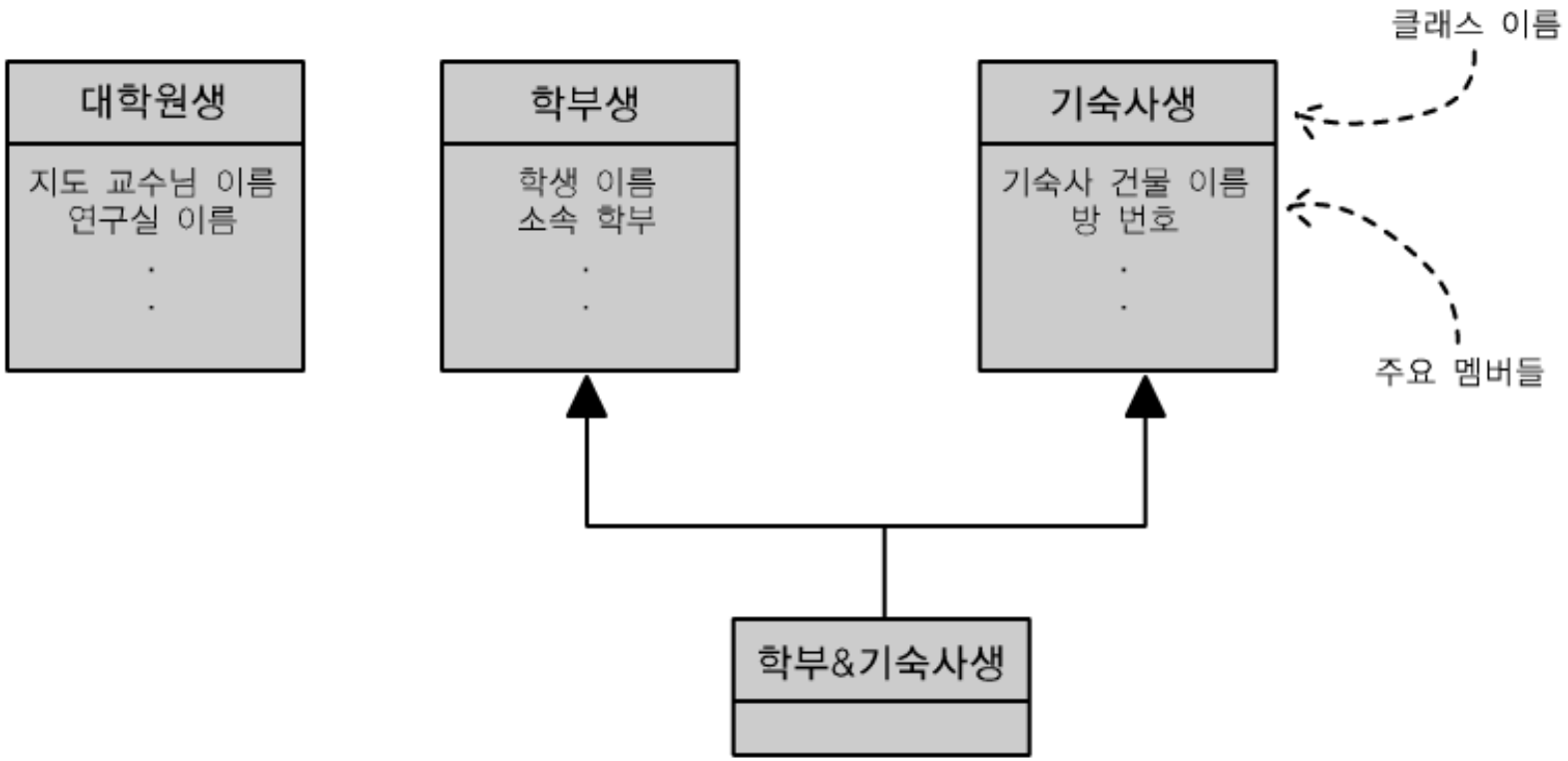
	자신의 멤버 함수에서 접근	자식 클래스의 멤버 함수에서 접근	외부에서 접근
private 멤버	O	X	X
protected 멤버	O	O	X
public 멤버	O	O	O

(O: 접근 가능, X: 접근 불가)

- 접근 제어 가이드라인
  - 외부로부터 숨겨야 하는 멤버는 protected로 지정한다.
  - 그 밖의 경우는 public으로 지정한다.
  - 반드시 자식 클래스에 숨기고 싶다면 private로 지정한다.

다중 상속의 필요성

- 두 개의 부모 클래스를 상속 받을 필요가 있는 경우



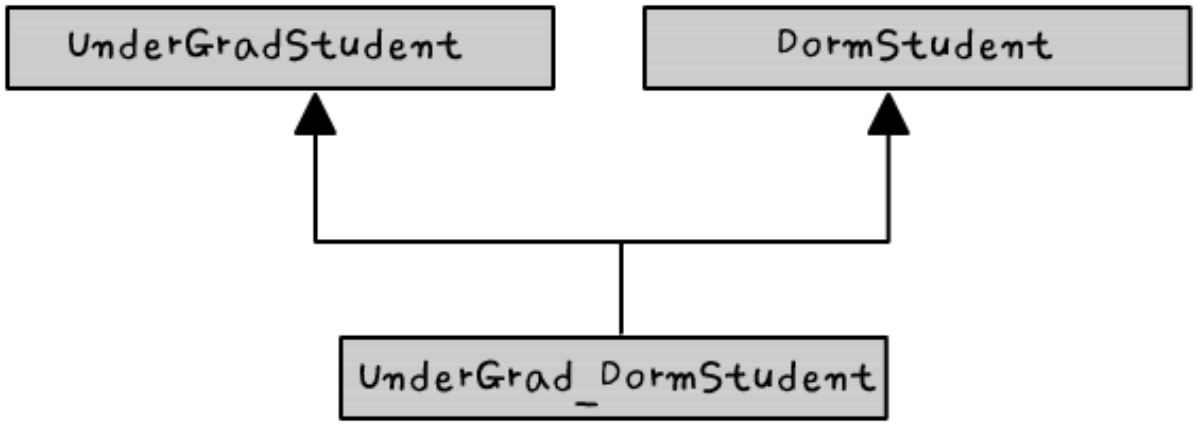
다중 상속의 사용(1)

- 다중 상속을 사용한 클래스 정의의 예

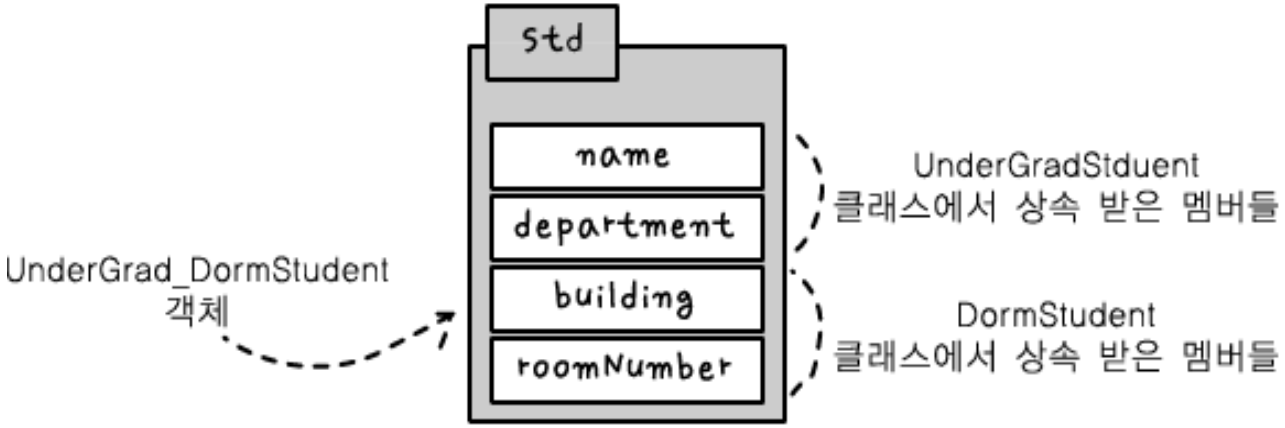
```
// 학부생 클래스
class UnderGradStudent
{
public:
          string name;         // 이름
          string department;   // 학부
};


// 기숙사생 클래스
class DormStudent
{
public:
          string building;     // 기숙사 명
          int roomNumber;      // 방번호
};

// 기숙사생이면서 학부생인 경우
class UnderGrad_DormStudent :
          public UnderGradStudent,
          public DormStudent
{
public:

};
```

다중 상속의 사용(2)

- 상속 계층도



- UnderGrad_DomStudent 클래스의 메모리 구조



다중 상속의 사용(3)

- 두 개의 부모 클래스에 같은 이름의 멤버가 있는 경우 충돌이 발생할 수 있다.

```
class UnderGradStudent
{
        ...
        void Warn();        // 학사 경고
};

class DormStudent
{
        ...
        void Warn();        // 벌점 부여
};

// 중간 생략

UnderGrad_DormStudent std;

std.Warn();                 // Error - 어떤 Warn() 을 말하는 지 알 수 없다.
std.UnderGradStudent::Warn(); // OK
```

포함과 상속의 구분

- Has-a 관계와 Is-a 관계
  - Has-a 관계 : A 가 B 를 가지고(포함하고) 있는 관계
    - 예) 자동차는 타이어를 가지고 있다.
  - Is-a 관계 : A 가 B 인 관계
    - 예) 사과는 과일이다.

- 포함과 상속을 구분해서 사용하기 위한 가이드라인
  - Has-a 관계의 경우에는 포함을 사용한다.
  - Is-a 관계의 경우에는 상속을 사용한다.

Q&A