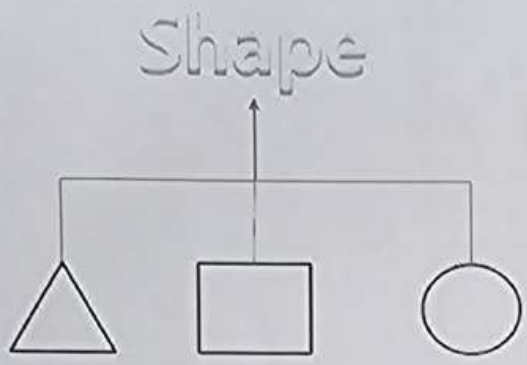
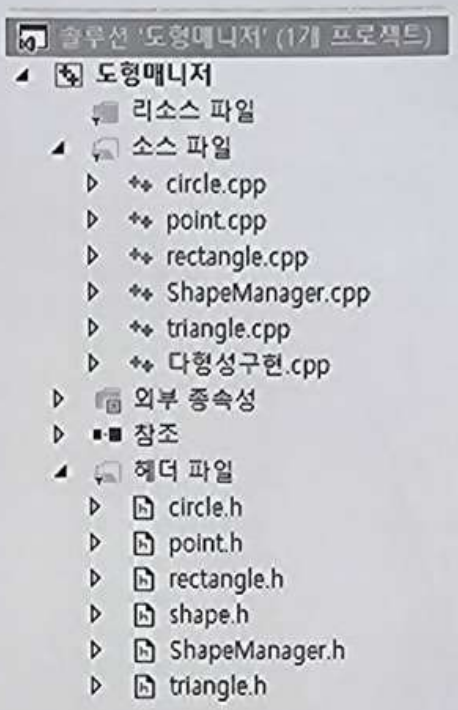
# C++ 상속을 통한 다형성 구현

2023. 11. 19 by Wulong




도형 상속 계층




## 상속과 다형성 실습

1. 새 프로젝트를 만들고 오른쪽 위 솔루션 탐색기와 같이 헤더 파일과 소스 파일을 추가하자.

2. 클래스 상속 계층도는 왼쪽 위 그림과 같다. 실행하는데 필요한 파일은 다음과 같다.

| | |
|---|---|
| Point.h + Point.cpp | (2차원 점) |
| + Shape.h | (Base class) |
| + Circle.h + Circle.cpp | (원) |
| + Rectangle.h + Rectangle.cpp | (사각형) |
| + Triangle.h + Triangle.cpp | (삼각형) |
| + ShapeManager.h + ShapeManager.cpp | (도형 관리 클래스) |
| + 다형성구현.cpp | (main()이 있는 파일) |

3. 코드를 모두 입력한 후 프로그램이 문제없이 실행되는지 확인하자. 실행되지 않는다면 오류 메시지를 살펴보고 문제를 해결해 보자.

- 이 프로젝트를 실행하면 다음과 같이 출력될 것이다.

```
----------------------------------------
관리하는 모든 도형을 그립니다
최대 100개의 도형을 담을 수 있습니다
모두 13개의 도형이 있습니다
----------------------------------------

[0] 삼각형 - (0,0), (0,0), (0,0)
[1] 삼각형 - (0,0), (1,1), (2,2)
[2] 원 - 중심점(1.23,4.56) 반지름 7.89
[3] 사각형 - (0,1), (0,0)
[4] 사각형 - (1,2), (2,3)
[5] 사각형 - (2,3), (4,6)
[6] 사각형 - (3,4), (6,9)
[7] 사각형 - (4,5), (8,12)
[8] 사각형 - (5,6), (10,15)
[9] 사각형 - (6,7), (12,18)
[10] 사각형 - (7,8), (14,21)
[11] 사각형 - (8,9), (16,24)
[12] 사각형 - (9,10), (18,27)

----------------------------------------
그리기를 마칩니다
----------------------------------------
```

4. 더 많은 새 도형을 추가해 보며 프로그램의 동작을 이해해 보자. 다형성이 어떻게 구현되고 있는지 확인하자.

- 새 도형을 추가하고 전체 도형을 출력하여 프로그램의 동작을 살펴보라.

5. 각 객체가 제대로 생성되고 소멸되는지 관찰해 보자.
잘못된 곳이 있다면 소스를 고쳐 한 byte의 메모리도 놓치지 않도록 하라.

- 프로그램 종료시 메모리를 제대로 반환하고 있는지 확인하라.

6. 프로그램이 메뉴에 따라 실행될 수 있도록 고쳐보자.

Menu(예)

- 원하는 도형 추가
  - 삼각형
  - 사각형
  - 원
- 전체 도형을 그리기
- 프로그램 끝내기

- 원하는 도형을 추가하려면 어느 부분에 무엇을 코딩해야 하는가 생각해 보자.

7. 메뉴에 항목을 추가하여 사용자가 선택한 도형을 제거할 수 있게 하자.

- 관리하는 도형 중 원하는 번호를 제거(몇 번째 도형을 제거할까요?)
- 선택한 클래스의 모든 도형을 제거(어떤 도형을 제거할까요?)
- 도형을 제거하기 전과 후의 메모리를 그림으로 그려 알아 보자.

8. 도형관리자가 관리할 수 있는 도형의 갯수가 꽉 찼다고 가정하자.
이런 경우에도 새로운 도형을 추가할 수 있도록 프로그램을 고쳐보자.

- 메모리의 내용을 생각하며 할 일의 순서를 적어보라.

9. 도형관리자가 관리하는 도형을 파일로 저장하고 읽어 오는 메뉴를 추가해 보자.

- 파일의 이름을 입력받아 만든 도형을 읽고 써보라.

- 파일에 저장하기
- 파일에서 읽어오기

10. 프로그램에 새로운 종류의 도형인 Line을 추가해 보자.

- 필요한 파일(Line.h, Line.cpp)을 프로젝트에 추가하라.
- class Line은 시작점과 끝점을 멤버로 갖는다.

11. Line이 잘 추가되었는가?
전체 파일 중에서 Line이 추가됨에 따라 고쳐야 했던 파일은 어떤 것이었나?

- 수정할 필요가 없는 파일이 있었는가?
- 수정할 부분이 있었다면 어느 파일의 어디였는가를 적어보라.
- 이 과정에서 어떤 점을 알 수 있었는가?

12. 마지막으로 이 프로그램의 문제점과 감상 등을 정리해 보자.

```
//-----------------------------------------------------------------------------
// Point.h          Point class declaration - 헤더 파일은 언제나 공개되어야 한다
//                  2차원 좌표 (x,y)를 나타낸다
//
// 2023. 11. 19     by Wulong
//-----------------------------------------------------------------------------
#ifndef _Point                              // 한 번만 include 할 수 있도록 함
#define _Point                              // 조건부 컴파일 - 구글링 할 것

struct Point                                // struct - default public
{
    double x, y;

    Point();                                // default constructor - 스페셜 함수
    Point(double, double);
    Point(const Point&) = default;          // 복사생성자 - 사용자가 만들 필요 없다는 의미
};
#endif


//-----------------------------------------------------------------------------
// Point.cpp        Point class definition - cpp는 컴파일하여 obj 또는 library로 제공한다
//                  2차원 좌표 (x,y)
//                  CPP의 내용은 컴파일하여 object 파일로 제공한다.
//                  (- 다른 사용자에게 구현 소스를 감출 수 있다)
//
// 2023. 11. 19     by Wulong
//-----------------------------------------------------------------------------
#include "point.h"

// 디폴트 생성자
Point::Point()
    : x(0.0), y(0.0)
{
};

// 인자 두 개를 받는 생성자
Point::Point(double a, double b) : x(a), y(b)
{
};


//-----------------------------------------------------------------------------
// Shape.h          class Shape - Virtual Base Class
//                  이 클래스는 추상 클래스(abstract class)이다.
//
// 2023. 11. 19     by Wulong
//-----------------------------------------------------------------------------
#ifndef _Shape                              // _Shape이 앞에서 정의되지 않았다면
#define _Shape                              // _Shape을 정의한다

class Shape
{
public:
```

```
    Shape() { };
    ~Shape() { };

    virtual void draw() const = 0;          // pure virtual function
};
#endif

//-----------------------------------------------------------------------------
// Circle.h         중심점의 좌표와 반지름으로 원을 정의할 수 있다.
//                  (다른 방식으로 원을 정의할 수도 있다)
//
// 2023. 11. 19     by Wulong
//-----------------------------------------------------------------------------
#include "point.h"
#include "shape.h"

// 헤더에서는 위의 두 include 대신에 다음과 같이 전방선언(forward declaration)을 할 수도 있다
// struct Point;
// class Shape;

class Circle : public Shape
{
    Point center;                   // 중심점의 좌표
    double rad;                     // 반지름

public:
    Circle();
    Circle(const Point&, int);
    Circle(const Circle&);          // 깊은 복사를 할 필요가 없다면 프로그래머가 복사생성자를
                                    // 프로그램할 필요는 없다.
                                    // 만약 어떤 이유로 프로그램해야 한다면 메모리를 이해하고
                                    // 코딩해야 한다

    // 위 설명을 다음 줄과 같이 선언하여 대치할 수 있다
    // Circle(const Circle&) = default;
    ~Circle();                      // 소멸자를 프로그램할 필요가 없다는 것을 생각해 보자

    virtual void draw() const override;
};


//-----------------------------------------------------------------------------
// Circle.cpp
//
// 2023. 11. 19     by Wulong
//-----------------------------------------------------------------------------
#include <iostream>
#include "circle.h"

Circle::Circle()
    : center(), rad(0.0)
{
}
```

```
Circle::Circle(const Point& c, int r)
: center(c), rad(r)
{
}

// 복사생성자를 프로그램할 이유가 있다면 멤버변수의 값을 복사해야한다.
// 복사생성자를 프로그램하면서 아무것도 하지 않으면 멤버변수는 복사되지 않는다.

Circle::Circle(const Circle& other)
    : center(other.center), rad(other.rad)
{
}

Circle::~Circle()
{
}

void Circle::draw() const
{
    // 그림을 그리는 대신 문자로 정보를 출력한다
    std::cout << "원 - 중심점(" << center.x << "," << center.y
            << ") 반지름 " << rad << '\n';
};


//-------------------------------------------------------------------------
// Triangle.h                세 점으로 삼각형을 정의할 수 있다
//
// 2023. 11. 19     by Wulong
//-------------------------------------------------------------------------
#include "point.h"                              // Point를 Triangle에서 사용
#include "shape.h"                              // Shape을 상속 받음

class Triangle : public Shape
{
    Point p1, p2, p3;

public:
    Triangle();
    Triangle(const Point&, const Point&, const Point&);
    Triangle(const Triangle&);
    ~Triangle();

    virtual void draw() const override;       // virtual function을 overriding
};



//-------------------------------------------------------------------------
// Triangle.cpp
//
// 2023. 11. 19     by Wulong
//-------------------------------------------------------------------------
#include <iostream>
#include "triangle.h"
```

```
Triangle::Triangle()
    : p1(), p2(), p3()
{
};

Triangle::Triangle(const Point& a, const Point& b, const Point& c)
    : p1(a), p2(b), p3(c)
{
};

Triangle::Triangle(const Triangle& other)
    : p1(other.p1), p2(other.p2), p3(other.p3)
{
}

Triangle::~Triangle()
{
}

void Triangle::draw() const
{
    std::cout << "삼각형 - (" << p1.x << "," << p1.y << "), ("
              << p2.x << "," << p2.y << "), ("
              << p3.x << "," << p3.y << ")" << '\n';
};


//-----------------------------------------------------------------------------
// Rectangle.h                대각선에 있는 점 2개로 사각형을 정의할 수 있다
//
// 2023. 11. 19      by Wulong
//-----------------------------------------------------------------------------
#include "point.h"
#include "shape.h"

class Rectangle : public Shape
{
    Point p1, p2;

public:
    Rectangle();
    Rectangle(const Point&, const Point&);
    Rectangle(const Rectangle&);
    ~Rectangle();

    virtual void draw() const override;
};


//-----------------------------------------------------------------------------
// Rectangle.cpp
//
// 2023. 11. 19      by Wulong
//-----------------------------------------------------------------------------
```

```
#include <iostream>
#include "rectangle.h"

Rectangle::Rectangle()
    : p1(), p2()
{
};

Rectangle::Rectangle(const Point& a, const Point& b)
    : p1(a), p2(b)
{
};

Rectangle::Rectangle(const Rectangle& other)
    : p1(other.p1), p2(other.p2)
{
}

Rectangle::~Rectangle()
{
}

void Rectangle::draw() const
{
    std::cout << "사각형 - (" << p1.x << "," << p1.y << "), ("
              << p2.x << "," << p2.y << ")" << '\n';
};



//-------------------------------------------------------------------------
// ShapeManager.h          여러 개의 도형을 관리하는 클래스
//
//-----------------------------------------------------
// Shape* (부모 클래스의 *)로 전체 도형을 관리하는 것이 핵심
//-----------------------------------------------------
// 2023. 11. 19      by Wulong
//-------------------------------------------------------------------------
#include "shape.h"

class ShapeManager {
    int nShape;                         // 현재 관리 중인 도형의 갯수
    int capacity;                       // 확보한 메모리에 담을 수 있는 도형의 최대 개수
    Shape** shapes;                     // 도형의 포인터들을 담을 수 있는 메모리

public:
    explicit ShapeManager(int n);       // 담을 수 있는 도형의 갯수를 생성자에 전달
                                        // explicit 사용법을 알아보자
    ~ShapeManager();
    ShapeManager(const ShapeManager&) = default;


    void insert(Shape*);                // 도형을 추가하는 함수
    void draw() const;                  // 전체 도형을 그리는 함수
};
```

```
//--------------------------------------------------------------------------
// ShapeManager.cpp       여러 개의 도형을 관리하는 클래스
//
// 2023. 11. 19     by Wulong
//--------------------------------------------------------------------------
#include <iostream>
#include "shapeManager.h"
using namespace std;

//-------------------------------
ShapeManager::ShapeManager(int n)
//-------------------------------
{
    nShape = 0;                              // 처음 만들어질 때는 관리하는 도형 갯수가 0임
    capacity = n;                            // 최대 n개의 도형을 담을 수 있음
    shapes = new Shape*[capacity];
}

//---------------------------
ShapeManager::~ShapeManager()
//---------------------------
{
    // 모든 객체가 정확하게 삭제되는지 반드시 확인하여야 한다.
    delete[] shapes;                         // 도형관리자가 관리하는 도형의 소멸자를 호출함
};

//-------------------------------
void ShapeManager::insert(Shape* a)
//-------------------------------
{
    shapes[nShape] = a;
    nShape++;
};

//---------------------------
void ShapeManager::draw() const
//---------------------------
{
    cout << "-----------------------------------------" << '\n';
    cout << "관리하는 모든 도형을 그립니다" << '\n';
    cout << "최대 " << capacity << "개의 도형을 담을 수 있습니다" << '\n';
    cout << "모두 " << nShape << "개의 도형이 있습니다" << '\n';
    cout << "-----------------------------------------" << '\n' << '\n';

    for (int i = 0; i < nShape; ++i) {
        cout << "[" << i << "] ";
        shapes[i]->draw();                   // 다형성이 구현된다.
    }
    cout << '\n';

    cout << "-----------------------------------------" << '\n';
    cout << "그리기를 마칩니다" << '\n';
    cout << "-----------------------------------------" << '\n' << '\n';
};
```

```
//---------------------------------------------------------------------------
// 다형성구현.cpp
//
//  찾아보고 알아볼 것
//
//          - 다형성(Ploymorphism)이란 무엇인가?
//          - C++ 언어에서는 어떻게 다형성을 구현하는가?
//
// 2023. 11. 19     by Wulong
//---------------------------------------------------------------------------
#include "point.h"
#include "triangle.h"
#include "circle.h"
#include "rectangle.h"
#include "ShapeManager.h"

//--------
int main( )
//--------
{
    // 이 프로그램은 프로젝트를 확인하기 위한 것이다.
    // 실습 순서대로 문제를 해결해 보자.

    // 1. 관리 클래스를 만듦
    ShapeManager sm(100);                   // 최대 100개의 도형을 관리함

    // 2. 도형들을 관리 클래스에 삽입함
    sm.insert(new Triangle());
    sm.insert(new Triangle(Point(0, 0), Point(1, 1), Point(2, 2)));
    sm.insert(new Circle(Point(1.23, 4.56), 7.89));

    for (int i = 0; i < 10; ++i)
        sm.insert(new Rectangle(Point(i, i + 1), Point(i * 2, i * 3)));

    // 3. 관리하고 있는 모든 도형을 그림
    sm.draw();

    // 이 프로그램에서 잘못된 점을 찾을 수 있는가?
    // 잘못된 점을 어떻게 찾을 수 있는가?
}
```