#include <iostream>

using namespace std;

class Shape { //기본클래스

protected: //멤버 선언한 클래스 및 파생클래스에서 접근 가능

int x, y;

Shape(int x, int y) { //생성자

this->x = x; this->y = y; //도형의 위치 정보 저장

}

virtual ~Shape() {} //소멸자

public:

virtual void draw() = 0; //순수 가상 함수 draw()작성

virtual double getArea() = 0; //순수 가상 함수 getArea()작성

};

class Rectangle :public Shape { //파생클래스//Shap 상속

int width, height;

public:

//도형의 위치.가로/세로 길이 등을 매개변수로 전달받아 저장

Rectangle(int x, int y, int width, int height) :Shape(x, y) { //Shap생성자 명시적 호출//위치 정보 전달

this->width = width; this->height = height;

}

//사각형의 위치 정보 출력

virtual void draw() {

cout << "Rectangle drawn at : (" << x << "," << y << ")" << endl;

}

//사각형의 넓이 출력

virtual double getArea() {

return this->width \* this->height;

}

//소멸자

virtual ~Rectangle() {}

};

class Circle :public Shape { //파생클래스//Shap 상속

int radius;

public:

//도형의 위치.반지름 등을 매개변수로 전달받아 저장

Circle(int x, int y, int radius) :Shape(x, y) {//Shap생성자 명시적 호출//위치 정보 전달

this->radius = radius;

}

//원의 위치 정보 출력

virtual void draw() {

cout << "Circle drawn at : (" << x << "," << y << ")" << endl;

}

//원의 넓이 출력

virtual double getArea() {

return this->radius \* this->radius \* 3.14;

}

//소멸자

virtual ~Circle() {}

};

int main() {

Shape\* pS;

Rectangle r(10, 20, 100, 100); // (10, 20)위치에 가로 100, 세로 100인 사각형 객체 생성

pS = &r;

//파생 클래스 객체인 r의 주소를 일반 클래스 포인터 변수에 저장(업캐스팅)

//업캐스팅을 통해서 pS는 매번 다른 파생 클래스의 객체 주소를 가르킬 수 있다

pS->draw(); //동적 바인딩에 의해서 pS의 내용 즉, 담겨져 있는 객체의 함수가 동적으로 호출

cout << "사각형 면적 = " << pS->getArea() << endl << endl;

Circle c(120, 150, 40);

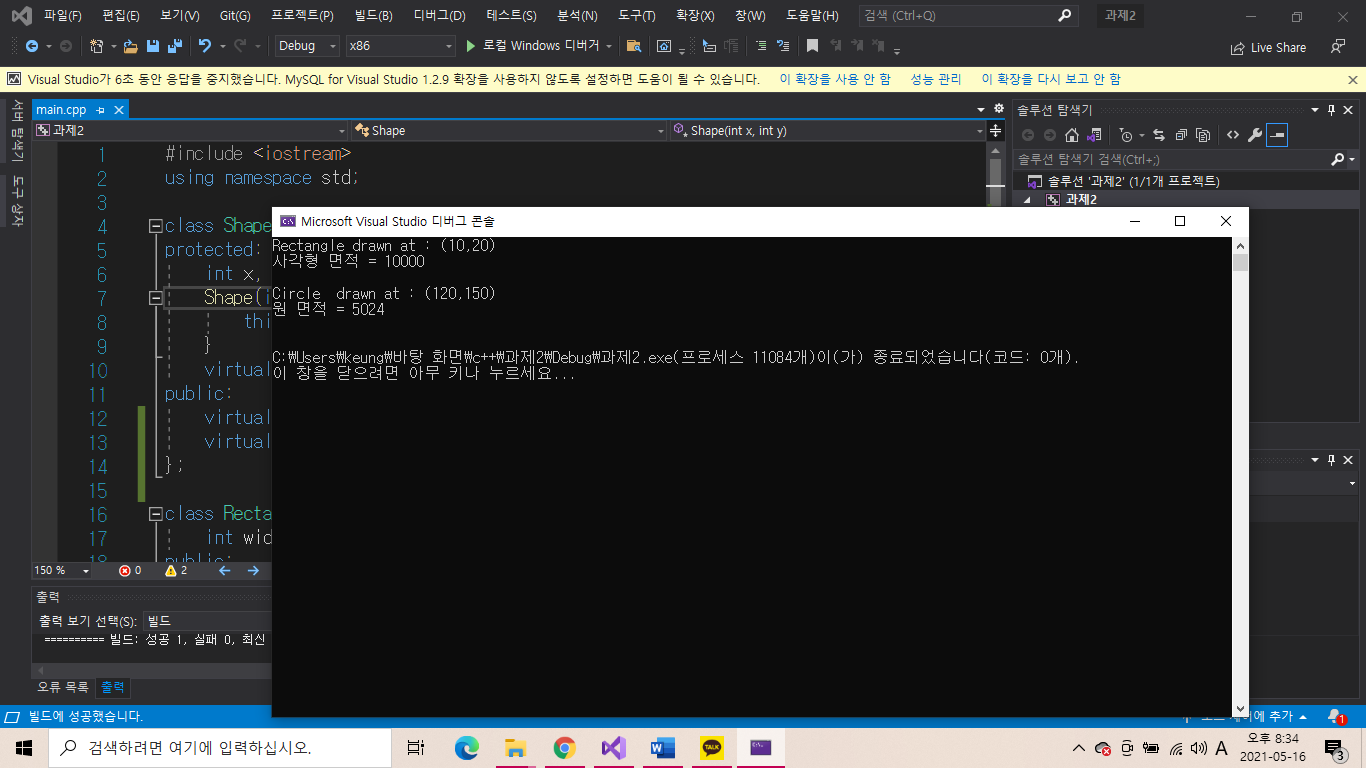
pS = &c;

pS->draw();

cout << "원 면적 = " << pS->getArea() << endl << endl;

return 0;

}



동적 바인딩: pS->draw()를 실행 했을 때, pS는 일반클래스의 포인터 변수이지만 현재 가르키고 있는 파생클래스의 draw() 함수를 동적으로 호출한다.

업캐스팅의 필요성: 업캐스팅을 통해서 기본클래스의 포인터로 파생 클래스의 객체를 가르킬 수 있다. 만약 오버라이딩된 멤버함수가 있다면 이 기본 클래스의 포인터를 이용해 파생 클래스의 함수를 가르킬 수가 있다. 따라서 굳이 각 클래스의 객체를 생성하고 그 객체 이름으로 멤버함수를 호출할 필요가 없다.