

```

#include <iostream>
using namespace std;

class Shape {
protected:
    int x, y;
public:
    Shape() { } //생성자 선언
    virtual void draw() = 0; //순수 가상 함수 선언
    virtual double getArea() = 0; //순수 가상 함수 선언
    virtual ~Shape(); //가상 소멸자
};
Shape::~Shape() { } //가상 소멸자 구현

class Rectangle : public Shape {
public:
    int width, height;
    Rectangle(int x, int y, int width, int height) { this->x = x, this->y = y, this->width
= width, this->height = height; } //생성자 선언
    }
    virtual void draw() { //위치 정보 출력
        cout << "Rectangle drawn at :(" << x << ", " << y << ")" << endl;
    }
    virtual double getArea() { //면적 계산
        int area = (width * height);
        return area;
    }
    virtual ~Rectangle(); //가상 소멸자
};
Rectangle::~Rectangle() { } //가상 소멸자 구현

class Circle : public Shape {
public:
    int radius;
    Circle(int x, int y, int radius) { this->x = x, this->y = y, this-
>radius=radius; } //생성자 선언
    }
    virtual void draw() { //위치 정보 출력
        cout << "Circle drawn at :(" << x << ", " << y << ")" << endl;
    }
    virtual double getArea() { //면적 계산
        double area = (3.14 * radius*radius);
        return area;
    }
    virtual ~Circle(); //가상 소멸자
};
Circle::~Circle() { } //가상 소멸자 구현

int main() {
    Shape* pS;
    Rectangle r(10, 20, 100, 100); // (10, 20)위치에 가로 100, 세로 100인 사각형 객체
    생성
    pS = &r;
    pS->draw();
}

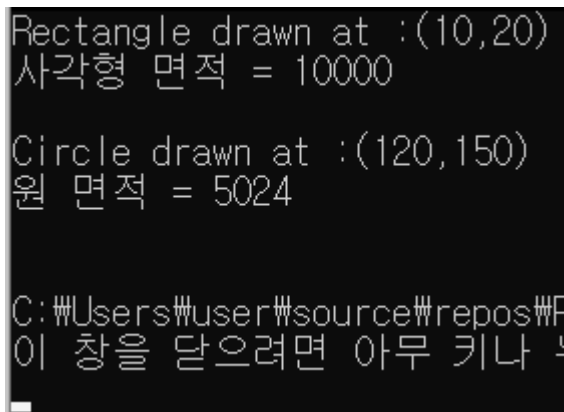
```

```

    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;
}

```



```

Rectangle drawn at :(10,20)
사각형 면적 = 10000

Circle drawn at :(120,150)
원 면적 = 5024

C:\Users\User\source\repos\F
이 창을 닫으려면 아무 키나 누르십시오

```

과제를 수행하면서 추상 클래스에서 순수 가상 함수를 이용하여 오버 라이딩을 적용할 수 있게 된 것 같다

동적 바인딩: (업캐스팅 후) 오버라이딩한 파생 클래스의 함수를 찾아 실행, 런타임 시점에 해당 메소드를 구현하고 있는 실제 객체 타입을 기준으로 실행될 함수를 호출한다. 소멸자 호출 시 동적 바인딩이 발생한다

업캐스팅의 필요성: 업캐스팅을 사용하면 파생 클래스들의 공통적인 부분들을 간단하게 만들 수 있다.(코드의 간결성)