

C++ Programming

10th Study: Object-Oriented Programming (6/8)

- Polymorphism
- Pure virtual method



C++ Korea 옥찬호 (utilForever@gmail.com)

Polymorphism

The provision of a single interface to entities of different types

Polymorphism

- 다형성이란 표준적인 멤버 변수 / 함수들을 정의해 두고, 그것을 따르는 객체들이라면 그 중 어느 객체를 이용하든 정상적으로 이용할 수 있다는 개념
- 클래스 정의는 객체와 이를 이용하는 코드 간 계약과 같아서, 예를 들어 사각형 객체는 그 정의에 의해 Rect 클래스의 멤버 변수와 함수를 지원해야만 함
- 이러한 개념은 슈퍼클래스에서도 마찬가지로 적용됨
 - 모든 사각형은 도형이므로, 모든 Rect 객체는 Shape 클래스의 멤버 변수와 함수를 지원해야만 함

Polymorphism

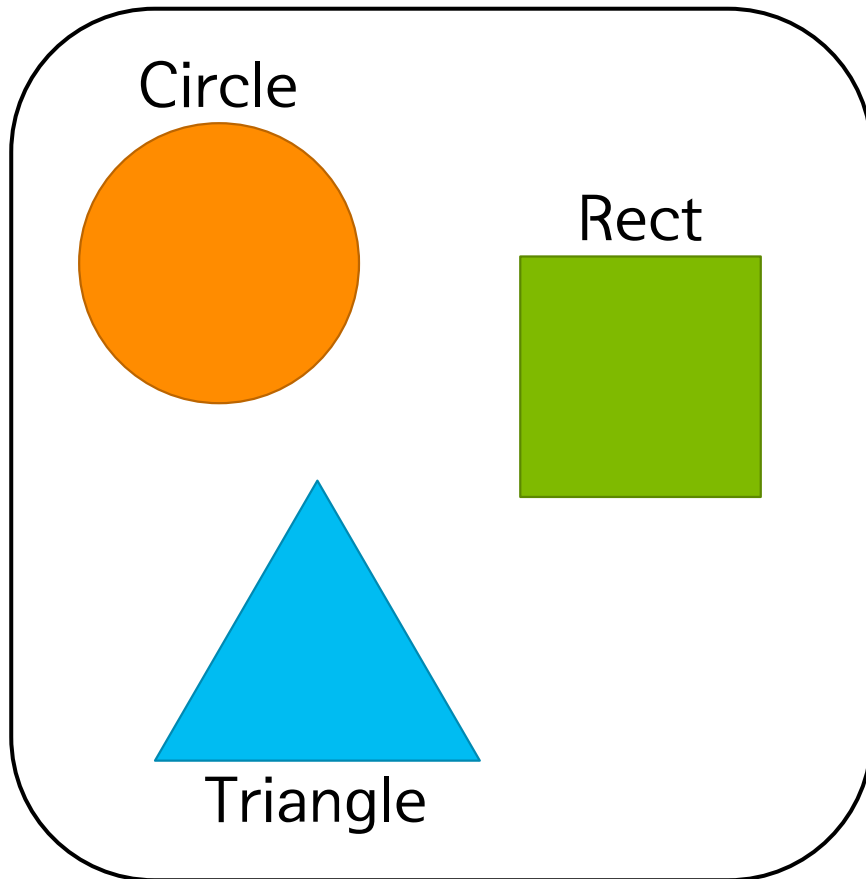
- 예를 들어, 도형을 그리는 프로그램에서 사용자가 모든 도형들의 넓이를 구한다고 가정
 - 각 도형의 넓이를 구하는 방법은 다름
 - 원 : 원주율(π) \times (반지름)²
 - 사각형 : 폭(가로) \times 높이(세로)
 - 삼각형 : (밑변 \times 높이) \div 2
 - 모든 도형이 Shape 클래스의 멤버라면
getArea() 함수를 통해 넓이를 구할 수 있음

Polymorphism

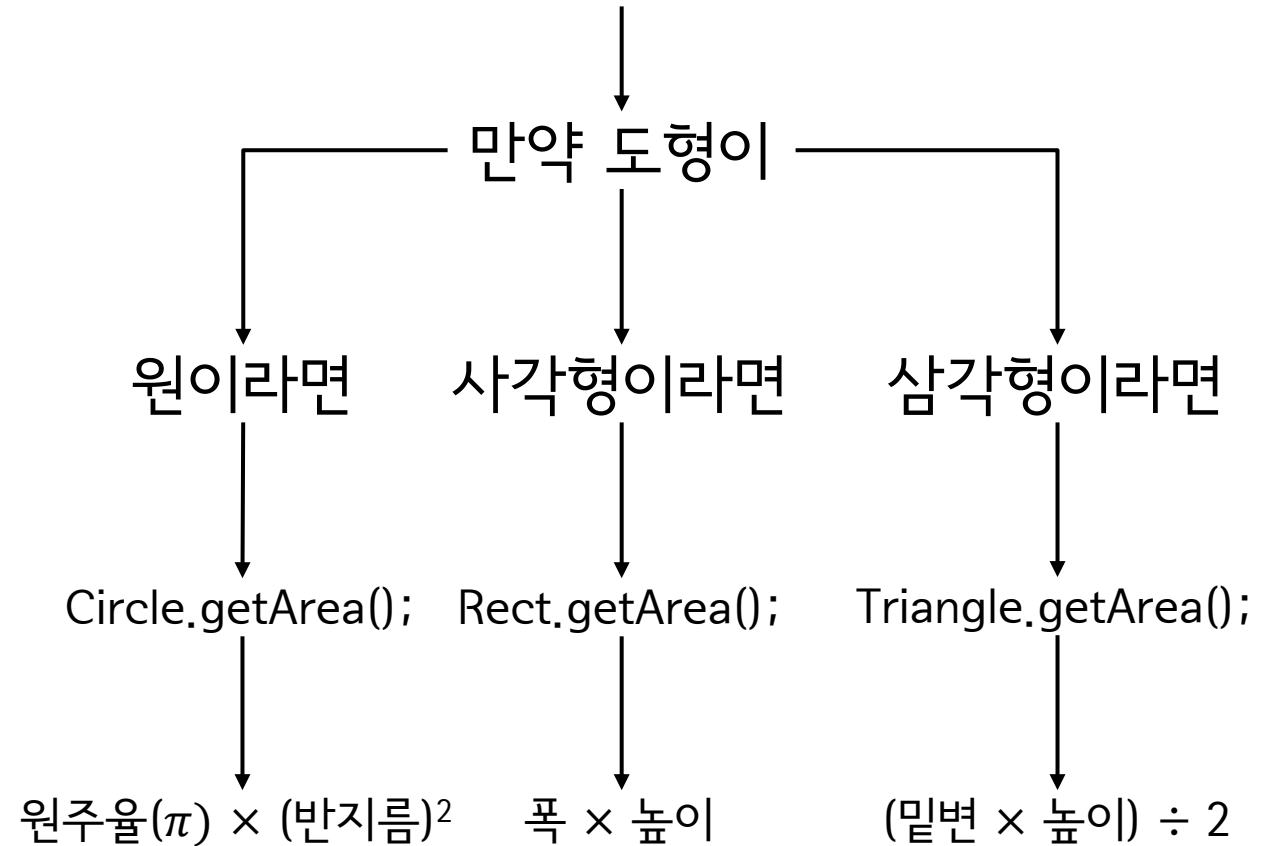
- 모든 도형이 Shape 클래스의 멤버라면
getArea() 함수를 통해 넓이를 구할 수 있음
- 도형마다 넓이를 구하는 방법이 다르기 때문에
메서드 오버라이딩을 적용해서 각자 다른 방식으로 넓이를 구함
- 중요한 점은, 각 도형의 넓이를 구하는 함수를 호출했을 뿐
각 도형이 어떤 방식으로 넓이를 구하는지는 전혀 알 필요가 없음
- 각 도형은 자신이 어떻게 넓이를 구해야 하는지 알고 있음

Polymorphism

Shape



Shape.getArea();



Polymorphism

```
class Shape
{
protected:
    std::string name;
    int width;
    int height;

public:
    Shape(std::string _name, int _width, int _height)
        : name(std::move(_name)), width(_width), height(_height) { }
    void printArea() { std::cout << "The area of " << name.c_str()
        << " is " << getArea() << std::endl; }
    virtual double getArea() { return 0; };
};
```

Polymorphism

```
class Circle : public Shape
{
protected:
    const double PI = 3.14;
public:
    Circle(std::string _name, int _width, int _height)
        : Shape(std::move(_name), _width, _height) { }
    virtual double getArea() { return PI * std::pow(width * 0.5, 2); };
};
```

```
class Rect : public Shape
{
public:
    Rect(std::string _name, int _width, int _height)
        : Shape(std::move(_name), _width, _height) { }
    virtual double getArea() { return width * height; };
};
```


Polymorphism

```
class Triangle : public Shape
{
public:
    Triangle(std::string _name, int _width, int _height)
        : Shape(std::move(_name), _width, _height) { }
    virtual double getArea() { return width * height * 0.5; };
};
```

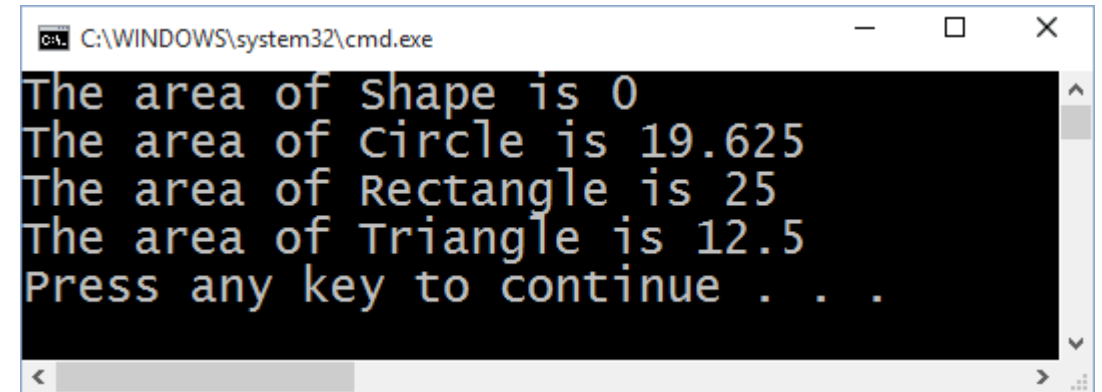
Polymorphism

```
int main()
{
    Shape* s = new Shape("Shape", 5, 5);
    s->printArea();

    s = new Circle("Circle", 5, 5);
    s->printArea();

    s = new Rect("Rectangle", 5, 5);
    s->printArea();

    s = new Triangle("Triangle", 5, 5);
    s->printArea();
}
```



The screenshot shows a Windows command prompt window titled "C:\WINDOWS\system32\cmd.exe". The output of the program is displayed as follows:

```
The area of Shape is 0
The area of Circle is 19.625
The area of Rectangle is 25
The area of Triangle is 12.5
Press any key to continue . . .
```

Pure virtual method

A virtual function that is required to be implemented by a derived class if that class is not abstract

Pure virtual method

- 순수 가상 함수란 클래스 정의 때 선언만 되고 구현부 정의는 되지 않는 멤버 함수를 말함
- 멤버 함수를 순수 가상 함수로 선언하면 컴파일러가 그 멤버 함수가 선언된 클래스에서 멤버 함수의 구현부를 찾지 않음
- 때문에 순수 가상 함수를 가진 클래스는 인스턴스화할 수 없음
→ 추상 클래스(Abstract Class)라고 부름
- 클래스가 순수 가상 함수를 하나라도 가지고 있으면 컴파일러에 의해 해당 클래스의 객체 생성이 금지됨

Pure virtual method

- 순수 가상 함수로 선언하기 위해서는 멤버 함수 뒤에 '= 0'을 붙이는 특별한 문법을 사용해야 함

`virtual double getArea() { return 0; };` ➡ `virtual double getArea() = 0;`

- 순수 가상 함수로 선언하면 .cpp 파일에서 멤버 함수 정의를 구현할 필요도 없고, 구현하더라도 컴파일 오류가 발생함

```
Shape* s = new Shape("Shape", 5, 5);
```

```
error C2259: 'Shape': cannot instantiate abstract class
: note: due to following members:
: note: 'double Shape::getArea(void)': is abstract
: note: see declaration of 'Shape::getArea'
```

- 하지만 서브 클래스를 인스턴스화하면 아무런 문제가 없음

```
Shape* s = new Circle("Circle", 5, 5);
```

Pure virtual method

- 가상 함수는 재정의해도 되는 함수,
서브클래스에서 반드시 재정의해야 하는 함수는 아님

```
class Super
{
public:
    virtual void f() { }
};
```

```
class Sub : public Super
{
};
```

```
int main()
{
    Sub sub;
    sub.f();
}
```

- 반면, 순수 가상 함수는 서브클래스에서 반드시 재정의해야 하는 함수

```
class Super
{
public:
    virtual void f() = 0;
};
```

```
class Sub : public Super
{
    // 구현하지 않을 경우
    // 오류 발생
    virtual void f() { }
};
```

```
int main()
{
    Sub sub;
    sub.f();
}
```