개요 | 제출 | 편집 | 코딩 결과

실습 099 - decorator (static polymorphism)

제출 마감일: 2023-06-09 23:59
업로드 가능한 파일 수: 3
제출 방식: 개인

목적

이 실습은 Mixin 상속을 활용하는 연습을 합니다.

설명

아이폰 앱을 만든다고 가정해 봅시다.

UIViewController 클래스를 상속한 ViewController 를 정의해서 원하는 사용자 입력을 처리할 수 있습니다.

애플에서 제공하는 프레임워크에 수정 없이도 자신만의 앱을 추가할 수 있습니다.

상속 이외에도 기존 코드를 수정하지 않고 새로운 기능을 추가할 수 있는 방법이 있을까요?

도형을 표현하는 Shape 클래스가 있습니다.

```
enum Color {RED, BLUE, GREEN};
class Shape {
public:
    virtual std::string toString() const = 0;
    virtual ~Shape() noexcept {};
};
```

이 클래스를 상속 받아 색상이 있는 도형 클래스와 투명한 도형 클래스를 추가하였습니다.

색상과 투명한 도형이 동시에 필요하여 ColoredTransparentShape 클래스를 하나 더 추가하였습니다.

만약, 위치 기능이 추가되면

색상, 투명, 위치, 색상+투명, 색상+위치, 투명+위치, 색상+투명+위치 클래스가 추가됩니다.

위 방법의 단점은

Shape 클래스의 파생 클래스로 Square 와 Circle 이 추가되면

클래스의 수가 폭발적으로 늘어나게 된다는 점입니다.

단순히 상속으로는 새로운 기능을 효율적으로 추가하는데 문제가 발생할 수 있다는 사실을 알게 되었습니다.

상속을 적용하기 어려울 때,

객체 지향에서는 컴포지션 (composition) 을 활용하면 좋은 해결책이 나올 때가 있습니다.

템플릿과 상속을 함께 사용해서 문제를 해결해 봅시다.

C++ 에서는 클래스를 정의할 때

자기 자신을 템플릿 인자로 가질 수 있다고 설명하였습니다. (Mixin 상속)

```
//ColoredShape.h
template <typename T>
class ColoredShape : public T {
public:
   ColoredShape() = default;
   ~ColoredShape() noexcept {}
   // your code here
private:
   Color color=RED;
};
```

```
//TransparentShape.h
template <typename T>
class TransparentShape : public T {
public:
   TransparentShape()=default;
   ~TransparentShape() noexcept {}
   // your code here
private:
   int transparent=100;
};
```

다음 생성자 형태를 참고하여 문제를 해결하세요.

```
template<typename...Args>
ColoredShape(Color color, Args...args) : color(color), T(std::forward<Args>(args)...){}
```

문제

주어진 메인 함수를 실행하여 정해진 문자열이 출력하도록

ColoredShape와 TransparentShape 클래스를 정의하시오.

```
//DecoratorTest.cpp

int main() {
    ColoredShape<TransparentShape<Square>> square{RED};
    square.setWidth(10);
    square.setHeight(10);
    std::cout << square.toString() << std::endl;

    TransparentShape<ColoredShape<Circle>> circle{70};
    circle.setRadius(5);
    std::cout << circle.toString() << std::endl;

    TransparentShape<ColoredShape<Circle>> red_circle{100, RED};
    red_circle.setRadius(7);
    std::cout << red_circle.toString() << std::endl;

    return 0;
}
```

<참고>

```
//Square.h
class Square : public Shape {
public:
    Square() = default;
    Square(int width, int height): width(width), height(height) {}
    ~Square() noexcept {};
    std::string toString() const override {
        std::ostringstream oss;
        oss << "Square has the width: " << width << ", height: " << height;
        return oss.str();
    };
    void setWidth(int width) { this->width = width; }
    void setHeight(int height) { this->height = height; }
private:
    int width=0, height=0;
};
```

```
//Circle.h
class Circle : public Shape {
public:
    Circle() = default;
    Circle(int radius): radius(radius) {}
    ~Circle() noexcept {};
    std::string toString() const override {
        std::ostringstream oss;
        oss << "Circle has the radius: " << radius;
        return oss.str();
    };
    void setRadius(int radius) { this->radius = radius; }
private:
    int radius = 0;
};
```

입력

없음

출력

Square has the width: 10, height: 10, transparent: 100, color: 0
Circle has the radius: 5, color: 0, transparent: 70
Circle has the radius: 7, color: 0, transparent: 100

제출파일

ColoredShape.h

TransparentShape.h

103.csv