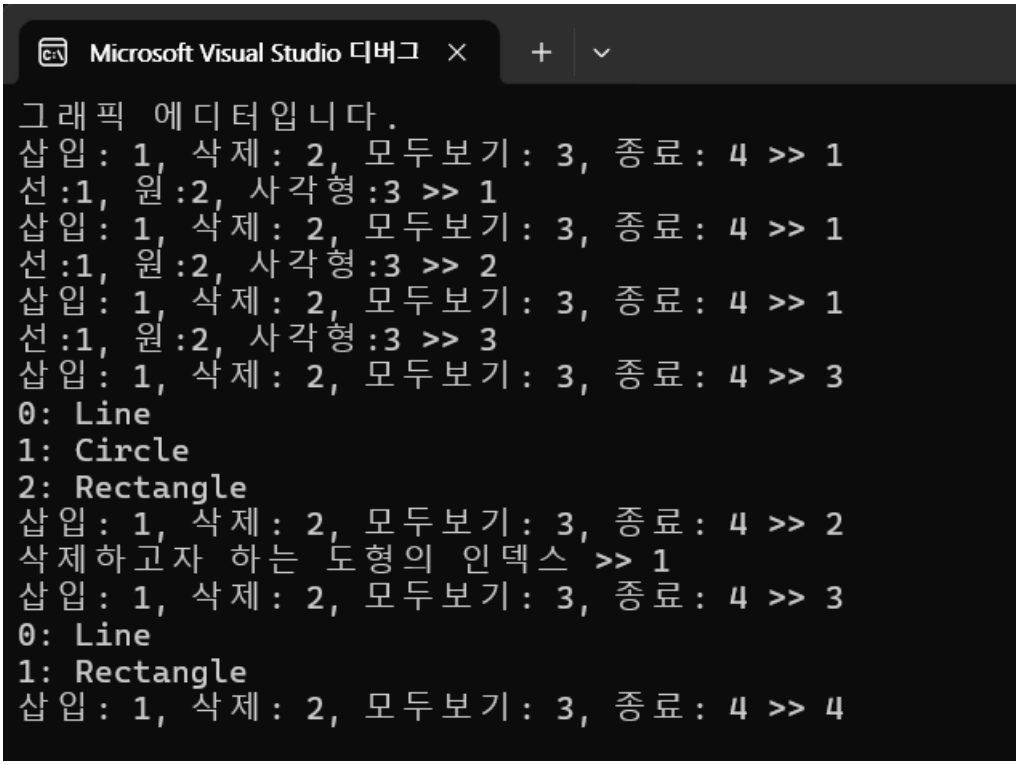
객체지향프로그래밍 과제#6

소프트웨어전공 202284012 김주원 / 소프트웨어전공 202304012 김도연

1. 소스 수행 결과 화면



2. 문제 정의

이 문제는 사용자가 선, 원, 사각형 도형을 삽입하고 삭제하며 삽입된 도형들을 목록에서 확인할 수 있는 그래픽 에디터 프로그램을 구현하는 문제이다. 사용자는 메뉴를 통해 도형을 삽입하거나 삭제할 수 있고 삽입된 도형은 순차적으로 저장되어 인덱스를 통해 선택 및 삭제가 가능하다. 또한 프로그램은 삽입된 도형들을 모두 보기 기능을 통해 현재 저장된 도형들을 출력하며 사용자는 도형의 종류에 상관없이 언제든지 목록을 확인할 수 있다. 도형을 삭제할 때는 지정된 인덱스를 입력하여 해당 도형을 제거할 수 있으며 종료 시에는 프로그램이 동적으로 할당된 메모리를 해제하여 메모리 누수를 방지한다. 이 프로그램은 객체 지향적으로 설계되어 Shape라는 추상 클래스를 통해 공통된 인터페이스를 제공하고 Line, Circle, Rect 클래스가 이를 상속하여 각 도형에 맞는 구현을 제공한다. 최종적으로 사용자 인터페이스와 도형 관리 기능을 효율적으로 결합하여 실용적인 도형 관리 시스템을 구현한다.

3. 문제 해결 방법

<Shape 클래스 선언부 – Shape.h>

1. 기본 로직

전체 시스템의 기반이 되는 추상화 계층을 제공하여 이를 위해 **추상 클래스로** 정의된다. 도형의 **공통적인 기능(인터페이스)을 제공**하며 실제 구현은 파생 클래스에서 담당하도록 설계한다. Shape 클래스를 상속한 **파생 클래스는 각각의 draw() 메서드를 재정의**하며 이를 통해 프로그램은 **Shape*를 사용하여 도형 객체를 동적으로 관리**할 수 있다. 이는 새로운 도형 클래스를 추가하거나 변경해도 **기존 코드의 수정 없이 유연하게 확장 가능**하도록 설계된다.

2. 헤더 파일

#ifndef SHAPE_H, #define SHAPE_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

#include <iostream>으로 cout, cin과 같은 표준 입출력 객체를 사용하기 위한 라이브러리 포함한다.

using namespace std;를 통해 std 네임스페이스에 있는 식별자를 쉽게 사용하도록 설정한다.

3. 함수 구현

순수 가상 함수 draw()

- virtual void draw() = 0;은 **순수 가상 함수로 구현이 없는 "인터페이스" 역할**
- 도형을 출력하는 동작을 정의하는 **순수 가상 함수**
- 이 함수는 모든 파생 클래스(Circle, Rect, Line)에서 구체적으로 구현해야 하며 **클래스의 다형성**을 가능하게 함

공통 메서드 paint()

- void paint() { draw(); }는 draw()를 호출하는 중개 역할
- 내부적으로 draw()를 호출하는 함수로 외부에서 도형 객체를 호출하고 관리하는 방식으로 사용됨
- draw()를 직접 호출하지 않고 paint()를 사용하는 이유는 Shape 클래스의 **공통된 인터페이스를 통해 일관성을 유지**하기 위함

- 파생 클래스의 구현에 따라 실제 동작이 달라지는 **다형성**을 구현하는 데 사용됨
- 예를 들어, Shape* shape = new Circle(); 형태로 paint()를 호출하면 Circle 클래스의 draw()가 실행됨

가상 소멸자

- virtual ~Shape() {}는 다형성을 활용한 동적 객체 삭제 시, 올바른 소멸자 호출을 보장한다. 예를 들어 delate shape;를 호출하면 Circle이나 Rect 같은 파생 클래스의 소멸자가 호출됨
- Shape 클래스의 **소멸자를 가상으로 선언**하여 **파생 클래스의 소멸자도 올바르게 호출되도록 보장**함 - 동적 메모리 관리에서 매우 중요한 부분

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 #ifndef와 #define으로 시작된 조건부 컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 #endif 사용한다.

< Shape 클래스 구현부 – Shape.cpp>

1. 기본 로직

Shape 클래스는 추상 클래스로서 다른 도형 클래스들이 이를 상속받아 구체적인 draw() 메서드를 구현할 수 있는 기본 인터페이스 역할을 한다. Shape 클래스에 순수 가상 함수 draw()를 포함하고 있기 때문에 Shape.cpp 파일에는 별도의 구현이 없고 헤더 파일에서 정의된 인터페이스만 포함되며 이를 상속받은 자식 클래스에서 각 도형에 맞는 draw() 메서드를 구현하게 된다.

2. 헤더 파일

#include " Shape.h"를 통해 Shape.h에서 선언된 Shape 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

3. 함수 구현

Shape 클래스는 순수 가상 함수 draw()를 포함하는 추상 클래스이기 때문에 구체적인 함수 구현이 없으며 이는 자식 클래스에서 구현됨

<Circle 클래스 선언부 – Circle.h>

1. 기본 로직

추상 클래스 Shape를 상속받은 **파생 클래스**이다. 원(Circle)의 동작을 정의하며 draw() 메서드를 재정의하여 "Circle"이라는 메시지를 출력한다.

2. 헤더 파일

#ifndef CIRCLE_H, #define CIRCLE_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

#include "Shape.h"를 통해 Circle 클래스는 **Shape 클래스의 정의를 가져와 도형의 관리와 출력과 같은 동작을 상속받는다.** Circle 클래스는 Shape 클래스를 상속받아 draw() 메서드를 재정의하며 이 메서드의 구체적인 구현은 Circle.cpp에서 처리된다.

3. 함수 구현

draw() 메서드 선언

- Shape 클래스에서 선언된 **순수 가상 함수를 재정의**하여 원을 출력하는 동작을 구현
- **protected 접근 제어로 선언**되어 외부에서는 직접 호출할 수 없으며 **shape 클래스와 파생 클래스 내부에서만 호출이 가능**함 - 이는 **캡슐화를 강화**하고 도형의 구체적인 동작을 내부에서만 제어할 수 있도록 하기 위함
- 이를 통해 각 **도형의 세부 구현은 은닉**되며 **공통된 인터페이스(paint( ))를 통해서만 도형의 동작을 제어**할 수 있음

구조

- Circle.h는 **클래스 선언만 포함**되며 **실제 구현은 Circle.cpp**에서 처리됨
- 이 설계는 헤더 파일의 **간결성을 유지**하고 변경 시 재컴파일 비용을 줄이는 데 도움됨

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 #ifndef와 #define으로 시작된 조건부 컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 #endif 사용한다.

< Circle 클래스 구현부 – Circle.cpp>

1. 기본 로직

Circle 클래스의 **draw() 메서드를 구현하여 도형의 이름을 출력**하는 역할한다. draw() 메서드는 Shape 클래스에서 선언된 **순수 가상 함수를 오버라이드** 한 것으로 이 구현에서는 "Circle"이라는 문자열을 출력하여 해당 도형이 **원임을 표시**한다. 이를 통해 Shape 클래스를 상속받은 다른 도형 클래스에서도 **일관된 방식으로 도형 이름을 출력**할 수 있다.

2. 헤더 파일

#include "Circle.h"를 통해 Circle.h에서 선언된 Circle 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

3. 함수 구현

draw() 메서드 구현

도형 이름을 출력하는 작업만 수행하며 출력 이후에 사용자에게 어떤 값도 반환하지 않으므로 void 사용

메서드 호출 시 **"Circle"이라는 문자열을 출력**하여 현재 처리 중인 도형이 "원"임을 콘솔에 표시

프로그램 내에서 도형의 이름을 시각적으로 확인할 수 있게 함

<Rect 클래스 선언부 – Rect.h>

1. 기본 로직

Shape를 상속받은 또 다른 **파생 클래스**이다. 사각형(Rectangle)의 동작을 정의하며 draw() 메서드를 재정의하여 “Rectangle”이라는 메시지를 출력한다.

2. 헤더 파일

#ifndef RECT_H, #define RECT_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

#include "Shape.h"를 통해 Rect 클래스는 **Shape 클래스의 정의를 가져와 도형의 관리와 출력과 같은 동작을 상속받는다.** Rect 클래스는 Shape 클래스를 상속받아 draw() 메서드를 재정의하며 이 메서드의 구체적인 구현은 Rect.cpp에서 처리된다.

3. 함수 구현

draw() 메서드

- Shape 클래스의 **순수 가상 함수를 재정의**하여 사각형을 출력하는 동작을 구현함
- **protected로 선언**되어 **클래스 내부와 상속받는 클래스에서만 접근이 가능**함
- draw() 메서드의 구현에서 "Rectangle"이라는 텍스트를 출력하도록 정의되어 있고 이를 통해 paint()를 호출하면 사각형에 해당하는 메시지를 출력함

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 #ifndef와 #define으로 시작된 조건부 컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 #endif 사용한다.

< Rect 클래스 구현부 – Rect.cpp>

1. 기본 로직

Rect 클래스의 **draw() 메서드를 구현하여 도형의 이름을 출력**하는 역할한다. draw() 메서드는 Shape 클래스에서 선언된 **순수 가상 함수를 오버라이드** 한 것으로 이 구현에서는 "Rect"라는 문자열을 출력하여 해당 도형이 **사각형임을 표시**한다.

2. 헤더 파일

#include "Rect.h"를 통해 Rect.h에서 선언된 Rect 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

3. 함수 구현

draw() 메서드 구현

도형 이름을 출력하는 작업만 수행하며 출력 이후에 사용자에게 어떤 값도 반환하지

않으므로 void 사용

메서드 호출 시 **"Rect"이라는 문자열을 출력**하여 현재 처리 중인 도형이 "사각형"임을 콘솔에 표시

프로그램 내에서 도형의 이름을 시각적으로 확인할 수 있게 함

<Line 클래스 선언부 – Line.h>

1. 기본 로직

Shape를 상속받아 선(Line)의 동작을 정의한 **파생 클래스**이다. draw() 메서드를 재정의하여 "Line"이라는 메시지를 출력한다.

2. 헤더 파일

#ifndef LINE_H, #define LINE_H, #endif를 사용하여 구성된 헤더 가드는 헤더 파일이 여러 번 포함되더라도 컴파일 에러가 발생하지 않도록 방지한다.

#include "Shape.h"를 통해 Line 클래스는 **Shape 클래스의 정의를 가져와 도형의 관리와 출력과 같은 동작을 상속받는다.** Line 클래스는 Shape 클래스를 상속받아 draw() 메서드를 재정의하며 이 메서드의 구체적인 구현은 Line.cpp에서 처리된다.

3. 함수 구현

draw() 메서드

- Shape의 **순수 가상 함수인 draw()를 재정의**하여 선을 그리는 동작을 구현함
- Circle.h 및 Rect.h와 **구조적으로 동일**하게 클래스의 선언부만 포함하며 다형성을 통해 **동적으로 관리**됨
- "Line"이라는 메시지를 출력하는 동작을 정의하고 이를 통해 **다른 도형들과 동일한 인터페이스(paint(), draw())로 관리**되며 **코드의 일관성**을 유지함
- 도형클래스를 추가해도 기존 구조를 유지하며 프로그램의 명확성을 보장함

4. 헤더 파일 보호

헤더 파일이 여러 번 포함되는 것을 방지하고 #ifndef와 #define으로 시작된 조건부

컴파일을 종료하며 코드의 안정성과 가독성을 높이기 위해 #endif 사용한다.

< Line 클래스 구현부 – Line.cpp>

1. 기본 로직

Line 클래스의 **draw() 메서드를 구현하여 도형의 이름을 출력**하는 역할한다. draw() 메서드는 Shape 클래스에서 선언된 **순수 가상 함수를 오버라이드** 한 것으로 이 구현에서는 "Line"라는 문자열을 출력하여 해당 도형이 **선임을 표시**한다.

2. 헤더 파일

#include "Line.h"를 통해 Line.h에서 선언된 Line 클래스의 정의를 포함하여 해당 클래스의 멤버 변수와 함수들에 접근할 수 있게 한다.

3. 함수 구현

draw() 메서드 구현

도형 이름을 출력하는 작업만 수행하며 출력 이후에 사용자에게 어떤 값도 반환하지 않으므로 void 사용

메서드 호출 시 **"Line"이라는 문자열을 출력**하여 현재 처리 중인 도형이 "선"임을 콘솔에 표시

프로그램 내에서 도형의 이름을 시각적으로 확인할 수 있게 함

<main 함수 – main.cpp>

1. 기본 로직

사용자가 입력한 명령에 따라 **도형을 삽입, 삭제, 목록을 조회할 수 있는 기능을 제공**하는 그래픽 에디터이다. **벡터를 사용하여** 동적으로 생성된 **도형 객체를 관리**하며 삽입된 도형은 순차적으로 저장되고 인덱스를 통해 접근 가능하다. 사용자가 도형을 삭제하면 벡터에서 해당 도형을 제거하고 동적으로 할당된 메모리를 해제하여 **메모리 누수를 방지**한다. 종료 시에는 저장된 모든 도형 객체를 안전하게 삭제하여 **자원을 정리하고 프로그램을 종료**한다.

2. 헤더 파일

#include <iostream>으로 cout, cin과 같은 표준 입출력 객체를 사용하기 위한 라이브러리 포함한다.

#include <vector>를 통해 도형 객체를 저장하고 관리하기 위해 동적 배열 컨테이너인 std::vector를 사용한다.

#include "Shape.h"으로 Shape 클래스를 포함하여 프로그램에서 도형의 공통 인터페이스와 동작을 정의하고 이를 활용할 수 있게 한다.

#include "Circle.h", #include "Rect.h", #include "Line.h"를 통해 Circle, Rect, Line 클래스의 선언을 포함하여 각각의 도형을 생성하고 관리하기 위해 필요한 정의를 가져온다.

using namespace std;를 통해 std 네임스페이스에 있는 식별자를 쉽게 사용하도록 설정한다.

3. 함수 구현

showMenu()

프로그램의 주요 메뉴를 출력하여 사용자가 선택할 수 있도록 안내

"삽입: 1, 삭제: 2, 모두보기: 3, 종료: 4 >>"라는 메시지를 콘솔에 출력

반환값없으며 사용자와의 상호작용을 위한 출력 역할만 수행

main()

vector<Shape*> shapes

- 도형 객체를 동적으로 저장하고 관리하기 위해 사용
- 각 도형 객체의 포인터를 저장하며 벡터는 동적 크기를 지원하므로 삽입, 삭제 작업에 유연하게 대응

그래픽 에디터 초기화

- "그래픽 에디터입니다."라는 메시지를 출력하여 프로그램의 시작을 알림

반복문 실행

- **while** 문을 사용하여 프로그램이 종료될 때까지 사용자 입력에 따라 동작 수행

- 종료 조건이 만족되면 break를 사용하여 루프를 탈출

도형 삽입

- "선:1, 원:2, 사각형:3 >>" 메시지를 통해 도형의 종류를 입력받음
- 사용자로부터 도형의 종류를 입력받기 위해 **type 변수** 사용
- **if-else** 문을 사용하여 입력 값에 따라 Line, Circle, Rect 객체를 동적으로 생성하고 **push_back()** 메서드를 사용하여 동적으로 생성된 도형 객체의 포인터를 **shapes** 벡터에 추가
- 잘못된 인덱스가 입력되면 **오류 메시지를 출력** - "잘못된 인덱스입니다. 다시 시도하세요."

도형 삭제

- 삭제할 도형의 인덱스를 입력받아 입력된 인덱스가 **벡터 범위 (0 <= index < shapes.size()) 내에 있을 경우**
- **delete shapes[index]**를 호출하여 해당 인덱스의 도형 객체를 벡터에서 제거하고 동적으로 할당된 메모리를 해제
- **shapes.erase(shapes.begin() + index)**를 호출하여 벡터에서 해당 요소를 제거
- 잘못된 인덱스가 입력되면 **오류 메시지를 출력** - "잘못된 인덱스입니다."

모두 보기

- for 문을 사용하여 벡터에 저장된 모든 도형을 순차적으로 출력
- 각 도형 객체의 **paint**() 메서드를 호출하여 도형 이름을 출력

프로그램 종료

- 사용자가 종료를 선택하면 for 문을 사용하여 벡터에 저장된 모든 객체를 순차적으로 삭제하고 동적으로 할당된 메모리를 해제
- 마지막으로 **shapes.clear()**를 호출하여 벡터를 비우고 프로그램을 종료

예외 처리

- 잘못된 입력값이 들어왔을 경우 적절한 오류 메시지를 출력하여 사용자의 입력을 다시 유도함