객체지향프로그래밍 - 과제 #6

- 202184021 소프트웨어전공 박지민

1. 문제 정의

1. 그래픽 에디터를 구현해야 한다.
2. 그래픽 에디터는 선(Line), 원(Circle), 사각형(Rectangle)을 삽입, 삭제, 조회할 수 있어야 한다.
3. 사용자는 다음의 기능을 선택할 수 있다:
   - 삽입 : 새로운 도형을 추가한다.
   - 삭제 : 특정 인덱스의 도형을 삭제한다.
   - 모두보기 : 현재 그래픽 에디터에 있는 모든 도형을 보여준다.
   - 종료 : 프로그램을 종료한다.
4. 도형들은 순서대로 삽입된다.
5. 선, 원, 사각형은 모두 도형이다.

2. 문제 해결 방법

2-1. 클래스 설계

- `Shape` 클래스 (추상 클래스)
  - 순수 가상 함수 `draw()`를 선언하여 하위 클래스에서 구현하도록 한다.
  - `paint()` 함수는 `draw()`를 호출한다.
  - 추상 클래스로 선언하여 직접 객체를 생성할 수 없도록 한다.
- `Line`, `Circle`, `Rect` 클래스
  - `Shape` 클래스를 상속받는다.
  - `draw()` 함수를 오버라이딩하여 각 도형에 맞는 출력을 한다.
- `GraphicEditor` 클래스
  - `vector<Shape*> v`: 도형 객체들을 저장하는 컨테이너이다.
  - `addShape(Shape* pShape)`: 새로운 도형을 벡터에 추가한다.
  - `removeShape(int index)`: 특정 인덱스의 도형을 삭제하고 메모리를 해제한다.
  - `showAllShapes()`: 벡터에 저장된 모든 도형을 순서대로 출력한다.
  - 생성자와 소멸자를 통해 초기화 및 메모리 관리를 한다.

2-2. 동작 과정

1. main 함수에서 `GraphicEditor` 객체를 생성하여 프로그램을 시작한다.
2. 사용자에게 다음 메뉴를 반복적으로 제공한다:
   - 삽입 : 추가할 도형의 종류를 선택하여 해당 도형 객체를 생성하고 addShape 함수를 통해 리스트에 추가한다.
   - 삭제 : 삭제하고자 하는 도형의 인덱스를 입력받아 removeShape 함수를 호출하여 해당 도형을 삭제한다.
   - 모두보기 : showAllShapes 함수를 호출하여 현재 리스트에 저장된 모든 도형의 인덱스와 타입을 출력한다.
   - 종료 : 프로그램을 종료한다.
3. 사용자의 선택에 따라 적절한 함수를 호출하여 기능을 수행
4. 예외 처리를 통해 잘못된 입력에 대한 대응을 한다.
5. 프로그램 종료 시 GraphicEditor 객체의 소멸자를 통해 모든 동적 할당된 도형 객체들을 delete를 사용하여 해제한다.

3. 아이디어 평가

- 객체지향 프로그래밍의 효율적 활용
  - 상속과 다형성을 통해 코드의 재사용성과 유연성을 높였다.
  - Shape 클래스를 추상화하여 공통된 인터페이스를 제공한다.
- **vector** 사용
  - 동적 배열인 `vector`를 사용하여 도형 객체들을 관리한다.
  - `vector`는 요소의 추가 및 삭제가 용이하며, 자동으로 메모리를 관리한다.
  - 도형의 삽입과 삭제가 빈번히 일어나기 때문에 효율적이다.
- **`iterator`** 사용
  - `vector<Shape*>::iterator`를 사용하여 벡터를 순회한다.
  - 반복자를 통해 컨테이너의 요소에 접근하고, 범용성을 높였다.
- 추상 클래스 **Shape**의 활용
  - `Shape`를 추상 클래스로 선언하여 공통 인터페이스를 제공한다.
  - 하위 클래스에서 `draw()` 함수를 구현하도록 강제한다.
  - 이를 통해 다형성을 구현하고, 코드의 유연성과 재사용성을 높였다.
- 동적 메모리 관리
  - `new`를 통해 동적으로 생성된 도형 객체들을 `delete`를 통해 명시적으로 해제한다.
  - 소멸자에서 남은 객체들을 모두 해제하여 메모리 누수를 방지한다.

4. 문제를 해결한 키 아이디어 또는 알고리즘 설명

- **vector** 사용
  - `vector`는 C++ 표준 라이브러리에서 제공하는 동적 배열 컨테이너로, 요소의 추가 및 삭제가 용이하다.
  - 도형의 개수가 가변적이고, 삽입 및 삭제가 빈번히 일어나므로 고정 크기의 배열보다 `vector`가 적합하다.
  - 자동으로 메모리를 관리하며, 필요에 따라 크기를 조절한다.
- **iterator** 사용
  - `iterator`는 컨테이너의 요소를 순회하기 위한 객체이다.
  - 벡터의 처음부터 끝까지 반복자를 사용하여 모든 도형에 접근할 수 있다.
  - 반복자를 사용하면 컨테이너의 내부 구조에 의존하지 않고 요소를 순회할 수 있다.
- 추상 클래스 **Shape**의 필요성
  - `Shape` 클래스를 추상 클래스로 선언하여 공통된 인터페이스를 정의한다.
  - `draw()` 함수를 순수 가상 함수로 선언하여 하위 클래스에서 반드시 구현하도록 한다.
  - 이를 통해 상속받는 클래스들이 공통된 방식으로 호출될 수 있으며, 다형성을 구현할 수 있다.
  - 추상 클래스를 사용하면 `Shape` 타입의 포인터나 참조를 통해 다양한 도형 객체를 일관된 방식으로 처리할 수 있다.
- 동적 메모리 관리와 **delete**의 중요성
  - `new`를 통해 동적으로 생성된 객체는 프로그램이 종료되거나 명시적으로 `delete`를 호출하기 전까지 메모리에 남아있다.
  - 소멸자에서 벡터에 남아있는 모든 객체를 해제하여 프로그램 종료 시 메모리 누수를 방지한다.