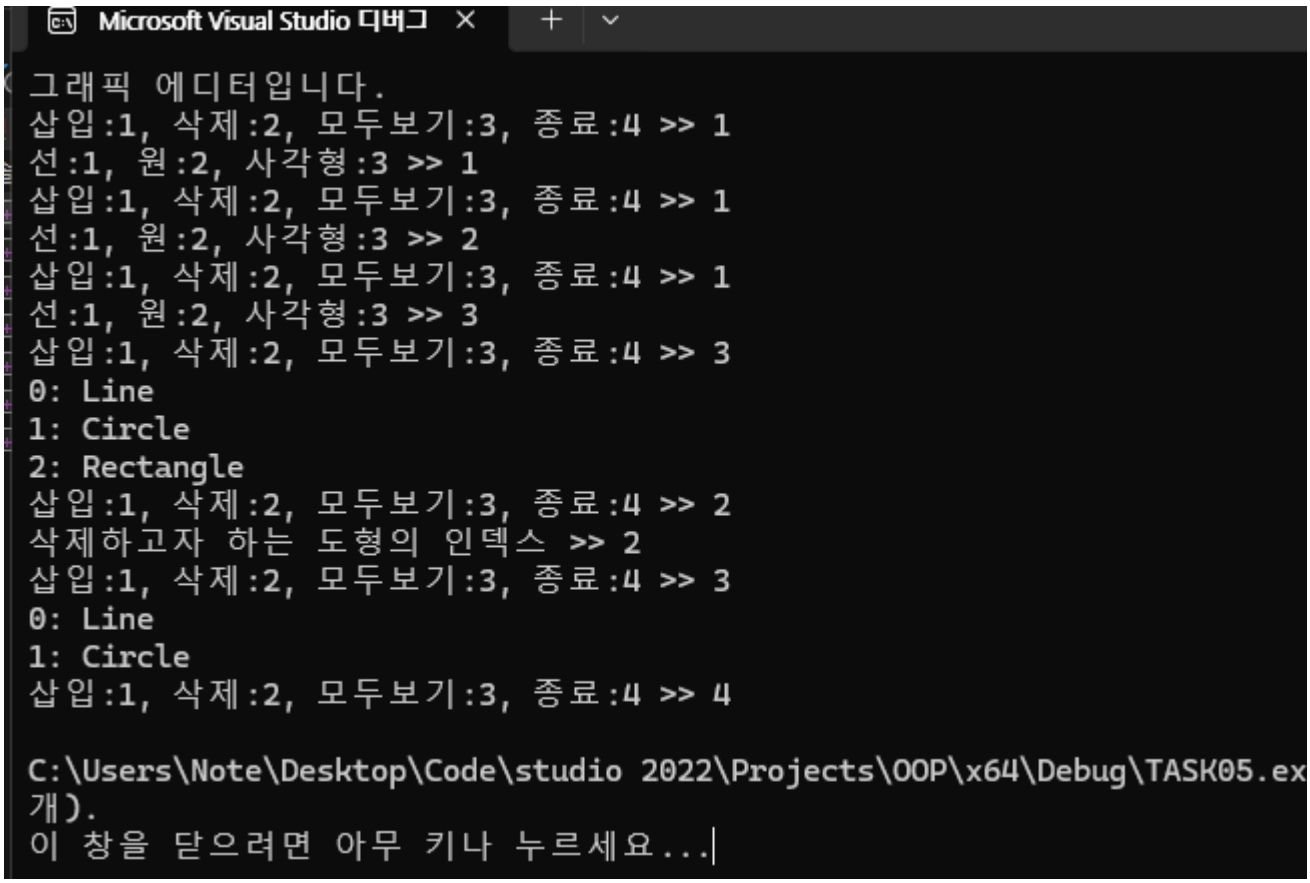
객체지향프로그래밍[02]
202304035 김유진
202104354 지현준

#실행 결과



#문제 정의

GraphicEditor 클래스에 도형(Shape)을 삽입, 삭제, 모두보기, 프로그램을 종료하는 기능을 만들고, Shape을 상속받는 Circle(원), Line(선), Rect(사각형) 클래스로 구성한다. 즉, GraphicEditor 클래스는 Shape 클래스를 상속받은 파생클래스의 객체들을 링크드 리스트로 관리하며, 사용자 입력을 통해 도형을 삽입하고, 특정 도형을 삭제하며 현재 삽입된 모든 도형을 출력하거나 프로그램을 종료할 수 있다. Shape 클래스는 순수 가상 함수인 draw()를 가지기 때문에 파생클래스인 Circle, Line, Rect 클래스는 각 도형에 따라 순수 가상 함수인 draw()를 구현한다. 따라서 문제는 다형성을 활용하여 Shape의 하위 클래스들이 동적으로 생성 및 관리될 수 있도록 설계하는 것이 주된 포인트다.

#문제 해결 방법

GraphicEditor 클래스는 도형들을 동적으로 생성하고 관리한다. Shape 클래스는 파생클래스가 파생클래스 자신을 그릴 수 있도록 하기 위해 순수 가상 함수인 draw()를 정의했다. 이에 따라 Shape 클래스는 추상클래스이므로 Shape 클래스를 상속받는 모든 파생클래스는 구현할 함수인 draw의 인터페이스로 제공받는다. 따라서 Shape 클래스의 파생클래스인 Circle, Line, Rect 클래스는 가상 함수인 draw() 함수를 오버라이딩 하여 자신을 그릴 수 있도록 구현했다.

삽입 기능에서 어떤 도형의 종류를 삽입할지 선택지를 출력하고 입력받는 부분은 UI 클래스에서 객체를 생성하지 않고도 호출할 수 있도록 static 멤버 함수로 정의했다. 사용자가 원하는 도형의 종류(int 타입의 num)를 입력하면, 선택된 도형(int 타입의 num)에 따라 해당 클래스(Circle, Line, Rect)의 객체를 동적으로 생성한 후, 리스트에 추가되도록 구현했다. 이때, 리스트가 비었다면(count==0) 현재 추가한 도형이 첫 번째 도형이 되어 pStart에 새 도형을 생성해 저장하고 pLast도 새 도형을 가리키도록 했다. 리스트가 비어있지 않다면(count != 0) 새 도형을 add 함수의 인자로 넘겨 새 도형을 끝에 추가하여 pLast가 새 도형을 가리키도록 했다. add 함수는 새로 생성된 도형(new Line())을 현재 도형의 next 포인터에 저장하고 새로 생성된 도형의 주소를 반환한다. 마지막으로 count를 증가시켜 도형의 개수를 업데이트 했다.

삭제 기능은 몇 번째 도형을 삭제할지를 출력하고 인덱스를 입력받는 부분은 UI 클래스에서 객체를 생성하지 않고도 호출할 수 있도록 static 멤버 함수로 정의했다. 사용자가 삭제하고자 하는 도형의 인덱스를 입력하면, 인덱스(num)가 도형의 개수를 넘는지를 확인하고 넘지 않는다면 리스트에서 해당 인덱스의 도형을 삭제하고 첫노드라면 pStart를 삭제할 노드의 다음 노드로 변경했다. 중간 노드라면 이전 노드가 삭제 노드의 다음 노드를 가리키도록 연결했다. 개수(count)를 하나 줄이고 메모리 해제(delete del)로 메모리 누수를 방지했다.

모두보기 기능은 리스트에 저장된 모든 도형의 이름을 출력하는 방식으로 구현했으며, Shape타입의 포인터 p를 이용하여 연결된 모든 도형을 방문하면서 paint() 함수를 호출하면, paint()는 동적 바인딩을 통해 Circle, Rect, Line 클래스에 재정의된 각각의 draw() 함수를 호출해 현재 연결된 모든 도형을 화면에 그리도록 했다.

종료 기능은 프로그램 실행을 종료하며, 종료 전 리스트에 저장된 모든 도형 객체의 메모리를 해제해 동적으로 할당된 메모리를 관리한다.

어떤 기능을 수행할지를 출력하고 입력받는 부분은 UI 클래스에서 객체를 생성하지 않고도 호출할 수 있도록 static 멤버 함수로 정의했다. 이 함수를 call() 함수에서 switch-case 문에서 받아 각 기능을 수행하도록 했다. 뿐만 아니라 call 함수에서 UI의 화면 출력 및 키 입력을 수행하는 static 멤버 함수를 호출하도록 했다. while문으로 사용해 부울 변수인 exit으로 관리해 종료 기능을 입력받을 때까지 진행되도록 했다.

main 함수에서는 GraphicEditor 객체를 동적으로 할당받고, 그 객체가 call을 호출하도록 했다. 프로그램 종료 시 GraphicEditor의 소멸자가 호출되어 리스트에 남아 있는 도형 객체들을 모두 삭제하고 메모리를 해제하도록 한다.

프로그램은 다형성을 활용해 Circle, Line, Rect 클래스의 객체들을 Shape 포인터로 관리하며, 각 도형의 이름을 반환하거나 삭제하는 동작을 효율적으로 처리할 수 있도록 설계되어 있다. Shape 클래스의 추상성을 기반으로 새롭게 추가될 수 있는 도형 클래스에 대해서도 유연하게 확장 가능하도록 설계되었다.

#아이디어 평가

(1) Shape 클래스를 추상 클래스로 설계하고 Circle, Line, Rect 클래스가 이를 상속받아 각 도형에 따라 자신을 그릴 수 있도록 구현하도록 하여 새로운 도형 클래스가 추가되어도 기존 코드를 수정하지 않고 간단히 확장할 수 있는 구조를 만들 수 있었다.
(2) GraphicEditor 클래스에서 도형들을 Shape 포인터 리스트로 관리하도록 설계하여 다양한 도형 객체를 다형성을 통해 처리할 수 있었다. 이를 통해 각 도형에 따라 별도의 처리를 분기하지 않고, 공통된 인터페이스를 활용하여 효율적으로 관리할 수 있었다.
(3) 도형의 삽입, 삭제, 모두보기, 종료와 같은 기능을 사용자의 명령에 따라 각각 구현하여 프로그램의 직관성을 높였다. 특히, 리스트를 순회하며 도형 정보를 출력하는 모두보기 기능은 리스트에 추가된 도형의 상태를 한눈에 파악할 수 있게 하여 편의성을 증가시켰다.
(4) 도형 삭제 기능에서 사용자가 입력한 번호를 통해 특정 도형을 삭제하고 메모리를 해제하도록 설계하여 메모리 누수를 방지하여 동적으로 생성된 객체를 효율적으로 관리할 수 있었다.
(5) call() 함수에서 사용자 입력을 기반으로 switch문을 사용하여 각 기능을 명확히 구분하고 실행 흐름을 제어하여 프로그램을 보기 쉽게 만들었다.
(6) 링크드 리스트를 사용함으로써, 도형을 동적으로 추가/삭제하고 메모리를 효율적으로 관리할 수 있었으며, 다형성과의 결합을 통해 다양한 도형을 유연하고 일관된 방식으로 처리할 수 있었다.

#문제를 해결한 키 아이디어 또는 알고리즘 설명

이 문제의 핵심 아이디어 중 하나는 virtual를 활용해 다형성을 구현한 것이다. Shape 클래스에서 도형의 공통 기능인 draw와 같은 멤버 함수를 virtual로 선언하여, 이를 상속받은 Circle, Line, Rect 클래스가 각자 고유한 방식으로 draw를 재정의할 수 있도록 했다. virtual 키워드를 사용함으로써, GraphicEditor 클래스는 도형 객체를 Shape*로 관리하면서도 실제 실행 시점에 각 도형의 draw 구현이 호출되도록 다형성을 구현하여 다양한 도형 타입을 처리하는 공통 인터페이스를 간단하게 유지할 수 있었다.