소스 구현 설명

202304178 구서연

문제 정의 :

콘솔 기반의 그래픽 에디터를 구현한 것으로, 도형(선, 원, 사각형)들을 동적으로 생성, 삭제하고 화면에 표시할 수 있다.

문제 해결 방법 및 키 아이디어 :

- 동적 생성과 삭제

도형(Line, Circle, Rect) 객체는 void GraphicEditor::input_new(int n) 메서드에서 new를 사용해 생성하였다. 그리고 생성된 도형들은 연결 리스트에 추가하여 관리하는 알고리즘으로 진행하였다.

```
void GraphicEditor::input_new(int n) {
    switch (n) {
    case 1: {
        if (node_size == 0) {
            pStart = new Line();
            pLast = pStart;
        }
        else {
            pLast = pLast->add(new Line());
        }
        node_size++;
        break;
    }
    case 2: {
        if (node_size == 0) {
            pStart = new Circle();
            pLast = pStart;
```

```
        }
        else {
            pLast = pLast->add(new Circle());
        }
        node_size++;
        break;
    }
    case 3: {
        if (node_size == 0) {
            pStart = new Rect();
            pLast = pStart;
        }
        else {
            pLast = pLast->add(new Rect());
        }
        node_size++;
        break;
    }
```

```
    default:
        cout << "메뉴를 잘못 선택하셨습니다.₩n";
    }
}
```

이후, 특정 인덱스의 도형을 삭제할 때는 delete를 사용하여 메모리를 해제하였다. bool GraphicEditor::del(int n) 메서드에서 해당 객체의 이전 노드가 다음 노드를 가리키도록 연결 리스트를 재구성하였다. 프로그램 종료 시, 생성된 모든 객체는 delete를 통해 삭제하도록 소멸자 GraphicEditor::~GraphicEditor()가 호출되면서 생성된 모든 객체가 delete를 통해 삭제하였다.

```
bool GraphicEditor::del(int n) {
    int k = 0;
    Shape* target_node = pStart;
    Shape* priv_node = nullptr;
    if (n == 0) {
        pStart = pStart->getNext();
        delete target_node;
    }
    else {
        while ((target_node != NULL) && (k < n)) {
            priv_node = target_node;
            target_node = target_node->getNext();
            k++;
        }
        if (target_node == NULL) {
            cout << "없는 노드입니다.₩n";
            return false;
        }
        else {
            priv_node->setNext(target_node);
            delete target_node;
        }
    }
    node_size--;
    return true;
}
```

```
GraphicEditor::~GraphicEditor() {
    Shape* current = pStart;
    while (current != NULL) {
        Shape* next = current->getNext();
        delete current;
        current = next;
    }
    pStart = NULL;
    pLast = NULL;
}
```

- 상속과 virtual의 활용

문제 조건에 나온 것처럼, Shape는 기본 클래스(base class). Line, Circle, Rect는 파생 클래스(derived class)로 Shape를 상속받게 만들었다. 그 후에 Shape의 paint() 메서드가 다형성을 이용해 알맞은 파생 클래스의 draw() 메서드를 호출하도록 했다.

```
void Shape::paint() {
    draw();
}
```

```
class Shape {
    Shape* next;
protected:
    virtual void draw();
```

```
class Line : public Shape {
public:
    virtual void draw();
};

class Circle : public Shape {
public:
    virtual void draw();
};

class Rect : public Shape {
public:
    virtual void draw();
};
```

Shape 클래스의 draw() 메서드와 이를 오버라이드한 Line, Circle, Rect 클래스의 draw() 메서드에 virtual을 적용시켜서 Shape의 포인터가 파생 클래스의 객체를 가리킬 경우, paint()를 호출하면 파생 클래스의 객체의 draw() 메서드가 실행된다.

아이디어 평가 :

- 동적 생성과 삭제

사용자의 명령으로 동적 생성한 도형 객체를 삭제한 후 거기서 끝내지 않고 해당 객체 앞 노드와 뒷노드를 연결해주었다. 그리고 종료 시에는 그냥 종료하면 남아있는 도형들이 반환이 안 돼서 메모리 누수가 발생한다. 이들을 프로그램 종료 시에 소멸자를 만들어 모두 반환을 해주었다.

- 상속과 virtual의 활용

문제 조건에 맞춰 shape을 각 도형 클래스가 상속하도록 하였다. virtual을 이용하여 파생 클래스에서 함수 오버라이드한 Line, Circle, Rect 클래스의 draw() 메서드를 paint()호출 시 알맞은 도형의 draw() 메서드가 호출도록 동적 바인딩을 해주었다.

이러한 방법들로 문제를 풀면 다음과 같이 올바르게 실행이 되는 것을 볼 수 있다.