

객체지향 프로그래밍 과제5

202284011 김연재

202004006 고창석

[문제 정의]

연결 리스트를 활용하여 도형 객체를 동적으로 관리하고, 사용자 입력을 받아 도형을 삽입, 삭제하거나 모든 도형을 출력하는 기능을 구현한다.

[문제 해결 방법]

```
4  class Shape {
5      Shape* next;
6  protected:
7      virtual void draw() = 0;
8  public:
9      Shape() { next = NULL; }
10     virtual ~Shape() {}
11     Shape* add(Shape* p) {
12         this->next = p;
13         return p;
14     }
15     void paint() {
16         draw();
17     }
18     Shape* getNext() { return next; }
19 };
```

Shape 클래스

역할: 모든 도형의 기본 클래스. 다른 도형 클래스(Line, Circle, Rectangle)가 이 클래스를 상속받는다.

구성요소:

next: 다음 도형을 가리키는 포인터로, 연결 리스트를 형성

draw(): 순수 가상 함수로, 하위 클래스에서 도형을 출력하는 역할을 정의

add(): next 포인터에 새로운 도형을 연결

paint(): draw() 메서드를 호출

getNext(): next 포인터를 반환

You, 16시간 전 | 1 author (You)

```
22 class Line : public Shape {
23 protected:
24     virtual void draw() {
25         cout << "Line" << endl;
26     }
27 };
28
```

You, 16시간 전 | 1 author (You)

```
29 class Circle : public Shape {
30 protected:
31     virtual void draw() {
32         cout << "Circle" << endl;
33     }
34 };
35
```

You, 16시간 전 | 1 author (You)

```
36 class Rectangle : public Shape {
37 protected:
38     virtual void draw() {
39         cout << "Rectangle" << endl;
40     }
41 };

```

Line, Circle, Rectangle 클래스

역할: 각각 "선", "원", "사각형"을 나타낸다.

구성요소:

draw() 메서드: 도형 이름을 출력

You, 19분 전 | 1 author (You)

```
43  class UI {
44  public:
45      static int getMenu() {
46          int key;
47          cout << "삽입:1, 삭제:2, 모두보기:3, 종료:4 >>";
48          cin >> key;
49          return key;
50      }
51      static int getShapeTypeToInsert() {
52          int key;
53          cout << "선:1, 원:2, 사각형:3 >>";
54          cin >> key;
55          return key;
56      }
57      static int getShapeIndexToDelete() {
58          int key;
59          cout << "삭제하고자 하는 도형의 인덱스 >>";
60          cin >> key;
61          return key;
62      }
63  };
```

UI 클래스

역할: 사용자 입력을 받는 데 사용된다.

구성요소:

getMenu(): 사용자에게 메뉴(삽입, 삭제, 모두 보기, 종료)를 선택하도록 한다.

getShapeTypeToInsert(): 삽입할 도형의 종류(선, 원, 사각형)를 입력받는다.

getShapeIndexToDelete(): 삭제할 도형의 인덱스를 입력받는다.

```

65 class GraphicEditor {
66     Shape* pStart;
67     Shape* pLast;
68 public:
69     GraphicEditor() { pStart = pLast = NULL; }
70     void insertItem(int type) {
71         Shape* p = NULL;
72         switch (type) {
73             case 1:
74                 p = new Line();
75                 break;
76             case 2:
77                 p = new Circle();
78                 break;
79             case 3:
80                 p = new Rectangle();
81                 break;
82             default:
83                 break;
84         }
85         if (pStart == NULL) {
86             pStart = p;
87             pLast = p;
88             return;
89         }
90         pLast->add(p);
91         pLast = pLast->getNext();
92     }
93
94     void deleteItem(int index) {
95         Shape* pre = pStart;
96         Shape* tmp = pStart;
97         if (pStart == NULL) {
98             cout << "도형이 없습니다!" << endl;
99             return;
100         }
101         for (int i = 1; i < index; i++) {
102             pre = tmp;
103             tmp = tmp->getNext();
104         }
105         if (tmp == pStart) {
106             pStart = tmp->getNext();
107             delete tmp;
108         }
109         else {
110             pre->add(tmp->getNext());
111             delete tmp;
112         }
113     }
114     void show() {
115         Shape* tmp = pStart;
116         int i = 0;
117         while (tmp != NULL) {
118             cout << i++ << ": ";
119             tmp->paint();
120             tmp = tmp->getNext();
121         }
122     }

```

GraphicEditor 클래스

역할: 그래픽 에디터의 주요 기능(도형 관리)을 구현합니다.

구성요소:

pStart, pLast: 연결 리스트의 시작 및 마지막 도형을 가리킵니다.

도형 삽입 (insertItem)

UI::getShapeTypeToInsert()를 호출해 삽입할 도형 종류를 입력받는다.

선택된 도형(Line, Circle, Rectangle)을 생성.

연결 리스트의 끝에 도형을 추가한다.

리스트가 비어 있으면 pStart와 pLast를 새 도형으로 설정.

비어 있지 않으면 pLast->add(p)를 통해 새 도형을 리스트에 연결하고 pLast를 갱신.

도형 삭제 (deleteItem)

UI::getShapeIndexToDelete()를 호출해 삭제할 도형의 인덱스를 입력받는다.

연결 리스트에서 해당 인덱스의 도형을 삭제.:

인덱스가 0이면 pStart를 갱신.

그 외에는 이전 도형(pre)의 next를 현재 도형의 next로 설정, 리스트를 갱신.

삭제된 도형 객체는 delete로 메모리를 해제.

도형 출력 (show)

연결 리스트의 모든 도형을 순차적으로 탐색하며 출력.:
인덱스(i)를 0부터 증가시키며 각 도형을 출력.

```
122 void run() {
123     cout << "그래픽 에디터입니다." << endl;
124     int menu, index, type;
125     while(true) {
126         menu = UI::getMenu();
127         switch (menu) {
128             case 1:
129                 type = UI::getShapeTypeToInsert();
130                 insertItem(type);
131                 break;
132             case 2:
133                 index = UI::getShapeIndexToDelete();
134                 deleteItem(index);
135                 break;
136             case 3:
137                 show();
138                 break;
139             default:
140                 return;
141         }
142     }
143 }
144 };
```

run(): 프로그램 실행의 메인 루프. 사용자로부터 명령을 입력받아 처리.

```
146 int main() {
147     GraphicEditor graphicEditor;
148     graphicEditor.run();
149
150     return 0;
151 }
```

main 함수

GraphicEditor 객체를 생성하고, run() 메서드를 호출한다.
프로그램을 시작함.

GraphicEditor::run() 메서드

사용자에게 반복적으로 메뉴를 표시하고, 명령에 따라 적절한 메서드를 호출
사용자는 다음 네 가지 작업 중 하나를 선택할 수 있다.

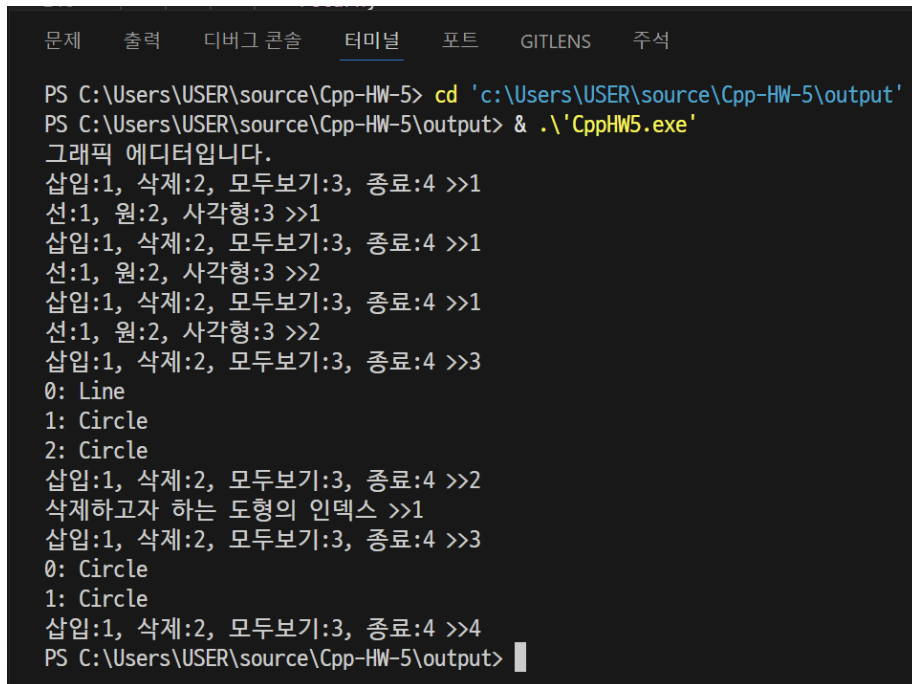
삽입: 도형을 선택하여 연결 리스트에 추가.

삭제: 연결 리스트에서 특정 인덱스의 도형을 삭제.

모두 보기: 모든 도형을 순서대로 출력.

종료: 프로그램 종료.

[소스 수행 결과 화면 캡처]



```
문제   출력   디버그 콘솔   터미널   포트   GITLENS   주석

PS C:\Users\USER\source\Cpp-HW-5> cd 'c:\Users\USER\source\Cpp-HW-5\output'
PS C:\Users\USER\source\Cpp-HW-5\output> & .\'CppHW5.exe'
그래픽 에디터입니다.
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>1
선:1, 원:2, 사각형:3 >>1
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>1
선:1, 원:2, 사각형:3 >>2
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>1
선:1, 원:2, 사각형:3 >>2
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>3
0: Line
1: Circle
2: Circle
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>2
삭제하고자 하는 도형의 인덱스 >>1
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>3
0: Circle
1: Circle
삽입:1, 삭제:2, 모두보기:3, 종료:4 >>4
PS C:\Users\USER\source\Cpp-HW-5\output>
```

[평가]

김연재: 이번 과제를 통해 연결 리스트를 활용한 동적 객체 관리의 장점을 실감할 수 있었다. 도형을 동적으로 생성하고 삭제할 수 있도록 설계한 덕분에 메모리 효율성을 높일 수 있었다. 특히, Shape 클래스를 추상 클래스로 설계하고 하위 클래스에서 이를 상속받아 기능을 구현함으로써 코드 재사용성과 확장성을 동시에 만족시킬 수 있었다.

고창석: 도형 삽입, 삭제, 출력 기능을 연결 리스트로 구현함으로써 동적 메모리 관리를 경험할 수 있었다. 상위 클래스에서 공통 기능을 제공하고, 하위 클래스에서 고유한 기능을 구현하는 방식은 객체 지향의 장점을 잘 보여줬다고 생각한다.