실습문제

- 교재 471페이지, 실습문제 9-10를 응용한 문제
  - 프로젝트명 : 학번_Chap_9_10
  - main 파일명 : 학번_Chap_9_10.cpp
  - 교재 [그림 9-13]을 활용한 간단한 그래픽 편집기를 콘솔 바탕으로 만들어 보자. 그래픽 편집기의 기능은 "자동삽입", "모두삭제", "모두보기", "모두이동", "종료"의 5가지이고, 실행과정 및 실행결과는 다음 두 페이지를 참고하라.
  - Shape 클래스와 이를 상속받은 Circle, Line, Rect 클래스는 [그림 9-13]을 이용하고, 필요한 클래스와 main() 함수를 작성하라.
  - 전체 프로그램의 구성도는 교과서 471 페이지 아래 힌트 부분을 참조하기 바란다.

실행화면 - 1

각 메뉴 항목을 실행한 후 항상 모든 객체를 화면에 보여 준다.

자동 삽입: 임의의 개수의 그래픽 객체를 자동으로 생성함, 각 객체들의 좌표 또한 자동 생성됨, 원의 경우 반지름 길이와 중심 좌표임

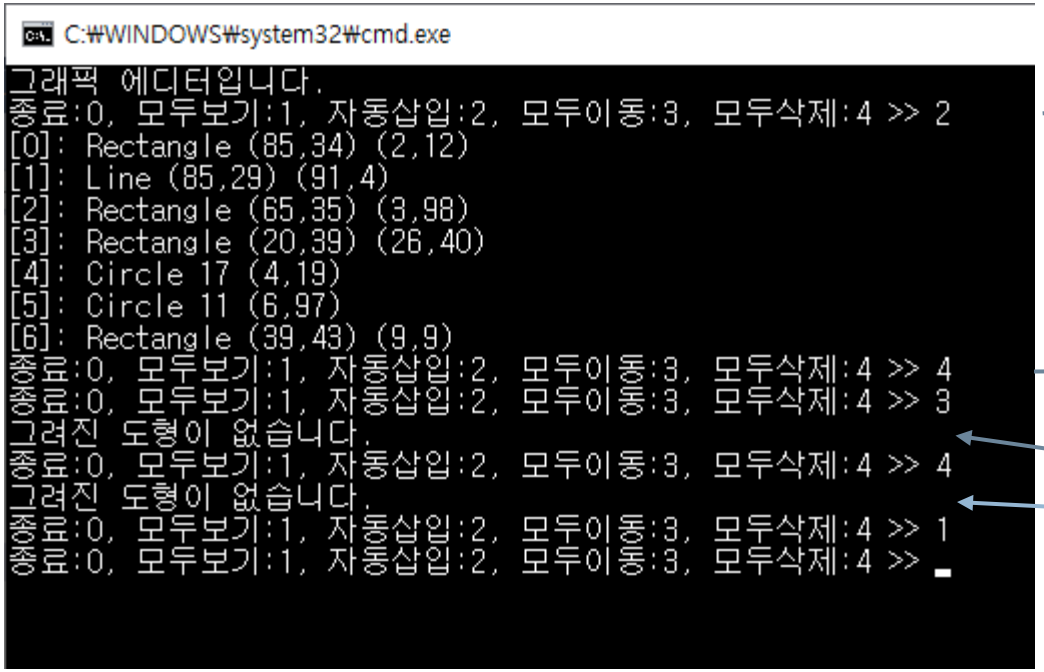
객체 이동: 사용자가 입력한 x, y 이동 량만큼 모든 그래픽 객체의 좌표를 이동함

모두 삭제: 모든 그래픽 객체를 삭제함

삭제 후 새로 객체들을 자동 삽입함 (개수는 난수로 생성)

프로그램 종료하기

실행화면 - 2


C:₩WINDOWS₩system32₩cmd.exe
그래픽 에디터입니다.
종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> 2
[0]: Rectangle (85,34) (2,12)
[1]: Line (85,29) (91,4)
[2]: Rectangle (65,35) (3,98)
[3]: Rectangle (20,39) (26,40)
[4]: Circle 17 (4,19)
[5]: Circle 11 (6,97)
[6]: Rectangle (39,43) (9,9)
종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> 4
종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> 3
그려진 도형이 없습니다.
종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> 4
그려진 도형이 없습니다.
종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> 1
종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >>
자동 삽입
모두 삭제
도형이 없을 경우 에러 메시지 출력 (모두 보기는 예외)

학번_Chap_9_10.cpp


```
main()에서 사용되는 클래스가 선언된 헤드 파일 include

int main()
{
    GraphicEditor 변수 g 선언
    g의 멤버 함수 run() 호출  // g는 포인터 변수가 아닌 일반 객체임
}

// 프로그램의 실행 결과는 앞의 실행화면 1, 2와 유사해야 함
// 난수를 이용하므로 좌표 값과 생성 그래픽 객체 종류 및 개수는 다를 수 있음
```

Shape 클래스와 이를 상속받은 클래스 작성

- 교재 [그림 9-13]을 보고 다음의 10개의 파일들을 먼저 작성하시오.
  - Point.h // 교재에 없는 파일임, 뒤 페이지 참고
  - Point.cpp // 하나의 좌표 값을 저장하는 Point 클래스
  - Shape.h
  - Shape.cpp
  - Circle.h
  - Circle.cpp
  - Rect.h
  - Rect.cpp
  - Line.h
  - Line.cpp

- 뒤 페이지의 Point.h와 UI.h를 참조하여 헤드 파일이 여러 번 include되어도 에러가 발생하지 않도록, `#ifndef, #define, #endif` 문장들을 **모든 헤드 파일**에 삽입하라. (앞 페이지 헤드파일들도 마찬가지)

기타 추가해야 할 파일들

- 추가로 다음 파일들을 작성하시오.
  - GraphicEditor.h
  - GraphicEditor.cpp
  - UI.h
  - UI.cpp
  - Factory.h // 교재에 없는 파일임
  - Factory.cpp // 객체들을 자동으로 생성해 주는 소스

- 헤드 파일이 여러 번 include되어도 에러가 발생하지 않도록, `#ifndef, #define, #endif` 문장들을 **모든 헤드 파일**에 삽입

파일 Point.h

- 화면의 한 점의 좌표 값을 가지는 Point 클래스
- 교재에 없는 클래스임

```
#ifndef POINT_H
#define POINT_H

#include <string>
using namespace std;

class Point { // 한 점의 x, y 좌표 값을 가지고 있는 클래스
    int x; // 점의 x 좌표 값
    int y; // 점의 y 좌표 값
public:
    Point(int x = 0, int y = 0);
    void move(int width, int height);
    string toString();
};

#endif
```

파일 Point.cpp


```
#include "Point.h"

// 객체 초기화: x, y 좌표 초기화
Point::Point(int x, int y) {
     함수 인자인 x, y를 이 객체의 x, y 멤버에 저장
}

// x, y 좌표를 각각 width, height만큼 이동시킨다.
void Point::move(int width, int height) {
    이 객체의 x, y 멤버에 각각 width, height를 더하여 좌표를 이동함
}

// (x,y) 좌표를 문자열로 변환하여 리턴, 예) "(10,15)"
// to_string(x): 정수 250을 "250"으로 변화함; <string> 헤드 파일에 선언되어 있음
string Point::toString() {
    return "(" + to_string(x) + "," + to_string(y) + ")";
}
```

파일 Shape.h

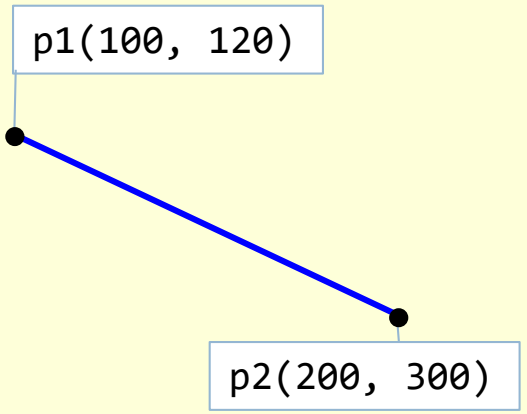
- 교재 [그림 9-13] 기존 Shape.h에 아래처럼 추가하라.

```
#include "Point.h"

class Shape {
    . . .
public:
    . . .

    // 기존 객체를 x, y 축 방향으로 width, height 만큼 각각 이동함
    virtual void move(int width, int height) = 0;
};
```

파일 Line.h

- 교재 [그림 9-13] 기존 Line.h에 아래처럼 추가하라.

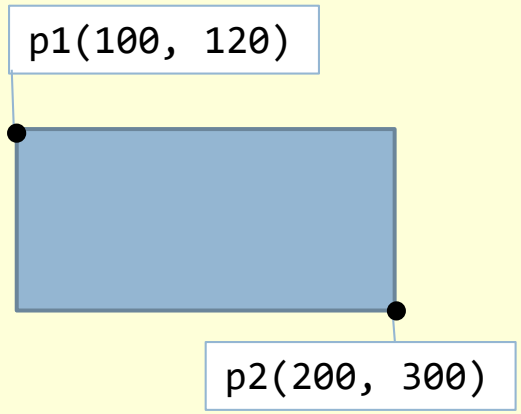
```
#include "Shape.h"

class Line : public Shape {
    Point p1;    // 라인의 시작 좌표
    Point p2;    // 라인의 끝 좌표
protected:
    . . .
public:
    Line(const Point &p1, const Point &p2);
    void move(int width, int height) override;
};
```



파일 Rect.h

- 교재 [그림 9-13] 기존 Rect.h에 아래처럼 추가하라.

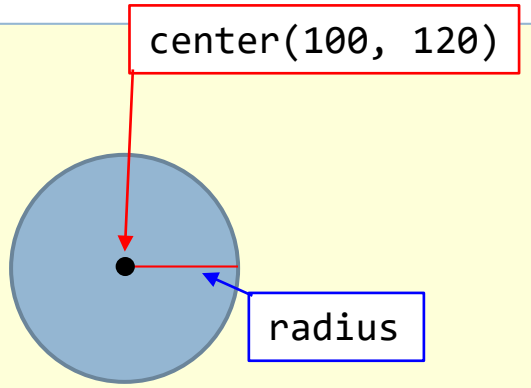
```
#include "Shape.h"

class Rect : public Shape {
    Point p1;    // 사각형의 왼쪽 위쪽 좌표
    Point p2;    // 사각형의 오른쪽 아래쪽 좌표
protected:
    . . .
public:
    Rect(const Point &p1, const Point &p2);
    void move(int width, int height) override;
};
```



파일 Circle.h

- 교재 [그림 9-13] 기존 Circle.h에 아래처럼 추가하라.

```
#include "Shape.h"

class Circle : public Shape {
    Point center;// 원의 중심 좌표
    int   radius;// 반지름의 길이
protected:
    . . .
public:
    Circle(int radius, const Point &center);
    void move(int width, int height) override;
};
```



파일 UI.h

- 모든 멤버 함수들이 모두 정적(static) 함수임
- 입력과 출력을 담당하는 전역 함수들의 모음
- static 함수 호출은 예제 6-10 참조

```
#ifndef UI_H
#define UI_H

// 아래 println(string), print(string)에서 string을 사용했기 때문에 include시킴
#include <string>
using namespace std;

class UI {
public:
    static void println(string msg);        // msg 문자열 및 줄 바꾸기 출력
    static void print(string msg);          // msg 문자열 출력
    static int  printGetInt(string msg);// msg 문자열 출력 후 정수 하나 읽어 리턴
    static int  getMainMenu();   // 메인 메뉴 종류 출력하고 메인 메뉴 값 입력 받아 리턴
    // 사용자로부터 x, y축으로 이동할 량 width와 height값을 입력 받음
    static void getWidthHeight(int &width, int &height);  // 정수 두 개 입력 받음
};

#endif
```

UI.cpp: 메시지 출력 함수


```
표준입출력 헤드파일 include 및 std 선언
UI 자신의 헤드 파일 include

// UI의 모든 멤버 함수들은 static 함수임; 함수 정의 때는 static 사용하지 않음

int UI::printGetInt(string msg) { // msg 문자열 출력 후 정수 값 하나 읽어 리턴
    문자열 msg 출력
    임의의 정수형 변수 하나를 선언
    키보드에서 정수 값 하나 읽어 변수에 저장          // 예제 2-3 참조
    변수 값 리턴
}

UI의 println 함수 {
    문자열 msg와 줄바꾸기를 출력
}

UI의 print(string msg) 함수 {
    문자열 msg 출력
}
```

UI.cpp: 메뉴 출력 및 입력 값 받기


```
// 메인 메뉴 종류 출력하고 메인 메뉴 값 입력 받아 리턴
UI의 getMainMenu 함수 {
    // printGetInt() 함수를 사용하여
    // "종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> " 를 출력하고
    // 입력한 메뉴 번호 값을 읽어 리턴함. 즉,
    return printGetInt("종료:0, 모두보기:1, 자동삽입:2, 모두이동:3, 모두삭제:4 >> ");
}
// 삽입할 도형 종류 출력하고 종류 값 입력 받아 리턴
UI의 getWidthHeight 함수 {
    "X축과 Y축으로 이동할 양은(정수 두개 입력)? >> " 를 출력
    정수 값 두개를 읽어 인자인 width와 height에 저장 // 교재 p.66 참조
}
```

파일 Line.cpp


```
#include "Line.h"
#include "UI.h"

// 생성자: 객체 멤버 초기화
Line::Line(const Point &p1, const Point &p2) {
    // 함수 인자로 넘어 온 p1과 p2를 이 객체의 p1, p2에 저장
    this->p1 = p1;
    this->p2 = p2;
}

void Line::draw() { // UI의 static 함수 호출
    // 화면에 라인의 좌표 값을 출력, 예) "Line (10,15) (20,30)"
    UI::println("Line " + p1.toString() + " " + p2.toString());
}

// 기존 객체의 위치를 width, height만큼 옮긴다.
void Line::move(int width, int height) {
    // 기존 p1의 위치를 width, height만큼 옮긴다.
    p1.move(width, height);
    // 기존 p2의 위치를 width, height만큼 옮긴다.
    p2.move(width, height);
}
```

파일 Rect.cpp와 Circle.cpp


```
// Rect.cpp: Line.cpp를 참고하여 작성할 것
필요한 헤드 파일 및 UI 헤드 파일 include

Rect 의 생성자 함수 {
    함수 인자로 넘어 온 p1과 p2를 이 객체의 p1, p2에 저장
}
Rect 의 draw() {  // UI의 적절한 멤버 함수 사용할 것
    화면에 사각형의 좌표 값을 출력, 예) "Rectangle (10,15) (20,30)"
}
Rect 의 move 함수 { 기존 p1과 p2의 위치를 width, height만큼 옮긴다. }
```

```
Circle.cpp: 필요한 헤드 파일 및 UI 헤드 파일 include

Circle 의 생성자 함수 {
    함수 인자로 넘어 온 radiu와 center를 이 객체의 해당 멤버에 저장
}
Circle 의 draw() { // 반지름은 to_string(radius)를 이용하여 문자열로 변화
                   // 이 함수 사용법은 Point.cpp의 toString() 함수 참조할 것
    // UI의 적절한 멤버 함수 사용할 것
    화면에 반지름과 중심 좌표 값을 출력, 예) "Circle 5 (20,30)"
}
Circle 의 move 함수 { 기존 중심 좌표의 위치를 width, height만큼 옮긴다. }
```

파일 GraphicEditor.h


```
class Shape; // 아래에 Shape*가 사용되었기 때문에 클래스 이름을 선언해야 함
             // 실제 이 클래스에 대한 정의는 Shape.h에 있음

class GraphicEditor {
    // 메인 메뉴의 종류
    enum { EXIT=0, ALL_PAINT=1, AUTO_INSERT=2, ALL_MOVE=3, ALL_REMOVE=4};
    Shape* pStart; // 삽입된 그래픽 객체들의 linked 리스트의 맨 처음을 가리킴
    Shape* pLast;  // 삽입된 그래픽 객체들의 linked 리스트의 맨 마지막을 가리킴
    void add(Shape* p); // 새로운 그래픽 객체 p를 맨 마지막인 pLast 다음에 추가
    bool empty();           // 객체가 하나도 없는지 체크
    void removeAllShapes(); // 모든 그래픽 객체 삭제
protected:
    void autoInsert();// 임의의 개수의 그래픽 객체를 자동으로 삽입함
    void allRemove(); // 모든 그래픽 객체들을 삭제
    void allPaint();  // 삽입된 모든 그래픽 객체들을 화면에 출력
    void allMove();   // 모든 객체들을 사용자가 입력한 x, y 이동 량만큼 이동
public:
    GraphicEditor();
    ~GraphicEditor();
    void run();   // 메인 메뉴를 보여 주고 사용자가 선택한 작업을 실행함
};
```

GraphicEditor.cpp: 생성자, 소멸자 함수


```
표준입출력 헤드파일 include 및 std선언
GraphicEditor.cpp에서 사용하는 모든 클래스의 헤드 파일 include
(즉, GraphicEditor.h, Shape.h, UI.h, Factory.h 파일)
(위 헤드 파일의 배치 순서도 중요함)
// GraphicEditor 클래스의 멤버인 pStart와 pLast의 용도는 [그림 9-12 참조할 것]

GraphicEditor 생성자 함수 {
    pStart와 pLast 멤버 데이터를 nullptr(NULL아닌 nullptr)로 초기화
}
GraphicEditor removeAllShapes 함수 {
    // 현재 생성된 모든 객체를 삭제: [그림 9-11]의 마지막 부분 참조할 것
    // Shape 객체의 linked list인 pStart에서 pLast까지 모두 삭제
    for (Shape *p = pStart, *q; p != nullptr; p = q) {
        p의 다음 객체에 대한 포인터를 구해 q에 저장;
        p가 포인터하는 객체 반납;
    }
    pStart와 pLast 멤버 데이터를 nullptr로 초기화
}
GraphicEditor 소멸자 함수 {
    removeAllShapes 함수 호출하여 리스트의 모든 객체 삭제
}
```

GraphicEditor.cpp: empty(), add() 함수


```
// 새로운 그래픽 객체 p를 맨 마지막인 pLast 다음에 추가
// GraphicEditor 클래스의 멤버인 pStart와 pLast의 용도는 [그림 9-12 참조할 것]
GraphicEditor add(Shape *p) 함수 {
    pStart가 nullptr이면 {
        // [그림 9-11] 12~13행 참조
        pStart, pLast를 p로 설정한 후 리턴
    }
    else {
        // [그림 9-11] 16행 부분 참조
        pLast 뒤에 p를 추가하고(pLast의 멤버 함수를 호출하여 p를 추가할 것)
        리턴 값을 pLast에 저장(pLast가 삽입된 마지막 도형을 포인터하게 함)
    }
}
// 객체가 하나도 없으면 에러 메시지 출력 후 true 리턴, 있을 경우 false 리턴
GraphicEditor empty() 함수 {
    pStart가 nullptr이면 {
        UI의 적절한 멤버 함수 이용하여 "그려진 도형이 없습니다." 출력
        적절한 값 리턴
    }
    적절한 값 리턴
}
```

GraphicEditor.cpp: allPaint(), autoInsert()


```
// 삽입된 모든 그래픽 객체들을 화면에 출력
GraphicEditor allPaint 함수 {
    // 처음부터 끝까지 linked list를 따라 가면서 도형을 그린다.
    // [그림 9-11] 중간부분 참조할 것
    Shape의 포인터형 변수 p를 선언하고 pStart로 초기화
    for (int i = 0; p != nullptr; ++i, p = p->getNext()) {
       // 인덱스 출력: 예), [1] 또는 [4]
       UI 적절한 멤버 함수 이용하여 "[" + to_string(i) + "]: " 출력
       도형을 그리는 p의 멤버 함수 호출 // 도형 그리기 public 함수 호출
    }
}
```

```
// 자동으로 객체를 생성하여 리스트에 추가한다.
GraphicEditor autoInsert 함수 {
   // 자동 삽입할 그래픽 객체의 개수를 얻어 옴(난수 발생)
   정수형 변수 size를 선언하고 Factory::getSize()를 호출하여 리턴 값으로 초기화
   for 문을 이용하여 size만큼 반복 수행
        Factory::create()호출하여 새로운 그래픽 객체를 자동 생성하고
        // Factory::create()의 리턴 값을 add()의 인자로 주면 됨
       생성된 객체를 GraphicEditor의 add() 함수 호출하여 리스트에 추가
   모든 객체를 화면에 보여줌 (allPaint() 사용)
}
```

GraphicEditor.cpp: allMove(), allRemove()


```
// 현재 생성된 모든 객체를 이동한다.
GraphicEditor allMove 함수 {
    객체가 전혀 없을 경우 에러 출력 후 여기서 리턴 // empty() 사용
    정수형 변수 width, height 선언 // 각 도형의 이동할 양
     // 아래 함수는 "참조에 의한 호출"임; 그림[5-8] 참조
    UI의 getWidthHeight() 호출하여 사용자로부터 width, height 입력 받음
    // 처음부터 끝까지 링크를 따라 가면서
    // 각 도형을 사용자가 지정한 width, height만큼 이동한다.
    for를 이용하여 리스트의 처음부터 끝까지 링크를 따라 가면서
    // for문은 allPaint()의 for를 적절히 응용 (필요 없는 것은 빼고)
         각 도형의 move 함수 호출 // 도형 옮기기
    모든 객체를 화면에 보여줌
}
```

```
// 현재 생성된 모든 객체를 삭제한다.
GraphicEditor allRemove 함수 {
    객체가 전혀 없을 경우 에러 출력 후 여기서 리턴 // empty() 사용
    removeAllShapes 함수 호출하여 리스트의 모든 객체 삭제
}
```

GraphicEditor.cpp: run() 함수


```
// 메인 메뉴를 보여 주고 사용자가 선택한 작업을 실행함
GraphicEditor run 함수 {
    // UI 클래스의 함수들은 모두 static 함수임; 함수 호출은 예제 6-10 참조
    UI의 적절한 함수를 사용하여 “그래픽 에디터입니다.” 출력
    while문을 이용하여 다음을 반복 수행 {
        정수형 변수 menu를 선언함과 동시에
            UI의 getMainMenu()를 호출하여 리턴 값으로 초기화
        switch(menu) {
            // 모두보기, ALL_PAINT는 enum 열거자이며 상수처럼 사용가능하며
            // GraphicEditor.h에 선언되어 있음
            case ALL_PAINT :
                 allPaint(); break;
            위와 같은 방법으로
            AUTO_INSERT면 autoInsert() 호출 // 자동삽입
            ALL_MOVE면 allMove()호출        // 모두이동
            ALL_REMOVE면 allRemove() 호출   // 모두삭제
            EXIT면 return                   // 끝내기, 이 함수에서 리턴함
            그 외(default)는 “명령 선택 오류” 출력 (UI의 멤버 함수 사용)
        }
    }
}
```

파일 Factory.h


```
#include "Shape.h"

class Factory {
public:
    // 자동 생성할 그래픽 객체의 종류
     enum { LINE = 0, CIRCLE = 1, RECT = 2 };
     static Shape *create(); // 임의의 그래픽 객체를 자동 생성함
    // shapeType은 LINE, CIRCLE, RECT 등 셋 중 하나의 값을 가짐
     static Shape *create(int shapeType);  // shapeType 객체 생성
     static int    getSize();    // 자동 생성할 그래픽 객체의 수 리턴
};
```

파일 Factory.cpp (1)


```
#include <random>
using namespace std;

#include "Factory.h"
#include "Point.h"
#include "Shape.h"
#include "Line.h"
#include "Circle.h"
#include "Rect.h"

class Rand {
   // 난수를 생성하는 엔진
     default_random_engine re;
   // 발생된 난수 값들이 일정 범위의 균등 분포(unfiform)로 매핑
     uniform_int_distribution<int> dist;
public:
   // [low, high] 범위 내의 값이 발생하도록 dist 초기화
     Rand(int low, int high) : dist(low, high) { }
   // [low, high] 범위 내의 난수 발생
     int operator() () { return dist(re); }
};
```

파일 Factory.cpp (2)


```
enum { MAX_XY = 99, MIN_SIZE = 5, MID_SIZE = 10, MAX_SIZE = 20 };

// [LINE, POLYGON] 범위의 난수 발생기, 자동 생성할 그래픽 종류 생성
static Rand randShapeType(Factory::LINE, Factory::RECT);
// [0, 99] 범위의 난수 발생기, x 또는 y 좌표 값 생성시 사용
static Rand randXY(0, MAX_XY);
// [5, 20] 범위의 난수 발생기, 원의 반지름 생성시 활용
static Rand randRadius(MIN_SIZE, MAX_SIZE);
// [5, 10] 범위의 난수 발생기, 다각형 점의 개수 또는 그래픽 객체의 개수 생성시 활용
static Rand randSize(MIN_SIZE, MID_SIZE);

// 다각형의 점들 위치를 저장하는 좌표 배열, 매번 동적할당하지 않음
static Point points[MID_SIZE];

// 새로운 Point 객체를 생성함: x, y 좌표는 자동 지정
static Point newPoint(){ return Point(randXY(), randXY()); }

// 자동 생성할 그래픽 객체의 개수를 난수로 생성
int Factory::getSize() {
    return randSize(); // [5, 10] 범위의 난수 생성
}
```

파일 Factory.cpp (3)


```
// shapeType 객체를 생성함: 각 객체의 멤버들은 자동으로 설정됨
Shape* Factory::create(int shapeType) {
    switch (shapeType) {
    case LINE:   // 라인
        return new Line(newPoint(), newPoint());
    case CIRCLE: // 원
        return new Circle(randRadius(), newPoint());
    case RECT:   // 사각형
        return new Rect(newPoint(), newPoint());
    }
    return nullptr;
}

// 라인, 원, 사각형 중 임의로 하나의 객체를 생성함
Shape* Factory::create() {
    // randShapeType(): [LINE, POLYGON] 범위의 난수를 발생
    return create(randShapeType());
}
```