Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №4

з дисципліни:
«Об’єктно-орієнтоване програмування»

Виконав:
Баран Павло Юрійович
студент групи ІП-03
Номер у списку групи: 1

Перевірив:
Порєв В. М.

Київ 2021

Варіант:

1. Статичний масив Shape *pcshape[N];
2. "Гумовий" слід при вводі об’єктів: пунктирна лінія червоного кольору.
3. Ввід прямокутника: по двом протилежним кутам.
4. Відображення прямокутника: зелений контур без заповнення.
5. Ввід еліпсу: по двом протилежним кутам охоплюючого прямокутника.
6. Відображення еліпсу: чорний контур з білим заповненням.
7. Позначка поточного типу об’єкту, що вводиться, в заголовку вікна.

Програмний код:
Lab2.cpp

```
#include "framework.h"
#include "lab2.h"
#include "my_editor.h"
#include "toolbar.h"
#include "point_shape.h"
#include "line_shape.h"
#include "rect_shape.h"
#include "ellipse_shape.h"
#include "cube_shape.h"
#include "OlineO_shape.h"

#define MAX_LOADSTRING 100

// Глобальные переменные:
HINSTANCE hInst;                                // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING];                  // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING];            // имя класса главного окна
MyEditor editor;
Toolbar toolbar;

// Отправить объявления функций, включенных в этот модуль кода:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
```

```
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR    lpCmdLine,
    _In_ int       nCmdShow)
{
    InitCommonControls();
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Разместите код здесь.

    // Инициализация глобальных строк
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB2, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Выполнить инициализацию приложения:
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_LAB2));

    MSG msg;

    // Цикл основного сообщения:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int)msg.wParam;
}



//
//  ФУНКЦИЯ: MyRegisterClass()
//
//  ЦЕЛЬ: Регистрирует класс окна.
```

```
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
   WNDCLASSEXW wcex;

   wcex.cbSize = sizeof(WNDCLASSEX);

   wcex.style = CS_HREDRAW | CS_VREDRAW;
   wcex.lpfnWndProc = WndProc;
   wcex.cbClsExtra = 0;
   wcex.cbWndExtra = 0;
   wcex.hInstance = hInstance;
   wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB2));
   wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
   wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
   wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LAB2);
   wcex.lpszClassName = szWindowClass;
   wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

   return RegisterClassExW(&wcex);
}

//
//   ФУНКЦИЯ: InitInstance(HINSTANCE, int)
//
//   ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно
//
//   КОММЕНТАРИИ:
//
//        В этой функции маркер экземпляра сохраняется в глобальной переменной, а
также
//        создается и выводится главное окно программы.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW |
WS_CLIPCHILDREN,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
```

```
   return TRUE;
}

//
//  ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  ЦЕЛЬ: Обрабатывает сообщения в главном окне.
//
//  WM_COMMAND  - обработать меню приложения
//  WM_PAINT    - Отрисовка главного окна
//  WM_DESTROY  - отправить сообщение о выходе и вернуться
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam,
LPARAM lParam)
{
   switch (message)
   {
   case WM_CREATE:
      editor.Start(new PointShape);
      toolbar.OnCreate(hWnd);
      break;
   case WM_SIZE:
      toolbar.OnSize(hWnd);
      break;
   case WM_LBUTTONDOWN:
      editor.OnLBdown(hWnd);
      break;
   case WM_LBUTTONUP:
      editor.OnLBup(hWnd);
      break;
   case WM_MOUSEMOVE:
      editor.OnMouseMove(hWnd);
      break;
   case WM_NOTIFY:
      toolbar.OnNotify(hWnd, lParam, wParam);
      break;
   case WM_PAINT:
      editor.OnPaint(hWnd);
      break;
   case WM_COMMAND:
   {
      int wmId = LOWORD(wParam);
      // Разобрать выбор в меню:
      switch (wmId)
      {
```

```
        case IDM_POINT:
            SetWindowTextW(hWnd, L"Режим вводу точок");
            editor.Start(new PointShape);
            break;
        case IDM_LINE:
            SetWindowTextW(hWnd, L"Режим вводу ліній");
            editor.Start(new LineShape);
            break;
        case IDM_RECTANGLE:
            SetWindowTextW(hWnd, L"Режим вводу прямокутників");
            editor.Start(new RectShape);
            break;
        case IDM_ELLIPSE:
            SetWindowTextW(hWnd, L"Режим вводу еліпсів");
            editor.Start(new EllipseShape);
            break;
        case IDM_CUBE:
            SetWindowTextW(hWnd, L"Режим вводу кубів");
            editor.Start(new CubeShape);
            break;
        case IDM_OLINEO:
            SetWindowTextW(hWnd, L"Режим вводу ліній з кружечками");
            editor.Start(new OLineOShape);
            break;
        case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
        case IDM_EXIT:
            DestroyWindow(hWnd);
            break;
        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    }
    break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

// Обработчик сообщений для окна "О программе".
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM
lParam)
```

```
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

my_editor.h

```
#include "shape.h"

class MyEditor {
public:
        int x1, y1, x2, y2;
        ~MyEditor();
        void Start(Shape *);
        void OnLBdown(HWND);
        void OnLBup(HWND);
        void OnMouseMove(HWND);
        void OnPaint(HWND);
};
```

my_editor.cpp

```
#include "framework.h"
#include "my_editor.h"

int arrayCounter = 0;
static int check = 0;
Shape* pcshape[101];

MyEditor::~MyEditor() {
        for (int i = 0; i < arrayCounter; i++)
                delete pcshape[i];
}
```

```
void MyEditor::Start(Shape* shape) {
	pcshape[arrayCounter] = shape;
}

void MyEditor::OnLBdown(HWND hWnd) {
	POINT pt;
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	x1 = x2 = pt.x;
	y1 = y2 = pt.y;
	check = 1;
}

void MyEditor::OnLBup(HWND hWnd) {
	POINT pt;
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	x2 = pt.x;
	y2 = pt.y;
	check = 0;
	pcshape[arrayCounter]->Set(x1, y1, x2, y2);

	arrayCounter++;
	InvalidateRect(hWnd, NULL, TRUE);
	pcshape[arrayCounter] = pcshape[arrayCounter - 1]->Copy();
}

void MyEditor::OnMouseMove(HWND hWnd) {
	if (check) {
		POINT pt;
		HDC hdc = GetDC(hWnd);

		SetROP2(hdc, R2_NOTXORPEN);

		MoveToEx(hdc, x1, y1, NULL);
		pcshape[arrayCounter]->Set(x1, y1, x2, y2);
		pcshape[arrayCounter]->Show(hdc, 1);

		GetCursorPos(&pt);
		ScreenToClient(hWnd, &pt);

		x2 = pt.x;
		y2 = pt.y;

		MoveToEx(hdc, x1, y1, NULL);
		pcshape[arrayCounter]->Set(x1, y1, x2, y2);
```

```
                pcshape[arrayCounter]->Show(hdc, 1);

                ReleaseDC(hWnd, hdc);
        }
}

void MyEditor::OnPaint(HWND hWnd) {
        PAINTSTRUCT ps;
        HDC hdc;
        hdc = BeginPaint(hWnd, &ps);
        for (int i = 0; i < arrayCounter; i++) {
                if (pcshape[i]) pcshape[i]->Show(hdc, 0);
        }
        EndPaint(hWnd, &ps);
}
```

shape.h

```
#pragma once

class Shape {
protected:
        long xs1, ys1, xs2, ys2;
public:
        void Set(long x1, long y1, long x2, long y2);
        virtual void Show(HDC, bool) = 0;
        virtual Shape* Copy() = 0;
};
```

shape.cpp

```
#include "framework.h"
#include "shape.h"

void Shape::Set(long x1, long y1, long x2, long y2) {
        xs1 = x1;
        ys1 = y1;
        xs2 = x2;
        ys2 = y2;
}
```

toolbar.h

```
#pragma once
#include "framework.h"
#include <commctrl.h>
#pragma comment(lib, "comctl32.lib")
```

```
class Toolbar
{
private:
	HWND hWndToolBar;
public:
	void OnCreate(HWND);
	void OnSize(HWND);
	void OnNotify(HWND, LPARAM, WPARAM);
};
```

toolbar.cpp

```
#include "toolbar.h"
#include "lab2.h"

void Toolbar::OnSize(HWND hWnd)
{
   RECT rc, rw;
   if (hWndToolBar)
   {
      GetClientRect(hWnd, &rc);
      GetWindowRect(hWndToolBar, &rw);
      MoveWindow(hWndToolBar, 0, 0, rc.right - rc.left, rw.bottom - rw.top, FALSE);
   }
}

void Toolbar::OnCreate(HWND hWndmother)
{
   HINSTANCE hInst = (HINSTANCE)GetWindowLong(hWndmother, GWL_HINSTANCE);
   TBBUTTON tbb[6];
   ZeroMemory(tbb, sizeof(tbb));
   tbb[0].iBitmap = 0;
   tbb[0].fsState = TBSTATE_ENABLED;
   tbb[0].fsStyle = TBSTYLE_BUTTON;
   tbb[0].idCommand = IDM_POINT;

   tbb[1].iBitmap = 1;
   tbb[1].fsState = TBSTATE_ENABLED;
   tbb[1].fsStyle = TBSTYLE_BUTTON;
   tbb[1].idCommand = IDM_LINE;

   tbb[2].iBitmap = 2;
   tbb[2].fsState = TBSTATE_ENABLED;
   tbb[2].fsStyle = TBSTYLE_BUTTON;
   tbb[2].idCommand = IDM_RECTANGLE;

   tbb[3].iBitmap = 3;
   tbb[3].fsState = TBSTATE_ENABLED;
   tbb[3].fsStyle = TBSTYLE_BUTTON;
   tbb[3].idCommand = IDM_ELLIPSE;

   tbb[4].iBitmap = 4;
```

```
    tbb[4].fsState = TBSTATE_ENABLED;
    tbb[4].fsStyle = TBSTYLE_BUTTON;
    tbb[4].idCommand = IDM_CUBE;

    tbb[5].iBitmap = 5;
    tbb[5].fsState = TBSTATE_ENABLED;
    tbb[5].fsStyle = TBSTYLE_BUTTON;
    tbb[5].idCommand = IDM_OLINEO;

    SendMessage(hWndToolBar, TB_ADDBUTTONS, 6, (LPARAM)&tbb);
    hWndToolBar = CreateToolbarEx(hWndmother,
        WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP |
TBSTYLE_TOOLTIPS, IDC_TOOLBAR,
        6, hInst, IDB_TOOLBAR,
        tbb,
        6,
        24, 24, 24, 24,
        sizeof(TBBUTTON));
}

void Toolbar::OnNotify(HWND hWnd, LPARAM lParam, WPARAM wParam)
{
    LPNMHDR pnmh = (LPNMHDR)lParam;
    if (pnmh->code == TTN_NEEDTEXT)
    {
        LPTOOLTIPTEXT lpttt = (LPTOOLTIPTEXT)lParam;
        switch (lpttt->hdr.idFrom)
        {
        case IDM_POINT:
            lstrcpy(lpttt->szText, L"Точка");
            break;
        case IDM_LINE:
            lstrcpy(lpttt->szText, L"Лінія");
            break;
        case IDM_RECTANGLE:
            lstrcpy(lpttt->szText, L"Прямокутник");
            break;
        case IDM_ELLIPSE:
            lstrcpy(lpttt->szText, L"Еліпс");
            break;
        case IDM_CUBE:
            lstrcpy(lpttt->szText, L"Куб");
            break;
        case IDM_OLINEO:
            lstrcpy(lpttt->szText, L"Лінія з кружечками");
            break;
        default: lstrcpy(lpttt->szText, L"Нічого");
        }
    }
}
```

OlineO_shape.h

```
#pragma once
#include "line_shape.h"
#include "ellipse_shape.h"

class OLineOShape : public LineShape, public EllipseShape {
public:
	void Show(HDC, bool isDash);
	Shape* Copy();
};
```

OlineO_shape.cpp

```
#include "framework.h"
#include "OlineO_shape.h"

void OLineOShape::Show(HDC hdc, bool isDash) {
	long x1, y1, x2, y2;
	x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;
	LineShape::Show(hdc, isDash);

	EllipseShape::Set(x1 + 10, y1 + 10, x1 - 10, y1 - 10);
	EllipseShape::Show(hdc, isDash);
	EllipseShape::Set(x2 + 10, y2 + 10, x2 - 10, y2 - 10);
	EllipseShape::Show(hdc, isDash);

	LineShape::Set(x1, y1, x2, y2);
}

Shape* OLineOShape::Copy() {
	return new OLineOShape();
}
```

cube_shape.h

```
#pragma once
#include "rect_shape.h"
#include "line_shape.h"

class CubeShape : public RectShape, public LineShape {
public:
	void Show(HDC, bool isDash);
	Shape* Copy();
};
```

cube_shape.cpp

```
#include "framework.h"
#include "cube_shape.h"

void CubeShape::Show(HDC hdc, bool isDash) {
	long x1, x2, y1, y2;
	x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;
```

```
	RectShape::Set(x1 - 75, y1 - 75, x1 + 75, y1 + 75);
	RectShape::Show(hdc, isDash);
	RectShape::Set(x2 - 75, y2 - 75, x2 + 75, y2 + 75);
	RectShape::Show(hdc, isDash);

	LineShape::Set(x1 - 75, y1 - 75, x2 - 75, y2 - 75);
	LineShape::Show(hdc, isDash);
	LineShape::Set(x1 - 75, y1 + 75, x2 - 75, y2 + 75);
	LineShape::Show(hdc, isDash);
	LineShape::Set(x1 + 75, y1 + 75, x2 + 75, y2 + 75);
	LineShape::Show(hdc, isDash);
	LineShape::Set(x1 + 75, y1 - 75, x2 + 75, y2 - 75);
	LineShape::Show(hdc, isDash);

	LineShape::Set(x1, y1, x2, y2);
}

Shape* CubeShape::Copy() {
	return new CubeShape();
}
```

ellipse_shape.cpp

```
#include "framework.h"
#include "ellipse_shape.h"


void EllipseShape::Show(HDC hdc, bool isDash) {
	HPEN hPen, hPenOld;
	HBRUSH hBrush, hBrushOld;
	if (isDash) {
		hPen = CreatePen(PS_DOT, 1, RGB(255, 0, 0));
		hPenOld = (HPEN)SelectObject(hdc, hPen);
		Arc(hdc, xs1, ys1, xs2, ys2, 0, 0, 0, 0);
		SelectObject(hdc, hPenOld);
		DeleteObject(hPen);
	}
	else {
		hBrush = CreateSolidBrush(RGB(255, 255, 255));
		hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);
		SelectObject(hdc, hBrush);
		Ellipse(hdc, xs1, ys1, xs2, ys2);
		SelectObject(hdc, hBrushOld);
		DeleteObject(hBrush);
	}
};

Shape* EllipseShape::Copy() {
	return new EllipseShape();
}
```

line_shape.cpp

```
#include "framework.h"
#include "line_shape.h"

void LineShape::Show(HDC hdc, bool isDash) {
	HPEN hPen, hPenOld;
	if (isDash) {
		hPen = CreatePen(PS_DOT, 1, RGB(255, 0, 0));
		hPenOld = (HPEN)SelectObject(hdc, hPen);
	}
	else {
		hPen = CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
		hPenOld = (HPEN)SelectObject(hdc, hPen);
	}

	MoveToEx(hdc, xs1, ys1, NULL);
	LineTo(hdc, xs2, ys2);
	SelectObject(hdc, hPenOld);
	DeleteObject(hPen);
}

Shape* LineShape::Copy() {
	return new LineShape();
}
```

point_shape.cpp

```
#include "framework.h"
#include "point_shape.h"

void PointShape::Show(HDC hdc, bool isDash) {
	SetPixel(hdc, xs1, ys1, RGB(0, 0, 0));
}

Shape* PointShape::Copy() {
	return new PointShape();
}
```

rect_shape.cpp

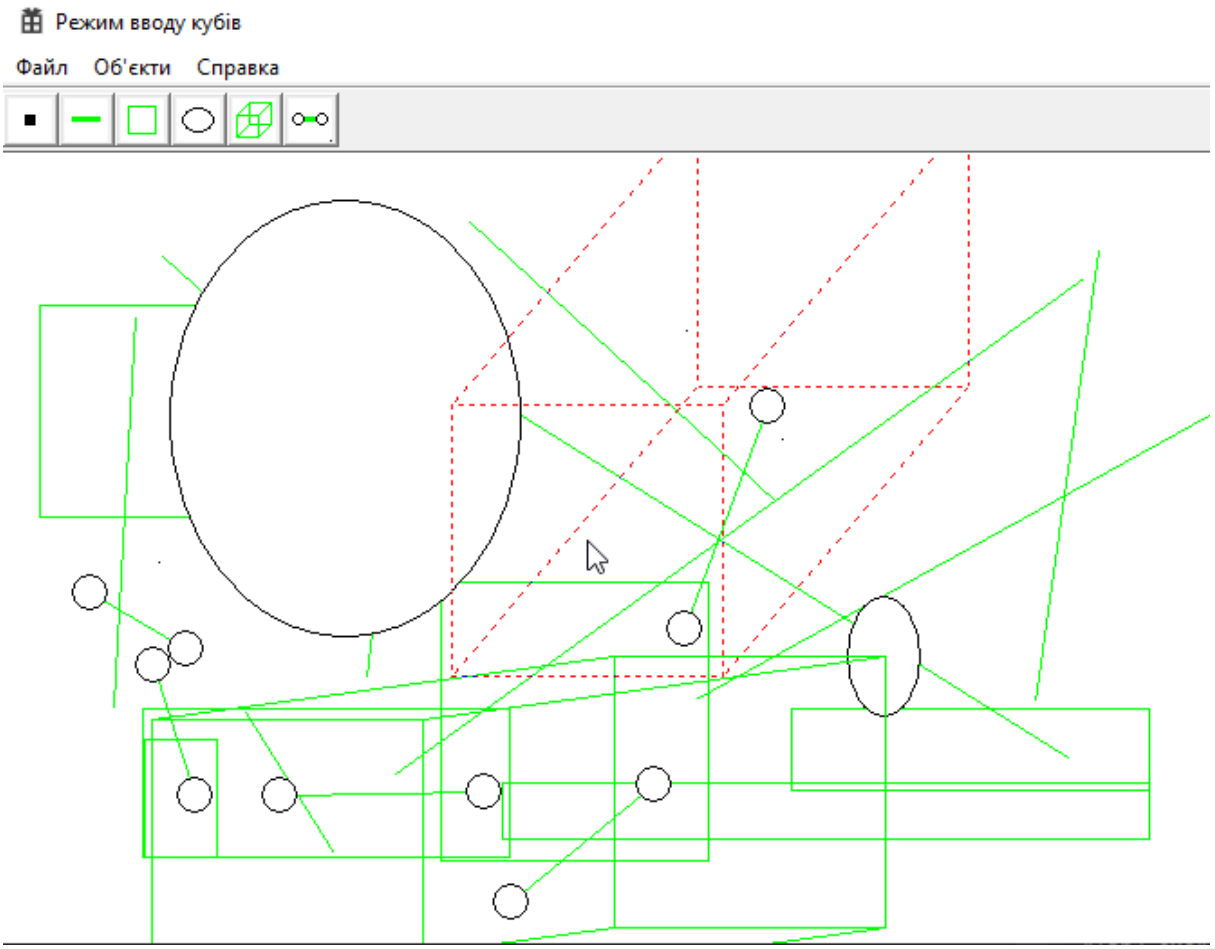
```
#include "framework.h"
#include "rect_shape.h"

void RectShape::Show(HDC hdc, bool isDash) {
	HPEN hPen, hPenOld;
	if (isDash) {
		hPen = CreatePen(PS_DOT, 1, RGB(255, 0, 0));
		hPenOld = (HPEN)SelectObject(hdc, hPen);
		MoveToEx(hdc, xs1, ys1, NULL);
		LineTo(hdc, xs1, ys2);
		LineTo(hdc, xs2, ys2);
```

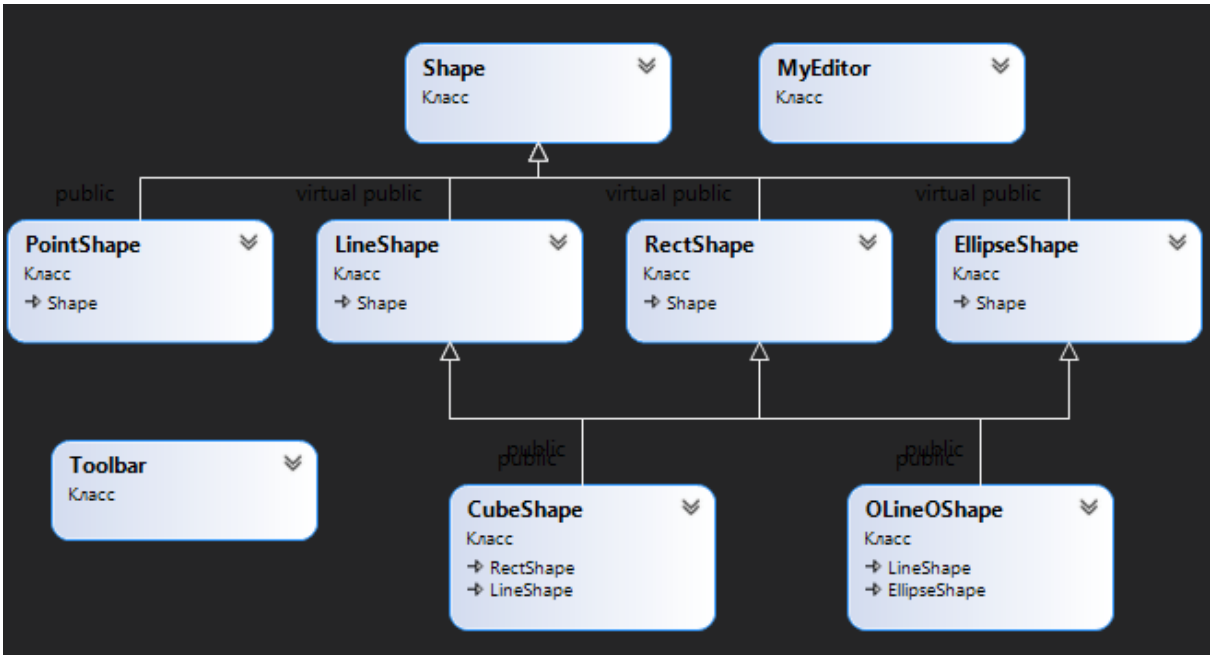
```
                LineTo(hdc, xs2, ys1);
                LineTo(hdc, xs1, ys1);
        }
        else {
                hPen = CreatePen(PS_SOLID, 1, RGB(0, 255, 0));
                hPenOld = (HPEN)SelectObject(hdc, hPen);
                MoveToEx(hdc, xs1, ys1, NULL);
                LineTo(hdc, xs1, ys2);
                LineTo(hdc, xs2, ys2);
                LineTo(hdc, xs2, ys1);
                LineTo(hdc, xs1, ys1);
        }
        SelectObject(hdc, hPenOld);
        DeleteObject(hPen);
}

Shape* RectShape::Copy() {
        return new RectShape();
}
```

Результати тестування програми:



Діаграма класів:



Висновки:

Виконуючи четверту лабораторну роботу я продовжив розробку графічного редактора та познайомився з множинним успадкуванням на C++.