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
«ООП»

Виконав:

Студент 2-го курсу групи ІМ-13
Нестеров Дмитро Васильович
номер у списку групи: 17

Перевірив:

Порєв Віктор Миколайович

Київ 2022

Мета:

Мета роботи – отримати вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для забезпечення зручного додавання нових типів об'єктів.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab4.

2. Написати вихідний текст програми згідно варіанту завдання.

3. Скомпілювати вихідний текст і отримати виконуваний файл програми.

4. Перевірити роботу програми. Налагодити програму.

5. Проаналізувати та прокоментувати результати та вихідний текст програми.

6. Оформити звіт.

Вихідні тексти файлів:

Lab2.h

```
#pragma once

#include "resource.h"
```

Lab2.cpp

```
#include "framework.h"
#include "Lab2.h"
#include "my_editor.h"
#include "toolbar.h"
#include "PointShape.h"
#include "LineShape.h"
#include "RectShape.h"
#include "EllipseShape.h"
#include "CubeShape.h"
#include "LineOOShape.h"


#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
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)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB2, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // Perform application initialization:
    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

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

    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int)msg.wParam;
}


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);
}
```

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance; // Store instance handle in our global variable

    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;
}
MyEditor editor;


LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
        editor.OnCreate(hWnd, hInst); //тут створимо Toolbar
        break;
    case WM_SIZE:
        editor.OnSize(hWnd);
        break;
    case WM_NOTIFY:
        editor.OnNotify(hWnd, wParam, lParam);
        break;
    case WM_LBUTTONDOWN: //натиснуто ліву кнопку миші у клієнтській частині вікна
        editor.OnLBdown(hWnd);
        break;
    case WM_LBUTTONUP: //відпущено ліву кнопку миші у клієнтській частині вікна
        editor.OnLBup(hWnd);
        break;
    case WM_MOUSEMOVE: //пересунуто мишу у клієнтській частині вікна
        editor.OnMouseMove(hWnd);
        break;
    case WM_PAINT: //потрібно оновлення зображення клієнтської частині вікна
        editor.OnPaint(hWnd);
        break;
    case WM_INITMENUPOPUP: //позначка пунктів меню - для окремих варіантів завдань
        editor.OnInitMenuPopup(hWnd, wParam);
        break;
    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        // Parse the menu selections:
        switch (wmId)
        {
        case ID_TOOL_POINT:
        case ID_32771:
            editor.Start(hWnd, new PointShape, wmId);
            break;
        case ID_TOOL_LINE:
        case ID_32772:
            editor.Start(hWnd, new LineShape, wmId);
```

```
            break;
        case ID_TOOL_RECT:
        case ID_32773:
            editor.Start(hWnd, new RectShape, wmId);
            break;
        case ID_TOOL_ELLIPSE:
        case ID_32774:
            editor.Start(hWnd, new EllipseShape, wmId);
            break;
        case ID_TOOL_FIGURE_CUBE:
        case ID_32775:
            editor.Start(hWnd, new CubeShape, wmId);
            break;
        case ID_TOOL_FIGURE_LineOO:
        case ID_32776:
            editor.Start(hWnd, new LineOOShape, wmId);
            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;
}

// Message handler for about box.
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;
}
```

toolbar.h

```
#pragma once
#pragma comment(lib, "comctl32.lib")
class ToolBar {
protected:
	HWND hwndToolBar = NULL;
	LPARAM oldlParam = NULL;
public:
	ToolBar(void);
	void OnCreate(HWND, HINSTANCE);
	void OnSize(HWND);
	void OnTool(HWND, LPARAM);
	void OnNotify(HWND, WPARAM, LPARAM);
};
```

toolbar.cpp

```
#include "framework.h"
#include "toolbar_resource.h"
#include "toolbar.h"
#include "resource.h"
#include <commctrl.h>


ToolBar::ToolBar(void) {}

void ToolBar::OnCreate(HWND hWnd, HINSTANCE hInst)
{
    TBBUTTON tbb[6]; //для Toolbar з чотирма кнопками
    ZeroMemory(tbb, sizeof(tbb));
    tbb[0].iBitmap = 0; //стандартне зображення
    tbb[0].fsState = TBSTATE_ENABLED;
    tbb[0].fsStyle = TBSTYLE_BUTTON; //тип елементу - кнопка
    tbb[0].idCommand = ID_TOOL_POINT; //цей ID буде у повідомленні WM_COMMAND

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

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

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

    tbb[4].iBitmap = 4;
    tbb[4].fsState = TBSTATE_ENABLED;
    tbb[4].fsStyle = TBSTYLE_BUTTON;
    tbb[4].idCommand = ID_TOOL_FIGURE_LineOO;

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

```
    hwndToolBar = CreateToolbarEx(hWnd,
         WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP | TBSTYLE_TOOLTIPS,
         IDC_MY_TOOLBAR, 4, hInst, IDB_BITMAP1, tbb,
         6, 24, 24, 24, 24, sizeof(TBBUTTON));
}

void ToolBar::OnSize(HWND hWnd)
{
    RECT rc, rw;
    if (hwndToolBar)
    {
        GetClientRect(hWnd, &rc); //нові розміри головного вікна
        GetWindowRect(hwndToolBar, &rw); //нам потрібно знати висоту Toolbar
        MoveWindow(hwndToolBar, 0, 0, rc.right - rc.left, rw.bottom - rw.top, FALSE);
    }
}

void ToolBar::OnTool(HWND hWnd, LPARAM lParam)
{
    if (oldlParam) SendMessage(hwndToolBar, TB_PRESSBUTTON, oldlParam, 0); //release old
button
    SendMessage(hwndToolBar, TB_PRESSBUTTON, lParam, 1); // press new button
    oldlParam = lParam;
}

void ToolBar::OnNotify(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
    LPNMHDR pnmh = (LPNMHDR)lParam;
    if (pnmh->code == TTN_NEEDTEXT)
    {
        LPTOOLTIPTEXT lpttt = (LPTOOLTIPTEXT)lParam;
        switch (lpttt->hdr.idFrom)
        {

        case ID_TOOL_POINT:
            lstrcpy(lpttt->szText, L"Точка");
            break;

        case ID_TOOL_LINE:
            lstrcpy(lpttt->szText, L"Лінія");
            break;

        case ID_TOOL_RECT:
            lstrcpy(lpttt->szText, L"Прямокутник");
            break;

        case ID_TOOL_ELLIPSE:
            lstrcpy(lpttt->szText, L"Еліпс");
            break;
        case ID_TOOL_FIGURE_LineOO:
            lstrcpy(lpttt->szText, L"Лінія з кружечками");
            break;
        case ID_TOOL_FIGURE_CUBE:
            lstrcpy(lpttt->szText, L"Куб");
            break;

        default: lstrcpy(lpttt->szText, L"Щось невідоме");
        }
    }
```

```
}
```

my_editor.h

```
#pragma once
#include "shape.h"
#include "ToolBar.h"
#define shape_size 117

class MyEditor
{
private:
	static Shape* pcshape[shape_size];
	static int nCurrentIndex;
	ToolBar toolbar;
	Shape* currShape;
	BOOL isPainting = false;
	POINT pt;
public:
	void Start(HWND, Shape*, LPARAM);
	void OnLBdown(HWND);
	void OnLBup(HWND);
	void OnMouseMove(HWND);
	void OnPaint(HWND);
	void OnInitMenuPopup(HWND, WPARAM);
	void OnCreate(HWND, HINSTANCE);
	void OnNotify(HWND, WPARAM, LPARAM);
	void OnSize(HWND);
};
```

my_editor.cpp

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

Shape* MyEditor::pcshape[shape_size];
int MyEditor::nCurrentIndex = 0;

void MyEditor::Start(HWND hWnd, Shape* shape, LPARAM lParam)
{
	currShape = shape;
	toolbar.OnTool(hWnd, lParam);
}

void MyEditor::OnLBdown(HWND hWnd)
{
	if (!currShape) return;
	isPainting = true;
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	pcshape[nCurrentIndex] = currShape->copyShape();
	pcshape[nCurrentIndex]->SetStart(pt.x, pt.y);
	pcshape[nCurrentIndex]->SetEnd(pt.x, pt.y);
}

void MyEditor::OnLBup(HWND hWnd)
{
	if (!isPainting || !currShape) return;
	isPainting = false;
	GetCursorPos(&pt);
```

```
	ScreenToClient(hWnd, &pt);
	pcshape[nCurrentIndex]->SetEnd(pt.x, pt.y);
	nCurrentIndex = (++nCurrentIndex) % shape_size;
	InvalidateRect(hWnd, NULL, TRUE);
}

void MyEditor::OnMouseMove(HWND hWnd)
{
	if (!isPainting || !currShape) return;
	pcshape[nCurrentIndex]->DrawRubberBand(hWnd);
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	pcshape[nCurrentIndex]->SetEnd(pt.x, pt.y);
	pcshape[nCurrentIndex]->DrawRubberBand(hWnd);

}

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

void MyEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	if (currShape) currShape->OnInitMenuPopup(hWnd, wParam);
}

void MyEditor::OnCreate(HWND hWnd, HINSTANCE hInst)
{
	toolbar.OnCreate(hWnd, hInst);
}

void MyEditor::OnNotify(HWND hWnd, WPARAM wParam, LPARAM lParam)
{
	toolbar.OnNotify(hWnd, wParam, lParam);
}

void MyEditor::OnSize(HWND hWnd)
{
	toolbar.OnSize(hWnd);
}
```

PointShape.h

```
#pragma once
#include "shape.h"

class PointShape : virtual public Shape
{
public:
	void Show(HDC);
	void DrawRubberBand(HWND);
	Shape* copyShape();
	void OnInitMenuPopup(HWND, WPARAM);
};
```

PointShape.cpp

```
#include "PointShape.h"

void PointShape::Show(HDC hdc)
{
	SetPixel(hdc, xs1, ys1, RGB(255, 0, 0));
}

void PointShape::DrawRubberBand(HWND hWnd) { }

Shape* PointShape::copyShape() { return new PointShape; }

void PointShape::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	HMENU hMenu, hSubMenu;
	hMenu = GetMenu(hWnd);
	hSubMenu = GetSubMenu(hMenu, 1);
	if ((HMENU)wParam == hSubMenu)
	{
		CheckMenuItem(hSubMenu, ID_32771, MF_CHECKED);
		CheckMenuItem(hSubMenu, ID_32772, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32773, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32774, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32775, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32776, MF_UNCHECKED);
	}
}
```

LineShape.h

```
#pragma once
#include "shape.h"

class LineShape : virtual public Shape {
public:
	virtual void Show(HDC);
	void DrawRubberBand(HWND);
	Shape* copyShape();
	void OnInitMenuPopup(HWND, WPARAM);
};
```

LineShape.cpp

```
#include "LineShape.h"

void LineShape::Show(HDC hdc)
{
	MoveToEx(hdc, xs1, ys1, NULL);
	LineTo(hdc, xs2, ys2);
}

void LineShape::DrawRubberBand(HWND hWnd)
{
	HPEN hPenOld, hPen;
	HDC hdc;
	hdc = GetDC(hWnd);
	SetROP2(hdc, R2_NOTXORPEN);
	hPen = CreatePen(PS_DOT, 1, 0);
	hPenOld = (HPEN)SelectObject(hdc, hPen);
```

```
	LineShape::Show(hdc);

	SelectObject(hdc, hPenOld);
	DeleteObject(hPen);
	ReleaseDC(hWnd, hdc);
}

Shape* LineShape::copyShape() { return new LineShape; }

void LineShape::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	HMENU hMenu, hSubMenu;
	hMenu = GetMenu(hWnd);
	hSubMenu = GetSubMenu(hMenu, 1);
	if ((HMENU)wParam == hSubMenu)
	{
		CheckMenuItem(hSubMenu, ID_32771, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32772, MF_CHECKED);
		CheckMenuItem(hSubMenu, ID_32773, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32774, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32775, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32776, MF_UNCHECKED);
	}
}
```

RectShape.h

```
#pragma once
#include "shape.h"

class RectShape : virtual public Shape {
public:
	void Show(HDC);
	void DrawRubberBand(HWND);
	Shape* copyShape();
	void OnInitMenuPopup(HWND, WPARAM);
};
```

RectShape.cpp

```
#include "RectShape.h"

void RectShape::Show(HDC hdc)
{
	HBRUSH hBrush, hBrushOld;

	hBrush = (HBRUSH)CreateSolidBrush(RGB(192, 192, 192)); //створюється пензль
	hBrushOld = (HBRUSH)SelectObject(hdc, hBrush); //пензль -> у hdc
	MoveToEx(hdc, xs1, ys1, NULL);
	LineTo(hdc, xs1, ys2);
	LineTo(hdc, xs2, ys2);
	LineTo(hdc, xs2, ys1);
	LineTo(hdc, xs1, ys1);
}

void RectShape::DrawRubberBand(HWND hWnd)
{
	HPEN hPenOld, hPen;
	HDC hdc;
```

```
        hdc = GetDC(hWnd);
        SetROP2(hdc, R2_NOTXORPEN);
        hPen = CreatePen(PS_DOT, 1, 0);
        hPenOld = (HPEN)SelectObject(hdc, hPen);

        RectShape::Show(hdc);

        SelectObject(hdc, hPenOld);
        DeleteObject(hPen);
        ReleaseDC(hWnd, hdc);

}

Shape* RectShape::copyShape() { return new RectShape; }

void RectShape::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
        HMENU hMenu, hSubMenu;
        hMenu = GetMenu(hWnd);
        hSubMenu = GetSubMenu(hMenu, 1);
        if ((HMENU)wParam == hSubMenu)
        {
                CheckMenuItem(hSubMenu, ID_32771, MF_UNCHECKED);
                CheckMenuItem(hSubMenu, ID_32772, MF_UNCHECKED);
                CheckMenuItem(hSubMenu, ID_32773, MF_CHECKED);
                CheckMenuItem(hSubMenu, ID_32774, MF_UNCHECKED);
                CheckMenuItem(hSubMenu, ID_32775, MF_UNCHECKED);
                CheckMenuItem(hSubMenu, ID_32776, MF_UNCHECKED);
        }
}
```

EllipseShape.h

```
#pragma once
#include "shape.h"

class EllipseShape : virtual public Shape {
public:
        virtual void Show(HDC);
        void DrawRubberBand(HWND);
        Shape* copyShape();
        void OnInitMenuPopup(HWND, WPARAM);
};
```

EllipseShape.cpp

```
#include "EllipseShape.h"

void EllipseShape::Show(HDC hdc)
{
    long xStart = xs2 - 2 * (xs2 - xs1);
    long yStart = ys2 - 2 * (ys2 - ys1);

    HBRUSH hBrush, hBrushOld;
    hBrush = (HBRUSH)CreateSolidBrush(RGB(192, 192, 192));
    hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);

    Ellipse(hdc, xStart, yStart, xs2, ys2);
    SelectObject(hdc, hBrushOld);
    DeleteObject(hBrush);
```

```
}

void EllipseShape::DrawRubberBand(HWND hWnd)
{
    HPEN hPenOld, hPen;
    HDC hdc;
    hdc = GetDC(hWnd);
    SetROP2(hdc, R2_NOTXORPEN);
    hPen = CreatePen(PS_DOT, 1, 0);
    hPenOld = (HPEN)SelectObject(hdc, hPen);

    int sizeX = xs1 - xs2;
    int sizeY = ys1 - ys2;
    Ellipse(hdc, xs1 - sizeX, ys1 - sizeY, xs1 + sizeX, ys1 + sizeY);

    SelectObject(hdc, hPenOld);
    DeleteObject(hPen);
    ReleaseDC(hWnd, hdc);
}

Shape* EllipseShape::copyShape() { return new EllipseShape; }

void EllipseShape::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
    HMENU hMenu, hSubMenu;
    hMenu = GetMenu(hWnd);
    hSubMenu = GetSubMenu(hMenu, 1);
    if ((HMENU)wParam == hSubMenu)
    {
        CheckMenuItem(hSubMenu, ID_32771, MF_UNCHECKED);
        CheckMenuItem(hSubMenu, ID_32772, MF_UNCHECKED);
        CheckMenuItem(hSubMenu, ID_32773, MF_UNCHECKED);
        CheckMenuItem(hSubMenu, ID_32774, MF_CHECKED);
        CheckMenuItem(hSubMenu, ID_32775, MF_UNCHECKED);
        CheckMenuItem(hSubMenu, ID_32776, MF_UNCHECKED);
    }
}
```

LineOOShape.h

```
#pragma once
#include "LineShape.h"
#include "EllipseShape.h"

class LineOOShape : public LineShape, public EllipseShape
{
public:
       virtual void Show(HDC);
       void DrawRubberBand(HWND);
       Shape* copyShape();
       void OnInitMenuPopup(HWND, WPARAM);
};
```

LineOOShape.cpp

```
#include "LineOOShape.h"

void LineOOShape::Show(HDC hdc)
```

```
{
	long x1, x2, y1, y2;

	x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

	LineShape::SetStart(x1 - 10, y1 - 10);
	LineShape::SetEnd(x2 - 10, y2 - 10);
	LineShape::Show(hdc);

	EllipseShape::SetStart(x1 - 10, y1 - 10);
	EllipseShape::SetEnd(x1 + 10, y1 + 10);

	EllipseShape::Show(hdc);

	EllipseShape::SetStart(x2 - 10, y2 - 10);
	EllipseShape::SetEnd(x2 + 10, y2 + 10);
	EllipseShape::Show(hdc);

	xs1 = x1; ys1 = y1; xs2 = x2; ys2 = y2;
}

void LineOOShape::DrawRubberBand(HWND hWnd)
{
	HDC hdc;
	hdc = GetDC(hWnd);

	long x1, x2, y1, y2;

	x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

	LineShape::SetStart(x1 - 10, y1 - 10);
	LineShape::SetEnd(x2 - 10, y2 - 10);
	LineShape::DrawRubberBand(hWnd);

	EllipseShape::SetStart(x1 - 10, y1 - 10);
	EllipseShape::SetEnd(x1 + 10, y1 + 10);
	EllipseShape::DrawRubberBand(hWnd);

	EllipseShape::SetStart(x2 - 10, y2 - 10);
	EllipseShape::SetEnd(x2 + 10, y2 + 10);
	EllipseShape::DrawRubberBand(hWnd);

	xs1 = x1; ys1 = y1; xs2 = x2; ys2 = y2;
}

Shape* LineOOShape::copyShape() { return new LineOOShape; }

void LineOOShape::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	HMENU hMenu, hSubMenu;
	hMenu = GetMenu(hWnd);
	hSubMenu = GetSubMenu(hMenu, 1);
	if ((HMENU)wParam == hSubMenu)
	{
		CheckMenuItem(hSubMenu, ID_32771, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32772, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32773, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32774, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32775, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32776, MF_CHECKED);
	}
```

```
}
```

CubeShape.cpp

```
#pragma once
#include "LineShape.h"
#include "RectShape.h"

class CubeShape : public LineShape, public RectShape
{
public:
	virtual void Show(HDC);
	void DrawRubberBand(HWND);
	Shape* copyShape();
	void OnInitMenuPopup(HWND, WPARAM);
};
```

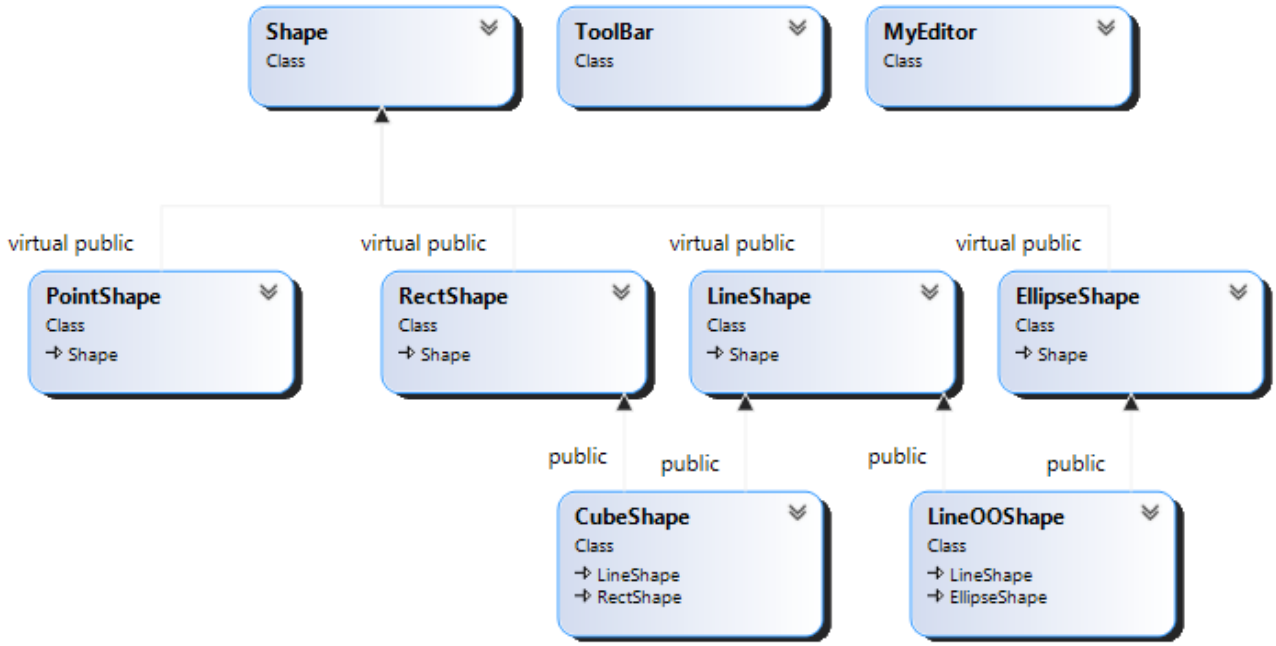
CubeShape.cpp

```
#include "CubeShape.h"

void CubeShape::Show(HDC hdc)
{
	long x1, x2, y1, y2;

	RectShape::Show(hdc);

	x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

	LineShape::SetStart(x1, y1);
	LineShape::SetEnd( x1 + 30, y1 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x1 + 30, y1 - 30);
	LineShape::SetEnd(x2 + 30, y1 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x2, y1);
	LineShape::SetEnd(x2 + 30, y1 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x2, y2);
	LineShape::SetEnd(x2 + 30, y2 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x2 + 30, y1 - 30);
	LineShape::SetEnd(x2 + 30, y2 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x1, y2);
	LineShape::SetEnd(x1 + 30, y2 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x1 + 30, y2 - 30);
	LineShape::SetEnd(x2 + 30, y2 - 30);
	LineShape::Show(hdc);

	LineShape::SetStart(x1 + 30, y1 - 30);
	LineShape::SetEnd(x1 + 30, y2 - 30);
```

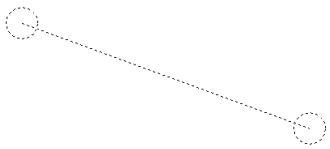
```
	LineShape::Show(hdc);

	xs1 = x1; ys1 = y1; xs2 = x2; ys2 = y2;
}


void CubeShape::DrawRubberBand(HWND hWnd)
{
	HDC hdc;
	hdc = GetDC(hWnd);

	long x1, x2, y1, y2;

	RectShape::DrawRubberBand(hWnd);

	x1 = xs1; y1 = ys1; x2 = xs2; y2 = ys2;

	LineShape::SetStart(x1, y1);
	LineShape::SetEnd(x1 + 30, y1 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x1 + 30, y1 - 30);
	LineShape::SetEnd(x2 + 30, y1 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x2, y1);
	LineShape::SetEnd(x2 + 30, y1 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x2, y2);
	LineShape::SetEnd(x2 + 30, y2 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x2 + 30, y1 - 30);
	LineShape::SetEnd(x2 + 30, y2 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x1, y2);
	LineShape::SetEnd(x1 + 30, y2 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x1 + 30, y2 - 30);
	LineShape::SetEnd(x2 + 30, y2 - 30);
	LineShape::DrawRubberBand(hWnd);
	LineShape::SetStart(x1 + 30, y1 - 30);
	LineShape::SetEnd(x1 + 30, y2 - 30);
	LineShape::DrawRubberBand(hWnd);

	xs1 = x1; ys1 = y1; xs2 = x2; ys2 = y2;
}

Shape* CubeShape::copyShape() { return new CubeShape; }

void CubeShape::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	HMENU hMenu, hSubMenu;
	hMenu = GetMenu(hWnd);
	hSubMenu = GetSubMenu(hMenu, 1);
	if ((HMENU)wParam == hSubMenu)
	{
		CheckMenuItem(hSubMenu, ID_32771, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32772, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32773, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32774, MF_UNCHECKED);
		CheckMenuItem(hSubMenu, ID_32775, MF_CHECKED);
		CheckMenuItem(hSubMenu, ID_32776, MF_UNCHECKED);
	}
```

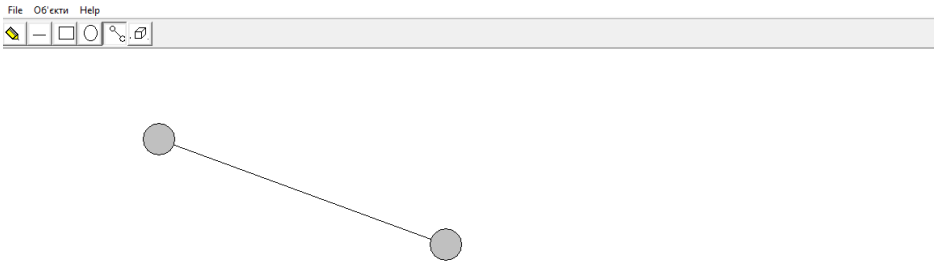
}

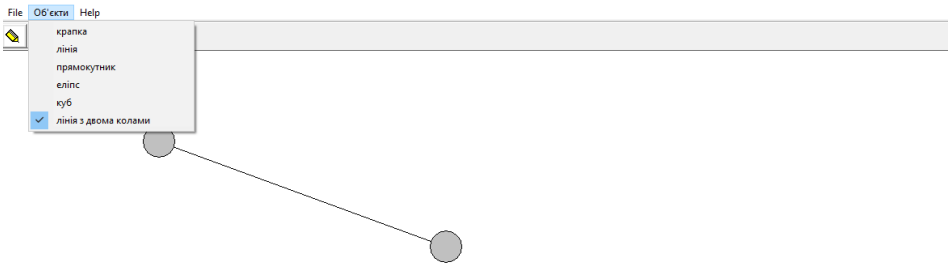
Діаграма класів

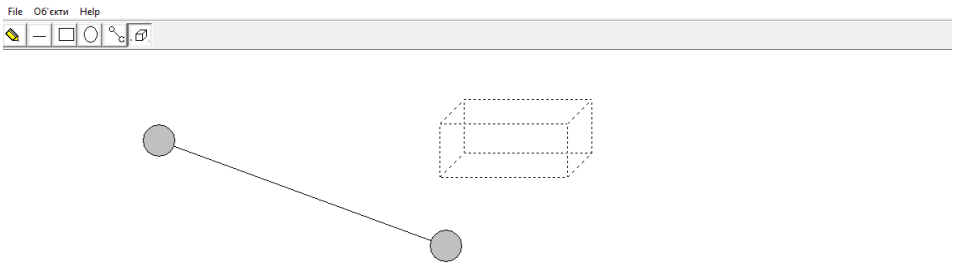


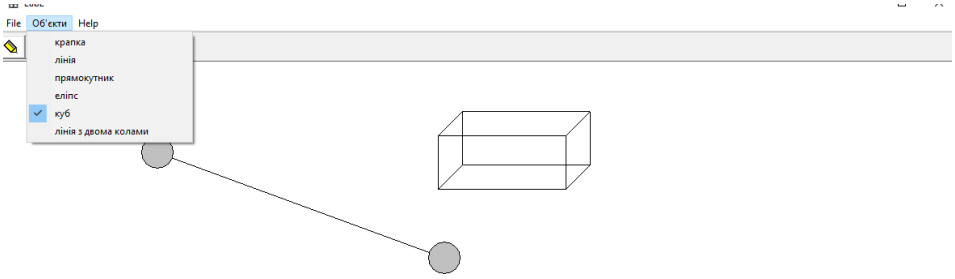
Скріншоти











Висновки. Під час виконання лабораторної роботи №4 я отримав вміння та навички проектування класів, виконавши модернізацію коду графічного редактора в об’єктно-орієнтованому стилі для забезпечення зручного додавання нових типів об'єктів. В результаті роботи я додав нові фігури, використам множинне успадкування та полегшив процес додавання нових фігур шляхом

змінення структури класів. Крім того, засобами Visual Studio C++ я створив діаграму класів, яку було додано до звіту.