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 №5

з дисципліни
«ООП»

Виконав:	Перевірив:
Студент 2-го курсу групи ІМ-13	Порєв Віктор Миколайович
Нестеров Дмитро Васильович	
номер у списку групи: 17	

Київ 2022

Мета:

Мета роботи – отримати вміння та навички програмувати багато віконний інтерфейс програми на С++ в об’єктно-орієнтованому стилі.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab5.

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.

17 варіант – сінглтон Меєрса

Вихідні тексти файлів:

Lab5.h

```
#pragma once

#include "resource.h"
```

Lab5.cpp

```

#include "framework.h"
#include "Lab5.h"
#include "my_editor.h"
#include "my_table.h"
#include <commctrl.h>
#pragma comment(lib, "comctl32.lib")
#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);

MyEditor& editor = editor.getInstance();
MyTable& table = table.getInstance();

int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                     _In_opt_ HINSTANCE hPrevInstance,
                     _In_ LPWSTR    lpCmdLine,
                     _In_ int       nCmdShow)
```

```
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    InitCommonControls();
    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB5, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

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

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_LAB5));

    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;
}



//
//  FUNCTION: MyRegisterClass()
//
//  PURPOSE: Registers the window class.
//
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_LAB5));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_LAB5);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
```

```
}

//
//   FUNCTION: InitInstance(HINSTANCE, int)
//
//   PURPOSE: Saves instance handle and creates main window
//
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
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;
}

//
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_LBUTTONDOWN:
        editor.OnLBdown(hWnd);
        break;
    case WM_LBUTTONUP:
        table.Add(editor.OnLBup(hWnd));
        break;
    case WM_MOUSEMOVE:
        editor.OnMouseMove(hWnd);
        break;
    case WM_PAINT:
        editor.RemoveItem(table.getRemove() - 1);
        table.setRemove(0);
        editor.OnPaint(hWnd, table.getSelected() - 1);
        break;

    case WM_CREATE:
        editor.OnCreate(hWnd, hInst);
        table.OnCreate(hWnd, hInst);
        break;
    case WM_SIZE:
        editor.OnSize(hWnd);
        break;
    case WM_NOTIFY:
        editor.OnNotify(hWnd, wParam, lParam);
        break;

    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);

            switch (wmId)
            {

            case ID_TOOL_POINT:
            case IDM_POINT:
                editor.Start(new PointShape, hWnd,  ID_TOOL_POINT, IDM_POINT, lParam);
                break;

            case ID_TOOL_LINE:
            case IDM_LINE:
                editor.Start(new LineShape, hWnd, ID_TOOL_LINE, IDM_LINE, lParam);
                break;

            case ID_TOOL_RECT:
            case IDM_RECT:
                editor.Start(new RectShape, hWnd, ID_TOOL_RECT, IDM_RECT, lParam);
                break;

            case ID_TOOL_ELLIPSE:
            case IDM_ELLIPSE:
                editor.Start(new EllipseShape, hWnd, ID_TOOL_ELLIPSE, IDM_ELLIPSE,
lParam);
                break;

            case ID_TOOL_LINEOO:
            case IDM_LINEOO:
                editor.Start(new LineOOShape, hWnd, ID_TOOL_LINEOO, IDM_LINEOO,
lParam);
                break;

            case ID_TOOL_CUBE:
            case IDM_CUBE:
                editor.Start(new CubeShape, hWnd, ID_TOOL_CUBE, IDM_CUBE, lParam);
                break;

            case IDM_TABLE:
            case ID_TABLE:
                table.Show();
                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;
}
```

Tool_bar.h

```
#pragma once
#pragma comment(lib, "comctl32.lib")
class ToolBar       {
protected:
       HWND hwndToolBar;
       LPARAM oldlParam;
public:
public:
       static ToolBar& getInstance()
       {
              static ToolBar instance;
              return instance;
       }
       ToolBar(void);
       void OnCreate(HWND, HINSTANCE);
       void OnSize(HWND);
       void OnNotify(HWND, WPARAM, LPARAM);
       void OnToolMove(HWND, LPARAM);
};
```

tool_bar.cpp

```
#include "framework.h"
#include "tool_bar.h"
#include "resource.h"
#include "framework.h"
```

```
#include <commctrl.h>

ToolBar::ToolBar(void) {}

void ToolBar::OnCreate(HWND hWnd, HINSTANCE hInst)
{
	SendMessage(hwndToolBar, TB_BUTTONSTRUCTSIZE, (WPARAM)sizeof(TBBUTTON), 0);

	TBBUTTON tbb[8];
	TBADDBITMAP tbab;

	tbab.hInst = HINST_COMMCTRL;
	tbab.nID = IDB_STD_SMALL_COLOR;
	SendMessage(hwndToolBar, TB_ADDBITMAP, 0, (LPARAM)&tbab);

	ZeroMemory(tbb, sizeof(tbb));
	tbb[0].iBitmap = 0;
	tbb[0].fsState = TBSTATE_ENABLED;
	tbb[0].fsStyle = TBSTYLE_BUTTON;
	tbb[0].idCommand = ID_TOOL_POINT;

	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_LINEOO;

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

	tbb[6].iBitmap = 0;
	tbb[6].fsState = TBSTATE_ENABLED;
	tbb[6].fsStyle = TBSTYLE_SEP;
	tbb[6].idCommand = 0;

	tbb[7].iBitmap = 6;
	tbb[7].fsState = TBSTATE_ENABLED;
	tbb[7].fsStyle = TBSTYLE_BUTTON;
	tbb[7].idCommand = ID_TABLE;

	SendMessage(hwndToolBar, TB_ADDBUTTONS, 6, (LPARAM)&tbb);

	hwndToolBar = CreateToolbarEx(hWnd,
		WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP |
TBSTYLE_TOOLTIPS,
            IDC_MY_TOOLBAR,
            7,
            hInst,
            IDB_BITMAP1,
            tbb,
            8,
            24, 24, 24, 24,
            sizeof(TBBUTTON));
}

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::OnToolMove(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"Крапка");
                  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_LINEOO:
                  lstrcpy(lpttt->szText, L"Гантеля");
                  break;
            case ID_TOOL_CUBE:
                  lstrcpy(lpttt->szText, L"Куб");
                  break;
            case ID_TABLE:
                  lstrcpy(lpttt->szText, L"Таблиця об'єктів");
                  break;
```

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

my_editor.h

```
#pragma once
#include "shape.h"
#include "point_shape.h"
#include "line_shape.h"
#include "rect_shape.h"
#include "ellipse_shape.h"
#include "lineoo_shape.h"
#include "cube_shape.h"
#include "tool_bar.h"
#include <string>

#define MY_SHAPE_ARRAY_SIZE			117

class MyEditor
{
private:
	MyEditor() {}
	MyEditor(const MyEditor& root) = delete;
	MyEditor& operator = (const MyEditor&) = delete;
	ToolBar toolbar;
	Shape** pshape = new Shape * [MY_SHAPE_ARRAY_SIZE] {};;
	int i = 0;
	Shape* parentShape = NULL;
	long xstart = 0, ystart = 0, xend = 0, yend = 0;
public:
	~MyEditor();
	static MyEditor& getInstance()
	{
		static MyEditor instance;
		return instance;
	}

	void Start(Shape*, HWND, UINT, UINT, LPARAM);
	void OnLBdown(HWND);
	std::wstring OnLBup(HWND);
	void RemoveItem(int);
	void OnMouseMove(HWND);
	void OnPaint(HWND, int);
	void OnCreate(HWND, HINSTANCE);
	void OnNotify(HWND, WPARAM, LPARAM);
	void OnSize(HWND);
};
```

my_editor.cpp

```
#include "my_editor.h"

MyEditor::~MyEditor()
{
	delete[] pshape;
}
```

```
void MyEditor::Start(Shape *shape, HWND hWnd, UINT toolID, UINT menuID, LPARAM lParam)
{
	parentShape = shape;
	toolbar.OnToolMove(hWnd, lParam);
}

void MyEditor::OnLBdown(HWND hWnd)
{
	if (parentShape)
	{
		POINT pt;

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

		xstart = pt.x;
		ystart = pt.y;

		pshape[i] = parentShape->copyShape();
		pshape[i]->Set(xstart, ystart, xend, yend);
	}
}

std::wstring MyEditor::OnLBup(HWND hWnd)
{
	std::wstring properties = L"";

	if (xstart)
	{
		POINT pt;

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

		xend = pt.x;
		yend = pt.y;

		pshape[i]->Set(xstart, ystart, xend, yend);
		properties = pshape[i]->getProperties();
		if (i < MY_SHAPE_ARRAY_SIZE - 1) i++;
		else i = 0;

		xstart = 0, ystart = 0, xend = 0, yend = 0;

		InvalidateRect(hWnd, NULL, TRUE);
	}

	return properties;
}

void MyEditor::OnMouseMove(HWND hWnd)
{
	if (xstart)
	{
		POINT pt;
		HPEN hPen = CreatePen(PS_DOT, 1, 0);
		HDC hdc = GetDC(hWnd);
		HPEN hPenOld = (HPEN)SelectObject(hdc, hPen);

		SetROP2(hdc, R2_NOTXORPEN);
```

```
                if (xend) pshape[i]->Gum(hdc);

                GetCursorPos(&pt);
                ScreenToClient(hWnd, &pt);
                xend = pt.x;
                yend = pt.y;

                pshape[i]->Set(xstart, ystart, xend, yend);
                pshape[i]->Gum(hdc);

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

void MyEditor::OnPaint(HWND hWnd, int selectedItem)
{
        PAINTSTRUCT ps;
        HDC hdc;
        hdc = BeginPaint(hWnd, &ps);
        for (int i = 0; i < MY_SHAPE_ARRAY_SIZE; i++)
                if (pshape[i])
                {
                        bool isSelected = selectedItem == i ? true : false;
                        pshape[i]->Show(hdc, isSelected);
                }
        EndPaint(hWnd, &ps);
}


void MyEditor::RemoveItem(int selectedItem)
{
        if (selectedItem >= 0)
        {
                for (int i = 0; i < MY_SHAPE_ARRAY_SIZE; i++)
                        if (i >= selectedItem)
                        {
                                Shape* next = pshape[i + 1];
                                if (next)
                                        pshape[i] = next;
                                else
                                {
                                        pshape[i] = NULL;
                                        break;
                                }
                        }
                i--;
        }
}

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);
}
```

Point_shape.h

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

class PointShape : virtual public Shape
{
public:
	PointShape(void);
	Shape* copyShape();

	std::wstring getName();
	void Draw(HDC);
	void Gum(HDC);
};
```

point_shape.cpp

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

PointShape::PointShape() {}

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

std::wstring PointShape::getName()
{
	xstart = xend;
	ystart = yend;
	xend = 0;
	yend = 0;
	return L"Крапка";
}

void PointShape::Draw(HDC hdc)
{
	Rectangle(hdc, xstart - 1, ystart - 1, xstart + 1, ystart + 1);
}

void PointShape::Gum(HDC hdc) {}
```

line_shape.h

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

class LineShape : virtual public Shape
{
public:
	LineShape(void);
	Shape* copyShape();

	std::wstring getName();
	void Draw(HDC);
	void Gum(HDC);
};
```

line_shape.cpp

```
#include "line_shape.h"

LineShape::LineShape() {}

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

std::wstring LineShape::getName()
{
	return L"Лінія";
}

void LineShape::Draw(HDC hdc)
{
	MoveToEx(hdc, xstart, ystart, NULL);
	LineTo(hdc, xend, yend);
}

void LineShape::Gum(HDC hdc)
{
	Draw(hdc);
}
```

rect_shape.h

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

class RectShape : virtual public Shape
{
public:
	RectShape(void);
	Shape* copyShape();

	std::wstring getName();
	void Draw(HDC);
	void Gum(HDC);
};
```

rect_shape.cpp

```
#include "rect_shape.h"

RectShape::RectShape() {}

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

std::wstring RectShape::getName()
{
	return L"Прямокутник";
}
```

```
void RectShape::Draw(HDC hdc)
{
	HBRUSH hBrush = (HBRUSH)CreateSolidBrush(RGB(220, 220, 220));
	HBRUSH hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);

	Rectangle(hdc, xstart, ystart, xend, yend);

	SelectObject(hdc, hBrushOld);
	DeleteObject(hBrush);
}

void RectShape::Gum(HDC hdc)
{
	MoveToEx(hdc, xstart, ystart, NULL);
	LineTo(hdc, xstart, yend);
	LineTo(hdc, xend, yend);
	LineTo(hdc, xend, ystart);
	LineTo(hdc, xstart, ystart);
}
```

ellipse_shape.h

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

class EllipseShape : virtual public Shape
{
public:
	EllipseShape(void);
	Shape* copyShape();

	std::wstring getName();
	void Draw(HDC);
	void Gum(HDC);
};
```

ellipse_shape.cpp

```
#include "ellipse_shape.h"

EllipseShape::EllipseShape() {}

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

std::wstring EllipseShape::getName()
{
	return L"Еліпс";
}

void EllipseShape::Draw(HDC hdc)
{
	Ellipse(hdc, xstart * 2 - xend, ystart * 2 - yend, xend, yend);
}

void EllipseShape::Gum(HDC hdc)
{
	Draw(hdc);
}
```

lineoo_shape.**h**

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

class LineOOShape : public LineShape, public EllipseShape
{
public:
	LineOOShape(void);
	Shape* copyShape();

	std::wstring getName();
	void Draw(HDC);
	void Gum(HDC);
};
```

lineoo_shape.**cpp**

```
#include "lineoo_shape.h"

LineOOShape::LineOOShape() {}

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

std::wstring LineOOShape::getName()
{
	return L"Гантеля";
}

void LineOOShape::Draw(HDC hdc)
{
	long x1 = xstart, y1 = ystart, x2 = xend, y2 = yend;

	LineShape::Draw(hdc);

	Set(x1, y1, x1 + 10, y1 + 10);
	EllipseShape::Draw(hdc);

	Set(x2, y2, x2 + 10, y2 + 10);
	EllipseShape::Draw(hdc);

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

void LineOOShape::Gum(HDC hdc)
{
	Draw(hdc);
}
```

cube_shape.h

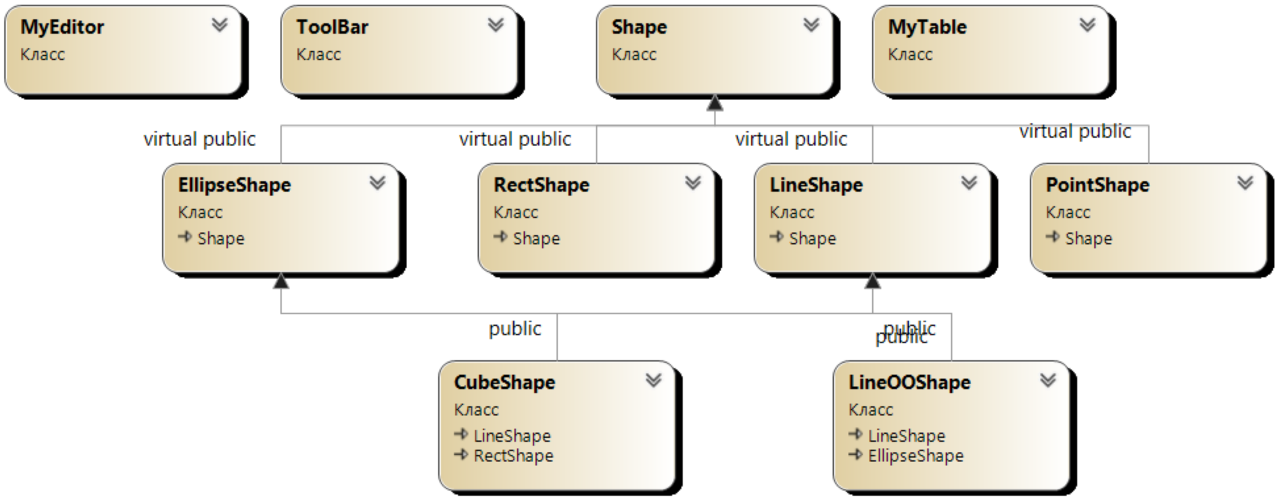
```
#pragma once
#include "line_shape.h"
#include "rect_shape.h"
```

```
class CubeShape : public LineShape, public RectShape
{
public:
	CubeShape(void);
	Shape* copyShape();

	std::wstring getName();
	void Draw(HDC);
	void Gum(HDC);
};
```

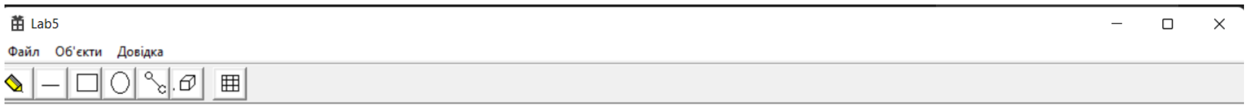
cube_shape.cpp

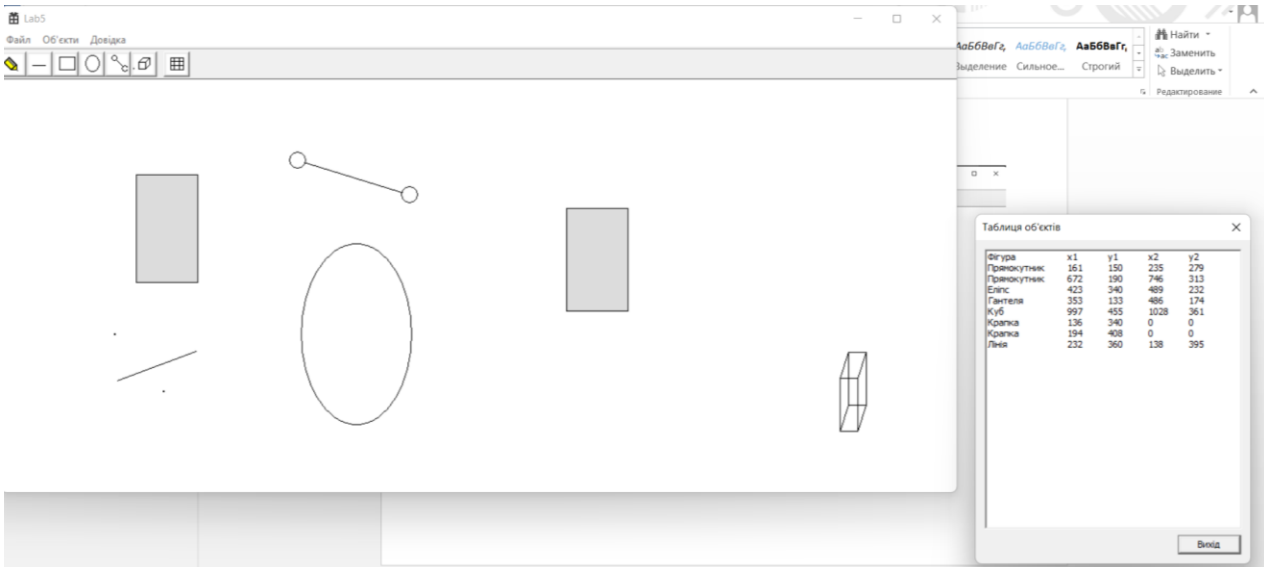
```
#include "cube_shape.h"

CubeShape::CubeShape() {}

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

std::wstring CubeShape::getName()
{
	return L"Куб";
}

void CubeShape::Draw(HDC hdc)
{
	long x1 = xstart, y1 = ystart, x2 = xend, y2 = yend;
	long xd = (x2 - x1) / 3, yd = (y2 - y1) / 3;

	Set(x1, y1, x2 - xd, y2 - yd);
	RectShape::Gum(hdc);

	Set(x1 + xd, y1 + yd, x2, y2);
	RectShape::Gum(hdc);

	Set(x1, y1, x1 + xd, y1 + yd);
	LineShape::Draw(hdc);

	Set(x2, y2, x2 - xd, y2 - yd);
	LineShape::Draw(hdc);

	Set(x1, y2 - yd, x1 + xd, y2);
	LineShape::Draw(hdc);

	Set(x2 - xd, y1, x2, y1 + yd);
	LineShape::Draw(hdc);

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

void CubeShape::Gum(HDC hdc)
{
	Draw(hdc);
}
```

Діаграма класів

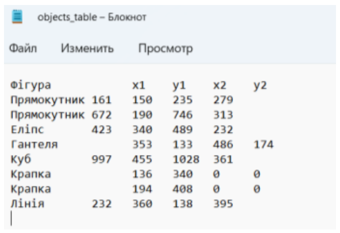


Скріншоти





Фігура	x1	y1	x2	y2
Прямокутник	161	150	235	279
Прямокутник	672	190	746	313
Еліпс	423	340	489	232
Гантеля	353	133	486	174
Куб	997	455	1028	361
Крапка	136	340	0	0
Крапка	194	408	0	0
Лінія	232	360	138	395



```
Фігура              x1   y1   x2   y2
Прямокутник 161     150  235  279
Прямокутник 672     190  746  313
Еліпс       423     340  489  232
Гантеля             353  133  486  174
Куб         997     455  1028 361
Крапка              136  340  0    0
Крапка              194  408  0    0
Лінія       232     360  138  395
```

Висновки. Під час виконання лабораторної роботи №5 я отримав вміння та навички використання сінглтону, роботи з файлами. У результаті роботи додано таблицю у якій можна виділити та видалити фігуру. Крім того, засобами Visual Studio C++ я створив діаграму класів, яку було додано до звіту.