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 №3

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 – Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши графічний інтерфейс користувача.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім'ям Lab3.

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.

Для вибору варіанту використовується Ж = Жлаб2 + 1, де Жлаб2 – номер студента в журналі, який використовувався для попередньої лаб. роботи №2.

для 17-го студента у списку (Жлаб2 = 17) Ж буде 18:

- динамічний масив Shape **pcshape;
кількість елементів масиву вказівників як для статичного, так і динамічного має бути N = Ж+100.
За варіантом це 118.

"гумовий" слід (18 mod 4 = 2) – суцільна лінія синього кольору

прямокутник:

ввід по двом протилежним кутам (18 mod 2 = 0)

чорний контур прямокутника без заповнення (18 mod 5 = 3)

еліпс:

- від центру до одного з кутів охоплюючого прямокутника (18 mod 2 = 1)

чорний контур з кольоровим заповненням (18 mod 5 = 3)

колір заповнення: сірий для (18 mod 6 = 0)

позначка поточного типу об'єкту: в меню (18 mod 2 = 0)

Вихідні тексти файлів:

Lab2.h

```
#pragma once

#include "resource.h"
```

Lab2.cpp

```
// Lab2.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "Lab2.h"
#include "shape_editor.h"
#include "toolbar.h"



ShapeObjectsEditor Dima;
ToolBar DimonsToolBar;

#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,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

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

    return TRUE;
}
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_CREATE:
        DimonsToolBar.OnCreate(hWnd, hInst);
        DimonsToolBar.OnTool(hWnd, ID_TOOL_POINT); //selecting point from start
        Dima.StartPointEditor();
        break;
```

```
case WM_SIZE:
    DimonsToolBar.OnSize(hWnd);
    break;

case WM_NOTIFY:
    DimonsToolBar.OnNotify(hWnd, wParam, lParam);
    break;
case WM_LBUTTONDOWN: //натиснуто ліву кнопку миші у клієнтській частині вікна
    Dima.OnLBdown(hWnd);
    break;
case WM_LBUTTONUP: //відпущено ліву кнопку миші у клієнтській частині вікна
    Dima.OnLBup(hWnd);
    break;
case WM_MOUSEMOVE: //пересунуто мишу у клієнтській частині вікна
    Dima.OnMouseMove(hWnd);
    break;
case WM_PAINT: //потрібно оновлення зображення клієнтської частині вікна
    Dima.OnPaint(hWnd);
    break;
case WM_INITMENUPOPUP: //позначка пунктів меню - для окремих варіантів завдань
    Dima.OnInitMenuPopup(hWnd, wParam);
    break;
case WM_COMMAND:
{
    int wmId = LOWORD(wParam);
    // Parse the menu selections:
    switch (wmId)
    {
    case ID_TOOL_POINT:
    case ID_32771:
        DimonsToolBar.OnTool(hWnd, ID_TOOL_POINT);
        Dima.StartPointEditor(); //початок вводу точкових об’єктів
        break;
    case ID_TOOL_LINE:
    case ID_32772:
        DimonsToolBar.OnTool(hWnd, ID_TOOL_LINE);
        Dima.StartLineEditor(); //початок вводу об’єктів-ліній
        break;
    case ID_TOOL_RECT:
    case ID_32773:
        DimonsToolBar.OnTool(hWnd, ID_TOOL_RECT);
        Dima.StartRectEditor(); //початок вводу прямокутників
        break;
    case ID_TOOL_ELLIPSE:
    case ID_32774:
        DimonsToolBar.OnTool(hWnd, ID_TOOL_ELLIPSE);
        Dima.StartEllipseEditor(); //початок вводу еліпсів
        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;
}
```

editor.h

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


class Editor {
public:
       virtual void OnLBdown(HWND) = 0;
       virtual void OnLBup(HWND) = 0;
       virtual void OnMouseMove(HWND) = 0;
       virtual void OnPaint(HWND) = 0;
};
```

shape_editor.h

```
#pragma once
class ShapeObjectsEditor
{
       //private:

public:
       ShapeObjectsEditor(void);
       ~ShapeObjectsEditor();
       void StartPointEditor();
       void StartLineEditor();
       void StartRectEditor();
       void StartEllipseEditor();
       void OnLBdown(HWND);
       void OnLBup(HWND);
```

```
        void OnMouseMove(HWND);
        void OnPaint(HWND);
        void OnInitMenuPopup(HWND, WPARAM);
};
```

shape_editor.cpp

```
#include "framework.h"
#include "shape_editor.h"
#include "shape.h"
#include "editor.h"
#include "ShapeEditor.h"
#include "point_editor.h"
#include "line_editor.h"
#include "rect_editor.h"
#include "ellipse_editor.h"

ShapeEditor* pse = NULL;

ShapeObjectsEditor::ShapeObjectsEditor(void) {}
ShapeObjectsEditor::~ShapeObjectsEditor() {}

void ShapeObjectsEditor::StartPointEditor() {
        if (pse) delete pse;
        pse = new PointEditor;
}

void ShapeObjectsEditor::StartLineEditor() {
        if (pse) delete pse;
        pse = new LineEditor;
}

void ShapeObjectsEditor::StartRectEditor() {
        if (pse) delete pse;
        pse = new RectEditor;
}

void ShapeObjectsEditor::StartEllipseEditor() {
        if (pse) delete pse;
        pse = new EllipseEditor;
}

void ShapeObjectsEditor::OnLBdown(HWND hWnd) {
        if (pse) pse->OnLBdown(hWnd);
}

void ShapeObjectsEditor::OnLBup(HWND hWnd) {
        if (pse) pse->OnLBup(hWnd);
}

void ShapeObjectsEditor::OnMouseMove(HWND hWnd) {
        if (pse) pse->OnMouseMove(hWnd);
}

void ShapeObjectsEditor::OnPaint(HWND hWnd) {
        if (pse) pse->OnPaint(hWnd);
}

void ShapeObjectsEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParam) {
        if (pse) pse->OnInitMenuPopup(hWnd, wParam);
```

```
}
```

shape.h

```
#pragma once
class Shape {
protected:
	long xs1, ys1, xs2, ys2;
public:
	void SetStart(long xStart, long yStart);
	void SetEnd(long xEnd, long yEnd);
	long getXStart(), getXEnd(), getYStart(), getYEnd();
	virtual void Show(HDC) = 0;
};
```

shape.cpp

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

void Shape::SetStart(long xStart, long yStart)
{
	xs1 = xStart;
	ys1 = yStart;
}

void Shape::SetEnd(long xEnd, long yEnd)
{
	xs2 = xEnd;
	ys2 = yEnd;
}

long Shape::getXStart()
{
	return xs1;
}

long Shape::getXEnd()
{
	return xs2;
}

long Shape::getYStart()
{
	return ys1;
}

long Shape::getYEnd()
{
	return ys2;
}
```

point_shape.h

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

class PointShape : public Shape {
public:
	void Show(HDC);
};
```

point_shape.cpp

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


void PointShape::Show(HDC hdc) {
	SetPixel(hdc, xs2, ys2, 0x00000000);
}
```

line_shape.h

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

class LineShape : public Shape {
public:
	void Show(HDC);
};
```

line_shape.cpp

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

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

rect_shape.h

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

class RectShape : public Shape {
public:
	void Show(HDC);
};
```

rect_shape.cpp

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

HBRUSH hBrush, hBrushOld;

void RectShape::Show(HDC hdc) {
	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);

	SelectObject(hdc, hBrushOld); //відновлюється пензль-попередник
	DeleteObject(hBrush);
}
```

ellipse_shape.h

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

class EllipseShape : public Shape {
public:
	void Show(HDC);
};
```

ellipse_shape.cpp

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

void EllipseShape::Show(HDC hdc) {

	long xStart = xs2 - 2 * (xs2 - xs1);  //до будь-якого кута
	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);
}
```

ShapeEditor.h

```
#pragma once

#include "editor.h"
#include "shape.h"

#define MY_SHAPE_ARRAY_SIZE 118


class ShapeEditor : public Editor {
protected:
	POINT pt;
	HMENU hMenu, hSubMenu;
	int index = 0;
	void GetPossibleIndex();
	static Shape** pcshape;
	bool isPainting = false;
	virtual void GetShape() = 0;
	//... корисні члени, які враховують специфіку Windows-програм
public:
	ShapeEditor(void);
    void OnLBdown(HWND); // Я не робив його віртуальним бо він однаковий для всіх 4 нащадків
	void OnLBup(HWND);  // теж не робив бо зміг зробити однаковим за домпомогою
віртуального метода GetShape()
	virtual void OnMouseMove(HWND) = 0;
	void OnPaint(HWND);
	virtual void OnInitMenuPopup(HWND, WPARAM) = 0;
};
```

ShapeEditor.cpp

```
#include "ShapeEditor.h"

Shape** ShapeEditor::pcshape = new Shape * [MY_SHAPE_ARRAY_SIZE] {};


ShapeEditor::ShapeEditor(void) { };

void ShapeEditor::OnLBdown(HWND hWnd) {
	isPainting = true;
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	GetPossibleIndex();
	GetShape();
	pcshape[index]->SetEnd(pt.x, pt.y);
	pcshape[index]->SetStart(pt.x, pt.y);
};

void ShapeEditor::OnLBup(HWND hWnd) {
	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	pcshape[index]->SetEnd(pt.x, pt.y);
	InvalidateRect(hWnd, NULL, TRUE);
	isPainting = false;
};

void ShapeEditor::GetPossibleIndex() {
	for (int i = 0; i < MY_SHAPE_ARRAY_SIZE; i++)
		if (!pcshape[i]) {
			index = i;
			break;
		}
};

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

point_editor.h

```
#pragma once
class PointEditor : public ShapeEditor {
	void GetShape();
	void OnMouseMove(HWND);
	void OnInitMenuPopup(HWND, WPARAM);
};
```

point_editor.cpp

```
#include "ShapeEditor.h"
```

```
#include "point_shape.h"
#include "point_editor.h"
#include "Resource.h"


void PointEditor::OnMouseMove(HWND hWnd) {
}

void PointEditor::GetShape() {
	pcshape[index] = new PointShape;
};

void PointEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	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);
	}
}
```

line_editor.h

```
#pragma once

class LineEditor : public ShapeEditor {
	void GetShape();
	void OnMouseMove(HWND);
	void OnInitMenuPopup(HWND, WPARAM);
};
```

line_editor.cpp

```
#include "ShapeEditor.h"
#include "line_shape.h"
#include "line_editor.h"
#include "Resource.h"


void LineEditor::OnMouseMove(HWND hWnd) {
	if (!isPainting) return;
	HPEN hPenOld, hPen;
	HDC hdc;
	hdc = GetDC(hWnd); //отримуємо контекст вікна для малювання
	SetROP2(hdc, R2_NOTXORPEN);
	hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
	hPenOld = (HPEN)SelectObject(hdc, hPen); //Малюються лінії "гумового" сліду
попереднього розташування курсору

	MoveToEx(hdc, pcshape[index]->getXStart(), pcshape[index]->getYStart(), NULL);
	LineTo(hdc, pcshape[index]->getXEnd(), pcshape[index]->getYEnd());

	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	pcshape[index]->SetEnd(pt.x, pt.y); //координати поточної точки курсору
	MoveToEx(hdc, pcshape[index]->getXStart(), pcshape[index]->getYStart(), NULL);
```

```
	LineTo(hdc, pcshape[index]->getXEnd(), pcshape[index]->getYEnd()); //Малюються лінії
"гумового" сліду для поточного розташування курсору

	SelectObject(hdc, hPenOld);
	DeleteObject(hPen);
	ReleaseDC(hWnd, hdc); //закриваємо контекст вікна
}

void LineEditor::GetShape() {
	pcshape[index] = new LineShape;
};

void LineEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	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);
	}
}
```

rect_editor.h

```
#pragma once

class RectEditor : public ShapeEditor {
	void OnMouseMove(HWND);
	void GetShape();
	void OnInitMenuPopup(HWND, WPARAM);
};
```

rect_editor.cpp

```
#include "ShapeEditor.h"
#include "rect_shape.h"
#include "rect_editor.h"
#include "Resource.h"

void RectEditor::OnMouseMove(HWND hWnd) {
	if (!isPainting) return;
	HPEN hPenOld, hPen;
	HDC hdc;
	hdc = GetDC(hWnd); //отримуємо контекст вікна для малювання
	SetROP2(hdc, R2_NOTXORPEN);
	hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
	hPenOld = (HPEN)SelectObject(hdc, hPen); //Малюються лінії "гумового" сліду
попереднього розташування курсору

	Rectangle(hdc, pcshape[index]->getXStart(), pcshape[index]->getYStart(),
pcshape[index]->getXEnd(), pcshape[index]->getYEnd());

	GetCursorPos(&pt);
	ScreenToClient(hWnd, &pt);
	pcshape[index]->SetEnd(pt.x, pt.y); //координати поточної точки курсору
	Rectangle(hdc, pcshape[index]->getXStart(), pcshape[index]->getYStart(),
pcshape[index]->getXEnd(), pcshape[index]->getYEnd());
```

```
	//Малюються лінії "гумового" сліду для поточного розташування курсору

	SelectObject(hdc, hPenOld);
	DeleteObject(hPen);
	ReleaseDC(hWnd, hdc); //закриваємо контекст вікна
}

void RectEditor::GetShape() {
	pcshape[index] = new RectShape;
};

void RectEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	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);
	}
}
```

ellipse_editor.h

```
#pragma once

class EllipseEditor : public ShapeEditor {
	void GetShape();
	void OnMouseMove(HWND);
	void OnInitMenuPopup(HWND, WPARAM);
};
```

ellipse _editor.cpp

```
#include "ShapeEditor.h"
#include "ellipse_shape.h"
#include "ellipse_editor.h"
#include "Resource.h"


void EllipseEditor::OnMouseMove(HWND hWnd) {
	if (!isPainting) return;
	HPEN hPenOld, hPen;
	HDC hdc;
	hdc = GetDC(hWnd); //отримуємо контекст вікна для малювання
	SetROP2(hdc, R2_NOTXORPEN);
	hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
	hPenOld = (HPEN)SelectObject(hdc, hPen); //Малюються лінії "гумового" сліду
попереднього розташування курсору

	long xStart = pcshape[index]->getXEnd() - (2 * (pcshape[index]->getXEnd() -
pcshape[index]->getXStart()));
	long yStart = pcshape[index]->getYEnd() - (2 * (pcshape[index]->getYEnd() -
pcshape[index]->getYStart()));

	Ellipse(hdc, xStart, yStart, pcshape[index]->getXEnd(), pcshape[index]->getYEnd());

	GetCursorPos(&pt);
```

```
	ScreenToClient(hWnd, &pt);
	pcshape[index]->SetEnd(pt.x, pt.y); //координати поточної точки курсору


	 xStart = pcshape[index]->getXEnd() - (2 * (pcshape[index]->getXEnd() -
pcshape[index]->getXStart()));
	 yStart = pcshape[index]->getYEnd() - (2 * (pcshape[index]->getYEnd() -
pcshape[index]->getYStart()));

	Ellipse(hdc, xStart, yStart, pcshape[index]->getXEnd(), pcshape[index]->getYEnd());
	//Малюються лінії "гумового" сліду для поточного розташування курсору

	SelectObject(hdc, hPenOld);
	DeleteObject(hPen);
	ReleaseDC(hWnd, hdc); //закриваємо контекст вікна
}

void EllipseEditor::GetShape() {
	pcshape[index] = new EllipseShape;
};

void EllipseEditor::OnInitMenuPopup(HWND hWnd, WPARAM wParam)
{
	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);

	}
}
```

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);
};
```

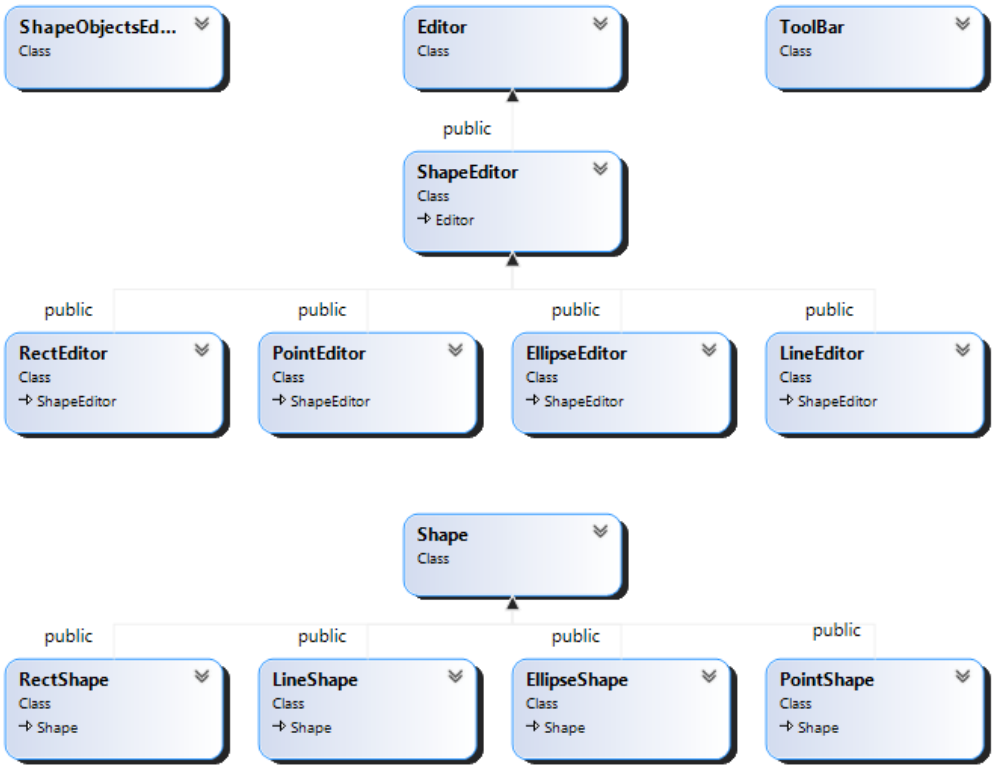
toolbar.cpp

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

ToolBar::ToolBar(void) {}
```

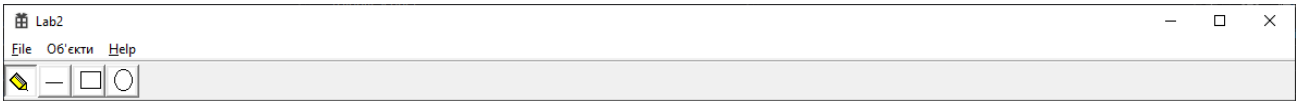
```
void ToolBar::OnCreate(HWND hWnd, HINSTANCE hInst)
{
    TBBUTTON tbb[4]; //для Toolbar з чотирма кнопками
    ZeroMemory(tbb, sizeof(tbb));
    tbb[0].iBitmap = 0; //стандартне зображення
    tbb[0].fsState = TBSTATE_ENABLED;
    tbb[0].fsStyle = TBSTYLE_BUTTON; //тип елементу - кнопка
    tbb[0].idCommand = 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;

    SendMessage(hwndToolBar, TB_ADDBUTTONS, 4, (LPARAM)&tbb);

    hwndToolBar = CreateToolbarEx(hWnd, //батьківське вікно
        WS_CHILD | WS_VISIBLE | WS_BORDER | WS_CLIPSIBLINGS | CCS_TOP |
        TBSTYLE_TOOLTIPS,
        IDC_MY_TOOLBAR, //ID дочірнього вікна Toolbar
        4,
        hInst,
        IDB_BITMAP1,
        tbb, //масив опису кнопок
        4, //кількість кнопок
        24, 24, 24, 24, //розташування та розміри
        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;

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

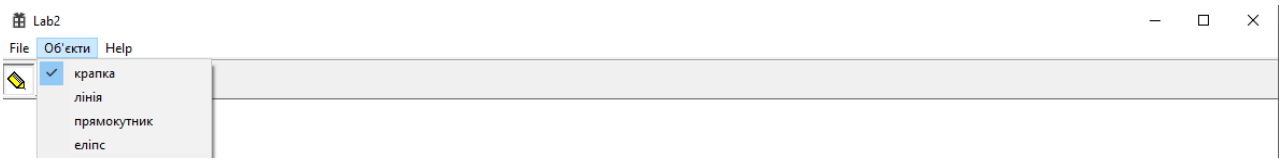
Діаграма класів



Скріншоти

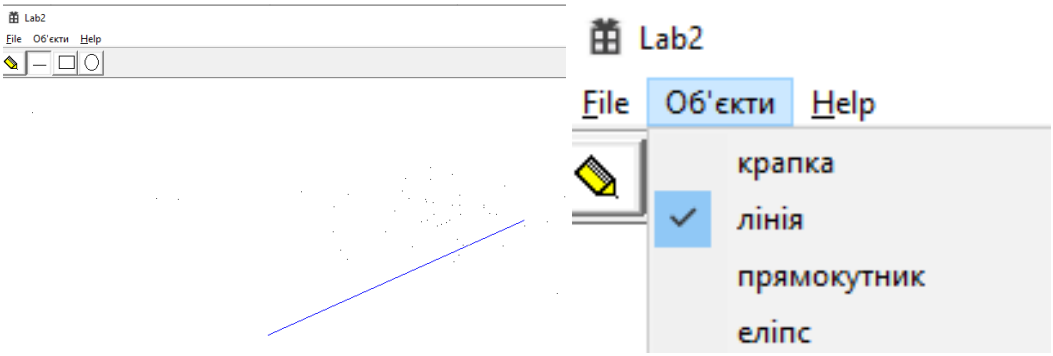


При натисканні на “об’єкти” бачимо 4 пункти меню – крапка, лінія, прямокутник, еліпс. За замовчуванням обрано крапку і це видно і в тулбарі, і в меню об’єкти.



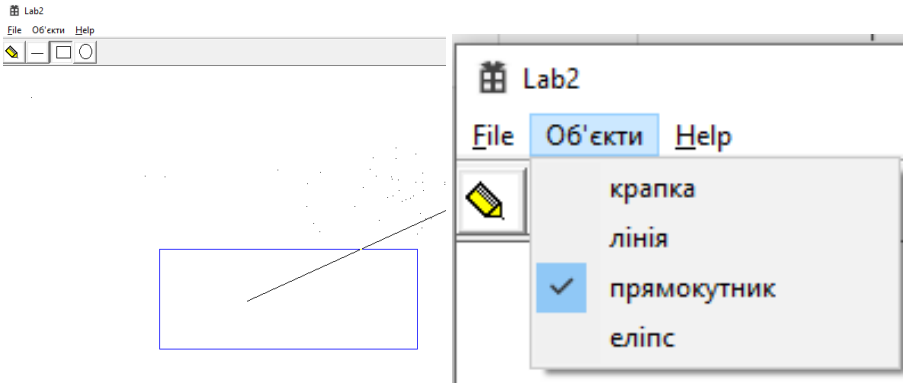
Крапки ставляться після відпускання миші, поставимо декілька.

Оберемо лінію у меню і почнемо її малювати, поки не відпустимо ліву клавішу миші, будемо бачити гумовий слід, за варіантом лінія синя та суцільна.

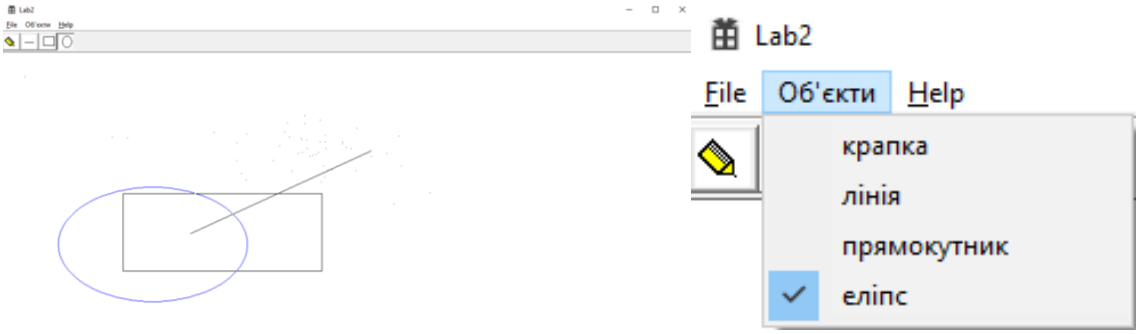


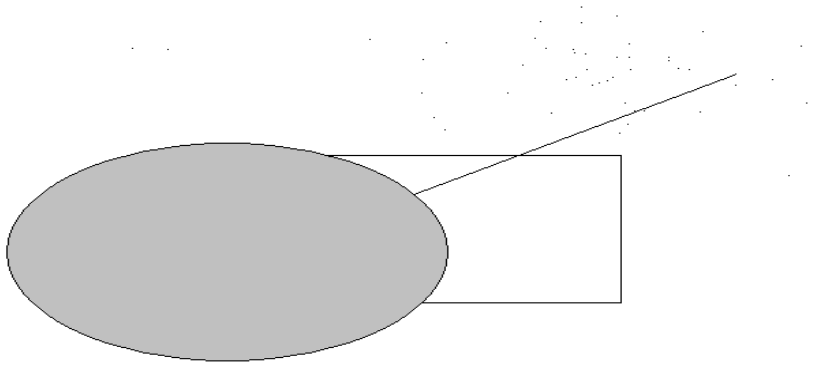
Після відпускання побачимо вже намальовану лінію.

Аналогічно для прямокутника. У мене він без заповнення і малюється по 2 кутам.

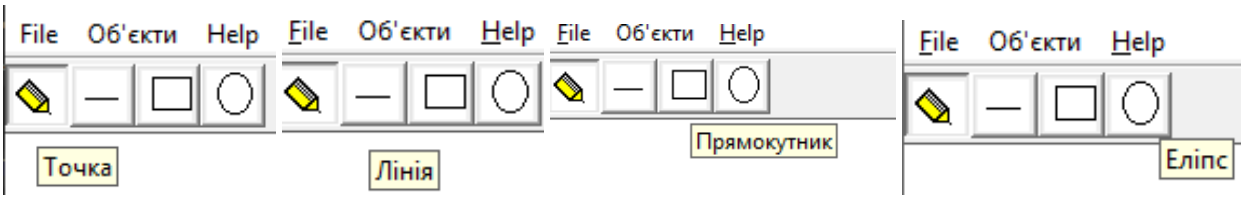


Аналогічно для еліпсу. Він у мене за варіантом сірий і від центру до будь-якого з кутів.





Як видно з скріншотів вище, при виборі іншого пункту меню тулбарі, а також у меню змінюється позначка що відображає поточний режим малювання. Також створив підказки у Toolbar.



Висновки. Під час виконання лабораторної роботи №3 я навчився створювати класи, описувати та реалізовувати їхні методи. Я ознайомився та попрацював із віртуальними методами у класах, навчився створювати похідні класи. На практиці познайомився з абстрактними та віртуальними класами та навчився використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм при поботі із класами у C++. В результаті роботи я запрограмував графічний редактор, за допомогою якого можна малювати деякі фігури. Крім того, засобами Visual Studio C++ я створив діаграму класів, яку було додано до звіту. Створив Toolbar та забезпечив відповідність пунктів меню і його кнопок, створив власні зображення на кнопках та додав підказки.