**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»

на тему «Розробка інтерфейсу користувача на C++»

Виконав: Перевірив:

студент групи ІП-93 Порєв Віктор Миколайович

Домінський Валентин Олексійович

номер залікової книжки: 9311

Київ 2020

**Мета:**

Мета роботи – отримати вміння та навички використовувати інкапсуляцію,

абстракцію типів, успадкування та поліморфізм на основі класів С++, 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.

**Варіанти завдань**

- статичний масив для Shape (10 mod 3 = 1) обсягом 110 об’єктів

- "гумовий" слід (10 mod 4 = 2) – суцільна лінія синього кольору

- прямокутник:

- по двом протилежним кутам (10 mod 2 = 0)

- чорний контур з білим заповненням (10 mod 5 = 0)

- еліпс:

- від центру до одного з кутів охоплюючого прямокутника (10 mod 2 = 0)

- чорний контур еліпсу без заповнення (10 mod 5 = 0)

-позначка поточного типу об’єкту:-в меню (метод OnInitMenuPopup) (10 mod 2 = 0)

**Вихідні тексти файлів:**

**Lab2.cpp:**

// Lab1.cpp : Defines the input point for the application.

//

// First Part

#include "framework.h"

#include "pch.h"

#include "Lab2.h"

#include "Resource.h"

#include "shape\_editor.h"

#define MAX\_LOADSTRING 100

#pragma region Variables

// Global variables:

HINSTANCE hInst**;** // Current instance

WCHAR szTitle**[**MAX\_LOADSTRING**];** // Header row text

WCHAR szWindowClass**[**MAX\_LOADSTRING**];** // Class name of main window

ShapeObjectsEditor editorShape**;**

LPCSTR currentShape**;**

const LPCSTR POINT\_NAME **=** "Крапка"**;**

const LPCSTR LINE\_NAME **=** "Лінія"**;**

const LPCSTR RECTANGLE\_NAME **=** "Прямокутник"**;**

const LPCSTR ELLIPSE\_NAME **=** "Овал"**;**

#pragma endregion

// Send 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**);**

#pragma region DefaultFunctions

// Second Part

// Enter Point "wWinMain"

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place the code here.

// Global line initialization

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 cycle:

while (GetMessage(&msg, nullptr, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int)msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// OBJECTIVE: To register the window class.

// Text of Function

/// <summary>

/// Register the window class.

/// </summary>

/// <param name="hInstance">The h instance.</param>

/// <returns></returns>

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);

}

//

// FUNCTION: InitInstance(HINSTANCE, int)

//

// OBJECTIVE: Saves the instance marker and creates the main window

//

// COMMENTARIES:

//

// In this function, the instance marker is saved in a global variable, and also

// the main program window is created and displayed.

//

/// <summary>

/// Saves the instance marker and creates the main window

/// </summary>

/// <param name="hInstance">The h instance.</param>

/// <param name="nCmdShow">The n command show.</param>

/// <returns></returns>

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Save instance marker in 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;

}

/// <summary>

/// Message handler for "About" window.

/// </summary>

/// <param name="hDlg">The h dialog.</param>

/// <param name="message">The message.</param>

/// <param name="wParam">The w parameter.</param>

/// <param name="lParam">The l parameter.</param>

/// <returns></returns>

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;

}

#pragma endregion

#pragma region ModifiedFuntions

// Third Part

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// OBJECTIVE: Processes messages in the main window.

//

// WM\_COMMAND - Process the application menu

// WM\_PAINT - Drawing of the main window

// WM\_DESTROY - Send message about exit and return

//

//

/// <summary>

/// Processes messages in the main window.

/// </summary>

/// <param name="hWnd">The h WND.</param>

/// <param name="message">The message.</param>

/// <param name="wParam">The w parameter.</param>

/// <param name="lParam">The l parameter.</param>

/// <returns></returns>

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_LBUTTONDOWN:

editorShape.OnLBdown(hWnd);

break;

case WM\_LBUTTONUP:

editorShape.OnLBup(hWnd);

break;

case WM\_MOUSEMOVE:

editorShape.OnMouseMove(hWnd);

break;

case WM\_PAINT:

editorShape.OnPaint(hWnd);

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_POINT:

editorShape.StartPointEditor();

currentShape = POINT\_NAME;

ChangeWindowText(hWnd, currentShape);

break;

case IDM\_LINE:

editorShape.StartLineEditor();

currentShape = LINE\_NAME;

ChangeWindowText(hWnd, currentShape);

break;

case IDM\_RECTANGLE:

editorShape.StartRectangleEditor();

currentShape = RECTANGLE\_NAME;

ChangeWindowText(hWnd, currentShape);

break;

case IDM\_ELLIPSE:

editorShape.StartEllipseEditor();

currentShape = ELLIPSE\_NAME;

ChangeWindowText(hWnd, currentShape);

break;

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

return DefWindowProcW(hWnd, message, wParam, lParam);

}

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProcW(hWnd, message, wParam, lParam);

}

return 0;

}

/// <summary>

/// Set main window text

/// </summary>

/// <param name="hWnd">window</param>

/// <param name="name">name</param>

void ChangeWindowText(HWND hWnd, LPCSTR name)

{

SetWindowTextA(hWnd, name);

}

#pragma endregion ModifiedFuntions

**Shape.cpp:**

#include "framework.h"

#include "pch.h"

#include "shape.h"

#include "colors.h"

#pragma region Functions

/// <summary>

/// // Get coords of points

/// </summary>

/// <param name="x1">first point</param>

/// <param name="y1">second point</param>

/// <param name="x2">third point</param>

/// <param name="y2">fourth point</param>

void Shape**::**Set**(**long x1**,** long y1**,** long x2**,** long y2**)**

**{**

xs1 **=** x1**;**

ys1 **=** y1**;**

xs2 **=** x2**;**

ys2 **=** y2**;**

**}**

/// <summary>

/// Shows the pixel

/// </summary>

/// <param name="hdc">handle to a device context</param>

void PointShape**::**Show**(**HDC hdc**)**

**{**

SetPixel**(**hdc**,** xs1**,** ys1**,** black**);** // Show point

**}**

/// <summary>

/// Shows the line

/// </summary>

/// <param name="hdc">handle to a device context</param>

void LineShape**::**Show**(**HDC hdc**)**

**{**

HPEN hPen**,** hPenOld**;**

hPen = CreatePen(PS\_SOLID, 1, black); // Create pen

hPenOld = (HPEN)SelectObject(hdc, hPen);

MoveToEx(hdc, xs1, ys1, NULL);

LineTo(hdc, xs2, ys2); // Create line

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

/// <summary>

/// Shows the rectangle

/// </summary>

/// <param name="hdc">handle to a device context</param>

void RectangleShape::Show(HDC hdc)

{

HPEN hPen, hPenOld;

hPen = CreatePen(PS\_SOLID, 1, black); // Create pen

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); // Create rectangle

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

}

/// <summary>

/// Shows the ellipse

/// </summary>

/// <param name="hdc">handle to a device context</param>

void EllipseShape::Show(HDC hdc)

{

HPEN hPen, hPenOld;

HBRUSH hBrush, hBrushOld;

hPen = CreatePen(PS\_SOLID, 1, black); // Create pen

hPenOld = (HPEN)SelectObject(hdc, hPen);

hBrush = CreateSolidBrush(blue);

hBrushOld = (HBRUSH)SelectObject(hdc, hBrush);

SelectObject(hdc, hBrush);

Ellipse(hdc, xs1, ys1, xs2, ys2);

SelectObject(hdc, hBrushOld);

DeleteObject(hBrush);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

};

#pragma endregion Functions

**Shape.h:**

#include "pch.h"

/// <summary>

/// Main class for shapes

/// </summary>

class Shape

**{**

protected**:**

long xs1**,** ys1**,** xs2**,** ys2**;**

public**:**

void Set**(**long x1**,** long y1**,** long x2**,** long y2**);**

virtual void Show**(**HDC**)** **=** 0**;**

**};**

/// <summary>

/// Class for points

/// </summary>

class PointShape **:** public Shape

**{**

public**:**

void Show**(**HDC**);**

**};**

/// <summary>

/// Class for lines

/// </summary>

class LineShape **:** public Shape

**{**

public**:**

void Show**(**HDC**);**

**};**

/// <summary>

/// Class for rectangles

/// </summary>

class RectangleShape **:** public Shape

**{**

public**:**

void Show(HDC);

};

/// <summary>

/// Class for ellipses

/// </summary>

class EllipseShape : public Shape

{

public:

void Show(HDC);

};

**Shape\_editor.cpp:**

#include "framework.h"

#include "pch.h"

#include "shape\_editor.h"

#include "shape.h"

#pragma region Variables

const int Size\_Of\_Array **=** 109**;**

Shape**\*\*** pcshape **=** **new** Shape **\*** **[**Size\_Of\_Array**];**

int size **=** 0**;**

bool isPressed**;**

#pragma endregion Variables

#pragma region Functions

#pragma region ShapeObjectsEditor

/// <summary>

/// Constructor

/// </summary>

ShapeObjectsEditor**::**ShapeObjectsEditor**()**

**{**

pse **=** **new** PointEditor**;**

**}**

/// <summary>

/// Destructor

/// </summary>

ShapeObjectsEditor**::~**ShapeObjectsEditor**()**

**{**

**for** **(**int i **=** 0**;** i **<** size**;** i**++)**

**{**

**delete** pcshape**[**i**];**

**}**

**}**

/// <summary>

/// Starts the PointEditor

/// </summary>

void ShapeObjectsEditor::StartPointEditor()

{

if (pse)

{

delete pse;

}

pse = new PointEditor;

}

/// <summary>

/// Starts the LineEditor

/// </summary>

void ShapeObjectsEditor::StartLineEditor()

{

if (pse)

{

delete pse;

}

pse = new LineEditor;

}

/// <summary>

/// Starts the RectangleEditor

/// </summary>

void ShapeObjectsEditor::StartRectangleEditor()

{

if (pse)

{

delete pse;

}

pse = new RectangleEditor;

}

/// <summary>

/// Starts the EllipseEditor

/// </summary>

void ShapeObjectsEditor::StartEllipseEditor()

{

if (pse)

{

delete pse;

}

pse = new EllipseEditor;

}

/// <summary>

/// Do something on left mouse button clicked

/// </summary>

/// <param name="hWnd">window</param>

void ShapeObjectsEditor::OnLBdown(HWND hWnd)

{

if (pse)

{

pse->OnLBdown(hWnd);

}

}

/// <summary>

/// Do something on left mouse button unclicked

/// </summary>

/// <param name="hWnd">window</param>

void ShapeObjectsEditor::OnLBup(HWND hWnd)

{

if (pse)

{

pse->OnLBup(hWnd);

}

}

/// <summary>

/// Do something on left mouse moving

/// </summary>

/// <param name="hWnd">window</param>

void ShapeObjectsEditor::OnMouseMove(HWND hWnd)

{

if (pse && isPressed)

{

pse->OnMouseMove(hWnd);

}

}

/// <summary>

/// Do something on paint

/// </summary>

/// <param name="hWnd">window</param>

void ShapeObjectsEditor::OnPaint(HWND hWnd)

{

ShapeEditor\* draw = new ShapeEditor;

draw->OnPaint(hWnd);

}

#pragma endregion ShapeObjectsEditor

#pragma region ShapeEditor

void ShapeEditor::OnMouseMove(HWND hWnd) {};

/// <summary>

/// Do something on left mouse button clicked

/// </summary>

/// <param name="hWnd">window</param>

void ShapeEditor::OnLBdown(HWND hWnd)

{

isPressed = TRUE;

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x1 = x2 = pt.x;

y1 = y2 = pt.y;

}

/// <summary>

/// Do something on left mouse button unclicked

/// </summary>

/// <param name="hWnd">window</param>

void ShapeEditor::OnLBup(HWND hWnd)

{

POINT pt;

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

isPressed = FALSE;

}

/// <summary>

/// Do something on paint

/// </summary>

/// <param name="hWnd">window</param>

void ShapeEditor::OnPaint(HWND hWnd)

{

PAINTSTRUCT ps;

HDC hdc;

hdc = BeginPaint(hWnd, &ps);

for (int i = 0; i < size; i++)

{

if (pcshape[i])

{

pcshape[i]->Show(hdc);

}

}

EndPaint(hWnd, &ps);

}

#pragma endregion ShapeEditor

#pragma region PointEditor

/// <summary>

/// Do something on left mouse button unclicked

/// </summary>

/// <param name="hWnd">window</param>

void PointEditor::OnLBup(HWND hWnd)

{

\_\_super::OnLBup(hWnd);

PointShape\* Point = new PointShape;

Point->Set(x1, y1, x2, y2);

pcshape[size] = Point;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

#pragma endregion PointEditor

#pragma region LineEditor

/// <summary>

/// Do something on left mouse button unclicked

/// </summary>

/// <param name="hWnd">window</param>

void LineEditor::OnLBup(HWND hWnd)

{

\_\_super::OnLBup(hWnd);

LineShape\* Line = new LineShape;

Line->Set(x1, y1, x2, y2);

pcshape[size] = Line;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

/// <summary>

/// Do something on Mouse moving

/// </summary>

/// <param name="hWnd">window</param>

void LineEditor::OnMouseMove(HWND hWnd)

{

POINT pt;

HPEN hPen, hPenOld;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

hPen = CreatePen(PS\_SOLID, 1, red);

hPenOld = (HPEN)SelectObject(hdc, hPen);

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

MoveToEx(hdc, x1, y1, NULL);

LineTo(hdc, x2, y2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

ReleaseDC(hWnd, hdc);

}

#pragma endregion LineEditor

#pragma region RectangleEditor

/// <summary>

/// Do something on left mouse button unclicked

/// </summary>

/// <param name="hWnd">window</param>

void RectangleEditor::OnLBup(HWND hWnd)

{

\_\_super::OnLBup(hWnd);

RectangleShape\* Rectangle = new RectangleShape;

Rectangle->Set(2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

pcshape[size] = Rectangle;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

/// <summary>

/// Do something on Mouse moving

/// </summary>

/// <param name="hWnd">window</param>

void RectangleEditor::OnMouseMove(HWND hWnd)

{

POINT pt;

HPEN hPen, hPenOld;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

hPen = CreatePen(PS\_SOLID, 1, red);

hPenOld = (HPEN)SelectObject(hdc, hPen);

Rectangle(hdc, 2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

Rectangle(hdc, 2 \* x1 - x2, 2 \* y1 - y2, x2, y2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

ReleaseDC(hWnd, hdc);

}

#pragma endregion RectangleEditor

#pragma region EllipseEditor

/// <summary>

/// Do something on left mouse button unclicked

/// </summary>

/// <param name="hWnd">window</param>

void EllipseEditor::OnLBup(HWND hWnd)

{

\_\_super::OnLBup(hWnd);

EllipseShape\* Ellipse = new EllipseShape;

Ellipse->Set(x1, y1, x2, y2);

pcshape[size] = Ellipse;

size++;

InvalidateRect(hWnd, NULL, TRUE);

}

/// <summary>

/// Do something on Mouse moving

/// </summary>

/// <param name="hWnd">window</param>

void EllipseEditor::OnMouseMove(HWND hWnd)

{

POINT pt;

HPEN hPen, hPenOld;

HDC hdc = GetDC(hWnd);

SetROP2(hdc, R2\_NOTXORPEN);

hPen = CreatePen(PS\_SOLID, 1, red);

hPenOld = (HPEN)SelectObject(hdc, hPen);

Ellipse(hdc, x1, y1, x2, y2);

GetCursorPos(&pt);

ScreenToClient(hWnd, &pt);

x2 = pt.x;

y2 = pt.y;

Ellipse(hdc, x1, y1, x2, y2);

SelectObject(hdc, hPenOld);

DeleteObject(hPen);

ReleaseDC(hWnd, hdc);

}

#pragma endregion EllipseEditor

#pragma endregion Functions

**Shape\_editor.h:**

#pragma once

#include "pch.h"

#include "editor.h"

#include "Resource.h"

#pragma region Editors

/// <summary>

/// Shape editor class for figures

/// </summary>

class ShapeEditor **:** public Editor

**{**

protected**:**

long x1**,** x2**,** y1**,** y2**;**

public**:**

void OnLBdown**(**HWND**);**

void OnLBup**(**HWND**);**

void OnMouseMove**(**HWND**);**

void OnPaint**(**HWND**);**

**};**

/// <summary>

/// Shape editor class for figure objects

/// </summary>

class ShapeObjectsEditor

**{**

private**:**

ShapeEditor**\*** pse**;**

public**:**

ShapeObjectsEditor**(**void**);**

**~**ShapeObjectsEditor**();**

void StartPointEditor**();**

void StartLineEditor**();**

void StartRectangleEditor**();**

void StartEllipseEditor**();**

void OnLBdown**(**HWND**);**

void OnLBup**(**HWND**);**

void OnMouseMove**(**HWND**);**

void OnPaint(HWND);

};

/// <summary>

/// Point editor class for points

/// </summary>

class PointEditor : public ShapeEditor

{

public:

void OnLBup(HWND);

};

/// <summary>

/// Line editor class for lines

/// </summary>

class LineEditor : public ShapeEditor

{

public:

void OnLBup(HWND);

void OnMouseMove(HWND);

};

/// <summary>

/// Rectangle editor class for rectangles

/// </summary>

class RectangleEditor : public ShapeEditor

{

public:

void OnLBup(HWND);

void OnMouseMove(HWND);

};

/// <summary>

/// Ellipse editor class for ellipses

/// </summary>

class EllipseEditor : public ShapeEditor

{

public:

void OnLBup(HWND);

void OnMouseMove(HWND);

};

#pragma endregion Editors

**Editor.h:**

#pragma once

#include "pch.h"

/// <summary>

/// Main interface

/// </summary>

class Editor

**{**

public**:**

virtual void OnLBdown**(**HWND**)** **=** 0**;**

virtual void OnLBup**(**HWND**)** **=** 0**;**

virtual void OnMouseMove**(**HWND**)** **=** 0**;**

virtual void OnPaint**(**HWND**)** **=** 0**;**

**};**

Editor

public:

* OnLBdown = 0
* OnLBup = 0
* OnMouseMove = 0
* OnPaint = 0

ShapeEditor

protected:

* x1, x2, y1, y2

public:

* OnLBdown
* OnLBup
* OnMouseMove
* OnPaint

PointEditor

public:

* OnLBup
* OnMouseMove

Shape

protected:

* xs1, ys1, xs2, ys2

public:

* Set
* Show = 0

Базовий клас

EllipseShape

public:

* Show

PointEditor

public:

* OnLBup

LineShape

public:

* Show

LineEditor

public:

* OnLBup
* OnMouseMove

Похідні класи

RectangleShape

public:

* Show

EllipseEditor

PointShape

public:

* OnLBup
* OnMouseMove

Похідні класи

ShapeObjectsEditor

public:

* Show

private:

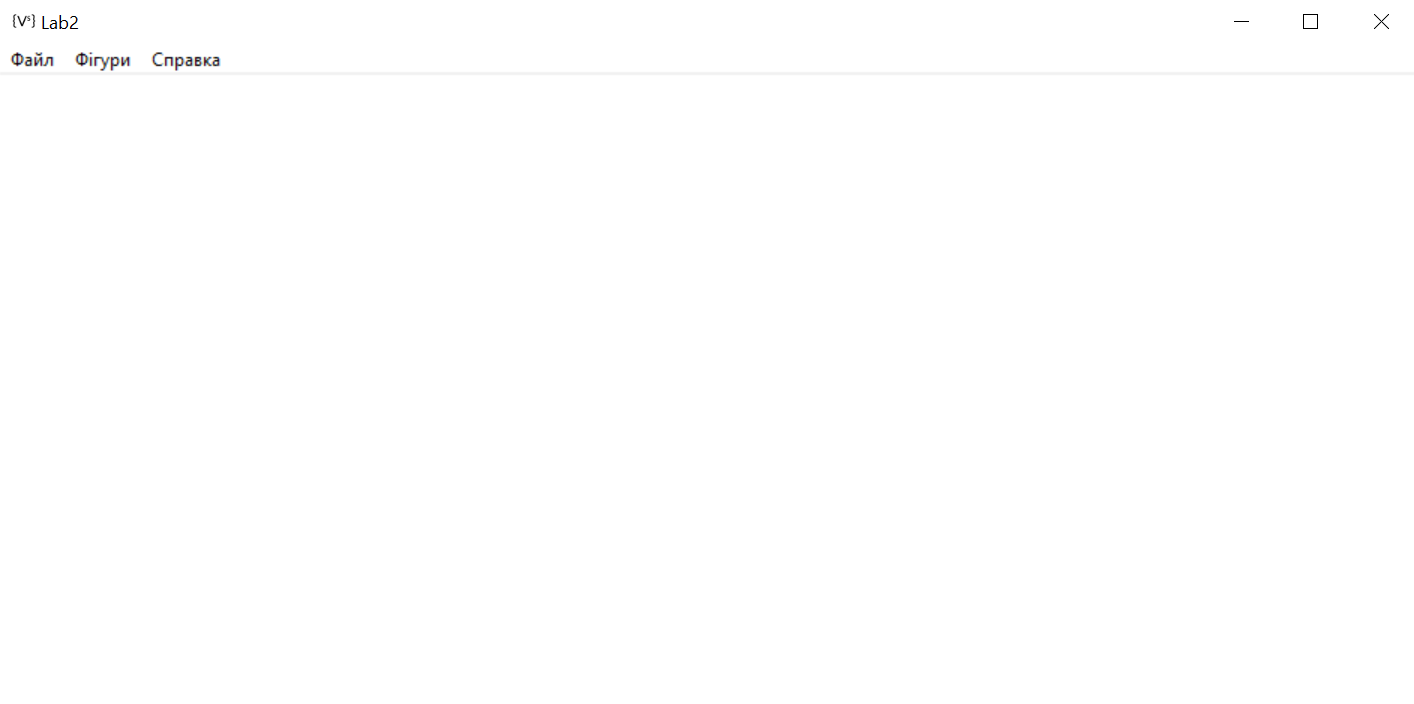
* pse
* ShapeEllipseEditor
* OnLBdown
* OnLBup
* OnMouseMove
* OnPaint

public:

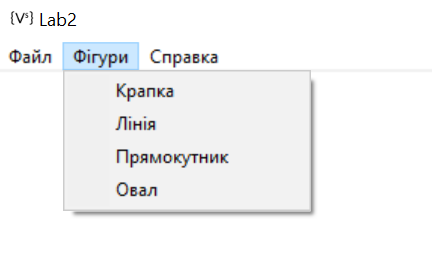
* ShapeObjectsEditor
* ~ShapeObjectsEditor
* StartPointEditor
* StartLineEditor
* StartRectangleEditor

**Скріншоти програми:**

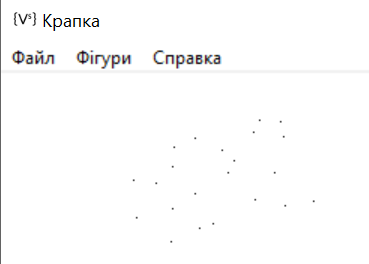
Початкове вікно:



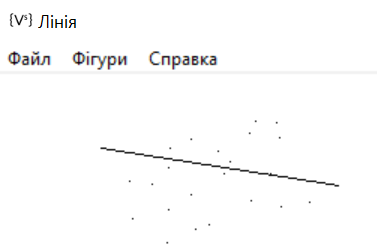
Вибір фігури:



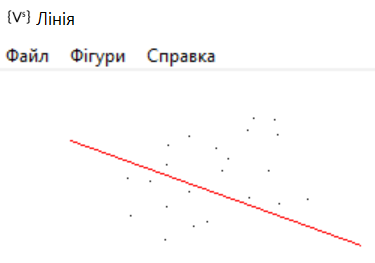
Введення крапок:



Введення ліній:



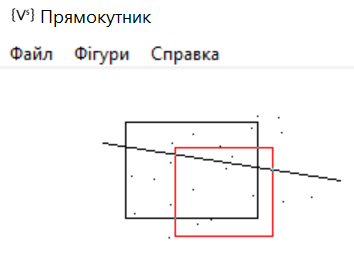
Гумовий слід ліній:



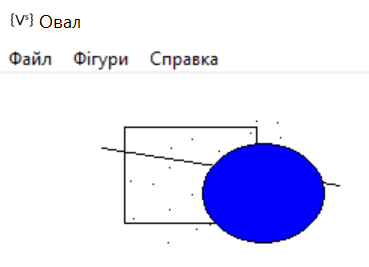
Введення прямокутників:



Гумовий слід прямокутників:



Введення еліпсів (овалів):



Гумовий слід еліпсів (овалів):



Також разом з іншими файлами є анімація (.gif) роботи програми

**Контрольні питання**

1. Що таке поліморфізм?

Поліморфі́зм— концепція в програмуванні та теорії типів, в основі якої лежить використання єдиного інтерфейсу для різнотипних сутностей або у використанні однакового символу для маніпуляцій над даними різного типу.

1. Обробку яких повідомлень потрібно виконувати для вводу об’єктів?

Початок вводу об’єктів (коли натискаєш на пункт у меню «Фігури»), натискання/відпускання лівої кнопки миші, рух миші

1. Що таке абстрактний клас і скільки іх у цій програмі?

Абстрактний клас – це базовий клас, від якого не можна створити екземпляру. В абстрактному класі можна описати абстрактні методи та властивості

У Нас їх три: Shape, ShapeEditor та Editor.

1. Як намалювати лінії та фігури потрібного кольору та стилю?

Для лінії:

hPen = CreatePen(Style, YourWidth, YourColor)

hPenOld = (HPEN)SelectObject(hdc, hPen)

SelectObject(hdc, hPenOld)

DeleteObject(hPen)

(Це підходить й для інших фігур)

Для точки:

*SetPixel(hdc, xs1, ys1, YourColor)*

1. Як відобразити щось у вікні програми?

За допомогою Show(), який Ми написали у програмі.

**Висновок:**

Навчився малювати фігури. Ознайомився з ООП, абстрактними класами, рівнями захисту, створенням класів.