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

«Об’єктно-орієнтоване програмування»

Виконав:

Студент групи ІМ-42

Марченко Даніїл

Олександрович

номер у списку групи: 20

Перевірив:

Порєв В. П.

Київ 2025

Варіанти завдань та основні вимоги

1. У звіті повинна бути схема успадкування класів – діаграма класів

2. Усі методи-обробники повідомлень, зокрема, і метод OnNotify, повинні бути функціями-членами деякого класу (класів).

3. Для вибору типу об’єкту в графічному редакторі Lab3 повинно бути вікно Toolbar з кнопками відповідно типам об'єктів. Кнопки дублюють підпункти меню "Об’єкти". Кнопки мають бути з підказками (tooltips). Меню "Об’єкти" повинно бути праворуч меню "Файл" та ліворуч меню "Довідка". Підпункти меню "Об’єкти" містять назви геометричних форм українською мовою. Геометричні форми згідно варіанту завдання.

4. Для вибору варіанту використовується значення Ж = Жлаб2 + 1, де Жлаб2 – номер студента в журналі(20), який використовувався для попередньої лаб. роботи No2.5. Масив вказівників для динамічних об’єктів типу Shape

- динамічний масив Shape **pcshape;

~~- статичний масив Shape *pcshape[N];~~

причому, кількість елементів масиву вказівників як для статичного, так і динамічного має бути N = Ж+100. N = 120

Динамічний масив обирають студенти, у яких варіант (Ж mod 3 = 0).

Решта студентів – статичний масив. Позначка mod означає залишок від ділення.

6. "Гумовий" слід при вводі об’єктів

- суцільна лінія червоного кольору для (Ж mod 4 = 1)

7. Чотири геометричні форми (крапка, лінія, прямокутник, еліпс) можуть мати наступні різновиди вводу та відображення.

7.1. Прямокутник

Увід прямокутника:

- від центру до одного з кутів для (Ж mod 2 = 1)

Відображення прямокутника:

- чорний контур з кольоровим заповненням для (Ж mod 5 = 1 або 2)

Кольори заповнення прямокутника:

- рожевий для (Ж mod 6 = 3)

7.2. Еліпс

Увід еліпсу:

- по двом протилежним кутам охоплюючого прямокутника для варіантів (Ж mod 2 = 1)

Відображення еліпсу:

- чорний контур з білим заповненням для (Ж mod 5 = 1)

8. Позначка поточного типу об’єкту, що вводиться

- в заголовку вікна для (Ж mod 2 = 1)

Примітка. Визначення кольорів та інші параметри варіантів можуть бути змінені викладачем шляхом оголошення студентам відповідного повідомлення завчасно перед постановкою завдань.

Вихідний текст файлів

Lab3.cpp

```
#include "framework.h"
#include "Lab31.h"
#include "shape_editor.h"
#include <windowsx.h>
#include <commctrl.h>
#pragma comment(lib, "comctl32.lib")

#define MAX_LOADSTRING 100
HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

ShapeObjectsEditor* g_editor = nullptr;


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);



    INITCOMMONCONTROLSEX icex;
    icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
    icex.dwICC = ICC_BAR_CLASSES;
    InitCommonControlsEx(&icex);

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB3, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LAB3));
    MSG msg;

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

    if (g_editor)
    {
        delete g_editor;
    }

    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_LAB3));
   wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
   wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
   wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LAB3);
   wcex.lpszClassName = szWindowClass;
   wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));
   return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance;
   HWND hWnd = CreateWindowW(szWindowClass, szTitle,
WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, 960, 640, 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:
      g_editor = new ShapeObjectsEditor(hWnd, hInst);
      g_editor->CreateToolbar();
      break;
   case WM_SIZE:
      if (g_editor)
      {
```

```
        g_editor->OnSize();
    }
    break;
case WM_NOTIFY:
    g_editor->OnNotify(hWnd, wParam, lParam);
    break;
case WM_INITMENUPOPUP:
    g_editor->OnInitMenuPopup(hWnd, wParam);
    break;
case WM_COMMAND:
{
    int wmId = LOWORD(wParam);
    switch (wmId)
    {
    case IDM_ABOUT:
        DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
        break;
    case IDM_EXIT:
        DestroyWindow(hWnd);
        break;
    case IDM_OBJ_POINT:
    case IDM_TOOL_POINT:
        g_editor->StartPointEditor();
        break;
    case IDM_OBJ_LINE:
    case IDM_TOOL_LINE:
        g_editor->StartLineEditor();
        break;
    case IDM_OBJ_RECT:
    case IDM_TOOL_RECT:
        g_editor->StartRectEditor();
        break;
    case IDM_OBJ_ELLIPSE:
    case IDM_TOOL_ELLIPSE:
        g_editor->StartEllipseEditor();
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
}
break;
case WM_LBUTTONDOWN:
    g_editor->OnLDown(hWnd, GET_X_LPARAM(lParam),
```

```
GET_Y_LPARAM(lParam));
        break;
    case WM_LBUTTONUP:
        g_editor->OnLUp(hWnd, GET_X_LPARAM(lParam), GET_Y_LPARAM(lParam));
        break;
    case WM_MOUSEMOVE:
        if (wParam & MK_LBUTTON)
            g_editor->OnMouseMove(hWnd, GET_X_LPARAM(lParam),
GET_Y_LPARAM(lParam));
        break;
    case WM_PAINT:
        g_editor->OnPaint(hWnd);
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

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;
}
```

shape.cpp

```
#include "shape.h"

Shape::~Shape() = default;

void Shape::Set(LONG ax1, LONG ay1, LONG ax2, LONG ay2) {
   x1 = ax1; y1 = ay1; x2 = ax2; y2 = ay2;
}

void PointShape::Show(HDC hdc) const {
   SetPixel(hdc, x1, y1, RGB(0, 0, 0));
}

void LineShape::Show(HDC hdc) const {
   MoveToEx(hdc, x1, y1, nullptr);
   LineTo(hdc, x2, y2);
}

void RectShape::Show(HDC hdc) const {
   HBRUSH hBrush = CreateSolidBrush(RGB(255, 192, 203));
   HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
   HPEN hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
   HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);

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

   SelectObject(hdc, hOldPen);
   DeleteObject(hPen);
   SelectObject(hdc, hOldBrush);
   DeleteObject(hBrush);
}

void EllipseShape::Show(HDC hdc) const {
   HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc,
GetStockObject(WHITE_BRUSH));
   HPEN hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
   HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);

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

   SelectObject(hdc, hOldPen);
   DeleteObject(hPen);
   SelectObject(hdc, hOldBrush);
}
```

shape.h

```
#pragma once
#include <windows.h>

class Shape {
```

```
protected:
   LONG x1{}, y1{}, x2{}, y2{};
public:
   virtual ~Shape();
   void Set(LONG ax1, LONG ay1, LONG ax2, LONG ay2);
   virtual void Show(HDC hdc) const = 0;
};


class PointShape : public Shape { public: void Show(HDC hdc) const override; };
class LineShape : public Shape { public: void Show(HDC hdc) const override; };
class RectShape : public Shape { public: void Show(HDC hdc) const override; };
class EllipseShape : public Shape { public: void Show(HDC hdc) const override; };
```

editor.cpp

```
#include "editor.h"
#include <algorithm>
#include "resource.h"

void PointEditor::OnLButtonDown(HWND hWnd, LONG x, LONG y) { this->x = x; this->y = y; }
void PointEditor::OnMouseMove(HWND hWnd, LONG x, LONG y) {}
void PointEditor::OnLButtonUp(HWND hWnd, LONG x, LONG y) { this->x = x; this->y = y; }
void PointEditor::OnPaint(HWND hWnd, HDC hdc) { SetPixel(hdc, x, y, RGB(0, 0, 0)); }
void PointEditor::OnInitMenuPopup(HMENU hMenu) {
   CheckMenuItem(hMenu, IDM_OBJ_POINT, MF_BYCOMMAND | MF_CHECKED);
   CheckMenuItem(hMenu, IDM_OBJ_LINE, MF_BYCOMMAND | MF_UNCHECKED);
   CheckMenuItem(hMenu, IDM_OBJ_RECT, MF_BYCOMMAND | MF_UNCHECKED);
   CheckMenuItem(hMenu, IDM_OBJ_ELLIPSE, MF_BYCOMMAND | MF_UNCHECKED);
}

void LineEditor::OnLButtonDown(HWND hWnd, LONG x, LONG y) { this->x1 = x; this->y1 = y; }
void LineEditor::OnMouseMove(HWND hWnd, LONG x, LONG y) { this->x2 = x; this->y2 = y; }
void LineEditor::OnLButtonUp(HWND hWnd, LONG x, LONG y) { this->x2 = x; this->y2 = y; }
void LineEditor::OnPaint(HWND hWnd, HDC hdc) {
   if (x1 != x2 || y1 != y2) {
      HPEN hPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
      HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
      MoveToEx(hdc, x1, y1, nullptr);
      LineTo(hdc, x2, y2);
```

```
        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);
    }
}
void LineEditor::OnInitMenuPopup(HMENU hMenu) {
    CheckMenuItem(hMenu, IDM_OBJ_POINT, MF_BYCOMMAND |
MF_UNCHECKED);
    CheckMenuItem(hMenu, IDM_OBJ_LINE, MF_BYCOMMAND | MF_CHECKED);
    CheckMenuItem(hMenu, IDM_OBJ_RECT, MF_BYCOMMAND |
MF_UNCHECKED);
    CheckMenuItem(hMenu, IDM_OBJ_ELLIPSE, MF_BYCOMMAND |
MF_UNCHECKED);
}

void RectEditor::OnLButtonDown(HWND hWnd, LONG x, LONG y) { this->x1 = x;
this->y1 = y; }
void RectEditor::OnMouseMove(HWND hWnd, LONG x, LONG y) { this->x2 = x; this-
>y2 = y; }
void RectEditor::OnLButtonUp(HWND hWnd, LONG x, LONG y) { this->x2 = x; this-
>y2 = y; }
void RectEditor::OnPaint(HWND hWnd, HDC hdc) {
    if (x1 != x2 || y1 != y2) {
        HPEN hPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
        HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
        HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc,
GetStockObject(NULL_BRUSH));

        LONG dx = abs(x2 - x1);
        LONG dy = abs(y2 - y1);
        Rectangle(hdc, x1 - dx, y1 - dy, x1 + dx, y1 + dy);


        SelectObject(hdc, hOldPen);
        DeleteObject(hPen);
        SelectObject(hdc, hOldBrush);
    }
}
void RectEditor::OnInitMenuPopup(HMENU hMenu) {
    CheckMenuItem(hMenu, IDM_OBJ_POINT, MF_BYCOMMAND |
MF_UNCHECKED);
    CheckMenuItem(hMenu, IDM_OBJ_LINE, MF_BYCOMMAND |
MF_UNCHECKED);
    CheckMenuItem(hMenu, IDM_OBJ_RECT, MF_BYCOMMAND | MF_CHECKED);
    CheckMenuItem(hMenu, IDM_OBJ_ELLIPSE, MF_BYCOMMAND |
MF_UNCHECKED);
}

void EllipseEditor::OnLButtonDown(HWND hWnd, LONG x, LONG y) { this->x1 = x;
this->y1 = y; }
```

```
void EllipseEditor::OnMouseMove(HWND hWnd, LONG x, LONG y) { this->x2 = x;
this->y2 = y; }
void EllipseEditor::OnLButtonUp(HWND hWnd, LONG x, LONG y) { this->x2 = x; this-
>y2 = y; }
void EllipseEditor::OnPaint(HWND hWnd, HDC hdc) {
   if (x1 != x2 || y1 != y2) {
      HPEN hPen = CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
      HPEN hOldPen = (HPEN)SelectObject(hdc, hPen);
      HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc,
GetStockObject(NULL_BRUSH));
      Ellipse(hdc, x1, y1, x2, y2);
      SelectObject(hdc, hOldPen);
      DeleteObject(hPen);
      SelectObject(hdc, hOldBrush);
   }
}
void EllipseEditor::OnInitMenuPopup(HMENU hMenu) {
   CheckMenuItem(hMenu, IDM_OBJ_POINT, MF_BYCOMMAND |
MF_UNCHECKED);
   CheckMenuItem(hMenu, IDM_OBJ_LINE, MF_BYCOMMAND |
MF_UNCHECKED);
   CheckMenuItem(hMenu, IDM_OBJ_RECT, MF_BYCOMMAND |
MF_UNCHECKED);
   CheckMenuItem(hMenu, IDM_OBJ_ELLIPSE, MF_BYCOMMAND |
MF_CHECKED);
}

void PointEditor::Reset() { x = 0; y = 0; }
void LineEditor::Reset() { x1 = 0; y1 = 0; x2 = 0; y2 = 0; }
void RectEditor::Reset() { x1 = 0; y1 = 0; x2 = 0; y2 = 0; }
void EllipseEditor::Reset() { x1 = 0; y1 = 0; x2 = 0; y2 = 0; }
```

editor.h

```
#pragma once
#include <windows.h>

class Editor {
public:
    virtual ~Editor() = default;
    virtual void OnLButtonDown(HWND hWnd, LONG x, LONG y) = 0;
    virtual void OnMouseMove(HWND hWnd, LONG x, LONG y) = 0;
    virtual void OnLButtonUp(HWND hWnd, LONG x, LONG y) = 0;
    virtual void OnPaint(HWND hWnd, HDC hdc) = 0;
};
class ShapeEditor : public Editor {
public:
    ShapeEditor() = default;
    virtual void OnLButtonDown(HWND hWnd, LONG x, LONG y) override = 0;
    virtual void OnMouseMove(HWND hWnd, LONG x, LONG y) override = 0;
    virtual void OnLButtonUp(HWND hWnd, LONG x, LONG y) override = 0;
    virtual void OnPaint(HWND hWnd, HDC hdc) override = 0;
    virtual void OnInitMenuPopup(HMENU hMenu) = 0;
    virtual void Reset() = 0;
};
class PointEditor : public ShapeEditor {
private:
    LONG x = 0, y = 0;
public:
    void OnLButtonDown(HWND hWnd, LONG x, LONG y) override;
    void OnMouseMove(HWND hWnd, LONG x, LONG y) override;
    void OnLButtonUp(HWND hWnd, LONG x, LONG y) override;
    void OnPaint(HWND hWnd, HDC hdc) override;
    void OnInitMenuPopup(HMENU hMenu) override;
```

```
    void Reset() override;
};
class LineEditor : public ShapeEditor {
private:
    LONG x1 = 0, y1 = 0, x2 = 0, y2 = 0;
public:
    void OnLButtonDown(HWND hWnd, LONG x, LONG y) override;
    void OnMouseMove(HWND hWnd, LONG x, LONG y) override;
    void OnLButtonUp(HWND hWnd, LONG x, LONG y) override;
    void OnPaint(HWND hWnd, HDC hdc) override;
    void OnInitMenuPopup(HMENU hMenu) override;
    void Reset() override;
};
class RectEditor : public ShapeEditor {
private:
    LONG x1 = 0, y1 = 0, x2 = 0, y2 = 0;
public:
    void OnLButtonDown(HWND hWnd, LONG x, LONG y) override;
    void OnMouseMove(HWND hWnd, LONG x, LONG y) override;
    void OnLButtonUp(HWND hWnd, LONG x, LONG y) override;
    void OnPaint(HWND hWnd, HDC hdc) override;
    void OnInitMenuPopup(HMENU hMenu) override;
    void Reset() override;
};
class EllipseEditor : public ShapeEditor {
private:
    LONG x1 = 0, y1 = 0, x2 = 0, y2 = 0;
public:
    void OnLButtonDown(HWND hWnd, LONG x, LONG y) override;
    void OnMouseMove(HWND hWnd, LONG x, LONG y) override;
    void OnLButtonUp(HWND hWnd, LONG x, LONG y) override;
```

```
    void OnPaint(HWND hWnd, HDC hdc) override;
    void OnInitMenuPopup(HMENU hMenu) override;
    void Reset() override;
};
```

shape_editor.cpp

```
#include "shape_editor.h"
#include "Lab31.h"
#include <windowsx.h>
#include <algorithm>
#include <string>
#include <commctrl.h>

ShapeObjectsEditor::ShapeObjectsEditor(HWND hWnd, HINSTANCE hInst) {
    m_hWnd = hWnd;
    m_hInst = hInst;
    m_max_objects = 121;
    m_objects = new Shape * [m_max_objects];
    for (int i = 0; i < m_max_objects; ++i) {
        m_objects[i] = nullptr;
    }
}

ShapeObjectsEditor::~ShapeObjectsEditor() {
    for (int i = 0; i < m_count; ++i) {
        if (m_objects[i]) {
            delete m_objects[i];
        }
    }
    delete[] m_objects;
    if (m_currentEditor) {
        delete m_currentEditor;
    }
}

void ShapeObjectsEditor::StartPointEditor() {
    if (m_count < m_max_objects) {
        if (m_currentEditor) delete m_currentEditor;
        m_currentEditor = new PointEditor();
        m_shapeFactory = []() { return new PointShape(); };
```

```
        SetWindowText(m_hWnd, L"Режим: Крапка");
        InvalidateRect(m_hWnd, nullptr, TRUE);
        if (m_hwndToolBar) {
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_POINT, TRUE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_LINE, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_RECT, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_ELLIPSE, FALSE);
        }
    }
}

void ShapeObjectsEditor::StartLineEditor() {
    if (m_count < m_max_objects) {
        if (m_currentEditor) delete m_currentEditor;
        m_currentEditor = new LineEditor();
        m_shapeFactory = []() { return new LineShape(); };
        SetWindowText(m_hWnd, L"Режим: Лінія");
        InvalidateRect(m_hWnd, nullptr, TRUE);
        if (m_hwndToolBar) {
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_POINT, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_LINE, TRUE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_RECT, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_ELLIPSE, FALSE);
        }
    }
}

void ShapeObjectsEditor::StartRectEditor() {
    if (m_count < m_max_objects) {
        if (m_currentEditor) delete m_currentEditor;
        m_currentEditor = new RectEditor();
        m_shapeFactory = []() { return new RectShape(); };
```

```
        SetWindowText(m_hWnd, L"Режим: Прямокутник");
        InvalidateRect(m_hWnd, nullptr, TRUE);
        if (m_hwndToolBar) {
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_POINT, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_LINE, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_RECT, TRUE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_ELLIPSE, FALSE);
        }
    }
}

void ShapeObjectsEditor::StartEllipseEditor() {
    if (m_count < m_max_objects) {
        if (m_currentEditor) delete m_currentEditor;
        m_currentEditor = new EllipseEditor();
        m_shapeFactory = []() { return new EllipseShape(); };
        SetWindowText(m_hWnd, L"Режим: Еліпс");
        InvalidateRect(m_hWnd, nullptr, TRUE);
        if (m_hwndToolBar) {
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_POINT, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_LINE, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_RECT, FALSE);
            SendMessage(m_hwndToolBar, TB_CHECKBUTTON,
IDM_TOOL_ELLIPSE, TRUE);
        }
    }
}

void ShapeObjectsEditor::OnLDown(HWND hWnd, int x, int y) {
    if (m_currentEditor) {
        x0 = x;
        y0 = y;
        m_currentEditor->OnLButtonDown(hWnd, x, y);
    }
```

```
}

void ShapeObjectsEditor::OnMouseMove(HWND hWnd, int x, int y) {
    if (m_currentEditor) {
        m_currentEditor->OnMouseMove(hWnd, x, y);
        InvalidateRect(hWnd, nullptr, TRUE);
    }
}

void ShapeObjectsEditor::OnLUp(HWND hWnd, int x, int y) {
    if (m_currentEditor && m_shapeFactory) {
        if (m_count >= m_max_objects) return;


        Shape* newShape = m_shapeFactory();


        if (dynamic_cast<RectEditor*>(m_currentEditor)) {
            LONG dx = abs(x - x0);
            LONG dy = abs(y - y0);
            newShape->Set(x0 - dx, y0 - dy, x0 + dx, y0 + dy);
        }
        else {
            newShape->Set(x0, y0, x, y);
        }

        m_objects[m_count++] = newShape;
        m_currentEditor->OnLButtonUp(hWnd, x, y);
        m_currentEditor->Reset();
    }
    InvalidateRect(hWnd, nullptr, TRUE);
}

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

```
    }
    if (m_currentEditor) {
        m_currentEditor->OnPaint(hWnd, hdc);
    }
    EndPaint(hWnd, &ps);
}

void ShapeObjectsEditor::OnInitMenuPopup(HWND, WPARAM wParam) {
    if (m_currentEditor) {
        m_currentEditor->OnInitMenuPopup((HMENU)wParam);
    }
}

void ShapeObjectsEditor::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 IDM_TOOL_POINT:
            lstrcpy(lpttt->szText, L"Крапка");
            break;
        case IDM_TOOL_LINE:
            lstrcpy(lpttt->szText, L"Лінія");
            break;
        case IDM_TOOL_RECT:
            lstrcpy(lpttt->szText, L"Прямокутник");
            break;
        case IDM_TOOL_ELLIPSE:
            lstrcpy(lpttt->szText, L"Еліпс");
            break;
        }
    }
}
void ShapeObjectsEditor::CreateToolbar()
{
    m_hwndToolBar = CreateWindowEx(0, TOOLBARCLASSNAME, NULL,
```

```
        WS_CHILD | WS_VISIBLE | WS_BORDER | TBSTYLE_TOOLTIPS,
        0, 0, 0, 0,
        m_hWnd, (HMENU)1, m_hInst, NULL);

    if (!m_hwndToolBar) return;

    SendMessage(m_hwndToolBar, TB_BUTTONSTRUCTSIZE,
(WPARAM)sizeof(TBBUTTON), 0);

    TBADDBITMAP tbab;
    tbab.hInst = m_hInst;
    tbab.nID = IDB_BITMAP1;
    SendMessage(m_hwndToolBar, TB_ADDBITMAP, 4, (LPARAM)&tbab);

    TBBUTTON tbb[4];
    ZeroMemory(tbb, sizeof(tbb));

    tbb[0] = { 0, IDM_TOOL_POINT,   TBSTATE_ENABLED,
TBSTYLE_BUTTON, {0}, 0, (INT_PTR)L"Крапка" };
    tbb[1] = { 1, IDM_TOOL_LINE,    TBSTATE_ENABLED,
TBSTYLE_BUTTON, {0}, 0, (INT_PTR)L"Лінія" };
    tbb[2] = { 2, IDM_TOOL_RECT,    TBSTATE_ENABLED,
TBSTYLE_BUTTON, {0}, 0, (INT_PTR)L"Прямокутник" };
    tbb[3] = { 3, IDM_TOOL_ELLIPSE, TBSTATE_ENABLED,
TBSTYLE_BUTTON, {0}, 0, (INT_PTR)L"Еліпс" };

    SendMessage(m_hwndToolBar, TB_ADDBUTTONS, 4, (LPARAM)&tbb);
}

void ShapeObjectsEditor::OnSize()
{
    if (m_hwndToolBar)
    {
        SendMessage(m_hwndToolBar, WM_SIZE, 0, 0);
    }
}
```

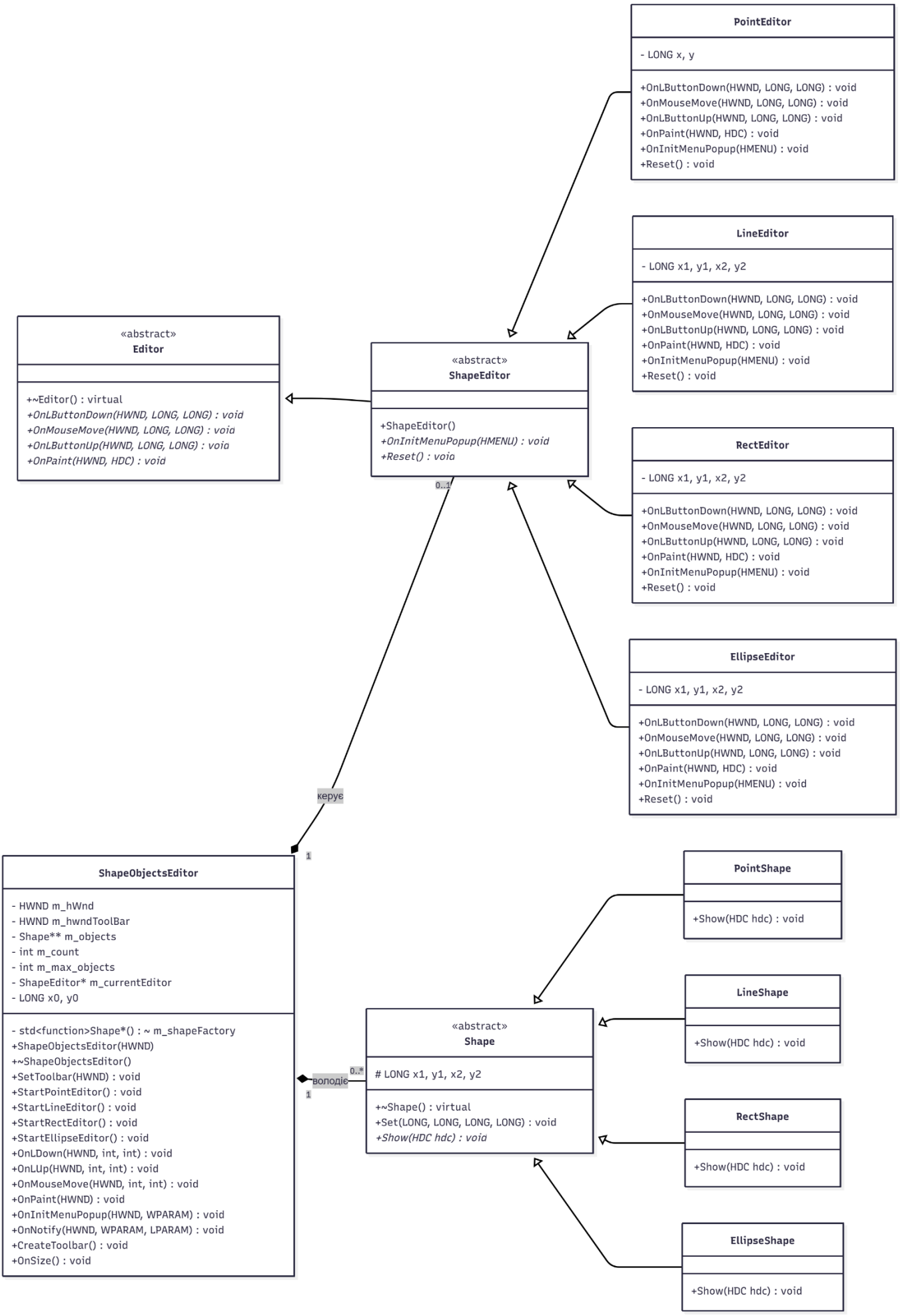
shape_editor.h

```
#pragma once
#include <windows.h>
```

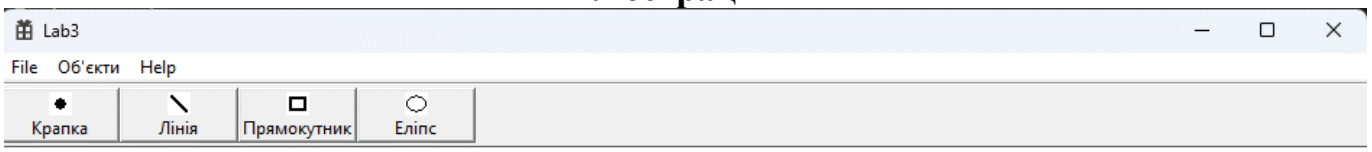
```
#include "editor.h"
#include "shape.h"
#include <functional>


class ShapeObjectsEditor {
public:
    ShapeObjectsEditor(HWND hWnd, HINSTANCE hInst);
    ~ShapeObjectsEditor();

    void CreateToolbar();
    void OnSize();

    void StartPointEditor();
    void StartLineEditor();
    void StartRectEditor();
    void StartEllipseEditor();

    void OnLDown(HWND hWnd, int x, int y);
    void OnLUp(HWND hWnd, int x, int y);
    void OnMouseMove(HWND hWnd, int x, int y);
    void OnPaint(HWND hWnd);
    void OnInitMenuPopup(HWND hWnd, WPARAM wParam);
    void OnNotify(HWND hWnd, WPARAM wParam, LPARAM lParam);

private:
    HWND m_hwndToolBar = NULL;
    HINSTANCE m_hInst = NULL;
    Shape** m_objects = nullptr;
    int m_count = 0;
    int m_max_objects = 0;
```

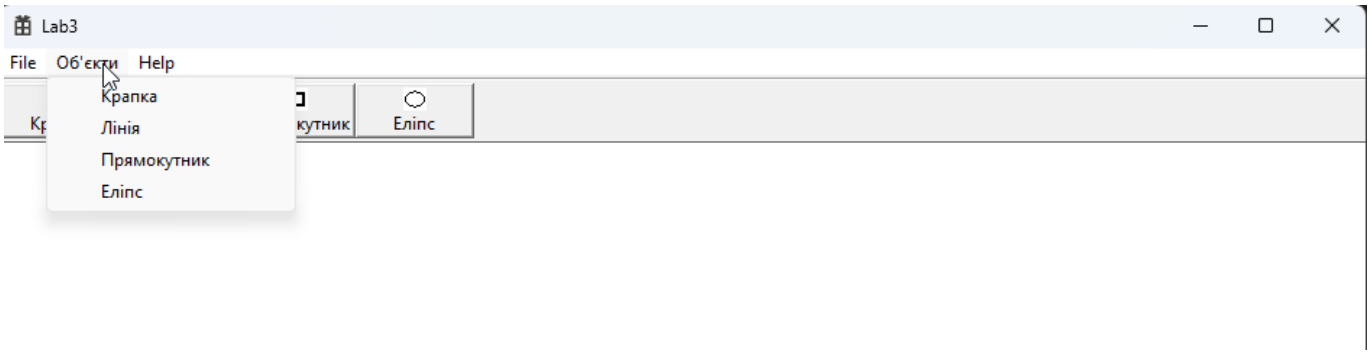
```
    std::function<Shape* ()> m_shapeFactory;

    LONG x0{}, y0{};
    ShapeEditor* m_currentEditor = nullptr;
    HWND m_hWnd = nullptr;
};
```

Діаграма класів

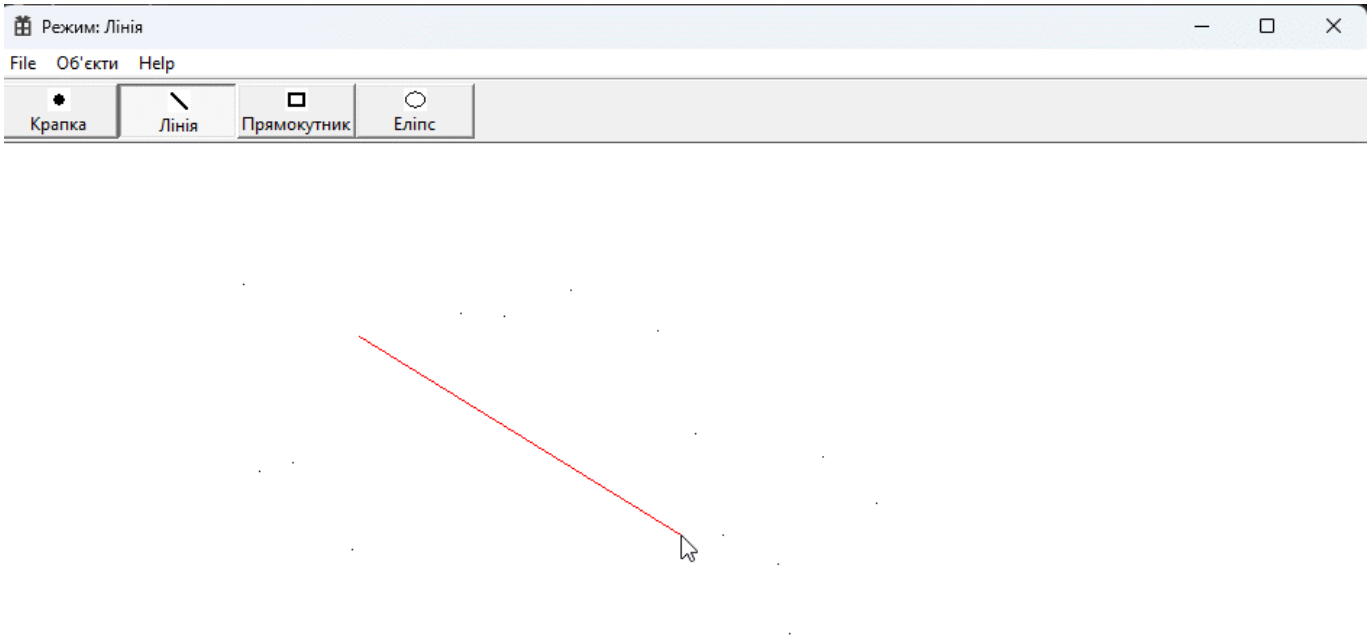


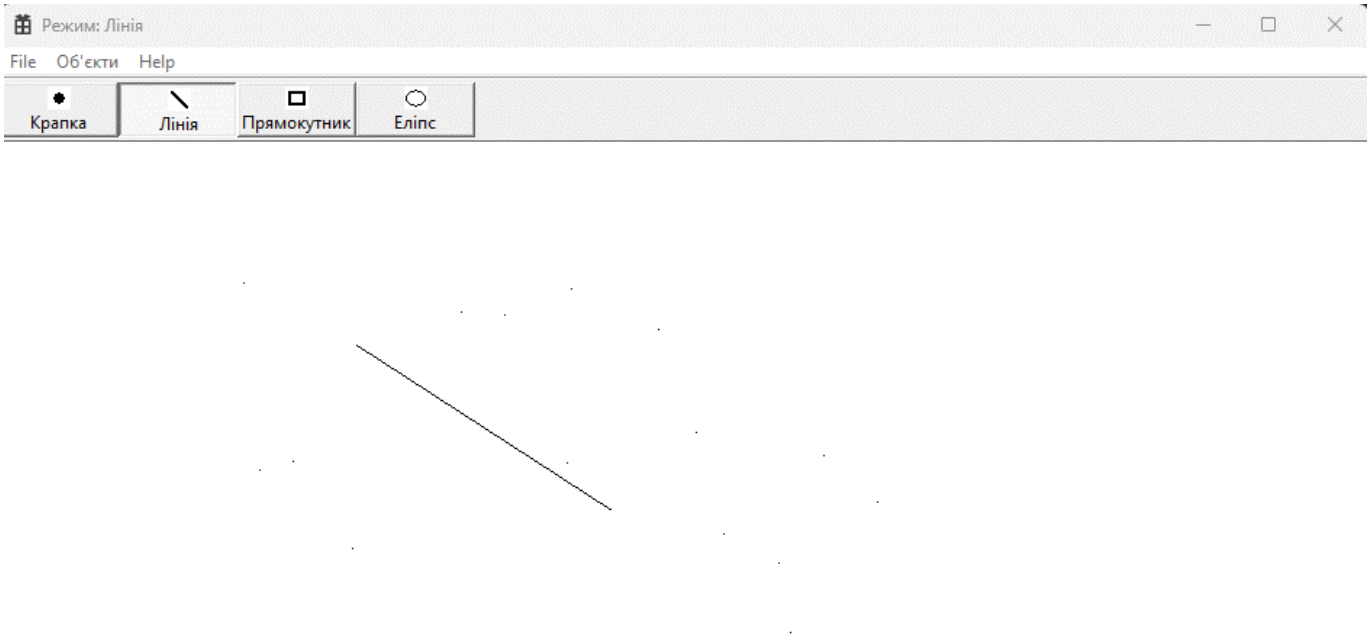
Ілюстрації

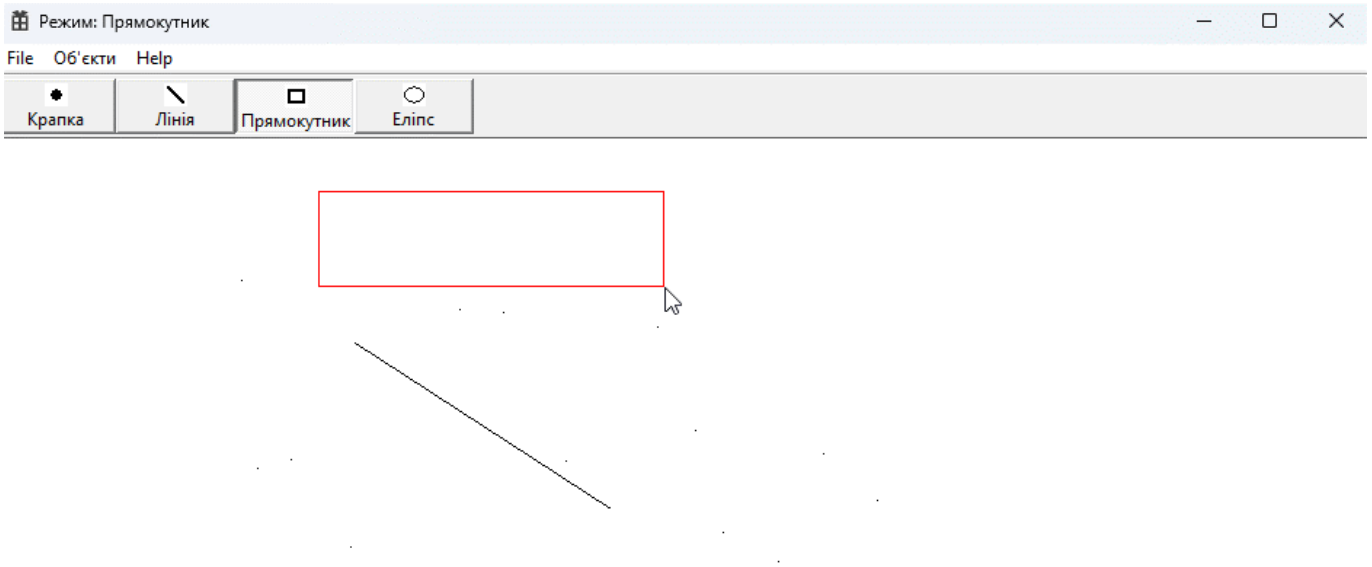

Lab3
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс

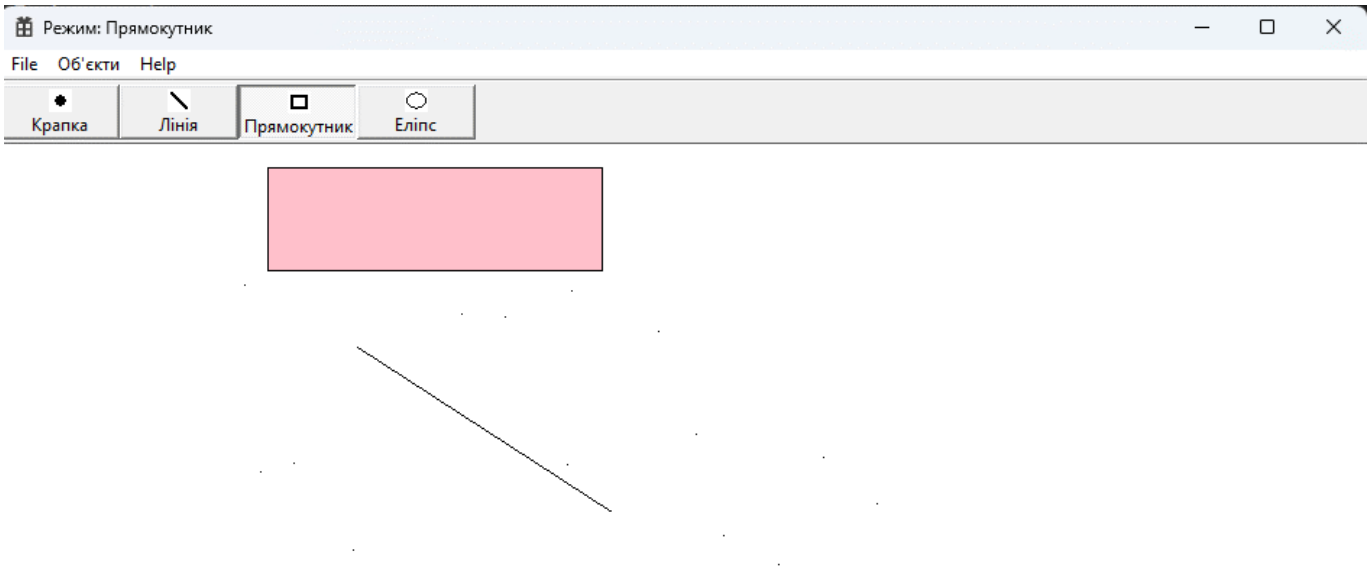

Lab3
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс
Еліпс


Режим: Крапка
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс

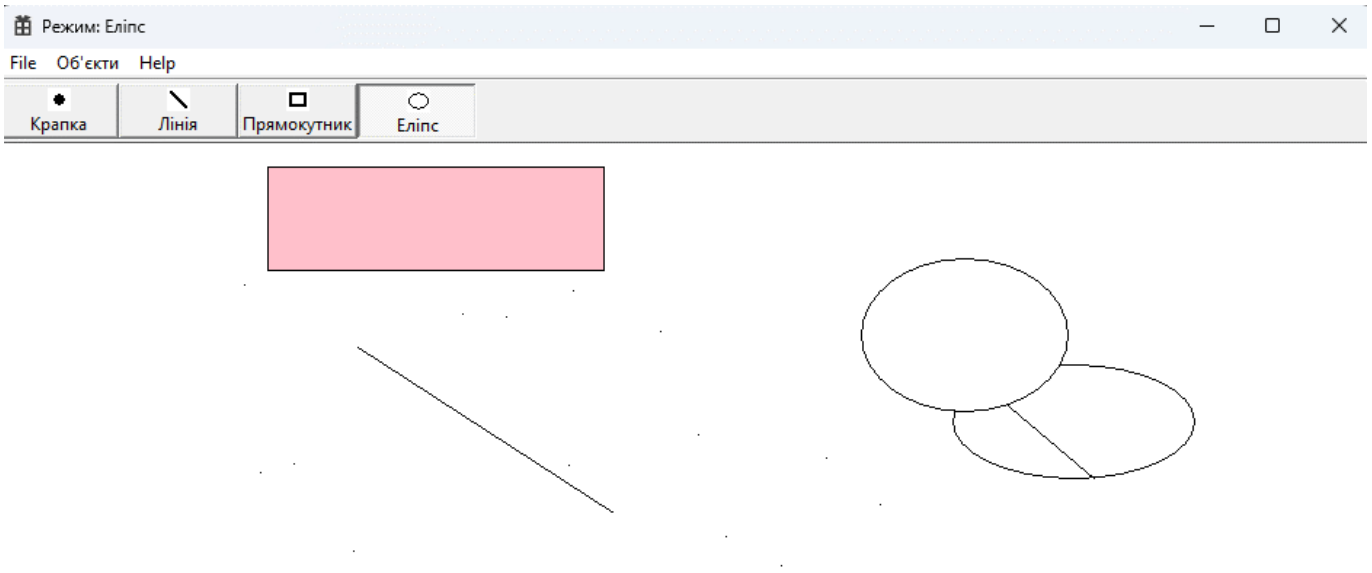

Режим: Лінія
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс

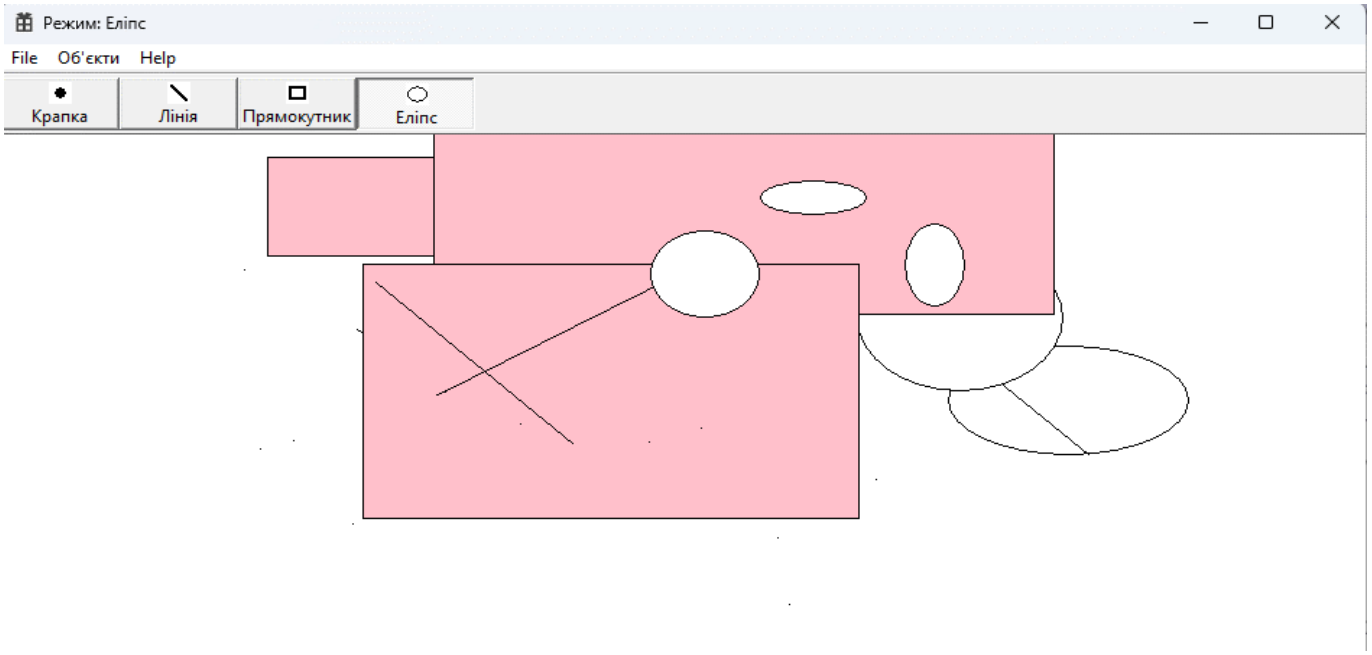

Режим: Лінія
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс


Режим: Прямокутник
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс


Режим: Прямокутник
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Еліпс


Режим: Елінс
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Елінс


Режим: Елінс
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Елінс


Режим: Елінс
File
Об'єкти
Help
Крапка
Лінія
Прямокутник
Елінс

Висновок

В ході виконання лабораторної роботи №3 “Розробка інтерфейсу користувача на C++” було отримано вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів мови C++, в ході програмування графічного інтерфейсу користувача. Також навчилися створювати власні іконки в Bitmap та використовувати їх в toolbar, який в свою чергу ми також навчилися створювати.