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

«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. Для усіх варіантів завдань необхідно дотримуватися вимог та положень, викладених вище у порядку виконання роботи та методичних рекомендаціях.

2. Номер варіанту завдання дорівнює номеру зі списку студентів у журналі.

Студенти з **непарним** номером (1, 3, 5, . . .) програмують глобальний статичний об'єкт класу MyEditor у вигляді **Singleton Меєрса**.

Студенти з **парним** номером (2, 4, 6, . . .) програмують об'єкт класу MyEditor на основі **класичної реалізації Singleton**.

3. Усі кольори та стилі геометричних форм – як у попередньої лаб. роботі №4.

4. Запрограмувати вікно таблиці. Для його відкриття та закриття передбачити окремий пункт меню. Вікно таблиці повинно автоматично закриватися при виході з програми.

5. Вікно таблиці – немодальне вікно діалогу. Таблиця повинна бути запрограмована як клас у окремому модулі. Інтерфейс модуля у вигляді оголошення класу таблиці

6. Запрограмувати запис файлу множини об'єктів, що вводяться

7. Оголошення класів для усіх типів об'єктів робити у окремих заголовочних файлах *.h, а визначення функцій членів – у окремих файлах *.cpp. Таким чином, програмний код для усіх наявних типів об'єктів розподіляється по множині окремих модулів.

8. Ієрархія класів та побудова модулів повинні бути зручними для можливостей додавання нових типів об'єктів без переписування коду вже існуючих модулів.

9. У звіті повинна бути схема успадкування класів – діаграма класів. Побудувати діаграму класів засобами Visual Studio C++.

11. **Бонуси-заохочення**, які можуть суттєво підвищити оцінку лабораторної роботи. Оцінка підвищується за виконання кожного пункту, з наведених нижче:

1). Якщо у вікні таблиці буде передбачено, щоб користувач міг виділити курсором рядок таблиці і відповідний об'єкт буде якось виділятися на зображенні у головному вікні.

2). Якщо у вікні таблиці користувач може виділити курсором рядок таблиці і відповідний об'єкт буде вилучено з масиву об'єктів.

При виконанні бонусів 1 та 2 забороняється робити для цього нові залежності модуля **my_table** від інших .cpp файлів. Тоді як надіслати повідомлення (наприклад, про виділення користувачем якогось рядка таблиці) від вікна таблиці клієнту цього вікна (наприклад, коду головного файлу .cpp)? Підказки можна знайти у матеріалі лекції стосовно технології **Callback,** а також патернів Observer, Listener.

3). Якщо програма не тільки записує у файл опис множини об'єктів, а ще й здатна завантажити такий файл і відобразити відповідні об'єкти у головному вікні та вікні таблиці

Вихідний текст файлів

Lab5.cpp

```
#include "framework.h"
#include "resource.h"
#include "my_editor.h"
#include "my_table.h"
#include <windowsx.h>
#include <commctrl.h>
#include <locale>
#pragma comment(lib, "comctl32.lib")

#define MAX_LOADSTRING 100

HINSTANCE hInst;
WCHAR szTitle[MAX_LOADSTRING];
WCHAR szWindowClass[MAX_LOADSTRING];

MyEditor* g_editor = nullptr;
MyTable g_table;
bool isTableVisible = false;

ATOM             MyRegisterClass(HINSTANCE hInstance);
BOOL             InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
void             TableEventsCallback(TableAction action, int index);
void             EditorEventsCallback(EditorAction action, const wchar_t* shapeName,
LONG x1, LONG y1, LONG x2, LONG y2, int index);

int APIENTRY wWinMain(_In_ HINSTANCE hInstance, _In_opt_ HINSTANCE
hPrevInstance, _In_ LPWSTR lpCmdLine, _In_ int nCmdShow) {
   std::locale::global(std::locale(""));
   UNREFERENCED_PARAMETER(hPrevInstance);
UNREFERENCED_PARAMETER(lpCmdLine);
   LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
   LoadStringW(hInstance, IDC_LAB51, szWindowClass, MAX_LOADSTRING);
   MyRegisterClass(hInstance);
   if (!InitInstance(hInstance, nCmdShow)) return FALSE;
   HACCEL hAccelTable = LoadAccelerators(hInstance,
MAKEINTRESOURCE(IDC_LAB51));
   MSG msg;
   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_LAB51));
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW); wcex.hbrBackground =
(HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = MAKEINTRESOURCEW(IDC_LAB51);
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 = MyEditor::getInstance(hWnd);
        g_table.RegisterCallback(TableEventsCallback);
        g_editor->RegisterCallback(EditorEventsCallback);
        g_editor->CreateToolbar(hInst);
        g_editor->Start(new PointShape(), IDM_OBJ_POINT);
        break;
    case WM_SIZE:
        g_editor->OnSize();
```

```
        break;
    case WM_NOTIFY:
        g_editor->OnNotify(hWnd, wParam, lParam);
        break;
    case WM_INITMENUPOPUP: {
        HMENU hMenu = GetMenu(hWnd);
        if (hMenu) {
            HMENU hSubMenu = GetSubMenu(hMenu, 1);
            if ((HMENU)wParam == hSubMenu) g_editor-
>OnInitMenuPopup((HMENU)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_VIEW_TABLE:
            if (!isTableVisible) {
                g_table.Show(hInst, hWnd); isTableVisible = true; g_table.Clear();
                int count = g_editor->GetCount();
                for (int i = 0; i < count; ++i) {
                    const Shape* shape = g_editor->GetShape(i);
                    if (shape) {
                        LONG x1, y1, x2, y2; shape->GetCoords(x1, y1, x2, y2);
                        g_table.AddEntry(shape->GetName(), x1, y1, x2, y2);
                    }
                }
            }
            break;
        case IDM_OBJ_POINT:   case IDM_TOOL_POINT:   g_editor->Start(new
PointShape(), IDM_OBJ_POINT); break;
        case IDM_OBJ_LINE:    case IDM_TOOL_LINE:    g_editor->Start(new
LineShape(), IDM_OBJ_LINE);  break;
        case IDM_OBJ_RECT:    case IDM_TOOL_RECT:    g_editor->Start(new
RectShape(), IDM_OBJ_RECT);  break;
        case IDM_OBJ_ELLIPSE: case IDM_TOOL_ELLIPSE: g_editor->Start(new
EllipseShape(), IDM_OBJ_ELLIPSE); break;
        case IDM_OBJ_LINEOO:  case IDM_TOOL_LINEOO:  g_editor->Start(new
LineOOShape(), IDM_OBJ_LINEOO); break;
        case IDM_OBJ_CUBE:    case IDM_TOOL_CUBE:    g_editor->Start(new
CubeShape(), IDM_OBJ_CUBE); 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;
}

void TableEventsCallback(TableAction action, int index) {
    switch (action) {
    case TABLE_ACTION_SELECT: g_editor->SelectShape(index); break;
    case TABLE_ACTION_DELETE: g_editor->DeleteShape(index); break;
    case TABLE_ACTION_CLOSE: isTableVisible = false; break;
    }
}
void EditorEventsCallback(EditorAction action, const wchar_t* shapeName, LONG x1,
LONG y1, LONG x2, LONG y2, int index) {
    switch (action) {
    case EDITOR_ACTION_ADD: g_table.AddEntry(shapeName, x1, y1, x2, y2); break;
    case EDITOR_ACTION_DELETE: g_table.RemoveEntry(index); break;
    }
}
```

shape.cpp

```
#include "shape.h"

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

void Shape::GetCoords(LONG& out_x1, LONG& out_y1, LONG& out_x2, LONG&
out_y2) const {
   out_x1 = this->x1; out_y1 = this->y1; out_x2 = this->x2; out_y2 = this->y2;
}
```

shape.h

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

class Shape {
protected:
   LONG x1{}, y1{}, x2{}, y2{};
public:
   virtual ~Shape() = default;
   virtual void Set(LONG ax1, LONG ay1, LONG ax2, LONG ay2);
   void GetCoords(LONG& out_x1, LONG& out_y1, LONG& out_x2, LONG& out_y2)
const;

   virtual void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) = 0;
   virtual const wchar_t* GetName() const = 0;
   virtual Shape* Clone() const = 0;
};
```

point_shape.cpp

```
#include "point_shape.h"

void PointShape::Show(HDC hdc, HPEN hPen, HBRUSH hBrush) {
   SetPixel(hdc, x1, y1, RGB(0, 0, 0));
}
Shape* PointShape::Clone() const {
   return new PointShape(*this);
}
```

point_shape.h

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

class PointShape : public virtual Shape {
public:
```

```
    void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) override;
    const wchar_t* GetName() const override { return L"Крапка"; }
    Shape* Clone() const override;
};
```

line_shape.cpp

```
#include "line_shape.h"

void LineShape::Show(HDC hdc, HPEN hPen, HBRUSH hBrush) {
    HPEN hOldPen = nullptr;
    bool ownPenCreated = false;
    if (hPen == nullptr) {
        hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
        ownPenCreated = true;
    }
    hOldPen = (HPEN)SelectObject(hdc, hPen);
    MoveToEx(hdc, x1, y1, nullptr);
    LineTo(hdc, x2, y2);
    SelectObject(hdc, hOldPen);
    if (ownPenCreated) DeleteObject(hPen);
}
Shape* LineShape::Clone() const {
    return new LineShape(*this);
}
```

line_shape.h

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

class LineShape : public virtual Shape {
public:
    void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) override;
    const wchar_t* GetName() const override { return L"Лінія"; }
    Shape* Clone() const override;
};
```

rect_shape.cpp

```
#include "rect_shape.h"

void RectShape::Show(HDC hdc, HPEN hPen, HBRUSH hBrush) {
    HPEN hOldPen = nullptr; HBRUSH hOldBrush = nullptr;
    bool ownPenCreated = false; bool ownBrushCreated = false;
    if (hPen == nullptr) {
        hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
        ownPenCreated = true;
    }
    if (hBrush == nullptr) {
        hBrush = CreateSolidBrush(RGB(255, 192, 203));
        ownBrushCreated = true;
    }
```

```
    hOldPen = (HPEN)SelectObject(hdc, hPen);
    hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
    Rectangle(hdc, x1, y1, x2, y2);
    SelectObject(hdc, hOldPen); SelectObject(hdc, hOldBrush);
    if (ownPenCreated) DeleteObject(hPen);
    if (ownBrushCreated) DeleteObject(hBrush);
}
Shape* RectShape::Clone() const {
    return new RectShape(*this);
}
```

rect_shape.h

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

class RectShape : public virtual Shape {
public:
    void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) override;
    const wchar_t* GetName() const override { return L"Прямокутник"; }
    Shape* Clone() const override;
};
```

ellipse_shape.cpp

```
#include "ellipse_shape.h"

void EllipseShape::Show(HDC hdc, HPEN hPen, HBRUSH hBrush) {
    HPEN hOldPen = nullptr; HBRUSH hOldBrush = nullptr;
    bool ownPenCreated = false;
    if (hPen == nullptr) {
        hPen = CreatePen(PS_SOLID, 1, RGB(0, 0, 0));
        ownPenCreated = true;
    }
    if (hBrush == nullptr) {
        hBrush = (HBRUSH)GetStockObject(WHITE_BRUSH);
    }
    hOldPen = (HPEN)SelectObject(hdc, hPen);
    hOldBrush = (HBRUSH)SelectObject(hdc, hBrush);
    Ellipse(hdc, x1, y1, x2, y2);
    SelectObject(hdc, hOldPen); SelectObject(hdc, hOldBrush);
    if (ownPenCreated) DeleteObject(hPen);
}
Shape* EllipseShape::Clone() const {
    return new EllipseShape(*this);
}
```

ellipse_shape.h

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

class EllipseShape : public virtual Shape {
public:
    void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) override;
    const wchar_t* GetName() const override { return L"Еліпс"; }
    Shape* Clone() const override;
};
```

lineoo_shape.cpp

```
#include "lineoo_shape.h"

void LineOOShape::Show(HDC hdc, HPEN hPen, HBRUSH hBrush) {
    LineShape::Show(hdc, hPen);
    LONG original_x1 = this->x1, original_y1 = this->y1;
    LONG original_x2 = this->x2, original_y2 = this->y2;
    const int radius = 5;

    this->x1 = original_x1 - radius;
    this->y1 = original_y1 - radius;
    this->x2 = original_x1 + radius;
    this->y2 = original_y1 + radius;

    EllipseShape::Show(hdc, hPen, hBrush);

    this->x1 = original_x2 - radius;
    this->y1 = original_y2 - radius;
    this->x2 = original_x2 + radius;
    this->y2 = original_y2 + radius;

    EllipseShape::Show(hdc, hPen, hBrush);
    this->x1 = original_x1;
    this->y1 = original_y1;
    this->x2 = original_x2;
    this->y2 = original_y2;
}
Shape* LineOOShape::Clone() const {
    return new LineOOShape(*this);
}
```

lineoo_shape.h

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

class LineOOShape : public LineShape, public EllipseShape {
public:
   void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) override;
   const wchar_t* GetName() const override { return L"Лінія з кружечками"; }
   Shape* Clone() const override;
};
```

cube_shape.cpp

```
#include "cube_shape.h"
#include <cmath>

void CubeShape::Show(HDC hdc, HPEN hPen, HBRUSH hBrush) {
   LONG sideX = abs(x2 - x1); LONG sideY = abs(y2 - y1);
   float offsetX = static_cast<float>(sideX) / 2.0f;
   float offsetY = static_cast<float>(sideY) / 2.0f;
   LONG fx1 = x1; LONG fy1 = y1; LONG fx2 = x1 + sideX; LONG fy2 = y1 + sideY;
   LONG bx1 = fx1 + static_cast<LONG>(round(offsetX)); LONG by1 = fy1 -
static_cast<LONG>(round(offsetY));
   LONG bx2 = fx2 + static_cast<LONG>(round(offsetX)); LONG by2 = fy2 -
static_cast<LONG>(round(offsetY));

   LONG original_x1 = this->x1, original_y1 = this->y1;
   LONG original_x2 = this->x2, original_y2 = this->y2;

   HBRUSH hActualBrush = hBrush;
   HBRUSH hOldBrush = nullptr;

   if (hActualBrush == nullptr) {
      hActualBrush = (HBRUSH)GetStockObject(NULL_BRUSH);
   }
   hOldBrush = (HBRUSH)SelectObject(hdc, hActualBrush);

   this->x1 = bx1; this->y1 = by1; this->x2 = bx2; this->y2 = by2;
   RectShape::Show(hdc, hPen, hActualBrush);
   this->x1 = fx1; this->y1 = fy1; this->x2 = fx2; this->y2 = fy2;
   RectShape::Show(hdc, hPen, hActualBrush);

   SelectObject(hdc, hOldBrush);

   this->x1 = fx1; this->y1 = fy1; this->x2 = bx1; this->y2 = by1; LineShape::Show(hdc,
hPen);
   this->x1 = fx2; this->y1 = fy1; this->x2 = bx2; this->y2 = by1; LineShape::Show(hdc,
```

```
hPen);
   this->x1 = fx2; this->y1 = fy2; this->x2 = bx2; this->y2 = by2; LineShape::Show(hdc,
hPen);
   this->x1 = fx1; this->y1 = fy2; this->x2 = bx1; this->y2 = by2; LineShape::Show(hdc,
hPen);

   this->x1 = original_x1; this->y1 = original_y1;
   this->x2 = original_x2; this->y2 = original_y2;
}
Shape* CubeShape::Clone() const {
   return new CubeShape(*this);
}
```

cube_shape.h

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

class CubeShape : public LineShape, public RectShape {
public:
   void Show(HDC hdc, HPEN hPen = nullptr, HBRUSH hBrush = nullptr) override;
   const wchar_t* GetName() const override { return L"Куб"; }
   Shape* Clone() const override;
};
```

my_editor.cpp

```
#include "my_editor.h"
#include "resource.h"
#include <windowsx.h>
#include <string>
#include <commctrl.h>
#include <fstream>
#include <codecvt>
#include <locale>
#include <cstdio>
#include <cmath>
#include <sstream>
#include "point_shape.h"
#include "line_shape.h"
#include "rect_shape.h"
#include "ellipse_shape.h"
#include "lineoo_shape.h"
#include "cube_shape.h"

MyEditor* MyEditor::p_instance = nullptr;

MyEditor* MyEditor::getInstance(HWND hWnd) {
```

```
    if (!p_instance) p_instance = new MyEditor(hWnd);
    return p_instance;
}

MyEditor::MyEditor(HWND hWnd) : m_hWnd(hWnd) {
    m_max_objects = 120;
    m_objects = new Shape * [m_max_objects];
    for (int i = 0; i < m_max_objects; ++i) m_objects[i] = nullptr;
    loadShapesFromFile();
}

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

void MyEditor::RegisterCallback(EditorCallback callback) { m_callback = callback; }
void MyEditor::SetToolbar(HWND hwnd) { m_hwndToolBar = hwnd; }

void MyEditor::CreateToolbar(HINSTANCE hInst) {
    INITCOMMONCONTROLSEX icex;
    icex.dwSize = sizeof(INITCOMMONCONTROLSEX);
    icex.dwICC = ICC_BAR_CLASSES;
    InitCommonControlsEx(&icex);

    m_hwndToolBar = CreateWindowEx(0, TOOLBARCLASSNAME, NULL,
        WS_CHILD | WS_VISIBLE | WS_BORDER | TBSTYLE_TOOLTIPS,
        0, 0, 0, 0, m_hWnd, (HMENU)1, hInst, NULL);
    if (!m_hwndToolBar) return;
    SendMessage(m_hwndToolBar, TB_BUTTONSTRUCTSIZE,
(WPARAM)sizeof(TBBUTTON), 0);
    TBADDBITMAP tbab;
    tbab.hInst = hInst;
    tbab.nID = IDB_BITMAP1;
    SendMessage(m_hwndToolBar, TB_ADDBITMAP, 6, (LPARAM)&tbab);
    TBBUTTON tbb[6];
    ZeroMemory(tbb, sizeof(tbb));
    tbb[0] = { 0, IDM_TOOL_POINT,   TBSTATE_ENABLED, TBSTYLE_BUTTON |
BTNS_CHECKGROUP, {0}, 0, (INT_PTR)L"Крапка" };
    tbb[1] = { 1, IDM_TOOL_LINE,    TBSTATE_ENABLED, TBSTYLE_BUTTON |
BTNS_CHECKGROUP, {0}, 0, (INT_PTR)L"Лінія" };
    tbb[2] = { 2, IDM_TOOL_RECT,    TBSTATE_ENABLED, TBSTYLE_BUTTON |
BTNS_CHECKGROUP, {0}, 0, (INT_PTR)L"Прямокутник" };
    tbb[3] = { 3, IDM_TOOL_ELLIPSE, TBSTATE_ENABLED, TBSTYLE_BUTTON |
BTNS_CHECKGROUP, {0}, 0, (INT_PTR)L"Еліпс" };
    tbb[4] = { 4, IDM_TOOL_LINEOO,  TBSTATE_ENABLED, TBSTYLE_BUTTON |
BTNS_CHECKGROUP, {0}, 0, (INT_PTR)L"ЛініяОО" };
    tbb[5] = { 5, IDM_TOOL_CUBE,    TBSTATE_ENABLED, TBSTYLE_BUTTON |
```

```
BTNS_CHECKGROUP, {0}, 0, (INT_PTR)L"Куб" };
   SendMessage(m_hwndToolBar, TB_ADDBUTTONS, 6, (LPARAM)&tbb);
}

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

void MyEditor::Start(Shape* prototype, int shapeId) {
   if (m_isDrawing) { m_isDrawing = false; InvalidateRect(m_hWnd, nullptr, TRUE); }
   if (m_prototype) delete m_prototype;
   m_prototype = prototype;

   m_currentShapeId = shapeId;

   m_selectedIndex = -1;
   InvalidateRect(m_hWnd, nullptr, TRUE);
   std::wstring shapeName = L"Режим: ";
   if (!m_prototype) shapeName += L"Не вибрано";
   else shapeName += m_prototype->GetName();
   SetWindowText(m_hWnd, shapeName.c_str());
   if (m_hwndToolBar && m_prototype) {
      SendMessage(m_hwndToolBar, TB_CHECKBUTTON, IDM_TOOL_POINT,
(m_currentShapeId == IDM_OBJ_POINT));
      SendMessage(m_hwndToolBar, TB_CHECKBUTTON, IDM_TOOL_LINE,
(m_currentShapeId == IDM_OBJ_LINE));
      SendMessage(m_hwndToolBar, TB_CHECKBUTTON, IDM_TOOL_RECT,
(m_currentShapeId == IDM_OBJ_RECT));
      SendMessage(m_hwndToolBar, TB_CHECKBUTTON, IDM_TOOL_ELLIPSE,
(m_currentShapeId == IDM_OBJ_ELLIPSE));
      SendMessage(m_hwndToolBar, TB_CHECKBUTTON, IDM_TOOL_LINEOO,
(m_currentShapeId == IDM_OBJ_LINEOO));
      SendMessage(m_hwndToolBar, TB_CHECKBUTTON, IDM_TOOL_CUBE,
(m_currentShapeId == IDM_OBJ_CUBE));
   }
}

void MyEditor::OnLDown(HWND hWnd, int x, int y) {
   if (m_prototype) {
      m_isDrawing = true; x0 = x_temp = x; y0 = y_temp = y;
      m_selectedIndex = -1; InvalidateRect(hWnd, nullptr, TRUE);
   }
}

void MyEditor::OnLUp(HWND hWnd, int x, int y) {
   if (m_isDrawing) {
      m_isDrawing = false;
      if (m_count < m_max_objects) {
         LONG fx1 = x0, fy1 = y0, fx2 = x, fy2 = y;
```

```
            bool isJustRect = (m_currentShapeId == IDM_OBJ_RECT);

            if (isJustRect) {
                LONG dx = abs(x - x0), dy = abs(y - y0);
                fx1 = x0 - dx; fy1 = y0 - dy; fx2 = x0 + dx; fy2 = y0 + dy;
            }

            if (!m_prototype) return;
            Shape* newShape = m_prototype->Clone();
            newShape->Set(fx1, fy1, fx2, fy2);
            if (newShape) {
                m_objects[m_count++] = newShape; saveShapeToFile(newShape);
                if (m_callback) {
                    LONG sx1, sy1, sx2, sy2; newShape->GetCoords(sx1, sy1, sx2, sy2);
                    m_callback(EDITOR_ACTION_ADD, newShape->GetName(), sx1, sy1, sx2,
sy2, m_count - 1);
                }
            }
        }
        InvalidateRect(hWnd, nullptr, TRUE);
    }
}

void MyEditor::OnMouseMove(HWND hWnd, int x, int y) {
    if (m_isDrawing) { x_temp = x; y_temp = y; InvalidateRect(hWnd, nullptr, TRUE); }
}

void MyEditor::OnPaint(HWND hWnd) {
    PAINTSTRUCT ps; HDC hdc = BeginPaint(hWnd, &ps);
    for (int i = 0; i < m_count; ++i) {
        if (m_objects[i]) {
            if (i == m_selectedIndex) {
                HPEN hSelectPen = CreatePen(PS_SOLID, 3, RGB(255, 0, 0));
                m_objects[i]->Show(hdc, hSelectPen, nullptr);
                DeleteObject(hSelectPen);
            }
            else {
                m_objects[i]->Show(hdc);
            }
        }
    }
    if (m_isDrawing && m_prototype) {
        HPEN hDotPen = CreatePen(PS_DOT, 1, RGB(0, 0, 0));
        HBRUSH hNullBrush = (HBRUSH)GetStockObject(NULL_BRUSH);
        HPEN hOldPen = (HPEN)SelectObject(hdc, hDotPen);
        HBRUSH hOldBrush = (HBRUSH)SelectObject(hdc, hNullBrush);
        LONG xs = x0, ys = y0, xe = x_temp, ye = y_temp;
        bool isJustRect = dynamic_cast<RectShape*>(m_prototype) &&
!dynamic_cast<CubeShape*>(m_prototype);
```

```
        if (isJustRect) { LONG dx = abs(xe - xs), dy = abs(ye - ys); xs = x0 - dx; ys = y0 - dy;
xe = x0 + dx; ye = y0 + dy; }
        m_prototype->Set(xs, ys, xe, ye);
        m_prototype->Show(hdc, hDotPen, hNullBrush);
        SelectObject(hdc, hOldPen); SelectObject(hdc, hOldBrush);
        DeleteObject(hDotPen);
    }
    EndPaint(hWnd, &ps);
}

void MyEditor::OnNotify(HWND hWnd, WPARAM wParam, LPARAM lParam) {
    if (lParam == NULL || m_hwndToolBar == NULL) return;
    LPNMHDR pnmh = (LPNMHDR)lParam;
    if (pnmh->hwndFrom == m_hwndToolBar && 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;
        case IDM_TOOL_LINEOO:  lstrcpy(lpttt->szText, L"Лінія з кружечками"); break;
        case IDM_TOOL_CUBE:    lstrcpy(lpttt->szText, L"Куб"); break;
        }
    }
}
Shape* MyEditor::createShapeByName(const std::wstring& name) {
    if (name == L"Крапка") return new PointShape();
    if (name == L"Лінія з кружечками") return new LineOOShape();
    if (name == L"Куб") return new CubeShape();
    if (name == L"Лінія") return new LineShape();
    if (name == L"Прямокутник") return new RectShape();
    if (name == L"Еліпс") return new EllipseShape();
    return nullptr;
}

void MyEditor::loadShapesFromFile() {
    std::wifstream wif("shapes.txt");
    if (!wif.is_open()) {
        return;
    }
    wif.imbue(std::locale(std::locale(),
        new std::codecvt_utf8<wchar_t, 0x10FFFF, std::consume_header>));

    std::wstring line;
    while (std::getline(wif, line)) {
        if (m_count >= m_max_objects) {
            break;
        }
```

```
        std::wstringstream wss(line);
        std::wstring shapeName;
        LONG x1, y1, x2, y2;
        if (std::getline(wss, shapeName, L'\t') &&
            (wss >> x1 >> y1 >> x2 >> y2))
        {
            Shape* newShape = createShapeByName(shapeName);

            if (newShape) {
                newShape->Set(x1, y1, x2, y2);
                m_objects[m_count++] = newShape;
            }
        }
    }
    wif.close();
}

void MyEditor::SelectShape(int index) {
    m_selectedIndex = (index >= 0 && index < m_count) ? index : -1;
    InvalidateRect(m_hWnd, nullptr, TRUE);
}

void MyEditor::updateFileAfterDeletion() {
    FILE* file; errno_t err = fopen_s(&file, "shapes.txt", "wb");
    if (err == 0 && file != nullptr) {
        unsigned char bom[] = { 0xEF, 0xBB, 0xBF }; fwrite(bom, sizeof(bom), 1, file);
        for (int i = 0; i < m_count; ++i) {
            if (m_objects[i]) {
                const wchar_t* snW = m_objects[i]->GetName(); std::string snA;
                int len = WideCharToMultiByte(CP_UTF8, 0, snW, -1, NULL, 0, NULL,
NULL);
                if (len > 0) { snA.resize(len - 1); WideCharToMultiByte(CP_UTF8, 0, snW, -1,
&snA[0], len, NULL, NULL); }
                LONG x1, y1, x2, y2; m_objects[i]->GetCoords(x1, y1, x2, y2);
                fprintf(file, "%s\t%ld\t%ld\t%ld\t%ld\r\n", snA.c_str(), x1, y1, x2, y2);
            }
        }
        fclose(file);
    }
}

void MyEditor::DeleteShape(int index) {
    if (index >= 0 && index < m_count) {
        delete m_objects[index];
        for (int i = index; i < m_count - 1; ++i) m_objects[i] = m_objects[i + 1];
        m_objects[m_count - 1] = nullptr; m_count--; m_selectedIndex = -1;
        if (m_callback) m_callback(EDITOR_ACTION_DELETE, nullptr, 0, 0, 0, 0, index);
        updateFileAfterDeletion();
        InvalidateRect(m_hWnd, nullptr, TRUE);
```

```
    }
}


void MyEditor::saveShapeToFile(Shape* shape) {
    if (!shape) return; FILE* file; errno_t err = fopen_s(&file, "shapes.txt", "ab");
    if (err != 0 || file == nullptr) return;
    fseek(file, 0, SEEK_END); if (ftell(file) == 0) { unsigned char bom[] = { 0xEF, 0xBB,
0xBF }; fwrite(bom, sizeof(bom), 1, file); }
    const wchar_t* snW = shape->GetName(); std::string snA;
    int len = WideCharToMultiByte(CP_UTF8, 0, snW, -1, NULL, 0, NULL, NULL);
    if (len > 0) { snA.resize(len - 1); WideCharToMultiByte(CP_UTF8, 0, snW, -1, &snA[0],
len, NULL, NULL); }
    LONG x1, y1, x2, y2; shape->GetCoords(x1, y1, x2, y2);
    fprintf(file, "%s\t%ld\t%ld\t%ld\t%ld\r\n", snA.c_str(), x1, y1, x2, y2);
    fclose(file);
}

int MyEditor::GetCount() const { return m_count; }
const Shape* MyEditor::GetShape(int index) const { return (index >= 0 && index <
m_count) ? m_objects[index] : nullptr; }

void MyEditor::OnInitMenuPopup(HMENU hMenu) {
    CheckMenuItem(hMenu, IDM_OBJ_POINT, MF_BYCOMMAND | (m_currentShapeId
== IDM_OBJ_POINT ? MF_CHECKED : MF_UNCHECKED));
    CheckMenuItem(hMenu, IDM_OBJ_LINE, MF_BYCOMMAND | (m_currentShapeId
== IDM_OBJ_LINE ? MF_CHECKED : MF_UNCHECKED));
    CheckMenuItem(hMenu, IDM_OBJ_RECT, MF_BYCOMMAND | (m_currentShapeId
== IDM_OBJ_RECT ? MF_CHECKED : MF_UNCHECKED));
    CheckMenuItem(hMenu, IDM_OBJ_ELLIPSE, MF_BYCOMMAND |
(m_currentShapeId == IDM_OBJ_ELLIPSE ? MF_CHECKED : MF_UNCHECKED));
    CheckMenuItem(hMenu, IDM_OBJ_LINEOO, MF_BYCOMMAND |
(m_currentShapeId == IDM_OBJ_LINEOO ? MF_CHECKED : MF_UNCHECKED));
    CheckMenuItem(hMenu, IDM_OBJ_CUBE, MF_BYCOMMAND | (m_currentShapeId
== IDM_OBJ_CUBE ? MF_CHECKED : MF_UNCHECKED));
}
```

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"

enum EditorAction { EDITOR_ACTION_ADD, EDITOR_ACTION_DELETE };
```

```
typedef void (*EditorCallback)(EditorAction action, const wchar_t* shapeName, LONG x1,
LONG y1, LONG x2, LONG y2, int index);

class MyEditor {
public:
    static MyEditor* getInstance(HWND hWnd);

    void RegisterCallback(EditorCallback callback);
    void SelectShape(int index);
    void DeleteShape(int index);
    int GetCount() const;
    const Shape* GetShape(int index) const;

    void SetToolbar(HWND hwnd);
    void CreateToolbar(HINSTANCE hInst);
    void OnSize();
    void Start(Shape* prototype, int shapeId);
    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 OnNotify(HWND hWnd, WPARAM wParam, LPARAM lParam);
    void OnInitMenuPopup(HMENU hMenu);

private:
    MyEditor(HWND hWnd);
    ~MyEditor();
    MyEditor(const MyEditor&) = delete;
    MyEditor& operator=(const MyEditor&) = delete;
    static MyEditor* p_instance;
    int m_currentShapeId = 0;
    HWND m_hWnd;
    EditorCallback m_callback = nullptr;
    int m_selectedIndex = -1;

    HWND m_hwndToolBar = NULL;
    Shape** m_objects = nullptr;
    int m_count = 0;
    int m_max_objects = 0;

    Shape* m_prototype = nullptr;
    bool m_isDrawing = false;
    LONG x0{}, y0{};
    LONG x_temp{}, y_temp{};

    void saveShapeToFile(Shape* shape);
    void updateFileAfterDeletion();
    void loadShapesFromFile();
    Shape* createShapeByName(const std::wstring& name);
```

```
};
```

my_table.cpp

```
#include "my_table.h"
#include "resource.h"
#include <wchar.h>

static MyTable* p_instance = nullptr;

MyTable::MyTable() {
   p_instance = this;
}

void MyTable::Show(HINSTANCE hInst, HWND hWndParent) {
   if (!m_hDlg) {
      m_hDlg = CreateDialog(hInst, MAKEINTRESOURCE(IDD_TABLE_VIEW),
hWndParent, DlgProc);
      ShowWindow(m_hDlg, SW_SHOW);
   }
   else {
      SetFocus(m_hDlg);
   }
}

void MyTable::AddEntry(const wchar_t* shapeName, LONG x1, LONG y1, LONG x2,
LONG y2) {
   if (m_hDlg) {
      wchar_t buffer[256];
      swprintf_s(buffer, 256, L"%s\t%ld\t%ld\t%ld\t%ld", shapeName, x1, y1, x2, y2);
      SendDlgItemMessage(m_hDlg, IDC_SHAPE_LIST, LB_INSERTSTRING,
(WPARAM)-1, (LPARAM)buffer);
   }
}

void MyTable::RemoveEntry(int index) {
   if (m_hDlg) {
      SendDlgItemMessage(m_hDlg, IDC_SHAPE_LIST, LB_DELETESTRING,
(WPARAM)index, 0);
   }
}


void MyTable::Clear() {
   if (m_hDlg) {
      SendDlgItemMessage(m_hDlg, IDC_SHAPE_LIST, LB_RESETCONTENT, 0, 0);
   }
}
```

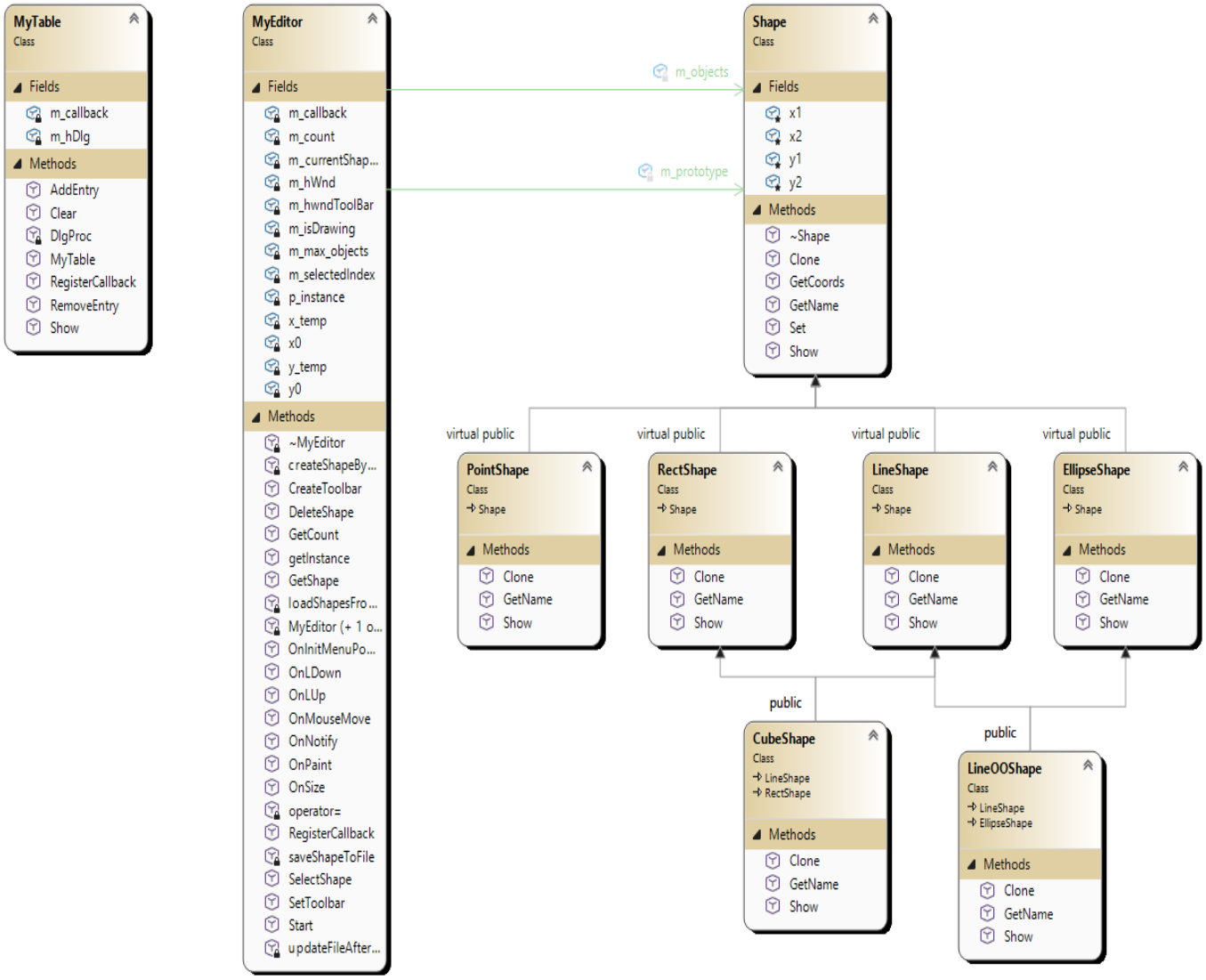
```
void MyTable::RegisterCallback(TableCallback callback) {
   m_callback = callback;
}

INT_PTR CALLBACK MyTable::DlgProc(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam) {
   switch (message) {
   case WM_INITDIALOG:
   {
      int tabs[] = { 100, 140, 180, 220 };
      SendDlgItemMessage(hDlg, IDC_SHAPE_LIST, LB_SETTABSTOPS, 4,
(LPARAM)tabs);

      return (INT_PTR)TRUE;
   }

   case WM_COMMAND:
      switch (LOWORD(wParam)) {
      case IDC_SHAPE_LIST:
         if (HIWORD(wParam) == LBN_SELCHANGE && p_instance->m_callback) {
            int selectedIndex = SendDlgItemMessage(hDlg, IDC_SHAPE_LIST,
LB_GETCURSEL, 0, 0);
            p_instance->m_callback(TABLE_ACTION_SELECT, selectedIndex);
         }
         break;
      case IDC_DELETE_BUTTON:
         if (p_instance->m_callback) {
            int selectedIndex = SendDlgItemMessage(hDlg, IDC_SHAPE_LIST,
LB_GETCURSEL, 0, 0);
            if (selectedIndex != LB_ERR) {
               p_instance->m_callback(TABLE_ACTION_DELETE, selectedIndex);
            }
         }
         break;
      case IDOK:
      case IDCANCEL:
         DestroyWindow(hDlg);
         break;
      }
      return (INT_PTR)TRUE;

   case WM_CLOSE:
      DestroyWindow(hDlg);
      return (INT_PTR)TRUE;

   case WM_DESTROY:
      p_instance->m_hDlg = NULL;
      if (p_instance->m_callback) {
         p_instance->m_callback(TABLE_ACTION_CLOSE, -1);
```

```
        }
        return (INT_PTR)TRUE;
    }
    return (INT_PTR)FALSE;
}
```

my_table.h

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

enum TableAction {
    TABLE_ACTION_SELECT,
    TABLE_ACTION_DELETE,
    TABLE_ACTION_CLOSE
};

typedef void (*TableCallback)(TableAction action, int index);

class MyTable {
public:
    MyTable();
    void Show(HINSTANCE hInst, HWND hWndParent);
    void AddEntry(const wchar_t* shapeName, LONG x1, LONG y1, LONG x2, LONG y2);
    void RemoveEntry(int index);
    void Clear();
    void RegisterCallback(TableCallback callback);

private:
    HWND m_hDlg = NULL;
    TableCallback m_callback = nullptr;
    static INT_PTR CALLBACK DlgProc(HWND hDlg, UINT message, WPARAM
wParam, LPARAM lParam);
};
```

Діаграма класів



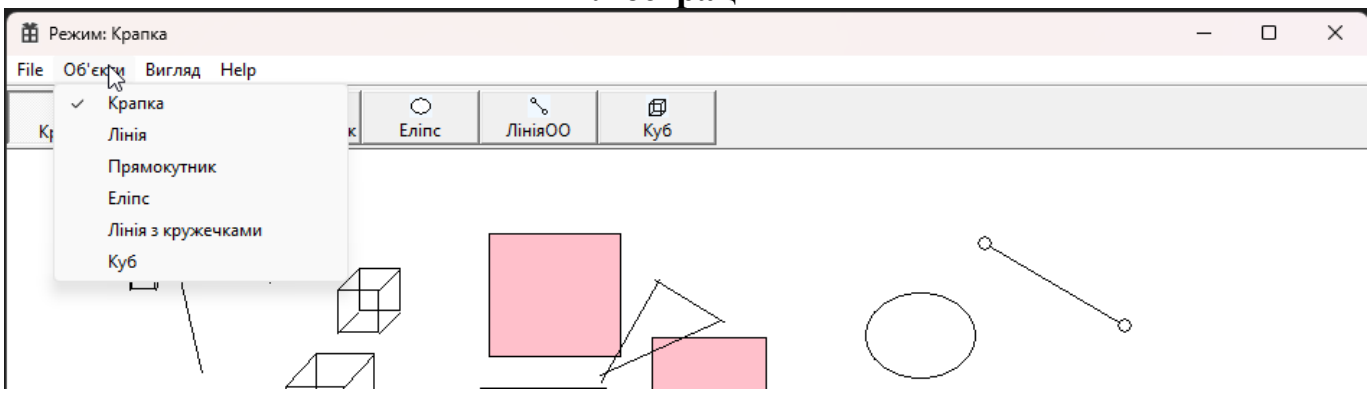
Приклад текстового файлу з множинами об’єктів

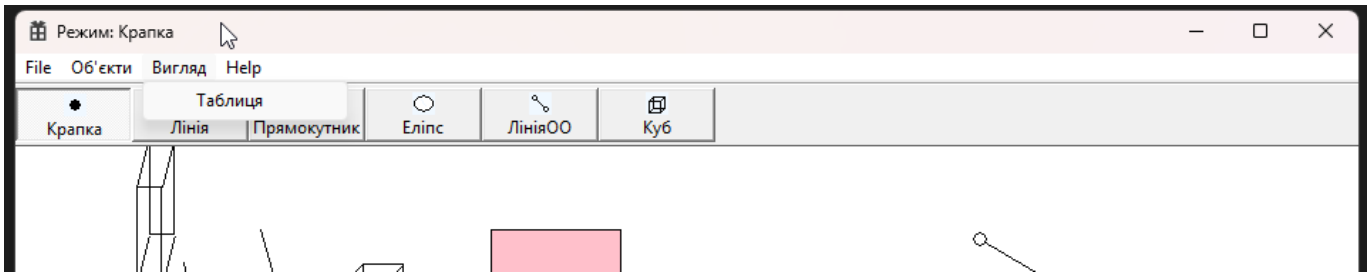
shapes

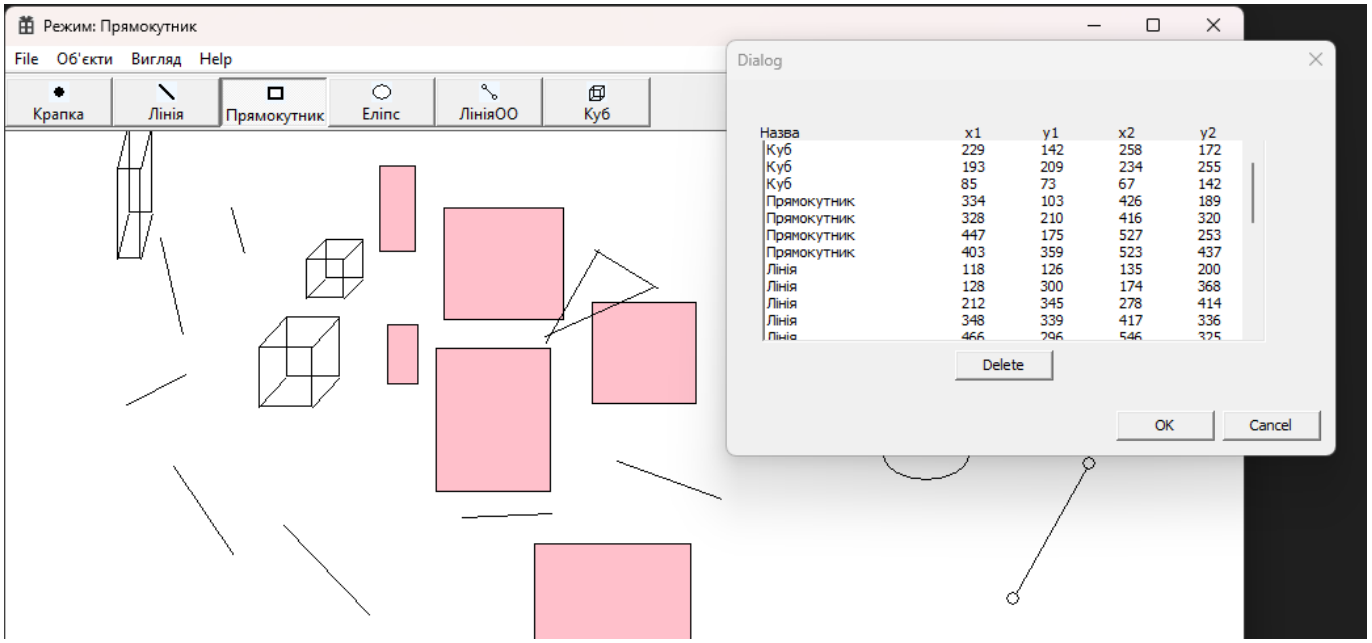
File Edit View

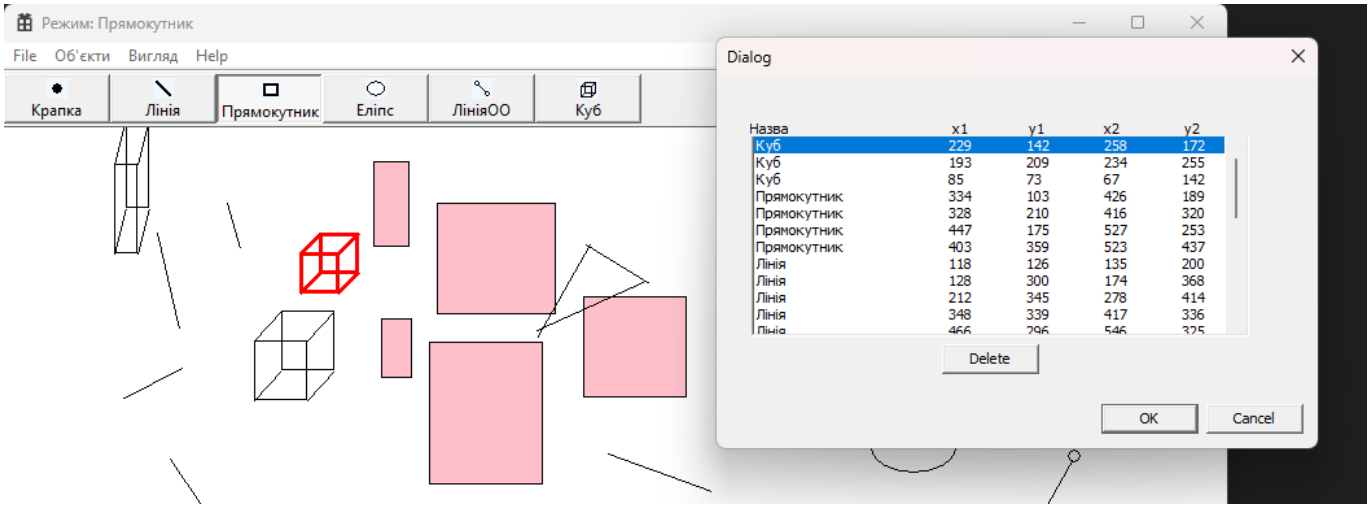
```
Куб     229     142     258     172
Куб     193     209     234     255
Куб     85      73      67      142
Прямокутник     334     103     426     189
Прямокутник     328     210     416     320
Прямокутник     447     175     527     253
Прямокутник     403     359     523     437
Лінія   118     126     135     200
Лінія   128     300     174     368
Лінія   212     345     278     414
Лінія   348     339     417     336
Лінія   466     296     546     325
Лінія   452     135     411     207
Лінія   449     135     498     165
Лінія   411     201     495     163
Лінія   566     135     566     135
Елінс   595     144     672     204
Елінс   669     271     735     311
Лінія з кружечками      678     110     775     167
Лінія з кружечками      826     298     768     401
Лінія   172     103     182     138
Лінія   137     230     91      254
Крапка  88      465     92      464
Крапка  189     491     190     491
Крапка  265     519     265     519
Куб     643     450     674     469
```

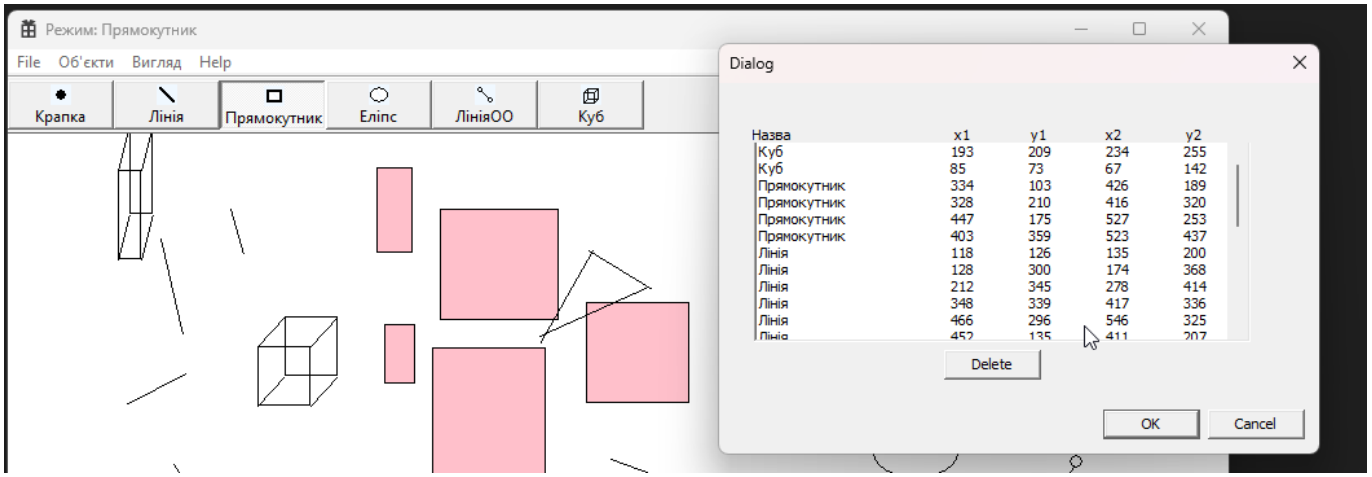
Ілюстрації

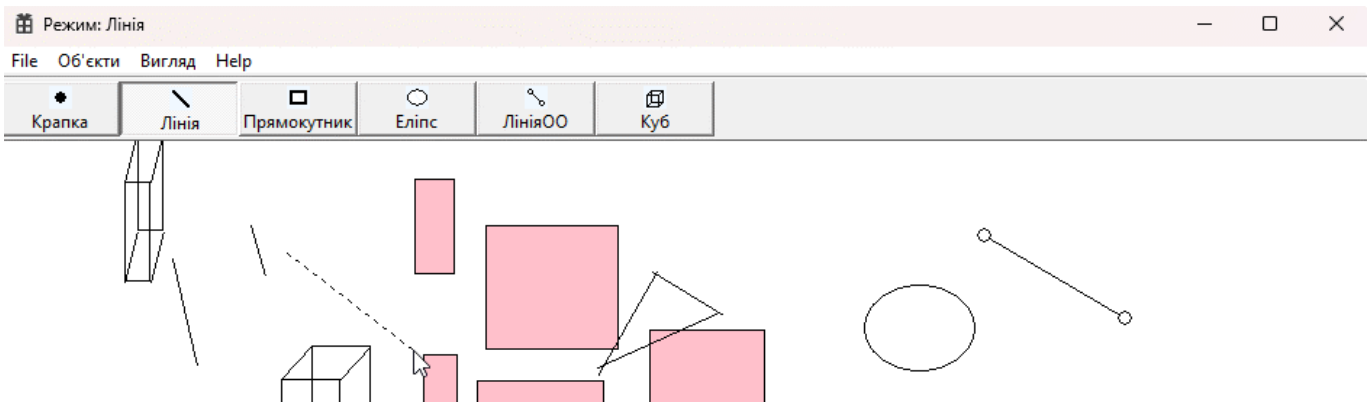

Режим: Крапка
File
Об'єкти
Вигляд
Help
Крапка
Лінія
Прямокутник
Еліпс
Лінія з кружечками
Куб
Еліпс
ЛініяОО
Куб

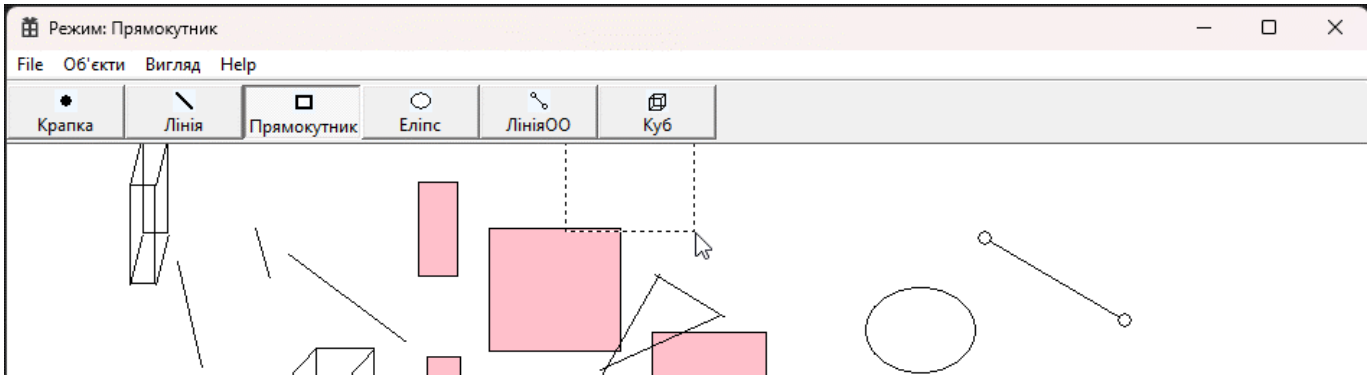

Режим: Крапка
File
Об'єкти
Вигляд
Help
Таблиця
Крапка
Лінія
Прямокутник
Еліпс
ЛініяОО
Куб

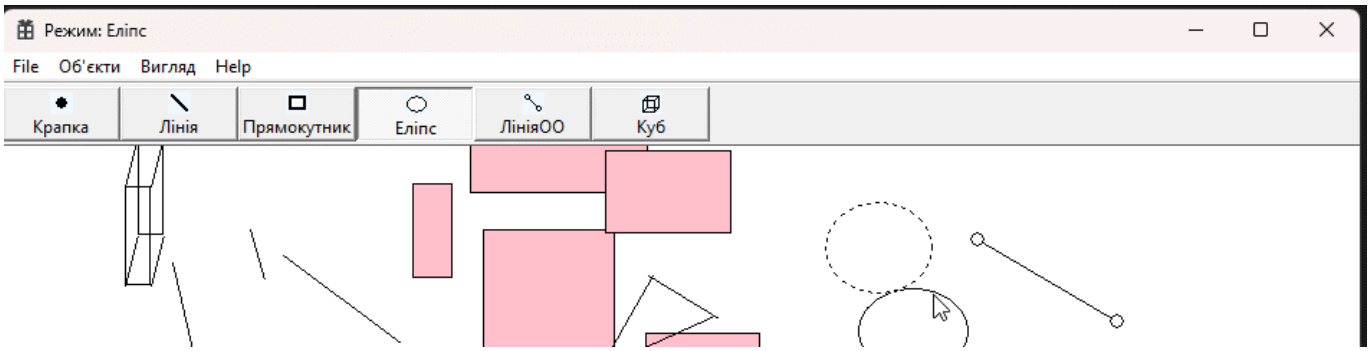

Режим: Прямокутник
File
Об'єкти
Вигляд
Help
Крапка
Лінія
Прямокутник
Еліпс
ЛініяОО
Куб
Dialog
Назва x1 y1 x2 y2
Куб 229 142 258 172
Куб 193 209 234 255
Куб 85 73 67 142
Прямокутник 334 103 426 189
Прямокутник 328 210 416 320
Прямокутник 447 175 527 253
Прямокутник 403 359 523 437
Лінія 118 126 135 200
Лінія 128 300 174 368
Лінія 212 345 278 414
Лінія 348 339 417 336
Delete
OK
Cancel

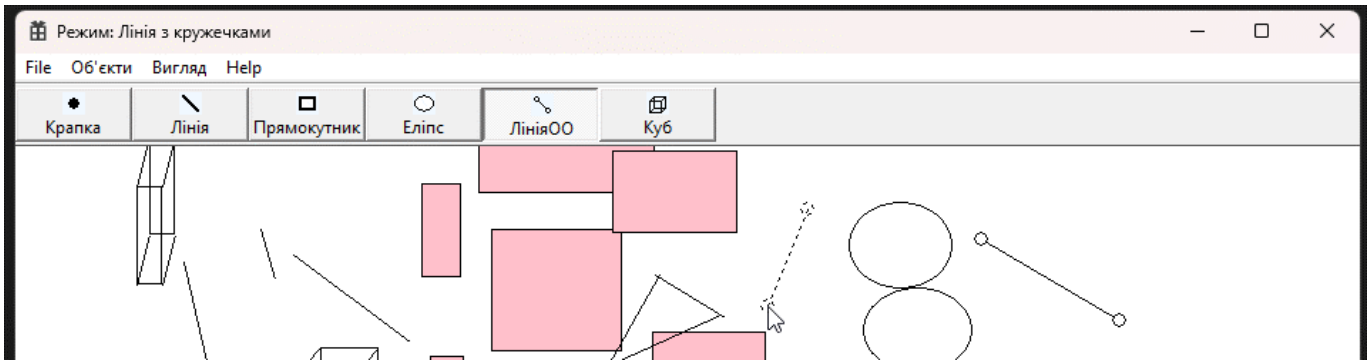

Режим: Прямокутник
File
Об'єкти
Вигляд
Help
Крапка
Лінія
Прямокутник
Еліпс
ЛініяОО
Куб
Dialog
Назва x1 y1 x2 y2
Куб 229 142 258 172
Куб 193 209 234 255
Куб 85 73 67 142
Прямокутник 334 103 426 189
Прямокутник 328 210 416 320
Прямокутник 447 175 527 253
Прямокутник 403 359 523 437
Лінія 118 126 135 200
Лінія 128 300 174 368
Лінія 212 345 278 414
Лінія 348 339 417 336
Delete
OK
Cancel

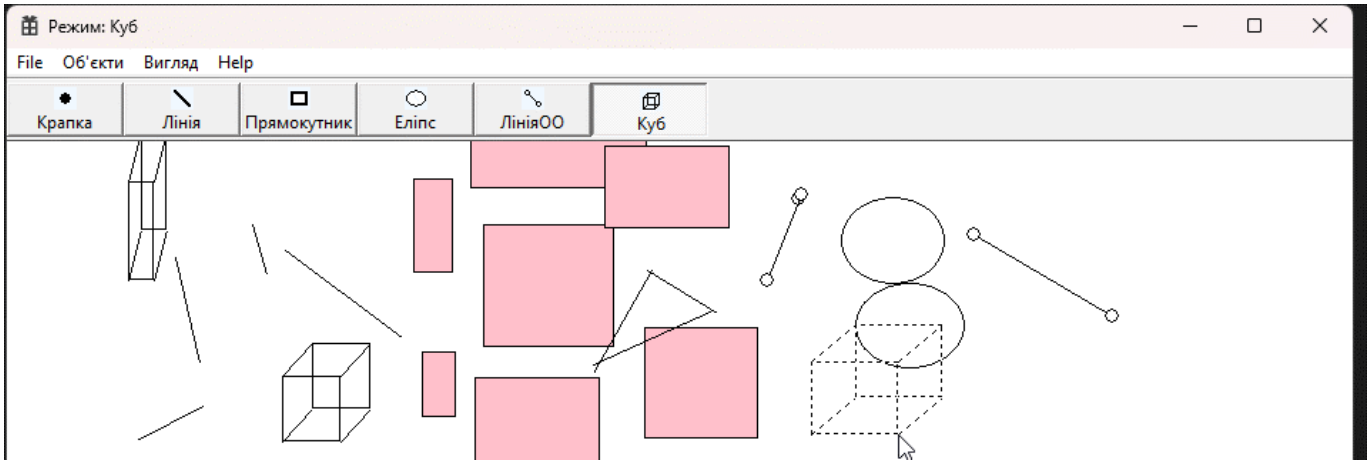

Режим: Прямокутник
File Об'єкти Вигляд Help
Крапка Лінія Прямокутник Еліпс ЛініяОО Куб
Dialog
Назва x1 y1 x2 y2
Куб 193 209 234 255
Куб 85 73 67 142
Прямокутник 334 103 426 189
Прямокутник 328 210 416 320
Прямокутник 447 175 527 253
Прямокутник 403 359 523 437
Лінія 118 126 135 200
Лінія 128 300 174 368
Лінія 212 345 278 414
Лінія 348 339 417 336
Лінія 466 296 546 325
Delete
OK
Cancel

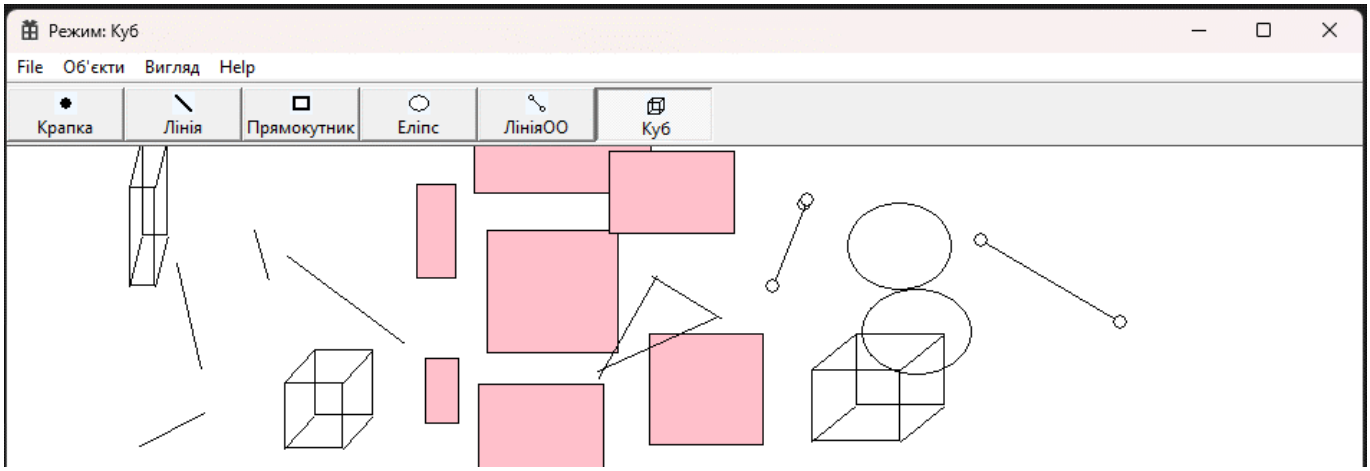

Режим: Лінія
File Об'єкти Вигляд Help
Крапка Лінія Прямокутник Еліпс ЛініяОО Куб


Режим: Прямокутник
File Об'єкти Вигляд Help
Крапка Лінія Прямокутник Еліпс ЛініяОО Куб


Режим: Еліпс
File Об'єкти Вигляд Help
Крапка Лінія Прямокутник Еліпс ЛініяОО Куб


Режим: Лінія з кружечками
File Об'єкти Вигляд Help
Крапка Лінія Прямокутник Еліпс ЛініяОО Куб


Режим: Куб
File
Об'єкти
Вигляд
Help
Крапка
Лінія
Прямокутник
Еліпс
ЛініяОО
Куб


Режим: Куб
File
Об'єкти
Вигляд
Help
Крапка
Лінія
Прямокутник
Еліпс
ЛініяОО
Куб

Висновок

В ході виконання лабораторної роботи №5 “Розробка багатовіконного інтерфейсу користувача для графічного редактора об’єктів ” було отримано вміння та навички програмувати багатовіконний інтерфейс програми на C++ в об’єктно-орієнтованому стилі. Також були отримані практичні навички використання патерну Singleton, а також технології Callback. І навчилися працювати з записом в файли і виводом з файлів в коді на мові C++.