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 №2

з дисципліни

«ООП»

на тему «Розробка графічного редактора об’єктів на C++»

Виконав:

студент групи ІП-93

Домінський Валентин Олексійович

номер залікової книжки: 9311

Перевірив:

Порєв Віктор Миколайович

Київ 2020

Мета:

Мета роботи – отримати вміння та навички використовувати інкапсуляцію, абстракцію типів, успадкування та поліморфізм на основі класів C++, запрограмувавши простий графічний редактор в об’єктно-орієнтованому стилі.

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 з ім’ям Lab2.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.

Варіанти завдань

- динамічний масив для Shape (9 mod 3 = 0) обсягом 109 об’єктів

- "гумовий" слід (9 mod 4 = 1) – суцільна лінія червоного кольору

- прямокутник:

    - ввід від центру до одного з кутів (9 mod 2 = 1)

    - чорний контур прямокутника без заповнення (9 mod 5 = 4)

- еліпс:

    - по двом протилежним кутам охоплюючого прямокутника (9 mod 2 = 1)

    - чорний контур з кольоровим заповненням (9 mod 5 = 4)

    - колір заповнення: блакитний (9 mod 6 = 3)

- позначка поточного типу об’єкту: в заголовку вікна (9 mod 2 = 1)

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:

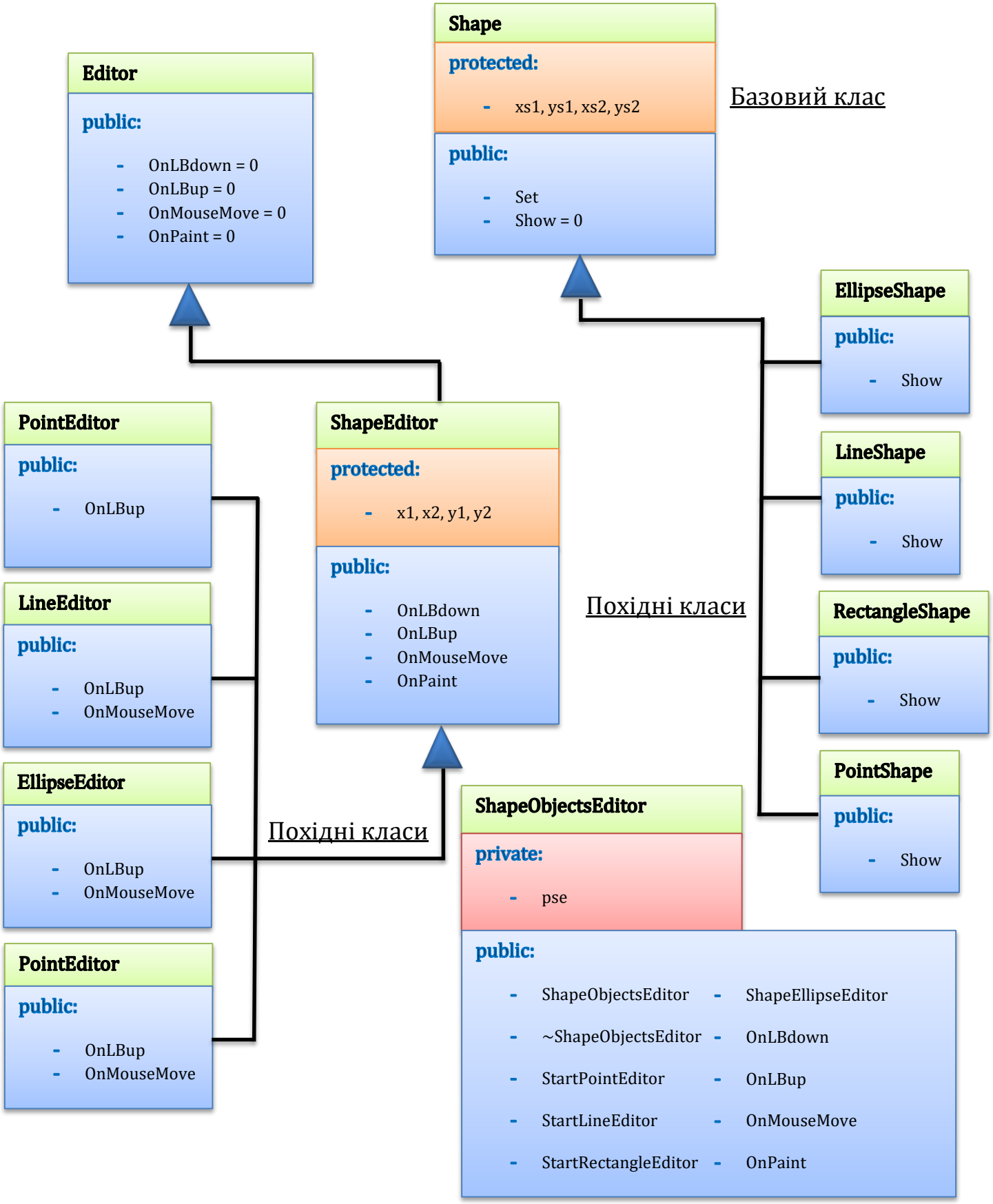
```
#pragma once
#include "pch.h"
#include "editor.h"
#include "Resource.h"

#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;
```

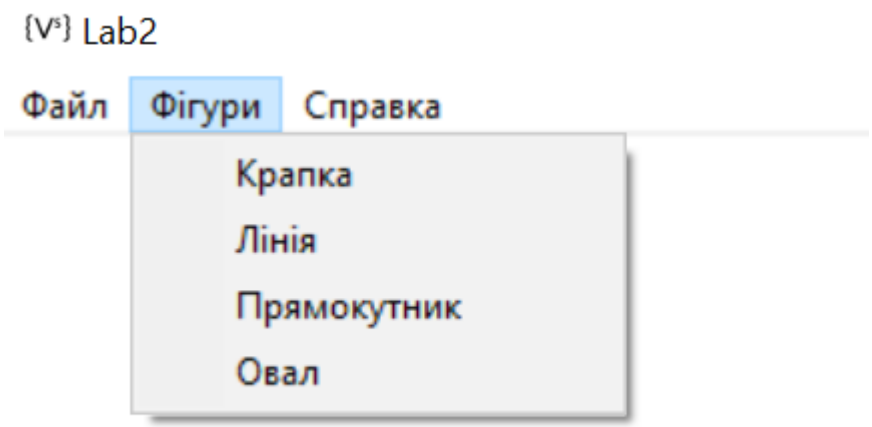
```
    virtual void OnMouseMove(HWND) = 0;
    virtual void OnPaint(HWND) = 0;
};
```


Shape
protected:
- xs1, ys1, xs2, ys2
public:
- Set
- Show = 0
Базовий клас
Editor
public:
- OnLBdown = 0
- OnLBup = 0
- OnMouseMove = 0
- OnPaint = 0
EllipseShape
public:
- Show
LineShape
public:
- Show
RectangleShape
public:
- Show
PointShape
public:
- Show
Похідні класи
ShapeEditor
protected:
- x1, x2, y1, y2
public:
- OnLBdown
- OnLBup
- OnMouseMove
- OnPaint
PointEditor
public:
- OnLBup
LineEditor
public:
- OnLBup
- OnMouseMove
EllipseEditor
public:
- OnLBup
- OnMouseMove
PointEditor
public:
- OnLBup
- OnMouseMove
Похідні класи
ShapeObjectsEditor
private:
- pse
public:
- ShapeObjectsEditor
- ~ShapeObjectsEditor
- StartPointEditor
- StartLineEditor
- StartRectangleEditor
- ShapeEllipseEditor
- OnLBdown
- OnLBup
- OnMouseMove
- OnPaint

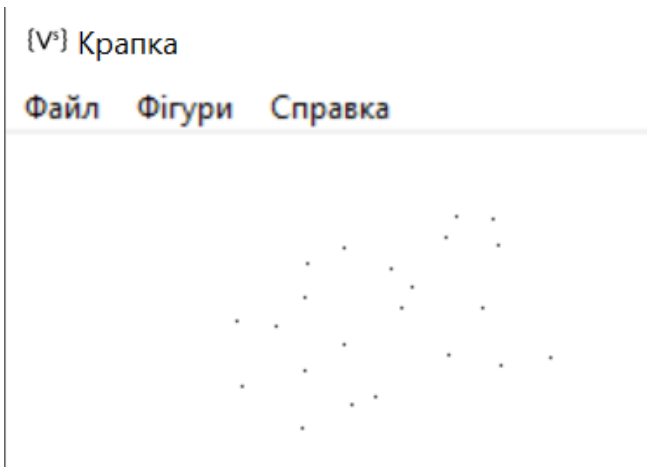
Скріншоти програми:

Початкове вікно:

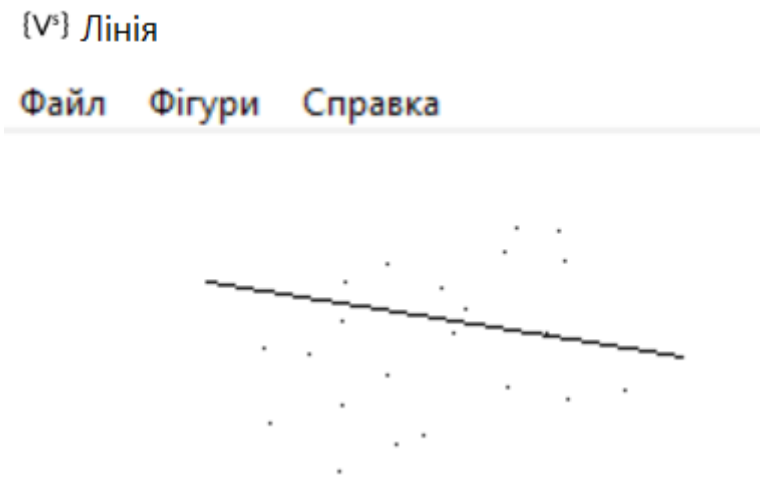
Вибір фігури:



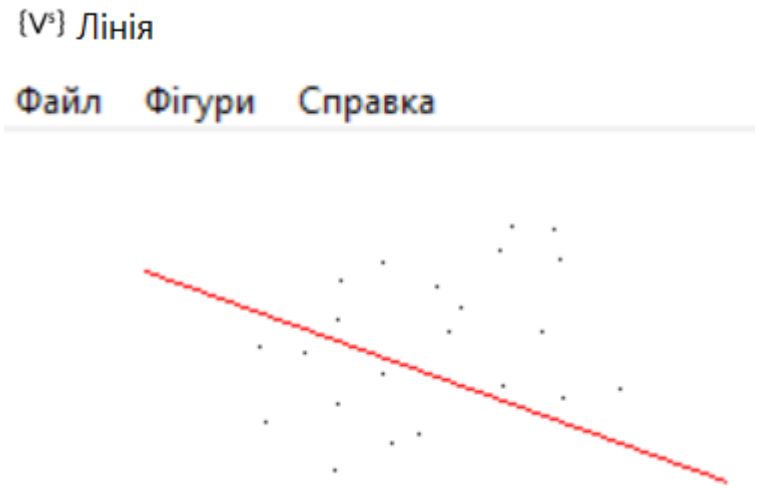
Введення крапок:



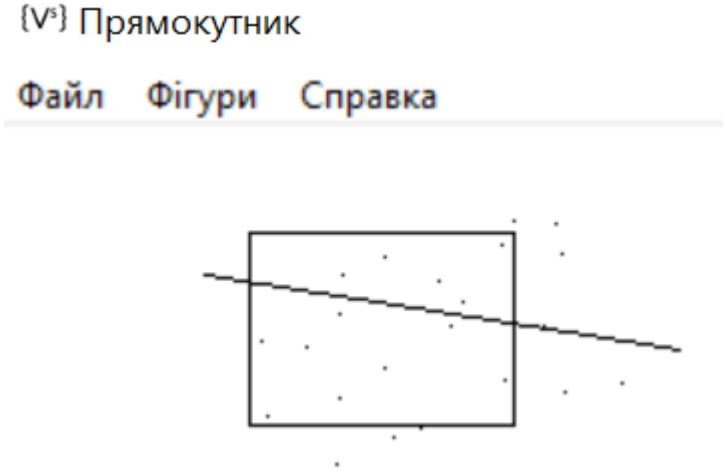
Введення ліній:



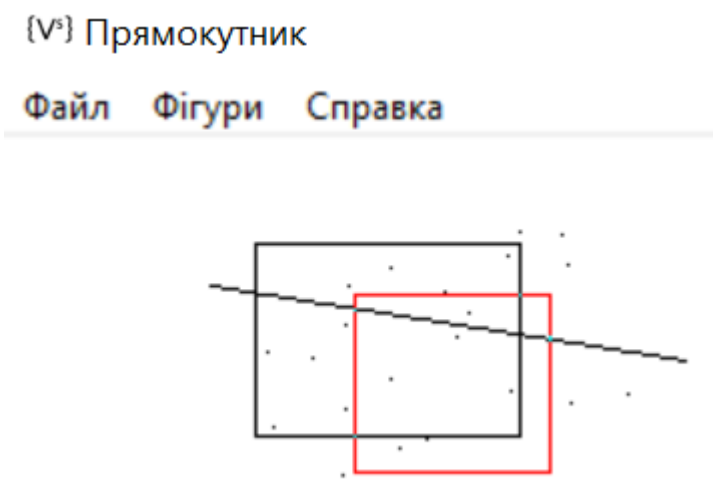
Гумовий слід ліній:



Введення прямокутників:

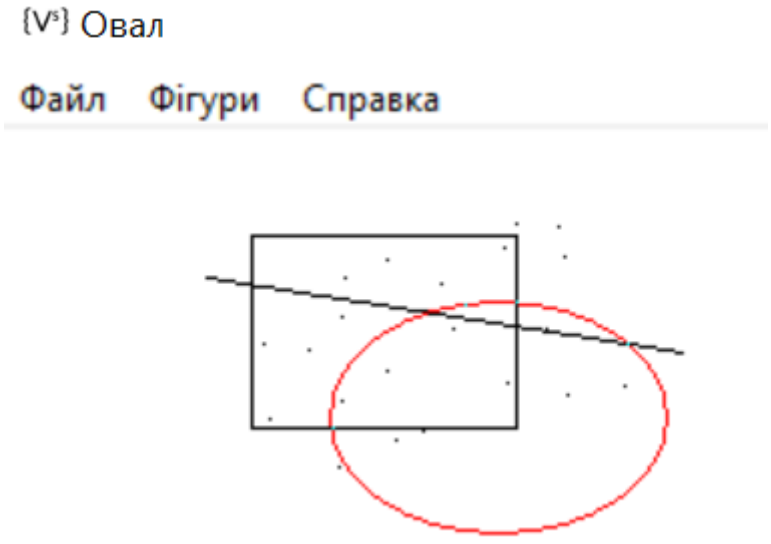


Гумовий слід прямокутників:



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.

2. 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

3. Що таке абстрактний клас і скільки їх 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.

4. 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)

5. 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.