Sketcher (Part 4)

Building Graphical User Interfaces (GUIs)

Using the Microsoft Foundation Classes (MFC)

All of these examples assume that Microsoft Visual C++ 2017 is the compiler being used. Based on a tutorial written by Dr. Rick Coleman.

You also must have the "Visual C++ MFC for x86 and x64" feature enabled in the VS Installer.

Exercise 5: Sketcher (Part 4) - a drawing program

Add a Rectangle class

Add a new **CRectangle.h** and **CRectangle.cpp** file. (Remember, right click the "Header Files" or "Source Files" directory. Select "Add->New Item..." from the popup menu. Select "Code" on the left and "C++ File (.cpp)" or "Header File (.h)" on the right and give it the appropriate name.

Copy and paste the following code into the **Rectangle.h** file.

```
//=========================================================
// CRectangle.h
// Class definition file for a Rectangle shape
// Author: Dr. Rick Coleman
//=========================================================
#pragma once

#include "Constants.h"
#include "CShape.h"

class CRectangle : public CShape
{
	public:
		CRectangle();
		~CRectangle();
		void Draw(CDC *pDC);
};
```

Copy and paste the following code into the **CRectangle.cpp** file.

```
//=========================================================
// CRectangle.cpp
// Class implementation file for a Rectangle shape
// Author: Dr. Rick Coleman
//=========================================================
#include "pch.h"
#include "CRectangle.h"

//---------------------------------
```

```
// Default constructor
//---------------------------------
CRectangle::CRectangle()
{
}

//---------------------------------
// Default destructor
//---------------------------------
CRectangle::~CRectangle()
{
}

//-----------------------------------------------
// Implement the draw function for Rectangles
//-----------------------------------------------
void CRectangle::Draw(CDC *pDC)
{
        // Create a pen for drawing
        CPen pen, *oldPen;
        // Create a brush for filling
        CBrush brush, *oldBrush = NULL;
        // If creating a transparent pen use PS_NULL style
        if(this->m_PenColor == COLOR_CLEAR)
                pen.CreatePen(PS_NULL, this->m_iPenWidth, COLOR_BLACK);
        else
                pen.CreatePen(this->m_iPenPattern, this->m_iPenWidth, this->m_PenColor);

        // If creating a transparent fill use a NULL_BRUSH
        if(this->m_BrushColor != COLOR_CLEAR)
        {
                if(this->m_iBrushPattern == BRUSH_PATTERN_SOLID)
                        brush.CreateSolidBrush(this->m_BrushColor);
                else
                        brush.CreateHatchBrush(this->m_iBrushPattern, this->m_BrushColor);
                oldBrush = pDC->SelectObject(&brush);
        }
        else
                pDC->SelectStockObject(NULL_BRUSH);

        // Set the drawing pen and hold the current pen
        oldPen = pDC->SelectObject(&pen);
        // Draw the shape
        pDC->Rectangle(this->m_EnclosingRect);
        // Reset the current pen
        pDC->SelectObject(oldPen);
        // Reset the current brush
        if(oldBrush != NULL)
                pDC->SelectObject(oldBrush);
        // Delete the pen we created to avoid memory leaks
        pen.DeleteObject();
        // Delete the brush we created if we did create one
        if(this->m_BrushColor != COLOR_CLEAR)
                brush.DeleteObject();
        else
                pDC->SelectStockObject(WHITE_BRUSH);
}
```

Add an Oval class

Add a new **COval.h** and **COval.cpp** file just as you did the Rectangle files.

Copy and paste the following code into the **COval.h** file.

```
//=============================================================
// COval.h
// Class definition file for an Oval shape
// Author: Dr. Rick Coleman
//=============================================================
#pragma once

#include "Constants.h"
#include "CShape.h"

class COval : public CShape
{
        public:
                COval();
                ~COval();
                void Draw(CDC *pDC);
};
```

Copy and paste the following code into the **COval.cpp** file.

```
//=============================================================
// COval.cpp
// Class implementation file for an Oval shape
// Author: Dr. Rick Coleman
//=============================================================
#include "pch.h"
#include "COval.h"

//---------------------------------
// Default constructor
//---------------------------------
COval::COval()
{
}

//---------------------------------
// Default destructor
//---------------------------------
COval::~COval()
{
}

//-----------------------------------------------
// Implement the draw function for Ovals
//-----------------------------------------------
void COval::Draw(CDC *pDC)
{
        // Create a pen for drawing
        CPen pen, *oldPen;
        // Create a brush for filling
        CBrush brush, *oldBrush = NULL;
        // If creating a transparent pen use PS_NULL style
        if(this->m_PenColor == COLOR_CLEAR)
                pen.CreatePen(PS_NULL, this->m_iPenWidth, COLOR_BLACK);
        else
                pen.CreatePen(this->m_iPenPattern, this->m_iPenWidth, this->m_PenColor);
        // If creating a transparent fill use a NULL_BRUSH
        if(this->m_BrushColor != COLOR_CLEAR)
        {
                if(this->m_iBrushPattern == BRUSH_PATTERN_SOLID)
                        brush.CreateSolidBrush(this->m_BrushColor);
                else
                        brush.CreateHatchBrush(this->m_iBrushPattern, this->m_BrushColor);
                oldBrush = pDC->SelectObject(&brush);
        }
        else
                pDC->SelectStockObject(NULL_BRUSH);

        // Set the drawing pen and hold the current pen
        oldPen = pDC->SelectObject(&pen);
        // Draw the shape
        pDC->Ellipse(this->m_EnclosingRect);
        // Reset the current pen
        pDC->SelectObject(oldPen);
        // Reset the current brush
        if(oldBrush != NULL)
                pDC->SelectObject(oldBrush);
        // Delete the pen we created to avoid memory leaks
        pen.DeleteObject();
        // Delete the brush we created if we did create one
        if(this->m_BrushColor != COLOR_CLEAR)
                brush.DeleteObject();
        else
                pDC->SelectStockObject(WHITE_BRUSH);
}
```

Add a Curve class

Add a new **CCurve.h** and **CCurve.cpp** file just as you did the Rectangle files.

Copy and paste the following code into the **CCurve.h** file.

```
//=============================================================
// CCurve.h
// Class definition file for a Curve shape
// Author: Dr. Rick Coleman
//=============================================================
#pragma once

#include "Constants.h"
#include "CShape.h"
#include <vector>

using namespace std;

class CCurve : public CShape
{
        private:
                vector<CPoint> *m_vPoints;  // Points defining the curve

        public:
                CCurve();
                ~CCurve();
                void Draw(CDC *pDC);
                void addPoint(CPoint *pPt);
                void addPoint(int X, int Y);
                void setEnclosingRect(CRect pRect);
                void setEnclosingRect(int left, int top, int right, int bottom);
                void setEnclosingRect(CPoint ul, CPoint lr);
};
```

Copy and paste the following code into the **CCurve.cpp** file.

```
//===========================================================
// CCurve.cpp
// Class implementation file for a Curve shape
// Author: Dr. Rick Coleman
//===========================================================
#include "pch.h"
#include "CCurve.h"

//---------------------------------
// Default constructor
//---------------------------------
CCurve::CCurve()
{
        m_vPoints = new vector<CPoint>();
}

//---------------------------------
// Default destructor
//---------------------------------
CCurve::~CCurve()
{
        m_vPoints->clear();
        delete m_vPoints;
}

//-----------------------------------------------
// Implement the draw function for Curves
//-----------------------------------------------
void CCurve::Draw(CDC *pDC)
{
        // Create a pen for drawing
        CPen pen, *oldPen;
        // If creating a transparent pen use PS_NULL style
        if(this->m_PenColor == COLOR_CLEAR)
                pen.CreatePen(PS_NULL, this->m_iPenWidth, COLOR_BLACK);
        else
                pen.CreatePen(this->m_iPenPattern, this->m_iPenWidth, this->m_PenColor);

        // Set the drawing pen and hold the current pen
        oldPen = pDC->SelectObject(&pen);
        // Draw the curve
        pDC->MoveTo(m_vPoints->begin()->x, m_vPoints->begin()->y);
        for(vector<CPoint>::iterator itr = m_vPoints->begin();
                        itr != m_vPoints->end(); itr++)
        {
                pDC->LineTo(itr->x, itr->y);
        }
        // Reset the current pen
        pDC->SelectObject(oldPen);
        // Delete the pen we created to avoid memory leaks
        pen.DeleteObject();
}

//-----------------------------------------------
// Add a point to the vector
//-----------------------------------------------
void CCurve::addPoint(CPoint *pPt)
```

```
{
        // Copy the point
        CPoint *newPt = new CPoint(pPt->x, pPt->y);
        m_vPoints->push_back(*newPt);
}

//------------------------------------------------
// Add a point to the vector
//------------------------------------------------
void CCurve::addPoint(int X, int Y)
{
        // Copy the point
        CPoint *newPt = new CPoint(X, Y);
        m_vPoints->push_back(*newPt);
}

// Override all the setEnclosingRectangle functions to
//  add the two points as the first points in the curve

//---------------------------------------------------
// Set the enclosing rectangle from another rectangle
//---------------------------------------------------
void CCurve::setEnclosingRect(CRect rect)
{
        this->addPoint(rect.left, rect.top);
        this->addPoint(rect.right, rect.bottom);
}

//--------------------------------------------------
// Set the enclosing rectangle from X,Y coordinates
//--------------------------------------------------
void CCurve::setEnclosingRect(int left, int top, int right, int bottom)
{
        this->addPoint(left, top);
        this->addPoint(right, bottom);
}

//----------------------------------------------------
// Set the enclosing rectangle from two CPoint objects
//----------------------------------------------------
void CCurve::setEnclosingRect(CPoint ul, CPoint lr)
{
        this->addPoint(ul.x, ul.y);
        this->addPoint(lr.x, lr.y);
}
```

Don't forget to add a #include for each of the .h files in *programName*View.h.

Finishing Up

Add functions to ***programName*Doc** class

We need to add two functions to the document class to let us store the CShape objects and draw all the objects. We also need to create a vector to store pointers to the CShape objects in.

Add the vector to the list of variables you have created in the ***programName*Doc.h** file. Don't forget to add the #include at the top of the file.

```
#include "Constants.h"
#include "CShape.h"
#include <vector>

using namespace std;
// ...
// Skip down to the variables to add the code below
// ...
private:
        int m_iCurShape;
        COLORREF m_CurPenColor;
        int m_iCurPenWidth;
        int m_iCurPenPattern;
        COLORREF m_CurBrushColor;
        int m_iCurBrushPattern;
        vector<CShape *> *m_vShapes;            // This is new
        vector<CShape *>::iterator m_Itr;       // This is new


public:
        int getCurrentShape();
        COLORREF getCurrentPenColor();
        int getCurrentPenWidth();
        int getCurrentPenPattern();
        COLORREF getCurrentBrushColor();
        int getCurrentBrushPattern();
        void AddShape(CShape *s);       // This is new
        CShape *getFirstShape();        // This is new
        CShape *getNextShape();         // This is new
```

Now add the function code to the ***progranName*Doc.cpp** file.

Add the following line to the constructor

```
        m_vShapes = new vector<CShape *>();
```

Next add the following functions

```
//----------------------------------------------------
// Get the first shape in the vector
//----------------------------------------------------
CShape *CDemo05SketcherDoc::getFirstShape()
{
        if(m_vShapes->size() == 0)
                return NULL;
        m_Itr = m_vShapes->begin();

        return (CShape *)(*m_Itr); // Remember m_Itr is a pointer to a pointer so
                                   //  we must dereference it to get the pointer to
                                   //  a CShape object.
}
//----------------------------------------------------
// Get the next shape in the vector
//----------------------------------------------------
CShape *CDemo05SketcherDoc::getNextShape()
{
        m_Itr++;
        if(m_Itr != m_vShapes->end())
        {
                return (CShape *)(*m_Itr); // Remember m_Itr is a pointer to a pointer so
                                           //  we must dereference it to get the pointer to
```

```
                                                //  a CShape object.
        }
        else
        {
                return NULL;
        }

}

//----------------------------------------------------
// Add a shape to the vector for this drawing
//----------------------------------------------------
void CDemo05SketcherDoc::AddShape(CShape *s)
{
        m_vShapes->push_back(s);
}
```

Modify functions in ***progranName*View** class

Change the **OnDraw()** code in ***programName*View.cpp** to the following:

```
void CDemo05SketcherView::OnDraw(CDC* pDC)
{
        CDemo05SketcherDoc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if (!pDoc)
                return;

        // Draw all the shapes
        CShape *s;
        s = pDoc->getFirstShape();
        while(s != NULL)
        {
                s->Draw(pDC);
                s = pDoc->getNextShape();
        }
}
```

Now you might be asking why don't we just call a draw function in the document class and let it do all the drawing since it has the vector of pointers to all the shapes. We could do that but we would then have to pass a Device Context to the document for it to draw into. But, the main reason we don't do this is because it violates an object-oriented programming principle. The document object should be where all data is stored to define a drawing. The view object should be where all the drawing is done. So, we leave all the drawing in the view and let it call the document for each shape to be drawn.

Modify the code in the **CreateShape()** function so that you can create all of the shapes.

```
//--------------------------------------------------------------
// Create a shape based on the current settings.
//--------------------------------------------------------------
CShape *CDemo05SketcherView::CreateShape(void)
{
        // Make sure we have a valid document just to be safe
        CDemo05SketcherDoc *pDoc = GetDocument();
        ASSERT_VALID(pDoc);                     // Crash and burn if it is not good

        // Select the shape type to create
        switch(pDoc->getCurrentShape())
        {
                case LINE : // Create a LINE object and return it
                {
                        CLine *newLine = new CLine();
                        newLine->setEnclosingRect(this->m_StartPoint, this->m_EndPoint);
                        newLine->setPenColor(pDoc->getCurrentPenColor());
                        newLine->setPenWidth(pDoc->getCurrentPenWidth());
                        newLine->setPenPattern(pDoc->getCurrentPenPattern());
                        return newLine;
                        break;
                }
                case RECTANGLE : // Create a RECTANGLE object and return it
                {
                        CRectangle *newRect = new CRectangle();
                        newRect->setEnclosingRect(this->m_StartPoint, this->m_EndPoint);
                        newRect->setPenColor(pDoc->getCurrentPenColor());
                        newRect->setPenWidth(pDoc->getCurrentPenWidth());
                        newRect->setPenPattern(pDoc->getCurrentPenPattern());
                        newRect->setBrushColor(pDoc->getCurrentBrushColor());
                        newRect->setBrushPattern(pDoc->getCurrentBrushPattern());
                        return newRect;
                        break;
                }
                case OVAL : // Create a OVAL object and return it
                {
                        COval *newOval = new COval();
                        newOval->setEnclosingRect(this->m_StartPoint, this->m_EndPoint);
                        newOval->setPenColor(pDoc->getCurrentPenColor());
                        newOval->setPenWidth(pDoc->getCurrentPenWidth());
                        newOval->setPenPattern(pDoc->getCurrentPenPattern());
                        newOval->setBrushColor(pDoc->getCurrentBrushColor());
                        newOval->setBrushPattern(pDoc->getCurrentBrushPattern());
                        return newOval;
                        break;
                }
                case CURVE : // Create a CURVE object and return it
                {
                        CCurve *newCurve = new CCurve();
                        newCurve->setEnclosingRect(this->m_StartPoint, this->m_EndPoint);
                        newCurve->setPenColor(pDoc->getCurrentPenColor());
                        newCurve->setPenWidth(pDoc->getCurrentPenWidth());
                        newCurve->setPenPattern(pDoc->getCurrentPenPattern());
                        return newCurve;
                        break;
                }
                default :   // Oops! something is wrong.
                        AfxMessageBox(_T("Bad Shape code"), MB_OK);
                        AfxAbort(); // Crash and burn
                        return NULL;
        }
        return NULL;
}
```

Modify the code in the **OnMouseMove()** function to handle the special case of CCurve in which we are saving all the points in the vector.

```
//--------------------------------------------------------------
// Handle mouse move events
//--------------------------------------------------------------
void CDemo05SketcherView::OnMouseMove(UINT nFlags, CPoint point)
{
        if(nFlags & MK_LBUTTON)  // Check to see if the left button is down
        {
                // Define a Device Contest object for this vies
                CClientDC aDC(this);
                aDC.SetROP2(R2_NOTXORPEN);              // Set the XOR drawing mode

                m_EndPoint = point;  // If so save the current point
```

```
            // Test for a previous temporary CShape object
            if(m_pTempShape != NULL)
            {
                    // Check to see if we are drawing a curve and need to save points
                    if(GetDocument()->getCurrentShape() == CURVE)
                    {
                            // We are drawing a curve so add a point
                            static_cast<CCurve*>(m_pTempShape)->addPoint(&m_EndPoint);
                            aDC.SetROP2(R2_COPYPEN); // Draw normal if curve
                            this->m_pTempShape->Draw(&aDC); // Draw it
                            return; // Done
                    }
                    // Redraw the old element so it disappears
                    this->m_pTempShape->Draw(&aDC);
                    delete m_pTempShape;    // Delete the old one
                    m_pTempShape = NULL;    // Reset the pointer to NULL
            }

            // Create a new temporary CShape object
            this->m_pTempShape = CreateShape();
            // Draw the temporary CShape object
            this->m_pTempShape->Draw(&aDC);
        }

        CView::OnMouseMove(nFlags, point);
}
```

Remember to go back and uncomment the line GetDocument()->AddShape(m_pTempShape); in the OnLButtonUp function of the "View" class.

Compile and run your Sketcher application and see how it works.

And now what...

There are many other things that could be added. Probably the first thing you would want to do is add the ability to save a drawing document and then reload it. You might also want to add other drawing features (shapes, special pens, and brushes).

All these are left as an exercise for the student...;-)