TEMA 16

Form1.h Diseño -> Form1 -> Eventos -> doble click en evento Paint

Color de fondo del formulario, cambiamos propiedad BackColor a White

Form1 -> Eventos -> doble click en MouseDown, MouseMove y MouseUp

Variables en Form1 para saber si se está dibujando (Vista de clases -> Form1 -> Agregar -> Agregar variable)

Acceso	Tipo de variable	Nombre de variable
private	bool	drawing
private	Point	firstPoint

(automáticamente se añade al constructor `drawing(false)` y `firstPoint(0)`, pero mejor comprobar)

Manejador de MouseDown

```
private: System::Void Form1_MouseDown(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (e->Button == System::Windows::Forms::MouseButtons::Left)
   {
      drawing = true;
      firstPoint = e->Location;
   }
}
```

Definición de los elementos en una nueva clase Elements.h

Ventana Explorador de Soluciones -> botón derecho sobre el nombre del proyecto -> Agregar -> Nuevo elemento -> (en el grupo Código) Archivo de encabezado (.h) -> nombre Elements.h y con este código

```
// Elements.h
// Defines element types

#pragma once

using namespace System;
using namespace System::Drawing;

namespace CLRSketcher (cambiar a nuestro nombre)
{
   public ref class Element abstract
   {
      protected:
         Point position;
         Color color;
         System::Drawing::Rectangle boundRect;

      public:
         virtual void Draw(Graphics^ g) abstract;
   };
}
```

Dentro del namespace de Elements.h definimos el elemento línea

```
public ref class Line : Element
{
   protected:
      Point end;

   public:
      // Constructor
      Line(Color color, Point start, Point end)
      {
         this ->color = color;
         position = start;
         this ->end = end;
         boundRect = System::Drawing::Rectangle(Math::Min(position.X, end.X),
                                                Math::Min(position.Y, end.Y),
                                                Math::Abs(position.X - end.X),
                                                Math::Abs(position.Y - end.Y));
         // Provide for lines that are horizontal or vertical
         if (boundRect.Width < 2) boundRect.Width = 2;
         if (boundRect.Height < 2) boundRect.Height = 2;
      }

      // Function to draw a line
      virtual void Draw(Graphics^ g) override
      {
         // Code to draw a line...
      }
};
```

Añadimos el pincel y completamos la función Draw

```
public ref class Element abstract
{
   protected:
      Pen^ pen;
      ...
};

Line(Color color, Point start, Point end)
{
   pen = gcnew Pen(color);
   this->color = color;
   ...

   ...
}

virtual void Draw(Graphics^ g) override
{
   g->DrawLine(pen, position, end);
}
```

Definimos los demás elementos

```
public ref class Rectangle : Element
{
   protected:
      int width;
      int height;

   public:
      Rectangle(Color color, Point p1, Point p2)
      {
         pen = gcnew Pen(color);
         this ->color = color;
         position = Point(Math::Min(p1.X, p2.X), Math::Min(p1.Y,p2.Y));
         width = Math::Abs(p1.X - p2.X);
         height = Math::Abs(p1.Y - p2.Y);
         boundRect = System::Drawing::Rectangle(position, Size(width, height));
```

```
        }

        virtual void Draw(Graphics^ g) override
        {
            g->DrawRectangle(pen, position.X, position.Y, width, height);
        }
};

public ref class Circle : Element
{
    protected:
        int width;
        int height;

    public:
        Circle(Color color, Point center, Point circum)
        {
            pen = gcnew Pen(color);
            this ->color = color;
            int radius = safe_cast<int>(Math::Sqrt((center.X-circum.X)*(center.X-circum.X) +
                                                   (center.Y-circum.Y)*(center.Y-circum.Y)));
            position = Point(center.X - radius, center.Y - radius);
            width = height = 2*radius;
            boundRect = System::Drawing::Rectangle(position, Size(width, height));
        }

        virtual void Draw(Graphics^ g) override
        {
            g->DrawEllipse(pen, position.X, position.Y, width, height);
        }
};

public ref class Curve : Element
{
    private:
        vector<Point>^ points;

    public:
        Curve(Color color, Point p1, Point p2)
        {
            pen = gcnew Pen(color);
            this ->color = color;
            points = gcnew vector<Point>();
            position = p1;
            points->push_back(p2 - Size(position));
            // Find the minimum and maximum coordinates
            int minX = p1.X < p2.X ? p1.X : p2.X;
            int minY = p1.Y < p2.Y ? p1.Y : p2.Y;
            int maxX = p1.X > p2.X ? p1.X : p2.X;
            int maxY = p1.Y > p2.Y ? p1.Y : p2.Y;
            int width = Math::Max(2, maxX - minX);
            int height = Math::Max(2, maxY - minY);
            boundRect = System::Drawing::Rectangle(minX, minY, width, height);
        }

        // Add a point to the curve
        void Add(Point p)
        {
            points->push_back(p - Size(position));
            // Modify the bounding rectangle to accommodate the new point
            if (p.X < boundRect.X)
            {
                boundRect.Width = boundRect.Right - p.X;
                boundRect.X = p.X;
            }
            else if (p.X > boundRect.Right)
                boundRect.Width = p.X - boundRect.Left;
            if (p.Y < boundRect.Y)
            {
                boundRect.Height = boundRect.Bottom - p.Y;
                boundRect.Y = p.Y;
            }
            else if (p.Y > boundRect.Bottom)
                boundRect.Height = p.Y - boundRect.Top;
        }
```

```
        virtual void Draw(Graphics^ g) override
        {
            Point previous(position);
            Point temp;
            for each (Point p in points)
            {
                temp = position + Size(p);
                g->DrawLine(pen, previous, temp);
                previous = temp;
            }
        }
};
```

No olvidemos añadir `#include <cliext/vector>` y `using namespace cliext;` al principio de Elements.h

Añadimos a Form1.h el siguiente elemento

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	Element^	tempElement	nullptr

No olvidemos añadir `#include "Elements.h"` al principio de Form1.h

Añadimos al manejador MouseMove de Form1.h

```
private: System::Void Form1_MouseMove(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (drawing)
   {
      switch (elementType)
      {
         case ElementType::LINE:
            tempElement = gcnew Line(color, firstPoint, e->Location);
            break;
         case ElementType::RECTANGLE:
            tempElement = gcnew Rectangle(color, firstPoint, e->Location);
            break;
         case ElementType::CIRCLE:
            tempElement = gcnew Circle(color, firstPoint, e->Location);
            break;
         case ElementType::CURVE:
            if (tempElement)
               safe_cast <Curve^>(tempElement)->Add(e->Location);
            else
               tempElement = gcnew Curve(color, firstPoint, e->Location);
            break;
      }
      Invalidate();
   }
}
```

Y para MouseUp

```
private: System::Void Form1_MouseUp(System::Object^ sender,
                                    System::Windows::Forms::MouseEventArgs^ e)
{
   if (!drawing)
      return;
   if (tempElement)
   {
      // Store the element in the sketch...
      tempElement = nullptr;
      // Invalidate();
   }
   drawing = false;
}
```

Y para Paint

```
private: System::Void Form1_Paint(System::Object^ sender,
                                  System::Windows::Forms::PaintEventArgs^ e)
{
   Graphics^ g = e->Graphics;
   // Code to draw the sketch...
   if (tempElement != nullptr)
   {
      tempElement->Draw(g);
   }
}
```

Aspecto del programa al finalizar el Tema 16