TEMA 18

Para añadir diálogos (creamos formularios pero luego les damos la forma de diálogos): Explorador de Soluciones -> botón derecho sobre nuestro proyecto -> Agregar -> Nuevo elemento -> (en el grupo de plantillas UI) Windows Form -> PenDialog.cs

Cambiamos la propiedad `FormBorderStyle` a `FixedDialog` y comprobamos que las propiedades `ControlBox`, `MinimizeBox`, y `MaximizeBox` estén inicializadas como `false`. También cambiamos `Text` a `Set Pen Width`

Añadimos un `GroupBox` y cambiamos su propiedad `Text` a `Select Pen Width`, y `(name)` a `penWidthGroupBox`

Añadimos 6 RadioButtons de forma estructurada para los espesores del pincel. Sus propiedades `Text` irán desde `Pen Width 1` hasta `6`, y `(name)` desde `penWidthButton1` hasta `6`, cambiando la propiedad `Checked` del botón 1 a `true`

Añadimos dos botones, Aceptar y Cancelar, con propiedad `Text OK` y `Cancel` respectivamente, `(name) penWidthOK` y `penWidthCancel` respectivamente, y en las propiedades `AcceptButton` y `CancelButton` del diálogo principal, les hacemos corresponder cada botón respectivamente, y las propiedades `DialogResult` de cada botón deben ser `OK` y `Cancel` respectivamente

Añadimos `#include "PenDialog.h"` en Form1.h y la variable

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
`private`	`PenDialog^`	`penDialog`	`gcnew PenDialog()`

Añadimos el siguiente código a PenDialog.h justo después de `#pragma endregion`

```
public: property float PenWidth
{
    float get()
    {
        if (penWidthButton1->Checked)
            return 1.0f;
        if (penWidthButton2->Checked)
            return 2.0f;
        if (penWidthButton3->Checked)
            return 3.0f;
        if (penWidthButton4->Checked)
            return 4.0f;
        if (penWidthButton5->Checked)
            return 5.0f;
        return 6.0f;
    }
}
```

Necesitamos un botón en la barra de herramientas para llamar a este diálogo

Tamaño	Color	ID	Nombre de archivo	
16x16	24-bit	IDB_PENWIDTH	penwidth.bmp	

Añadimos un separador y un botón nuevos a la barra de herramientas de Form1.h, con `(name) penWidthButton` y `ToolTipText Change pen width` y le insertamos el bitmap como `Image`, y le añadimos el siguiente código a su evento Click

```
private: System::Void penWidthButton _Click(System::Object^ sender,
                                            System::EventArgs^ e)
{
   if (penDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK)
   {
      // Get the pen width ...
   }
}
```

Añadimos en Form1.h la variable

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	float	penWidth	1.0f

Y completamos el manejador con `penWidth = penDialog->PenWidth;`

Modificamos las funciones Line, Rectangle, Circle y Curve de la clase Elements.h de la siguiente forma, por ejemplo para Line (cuidado con los diferentes parámetros de entrada, por ejemplo P1 en vez de start)

```
Line(Color color, Point start, Point end, float penWidth)
{
   pen = gcnew Pen(color, penWidth);
   this->color = color;
   position = start;
   this->end = end - Size(start);
   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));
   boundRect.Inflate(safe_cast<int>(penWidth), safe_cast<int>(penWidth));
}
```

Y ahora sólo queda ampliar la función MouseMove que crea los elementos

```
private: System::Void Form1_MouseMove(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (drawing)
   {
      if (tempElement)
         Invalidate(tempElement->Bound); // The old element region
      switch (elementType)
      {
         case ElementType::LINE:
            tempElement = gcnew Line(color, firstPoint, e->Location, penWidth);
            break;
         case ElementType::RECTANGLE:
            tempElement = gcnew Rectangle(color, firstPoint, e->Location,
penWidth);
```

```
            break;
         case ElementType::CIRCLE:
            tempElement = gcnew Circle(color, firstPoint, e->Location, penWidth);
            break;
         case ElementType::CURVE:
            if (tempElement)
               safe_cast<Curve^>(tempElement)->Add(e->Location);
            else
               tempElement = gcnew Curve(color, firstPoint, e->Location, penWidth);
            break;
      }
      // Rest of the code as before...
   }
}
```

Añadimos un ComboBox a la barra de herramientas de Form1.h para seleccionar diferentes estilos de línea, entonces cambiamos `(name)` a `lineStyleComboBox`. En la propiedad `Items`, seleccionando `(Collection)` podemos añadir todas las opciones que necesitamos, que serán `Solid lines`, `Dashed lines`, `Dotted lines`, `Dash-dotted lines` y `Dash dot dotted lines` y corresponderán a índices del 0 al 4. También cambiamos el tamaño en `Size` a `150,25`, la propiedad `FlatStyle` a `Standard`, y el `ToolTipText` a `Choose line style` (no tiene ToolTipText en el 2008). Al constructor de Form1.h, después de `//TODO`, le añadimos `lineStyleComboBox->SelectedIndex = 0;`

Para guardar el estilo de línea creamos en Form1.h la variable

Acceso	Tipo de variable	Nombre de variable
`private`	`DashStyle`	`lineStyle`

Añadimos `using namespace System::Drawing::Drawing2D;` (si no lo añadimos, tendríamos que proporcionar `System::Drawing::Drawing2D::DashStyle` como el tipo de variable de `lineStyle`, esta clase contiene todos los tipos de línea que necesitamos)

Para el evento `SelectedIndexChanged` del ComboBox añadimos el siguiente manejador

```
private: System::Void lineStyleComboBox_SelectedIndexChanged(System::Object^ sender,
                                                             System::EventArgs^ e)
{
   switch (lineStyleComboBox->SelectedIndex)
   {
      case 1:
         lineStyle = DashStyle::Dash;
         break;
      case 2:
         lineStyle = DashStyle::Dot;
         break;
      case 3:
         lineStyle = DashStyle::DashDot;
         break;
      case 4:
         lineStyle = DashStyle::DashDotDot;
         break;
      default:
         lineStyle = DashStyle::Solid;
         break;
   }
}
```

Añadimos `using namespace System::Drawing::Drawing2D;` a la clase Elements.h para poder usar DashStyle en los elementos, como por ejemplo en Line, e igualmente en los demás

```
Line(Color color, Point start, Point end, float penWidth, DashStyle style)
{
   pen = gcnew Pen(color, penWidth);
   pen->DashStyle = style;
   // Rest of the code for the function as before...
}
```

Y por último las llamadas en la función MouseMove (sólo añadir `lineStyle` al final de las llamadas)

```
private: System::Void Form1_MouseMove(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (drawing)
   {
      if (tempElement)
         Invalidate(tempElement->bound); // Invalidate old element area
      switch (elementType)
      {
         case ElementType::LINE:
            tempElement = gcnew Line(color, firstPoint, e->Location,
                                     penWidth, lineStyle);
            break;
         case ElementType::RECTANGLE:
            tempElement = gcnew Rectangle(color, firstPoint, e->Location,
                                          penWidth, lineStyle);
            break;
         case ElementType::CIRCLE:
            tempElement = gcnew Circle(color, firstPoint, e->Location,
                                       penWidth, lineStyle);
            break;
         case ElementType::CURVE:
            if (tempElement)
               safe_cast<Curve^>(tempElement)->Add(e->Location);
            else
               tempElement = gcnew Curve(color, firstPoint, e->Location,
                                         penWidth, lineStyle);
            break;
      }
      // Rest of the code for the function as before...
   }
}
```

Para añadir texto, incluímos `TEXT` al enumerador ElementType de Form1.h, y en su Diseño, un botón Text con `(name) textToolStripMenuItem` en el MenuStrip de los elementos, creamos un manejador con el evento Click, y un bitmap para un botón en la barra de herramientas con ToolTipText Write text. También tenemos que tener en cuenta que funcionen los Check, y no olvidarnos de incluir Text en el ContextMenu también

Tamaño	Color	ID	Nombre de archivo	
16x16	24-bit	IDB_TEXT	text.bmp	

```
enum class ElementType {LINE, RECTANGLE, CIRCLE, CURVE, TEXT};
```

Tras cambiar (name) del nuevo botón de texto de la barra de herramientas a toolStripTextButton, ampliar el manejador elementToolStripMenuItem_DropDownOpening

```
textToolStripMenuItem->Checked = elementType == ElementType::TEXT;
```

Ampliar manejador SetElementTypeButtonsState

```
toolStripTextButton->Checked = elementType == ElementType::TEXT;
```

En el manejador contextMenuStrip1_Opening, después de cambiar su (name) por textContextMenuItem

```
contextMenuStrip1->Items->Add(textContextMenuItem);
textContextMenuItem->Checked = elementType == ElementType::TEXT;
```

Y finalmente el manejador del botón del MenuStrip quedaría como

```
private: System::Void textToolStripMenuItem_Click(System::Object^ sender,
                                                  System::EventArgs^ e)
{
   elementType = ElementType::TEXT;
   SetElementTypeButtonsState();
}
```

Añadimos a Form1.h la variable

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	System::Drawing::Font^	textFont	Font (en //TODO)

Y un botón para el tipo de fuente a la barra de herramientas con (name) toolStripFontButton. Al que incluiremos código en su manejador a través del evento Click

Tamaño	Color	ID	Nombre de archivo	
16x16	24-bit	IDB_FONT	font.bmp	

```
private: System::Void toolStripFontButton_Click(System::Object^ sender,
                                                System::EventArgs^ e)
{
   if (fontDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK)
   {
      textFont = fontDialog1->Font;
   }
}
```

Para acceder al diálogo de cambio de fuente, añadimos la herramienta `FontDialog` a Form1.h desde la ventana de diseño

Añadimos un nuevo elemento como texto a los ya existentes en Elements.h

```
public ref class TextElement : Element
{
   protected:
      String^ text;
      SolidBrush^ brush;
      Font^ font;
   public:
      TextElement(Color color, Point p, String^ text, Font^ font)
      {
         this ->color = color;
         brush = gcnew SolidBrush(color);
         position = p;
         this ->text = text;
         this ->font = font;
         int height = font->Height; // Height of text string
         int width = static_cast<int>(font->Size*text->Length); // Width of string
         boundRect = System::Drawing::Rectangle(position, Size(width, height));
         boundRect.Inflate(2,2); // Allow for descenders
      }
      virtual void Draw(Graphics^ g) override
      {
         brush->Color = highlighted ? highlightColor : color;
         g->TranslateTransform(safe_cast<float>(position.X),
                               safe_cast<float>(position.Y));
         g->DrawString(text, font, brush, Point(0,0));
         g->ResetTransform();
      }
};
```

Creamos un nuevo formulario en forma de diálogo: Explorador de Soluciones -> click derecho sobre nuestro proyecto -> Agregar -> Nuevo elemento -> (en el grupo de plantillas UI) Windows Form -> TextDialog.cs

Cambiamos Text a `Create Text Element`, y como con PenDialog.h, la propiedad `FormBorderStyle` a `FixedDialog` y las propiedades `MaximizeBox`, `MinimizeBox`, y `ControlBox` a `false`

Añadimos los botones del Cuadro de herramientas, OK y Cancel, con su correspondiente propiedad `DialogResult` adecuada a cada uno (OK y Cancel respectivamente), con `(name)` `textOKButton` y `textCancelButton` respectivamente, y las propiedades `AcceptButton` y `CancelButton` de TextDialog.h, a `textOKButton` y `textCancelButton` respectivamente

También añadimos un TextBox a TextDialog.h, en su propio checkbox cambiamos a enable su propiedad Multiline

Para asegurarnos del correcto orden de los TabIndex, seleccionamos los tres objetos, y en Ver -> Orden de tabulación, comprobamos que el TextBox tiene el 0, y que el botón OK y Cancel el 1 y el 2 respectivamente

Añadimos en TextDialog.h tras `#pragma endregion`

```
// Property to access the text in the edit box
public: property String^ TextString
{
   String^ get() { return textBox1->Text; }
   void set(String^ text) { textBox1->Text = text; }
}

// Set the edit box font
public: property System::Drawing::Font^ TextFont
{
   void set(System::Drawing::Font^ font) { textBox1->Font = font; }
}
```

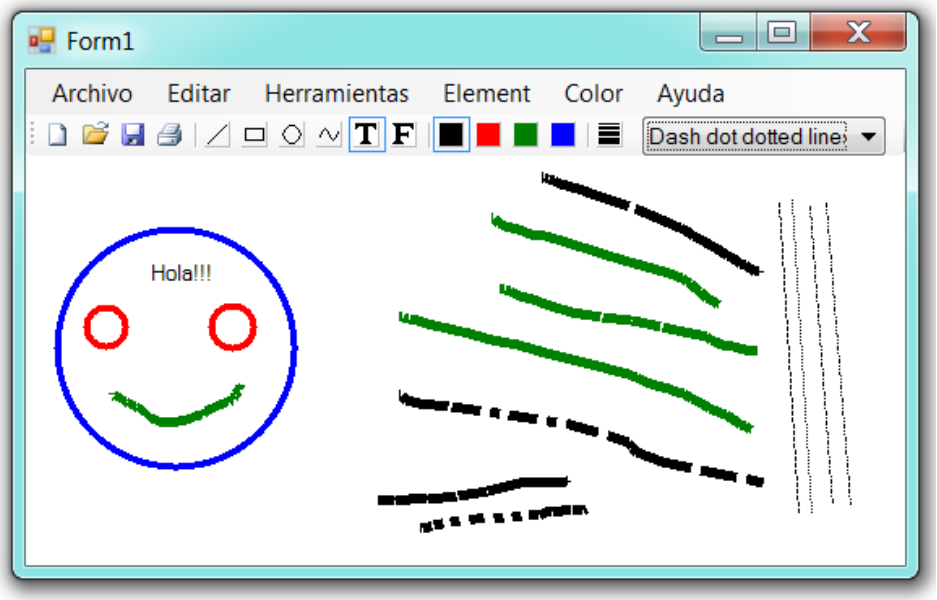
Añadimos en Form1.h `#include "TextDialog.h"` y la variable

Acceso	Tipo de variable	Nombre de variable	Inicialización en constructor
private	TextDialog^	textDialog	gcnew TextDialog()

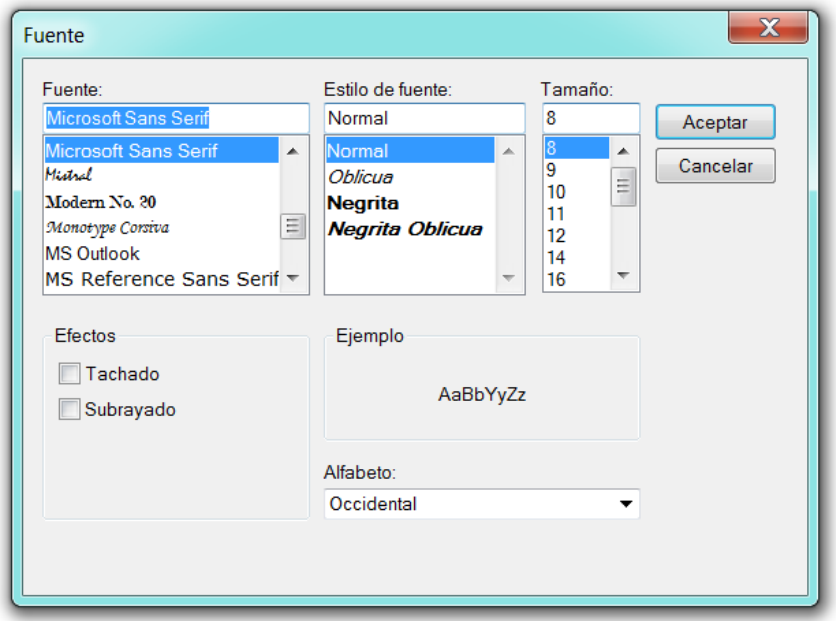
Y por último modificamos el manejador MouseDown de Form1.h de la siguiente manera

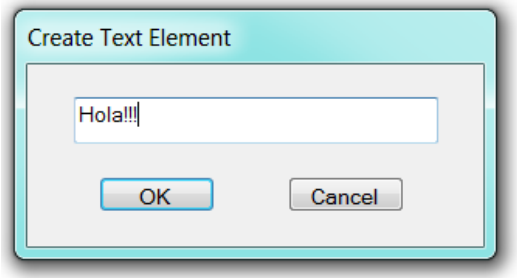
```
private: System::Void Form1_MouseDown(System::Object^ sender,
                                      System::Windows::Forms::MouseEventArgs^ e)
{
   if (e->Button == System::Windows::Forms::MouseButtons::Left)
   {
      firstPoint = e->Location;
      if (mode == Mode::Normal)
      {
         if (elementType == ElementType::TEXT)
         {
            textDialog->TextString = L""; // Reset the text box string
            textDialog->TextFont = textFont; // Set the font for the edit box
            if (textDialog->ShowDialog() == System::Windows::Forms::DialogResult::OK)
            {
               tempElement = gcnew TextElement(color, firstPoint,
                                               textDialog->TextString, textFont);
               sketch += tempElement;
               Invalidate(tempElement->bound); // The text element region
               tempElement = nullptr;
               Update();
            }
            drawing = false;
         }
         else
         {
            drawing = true;
         }
      }
   }
}
```

Aspecto del programa al finalizar el Tema 18



Y de sus diálogos de Espesor del Pncel, de Fuentes para el texto, y de Texto


Fuente
Fuente:
Estilo de fuente:
Tamaño:
Microsoft Sans Serif
Normal
8
Aceptar
Cancelar
Microsoft Sans Serif
Mistral
Modern No. 20
Monotype Corsiva
MS Outlook
MS Reference Sans Serif
Normal
Oblicua
Negrita
Negrita Oblicua
8
9
10
11
12
14
16
Efectos
Tachado
Subrayado
Ejemplo
AaBbYyZz
Alfabeto:
Occidental


Create Text Element
Hola!!!
OK
Cancel