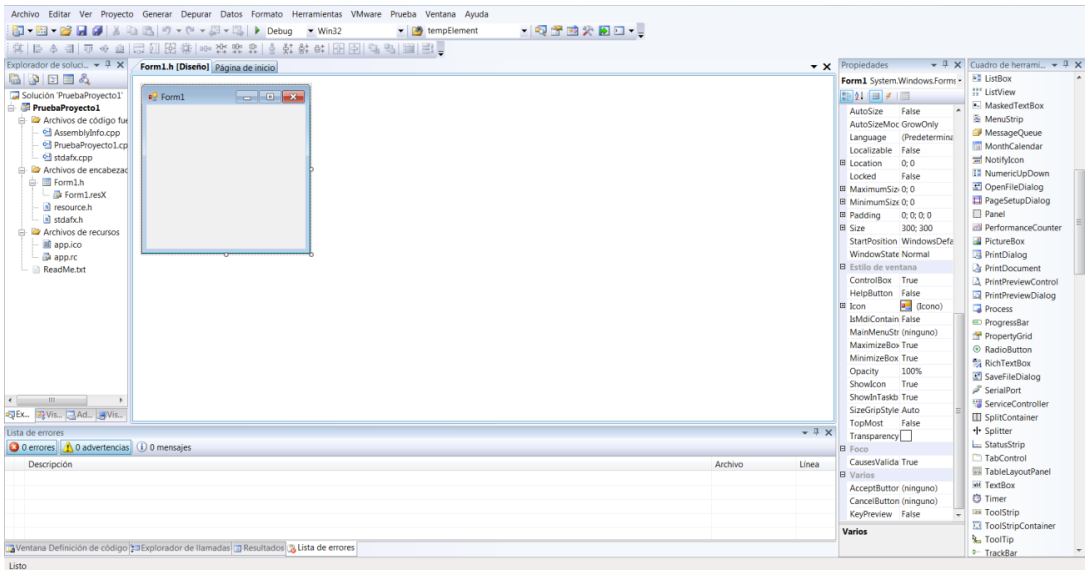
TEMA 15

Abrimos Visual Studio 2008

Crear proyecto: plantilla Windows Forms Applications, nombre del proyecto (el que queramos)

Aspecto del entorno de desarrollo Visual Studio 2008



Código automático al comienzo del programa

```
#pragma once


namespace PruebaProyecto1 {

using namespace System;
using namespace System::ComponentModel;
using namespace System::Collections;
using namespace System::Windows::Forms;
using namespace System::Data;
using namespace System::Drawing;

/// <summary>
/// Resumen de Form1
///
/// ADVERTENCIA: si cambia el nombre de esta clase, deberá cambiar la propiedad
///'Nombre de archivos de recursos' de la herramienta de compilación de recursos
/// administraos asociada con todos los archivos .resx de los que depende esta
/// clase. De lo contrario, los diseñadores no podrán interactuar correctamente
/// con los recursos adaptados asociados con este formulario.
/// </summary>
public ref class Form1 : public System::Windows::Forms::Form
{
   public:
      Form1(void)
      {
         InitializeComponent();
         //
         //TODO: agregar código de constructor aquí
         //
      }
```

```
   protected:
      /// <summary>
      /// Limpiar los recursos que se estén utilizando.
      /// </summary>
      ~Form1()
      {
         if (components)
         {
            delete components;
         }
      }

   private:
      /// <summary>
      /// Variable del diseñador requerida.
      /// </summary>
      System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
      /// <summary>
      /// Método necesario para admitir el Diseñador. No se puede modificar
      /// el contenido del método con el editor de código.
      /// </summary>
      void InitializeComponent(void)
      {
         this->components = gcnew System::ComponentModel::Container();
         this->Size = System::Drawing::Size(300,300);
         this->Text = L"Form1";
         this->Padding = System::Windows::Forms::Padding(0);
         this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
      }
#pragma endregion
   };
}
```

Form1.h Diseño -> Propiedades -> Grupo Layout (Diseño) -> Size -> 500, 350 -> Herramientas -> MenuStrip -> Insertar elementos estándar

Al MenuStrip le añadimos los desplegables E&lement con Line, Rectangle, Circle y Curve, y a &Color con Black, Red, Green y Blue dentro, entre Herramientas y Ayuda

Botón derecho en Line y en Black para señalar Checked

Propiedades -> ShortcutKey por ejemplo en Line puede ser Ctrl+Shift+L (cuidado con no repetir con otras que ya existen por defecto)

Manejador de Eventos para el menú Element -> Line -> ventana de Propiedades -> Eventos -> doble click en evento Click

```
private: System::Void lineToolStripMenuItem_Click(System::Object^ sender,
                                                  System::EventArgs^ e)
{
}
```

Igual para los demás Element y Color

Form1.h Código -> añadir después de los using namespace

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

Tipo de elemento y color son variables modales

Vista de Clases -> Agregar -> Agregar variable

Acceso	Tipo de variable	Nombre de variable
private	ElementType	elementType
private	Color	color

Añadimos el constructor

```
Form1(void) : elementType(ElementType::LINE), color(Color::Black)
{
InitializeComponent();
//
//TODO: Add the constructor code here
//
}
```

Ya podemos completar los manejadores de elementos y colores según corresponda

```
elementType = ElementType::LINE;
...
color = Color::Black;
```

Para que cambien los checks en elementos y colores, doble click en el evento de Element y Color 'DropDownOpening'

```
private: System::Void elementToolStripMenuItem_DropDownOpening(System::Object^ sender,
                                                               System::EventArgs^ e)
{
   lineToolStripMenuItem->Checked = elementType == ElementType::LINE;
   rectangleToolStripMenuItem->Checked = elementType == ElementType::RECTANGLE;
   circleToolStripMenuItem->Checked = elementType == ElementType::CIRCLE;
   curveToolStripMenuItem->Checked = elementType == ElementType::CURVE;
}

private: System::Void colorToolStripMenuItem_DropDownOpening(System::Object^ sender,
                                                             System::EventArgs^ e)
{
   blackToolStripMenuItem->Checked = color == Color::Black;
   redToolStripMenuItem->Checked = color == Color::Red;
   greenToolStripMenuItem->Checked = color == Color::Green;
   blueToolStripMenuItem->Checked = color == Color::Blue;
}
```

Añadir ToolBar -> ToolStrip en la ventana de herramientas. Insertar elementos estándar, sólo se dejan Nuevo, Abrir, Guardar e Imprimir con un Separador, lo demás se elimina. Se añaden 4 botones para los elementos, un separador, y otros 4 botones para los colores

Crear Bitmap images para los botones de la barra de herramientas

Vista de Recursos -> click izquierdo en app.rc -> agregar recurso -> Bitmap -> Nuevo

Tamaño	Color	ID	Nombre de archivo	
16x16	24-bit	IDB_LINE	line.bmp	
"	"	IDB_RECTANGLE	rectangle.bmp	
"	"	IDB_CIRCLE	circle.bmp	
"	"	IDB_CURVE	curve.bmp	
"	"	IDB_BLACK	black.bmp	
"	"	IDB_RED	red.bmp	
"	"	IDB_GREEN	green.bmp	
"	"	IDB_BLUE	blue.bmp	

Se añaden los bitmaps a los botones con la propiedad Image

De cada botón en la barra de herramientas, cambiamos (Name) y ToolTipText a `toolStripLineButton` y `Draw lines` respectivamente para la línea, análogamente para lo demás

Cambiar la propiedad ImageTransparentColor de los botones a White

Para no repetir el manejador de los botones de la barra de herramientas con los del menú, por ejemplo para la línea sería Evento -> Click -> y seleccionamos el manejador lineToolStripMenuItem_Click que ya habíamos hecho para el MenuStrip, y así quedan relacionados. Lo mismo con los demás elementos y colores

Para que cambien los checks de las barra de herramientas no se dispone de un evento como DropDownOpening para los MenuStrip, por eso tenemos que crear funciones que realicen esa labor

Vista de Clases -> botón derecho en Form1 -> Agregar -> Agregar función

Tipo de valor devuelto	Tipo de parámetro	Acceso	Nombre de la función
void	void	private	SetElementTypeButtonsState
void	void	private	SetColorButtonsState

```
// Set the state of the element type toolbar buttons
void SetElementTypeButtonsState(void)
{
   toolStripLineButton->Checked = elementType == ElementType::LINE;
   toolStripRectangleButton->Checked = elementType == ElementType::RECTANGLE;
   toolStripCircleButton->Checked = elementType == ElementType::CIRCLE;
   toolStripCurveButton->Checked = elementType == ElementType::CURVE;
}

// Set the state of the color toolbar buttons
void SetColorButtonsState(void)
{
   toolStripBlackButton->Checked = color == Color::Black;
   toolStripRedButton->Checked = color == Color::Red;
   toolStripGreenButton->Checked = color == Color::Green;
   toolStripBlueButton->Checked = color == Color::Blue;
}
```

Añadimos al constructor

```
Form1(void) : elementType(ElementType::LINE), color(Color::Black)
{
    InitializeComponent();
    //
    SetElementTypeButtonsState();
    SetColorButtonsState();
    //
}
```

Actualizamos todos los manejadores de elementos y colores añadiendo SetElementTypeButtonsState(); y SetColorButtonsState(); respectivamente

Aspecto del programa al finalizar el Tema 15