

# Tema Windows Form en C#

## Contenido

1	Introducción .....	1
1.1	Características .....	1
1.2	Representación y manipulación de datos .....	2
1.3	Implementación de aplicaciones en equipos cliente .....	3
2	Creación de aplicaciones Windows Form .....	4
2.1	Controles mas usados .....	19
2.1.1	Ejercicio .....	22
2.2	Manejando datos con LINQ y formularios .....	23

## 1 Introducción

Windows Forms es un marco de interfaz de usuario para compilar aplicaciones de escritorio de Windows. Proporciona una de las formas más productivas de crear aplicaciones de escritorio basadas en el diseñador visual proporcionado en Visual Studio. Funciones como la colocación de controles visuales mediante arrastrar y colocar facilita la compilación de aplicaciones de escritorio.

Con Windows Forms, puede desarrollar aplicaciones enriquecidas gráficamente que son fáciles de implementar, y actualizar, y con las que se puede trabajar sin conexión o mientras están conectadas a Internet. Las aplicaciones de Windows Forms pueden acceder al hardware local y al sistema de archivos del equipo en el que se ejecutan.

Hay dos implementaciones de Windows Forms:

La implementación de código abierto hospedada en GitHub.

1. Esta versión se ejecuta en .NET Core 5 y .NET Core 3.1. El diseñador visual de Windows Forms requiere, como mínimo, Visual Studio 2019 versión 16.8 Preview.
2. La implementación de .NET Framework 4 compatible con Visual Studio 2019 y Visual Studio 2017..NET Framework 4 es una versión de .NET solo para Windows y se considera un componente del sistema operativo Windows. Esta versión de Windows Forms se distribuye con .NET Framework.

### 1.1 Características

Windows Forms es una tecnología de interfaz de usuario para .NET, un conjunto de bibliotecas administradas que simplifican las tareas comunes de las aplicaciones, como leer y escribir en el sistema de archivos. Cuando se usa un entorno de desarrollo como Visual

Studio, se pueden crear aplicaciones cliente inteligentes de Windows Forms que muestren información, soliciten la entrada a los usuarios y se comuniquen con equipos remotos a través de una red.

En Windows Forms, un formulario es una superficie visual en la que se muestra información al usuario. Normalmente, las aplicaciones de Windows Forms se compilan mediante la adición de controles a formularios y el desarrollo de respuestas a las acciones del usuario, como clics del ratón o pulsaciones de teclas. Un control es un elemento de interfaz de usuario (IU) discreto que muestra datos o acepta la entrada de datos.

Cuando un usuario realiza una acción en un formulario o en uno de sus controles, la acción genera un evento. La aplicación reacciona a estos eventos mediante código y procesa los eventos cuando se producen.

Windows Forms contiene diversos controles que puede agregar a los formularios: controles que muestran cuadros de texto, botones, cuadros desplegables, botones de radio e incluso páginas web. Si un control existente no satisface las necesidades, Windows Forms también permite crear controles personalizados mediante la clase UserControl.

Windows Forms tiene controles de interfaz de usuario enriquecidos que emulan las características de aplicaciones de tecnología avanzada como Microsoft Office. Los controles ToolStrip y MenuStrip permiten crear barras de herramientas y menús que contienen texto e imágenes, muestran submenús y hospedan otros controles como cuadros de texto y cuadros combinados.

Con el Diseñador de Windows Forms de arrastrar y colocar de Visual Studio, puede crear fácilmente aplicaciones de Windows Forms. Simplemente seleccione los controles con el cursor y colóquelos donde quiera en el formulario. El diseñador proporciona herramientas como líneas de cuadrícula y líneas de ajuste para minimizar la molestia de alinear los controles. Puede usar los controles FlowLayoutPanel, TableLayoutPanel y SplitContainer para crear diseños de formularios avanzados en menos tiempo.

Por último, si debe crear sus propios elementos de interfaz de usuario personalizados, el espacio de nombres System.Drawing contiene una gran selección de clases para representar líneas, círculos y otras formas directamente en un formulario.

## **1.2 Representación y manipulación de datos**

Muchas aplicaciones tienen que mostrar datos procedentes de una base de datos, un archivo XML o JSON, un servicio web u otro origen de datos. Windows Forms proporciona un control flexible denominado control DataGridView para mostrar esa información tabulada en un formato tradicional de filas y columnas, de modo que cada dato ocupe su propia celda. Al usar DataGridView, puede personalizar la apariencia de celdas individuales, bloquear en su posición filas y columnas arbitrarias y mostrar controles complejos dentro de las celdas, entre otras características.

La conexión a orígenes de datos a través de una red es una tarea sencilla con Windows Forms. El componente BindingSource representa una conexión a un origen de datos y expone métodos para enlazar datos a controles, desplazarse a los registros anteriores y siguientes,

modificar registros y guardar los cambios en el origen. El control BindingNavigator proporciona una interfaz sencilla en el componente BindingSource para que los usuarios se desplacen por los registros.

Puede crear fácilmente controles enlazados a datos mediante la ventana Orígenes de datos de Visual Studio. En la ventana se muestran los orígenes de datos como bases de datos, servicios web y objetos del proyecto. Para crear controles enlazados a datos, arrastre los elementos desde esta ventana hasta los formularios de su proyecto. También puede enlazar controles existentes a datos si arrastra los objetos desde la ventana Orígenes de datos a los controles existentes.

Otro tipo de enlace de datos que puede administrar en Windows Forms es el de configuración. La mayoría de las aplicaciones deben conservar cierta información sobre su estado de tiempo de ejecución, como el último tamaño conocido de los formularios, y conservar los datos de preferencias del usuario, como las ubicaciones predeterminadas de los archivos guardados. La característica Configuración de la aplicación aborda estos requisitos al proporcionar una manera sencilla de almacenar ambos tipos de configuración en el equipo cliente. Después de definir esta configuración mediante Visual Studio o un editor de código, se conserva como XML y se vuelve a leer automáticamente en memoria en tiempo de ejecución.

También se usa LINQ como origen o intermediario de datos entre aplicaciones Form y bases de datos o XML. Es lo que haremos en estos apuntes.

### **1.3 Implementación de aplicaciones en equipos cliente**

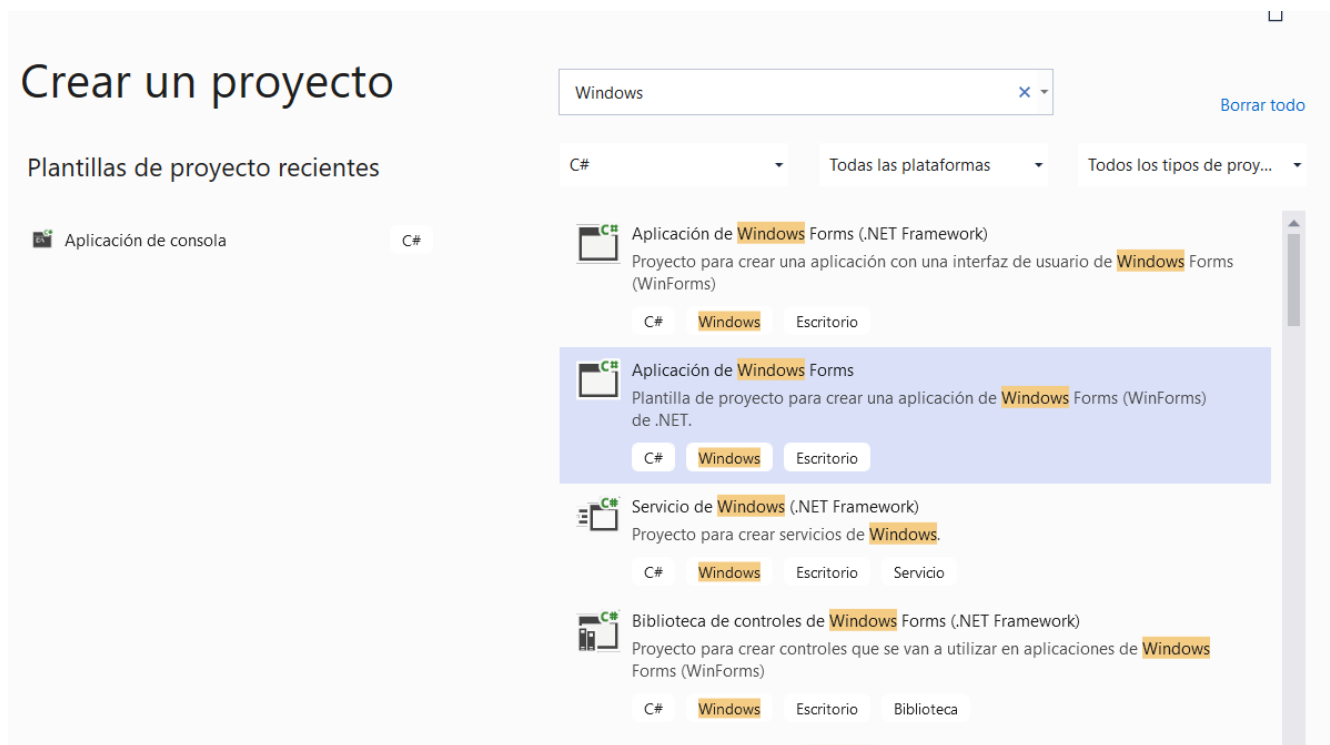
Una vez que haya escrito la aplicación, tendrá que enviarla a los usuarios para que puedan instalarla y ejecutarla en sus equipos cliente. Cuando se usa la tecnología ClickOnce, puede implementar las aplicaciones desde Visual Studio con solo unos cuantos clics y proporcionar a los usuarios una dirección URL que apunte a la aplicación en la web. ClickOnce administra todos los elementos y dependencias de la aplicación y garantiza que se instale correctamente en el equipo cliente.

Las aplicaciones ClickOnce se pueden configurar para que solo se ejecuten cuando el usuario está conectado a la red, o bien para ejecutarse tanto en línea como sin conexión. Al especificar que una aplicación debe admitir el funcionamiento sin conexión, ClickOnce agrega un vínculo a la aplicación en el menú Inicio del usuario. Después, el usuario puede abrir la aplicación sin usar la dirección URL.

Cuando se actualiza la aplicación, se publica un nuevo manifiesto de implementación y una nueva copia de la aplicación en el servidor web. ClickOnce detectará que hay una actualización disponible y actualizará la instalación del usuario. No se requiere ninguna programación personalizada para actualizar las aplicaciones antiguas.

## 2 Creación de aplicaciones Windows Form

Lo primero que vamos a hacer es crear un proyecto nuevo Windows Form en Visual Studio 2019. Abrimos nuestro Visual Studio 2019, creamos un nuevo proyecto y elegimos la opción Windows Form para .NET



Le llamamos PrimeraAplicación, pulsamos en Siguiente.

# Configure su nuevo proyecto

Aplicación de Windows Forms

C#

Windows

Escritorio

Nombre del proyecto

PrimeraAplicacion

Ubicación

C:\Users\carlo\OneDrive\Brianda20202021\interfaces\NET\Tema1\Proyectos\Tema3

...

Solución

Crear nueva solución

Nombre de la solución ⓘ

PrimeraAplicacion

☐

Colocar la solución y el proyecto en el mismo directorio

Elegimos Net 5.0 y pulsamos en el botón de Crear. Lo primero que nos aparecerá es la vista de diseño de la ventana principal (Form1) de nuestra aplicación.

# Información adicional

Aplicación de Windows Forms   C#   Windows   Escritorio

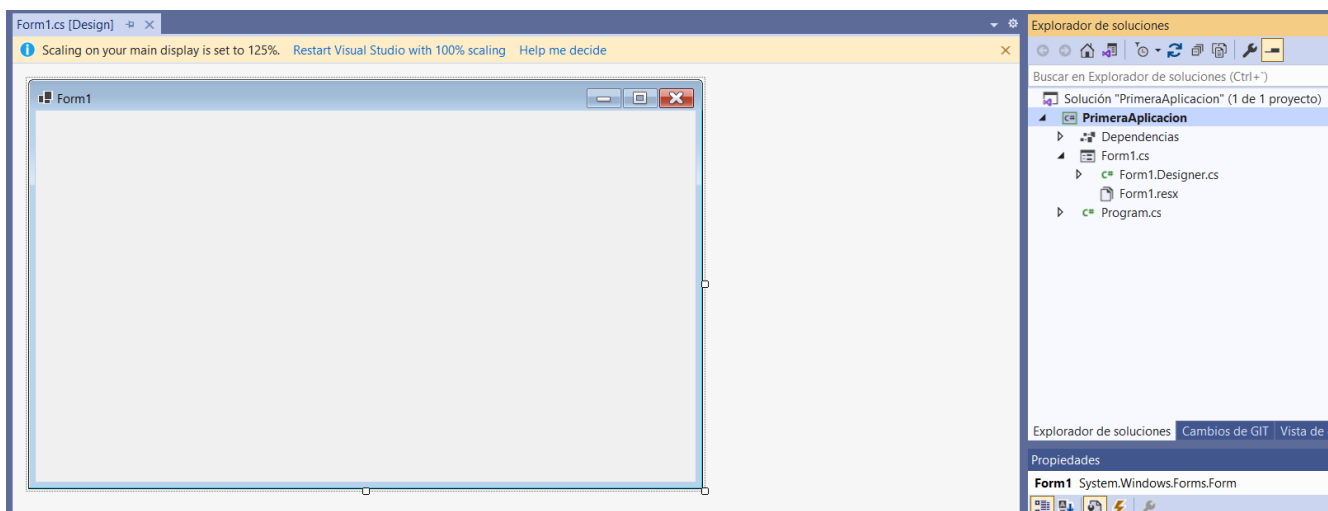
Plataforma de destino ⓘ

.NET 5.0 (Actual)

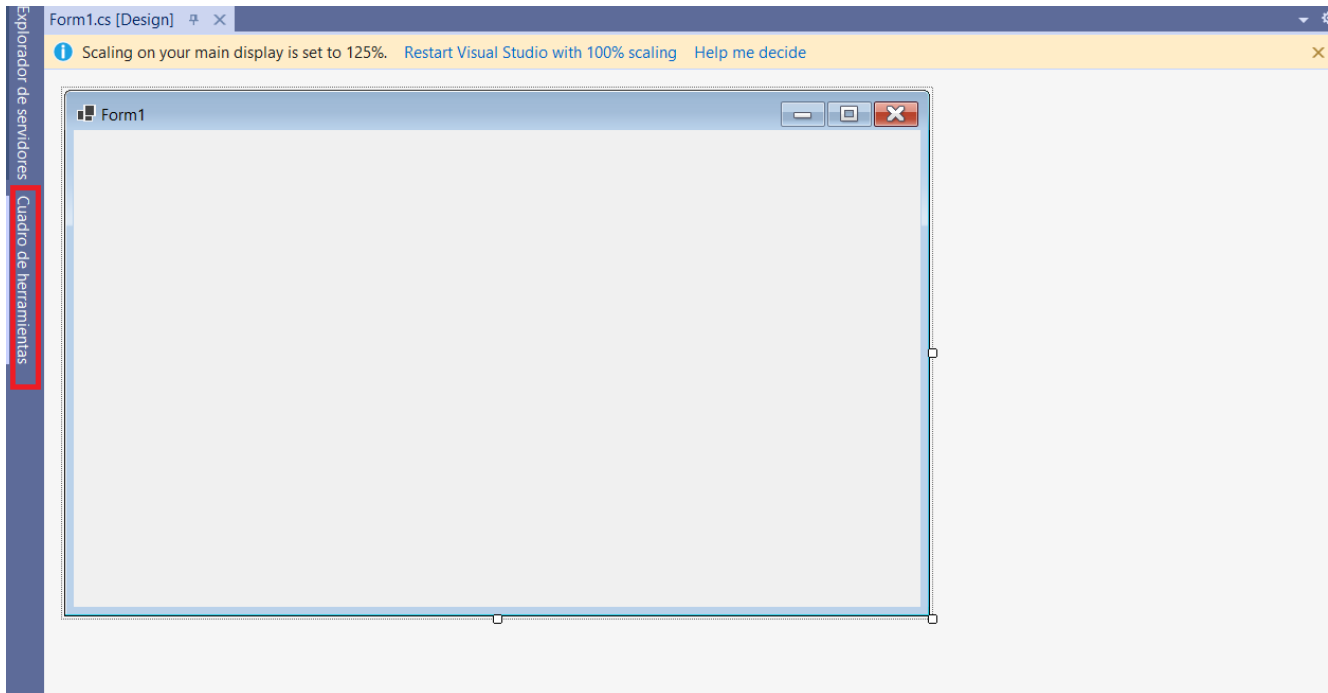
Atrás

Crear

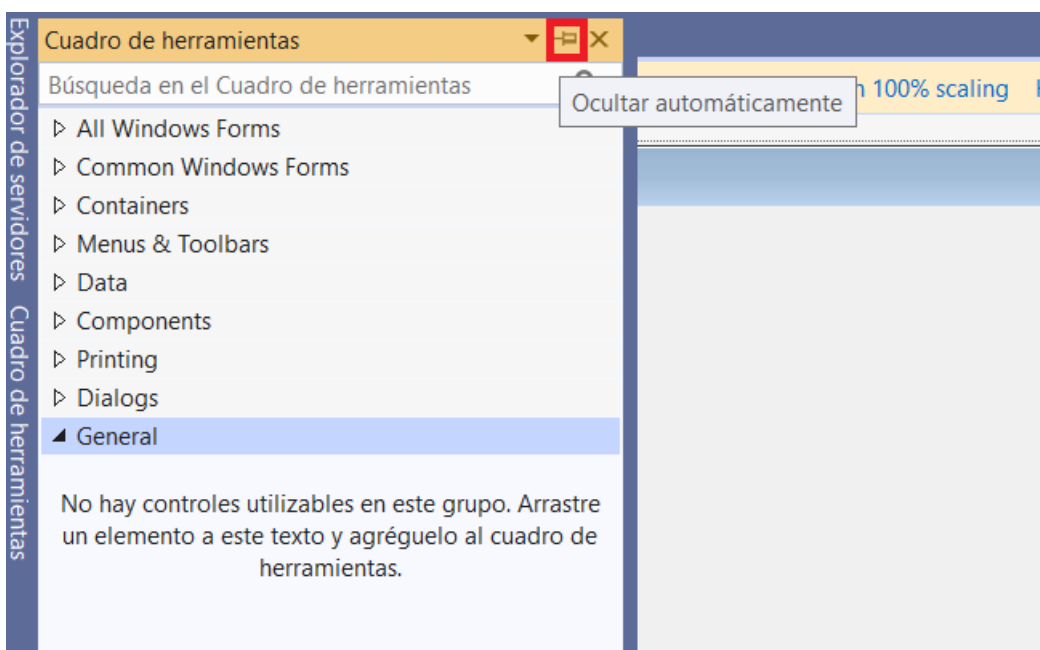
Como veis el Form1 es una clase C# con un fichero `cs` Form1.Designer.cs, y además un fichero de recursos Form1.resx.

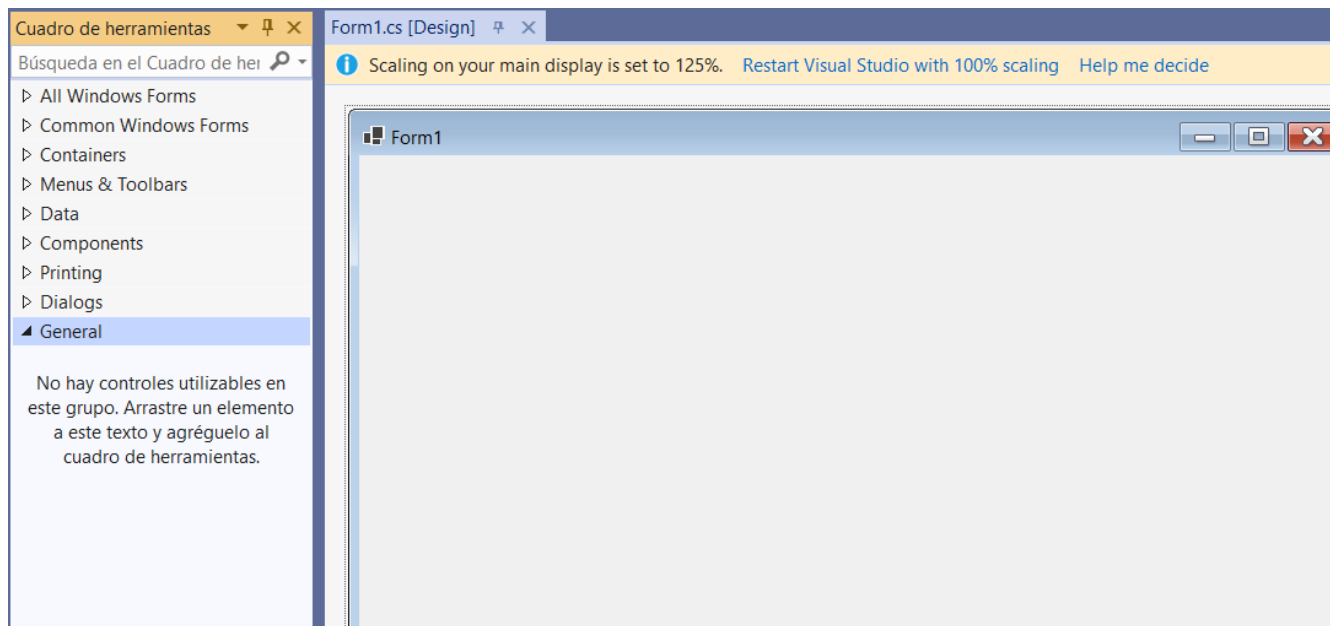


1. Si no ve la opción flotante **Cuadro de herramientas**, puede abrirla desde la barra de menús. Para ello, seleccione **Ver > Cuadro de herramientas**. También puedes presionar **CTRL+Alt+X**.



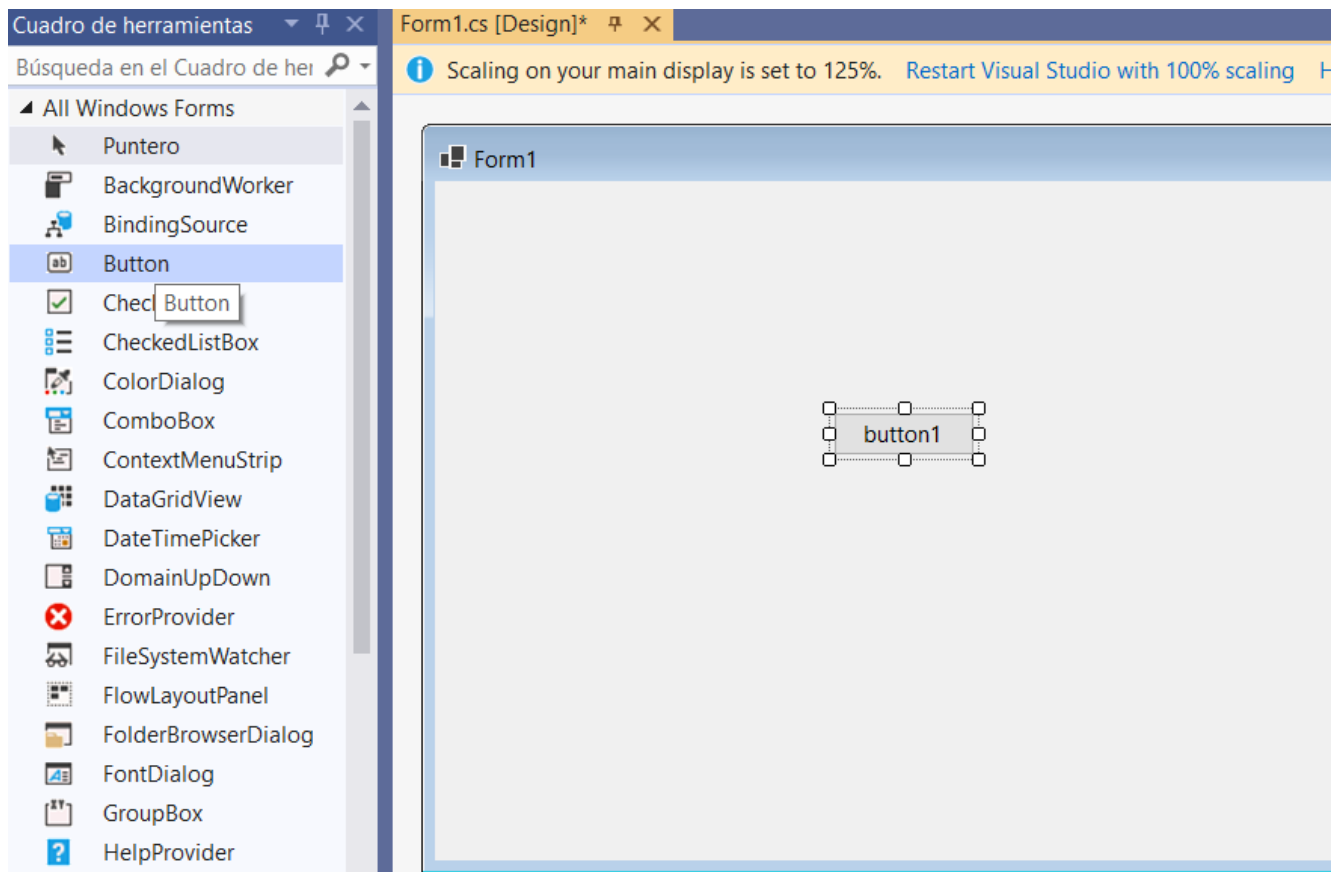
2. Elija el icono **Anclar** para acoplar la ventana **Cuadro de herramientas**. Para ello pulsad en la chincheta, de esta manera veréis el formulario y el cuadro de herramientas a la vez, muy útil para diseñar los interfaces.



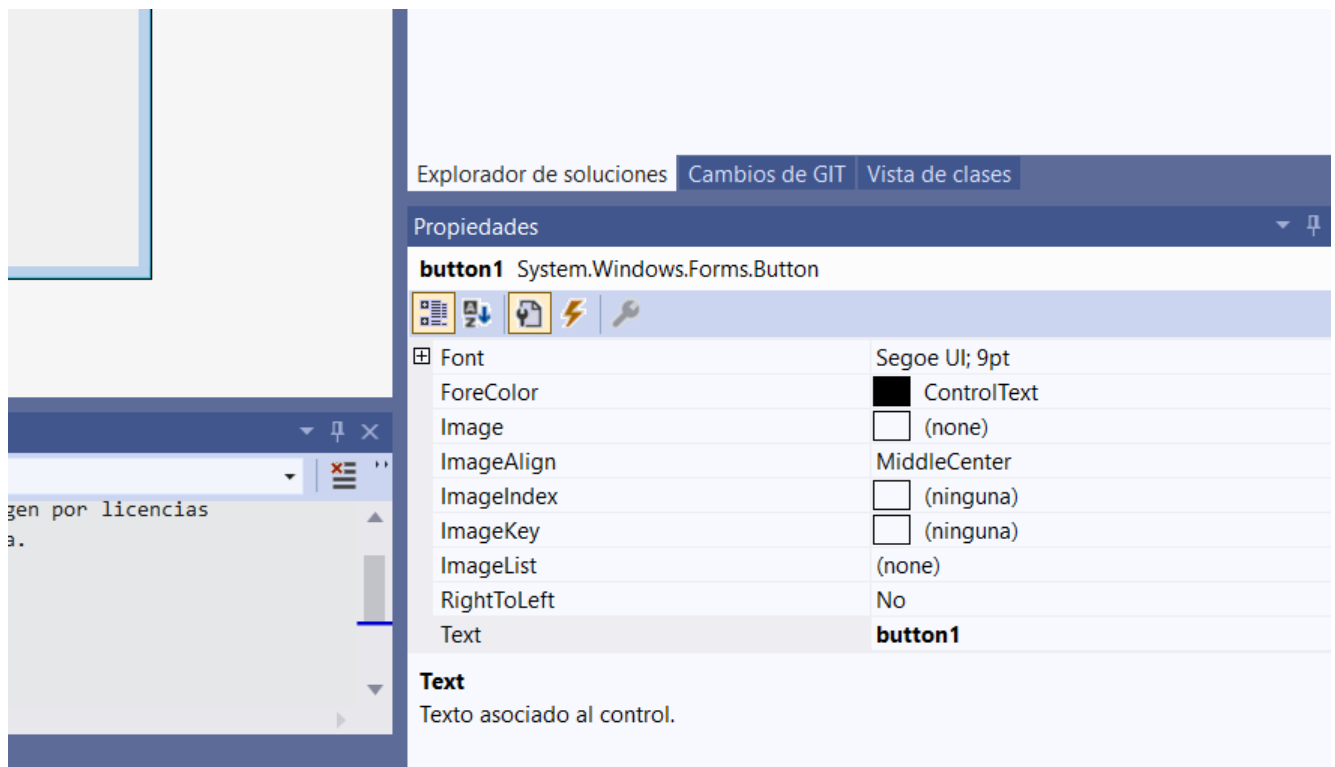


Desplegad el Menu All Windows Form del cuadro de herramientas y arrastar el componente Button hacia el formulario.





Si seleccionais el formulario o el botón tenéis la ventana de propiedades en la esquina inferior derecha. Modificar la propiedad Text y escribir Ok.

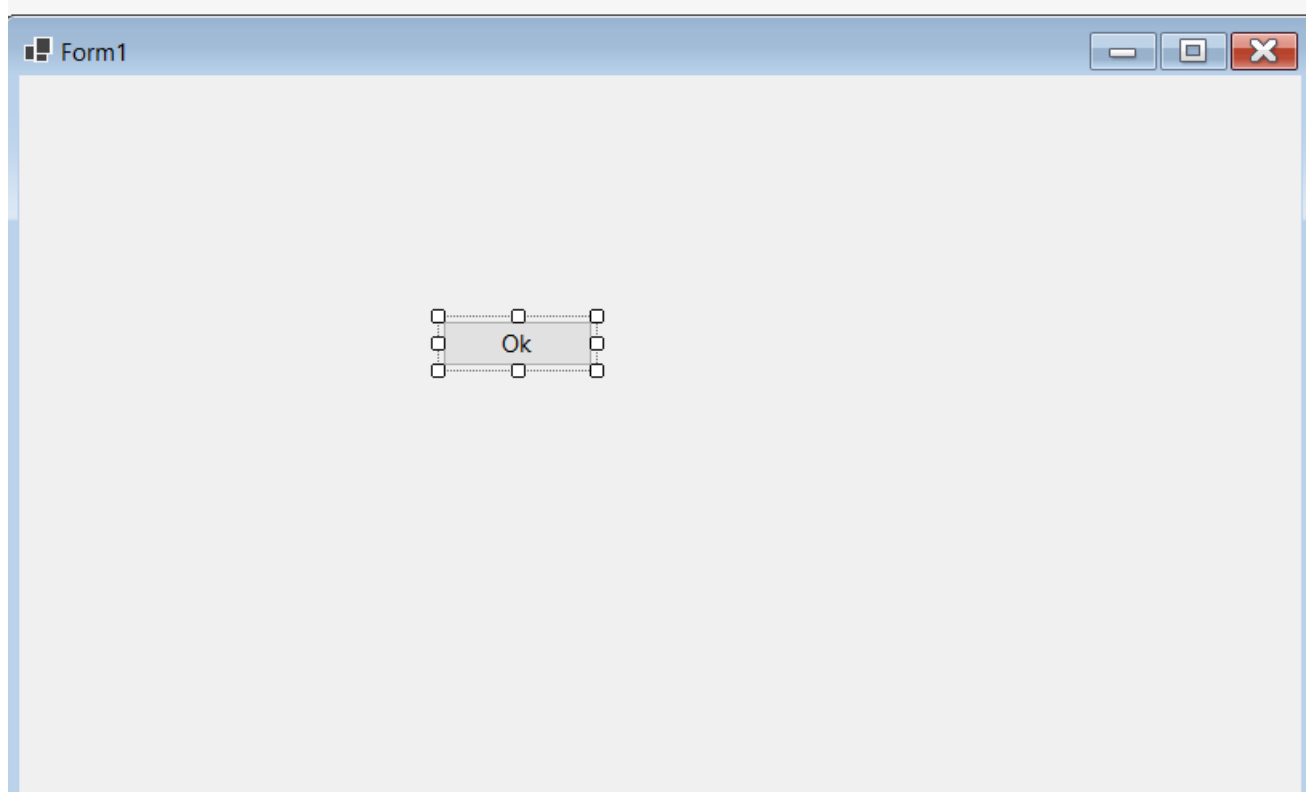


## Personalizando los controles (Propiedades)

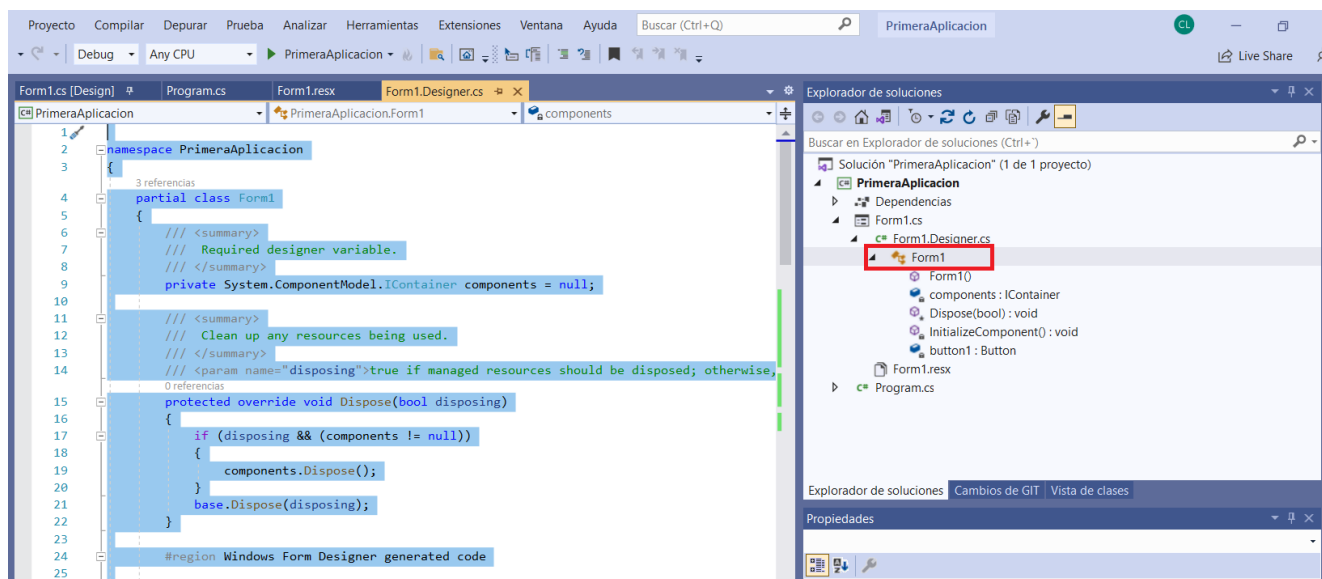
- Una propiedad es una característica de un objeto (color, tamaño, tipo de letra, etc.).
- La ventana de “Propiedades” muestra todas las propiedades del objeto que se encuentra seleccionado con el mouse, y permite modificar sus valores.



Os cambiará el texto del botón dentro del formulario. Podemos modificar muchas propiedades desde el diseñador.



Pero los cambios cuando añadimos componentes tienen su efecto en código igualmente en el Form1\_Designer.cs. Vamos a abrir el código para FormDesigner. Haced doble click sobre Form1 en el explorador de aplicaciones:



Vereis el código completo. A destacar tenemos unas cuantas propiedades y métodos.

## Propiedades

El contenedor donde iran situados todos los componentes, botones, etiquetas ,etc.

```
/// </summary>  
private System.ComponentModel.IContainer components = null;
```

Al final del Código que se genera automáticamente, los componentes, en este caso un botón que hemos añadido.

```
#endregion  
  
private System.Windows.Forms.Button button1;
```

Métodos.

El método Dispose elimina de memoria el formulario. Para ello primero libera de memoria el contenedor con todos sus componentes.

```
components.Dispose();
```

Y luego el formulario llamando al método Dispose de la clase Padre

```
base.Dispose(disposing);  
  
protected override void Dispose(bool disposing)  
{  
    if (disposing && (components != null))  
    {  
        components.Dispose();  
    }  
    base.Dispose(disposing);  
}
```

El método Initialize componentes, que inicializa el formulario y todos sus componentes. Como veis el cambio que hemos realizado en la ventana propiedades se refleja en código.

El método **SuspendLayout** suspende temporalmente el visualizar el formulario y nos permite añadir componentes y modificar propiedades. Cuando todos los componentes y propiedades han sido añadidos llamamos al metodo **ResumeLayout** para visualizar el formulario.

```
private void InitializeComponent()  
{  
    this.button1 = new System.Windows.Forms.Button();  
    this.SuspendLayout();  
    //  
    // button1
```

```

//
this.button1.Location = new System.Drawing.Point(264, 153);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(94, 29);
this.button1.TabIndex = 0;
this.button1.Text = "Ok";
this.button1.UseVisualStyleBackColor = true;
//
// Form1
//
this.AutoScaleDimensions = new System.Drawing.SizeF(8F, 20F);
this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
this.ClientSize = new System.Drawing.Size(800, 450);
this.Controls.Add(this.button1);
this.Name = "Form1";
this.Text = "Form1";
this.ResumeLayout(false);
}

```

```

namespace PrimeraAplicacion
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed; otherwise,
        false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.button1 = new System.Windows.Forms.Button();
            this.SuspendLayout();
            //
            // button1
            //
            this.button1.Location = new System.Drawing.Point(264, 153);
            this.button1.Name = "button1";

```

```

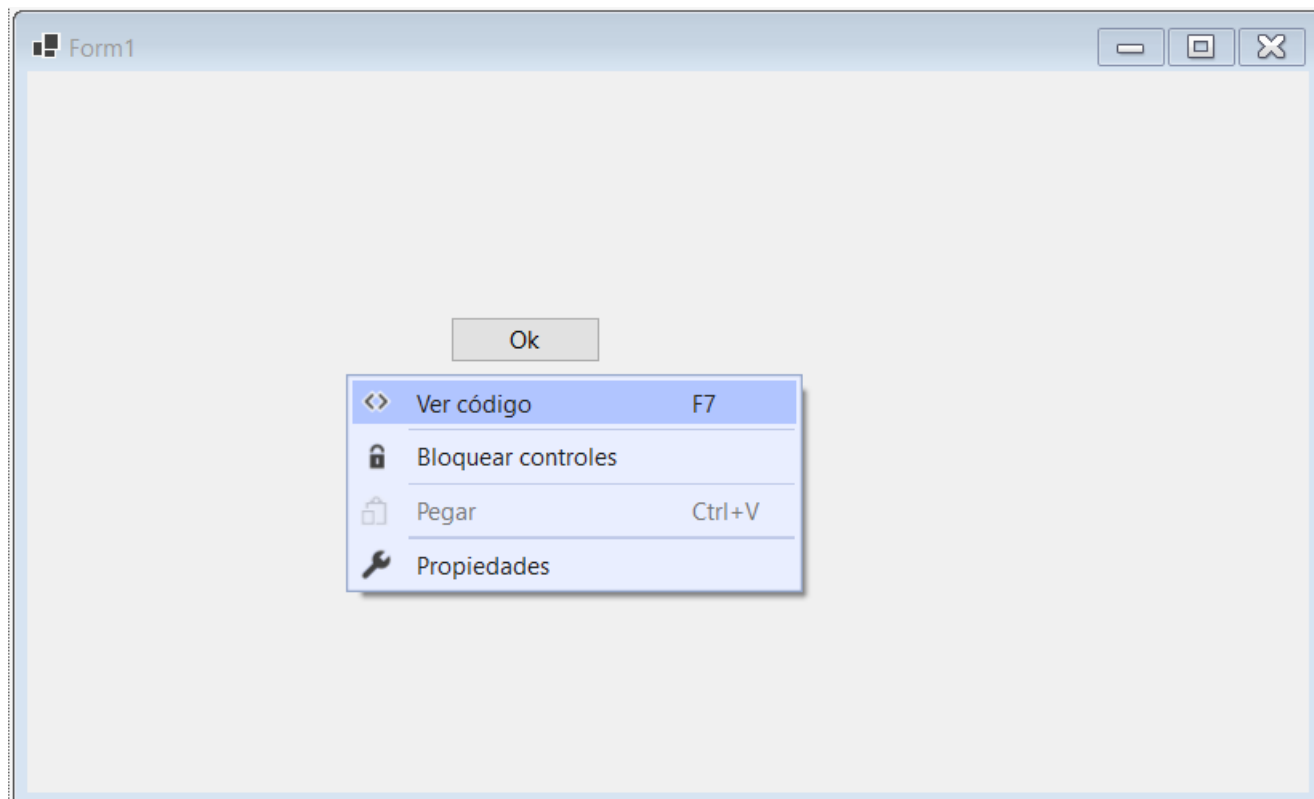
        this.button1.Size = new System.Drawing.Size(94, 29);
        this.button1.TabIndex = 0;
        this.button1.Text = "Ok";
        this.button1.UseVisualStyleBackColor = true;
        //
        // Form1
        //
        this.AutoScaleMode = new System.Drawing.SizeF(8F, 20F);
        this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
        this.ClientSize = new System.Drawing.Size(800, 450);
        this.Controls.Add(this.button1);
        this.Name = "Form1";
        this.Text = "Form1";
        this.ResumeLayout(false);
    }

#endregion

private System.Windows.Forms.Button button1;
}
}

```

El código de manejo del formulario y sus eventos está sin embargo en Form1.cs. Para ver el código del formulario debéis realizar la siguiente acción. Si haces botón derecho del ratón sobre el formulario os aparecerá un menú contextual.



Fijaos como el Formulario Form1 en su constructor llama a InitializeComponent definido en Form1\_Designer.cs. Podéis ver como dos zonas separadas de código, una

```
using System;
```

```

using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace PrimeraAplicacion
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}

```

Ahora vamos a ver el código del programa principal Program.cs

La anotación atributo **STAThread** indica que el modelo de hilos COM para la aplicación es de un solo hilo. Este atributo debe estar presente en el punto de entrada de cualquier aplicación que utilice Formularios Windows Forms; si se omite, los componentes de Windows podrían no funcionar correctamente. Si el atributo no está presente, la aplicación usa el modelo de multihilo, que no se admite para Windows Forms.

Como veis tenemos como punto de entrada a la aplicación la función Main que establece tres parámetros antes de lanzar la ventana. Se usa la clase Application para controlar toda la configuración relativa a la Aplicación.

Application Proporciona métodos y propiedades `static` para administrar una aplicación, como métodos para iniciar y detener una aplicación o para procesar mensajes de Windows, y propiedades para obtener información sobre una aplicación. Esta clase no puede heredarse, lleva el modificador `sealed`.

```
public sealed class Application
```

El primero indica que pasamos a alta definición de la pantalla, Establece el modo con valores altos de PPP (puntos por pulgada) para el hilo de la Aplicación..

```
Application.SetHighDpiMode(HighDpiMode.SystemAware);
```

El Segundo habilita los estilos visuales en la aplicación.

```
Application.EnableVisualStyles();
```

El tercero establece que clase se usará para renderizar el texto de nuestros formularios. El

valor predeterminado que se va a utilizar con los nuevos controles tiene dos posibilidades. Si true es, los nuevos controles que admiten UseCompatibleTextRendering usan la clase basada en GDI+ Graphics para la representación de texto; si es false , los nuevos controles usan la clase basada en GDI TextRenderer .

```
Application.SetCompatibleTextRenderingDefault(false);
```

Y para finalizar Main carga el formulario Form1. El método run cuando recibe un Formulario , Inicia la ejecución de un bucle de mensajes de aplicación estándar en el hilo actual y hace que el formulario especificado esté visible. Esto quiere decir que se inicia en el hilo actual un bucle de manera que la aplicación este abierta hasta que se produzca un evento como pulsar el botón salir X, del formulario, en que se saldrá del bucle y se terminará la aplicación. Cuando se produce un error también se sale del bucle y se termina la aplicación.

```
Application.Run(new Form1());
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

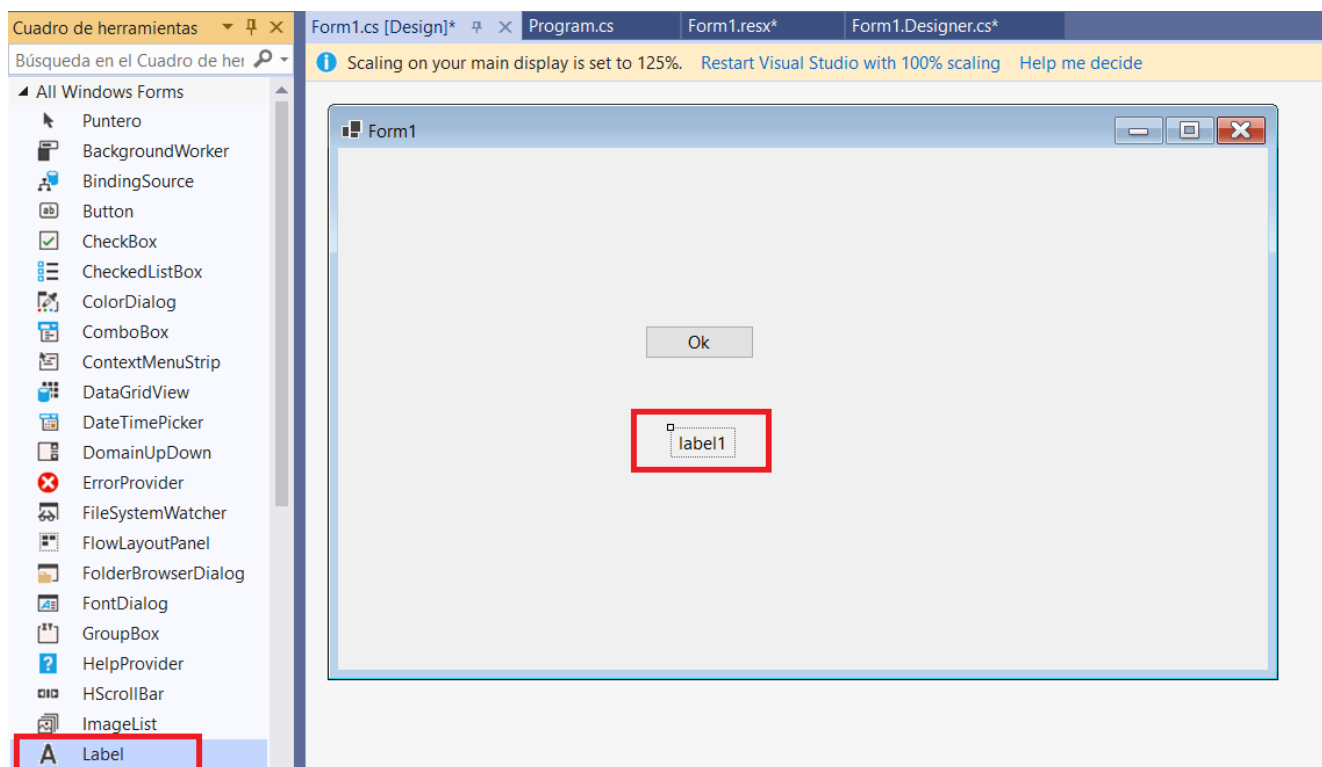
namespace PrimeraAplicacion
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.SetHighDpiMode(HighDpiMode.SystemAware);
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Iremos introduciendo propiedades y métodos de la clase Application sobre demanda, es decir conforme los vayamos usando. En este enlace tenéis toda la información sobre la clase Application

<https://docs.microsoft.com/es-es/dotnet/api/system.windows.forms.application?view=net-5.0>

Añadimos a nuestro formulario una etiqueta como paso previo a añadir líneas de código,



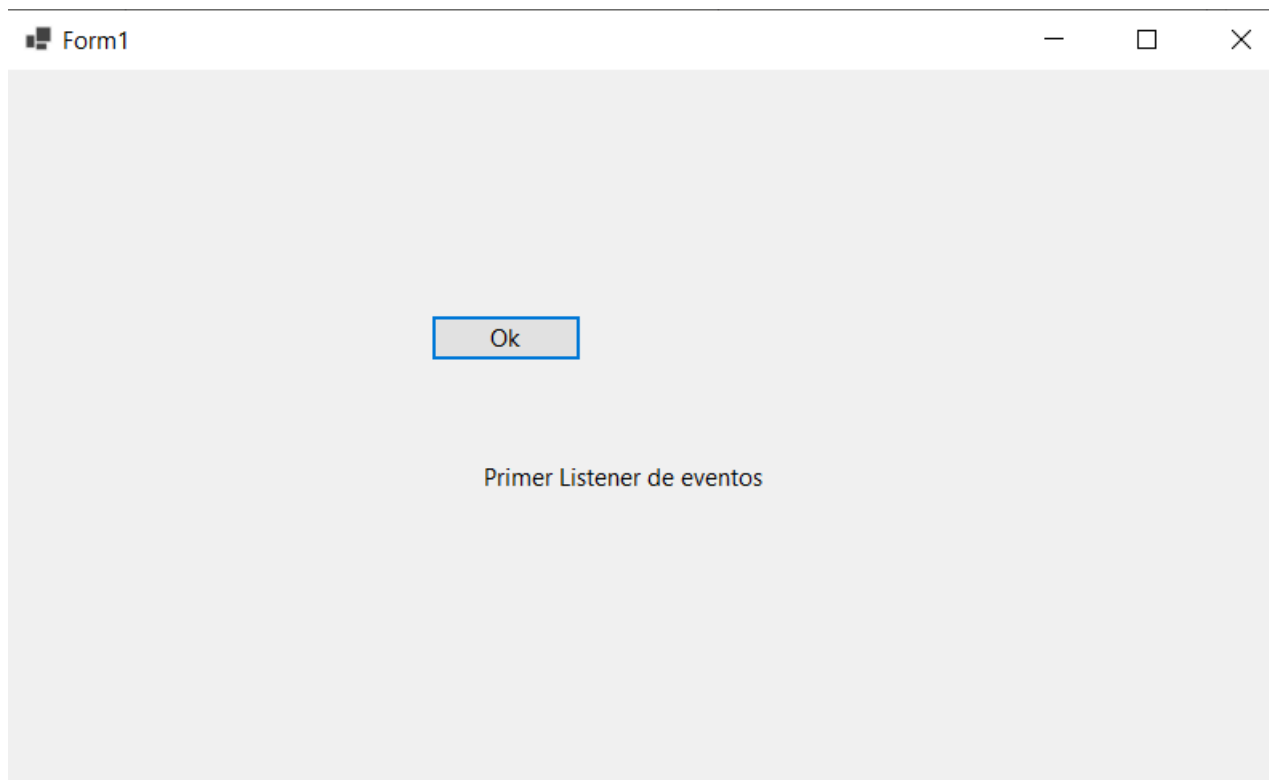


Si haces doble click sobre el botón de Ok, automáticamente se añadirá un método `button1_Click` para gestionar el Evento `onClick` sobre el botón. Añidid el código marcado en amarillo

```
private void button1_Click(object sender, EventArgs e)
{
    label1.Text = " Primer Listener de eventos";
}
```

Donde `sender` es el objeto, componente que ha producido el evento, y `EventArgs e` es un objeto que tiene valores asociados al evento.

Ejecutamos la aplicación. Al hacer click sobre el botón se ejecutará el código en el interior del método

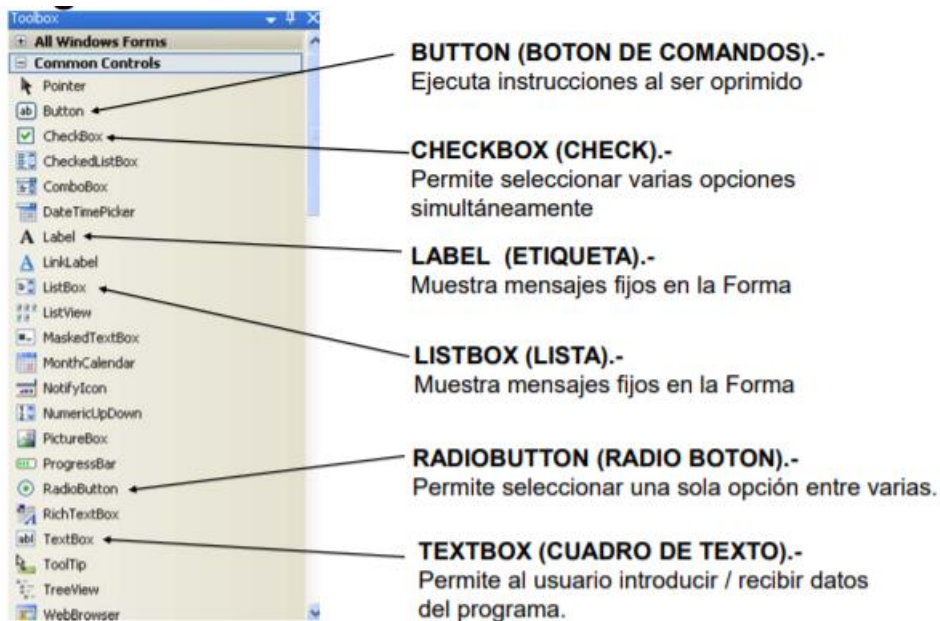


Si abrimos el código de Form1\_Designer.cs nos encontraremos que el componente botón tiene cambios en el método InitializeComponent. Fijaos como a la propiedad de button1.Click se le asigna un Manejador de eventos EventHandler que recibe como parámetro, un delegado, una función que es precisamente el método OnClick que hemos definido antes `private void button1_Click(object sender, EventArgs e)`

```
private void InitializeComponent()
{
    this.button1 = new System.Windows.Forms.Button();
    this.label1 = new System.Windows.Forms.Label();
    this.SuspendLayout();
    //
    // button1
    //
    this.button1.Location = new System.Drawing.Point(264, 153);
    this.button1.Name = "button1";
    this.button1.Size = new System.Drawing.Size(94, 29);
    this.button1.TabIndex = 0;
    this.button1.Text = "Ok";
    this.button1.UseVisualStyleBackColor = true;
    this.button1.Click += new System.EventHandler(this.button1_Click);
    //
}
```

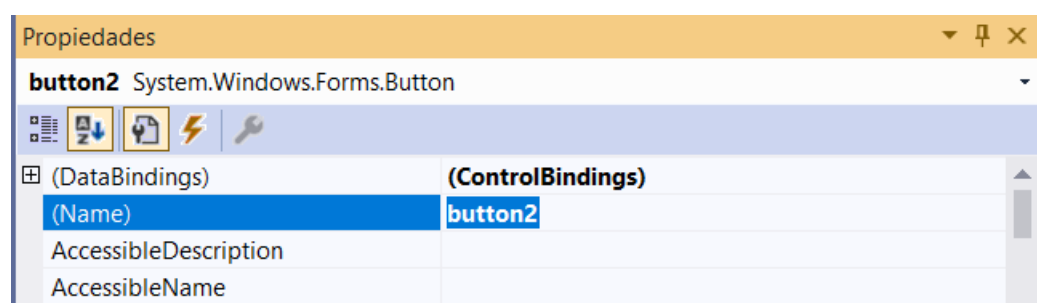
## 2.1 Controles mas usados

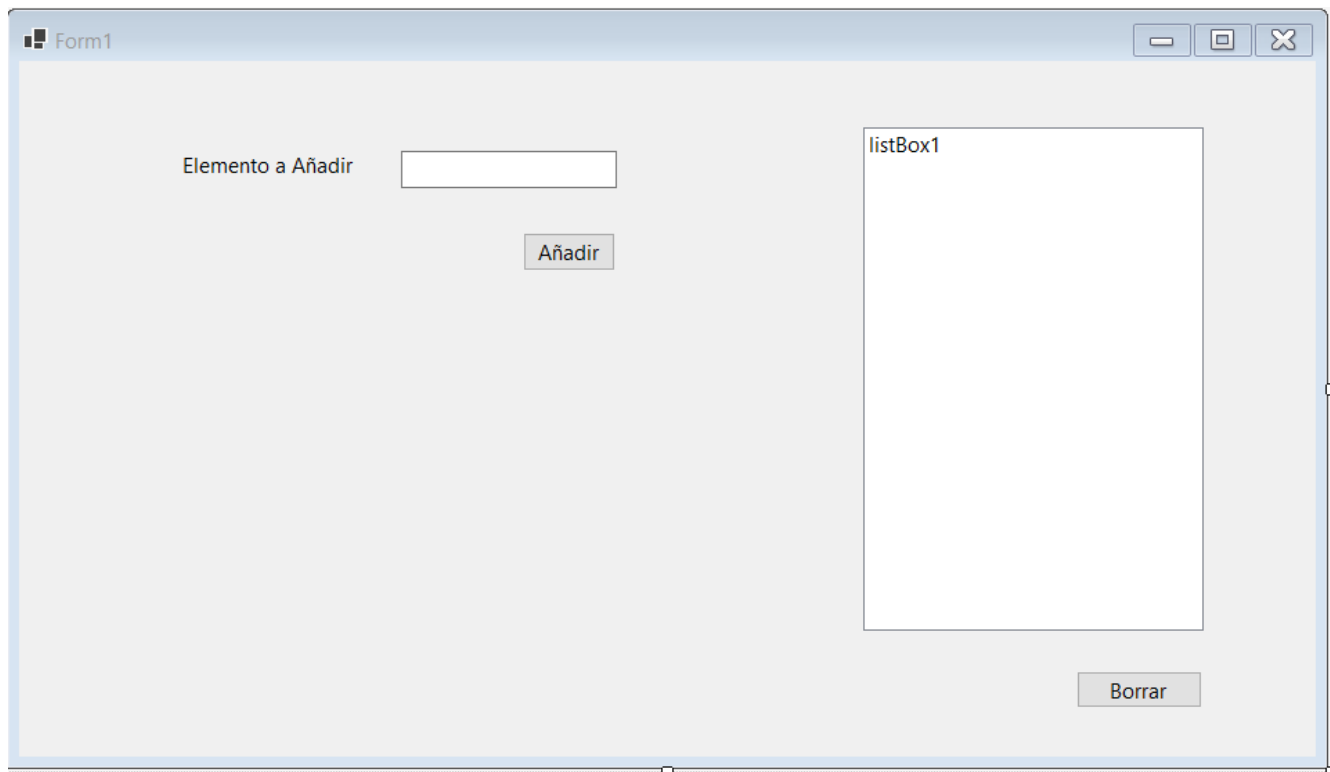
En la siguiente imagen tenéis una breve descripción de los controles mas usados. Vamos a desarrollar una pequeña aplicación con estos controles.



Creamos un nuevo proyecto AplicacionFormControles. Vais a crear el siguiente diseño el el formulario. A la derecha tenéis un ListBox. Lo que vamos a hacer es añadir al ListBox cada elemento que escribamos en código siempre que el TextBox no este vacio cuando pulsemos el botón añadir. El botón borrar eliminará el elemento del Listbox.

Para cambiar el nombre al control, usamos la propiedad Name. Está cambiará el nombre dela propiedad de la clase Form que almacena el control en código.





Para añadir elementos al ListBox añadimos un Listener para el botón Añadir

Comprobamos que el Texto del TextBox no es vacío ni caracteres blancos con el Trim

```
if (txtAnadir.Text.Trim().Length>0)
```

Añadimos a la lista de Items de listBox1 accediendo a la propiedad Items el texto que hay en el textBox.

```
listElementos.Items.Add(txtAnadir.Text);
```

Si el campo de texto está vacío mandamos un mensaje indicando el error. En este caso con usando la clase MessageBox

Los tipos de botones son Ok

```
MessageBoxButtons buttons = MessageBoxButtons.OK;
```

Le damos un mensaje y un título al diálogo de error.

```
string Mensaje = "El campo de texto para añadir está vacío";  
string Titulo = "Texto vacío";  
  
DialogResult result;
```

Mostramos el dialogo de error.

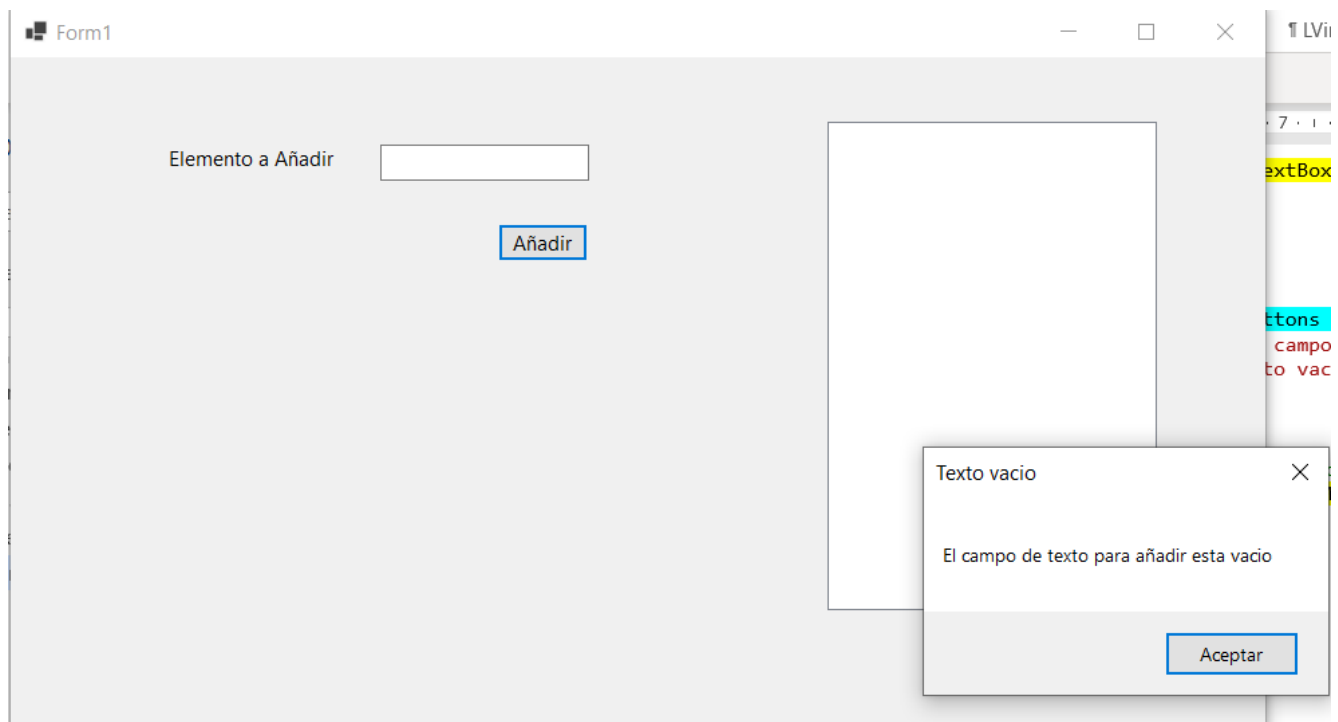
```
// Displays the MessageBox.
result = MessageBox.Show(Mensaje, Titulo, buttons);

private void button1_Click(object sender, EventArgs e)
{
    if (txtAnadir.Text.Trim().Length>0)
    {
        listElementos.Items.Add(txtAnadir.Text);
    } else
    {
        MessageBoxButtons buttons = MessageBoxButtons.OK;
        string Mensaje = "El campo de texto para añadir esta vacio";
        string Titulo = "Texto vacio";

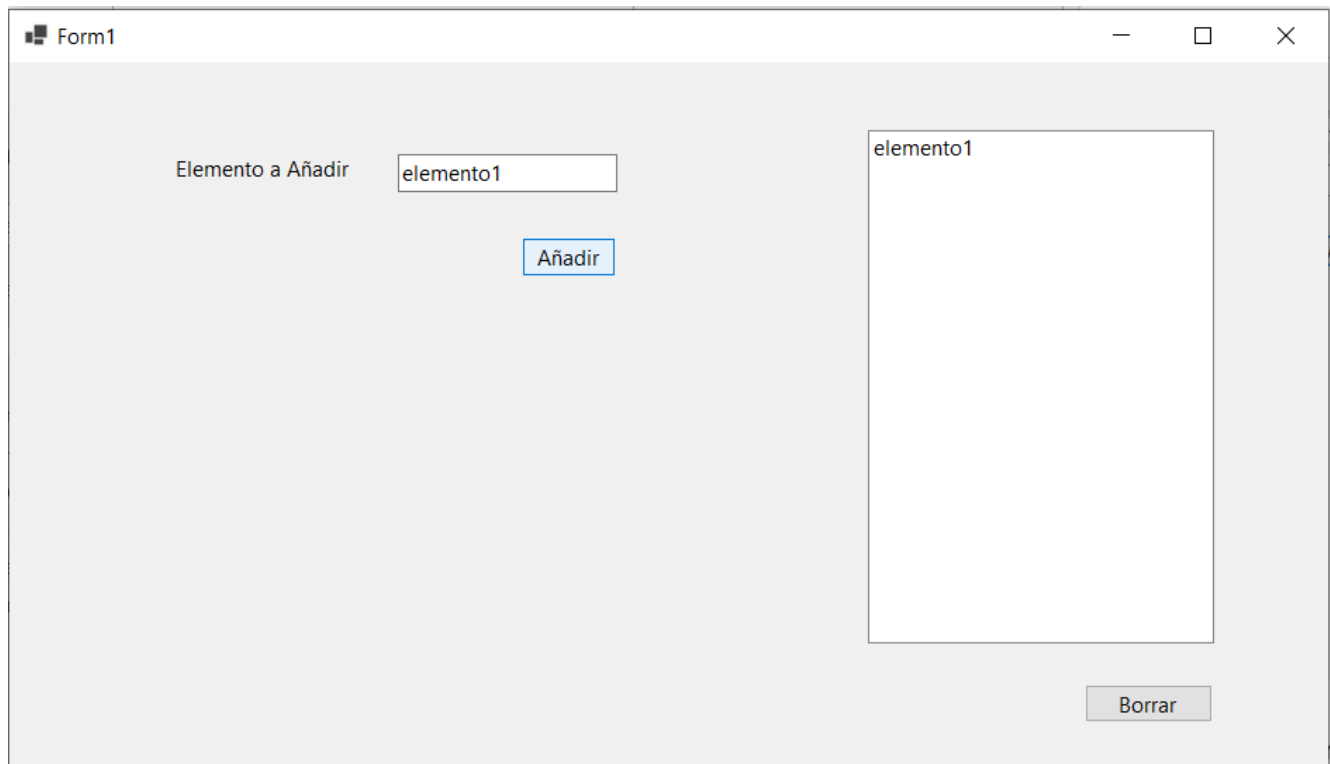
        DialogResult result;

        // Displays the MessageBox.
        result = MessageBox.Show(Mensaje, Titulo, buttons);
    }
}
```

Con la caja de texto vacia mandamos el mensaje de error.



Con la caja de texto llena, insertamos un elemento en la listBox.



Si veis el código del designer

Hemos cambiado el nombre de los controles con la propiedad Name se refleja en código. Es aconsejable usar esta notación:

Empezamos con lbl para Labels.  
Empezamos con btn para Button  
Para listas list o lst.  
Para radiobutton rdb o rdbtn

```
#endregion
```

```
private System.Windows.Forms.TextBox txtAnadir;  
private System.Windows.Forms.Label lblAnadir;  
private System.Windows.Forms.Button btnAnadir;  
private System.Windows.Forms.ListBox listElementos;  
private System.Windows.Forms.Button btnBorrar;
```

### 2.1.1 Ejercicio

Borrar el item seleccionado en el ListBox cuando pulsemos el botón borrar. Sólo borrará si hay un item seleccionado. Si no se mandará un mensaje indicando que no hay selección en la ListBox.

## 2.2 Manejando datos con LINQ y formularios

Creamos un proyecto esta vez que se llame AplicacionConLINQ de tipo Windows Form.

Vamos a cambiar de base de datos para este proyecto. Vamos a usar la versión reducida de SQL Server que viene embebida en Visual Studio 2019. Esta base de datos es para desarrollo no para un entorno de producción, con lo que sirve para pruebas locales. Instalamos los paquetes para manejar EntityFramework y SqlServer.

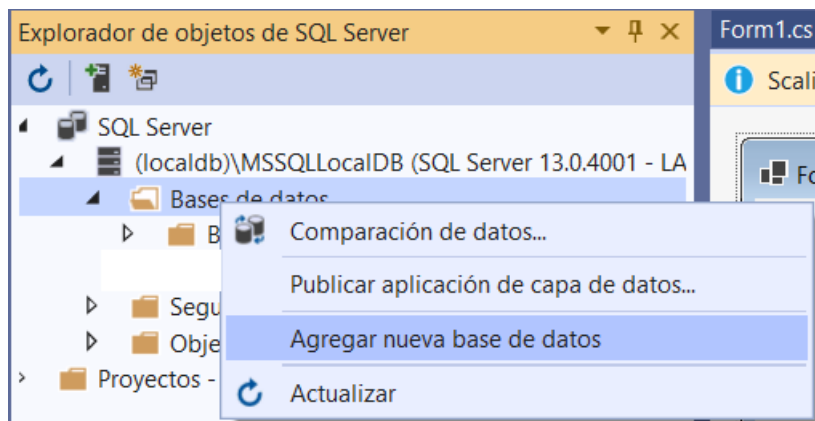
`Install-Package Microsoft.EntityFrameworkCore.SqlServer`

`Install-Package Microsoft.EntityFrameworkCore.Tools`

Vamos a crear una base de datos en el SQL Server local. Para ello abrimos el menú Ver y la pestaña Explorador de Objetos SQL Server.

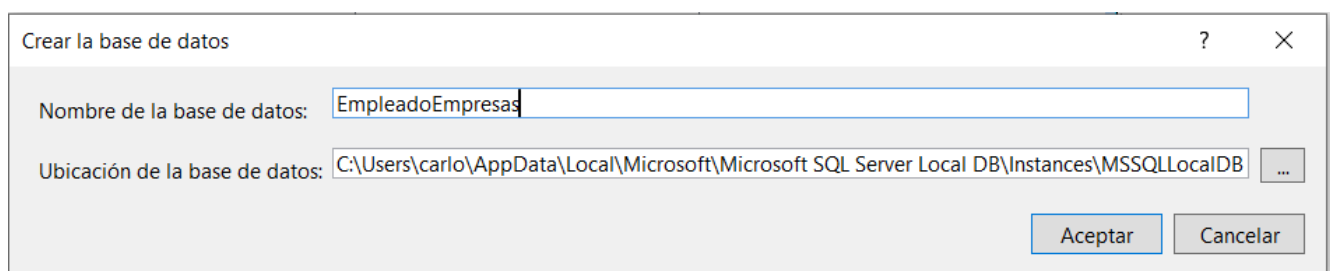


Seleccionamos en el SQLServer local MSSQLLocalDB, la carpeta Bases de Datos y pulsando en el botón derecho sobre esa carpeta pulsamos en agregar nueva base de datos. La instalación de LocalDB copia un conjunto de archivos mínimo necesario para iniciar el Motor de base de datos de SQL Server. Una vez que LocalDB está instalado, puede iniciar una conexión mediante una cadena de conexión especial.



Le llamamos a la Base de datos EmpleadoEmpresas. Se crearan dos ficheros en la ruta seleccionada. EmpleadoEmpresas.ldf y EmpleadoEmpresas.mdf. El fichero ldf es el log file de la base de datos, y el fichero mdf es el master, donde están los datos y el esquema de base de datos.

LocalDB se inicia a petición y se ejecuta en modo de usuario, sin necesidad de una configuración compleja. De forma predeterminada, LocalDB crea archivos de base de datos .mdf.

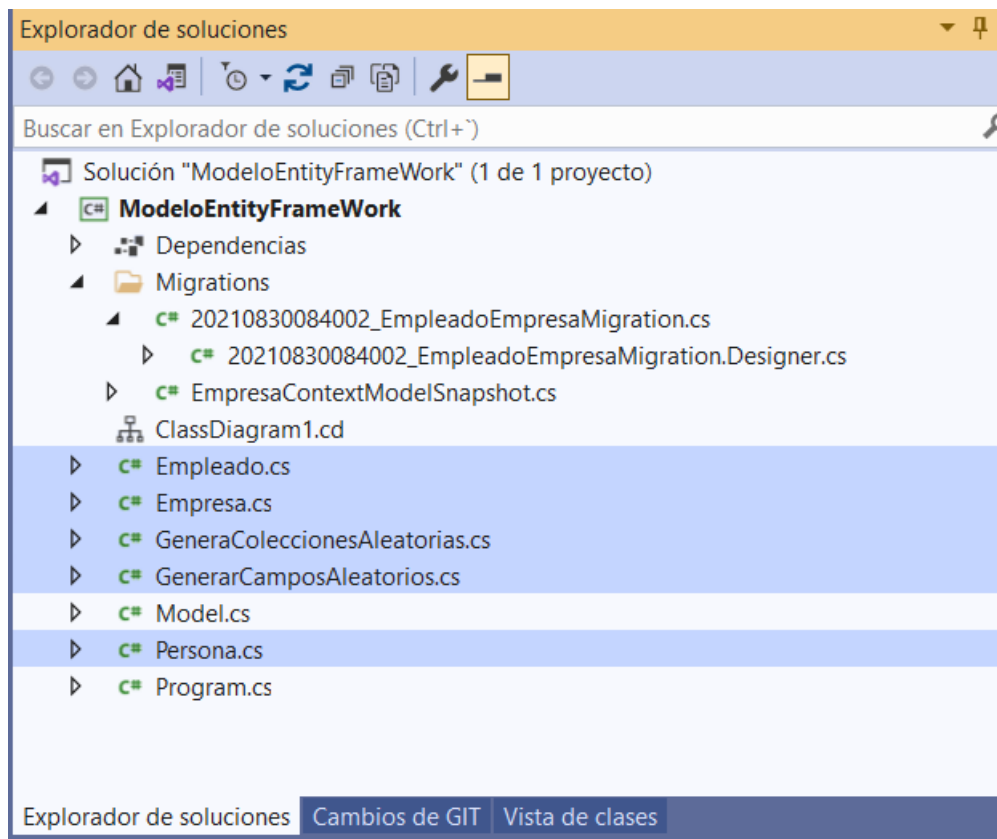


Carlo Ladera > AppData > Local > Microsoft > Microsoft SQL Server Local DB > Instances > MSSQLLocalDB

Nombre	Fecha de modificación	Tipo	Tamaño
EmpleadoEmpresas.ldf	31/08/2021 12:11	Archivo LDF	8.192 KB
EmpleadoEmpresas.mdf	31/08/2021 12:11	Archivo MDF	8.192 KB

Ya tenemos la base de datos creada, ahora vamos a copiar nuestro modelo de datos de la Aplicación que hicimos anteriormente ModeloEntityFramework. Copiáis las cinco clases marcadas

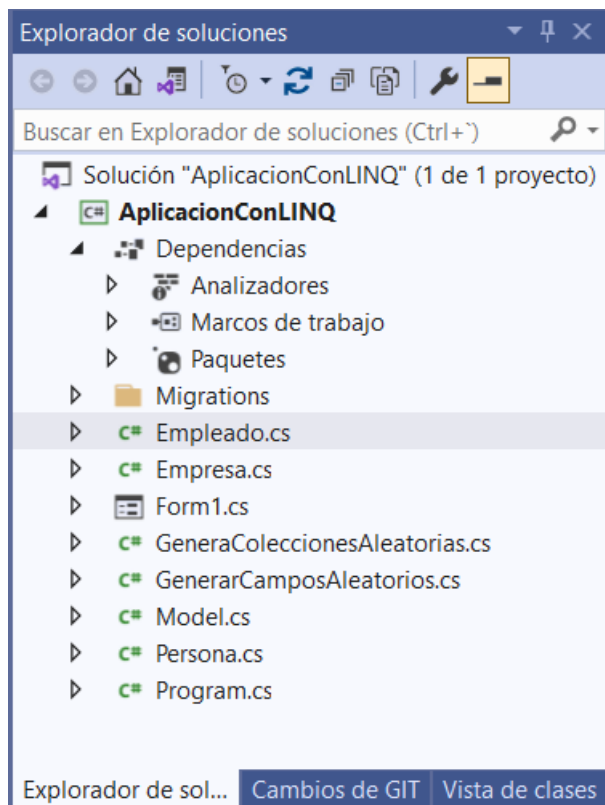




Copiarlas al proyecto nuevo y cambiar el espacio de nombres `ModeloEntityFrameWork` por `AplicaciónConLINQ` para todas ellas.

```
namespace ModeloEntityFrameWork
{
    [Table("Empleados")]
    public class Empleado : Persona
    {
    }
```

```
namespace AplicacionConLINQ
{
    [Table("Empleados")]
    public class Empleado : Persona
    {
    }
```



Tras añadir las clases y cambiar el espacio de nombres vamos a crear el modelo de datos con la API Fluent pero en este caso para la base de datos que hemos creado, EmpresaEmpleado.bmf de SQL Server. Debemos hacer pequeñas modificaciones para conectarnos a esta nueva base de datos. Entre otras cosas la cadena de conexión varía bastante de una base de datos a otra. Este es el nuevo modelo, hay que añadir el fichero Model.cs de nuevo. Los cambios están marcados en los cambios de código y los explicaremos además.

En el método OnConfiguring cambia el método al que llamamos y la cadena de conexión. Incluimos unos cuantos parámetros que son importantes para SQLServer. Lo primero es que el método al que llamamos ahora es `options.UseSqlServer`.

Lo segundo para hacer referencia a la base de datos local usamos = `(localdb)\\MSSQLLocalDB`, donde (localdb) es una propiedad que apunta a donde está instalado el pequeño motor SQLServer de base de datos `C:\Users\carlo\AppData\Local\Microsoft\Microsoft SQL Server Local DB\Instances`.

MSSQLLocalDB: es el motor de base de datos local SQLServer.

En la propiedad Initial Catalog indicámosle donde se haya nuestro fichero master de base de datos.

```
Initial Catalog =[AbsoluteFolderPath]\\EmpleadoEmpresa.MDF
```

Activamos la seguridad integrada .Net

```
Integrated Security = True;
```

Establecemos un tiempo de conexión de 30 segundos con la base de datos

```
Connect Timeout = 30;
```

La transmisión de datos sin encriptar, y no usamos certificados de autenticación para conectarnos al servidor. Estamos en local en un entorno de desarrollo-

```
Encrypt = False;
```

```
TrustServerCertificate = False;
```

La aplicación es de lectura y escritura en base de datos con la propiedad ApplicationIntent.

```
ApplicationIntent = ReadWrite;
```

La siguiente propiedad permite una detección más rápida del Servidor SQLServer si esta activa, a True. En este caso trabajando en local no nos merece la pena.

```
MultiSubnetFailover = False"
```

```
protected override void OnConfiguring(DbContextOptionsBuilder options)
```

```
options.UseSqlServer($"Data Source = (localdb)\\MSSQLLocalDB; Initial Catalog  
=[AbsolutePath]\\EmpleadoEmpresa.MDF; Integrated Security = True; Connect Timeout = 30;  
Encrypt = False; TrustServerCertificate = False; ApplicationIntent = ReadWrite;  
MultiSubnetFailover = False");
```

Además hemos cambiado el constructor, ahora hay una sola línea donde guardamos el nombre de la base de datos en la propiedad DbName.

```
public string DbName { get; private set; }
```

```
public EmpresaContext() {
```

```
    DbName = $"EmpleadosEmpresa.mdf";
```

Esto son todos los cambios realizados. Como veis la capa de datos es independiente del modelo de negocio. Podemos cambiar de motor base de datos fácilmente realizando ligeros cambios en nuestro fichero Model.cs. El resto de la aplicación no se verá afectada.

## Model.cs

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AplicacionConLINQ {

    public class EmpresaContext : DbContext
    {
        public DbSet<Empleado> Empleados { get; set; }
        public DbSet<Empresa> Empresas { get; set; }

        public string DbName { get; private set; }

        public EmpresaContext() {

            DbName = $"EmpleadosEmpresa.mdf";

        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Empleado>()
                .HasKey(e => new { e.Id, e.IdEmpresa });

            modelBuilder.Entity<Empresa>()
                .HasMany<Empleado>(g => g.Empleados)
                .WithOne()
                .HasForeignKey(emp => emp.IdEmpresa)
                .OnDelete(DeleteBehavior.Cascade);

        }

        // The following configures EF to create a Sqlite database file in the
        // special "local" folder for your platform.
        protected override void OnConfiguring(DbContextOptionsBuilder options)
        {
            options.UseSqlServer($"Data Source = (localdb)\\MSSQLLocalDB; Initial Catalog
            =[AbsolutePath]\\EmpleadoEmpresa.MDF; Integrated Security = True; Connect Timeout = 30;
            Encrypt = False; TrustServerCertificate = False; ApplicationIntent = ReadWrite;
            MultiSubnetFailover = False");
        }

    }

}
```

Ahora realizamos la migración en la consola de paquetes NuGet :



```
Add-Migration MigrationEmpleadoSQLServerForm
```

Y creamos la base de datos con el comando

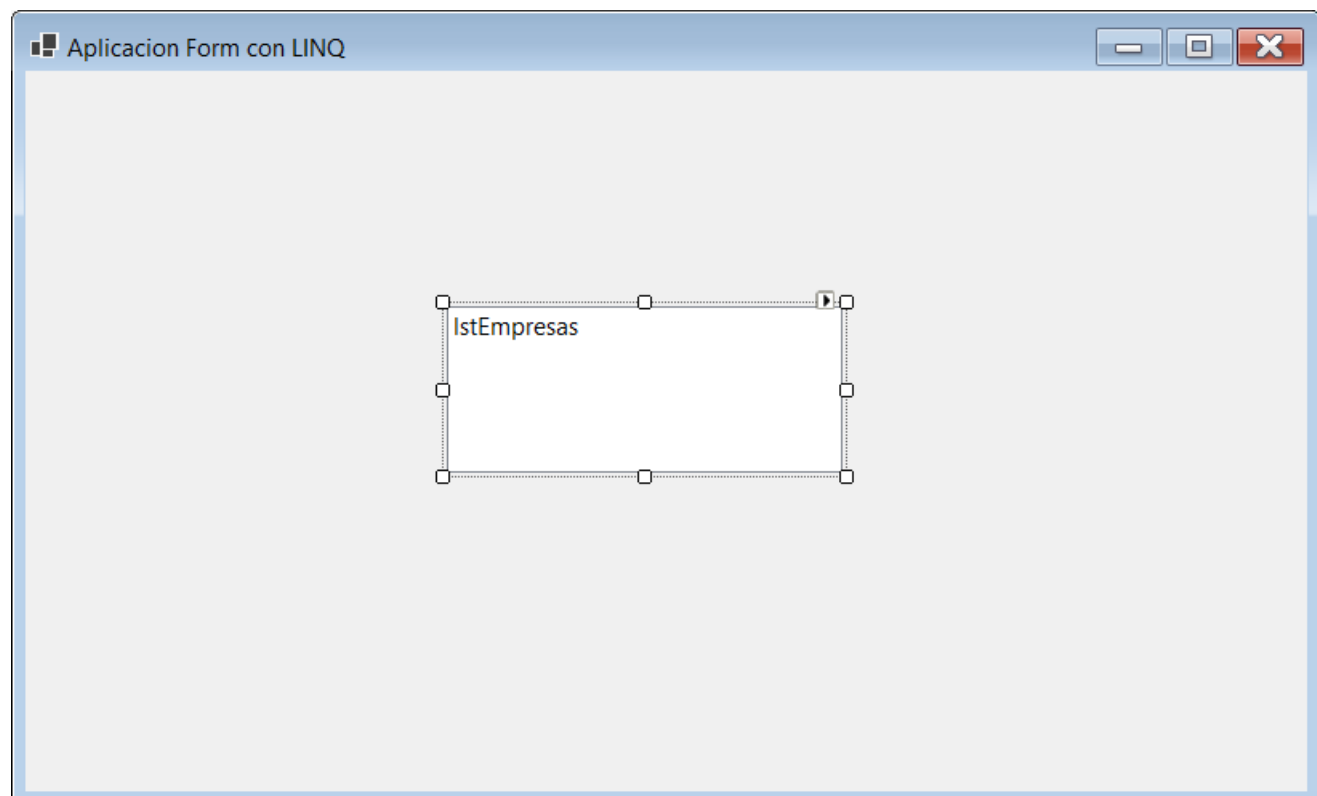
```
Update-Database
```

Comprobar que se han creado los dos ficheros de base de datos en vuestra carpeta MSSQLLocalDB.

s Cano Ladera > AppData > Local > Microsoft > Microsoft SQL Server Local DB > Instances > MSSQLLocalDB

Nombre	Fecha de modificación	Tipo	Tamaño
 EmpleadoEmpresas.ldf	31/08/2021 12:11	Archivo LDF	8.192 KB
 EmpleadoEmpresas.mdf	31/08/2021 12:11	Archivo MDF	8.192 KB

Ahora vamos a añadir al formulario principal que llamareis VentanaPrincipal, un listBox de nombre lstEmpresas.



Y además vamos a añadir el siguiente código para generar datos para la base de datos, en Form1.cs

Como veis en el constructor de Form1.cs

Inicializamos la base de datos con valores

```
inicializarBaseDeDatos();
```

Cargamos los componentes.

```
InitializeComponent();
```

Cargamos el listBox con la razón social de la empresa en cuestión.

```
db.Empresas.Select(empire => empire).ToList().ForEach(empire =>
lstEmpresas.Items.Add(empire.RazonSocial));
```

En el método `private void inicializarBaseDeDatos()` hacemos lo mismo que en ejemplos anteriores de LINQ, borrar la base de datos si esta llena, generar valores aleatorios y llenar la base de datos.

```
using AplicacionConLINQ.Debug;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace AplicacionConLINQ
{
    public partial class VentanaPrincipal : Form
    {
        public EmpresaContext db { get; set; }
        public VentanaPrincipal()
        {
            inicializarBaseDeDatos();
            InitializeComponent();

            db.Empresas.Select(empire => empire).ToList().ForEach(empire =>
lstEmpresas.Items.Add(empire.RazonSocial));
        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
        }
    }
}
```

```

private void inicializarBaseDeDatos()
{
    db = new EmpresaContext();

    db.Empleados.Select(emp => emp).ToList().ForEach(emp => db.Remove(emp));
    db.SaveChanges();

    db.Empleados.Select(emp => emp).ToList().ForEach(emp => db.Remove(emp));
    db.SaveChanges();

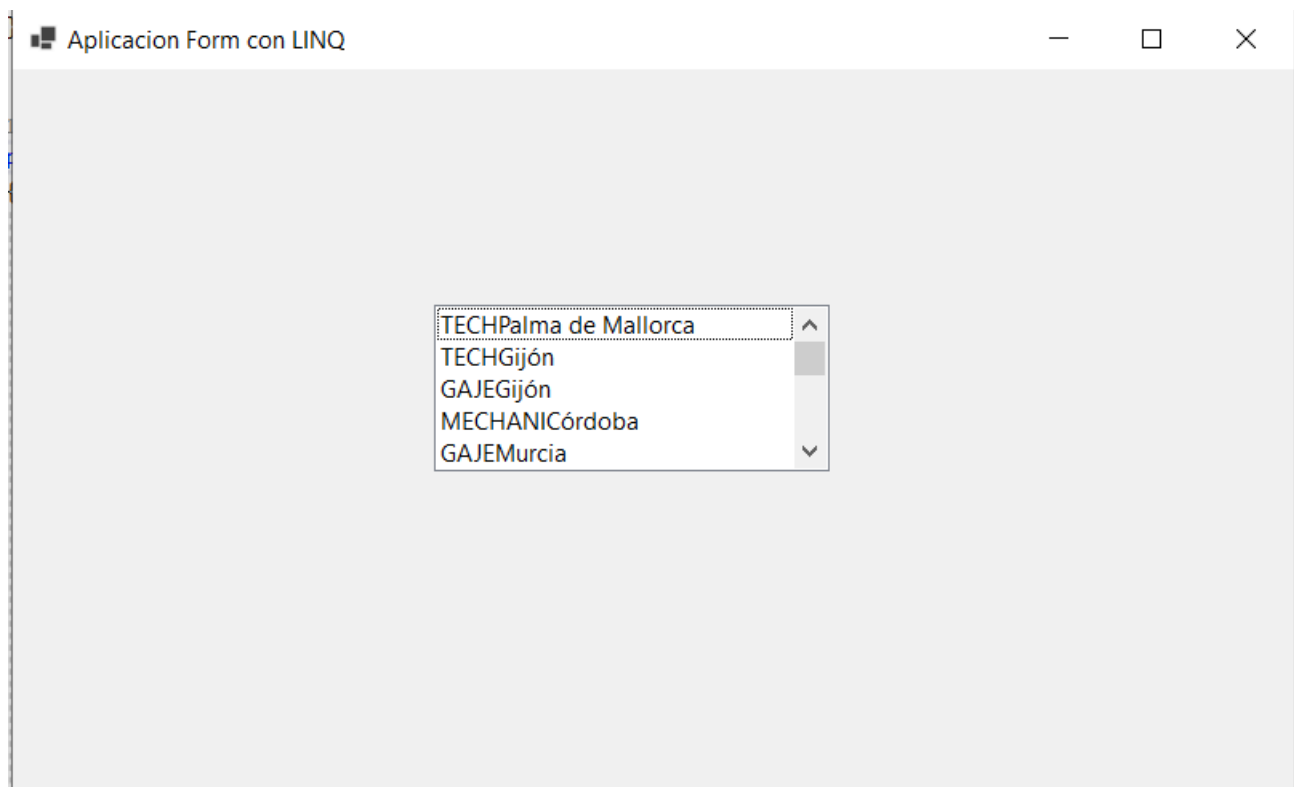
    List<Empresa> empresas = GeneraColeccionesAleatorias.listaEmpresas(60);

    empresas.ForEach(empresa => db.Add(empresa));

    db.SaveChanges();
}
}
}

```

Ejecutar el programa y deberéis obtener un resultado similar a este, donde el ListBox ha sido cargado con datos de la base de datos SQLServer.



Después de ejecutar por primera vez el programa comentar el método inicializar base de datos. La base de datos esta ya llena no hace falta rellenarla muchas veces.

```
public VentanaPrincipal()  
{  
    //inicializarBaseDeDatos();  
    InitializeComponent();  
  
    db.Emresas.Select(empres => empres).ToList().ForEach(empres =>  
lstEmresas.Items.Add(empres.RazonSocial));  
}
```