Tabla de contenido

1. Formularios y Controles

Un formulario Windows representa la conocida ventana, que se utiliza en las aplicaciones ejecutadas bajo alguno de los sistemas operativos de la familia Windows. Tendremos que fijarnos un poquito más en estas ventanas para mantener los estándares ya fijados y que ayudaran al usuario a gestionar mejor nuestra aplicación.

Un control, por otra parte, es aquel elemento situado dentro de una ventana o formulario, y que permite al usuario de la aplicación Windows, interactuar con la misma, para introducir datos o recuperar información.

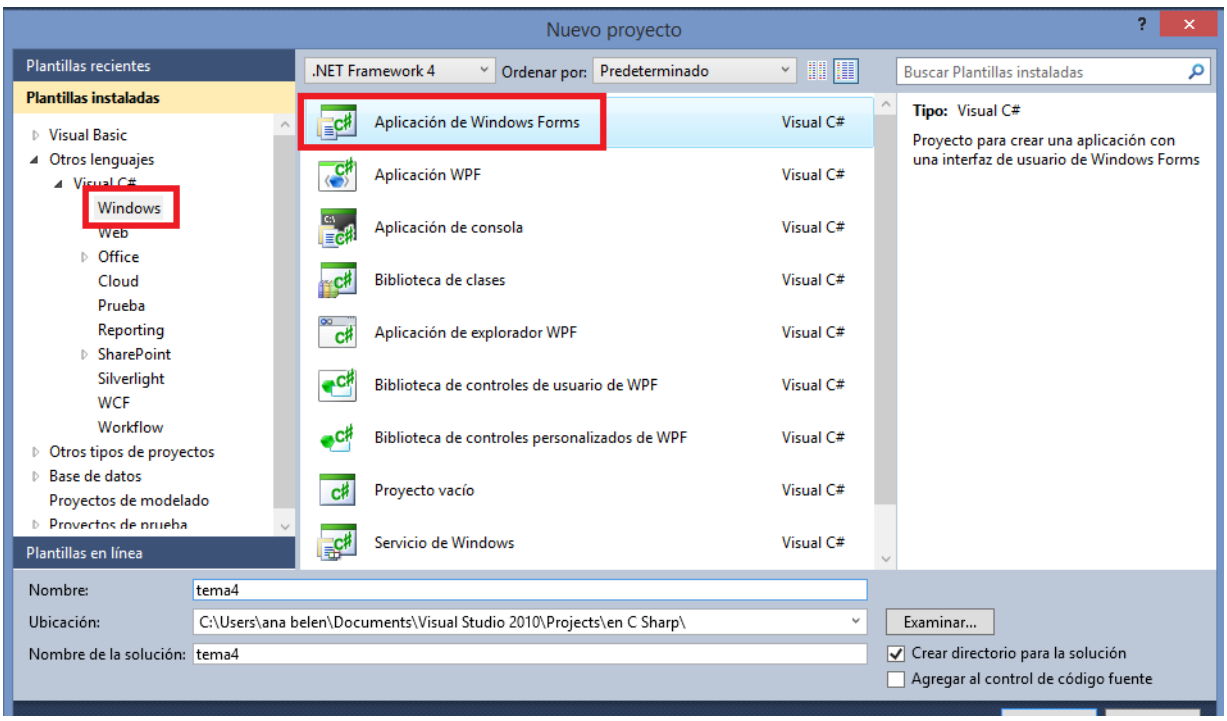
Dentro de .NET, las ventanas clásicas Windows, reciben la denominación de Windows Forms, o WinForms, para diferenciarlas de los formularios Web o WebForms, que son los que se ejecutan en páginas ASP.NET.

System.Windows.Forms es el espacio de nombres que contiene todos los tipos del entorno, a través de los cuales podremos desarrollar aplicaciones compuestas por formularios Windows, junto a los correspondientes controles que permiten al usuario la interacción con el programa.

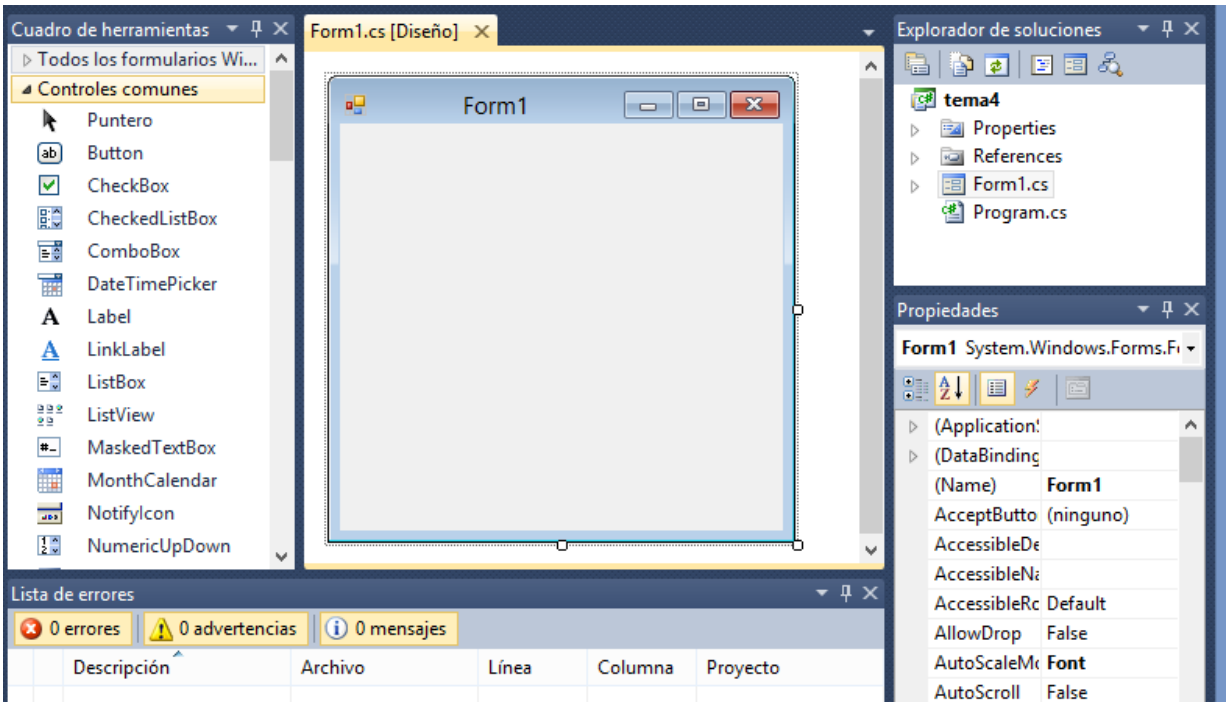
La clase **Form** contiene todos los miembros para la creación y manipulación de formularios. Tras instanciar un objeto de Form, mediante la configuración de las adecuadas propiedades, podemos crear formularios estándar, de diálogo, de interfaz múltiple o MDI, con diferentes bordes, etc.

2. Creación de un Formulario Básico

Lo primero, en Nuevo Proyecto, en Plantillas, ya no seleccionamos Aplicación de Consola, sino de Windows Form



Y observamos que ya mi escritorio cambia, apareciendo en el **Explorador de Soluciones**, además **Programs**, el fichero **Form1**, que por defecto ya aparece abierto (Formulario en Blanco) y el **Cuadro de Herramientas** activada, y la **Ventana de Propiedades** para cada uno de los objetos que seleccionemos, en este caso la del Form1.



2.1. Diseño

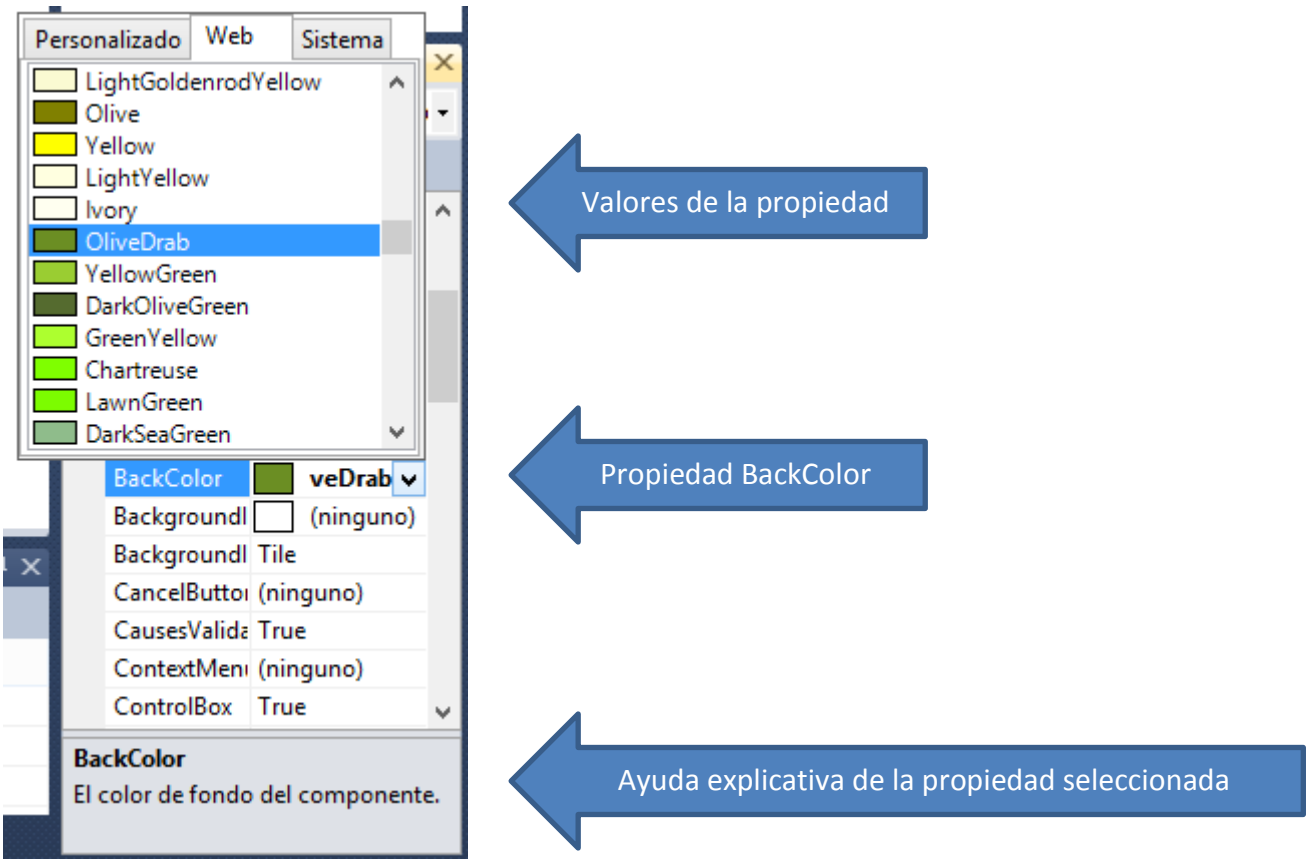
En lo que respecta al diseño del formulario, podemos modificar su tamaño haciendo clic sobre las guías de redimensión que tiene en los bordes de la plantilla de diseño, y arrastrando hasta dar el tamaño deseado.

Las guías de color blanco son las que permiten modificar el tamaño. Por ejemplo, si vamos a incluir muchos controles, o un título largo, y el tamaño que tiene por defecto no es lo bastante grande, lo ampliaremos hasta quedar como muestra la siguiente imagen.

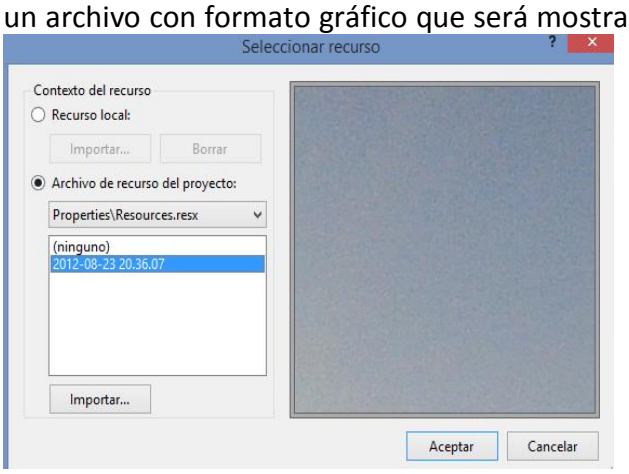
También podemos conseguir el mismo efecto de cambiar el tamaño del formulario desde la ventana de propiedades, asignando valores a la propiedad **Size**. Para ello, haremos clic en el icono de expansión de esta propiedad y daremos valores a sus elementos X e Y. Vemos que una vez que modificamos el valor que viene por defecto, este se pondrá en negrita.

Igualmente haremos con la propiedad **Location**, de modo que cambiaremos las coordenadas iniciales en las que el formulario será visualizado. Para que el formulario se visualice en estas coordenadas que establecemos la propiedad **StartPosition** a valor **Manual**.

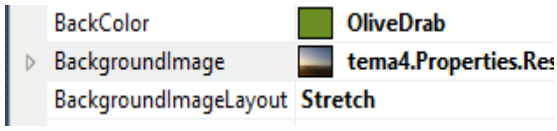
Para modificar el color de fondo, seleccionaremos el color de la propiedad **BackColor**:



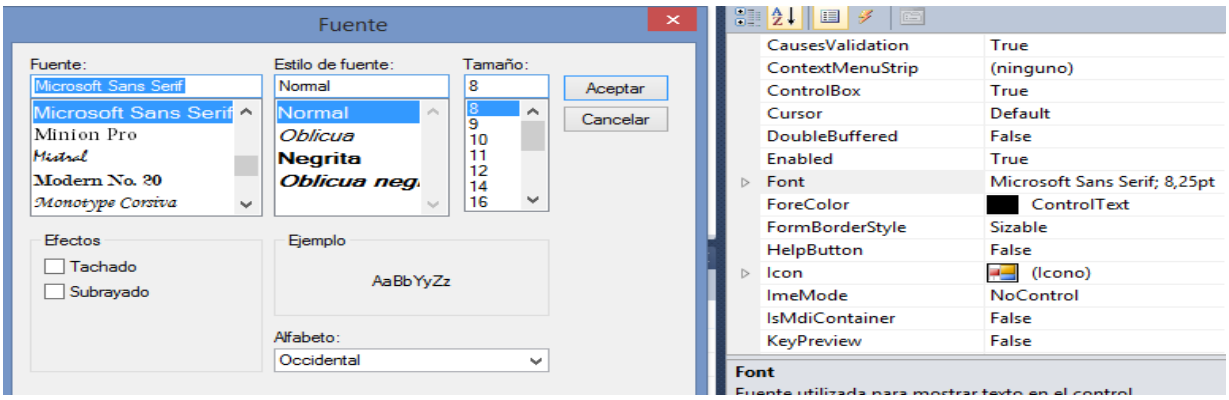
Si queremos una imagen de **fondo,** en vez de un color, recurriremos a la propiedad **BackgroundImage**, que nos mostrará una caja de diálogo, mediante la que seleccionaremos un archivo con formato gráfico que será mostrado en la superficie de la ventana. Si por algún motivo, necesitamos eliminar dicha imagen de fondo para el formulario, haremos clic en Ninguno.



En este caso al elegir una foto… tendremos que ajustarla al tamaño de nuestro formulario, con lo que también tenemos que tocar la propiedad **BackgroundImageLayout**, y elegir **Stretch**



Es bueno que también configuremos el tipo de letra que queremos por defecto para todos los controles que insertemos en nuestro formulario, para eso tocaremos las propiedades **Font** y **ForeColor**.



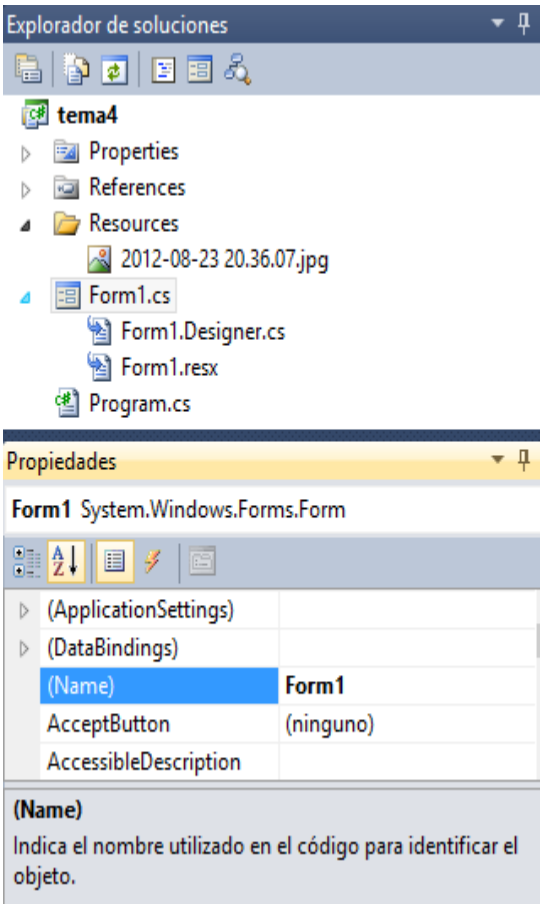
Podemos modificar nuestra barra de Título, tanto lo que pone en ella, que nunca debe quedarse escrito "Form1", a través de la propiedad **Text**. Así como si queremos que aparezcan o no los **botones** de Maximizar, Minimizar, Cerrar y Ayuda:

- Podemos quitarlos todos poniendo a **False** la propiedad **ControlBox**.
- O de forma independiente: **MaximizeBox**, **MinimizeBox**

2.2. Cambiar el nombre al Formulario

Y otras propiedades que iremos utilizando según vayamos avanzando en nuestro diseño.

Pero la más importante y que hemos dejado para último lugar en este punto… pero que debe ser la primera que ajustemos debe ser la propiedad **(Name)**



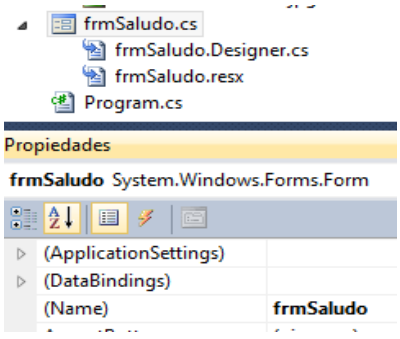
Es importante, aunque no es obligado, que el nombre de nuestro formulario sea igual al nombre del fichero.cs, nos evitará errores.

Para nuestros formularios utilizaremos la nomenclatura **frmNombreForm**

- **frm**: Para indicar que es un objeto Form
- **NombreForm**: Seguido, con minúsculas y Mayúsculas para dar un significado al nombre de lo que va a contener dicho formulario.

Para cambiar el nombre del Fichero, nos iremos al **Explorador de Soluciones** y ahí lo cambiamos.

Quedando:

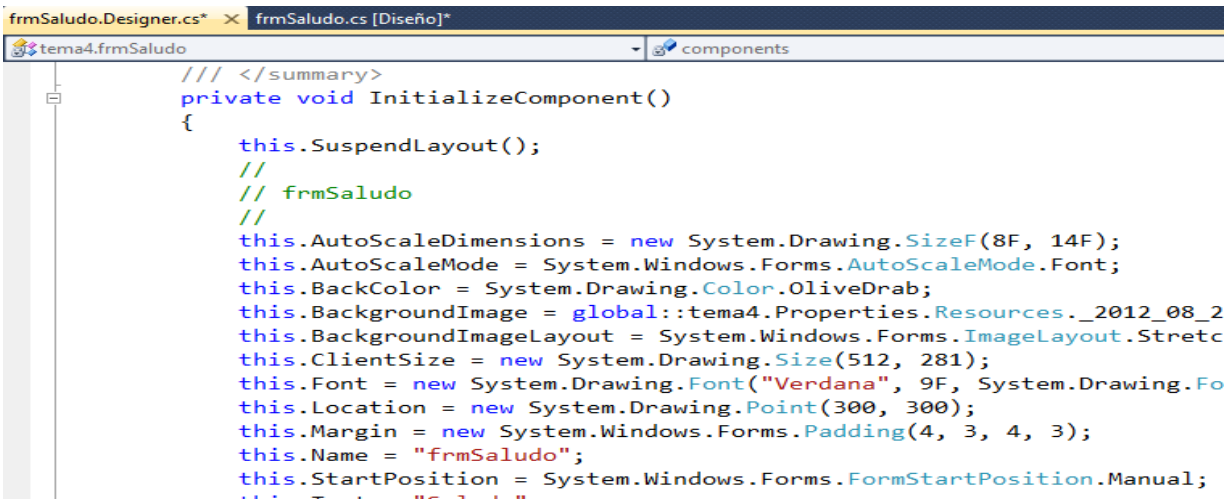


2.3. Código

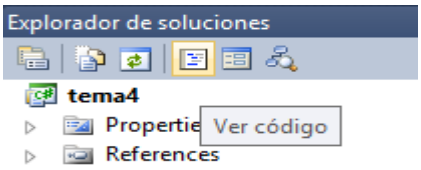
Cuando creamos un formulario desde Visual Studio .NET del modo en que acabamos de mostrar, el diseñador del formulario genera por nosotros el código del formulario, que consiste en una clase que hereda de la clase base Form. El nombre de la clase es el mismo que hemos asignado a la propiedad Name en la ventana de propiedades del diseñador, en este caso Form1.

El código es grabado en un archivo con la extensión **.cs** (o vb si estuviéramos en VisualBasic), que tiene el nombre del formulario: **Form1.cs**, en este ejemplo.

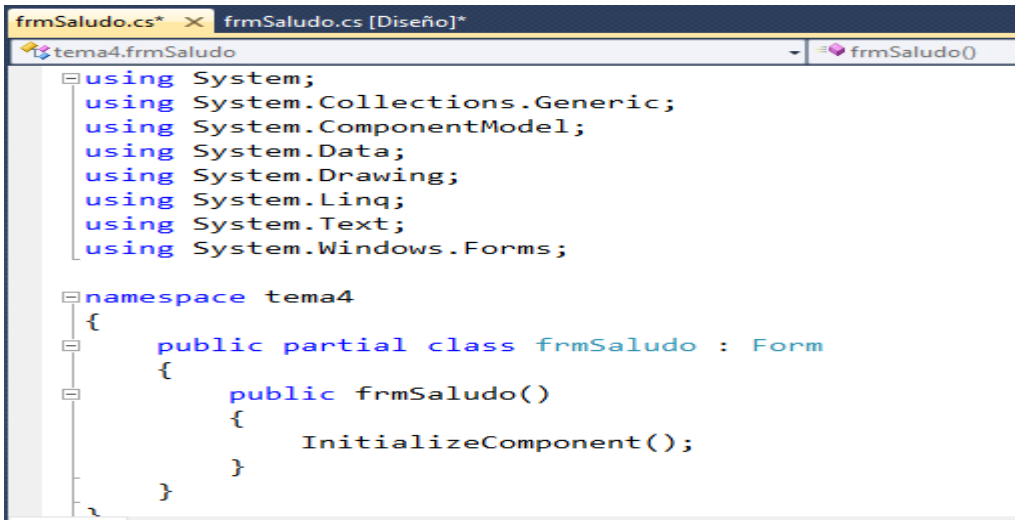
En el fichero **.Designer.cs** es donde se va escribiendo el código que generamos a través del diseñador:



Para ver dicho código, tan sólo tenemos que hacer clic derecho sobre el formulario, y en el menú contextual seleccionar **Ver código**, lo que abrirá la ventana del editor de código del IDE, mostrando el código de nuestro formulario.



Donde nosotros escribiremos nuestro código será:



Es posible modificar este código generado por el diseñador, para completar aquellos aspectos que necesitemos del formulario. Sin embargo, no debemos modificar el método **InitializeComponent( );** ya que se trata de un método directamente relacionado con el aspecto visual del formulario, y su edición podría dejar el formulario inservible.

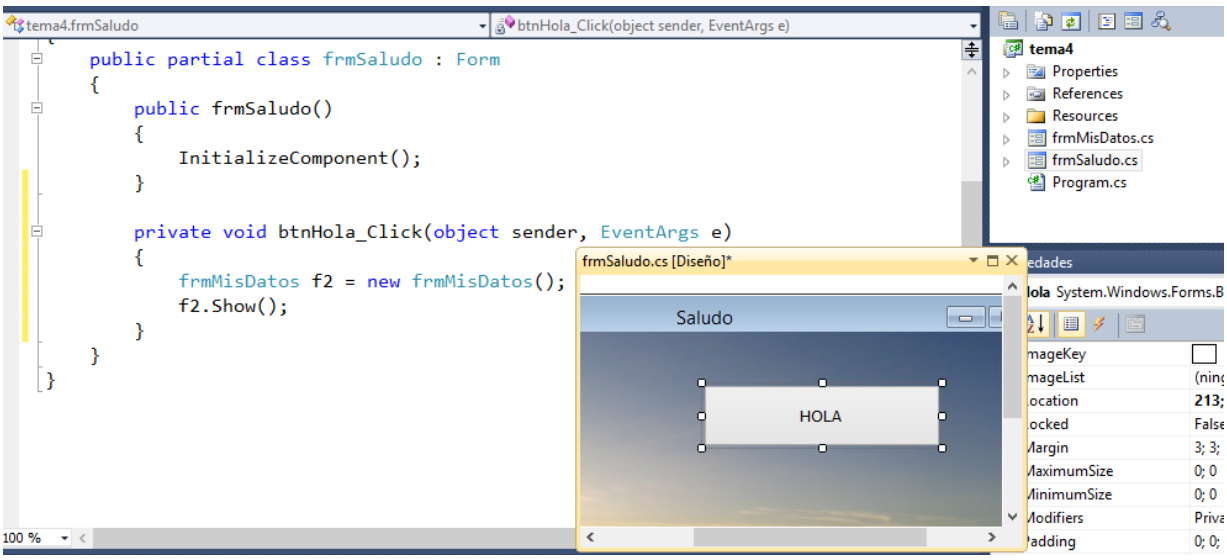
Sin olvidarnos de nuestro fichero **Program.cs**

```
tema4.Program                                   Main()

namespace tema4
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmSaludo());
        }
    }
}
```

2.4. Abrir formularios

Cuando queremos abrir un formulario desde otro… deberemos hacer una llamada a instanciar el nuevo objeto formulario:

Si en mi formulario **frmSaludo** tenemos un botón "**Hola**" llamado **btnHola**, su código para abrir el nuevo formulario: **frmMisDatos** será el siguiente:



2.5. Formulario contenedor de otros formularios

Podemos hacer que cuando desde un formulario llamamos a otro, este no se abra en otra ventana sino dentro del formulario que lo ha llamado, para ello seguimos los siguientes pasos:

- Indicamos al formulario contenedor que va a ser padre para ello: Su propiedad **IsMdiContainer** la ponemos a True.
- En el código, cuando llamamos a mostrar el otro formulario, solo tenemos añadir la instrucción en la que presentamos al padre, quedando todo lo demás igual. Observa que para hacer referencia al formulario en el que me encuentro, utilizo la palabra clave **this**.

```
private void btnHola_Click(object sender, EventArgs e)
{
    frmMisDatos f2 = new frmMisDatos();

    f2.MdiParent = this;
    f2.Show();
}
```

- Para cerrar formularios hijos, comprobamos que hay alguno abierto, y vemos que sería un vector de formularios hijos… pudiendo acceder a ellos por la posición o por la propiedad **Name**.

```
private void btnCerrar_Click(object sender, EventArgs e)
{
    if (this.MdiChildren.Length > 0)
        this.MdiChildren[0].Close();
    else MessageBox.Show("No hay ningún formulario hijo");

    if(this.MdiChildren[0].Name =="frmMisDatos")
        this.MdiChildren[0].Close();

}
```

3. Los Controles

Los controles proporcionan al usuario el medio para comunicarse con nuestro formulario, y en definitiva, con la aplicación. Por ello, en los siguientes apartados, mostraremos los principales aspectos que debemos tener en cuenta a la hora de su creación, manipulación y codificación.

3.1. Generalidades

Una vez creado un proyecto, o después de añadir un nuevo formulario, para utilizar controles en el mismo, tendremos que tomarlos de la ventana **Cuadro de herramientas** disponible en el IDE de VS.NET, y añadirlos al formulario.

Para añadir un control en el formulario, proceso que también se conoce como dibujar un control, debemos seleccionar primeramente el control a utilizar de la lista que aparece en el cuadro de herramientas. Hacer clic sobre el control, situar el cursor del ratón en la superficie del formulario y hacer clic en él, arrastrando hasta dar la forma deseada; de esta manera, proporcionamos al control en un solo paso la ubicación y tamaño iniciales.

Un control, al igual que un formulario, dispone a su alrededor de un conjunto de guías de redimensión, de modo que si después de situarlo en el formulario, queremos modificar su tamaño, sólo tenemos que hacer clic sobre alguna de estas guías, y arrastrar modificando las dimensiones del control.

Además de utilizando el ratón, podemos desplazar un control, manteniendo pulsada la **tecla ctrl.**., y pulsando además, algunas de las teclas de dirección.

3.2. Ajuste de la Cuadrícula

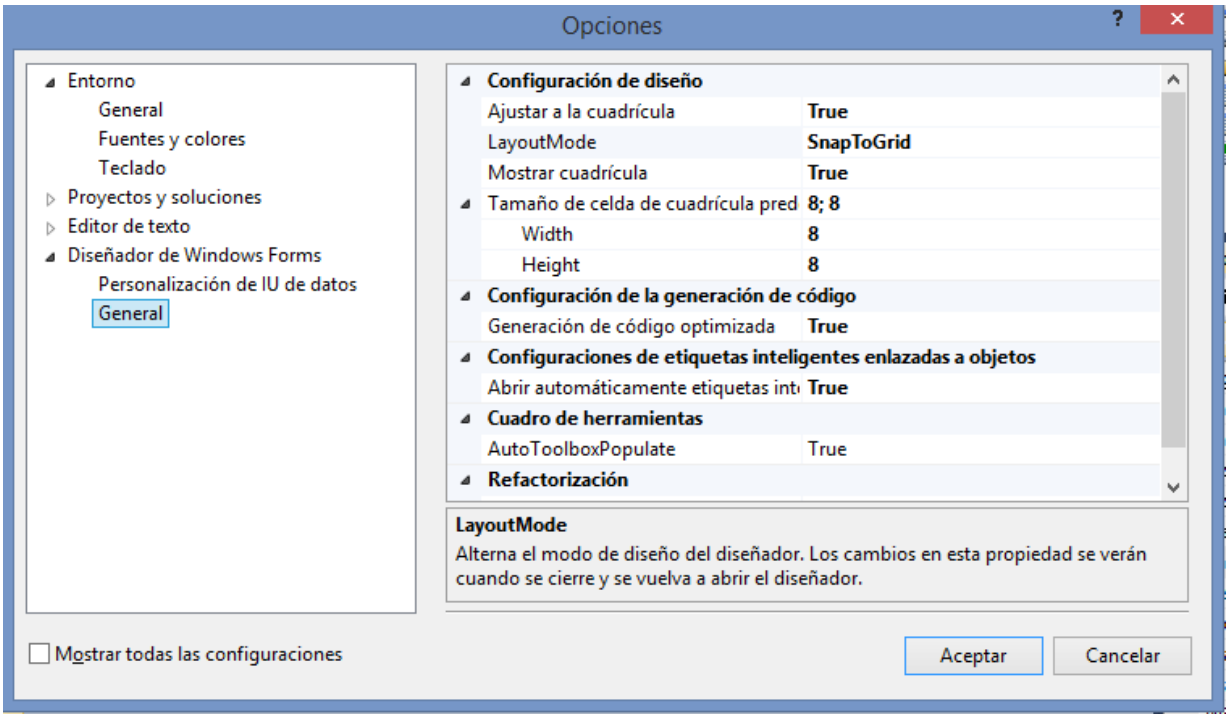
La cuadrícula de diseño del formulario, consiste en el conjunto de líneas de puntos que surcan la superficie del formulario, **de forma visible o invisible**, y nos sirven como ayuda, a la hora de realizar un ajuste preciso de un control en una posición determinada.

Al haber realizado algunas prácticas situando controles en el formulario, te habrás percatado de que cuando movemos un control con el ratón, dicho control se ajusta a la cuadrícula obligatoriamente, por lo que no podemos ubicarlo entre dos líneas de puntos de la cuadrícula.

Para modificar lo que ya viene por defecto, incluso que los puntitos sean visibles o no… se configura desde **el menú Herramientas** → Opciones → Diseñador de Windows Forms → General.

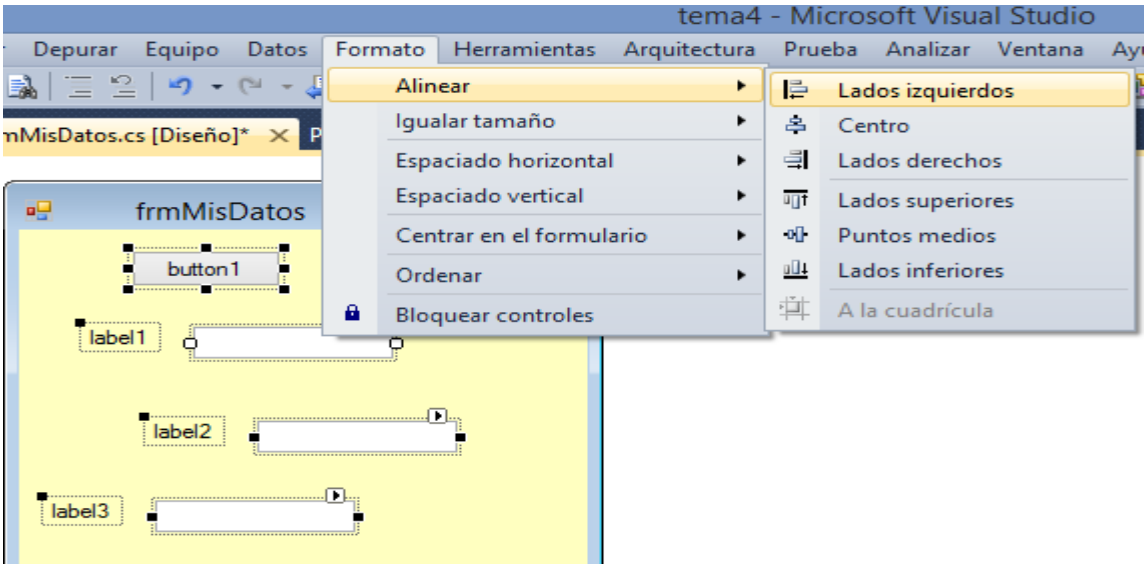
- Observa la siguiente imagen y los valores dados… Estos serán aplicados, al Aceptar esta ventana, y al cerrar y volver abrir los formularios.
- Modificando la propiedad **GridSize** (tamaño de celda de cuadrícula), al cambiar los valores de espaciado de puntos que tiene la rejilla (8x8), modificamos el ajuste de forma que cuanto menor sea ese valor, más junta estará la trama de puntos de la rejilla, con lo que podremos ajustar de forma más exacta el control. Este ajuste es válido sólo para el formulario sobre el que lo aplicamos.

- La propiedad **LayaoutMode** es la que nos permite hacer los puntitos visibles o invisibles.



3.3. Organización de los controles del formulario

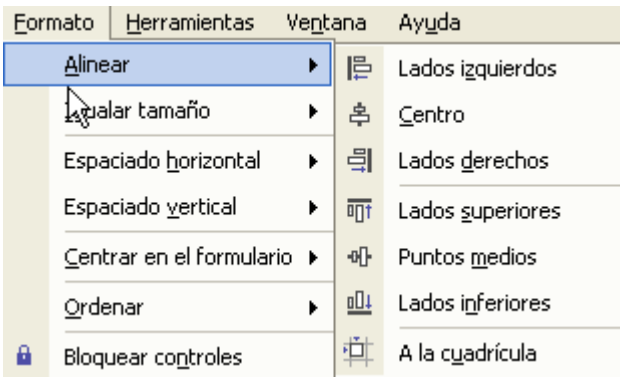
Cuando tenemos un grupo numeroso de controles en el formulario, que necesitamos mover de posición, o cambiar su tamaño, para redistribuir el espacio, podemos optar por cambiar uno a uno los controles, tarea pesada y nada aconsejable, o bien, podemos seleccionar todos los controles a modificar, y realizar esta tarea en un único paso, mediante las opciones del **menú Formato**.



En primer lugar, **para seleccionarlos todos**, debemos hacer clic sobre el formulario y arrastrar, de modo que el rectángulo de selección que aparece, abarque a los controles, que quedarán con sus correspondientes marcas de redimensión visibles, señal de que están seleccionados. O bien, pulsamos la tecla **ctrl**. y con el ratón clikeamos sobre los controles que queremos seleccionar.

En este punto, podemos hacer clic en uno de los controles y desplazarlos todos conjuntamente por el formulario, o bien, hacer clic en una de las guías de redimensión y cambiar su tamaño, lo que afectará a todos los controles seleccionados. Si necesitamos de alguna acción especial, utilizaremos las opciones del menú Formato del IDE.

Por ejemplo, podemos ejecutar la opción *Formato →Alinear → Lados izquierdos*, de modo que todos los controles se alinearán por la izquierda, tomando como referencia el control que tiene las marcas de redimensión negras.



Después ejecutaremos la opción de menú *Formato + Igualar tamaño + Ambos*, que ajustará tanto el ancho como el alto de todos los controles seleccionados.

Para evitar que, una vez completado el diseño y ajuste de todos los controles, accidentalmente podamos modificar alguno, seleccionaremos la opción de menú ***Formato → Bloquear controles***, que bloqueará los controles seleccionados, impidiendo que puedan ser movidos o modificado su tamaño. Para desbloquear los controles del formulario, debemos seleccionar al menos uno y volver a utilizar esta opción de menú, que desbloqueará todos los controles.

Una característica interesante del bloqueo de controles, consiste en que una vez que tengamos bloqueados los controles del formulario, si añadimos un nuevo control, este no estará inicialmente bloqueado, lo que facilita su diseño. Una vez que hayamos finalizado de diseñar el último control, lo seleccionaremos en el formulario y seleccionaremos la opción de bloqueo de controles, de modo que ya estarán bloqueados todos de nuevo.

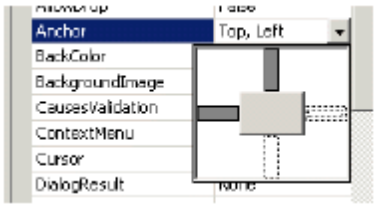
El bloqueo de controles es a efecto Diseño, pero no a efecto ejecución. Es decir, en tiempo de ejecución, yo podría ejecutar una instrucción que me modificase el tamaño de un control bloqueado en Diseño.

3.4. Anclaje de Controles

La propiedad **Anchor**, existente en un gran número de controles, nos permite *anclar* dicho control a uno o varios bordes del formulario.

Cuando un control es anclado a un borde, la distancia entre el control y dicho borde será siempre la misma, aunque redimensionemos el formulario.

Para establecer esta propiedad, debemos pasar a la ventana de propiedades del control, y en Anchor, pulsar el botón que dispone, y que nos mostrará una representación de los bordes para anclar.

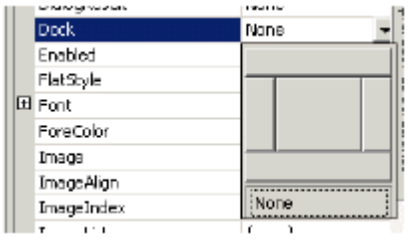


Las zonas de color gris oscuro representan los bordes del control que ya están anclados a los bordes del formulario. Debemos marcar y desmarcar respectivamente estos elementos según los bordes que necesitemos anclar. Por defecto, los controles se encuentran inicialmente anclados a los bordes superior e izquierdo (Top, Left), como hemos comprobado en la anterior figura.

3.5. Acople de controles

A través de la propiedad **Dock** de los controles, podremos *acoplar* un control a uno de los bordes de un formulario, consiguiendo que dicho control permanezca pegado a ese borde del formulario en todo momento.

Para seleccionar el tipo de acople, haremos clic en el botón que tiene la propiedad Dock en la ventana de propiedades, y que nos mostrará un guía de los tipos de acople disponibles.



Por defecto, los controles no se encuentran acoplados al insertarse en el formulario, y sólo es posible establecer un tipo de acople en cada ocasión.

Vemos que también tenemos la opción Fill que sería el "completo", nos puede venir bien por ejemplo para cuando abrimos formularios hijos y quiero que ocupen la totalidad del área del formulario padre, porque todo lo que hacemos a través del diseño, también lo podemos hacer a través del código:

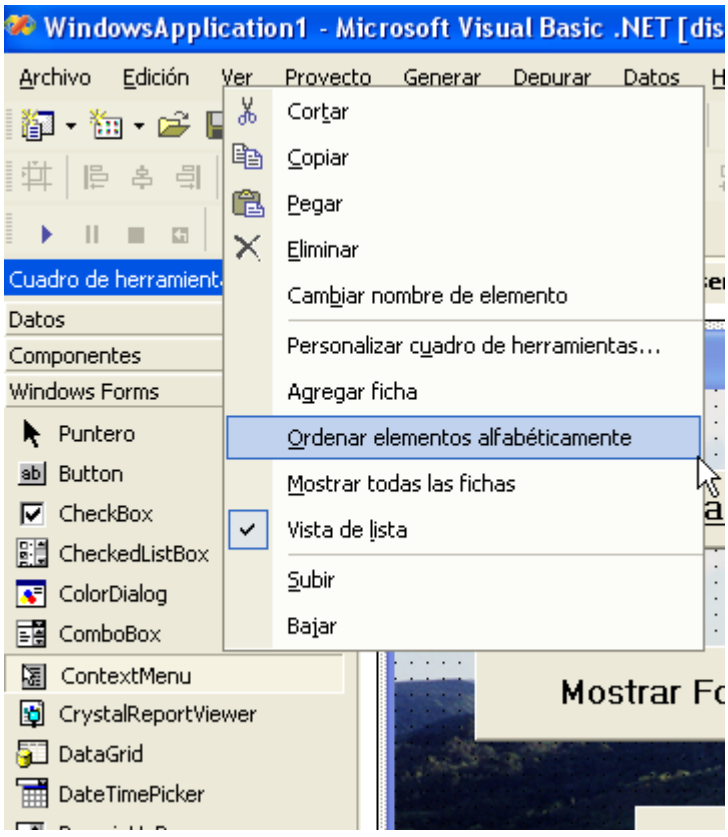
```
private void btnHola_Click(object sender, EventArgs e)
{
    frmMisDatos f2 = new frmMisDatos();

    f2.MdiParent = this;
    f2.Dock = DockStyle.Fill;    // Se acoplará por completo (fill)
                                 //     al formulario padre
    f2.Show();

}
```

4. Controles más habituales

Como habrás podido comprobar, el número de controles del cuadro de herramientas es muy numeroso, por lo que es posible que no veamos todos, aunque sí aquellos que podamos utilizar con más frecuencia.

Para ver los distintos controles en este tema, están recogidos por orden alfabético. Aunque no necesariamente a la hora de trabajar con ellos lo haremos en este mismo orden.

Sabemos que nuestro cuadro de herramientas podemos ordenarlo alfabéticamente, y que una vez ordenado, ya no podemos "desordenarlo".

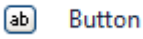


En los subtítulos de este punto, verás el nombre de cada control que lo identifica en el cuadro de herramientas, su traducción en español, **las tres letras que utilizaremos para identificarlo** con nomenclatura polaca o de prefijos y el icono que vemos en el cuadro de herramientas.

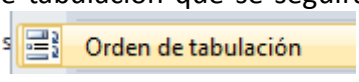
Aquellas propiedades que sean similares para distintos controles no se repetirán si ya se han visto anteriormente para otro.

Vemos también que el Cuadro de Herramientas tiene los **controles agrupados por fichas**. Aunque la primera es en la que están "Todos"

4.1. Button – Botón de comando – btn



Este control representa un botón de pulsación, conocido en versiones anteriores de VB como CommandButton. Entre el nutrido conjunto de propiedades de este control, destacaremos las siguientes.

- **(Name)** Nombre que lo identificará en el código. Esta propiedad es obligatoria modificarla.
- **Anchor** Anclaje del botón, para que cuando se modifique el tamaño del formulario mantenga las distancias.
- **BackColor**. Color de fondo para el botón.
- **BackgroundImage**. Imagen de fondo para el botón.
- **Cursor**. Permite modificar el cursor del ratón que por defecto tiene el botón.
- **Dock**. Acoplamiento del control en el formulario
- **Enabled**. Habilita el control para que pueda ser pulsado por el usuario.
- **FlatStyle**. Tipo de resaltado para el botón. Por defecto, el botón aparece con un cierto relieve, que al ser pulsado, proporciona el efecto de hundirse y recuperar nuevamente su estado, pero podemos, mediante esta propiedad, hacer que el botón se muestre en modo plano, con un ligero remarcado al pulsarse, etc.
- **Font**. Cambia el tipo de letra y todas las características del tipo elegido, para el texto del botón.
- **ForeColor**. Indica el color del texto que aparece en el botón
- **Image**. Imagen que podemos mostrar en el botón como complemento a su título, o bien, en el caso de que no asignemos un texto al botón, nos permitirá describir su funcionalidad.
- **ImageAlign**. Al igual que para el texto, esta propiedad nos permite situar la imagen en una zona del botón distinta de la central, que es en la que se ubica por defecto.
- **ImageIndex**. Indica qué imagen elegimos de la **ImageList** que hemos vinculado al formulario.
- **Location**. Posición de la esquina superior izquierda del control con respecto al contenedor. Principalmente desde el diseño.
- **Locked**. Determina si se puede mover o modificar el tamaño. Es hacer de forma individual lo que podemos hacer para todos los controles desde Formato → Bloquear Controles.
- **TabIndex**. Indica el orden de tabulación que ocupa dicho control. El primer control será aquel cuyo TabIndex valga 0. Para saber el orden de tabulación que se seguirá en un formulario, pinchar en **Ver → Orden de Tabulación**. 

- **TabStop**. Indica si el usuario puede utilizar la tecla del Tabulador para poder posicionarse sobre el botón.
- **Text**. Cadena con el título del botón. Lo que el usuario ve.
- **TextAlign**. Alineación o disposición del título dentro del área del botón; por defecto aparece centrado.
- **Visible**. Muestra el botón o no en tiempo de ejecución.

Ejercicio – Muestra el siguiente formulario en el que según la medida de los lados A, B y C introducida por el usuario, nos debe decir si el triángulo es equilátero, isósceles o escaleno.

4.2. CheckBox – Casilla de Verificación – chk

 CheckBox

Este control muestra una casilla de verificación, que podemos marcar para establecer un estado.

Generalmente el estado de un CheckBox es marcado (verdadero) o desmarcado (falso), sin embargo, podemos configurar el control para que sea detectado un tercer estado, que se denomina indeterminado, en el cual, el control se muestra con la marca en la casilla pero en un color de tono gris.

Las propiedades remarcables de este control son las siguientes.

- **Appearance**. Indica la forma del control, si Normal, como una típica casilla de verificación, o como Button, es decir, como un botón que si está seleccionado se queda como presionado, y si no, se queda, se queda realzado.
- **Autocheck**. La casilla cambia automáticamente de estado cuando se selecciona
- **CheckAlign**. Permite establecer de modo visual la ubicación de la casilla de verificación dentro del área del control.
- **Checked**. Valor lógico que devuelve True cuando la casilla está marcada, y False cuando está desmarcada.
- **CheckState**. Valor del tipo enumerado CheckState, que indica el estado del control. Checked, marcado; Unchecked, desmarcado; e Indeterminate, indeterminado.
- **ThreeState**. Por defecto, un control de este tipo sólo tiene dos estados, pero asignando True a esta propiedad, conseguimos que sea un control de tres estados.

4.3. ChekedListBox – Listas de verificación – chkl

 CheckedListBox

Muestra un objeto **ListBox** en el que se muestra una casilla de verificación a la izquierda de cada elemento. Su utilización es muy similar a este control, por ello lo veremos junto a él.

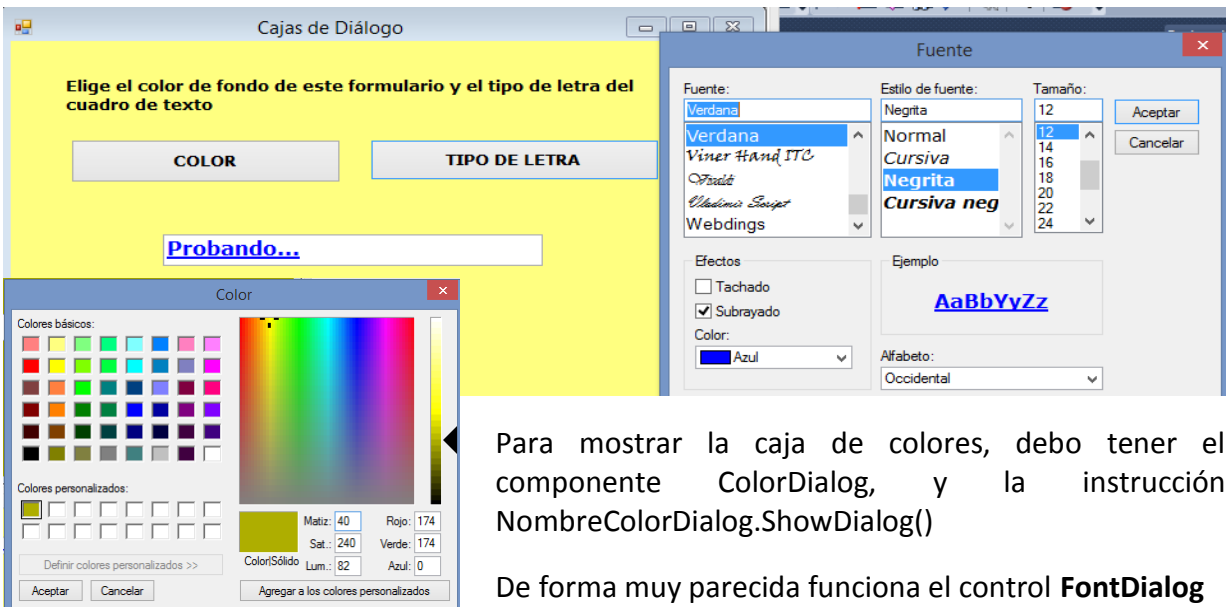
4.4. ColorDialog – Caja de Colores – Colores

 ColorDialog

Representa un cuadro de diálogo común que muestra los colores disponibles, así como los controles que permiten a los usuarios definir colores personalizados.

La propiedad a destacar sería la de **FullOpen** que controla si inicialmente muestra la sección de definir colores personalizados o no.

Ejercicio - Creamos un formulario en el que el usuario pueda elegir el color de fondo del formulario y el tipo de letra y color del cuadro de texto. Muestra tanto la Caja de Colores (ColorDialog) como la Caja de Fuente (FontDialog) para ello.



Para mostrar la caja de colores, debo tener el componente ColorDialog, y la instrucción NombreColorDialog.ShowDialog()

De forma muy parecida funciona el control **FontDialog**

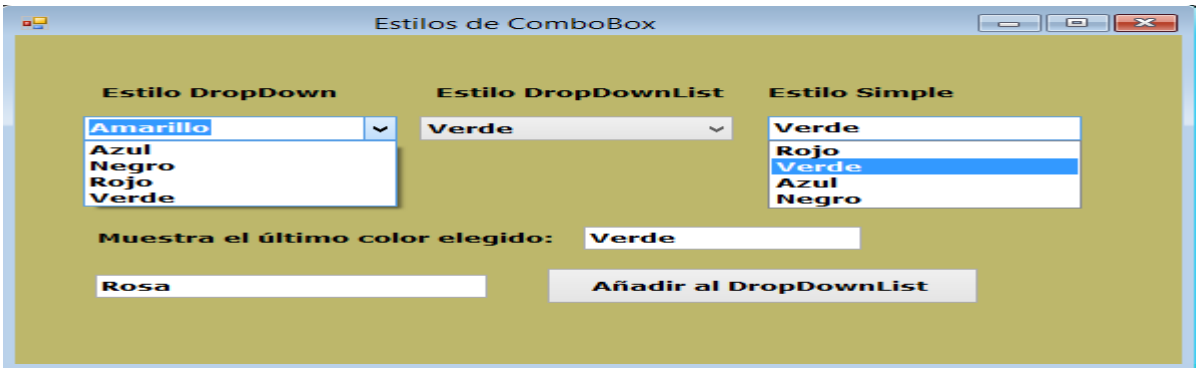
4.5. ComboBox – Lista desplegable – cmbb –

El ComboBox es un control basado en la combinación (de ahí su nombre) de dos controles el TextBox y ListBox.

Un control ComboBox dispone de una zona de edición de texto y una lista de valores, que podemos desplegar desde el cuadro de edición.

El estilo de visualización por defecto de este control, muestra el cuadro de texto y la lista oculta, aunque mediante la propiedad **DropDownStyle** podemos cambiar dicho estilo.

Ejercicio: Crea el siguiente formulario con tres ComboBox, cada uno con "estilo" distinto, de forma que cuando elijas un color de la lista este aparezca en el TextBox ColorElegido y podamos añadir nuevos colores al DropDownStyle a través del otro textBox.



La propiedad DropDownStyle también influye en una diferencia importante de comportamiento entre el estilo **DropDownList** y los demás, dado que cuando creamos un ComboBox con el mencionado estilo, el cuadro de texto sólo podrá mostrar información, no permitiendo que esta sea modificada.

En el caso de que la lista desplegable sea muy grande, mediante la propiedad **MaxDropDownItems**, asignaremos el número de elementos máximo que mostrará la lista del control.

El resto de propiedades y métodos son comunes con los controles TextBox y ListBox..

4.6. DateTimePicker – Calendario – dtp

El control **DateTimePicker** se utiliza para permitir al usuario seleccionar una fecha y una hora, y para mostrar esa fecha y esa hora en el formato especificado. Se pueden limitar las fechas y las horas que se pueden seleccionar al establecer las propiedades **MinDate** y **MaxDate**.

Para cambiar la presentación de la parte del control que corresponde al calendario, establezca las propiedades **CalendarForeColor**, **CalendarFont**, **CalendarTitleBackColor**, **CalendarTitleForeColor**, **CalendarTrailingForeColor** y **CalendarMonthBackground**.

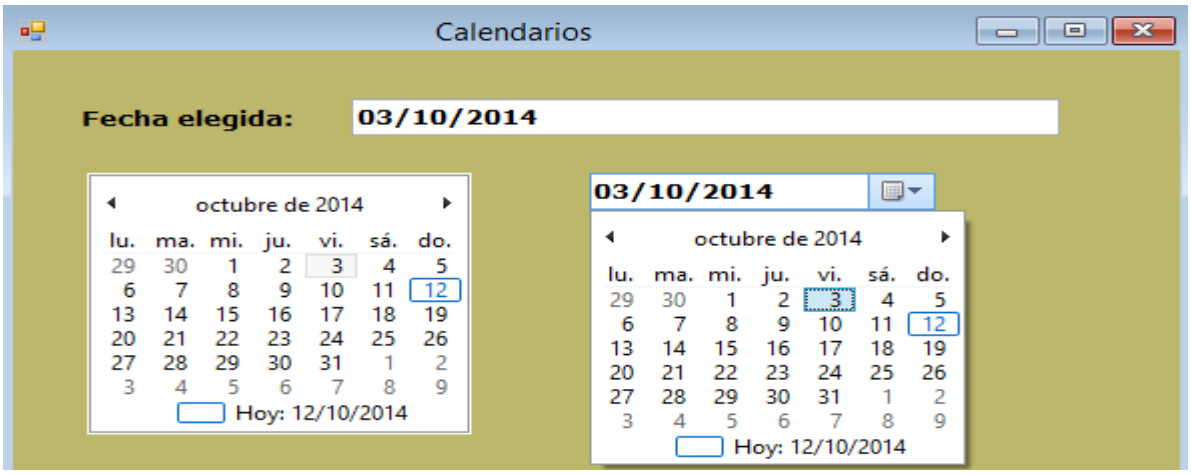
La propiedad **Format** establece el formato en el que se mostrará la fecha y hora del sistema. Se puede crear un estilo de formato propio, eligiendo en Format → Custom y en la propiedad **CustomFormat** indicando la cadena de formato personalizado. La cadena de formato personalizado puede ser una combinación de caracteres de campos personalizados y de otros caracteres literales. Por ejemplo, se puede presentar la fecha como "June 01, 2001 - Friday" al establecer la propiedad **CustomFormat** en "MMMM dd, yyyy - dddd". Para obtener más información, vea Cadenas de formato de fecha y hora.

Si desea utilizar un control de estilo de flechas para ajustar el valor de fecha y hora, establezca la propiedad **ShowUpDown** en **true**. El control de calendario no se desplegará cuando se seleccione el control. Para ajustar la fecha y la hora, se puede seleccionar cada elemento por separado y utilizar los botones Arriba y Abajo para cambiar el valor.

Este control guarda una estrecha semejanza al control **MonthCalendar** (Ver)

*Ejercicio: Creamos el siguiente formulario, en el cual cada vez que modifiquemos la fecha en el **DateTimePicker**, en el **MonthCalendar** aparecerá el mes de la fecha elegida y el día señalado. Y de igual forma, cada vez que seleccionemos un día en el MonthCalendar, aparecerá dicho día en el DateTimePicker*

Se presupone que se puede configurar los colores de fondo, tipo de letra, seleccionar el día de hoy, los de fiesta, poner en negrita, etc… pero como podéis ver en el ejercicio siguiente, luego no muestra la configuración prefijada



Para ello solo es necesaria la instrucción:

```
this.mesCalendar.SelectionStart = this.dtpFecha.Value;
```

O

```
this.txtFecha.Text = this.mesCalendar.SelectionStart.ToShortDateString();
```

4.7. DomainUpDown – DominioTexto - dud -

Obtiene o establece el texto que se muestra en el control de flechas, y del cual podrá seleccionar un elemento el usuario.

Destacamos las siguientes propiedades:

- **Ítems.** Colección de elementos que por defecto aparecerán
  - Para añadir nuevos elementos, con dudN.items.add (“cadena”)
- **Sorted.** Controla si los elementos de la lista están ordenados
- **Wrap.** Indica si crea una lista circular (del último se pasa al primero) o no.

4.8. ErrorProvider – Error

ErrorProvider

Proporciona una interfaz de usuario para indicar que un control de un formulario tiene un error asociado.

Propiedades a destacar:

- **BlinkRate.** Velocidad en milisegundos a la que parpadea el icono
- **BlinkStyle.** Establece cuándo parpadea el icono
- El método que lo hace aparecer y desaparecer es **SetError**
  - Se le indica el control al que se le asocia
  - El mensaje que quieras. Para quitarlo, el mensaje debe ser “”

4.9. FontDialog – FormatoFuente – fuente

FontDialog

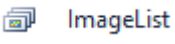
Es muy similar al ColorDialog pero lo que muestra es la ventana del formato fuente. Por lo demás, todo igual.

4.10. GroupBox – Cajas de Grupo – grp

Este control nos permite, como indica su nombre, agrupar controles en su interior, tanto RadioButton como de otro tipo, ya que se trata de un control contenedor.

Una vez dibujado un GroupBox sobre un formulario, podemos arrastrar y soltar sobre él, controles ya existentes en el formulario, o crear nuevos controles dentro de dicho control. De esta forma, podremos como aislar un grupo de controles de otro. Por ejemplo, con los RadioButton, si están todos en el mismo contenedor, ejemplo el formulario, sólo se podría elegir uno de todos ellos, en cambio, si los agrupamos en distintos grupos, de cada grupo podríamos elegir uno.

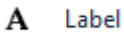
4.11. ImageList – Lista de Imágenes – imgl



Proporciona métodos para administrar una colección de objetos Image

Normalmente, la clase ImageList la utilizan otros controles, como **ListView**, **TreeView** o **ToolBar**. Se pueden agregar mapas de bits, iconos o metarchivos a ImageList y los demás controles podrán utilizar las imágenes que necesiten.

4.12. Label – Etiquetas – lbl



El control Label o Etiqueta, muestra un texto informativo al usuario. Podemos utilizar este control como complemento a otro control, por ejemplo, situándolo junto a un TextBox, de modo que indiquemos al usuario el tipo de dato que esperamos que introduzca en la caja de texto. Existen controles, como el ChekBox que ya lo llevan incorporado.

Se trata de un *control estático*; esto quiere decir que el usuario no puede interaccionar con él, a diferencia, por ejemplo, de un control Button, sobre el que sí podemos actuar pulsándolo; o de un TextBox, en el que podemos escribir texto.

Destacamos las siguientes propiedades:

- **Autosize:** A False si queremos que el contenido de la etiqueta ocupe varias líneas
- **BorderSyle.** permite definir un borde o recuadro alrededor del control, o que dicho borde tenga un efecto 3D; por defecto se muestra sin borde.
- **UseMnemonic.** subraya la letra posterior a & que se haya escrito en Text, para así poder acceder directamente a la etiqueta pulsando alt + letra subrayada

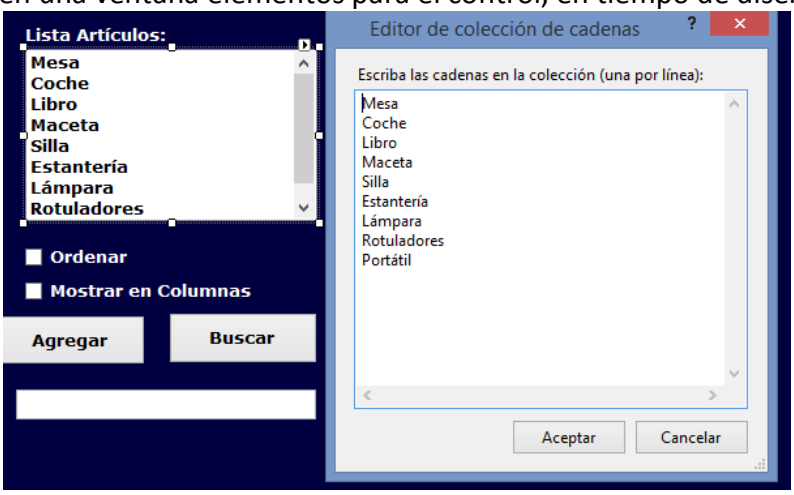
4.13. ListBox – Cuadros de lista – lst

Un control ListBox contiene una lista de valores, de los cuales, el usuario puede seleccionar uno o varios simultáneamente. Entre las principales propiedades de este control, podemos resaltar las siguientes:

- **ColumnWidth**. Indica el ancho que tiene que tener cada columna cuando la propiedad MultiColumn está a true.

- **HorizontalExtends**. Indica en pixels hasta qué anchura me permite tener en el cuadro de lista, siempre y cuando tenga a true la siguiente propiedad. Tengo que tener en cuenta el tamaño del cuadro de vista en **Size (x)**
- **HorizontalScrollBar.** Muestro una barra de desplazamiento horizontal si existen elementos que se salgan por el borde derecho, si la anterior propiedad está a 0, o la muestra siempre hasta simular un cuadro del tamaño indicado en la propiedad HorizontalExtends
- **IntegralHeight**. Los valores de la lista son mostrados al completo cuando esta propiedad contiene True. Sin embargo, al asignar el valor False, según el tamaño del control, puede que el último valor de la lista se visualiza sólo en parte. La siguiente imagen muestra un ListBox con esta propiedad a False.

- **Items**. Contiene la lista de valores que visualiza el control. Se trata de un tipo **ListBox**.**ObjectCollection**, de manera que el contenido de la lista puede ser tanto tipos carácter, como numéricos y objetos de distintas clases. Al seleccionar esta propiedad en la ventana de propiedades del control, y pulsar el botón que contiene, podemos introducir en una ventana elementos para el control, en tiempo de diseño:



- **MultiColumn**. Visualiza el contenido de la lista en una o varias columnas en función de si asignamos False o True respectivamente a esta propiedad.
- **ScrollAlwaysVisible**. Indica si debe aparecer siempre una barra de desplazamiento vertical, o solo cuando sea necesaria.
- **SelectionMode**. Establece el modo en el que vamos a poder seleccionar los elementos de la lista. Si esta propiedad contiene **None**, no se realizará selección; **One**, permite seleccionar los valores uno a uno; **MultiSimple** permite seleccionar múltiples valores de la lista pero debemos seleccionarlos independientemente; por último, **MultiExtended** nos posibilita la selección múltiple, con la ventaja de que podemos hacer clic en un valor, y arrastrar, seleccionando en la misma operación varios elementos de la lista.

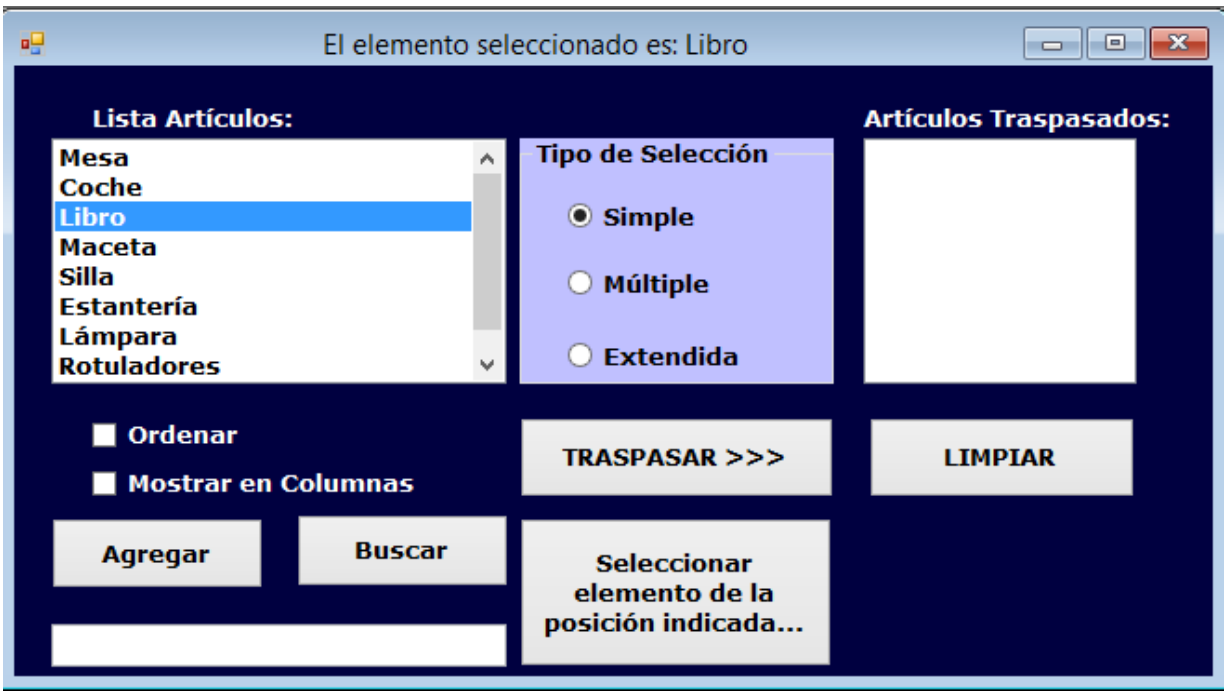
- **Sorted**. Cuando esta propiedad contiene el valor True, ordena el contenido de la lista. Cuando contiene False, los elementos que hubiera previamente ordenados, permanecen con dicho orden, mientras que los nuevos no serán ordenados.

Para el código será muy importante tener los siguientes métodos en cuenta:

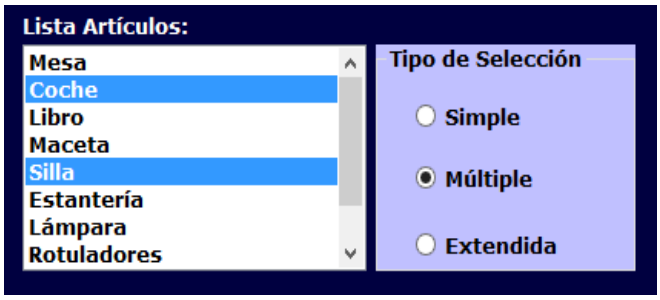
- **ítems.count.** Me indica cuántos elementos tiene la caja de lista.
- **ítems(index).** Indica el dato del elemento que ocupa la posición index.
- **ítems.Add**. Me añade un elemento a mi lista de elementos
- **ítems.Clear.** Elimina todos los elementos de la lista
- **ítems.RemoveAt(index)** Elimina el elemento que ocupa la posición index.
- **SelectedItem**. Devuelve el elemento de la lista actualmente seleccionado.
- **Selecteditems**. Devuelve una colección ListBox.SelectedObjectCollection, que contiene los elementos de la lista que han sido seleccionados.
- **SelectedIndex**. Informa del elemento de la lista seleccionado, a través del índice de la colección que contiene los elementos del ListBox. Para mostrar algunas de las funcionalidades de este control, utilizaremos el proyecto de ejemplo:

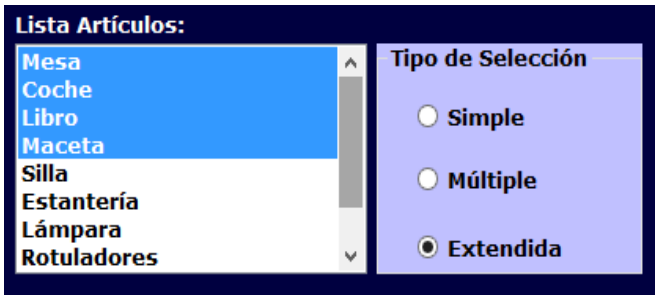
*Ejercicio: llamado **CajaList.** Consiste en un formulario que contiene un ListBox principal, con el nombre **lstArticulos**, que dispone de una serie de valores asignados en tiempo de diseño.*

Cada vez que hacemos clic en alguno de los valores, se produce el evento **SelectedIndexChanged**, que utilizamos para mostrar en este caso, el nombre del elemento en el título del formulario.

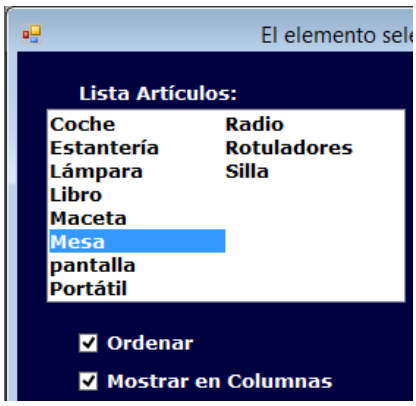


Como vemos, por defecto solo podremos seleccionar un elemento de la lista. En caso de querer cambiar el tipo de selección, lo haremos a través de los botones de opción:





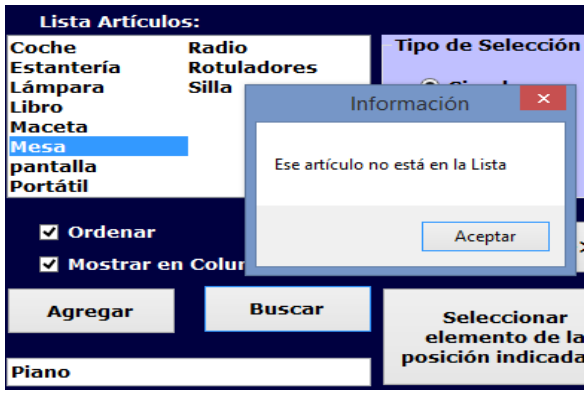
Las casillas de verificación:



- **Ordenar**: Ordena la lista de valores, de forma que no solo se ordenan los elementos que ya estén, sino también si se añade alguno nuevo.
- **Mostrar en columnas**: Te muestra en dos columnas los datos que tengamos. Es conveniente que en la propiedad ColumnWidth hayamos establecido una columna mínima (p.ej. 50)

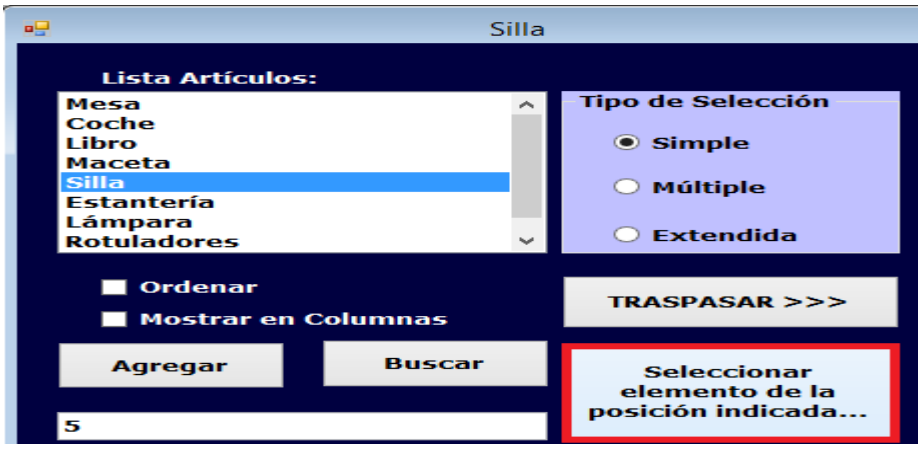
El botón **Agregar**, añadirá el elemento colocado en el cuadro de texto (**txtArticulo**) a la lista de Artículos.

El botón **Buscar**, me indicará, mediante una ventana de diálogo, si el dato escrito en el cuadro de texto está en la lista de Artículos.



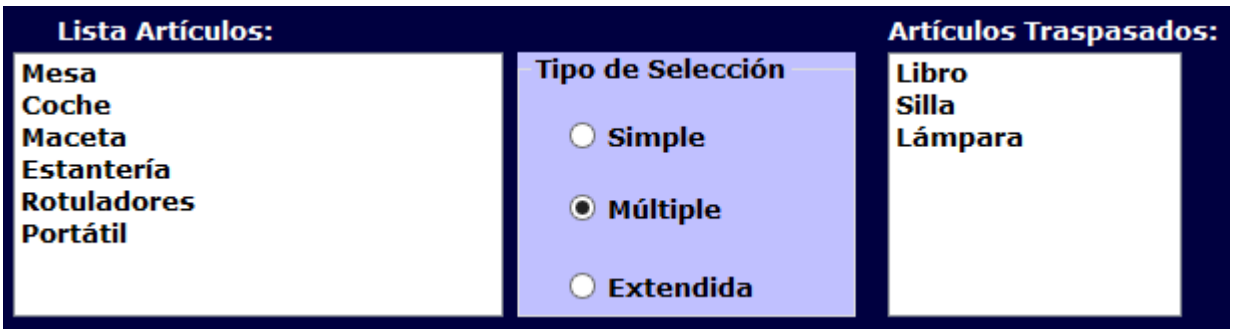

```
private void btnBuscar_Click(object sender, EventArgs e)
{
    if (this.txtArticulo.Text.Equals(string.Empty))
        MessageBox.Show("Debes escribir un artículo", "Error", MessageBoxButtons.OK);
    else
    {
        int i;
        i = this.lstArticulos.FindStringExact(this.txtArticulo.Text);

        if (i == -1) MessageBox.Show("Ese artículo no está en la Lista", "Información", MessageBoxButtons.OK);
        else    // Lo seleccionamos
            this.lstArticulos.SelectedIndex = i;
    }
}
```

Para seleccionar valores de la lista mediante el código del programa. Al pulsar el botón **btnSeleccionar**, podemos utilizar el método **SetSelected( )** o el **SelectedIndex** para realizar esta tarea. En ambos pasamos el índice de la lista con el que queremos seleccionar, recordando que es un vector y el primer elemento ocupará la posición 0 y el último Count-1. Con SetSelected podemos poner y quitar la selección, asignando el valor True para o False.




```
private void btnSelecPos_Click(object sender, EventArgs e)
{
    if (this.txtArticulo.Text.Equals(string.Empty))
        MessageBox.Show("Debes escribir un artículo", "Error", MessageBoxButtons.OK);
    else
    {
        int i;
        i = Convert.ToInt32(this.txtArticulo.Text);
        if (i > 0 && i < this.lstArticulos.Items.Count)
            // this.lstArticulos.SelectedIndex = i - 1;
            this.lstArticulos.SetSelected(i - 1, true);
    }
}
```

Con el botón **Traspasar**, pasaremos al **lstTrasps** los elementos seleccionados del lstArticulos. Usaremos para tomar los elementos seleccionados de lstArticulos, y pasarlos al otro ListBox del formulario. La propiedad **SelectedItems** del control lstArticulos, devuelve una colección con sus elementos seleccionados y



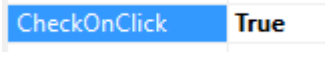

```
private void btnTraspasar_Click(object sender, EventArgs e)
{
    while(this.lstArticulos.SelectedItems.Count>0)
    {
        // Añadimos un elemento seleccionado a la Lista de Traspasados
        this.lstTrasps.Items.Add(this.lstArticulos.SelectedItems[0]);
        // Una vez traspasado lo ELIMINAMOS de la lista de Artículos
        this.lstArticulos.Items.Remove(this.lstArticulos.SelectedItems[0]);
    }
}
```

Por último, el botón **Limpiar**, me deja en blanco la lista de valores seleccionados.

Añade una nueva parte al formulario de forma que utilicemos de forma similar un **ChekedListBox**. Como puedes ver en la siguiente imagen, se crea un chekedlistbox que desempeña la misma función que el listbox de la izquierda, el lstArticulos.

Crea otros dos botones **Traspasar** y **Agregar** (los de la derecha) que realicen la misma función anterior pero con respecto al chekedListBox, y no con el ListBox.

Para ello tendrás en cuenta las mismas propiedades que para ListBox. Y lo único que deberás cambiar es en el código es coger los elementos `this.lstchkFrutas.CheckedItems`

Recuerda poner la propiedad 

para que al primer click ya se chequee el elemento.

4.14. MonthCalendar – Calendario – Mes

Representa un control de calendario mensual estándar de Windows. Mantiene un estrecho paralelismo con el DateTimePicker

Las propiedades que destacamos son:

- **SelectionRange.** Donde se establece el rango de fecha inicial y final para elegir.
- **ShowToday** y **ShowTodayCircle.** Para que aparezca la fecha del día, y señalada en el mes con un círculo rojo.
- El método **SetDate**, para seleccionar una fecha
- El método **SelectionStart** para obtener el elemento seleccionado por el usuario

4.15. NumericUpDown – Dominios de Nºs – num

Similar al control llamado DomainUpDown, nos ofrece un cuadro de texto con dos flechitas, para poder seleccionar un número.

Las propiedades que tendremos que tener en cuenta son:

- **Minimum y Maximun:** Indican el rango de valores que va a albergar.
- **ThousandsSeparator.** Indica si muestra o no el separador de miles

4.16. Panel – panel – panel

Panel es un control que contiene otros controles. Se puede utilizar **Panel** para agrupar colecciones de controles, como un grupo de controles **RadioButton**. Al igual que sucede con otros controles contenedores, como el control **GroupBox**, si la propiedad **Enabled** del control **Panel** está establecida en **false**, los controles contenidos dentro de **Panel** también se deshabilitarán.

El control **Panel** se muestra de forma predeterminada sin bordes. Para proporcionar un borde estándar o tridimensional, se utiliza la propiedad **BorderStyle** para distinguir el área del panel de otras áreas del formulario.

Se puede utilizar la propiedad **AutoScroll** para habilitar barras de desplazamiento en el control **Panel**. Cuando la propiedad **AutoScroll** está establecida en **true**, es posible desplazarse a cualquier control situado dentro de **Panel**, aunque fuera de su región visible, con las barras de desplazamiento proporcionadas.

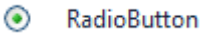
4.17. PictureBox – Dibujo – pic

Representa un control de cuadro de imagen de Windows para mostrar una imagen.

Destacaría las siguientes propiedades:

- **Image.** Indica el fichero imagen que mostraremos
- **SizeMode.** Para indicar cómo quieres ajustar la imagen al tamaño del cuadro

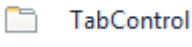
4.18. RadioButton – Botones de Opción – rdb



Los controles RadioButton nos permiten definir conjuntos de opciones autoexcluyentes, de modo que situando varios controles de este tipo en un formulario, sólo podremos tener seleccionado uno en cada ocasión. Por lo demás, muy similares a los CheckBox

Ejercicio – Colores: Como observarás, es muy simple, consiste en modificar el color del cuadro de texto, según la opción escogida, y el formato de Mayúsculas o minúsculas, y de centrado o no, según lo elegido.

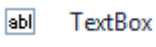
4.19. TabControl – Fichas – tab



Administra un conjunto relacionado de páginas de fichas resentadas por objetos **TabPage**, que se agregan mediante la propiedad **TabPages**

Desde el código nos podremos referir a las propiedades de las páginas o bien directamente con su nombre (Name), o bien como elemento de la colección del TabControl.TabPages(index)

4.20. TextBox – Cuadro de Texto – txt



Un control TextBox muestra un recuadro en el que podemos introducir texto. Para poder escribir texto en un control de este tipo, debemos darle primeramente el foco, lo que detectaremos cuando el control muestre el cursor de escritura en su interior.

El modo de dar a un control el foco de entrada, consiste en hacer clic sobre él, o bien, pulsar la tecla [TAB], pasando el foco hasta el control deseado. Cuando un control recibe el foco, el sistema operativo lo remarca visualmente o en el caso de controles de escritura, muestra el cursor de escritura en su interior. Si lo hacemos a través del código, emplearemos la instrucción:

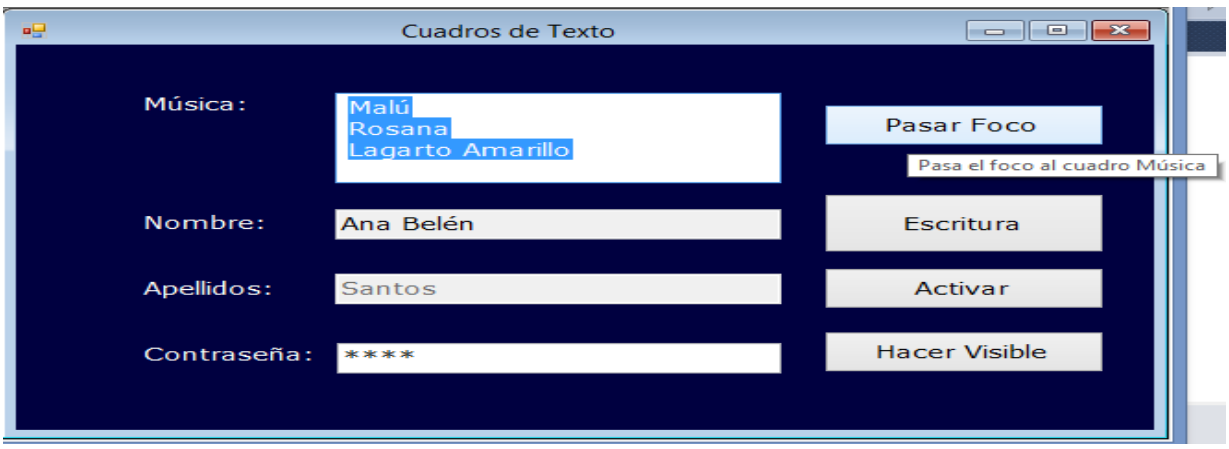
```
this.txtMusica.Focus();   //No con todos los controles funciona, ni en todos los casos
```

Entre las propiedades disponibles por este control, destacaremos las siguientes:

- **AutoSize**. Cuando esta propiedad tenga el valor True, al modificar el tamaño del tipo de letra del control, dicho control se redimensionará automáticamente, ajustando su tamaño al del tipo de letra establecido, para aquellos cuadros de texto que contendrán solo una línea
- **CharacterCasing**. Esta propiedad, permite que el control convierta automáticamente el texto a mayúsculas o minúsculas según lo estamos escribiendo.
- **HideSelecction**. Si está a True, la selección del texto desaparecerá cuando el cuadro de texto pierda el foco.
- **Lines**. Líneas de texto que queremos que aparezcan en un cuadro de texto que permita múltiples líneas. Si no se verán sobrepuestas
- **MaxLength**. Valor numérico que establece el número máximo de caracteres que podremos escribir en el control.
- **Multiline**. Permite establecer si podemos escribir una o varias líneas. Por defecto contiene False, por lo que sólo podemos escribir el texto en una línea.
- **PasswordChar**. Carácter de tipo máscara, que será visualizado por cada carácter que escriba el usuario en el control. De esta forma, podemos dar a un cuadro de texto el estilo de un campo de introducción de contraseña.
- **ReadOnly**. Permite indicar si el contenido del control será de sólo lectura o bien, podremos editarlo.
- **ScrollBars**. Permite para controles de edición de múltiples líneas, indicar qué barras de desplazamiento se muestran por defecto
- **WordWrap**. En controles multilínea, cuando su valor es True, al llegar al final del control cuando estamos escribiendo, realiza un desplazamiento automático del cursor de escritura a la siguiente línea de texto.

Ejercicio: Crea el siguiente formulario en el que como ves hay 4 cuadros de texto, uno con líneas múltiples, otro con los caracteres codificados y otros dos normales. Como ves el botón primero, pasará el foco al cuadro de texto Contraseña, y los otros tres botones actuarán de forma intermitente:

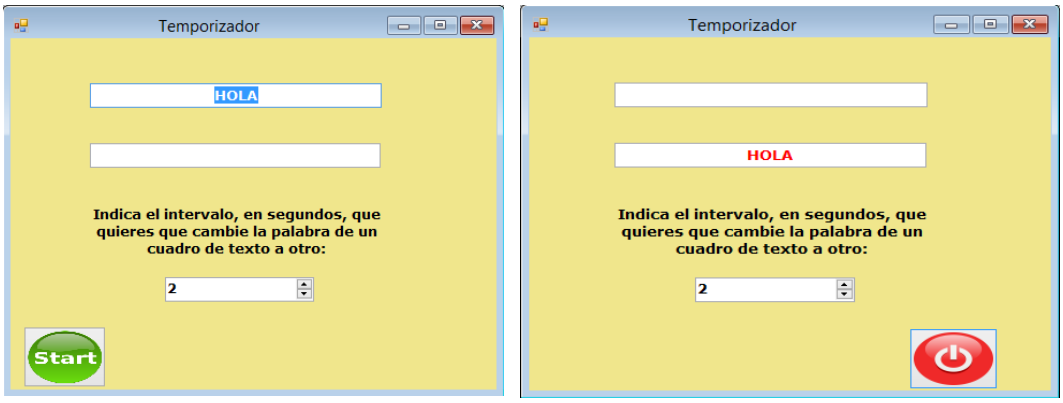
- Uno haciendo de Solo lectura o de Lectura/Escritura el cuadro Nombre
- Otro Activando o Desactivando el cuadro de Apellidos
- Otro haciendo Visible u Ocultando el cuadro de texto contraseña



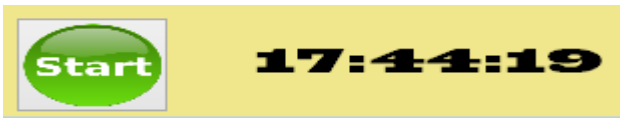
4.21. Timer – Temporizador – Rlj - Timer

El control Timer nos permite la ejecución de código a intervalos de tiempo predeterminados.

*Ejercicio: En el formulario ejemplo **Temporizador**, que consiste en traspasar a intervalos de tiempo, en segundos, indicados por el usuario a través del NumericUpDown (mínimo 1 y máximo 10), el contenido de un TextBox a otro de **solo lectura**. El usuario comenzará a traspasar el texto introducido (debes comprobar que existe texto en alguno de los cuadros de texto) en el momento que haga clic en Start, entonces se pondrá de solo lectura el NumericUpDown utilizado, se hará visible el botón Stop y desaparecerá el botón Start.*



Ejercicio – Añade que un cronómetro que muestre la hora del sistema



4.22. ToolTip – etiquetas – tt ToolTip

El control **ToolTip** permite proporcionar ayuda a los usuarios cuando sitúan el cursor del mouse sobre un control. Normalmente se utiliza para notificar a los usuarios el uso al que está destinado un control. Por ejemplo, para un control TextBox que acepta un nombre, podría especificar un texto de información sobre herramientas que indique el formato del nombre que se debe escribir en el control. Además de para proporcionar ayuda, se puede utilizar para proporcionar información de estado en tiempo de ejecución.

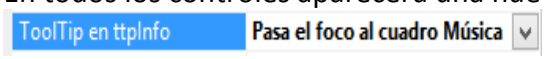
Las propiedades que destacan son:

- **AutomaticDelay.** Establece los valores apropiados de temporización de la etiqueta, cuándo se muestra, durante cuánto tiempo y el tiempo pasado desde que desaparece una etiqueta y sale otra.
- **ShowAlways.** Indica si debe aparecer aunque el control o el formulario esté inactivo.
- **isBallon:** El mensaje en una viñeta.

Para asociar un texto de información a un control se utiliza el método SetTooltip, y lo puedes utilizar con tantos controles como quieras. Generalmente se establece en el procedimiento Form_Load.

Para poner el mensaje que queremos mostrar tenemos dos opciones:

- En todos los controles aparecerá una nueva propiedad:

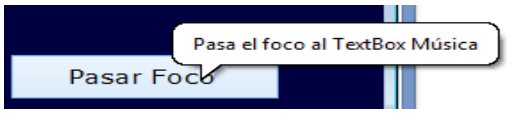


en ella puedes poner el mensaje.

- O bien desde código:

```
this.ttpInfo.SetToolTip(this.btnFoco, "Pasa el foco al TextBox Música");
```

Podría quedar algo así:



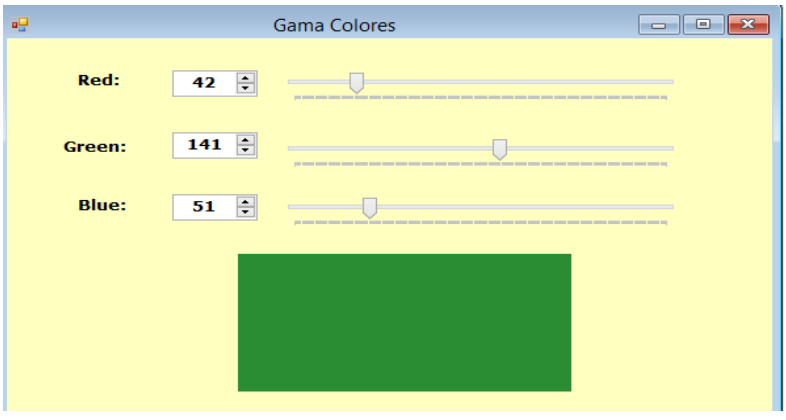
4.23. TrackBar – Barra Numérica – trk

TrackBar es un control desplazable similar al control **ScrollBar**. Para configurar los intervalos por los que se desplaza el valor de la propiedad **Value** de una barra de seguimiento, establezca la propiedad **Minimum** para especificar el extremo inferior del intervalo y la propiedad **Maximum** para especificar el extremo superior del intervalo.

La propiedad **LargeChange** define el incremento que se debe sumar o restar de la propiedad **Value** al hacer clic en uno de los lados del control deslizante, o mediante las teclas tabuladota o retroceso o avance de página. La barra de seguimiento se puede mostrar horizontal o verticalmente, mediante su propiedad **Orientación**.

También puedes modificar donde quieres que aparezcan los pasos o marcas, con la propiedad **TickStyle**.

Este control se puede utilizar para entrar los datos numéricos obtenidos mediante la propiedad **Value**. Estos datos numéricos se pueden mostrar en un control o se pueden utilizar en el código.



Ejercicio – Gama de Colores

Como ves vamos a crear el siguiente formulario de forma que modificando los valores de Rojo, Verde y Azul consigamos el código RGB y su color correspondiente.

Con la siguiente instrucción yo conseguiría el color de pantalla

```
Color miColor = Color.FromArgb(42,141,51);
```

Y quedarían más controles que ya iremos viendo con los ejercicios….