

Bloque 2. Visual Basic .NET

UT 3. Confección de interfaces de usuario y creación de componentes visuales

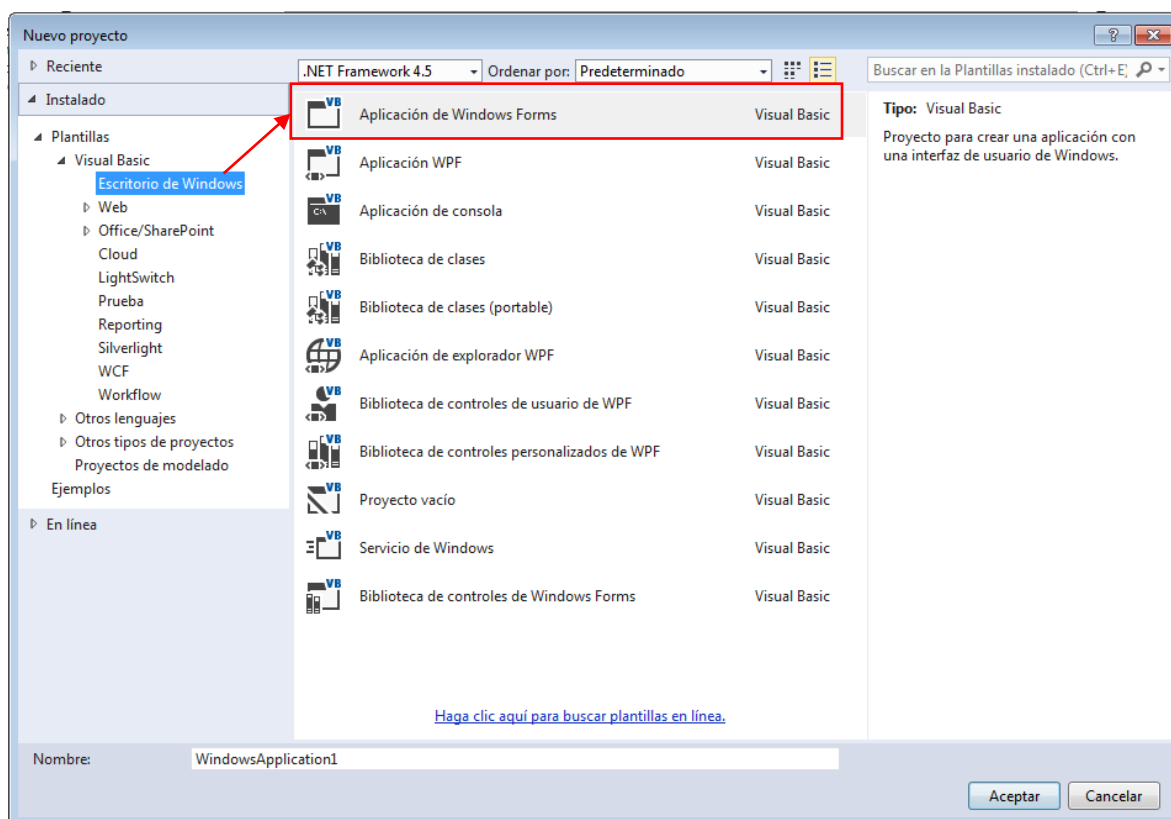
TEMA 2

El entorno de desarrollo

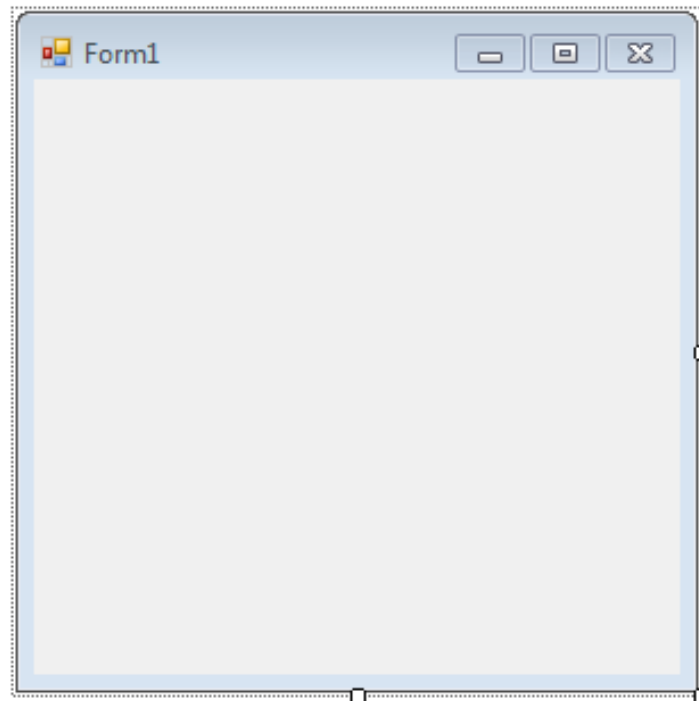
1. El IDE o Entorno Integrado de Desarrollo

Lo primero que tenemos que aprender antes de meternos de lleno con la programación es aprender a manejarnos con el entorno de desarrollo (IDE) de Visual Studio .NET. Como comentamos en el primer tema, Visual Studio es ya un único entorno para todo lo referente al desarrollo con Microsoft y su .NET. Por tanto, si además de usar el VB.NET queremos pasar a otros lenguajes como el C# o ASP.NET, no tendremos que utilizar otro programa. Incluso podremos trabajar con varios proyectos en el mismo IDE, aunque sean de lenguajes diferentes.

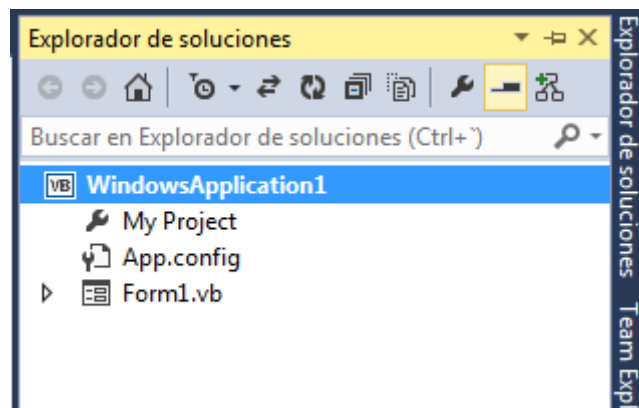
Lo primero que vamos a hacer es cargar el entorno de desarrollo del Visual Studio .NET, así que ejecutamos el programa y creamos un nuevo proyecto de tipo “Aplicación para Windows”:



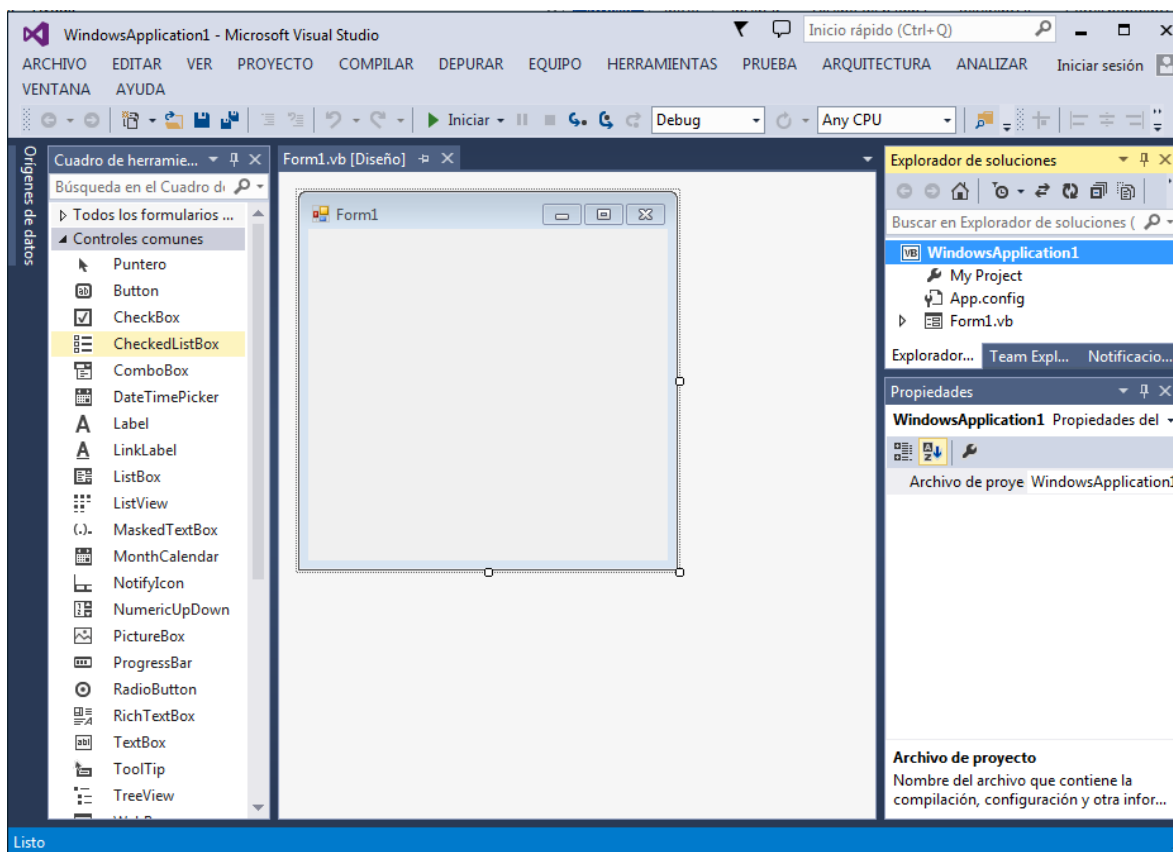
En el proyecto, se habrá creado un formulario que se mostrará en la pantalla central de forma automática.



En el lado derecho de la pantalla, verás que hay un "panel" o ventana en la que se indica el proyecto actual y se muestran los ficheros que lo componen. Ese panel es el **Explorador de Soluciones**:



Una vez cargado el formulario, la pantalla tendrá un aspecto similar a este:

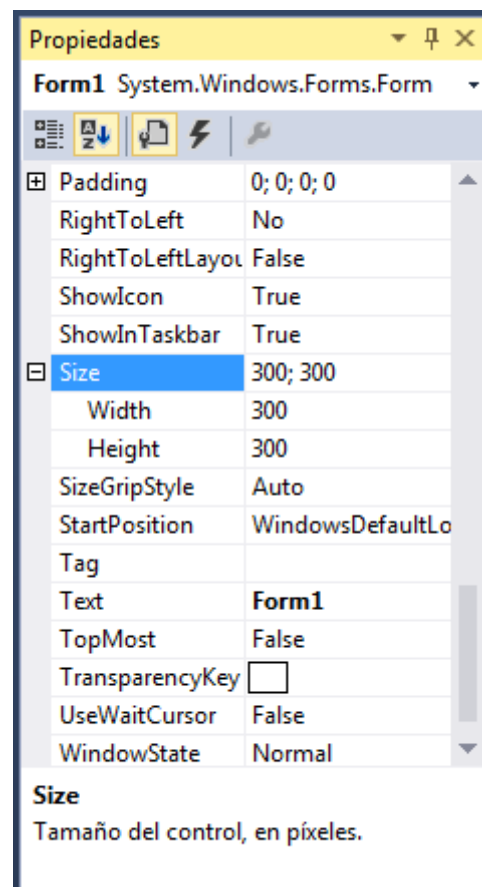


Donde distinguimos en la parte izquierda el cuadro de herramientas, en el centro el formulario principal y en la parte derecha las ventanas de propiedades y el explorador de soluciones.

1.1. Cambiar propiedades del IDE

El tamaño del formulario que se ha insertado por defecto es de 300 x 300 píxeles. Vamos a cambiarlo a 500 x 500.

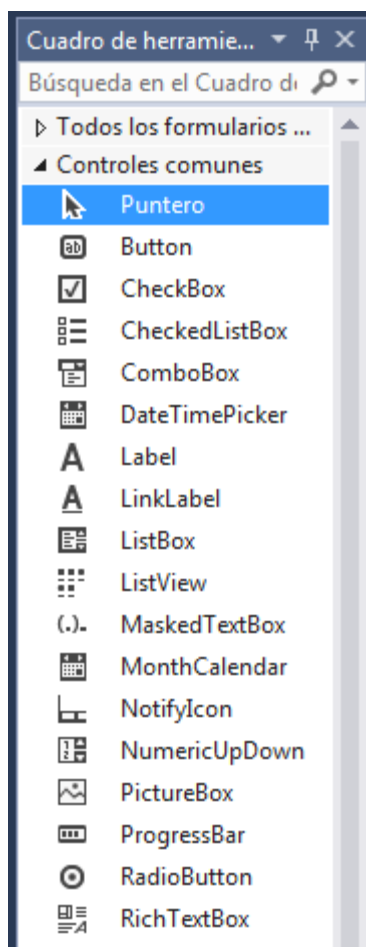
Seleccionamos el formulario haciendo clic en él y vamos a ver sus propiedades. En el panel de la derecha, debajo del explorador de soluciones, está la ventana de propiedades del elemento que actualmente esté seleccionado, en nuestro caso son las propiedades del Form1. Vamos a buscar el elemento **Size** para poder cambiar el tamaño del mismo. Se muestran dos valores separados por punto y coma, pero también hay una cruz a la izquierda de la palabra Size. Si pulsas en esa cruz, se mostrarán los tamaños a lo que se refiere esos valores separados por punto y coma.



Posiciónate en ellos y asígnales el valor 500. De esta forma, tendremos un formulario más grande, en el que será más fácil posicionar los controles que queramos añadir al formulario en cuestión. Además hemos visto debajo de las propiedades que cuando seleccionamos una nos aparece una descripción de ella: Size, determina el tamaño del control en píxeles. Esto será de gran ayuda para más adelante. Asimismo nos encontramos con algo importante: las propiedades. En este caso los formularios tienen todas las propiedades que aparecen en esta ventana y que hacen referencia a su aspecto y comportamiento. En nuestro caso hemos modificado la propiedad Size para cambiar el tamaño del formulario.

Fíjate que después de haber cambiado esos valores, los mismos se muestran en "negrita", indicándonos de esta forma que son valores que nosotros hemos asignado, distintos a los que tiene por defecto. Si el tamaño del formulario no es el adecuado para tu gusto puedes cambiarlo y dejarlo como estaba, lo importante es que ya hemos cambiado una propiedad del formulario.

1.2 Insertar controles



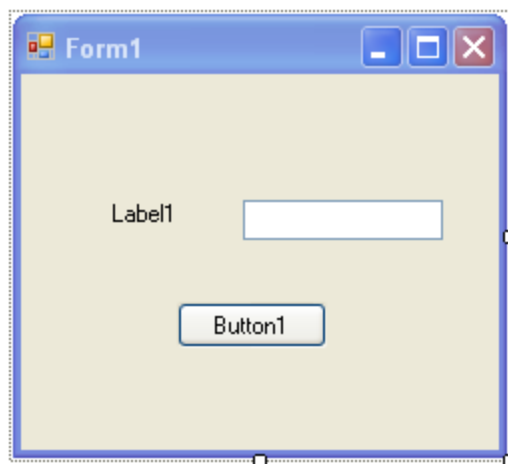
Los **controles** son los elementos que insertamos dentro de un formulario y que nos va a permitir interactuar entre el usuario y el código. Controles son: botones, cuadros de texto, etiquetas, cuadros desplegables, cuadrículas de datos,... En definitiva, todos y cada uno de los elementos que vemos en los formularios de todas las aplicaciones. La lista de controles básicos disponibles la tenemos a la izquierda:

Para añadir controles al formulario utilizaremos esta barra. Por ejemplo, para añadir una etiqueta (Label) y una caja de texto (TextBox), simplemente haremos doble-clic sobre esos elementos de la barra de herramientas y se añadirán al formulario.

Tenemos varias formas de añadir estos controles:

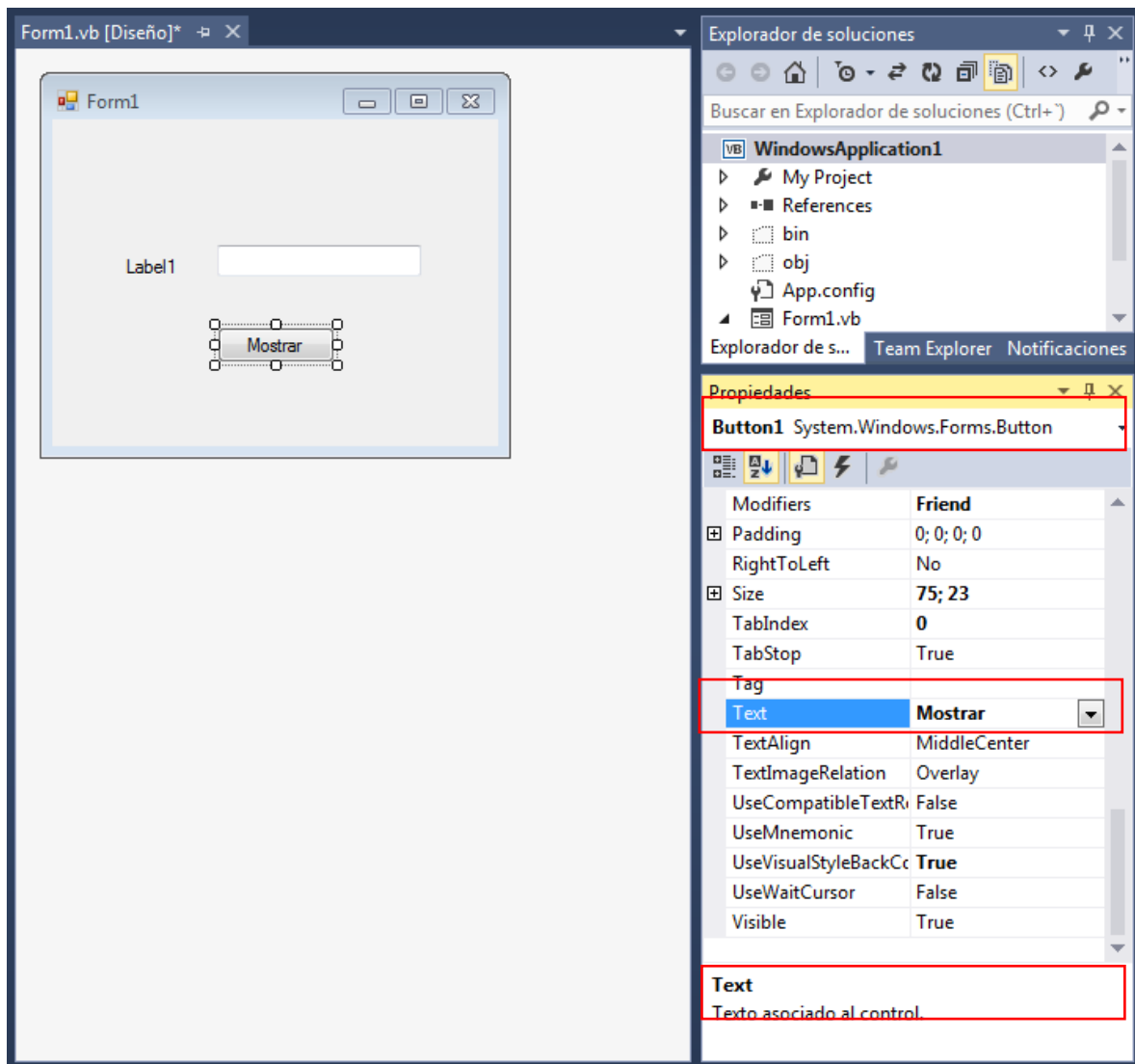
- Haciendo doble clic sobre el icono del control. Se colocará un control en la parte izquierda superior del formulario.
- Haciendo clic sobre el icono del control y después clic sobre el formulario. El control se insertará en el punto que hemos indicado y se extenderá hacia la derecha y abajo según su tamaño predeterminado.
- La tercera forma es la más utilizada. Hacemos clic sobre el control y luego en el formulario. Pero esta vez sin soltar el botón izquierdo, vamos dibujando con el ratón la posición y tamaño del control.
- Arrastrando desde la barra de controles.

Para nuestro ejemplo añadiremos un botón (Button) y lo situaremos debajo del cuadro de texto (Textbox). Luego añadimos una etiqueta (Label) para que quede de esta forma:

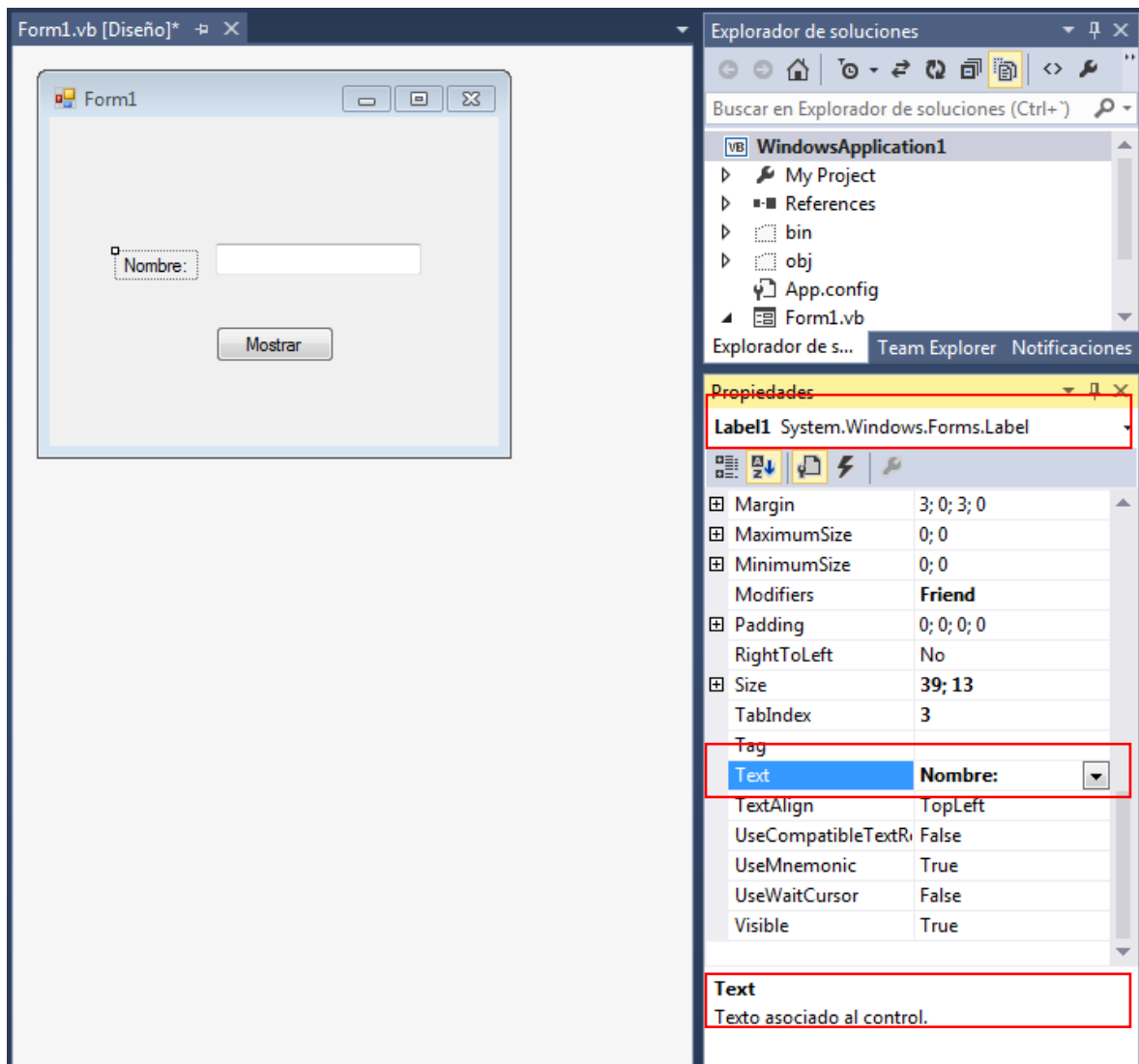


Como vemos por defecto el IDE pone unos nombres genéricos a los controles: label1, textbox1, button1, es decir, utiliza el tipo de control y lo va numerando: label1, label2, label3,... Todos los controles deben tener un nombre único, así que hasta que se los asignemos el IDE le pone este genérico. En nuestros proyectos cambiaremos esos nombres de controles por otros nombres más descriptivos, por ejemplo: txt_nombre, txt_apellidos, txt_telefono en lugar de los puestos por el IDE: textbox1, textbox2, textbox3,...

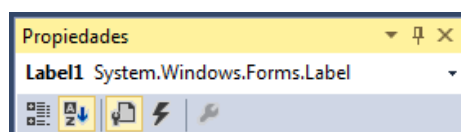
Ahora vamos a cambiar el texto que contiene el botón "Button1". Para cambiarle este texto hay que utilizar la ventana de propiedades, en esta ocasión el elemento que nos interesa de esa ventana de propiedades es **Text**, escribimos en esta propiedad la palabra "Mostrar" y cuando pulses Intro o el tabulador veremos que el texto del botón se ha actualizado:



Hacemos lo mismo con la etiqueta. Recuerda que hay que seleccionarla primero haciendo clic para que se muestren las propiedades de la etiqueta, escribimos "**Nombre:**" y pulsamos intro o el tabulador:



Debemos fijarnos siempre en la parte superior de la ventana para comprobar que pone el nombre del control que estamos editando y de qué tipo es:

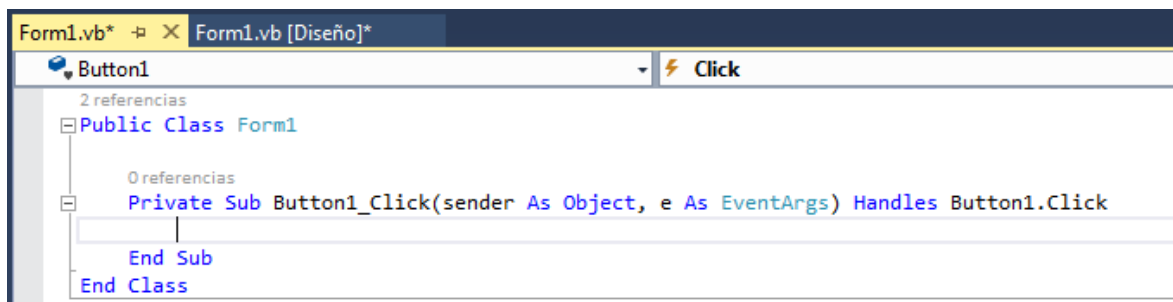


En este caso es una etiqueta (label) que la ha creado a partir de la clase "System.Windows.Forms.Label"

Al principio y para asegurarnos de que vamos escribiendo bien estas propiedades nos fijaremos bien en que tenemos seleccionado el control adecuado. En la pantalla anterior podemos ver en el título superior el nombre del control: "Label1" así sabemos que estamos escribiendo la propiedad en el control adecuado.

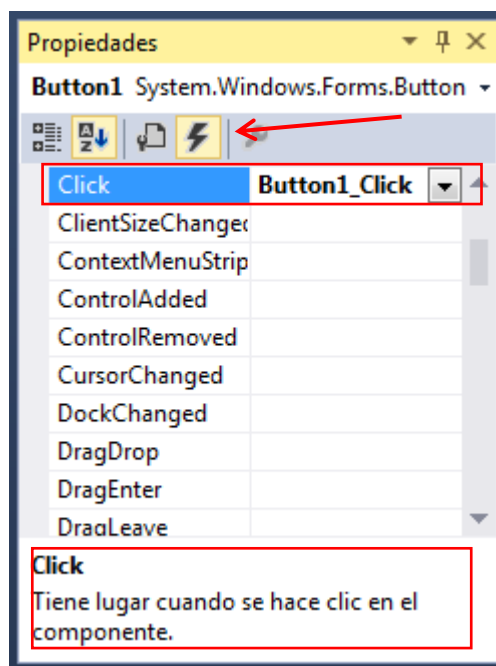
Ahora vamos a escribir código para que se ejecute cada vez que se haga clic en el botón que hemos añadido. Para ello, selecciona el botón Mostrar y hacemos doble clic en él, se mostrará

una nueva ventana. En este caso será la ventana de código asociada con el formulario que tenemos en nuestro proyecto. Nos mostrará:

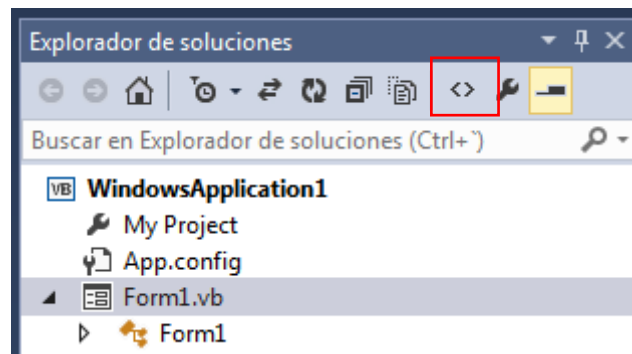


```
Form1.vb* [X] Form1.vb [Diseño]*
Button1 Click
2 referencias
Public Class Form1
    0 referencias
    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    End Sub
End Class
```

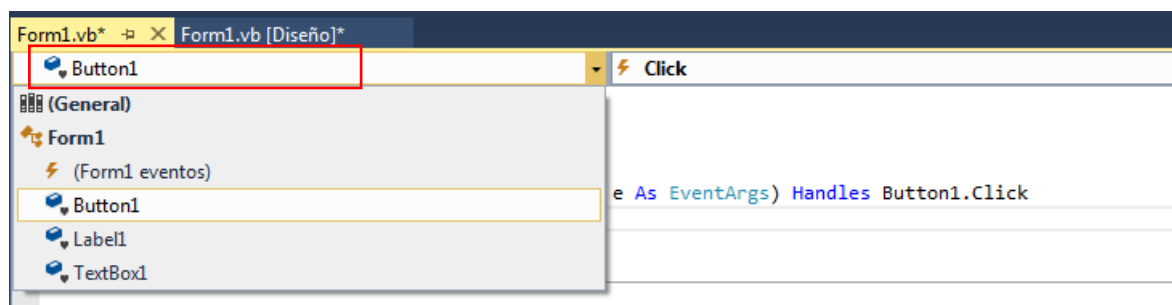
Aquí empieza la programación. Que nosotros pulsemos doble clic en el botón y que aparezca un fragmento de código significa lo siguiente. VB .NET interpreta que quieres poner código para realizar alguna acción cuando se haga clic sobre el botón. Por lo tanto, te muestra ya el "procedimiento" o parte de código que se va a ejecutar cuando suceda esto. A esta forma de trabajar se le llama programación orientada a eventos. Es decir, cuando se produzca el evento de pulsar el botón (clic) ejecutas este código. Por lo tanto, vemos que los controles además de tener propiedades (que modifican su aspecto) también atienden a una serie de eventos (clic, doble clic). Los eventos a los que atienden los controles los podemos ver en la ventana de propiedades, seleccionando:



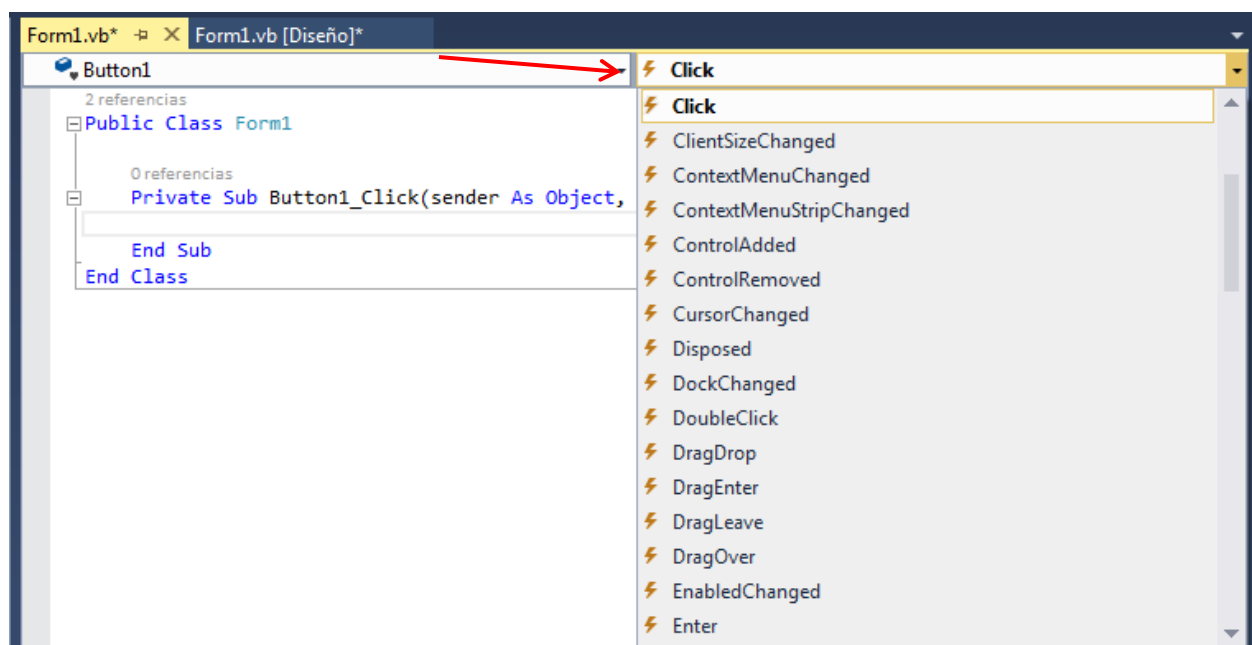
Hay otra forma de acceder a los eventos de los controles y es ésta: nos vamos a la vista del código del formulario pulsando en:



Llegaremos otra vez a la ventana del código de antes. Hacemos clic en el desplegable de arriba a la izquierda y veremos:

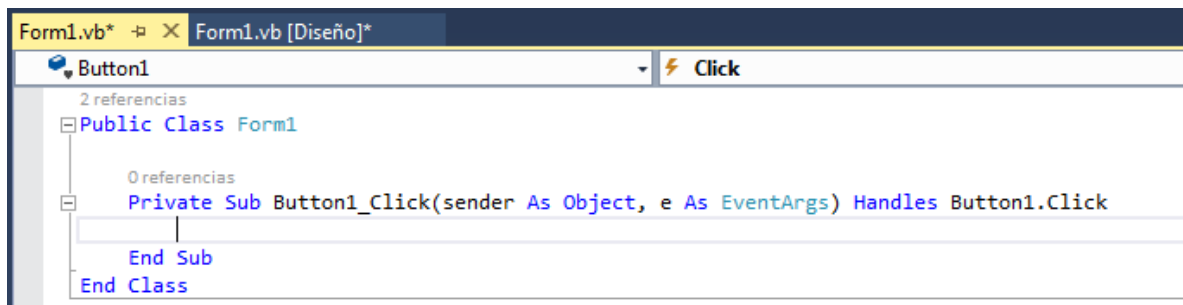


En esta lista podemos ver los objetos que hemos añadido en nuestro formulario (botón, etiqueta y cuadro de texto). Seleccionamos el botón “Button1” y nos fijamos ahora en el desplegable de la derecha:



Esta lista tiene todos los eventos a los que puede atender nuestro objeto, en este caso el botón. Por tanto, cuando necesitemos acceder al “evento tal” del objeto iremos a la vista de código, seleccionaremos el objeto en la lista de la izquierda y el evento en la derecha.

Volvamos a la pantalla del código para el evento clic del botón:



Esta pantalla nos muestra mucha más información de lo que parece en principio. Nos está indicando en la parte superior que estamos trabajando con "Form1.vb". Debajo nos indica a la izquierda que estamos con el control llamado "Button1" y a la derecha trabajando con el evento clic. Como la propiedad más importante de los botones es el clic, eso es lo que nos mostrará de forma predeterminada al hacer doble clic sobre él.

Recordamos que lo que queremos hacer ahora, es escribir el código que se ejecutará cuando se haga clic en el botón, lo cual producirá el evento clic asociado con dicho botón. Ese evento se producirá si se hace un clic propiamente dicho, es decir, con el ratón, o bien porque se pulse intro o la barra espaciadora cuando el botón tenga el enfoque.

Nota: El foco o enfoque es cuando un determinado control está seleccionado. Cuando pulsamos la tecla tabulador para pasar a otro control lo que estamos haciendo es activar el enfoque. El anterior control lógicamente deja de estar seleccionado, luego pierde el enfoque.

La nomenclatura, (forma de llamar a los elementos), para los eventos de Visual Basic sigue el siguiente esquema:

```
[nombre del control] [guión bajo] [nombre del evento]
button1_click ' ejecuta el código cuando se hace clic en el
elemento button1
```

Pero esto sólo es una sugerencia que VB.NET nos hace. Podemos dejar el nombre que Visual Basic nos sugiere o podemos poner el nombre que nosotros queramos. Lo importante aquí es la parte final de la línea de declaración del procedimiento: **Handles Button1.Click**, con esto es con lo que el compilador/intérprete de VB sabe que este procedimiento (fragmento de programa) es un evento y que dicho evento es el evento clic del objeto Button1.

Un detalle: el nombre Button1 es porque hemos dejado el nombre que por defecto el IDE de Visual Studio asigna a los controles que se añaden a los formularios. Si queremos que se llame de otra forma, simplemente mostramos el formulario, seleccionamos el control al que queremos cambiarle el nombre, seleccionamos la propiedad *Name* en la ventana de propiedades y cambiamos el nombre que indica por defecto por el que nosotros queramos... o casi, ya que para

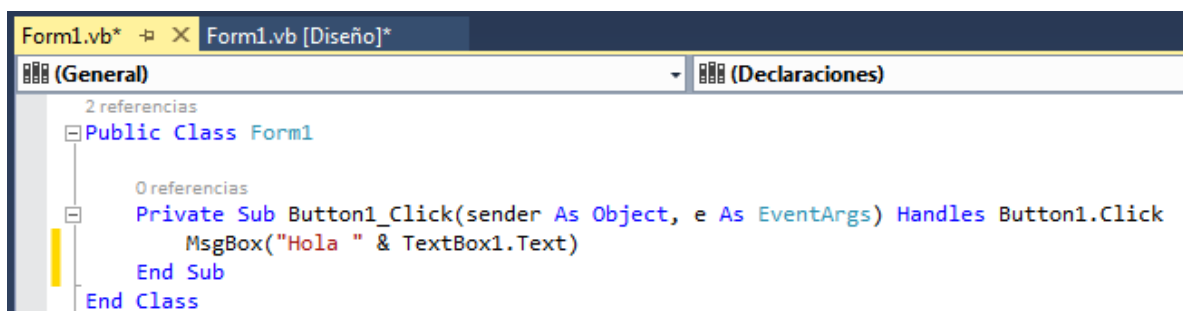
los nombres de los controles, así como para otras cosas que utilicemos en Visual Basic hay que seguir ciertas normas:

- El nombre debe empezar por una letra o un guión bajo.
- El nombre sólo puede contener letras, números y el guión bajo.
- Es buena técnica poner las iniciales del tipo de control que es: lbl_nombre (label de un nombre), btn_aceptar (botón de aceptar), chk_estado (casilla de verificación de estado), txt_nombre (cuadro de texto)

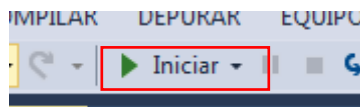
Así que, si queremos cambiar el nombre al evento que se produce cuando se hace clic en el botón, escribiremos ese nombre después de Private Sub, aunque no es necesario cambiar el nombre del evento, ya que, al menos por ahora, nos sirve tal y como está.

Lo que sí importa es lo que escribamos cuando ese evento se produzca, en este caso, vamos a hacer que se muestre un cuadro de diálogo mostrándonos el nombre que previamente hemos escrito en el cuadro de texto.

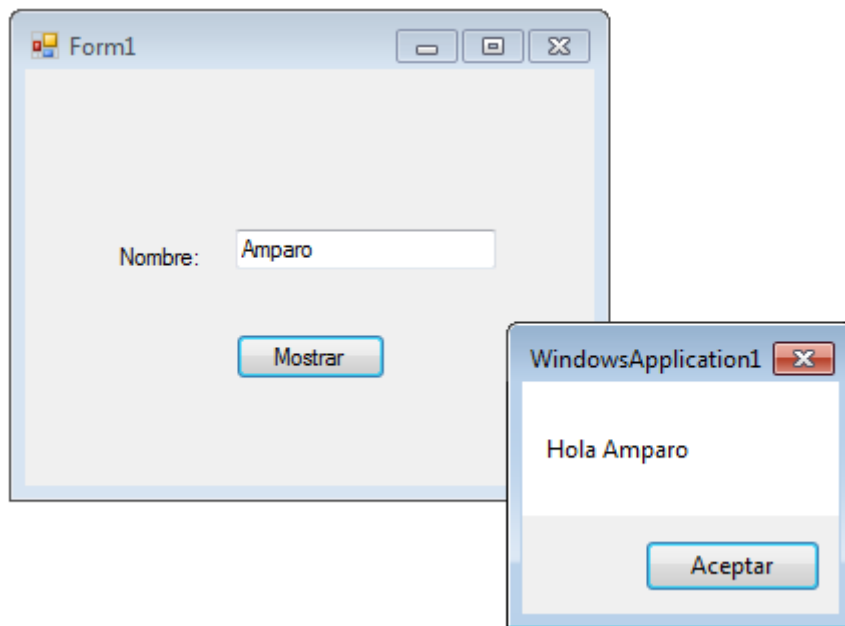
Escribimos el siguiente código en el hueco dejado por Visual Basic entre las líneas que hay entre Private Sub... y End Sub



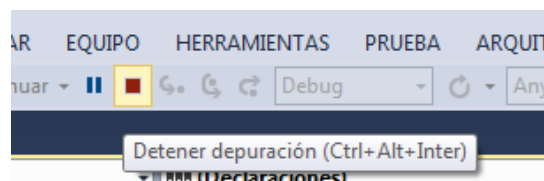
Antes de explicar qué estamos haciendo, pulsamos F5 para que se ejecute el código que hemos escrito o pulsa en el botón "play" que está en la barra de botones.



Cuando aparezca el formulario, escribiremos algo en el cuadro de texto y pulsaremos en el botón "Mostrar". Veremos que se muestra un cuadro de diálogo con el texto "Hola" y a continuación lo que hayamos escrito en el cuadro de texto (TextBox):



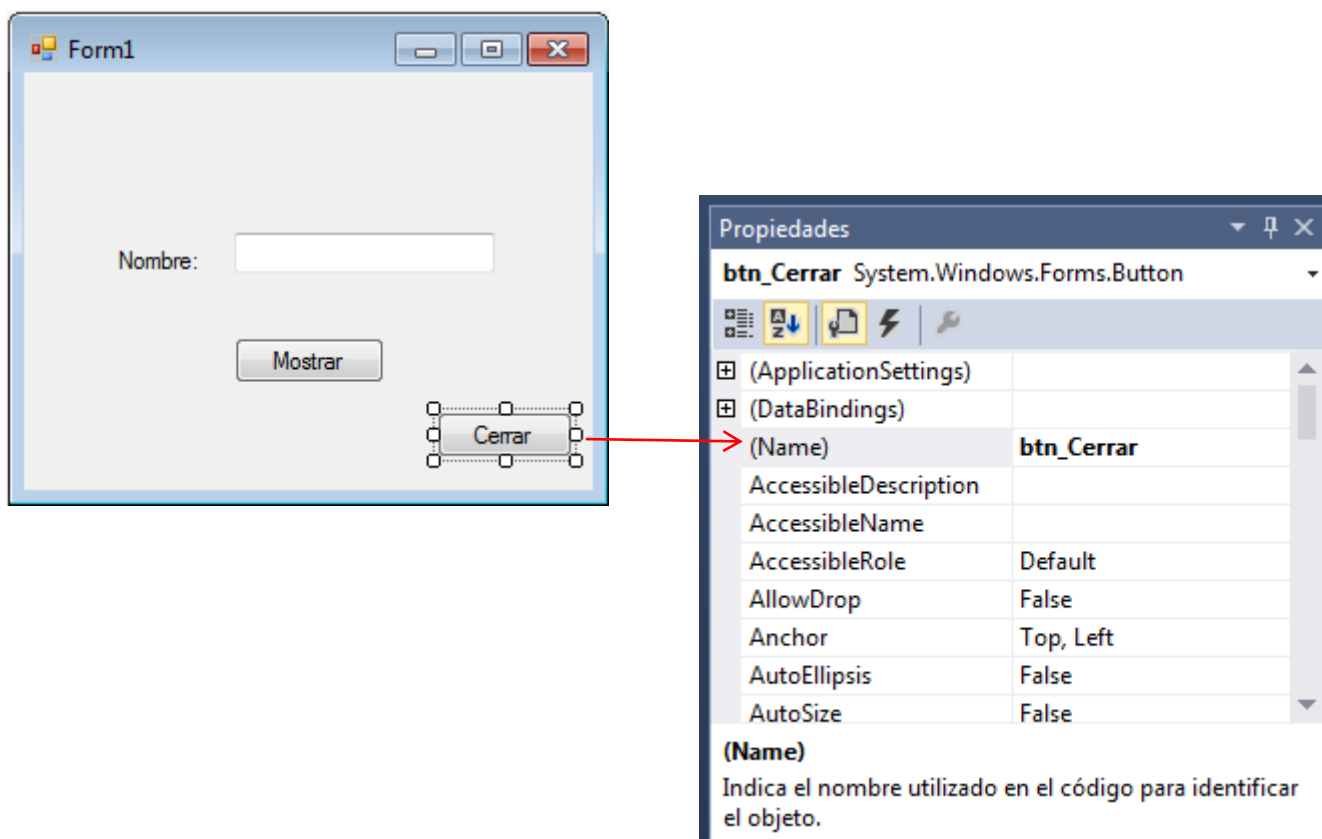
Ya tenemos otra aplicación Windows funcionando creada con Visual Basic .NET. Para terminar pulsamos en la "x" del formulario principal o en el botón de terminar de la barra de tareas.



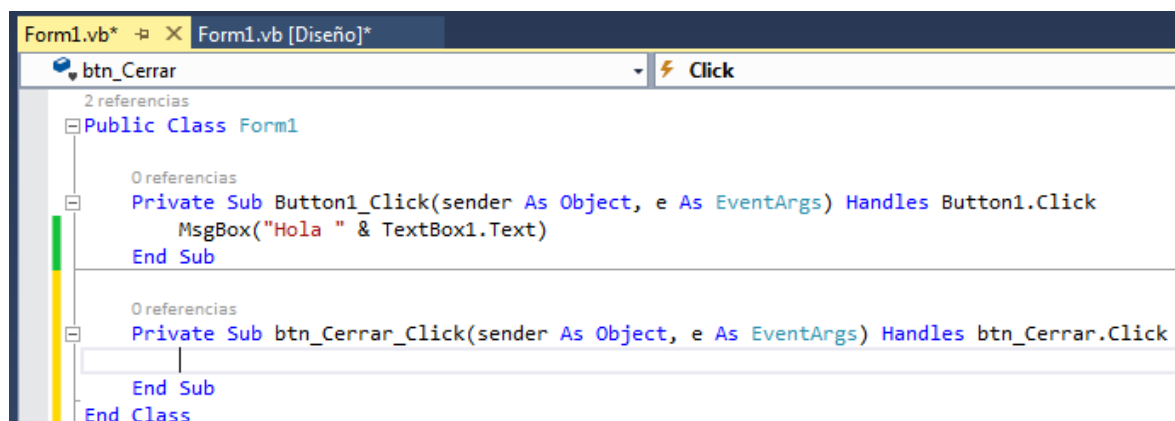
Ahora vamos a añadir otro botón que llamaremos *btn_Cerrar* con el texto "Cerrar". (el nombre es por "btn" de botón y Cerrar de la acción: btn_Cerrar). Para esto primero vamos al IDE para añadir otro botón. Ahora cambiaremos dos propiedades: Name para ponerle btn_Cerrar y Text para ponerle Cerrar:

Nota: Si estamos con el editor de código y queremos pasar al diseñador del formulario haremos doble clic en el formulario en la parte derecha en el "Explorador de soluciones".

Seguimos con el ejemplo, pintamos un nuevo botón y cambiamos las dos propiedades comentadas anteriormente:



Obviamente al ponerle esta vez un nombre al control, en lugar de llamarse Button2 se llamará como le hemos indicado btn_Cerrar y así lo veremos en el código. Pulsamos doble clic en él para escribir el código que queremos que ejecute:



y escribimos la instrucción

`Me.Close()` (Me hace referencia al formulario activo en ese momento)

Pulsamos F5 para ejecutar el programa y la diferencia respecto al programa anterior es que ahora al pulsar este botón el programa termina, cierra el formulario y nos devuelve el entorno de desarrollo. De esta forma el programa está un poco más completo.

Ahora veamos con detalle el código que hemos usado en los dos eventos (clic del botón Mostrar y clic del botón Cerrar):

```
MsgBox("Hola " & TextBox1.Text)
```

MsgBox que es una función o método, (realmente es una clase, como casi todo en .NET), cuya tarea es mostrar en un cuadro de diálogo lo que le indiquemos en el primer **parámetro**, también tiene otros parámetros opcionales, pero por ahora usemos sólo el primero que es obligatorio. Tenemos pues por un lado una instrucción MsgBox que escribe un texto en pantalla y que admite parámetros. Estos parámetros le proporcionan los datos de qué debe hacer, en este caso le proporcionamos un texto para que lo escriba: MsgBox ("Hola"). Los parámetros opcionales sirven para añadirle más funcionalidad o simplemente para que modifique su aspecto... En nuestro ejemplo podría tener un título el cuadro de texto o el mensaje podría ir acompañado de un icono.

En Visual Basic.NET todos los procedimientos que reciban parámetros deben usarse con los paréntesis. Por tanto, para indicarle que es lo que queremos que muestre, tendremos que hacerlo dentro de los paréntesis. En este caso, queremos que se muestre la palabra **"Hola "** y lo que haya en la caja de texto.

La palabra *Hola (seguida de un espacio)* es una **constante**, es decir, siempre será la palabra "Hola" seguida de un espacio. Una vez que el programa esté compilado (listo para ejecutarse por otros usuarios), siempre será lo mismo, por eso se llama constante porque no cambia. En este caso una constante alfanumérica o de cadena, por eso va entre comillas, ya que todos los *literales* o cadenas que queramos usar en nuestros proyectos deben ir dentro de comillas dobles.

Por otro lado **TextBox1.Text** representa el texto que hay en el cuadro de texto y por tanto devolverá lo que en él hayamos escrito. Estamos haciendo referencia a la propiedad Text de un control llamado TextBox1. En nuestro ejemplo pintamos un cuadro de texto y VB .NET le puso el nombre genérico de TextBox1, si modificamos su propiedad Text (recordemos el botón Cerrar o la etiqueta por ejemplo) cambiará el contenido de texto que contiene.

Por último, para que esas dos cadenas de caracteres, la constante Hola y el contenido de la propiedad Text del control TextBox1, se puedan unir para usarla como una sola cadena, usaremos el signo & (ampersand) que sirve para concatenar cadenas de caracteres y hacer que Visual Basic entienda que es una sola.

Por tanto, si la propiedad Text del control TextBox1 contenía la cadena TextBox1, al unir las dos cadenas resultará una nueva con el siguiente contenido: **"Hola TextBox1"** que no es ni más ni menos que la "suma" de las dos cadenas que teníamos.

En el método del evento Clic del botón cerrar hemos escrito: Me.Close(). **Me** representa al objeto o clase Form1 (el formulario) y el método **Close** lo que hace es cerrar el formulario, igual que cuando pulsamos en el botón cerrar del formulario. Otra vez hemos utilizado una propiedad (mejor dicho método) de un elemento, por lo tanto, la sintaxis es ya sencilla: elemento.acción

1.3. Más sobre el código Vb .NET



Volvamos a revisar algunos elementos más teóricos del código que tenemos en pantalla; vamos a explicar el porqué del resto de las líneas de código. Así iremos conociendo términos que luego en temas posteriores acabaremos de encajar y entender. Partimos de nuestro código de ejemplo:

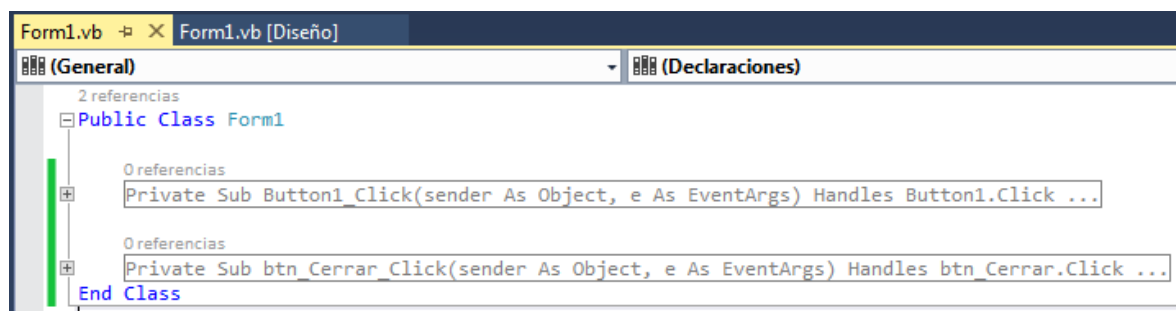
```
Public Class Form1

    Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
        MsgBox("Hola " & TextBox1.Text)
    End Sub

End Class
```

La técnica llamada **Esquematización** u **Outlining** permite definir zonas y regiones de código que pueden ser expandidas o contraídas desde el propio editor, haciendo clic en los indicadores de

código:  



1.4. Análisis del código fuente del formulario

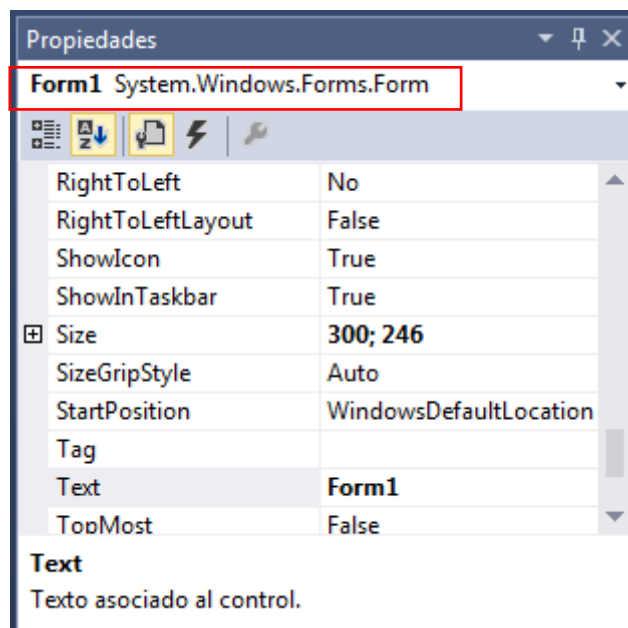
La clase formulario

Todos los formularios son objetos y el código de un objeto se escribe con una clase. Recordemos un poco estos términos, un objeto es cada uno de los elementos de programación de que disponemos: formularios, botones, bases de datos,... y una clase es la plantilla para crear ese elemento. Es decir, con la clase “botón” podemos crear y manipular botones.

Como el formulario es una clase, y debemos crear este formulario hay que utilizar las palabras clave “Class” y “End Class” y heredar esta clase de la clase “Form”. Esto significa que tenemos que crear un formulario nuevo con el nombre por defecto “Form1”. Veamos el código:

```
Public Class Form1
    ...
End Class
```

Que significa crea un formulario llamado Form1. Si vemos el editor y marcamos el formulario:

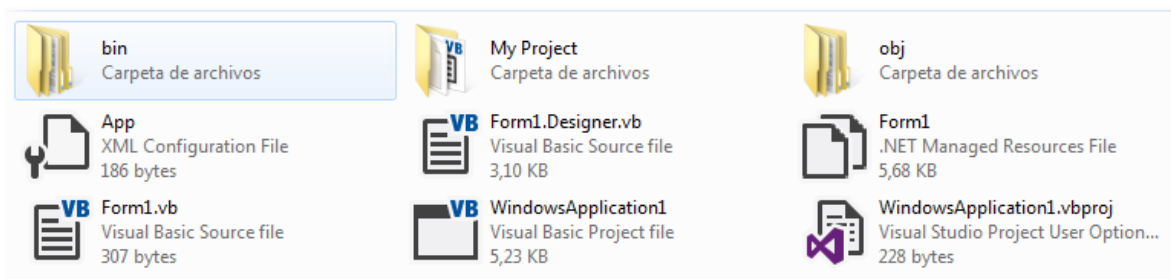


Form1 es un objeto creado a partir de la clase “System.Windows.Forms.Form”

Nota: El espacio de nombres más importante dijimos en el primer tema que era System. De éste parten otros agrupados según sus acciones: acceso a bases de datos: System.Data, para dibujar: System.Drawing, para trabajar con los elementos de Windows: System.Windows y de ahí uno de los componentes de los entornos que son los formularios: System.Windows.Forms. Este spacename contiene todos los elementos de los formularios: cuadros de texto, botones, lista,... y obviamente el propio formulario. Recordemos como antes vimos que el botón pertenecía a la clase “Button”, que pertenece al mismo espacio de nombres que el formulario.

Código oculto del programa

Vamos a ver la carpeta donde tenemos almacenada nuestra aplicación Visual Basic .NET donde nos encontraremos con estos archivos:

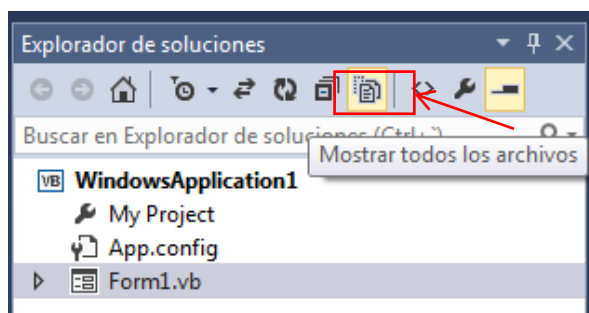


Aunque en el tema anterior dijimos que sólo existían ficheros con extensión .vb, no es del todo cierto, internamente tiene algunos ficheros más. Vamos a leer uno de ellos para comprender las tripas de nuestro programa. Abre con el editor de textos el fichero “Form1.Designer.vb”.

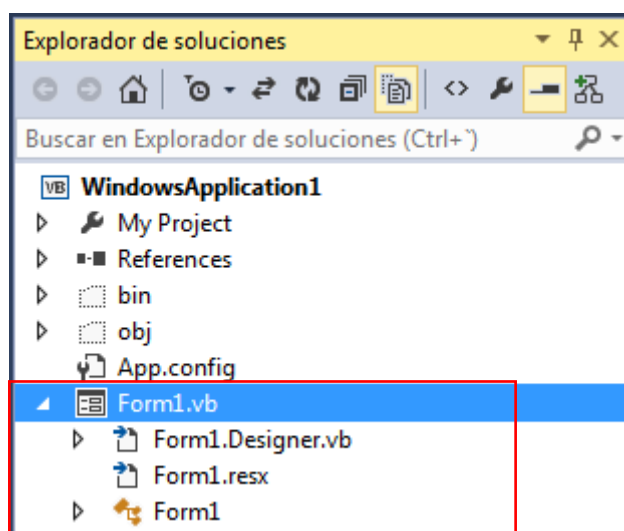
Nota: Si queremos abrir un proyecto en nuestro IDE tendremos que buscar los ficheros con extensión “sln” de solución. Ojo porque si pulsamos en los otros ficheros se abrirán de forma

individual y no como parte del proyecto. Así que hay que tener cuidado si los abres desde el explorador de archivos. Si lo haces desde el IDE no hay problemas ya que por defecto intentará abrir ficheros del tipo.

Volvamos con nuestros ficheros de extensión .vb. Éstos contienen todo lo necesario para crear la aplicación y los controles que hemos pintado. Vamos a indicar en el IDE que nos muestre todos estos ficheros ocultos:



Mostrará ahora un árbol de carpetas más extenso:

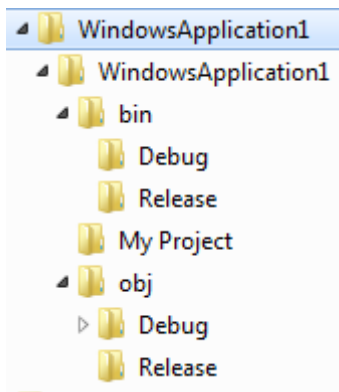


Si expandimos el formulario "Form1.vb" podemos ver su diseñador, es decir, el código necesario para crearlo. Hacemos doble clic sobre este fichero para ver su contenido.

Una vez grabado el proyecto podemos ver de qué partes se compone...

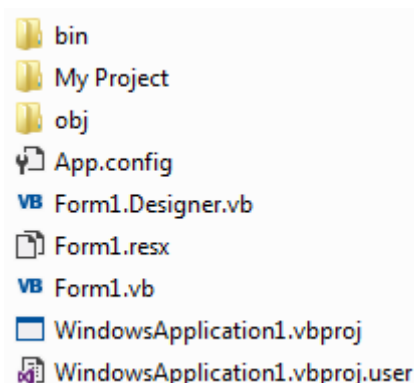
1.5. Ficheros de un proyecto

Al crear un proyecto éste crea unas carpetas con varios ficheros que todos conforman el proyecto. En nuestro ejemplo:



Veamos un poco los ficheros que componen el proyecto según su extensión:

- vb. Es el código fuente Visual Basic. Estos ficheros pueden contener módulos, módulos de clase, formularios,...
- vbproj. Fichero del proyecto. Tiene toda la información de los elementos que forman parte del proyecto.
- sln. Solución. Una solución es el medio que utiliza .NET para agrupar varios proyectos. Estos pueden estar escritos en el mismo lenguaje o distintos.
- vbproj.user. Información sobre las opciones de usuario de proyecto.
- resx. Plantilla de recursos en formato XML.
- exe. Aplicación ejecutable
- pdb. Información sobre la depuración de la aplicación
- designer.vb. Asociado a cada formulario. Contiene el código necesario para la generación de los formularios.

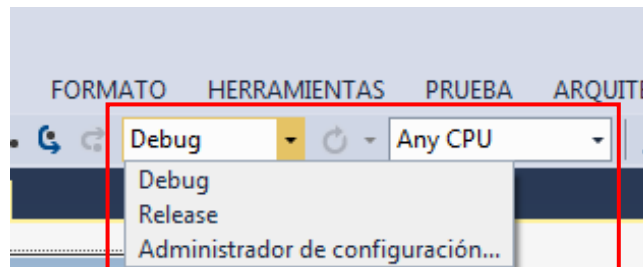


En la carpeta “bin” del proyecto se genera el fichero ejecutable que es lo único que necesitamos para ejecutar nuestro programa en otros equipos.

Dentro de la carpeta “bin” y “obj” hay dos carpetas más: “debug” y “release”. Seguramente las carpetas *release* aún no existirán o estarán vacías. Esto es porque todavía no hemos generado el

programa en su versión final (release), sino que lo hemos hecho en el modo de depuración (debug). En este último modo el IDE introduce multitud de código de control en el programa para que podamos hacer una ejecución controlada y así poder depurar minuciosamente el funcionamiento.

Para activar que el programa se ejecute en un modo y otro utilizaremos la opción:



Si lo ponemos en modo “release” para generar el programa final, veremos como sí nos crea los ficheros adecuados en las carpetas correspondientes.

Una vez probado, dejaremos la opción de depuración “Debug”.

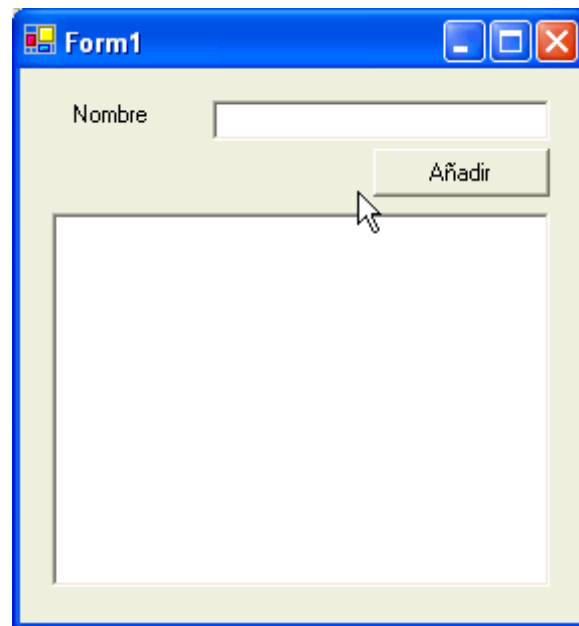
2. El IDE. Formularios

Vamos a seguir practicando con el entorno de desarrollo integrado. Por ejemplo, vamos a hacer que los controles que tengamos en un formulario se adapten de forma automática al nuevo tamaño de la ventana así como a los distintos tamaños de fuentes. No todo el mundo usa la misma resolución de pantalla que nosotros en la edición del programa ni los mismos tamaños de letras. Ya habréis visto en algunas aplicaciones cómo al maximizar la ventana se descolocan los controles o simplemente se quedan en una esquina de la ventana. Veamos cómo hacer esto.

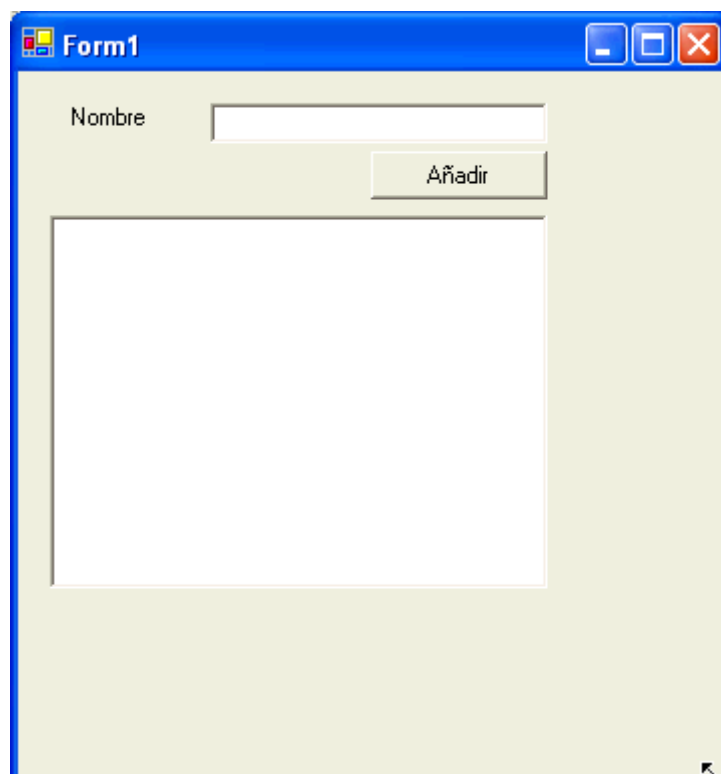
2.1. Redimensionar los controles en un formulario

Veamos un caso típico: el tamaño de una ventana de Windows, (que al fin y al cabo es un formulario), se puede hacer redimensionable, es decir, que el usuario pueda cambiar de tamaño. Lo normal es que los controles que dicho formulario contiene se adapten al nuevo tamaño de la ventana, con la idea de que no queden huecos vacíos al cambiar el tamaño de la ventana.

Por ejemplo, si tenemos esta ventana (o formulario):

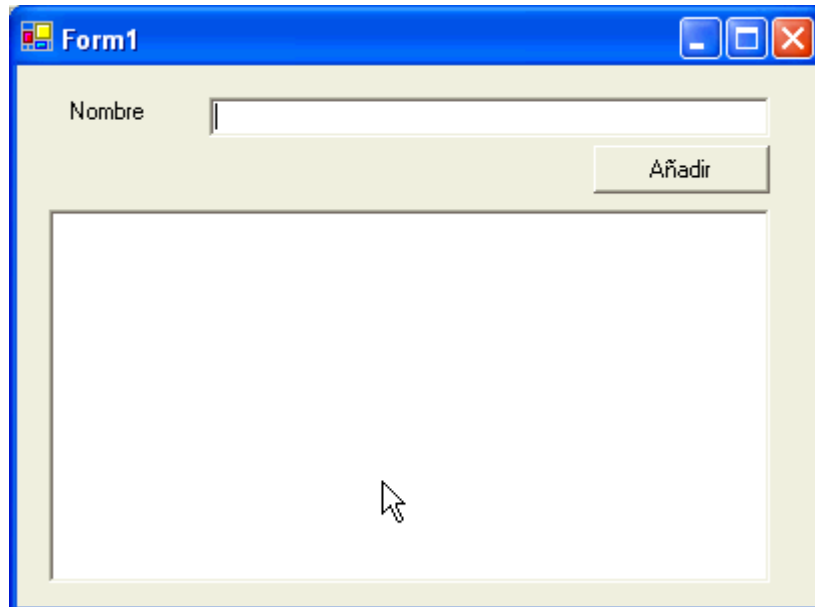


y la agrandamos, por ejemplo, para que tenga este otro aspecto:



Comprobaremos que la ventana se ha agrandado, pero los controles que hay en ella siguen teniendo el mismo tamaño y la misma posición que en la ventana anterior, es decir, que cuando se diseñó el formulario.

Pues bien, la idea de lo que queremos hacer aquí es que al cambiar el tamaño de la ventana se ajusten los controles al nuevo tamaño, para que tuviesen este otro aspecto:



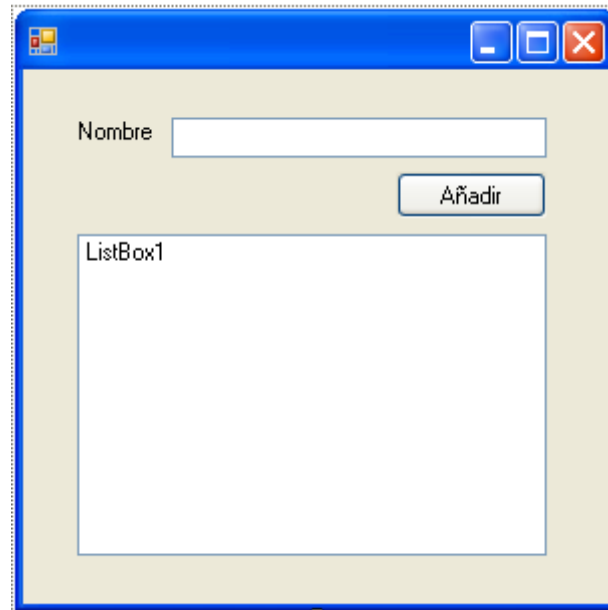
Para que esto sea posible de forma automática, hay que hacer unas cuantas asignaciones a los controles, de forma que podamos indicarle qué tienen que hacer cuando el tamaño de la ventana varíe. En este ejemplo, lo correcto sería que:

- La caja de texto superior se agrandase hacia el lado derecho.
- El botón Añadir se moviese hacia el extremo derecho del formulario.
- La lista se ajustara al ancho y también al alto de la ventana.

Todo esto lo podemos hacer en tiempo de diseño, es decir, cuando tenemos el formulario en el entorno integrado o bien lo podemos codificar dentro del propio formulario. Dónde hacerlo queda a tu criterio. Veamos cómo hacerlo en los dos casos y queda a tu elección cual de los dos métodos quieres utilizar.

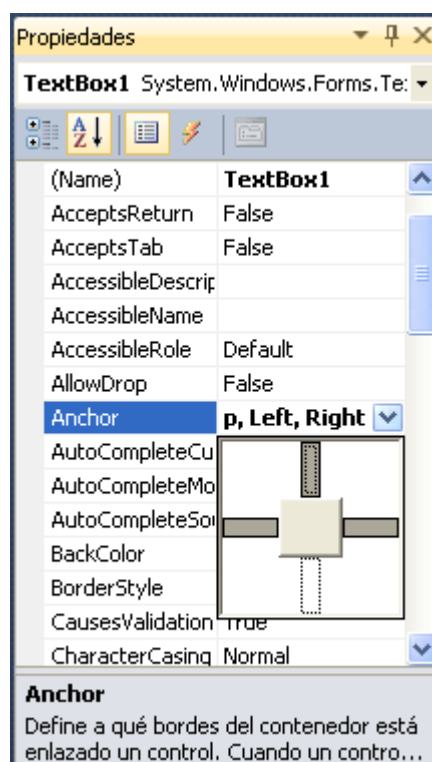
Antes de empezar vamos a crear un nuevo proyecto. Creamos un proyecto del tipo Windows, (aplicación Windows o Windows Application), y le ponemos como nombre WinApp3.

Añadimos una etiqueta (Label), un cuadro de texto (TextBox), un botón (Button) y un cuadro de lista (ListBox). Dejamos los nombres que el IDE ha puesto, salvo para el botón, el cual se llamará btn_Add. Colócalos como hemos visto en la figura de arriba en el primer caso:



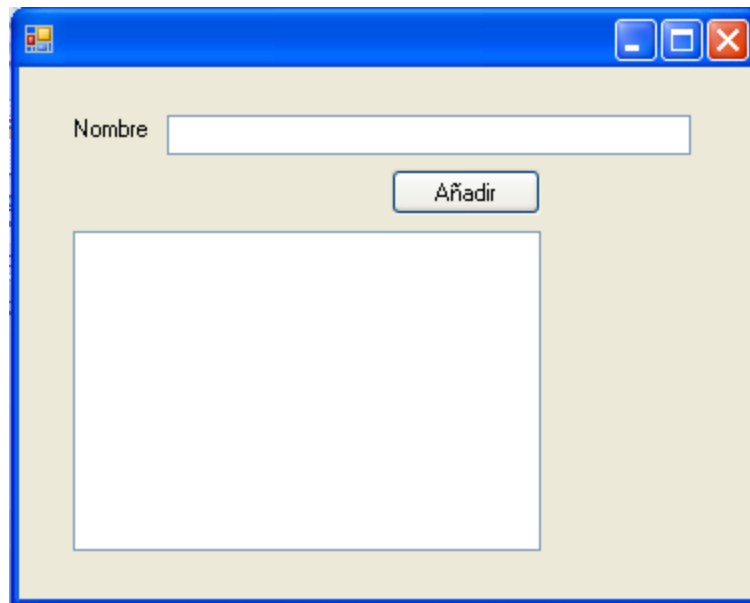
Ejemplo 1. Utilizando el IDE (en diseño)

Comenzamos: selecciona el cuadro de texto (TextBox1) y en la ventana de propiedades, selecciona **Anchor**, verás que por defecto estarán los valores Top y Left, esto quiere decir que la caja de textos estará "anclada" arriba y a la izquierda, pero ahora vamos a seleccionar también la derecha. Cuando pulses en la lista desplegable verás que se muestra una imagen con cuatro líneas:



Inicialmente sólo estaban marcadas la superior y la izquierda con color más oscuro. Pulsa en la línea de la derecha, para que se ponga gris, de esta forma estaremos indicándole que también se "ancla" a la derecha.

Vamos a comprobar que esto funciona... Pulsa en el botón de ejecución o en la tecla F5, cuando tengas el formulario amplía el tamaño de la ventana, verás lo siguiente:

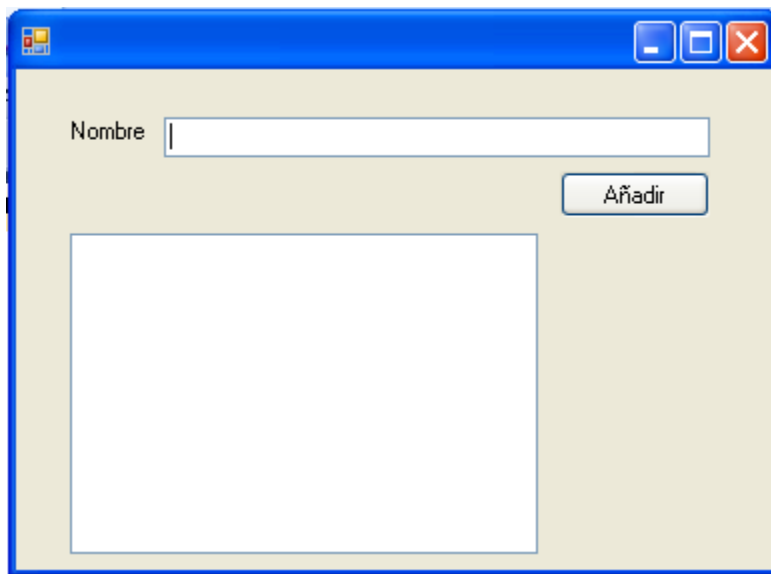
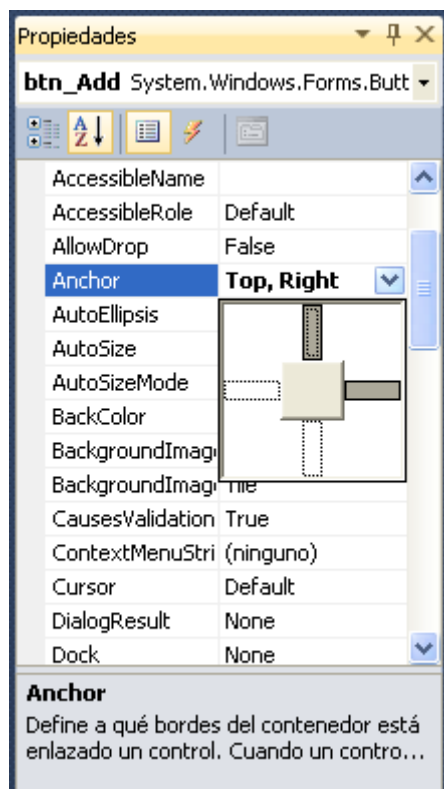


Perfecto, el cuadro de texto se ha adaptado al nuevo tamaño. Ahora encojemos la ventana e incluso haz que no se vea el botón. Podremos comprobar que el cuadro de texto sigue estando proporcionalmente igual de separado de los extremos superior, derecho e izquierdo y se adapta al tamaño de la ventana.

Incluso si intentamos hacer la ventana muy pequeña, el ancho se quedará justo en la parte izquierda de la caja de texto, con el alto podemos hacer que casi desaparezca, (salvo el título de la ventana, la barra de arriba, que se mantiene).

Ahora vamos a "anclar" el botón. Selecciona el botón y en la ventana de propiedades selecciona la propiedad *Anchor*. En este caso, lo que nos interesa es que el botón se desplace a la derecha, pero que no se haga más grande. Para ello, debes seleccionar las líneas de la derecha y la de arriba.

Es decir: áncrate en la parte de arriba y en la derecha, de forma que si cambiamos el tamaño del formulario, el botón se desplazará a la derecha o a la izquierda, pero no cambiará de tamaño, como le ocurre al cuadro de texto:



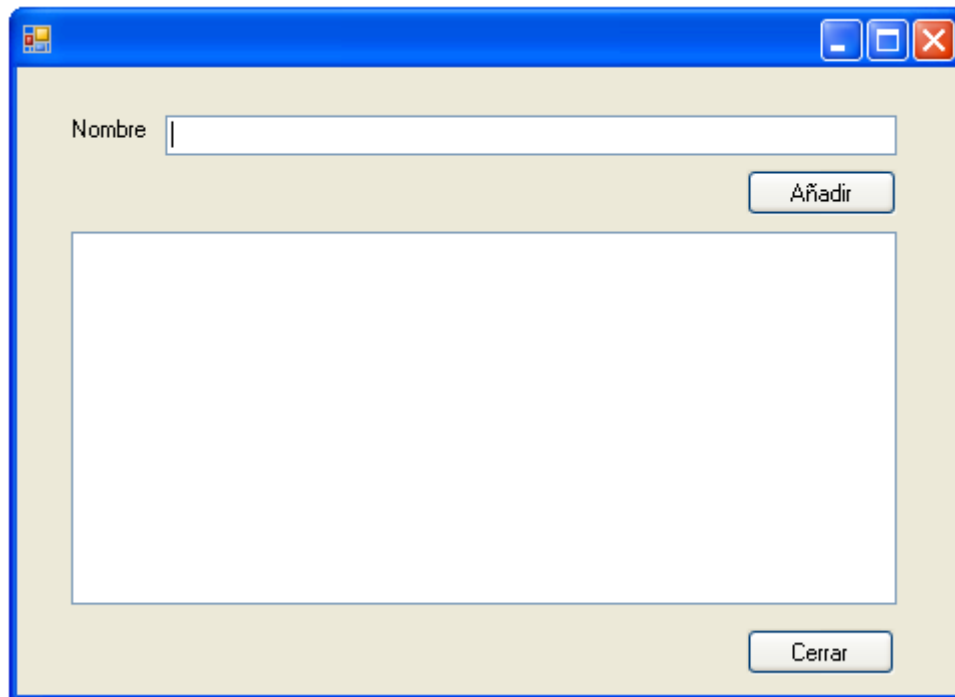
Si pulsamos F5 veremos que el botón llegará casi a tocar el lado izquierdo del formulario y allí se quedará, no permitiendo que se haga más pequeño.

Por último vamos a anclar el cuadro de lista... en este caso debe hacerse grande tanto hacia la derecha como hacia la izquierda e incluso cuando se estira el formulario desde la parte inferior, pero en la parte superior debe mantenerse en la misma posición. Simplemente debemos marcar las cuatro posiciones y hará que se ajuste siempre en la parte inferior y derecha.

Vamos a complicar un poco más la cosa y vamos a añadirle otro botón. En este caso, dicho botón estará en la parte inferior derecha del formulario, será el botón cerrar y al pulsarlo hay que cerrar el formulario.

Utilizaremos el método *close* del formulario en el evento clic del botón. Es decir, *Me.Close* en el evento Clic de botón, que vamos a llamar *btn_Cerrar*. Como decíamos antes, este botón se debería anclar en la parte inferior derecha, por tanto, los valores que hay que asignar en *Anchor* son precisamente esos: *Bottom* y *Right* (abajo y derecha).

Como habrás notado, con el *Label1* no hay que hacer nada, ya que por defecto el tamaño se ajusta por arriba y a la izquierda, por tanto, se quedará en la misma posición. Son los valores por defecto de la propiedad *Anchor*, por eso no es necesario asignarle nada.



Ejemplo 2. Por código (en tiempo de ejecución)

Volvamos a nuestro IDE y hacemos doble clic en el formulario, dentro de él, no en los controles (botones, cuadro de lista, ..) debe ser en el formulario. Se debe mostrar el evento *Form_Load*:

```
Private Sub Form1_Load(ByVal sender As Object, e As EventArgs) Handles MyBase.Load  
  
End Sub
```

Este código es el que se va a ejecutar cuando se cargue el formulario. Vimos cuando hicimos doble clic en el botón que nos presentaba el código para ese evento. Si lo hacemos en el formulario se presenta el código que ejecutará al cargar (Load) el formulario Form1: **Form1_Load**, además aparecen una serie de parámetros que ya veremos más adelante. Esto se ejecuta pues cuando el formulario "se carga" en la memoria, justo antes de mostrarse, por tanto aquí es un buen sitio para hacer algunas "inicializaciones" o asignaciones que nuestro formulario necesite. De hecho es muy utilizado para que podamos limpiar el contenido de la lista, el del cuadro de texto, etc. e incluso hacer las asignaciones para que los controles se queden "anclados" en la posición que nosotros le indiquemos.

Vamos a ver primero cómo se "declara" este evento, aunque el VB lo hace automáticamente por nosotros, es conveniente verlo para ir aclarando conceptos... que tal vez ahora no necesites, pero en un futuro casi seguro que te hubiese gustado haberlo sabido.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _  
    System.EventArgs) Handles MyBase.Load  
  
    Me.TextBox1.Text = ""  
    Me.ListBox1.Items.Clear()  
  
End Sub
```

Lo primero que hay que entender es que la palabra **Handles** es la palabra que le indica al compilador de Visual Basic .NET qué evento es el que "manipula" o maneja este procedimiento. Esto, (la declaración del Sub), se encarga de manejar el evento Load del objeto MyBase. Resumiendo: es el código que se va a ejecutar cuando (handles) se inicie el formulario (mybase.load).

El objeto **Mybase** se refiere al objeto base del que se deriva el formulario, recuerda que en .NET todo está basado en objetos y en programación orientada a objetos y todo objeto se deriva de un objeto básico o que está más bajo en la escala de las clases, es decir, un formulario se basa en la clase **System.Windows.Forms.Form** y a esa clase es a la que hace referencia el objeto MyBase, mientras que **Me** se refiere a la clase actual, la que se ha derivado de dicha clase Form o, por extensión, a cualquier clase, como veremos en futuras ocasiones.

Veamos ahora el código para que los controles se anclen al formulario de forma que se adapten al nuevo tamaño del mismo:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As _  
    System.EventArgs) Handles MyBase.Load  
  
    Me.TextBox1.Text = ""  
    Me.ListBox1.Items.Clear()  
    ' Asignar los valores para "anclar" los controles al formulario  
    ' El TextBox1 se anclará Arriba, Izquierda y Derecha  
    TextBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left Or _  
    AnchorStyles.Right  
    ' El botón Añadir lo hará Arriba y a la derecha:  
    btn_Add.Anchor = AnchorStyles.Top Or AnchorStyles.Right  
    ' El listbox lo hará en los cuatro vértices:  
    ListBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left Or _  
    AnchorStyles.Right Or AnchorStyles.Bottom  
    ' El botón cerrar sólo lo hará a la derecha y abajo  
    btn_Cerrar.Anchor = AnchorStyles.Right Or AnchorStyles.Bottom  
  
End Sub
```

Cuando se inicie el formulario se realizarán las siguientes operaciones: limpiamos el contenido del cuadro de texto, simplemente asignándole una cadena vacía a la propiedad que almacena el texto que se escribe en pantalla Text: Me.Textbox1.Text="". Muchas veces los objetos se leen al revés para interpretarlos mejor. En este caso sería: asignamos una cadena vacía a la propiedad Text del cuadro de texto llamado Textbox1 del formulario actual Me.

Listbox es un cuadro de lista, es decir, es una lista de elementos. Inicialmente cuando se ejecute el programa quiero borrar todos los elementos así que los borro llamando al método **Clear** de la colección de elementos, así la dejo en blanco. El resto del código asigna a la propiedad **Anchor** (anclaje) los valores que queremos y que luego condicionarán su funcionamiento. Por ejemplo:

```
TextBox1.Anchor = AnchorStyles.Top Or AnchorStyles.Left Or _  
AnchorStyles.Right
```

hará que el cuadro de texto se ancle arriba a la izquierda y a la derecha. Los **Ors** que se están utilizando sirven para "sumar" y el resultado sería el mismo que si usáramos el signo de suma, pero la razón de usar **Or** es porque lo que queremos hacer es una suma de bits. Realmente da lo mismo usar la suma que **Or** en este caso, pero dejemos el **Or** que es lo apropiado ya que no estamos sumando números sino concatenando propiedades.

El efecto es el mismo que en tiempo de diseño. Pero de esta forma sabemos cómo modificar el comportamiento del anclaje de los controles en la ejecución del programa. Por ejemplo, en algunas ocasiones has visto un formulario que tiene un botón de "más información". Al pulsarle la ventana o formulario se amplía poniendo más información en pantalla. Se han modificado la colocación y tamaño de algunos de los controles del formulario en tiempo de ejecución.

2.2. Redimensionar el tipo de letra en un formulario

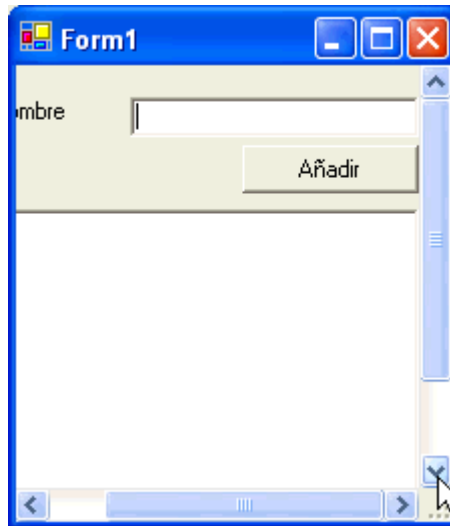
Sigamos ajustando nuestro formulario. Esta vez que los controles se adapten también a otros tamaños de fuentes, no a los que nosotros tenemos en nuestro equipo, ya que hay usuarios que por necesidades tienen que poner tamaños de fuentes más grandes e incluso más pequeñas.

La propiedad que hace eso posible es **AutoScaleMode**. Por defecto, tiene el valor *Font*. El escalado por *Font* es útil si se desea tener un control o un formulario para expandir o reducir según el tamaño de las fuentes en el sistema operativo, y debe usarse cuando no importa el tamaño absoluto del control o formulario. El escalado por *Dpi* es útil cuando se desea cambiar el tamaño de un control o un formulario con respecto a la pantalla. Por ejemplo, puede que se desee usar el ajuste de escala de puntos por pulgada (PPP) en un control que muestre un gráfico u otro gráfico para que siempre ocupe un determinado porcentaje de la pantalla.

Por tanto, los formularios, sin necesidad de que hagamos nada, se auto ajustarán al tamaño de las fuentes.

Otro tema interesante que tienen los formularios es la propiedad **AutoScroll**. Si asignamos el valor *True* (verdadero) a esta propiedad, hacemos que cuando el formulario se haga muy pequeño o muy estrecho, se muestren unas barras de desplazamiento (scrolls) para que pulsando en ellas podamos ver el contenido del mismo.

Por ejemplo, si no hubiésemos "anclado" nuestros controles, al hacer el formulario más estrecho se mostrarían unas barras de desplazamiento para que podamos ver los controles que están contenidos en el formulario como en esta figura:



Sigamos escribiendo algunos ejemplos de código para seguir familiarizándonos con el IDE de .NET. Queremos hacer lo siguiente en nuestro programa:

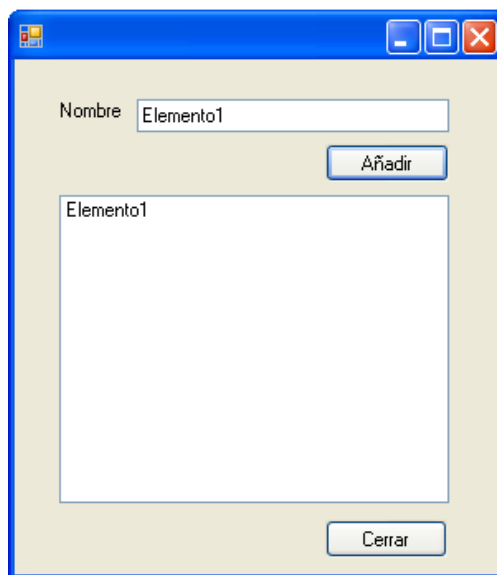
1. Al pulsar en el botón *Añadir*, se añadirá a la lista lo que hayas escrito en la caja de texto.
2. Al pulsar *Intro* será como si hubieses pulsado en el botón *Añadir*.
3. Al pulsar *Esc* es como si hubieses pulsado en el botón *Cerrar*.
4. Al pulsar en uno de los elementos de la lista, éste se mostrará en la caja de texto.
5. Al seleccionar un elemento de la lista y pulsar la tecla *Supr* (o *Del* si tu teclado está en inglés), dicho elemento se borrará de la lista, pero se quedará en la caja de texto, ya que al seleccionarlo para poder pulsar la tecla suprimir se habrá mostrado...

Veamos cómo hacer esto.

1. Lo de pulsar en *Añadir* y hacer algo: simplemente codificamos lo que haya que codificar en el evento *Clic* del botón `btn_Add`. Para que nos muestre ese evento, sencillamente haremos doble-click en el botón y el VB te mostrará el evento en cuestión y añadimos este código:

```
Private Sub btn_Add_Click(sender As Object, e As EventArgs) Handles  
    btn_Add.Click  
        ListBox1.Items.Add(TextBox1.Text)  
End Sub
```

Veamos qué hace este código en ejecución:



Conozcamos que hay en esta instrucción del procedimiento que se va a ejecutar cuando se haga clic (evento clic) del botón. Antes, recordemos el código que vimos en el evento `Form_Load`, teníamos:

`Me.ListBox1.Items.Clear()`

- **Me** hace referencia a la clase actual, es decir, al formulario.
- **Items** son los elementos que tiene el objeto *ListBox*
- **Clear** es un método de *Items* que se encarga de limpiar los elementos de la lista, es decir, los borra.

Por tanto, esa línea lo que hace es borrar los elementos del *listbox*.

Ahora lo que necesitamos no es borrarlos, sino añadir nuevos elementos a la lista. Como ya sabemos que **Items** es el contenedor en el que se guardan los elementos de la lista, lo único que tenemos que saber es cómo añadir nuevos elementos a la lista. La respuesta es mediante la utilización del método **Add** que añade elementos a la lista de un *ListBox*: Para añadir elementos a un *listbox* se usa el método *Add* de *Items*.

En la colección *Items* añadimos el contenido de nuestro cuadro de texto de pruebas "TextBox". Pondremos esta instrucción:

```
Items.Add(TextBox1.Text)
```

El resultado es que se añade "add" a la lista de elementos "Items" el texto "text" que contiene el cuadro de texto "Textbox1", con lo que aparece reflejado inmediatamente en el cuadro de lista.

2.3. ¿Qué es una colección?

Una **colección** es una forma de agrupar y administrar objetos relacionados. En el caso anterior un objeto colección *Items* es una colección de elementos, es decir, los elementos están guardados en

una "lista" de datos. Normalmente trabajaremos con muchos tipos de colecciones: los iconos de la barra de herramientas es una colección de objetos de tipo botón, las impresoras instaladas en nuestro ordenador es una colección de objetos impresora, un libro de trabajo de Excel es una colección de hojas de cálculo,...

Ya hemos añadido un elemento a la lista, si seguimos pulsando en añadir vemos cómo vamos creando la lista con todos los elementos deseados. Veamos ahora cómo saber cuál es el elemento que se ha seleccionado de la lista y cómo asignarlo al TextBox. Es decir, el usuario selecciona un elemento y queremos que se escriba en el cuadro de texto. Observa este código:

```
Private Sub ListBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
    Handles ListBox1.SelectedIndexChanged
        ' Cuando pulsamos en un elemento de la lista...
        With ListBox1
            TextBox1.Text = .SelectedItem
        End With
    End Sub
```

En este caso no es el evento Clic, sino que el evento en cuestión es **SelectedIndexChanged**. Este evento se activa cuando se selecciona un elemento: cambia el índice del elemento seleccionado.

Lo que hacemos en este evento es asignar al contenido del cuadro de texto (su propiedad "Text") el elemento de la lista seleccionado: la propiedad **SelectedItem** representa al elemento seleccionado.

Además, ¿qué es esto del *With... End With*? Cuando estamos trabajando con varias propiedades de un control u objeto (cuadro de lista en este caso) podemos hacerlo de esta forma:

```
TextBox1.Text = Listbox1.SelectedItem
```

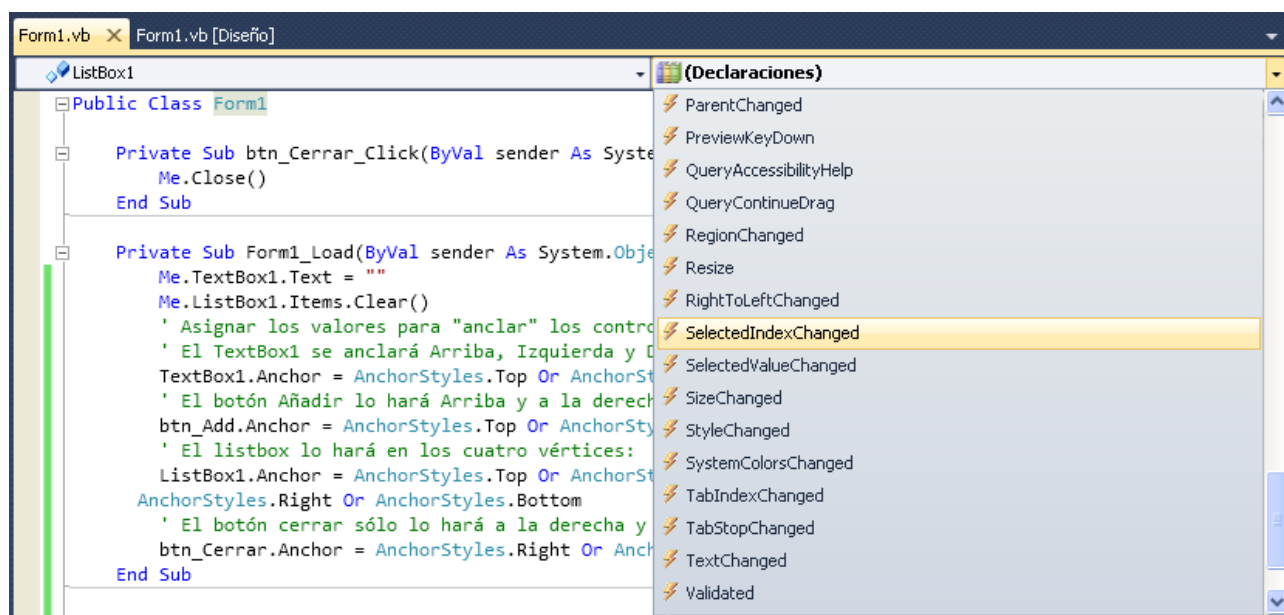
Es decir, le indicamos el método o propiedad de control separado por un punto. Pero en ocasiones escribiremos muchas instrucciones sobre ese cuadro de lista. Para evitar escribir tantas veces el objeto con el que estamos trabajando podemos utilizar la instrucción *With...End With*

```
With ListBox1
    TextBox1.Text = .SelectedItem
End With
```

Que significa: "con el cuadro de lista Listbox1" haz lo siguiente... y cuando se encuentre una propiedad con el punto delante `.SelectedItem` sabe que debe ser del control que se indicó en el *With*. Es para simplificar la sintaxis, la utilidad no va más allá de hacer el código más legible y puede omitirse tranquilamente su utilización y escribir en su lugar el objeto completo.

2.4. ¿Cómo hacemos para escribir código en otros eventos?

Para poder usar otros eventos de un objeto, debemos realizar esto en la ventana de código. Primero seleccionamos de la lista desplegable de la izquierda el objeto con el que queremos trabajar, en nuestro caso el cuadro de lista. Automáticamente se actualiza un desplegable a la derecha con los eventos disponibles para este objeto:



Vemos en esta pantalla que a la izquierda está seleccionado el cuadro de lista *ListBox1* y a la derecha aparecen todos los eventos disponibles para este cuadro de lista.

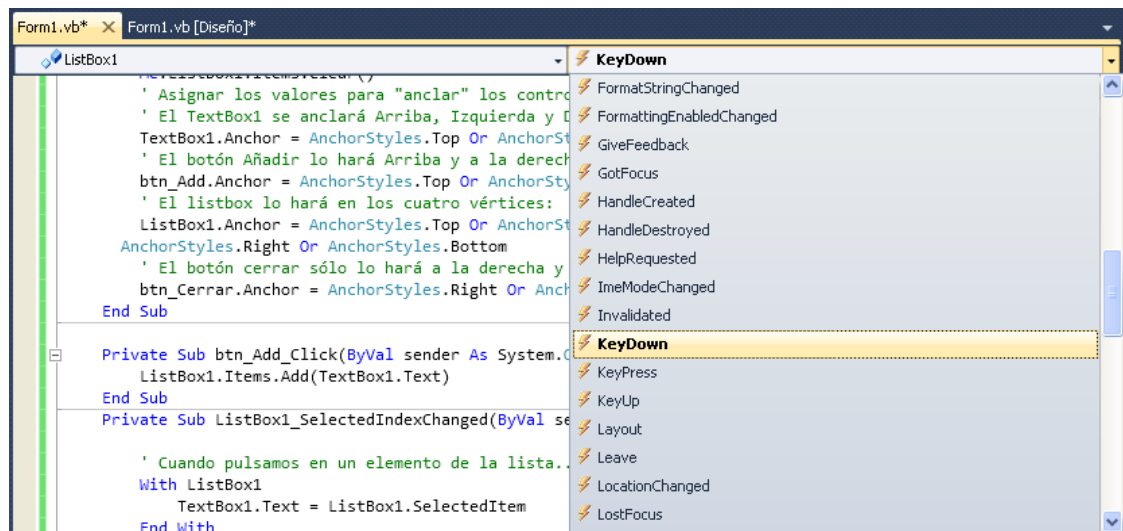
Nota: Cada control tiene sus propios eventos. Muchos son comunes, por ejemplo, el evento clic está disponible en casi todos. Luego tiene los propios que dan funcionalidad distinta a cada control.

Para trabajar con el evento que queremos, desplegaremos la lista de la derecha y seleccionamos el adecuado.

Cómo manejar eventos de teclado.

En el ejemplo anterior hemos controlado qué hacer cuando el usuario hace clic con el ratón en un elemento de la lista, ahora vamos a controlar qué hacer con determinadas teclas. Para esto vamos a utilizar el evento que controla cuándo se pulsan las teclas: el evento **KeyPress**.

Seguimos con el ejemplo anterior, vamos a la pantalla del código, arriba a la izquierda seleccionamos el cuadro de lista y a la derecha buscamos el evento **KeyDown**:



El procedimiento que nos escribirá debajo es:

```
Private Sub ListBox1_KeyDown(sender As Object, e As KeyEventArgs) _
    Handles ListBox1.KeyDown

End Sub
```

Pero si hemos dicho que íbamos a utilizar el KeyPress ¿por qué cogemos ahora el KeyDown?. Bien, veamos los eventos que tenemos para manejar el teclado:

- **KeyPress:** Se produce cuando el usuario presiona y suelta una tecla
- **KeyDown:** Se producen cuando el usuario presiona una tecla
- **KeyUp:** Se producen cuando el usuario suelta una tecla

Cada uno de estos eventos tiene su utilidad. Por ejemplo, el KeyPress digamos que es el "normal", cuando se presiona una tecla normalmente y se suelta. Pero ¿y si queremos utilizar dos teclas simultáneas? Debemos saber si está la primera tecla pulsada, y en esa situación KeyDown será la opción adecuada, ya que con este evento se interceptan las pulsaciones de tecla cuando se empieza a pulsar la misma. Además existen muchas teclas especiales que sólo funcionan con KeyDown como es el caso de la tecla Supr (suprimir) que es una tecla especial y no se detecta en el evento KeyPress.

Con esto, ya sabemos cómo detectar cuando se pulsa la tecla suprimir (Del), ahora debemos borrar el elemento de la lista que esté activo o seleccionado:

```
Private Sub ListBox1_KeyDown(sender As Object, e As KeyEventArgs) _
    Handles ListBox1.KeyDown
    If e.KeyCode = Keys.Delete Then
        With ListBox1
            .Items.Remove(.SelectedItem)
        End With
    End If
End Sub
```


La explicación es la siguiente: Cuando se ejecuta este procedimiento una variable llamada "e" indica el código de la tecla que se ha pulsado. La comparamos entonces para ver si la tecla de suprimir: `e.KeyCode = Keys.Delete`. Si es así, eliminamos el elemento que está seleccionado.

La variable "e" está en la definición del evento de este cuadro de lista:

```
Private Sub ListBox1_KeyDown(sender As Object, e As KeyEventArgs) _  
    Handles ListBox1.KeyDown
```

Cada evento nos proporcionará cierta información según sea su naturaleza. Está claro que si estamos seleccionando el evento de pulsar una tecla "KeyDown" nos tendría que decir qué tecla es la pulsada. Así es, lo dice en el parámetro de entrada:

`e As KeyEventArgs`

que nos dice que nos proporciona una variable que se llama "e" que es de tipo "KeyEventArgs", es decir, de tipo evento de teclado o tecla.

Recuerda que **SelectedItem** nos indica el elemento que actualmente está seleccionado y usando el método **Remove** de la colección **Items**, lo quitamos de la lista.

Nota: El parámetro de entrada llamado "e" que veremos en los eventos tiene la información adecuada al evento. Si es de tecla, cuál es la que se ha pulsado. Si es de selección de un elemento de la lista, el elemento que se ha seleccionado, si es la de mover un cuadro por la pantalla tendrá las coordenadas de dónde se ha desplazado.

Por tanto, "e" puede tener un solo valor o varios, dependiendo del evento que esté detectando.

Selecciones múltiples en un cuadro de lista

En este ejemplo queremos hacer una selección múltiple de un cuadro de lista y suprimir todos los que estén seleccionados. Con lo que sabemos tenemos ya una idea aproximada de cómo hacerlo pero nos falta alguna cosa. Veamos.

Una selección múltiple (seleccionar varios elementos a la vez de una lista) es algo habitual y que debemos tener en cuenta en nuestros programas. Cuando permitimos múltiple selección en un cuadro de lista (ListBox), podemos seleccionar un elemento o varios, en este caso éstos pueden estar consecutivos o no. Por ejemplo, si seleccionas un elemento de la lista y manteniendo pulsada la tecla *Shift* (mayúsculas), pulsas en otro que está más arriba o más abajo, se seleccionan todos los elementos intermedios, (esto es habitual de Windows); también puedes seleccionar elementos no contiguos, si pulsas la tecla *Control* y con el ratón vas haciendo clic en elementos no consecutivos.

Lo primero que necesitamos saber es:

¿Cómo hacer que un ListBox permita múltiple selección?

Ya que por defecto sólo se puede seleccionar un elemento a un mismo tiempo.

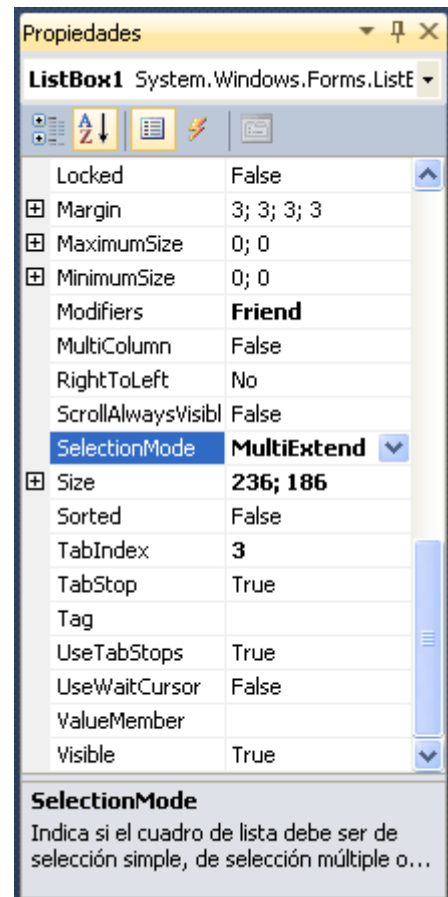
Para que un ListBox permita múltiple selección de elementos, hay que asignar a la propiedad **SelectionMode** el valor **MultiExtended**. Por tanto, selecciona el cuadro de lista ListBox1 y en la ventana de propiedades, asigna dicho valor a la propiedad SelectionMode.

Ahora tenemos que hacer más cosas cuando se detecta la pulsación de la tecla suprimir en el evento KeyDown, ya que tenemos que saber qué elementos están seleccionados para poder borrarlos. Es decir, de alguna manera debemos saber qué elementos están seleccionados.

Lo primero que tenemos que hacer es recorrer todos los elementos del ListBox para saber si está o no seleccionado, pero ese recorrido hay que hacerlo desde atrás hacia adelante... ¿por qué? Porque si lo hiciéramos desde el principio de la lista, al eliminar un elemento de dicha lista, el número de elementos variaría y tendríamos problemas cuando llegásemos al final, ya que no será el mismo número de elementos después de haber borrado alguno, mientras que al recorrer los elementos desde el final hacia adelante, no importará que borremos alguno del final, ya que el siguiente que comprobaremos estará más al principio que el recién borrado y no tendremos problemas.

Veamos primero el código que habría que usar y después lo comentamos.

```
Private Sub ListBox1_KeyDown(sender As Object, e _ As KeyEventArgs) _  
Handles ListBox1.KeyDown  
  
    If e.KeyCode = Keys.Delete Then  
        ' Borrar las palabras seleccionadas del listbox  
        Dim i As Integer  
        ' Elemento en la posición i del listbox  
        With ListBox1  
            For i = .SelectedItems.Count - 1 To 0 Step -1  
                .Items.Remove(.SelectedItems.Item(i))  
            Next  
        End With  
    End If  
  
End Sub
```



La parte del **If e.KeyCode = Keys.Delete Then** ya la vimos antes. Sirve para detectar la pulsación de la tecla de borrado

Dim i As Integer indica al programa que vamos a usar un número y que ese número lo guarde en la *variable* i.

With ListBox1 el With se utiliza para simplificar las cosas, ya lo comentamos antes.

Visual Basic .Net nos indica en ese bucle los elementos que están seleccionados gracias a la colección de elementos seleccionados "SelectedItems.Count. Así que devolverá una colección con los elementos que ha seleccionado el usuario. Además todas las colecciones tienen una propiedad llamada **Count** que me dice cuántos elementos hay en esa colección. Pues bien, si hacemos:

For i = .SelectedItems.Count - 1 To 0 Step -1

le estamos indicando que use la *variable* i para ir guardando los valores que resulten de contar desde el número de elementos que hay seleccionados hasta cero. El Step -1 se usa para contar hacia atrás, (de mayor a menor), pero eso, al igual que el For, también lo veremos más adelante.

SelectedItems es una colección en la que están los elementos que hay seleccionados en el ListBox.

.Items.Remove(.SelectedItems.Item(i)) Esto elimina el elemento que está seleccionado y que ocupa la posición i dentro de la colección de elementos seleccionados.

Next indica que continúe el bucle o la cuenta que estamos llevando con la variable i. De esta forma, al haber usado el Step -1, lo que hacemos es contar hacia atrás y si, por ejemplo, i valía 3, al llegar aquí, valdrá 2, es decir, restamos 1: el valor indicado en Step

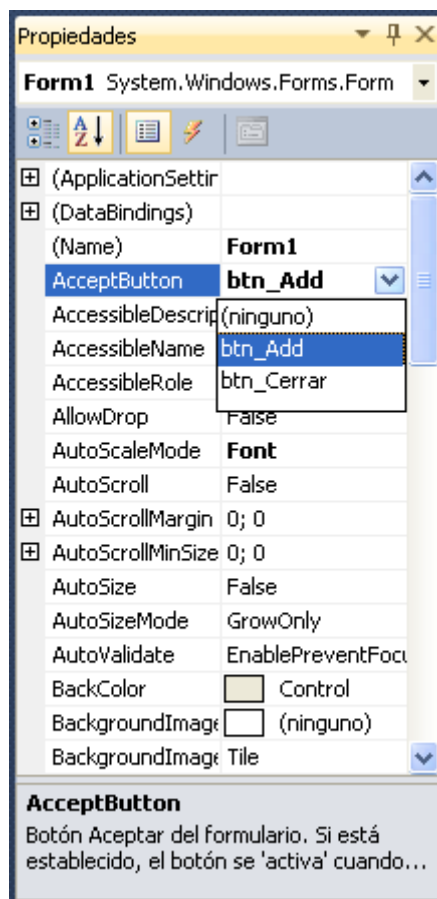
End With indica hasta dónde llega el With ListBox1

End If le dice que hasta aquí llega la comprobación que hicimos sobre si la tecla pulsada era la de suprimir.

Recordemos un poco otra vez lo del With.

Por tanto, si usamos **With Objeto**, podemos sustituir a **Objeto** por el punto, siempre y cuando ese punto, (y la propiedad o método correspondiente), esté dentro del par With... End With.

Nos queda decirle al formulario que cuando se pulse el botón de aceptar haga lo mismo que si pulsamos el botón de Añadir. Para esto nos vamos al diseñador y buscamos la siguiente propiedad:



Esto es qué debe hacer cuando se pulse el botón Intro. Normalmente cuando trabajamos con formularios en cualquier programa, podemos movernos por los distintos elementos pulsando la tecla del tabulador. Además, la tecla de “intro” suele estar asociada a alguna acción de ese formulario: cerrar, añadir un elemento, guardar datos,...

Lo que queremos hacer en nuestro ejemplo es asociar el código de añadir el elemento con un botón de los que hemos puesto nosotros: *Añadir*. Como vemos en la imagen anterior, al pinchar en este desplegable el entorno es lo suficientemente inteligente como para mostrarme la lista botones que tengo en mi formulario: selecciono entonces *btn_Add*.

Veamos ahora cómo le asociamos que al pulsar la tecla escape (*Esc*) termine el formulario. Para esto cambiamos la propiedad **CancelButton**.

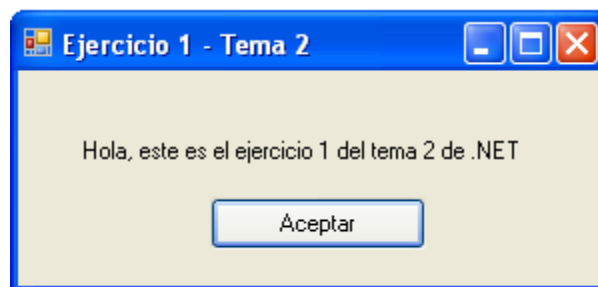
Por tanto, selecciona de la lista desplegable el botón *btn_Cerrar* y así al pulsar *Esc* se cerrará la aplicación.

Probemos ahora todo el código y así lo comprobarás: Escribe algo en el cuadro de texto y pulsa Intro, verás que se añade a la lista, después pulsa ESC y verás que se cierra el formulario o ventana con lo que se da por terminada la aplicación.

Ejercicios

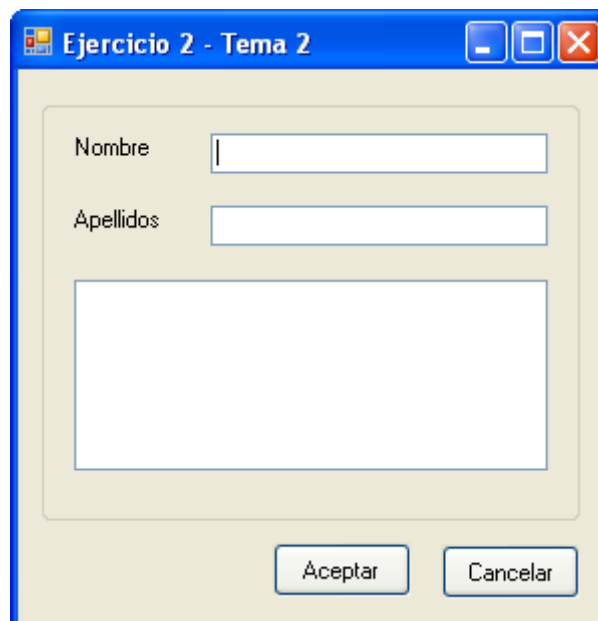
Ejercicio 1


Crea una nueva aplicación de Windows que muestre un mensaje de bienvenida al ejecutarse:



Ejercicio 2

Crea un formulario de este tipo anclando correctamente el cuadro de lista, los botones y cuadros de texto, para que al cambiar de tamaño mantenga las posiciones:

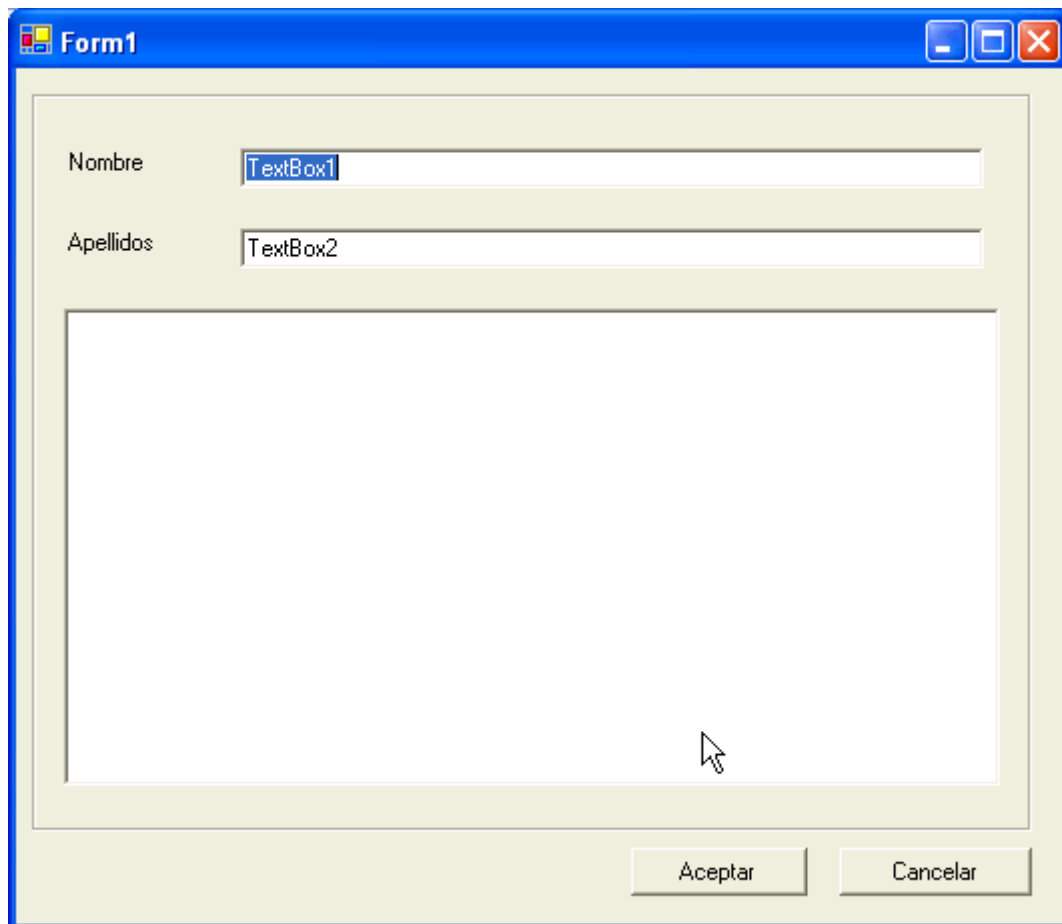




Form1

Nombre

Apellidos



Form1

Nombre

Apellidos

Nota: Para el marco interior utiliza un control llamado GroupBox y borra el texto de su propiedad Text.

Ejercicio 3

Haz un programa que haga estas acciones:

1. Que vaya insertando los elementos que escribimos en un cuadro de texto.
2. Cuando insertemos un elemento debe limpiar el cuadro de texto.
3. Cuando seleccionemos un elemento debe aparecer en el cuadro de texto "Seleccionado ahora"
4. Poner dos botones, uno para borrar el elemento seleccionado y otro para borrar todos.

