Trabajo Practico Windows Forms Paso a Paso

Desarrolle los siguientes Ejemplos de Desarrollo de aplicaciones de escritorio con Windows Forms:

Ejemplo 1

En este ejemplo, veremos cómo construir una aplicación para escritorio usando Windows Forms.

Para crear este ejemplo, iniciaremos el Visual Studio .NET y crearemos una aplicación de Windows usando el lenguaje de programación C#, por tanto empezaremos cargando el IDE de Visual Studio .NET y seleccionaremos este tipo de aplicación, veamos los pasos a seguir para conseguir nuestro objetivo:

1- Iniciamos el Visual Studio .NET

2- En el menú de Archivo, seleccionamos Nuevo... y del menú mostrado, seleccionaremos Proyecto... (también podemos usar el acceso rápido Ctrl+N)

3- Se mostrará un cuadro de diálogo como el mostrado en la figura 1:

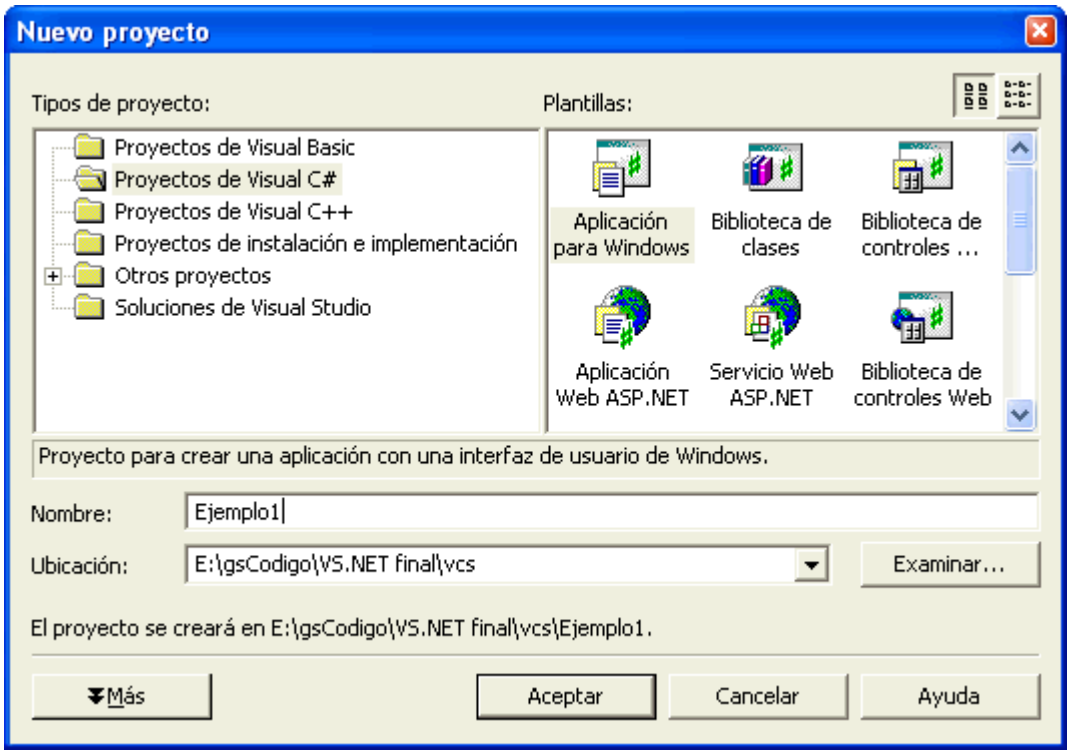

Figura 1: Nuevo proyecto de Aplicación de Windows (C#)

4- En el nombre indicaremos: **Ejemplo1**, tal como se muestra en la figura anterior. Ese será el nombre que Visual Studio .NET utilizará como nombre del ejecutable.

5- Cada vez que creamos un nuevo proyecto de aplicación para Windows, el entorno de desarrollo (IDE) añade un nuevo formulario, el cual, por defecto, tendrá el nombre Form1. Para este ejemplo, vamos a dejar el nombre indicado, en otra ocasión veremos lo que tendremos que hacer para cambiar el nombre asignado de forma predeterminada por el entorno de desarrollo, y utilizar el que deseemos.

6- A este formulario vamos a añadirle una serie de controles con los que interactuaremos para aprender las características más básicas que usaremos en la mayoría de las aplicaciones de Windows Forms (aplicaciones de Windows). Estas mismas acciones serán las más comunes en la mayoría de las aplicaciones de .NET Framework: Los eventos o mensajes producidos por los controles.

7- Los controles que añadiremos serán: dos etiquetas (Label), una caja de textos (TextBox) y dos botones (Button).

8- Para añadir un nuevo control al formulario, tendremos que mostrar el cuadro de herramientas situado, de forma predeterminada, en la parte izquierda del entorno de desarrollo. Si no lo tuviéramos visible podemos mostrarlo pulsando la combinación de teclas: Ctrl+Alt+X o bien seleccionando ***Cuadro de herramientas*** del menú ***Ver***.

9- Seleccionaremos la ficha ***Windows Forms*** y de los controles mostrados, haremos doble pulsación (doble-click) en el icono del control que queremos añadir, en esta ocasión empezaremos por un control de tipo Label.

10- A continuación añadiremos un control TextBox el cual posicionaremos junto a la etiqueta añadida anteriormente.

11- Seguiremos añadiendo el resto de controles, para que el aspecto sea el mostrado en la figura 2:

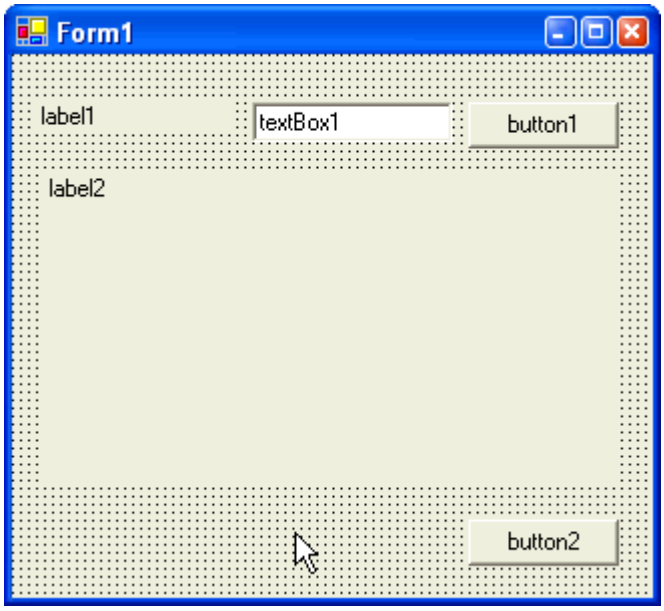

Figura 2: El formulario en tiempo de diseño

12- Ahora vamos a asignar los valores a unas cuantas propiedades de los controles, para ello seleccionaremos el control que queremos modificar y mostraremos la ventana de ***Propiedades***, si no la tenemos visible, pulsaremos F4.

13- Empezaremos por la primera de las etiquetas, (label1), por tanto, la seleccionaremos y pulsaremos F4, de forma predeterminada el cursor estará en la propiedad Text, en ella escribiremos el texto a mostrar en esa etiqueta, en esta ocasión será: Saludo; tal como podemos ver en la figura 3:

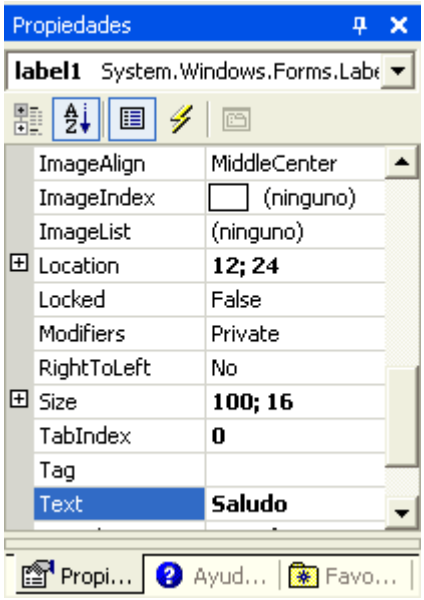

Figura 3: La ventana de propiedades

14- Seguiremos asignando el texto del botón que está junto a la caja de textos (button1) y lo cambiaremos a Saludar.

15- Haremos lo mismo con el otro botón (button2), al que asignaremos el texto Cerrar.

16- Para que los controles se adapten al tamaño del formulario, vamos a asignar la propiedad **Anchor** de los mismos, de forma que tengan los siguientes valores:

Control	Valor de Anchor	Comentarios
label1	Top, Left	Valores predeterminados
textBox1	Top, Left, Right	Para que se adapte al ancho del formulario
button1	Top, Right	Para anclarlo a la derecha
label2	Top, Bottom, Left, Right	Para que se adapte al ancho y alto
button2	Bottom, Right	Para anclarlo en la esquina inferior derecha

17- Una vez asignados estos valores, si cambiamos el tamaño del formulario, comprobaremos que los controles se adaptan al tamaño del mismo.

18- Lo siguiente que haremos será asignar los eventos que vamos a interceptar, en este caso sólo interceptaremos la pulsación (click) de los dos botones.

19- Para escribir el código correspondiente al evento Click, podemos hacerlo de dos formas diferentes, pero la más sencilla (y válida tanto para C# como para Visual Basic .NET) es hacer doble pulsación sobre el botón, de esa forma se mostrará la ventana de código con la "plantilla" del procedimiento de evento.

20- En el caso de que estemos usando C# (como es el caso de este ejemplo), podemos asignar el evento de la siguiente forma:

- Seleccionamos el control, (por ejemplo button1),
- mostramos la ventana de propiedades (pulsa F4 si no está visible),
- seleccionamos el símbolo del rayo amarillo, (ver la figura 4),
- de forma predeterminada estará seleccionado el evento Click, (si no lo estuviera, habría que seleccionarlo),
- hacemos doble pulsación y se mostrará la "plantilla" del código.

Nota:
Si quisiéramos capturar otro evento, simplemente habría que seleccionarlo y hacer doble pulsación.

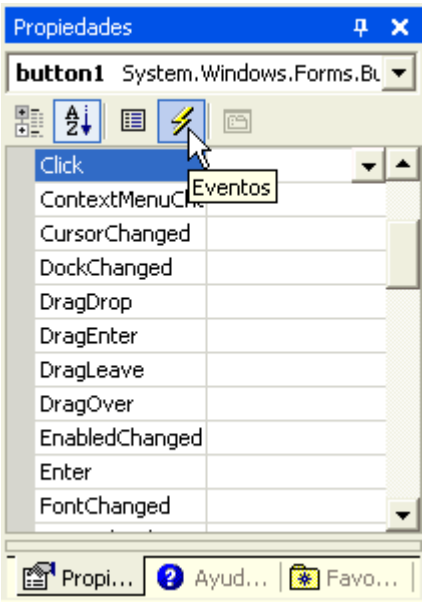

Figura 4: Los eventos de los controles en C#

21- Además de la plantilla del método (o función) que será llamado cuando se produzca el evento, en el caso de C#, también se creará una asignación para indicar cual es el manejador (handler) de dicho evento.

A continuación se muestra el código para asignar el método que se usará para interceptar el evento, así como el código a ejecutar cuando éste se produzca; en este caso, lo que hacemos es mostrar en la etiqueta label2 un saludo usando el contenido de la caja de textos.

```
this.button1.Click += new System.EventHandler(this.button1_Click);


private void button1_Click(object sender, System.EventArgs e)
{
   label2.Text = "Hola, " + textBox1.Text;
}
```

22- Debido a que C# tiene en cuenta la diferencia entre mayúsculas y minúsculas, podemos usar "**this**" para no cometer errores tipográficos, en la siguiente figura, vemos cómo al escribir **this** seguida de un punto, se muestran las propiedades, métodos y controles que pertenecen al formulario, ya que **this** representa a la clase actual.

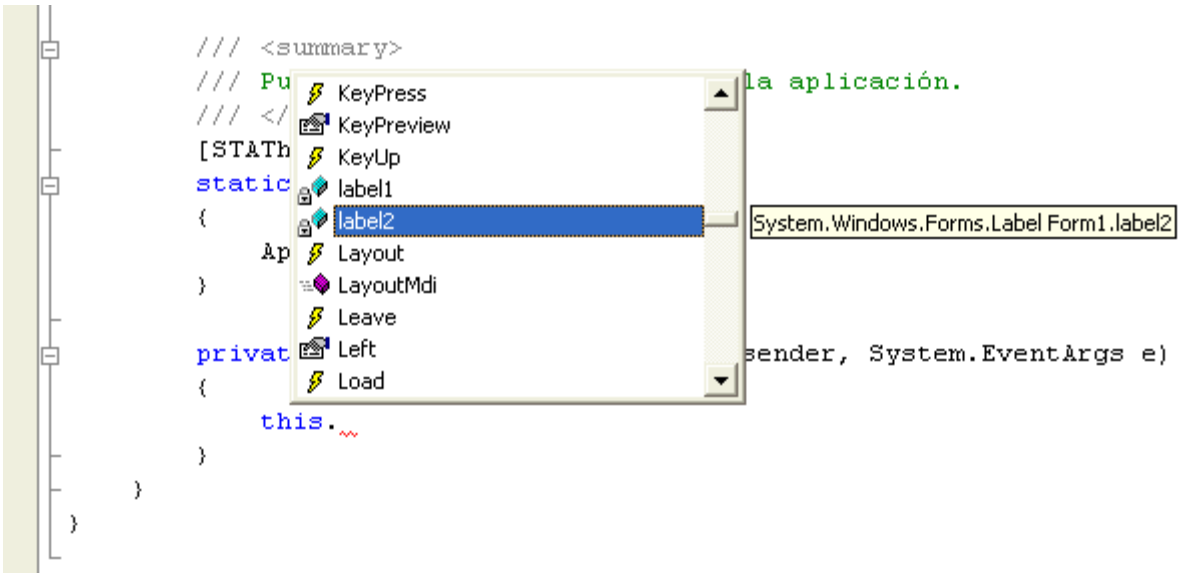

Figura 5: Usar Intellisense para ver los miembros de una clase.

23- Por último, crearemos el código para interceptar el evento Click del segundo botón, (button2), de forma que cuando se procese, cerremos el formulario y de paso se termine la ejecución del programa.

El código sería el siguiente:

```
private void button2_Click(object sender, System.EventArgs e)
{
   this.Close();
}
```

24- Ahora podemos comprobar que todo funciona bien, para ello, pulsaremos F5 para que se compile y ejecute el proyecto.

Ejercicio:

A continuación te propongo un pequeño ejercicio, el cual consiste en interceptar el evento predeterminado de las cajas de texto: TextChanged, de forma que se muestre en la etiqueta (label2) el contenido de la caja de textos (textBox1) cada vez que se produzca dicho evento o lo que es lo mismo: cada vez que cambie el contenido de la caja de textos.

RESOLUCION

Para interceptar el evento predeterminado de un control y hacer que se muestre la pantalla de código; la forma más simple de conseguirlo es haciendo doble click en el control en cuestión, en este caso será sobre la caja de textos.

También podemos crear el manejador de un evento usando la ventana de propiedades, para ello, tendremos que seleccionar el control en cuestión, pulsar F4 para mostrar la ventana de propiedades, seleccionar el icono del rayo amarillo (ver figura 4) y de la lista mostrada, seleccionar el evento que queremos interceptar (en este caso será TextChanged) y hacer doble pulsación sobre el nombre de dicho evento, de esta forma tendremos la plantilla del evento, en la que escribiremos el

siguiente código para conseguir que se muestre el mensaje cuando el contenido de la caja de textos se modifique.

```
private void textBox1_TextChanged(object sender, System.EventArgs e)
{
   label2.Text = "Hola, " + textBox1.Text;
}
```

Si te fijas, el código a usar es el mismo que el utilizado en el evento Click del botón Saludar (button1), ya que esa es la intención: mostrar el saludo conforme se va escribiendo.
Si pulsas F5, verás que el texto mostrado en la etiqueta irá cambiando conforme escribas en la caja de textos.

Ejemplo 2

En este segundo ejemplo vamos a crear una aplicación de Windows Forms en la que tendremos en cuenta ciertas de las características que ya vimos en el ejemplo anterior, como son la adaptación de nuestros controles al tamaño del formulario, además de utilizar controles contenedores en los que insertar los distintos controles que formarán parte del diseño de nuestra aplicación.

Los pasos a seguir para crear este ejemplo serán los que habitualmente utilizaremos en cualquier aplicación creada con Visual Studio .NET y que son los mismos mostrados en el ejemplo anterior, por tanto no voy a explicar cómo crear una nueva aplicación ni como insertar los controles en el formulario, sólo veremos las "novedades" con respecto al mencionado ejemplo anterior. Lo que si haré es indicar esos pasos.

1- Iniciamos el Visual Studio .NET

2- Creamos un nuevo proyecto de Windows Forms (aplicación de escritorio) en el lenguaje de nuestra predilección.

3- Como ya hemos podido comprobar, cada vez que creamos un nuevo proyecto de aplicación para Windows, se añade un formulario cuyo nombre será Form1.

4- Si queremos que el nombre del formulario sea otro del que el entorno de desarrollo asigna por defecto, debemos seguir estos pasos:

5- Seleccionamos el archivo (formulario) en el explorador de soluciones y le cambiamos el nombre. Con esto lo único que conseguimos es que el nombre del archivo sea distinto, pero no cambia el nombre de la clase que tendrá ese formulario.

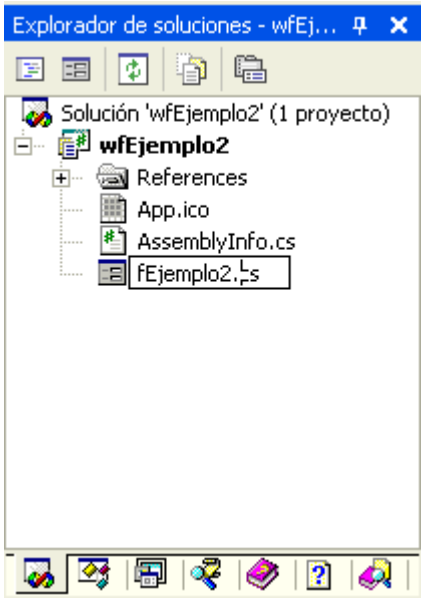

Figura 1, El explorador de soluciones

6- Debemos mostrar el código del formulario y cambiar todos los **Form1** por el nombre que queramos que tenga nuestra clase.

Nota:
Hay que tener presente de que parte del código está oculto, sobre todo la parte que genera el diseñador de formularios, el cual suele estar en una sección denominada "Windows Form Designer generated code".

```
        }

        Windows Form Designer generated code

        /// <summary>
        /// Punto de entrada principal de la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.Run(new fEjemplo2());
        }
    }
}
```

Figura 2, Parte del código generado estará oculto.

7- Si el proyecto lo hemos creado usando Visual Basic .NET, tendremos que indicarle cómo se llama el objeto inicial de nuestra aplicación de Windows. Para cambiar ese "objeto inicial" tendremos que mostrar el cuadro de diálogo del proyecto y seleccionar de la lista desplegable el nuevo formulario que queramos que tenga la aplicación como formulario de inicio.

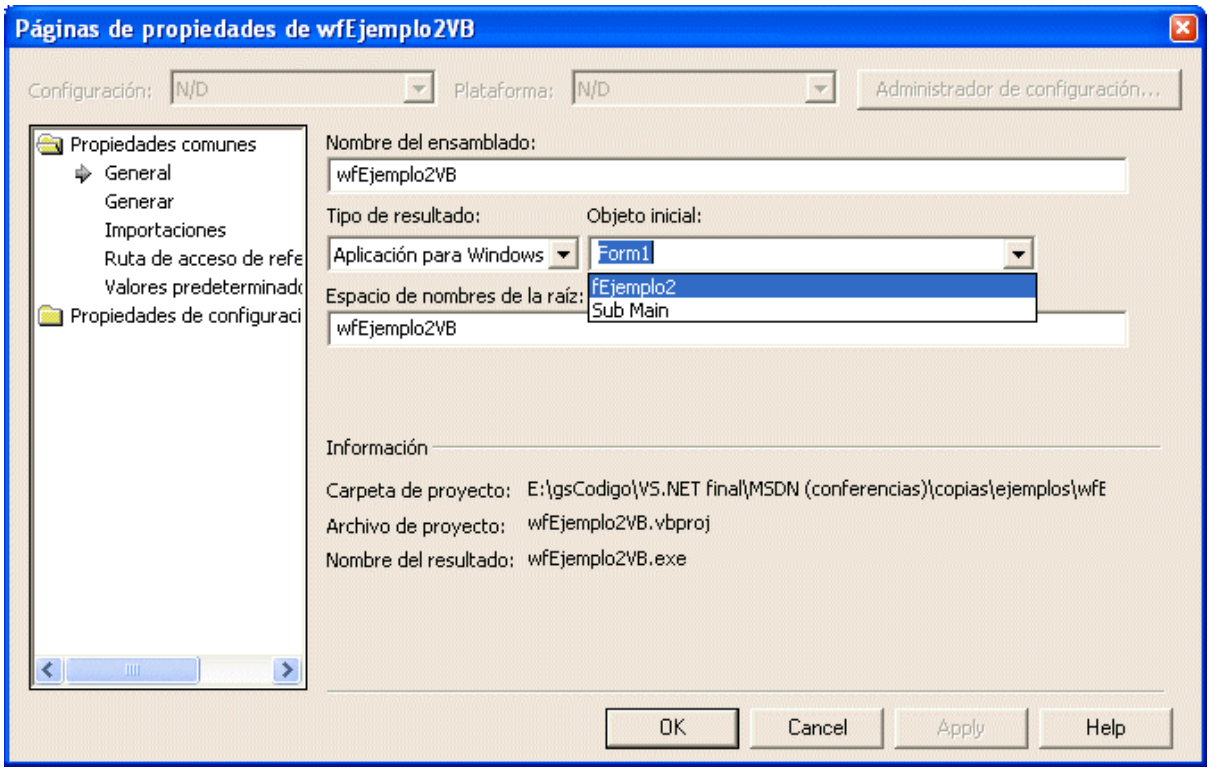

Figura 3, En Visual Basic .NET hay que cambiar también el "objeto" inicial.

8- A continuación añadimos los controles que necesitemos y los posicionaremos en el formulario.

Nota:
Recuerda asignar los valores de la propiedad **Anchor** de cada uno de esos controles para que se adapten al nuevo tamaño que el usuario de nuestra aplicación quiera darle a la ventana, (siempre que así lo creas conveniente).
No te voy a indicar los valores a asignar, simplemente te mostraré los controles que tendrás que incluir en el formulario.

Lo único que te indicaré "paso a paso" será la forma de incluir controles dentro de las fichas (o tabs) que usaremos en este ejemplo.

9- En el formulario de este ejemplo vamos a usar un control **Tab** para incluir varios controles en las distintas fichas, cada una de las fichas podrán contener otros controles, pero la forma que ahora tiene el .NET Framework para manejar esos contenedores es diferente a como antes lo hacíamos (si es que has programado anteriormente con los controles comunes de Windows en Visual Basic, por ejemplo).
Estos controles que hacen de contenedor de otros controles, tienen la posibilidad de poder mostrar de forma automática las barras de scroll, (tanto horizontal como vertical), cuando sea necesario, para ello asignaremos un valor verdadero (True) a la propiedad **AutoScroll**.

Veamos el aspecto del formulario con los controles y después entraremos en detalles:

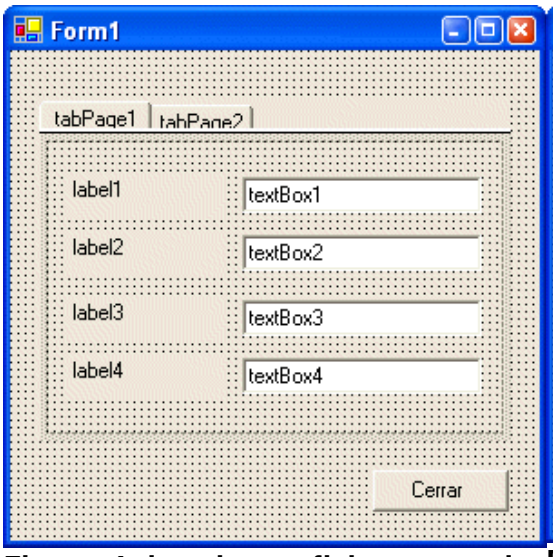

Figura 4, la primera ficha con varios controles

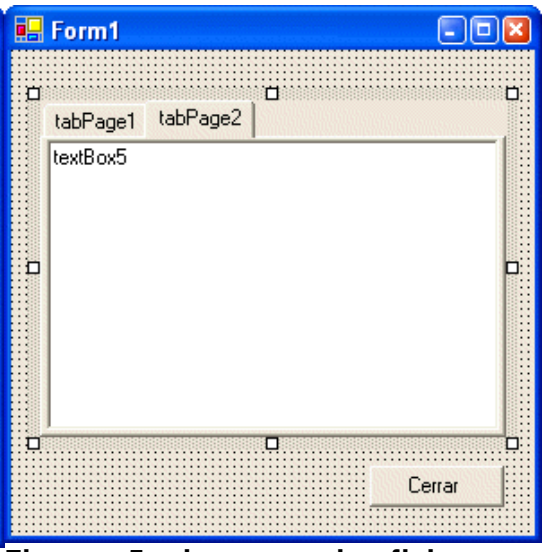

Figura 5, la segunda ficha con TextBox multilínea

10- El formulario contiene un control **TabControl** con dos fichas (**TabPage**) y un botón.
La primera ficha contiene 4 etiquetas y 4 cajas de texto.
La segunda ficha contiene una caja de textos, con los siguientes valores en las propiedades: **MultiLine** = True y **Dock** = Fill, de esta forma, la caja de texto ocupará todo el espacio de la ficha que lo contiene y permitirá que se muestren varias líneas de código.

En la primera ficha asignaremos un valor True a la propiedad **AutoScroll**, de forma que si el tamaño del formulario es pequeño, se muestren las barras de desplazamiento (scroll bars) que sean necesarias.

11- Otra de las propiedades que utilizaremos en nuestro formulario será la que nos permita controlar el tamaño más pequeño que podrá tener. Para conseguir eso, asignaremos a la propiedad **MinimumSize** el valor 200; 200, esto indica que tanto el ancho como el alto mínimo que queremos para nuestro formulario será de 200 puntos.
Cuando estemos en tiempo de diseño o en ejecución, no podremos asignar un valor menor al indicado, con lo cual nos aseguramos de que, como mínimo ese sea el tamaño del formulario.

12- Ahora veamos cómo crear nuevas fichas en el TabControl, (ya que cuando añadimos un control de fichas, por defecto no contiene ninguna).

13- La mejor forma es seleccionando dicho control y pulsando con el botón secundario del ratón, de esta forma se mostrará un menú contextual, entre cuyas opciones está la de *Agregar ficha*, tal como vemos en la siguiente figura:

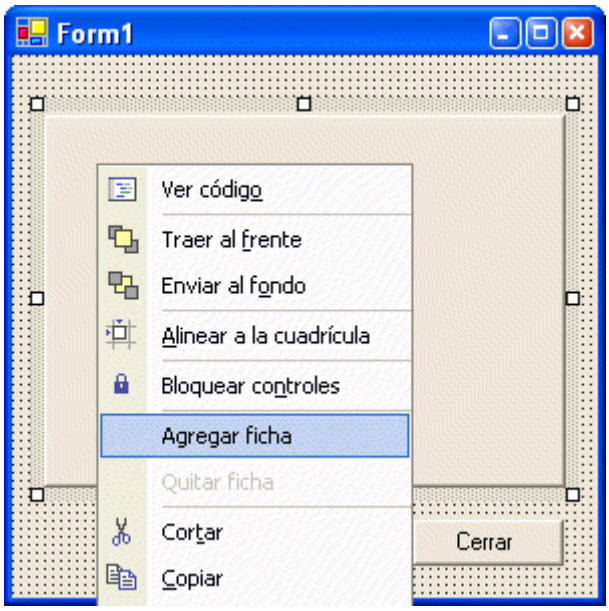

Figura 6, Menú contextual para agregar nueva ficha al TabControl

14- Para añadir una nueva ficha seguiremos el mismo procedimiento o también podemos hacerlo desde la ventana de propiedades, bien seleccionando el link *Agregar ficha* o pulsando en los tres puntos que hay junto a la propiedad **TabPages** de dicha ventana de propiedades, tal como podemos comprobar en la siguiente figura:

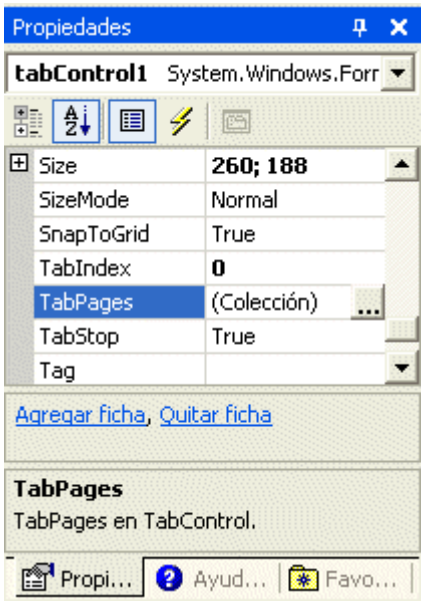

Figura 7, propiedades del control TabControl

15- Para añadir controles a cualquiera de las dos fichas, haremos lo siguiente:

16- Seleccionaremos la ficha en la que queremos añadir los controles, en este caso será la primera. A la que añadiremos 4 etiquetas y 4 cajas de texto, tal como comenté al principio.

Nota:
Una vez que hayamos añadido una etiqueta y una caja de textos, podemos seleccionarlas para copiarlas, de esta forma, simplemente pegando los dos

controles copiados tendremos otros dos controles, así hasta conseguir el total de 4 pares de etiquetas y cajas de texto.

17- Estos controles podemos asignarle los valores a la propiedad Anchor de forma que se adapten o "anclen" al contenedor en el que están insertados.

18- Una vez que tenemos todos los controles en el formulario y las fichas (tabs), podemos comprobar si las asignaciones realizadas tanto a las propiedades del formulario como a la de los demás controles funcionan como esperábamos, para comprobarlo, podemos cambiar el tamaño del formulario en tiempo de diseño y si nos satisface las asignaciones realizadas, podemos probarlo en tiempo de ejecución, para comprobar que todo funciona igualmente bien.

19- El único código que vamos a escribir en el formulario será el correspondiente al evento **Click** del botón Cerrar (button1), para insertar el código, haz doble click sobre ese control y se mostrará la ventana de código y escribe lo siguiente según el lenguaje que hayas usada para crear la aplicación.

En C# sería:

```
private void button1_Click(object sender, System.EventArgs e)
{
   this.Close();
}
```

En Visual Basic .NET sería este otro:

```
Private Sub button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles button1.Click
   Me.Close()
End Sub
```

Nota:
En definitiva se llama al método **Close** de la instancia actual, (el objeto creado en la memoria a partir de la clase formulario), y en ambos casos se podría haber prescindido de la instrucción que referencia a la instancia actual de un objeto: **this** (C#) o **Me** (VB)

En la siguiente tabla te muestro los valores de cada una de las propiedades que hemos modificado para que los controles se adapten al nuevo tamaño del formulario (o el contenedor), así como el resto de las propiedades en las que se han hecho cambios.

Nota:
Cuando no se indique un cambio en alguna de las propiedades, querrá decir que se ha dejado el valor que el entorno de desarrollo le da de forma predeterminada. En el caso de que hayas cambiado alguna propiedad y quieras que vuelva a tomar el valor "por defecto", haz lo siguiente:

-Selecciona el control en el que quieres restablecer el valor de una propiedad.
-Selecciona la propiedad que quieres restablecer.

-Pulsa en el botón secundario del ratón y del menú desplegable que se mostrará, selecciona la primera opción: Restablecer, si dicha opción estuviese deshabilitada, significa que el valor que contiene es el que tenía originalmente.

Control (nombre)	Propiedad	Valor
Formulario (fEjemplo2)	MinimumSize	200; 200
TabControl (tabControl1)	Anchor	Top, Bottom, Left, Right
TabPage (tabPage1)	AutoScroll	True
Los 4 TextBox de tabPage1	Anchor	Top, Left, Right
El TextBox de tabPage2	Docking	Fill
El botón Cerrar	Anchor	Bottom, Right

Ejemplo 3

En este tercer ejemplo "paso a paso" sobre Windows Forms, vamos a crear controles personalizados, así como una aplicación que utilice esos controles creados por nosotros.

Los controles que podemos crear con Visual Studio .NET, (realmente con .NET Framework), pueden ser de tres tipos:
1- Usando herencia, de esta forma aprovechamos toda la funcionalidad de un control existente al que sólo tendremos que indicarle el código necesario para que haga las cosas que nosotros queremos que haga y que no estén implementadas en el control que queremos "personalizar"... por ejemplo para que sólo acepte números.
2- Usando varios controles ya existentes para poder crear un "control de usuario" (o control compuesto). Esta sería la forma que "antes" se usaba para poder crear nuestros propios controles. Esta forma de personalizar los controles es útil cuando necesitamos más de un control, por ejemplo, si queremos crear un control que contenga una etiqueta y una caja de textos. Esto es así, ya que con .NET Framework no podemos usar la herencia múltiple y por tanto no podríamos crear una nueva clase que herede las clases Label y TextBox.
3- Creando el control a partir de cero, es decir, nos tenemos que encargar de **todos** los pormenores y detalles, por ejemplo, si el control va a mostrar algo de forma gráfica, nos tendremos que encargar de dibujar el control cuando el sistema operativo nos avise de ese "detalle", normalmente la forma de avisarnos de que hay que dibujar el control, es mediante el evento **Paint**.

En este ejemplo, los controles los vamos a crear usando herencia, para que comprobemos lo fácil que es crear un control personalizado, de forma que se adapte a nuestras necesidades o preferencias.
En concreto vamos a crear un TextBox personalizado para que sólo acepte números.
Como ya comenté en el párrafo anterior, usando la herencia nos aprovechamos de todas las características que ya tenga el control en el que nos vamos a basar, por tanto sólo tendremos que escribir el código que nos permita personalizar el control a nuestras preferencias, el resto del código "normal" ya lo tendremos heredado del control que hemos utilizado como base; debido a que el control que vamos a crear se basa en un control TextBox, todas las características de ese control lo vamos a

tener a nuestra disposición, lo único que tendremos que escribir o codificar son las características que queremos personalizar. Como el control que vamos a crear es una caja de textos que sólo admita números, vamos a personalizar el procedimiento que detecta la pulsación de teclas (OnKeyPress), para que se acepten sólo las pulsaciones que nosotros queramos considerar como números; también vamos a codificar la propiedad Text, que será la que se usará cuando asignemos un nuevo valor a la caja de textos, de forma que se compruebe si el valor asignado realmente representa sólo números.

Los pasos a seguir para crear este ejemplo serán los mismos mostrados en los ejemplos anteriores, por tanto sólo indicaré las cosas que realmente sean diferentes a otros tipos de proyectos. De todas formas, aunque no se indique de forma explícita, voy a indicar cuales serán esos pasos.

1- Iniciamos el Visual Studio .NET

2- Creamos un nuevo proyecto del tipo **Biblioteca de controles Windows** usando el lenguaje de nuestra predilección.

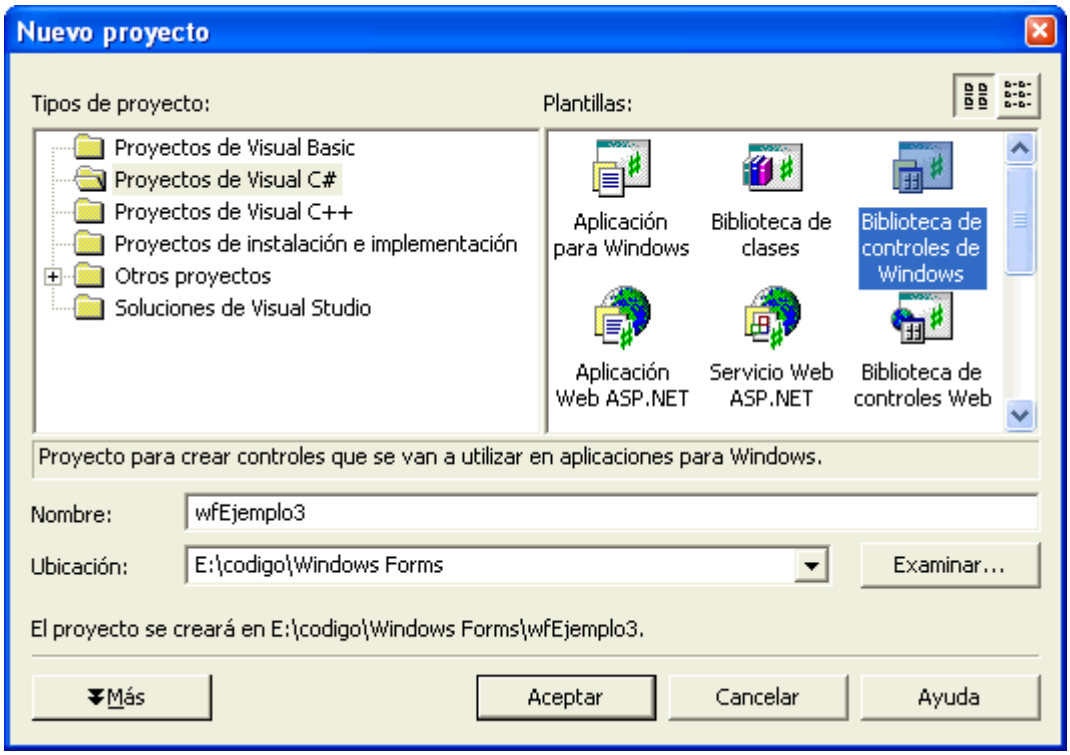

Figura 1, Diálogo para Nuevo proyecto

3- Cuando creamos un proyecto del tipo Biblioteca de controles de Windows, se añade un contenedor del tipo Control de usuario, este sería el tipo de control que tendríamos que usar para crear controles compuestos por controles ya existentes, esta sería la forma más parecida a la usada cuando creábamos controles ActiveX. Ese control no lo necesitaremos para crear nuestro control "heredado", por tanto, iremos al Explorador de soluciones, seleccionaremos el archivo UserControl1.cs (si el idioma seleccionado es C#, en caso de que sea otro el lenguaje usado, la extensión del archivo será diferente) y lo eliminaremos.

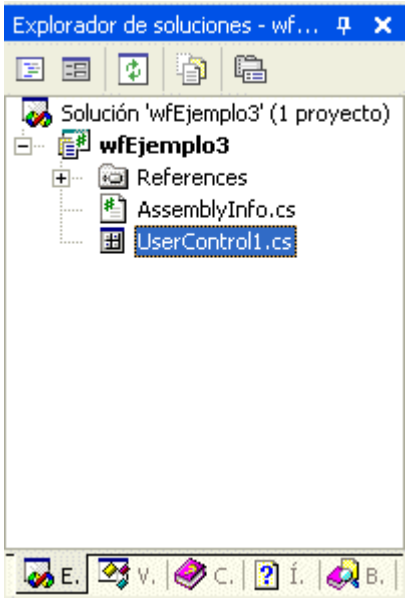

Figura 2, El explorador de soluciones

4- El siguiente paso es agregar una clase al proyecto, para ello iremos al menú Proyecto y seleccionaremos la opción "Agregar clase..."

5- Se mostrará un cuadro de diálogo como el mostrado en la figura 3, ahí le indicaremos que es una clase y el nombre será: **TextBoxNum.cs**, (recuerda que la extensión dependerá del lenguaje que estemos usando, en caso de Visual Basic .NET, la extensión sería .vb).

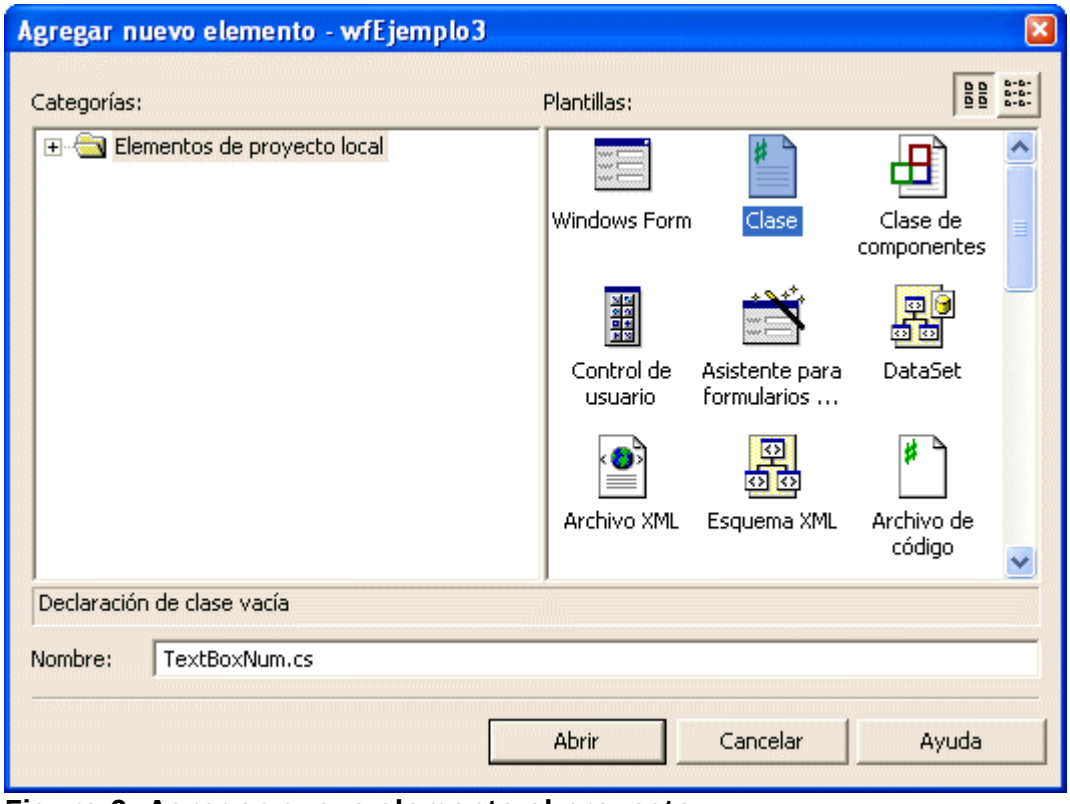

Figura 3, Agregar nuevo elemento al proyecto

6- Al añadir una nueva clase, se mostrará la plantilla de esa nueva clase, como nosotros queremos crear una clase derivada de la clase TextBox, (la clase caja de

textos), en la declaración de la clase debemos indicarle que la clase TextBoxNum se deriva de TextBox, para hacer esto, tendremos que escribir lo siguiente:

En C# la declaración de la clase quedaría así:
```
public class TextBoxNum : TextBox
```

Esta es la forma de indicar la clase de la que se hereda en el lenguaje C#.

En Visual Basic .NET la declaración de la clase sería de esta otra forma:
```
Public Class TextBoxNum : Inherits TextBox
```

7- El siguiente paso será escribir el código para implementar nuestras preferencias, como ya indiqué al principio, este control heredado servirá para aceptar sólo números, por tanto vamos a crear una función que "filtre" cada uno de los caracteres que se vayan escribiendo en la caja de textos o bien cuando se asigne un nuevo valor a la propiedad Text. Por tanto sólo vamos a escribir la susodicha función, además del procedimiento **OnKeyPress** que será el que llame el .NET cuando el usuario pulse una tecla, también vamos a escribir el código de la propiedad **Text**.

8- Antes de escribir el código real de nuestro control, tenemos que indicarle al compilador dónde encontrar la clase que vamos a usar de base de nuestro control. Como sabemos todos los controles para aplicaciones Windows están dentro del espacio de nombres **Windows.Forms**, por tanto necesitamos agregar una referencia (o importación) al espacio de nombres Windows.Forms, para ello lo indicaremos de la siguiente forma:
```
using System.Windows.Forms;
```
(recuerda que este sería el código para C#)

Nota:
Si no añadimos esta importación del espacio de nombres Windows.Forms, al compilar se producirá un error de que no se ha encontrado la clase TextBox.

9- Empecemos viendo el código de la función que comprueba si el carácter pulsado es de los que vamos a aceptar.

```
// array con los dígitos aceptados por el textbox numérico
char[] digitos = new char[]{'0','1','2','3','4','5','6','7','8','9',
                            '.',',','-','\b'};
// esta función permite controlar si el carácter es de los admitidos
protected virtual bool CaracterCorrecto(char c)
{
   // devolverá true si el carácter está en el array
   return (Array.IndexOf(digitos, c) != -1);
}
```

Nota:
La función **CaracterCorrecto** está declarada como **protected virtual**; cuando declaramos una función con el modificador de acceso **protected** dicha función sólo será visible dentro de la propia clase y en clases derivadas de esta, por otro lado, el modificador **virtual** indica que esta función puede ser reemplazada por otra en una clase derivada, un ejemplo de esto último lo veremos a continuación.

Lo que aquí hacemos es crear un array (matriz) de tipo **char** con los dígitos que queremos aceptar: los números del 0 al 9, además del punto, la coma, el signo menos y la tecla de "borrar hacia atrás".

La función devuelve **true** o **false** según el carácter comprobado esté o no dentro del array. Para comprobar si dicho carácter está incluido en el array, usamos el método **IndexOf** de la clase **Array**, el cual comprueba si un carácter está incluido o no en el array indicado, en caso de que no se haya encontrado, se devolverá **-1**.

10- Cuando el usuario pulsa en una tecla, el sistema operativo se encarga de llamar al método **OnKeyPress** (entre otros) de la clase, éste método será el que produzca el evento **KeyPress** del control insertado en el formulario (o contenedor). Por tanto, necesitamos interceptar ese procedimiento para comprobar si la tecla pulsada es una de las que nosotros vamos a aceptar. Los métodos que interceptan los eventos de una clase, se declaran **protected** para que sólo sean accesibles desde la propia clase o de las clases derivadas, por tanto si reemplazamos dicho método en nuestra clase, podremos comprobar dicha tecla y aceptarla o no, de eso se encarga el método **OnKeyPress** que vamos a definir en nuestra clase derivada de **TextBox**, cuyo código es:

```
// el procedimiento que intercepta la pulsación de teclas
protected override void OnKeyPress(KeyPressEventArgs e)
{
   // Comprobar si aceptamos el carácter pulsado,
   // si el carácter pulsado no está en el array, no aceptarlo
   if( !CaracterCorrecto(e.KeyChar) )
      e.Handled = true;
   // es conveniente llamar a el procedimiento
   // del mismo nombre de la clase base
   base.OnKeyPress(e);
}
```

Este procedimiento, al estar declarado como **override**, reemplaza al procedimiento de mismo nombre de la clase base, por tanto, éste será el que se utilice al producirse la pulsación de una tecla.
Lo que hacemos es comprobar si no es uno de los caracteres aceptados y en caso de ser así, (se cumplirá la condición **if**), le asignamos un valor **true** a la propiedad **Handled** con lo que conseguimos que se ignore dicha pulsación.
Por último llamamos al método del mismo nombre de la clase base, por si esa clase hiciera a su vez otras comprobaciones.

11- Por último, vamos a reemplazar la propiedad **Text** para que también se comprueben los caracteres aceptables al asignar un nuevo valor a dicha propiedad. En este caso lo que haremos es recorrer cada uno de los caracteres de la cadena asignada y aceptar sólo los que sean correctos, por tanto llamaremos a la función **CarracterCorrecto** para que se encargue de hacer dicha comprobación.

```
// La propiedad Text "sobrescrita" para adaptarla
// a nuestra nueva implementación
public override string Text
{
   get
   {
      return base.Text;
   }
   set
   {
      // aceptar sólo dígitos
      string s = "";
      foreach(char c in value)
```

```
        {
            // si es un carácter correcto,
            // lo agregamos a la nueva cadena
            if( CaracterCorrecto(c) )
                s += c;
        }
        base.Text = s;
    }
}
```

El bloque **get**, que es el que se llama al recuperar el contenido de una propiedad, simplemente devuelve el contenido de la propiedad Text de la clase base, ya que no tenemos que hacer ninguna comprobación.
En el bloque **set**, que es el que se utiliza al asignar un nuevo valor a una propiedad, es donde hacemos lo comentado anteriormente, para ello recorremos cada uno de los caracteres de la cadena a asignar (indicada por **value**) y si dicho carácter es de los aceptables, lo asignamos a la variable **s**, de forma que al final del bucle, dicha variable contenga sólo los caracteres que hemos aceptado. Nuevamente usamos la misma propiedad de la clase base por si en dicha propiedad se hicieran algunas otras comprobaciones.

Nota:
Es recomendable que siempre que derivemos una clase y reemplacemos alguna propiedad, usar dicha propiedad de la clase base tanto para devolver el valor como para asignar el nuevo contenido, esto nos permitirá continuar la cadena, de forma que si en la clase base se hacen algunas comprobaciones, éstas no se pierdan. En nuestro código, si en lugar de usar: **base.Text = s;** hubiésemos devuelto directamente el valor de la variable **s**, seguramente no habría ningún problema, pero si alguien usa nuestra clase para crear una nueva caja de textos, las validaciones que hacemos al asignar a la propiedad Text se perderían si no simplemente devolviera el nuevo valor.
Además de que al usar la propiedad Text de la clase base como contenedor del valor asignado, nos evitamos tener que declarar una variable privada que se encargue de "contener" dicho valor. Imagínate lo que podría ocurrir si sólo devolvemos el valor de la variable **s** y no lo "conservamos": **¡Nunca se podría cambiar el contenido de dicha propiedad!**

12- Una vez que tenemos todo el código necesario para hacer que nuestra clase derivada de **TextBox** sólo acepte números, podemos compilarlo para que podamos usarlo en una aplicación de prueba.

13- Vamos a crear un proyecto de prueba. Para ello vamos a agregar un nuevo proyecto a nuestra "solución". Para conseguir nuestro objetivo, sigue estos pasos:

14- En el menú *Archivo*, selecciona *Agregar proyecto* y del menú mostrado, selecciona *Nuevo proyecto...*, tal como se muestra en la figura 4:

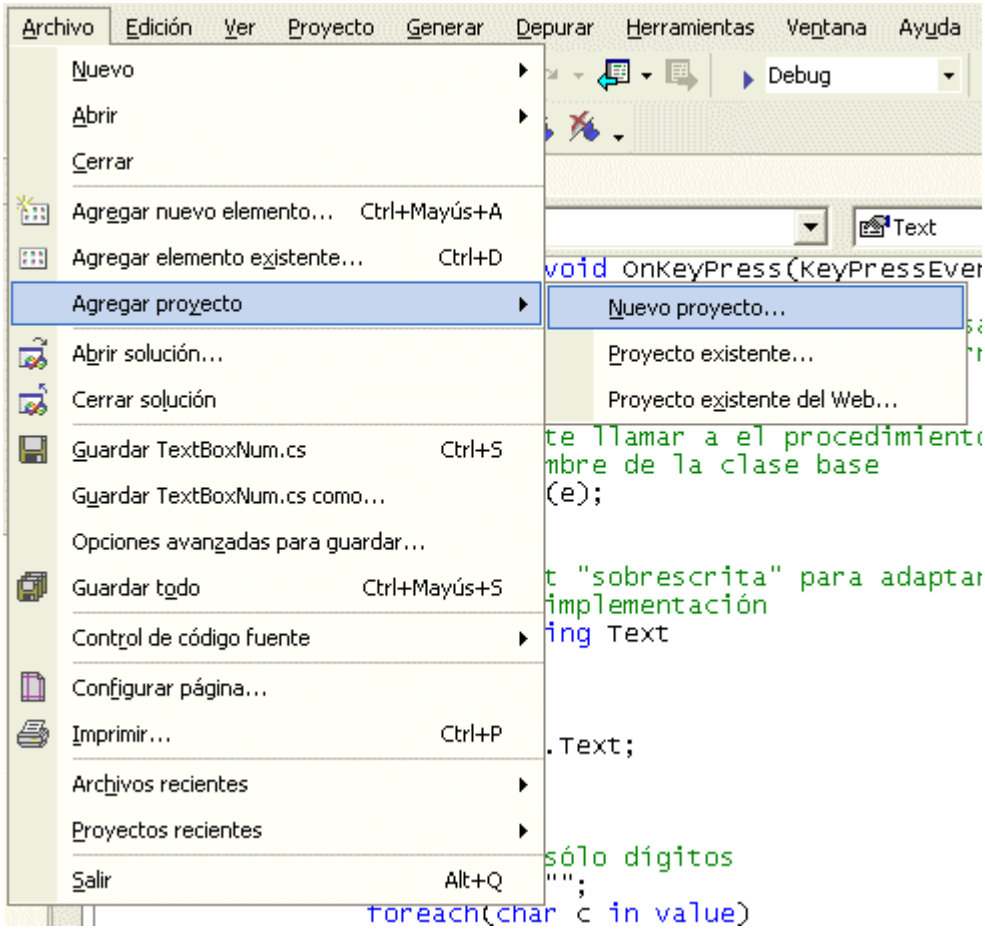

Figura 4, Agregar un nuevo proyecto a la solución existente

15- Se mostrará el cuadro de diálogo de selección de nuevo proyecto, del cual seleccionaremos Aplicación para Windows, esto agregará un proyecto con un formulario en el que añadiremos nuestro control.

16- Para que podamos agregar nuestro control al formulario, lo mejor será añadirlo al cuadro de herramientas (donde están los controles incluidos con Visual Studio .NET), para conseguirlo podemos hacerlo de varias formas. Debido a que VS.NET nos permite personalizar el cuadro de herramientas, vamos a crear nuestra propia ficha y en ella vamos a agregar el control TextBoxNum, veamos cómo:

Nota:
Si en lugar de agregar una nueva ficha, decides agregarlo a cualquiera de las existentes, te puedes saltar el siguiente paso.

17- Mostramos el *Cuadro de herramientas*, pulsamos con el botón secundario del ratón sobre él y del menú desplegable mostrado, seleccionamos *Agregar ficha*, tal como se muestra en la figura 5.

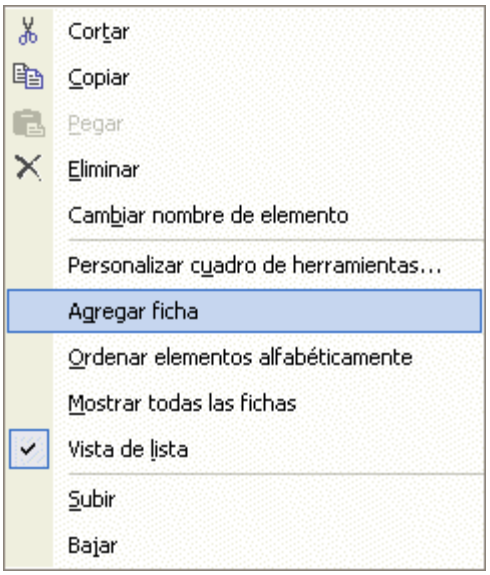

Figura 5, Opciones para personalizar el Cuadro de herramientas

Se agregará una nueva ficha a dicho Cuadro de herramientas (cuando agregamos una ficha, en la parte inferior del Cuadro de herramientas tendremos que escribir el nombre de la nueva ficha).

18- Ahora vamos a agregar nuestro control a la nueva ficha (o a la que esté en ese momento mostrada). Para ello, pulsa con el botón secundario del ratón sobre la ficha, del menú desplegable (mostrado en la figura 5), selecciona *Personalizar cuadro de herramientas...* Se mostrará un cuadro de diálogo con dos fichas, selecciona *Componentes de .NET Framework* y pulsa en el botón Examinar... Localiza el directorio del proyecto **TextBoxNum** y dentro del directorio **bin** estará la librería con nuestro nuevo control (la extensión será .DLL) Al seleccionarla, se mostrará dentro de la lista del cuadro de diálogo. Pulsamos aceptar y la tendremos disponible en la ficha del Cuadro de herramientas.

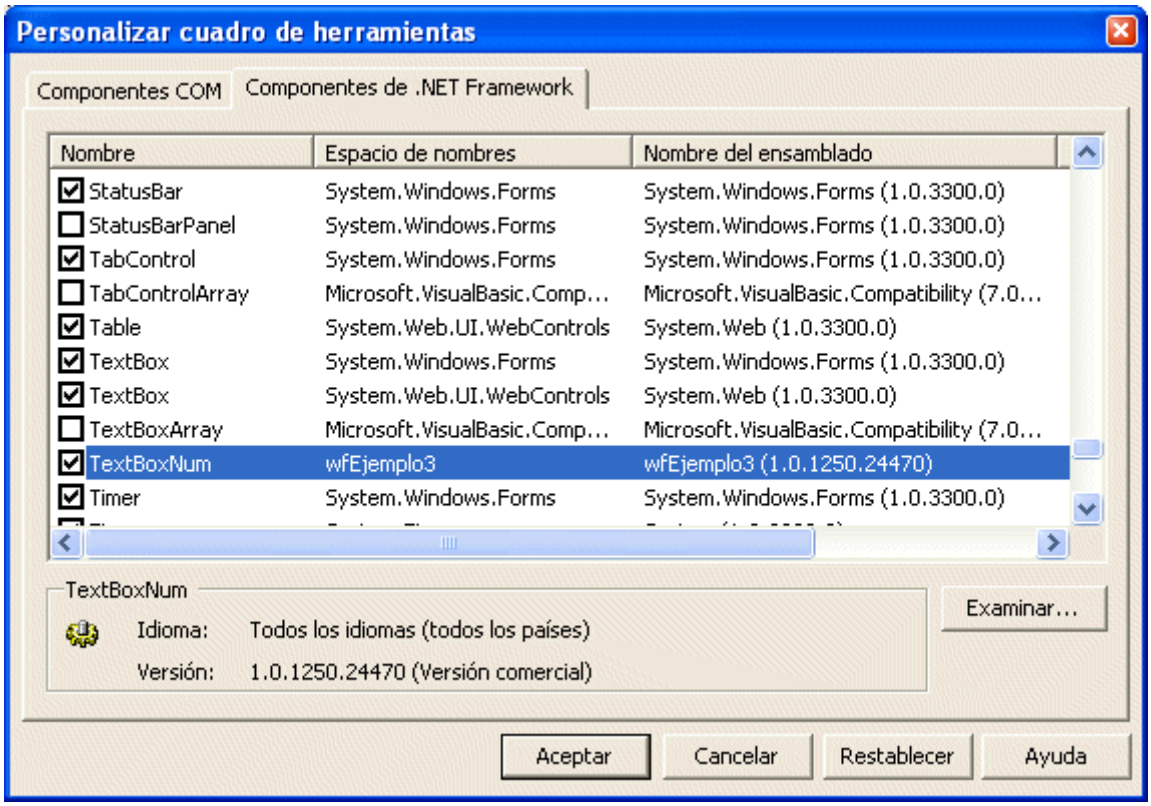

Figura 6, Personalizar cuadro de herramientas con el nuevo control

19- Una vez que tenemos nuestro control en el Cuadro de herramientas, ya podemos añadirlo al formulario como cualquier otro control.
Al agregar el control al formulario, en las referencias del proyecto se habrá agregado la referencia oportuna a la librería en la que se encuentra nuestra clase.

Como dato curioso, fíjate en el texto mostrado, debido a que el control no admite letras, sólo se mostrará el número 1, esto es así, porque por defecto, el contenido de la propiedad Text de las cajas de texto es el nombre del control y como el nuestro no admite letras, simplemente se queda con el número de **TextBoxNum1** (que es cómo se llamará el control agregado).

20- **Como ejercicio** podrías añadir una propiedad al control de forma que nos permita indicar si admitimos números decimales o no, la propiedad podría llamarse **ConDecimales** y ser del tipo **boolean**, de forma que si el valor que contiene es **true**, admitirá decimales (tal como está ahora) y si vale **false**, no admitirá decimales (ni el separador de miles).

RESOLUCION

El ejercicio propuesto era agregar una nueva propiedad a nuestra clase TextBoxNum, dicha propiedad admitirá sólo valores booleanos, por tanto el tipo de la propiedad será del tipo de datos bool. Como en todas las propiedades, siempre (o casi siempre) necesitaremos una variable que conserve el valor, por tanto necesitaremos una variable privada del mismo tipo que la propiedad. El "casi siempre" es para los casos en los que se usa la clase base, tal como vimos en el código del control.
Veamos el código y las modificaciones que habría que hacer al que ya teníamos escrito. De éste último sólo tendremos que modificar la función que comprueba si el

carácter se admite o no, y ahí comprobaremos si no admitimos decimales, en ese caso, no aceptaremos ni el punto ni la coma.

La declaración de la nueva propiedad **ConDecimales** y la variable privada, cuyo valor inicial será **true** para que en principio (o por defecto) admita decimales.

```
// Propiedad para indicar si se admiten o no decimales
//
// variable privada para contener el valor de la propiedad
bool mConDecimales = true;
//
// la declaramos virtual por si alguien quiere reemplazarla
public virtual bool ConDecimales
{
   get
   {
      return mConDecimales;
   }
   set
   {
      mConDecimales = value;
   }
}
```

La propiedad la declaramos **public** para que sea accesible externamente y **virtual** para dar la opción a quién quiera usar nuestra clase como clase base para crear su propia versión.

Ahora veamos los cambios a realizar al método **CaracterCorrecto** para que tenga en cuenta la nueva propiedad, el resto del código sigue siendo el mismo que teníamos originalmente.

```
// esta función permite controlar si el carácter es de los admitidos
protected virtual bool CaracterCorrecto(char c)
{
   // si ConDecimales = false,
   // no admitir ni el punto ni la coma
   if( ConDecimales == false )
      if( c == '.' || c == ',' )
         return false;
   // devolverá true si el carácter está en el array
   return (Array.IndexOf(digitos, c) != -1);
}
```

Una vez que hayamos hecho estos cambios, podemos generar (o compilar) la librería para poder usarla en el formulario de prueba.

Nota:
Si quieres que se muestre una descripción de la propiedad **ConDecimales** en la ventana de propiedades, tendrás que agregar una importación del espacio de nombres **System.ComponentModel** y aplicar el atributo **Description** a la propiedad:

```
[Description("Indica si se admitirá decimales o no")]
public virtual bool ConDecimales
{
```