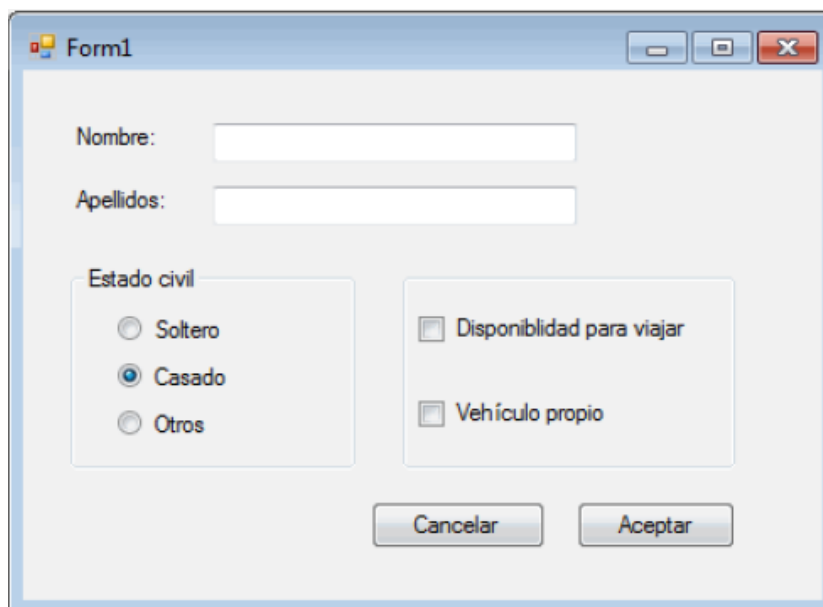


**TEMA 11****Cuadros de diálogo. Crear controles personalizados****1. Orden de tabulación de los controles**

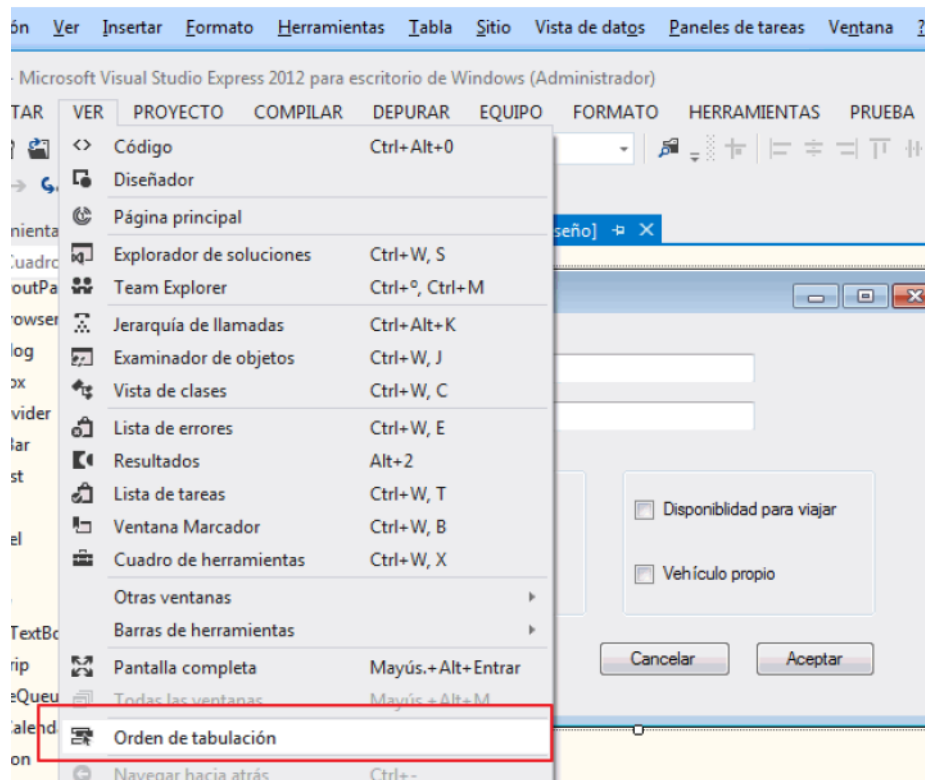
Al comenzar con los controles comentamos que había dos propiedades comunes a todos los controles que veríamos más adelante. Estas propiedades son:

Propiedad	Descripción
TabIndex	Índice del orden de tabulación del control
TabStop	Indica si el usuario puede utilizar la tecla tabulador para poder poner el foco en el control

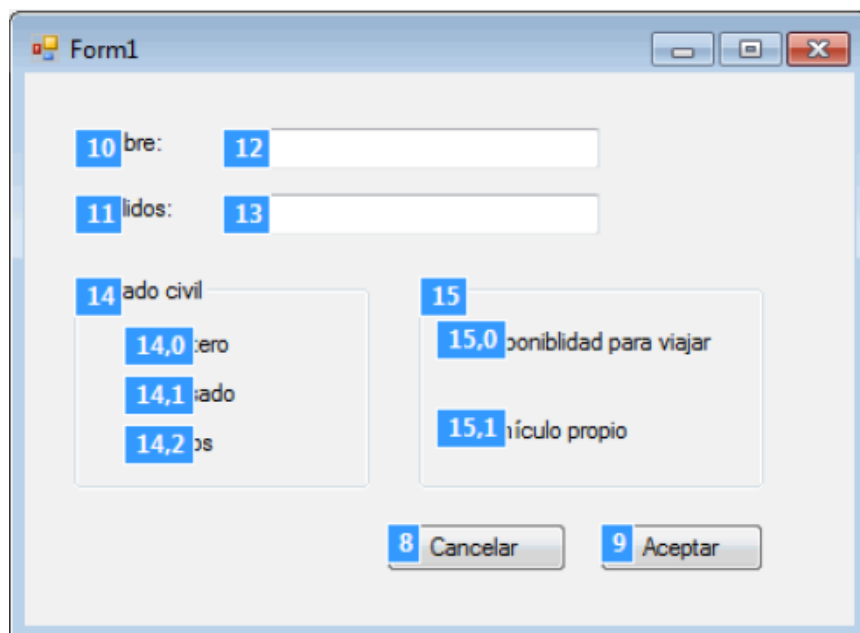
La utilidad es muy sencilla, muchas veces estamos acostumbrados a movernos por los controles de una pantalla con la tecla tabulador en lugar de utilizar el ratón. Vamos a ver cómo podemos modificar la secuencia del movimiento de los controles con la tecla tabulador. Vemos esta pantalla con estos controles:



Tenemos varias etiquetas, cuadros de texto, botones de opción, casillas de verificación y botones de comando. Las he dibujado como he querido, es decir, sin seguir un orden estricto. ¿Cómo actuará la tecla tabulador? Bien en el IDE de .NET tenemos una nueva y cómoda utilidad que nos dice gráficamente el valor que tiene la propiedad "TabIndex". Para esto seleccionamos el formulario y pulsamos la opción "Ver" y luego "Orden de tabulación":



Cambiará de esta forma:



Esta numeración indica que primero enfocará el botón de cancelar (número 8). Deberían ser primero las etiquetas 10 y 11 pero las etiquetas no pueden recibir enfoque porque no disponen de esta propiedad (lógico ya que no se pueden editar).

Así que debemos poner un orden racional a esta propiedad para que sea cómodo moverse por la pantalla. Para esto simplemente iremos haciendo clic en los números en azul para indicarle cual va a ser el orden que queremos que siga. Lo único que tenemos que tener en cuenta es que las etiquetas no se verán afectadas por el orden de tabulación ya que, como hemos comentado, al no poderse editar nunca reciben el enfoque:

Form1

10 bre: 1

11 lidos: 2

3 tado civil

3,0 ltero

3,1 asado

3,2 tros

4

4,0 isponibilidad para viajar

4,1 ehículo propio

5 Cancelar

6 Aceptar

Esta pantalla sería el resultado de ordenar estas tabulaciones. Con esto facilitamos mucho el trabajo a los usuarios.

La otra propiedad que hemos comentado antes era "TabStop" que indica al programa si ese control debe funcionar con los tabuladores como hemos visto hasta ahora. ¿Por que? Muy sencillo, supón que tienen un control "RichTextBox", es decir, un miniprocador de textos donde el usuario puede escribir gran cantidad de texto con formato, si no cambiamos esta propiedad cuando el usuario pulse la tecla del tabulador pasará el enfoque a otro control y eso no es lo que quiero porque como estamos escribiendo con formato debe escribir también estas teclas. Para estos casos entonces (muy pocos) desactivaremos la captura de estas teclas poniendo "TabStop=False"

## Ejercicio 1

Crea un cuadro de diálogo como del de este ejemplo y coloca los tabuladores correctamente para que complete la secuencia.

Form1

Nombre:

Apellidos:

Estado civil

☐ Soltero

☒ Casado

☐ Otros

☐ Disponibilidad para viajar

☐ Vehículo propio

Cancelar

Aceptar

## 2. Cuadros de diálogo

Los cuadros de diálogo son elementos muy utilizados en los programas. Con ellos mostraremos información al usuario o recogeremos información para su proceso posterior. Técnicamente un cuadro de diálogo es un formulario con un estilo de borde fijo, es decir, que entre otras cosas, no se puede redimensionar.

Si abrimos el cuadro de diálogo de opciones en cualquier programa de Windows veremos que es un cuadro que no permite su redimensionamiento, es de tamaño fijo. Estas son sus características más habituales:

- Un cuadro de diálogo no es redimensionable.
- Un cuadro de diálogo puede incluir una barra de título, un cuadro de menú de control y los botones de maximizar-minimizar. Aunque normalmente los veremos sólo con el título y el botón de cerrar a la derecha.

VB.NET incluye varios cuadros de diálogo que podemos utilizar en nuestros programas, uno de ellos es el cuadro de mensaje y el otro es un conjunto de cuadros de diálogo comunes de Windows que veremos ahora.

### 2.1. Mensajes

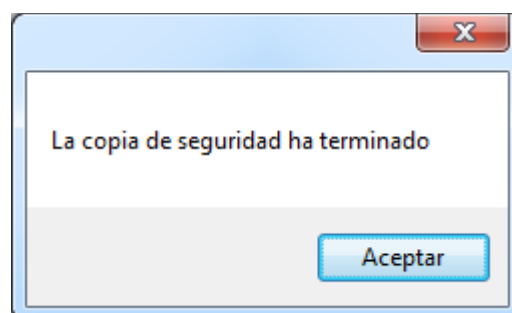
Un cuadro de mensaje muestra un texto al usuario con algún mensaje explicativo, de alerta o de peligro. Por ejemplo, un mensaje indicando que se ha terminado un proceso, una alerta de poco espacio en disco o un mensaje de error de que no hay disco en la unidad.

**Nota:** MsgBox ofrece un resultado parecido pero con menos opciones. Era el utilizado en versiones anteriores de Visual Basic de ahí que permanezca, pero MessageBox está más preparado para .NET y es el que utilizaremos siempre.

Para mostrar mensajes de texto utilizaremos la clase MessageBox y el método "Show", por ejemplo:

```
MessageBox.Show("La copia de seguridad ha terminado")
```

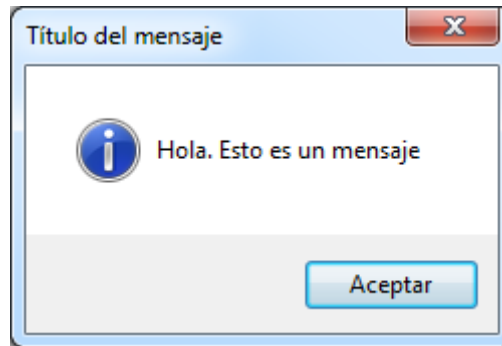
Produce la salida:



Pero tiene muchas opciones para personalizarlo: desde el título, el número de botones o iconos de información, aviso y peligro. Veamos estas opciones.

De momento veamos más opciones para personalizarlo, por ejemplo, queremos poner un título y un icono de información:

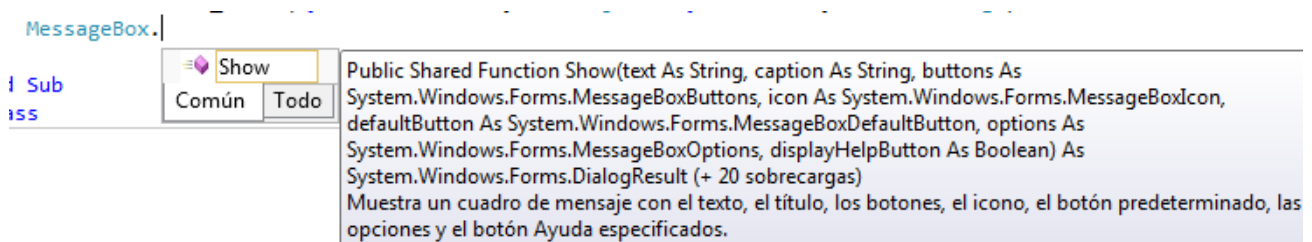
```
MessageBox.Show("Hola. Esto es un mensaje", "Título del mensaje", _  
MessageBoxButtons.OK, MessageBoxIcon.Information)
```



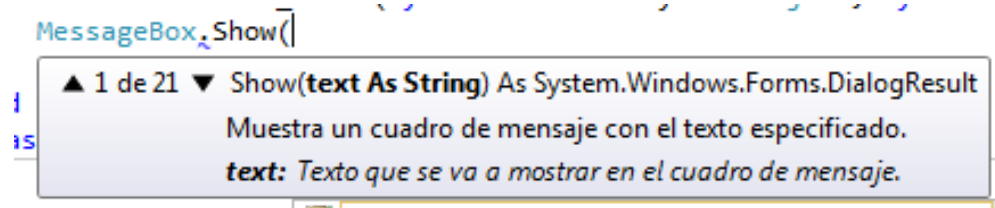
Si has intentado escribir esto en el IDE habrás visto la ayuda que nos proporciona al escribir, y aunque parece exagerada y más molesta que útil, es una buena ayuda para ver la sobrecarga de esta función. Sin esta ayuda tendríamos que estar continuamente consultando un libro o la ayuda en línea para saber qué parámetros debemos poner y qué opciones admite.

Primero hay que decir que este procedimiento MessageBox está sobrecargado, es decir, admite varias formas distintas de utilizarse con un número distinto de parámetros en cada uno de ellos. Pero tranquilo veremos que es sencillo. Abrimos el IDE y detrás del evento clic de un botón en un formulario comenzamos a escribir la instrucción que muestra un mensaje, o mejor dicho, llamar al método Show del objeto MessageBox:

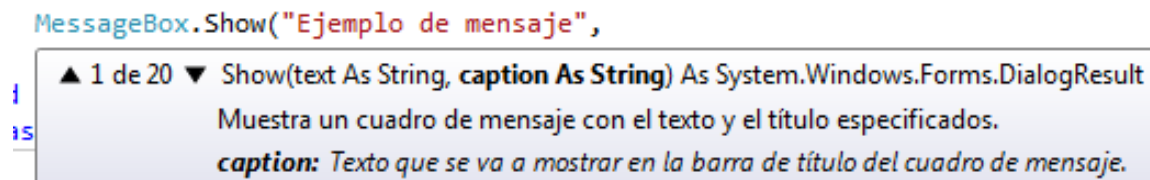
Al escribir la palabra MessageBox y luego el punto "." el IDE, como en otras ocasiones, nos muestra la lista de propiedades y métodos que tiene el objeto. En este caso sólo hay un método "Show" todo lo que aparece a la derecha es la lista de llamadas que se pueden hacer con "MessageBox.Show", como es muy versátil y está sobrecargado nos muestra todas las posibilidades de llamadas que tiene.



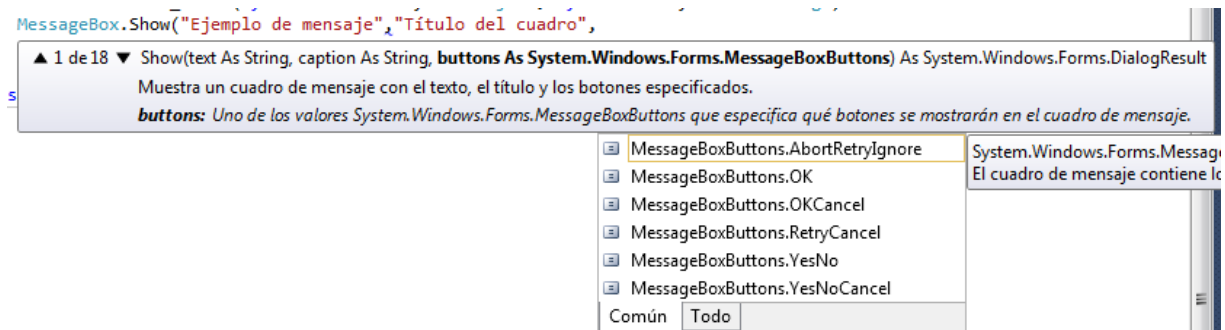
Seguimos, seleccionamos la palabra Show con el espacio o la escribimos. Ahora vamos a introducir los parámetros necesarios para escribir el mensaje, el título, elegir los botones que queremos poner y el icono de información. Escribimos un paréntesis "(":



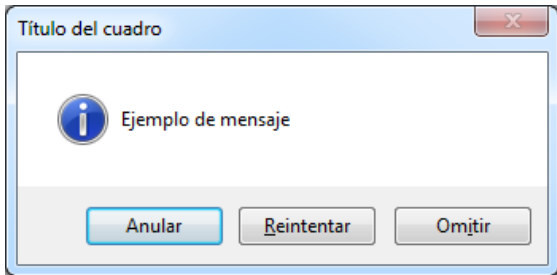
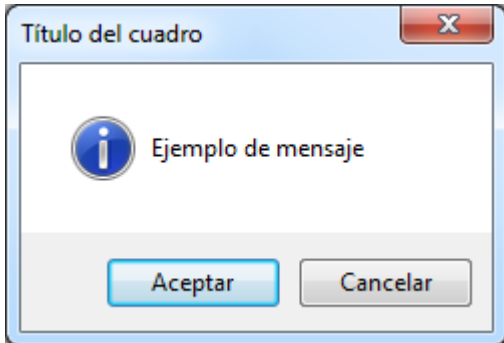
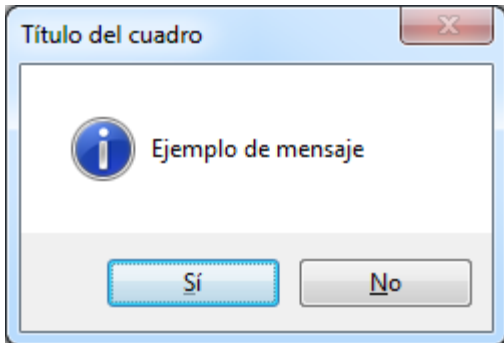
Al abrir el paréntesis el IDE nos indica los parámetros que debemos poner, los indicadores de la izquierda, donde podemos ver dos flechas y "1 de 21" indican los parámetros que debemos introducir según el tipo de MessageBox que queremos escribir. Sigamos con el estándar pero ahora ya podemos leer la ayuda que nos ofrece... en negrita nos pone "text as String" que nos avisa de que el primer parámetro es el contenido del cuadro de texto y su tipo de datos es de tipo Text, así que escribimos un texto de ejemplo entre comillas y escribimos una coma "," para pasar de parámetro:

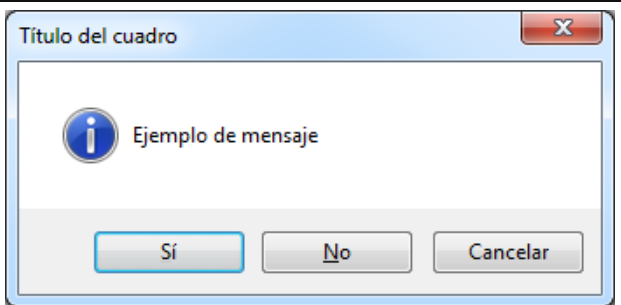


Como ves ahora le toca el turno a la propiedad "caption" que es el título del cuadro de texto que es también de tipo texto, así que lo escribimos entre comillas y escribimos otra coma ",":

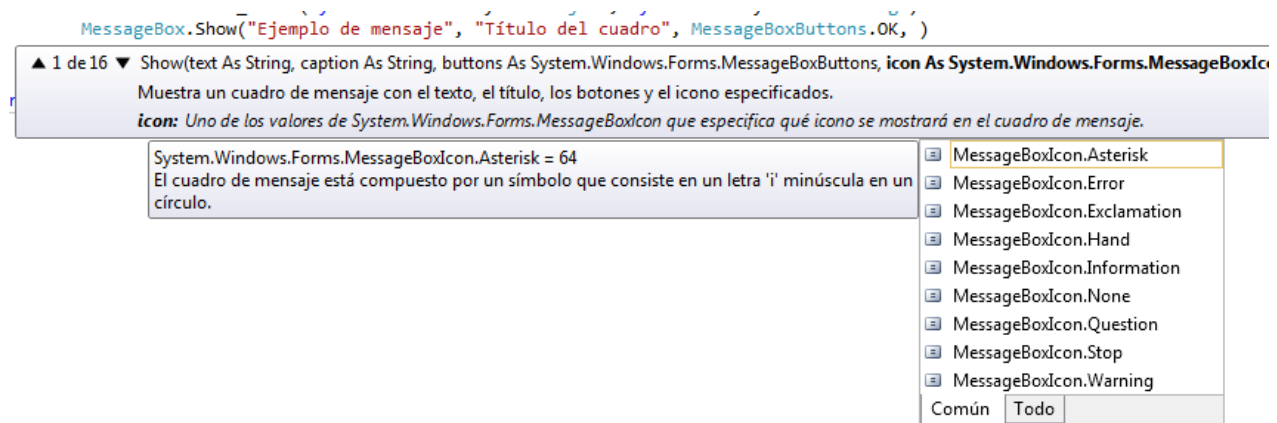


Ahora le toca el turno a los botones que queremos que muestre el cuadro de mensaje. Como hay que escribir un parámetro nos muestra ya en una lista las opciones disponibles, esto es una ayuda enorme. En otras condiciones tendríamos que acudir a la ayuda, buscar los parámetros para el campo tipo de botón, copiarlo y venir al editor y pegarlo. Los botones que podemos seleccionar serán estos:

Valor	Descripción	Cuadro resultante
MessageBoxButtons.AbortRetryIgnore	Botones de Anular, Reintentar y Omitir	
MessageBoxButtons.Ok	Botón de Aceptar	
MessageBoxButtons.OkCancel	Botones de Aceptar y Cancelar	
MessageBoxButtons.RetryCancel	Botones de Reintentar y Cancelar	
MessageBoxButtons.YesNo	Botones de Sí y No	





MessageBoxButtons.YesNoCancel	Botones de Sí, No y Cancelar	
-------------------------------	------------------------------	--

En nuestro ejemplo vamos a seleccionar sólo el de un botón: MessageBoxButtons.Ok. Lo seleccionamos con la tecla TAB o el espacio, pulsamos otra vez la coma "," para el siguiente y último parámetro, el icono:



Este parámetro incluye un icono en nuestro cuadro de diálogo. En esta tabla tienes una imagen del icono que aparecerá. Luego, en nuestro caso, seleccionaremos el de información:

`MessageBoxIcon.Information`

Valor	Icono	Valor	Icono
Asterisk, Information		Exclamation, Warning	
Error, Hand, Stop		Question	

Así que finalmente nuestro mensaje queda con los parámetros:

```
MessageBox.Show("Ejemplo de mensaje", "Título del cuadro", _
    MessageBoxButtons.OK, MessageBoxIcon.Information)
```



Podemos utilizar la instrucción `MessageBox` sólo con un parámetro como vimos en el primer ejemplo, o con varios, dependiendo del nivel de personalización que queramos:

- Sin título ni icono y botón Aceptar

```
MessageBox.Show("Esto es una prueba")
```

- Con Título y botones.

```
MessageBox.Show("Hola", "titulo", MessageBoxButtons.OKCancel)
```

- ...

**Nota:** Los dos parámetros que quedan se refieren al botón que queramos que quede activo en el caso del mensaje con varios botones, podremos dejar marcado el Aceptar o el Cancelar por ejemplo. Y el otro parámetro hace referencia al justificado del texto.

## 2.2. Mensajes como funciones

Los mensajes los hemos visto como procedimientos, es decir, muestran un mensaje y ya está. Pero según el caso que hayamos elegido queremos que nos devuelva un valor. Lógico, si le he dicho al usuario que pulse en "Aceptar" o "Cancelar" tendré que saber qué ha hecho. Por lo tanto se comporta como una función porque nos devuelve ese valor. Veamos un ejemplo, escribe lo siguiente:

```
Dim respuesta As Long
```

```
respuesta = MessageBox.Show("Ejemplo de mensaje", "hola", _  
    MessageBoxButtons.OKCancel, MessageBoxIcon.Information)
```

En este ejemplo, en la variable `respuesta` obtendremos el valor seleccionado por el usuario:

Miembro	Constante	Descripción
<b>OK</b>	<b>vbOK</b>	Se hizo clic en el botón <b>Aceptar</b> .
<b>Cancel</b>	<b>vbCancel</b>	Se hizo clic en el botón <b>Cancelar</b> .
<b>Abort</b>	<b>vbAbort</b>	Se hizo clic en el botón <b>Anular</b> .
<b>Retry</b>	<b>vbRetry</b>	Se hizo clic en el botón <b>Reintentar</b> .
<b>Ignore</b>	<b>vbIgnore</b>	Se hizo clic en el botón <b>Omitir</b> .
<b>Yes</b>	<b>vbYes</b>	Se hizo clic en el botón <b>Sí</b> .
<b>No</b>	<b>vbNo</b>	Se hizo clic en el botón <b>No</b> .

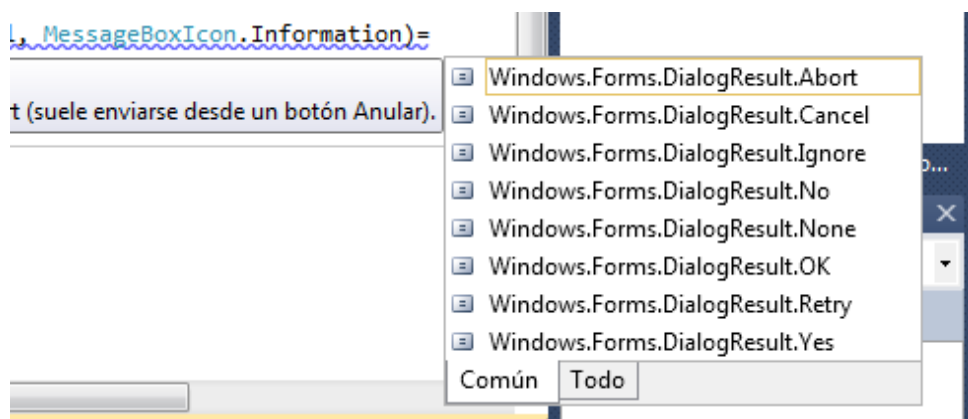
Para tratarlo podemos hacerlo de esta forma...

```
If respuesta = vbOK Then  
    ' ejecuta las instrucciones si ha hecho clic en Aceptar  
Else  
    ' instrucciones si pulsa Cancelar  
End If
```

Finalmente podemos incluir esta comparación en una instrucción If, escríbela tal cual para que veas como se integra en nuestro código. Al escribir el último signo igual

```
if MessageBox.Show("Ejemplo de mensaje", "hola", MessageBoxButtons.OKCancel, _  
MessageBoxIcon.Information)=
```

obtenemos ya la lista de los posibles valores devueltos:

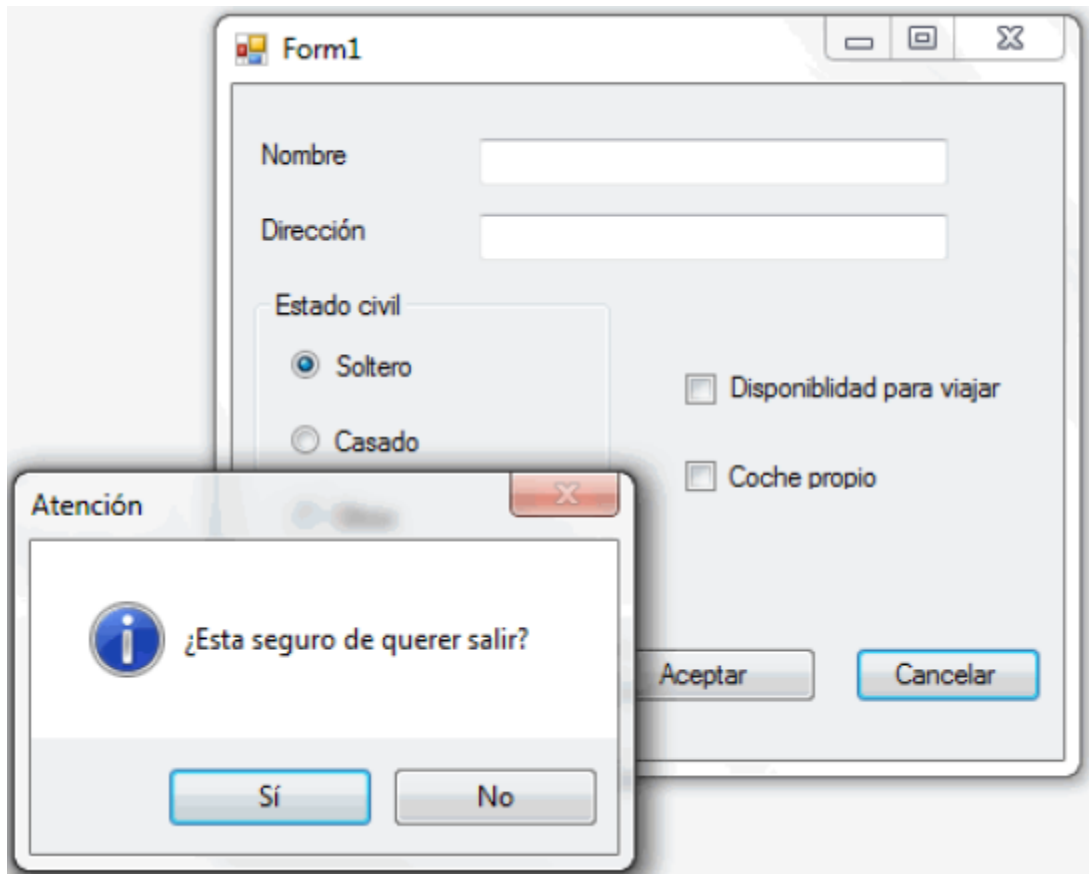


El resultado es que con una sola instrucción hemos hecho toda la comprobación. Lógicamente, si hemos dicho que queremos un cuadro de mensaje con los botones de “Sí/No”, en esta comparación trabajaremos con los valores `DialogResult.Yes` y `DialogResult.No`. Para quedar finalmente:

```
If MessageBox.Show("Ejemplo de mensaje", "hola", MessageBoxButtons.YesNo,  
MessageBoxIcon.Information) = DialogResult.Yes Then  
    'instrucciones si se pulsa Si  
Else  
    'instrucciones si se pulsa No  
End If
```

## Ejercicio 2

Modifica el programa para que cuando se pulse Cancelar un cuadro de diálogo pregunte si queremos salir o no. Debe salir sólo si se pulsa Sí.

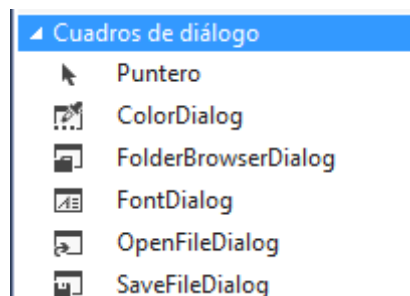


Debe mostrar título e icono de información.

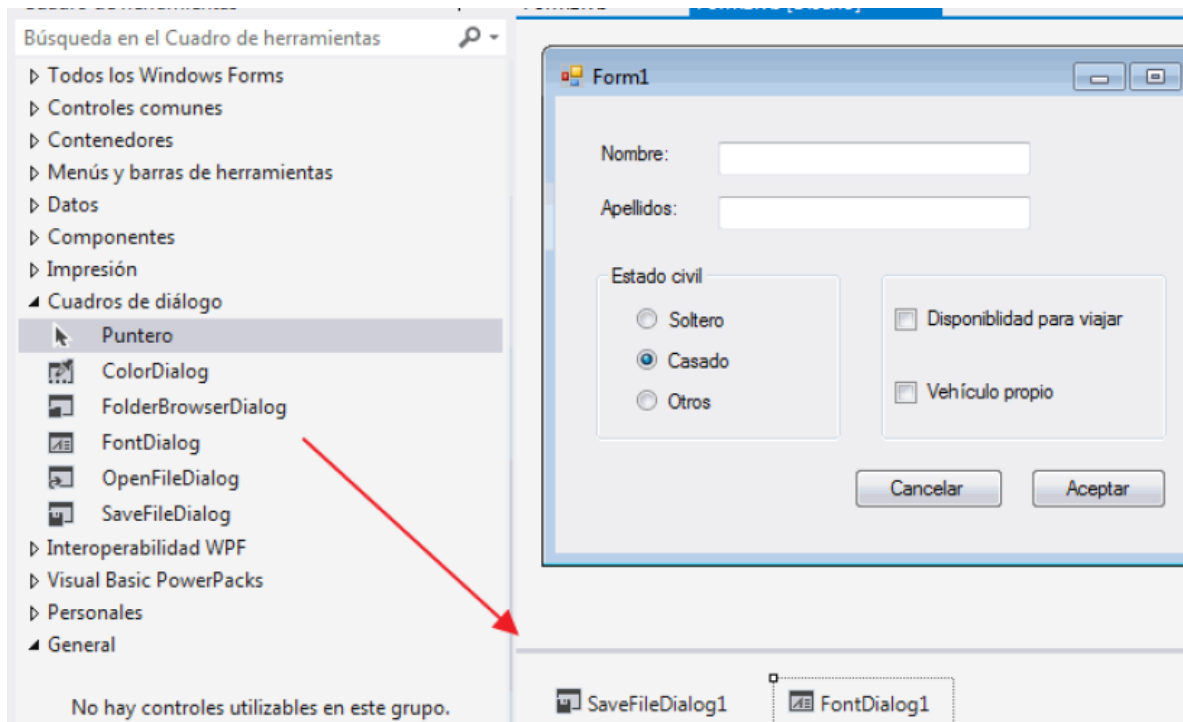
### 3. Cuadros de diálogo comunes

Cuando necesitemos seleccionar un tipo de letra de Windows, colores, un fichero, ... sería muy costoso crear cuadros de diálogo particulares para cada una de estas cosas. Así que lo mejor sería utilizar los propios cuadros de diálogo de Windows, que .NET nos proporciona en los llamados “cuadros de diálogo comunes”. Veamos los cuadros que hay disponibles y cómo los incluiremos en nuestro código.

Para incluirlos en nuestros programas los seleccionaremos de la barra de controles:




Para incluir un cuadro de este tipo en el formulario procederemos como siempre: arrastrándolo de la lista de controles o haciendo doble clic sobre él. Como no es visible en el formulario hasta que no lo invoquemos desde el código aparecerá en la parte inferior de la pantalla indicando que está disponible para utilizarse. En ese punto, es decir, en el diseñador podremos modificar sus propiedades para acabar de personalizarlo y como en el resto de los controles quitar su nombre genérico y ponerle un nombre más cómodo para tratarlo así en el código:



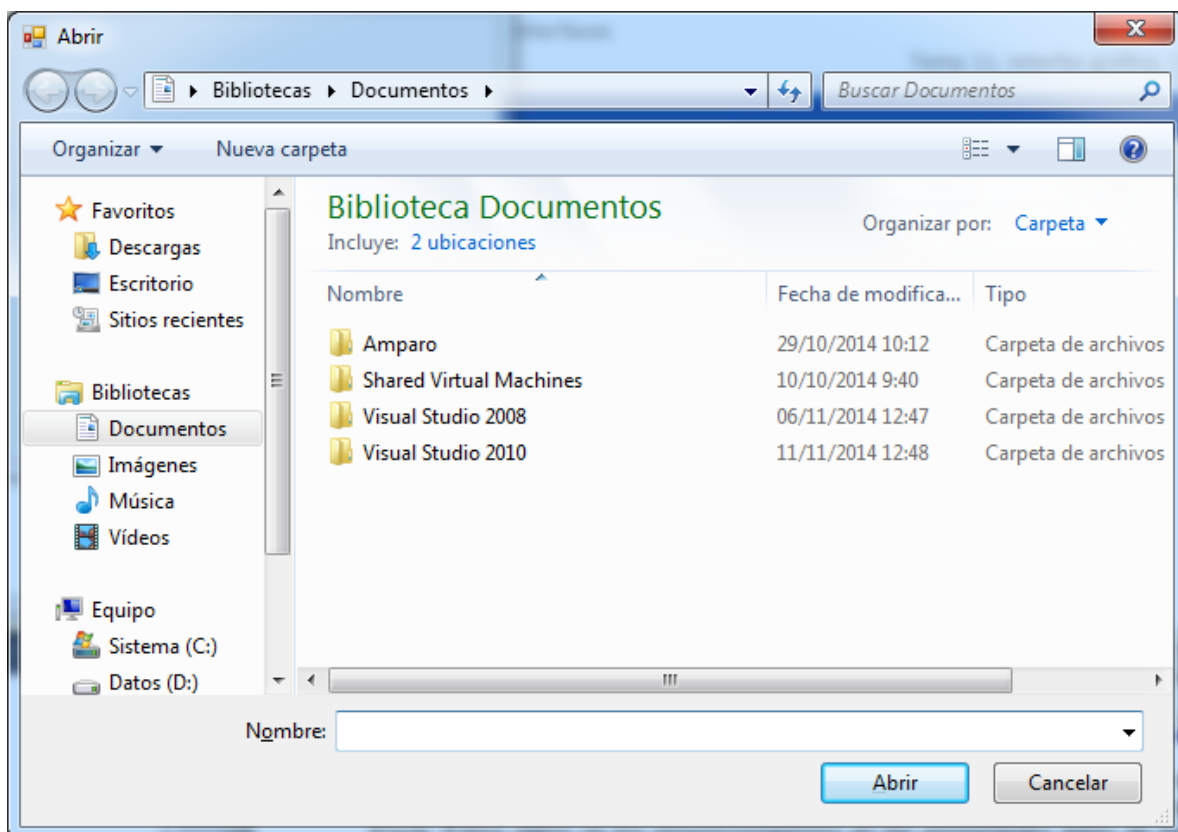
En la parte inferior puedes ver los dos cuadros de diálogo que he insertado. Veamos ahora cuáles tenemos y cómo funciona cada uno de ellos.

### 3.1. Cuadro de diálogo Abrir fichero (OpenFileDialog)

 **OpenFileDialog** Este cuadro es el típico de abrir fichero y que nos va a permitir desplazarnos por las carpetas y unidades para seleccionar un fichero. Es exactamente el mismo que vemos al abrir un archivo o fichero desde cualquier programa de Windows.

De hecho los programas de Windows utilizan este cuadro de diálogo que ofrece el sistema operativo. Es lógico que para las operaciones comunes cada programa no implemente por separado sus propios diálogos comunes, sino que lo mejor es acudir a los que proporciona el sistema operativo.

Por defecto el aspecto que ofrece es:



Muestra por defecto la carpeta “Mis documentos” y sin ningún patrón de ficheros. Un patrón es el desplegable inferior que indica qué tipo de archivos debe mostrar, por ejemplo: \*.xls Libros de Excel. Estos datos se los proporcionaremos en las propiedades antes de llamarlo para que se filtre según ese patrón.

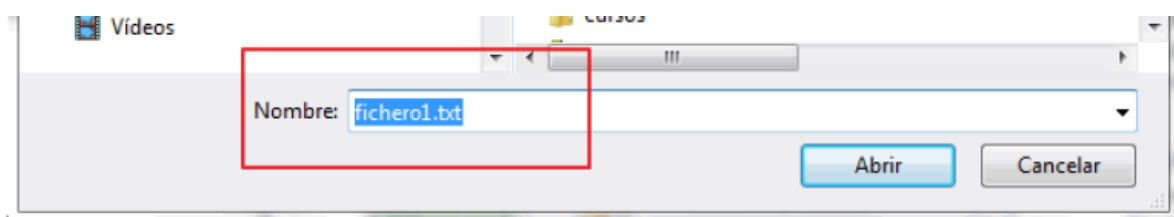
**Nota**, El cuadro en sí no abre ningún fichero, simplemente es una forma de buscar o indicar un fichero al programa, al seleccionarlo nos lo asignará en una variable y podremos utilizarlo en nuestro código...

Primero y como siempre vamos a mostrar las propiedades más importantes:

Propiedad	Descripción
Name	Nombre del control
AddExtension	Obtiene o establece un valor que indica si el cuadro de diálogo agrega automáticamente una extensión a un nombre de archivo en caso de que el usuario omita dicha extensión
CheckFileExists	Obtiene o establece un valor que indica si el cuadro de diálogo muestra una advertencia cuando el usuario especifica un nombre de archivo que no existe.
CheckPathExists	Obtiene o establece un valor que indica si el cuadro de diálogo muestra una advertencia cuando el usuario especifica una ruta de acceso que no existe.
DefaultExt	Obtiene o establece la extensión de nombre de archivo predeterminada
DereferenceLinks	Controla si las teclas de método abreviado deben quedar sin referencia antes de volver del cuadro de diálogo
FileName	Primer archivo que se muestra en el cuadro de diálogo o el último archivo seleccionado por el usuario
Filter	Filtro de archivo para mostrar en el cuadro de diálogo: *.xls
FilterIndex	Índice del filtro de archivo seleccionado en el cuadro de diálogo. El primer elemento tiene de índice el 1
InitialDirectory	Directorio inicial del cuadro de diálogo
MultiSelect	Controla si se pueden seleccionar múltiples archivos en el cuadro de diálogo
ReadOnlyChecked	Estado de la casilla de verificación de sólo lectura en el cuadro de diálogo
RestoreDirectory	Controla si el cuadro de diálogo restaura el directorio actual antes de cerrar
ShowHelp	Habilita el botón ayuda
ShowReadOnly	Controla si muestra la casilla de verificación de sólo lectura en el cuadro de diálogo
Title	Título del cuadro de diálogo
ValidateNames	Controla si el cuadro de diálogo comprueba que los nombres de archivo no contienen caracteres o secuencias no válidas

Una vez que hemos indicado las propiedades que queremos, vamos a utilizar el método "ShowDialog" para mostrarlo. Por ejemplo, queremos mostrar el cuadro de diálogo de abrir fichero y queremos que muestre un nombre de fichero en particular, por ejemplo fichero1.txt, podríamos poner:

```
With cuadro_abrirFichero
    .FileName = "fichero1.txt"
    .ShowDialog()
End With
```



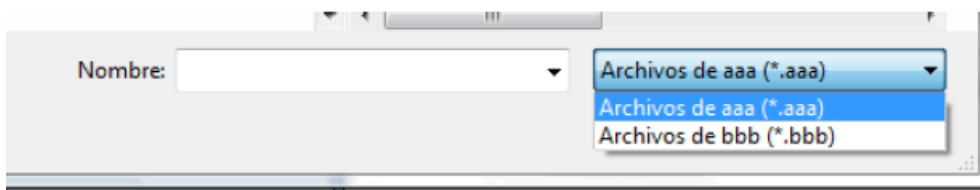
Recuerda que el `With... End With` es una forma abreviada de escribir código cuando estamos haciendo referencia a varias propiedades o elementos de un objeto, lo anterior sería equivalente a:

```
cuadro_abrirFichero.FileName = "fichero1.txt"  
cuadro_abrirFichero.ShowDialog()
```

Veamos más ejemplos. En este caso queremos que muestre los ficheros con extensión `.aaa` y `.bbb` y que los filtre automáticamente. Para esto utilizaremos la propiedad `Filter`:

```
cuadro_abrirFichero.Filter = "Archivos de aaa (*.aaa)|*.aaa|Archivos de bbb (*.bbb)|*.bbb"  
cuadro_abrirFichero.ShowDialog()
```

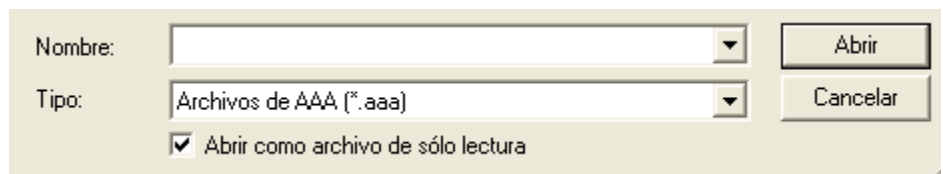
Esto produce:



Luego podemos poner todos los filtros que queramos separados por el carácter `|`. Escribimos el literal que queremos que nos muestre y detrás de ese símbolo el filtro que debe hacer:

```
"Archivos de aaa (*.aaa)|*.aaa"
```

Si queremos poner más como en el caso anterior, seguimos concatenando con el carácter `|`. Las demás propiedades terminarán de personalizar nuestro cuadro. Por ejemplo, las propiedades que activan la presencia de la casilla de verificación de mostrar sólo lectura, producen que en la parte inferior se muestre:



Por un lado `ShowReadOnly` muestra la casilla y `ReadOnlyChecked` indica si se muestra seleccionada sí o no. Como hemos comentado antes, todo esto es informativo, luego nosotros haremos lo necesario con la cadena de texto que nos devuelve que es el nombre del fichero seleccionado por el usuario, mira este ejemplo:


```
Dim nombrefichero As String
```

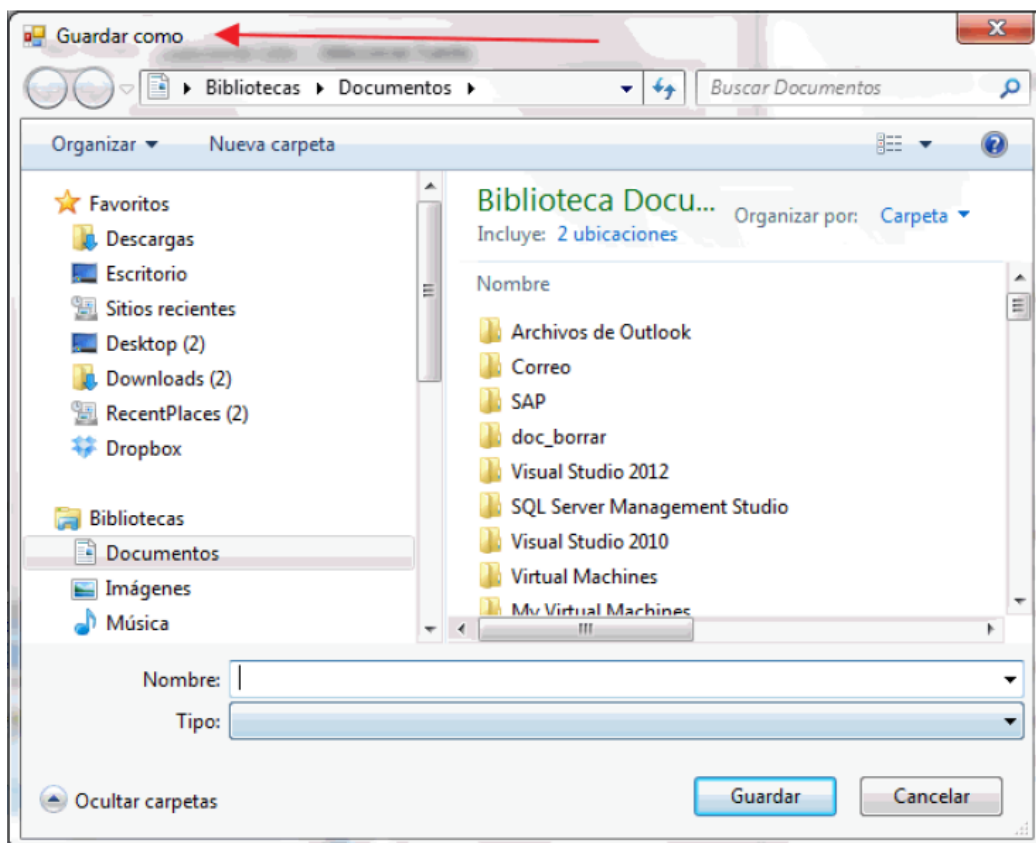
```
cuadro_abrirFichero.Filter = "Todos los archivos (*.*)|*.*"  
cuadro_abrirFichero.ShowDialog()
```

```
nombrefichero = cuadro_abrirFichero.FileName  
MessageBox.Show("Has seleccionado: " & nombrefichero)
```

Cópialo dentro del evento clic de un botón y observa su funcionamiento. Recuerda incluir un control de abrir fichero en el proyecto y ponle de nombre el del ejemplo `"cuadro_abrirfichero"`

### 3.2. Cuadro de diálogo Guardar fichero (SaveFileDialog)

 **SaveFileDialog** Este cuadro es muy similar al anterior. Permite al usuario especificar un nombre y una carpeta para almacenar un fichero. Como en el caso anterior no guarda ningún dato, sino que al pulsar el botón Guardar se cierra el cuadro y nos devuelve los valores de la carpeta y fichero elegidos por el usuario en dos propiedades.



Las propiedades son las mismas que en el caso anterior y además estas:


Propiedad	Descripción
CreatePrompt	Controla si debe avisar al usuario cuando se va a crear un nuevo archivo. Sólo es aplicable si "ValidateNames" está establecida a True.
OverwritePrompt	Controla si debe avisar al usuario cuando se va a sobrescribir un archivo ya existente. Sólo es aplicable si "ValidateNames" está establecida a True
ValidateNames	Controla si el cuadro de diálogo comprueba que los nombres de archivo no contienen caracteres o secuencias no válidas.

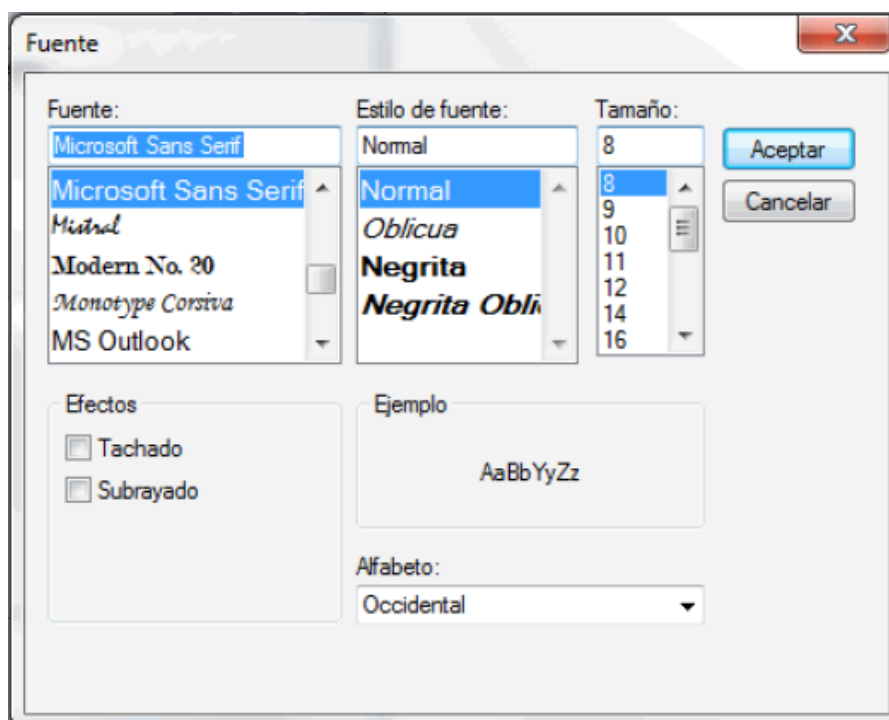
Por ejemplo, si queremos mostrar el cuadro de diálogo de Guardar pero que nos filtre para los archivos con extensión \*.txt y opcionalmente todos los archivos:

```
With cuadro_guardarfichero
    .Filter = "Archivos de Texto (*.txt) | *.txt | Todos los archivos (*.*) | *.*"
    .ShowDialog()
End With
```



### 3.3. Cuadro de diálogo tipo de letra (FontDialog)

 **FontDialog** Nos encontramos ante otro cuadro de diálogo preconfigurado y muy útil cuando queramos seleccionar o modificar algún tipo de letra. Este cuadro muestra los tipos de letra instalados en nuestro equipo y permite al usuario seleccionar un tipo de letra, estilo y tamaño. Además se pueden activar algunos efectos como "subrayado" o "tachado":



Como siempre veamos las propiedades más interesantes de este cuadro de diálogo:

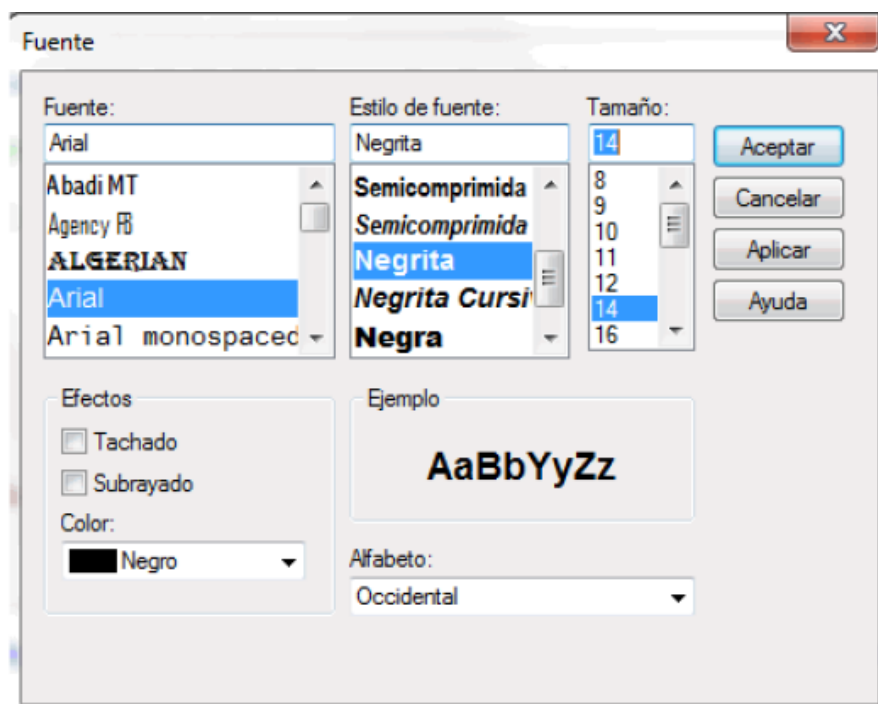
Propiedad	Descripción
AllowScriptChange	Obtiene o establece un valor que indica si el usuario puede cambiar el juego de caracteres especificado en el cuadro combinado <b>Script</b> para mostrar un juego de caracteres diferente del que aparece. El cuadro combinado <b>Script</b> que se encuentra en el cuadro de diálogo de fuentes contiene los juegos de caracteres asociados a la fuente seleccionada.
AllowSimulations	Obtiene o establece un valor que indica si el cuadro de diálogo permite las simulaciones de fuente de la interfaz de dispositivo gráfico (GDI).
AllowVectorFonts	Obtiene o establece un valor que indica si se permite la selección de fuentes vectoriales en el cuadro de diálogo.
AllowVerticalFonts	Obtiene o establece un valor que indica si el cuadro de diálogo muestra tanto fuentes horizontales como verticales o sólo fuentes horizontales.
Color	Obtiene o establece el color de la fuente seleccionada.
FixedPitchOnly	Obtiene o establece un valor que indica si el cuadro de diálogo permite sólo la selección de fuentes de punto fijo.
Font	Obtiene o establece la fuente seleccionada
FontMustExit	Obtiene o establece un valor que indica si el cuadro de diálogo debe especificar una condición de error cuando el usuario intente seleccionar una fuente o un estilo que no exista.

MaxSize	Obtiene o establece el tamaño de punto máximo que un usuario puede seleccionar
MinSize	Obtiene o establece el tamaño de punto mínimo que un usuario puede seleccionar.
ScriptsOnly	Obtiene o establece un valor que indica si el cuadro de diálogo permite la selección de fuentes para todos los juegos de caracteres no OEM y Symbol, así como para el juego de caracteres ANSI.
ShowApply	Obtiene o establece un valor que indica si el cuadro de diálogo contiene un botón <b>Aplicar</b>
ShowColor	Obtiene o establece un valor que indica si el cuadro de diálogo muestra la opción de color.
ShowEffects	Obtiene o establece un valor que indica si el cuadro de diálogo contiene controles que permiten al usuario especificar opciones de tachado, subrayado y color del texto.
ShowHelp	Obtiene o establece un valor que indica si el cuadro de diálogo presenta un botón Ayuda.

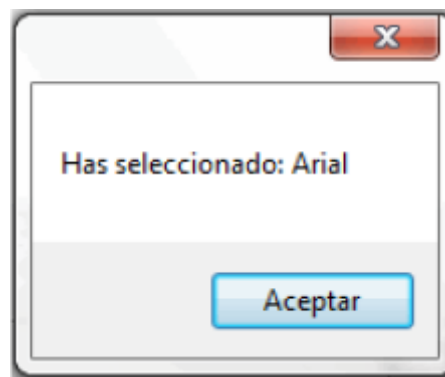
Veamos un ejemplo, queremos seleccionar una fuente y queremos que el cuadro muestre el botón de Aplicar, la selección de color, efectos y ayuda, luego mostraremos la fuente seleccionando:

```
With cuadro_tipoLetra
    .ShowApply = True
    .ShowEffects = True
    .ShowColor = True
    .ShowHelp = True
    .ShowDialog()
    MessageBox.Show("Has seleccionado: " & .Font.Name)
End With
```

Da como resultado el siguiente cuadro:



Y devuelve al pulsar *Aceptar*:



Claro que aquí hay truco... ¿por qué he puesto `Font.Name` al escribir el resultado? Bueno pues porque `Font` es un objeto que tiene sus propiedades como nombre, color, tamaño,... Así cuando el usuario indique un tipo de letra lo que nos va a devolver el cuadro no es un nombre como en el caso de los cuadros de abrir y guardar fichero sino que nos va a devolver un objeto `Font` con todos los detalles que le hemos indicado: tipo de letra, color, efectos. Veamos un poco el objeto `Font` para saber qué podemos devolver.


## Clase Font

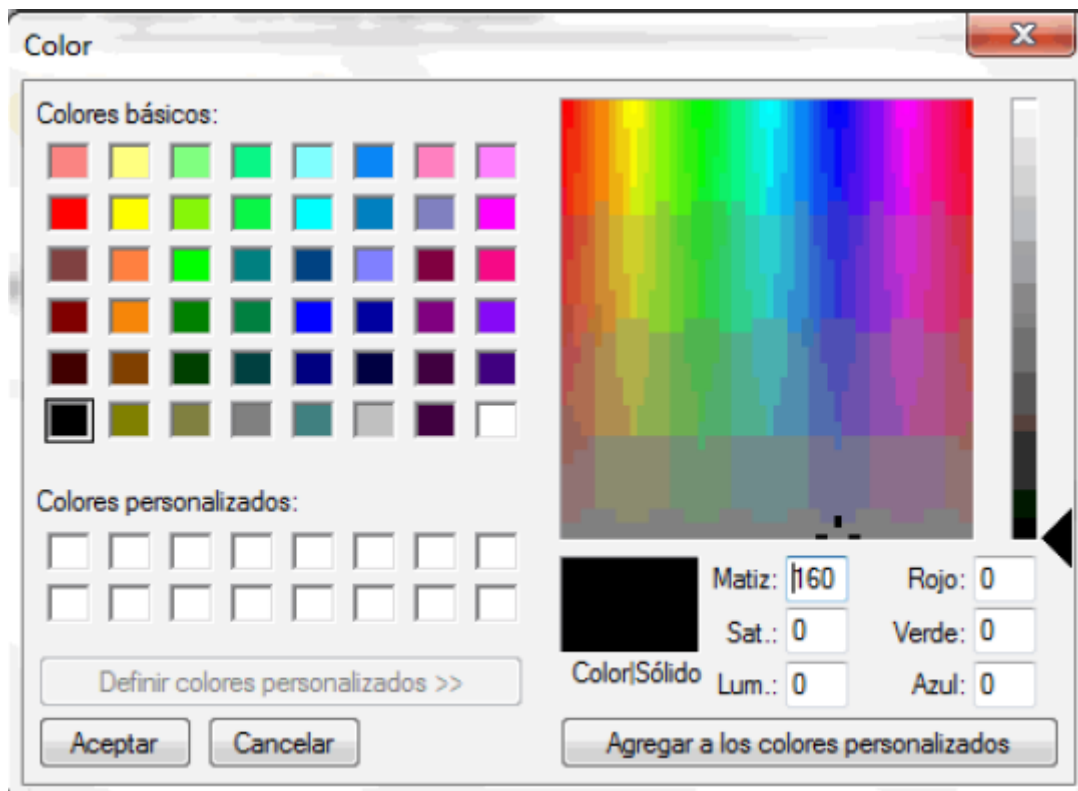
Define un formato concreto para el texto, incluidos el nombre de fuente, el tamaño y los atributos de estilo. Esta clase no se puede heredar.

Las propiedades más importantes son:

Propiedad	Descripción
<b>Bold</b>	Obtiene un valor que indica si este objeto <b>Font</b> está en negrita
<b>Height</b>	Obtiene el interlineado de esta fuente.
<b>Italic</b>	Obtiene un valor que indica si este objeto <b>Font</b> está en cursiva.
<b>Name</b>	Obtiene el nombre del tipo de letra.
<b>Size</b>	Obtiene el tamaño de la fuente expresado en la unidad de este objeto <b>Font</b> .
<b>StrikeOut</b>	Obtiene un valor que indica si este objeto <b>Font</b> especifica una línea horizontal de tachado de la fuente.
<b>Font</b>	Obtiene o establece la fuente seleccionada
<b>UnderLine</b>	Obtiene un valor que indica si este objeto <b>Font</b> está subrayado.

### 3.4. Cuadro de diálogo de selección de color (ColorDialog)

 **ColorDialog** Este cuadro de diálogo permite al usuario seleccionar un color de la paleta o personalizar uno para incluirlo en la paleta de colores actual:




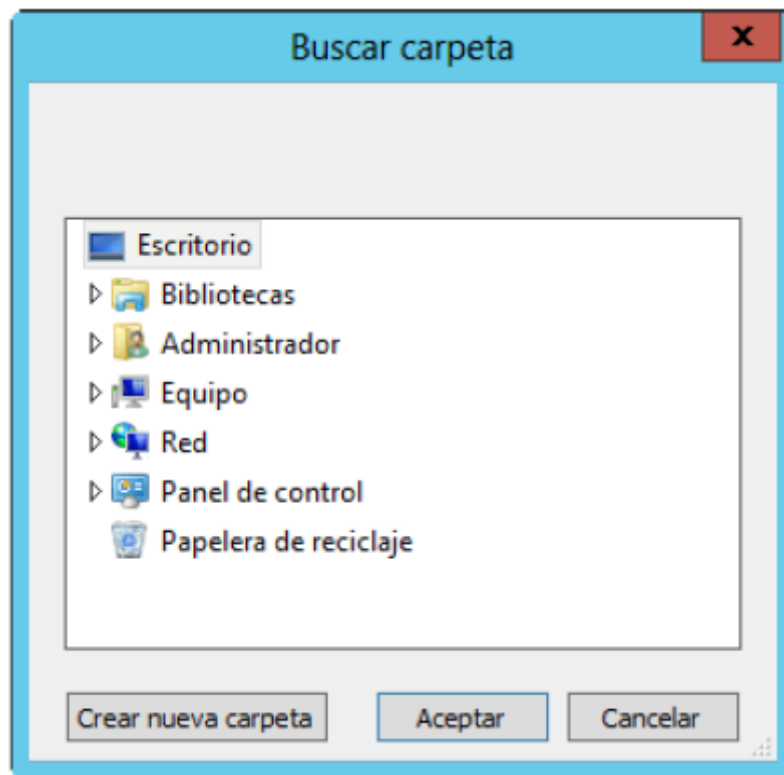
La propiedad Color es la que almacena o devuelve el color actual o seleccionado. Las propiedades en este caso son:

Propiedad	Descripción
AllowFullOpen	Obtiene o establece un valor que indica si el usuario puede utilizar el cuadro de diálogo para definir colores personalizados.
AnyColor	Obtiene o establece un valor que indica si el cuadro de diálogo muestra todos los colores disponibles en el conjunto de colores básicos.
Color	Obtiene o establece el color seleccionado por el usuario.
FullOpen	Obtiene o establece un valor que indica si los controles utilizados para crear colores personalizados son visibles al abrir el cuadro de diálogo
ShowHelp	Obtiene o establece un valor que indica si aparecerá el botón Ayuda en el cuadro de diálogo de colores
SolidColorOnly	Obtiene o establece un valor que indica si el cuadro de diálogo muestra únicamente colores sólidos para elegir.

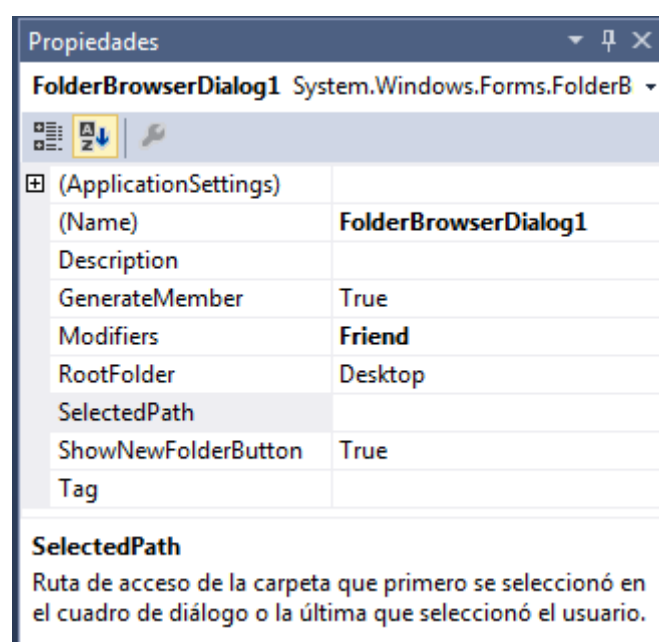
Si no se selecciona ningún color, el valor predeterminado es el negro.

### 3.5. Cuadro de diálogo de selección de directorio (FolderBrowserDialog)

 **FolderBrowserDialog** Este cuadro es más sencillo que los anteriores de archivos, ya que simplemente nos muestra la exploración de carpetas para seleccionar una de ellas. Veámoslo en funcionamiento:



Las propiedades las podemos ver directamente al seleccionar el control y explorar el cuadro de propiedades:



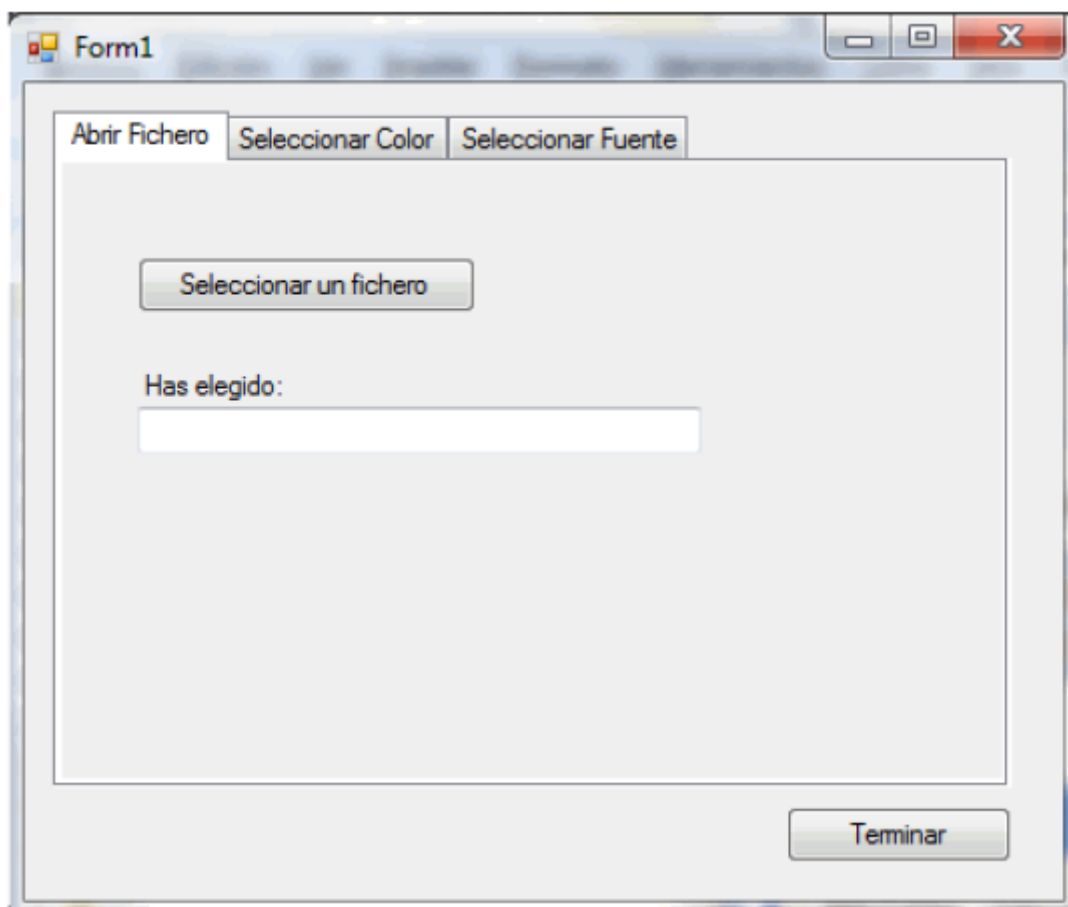
Como propiedad exclusiva, es la posibilidad de que nos muestre el botón para crear nueva carpeta. La otra propiedad interesante es la de indicar la carpeta raíz "RootFolder", para comenzar la exploración en ese punto.

"SelectPath" devuelve la carpeta seleccionada por el usuario.

## Ejemplo 1

Vamos a crear tres solapas de forma que en cada una de ellas haya un ejemplo de los cuadros de diálogo vistos.

- En la primera solapa podemos seleccionar un fichero:



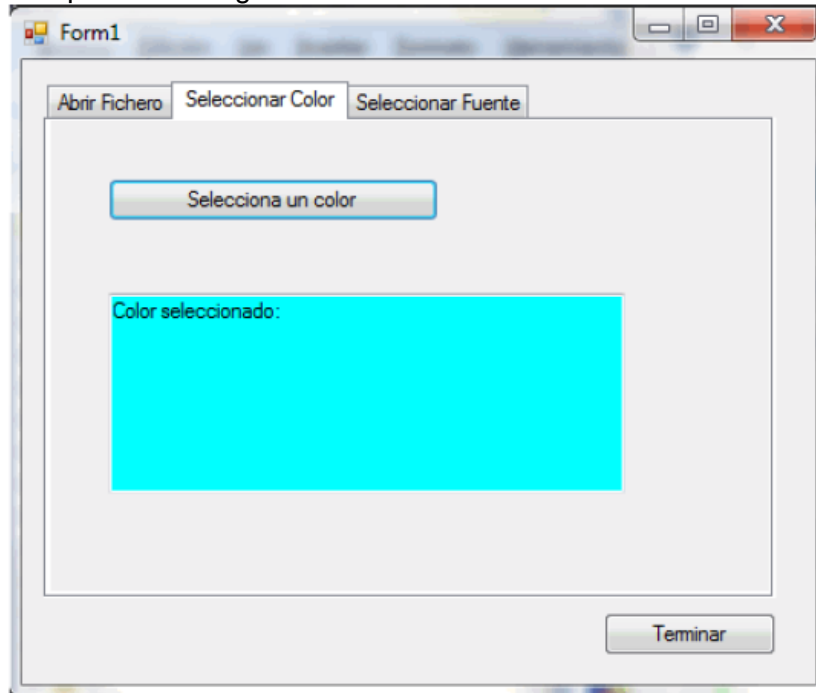
Para abrir un fichero pondremos un código como este, por ejemplo:

```
Private Sub btn_fichero_Click(sender As Object, e As EventArgs) Handles btn_fichei

    'Filtro para los archivos
    Me.dlg_abrir.Filter = "Todos lo archivos (*.*)|*.*"
    'Mostramos el cuadro
    Me.dlg_abrir.ShowDialog()

    'El resultado está en FileName
    Me.txt_fichero.Text = dlg_abrir.FileName
End Sub
```

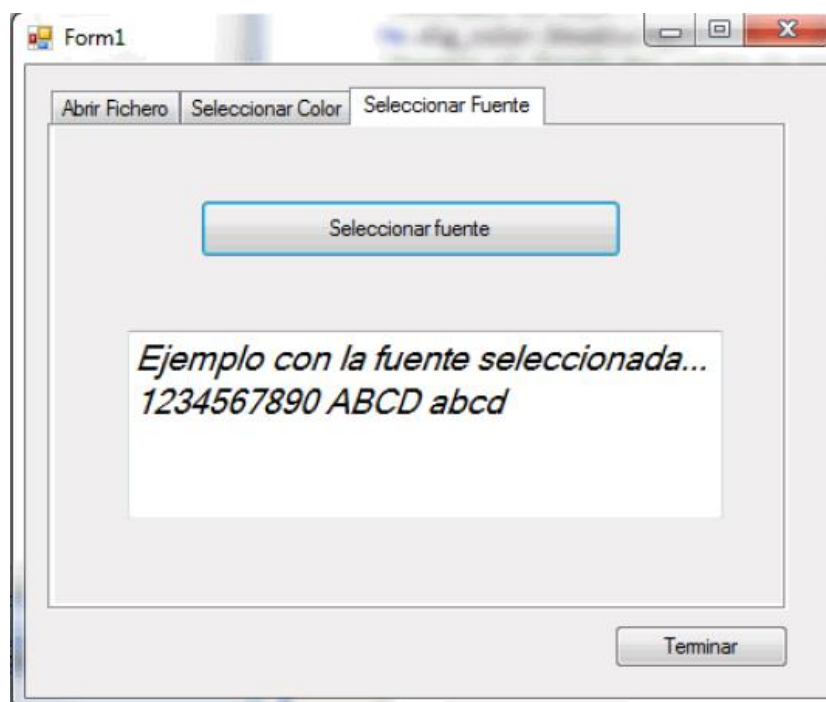
- En la segunda podemos elegir un color:



El código es muy similar al anterior, sólo cambia la recogida de datos, que en este caso es el color "Me.dlg\_color.Color":

```
Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    'Mostramos el cuadro
    Me.dlg_color.ShowDialog()
    'Ponemos el fondo del cuadro de texto con este color
    Me.txt_ejemplo.BackColor = Me.dlg_color.Color
End Sub
```

- O seleccionar una fuente:

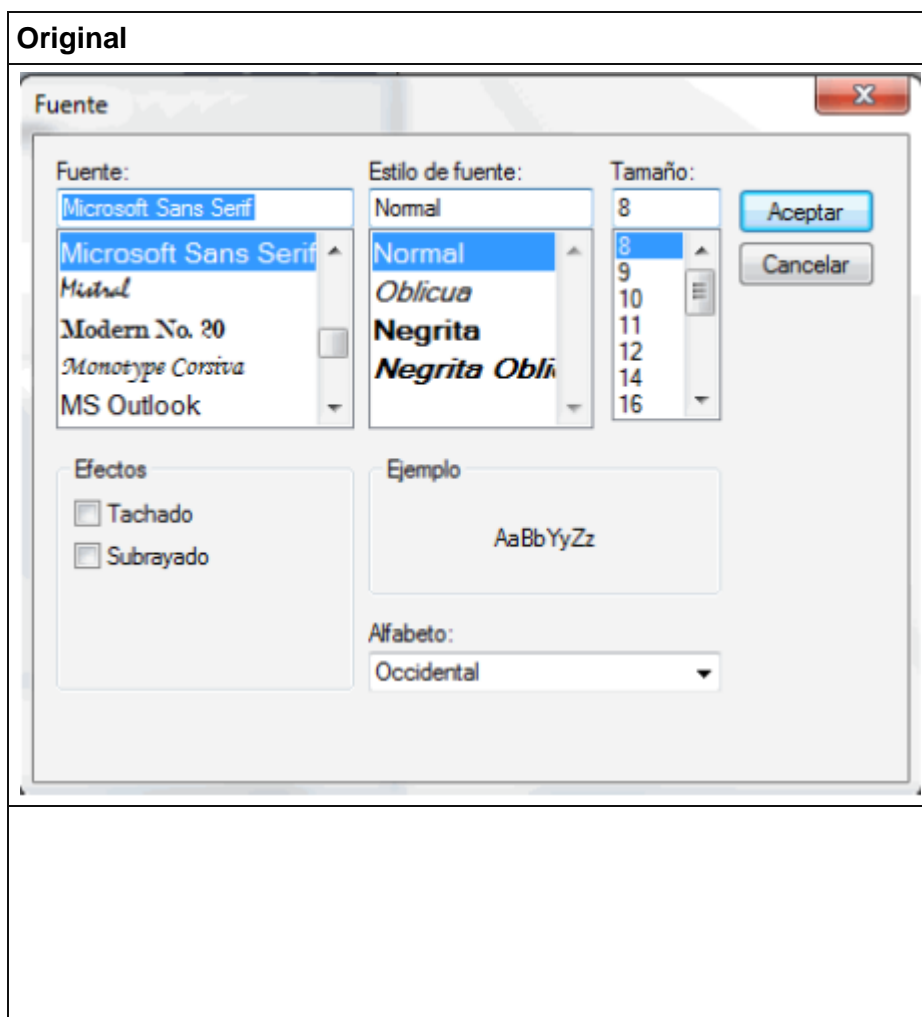


Observa el código para mostrar este cuadro de diálogo:

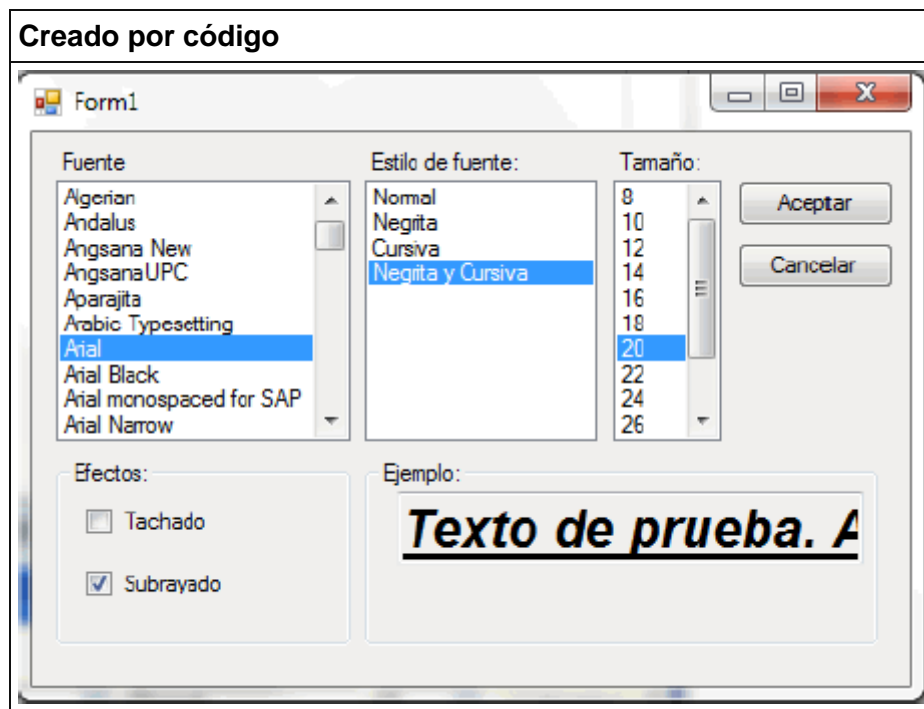
```
Private Sub btn_fuente_Click(sender As Object, e As EventArgs) Handles btn_fuente.Click
    'Muestro el cuadro de diálogo y asigno el resultado...
    Me.dlg_fuente.ShowDialog()
    Me.Txt_fuente.Font = Me.dlg_fuente.Font
End Sub
```

## Ejemplo 2

Vamos a realizar otro interesante ejemplo y es simular el cuadro de diálogo de selección de fuentes enteramente por código. Compara el original de Windows con el que vamos a crear a mano:

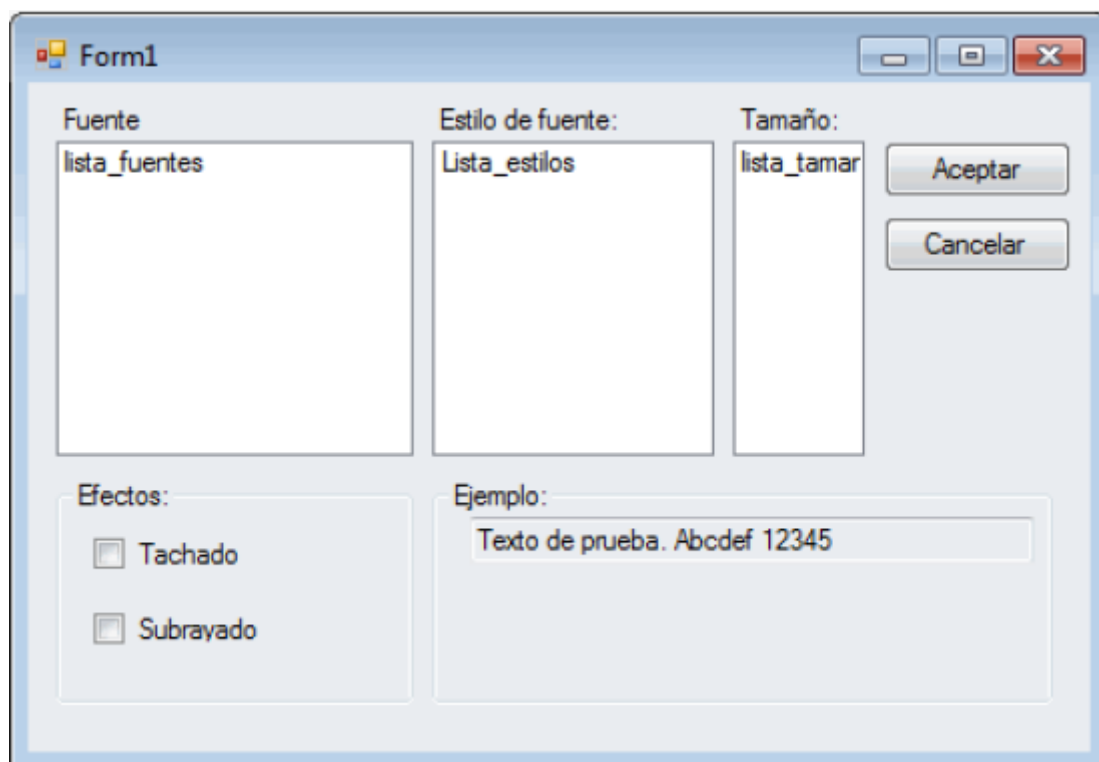






Con alguna pequeña diferencia pero básica ofrecemos lo mismo, veamos el código.

Primero dibujamos los elementos que van a componer la pantalla: tres cuadros de lista para los tamaños, dos botones, un cuadro de texto para el ejemplo y dos casillas de verificación para las opciones de tachado y subrayado:



Vamos a dividir en tres partes el código:

- Comienzo de la aplicación: llenamos los cuadros de lista con los datos de las fuentes, estilos y tamaños

- Exploramos los eventos de todos los elementos para actualizar el tipo de letra del ejemplo
- Procedimiento para escribir el texto resultante

## Comienzo del programa

Debemos iniciar todos los elementos así que:

```
Private Sub frm_principal_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim i As Byte
    Dim tipo As FontFamily
    'Iniciamos los cuadros de lista

    'Cuadro de lista de tipos de letra:
    For Each tipo In FontFamily.Families
        'Añadimos el nombre de la fuente
        Me.lista_fuentes.Items.Add(tipo.Name)
    Next

    'Cuadro de lista de los estilos:
    Me.lista_estilos.Items.Add("Normal")
    Me.lista_estilos.Items.Add("Negrita")
    Me.lista_estilos.Items.Add("Cursiva")
    Me.lista_estilos.Items.Add("Negrita y Cursiva")

    'Cuadro de lista de los tamaños:
    For i = 8 To 32 Step 2
        Me.lista_tamanos.Items.Add(i)
    Next

    'Activamos unas opciones predeterminadas:
    Me.lista_fuentes.SelectedIndex = 0
    Me.lista_estilos.SelectedIndex = 0
    Me.lista_tamanos.SelectedIndex = 0

    'Y llamamos a un procedimiento para que escriba el texto
    'de ejemplo con los datos actuales
    escribe_ejemplo()
End Sub
```

En la primera parte, recorreremos la colección de fuentes instaladas en nuestro equipo con FontFamily, luego rellenamos con varios estilos el cuadro de lista de los estilos y finalmente llenamos con los valores 8 a 32 el cuadro para el tamaño de la fuente. Por último, marcamos como elementos seleccionados los primeros de cada lista y llamamos al procedimiento que escriba el ejemplo con el tipo seleccionado.

## Detectar los eventos

Cada vez que se modifique un elemento del cuadro llamaremos al procedimiento para que escriba el ejemplo con los valores seleccionados. Así que utilizaré los eventos de los cuadro de lista y de las casillas de verificación:

```
Private Sub lista_fuentes_SelectedIndexChanged(sender As Object, e As EventArgs)
    escribe_ejemplo()
End Sub

Private Sub Lista_estilos_SelectedIndexChanged(sender As Object, e As EventArgs)
    escribe_ejemplo()
End Sub

Private Sub lista_tamanos_SelectedIndexChanged(sender As Object, e As EventArgs)
    escribe_ejemplo()
End Sub

Private Sub chk_subrayado_CheckedChanged(sender As Object, e As EventArgs) Handle
    escribe_ejemplo()
End Sub

Private Sub chk_tachado_CheckedChanged(sender As Object, e As EventArgs) Handle:
    escribe_ejemplo()
End Sub
```

## Escribir ejemplo

Finalmente con los valores elegidos por el usuario escribimos el texto:

```
Sub escribe_ejemplo()
    'Escribo el texto si hay algún tamaño seleccionado, sino daría error
    If Me.lista_tamanos.SelectedIndex <> -1 Then
        'Defino ya la fuente y tamaños del cuadro de lista
        Dim fuente As String = Me.lista_fuentes.SelectedItem.ToString
        'Convierto de string a integer la lectura del cuadro de lista
        Dim tam As Integer = CType(Me.lista_tamanos.SelectedItem.ToString, Integer)
        Dim estilo As FontStyle

        'Miro la lista de los estilo y voy añadiendo los valores
        Select Case Me.Lista_estilos.SelectedIndex
            Case 0
                estilo = FontStyle.Regular
            Case 1
                estilo = FontStyle.Italic
            Case 2
                estilo = FontStyle.Bold
            Case 3
                estilo = FontStyle.Bold Or FontStyle.Italic
        End Select

        'Miro se ha seleccionado subrayado
        If Me.chk_subrayado.Checked = True Then
            estilo = estilo Or FontStyle.Underline
        End If

        'Miro se ha seleccionado tachado
        If Me.chk_tachado.Checked = True Then
            estilo = estilo Or FontStyle.Strikeout
        End If

        'Defino la fuente resultante...
        Dim mifuentes As New Font(fuente, tam, estilo)

        'Escribo el ejemplo
        Me.txt_ejemplo.Font = mifuentes
    End If
End Sub
```

La clase Font es de sólo lectura así que no podemos ir cambiando las propiedades de Font del cuadro de texto. Para solucionar esto simplemente creamos un fuente con los valores de: tipo, tamaño y estilo: Dim mifuente as New Font (fuente, tam, estilo) y se la asigno al cuadro de texto.

Para conseguir estos tres valores simplemente leo los valores de la interfaz. Para el caso del tipo de letra:


```
Dim fuente As String = Me.lista_fuentes.SelectedItem.ToString
```

Que es el tipo de letra seleccionado en el cuadro de lista. Para el tamaño:

```
Dim tam As Integer = CType(Me.lista_tamanos.SelectedItem.ToString, Integer)
```

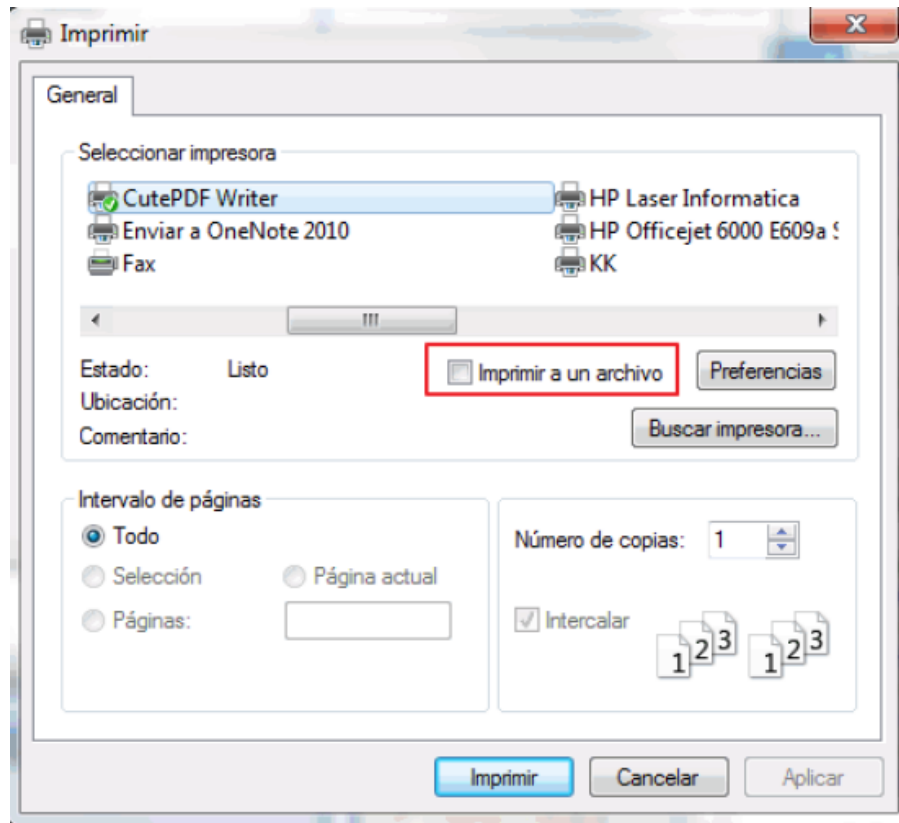
Teniendo la precaución de convertir a entero el valor leído en el cuadro de lista. Finalmente los estilos es un grupo de constantes ya definidas de tipo FontStyle. Estos estilos hay que sumarlos, es decir, concatenarlos con el operador Or para que vaya almacenando los estilos. Así que simplemente veo con el Select Case qué estilo ha seleccionado y si ha marcado alguna casilla de verificación.

### 3.6. Cuadro de diálogo de Impresión (PrintDialog)


 **PrintDialog** Este cuadro permite al usuario seleccionar una impresora, elegir el número de páginas a imprimir y otros parámetros de impresión. También permite seleccionar qué imprimir: todo, una página, un rango o una selección. Las propiedades son:

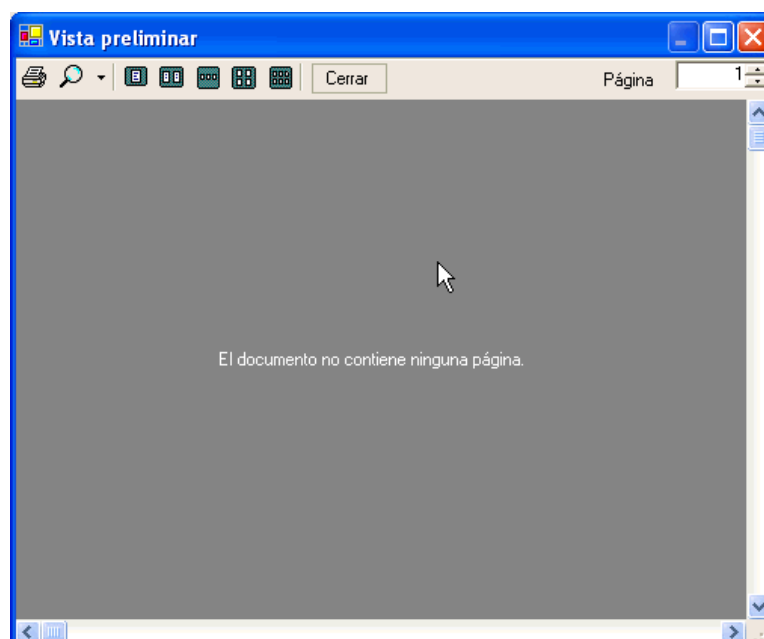
Propiedad	Descripción
AllowPrintToFile	Habilita y deshabilita la casilla <i>Imprimir a un archivo</i> .
AllowSelection	Activa la opción de permitir selección
AllowSomePages	Obtiene o establece un valor que indica si se habilita el botón de opción Páginas.
Document	Obtiene o establece un valor que indica el PrintDocument del que se obtendrá PrinterSettings.
PrintToFile	Obtiene o establece un valor que indica si la casilla de verificación <i>Imprimir a un archivo</i> está activada
ShowHelp	Obtiene o establece un valor que indica si se muestra el botón Ayuda.
ShowNetwork	Obtiene o establece un valor que indica si se muestra el botón Red.

Por ejemplo, podemos utilizar la propiedad "AllowPrintToFile" que habilita la casilla de verificación *Imprimir a archivo*.



### 3.7. Control de vista previa (PrintPreviewDialog)

 **PrintPreviewDialog** Este cuadro de diálogo muestra la vista previa de un documento enviado a la impresora:



Este control es el único que contiene otro interno: **PrintPreviewControl**. Expone una serie de propiedades como filas y columnas que determinan el número de páginas que mostrará en horizontal y vertical. Por lo tanto, como es un objeto dentro de otro para acceder a esta propiedad tendríamos que escribir:

```
PrintPreviewDialog1.PrintPreviewControl.Columns
```

Puesto que PrintPreviewControl está contenido automáticamente dentro del control PrintPreviewDialog, cuando lo añadimos al formulario no hace falta incluir PrintPreviewControl. Veamos las propiedades de los dos controles:


Propiedades de PrintPreviewControl:

Propiedad	Descripción
AutoZoom	Obtiene o establece un valor que indica si el cambio de tamaño del control o el cambio de número de páginas que se muestra automáticamente se ajusta a la propiedad Zoom.
BackColor	Obtiene o establece el color de fondo del control.
BackgroundImage	Obtiene o establece la imagen de fondo que se muestra en el control.
Columns	Obtiene o establece el número de páginas que aparecen en la pantalla con orientación horizontal.
ContextMenu	Obtiene o establece el menú contextual asociado al control.
Dock	Obtiene o establece el borde del contenedor principal al que está acoplado un control.
Document	Obtiene o establece un valor que indica el documento del que se va a obtener una vista previa.
Enabled	Obtiene o establece un valor que indica si el control puede responder a la interacción del usuario.
Font	Obtiene o establece la fuente del texto que muestra el control
ForeColor	Obtiene o establece el color de primer plano del control.
Location	Obtiene o establece las coordenadas de la esquina superior izquierda del control en relación con la esquina superior izquierda de su contenedor.
Rows	Obtiene o establece el número de páginas que aparecen en la pantalla en sentido vertical.
StartPage	Obtiene o establece el número de página de la página superior izquierda.
Zoom	Obtiene o establece un valor que indica el tamaño con el que aparecerán las páginas.

Y las propiedades para el control PrintPreviewDialog:

Propiedad	Descripción
AcceptButton	Obtiene o establece el botón del formulario que se activa cuando el usuario presiona la tecla ENTRAR.
AutoScale	Obtiene o establece un valor que indica si en el formulario se permite el desplazamiento automático.
AutoScroll	Obtiene o establece un valor que indica si en el formulario se permite el desplazamiento automático.
BackColor	Obtiene o establece el color de fondo del control
BackgroundImage	Obtiene o establece la imagen de fondo que se muestra en el control.
Document	Obtiene o establece el documento del que se desea la vista previa.
...	Las propiedades de un formulario: ShowIntTaskbar, StartPositiom, Menu, MaximizeBox, ControlBox, Icon, TransparencyKey, ...

### 3.8. Cuadro de diálogo de configuración de página PageSetupDialog

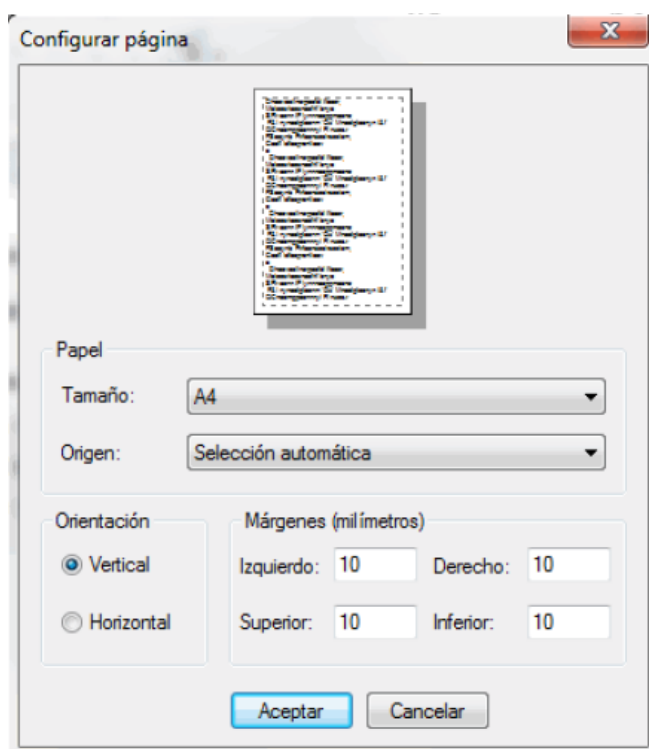
 **PageSetupDialog** Este control muestra un cuadro de diálogo que permite al usuario establecer los detalles para la impresión. Permite establecer márgenes, encabezados y pie de página y orientación de la página.

Para configurar su comportamiento utilizaremos como siempre sus propiedades:

Propiedad	Descripción
AllowMargins	Obtiene o establece un valor que indica si está habilitada la sección de márgenes del cuadro de diálogo.
AllowOrientation	Obtiene o establece un valor que indica si está habilitada la sección de orientación (horizontal o vertical) del cuadro de diálogo.
AllowPaper	Obtiene o establece un valor que indica si se habilita la sección de papel (tamaño y origen de papel) del cuadro de diálogo.
AllowPrinter	Obtiene o establece un valor que indica si el botón Impresora está habilitado
Document	Obtiene o establece un valor que indica el PrintDocument del que se obtendrá la configuración de página.
MinMargins	Obtiene o establece un valor que indica los márgenes mínimos que se permite seleccionar al usuario en centésimas de pulgada.
ShowHelp	Obtiene o establece un valor que indica si está visible el botón Ayuda.
ShowNetwork	Obtiene o establece un valor que indica si está visible el botón Red.

Por ejemplo, podemos utilizar la propiedad AllowMargins para habilitar la edición de márgenes y la propiedad ShowNetwork para que muestre el botón “Red”:

```
With PageSetupDialog1
    .AllowMargins=True
    .ShowNetwork=True
    .ShowDialog()
End With
```





## 4. Crear componentes personalizados

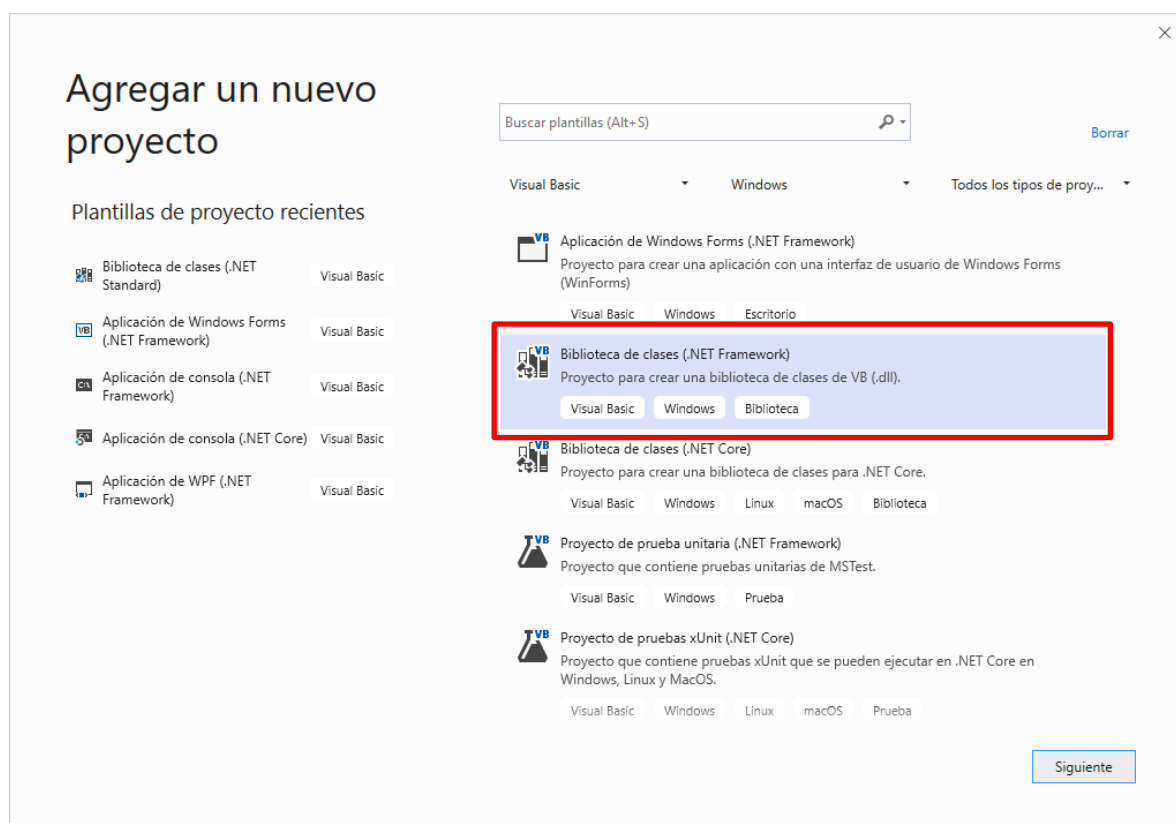
Una de las capacidades más interesantes de .NET es la de poder crear componentes y controles personalizados. Vamos a hacer dos ejemplos que serán un buen punto de partida porque serán como una plantilla para nosotros y será fácil ampliarlos para nuestros programas ya que lo complicado es crear la base del control. En el primer caso será un componente y en el segundo un control. Digamos que la principal diferencia es que el componente no tiene interfaz de usuario y el control sí.

### Ejemplo 1. Crear un componente personalizado

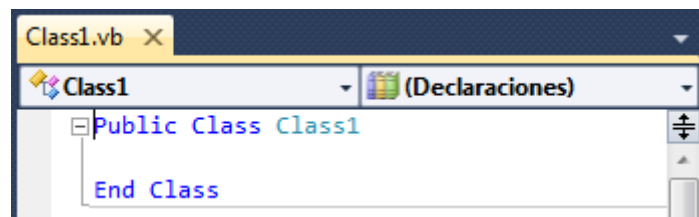
En este ejemplo crearemos un proyecto de biblioteca de clases para un componente. Primero, añadiremos constructores, destructores y una propiedad, finalmente lo incluiremos en un programa. En el IDE para este ejemplo debemos tener abiertas por comodidad la lista de tareas, el explorador de soluciones, las propiedades y el cuadro de herramientas.

#### 1. Crear el proyecto de biblioteca de clases

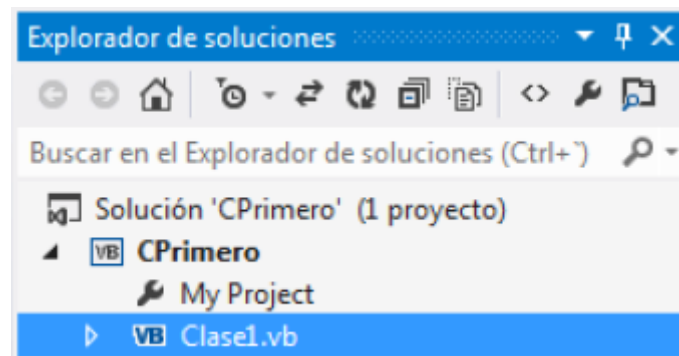
Para crear un componente necesitamos crear un proyecto de biblioteca de clases, así que seleccionamos nuevo proyecto y elegimos uno de este tipo:



Al cargarse se muestra el siguiente código de la Clase Class1:



Ahora en el explorador de soluciones hacemos clic en "Class1.vb" y en la página de propiedades le cambiamos el nombre a "Clase1.vb" y almacenamos el proyecto.



## 2. Añadir constructores y destructores

Ya sabemos qué son estos elementos. Son las instrucciones que se ejecutarán para la creación y eliminación de la clase. La función de nuestro componente es la de mantener en memoria un contador de cuántos objetos "clase1" están en memoria. Cuando se inicialice "clase1" ejecutaremos un incremento de un contador y cuando se elimine lo decrementaremos.

En la ventana de código añadimos el código necesario para almacenar el número total de instancias y el número ID de cada instancia:

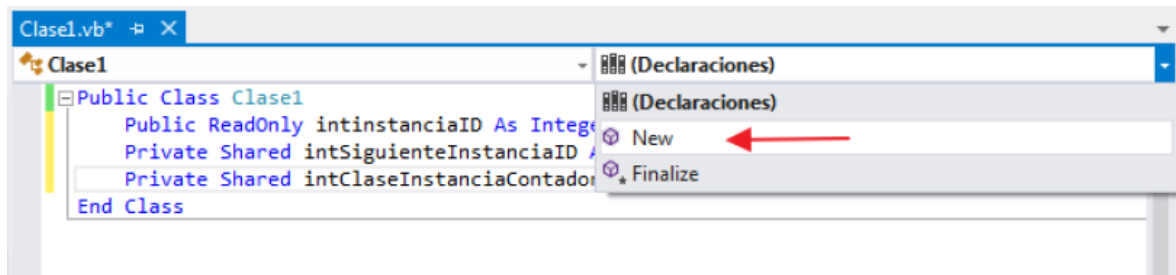
```
Public Class Clase1
    Public ReadOnly intInstanciaID As Integer

    Private Shared intSiguienteInstanciaID As Integer = 0
    Private Shared intClaseInstanciaContador As Integer = 0

End Class
```

Las variables compartidas como intClaseInstanciaContador o intSiguienteInstanciaID se inician la primera vez que se hace referencia a la clase "Clase1". Todas las instancias de "Clase1" que acceden a estos miembros utilizarán las mismas ubicaciones de memoria. El miembro de sólo lectura como intInstanciaID sólo puede establecerse desde el constructor.

Sigamos, ahora vamos a escribir código en la parte "Public Sub New", es decir, el constructor predeterminado para la clase "Clase1". Para que nos aparezca este código lo podemos seleccionar de la lista desplegable de la derecha en el IDE:

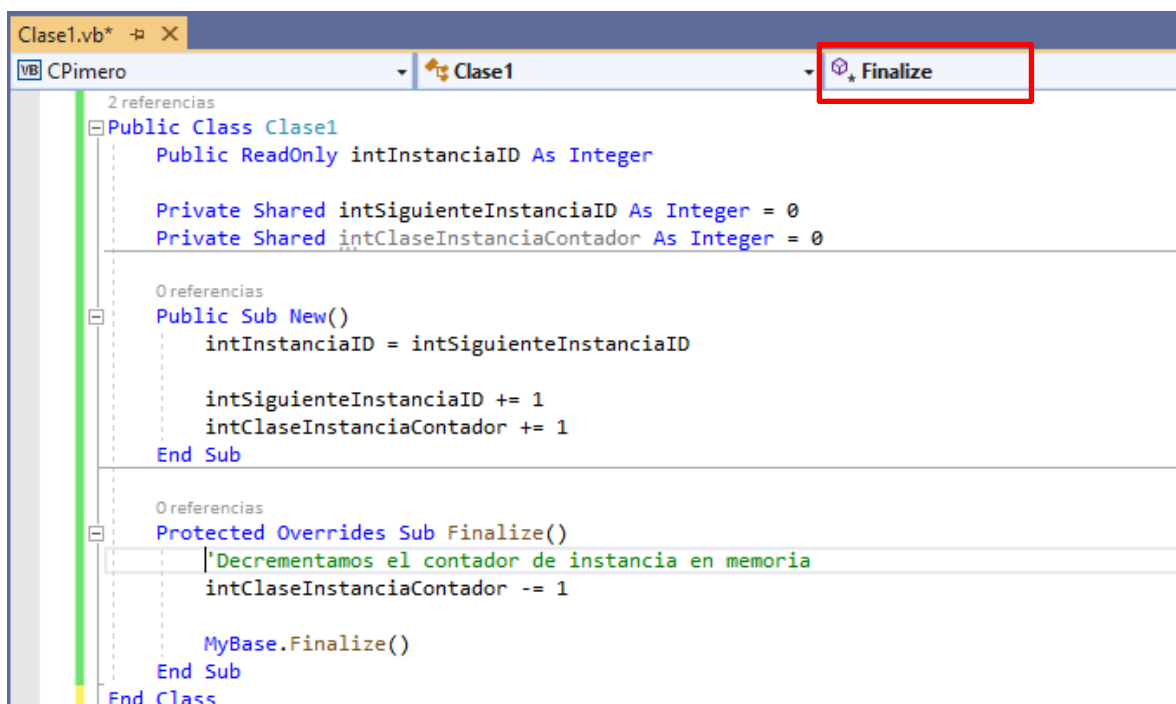


Añadiremos el siguiente código en el constructor:

```
Public Sub New()
    instanciaID = siguienteInstanciaID

    siguienteInstanciaID = siguienteInstanciaID + 1
    claseInstanciaContador = claseInstanciaContador + 1
End Sub
```

Con lo que conseguimos el objetivo que es que cada vez que se cree podamos incrementar el contador. Ahora en el destructor escribimos:



Ya tendríamos todo listo en este componente. Ahora vamos a añadir una propiedad a esta clase y luego ya probaremos el código.

### 3. Añadir una propiedad a la clase

Ahora vamos a añadir una única propiedad a nuestra clase. La propiedad compartida `claseInstanciaContador` almacena el número de objetos "Clase1" en memoria.

Para crear una propiedad para esta clase añadiremos la siguiente declaración a la clase:

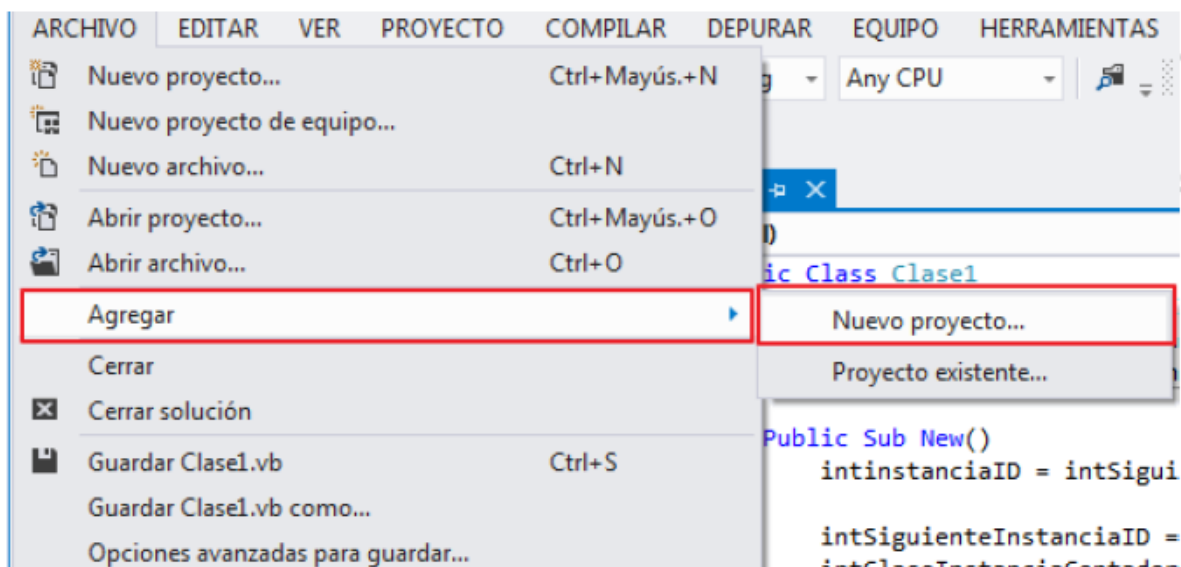
```
Public Shared ReadOnly Property ContadorInstancias() As Integer
    Get
        Return intClaseInstanciaContador
    End Get
End Property
```

Que corresponde a la nomenclatura estándar para que podamos ver una propiedad desde el exterior de las clases.

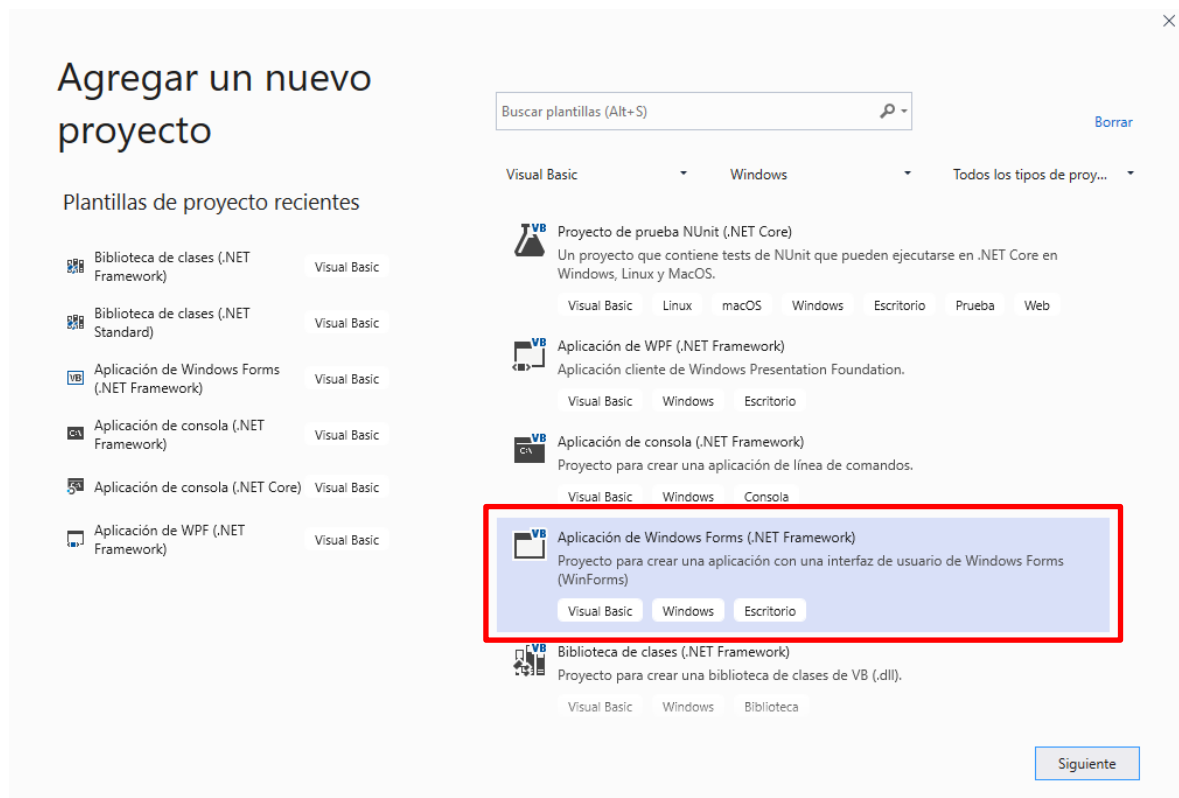
#### 4. Probar el componente

Para probar el componente que hemos creado necesitamos un nuevo proyecto cliente que utilice este componente. Para realizar esto necesitaremos establecer este proyecto como proyecto de inicio para asegurarnos de que será el primero en ejecutarse cuando se inicie la solución. Llamaremos al proyecto "TestClase1". Para añadirlo al proyecto:

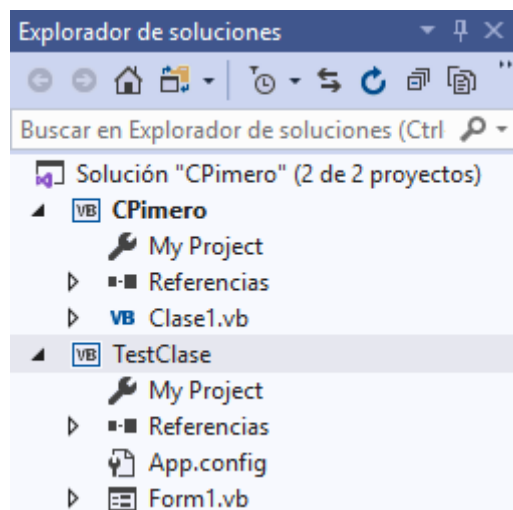
En el menú Archivo seleccionamos *Agregar*, luego *Nuevo Proyecto* para abrir el cuadro de diálogo Agregar Nuevo proyecto:



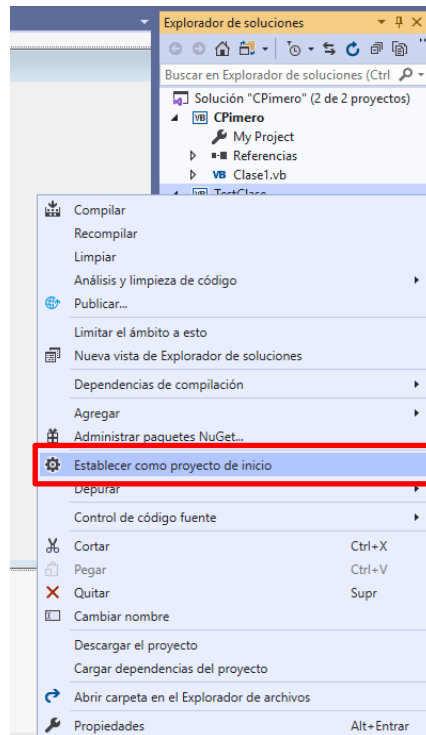
Seleccionamos ahora Aplicación Windows y le ponemos de nombre "TestClase":



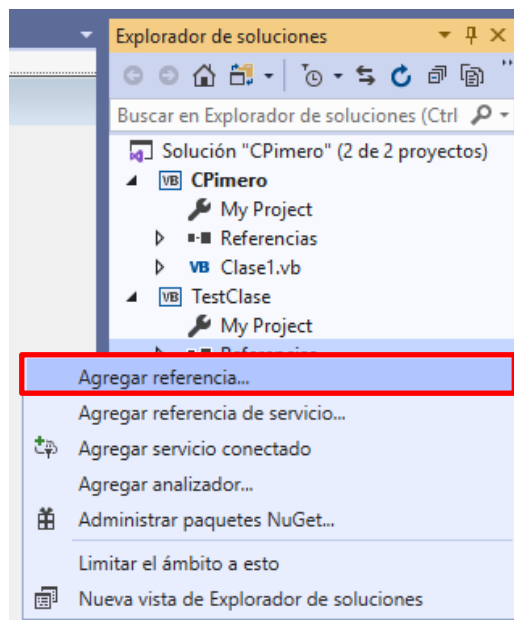
Ahora tenemos dos proyectos dentro de la misma solución: la clase y el programa:



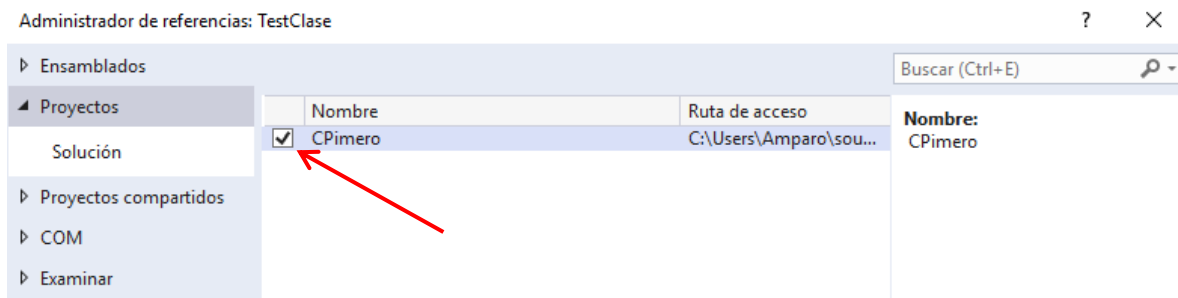
Ahora debemos activar este programa Windows. Para esto en el explorador de soluciones haremos clic con el botón derecho del ratón en "TestClase" e indicaremos que es el proyecto de inicio. Nuestra solución se compone ahora de dos proyectos y debe saber por cuál debe empezar:



Para obtener el nombre de nuestra clase en este proyecto debemos añadir una referencia a nuestro proyecto. Para añadir una referencia al proyecto de biblioteca de clase en el explorador de proyectos, hacemos clic con el botón derecho en "Referencias" y seleccionamos "Agregar Referencia":

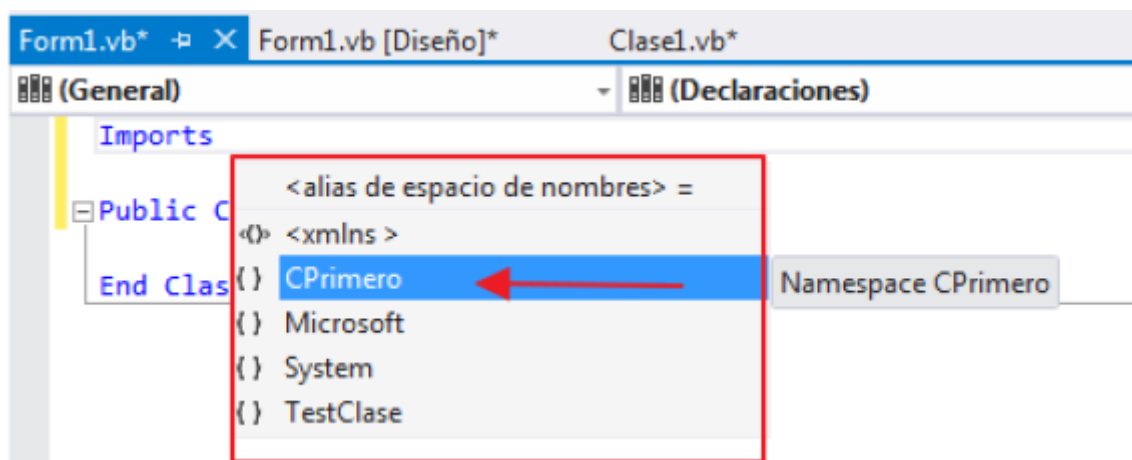


Seleccionamos la ficha "Proyectos", y hacemos clic en el proyecto:



Veremos que se añade a la lista de componentes de la parte inferior y pulsamos Aceptar. Finalmente pulsamos en "Ver código" del formulario "Form1.vb" de este proyecto para poder trabajar con él.

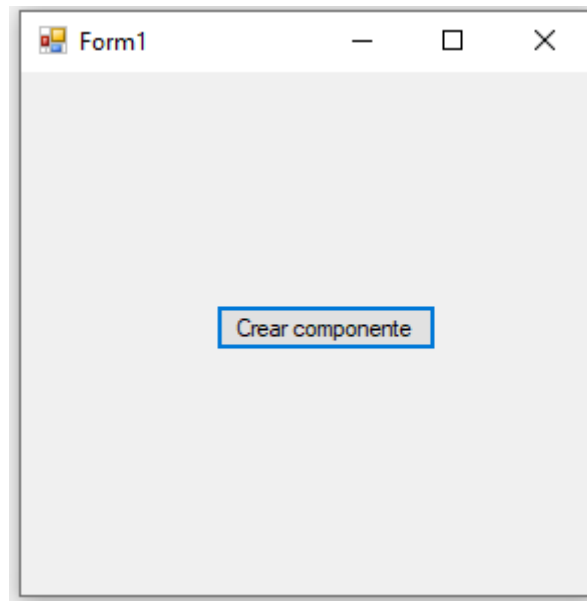
Lo que estamos realizando es para poder utilizar esta referencia en nuestro proyecto. Finalmente nos queda importarla, al hacer esto podemos referirnos a la clase como "Clase1" omitiendo el nombre de la biblioteca. Para añadir la instrucción de importación añadiremos esta instrucción a la lista de "Imports" de la parte superior del código del Form1:



Además vemos que al escribir la palabra "Imports" aparecen los componentes disponibles.

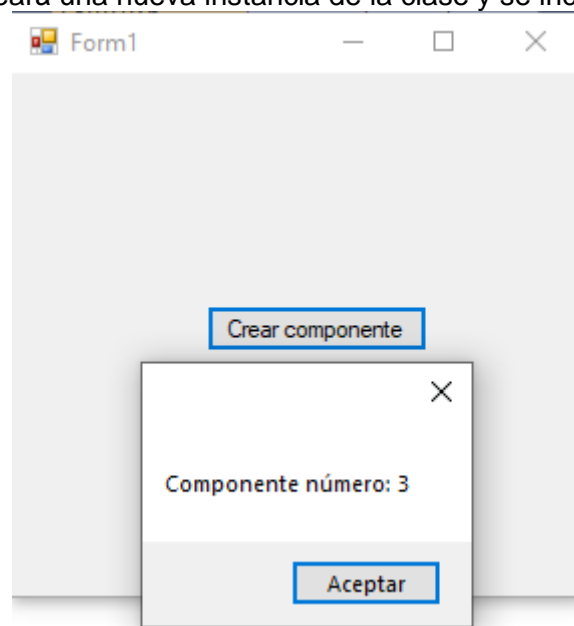
## 5. Utilizar el componente

Ya podemos utilizar el componente en nuestro programa, vamos a añadir un botón en el formulario para crear nuevas instancias de "Clase1". Añadimos un botón y le ponemos en el evento "Click":



```
Public Class Form1
    Private Sub btnCrear_Click(sender As Object, e As EventArgs) Handles btnCrear.Click
        Dim instancia As New CPimero.Clase1
        MessageBox.Show("Componente número: " & instancia.contadorInstancias)
    End Sub
End Class
```

Así que en cada clic se creará una nueva instancia de la clase y se incrementará el contador.





## Ejemplo 2. Crear un control personalizado

Un control es un componente con interfaz de usuario. Al crearlo, aparecerá en el cuadro de herramientas como un control más que podremos incluir en nuestras aplicaciones.

La creación de controles tiene muchas similitudes con la creación de los componentes, pero veamos los problemas. Supongamos que disponemos de un control que utilizamos habitualmente y de cual estamos relativamente contentos, excepto por un inconveniente, le falta algo o no hace exactamente lo que queremos. Muchas veces necesitaremos precisamente esto, modificar un control existente y no crear uno nuevo que sería muy costoso en tiempo.

Esto ya sabemos que lo podemos hacer gracias a la herencia: podemos heredar la capacidad de un control. Por ejemplo, a partir de un control de botón de comando: podemos crear otro con los matices que queramos ampliar y colocarlo en la barra de herramientas.

Para este ejemplo, crearemos un control personalizado que herede de un botón de comando con un pequeño cambio y es que tenga un color de fondo diferente.

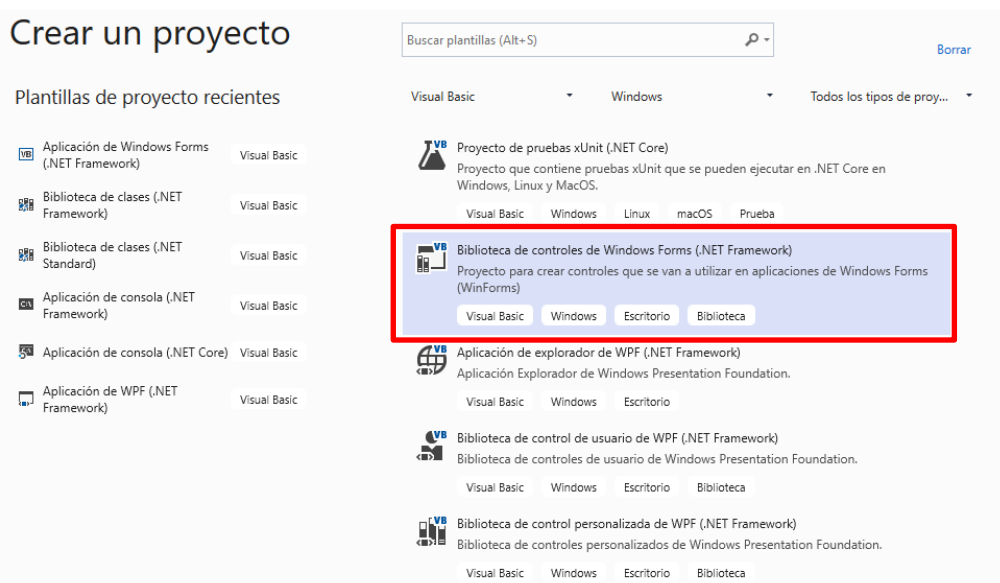
Es un pequeño cambio pero una vez aprendamos la forma de hacer estas modificaciones podremos modificar otros controles para modificar o mejorar su aspecto y funcionalidades. Comencemos con el proyecto.

### Descripción

En este ejemplo crearemos un control personalizado heredado del control de botón de opción de Windows. Nuestro control personalizado será prácticamente igual que el de Windows pero con el color de fondo en blanco. Ten activado el explorador de soluciones, el cuadro de herramientas, la ventana de propiedades y la lista de tareas.

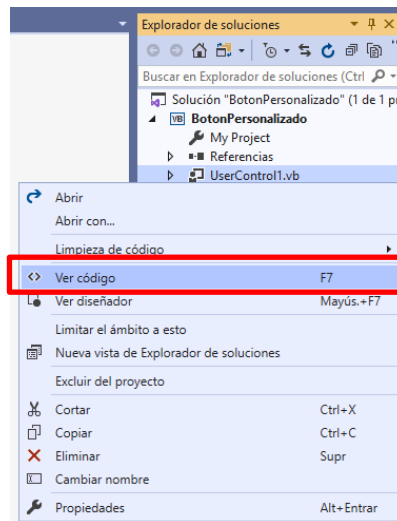
### Creación del proyecto

Necesitamos crear un proyecto para crear un componente. Por suerte entre las plantillas para crear proyectos tenemos uno que es una "Biblioteca de controles de Windows Forms":



Llamaremos al proyecto "BotónPersonalizado".

En el explorador de soluciones haz clic con el botón derecho en el archivo "UserControl1.vb" y selecciona Ver código desde el menú:

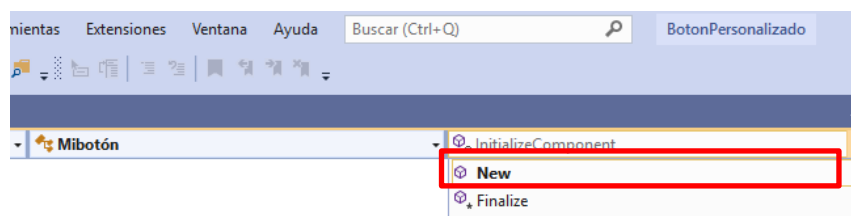


Ahora vamos a cambiar el nombre del archivo y de la clase. En el explorador de soluciones haz clic derecho sobre el archivo "UserControl1.vb" y elige "Cambiar nombre". Le cambiamos el nombre a "MiBotón.vb"

### Añadir constructores y destructores

Ahora debemos añadir los constructores y destructores para controlar la forma en la que "MiBotón" debe inicializarse y eliminarse. Nuestro control debe ser igual que el botón pero con el fondo en blanco.

Para añadir el código en el control escribe "Public Sub New", que es el constructor predeterminado de "MiBotón" o elige *New* del desplegable de la derecha y el código se genera automáticamente.



Después de llamar a "InitializeComponent" añade este código para que establezca el color de fondo a blanco.

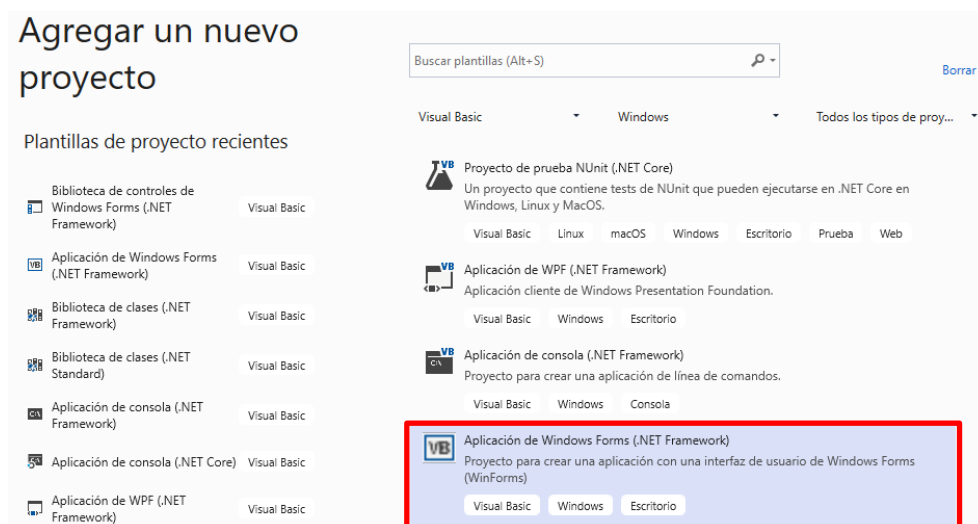
```
Public Class Mibotón
    ' Esta llamada es exigida por el diseñador.
    InitializeComponent()

    ' Agregue cualquier inicialización después de la llamada a InitializeComponent().
    Me.BackColor = Color.White
End Sub
End Class
```

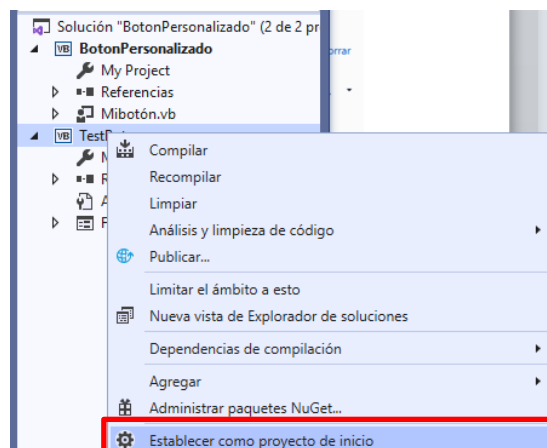
## Probar el componente

Para probar el componente necesitaremos crear un proyecto que se comporte como cliente de este control:

1. En el menú **Archivo** seleccionamos **Agregar** y luego **Nuevo Proyecto**.
2. En las plantillas de proyecto seleccionamos aplicación de Windows Forms. Como nombre podemos poner "TestBotón":

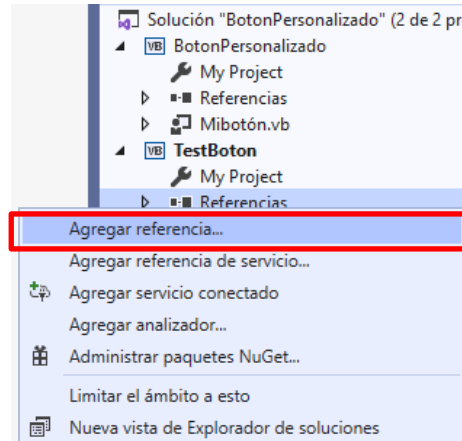


3. En el explorador de soluciones, haz clic con el botón derecho en "TestBotón" y establéclo como proyecto de inicio:



Para poder disponer de los nombres de los controles completos necesitamos añadir una referencia al proyecto de la biblioteca de controles de Windows. Para añadir una referencia:

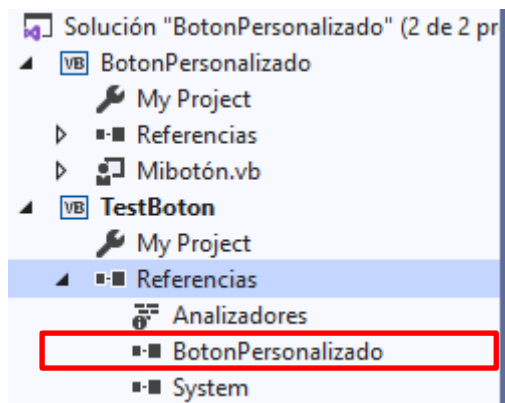
1. En el Explorador de soluciones, haz clic con el botón derecho en "TestBotón" y selecciona "Agregar Referencia"



2. En la ventana de la Agregar referencia, en la ficha "proyectos", haz clic en Aceptar:



3. Observa ahora que en *Referencias* se ha añadido esta referencia. Así nuestro nuevo control estará disponible en la aplicación "TestBotón":



4. En el explorador de soluciones, haz clic con el botón derecho en el formulario "Form1.vb" y selecciona "Ver código".

Si añadimos una instrucción de "Imports" permitiremos que hagamos referencia a "BotónPersonalizado" omitiendo su nombre de biblioteca. Esto hace más fácil el uso de componentes. Para añadir una instrucción de este tipo escribe en la parte superior del código:

```
Imports BotónPersonalizado

Public Class Form1

End Class
```

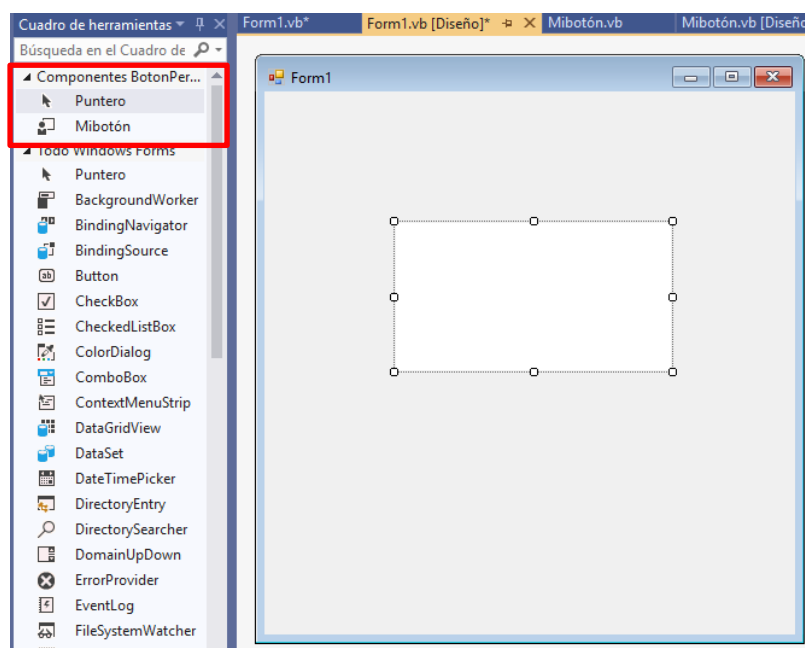
### Utilizar el componente

Ya podemos utilizar el control en nuestro proyecto, necesitamos añadir un botón a nuestro formulario para crear una nueva instancia de "MiBotón"

Para utilizarlo simplemente declaramos un nuevo objeto de este tipo y lo mostramos:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    Dim botón As New Mibotón
    botón.Show()
End Sub
```

Ahora el IDE te mostrará la disponibilidad de la clase y métodos de "Mibotón". Si lo añadimos desde el IDE:



Y vemos que funciona correctamente al poner el color en blanco.