LA PROGRAMACION ORIENTADA A OBJETOS.

En el pasado, con los lenguajes procedurales tradicionales como Basic, Pascal, C o COBOL, el programador controlaba la operativa de ejecución de una aplicación. Un programa controlaba cuando se abrían y cerraban las ventanas y el movimiento del cursor de un campo a otro. El programador determinaba dicha secuencia codificándola en el programa.

En las aplicaciones modernas con interfaces graficas, es el usuario el que gestiona la secuencia de eventos que se producen en un programa en ejecución. El usuario determina cuándo comiénzala aplicación, cuándo de abre o se cierra una ventana, cuándo se introduce el siguiente campo o cuándo finaliza la aplicación. En lugar del programa o el programador, es el usuario el que controla los eventos.

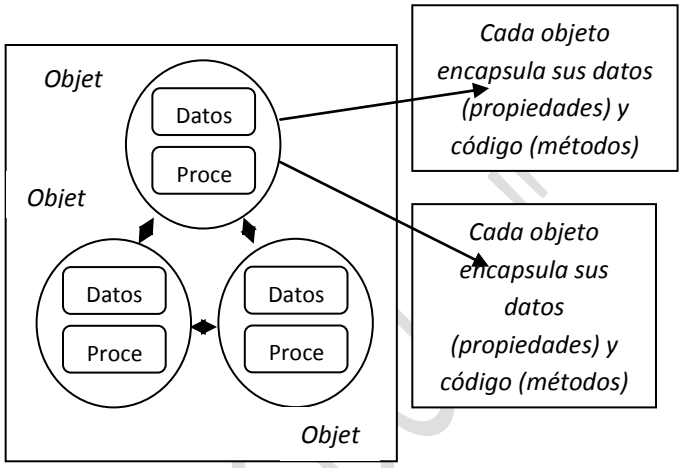
En un entorno en el que el usuario es el que provoca los eventos a través de una interfaz grafica de usuario, es fácil escribir programas con los nuevos lenguajes orientados a objetos y eventos.

Aunque Visual FoxPro admite la programación estándar por procedimientos, se ha ampliado la capacidad del lenguaje parea proporcionar la potencia y la flexibilidad propias de la programación orientada a objetos.

El diseño orientado a objetos y la programación orientada a objetos representan un cambio de perspectiva con respecto a la programación estándar por procedimientos: En lugar de pensar en el flujo del programa desde la primera hasta la ultima línea de código, se debe pensar en la creación de objetos: componentes autocontenidos de una aplicación que tienen funcionalidad privada además de la funcionalidad que se puede exponer al usuario.

ESTRUCTURA DE UNA APLICACIÓN BAJO EL ENFOQUE POO

Una aplicación orientada a objetos no es mas que una colección de objetos, cada uno de los cuales posee sus propias características o datos que determinan su comportamiento, además posee sus propios métodos o código asociado, los cuales van a responder a las acciones que se den sobre dicho objeto.



PROPIEDADES, METODOS Y EVENTOS.

En términos generales, podemos decir que cuando uno de los elementos que conforman un sistema o aplicación son considerados como objetos, llámense tablas, base de datos, procedimientos, controles, ventanas, etc. Cada uno de estos objetos posee una serie de características denominadas Propiedades y están expuestos a una serie de acciones por parte del usuario por medio de los cuales se permitirá su manipulación o manejo. A estas acciones que el usuario puede realizar en cada uno de estos objetos se les denomina Eventos. Ahora bien, cada evento tiene asociado un código (instrucciones) que será ejecutado cuando se produzca el evento al cual está asociado. A este código de acción asociado a un evento se le denomina Método.

Según lo expuesto, un objeto posee Propiedades, Métodos y Eventos.

CLASE Y OBJETOS.

Las clases y los objetos están estrechamente relacionados, pero no son lo mismo. Una clase es la generalización de un objeto, es decir, el plano o esquema bajo el cual se contribuye o define un objeto. Una clase contiene información sobre el cual debe ser la apariencia (propiedades) y el comportamiento de un objeto) métodos), por lo que, al definir objetos, éstos adoptaran las características de la clase bajo la cual se

define. A la acción de definir un objeto en base a una clase predefinida se le denomina instanciamiento. Por ejemplo, un esquema eléctrico y de diseño de un teléfono seria algo similar a una clase. El objeto, o una instancia de dicha clase seria el teléfono en sí.

Todas las propiedades, eventos, y métodos de un objeto se especifican en la definición de clase. Además, las clases tienen las siguientes características.

- ✓ Encapsulamiento.
- ✓ Subclases
- ✓ Herencia.

ENCAPSULAMIENTO. Es la propiedad o característica bajo la cual se agrupan los datos y el código asociado a una determinada clase ocultando de esta manera la complejidad que ésta puede tener. La ventaja de ignorar los detalles internos de un objeto para poder centrarse en los aspectos del objeto que necesita utilizar se denomina abstracción.

Continuando con el ejemplo del teléfono, al realizar o recibir una llamada, no nos interesa el proceso interno que se realiza en éste, como por ejemplo el proceso de conversión se centra específicamente en la utilización del aparato en sí.

HERENCIA. La herencia es el mecanismo que nos va a permitir reutilizar código de manera fácil y ordenada. La herencia se produce al crear subclases con toda la funcionalidad de la clase base. Cada una de estas subclases puede tener sus propias características o métodos las cuales se agregan y los ya heredados. Por otra parte, cualquier modificación que realicemos en una clase base, se ve reflejada automáticamente en todas las subclases que heredan de ella.

Continuando con el ejemplo del teléfono, si la clase es un teléfono básico, este podrá tener subclase que tenga la funcionalidad del teléfono original y todas las características que especializadas que desee asignarle.

POLIFORMISMO. Consiste en que varios objetos, aunque partan de clases distintas, tengan métodos o propiedades con el mismo nombre y que actúen de distinta forma. Tiene una relación directa con la herencia, ya que un método o propiedad heredado se puede sobreescribir y por lo tanto puede actuar de distinta forma de lo que hacia en la clase base.

LA PROGRAMACION ORIENTADA A OBJETOS EN VISUAL FOXPRO.

Los conceptos explicados anteriormente describen la Programación Orientada a Objetos de una manera genérica, que puede ser aplicada cualquier lenguaje de programación que trabaje bajo este enfoque, ahora nos centraremos en como Visual FoxPro maneja estos conceptos.

TIPOS DE CLASES EN VISUAL FOXPRO.

Hay dos tipos principales de clases en Visual FoxPro y por extensión, de objetos: las clases de control y las clases contenedoras. La siguiente tabla muestra todas las clases definidas y disponibles.

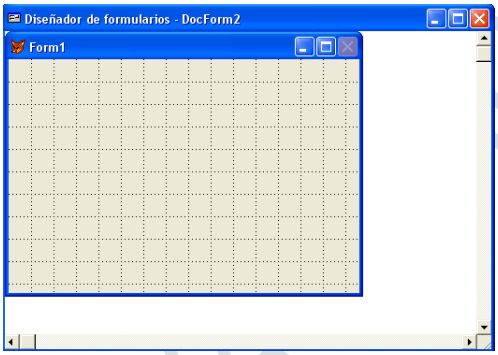
CONTROLES	CONTENEDORES
Casilla de verificación	Columna
Cuadro combinado	Grupo de botones de comando
Botón de comando	Formulario
Control	Conjunto de formularios*
Personal*	Barra de herramientas.
Cuadro de edición	Marco de paginas
Encabezado	Pagina
Imagen	Cuadricula
Linea	Grupo de botones de opción
Etiqueta	Contenedor
Cuadro de lista	
Control dependiente OLE	
Control contenedor OLE	
Forma	
Control numérico	
Cuadro de texto	

Cronometro*

Los contenedores son aquellas clases que pueden contener dentro de sí a otras clases. Los objetos contenidos se denominan objeto miembro y automáticamente pasan a depender de su contenedor. La tabla siguiente muestra la relación entre las clases contenedores y los objetos que pueden contener.

CONTENEDOR	OBJETO QUE PUEDE CONTENER
Grupo de botones de comando	Botones de comando
Formularios	Marcos de páginas, cualquier control, contenedores.
Conjunto de formularios	Formularios, barras de herramientas.
Barra de herramientas	Cualquier control, contenedor, marco de página.
Marcos de página	Paginas
Pagina	Cualquier control, contenedor.
Cuadricula	Columnas de cuadricula
Columnas de cuadricula	Encabezados de columnas, cualquier otro objeto que sea un formulario, conjunto de formularios, columnas de cuadricula y barra de herramientas.
Grupo de botones de opción	Botones de opción
Control	Cualquier control
Contenedor	Cualquier control

EL FORMULARIO. Un formulario es un objeto contenedor desde el cual se puede desarrollar cualquier aplicación Visual FoxPro. Un formulario puede contener controles a través de los cuales el usuario podrá ver, modificar e introducir datos en una base. Los formularios ofrecen una interfaz entre una base de datos y el usuario lo que permite realizar las tareas de administración de información de la manera más sencilla posible.



PROPIEDADES, EVENTOS Y MÉTODOS.

Un formulario almacena u objetos, cada uno de estos controles tienen sus propias características o atributos (en Visual FoxPro se denominan Propiedades) que determinan su apariencia o un aspecto de su comportamiento. Estas propiedades se pueden establecer durante el diseño de formulario (tiempo de diseño) o durante la ejecución del mismo (tiempo de ejecución), Por ejemplo, el botón de comando (control CommandButton) tiene las propiedades Caption y BackColor que permiten indicar el texto que mostrará el botón de comando y el color del mismo.

Cuando se ejecuta un formulario y el usuario realiza ciertas acciones sobre los controles, como ingresar datos o hacer click en un botón de comando, se producen ciertos eventos desencadenados por el usuario.

PROPIEDADES DE UN FORMULARIO.

Un formulario tiene propiedades que afectan su apariencia y comportamiento y responde a ciertos eventos durante la ejecución del programa, Veamos algunas propiedades:

1. **ActiveControl**, Hace referencia al control activo de un objeto.
   SINTAXIS:
   Objeto.ActiveControl.Propiedad[= Valor]
   Valores:

Propiedad La propiedad que se devuelve o establece.

Valor La propiedad actual o nueva.

Si el objeto esta activo, ActiveControl hace referencia al control que tiene el enfoque. Se generará un error si el objeto no esta activo.

ActiveForm, Permite tener acceso a las propiedades y los métodos del Form activo en u conjunto de formularios.

SINTAXIS:

Objeto.ActiveForm.Propiedad[= Valor]

- 0 -

Objeto.ActiveForm.Método

Valores:

Propiedad Cualquier propiedad del formulario activo contenido en el conjunto de formularios.

Valor El valor existente o un nuevo valor de la Propiedad.

Metodo Cualquier método del formulario activo contenido en el conjunto de formularios.

2. **AlwaysOnTop**, Controla si un formulario siempre está situado sobre las demás ventanas abiertas.

   SINTAXIS:

   Objeto.AlwaysOnTop[= lExp]

   Configuraciones:

   Verdadero (.T.) El objeto Form siempre está visible.

   Falso (.F.) (Predeterminado) El objeto Form puede estar oculto por otra ventana.

3. **AutoCenter**, Determina si el formulario se centra a sí mismo en la ventana principal de Visual FoxPro la primera vez que se despliega.

   SINTAXIS:

   Objeto.AutoCenter[= IExp]

   Valores para la propiedad AutoCenter:

   Verdadero (.T.) El Form se centra en la ventana de Visual FoxPro.

   Falso (.F.) (Predeterminado) El Form se sitúa en las coordenas especificadas en las propiedades Left y Top.

4. **BackColor y ForeColor**, Estas propiedades permiten establecer el color de fondo y de primer plano para mostrar texto en un control.

   SINTAXIS:

   Objeto.BackColor[= nColor]

   Objeto.ForeColor[= nColor]

   Donde, nColor especifica un valor único de color.

   Utilice la función RGB ( ) para especificar un color cualquiera. La sintaxis de esta función es:

   RGB(rojo%, verde%, azul%)

   Los componentes rojo, verde y azul están representados por un número entre 0 y 255 que especifican el nivel de rojo, verde y azul, respectivamente. La función RGB convierte los tres colores componentes en un nColor compuesto.

   Valores para algunos colores estándar:

COLOR	CONFIGURACIÓN RGB
Blanco	255, 255, 255
Negro	0, 0, 0,
Rojo	255, 0, 0
Amarillo	255, 255, 0
Verde	0, 255, 0
Azul	0, 0, 255

5. **BorderStyle**, Determina el estilo de borde de un formulario.

   SINTAXIS:

   Objeto.BorderStyle[= nvalor]

   Valores para la propiedad BorderStyle:

   0 Ninguno.
   1 Sencillo.
   2 Borde línea doble.
   3 Borde de tamaño ajustable.

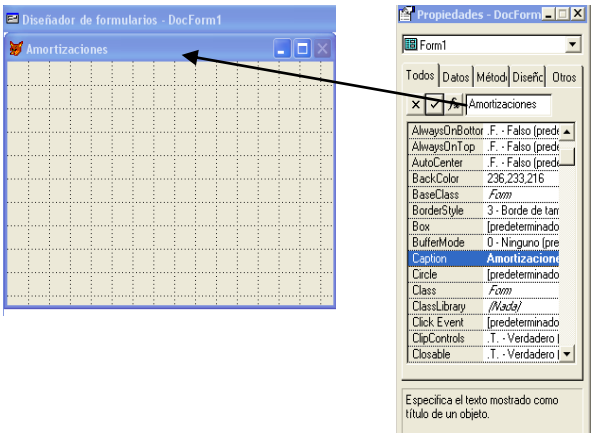
6. **Caption,** Utilice esta propiedad para establecer el texto que se mostrará en la barra de titulo del formulario.

   SINTAXIS:

Objeto.BackColor [= ncolor]

Ejemplo:

THISFOR.Captión="Amortizaciones"

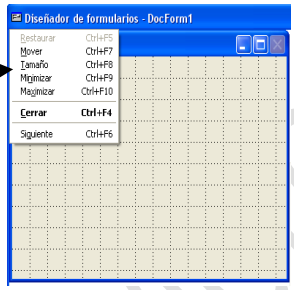


7. **Closable**, Establezca a verdadero a .T. esta propiedad y el formulario se cerrará haciendo doble clic en el icono del menú desplegable del formulario.

   SINTAXIS:

   Objeto.Closable[= IExp]

Menú desplegable



8. **ControlBox,** Específica si se muestra en tiempo de ejecución el icono del menú despegable en la esquina superior izquierdo del formulario.
9. **Icon**, Determina qué archivo de icono mostrará un formulario en cuando es minimizado.
10. **MaxButton.** Establezca a falsa .F. esta propiedad para inhabilitar el botón maximizar del extremo derecho de la barra del título

    SINTAXIS:

    Objeto.MaxButton[= IExp]
11. **MinButton**. Establezca a False .F. esta propiedad para inhabilitar el botón minimizar del extremo derecho de la barra del titulo.

    SINTAXIS:

    Objeto.MinButton[= IExp]
12. **MOVABLE**. Dtermina sí el usuario puede mover un formulario en tiempo de ejecución arrastrándole desde la barra de titulo.

    SINTAXIS:

    Objeto.Movable[= IExp]

    Valores para la propiedad Movable:

    Verdadero (.T.) (Prederminado) Se puede mover el objeto.

    Falso (.F.) No se puede mover el objeto.
13. **Name**. Nombre del objeto al que se hace referencia en el código.

    SINTAXIS:

    Objeto.Name[= cNombre]
14. **Picture.** Determina el archivo de gráfico que se va a mostrar como icono del formulario.

    SINTAXIS:

    Objeto.Picture[= cNombArch]

    El grafico puede ser de tipo: BMP, WMF, JPG o GIF.
15. **ScaleMode**. Determina la unidad de medida para las coordenadas de un objeto cuando se usan métodos gráficos o se colocan controles.

    SINTAXIS:

    Objeto.ScalaMode[= nModo]

    Valores para ScaleMode:

    0 **Fóxeles** (Equivale a la altura máxima y el ancho promedio de un carácter en la fuente actual.)

    3 **Pixeles** (Unidad más pequeña de resolución de un monitor o de una impresora).
16. **Scrollbars**. Determina el tipo de barras de desplazamiento que tendrá un formulario.

    SINTAXIS:

    [Formulario]Control.ScrollBars[= nTipo]

    Valores para la propiedad ScrollBars:

    0 (Predeterminado) Ninguno.

    1 Horizontal.

    2 Vertical,

    3 Vertical y horizontal.
17. **ShwTips.** Permite determinar si se muestra información sobre herramientas para los controles del formulario.

    SINTAXIS:

    Objeto.ShowTips [= lExpr]

Valores:

Verdadero (.T.)	Muestra información.
Falso (.F.)	(Predeterminado) No muestra información.

18. TitleBar.

EL CONTROL CUADRO DE TEXTO (TextBox) 

El control Cuadro de texto permite al usuario el ingreso de datos y su almacenamiento en variables de memoria o campos; también permite mostrara información que el usuario pude modificar.

PROPIEDADES DEL CONTROL TextBox.

1. **BordeColor**. Especifica el color del borde de un objeto.
   SINTAXIS:
   Objeto.BorderColor[= nColor]
   El Valor nColor especifica el valor de color del borde. Utilice la función RGB () para especificar un color. Establezca la propiedad SpecilEffect en 1 – Normal para aplicar la propiedad BorderColor a un cuadro de texto.
2. **Century.** Especifica si se mostrará la parte de una fecha relativa al siglo.
   SINTAXIS:
   Objeto.Century[= nValor]
   Valores que puede adoptar Nvalor:
   0 Off. No muestra la parte de la fecha relativa al siglo.
   1 (Predeterminado) On. Muestra la parte d ela fecha relativa al siglo.
   2 El Valor SET CENTURY determina si se mostrará la parte de la fecha realtiva al siglo.
3. **ControlSource.** Especifica el origen de datos con el que esta vinculado el objeto, es decir, de donde provienen los datos. Puede ser una variable o un campo.
   SINTAXIS:
   Objeto.ControlSource[= cNombre]
   Donde CNombre es una variable o un campo.
   Una vez se ha establecido la propiedad ControlSource, la propiedad Valué siempre tiene el mismo tipo y valor de datos que la variable o el campo en el que está establecida la propiedad ControlSource.
4. **DateFormat.** Especifica el formato para valores Date y DateTime.
   SINTAXIS:
   Objeto.BorderColor[= nColor]
   Valores que puede adoptar nValor.
   - 0 (Predeterminado). El formato lo determina SET DATE.
   - 1 Norteamericano (dd/mm/aa)
   - 2 ANSI (aa.mm.dd)
   - 3 Britanico (dd/mm/aa)
   - 4 Italiano (dd.mm.aa)
   - 5 Frances (dd/mm/aa)
   - 6 Aleman (dd.mm.aa)
   - 7 Japones (aa/mm/dd)
   - 8 Taiwanés (aa/mm/dd)
   - 9 EE.UU. (mm-dd-aa)
   - 10 MDA (mm/dd/aa)
   - 11 DMA (dd/mm/aa)
   - 12 AMD (aa/mm/dd)
5. **Format**. Especifica el formato de entrada y salida de la propiedad Value de control.
   Control.Format [=cFunción]
   Los valores validos de Cfunción:
   1 Convierte a mayúsculas los caracteres alfabéticos.
   $ Muestra el símbolo de moneda.
   ^ Muestra datos numéricos utilizando anotación científica.
   A Solo admite características alfabéticas. No admite espacios o signos de puntuación.
   D Usa el formato de datos especificado como SET DATE.
   E Usa el formato de datos especificado como SET DATE BRITISH.

K Selecciona el contenido del cuadro de texto cuando éste toma el control.

L Muestra ceros a la izquierda (en lugar de espacios).

M Incluido por compatibilidad con ve3rsiones anteriores.

R Permite múltiplos opciones preestablecidas. La lista de elementos delimitados por comas.

T Elimina los espacios en blanco iníciales y finales del campo de entrada.

6. HOURS.
7.

CONTROL GRUPO DE OPCIONES.(Option Group)

Este control permite crear un grupo de botones de opción de las que el usuario sólo puede elegir una. Cada botón de opción contenido en el usuario sólo puede elegir una. Cada botón de opción contenido en el control OptionGroup es un objeto con propiedades y eventos propios. Al activar un botón de opción se desactiva de inmediato el botón anterior.

ALGUNAS PROPIEDADES DE CONTROL Option Group:

1. **ButtonCount.** Determina el número de botones del control OptionGroup.

   SINTAXIS:

   Control.ButtonCount[= nNúm]

   Donde nNúm es el número de botones que tendrá el control.

2. Name. Nombre del control a nivel de código.

   SINTAXIS:

   Objeto.Name[= nNombre]

3. Value. Utilece está propiedad para determinar qué botón del grupo causo el evento.

   SINTAXIS:

   Objeto.Value[= nValor]

   Desarrollemos una aplicación que nos permita modificar el tamaño, fuente y color de una cadena de caracteres.

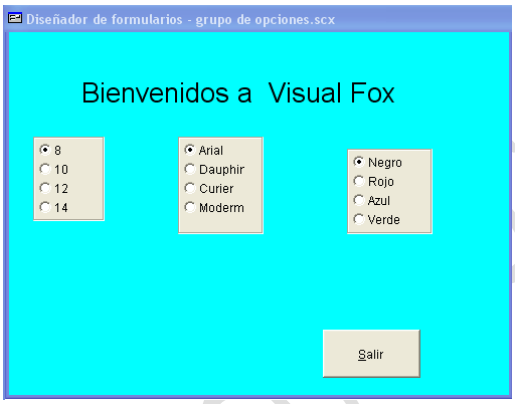
   PASOS:

   a) Cree un nuevo formulario.
   b) Agregue un control Label, tres controles OptionGroup y un control CommandButton en el formulario.
   c) Asigne las siguientes propiedades a los controles.

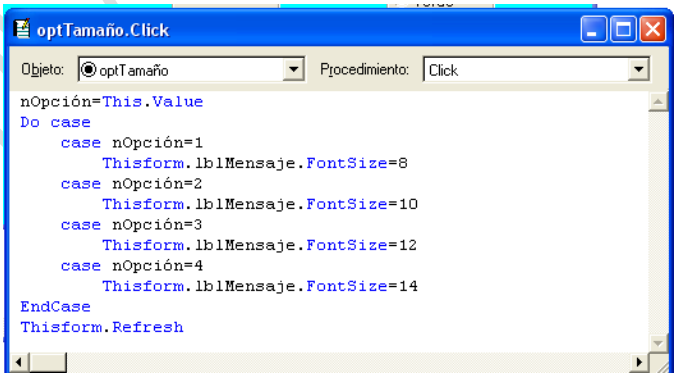
Control	Propiedad	Valor.
Label1	Name	lblMensaje
	Caption	Bienvenidos a Visual FoxPro.
OptionGroup 1	Name	optTamaño
	ButtonCount	4
OptionGroup 2	Name	optFuente
	ButtonCount	4
OptionGroup 3	Name	optColor
	ButtonCount	4
CommanButton 1	Name	cmdSaalir
	Caption	\<Salir.

Hasta ahora, el formulario debe tener el siguiente aspecto:
Modifiquemos las propiedades de los botones de opción (OptionButton) contenidos en los controles Grupo de opciones.

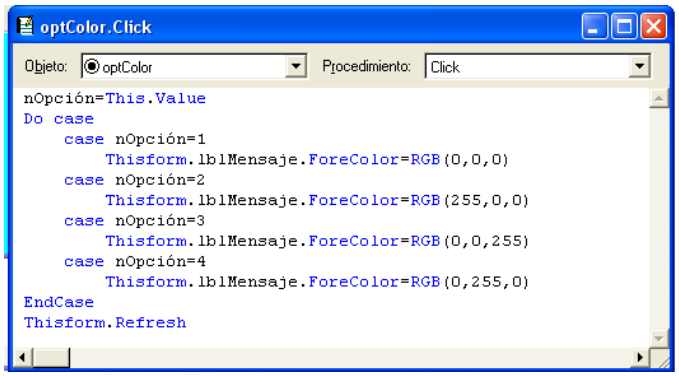
- Situe puntero en el control optTamaño, haga clic con el botón derecho del mouse y seleccione la opción Modificar del menú contextual.
- Haga clic en el botón de opción Option 1 para seleccionarlo.
- En la ventana de propiedades, asigne el valor "8" a la propiedad Caption del botón de opción seleccionado.
- Modifique las propiedades de todos los botones de opción. Al finalizar, el formulario debe ser similar a la siguiente ilustración.



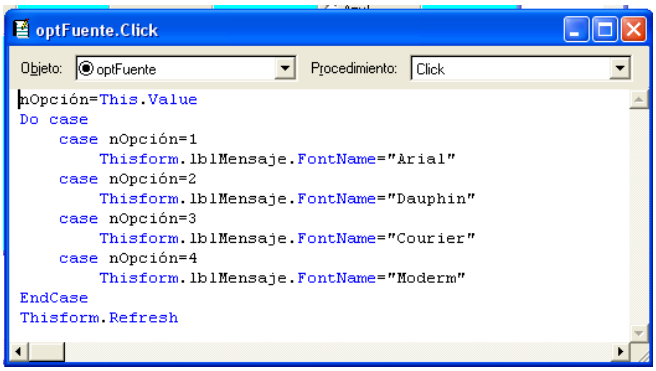
- Asociemos un procedimiento al control optTamaño accionado por el contenido en el control Label debe cambiar.
  - En el formulario haga doble clic en el control optTamaño para visualizar la ventana código.
  - Despliegue el cuadro de lista Procedimiento y elija el evento Clik. Digite el siguiente Código:



  - Para optFuente.



- Para optColor



-

- Haga clic en el comando Guardar del menú archivo. En el cuadro de dialogo. Guardar como, elija una carpeta e introduzca un nombre para Formato. scx.

EL CONTROL CASILLA DE VERIFICACIÓN (CheckBox)

Este control permite presentar varias opciones de las que el usuario puede elegir una o mas de una. Una casilla de verificación admite dos estados básicos: 0 y 1. Si esta activada el valor contenido en la propiedad Value es 1; en caso contrario, el valor contenido es 0.

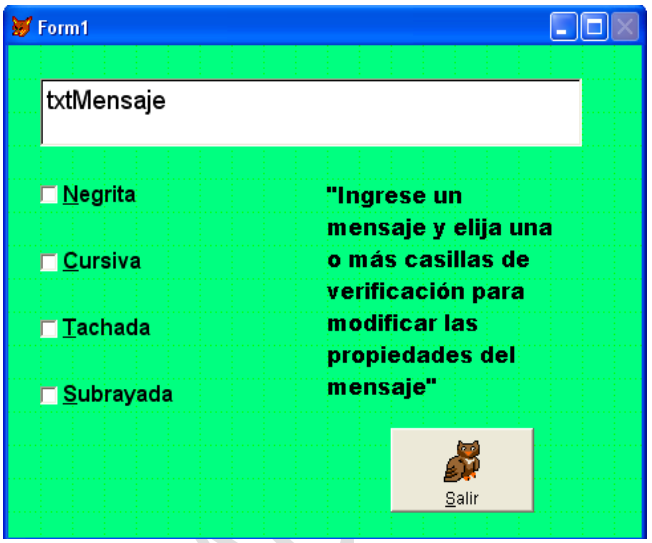
ALGUNAS PROPIEDADES:

1. **Caption**. Descripcion que acompaña al control CheckBox.
   SINTAXIS:
   Objeto Caption[=cTexto]
   El valor cTexto especifica el texto que se muestra con un objeto.
   Asigne la tecla de acceso al control CheckBox. Para ello, incluya los caracteres "\<" antes del carácter que desea asignar como tecla de aceso. Al pulsar ALT y el carácter especificado, el enfoque se moverá al control. El efecto es el mismo que hacer clic en el control.
2. **ControlSource.** Especifica el origen de datos, puede ser un campo o una variable numérica.
   Sintaxis:
   Objeto.ControlSource[= cNombre]
   Donde, cNombre es una varieble o un campo.
   Establecida la propiedad ControlSource como un campo o una variable o el campo en el que está establecida la propiedad ControlSource.
3. **Value**. Especifica el estado actual del control CheckBox.
   SINTAXIS:
   [Formulario.]Control.Value[= cValor]
   Los valores para la propiedad Value son:
   0 (Predeterminado) Desactivada.
   1 Activada

EJEMPLO: Desarrollemos una aplicación que permita aplicar el atributo de negrita, cursiva, tachada y subrayada a una cadena de caracteres.

PASOS:

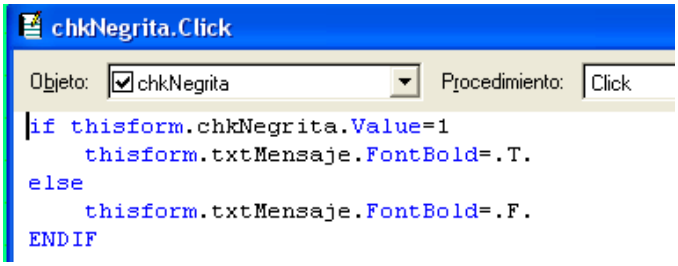
- Cree un nuevo formulario.
- Agregue un cuadro de texto, cuatro casillas de verificación, una etiqueta y un botón de comando en el formulario.
- Modifique las propiedades de los controles tal como lo muestra la siguiente ilustración.



La siguiente tabla muestra las propiedades asignadas a los controles:

Control	Propiedad	Valor.
TextBox	Name	txtMensaje
CheckBox	Name	chkNegrita
	Caption	\<Negrita
CheckBox	Name	chkCursiva
	Caption	\<Cursiva
CheckBox	Name	chkTacahada
	Caption	\<Tachada
CheckBox	Name	chkSubrayada
	Caption	\<Rayada
	Caption	"Ingrese un Mensaje y elija una o más casillas de verificación para modificar las propiedades del mensaje.
CommanButton 1	Name	cmdSalir
	Caption	\<Salir.

En el formulario, haga doble clic en la casilla de verificación Negrita. En la ventana Codigo digete el siguiente procedimiento accionada por el evento Click:



En el formulario, haga doble clic en la casilla de verificacion Cursiva y en la ventana de Codigo digete el procedimiento:

chkCursiva.Click

Objeto: chkCursiva Procedimiento: Click

```
if thisform.chkCursiva.Value=1
    thisform.txtMensaje.FontItalic=.T.
else
    thisform.txtMensaje.FontItalic=.F.
ENDIF
```

chkTachado.Click

Objeto: chkTachado Procedimiento: Click

```
if thisform.chkTachado.Value=1
    thisform.txtMensaje.FontStrikeThru=.T.
else
    thisform.txtMensaje.FontStrikeThru=.F.
ENDIF
```

En el formulario, haga doble clic en la casilla de verificacion Tachada y en la ventana de Codigo digete el procedimiento:

En el formulario, haga doble clic en la casilla de verificacion Subrayada y en la ventana de Codigo digete el procedimiento:

chkSubrayada.Click

Objeto: chkSubrayada Procedimiento: Click

```
if thisform.chkSubrayado.Value=1
    thisform.txtMensaje.FontUnderline=.T.
else
    thisform.txtMensaje.FontUnderline=.F.
ENDIF
```

Guarde el formulario con el nombre ControlCheck. Haga clic en el botón Ejecutar 

CONTROL GRUPO DE BOTONES DE COMANDO.

El control Grupo de botones de comando (CommandGroup) se utiliza para crear un conjunto de botones de comando que pueden manipularse individualmente como grupo. Utilice este control cuando necesite utilizar botones de comando que estén relacionados en funcionalidad.

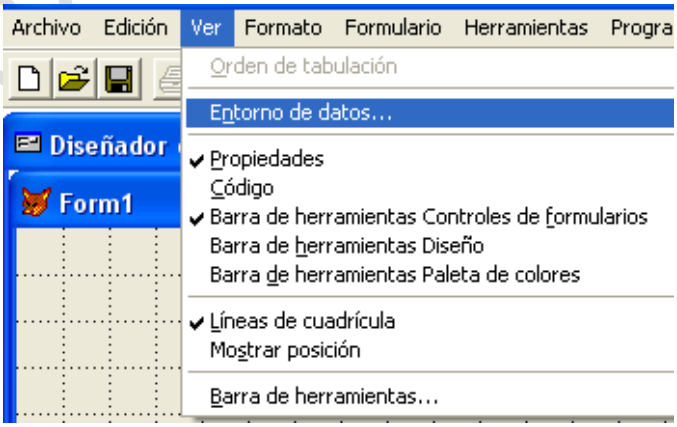
ALGUNAS PROPIEDADES:

1. **BackStyle**. Determina si el fondo del control CommandGroup será transparente u opaco.
   SINTAXIS.
   Objeto.BackStyle[=Estilo]
   Los valores posibles de BackStyle son:
   0 Transparente. Lo que esté detrás del objeto será visible. Se pasa por alto la propiedad BackColor.
   1 (Predeterminado) Opaco. La porpiedad BackColor del objeto llena el control y opaca cualquier color o grafico que haya detrás.
2. **BorderStyle**. Determina el número de botones que tendrá el control CommandGroup.
   SINTAXIS.
   Objeto.BorderStyle[= nEstilo]
   Configuraciones posibles de la propiedad BorderStyle:
   0 Ninguno.
   1 (Predeterminado) Sencillo fijo.
3. **ButtonCount**. Determina el número de botones que tendrá el control CommandGroup.
   SINTAXIS:
   Control.ButtonCount[= nNúm]
   El valor nNúm especifica el número de botones del control.
4. **Buttons**. Es una matriz que se crea al crear el grupo de botones. Utilice esta propiedad para establecer propiedades y llamar a métodos para todos los botones del grupo.
   SINTAXIS:
   Control.buttons (índice).(propiedad) = Valor
   o bien Control.Buttons (índice).(Método)
   Valores:
   Indice Un número entero entre 1 y el número de botones especificado en la propiedad ButtonCount.
   Propiedad Una propiedad de un control CommandButton.
   Valor El valor de Propiedad.
   Metodo Un método de un control CommandButton.
5. **Value**. Utilice esta propiedad para determinar que botón del grupo causo un evento.

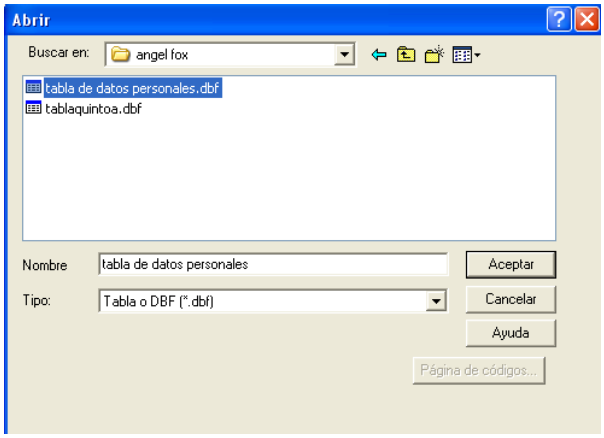
EJEMPLO:

a) Cree un nuevo formulario.

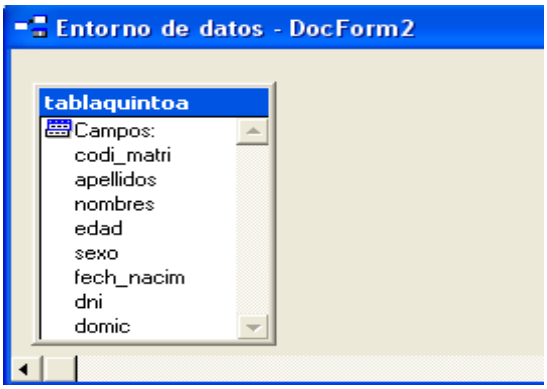
b) Seleccione el comando **ENTORNO DE DATOS** del Menú **VER.** Visual FoxPro muestra el entorno de datos y el cuadro de dialogo Agregar tabal o vista.



c) En Tablas de la base de datos, elija la tabla Clientes.dbf y haga clic en **AGREGAR.**

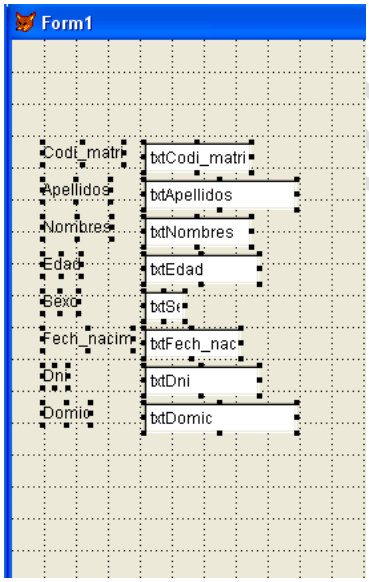


d) Aparece el entorno de datos con los campos de la tabla antes seleccionado.

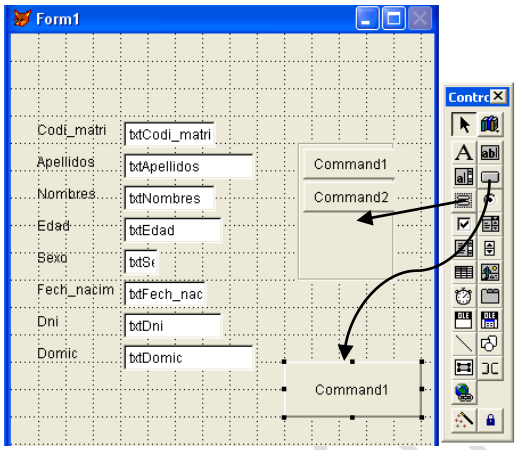


e) En el entorno de datos, seleccione los campos Codi_matri, apellidos, nombres, etc. Lo haces de esta manera primero seleccione el campo Codi_matri, luego mantenga presionado SHIF y haga clic en el ultimo campo de la tabla osea en este caso en domic, luego arrastre hacia el formulario, pulsando con el botón DERECHO, y en la pantalla se visualizará lo siguiente:

f) Seleccione CREAR MULTIPLES CONTROLES AQUÍ. Y los campos seleccionados son agregados en el formulario.



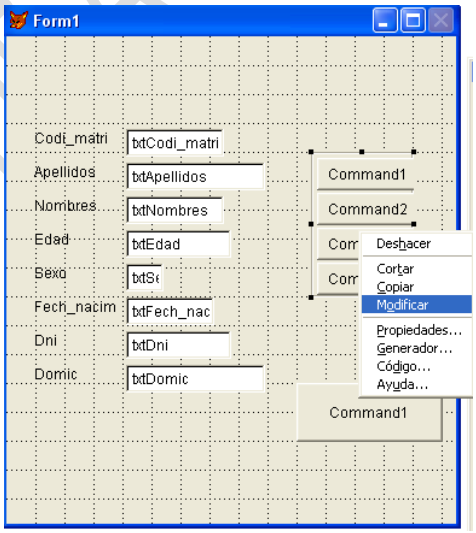
g) Agregue un control CommandGroup y un control CommandButton.



h) Seleccione el control CommandGroup y, en la ventana de propiedades modifique las propiedades:

- NAME **gbc**Puntero
- ButtonCount 4

Modifiquemos las propiedades de los botones incluidos en el control CommandGroup. Para ello haga clic con el botón derecho del mouse en el control CommandGroup y, en el menú contextual que aparece, seleccione el comando **MODIFICAR.**



i) Una vez de que se seleccione MODIFICAR, el grupo de botones de comando aparece un color **celeste** y seleccione command 1 del grupo de comandos y en propiedades cambie: CAPTION, asigene el valor de **I< ,** command 2 cambie a **< ,** command 3 cambie a **> ,** command 4 cambie a **I> ó (primer registro, retroceder un registro, avanzar un registro, o situarse ultimo registro).**

j) Seleccione el botón de comando independiente y modifique las propiedades: NAME (cmdSalir), CAPTION (\<Salir).

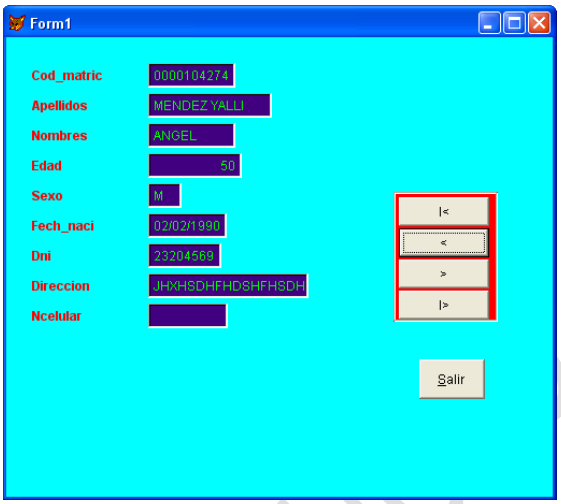
k) Asociemos un procedimiento al control CommandGroup accionado por el evento CLICK. Haga doble clic en el control CommandGroup. En la ventana código digite el siguiente código:

Objeto **gbcPuntero**	evento

nOpción=this.Value

```
if Not BOF()
this.buttons[1].Enabled=.T.
this.buttons[2].Enabled=.T.
endif

if Not EOF()
this.buttons[3].Enabled=.T.
this.buttons[4].Enabled=.T.
endif
DO CASE
        case nOpción=1
                this.buttons[1].Enabled=.F.
                this.buttons[2].Enabled=.F.
                GO TOP
        case nOpción=2
                IF Not BOF()
                Skip - 1
                IF BOF()
                this.buttons[1].Enabled=.F.
                this.buttons[2].Enabled=.F.
                GO TOP
        ENDIF
    ENDIF
        case nOpción=3
                IF Not EOF()
                Skip
                IF EOF()
                this.buttons[3].Enabled=.F.
                this.buttons[4].Enabled=.F.
                GO BOTTOM
        ENDIF
    ENDIF
    case nOpción=4
                this.buttons[3].Enabled=.F.
                this.buttons[4].Enabled=.F.
                GO BOTTOM
    ENDCASE
    THISFORM.REFRESH
```



l) Seleccione el botón comando Salir y digite lo siguiente:

Objeto **cmdSalir**	evento **clic**

```
RELEASE THISFORM
```

Guarde el formulario con el nombre de ver registro de clientes. Haga clic en el botón Ejecutar:

EL CONTROL CUADRO DE LISTA (ListBox) 

Un cuadro de lista muestra una lista de elementos en la que es posible elegir uno o más elementos.

PROPIEDADES:

1. **BoundColum**, Determina que columna de un control LisBox o ComboBox de varias columnas se asociará con la propiedad Value, es decir cuando se elija un elemento de un cuadro de lista, el contenido de la fila situada en la columna especificada por esta propiedad se almacena en la propiedad Value.
   SINTAXIS:
   Control.BoundColum[=nColumna]
   nColumna especifica la columna que se asociará con la propiedad Value. El valor predeterminado de nColumna es 1.
2. **ColumnCount,** Específica el número de columnas en un control ComboBox o ListBox.
   SINTAXIS:
   Objeto.ColumnCount[=nColumna]
   Si establece ColumnCount con =, se mostrará la primera columna basada en la propiedad RowSource o en los elementos agregados con el método Additem.
3. **ColumnLines**. Indica si se han de mostrar las líneas que hay entre las columnas en un cuadro de lista.
   SINTAXIS:
   Control.ColumnList[= IExp]
   Las configuraciones posibles de esta propiedad son:
   Verdadero (Predeterminado) Las Líneas son visibles.
   Falso Las líneas no son visibles.
4. **ColumnWidths.** Especifica el ancho de cada una de las columnas definidas en la lista.
   SINTAXIS:
   Control.ColumnWidths[="AnchoCol1, AnchoCol2, ….AnchoColn]
   Por ejemplo:
   THIS COLUMNWIDTHS = "6, 9, 10"
   Especifica que las columnas 1, 2 y 3 tendrán un ancho de 6, 9 y 10 unidades de medida especificada en la propiedad ScaleMode del Form.
5. **ControlSource.** Especifica el origen de datos del que depende un objeto.
   SINTAXIS:
   Objeto.ControlSource[0cNombres]
   El valor cNombre es una variable o un campo.
   Una vez se ha establecido la propiedad ControlSource como un campo o una variable, la propiedad Value siempre tiene el mismo valor y tipo de datos que la propiedad o el campo en el que está establecida la propiedad ControlSource.
6. **List.** Una matriz empelada para tener acceso a los elementos de un control ComboBox o ListBox.
   SINTAXIS:
   Control.List(nFila [, nCol])[=cCar]
   Valores:
   nFila Especifica la fila del elemento.
   nCol Especifica la columna del elemento.
7. **ListIndex**. Devuelve el número de índice correspondiente al elemento seleccionado en un control ComboBox ó ListBox.
   SINTAXIS:
   Control.LisIndex[=nIndice]
   Valores que puede devolver la propiedad ListIndex:
   (Predeterminado) Indica que no hay elementos elegidos.
   …ListCount El índice del elemento elegido.
8. **ListCoun**t. Numero total de elementos de la lista.
   Sintaxis:
   Control.ListCount
9. **MoverBars**. Especifica si las barras de movimiento se muestran en un control de forma interactiva.
   SINTAXIS:
   CuadroLista.MoverBars[=IExp]
   Valores de la propiedad MoverBars:
   Verdadero (.T.) Muestra las barras de Movimiento.
   Falso (.F.) (Predeterminado) No muestra las barras de movimiento.
10. **MultiSelect.** Especifica si el usuario podrá realizar varias selecciones en un control ListBox.
    SINTAXIS:
    CusdroLista.MultiSelect[=nOpcion]
11. **ROwSource.** Especifica el origen de los valores en un control ComboBox ó ListBox.
    SINTAXIS:
12. **Control.RowSource**[=cNombre]
    El valor cNombre especifica el origen de los valores.
13. **RowSourceType**. Especifica el tipo de origen para los valores de un control. Disponible en tiempo de diseño y en tiempo de ejecución.
    SINTAXIS:
    Control.RowSourceType[=Origen]
14. **Selected**. Determina si un elemento esta seleccionado.
    SINTAXIS:
    [Formulario.]Control.Selected(nIndice)[=IExp]
    Utilice esta propiedad para comprobar que elementos de la lista están seleccionados. Por ejemplo, para comprobar si el cuarto elemento de un ListBox está seleccionado, ejecute el siguiente código:

```
IF ListaFrutas.Selectd(4)
        WAIT WINDOW "¡Está seleccionado!"
ELSE
        WAIT WINDOW "¡No está seleccionado!"
ENDIF
```

15. **SelectedItemBackColor y SelectedItemForeColor.** Determina el color de fondo o de primer plano para los elementos seleccionados de un ComboBox ó ListBox.
    SINTAXIS:
    Control.SelectedItemBackColor[=nColor]
    Control.SelectedItemForeColor[=nColor]
    nColor especifica un numero para representar el color.
16. **Sorted.** Determina si los elementos de la lista se ordenaran de manera alfabetica.
    SINTAXIS:
    [Formulario.]Control.Sorted[=IExp]
    El valor IExp puede adoptar los valores:

Verdadero (.T.) Los elementos se ordenaran alfabéticamente sin distinguir mayúsculas de minúsculas.

Falso (F.) (Predeterminada) Los elementos no se ordenaran alfabéticamente.

Esta propiedad solo esta disponible si la propiedad RowSourceType se ha estabelcido como 0 (Ningno) ó 1 (Valor).

17. **Value.** Contien el valor que se ha elegido en la primera columna de un cuadro de lista.

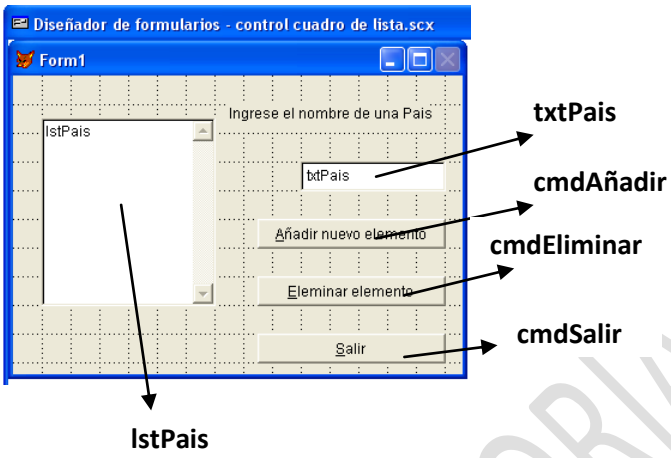
SINTAXIS:

[Formulario.]Control.Value[=nValor]

El valor nValor es la cadena de caracteres o el valor numérico que esté seleccionado en ese momento.

EJEMPLO:

DESARROLLEMOS UNA APLICACIÓN QUE PERMITE AÑADIR, ELIMINAR Y SELECCIONAR ELEMENTOS EN UNA LISTA EN TIEMPO DE EJECUCIÓN. LA INTERFAZ DEBE SER SIMILAR A LA SIGUIENTE ILUSTRACIÓN:

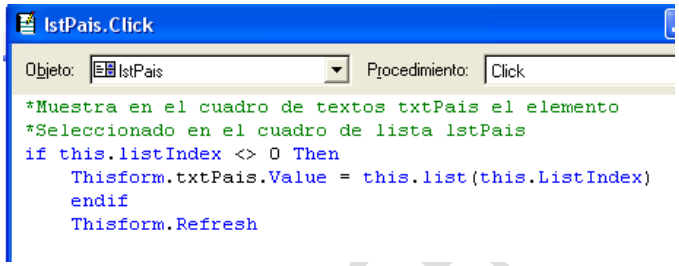


LA SIGUIENTE TABLA MUESTRA LAS PROPIEDADES DE LOS COMANDOS.

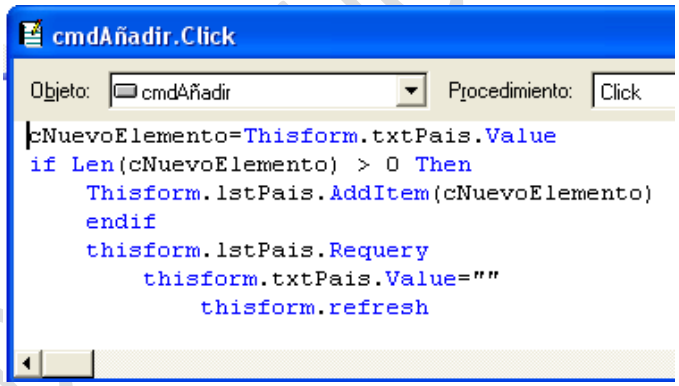
Control	Propiedad	Valor.
ListBox	Name	lstPais
	RowSourceType	0
	RowSource	(Nada)
	Sorted	.T.
Label	Caption	Ingrese el nombre de un país.
TextBox	Name	ctxtPais
CommanButton 1	Name	cmdAñadir
	Caption	\<Añadir elemento
CommanButton 2	Name	cmdEliminar
	Caption	\<Eliminar elemento
CommanButton 3	Name	cmdSalir
	Caption	\<Salir

Guarde el formulario con el nombre de Lista de países.

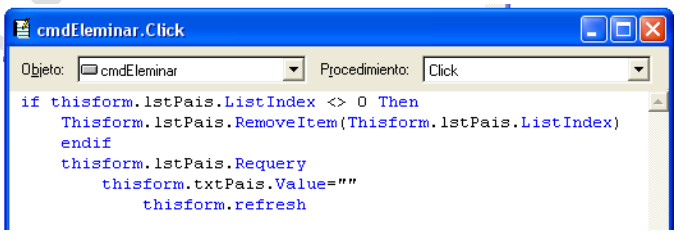
En el formulario, haga clic en el cuadro de listaPais. En la ventana código, digite el siguiente accionado por el evento Click.



En el comando AÑADIR.

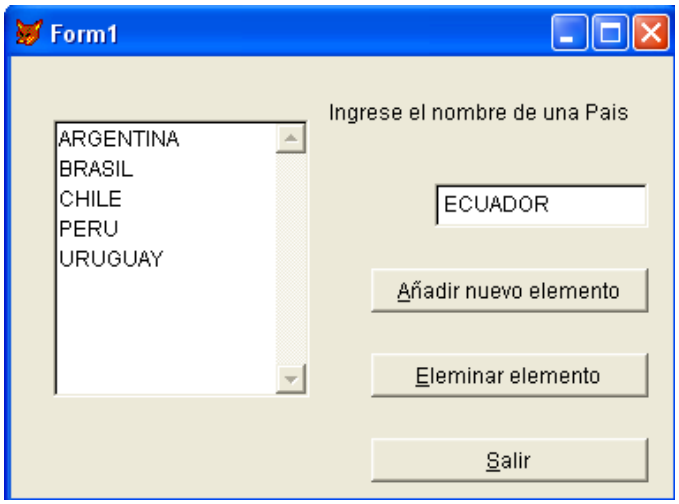


En el comando ELIMINAR



En el comando SALIR

Haga clic en el botón EJECUTAR  Ingrese el nombre de un pais en el cuadro de texto y haga clic en el botón 2Añadir nuevo elemento" para añadirlo a la lista.



Guarde su trabajo.

El Control Cuadro combinado (ComboBox)

Este control es una combinación de un cuadro de texto y un cuadro de lista desplegable. Para seleccionarlo un elemento de la lista contenida en un cuadro combinado se debe pulsar en el botón situado a la derecha del control, o bien escribir un valor manualmente en la parte del cuadro de texto.

PROPIEDADES:

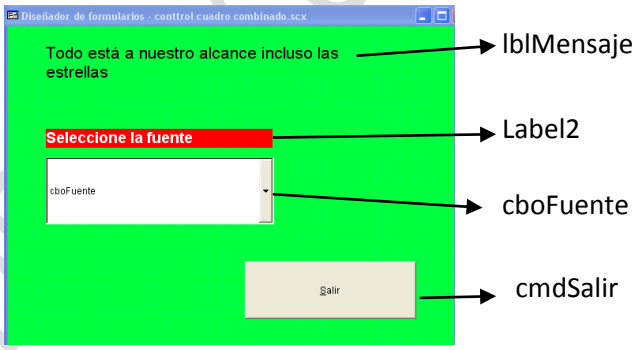
1. **ControlSource**. Especifica el campo de la tabla en el que se almacena el valor que elige o escribe el usuario.
2. **DisplayCount**. Especifica el número máximo de elementos que mostrará la lista.
   SINTAXIS:
   Objeto.displayCount[=nExp]
   El valor predeterminado de nExp es cero, lo que hace que la lista muestre 7 elementos como máximo. Si el valor 1 es asignado a nExp, la parte de lista no mostrara flechas de desplazamiento.
3. **InputMask.** Para cuadros combinados desplegables. Especifica el tipo de valores que se pueden escribir.
4. **IncrementalSearch**. Especifica si el control intenta hacer coincidir un elemento de la lista a mediad que el usuario escribe cada letra.
   SINTAXIS:
   Control.IncrementalSearch[=Exp]
   Valores de la propiedad incremental.
   Por ejemplo, para buscar la palabra "Courier", escriba C-O, etc.
5. **ListCount,** Número total de elementos de la lista. NJop está disponible en tiempo de diseño; es de solo lectura en tiempo de ejecución.
   SINTAXIS:
   Control.ListCount.
6. **RowSource.** Especifica el origen de los elementos del cuadro combinado.
7. **RowSourceType.** Especifica el tipo de origen del cuadro combinado. Los tipos de origen de fila de un cuadro combinado son iguales que los de una lista.
8. **Style.** Permite definir dos estilos de cuadros combinados: un cuadro combinado desplegable o una lista desplegable.
   SINTAXIS.
   [Formulario]Control.Style[=nTipo]
9. **Text.** Contiene el texto sin formato escrito en la parte de cuadro de texto del control ComboBox. No esta disponible en tiempo de diseño; es de solo lectura en tiempo de ejecución.
   SINTAXIS:
   Objeto.Text
10. **Value.** Especifica el estado actual de un control. Es de solo lectura para los controles Combox y ListBox.
    SINTAXIS_
    [Formulario.]Control.Value[=nValor]

EJEMPLO: DESARROLLEMOS UNA APLICACIÓN QUE MODIFIQUE LA FUENTE DE UN MENSAJE.

a) Cree un nuevo formulario.

b) Agregue dos controles Label, un control CosmboBox y un control CommandButton en el Formulario.

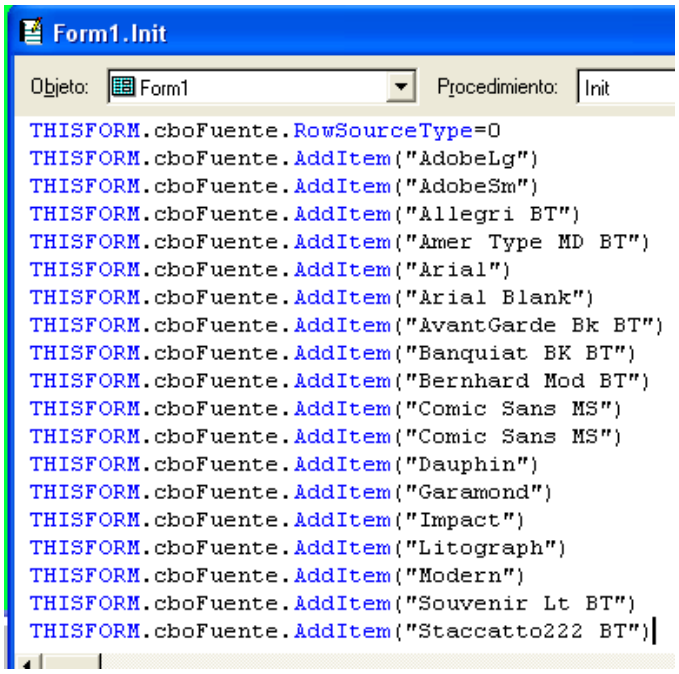
c) Asigne un nombre a cada uno de los controles tal como lo muestra la siguiente ilustración.



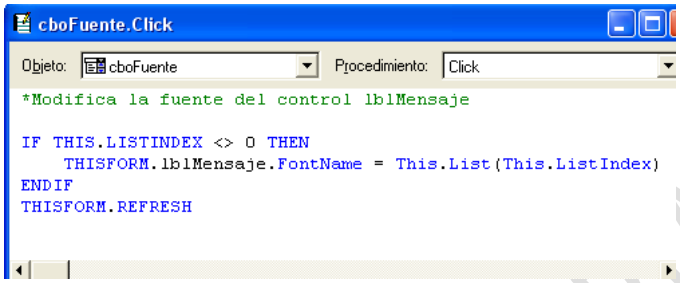
d) La siguiente tabla muestra las propiedades asignadas a los controles:

Control	Propiedad	Valor.
Label1	Name	lblMensaje
	Caption	Todo está a nuestro alcance incluso las estrellas.
	FontName	Arial.
	FontSize	16
Label2	Caption	Seleccione
Fuente		
ComboBox1	Name	cboFuente
	Style	2 – Lista desplegable
CommanButton	Name	cmdSalir
	Caption	\<Salir

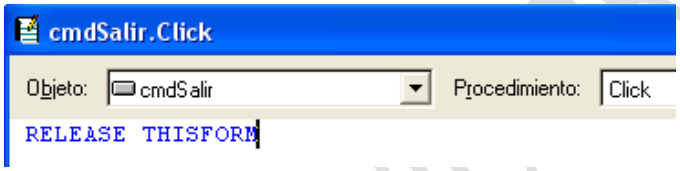
e) Asociemos un procedimiento al control Form accionado por el evento INIT. Cuando el usuario inicie el programa, este procedimiento debe añadir elementos a la lista del control cboFuente.



Para el cboFuente:



Para cmdSalir.



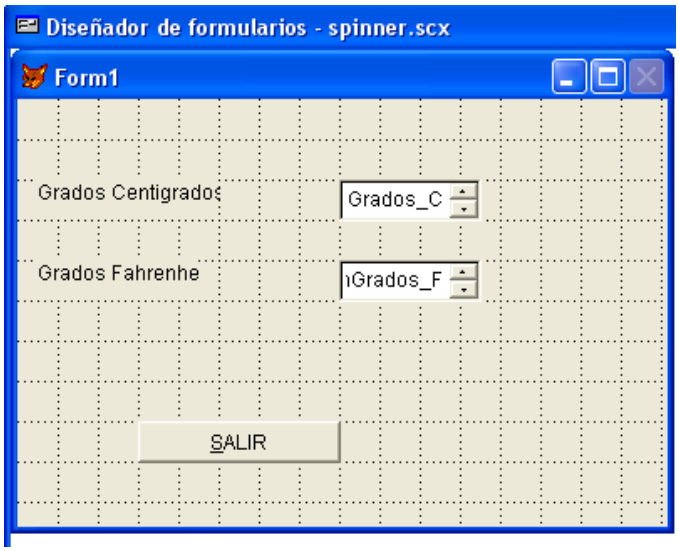
Guarde su trabajo con control cuadro combinado en su carpeta respectiva.

¡TU PUEDESSSSSSSSSSSS!.

EL CONTROL NUMERICO (Spinner)

Este control permite que el usuario pueda elegir en un intervalo de valores numéricos al "desplazarse" por los valores cuando hace clic en las flechas arriba y abajo del control Spinner utilizando el teclado.

EJEMPLO: DESARROLLEMOS UNA PLICAICON QUE CONVIERTA GRADOS CENTIGRADOS A FAHRENHEIT Y VICECERSA. LA APLICACIÓNN TENDRÁ UNA INTERFAZ SIMLAR A LA SIGUIENTE ILUSTRACIÓN:



LA SIGUIENTE TABLA MUESTRA LAS PROPIEDADES ASIGNADAS A LOS CONTROLES:

CONTROL	PROPIEDAD	VALOR
Label1	Caption	Grados Centigrados
Label2	Caption	Grados Fahrenheit
Spinnner1	Name	**spn**Grados_C
	Format	k
	Increment	1
	InputMask	99,999.99
	KeyboardHighValue	1000
	KeyboardLowVAlue	-1000
	SpinnerHighValue	1000
	SpinnerLowValue	-1000
Spinnner1	Name	**spn**Grados_F
	Format	k
	Increment	1
	InputMask	99,999.99
	KeyboardHighValue	1000
	KeyboardLowVAlue	-1000
	SpinnerHighValue	1000
	SpinnerLowValue	-1000
Command1	Name	cmdSalir
	Caption	\<Salir

FORMULA:

$$C = \frac{5}{9}(F - 32)$$

$$F = 32 + \frac{9}{5}C$$

- En el formulario haga doble clic en el control **spn**Grados_C para visualizar la ventana de código.
- Despliegue e cuadro Procedimiento y seleccione el evento **InteractiveChange.**

  ThisForm. **spn**Grados_F.Value = This.Value * 9/5 + 32
- En el formulario haga doble clic en el control **spn**Grados_F para visualizar la ventana de código.
- Despliegue e cuadro Procedimiento y seleccione el evento **InteractiveChange.**

  ThisForm. **spn**Grados_C.Value =(This.Value - 32) * 5/9
- Comando Salir:

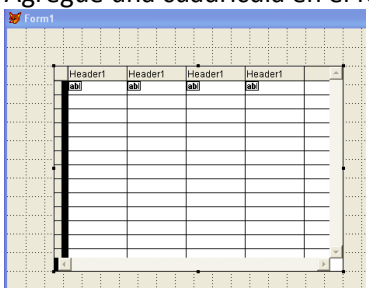
  Release ThisForm

EL CONTROL CUADRICULA (GRID).

Es una herramienta muy potente que proporciona Visual FoxPro para mostrar y manipular múltiples filas de datos. Este control puede contener columnas. Las columnas, a su vez, pueden contener encabezados y controles y métodos, lo que proporciona un gran control sobre los elementos de la cuadricula.

EJEMPLO: DESARROLLEMOS UNA APLICACIÓN QUE MUESTRE DENTRO DE UNA CUADRICULA LOS DATOS DE LA TABLA CLIENTES. DBF.

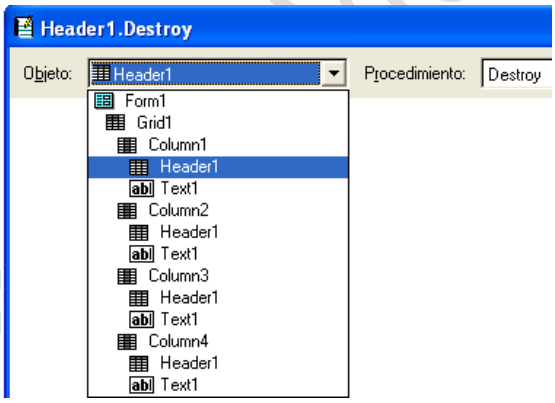
1. Cree un nuevo formulario.
2. Agregue una cuadricula en el formulario.



3. Propiedades:

PROPIEDAD	VALOR
RecordSourceType	0 – Tabla
RecordSource	D:\VFOXPRO6\Clientes.dbf
ColumnCount	4

4. Asigne el titulo "Id Cliente" al encabezado (Header1) de la primera comumna (Column1) de la cuadricula.



5.