Fundamentos de Interacción Persona-Ordenador

7. Diseño de una aplicación Windows Forms en VB.NET

Luís Rodríguez Baena (luis.rodriguez@upsam.net)

Universidad Pontificia de Salamanca (campus Madrid)
Escuela Superior de Ingeniería y Arquitectura

Programación en Windows

- Programación convencional (lineal)
  - Acciones previsibles e independientes del entorno donde se ejecutan.
  - Opciones de usuario limitadas a las posibilidades que el programador dicte.
    - ✓ El control de las opciones se hace por medio de bucles y estructuras selectivas.
  - No adecuada para entornos gráficos o multitarea.
    - ✓ El número de eventos disponibles es demasiado grande para poder controlarlos todos.
    - ✓ La elección de orden del proceso de eventos es compleja.
    - ✓ La estructura de un programa lineal no facilita la espera a que se produzcan los eventos.

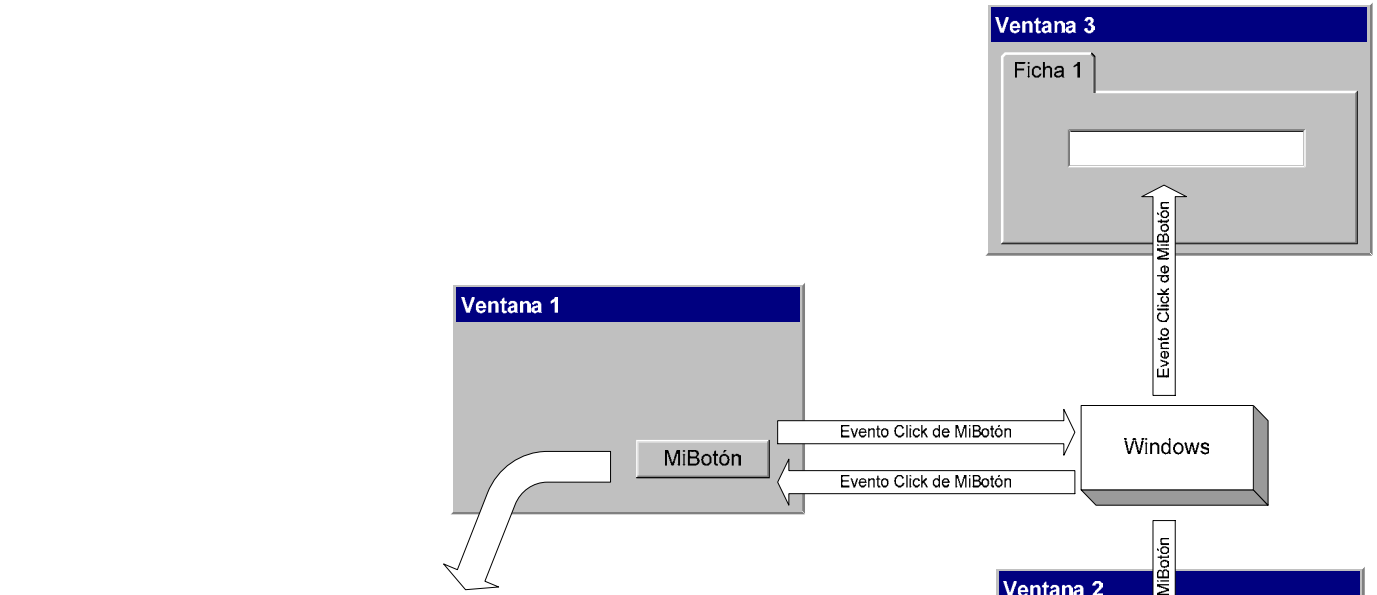
Programación en Windows (II)

- Conceptos clave en la programación en Windows.
  - Ventanas.
    - ✓ Región de la pantalla.
      - Ventanas de documentos, botones, listas desplegables, cuadros de diálogo.
    - ✓ El SO administra todas las ventanas asignándolas un identificador.
  - Eventos.
    - ✓ Acción que se ejecuta sobre el sistema.
    - ✓ El sistema operativo rastrea continuamente las ventanas en busca de sucesos.
  - Mensajes.
    - ✓ Cuando se produce un evento se envía un mensaje al sistema operativo.
    - ✓ El mensaje guarda información sobre el suceso y la ventana que lo ha producido.
    - ✓ El sistema operativo lo registra y almacena en una cola de mensajes.

Programación en Windows (III)

- Programación orientada a eventos
  - El entorno (sistema operativo, usuario, etc.) puede actuar sobre el programa en cualquier momento.
  - El programa debe responder a las acciones del entorno no proporcionadas de forma lineal.
  - No se debe prever un desarrollo lineal del flujo del programa.
    - ✓ Las distintas acciones se activan como respuesta a sucesos que ocurren en el entorno.
  - Al ejecutarse una aplicación basada en eventos
    - ✓ Windows rastrea las ventanas.
    - ✓ Si se detecta un evento en alguna ventana manda un mensaje al sistema operativo y lo almacena en la cola de mensajes
    - ✓ El sistema operativo lo procesa y lo transmite a las demás ventanas, indicando el evento y el identificador de la ventana que lo produce (`Handle`).
    - ✓ La aplicación busca el controlador de eventos asociado a ese evento en el control y, si existe, ejecuta el código correspondiente.

Programación en Windows (IV)



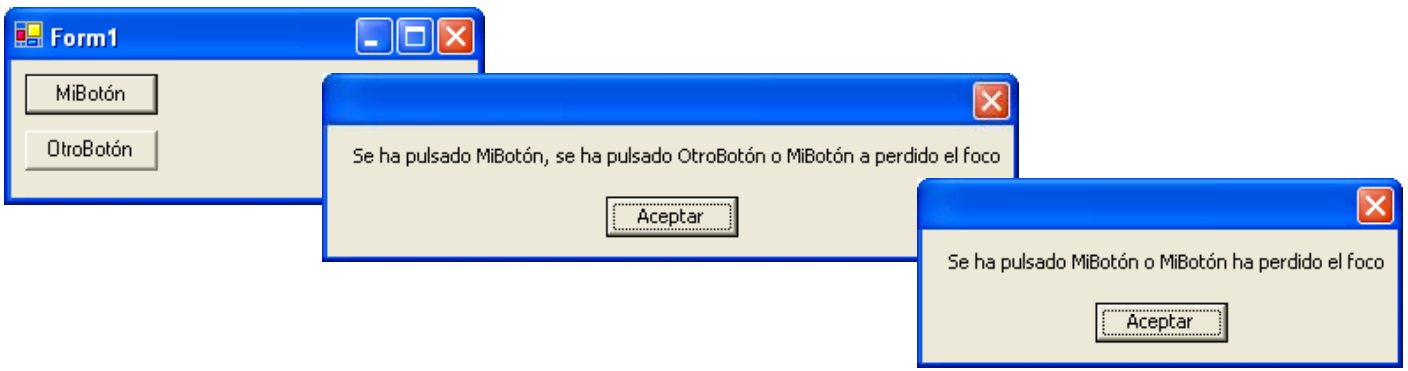

```
Private Sub MiBotón_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles MiBotón.Click
...
End Sub
```

Programación en Windows (V)

- Estructura de un procedimiento de evento.
  - Cada componente de Windows Forms tiene asociado una serie de eventos a los que responde.
    - ✓ Los controladores de eventos tienen dos argumentos:
      - `Sender`, de tipo `Object` y tiene una referencia al objeto que lo ha producido.
      - `e`, un objeto de la clase `EventArgs` o alguna de sus derivadas con información del evento.
    - ✓ El nombre corresponde con el nombre del control.
    - ✓ La cláusula `Handles` indica que métodos de eventos están asociados al procedimiento.

Programación en Windows (VI)

```
Private Sub MiEvento(ByVal sender As System.Object, _
                     ByVal e As System.EventArgs) _
                     Handles MiBotón.Click, MiBotón.Leave, OtroBotón.Click
   MessageBox.Show("Se ha pulsado MiBotón, se ha pulsado OtroBotón " & _
                   "o MiBotón a perdido el foco")
   If sender Is MiBotón Then
       MessageBox.Show("Se ha pulsado MiBotón o MiBotón ha perdido el foco")
   Else
       MessageBox.Show("Se ha pulsado OtroBotón")
   End If
End Sub
```



Programación en Windows (VII)

- La instrucción `AddHandler`, permite asociar un evento a un controlador de eventos determinado, distinto del procedimiento de evento predeterminado.
  - Esto permite activar y desactivar los eventos a voluntad.

AddHandler *NombreObjeto.NombreEvento, AddressOf NombreControladorDeEventos*

```
AddHandler MiBotón.Click, AddressOf MiEvento
AddHandler MiBotón.Leave, AddressOf MiEvento
AddHandler OtroBotón.Click, AddressOf MiEvento
…
Private Sub MiEvento(ByVal sender As Object, _
                ByVal e As EventArgs) 'No lleva claúsula Handles
        MessageBox.Show("Se ha pulsado MiBotón " & _
                        "u OtroBotón " & _
                        "o MiBotón ha perdido el foco")
        If sender Is MiBotón Then
            MessageBox.Show("Se ha pulsado MiBotón " & _
                            "o mi botón ha perdido el foco")
        Else
            MessageBox.Show("Se ha pulsado OtroBotón")
        End If
End Sub
```

- La instrucción `RemoveHandler`, permite desactivar un controlador de eventos.

RemoveHandler *NombreObjeto.NombreEvento, AddressOf NombreControladorDeEventos*

Aplicaciones Windows Forms

- Se desarrolla alrededor de uno o más formularios.
- Generación automática de código.
  - Visual Studio genera código en tres sitios distintos:
    - ✓ Archivo `Application.Designer.vb`.
      - Uno por proyecto
      - Está dentro del directorio `My Proyect` del proyecto.
      - Incluye las características generales de la aplicación y formulario o módulo de arranque.
    - ✓ Archivo `Form`*X*`.Designer.vb`.
      - Uno por formulario.
      - Dentro del directorio de proyecto.
      - Implementación parcial de la clase `Form`.
      - Incluye el código necesario para crear y destruir los controles que se incluyan en el formulario.
    - ✓ Archivo `Form`*X*`.vb`.
      - Clase `Form`*X* con la declaración del resto de la clase.
      - Incluye el código de usuario para manejar la aplicación

Tareas comunes: texto

- Propiedad `Text`.
  - Establece u obtiene el texto asociado al control.
  - Presente en todos los controles que tienen texto estático o editable.
  - En texto estático, el carácter `&` se utiliza para determinar la tecla de acceso.
- Propiedad `TextAlign`.
  - Alineación del texto
  - Presente en los controles `Label`, `TextBox`, `Button`, `CheckBox`, `RadioButton`, `NumericUpDown` y `DomainUpDown`.
  - Para los controles `Label`, `Button`, `CheckBox` y `RadioButton` puede tomar alguno de los valores de la enumeración `ContentAligment`.
    - ✓ `BottomCenter`, `BottomLeft`, `BottomRight`, `MiddleCenter`, `MiddleLeft`, `MiddleRight`, `TopCenter` , `TopLeft`, `TopRight`.
  - Para el resto puede tomar alguno de los valores de la enumeración `HorizontalAligment`.
    - ✓ `Center`, `Left`, `Right`.

Tareas comunes: color

- Propiedades `ForeColor` y `BackColor`.
  - Establece u obtienen el color de primer plano y el color de fondo.
  - Su valor es un dato de la estructura `System.Drawing.Color`.
- Miembros de la estructura `Color`.
  - Método estático `Color.FromArgb(`*`rojo`*`, `*`verde`*`, `*`azul`*`)`.
  - Método estático `Color.FromKnownColor(`*`nombreColorConocido`*`)`.
  - Método estático `Color.FromName(`*`cadena`*`)`.
  - Propiedades `R`, `G`, `B`.

```
MiBotón.BackColor = Color.FromArgb(0, 0, 255) 'Color de fondo azul
Me.BackColor = Color.FromKnownColor(KnownColor.Yellow) ‘Amarillo
OtroBotón.BackColor = Color.FromName("Green") 'Color de fondo verde
Dim c As System.Drawing.Color = MiBotón.BackColor
MessageBox.Show(c.R & "-" & c.G & "-" & c.B) 'Devuelve 0-0-255
```

Tareas comunes: fuentes

- Propiedad `Font`.
  - Hace referencia a un objeto `System.Drawing.Font`.
  - En tiempo de ejecución la modificación de las características de la fuente implica la creación de una nueva instancia de la clase.

```
'Para cambiar el estilo de la fuente a negrita
'MiBotón.Font.Bold = True no es válido
MiBotón.Font = New Font(MiBotón.Font, FontStyle.Bold)
```

  - Las fuentes de los componentes de un objeto contenedor, toman las características de los objetos contenidos.

Propiedades del objeto Font		
Propiedad	**Descripción**	**Valores**
Bold	Obtiene un valor que indica si el objeto Font está en negrita	True o False
Italic	Obtiene un valor que indica si el objeto Font está en cursiva	True o False
Name	Obtiene una representación del tipo de letra del objeto Font	Cadena
Size	Obtiene el tamaño del objeto Font	Real de simple precisión
Strikeout	Otiene un valor que indica si el objeto Font está tachado	True o False
Underline	Otiene un valor que indica si el objeto Font está subrayado	True o False
Unit	Obtiene la unidad de medida del objeto Font	Un miembro de GraphicsUnit (Inch, Millimeter, Point,…)

Tareas comunes: tamaño y posición

- Propiedad `Location`.
  - Hace referencia a una estructura de tipo `System.Drawing.Point` que identifica la posición de la esquina superior izquierda del componente.
  - Estructura `Point`.
    - ✓ Constructor: `Point(X,Y)`.
    - ✓ Propiedades `X` e `Y`.
  - Se puede establecer en tiempo de diseño (ventana de propiedades) o de ejecución.
    - ✓ Ejemplo:

```
'Pone el botón en la esquina superior izquierda del formulario
OtroBotón.Location = New Point(0,0)
```

Tareas comunes: tamaño y posición (II)

- Propiedad `Size`.
  - Hace referencia a una estructura `System.Drawing.Size`.
    - ✓ Constructor: `Size(ancho,alto)`.
    - ✓ Miembros `Width` y `Height`.

```
OtroBotón.Size = New Size(100, 50)
MiBotón.Size = OtroBotón.Size
```

- Método `SetBounds()`.
  - Establece la posición y el tamaño de un componente.

    control`.SetBounds(`*x*`,`*y*`,`*ancho*`,`*alto*`)`

```
'Iguala el tamaño de OtroBotón a MiBotón y lo coloca en la posición 0,0
OtroBotón.SetBounds(0, 0, MiBotón.Size.Width, MiBotón.Size.Height)
'Iguala el tamaño del formulario al de la pantalla
Me.SetBounds(0, 0, Screen.PrimaryScreen.WorkingArea.Width, _
             Screen.PrimaryScreen.WorkingArea.Height)
```

Tareas comunes: tamaño y posición (III)

- Propiedad `Bounds`.
  - Hace referencia a una estructura de tipo `System.Drawing.Rectangle`.
    - ✓ Propiedades `X`, `Y`, `Width` y `Height`.

```
OtroBotón.Bounds = MiBotón.Bounds 'Pone a OtroBotón encima de MiBotón
```

- Propiedad `ClientSize`.
  - Devuelve un objeto `Size` con el tamaño del área cliente del control.
- Propiedad `ClientRectangle`.
  - Devuelve un objeto `Rectangle` con el rectángulo del área cliente del control.

```
OtroBotón.Bounds=New Rectangle(0,0,Me.ClientSize.Width,Me.ClientSize.Height)
OtroBotón.Bounds = Me.ClientRectangle 'Hace lo mismo que lo anterior
```

Tareas comunes: tamaño y posición (IV)

Propiedad	Descripción	Valores
Location	Obtiene o establece el punto superior izquierdo del control	Una estructura Point
Size	Obtiene o establece el tamaño del control	Una estructura Size
Left, Top, Width, Height	Coordenadas individuales del control (obsoletas)	Un valor entero
Right	Coordenada X del borde derecho	Un valor entero
Bottom	Coordenada Y del borde inferior	Un valor entero
Bounds	Establece u obtiene el rectángulo que identifica la posición y el tamaño del control	Una estructura Rectangle
ClientRectangle	El rectángulo del área cliente del control	Una estructura Rectangle
ClientSize	Dimensiones del área cliente del control	Una estructura Size
Anchor	Distancia desde el borde del contenedor al control (ver tutorial del entorno)	Un miembro de la enumeración AnchorStyles
Dock	Establece que bordes del control se encuentran acoplados a su contenedor	Un miembro de la enumeración DockStyles
Método	**Descripción**	**Devuelve**
BringToFront	Trae el objeto a primer plano	
SendToBack	Lleva el objeto al fondo	
SetBounds(*X*,*Y*,*ancho*, *alto*)	Define el rectángulo que define la posición y tamaño del control	
SetSize(*ancho*, *alto*)	Define el tamaño que define un control	

Tareas comunes: manejo del teclado

- Eventos `KeyPress`, `KeyDown` y `KeyUp`.
  - Se ejecutan en el siguiente orden:
    - ✓ `KeyDown`
    - ✓ `KeyPress`
    - ✓ `KeyUp`.
- Evento `KeyPress`.

```
control_KeyPress(sender As Object, e As KeyPressEventArgs)
```

  - `sender` es una referencia al objeto que ha enviado el evento.
  - `e` es una referencia a un objeto de la clase `System.Windows.Forms.KeyPressEventArgs`.
    - ✓ Miembros de `KeyPressEventArgs`:
      - `KeyChar`, representa el carácter que se ha pulsado.
      - `Handled`, un valor lógico. Si se pone a `True`, indica que el evento se ha procesado y no hay que hacer nada más.

Tareas comunes: manejo del teclado (II)

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, _
                              ByVal e As System.Windows.Forms.KeyPressEventArgs) _
                              Handles TextBox1.KeyPress
   'Procesa sólo las teclas numéricas y las teclas de control
   If Not (Char.IsDigit(e.KeyChar) Or Char.IsControl(e.KeyChar)) Then
       'El motor ignora la tecla
       e.Handled = True
   End If
End Sub

Private Sub TextBox2_KeyPress(ByVal sender As Object, _
                              ByVal e As System.Windows.Forms.KeyPressEventArgs) _
                              Handles TextBox2.KeyPress
   'Convierte los caracteres alfabéticos a mayúsculas
   If Char.IsLetter(e.KeyChar) Then
       'La propiedad SelectedText devuelve el texto seleccionado
       'Puede ser una cadena nula si no hay seleccionado ningún texto
       'En ese caso será una cadena nula situada en la posición del cursor
        TextBox2.SelectedText = Char.ToUpper(e.KeyChar)
        e.Handled = True
   End If
End Sub
```

Tareas comunes: manejo del teclado (III)

- Eventos `KeyUp` y `KeyDown`.
  - Permiten detectar las teclas especiales mediante el argumento e de la clase `KeyEventArgs`.
    - ✓ Miembros de `KeyEventArgs`:
      - `Handled`.
      - `Alt`, `Control`, `Shift`.
      - `KeyCode`. Contiene el código de la tecla pulsada, un dato la enumeración `Keys` (`Keys.A`..`Keys.Z`, `Keys.D0`..`Keys.D9`, `Keys.F1`..`Keys.F2`, etc.).

```
Private Sub TextBox2_KeyDown(ByVal sender As Object, _
                             ByVal e As System.Windows.Forms.KeyEventArgs) _
                             Handles TextBox2.KeyDown
    'Detecta si se ha pulsado la tecla Shift+F1
    If e.Shift And e.KeyCode = Keys.F1 Then
        MsgBox("Se ha pulsado Shift+F1")
    End If
End Sub
```

Tareas comunes: manejo del ratón

- Pulsación de teclas:
  - Eventos `Click`, `DoubleClick`, `MouseUp`, `MouseDown` y `MouseWheel`.
- Movimiento del ratón.
  - Eventos `MouseMove`, `MouseEnter`, `MouseLeave` y `MouseHover`.
- Orden de procesamiento de eventos:
  1. `MouseEnter`.
  2. `MouseMove`.
  3. `MouseHover`/`MouseDown-Click-DoubleClick`/`MouseWheel`.
  4. `MouseUp`.
  5. `MouseLeave`.

Tareas comunes: manejo del ratón (II)

- `MouseMove`, `MouseDown`, `MouseWheel` y `MouseUp` reciben un argumento de la clase `MouseEventArgs`.
  - Miembros de `MouseEventArgs`.

Miembros de MouseEventArgs		
Propiedad	**Descripción**	**Valores**
Button	Obtiene el botón del ratón que se presionó.	Un miembro de la enumeración MouseButtons (Left, Middle, None, Rigth, XButton1 o XButton2)
Clicks	Obtiene el número de veces que el botón del ratón se presionó y se soltó.	Un entero con el número de veces que se pulsó y soltó el botón
Delta	Obtiene un recuento con signo que indica el número de pasos de trinquete que ha girado la rueda del ratón. Un paso de trinquete es una muesca de la rueda del ratón.	Entero
X	Obtiene la coordenada x del ratón.	Entero
Y	Obtiene la coordenada y del ratón.	Entero

Tareas comunes: control del foco de entrada

Propiedad	Descripción	Valores
Enabled	Obtiene o establece el estado de activado o desactivado del control	Lógico
TabStop	Determina si el control va a entrar en el orden de tabulación	Lógico
TabIndex	Determina el orden en que el control va a entrar en el orden de tabulación	Entero
Visible	Obtiene o establece si un control es visible	Lógico
CausesValidation	Determina si un control va a provocar un evento de validación	Lógico
CanFocus	Determina si un control puede tomar el foco de entrada (si Visible y Enabled están a True)	Lógico
Focused	Determina si un control tiene el foco	Lógico

Método	Descripción	Valores devueltos
Focus()	Da el foco a un control. Su uso es obligatorio en controles que no se pueden seleccionar (Panel, GroupBox, PictureBox, ProgressBar, Splitter, Label, LinkLabel cuando no hay ningún enlace).	Lógico (True si se ha podido seleccionar el control o false en caso contrario)
GetNextFocus(*control*, *adelante*)	Obtiene el siguiente o anterior control en el orden de tabulación (si *adelante* es True, obtiene el siguiente)	Control
Select()	Establece el foco en un control. No se puede utilizar en controles que no se pueden seleccionar. En el resto es igual a Focus().	Ninguno

Tareas comunes: control del foco de entrada (II)

- Cuando un control entra en foco se producen los siguientes eventos:
  1. Enter.
  2. GotFocus.
  3. Leave.
  4. Validating.
  5. Validated.

```
Private Sub TextBox3_Validating(ByVal sender As Object, _
                                ByVal e As System.ComponentModel.CancelEventArgs) _
                                Handles TextBox3.Validating
    'Sólo permite dejar el control si se introduce un valor numérico positivo
    If Not IsNumeric(TextBox3.Text) OrElse CInt(TextBox3.Text) <= 0 Then
        MessageBox.Show("Se debe introducir un valor numérico mayor a 0")
        TextBox3.Text = String.Empty
        e.Cancel = True
    End If
End Sub
```

La clase Form

- Representa una ventana o cuadro de diálogo de la aplicación.
- Desde el punto de vista de la interfaz, se utilizará cómo un contenedor de controles.
- Desde el punto de vista de la aplicación, será un objeto heredado de la clase `Form` y que constituye el punto de entrada de la aplicación.
  - Normalmente contendrá las declaraciones y el código de la aplicación.
- En el archivo `Formx.designer.vb`…

```
Partial Class Form1
    Inherits System.Windows.Forms.Form
   'Código generado por Visual Studio con las características del formulario
    ...
End Class
```

- En el archivo `Formx.vb` …

```
Public Class Form1
   'Código de usuario para manejar el formulario
    ...
End Class
```

La clase Form (II)

- Ciclo de vida de un formulario. Eventos que intervienen.
  1. Evento `Load()`.
     - Se produce cuando el formulario se carga por primera vez y antes de que se muestre.
     - Es el lugar adecuado para meter el código necesario para inicializar variables, abrir bases de datos, dar contenido a los controles, etc.
  2. Evento `Shown()`
     - Se produce la primera vez que se muestra.
  3. Evento `Activated()`.
     - Se produce cada vez que el formulario entra en foco, ya sea por una acción del usuario o por el código del programa.
     - Este evento se puede usar para la actualización del contenido con los cambios que pudieran haberse producido cuando no estaba activado.

La clase Form (III)

- Ciclo de vida de un formulario (*continuación*)
  4. Evento `Deactivate()`.
     - Se produce cuando el formulario pierde el foco.
     - Puede utilizarse para actualizar el contenido de otra ventana con los datos del formulario que ha perdido el foco.
  5. Evento `FormClosing()`.
     - Se produce cuando se da la orden de cerrar el formulario, pero antes de que se cierre.
     - Es posible cancelar la acción de cierre poniendo a True la propiedad `Cancel` del argumento `FormClosingEventArgs` del control.
  6. Evento `FormClosed()`.
     - Se produce después de haberse cerrado el formulario.
     - Se puede utilizar para liberar recursos utilizados por el formulario, almacenar la información producida por él o actualizar otro formulario.

Mover y cambiar el tamaño del formulario

- Propiedades `Size`, `Location`, `Bounds`.
- Propiedades `DesktopLocation` y `DesktopBounds`.
  - Establecen la posición (un objeto `Point`) y el tamaño (un objeto `Size`) a partir del área del escritorio no ocupada por la barra de tareas.
    - ✓ Realizan acciones distintas a `Location` y `Bounds` si la barra de tareas está no esta acoplada a la parte inferior.
- Métodos `SetDesktopLocation` y `SetDesktopBounds`.
  - `SetDesktopLocation(`*x*`,`*y*`)`
  - `SetDesktopBounds(`*x*`,`*y*`,`*ancho*`,`*alto*`)`

```
'Establece la posición y el tamaño de la pantalla activa
'Screen.PrimaryScreen hace referencia a la pantalla principal
'La propiedad WorkingArea devuelve el tamaño y posición de una pantalla
 Me.DesktopBounds = Screen.PrimaryScreen.WorkingArea
'Establece el tamaño del formulario a 1/4 del tamaño del escritorio
'y lo centra en el cuadrante inferior derecho del mismo
Me.SetDesktopBounds(Screen.PrimaryScreen.WorkingArea.Width / 2, _
                    Screen.PrimaryScreen.WorkingArea.Height / 2, _
                    Screen.PrimaryScreen.WorkingArea.Width / 2, _
                    Screen.PrimaryScreen.WorkingArea.Height / 2)
```

Mover y cambiar el tamaño del formulario (II)

- Métodos `CenterToScreen()` y `CenterToParent()`.
  - Centran el formulario en la pantalla y en el formulario padre (en el caso de que sea una aplicación MDI).
- Propiedad `TopMost`.
  - Asignando un valor `True`, el formulario siempre aparece por encima del resto.
- Propiedad `StartPosition`.
  - Establece la posición de inicio del formulario.

Miembros de StartPosition	Decripción
CenterParent	El formulario está centrado en los límites de su formulario principal.
CenterScreen	El formulario está centrado en la pantalla actual y tiene las dimensiones especificadas en el tamaño del formulario.
Manual	La posición del formulario viene determinado por la propiedad Location
WindowsDefaultBounds	El formulario se encuentra colocado en la ubicación predeterminada de Windows y tiene los límites establecidos por Windows de forma predeterminada.
WindowsDefaultLocation	El formulario se encuentra colocado en la ubicación predeterminada de Windows y tiene las dimensiones especificadas en el tamaño del formulario

Modificar el aspecto del formulario

- Propiedad `BackgroundImage`.
  - Establece la imagen de fondo del formulario.
- Propiedad `Icon`.
  - Establece el icono de la barra de títulos del formulario.
- Propiedades `ControlBox`, `MaximizeBox`, `MinimizeBox`, `HelpButton`.
  - Contienen un valor lógico que establece si el botón del menú de control, maximizar, minimizar o el botón de ayuda aparecen en el formulario.
- Propiedad `Opacity`.
  - Establece mediante un número real el nivel de transparencia de un formulario.
    - ✓ De forma predeterminada el nivel de transparencia es de 1,00.
- Propiedad `TransparencyKey`.
  - Establece el color que será transparente en el formulario.
    - ✓ `Me.TranparencyKey = Me.Backcolor 'Hace transparente el fondo del formulario.`

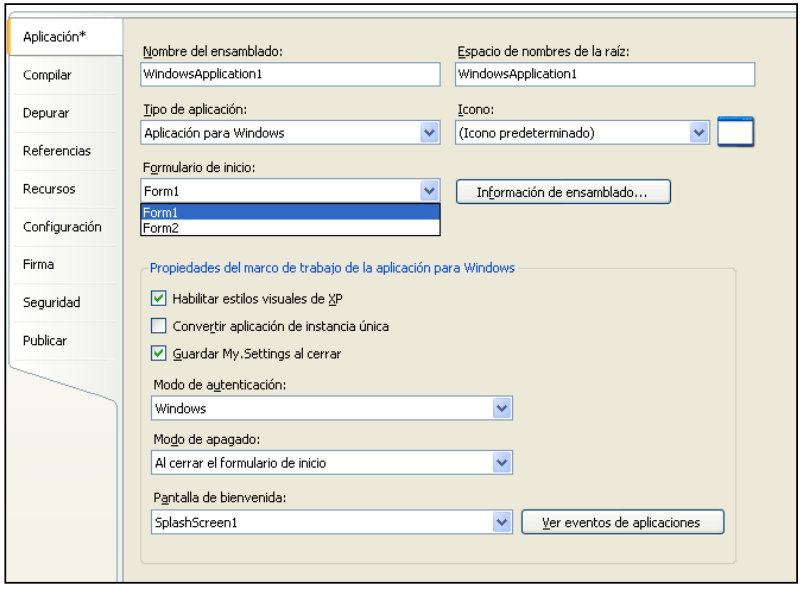
Modificar el aspecto del formulario (II)

- Propiedad `FormBorderStyle`.
  - Permite tomar alguno de estos valores:

Parámetro	Descripción
None	Ninguno (ningún borde ni elemento relacionado con él). Se utiliza para los formularios de inicio (por ejemplo, pantallas de *splash*).
Fixed 3D	Se utiliza cuando se desea un efecto de borde tridimensional. No se puede cambiar de tamaño. Puede incluir en la barra de título un botón de menú de control y botones Maximizar y Minimizar.
Fixed Dialog	Se utiliza para los cuadros de diálogo. Presenta un borde grueso. No se puede cambiar de tamaño. Puede incluir en la barra de título un cuadro de menú de control, y botones Maximizar y Minimizar.
Fixed Single.	No se puede cambiar de tamaño. Presenta un borde de una sóla línea. Puede incluir cuadro de menú de control y botones Maximizar y minimizar. Sólo puede cambiar de tamaño con los botones Maximizar y Minimizar.
Fixed Tool Window	Se utiliza para las ventanas de herramientas. Muestra una ventana de tamaño no ajustable con un botón Cerrar y texto de barra de título con un tamaño de fuente reducido. El formulario no aparece en la barra de herramientas de Windows.
Sizable	Con frecuencia se utiliza como ventana principal. Se le puede cambiar el tamaño. Puede incluir un menú de control y botones Maximizar y Minimizar. Puede cambiar de tamaño mediante el cuadro de menú de control, los botones Maximizar y Minimizar de la barra de título, o mediante el ratón.
SizableToolWindow	Ventana de herramientas de tamaño variable. Una ventana de herramientas no aparece en la barra de tareas ni en la ventana que aparece cuando el usuario presiona ALT+TAB.

Mostrar formularios

- Formulario de inicio.
  - Se selecciona en la página Aplicación del Diseñador de proyectos.
  - Dependiendo del tipo de aplicación se puede seleccionar:
    - ✓ Para aplicaciones de consola.
      - Sub Main de un módulo.
    - ✓ Para aplicaciones Windows.
      - Cualquiera de los formularios de la lista "Formulario de inicio"
    - ✓ Biblioteca de clases.
      - No existe un objeto inicial.
  - Se puede establecer en tiempo de ejecución mediante código en el método `Main()` mediante el método `Run` del objeto `Application`.




```
Sub Main()
    Dim frm As New Form1
    Application.Run(frm)
End Sub
```

Mostrar formularios (II)

- Mostrar formularios secundarios no modales.
  - Se debe crea una instancia del formulario y aplicar el método `Show()`.

```
'El proyecto incluye la clase Form3
Dim frm As New Form3
frm.Show()
```

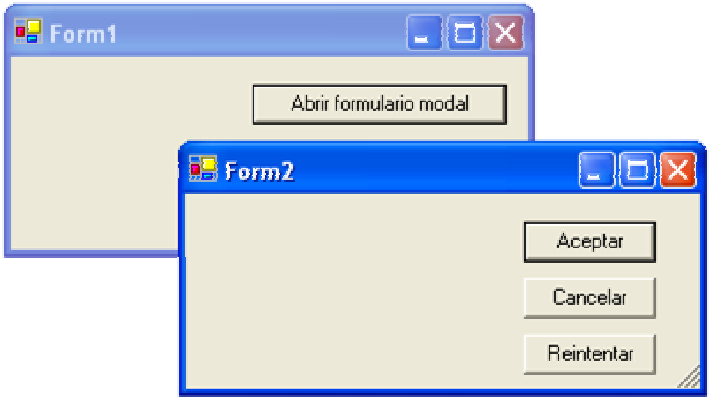
  - En el Visual Basic de .NET Framework 2.0, se puede acceder a instancias de los formularios a través del objeto My.Forms.

```
'El proyecto incluye la clase Form3
 My.Forms.Form2.Show()
```

- Mostrar formularios modales.
  - Se crea una instancia del formulario y se usa el método `ShowDialog()`.
  - El propietario será el formulario activo al hacer la llamada.
    - ✓ `ShowDialog()` puede pasar como argumento una referencia a otro formulario para cambiar el formulario propietario.
  - `ShowDialog()` devuelve un elemento del enumerado `DialogResult`.
    - ✓ La propiedad `DialogResult` del formulario permite especificar que valor del enumerado devuelve (p.e. `Me.DialogResult = DialogResult.Yes`).
      - Al asignar esta propiedad, el formulario se cierra.

Mostrar formularios (III)

- Los formularios modales y no modales tienen distinto comportamiento:
  - En los no modales, al abrir con el método `Show`, continúa el evento que ha realizado la llamada.
  - En los modales, al abrir con el método `ShowDialog`, el evento que ha realizado la llamada se detiene hasta que se cierra el formulario modal.




```
'En Form1…
Private Sub Button1_Click(…)…
  Dim frm As New Form2
  Dim r As DialogResult = frm.ShowDialog()
  Select Case r
    Case DialogResult.OK
      'Acciones cuando se pulsa Aceptar
    Case DialogResult.Cancel
      'Acciones cuando se pulsa Cancelar
    Case DialogResult.Retry
      'Acciones cuando se pulsa Reintentar
   End Select
End Sub
```

```
'En el botón Aceptar de Form2
Private Sub Button1_Click
   'Esto también cierra el formulario
       Me.DialogResult = DialogResult.OK
End Sub
```

Compartir información entre formularios

- Con formularios modales.
  - Se puede acceder a los controles de un formulario modal desde el formulario que lo llama.

```
Dim frm As New Form2
frm.ShowDialog()
'Accede al contenido de TextBox1 en Form2
MessageBox.Show(frm.TextBox1.Text)
```

  - Se puede acceder a las variables públicas del formulario modal desde el formulario que lo llama.

```
'En Form2
Public a As Integer = 10
…
'En Form1
Dim frm As New Form2
frm.ShowDialog()
'Accede al contenido de la variable a de form2
MessageBox.Show("A = " & frm.a)
```

Compartir información entre formularios (II)

- Con formularios no modales o en las ventanas secundarias
  - Se pueden utilizar variables globales en la ventana principal o en un módulo de código.
    - ✓ Cómo no tenemos una referencia a la instancia donde está declarada la variable, hay que hacer que la variable sea compartida.

```
'En Form1
Public Shared otraVariable As Integer = 100
…
'En Form2
MessageBox.Show(Form1.otraVariable)
```

  - También se pueden poner las variables en un módulo de código.

```
'En Module1
Public MásVariables as Integer = 200
…
'En Form1 o en Form2
MessageBox.Show(MásVariables)
```

Compartir información entre formularios (III)

- Acceder a la información de un formulario por medio de `My.Forms`.
  - `My.Forms` proporciona una instancia de cada formulario en el proyecto actual.
    - ✓ Para acceder a cada formulario, el nombre de la propiedad que hay que llamar es igual que el nombre de la clase que forma el formulario.
      - `My.Forms.Form1.Show()`
    - ✓ La primera vez que se accede a un formulario con `My.Forms`, se crea la instancia del mismo. Las veces siguientes, se accederá a la instancia creada anteriormente.
  - Sólo proporciona acceso a los formularios en aplicaciones Windows Forms, no en aplicaciones de consola o en formularios contenidos en DLL.
  - A partir de la instancia proporcionada es posible acceder a todos los miembros del formulario.
  - Para acceder a todos los formularios abiertos de una aplicación en un momento dado se puede utilizar la propiedad. `My.Application.OpenForms`, que devuelve una colección con todos los formularios de la aplicación.

```
'Escribe en una etiqueta, el título de todos los formularios abiertos
For Each frm As Form In My.Application.OpenForms
   Label1.Text = Label1.Text & " " & frm.Text
Next
```

Compartir información entre formularios (IV)

- Ejemplo: intercambiar información entre dos formularios con `My.Forms`.

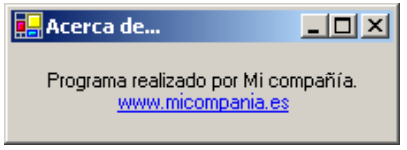

```
Public Class Form1
    'Cada vez que se pulsa el botón, el contenido del textbox pasa a form2
    Private Sub Button1_Click(ByVal sender As System.Object, _
                              ByVal e As System.EventArgs) _
                              Handles Button1.Click
        My.Forms.Form2.TextBox1.Text = TextBox1.Text
        My.Forms.Form2.Show()
    End Sub
End Class

Public Class Form2
    'Al cerrar Form2, el contenido del TextBox para a Form1
    Private Sub Form2_FormClosing(ByVal sender As Object, _
                                  ByVal e As _
                                  System.Windows.Forms.FormClosingEventArgs) _
                                  Handles Me.FormClosing
        My.Forms.Form1.TextBox1.Text = TextBox1.Text
    End Sub
End Class
```

Clase Label

- Representa un campo de texto estático.
  - Propiedad Text.
    - ✓ Obtiene o establece el texto de la etiqueta en modo de diseño o ejecución.
    - ✓ Propiedad `TextAlign`.
      - Alineación del texto. Puede contener alguno de los valores del enumerado `ContentAlign` (`BottomCenter`, `BottomLeft`, `BottomRight`, `MiddleCenter`, `MiddleLeft`, `MiddleRight`, `TopCenter`, `TopLeft`, `TopRight`).
  - Aspecto, tamaño, posición.
    - ✓ Propiedades `Location`, `Size`, `Backcolor`, `Forecolor`, `Font`.
    - ✓ Propiedad `BorderStyle`.
      - Puede tomar alguno de los valores de la enumeración `BorderStyle`: `Fixed3D`, `FixedSingle` o `None`.
    - ✓ Propiedad `Autosize`.
      - Un valor lógico `True` hace que el tamaño de la etiqueta se adecue al tamaño del texto.
        - True es el valor por omisión.
    - ✓ Propiedad `Autoelipsis`.
      - Si esta a `True`, visualiza puntos suspensivos si el texto de la etiqueta se extiende más allá de su longitud.
    - ✓ Propiedad `Image`.
      - Permite establecer una imagen en la etiqueta. La propiedad `ImageAlign` permite establecer la alineación de dicha imagen.

Clase LinkLabel

- Hereda de `Label`.
  - Permite establecer un enlace en la etiqueta.
  - Propiedad `LinkArea`.
    - ✓ Establece el área de la imagen.
    - ✓ Se le debe asignar un objeto de la clase `LinkArea`.
      - Constructor de `LinkArea`: `LinkArea(inicio,longitud)`.
  - Evento `LinkClicked`.
    - ✓ Se produce cuando se pulsa sobre el enlace.




```
LinkLabel1.Text = "Programa realizado por Mi compañía. www.micompania.es"
LinkLabel1.LinkArea = New LinkArea(35, 53)
…
Private Sub LinkLabel1_LinkClicked(ByVal sender As System.Object, _
  ByVal e As System.Windows.Forms.LinkLabelLinkClickedEventArgs)_
            Handles LinkLabel1.LinkClicked
    'La orden Shell permite ejecutar un archivo del sistema
    Shell("explorer http://www.micompania.es")
End Sub
```

Clase TextBox

- Representa un campo de texto editable por el usuario.
  - Propiedad `Text`.
    - ✓ Permite obtener o establecer el texto del control.
  - Propiedad `TextAlign`.
    - ✓ Permite establecer la alineación del texto. Su contenido es un miembro del enumerado `HorizontalAligment` (`Right`, `Center`, `Left`).
  - Cuadros multilínea.
    - ✓ La propiedad `Multiline` permite cuadros multilínea.
    - ✓ La propiedad `WordWrap` permite el salto de línea automático.
    - ✓ La propiedad `ScrollBars`, añade barras de desplazamiento al cuadro de texto multilinea.
    - ✓ Todas las líneas se almacenan en el array de cadenas `Lines`.

Clase TextBox (II)

- Modificar el contenido.
  - Cada vez que se modifica el texto del control se produce el evento `TextChanged()`.
  - La propiedad `Modified` se pone a `True` si el cuadro se ha modificado.
  - Propiedad `CharacterCasing`, permite convertir el texto en a mayúsculas (`Upper`) o minúsculas (`Lower`).
  - Propiedad `PasswordChar`.
    - ✓ Todo el texto escrito se visualiza como el carácter asignado a esa propiedad.
  - Propiedad `UseSystemPasswordChar`.
    - ✓ Todo el texto escrito se visualiza como el carácter que el sistema utiliza como carácter de contraseña.
  - Método `AppendText(cadena)`.
    - ✓ Añade la cadena al final del cuadro de texto.
  - Propiedad `ReadOnly`.
    - ✓ Impide modificar el cuadro de texto.

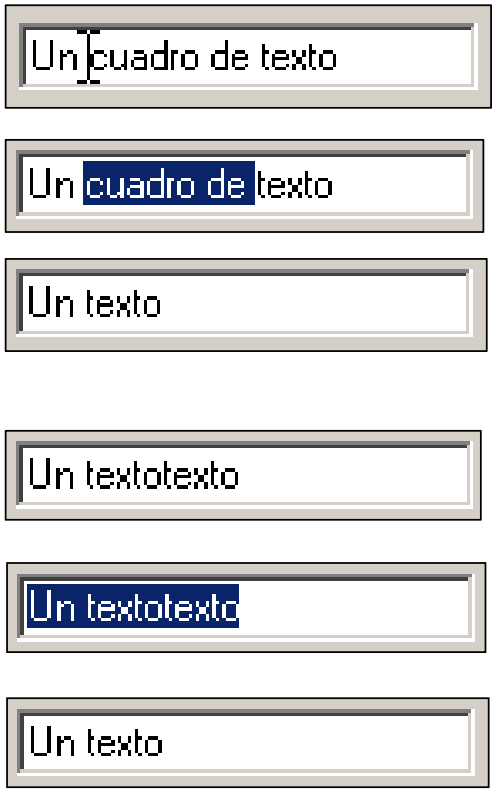
Clase TextBox (III)

- Autocompletar el contenido de un `TextBox`.
  - La propiedad `AutoCompleteMode` permite indicar si queremos que se autocomplete el contenido de los escrito en un TextBox:
    - ✓ `None`, no se autocompleta.
    - ✓ `Append`, al teclear los primeros caracteres añaden los que faltan.
    - ✓ `Sugest`, despliega una lista con las posibles opciones a completar.
    - ✓ `Sugest Append`, añade los que faltan y despliega la lista.
  - La propiedad `AutoCompleteSource`, indica el origen de los datos a autocompletar.
    - ✓ `FileSystem` Especifica el sistema de archivos como origen.
    - ✓ `HistoryList` Incluye los URL en la lista de historial.
    - ✓ `RecentlyUsedList` Incluye los URL de la lista de las direcciones usadas recientemente.
    - ✓ `AllUrl` Especifica el equivalente de `HistoryList` y `RecentlyUsedList` como el origen.
    - ✓ `AllSystemSources` Especifica el equivalente de `FileSystem` y `AllUrl` como el origen.
    - ✓ `FileSystemDirectories` Especifica que sólo los nombres de directorio y no los nombres de archivo se finalizarán automáticamente.
    - ✓ `CustomSource` Especifica que se utilizarán las cadenas que formen la propiedad `AutoCompleteCustomSource`.

Clase TextBox (IV)

- Control del punto de inserción y del texto seleccionado.
  - La propiedad `SelectionStart` permite obtener o establecer el punto de inserción.
    - ✓ Un valor 0 indica que el punto de inserción está antes del primer carácter.
  - La propiedad `SelectionLength` permite obtener o establecer la longitud del texto seleccionado.
    - ✓ Un valor igual a 0 indica que no hay seleccionado ningún carácter.
  - La propiedad `SelectedText` obtiene o establece el texto seleccionado.
    - ✓ Una cadena nula elimina indica que no hay texto seleccionado o elimina el mismo.
  - Los métodos `Cut()`, `Copy()` o `Paste()`, permiten cortar, copiar o pegar el texto.
  - El método `SelectAll()` selecciona todo el contenido del cuadro de texto.
  - El método `Select(`*`inicio`*`,`*`longitud`*`)` permite seleccionar una porción de texto.

Clase TextBox (V)




```
TextBox1.SelectionStart = 3


TextBox1.SelectionLength = 10


TextBox1.SelectedText = ""


TextBox1.SelectionLength = 5
TextBox1.Copy()
TextBox1.SelectionStart = TextBox1.TextLength
TextBox1.Paste()

TextBox1.SelectAll()


TextBox1.Select(3, 5)
TextBox1.Cut()
```

Clase Button

- Desciende de la clase `System.Windows.Forms.ButtonBase`, de la que también descienden la clase `RadioButton` y `CheckBox`.
- La propiedad `Text`, establece la etiqueta del control y permite asignar una tecla de acceso.
- Apariencia del botón.
  - La propiedad `Image`, permite establecer una imagen para el control.
    - ✓ La propiedad `ImageAlign`, permite establecer su posición en el control y puede tomar alguno de los valores de `ContentAlignment` (véase página 12).
  - La propiedad `BackgroundImage` permite repetir una imagen a lo largo del área que ocupa el control.
- Botones por omisión.
  - Las propiedades `AcceptButton` y `CancelButton` **del formulario** permiten especificar que botón se ejecutará al pulsar ENTER o al pulsar ESC.
  - La propiedad `DialogResult`, permite especificar que valor devolverá el botón al ser pulsado en un cuadro de diálogo modal.
    - ✓ Hace lo mismo que establecer la propiedad `DialogResult` del formulario.

Clase PictureBox

- Actúa como contenedor de imágenes de mapas de bits.
- La imagen se guarda en un objeto de la clase `Image` accesible mediante la propiedad `Image` de la clase.
  - El método `FromFile(`*`espArchivo`*`)` devuelve un objeto de la clase `Image` a partir de un archivo `.bmp`, `.jpg`, `.ico`, `.gif`, `.wmf` o `.png` contenido en disco.

    ```
    PictureBox1.Image = FromFile("C:\imágenes\MiImagen.jpg")
    ```

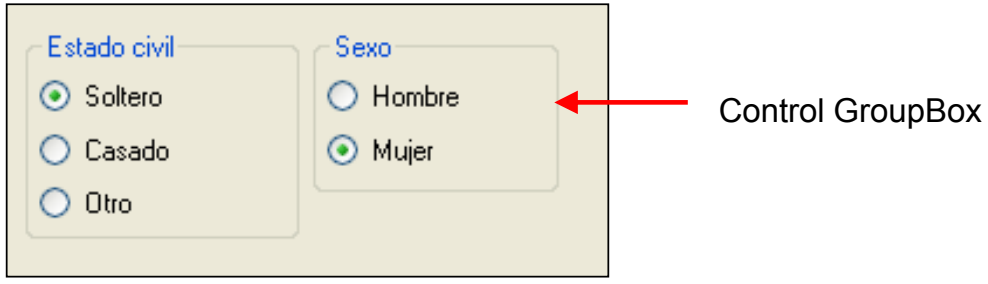
  - Para eliminar dicho contenido hay que asignar a la propiedad `Image` de PictureBox el literal `Nothing`.

    ```
    PictureBox1.Image = Nothing
    ```

- Tamaño y posición de la imagen en el control
  - La propiedad `SizeMode` permite acomodar la imagen en el control. Puede tomar alguno de los siguientes valores:
    - ✓ `Autosize`. El tamaño del control se adecua al de la imagen.
    - ✓ `CenterImage`. La imagen se centra en el control, recortándola si es necesario.
    - ✓ `Normal`. La imagen se sitúa en la esquina superior izquierda del control, recortándola si es necesario.
    - ✓ `StretchImage`. La imagen se adapta al tamaño del control.

Clase RadioButton

- Representa un grupo de opciones excluyentes.
  - Se pueden crear grupos independientes siempre que aparezcan en contenedores distintos.
    - ✓ Los controles contenedores son el objeto `Form`, el objeto `Panel` y el objeto `GroupBox`.



- La propiedad `Checked` devuelve un valor lógico dependiendo del estado del control.
- Eventos del control.
  - Además del evento `Click` presente en casi todos los controles, es posible controlar el evento `CheckedChanged` que se produce cuanto la propiedad `Checked` cambia de estado.

Clase RadioButton (II)

- Aspecto del control.
  - Propiedad `Image`.
  - Las propiedades `TextAlign`, `ImageAlign` y `CheckAlign` pueden tomar algún valor del enumerado `ContentAlignment`.
  - Propiedad `FlatStyle`.
    - ✓ Puede tomar alguno de los valores del enumerado `FlatStyle`:
      - `Standard`. Aspecto tridimensional.
      - `Flat`. Aspecto plano.
      - `Popup`. Aspecto plano hasta que el cursor pasa por encima.
      - `System`. Toma el aspecto predeterminado del sistema.
  - Propiedad `Appareance`.
    - ✓ `Normal`. Toma el aspecto normal.
    - ✓ `Button`. Toma el aspecto de botón.

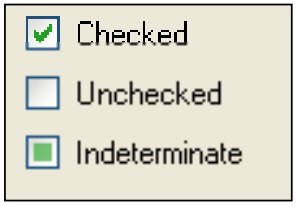
Control RadioButton (III)

- Manejo del control.
  - Cambiar el tratamiento de la persona (Don o Doña) según esté marcado el `RadioButton Hombre` o el `RadioButton Mujer`.

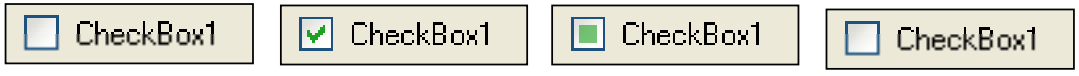
```
Public tratamiento As String
...
Private Sub Sexo_CheckedChanged(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles radHombre.CheckedChanged, _
                                        radMujer.CheckedChanged
    If radHombre.Checked Then
        Tratamiento = "Don"
    Else
        Tratamiento = "Doña"
    End If
End Sub
```

Clase CheckBox

- Representa un grupo de opciones no excluyentes.
  - El control puede devolver de forma predeterminada tres estados distintos: `Checked`, `Unchecked` o `Indeterminate`.



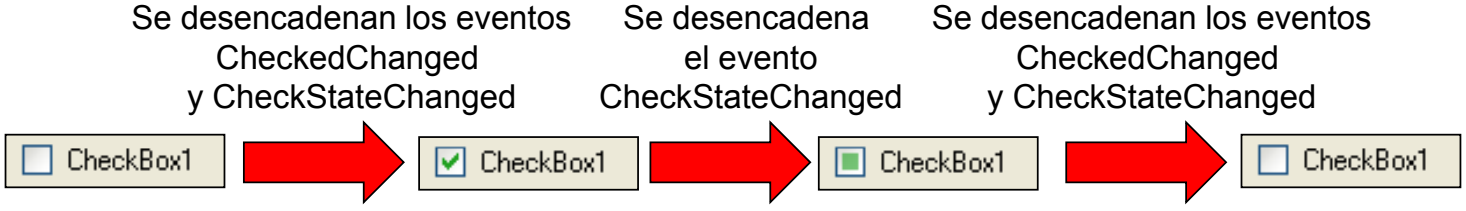
  - La propiedad lógica `ThreeState` permite determinar si el control va a tener esos tres estados o únicamente `Checked` y `Unchecked`.
  - La propiedad `Autocheck` permite pasar automáticamente de un estado a otro.
    - ✓ Con las propiedades `ThreeState` y `AutoCheck` a `True`…



Estado inicial | Primer Click | Segundo Click | Tercer Click

Clase CheckBox (II)

- El estado del control se puede obtener o establecer a través de las propiedades:
  - `Checked`, devuelve un valor lógico según esté o no marcado el control.
    - ✓ Devuelve `True` cuando el estado del control es `Checked` o `Indeterminate`.
  - `CheckState`, devuelve un valor del enumerado `CheckState`:
    - ✓ `CheckState.Checked`, `CheckState.Indeterminate` y `CheckState.Unchecked`.
- Eventos (se desencadenan en este orden):
  - `CheckedChanged`, se desencadena cuando cambia el estado de la propiedad `Checked`.
  - `CheckStateChanged`, se desencadena cuando cambia el valor de la propiedad `CheckState`.
  - `Click`, se desencadena cuando el usuario pulsa sobre el control.



Clase TrackBar

- Proporciona una barra de seguimiento (control deslizante).
  - Se utilizará para asignar de forma gráfica valores numéricos continuos.
    - ✓ Por ejemplo controles de volumen u otros valores analógicos.
- Propiedad `Value`.
  - Proporciona un valor entero representado por el cuadro de desplazamiento de la barra.
- Propiedades `Minimum` y `Maximum`.
  - Valores máximo y mínimo permitidos en la barra.
- Propiedad `LargeChange`.
  - Representa el incremento o decremento que se produce en la propiedad `Value` cuando utilizan las teclas AvPág o RePág.
- Propiedad `SmallChange`.
  - Representa el incremento o decremento que se produce en la propiedad `Value` cuando se utilizan las teclas del cursor.

Clase TrackBar (II)

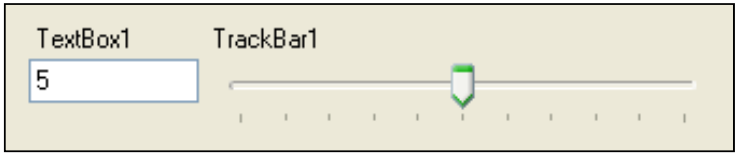
- Propiedad `Orientation`.
  - Permite definir la orientación (`Horizontal` o `Vertical`) del control.
- `TickFrecuency`.
  - Un entero que permite establecer la distancia entre las marcas del control.
- `TickStyle`.
  - Un miembro del enumerado `TickStyle` que especifica que marcas aparecerán en el control.
    - ✓ `BottomRight` (valor predeterminado). Las marcas aparecen abajo o a la derecha según la orientación del control.
    - ✓ `None`. No aparecen las marcas.
    - ✓ `Both`. Las marcas aparecen a ambos lados.
    - ✓ `TopLeft`. Las marcas aparecen arriba o a la izquierda.

Clase TrackBar (III)

- Eventos.
  - Evento `Scroll`.
    - ✓ Se produce cuando se desplaza el cuadro de desplazamiento mediante el ratón o el teclado.
  - Evento `ValueChanged`.
    - ✓ Se produce cuando cambia la propiedad Value, ya sea por código o por una acción del usuario.

Clase TrackBar (IV)

- Enlazar un control `TrackBar` a un cuadro de texto.
  - La información gráfica de los valores que proporciona el control se debe acompañar de una referencia numérica.




```
Private Sub TrackBar1_Scroll(ByVal sender As System.Object, _
                             ByVal e As System.EventArgs) _
                             Handles TrackBar1.Scroll
        TextBox1.Text = TrackBar1.Value
End Sub
```

Clase TrackBar (V)

- Enlazar un cuadro de texto a un control `ScrollBar`.

```
Private Sub TextBox1_TextChanged(ByVal sender As System.Object, _
                                 ByVal e As System.EventArgs) _
                                 Handles TextBox1.TextChanged
    'Sólo se modifica el valor del trackbar si el cuadro de texto
    'tiene un valor numérico
    If IsNumeric(TextBox1.Text) Then
        If TextBox1.Text < TrackBar1.Minimum Then
            'Si el valor es menor que el mínimo, se iguala al mínimo
            TextBox1.Text = TrackBar1.Minimum
        ElseIf TextBox1.Text > TrackBar1.Maximum Then
            'Si el valor es mayor que el máximo se iguala al máximo
            TextBox1.Text = TrackBar1.Maximum
        End If
        'Una vez que se tiene un valor correcto, se cambia
        'la propiedad value del Trackbar
        TrackBar1.Value = TextBox1.Text
    End If
End Sub
```

Clases HScrollBar y VScrollBar

- Proporcionan barras de desplazamiento horizontal y vertical.
  - Se pueden utilizar para proporcionar desplazamiento en controles que no las incluyan o para asignar de forma gráfica valores numéricos.
- Existen dos controles:
  - HScrollBar, barra de desplazamiento horizontal
  - VScrollBar, barra de desplazamiento vertical
- Las propiedades y el modo de manejo es similar al control TrackBar.
- No se recomienda su uso para no confundir el control con las barras de desplazamiento incluidas en algunos controles.
  - Es preferible utilizar el control TrackBar.

Clase NumericUpDown

- Proporciona un mecanismo para introducir valores numéricos.
  - Está formado por un cuatro de texto y dos flechas.

- La propiedad `Value` establece o devuelve el valor del cuadro de texto asociado.
  - Propiedad `Maximum` y `Minimum`.
  - Propiedad `Increment`. Establece o devuelve el incremento o decremento al pulsar cualquiera de las flechas.
  - Propiedad `ReadOnly`. Un valor True impide al usuario modificar los valores del cuadro de texto.
- Aspecto del control.
  - Propiedades `TextAlign` (se puede alinear a la izquierda derecha o centro) y `UpDownAlign` (se puede alinear a la izquierda o a la derecha).
  - Propiedad `DecimalPlaces`.
  - Propiedad `Hexadecimal`.
- Métodos `UpButton` y `DownButton`.
  - Realizan el mismo efecto que pulsar las teclas arriba o abajo.
- El cambio de valor del control se intercepta mediante el evento `ValueChanged`.

Clase DomainUpDown

- Tiene una funcionalidad y un aspecto similar a la clase `NumericUpDown`.
  - Permite establecer una cadena de texto a partir de una serie de elementos.
- Los elementos del control se guardan en la propiedad `Items`.
  - Se pueden insertar en tiempo de diseño en la ventana de propiedades.
  - Se pueden insertar en tiempo de ejecución mediante el método `Add` de la colección `Items`.
  - Se pueden eliminar en tiempo de ejecución mediante la propiedad `Remove`.
  - La propiedad `SelectedItem` devuelve el valor del elemento seleccionado del control.
  - La propiedad `SelectedIndex` devuelve o establece el índice del elemento seleccionado del control.
- Propiedad `Wrap`.
  - Un valor `True`, permite realizar un ciclo por dichos elementos (del último de la lista pasará al primero).
- El evento `SelectedItemChanged` se produce cuando cambia el valor de la propiedad `SelectedItem`.
  - Si el valor se cambia mediante el cuadro de texto se produce el evento `TextChanged`.
    - ✓ También cambia la propiedad `SelectedItem` (y se desencadena el evento `SelectedItemChanged`) a -1.

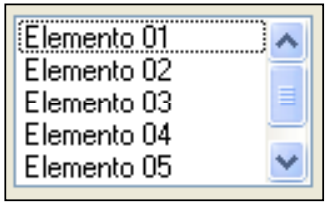
La clase ListBox

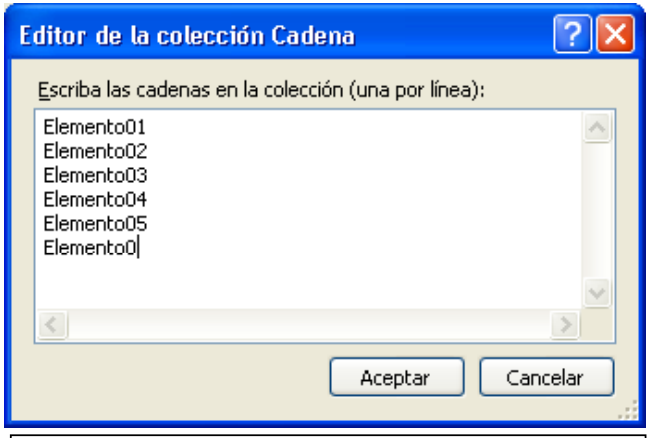
- Muestra una serie de elementos de los que el usuario puede seleccionar uno o más.
- Los elementos incluidos en el control se guardan en la propiedad `Items`.
  - `Items` es una colección del tipo `ListBox.ObjectCollection` que puede incluir cualquier tipo de objeto utilizado en .NET.
- Los elementos seleccionados se guardan en la propiedad `SelectedItems`, una colección del tipo `ListBox.SelectedObjectCollection`.
- Los índices de los elementos seleccionados se guardan en la propiedad `SelectedIndices`, una colección del tipo `ListBox.SelectedIndexCollection`.

Clase ListBox (II)

- La colección `Items`.
  - Representa a los objetos incluidos en la lista.
  - Agregar elementos a la colección.
    - ✓ Se pueden agregar en tiempo de diseño mediante el editor de la propiedad.
      - En tiempo de diseño sólo es posible añadir cadenas.
    - ✓ En tiempo de ejecución se pueden agregar mediante el método `Add`.

      `ObjetoListBox``.Items.Add(`*`objeto`*`)`

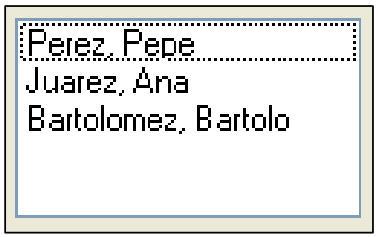





```
ListBox1.Items.Add("Elemento 01")
ListBox1.Items.Add("Elemento 02")
ListBox1.Items.Add("Elemento 03")
ListBox1.Items.Add("Elemento 04")
ListBox1.Items.Add("Elemento 05")
ListBox1.Items.Add("Elemento 06")
```

Clase ListBox (III)

- La colección `Items`.
  - Agregar elementos a la colección *(continuación)*.
    - ✓ Mediante el método `Add` es posible añadir cualquier tipo de objetos.




```
Structure persona
 Dim id As Integer
 Dim nombre As String
 Dim apellidos As String
 Sub New(ByVal id As Integer, ByVal ape As String, ByVal nom As
    String)
     Me.id = id
     nombre = nom
     apellidos = ape
 End Sub
 'El método toString permite convertir un objeto en una cadena
 Overrides Function toString() As String
     Return apellidos & ", " & nombre
 End Function
End Structure
...
lstPersonas.Items.Add(New persona(123, "Perez", "Pepe"))
lstPersonas.Items.Add(New persona(323, "Juarez", "Ana"))
lstPersonas.Items.Add(New persona(333, "Bartolomez", "Bartolo"))
```

Clase ListBox (IV)

- La colección `Items`.
  - Agregar elementos a la colección *(continuación)*.
    - ✓ El método `Insert` permite añadir un elemento en una posición específica mayor o igual que 0 y menor o igual que el número de elementos.

      ObjetoListBox`.Items.Insert(`*índice*`,`*objeto*`)`

      `Listbox1.Items.Insert(3,"Nuevo elemento")`
    - ✓ Se puede insertar cualquier objeto de cualquier tipo en la lista,

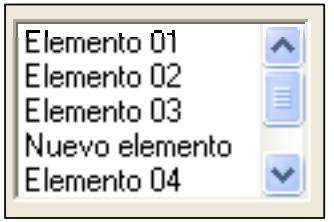
      ```
      lstPersonas.Items.Insert(0, New Persona(456,
        "Estévez", "Esteban"))
      ```
  - Eliminar elementos de la colección.
    - ✓ El método `Remove` permite eliminar un elemento de la colección a partir de su valor.

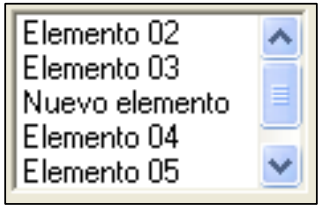
      ObjetoListBox`.Items.Remove(`*objeto*`)`

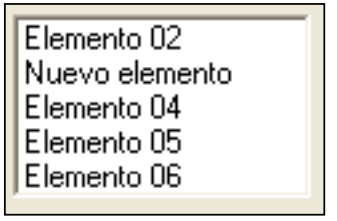
      `ListBox1.Items.Remove("Elemento 01")`
    - ✓ El método `RemoveAt` permite eliminar un elemento de la colección a partir de su índice.

      ObjetoListBox`.Items.Remove(`*índice*`)`

      `ListBox1.Items.RemoveAt(1)`
    - ✓ El método `Clear` permite eliminar todos los elementos de la colección.







Clase ListBox (V)

- Eliminar objetos de la lista…
  - Se puede pasar como argumento una referencia a un objeto de la lista.

```
'p es una referencia a un elemento de la lista
'no un dato de tipo persona cualquiera
'No valdría si Dim p as persona = new Persona(323, "Juarez", "Ana")
Dim p As Persona = lstPersonas.Items(1)
lstPersonas.Items.Remove(p) 'Elimina el segundo elemento de la lista
```

  - Si queremos buscar y borrar un elemento concreto habrá que realizar una búsqueda.

```
Dim p As persona = New persona(323, "Juarez", "Ana")
'Elimina el objeto p (Ana Juarez) de la lista
'a partir de la búsqueda de su identificador
'Es necesario codificar la función Equal en la estructura Persona
For Each obj As persona In lstPersonas.Items
   If obj.Equals(p) Then
       lstPersonas.Items.Remove(obj)
       Exit For
   End If
Next
...
'En la estructura Persona
Overloads Function Equals(ByVal o As persona) As Boolean
    Return o.id = id
End Function
```

Clase ListBox (VI)

- La colección `Items`.
  - La propiedad `Count` devuelve el número de elementos de la colección.
  - Buscar elementos en la colección.
    - ✓ La propiedad `Contains` devuelve un valor lógico `True` si el elemento que se pasa como argumento está incluido en la colección.
      - *`ObjetoListBox`*`.Items.Contains(`*`objeto`*`)`
    - ✓ La propiedad `IndexOf` devuelve el índice del objeto que se pasa como argumento.
      - Devuelve -1 si el objeto no se encuentra.

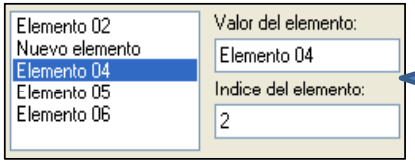
      `ObjetoListBox``.Items.IndexOf(`*`objeto`*`)`
        - En el código de la página anterior se podría haber puesto…

        ```
        lstPersonas.Items.RemoveAt(lstPersonas.Items.IndexOf(obj))
        ```

Clase ListBox (VII)

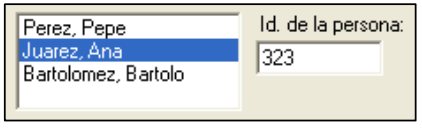
- Trabajar con elementos de la lista.
  - La propiedad `SelectedIndex` devuelve el índice del elemento seleccionado de la lista.
    - ✓ Devuelve -1 si no se ha seleccionado ninguno.
  - La propiedad `SelectedItem` devuelve el elemento seleccionado de la lista.
    - ✓ Devuelve el literal `Nothing` si no se ha seleccionado ninguno.
  - La propiedad `Text` devuelve el contenido del elemento seleccionado convertido a cadena.
- Eventos.
  - Eventos `Click` y `DoubleClick`.
  - Evento `SelectedItemChanged`.
    - ✓ Se produce cuando cambia el valor de la propiedad `SelectedItem`.
  - Evento `SelectedIndexChanged`.
    - ✓ Se produce cuando cambia el valor de la propiedad `SelectedIndex`.

Clase ListBox (VIII)



Al seleccionar un elemento, aparece su contenido y su posición

```
Private Sub ListBox1_SelectedIndexChanged(ByVal sender As System.Object, _
                                          ByVal e As System.EventArgs) _
                                         Handles ListBox1.SelectedIndexChanged
    TextBox1.Text = (ListBox1.SelectedItem)
    TextBox2.Text = (ListBox1.SelectedIndex)
End Sub
```



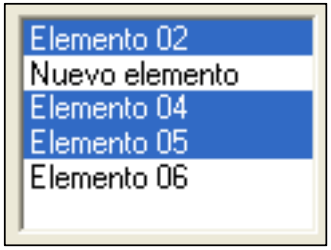
Al seleccionar una persona, aparece su identificador

```
Private Sub lstPersonas_SelectedIndexChanged(ByVal sender As System.Object, _
                                             ByVal e As System.EventArgs) _
                                            Handles lstPersonas.SelectedIndexChanged
    Dim p As persona = lstPersonas.SelectedItem
    TextBox3.Text = p.id
End Sub
```

Clase ListBox (IX)

- Selección múltiple.
  - La propiedad `SelectionMode` permite seleccionar varios elementos al mismo tiempo. Puede tomar alguno de los siguientes valores:
    - ✓ `None`. No se puede seleccionar ningún elemento.
    - ✓ `One`. Sólo es posible seleccionar un valor (valor predeterminado).
    - ✓ `MultiSimple`. Permite seleccionar varios elementos.
      - La selección se realizar marcando cada elemento con el ratón o la barra espaciadora.
    - ✓ `MultiExtended`. Permite seleccionar varios elemento.
      - La selección se puede realizar marcando cada elemento y utilizando las teclas CTRL, SHIFT o las teclas del cursor.
  - La colección `SelectedItems` guarda los objetos seleccionados.
  - La colección `SelectedIndices` guarda los índices de los elementos seleccionados.
  - El método `GetSelected(indice)` permite saber si un elemento ha sido seleccionado.
  - El método `SetSelected(índice,valor)` permite modificar el estado de un elemento determinado.

La clase ListBox (X)



Colección Items

Índice	Objeto	Estado de la selección
0	Elemento 02	Seleccionado
1	Nuevo elemento	No seleccionado
2	Elemento 04	Seleccionado
3	Elemento 05	Seleccionado
4	Elemento 06	No seleccionado

Colección SelectedItems

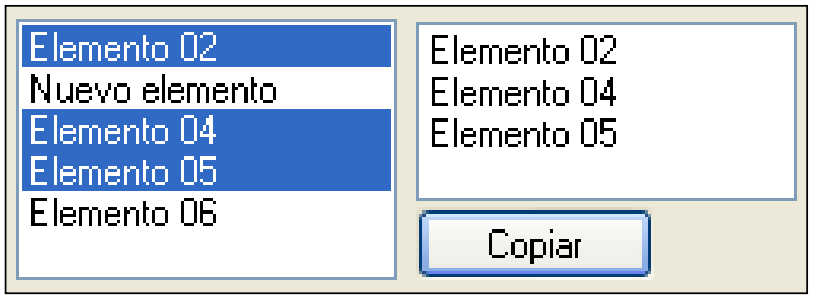
Índice	Objeto
0	Elemento 02
1	Elemento 04
2	Elemento 05

Colección SelectedIndices

Índice	Objeto
0	0
1	2
2	3

La clase ListBox (XI)

- Ejemplo: copiar los elementos seleccionados de un `ListBox` a otro al pulsar el botón Copiar:



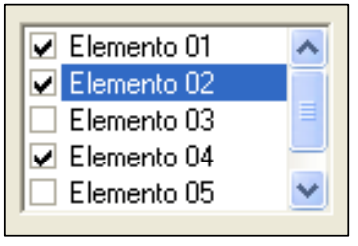

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) _
                          Handles Button1.Click
    For Each elem As Object In ListBox1.SelectedItems
        ListBox2.Items.Add(elem)
    Next
End Sub
```

Clase ListBox (XII)

- Otras propiedades.
  - Propiedad `Sorted`. Un valor `True` permite ordenar los elementos.
    - ✓ Cuando la propiedad está a `True`, el método `Add` e `Insert` añaden los elementos ordenados.
  - Barras de desplazamiento.
    - ✓ La propiedad `ScrollAlwaysVisible` determina si se verá siembre la barra de desplazamiento.
    - ✓ La propiedad `HorizontalScrollbar` permite visualizar una barra de desplazamiento horizontal.
  - Propiedad `IntegralHeight`.
    - ✓ Indica si la altura de la lista sólo puede visualizar elementos completos.
      - Un valor a `True` (predeterminado) impide que se visualicen elementos parcialmente.

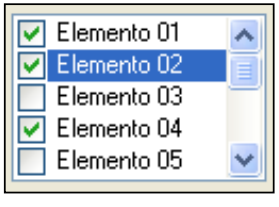
Clase CheckedListBox

- Hereda de `ListBox` y utiliza sus mismos miembros.
  - Permite visualizar un cuadro de lista con casillas de verificación a la izquierda de sus elementos.
- No permite la selección de varios objetos, aunque si permite marcar las casillas de varios de ellos.
- Las colecciones `SelectedItems` y `SelectedIndices` se sustituyen por `CheckedItems` y `CheckedIndices`.
- El evento `ItemCheck` se produce cuando cambia el estado de alguno de sus elementos.
  - Utiliza un argumento del tipo `System.Windows.Forms.ItemCheckEventArgs` con los siguientes miembros:
    - ✓ `Index`. Índice del elemento que va a cambiar.
    - ✓ `CurrentValue`. Estado actual del elemento (`Checked`, `Unchecked`, `Indeterminate`).
    - ✓ `NewValue`. Nuevo estado del elemento.

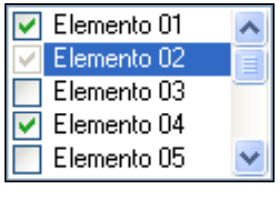


Clase CheckedListBox (II)

- Método `GetItemChecked(índice)`.
  - Devuelve `True` si el elemento está activado (estado `Checked` o `Indeterminate`) o `False` en caso contrario.
- Método `SetItemChecked(índice,estado)`.
  - Permite establecer el elemento a los estados `Checked` o `Unchecked`.
- Método `GetItemCheckState(índice)`.
  - Permite obtener el estado del elemento.
    - ✓ Devuelve `CheckedState.Checked`, `CheckedState.Unchecked` o `CheckedState.Indeterminate`.
- Método `SetItemCheckState(índice, estado)`.
  - Permite establecer el estado del elemento a `CheckedState.Checked`, `CheckedState.Unchecked` o `CheckedState.Indeterminate`.



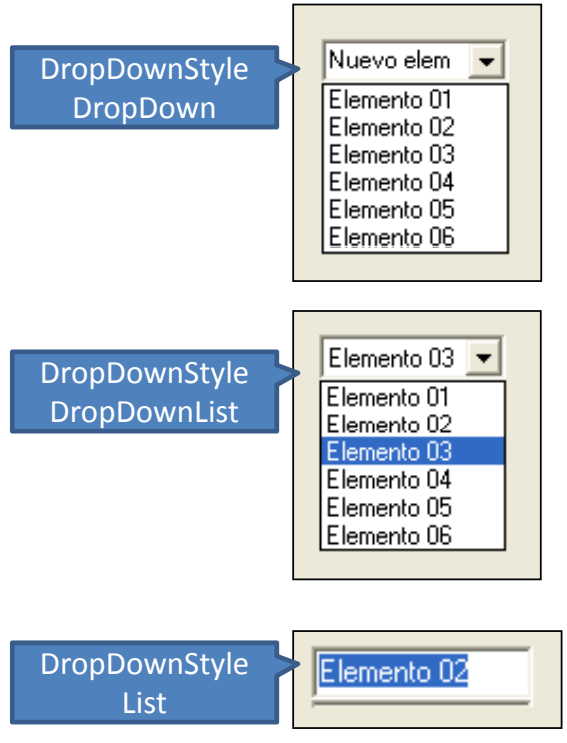

```
Debug.WriteLine(CheckedListBox1.GetItemChecked(0)) 'Escribe True
Debug.WriteLine(CheckedListBox1.GetItemChecked(2)) 'Escribe False
```




```
CheckedListBox1.SetItemCheckState(1, CheckState.Indeterminate)
Debug.WriteLine(CheckedListBox1.GetItemCheckState(0))'Escribe Checked
Debug.WriteLine(CheckedListBox1.GetItemCheckState(1))'Escribe Indeterminate
Debug.WriteLine(CheckedListBox1.GetItemCheckState(2))'Escribe Unchecked
```

Clase ComboBox

- Combina un cuadro de lista con un cuadro de texto.
  - Presenta la mayoría de las propiedades, métodos y eventos de ambos controles.
    - ✓ No permite multiselección.
    - ✓ No captura el evento `DoubleClick`.
    - ✓ La propiedad `SelectedIndex` también vale -1 si el usuario está editando el texto.
- Propiedad `DropDownStyle`.
  - `DropDown`. Un cuadro de lista desplegable en el que el usuario puede editar el texto.
  - `DropDownList`. Un cuadro de lista desplegable en el que el usuario no puede editar texto.
    - ✓ Se puede acceder a los elementos a partir de la inicial.
  - `Simple`. Una lista no desplegable en la que sólo se ve el elemento seleccionado o el que edita el usuario.

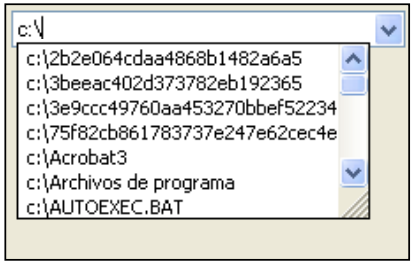


Clase ComboBox (II)

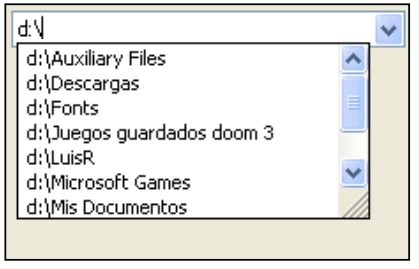
- Autocompletar el contenido de un `ComboBox`.
  - La propiedad `AutoCompleteMode` permite indicar si queremos que se autocomplete el contenido de los escrito en un `ComboBox`:
    - ✓ `None`, no se autocompleta.
    - ✓ `Append`, al teclear los primeros caracteres añaden los que faltan.
    - ✓ `Sugest`, despliega una lista con las posibles opciones a completar.
    - ✓ `SugestAppend`, añade los caracteres que faltan y despliega la lista.
  - La propiedad `AutoCompleteSource`, indica el origen de los datos a autocompletar.
    - ✓ `FileSystem` Especifica el sistema de archivos como origen.
    - ✓ `HistoryList` Incluye los URL en la lista de historial.
    - ✓ `RecentlyUsedList` Incluye los URL de la lista de las direcciones usadas recientemente.
    - ✓ `AllUrl` Especifica el equivalente de `HistoryList` y `RecentlyUsedList` como el origen.
    - ✓ `AllSystemSources` Especifica el equivalente de `FileSystem` y `AllUrl` como el origen.
    - ✓ `FileSystemDirectories` Especifica que sólo los nombres de directorio y no los nombres de archivo se finalizarán automáticamente.
    - ✓ `ListItems`. Especifica que los elementos de la lista son el origen.
    - ✓ `CustomSource` Especifica que se utilizarán las cadenas que formen la propiedad `AutoCompleteCustomSource`.

Clase ComboBox (III)

- AutocompleteSource

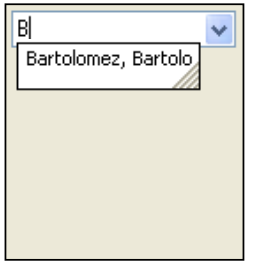


FileSystem

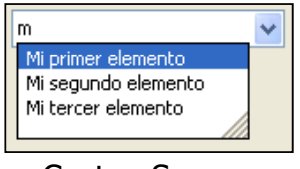


FileSystemDirectories

HistoryList, AllUrl
y RecentlyUsedList



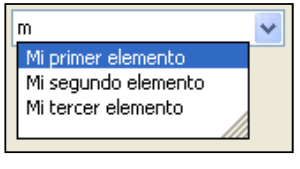
ListItems



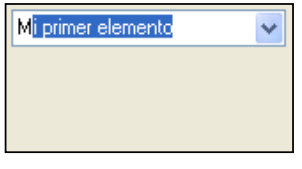
CustomSource

Con AutoCompleteSource a CustomSource es necesario rellenar los elementos de la colección personalizada en la propiedad AutoCompleteCustomSource

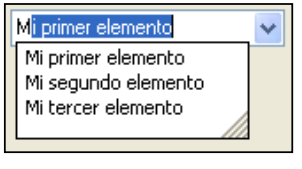
- AutocompleteMode.



Sugest



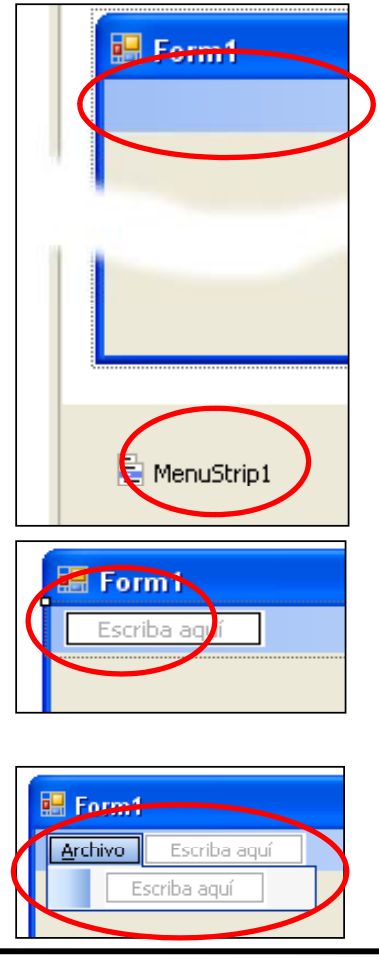
Append



SugestAppend

Menús

- Los menús se construyen a partir de la clase `MenuStrip`.
  - `MenuStrip` representa un contenedor de la estructura de menús de un formulario.
- Crear un menú.
  - Al arrastrar un menú en el formulario, aparecerá en la bandeja de componentes y el área donde aparecerá el menú acoplada en la parte superior del formulario.
- Elementos del menú.
  - Son objetos de la clase `ToolStripMenuItem`.
  - Al seleccionar la barra de menús o el control `MenuStrip` en la bandeja de componentes, el entorno permitirá escribir el título del menú.
  - A medida que se dan nombres a los elementos `ToolStripMenuItem` del menú, aparecen posiciones para un nuevo elemento de menú del mismo nivel o un menú desplegable



Menús (II)

- Títulos de los menús.
  - El carácter *ampersand* (&) hace que el carácter siguiente se convierta en la tecla de acceso rápido.
    - ✓ Las recomendaciones de diseño de la interfaz indican que **todos** los elementos de un menú deben tener tecla de acceso rápido.
- Nombres de los objetos `MenuStrip` y `ToolStripMenuItem`.
  - Puesto que un formulario sólo tendrá normalmente un elemento `MenuStrip` la mayoría de las veces no será necesario dar un nombre distinto.
  - En los elementos `ToolStripMenuItem` Visual Studio pone por omisión un nombre formado por el título y el sufijo `ToolStripMenuItem` (por ejemplo, `ArchivoToolStripMenuItem`).
    - ✓ Para los submenús, se recomienda utilizar para el nombre, el título del menú de jerarquía superior y el nombre del actual.
      - Por ejemplo una opción Nuevo dentro del menú Archivo podría tener el nombre `ArchivoNuevoToolStripMenuItem`.

Menús (III)

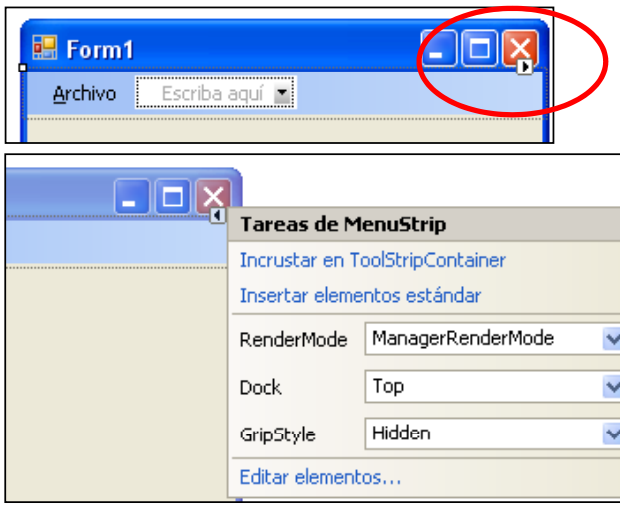
- Tipos de elementos de menú.
  - Por omisión el aspecto del elemento de menú es una etiqueta con texto estático.
  - Es posible cambiar ese aspecto para mostrar un ComboBox o un TextBox.
    - ✓ Al pulsar con el botón secundario en un elemento de menú, seleccionar la opción "Convertir en".
      - `MenuItem`. El aspecto por omisión.
      - `ComboBox`. Aparece una lista desplegable. La propiedad `Items` del elemento de menú permite añadir elementos.
        - Se pueden añadir elementos al ComboBox de forma dinámica con el método `Add` de la propiedad `Item` del objeto `ToolStripMenuItem`.
        - Se puede acceder al texto seleccionado por la propiedad Text del objeto.
      - `TextBox`. Aparece como un cuadro de texto editable.
        - Se puede acceder al texto mediante la propiedad `Text` del objeto `ToolStripMenuItem`.
      - Los elementos de tipo ComboBox y TextBox **no pueden** tener submenús.

Menús (IV)

- Aspecto del menú.
  - Para agregar un separador entre dos elementos de menú, en el menú contextual del elemento, seleccionar la opción "Insertar" y en el submenú "Separator".
  - Marcas de verificación.
    - ✓ La propiedad `Checked` permite añadir una marca de verificación al menú.
    - ✓ En tiempo de ejecución, mediante código, es posible modificar la marca mediante la propiedad `CheckState`.
      - Puede tomar los valores `Checked`, `Unchecked` o `Indeterminate`.
    - ✓ Las propiedades `Checked` y `CheckState` también permiten obtener el estado de verificación.
    - ✓ La propiedad `CheckOnClick`, permiten modificar el estado de la verificación al hacer clic.
  - Imágenes.
    - ✓ Si se trata de un elemento de menú de tipo `MenuItem` es posible añadir una imagen al margen con la propiedad `Image`.
    - ✓ Si la casilla de verificación está activada, aparecerá un recuadro rodeando la imagen.
  - Habilitar y deshabilitar elementos de un menú.
    - ✓ La propiedad `Enabled`, permite deshabilitar las opciones no disponibles en un momento dado.
      - No debería ser posible acceder a aquellas opciones no disponibles: la interfaz debe mostrar pistas visuales.
      - También es posible que no se muestren las opciones mediante la propiedad `Visible`.

Menús (V)

- Teclas de método abreviado.
  - La propiedad `ShortcutKeys` permite asociar una tecla de método abreviado al elemento de menú.
    - ✓ Sólo deben tener teclas de método abreviado las opciones finales de menú.
    - ✓ Si la propiedad `ShowShortcutKeys` está a `True`, aparecerá la combinación de teclas a la derecha.
- Añadir opciones estándar de menú.
  - En el glifo ( ) de etiqueta inteligente del control `MenuStrip` y seleccionar "Insertar elementos estándar".
  - Se añaden los elementos estándar de un menú Windows.



Menús (VI)

- Controlar los eventos.
  - Para asociar una acción a cualquier elemento de un menú se utilizará el evento `Click`.

```
Private Sub ArchivoNuevoToolStripMenuItem_Click(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles NuevoToolStripMenuItem.Click
  'Introducir el código correspondiente a la opción Nuevo del menú Archivo
End Sub
```

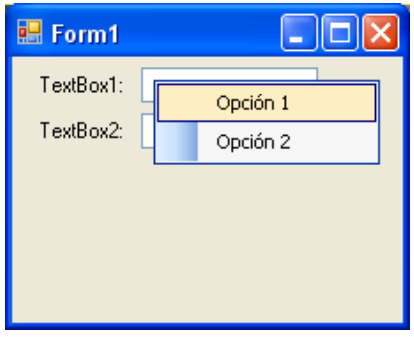
  - Si se utilizan elementos de menú con casillas de verificación, los eventos `CheckedChanged` y `CheckStateChanged` permite verificar si se ha modificado su estado.
    - ✓ Funcionan de la misma forma que sus equivalentes de la clase `CheckBox`.

Menús (VII)

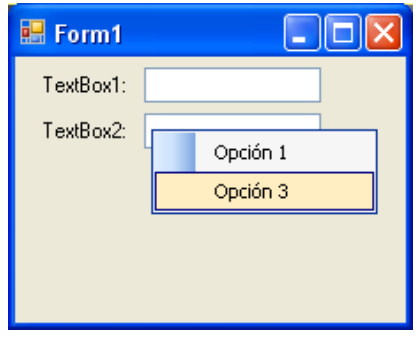
- Menús emergentes (menús contextuales o *Popup*).
  - Se activan al hacer clic con el botón secundario en un control.
  - El contenedor será en este caso un objeto de la clase `ContextMenuStrip`.
    - ✓ También hay que arrastrarlo a la bandeja de componentes.
    - ✓ Aunque en un formulario normalmente sólo hay un menú principal (objeto de la clase `MenuStrip`), puede haber tantos menús emergentes cómo se desee.
    - ✓ El objeto `ContextMenuStrip` contendrá los elementos de menú (`ToolStripMenuItem`).
    - ✓ Para asociar el menú emergente a un control o formulario, será necesario indicarlo en la propiedad `ContextMenuStrip` del control o formulario.

Menús (VIII)

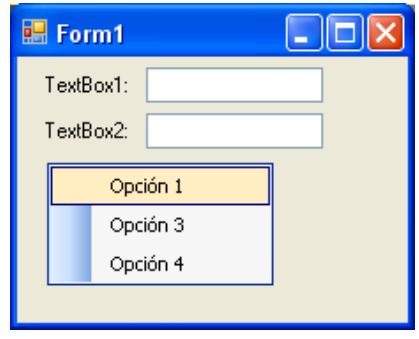
- El evento `Opening` se produce antes de que se abra el menú contextual.
  - Se puede utilizar para añadir distintas opciones a un menú contextual dependiendo del control que se ha abierto.
    - ✓ La propiedad `SourceControl` de la clase `ContextMenuStrip`, permite guardar una referencia al objeto sobre el que se abrió el menú emergente.
  - En el ejemplo, el mismo objeto `ContextMenuStrip` muestra tres menús emergentes distintos, aunque con opciones compartidas.



El menú emergente del control TextBox1 muestra los elementos Opción 1 y Opción 2



El menú emergente del control TextBox2 muestra los elementos Opción 1 y Opción 3



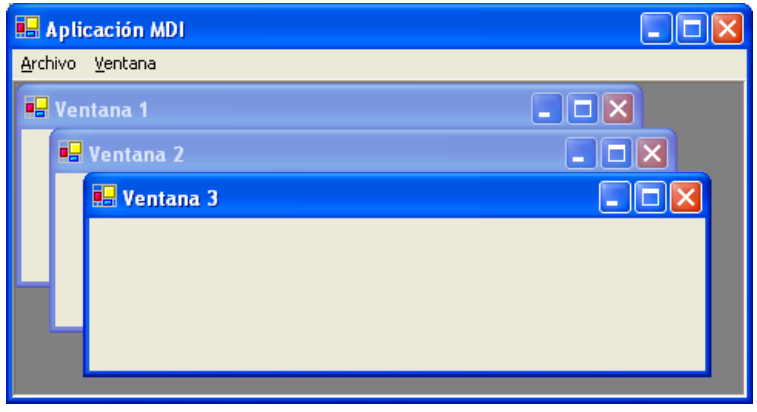
El menú emergente del formulario muestra los elementos Opción 1, Opción 3 y Opción 4

Menús (IX)

```
'Se supone que TextBox1, Textbox2 y Form1 tienen
'la propiedad ContextMenuStrip a ContextMenuStrip1.
'También existen los elementos de menú Opción1, Opción2, Opción3 y Opción4
Private Sub ContextMenuStrip1_Opening(ByVal sender As System.Object, _
           ByVal e As System.ComponentModel.CancelEventArgs) _
           Handles ContextMenuStrip1.Opening
    If ContextMenuStrip1.SourceControl Is TextBox1 Then
        'Limpia el contenido anterior del menú
        ContextMenuStrip1.Items.Clear()
        ContextMenuStrip1.Items.Add(Opción1ToolStripMenuItem)
        ContextMenuStrip1.Items.Add(Opción2ToolStripMenuItem)
    ElseIf ContextMenuStrip1.SourceControl Is TextBox2 Then
        ContextMenuStrip1.Items.Clear()
        ContextMenuStrip1.Items.Add(Opción1ToolStripMenuItem)
        ContextMenuStrip1.Items.Add(Opción3ToolStripMenuItem)
    ElseIf ContextMenuStrip1.SourceControl Is Me Then
        ContextMenuStrip1.Items.Clear()
        ContextMenuStrip1.Items.Add(Opción1ToolStripMenuItem)
        ContextMenuStrip1.Items.Add(Opción3ToolStripMenuItem)
        ContextMenuStrip1.Items.Add(Opción4ToolStripMenuItem)
    End If
End Sub
```

Formularios MDI

- Aplicaciones SDI (*Single Document Interface*).
  - La aplicación sólo permite tener abierta una única ventana al mismo tiempo (por ejemplo la aplicación WordPad).
- Aplicaciones MDI (*Multiple Document Interface*).
  - Existe una ventana MDI primaria (padre) que actúa como contenedor de ventanas MDI secundarias (hijas).
    - ✓ Es útil cuando una aplicación requiere de varias ventanas de características generales o para navegar entre las distintas ventanas de una aplicación.
  - En algunas aplicaciones actuales (cómo Office 2007) se sigue un modelo similar al MDI:
    - ✓ La aplicación mantiene varias ventanas de documento abiertas, aunque no existe una ventana primaria contenedora.



Formularios MDI (II)

- Formulario MDI primario.
  - Un objeto de la clase `Form` con la propiedad `IsMdiContainer` a `True`.
- Formulario MDI secundario.
  - Un objeto de la clase `Form` cuya propiedad `MdiParent` apunta al formulario MDI primario.
- Abrir una ventana MDI secundaria.

```
Private Sub AbrirFormularioHijo()
  Dim frm As New FormularioHijo     'La clase FormularioHijo ya está creada
  frm.MdiParent = Me                'MdiParent apunta al formulario actual
  Static Dim numHijos As Integer    'Esta variable sirve para el título
  numHijos += 1                     'del formulario hijo.Se incrementa en 1
  frm.Text = "Ventana " & numHijos 'Nuevo título de la ventana
  frm.Show()                        'Por último se muestra el formulario
End Sub
```

Formularios MDI (III)

- Acceso a los formularios hijo.
  - Los formularios primarios guardan en la propiedad `MdiChildren` la colección de formularios secundarios.

```
'Da color rojo a todos los formularios secundarios desde un formulario hijo
For Each frm As Form In My.Forms.frmAplicaciónMDI.MdiChildren
    frm.BackColor = Color.Red
Next
```

  - La propiedad `ActiveMdiChild` de la clase `Form` devuelve una referencia al formulario hijo activo o `Nothing` si no existe ningún formulario hijo activo.
  - El método `ActivateMdiChild(`*`referenickFormularioHijo`*`)` activa una ventana secundaria concreta.
  - El evento `MdiChildActivate` se desencadena cuando se abre un formulario secundario.

```
'Cambia el título del formulario primario cada vez que cambia el secundario
Private Sub Form1_MdiChildActivate(ByVal sender As Object, _
         ByVal e As System.EventArgs) Handles MyBase.MdiChildActivate
    If Me.ActiveMdiChild Is Nothing Then
        Me.Text = "Aplicación MDI"
    Else
        Me.Text = "Aplicación MDI - " & Me.ActiveMdiChild.Text
    End If
End Sub
```

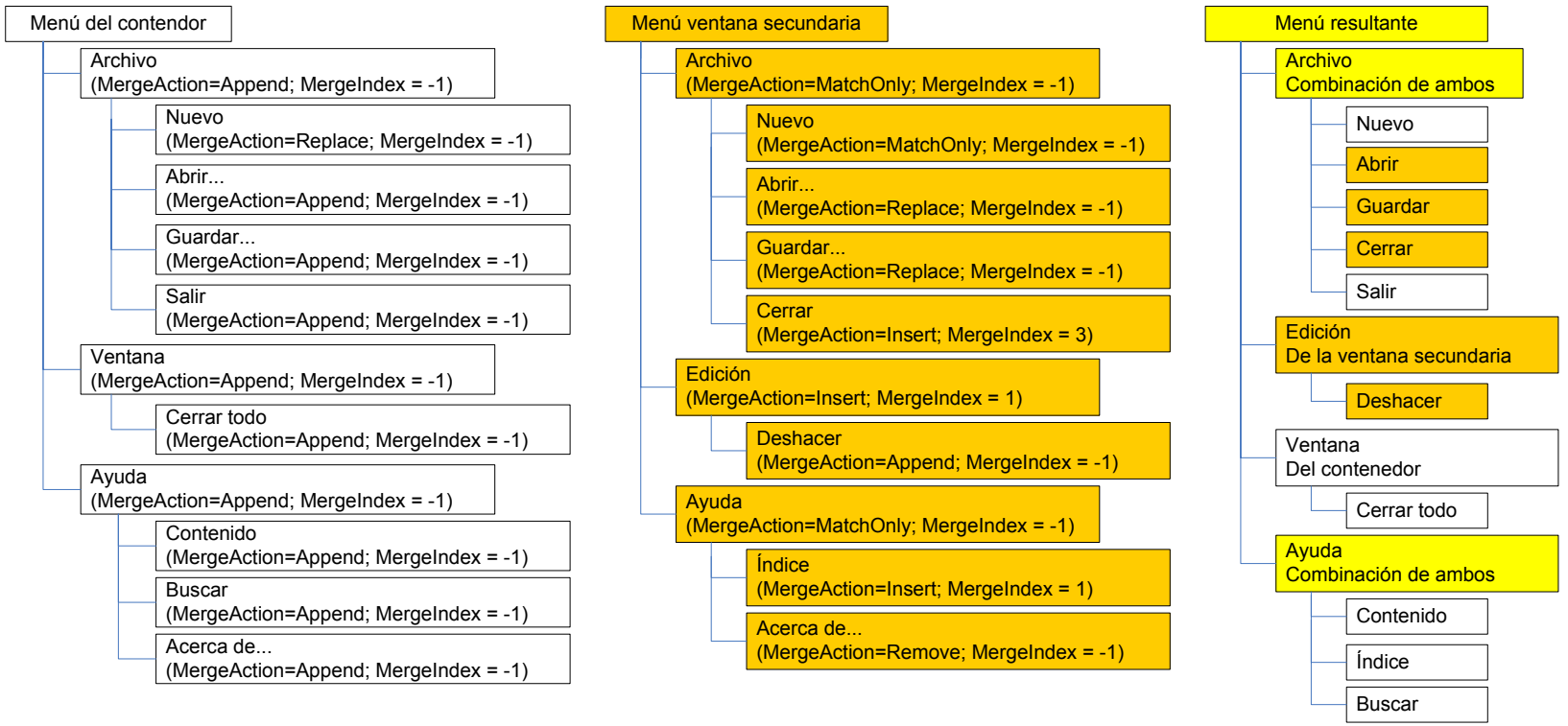
Formularios MDI (IV)

- El menú Ventana.
  - En ocasiones las aplicaciones MDI presentan una opción de menú para administrar los formularios hijo.
  - Para que en ese menú aparezca una lista con todas las ventanas secundarias, es necesario indicar al objeto `MenuStrip` de la ventana padre en que elemento aparecerá la lista de ventanas mediante la propiedad `MdiWindowsListItem`.
    - ✓ Normalmente se tratará de un menú de primer nivel (p.e. el menú Ventana).
  - Para organizar las ventanas secundarias se utiliza el método `LayoutMdi` de la clase `Form`.
    - ✓ Organización en cascada.
      ```
      Me.LayoutMdi(MdiLayout.Cascade)
      ```
    - ✓ Organización en mosaico horizontal.
      ```
      Me.LayoutMdi(MdiLayout.TileHorizontal)
      ```
    - ✓ Organización en mosaico vertical.
      ```
      Me.LayoutMdi(MdiLayout.TileVertical)
      ```
    - ✓ Organiza los iconos de las ventanas en el caso de que estén minimizadas.
      ```
      Me.LayoutMdi(MdiLayout.ArrangeIcons)
      ```

Formularios MDI (V)

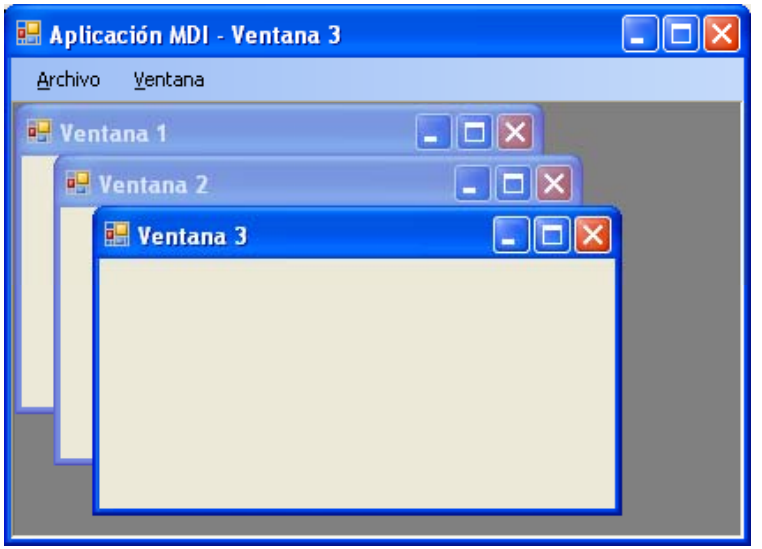
- Combinación de menús.
  - Los menús del formulario secundario activo se combinarán con el del formulario contenedor.
    - ✓ La propiedad `AllowMerge` de la clase `Form` posibilita o impide la combinación de menús.
    - ✓ La propiedad `MergeAction` especifica el tipo de combinación que se utilizará.
      - Su valor es un miembro del enumerado `MenuAction`:
        - `Append`. Los elemento del formulario secundario se añaden al final de los del formulario contenedor.
        - `Insert`. Inserta el elemento en el formulario contendor en la posición indicada por la propiedad `MergeIndex`.
        - `Replace`. Reemplaza el elemento coincidente (el que tenga el mismo texto en la etiqueta).
        - `Remove`. Elimina los elementos que tengan el mismo nombre.
        - `MatchOnly`. Realiza alguna acción si los elementos coinciden. La acción a realizar dependerá de los elementos de los submenús o del valor `MergeAction` del otro elemento coincidente.

Formularios MDI (VI)



Formularios MDI (VII)

- Aplicación de ejemplo.
  - Sólo será capaz de crear nuevas ventanas, cerrarlas y organizarlas.
  - El formulario principal sólo tendrá el menú archivo con dos opciones de menú:
    - ✓ Cada vez que se da la opción Nuevo del menú Archivo se abrirá una nueva ventana con el título "Ventana xxx".
    - ✓ La opción Salir del menú Archivo terminará la aplicación.
  - La ventana secundaria tendrá los menús Archivo y Ventana.
    - ✓ El menú archivo se combinará con el de la ventana principal y añadirá la opción Cerrar que cerrará la ventana.
    - ✓ El menú ventana tendrá como opciones:
      - Cerrar todo. Cierra todas las ventanas.
      - Una lista de las ventanas abiertas.
      - Una opción Organizar con submenús para organizar las ventanas en cascada, en mosaico horizontal, mosaico vertical u organizar iconos.



Formularios MDI (VIII)

```
'Código del formulario principal
'Este evento se produce al intentar cerrar el formulario principal
Private Sub frmAplicaciónMDI_FormClosing(ByVal sender As Object, _
                        ByVal e As System.Windows.Forms.FormClosingEventArgs) _
                        Handles Me.FormClosing
    'Si existen hijos abiertos
    If Me.MdiChildren.Length <> 0 Then
        'Se preguna si se desea seguir cerrando
        If MessageBox.Show("Todavía quedan ventanas abiertas ¿Desea continuar", _
                           "Aplicación MDI", MessageBoxButtons.YesNo, _
                            MessageBoxIcon.Exclamation, _
                             MessageBoxDefaultButton.Button2) = _
                                        Windows.Forms.DialogResult.No Then
            'Si no se quiere cerrar la aplicación
            'la propiedad Cancel del evento se pone a True,
            'por lo que se cancela la operación de cierre
            e.Cancel = True
        End If
    End If
End Sub
```

Formularios MDI (IX)

```
'Cuando se activa un formulario MDI hijo
Private Sub frmAplicaciónMDI_MdiChildActivate(ByVal sender As Object, _
                    ByVal e As System.EventArgs) Handles Me.MdiChildActivate
    'Si no hay ningún formulario segundario
    If Me.ActiveMdiChild Is Nothing Then
        Me.Text = "Aplicación MDI"
        'De esta forma no se muestra el menú ventana cuando no hay ventanas hijas
        VentanaToolStripMenuItem.Visible = False
    Else
        'Si no se pone el título del formulario seguido del nombre de la ventana
        Me.Text = "Aplicación MDI - " & Me.ActiveMdiChild.Text
        'De esta forma se muestra el menú ventana cuando  hay ventanas hijas
        VentanaToolStripMenuItem.Visible = True
    End If
End Sub
Private Sub NuevoToolStripMenuItem_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) Handles NuevoToolStripMenuItem.Click
   Dim frm As New frmFormularioHijo
   frm.MdiParent = Me
   Static numHijos As Integer        'numHijos lleva un contador permanente
   '                                  de ventanas secunarias
   numHijos += 1
   frm.Text = "Ventana " & numHijos  'El título de la ventana incluy el numHijos
   frm.Show()
End Sub
```

Formularios MDI (X)

```
'Código del formulario secundario
Private Sub MosaicohorizontalToolStripMenuItem_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles MosaicohorizontalToolStripMenuItem.Click
   My.Forms.frmAplicaciónMDI.LayoutMdi(MdiLayout.TileHorizontal)
End Sub

Private Sub MosaicoverticalToolStripMenuItem_Click(ByVal sender As Object, _
                           ByVal e As System.EventArgs) _
                           Handles MosaicoverticalToolStripMenuItem.Click
    My.Forms.frmAplicaciónMDI.LayoutMdi(MdiLayout.TileVertical)
End Sub

Private Sub CascadaToolStripMenuItem_Click(ByVal sender As System.Object, _
                        ByVal e As System.EventArgs) _
                        Handles CascadaToolStripMenuItem.Click
    My.Forms.frmAplicaciónMDI.LayoutMdi(MdiLayout.Cascade)
End Sub

Private Sub OrganizarIconosToolStripMenuItem_Click(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles OrganizarIconosToolStripMenuItem.Click
    My.Forms.frmAplicaciónMDI.LayoutMdi(MdiLayout.ArrangeIcons)
End Sub
```

Formularios MDI (XI)

```
Private Sub CerrarTodooolStripMenuItem_Click(ByVal sender As System.Object, _
                                ByVal e As System.EventArgs) _
                                Handles CerrarTodoToolStripMenuItem.Click
    'Recorre todos los formularios de la ventana contenedora
    'y los cierra
    For Each frm As Form In My.Forms.frmAplicaciónMDI.MdiChildren
        frm.Close()
    Next
End Sub

Private Sub CerrarToolStripMenuItem_Click(ByVal sender As System.Object, _
                               ByVal e As System.EventArgs) _
                               Handles CerrarToolStripMenuItem.Click
    'Cierra la ventana actual (es decir el formulario hijo activo
    Me.Close()
End Sub
```

Cuadros de diálogo comunes

- Windows ofrece una serie de cuadros de diálogos comunes que puede utilizar cualquier aplicación Windows.
- Desde .NET es posible acceder a los siguientes cuadros de diálogos:
  - Abrir archivo (clase `OpenFileDialog`).
  - Seleccionar carpeta (clase `FolderBrowserDialog`).
  - Guardar como (clase `SaveFileDialog`).
  - Colores (clase `ColorDialog`).
  - Fuentes (clase `FontDialog`).
  - Imprimir (clase `PrintDialog`).
  - Configurar página (clase `PageSetupDialog`).
  - Vista previa (clase `PrintPreviewDialog`).
- Todas las clases tienen los siguientes miembros comunes:
  - Método `ShowDialog()`. Muestra el cuadro y devuelve `DialogResult.Ok` o `DialogResult.Cancel` dependiendo del botón que pulse el usuario.
  - Método `Reset()` que restaura todas las propiedades a su valor por omisión.
- Los cuadros de diálogos no realizan por si ninguna acción, sólo sirven para seleccionar valores que, mediante propiedades, se utilizarán más tarde.

Cuadro de diálogo OpenFileDialog

- Muestra el típico cuadro de diálogo para seleccionar uno o varios archivos.
  - Se utiliza para seleccionar un archivo.
  - En principio, aparecen todos los archivos de la carpeta inicial o la que indique la propiedad `InitialDirectory`.
- La propiedad `Title` permite especificar el título del cuadro de diálogo, por omisión aparecerá la cadena "Abrir".
- Recuperar el archivo seleccionado.
  - La propiedad `FileName` guarda el nombre del archivo seleccionado.
  - Si se escribe directamente el nombre del archivo en el cuadro de texto correspondiente es posible detectar si el archivo o la carpeta existe mediante las propiedades lógicas `CheckFileExist` y `CheckPathExist`.
    - ✓ Si están a `True`, se detectará si existe ese nombre de archivo o carpeta al pulsar el botón Aceptar del cuadro de diálogo.

Cuadro de diálogo OpenFileDialog (II)

- Filtrar los archivos seleccionados.
  - La propiedad `Filter` permite añadir elementos a la lista de tipos de archivos.
    - ✓ El valor de esa propiedad será una cadena con el siguiente formato:
      - Texto1|filtro1|Texto2|filtro2….
      - Para que aparezcan todos los archivos, archivos txt o algunos archivos gráficos:

```
OpenFileDialog1.Filter = "Todos los archivos (*.*) |*.*|" & _
                         "Archivos de texto (*.txt) |*.txt| " & _
                         "Archivos gráficos |*.gif;*.bmp;*.jpg"
```

  - La propiedad `FilterIndex` permite decidir mediante un número entero el índice del tipo de archivo que aparecerá inicialmente.
- La propiedad `ShowReadOnly` es un valor lógico que permite que aparezca la casilla de verificación "Abrir como sólo lectura".
  - La propiedad `ReadOnlyChecked` permitirá recuperar el valor introducido por el usuario en esa casilla para su posterior proceso.

Cuadro de diálogo OpenFileDialog (III)

- Selección múltiple.
  - La propiedad `Multiselect` permite seleccionar múltiples archivos del cuadro de diálogo.
  - En este caso los archivos seleccionados se cargan en un array de cadena representado por la propiedad `FileNames`.
  - Ejemplo: carga en un `ListBox` nombres de los archivos seleccionados.

```
If OpenFileDialog1.ShowDialog() = Windows.Forms.DialogResult.OK Then
    'Recorre todos los elementos del array FileNames
    For Each str As String In OpenFileDialog1.FileNames
        'FileNames contiene la especificación completa del archivo
        'Para obtener sólo el nombre, troceo la cadena con Split
        Dim aux() As String = str.Split("\")
        'y me quedo con el último elemento del array resultante
        ListBox1.Items.Add(aux(aux.GetUpperBound(0)))
    Next
End If
```

Cuadro de diálogo OpenFileDialog (IV)

- Ejemplo: seleccionar un archivo gráfico y cargarlo en un control `PictureBox`.


```
Private Sub Cargar_Click(ByVal sender As System.Object, _
                         ByVal e As System.EventArgs) Handles Cargar.Click
   OpenFileDialog1.Filter = "Todos los archivos (*.*)|*.*|" & _
                            "Archivos gráficos|*.bmp;*.gif;*.jpg;*.png"
   OpenFileDialog1.FilterIndex = 1
   If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
       PictureBox1.Image = Image.FromFile(OpenFileDialog1.FileName)
   Else
       PictureBox1.Image = Nothing
   End If
End Sub
```

Cuadro de diálogo FolderBrowserDialog

- Permite seleccionar una carpeta a partir de un cuadro de diálogo estándar.
- La propiedad `SelectedPath` devuelve el nombre de la carpeta seleccionada.
- Es posible activar u ocultar el botón "Nueva carpeta" mediante la propiedad `ShowNewFolderButton`.
- La carpeta inicial será la que indique la propiedad `RootFolder`.
  - Puede tomar alguno de los valores de la enumeración `Environment.SpecialFolder`:
    - ✓ `Desktop`. El escritorio.
    - ✓ `MyComputer`. Mi PC.
    - ✓ `Personal`. Mis documentos.
    - ✓ `MyMusic`. Mi música.
    - ✓ `MyPictures`. Mis imágenes.
    - ✓ `ProgramFiles`. Archivos de programa
    - ✓ …

Cuadro de diálogo SaveFileDialog

- Similar al cuadro Abrir archivo pero con distinta funcionalidad.
  - Se utiliza para dar un nombre a un archivo.
- Mantiene las propiedades `Title`, `InitialDirectory`, `Filename`, `Filter`, `FilterIndex`.
- No admite multiselección.
  - La propiedad `Filenames` pierde su sentido.
- Las propiedades `CheckFileExist`, `CheckPathExist`, `ShowReadOnly` y `ReadOnlyCheck` existen, pero pierden su utilidad.
- Como nuevas propiedades tiene `CreatePrompt` y `OverwritePrompt` que avisan si se va a crear un archivo nuevo o se va a sobreescribir un archivo.
- La propiedad `DefaultExt` permite incluir una cadena para la extensión por omisión del archivo.

Cuadro de diálogo ColorDialog

- Permite seleccionar un color de la paleta.
- Propiedad `FullOpen`.
  - Con un valor `True`, el cuadro de diálogo se abre con la paleta de colores personalizados.
- Propiedad `AllowFullOpen`.
  - Con un valor `True`, permite a los usuarios elegir un color personalizado.
- La propiedad `Color` permite devuelve el color seleccionado al pulsar el botón Aceptar.

```
If ColorDialog1.ShowDialog() = DialogResult.OK Then
    Button5.BackColor = ColorDialog1.Color
End If
```

Cuadro de diálogo FontDialog

- Permite seleccionar una fuente cuyas características devolverá en la propiedad `Font` que se puede asignar a la propiedad `Font` de cualquier clase que disponga de ella.

```
If FontDialog1.ShowDialog() = DialogResult.OK Then
    Label1.Font = FontDialog1.Font
End If
```

- En principio aparecen todas las fuentes, pero podemos seleccionar las deseadas mediante un valor lógico en las propiedades:
  - `AllowVectorFonts`. Admite o no fuentes vectoriales.
  - `AllowVerticalFonts`. Admite o no fuentes verticales.
  - `FixedPichOnly`. Admite sólo fuentes de paso fijo.
- Las propiedades `MaxSize` y `MinSize` permiten definir el tamaño máximo y mínimo de la lista de tamaños de fuentes.

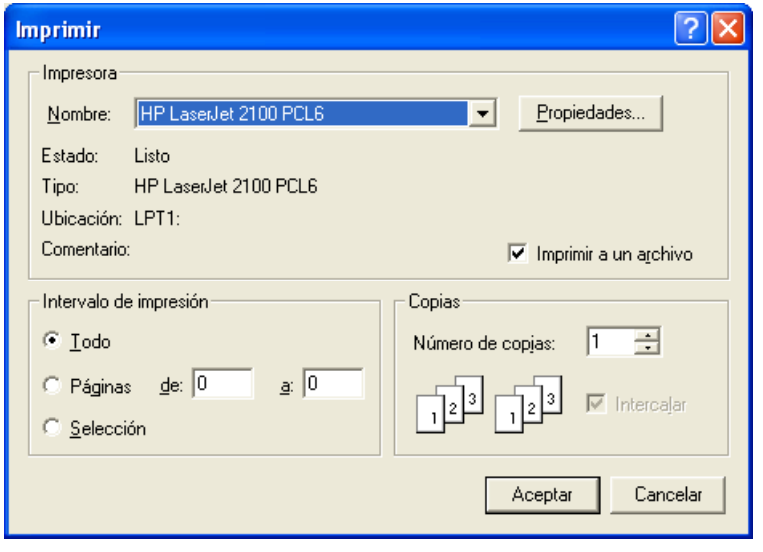
Cuadro de diálogo FontDialog (II)

- Características especiales.
  - La propiedad `ShowEffects` permite mostrar las casillas de verificación de subrayado y tachado.
  - La propiedad `ShowColor` permite mostar y seleccionar de una lista de colores.
    - ✓ El color seleccionado será recogido en la propiedad Color que será necesario asignar independientemente de la fuente seleccionada.
      `Label1.ForeColor = FontDialog1.Color`
- La propiedad `ShowApply` permite mostrar en el cuadro de diálogo el botón aplicar.
  - El evento `Apply` permitirá controlar si se ha pulsado dicho botón y, mediante código, mostrar una vista previa de la nueva fuente.

```
...
Dim fuente As Font = Label1.Font 'Para poder recuperar la fuente si se pulsa Cancelar
If FontDialog1.ShowDialog() = DialogResult.OK Then
    Label1.Font = FontDialog1.Font
Else
    Label1.Font = fuente         'Si se pulsa Cancelar se vuelve a la fuente original
End If
...
Private Sub FontDialog1_Apply(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) Handles FontDialog1.Apply
    Label1.Font = FontDialog1.Font
End Sub
```

Cuadro de diálogo PrintDialog

- Permite especificar las propiedades de un objeto `PrinterSettings`, necesario para realizar una impresión.
- Especificar que propiedades podemos cambiar:
  - `AllowPrintToFile`. Habilita o deshabilita la casilla de verificación Imprimir a un archivo.
  - `AllowSelection`. Habilita o deshabilita el botón de radio Selección.
  - `AllowSomePages`. Habilita o deshabilita el botón de radio Páginas para imprimir sólo un intervalo de páginas.
- `PrintToFile`. Devuelve un valor lógico si está marcada la casilla de verificación.
- La propiedad `PrinterSettings` devuelve el objeto `PrinterSettings` modificado por el cuadro de diálogo.



Cuadro de diálogo PrintDialog (II)

- Propiedades del objeto `PrinterSettings`.
  - `PrinterName`. Nombre de la impresora seleccionada.
  - `Copies`. El número de copias marcado en el cuadro Numero de copias.
  - `Collate`. Devuelve o establece si se ha marcado la casilla de verificación intercalar.
  - `PrintRange`. Devuelve o establece el botón de radio marcado en el intervalo de impresión.
    - ✓ Puede tomar alguno de los siguientes valores:
      - `PrintRange.AllPages`.
      - `PrintRange.Selection`.
      - `PrintRange.SomePages`.
  - `FromPage`. Devuelve o establece la página inicial.
  - `ToPage`. Devuelve o establece la página final.

Introducción a la impresión

- Para poder trabajar con las clases de impresión es necesario importar el espacio de nombres al comienzo del código:

```
Imports System.drawing.printing
```

- Para realizar la impresión es preciso seguir los siguientes pasos:
  - Crear y definir un objeto `PrintDocument`.
    - ✓ Se puede crear arrastrándolo desde la caja de herramientas a la bandeja de componentes o declarando una variable `PrintDocument` a nivel de módulo.
  - Crear y definir un objeto `PrinterSettings`.
    - ✓ Se puede realizar a partir de la propiedad `PrinterSettings` del cuadro de diálogo `PrintDialog`.
    - ✓ Las especificaciones de impresión del objeto `PrintDocument` se tomarán a partir de este objeto `PrinterSettings`.
    - ✓ El método `Print` de la clase `PrinterSettings` comenzará la impresión.


```
If PrintDialog1.ShowDialog() = DialogResult.OK Then
    PrintDocument1.PrinterSettings = PrintDialog1.PrinterSettings
    PrintDocument1.Print()
End If
```

Introducción a la impresión (II)

- El contenido de lo que se va a imprimir se debe realizar dentro del evento `PrintPage` de la clase `PrintDocument`.

```
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
                  ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
                  Handles PrintDocument1.PrintPage
   ...
End Sub
```

- `PrintPage` recibe un argumento de la clase `PrintPageEventArgs` que se utiliza para tomar las características de la página y el objeto `Graphics` que se imprimirá.
  - `e.MarginBounds`. Rectángulo con los márgenes de la página.
  - `e.PageBounds`. Rectángulo con los límites físicos de la página.
  - `e.HasMorePages`. Valor lógico que indica si hay o no más páginas a imprimir.
  - `e.PageSettings`. Configuración de la página para la página actual.

Introducción a la impresión (III)

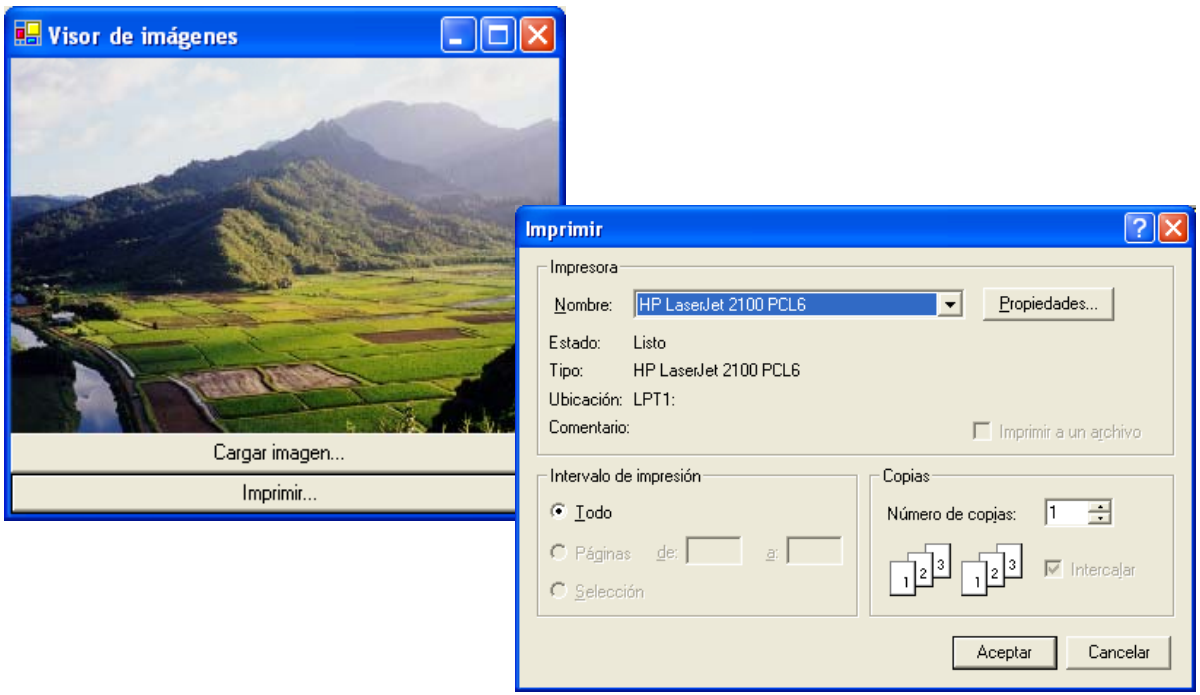
- `e.Graphics` permite definir lo que se va a imprimir con los métodos `DrawXXX` de la clase `Graphics`.
  - `e.Graphics.DrawString(`*`cadena`*`,`*`fuente`*`,`*`pincel`*`,`*`X`*`,`*`Y`*`)`.
    - ✓ Imprime la cadena con la fuente seleccionada y el color del pincel seleccionado en la posición de la página X,Y.
  - `e.Graphics.DrawImage(`*`image`*`,`*`X`*`,`*`Y`*`,`*`ancho`*`,`*`alto`*`)`.
    - ✓ Imprime el objeto Image especificado en la posición X e Y de la página con un ancho y alto específico.

```
'Imprime el contenido de TextBox1
Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
                  ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
                  Handles PrintDocument1.PrintPage
    Dim margenIzq As Single = e.MarginBounds.Left
    Dim margenSup As Single = e.MarginBounds.Top
    e.Graphics.DrawString(TextBox1.Text, Me.Font, Brushes.Black, _
                          margenIzq, margenSup)
End Sub
```

  - ✓ En http://msdn.microsoft.com/en-us/library/xdt36c58.aspx se puede encontrar información adicional sobre la impresión.

Introducción a la impresión (IV)

- Ejemplo: añadir al ejemplo de la dispositiva 99 un botón para imprimir la imagen.

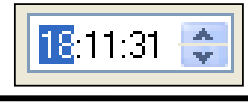


Introducción a la impresión (V)

```
Imports System.Drawing.Printing
...
Private Sub Imprimir_Click(ByVal sender As System.Object, _
                           ByVal e As System.EventArgs) Handles Imprimir.Click
    PrintDialog1.AllowPrintToFile = False
    PrintDialog1.AllowSelection = False
    PrintDialog1.AllowSomePages = False
    PrintDialog1.PrinterSettings = New PrinterSettings
    If PrintDialog1.ShowDialog() = DialogResult.OK Then
        PrintDocument1.PrinterSettings = PrintDialog1.PrinterSettings
        Try
            PrintDocument1.Print()
        Catch ex As Exception
            MessageBox.Show("Error de impresión", "Imprimir imagen", _
                            MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
        End Try
    End If
End Sub

Private Sub PrintDocument1_PrintPage(ByVal sender As System.Object, _
                           ByVal e As System.Drawing.Printing.PrintPageEventArgs) _
                           Handles PrintDocument1.PrintPage
    e.Graphics.DrawImage(PictureBox1.Image, e.MarginBounds.Left, e.MarginBounds.Top, _
                                        e.MarginBounds.Width, PictureBox1.Image.Height)
End Sub
```

Clase DateTimePicker

- Proporciona un mecanismo para la introducción de valores de tipo `DateTime`.
  - Presenta el aspecto de una lista deplegable que se despliega como un calendario.
  - Es posible limitar las fechas a visualizar mediante las propiedades `MaxDate` y `MinDate`.
- Obtener el valor.
  - La propiedad `Text` obtiene o establece una cadena con el valor que aparece en el control.
  - La propiedad `Value` obtiene o establece un valor de tipo `DateTime`.
  - La propiedades `Day`, `Month`, `Year`, `DayOfWeek`, `Hour`, `Minute`, `Second` y `Millisecond` devuelven las partes de la fecha.
- Formato del control.
  - La propiedad `Format` permite establecer el formato de fecha y hora que se visualiza.
  - Es posible mostrar un control para seleccionar horas marcando la propiedad ShowUpDown a True y la propiedad Format a Time.



Clase MonthCalendar

- Muestra una interfaz gráfica en forma de calendario mediante la que el usuario puede manejar información relativa a fechas.
  - El número de meses que aparece se puede modificar mediante la propiedad `CalendarDimensions`.
- Permite obtener o establecer rangos de fechas.
  - La propiedad `MaxSelectionCount` permite determinar el número máximo de días seleccionados.
  - La propiedades `SelectionStart` y `SelectionEnd` obtienen o establecen la fecha de inicio y fin.
  - La propiedad `SelectionRange` establece o devuelve un objeto de tipo `SelectionRange` que contiene dos fechas con el inicio y el fin del periodo.

```
'Selecciona el día actual y los tres siguientes
'y muestra las fechas de inicio y fin en etiquetas
MonthCalendar1.SelectionRange = _
      New SelectionRange(Now(), Now().AddDays(3))
Label1.Text = MonthCalendar1.SelectionRange.Start
Label2.Text = MonthCalendar1.SelectionRange.End
```

Clase Timer

- Implementa un temporizador que produce un evento en los intervalos fijados por el programador.
- El evento `Tick`, se producirá cada vez que se cumple el intervalo previsto por el programador y el control está activado.
- La propiedad `Interval` permite fijar en milisegundos el intervalo de tiempo.
- La propiedad `Enabled` admite un valor lógico que permite activar o desactivar el temporizador.

```
'Muestra en Label1 un reloj que se actualiza cada milisegundo
Private Sub Timer1_Tick(ByVal sender As Object, ByVal e As System.EventArgs) _
                            Handles Timer1.Tick
        'Label1 muestra la hora del sistema actualizada
        'La propiedad Interval se debe establecer a 1000
        'La propiedad Enabled se debe establecer a True
        Label1.Text = DateTime.Now.ToString("hh:mm:ss")
End Sub
```

Clase ToolTip

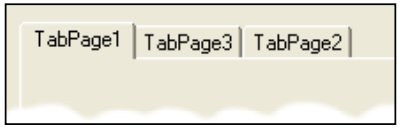
- Muestra texto de ayuda cuando el curso se para sobre el control.
- El control `ToolTip` se coloca en la bandeja de componentes.
  - Se puede usar un único control `ToolTip` para todos los componentes del formulario.
    - ✓ En tiempo de diseño, para asociar un control con un `ToolTip`, en la ventana de propiedades **del control al que se quiere añadir el texto** se introduce el texto en "ToolTip en ToolTip1".
    - ✓ En tiempo de ejecución, se puede hacer mediante el método SetToolTip.

```
ToolTip1.SetToolTip(Button1, "Guardar cambios")
```

- Propiedades que controlan el tiempo de retardo del control.
  - `InitialDelay`, tiempo en milisegundos que el usuario debe apuntar al control asociado para que aparezca la información del control.
  - `ReshowDelay`, tiempo que tarda en aparecer el texto cuando el ratón de mueve desde un control asociado a otro.
  - `AutoPopDelay`, tiempo durante el cual se muestra la información del control asociado.
  - `AutomaticDelay`, permite establecer las demás propiedades en función del valor asignado a esta propiedad:
    - ✓ Si `AutomaticDelay` tiene el valor N…
      - o `InitialDelay` se establece en N, `ReshowDelay` se establece en N/5 y `AutoPopDelay` se establece en 5N.

Clase TabControl

- Muestra un formulario con múltiples fichas similares a las pestañas de las carpetas.

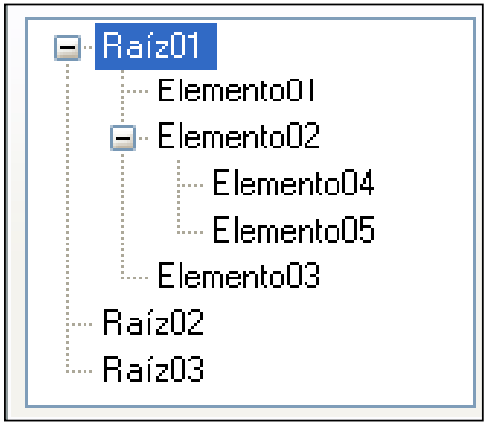


  - Cada ficha puede tener varios controles.
  - Se utilizan para cuadros de diálogo con varias páginas (por ejemplo para páginas de propiedades).
- La propiedad `TabPages` hace referencia a una colección con las fichas.
  - Cada ficha es un objeto de tipo `TabPages`.
    - ✓ Cada objeto `TabPage` es un contenedor de otros controles.
  - Se pueden añadir en tiempo de diseño o en tiempo de ejecución con el método `Add`.

```
TabControl1.TabPages.Add(New TabPage("Otra ficha"))
```

Clase TreeView

- Muestra un conjunto de elementos organizado de forma jerárquica.
- Cada elemento es un objeto de la clase `TreeNode` a la que se accede por medio de la propiedad `Nodes` del elemento.
  - La propiedad `Nodes` del control `TreeView` contendría una colección de nodos de tipo nodo raíz.
  - Cada nodo sería un objeto de la clase `TreeNode` que a su vez tiene una colección de nodos representada también por su propiedad `Nodes`.
    - ✓ Cada nodo permite tener asociado un icono con la propiedad `ImageKey`.
      - `ImageKey` es un número que hacer referencia a una imagen almacenada en un control `ImageList` al que se referencia por la propiedad `ImageList`.



Colección Nodes de TreeView:

- Raíz01, Raíz02 y Raíz03

Colección Nodes de Raíz01:

- Elemento01, Elemento02, Elemento03.

Colección Nodes de Elemento02:

- Elemento04, Elemento05.

Clase TreeView (II)

- Añadir nodos mediante programación.
  - El método `Add` de la colección `Nodes` permite añadir nodos.
    - ✓ Recibe como argumento una cadena con el nombre de la etiqueta o un objeto de la clase TreeNode.
      - Añade un nodo raíz al objeto TreeView.

```
TreeView1.SelectedNode.Nodes.Add("Nuevo nodo")
```

      - Añade un nodo al nodo seleccionado.

```
Dim miNodo As New TreeNode("Nuevo nodo")
'La propiedad SelectedNode referencia al nodo seleccionado
TreeView1.SelectedNode.Nodes.Add(miNodo)
```

- Eliminar nodos mediante programación.
  - El método `Remove` de la clase `TreeNode` elimina el nodo y sus nodos hijos.
- Contenido de un nodo.
  - La propiedad `Text` de la clase `TreeView` muestra la etiqueta del nodo seleccionado.
  - La propiedad `Text` de la clase `TreeNode` muestra la etiqueta del nodo.
  - La propiedad `FullPath` de la clase `TreeNode` devuelve una cadena con la ruta de acceso del nodo

Clase TreeView (III)

- Ejemplo: Añadir y eliminar nodos mediante programación.
  - El botón Añadir insertará un nuevo nodo a partir del nodo seleccionado con la etiqueta del cuadro de texto.
    - ✓ Si la casilla de verificación Nodo raíz está activada, lo añadirá como nodo raíz.
  - El botón Eliminar quitará el nodo seleccionado y todos sus hijos.
    - ✓ Si el nodo tiene hijos, un cuadro de mensaje informará de la incidencia y permitirá cancelar la operación.

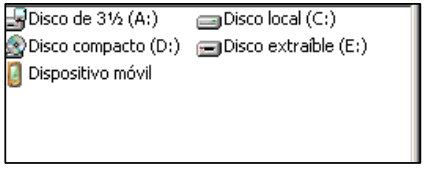
Clase TreeView (IV)

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                         ByVal e As System.EventArgs) Handles Button1.Click
  If (Not CheckBox1.Checked) And (Not TreeView1.SelectedNode Is Nothing) Then
      TreeView1.SelectedNode.Nodes.Add(TextBox1.Text)
  Else
      TreeView1.Nodes.Add(TextBox1.Text)
  End If
End Sub
Private Sub CheckBox1_CheckedChanged(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles CheckBox1.CheckedChanged
  If CheckBox1.Checked Then
      TreeView1.SelectedNode = Nothing
  End If
End Sub
Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
    Handles Button2.Click
  If TreeView1.SelectedNode.GetNodeCount(True) <> 0 Then
      If MessageBox.Show("El nodo tiene hijos ¿Desea continuar?", _
                  "Eliminar un nodo",MessageBoxButtons.YesNo, MessageBoxIcon.Question, _
                  MessageBoxDefaultButton.Button2)=Windows.Forms.DialogResult.Yes Then
          TreeView1.SelectedNode.Remove()
      End If
  Else
      TreeView1.SelectedNode.Remove()
  End If
End Sub
```

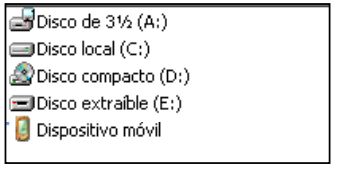
Clase ListView

- Muestra una lista de elementos con iconos o encolumnado.
- Puede mostrar cuatro vistas mediante los valores de la propiedad `View`:
  - `LargeIcon`.
  - `SmallIcon`.
  - `List`.
  - `Details`.

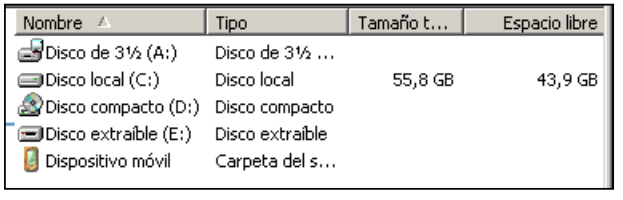
`LargeIcon`



`SmallIcon`



`List`



`Details`

Clase ListView (II)

- Agregar elementos.
  - Cada elemento es un dato de tipo `ListViewItem`.
  - Se pueden agregar en tiempo de diseño mediante el editor de la propiedad `Items` de la ventana de propiedades.
    - ✓ `Items` hace referencia a una colección de `ListViewItems`.
  - En tiempo de ejecución se añaden con el método `Add` de la colección `ListViewItemsCollection`.

```
'Añade un elemento con la etiqueta "Elemento 1"
ListView1.Items.Add("Elemento 1")
‘Añade un nuevo elemento miItem
Dim miItem As New ListViewItem("Elemento 1")
ListView1.Items.Add(miItem)
'Añade un nuevo elemento "Elemento 1",
'con el primer icono de la lista de imágenes.
ListView1.Items.Add("Elemento 2", 0)
```

Clase ListView (III)

- Agregar elementos en columnas.
  - El formato tabular sólo está disponible con la propiedad `View` establecida a `Details`.
  - Es necesario crear las columnas mediante el editor de columnas al que se accede mediante la propiedad Columns.
  - El contenido de la primera columna corresponde al elemento `ListViewItem`.
    - ✓ Cada columna siguiente es un elemento de la colección `SubItems` de la clase `ListViewItem`.
      - Se puede contruir un `ListViewItem` con sus elementos a partir de un array de cadenas.

```
Dim items() As String = New String() { _
        TextBox1.Text, TextBox2.Text, TextBox3.Text}
ListView1.Items.Add(New ListViewItem(items))
```

Clase ListView (IV)

- Referencia a los elementos.
  - Propiedad `SelectedIndices`. Devuelve una colección con los índices seleccionados.
    - ✓ Hay que tener en cuenta que se pueden seleccionar varios elementos.
      - `ListView1.SelectedIndices(0).Item`, devuelve un entero con el índice el primer elemento seleccionado.
  - Propiedad `SelectedItems`. Devuelve una colección de `ListViewItem` con los elementos seleccionados.
    - `ListView1.SelectedItems(0).Text`, devuelve la etiqueta del primer elemento seleccionado.
  - Propiedad `FocusedItem`. Devuelve el `ListViewItem` que ha recibido el foco.
- Referencia a los subelementos.
  - Se realiza a partir de la colección `SubItems` del elemento.
    - ✓ `ListView1.SelectedItems(0).SubItems(0).Text`, devuelve la etiqueta de l a primer subelemento de la fila seleccionada.

Clase ListView (V)

- Control del elemento seleccionado.
  - Evento `ItemActivate`. Se produce cuando se activa un elemento.
  - Evento `SelectedIndexChanged`. Se produce cuando se cambia el índice del elemento seleccionado.
    - ✓ Se produce antes de ItemActivate.
- Eliminación de elementos.
  - Método `RemoveAt` de la colección `ListViewItemsCollection`, indicándole el índice del elemento a borrar.

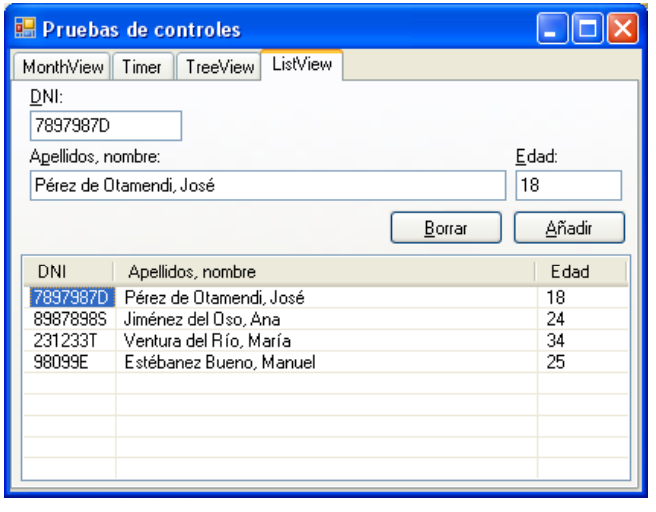
```
ListView1.Items.RemoveAt(ListView1.SelectedIndices(0))
```

  - Método `Remove` de la colección `ListViewItemsCollection`, indicándole el elemento a borrar.

```
ListView1.Items.Remove(ListView1.FocusedItem)
```

Clase ListView (VI)

- Ejemplo: Almacenar datos de personas en un control `ListView`.
  - Al pulsar el botón Añadir, se almacenarán los datos de los cuadros de texto en el `ListView`.
  - Al pulsar el botón Borrar, se eliminará el elemento seleccionado.
  - Al pulsar sobre un elemento del `ListView` aparecerán sus datos en los cuadros de texto.



Clase ListView (VII)

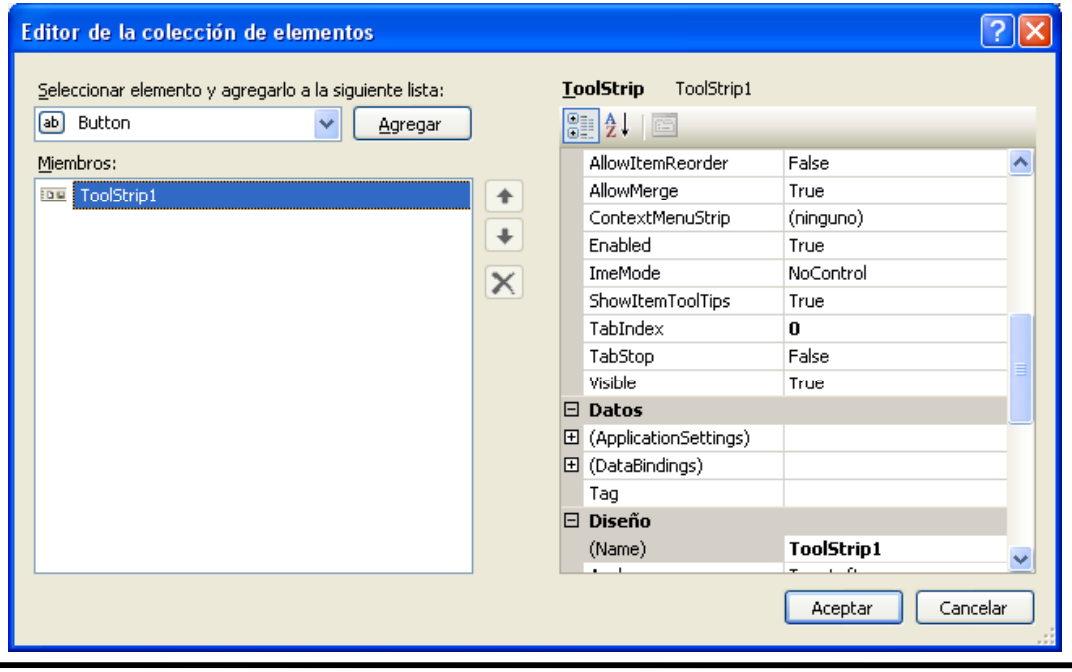
```
Private Sub Button1_Click(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles Button1.Click
  Dim items() As String = New String() { _
                TextBox1.Text, TextBox2.Text, TextBox3.Text}
  ListView1.Items.Add(New ListViewItem(items))
End Sub

Private Sub Button2_Click(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles Button2.Click
   ListView1.Items.Remove(ListView1.FocusedItem)
End Sub

Private Sub ListView1_SelectedIndexChanged(ByVal sender As System.Object, _
              ByVal e As System.EventArgs) Handles ListView1.ItemActivate
  Dim item As ListViewItem
  item = ListView1.SelectedItems(0)
  TextBox1.Text = item.Text
  TextBox2.Text = item.SubItems(1).Text
  TextBox3.Text = item.SubItems(2).Text
End Sub
```

Clase ImageList

- Se utiliza como contenedor de imágenes que podrán ser utilizadas por otros controles a partir del índice de cada imagen.
- Se puede hacer referencia a un `ImageList` en los controles `ListView`, `TreeView`, `TabControl`, `Button`, `CheckBox`, `RadioButton` y `Label`.
  - En el control `ListView` la referencia a la lista de imágenes se hace mediante las propiedades `LargeImageList` y `SmallImageList`.
  - En el resto, a partir de la propiedad `ImageList`.
- La imágenes se gestionan en tiempo de diseño mediante el "Editor de la colección Images" accesible por la propiedad `Images` del `ImageList`.
- La referencia a una imagen concreta de la selección se hace en cada control por medio de la propiedad `ImageIndex`.

Clase ToolStrip

- Crea una barra de herramientas que puede contener botones, cuadros de texto, botones desplegables, ComboBox, etiquetas o separadores.
  - Para agregar elementos en tiempo de diseño a la barra utiliza la colección `Items` de la ventana de propiedades.



Clase ToolStrip (II)

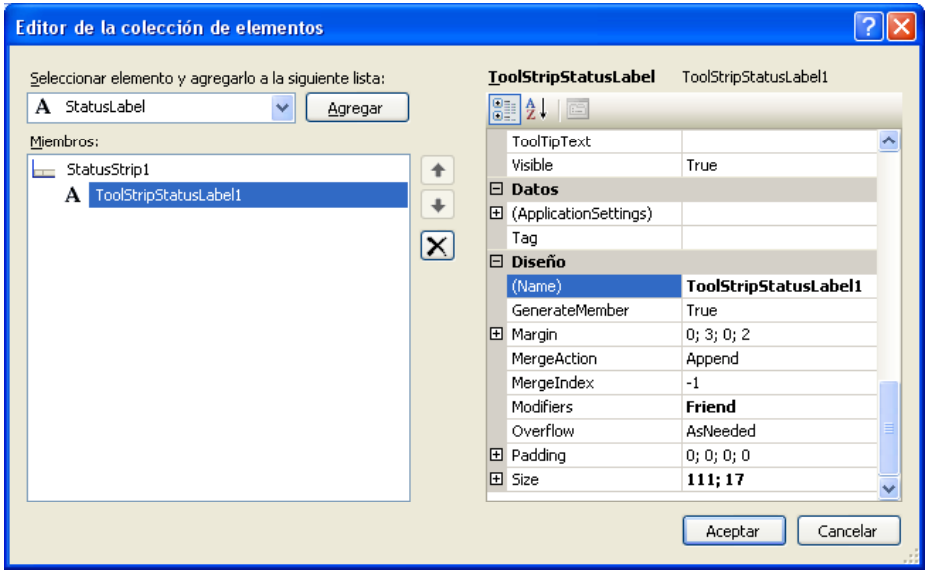
- Elementos de `ToolStrip`.
  - `Button` (clase `ToolStripButton`).
    - ✓ Mediante la propiedad `DisplayStyle` podemos asociarle una imagen (propiedad `Image`), un text (propiedad `Text`) o imagen y texto.
    - ✓ La propiedad `ToolTipText` permite asociarle un texto con información de la herramienta.
    - ✓ El evento `Click` del control `ToolStripButton` permite controlar su comportamiento.
  - `Label` (clase `ToolStripLabel`).
    - ✓ Representa texto y/o imágenes no seleccionables.
    - ✓ Presenta características similares al control `Label`.
  - `SplitButton` (clase `ToolStripSplitButon`).
    - ✓ Combina un botón con un menú desplegable con las mismas posibilidades que la clase `ToolStripMenuItem`, con las características ya apuntadas en el apartado de menús.
    - ✓ Registra las acciones, tanto al pulsar sobre el botón, como al desplegar la lista y seleccionar las opciones.

Clase ToolStrip (III)

- Elementos de `ToolStrip` (*continuación*).
  - `DropDownButton` (clase `ToolStripDropDownButon`).
    - ✓ Al pulsarlo muestra un menú desplegable.
      - El Click del botón sólo despliega el menú.
  - `Separator` (clase `ToolStripSeparator`).
  - `ComboBox` (clase ToolStripComboBox).
    - ✓ Representa un `ComboBox`, con características similares.
  - `TextBox` (clase `ToolStripTextBox`).
    - ✓ Representa un `TextBox` con características similares.
  - `ProgressBar` (clase `ToolStripProgressBar`).
    - ✓ Representa una barra de progreso.
      - Las propiedades `Minimum` y `Maximum` representan los valores mínimos de la barra.
      - La propiedad `Value` representa el valor actual de la barra.
      - La propiedad `Step` representa el valor con el que se incrementará la barra de progreso al ejecutar el método `PerformStep`.
      - El método `Increment`, permite incrementar la propiedad `Value` al margen del valor que tenga `Step`.

Control StatusStrip

- Barra de estado que muestra información sobre los objetos que se visualizan en un formulario o de las acciones que se están ejecutando.
- Normalmente está compuesto de objetos `ToolStripStatusLabel`, aunque también puede mostrar `ToolStripDropDownButton`, `ToolStripSplitButton` y `ToolStripProgessBar`.
- Para añadir estos controles se utiliza el editor de la colección de elemento StatusStrip.



Control StatusStrip (II)

- Elemento ToolStripStatusLabel.
  - Propiedad Spring.
    - ✓ Determina si la etiqueta ocupa todo el espacio disponible de la barra de estado.
  - Propiedad BorderSides.
    - ✓ Indica que bordes de la etiqueta se van a mostrar.
  - Propiedad BorderStyle.
    - ✓ Tipo de borde que se va a mostrar.
- Ejemplo.
  - Muestra información sobre un botón al pasar el cursor sobre él.

```
Private Sub Button1_MouseHover(ByVal sender As Object, _
                    ByVal e As System.EventArgs) Handles Button1.MouseHover
   'Al pasar el ratón sobre el botón se muestra el texto en la barra de estado
   ToolStripStatusLabel1.Text = "Guardar los cambios"
End Sub
Private Sub Form2_MouseHover(ByVal sender As Object, _
                           ByVal e As System.EventArgs) Handles Me.MouseHover
    'Es necesario para borrar el texto cuando se sale del botón
    ToolStripStatusLabel1.Text = ""
End Sub
```