Sistemas y Aplicaciones Informáticas

Programación en Visual Basic 6.0

1. Fundamentos de programación.

1.1. Formularios.

1.1.1. Relación entre formularios y controles.

- Los elementos de desarrollo básicos que se usan para crear la interfaz son los siguientes:
  - *Formularios*. Son objetos que exponen las propiedades que definen su apariencia, los métodos que definen su comportamiento y los eventos que definen la forma en que interactúan con el usuario.
  - *Controles*. Son objetos que están contenidos en los objetos de formularios. Cada tipo de control tiene su propio conjunto de propiedades, métodos y eventos, que lo hacen adecuado para una finalidad determinada.
- Si bien los controles son objetos independientes, existe una cierta *relación primaria y secundaria* entre los formularios y los controles. Todos los controles son secundarios del formulario en el que se dibujan. Además la ubicación de los controles es relativa al formulario primario. Mover un formulario (*frm*) mueve también los controles, y no se pueden mover los controles fuera de los límites del formulario primario.

1.1.2. Propiedades, eventos y métodos de un formulario.

1.1.2.1. Propiedades.

- **Caption** [= *cadena*]. Determina el texto que muestra la barra de título del formulario.
- **Icon** [= *icono*]. Establece el icono que aparece cuando se minimiza un formulario. El archivo debe tener la extensión .ico y el formato correspondiente.
- **MaxButton** [= **True** | **False**]. Determina si se puede maximizar (**True**) o no (**False**).
- **MinButton** [= **True** | **False**]. Determina si se puede minimizar (**True**) o no (**False**).
- **ControlBox** [= **True** | **False**]. Muestra o no un cuadro del menú **Control** en el formulario.
- **BorderStyle** = [*valor*]. Controla el estilo del borde del formulario. Los valores admitidos son:
  - **vbBSNone**. Ningún borde ni elemento relacionado con él.
  - **vbFixedSingle**. Simple fijo. Puede incluir un cuadro del menú **Control**, una barra de título, un botón **Maximizar** y un botón **Minimizar**. Sólo cambia el tamaño con estos últimos.
  - **vbSizable**. Predeterminado, tamaño ajustable y botones **Maximizar** y **Minimizar**.
  - **vbFixedDouble**. Diálogo fijo. Puede incluir un cuadro del menú **Control** y una barra de título, pero no los botones **Maximizar** ni **Minimizar**. No se puede cambiar de tamaño.
- **Height** [= *número*] y **Width** [= *número*]. Determinan el alto y ancho externos respectivamente del formulario en twips, incluyendo los bordes y la barra de título.
- **Left** [= *valor*] y **Top** [= *valor*]. Determinan la ubicación del formulario en twips con relación a la esquina superior izquierda de la pantalla.
- **WindowState** [**=** *valor*]. Establece si el formulario se inicia en estado normal (**vbNormal** predeterminado), maximizado (**vbMaximized**) o minimizado (**vbMinimized**).
- **Picture** [= *imagen*]. Muestra una imagen directamente sobre el fondo del formulario.
- **Visible** [= **True** | **False**]. Determina si el formulario es visible (**True**) o está oculto (**False**).

- **Enabled** [= **True** | **False**]. Devuelve o establece un valor que determina si un formulario puede responder a eventos generados por el usuario (**True**) o no responde (**False**).
- **MousePointer** [= *valor*]. Devuelve o establece un valor que indica el tipo de puntero de *mouse* que aparece cuando éste se sitúa sobre un área determinada del formulario. El puntero seleccionado aparece en las áreas en blanco del formulario y sobre los controles con la propiedad **MousePointer** establecida a **vbDefault**. Algunos de los valores admitidos son **vbDefault**, **vbHourglass**, **vbSizePointer**, **vbNoDrop**, **vbArrow** y **vbCrosshair**.
- **MouseIcon** [= *icono*]. Establece un icono o un cursor personalizado. El archivo debe tener la extensión .ico o .cur y el formato correspondiente. Para poder utilizar este puntero personalizado, es necesario que **MousePointer** esté establecido a **vbCustom**.
- **KeyPreview** [= **True** | **False**]. Devuelve o establece un valor que determina si el formulario recibe todos los eventos del teclado del control que tiene el enfoque antes de que el control los reconozca (**True**) o no (**False** predeterminado).
- **Font**. Devuelve un objeto **Font** asociado al formulario y cuyas propiedades contienen la información necesaria para dar formato al texto. Estas propiedades son:
  - **Name** [= *cadena*]. Devuelve o establece el nombre de la fuente.
  - **Bold** [= **True** | **False**]. Activa o desactiva el formato de negrita.
  - **Italic** [= **True** | **False**]. Activa o desactiva el formato de cursiva.
  - **Underline** [= **True** | **False**]. Activa o desactiva el formato de subrayado.
  - **StrikeThrough** [= **True** | **False**]. Activa o desactiva el formato de tachado.
  - **Size** [= *número*]. Devuelve o establece el tamaño de fuente en puntos.
- **FontTransparent** [= **True** | **False**]. Determina si el texto y los gráficos de fondo se muestran a través del texto presentado en el formulario (**True** predeterminado) o no están ocultos (**False**).
- **ActiveForm**. Devuelve el formulario activo, es decir, aquel que contiene el control que tiene el enfoque. Se puede usar para tener acceso a las propiedades de un formulario o para invocar sus métodos. Si se desea pasar a un procedimiento, se debe declarar el argumento en ese procedimiento con la cláusula `As Form`.
- **ActiveControl**. Especifica el control que tendría el enfoque si el formulario al que se hace referencia estuviera activo. Cada formulario puede tener un control activo independientemente de que el formulario esté activo o no. Se puede usar para tener acceso a las propiedades de un control o para invocar sus métodos. Si todos los controles del formulario son invisibles o están desactivados, se producirá un error en tiempo de ejecución. Si se desea pasar a un procedimiento, se debe declarar el argumento en ese procedimiento con la cláusula `As Control`.

1.1.2.2. Eventos.

1.1.2.2.1. Carga de formularios.

1. **Initialize( )**. Se produce cuando una aplicación crea una instancia de un formulario. Se utiliza para inicializar las variables definidas a nivel de módulo en la instancia del formulario.
2. **Load( )**. Ocurre al cargar el formulario principal al arrancar la ejecución de un programa; con el resto de los formularios como resultado de una instrucción **Load** o al hacer referencia a alguna

propiedad o control de un formulario descargado. Se utiliza para dar valor a sus propiedades y a las de los controles que dependen de dicho formulario. El formulario no es visible.

3. **Activate( )**. Se produce siempre que se convierte en el formulario activo mediante una acción del usuario (**Click**) o mediante el método **Show**. Sólo ocurre cuando el formulario es visible.
4. **Paint( )**. Sucede cuando se hace visible por primera vez, y también cuando el formulario entero o una parte del mismo se expone después de haberse movido o ampliado, o después de haberse movido una ventana que lo estaba cubriendo. Es el lugar ideal para colocar los métodos gráficos de un formulario cuando la propiedad **AutoRedraw** del formulario es **False**.

1.1.2.2.2. Descarga de formularios.

1. **Deactivate( )**. Se produce cuando un formulario deja de ser activo dentro de una aplicación. Con este evento se podría guardar los cambios efectuados en un archivo o en una base de datos.
2. **QueryUnload(***Cancel* **As Integer**, *UnloadMode* **As Integer)**. Ocurre en todos los formularios antes de que se cierre un formulario concreto o una aplicación. Se utiliza para asegurarse de que no hay tareas sin finalizar antes de que esa aplicación se cierre. Si *Cancel* se establece a cualquier valor distinto de cero (**True**), se detiene el cierre del formulario y de la aplicación. *UnloadMode* puede tener los siguientes valores:
   - **vbFormControlMenu**. Se eligió el comando **Cerrar** del menú **Control** del formulario.
   - **vbFormCode**. Se invocó la instrucción **Unload** desde el código.
   - **vbAppWindows**. La sesión actual de Windows está finalizando.
   - **vbAppTaskManager**. El Administrador de tareas de Windows está cerrando la aplicación.
   - **vbFormMDIForm**. Un formulario MDI secundario se cierra porque el primario se cierra.
   - **vbFormOwner**. Un formulario se cierra porque su formulario propietario se está cerrando.
3. **Unload(***Cancel* **As Integer)**. Ocurre al descargarse un formulario. Este evento se desencadena porque un usuario cierra el formulario mediante el comando **Cerrar** del menú **Control** o una instrucción **Unload**. Si *Cancel* se establece a cualquier valor distinto de cero (**True**), se detiene el cierre del formulario, pero no detiene los demás eventos. Se utiliza para especificar acciones que deben tener lugar cuando se descargue el formulario.
4. **Terminate( )**. Se produce cuando todas las referencias a una instancia de un formulario se quitan de la memoria estableciendo a **Nothing** todas las variables que hacen referencia al objeto. Si la aplicación invoca la instrucción **End** el evento Terminate no se desencadenará.

1.1.2.2.3. Otros eventos.

- **Resize( )**. Se desencadena siempre que se cambia el tamaño de un formulario, ya sea por una acción del usuario o a través del código. Esto permite mover o cambiar el tamaño de los controles del formulario. Si la propiedad **AutoRedraw** está a **False** y el formulario cambia de tamaño, se llama a los eventos **Resize(.)** y **Paint( )**, en este orden.
- **GotFocus( )**. Se produce cuando el formulario recibe el enfoque. En ese momento tiene la capacidad de recibir clics del *mouse* en cualquier momento. Un formulario sólo puede recibir el enfoque si sus propiedades **Enabled** y **Visible** tienen el valor **True**, y sólo cuando todos los controles visibles que contiene están desactivados.

- **LostFocus( )**. Se produce cuando el formulario pierde el enfoque. Se usa para validar actualizaciones, o para modificar las condiciones establecidas en el evento **GotFocus(.)**.
- **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en un área en blanco del formulario o en un control desactivado.
- **DblClick( )**. Ocurre cuando el presiona y suelta dos veces el botón primario del *mouse* en un área en blanco del formulario o en un control desactivado. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.

1.1.2.3. Métodos.

- **Show** *estilo*. Para hacer visible un formulario. Si está cargado genera los eventos **Activate( )** y **Paint( )**. Si no está cargado genera los cuatro eventos de carga. Es equivalente a asignar el valor **True** a la propiedad **Visible** del formulario. El argumento *estilo* puede tener dos valores:
  - **vbModal**. Un formulario *modal* se debe cerrar (ocultar o descargar) antes de poder continuar trabajando con el resto de la aplicación. El código que sigue al método **Show** no se ejecutará hasta que se cierre el formulario.
  - **vbModeless**. Predeterminado, un formulario *no modal* permite continuar trabajando con el resto de la aplicación sin tener que cerrar el formulario. El código que sigue al método **Show** se ejecuta inmediatamente después de presentar el formulario.
- **Hide**. Para ocultar un formulario sin descargarlo. Sus variables y propiedades siguen estando accesibles y conservando sus valores. Si el formulario no está cargado, **Hide** carga el formulario pero no lo presenta. Es equivalente a asignar el valor **False** a la propiedad **Visible** del formulario.
- **SetFocus**. Mueve el enfoque al formulario especificado. Un formulario sólo puede recibir el enfoque si sus propiedades **Enabled** y **Visible** tienen el valor **True**, y sólo cuando todos los controles visibles que contiene están desactivados. Para utilizar este método en el propio procedimiento **Form_Load( )**, primero debe mostrarse el formulario con el método **Show**.
- **PopupMenu** *nombreMenú, indicadores, x, y*. Presenta un menú emergente en la posición actual del *mouse* o en las coordenadas especificadas en un formulario, donde *nombreMenú* es el nombre del menú emergente, *x* e *y* son las coordenadas donde se presenta el menú emergente (si se omiten, se usan las coordenadas del *mouse*) e *indicadores* puede tomar los siguientes valores:
  - *Ubicación del menú emergente*:
    - **vbPopupMenuLeftAlign**. Predeterminado, el lado izquierdo del menú se sitúa en *x*.
    - **vbPopupMenuCenterAlign**. El menú emergente se centra en *x*.
    - **vbPopupMenuRightAlign**. El lado derecho del menú emergente se sitúa en *x*.
  - *Comportamiento del menú emergente*:
    - **vbPopupMenuLeftButton**. Predeterminado, los elementos del menú emergente sólo reaccionan a los clics del *mouse* cuando se usa el botón primario del *mouse*.
    - **vbPopupMenuRightButton**. Los elementos del menú emergente reaccionan a los clics del *mouse* cuando se usan los botones primario o secundario.

  Para especificar un indicador, combine una constante de cada grupo con el operador **Or**. Como estándar, se utiliza el evento **MouseUp** para detectar el clic el usuario con el botón secundario

del *mouse* para presentar el menú emergente. Todo el código que siga a una llamada al método **PopupMenu** no se ejecutará hasta que el usuario seleccione un elemento del menú o lo cancele.

- **Move** *izquierda*, *superior*, *ancho*, *alto*. Mueve un formulario por la pantalla. Los argumentos *izquierda* y *superior* determinan la ubicación del formulario en twips con relación a la esquina superior izquierda de la pantalla. Los argumentos *ancho* y *alto* indican los nuevos alto y ancho externos respectivamente del formulario en twips, incluyendo los bordes y la barra de título.
- **Refresh**. Vuelve a dibujar completamente el formulario mientras se carga otro formulario.

1.2. Módulos de código.

- Un proyecto de Visual Basic consta de:
  - *Un archivo de proyecto (.vbp).* Es una lista de todos los archivos y objetos asociados con el proyecto, así como información acerca de las opciones de entorno establecidas.
  - *Un archivo para cada formulario (.frm).* Pueden contener texto descriptivo del formulario y sus controles, incluyendo los valores de sus propiedades. También pueden contener declaraciones de formulario de constantes, variables y procedimientos externos, así como procedimientos de evento y procedimientos generales.
  - *Un archivo de datos binarios para cada formulario (.frx).* Contiene datos sobre propiedades de controles del formulario. Estos archivos no se pueden modificar y los genera automáticamente cualquier archivo .frm que tenga propiedades en formato binario.
  - *Opcionalmente, un archivo para cada módulo de clase (.cls).* Son similares a los módulos de formulario, excepto en que no tienen interfaz de usuario visible. Puede usar módulos de clase para crear sus propios objetos, incluyendo código para métodos y propiedades.
  - *Opcionalmente, un archivo para cada módulo estándar (.bas).* Pueden contener declaraciones públicas o a nivel de módulo de tipos, constantes, variables, procedimientos externos y procedimientos públicos.
  - *Opcionalmente, uno o más archivos con controles ActiveX (.ocx).* Son controles opcionales que se pueden agregar al cuadro de herramientas y se pueden usar en formularios.
  - *Opcionalmente, un único archivo de recursos (.res).* Contienen mapas de bits, cadenas de texto y otros datos específicos que pueda cambiar entre versiones traducidas o entre configuraciones específicas. Un proyecto sólo puede contener un archivo de recursos.
- Todos los archivos y objetos también se pueden compartir con otros proyectos. Un archivo concreto, como un formulario, puede formar parte de varios proyectos. Cuando se agrega un archivo a un proyecto, simplemente se está incluyendo en el proyecto una referencia al archivo existente, no agregando una copia de un archivo. Por tanto, si efectúa cambios en un archivo y lo guarda, los cambios afectarán a cualquier proyecto que contenga dicho archivo.

1.3. Conceptos generales de código.

1.3.1. Generalidades.

- El símbolo de comentario (') indica que las palabras que van a continuación son comentarios. Éstos pueden seguir a una instrucción en la misma línea o pueden ocupar una línea completa.

- Se puede dividir una instrucción larga en varias líneas si se utiliza el *carácter de continuación de línea*, formado por un espacio en blanco seguido de un signo de subrayado (_). No puede poner un comentario después de un carácter de continuación de línea en la misma línea.
- Normalmente hay una instrucción por línea y no hay ningún terminador de instrucción. Se puede colocar dos o más instrucciones en una línea si utiliza un signo de dos puntos (:) para separarlas.
- En ocasiones es conveniente usar números hexadecimales (base 16) o números octales (base 8). Visual Basic representa los números hexadecimales con el prefijo &H y los octales con &O.
- Las constantes literales de tipo **String** deben ir encerradas entre comillas dobles (""). Algunas veces aparecen comillas en las cadenas de texto. En este caso debe insertar un par adicional de comillas por cada par que desee presentar en una cadena de texto, o sustituir las comillas del texto por *Chr(34)* y concatenar las expresiones utilizando el operador &.
- Los nombres de procedimientos, variables y constantes deben seguir estas directrices:
  - Deben comenzar por una letra. No pueden ser iguales que las palabras clave restringidas.
  - No se admiten espacios o caracteres en blanco, puntos, ni caracteres especiales de declaración de tipos (%, &, #, !, @, $). No se distingue entre minúsculas y mayúsculas.
  - No pueden superar los 255 caracteres. Los nombres de controles, formularios, clases y módulos no deben exceder los 40 caracteres.
- Una tecla de acceso es una tecla presionada mientras se mantiene presionada la tecla ALT que permite al usuario seleccionar un control, desencadenando el evento **Click**. Para asignar una tecla de acceso al control, en la propiedad **Caption** incluya un signo & inmediatamente delante del carácter que desea designar como tecla de acceso. Para incluir un signo & en un título sin crear una tecla de acceso, incluya dos signos (&&) y no habrá caracteres subrayados.
- Para establecer varias propiedades de un mismo objeto se utiliza la instrucción **With...End With**, que pueden anidarse entre sí. Por ejemplo:

```
With Command1
      .Caption = "Aceptar"
      .Visible = True
      .Top = 200
      .Left = 5000
      .Enabled = True
End With
```

1.3.2. Carga y descarga de formularios.

- De forma predeterminada, el primer formulario de la aplicación es el *formulario inicial*. Cuando la aplicación inicia la ejecución, el primer código que se ejecuta es el del evento Form_Initialize de dicho formulario. Si quiere cambiar el formulario inicial, seleccione el formulario que desee en el cuadro **Objeto inicial** de la ficha **General** de las **Propiedades del proyecto**.
- Puede hacer que la aplicación se inicie sin cargar ningún formulario creando un procedimiento llamado **Main( )** en un módulo estándar. Este procedimiento tiene que ser un procedimiento **Sub** y no puede estar en un módulo de formulario. Para ello, seleccione **Sub Main** en el cuadro **Objeto inicial** de la ficha **General** de las **Propiedades del proyecto**.

- Para cerrar una aplicación es necesario descargar el formulario principal con la instrucción **Unload Me**. La palabra reservada **Me** siempre hace referencia al formulario actual. Si tiene más de un formulario, puede usar la colección **Forms** y la instrucción **Unload**.
- La instrucción **End** termina una aplicación inmediatamente. Después de esta instrucción no se ejecutan los procedimientos de evento QueryUnload, Unload o Terminate de ningún formulario.
- Existen varias acciones de presentación de formularios:
  - *Cargar un formulario en memoria, pero no presentarlo*. Utilice la instrucción **Load** o haga referencia a una propiedad o control del formulario.
  - *Cargar y presentar un formulario no modal*. Utilice el método **Show**.
  - *Cargar y presentar un formulario modal*. Utilice el método **Show** con *estilo* = **vbModal**.
  - *Mostrar un formulario previamente cargado en memoria*. Utilice el método **Show** o establezca su propiedad **Visible** a **True**.
  - *Ocultar un formulario*. Utilice el método **Hide** o establezca su propiedad **Visible** a **False**.
  - *Descargar un formulario de memoria*. Utilice la instrucción **Unload**.

1.4. Variables, constantes y tipos de datos.

1.4.1. Variables.

1.4.1.1. Descripción. Variables estáticas.

- Una variable se declara mediante la instrucción **Dim**, proporcionando un nombre a la variable:

  Dim *nombreVariable1* [**As** *tipo1*] [, *nombreVariable2* [**As** *tipo2*]]

  Las variables que se declaran en un procedimiento con **Dim** sólo existen mientras se ejecuta el procedimiento. Cuando termina el procedimiento, desaparece el valor de la variable.
- Para declara una variable estática en un procedimiento, se utiliza la palabra reservada **Static**:

  Static *nombreVariable* [**As** *tipo*]

  La primera vez que se ejecuta el procedimiento la variable se inicializa al valor predeterminado según su tipo de dato, y en sucesivas ejecuciones se mantiene el valor asignado por última vez. Si incluye la palabra reservada **Static** al principio del encabezado del procedimiento, todas las variables del procedimiento sean estáticas sin tener en cuenta si se declararon con **Static**, **Dim**, **Private** o de manera implícita dentro del procedimiento.
- La cláusula opcional **As** *tipo* permite definir el tipo de dato o la clase de la variable que va a declarar. Las variables se pueden declarar:
  - *Dentro de un procedimiento de un módulo de formulario (.frm).* Su ámbito está restringido al procedimiento en el que están declaradas.
  - *En la sección Declaraciones de un módulo de formulario (.frm).* Su ámbito está restringido al módulo de formulario en el que están declaradas.
  - *En la sección Declaraciones de un módulo estándar (.bas) o de clase (.cls).* Su ámbito comprende todos los módulos de formulario de la aplicación.
- No es necesario declarar una variable antes de usarla. Para evitar problemas, puede establecer que Visual Basic le avise siempre que encuentre un nombre que no se haya declarado explícitamente como una variable, incluyendo la instrucción `Option Explicit` en la sección Declaraciones de todos los módulos de formulario, estándar o de clase.

1.4.1.2. Accesibilidad de las variables.

- La palabra reservada **Dim** equivale a **Private**, que significa que la variable sólo es accesible desde dentro de su ámbito. Si se desea que una variable sea accesible desde fuera de su ámbito (a excepción de las variables de procedimiento que siempre se declaran con **Dim**), hay que declararla con la palabra reservada **Public**:

  Public *nombreVariable* [**As** *tipo*]

- Por defecto, una variable a nivel de módulo de formulario está disponible para todos los procedimientos del módulo, pero no para otros módulos de formulario. Se declaran con la palabra reservada **Private** en la sección Declaraciones al principio del módulo:

  Private *nombreVariable* [**As** *tipo*]

- A nivel de módulo de formulario, no hay diferencia entre **Private** y **Dim**, pero es preferible **Private** porque contrasta con **Public** y hace que el código sea más fácil de comprender. La accesibilidad de las variables según su lugar y modo de declaración se resume en lo siguiente:
  - Declaración en un procedimiento con **Dim**. Accesible desde el propio procedimiento.
  - Declaración en un módulo de formulario (*.frm*):
    - Con **Public**. Accesible desde cualquier procedimiento del propio formulario, y desde otros formularios precedida del nombre del formulario en el que se ha declarado.
    - Con **Private**. Accesible desde cualquier procedimiento del propio formulario.
  - Declaración en un módulo estándar (*.bas*) o de clase (*.cls*):
    - Con **Public**. Accesible desde todos los formularios de la aplicación.
    - Con **Private**. Accesible desde cualquier procedimiento de ese módulo.

1.4.1.3. Coincidencia en el nombre de variables.

- Si hay variables públicas con el mismo nombre en módulos distintos, es posible diferenciarlas en el código haciendo referencia al módulo en el que están declaradas (`Module1.intx` y `Form1.intx`). Si no se especifica el módulo, se toma el valor de la variable más local.
- También puede tener una variable con el mismo nombre en ámbitos distintos. En general, cuando las variables tienen el mismo nombre pero distinto alcance, la variable más local siempre tiene preferencia de acceso sobre las variables menos locales.

1.4.2. Constantes.

- La sintaxis para declarar una constante es la siguiente:

  [**Public** | **Private**] **Const** *nombreConstante*[**As** *tipo*] = *expresión*

- El argumento *nombreConstante* es un nombre simbólico válido (las reglas son las mismas que para crear nombres de variable) y *expresión* está compuesta por números, cadenas, constantes previamente definidas y operadores numéricos o de cadena. No puede usar llamadas a funciones. Puede colocar más de una declaración de constante en una única línea si las separa con comas:

```
Public Const conPi = 3.14, conMaxPlanetas = 9
```

- **Const** tiene el mismo ámbito y accesibilidad que una declaración de variable:
  - Para crear una constante que sólo exista en un procedimiento, declárela dentro de él.
  - Para crear una constante disponible para todos los procedimientos de un módulo, pero no para el código que está fuera del módulo, declárela en la sección Declaraciones del módulo.

- Para crear una constante disponible en toda la aplicación, declare la constante en la sección Declaraciones de un módulo estándar y coloque delante de **Const** la palabra clave **Public**.
- No se pueden declarar constantes públicas en un módulo de clase o de formulario.

1.4.3. Tipos de datos.

- Todas las variables tienen un tipo de datos que determina los valores que pueden almacenar y las operaciones que se pueden realizar con ellas. De forma predeterminada, si no proporciona un tipo de dato con la cláusula opcional **As** *tipo*, la variable toma el tipo de dato **Variant**.
- Los tipos de datos existentes en Visual Basic son los siguientes:
  - **Boolean**. Puede tomar valores **True** o **False**. El valor predeterminado es **False**.
  - **Byte**. Entero sin signo de 1 byte, puede tomar valores de 0 a 255. Si la variable contiene datos binarios, declárela como una matriz de tipo **Byte**.
  - **Integer**. Entero con signo de 2 bytes, puede tomar valores de -32768 a 32767.
  - **Long**. Entero con signo de 4 bytes, puede tomar valores de -2147483648 a 2147483647.
  - **Single**. Real en coma flotante de 4 bytes, puede tomar valores de -3.40e+38 a 3.40e+38.
  - **Double**. Real en coma flotante de 8 bytes, puede tomar valores de -1.79e+308 a 1.79e+308.
  - **Currency**. Real en coma fija de 8 bytes, puede tomar valores de -9.22e+14 a 9.22e+14. Acepta hasta 4 dígitos a la derecha de la coma decimal y hasta 15 dígitos a la izquierda. Es adecuado para cálculos monetarios, no está sujeto a errores de redondeo de decimales.
  - **String**. Cadena de caracteres de 4 bytes más 1 byte/car hasta 64 KB. Por defecto, es una *cadena de longitud variable*. Especifique una *cadena de longitud fija* con esta sintaxis:

    [**Public | Private | Dim**] *nombreCadena* **As String *** *tamaño*

    En este caso, la variable se rellenará con espacios en blanco hasta completar el tamaño especificado. Si asigna una cadena demasiado larga a la variable, los caracteres se truncarán. Las cadenas de longitud fija se pueden declarar en módulos estándar como **Public** o **Private**. En módulos de clase y formulario, deben declararse como **Private**.
  - **Date**. Fecha y hora de 8 bytes, puede tomar valores desde 01/01/0100 hasta 31/12/9999 y desde 0:00:00 hasta 23:59:59. Si se convierten otros tipos de datos numéricos en **Date**, los valores que hay a la izquierda de la coma decimal representan la información de fecha, y los que hay a la izquierda representan la hora. La medianoche es 0 y el mediodía es 0,5.
  - **Object**. Dirección de memoria de 4 bytes, hace referencia a objetos. Puede asignarse con la instrucción **Set** para referirse a cualquier objeto real reconocido por la aplicación:

    ```
    Dim objDb As Object
    Set objDb = OpenDatabase("c:\Vb5\biblio.mdb")
    ```

    Intente usar clases específicas como **TextBox** en vez de **Control** o **Database** en vez de **Object**, ya que permite a la aplicación funcionar más rápido en tiempo de ejecución.
  - **Variant**. Puede almacenar todos los tipos de datos del sistema, excepto los definidos por el usuario. Siempre ocupa 16 bytes. También puede contener tres valores especiales:
    - *Empty*. Una variable **Variant** tiene el valor **Empty** antes de asignarle un valor. Es un valor especial distinto de 0, una cadena de longitud cero ("") o el valor **Null**. Puede probar el valor **Empty** con la función **IsEmpty(***expresión***)**, que devuelve **True** o **False**.

- *Null*. Se utiliza en aplicaciones de bases de datos para indicar datos desconocidos o que faltan. Puede probar el valor **Null** con la función **IsNull(***expresión***)**, que devuelve **True** o **False**. Las expresiones que utilizan **Null** dan como resultado siempre un **Null**.
- *Error*. Es un valor especial que se utiliza para indicar un número de error definido por el usuario en un procedimiento, o para elegir alternativas basadas en el valor del error evitando el tratamiento general de errores en tiempo de ejecución. Puede probar el valor **Error** con la función **IsError(***expresión***)**, que devuelve **True** o **False**. Los valores de error se crean convirtiendo números reales mediante la función **CVErr(***númeroerror***)**.

– Puede asignar una cadena a una variable numérica si la cadena representa un valor numérico. Visual Basic convierte automáticamente las variables al tipo de dato apropiado. Sin embargo, es preferible utilizar las siguientes funciones de conversión de tipos de datos específicos:

* **CBool(**expresión**)**. Convierte una expresión de cadena o numérica en **Boolean**.
* **CByte(**expresión**)**. Convierte una expresión de cadena o numérica en **Byte**.
* **CCur(**expresión**)**. Convierte una expresión de cadena o numérica en **Currency**.
* **CDate(**expresión**)**. Convierte una expresión de cadena o numérica en **Date**.
* **CDbl(**expresión**)**. Convierte una expresión de cadena o numérica en **Double**.
* **CInt(**expresión**)**. Convierte una expresión de cadena o numérica en **Integer**.
* **CLng(**expresión**)**. Convierte una expresión de cadena o numérica en **Long**.
* **CSng(**expresión**)**. Convierte una expresión de cadena o numérica en **Single**.
* **CStr(**expresión**)**. Convierte una expresión de cadena o numérica en **String**.
* **CVar(**expresión**)**. Convierte una expresión de cadena o numérica en **Variant**.

– Las expresiones que se pasan a una función de conversión deben ser válidos para el tipo de dato de destino o se producirá un error. Por ejemplo, si intenta convertir un tipo **Long** en un **Integer**, el tipo **Long** debe estar en el intervalo válido del tipo de dato **Integer**.

1.4.4. Tipos de datos definidos por el usuario.

– Puede crear un tipo definido por el usuario con la instrucción **Type**, que debe colocar en la sección Declaraciones del módulo. Los tipos definidos por el usuario pueden declararse **Private** o **Public**. Por ejemplo:

```
[Private | Public] Type SystemInfo
      CPU As Variant
      Memory As Long
      VideoColors As Integer
      Cost As Currency
      PurchaseDate As Variant
End Type
```

– Un tipo definido por el usuario puede contener cualquiera de los tipos de datos fundamentales, matrices estáticas y dinámicas, objetos y otros tipos definidos por el usuario. El anidamiento de estructuras de datos puede ser tan complejo como se quiera.

- Para el mismo tipo definido por el usuario se pueden declarar variables de procedimiento y variables privadas o públicas de módulo. El lugar y el ámbito de la declaración de los tipos definidos por el usuario y sus variables se expone a continuación:
  - En los procedimientos las *variables* pueden declararse sólo como **privadas**.
  - En los módulos de formulario:
    - Los *tipos definidos por el usuario* pueden crearse sólo como **privados**.
    - Las *variables* pueden declararse sólo como **privadas**.
  - En los módulos estandar y de clase:
    - Los *tipos definidos por el usuario* pueden crearse como **privados** o **públicos**.
    - Las *variables* pueden declararse sólo como **privadas** o **públicas**.
- Asignar y recuperar los valores de los elementos de las variables de tipos definidos por el usuario es similar a establecer y obtener propiedades. También se puede asignar una variable a otra si ambas son del mismo tipo definido por el usuario:

```
Dim MiSistema As SystemInfo, SuSistema As SystemInfo
MiSistema.CPU = "Pentium 4"
SuSistema = MiSistema
```

1.4.5. Operadores.

- **Aritméticos**. Exponenciación (^), cambio de signo (-), multiplicación (*****), división (/), división entera (\), resto de una división entera (**Mod**), suma (+) y resta (-). Cuando en una expresión aritmética intervienen operandos de diferentes tipos, el resultado se expresa, generalmente, en la misma precisión que la del operando que la tiene más alta. El orden, de menor a mayor, según la precisión es **Integer**, **Long**, **Single**, **Double** y **Currency**.
- **De cadenas**. Concatenación de dos cadenas (**&**).
- **Relacionales**. Igual a (=), distinto (<>), menor que (<), menor o igual que (<=), mayor que (>), mayor o igual que (>=). Los operadores relacionales comparan dos expresiones dando un resultado **True** (verdadero), **False** (falso) o **Null** (no válido).
- **Lógicos**. Negación (**Not**), and (**And**), or inclusivo (**Or**), or exclusivo (**Xor**), equivalencia u opuesto a Xor (**Eqv**), implicación (**Imp**), que devuelve **False** si el primer operando es **True** y el segundo operando es **False** y devuelve **True** en los demás casos.
- **Otros operadores**. Comparar dos expresiones de caracteres (**Like**), comparar dos referencias a objetos (**Is**). El operador **Like** sirve para comparar dos cadenas de caracteres. La sintaxis para este operador es la siguiente: *Respuesta* = *Cadena1* **Like** *Cadena2*, donde *Respuesta* es **True** si *Cadena1* coincide con *Cadena2*, **False** si no coinciden y **Null** si *Cadena1* y/o *Cadena2* son **Null**.

1.4.6. Matrices estáticas y dinámicas.

- Permiten hacer referencia por el mismo nombre a una serie de variables del mismo tipo de datos y usar un índice para distinguirlas. Tienen un límite superior e inferior, y los elementos de la matriz son contiguos dentro de esos límites. Puede declarar una matriz de cualquiera de los tipos de datos fundamentales, incluyendo los tipos definidos por el usuario y tipos de objetos.

- Las *matrices estáticas* tienen siempre el mismo tamaño. Para declarar una matriz estática, ponga a continuación de su nombre el límite superior entre paréntesis. El límite inferior predeterminado es 0. La siguiente declaración crea una matriz de 15 elementos, con índices que van de 0 a 14:

```
Dim Contadores(14) As Integer
```

- Para especificar el límite inferior, proporciónelo explícitamente mediante la palabra clave **To**. La siguiente declaración crea una matriz de 15 elementos, con índices que van de 1 a 15:

```
Dim Contadores(1 To 15) As Integer
```

- Se pueden declarar matrices de varias dimensiones con límites inferiores implícitos y explícitos:

```
Static MatrizA(9, 9) As Double
Static MatrizA(1 To 10, 1 To 10) As Double
```

- Las *matrices dinámicas* puede cambiar de tamaño en cualquier momento. Declare una matriz dinámica proporcionándole una lista de dimensiones vacía, y asígnele posteriormente el número real de elementos con la instrucción **ReDim**.

```
Dim MatrizDyn()
ReDim MatrizDyn(X + 1)
```

- Cada **ReDim** puede cambiar el número de elementos, así como los límites inferior y superior de cada dimensión. Sin embargo, no se puede cambiar el número de dimensiones de la matriz. Con cada instrucción **ReDim** perderá todos los valores almacenados hasta ese momento en la matriz.
- Para cambiar el tamaño de la matriz sin perder los datos de la misma puede usar **ReDim** con la palabra clave **Preserve**. Sólo se puede cambiar el límite superior de la última dimensión de la matriz cuando se utiliza **Preserve**. En caso contrario, se produce un error en ejecución.

```
ReDim Preserve MatrizDyn(UBound(MatrizDyn) + 1)
```

- En algunos casos, también se puede asignar el contenido de una matriz a otra. Si la matriz que se usa en el lado izquierdo de la asignación es dinámica, la asignación siempre será correcta, ya que la parte izquierda puede cambiar el tamaño para coincidir con la parte derecha. Si la matriz que se usa en el lado izquierdo de la asignación es estática, falla con un error de compilación.

1.5. Procedimientos.

1.5.1. Procedimientos Sub.

- La sintaxis de un procedimiento **Sub** es la siguiente:

[Private | Public][Static] Sub *nombreProcedimiento* (*argumentos*)

instrucciones

End Sub

- Un procedimiento **Sub** es un bloque de código que no devuelve un valor. Cada vez que se llama al procedimiento se ejecutan las *instrucciones* que hay entre **Sub** y **End Sub**. Puede salir de un procedimiento **Sub** mediante la instrucción **Exit Sub**, que puede aparecer tantas veces como sea necesario, en cualquier parte del cuerpo del procedimiento. Se pueden colocar los procedimientos **Sub** en módulos estándar, módulos de clase y módulos de formulario.
- Existen dos tipos de procedimientos **Sub**:

- *Procedimientos generales*. Indican cómo realizar una tarea específica. Una vez que se define un procedimiento general, se le debe llamar específicamente desde la aplicación. De forma predeterminada, los procedimientos generales son **Public**.
- *Procedimientos de evento*. Cuando un objeto en Visual Basic reconoce que se ha producido un evento, llama automáticamente al procedimiento de evento correspondiente al evento. De forma predeterminada, todos los procedimientos de evento son **Private**:
  - Un procedimiento de evento de un control combina el nombre real del control especificado en la propiedad **Name**, un carácter de subrayado (_) y el nombre del evento. Cuando el nombre de un procedimiento no coincide con el nombre de un control, se convierte en un procedimiento general. Por ejemplo cmdPlay_Click.
  - Un procedimiento de evento de un formulario combina la palabra "Form", un carácter de subrayado y el nombre del evento. Como los controles, los formularios tienen nombres únicos, pero no se utilizan en los nombres de los procedimientos de evento.

– La llamada a un procedimiento **Sub** es una instrucción única y no devuelve un valor en su nombre. Sin embargo, un procedimiento **Sub** puede modificar los valores de las argumentos que se le pasan. Hay dos maneras de llamar a un procedimiento **Sub**:

```
Call MiProc (PrimerArgumento, SegundoArgumento)
MiProc PrimerArgumento, SegundoArgumento
```

1.5.2. Procedimientos Function.

– La sintaxis de un procedimiento **Function** es la siguiente:

[Private | Public][Static] Function *nombreProcedimiento* (*argumentos*) [**As** *tipo*]
instrucciones
End Function

– Un procedimiento **Function** es un bloque de código que puede devolver un valor del tipo de dato declarado en la cláusula **As** *tipo*. Esto incluye cualquiera de los tipos de datos fundamentales, matrices estáticas y dinámicas, objetos y tipos definidos por el usuario. Por ejemplo:

```
Public Function ArrayFunction(b As Byte) As Byte()
      Dim x(2) As Byte
      x(0) = b
      x(1) = b + CByte(200)
      x(2) = b + b
      ArrayFunction = x
End Function
```

– Aunque es posible devolver la matriz asignando otra matriz (`ArrayFunction = x()`), no es recomendable por motivos de rendimiento. Debe especificar un tipo para una función que devuelva una matriz. Puede ser un tipo **Variant**. `Function X() As Variant()` siempre funcionará, mientras que `Function X() As ()` generaría un error.

– Cada vez que se llama al procedimiento se ejecutan las *instrucciones* que hay entre **Function** y **End Function**. Puede salir de un procedimiento **Function** mediante la instrucción **Exit Function**, que puede aparecer tantas veces como sea necesario, en cualquier parte del cuerpo del

procedimiento. Se pueden colocar los procedimientos **Function** en módulos estándar, módulos de clase y módulos de formulario.

- Los procedimientos **Function** se caracterizan por lo siguiente:
  - Los procedimientos **Function** tienen tipos de datos, al igual que las variables, que determinan el tipo del valor devuelto. En ausencia de **As**, el tipo predeterminado es **Variant**.
  - El nombre del procedimiento es el valor de retorno y actúa como una variable dentro del cuerpo del procedimiento (*nombreProcedimiento = expresión*). Si no se efectúa esta asignación, el resultado devuelto será cero si la función es numérica, nulo ("") si la función es de tipo **String**, o *Empty* si la función es de tipo **Variant**. Cuando el procedimiento **Function** devuelve un valor, se puede convertir en parte de una expresión mayor.
  - Generalmente, se llama a una función incluyendo el nombre y los argumentos entre paréntesis del procedimiento en la parte derecha de una expresión mayor.

```
valorRetorno = MiFunc (PrimerArgumento, SegundoArgumento)
```

- Normalmente se llama a un procedimiento **Function** utilizando su nombre en una expresión. También es posible llamar a una función igual que se llama a un procedimiento **Sub**. Cuando llama a una función de esta manera, Visual Basic desecha el valor devuelto.

1.5.3. Trabajo con procedimientos.

1.5.3.1. Llamar a procedimientos públicos de otros módulos.

- Se puede llamar a los procedimientos públicos de otros módulos desde cualquier parte del proyecto. Necesitará especificar el módulo que contiene el procedimiento al que está llamando.
- Las técnicas para hacerlo dependen del tipo de módulo que contiene el procedimiento:
  - *Procedimientos de formulario*. Todas las llamadas que se hacen desde fuera del módulo de formulario deben señalar al módulo de formulario que contiene el procedimiento. Si un procedimiento llamado SomeSub está en un módulo de formulario llamado Form1, puede llamar al procedimiento en Form1 mediante esta instrucción:

```
Call Form1.SomeSub(argumentos)
```

  - *Procedimientos de clase*. Requiere que la llamada al procedimiento esté calificada con una variable que señale a una instancia de la clase. Debe declararse primero la instancia de la clase como una variable de objeto y el nombre de la clase debe hacer referencia a ella. Por ejemplo, DemoClass es una instancia de una clase llamada Class1:

```
Dim DemoClass as New Class1
DemoClass.SomeSub
```

  - *Procedimientos de módulo estándar*. Si el nombre de un procedimiento es único, no necesita incluir el nombre del módulo en la llamada. Si dos o más módulos contienen un procedimiento con el mismo nombre, debe calificarlo con el nombre del módulo estándar.

1.5.3.2. Pasar argumentos a procedimientos.

- Los argumentos de los procedimientos que escriba tienen el tipo de dato **Variant** de forma predeterminada. Sin embargo, puede declarar otros tipos de datos para los argumentos con la cláusula **As** *tipo*. Si especifica el tipo de dato de un argumento, debe pasar un valor de ese tipo.

```
Function QueComer (DiaSemana As String, Hora _
```

```
        As Integer) As String
        …
    End Function
```

- Hay dos maneras de pasar los argumentos a los procedimientos:
  - *Por valor*. Sólo se pasa una copia de la variable. Si el procedimiento cambia el valor, el cambio afecta sólo a la copia y no a la variable propiamente dicha. Utilice la palabra clave **ByVal** para indicar un argumento pasado por valor. Por ejemplo:
    ```
    Sub Cuentas (ByVal intNumCuenta as Integer)
    ```
  - *Por referencia (predeterminado)*. Le da al procedimiento acceso al contenido real de la variable en su ubicación de dirección de memoria. Como resultado, el procedimiento al que se ha pasado el valor de la variable se puede modificar de manera permanente.
- Puede especificar un argumento opcional si coloca la palabra clave **Optional** en el argumento. Si especifica un argumento opcional, todos los argumentos subsiguientes de la lista de argumentos deben ser también opcionales y se deben declarar con la palabra clave **Optional**. También es posible especificar un valor predeterminado para un argumento opcional. El siguiente ejemplo devuelve un valor predeterminado si no se pasa el argumento opcional:
```
Sub ListText(x As String, Optional y As _
      Integer = 12345)
      …
End Sub
```
- Se puede utilizar la función **IsMissing(***nombreArgumento***)**, que devuelve **True** o **False**, para comprobar si se ha introducido un argumento opcional. **IsMissing** no funciona en tipos de datos simples tales como **Integer** o **Double**. En este caso, si el parámetro opcional tiene un valor predeterminado, al omitir el argumento, éste tendrá el valor predeterminado:
```
Sub MiSub(Optional MiVar As String = "specialvalue")
   If MiVar = "specialvalue" Then
      … ' se omitió MiVar.
   End If
End Sub
```
- Generalmente, el número de argumentos en la llamada a un procedimiento debe ser el mismo que los especificados en el procedimiento. Usar la palabra clave **ParamArray** le permite especificar que el procedimiento aceptará un número arbitrario de argumentos.
```
Sub Sum(ParamArray intNums())
      …
End Sub
```
- Puede pasar objetos a procedimientos. También es posible pasar un objeto a un argumento por referencia, y desde el procedimiento establecer el argumento al nuevo objeto. También puede pasar variables declaradas mediante un tipo definido por el usuario. Si desea pasar un tipo definido por el usuario en un módulo de formulario, el procedimiento debe ser privado.

- Los tipos definidos por el usuario siempre se pasan por referencia, de modo que en el caso de tipos definidos por el usuario con matrices de gran tamaño, esto podría originar una reducción de las prestaciones. En tal caso, es preferible extraer y pasar únicamente los datos necesarios.

1.6. Estructuras de control.

1.6.1. Estructuras de decisión.

1.6.1.1. If…Then.

- Use la estructura **If...Then** para ejecutar una o más instrucciones basadas en una condición. Puede usar la sintaxis de una línea o un *bloque* de varias líneas:

  If *condición* **Then** *instrucción*

  If *condición* **Then**

  Instrucciones

  End If

- *Condición* normalmente es una comparación, pero puede ser cualquier expresión que dé como resultado un valor numérico. Visual Basic interpreta este valor como **True** o **False**; un valor numérico cero es **False** y se considera **True** cualquier valor numérico distinto de cero. Si *condición* es **True**, se ejecutan todas las *instrucciones* que siguen a la palabra clave **Then**.
- El formato de una única línea de **If...Then** no utiliza la instrucción **End If**. Si desea ejecutar más de una línea de código cuando *condición* sea **True**, debe usar la sintaxis de bloque de varias líneas **If...Then...End If**.

1.6.1.2. If…Then…Else.

- Un bloque **If...Then...Else** define varios bloques de instrucciones, uno de los cuales se ejecutará:

  If *condición1* **Then**

  [*bloque de instrucciones 1*]

  [**ElseIf** *condición2* **Then**

  [*bloque de instrucciones 2*]] ...

  [**Else**

  [*bloque de instrucciones n*]]

  End If

- Visual Basic evalúa primero *condición1*. Si es **False**, Visual Basic procede a evaluar *condición2* y así sucesivamente. Cuando encuentra una condición **True**, Visual Basic ejecuta el bloque de instrucciones correspondientes y después ejecuta el código que sigue a **End If**. Puede incluir un bloque de instrucciones **Else**, que Visual Basic ejecutará si ninguna de las condiciones es **True**.
- **If...Then...ElseIf** es un caso especial de **If...Then...Else**. Puede tener cualquier número de cláusulas **ElseIf** o ninguna, y puede incluir una cláusula **Else** teniendo o no cláusulas **ElseIf**.

1.6.1.3. Select Case.

- La estructura **Select Case** funciona con una expresión que se evalúa una vez al principio de la estructura. Visual Basic compara el resultado de esta expresión con los valores de cada **Case** de la estructura. Si hay una coincidencia, ejecuta el bloque de instrucciones asociado a ese **Case**:

  Select Case *expresiónPrueba*

  [**Case** *listaExpresiones1*

[*bloque de instrucciones 1*]]
[**Case** *listaExpresiones2*
[*bloque de instrucciones 2*]]
…
[**Case Else**
[*bloque de instrucciones n*]]
End Select

- Cada *listaExpresiones* es una lista de uno o más valores. Si hay más de un valor en una lista, se separan los valores con comas. Cada *bloque de instrucciones* contiene cero o más instrucciones. Si más de un **Case** coincide con la expresión de prueba, sólo se ejecutará el bloque de instrucciones asociado con la primera coincidencia. Se ejecutan las instrucciones de la cláusula opcional **Case Else** si ningún valor de *listaExpresiones* coincide con la expresión de prueba.
- La estructura **Select Case** evalúa una expresión cada vez al principio de la estructura. Por el contrario, la estructura **If...Then...Else** puede evaluar una expresión diferente en cada instrucción **ElseIf**. Sólo puede sustituir una estructura **If...Then...Else** con una estructura **Select Case** si la instrucción **If** y cada instrucción **ElseIf** evalúa la misma expresión.

1.6.2. Estructuras iterativas.

1.6.2.1. Do…Loop.

- Utilice el bucle **Do** para ejecutar un bloque de instrucciones un número indefinido de veces. Todas las variantes de la instrucción **Do...Loop** evalúan una condición numérica para determinar si continúa la ejecución. Como ocurre con **If...Then**, la *condición* debe ser un valor o una expresión que dé como resultado **False** (cero) o **True** (distinto de cero).
- En el **Do...Loop** siguiente, primero se evalúa *condición*. Si es **False**, se salta todas las instrucciones. Si es **True**, ejecuta las instrucciones, vuelve a la instrucción **Do While** y evalúa la condición de nuevo. Nunca se ejecutan las instrucciones si *condición* es **False** inicialmente:

Do While *condición*
Instrucciones
Loop

- Otra variante de la instrucción **Do...Loop** ejecuta las instrucciones primero y prueba *condición* después de cada ejecución. Esta variación garantiza al menos una ejecución de *instrucciones*:

Do
Instrucciones
Loop While *condición*

- Hay otras dos variantes que repiten el bucle siempre y cuando *condición* sea **False**. La primera hace el bucle cero o más veces, y la segunda hace el bucle al menos una vez:

Do Until *condición*
Instrucciones
Loop
Do
Instrucciones

Loop Until *condición*

1.6.2.2. For…Next.

- Utilice el bucle **For…Next** para ejecutar un bloque de instrucciones un número indefinido de veces. El bucle **For** utiliza una variable contador que incrementa o reduce su valor en cada repetición del bucle. La sintaxis es la siguiente:

  For *contador = iniciar* **To** *finalizar* [**Step** *incremento*]
  Instrucciones
  Next [*contador*]

- Los argumentos *contador*, *iniciar*, *finalizar e incremento* son todos numéricos. El argumento *incremento* puede ser positivo o negativo. Si *incremento* es positivo, *iniciar* debe ser menor o igual que *finalizar* o no se ejecutarán las instrucciones del bucle. Si *incremento* es negativo, *iniciar* debe ser mayor o igual que *finalizar* para que se ejecute el cuerpo del bucle. Si no se establece **Step**, el valor predeterminado de *incremento* es 1.
- Al ejecutar el bucle **For**, Visual Basic:
  1. Establece *contador* al mismo valor que *iniciar*.
  2. Comprueba si *contador* es mayor que *finalizar*. Si lo es, Visual Basic sale del bucle. Si *incremento* es negativo, Visual Basic comprueba si *contador* es menor que *finalizar*.
  3. Ejecuta *instrucciones* e incrementa *contador* en 1.
  4. Repite los pasos 2 y 3.

1.6.2.3. For Each…Next.

- El bucle **For Each...Next** es similar al bucle **For...Next**, pero repite un grupo de instrucciones por cada elemento de una matriz o de una colección de objetos.
- La sintaxis del bucle **For Each...Next** es la siguiente:

  For Each *elemento* **In** *grupo*
  Instrucciones
  Next [*elemento*]

- La variable *elemento* se utiliza para iterar por los elementos de la colección o de la matriz:
  - Para las colecciones, *elemento* solamente puede ser una variable **Variant**, una variable **Object** genérica o cualquier variable de objeto específica.
  - Para las matrices, *elemento* solamente puede ser una variable tipo **Variant**.
- La variable *grupo* es el nombre de una colección de objetos o de una matriz de variables **Variant**. No puede usar **For Each...Next** con una matriz de tipos definidos por el usuario porque una variable **Variant** no puede contener un tipo definido por el usuario.
- La entrada al bloque **For Each** se produce si hay al menos un elemento en *grupo*. Una vez que se ha entrado en el bucle, todas las instrucciones en el bucle se ejecutan para el primer elemento en *grupo*. Después, mientras haya más elementos en *grupo*, las instrucciones en el bucle continúan ejecutándose para cada elemento. Cuando no hay más elementos en el *grupo*, se sale del bucle y la ejecución continúa con la instrucción que sigue a la instrucción **Next**.

1.6.3. Trabajo con estructuras de control.

- Puede anidar las estructuras de control en Visual Basic en tantos niveles como desee.

- La instrucción **Exit** le permite salir directamente de un bucle **For** y de un bucle **Do**. Ambas pueden aparecer tantas veces como sea necesario dentro de un bucle **For** y dentro de un bucle **Do** respectivamente. **Exit Do** funciona con todas las variantes de la sintaxis del bucle **Do**:

  For *contador* **=** *iniciar* **To** *finalizar* [**Step** *incremento*]
  [*bloque de instrucciones*]
  [**Exit For**]
  [*bloque de instrucciones*]
  Next [*contador*[, *contador*] [,...]]
  Do [{**While** | **Until**} *condición*]
  [*bloque de instrucciones*]
  [**Exit Do**]
  [*bloque de instrucciones*]
  Loop [{**While** | **Until**} *condición*]

1.7. Objetos.

1.7.1. Fundamentos de objetos.

- Un *objeto* es una combinación de código y datos que se puede tratar como una unidad. Cada objeto de Visual Basic se define mediante una *clase*. Se utiliza la clase para crear objetos. Se crean todos los objetos como instancias de sus clases. Cada objeto comparte un conjunto de características y capacidades comunes definidos por la clase a la que pertenece.
- Los objetos de Visual Basic aceptan propiedades, métodos y eventos. En Visual Basic, los datos de un objeto (configuración o atributos) se llaman *propiedades*, mientras que los diversos procedimientos que pueden operar sobre el objeto se conocen como sus *métodos*. Un *evento* es una acción reconocida por un objeto, y puede escribir código que responda a ese evento.
- Establezca el valor de una propiedad cuando desee modificar la apariencia o el comportamiento de un objeto. Para establecer el valor de una propiedad, utilice la sintaxis siguiente:

  objeto.propiedad = expresión
- Obtenga el valor de una propiedad cuando desee encontrar el estado de un objeto antes de que el código realice acciones adicionales (como asignar el valor a otro objeto). En la mayoría de los casos, para obtener el valor de una propiedad se utiliza la sintaxis siguiente:

  variable = objeto.propiedad
- Los métodos pueden afectar a los valores de las propiedades. Cuando utiliza un método en el código, la forma en que escribe la instrucción depende de los argumentos que necesite el método y de si el método devuelve o no un valor. Cuando un método no necesita argumentos, escriba el código mediante la sintaxis siguiente:

  objeto.método
- Si el método necesita más de un argumento, separe los argumentos mediante comas. Si conserva el valor de devolución de un método, debe poner los argumentos entre paréntesis. Si no hay valor de devolución, los argumentos se ponen sin paréntesis.

1.7.2. Creación de objetos.

- Para usar un objeto tiene que mantener una *referencia* a él en una *variable de objeto*, que puede ser un *enlace en tiempo de compilación* (más lento) o un *enlace en tiempo de diseño*. Éstas últimas pueden ser *enlaces DispID* o *enlaces vtable* (más rápidos).
- Una variable de objeto se declara de la misma manera que las demás variables. Las únicas diferencias son la palabra clave opcional **New** y el argumento *clase*:

  {**Dim | ReDim | Static | Private | Public**} *variable* **As** [**New**] *clase*

- La palabra clave **New** crea una nueva instancia de la clase en el instante en que se hace referencia a *variable* por primera vez desde el código. Si no se utiliza **New**, para usar una variable de objeto es necesario asignarle un objeto o una nueva instancia de una clase:

  Set *variable* = *objeto*

  Set *variable* = **New** *clase*

- Se puede usar la palabra clave **New** para crear colecciones y objetos desde clases definidas en módulos de clase. Sin embargo, no puede usarse para crear:
  - Variables de los tipos de datos fundamentales (`Dim X As New Integer`).
  - Variables de cualquier tipo genérico de objeto (`Dim X As New Control`).
  - Variables de cualquier tipo de control específico (`Dim X As New ListBox`).
  - Variables de cualquier control específico (`Dim X As New lstNames`).
- El tipo *clase* determina el tipo de objeto que se crea. Puede crear instancias de formularios, clases definidas en módulos de clase y colecciones. Puede declarar una variable de objeto que:
  - Haga referencia a un formulario concreto de la aplicación.

    ```
    Dim FormVar As New frmMain
    ```

  - Pueda hacer referencia a cualquier formulario de la aplicación

    ```
    Dim anyForm As Form
    ```

  - Haga referencia a un tipo de control concreto de la aplicación.

    ```
    Dim anyText As TextBox
    ```

  - Pueda hacer referencia a cualquier control de la aplicación

    ```
    Dim anyControl As Control
    ```

- Cada formulario que se crea en tiempo de diseño es una clase, cuyo nombre es el de la propiedad **Name** del formulario. Se puede usar la palabra clave **New** para crear nuevas instancias de esa clase. Cada instancia tiene los mismos controles que en el formulario en tiempo de diseño:

  ```
  Dim x As New Ejemplo
  x.Show
  ```

- Puede declarar y usar matrices estáticas y dinámicas de un tipo de objeto de la misma manera en que declara y utiliza matrices de cualquier tipo de datos. Si declara una matriz de formularios con la palabra clave **New**, se crea automáticamente una nueva instancia de formulario por cada elemento de la matriz. Sin embargo, no puede declarar las matrices de controles con la palabra clave **New**. Si declara una matriz de un tipo específico de control, sólo puede asignar a la matriz controles de dicho tipo, aunque pueden provenir de formularios diferentes:

  ```
  ReDim ActiveImages(10) As Image
  ```

- Las variables de objetos específicos deben hacer referencia a un tipo específico de objeto o clase. Las variables genéricas de objetos se pueden referir a uno de los muchos tipos específicos de objetos. Hay varios tipos genéricos de objetos en Visual Basic:
  - *Form*. El objeto al que hace referencia es cualquier formulario de la aplicación.
  - *Control*. El objeto al que hace referencia es cualquier control de la aplicación.
  - *Object*. El objeto al que hace referencia es cualquier objeto.
- Las variables genéricas de objetos declaradas **As Object** pueden almacenar objetos de muchas clases diferentes. De la misma forma, las variables declaradas **As Form** y **As Control** pueden contener formularios y controles de distintas clases. Para distinguir el tipo de objeto que contiene se puede utilizar la función **TypeName** en cualquier lugar del código.
- Puede usar el operador **Is** para comprobar la identidad de las referencias de objeto de Visual Basic. Si compara dos referencias diferentes al mismo objeto, el operador **Is** devuelve **True**:

```
Dim MiObj, SuObj, EsteObj, AquelObj, OtroObj
Dim MiPrueba As Boolean
' Asigna referencias de objeto.
Set SuObj = MiObj
Set EsteObj = MiObj
Set AquelObj = OtroObj
MiPrueba = SuObj Is EsteObj ' Devuelve True.
MiPrueba = AquelObj Is EsteObj ' Devuelve False.
```

- Cada objeto utiliza memoria y recursos del sistema. Para liberar los recursos:
  - Utilice **Unload** para descargar un formulario o un control de la memoria.
  - Utilice **Nothing** para liberar los recursos utilizados por una variable de objeto. Asigne **Nothing** a una variable de objeto con la instrucción **Set**.

1.7.3. Tipos de objetos.

1.7.3.1. El objeto App.

- El objeto **App** es un objeto global al que se tiene acceso con la palabra clave **App**. Determina o especifica información acerca del título de la aplicación, información de la versión, la ruta de acceso y el nombre de su archivo ejecutable y los archivos de Ayuda, y si está ejecutándose o no una instancia anterior de la aplicación.
- Tiene la propiedad **Path** [= *nombreRuta*]. Especifica la ruta de acceso del archivo .vbp de proyecto cuando se ejecuta la aplicación desde el entorno de desarrollo, o la ruta de acceso del archivo .exe cuando se ejecuta la aplicación como un archivo ejecutable.

1.7.3.2. El objeto Screen.

- El objeto **Screen** es todo el escritorio de Windows, controlando el puntero del *mouse* fuera de los formularios de la aplicación en tiempo de ejecución.
- Puede definir la propiedad **MousePointer** del objeto **Screen** como puntero de reloj de arena mientras se presenta un formulario modal. También dispone de las propiedades **ActiveForm** y **ActiveControl** para obtener el formulario y el control activos respectivamente de la pantalla.

- Cuando se cambia el tamaño de un formulario o se mueve, debe comprobarse las propiedades **Height** y **Width** del objeto **Screen** para asegurarse de que el formulario cabrá en la pantalla.

1.7.3.3. El objeto Clipboard.

1.7.3.3.1. Descripción.

- El objeto **Clipboard** se utiliza para manipular el texto y los gráficos del Portapapeles. Puede usar este objeto para permitir que el usuario copie, corte y pegue texto o gráficos desde su aplicación. Puede contener varios datos separados mientras tengan un formato diferente.
- Todas las aplicaciones Windows comparten el objeto **Clipboard** y su contenido puede variar cuando pase a otra aplicación. Por tanto, antes de copiar en el objeto **Clipboard** debe eliminar su contenido. Los datos del Portapapeles se pierden cuando se copia otro conjunto de datos del mismo formato en el Portapapeles, desde el código o mediante un comando de menú.

1.7.3.3.2. Métodos.

- **GetFormat(***formato***)**. Devuelve un entero que indica si un elemento del objeto **Clipboard** coincide con *formato*. Devuelve **True** si algún elemento del objeto **Clipboard** coincide con el formato especificado, de lo contrario devuelve **False**. Los valores admitidos para *formato* son:
  - **vbCFLink**. Vínculo de intercambio dinámico de datos.
  - **vbCFText**. Texto.
  - **vbCFBitmap**. Mapa de bits.
  - **vbCFMetafile**. Metarchivo.
  - **vbCFDIB**. Mapa de bits independiente del dispositivo.
  - **vbCFPalette**. Paleta de colores.
- **SetText** *datos*. Pone una cadena de texto en el objeto **Clipboard**.
- **GetText( )**. Recupera una cadena de texto del objeto **Clipboard**. Si no hay ninguna cadena de texto almacenada, se devuelve una cadena de longitud cero ("").
- **SetData** *datos, formato*. Pone una imagen en el objeto **Clipboard** usando el formato gráfico especificado. Si se omite *formato*, **SetData** determina automáticamente el formato gráfico.
- **GetData(***formato***)**. Recupera una imagen del objeto **Clipboard**. Si se omite *formato*, **GetData** utiliza automáticamente el formato apropiado. Si no hay ningún gráfico almacenado que coincida con el formato esperado, no se devuelve nada.
- **Clear**. Borra el contenido del objeto **Clipboard**.

1.8. Colecciones.

1.8.1. Fundamentos de colecciones.

- Una colección es una manera de agrupar elementos relacionados. Visual Basic proporciona la clase genérica **Collection** para ofrecer la posibilidad de definir colecciones propias. Se pueden crear tantos objetos **Collection** (instancias de la clase **Collection**) como sean necesarios.
- Las colecciones son objetos en sí mismas, y no es necesario volver a dimensionar los objetos **Collection** a medida que se agregan y quitan miembros. Los objetos **Collection** almacenan cada elemento en una variable **Variant**. Por tanto, se pueden almacenar los tipos de datos fundamentales, los objetos y las matrices estándar, pero no los tipos definidos por el usuario.
- Las colecciones se declaran de la siguiente manera:

```
[Private | Public] nombreColección As New Collection
```

La palabra clave **New** de la declaración de la variable `nombreColección` provoca la creación de un objeto **Collection** la primera vez que se hace referencia a dicha variable desde el código. Como **Collection** es una clase, no un tipo de datos, es necesario crear una instancia y mantener una referencia a dicha instancia (objeto) en una variable.

- Los elementos de una colección se distinguen por una clave que puede ser un identificador de tipo **String** o un índice de tipo **Long** comprendido entre uno y el valor de la propiedad **Count**. Si desea que una clave se interprete como identificador y la variable que contiene la clave no es de tipo **String**, utilice **CStr** para convertirla. Si desea que una clave se interprete como índice y la variable que contiene la clave no es de un tipo de datos numéricos, utilice **CLng** para convertirla.
- Una colección puede tener *base cero* o *base uno*, dependiendo de su índice inicial. Las colecciones más antiguas de Visual Basic suelen tener base cero, mientras que las más recientes suelen tener base uno. Ejemplos de colecciones con base cero son las colecciones **Forms** y **Controls**. El objeto **Collection** es un ejemplo de colección con base uno.
- Al igual que cualquier otro objeto, el objeto **Collection** se destruye cuando se establece a **Nothing** la última variable que contiene una referencia a él o cuando pierde su alcance.
- Las colecciones más utilizadas en Visual Basic son las siguientes:
  - **Forms**. Contiene los formularios cargados actualmente.
  - **Controls**. Contiene los controles de un formulario.
  - **Printers**. Contiene los objetos **Printer** disponibles.

1.8.2. Propiedades y métodos del objeto Collection.

1.8.2.1. Propiedades.

- **Count**. Devuelve el número de elementos de la colección. Es de sólo lectura.

1.8.2.2. Métodos.

- **Add** *elemento*, *clave*, *before*, *after*. Agrega elementos a la colección ajustando el tamaño de la colección automáticamente. El argumento *clave* puede ser el índice del elemento que se desea añadir o su identificador. Los argumentos *before* y *after* especifica que el nuevo miembro se colocará antes y después respectivamente del miembro identificado por el argumento. Es posible especificar posiciones con *before* o con *after*, pero no con ambos a la vez.
- **Remove** *clave*. Elimina un elemento de la colección. El argumento *clave* puede ser el índice del elemento que se desea eliminar o su identificador.
- **Item(***clave***)**. Devuelve un elemento de la colección. El argumento *clave* puede ser el índice del elemento que se desea recuperar o su identificador. La sintaxis es la siguiente:

  [Set] *variable* = *colección*.**Item(***índice***)**

1.8.3. Trabajo con colecciones.

- Cada uno de los objetos de una colección tiene sus propias características y métodos. Puede usar dos técnicas generales para dirigirse a un miembro de un objeto colección:
  - Especifique el nombre del miembro. Las expresiones siguientes son equivalentes:

```
Controls("List1")
Controls!List1
```

- Utilice el número de índice del miembro:

```
Controls(3)
```

- Puede usar **For Each ... Next** o un índice para recorrer la colección. En este caso **For Each** es notablemente más rápido que la iteración mediante índice, aunque esto depende de cómo almacena la colección los datos internamente.

```
Dim emp As Employee
For Each emp In colEmployees
      emp.Rate = emp.Rate * 1.1
Next emp
```

1.9. Control de errores.

1.9.1. El objeto Err.

- Cuando se produce un error, Visual Basic establece las diferentes propiedades del objeto de error **Err**. Se usar el objeto **Err** y sus propiedades en una rutina de tratamiento de errores para responder a una situación de error. Las propiedades más importantes son:
  - **Number**. Es un valor numérico que representa el error en tiempo de ejecución más reciente.
  - **Description**. Contiene una cadena descriptiva asociada al número de error actual.
  - **Source**. Es una cadena que representa al objeto que generó el error.
- Al controlar errores, no se debe utilizar ninguna propiedad de **Err** distinta de **Number** mediante programa, ya que su único uso válido de estas propiedades es mostrar una serie de información a un usuario final cuando no se puede manejar un error.

1.9.2. Interceptación y tratamiento de errores.

- Para establecer una interceptación de errores que salte a una rutina de tratamiento de errores en un procedimiento, use una instrucción **On Error GoTo** *línea*, donde *línea* es la etiqueta que identifica el código de tratamiento de errores. La interceptación de errores se activa cuando se ejecuta la instrucción **On Error**, y permanece activada hasta que el procedimiento termina su ejecución mediante **Exit Sub**, **Exit Function**, **End Sub** o **End Function**.
- Aunque sólo puede haber una interceptación de errores activada a la vez en cualquier procedimiento dado, puede crear varias interceptaciones de error alternativas y activar distintas interceptaciones en momentos diferentes. También puede usar la instrucción **On Error GoTo 0** para desactivar el tratamiento de errores en cualquier lugar de un procedimiento, incluso dentro de la misma rutina de tratamiento de errores.
- Una rutina de tratamiento de errores comienza con una etiqueta de línea seguida por un signo de dos puntos. El código de tratamiento de errores se sitúa al final del procedimiento con una instrucción **Exit Sub** o **Exit Function** inmediatamente delante de la etiqueta de línea. Esto permite que el procedimiento evite la ejecución de código de tratamiento de errores si no se produce ningún error. El cuerpo de la rutina de tratamiento de errores contiene el código que realmente trata el error, normalmente en forma de una instrucción **Case** o **If…Then…Else**.
- Hay varias maneras de salir de una rutina de tratamiento de errores:
  - **Resume**. Reanuda la ejecución del programa en la instrucción que causó el error. Se usa para repetir una operación cuando la rutina de tratamiento de errores puede corregir el error.

- **Resume Next**. Reanuda la ejecución del programa en la instrucción que sigue inmediatamente a la que causó el error. Se usa cuando la rutina de tratamiento de errores no puede corregir el error, o cuando éste se produce dentro de un bucle.
- **Resume** *línea*. Reanuda la ejecución del programa en la etiqueta especificada por *línea*, que debe estar en el mismo procedimiento que la rutina de tratamiento de errores.
- **Err.Raise Number:=** *número*. Desencadena un error en tiempo de ejecución. Cuando se ejecuta esta instrucción dentro de la rutina de tratamiento de errores, Visual Basic busca en la lista de llamadas otra rutina de tratamiento de errores.

1.9.3. Jerarquía del tratamiento de errores.

- Si se produce un error en un procedimiento que no tiene una rutina de tratamiento de errores activada, Visual Basic busca hacia atrás en sentido ascendente en la cadena de procedimientos que se han invocado para llegar al punto actual de ejecución, y ejecuta la primera rutina de tratamiento de errores activada que encuentre.
- Si no encuentra ningún controlador de errores activado en ningún lugar de la lista de llamadas, presenta un mensaje predeterminado de error inesperado y detiene la ejecución.
- Si Visual Basic encuentra un controlador de errores activado en un procedimiento, la ejecución continúa en esa rutina como si el error se hubiera producido en dicho procedimiento. Si en la rutina de tratamiento de errores se ejecuta una instrucción:
  - **Resume**. Se devuelve el control a la instrucción de llamada a procedimiento ejecutada más recientemente dentro del procedimiento que contiene el controlador de errores activado.
  - **Resume Next**. Se devuelve el control a la instrucción inmediatamente posterior a la instrucción de llamada a procedimiento ejecutada más recientemente dentro del procedimiento que contiene el controlador de errores activado.
- Si la rutina de tratamiento de errores no incluye el error que se ha producido realmente, podria producirse un bucle si se ejecuta una instrucción **Resume**. Para evitar este tipo de situaciones, use el método **Raise** del objeto **Err** en una instrucción **Case Else** del controlador.

1.9.4. Tratamiento incorporado de errores.

- El *tratamiento de errores incorporado* consiste en buscar los errores inmediatamente después de cada línea que pueda producir un error. Este tratamiento puede plantearse de diferentes maneras:
  - Crear funciones que devuelvan en vez de un valor un número de error, si éste se produce.
  - Provocar un error de Visual Basic con **Err.Raise Number:=** *número* en un procedimiento y tratar el error en el controlador de errores del procedimiento invocante. Para este tipo de tratamiento hay que incluir la instrucción **On Error Resume Next** antes de la llamada.
  - Escribir una función para devolver un tipo de datos **Variant** y usar dicho tipo para indicar al procedimiento de llamada que se ha producido un error. Un tipo **Variant** tiene una etiqueta que indica qué tipos de datos están contenidos en la variable y puede etiquetarse como un código de error de Visual Basic mediante **CVErr(***Err.Number***)**.
- Use el método **Clear** si necesita borrar de manera explícita el objeto **Err** después de haber tratado un error cuando se utiliza un tratamiento de errores incorporado con **On Error Resume**

Next. Visual Basic llama automáticamente al método **Clear** siempre que ejecuta cualquier tipo de instrucción **Resume**, **Exit Sub**, **Exit Function** o cualquier instrucción **On Error**.

2. Controles estándar.

2.1. Propiedades, eventos, métodos y funciones generales.

2.1.1. Propiedades.

- **Name**. Establece el nombre con el que se hará referencia al objeto en el código.
- **Parent**. Devuelve el formulario, objeto o colección que contiene un control u otro objeto o colección. Se utiliza para tener acceso a las propiedades, los métodos o los controles del primario de un objeto que se pasa como argumento de un procedimiento.
- **Height** [= *número*] y **Width** [= *número*]. Determinan el alto y ancho respectivamente del control en unidades del sistema de coordenadas del contenedor.
- **Left** [= *valor*] y **Top** [= *valor*]. Determinan la ubicación del control con relación a la esquina superior izquierda de su contenedor, en unidades del sistema de coordenadas del contenedor.
- **Font**. Devuelve un objeto **Font** asociado a un control y cuyas propiedades contienen la información necesaria para dar formato al texto. Estas propiedades son:
  - **Name** [= *cadena*]. Devuelve o establece el nombre de la fuente.
  - **Bold** [= **True** | **False**]. Activa o desactiva el formato de negrita.
  - **Italic** [= **True** | **False**]. Activa o desactiva el formato de cursiva.
  - **Underline** [= **True** | **False**]. Activa o desactiva el formato de subrayado.
  - **StrikeThrough** [= **True** | **False**]. Activa o desactiva el formato de tachado.
  - **Size** [= *número*]. Devuelve o establece el tamaño de fuente en puntos.
- **Enabled** [= **True** | **False**]. Devuelve o establece un valor que determina si un control puede responder a eventos generados por el usuario (**True**) o no responde (**False**).
- **Visible** [= **True** | **False**]. Determina si el control es visible (**True**) o está oculto (**False**).
- **CausesValidation** [= **True** | **False**]. Devuelve o establece un valor que determina si el control desde el que ha cambiado el enfoque activa su evento **Validate(**[*Cancel* **As Boolean**]**)** (**True** predeterminado) o no lo activa (**False**).
- **TabIndex** [= *índice*]. Devuelve o establece el lugar que ocupa en el orden de tabulación con la tecla TAB un determinado control dentro de su formulario primario. Cada formulario tiene su propio orden de tabulación. El valor *índice* es un número entero entre 0 y (*n*–1), donde *n* es el número de controles del formulario que tienen la propiedad **TabIndex**. Los controles que no pueden obtener el enfoque, al igual que los controles desactivados o invisibles, permanecen en el orden de tabulación pero se pasan por alto al tabular.
- **TabStop** [= **True** | **False**]. Devuelve o establece un valor que indica si el usuario puede usar la tecla TAB para llevar el enfoque al control. Si tiene el valor **False**, el control se salta en la tabulación, aunque el objeto mantiene su lugar en el orden de tabulación determinado por la propiedad **TabIndex**. El valor predeterminado es **True**.
- **MousePointer** [= *valor*]. Devuelve o establece un valor que indica el tipo de puntero de *mouse* que aparece cuando éste se sitúa sobre el control. El puntero seleccionado aparece cuando el

mouse está sobre el control correspondiente. Algunos de los valores admitidos son **vbDefault**, **vbHourglass**, **vbSizePointer**, **vbNoDrop**, **vbArrow** y **vbCrosshair**.

- **MouseIcon** [= *icono*]. Establece un icono o un cursor personalizado. El archivo debe tener la extensión .ico o .cur y el formato correspondiente. Para poder utilizar este puntero personalizado, es necesario que **MousePointer** esté establecido a **vbCustom**.

2.1.2. Eventos.

- **GotFocus( )**. Se produce cuando el control recibe el enfoque. En ese momento tiene la capacidad de recibir clics del *mouse* o entradas por teclado en cualquier momento. Un control sólo puede recibir el enfoque si sus propiedades **Enabled** y **Visible** tienen el valor **True**. Los controles que no pueden recibir el enfoque son **Frame**, **Image**, **Label**, **Line**, **Shape** y **Timer**.
- **LostFocus( )**. Se produce cuando el formulario pierde el enfoque. Se usa para validar actualizaciones, o para modificar las condiciones establecidas en el evento **GotFocus(.)**.
- **Validate(**[*Cancel* **As Boolean**]**)**. Ocurre antes de que un control pierda el enfoque siempre y cuando el control que va a recibir el foco tiene su propiedad **CausesValidation** establecida a **True**. Si el argumento *Cancel* se establece al valor **True**, el control mantiene el enfoque. Es más adecuado que el evento **LostFocus( )** para validar los datos porque puede retener el enfoque.

2.1.3. Métodos.

- **SetFocus**. Mueve el enfoque al control especificado. Un control sólo puede recibir el enfoque si sus propiedades **Enabled** y **Visible** tienen el valor **True**. Algunos controles no pueden recibir el enfoque. Son el control **Frame**, el control **Image**, el control **Label**, el control **Line**, el control **Shape** y el control **Timer**.
- **Move** *izquierda*, *superior*, *ancho*, *alto*. Mueve el control por su objeto contenedor. Los argumentos *izquierda* y *superior* determinan la ubicación del control con relación a la esquina superior izquierda de su contenedor, en unidades del sistema de coordenadas del contenedor. Los argumentos *ancho* y *alto* indican los nuevos alto y ancho respectivamente del control en twips.
- **Refresh**. Vuelve a dibujar completamente el control para actualizar su contenido.

2.1.4. Funciones.

- **MsgBox(***prompt*[, *buttons*][, *title*]**)**. Presenta un mensaje dentro de un cuadro de diálogo modal y espera a que el usuario haga clic en un botón. Tiene los siguientes argumentos: *prompt* es la cadena que se muestra como mensaje, *buttons* es una expresión numérica que corresponde a la suma de los valores que especifican la apariencia y los botones del cuadro de diálogo, y *title* es la cadena que se muestra en la barra de título. El argumento *buttons* puede tener estos valores:
  - **vbOKOnly**. Muestra solamente el botón *Aceptar*.
  - **vbOKCancel**. Muestra los botones *Aceptar* y *Cancelar*.
  - **vbAbortRetryIgnore**. Muestra los botones *Anular*, *Reintentar* e *Ignorar*.
  - **vbYesNoCancel**. Muestra los botones *Sí*, *No* y *Cancelar*.
  - **vbYesNo**. Muestra los botones *Sí* y *No*.
  - **vbRetryCancel**. Muestra los botones *Reintentar* y *Cancelar*.
  - **vbCritical**. Muestra el icono de *mensaje crítico*.
  - **vbQuestion**. Muestra el icono de *pregunta de advertencia*.

- **vbExclamation**. Muestra el icono de *mensaje de advertencia*.
- **vbInformation**. Muestra el icono de *mensaje de información*.

La función devuelve un tipo **Integer** correspondiente al botón elegido por el usuario, que puede tomar los siguientes valores: **vbOK**, **vbCancel**, **vbAbort**, **vbRetry**, **vbIgnore**, **vbYes** y **vbNo**.

- **InputBox(***prompt*[, *title*][, *default*]**)**. Presenta un mensaje dentro de un cuadro de diálogo modal con un **TextBox** solicitando datos, espera que el usuario escriba un texto o haga clic en un botón y devuelve un tipo **String** con el contenido del cuadro de texto. Tiene los siguientes argumentos: *prompt* es la cadena que se muestra como mensaje, *title* es la cadena que se muestra en la barra de título y *default* es la cadena que se muestra como respuesta predeterminada en el **TextBox**.
- **UBound(***nombre_matriz*[, *dimensión*]**)**. Devuelve el mayor subíndice disponible para la dimensión indicada de una matriz. La primera dimensión se indica con 1, la segunda dimensión se indica con 2 y así sucesivamente. Si *dimensión* se omite, se supone que es 1.
- **LBound(***nombre_matriz*[, *dimensión*]**)**. Devuelve el menor subíndice disponible para la dimensión indicada de una matriz. La primera dimensión se indica con 1, la segunda dimensión se indica con 2 y así sucesivamente. Si *dimensión* se omite, se supone que es 1.
- **LoadPicture(**[*nombreArchivo*]**)**. Aquellos controles que pueden mostrar imágenes, pueden cargarlas en tiempo de ejecución con esta función, siendo *nombreArchivo* una cadena con la ruta completa del archivo de imagen. También puede usarse el valor de la propiedad **Picture** de otro control para presentar o reemplazar una imagen. Para borrar una imagen cargada en un control en tiempo de diseño o en tiempo de ejecución, utilice la función **LoadPicture** sin argumento.
- **TypeName(***nombreVariable***)**. Devuelve una cadena con el nombre del tipo de datos de *nombreVariable*, que puede ser de cualquier tipo excepto de un tipo definido por el usuario. Si *nombrevariable* es una matriz, le añade un paréntesis vacío. Por ejemplo, si *nombrevariable* es una matriz de números enteros, la función **TypeName** devuelve "`Integer()`".
- **Len(***cadena***)**. Devuelve un tipo **Long** que contiene el número de caracteres en una cadena.
- **Asc(***cadena***)**. Devuelve un tipo **Integer** que representa el código ASCII de la primera letra de *cadena*. Si *cadena* no contiene caracteres, se produce un error en tiempo de ejecución.
- **Chr(***códigocar***)**. Devuelve un tipo **String** que contiene el carácter asociado con el código de carácter especificado. El argumento *códigocar* es un tipo **Long** que identifica a un carácter.
- **UCase(***cadena***)**. Devuelve un tipo **String** que contiene la conversión de *cadena* a mayúsculas.
- **LCase(***cadena***)**. Devuelve un tipo **String** que contiene la conversión de *cadena* a minúsculas.
- **Left(***string, length***)** y **Right(***string, length***)**. Devuelve un tipo **String** que contiene un número *length* de caracteres del lado izquierdo (**Left**) y del lado derecho (**Right**) de una cadena *string*.
- **Mid(***string, start[, length*]**)**. Devuelve un tipo **String** que contiene un número *length* de caracteres a partir de la posición *start* de una cadena *string*. Si se omite *length*, se devuelven todos los caracteres desde la posición *start* hasta el final de la cadena *string*.
- **InStr(***[start,]string1, string2[, compare*]**)**. Devuelve un tipo **Long** que especifica la posición de la primera aparición de una cadena *string2* en otra cadena *string1*. El argumento *start* establece la posición inicial para cada búsqueda, que por defecto es el primer carácter; *compare* puede ser **vbBinaryCompare** (comparación binaria) o **vbTextCompare** (comparación de texto).

- **LTrim(***cadena***)**, **RTrim(***cadena***)** y **Trim(***cadena***)**. Devuelve un tipo **String** que contiene una copia de una cadena determinada sin espacios a la izquierda (**LTrim**), sin espacios a la derecha (**RTrim**) o sin espacios ni a la derecha ni a la izquierda (**Trim**).
- **Rnd**. Devuelve un número aleatorio menor que 1, pero mayor o igual que cero. Antes de llamar a la función **Rnd**, utilice la instrucción **Randomize** para inicializar el generador de números aleatorios con un valor de semilla basado en el reloj del sistema.
- **DoEvents( )**. Cede el control de la ejecución al sistema operativo sin perder el enfoque, para que éste pueda procesar los eventos en segundo plano de otras aplicaciones. El control retorna cuando el sistema operativo ha terminado de procesar los eventos en cola. Se debe evitar **DoEvents** cuando se utilizan datos globales, o si existe la posibilidad de que otras aplicaciones interactúen con el procedimiento durante el tiempo en que éste ha cedido el control.
- **Now**. Devuelve un valor de tipo **Date** que especifica la fecha y hora actuales de acuerdo con la configuración de la fecha y la hora del sistema.
- **Time**. Devuelve un valor de tipo **Date** indicando la hora actual del sistema.
- **Format(***expresión*[, *formato*]**)**. Convierte un valor numérico en una cadena de texto y proporciona control sobre la apariencia de la cadena, siendo *expresión* el número que se va a convertir y *formato* una cadena compuesta por símbolos que determinan el formato del número. A continuación se enumeran los símbolos más utilizados:
  - **0** Marcador de posición de dígito; imprime el dígito que corresponda o un cero.
  - **#** Marcador de posición de dígito; imprime el dígito que corresponda o nada.
  - **.** Marcador de posición decimal.
  - **,** Separador de millares.
  - **– + $ ( ) espacio** Carácter literal; se presentan exactamente como están escritos.

  Por ejemplo:
  - **Format**(8315.4, "00000.00") da como resultado "08315.40".
  - **Format**(8315.4, "#####.##") da como resultado "8315.4".
  - **Format**(8315.4, "##,##0.00") da como resultado "8,315.40".
  - **Format**(315.4,"$##0.00") da como resultado "$315.40".

  Para imprimir fechas y horas con formato, utilice símbolos que representen fechas y horas:
  - **Format**(**Now**, "*m/d/aa*") da como resultado 1/27/93
  - **Format**(**Now**, "*dddd, mmmm dd, aaaa*") da como resultado Wednesday, January 27, 1993
  - **Format**(**Now**, "*d-mmm*") da como resultado 27-Jan
  - **Format**(**Now**, "*mmmm-aa*") da como resultado January-93
  - **Format**(**Now**, "*hh:mm* AM/PM") da como resultado 07:18 AM
  - **Format**(**Now**, "*h:mm:ss* a/p") da como resultado 7:18:00 a
  - **Format**(**Now**, "*d-mmmm h:mm*") da como resultado 3-January 7:18
  - **Format**(**Now**, "*ddddd hhhhh*") da como resultado 18/04/2006 18:22:38

  También se pueden especificar los siguientes formatos estándar en el argumento *formato*:
  - `"General Number"`. Muestra los números sin separador de millares.

- `"Currency"`. Muestra los números con separador de millares, si procede; muestra dos dígitos a la derecha del separador decimal.
- `"Standard"`. Muestra los números con separador de millares, al menos un dígito a la izquierda y dos dígitos a la derecha del separador decimal.
- `"Percent"`. Multiplica el valor por 100 con un signo de porcentaje al final.
- `"Scientific"`. Utiliza la notación científica estándar.
- `"General Date"`. Muestra la fecha y la hora si *expresión* contiene ambas. Si *expresión* sólo es una fecha o una hora, no presenta la información que falta.
- `"Long Date"`. Utiliza el formato **Fecha larga** especificado en el sistema del usuario.
- `"Medium Date"`. Utiliza el formato *dd-mmm-aa* (por ejemplo, 03-Abr-93).
- `"Short Date"`. Utiliza el formato **Fecha corta** especificado en el sistema del usuario.
- `"Long Time"`. Muestra la hora según el formato de hora larga del sistema del usuario; incluye horas, minutos y segundos.
- `"Medium Time"`. Muestra la hora, los minutos con el formato "*hh:mm* AM/PM".
- `"Short Time"`. Muestra la hora y los minutos con el formato *hh:mm*.

– **RGB(***red*, *green*, *blue***)**. Devuelve un número entero tipo Long que representa un valor de color RGB. Los argumentos *red*, *green*, *blue* son números enteros entre 0 y 255 que corresponden a la intensidad de los componentes rojo, verde *y* azul. 0 denota la intensidad mínima y 255 la máxima. Cualquier argumento que sea superior a 255 se considerará como 255.

2.2. Controles de texto.

2.2.1. Label.

2.2.1.1. Descripción.

– Los controles **Label** (*lbl*) se utilizan cuando se desea que la aplicación muestre texto que el usuario no pueda modificar directamente en un formulario. En tiempo de ejecución, pueden presentar información como respuesta a un evento o a un proceso de la aplicación.

– Para asignar una tecla de acceso directo a un control sin título, utilice un control **Label** con el control. Como los **Label** no pueden recibir el enfoque, éste pasa automáticamente al siguiente control del orden de tabulación. Utilice esta técnica para asignar teclas de acceso directo a los controles **TextBox**, **PictureBox**, **ComboBox**, **ListBox**, **DriveListBox**, **DirListBox** y **Image**.

2.2.1.2. Propiedades.

– **Caption** [= *cadena*]. Determina el texto que se muestra en el control.

– **BorderStyle** [= *valor*]. Controla el estilo del borde del control. Los valores admitidos son:

- **vbBSNone**. Predeterminado, ningún borde ni elemento relacionado con él.
- **vbFixedSingle**. Simple fijo.

– **BackStyle** [= *número*]. Devuelve o establece un valor que determina si el control es transparente u opaco. Si *número* es 0, el color de fondo y los gráficos son visibles a través del control. Si *número* es 1 (predeterminado), el control se llena con el valor de su propiedad **BackColor**, ocultando los colores o gráficos que haya tras él.

- **AutoSize** [= **True** | **False**]. Determina si se cambia automáticamente el tamaño de la etiqueta. Si tiene el valor **True**, la etiqueta crece horizontalmente para ajustarse a su contenido.
- **WordWrap** [= **True** | **False**]. Hace que la etiqueta crezca verticalmente para ajustarse a su contenido, mientras conserva el mismo ancho. Para que tenga efecto la propiedad **WordWrap** de la etiqueta, **AutoSize** debe tener el valor **True**.
- **Alignment** [= *número*]. Devuelve o establece un valor que determina la alineación del texto del control. Los valores admitidos son:
  - **vbLeftJustify**. Predeterminado, el texto está alineado a la izquierda.
  - **vbRightJustify**. El texto está alineado a la derecha.
  - **vbCenter**. El texto está centrado.

2.2.1.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en el control.
- **DblClick( )**. Ocurre cuando el usuario presiona y suelta dos veces el botón primario del *mouse* en el control. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.
- **Change( )**. Ocurre cuando un vínculo DDE (Dynamic Data Exchange) actualiza los datos o cuando se cambia el valor de la propiedad **Caption** por código.

2.2.2. TextBox.

2.2.2.1. Descripción.

- Los controles **TextBox** (*txt*) permiten obtener información del usuario o mostrar texto. Se usan cuando se quiere mostrar texto que el usuario pueda modificar directamente.
- No se pueden introducir saltos de línea en la ventana Propiedades en tiempo de diseño. Dentro de un procedimiento, para crear un salto de línea hay que usar la constante **vbCrLf** para insertar una combinación de retorno de carro y avance de línea.
- La primera vez que un cuadro de texto recibe el enfoque, el cursor del control **TextBox** está a la izquierda de cualquier texto existente. El usuario puede mover el cursor desde el teclado o con el *mouse*. Si el cuadro de texto pierde el enfoque y lo vuelve a recuperar, el punto de inserción estará donde el usuario lo hubiera dejado la última vez.

2.2.2.2. Propiedades.

- **Text** [= *cadena*]. Devuelve o establece el texto contenido en el área de edición.
- **Locked** [= **True** | **False**]. Devuelve o establece un valor que determina si un control **TextBox** se puede modificar. Si es **True**, el texto del control se puede desplazar y resaltar, pero no se puede modificar. El programa puede cambiar el texto si cambia la propiedad **Text**.
- **MultiLine** [= **True** | **False**]. Con el valor **True**, el control **TextBox** acepta o muestra múltiples líneas de texto en tiempo de ejecución. Ajusta automáticamente el texto a la línea siguiente siempre y cuando no haya una **ScrollBar** horizontal.
- **ScrollBars** [= *valor*]. Devuelve o establece un valor que determina si el control tiene barras de desplazamiento vertical u horizontal. Los valores admitidos son:
  - **vbSBNone**. Predeterminado, ninguna barra de desplazamiento.

- **vbHorizontal**. Barras de desplazamiento horizontal.
- **vbVertical**. Barras de desplazamiento vertical.
- **vbBoth**. Barras de desplazamiento horizontal y vertical.

– **SelStart** [= *índice*]. Devuelve o establece el punto inicial del texto seleccionado, siendo 0 la posición situada más a la izquierda. Si tiene un valor igual o mayor que el número de caracteres que hay en el cuadro de texto, el cursor se situará después del último carácter.

– **SelLength** [= *número*]. Devuelve o establece el número de caracteres seleccionados. Si se asigna un número mayor que 0 se seleccionarán y resaltarán ese número de caracteres a partir del punto de inserción actual indicado por **SelStart**.

– **SelText** [= *valor*]. Devuelve o establece una cadena con el texto seleccionado actualmente o es una cadena de longitud cero ("") si no hay caracteres seleccionados. Si no hay texto seleccionado, **SelText** insertará su texto en el punto de inserción actual.

2.2.2.3. Eventos.

– **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en el control.

– **DblClick( )**. Ocurre cuando el usuario presiona y suelta dos veces el botón primario del *mouse* en el control. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.

– **Change( )**. Ocurre cuando un vínculo DDE (Dynamic Data Exchange) actualiza los datos, cuando un usuario cambia el texto o cuando se cambia el valor de la propiedad **Text** por código.

2.2.3. ListBox.

2.2.3.1. Descripción.

– Los controles **ListBox** (*lst*) se utilizan cuando existe una lista desplegable de opciones entre las que puede elegir el usuario. Las opciones se muestran verticalmente en una única columna, aunque también puede establecer múltiples columnas. Si el número de elementos supera a los que se pueden mostrar, aparecen automáticamente barras de desplazamiento en el control.

– Una práctica recomendada con los eventos de los controles **ListBox**, especialmente cuando aparece como parte de un cuadro de diálogo, es agregar un control **CommandButton** para usarlo con el **ListBox**. El procedimiento de evento **Click** de dicho botón usará la selección del control **ListBox** y ejecutará la acción correspondiente. Hacer doble clic en un elemento de la lista debe tener el mismo efecto que seleccionar el elemento y después hacer clic en el **CommandButton**.

2.2.3.2. Propiedades.

– **List**(*índice*) [= *cadena*]. Devuelve los elementos contenidos en la lista del control. El argumento *índice* permite tener acceso a un elemento de la lista. Un *índice* igual a 0 representa la posición del primer elemento. Si **List**(*índice*) no existe se obtiene una cadena vacía ("").

– **ListCount**. Devuelve el número total de elementos contenidos en la lista del control.

– **ListIndex** [= *índice*]. Devuelve o establece el índice del elemento seleccionado actualmente en el control. Los valores de **ListIndex** están comprendidos entre 0 y **ListCount** – 1, y toma el valor –1 si no hay ningún elemento seleccionado. Al establecer la propiedad **ListIndex**, también se genera un evento **Click** en el control.

- **NewIndex**. Devuelve el índice del último elemento agregado. Devuelve –1 cuando no hay ningún elemento o cuando se ha eliminado un elemento después de agregar el último elemento.
- **Sorted** [= **True** | **False**]. Devuelve un valor que indica si los elementos de un control se colocan automáticamente en orden alfabético sin distinguir entre mayúsculas y minúsculas.
- **Columns** [= *número*]. Devuelve o establece un valor que determina el número de columnas del control. Puede tener los valores siguientes:
  - **0**. Cuadro de lista de una única columna con desplazamiento vertical.
  - **1**. Cuadro de lista de una única columna con desplazamiento horizontal.
  - **>1**. Cuadro de lista de múltiples columnas con desplazamiento horizontal.
- **MultiSelect** [**=** *valor*]. Devuelve o establece un valor que indica si el usuario puede realizar selecciones múltiples. Puede tener los valores siguientes:
  - **MultiSelect** = 0. Predeterminado, sin selección múltiple.
  - **MultiSelect** = 1. Selección múltiple simple.
  - **MultiSelect** = 2. Selección múltiple extendida.
- **Selected(***índice***)** [= **True** | **False**]. Devuelve o establece el estado de selección de un elemento del control. Es una matriz de valores booleanos con el mismo número de elementos que la propiedad **List**. Si está a **False** (predeterminado), el elemento correspondiente a *índice* no está seleccionado. Esta propiedad es útil cuando el usuario puede realizar selecciones múltiples.

2.2.3.3. Eventos.

- **Click( )**. Ocurre cuando el usuario selecciona un elemento del control pulsando las teclas de dirección o haciendo clic con el botón primario del *mouse*.
- **DblClick( )**. Ocurre cuando el usuario selecciona un elemento del control haciendo doble clic con el botón primario del *mouse*. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.

2.2.3.4. Métodos.

- **AddItem** *elemento, índice*. Agrega un nuevo elemento al control, siendo *elemento* la cadena que se agrega a la lista, e *índice* la posición de la lista en la que se insertará. Si se omite *índice* el elemento se inserta al final, o en el orden apropiado si la propiedad **Sorted** es **True**. Si se incluye *índice* con la propiedad **Sorted** a **True** se pueden producir resultados impredecibles.
- **RemoveItem** *índice*. Elimina un elemento del control, siendo *índice* la posición de la lista correspondiente al elemento que se eliminará.
- **Clear**. Borra el contenido del control.

2.2.4. ComboBox.

2.2.4.1. Descripción.

- Los controles **ComboBox** (*cbo*) reúnen las características de un control **TextBox** y un control **ListBox**. El usuario puede elegir de la lista de opciones *sugeridas* o escribir una opción.
- Las opciones se muestran verticalmente en una única columna, aunque también puede establecer múltiples columnas. Si el número de elementos supera a los que se pueden mostrar, aparecen automáticamente barras de desplazamiento horizontales y verticales en el control.

2.2.4.2. Propiedades.

- **Text** [= *cadena*]. Devuelve o establece el texto contenido en el área de edición.
- **Style** [**=** *valor*]. Devuelve o establece la apariencia del control. Los valores admitidos son:
  - **vbComboDropDown**. Predeterminado, especifica un **ComboBox** desplegable en el que el usuario puede escribir texto directamente como en un **TextBox**, o hacer clic en la flecha de la parte derecha del **ComboBox** para abrir una lista de opciones. Si selecciona una de las opciones, se inserta en el **TextBox** de la parte superior del **ComboBox**.
  - **vbComboSimple**. Especifica un **ComboBox** simple en el que la lista se presenta siempre. El usuario también puede escribir texto directamente o seleccionar una opción de la lista.
  - **vbComboDropDownList**. Especifica un **ComboBox** desplegable en el que el usuario debe hacer clic en la flecha de la parte derecha del **ComboBox** para abrir una lista de opciones. En este caso el usuario no puede escribir, sólo puede seleccionar un elemento de la lista.
- **List**(*índice*) [= *cadena*]. Devuelve los elementos contenidos en la lista del control. El argumento *índice* permite tener acceso a un elemento de la lista. Un *índice* igual a 0 representa la posición del primer elemento. Si **List**(*índice*) no existe se obtiene una cadena vacía ("").
- **ListCount**. Devuelve el número total de elementos contenidos en la lista del control.
- **ListIndex** [= *índice*]. Devuelve o establece el índice del elemento seleccionado actualmente en el control. Si *índice* es –1 (predeterminado), significa que no hay ningún elemento seleccionado y que el usuario ha escrito texto nuevo. En otro caso, devuelve el índice del elemento seleccionado. Los valores de **ListIndex** están comprendidos entre 0 y **ListCount** – 1. Al establecer la propiedad **ListIndex**, también se genera un evento **Click** en el control.
- **NewIndex**. Devuelve el índice del último elemento agregado. Devuelve –1 cuando no hay ningún elemento o cuando se ha eliminado un elemento después de agregar el último elemento.
- **Sorted** [= **True** | **False**]. Devuelve un valor que indica si los elementos de un control se colocan automáticamente en orden alfabético sin distinguir entre mayúsculas y minúsculas.

2.2.4.3. Eventos.

- **Click( )**. Ocurre cuando el usuario selecciona un elemento del control presionando las teclas de dirección o haciendo clic con el botón del *mouse*.
- **DblClick( )**. Ocurre cuando el usuario selecciona un elemento del control haciendo doble clic con el botón primario del *mouse*. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.
- **Change( )**. Ocurre sólo si la propiedad **Style** es **vbComboDropDown** o **vbComboSimple**, y el usuario cambia el texto o se cambia el valor de la propiedad **Text** por código.

2.2.4.4. Métodos.

- **AddItem** *elemento, índice*. Agrega un nuevo elemento al control, siendo *elemento* la cadena que se agrega a la lista, e *índice* la posición de la lista en la que se insertará. Si se omite *índice* el elemento se inserta al final, o en el orden apropiado si la propiedad **Sorted** es **True**. Si se incluye *índice* con la propiedad **Sorted** a **True** se pueden producir resultados impredecibles.

- **RemoveItem** *índice*. Elimina un elemento del control, siendo *índice* la posición de la lista correspondiente al elemento que se eliminará.
- **Clear**. Borra el contenido del control.

2.3. Controles de opciones.

2.3.1. CommandButton.

2.3.1.1. Descripción.

- El control **CommandButton** (*cmd*) se utiliza para iniciar, interrumpir o terminar un proceso.

2.3.1.2. Propiedades.

- **Value** [= **True** | **False**]. Devuelve o establece un valor que determina si se ha elegido el botón.
- **Caption** [= *cadena*]. Determina el texto que se muestra en el control.
- **Default** [= **True** | **False**]. Si tiene el valor **True**, al presionar ENTRAR se elige el botón y se desencadena el evento **Click**, incluso aunque cambie el enfoque a un control diferente.
- **Cancel** [= **True** | **False**]. Si tiene el valor **True**, al presionar ESC se elige el botón y se desencadena el evento **Click**, incluso aunque cambie el enfoque a un control diferente.
- **Style** [= *valor*]. Devuelve o establece la apariencia del control. Los valores admitidos son:
  - **vbButtonStandard**. Predeterminado, el control aparece con un estilo estándar.
  - **vbButtonGraphical**. El control se muestra con un estilo gráfico.
- **Picture** [= *imagen*]. Devuelve o establece una imagen que se mostrará en el control. Contiene el nombre de archivo de imagen que se desea mostrar, que se guarda y se carga con el formulario.
- **DownPicture** [= *imagen*]. Hace referencia a una imagen que se muestra cuando el control está presionado y la propiedad **Style** es **vbButtonGraphical**. Si no se asigna ninguna imagen a esta propiedad cuando se presiona el control, se usa la imagen asignada a la propiedad **Picture**.
- **DisabledPicture** [= *imagen*]. Hace referencia a una imagen que se muestra cuando el control está desactivado y la propiedad **Style** es **vbButtonGraphical**. Si no se asigna ninguna imagen a esta propiedad cuando se presiona el control, se usa la imagen asignada a la propiedad **Picture**.

2.3.1.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta el botón primario del *mouse* en el control, o establece el valor de su propiedad **Value** a **True**.

2.3.2. CheckBox.

2.3.2.1. Descripción.

- Los controles **CheckBox** (*chk*) se utilizan cuando existe un conjunto pequeño de opciones entre las que el usuario puede elegir una o más. Presentan una marca de verificación al activarse. Se suelen usar para presentar al usuario una opción de tipo Sí o No, o Verdadero o Falso.
- Un control **CheckBox** indica si una condición determinada está activada o desactivada, y funciona independientemente del resto de controles **CheckBox**. El usuario puede activar cualquier número de controles **CheckBox** al mismo tiempo.

2.3.2.2. Propiedades.

- **Value** [= *valor*]. Devuelve o establece el estado del control. Los valores admitidos son:
  - **vbUnchecked**. Predeterminado, desactivado.
  - **vbChecked**. Activado.

    - **vbGrayed**. Deshabilitado.
- **Caption** [= *cadena*]. Determina el texto que se muestra en el control.
- **Style** [= *valor*]. Devuelve o establece la apariencia del control. Los valores admitidos son:
    - **vbButtonStandard**. Predeterminado, el control aparece con un estilo estándar.
    - **vbButtonGraphical**. El control se muestra con un estilo gráfico.
- **Picture** [= *imagen*]. Devuelve o establece una imagen que se mostrará en el control. Contiene el nombre de archivo de imagen que se desea mostrar, que se guarda y se carga con el formulario.
- **DownPicture** [= *imagen*]. Hace referencia a una imagen que se muestra cuando el control está presionado y la propiedad **Style** es **vbButtonGraphical**. Si no se asigna ninguna imagen a esta propiedad cuando se presiona el control, se usa la imagen asignada a la propiedad **Picture**.
- **DisabledPicture** [= *imagen*]. Hace referencia a una imagen que se muestra cuando el control está desactivado y la propiedad **Style** es **vbButtonGraphical**. Si no se asigna ninguna imagen a esta propiedad cuando se presiona el control, se usa la imagen asignada a la propiedad **Picture**.

2.3.2.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta el botón primario del *mouse* en el control, o cambia el valor de su propiedad **Value**.

2.3.3. OptionButton.

2.3.3.1. Descripción.

- Los controles **OptionButton** (*opt*) se utilizan cuando existe un conjunto pequeño de opciones entre las que el usuario sólo puede elegir una. Presentan al usuario un conjunto de dos o más opciones y deben funcionar siempre como parte de un grupo independiente; al activar un botón de opción se desactivan inmediatamente todos los demás botones del grupo.
- Todos los controles **OptionButton** que haya dentro de un control **Frame** o de un control **PictureBox** determinado constituyen un grupo independiente, y cada uno de ellos tiene un nombre que se utilizará para referenciarlos en el código. Dibuje siempre primero el control **Frame** o el control **PictureBox**, y después dibuje encima los controles **OptionButton**.
- Un grupo de controles **OptionButton** tiene una única tabulación. El botón cuya propiedad **Value** es **True** tiene la propiedad **TabStop** establecida automáticamente como **True**, mientras que los demás botones tienen la propiedad **TabStop** como **False**.

2.3.3.2. Propiedades.

- **Value** [= **True** | **False**]. Devuelve o establece un valor que determina si se ha elegido el botón. Al establecer a **True** la propiedad **Value**, el botón correspondiente permanecerá seleccionado hasta que un usuario seleccione otro botón diferente o hasta que lo modifique el código.
- **Caption** [= *cadena*]. Determina el texto que se muestra en el control.
- **Style** [= *valor*]. Devuelve o establece la apariencia del control. Los valores admitidos son:
    - **vbButtonStandard**. Predeterminado, el control aparece con un estilo estándar.
    - **vbButtonGraphical**. El control se muestra con un estilo gráfico.
- **Picture** [= *imagen*]. Devuelve o establece una imagen que se mostrará en el control. Contiene el nombre de archivo de imagen que se desea mostrar, que se guarda y se carga con el formulario.

- **DownPicture** [= *imagen*]. Hace referencia a una imagen que se muestra cuando el control está presionado y la propiedad **Style** es **vbButtonGraphical**. Si no se asigna ninguna imagen a esta propiedad cuando se presiona el control, se usa la imagen asignada a la propiedad **Picture**.
- **DisabledPicture** [= *imagen*]. Hace referencia a una imagen que se muestra cuando el control está desactivado y la propiedad **Style** es **vbButtonGraphical**. Si no se asigna ninguna imagen a esta propiedad cuando se presiona el control, se usa la imagen asignada a la propiedad **Picture**.

2.3.3.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta el botón primario del *mouse* en el control, o establece el valor de su propiedad **Value** a **True**.
- **DblClick( )**. Ocurre cuando el usuario selecciona un elemento del control haciendo doble clic con el botón primario del *mouse*. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.

2.3.4. Menu.

2.3.4.1. Descripción.

- La *barra de menús* aparece en el formulario inmediatamente debajo de la *barra de título* y contiene uno o más *títulos de menús*. Cuando hace clic en un título de menú, se despliega un menú que contiene una lista de elementos de menú. Los elementos de menú pueden incluir comandos, barras de separación y títulos de submenús. Cada elemento de menú que ve el usuario corresponde a un control de menú definido en el **Editor de menús**.
- Algunos elementos de menú (*mnu*) realizan una acción directamente. Otros muestran una ventana que requiere que el usuario proporcione información que la aplicación necesita para realizar la acción. Estos elementos de menú deben ir seguidos de puntos suspensivos (…).
- Cuando el usuario elige un elemento de menú, se produce un evento **Click**, por tanto es necesario escribir un procedimiento de evento **Click** para cada uno. Todos los controles de menús activados y visibles, excepto las barras separadoras, reconocen el evento **Click**.

2.3.4.2. Propiedades.

- **Caption** [= *cadena*]. Determina el texto que se muestra en el control. Asignando un guión (-), aparece como una barra separadora que divide los elementos del menú en grupos lógicos.
- **Checked** [= **True** | **False**]. Devuelve o establece un valor que determina si se muestra (**True**) o no (**False** predeterminado) una marca de verificación junto a un elemento de menú.

2.3.4.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta el botón primario del *mouse* en el control.

2.4. Controles gráficos.

2.4.1. PictureBox.

2.4.1.1. Descripción.

- El control **PictureBox** (*pct*) sirve para mostrar una imagen, que puede ser un mapa de bits (*.bmp*), un icono (*.ico*), un metarchivo (*.wmf*), un archivo GIF o un archivo JPEG.
- Son contenedores de otros controles. Los controles contenidos se mueven con el **PictureBox**, y sus propiedades **Top** y **Left** serán relativas al **PictureBox** en lugar de ser relativas al formulario.

Se utiliza para crear barras de herramientas o barras de estado, situando controles **Image** para que actúen como botones o agregando controles **Label** para presentar mensajes de estado.

- Sirven para utilizar métodos gráficos (**Circle**, **Line**, **Point** y **Pset**) o de impresión (**Print** y **Cls**).

2.4.1.2. Propiedades.

- **Picture** [= *imagen*]. Devuelve o establece una imagen que se mostrará en el control. Contiene el nombre de archivo de imagen que se desea mostrar, que se guarda y se carga con el formulario.
- **AutoSize** [= **True** | **False**]. Devuelve o establece un valor que determina si el tamaño de un control cambia automáticamente para presentar todo su contenido. El valor **True** hace que cambie de tamaño automáticamente para coincidir con las dimensiones de su contenido, mientras que el valor **False** (el predeterminado) hace que la imagen se recorte.
- **Align** [= *número*]. Devuelve o establece un valor que determina si un objeto se presenta en cualquier tamaño y posición dentro de un formulario o si se presenta en la parte superior, inferior, izquierda o derecha del mismo y si su tamaño se ajusta automáticamente al ancho del formulario. Es útil para crear barras de herramientas o de estado. Los valores admitidos son:
  - **vbAlignNone**. Predeterminado, el tamaño y la posición pueden establecerse en tiempo de diseño o en el código. Este valor se pasa por alto si el objeto está en un formulario MDI.
  - **vbAlignTop**. Predeterminado en formularios MDI, el control está en la parte superior del formulario y su ancho es igual al valor de la propiedad **ScaleWidth** del formulario.
  - **vbAlignBottom**. El control está en la parte inferior del formulario y su ancho es igual al valor de la propiedad **ScaleWidth** del formulario.
  - **vbAlignLeft**. El control está en la parte izquierda del formulario y su ancho es igual al valor de la propiedad **ScaleWidth** del formulario.
  - **vbAlignRight**. El control está en la parte derecha del formulario y su ancho es igual al valor de la propiedad **ScaleWidth** del formulario.
- **FontTransparent** [= **True** | **False**]. Determina si el texto y los gráficos de fondo se muestran a través del texto presentado en el control (**True** predeterminado) o no están ocultos (**False**).

2.4.1.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en el control.
- **DblClick( )**. Ocurre cuando el usuario presiona y suelta dos veces el botón primario del *mouse* en el control. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.
- **Change( )**. Ocurre cuando un vínculo DDE (Dynamic Data Exchange) actualiza los datos o cuando se cambia el valor de la propiedad **Picture** por código.
- **Paint( )**. Sucede cuando el control entero o una parte del mismo se expone después de haberse movido o ampliado, o después de haberse movido una ventana que lo estaba cubriendo. Es el lugar ideal para colocar los métodos gráficos del control cuando la propiedad **AutoRedraw** del control es **False**.
- **Resize(.)**. Se desencadena siempre que se cambia el tamaño del control, ya sea por una acción del usuario o a través del código. Esto permite mover o cambiar el tamaño de los controles que

contiene. Si la propiedad **AutoRedraw** está a **False** y el formulario cambia de tamaño, se llama a los eventos **Resize(.)** y **Paint( )**, en este orden.

2.4.2. Image.

2.4.2.1. Descripción.

- El control **Image** (*img*) es similar al **PictureBox**, pero sólo se usa para mostrar imágenes. No tiene capacidad para actuar como contenedor de otros controles ni admite los métodos avanzados del **PictureBox**. Requiere menos recursos del sistema y se muestra más deprisa que éste.

2.4.2.2. Propiedades.

- **Picture** [= *imagen*]. Devuelve o establece una imagen que se mostrará en un control. Contiene el nombre de archivo de imagen que se desea mostrar, que se guarda y se carga con el formulario.
- **Stretch** [= **True** | **False**]. Devuelve o establece un valor que indica si una imagen cambia su tamaño para ajustarse al del control. El valor **True** hace que la imagen se ajuste al tamaño del control, lo que puede hacer que la imagen aparezca distorsionada, mientras que el valor **False** (el predeterminado) hace que el control se ajuste a las dimensiones de la imagen.
- **BorderStyle** [= *valor*]. Controla el estilo del borde del control. Los valores admitidos son:
  - **vbBSNone**. Predeterminado, ningún borde ni elemento relacionado con él.
  - **vbFixedSingle**. Simple fijo.

2.4.2.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en el control.
- **DblClick( )**. Ocurre cuando el usuario presiona y suelta dos veces el botón primario del *mouse* en el control. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.

2.4.3. Line.

2.4.3.1. Descripción.

- El control **Line** (*lin*) se utiliza para crear segmentos de línea en un formulario, un control **Frame** o un control **PictureBox**. Tiene una funcionalidad limitada y está pensado para usos sencillos, como presentar e imprimir. Para usos más avanzados debe usar el método **Line**.

2.4.3.2. Propiedades.

- **X1** [= *valor*], **Y1** [= *valor*], **X2** [= *valor*] y **Y2** [= *valor*]. Devuelven o establecen las coordenadas del punto inicial (**X1**, **Y1**) y del punto final (**X2**, **Y2**) del control con relación a la esquina superior izquierda de su contenedor, en unidades del sistema de coordenadas del contenedor. Las coordenadas horizontales son **X1** y **X2**, y las verticales son **Y1** e **Y2**.

2.4.4. Shape.

2.4.4.1. Descripción.

- El control **Shape** (*shp*) se utiliza para crear una serie de formas predefinidas en un formulario, un control **Frame** o un control **PictureBox**. Estas formas son: rectángulo, cuadrado, elipse, círculo, rectángulo redondeado y cuadrado redondeado. Tiene una funcionalidad limitada y está pensado para usos sencillos. Para usos más avanzados debe usar los métodos **Line** y **Circle**.

2.4.4.2. Propiedades.

- **Shape** [= *valor*]. Devuelve o establece la apariencia del control. Los valores admitidos son:
  - **vbShapeRectangle**. Predeterminado, rectángulo.
  - **vbShapeSquare**. Cuadrado.
  - **vbShapeOval**. Elipse.
  - **vbShapeCircle**. Círculo.
  - **vbShapeRoundedRectangle**. Rectángulo redondeado.
  - **vbShapeRoundedSquare**. Cuadrado redondeado.
- **BorderStyle** [= *valor*]. Controla el estilo de línea del control. Los valores admitidos son:
  - **vbTransparent**. Transparente.
  - **vbBSSolid**. Predeterminado, continua.
  - **vbBSDash**. Guión.
  - **vbBSDot**. Punto.
  - **vbBSDashDot**. Guión-punto.
  - **vbBSDashDotDot**. Guión-punto-punto.
  - **vbBSInsideSolid**. Continua interna.
- **BackStyle** [= *número*]. Devuelve o establece un valor que determina si el control es transparente u opaco. Si *número* es 0, el color de fondo y los gráficos son visibles a través del control. Si *número* es 1 (predeterminado), el control se llena con el valor de su propiedad **BackColor**, ocultando los colores o gráficos que haya tras él.

2.5. Otros controles.

2.5.1. Frame.

2.5.1.1. Descripción.

- El control **Frame** (*fra*) proporciona un agrupamiento identificable para controles. También puede usar un control **Frame** para subdividir un formulario funcionalmente; por ejemplo, para separar grupos de controles **OptionButton**.
- Para agrupar controles, dibuje primero el control **Frame** y, a continuación, dibuje los controles dentro del **Frame**. De este modo podrá mover al mismo tiempo el **Frame** y los controles que contiene. Si dibuja un control fuera del **Frame** y después intenta moverlo dentro de éste, el control se colocará sobre el **Frame** y deberá mover el **Frame** y los controles por separado.

2.5.1.2. Propiedades.

- **Caption** [= *cadena*]. Determina el texto que se muestra en el control.
- **BorderStyle** [= *valor*]. Controla el estilo del borde del control. Los valores admitidos son:
  - **vbBSNone**. Predeterminado, ningún borde ni elemento relacionado con él.
  - **vbFixedSingle**. Simple fijo.

2.5.1.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en el control.
- **DblClick( )**. Ocurre cuando el usuario presiona y suelta dos veces el botón primario del *mouse* en el control. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el

objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.

2.5.2. HScrollBar y VScrollBar.

2.5.2.1. Descripción.

- Los controles **HScrollBar** (*hsb*) y **VScrollBar** (*vsb*) indican la posición actual en una escala, y pueden utilizarse para controlar la entrada de datos. Son controles diferentes a las barras de desplazamiento adjuntas a los controles **TextBox**, **ListBox** y **ComboBox**.
- La posición actual en una escala viene dada por el movimiento del cuadro de desplazamiento a lo largo de la barra de desplazamiento. El valor devuelto depende de las propiedades del control.
- Cuando la posición del cuadro de desplazamiento tiene el valor *mínimo*, éste se mueve a la posición situada más a la izquierda (en las barras de desplazamiento horizontal) o a la posición más alta (en las barras de desplazamiento vertical). Cuando el cuadro de desplazamiento tiene el valor *máximo*, se mueve a la posición situada más a la derecha o a la más baja.

2.5.2.2. Propiedades.

- **Value** [= *valor*]. Devuelve o establece el valor de la posición del cuadro de desplazamiento. De forma predeterminada es 0, y como máximo puede oscilar entre –32.768 y 32.767.
- **Min** [**=** *valor*]. Devuelve o establece el valor mínimo de la propiedad **Value** cuando el cuadro de desplazamiento se encuentra en el extremo superior o izquierdo. De forma predeterminada es 0, y como máximo puede oscilar entre –32.768 y 32.767.
- **Max** [**=** *valor*]. Devuelve o establece el valor máximo de la propiedad **Value** cuando el cuadro de desplazamiento se encuentra en el extremo inferior o derecho. De forma predeterminada es 32767, y como máximo puede oscilar entre –32.768 y 32.767.
- **LargeChange** [= *número*]. Devuelve o establece el cambio que se producirá en el valor de la propiedad **Value** cuando el usuario haga clic en el área situada entre el cuadro de desplazamiento y la flecha de desplazamiento. De forma predeterminada, está establecida a 1.
- **SmallChange** [= *número*]. Devuelve o establece el cambio que se producirá en el valor de la propiedad **Value** cuando el usuario haga clic en la flecha de desplazamiento. De forma predeterminada, está establecida a 1.

2.5.2.3. Eventos.

- **Scroll( )**. Ocurre al mover el cuadro de desplazamiento. No ocurre si se hace clic en las flechas o en la barra de desplazamiento. Proporciona acceso al valor de la barra de desplazamiento mientras se arrastra el cuadro de desplazamiento.
- **Change()**. Ocurre después de mover el cuadro de desplazamiento. Se produce después de liberar el cuadro de desplazamiento o cuando se hace clic en la barra o en las flechas de desplazamiento.

2.5.3. DriveListBox.

2.5.3.1. Descripción.

- El control **DriveListBox** (*drv*) tiene la apariencia de un control **ComboBox**. Proporciona una lista desplegable de unidades entre las que el usuario puede elegir. Cuando este control tiene el enfoque, el usuario puede escribir cualquier unidad válida, o hacer clic en la flecha de la derecha de la lista desplegando todas las unidades válidas disponibles.

- La elección de una unidad en el cuadro de lista no cambia automáticamente la unidad de trabajo actual en el sistema. Esto debe hacerse mediante la instrucción **ChDrive** *unidad*, donde *unidad* es una cadena que especifica una unidad de disco existente (`ChDrive Drive1.Drive`).

2.5.3.2. Propiedades.

- **Drive** [= *unidad*]. Devuelve o establece la unidad seleccionada en tiempo de ejecución. El valor predeterminado es la unidad actual. Cuando se establece esta propiedad sólo es significativo el primer carácter de la cadena sin distinguir entre mayúsculas y minúsculas.

2.5.3.3. Eventos.

- **Change( )**. Se produce cuando cambia la unidad seleccionada. Ocurre cuando el usuario selecciona una nueva unidad con un clic del *mouse o* con una tecla de dirección, o cuando cambia el valor de la propiedad **Drive** por código. Para mantener enlazado un control **DriveListBox** con un control **DirListBox** debe codificarse lo siguiente en el procedimiento de evento **Change** del primero:

```
Private Sub Drive1_Change()
      Dir1.Path = Drive1.Drive
End Sub
```

2.5.4. DirListBox.

2.5.4.1. Descripción.

- El control **DirListBox** (*dir*) es similar al control **ListBox**, pero con la capacidad de presentar la estructura de directorios de la unidad actual del sistema, empezando por el directorio raíz.
- La elección de un directorio en el cuadro de lista no cambia automáticamente el directorio actual en el sistema. Esto debe hacerse mediante la instrucción **ChDir** *ruta*, donde *ruta* es una cadena que especifica un directorio (`ChDir Dir1.Path`). La *ruta* puede incluir la unidad de disco. Si ésta no se especifica, se asume la unidad actual. La instrucción **ChDir** cambia el directorio predeterminado, pero no la unidad predeterminada.

2.5.4.2. Propiedades.

- **Path** [= *nombreRuta*]. Devuelve o establece la ruta de acceso, incluida la unidad, del directorio seleccionado en tiempo de ejecución. Por ejemplo C:\Ob o C:\Windows\System.
- **ListCount**. Devuelve el número de subdirectorios bajo el directorio actual en la lista del control.
- **ListIndex** [= *índice*]. Contiene un número entero que permite identificar directorios concretos. El directorio especificado por la propiedad **Path** siempre tiene el valor –1, el directorio inmediatamente superior a él tiene un valor de –2, y así sucesivamente hasta el directorio raíz. El primer subdirectorio del directorio especificado por la propiedad **Path** tiene un valor de 0, el siguiente nivel tiene un valor de 1 y así sucesivamente.

2.5.4.3. Eventos.

- **Click( )**. Ocurre cuando el usuario selecciona un elemento del control pulsando las teclas de dirección o haciendo clic con el botón primario del *mouse*.
- **Change( )**. Se produce cuando cambia el directorio seleccionado. Ocurre cuando el usuario hace doble clic en un nuevo directorio o cuando cambia el valor de la propiedad **Path** por código.

Para mantener enlazado un control **DirListBox** con un control **FileListBox** debe codificarse lo siguiente en el procedimiento de evento **Change** del primero:

```
Private Sub Dir1_Change()
      File1.Path = Dir1.Path
End Sub
```

2.5.5. FileListBox.

2.5.5.1. Descripción.

- El control **FileListBox** (*fil*) también se parece al control **ListBox**, pero con la capacidad de presentar una lista de nombres de archivos del directorio seleccionado.
- Los atributos del archivo seleccionado actualmente se encuentran disponibles a través de las propiedades del control. No se pueden usar estas propiedades para establecer los atributos de los archivos. Esto debe hacerse mediante la instrucción **SetAttr** *pathname*, *attributes* donde *pathname* es una cadena que especifica el nombre del archivo, que puede incluir el directorio y la unidad de disco. El argumento *attributes* puede tomar uno de estos valores:
  - **vbNormal**. Normal (predeterminado).
  - **vbReadOnly**. Sólo lectura.
  - **vbHidden**. Oculto.
  - **vbSystem**. Archivo de sistema.
  - **vbArchive**. El archivo se ha modificado después de efectuarse la última copia de seguridad.

2.5.5.2. Propiedades.

- **Path** [= *nombreRuta*]. Devuelve o establece la ruta de acceso, incluida la unidad, del archivo seleccionado en tiempo de ejecución. Por ejemplo C:\Ob o C:\Windows\System.
- **FileName** [= *nombreRuta*]. Devuelve o establece el nombre del archivo seleccionado en tiempo de ejecución, donde *nombreRuta* puede incluir una unidad y una ruta. En este caso provoca el cambio del valor de la propiedad **Path**. Para tener la ruta completa del archivo:

```
Dim rutaCompleta As String
rutaCompleta = File1.Path & "\" & File1.FileName
```

- **Pattern** [= *valor*]. Devuelve o establece un valor que indica los nombres de archivo que muestra el control. El argumento *valor* es una cadena que puede aceptar una lista de patrones delimitada por caracteres de punto y coma (;). Admite los caracteres comodín ? y *.
- **Archive** [= **True** | **False**]. Devuelve o establece un valor que permite mostrar (**True** predeterminado) o no (**False**) los archivos modificados después de efectuarse la última copia de seguridad.
- **Normal** [= **True** | **False**]. Devuelve o establece un valor que permite mostrar (**True** predeterminado) o no (**False**) los archivos normales.
- **System** [= **True** | **False**]. Devuelve o establece un valor que permite mostrar (**True**) o no (**False** predeterminado) los archivos de sistema.
- **Hidden** [= **True** | **False**]. Devuelve o establece un valor que permite mostrar (**True**) o no (**False** predeterminado) los archivos ocultos.

- **ReadOnly** [= **True** | **False**]. Devuelve o establece un valor que permite mostrar (**True** predeterminado) o no (**False**) los archivos de sólo lectura.
- **MultiSelect** [= *valor*]. Devuelve o establece un valor que indica si el usuario puede realizar selecciones múltiples. Puede tener los valores siguientes:
  - **MultiSelect** = 0. Predeterminado, sin selección múltiple.
  - **MultiSelect** = 1. Selección múltiple simple.
  - **MultiSelect** = 2. Selección múltiple extendida.
- **Selected(***índice***)** [= **True** | **False**]. Devuelve o establece el estado de selección de un elemento del control. Es una matriz de valores booleanos con el mismo número de elementos que la propiedad **List**. Si está a **False** (predeterminado), el elemento correspondiente a *índice* no está seleccionado. Esta propiedad es útil cuando el usuario puede realizar selecciones múltiples.

2.5.5.3. Eventos.

- **Click( )**. Ocurre cuando el usuario selecciona un elemento del control pulsando las teclas de dirección o haciendo clic con el botón primario del *mouse*.
- **DblClick( )**. Ocurre cuando el usuario selecciona un elemento del control haciendo doble clic con el botón primario del *mouse*. Si **DblClick** no se produce dentro del límite de tiempo de doble clic del sistema, el objeto reconoce otro evento **Click**. Si hay código en el evento **Click**, nunca se producirá el evento **DblClick**, ya que **Click** es el primero que se activa.
- **PathChange( )**. Se produce cuando cambia el archivo o el directorio seleccionado. Ocurre cuando el usuario hace doble clic en un nuevo archivo, o cuando cambia el valor de la propiedad **FileName** o la propiedad **Path** por código.

2.5.6. Timer.

2.5.6.1. Descripción.

- El control **Timer** (*tmr*) se usa para generar un evento a intervalos periódicos. Debe estar asociado a un formulario, y no es visible en tiempo de ejecución. Su posición y su tamaño no tienen relevancia. Es útil para ejecutar código sin que sea necesaria la actuación del usuario.
- La duración del intervalo depende de la precisión que desee. Como existe cierta posibilidad de errores, haga que la precisión del intervalo sea el doble de la deseada. Cuanto mayor es la frecuencia del evento **Timer**, más tiempo del procesador se utiliza en las respuestas al evento. Esto puede afectar al rendimiento en general.

2.5.6.2. Propiedades.

- **Enabled** [= **True** | **False**]. Devuelve o establece un valor que determina si el control empieza a funcionar en cuanto se carga el formulario (**True**) o se suspende el funcionamiento (**False**).
- **Interval** [= *milisegundos*]. Especifica el número de milisegundos transcurridos entre un evento del control y el siguiente. El intervalo puede estar entre 0 y 64.767, ambos inclusive. No se garantiza que el intervalo tenga siempre la misma duración. Si el sistema está muy cargado, es posible que la aplicación no reciba los eventos **Timer** con la frecuencia especificada.

2.5.6.3. Eventos.

- **Timer( ).** Ocurre cuando ha transcurrido el intervalo preestablecido en la propiedad **Interval**. En el procedimiento de evento correspondiente debe indicarse qué hacer cada vez que se agote el intervalo de tiempo.

2.5.7. CommonDialog (ActiveX).

2.5.7.1. Descripción.

- El control **CommonDialog** (*dlg*) proporciona un conjunto de cuadros de diálogo estándar para operaciones como abrir y guardar archivos, establecer opciones de impresión, seleccionar colores y fuentes, y presentar Ayuda mediante el sistema de Ayuda de Windows.
- Puede usarlo en una aplicación si lo agrega a un formulario en tiempo de diseño y establece sus propiedades. En tiempo de ejecución es invisible y sólo aparece al invocar el método apropiado.

2.5.7.2. Propiedades.

- **CancelError** [= **True** | **False**]. Devuelve o establece un valor que indica si se generará un error cuando el usuario haga clic en el botón **Cancelar**. Si tiene el valor **True**, puede detectar que se hizo **Click** en el botón **Cancelar** si intercepta el error (`On Error GoTo Etiqueta`) durante la presentación del cuadro de diálogo. Si es **False** (predeterminado) no se genera ningún error.

2.5.7.3. Métodos.

- **ShowOpen**. El cuadro de diálogo **Abrir** permite que el usuario especifique una unidad, un directorio, una extensión de archivo y un nombre de archivo. Se utiliza de la siguiente manera:
  - Especifique la lista de filtros de archivo que aparecen en *Tipos de archivo*. Para hacer esto, establezca la propiedad **Filter** con el siguiente formato:

    descripción1 | *filtro1* | *descripción2* | *filtro2*...
  - *Descripción* es la cadena que se presenta en el cuadro de lista, por ejemplo "Archivos de texto (*.txt)". *Filtro* es el filtro de archivos real, por ejemplo "*.txt". Cada pareja *descripción* | *filtro* tiene que estar separada por el símbolo de canalización (|). Se puede especificar el filto predeterminado mediante la propiedad **FilterIndex**, asignándole el número de la pareja *descripción* | *filtro* por defecto deseada.
  - Utilice el método **ShowOpen** para presentar el cuadro de diálogo.
  - Utilice la propiedad **FileName** para obtener el nombre del archivo seleccionado.
- **ShowSave**. El cuadro de diálogo **Guardar como** es idéntico en apariencia al cuadro de diálogo **Abrir**, excepto en el título y en que los nombres de archivo aparecen atenuados. En tiempo de ejecución, cuando el usuario elige un archivo y cierra el cuadro de diálogo, se utiliza la propiedad **FileName** para obtener el nombre del archivo seleccionado.
- **ShowColor**. El cuadro de diálogo **Color** permite al usuario seleccionar un color de una paleta o crear y seleccionar un color personalizado. Se utiliza de la siguiente manera:
  - Establezca la propiedad **Flags** del control **CommonDialog** a la constante **cdlCCRGBInit**.
  - Utilice el método **ShowColor** para presentar el cuadro de diálogo.
  - Utilice la propiedad **Color** para obtener el valor RGB del color seleccionado por el usuario.
- **ShowFont**. El cuadro de diálogo **Fuente** permite que el usuario seleccione una fuente por el tamaño, el color y el estilo. Se utiliza de la siguiente manera:

- Establezca la propiedad **Flags** a uno de los siguientes valores de constantes:
  - **cdlCFScreenFonts** (fuentes de pantalla).
  - **cdlCFPrinterFonts** (fuentes de impresora).
  - **cdlCFBoth** (fuentes de pantalla y de impresora)
- Utilice el método **ShowFont** para presentar el cuadro de diálogo.
- Utilice las siguientes propiedades acerca de la fuente seleccionada:
  - **Color**. Determina el color seleccionado. Para usar esta propiedad, primero tiene que establecer la propiedad **Flags** a **cdlCFEffects** (`cdlCFBoth Or cdlCFEffects`).
  - **FontBold**. Determina si se ha seleccionado negrita.
  - **FontItalic**. Determina si se ha seleccionado cursiva.
  - **FontStrikethru**. Determina si se ha seleccionado tachado. Para usar esta propiedad, primero tiene que establecer la propiedad **Flags** a **cdlCFEffects**.
  - **FontUnderline**. Determina si se ha seleccionado subrayado. Para usar esta propiedad, primero tiene que establecer la propiedad **Flags** a **cdlCFEffects**.
  - **FontName**. Determina el nombre de la fuente seleccionada.
  - **FontSize**. Determina el tamaño de fuente seleccionado.

– **ShowPrinter**. El cuadro de diálogo **Imprimir** permite que el usuario especifique cómo va a imprimir, no envía realmente datos a la impresora. Debe escribir código para imprimir los datos en el formato que los usuarios seleccionan. Se utiliza de la siguiente manera:
- Utilice el método **ShowPrinter** para presentar el cuadro de diálogo.
- Utilice las siguientes propiedades acerca de la selección del usuario:
  - **Copies**. Determina el número de copias que se van a imprimir.
  - **FromPage**. Determina la página por la que se empieza a imprimir.
  - **ToPage**. Determina la página en la que termina la impresión.
  - **hDC**. Determina el contexto de dispositivo de la impresora seleccionada.
  - **Orientation**. Determina la orientación de la página (horizontal o vertical).

– **ShowHelp**. Permite presentar un archivo de Ayuda. Establezca las propiedades **HelpCommand** y **HelpFile**, y utilice el método **ShowHelp** para presentar el archivo de Ayuda especificado.

2.6. Matrices de controles.

2.6.1. Creación de matrices de controles.

– Una *matriz de controles* es un grupo de controles del mismo tipo que comparten el mismo nombre y los mismos procedimientos de evento. Son útiles cuando varios controles comparten código. Cada elemento de una matriz de controles tienen su propios valores de propiedades.

– Las matrices de controles tienen al menos un elemento y pueden crecer hasta contener tantos elementos como permitan los recursos del sistema. Si desea crear una instancia nueva de un control en tiempo de ejecución, dicho control tiene que ser miembro de una matriz de controles. Los nuevos elementos heredan los procedimientos de evento comunes de toda la matriz.

– Hay tres maneras de crear una matriz de controles en tiempo de diseño:
- Asignar el mismo nombre a más de un control.
- Copiar un control existente y después pegarlo en el formulario.

  - Establecer la propiedad **Index** del control a un valor distinto de **Null**.
- Los procedimientos de evento de una matriz de controles tienen el argumento añadido *Index* **As Integer**, por el cual es posible identificar de manera única al control que ha recibido el evento.
- Puede agregar y quitar controles de una matriz de controles existente mediante las instrucciones **Load** y **Unload** en tiempo de ejecución. Debe crear un control en tiempo de diseño con la propiedad **Index** establecida a 0. Después, en tiempo de ejecución, utilice la sintaxis siguiente:

  Load *objeto***(***índice%***)**

  Unload *objeto***(***índice%***)**

  siendo *objeto* el nombre de la matriz de controles a la que se va a agregar o eliminar el control, e *índice%* el número de índice del control dentro de la matriz. Visual Basic genera un error si intenta usar la instrucción **Load** con un número de índice ya utilizado en la matriz.
- Puede usar la instrucción **Unload** para quitar cualquier control creado con **Load**, pero no para quitar controles creados en tiempo de diseño, formen parte o no de una matriz de controles.
- Cuando carga un nuevo elemento en una matriz de controles, la mayoría de los valores de sus propiedades se copian del elemento más bajo existente en la matriz (en este caso, el elemento con el valor de índice 0). Los valores de las propiedades **Visible**, **Index** y **TabIndex** no se copian automáticamente a los nuevos elementos de una matriz de controles, de modo que *para hacer visible un control recién agregado*, debe establecer su propiedad **Visible** a **True**.

2.6.2. Matrices de controles de menús.

- Una *matriz de controles de menús* es un conjunto de elementos de menú del mismo menú que comparten el mismo nombre y procedimientos de evento. Se utiliza para simplificar el código, y para crear nuevos elementos de menú en tiempo de ejecución. Por ejemplo, para almacenar una lista de los archivos abiertos recientemente.
- Los elementos de una matriz de controles de menús deben estar contiguos en el cuadro de lista del control de menús y deben tener el mismo nivel de sangría. Cuando vaya a crear matrices de controles de menús, asegúrese de incluir barras separadoras en el menú.
- Cada elemento de la matriz de controles de menús se identifica mediante un valor de índice único, indicado por la propiedad **Index** en el **Editor de menús**. Cuando un miembro de una matriz de controles reconoce un evento, Visual Basic pasa el valor de su propiedad **Index** al procedimiento de evento como un argumento adicional. Si a un control de menús se le asigna un valor a su propiedad **Index** en tiempo de diseño, se convierte automáticamente en un elemento de una matriz de controles, incluso aunque no se hayan creado más elementos.

2.7. Responder a eventos del mouse y del teclado.

2.7.1. Responder a los eventos del mouse.

2.7.1.1. Eventos del mouse.

- Los eventos del *mouse* se utilizan para reconocer y responder a los diversos estados del *mouse* como eventos distintos. Los objetos reconocen estos eventos de la siguiente manera:
  - En el caso de un *formulario*, cuando el puntero está sobre una zona donde no hay controles.
  - En el caso de un *control*, cuando el puntero está sobre el control.

- Cuando el usuario mantiene presionado un botón del *mouse*, el objeto continúa reconociendo todos los eventos del *mouse* hasta que suelta dicho botón, incluso si mueve el puntero fuera del objeto. Los eventos del *mouse* no deben confundirse con los eventos **Click** y **DblClick**, puesto que éstos reconocen cuándo se ha presionado y soltado un botón del *mouse* como una acción.

2.7.1.2. Eventos MouseDown, MouseUp y MouseMove.

- La mayoría de los controles reconocen estos eventos del *mouse*:
  - **MouseDown(**[*Index* **As Integer**,]*Button* **As Integer**, *Shift* **As Integer**, *X* **As Single**, *Y* **As Single)**
  - **MouseUp(**[*Index* **As Integer**,]*Button* **As Integer**, *Shift* **As Integer**, *X* **As Single**, *Y* **As Single)**
  - **MouseMove(**[*Index* **As Integer**,]*Button* **As Integer**, *Shift* **As Integer**, *X* **As Single**, *Y* **As Single)**
- El evento **MouseDown** ocurre cuando el usuario presiona cualquier botón del *mouse*, el evento **MouseUp** sucede cuando el usuario suelta cualquier botón del *mouse* y el evento **MouseMove** ocurre cada vez que el puntero del *mouse* se mueve a una nueva posición de la pantalla.
- Los tres eventos del *mouse* usan los siguientes argumentos:
  - *Index*. Identifica al control que ha recibido el evento si está en una matriz de controles.
  - *Button*. Entero que devuelve el estado de los botones del *mouse* al producirse el evento.
  - *Shift*. Entero que corresponde al estado de las teclas MAYÚS, CTRL y ALT.
  - *X, Y*. Ubicación del puntero en el sistema de coordenadas del objeto que recibe el evento.
- Los eventos **MouseDown**, **MouseUp** y **MouseMove** devuelven el argumento *Button* para indicar qué botón o botones del *mouse* están presionados al producirse el evento. A continuación se muestran los valores que puede tomar este argumento:
  - **vbLeftButton**. Se ha presionado el botón primario.
  - **vbRightButton**. Se ha presionado el botón secundario.
  - **vbMiddleButton**. Se ha presionado el botón central.
- El argumento *Button* se usa con **MouseDown** para determinar qué botón se ha presionado y con **MouseUp** para determinar qué botón se ha soltado. Para estos eventos, el argumento *Button* indica exactamente un botón por evento. Es decir, si se presiona o se suelta más de un botón, Visual Basic interpreta la acción como dos o más eventos **MouseDown** o **MouseUp** distintos.
- En el caso del evento **MouseMove**, el argumento *Button* indica el estado actual de todos los botones. Por tanto, además de los anteriores, puede tomar los siguientes valores:
  - **vbLeftButton** + **vbRightButton**. Se han presionado los botones primario y secundario.
  - **vbLeftButton** + **vbMiddleButton**. Se han presionado los botones primario y central.
  - **vbRightButton** + **vbMiddleButton**. Se han presionado los botones secundario y central.
  - **vbRightButton** + **vbMiddleButton** + **vbLeftButton**. Se han presionado los tres botones.
- El argumento *Shift* indica si las teclas MAYÚS, CTRL y ALT están presionadas al producirse el evento, y en qué combinación. Los valores que puede tomar son los siguientes:
  - **vbShiftMask**. Tecla MAYÚS presionada.
  - **vbCtrlMask**. Tecla CTRL presionada.
  - **vbAltMask**. Tecla ALT presionada.
  - **vbShiftMask** + **vbCtrlMask**. Teclas MAYÚS y CTRL presionadas.
  - **vbShiftMask** + **vbAltMask**. Teclas MAYÚS y ALT presionadas.

- **vbCtrlMask** + **vbAltMask**. Teclas CTRL y ALT presionadas.
- **vbCtrlMask** + **vbAltMask** + **vbShiftMask**. Teclas MAYÚS, CTRL y ALT presionadas.

2.7.1.3. Arrastrar y colocar.

2.7.1.3.1. Descripción.

- Arrastrar es la acción de mantener presionado un botón del *mouse* y mover un control, y colocar es la acción de soltar el botón. Arrastrar un control en tiempo de ejecución no cambia automáticamente su posición, sino que indica que se debe llevar a cabo alguna acción.
- Hay dos términos importantes referidos a las operaciones de arrastrar y colocar:
  - *Origen*. Es el control que se arrastra, que puede ser cualquier objeto excepto un menú o un control **Timer**, **Line** o **Shape**.
  - *Destino*. Es el objeto sobre el que el usuario coloca el control, que puede ser un formulario si el puntero está en una área vacía del formulario, o un control si la posición del *mouse* está dentro de los bordes de dicho control al soltar el botón.
- Todos los controles excepto los menús y los controles **Timer**, **Line** y **Shape** admiten las propiedades **DragMode** y **DragIcon** y el método **Drag**. Los formularios reconocen los eventos **DragDrop** y **DragOver**, pero no admiten las propiedades **DragMode** y **DragIcon** o el método **Drag**. Sólo es posible arrastrar los controles cuando no tienen el enfoque. Para evitar que un control obtenga el enfoque, establezca la propiedad **TabStop** a **False**.

2.7.1.3.2. Propiedades.

- **DragMode** [= *número*]. Devuelve o establece un valor que determina el modo de arrastre en una operación de arrastrar y colocar. Los valores admitidos son:
  - **vbManual**. Predeterminado, requiere el uso del método **Drag** para iniciar una operación de arrastrar y colocar en el control de origen.
  - **vbAutomatic**. Automático, al hacer clic en el control de origen se inicia automáticamente una operación de arrastrar y colocar.
- **DragIcon** [= *icono*]. Establece el icono que aparece como puntero en una operación de arrastrar y colocar. El archivo debe tener la extensión .ico y el formato correspondiente. Normalmente, se define como parte de un procedimiento de evento **MouseDown** o **DragOver**.

2.7.1.3.3. Eventos.

- **DragDrop(**[*Index* **As Integer**,]*Source* **As Control**, *X* **As Single**, *Y* **As Single)**. Ocurre cuando se completa una operación de arrastrar y colocar, al arrastrar un control sobre un objeto y soltar el botón del *mouse*, o usar el método **Drag** con su argumento *acción* establecido a **vbEndDrag**:
  - *Index*. Identifica al control que ha recibido el evento si está en una matriz de controles.
  - S*ource*. Identifica al control colocado sobre el destino, y se puede hacer referencia a sus propiedades o llamar a uno de sus métodos. Se puede usar la función **TypeName** para determinar el tipo de control colocado.
  - *X, Y*. Ubicación del puntero en el sistema de coordenadas del objeto que recibe el evento.
- **DragOver(**[*Index* **As Integer**,]*Source* **As Control**, *X* **As Single**, *X* **As Single**, *State* **As Integer)**. Ocurre cuando una operación de arrastrar y colocar está en curso. Puede usar este evento para controlar el puntero del *mouse* a medida que entra, sale o descansa directamente sobre un destino

válido. El argumento *State* corresponde al estado de transición del control que se está arrastrando en relación al formulario o un control de destino, y puede tomar los siguientes valores:

- **State = 0**. El control origen está entrando en el objeto de destino.
- **State = 1**. El control origen está saliendo del objeto de destino.
- **State = 2**. El control origen está dentro del objeto de destino.

2.7.1.3.4. Métodos.

- **Drag** *acción*. Inicia, termina o cancela de manera manual una operación de arrastrar y colocar. El argumento *acción* puede tomar los siguientes valores:
  - **vbBeginDrag**. Inicia el arrastre del objeto origen.
  - **vbEndDrag**. Termina el arrastre, coloca el objeto origen y produce un evento **DragDrop**.
  - **vbCancel**. Cancela la operación de arrastre y no se desencadena ningún evento **DragDrop**.

2.7.2. Responder a los eventos del teclado.

2.7.2.1. Eventos del teclado.

- Los eventos del teclado se pueden detectar a nivel de:
  - *Control*. Sólo el objeto que tiene el enfoque puede recibir un evento del teclado.
  - *Formulario*. Éste recibe todos los eventos del teclado del control que tiene el enfoque antes de que el control los reconozca. Esto es útil cuando se desea realizar la misma acción cada vez que se presione una tecla determinada, sin tener en cuenta qué control tiene el enfoque.
- Los eventos del teclado no se excluyen mutuamente. Cuando el usuario presiona una tecla se generan los eventos **KeyDown** y **KeyPress**, seguidos del evento **KeyUp** cuando el usuario suelta la tecla. Cuando el usuario presiona una tecla que **KeyPress** no detecta, sólo ocurre un evento **KeyDown** seguido del evento **KeyUp**.
- Los eventos de teclado no se producen cuando el usuario pulsa una tecla de método abreviado para un control de menú. Tampoco cuando un control **CommandButton** tiene su propiedad **Default** a **True** y se pulsa ENTRAR, o su propiedad **Cancel** a **True** y se pulsa ESC. La tecla TAB mueve el enfoque de un control a otro y no desencadena un evento de teclado a menos que todos los controles del formulario estén desactivados o que haya establecido **TabStop** a **False**.

2.7.2.2. Eventos KeyPress, KeyDown y KeyUp.

- **KeyPress(**[*Index* **As Integer**,]*KeyAscii* **As Integer)**. Ocurre cuando el usuario presiona y suelta una tecla correspondiente a un carácter ASCII estándar mientras el control tiene el enfoque. El argumento *KeyAscii* devuelve un valor entero que corresponde a un código de carácter ASCII. Si se modifica el argumento *KeyAscii*, se envía un carácter diferente al control que recibe el evento. Si *KeyAscii* es 0, la pulsación de tecla se cancela y el control no recibe ningún carácter.
- Existen algunas teclas no reconocidas por el evento **KeyPress**, como las teclas de función (F1 a F12), las teclas de edición (INS, SUPR o RETROCESO), las teclas de desplazamiento y cualquier combinación de éstas con modificadores del teclado. Para ello existen dos eventos:
  - **KeyDown(**[*Index* **As Integer**,]*KeyCode* **As Integer**, *Shift* **As Integer)**. Ocurre cuando el usuario presiona una tecla mientas el control tiene el enfoque.
  - **KeyUp(**[*Index* **As Integer**,]*KeyCode* **As Integer**, *Shift* **As Integer)**. Ocurre cuando el usuario suelta una tecla mientas el control tiene el enfoque.

El argumento *KeyCode* devuelve un valor entero que corresponde a un código de tecla física. Los códigos para las teclas de letras son los mismos que los códigos ASCII para el carácter en mayúsculas de esa letra. Los códigos de tecla para las teclas de números y de signos de puntuación son los mismos que el código ASCII del número en la tecla. Si *KeyCode* es 0, la pulsación de tecla se cancela y el control no recibe ningún carácter.

El argumento *Shift* permite determinar si se ha escrito una letra mayúscula o minúscula, o un número o un signo de puntuación. Los valores admitidos son:

- **vbShiftMask**. Tecla MAYÚS presionada.
- **vbCtrlMask**. Tecla CTRL presionada.
- **vbAltMask**. Tecla ALT presionada.
- **vbShiftMask** + **vbCtrlMask**. Teclas MAYÚS y CTRL presionadas.
- **vbShiftMask** + **vbAltMask**. Teclas MAYÚS y ALT presionadas.
- **vbCtrlMask** + **vbAltMask**. Teclas CTRL y ALT presionadas.
- **vbCtrlMask** + **vbAltMask** + **vbShiftMask**. Teclas MAYÚS, CTRL y ALT presionadas.

2.7.2.3. Responder a los eventos a nivel de formulario.

- Para llevar a cabo la detección de eventos del teclado a nivel de formulario, establezca la propiedad **KeyPreview** del formulario a **True**.
- Para controlar los eventos de teclado solamente a nivel de formulario y no permitir que los controles los reciban, establezca *KeyAscii* a 0 en el evento **KeyPress** del formulario, y *KeyCode* a 0 en el evento **KeyDown** del formulario.

3. Texto y gráficos.

3.1. El sistema de coordenadas. Capas de dibujo.

- El sistema de coordenadas es una cuadrícula bidimensional que define ubicaciones en la pantalla, en un formulario, en un control **PictureBox** o en un objeto **Printer**. Las ubicaciones se definen mediante las coordenadas del formulario (*X*, *Y*). El valor de *X* es la ubicación del punto sobre el eje x, y el valor de *Y* es la ubicación del punto sobre el eje y.
- El sistema de coordenadas de Visual Basic tiene las siguientes características:
  - Cuando mueve o cambia el tamaño de un control, utiliza el sistema de coordenadas del contenedor del control, que puede ser un formulario, un **Frame** o un **PictureBox**.
  - Todos los métodos gráficos y el método **Print** utilizan el sistema de coordenadas del contenedor, que de manera predeterminada empieza por la coordenada (0, 0) en la esquina superior izquierda del contenedor, y utiliza *twips* (un centímetro contiene 567 twips).
  - La *escala* define las unidades de medida utilizadas a lo largo de un eje de coordenadas, y cada eje puede tener su propia escala. Se puede modificar la dirección del eje, el punto de inicio y la escala del sistema de coordenadas.
- Las capas de un formulario o de otro contenedor desde el frente al fondo son las siguientes:
  - *Frente*. Controles no gráficos como **CommandButton**, **CheckBox** y **FileListBox**.
  - *Media*. Controles gráficos (**Image**, **Line** y **Shape**) y controles **Label**.
  - *Fondo*. Resultados de los métodos gráficos.

- El contenido de una capa cubre el contenido de la capa inferior, de manera que los gráficos creados con los controles gráficos aparecen detrás de los otros controles del formulario y todos los gráficos creados con los métodos gráficos aparecen bajo todos los controles, gráficos o no.

3.2. Propiedades generales.

3.2.1. Presentación, ubicación y escala del dibujo.

- **AutoRedraw** [= **True** | **False**]. Con el valor predeterminado **False**, los gráficos creados con métodos gráficos en un formulario o un **PictureBox** se pierden si otra ventana los oculta temporalmente, y los procedimientos de evento **Paint** se ejecutan cada vez que se debe volver a dibujar una parte del formulario o del **PictureBox**. Cuando se establece a **True**, se aplican los métodos gráficos a un lienzo en memoria, y la aplicación copia automáticamente el contenido de este lienzo para volver a presentar los gráficos ocultos temporalmente por otra ventana. En el caso de un **PictureBox**, sólo guarda en memoria el contenido visible del control.
- **CurrentX** [= *x*] y **CurrentY** [= *y*]. Devuelven o establecen las coordenadas horizontal y vertical respectivamente de un objeto para el siguiente método gráfico o de texto. Se miden a partir de la esquina superior izquierda, y se expresan en twips o en la unidad de medida actual del objeto.
- **ScaleLeft** [= *valor*] y **ScaleTop** [= *valor*]. Devuelven o establecen valores numéricos enteros, reales, positivos o negativos a la esquina superior izquierda de un objeto. Por ejemplo, para definir la esquina superior izquierda del formulario actual como (100, 100):

```
ScaleLeft = 100
ScaleTop = 100
```

- **ScaleHeight** [= *valor*] y **ScaleWidth** [= *valor*]. Devuelven o establecen valores numéricos enteros, reales, positivos o negativos para las unidades de medida horizontal y vertical respectivamente del área disponible *dentro* de un objeto, a diferencia de **Height** y **Width**, que definen las dimensiones externas del objeto y se expresan siempre en términos del sistema de coordenadas del *contenedor*. Los valores negativos hacen que cambie la orientación del sistema de coordenadas. Por ejemplo, para definir la unidad horizontal como 1/1000 del ancho interno y y la unidad vertical como 1/500 del alto interno del formulario actual:

```
ScaleWidth = 1000
ScaleHeight = 500
```

- **ScaleMode** [= *valor*]. Devuelve o establece un valor que indica la unidad de medida de las coordenadas al usar métodos gráficos o al situar controles. Al establecer **ScaleMode** a un valor distinto de **vbUser**, automáticamente **ScaleHeight** y **ScaleWidth** cambian a la nueva unidad de medida, y **ScaleLeft** y **ScaleTop** se establecen a 0. Los valores admitidos son:
  - **vbUser**. Valor establecido automáticamente que indica que una o más de las propiedades **ScaleHeight**, **ScaleWidth**, **ScaleLeft** y **ScaleTop** tienen valores personalizados.
  - **vbTwips**. Twips. Es la escala predeterminada. En una pulgada hay 1.440 twips.
  - **vbPoints**. Puntos. En una pulgada hay 72 puntos.
  - **vbPixels**. Píxeles. Un píxel es la unidad mínima de resolución de una pantalla o de una impresora. El número de píxeles por pulgada depende de la resolución del dispositivo.
  - **vbCharacters**. Caracteres. Un carácter tiene 1/6 pulgada de alto y 1/12 pulgada de ancho.

- **vbInches**. Pulgadas.
- **vbMillimeters**. Milímetros.
- **vbCentimeters**. Centímetros.

3.2.2. Técnicas de dibujo.

- **DrawMode** [= *número*]. Devuelve o establece un valor que determina la apariencia del resultado de un método gráfico o la apariencia de un control **Shape** o **Line**. El valor predeterminado es **vbCopyPen**, que utiliza el color especificado por la propiedad **ForeColor**.
- **DrawStyle** [= *número*]. Devuelve o establece un valor que determina el estilo de línea del resultado de métodos gráficos. Los valores admitidos son:
  - **vbSolid**. Predeterminado, continua.
  - **vbDash**. Guión.
  - **vbDot**. Punto.
  - **vbDashDot**. Guión-punto.
  - **vbDashDotDot**. Guión-punto-punto.
  - **vbInsideSolid**. Continua interna.
- **DrawWidth** [= *tamaño*]. Devuelve o establece el ancho de línea para el resultado de los métodos gráficos. El valor *tamaño* es una una expresión numérica comprendida entre 1 y 32.767 que representa el ancho de la línea en píxeles. El valor predeterminado es un píxel de ancho.
- **BorderStyle** [= *valor*]. Controla el estilo de línea del control. Los valores admitidos son:
  - **vbTransparent**. Transparente.
  - **vbBSSolid**. Predeterminado, continua.
  - **vbBSDash**. Guión.
  - **vbBSDot**. Punto.
  - **vbBSDashDot**. Guión-punto.
  - **vbBSDashDotDot**. Guión-punto-punto.
  - **vbBSInsideSolid**. Continua interna.
- **BorderWidth** [= *número*]. Devuelve o establece el ancho del borde de un control. Si es mayor que 1, los únicos valores efectivos de **BorderStyle** serán **vbBSSolid** y **vbBSInsideSolid**.

3.2.3. Técnicas de relleno y colores.

- **FillStyle** [= *número*]. Devuelve o establece el patrón usado para llenar los círculos y los cuadros creados con los métodos gráficos **Circle** y **Line**. Los valores admitidos son:
  - **vbFSSolid**. Sólido. Llena el cuadro con el color establecido en la propiedad **FillColor**.
  - **vbFSTransparent**. Predeterminado, transparente. El objeto aparece vacío.
  - **vbHorizontalLine**. Líneas horizontales.
  - **vbVerticalLine**. Líneas verticales.
  - **vbUpwardDiagonal**. Líneas diagonales hacia arriba.
  - **vbDownwardDiagonal**. Líneas diagonales hacia abajo.
  - **vbCross**. Líneas cruzadas.
  - **vbDiagonalCross**. Líneas cruzadas en diagonal.

- **FillColor** [= *color*]. Devuelve o establece el color usado para llenar círculos y cuadros creados con los métodos gráficos **Circle** y **Line**. Los valores admitidos para *color* son colores RGB normales especificados con la paleta de colores, o en el código mediante la función **RGB** o por las constantes de color. De forma predeterminada, **FillColor** está establecido a 0 (negro).
- **BackColor** [= *color*] y **ForeColor** [= *color*]. Devuelve o establece el color de fondo y el color de primer plano utilizado para mostrar texto y gráficos respectivamente en el formulario o en el control. Los valores admitidos para *color* son colores RGB normales especificados con la paleta de colores, o en el código mediante la función **RGB** o por las constantes de color.
- **BorderColor** [= *color*]. Devuelve o establece el color de línea del control. Los valores admitidos para *color* son colores RGB normales especificados con la paleta de colores, o en el código mediante la función **RGB** o por las constantes de color. Cuando **BorderStyle** es **vbTransparent**, se pasa por alto la propiedad **BorderColor**.

3.3. Métodos de texto.

- **Print** [*listaSalida*] [{ ; | , }]. Presenta texto en el formulario en las coordenadas indicadas por las propiedades **CurrentX** y **CurrentY**. El argumento *listaSalida* es el texto que aparece en el formulario, se pueden especificar múltiples elementos en separados por coma, punto y coma o ambos. Con punto y coma (;) se imprime un elemento a continuación de otro sin espacios intermedios, con coma (,) se salta a la siguiente posición de tabulación. De manera predeterminada, el método **Print** imprime el texto y pasa a la línea siguiente.
- **TextHeight(***cadena***)**. Devuelve el alto de una cadena de texto, expresado en unidades del sistema de coordenadas del contenedor, tal y como se imprimiría con la fuente actual del formulario o del control **PictureBox**. Si el argumento *cadena* contiene caracteres de retorno de carro (**Chr**(13)), devuelve el alto de todas las líneas de texto contenidas en la cadena.
- **TextWidth(***cadena***)**. Devuelve el ancho de una cadena de texto, expresado en unidades del sistema de coordenadas del contenedor, tal y como se imprimiría con la fuente actual del formulario o del control **PictureBox**. Si el argumento *cadena* contiene caracteres de retorno de carro (**Chr**(13)), devuelve el ancho de la línea más larga.
- **Cls**. Borra el texto generado en tiempo de ejecución, y restablece **CurrentX** y **CurrentY** a 0.

3.4. Métodos gráficos.

- **Scale (***x1*, *y1***) - (***x2*, *y2***)**. Define el sistema de coordenadas de un objeto, de tal manera que los valores de *x1* y *y1* determinan los valores de las propiedades **ScaleLeft** y **ScaleTop**, mientras que *x1 – x2* y *y1 – y2* determinan los valores de **ScaleWidth** y **ScaleHeight** respectivamente. Especificar un valor de *x1 > x2* o *y1 > y2* tiene el mismo efecto que establecer **ScaleWidth** o **ScaleHeight** a un valor negativo. Si quiere volver a la escala predeterminada, utilice el método **Scale** sin ningún argumento. Por ejemplo, para establecer el origen de coordenadas del formulario actual en (100, 100), y definir un alto y ancho interiores de 100 unidades:

```
Scale (100, 100)-(200, 200)
```

- **ScaleX (***ancho*, *escalaOrigen*, *escalaDestino***)** y **ScaleY (***alto*, *escalaOrigen*, *escalaDestino***)**. Toman el valor del *ancho* o el *alto* con su unidad de medida especificada en *escalaOrigen*, y lo

convierten al valor correspondiente en las unidades de medida especificadas por *escalaDestino*. Los posibles valores de ambas escalas son los mismos que para la propiedad **ScaleMode**.

- **PrintForm**. Envía una imagen del formulario especificado a la impresora. Se imprime todo el formulario, incluso si alguna parte del formulario no es visible en la pantalla. Sin embargo, los gráficos sólo se imprimen si la propiedad **AutoRedraw** del formulario es **True**.
- **PSet** [**Step**] **(***x***,** *y***),** [*color*]. Establece el color de un píxel individual. Los argumentos *x* e *y* pueden ser enteros o fraccionarios. La sintaxis **Step (***x***,** *y***)** significa que la ubicación del punto especificado es relativa al último punto dibujado, y se obtiene añadiendo a éste los valores *x* e *y*. Si no se incluye el argumento *color*, **PSet** establece el píxel al color **ForeColor**.
- **Point(***x, y***)**. Devuelve el color RGB del punto especificado como un número entero tipo Long. Si el punto definido por las coordenadas *x* e *y* está fuera del *objeto*, el método **Point** devuelve -1.
- **Line** [[**Step**] **(***x1***,** *y1***)**] [**Step**] **- (***x2***,** *y2***),** [*color*]**,** [**B**][**F**]. Dibuja una línea, un rectángulo o un cuadro relleno. Los argumentos (*x1, y1*) y (*x2, y2*) indican las coordenadas del punto inicial y final respectivamente. La pareja de coordenadas (*x1, y1*) es opcional, si se omite se utiliza el último punto dibujado por el método gráfico o el método **Print** anterior, o el valor de **CurrentX** y **CurrentY**. Con la opción **B**, se dibuja un rectángulo y trata los puntos especificados como las esquinas opuestas del rectángulo. Con la opción **F** detrás de **B**, el método **Line** pasa por alto **FillColor** y **FillStyle**, y rellena siempre con un color sólido. **F** no puede usarse sin **B**.
- **Circle** [**Step**] **(***x, y***),** *radio,* [*color, inicio, fin, aspecto*]. Dibuja un círculo, un arco o una elipse. Para dibujar un círculo, se necesitan los argumentos *x* e *y* (coordenadas del centro) y *radio*. Para dibujar un arco, se debe proporcionar los argumentos de ángulo en radianes para definir el *inicio* y el *fin* del arco. Si *inicio* o *fin* son negativos, se dibuja una línea que conecta el centro del círculo con el extremo negativo. Para dibujar una elipse, el argumento *aspecto* especifica la razón entre las dimensiones vertical y horizontal. Si *aspecto* es menor que uno, *radio* es el radio x; si *aspecto* es mayor o igual que uno, *radio es el radio y*.
- **Cls**. Borra los gráficos creados en tiempo de ejecución, y restablece **CurrentX** y **CurrentY** a 0.

4. Unidades, carpetas y archivos.

4.1. Modelo de objetos del sistema de archivos.

4.1.1. Trabajo con el sistema de archivos.

4.1.1.1. Descripción.

- El modelo de objetos del sistema de archivos (FSO) proporciona una solución basada en objetos para trabajar con carpetas y archivos. Permite la creación y gestión de archivos secuenciales de texto. Sin embargo, no permite la creación o gestión de archivos binarios o de acceso aleatorio.
- El objeto **FileSystemObject** es el objeto principal, proporciona acceso al sistema de archivos. Permite crear, eliminar y obtener información, y controlar unidades, carpetas y archivos.
- Para programar con el modelo de objetos FSO es necesario crear un objeto **FileSystemObject** para trabajar con él. Puede hacerse declarando una variable como del tipo **FileSystemObject**:

```
Dim fso As New FileSystemObject
```

- La propiedad **Drives** devuelve una colección formada por los objetos **Drive** disponibles en la máquina local. Puede recorrer los miembros de la colección **Drives** utilizando **For Each...Next**.

4.1.1.2. Métodos.

- **CreateFolder(***nombreDeLaCarpeta***)**. Crea una carpeta y devuelve un objeto **Folder** para tener acceso a sus propiedades. Se produce un error si ya existe la carpeta especificada.
- **CreateTextFile(***nombreDeArchivo* [**,** *sobrescribir*[**,** *unicode*]]**)**. Crea un archivo y devuelve un objeto **TextStream** que se puede utilizar para leer o escribir el archivo. Si *sobrescribir* es **True**, en caso de que el archivo ya exista se sobreescribe; por defecto es **False**. Si *unicode* es **True**, se crea como un archivo Unicode; por defecto es **False** y se crea como un archivo ASCII.
- **OpenTextFile(***nombreDeArchivo* [**,** *modoES* [**,** *crear*[**,** *formato*]]]**)**. Abre un archivo y devuelve un objeto **TextStream** que se puede utilizar para leer o escribir el archivo. El argumento *modoES* puede ser **ForReading** (sólo lectura), **ForWriting** (sólo escritura) o **ForAppending** (lectura y escritura al final del archivo). Si *crear* es **True**, en caso de que el archivo no exista se crea; por defecto es **False**. Si *formato* es **TristateTrue**, se abre como un archivo Unicode; si tiene el valor predeterminado **TristateFalse** se abre como un archivo ASCII.
- **DeleteFolder** *especCarpeta*[, *forzar*]. Elimina la carpeta especificada y su contenido. El argumento *especCarpeta* puede contener caracteres comodín en la última carpeta de la ruta. Si no se encuentran carpetas coincidentes, se produce un error. Si *forzar* es **True**, también se eliminan las carpetas de sólo lectura; por defecto es **False**.
- **DeleteFile** *especArchivo*[, *forzar*]. Elimina el archivo especificado. El argumento *especArchivo* puede contener caracteres comodín para procesar uno o más archivos. Si no se encuentran archivos coincidentes, se produce un error. Si *forzar* es **True**, también se eliminan los archivos de sólo lectura; por defecto es **False**.
- **CopyFolder** *origen*, *destino*[, *sobrescribir*]. Copia una carpeta y sus subcarpetas desde la ruta *origen* (que puede contener caracteres comodín en la última carpeta de la ruta) hasta la ruta *destino*. Si no se encuentran carpetas coincidentes, se produce un error. Si *sobrescribir* es **True**, en caso de que las carpetas ya existan se sobreescriben; por defecto es **False**.
- **CopyFile** *origen*, *destino* [, *sobrescribir*]. Copia uno o más archivos desde la ruta *origen* (que puede contener caracteres comodín para procesar uno o más archivos) hasta la ruta *destino*. Si no se encuentran archivos coincidentes, se produce un error. Si *sobrescribir* es **True**, en caso de que los archivos ya existan se sobreescriben; por defecto es **False**.
- **MoveFolder** *origen*, *destino*. Mueve una carpeta y sus subcarpetas desde la ruta *origen* (que puede contener caracteres comodín en la última carpeta de la ruta) hasta la ruta *destino*. Si no se encuentran carpetas coincidentes, se produce un error.
- **MoveFile** *origen*, *destino*. Mueve uno o más archivos desde la ruta *origen* (que puede contener caracteres comodín para procesar uno o más archivos) hasta la ruta *destino*. Si no se encuentran archivos coincidentes, se produce un error.
- **DriveExists(***Unidad***)**. Devuelve **True** si existe la unidad especificada y **False** si no existe.
- **FolderExists(***Carpeta***)**. Devuelve **True** si existe la carpeta especificada y **False** si no existe.
- **FileExists(***Archivo***)**. Devuelve **True** si existe el archivo especificado y **False** si no existe.

- **GetDrive** *Unidad*. Devuelve un objeto **Drive** que corresponde a la unidad en una ruta especificada para tener acceso a sus propiedades. El argumento *Unidad* se puede obtener mediante la expresión **GetDriveName(GetAbsolutePathName(***ruta***))**.
- **GetFolder(***Carpeta***)**. Devuelve un objeto **Folder** que corresponde a la carpeta en una ruta especificada para tener acceso a sus propiedades. Se produce un error si la carpeta no existe.
- **GetFile(***Archivo***)**. Devuelve un objeto **File** que corresponde al archivo en una ruta especificada para tener acceso a sus propiedades. Se produce un error si el archivo no existe.

```
Dim fso As New FileSystemObject
Dim fil As File, fldr As Folder, ts As TextStream
Set fil = fso.GetFile("c:\file1.txt")
Set fldr = fso.CreateFolder("C:\MiPrueba")
Set ts = fso.OpenTextFile("c:\file2.txt", ForWriting, True)
```

4.1.2. Trabajo con unidades, carpetas y archivos.

- El objeto **Drive** proporciona acceso a las propiedades de las unidades del sistema, que pueden ser discos duros, unidades CD-ROM/DVD y unidades de red. Algunas de sus propiedades son:
  - **TotalSize**. Devuelve el espacio total en bytes.
  - **FreeSpace**. Devuelve la cantidad de espacio libre disponible.
  - **DriveLetter**. Devuelve la letra de la unidad. Si la unidad especificada no está asociada con ninguna letra, devuelve una cadena de longitud cero ("").
  - **FileSystem**. Devuelve el tipo de sistema de archivos que utiliza la unidad especificada.
  - **IsReady**. Devuelve **True** si la unidad especificada está preparada y **False** si no lo está.
- El objeto **Folder** proporciona acceso a las propiedades de una carpeta. Algunas de ellas son:
  - **SubFolders**. Devuelve una colección formada por los objetos **Folder** contenidos en la carpeta especificada, incluyendo las carpetas del sistema y las carpetas ocultas.
  - **Files**. Devuelve una colección formada por los objetos **File** contenidos en la carpeta especificada, incluyendo los archivos del sistema y los archivos ocultos.
  - **Path**. Devuelve la ruta de la carpeta especificada.
  - **Name** [= *nombreNuevo*]. Establece o devuelve el nombre de una carpeta.
  - **Attributes** [= *atributosNuevos*]. Devuelve o establece los atributos de una carpeta.
  - **Drive**. Devuelve la letra de la unidad en la que reside la carpeta.
  - **Size**. Devuelve el tamaño en bytes de los archivos y subcarpetas contenidos en la carpeta.
- El objeto **File** proporciona acceso a las propiedades de un archivo. Algunas de ellas son:
  - **Path**. Devuelve la ruta del archivo especificada.
  - **Name** [= *nombreNuevo*]. Establece o devuelve el nombre de un archivo.
  - **Attributes** [= *atributosNuevos*]. Devuelve o establece los atributos de un archivo.
  - **Drive**. Devuelve la letra de la unidad en la que reside el archivo.
  - **Size**. Devuelve el tamaño en bytes del archivo especificado.

4.1.3. Trabajo con archivos secuenciales.

4.1.3.1. Descripción.

- Un archivo s*ecuencial* está estructurado en bloques continuos de texto. Los datos *no* están divididos en series de registros, sino que cada carácter del archivo representa un carácter de texto o una secuencia de formato de texto, como un carácter de nueva línea.
- El modelo de objetos FSO tiene el objeto **TextStream** que permite leer y escribir archivos secuenciales de texto, y proporciona una serie de propiedades y un conjunto de métodos.

4.1.3.2. Propiedades.

- **Line**. Devuelve el número de línea actual en un archivo secuencial. Una vez que se abre un archivo inicialmente y antes de que se escriba algo, la propiedad **Line** es igual a 1.
- **Column**. Devuelve el número de la columna de la posición actual del carácter en un archivo secuencial. Después de escribir un carácter nueva línea, pero antes de que se escriba cualquier otro carácter, **Column** es igual a 1.
- **AtEndOfLine**. Devuelve **True** si el puntero de lectura está al final de línea de un archivo, y **False** en caso contrario. Sólo es aplicable a archivos abiertos **ForReading**.
- **AtEndOfStream**. Devuelve **True** si el puntero de lectura está al final de un archivo, y **False** en caso contrario. Sólo es aplicable a archivos abiertos **ForReading**.

4.1.3.3. Métodos.

- **Read(***caracteres***)**. Lee un número especificado de caracteres y devuelve la cadena resultante.
- **Skip(***caracteres***)**. Lee y descarta un número especificado de caracteres.
- **ReadLine**. Lee una línea sin incluir el carácter nueva línea y devuelve la cadena resultante.
- **SkipLine**. Lee y descarta todos los caracteres hasta el próximo carácter nueva línea incluido.
- **ReadAll**. Lee un archivo completo y devuelve la cadena resultante.
- **Write(***cadena***)**. Escribe una cadena especificada sin caracteres intermedios entre cada cadena.
- **WriteLine(**[*cadena*]**)**. Escribe un caracter nueva línea y una cadena especificada.
- **WriteBlankLines(***líneas***)**. Escribe un número especificado de caracteres nueva línea.
- **Close**. Cierra un archivo abierto.

4.2. Archivos aleatorios y binarios.

4.2.1. Trabajo con archivos aleatorios.

- Un archivo *aleatorio* está estructurado en *registros* de longitud idéntica, cada uno de los cuales contiene uno o más campos. Puede usar tipos definidos por el usuario mediante **Type…End Type** para crear registros cuyos campos sean de tipos de datos diferentes, teniendo en cuenta que los campos **String** deben tener longitud fija. Los datos se almacenan como información binaria.
- Para el manejo de archivos aleatorios se cuenta con las siguientes instrucciones:
  - **Open** *nombreRutaAcceso* **As** *númeroArchivo* **Len** = *longitudReg*. Activa operaciones de entrada/salida (E/S) con un archivo. El argumento *númeroArchivo* es un número obtenido con la función **FreeFile**, y el argumento *longitudReg* especifica el tamaño de cada registro en bytes obtenido con la función **Len(***variableRegistro***)**.
  - **Close** [[#]*númeroarchivo*] [**,** [#]*númeroarchivo*]. Termina las operaciones de entrada/salida (E/S) en un archivo abierto con la instrucción **Open**. Si se omiten los números de archivo, se

cierran todos los archivos activos abiertos con la instrucción **Open**. Cuando se ejecuta la instrucción **Close**, la asociación de un archivo con su número de archivo termina.

- **Put** [#]*númeroarchivo***,** [*númeroregistro*]**,** *nombrevariable.* Escribe en un archivo los datos contenidos en *nombrevariable*. El argumento *númeroregistro* indica el número de registro en el cual se comienza a escribir. El primer registro de un archivo se encuentra en la posición 1, el segundo en la posición 2 y así sucesivamente. Para agregar registros al final, establezca el valor de *númeroregistro* a uno más que el número de registros del archivo.
- **Get** [#]*númeroarchivo***,** [*númeroregistro*]**,** *nombrevariable.* Lee datos de un archivo abierto y coloca la información en *nombrevariable*. Si se omite *númeroregistro*, se lee el siguiente registro que se encuentra después de la última instrucción **Get** o **Put**, o al que señala la última función **Seek**.

4.2.2. Trabajo con archivos binarios.

- Un archivo *binario* no tiene estructura, los bytes del archivo pueden representar cualquier cosa. Es similar al acceso aleatorio, excepto porque no se hacen suposiciones sobre los tipos de datos o la longitud del registro.. Cuando escriba datos binarios a un archivo, use una variable que sea una matriz de tipo de datos **Byte**, en vez de una variable **String**.
- Para el manejo de archivos aleatorios se cuenta con las siguientes instrucciones:
  - **Open** *nombreRutaAcceso* **For Binary As** *númeroArchivo*. Activa operaciones de entrada/salida (E/S) con un archivo. El argumento *númeroArchivo* es un número obtenido con la función **FreeFile**.
  - **Close** [[#]*númeroarchivo*] [**,** [#]*númeroarchivo*]. Termina las operaciones de entrada/salida (E/S) en un archivo abierto con la instrucción **Open**. Si se omiten los números de archivo, se cierran todos los archivos activos abiertos con la instrucción **Open**. Cuando se ejecuta la instrucción **Close**, la asociación de un archivo con su número de archivo termina.
  - **Put** [#]*númeroarchivo***,** [*númerobyte*]**,** *nombrevariable.* Escribe en un archivo los datos contenidos en *nombrevariable*. El argumento *númerobyte* indica el número de byte en el cual se comienza a escribir. El primer registro de un archivo se encuentra en la posición 1, el segundo en la posición 2 y así sucesivamente. Para agregar bytes al final, establezca el valor de *númerobytes* a uno más que el número de bytes del archivo.
  - **Get** [#]*númeroarchivo***,** [*númerobyte*]**,** *nombrevariable.* Lee datos de un archivo abierto y coloca la información en *nombrevariable*. Si se omite *númerobyte*, se lee el siguiente byte que se encuentra después de la última instrucción **Get** o **Put**, o al que señala la última función **Seek**.

4.2.3. Funciones utilizadas con archivos.

- **Len(***nombrevar***)**. Devuelve el número de bytes necesarios para almacenar una variable.
- **FileLen(***nombre_ruta***)**. Devuelve la longitud en bytes de un archivo cerrado.
- **LOF(***númeroarchivo***)**. Devuelve el tamaño en bytes de un archivo abierto mediante **Open**.
- **FreeFile**[**(***númerointervalo***)**]. Devuelve el siguiente número de archivo disponible para su uso en la instrucción **Open**. Se utiliza para proporcionar un número de archivo que no esté en uso.

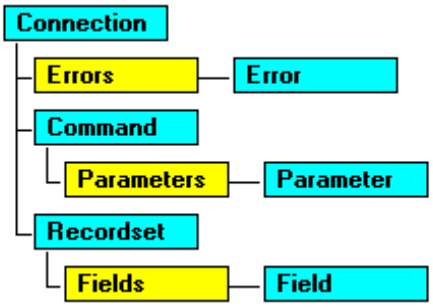
- **EOF(***númeroarchivo***)**. Devuelve un valor **True** al llegar al final del archivo, y devuelve **False** hasta que la última instrucción **Get** que se haya ejecutado no pueda leer el registro completo.
- **Seek(***númeroarchivo***)**. En archivos *aleatorios* devuelve el número del siguiente registro a leer o escribir, y en archivos *binarios* devuelve la posición de byte a leer o escribir.
- **Loc(***númeroarchivo***)**. En archivos *aleatorios* devuelve el número del último registro leído o escrito, y en archivos *binarios* devuelve la posición del último byte leído o escrito.

5. ActiveX Data Objects (ADO).

5.1. Modelo de objetos ADO.

5.1.1. Estructura de objetos ADO.

- Una aplicación basada en datos es una aplicación en la que la mayor parte del procesamiento implica consultar una base de datos y devolver resultados. El origen de datos típico es una base de datos relacional compatible con el estándar Open Database Connectivity (ODBC), que se manipula mediante comandos escritos en Structured Query Language (SQL).
- La solución general que ofrece Microsoft a este problema es OLE DB, un conjunto de interfaces Component Object Model (COM) que proporciona un acceso uniforme a los datos almacenados en orígenes de información diversos. Sin embargo, su manejo desde una aplicación es complejo, y se necesita una API como ActiveX Data Objects (ADO) entre ambos.
- El modelo de objetos ADO está basado en la siguiente estructura de objetos:



5.1.2. Objetos ADO.

- El acceso desde la aplicación a un origen de datos se realiza a través de una *conexión*. Un objeto **Connection** representa una conexión abierta con un origen de datos, y se utiliza para especificar un proveedor concreto y cualquier parámetro.
- Un *comando* emitido a través de una conexión establecida agrega, suprime o actualiza los datos del origen de datos, o recupera los datos de una tabla en forma de filas. Un objeto **Command** es la definición de un comando específico que se piensa ejecutar contra un origen de datos, y se utiliza para consultar una base de datos y obtener registros en un objeto **Recordset**. También permite actualizar los datos y manipular la estructura de una base de datos.
- Si el comando es una consulta que devuelve datos como filas de una tabla, entonces esas filas se colocan en almacenamiento local. Un objeto **Recordset** representa todo el conjunto de registros de una tabla o del resultado de un comando ejecutado. En cualquier momento, el objeto **Recordset** sólo hace referencia a un único registro dentro del conjunto, llamado registro actual. Todos los objetos **Recordset** se construyen utilizando filas y columnas.

- Hay cuatro tipos diferentes de cursores en ADO:
  - *Dinámico* (**adOpenDynamic**). Permite ver inserciones, modificaciones y eliminaciones de otros usuarios, y realizar todos los tipos de movimientos a través del **Recordset**.
  - *Conjunto de claves* (**adOpenKeyset**). Permite ver sólo modificaciones de otros usuarios, y realizar todos los tipos de movimientos a través del **Recordset**.
  - *Estático* (**adOpenStatic**). Las inserciones, modificaciones o eliminaciones de otros usuarios no son visibles, y permite realizar todos los tipos de movimientos a través del **Recordset**.
  - *Forward-Only* (**adOpenForwardOnly**). Permite ver inserciones, modificaciones y eliminaciones de otros usuarios, y sólo permite recorrer el **Recordset** hacia delante.
- Un objeto **Field** representa una columna de datos con un tipo de datos común. Un objeto **Recordset** tiene una colección **Fields** que contiene objetos **Field**, y cada uno de ellos se corresponde con una columna del **Recordset**. Cada objeto **Field** tiene la propiedad **Name** que indica el nombre de la columna, y la propiedad **Value** que indica su valor.
- El conjunto de objetos **Error** de la colección **Errors** describe todos los errores que se producen en respuesta a una única instrucción. Cuando una operación ADO genera un error, se borra la colección **Errors** y el nuevo conjunto de objetos **Error** se agrega a la colección **Errors**.
- La colección **Errors** del objeto **Connection** sólo se borra y se llena cuando el proveedor genera un nuevo error, o cuando se invoca el método **Clear**. Cada objeto **Error** representa un error específico del proveedor, no un error ADO. Cuando ocurre un error específico de ADO, se dispara un evento **On Error** y aparece en el objeto **Err**.
- Puede recorrer los miembros de la colección **Errors** utilizando **For Each...Next** para comprobar las siguientes propiedades de cada objeto **Error**:
  - **Number**. Es un valor numérico que representa el error en tiempo de ejecución más reciente.
  - **Description**. Contiene una cadena descriptiva asociada al número de error actual.
  - **Source**. Es una cadena que representa al objeto que generó el error.
- ADO proporciona estos tipos de colecciones:
  - El objeto **Connection** contiene la colección *Errors*, que contiene todos los objetos de **Error** creados en respuesta a un fallo único que implica al origen de datos.
  - El objeto **Recordset** contiene la colección *Fields*, que contiene todos los objetos **Field** que definen las columnas de ese objeto **Recordset**.

5.2. Modelo de programación ADO.

5.2.1. Conexión a base de datos y comienzo de transacción.

- Para empezar a manejar la base de datos, es necesario crear un objeto **Connection**. De manera opcional, se puede comenzar una transacción. La sintaxis de los métodos es la siguiente:

  connection.**Open** *ConnectionString*

  connection.**BeginTrans**

- Por ejemplo:

```
Dim cnn1 As New ADODB.Connection ' Abre una conexión
cnn1.CursorLocation = adUseClient
' Para que funcione la propiedad RecordCount del Recordset
```

```
cnn1.Open "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=C:\DATOS.MDB"
cnn1.BeginTrans
```

5.2.2. Ejecución de comandos SQL.

5.2.2.1. Mediante el objeto Command.

- En primer lugar es necesario crear un objeto **Command** para representar un comando SQL. Después hay que asignarle un valor a tres propiedades de este objeto:
  - **ActiveConnection**. Debe estar establecida al objeto **Connection** correspondiente.
  - **CommandText**. Debe contener la cadena del comando SQL. Si se especifica una consulta que devuelve filas, los resultados generados se almacenan en un nuevo objeto **Recordset**, que siempre es un cursor *Forward-Only* de sólo lectura.
  - **CommandType**. Debe contener el valor **adCmdText**.
- A continuación se utiliza el método **Execute** del objeto **Command**. La sintaxis es la siguiente:

Set *recordset* = *command***.Execute( )**	Si devuelve filas
*command***.Execute**	Si no devuelve filas

- Por ejemplo:

```
Dim cmdChange As ADODB.Command
' Crea un objeto Command y usa su método de ejecución
Set cmdChange = New ADODB.Command
Set cmdChange.ActiveConnection = cnn1
cmdChange.CommandType = adCmdText
cmdChange.CommandText = "UPDATE Títulos SET Tipo = " & _
"'auto_ayuda' WHERE Tipo = 'psicología'"
cmdChange.Execute
```

5.2.2.2. Mediante el objeto Connection.

- Se utiliza el método **Execute** del objeto **Connection**, pasando en el argumento *CommandText* la cadena del comando SQL. Si se especifica una consulta que devuelve filas, los resultados generados se almacenan en un nuevo objeto **Recordset**, que siempre es un cursor *Forward-Only* de sólo lectura. La sintaxis es la siguiente:

Set *recordset* = *connection***.Execute (***CommandText***)**	Si devuelve filas
*connection***.Execute** *CommandText*	Si no devuelve filas

- Por ejemplo:

```
Dim rstAutores As ADODB.Recordset
' Usa el método de ejecución del objeto Connection
Set rstAutores = cnn1.Execute ("SELECT * FROM authors " & _
    "WHERE year = 1971")
```

5.2.2.3. Mediante el objeto Recordset.

- Se utiliza el método **Open** del objeto **Recordset**, pasando los siguientes argumentos:
  - *Source*. La cadena del comando SQL.
  - *ActiveConnection*. El objeto **Connection** correspondiente.

  - *CursorType*. Uno de los cuatro valores posibles: **adOpenDynamic**, **adOpenKeyset**, **adOpenStatic** o **adOpenForwardOnly** (predeterminado).
  - *LockType*. Uno de los cuatro valores posibles: **adLockPessimistic**, **adLockOptimistic**, **adLockBatchOptimistic** o **adLockReadOnly** (predeterminado).
  - *Options*. El valor **adCmdText**.
- La sintaxis es la siguiente:

  recordset.**Open** *Source*, *ActiveConnection*, *CursorType*, *LockType*, *Options*
- Por ejemplo:

```
Dim rstEmployees As ADODB.Recordset
' Crea un objeto Recordset y usa su método Open
Set rstEmployees = New ADODB.Recordset
rstEmployees.Open "SELECT * FROM employees " & _
   "WHERE year = 1971", cnn1, adOpenDynamic, , adCmdText
```

5.2.3. Recorrido, filtrado y ordenación del objeto Recordset.

5.2.3.1. Propiedades.

- **BOF** y **EOF**. La propiedad **BOF** devuelve **True** si la posición del puntero de registro actual está antes del primer registro y **False** en el resto de casos. La propiedad **EOF** devuelve **True** si la posición del puntero de registro actual está después del último registro y **False** en el resto de casos. Si se abre un objeto **Recordset** que no contiene registros, las propiedades **BOF** y **EOF** se establecen a **True**. Cuando **BOF** o **EOF** es **True**, el registro actual no contiene ningún registro.
- **RecordCount**. Indica el número actual de registros de un objeto **Recordset**. Si se abre un objeto **Recordset** que no contiene registros, el valor de la propiedad **RecordCount** es cero.
- **Filter** = *cadenaCriterios*. Descarta de manera selectiva registros de un objeto **Recordset**, afectando a la propiedad **RecordCount**. Al establecer la propiedad **Filter**, el registro actual se moverá al primer registro que cumpla los criterios. De manera similar, cuando se borra la propiedad **Filter**, la posición del registro actual pasa al primer registro del **Recordset**. El argumento *cadenaCriterios* se compone de cláusulas con el formato *NombreCampo -Operador- Valor* que pueden estar concatenadas con **AND** u **OR**:
  - *NombreCampo* debe ser un nombre de campo válido del **Recordset**, si contiene espacios deberá ponerlo entre corchetes.
  - *Operador* debe ser uno de los siguientes: <, >, <=, >=, <>, = o **LIKE**. Si *Operador* es **LIKE**, los comodines que se admiten son el asterisco (*) y el signo de porcentaje (%).
  - *Valor* es el valor con el que comparará los valores de los campos (por ejemplo, `'Smith'`, `#8/24/95#`, `12.345` o `$50.00`). Utilice comillas sencillas con las cadenas y el símbolo de almohadilla (#) con las fechas. *Valor* no puede ser **Null**.
- **Sort** = *cadenaCriterios*. Establece un valor de tipo **String** con los nombres de los campos utilizados para ordenar separados por comas. Cada nombre es un objeto **Field** del objeto **Recordset** y opcionalmente está seguido de un espacio en blanco y de la palabra clave **ASCENDING** o **DESCENDING**, que especifica el orden del campo. Al establecer la propiedad **Sort** a una cadena vacía, se restablecen las filas a su orden original.

5.2.3.2. Métodos.

- **MoveFirst**. Mueve el puntero de registro actual al primer registro del **Recordset**.
- **MoveLast**. Mueve el puntero de registro actual al último registro del **Recordset**.
- **MoveNext**. Mueve el puntero de registro actual un registro hacia delante hacia el final del **Recordset**. Si el registro actual es el último registro y se llama al método **MoveNext**, ADO establece el registro actual a la posición posterior al último registro del **Recordset** (**EOF** es **True**). El intento de avanzar cuando la propiedad **EOF** ya sea **True** genera un error.
- **MovePrevious**. Mueve el puntero de registro actual un registro hacia atrás hacia el principio del **Recordset**. Si el registro actual es el primer registro y se llama al método **MovePrevious**, ADO establece el registro actual a la posición anterior al primer registro del recordset (**BOF** es **True**). El intento de retroceder cuando la propiedad **BOF** ya sea **True** genera un error.
- **Move** *NumRecords*, *Start*. Mueve el puntero de registro actual un número *NumRecords* de registros a partir del registro actual por defecto, o desde el primer (*Start* = **adBookmarkFirst**) o el último (*Start* = **adBookmarkLast**) registro. Si *NumRecords* es mayor o menor que cero, el puntero de registro actual avanza hacia el final o el principio respectivamente del **Recordset**.
- **Close**. Libera los datos asociados y los accesos exclusivos utilizados por el objeto **Recordset**.
- **Requery**. Actualiza todo el contenido de un objeto **Recordset** desde el origen de datos volviendo a emitir el comando original y recuperando los datos una segunda vez. Llamar a este método es equivalente a llamar a los métodos **Close** y **Open** sucesivamente sin cambiar ninguna propiedad del objeto **Recordset**. Si se está modificando el registro actual o agregando un nuevo registro, se produce un error.

5.2.4. Actualización de datos en el objeto Recordset y en la base de datos.

- **AddNew**. Crea e inicializa un nuevo registro en un objeto **Recordset**. A continuación hay que asignar los valores a los campos del **Recordset**. Después de invocar el método **AddNew**, el nuevo registro se convierte en el registro actual.
- **Delete**. Marca el registro actual para su eliminación. El registro eliminado sigue siendo el actual hasta que se pasa a otro registro diferente. Después de dejar el registro eliminado, ya no es accesible. Si el intento de eliminar registros falla, el proveedor devuelve advertencias en la colección **Errors** pero no detiene la ejecución del programa.
- **Update**. Guarda los cambios realizados en el registro actual de un objeto **Recordset** desde que se llamó al método **AddNew** o desde que se cambió cualquier valor de campo en un registro existente, manteniendo el puntero de registro actual. Si se sale del registro que está modificando o agregando, ADO llamará automáticamente a **Update** para guardar los cambios.
- **CancelUpdate**. Cancela las modificaciones efectuadas en el registro actual o ignora un registro recién agregado. No se puede deshacer las modificaciones del registro actual o de un nuevo registro después de invocar el método **Update**, a menos que formen parte de una transacción. El puntero de registro actual vuelve al registro actual anterior a la llamada a **AddNew**. Por ejemplo:

```
rstEmployees.AddNew
rstEmployees.Fields("emp_id") = strID
rstEmployees.Fields("fname") = strFirstName
```

```
rstEmployees.Fields("lname") = strLastName
If MsgBox("Guardar cambios?", vbYesNo) = vbYes Then
   rstEmployees.Update
Else
   rstEmployees.CancelUpdate
End If
```

5.2.5. Final de transacción y cierre de conexión a base de datos.

- Si se utilizó una transacción, se pueden aceptar o rechazar los cambios realizados durante la transacción. Por último hay que cerrar la conexión a la base de datos. La sintaxis es la siguiente:

connection.**CommitTrans**	Si acepta los cambios
connection.**RollbackTrans**	Si rechaza los cambios
connection.**Close**	

- Por ejemplo:

```
' Pregunta al usuario si desea confirmar los cambios
If MsgBox("¿Desea guardar cambios?", vbYesNo) = vbYes Then
   cnn1.CommitTrans
Else
   cnn1.RollbackTrans
End If
cnn1.Close
```

5.3. Controles ADO.

5.3.1. Controles de enlace de datos.

5.3.1.1. Descripción.

- Un control de enlace de datos es un control que se conecta a un proveedor de datos OLE DB a través de un control de datos ADO. Mediante esta conexión puede tener acceso a uno o varios campos específicos de una base de datos. Los controles de enlace de datos que administran un único campo suelen mostrar el valor del campo específico del registro actual.
- Cuando un control de datos ADO se mueve de un registro al siguiente, todos los controles de enlace de datos conectados a dicho control pasan a presentar datos de los campos del registro actual. Cuando los usuarios modifican los datos en un control de enlace de datos y los mueven a un registro diferente, las modificaciones se guardan automáticamente en la base de datos.
- Algunos de los controles intrínsecos pueden ser de enlace de datos, como los controles **CheckBox**, **ComboBox**, **Image**, **Label**, **ListBox**, **PictureBox** y **TextBox**. Visual Basic incluye distintos controles ActiveX de enlace de datos, como los controles **DataCombo** y **DataList**.

5.3.1.2. Propiedades.

- **DataSource** [=*datasource*]. Devuelve o establece el origen de datos para enlazar el control con una base de datos. El parámetro *datasource* es una referencia a un control de datos ADO. Utilice la instrucción **Set** para establecer la propiedad **DataSource**.
- **DataField** [= *cadena*]. Devuelve o establece el nombre de uno de los campos del origen de datos especificado en la propiedad **DataSource**.

5.3.2. Adodc (control de datos ADO).

5.3.2.1. Descripción.

- El control de datos **ADO** permite crear de una manera rápida conexiones entre controles de enlace de datos y proveedores de datos. Tiene la ventaja de que es un control gráfico (con botones **Adelante** y **Atrás**) y una interfaz fácil de usar que le permite crear aplicaciones para bases de datos con un mínimo de código.
- Permite definir un conjunto de registros basados en una consulta SQL y pasar valores de un campo de datos a controles de enlace de datos, en los que puedan mostrar o cambiar los valores. También permite agregar nuevos registros o actualizar una base de datos según los cambios hechos en los datos que aparecen en los controles de enlace de datos.

5.3.2.2. Propiedades.

- **ConnectionString**. Especifica un origen de datos pasando una cadena de conexión detallada que contiene una serie de instrucciones *argumento* = *valor* separadas con punto y coma. Por ejemplo `"Provider=Microsoft.Jet.OLEDB.4.0; Data Source=C:\DATOS.MDB"`
- **RecordSource**. Debe contener la cadena de la consulta SQL a ejecutar.
- **CommandType**. Debe contener el valor **adCmdText**.
- **CursorType**. Uno de los siguientes valores **adOpenDynamic**, **adOpenKeyset** o **adOpenStatic**.

5.3.3. MSFlexGrid.

5.3.3.1. Descripción.

- El control Microsoft FlexGrid (**MSFlexGrid**) muestra datos de tablas y efectúa operaciones en ellos. Proporciona una flexibilidad completa para ordenar, combinar y aplicar formato a tablas que contienen cadenas e imágenes.
- Es posible colocar texto, una imagen o ambas cosas en cualquier celda de un control **MSFlexGrid**. Puede especificar la celda actual mediante código o bien el usuario puede cambiar la misma en tiempo de ejecución mediante el *mouse* o las flechas del cursor.

5.3.3.2. Propiedades.

- **Row** [= *número*] y **Col** [= *número*]. Devuelven o establecen las coordenadas de la celda actual. Las filas y las columnas se numeran desde cero, comenzando por arriba y por la izquierda respectivamente. Al establecer estas propiedades se restablecen automáticamente **RowSel** y **ColSel**, de manera que la selección se convierte en la celda actual. Por tanto, para especificar una selección de bloque, debe establecer primero **Row** y **Col**, y después **RowSel** y **ColSel**.
- **RowSel** [=*valor*] y **ColSel** [= *valor*]. Devuelven o establecen una selección de celdas como la región comprendida entre las filas **Row** y **RowSel** y las columnas **Col** y **ColSel**. **RowSel** puede estar encima o debajo de **Row**, y **ColSel** puede estar a la izquierda o a la derecha de **Col**.
- **FixedRows** [= *valor*] y **FixedCols** [= *valor*]. Devuelven o establecen el número total de filas en la parte superior, y de columnas fijas en la parte izquierda respectivamente.
- **MouseRow** [=*valor*] y **MouseCol** [=*valor*]. Devuelven la fila y la columna actual del *mouse*. Permiten comprobar si el usuario ha hecho clic en filas o en columnas fijas.
- **Rows** [= *valor*] y **Cols** [= *valor*]. Devuelven o establecen el número total de filas y de columnas respectivamente. El número de filas y de columnas mínimo es cero. El valor de **Rows** y **Cols**

debe ser como mínimo uno más que el valor de las propiedades **FixedRows** y **FixedCols** respectivamente, excepto cuando ambas propiedades están establecidas en cero.

- **TextMatrix(***rowindex, colindex***)** [=*cadena*]. Devuelve o establece el contenido de texto de una determinada celda sin cambiar las propiedades **Row** y **Col**.
- **Text** [=*cadena*]. Devuelve o establece el contenido de la celda actual definida por **Row** y **Col**.
- **Sort** [=*valor*]. Establece un valor que ordena de izquierda a derecha las filas completas seleccionadas de acuerdo con un criterio seleccionado. Para especificar el intervalo que se va a ordenar, establezca las propiedades **Row**, **RowSel**, **Col** y **ColSel**. Los valores admitidos para *valor* son **flexSortNone**, **flexSortGenericAscending** y **flexSortGenericDescending**.

5.3.3.3. Eventos.

- **Click( )**. Ocurre cuando el usuario presiona y suelta un botón del *mouse* en el control.
- **SelChange()**. Ocurre cuando el intervalo seleccionado cambia a una celda o a un intervalo de celdas diferente. Este evento puede ocurrir por programa si cambia la región seleccionada mediante las propiedades **Row**, **RowSel**, **Col** y **ColSel**.
- **LeaveCell()**. Ocurre inmediatamente antes de que la celda activa actual cambie a una celda diferente. Este evento sirve para validar los contenidos de una celda, pero no se desencadena cuando el enfoque se mueve a un control diferente.
- **EnterCell()**. Ocurre cuando la celda activa actual cambia a una celda diferente. Al hacer clic en una fila fija, se produce este evento en la primera columna no fija de dicha fila. Este evento no ocurre si se arrastra el *mouse* sobre una celda.
- **RowColChange()**. Ocurre cuando la celda activa actual cambia a una celda diferente porque el usuario hace clic en una nueva celda, y después de **LeaveCell()** y **EnterCell()**.

5.3.3.4. Métodos.

- **AddItem(***string, index***)**. Agrega una fila en la posición especificada en *index*. Para la primera fila, *index* es igual a cero. Si se omite *index*, la nueva fila será la última fila del control. Utilice el carácter de tabulación (**vbTab**) en *string* para separa el contenido de las diferentes columnas.
- **RemoveItem(***index***)**. Elimina la fila especificada. Para la primera fila, *index* es igual a cero.
- **Clear**. Elimina todo el contenido de las celdas del control.