Visual Basic .NET

Referencia de Visual Basic .NET:

Visual Basic

Visual Basic .NET para usuarios veteranos de Visual Basic

*Para obtener más información de objetos, métodos y propiedades, utilice el Examinador de Objetos de Visual Basic, puede acceder a este a través de la tecla [**F2**] y busque lo que necesite.*

Índice Temático

Básico » Intermedio » Avanzado

- Operadores
  - Operadores Aritméticos
  - Operadores String
  - Operadores Comparativos
  - Operadores Lógicos
  - Operadores Abreviados
  - Precedencia de Operadores
- Sentencias de Control
  - Do ... Loop
  - Exit
- Control Timer
- Sentencia Rnd()
- Método Randomize()
- Depuración, Tipos de errores
  - Ventana de Comandos
  - Ventana de Comandos - Inmediata
- Control ListBox
- Función IsNumeric
- Función Trim, LTrim, RTrim
- Control CheckBox
- Control RadioButton
- Evento KeyPress
- Control Windows Media Player

Básico » Intermedio » Avanzado

- Try ... Catch
  - Err
- Módulo
- Matrices/Arreglos/Vectores
- Clase Array
- Colecciones
- Manipulación de Archivos de Texto
- Manipulación de Cadenas
- Funciones Chr Asc
- Automatización
- Componente Process
- Instalación Distribuida
- Formularios
- StreamReader
- StreamWriter
- Creación de controles en tiempo de ejecución
- Establecer el objeto de inicio
- Gráficos
  - Sistema de Coordenadas
  - Clase System.Drawing.Graphics
  - Evento Paint
  - Animación : Top - Left - Location - SetBounds
  - Opacidad en Formularios
- Programación Orientada a Objetos
  - Clases y Objetos

    - Encapsulación
    - Herencia
    - Polimorfismo
- Agregar una Clase
    - Campos
    - Propiedades
    - Métodos
    - Constructores

Básico » Intermedio » Avanzado

- Impresión
    - Impresión de un gráfico
    - Impresión de un texto
    - Impresión de un archivo
- Bases de Datos
    - ADO.NET
    - Conexión
        - Creación de una Conexión
    - Adaptador de Datos
        - Creación de un Adaptador de Datos
    - DataSet
        - Generación de un Dataset
    - Data-aware
    - Controles Enlazados
    - Método Fill
    - Navegación de un Dataset
    - Manipulación de la Base de Datos
    - Bases de Datos y Datagrid
    - Modificación de la base de datos con DataGrid
- Web
    - Web Forms
    - Arquitectura Tres Capas (Three-Tier)
        - Capa de Presentación - Presentation Layer
        - Capa de Aplicación - Application Layer
        - Capa de Datos - Data Layer
    - State Management
    - Técnicas de Administración de Estados del Lado del Cliente
        - View State
        - Query String
        - Cookies
    - Técnicas de Administración de Estados del Lado del Servidor
        - Application State
        - Sesion State
        - Database Support
- Sistemas Inteligentes

¿Qué es Visual Basic .NET

Es un lenguaje orientado a objetos y eventos que soporta encapsulación, herencia y polimorfismo.

Es una mejora a Visual Basic formando parte de Visual Studio y compartiendo el entorno de desarrollo con Microsoft Visual

C++ .NET, Microsoft Visual C# .NET, etc.

Entorno de Desarrollo

El Entorno de Desarrollo recibe el nombre de *Entorno de Desarrollo de Microsoft Visual Studio .NET*. Este entorno es personalizable y contiene todas las herramientas necesarias para construir programas para Microsoft Windows.

El Entorno de Desarrollo contiene múltiples ventanas y múltiples funcionalidades y es por consecuencia llamado un entorno de desarrollo integrado (***integrated development environment*** IDE).

La ventana central es la ventana de diseño (**Designer Window**), la cual contiene el formulario a desarrollar.

La caja de herramientas (**ToolBox**) se localiza de lado izquierdo. En el extremo derecho tenemos la ventana de explorador de soluciones (**Solution Explorer**).

La ventana de propiedades (**Properties window**) contiene tres partes:

1. La parte superior contiene un combo box que muestra el nombre y la clase del objeto seleccionado.
2. La parte media contiene la lista de propiedades del objeto seleccionado, de lado derecho contiene un conjunto de cajas para ver y editar el valor de la propiedad seleccionada.
3. La parte inferior es un cuadro descriptivo que proporciona una breve descripción de la propiedad seleccionada.

Es necesario tener instalado el Visual Studio .NET, al ejecutarlo se presenta una página de inicio, en caso de no presentarse entonces de clic en Help/Show Start Page. En esta página será posible establecer su perfil, por ejemplo identificarse como Desarrollador Visual Studio o más especifico como Desarrollador Visual Basic con lo cual Visual Studio configura de inmediato el entorno de desarrollo para programar en Visual Basic.

Para iniciar un nuevo proyecto, de clic en la opción **Projects** y clic en el botón [**New Project**], esta acción abre una ventana donde se indicará el archivo a abrir, los proyectos Visual Basic .NET tiene la extensión **.vbproj**. Una vez que abre el projecto si la página de inicio estaba visible continuará así y en el Explorador de Soluciones (*Solution Explorer*) se cargan los archivos correspondientes al proyecto.

En Visual Basic .NET existen dos archivos:

1. Un archivo de projecto `.vbproj`, el cual contiene información específica para una determinada tarea de programación.
2. Un archivo de solución `.sln`, el cual contiene información relacionada con uno o más proyectos. Este tipo de archivo puede administrar varios projectos relacionados entre sí y son similares a los archivos de grupos de proyecto (`.vbg`) en Visual Basic 6

Si la solución tiene un único proyecto, abrir el archivo de proyecto `.vbproj` o el archivo de solución `.sln` tiene el mismo resultado, pero si la solución es multiproyecto entonces deberá abrir el archivo de solución.

Best Practices: *Procure siempre abrir el archivo de solución `.sln`.*

Los formularios en Visual Basic .NET tienen la extensión `.vb`. Se mostraran a manera de pestañas la página de inicio, la vista de diseño y el código del formulario.

Para evitar el acoplamiento de ventanas, mientras arrastre la ventana pulse la tecla [**Ctrl**], si desea integrar la ventana como pestaña entonces arrastre la ventana sobre otras pestañas y libere.

El control **Image** desaparece en Visual Studio.

Ya no tendrá que utilizar el tabulador para identar su código.

.NET Framework

Visual Studio .NET tiene una nueva herramienta que comparte con Visual Basic, Visual C++, Visual C#, etc. llamada `.NET Framework` que además es una interfaz subyacente que forma parte del propio sistema operativo Windows.

La estrucutra de `.NET Framework` es por Clases mismas que puede incorporar a sus proyectos a través de la instrucción `Imports`, por ejemplo una de sus Clases es `System.Math` la cual soporta los siguientes métodos

Método	Descripción
`Abs(n)`	Calcula el valor absoluto de *n*
`Atan(n)`	Calcula el arcotangente de *n* en radianes
`Cos(n)`	Calcula el coseno del ángulo *n* expresado en radianes
`Exp(n)`	Calcula el constante de *e* elevada a *n*
`Sign(n)`	Regresa `-1` si *n* es menor que cero, 0 si *n* es cero y `+1` si *n* es mayor a cero
`Sin(n)`	Calcula el seno del ángulo *n* expresado en radianes
`Sqr(n)`	Calcula la raíz cuadrada de *n*.
`Tan(n)`	Calcula la tangente del ángulo *n* expresado en radianes

La declaración de `Imports` debe ser a nivel de formulario:

```
Imports System.Math
Public Class Form1
.
.
.
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)-
 Handles MyBase.Load
txtOutput.Text += Sqrt(9) & vbNewLine
.
.
.
```

.NET

La biblioteca de clases `.NET` es una biblioteca de clases incluida en el *Microsoft .NET Framework* y está diseñada para ser la base sobre las cuales las aplicaciones .NET son construidas.

namespaces

La biblioteca **.NET** contiene un número considerable de clases con código reutilizable, para ayudar a controlar esta complejidad **.NET** utiliza `namespaces`, colecciones de clases relacionadas.

Nuevo Proyecto

De clic en el botón [**New Project**] o File/New/Project, como tipo de proyecto seleccione **Visual Basic Project**, como plantilla seleccione **Windows Application**, por último indique la ubicación donde desea almacenar su proyecto. Al dar clic Visual Studio configura el entorno de desarrollo y crea un directorio con el mismo nombre que especifico para la aplicación.

Aplicación Windows Ejecutable

Las aplicaciones Windows creadas con Visual Basic .NET tienen la extensión `.exe` mismas que podrán ser ejecutadas en cualquier equipo que tenga instalado Microsoft Windows. Visual Basic .NET instala de de manera automática los archivos de soporte, incluyendo las bibliotecas de vínculos dinámicos y archivos de *.NET framework*.

Visual Studio puede crear dos tipos de archivos ejecutables:

1. `Debug` - Compilación de depuración, se utiliza cuando se prueba y depura un programa (**Default**).
2. `Release` - Versión de edición, se utiliza cuando se termina el programa siendo esta una versión optimizada de menor tamaño

Para crear un archivo ejecutable, de clic en Build/Configuration Manager para decidir sobre el tipo de archivo ejecutable, además de otras opciones como la plataforma para la cual desea crear la aplicación.

De clic en Build/Build Solution.

Lo que Visual Basic hace es crear una directorio binaria dentro del directorio que contiene el proyecto y compilará el código fuente, dando como resultado un archivo .exe con el nombre de su aplicación.

Programación Orientada a Eventos

Visual Basic .NET soporta la **Programación Orientada a Eventos** en la cual las aplicaciones reconocen y responden a eventos.

Evento

Un **Evento** es una acción o acontecimiento reconocido por algunos objetos para los cuales es necesario escribir el código para responder a dicho evento. Los eventos pueden ocurrir como resultado de una acción del usuario (onClick), por invocación a través de código o disparados por el sistema (Timer Tick Event).

Manejador de Eventos

Un **Manejador de Eventos** contiene código que responde a eventos particulares. Un desarrollador diseña cuidadosamente sus aplicaciones determinando los controles disponibles para el usuario y los eventos apropiados asociados a estos controles, entonces, el desarrollador escribe el código para integrar los eventos consistentes con el diseño de la aplicación.

Procedimientos

Un **procedimiento** es un conjunto de sentencias que realizan una acción lógica. Existen tres tipos de procedimientos en Visual Basic .NET:

1. **Event procedures/Event handler**, procedimiento que contiene código que es ejecutado en respuesta a un evento. Cuando el evento es disparado el código dentro del manejador de eventos es ejecutado.

Visual Basic .NET para los manejadores de eventos utiliza una convención estándar la cual combina el nombre del objeto seguido de un guión bajo y el nombre del evento.

```
Private|Public Sub objeto_Evento(parámetros) handles Objeto.Evento
   sentencias
End Sub
```

Cada manejador de eventos provee dos parámetros, el primer parámetro llamado `sender` provee una referencia al objeto que dispara el evento, el segundo parámetro es un objeto cuyo tipo de dato depende del evento que es manejado. Ambos parámetros son pasados por valor.

Si un parámetro es declarado por referencia `ByRef` el parámetro apunta al argumento actual. Por default los argumentos se pasan por valor `ByVal` el parámetro es una copia local del argumento.

2. **Sub procedures**, contiene código que el desarrollador crea para realizar una acción lógica.
3. **Function procedures**, contiene código que el desarrollador crea para realizar una acción lógica y regresa un valor, el valor que una función envia de regreso al programa que lo invoco es llamado *valor de regreso*. Para regresar un valor se utiliza la sentencia `Return`.

Ámbito de las variables

Cuando es declarada una variable también se define su ámbito, el ámbito de una variable es la región de código en la cual la variable se referencia directamente. Existen dos tipos de ámbitos de las variables:

1. **Local**, es una variable declarada dentro de un procedimiento y se destruye cuando el procedimiento termina de ejecutarse.
2. **Módular**, es una variable declarada a nivel módulo fuera de cualquier procedimiento y son declaradas en la parte superior del Editor de Código arriba del primer procedimiento, este espacio es llamado Sección de Declaraciones Generales (*General Declaration Section*).

Convenciones para el nombre de los controles

Es recomendable utilizar convenciones para el nombre de los controles, es decir, que al momento de dar lectura al código sea fácil de entender y comprender, por lo que el establecimiento de convenciones ayuda a identificar que control se empleo, por ejemplo, si emplea un control botón emplee siempre como prefijo **btn** depués completelo con un nombre descriptivo acorde a la funcionalidad que este tendrá.

Control	Prefijo
`Button`	**Btn**
`Label`	**Lbl**
`PictureBox`	**Pic**
`Timer`	**Tmr**
`Text Box`	**Txt**
`List Box`	**Lst**
`Combo Box`	**Cbo**
`Check Box`	**Chk**
`Radio Button`	**Rad**

Control Button (*antes CommandButton*)

El control **CommandButton** en Visual Basic .NET recibe el nombre de `Button`, la propiedad **Caption** ahora recibe el nombre de `Text`, ocurre lo mismo para el caso del control `Label`.

Control Label

El control `Label` ahora su propiedad **Caption** recibe el nombre de `Text` y por ejemplo la antes propiedad **Alignment** ahora recibe el nombre de `TextAlign` la cual tiene más opciones de alineación.

Control TextBox

El control `TextBox` tiene una capacidad de almacenamiento de 32 Kbytes de texto.

Control DateTimePicker

El control `DateTimePicker` muestra por defecto la fecha actual que es posible modificar a través de su propiedad value.

Este control puede mostrar fechas u horas, si desea mostrar horas utilice:

```
nombreControl.Format = DateTimePickerFormat.Time
```

Control LinkLabel

A través de este control es posible abrir el navegador por default y acceder a un URL especifico:

```
LinkLabel1.LinkVisited = True
System.Diagnostics.Process.Start(LinkLabel1.Text)
```

Donde `LinkLabel1.Text` es el valor que tiene asignado la propiedad `Text` por ejemplo file:///C:/pagina.html si quiere probar sin tener acceso a internet o si quiere probar con acceso a internet seria http://www.elSitioWeb.com.

Si requiere especificar el navegador con el cual desea abrir el URL basta con señalarlo:

```
System.Diagnostics.Process.Start("firefox.exe", LinkLabel2.Text)
        'La página se mostrara utilizando el navegador firefox
```

Es necesario señalar algo importante del código y adentrarnos a la programación .NET, ya que el método `Start` de la clase `Process` inicia en memoria un *proceso* de programa ejecutable para el navegador.

La clase `Process` hace mucho más que eso pero la parte a destacar es que forma parte de la biblioteca de objetos `System.Diagnostics` la cual los programadores Visual Basic .NET llaman **espacio de nombres** `System.Diagnostics`.

En cuanto al método `Process.Start` destaca que además de abrir una página web puede ser utilizado para ejecutar otras aplicaciones:

```
System.Diagnostics.Process.Start(LinkLabel3.Text)
'Donde el valor de LinkLabel3.Text es "winword"
```

El método `Start` utiliza 2 argumentos, el primero especifica la aplicación a emplear y el segundo especifica el archivo que abrira la aplicación, es decir, el siguiente ejemplo abre el archivo indicado con Microsoft Excel:

```
System.Diagnostics.Process.Start("excel", "c:\pagos.xls")
```

Controles Anteriores y Otros Controles

Es posible utilizar antiguos controles ActiveX y utilizarlos con algunas limitaciones, teniendo como restricción técnica que deberán estar en un contenedor, por ejemplo de clic con el botón derecho en el separador **Windows Forms** seleccione la opción **Add/Remove items** y aparecerá una ventana donde dará clic en el folder **COM Components** y eligira el componente que desea agregar.

Control MainMenu

El Control **MainMenu** agrega menús a un programa, el manú se configura mediante sus propiedades.

Para agregar un menú simplemente seleccione el control **MainMenu** el cual de manera automática se ajusta al tamaño del formulario, este menú es de fácil manipulación pues basta con escribir el nombre para las opciones y a través del menú popup es posible modificarlo.

Una vez agregado el control **MainMenu** también se agrega un panel debajo del formulario la cual recibe el nombre de `Bandeja de Componentes` a través del cual será posible definir y modificar sus propiedades.

Ahora bien lo que se muestra en la parte superior del formulario es un representación visual del menú misma que recibre el nombre de diseñador de menú, pero el objeto menú principal se muestra en la parte inferior dentro de la bandeja de componentes.

Para definir teclas de acceso para las opciones del menú, continua con la forma de hacerlo en Visual Basic 6 que es anteponiendo al caracter al cual deseamos hacer referencia el símbolo `&`

Si después de escribir las opciones de su menú desea cambiar el orden basta con seleccionar y mover la opción a la posición deseada.

Si desea eliminar una opción del menú basta con seleccionarla y pulsar la tecla [**Supr**] o [**Del**].

Para agregar código a cada opción del menú basta con dar doble clic sobre la opción deseada:

```
Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MenuItem2.Click
```

```
    MsgBox("opción 1_1")
End Sub
Private Sub MenuItem3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MenuItem3.Click
    MsgBox("opción 1_2")
End Sub
Private Sub MenuItem5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MenuItem5.Click
    MsgBox("opción 2_1")
End Sub
```

Funciones Intrinsecas

- `Val`, regresa el número contenido en un argumento, se detiene en el primer caracter no numérico.
- `CDbl`, regresa un valor `double` si el argumento puede ser convertido a `double`
- `IsNumeric`, regresa `True` si el argumento puede ser convertido a `double`
- `Rnd`, regresa un valor `random` entre 0 y 1
- `Abs`, regresa el valor absoluto de un número (es necesario importar `System.Math`)
- `Int`, regresa la parte entera de un número
- `FormatCurrency`, regresa un string formateado a moneda y redondeado a dos decimales
- `Formar(expr,str)`, convierte una expresión a un formato string específico.

Funciones Financieras

Visual Basic .NET provee funciones financieras como `Pmt(Rate,Nper,PV)` para determinar pagos mensuales y `FV(Rate,Nper,Pmt)` para determinar el valor futuro de una anualidad basada en pagos fijos periodicos e intereses de tasa fija.

Funciones de Tiempo y Hora

En Visual Basic .NET existen funciones a través de las cuales es posible manipular el tiempo y la hora:

- `TimeString`, regresa la hora actual del sistema.
- `DateString`, regresa la fecha actual del sistema.
- `Now`, regresa un valor codificado que representa la hora y fecha actual del sistema.
- `Hour(hora)`, regresa el número de hora actual del sistema.
- `Minute(hora)`, regresa el número de minuto actual del sistema.
- `Second(hora)`, regresa el múmero del segundo actual del sistema.
- `Day(fecha)`, regresa el número del día actual del sistema.
- `Month(fecha)`, regresa el número de mes actual del sistema.
- `Year(fecha)`, regresa el año actual del sistema.
- `Weekday(fecha)`, regresa el número que representa el día de la semana. (1 = Domingo, 2 = Lunes, ...).

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
     Handles Button1.Click
    txtTimeString.Text = TimeString
End Sub

Private Sub DateString_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
     Handles Button2.Click
    txtDateString.Text = DateString
End Sub
```

```
Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button3.Click
    txtNow.Text = Now
End Sub

Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button4.Click
    txtHour.Text = Hour(Now)
End Sub

Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button5.Click
    txtMinute.Text = Minute(Now)
End Sub

Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button6.Click
    txtSecond.Text = Second(Now)
End Sub

Private Sub Button7_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button7.Click
    txtDay.Text = Microsoft.VisualBasic.DateAndTime.Day(Now)
End Sub

Private Sub Button8_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button8.Click
    txtMonth.Text = Month(Now)
End Sub

Private Sub Button9_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button9.Click
    txtYear.Text = Year(Now)
End Sub

Private Sub Button10_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)_
      Handles Button10.Click
    txtWeekday.Text = Weekday(Now)
End Sub
```

Cuadros de Dialogo

Visual Basic .NET cuanta con siete tipos de cuadro de dialogo que facilitan la construcción de interfaz de usuario:

1. `OpenFileDialog`, obtiene unidad, nombre de directorio y nombre de un archivo existente.
2. `SaveFileDialog`, obtiene unidad, nombre de directorio y nombre para un archivo nuevo.
3. `FontDialog`, para que el usuario seleccione una configuración para la fuente.
4. `ColorDialog`, para que el usuario seleccione un color de una paleta.
5. `PrintDialog`, para que el usuario defina opciones de impresión
6. `PrintPreviewDialog`, muestra al usuario una vista previa de impresión.
7. `PageSetupDialog`, para que el usuario controle las opciones de configuración de página.

Al momento de diseñar la interfaz de usuario los controles de cuadro de dialogo no aparecen en la pantalla pero si se muestran visibles al pie de la misma, pudiendo acceder a sus propiedades para configurar el control.

Si requiere hacer visible el cuadro de dialogo en modo ejecución tendrá que emplear el método `ShowDialog()`.

El método `ShowDialog()` regresa un valor denominado `DialogResult`, indicando el botón del cuadro de diálogo en el que

el usuario dio clic.

OpenFileDialog

El control **OpenFileDialog** representa un dialogo preconfigurado para seleccionar un archivo que será abierto. La propiedad `InitialDirectory` especifica el directorio inicial desplegado por la caja de dialogo de archivo. La propiedad `Filter` determina el formato de archivos que podrán ser presentados en la caja de dialogo, para desplegar la caja de dialogo de archivo se utiliza su método `ShowDialog`, la propiedad `FileName` mantiene es un string que contiene el nombre del archivo seleccionado.

Suponga un ejercicio donde tiene un control `OpenFileDialog` y otro `PictureBox`, a través del cuadro de dialogo es posible seleccionar una imagen, misma que será cargada y presentada por el control de imagen (*.bmp, *.emf, *.wmf, *.ico, *.cur, *.jpg, *.jpeg, *.png, *.gif), es posible utilizar el método `Filter` para especificar solamente los archivos que tengan el formato especificado (entre formato y formato se utiliza como separador el símbolo `|` *pipe*):

```
ofd.Filter = "formato 1|*.jpg|formato 2|*.bmp"
ofd.ShowDialog()
pb.Image = System.Drawing.Image.FromFile(ofd.FileName)
Dim dr
dr = ofd.ShowDialog()
MsgBox(dr)
```

La siguiente línea descarga el control de imagen:

```
pb.Image = Nothing
```

ColorDialog

El control **ColorDialog** representa un dialogo preconfigurado que despliega la caja de dialogo de color estándar, permitiendo al usuario seleccionar el color o definir un color personalizado, para desplegar la caja de dialogo de color se utiliza su método `ShowDialog`, la propiedad `color` mantiene el color seleccionado por el usuario.

Es posible agregar el control **ColorDialog** al formulario aunque esté no este visiblemente contenido en el formulario pero si visible en la bandeja de componentes o es posible agregar el control **ColorDialog** a través de código declarando un objeto de este tipo.

```
    REM Agregando un control ColorDialog al formulario
Private Sub BtnBgColor_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnBgColor.Click
    CD.ShowDialog()
    LblText.BackColor = CD.Color
End Sub
    REM Declarando un objeto tipo ColorDialog via código
Private Sub BtnForeColor_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnForeColor.Click
    Dim FC As New ColorDialog
    FC.ShowDialog()
    LblText.ForeColor = FC.Color
End Sub
```

FontDialog

El control **FontDialog** representa un dialogo preconfigurado que despliega la caja de dialogo de fuente estándar, por default la caja de dialogo muestra cajas de listas para la fuente, estilo-fuente, y tamaño, cajas de chequeo para efectos como subrayado o tachado, también muestra un ejemplo de como la fuente podría aparecer, para desplegar la caja de dialogo de fuente se utiliza su método `ShowDialog`, la propiedad `font` mantiene el color seleccionado por el usuario.

Es posible agregar el control **FontDialog** al formulario aunque este no esté visiblemente contenido en el formulario pero si visible en la bandeja de componentes o es posible agregar el control **FontDialog** a través de código declarando un objeto de este tipo.

```
Private Sub BtnFont_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnFont.Click
    Dim FD As New FontDialog
    FD.ShowDialog()
    LblText.Font = FD.Font
End Sub
```

PrintDialog

El control **PrintDialog** es una caja de dialogo preconfigurada, utilizada para seleccionar una impresora, escoger las páginas a imprimir y determinar otras características de impresión en aplicaciones Windows. Es posible habilitar al usuario para imprimir varias partes de sus documentos: imprimir todo, imprimir un rango de páginas o imprimir una selección. La propiedad `Document` se establece con un objeto `PrintDocument`, para desplegar la caja de dialogo de impresión utilice el método `ShowDialog`, la caja de dialogo de impresión almacena la configuración del usuario en el objeto `PrintDocument` y para imprimir utilice el método `Print`.

PrintPreviewDialog

El control **PrintPreviewDialog** es una caja de dialogo preconfigurada, para presentar como podría ser el documento cuando se imprima. La propiedad `Document` se establece con un objeto `PrintDocument`, el cual tiene propiedades que describen que será impreso y la habilidad para imprimir dentro de una aplicación Windows, para desplegar la caja de dialogo de impresión-previa utilice el método `ShowDialog`,

PageSetupDialog

El control **PageSetupDialog** es una caja de dialogo preconfigurada que permite al usuario manipular la configuración de páginas, incluyendo margenes y orientación del papel. La propiedad `Document` se establece con un objeto `PrintDocument`, para desplegar la caja de dialogo de PageSettings utilice el método `ShowDialog`, la selección del usuario se almacena en la propiedad `PageSettings` y debería entonces ser copiado al objeto `PrintDocument`.

Dim as

Visual Basic .NET no permite el uso del tipo de dato `Variant` y todas las variables deberán ser declaradas por las instrucciones `Dim` *`nombreVarible`* `as tipoDato`.

`Dim` es la abreviatura de **Dimensión**, que sirve para reservar espacio para la variable. La sentencia **Dim** especifica el

nombre de la variable y su tipo de dato asociado.

Las variables deben tener un nombre único llamado **Identificador**, los identificadores se forman siguiendo las siguientes reglas:

- Comenzar con una letra o guión bajo (*underscore*)
- Deben contener letras, digitos y guiones bajos.
- No pueden ser palabras reservadas.

Recomendaciones:

- El nombre de las variables deberían comenzar con una letra
- El nombre de las variables deberían distinguir cada nueva palabra escribiendo la primer letra en mayúscula
- El nombre de las variables deberían ser lo suficientemente largo para ser significativo y lo suficientemente corto para ser manejable.

Una novedad en Visual Basic .NET es la declaración e inicialización de variables, ya que es posible hacer esto al mismo tiempo:

```
Dim nombreVariable as tipoDato = valorInicial
```

Si no se define un tipo de dato para la variable, entonces asume el tipo de dato por default que es `Object`.

Es posible declarar variables como se hacia en Visual Basic 6, si emplea la instrucción `Option Explicit Off`.

```
Option Explicit Off
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_
          Handles MyBase.Load
        v = "xxx" & 5
        MsgBox(v)
    End Sub
End Class
```

Visual Basic .NET incorpora la instrucción `Option Strict` si desea que los tipos de datos en las operaciones sean del mismo tipo y evitar la conversión y perdida de datos.

Ámbito de las Variables

- **Local**, visible únicamente a nivel procedimiento.
- **Pública y al inicio del formulario**, visible únicamente a nivel formulario.
- **Pública y al inicio del módulo**, visible únicamente a nivel proyecto.

InputBox

A través de la función `InputBox` es posible manipular la entrada de datos por parte del usuario, pudiendo almacenar en una variable los datos ingresados por este.

```
Dim sRespuesta As String
sRespuesta = InputBox("Password : ", "Usuario", , 500, 500)
```

```
If sRespuesta <> vbNullString Then
    MsgBox("Validación de usuario y password pendiente...")
Else
    MsgBox("Es necesario proporcionar su password")
End If
```

El primer parámetro que recibe esta función es un texto indicativo para el usuario, un texto para el título de la ventana, un valor inicial para el campo donde el usuario ingresará datos, un número que representa la coordenada X y otro número que representa la coordenada los cuales indican la posición en donde se presentará la ventana de ingreso de datos.

Tipos de Datos

Los tipos de datos numericos en Visual Basic .NET se agrupan en dos categorias: enteros y de punto flotante.

Los tipos de datos numericos a su vez se agrupan en, **Byte, Short, Integer y Long**.

Los tipos de datos de punto flotante son **Single** y **Double**.

Los tipos de datos **Char** almacenan un sólo caracter en formato Unicode.

Los tipos de datos **String** almacenan una secuencia de caracteres Unicode.

Unicode es un sistema internacional de codificación de 16-bit que cubre valores para más de 45,000 caracteres. Un caracter Unicode es almacenado como un valor numerico sin signo de 16-bit, de 0 a 65535. Los primeros 128 (0-127) caracteres Unicode corresponden al conjunto de caracteres ASCII. Del 32 al 127 corresponden a caracteres alfanumericos y simbolos para un teclado US. Del 128 al 255 representan caracteres especiales.

El tipo de dato **Boolean** es un valor sin signo el cual es interpretado como falso o verdadero.

El tipo de dato **Date** almacena fechas y tiempo, el rango de valores posible es del 12:00:00 AM Enero 1 al 31 de Diciembre del 9999 11:59:59 PM. Los valores para las fechas deben ser encerrados entre el signo # y con el formato **m/d/aaaa**, por ejemplo #12/7/1971#.

El tipo de dato **Object** es el tipo de dato universal en Visual Basic .NET y es también el tipo de dato por default para las variables que son declaradas sin especificar su tipo de dato.

Tipo de Dato	Tamaño	Rango		Función de Conversión	Validación de Conversión
		Desde	Hasta		
Boolean	2 bytes	True	False	CBool	Cualquier dato de tipo String o valor numérico
Byte	1 byte	0	255(unsigned)	CByte	De 0 a 255, las fracciones se redondean
Char	2 bytes	0	65535	CByte	Cualquier expresión String válida o valor en el rango de 0 a 65535
Date	8 bytes	0:00:00 Enero 1 del	11:59:59 PM Deciembre 31 del	CDate	Cualquier expresión

		0001 9999		válida de fecha y tiempo
Decimal	16 bytes	0 +/-79,228,162,514,264,337,593,543,950,335(sin punto decimal) 0 +/-7.9228162514264337593543950335(con 28 digitos en la parte decimal)	CDec	Números comprendidos en su rango
Double (punto foltante de doble precisión)	8 bytes	-1.79769313486231570E+308 -4.94065645841246544E-324 4.94065645841246544E-324 1.79769313486231570E+308	CDbl	Números comprendidos en su rango
Integer	4 bytes	-2,147,483,648 2,147,483,647	CInt	Números comprendidos en su rango
Long	8 bytes	-9,223,372,036,854,775,808 9,223,372,036,854,775,807	CLng	Números comprendidos en su rango
Object	4 bytes	Object	CObj	Cualquier expresión válida
Short	2 bytes	-32,768 32,767	CShort	Números comprendidos en su rango, las fracciones son redondeadas
Single (punto decimal de precision simple)	4 bytes	-3.4028235E+38 -1.401298E-45 1.401298E-45 3.4028235E+38	CSng	Números comprendidos en su rango, las fracciones son redondeadas
String	4 bytes	Depende de la plataforma en la que se integre	CStr	Aproximadamente 2 billones de caracteres Unicode.

Tipos de Datos Definidos por el Usuario : Structure

Visual Basic permite al desarrollador crear sus propios tipos de datos, esto es posible.NET a través del empleo de la instrucción `Structure`, ejemplo:

```
Structure Persona
        Dim sNombre As String
        Dim nEdad As Integer
        Dim dFechaNac As Date
End Structure
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_
                Handles MyBase.Load
        Dim Empleado As Persona
        Empleado.sNombre = "Gerardo Ángeles Nava"
        Empleado.nEdad = 33
        Empleado.dFechaNac = "21 / 9 / 1971"
        txtOutput.Text = Empleado.sNombre & vbNewLine
        txtOutput.Text += Empleado.nEdad & vbNewLine
        txtOutput.Text += Empleado.dFechaNac
End Sub
```

Cada variable declarada dentro de la estructura es llamado un **member**.

Definición una estructura vacia:

```
Dim Empleado As Persona = {}
```

Constantes

Una **constante** es aquella que almacena un valor que no es posible cambiar durante la ejecución del programa.

Se recomienda que el nombre de una constante sea escrito en su totalidad en mayúsculas.

```
Const PI As Double = 3.14159265
```

Si desea que la constante esté disponible para todos los formularios y módulos de la aplicación deberá ser declarada en un módulo y anteponer a su declaración la palabra reservada `Public`.

```
REM Escribir en un Módulo
Public Const PI As Double = 3.14159265
```

Operadores

Visual Basic .NET dispone de los siguientes operadores matemáticos:

Operador	Descripción
+	Suma
-	Resta
*	Multiplicación
/	División
\	División entera (parte entera de la división)
Mod	Residuo (resto de la división entera)
^	Exponenciación (elevar a una potencia)
&	Concatenación de Cadenas

```
Dim nInc As Integer
For nInc = 1 To 10
    If nInc Mod 2 = 0 Then
        txtOutput.Text += nInc & " es par " & vbNewLine
    Else
        txtOutput.Text += nInc & " no es par " & vbNewLine
    End If
Next
txtOutput.Text += "10/0 = " & CStr(10 / 0)
```

La división entre 0 no está permitida en los cálculos matemáticos ya que produce un resultado infinito, en la versión Visual

Basic 6 hacer esto produce un error de ejecución 11 "*División entre 0*", pero en Visual Basic .NET se maneja esta situación automáticamente y muestra un valor de ***Infinito***.

Ponga atención en el siguiente código y tenga cuidado al comparar resultados ya que el valor que regresa la excepción del cálculo matemático entre 0 es `Infinito` con la primer letra en mayúscula:

```
If CStr(10 / 0) = "infinito" Then
    MsgBox("manejador de cálculo")
Else
    MsgBox("No es lo mismo -infinito- e -Infinito-")
End If
If CStr(10 / 0) = "Infinito" Then MsgBox("manejador de cálculo")
```

Operadores Aritméticos

Los operadores aritméticos requieren operadores numéricos (`^,*,/,\,Mod,+,-`) y producen resultados numéricos.

Operadores String

El operador de concatenación (`&`) String requiere operadores String y producen resultados String.

Operadores Comparativos

Los operadores Comparativos requieren operadores Comparativos (`>,<,>=,<=,=,<>`) y producen un resultado lógico (`True or False`).

Operadores Lógicos

Los operadores Lógicos requieren operadores Lógicos (`NOT,AND,OR,XOR`) y producen un resultado lógico.

Operadores Abreviados

Visual Basic .NET incorpora nuevos operadores abreviados que facilitan la escritura de código, por ejemplo:

```
x = x + 1 'Antes escribia
x += 1 'Ahora puede escribir
```

Operadores Abreviados
+=
-=
*=
/=
\=
^
&=

Precedencia de Operadores

A continuación se muestra el orden de precedencia/prioridad/jerarquía de los operadores en Visual Basic .NET

Precedencia de Operadores
( )
^
-
*/
\
Mod
+-

Sentencias de Control

Trataremos de destacar las novedades y cambios en cuando lo referente a Visual Basic .NET y citaremos la referencia al Tutorial de Visual Basic 6.

Sentencias de Control

A manera de resumen recordaremos que `For` se utiliza cuando se conoce de antemano el número de iteracciones a ejecutarse. `Do` se utiliza cuando no se conoce de antemano el número de iteraciones a ejecutarse. Si al menos debe ejecutarse 1 iteración es necesario colocar la expresión condicional al final del ciclo. Evite ciclos infinitos, para ello asegurese de que estos ciclos tengan una expresión condicional de salida. Utilice `Until` a diferencia de `While` cuando requiera que la expresión condicional sea la contraria, por ejemplo con `While` la expresión condicional podría ser `A <> B` y con `Until` podría ser `A = B`.

La única sentencia de control que tiene un cambio relevante sintacticamente es `While` ya que antes se escribia `While ... Wend` ahora es necesario escribir `While ... End While`.

Visual Basic .NET incorpora 2 nuevos operadores lógicos para utilizarlos en las sentencias de control, a continuación se mostrará la tabla de operadores lógicos anteriores y nuevos:

Operadores Lógicos	Descripción
And	Las 2 expresiones deben ser verdaderas
Or	Alguna de las 2 expresiones es verdadera
Not	Negación del resultado de la expresión
Xor	Si 1 y sólo 1 de las expresiones es verdadera
AndAlso	Si la primer y segunda condición son verdaderas
OrElse	Si la primer o segunda condición es verdadera

Los nuevos operadores lógicos `AndAlso` y `OrElse` reciben el nombre de *sistema de corto-circuito*.

```
Dim nCalificacion As Integer = 0
If nCalificacion <= 0 AndAlso nCalificacion / 0 Then
    MsgBox("Error de lógica")
Else
    MsgBox("Continuar cálculo")
```

```
End If
```

Parece que el operador lógico `AndAlso` como `OrElse` nos ahorran escribir un `If` dentro de otro `If`/`else` y quizá evitar un error en tiempo de ejecución.

```
Dim nCalificacion As Integer = 0
If nCalificacion < 0 OrElse nCalificacion / 0 Then
    MsgBox("Error de lógica")
Else
    MsgBox("Continuar cálculo")
End If
```

Do ... Loop

La sentencia de control `Do ... Loop` es la sentencia general iterativa, la cual permite ejecutar repetitivamente un grupo de sentencias hasta que una condición sea cumpla. Existen cuatro versiones de la sentencia `Do ... Loop` :

```
1.
2.    Do While condición
3.       sentencias
4.    Loop
5.
6.
7.    Do Until condición
8.       sentencias
9.    Loop
10.
11.
12.   Do
13.      sentencias
14.   Loop While condición
15.
16.
17.   Do
18.      sentencias
19.   Loop Until condición
20.
```

Exit

La sentencia `Exit` permite salir inmediatamente de una decisión, ciclo o procedimiento.

Control Timer

A través del control `Timer` es posible ejecutar una instrucción en un intervalo de tiempo especifico, este `Timer` se activa estableciendo su propiedad `Interval` con el valor `True` y se ejecutará hasta que por medio de la acción del usuario lo detenga o se desactive el temporizador.

Cuando agrega un `Timer` este no es visible en el formulario pero si en la parte inferior de la pantalla (Bandeja de Componentes), justo debajo del formulario.

En Visual Basic .NET el nombre del control cambia su nombre por `Timer_Tick`

Por ejemplo si desea que se ejecute una acción cada segundo, será necesario cambiar el valor de la propiedad `Interval`

a `1000` milisegundos.

Para iniciar la ejecución del propio `Timer` es necesario establecer el valor de la propiedad `Enabled` a `True`.

Para detencer la ejecución del `Timer` es necesario invocar el método `Stop()`.

```
    Dim i As Integer = 0
    Private Sub btnInicio_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnInicio.Click
        Timer1.Enabled = True
    End Sub
    Private Sub Timer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles Timer1.Tick
        lblTic.Text = CStr(i)
        i += 1
    End Sub
    Private Sub btnFin_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnFin.Click
        Timer1.Stop()
    End Sub
```

Sentencia Rnd()

Visual Basic .NET provee la función `Rnd()` la cual genera un valor aleatorio (random) entre 0.0 y 1.0

```
Private Sub BtnMakeRnd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnMakeRnd.Click
    TxtOut.Text = Rnd()
End Sub
```

Note que cada vez que da clic al botón se genera un número aleatorio (random) y se muestra en la caja de texto, pero cierre el programa y ejecutelo de nuevo, notará que siempre dá la misma secuencia de números, es por ello que a estos números se les llame pseudo-aleatorios.

Si requiere obtener un número entero a partir del número generado, en vez de la fracción, entonces multipliquelo por 10 y después redondeelo utilizando la función de redondeo de la clase `Math`

```
Private Sub BtnMakeRnd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnMakeRnd.Click
    TxtOut.Text = Rnd()
    TxtOut2.Text = TxtOut.Text * 10
    TxtOut3.Text = Math.Round(TxtOut.Text * 10)
End Sub
```

Otra manera de obtener la parte entera es utilizar la función `Int`, si desea que el número generado se encuentre en un rango, solamente tiene que hacer una multiplicación del número del límite superior:

```
Private Sub BtnMakeRnd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnMakeRnd.Click
    If Trim(TxtRange.Text) <> vbNullString Then
        TxtOut4.Text = Int(Rnd() * (TxtRange.Text + 1))
```

```
    Else
        TxtOut.Text = Rnd()
        TxtOut2.Text = TxtOut.Text * 10
        TxtOut3.Text = Math.Round(TxtOut.Text * 10)
    End If
End Sub
```

Salida, los números generados comprenden el rango de 0 a límite superior.

Randomize

La función Rnd crea siempre la misma secuencia de números aleatorios, para crear un auténtico número aleatorio utilice el método `Randomize` la cual utiliza el reloj de la PC para crear un punto de inicio aleatorio el cual será utilizado después por la función Rnd.

```
Private Sub BtnRndmize_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnRndmize.Click
    VBMath.Randomize() ' Initialize random-number generator.
    TxtOut.Text = Rnd()
End Sub
```

El método `Randomize` pertenece a la Clase **Microsoft.VisualBasic.VBMath**.

Depuración, Tipos de errores

Existen 3 tipos básicos de errores, los fáciles de encontrar, solucionar y que no causan problemas de mantenimiento (sintaxis), los difíciles de encontrar, solucionar y que causan probelmas potenciales de mantenimiento (lógica):

1. Errores de compilación o de Sintaxis, son aquellos errores en los cuales la estructura sintáctica definida por el lenguaje no concuerda con la escrita en una sentencia, es decir, ocurren cuando el código escrito no sigue o viola las reglas del lenguaje, para ello Visual Studio reconoce un error de sintaxis subraya de azul el error y pasando por encima de este el mouse aparece un mensaje de error.
2. Errores en tiempo de ejecución, son todos aquellos errores no descubiertos por el compilador y que provocan una mal función del programa. El típico caso de una expresión entre tipos de datos distintos, que en Visual Basic 6 presenta el siguiente mensaje:
3.
4. `Run-time error '13': Type mismatch`

   Ahora en Visual Basic .NET un error en tiempo de ejecución genera una excepción, lo cual significa que esta situación excepcional requiere de un manejo especial.

5. Errores lógicos, son todos aquellos errores de tipo **HumanWare**, es decir, fallas en la forma de pensar de la persona y que trasciende al código, este tipo de errores son los que consumen más tiempo al tratar de hacer un programa libre de errores y son además los errores más dificiles de corregir, por ser el resultado de una planificación y razonamiento equivocados.

   Los errores lógicos son errores en el diseño o implementación de la solución que provocan un comportamiento incorrecto. Por ejemplo piense en que se requiere obtener un promedio y alguien diseño o implemento la siguiente formula:

   ```
   nPromedio = Calificacion1 + Calificacion2 / 2
   ```

Si la formula se escribe tal cual, la precedencia de operadores nos indica que sumará la Calificacion1 al resultado de la división de la Calificacion2 / 2, lo cual no es el promedio, lo que estariamos esperando es que primero hiciera una sumatoria y después realizara una división, el error aquí es que harían falta unos paréntesis.

```
nPromedio = (Calificacion1 + Calificacion2) / 2
```

La razón por la cual son los errores más dificiles de corregir es porque la mayoria de los escenarios funciona sin problema, pero vasta con que un escenario no se cumpla para hacer fallar el programa.

En el siguiente ejemplo el compilador no detecta que se trata de acceder a un índice que no existe, por lo que al intentar acceder se genera una excepción:

```
Dim a
a = "uno.dos.tres"
a = a.Split(".")
MsgBox(a(3)) ' el elemento 3 no existe, el arreglo inicia en el elemento 0 y termina en 2
```

Una de las potencialidades de Visual Basic a diferencia de otros poderosos lenguajes es que tiene integrada una herramienta de depuración de errores, con la cual es posible ejecutar el código:

- Paso a paso por instrucciones
- Paso a paso por procedimientos
- Paso a paso para salir
- Así como también es posible establecer puntos de interrupción en el código.

Una novedad en Visual Basic .NET es que los puntos de interrupción se mantienen aún si se cierra Visual Basic .NET

Ventana de Comandos

A través del depurador de Visual Basic .NET es posible abrir una ventana que nos permita conocer el valor que tienen las variables en cierto momento o para requerir de una ayuda auxiliar.

La **Ventana de Comandos** nos permite escribir por ejemplo un objeto y al escribir la notación de punto (**.**) aparecen las propiedades y métodos de este objeto, por ejemplo escriba:

```
>file.
```

Para abrir o activar esta ventana de clic en Debug/Windows/Inmediate, esta ventana es capaz de soportar dos modos:

1. En Modo Inmediato (*Inmediate*)
2. En Modo Comando (*Command*)

Para cambiar de un modo a otro simplemente escriba:

1. `>cmd`, si está en modo Inmediato y se desea pasar a modo Comando
2. `>immed`, si está en modo Comando y se desea pasar a modo Inmediato

Ventana de Comandos - Inmediata

La **Ventana de Comandos - Inmediata**, nos permite evaluar o conocer el valor de una variable por ejemplo si su programa utiliza una variable llamada `sNombre` y quiere conocer que valor tiene entonce en la ventana de comandos en modo inmediato, escriba:

```
?sNombre
```

A continuación se presentará su valor.

Control ListBox

El control `ListBox` hace visible una lista de items, donde el usuario puede seleccionar items en la lista utilizando los clics del mouse.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) -
    Handles MyBase.Load
    LstBox.Items.Add("Red")
    LstBox.Items.Add("Green")
    LstBox.Items.Add("Blue")
End Sub

Private Sub LstBox_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles LstBox.SelectedIndexChanged
    LstBoxOut.Items.Add(LstBox.SelectedItem)
End Sub

Private Sub LstBoxOut_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
_
    Handles LstBoxOut.SelectedIndexChanged
    LstBoxOut.Items.Remove(LstBoxOut.SelectedItem)
End Sub
```

Función IsNumeric

La función `IsNumeric()` regresa un valor booleano el cual indica cuando una expresión puede ser evaluada como un número.

Esta función es básica de una aplicación para validar la entrada correcta de datos de tipo numérico.

```
Private Sub btnIsNumber_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnIsNumber.Click
    If IsNumeric(TxtInput.Text) Then
        MsgBox("Si es número")
    Else
        MsgBox("No es número")
    End If
End Sub
```

Tendrá que hacer su función de validación tan robusta como la requiera, por ejemplo una validación previa en una entrada de datos, seria pensar que el usuario intento dar un número pero al escribir, sin desearlo digito un espacio, para la perspectiva del usuario novato, quizá vea que efectivamente escribio un número y no le toma importancia a los espacios en blanco, por lo tanto lo primero que tendriamos que realizar es la eliminación de los espacios en blanco, pero no sólo los que probablemente esten al inicio, sino los que también estén en la parte intermedia y los que estén en la parte final, para

la eliminación de los espacios en blanco al inicio y al final de la entrada del usuario podriamos utilizar una sola función Trim, pero, ¿Qué hay de los espacios en blanco intermedios?.

Trim, LTrim, RTrim

Visual Basic .NET provee mecanismos para la eliminación de espacios en blanco contenidos en un string a través de las siguientes funciones:

1. `LTrim`, regresa un string que contiene una copia de un string especifico al cual se eliminaron los espacios en blanco contenidos al inicio del string.
2. `RTrim`, regresa un string que contiene una copia de un string especifico al cual se eliminaron los espacios en blanco contenidos al final del string.
3. `Trim`, regresa un string que contiene una copia de un string especifico al cual se eliminaron los espacios en blanco contenidos al inicio y al final del string.

Control CheckBox

Un control `CheckBox` indica cuando un valor particular esta encendido o apagado, verdadero o falso, si o no, puede ser utilizado también para seleccionar múltiples items de una lista de opciones.

Control RadioButton

Un control `RadioButton` permite al usuario seleccionar sólo un item de una lista de opciones.

Evento KeyPress

El evento `KeyPress` es utilizado para capturar la tecla digitada por el usuario. Este evento distingue entre letras mayúsculas y minúsculas. El segundo argumento del este evento expone dos propiedades: Handled y KeyChar, la propiedad KeyChar es el caracter correspondiente a la tecla digitada. La propiedad Handled es un valor booleano, si se establece este valor como `True` entonces indicamos al formulario que no procese el evento.

El siguiente ejemplo evita que se digite un número:

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal _
    e As System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
    If IsNumeric(e.KeyChar) Then e.Handled = True
End Sub
```

El siguiente ejemplo válida que la entrada unicamente acepte números (permite borrar la entrada):

```
Private Sub TextBox1_KeyPress(ByVal sender As Object, ByVal _
    e As System.Windows.Forms.KeyPressEventArgs) Handles TextBox1.KeyPress
    REM If IsNumeric(e.KeyChar) Then e.Handled = True
    If (Asc(e.KeyChar)) >= 48 And (Asc(e.KeyChar)) <= 57 Or (Asc(e.KeyChar)) =
System.Windows.Forms.Keys.Back Then
        e.Handled = False
    Else
        e.Handled = True
    End If
End Sub
```

El siguiente ejemplo válida que la entrada unicamente acepte letras (permite borrar la entrada):

```
Private Sub txtLetras_KeyPress(ByVal sender As Object, ByVal _
    e As System.Windows.Forms.KeyPressEventArgs) Handles txtLetras.KeyPress
    If (Asc(e.KeyChar)) >= 65 And (Asc(e.KeyChar)) <= 90 Or _
             (Asc(e.KeyChar)) >= 97 And (Asc(e.KeyChar)) <= 122 Or _
             (Asc(e.KeyChar)) = System.Windows.Forms.Keys.Back Then
        e.Handled = False
    Else
        e.Handled = True
    End If
End Sub
```

Control Windows Media Player

El control **Windows Media Player** reproduce video y archivos de sonido en distintos formatos, incluyendo MPEG, AVI, WAV y MIDI.

Este control no es parte de la caja de herramientas estándar, para agregarlo seleccione Add/Remove ToolBox Items/COM/Windows Media Player

```
Private Sub BtnOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnOpen.Click
    Dim OFD As New OpenFileDialog
    OFD.ShowDialog()
    wmp.openPlayer(OFD.FileName)
End Sub
```

Try ... Catch

Sin duda una de las grandes novedades de Visual Basic .NET es la instrucción `Try - Catch`.

El mecanismo `Try - Catch` sirve para atrapar errores, tal como lo hace On Error Goto, Resume y Resume Next pero estructural y conceptualmente diferente.

Con el mecanismo `Try - Catch` es posible escribir manejadores de errores estructurados ofreciendo una manera eficaz de resolver los errores en tiempo de ejecución.

Los mecanismos previos a `Try - Catch` continuan vigentes, incluso existe un nuevo método llamado `Err.GetException`, que obtiene la excepción que ocurrio al momento de generarse el error.

¿**Qué es un error en tiempo de ejecución**?, no es más que un error inesperado del cual un programa en Visual Basic .NET no se puede recuperar, por no poderse completar una instrucción entonces Visual Basic .NET no tiene instrucciones precisas que ejecutar en estos casos cuando se genera el error.

Para lidiar con los errores en tiempo de ejecución Visual Basic tiene este nuevo mecanismo `Try - Catch` para controlar los errores, ya que este mecanismo atrapa el error definiendo las acciones a seguir cuando se identifica el error.

Algo importante a anazalir y diseñar es cuando y como integrar un mecanismo `Try - Catch`. Así que debera emplearse en aquellas situaciones suceptibles a error.

La manera en que debe utilizar este mecanismo es la siguiente:

```
Try
        sentencia(s) que pueden generar un error en tiempo de ejecución
Catch
        (el error se genero)
        sentencia(s) que definen las acciones a seguir ya que se genero el error
Finally
        sentencia(s) que reestablecen las condiciones antes de generarse el error.
End Try
```

La palabra reservada `Finally` es opcional.

El código contenido dentro de un bloque `Try` recibe el nombre de *código protegido*, porque evita la interrupción del programa o aplicación y ejecuta las instrucciones contenidas en `Catch`.

Para probar más de una condición de error en tiempo de ejecución utilice la sentencia `Catch When`:

```
Try
        'Evaluar expresión
Catch When Err.Number = 13
        'Ocurrio un error de tipo de datos, no coinciden los tipos
Catch When Err.Number = 6
        'Ocurrio un error en el control del índice, desbordamiento
Catch
        'Controlar el error
End Try
```

Por último, este mecanismo al igual que las sentencias de control tiene una manera de salir del bloque ya sea este `Try` o `Catch` a través del empleo de la instrucción `Exit try`, pero si contiene una sentencia `Finally` el código contenido en esta si se ejecuta, únicamente aplica para `Try` y `Catch` pues el objetivo de `Finally` es precisamente el ejecutar pase lo que pase su contenido.

```
Try
        'sentencia(s) que pueden generar un error en tiempo de ejecución
        'Bajo determinada condición Exit try
Catch
        '(el error se genero)
        'sentencia(s) que definen las acciones a seguir ya que se genero el error
        'Bajo determinada condición Exit try
Finally
        'sentencia(s) que reestablecen las condiciones antes de generarse el error.
End Try
```

Err

Recordemos que el objeto Err tiene propiedades utiles como `Number` para obtener el número de error que se produjo, `Description` para obtener una descripción del error ocurrido, todo ello referente al último error ocurrido.

Módulo

La novedad en Visual Basic .NET en lo referente a módulos es que ahora el módulo tiene instrucciones de inicio y fin que lo delimitan:

Otra novedad en Visual Basic .NET es que los argumentos de los procedimientos se pasan `ByVal`, es decir, no se modifica el valor original de la variable sino que se manda una copia del valor, por lo que quizá sea necesario la mayor parte de las veces indicar que se pasan `ByRef`.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    Dim i As Integer = 5
    multiplica(i)
    MsgBox(i)
End Sub

    Module miModulo
        Sub multiplica(ByVal j As Integer)
            j *= j
        End Sub
    End Module
```

Salida: 5

Si cambia `ByVal` por `ByRef` la **Salida** es `25`.

Al invocar un procedimiento Visual Basic .NET de manera automática encierra entre paréntesis los argumentos.

Sin duda la mayor novedad para las funciones es que ahora es posible utilizar la sintaxis `Return` para regresar el resultado y seguira vigente hacerlo como se hacia en Visual Basic 6, que era utilizar el nombre de la función y asignarle el resultado.

Cuando la función encuentre la palabra reservada `Return` de manera inmediata se detiene su ejecución y regresará si se establecio en su caso un valor.

Las funciones que son declaradas en los módulos estándar por defecto son públicas.

Recuerde que la diferencia entre una función y un procedimiento es que la función regresa valores y el procedimiento no.

Matrices/Arrays/Arreglos/Vectores

Optaremos por referirnos a esta estrutura como **Arreglo**.

La novedad para los arreglos en Visual Basic .NET es que el índice comienza en `0` y **no** podrá utilizarse la opción `Option Base` para redefinirlo, es decir, darle cualquier otro valor.

Efectos de las arreglos basados en `0`:

```
    Dim aColores(2) As String
    aColores(0) = "Rojo"
    aColores(1) = "Verde"
    aColores(2) = "Azul"
```

- No es posible declarar arreglos con la palabra reservada `To`
- `LBound` regresará siempre el valor `0` porque el límite inferior de un arreglo es `0`
- `UBound` regresará como valor el número de elementos menos 1

```
Dim aNumeros() As Integer = {10, 20, 30, 40, 50}
Dim i As Integer
txtOut.Text += "Left Bound : " & LBound(aNumeros) & vbNewLine
txtOut.Text += "Upper Bound : " & UBound(aNumeros) & vbNewLine
For i = 0 To UBound(aNumeros)
    txtOut.Text += i & " : " & aNumeros(i) & vbNewLine
Next
```

- No es posible utilizar `Redim` en la declaración inicial
- No es posible utilizar `Redim` para cambiar la dimensión a un arreglo existente

1. **Arreglos Estáticos**, el número de sus elementos siempre será el mismo.
2. **Arreglos Dinámicos**, el número de sus elementos puede cambiar durante la ejecución del programa.

> Para los arreglos dinámicos debe tenerse en mente que al declarar el arreglo su tamaño será definido como el *número de elementos menos 1*.
>
> Una vez declarado un arreglo no es posible cambiar el número de dimensiones utilizando `Redim`, es decir si se declaro un arreglo unidimensional con `Redim` no podrá redefinirlo como bidimensional.

`Redim Preserve`, si modifica la dimensión de una matriz que ya contenga datos, estos se perderán, ya que al ejecutar la instrucción `Redim` el contenido del arreglo dinámico se definirá a su valor predeterminado `0` o `null`, muy bueno para eliminar el contenido de un arreglo, pero muy malo si se desea mantener los valores, por lo que la solución para no eliminar los valores al redefinir un arreglo es utilizar la instrucción `Preserve`. La única regla es respetar el número de dimensiones original.

Existe una limitación, ya que sólo es posible cambiar el tamaño de la última dimensión, si es que el arreglo tiene más de una dimensión. En el caso de arreglos unidimensionales podría parecer no tener restricción.

```
Dim aMultidimensional(,,,) As Integer
ReDim aMultidimensional(10, 20, 30, 40)
ReDim Preserve aMultidimensional(10, 20, 30, 80)
```

Clase Array

La Clase `Array` provee métodos para crear, manipular, buscar y ordenar arreglos, por lo que sirven como la clase base para todos los arreglos en el *runtime* del lenguaje común. Forma parte la biblioteca .NET.

La Clase `Array` contiene el método `Sort` el cual recibe como argumento un arreglo y su objetivo es ordenar el contenido del arreglo:

```
Dim i As Integer
Dim aNumeros(9) As Integer
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    For i = 0 To 9
        Randomize()
```

```
            aNumeros(i) = Rnd() * 9
        Next
        ShowElements(aNumeros, ListBox1)
    End Sub
    Private Sub ShowElements(ByRef a As Array, ByVal list As ListBox)
        For i = 0 To 9
            list.Items.Add(a(i))
        Next
    End Sub
    Private Sub BtnOrder_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles BtnOrder.Click
        Array.Sort(aNumeros)
        ShowElements(aNumeros, ListBox2)
    End Sub
```

Colecciones

Las colecciones se utilizan o sirven para contener objetos, por ejemplo:

```
        'Declarar una variable del tipo Control para representar controles de formulario
        Dim ctrl As Control
        Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
            Dim i As Integer = 1
            For Each Me.ctrl In Controls
                If ctrl.Name <> "x" Then
                    ctrl.Text = "boton " & i
                    i += 1
                End If
            Next
        End Sub
```

Más sobre colecciones

Manipulación de Archivos de Texto

En Visual Basic .NET existen nuevas funciones para el manejo de archivos de texto proporcionandas por el objeto `Filesystem`:

- `FileOpen(numeroArchivo,directorioNombreArchivo,modoApertura)`, **abre** un archivo de texto para entrada y salida.

  Donde el primer argumento es un número entre 1 y 255 cuya función es controlar los archivos abiertos que tiene un programa, en el segundo argumento se especifica la ruta y nombre del archivo que deseamos abrir y por último es necesario especificar en que modo se abrira el archivo, es decir, si deseamos agregar datos nuevos utilizamos el método `Append`, si deseamos leer su contenido utilizamos `output`, si deseamos escribir `input` o también es posible abrirlo en modo binario `binary` o en modo aleatorio `random`.

  ```
  ofd.Filter = "txt|*.txt|ini|*.ini|log|*.log|inf|*.inf"
  ofd.ShowDialog()
  FileOpen(1, ofd.FileName, OpenMode.Input)
  ```

  Precaución, tenga cuidado con el *modo* de apertura de archivos ya que si abre un archivo con `Output` se elimina el contenido del archivo dejandolo vacio y en espera de nuevos datos. Cuando requiera leer su

contenido utilice `Input`.

- `LineInput`, **lee** una línea de entrada desde el archivo de texto.

```
        Dim sOutAux As String
        If Not EOF(1) Then
            sOutAux = txtOut.Text
            txtOut.Text = vbNullString
            txtOut.Text += sOutAux & nLine & " : " & LineInput(1) & vbNewLine
            nLine += 1
        Else
            MsgBox("Fin de archivo, no hay más líneas que leer")
        End If
```

- `EOF`, comprueba el **final** del archivo de texto.

```
        If EOF(1) Then MsgBox("Fin de archivo")
```

- `FileClose`, **cierra** el archivo de texto.

```
        FileClose(1)
```

Utilice `TextBox.Select(1,0)` para eliminar la selección de texto.

Como nota aclaratoria los archivos de texto son diferentes a los archivos de documentos, los cuales tienen códigos de formato, cuando nos referimos a archivos de texto entiendase un archivo que contiene únicamente caracteres reconocibles y quizá su formato sea `txt`, `ini`, `log` o `inf`.

- `PrintLine(numeroArchivo, objeto)`, esta función escribe datos con formato a un archivo secuencial

El siguiente ejemplo ilustra el ejemplo algunas de las funciones de manipulación de archivos de texto:

```
Dim nLine As Integer = 1
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    btnLine.Enabled = False
    btnClose.Enabled = False
    btnSave.Enabled = False
End Sub
Private Sub btnOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnOpen.Click
    ofd.Filter = "txt|*.txt|ini|*.ini|log|*.log|inf|*.inf"
    ofd.ShowDialog()
    FileOpen(1, ofd.FileName, OpenMode.Input)
    btnOpen.Enabled = False
    btnLine.Enabled = True
    btnSave.Enabled = True
End Sub
Private Sub btnLine_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnLine.Click
    If Not EOF(1) Then
        txtOut.Text += nLine & " : " & LineInput(1) & vbNewLine
        nLine += 1
    Else
        MsgBox("Fin de archivo, no hay más líneas que leer")
```

```
        btnLine.Enabled = False
        btnClose.Enabled = True
        txtOut.Select(1, 0)
    End If
End Sub
Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnClose.Click
    FileClose(1)
    btnSave.Enabled = False
    btnClose.Enabled = False
End Sub
Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnSave.Click
    sfd.Filter = "Archivos de texto txt|*.txt"
    sfd.ShowDialog()
    If sfd.FileName <> vbNullString Then
        FileOpen(2, sfd.FileName, OpenMode.Output)
        PrintLine(2, txtOut.Text)
        FileClose(2)
        MsgBox("Archivo almacenado")
    End If
End Sub
```

Manipulación de Cadenas

- El método `Concat` hace posible concatenar cadenas de texto.
- Utilice la función `UCase` o método `ToUpper`, para convertir una cadena a mayúsculas.
- Utilice la función `LCase` o método `ToLower`, para convertir una cadena a minúsculas.
- Utilice la función `Len` o método `Length`, para determinar el número de caracteres que tiene una cadena.
- Utilice la función `Mid` o método `Substring`, obtiene un número fijo de caracteres a partir de una posición dada de una cadena. (*El primer elemento de una cadena tiene el índice `0`*).
- Utilice la función `InStr` o método `IndexOf`, si una cadena se encuentra contenida en otra regresa la posición a partir de la cual encontro la cadena.
- Utilice la función `Trim` o método `Trim`, elimina los caracteres en blanco iniciales y finales de una cadena.
- Utilice el método `Remove`, para eliminar caracteres de la parte central de una cadena.
- Utilice el método `Insert`, para agregar caracteres a la parte central de una cadena.
- Utilice la función `StrCmp`, compara cadenas y detecta diferencias en el uso de mayúsculas y minúsculas, regresa los siguientes valores:
    - `-1`, sorts ahead
    - `0`, las cadenas son iguales
    - `1`, sorts after

Ejemplo:

```
Dim s As String
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    lblTitulo.Text = s.Concat("   ", "www", ".", "informatique", ".", "com", ".", _
             "mx", " ", "Tutoriales de Programación   ")
End Sub

Private Sub btnLower_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnLower.Click
    txtOut.Text = lblTitulo.Text.ToLower
End Sub

Private Sub btnUpper_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnUpper.Click
```

```
    txtOut.Text = lblTitulo.Text.ToUpper
End Sub

Private Sub btnLen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnLen.Click
    txtOut.Text = lblTitulo.Text.Length
End Sub

Private Sub btnSubStr_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnSubStr.Click
    txtOut.Text = lblTitulo.Text.Substring(14)
End Sub

Private Sub btnIndexOf_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnIndexOf.Click
    txtOut.Text = lblTitulo.Text.IndexOf("informatique")
End Sub

Private Sub btnTrim_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnTrim.Click
    txtOut.Text = lblTitulo.Text.Trim
End Sub

Private Sub btnRemove_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnRemove.Click
    txtOut.Text = lblTitulo.Text.Remove(lblTitulo.Text.IndexOf("informatique"), 12)
End Sub

Private Sub btnInsert_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnInsert.Click
    txtOut.Text = lblTitulo.Text.Insert(0, "http://")
End Sub

Private Sub btnStrComp_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnStrComp0.Click
    txtOut.Text = StrComp(lblTitulo.Text, lblTitulo.Text, CompareMethod.Binary) & _
            " is equal."
End Sub

Private Sub btnStrComp_Click_1(ByVal sender As System.Object, ByVal e As System.EventArgs) -
    Handles btnStrComp1.Click
    txtOut.Text = StrComp(lblTitulo.Text, UCase(lblTitulo.Text), CompareMethod.Binary) & _
            " sorts after."
End Sub

Private Sub btnStrCompL1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnStrCompL1.Click
    txtOut.Text = StrComp(lblTitulo.Text, "abc" & lblTitulo.Text & "abc", CompareMethod.Binary) & _
            " sorts ahead."
End Sub
```

Funciones Chr Asc

- `Chr`, regresa el caracter asociado con el código ASCII especificado.
- `Asc`, regresa un valor entero el cual representa el código ASCII del caracter correspondiente.

A través de estas funciones es posible determinar el código ASCII de una letra o su propio código ASCII, es decir, por ejemplo si deseamos obtener el código ASCII de la letra 'a' entonces empleados la función `Asc`, si conocemos el código ASCII y deseamos saber el caracter asociado a este entonces empleamos la función `Chr`

Visual Basic .NET cuando hace una comparación entre cadenas se basa en el código ASCII de cada uno de los caracteres

que compone la cadena por lo que una comparación entre las cadenas `"A" = "a"` da como resultado `False`, pues en realidad estaria comparando `65` y `97`.

```
Dim bResultado As Boolean = ("A" = "a")
MsgBox(bResultado & " : (" & Asc("A") & " = " & Asc("a") & ")")
```

Automatización

La **automatización** (*automation*) es una tecnología basada en el estándar de interoperabilidad denominado Modelo de Objetos Componentes (COM).

El objetivo de automatización es utilizar las funciones de una aplicación en otra aplicación.

Las aplicaciones basadas en Windows que exponen sus objetos se denominan aplicaciones **objeto** o **servidoras** y los programas que utilizan estos objetos se denominan aplicaciones **controladoras** o **clientes**.

Por ejemplo si desea utilizar Excel desde Visual Basic .NET :

1. Agregue una referencia excel (Project/Add Reference/Microsoft Excel *versión* Object Library)
2. Declare una variable de tipo objeto Excel

```
3.
4.          Dim xlsApp As Excel.Application
5.          xlsApp = CType(CreateObject("Excel.Application"), Excel.Application)
```

> **Nota**: si no agrega la referencia entonces no tendrá disponibles las referencias del objeto Excel. En Visual Basic 6 era posible emplear la técnica *enlace en tiempo de ejecución* en la cual se declaraba una variable de tipo objeto y en tiempo de ejecución se asignaba un tipo de aplicación específico, esta técnica ya no se recomienda.
>
> Lo recomendable es emplear la técnica de *enlace en tiempo de compilación* donde se asigna en tiempo de diseño un tipo a las variables que almacenan los objetos Automatización, para ser enlazados a los datos durante la compilación.
>
> A través de la función `CType` es el mecanismo que durante la compilación devuelve el tipo de aplicación específica a la variable de aplicación.

Ejemplo:

```
Dim xlsApp As Excel.Application
Dim xlsBook As Excel.Workbook
Dim xlsSheet As Excel.Worksheet
Dim sFile As String

xlsApp = CType(CreateObject("Excel.Application"), Excel.Application)
xlsBook = CType(xlsApp.Workbooks.Add, Excel.Workbook)
xlsSheet = CType(xlsBook.Worksheets(1), Excel.Worksheet)

xlsSheet.Cells(1, 1) = "www"
xlsSheet.Cells(1, 2) = "informatique"
xlsSheet.Cells(1, 3) = "com"
```

```
    xlsSheet.Cells(1, 4) = "mx"

    xlsSheet.Range("B1").Font.Bold = True
    xlsSheet.Application.Visible = False
    sFile = "c:\prueba" & Now.Day & Now.Month & Now.Year & Now.Hour & Now.Minute & Now.Second & ".xls"
    xlsSheet.SaveAs(sFile)
    xlsSheet = Nothing
    xlsBook.Close()
    xlsBook = Nothing
    xlsApp.Quit()
    xlsApp = Nothing
    MsgBox("Se ha creado el archivo : " & sFile)
```

Al crear este tipo de aplicaciones deberá tener un buen dominio de su funcionamiento, ya que puede observar a través del **administrador de programas** que al iniciar la aplicación se crea un objeto `excel.exe`, al emplear `Excel.Application` desde Visual Basic .NET se crea otro objeto `EXCEL.EXE` y solo se cierran estos objetos hasta que se cierre la aplicación, por lo que será necesario una buena administración de las aplicaciones a fin de no dejarlas abiertas o que originen conflictos con otras aplicaciones que el usuario abra o cierre.

Componente Process

Es posible utilizar el método `Process.Start` para iniciar cualquier aplicación que se encuentre registrada en el Sistema. Si la extensión del archivo a ejecutar es reconocida por el Sistema no será necesario especificar su ubicación.

Por ejemplo para iniciar la calculadora de Windows:

```
    System.Diagnostics.Process.Start("calc.exe")
```

Una vez que inica una aplicación Visual Basic .NET no puede detenerla. La solución es utilizar una de las nuevas funciones de Visual Studio referentes al control de procesos en Windows.

De clic a la sección **Components** agregue el componente `Process`, de clic a su propiedad `StartInfo` y asigne el valor **calc.exe** a su propiedad `FileName`, es posible pasar argumentos al abrir la aplicación para ello asigne los valores correspondientes a las propiedad `Arguments`.

De esta manera para abrir una aplicación utilice el método `Start()` y para cerrarla el método `CloseMainWindow()`. También es posible cerrar aplicaciones utilizando el método `Kill`, pero tenga cuidado ya que esta técnica no le permitira almacenar los cambios realizados en su trabajo..

Será posible incluir clases como `Threading` y `Diagnostics` para manipular vía código, no serán necesarios si únicamente emplea `Start` y `CloseMainWindow`

```
Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnCalc.Click
    System.Diagnostics.Process.Start("calc.exe")
End Sub

Private Sub btnOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnOpen.Click
    Proc.Start()
End Sub
```

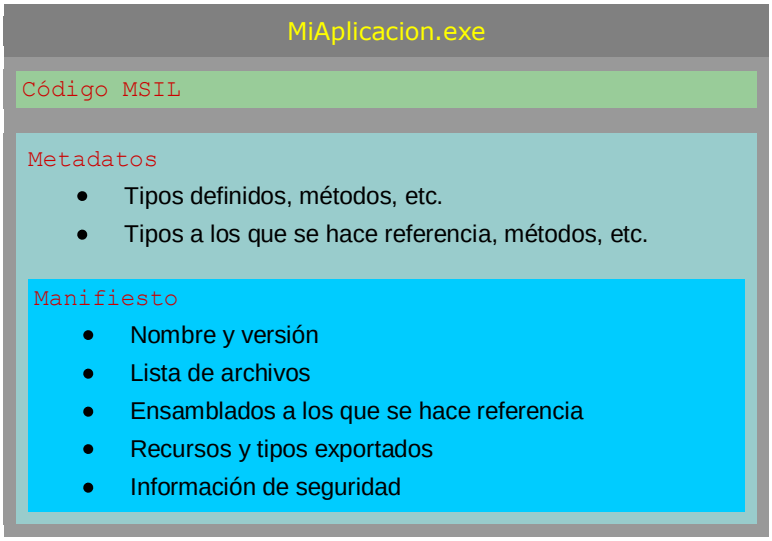
```
Private Sub btnClose_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnClose.Click
    Proc.CloseMainWindow()
End Sub
```

Instalación Distribuida

La distribución de aplicaciones en Visual Studio .NET se hacen agregando un **proyecto de instalación distribuida** a la solución que se desea distribuir. Será necesario configurar el proyecto de instalación distribuida para el tipo de instalación que desee ejecutar.

Una aplicación se ensambla en cuatro elementos:

1. **Código de Lenguaje Intermedio de Microsoft (MSIL)**, que es el código del programa compilado en un lenguaje comprensible por el runtime del lenguaje común.
2. **Metadatos**, información sobre los tipos, métodos y demás elementos definidos en el código.
3. **Manifiesto**, información sobre nombre, versión, lista de los archivos contenidos en el ensamblado, información de seguridad, información sobre el ensamblado.
4. **Archivos y recursos de soporte**.



Cuando la aplicación queda ensamblada, el sistema operativo no tendrá que registrar formalmente las aplicaciones para poder ejecutarlas ya que estas son comprensibles y autodescriptibles para el sistema.

Es posible instalar una aplicación Visual Studio .NET copiando unicamente el ensamblado generado en un equipo que tenga instalado **.NET Framework**.

Para crear un proyecto de instalación distribuida solo es necesario ejecutar el asistente para proyectos de instalación. Este proyecto puede ser personalizado unicamente definiendo sus propiedades.

Para crear un programa de instalación completo, los archivos de instalación distribuida deberán incluir el .NET Framework redistribuible.

Pasos para la creación de un proyecto de instalación distribuida:

- Abrir la aplicación

- File/New/Project
- Project Types/Setup and deployment Projects (según la versión que tenga instalada, es posible que existan 4 plantillas)
- Clic en plantilla setup project
- Escribir nombre y seleccionar ubicación
- Seleccionar Add to solution
- Clic en el botón [OK]
- (Se agrego un proyecto de instalación con el nombre que le asigno)
- Seleccione el proyecto de instalación
- Project/Add/Project Output
- (Aparece un cuadro de dialogo, para configurar su proyecto)
- Clic en el botón [OK]
- Seleccione el proyecto de instalación
- Project/Add/File
- Seleccionar archivos adicionales que se deseen incluir en su proyecto de instalación distribuida.
- La selección de archivos se ve reflejada en el explorador de soluciones
- El proyecto queda en espera de una próxima generación de solución y creará el programa de instalación dentro de la ubicación especificada.
- Se almacena un archivo `.msi` (windows installer) que podrá utilizar para instalar su aplicación.
- Buil/Configuration Manager (configure su proyecto para las plataformas a aplicar, depende la instalación que tenga)
- En configuración seleccione **release**
- Seleccione el proyecto de instalación
- Abra la ventana de propiedades y defina los valores para cada una de ellas, por ejemplo Autor
- Seleccione el proyecto de instalación, clic con el botón derecho, seleccione propiedades
- La acción anterior abre una ventana de Property pages
- Seleccione las opciones de su preferencia, para definir la conficuración del proyecto
- Build/Build Solution, esto compilara la solución, incluyendo la versión final y del proyecto de instalación distribuida
- Busque en el directorio **Release** el archivo `.exe` y pruebelo (aqui también se encuentra el archivo `.msi`).

Plantillas:

1. **Cab**, crea uno o más archivos contenedores, recomendable para descargas parciales remotas.
2. **Módulo**, propósito general
3. **Instalación**, utiliza Windows Installer
4. **Instalación Web**, utiliza Windows Installer y un servidor web para realizar instalaciones desde Internet.

Formularios

Un formulario en Visual Basic .NET hereda sus propiedades de la Clase `System.Windows.Forms.Form`.

Para abrir un formulario especifico en Visual Basic .NET digamos que más que novedad es algo un poco más tedioso, pues antes de poder abrir el formulario es necesario crear una variable del tipo del formulario que deseamos abrir y despues utilizar el método `Show` (que abre el formulario en modo no-modal) de esta variable objeto.

```
Dim frm2 As New Form2
frm2.Show()
```

La forma en que Visual Basic 6 abre formularios recibe el nombre de *generación implícita de instancias*, pero Visual Basic .NET exige que se declare específicamente una variable de tipo formulario antes de utilizarlo.

Ahora en Visual Basic .NET para abrir un formulario de manera **Modal** es necesario especificarlo a través del método `ShowDialog`,

```
Dim frm2 As New Form2
frm2.ShowDialog()
```

Para definir las coordenadas a partir de las cuales se desea ubicar el formulario en la pantalla en Visual Basic .NET existe una propiedad llamada `DesktopBounds`, la cual únicamente puede ser leida o definida en tiempo de ejecución y recibe como argumentos las dimensiones de un rectángulo en pares (esquina superior izquierda y esquina inferior derecha). Estas coordenadas se expresan en píxeles y toman como referencia la parte superior izquierda de la pantalla.

```
Dim oRect As New Rectangle(0, 0, 300, 300)
frm2.DesktopBounds = oRect
```

Otra forma de establecer la posición de un formulario durante el tiempo de diseño, pero con menos opciones es la propiedad `StartPosition`, cuyo valor de argumento define la posición que tendrá el formulario.

```
frm2.StartPosition = FormStartPosition.CenterScreen
```

Es posible manipular el estado minimizado, maximizado y restaurado de las ventanas a través de las propiedades correspondientes:

```
WindowState = FormWindowState.Minimized
WindowState = FormWindowState.Maximized
WindowState = FormWindowState.Normal
```

También es posible establecer el tamaño máximo o mánimo de las ventanas:

```
Dim nMax As New Size(200, 200)
MaximumSize = nMax
WindowState = FormWindowState.Maximized
```

StreamReader

La Clase `StreamReader` es una opción más del .NET Framework para la manipulación de archivos de texto. Esta clase entre otros contiene un método `ReadToEnd` cuya objetivo es leer un archivo desde la posición actual hasta el final.

Para hacer uso de la esta clase es necesario incluir la biblioteca de Clase `System.IO` al principio del código del formulario.

```
Imports System.IO
Public Class Form1
.
.
.
Dim sr As StreamReader
sr = New StreamReader("c:\Test.txt")
txtOut.Text = sr.ReadToEnd
```

```
sr.Close()
```

La Clase `StreamReader` está diseñada para la entrada de caracteres, el archivo a ser leido puede ser abierto utilizando la función `File.OpenText(path)`, donde *path* especifica la ubicación del archivo, así como el archivo de entrada.

Una vez que el **reader** (lector) tiene asignado un archivo abierto una serie de métodos **stream reader** (flujo de lectura) pueden ser invocados para leer la información del archivo:

Stream = Flujo
Buffer = espacio de memoria diseñada para el almacenamiento temporal de datos

Método	Descripción
`Close`	Cierra el **StreamReader** y libera cualquier recurso del sistema asociado al **reader**
`Peek`	Regresa el próximo caracter disponible pero no lo consume.
`Read`	Lee el próximo caracter o próximo conjunto de caracteres de la entrada **stream**
`ReadBlock`	Lee una cantidad máxima de caracteres del **stream** actual y escribe los datos al **buffer**
`ReadLine`	Lee una línea de caracteres del **stream** actual y regresa los datos como string
`ReadToEnd`	Lee un **stream** desde la posición actual hasta el final del **stream**

StreamWriter

La Clase **StreamWriter** está diseñada para la salida de caracteres, el archivo de escritura puede ser asignado utilizando la función `File.CreateText(path)`, donde *path* especifica la ubicación del archivo, así como el archivo de salida:

- Si el archivo especificado no existe
- Si el archivo existe su contenido es sobreescrito

Una vez que el **writer** tiene asignado un archivo abierto, una serie de métodos **writer** pueden ser invocados para escribir información al archivo:

Stream = Flujo
Buffer = espacio de memoria diseñada para el almacenamiento temporal de datos

Método	Descripción
`Close`	Cierra el actual **StreamWriter** y el **stream** subyacente
`Flush`	Limpia el **buffer** para el actual **writer** y provoca que cualquier dato contenido en el **buffer** sea escrito al **stream** subyacente
`Write`	Escribe al **stream**
`WriteLine`	Escribe datos tal y como son especificados por los parámetros sobrecargados, seguidos de una línea de terminación.

Creación de controles en tiempo de ejecución

En Visual Basic .NET, es posible agregar controles en tiempo de ejecución, siguiendo una sintaxis similar a esta:

```
Dim btnOK As New Button
btnOK.Text = "OK"
btnOK.Location = New Point(312, 150)
Me.Controls.Add(btnOK)
```

Tenga cuidado de que sus controles al momento de agregarlos no queden debajo de otros controles ocultandolos ante la mirada del usuario.

Existen nuevas propiedades para los controles que hacen posible manejar automáticamente su posición `Dock` y `Anchor`, por ejemplo el valor de la propiedad `Dock` puede ser `Bottom` con lo cual el control se ajusta al tamaño del formulario y queda posicionado en la parte inferior de la pantalla:

```
btnDock.Dock = DockStyle.Bottom
```

Otros posibles valores son:

- `DockStyle.Top`
- `DockStyle.Fill`
- `DockStyle.Left`
- `DockStyle.Right`
- `DockStyle.None`, para anular el funcionamiento de los valores anteriores

Establecer el objeto de inicio

También en Visual Basic .NET es posible establecer el punto de inicio de la aplicación, ya que puede contener múltiples formularios es necesario especificar con cual iniciara la aplicación o también puede contar con múltiples procedimientos por lo que también podrá definir uno de ellos como punto de partida.

Para configurar el punto de inicio seleccione Project/Properties y especique el objeto de inicio (*Startup object*) que puede ser un formulario o procedimiento.

Si requiere de una aplicación Visual Basic .NET sin interfaz gráfica, es posible crear una **aplicación consola** la cual procesa datos de entrada y genera salidas mediante una consola de línea de comandos.

Gráficos

En Visual Basic .NET no existen los controles de dibujo pues la novedad será emplear los servicios de gráficos **GDI+** a través del espacio de nombres `System.Drawing`

Instrucciones como Circle, Line y PSet, son sustituidas por los métodos `DrawEllipse`, `DrawLine` y la estructura Point de la clase `System.Drwaing.Graphics`.

El sistemas de coordenas predeterminado de Visual Basic .NET utiliza píxeles en lugar de Twips

En Visual Basic .NET los controles no tienen un método **Move**, pero es posible desplazarlos rápidamente al actualizar las propiedades `Left`, `Top` o `Location` de los controles o también utilizando el método `SetBounds`.

Las propiedades **DragIcon** y **DragMode** ya no están disponibles en Visual Basic .NET aunque el evento `DragDrop` continua.

Visual Basic .NET soporta más tipos de formato para imagenes, el espacio de nombres `System.Drawing.Imaging` incluye funciones que trabajan con los formatos siguientes:

- BMP
- EMF
- EXIF
- GIF
- Icon
- JPEG
- MemoryBMP
- PNG
- TIFF
- WMF

A través de la utilización de funciones GDI+ del espacio de nombres `System.Drawing` que es una nueva `API` (*Interfaz de Programación de Aplicaiones*) será posible crear gráficos.

Sistema de Coordenadas

El punto de origen del sistema de coordenas es la esquina superior izquierda del formulario. El sistema de coordenadas predeterminado se conforma de renglones (**Eje Horizontal X**) y columnas (**Eje Vertical Y**), de elementos de imagen independientes del dispositivo (píxeles) los cuales representan los puntos más pequeños que un formulario puede ubicar.

Las coordenas ***(x,y)*** de la esquina superior izquierda de un formulario son siempre `(0,0)`.

Visual Basic .NET trabaja en conjunto con el controlador de vídeo de la máquina para calcular como se deben presentar los píxeles del formulario y como deben aparecer en la pantalla figuras como líneas, rectangulos, curvas y círculos.

Clase System.Drawing.Graphics

El espacio de nombres `System.Drawing` contiene diversas clases con las que es posible crear dibujos en un programa. Esta clase dispone de métodos y propiedades para dibujar figuras en un formulario.

A continuación se presenta una lista de figuras geométricas básicas y el método que emplea la clase `System.Drawing.Graphics` para crearlas:

Figura	Método	Descripción
Línea	`DrawLine`	Línea sencilla entre dos puntos
Rectángulo	`DrawRectangle`	Cuadrado o Rectángulo dado por cuatro puntos
Arco	`DrawArc`	Línea curva entre dos puntos
Círculo/Elipse	`DrawEllipse`	Figura elíptica contenida en un ractángulo
Polígono	`DrawPolygon`	Figura compleja con un número de puntos y lados variable
Curva	`Drawcurve`	Línea curva que pasa por un número variable de puntos
Bézier splines	`DrawBezier`	Curva dibujada utilizando cuatro puntos (puntos 2 y 3 son puntos de control)

Los métodos mencionados en la tabla crean figuras vacías, pero existen métodos con el prefijo **Fill** que dibujan figuras que rellenan automáticamente con un color.

Para hacer uso de la clase `System.Drawing.Graphics` es necesario crear un objeto de tipo `Graphics`, un objeto `Pen` o `Brush` para indicar los atributos de la figura, el objeto `Pen` se utiliza como argumento de los métodos cuando no se rellena con color. El objeto `Brush` se utiliza como argumento cuando se requiere un color de relleno.

También será necesario generar una instancia de la variable `Graphics` mediante el método `CreateGraphics` para Windows Form.

El espacio de nombres `System.Drawing.Graphics` se incluye en el proyecto de manera automática por lo que no es necesario importarla.

```
Dim Grafico As System.Drawing.Graphics
Dim Lapiz As New System.Drawing.Pen(System.Drawing.Color.Blue)
Grafico = Me.CreateGraphics
Grafico.DrawLine(Lapiz, 20, 30, 100, 80)
```

Este código si lo incluye en el evento `Load` del formulario dara la sensación de no hacer nada, esto es por un efecto que se produce con los gráficos, la solución es agregar el código en el método `Paint` del formulario.

Evento Paint

Al dibujar una figura, esta será visible mientras:

- Otro control o figura no se sobreponga
- Si se minimiza y maximiza la ventana del formulario

Para evitar que la figura desaparezca es necesario utilizar el evento `Paint` del formulario, para que cada vez que se refresque el formulario el gráfico también se dibujo de nuevo.

```
Private Sub Form1_Paint(ByVal sender As Object, ByVal e As System.Windows.Forms.PaintEventArgs) _
Handles MyBase.Paint
        Dim Grafico As System.Drawing.Graphics
        Dim Lapiz As New System.Drawing.Pen(System.Drawing.Color.Blue)
        Grafico = Me.CreateGraphics
        Grafico.DrawLine(Lapiz, 20, 30, 100, 80)
End Sub
```

Animación : Top - Left - Location - SetBounds

La **animación** es la simulación de movimiento generado por la visualización rápida de series de imágenes correlativas en la pantalla.

Visual Basic .NET no incluye el método **Move** (el cual permitia mover objetos en el sistema de coordenadas) pero es posible utilizar:

- `Left`, propiedad que mueve un objeto horizontalmente

- `Top`, propiedad que mueve un objeto verticalmente
- `Location`, propiedad que mueve un objeto a una posición especifico
- `SetBounds`, método que define los límites de un objeto a una posición y tamaño especificos

```
Private Sub btnIzq_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnIzq.Click
    btn.Left -= 10
End Sub

Private Sub btnDer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnDer.Click
    btn.Left += 10
End Sub
Private Sub btnUp_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnUp.Click
    btn.Top -= 10
End Sub

Private Sub btnDw_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnDw.Click
    btn.Top += 10
End Sub
Private Sub btnMove_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnMove.Click
    Dim newPos As New Point(100, 100)
    If btn.Top = 32 And btn.Left = 104 Then
        btn.Location = newPos
    Else
        btn.Top = 32
        btn.Left = 104
    End If
End Sub
Private Sub btnSize_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnSize.Click
    If btn.Width = 75 And btn.Height = 23 Then
        btn.SetBounds(btn.Left, btn.Top, btn.Width + 10, btn.Height + 10)
    Else
        btn.SetBounds(btn.Left, btn.Top, 75, 23)
    End If
End Sub
```

Opacidad en Formularios

La novedad gráfica en los formularios para Visual Basic .NET es la propiedad `Opacity` a través de la cual puede hacer un efecto de transparencia en sus formularios.

Tenga en cuenta que el valor de la opacidad está en el rango del 0 al 1.

```
Me.Opacity -= 0.1
```

Programación Orientada a Objetos

Se considerada un **lenguaje orientado a objetos** si soporta las siguientes tres características **Encapsulación**, **Herencia** y **Polimorfismo**.

La programación orientada a objetos es un excelente ejemplo de un mejoramiento creciente. Los objetos son piezas modulares con interfaces bien definidas que explican el uso apropiado de los objetos. Los objetos emplean encapsulación para prevenir accesos impropios a la estructura interna de un objeto. Los objetos soportan herencia para perfeccionar el

código reutilizado y diseño lógico.

Clases y Objetos

Un **Objeto** es una combinación de datos y acciones que pueden ser tratados como unidad.

Una **Clase** es una estructura de un objeto, un diseño que describe las propiedades (datos) y métodos (acciones) de un objeto.

Encapsulación

Encapsulación, referente a un grupo de propiedades y métodos que pueden ser tratados como una unidad u objeto, además de proteger el contenido interno de un objeto a través de una averia o referencia incorrecta por código externo. Con la apropiada encapsulación un objeto es solamente referenciado a través de una interfaz formal evitando **efectos laterales**, es decir, referencias inesperadas y cambios indeseables que ocurran en adición a el comportamiento intencionado.

Una de las reglas básicas de la encapsulación es que los datos de las clases deberían ser modificados o recuperados sólo a través de procedimientos apropiados, limitando interactuar al objeto con código externo y manteniendo las operaciones internas del objeto invisibles hacia el mundo exterior, de esta manera el contenido interno del objeto es protegido de daños accidentales o intencionados por código externo.

La encapsulación también permite controlar como los datos y procedimientos son utilizados, para ello se pueden utilizar los **modificadores** `Private` o `Protected` para evitar que un procedimiento externo ejecute un método de clase o evitar la lectura y modificación de datos en las propiedades y campos.

Data hiding es una técnica en la cual se declaran detalles internos de una clase como `Private` para prevenirlos de ser utilizados fuera de la clase.

Herencia

La **Herencia** describe la habilidad para crear una nueva clase basada en la existencia de una clase existente, donde esta clase existente recibe el nombre de **Clase Base** y la nueva clase derivada de la clase base es llamada **Clase Derivada**. La clase derivada hereda las propiedades, métodos y eventos de la clase base y puede ser personalizada agregando nuevas propiedades y métodos.

Visual Basic .NET agrega el mecanismo de **Herencia**, que en versiones anteriores del Lenguaje no la incluia, solo incorporaba ciertas características de la programación orientada a objetos.

La Herencia es un mecanismo por medio del cual una Clase puede adquirir las características de comportamiento e interfaz de otra Clase.

En Visual Basic .NET las clases se definen utilizando la siguiente sintaxis:

```
Public Class
.
.
.
```

```
End Class
```

Las propiedades de las clases cambian de sintaxis y ya no se utiliza **Property Get, Let** y **Set**.

Para que una Clase herede la interfaz y comportamiento de otra Clase existente se utiliza la palabra clave reservada `Inherits`.

Polimorfismo

El **Polimorfismo** es la habilidad de los objetos de diferentes clases para responder apropiadamente a nombres u operadores de métodos identicos, el polimorfismo permite utilizar nombres compartidos y el sistema podría aplicar el código apropiado para un objeto particular.

Agregar una Clase

La definición de una clase consiste de campos, propiedades y métodos, un campo es una variable en la clase y usualmente es privada, una propiedad es una programación constructora que tipicamente provee la interfaz para un campo en una clase, una propiedad contiene procedimientos especiales `Get` y `Set` que permiten al código externo hacer referencia al campo en un sentido que mantiene la encapsulación de datos, un método es una función o procedimiento dentro de una clase. La definición de la clase puede también contener métodos constructores que son invocados cuando un nuevo objeto es instanciado desde una clase. Es conveniente listar primero los campos, después las propiedades y los métodos constructores y por último cualquier otro método adicional.

La definición de una clase consiste de campos, propiedades y métodos, un campo es una variable en la clase y usualmente es privada, una propiedad es una programación constructora que tipicamente provee la interfaz para un campo en una clase, una propiedad contiene procedimientos especiales `Get` y `Set` que permiten al código externo hacer referencia al campo en un sentido que mantiene la encapsulación de datos, un método es una función o procedimiento dentro de una clase. La definición de la clase puede también contener métodos constructores que son invocados cuando un nuevo objeto es instanciado desde una clase. Es conveniente listar primero los campos, después las propiedades y los métodos constructores y por último cualquier otro método adicional.

```
Private|Public Class nombreClase
        campos
        propiedades
        constructores
        métodos
End Class
```

En Visual Basic .NET para agregar una **Clase** seleccione: **Project**/**Add Class** y escriba un nombre para la Clase.

Escriba las variables para su clase:

```
Public Class Persona
    Private sNombre As String
    Private nEdad As Integer
    Private bSexo As Boolean
End Class
```

A continuación escriba las funciones para sus operaciones, escriba el nombre de la propiedad y su tipo, presione **Enter** y de manera automática se agrega el código de la propiedad el cual es necesario completar:

```
Public Class Persona
    Private sNombre As String
    Private nEdad As Integer
    Private bSexo As Boolean
    Public Property Nombre() As String
        Get
            Return sNombre
        End Get
        Set(ByVal Value As String)
            sNombre = Value
        End Set
    End Property
End Class
```

Si necesita agregar un método a su clase, entonces escriba su procedimiento:

```
Public Sub sexoPersona()
    If bSexo Then
        MsgBox("Hombre")
    Else
        MsgBox("Mujer")
    End If
End Sub
```

Ejemplo de la Clase `Persona`:

```
Public Class Persona
    Private sNombre As String
    Private nEdad As Integer
    Private bSexo As Boolean
    Public Property Nombre() As String
        Get
            Return sNombre
        End Get
        Set(ByVal Value As String)
            sNombre = Value
        End Set
    End Property
    Public Property edad() As Integer
        Get
            Return nEdad
        End Get
        Set(ByVal Value As Integer)
            nEdad = Value
        End Set
    End Property
    Public Property sexo() As Boolean
        Get
            Return bsexo
        End Get
        Set(ByVal Value As Boolean)
            bsexo = Value
        End Set
    End Property
    Public Sub sexoPersona()
```

```
            If bSexo Then
                MsgBox("Hombre")
            Else
                MsgBox("Mujer")
            End If
        End Sub
    End Class
```

`Interfaz` de la Clase **Persona**:

```
    Dim Empleado As New Persona
    Empleado.Nombre = txtNombre.Text
    Empleado.edad = CInt(txtEdad.Text)
    Empleado.sexo = txtSexo.Text
    Empleado.sexoPersona()
```

Como puede notar las declaraciones de obtención y asignación de valores para las propiedades coinciden no en nombre ya que puede declarar la primer parte de las propiedades con un nombre y con otro la declaración de la propiedad por completo, es decir, la parte donde se asigna y recupera el valor, la única relación existente es por el valor que se asigna o por el valor que se regresa.

Atención es posible crear más de una Clase dentro de un módulo de Clase siempre y cuando las clases se encuentren delimitadas por las intrucciones de inicio (`Public Class`) y final (`End Class`) de la Clase.

Así que, si desea crear una Clase que herede de la Clase `Persona` puede incluir en el mismo módulo de Clase `Persona`, la Clase que hereda, por ejemplo `Director` y escribir dentro del bloque de la clase la declaración que indica que hereda el comportamiento y propiedades de otra Clase a través de la palabra reservada `Inherits`.

Debajo del código de la Clase `Persona` escriba:

```
    Public Class Director
        Inherits Persona
        Private sArea As String
        Public Property Area() As String
            Get
                Return sArea
            End Get
            Set(ByVal Value As String)
                sArea = Value
            End Set
        End Property
    End Class
```

En su interfaz agregue:

```
    Dim DirectorRegional As New Director
    DirectorRegional.Area = "Finanzas"
    MsgBox(DirectorRegional.Area)
```

Campos

Los **campos** proveen almacenamiento para los datos en un objeto y son tratados como variables, usualmente son privados, existen dos diseños para los campos:

1. Si los campos son declarados `Private` se hacen visibles sólo para los métodos dentro de la clase, lo cual incrementa el ocultamiento de los datos (data hiding) y minimiza la posibilidad de efectos laterales (side effects)
2. Consiste en una convención en el nombre de los campos, si estos comienzan con la letra **F** mayúscula será un indicador claro de que un campo-objeto comienza a referenciarse.

Propiedades

Los campos privados de una clase no pueden ser accesados por código externo, por lo que si es requerido que los campos sean leidos o cambiados, para ello será necesario incluir procedimientos de propiedades (*property procedures*) en la definición de la clase.

Los procedimientos de propiedades dan el control de clase sobre como los campos son asignados o regresados. El nombre del procedimiento de propiedad es hecho visible al código externo.

El procedimiento de propiedad `Get` tipicamente recupera un campo privado.

El procedimiento de propiedad `Set` tipicamente asigna un nuevo valor al campo privado.

Para que un código externo pueda ver el valor de un campo pero no pueda cambiar su valor es necesario que el campo sea sólo de lectura, lo cual es posible antecediendo al nombre del procedimiento de propiedad la palabra reservada `ReadOnly`, entonces VB.NET podría omitir el bloque `Set/End Set` porque es innecesario.

```
[ReadOnly] property nombrePropiedad as tipoDato
        Get
                return nombreCampo
        End Get
        [Set(ByVal valor as tipoDato)
                nombreCampo = valor
        End Set]
```

Métodos

Los **métodos** son procedimientos definidos dentro de la clase. Los procedimientos tienen acceso a todos los datos dentro del objeto incluso si son privados.

```
[Private|Public] Sub nombreMetodo([parámetros])
        sentencias
End Sub
[Private|Public] Function nombreMetodo([parámetros]) as tipoDato
        sentencias
End Function
```

Constructores

Un **Constructor** es un método especial que se ejecuta durante la creación de un objeto. Todos los métodos constructores son procedimientos llamados `New`. Una clase puede tener cero, uno o más métodos constructores.

Si una clase tiene más de un método constructor lo que distingue un método constructor de otro es el tipo de dato y número de parámetros que lo define.

```
Sub New([parámetros])
        sentencias
End Sub
```

Cuando se define a una clase derivada de otra clase, la primer línea de un constructor es tipicamente una invocación al constructor de la clase base. Una clase base es referenciada utilizando la palabra reservada `MyBase`.

```
Sub New([parámetros])
        MyBase.New([parámetros])
End Sub
```

Impresión

Si requiere imprimir, Visual Basic .NET utiliza como mecanismo la Clase `PrintDocument` (agregando el control al formulario o a través de código), en vez de hacerlo con el objeto **Printer** como se hace en Visual Basic 6.

Para la interfaz de impresión es posible utilizar los controles de cuadros de diálogo como PrintDialog, PrintPreviewDialog y pageSetupDialog

Si requiere imprimir varias páginas deberá crear un manejador de eventos `PrintPage` que imprima un documento página a página.

La Clase `PrintDocument` dispone de objetos como `PrinterSettings` que define la impresión predeterminada para una impresora, el objeto `PageSettings` que define la impresión predeterminada para una página o el objeto `PrintPageEventArgs`, que define la información de eventos de la página a imprimir.

El espacio de nombres `System.Drawing.Printing` contiene la clase `PrintDocument`, al agregar un control **PrintDocument** a un formulario, algunos objetos se agregan de manera automática al proyecto, pero será necesario importar al principio del formulario:

`Imports` System.Drawing.Printing

Cuando es invocado el método `PrintDocument`, el evento `PrintPage` es invocado para cada página que se imprima. El método `Graphics.MeasureString` cuantifica el tamaño del string y determina el número de caracteres adecuado y líneas para el string especificado, fuente, tamaño máximo y formato. El método `Graphics.PrintString` dibuja el string utilizando font, brush, destination point y format. La propiedad `HasMorePages` indica cuando una página adicional debería ser impresa.

Impresión de un gráfico

Importe la Clase `System.Drawing.Printing` al inicio del formulario:

```
Imports System.Drawing.Printing
```

```
Public Class Form1
.
.
.
```

Agregue un control `PrintDocument` (no es visible en su formulario, pero si debajo de este)

Escriba el código del manejador:

```
Private Sub imprimirImagen(ByVal sender As System.Object, ByVal ePrint As PrintPageEventArgs)
    ePrint.Graphics.DrawImage(Image.FromFile("c:\relog.ico"), ePrint.Graphics.VisibleClipBounds)
    ePrint.HasMorePages = False
End Sub
```

Agregue un botón (utilicelo como requiera)

```
Private Sub btnPrint_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnPrint.Click
    Try
        AddHandler printDoc.PrintPage, AddressOf Me.imprimirImagen
        printDoc.Print()
    Catch ex As Exception
        MsgBox("Ex : " & ex.ToString)
    End Try
End Sub
```

Impresión de un texto

A diferencia del ejercicio anterior, conoceremos como imprimir un texto, **sin agregar** un control `PrintDocument`, pero si agregando en su lugar código y también en vez de un manejador de evento, utilizaremos un procedimiento:

Importar la clase:

```
Imports System.Drawing.Printing
```

Escribir en lugar del manejador de evento un procedimiento:

```
Private Sub ImprimirTexto(ByVal sender As System.Object, ByVal ePrint As PrintPageEventArgs)
    ePrint.Graphics.DrawString(txt.Text, New Font("Arial", 11, FontStyle.Regular), _
             Brushes.Black, 100, 100)
    ePrint.HasMorePages = False
End Sub
```

En lugar de agregar un control `PrintDocument`, lo haremos via código:

```
Dim prnTxt As New PrintDocument
```

Agregar una caja de texto

Agregar un botón, para imprimir el contenido de la caja de texto, en el evento clic del botón, escribir:

```
Private Sub bntPrnTxt_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles bntPrnTxt.Click
    Try
        Dim prnTxt As New PrintDocument
        AddHandler prnTxt.PrintPage, AddressOf Me.ImprimirTexto
        prnTxt.Print()
    Catch ex As Exception
        MsgBox("Ex : " & ex.ToString)
    End Try
End Sub
```

Importante, tenga en cuenta las limitaciones que se tienen, pues no se permite imprimir texto que sobrepase el margen derecho del papel, tampoco se ajustan automáticamente las líneas cuando se alcanza el borde del papel, por ejemplo si imprime un archivo que no tiene retornos de carro al final de las líneas será necesario escribir código que lo controle. Tampoco es posible imprimir más de una página de texto, pues lo que sobrepase una página se ignora, por lo tanto no se imprime.

Para controlar la impresión de varias páginas es necesario crear una página virtual, para ello utilice el evento `PrintPage` o el método `Graphics.MeasureString`.

Impresión de un archivo

Importar clases

```
Imports System.IO
Imports System.Drawing.Printing
```

Debajo del código generado automáticamente por Visual Basic .NET (*Windows Form Designer generated code*) escriba

```
Private OptPrnPage As New PageSettings
Private StrToPrn As String
Private FontPrn As New Font("Arial", 12)
```

Agregar un control OpenFileDialog

Agregar un control de texto enriquecido

Agregar un botón para abrir un archivo de texto y en el evento clic escribir

```
Private Sub btnOpenFile_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnOpenFile.Click
    OFD.Filter = "txt|*.txt"
    OFD.ShowDialog()
    If OFD.FileName <> vbNullString Then
        Try
            Dim FS As New FileStream(OFD.FileName, FileMode.Open)
            RTB.LoadFile(FS, RichTextBoxStreamType.PlainText)
            FS.Close()
            StrToPrn = RTB.Text
        Catch ex As Exception
            MsgBox("Ex : " & ex.ToString)
```

```
            End Try
        End If
    End Sub
```

Agregar un control PrintDialog

Agregue un botón para mandar imprimir el archivo y en el evento clic escriba

```
    Private Sub btnPrnFile_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles btnPrnFile.Click
        Try
            prnFile.DefaultPageSettings = OptPrnPage
            StrToPrn = RTB.Text
            PD.Document = prnFile
            Dim DR As DialogResult = PD.ShowDialog
            If DR = DR.OK Then
                prnFile.Print()
            End If
        Catch ex As Exception
            MsgBox("Ex : " & ex.ToString)
        End Try
    End Sub
```

Dé doble clic para agregar al control `PrnFile` (PrintDocument1) el código del evento clic

```
    Private Sub PrnFile_PrintPage(ByVal sender As System.Object, ByVal e As _
        System.Drawing.Printing.PrintPageEventArgs) Handles PrnFile.PrintPage
        Dim nChars As Integer
        Dim nLines As Integer
        Dim sPage As String
        Dim sFormat As New StringFormat
        Dim rectAng As New RectangleF(e.MarginBounds.Left, e.MarginBounds.Top, _
                e.MarginBounds.Width, e.MarginBounds.Height)
        Dim MySize As New SizeF(e.MarginBounds.Width, e.MarginBounds.Height -
FontPrn.GetHeight(e.Graphics))
        sFormat.Trimming = StringTrimming.Word
        e.Graphics.MeasureString(StrToPrn, FontPrn, MySize, sFormat, nChars, nLines)
        StrToPrn = StrToPrn.Substring(0, nChars)
        e.Graphics.DrawString(sPage, FontPrn, Brushes.Black, rectAng, sFormat)
        If nChars < StrToPrn.Length Then
            StrToPrn = StrToPrn.Substring(nChars)
            e.HasMorePages = True
        Else
            e.HasMorePages = False
            StrToPrn = RTB.Text
        End If
    End Sub
```

Bases de Datos

Comenzamos con la novedad de que el nuevo estándar de Microsoft para trabajar con Bases de Datos es `ADO.NET` (Modelo de datos estándar para todos los programas en Microsoft Visual Studio .NET) el cual se basa en una nueva tecnología de acceso a datos de Microsoft llamada `ADO+`, este nuevo estándar sustituye a **RDO** y **ADO**. Desaparecen los controles **Data** y **ADO Data**

Anteriormente una base de datos se representaba en un programa como un objeto **recordset**, ahora será el objeto

`dataset` que es una imagen no relacionada de la tabla de la base de datos a la que se accede.

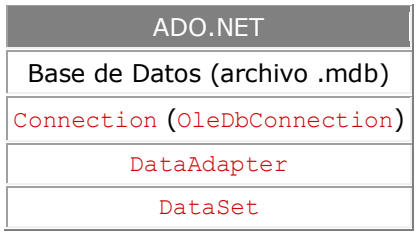
El formato interno de `ADO.NET` es **XML** (*Extensible Markup Language*) el cual se diseño para el intercambio de datos estructurado a través de Internet y otros contextos.

`ADO.NET` ofrece acceso a un mayor de formatos de base de datos y fué diseñado para su utilización en Internet.

Visual Studio y `ADO.NET` incluyen las herramientas necesarias para acceder al formato de archivos Access (y otros formatos), por lo que no es necesario tenerlo instalado.

ADO.NET

ADO.NET es la arquitectura de acceso a datos para .NET Framework y provee los objetos `Connection`, `DataAdapter`, y `DataSet` para facilitar el acceso a datos en una base de datos.



Conexión

Un objeto `Connection` establece una liga de la aplicación al archivo de base de datos, también especifica el *tipo* y *ubicación* del archivo de base de datos.

Para establecer una conexión a una base de datos seleccione **View/Server Explorer**

El **Explorador de Servidores** es una herramienta gráfica a través de la cual es posible establecer conexiones a fuentes de datos basadas en Internet, Cliente/Servidor o Locales, así como también es posible examinar la estructura de las tablas de una base de datos.

Para realizar una conexión haga clic en el botón **Connect to database** y configure la conexión. Por ejemplo si trabaja con Access entonces seleccione como proveedor **Microsoft Jet 4.0 OLE DB**, el cual es un componente diseñado para conectarse a bases de datos de Microsoft Access.

De clic en el botón **Probar Conexión**

Si la conexión fue satisfactoria entonces podra navegar a través del Explorador de Servidores sobre los objetos de la base de datos.

Creación de una Conexión

El primer paso para la creación de un objeto `Connection` es agregar un control `OleDbConnection` a la forma, el cual no es visible dentro de la forma pero si esta contenido en la bandeja de componentes. Este control esta contenido en el grupo **Data** de la caja de herramientas.

El segundo paso es seleccionar el control y establacer su propiedad `ConnectionString` seleccionando **New Connection**, entonces se despliega la ventana **Data Link Properties** (Propiedades de Vínculo de Datos), el proveedor de base de datos a seleccionar depende del tipo de base de datos a la cual se está conectado, por ejemplo para Access 2000 o Access XP se selecciona el proveedor *Jet 4.0 OLE DB Provider*.

Después de seleccionar el proveedor de base de datos se especifica la ubicación del archivo de base de datos, y si es requerido un login y contraseña se proporcionan para poder probar la conexión dando clic en **Probar Conexión**, si la conexión fue satisfactoria entonces es posible utilizar el objeto `Connection` para enviar y recibir datos entre la aplicación y la base de datos.

Los formatos de datos son diferentes entre los dos programas, por lo que la trasferencia de datos podría requerir un DataAdapter.

Adaptador de Datos - dataAdapter

Una vez que se ha establecido la conexión con la base de datos es necesario crear un **Adaptador de Datos** el cual permitira extraer información específica de la base de datos además de servir como base al objeto `dataset` el cual es una representación de los datos que se manipularan en la aplicación.

Para crear un adaptador de datos simplemente arrastre desde el Explorador de Servidores el icono gráfico de una tabla hasta el diseñador de Windows Forms, con lo cual se crean los objetos **Adaptador de Datos** y **Conector de Datos** en la bandeja de componentes.

Otra manera de crear un adaptador de datos es seleccionar la sección **Data** de la caja de herramientas y agregar un control `DataAdapter` y un `DataSet`.

Creación de un Adaptador de Datos

Una vez que se establece la conexión, el siguiente paso es crear un `DataAdapter` (Adaptador de datos). Un `DataAdapter` realiza el trabajo de pasar la información entre la base de datos y la aplicación. Un comando SQL es parte de un `DataAdapter`.

¿Por qué si ya se establecio la conexión, se requiere de un Adaptador de Datos?, porque el Adaptador de Datos trabaja como un lenguaje intérprete ya que la base de datos almacena datos es un sentido que la aplicación no podría entender así como la aplicación almacena datos en el sentido de que la base de datos no podría entender, también contiene SQL que especifica que información se accesa a través de la conexión. El Adaptador de Datos entiende ambos formatos de datos y los traduce apropiadamente para que los reciba la aplicación.

De clic al grupo **Data** de la caja de herramientas y Agregue un control `OleDbDataAdapter`, de manera automática inicia el Asistente de configuración del Adaptador de Datos, al dar clic en siguiente la segunda ventana del Asistente requiere que se especifique la conexión a utilizar para este Adaptador de Datos, al dar clic en siguiente aparece la tercer ventana del Asistente donde es posible seleccionar el tipo de consulta que se hará a la base de datos, pero como se esta utilizando un control `OleDbDataAdapter` la única opción es utilizar **Use SQL statements** (sentencias SQL). Si la base de datos de la aplicación fuera Microsoft SQLServer entonces se utilizaria un control **SQLDataAdapter** el cual también permite trabajar con procedimientos almacenados, al dar clic en siguiente se presenta la cuarta ventana del Asistente donde será necesario escribir una consulta SQL, donde es posible utilizar el *Constructor de consultas* o las *Opciones avanzadas*, al dar clic en siguiente aparece la última ventana del Asistente donde se notifica el resultado de la configuración.

El Asistente crea un Adaptador de Datos con un nombre por default, probablemente **OleDbDataAdapter1**, es posible seleccionar el control `DataAdapter` y cambiar el nombre a través de la propiedad `Name`.

DataSet

Una vez creado el adaptador de datos es necesario crear un objeto que represente los datos a utilizar en la aplicación, este objeto recibe el nombre de `DataSet` y constituye una representación de los datos proporcionados por la conexión y extraidos por el adaptador.

Un conjunto de datos puede contener información de una o varias tablas de la base de datos, resultado quizá de una instrucción SQL.

A diferencia de un **RecordSet** los `DataSet` solamente *representan* los datos de la base de datos. Cuando se modifica un `DataSet` no se modifican las tablas de la base de datos original ya que la modificación no se produce mientras no se envíe un comando que escriba los datos en la base de datos original.

Seleccione **Data/Generate Dataset** asigne un nombre al conjunto de datos y agreguelo al diseñador

Este DataSet o Conjunto de Datos aparece en la bandeja de componentes y además Visual Studio agrega un archivo llamado `NombreDataSet.xsd` al explorador de soluciones el cual representa el esquema de la base de datos en XML y describe las tablas, campos, tipos de datos, etc.

Generación de un Dataset

Una vez que se establece la Conexión y se crea un Adaptador de Datos, el siguiente paso es crear un `Dataset`. Un `Dataset` es una copia local temporal de la información en la tabla. **ADO.NET** soporta usos más avanzados que incluyen múltiples **Datasets** con múltiples tablas.

Para crear un `Dataset`, seleccione el Objeto Adaptador de Datos creado, de clic al botón derecho del mouse y seleccione *Generate Dataset* del menú emergente, aparece entonces un cuadro de dialogo para Generar el Dataset, donde se especifica que se está creando un nuevo Dataset, puede utilizar como prefijo **Ds** para el nombre del Dataset.

Finalmente es probable que en este punto en su aplicación existan 3 componentes en la bandeja: Connection, DataAdapter y Dataset. Donde la Conexión es una tuberia entre la aplicación y la base de datos, el Adaptador de Datos es un intérprete entre la aplicación y la base de datos y el Dataset es la traducción de una copia local de los datos en la base de datos, ya que los datos en el Dataset son traducidos quizá ahora pueden ser accesados por la aplicación.

Data-aware

Un control **data-ware** o Control Enlazado es un control que puede ser ligado o vínculado a un Dataset, cuando el control es ligado automáticamente despliega la información que recibe del Dataset. Los controles **data-ware** tienen una propiedad `DataBound`.

Controles Enlazados

Un **Control Enlazado** es aquel que esta vínculado con la fuente de datos cuando sus propiedades `DataBindings` pueden ser seleccionados campos válidos del conjunto de datos. Entre los controles enlazados se encuentran **TextBox**, **ComboBox**, **ListBox**, **ChackBox**, **RadioButton**, **DataGrid** y **PictureBox**.

Agregue una caja de texto y en su sección `DataBindings`, en la propiedad `Text` seleccione un campor de la base de datos.

Agregue un botón y en el evento clic agregue:

```
Ds.Clear() 'nombre del control en la bandeja de componentes
Adapter.Fill(Ds) 'nombre del control en la bandeja de componentes
```

El método `Fill` deberá cargar manualmente el adaptador y enlaza la caja de texto con la información del conjunto de datos.

Método Fill

Una vez que se utiliza y liga un control data-aware o enlazado, no despiegan información porque el Dataset inicialmente está vacío. El método `Fill` del objeto **DataAdapter** es utilizado para cargar el Dataset.

```
DataAdapter.Fill(Dataset)
```

Navegación de un Dataset

Cuando se utiliza un control data-aware se víncula o asocia su propiedad `Databindings` con un campo de una tabla de la base de datos y de está manera despliega el valor contenido en dicho campo en base al registro actual en el Dataset especificado. Al cambiar la posición del registro actual por consiguiente cambia el dato desplegado en el control data-aware empleado.

Cada formulario tiene un objeto `BindingContext` que mantiene el rastro de todos los recursos de datos asociados con la forma. Para referirse a una tabla dentro del Dataset utilice:

```
BindingContext(Dataset, "NombreTabla")
```

El objeto **BindingContext** tiene una propiedad `Position` que indica el registro actual y una propiedad `Count` que indica el Total de registros en el Dataset. El primer registro tiene la posición `0`, por lo que el último registro tiene la posición 1 menos el valor de `Count`.

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    DaContratos.Fill(DsContratos1)
    BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position = 0
    LblCount.Text = BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Count - 1
    InfoReg()
End Sub
Private Sub BtnPrev_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnPrev.Click
    If BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position = 0 Then
        MsgBox("Primer Registro")
    Else
        BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position -= 1
        InfoReg()
    End If
End Sub
Private Sub BtnNext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnNext.Click
```

```
    If BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position = _
            BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Count - 1 Then
        MsgBox("Último Registro")
    Else
        BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position += 1
        InfoReg()
    End If
End Sub
Private Sub InfoReg()
    LblRegNum.Text = BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position
End Sub
Private Sub BtnFirst_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnFirst.Click
    BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position = 0
    InfoReg()
End Sub
Private Sub BtnLast_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles BtnLast.Click
    BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Position = _
            BindingContext(DsContratos1, "TDetDato_Serv_Param_Asoc").Count - 1
    InfoReg()
End Sub
```

Manipulación de la Base de Datos

Una vez que se establece la conexión y se relacionan los datos con un control que los visualice, es necesario crear controles a través de los cuales se manipulen los datos obtenidos.

Al momento de cargar los datos el control utilizado para visualizar los datos apunta al primer registro, pero ¿Que hay si se requiere ver un registro especifico o si se desea avanzar hacia el siguiente o retroceder al anterior?.

`ADO.NET` almacena la información del registro actual y el número total de registros a través del objeto `CurrencyManager` que tiene cada conjunto de datos (DataSet) y cada Windows Form tiene un objeto `BindingContext` que se encarga de almacenar la información de todos los objetos **CurrencyManager** del formulario.

Para desplazarse al primer registro utilice:

```
Me.BindingContext(Ds1, "TParametro").Position = 0
lblPos.Text = Me.BindingContext(Ds1, "TParametro").Position
```

Para desplazarse al último registro utilice:

```
Me.BindingContext(Ds1, "TParametro").Position = Me.BindingContext(Ds1, "TParametro").Count - 1
lblPos.Text = Me.BindingContext(Ds1, "TParametro").Position
```

Para desplazarse al registro anterior utilice:

```
If Me.BindingContext(Ds1, "TParametro").Position = 0 Then
    MsgBox("Primer Registro")
End If
Me.BindingContext(Ds1, "TParametro").Position -= 1
lblPos.Text = Me.BindingContext(Ds1, "TParametro").Position
```

Para desplazarse al registro siguiente utilice:

```
        If Me.BindingContext(Ds1, "TParametro").Position = Me.BindingContext(Ds1, "TParametro").Count - 1
Then
            MsgBox("Último Registro")
        End If
        Me.BindingContext(Ds1, "TParametro").Position += 1
        lblPos.Text = Me.BindingContext(Ds1, "TParametro").Position
```

Donde, `"TParametro"`, es el nombre de la tabla a la que se accede, `lblPos.Text` controla la posición del registro actual.

Bases de Datos y DataGrid

Un control `DataGrid` presenta información a manera de tabla (renglones y columnas).

El acceso a datos es controlado por los objetos `DataSet` y `DataAdapter`

Para víncular un control `DataGrid` a una base de datos, se utilizan las propiedades `DataSource` y `DataMember`.

1. Establezca una conexión válida
2. Arrastre una tabla de la base de datos al diseñador
3. Genere un conjunto de datos (**Data/Generate Dataset**)
4. Víncule el control `DataGrid` a la base de datos.

    DataSource - DataSet.Tabla

    Una recomendación es que al generar el conjunto de datos, escriba el nombre de la base de datos

5. En el evento `Load` del Formulario, escriba:

```
6.
7.    Contratos1.Clear()
8.    OleAdapter.Fill(Contratos1)
```

    Donde `Contratos1` es el nombre del conjunto de datos (DataSet).

Modificación de la base de datos con DataGrid

Un objeto **DataSet** (conjunto de datos) contiene una copia original de la base de datos, si requiere modificar los datos se hará a través del objeto `DataAdapter` para almacenar los cambios.

Si la modificación de la base de datos se desea hacer a través de un control **DataGrid** entonces considere lo siguiente:

- Si la propiedad `ReadOnly` del control **DataGrid** contiene el valor `False` es posible modificar la base de datos.
- Si la propiedad `ReadOnly` del control **DataGrid** contiene el valor `True` **no** es posible modificar la base de datos.

Para almacenar los cambios realizados, escriba:

```
        Try
```

```
        OleAdapter.Update(Contratos1)
    Catch ex As Exception
        MsgBox(ex.ToString)
    End Try
```

Tip, si cambia el valor de la propiedad `BackColor` del **DataGrid** se produce un efecto en el cual el color de fondo del primer renglón aparece con el color seleccionado y el renglón siguiente aparece en blanco.

Web

La versión inicial de Visual Basic .NET incluye Internet Explorer 6. Para hacer uso de las funciones de Internet Explorer en una aplicación Visual Basic .NET es necesario agregar una referencia a la biblioteca de objetos COM `Microsoft Internet Controls` (*SHDocVw*), para ello seleccione **project/Add Reference**.

El objeto Internet Explorer está contenido en la Clase `InternetExplorer` la cual es miembro de la biblioteca `SHDocVw`.

Quizá el método que utilizará con frecuencia sea `Navigate` el cual abre un documento cuya ubicación se especifica a través de uno de sus parámetros llamado **URL**, **Flags** especifica si se agregará al historial o al caché de Internet Explorer, el único parámetro obligatorio es `URL` los demás son opcionales.

Para visualizar un documento web siga estos pasos:

1. Agregue la referencia COM `Microsoft Internet Controls`
2. Declare un objeto de tipo `SHDocVw.InternetExplorer`
3. Cree una instancia de `SHDocVw.InternetExplorer`
4. Utilice la propiedad `Visible` para ocultar o hacer visible IE
5. Utilice el método `Navigate` para mostrar un documento en el IE.

```
Dim oIE As SHDocVw.InternetExplorer
oIE = New SHDocVw.InternetExplorer
oIE.Visible = True
oIE.Navigate("file:///C:/webSite/informatique/vbNet/default.html#web")
```

Si todo marcha bien, al ejecutar su aplicación se deberá abrir el navegador Internet Explorer y visualizará en él el documento específicado.

Si requiere controlar los eventos de IE, entonces tendrá que especificarlo en la declaración a través de la palabra reservada `WithEvents`, además de declarar el objeto a nivel formulario:

```
Public WithEvents oIE As SHDocVw.InternetExplorer
```

Ahora tendrá que sobrecargar el evento que usted desee, por ejemplo:

```
Public WithEvents oIE As SHDocVw.InternetExplorer
Private Sub btnOpenLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles btnOpenLoad.Click
    oIE = New SHDocVw.InternetExplorer
    oIE.Visible = True
    oIE.Navigate(txtURL.Text)
End Sub
Private Sub oIE_DocumentComplete(ByVal pDisp As Object, ByRef URL As Object) _
```

```
    Handles oIE.DocumentComplete
    MsgBox("El documento ha sido cargado")
End Sub
```

Web Forms

Web Forms es un nuevo modelo de programación para interfaces de usuario de Internet basado en **ASP.NET** que sustituye a *WebClasses* y el **Diseñador de Web Forms** sustituye al *Diseñador de páginas DHTML*. El **Diseñador de Web Forms** es parte de Visual Studio .NET (también disponible para C# .NET).

Web Forms es el componente de diseño de ASP.NET (plataforma de desarrollo Web para Microsoft rediseñada a partir de cero basándose en .NET Framework), que permite crear y administrar interfaces de usuario de Internet o páginas web.

Visual Studio soporta **Web Forms** para la creación rápida y fácil de interfaces-usuario para aplicaciones web ASP.NET, **Web Forms** combina la velocidad y facilidad de un entorno de Desarrollo Rápido de Aplicaciónes (*RAD*, Rapid Application Development) con el poder de la compilación de los lenguajes de programación.

Cada Web Form está compuesta de dos archivos:

1. **Página Web Forms/User Interface Form** (contiene páginas HTML y los controles para crear la interfaz de usuario `.aspx`)
2. **Archivo de código correspondiente/Code-Behind** (Módulo de código que contiene el código que corresponde a la página Web Forms `.aspx.vb`)

En Visual Studio es posible ver los archivos ASPX y VB.NET como dos vistas de la misma página. El servidor web compilará ambos archivos y crea una nueva clase que contiene HTML estático, controles-servidor ASP.NET y código de la forma compilado en conjunto, la clase genera el HTML que envia al cliente cada vez que la página es requerida.

Las **Web Forms** (requieren de .NET Framework en el servidor para ejecutarse en la máquina-servidor) son similares a **Windows Forms** (requieren de .NET Framework para ejecutarse en la máquina-cliente) pues ambos son event driven, es decir, que se puede responder a eventos que ocurren en los controles-servidor de la página HTML. Los manejadores de eventos para los controles-servidor son escritos en el código detrás de la página utilizando sintaxis VB.NET.

Una aplicación Web puede contener módulos de código (`.vb`), documentos HTML (`.htm/html`), información de configuración (`Web.config`) y (`Global.asax`), así como otros componentes.

Las aplicaciones ASP.NET contienen un archivo especial llamado `Global.asax` utilizado para establecer cualquier objeto global requerido por la aplicación web. El archivo es compilado con la primer petición a una página de la aplicación web. Cuando este archivo es modificado, el archivo es recompilado y la aplicación web reiniciara en la próxima petición de una página.

Para crear una aplicación Web de ASP.NET se utilizan los controles de las secciones **HTML** o **Web Forms** de la caja de herramientas.

Los controles Web Forms son *controles de servidor*, es decir, que se ejecutan y pueden ser programados en el servidor web. Los controles HTML son por default *controles cliente*, es decir, que se ejecutan en el navegador del usuario final, aunque es posible configurar estos controles HTML como *controles de servidor* dando el valor **Server** a su propiedad `Runat`.

Los **Controles HTML** son los controles comunes utilizados en el HTML, los **Controles Web Forms** son más potentes y poseen propiedades, métodos y eventos.

No es necesario que un usuario tenga la última versión del navegador ya que Visual Studio .NET contiene la propiedad `targetSchema` en el objeto `DOCUMENT` para especificar una versión del navegador y soporta Internet Explorer 3.02, 3.0, 5.0 y Navigator 4.0, por default tiene el valor de Internet Explorer 5.0, así que el valor seleccionado afecta el código HTML generado y las funciones disponibles en Visual Studio.

Si requiere crear una aplicación Web en Visual Basic .NET, cree un proyecto **Aplicación Web ASP.NET** (ASP.NET Web Application), también requiere de Windows 2000 o Windows XP Professional, IIS, Extensiones de FrontPage 2000 y bibliotecas .NET Framework, si tiene XP Home Edition no podrá crear aplicaciones Web ASP.NET **localmente**, pero si podrá crearlas si accede a un Servidor Web Remoto configurado adecuadamente.

Se recomienda instalar IIS y Extensiones Front Page **antes** de instalar .NET Framework y Visual Studio .NET, si trata de instalar IIS y Extensiones Front Page después de .NET Framework deberá repararlo pues .NET Framework registra las extensiones a través de IIS.

Arquitectura Tres Capas (Three-Tier)

Aplicaciones sofisticadas que involucran bases de datos y son con frecuencia divididas en tres capas basadas en la partición lógica de servicios fundamentales:

Capa de **Presentación**/Presentation Layer, *navegador* en la máquina-cliente

Capa de **Aplicación**/Application Layer, servidor web IIS que contiene las *páginas ASP.NET*

Capa de **Datos**/Data Layer, servidor de base de datos que contiene los *archivos de base de datos y servicios ADO.NET*

Las aplicaciones de tres capas protegen los recursos del servidor de base de datos reduciendo el número de conexiones actuales a la base de datos. Las conexiones al servidor de base de datos se realizan por las aplicaciones del servidor en lugar de ser realizadas por las máquinas cliente. Los usuarios no requieren de conexiones individuales o persistentes al servidor de base de datos, lo cual no sólo protege los recursos de base de datos, también facilita el mantenimiento de cuentas y fortalece la seguridad de datos.

Capa de Presentación - Presentation Layer

La **Capa de Presentación** se ejecuta sobre la máquina del usuario (cliente) y provee de una interfaz de aplicación, un cliente robusto realiza procesos significativos con menos carga en la capa de aplicación, un cliente menos robusto tipicamente involucra un navegador (*browser*) que despliega HTML con procesos poco significativos y requiere de una mayor carga en la capa de aplicación.

Capa de Aplicación - Application Layer

La **Capa de Aplicación** provee varios módulos y servicios que son esenciales para la solución, incluyendo el procesamiento basado en las reglas de negocio. Si las reglas de negocio cambian solamente los servicios en la capa de aplicación necesitan ser cambiados para implementar los cambios a través del sistema.

La **Capa de Aplicación** también provee un mediador entre los manejadores de la capa de aplicación y la capa de datos. Los requerimientos del usuario para los servicios de datos son manejados por la capa de aplicación la cual puede reducir el número de conexiones a la base de datos, por ejemplo todos los usuarios pueden compartir una conexión entre la capa de aplicación y la capa de datos, antes que cada usuario (cliente) requiera de una conexión de datos.

Capa de Datos - Data Layer

La **Capa de Datos** es la responsable de todos los accesos a la base de datos requeridos por la solución, esta capa comunmente provee el soporte para agregar, eliminar, actualizar y recuperar información de la base de datos.

State Management

HTTP es un protocolo *stateless* (sin estado), es decir, cada petición de una nueva página web es procesada sin ningún conocimiento de peticiones de páginas previas.

State Management (administración de estados) se refiere a las técnicas en las cuales los desarrolladores mantienen el estado de una aplicación web a través de múltiples peticiones de páginas.

En ASP.NET existen varias opciones para que los desarrolladores mantengan el estado de un sitio web, algunas de ellas involucran mantener información en la máquina-cliente, otras en la máquina-servidor.

Técnicas de Administración de Estados del Lado del Cliente

Existen varias Técnicas de Administración de Estados del Lado del Cliente (Client-Side State Management Techniques) disponibles en ASP.NET, las cuales involucran almacenamiento de información en la máquina-cliente, un usuario puede manipular la información la cual puede resultar en un estado incorrecto y crear un compromiso potencial de seguridad, por ello que la administración de estados de lado del cliente debería ser limitado a aplicaciones no criticas o seguras y soluciones intranet:

View State

Una página web es re-creada en cada petición, sin esfuerzos de administración de estados, toda la información asociada con la página y los controles en las páginas podrían perderse. ASP.NET provee una facilidad llamada **View State** (estado de vista) que representa el estado de la página cuando fué procesada en el servidor. Cuando la página es enviada de regreso al servidor ASP.NET utiliza el **View State** para recuperar la información apropiada en la página. El **View State** es visible en el código HTML siendo un asunto potencial de seguridad.

Query String

Un **Query String** (Cadena de Consulta) es información que se agrega al final del URL. Un **Query String** tipicamente comienzan con el caracter `?` seguido de información especifica para la aplicación, por ejemplo:

```
http://informatique.com.mx?login?usuario=gangeles
```

Un **Query String** puede ser utilizado para intercambiar datos entre el cliente y el servidor, de una página a otra. Un **Query String** no es seguro porque la información es visible en el navegador, además que la mayoria de los navegadores tienen una longitud máxima de 255 caracteres para el URL, lo cual limita la información que puede ser enviada utilizando un

Query String.

Cookies

Una **Cookie** es un texto pequeño almacenado en la máquina-cliente, es decir, un archivo de texto y no un programa o *plug-in*, el navegador anexa la **Cookie** en cada nueva petición HTTP antes que enviarla al servidor para que los datos puedan ser leidos y respondidos apropiedamente.

La información almacenada en una **Cookie** puede ser expuesta y por lo tanto no es lo mejor para mantener información, además los usuarios podrían deshabilitar la opción en sus navegadores para no utilizar **Cookies**, entonces la aplicación no debería ser dependiente de un dato en la **Cookie**.

Técnicas de Administración de Estados del Lado del Servidor

Existen también varias Técnicas de Administración de Estados del Lado del Servidor (Server-Side State Management Techniques) disponibles en ASP.NET. Ya que las opciones del lado del servidor almacenan información fuera del alcance del cliente la información es más segura. Los estados de administración del lado del servidor deberían ser incluidos en todas las aplicaciones que requieran soluciones seguras:

Application State

Application State (Estado de Aplicación), es la suma de todos los archivos, páginas y código que reside en el servidor. Cuando se ejecuta una aplicación web, ASP.NET mantiene información referente a la aplicación en el **Application State**, el cual es creado la primer vez que un cliente hace una petición de un URL dentro de la aplicación ASP.NET, el **Application State** se mantiene en la memoria del servidor hasta que un servidor web se apaga o hasta que la aplicación se modifique.

El **Application State** permite a los desarrolladores crear variables de aplicación (*application variables*) que pueden establecerse y leerse a través de la duración de vida de la aplicación. Una variable de aplicación se compone de la palabra reservada `Application` seguido entre paréntesis (y comillas) del nombre de la variable, por ejemplo: `Application("UsuariosConectados")`, estas variables se crean automáticamente en la primer referencia. Las variables de aplicación son de ámbito global y accesibles desde cualquier página de la aplicación web y sus valores son independientes de un usuario especifico.

Session State

Una sesión es el período de tiempo que un único navegador interactua con la aplicación web, cada vez que un nuevo navegador invoca una aplicación web, una nueva sesión es creada para el navegador. Cuando una nueva sesión es creada, ASP.NET mantiene información referente a la sesión en el **Session State**.

El **Session State** permite a los desarrolladores crear variables de sesión (*session variables*) que pueden establecerse y leerse a través de la duración de vida de la sesión. Las variables de sesión se componen de la palabra reservada `Session` seguido entre paréntesis (y comillas) del nombre de la variable, por ejemplo: `Session("User")`, las variables de sesión pueden ser accesadas desde cualquier página de la aplicación web por cada petición a la aplicación. Las sesiones y sus variables expiran después de veinte minutos de inactividad. Si el navegador hace una petición después del tiempo de expiración (20 minutos) la aplicación web crea una nueva sesión para el navegador.

Database Support

Los datos almacenados en las variables de aplicación y sesión podrían perderse, si la aplicación es interrumpida, para sobrevivir a estas interrrupciones, es estado de información debería ser almacenado en la base de datos.

Mantener un estado de información en la base de datos es también útil cuando la información a ser almacenada es significativa en tamaño, la dimensión del procesamiento y almacenamiento puede ser cargado en la capa de datos permitiendo mejorar el desempeño de la capa de aplicación.

Para una aplicación web profesional es muy común mantener un estado de información en al base de datos relacional por las siguientes razones:

- `Seguridad`, la información del usuario almacenada en la base de datos es un nivel extra eliminado de las capa de presentación, haciendo que los datos sean menos disponibles para uso indebido. Si la base de datos contiene información importante puede ser protegida a través de un usuario y contraseña para evitar accesos no deseados.
- `Consultas`, el almacenamiento de datos en la base de datos da a la aplicación el poder y funcionalidad de la base de datos en general, incluyendo la habilidad para consultar información específica.
- `Capacidad`, las bases de datos son especialmente buenas para mantener cantidades grandes de información y los servicios de datos pueden fraccionarse a la capa de datos que reside en uno o más servidores de datos, permitiendo a la aplicación web evitar la disminución del desempeño.
- `Extracción de Datos`, teniendo la capacidad de base de datos una aplicación, podría mantener referencias sobre cuantas veces el usuario ha visitado la aplicación web.

Sistemas Inteligentes

Los **Sistemas Inteligentes** se enfocan sobre el procesamiento y producción de conocimiento.

Desde la perspectiva de la computación los datos se refieren a números, caracteres o imagenes sin contexto, los datos por si solos no significan nada o carecen de significado, cuando los datos son procesados en un contexto se convierten en información, como la información es recopilada también puede ser procesada por patrones creando conocimiento. La inteligencia es la habilidad de adquirir conocimiento, finalmente la sabiduría es un comportamiento apropiado guiado por el conocimiento.

Un **Sistema Inteligente** extiende la tradicional función de la computación para también incluir la adquisición y aplicación de conocimiento.

Una estrategia para capturar y utilizar el conocimiento es a través de heurísticas, una **heurística** es una regla de conocimiento basada en la experiencia o que induce a buscar la solución.

Las heurísticas pueden ser utilizadas por los **Sistemas Inteligentes** para guíar el proceso de descubrimiento y a través de heurísticas-guía el comportamiento no es garantizado para ser optimo. Las heurísticas quizá sean de propósito general o dominio específico. Las heurísticas de propósito general a menudo suenan como el sentido común, es decir, en base a la experiencia que es la sabiduría. Las heurísticas de dominio específico suenan como reglas o estrategias que son problemas especificos, tipicamente es un comportamiento experto que incorpora conocimiento.

El uso del conocimiento heurístico para guiar el comportamiento es un tipo de sistema inteligente.

Una función heurística regresa un valor basado en un conocimiento heurístico, esta función puede ser utilizada para evaluar múltiples posibilidades y entonces seleccionar el mejor valor.

El **aprendizaje** involucra una modificación del comportamiento basado en la experiencia. Los sistemas que aprenden son comunmente clasificados como sistemas inteligentes.

Un sistema informático puede ser construido para que mejore a través del aprendizaje de prueba y error basado en el refuerzo positivo o negativo.