

# **VISUAL STUDIO.NET**

**Manual del  
Participante**

## Contenido del Curso

<b>MODULO 1: VISUAL STUDIO .NET .....</b>	<b>4</b>
INTRODUCCIÓN A VISUAL STUDIO .NET .....	4
Definiendo Visual Studio .NET .....	4
Herramienta Rápida de Desarrollo (RAD) .....	5
LENGUAJES .NET .....	6
Neutralidad de Lenguajes .NET .....	6
Lenguajes en .NET .....	7
ENTORNO INTEGRADO DE DESARROLLO (IDE) .....	8
Descripción del IDE Compartido .....	8
Administración de Ventanas .....	12
Diseñadores .....	12
Herramientas de Datos .....	13
<b>MODULO 2: VISUAL BASIC .NET .....</b>	<b>15</b>
INTRODUCCIÓN .....	15
CARACTERÍSTICAS DEL LENGUAJE .....	16
Tipos de Datos .....	16
Variables .....	17
Arrays .....	18
Procedimientos .....	20
MANEJO DE THREADS .....	21
Implementación de Threads .....	21
Estado de un Thread .....	22
DEPURACIÓN .....	23
Barras de Depuración .....	23
Ventanas de Depuración .....	26
CONTROL DE EXCEPCIONES .....	30
Tipos de Errores .....	30
Formas de Controlar Excepciones .....	31
Opciones de Control de Excepciones .....	33
LABORATORIO 3: .....	34
Ejercicio 1: Reconociendo VB .NET y Trabajando con el Lenguaje .....	34
Ejercicio 2: Implementando Multi Thread y Control de Excepciones .....	38
<b>MÓDULO 3: CREANDO APLICACIONES PARA WINDOWS .....</b>	<b>41</b>
USANDO WINDOWS FORMS .....	41
Introducción .....	41
Objeto Formulario .....	42
Uso del ToolBox .....	46
USANDO CONTROLES PARA WINDOWS FORMS .....	47
Controles Label, TextBox y Button .....	47
Controles GroupBox, RadioButton y CheckBox .....	51
Controles ListBox, CheckedListBox y ComboBox .....	55
INTERFACES .....	60
Introducción .....	60
Creando Aplicaciones MDI .....	60
Controles TreeView y ListView .....	61

AÑADIENDO MENÚS, DIÁLOGOS Y BARRAS .....	65
Menús.....	65
Diálogos .....	66
Barras.....	70
LABORATORIO 3:.....	72
Ejercicio 1: Usando Controles para Windows.....	72
Ejercicio 2: Creando aplicaciones MDI con Menús, Diálogos y Barras .....	79
<b>MÓDULO 4: CREANDO COMPONENTES .NET .....</b>	<b>87</b>
ELEMENTOS DE UNA CLASE (MIEMBROS) .....	88
Clase .....	89
Constantes, Campos y Enumeraciones .....	91
Propiedades .....	94
Métodos.....	96
Eventos .....	98
Constructores y destructores .....	100
CREANDO UNA LIBRERÍA DE CLASES .....	102
Eligiendo el Tipo de Aplicación.....	102
Añadiendo una Clase .....	103
Creando Propiedades, Métodos, Eventos.....	105
Probando y Usando la Librería de Clases.....	106
HERENCIA DE CLASES.....	109
Introducción a la Herencia de Clases.....	109
Implementando Herencia en una Clase .....	110
Sentencias para trabajar con Herencia .....	112
Polimorfismo.....	114
LABORATORIO 4:.....	115
Ejercicio 1: Creando y Usando una Librería de Clases.....	115
Ejercicio 2: Trabajando con Herencia de Clases.....	119

# Modulo 1

# Visual Studio .NET

---

## *Introducción a Visual Studio .NET*

### **Definiendo Visual Studio .NET**

Visual Studio .NET es la Herramienta Rápida de Desarrollo (RAD) de Microsoft para la siguiente generación de Internet que son los Servicios Web XML. Esta herramienta permite la creación de aplicaciones usando el Marco .NET, es decir usando el CLR, la Librería de Clases, ADO .NET, ASP .NET, etc.

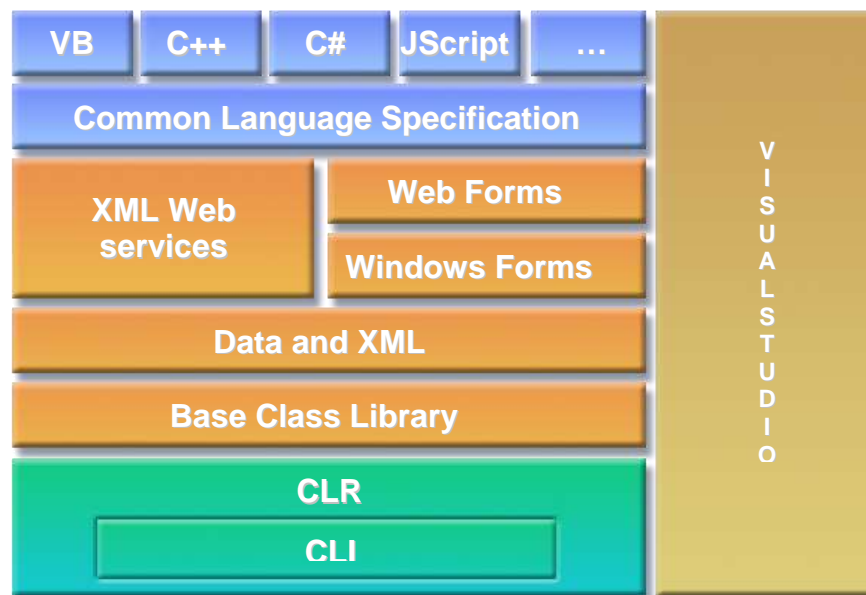
Es un software que brinda las herramientas necesarias para crear, distribuir, administrar y dar mantenimiento a aplicaciones Web distribuidas que usan Servicios Web XML, todo esto con una gran facilidad, rapidez y bajo costo.

Se puede crear aplicaciones Web directamente usando el Framework .NET y algún programa editor, por ejemplo el Bloc de Notas, pero el tiempo que llevaría el desarrollo no justificaría el ahorro de costos, en cambio, si se utiliza una herramienta como Visual Studio .NET el tiempo de desarrollo se reduciría enormemente.

Visual Studio .NET permite también la integración y el uso cruzado de lenguajes de programación: Visual Basic .NET, Visual C# .NET, Visual C++ .NET y JScript .NET

A diferencia de la versión anterior no existe Visual Interdev, ni Visual J++, además Visual Foxpro .NET no comparte las características unificadas del Marco .NET

**Figura 1.1: Estructura del Marco .NET y Visual Studio**



### Herramienta Rápida de Desarrollo (RAD)

La principal ventaja de Visual Studio .NET es realizar la creación de aplicaciones de forma fácil y rápida, tan solo con arrastrar y soltar objetos se pueden crear desde aplicaciones Windows hasta Servicios Web XML.

Entre algunas de las ventajas del soporte RAD de Visual Studio tenemos:

➤ **Creación de Páginas Web mediante Formularios Web**

Visual Studio .NET incluye un diseñador de páginas Web HTML y ASP .NET basado en formularios Web, el diseñador permite arrastrar controles, clases de datos, y otros objetos y configurar sus propiedades como si fuese un formulario de una aplicación para Windows.

➤ **Creación de Servicios Web XML**

Para crear Servicios Web XML, Visual Studio .NET incluye una plantilla con Servicios Web de ejemplo, los cuales puedes modificar y personalizar a tu medida, eligiendo el lenguaje que desees, que puede ser Visual Basic .NET, Visual C# .NET o Visual C++ .NET

➤ **Acceso a Servicios Web XML**

Una vez creado los Servicios Web XML deben usarse en otras aplicaciones del negocio, para ello Visual Studio .NET cuenta con el Explorador de Servidores (Server Explorer) que permite ver los Servicios Web publicados y usarlos con solo un arrastre. También podemos usar un Servicio Web

haciendo referencia desde un proyecto mediante la opción "Add Web Referente" del menú "Project".

### ➤ **Creación de Componentes .NET**

Crear componentes o controles de usuario es tan simple como crear un formulario, ya que usando la herencia se puede pasar todas las características de un objeto a otro, esto esta presente en todos los objetos creados en Visual Studio .NET, sean visuales o no.

### ➤ **Creación de archivos XML**

Con el diseñador de XML, crear un archivo XML es mas fácil que nunca, ya que se muestra de colores el código y se auto completan los Tags que uno va escribiendo. Este maneja 3 vistas: XML, esquemas y datos.

Existen mas características RAD en Visual Studio .NET, las cuales trataremos mas adelante.

## ***Lenguajes .NET***

### **Neutralidad de Lenguajes .NET**

El Marco .NET es neutral con respecto al lenguaje y admite prácticamente cualquiera de ellos. Esto trae consigo los siguientes beneficios para el desarrollador:

### ➤ **Código reusable y compartido**

Antes no existía una integración total del equipo de desarrollo cuando cada grupo usaba herramientas diferentes como Visual Basic 6, Visual C++ 6 o Visual J++ 6, en cambio ahora, el código escrito en cualquier lenguaje puede ser usado desde otro, ya que todas son clases .NET.

### ➤ **Acceso a APIs igual para todos los lenguajes**

Actualmente, todos los lenguajes del Marco .NET comparten las mismas clases o APIS del sistema, antes cada lenguaje accedía a las APIs de su manera, de ellos C++ era el mas fuerte, hoy en día con .NET no existen diferencias entre potencia del lenguaje.

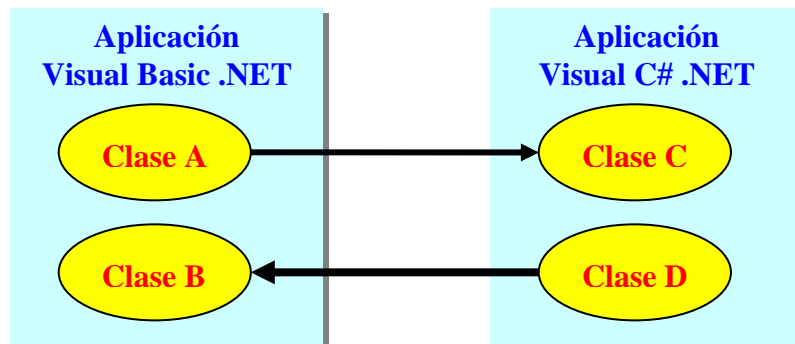
### ➤ **Herencia cruzada entre lenguajes**

Se puede crear una clase en un lenguaje y heredarse desde otra clase escrita en diferente lenguaje .NET, lo que permite la reutilización total del código por parte de diferentes desarrolladores.

### ➤ **Manejo de errores cruzado entre lenguajes**

Se puede controlar errores desde una clase por mas que el error ocurra en un objeto creado en otro lenguaje distinto al de la clase que controla el error, también se puede realizar el seguimiento de una aplicación aunque incluya llamadas a otros lenguajes, etc.

**Figura 1.2: Relación cruzada entre Lenguajes .NET**



### Lenguajes en .NET

En Visual Studio .NET vienen los siguientes Lenguajes de Programación:

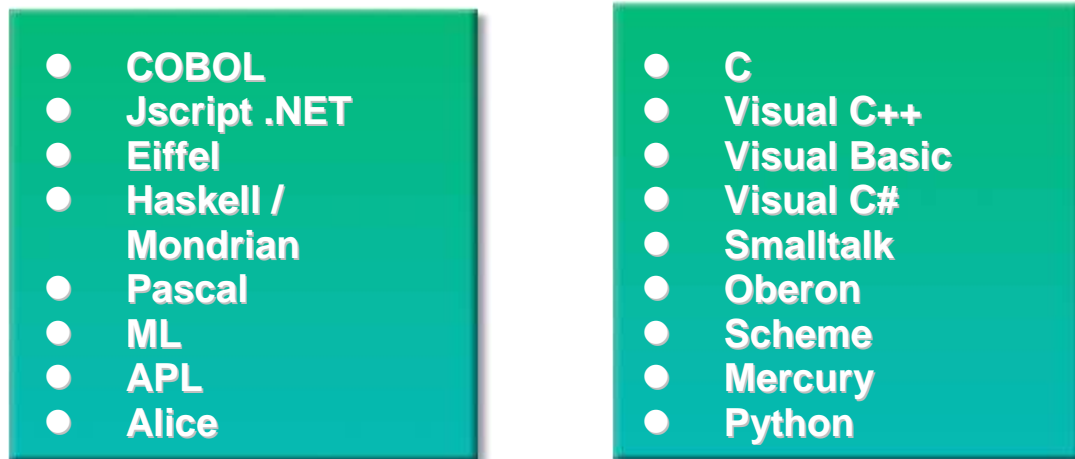
- Visual Basic .NET
- Visual C# .NET
- Visual C++ .NET
- Visual Foxpro .NET (No administrado por el Marco .NET)
- Visual JScript .NET

Además de estos lenguajes, el Marco .NET soporta otros lenguajes, entre los cuales destacan:

- COBOL: <http://www.adtools.com/info/whitepaper/net.html/>
- Pascal: <http://www2.fit.qut.edu.au/CompSci/PLAS//ComponentPascal/>
- SmallTalk: <http://www.qks.com/>
- Eiffel: <http://dotnet.eiffel.com/>
- ML: <http://research.microsoft.com/Projects/SML.NET/index.htm>
- APL: <http://www.dyadic.com/>
- Oberon: <http://www.oberon.ethz.ch/lightning/>
- Scheme: <http://rover.cs.nwu.edu/~scheme/>
- Mercury: <http://www.cs.mu.oz.au/research/mercury/>
- Python: <http://aspn.activestate.com/ASP.NET/index>
- Haskell: <http://haskell.cs.yale.edu/ghc/>
- Mondrian: <http://www.mondrian-script.org>

Se ha dado el nombre de algunos lenguajes junto con sus respectivas páginas Web donde se puede encontrar información sobre estos e inclusive bajarse el compilador del lenguaje compatible con .NET.

**Figura 1.3: Listado de Lenguajes .NET**



### ***Entorno Integrado de Desarrollo (IDE)***

#### **Descripción del IDE Compartido**

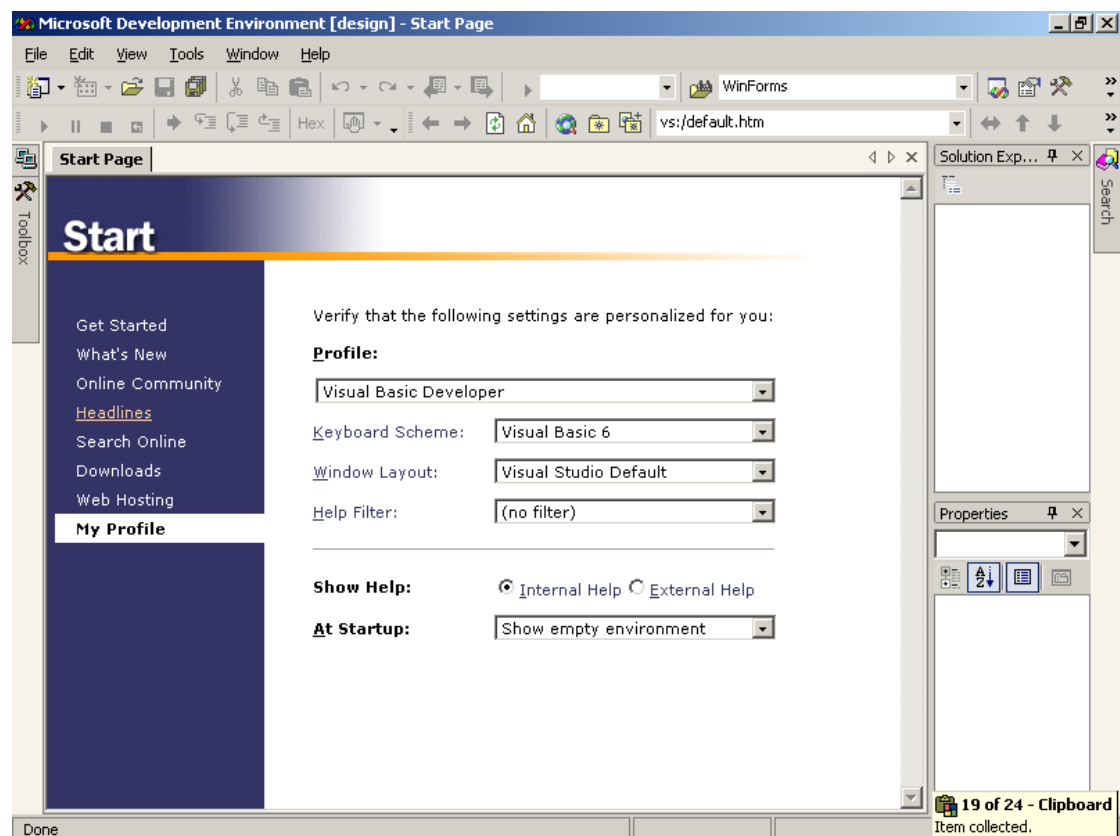
Visual Studio .NET tiene un Entorno Integrado de Desarrollo único o compartido para crear aplicaciones usando cualquiera de los Lenguajes de Programación, que pueden ser Visual Basic, Visual C++ o C#.

En esta nueva versión de Visual Studio, Visual Foxpro mantiene su propio IDE (similar al de la Versión 6), además Visual Interdev ya no es parte de Visual Studio, ya que las herramientas de desarrollo para Web están disponibles a través de los Web Forms disponibles desde el IDE común.

Al iniciar Visual Studio .NET aparece (por defecto) la Página de Inicio, tal como se muestra en la Figura 1.4

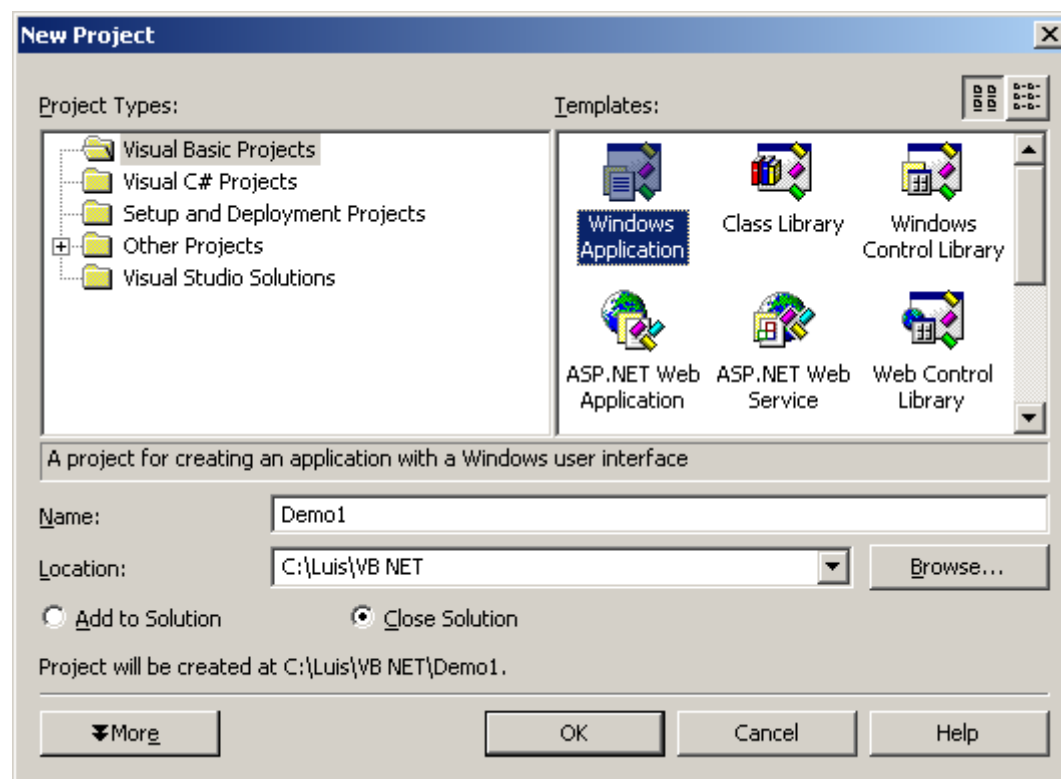
***Figura 1.4: Ventana de Inicio del Visual Studio .NET***





Desde esta página de inicio podemos elegir la opción “Get Started” para crear un nuevo proyecto o abrir uno existente o reportar un error del IDE de Visual Studio, si elegimos “New Project” se presentará la Ventana que se muestra en la Figura 2.5

**Figura 1.5: Ventana de Crear un Nuevo Proyecto**

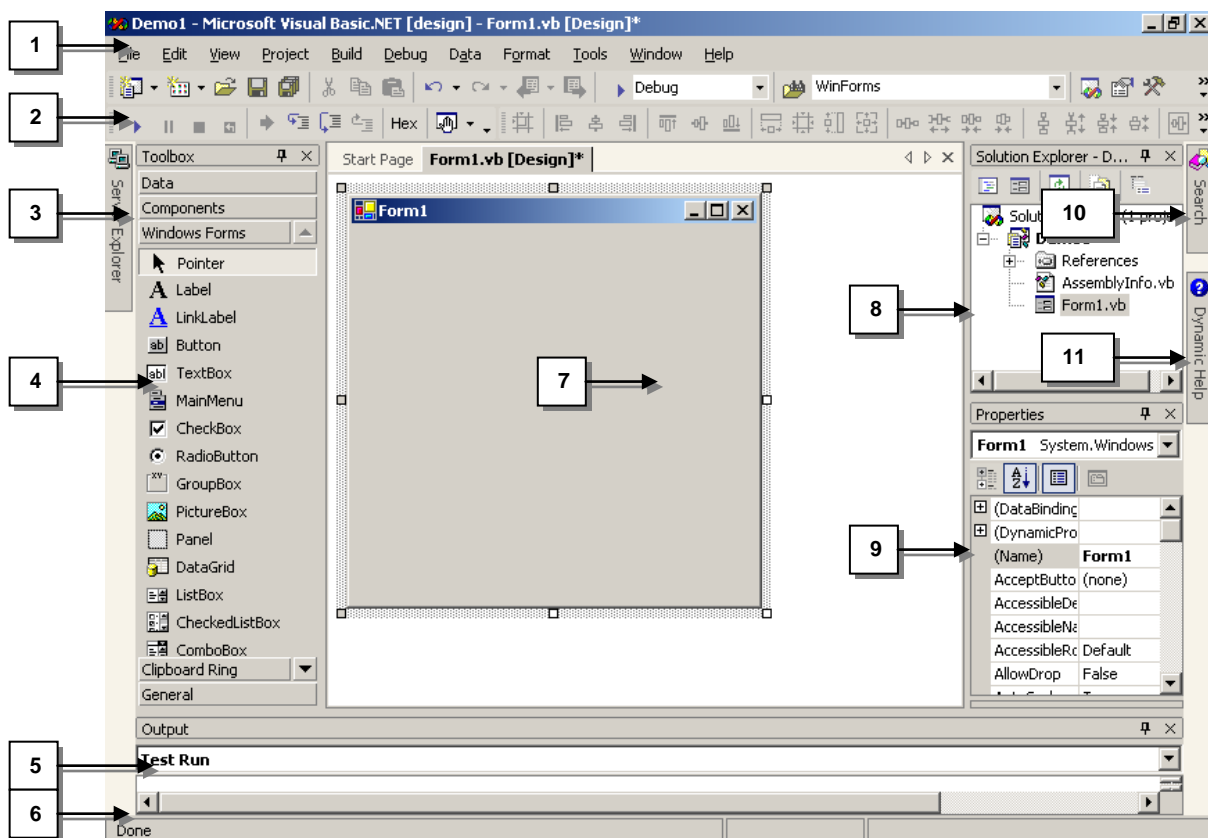


Esta ventana está dividida 2 secciones: en el lado izquierdo se encuentran los tipos de proyectos que se pueden realizar (Visual Basic, Visual C#, Visual C++, etc) y en el lado derecho se encuentran las plantillas o tipos de aplicaciones, que varían de acuerdo al tipo de proyecto.

Si se elige Visual Basic o Visual C#, las plantillas se pueden dividir en tres: para Windows, para Web (Aplicaciones, Servicios, Librería de Clases, Librería de Controles, Proyecto Vacío) y de Consola.

En el caso de elegir como tipo de proyecto "Visual Basic" y como plantilla "Windows Application" hay que seleccionar la ubicación del nuevo proyecto y escribir el nombre de este, el cual creará una carpeta con el mismo nombre en el lugar seleccionado. A continuación la figura 1.6 muestra el IDE compartido de Visual Studio .NET en el caso de elegir una Aplicación para Windows.

Figura 1.6: IDE Compartido de Visual Studio .NET



Entre las partes del nuevo IDE de Visual Studio .NET tenemos:

1. Menu Bar
2. ToolBars
3. Server Explorer Window (Ctrl + Alt + S)
4. Toolbox (Ctrl + Alt + X)
5. Output Window (Ctrl + Alt + O)
6. Status Bar
7. Windows Form Designer
8. Solution Explorer Window (Ctrl + R)
9. Properties Window (F4)
10. Search Window (Ctrl + Alt + F3)
11. Dynamic Help Window (Ctrl + F1)

Existen nuevas ventanas en Visual Studio .NET entre las cuales tenemos:

- Class View (Ctrl + Shift + C)
- Resource View (Ctrl + Shift + E)
- Macro Explorer (Alt + F8)
- Document Outline (Ctrl Alt + T)
- Task List (Ctrl + Alt + K)
- Command Window (Ctrl + Alt + A)
- Find Symbol Results (Ctrl +Alt + Y)

**Nota:** Para mostrar u ocultar cualquier ventana del IDE elegir el menú “View”

### Administración de Ventanas

El manejo de ventanas en Visual Studio .NET es más simple y rápido pudiendo acceder a cualquier elemento de manera fácil, debido a las nuevas características de Administración de ventanas, tales como:

- **Auto Ocultar:** Esta característica es nueva en Visual Studio .NET y permite ocultar una ventana permitiendo liberar espacio en el IDE, para mostrar nuevamente la ventana solo hay que ubicar el mouse cerca del nombre de la ventana que aparece en una ficha.
- **Ventanas Acoplables:** Al igual que Visual Basic 6, esta nueva versión permite acoplar ventanas las cuales estarán fijas en el IDE. Podemos elegir si una ventana se va a “Auto Ocultar” o si se va a “Acoplar”. Al acoplar la ventana tendremos la posibilidad de ver siempre su contenido.
- **Fichas de Documentos:** En la versión anterior de Visual Studio el trabajo con varios documentos era tedioso por que para acceder a un documento abierto (por ejemplo un módulo de formulario) había que hacerlo mediante el menú “Window” o dando clic en el botón “View Code” o doble clic sobre el nombre del objeto. Ahora el acceso es muy rápido a través de las fichas que hay en la parte superior del Editor.
- **Navegación a través del IDE:** Podemos navegar a través de los documentos visitados usando la barra Web, pudiendo ir hacia “Atrás”, “Adelante”, “Detener”, “Actualizar”, “Ir al inicio” como si se tratase de un Browser y si navegáramos a través de páginas Web, lo que facilita la búsqueda de una pagina ya abierta.
- **Ventana de Ayuda Rápida:** Una de las características mas importantes de Visual Studio .NET es la “ayuda inteligente” o “ayuda rápida” que permite mostrar en una ventana todos los tópicos relacionados a donde se encuentre el cursor (si esta en el editor) o al objeto seleccionado (si estamos en el diseñador de formulario), por ejemplo, si estamos en el editor escribiendo una función aparecerán los tópicos relacionados a ésta, si nos encontramos seleccionando un control, aparecerán los temas referentes a éste.

Todas estas nuevas características hacen que el trabajo del desarrollador sea más productivo, centrándose en la lógica de la aplicación y no en el mantenimiento de ésta ya que es más fácil al utilizar las nuevas características de Administración de Ventanas, anteriormente comentadas.

### Diseñadores

Para realizar la construcción de aplicaciones o creación de componentes o servicios disponemos de Diseñadores que facilitan la labor de construcción de interfaces, creación de sentencias, etc.

La mayoría de diseñadores se habilitan al elegir una plantilla de Visual Studio .NET y casi todos generan código al diseñar controles sobre el contenedor respectivo; característica totalmente distinta a la forma de trabajo en Visual Basic 6, que ocultaba el código generado por el diseñador.

Entre los diseñadores que trae Visual Studio .NET tenemos:

- **Windows Form Designer:** Se muestra al elegir cualquiera de dos plantillas: “Windows Application” o “Windows Control Library”, habilitando en el Toolbox los controles para Windows que serán usados para construir la interfase de la aplicación arrastrando dichos controles hacia el formulario o control de usuario.
- **Web Form Designer:** Se muestra al elegir la plantilla “Web Application” habilitando en el Toolbox los controles para Web y los controles HTML que serán usados para construir la página Web que correrá en el servidor IIS (archivo aspx) arrastrando dichos controles hacia el formulario Web.
- **Component Designer:** Este diseñador se muestra al elegir una de dos plantillas: “Class Library” o “Windows Service” y también trabaja con los controles para windows, creando una interfase reusable desde otra aplicación.
- **Web Service Designer:** Sirve para diseñar servicios Web y es mostrado al elegir una plantilla “Web Service”, también trabaja con los controles para Windows, componentes, etc.

Existen más diseñadores, pero que lo trataremos en la categoría de herramientas de datos, debido al tipo de trabajo que realizan con datos, el cual se trata como tema siguiente.

### Herramientas de Datos

Si se quiere realizar un trabajo rápido con datos, tal como modificar la estructura de la Base de Datos, crear tablas, consultas, procedimientos almacenados, etc., existen herramientas que permiten realizar esta labor reduciendo enormemente el proceso de desarrollo en el caso de hacerse por otros medios.

Entre las principales herramientas que trabajan con datos tenemos:

- **Server Explorer:** Sin duda es una de las principales herramientas de Visual Studio .NET y no solo para acceder a datos sino para mostrar y administrar los diferentes servidores o recursos del sistema, tales como Base de Datos, Servicios Web, Aplicaciones COM+, etc. Con solo arrastrar el objeto éste puede ser usado en una aplicación. También se tratará con mayor detalle en el módulo de acceso a datos.
- **DataAdapter Wizard:** Es un Asistente que permite crear un DataAdapter que es un Comando (Select, Insert, Update, Delete) con el cual se podrá generar un conjunto de registros o Dataset. La misma función puede ser cubierta por el Server Explorer con solo arrastrar los campos hacia el formulario.
- **Query Designer:** Es un Diseñador que permite crear Consultas SQL de manera sencilla arrastrando tablas o consultas sobre éste y eligiendo los campos que se verán en la consulta de datos, también se puede realizar filtros o especificar criterios de selección. Además no solo se pueden construir consultas Select sino también se pueden crear consultas Insert, Update o Delete, etc.
- **DataBase Project:** Es un tipo de plantilla que sirve para trabajar con una Base de Datos, para lo cual debe existir una conexión a un origen de datos, este tipo de proyecto da la posibilidad de crear

y modificar scripts de creación de tablas, consultas, vistas, desencadenantes, procedimientos almacenados, etc.

- **Editor de Script:** Uno de las principales herramientas para trabajar con base de datos remotas como SQL Server, Oracle, etc, es utilizar el editor de Scripts que permite crear tablas, consultas, vistas, etc. Mostrando con colores las sentencias o palabras reservadas del lenguaje Transact-SQL.
- **Depurador de Procedimientos Almacenados:** Visual Studio .NET incorpora un Depurador de Store Procedure que puede realizar seguimientos paso a paso por línea de código, por sentencia o por instrucción, además crea puntos de interrupción, permitiendo un mayor control y seguimiento del código en caso de errores.

Todas estas herramientas mencionadas, son nuevas en Visual Studio .NET, ha excepción del Query Builder que es el mismo de la versión anterior de Visual Studio.

Como se habrá dado cuenta muchas herramientas de acceso a datos de Visual Basic 6 han sido eliminadas, tales como: Data Environment, Data View, Data Report, y otras más, pero en su reemplazo existen las que ya hemos mencionado.

# Modulo 2

# Visual Basic .NET

---

## *Introducción*

En los módulos anteriores hemos tratado el Marco .NET y Visual Studio .NET, ahora trataremos Visual Basic .NET, pero hay que remarcar que las características del lenguaje dependen del Marco .NET y las herramientas son compartidas por el IDE de Visual Studio .NET

Visual Basic .NET es la versión 7 de ésta popular herramienta, ésta última versión tiene cambios radicales, tanto en su concepción (.NET), en el lenguaje, las herramientas usadas, etc. Entre las nuevas características de Visual Basic .NET tenemos:

- Dos tipos de desarrollos bien diferenciados:
  - ✓ Aplicaciones para Windows
  - ✓ Aplicaciones para Internet
- Acceso a Datos usando ADO .NET, el cual permite trabajar con DataSets desconectados
- Nuevo Depurador que permite realizar seguimiento de código escrito en diferentes lenguajes .NET
- Creación y uso de XML para intercambio de datos entre aplicaciones

- Lenguaje Orientado a Objetos, con soporte de Herencia múltiple, y Polimorfismo a través de la sobrecarga de propiedades, métodos y funciones con el mismo nombre
- Control de errores o excepciones en forma estructurada (Try..Catch..Finally)
- Soporte de multithread para que la aplicación pueda ejecutar múltiples tareas en forma independiente.
- Uso de NameSpaces para referirse a una clase que se va a usar en la aplicación. Los Assemblies reemplazan a la Librería de Tipos, en un Assemblies pueden existir uno o más NameSpaces
- Reestructuración en los Tipos de Datos; existen nuevos tipos de datos y se han modificado y eliminado ciertos tipos de datos.
- Cambio en el Lenguaje: nuevas forma de declarar variables, conversión explícita de tipos de datos (no existe conversión forzosa), no existen procedimientos sino funciones, etc.

### Características del Lenguaje

#### Tipos de Datos

Tipo V. Basic	Estructura Tipo .NET Runtime	Tamaño Almac.	Rango de Valores
Boolean	System.Boolean	4 bytes	True o False
Byte	System.Byte	1 byte	0 to 255 (sin signo)
Char	System.Char	2 bytes	0 to 65535 (sin signo)
Date	System.DateTime	8 bytes	Enero 1, 1 CE hasta Diciembre 31, 9999
Decimal	System.Decimal	12 bytes	+/-79,228,162,514,264,337,593,543,950,335 sin punto decimal; +/-7.9228162514264337593543950335 con 28 posiciones a la derecha del decimal; número mas corto (no 0) es +/-0.00000000000000000000000000000001
Double (doble-precisión punto-flotante)	System.Double	8 bytes	-1.79769313486231E308 hasta -4.94065645841247E-324 para valores negativos; 4.94065645841247E-324 hasta 1.79769313486232E308 para valores positivos
Integer	System.Int32	4 bytes	-2,147,483,648 to 2,147,483,647
Long (Entero largo)	System.Int64	8 bytes	-9,223,372,036,854,775,808 hasta 9,223,372,036,854,775,807
Object	System.Object (class)	4 bytes	Cualquier tipo de dato
Short	System.Int16	2 bytes	-32,768 to 32,767
Single (simple precisión punto-flotante)	System.Single	4 bytes	-3.402823E38 hasta -1.401298E-45 para valores negativos; 1.401298E-45 hasta 3.402823E38 para valores positivos
String (tamaño-	System.String (class)	10 bytes + (20 hasta aproximadamente 2 billones de caracteres * tamaño Unicode	



variable) User-Defined Type (estructura)	(heredado desde System.ValueType)	cadena) Suma de tamaños de sus miembros	Cada miembro de la estructura tiene un rango determinado, es decir pueden tener sus propios tipos de datos distintos unos de otros
---	--------------------------------------	--	--

**Notas:** Se ha eliminado el tipo de dato Variant y es reemplazado por Object, también el dato Currency ahora es Decimal y el Type ahora es Structure. Además no existen String de tamaño fijo, sino que todos son dinámicos.

## Variables

Una variable es un dato temporal en memoria que tiene un nombre, un tipo de dato, un tiempo de vida y un alcance, los cuales lo dan la forma como se declare ésta.

Una variable debe cumplir con las siguientes reglas:

- Debe iniciar con un carácter alfabético.
- Debería contener solo caracteres alfabéticos, dígitos y carácter de subrayado.
- El nombre no debe exceder a 255 caracteres, etc.

## Declaración de Variables

A diferencia de Visual Basic 6, en VB .NET se pueden declarar varias variables en una sola instrucción y además se puede asignar directamente sus valores. Otra observación es que es necesario definir el tipo de declaración y el tipo de dato (antes si no se hacia se asumía un tipo de declaración y un tipo de dato variant, que ahora no existe).

**Sintaxis:** <Tipo de Declaración> <Variable(s)> As <Tipo de Dato> [= <Valor>]

Existen varios tipos de declaración que detallamos a continuación en la siguiente tabla:

Declaración	Lugar de Declaración	Alcance o Ámbito
Public	Módulo o Clase	Global, en todo el proyecto.
Protected	Clase	En la clase declarada o en una derivada.
Friend	Clase	En el Assemblies.
Private	Módulo	Solo en el módulo.
Dim	Procedimiento	Solo en el Procedimiento.
Static	Procedimiento	Solo en el Procedimiento.

## Alcance de Variables

Para las variables declaradas a nivel de procedimiento (Dim y Static) existe un nuevo alcance que es a nivel de estructura o bloque, que puede ser For – Next, If – End If, Do – Loop, etc. Las variables definidas dentro de un bloque solo valdrán en este bloque.

## Opciones de Trabajo con Variables

Por defecto en VB NET es necesario declarar las variables usadas (**Option Explicit**) y también es necesario que se asigne el mismo tipo de dato a la variable (**Option Strict**), si deseamos Declaración

implícita (por defecto Object) y Conversión Forzosa de tipos (ForeCast), aunque no es lo recomendable por performance, podemos conseguirlo de dos formas: mediante "Propiedades" del proyecto, opción "Build" y elegir "Off" en las listas de "Option Explicit" y Option Strict" o mediante declaración al inicio de todo el código:

Option Explicit Off  
Option Strict Off

## Arrays

Un array o arreglo es un conjunto de variables que tienen el mismo nombre pero diferente índice que permite simplificar el uso de éstas y aumentar la velocidad de acceso a los datos que almacena el array.

El array puede tener uno o más dimensiones (hasta 60) y cada dimensión tiene un límite inferior que siempre es 0 y un límite superior que es equivalente al tamaño de la dimensión del array menos 1. Esta característica es distinta que en la versión anterior, en donde el límite inferior del array podía empezar en cualquier número.

La clase base .NET de donde se heredan los array es "Array" y pertenece al siguiente Namespace: **System.Array**.

### *Declaración de Arrays*

A diferencia de Visual Basic 6, en VB .NET se pueden declarar arrays definiendo el número de dimensiones pero sin indicar el tamaño. Cabe resaltar que solo se puede declarar e inicializar un array que no tiene tamaño definido.

Otra diferencia es que no existe la sentencia Option Base que haga que el límite inferior del array empiece en 0 o en 1, éste siempre empezará en 0 e irá hasta n-1.

#### **Sintaxis:**

<Tipo de Declaración> <Array>([Tamaño]) As <Tipo de Dato>[=<Valores>]

#### **Ejemplos:**

```
Dim Alumnos(30),Cursos(10) As String  
Dim Edades() As Byte={18,20,25,27}  
Dim Sueldos( , ) As Decimal
```

### *Redimensionando Arrays*

Una vez definido la dimensión de un array, éste puede modificarse usando la sentencia ReDim, siempre y cuando el array haya sido declarado como dinámico (con Dim).

#### **Sintaxis:**

ReDim [Preserve] <Array>([Tamaño]) As <Tipo de Dato>[=<Valores>]

#### **Ejemplo:**

```
Dim I, Arreglo() As Integer
Redim Arreglo(5)
For I = 0 To Ubound(Arreglo)
    Arreglo(I) = I
Next I
```

## Procedimientos

Un Procedimiento es un bloque de código o conjunto de instrucciones que es definido en la aplicación y que puede ser usado varias veces mediante una llamada.

Dos características nuevas de los procedimientos, incorporadas en esta versión son:

- **Recursividad:** Es la capacidad del procedimiento para llamarse así mismo.
- **Sobrecarga:** Consiste en que varios procedimientos pueden tener el mismo nombre.

En Visual Basic tenemos varios Tipos de Procedimientos:

- **Subrutinas:** Ejecutan una acción sin retornar un valor.
- **Funciones:** Ejecutan una acción retornando un valor.
- **De Eventos:** Se desencadenan con la interacción del usuario o ante algún evento.
- **De Propiedades:** Devuelven y asignan valores a propiedades de un objeto.

### ***Declaración de un Procedimiento***

#### **Subrutina:**

```
[Public | Private | Friend] Sub <Nombre>([Optional] [ByVal | ByRef] <Par> As <Tipo>)  
    <Sentencia>  
    [Exit Sub]  
End Sub
```

#### **Función:**

```
[Public | Private | Friend] Function <Nombre>(<Parámetros>) As <Tipo>  
    <Sentencia>  
    [Exit Function]  
    [<Nombre>=<Valor> | Return(Valor)]  
End Function
```

#### **Notas:**

- ✓ El tipo de argumento por defecto es ByVal (en la versión anterior era ByRef)
- ✓ Si se usa Optional debe inicializarse con un valor (antes no era obligatorio)
- ✓ Se puede usar Return para regresar a la llamada del procedimiento.

### **Llamada a un Procedimiento**

Antes existía una forma de llamar a una subrutina y dos formas de llamar funciones (o como subrutina o como función, ésta última mediante paréntesis). En cambio ahora, existe una sola forma de llamar procedimientos, sea subrutinas o funciones, que es escribiendo el nombre seguido de paréntesis y entre éstos los parámetros (si los hay).

#### **Sintaxis:**

```
[Variable]=<Nombre de la Sub o Function>([Parámetro(s)])
```

## **Manejo de Threads**

### **Threads**

Un thread es la unidad básica para que el Sistema Operativo pueda ejecutar un proceso. Una aplicación (AppDomain) siempre inicia un solo thread, pero este a su vez puede iniciar otros thread. Al proceso de ejecutar varios Thread le llamaremos Threading.

La ventaja principal de los Threads es tener varias actividades ocurriendo en forma simultánea, lo cual es una gran posibilidad para que los desarrolladores puedan trabajar con varios procesos sin perjudicar otras tareas. Por ejemplo, el usuario puede interactuar con la aplicación mientras se va ejecutando una consulta de miles de registros.

Se recomienda el uso de Threads en las siguientes situaciones:

- Para comunicaciones sobre una red, servidor Web o Servidor de Base de Datos.
- Al ejecutar operaciones que demoren bastante tiempo.
- Para mantener siempre disponible la comunicación entre el usuario y la interface, mientras se van ejecutando tareas en segundo plano, etc.

El uso de Threads intensivamente disminuye los recursos del sistema operativo, por tanto solo se recomienda usar en los casos ya descritos, sino la performance de la aplicación disminuirá.

### **Implementación de Threads**

Para implementar Threads se usa el NameSpace: “**System.Threading.Thread**” y luego se hace uso de los métodos que se definen a continuación:

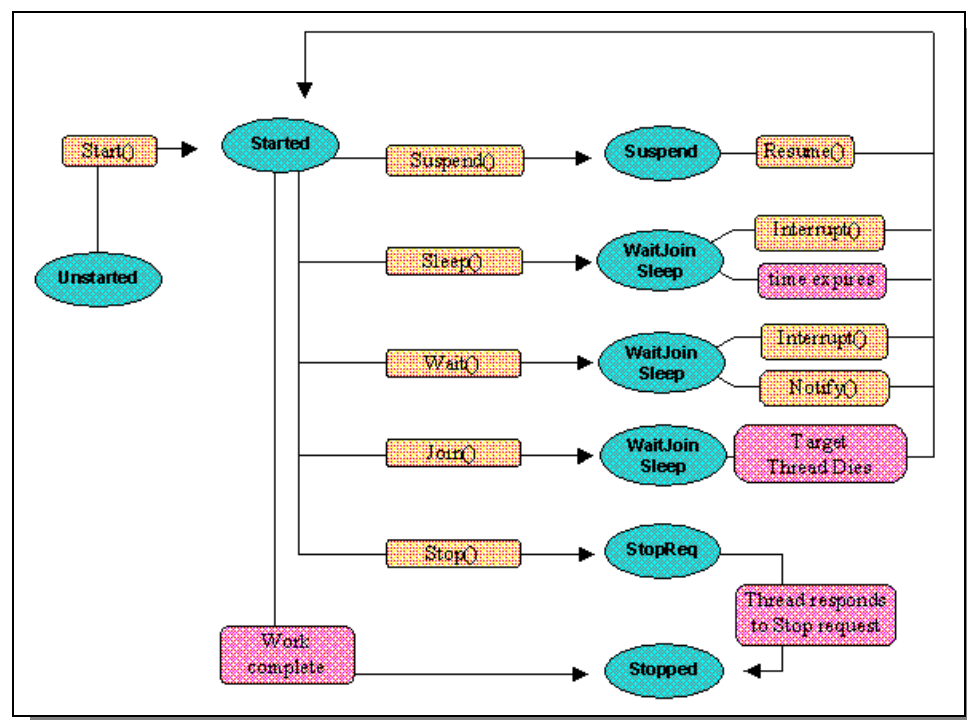
- **Start**: Inicia un thread, el cual es un proceso de llamada asíncrona. Para saber el estado del Thread hay que usar las propiedades ThreadState y IsAlive.
- **Abort**: Cancela un thread iniciado, si deseamos saber el estado nuevamente podemos usar las propiedades ThreadState y IsAlive.
- **Sleep**: Ocasiona una pausa en milisegundos del bloque de instrucciones.
- **Suspend**: También ocasiona una pausa en el bloque de instrucciones.
- **Resume**: Reinicia una pausa originada con el método Suspend.
- **Interrupt**: Interrumpe una pausa originando una excepción.
- **Join**: Espera por un thread.

## Estado de un Thread

Un thread puede tener diferentes estados en un mismo tiempo, para saber su estado se encuentra la propiedad ThreadState que devuelve un valor que indica el estado actual del thread.

Acción	Estado de Transición
Otro thread llama a Thread.Start	Unchanged
El thread inicia su ejecución	Running
El thread llama a Thread.Sleep	WaitSleepJoin
El thread llama a Monitor. Espera en otro objeto	WaitSleepJoin
El thread llama a Thread.Join en otro thread	WaitSleepJoin
Otro thread llama a Thread.Suspend	SuspendRequested
El thread responde a un requerimiento de Thread.Suspend	Suspended
Otro thread llama a Thread.Resume	Running
Otro thread llama a Thread.Interrupt	Running
Otro thread llama a Thread.Abort	AbortRequested
El thread responde a Thread.Abort	Aborted

Figura 2.1: Diagrama de Estado de un Thread



## Depuración

La Depuración es el proceso de realizar un seguimiento a una aplicación para analizar variables, expresiones, objetos, etc. y probar sus valores en diferentes escenarios, así como probar el desempeño de la aplicación.

En Visual Studio .NET, existe un mismo depurador para Visual Basic .NET y C# (código administrado), el cual tiene las siguientes mejoras:

- **Depurar a través de Diferentes Lenguajes:** Se puede depurar aplicaciones escritas en diferentes lenguajes que son parte de una misma solución, por ejemplo una aplicación cliente para la interface de usuario escrita en Visual Basic o C# y una aplicación servidor escrita en Visual C++.
- **Adjuntar Programas en Ejecución:** Es posible adjuntar un programa que se está ejecutando al depurador, y depurar el programa como si estuviera en el IDE de Visual Studio. Esto se realiza a través de la Ventana de Procesos, que muestra todos los procesos que están ejecutándose mientras se corre una aplicación.
- **Depuración Remota:** Se puede añadir y depurar un proceso que está ejecutándose en una computadora remota, por ejemplo podemos depurar una aplicación cliente Windows que llama a un Web Service que está corriendo en otra máquina, pudiendo depurar a este Web Service como si estuviera en la máquina donde está corriendo la aplicación cliente.
- **Depuración de Aplicaciones Multi Thread:** Una nueva característica de Visual Basic es el soporte de aplicaciones multi thread, para lo cual se dispone de la Ventana de Threads en donde se muestra los threads que se encuentran en ejecución.
- **Depuración de Aplicaciones Web:** Esta característica ha sido mejorada, permitiendo adjuntar una página ASP .NET al proceso que se encuentra en ejecución, que puede ser una aplicación Web y realizar el proceso de depuración de la página como si estuviera en la computadora local, etc.

Para realizar la Depuración se dispone de dos tipos de herramientas, que son:

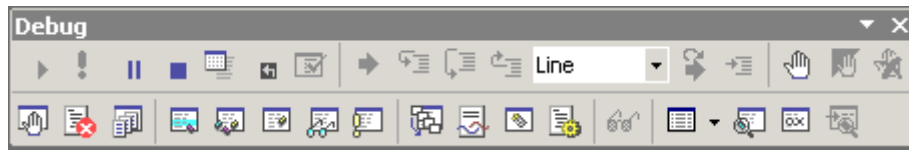
- **Barras de Depuración:** Contienen los comandos para realizar la depuración, como el seguimiento paso a paso, fijar puntos de interrupción, mostrar las ventanas de depuración, etc.
- **Ventanas de Depuración:** Son ventanas donde se muestra el estado en que se encuentran las variables, expresiones, procedimientos, objetos, etc. Algunas permiten el análisis o inspección y otras la visualización o modificación del estado de objetos.


















## Barras de Depuración

Existen dos barras de depuración que a continuación se describen:

- **Barra de Depuración:** Es la principal barra que contiene todos los comandos de depuración (34 en total), desde ejecutar una aplicación hasta fijar desensamblar.

**Figura 2.2: Barra de Depuración**



-  Inicia la ejecución de la aplicación.
-  Ejecuta la aplicación sin entrar en depuración.
-  Interrumpe la ejecución e ingresa al modo pausa.
-  Finaliza la ejecución de la aplicación.
-  Quita todas las aplicaciones anexadas.
-  Reinicia nuevamente la ejecución de la aplicación.
-  Aplica los cambios realizados al código si es que estamos en modo pausa.
-  Muestra la siguiente sentencia a depurarse.
-  Ejecuta paso a paso incluyendo procedimientos.
-  Ejecuta paso a paso sin incluir procedimientos.
-  Retrocede al paso anterior en un seguimiento paso a paso.
-  Line Indica el tipo de depuración paso a paso; puede ser por línea (por defecto), por sentencia o por instrucción.
-  Pasa a la siguiente sentencia a depurarse.
-  Ejecuta la sentencia o línea especificada por el Cursor del mouse en una ventana.
-  Inserta un punto de interrupción donde se detendrá la ejecución.
-  Habilita o deshabilita un punto de interrupción previamente insertado.
-  Borra o elimina todos los puntos de interrupción fijados.





Visualiza la ventana de Breakpoints.



Presenta la ventana de Excepciones para controlar errores.



Muestra la ventana de documentos ejecutándose.



Visualiza la ventana Autos.



Presenta la ventana Local.



Muestra la Ventana This que contiene la clase actual.



Activa la ventana Watch conteniendo las expresiones de análisis.



Visualiza la ventana Immediate.



Presenta la ventana Call Stack o de llamada a la pila.



Muestra la ventana de Threads.



Activa la ventana de Módulos.



Visualiza la ventana de Procesos en ejecución.



Presenta la ventana QuickWatch o de Análisis Rápido.



Muestra la ventana de contenido de Memoria.



Muestra la ventana del Desensamblador de código.



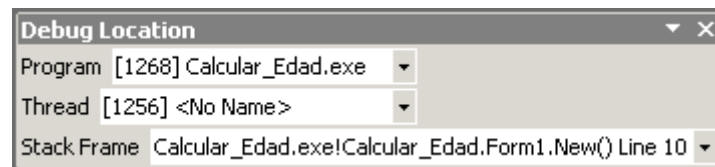
Visualiza la ventana de Registros del procesador.



Presenta la ventana del desensamblador para fijar cursor.

- **Barra de Ubicación de Depuración:** Es una nueva barra que muestra información de la aplicación que se está depurando, tal como el nombre del programa, el thread y el procedimiento que se encuentra en ejecución (en la pila).

**Figura 2.3: Barra de Ubicación de Depuración**



Program [1268] Calcular\_Edad.exe    Muestra el Programa que se está depurando.

Thread [1256] <No Name>    Visualiza el Thread que se está depurando.

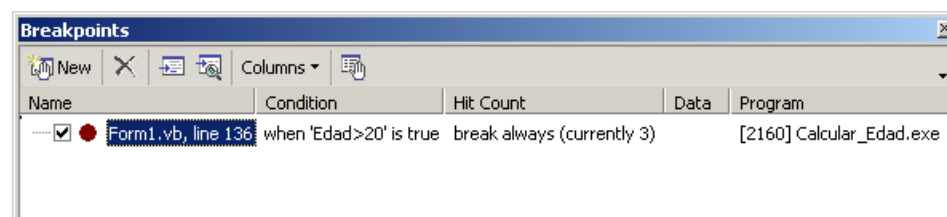
Stack Frame Calcular\_Edad.exe!Calcular\_Edad.Form1.New() Line 10    Muestra el Procedimiento que se encuentra en ejecución.

## Ventanas de Depuración

En esta nueva versión de Visual Basic existen alrededor de 17 ventanas de depuración, entre las cuales veremos algunas:

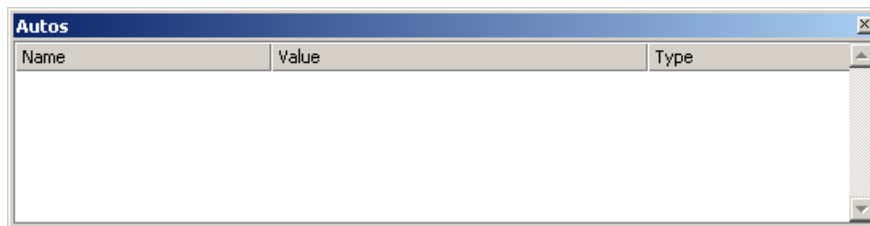
- **Ventana Breakpoints:** Presenta información de los puntos de interrupción insertados tal como la ubicación, condición, numero de veces que ingresa, datos y nombre del programa que está en ejecución.

**Figura 2.4: Ventana Breakpoints**



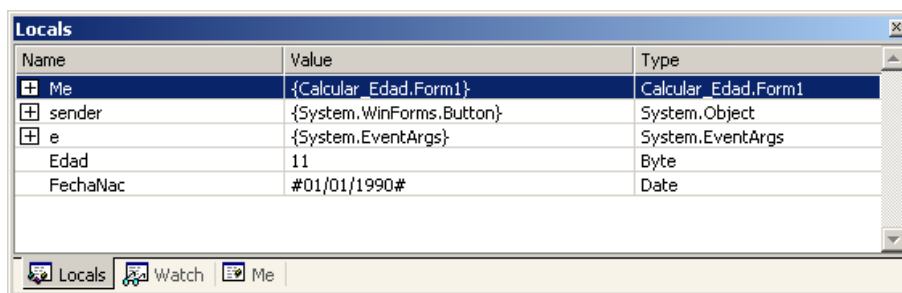
- **Ventana Autos:** Muestra información de las variables usadas en la sentencia actual y en las sentencias anteriores; la sentencia actual es aquella en donde se encuentra la depuración. Esta ventana no puede reconocer arrays ni estructuras.

**Figura 2.5: Ventana Autos**



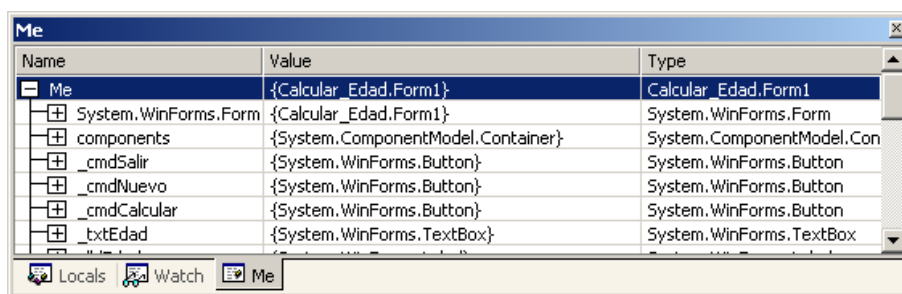
- **Ventana Locals:** Inspecciona variables, parámetros y objetos relacionados con el procedimiento actual o que se encuentra en depuración. También permite la modificación de valores dando un doble clic sobre estos.

**Figura 2.6: Ventana Locals**



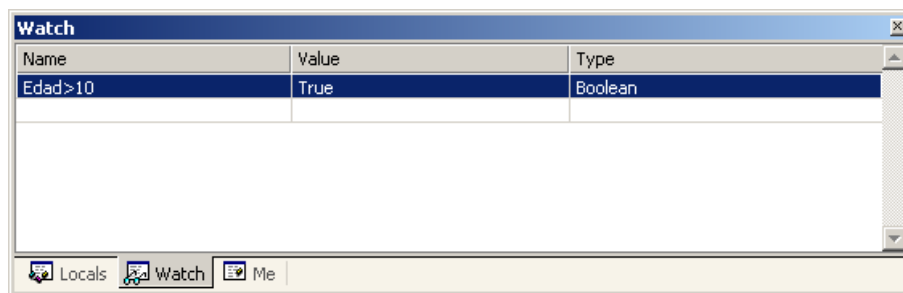
- **Ventana This:** Inspecciona el objeto (módulo) que se encuentra en depuración, mostrando su contenido. También permite la modificación de valores dando un doble clic sobre estos.

**Figura 2.7: Ventana This**



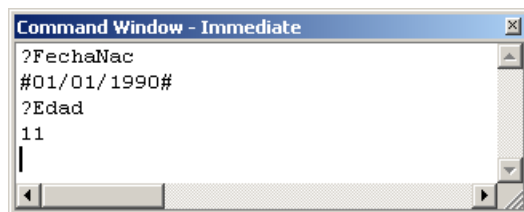
- **Ventana Watch:** Inspecciona una variable o expresión previamente definida. También permite añadir una expresión de análisis, modificar sus valores y eliminarla.

**Figura 2.8: Ventana Watch**



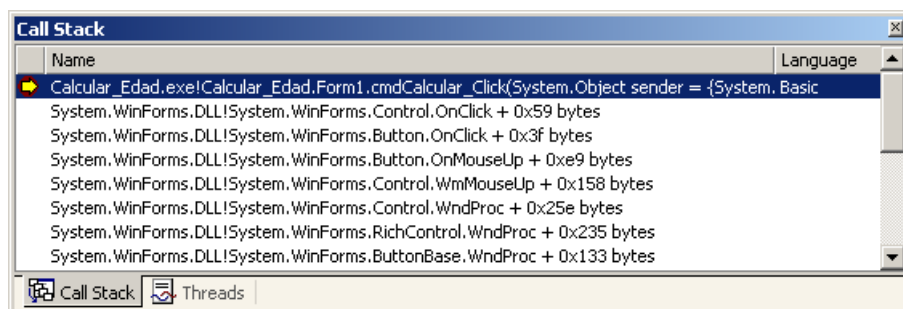
- **Ventana Immediate:** Se usa para preguntar y/o modificar valores de variables que se encuentran en ejecución o públicas, también ejecuta instrucciones o comandos.

**Figura 2.9: Ventana Immediate**



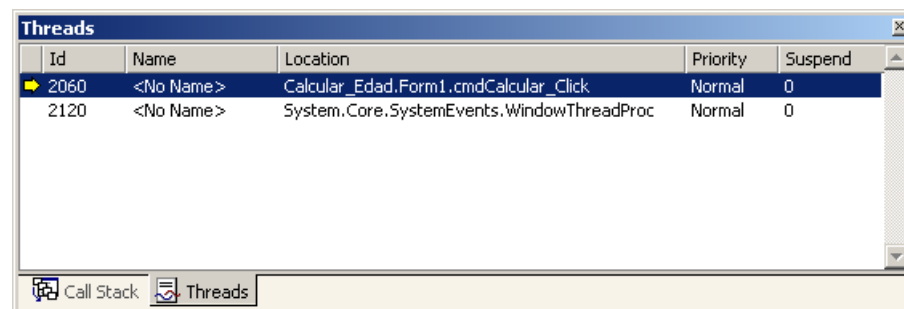
- **Ventana Call Stack:** Visualiza los procedimientos que se encuentran en ejecución en la memoria dinámica o pila, si es una aplicación .NET también señala el lenguaje.

**Figura 2.10: Ventana Call Stack**



- **Ventana Threads:** Muestra todos los threads que se encuentran actualmente en ejecución mientras se está depurando la aplicación, presentando información de su código, nombre, ubicación, prioridad y si se encuentra suspendido o no.

**Figura 2.11: Ventana Threads**



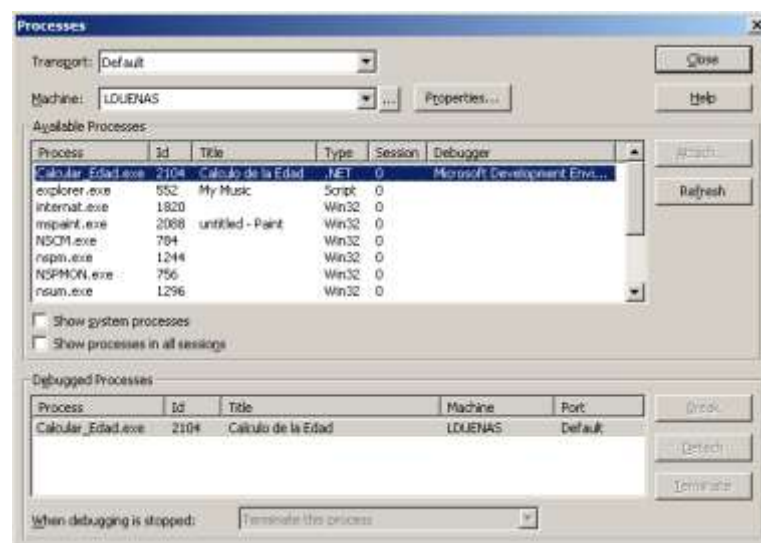
- **Ventana Modules:** Presenta información sobre los módulos cargados en memoria (la Aplicación y sus DLLs) mostrando su nombre, dirección, ruta, versión, programa, etc.

**Figura 2.12: Ventana Modules**

Name	Address	Path	Order	Version	Program	Information
Microsoft.VisualBasic.DLL	53140000-53172000	D:\WINNT\Microsoft.NET...	7	7.0.9030.0	[2104] Calcular_Edad.exe	Symbols loaded.
System.Drawing.DLL	5F250000-5F2A8000	D:\WINNT\Microsoft.NET...	6	1.0.2204.21	[2104] Calcular_Edad.exe	Symbols loaded.
Microsoft.Win32.Interop.DLL	5EEA0000-5EF4C000	D:\WINNT\Microsoft.NET...	5	1.0.2204.21	[2104] Calcular_Edad.exe	Symbols loaded.
System.DLL	5F1A0000-5F232000	D:\WINNT\Microsoft.NET...	4	1.0.2204.21	[2104] Calcular_Edad.exe	Symbols loaded.
System.Windows.Forms.DLL	5F7F0000-5F950000	D:\WINNT\Microsoft.NET...	3	1.0.2204.21	[2104] Calcular_Edad.exe	Symbols loaded.
Calcular_Edad.exe	00400000-0040A000	C:\Luis\VB\VB7\Labs\Calc...	2	1.0.434.42276	[2104] Calcular_Edad.exe	Symbols loaded.
mscorlib.dll	6DA30000-6DBBE000	D:\WINNT\Microsoft.NET...	1	1.0.2204.21	[2104] Calcular_Edad.exe	Symbols loaded.

- **Ventana Processes:** Visualiza los procesos que se encuentran en ejecución en el sistema, también permite anexar y desanexar procesos externos para depurarlos, etc.

**Figura 2.13: Ventana Processes**



## Control de Excepciones

Durante el desarrollo y ejecución de toda aplicación pueden presentarse diferentes tipos de errores, los cuales impiden el normal funcionamiento de la aplicación. A estos errores se les llama Excepciones.

### Tipos de Errores

Los errores o excepciones se pueden clasificar en 3 tipos:

- **Errores de Sintaxis:** Suceden al escribir código en la aplicación; como por ejemplo una instrucción mal escrita, omisión de un parámetro obligatorio en una función, etc.

Visual Basic notifica de cualquier error de sintaxis mostrando una marca de subrayado azul (por defecto) con un comentario indicando la causa del error.

Una ayuda para corregir errores de sintaxis, es usar la sentencia **Option Explicit** que fuerce a declarar variables y evitar expresiones inválidas.

- **Errores Lógicos:** Ocurren una vez usada la aplicación y consiste en resultados inesperados o no deseados; por ejemplo una función que debe devolver el sueldo neto está devolviendo un valor de cero o negativo, o una subrutina que debe eliminar un archivo temporal no lo está borrando.

Para corregir este tipo de errores se hace uso de las herramientas de depuración, como por ejemplo un seguimiento paso a paso, o inspeccionar el valor de una variable o expresión.

También podemos disminuir errores o excepciones de tipos de datos no deseados usando la sentencia **Option Strict** que evita la conversión forzosa y verifica que el tipo de dato asignado sea del mismo tipo que la variable o función, o que un parámetro pasado sea del mismo tipo, etc.

- **Errores en Tiempo de Ejecución:** Este tipo de errores suceden en plena ejecución de la aplicación, después de haber sido compilado el código. No son errores de mala escritura ni de lógica, sino mas bien de alguna excepción del sistema, como por ejemplo tratar de leer un archivo que no existe o que está abierto, realizar una división entre cero, etc.

Para controlar los errores en tiempo de ejecución disponemos de los Manipuladores o Controladores de Error, que evitan la caída del programa y permiten que siga funcionando. A continuación veremos las formas de implementar el control de este tipo de errores.

## Formas de Controlar Excepciones

Existen dos formas de controlar errores o excepciones en Visual Basic .NET:

- **Control No Estructurado:** Se implementa usando la sentencia **On Error GoTo**. Es la forma como se controla errores en las versiones anteriores a Visual Basic .NET y consiste en el uso de etiquetas, es decir recurrir a la programación etiquetada, cuando ocurre algún error toma el control el código que se encuentra a continuación de la etiqueta.

Existen varias sintaxis o formas de usar la sentencia On Error, tal como se define:

1. **On Error Resume Next:** Indica que si ocurre un error en tiempo de ejecución el flujo continúe en la siguiente línea después de la que originó el error.
2. **On Error GoTo 0:** Deshabilita cualquier Controlador de error, previamente declarado en el procedimiento actual, configurando este a Nothing.
3. **On Error GoTo -1:** Deshabilita cualquier error o excepción que ocurra en cualquier línea del procedimiento actual, configurando este a Nothing.
4. **On Error GoTo Etiqueta:** Si un error en tiempo de ejecución ocurre envía el control a la instrucción que está debajo de la etiqueta definida. Es la mejor forma no estructurada de controlar errores, ya que se puede personalizar mensajes.

La forma de implementar esta sentencia en un procedimiento es:

```
Inicio Procedimiento()  
    On Error GoTo <Etiqueta>  
    <Instrucciones>  
    Exit Sub  
<Etiqueta>:  
    <Instrucciones>  
    [Resume | Resume Next | Resume Etiqueta]  
Fin Procedimiento
```

Para salir de la etiqueta, generalmente se usa la instrucción Resume, de 3 formas:

- ✓ **Resume:** Vuelve a la instrucción que produjo el error.
- ✓ **Resume Next:** Regresa el flujo a la siguiente instrucción después de la que produjo el error.
- ✓ **Resume Etiqueta:** Bifurca el control a una etiqueta definida por el usuario.

Al usar la sentencia On Error GoTo si un error en tiempo de ejecución ocurre el control es tomado por el código de la Etiqueta y la información del error ocurrido es almacenada en el **Objeto Err**, tal como el número del error, su descripción y el origen.

- **Control Estructurado:** Es la forma mas recomendable de controlar errores y es una nueva característica de Visual Basic .NET; se implementa usando la estructura **Try..Catch..Finally**.

La forma de implementar esta sentencia en un procedimiento es:

```
Inicio Procedimiento()  
    Try  
        <Instrucciones Try>  
        [Exit Try]  
    [Catch 1 [<Excepción> [As <Tipo Dato>]] [When <Expresión>]]  
        <Instrucciones Catch 1>  
        [Exit Try]  
    [Catch 2 [<Excepción> [As <Tipo Dato>]] [When <Expresión>]]  
        <Instrucciones Catch 2>  
        [Exit Try]  
        :  
        :  
    [Catch n [<Excepción> [As <Tipo Dato>]] [When <Expresión>]]  
        <Instrucciones Catch n>  
        [Exit Try]  
    Finally  
        <Instrucciones Finally>  
    End Try  
Fin Procedimiento
```

Cuando usamos la sentencia Catch se hace referencia a una variable <Excepción> que es del Tipo de Dato **Exception** (nombre de la clase) o una clase derivada de ésta, la cual contiene información sobre el error ocurrido, tales como su número, descripción, origen, etc.

En la sentencia Catch también podemos analizar el error usando la palabra When seguida de una expresión, lo que permite filtrar el error de acuerdo a un criterio, que generalmente es el número del error.

Para salir de la estructura Try..Catch..Finally se hace uso de la sentencia Exit Try que ejecuta la siguiente línea a la sentencia End Try.

En la estructura Try..Catch..Finally se ejecuta las instrucciones que hubieran debajo del Try, si un error en tiempo de ejecución ocurre, el flujo es pasado a la sentencia Catch que contenga el control de dicho error, si dicho error no es especificado por el Catch se mostrará un mensaje de error normal. Finalmente se ejecuta las instrucciones del Finally.

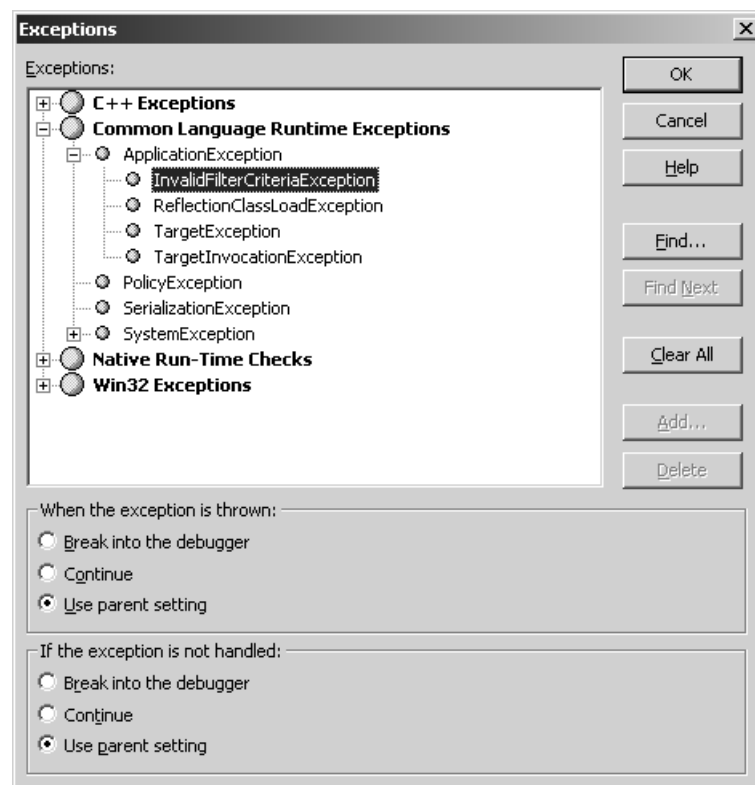


## Opciones de Control de Excepciones

Para controlar excepciones no solo basta usar una forma de control, sea no estructurada o estructurada, sino también es necesario configurar las opciones de control de excepciones para lo cual del menú "Debug" se elige "Windows" y luego "Exceptions" ó también se puede elegir directamente el botón "Exceptions" de la Barra de Depuración.

Aparecerá la ventana de control de excepciones que se muestra a continuación:

**Figura 2.14: Ventana Exceptions**



Existen 2 momentos en que se pueden controlar las excepciones que son:

- **When the exception is thrown:** Es cuando la excepción ocurre y se aconseja configurarlo en "Continue".
- **If the exception is not handled:** Ocurre cuando la excepción no es controlada, es decir cuando no existe controlador de error, es aconsejable que esté en "Break into the debugger".

Ambos momentos tienen 3 opciones o formas de control, que son:

- **Break into the debugger:** Para y muestra el depurador con la línea donde ocurrió el error en tiempo de ejecución.
- **Continue:** Continúa la ejecución del programa. Si no existe controlador Finaliza la ejecución del programa.
- **Use parent setting:** Indica que se va a usar la configuración de la opción de nivel superior o padre.

### **Laboratorio 3:**

El presente laboratorio tiene como objetivo conocer el Entorno de Visual Studio .NET y usar algunas características nuevas del Lenguaje Visual Basic .NET, así como implementar Multi Threads y Controlar Excepciones.

Este primer laboratorio está dividido en 2 ejercicios que tienen una duración aproximada de **35 minutos**.

#### **Ejercicio 1: Reconociendo VB .NET y Trabajando con el Lenguaje**

##### ***Duración: 20 minutos***








-  Cargue Visual Studio .NET, aparecerá la pagina de inicio, en ella elegir la opción “Get Started” y luego elegir “Create New Project”, observe los diferentes tipos de proyectos y sus respectivas plantillas.
-  Elegir un tipo de proyecto “Visual Basic” y una plantilla de “Aplicación Windows”, seleccionar como ubicación la carpeta “C:\VBNET\Labs” y como nombre ingrese “Lab03\_1”.
-  Observe el IDE de Visual Studio .NET cuando se trata de una aplicación para Windows. ¿Qué ventanas son nuevas en esta versión?. Luego, pruebe la característica de “Auto Ocultar” de las ventanas y la ayuda dinámica.
-  Vamos a empezar a crear una aplicación en la que en un formulario se ingresen los datos de unos alumnos, tal como su nombre, notas de sus exámenes parcial y final y se calculará el promedio y condición. Además, en otro formulario se mostrará un informe del total de alumnos ingresados, aprobados, desaprobados y el promedio total
-  Primero en la ventana del “Solution Explorer” seleccionar el nombre del formulario y en la ventana de propiedades escribir en la propiedad “FileName” el nombre “frmDatos.vb”.
-  Realizar el diseño del formulario similar al de la figura 1.18, ingresando como prefijo de la propiedad “Name” de las etiquetas las iniciales “lbl”, para los textos “txt” y para los botones “btn”, seguido del nombre que generalmente es similar al contenido de la propiedad “Text”.
-  Después de realizar el diseño y configurar las respectivas propiedades de los objetos, visualice el código del formulario. Observe las nuevas características del Lenguaje y notes la diferencias entre esta versión y la anterior.

Figura 2.15: Diseño del formulario frmDatos


Para realizar los cálculos de la aplicación vamos a crear variables y funciones públicas en un módulo estándar, para lo cual del menú "Project" elegimos la opción "Add Module" y luego escribimos en el nombre "Modulo.vb"

Escribir el siguiente código en el módulo:

```
Module Modulo
    Public NTotal, NProbados, NDesaprobados As Byte
    Public Suma, PromTot As Single

    Public Function Calcular_Promedio(ByVal Par As Byte, _
                                     ByVal Fin As Byte) As Single
        Return ((Convert.ToSingle(Par + 2 * Fin)) / 3)
    End Function

    Public Function Calcular_Condicion(ByVal Pro As Single) _
                                     As String
        NTotal = +1
        Suma = +Pro
        If Pro > 10.5 Then
            NProbados = +1
            Return ("Aprobado")
        Else
            NDesaprobados = +1
            Return ("Desaprobado")
        End If
    End Function
End Module
```


-  Regresar al formulario y proceder a escribir código en los eventos de los botones, tal como se muestra a continuación:

```
Protected Sub btnCalcular_Click(ByVal sender As Object, ...)
    Dim Par, Fin As Byte
    Dim Pro As Single
    Par = Convert.ToByte(txtParcial.Text)
    Fin = Convert.ToByte(txtFinal.Text)
    Pro = Calcular_Promedio(Par, Fin)
    txtPromedio.Text = Format(Pro, "#.00")
    txtCondicion.Text = Calcular_Condicion(Pro)
End Sub
```

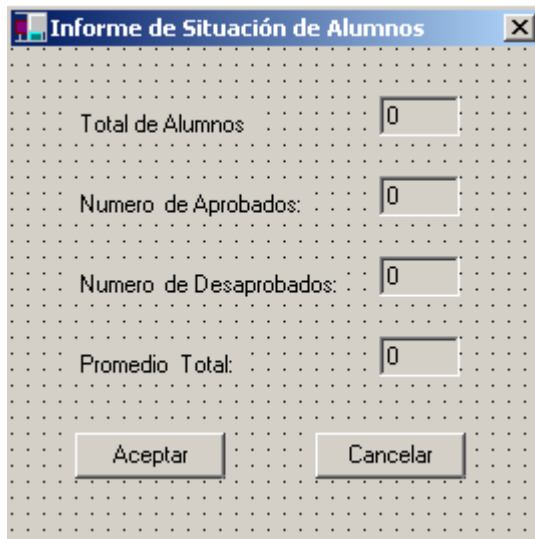
```
Protected Sub btnNuevo_Click(ByVal sender As Object, ...)
    Dim X As Control
    For Each X In Me.Controls
        If TypeOf X Is TextBox Then X.Text = ""
    Next
    txtNombre.Focus()
End Sub
```

```
Protected Sub btnInforme_Click(ByVal sender As Object, ...)
    Dim X As New frmInforme()
    X.ShowDialog()
End Sub
```

```
Protected Sub btnSalir_Click(ByVal sender As Object, ...)
    Me.Close
End Sub
```

-  Para mostrar el informe de situación de alumnos ir al menú "Project" y elegir la opción "Add Windows Form" y en el nombre escribir "frmInforme.vb". Luego realizar el diseño del formulario, tal como se muestra en la figura de abajo.

**Figura 2.16: Diseño del formulario frmInforme**



- ⌨ Ingresar al código del formulario “frmInforme” y proceder a escribir código en el evento “Load” para que se muestre las estadísticas de los alumnos:

```
Public Sub frmInforme_Load(ByVal sender As Object, ...)
    txtTotal.Text = NTotal.ToString
    txtAprobados.Text = NAprobados.ToString
    txtDesaprobados.Text = NDesaprobados.ToString
    txtPromedioTotal.Text = Format((Suma / NTotal), "#.00")
End Sub
```

- ⌨ Luego programar en los botones de “Aceptar” y “Cancelar” el regreso y la finalización de la aplicación respectivamente, similar al código mostrado abajo:





```
Protected Sub btnAceptar_Click(ByVal sender As Object, ...)
    Me.Close()
End Sub

Protected Sub btnCancelar_Click(ByVal sender As Object, ...)
    End
End Sub
```

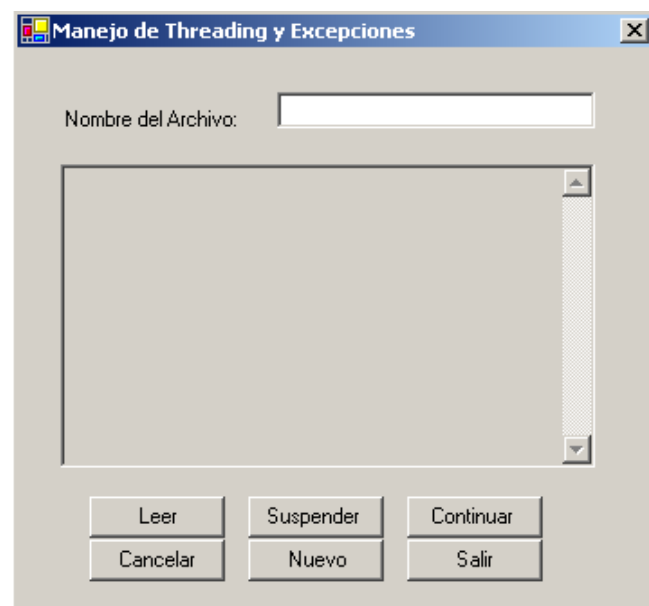
- ⌨ Grabar y ejecutar la aplicación; para probar ingresar como mínimo los datos de dos alumnos, mostrar el informe y finalizar.
- ⌨ Diríjase al “Explorador de Windows” y analice el contenido de la carpeta “Lab01\_1”, observe que existen varios tipos de extensiones, **sln** para la solución, **vbproj** para la aplicación y **vb** para los formularios y el módulo.
- ⌨ Finalmente, ingrese a la carpeta “bin” de “Lab01\_1” y observe dos archivos: uno ejecutable (extensión **exe**) y otro archivo intermedio (extensión **pdb**), ¿Cuál de estos es conocido como Assemblies?


## Ejercicio 2: Implementando Multi Thread y Control de Excepciones

**Duración: 15 minutos**

-  Ingrese a Visual Studio .NET y del menú "File" elegir "New" y luego "Project" o simplemente pulsar [Ctrl] + [N], luego elegir un tipo de proyecto "Visual Basic" y una plantilla de "Aplicación Windows", seleccionar como ubicación la carpeta "C:\VBNET\Labs" y como nombre ingresar "Lab03\_2".
-  Se va a crear una aplicación en la que se implemente múltiples threads para iniciar la lectura de un archivo de texto extenso y poder cancelar este proceso si es necesario, además para evitar los errores en tiempo de ejecución haremos uso del control estructurado de excepciones visto en este módulo.
-  Primero en la ventana del "Solution Explorer" seleccionar el nombre del formulario y en la ventana de propiedades escribir en la propiedad "FileName" el nombre "frmArchivo.vb".
-  Realizar el diseño del formulario similar al presentado aquí abajo:

**Figura 2.17: Diseño del formulario frmArchivo**



-  Declarar una variable llamada Hilo de tipo thread y definir una subrutina que lea el archivo de texto cuya ruta y nombre está especificado en el cuadro de texto, esta rutina usa Try..Catch..Finally para controlar errores en tiempo de ejecución.

```
Dim Hilo As System.Threading.Thread
```

```
Sub LeerArchivo()
```


```
    Dim Flujo As System.IO.StreamReader
```

```
    Dim Linea As String = " "
```

```

Try
    Flujo = System.IO.File.OpenText(txtNombre.Text)
    Do Until IsNothing(Linea)
        Linea = Flujo.ReadLine
        txtArchivo.Text = txtArchivo.Text + Linea + Chr(13)
                                   + Chr(10)
    Loop
Catch X As IO.IOException
    MsgBox(X.Message, MsgBoxStyle.Information, "No se pudo")
    txtNombre.Clear()
Exit Sub
End Try
Flujo.Close()
btnSuspende.Enabled = False
btnContinuar.Enabled = False
btnCancelar.Enabled = False
End Sub

```

 Escribir código en el evento “TextChanged” de “txtNombre” y en los eventos “Click” de los “Botones”.

```

Private Sub txtNombre_TextChanged(ByVal sender As ...) Handles ...
    btnLeer.Enabled = txtNombre.Text <> ""
End Sub

```

```

Private Sub btnLeer_Click(ByVal sender As ...) Handles ...
    Hilo = New System.Threading.Thread(AddressOf LeerArchivo)
    Hilo.Start()
    btnSuspende.Enabled = True
    btnCancelar.Enabled = True
End Sub

```

```

Private Sub btnCancelar_Click(ByVal sender As ...) Handles ...
    Hilo.Abort()
    btnLeer.Enabled = False
    btnSuspende.Enabled = False
    btnCancelar.Enabled = False
End Sub

```

```

Private Sub btnNuevo_Click(ByVal sender As ...) Handles ...
    txtNombre.Clear()
    txtArchivo.Clear()
    txtNombre.Focus()
End Sub

```

```


Private Sub btnSuspende_Click(ByVal sender As ...) Handles ...
    Hilo.Suspend()
    btnLeer.Enabled = False
    btnContinuar.Enabled = True

```

```
    btnCancelar.Enabled = False  
End Sub
```

```
Private Sub btnContinuar_Click(ByVal sender As ...) Handles ...  
    Hilo.Resume()  
End Sub
```

```
Private Sub btnSalir_Click(ByVal sender As ...) Handles ...  
    Me.Close()  
End Sub
```

 Finalmente, grabar y ejecutar la aplicación.



# Módulo 3

## Creando Aplicaciones para Windows

---

### *Usando Windows Forms*

#### Introducción

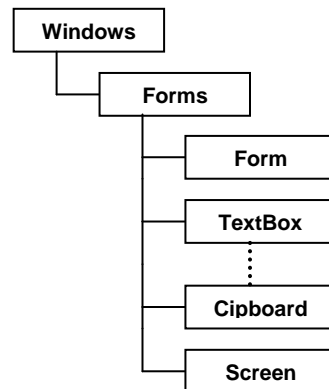
Windows es una clase base del Marco .NET usada para crear aplicaciones que correrán sobre Windows, esta se encuentra disponible al elegir en Visual Basic la plantilla “Windows Application”.

Al elegir una aplicación para Windows automáticamente se realizan los siguientes cambios:

- En el ToolBox aparecerá una ficha llamada “Windows Forms” conteniendo los controles para trabajar con Formularios Windows y Controles de Usuario.
- En el código del formulario aparecerá la referencia a la clase base heredada:  
`Inherits System.Windows.Forms.Form`


Para ilustrar mejor la funcionalidad que podemos obtener de Windows tenemos la figura 4.1, que resume los principales objetos usados en Windows, para lo cual se parte de la clase base llamada "System", luego se muestran algunos de los objetos de la clase "Drawing" y de la clase "WinForms" anteriormente comentadas.

**Figura 3.1: Modelo de Objetos para Windows**



Cabe comentar que dentro de la clase Windows encontramos definido el formulario y los controles para Windows (Label, Button, TextBox, Menu, etc), así como también algunos objetos no visuales de utilidad como Application (reemplaza al objeto App de VB6), Clipboard, Help, Screen, etc.

### Objeto Formulario

El objeto Formulario  `Form` es el contenedor principal de toda aplicación para Windows y se encuentra en el siguiente NameSpace:

**"System.Windows.Forms.Form"**

En Visual Studio .NET el formulario ha sufrido muchos cambios, tanto en propiedades, métodos y eventos, tal como se muestra en los siguientes cuadros:

### ***Propiedades***

<b>Propiedad</b>	<b>Descripción</b>
Autoscroll	Es una nueva propiedad que permite desplazarse por el formulario a través de una barra si es que los controles sobrepasan el área cliente.
BackColor	Especifica el color de fondo del formulario.
BackgroundImage	Antes llamada Picture. Permite mostrar una imagen de fondo sobre el formulario, tiene 2 tamaños: cascada y centrado en pantalla.
BorderStyle	Controla la apariencia del borde del formulario y los controles que se presentan en la barra de título. Tiene 6 opciones.
ControlBox	Si esta en True muestra el menú de controles de la barra de título, si esta en False no los muestra.
Cursor	Especifica el cursor que aparecerá al situar el mouse sobre el formulario. Antes era la propiedad MousePointer y si se quería un cursor personalizado se configuraba MouseIcon, ahora solo existe Cursor y sus gráficas son vistas en la lista.
Font	Configura la fuente de los textos de los controles ubicados en el formulario y de los textos mostrados con métodos de dibujo.
ForeColor	Especifica el color del texto de los controles (excepto el TextBox) y de los textos mostrados con métodos de dibujo.
GridSize	Determina el tamaño de las rejillas que se muestran en tiempo de diseño para diseñar controles.
Icon	Indica el icono del formulario, este se muestra en la barra de título de la ventana y en la barra de tareas de Windows.
IsMdiContainer	Determina si es que el formulario es un MDI, antes se creaba un formulario MDI añadiéndolo del menú "Project" y un formulario hijo configurando la propiedad MDIChild en True. Ahora solo se configura para ambos la propiedad IsMdiContainer.
Location	Indica la posición del formulario con respecto a la esquina superior izquierda de la pantalla. Antes había que configurar la propiedad Top y Left, ahora los valores de X e Y.
Opacity	Es una nueva propiedad, que indica la forma de visualización del formulario, que puede ser desde opaco (100%) hasta transparente (0%). Antes para hacer transparente se usaba la API SetWindowRgn
RightToLeft	Determina la alineación de los textos con respecto a sus controles, por

	defecto es No, es decir se alinean de izquierda a derecha; si es True se alinearán de derecha a izquierda.
Size	Configura el tamaño del formulario en píxeles.
StartPosition	Indica la posición en que aparecerá por primera vez el formulario con respecto a la pantalla. Tiene 5 opciones.
Text	Antes se llamaba Caption y permite mostrar el texto de la barra de título en el formulario.
TopMost	Posiciona en primer plano la ventana, siempre y cuando no este desactivada. Antes se podía hacer esto con la API WindowsOnTop.
WindowState	Determina la forma en que se presentará la ventana, puede ser Normal, Minimizada o Maximizada.

### **Métodos**

Método	Descripción
Activate	Activa el formulario y le da el foco.
ActivateControl	Activa un control del formulario.
Close	Cierra un formulario descargándolo de la memoria.
Focus	Pone el foco sobre el formulario.
Hide	Oculto el formulario, sin descargarlo de la memoria.
Refresh	Repinta el formulario y sus controles.
SetLocation	Ubica el formulario en una cierta posición de la pantalla.
SetSize	Configura el tamaño de la ventana en píxeles.
Show	Muestra un formulario como ventana no modal (modeles).
ShowDialog	Muestra un formulario como ventana modal (modal).

### **Eventos**

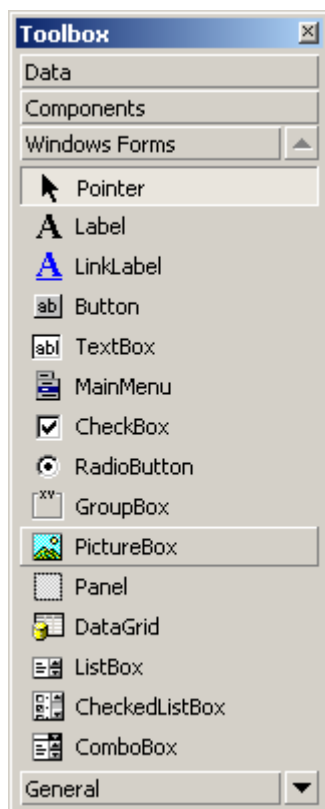
<b>Evento</b>	<b>Descripción</b>
Activated	Ocurre al activarse el formulario.
Click	Se desencadena al dar clic con el mouse sobre el formulario.
Closed	Se habilita al cerrar el formulario. Es similar al evento Unload de Visual Basic 6.
Closing	Ocurre mientras se está cerrando el formulario. Es similar al evento QueryClose de Visual Basic 6. También se puede cancelar la salida.
Deactivated	Ocurre al desactivarse el formulario.
DoubleClick	Se desencadena al dar doble clic con el mouse sobre el formulario.
GotFocus	Ocurre al ingresar el foco sobre el formulario.
Load	Se produce al cargar los controles sobre el formulario
LostFocus	Ocurre al salir el foco del formulario.
MouseEnter	Se habilita al ingresar el mouse sobre el área cliente del formulario.
MouseLeave	Se habilita al salir el mouse del área cliente del formulario.
MouseMove	Se desencadena al pasar el mouse sobre el formulario.
Move	Este evento se habilita al mover la ventana o formulario.
Paint	Ocurre al pintarse la ventana en pantalla.
Resize	Ocurre cada vez que se modifica de tamaño el formulario.

## Uso del ToolBox

El ToolBox es la caja de herramientas donde se encuentran los controles que se van a usar para diseñar la interfaz de los diferentes tipos de aplicaciones, este varía de acuerdo al tipo de plantilla elegida.

A continuación se presenta el ToolBox cuando elegimos una plantilla "Windows Application"; éste presenta varias fichas: "Windows Forms", "Data", "Components" y "General".

**Figura 3.2: ToolBox para Aplicaciones Windows**



Para usar un control del ToolBox solo hay que elegir la ficha adecuada y luego seleccionar el control y arrastrarlo sobre el formulario o contenedor donde se desea ubicarlo, también se puede dar doble clic sobre el control y aparecerá por defecto en la posición 0,0 del formulario (antes se ubicaba al centro de éste).

## Usando Controles para Windows Forms

### Controles Label, TextBox y Button

#### Control Label Label

#### Propiedades

Propiedad	Descripción
Anchor	Es una nueva propiedad que permite ajustar el ancho del control.
AutoSize	Ajusta el texto de la etiqueta al tamaño del control.
BackColor	Especifica el color de fondo de la etiqueta.
BorderStyle	Controla la apariencia del borde de la etiqueta. Tiene 3 opciones.
Cursor	Especifica el cursor que aparece al situar el mouse sobre la etiqueta.
Dock	Da la posibilidad de acoplar la etiqueta a un lado del contenedor, puede ser arriba, abajo, izquierda, derecha o al centro.
Enabled	Habilita o deshabilita la etiqueta.
Font	Configura la fuente del texto de la etiqueta.
ForeColor	Especifica el color del texto de la etiqueta.
Location	Indica la posición de la etiqueta con respecto a su contenedor.
Locked	Bloquea el control para que no se mueva o modifique de tamaño.
RightToLeft	Determina la alineación del texto con respecto al control.
Size	Configura el tamaño del control en píxeles.
Text	Visualiza el texto de la etiqueta.
TextAlign	Alinea el texto hacia el control, sea: izquierda, derecha o centro.
Visible	Visualiza o no el control.

### **Métodos**

Método	Descripción
FindForm	Devuelve el formulario en el que se encuentra el control.
Focus	Pone el foco sobre la etiqueta.
Hide	Oculto la etiqueta, sin descargarla de la memoria.
Refresh	Repinta la etiqueta.
SetLocation	Ubica la etiqueta en una cierta posición de la pantalla.
SetSize	Configura el tamaño de la etiqueta.
Show	Pone visible la etiqueta.

### **Eventos**

Evento	Descripción
Click	Se desencadena al dar clic con el mouse sobre la etiqueta.
DoubleClick	Se desencadena al dar doble clic con el mouse sobre la etiqueta.
GotFocus	Ocurre al ingresar el foco sobre el control.
LostFocus	Ocurre al salir el foco del control.
MouseEnter	Se habilita al ingresar el mouse sobre la etiqueta.
MouseLeave	Se habilita al salir el mouse de la etiqueta.
MouseMove	Se desencadena al pasar el mouse sobre la etiqueta.



## Control TextBox TextBox

### Propiedades

Este control tiene propiedades similares al control Label, entre aquellas propiedades exclusivas de este control tenemos:

Propiedad	Descripción
CharacterCasing	Nueva propiedad que convierte a mayúsculas o minúsculas el texto.
Lines	Muestra el contenido de cada línea del texto.
MaxLength	Determina el número de caracteres que se pueden ingresar en éste.
MultiLine	Si es True se pueden escribir varias líneas de texto.
PasswordChar	Señala el carácter que aparecerá como mascara de entrada.
ReadOnly	Indica que el control solo se puede ver pero no editar. Antes se llamaba Locked.
ScrollBars	Habilita las barras de desplazamiento si el control es multilínea.
WordWrap	Cambia de línea al llegar al final de un texto multilínea.

### Métodos

Método	Descripción
AppendText	Añade texto al final del texto actual.
Clear	Borra el contenido del cuadro de texto.
Copy	Copia el texto y lo envía al portapapeles.
Cut	Corta el texto y lo envía al portapapeles.
Paste	Pega el texto del portapapeles al cuadro.
ResetText	Inicializa el texto.
Select	Selecciona el texto.
Undo	Deshace el último cambio en el texto.

## Eventos

Evento	Descripción
KeyDown	Ocurre al pulsar hacia abajo una tecla extendida.
KeyPress	Ocurre al pulsar una tecla normal. También se desencadena antes el evento KeyDown y después el evento KeyUp.
KeyUp	Ocurre al soltar una tecla extendida previamente pulsada.
TextChanged	Es un nuevo evento que reemplaza al evento change, es decir ocurre al cambiar el texto.
Validated	Se habilita después de validarse el control.
Validating	Se habilita cuando el control está validándose.

## Control Button Button

### Propiedades

Este control también tiene propiedades similares al control Label, entre aquellas propiedades exclusivas y nuevas de esta versión, tenemos:

Propiedad	Descripción
Name	Generalmente usaremos el prefijo btn.
BackgroundImage	Especifica la imagen de fondo que usará el botón.
DialogResult	Determina el valor del formulario padre si se da clic sobre el botón.
FlatStyle	Determina el estilo o apariencia del control. Tiene 3 valores.
Image	Imagen que se mostrará en el control.
ImageAlign	Alineación de la imagen dentro del control. Tiene 9 opciones.

### Métodos

Método	Descripción
NotifyDefault	Indica si el botón será el control por defecto. Es de tipo lógico.

PerformClick	Ejecuta el evento clic del botón.
--------------	-----------------------------------

### **Eventos**

Evento	Descripción
Click	Se desencadena al dar clic con el mouse sobre la etiqueta.
GotFocus	Ocurre al ingresar el foco sobre el botón.
LostFocus	Ocurre al salir el foco del botón.
MouseEnter	Se habilita al ingresar el mouse sobre el botón.
MouseLeave	Se habilita al salir el mouse del botón.
MouseMove	Se desencadena al pasar el mouse sobre el botón.

## Controles GroupBox, RadioButton y CheckBox

### **Control GroupBox** GroupBox

Antes conocido como Frame, es un contenedor que se utiliza para agrupar varias opciones, que pueden ser: de opción única como los RadioButton o de opción múltiple como los CheckBox.

Este control se utiliza como contenedor y por si solo no tiene mucha funcionalidad, es por eso, que solo veremos sus principales propiedades, métodos y eventos.

### **Propiedades**

Propiedad	Descripción
Name	Generalmente usaremos el prefijo gbx.
Enabled	Determina si el control estará habilitado o deshabilitado.
Text	Indica el texto que se mostrará como encabezado del control.
Visible	Muestra u oculta al control y todo su contenido.

### Métodos

Método	Descripción
Focus	Pone el foco sobre el control.
Hide	Oculto el control, sin descargarlo de la memoria.
Show	Pone visible el cuadro de grupo.

### Eventos

Evento	Descripción
GotFocus	Ocurre al ingresar el foco sobre el control.
LostFocus	Ocurre al salir el foco del control.

### Control RadioButton RadioButton

Antes conocido como OptionButton, es un control en el que solo se puede seleccionar uno por contenedor.

### Propiedades

Propiedad	Descripción
Name	Generalmente usaremos el prefijo rbn.
Appearance	Controla la apariencia del control, puede ser: Normal (como botón de opción) o Button (como botón de comando).
AutoCheck	Cambia de estado cada vez que se da clic al botón.
CheckAlign	Controla la alineación del botón. Hay 9 posiciones.
Checked	Indica si el botón ha sido seleccionado o no.

### Métodos

Método	Descripción
Focus	Pone el foco sobre el radiobutton.
Hide	Oculto el radiobutton, sin descargarlo de la memoria.
Show	Pone visible el radiobutton.

--	--

### Eventos

Evento	Descripción
CheckedChanged	Ocurre al cambiar la propiedad checked del radiobutton.
Click	Se desencadena al dar clic con el mouse sobre el botón.
DoubleClick	Se desencadena al dar doble clic con el mouse sobre el botón.

### Control **CheckBox** `CheckBox`

Este control mantiene el mismo nombre anterior, es un control en el que se pueden seleccionar varios por contenedor.

### Propiedades

Propiedad	Descripción
Name	Generalmente usaremos el prefijo chk.
Appearance	Controla la apariencia del control, puede ser: Normal (como casilla) o Button (como botón de comando).
AutoCheck	Cambia de estado cada vez que se da clic al botón.
CheckAlign	Controla la alineación del checkbox. Hay 9 posiciones.
Checked	Indica si el checkbox ha sido seleccionado o no.
CheckState	Devuelve el estado del checkbox, que puede ser: Unchecked (sin marcar), Checked (marcado) o Indeterminate (gris).
ThreeState	Habilita o deshabilita el estado indeterminado del checkbox cada vez que se da el tercer clic.

### Métodos

Método	Descripción
Focus	Pone el foco sobre el checkbox.
Hide	Oculto el checkbox, sin descargarlo de la memoria.
Show	Pone visible el checkbox.

--	--

## Eventos

Evento	Descripción
CheckedChanged	Ocurre al cambiar el valor de la propiedad Checked del control.
CheckStateChanged	Ocurre al cambiar el valor de la propiedad CheckState del control.

## Controles ListBox, CheckedListBox y ComboBox

### Control **ListBox** ListBox

#### Propiedades

Propiedad	Descripción
Name	Generalmente usaremos el prefijo lst.
ColumnWidth	Indica el ancho de cada columna en una lista de varias columnas.
HorizontalExtent	Indica el ancho mínimo en píxeles que se requiere para que aparezca la barra horizontal.
HorizontalScrollbar	Muestra u oculta la barra de desplazamiento horizontal de la lista.
IntegralHeight	Determina que las opciones de la lista se vean en forma completa.
ItemHeight	Devuelve el alto en píxeles de cada elemento de la lista.
Items	Es la principal propiedad y se refiere a los elementos de la lista.
MultiColumn	Indica si los elementos se pueden ver en varias columnas.
ScrollAlwaysVisible	Visualiza siempre las 2 barras de desplazamiento.
SelectionMode	Determina la forma de selección que puede ser: None (ninguno), One (uno), MultiSimple (varios con click) y MultiExtended (varios con shift + click o ctrl + click).
Sorted	Ordena la lista en forma ascendente.
SelectedIndex	Devuelve o establece el índice del elemento seleccionado.
SelectedItem	Devuelve el ítem seleccionado de la lista.

### **Métodos**

Método	Descripción
FindString	Devuelve el índice de un elemento buscado en una lista. Si no existe devuelve -1 y si existe devuelve un número mayor que -1.
FindStringExact	Realiza una labor similar al método anterior pero compara con exactitud la cadena.
GetSelected	Devuelve true si un elemento ha sido seleccionado o false si no.
InsertItem	Inserta un elemento en una cierta posición de la lista.

### **Eventos**

Evento	Descripción
DoubleClick	Ocurre al dar dos veces clic sobre la lista.
SelectedIndexChanged	Ocurre al cambiar el índice del elemento seleccionado.

### **Colección Items**

Para trabajar con los elementos de una lista se hace uso de la colección Items, la cual se detalla a continuación:

### **Propiedades**

Propiedad	Descripción
All	Devuelve un objeto con todos los elementos de la lista.
Count	Devuelve el número de elementos de la lista.

### **Métodos**

Método	Descripción
Add	Añade un elemento al final de la lista.
Clear	Borra todos los elementos de la lista.
Insert	Inserta un elemento en la posición indicada por el índice.
Remove	Elimina un elemento de la lista de acuerdo a su índice.



### Control **CheckedListBox** `CheckedListBox`

Es un nuevo control que antes se obtenía configurando la propiedad `style` del control `Listbox` a `checked`. Como es similar al control `Listbox` solo mencionaremos las características distintas que tiene el control `CheckedListBox`.

#### **Propiedades**

Propiedad	Descripción
Name	Generalmente usaremos el prefijo <code>ckl</code> .
CheckOnClick	Establece si el control podrá ser fijado la primera vez al dar click.
ThreeDCheckBox	Indica si la apariencia de los items se mostrará en 3D o plano.

#### **Métodos**

Método	Descripción
GetItemChecked	Devuelve <code>true</code> si un cierto item ha sido seleccionado o <code>false</code> si no.
GetItemCheckState	Devuelve el valor de la propiedad <code>CheckState</code> de un cierto item.
SetItemChecked	Establece o quita la selección de un cierto elemento.
SetItemCheckState	Establece la propiedad <code>CheckState</code> de un cierto elemento.

#### **Eventos**

Evento	Descripción
ItemCheck	Ocurre al seleccionar un elemento y poner el check en <code>true</code> .
SelectedIndexChanged	Ocurre al seleccionar otro elemento.

### Control **ComboBox** `ComboBox`

## Propiedades

Propiedad	Descripción
Name	Generalmente usaremos el prefijo cbo.
Items	Es la principal propiedad y se refiere a los elementos del combo.
MaxDropDownItems	Indica el máximo número de elementos que se mostrarán al desplegarse el combo.
MaxLength	Determina el máximo número de caracteres que se podrán escribir en el cuadro de texto del combo.
Sorted	Ordena los elementos del combo en forma ascendente.
Style	Especifica el tipo de combo que puede ser: Simple, DropDown (por defecto), y DropDownList.
SelectedIndex	Devuelve o establece el índice del elemento seleccionado.
SelectedItem	Devuelve el ítem seleccionado de la lista.
SelectedText	Devuelve el texto seleccionado de la lista.
Text	Se refiere al texto escrito en el cuadro del combo.

## Métodos

Método	Descripción
FindString	Devuelve el índice de un elemento buscado en el combo. Si no existe devuelve -1 y si existe devuelve un número mayor que -1.
FindStringExact	Realiza una labor similar al método anterior pero compara con exactitud la cadena.

### Eventos

Evento	Descripción
Click	Ocurre al dar clic con el mouse a un elemento de la lista.
DoubleClick	Se da al dar dos veces clic sobre un elemento de la lista.
SelectedIndexChanged	Ocurre al cambiar el índice del elemento seleccionado.
SelectionChangeCommitted	Se da cuando se selecciona un elemento del combo.
TextChanged	Ocurre al cambiar la propiedad Text del combo.

### **Colección Items**

La colección Items del combo es similar a la del ListBox.

### **Propiedades**

Propiedad	Descripción
All	Devuelve un objeto con todos los elementos del combo.
Count	Devuelve el número de elementos del combo.

### **Métodos**

Método	Descripción
Add	Añade un elemento al final del combo.
Clear	Borra todos los elementos del combo.
Insert	Inserta un elemento en la posición indicada por el índice.
Remove	Elimina un elemento del combo de acuerdo a su índice.

## **Interfaces**

### **Introducción**

Una interface es el medio de comunicación entre 2 entidades, en nuestro caso, la interface sirve de enlace entre el usuario y la aplicación.

En la evolución de la computación se inicia con interfaces de texto o de consola, las cuales predominan desde los inicios de la computación hasta casi la mitad de la década del 80, luego aparecen las interfaces gráficas.

Desde que trabajamos en ambiente Windows, las interfaces han ido evolucionando de acuerdo a la facilidad del usuario para acceder a los elementos de la aplicación, y entre las principales interfaces tenemos:

- **SDI (Single Document Interface):** Interface de Simple Documento, muestra una sola ventana con un cierto documento en la aplicación; el acceso a las ventanas es secuencial, por lo que no es tan recomendable. Algunas aplicaciones con SDI son los accesorios de Windows: Bloc de notas, Paint, Wordpad, etc.
- **MDI (Multiple Document Interface):** Interface de Múltiples Documentos, muestra varios documentos en sus respectivas ventanas, las cuales aparecen sobre una ventana principal; el acceso a las ventanas es directo porque generalmente en la ventana padre existe un menú. Algunas aplicaciones con MDI son los programas de Office: Word y Excel.
- **TreeView - ListView (Vistas Árbol – Lista):** Muestra los elementos de la aplicación en un árbol (TreeView) y en el lado derecho muestra una lista con los detalles (ListView); puede mostrarse junto a un SDI como en el caso del Explorador de archivos de Windows o puede mostrarse junto a un SDI como en el caso del Enterprise Manager de SQL Server 6 o superior.

Con la evolución de Internet también se distinguen diferentes tipos de interfaces en el browser, pero que no se tocan en este capítulo, si no que nos centraremos en la creación de interfaces para aplicaciones Windows.

### **Creando Aplicaciones MDI**

Una aplicación MDI consta de 2 partes: un Formulario MDI Padre y uno o mas Formularios MDI Hijos, la creación de ambos es muy sencilla en VB .NET y se explica a continuación:

#### **Creando un Formulario MDI Padre**

Para crear un Formulario MDI padre solo hay que configurar la propiedad **IsMdiContainer** del formulario a True.

A diferencia de la versión anterior de Visual Basic, esta versión permite colocar cualquier control WinForm dentro del formulario MDI, pero esto hará que los formularios hijos se muestren en segundo plano, ya que en primer plano se verán los controles del formulario MDI padre.

### **Creando un Formulario MDI Hijo**

Para crear un Formulario MDI hijo solo hay que configurar la Propiedad **Parent** (disponible solo en tiempo de ejecución) del formulario hijo apuntando al formulario padre y luego usar el método **Show** para mostrarlo.

El siguiente código muestra como mostrar un formulario hijo desde un menú:

```
Protected Sub mnuArchivo_Nuevo_Click(ByVal sender As Object, ...)  
    Dim X As New frmHijo()  
    X.MDIParent = frmPadre  
    X.Show()  
End Sub
```

### **Organizando Formularios MDI Hijos**

Si es que desea organizar los formularios MDI hijos se debe usar el método **LayoutMDI** del formulario MDI padre junto con una constante de tipo MDILayout, que tiene 4 valores: Arrangelcons, Cascade, TileHorizontal y TileVertical.

A continuación se muestra como ordenar en Cascada los formularios MDI hijos de un formulario MDI Padre llamado frmPadre:

```
frmPadre.LayoutMDI(MDILayout.Cascade)
```

### **Controles TreeView y ListView**


Estos 2 controles casi siempre trabajan juntos, uno muestra los elementos de la aplicación y el otro su contenido o detalle. Antes estaban disponibles como controles ActiveX, ahora en VB NET están disponibles como controles del WinForm.

Ambos controles generalmente usan imágenes para mostrar los elementos, por lo cual primero veremos el uso del control ImageList para almacenar imágenes de las vistas.

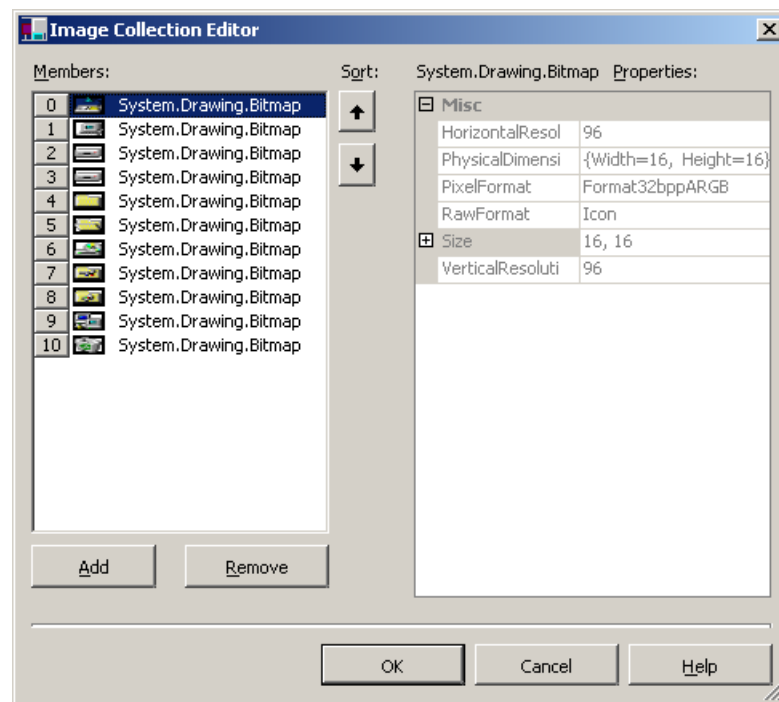
### **Trabajando con el ImageList** ImageList

Para llenar una lista de imágenes realice los siguientes pasos:

- Doble clic al control ImageList del ToolBox y se mostrará en la parte inferior del diseñador de formularios.

- Configurar la propiedad "Name" usando el prefijo `ils` seguido del nombre y la propiedad "ImageSize" que define el tamaño de las imágenes: 16, 32, 48 etc.
- Seleccionar la propiedad "Image" que es una colección y pulsar sobre el botón , entonces aparecerá el diálogo "Image Collection Editor" en el cual se añadirán las imágenes con el botón "Add" y se eliminarán con "Remove".


**Figura 3.3: Editor de la Colección de Imágenes**



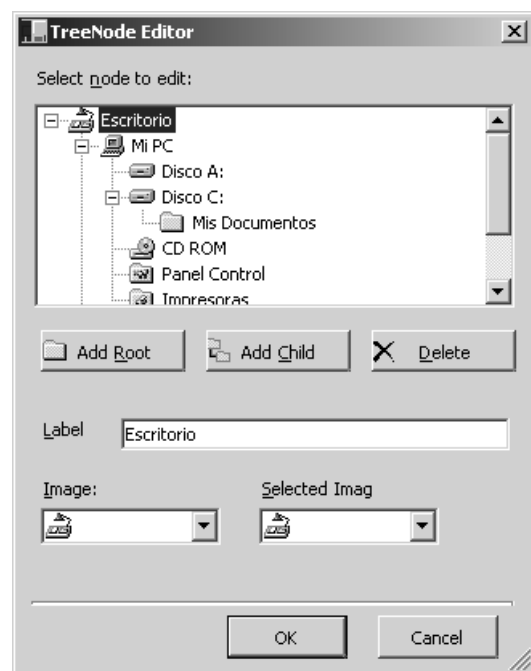
### Trabajando con el TreeView TreeView

El trabajo con el TreeView es mas sencillo con VB NET, para lo cual se realiza los siguientes pasos:

- Llenar un ImageList con las imágenes que se usaran en el TreeView.
- Dar doble clic al control TreeView y configurar la propiedad "Name" escribiendo el prefijo `tw` seguido del nombre.
- Configurar la propiedad "Anchor" en `TopBottomLeft` para que al modificarse de tamaño el formulario se ajuste automáticamente el tamaño del TreeView.
- Configurar la propiedad "ImageList" eligiendo el nombre de la Lista de Imágenes.


- Seleccionar la propiedad “Nodes” que es una colección y pulsar sobre el botón , aparecerá el dialogo “TreeNode Editor”.
- Para crear un nodo principal dar clic en el botón “Add Root”, luego seleccionar el nodo y escribir en “Label” el texto que irá en el nodo, finalmente en las listas “Images” y “Selected Imag” elegir las imágenes que se verán en el nodo.
- Para crear un nodo hijo dar clic en el botón “Add Child” y seguir el mismo paso anterior, es decir, seleccionar el nodo hijo, escribir el “Label” y llenar las listas.

**Figura 3.4: Editor de la Colección de Nodos**

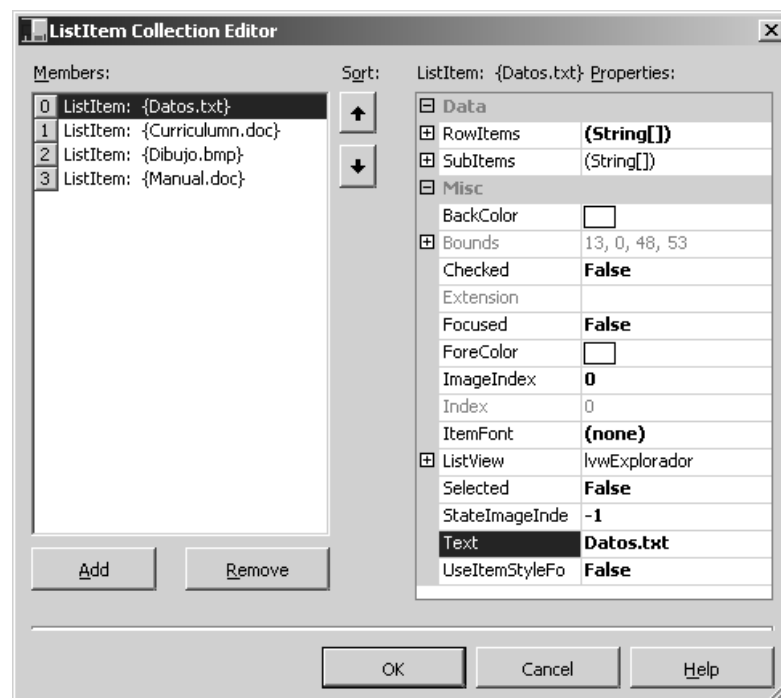


## Trabajando con el ListView ListView

El trabajo con el ListView es similar al del TreeView, realizándose los siguientes pasos:

- Llenar dos ImageList con las imágenes que se usaran en el ListView para la vista de Iconos grandes y otra para la vista de Iconos pequeños.
- Dar doble clic al control ListView y configurar la propiedad "Name" escribiendo el prefijo lvw seguido del nombre.
- Configurar la propiedad "Anchor" en All para que al modificarse de tamaño el formulario se ajuste automáticamente el tamaño del ListView.
- Configurar las propiedad "LargeImageList" y "SmallImageList" eligiendo el nombre de las Listas de Imágenes grande y pequeña respectivamente.
- Seleccionar la propiedad "ListItems" que es una colección y pulsar sobre el botón , aparecerá el dialogo "ListItem Collection Editor", en el cual se añadirán items con el botón "Add" y se eliminarán con "Remove".
- Para añadir un ListItem clic en "Add" y escribir en "Text" el texto del item, en "Index" indicar el índice de la imagen de las listas. Además de llenar la colección de ListItems también se debe llenar la colección de "Columns" para las cabeceras de las columnas en la vista detalle.

**Figura 3.5: Editor de la Colección de ListItem**





## ***Añadiendo Menús, Diálogos y Barras***

Una vez creada la interface de la aplicación, es necesario aumentar características que ayuden a facilitar el trabajo al usuario; tales como: menús que permitan organizar opciones, diálogos que faciliten la elección de archivos, colores, fuentes, etc. y barras de herramientas que permitan elegir opciones rápidamente.

### **Menús**

Un menú muestra un conjunto de opciones distribuidas u organizadas de acuerdo a categorías que el usuario defina.

En Windows, existen dos tipos de menús muy usados:

1. **Menús Principales:** Se acoplan en algún extremo del formulario, generalmente, en la parte superior de éste.
2. **Menús Contextuales:** También llamados Flotantes, generalmente, se muestran al dar clic derecho sobre algún objeto y su contenido varía de acuerdo al contexto.

### **Control MainMenu** MainMenu

Permite crear un menú principal, para lo cual se realizan los siguientes pasos:

- Dar doble clic sobre el control “MainMenu” del ToolBox y se mostrarán 2 objetos: uno en la parte superior del formulario que es donde se crearán las opciones del menú y otro en la parte inferior del diseñador de formularios que representa a todo el menú.
- Para crear una opción del menú solo hay que escribir directamente donde dice “Type Here” (propiedad Text), luego configuramos el nombre de la opción mediante la propiedad “Name” usando el prefijo mnu seguido del nombre.
- Si deseamos crear un acceso directo para alguna opción del menú configuramos la propiedad “ShortCut” eligiendo un valor de la lista.
- Para crear una opción que sea un separador simplemente en el “Text” escribir “-”.
- Después de crear todas las opciones del menú principal escribir código para cada opción, generalmente en el evento “Click”. Aunque si deseamos realizar una acción como mostrar un mensaje al pasar por la opción se puede usar el evento “Select”.

## Control ContextMenu ContextMenu

Se usa para crear un menú contextual, para lo cual se realizan los siguientes pasos:

- Dar doble clic sobre el control “ContextMenu” del ToolBox y se mostrarán 2 objetos: uno en la parte superior del formulario que es donde se crearán las opciones del menú y otro en la parte inferior del diseñador de formularios que representa a todo el menú contextual.
- La creación de las opciones del menú contextual es similar a la creación de un menú principal; aunque si se desea crear un menú contextual de un solo nivel, las opciones se deben crear en forma horizontal (pero se verán en forma vertical).
- Finalmente el menú contextual debe mostrarse al dar clic derecho sobre un cierto objeto (generalmente un control); antes se conseguía esto programando en el evento MouseDown del objeto; ahora solo configuramos la propiedad “ContextMenu” del objeto asignándole el objeto menú contextual.

## Diálogos

### Controles OpenFileDialog OpenFileDialog y SaveFileDialog SaveFileDialog

Estos controles se usan para facilitar el trabajo con archivos, el primero se refiere al diálogo de “Abrir Archivo” y el segundo al diálogo “Guardar Archivo”, que en la versión anterior estaban disponibles como Controles ActiveX.

Ambos tienen características similares que detallamos a continuación:

#### Propiedades

Propiedad	Descripción
Name	Para el OpenFileDialog generalmente usaremos el prefijo odg. Para el SaveFileDialog generalmente usaremos el prefijo sdg.
AddExtension	Añade automáticamente la extensión al nombre del archivo.
CheckFileExists	Chequea que exista el archivo antes de regresar del diálogo.
CheckPathExists	Chequea que exista la ruta del archivo antes de regresar del diálogo.
CreatePrompt	Solo para el diálogo de Guardar. Si la propiedad ValidateName es true pide confirmación al usuario cuando el archivo es creado.
DefaultEx	Indica la extensión por defecto del archivo.
FileName	Indica el archivo escrito o seleccionado del diálogo.

Filter	Especifica el tipo de archivo que se mostrará en el diálogo.
FilterIndex	Determina el índice del filtro del diálogo. Este empieza en 1 y depende de la lista de tipos de archivos configurada por Filter.
InitialDirectory	Muestra un cierto directorio inicial para los archivos del diálogo.
Multiselect	Solo para el diálogo de Abrir. Determina si se pueden seleccionar varios archivos a la hora de abrir.
OverwritePrompt	Solo para el diálogo de Guardar. Si la propiedad ValidateName es true pide confirmación al usuario cuando un archivo creado existe.
ReadOnlyChecked	Solo para el diálogo de Abrir. Determina el estado del checkbox ReadOnly en el diálogo de abrir.
RestoreDirectory	Controla si el diálogo restaura el directorio actual antes de cerrarse.
ShowHelp	Visualiza o no el botón de Ayuda en el diálogo.
ShowReadOnly	Solo para el diálogo de Abrir. Determina si se muestra o no el checkbox ReadOnly en el diálogo de abrir.
Tile	Indica el título a mostrarse en la barra de título del diálogo.
ValidateNames	Controla que el nombre del archivo no tenga caracteres inválidos.

### **Métodos**

Método	Descripción
OpenFile	Devuelve un Stream indicando el archivo abierto en el diálogo de abrir o grabado en el diálogo de guardar.
ShowDialog	Muestra el diálogo de archivo, sea de abrir o de guardar.

### **Eventos**

Evento	Descripción
FileOk	Ocurre al dar clic sobre el botón OK del diálogo de archivo.

## Control **FontDialog** FontDialog

Este control se usa para mostrar el diálogo de fuente y poder acceder a sus características como tipo de fuente, tamaños, estilos, efectos, etc.

### **Propiedades**

Propiedad	Descripción
Name	Generalmente usaremos el prefijo fdg.
AllowScriptChange	Controla si el conjunto de caracteres de fuente puede ser cambiado.
Color	Devuelve el color de fuente seleccionado en el diálogo.
Font	Determina la fuente seleccionada en el diálogo. Es un objeto.
FontMustExist	Indica si se mostrará un reporte de error al no existir una fuente.
MaxSize	Máximo tamaño de la fuente en puntos.
MinSize	Mínimo tamaño de la fuente en puntos.
ScriptsOnly	Controla si excluirá los caracteres OEM y símbolos.
ShowApply	Determina si se verá el botón de Aplicar en el diálogo.
ShowColor	Indica si se mostrará el color elegido del diálogo.
ShowEffects	Muestra el cuadro de efectos que trae: subrayado, tachado y color.
ShowHelp	Visualiza o no el botón de Ayuda en el diálogo.

### **Métodos**

Método	Descripción
ShowDialog	Muestra el diálogo de fuente.

### **Eventos**

Evento	Descripción
Apply	Ocurre al dar clic sobre el botón de aplicar del diálogo de fuente.

## Control ColorDialog ColorDialog

Este control se usa para mostrar el diálogo de colores y poder acceder a sus características como seleccionar un color sólido o personalizado.

### **Propiedades**

Propiedad	Descripción
Name	Generalmente usaremos el prefijo cdg.
AllowFullOpen	Habilita o no el botón de personalizar colores.
AnyColor	Controla si cualquier color puede ser seleccionado.
Color	Indica el color seleccionado en el diálogo.
FullOpen	Determina si la sección de colores personalizados será inicialmente vista.
ShowHelp	Visualiza o no el botón de Ayuda en el diálogo.
SolidColorOnly	Controla si solo los colores sólidos pueden ser seleccionados.

### **Métodos**

Método	Descripción
ShowDialog	Muestra el diálogo de colores.

### **Eventos**


Evento	Descripción
HelpRequested	Ocurre al dar clic sobre el botón de ayuda del diálogo de color.

### Barras

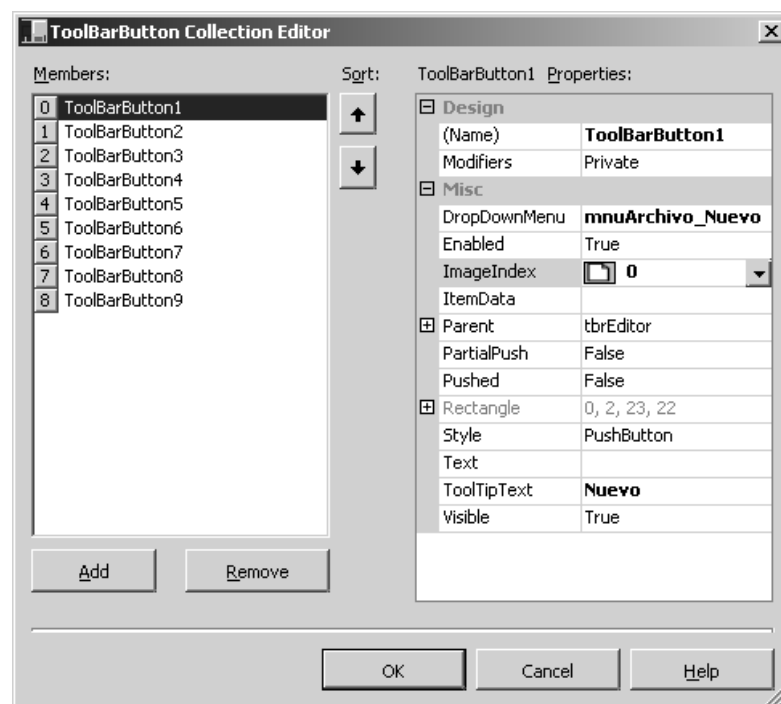
Las Barras son muy importantes en una aplicación ya que permiten mostrar algunos accesos directos, mostrar el estado en que se encuentra la aplicación, etc.

#### Control **ToolBar** ToolBar

Sirve para crear una Barra de Herramientas, para lo cual se realizan los siguientes pasos:


- Llenar un ImageList con las imágenes que se usaran en el ToolBar.
- Dar doble clic al control ToolBar y configurar la propiedad "Name" escribiendo el prefijo tbr seguido del nombre. Luego configurar la propiedad "ImageList" eligiendo el nombre de la Lista de Imágenes.
- Seleccionar la propiedad "Buttons" que es una colección y pulsar sobre el botón , aparecerá el dialogo "ToolBarButton Collection Editor".
- Para crear un botón dar clic en el botón "Add", luego modificar las propiedades "ImageIndex" para indicar el índice de la imagen a mostrar, opcionalmente en "ToolTipText" escribir un comentario y en "Text" escribir un título.
- Finalmente programar en el evento "ButtonClick" las acciones de los botones.

**Figura 3.6: Editor de la Colección de Botones del ToolBar**

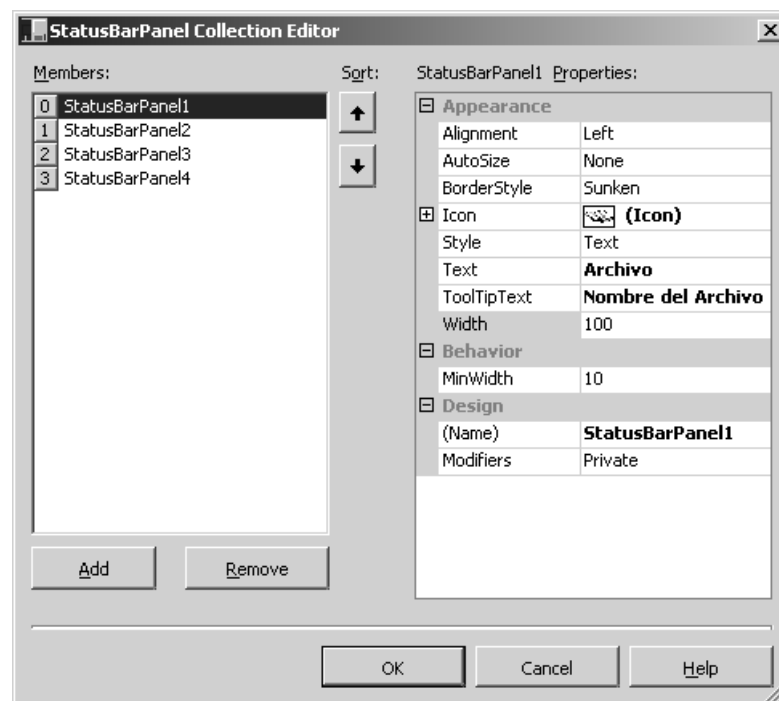


## Control StatusBar

Se usa para crear una Barra de Estado, para lo cual se realizan los siguientes pasos:

- Dar doble clic al control StatusBar y configurar la propiedad "Name" escribiendo el prefijo sbr seguido del nombre.
- Configurar la propiedad "ShowPanels" en true para poder ver los paneles.
- Seleccionar la propiedad "Panels" que es una colección y pulsar sobre el botón , aparecerá el dialogo "StatusBarPanels Collection Editor".
- Para crear un panel dar clic en el botón "Add", luego modificar sus propiedades, tales como: "Alignment", "BorderStyle", "Icon", "Style", "Text", "ToolTipText" y "Width"
- Si se desea realizar alguna tarea al dar clic a un cierto panel, programar en el evento "PanelClick" las acciones que realizará el panel.

**Figura 3.7: Editor de la Colección de Paneles del StatusBar**



**Nota:** En esta versión de Visual Basic los paneles no traen una propiedad que permita mostrar automáticamente el estado de las teclas CapsLock, NumLock, o que muestre la fecha u hora, etc.

### Laboratorio 3:

El presente laboratorio tiene como objetivo trabajar con aplicaciones para Windows, usando controles “Windows Forms” y añadiendo características como barras de herramientas, barras de estado, diálogos, etc. Este laboratorio está dividido en 2 ejercicios que tienen una duración aproximada de **50 minutos**.

#### Ejercicio 1: Usando Controles para Windows

**Duración: 20 minutos**

- 🖥 Elegir un nuevo proyecto “Visual Basic” y una plantilla de “Aplicación Windows”, seleccionar en ubicación la carpeta “C:\VBNET\Labs” y como nombre escribir “Lab04\_1”.
- 🖥 Vamos a crear una aplicación de tipo Proforma, que calcule el precio a pagar por un Computador eligiendo las partes básicas y opcionalmente sus accesorios; para esto, el nombre del formulario será “frmProforma.vb”.
- 🖥 Realizar el diseño del formulario añadiendo un “TabControl” con 3 “TabPage” y diseñar las fichas, tal como se muestran en la figuras de abajo:

**Figura 3.8: Diseño de la primera ficha de frmProforma**

The screenshot shows a Windows application window titled "Proforma de Venta de PCs". It features a tab control with three tabs: "Configuracion Básica", "Accesorios", and "General". The "Configuracion Básica" tab is currently selected. The form is designed on a grid background and contains the following controls:

- Tipo de Procesador:** A dropdown menu.
- Precio Procesador:** A text box for entering the price.
- Memoria:** A text box labeled "IstMemoria".
- Disco Duro:** A text box labeled "IstDisco".
- Precio Memoria:** A text box for entering the price.
- Precio Disco Duro:** A text box for entering the price.
- Tipo de Monitor:** A group box containing two radio buttons: "14" a Color" and "15" a Color".
- Tipo de Teclado:** A group box containing two radio buttons: "Simple" and "De Lujo".
- Precio Monitor:** A text box for entering the price.
- Precio Teclado:** A text box for entering the price.



**Figura 3.9: Diseño de la segunda ficha de frmProforma**

Proforma de Venta de PCs

Configuracion Básica   Accesorios   General

☐ Lectora de Disco

Tipo de Lectora:

☐ Modelo 1

☐ Modelo 2

☐ Modelo 3

Precio Lectora Disco:

☐ Lectora de CD ROM

Tipo de Lectora:

☐ 40X

☐ 60X

☐ 80X

Precio Lectora CD:

☐ Impresora

lstImpresora

Precio Impresora:

☐ Scanner

lstScanner

Precio Scanner:

**Nota:** Los "GroupBox" de lectoras y los "ListBox" están deshabilitados.

**Figura 3.10: Diseño de la tercera ficha de frmProforma**

**Proforma de Venta de PCs**

Configuración Básica | Accesorios | General

Nombre del Cliente:  Telefono:

Dirección:


Precio Total de Configuración Básica:


Precio Total de Accesorios:


Precio Total de Venta:


Total I.G.V. (18%):

Precio Total A Pagar:


 Calcular

 Nuevo


 Salir

-  Luego de realizar el diseño y configurar las propiedades de los controles; lo primero que haremos es escribir una rutina que permita llenar el combo y las listas, de la siguiente forma:

```
Public Sub Llenar_Listas()  
    With cboProcesador.Items  
        .Add("Pentium II 350 Mhz")  
        .Add("Pentium III 400 Mhz")  
        .Add("Pentium III 500 Mhz")  
        .Add("Pentium III 700 Mhz")  
    End With  
    With lstMemoria.Items  
        .Add("32 Mb")  
        .Add("64 Mb")  
        .Add("128 Mb")  
    End With  
    With lstDisco.Items  
        .Add("10 Gb")  
        .Add("20 Gb")  
        .Add("30 Gb")  
    End With  
    With lstImpresora.Items  
        .Add("Stylus Color 400")  
        .Add("Stylus Color 500")  
        .Add("Stylus Color 700")  
    End With  
    With lstScanner.Items  
        .Add("Pequeño")  
        .Add("Mediano")  
        .Add("Grande")  
    End With  
End Sub
```


-  A continuación, se debe llamar a la rutina después que se crea el formulario:

```
Public Sub frmProforma_Load(...)  
    Llenar_Listas()  
End Sub
```


-  Se debe programar el combo y las listas de la primera ficha "Configuración Básica" para que al elegir un tipo de procesador, memoria o disco duro, se muestre el precio respectivo:

```
Public Sub cboProcesador_SelectedIndexChanged(ByVal sender As ...)  
    Select Case cboProcesador.SelectedIndex  
        Case 0  
            txtPrecioProcesador.Text = "100"  
        Case 1  
            txtPrecioProcesador.Text = "150"  
        Case 2
```

```
        txtPrecioProcesador.Text = "200"  
    Case 3  
        txtPrecioProcesador.Text = "300"  
End Select  
End Sub  
  
Public Sub lstMemoria_SelectedIndexChanged(ByVal sender As ...)  
    Select Case lstMemoria.SelectedIndex  
    Case 0  
        txtPrecioMemoria.Text = "30"  
    Case 1  
        txtPrecioMemoria.Text = "50"  
    Case 2  
        txtPrecioMemoria.Text = "70"  
    End Select  
End Sub  
  
Public Sub lstDisco_SelectedIndexChanged(ByVal sender As ...)  
    Select Case lstDisco.SelectedIndex  
    Case 0  
        txtPrecioDisco.Text = "80"  
    Case 1  
        txtPrecioDisco.Text = "100"  
    Case 2  
        txtPrecioDisco.Text = "120"  
    End Select  
End Sub
```

 También se debe realizar lo mismo al elegir una opción del grupo de botones, tanto del monitor como del teclado:

```
Public Sub rbnMonitor14_Click(ByVal sender As Object, ...)  
    txtPrecioMonitor.Text = "150"  
End Sub  
  
Public Sub rbnMonitor15_Click(ByVal sender As Object, ...)  
    txtPrecioMonitor.Text = "200"  
End Sub  
  
Public Sub rbnTecladoSimple_Click(ByVal sender As Object, ...)  
    txtPrecioTeclado.Text = "15"  
End Sub  
  
Public Sub rbnTecladoLujo_Click(ByVal sender As Object, ...)  
    txtPrecioTeclado.Text = "30"  
End Sub
```

-  Programamos la segunda ficha de “Accesorios”, iniciando por la casilla de lectora de disco, para que al elegir este accesorio se habilite sus opciones (ya que las deshabilitamos en diseño) y al elegir un modelo se muestre su precio:

```
Public Sub chkLectoraDisco_Click(ByVal sender As Object, ...)
    gbxDLectoraDisco.Enabled = chkLectoraDisco.Checked
End Sub
```

```
Public Sub btnLectoraDisco1_Click(ByVal sender As Object, ...)
    txtPrecioLectoraDisco.Text = "20"
End Sub
```

```
Public Sub btnLectoraDisco2_Click(ByVal sender As Object, ...)
    txtPrecioLectoraDisco.Text = "40"
End Sub
```

```
Public Sub btnLectoraDisco3_Click(ByVal sender As Object, ...)
    txtPrecioLectoraDisco.Text = "50"
End Sub
```

-  De manera similar, lo hacemos con la lectora de CD ROM:

```
Public Sub chkLectoraCD_Click(ByVal sender As Object, ...)
    gbxDLectoraCD.Enabled = chkLectoraCD.Checked
End Sub
```

```
Public Sub btnLectoraCD40X_Click(ByVal sender As Object, ...)
    txtPrecioLectoraCD.Text = "40"
End Sub
```

```
Public Sub btnLectoraCD60X_Click(ByVal sender As Object, ...)
    txtPrecioLectoraCD.Text = "50"
End Sub
```

```
Public Sub btnLectoraCD80X_Click(ByVal sender As Object, ...)
    txtPrecioLectoraCD.Text = "70"
End Sub
```

-  Para la opción de Impresoras y Scanner se realiza el mismo procedimiento:

```
Public Sub chkImpresora_Click(ByVal sender As Object, ...)
    lstImpresora.Enabled = chkImpresora.Checked
End Sub
```

```
Public Sub lstImpresora_SelectedIndexChanged(ByVal sender As ...)
    lstImpresora.SelectedIndexChanged
    Select Case lstImpresora.SelectedIndex
        Case 0
```


```

        txtPrecioImpresora.Text = "100"
    Case 1
        txtPrecioImpresora.Text = "200"
    Case 2
        txtPrecioImpresora.Text = "300"
End Select
End Sub

Public Sub chkScanner_Click(ByVal sender As Object, ...)
    lstScanner.Enabled = chkScanner.Checked
End Sub

Public Sub lstScanner_SelectedIndexChanged(ByVal sender As ...)
    Select Case lstScanner.SelectedIndex
    Case 0
        txtPrecioScanner.Text = "100"
    Case 1
        txtPrecioScanner.Text = "200"
    Case 2
        txtPrecioScanner.Text = "400"
    End Select
End Sub

```

-  Finalmente, programamos los botones de la tercera ficha "General", que calculen el precio a pagar, limpie los datos ingresados y finalice la aplicación respectivamente:

```

Public Sub btnCalcular_Click(ByVal sender As Object, ...)
    Dim PrePro, PreMem, PreDis, PreMon, PreTec As Integer
    Dim PreLDi, PreLCD, PreImp, PreSca As Integer
    Dim TotBas, TotAcc, TotVen, TotIGV, TotPag As Single
    PrePro = txtPrecioProcesador.Text.ToInt16
    PreMem = txtPrecioMemoria.Text.ToInt16
    PreDis = txtPrecioDisco.Text.ToInt16
    PreMon = txtPrecioMonitor.Text.ToInt16
    PreTec = txtPrecioTeclado.Text.ToInt16
    PreLDi = txtPrecioLectoraDisco.Text.ToInt16
    PreLCD = txtPrecioLectoraCD.Text.ToInt16
    PreImp = txtPrecioImpresora.Text.ToInt16
    PreSca = txtPrecioScanner.Text.ToInt16
    TotBas = PrePro + PreMem + PreDis + PreMon + PreTec
    TotAcc = PreLDi + PreLCD + PreImp + PreSca
    TotVen = TotBas + TotAcc
    TotIGV = (0.18 * TotVen).ToSingle
    TotPag = TotVen + TotIGV
    txtTotalBasico.Text = TotBas.ToString
    txtTotalAccesorios.Text = TotAcc.ToString
    txtTotalVenta.Text = TotVen.ToString
    txtTotalIGV.Text = TotIGV.ToString
    txtTotalPagar.Text = TotPag.ToString

```





End Sub

```
Public Sub btnNuevo_Click(ByVal sender As Object, ...)
    Dim X As Control
    For Each X In Controls
        If ((TypeOf X Is TextBox) Or (TypeOf X Is ComboBox)) Then
            X.Text = ""
        End If
    Next
End Sub
```

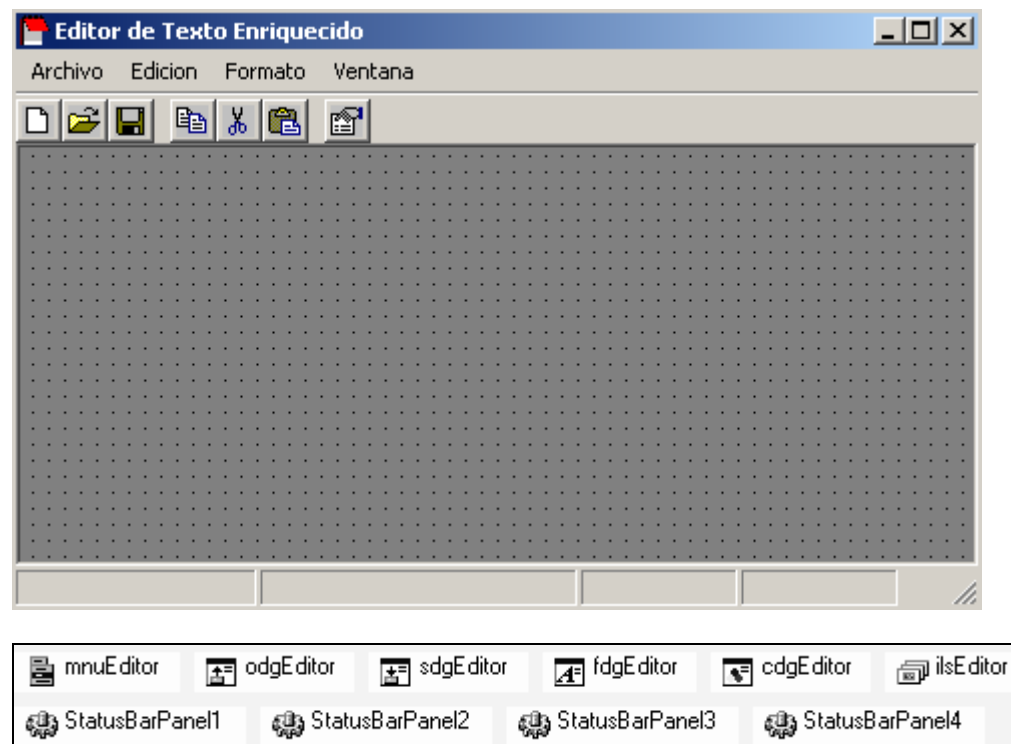
```
Public Sub btnSalir_Click(ByVal sender As Object, ...)
    End
End Sub
```

## Ejercicio 2: Creando aplicaciones MDI con Menús, Diálogos y Barras

**Duración: 30 minutos**

-  Estando en Visual Studio .NET, elegir un nuevo proyecto de “Visual Basic” y una plantilla de “Aplicación Windows”, seleccionar como ubicación la carpeta “C:\VBNET\Labs” y como nombre ingresar “Lab04\_2”.
-  Se va a crear un editor de texto enriquecido, es decir un programa que lea y recupere archivos RTF para lo cual necesitamos un formulario principal de tipo MDI y un formulario hijo para mostrar los documentos RTF.
-  En la ventana del “Solution Explorer” seleccionar el nombre del formulario y en la ventana de propiedades escribir en “FileName” el nombre “mdiEditor.vb”, para que este formulario sea MDI configurar su propiedad “IsMDIContainer” en “True”.
-  Realizar el diseño del formulario “mdiEditor” que contenga un menú de 4 opciones, una barra de herramientas con 9 botones (2 separadores) y una barra de estado con 4 paneles, tal como se muestra en la siguiente figura:

**Figura 3.11: Diseño del formulario mdiEditor**

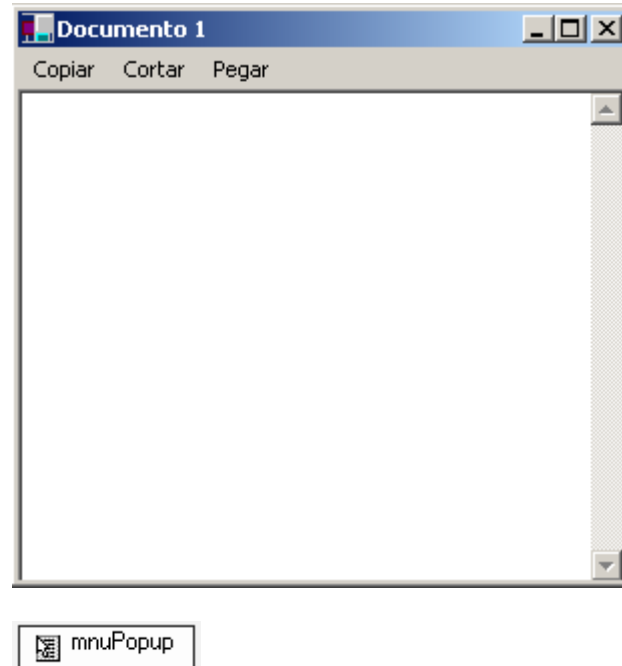


- 📁 También necesitamos añadir diálogos de abrir, guardar, fuente y colores, y una lista de imágenes para almacenar las figuras de los botones de la barra de herramientas.



- Seguidamente, añadir otro formulario colocándole como nombre “frmDocumento.vb” y luego añadirle un menú contextual llamado “mnuPopup” con 3 opciones (Copiar, Cortar y Pegar) y un RichTextBox de nombre “rtbEditor” con la propiedad “Anchor” en “All” para que se ajuste al tamaño de la ventana al cambiar de tamaño.

**Figura 3.12: Diseño del formulario frmDocumento**



- Proceder a insertar un módulo estándar, donde se defina una variable que almacene el número de documentos abiertos, tal como sigue:

```
Module Modulo
    Public N As Byte
End Module
```

- Ingresar al código del primer formulario “mdiEditor” y escribir las siguientes rutinas de validación:

```
Public Sub Validar_Menus_Boton(ByVal Habilitado As Boolean)
    mnuArchivo_Guardar.Enabled = Habilitado
    mnuEdicion.Enabled = Habilitado
    mnuFormato.Enabled = Habilitado
    mnuVentana.Enabled = Habilitado
    ToolBarButton3.Enabled = Habilitado
End Sub
```

```
Public Sub Validar_Copiar_Cortar(ByVal Habilitado As Boolean)
    mnuEdicion_Copiar.Enabled = Habilitado
    mnuEdicion_Cortar.Enabled = Habilitado
```

```


    ToolBarButton5.Enabled = Habilitado
    ToolBarButton6.Enabled = Habilitado
End Sub

Public Sub Validar_Pegar(ByVal Habilitado As Boolean)
    mnuEdicion_Pegar.Enabled = Habilitado
    ToolBarButton7.Enabled = Habilitado
End Sub

Public Sub Iniciar_Editor()
    Validar_Menus_Boton(False)
    Validar_Copiar_Cortar(False)
    Validar_Pegar(False)
End Sub

Public Sub Cambiar_Nombre(ByVal Nombre As String)
    Me.ActiveMDIChild.Text = Nombre
    StatusBarPanel1.Text = Nombre
End Sub


```

-  Programar en el inicio del formulario la configuración inicial del editor, tal como se muestra a continuación:

```

Public Sub frmEditor_Load(...)
    StatusBarPanel3.Text = "0 Docs"
    StatusBarPanel3.ToolTipText = "Numero de Documentos Abiertos"
    StatusBarPanel4.Text = Date.Now.ToLongTimeString
    StatusBarPanel4.ToolTipText = Date.Today.ToLongDateString
    Iniciar_Editor()
End Sub

```

-  Escribir el siguiente código para las opciones del menú "Archivo":

```

Public Sub mnuArchivo_Nuevo_Click(ByVal sender As Object, ...)
    Dim X As New Form2()
    N += 1
    X.MDIParent = Form1
    X.Text = "Documento " & N.ToString
    X.Show()
    Validar_Menus_Boton(True)
End Sub

Public Sub mnuArchivo_Abrir_Click(ByVal sender As Object, ...)
    With odgEditor
        .Title = "Abrir archivo de texto enriquecido"
        .Filter = "Archivo de texto enriquecido (*.rtf)|*.rtf"
        If .ShowDialog()=DialogResult.OK And .FileName<>"" Then
            If Me.ActiveMDIChild = Nothing Then _

```

```


                                mnuArchivo_Nuevo.PerformClick()
                Dim X As RichTextBox = _
                    Me.ActiveMDIChild.ActiveControl()
                X.LoadFile(.FileName)
                Cambiar_Nombre(.FileName)
            End If
        End With
    End Sub

    Public Sub mnuArchivo_Guardar_Click(ByVal sender As Object, ...)
        With sdgEditor
            .Title = "Guardar archivo de texto enriquecido"
            .Filter = "Archivo de texto enriquecido (*.rtf)|*.rtf"
            If .ShowDialog()=DialogResult.OK And .FileName<>"" Then
                Dim X As RichTextBox = _
                    Me.ActiveMDIChild.ActiveControl()

                X.SaveFile(.FileName)
                Cambiar_Nombre(.FileName)
            End If
        End With
    End Sub

    Public Sub mnuArchivo_Salir_Click(ByVal sender As Object, ...)
        End
    End Sub

```

 Programar las opciones del menú “Edición”, tal como sigue:


```

    Public Sub mnuEdicion_Copiar_Click(ByVal sender As Object, ...)
        Dim X As RichTextBox = Me.ActiveMDIChild.ActiveControl()
        Clipboard.SetDataObject(X.SelectedText)
    End Sub

    Public Sub mnuEdicion_Cortar_Click(ByVal sender As Object, ...)
        Dim X As RichTextBox = Me.ActiveMDIChild.ActiveControl()
        Clipboard.SetDataObject(X.SelectedText)
        X.SelectedText = ""
    End Sub


    Public Sub mnuEdicion_Pegar_Click(ByVal sender As Object, ...)
        Dim X As RichTextBox = Me.ActiveMDIChild.ActiveControl()
        Dim iData As IDataObject = Clipboard.GetDataObject()
        If iData.GetDataPresent(DataFormats.RTF) Then
            X.SelectedText = iData.GetData(DataFormats.Text)
        End If
    End Sub

```

 A continuación, escribir código para las opciones del menú “Formato”:

```
Public Sub mnuFormato_Fondo_Click(ByVal sender As Object, ...)
    Dim X As RichTextBox = Me.ActiveMDIChild.ActiveControl()
    With cdgEditor
        If .ShowDialog() = DialogResult.OK Then
            X.BackColor = .Color
        End If
    End With
End Sub
```

```
Public Sub mnuFormato_Fuente_Click(ByVal sender As Object, ...)
    Dim X As RichTextBox = Me.ActiveMDIChild.ActiveControl()
    With fdgEditor
        .ShowColor = True
        If .ShowDialog() = DialogResult.OK Then
            X.ForeColor = .Color
        End If
    End With
End Sub
```

 Para organizar la presentación de las ventanas hijas, escribir código para las opciones del menú "Ventana":

```
Public Sub mnuVentana_Cascada_Click(ByVal sender As Object, ...)
    Form1.LayoutMDI(MDILayout.Cascade)
End Sub
```

```
Public Sub mnuVentana_MosaicoV_Click(ByVal sender As Object, ...)
    Form1.LayoutMDI(MDILayout.TileVertical)
End Sub
```


```
Public Sub mnuVentana_MosaicoH_Click(ByVal sender As Object, ...)
    Form1.LayoutMDI(MDILayout.TileHorizontal)
End Sub
```

```
Public Sub mnuVentana_OrganizarI_Click(ByVal sender As Object, ...)
    Form1.LayoutMDI(MDILayout.Cascade)
End Sub
```


 Una vez escrito el código para los menús, hacerlo para los botones de la barra de herramientas:

```
Public Sub tbrEditor_ButtonClick(ByVal sender As Object, ...)
    Select Case e.button.ImageIndex
        Case 0
            mnuArchivo_Nuevo.PerformClick()
        Case 1
            mnuArchivo_Abrir.PerformClick()
        Case 2
```


```
        mnuArchivo_Guardar.PerformClick()  
Case 3  
        mnuEdicion_Copiar.PerformClick()  
Case 4  
        mnuEdicion_Cortar.PerformClick()  
Case 5  
        mnuEdicion_Pegar.PerformClick()  
Case 6  
        End  
End Select  
End Sub
```

-  Para que cada vez que se activa una ventana de documento, se vea el nombre del documento en el primer panel de la barra de estado, escribir lo siguiente:

```
Public Sub Form1_MDICHildActivate(ByVal sender As Object, ...)  
    StatusBarPanel1.Text = Me.ActiveMDICHild.Text  
End Sub
```

-  Ingresar al formulario "frmDocumento" y escribir código para que cada vez que se cierre una ventana de documento se disminuya la variable de contador y se verifique que si no hay ventanas abiertas se inicialice el editor:

```
Public Sub Form2_Closed(ByVal sender As Object, ...)  
    N -= 1  
    If N = 0 Then  
        Dim X As Form1 = Me.MDIParent  
        X.Iniciar_Editor()  
    End If  
End Sub
```

-  Para habilitar las opciones del menú y los botones de "Edición", escribimos:

```
Public Sub rtbEditor_SelectionChange(ByVal sender As Object, ...)  
    Dim X As Form1 = Me.MDIParent  
    X.Validar_Copiar_Cortar(rtbEditor.SelectedText <> "")  
    X.Validar_Pegar(Clipboard.GetDataObject <> Nothing)  
    mnuCopiar.Enabled = rtbEditor.SelectedText <> ""  
    mnuCortar.Enabled = rtbEditor.SelectedText <> ""  
    mnuPegar.Enabled = Clipboard.GetDataObject <> Nothing  
End Sub
```

-  Finalmente, programamos las opciones del menú contextual de edición:

```
Public Sub mnuCopiar_Click(ByVal sender As Object, ...)  
    Clipboard.SetDataObject(rtbEditor.SelectedText)  
End Sub
```

```
Public Sub mnuCortar_Click(ByVal sender As Object, ...)
    Clipboard.SetDataObject(rtbEditor.SelectedText)
    rtbEditor.SelectedText = ""
End Sub

Public Sub mnuPegar_Click(ByVal sender As Object, ...)
    Dim iData As IDataObject = Clipboard.GetDataObject()
    If iData.GetDataPresent(DataFormats.RTF) Then
        rtbEditor.SelectedText = iData.GetData(DataFormats.Text)
    End If
End Sub
```

# Módulo 4

# Creando

# Componentes .NET

---

Visual Basic .NET tiene todas las características de la Programación Orientada a Objetos (POO), ya que el Marco .NET soporta los requerimientos para que todos los lenguajes que trabajen con él usen POO. Estas características son:

## 1. Encapsulación

Cuando creamos un componente .NET este se encuentra encapsulado, ya que oculta la lógica de programación a los usuarios que lo utilizan, permitiendo manejar dicho objeto a través de sus miembros, tales como propiedades y métodos, realizando el desarrollo de aplicaciones mas simple, al ocultar la complejidad del código (encapsular).

## 2. Herencia

La herencia es la característica en la cual una clase llamada "Clase Base" pasa o hereda todas sus características a otra llamada "Clase Derivada", permitiendo la total reusabilidad del código escrito en las aplicaciones. La herencia de clases es una nueva característica de Visual Basic .NET y no solo es a nivel de clases creadas en éste lenguaje sino a través de cualquier lenguaje del Marco .NET.

## 3. Polimorfismo

Otra característica interesante de la POO es el polimorfismo, en el caso de Visual Basic .NET éste se puede crear cuando en una clase derivada se implementa de manera distinta un método heredado de la clase base. Es decir, podemos tener un mismo método con dos comportamientos distintos (códigos distintos) de acuerdo al tipo de objeto, que puede ser creado de la clase base o de la derivada.

Como vemos las nuevas características de la Programación Orientación a Objetos (POO) mezcladas con la facilidad de uso de la Programación Orientada a Componentes (POC) dan como resultado la creación de aplicaciones poderosas y con un bajo costo de mantenimiento.

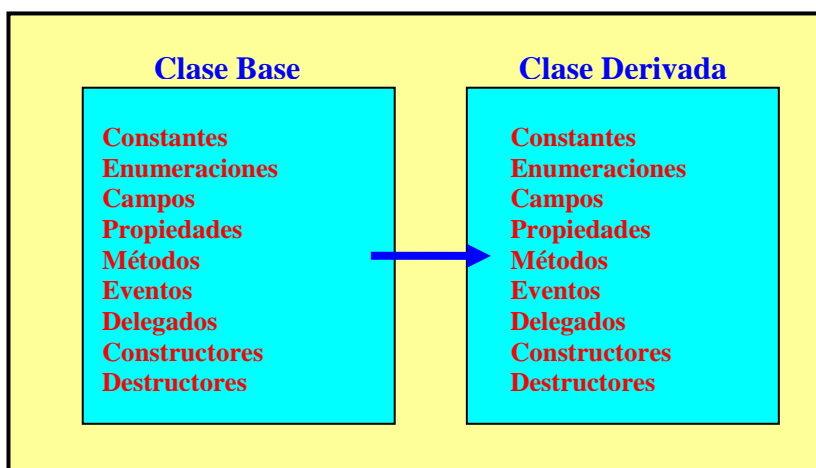
Sin duda, la reusabilidad y encapsulación ofrecida por la tecnología COM basada en componentes se ve incrementada por la herencia de clases y el polimorfismo ofrecida por la tecnología .NET orientada a objetos; lográndose una verdadera integración entre aplicaciones.

Para finalizar esta introducción a los componentes .NET, la otra gran ventaja con respecto a COM es la distribución de componentes, que en este último era una pesadilla debido había que registrar componentes y lidiar con los problemas de compatibilidad de versiones; en cambio con .NET los componentes no se registran y su distribución es automática con solo copiar y pegar la aplicación.

### ***Elementos de una Clase (Miembros)***

Todos los componentes están formados por clases y estas a su vez se componen de elementos o miembros, los cuales trataremos en este capítulo. Para ilustrar mejor tenemos el siguiente gráfico.

**Figura 4.1: Estructura de un Componente .NET**



A diferencia de Visual Basic 6 en donde las clases podían tener solo propiedades, métodos, eventos y constantes enumeradas; en Visual Basic .NET las clases pueden tener campos, delegados, constructores y destructores. Además pueda ser que una clase herede de otra que este dentro del componente o en otro componente .NET.



## Clase

Una clase es la plantilla para crear el objeto, es aquí donde se definen las partes del objeto: datos (propiedades, constantes, enumeraciones, campos) y procedimientos que operan sobre los datos (métodos).

La clase define un nuevo tipo de datos que resulta de la abstracción de algún elemento en la aplicación, por tanto, es necesario diseñar bien la aplicación antes de crear la clase, ya que esta solo implementa el diseño de objetos previamente realizado.

### Declaración de una Clase

A diferencia de Visual Basic 6, donde el tipo de clase estaba dado por la propiedad "Instancing" de la clase, que podía ser privada (Private), dependiente (PublicNotCreateTable) o pública (SingleUse, GlobalSingleUse, MultiUse, o GlobalMultiUse); en VB .NET no existe propiedades para el módulo de clase, ya que las características de la clase dependen de la forma de su declaración.

#### Sintaxis:

```
[Tipo de Declaración] Class <Nombre Clase>
    <Definición de miembros de la clase>
    <...>
End Class
```

Existen varias formas de declarar una clase, que detallamos a continuación en la siguiente tabla:

Declaración	Alcance o Ámbito
Public	Puede usarse en cualquier otra clase del componente o en las aplicaciones clientes.
Private	No puede usarse en otras clases del componente ni en las aplicaciones clientes.
Protected	Solo puede ser usada por las clases derivadas de éste, pero no por las aplicaciones clientes.
Friend	Solo puede ser usada por las otras clases del componente, pero no por las aplicaciones clientes.
Protected Friend	Es una combinación de ambas, es decir, la clase puede ser usada por otras clases del componente y por las clases derivadas.
Shadows	Indica que los miembros de la clase pueden ocultarse en la clase derivada, es decir que al heredar se ocultan ciertos miembros.
MustInherit	Determina que los miembros de la clase pueden heredarse a una clase derivada, pero no puede ser creada por aplicaciones clientes
NotInheritable	La clase no puede heredarse pero si instanciarse desde aplicaciones clientes.



### Ejemplo:

Si queremos crear una aplicación de Planilla necesitamos diseñar un componente que manipule información del empleado, sus horas trabajadas, tardanzas, faltas, etc.

Para crear la clase Empleado, de tal manera que pueda heredarse y utilizarse tanto dentro del componente como en las aplicaciones clientes, definiríamos la clase en Visual Basic, tal como sigue:

```
Public Class Empleado
```

```
End Class
```

## Constantes, Campos y Enumeraciones

### Constantes

Una constante es un dato que permanece fijo en toda la ejecución de la aplicación, a diferencia de la variable, cuyo valor va cambiándose en tiempo de ejecución.

Se utiliza para facilitar el mantenimiento del código, ya que si definimos un valor en una constante, si este cambia, con solo cambiar el valor de la constante, la aplicación trabajaría con el nuevo valor.

### Sintaxis:

```
[Public | Private] Const <Nombre> [As <Tipo Dato>]=<Valor>
```

### Ejemplo:

Podemos crear un par de constantes que almacenen el valor de las tasas de los impuestos a la renta e impuesto extraordinario de solidaridad, para propósitos del cálculo de descuentos.

```
Private Const TasalRenta As Single = 0.1  
Private Const TasalES As Single = 0.02
```

### Nota:

En Visual Basic .NET si no se define el tipo de dato de la constante se asume que es Object, ya no Variant como en Visual Basic 6, debido a que éste tipo de dato ya no existe.

### Campos

Un campo es una variable local para la clase, es decir, solo puede ser usada internamente por los miembros de la clase pero no por otras clases o por aplicaciones clientes.

El campo solo puede ser una variable privada o protegida que se use dentro de la clase para almacenar un valor usado por alguna propiedad o método, sin que éste pueda ser visto o instanciado por otras aplicaciones.

### Sintaxis:

```
[Private | Friend] <Nombre Campo> [As <Tipo Dato>]
```

### Ejemplo:

Podemos crear un par de campos que almacenen el cálculo de los impuestos a la renta e impuesto extraordinario de solidaridad, para el total de descuentos.

```
Private IES As Single  
Private IRenta As Single
```

### Enumeraciones

Una enumeración es una estructura de datos personalizada que define un conjunto de valores enteros. Por defecto, el primer dato enumerado empieza en 0, el segundo en 1 y así sucesivamente, aunque esto puede cambiarse al momento de definir la enumeración.

Las enumeraciones son útiles para dar claridad al programa y evitar errores de asignación de variables, ya que si definimos una variable de tipo enumerada, el valor que se puede asignar está limitado por el rango de valores definido en la enumeración.

### Sintaxis:

```
[Public | Private | Protected | Friend] Enum <Nombre Enumeración>  
    <Elemento1> [= <Valor1>]  
    <Elemento2> [= <Valor2>]  
    <Elemento3> [= <Valor3>]  
    <...>  
End Enum
```

### Ejemplo:

Vamos a crear dos enumeraciones, la primera que describa los cargos de los empleados que se usará desde otras clases y aplicaciones, y la segunda que defina las áreas de la empresa para propósitos internos de trabajar con información del empleado.

```
Public Enum Cargos  
    Auxiliar  
    Técnico  
    Ingeniero  
    Secretaria  
    Vendedor  
End Enum
```

```
Private Enum Areas  
    Gerencia = 100  
    Contabilidad = 200  
    Producción = 300  
    Sistemas = 400  
    Ventas = 500  
End Enum
```

Para usar estas enumeraciones creadas, tan solo basta declarar una variable de estos tipos y asignarle un valor definido, por ejemplo, para el caso anterior sería como sigue:

Dim Cargo As Cargos  
Cargo = Cargos.Vendedor  
Dim Area As Areas  
Area = Areas.Ventas

## Propiedades

Una propiedad es una característica del objeto que tiene información sobre un cierto atributo de éste como por ejemplo su nombre, descripción, etc. Las propiedades son métodos que se trabajan como si fuesen campos, con la diferencia que el campo es interno solo para uso de la clase, en cambio las propiedades se usan fuera de la clase.

A diferencia de la versión anterior de Visual Basic que existían 3 estructuras para declarar una propiedad (Property Get, Let, y Set), ahora, en Visual Basic .NET solo existe una estructura llamada Property donde internamente se define si la propiedad es de lectura (Get) y/o escritura (Set), tal como se muestra debajo.

### Sintaxis:

```
[Tipo de Declaración] Property <Nombre > [As <Tipo Dato>]
    [[ReadOnly] Get
        <Instrucciones>
    End Get]
    [[WriteOnly] Set (ByVal Value [As <Tipo Dato>])]
        <Instrucciones>
    End Set]
End Property
```

Existen muchas formas de declarar una propiedad, que explicamos en la sgte tabla:

Declaración	Uso
Default	Indica que la propiedad es por defecto (no es necesario escribirla)
Public	Puede ser usada desde cualquier clase o aplicación.
Private	Solo puede ser accesada desde dentro de la clase.
Protected	Se usa desde dentro de la clase o desde una clase derivada.
Friend	Puede ser usada desde otras clases del componente pero no fuera.
Protected Friend	Tiene las características de la protegida y amiga, es decir, se usa dentro de la clase, desde una clase derivada o desde otra clase del mismo componente, pero no en aplicaciones clientes.
ReadOnly	Indica que el valor de la propiedad solo puede ser recuperado pero no escrito. Solo puede tener la cláusula Get y no Set.
WriteOnly	Indica que el valor de la propiedad solo puede ser escrito pero no devuelto. Solo puede tener la cláusula Set y no Get.
Overloads	Permite que la propiedad de la clase base sea sobrecargada, es decir definida varias veces en las clases derivadas pero con diferentes parámetros que la

	propiedad definida en la clase base.
Overrides	Permite sobrescribir la propiedad por otra propiedad con el mismo nombre en una clase derivada.
Overridable	Permite sobrescribir la propiedad por un método con el mismo nombre en una clase derivada.
NotOverridable	Indica que la propiedad no puede ser sobrescrita por nadie en ninguna clase derivada.
MustOverride	Indica que la propiedad no puede ser implementada en esta clase y puede ser implementada en una clase derivada.
Shadows	Se usa para ocultar una propiedad de tal manera que no pueda implementarse en una clase derivada.
Shared	Permite que la propiedad sea compartida y pueda llamarse sin necesidad de instanciar a la clase sino a través de su nombre.

#### Ejemplo:

Para nuestra clase Empleado, podemos implementar las propiedades código, nombre y básico del empleado, de tal manera que se puedan leer y escribir en cualquier parte.

```

Private mvarCodigo As Integer
Private mvarNombre As String
Private mvarSueldo As Single

Public Property Codigo() As Integer
    Get
        Codigo = mvarCodigo
    End Get
    Set(ByVal Value As Integer)
        mvarCodigo = Value
    End Set
End Property

Public Property Nombre() As String
    Get
        Nombre = mvarNombre
    End Get
    Set(ByVal Value As String)
        mvarNombre = Value
    End Set
End Property

Public Property Sueldo() As Single

```

```
Get
    Sueldo = mvarSueldo
End Get
Set(ByVal Value As Single)
    mvarSueldo = Value
End Set
End Property
```

## Métodos

Un método es un conjunto de instrucciones que modifica el estado de las propiedades; en términos de objetos, un método es un servicio o función del objeto, mientras que en términos de código un método es un procedimiento o función que realiza una tarea específica.

En Visual Basic .NET todos los miembros de una clase (propiedades, eventos, constructores, destructores, etc.) son en realidad métodos; claro, estos últimos son métodos especiales.

Un procedimiento o función en una clase es un método, y sus características son las mismas que cuando se encuentran en un módulo estándar, es decir, si es procedimiento la declaración es con Sub, y si es función la declaración es con Function, tal como se muestra en la sintaxis de abajo.

### Sintaxis:

```
[Tipo Declaración] [Sub | Function] <Nombre >([Param]) [As <Tipo Dato>]
    <Instrucciones>
    [Exit [Sub | Function]]
End Property
```

En cuanto al tipo de declaración, es muy similar al de las propiedades, que describimos anteriormente en una tabla, es decir, puede declararse el método como Public, Private, Protected, Friend, Protected Friend y Shadows.

También hay que recordar que si el método es una función siempre se deberá devolver un valor, que se puede hacer de dos formas: de manera clásica asignando un valor al nombre de la función o de una nueva forma escribiendo la sentencia return y el valor.

### Ejemplo:

En nuestro ejemplo de Empleado, podemos implementar dos métodos públicos, uno que permita asignar valores a las propiedades creadas: código, nombre y sueldo del empleado y otro método que permita actualizar el sueldo.

```
Public Sub CrearEmpleado(ByVal vCodigo As Integer, _
    ByVal vNombre As String, ByVal vSueldo As Single)
    mvarCodigo = vCodigo
    mvarNombre = vNombre
    mvarSueldo = vSueldo
End Sub

Public Sub ActualizarSueldo(ByVal vNuevoSueldo As Single)
```



```
mvarSueldo = vNuevoSueldo  
End Sub
```

## Eventos

Un evento es un suceso que le ocurre al objeto y que le indica o notifica sobre un cambio en sus atributos o propiedades. Un evento es necesario para controlar el estado de las propiedades e informar a las aplicaciones del nuevo estado, para que estas realicen la acción correspondiente.

Es fácil entender eventos asociados a objetos visuales como los controles, por ejemplo en el objeto "Button" se tiene el evento "Click" que se desencadena al seleccionar el botón, el evento "MouseMove" que ocurre al pasar el mouse por el botón, y así hay muchos eventos asociados al objeto botón; pero, cuando trabajamos con clases, los eventos son más abstractos y un poco más difíciles de entender, ya que podía confundirse con los métodos.

Para aclarar el concepto de eventos definamos mentalmente una clase llamada "Cuenta" para un componente llamado "Banco". Esta clase cuenta tiene propiedades como "NumeroCta", "FechaApertura", "TipoCta", "NumeroTarjeta" y "Saldo". Además tiene métodos como "Depósito", "Retiro" y "Transferencia" que modifican el estado del saldo. Cuando hacemos un "Retiro" de nuestra cuenta donde el monto a retirar supera al "Saldo" se desencadena un evento llamado "SaldoInsuficiente" y también cuando durante el día hemos retirado más del monto permitido se desencadena el evento "LimiteRetiroDiario".

Para crear un evento en la clase primero declare el evento con la sentencia "Event" y luego llámelo con "RaiseEvent", tal como se muestra en la sintaxis.

### Declaración Sintaxis:

**[Tipo Declaración] Event** <Nombre> ([Parámetro(s)])

### Declaración Llamada:

**RaiseEvent** <Nombre> ([Parámetro(s)])

El tipo de declaración, es igual que la de los métodos, es decir, puede ser Public, Private, Protected, Friend, Protected Friend y Shadows.

### Ejemplo:

Crear dos eventos que nos informen cuando el sueldo es bajo o alto; para nuestra realidad un sueldo será bajo cuando es menor a 500 y alto cuando sea mayor a 3000.

```
Public Event SueldoBajo()
```

```
Public Event SueldoAlto()
```

```
Private Sub ChequearSueldo()
```

```
    If mvarSueldo < 500 Then
```

```
        RaiseEvent SueldoBajo()
```

```
    ElseIf mvarSueldo > 3000 Then
```

```
        RaiseEvent SueldoAlto()
```

```
    End If
```

```
End Sub
```

Para finalizar, deberíamos llamar a la rutina “ChequearSueldo” al final de “CrearEmpleado” y “ActualizarSueldo” que es donde se modifica el “Sueldo”.

## Constructores y Destructores

### Constructores

Un constructor es un método que se usa para inicializar características del objeto. Todas las clases de Visual Basic .NET tienen un constructor por defecto que es el método "New", pero se pueden agregar varios constructores "New" diferenciándose por el número de parámetros.

#### Sintaxis:

```
Public Sub New ()  
    <Instrucciones>  
End Sub  
Public Sub New ([<Param1> As <Tipo Dato>])  
    <Instrucciones>  
End Sub  
Public Sub New ([<Param1> As <Tipo Dato>, <Param2> As <Tipo Dato>])  
    <Instrucciones>  
End Sub  
Public Sub New ([<P1> As <Dato>, <P2> As <Dato>, ..., <PN> As <Dato>])  
    <Instrucciones>  
End Sub
```

#### Nota:

Es requisito indispensable escribir los métodos constructores al inicio de la clase, de lo contrario no funcionarían.

#### Ejemplo:

Continuando con nuestro ejemplo de la clase "Empleado" podemos crear 4 constructores que permitan crear de cuatro maneras distintas un objeto de tipo empleado.

```
Public Sub New()  
End Sub
```

```
Public Sub New(ByVal vCodigo As Integer)  
    mvarCodigo = vCodigo  
End Sub
```

```
Public Sub New(ByVal vCodigo As Integer, ByVal vNombre As String)  
    mvarCodigo = vCodigo  
    mvarNombre = vNombre  
End Sub
```

```
Public Sub New(ByVal vCodigo As Integer, ByVal vNombre As String,  
               ByVal vSueldo As Single)  
    mvarCodigo = vCodigo  
    mvarNombre = vNombre  
    mvarSueldo = vSueldo
```

End Sub

## ***Destructores***

Un destructor es un método que se usa para limpiar características del objeto antes que sea destruido y liberado de la memoria. Por ejemplo, si tenemos una clase que accede a datos y que abre una conexión hacia una base de datos para crear un Dataset, entonces recitamos un destructor que permita cerrar el Dataset y la conexión antes de destruir el objeto que apunta a dicha clase.

El Framework .NET solo provee destructores a Visual C#, en cambio, en Visual Basic .NET no existe un método especial para crear destructores; en vez de ello se crea un método cualquiera que realice la labor de iniciar variables, liberar recursos, etc.

### **Sintaxis:**

```
Public Sub <Nombre Destructor>()  
    <Instrucciones>  
End Sub
```

### **Ejemplo:**

Para la clase “Empleado” podemos crear un destructor llamado “Finalizar” que permita liberar las variables usadas por las propiedades para almacenar sus valores.

```
Public Sub Finalizar()  
    mvarCodigo = Nothing  
    mvarNombre = Nothing  
    mvarSueldo = Nothing  
End Sub
```

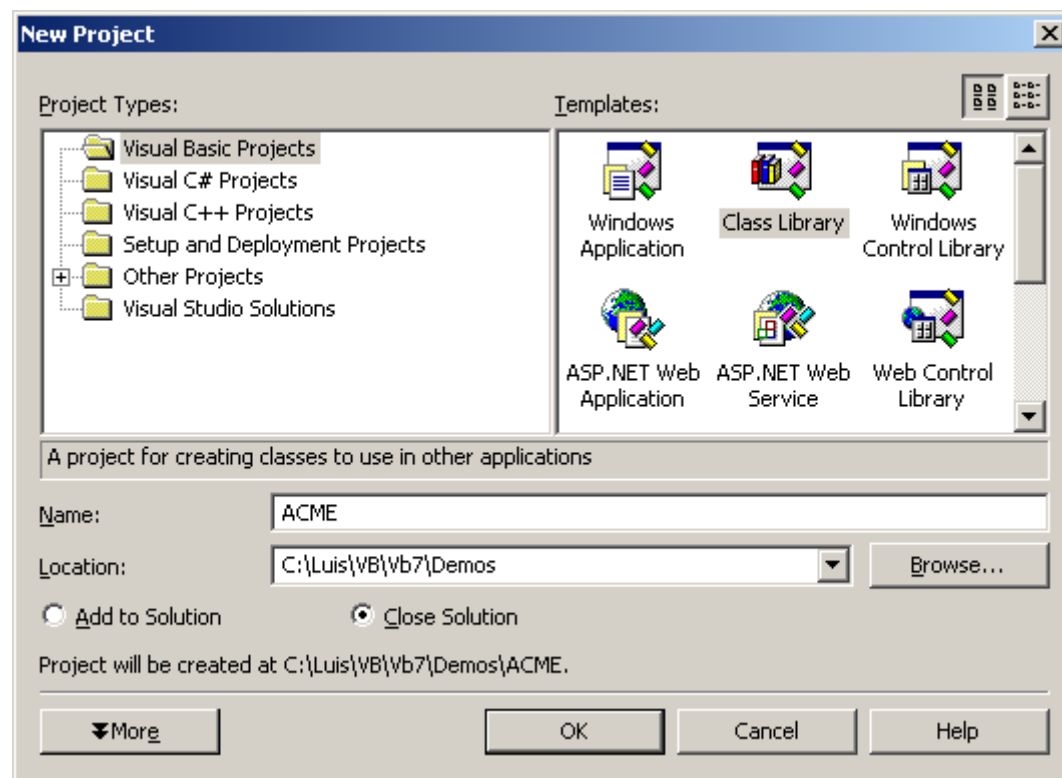
## ***Creando una Librería de Clases***

Después de ver los elementos de una clase, es momento de crear Componentes .NET los cuales también son conocidos como “Librería de Clases”, para lo cual existen varias etapas que detallamos a continuación.

### **Eligiendo el Tipo de Aplicación**

- Del menú “File”, elegir “New” y luego “Project” o pulsar [Ctrl + N]
- En la opción “Project Types” elegir “Visual Basic Projects” y en la opción “Templates” elegir “Class Library”

***Figura 4.2: Ventana para crear una Librería de Clases***



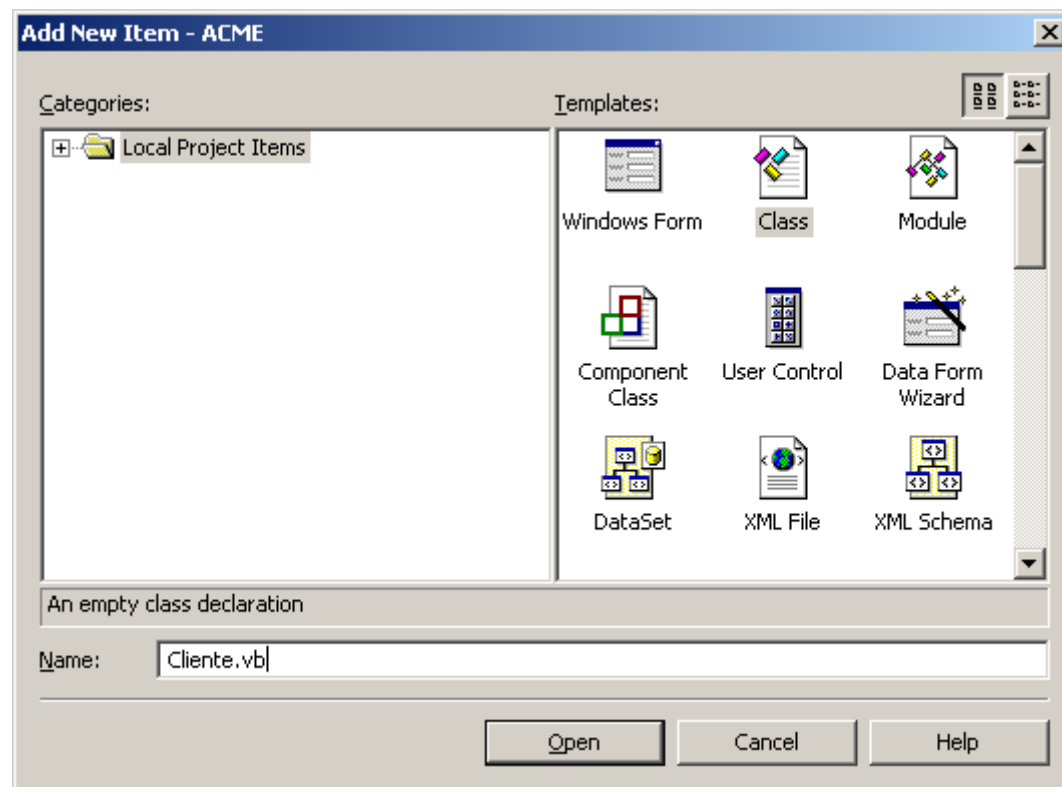
- Seleccionar la ubicación y escribir un nombre adecuado para el proyecto, generalmente, se acostumbra colocar el nombre de la Empresa, nosotros podemos llamar al componente “ACME” y luego “OK”.
- Inmediatamente aparecerá una clase llamada “Class1”, proceder a cambiar de nombre físico y lógico, por ejemplo ponerle como nombre “Empleado”.

### Añadiendo una Clase

Por defecto toda librería trae una clase, pero si queremos crear mas clase realizamos lo siguiente:

- Del menú “Project” elegimos la opción “Add Class”
- Escribimos el nombre físico de la clase, este a su vez será por defecto el nombre lógico y damos clic en “Open”.

**Figura 4.3: Ventana para crear una Clase**



1. Por defecto aparecerá la siguiente estructura:

```
Public Class Cliente
```

```
End Class
```

2. Dependiendo del alcance que queramos que tenga el objeto se puede modificar el tipo de declaración "Public" a una de las vistas anteriormente, tales como: "Private", "Protected", "Friend", "Shadows", etc.



### Creando Propiedades, Métodos, Eventos

La creación de propiedades, métodos y eventos se realiza dentro del módulo de clase tal como se mostró anteriormente. Por ejemplo para la clase "Empleado" podemos crear lo siguiente.

3. Añadiendo una propiedad:

```
Private mvarSueldo As Single
Public Property Sueldo() As Single
    Get
        Sueldo = mvarSueldo
    End Get
    Set(ByVal Value As Single)
        mvarSueldo = Value
    End Set
End Property
```

4. Añadiendo un método:

```
Public Sub ActualizarSueldo(ByVal vNuevoSueldo As Single)
    mvarSueldo = vNuevoSueldo
End Sub
```

5. Añadiendo eventos:

```
Public Event SueldoBajo()
Public Event SueldoAlto()
```

6. Llamando eventos:

```
Sub ChequearSueldo()
    If mvarSueldo < 500 Then
        RaiseEvent SueldoBajo()
    ElseIf mvarSueldo > 3000 Then
        RaiseEvent SueldoAlto()
    End If
End Sub

Public Sub ActualizarSueldo(ByVal vNuevoSueldo As Single)
    mvarSueldo = vNuevoSueldo
    ChequearSueldo()
End Sub
```

También podemos definir en el módulo de clase constantes, campos y enumeraciones; además, podemos añadir constructores y destructores para la clase, tal como vimos en la sección anterior.

## Probando y Usando la Librería de Clases

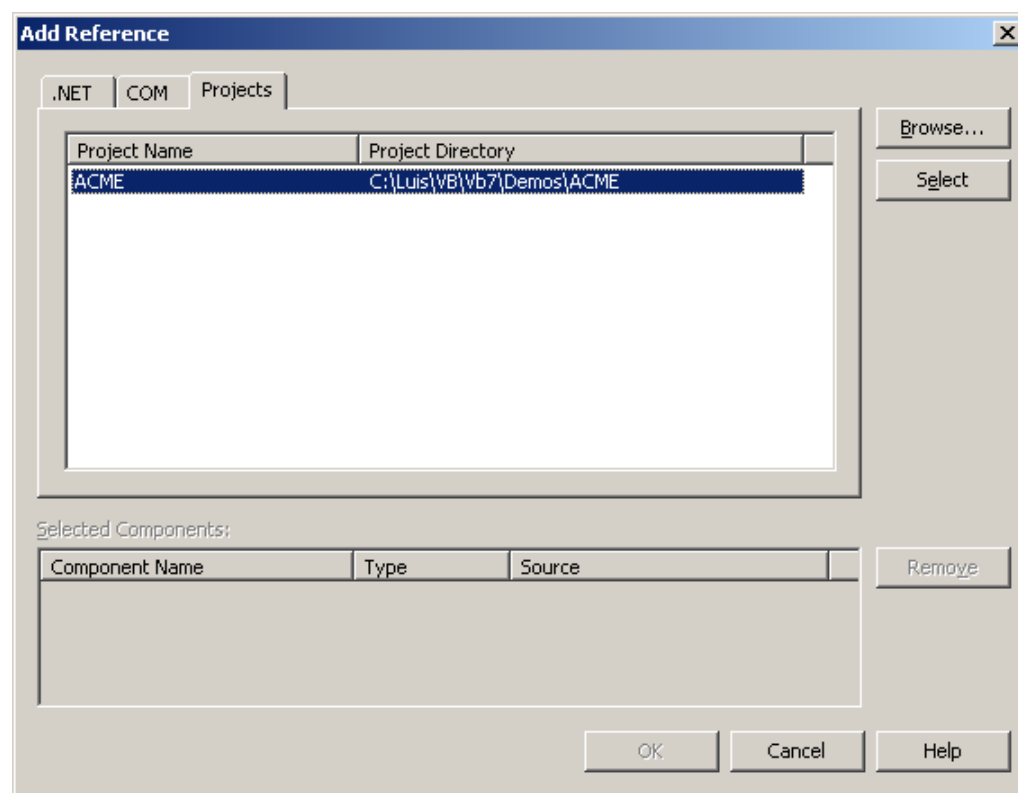
### Probar la Librería de Clases

Para probar el funcionamiento de la Librería de Clases es necesario crear una aplicación que use los miembros de las clases de la librería. Esta aplicación la llamaremos el "Cliente" del componente y puede estar escrito en cualquier lenguaje del Framework .NET, es decir, si creamos una librería en Visual Basic .NET, la podemos probar o usar en una aplicación Visual Basic .NET, Visual C#, Visual C++, COBOL, Pascal, C++, etc.

Vamos a asumir que se desea crear una aplicación cliente en Visual Basic .NET que pruebe la librería de clases creada, para esto necesitamos realizar lo siguiente:

7. Del menú "File", elegir "Add Project", y luego seleccionar "New Project". Elegir una plantilla de aplicación para Windows o Web en Visual Basic .NET
8. Elegir el directorio donde se creará la aplicación, escribir el nombre y "OK". Aparecerá un nuevo proyecto en la solución, el cual se ejecuta por defecto.
9. Para indicar que el proyecto de la aplicación es el que se ejecuta dar clic derecho sobre el nombre del proyecto en la ventana del "Solution Explorer" y elegir "Set as StartUp Project".

**Figura 4.4: Ventana para Añadir una Referencia a la Librería**



10. Para usar la librería desde la aplicación creada hacer una referencia a esta, desde el menú "Project" eligiendo "Add Reference..."
11. Elegir la ficha "Projects" de la ventana de añadir referencia y se mostrará la librería creada en la misma solución, tal como se muestra en la figura anterior.
12. Dar clic al botón "Select" y luego "OK"; inmediatamente, aparecerá en la carpeta "References" del proyecto en la ventana del "Solution Explorer".
13. Declarar una variable objeto que apunte a una clase del componente o librería, para lo cual existen 3 formas que ilustraremos con la clase empleado:
  1. Primera forma:  
`Dim objEmpleado As ACME.Empleado`  
`objEmpleado = New ACME.Empleado()`
  2. Segunda forma:  
`Dim objEmpleado As ACME.Empleado = New ACME.Empleado()`
  3. Tercera forma:  
`Dim objEmpleado As New ACME.Empleado`
14. Usar las propiedades y métodos del objeto, tal como se muestra en el ejemplo:  
`objEmpleado.CrearEmpleado(123, "Luis Dueñas", 1000)`
15. Si deseamos usar los eventos, entonces tendremos que declarar la variable en la sección de declaraciones generales, de la siguiente forma:  
  
`Private WithEvents objEmpleado As New ACME.Empleado()`
16. Para liberar la variable objeto realizar la siguiente asignación:  
`objEmpleado = Nothing`

### Nota:

La anterior sentencia en realidad no destruye la variable, sino que la desactiva, el encargado de destruir definitivamente al objeto es el "Garbage Collector".

### Usar la Librería de Clases

Anteriormente, habíamos visto como probar la librería junto con una aplicación en la misma solución, lo que permite realizar depuración paso a paso, pero cuando ya está probada la librería no es necesario tener el código fuente, sino tan solo el archivo DLL.

Para usar la librería en cualquier aplicación solo hay que hacer una referencia y en la opción "Projects" hay que seleccionar el nombre de la DLL que se encuentra en la carpeta del proyecto. Después se realizan los mismos pasos que para probarla.



## Herencia de Clases

### Introducción a la Herencia de Clases

La parte principal de la Programación Orientada a Objetos (POO) es la **herencia de clases**, es decir, la característica de definir una clase que sirva de base para otras **clases derivadas**, las clases derivadas tendrán los miembros de la **clase base**: propiedades, métodos, eventos, etc.

Los miembros heredados por la clase derivada pueden sobre escribirse e implementarse de otra forma, además la clase derivada puede tener sus propios miembros y servir de clase base para otras clases, lográndose la reutilización de objetos a través de la herencia.

Otra forma de herencia es a través del **polimorfismo**, que es una característica de la POO que consiste en definir una **clase abstracta** con propiedades y métodos que serán implementados de diferentes formas por otras clases, es decir, con un mismo nombre de propiedad o método se obtiene funcionalidad distinta de acuerdo al tipo de objeto.

En .NET solo existe **herencia simple** y no herencia múltiple, es decir, una clase derivada solo puede heredar de una clase base. Haciendo una analogía, si a la clase base le llamamos “padre” y a la clase derivada le llamamos “hijo” diríamos que la herencia simple consiste en que un “hijo” solo puede tener un solo “padre”, lo que parece algo natural y coherente.

Si deseamos simular **herencia múltiple** en Visual Basic .NET recurrimos a las **interfaces**, que permiten definir propiedades y métodos en una clase sin código, luego desde una clase se puede implementar varias interfaces, lográndose una herencia múltiple pero a nivel de definición y no de código, ya que la implementación será distinta en cada clase.

En general, la herencia de clases permite reusar código y facilitar el mantenimiento de las aplicaciones, ya que cuando se desea modificar características de un objeto solo hay que cambiar la clase adecuada.

Con .NET podemos implementar la herencia de cualquier clase pública de la librería de clases base, la cual tiene una gran cantidad de clases de diferentes tipos, tales como Windows, Data, XML, ASP .NET, System, etc. Pero, también podemos implementar herencia de clases creadas por nosotros, sin importar el lenguaje en que fueron creadas.

## Implementando Herencia en una Clase

Para crear una herencia de clases se usa la instrucción **Inherits** seguida de la clase base de donde se heredarán los miembros para la clase actual (clase derivada), tal como se muestra debajo.

### Sintaxis:

**Inherits** <Clase Base>

### Notas:

Dos observaciones importantes que hay que tener en cuenta son:

1. La instrucción Inherits debe escribirse en la primera línea de la clase derivada.
2. Solo puede existir una instrucción Inherits, ya que solo existe herencia simple.

### Ejemplo:

Podemos crear una clase llamada "Vendedor" que herede de la clase "Empleado" que habíamos creado anteriormente, y crear dos propiedades, una llamada "Venta" y otra llamada "Comision", tal como sigue:

```
Public Class Vendedor
    Inherits Empleado
    Private mvarVenta As Single
    Private mvarComision As Single
```

```
    Property Venta() As Single
        Get
            Venta = mvarVenta
        End Get
        Set(ByVal Value As Single)
            mvarVenta = Value
        End Set
    End Property
```

```
    Property Comision() As Single
        Get
            Comision = mvarComision
        End Get
        Set(ByVal Value As Single)
            mvarComision = Value
        End Set
    End Property
```

```
End Class
```

Finalmente, la clase "Vendedor" tendrá

- 5 propiedades: 3 heredadas: "Codigo", "Nombre" y "Basico" y 2 propias: "Venta" y "Comision".
- 2 métodos heredados: "CrearEmpleado" y "ActualizarBasico".
- 2 eventos heredados: "BasicoBajo" y "BasicoAlto".



## Sentencias para trabajar con Herencia

Para trabajar con herencia de clases existen varias instrucciones que hay que conocer tanto a nivel de la clase como de sus miembros que definen las características de la herencia, las cuales explicamos a continuación.

### ***Declaración de Clases Base***

Para declarar una clase base existen varias formas que fueron vistas en temas anteriores, ahora afianzaremos solo los tipos de declaraciones que posibilitan o limitan el uso de herencia de una clase base:

#### **1. MustInherit**

Permite crear una clase que solo sirva como clase base, es decir, que sirva solo para implementar herencia en otras clases, pero no podrá crearse objetos de esta clase.

**Sintaxis:**

```
MustInherit Class <Nombre Clase Base>  
    <Código de la clase>  
End Class
```

#### **2. NotInheritable**

Se usa para crear una clase que solo pueda crear objetos o aplicaciones clientes, pero que no pueda servir para heredarse en otra clase.

**Sintaxis:**

```
NotInheritable Class <Nombre Clase>  
    <Código de la clase>  
End Class
```

### ***SobreEscribiendo Propiedades y Métodos en Clases Derivadas***

Para declarar una propiedad o método en una clase derivada o clase que hereda de una clase base, hay que tener ciertas consideraciones de acuerdo al tipo de declaración, que se explican a continuación.

#### **1. Overridable**

Permite crear una propiedad o método que puede ser sobre escrito en una clase derivada. Esta declaración se hace en la propiedad o método de la clase base.

#### **2. Overrides**

Se usa para sobre escribir una propiedad o método que fue definido como "Overridable" en la clase base. Esta declaración se hace en la propiedad o método de la clase derivada.



### 3. **NotOverridable**

Impide que una propiedad o método pueda ser sobre escrito en una clase derivada. La definición se realiza en la propiedad o método de la clase base.

Por defecto, todas las propiedades o métodos públicos definidos en una clase base no pueden ser sobre escritos en la clase derivada.

### 4. **MustOverride**

Permite crear una propiedad o método que será obligatorio sobre escribirlo en la clase derivada. Esta declaración se realiza en la propiedad o método de la clase base que ha sido definida como MustInherit.

## **Palabras claves MyBase y MyClass**

Se puede usar las palabras clave MyBase y MyClass al trabajar con herencia, tal como se muestra a continuación:

### 1. **MyBase**

Se usa para llamar a miembros de la clase base desde la clase derivada. Es decir en vez de usar el nombre de la clase seguido de la propiedad o método se usa la palabra clave MyBase seguida de la propiedad o método.

Este tipo de llamada es útil cuando se trabaja con métodos sobre escritos en una clase derivada y se necesita invocar al método de la clase base que será sobre escrito, tal como se muestra en el siguiente ejemplo.

#### **Ejemplo:**

Suponiendo que el método "CrearEmpleado" de la clase "Empleado" haya sido creado como "Overridable", y se desea sobre escribir en la clase "Vendedor" para calcular correctamente el sueldo del vendedor incluyendo las comisiones, entonces, tendríamos lo siguiente:

```
Public Class Vendedor
    Inherits Empleado

    Public Overrides Sub CrearEmpleado(ByVal vCodigo As Integer, _
                                      ByVal vNombre As String, ByVal vSueldo As Single)
        vSueldo = vSueldo + mvarVenta * mvarComision
        MyBase.CrearEmpleado(vCodigo,vNombre,vSueldo)
    End Sub

End Class
```

### 2. **MyClass**

Se usa para llamar a métodos sobre escribibles desde la clase derivada, y diferenciarlos de los métodos heredados desde la clase base.

## Polimorfismo

El polimorfismo consiste en la funcionalidad múltiple que puede tener un miembro de una clase para comportarse de diversas maneras de acuerdo al tipo de objeto que lo implemente.

Existen dos formas de implementar el polimorfismo en Visual Basic .NET:

### 1. Polimorfismo basado en Herencia

Es una nueva forma de crear multiple funcionalidad para un método de una clase base que puede ser sobre escrito por otros métodos con el mismo nombre en las clases derivadas.

#### Ejemplo:

Tomemos como caso el ejemplo anterior donde habíamos sobre escrito el método "CrearEmpleado" de la clase "Empleado" modificandose en la clase derivada "Vendedor" para incluir lo recibido por comisiones de ventas. Crearemos el método llamado "MostrarSueldo" que permita crear el empleado y mostrar cuanto gana, usando el polimorfismo dependiendo si es vendedor o no.

```
Sub MostrarSueldo(ByVal vEmpleado As Empleado, _  
                  ByVal vCodigo As Integer, ByVal vNombre As String, _  
                  ByVal vSueldo As Single)  
    vEmpleado.CrearEmpleado(vCodigo,vNombre,vSueldo)  
    MsgBox(vNombre & " gana s/. " & vSueldo)  
End Sub
```

```
Sub ProbarPolimorfismo()  
    Dim objEmpleado As New Empleado  
    Dim objVendedor As New Vendedor  
    objEmpleado.CrearEmpleado(100, "Luis Dueñas", 1000)  
    objVendedor.Venta=1000  
    objVendedor.Comision=0.10  
    objVendedor.CrearEmpleado(100, "Luis Dueñas", 1000)  
End Sub
```

En este ejemplo el resultado será para el primer caso el mensaje "Luis Dueñas gana 1000" y en el segundo caso el mensaje "Luis Dueñas gana 1100".

### 2. Polimorfismo basado en Interfaces

Este tipo de polimorfismo se usa también en Visual Basic 6 y consiste en crear una interface donde se definan nombres de propiedades y/o métodos, y luego se implementen con la sentencia "Implements" en varias clases, escribiendo diferentes códigos o implementaciones para las propiedades y métodos de cada clase.

Este último tipo de polimorfismo no se va a tratar, debido a que la manera natural de implementar polimorfismo es a través de la herencia de clases sobre escribiendo propiedades o métodos.


#### **Laboratorio 4:**


Este laboratorio pretende enseñar como se trabajan con componentes en Visual Basic .NET, primero se verá un laboratorio de cómo crear una librería de clases y luego veremos como trabajar con herencia de clases. El laboratorio 6 se divide en dos ejercicios que duran aproximadamente **60 minutos**.

#### **Ejercicio 1: Creando y Usando una Librería de Clases**

##### ***Duración: 25 minutos***

Se va a construir una librería que permita manejar el inventario de productos en almacén y realizar un mantenimiento de estos. Para esto realizar los siguientes pasos:

 Elegir un nuevo proyecto "Visual Basic" y una plantilla "Class Library", seleccionar en ubicación la carpeta "C:\VBNET\Labs" y como nombre escribir "Libreria\_Clases".

 Escribir "Producto" como nombre físico y lógico de la clase:

```
Public Class Producto
```

```
End Class
```

 Creando propiedades para la clase producto:

```
Private mvarCodigo As String  
Private mvarNombre As String  
Private mvarPrecio As Single  
Private mvarStock As Integer
```

```
Public Property Codigo() As String  
Get  
    Codigo = mvarCodigo  
End Get  
Set(ByVal Value As String)  
    mvarCodigo = Value  
End Set  
End Property
```

```
Public Property Nombre() As String  
Get  
    Nombre = mvarNombre  
End Get  
Set(ByVal Value As String)  
    mvarNombre = Value  
End Set
```

End Property

Public Property Precio() As Single

Get

Precio = mvarPrecio

End Get

Set(ByVal Value As Single)

mvarPrecio = Value

End Set

End Property

Public Property Stock() As Integer

Get

Stock = mvarStock


End Get

Set(ByVal Value As Integer)

mvarStock = Value

End Set

End Property

 Creando métodos para la clase producto:

Public Sub CrearProducto(ByVal vCodigo As String, ByVal vNombre

As String, ByVal vPrecio As Single, ByVal vStock As Integer)

mvarCodigo = vCodigo

mvarNombre = vNombre

mvarPrecio = vPrecio

mvarStock = vStock

End Sub

Public Sub ActualizarPrecio(ByVal vOperacionPrecio As Byte,

ByVal vTasa As Single)

If vOperacionPrecio = 1 Then

mvarPrecio = (1 + (vTasa / 100)) \* mvarPrecio

Elseif vOperacionPrecio = 2 Then

mvarPrecio = (1 - (vTasa / 100)) \* mvarPrecio

End If

End Sub

Public Sub ActualizarStock(ByVal vOperacionStock As Byte,

ByVal vCantidad As Integer)

If vOperacionStock = 1 Then

mvarStock = mvarStock + vCantidad

Elseif vOperacionStock = 2 Then

mvarStock = mvarStock - vCantidad

End If

End Sub

- 🖥️ Luego, añadimos una aplicación para Windows que permita realizar el mantenimiento de productos usando la librería creada.
- 🖥️ Del menú “File” elegimos “Add Project”, y seleccionamos “New Project”. Elegimos una “Aplicación para Windows” y le damos el nombre de “Prueba\_Libreria\_Clases”.
- 🖥️ Configuramos para que la aplicación sea la que inicia, dando clic derecho al nombre en la ventana del “Solution Explorer” y eligiendo “Set as StartUp Project”.
- 🖥️ Diseñamos un formulario llamado “frmProducto” que tenga un “TabControl” con 3 fichas, una de ingreso de productos, otra de actualización de precios y stock y otra de consulta, diseñadas tal como muestran las figuras de abajo:

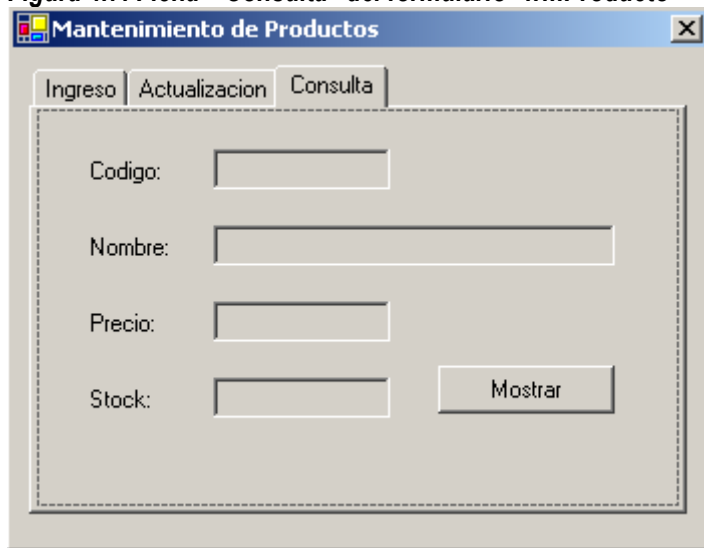
**Figura 4.5: Ficha “Ingreso” del formulario “frmProducto”**

The screenshot shows a Windows application window titled "Mantenimiento de Productos". It contains a "TabControl" with three tabs: "Ingreso", "Actualizacion", and "Consulta". The "Ingreso" tab is selected. Inside this tab, there are four text input fields labeled "Codigo:", "Nombre:", "Precio:", and "Stock:". To the right of the "Stock:" field is a button labeled "Crear".

**Figura 4.6: Ficha “Actualizacion” del formulario “frmProducto”**

The screenshot shows the same application window, but with the "Actualizacion" tab selected. The "Ingreso" and "Consulta" tabs are now disabled. The "Actualizacion" tab contains two groups of controls. The first group, titled "Modificacion Precio", has two radio buttons: "Aumento" and "Disminucion". Below these are two text input fields: "Tasa:" and "Cantidad:". The second group, titled "Modificacion Stock", also has two radio buttons: "Aumento" and "Disminucion". Below these are two buttons: "Actualizar Precio" and "Actualizar Stock".

**Figura 4.7: Ficha "Consulta" del formulario "frmProducto"**



Después de realizar el diseño y antes de escribir código, para usar la librería primero debemos hacer una referencia a ésta.

Del menú "Project" elegir "Add Reference...", seleccionar la ficha "Projects", elegir la librería creada y clic en "Select", luego "OK".

Declarar e inicializar una variable objeto de tipo de la clase producto, tal como sigue:  
`Private objProducto As New Libreria_Clases.Producto()`

Programando en el botón de crear producto:

```
Private Sub btnCrear_Click(...) Handles btnCrear.Click
    objProducto.CrearProducto(txtCodigo_Ing.Text, _
        txtNombre_Ing.Text, Convert.ToSingle(txtPrecio_Ing.Text), _
        Convert.ToInt16(txtStock_Ing.Text))
    txtCodigo_Ing.Clear()
    txtNombre_Ing.Clear()
    txtPrecio_Ing.Clear()
    txtStock_Ing.Clear()
End Sub
```

Programando los botones de actualizar precio y stock del producto:

```
Private Sub btnActualizarPrecio_Click(...) Handles ...
    If rbnPrecioAumento.Checked = True Then
        objProducto.ActualizarPrecio(1,
            Convert.ToSingle(txtTasa.Text))
    ElseIf rbnPrecioDisminucion.Checked = True Then
        objProducto.ActualizarPrecio(2,
```

```

Convert.ToSingle(txtTasa.Text))

End If
txtTasa.Clear()
End Sub

Private Sub btnStock_Click(...) Handles btnStock.Click
    If rbnStockAumento.Checked = True Then
        objProducto.ActualizarStock(1,
            Convert.ToInt16(txtCantidad.Text))
    ElseIf rbnStockDisminucion.Checked = True Then
        objProducto.ActualizarStock(2,
            Convert.ToInt16(txtCantidad.Text))
    End If
    txtCantidad.Clear()
End Sub

```

 Finalmente, mostrando los datos del producto:

```



Private Sub btnMostrar_Click(...) Handles btnMostrar.Click
    With objProducto
        txtCodigo_Con.Text = .Codigo
        txtNombre_Con.Text = .Nombre
        txtPrecio_Con.Text = .Precio.ToString
        txtStock_Con.Text = .Stock.ToString
    End With
End Sub

```

## Ejercicio 2: Trabajando con Herencia de Clases

**Duración: 35 minutos**

En este laboratorio vamos a construir una librería de clases para un Instituto Superior Tecnológico que permita matricular a los alumnos en un cierto curso libre dictado por un cierto profesor. Para lo cual realizamos los siguientes pasos:

-  Elegir un nuevo proyecto “Visual Basic” y una plantilla “Class Library”, seleccionar en ubicación la carpeta “C:\VBNET\Labs” y como nombre escribir “Libreria\_Herencia”.
-  Crear la clase “Persona”, modificando el nombre de la clase “Class1” por el de “Persona”, tanto física como lógicamente; luego escribir código para crear las propiedades: nombre, fechanac, edad y direccion; y el método crearpersona:

```

Public Class Persona
    Private mvarNombre As String
    Private mvarFechaNac As Date
    Private mvarEdad As Byte
    Private mvarDireccion As String

```

```

Property Nombre() As String
    Get
        Nombre = mvarNombre
    End Get
    Set(ByVal Value As String)
        mvarNombre = Value
    End Set
End Property


Property FechaNac() As Date
    Get
        FechaNac = mvarFechaNac
    End Get
    Set(ByVal Value As Date)
        mvarFechaNac = Value
    End Set
End Property

ReadOnly Property Edad() As Byte
    Get
        Edad = mvarEdad
    End Get
End Property

Property Direccion() As String
    Get
        Direccion = mvarDireccion
    End Get
    Set(ByVal Value As String)
        mvarDireccion = Value
    End Set
End Property

Sub CrearPersona(ByVal vNombre As String, _
                ByVal vFechaNac As Date, ByVal vDireccion As String)
    mvarNombre = vNombre
    mvarFechaNac = vFechaNac
    mvarDireccion = vDireccion
    mvarEdad = Convert.ToByte(Date.Today.Year-vFechaNac.Year)
End Sub
End Class

```

-  Crear la clase "Alumno" que hereda de "Persona", para lo cual añadimos una clase al componente; del menú "Project" elegimos "Add Class..." y escribimos como nombre "Persona.vb" y "Open". Luego escribimos el siguiente código:

```

Public Class Alumno
    Inherits Persona

```



```


Private mvarCodigo As String
Private mvarEspecialidad As String

Property Codigo() As String
    Get
        Codigo = mvarCodigo
    End Get
    Set(ByVal Value As String)
        mvarCodigo = Value
    End Set
End Property

Property Especialidad() As String
    Get
        Especialidad = mvarEspecialidad
    End Get
    Set(ByVal Value As String)
        mvarEspecialidad = Value
    End Set
End Property

Sub CrearAlumno(ByVal vNombre As String, _
    ByVal vFechaNac As Date, ByVal vDireccion As String, _
    ByVal vCodigo As String, ByVal vEspecialidad As String)
    CrearPersona(vNombre, vFechaNac, vDireccion)
    mvarCodigo = vCodigo
    mvarEspecialidad = vEspecialidad
End Sub
End Class

```

-  Crear la clase “Profesor” que también hereda de “Persona”, para lo cual añadimos una clase al componente; del menú “Project” elegimos “Add Class...” y escribimos como nombre “Profesor.vb” y “Open”. Luego escribimos el siguiente código:

```

Public Class Profesor
    Inherits Persona
    Private mvarCodigo As String
    Private mvarTipo As String

    Property Codigo() As String
        Get
            Codigo = mvarCodigo
        End Get
        Set
            mvarCodigo = Value
        End Set
    End Property


```

```

Property Tipo() As String
    Get
        Tipo = mvarTipo
    End Get
    Set
        mvarTipo = Value
    End Set
End Property

Sub CrearProfesor(ByVal vNombre As String, _
    ByVal vFechaNac As Date, ByVal vDireccion As String, _
    ByVal vCodigo As String, ByVal vTipo As String)
    CrearPersona(vNombre, vFechaNac, vDireccion)
    mvarCodigo = vCodigo
    mvarTipo = vTipo
End Sub
End Class

```

 Finalmente, crear la clase “Curso”, para lo cual añadimos una clase al componente; del menú “Project” elegimos “Add Class...” y escribimos como nombre “Curso.vb” y “Open”. Luego escribimos el siguiente código:

```

Public Class Curso
    Private mvarCodigo As String
    Private mvarNombre As String
    Private mvarTotalHoras As Byte
    Private mvarCostoTotal As Single
    Private mvarCostoHora As Single

    Property Codigo() As String
        Get
            Codigo = mvarCodigo
        End Get
        Set(ByVal Value As String)
            mvarCodigo = Value
        End Set
    End Property

    Property Nombre() As String
        Get
            Nombre = mvarNombre
        End Get
        Set(ByVal Value As String)
            mvarNombre = Value
        End Set
    End Property

    Property TotalHoras() As Byte

```

```


    Get
        TotalHoras = mvarTotalHoras
    End Get
    Set(ByVal Value As Byte)
        mvarTotalHoras = Value
    End Set
End Property


Property CostoTotal() As Single
    Get
        CostoTotal = mvarCostoTotal
    End Get
    Set(ByVal Value As Single)
        mvarCostoTotal = Value
    End Set
End Property

ReadOnly Property CostoHora() As Single
    Get
        CostoHora = mvarCostoHora
    End Get
End Property

Sub CrearCurso(ByVal vCodigo As String, _
               ByVal vNombre As String, ByVal vTotalHoras As Byte, _
               ByVal vCostoTotal As Single)
    mvarCodigo = vCodigo
    mvarNombre = vNombre
    mvarTotalHoras = vTotalHoras
    mvarCostoTotal = vCostoTotal
    mvarCostoHora = mvarCostoTotal / mvarTotalHoras
End Sub
End Class

```

 Para probar la librería de clase añadimos un proyecto a la solución, del menú “File”, elegimos “Add Project” y luego “New Project...”, seleccionando una aplicación para Windows a la cual llamaremos “Prueba\_Libreria\_Herencia”.

 La aplicación tendrá un formulario llamado “frmMatricula” con 3 fichas, donde se realizarán el ingreso de datos del alumno, el curso y el profesor para la matrícula, tal como se muestran en las figuras de abajo.

**Figura 4.8: Ficha “Alumno” del formulario “frmMatricula”**

Matricula de Alumnos en Cursos Libres

Datos del Alumno | Datos del Curso | Datos del Profesor

Codigo:  Nombre:

Fecha Nac:  Direccion:

Edad:  Especialidad:

Matricular Consultar Salir

**Figura 4.9: Ficha “Curso” del formulario “frmMatricula”**

The screenshot shows a Windows application window titled "Matricula de Alumnos en Cursos Libres". It has three tabs: "Datos del Alumno", "Datos del Curso" (which is selected), and "Datos del Profesor". The "Datos del Curso" tab contains the following fields:

Codigo:	Nombre:	
<input type="text"/>	<input type="text"/>	
Total Horas:	Costo Total:	Costo Hora:
<input type="text"/>	<input type="text"/>	<input type="text"/>



At the bottom of the form are three buttons: "Matricular", "Consultar", and "Salir".

**Figura 4.10: Ficha “Profesor” del formulario “frmMatricula”**

The screenshot shows the same application window, but with the "Datos del Profesor" tab selected. The fields in this tab are:

Codigo:	Nombre:
<input type="text"/>	<input type="text"/>
Fecha Nac:	Direccion:
<input type="text"/>	<input type="text"/>
Edad:	Tipo:
<input type="text"/>	<input type="text"/>

The buttons "Matricular", "Consultar", and "Salir" remain at the bottom.

-  Lo primero que tenemos que hacer para probar la librería en la aplicación es hacer una referencia a esta, eligiendo del menú "Project", "Add References...", seleccionar la ficha "Projects" y elegir la librería, clic en "Select" y "OK".
-  Para que la aplicación Windows que vamos a crear sea la que se ejecute dar clic derecho al nombre del proyecto en la ventana del "Solution Explorer" y elegir "Set as StartUp Project".

- 📖 En las declaraciones generales definimos variables objetos para manipular datos del alumno, curso y el profesor, así como una rutina que permita limpiar textos.

```
Private objAlumno As New Libreria_Herencia.Alumno()  
Private objCurso As New Libreria_Herencia.Curso()  
Private objProfesor As New Libreria_Herencia.Profesor()
```


```
Sub LimpiarTextos()  
    txtAlumno_Codigo.Clear()  
    txtAlumno_Nombre.Clear()  
    txtAlumno_FechaNac.Clear()  
    txtAlumno_Direccion.Clear()  
    txtAlumno_Edad.Clear()  
    txtAlumno_Especialidad.Clear()  
    txtCurso_Codigo.Clear()  
    txtCurso_Nombre.Clear()  
    txtCurso_TotalHoras.Clear()  
    txtCurso_CostoTotal.Clear()  
    txtCurso_CostoHora.Clear()  
    txtProfesor_Codigo.Clear()  
    txtProfesor_Nombre.Clear()  
    txtProfesor_FechaNac.Clear()  
    txtProfesor_Direccion.Clear()  
    txtProfesor_Edad.Clear()  
    txtProfesor_Tipo.Clear()  
End Sub
```

- 📖 Programamos el botón “Matricular” para que guarde los datos de los objetos creados anteriormente.

```
Private Sub btnMatricular_Click(...) Handles btnMatricular.Click  
    objAlumno.CrearAlumno(txtAlumno_Nombre.Text, _  
        Convert.ToDateTime(txtAlumno_FechaNac.Text), _  
        txtAlumno_Direccion.Text, txtAlumno_Codigo.Text, _  
        txtAlumno_Especialidad.Text)  
    objCurso.CrearCurso(txtCurso_Codigo.Text, _  
        txtCurso_Nombre.Text, _  
        Convert.ToByte(txtCurso_TotalHoras.Text), _  
        Convert.ToSingle(txtCurso_CostoTotal.Text))  
    objProfesor.CrearProfesor(txtProfesor_Nombre.Text, _  
        Convert.ToDateTime(txtProfesor_FechaNac.Text), _  
        txtProfesor_Direccion.Text, _  
        txtProfesor_Codigo.Text, txtProfesor_Tipo.Text)  
    LimpiarTextos()  
End Sub
```

- 📖 Escribimos código en el botón “Consultar” para mostrar los datos almacenados en los objetos.

```
Private Sub btnConsultar_Click(...) Handles btnConsultar.Click
    With objAlumno
        txtAlumno_Codigo.Text = .Codigo
        txtAlumno_Nombre.Text = .Nombre
        txtAlumno_FechaNac.Text=Format(.FechaNac, "dd/MM/yyyy")
        txtAlumno_Direccion.Text = .Direccion
        txtAlumno_Edad.Text = .Edad.ToString
        txtAlumno_Especialidad.Text = .Especialidad
    End With
    With objCurso
        txtCurso_Codigo.Text = .Codigo
        txtCurso_Nombre.Text = .Nombre
        txtCurso_TotalHoras.Text = .TotalHoras.ToString
        txtCurso_CostoTotal.Text = .CostoTotal.ToString
        txtCurso_CostoHora.Text = .CostoHora.ToString
    End With
    With objProfesor
        txtProfesor_Codigo.Text = .Codigo
        txtProfesor_Nombre.Text = .Nombre
        txtProfesor_FechaNac.Text=Format(.FechaNac, "dd/MM/yyyy")
        txtProfesor_Direccion.Text = .Direccion
        txtProfesor_Edad.Text = .Edad.ToString
        txtProfesor_Tipo.Text = .Tipo
    End With
End Sub
```

 Finalmente, programamos el botón de “Salir” de la aplicación.

```
Private Sub btnSalir_Click(...) Handles btnSalir.Click
    Me.Close()
End Sub
```