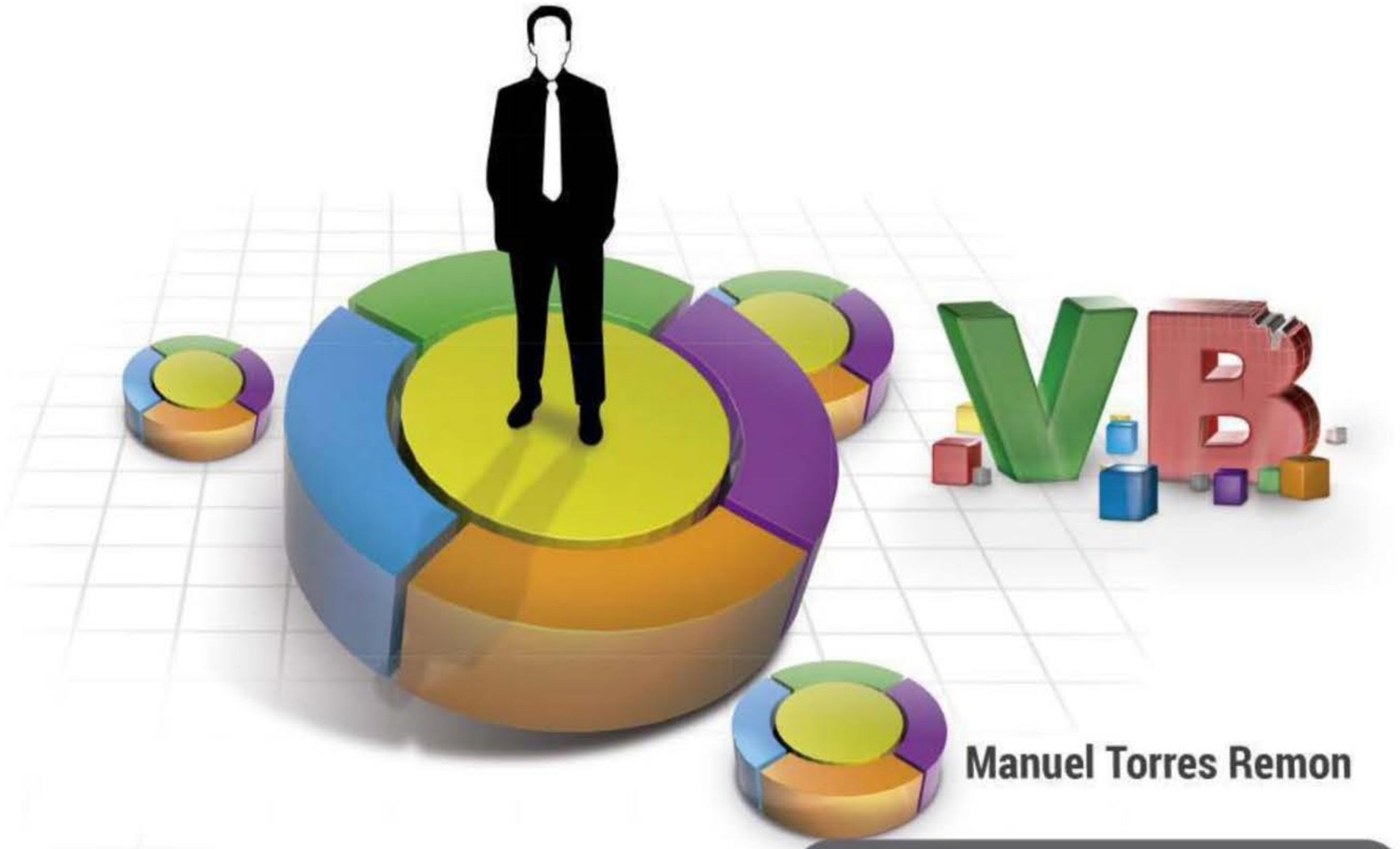
EDITORIAL MACRO®

Desarrollo de aplicaciones con Visual Basic 2015

Temas principales:

Componentes del entorno • Manejo de variables • Uso de sentencias
Aplicaciones con metodología modular • Programación orientada a objetos



Manuel Torres Remon

Incluye:
Material de descarga

Desarrollo de aplicaciones con

Visual Basic 2015

EDITORIAL MACRO®

Perú - México - Colombia - Chile - Ecuador - España - Bolivia - Uruguay - Guatemala - Costa Rica

¿Cómo ingresar a nuestra plataforma?

Material de descarga

1. Ingresa a **www.editorialmacro.com** e inicia sesión.
2. Encuentra la etiqueta **Plataformas** y selecciona **Material de descarga.**
3. Busca el libro e ingresa el código de acceso.
4. Haz clic en **Entrar.**

¡Y listo! Ahora podrás disfrutar de esta nueva experiencia virtual.

* Código único e intransferible.

EDITORIAL MACRO®

Desarrollo de aplicaciones con Visual Basic 2015
Autor: Manuel Torres Remon

© Derechos de autor registrados:
Empresa Editora Macro EIRL

© Derechos de edición, arte gráfico y diagramación reservados:
Empresa Editora Macro EIRL

Coordinación de edición:
Magaly Ramon Quiroz

Diseño de portada:
Alessandra Bonilla Zapata

Diagramación:
Julissa Ventocilla Fernández

Edición a cargo de:
© Empresa Editora Macro EIRL
Av. Paseo de la República N.° 5613, Miraflores, Lima, Perú

Teléfono: (511) 748 0560
E-mail: proyectoeditorial@editorialmacro.com
Página web: www.editorialmacro.com

Primera edición: Mayo 2017
Tiraje: 1300 ejemplares

Impresión
Talleres gráficos de la Empresa Editora Macro EIRL
Jr. San Agustín N.° 612-624, Surquillo, Lima, Perú
Mayo 2017

ISBN N.° 978-612-304-519-7
Hecho el depósito legal en la Biblioteca Nacional del Perú N.° 2016-13696

Prohibida la reproducción parcial o total, por cualquier medio o método, de este libro sin previa autorización de la Empresa Editora Macro EIRL.

Manuel Torres Remon

Licenciado en Informática. Se ha dedicado a la consultoría y docencia de cursos de tecnología desde hace 15 años. A su vez, ha brindado diversos talleres de capacitación en las instituciones más importantes de Lima (Perú).

Realizó sus estudios en el Instituto de Educación Superior Manuel Arévalo Cáceres, en la Universidad Alas Peruanas y en la Universidad Peruana del Norte. En todas estas instituciones, obtuvo una esmerada formación profesional que ha podido demostrar ampliamente a lo largo de su trayectoria docente. Actualmente, se desempeña como profesor de tecnología en los institutos Manuel Arévalo Cáceres, Cibertec y Unimáster, en donde imparte los cursos de Programación, Base de Datos y Análisis de Sistemas.

Ha publicado libros como Programación Transact con SQL Server 2012, Desarrollo de aplicaciones con JAVA, JCreator, JDeveloper NetBeans, Desarrollo de aplicaciones web con PHP, Diseño web con HTML5 y CSS3, Aplicaciones VBA con Excel, y Guía práctica Macros, aplicaciones VBA con Excel.

Para cualquier duda o consulta sobre el libro, puede escribir al siguiente correo: manuel.torresr@hotmail.com.

Dedicatoria

Este libro está dedicado con mucho cariño y aprecio a mis pequeñas Ángela Victoria y Fernanda Ximena, que son y seguirán siendo mi fuente de inspiración, y mucho más a mi esposa Luz por comprenderme en todo lo que me propongo.

Agradecimientos

Todo libro escrito es un sacrificio que tiene que ser valorado y esto solo se dará si usted aprecia este material, como lo aprecio yo. Escuché, alguna vez, la frase: «La lectura de un libro enriquece de conocimientos y empobrece la ignorancia», pienso que es cierto, aunque hoy un libro impreso es difícil de leer, pues son pocas las personas que cargan uno en la mano, ya que muchos de ellos han sido reemplazados por los dispositivos móviles. Por eso, mi primer agradecimiento es para usted, amigo lector, por adquirir este material que ha significado tiempo y esfuerzo para mí.

Asimismo, deseo agradecer a la gran familia Macro por confiar nuevamente en mi persona para el desarrollo del libro.

Finalmente, agradecer de gran manera a mis pequeñas hijas Ángela Victoria y Fernanda Ximena Torres Lázaro y a mi esposa Luz, porque gracias a que sacrifiqué el tiempo que pude pasar con ellas puedo presentarles este libro.

Índice

Introducción 15

CAPÍTULO 1: Introducción al Visual Studio 2015 17

1.1 Introducción al Visual Studio 2015 19
1.2 Proceso de instalación del Visual Studio 2015 27
1.3 IDE Visual Studio 2015 32
1.4 Soluciones y proyectos Visual Studio 2015 41
1.5 Desarrollo de aplicaciones tipo consola 48
Caso desarrollado 1: Cálculo de sueldos 50
Caso desarrollado 2: Venta de helados 53
Caso desarrollado 3: Promedio de notas 55
Caso desarrollado 4: Hospedaje 56
1.6 Retroalimentación 58
1.7 Ejercicios propuestos 60
Ejercicio propuesto 1: Conversión de grados 60
Ejercicio propuesto 2: Venta de autos 60

CAPÍTULO 2: Controles de formularios 61

2.1 Trabajando con Windows Forms 63
2.2 Controles visuales 72
2.3 Notación para los nombres de los controles 73
2.4 Control Form 74
2.5 Controles de visualización de información 77
2.6 Controles de ingreso de información 81
2.7 Controles de acción 85
2.8 Controles de opción simple y múltiple 88
2.9 Controles de muestra de información 96
2.10 Controles contenedores de otros controles 102
2.11 Controles de manejo de imágenes 104
2.12 Controles de manejo de tiempos 105
2.13 Operaciones básicas con los formularios 109
2.14 Eventos de los controles 112
2.15 Editor de código 115
2.16 Desarrollo de aplicaciones tipo Windows Forms 116

Caso desarrollado 1: Manejo del control RichTextBox....117
Caso desarrollado 2: Cálculo de sueldos....121
Caso desarrollado 3: Venta de helados....123
Caso desarrollado 4: Promedio de notas....126
Caso desarrollado 5: Hospedaje....128

CAPÍTULO 3: Fundamentos de programación....131

3.1 Componentes de un código de aplicación....133
3.2 Las variables....140
3.3 Operadores....143
3.4 Métodos matemáticos de .NET Framework....146
3.5 Conversiones de tipo....148
3.6 Clase Strings....151
3.7 Enumeraciones....153
3.8 La clase MessageBox....156
3.9 Desarrollo de aplicaciones secuenciales....159
Caso desarrollado 1: Declaración de variable básica - Pago de vendedores....159
Caso desarrollado 2: Declaración por literales de tipo identificador - Venta de productos....163
Caso desarrollado 3: Declaración variante - Cajero automático....166
Caso desarrollado 4: Conversión por palabra clave - Temperatura....168
Caso desarrollado 5: Conversión usando CType - Donación....171
Caso desarrollado 6: Método Parse - Distancia....174
Caso desarrollado 7: Manejo de cadenas - Generación de códigos y correos corporativos...177

CAPÍTULO 4: Estructuras de decisión....181

4.1 Operadores para estructuras de decisión....183
4.2 Estructura condicional If....184
4.3 Sentencia de condición múltiple simple Choose....193
4.4 Sentencia de condición múltiple....195
4.5 Desarrollo de aplicaciones condicionales....198
Caso desarrollado 1: If simple - Registro de venta de gasolina....198
Caso desarrollado 2: If doble - Control de contratos de publicidad....203
Caso desarrollado 3: If doblemente enlazado - Simulador de pensión para estudiantes ... 208
Caso desarrollado 4: Select Case - Control de venta de productos por temporada....213

CAPÍTULO 5: Sentencias repetitivas....219

5.1 Los contadores....221
5.2 Operaciones sobre los contadores....223

5.3 Los acumuladores 225
5.4 Operaciones sobre los acumuladores 226
5.5 Sentencia While - End While 228
5.6 Sentencia Do - Loop While 231
5.7 Sentencia For 234
5.8 Desarrollo de aplicaciones repetitivas 236
Caso desarrollado 1: Contadores - Control de vehículos en estacionamiento 237
Caso desarrollado 2: Acumuladores - Control de entradas de concierto 242
Caso desarrollado 3: While... End While - Financiamiento de vehículo Simulador 248
Caso desarrollado 4: Do... Loop While - Control de registro de nuevos libros 256
Caso desarrollado 5: For - Control de pensión de estudiantes 265

CAPÍTULO 6: Programación modular 275

6.1 Introducción a la programación modular 277
6.2 Subprocedimientos 284
6.3 Procedimiento de función 285
6.4 Procedimiento de propiedad 288
6.5 Miembros de la biblioteca en tiempo de ejecución 289
6.6 Microsoft Visual Basic Information 310
6.7 Funciones de conversión 311
6.8 Función lambda 312
Caso desarrollado 1: Subprocedimiento sin parámetros - Fábrica de vidrio 314
Caso desarrollado 2: Subprocedimiento con parámetros - Proforma de autos 324
Caso desarrollado 3: Procedimiento de función sin parámetros - Control de cotización 335
Caso desarrollado 4: Procedimiento de función con parámetros - Fabricación de mesas 341
Caso desarrollado 5: Procedimiento de Propiedad - Control de pago por consumo 349
Caso desarrollado 6: Funciones de cadena - Registro de empleado 355
Caso desarrollado 7: Funciones de fecha - Simulador de pago por préstamo 359
Caso desarrollado 8: Uso del InputBox - Casa de cambio 364
Caso desarrollado 9: Función lambda - Control de cotizaciones 368

CAPÍTULO 7: Introducción a la programación orientada a objetos 375

7.1 Introducción a la programación orientada a objetos 377
7.2 Los atributos de clase 385
7.3 Los objetos 387
Caso desarrollado 1: Manejo de clase y objeto - Fábrica de vidrio 390

Referencias bibliográficas 399

Introducción

El libro *Desarrollo de aplicaciones con Visual Basic 2015* les permitirá crear aplicaciones de escritorio usando como plataforma de trabajo al Visual Studio 2015.

Para cumplir con dicho objetivo hemos dividido el libro en siete capítulos. El primero hace referencia a todos los conceptos básicos que usted debe conocer con respecto del Visual Studio 2015. Asimismo, se desarrollará al final del mismo aplicaciones de tipo consola como un inicio de la programación Visual Basic.

En el capítulo dos se hace un reconocimiento de todos los elementos que puede contener una aplicación en Visual Basic 2015, así como las principales herramientas con las que podemos contar en el desarrollo de nuestras aplicaciones. Al final del mismo se expondrán algunos casos prácticos usando formularios de tipo Windows.

En el capítulo tres se ingresará al desarrollo de aplicaciones usando variables, operadores y todo lo que una aplicación debe contener. El enfoque se realizará especialmente en el código de la aplicación, explicando varios casos desarrollados.

En el capítulo cuatro se explicará el uso de las sentencias condicionales como parte principal en el desarrollo de aplicaciones. Al final se desarrollarán casos usando las diferentes sentencias que ofrece el lenguaje Visual Basic.

El capítulo cinco iniciará con el uso de los contadores y acumuladores, siguiendo con las principales sentencias de repetición con el *For* y *While*, explicándolos con casos desarrollados al final del capítulo.

En el capítulo seis se hará uso de la programación modular. Se comprobará cómo esta nos ayuda a mejorar nuestras aplicaciones mediante el uso de funciones, procedimientos y la nueva expresión lambda, que serán explicados mediante el desarrollo de casos al final del capítulo.

Finalmente, en el capítulo siete se hará uso de una introducción a la programación orientada a objetos con Visual Basic 2015, explicando con un caso desarrollado el uso de la clase y los objetos.

Capítulo 1

Introducción al Visual Studio 2015

Capacidad terminal:

- Reconocer conceptos básicos del entorno de Visual Studio 2015.
- Reconocer los pasos para la instalación de Visual Studio 2015.
- Reconocer los elementos que componen el entorno de Visual Studio 2015.
- Crear proyectos y soluciones de tipo Visual Basic 2015.
- Desarrollar aplicaciones Visual Basic con entorno tipo consola.

Contenido

- Introducción al Visual Studio 2015
- Proceso de instalación del Visual Studio 2015
- IDE Visual Studio 2015
- Soluciones y proyectos Visual Studio 2015
- Desarrollo de aplicaciones tipo consola
- Retroalimentación
- Ejercicios propuestos

1.1 Introducción al Visual Studio 2015

Visual Studio siempre se ha caracterizado por proponer comodidad al programador, tal es así que integra varios lenguajes de programación en un solo entorno; desde aquí podrá crear aplicaciones profesionales para todo tipo de dispositivos. Este material se enfoca en las de escritorio, como el inicio en el desarrollo de aplicaciones, pero también puede implementar aplicaciones enfocadas para web, móviles o la nube.

VS2015 es considerado como un IDE (*Integrated Development Environment* o entorno de desarrollo integrado) que proporciona un servicio integral al programador haciendo un ambiente agradable de desarrollo. Como IDE ofrece un editor de código totalmente intuitivo, haciendo que el nuevo programador se acondicione en muy corto tiempo. Asimismo, cuenta con el Intellisense o también llamado código inteligente, que permite autocompletar el código mientras escribe una sentencia. En cuanto a la programación móvil, se puede implementar aplicaciones usando código para iOS y Android.

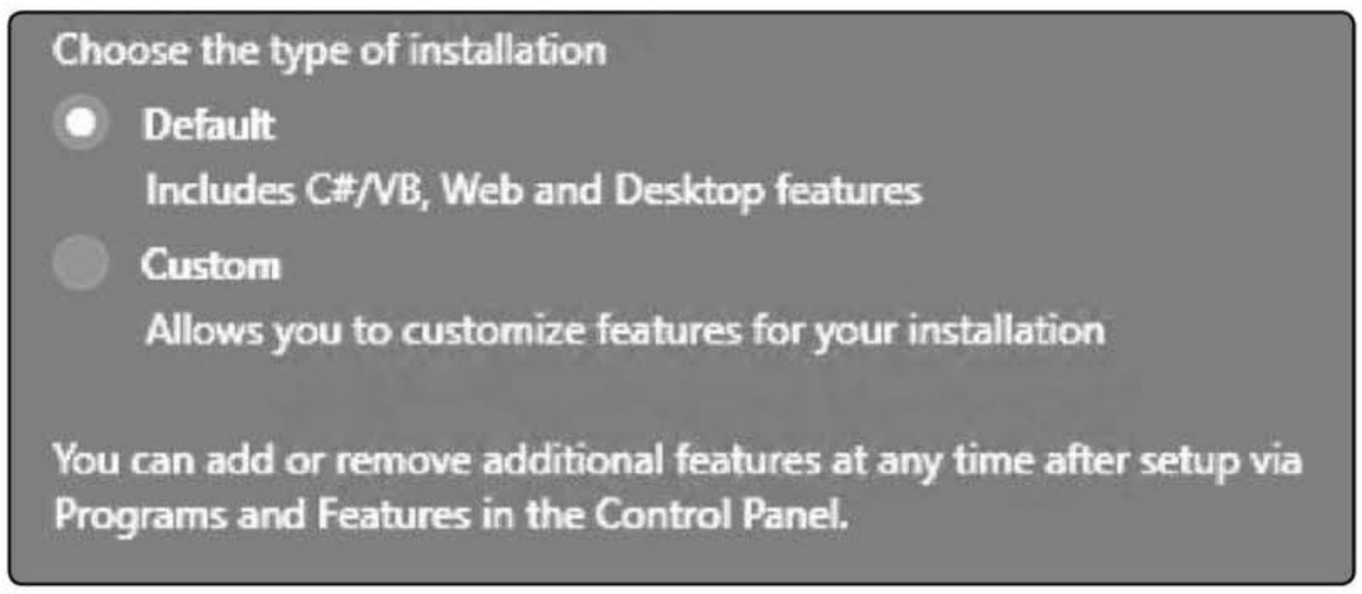
Finalmente, se puede decir que Visual Studio 2015 provee un conjunto de herramientas integradas que nos permitirán desarrollar programas enfocados en aplicaciones de plataforma, que servirán como base para otros tipos de aplicaciones que se necesiten para el futuro.

1.1.1 Características

Microsoft tiene el objetivo de reconocer a Visual Studio como un entorno de desarrollo de aplicaciones multiplataforma y multidispositivo; asimismo, desea hacer que estas aplicaciones puedan ser desplegadas en la nube. Sus principales características son las siguientes:

- El framework .NET y el Visual Studio Code, más conocido como VS Code, son considerados *open source*, abriendo así un camino importante para los desarrolladores de la comunidad Microsoft, que quieran realizar mejoras en el lenguaje. Para visualizar los códigos compartidos se puede visitar la siguiente página: **<https://code.visualstudio.com/updates#_vs-code-is-open-source>**.

- La distribución de Visual Studio es de forma gratuita tanto para entornos Windows, Mac y Linux. Asimismo, la instalación se ha dividido en componentes para que al realizarla se pueda seleccionar los elementos necesarios, esto le permite a Visual Studio realizar una instalación ligera. Pero si después de esta se necesita agregar algún elemento se podrá realizar de manera sencilla.



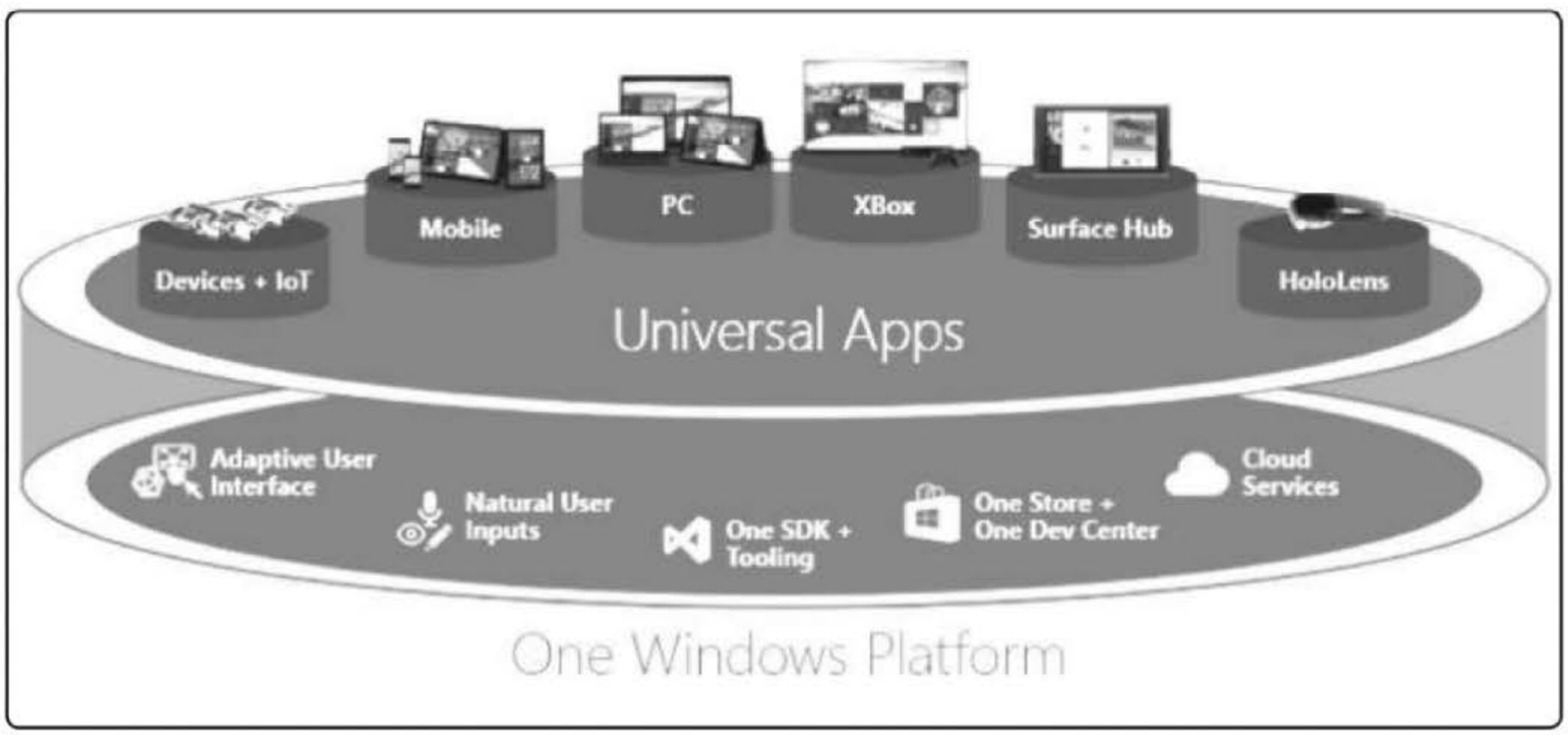
- Una de las novedades más resaltantes de esta versión es la cuenta de usuario Visual Studio, eso quiere decir que para empezar con el desarrollo de aplicación se debe tener una cuenta de usuario, el cual permitirá acceder a recursos en línea y otras novedades. También, se debe mencionar que se puede tener varias cuentas de Visual Studio.

  Al iniciar sesión en Visual Studio, de forma automática ocurre lo mismo en todas las instancias de Visual Studio 2015. Adicional a esto, Visual Studio también transmite la lista de cuentas con su cuenta de personalización para que se pueda empezar rápidamente a trabajar con una de esas cuentas en otros dispositivos.

- Con respecto al trabajo con aplicaciones móviles, Visual Studio presenta una integración entre HTML/JavaScript y Apache Córdova, debido a la colaboración entre esta última, de código abierto, y Microsoft. Con Visual Studio Tools para Apache Córdova, todas las aplicaciones tendrán un acceso a capacidades de dispositivos nativos en todas las aplicaciones a través de complementos que ofrecen una API común de JavaScript.

- La plataforma universal de Windows permite implementar aplicaciones bajo la plataforma de Windows 10, que podrán ser ejecutadas en cualquier tipo de dispositivo, desde móviles hasta equipos de escritorio.

  Para lograr esto, se debe ejecutar la aplicación en un teléfono o en un equipo de escritorio con Windows 10, también podría ser ejecutado en un Xbox. La idea principal es que todas las plataformas cuenten con la extensión SDK para poder agregar a sus aplicaciones el comportamiento Universal Apps, en orden de aprovechar los comportamientos específicos de cada plataforma. Se debe tener en cuenta que las aplicaciones podrán ser desarrolladas en C#, Visual Basic, C++ o JavaScript.

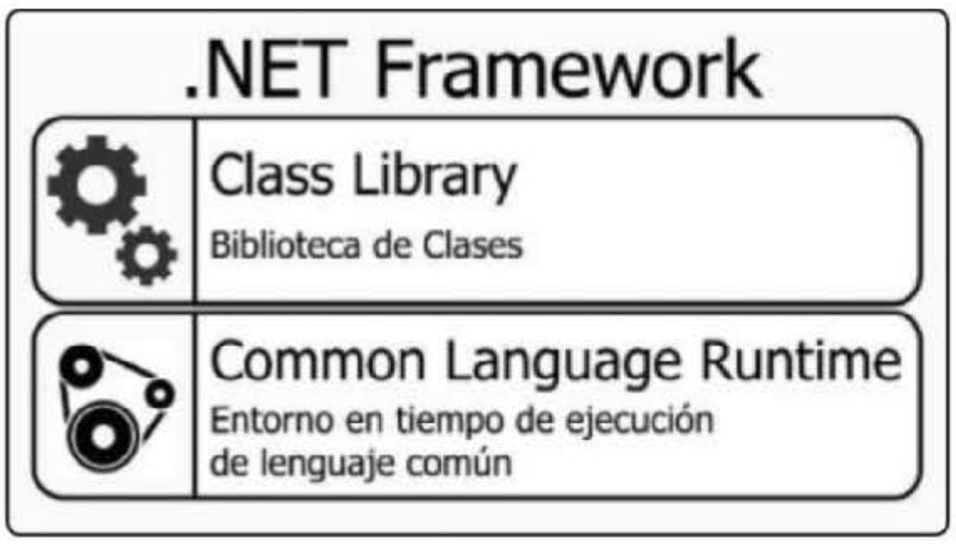


- Incorpora el compilador Roslyn, el cual se encarga de realizar la comprobación de sintaxis en tiempo real; asimismo, autocompleta el código en la ventana Código fuente y presenta aquí un código de sugerencia ante un error mediante bombillas ubicadas en el lado izquierdo. Los componentes del compilador Roslyn son:
  - Árbol de sintaxis: permite crear y recorrer el árbol sintáctico que está utilizando el compilador.
  - Símbolos: analizan la estructura de las clases sin necesidad de profundizar en un lenguaje en concreto.
  - Binding y análisis de flujos: proporciona la información semántica de código, dónde, cuándo y cómo se están utilizando las clases y sus propiedades y métodos.

1.1.2 Microsoft .NET Framework

Antes de iniciar con el desarrollo de aplicaciones, primero se debe conocer algunos conceptos básicos sobre los componentes de Visual y cómo interactúan con este desarrollo.

Se inicia con el .NET Framework, el cual es considerado como un entorno de ejecución Runtime, que administra de manera eficaz las aplicaciones en cualquier sistema operativo. Asimismo, una de sus características más importantes es que proporciona muchos servicios a las aplicaciones cuando estas se encuentran en ejecución, es así como los desarrolladores implementan sus aplicaciones con un código estable, sólido y confiable.



Entre los principales componentes del .NET Framework se encuentra el Common Language Runtime (CLR), el cual es considerado como el motor de ejecución que controla las aplicaciones en etapa de ejecución; y a la biblioteca de clases de .NET Framework, la que proporciona una base de códigos probados y reutilizables, los cuales pueden ser usados por los programadores en el desarrollo de sus propias aplicaciones.

Entre los principales servicios que ofrece .NET Framework a las aplicaciones en ejecución, se encuentran:

- Proporciona servicios para la administración de la memoria en el desarrollo de aplicaciones.
- Proporciona tipos de datos comunes entre los lenguajes de programación del entorno Visual Studio.
- Proporciona una amplia biblioteca de clases, la cual permite controlar operaciones comunes en el desarrollo de una aplicación. Asimismo, proporciona bibliotecas especializadas para un tipo de desarrollo como ASP.NET, ADO.NET y WCF.
- Proporciona interoperabilidad entre los diferentes lenguajes de programación, generando un código llamado Lenguaje Intermedio Comun (CIL).
- Proporciona compatibilidad entre las versiones posteriores de Visual Studio.
- Proporciona compatibilidad entre diversos sistemas operativos llamado multitargeting.

1.1.3. Biblioteca de clases

.NET Framework cuenta con una biblioteca completa de clases, interfaces y tipos de valor, que proporcionan acceso a la funcionalidad en el desarrollo de una aplicación. Se podría decir que es la base fundamental sobre la que se compila todo tipo de aplicación.

Cuando se crea un nuevo proyecto de aplicación Basic en Visual Studio se incorporan referencias a los ensamblados más comunes y necesarios para la aplicación. Si necesita usar otras referencias se podrán añadir según necesite el proyecto de aplicación.

Agregar una referencia desde el Explorador de Soluciones

- Haga clic derecho sobre el proyecto.
- Seleccione **Agregar > Referencia**.

Agregar una referencia desde la barra de menús

- **Proyecto > Agregar referencia**.

Agregar una referencia desde la carpeta Referencias del Explorador de Soluciones

- Haga clic derecho sobre la carpeta **Referencia**.
- Seleccione **Agregar referencia**.

La biblioteca de clases está compuesta por espacios de nombres, estos se caracterizan por contener tipos que se pueden utilizar en las aplicaciones como clases, estructuras, enumeraciones, delegados e interfaces. A continuación, una lista de los espacios de nombres contenidas dentro de la biblioteca de clases más utilizadas:

Espacio de nombres	Descripción
System	Es el principal espacio de nombres usada en todas las aplicaciones de Visual Studio; contiene múltiples clases que definen tipos de datos de referencia y de valor. Entre sus principales clases están: • **Array**: Representa la manipulación de la información referente a matrices; asimismo, permite administrar la información mediante la búsqueda y la ordenación de sus datos. • **Console**: Representa la manipulación de los datos de entrada y salida para aplicaciones de consola; asimismo, presenta un control de excepciones. • **Convert**: Representa la conversión que se puede dar en un valor de un tipo a otro. • **Exception**: Representa los errores genéricos que se puede ocasionar en una aplicación. • **Math**: Representa a todas las funciones matemáticas, trigonométricas, logarítmicas y muchas otras funciones.

	• **Random**: Representa a un generador de números aleatorios. • **String**: Representa a una serie de caracteres de tipo Unicode. • **Type**: Representa a todo tipo de declaración en una aplicación.
System.Collections	Contiene una serie de clases que permite manejar información masiva como si se tratase de un arreglo, con la diferencia que estos datos son llamados colecciones como ArrayList, Pilas, Colas y Diccionarios.
System.Configuration	Representa a la información administrada por un archivo de configuración; en este caso se usará al agregar un modelo de datos cliente-servidor a la aplicación.
System.Data	Representa un conjunto de clases que permiten tener acceso a la información de diversas fuentes de datos, como podría ser SQL Server mediante la arquitectura ADO.NET.
System.IO	Representa las clases que permiten manipular información referente a archivos; con ellos se podrá leer y grabar información dentro de un archivo de texto, binario o comprimido.
System.Security	Representa las clases que permiten manipular la seguridad de los datos como una criptografía.
System.Threading	Representa las clases que permiten controlar la ejecución de varios procesos. Estos pueden ser asíncronos o simultáneos.

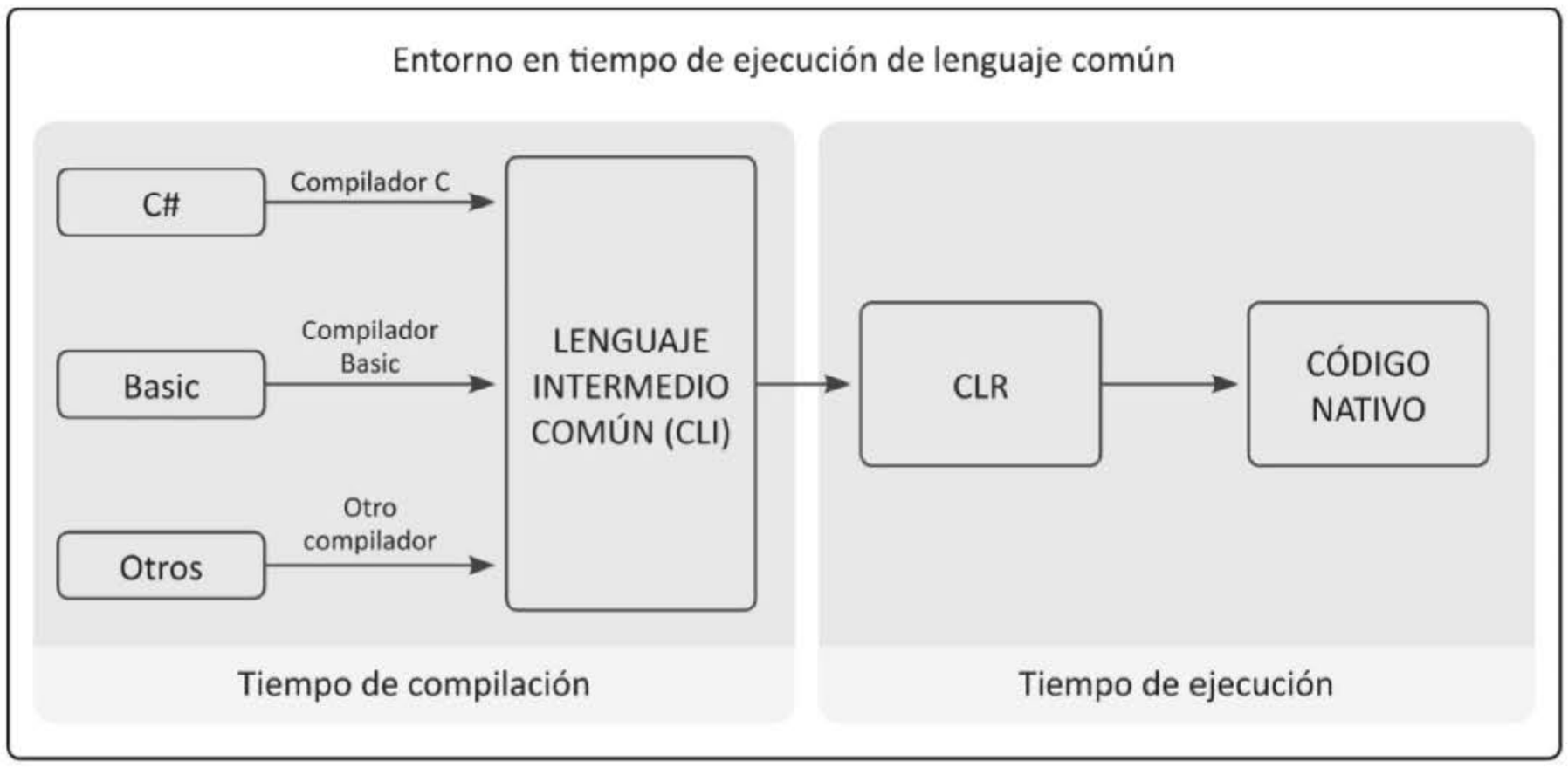
1.1.4 Entorno en tiempo de ejecución de lenguaje común

Cuando se ejecuta una aplicación, el .NET Framework proporciona un entorno que facilita el proceso de desarrollo; dicho entorno lo realiza en un tiempo de ejecución y es denominado CLR o Common Language Runtime, el cual se encarga de ejecutar el código y proporcionar los servicios necesarios para dicho proceso. Entre las principales características que presenta están:

- Administra de manera coherente la memoria del computador.
- Administra la ejecución de subprocesos.
- Administra la compilación y demás servicios que solicite el sistema.
- Implementa una infraestructura estricta de comprobación de tipos y código llamada CTS o Common Type System, el cual garantiza que todo código sea autodescriptivo.
- La característica más resaltante del CLR es la forma de administrar los objetos, ya que permite administrarlos automáticamente así como a su referencia y liberación, cuando estos no son utilizados.
- CLR aplica la interoperabilidad, ya que permite al programador usar sus componentes COM o DLLs, desarrolladas en diferentes versiones.

- Compilación JIT, permite ejecutar todo el código administrado en lenguaje maquina nativo del sistema en el que se ejecuta la aplicación; mientras tanto, administra la memoria mejorando enormemente su rendimiento.
- SIMD (Single Instruction Multiple Data) consiste en un repositorio de datos, el cual despacha las instrucciones a diferentes unidades de procesamiento; la idea principal es crear un solo código que pueda ser compilado de manera sincronizada, creando un *pool* de instrucciones.

La siguiente imagen muestra la forma de trabajo del entorno en tiempo de ejecución del Runtime:



Cuando el compilador interpreta el código desde un lenguaje de programación genera un código intermedio antes de generar uno nativo; se debe tener en cuenta que el código nativo es aquel que puede ser interpretado por el procesador. A este se le llama código administrado y es generado por el CLI (Common Intermediate Language) o MSIL (Microsoft Intermediate Language). Finalmente entra JIT (Just in time), el cual se encarga de convertir el código generado por MSIL en código nativo entendible por el sistema operativo. Toda esta actividad se realiza en microsegundos, pero la velocidad original será determinada por el dispositivo donde se ejecuta la aplicación.

1.1.5 Versiones de la plataforma .NET

En el año 1996 Microsoft contaba con 4 productos independientes, esto creaba desarrolladores de diferentes líneas a pesar de usar un producto del mismo proveedor, estos eran Visual Basic, Visual C++, Visual FoxPro y Visual SourceSafe. El producto que permitía una integración entre todos, era el Visual SourceSafe pero no resultaba productivo que un programador nativo de una línea se integre a los demás.

Es así que en 1997 decide integrar los productos mencionados anteriormente en uno solo llamándolo Visual Studio 97, y lo muestra al mundo en febrero del mismo año. De allí en adelante han surgido diferentes versiones que se verán a continuación:

Versiones visual	Productos añadidos	Fecha de lanzamiento
Microsoft Visual Studio 97	ASP. y MSDN	Febrero 1997
Microsoft Visual Basic 6.0	Visual J++	Junio 1998
Visual Studio .net Professional	C#, J#, Visual Basic .NET, ASP .NET	Febrero 2002
Visual Studio .NET 2003	Visio 2012 para UML	Abril 2003
Microsoft Visual Studio 2005	Windows Mobile	Noviembre 2005
Microsoft Visual Studio 2008	WPF, WCF, SilverLight, LinQ, Expression y Reporting	Noviembre 2007
Microsoft Visual Studio 2010	F#, Windows Phone 7	Abril 2010
Visual Studio 2012 Professional	WinRt, TypeScript y Python	Setiembre 2012
Visual Studio 2013	Azure	Octubre 2013
Visual Studio 2015	Xamarin, Apache Cordova	Julio 2015

1.2 Proceso de instalación del Visual Studio 2015

Visual Studio 2015 ofrece un completo entorno de desarrollo integrado para crear aplicaciones en entornos como Windows, Android e iOS, además de aplicaciones web y servicios de nube innovadores. Y, tal como se había mencionado, la distribución de Visual es gratuita mediante la comunidad.

1.2.1 Link de descarga

<https://www.visualstudio.com/es-es/downloads/download-visual-studio-vs.aspx>

1.2.2 Requisitos mínimos

Para la instalación de Visual Studio 2015 se dividirá en secciones la línea de requisitos; se debe tomar en cuenta que estos representan lo mínimo que se debe contar para una instalación estable:

Hardware	• Procesador de 1.6 GHz o superior. • 1GB de memoria RAM. • 4GB de espacio disponible en el disco duro. • Resolución de pantalla de 1024x768 o superior.
Software	• Windows 7 Service Pack 1 o superior. • Windows server 2012 R2.
Opcional	• 1.5GB de memoria RAM solo si usara una máquina virtual para ejecutar las aplicaciones.

1.2.3 Proceso de instalación

Aquí se inicia el proceso de instalación de Visual Studio 2015 y se explicará paso a paso algunas referencias importantes, porque hoy en día los asistentes de instalación son muy intuitivos y no se necesita de un experto para instalar aplicaciones. A continuación, las imágenes que representan los pasos para la instalación:

Paso 1: Iniciando el proceso de instalación

El proceso de instalación inicia solicitando la ubicación en donde se guardarán los archivos de instalación; aquí no se modificará nada puesto que todas las aplicaciones se instalan en **Archivos de programa**; desde esta ventana solo se seleccionará **Siguiente>**.

Paso 2: Seleccionando características

Como se mencionará en este capítulo, una de las características de Visual Studio 2015 es que el programador podrá seleccionar qué elementos desea instalar y hacer mucho más ligero al entorno; a continuación, se puede presionar **Next** para realizar una instalación de forma básica, esto incluye los principales lenguajes como Visual C# y Visual Basic. También se podría seleccionar todas las características activando el check **Select All** y así realizar una instalación completa de la aplicación.

Paso 3: Proceso de instalación

Antes de comenzar con la instalación se debe comprobar que el espacio solicitado sea el adecuado; en esta ventana solo se debe presionar el botón **Install**.

1.2.4 Cambiar el idioma del entorno

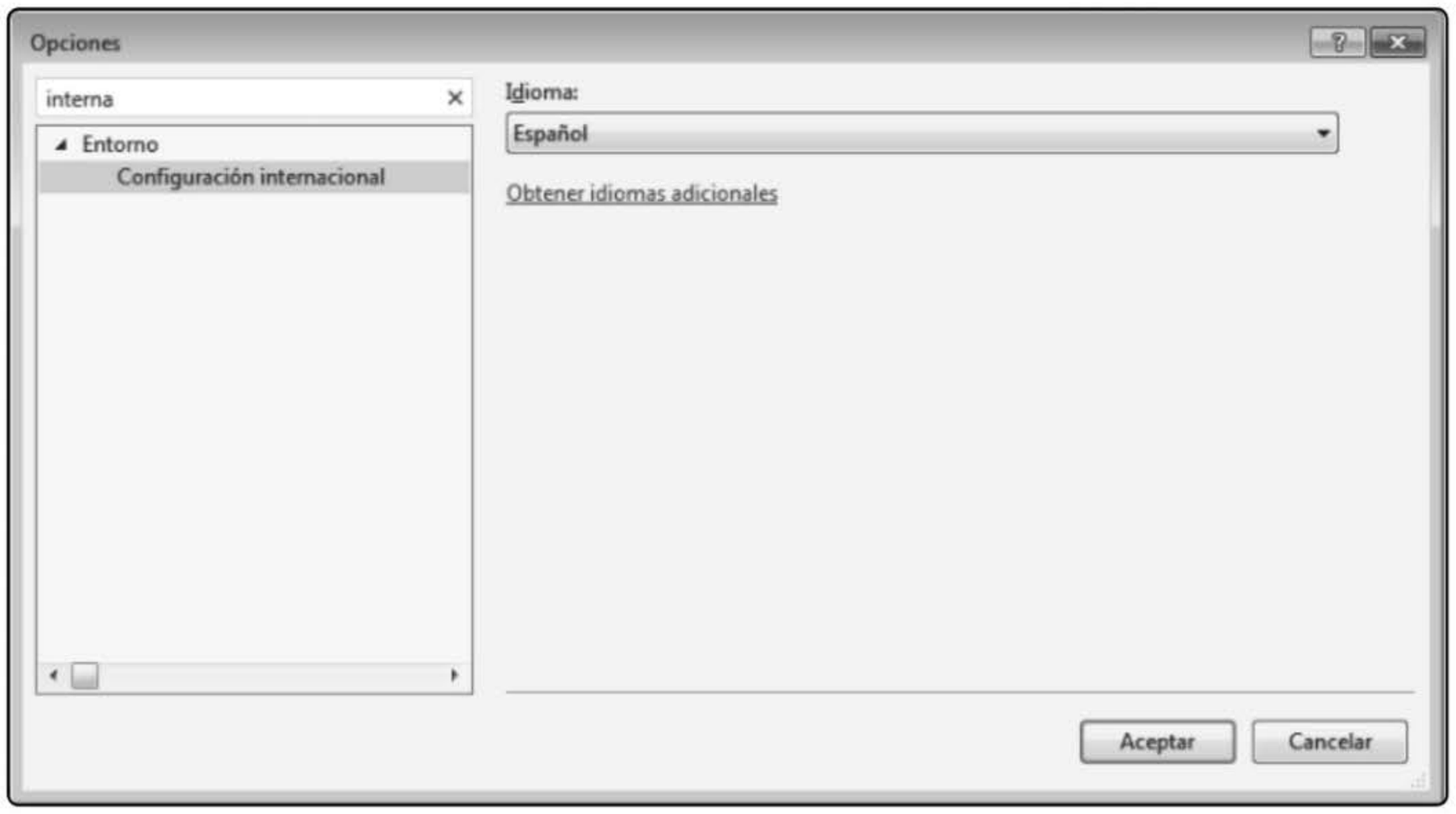
El entorno de Visual Studio 2015 ofrece la posibilidad de modificarlos en diferentes idiomas, para esto se debe descargar el paquete de idiomas de Microsoft Visual Studio 2015.

URL de descarga:

<https://www.microsoft.com/es-ES/download/details.aspx?id=48157>

La instalación del paquete de idioma es sencilla, será cuestión de tener el Visual Studio 2015 cerrado al momento de la instalación. Una vez instalado se pueden realizar los siguientes pasos desde Visual Studio 2015:

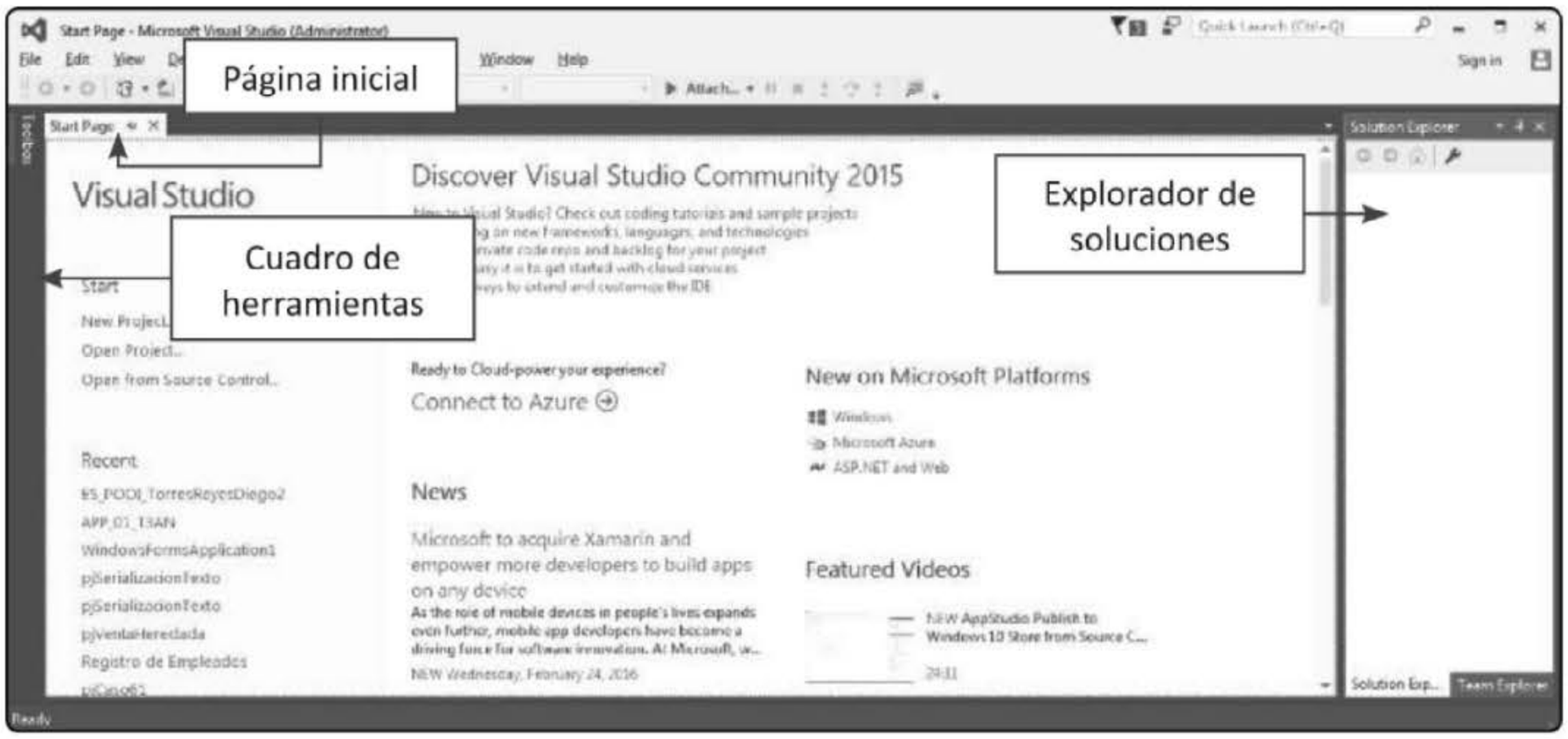
- Seleccione **Tools > Option**.
- Escriba la palabra <internacional> en el cuadro de texto.
- Seleccione **Configuración internacional**.
- Seleccione el idioma adecuado; en este caso debe seleccionarse **Español** y, finalmente, presione el botón **Aceptar**.



1.3 IDE Visual Studio 2015

1.3.1 Pantalla inicial

La pantalla de inicio es la primera que se visualiza al comienzo de Visual Studio; esto se realiza de forma automática. Para poder visualizarla seleccione la pantalla de inicio. Seleccione **Ver > Pagina de Inicio (View > Start Page)**.



Opciones rápidas que se pueden emplear en la pantalla inicial del Visual Studio 2015

Mostrar y ocultar el cuadro de herramientas

- <Ctrl + Alt + X>

Mostrar y ocultar el explorador de soluciones

- <Ctrl + Alt + L>

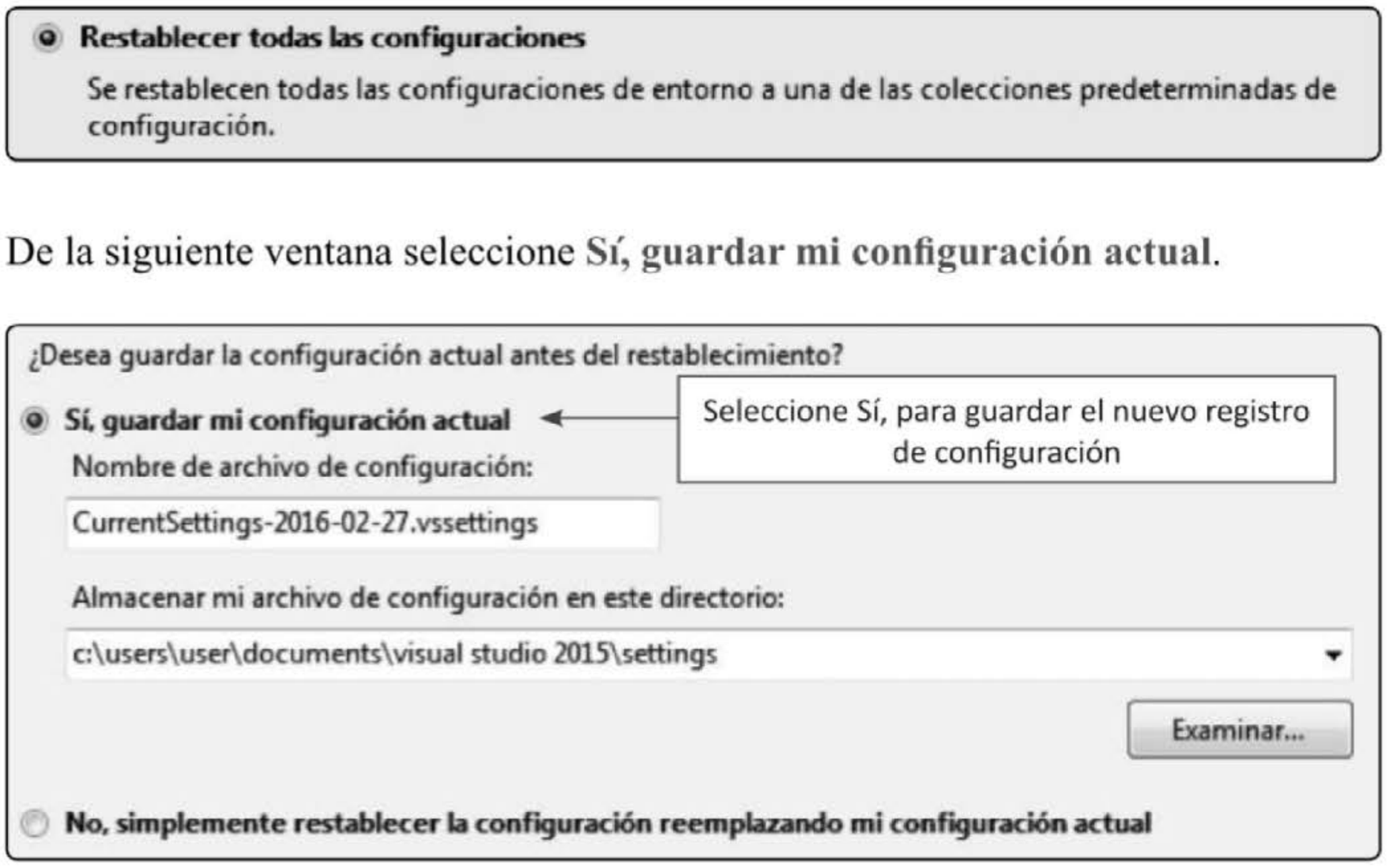
Mostrar y ocultar las propiedades

- <F4>

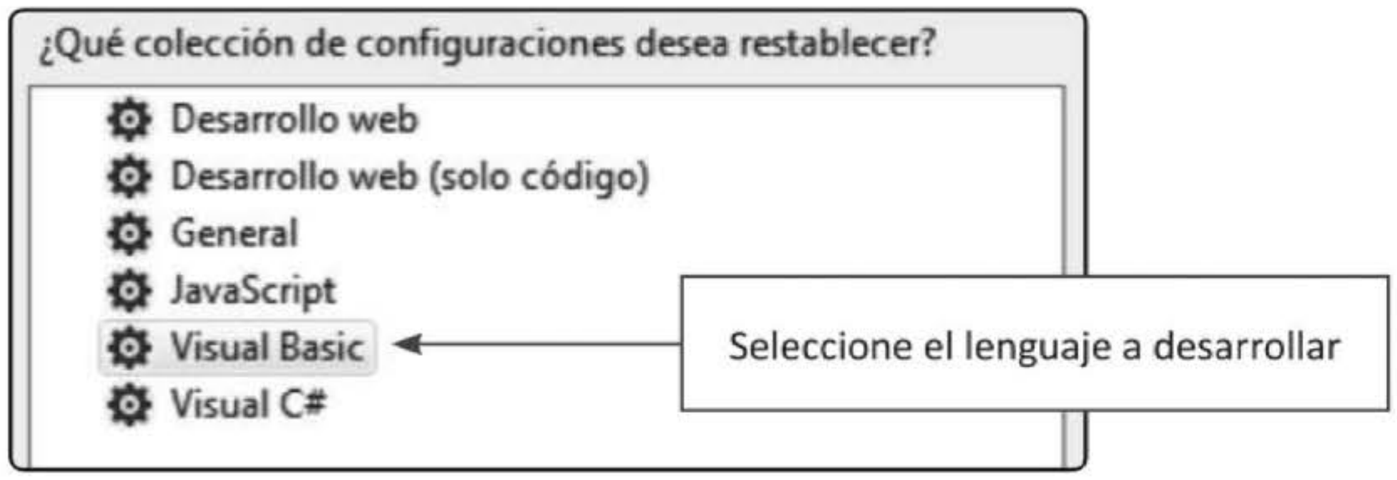
1.3.2 Selección del lenguaje a desarrollar

La característica principal del Visual Studio 2015 es la multiplataforma, por lo tanto, antes de comenzar a desarrollar aplicaciones se debe seleccionar el lenguaje de programación Visual Basic, el cual se desarrollará en este material:

- **Herramientas > Importar y exportar configuraciones**.
- Seleccione **Restablecer todas las configuraciones**.



- De la siguiente ventana, seleccione Visual Basic y, luego, el botón **Finalizar**.



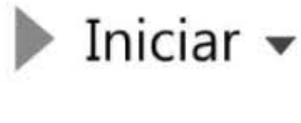
Tenga en cuenta que la configuración realizada solo se debe realizar una vez, ya que esta quedará como un valor predeterminado en Visual Studio 2015.

1.3.3 Barra de herramientas estándar

Esta barra permitirá guardar, abrir, ejecutar y pausar una aplicación; asimismo se podrá asignar comentarios a un bloque de código. Solo se debe tener en cuenta qué botones realizan dichas actividades.

- Acceso: **Ver > Barra de Herramientas > Estándar**

Descripción de los principales botones:

Ícono	Herramienta	Descripción
	Navegar atrás Navegar adelante	Si se encuentra en el editor de código, el botón de navegador desplazará el cursor por las líneas de código anteriormente seleccionadas. En caso de encontrarse en otro entorno, entonces el botón de navegador lo desplazará entre dichos entornos.
	Nuevo	Presenta dos opciones: la primera permite crear un nuevo proyecto y la segunda, crear un nuevo sitio web.
	Abrir	Permite abrir cualquier tipo de archivo perteneciente a una aplicación desarrollada en Visual Studio.
	Guardar	Permite grabar el desarrollo del entorno actual.
	Guardar todo	Permite grabar el desarrollo efectuado en todo el proyecto. Se recomienda seleccionar este botón cuando se quiere guardar todos los componentes del proyecto, ya sea Windows Forms, códigos, clases, etc.
	Comentario	Permite asignar y quitar un comentario a una línea de código; solo cuando se requiera en el editor de código.
	Iniciar o Ejecutar	Permite ejecutar una aplicación y así visualizar el desarrollo correcto de nuestros proyectos; otra forma de ejecutar una aplicación es presionando la tecla <F5>.

1.3.4 Cuadro de herramientas

Muestra controles según el entorno de desarrollo seleccionado; estos permitirán colocar controles en cualquier entorno. En este caso se trabajará con controles para aplicaciones Windows Forms.

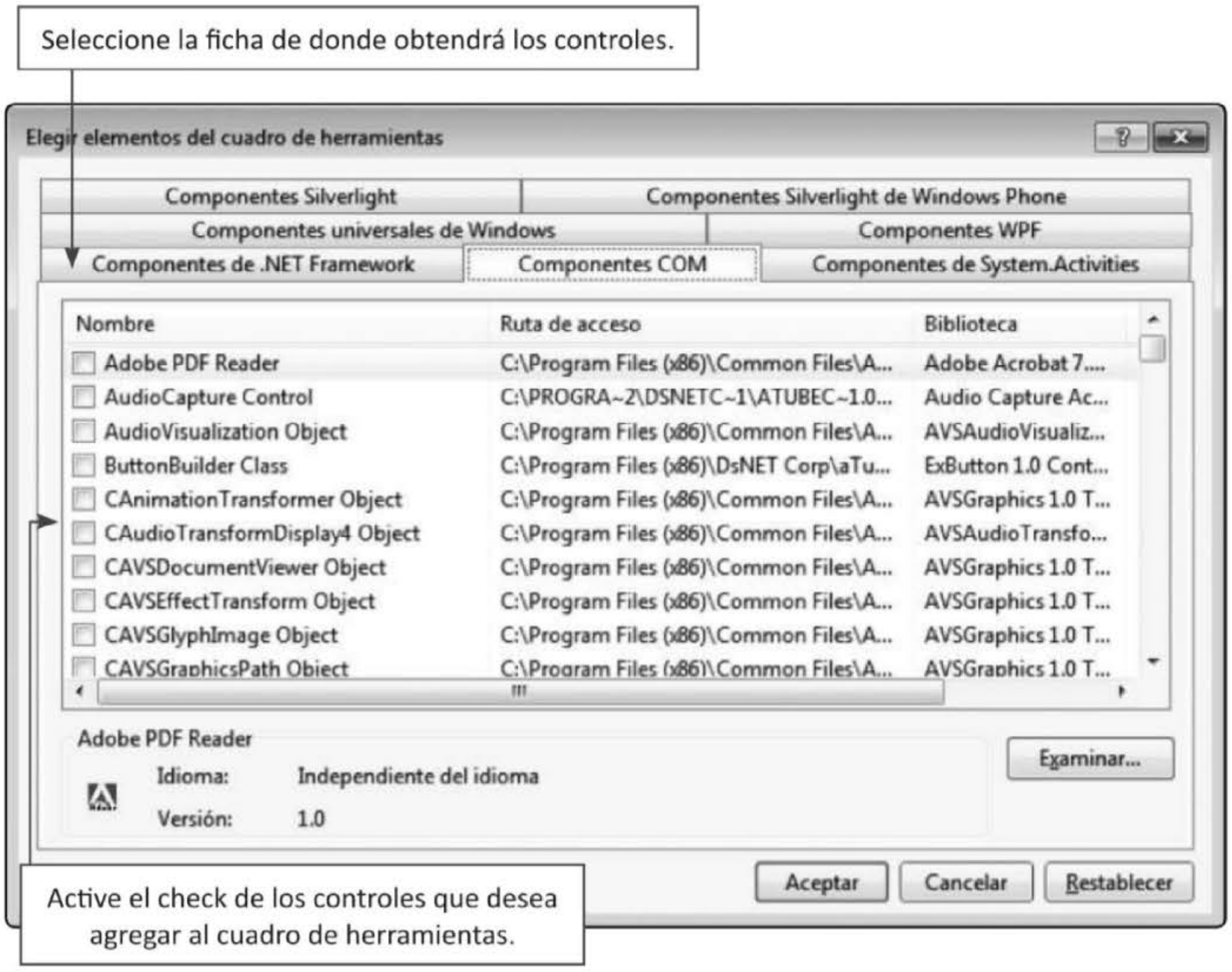
- Acceso : **Ver > Cuadro de herramientas**
- Tecla de acceso : <Alt + Ctrl + X>

Entre sus principales fichas están:

Categoría	Descripción
Todo Windows Forms	Presenta una lista de controles referente al desarrollo de aplicaciones Windows Forms.
Controles comunes	Presenta una lista de los controles más usados en el desarrollo de aplicaciones Windows Forms como botones, cuadros de chequeo, cuadros combinados, etiquetas, cuadros de lista, cuadros de texto, etc.
Contenedores	Presenta una lista de controles que pueden contener a otros controles como un grupo, panel o un control de tabuladores.
Menús y barras de herramientas	Presenta una lista de controles que pueden controlar opciones agrupadas como un menú de opciones, barra de herramientas, barra de estado, etc.
Datos	Presenta una lista de controles que permiten controlar la información a partir de una conexión a un gestor de base de datos, como un conjunto de datos, DataGridView o un gráfico.

Es necesario tener en cuenta que Visual Studio presenta muchos controles, los cuales no pueden ser mostrados en el cuadro de herramientas, por lo tanto se puede adicionar controles realizando los siguientes pasos:

- Haga clic derecho sobre el cuadro de herramientas.
- Seleccione **Elegir elementos** y siga los siguientes pasos.



Opciones que se presentan en el cuadro de herramientas:

Crear un nuevo grupo de herramientas personalizadas

- Haga clic derecho sobre las herramientas.
- Seleccione **Agregar pestaña** y asígnele un nombre.

Agregar controles a la pestaña desde el cuadro de herramientas

- Haga clic derecho sobre las herramientas de una determinada pestaña.
- Seleccione **Elegir elementos**.
- Seleccione un control desde las fichas.

Eliminar una pestaña desde el cuadro de herramientas

- Haga clic derecho sobre la pestaña.
- Seleccione **Eliminar pestaña**.

Colocar controles repetidos en un pestaña personalizada

- Haga clic derecho sobre un control.
- Seleccione la opción **Copiar**.
- Haga clic derecho sobre la ficha.
- Seleccione la opción **Pegar**.

Restablecer los controles originales del cuadro de herramientas

- Haga clic derecho sobre un control.
- Seleccione la opción **Restablecer cuadro de herramientas**.

1.3.5 Explorador de soluciones

Mediante el uso del Explorador de soluciones, puede administrar los elementos de una solución o un proyecto; asimismo puede examinar el código. En función del elemento especificado, el tipo de proyecto determinará los comandos de administración que van a estar disponibles en las ventanas.

- Acceso : **Ver > Explorador de soluciones**
- Tecla de acceso : <Ctrl + Alt + L>

Entre sus principales herramientas están:

Botón	Descripción
	Permite restaurar los elementos originales del Explorador de soluciones y lo ubica en el archivo principal de la solución o proyecto.
	Permite contraer la lista de archivos contenidos en las opciones del Explorador de soluciones.
	Permite mostrar todos los elementos visibles y no visibles del Explorador de soluciones.
	Permite mostrar las propiedades pertenecientes a la solución o proyecto.
	Permite visualizar el código cuando nos encontramos en un proyecto de tipo Windows Forms.
	Permite actualizar los elementos mostrados en el Explorador de soluciones.

Los archivos que contiene el Explorador de soluciones:

Botón	Descripción
Solución 'Sistema de Control de Ventas' (2 proyectos)	Es el nombre de la solución actual, entre paréntesis siempre se mostrará la cantidad de proyectos que contiene la solución. El valor por defecto es uno, ya que al crear una solución automáticamente tiene un proyecto.
pjEjercicio01	Es el nombre del proyecto actual; es a partir de aquí donde se controlan todos los archivos contenidos.
Referencias	Muestra todas las referencias activas para un determinado proyecto de aplicación.
App.config	Muestra el contenido del archivo de configuraciones.
Form1.vb	Muestra el contenido del formulario perteneciente al proyecto.

Operaciones básicas desde el Explorador de soluciones:

Agregar un elemento a la solución

- Haga clic sobre la solución.
- Haga clic derecho sobre cualquier sección del Explorador de soluciones.
- Seleccione **Agregar**.

Agregar un elemento al proyecto

- Haga clic sobre el proyecto.
- Haga clic derecho sobre cualquier sección del Explorador de soluciones.
- Seleccione **Agregar**.

Eliminar un proyecto

Tener en cuenta que solo se podrá eliminar si hay más de un proyecto en la solución.

- Haga clic derecho sobre un determinado proyecto.
- Seleccione **Quitar**.

Establecer un proyecto de inicio

Tener en cuenta que solo podrá establecer como proyecto de inicio si hay más de un proyecto en la solución.

- Haga clic derecho sobre un determinado proyecto.
- Seleccione **Establecer como proyecto de inicio**.

1.3.6 Ventana de propiedades

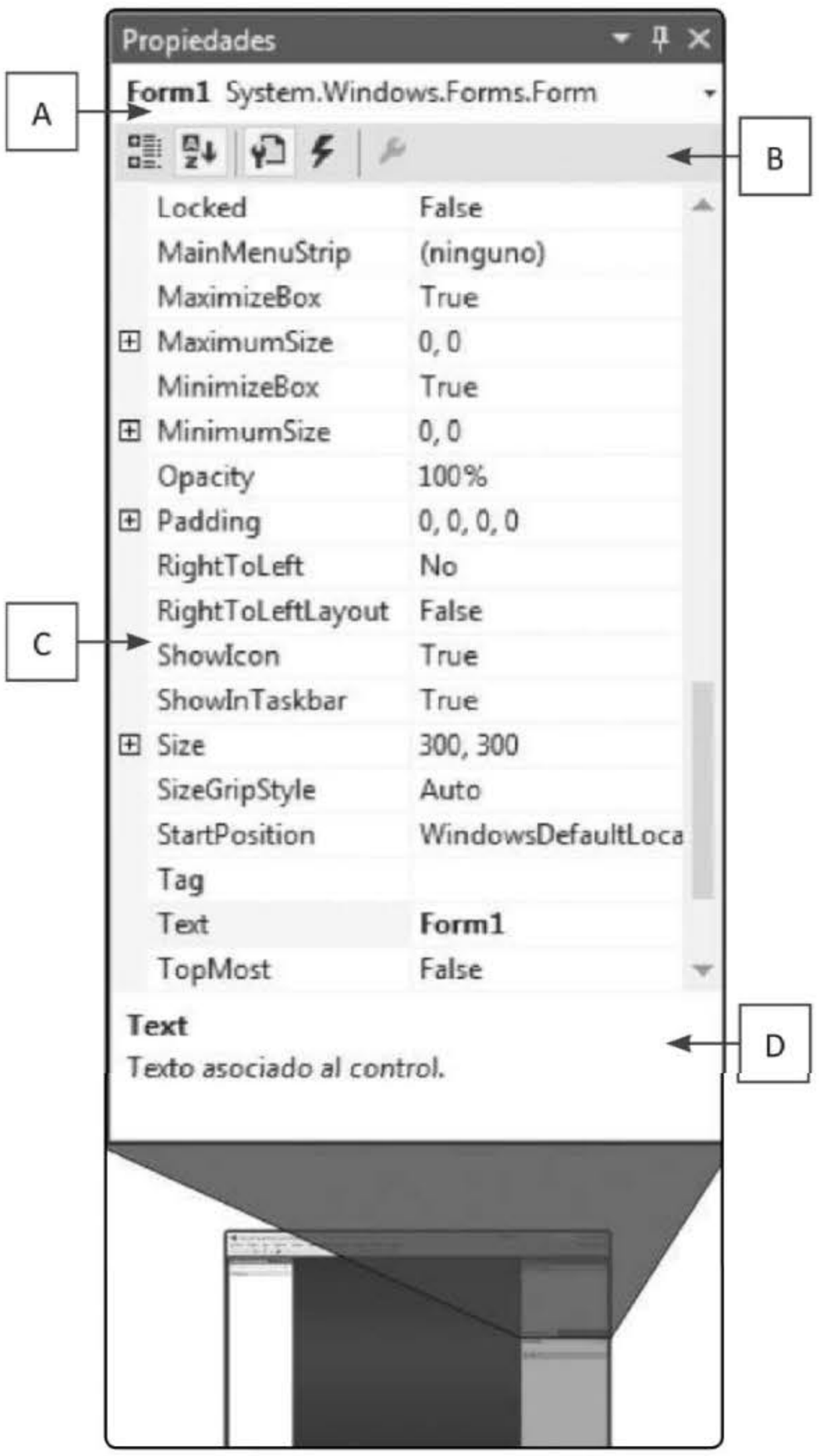
Esta ventana presenta una lista de propiedades que permite personalizar un control. También se debe considerar que dicha ventana mostrará propiedades dependiendo de lo seleccionado, ya que también se puede mostrar propiedades de los archivos contenidos en el Explorador de soluciones.

- Acceso:

  Ver > Ventana propiedades

- Tecla de acceso:

  <F4>



Partes de la ventana de propiedades:

	Botón	Descripción
A		Form1 representa el nombre del objeto seleccionado. **System.Windows.Forms.Form** representa la clase del cual se está mostrando el objeto.
B		• Mostrando las propiedades en forma de categorías. • Mostrando las propiedades ordenadas de forma ascendente (predeterminado). • Mostrando las propiedades de un determinado objeto. • Mostrando los eventos de un determinado objeto.
C		El símbolo + indica que una propiedad tiene más opciones que pueden ser desplegadas. Si necesita cambiar el valor de una propiedad puede dar doble sobre el nombre de la propiedad. Finalmente, el valor de la propiedad aparece en el lado derecho, el cual puede ser personalizado por el usuario; también se puede modificar haciendo doble clic sobre el valor.
D		En el pie de la ventana de propiedades se muestra un diálogo que indica algunas características de la propiedad seleccionada.

Operaciones básicas desde las propiedades:

Modificar el valor de una propiedad

Esto se da cuando la propiedad tiene valores de modificación, no cuando la propiedad presenta una subventana.

- Ubique la propiedad.
- Haga doble clic sobre sí misma.

Implementar código en un evento del control

En ocasiones se necesitará acceder a un evento especial del control, para lo cual primero se debe conocer la capacidad que tiene cada evento en el control y después realizar los siguientes pasos:

- Haga clic sobre el control desde el formulario.
- Seleccione **Eventos**.
- Haga doble clic en uno de los eventos.

1.4 Soluciones y proyectos Visual Studio 2015

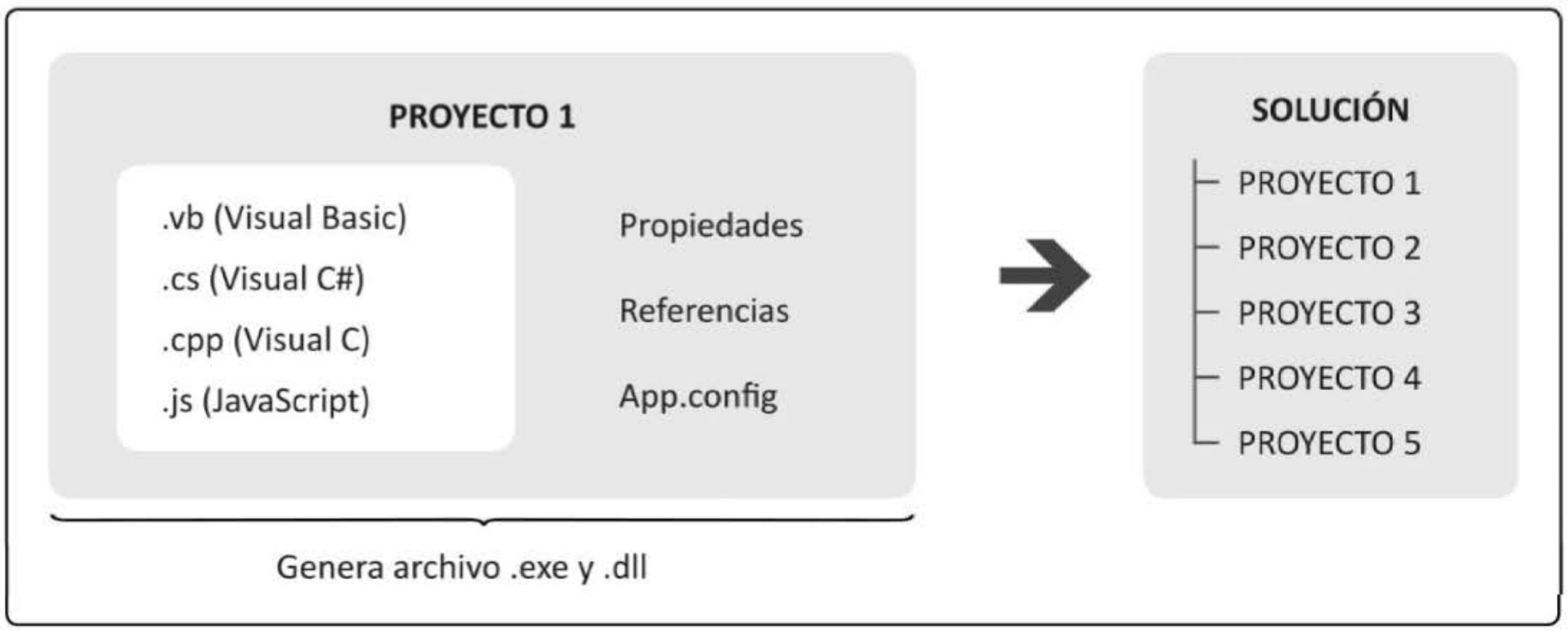
En Visual Studio se puede crear aplicaciones, sitios web, aplicación web, script, aplicación móvil, etc., y en todos ellos siempre se debe empezar por un proyecto. Inicialmente, un proyecto tiene los siguientes componentes:

- Archivo con código fuente.
- Íconos usados en una aplicación.
- Imágenes usadas en una aplicación.
- Archivo de datos, el cual contiene información relevante para la aplicación, así como la configuración del compilador.

Todos estos elementos son necesarios para que se ejecute una aplicación implementada en Visual Studio. Es necesario tener en cuenta que un proyecto es un archivo de tipo XML, el cual contiene información muy importante para la compilación de la aplicación; este archivo de proyecto tiene la extensión VBPROJ. Otros dos aspectos principales son:

- Al compilar una aplicación, el motor de MSBuild (Microsoft Build Engine) proporciona un esquema XML, el cual contiene un archivo de proyecto que controla el proceso de compilación de un proyecto que finalmente construye un archivo ejecutable. Todo proyecto de Visual Studio importa todas las configuraciones y procesos de compilación para realizar el trabajo de desarrollo necesario para la ejecución correcta.
- Todos los proyectos que se implementen en Visual Studio están contenidos en una solución, con ello un conjunto de información que permite una correcta compilación. En síntesis, una solución es un archivo de texto con su propio formato y no permite modificar su contenido; esta tiene como extensión **.sln**.

El siguiente diagrama muestra la relación entre proyectos y soluciones, y los elementos que estos contienen de forma lógica.

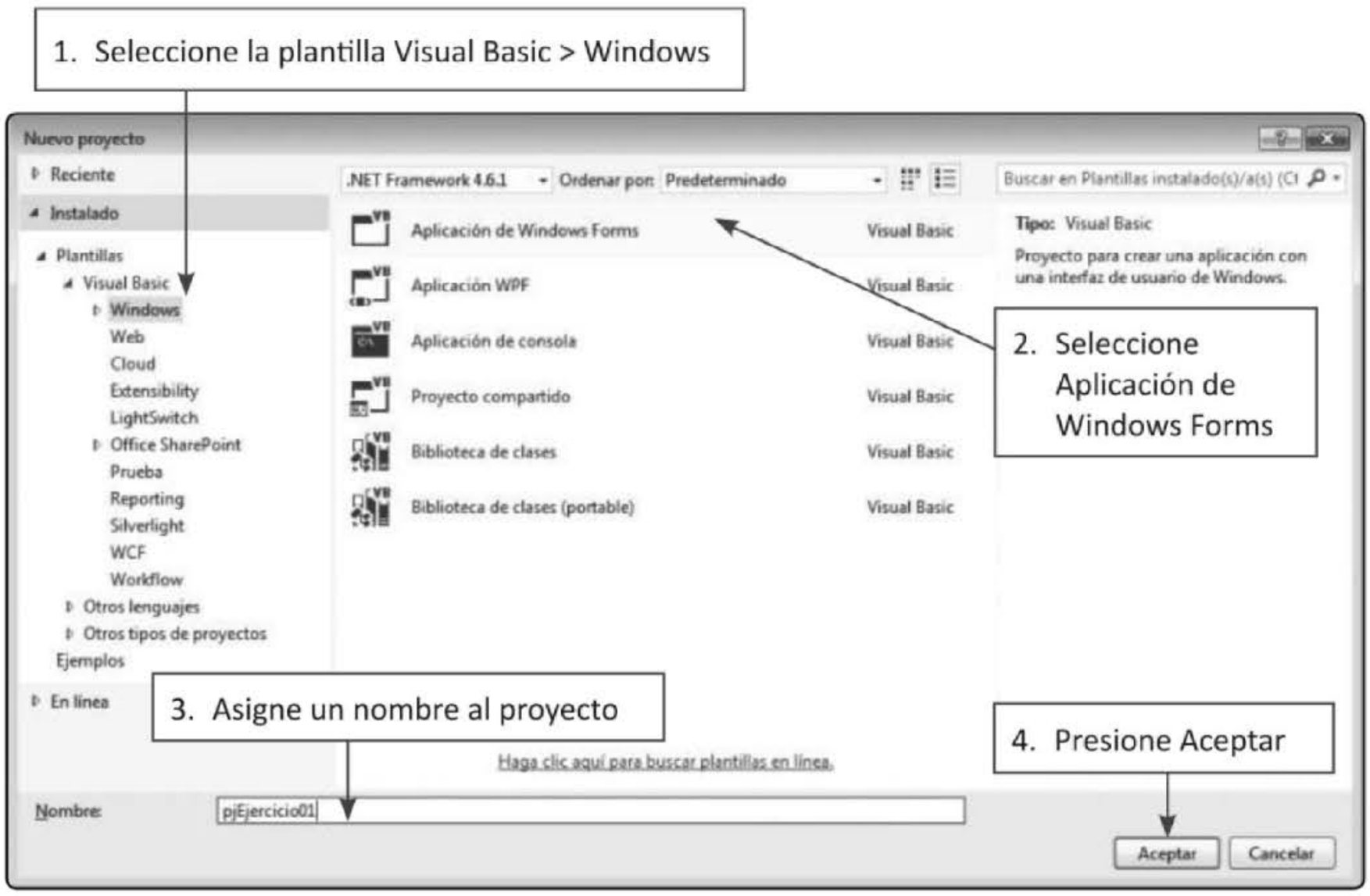


1.4.1 Creando un nuevo proyecto de aplicación para Visual Basic

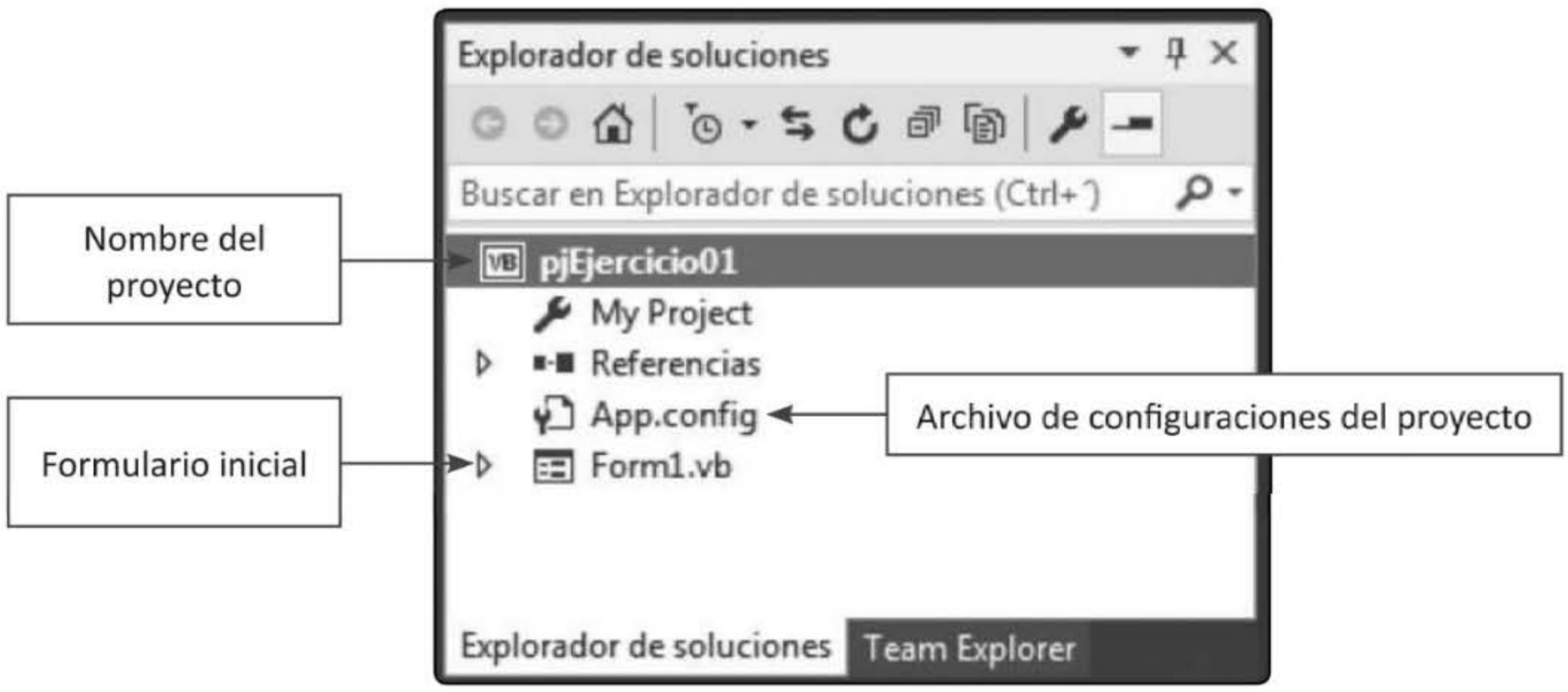
La creación de proyectos será la forma de representar una aplicación en todo el material expuesto, de tal forma que estos pasos se repetirán para todos los casos desarrollados. Debido a que se desarrollarán aplicaciones de plataforma se seleccionará aplicaciones de tipo Windows Forms de la plantilla Visual Basic.

Para la creación de un proyecto nuevo:

- Siga la ruta: **Archivo > Nuevo proyecto** y siga los pasos señalados en el cuadro.



En el Explorador de soluciones se puede visualizar los archivos iniciales que contiene un proyecto.



Se debe tener en cuenta que un proyecto se crea en base a una solución, la cual aún no se visualiza en el Explorador de soluciones; esta se visualizará al agregar un proyecto nuevo a la solución. Mientras tanto, se analiza el contenido mostrado:

Elementos	Descripción
My Project	Contiene información que puede ser administrada en referencia al proyecto; por ejemplo, desde aquí se puede definir el formulario de inicio o cambiar el nombre del proyecto.
Referencias	Contiene información de todas las librerías que inicialmente presenta un proyecto, desde aquí se puede agregar o eliminarlos. Las librerías necesarias para la implementación de la aplicación aparecen de forma predeterminada.
App.config	Contiene información de tipo XML que pertenece a la configuración de la aplicación.
Form1.vb	Es el formulario por defecto al crear una aplicación de tipo Windows Form.

1.4.2 Guardando un proyecto de aplicación Visual Basic

En Visual Basic se puede implementar y ejecutar una aplicación sin necesidad de guardarla, pero se debe considerar que si por algún motivo se cierra el entorno Visual, probablemente la aplicación se pierda. Por lo tanto, se debe guardar el proyecto cada cierto tiempo.

Para guardar el proyecto:

- Siga la ruta **Archivo > Guardar todo** (<Ctrl+Shift+S>) y siga los siguientes pasos:

Inicialmente, el nombre del proyecto es el mismo especificado al momento de crearlo. Lo que se debe especificar es la ubicación del proyecto mediante el botón **Examinar**. Observe que el nombre de la solución es la misma especificada en el proyecto.

Como toda aplicación Microsoft existe una carpeta por defecto en donde se guardan todos los proyectos que se realiza en Visual Basic, esto se da solo si no se selecciona una ubicación del proyecto al momento de guardar todo.

Ruta de la carpeta por defecto:

```
C:\Users\user\Documents\VISUAL STUDIO 2015\PROJECTS
```

1.4.3 Abriendo un proyecto de aplicación Visual Basic

Los proyectos por defecto se almacenan dentro de la carpeta Projects. A continuación, los pasos para abrir un proyecto desde esta carpeta.

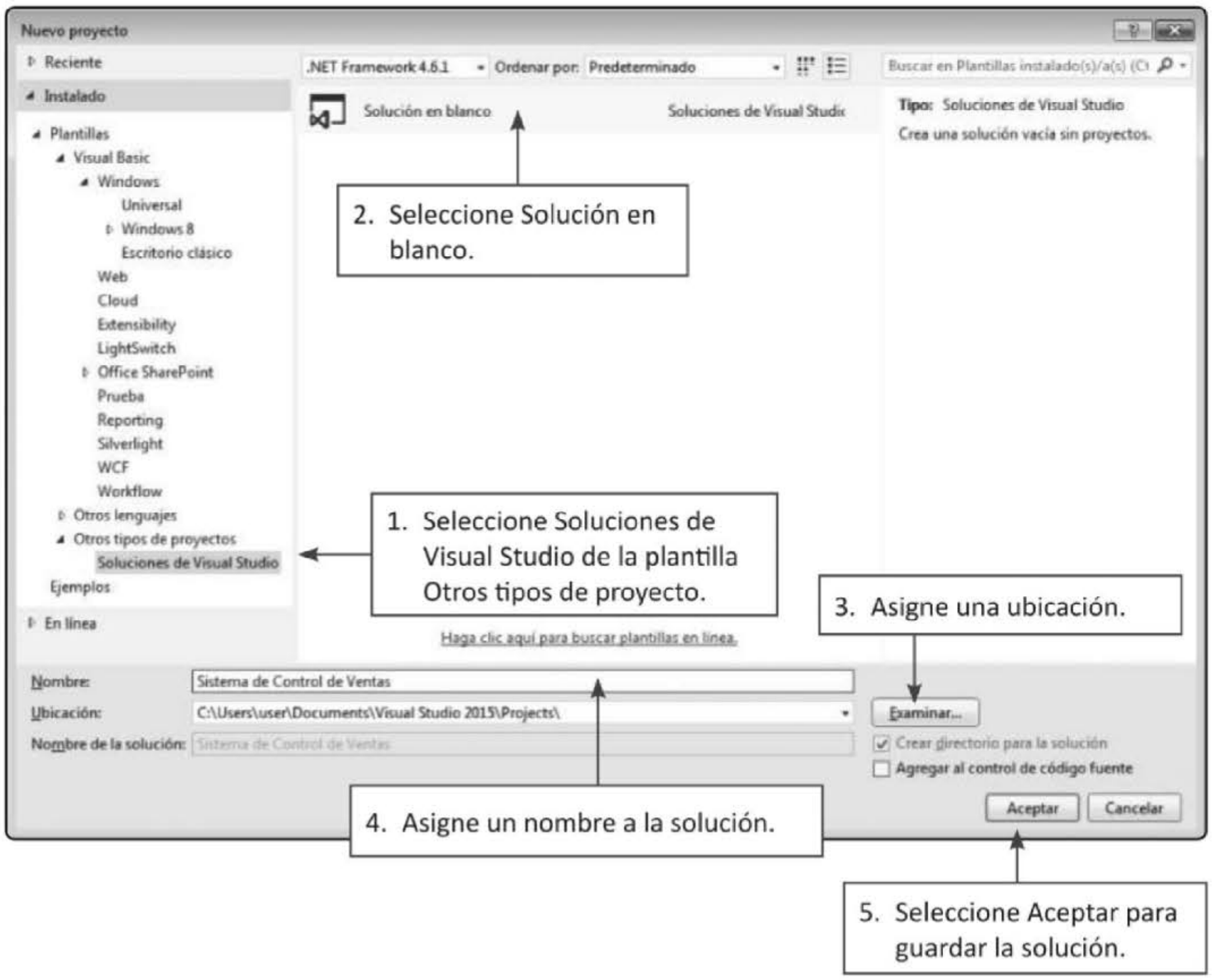
Paso 1: Ingresar a la carpeta del proyecto.

Paso 2: Seleccione el archivo con extensión **.sln** para lograr visualizar todo el proyecto.

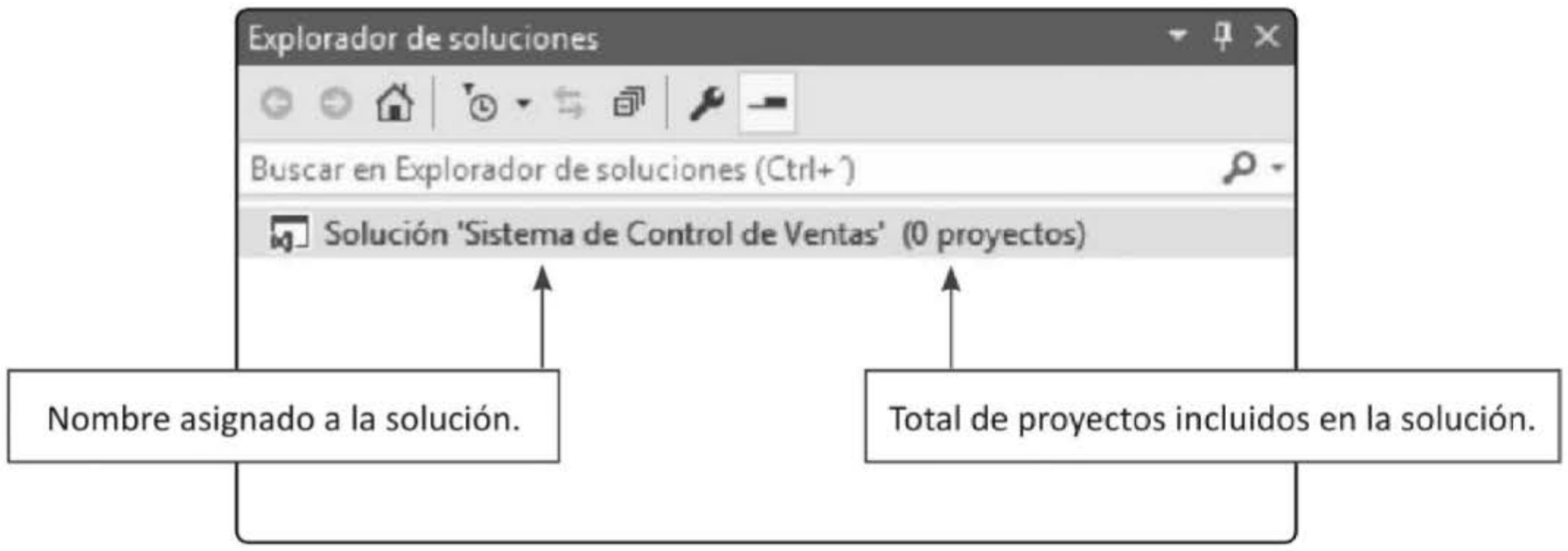
1.4.4 Creando una solución vacía

Una solución puede contener uno o más proyectos; por lo tanto, para crear una solución vacía es necesario hacer lo siguiente:

- Siga la ruta **Archivo > Nuevo proyecto**, y realice los siguientes pasos:



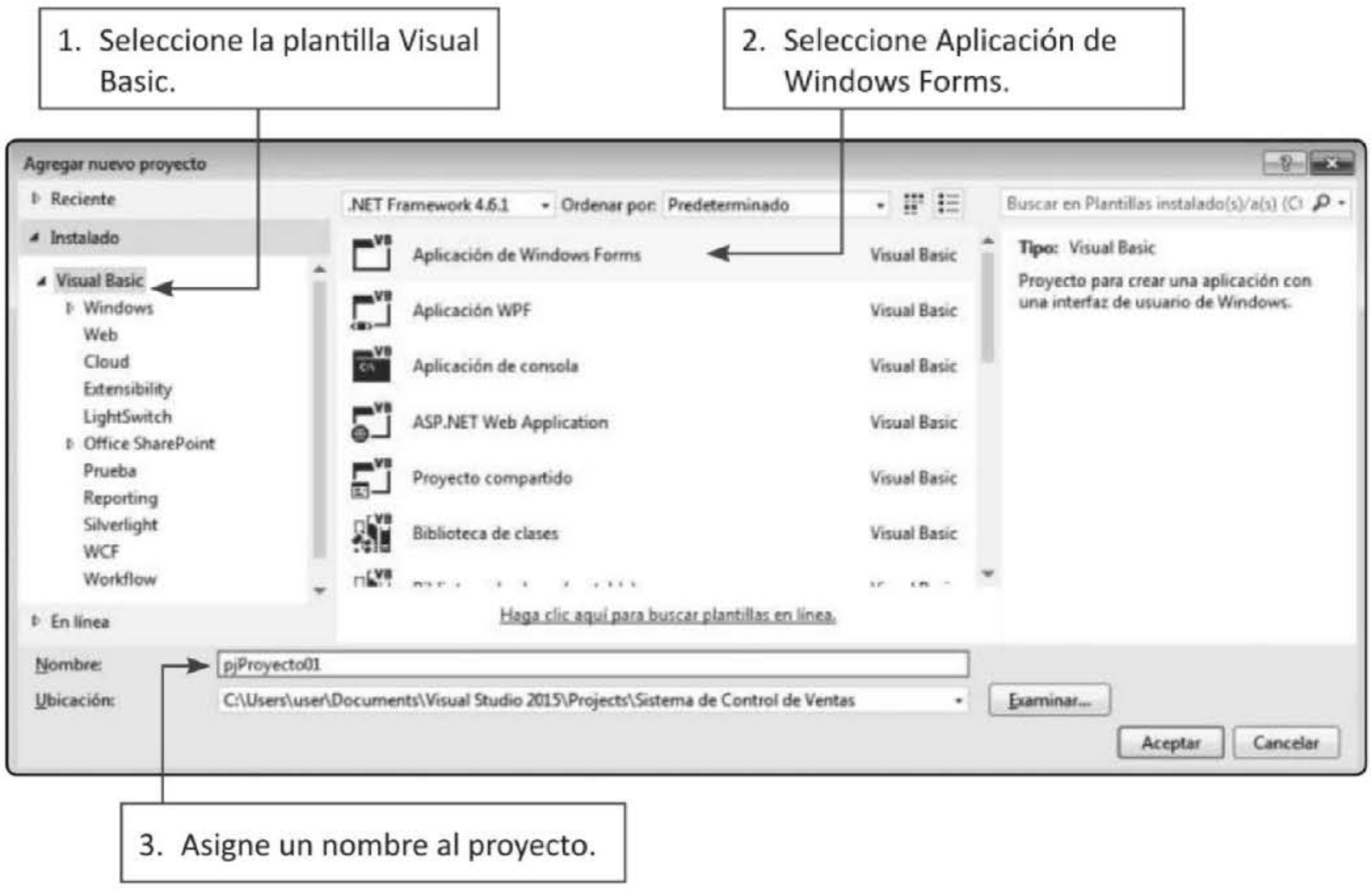
En la ventana Explorador de soluciones se muestra de la siguiente manera:



Como se observa, entre paréntesis muestra la cantidad de proyectos que tiene actualmente la solución. A continuación, los pasos para agregar un proyecto a una solución activa:

- Haga clic derecho sobre la solución desde la ventana Explorador de soluciones.

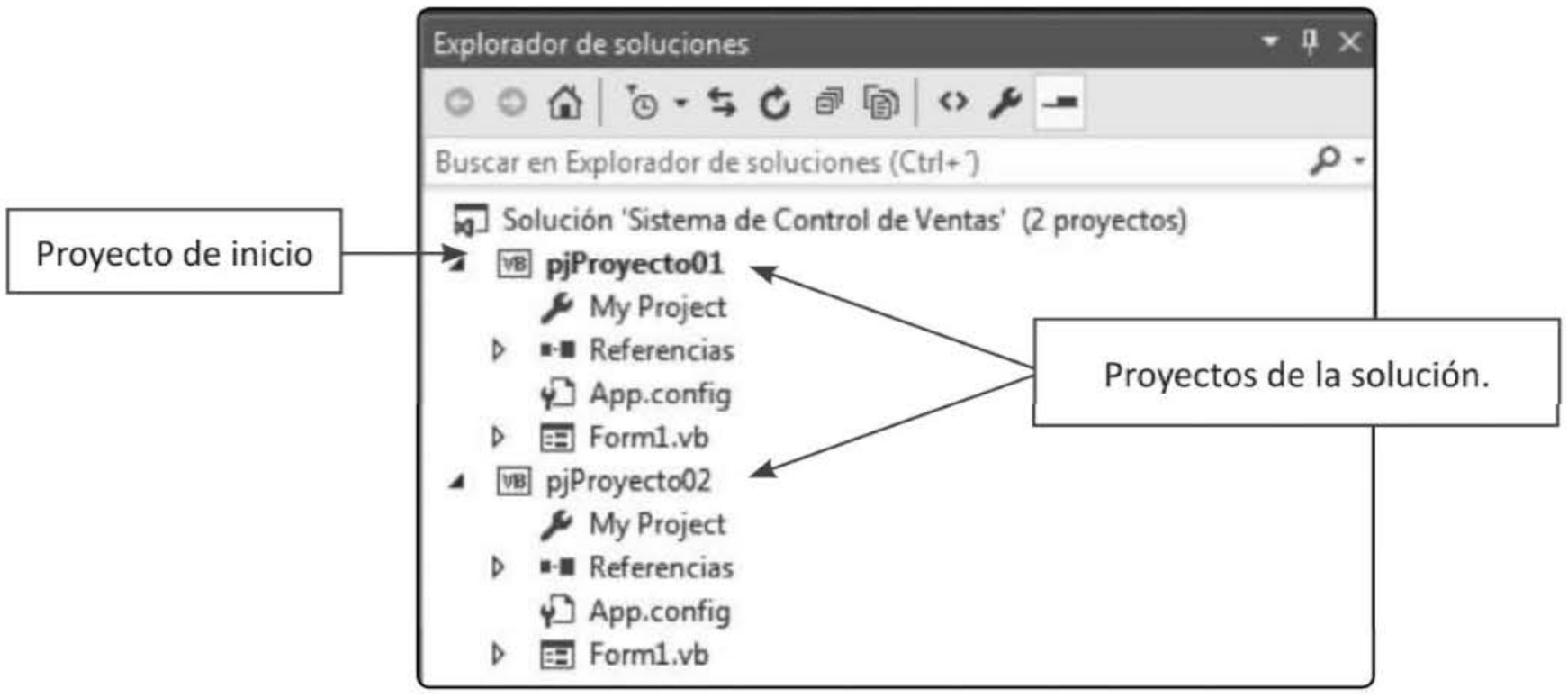
- Seleccione **Agregar > Nuevo proyecto** y realice los siguientes pasos:



Se podría optar por una segunda forma para agregar un proyecto a la solución realizando los siguientes pasos:

- Desde el menú seleccione **Archivo > Agregar > Nuevo proyecto**
- Asigne un nombre al proyecto y presione el botón **Aceptar**

Finalmente, se puede observar que al agregar dos proyectos a la solución el Explorador de soluciones muestra el siguiente aspecto:



Consideraciones con respecto a los proyectos dentro de una solución:

- Si dentro de una solución se encuentran dos o más proyectos, uno de ellos tiene la asignación de proyecto de inicio que puede ser modificado según la necesidad del usuario.
- Para detectar al proyecto de inicio visualice los proyectos desde el Explorador de soluciones y notará que uno de ellos se encuentra en negrita. Ese es el proyecto de inicio, como se muestra en la imagen anterior.
- Debe tener en cuenta que el primer proyecto agregado a la solución siempre será el proyecto de inicio.
- Al eliminar una solución completa se estaría eliminando todos los proyectos contenidos en él.
- Normalmente se usa una solución para aplicaciones que trabajen con capas de programación; en la cual cada proyecto representará a una capa.

Operaciones básicas en la solución:

Agregar un proyecto

- Haga clic derecho sobre la solución.
- Seleccione **Agregar > Nuevo proyecto**.

Eliminar un proyecto

- Haga clic derecho sobre un determinado proyecto.
- Seleccione **Quitar**.

Establecer un proyecto de inicio

- Haga clic derecho sobre un determinado proyecto.
- Seleccione **Establecer como proyecto de inicio**.

Cerrar un proyecto

- Siga la ruta **Archivo > Cerrar proyecto**.

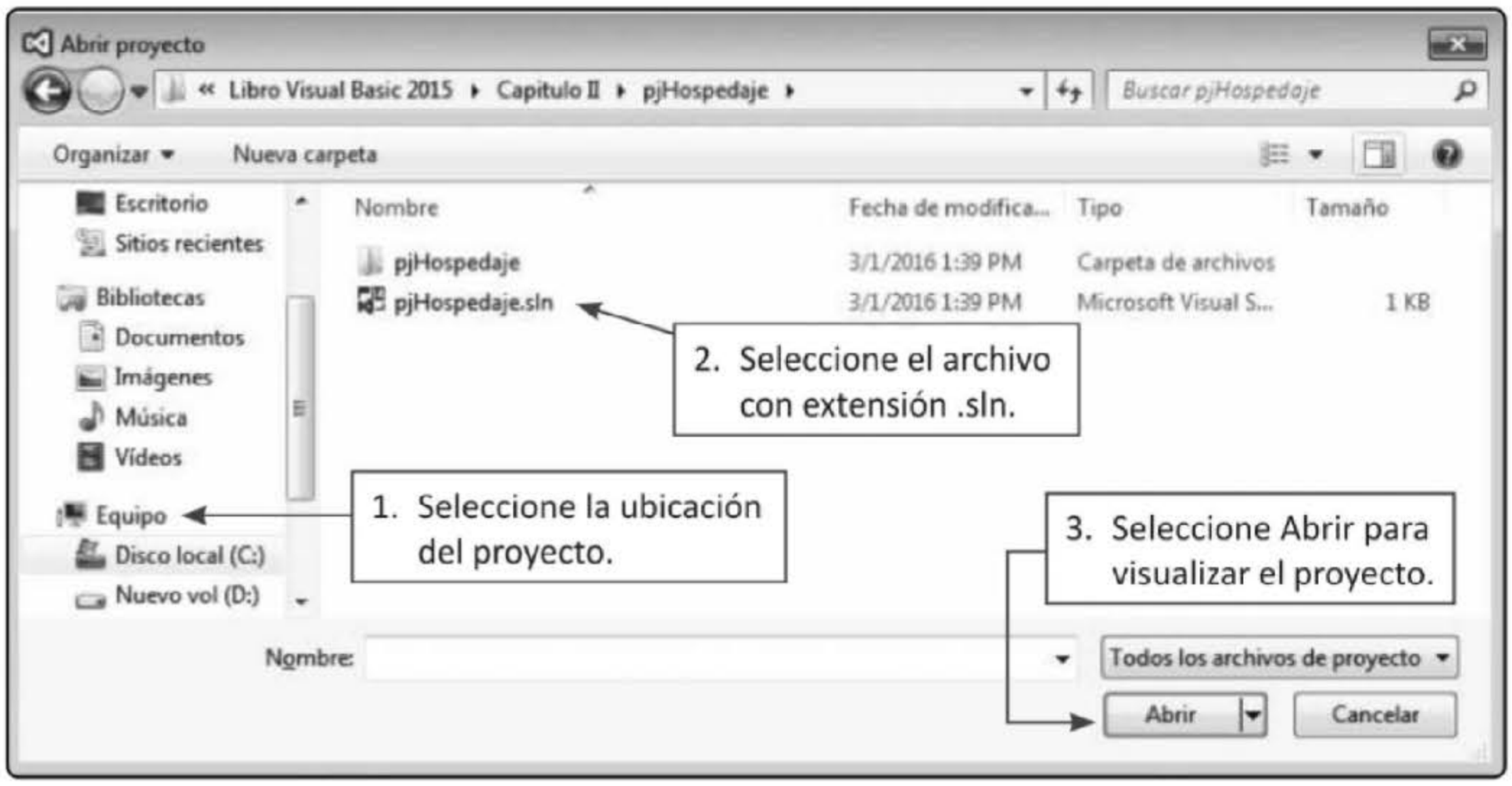
Cerrar una solución

- Siga la ruta **Archivo > Cerrar solución**.

1.4.5 Abriendo una solución de Visual Basic

Si se tiene en cuenta que uno o más proyectos pueden estar contenidos en una misma solución; cuando se necesite abrir un proyecto contenido en una solución en realidad se tendrá que abrir la solución, realizando los siguientes pasos:

- Siga la ruta **Archivo > Abrir proyecto** (<Ctrl+O>) y realice los siguientes pasos:

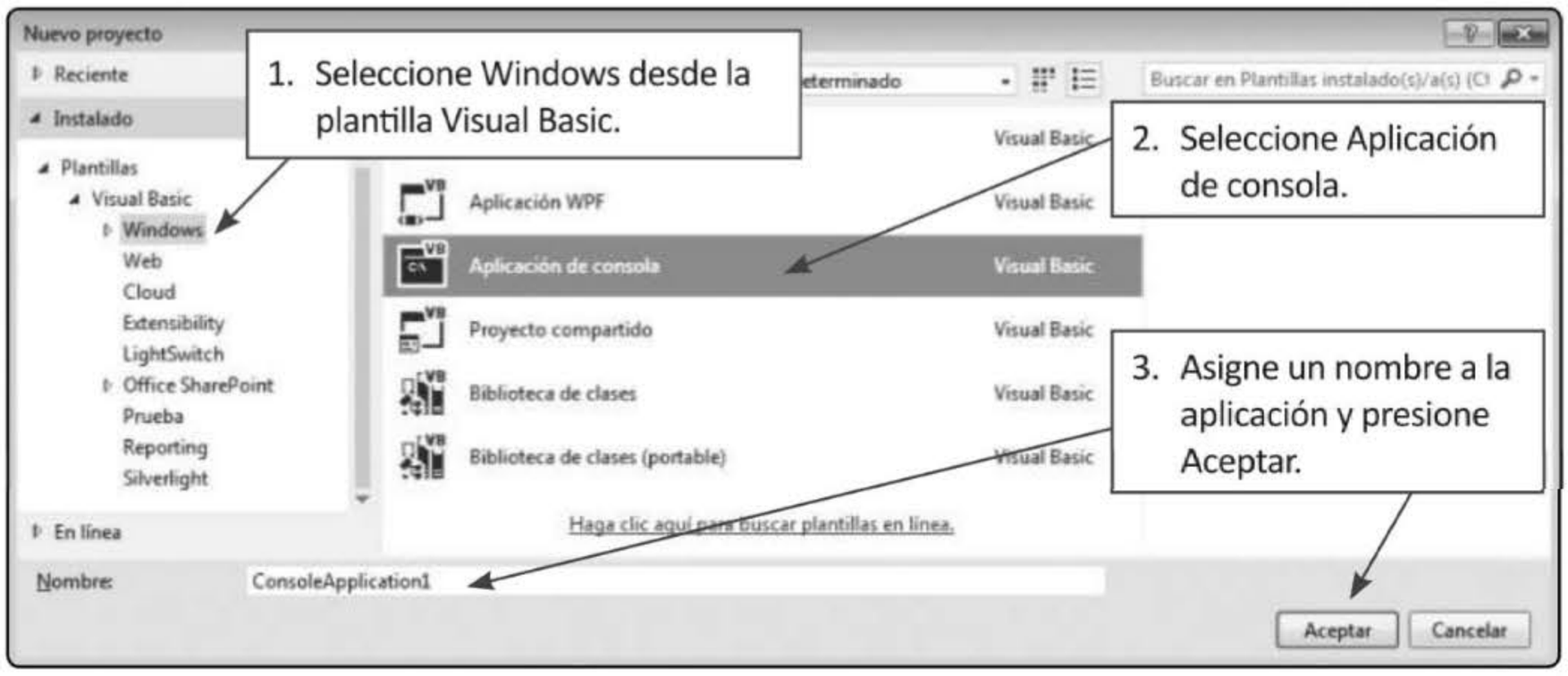


Recuerde que la ubicación por defecto de las aplicaciones se encuentra en **Mis Documentos > Visual Studio 2015 > Projects**.

1.5 Desarrollo de aplicaciones tipo consola

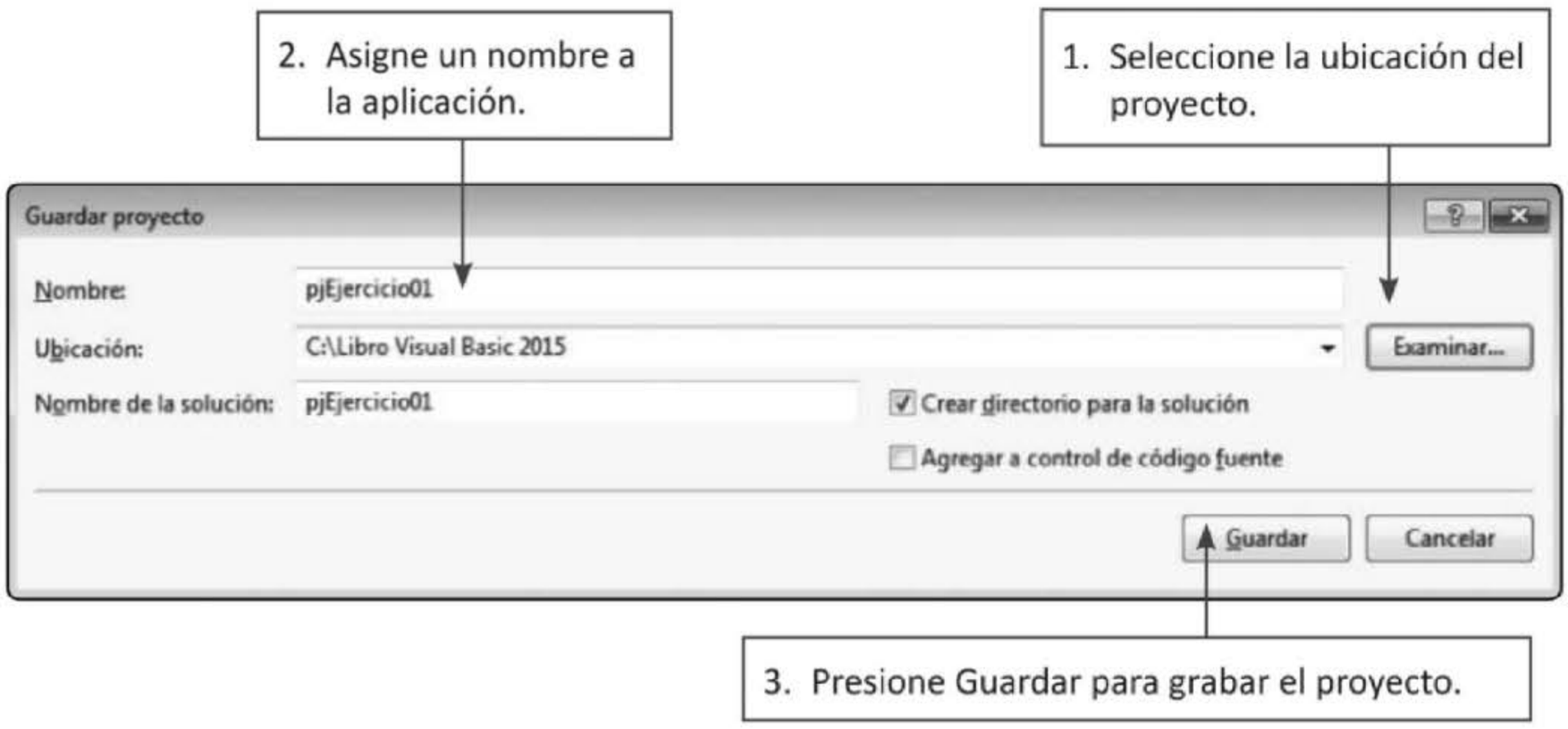
Ahora se implementarán aplicaciones que se concentren en la lógica de programación y no en el modo visual; esto se desarrollará con aplicaciones de tipo consola. Para crear una aplicación se debe hacer lo siguiente:

- Siga la ruta **Archivo > Nuevo proyecto** y realice los siguientes pasos:



1.5.1 Guardar el proyecto de tipo consola

- Siga la ruta **Archivo > Guardar todo** (<Ctrl+Shift+S>) y realice los siguientes pasos.



1.5.2 Sentencias básicas en el desarrollo tipo consola

Para implementar una aplicación de tipo consola se debe usar sentencias que permitan organizar de manera adecuada el código. A continuación, las principales sentencias:

Declarar una variable

En toda aplicación se debe declarar las variables a usar, el formato de declaración tiene el siguiente formato:

- `Declaración de una sola variable: Dim variable As tipo`
- `Declaración de dos variables: Dim variable1 As tipo, variable2 As tipo`

Mostrar un mensaje por pantalla

Un mensaje representa la única forma de comunicar al usuario con la aplicación; se usa para mostrar títulos o solicitar algún dato al usuario. Su formato es:

```
Console.Write("Mensaje")
Console.Writeline("Mensaje")
```

Imprimir un valor desde una variable

Cuando se realiza un cálculo será necesario exponerlo al usuario. Se presentan dos formatos para la impresión:

- `Imprimir solo valor: Console.Write(variable)`
- `Imprimir mensaje y valor: Console.Write("Mensaje:" + variable)`

Imprimir valores numéricos

Todo valor numérico podrá ser impreso solo si se convierte a texto. Usando el siguiente formato:

```
valorNumerico.ToString()
```

De aquí se desprende que, si quiere imprimir con formato de moneda, se usa el siguiente formato:

```
valorNumerico.ToString("C")
```

o, si quiere imprimir a dos decimales, use el siguiente formato:

```
valorNumerico.ToString("0.00")
```

Pausar la muestra de resultados

Al implementar una aplicación de tipo consola, se debe colocar la sentencia `Console.ReadKey()` para poder pausar la impresión y poder así visualizar los resultados de la aplicación.

Caso desarrollado 1 Cálculo de sueldos

Aplicación que permita determinar el sueldo básico, descuento y neto que debe percibir un empleado de acuerdo a las horas de trabajo y a un costo por hora.

Tener en cuenta los siguientes aspectos:

- Debe ingresar el nombre del empleado, horas trabajadas y costo por hora.
- El sueldo básico se obtiene del producto entre las horas trabajadas y el costo por hora.
- El descuento se obtiene aplicando el 12% al sueldo básico.
- El neto se obtiene de la diferencia entre el sueldo básico y el descuento.
- Mostrar los montos con el formato de moneda.

Solución:

```
Module Module1
    Sub Main()
        'DECLARACION DE VARIABLES
        Dim empleado As String
        Dim hTrabajadas As Integer, costoHora As Double
```

```
        'INGRESO DE VALORES
        Console.WriteLine("** PAGO DE EMPLEADOS **")
        Console.Write("Nombre del empleado: ")
        empleado = Console.ReadLine()
        Console.Write("Horas trabajadas: ")
        hTrabajadas = Console.ReadLine()
        Console.Write("Costo por hora: ")
        costoHora = Console.ReadLine()

        'REALIZANDO CALCULOS
        Dim basico As Double = hTrabajadas * costoHora
        Dim descuento As Double = basico * 0.12
        Dim neto As Double = basico - descuento

        'IMPRIMIR RESULTADOS
        Console.WriteLine("------------------------------")
        Console.WriteLine("EMPLEADO: " + empleado)
        Console.WriteLine("HORAS TRABAJADAS: " + hTrabajadas.ToString())
        Console.WriteLine("COSTO HORA: " + costoHora.ToString("C"))
        Console.WriteLine("------------------------------")
        Console.WriteLine("BASICO: " + basico.ToString("C"))
        Console.WriteLine("DESCUENTO: " + descuento.ToString("C"))
        Console.WriteLine("NETO: " + neto.ToString("C"))

        Console.ReadKey()
    End Sub
End Module
```

Especificaciones en el código:

- Para asignar el código de la aplicación debe especificarse dentro del bloque.

```
Sub Main(){
   'Código de implementación de la aplicación
}
```

- Se declara la variable a usar en la aplicación.

```
Dim empleado As String
Dim hTrabajadas As Integer, costoHora As Double
```

- Mostrar el mensaje en pantalla Pago de empleados con la sentencia Console. WriteLine() y para obtener el nombre del empleado desde el teclado debe usarse la sentencia Console.ReadLine().

```
Console.WriteLine("** PAGO DE EMPLEADOS **")
Console.Write("Nombre del empleado: ")
empleado = Console.ReadLine()
```

Donde empleado es la variable que recepcionará el valor del nombre del empleado.

- Ahora, se declara las variables básico, descuento y neto para asignar directamente un valor basado en las formulas propuestas solicitadas por el caso.

```
Dim basico As Double = hTrabajadas * costoHora
Dim descuento As Double = basico * 0.12
Dim neto As Double = basico - descuento
```

- Código para mostrar información obtenidos a partir de las variables.

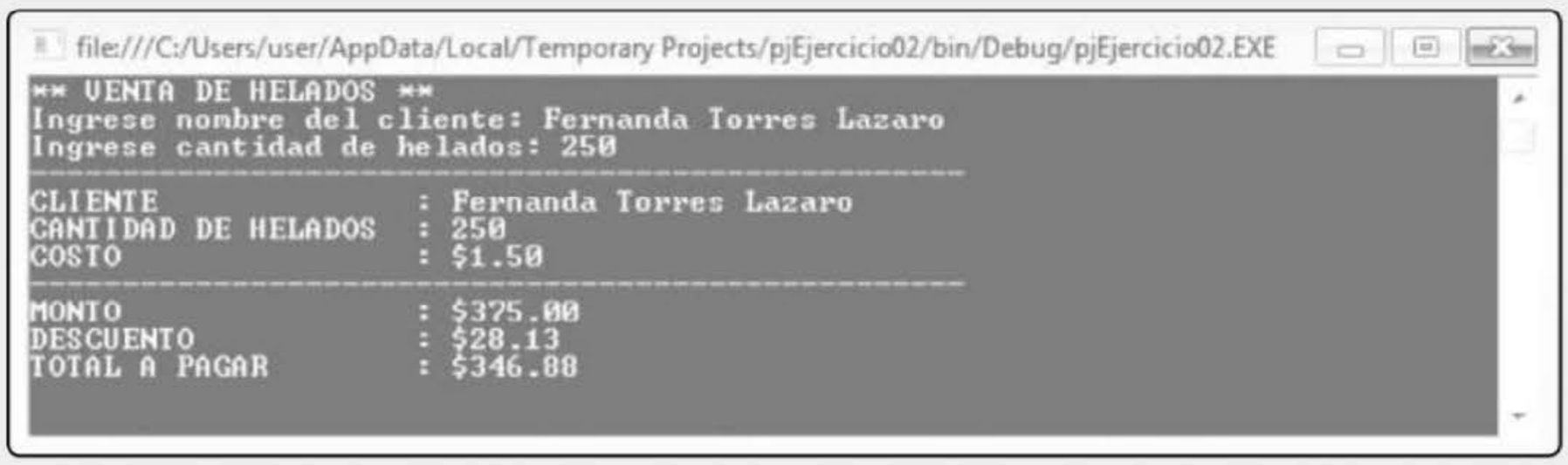
```
Console.WriteLine("EMPLEADO: " + empleado)
```

- Finalmente, se asigna la función ReadKey() para mostrar los resultados esperados.

```
Console.ReadKey()
```

Caso desarrollado 2 Venta de helados

Aplicación que permita determinar el monto, descuento y total a pagar que debe realizar un cliente por la compra de helados de un mismo tipo.



Tener en cuenta los siguientes aspectos:

- Debe ingresar el nombre del cliente y la cantidad de helados a comprar.
- El precio del producto es un valor constante que asciende a $1.50.
- El monto se calcula del producto entre el precio y la cantidad comprada de helados.
- El descuento se obtiene aplicando el 7.5% del monto.
- El total a pagar se obtiene de la diferencia entre el monto y el descuento.
- Mostrar los montos con el formato de moneda.

Solución:

```
Module Module1

    Sub Main()
        'DEFINICION DE CONSTANTES
        Const precio As Double = 1.5

        'DECLARACION DE VARIABLES
        Dim cliente As String
        Dim cantidad As Integer

        'INGRESO DE VALORES
        Console.WriteLine("** VENTA DE HELADOS **")
        Console.Write("Ingrese nombre del cliente: ")
        cliente = Console.ReadLine()
        Console.Write("Ingrese cantidad de helados: ")
        cantidad = Console.ReadLine()

        'REALIZANDO CALCULOS
        Dim monto As Double = cantidad * precio
        Dim descuento As Double = monto * (7.5 / 100)
        Dim total As Double = monto - descuento
```

```
        'IMPRIMIR RESULTADOS
        Console.WriteLine("-----------------------------------------")
        Console.WriteLine("CLIENTE                : " + cliente)
        Console.WriteLine("CANTIDAD DE HELADOS  : " + cantidad.ToString())
        Console.WriteLine("COSTO                : " + precio.ToString("C"))
        Console.WriteLine("-----------------------------------------")
        Console.WriteLine("MONTO                  : " + monto.ToString("C"))
        Console.WriteLine("DESCUENTO          : " + descuento.ToString("C"))
        Console.WriteLine("TOTAL A PAGAR        : " + total.ToString("C"))

        Console.ReadKey()

    End Sub

End Module
```

Especificaciones en el código:

- La definición de la constante precio se realiza usando la palabra reservada Const, asimismo definiendo el tipo de datos y, finalmente, asignando el valor de la constante.

```
Const precio As Double = 1.5
```

- Para aplicar un porcentaje se usa la división porcentual sobre el valor solicitado y se le asigna a la variable descuento.

```
Dim descuento As Double = monto * (7.5 / 100)
```

- Para mostrar un valor numérico en tipo moneda se usa el método ToString con el parámetro C, que permite mostrar el símbolo de moneda

```
Console.WriteLine("TOTAL A PAGAR        : " + total.ToString("C"))
```

Caso desarrollado 3 Promedio de notas

Aplicación que permita determinar el promedio de un alumno dependiendo del ingreso de cuatro notas, para lo cual cada evaluación tiene un peso correspondiente.

Tener en cuenta los siguientes aspectos:

- Debe ingresar el nombre del alumno y sus cuatro notas.
- Los pesos de las notas son 10%, 20%, 30% y 40% según el orden de ingreso.
- El promedio de notas se obtiene del valor porcentual de cada nota según el peso.
- Mostrar los resultados en un formato numérico de dos decimales.

Solución:

```
Module Module1

    Sub Main()
        'DECLARACION DE VARIABLES
        Dim alumno As String
        Dim n1 As Integer, n2 As Integer, n3 As Integer, n4 As Integer

        'INGRESO DE VALORES
        Console.WriteLine("** PROMEDIO DE NOTAS **")
        Console.Write("Ingrese nombre del alumno: ")
        alumno = Console.ReadLine()
        Console.Write("Ingrese nota 1: ")
        n1 = Console.ReadLine()
        Console.Write("Ingrese nota 2: ")
        n2 = Console.ReadLine()
        Console.Write("Ingrese nota 3: ")
        n3 = Console.ReadLine()
        Console.Write("Ingrese nota 4: ")
        n4 = Console.ReadLine()

        'REALIZANDO CALCULOS
        Dim promedio As Double = (n1 * 0.1 + n2 * 0.2 + n3 * 0.3 + n4 * 0.4)
```

```
        'IMPRIMIR RESULTADOS
        Console.WriteLine("-----------------------------------------")
        Console.WriteLine("ALUMNO: " + alumno)
        Console.WriteLine("NOTA 1|NOTA 2|NOTA 3|NOTA 4")
        Console.WriteLine(n1.ToString("0.00") + vbTab + n2.ToString("0.00") + vbTab
                          + n3.ToString("0.00") + vbTab + n4.ToString("0.00"))
        Console.WriteLine("-----------------------------------------")
        Console.WriteLine("PROMEDIO: " + promedio.ToString("0.00"))

        Console.ReadKey()
    End Sub

End Module
```

Especificaciones en el código:

- Para el cálculo de las notas se usa un valor porcentual, el cual es implementado directamente aplicando la regla porcentual; por ejemplo, 10% es análogo a 0.1. De la misma manera se aplica a todos los demás porcentajes.

```
Dim promedio As Double = (n1 * 0.1 + n2 * 0.2 + n3 * 0.3 + n4 * 0.4)
```

- Para que las notas se muestren en la impresión de forma tabulada, se usa la sentencia VbTab.

```
Console.WriteLine(n1.ToString("0.00") + vbTab + n2.ToString("0.00") +
                  vbTab + n3.ToString("0.00") + vbTab + n4.ToString("0.00"))
```

Caso desarrollado 4 Hospedaje

Aplicación que permita determinar el monto real por hospedaje de un determinado cliente a un solo tipo de habitación.

```
file:///C:/Users/user/AppData/Local/Temporary Projects/pjEjercicio04/bin/Debug/pjEjercicio04.EXE
** HOSPEDAJE - CALCULO DE PAGO **
Ingrese nombre del cliente: Luz Lazaro Menor
Ingrese la cantidad de dias: 12
-----------------------------------------------------
CLIENTE           : Luz Lazaro Menor
DIAS DE HOSPEDAJE: 12
COSTO             : $35.50
-----------------------------------------------------
MONTO             : $426.00
```

Tener en cuenta los siguientes aspectos:

- Debe ingresar el nombre del cliente y la cantidad de días a hospedarse.
- El costo por día es un valor constante que asciende a $35.50.
- El monto resulta del producto entre los días de hospedaje y el costo por día.
- Mostrar los montos con el formato de moneda.

Solución:

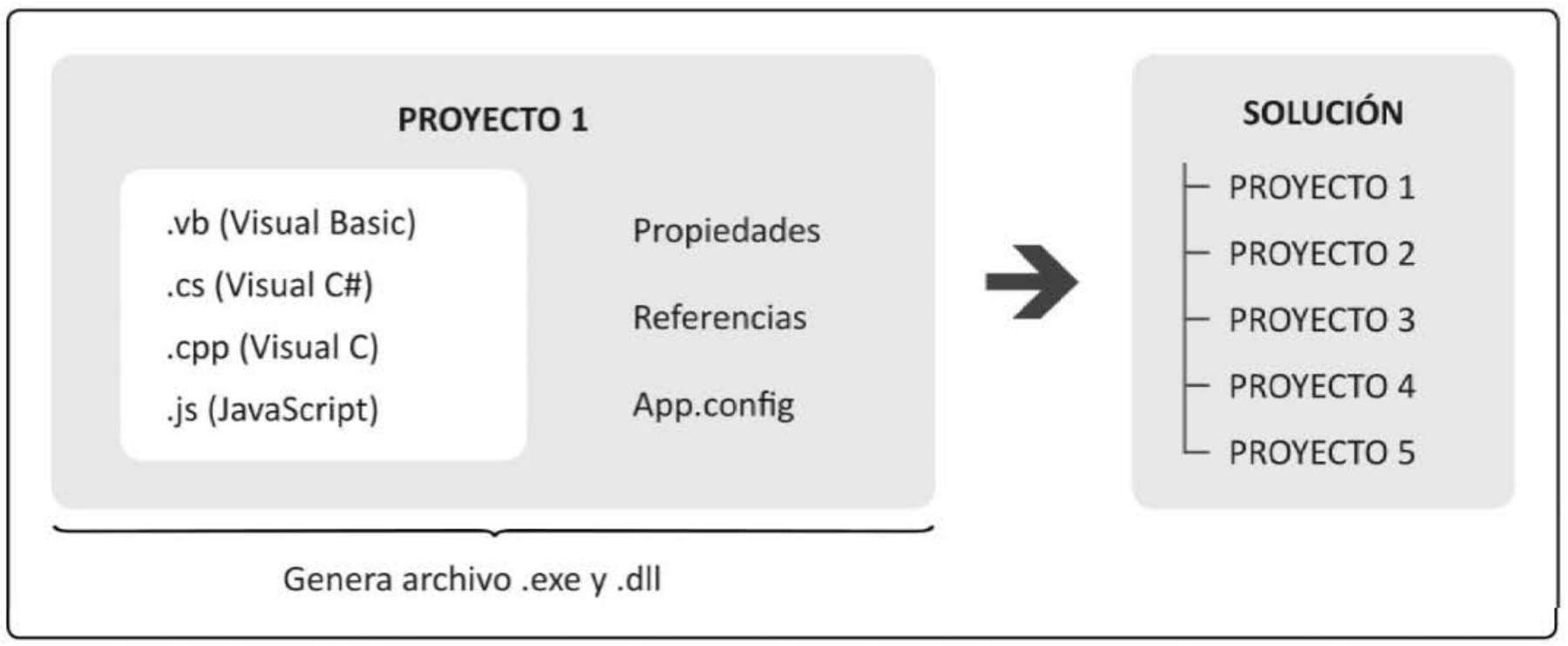
```
Module Module1

    Sub Main()
        'DECLARACION DE CONSTANTES
        Const costo As Double = 35.5

        'DECLARACION DE VARIABLES
        Dim cliente As String
        Dim dias As Integer

        'INGRESO DE VALORES
        Console.WriteLine("** HOSPEDAJE - CALCULO DE PAGO **")
        Console.Write("Ingrese nombre del cliente: ")
        cliente = Console.ReadLine()
        Console.Write("Ingrese la cantidad de días: ")
        dias = Console.ReadLine()

        'REALIZANDO CALCULOS
        Dim monto As Double = dias * costo

        'IMPRIMIR RESULTADOS
        Console.WriteLine("----------------------------------------")
        Console.WriteLine("CLIENTE          : " + cliente)
        Console.WriteLine("DIAS DE HOSPEDAJE: " + dias.ToString())
        Console.WriteLine("COSTO            : " + costo.ToString("C"))
        Console.WriteLine("----------------------------------------")
        Console.WriteLine("MONTO            : " + monto.ToString("C"))

        Console.ReadKey()
    End Sub

End Module
```

1.6 Retroalimentación

En este punto se hará un resumen rápido de todo lo visto en el capítulo:

- Visual Studio 2015 se caracteriza por proponer comodidad al programador, integrando muchos lenguajes en un solo entorno.
- Visual Studio 2015 es considerado como un IDE, el cual proporciona un servicio integral al programador, considerado como un editor intuitivo porque hasta un programador novato se sienta cómodo desarrollando.
- El framework .NET y el Visual Studio Code, más conocido como VS Code, son considerados OpenSource, abriendo así un camino importante para los desarrolladores de la comunidad Microsoft que quieran realizar mejoras en el lenguaje.
- La distribución de Visual Studio es de forma gratuita para entornos Windows, Mac y Linux; asimismo, la instalación se ha dividido en componentes para que al realizarla se pueda seleccionar los elementos necesarios, esto le permite a Visual Studio realizar una instalación ligera.
- Una de las novedades más resaltantes de esta versión es la cuenta de usuario Visual Studio; eso quiere decir que para empezar con el desarrollo de aplicación se debe tener una cuenta de usuario, la cual le permitirá acceder a recursos en línea y otras novedades. Asimismo, es posible tener varias cuentas de Visual Studio.
- Respecto al trabajo de aplicaciones móviles, Visual Studio presenta un trabajo integrado entre HTML/JavaScript y Apache Córdova, esto debido a la colaboración entre Microsoft y la comunidad Apache Córdova de código abierto.
- La plataforma universal de Windows permite implementar aplicaciones bajo la plataforma de Windows 10, que podrán ser ejecutadas en cualquier tipo de dispositivo, desde móviles hasta equipos de escritorio.
- Incorpora el compilador Roslyn, el cual se encarga de realizar la comprobación de sintaxis en tiempo real; asimismo, autocompleta el código en la ventana de código fuente y presenta un código de sugerencia ante un error mediante bombillas ubicadas en el lado izquierdo de la ventana de código.
- Entre los principales componentes del .NET Framework se encuentra el Common Language Runtime (CLR), el cual es considerado como el motor de ejecución que controla las aplicaciones en etapa de ejecución, y la biblioteca de clases de .NET Framework, la cual proporciona una biblioteca de código probado y reutilizable que puede ser usada por los programadores en el desarrollo de sus propias aplicaciones.

- En el año 1996 Microsoft contaba con 4 productos independientes; esto creaba desarrolladores de diferentes líneas a pesar de usar un producto del mismo proveedor; estos son Visual Basic, Visual C++, Visual FoxPro y Visual SourceSafe. El producto que permitía una integración entre todos era el Visual SourceSafe pero no resultaba productivo que un programador nativo de una línea se integre a los demás. Es así que en 1997 decide integrar los productos mencionados anteriormente en uno solo, llamándolo Visual Studio 97, y lo muestra al mundo en febrero del mismo año.
- Visual Studio 2015 ofrece un completo entorno de desarrollo integrado para crear aplicaciones en entornos como Windows, Android e iOS, además de aplicaciones web y servicios de nube innovadores; por tanto, su distribución de Visual es gratuito mediante la comunidad. Use el siguiente Link para la descarga **<https://www.visualstudio.com/es-es/downloads/download-visual-studio-vs.aspx>**.
- El entorno de Visual Studio 2015 ofrece la posibilidad de modificarlos en diferentes idiomas, para esto debemos descargar el paquete de idioma de Microsoft Visual Studio 2015. Use la siguiente URL para la descarga del paquete de idioma: **<https://www.microsoft.com/es-ES/download/details.aspx?id=48157>**.
- En Visual Studio se pueden crear aplicaciones, sitios web, aplicaciones web, script, aplicaciones móviles, etc., y en todos ellos siempre se debe empezar por un proyecto. Inicialmente, un proyecto tiene los siguientes componentes: archivo con código fuente, iconos e imágenes usados en una aplicación, y archivo de datos que contiene información relevante para la aplicación así como la configuración del compilador.
- El siguiente diagrama muestra la relación entre proyectos y soluciones, y los elementos que estos contienen de forma lógica.



1.7 Ejercicios propuestos

De acuerdo con los ejercicios desarrollados, ponga en práctica sus habilidades:

Ejercicio propuesto 1 Conversión de grados

Aplicación que permita determinar la temperatura en grados Kelvin (K) y Rankine (R) a partir de un valor en grados centígrados.

Tener en cuenta los siguientes aspectos:

- Debe ingresar el valor en grados centígrados.
- Use la siguientes fórmulas:

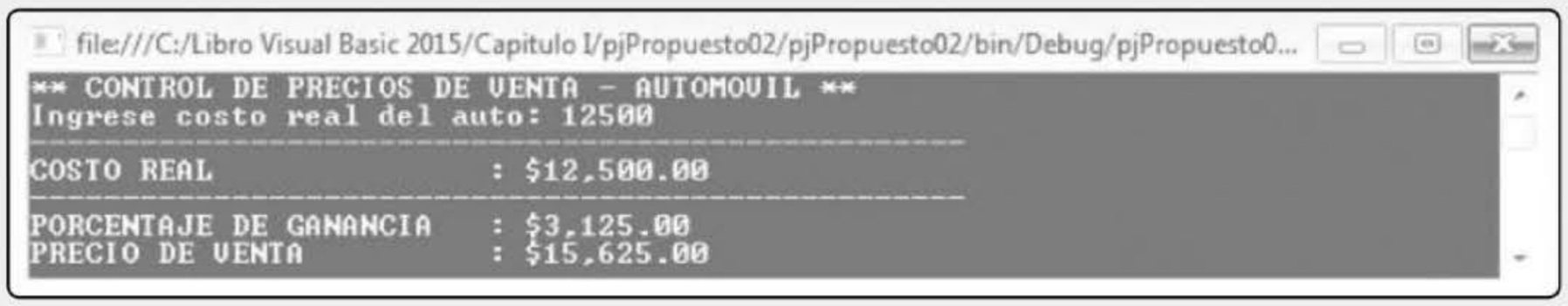
$$rankine = \frac{9 \times Centigrados}{5} + 491.67$$

$$kelvin = Centigrados + 273.15$$

- Mostrar los valores resultantes a dos decimales usando el método ToString("0.00").

Ejercicio propuesto 2 Venta de autos

Una empresa de venta de automóviles nuevos necesita una aplicación que permita determinar el precio de costo de un determinado auto, sabiendo que la empresa tiene como política ganar el 25 % sobre el valor de auto.



Tener en cuenta los siguientes aspectos:

- Debe ingresar el costo real del auto.
- Determinar el precio de venta para obtener una ganancia del 25% sobre el costo real del auto.
- Mostrar los valores resultantes en formato de moneda.

Capítulo 2

Controles de formularios

Capacidad terminal:

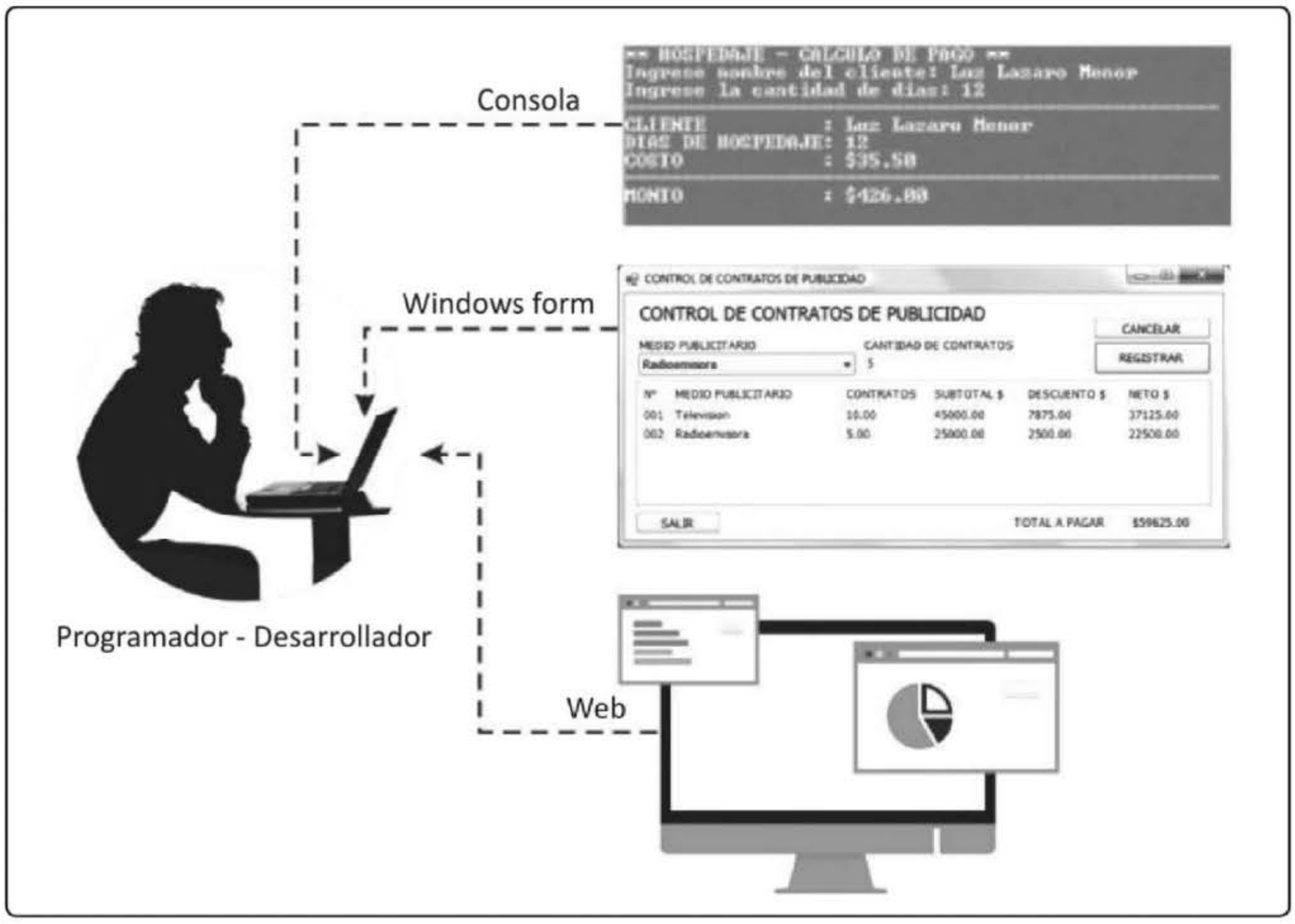
Conocer la forma adecuada de realizar la instalación de Visual Studio 2015, así como reconocer todos los elementos que componen su entorno.

Contenido

- Trabajando con Windows Forms
- Controles visuales
- Notación para los nombres de los controles
- Control Form
- Controles de visualización de información
- Controles de ingreso de información
- Controles de acción
- Controles de opción simple y múltiple
- Controles de muestra de información
- Controles contenedores de otros controles
- Controles de manejo de imágenes
- Controles de manejo de tiempos
- Operaciones básicas con los formularios
- Eventos de los controles
- Editor de código
- Desarrollo de aplicaciones tipo Windows Forms

2.1 Trabajando con Windows Forms

Cuando se desarrollan aplicaciones se supone usar formularios o páginas web como plataforma de trabajo y Visual Studio está enfocado en todo tipo de aplicación que los usuarios finales necesiten. Aquellos que se inician en la programación deben concentrarse en tres tipos de aplicaciones, que se muestran en el siguiente gráfico:



Como se puede observar, se cuenta con aplicaciones de tipo consola, windows forms y web, que son necesarios para el inicio en la programación. Las características de estos tipos son:

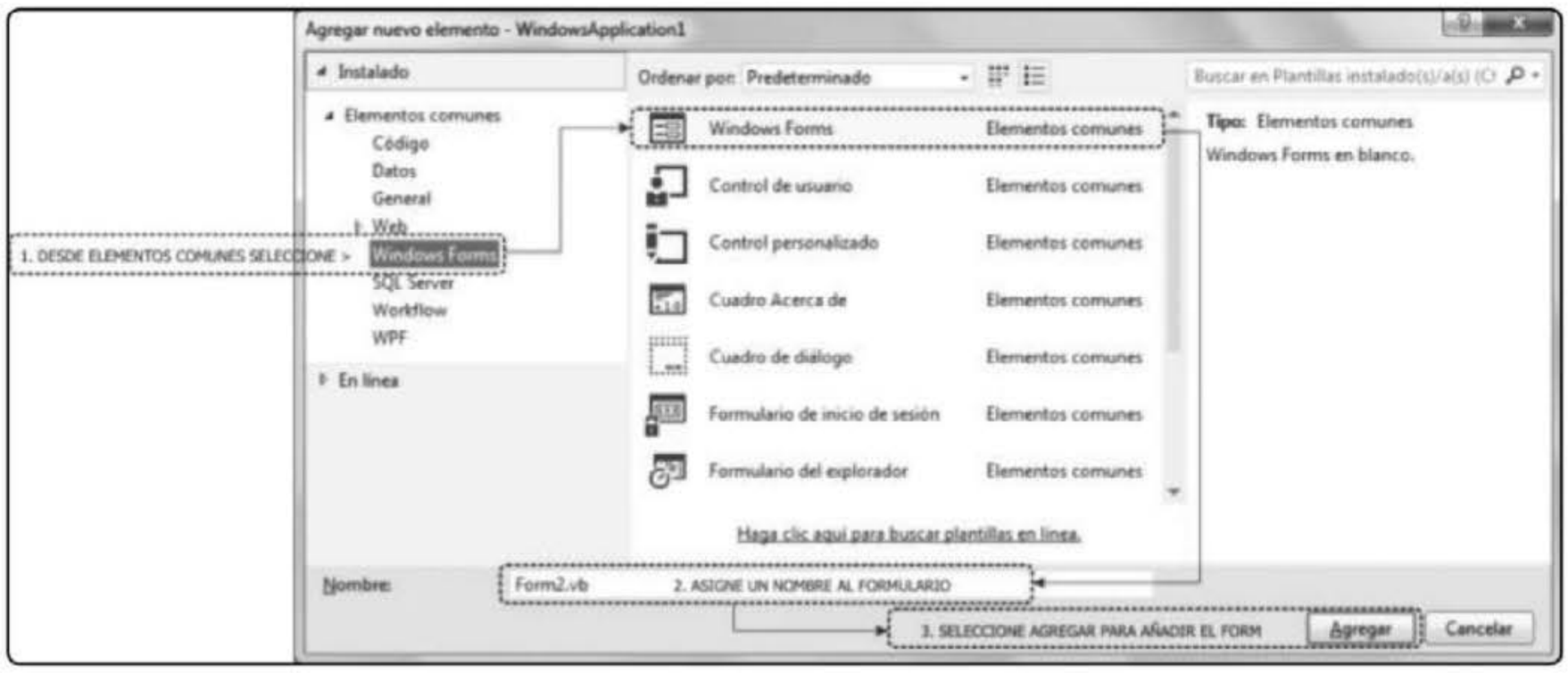
Tipo de aplicación	Descripción
Consola	Es una aplicación que se ejecuta en una ventana tipo **DOS** en línea de comandos; se concentra en la programación más que en la parte visual de la aplicación.
Windows Forms	Es una aplicación considerada como cliente porque se ejecuta en el equipo del usuario; esta normalmente muestra información y permiten que el usuario acceda a un servidor de datos.
Web	Es una aplicación que tiene como interfaz de usuario a un explorador como Internet Explorer, Google Chrome u otros. Una de las principales ventajas de este tipo de aplicaciones es que no tiene costo de distribución, ya que los usuarios solo deben contar con un navegador web.

2.1.1 Agregar un formulario al proyecto

Los formularios son objetos administrables, por lo tanto, se pueden agregar o eliminar tantas veces se necesite.

Para agregar un formulario al proyecto:

- Haga clic derecho sobre el **proyecto > Agregar > Windows Forms** y realice los siguientes pasos:



Se puede realizar una segunda forma de agregar un formulario al proyecto, realizando los siguientes pasos:

- Desde el menú principal seleccione **Proyecto > Agregar Windows Forms**.
- Seleccione el elemento común **Windows Forms > Windows Forms**, asigne un nombre al formulario y presione el botón **Agregar**.

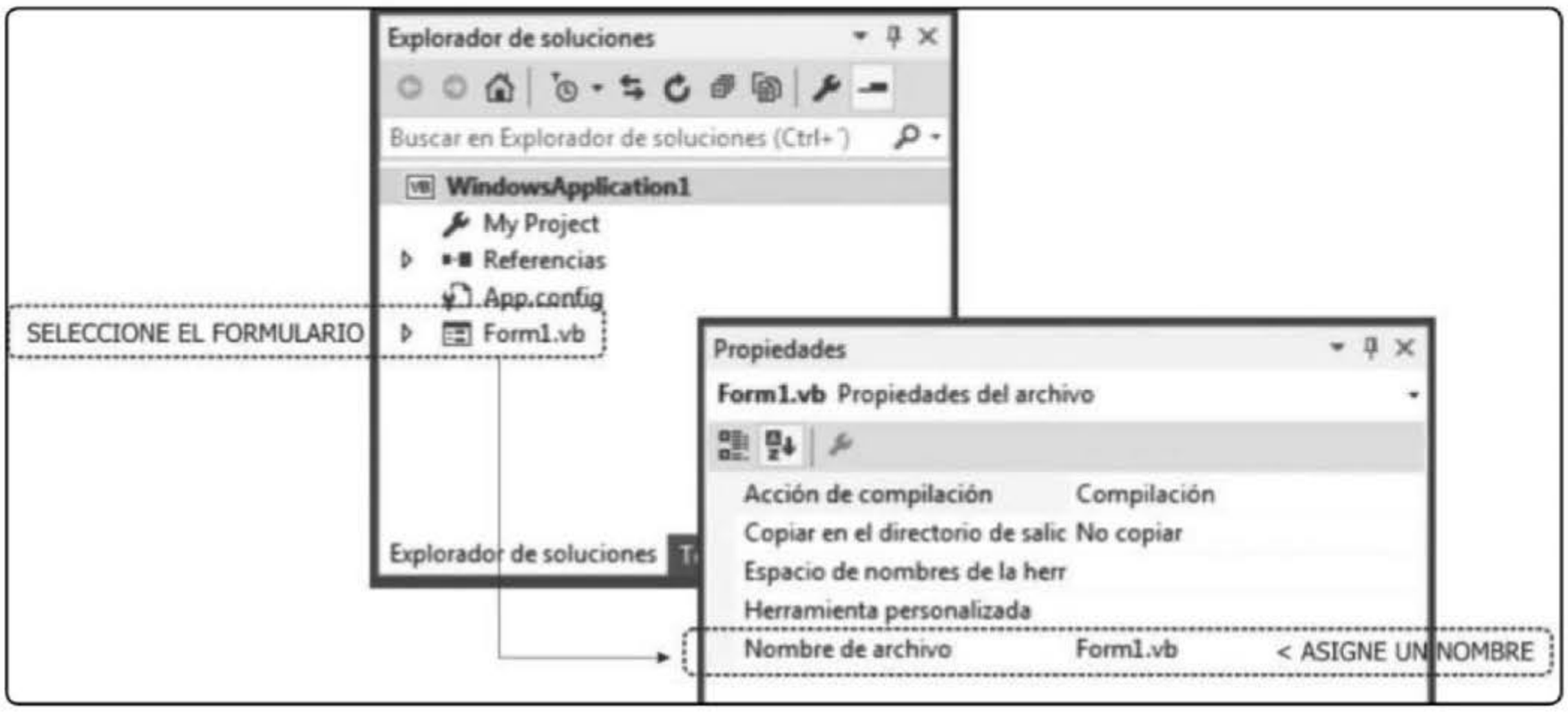
2.1.2 Asignar un nombre al formulario

Es necesario tener en cuenta que asignar un nombre al formulario corresponde a dos responsabilidades, ya que el formulario es un objeto y a la vez un archivo del proyecto. Si se asigna directamente el nombre al formulario como archivo entonces su nombre como objeto será el mismo. Pero si se realiza al revés, es decir, si primero se le asigna un nombre como objeto, entonces también deberá especificar el nombre del formulario como archivo.

Realice los siguientes pasos para asignar un nombre como archivo:

- Seleccione el formulario desde el Explorador de soluciones.

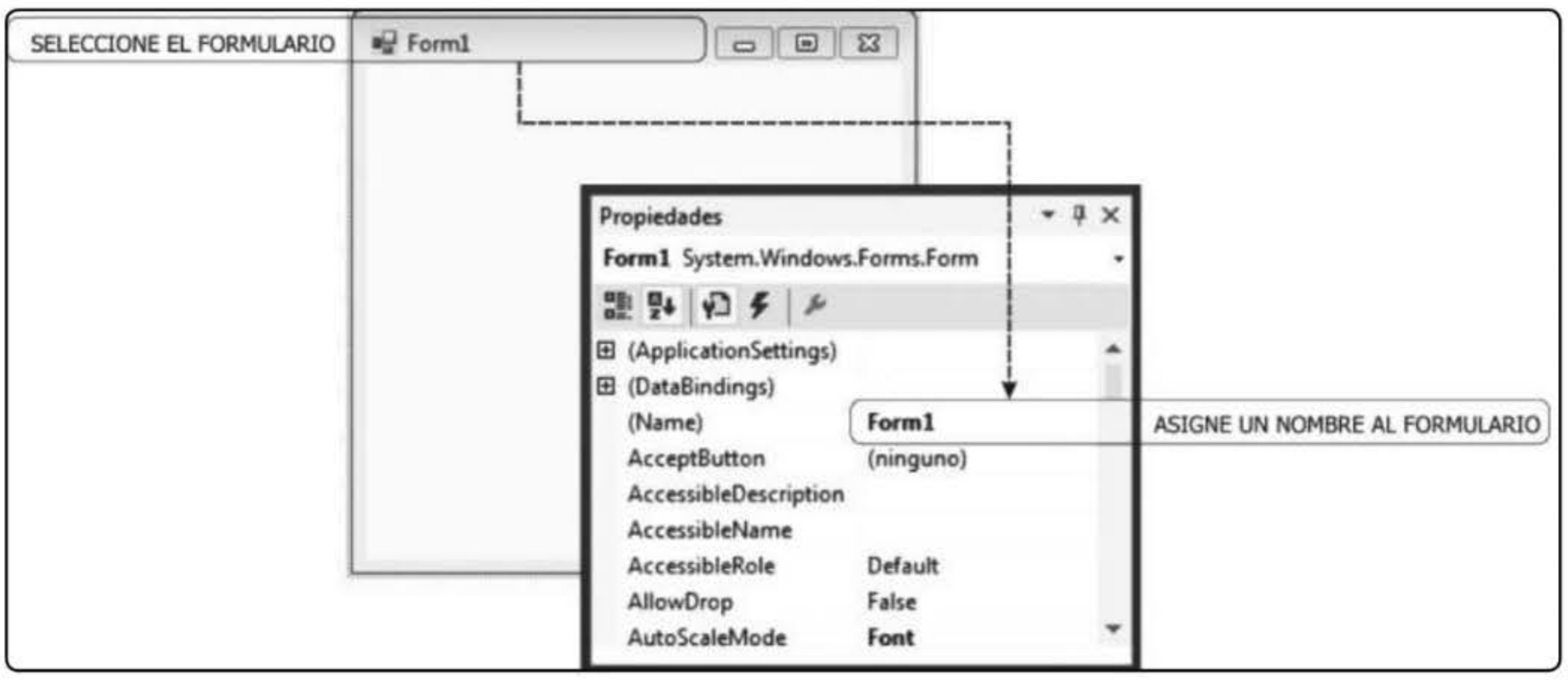
- Asigne un nombre desde el panel de propiedades, tal como se muestra en la siguiente imagen:



Tenga en cuenta que al asignar un nombre debe mantener la extensión **.VB** mostrada como valor de la propiedad **Nombre de archivo**; considere que no colocar la extensión conlleva a no reconocer dicho archivo como tipo Visual Basic. Se recomienda que los nombres de los formularios inicien con las letras **frm** para llevar un mejor control de los formularios en un mismo proyecto.

Realice los siguientes pasos para asignar un nombre al formulario como objeto:

- Haga clic en cualquier lugar de la barra de título del formulario.
- Ubíquese en la propiedad (Name) desde el panel de propiedades.
- Asigne un nombre, tal como se muestra en la siguiente imagen:



2.1.3 Eliminar un formulario del proyecto

Se debe tener en cuenta que cuando se elimina un formulario también se eliminan todos los controles agregados en él. Realice los siguientes pasos para eliminar un formulario:

- Seleccione el formulario desde el Explorador de soluciones.
- Haga clic derecho sobre el formulario y seleccione **Eliminar** o simplemente haga clic en el botón **Suprimir** desde el teclado.

2.1.4 Establecer un formulario de inicio dentro de un proyecto

Cuando se ejecuta una aplicación solo puede mostrarse un formulario a la vez. Si tenemos en cuenta que en un proyecto puede haber varios formularios, uno de ellos debe visualizar el usuario; normalmente este formulario comanda frente a los otros en un proyecto de sistema, ya que puede invocar a los demás formularios.

Ver el siguiente escenario: Se cuenta con un proyecto llamado WindowsApplication1, el cual cuenta con tres formularios: **frmCompra**, **frmListado** y **frmVenta** y se necesita que **frmCompra** sea el formulario de inicio.

Ahora, los pasos para asignar un formulario de inicio en un proyecto:

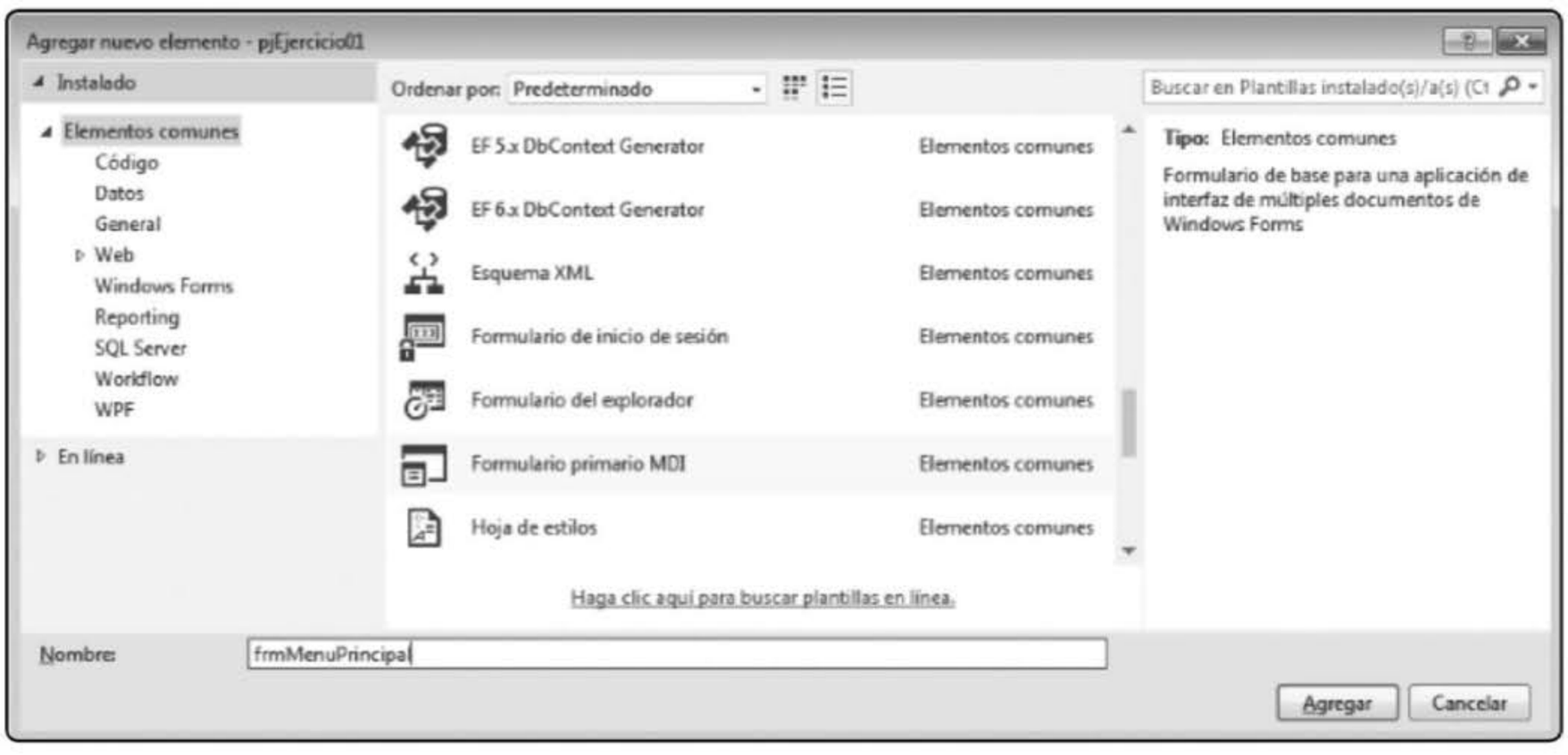
- Haga clic derecho sobre el proyecto **Windows Application** y seleccione **Propiedades**, desde el Explorador de soluciones.
- En la sección **Formulario de inicio** seleccione el formulario **frmCompra**.
- Finalmente, pruebe el cambio presionando F5 para ejecutar la aplicación y comprobar que el formulario mostrado es **frmCompra**.

2.1.5 Formulario primario MDI

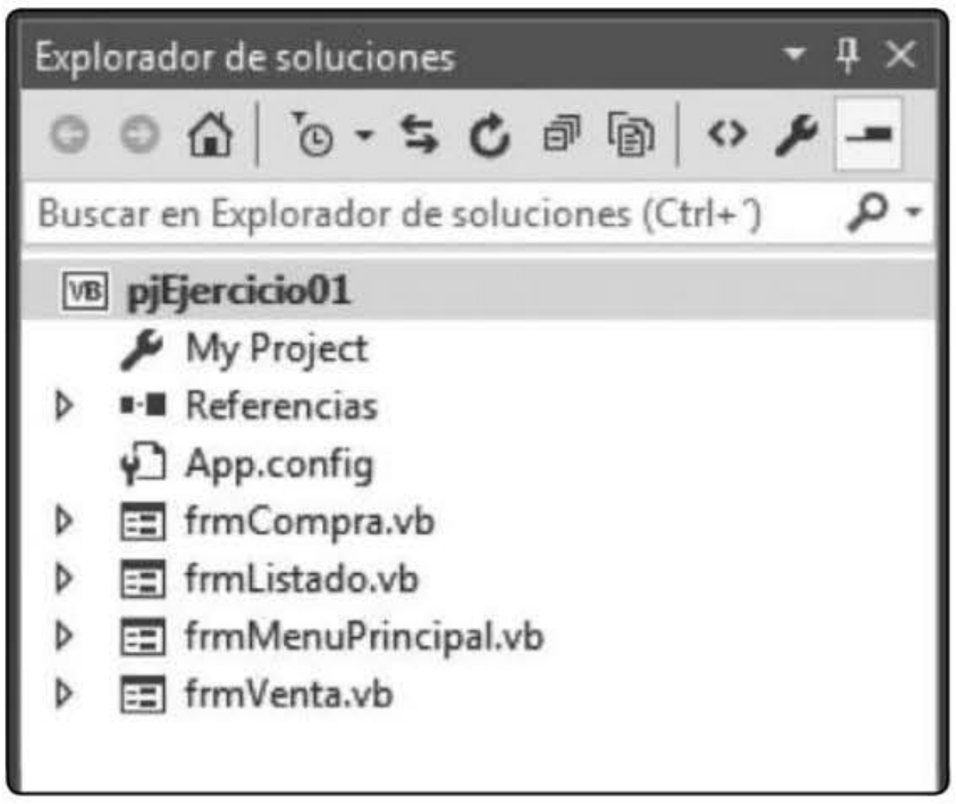
Es un tipo de formulario maestro, el cual permite tener el control de todos los formularios de un mismo proyecto; la diferencia con un formulario Windows Forms es que MDI presenta una barra de menú, barra de herramientas y una de estado que pueden ser modificadas por el desarrollador.

Realice los siguientes pasos para agregar un formulario MDI:

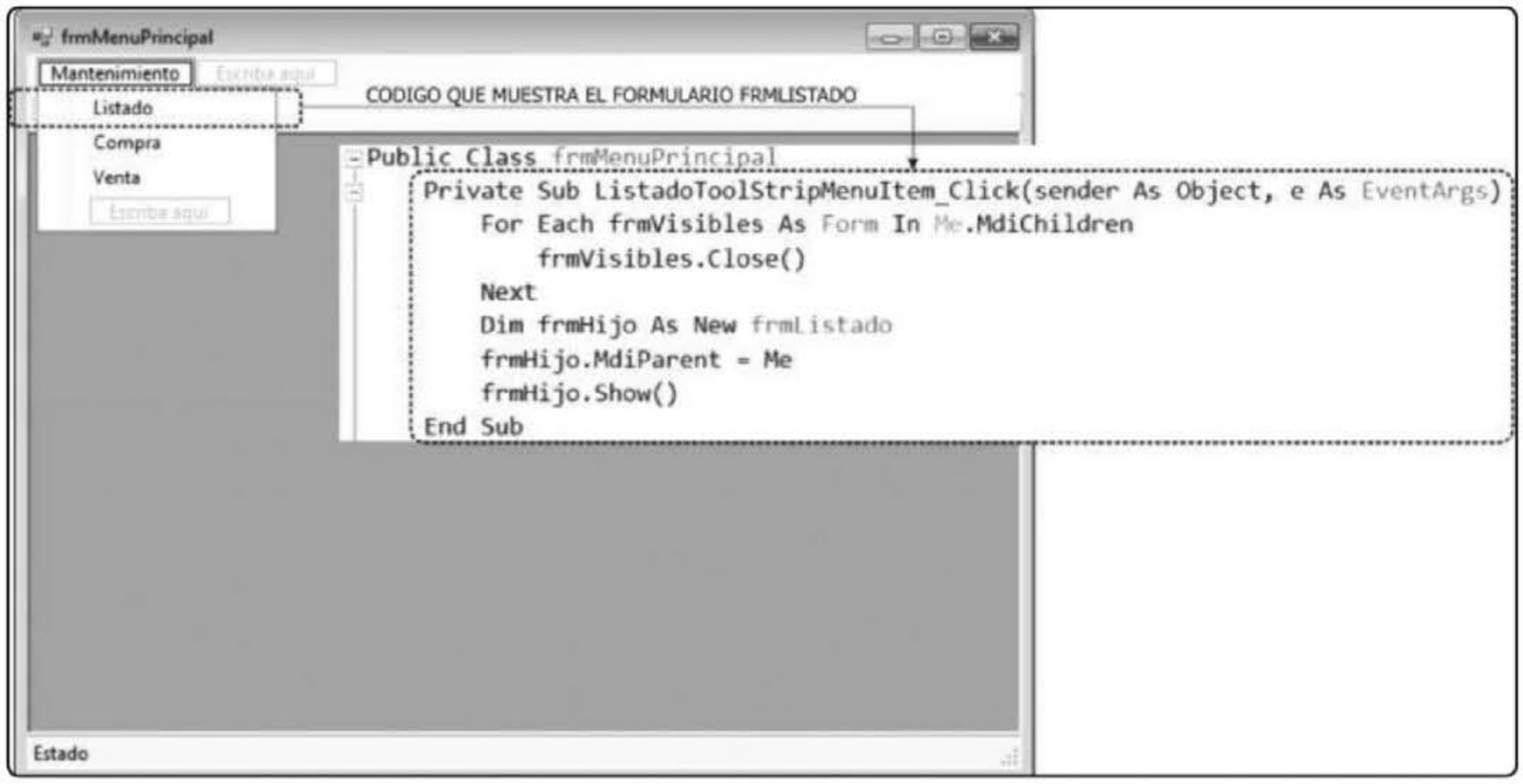
- Haga clic derecho en el **proyecto > Agregar > Windows Forms**.
- Seleccione desde los elementos comunes Windows Forms, luego **Formulario primario MDI** y asigne un nombre tal como se muestra en la siguiente imagen:



- Luego, el explorador de soluciones debe tener el siguiente aspecto:



Como verá, el formulario MDI se visualiza como un formulario básico; lo único que se debe realizar es establecer un nombre al formulario y a la vez asignarlo como formulario de inicio. Inicialmente, un formulario primario MDI tiene el aspecto de un formulario básico de Windows, el cual puede ser administrado de la mejor manera por el programador. Por ejemplo:



Una vez establecidos los elementos que compone un menú de opciones será necesario agregarle código para poder invocar a los formularios, los cuales se comportarán como formularios hijos (Windows Form) frente al formulario maestro (MDI).

Se analizará el código que permite invocar un Windows Form desde un formulario maestro MDI:

```
For Each frmVisibles As Form in Me.MdiChildren
  frmVisibles.Close()
Next
```

El código permite cerrar todos los formularios que se encuentren activos, esto se realiza con la intención de no mostrar formularios repetidos, pues cada vez que se invoque a una ventana primero se cerrarán todas las abiertas. La sentencia For Each permite recorrer por todos los formularios contenidos en el formulario maestro, estos son reconocidos por la cláusula MdiChildren y una vez encontrados se cerrarán con la cláusula Close().

```
Dim frmHijo As New frmListado
frmHijo.MdiParent = Me
frmHijo.Show()
```

Se crea un objeto de la clase **frmListado** perteneciente al formulario del mismo nombre, luego se adhiere dicho formulario al formulario maestro usando la cláusula MdiParent = Me. Finalmente, para mostrar el formulario usamos el método Show().

2.1.6 Diseñar una interfaz de usuario

La interfaz es la única forma que tiene el usuario de la aplicación de comunicarse con el sistema. Normalmente, una interfaz es un trabajo integral en la cual se usan criterios de diseño gráfico y ergonomía, los cuales tienen por objetivo crear aplicaciones interactivas, sencillas e intuitivas centradas íntegramente en los usuarios.

La Graphic User Interface (GUI) es la denominación en ingles de las interfaces de usuario; inicialmente fue presentada por Xerox en los años 70 y fue en los 80 que Apple las hizo conocidas mundialmente. Existen algunos principios que norman el diseño de las GUI, las cuales se nombran a continuación:

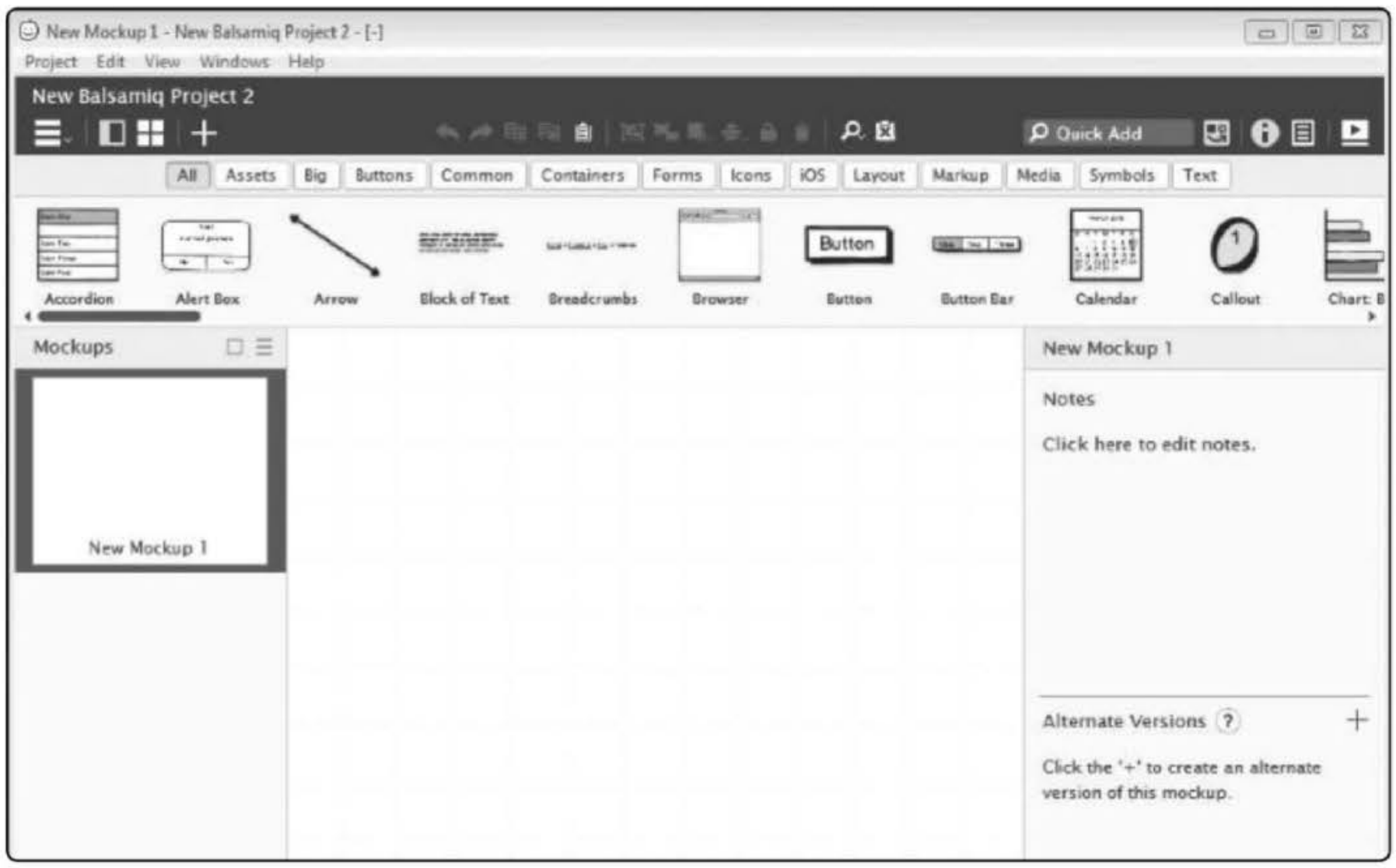
- **Enfocado al usuario**: Se debe diseñar una GUI que use términos y conceptos habituales de aquellas personas que harán uso de la aplicación.
- **Consistencia**: Los valores ingresados por los usuarios deben ser validados de tal forma que sean diferenciados los números de los textos y viceversa.
- **Comportamiento**: La GUI debe realizar actividades que permitan al usuario agilizarlas, generando un comportamiento generalizado para todos los usuarios.
- **Guía**: Al interactuar con la GUI se debe mostrar mensajes que permitan un desempeño adecuado frente al nuevo usuario.
- **Diversidad**: La GUI debe permitir la adaptabilidad de todos los usuarios de la aplicación, de tal forma que se implementen interfaces de acuerdo al perfil del usuario.

Para poder desarrollar un prototipo de GUI antes de implementarlo en Visual se puede usar la aplicación Mockup Balsamiq, el cual permitirá crear modelos que deben ser analizados por las personas que usarán la aplicación; una vez aprobada podrá ser diseñada en el entorno de Visual.

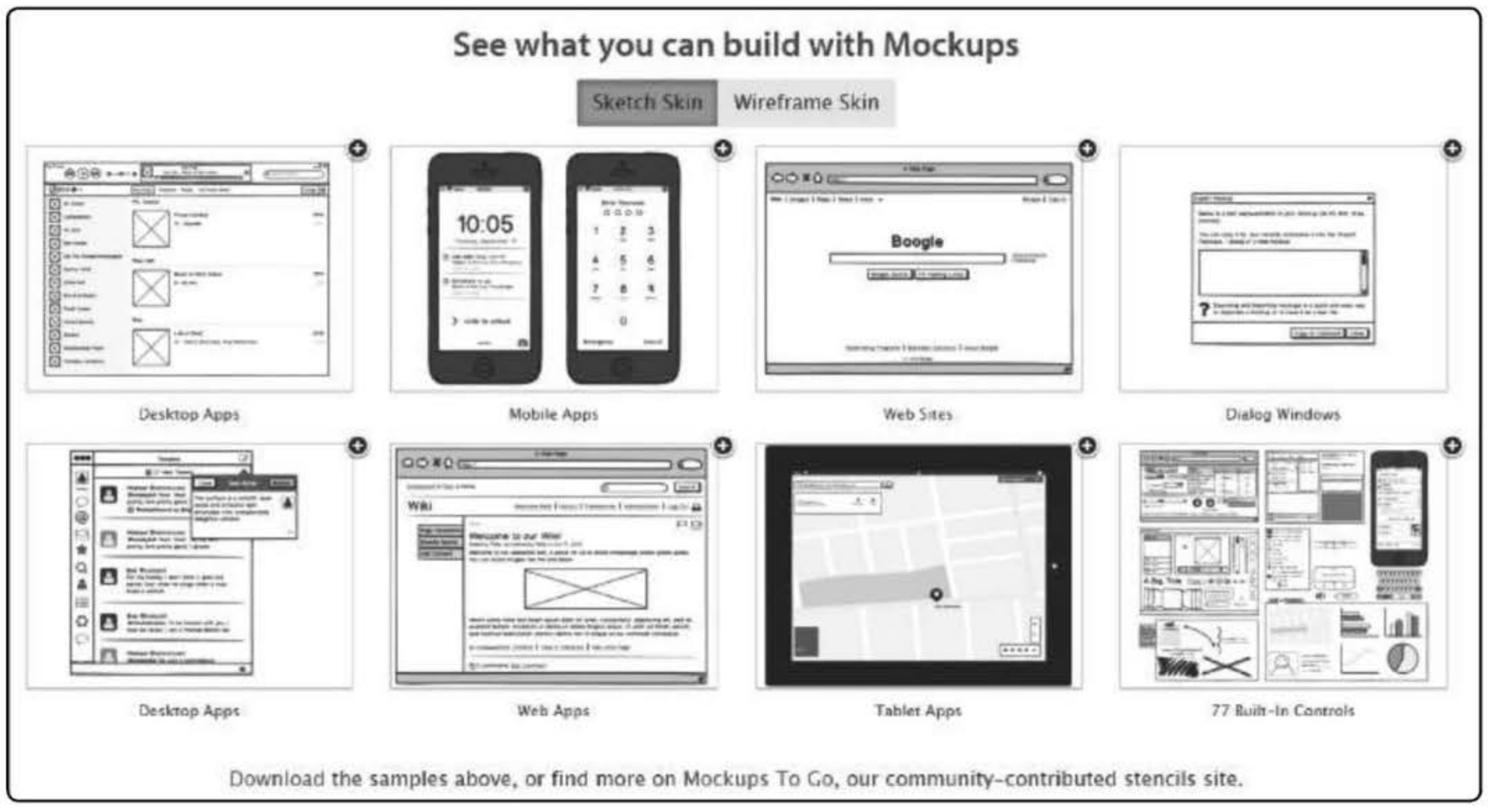
URL: <**https://balsamiq.com/products/mockups/**>

Pantalla inicial:

Seleccione el tipo de descarga. En este caso se puede seleccionar **Download Desktop App** para iniciar la descarga. El proceso de instalación es simple y sencillo; a continuación, se visualiza la pantalla principal:



La página del Mockup Balsamiq presenta algunas plantillas mostrando qué diseños se pueden desarrollar con esta aplicación.



El modelo presentado a continuación muestra la posible interfaz que puede tener una aplicación Windows Form, el cual controla el pago de los empleados bajo un determinado criterio.

Modelo: Mockup Balsamiq

Modelo: Visual Basic 2015

2.1.7 Ejecutar una aplicación

Se debe tener en cuenta dos tiempos:

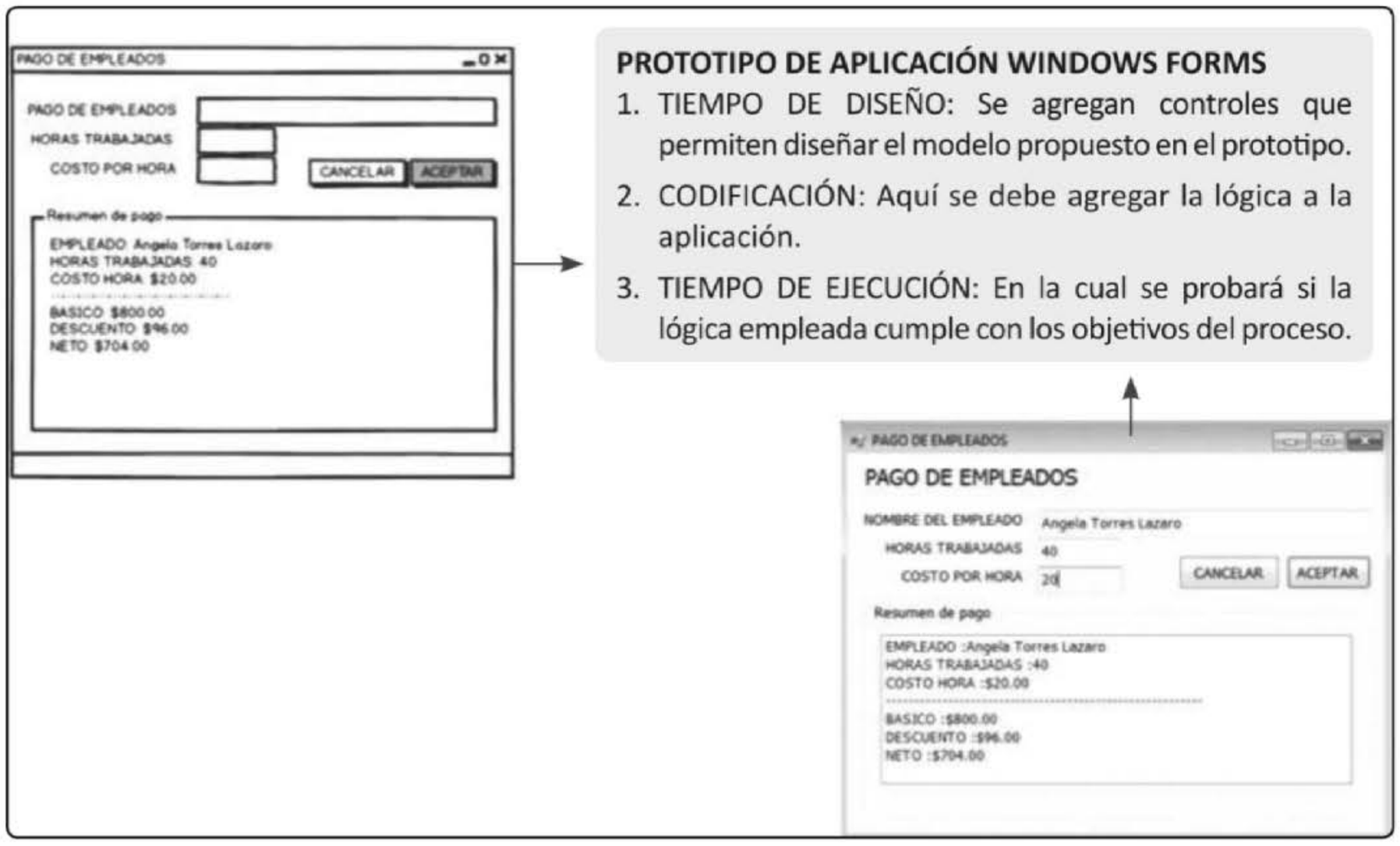
Especificación de tiempos	Descripción
Tiempo de diseño	Es el período en que el programador diseña un modelo que se denomina interfaz gráfica, que será expuesta al usuario final de la aplicación. Este tiempo se caracteriza por colocar los controles visuales en un contenedor.
Tiempo de ejecución	Es el período en el que una determinada aplicación es ejecutada por el sistema operativo. Es también llamado aplicación en ejecución, en la cual podría generarse diferentes tipos de errores llamados errores en tiempo de ejecución, que pueden ser administrados por mecanismos llamados control de excepciones.

Cuando se ejecuta una aplicación se debe considerar los siguientes aspectos:

- La aplicación no debe presentar errores en compilación.
- Se debe tener en cuenta qué proyecto se encuentra establecido como proyecto de inicio.
- Se debe tener en cuenta qué formulario se encuentra como formulario de inicio.

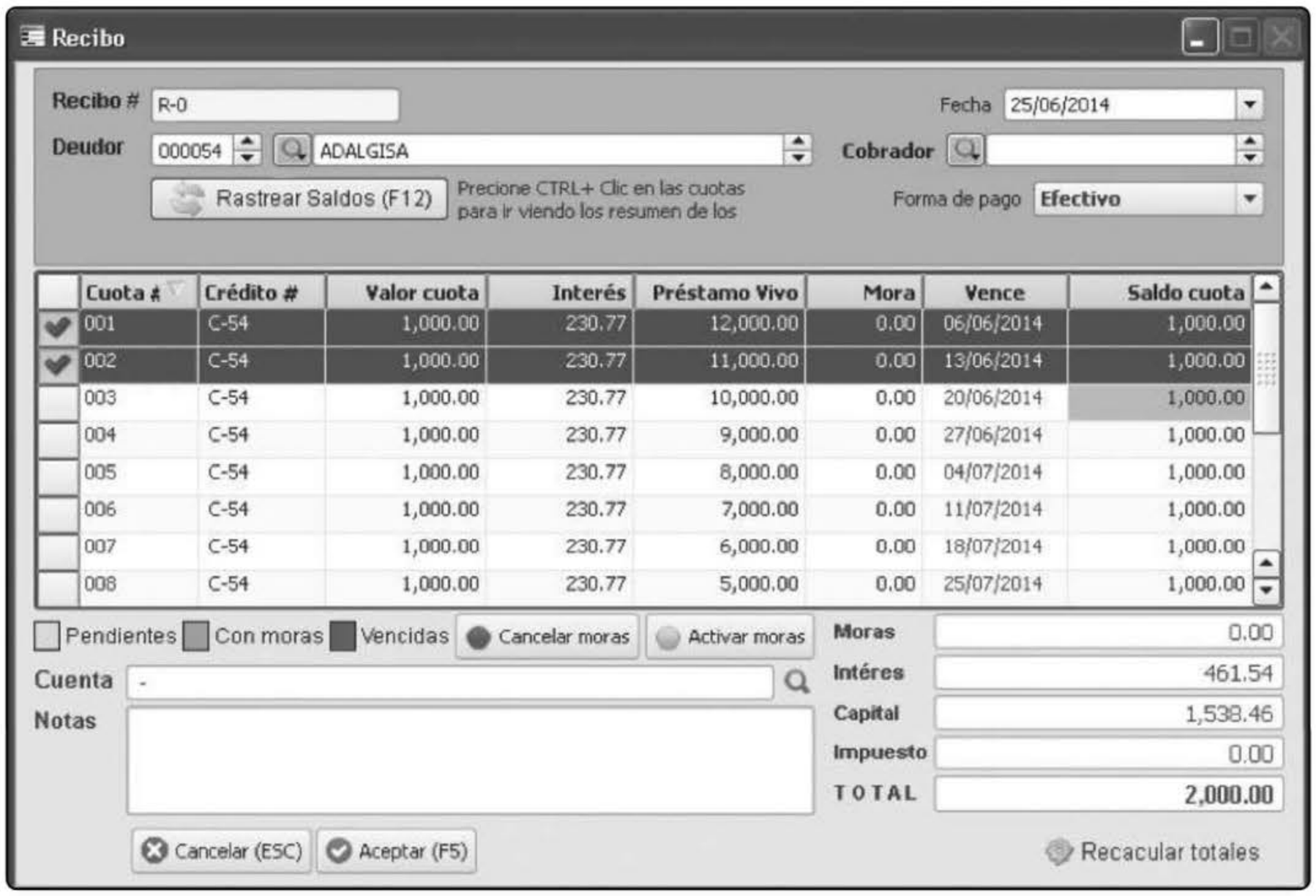
Ahora, las formas de ejecutar una aplicación:

- Haga clic en <F5> desde el teclado
- Haga clic en el botón ▶ Iniciar ▾



2.2 Controles visuales

Los controles permitirán crear entornos visuales sencillos e interactivos, pues estos serán los que interactúen con el usuario de la aplicación. Cada control tiene una misión en el desarrollo del entorno de la aplicación. A continuación, se observa un formulario que contiene los controles integrados en uno solo:



Como se puede observar, el trabajo final se enfoca en el desarrollo de un proceso que permita satisfacer una necesidad de negocio para la empresa. Es así que se debe aprender a integrar correctamente los controles visuales con el código Basic. No olvidar que el aspecto del formulario dependerá en un cien por ciento del buen criterio del programador.

2.3 Notación para los nombres de los controles

Todo elemento que contiene un proyecto tiene un nombre y este puede ser modificados a conveniencia del programador. Es así que existe una notación estándar que permite diferenciar el control dependiendo de la clase de donde proviene, creando así una buena práctica en la asignación de nombres a los controles contenidos en un Windows Form. A continuación, una tabla de los prefijos más conocidos para los principales controles; la notación es referencial:

Ficha: **Controles comunes**

Control	Prefijo	Descripción
Form	frm	Windows Forms, el considerado como un contenedor de controles; es aquí donde el programador diseñará un entorno enfocado en el usuario final.
Button	btn	Botón de comando es el encargado de ejecutar un determinado proceso en una aplicación; por ejemplo, calcular, eliminar, listar, etc.
CheckBox	chk	Botón de chequeo, permite asignar cajas de chequeo que permitirán al usuario seleccionar múltiples opciones en un mismo formulario.
ComboBox	cbo	Cuadro combinado, permite asignar varios valores en un solo control, los cuales podrán ser seleccionados por el usuario a partir de una selección desplegable.
DateTimePicker	dt	Control de fecha y hora, permite al usuario seleccionar en un solo elemento una fecha y una hora.
Label	lbl	Etiqueta, considerado como texto estático, el cual permite al programador mostrar información al usuario que no podrá ser modificada.
ListBox	lst	Cuadro de lista, permite asignar varios valores en un solo control, los cuales se mantienen visibles.
ListView	lv	Vista de lista, permite mostrar información matricial en un solo control; se puede usar para listar información tal como se vería en Excel.
MaskedTextBox	txt	Máscara de caja de texto, su trabajo es muy similar al cuadro de texto con la diferencia de que permite personalizar el ingreso mediante máscaras; podríamos usarlo para el ingreso de fechas u horas en un formato adecuado.

Control	Prefijo	Descripción
MonthCalendar	mc	Mes de calendario, permite seleccionar una fecha y ocupa un lugar considerable dentro del formulario.
PictureBox	pb	Cuadro de imagen, permite mostrar una imagen dentro de un marco en el formulario.
RadioButton	rb	Botón de radio, permite asignar múltiples opciones mediante botones pero solo uno de ellos podrá ser seleccionado.
TextBox	txt	Cuadro de texto, permite ingresar un valor de cualquier tipo, los cuales son administrados por el programador.

Ficha: **Contenedores**

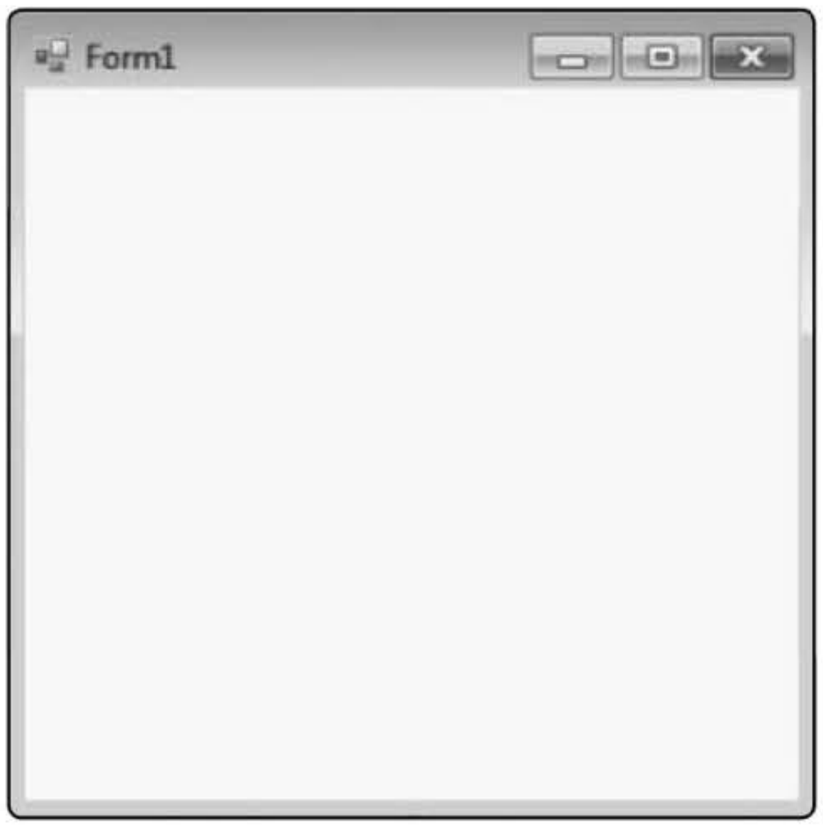
Control	Prefijo	Descripción
GroupBox	gb	Cuadro de grupo, el cual permite agrupar controles dentro de un formulario.
TabControl	tc	Control tabulador, el cual permite agrupar controles dentro de fichas.

Ficha: **Menús y barras de herramientas**

Control	Prefijo	Descripción
MenuStrip	Ms	Menú de opciones, permite implementar un menú de opciones en un formulario.
ToolStrip	ts	Barra de opciones, permite asignar una barra de herramientas en un formulario.

2.4 Control Form

Representa al objeto de la clase Form, el cual permite diseñar una interfaz amigable al usuario final de la aplicación.



Propiedad	Descripción
(Name)	Permite asignar un nombre del control Form.
Acceptbutton	Permite asignar un control de tipo botón como predeterminado; es decir, las acciones del botón se realizarán al presionar la tecla Enter.
BackColor	Permite asignar un color de fondo al formulario.
BackgroundImage	Permite asignar una imagen de fondo al formulario.
BackgroundImageLayout	Permite definir cómo se mostrará la imagen de fondo del formulario asignado con la propiedad BackGroungImage.
ControlBox	Activa y desactiva los botones de control del formulario (maximizar, minimizar y salir).
FormBorderStyle	Especifica la apariencia que tendrán los bordes del formulario.
Locked	Bloquea y desbloquea el cambio de tamaño del formulario en tiempo de diseño.
MaximizeBox	Activa y desactiva el control maximizado del formulario.
MinimizeBox	Activa y desactiva el control minimizado del formulario.
Size	Determina el tamaño en ancho y alto del formulario.
StartPosition	Determina la posición inicial del formulario al correr la aplicación.
Text	Muestra el título del formulario.
WindowState	Determina el estado inicial del formulario al ejecutar la aplicación; se refiere al maximizar, restaurar o minimizar.

Ahora, algunas opciones que se pueden presentar sobre el objeto Form:

- Asignar el nombre al formulario.
  - ➢ Seleccione el formulario desde su barra de título.
  - ➢ En la propiedad (Name) asigne un nombre según la notación de asignación de nombres.
- Asignar un botón como predeterminado. Es decir, cuando el usuario quiera seleccionar un determinado botón lo debe realizar con la tecla Enter. Tenga en cuenta que, desde un conjunto de botones, uno solo podrá tener la propiedad de predeterminado.
  - ➢ Seleccione el formulario desde su barra de título.
  - ➢ Seleccione el nombre del botón desde la propiedad AcceptButton.
- Asignar y quitar un color de fondo al formulario.
  - ➢ Seleccione el formulario desde su barra de título.
  - ➢ En la propiedad **BackColor > Personalizado** y seleccione un color.

- Para eliminar el color de fondo.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad BackColor borre el contenido con la tecla Suprimir.
- Asignar y quitar una imagen de fondo.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad BackgroungImage, seleccione tres puntos **(…) > Recurso local > Importar**.
  - Seleccione la imagen y presione el botón **Aceptar**.
- Para quitar la imagen de fondo.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad BackgroungImage borre el contenido con la tecla Suprimir.
- Desactivar los botones de control minimizar, maximizar y cerrar.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad ControlBox seleccione la opción **False**.
- Establecer un valor fijo en ancho y alto del formulario.
  - Primero, asigne un ancho y alto adecuado al formulario, esto lo puede realizar desde las esquinas del formulario porque presenta controladores de tamaño.
  - En la propiedad FormBorderStyle seleccione **FixedSingle**.
  - Y en la propiedad ControlBox seleccione **False**.
- Asignar la posición central del formulario en la pantalla al ejecutar una aplicación.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad StartPosition seleccione **CenterScreen**.
- Asignar un título al formulario.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad Text asigne un texto.
- Asignar el estado de maximizado al formulario.
  - Seleccione el formulario desde su barra de título.
  - En la propiedad WindowState seleccione **Maximized**.
- Colocar controles en el formulario.
  - Primera forma:
    - Arrastre un control desde el cuadro de herramientas hacia el formulario.

- Segunda forma:
  - Haga doble clic sobre un control desde el cuadro de herramientas.

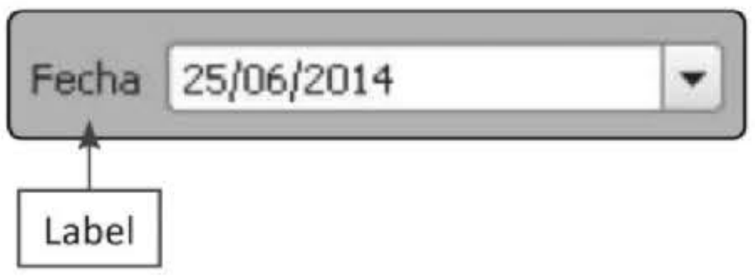
- Replicar controles en el formulario.
  - Agregue un control al formulario.
  - Haga clic en la tecla <Ctrl>, arrastre el control; tendrá una copia exacta del control, inclusive el contenido será similar, pero la diferencia se encuentra en su nombre.

2.5 Controles de visualización de información

Estos controles permiten mostrar información que permitirá al usuario a operar de la mejor manera una determinada aplicación Visual Basic.

2.5.1 Control Label

Representa al objeto de la clase Label, que permite colocar textos estáticos dentro de un formulario, los cuales pueden mostrar un texto descriptivo o una información importante al usuario.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto Label. Hay que tener en cuenta que si el objeto es un texto informativo al usuario no será necesario modificar su nombre; en cambio, si necesita un objeto Label para mostrar el resultado de una aplicación, sí debe asignar un nombre, por ejemplo, **lblSueldo**, que permitirá mostrar el sueldo como resultado de un proceso de pago.
Autosize	Permite autoajustar el marco que contiene al texto del objeto Label. Si lo activamos, el marco se autoajusta en contenido.
BorderStyle	Permite establecer un estilo de borde al objeto Label. Tenemos None, FixedSingle, Fixed3D; que simplemente son estilos de borde del control.
Font	Permite definir la fuente, estilo, tamaño y efectos al contenido del objeto Label.
Forecolor	Permite asignar un color al texto contenido dentro del objeto Label.
Locked	Permite bloquear en movimiento y tamaño al objeto Label dentro del formulario; cuando intente realizar una de estas acciones aparecerá un candado al costado del objeto Label.
Text	Permite asignar un contenido al objeto Label; este será visible por el usuario.

Ahora, el siguiente caso práctico que muestra información del sistema:

Código de solución:

```
Public Class frmLabel
Private Sub btnMostrar_Click(sender As Object, e As EventArgs) Handles
btnMostrar.Click
    lblMaquina.Text = Environment.MachineName
    lblUsuario.Text = Environment.UserName.ToString
    lblSistema.Text = Environment.OSVersion.ToString
    lbl64.Text = Environment.Is64BitOperatingSystem
End Sub
End Class
```

La clase Environment proporciona información acerca del entorno y la plataforma actual; es a partir de sus propiedades que se puede obtener información del sistema actual.

2.5.2 Control LinkLabel

Representa al objeto de la clase LinkLabel, el cual permite colocar textos estáticos con enlaces a URL; es decir, se puede visitar una página web desde el formulario.

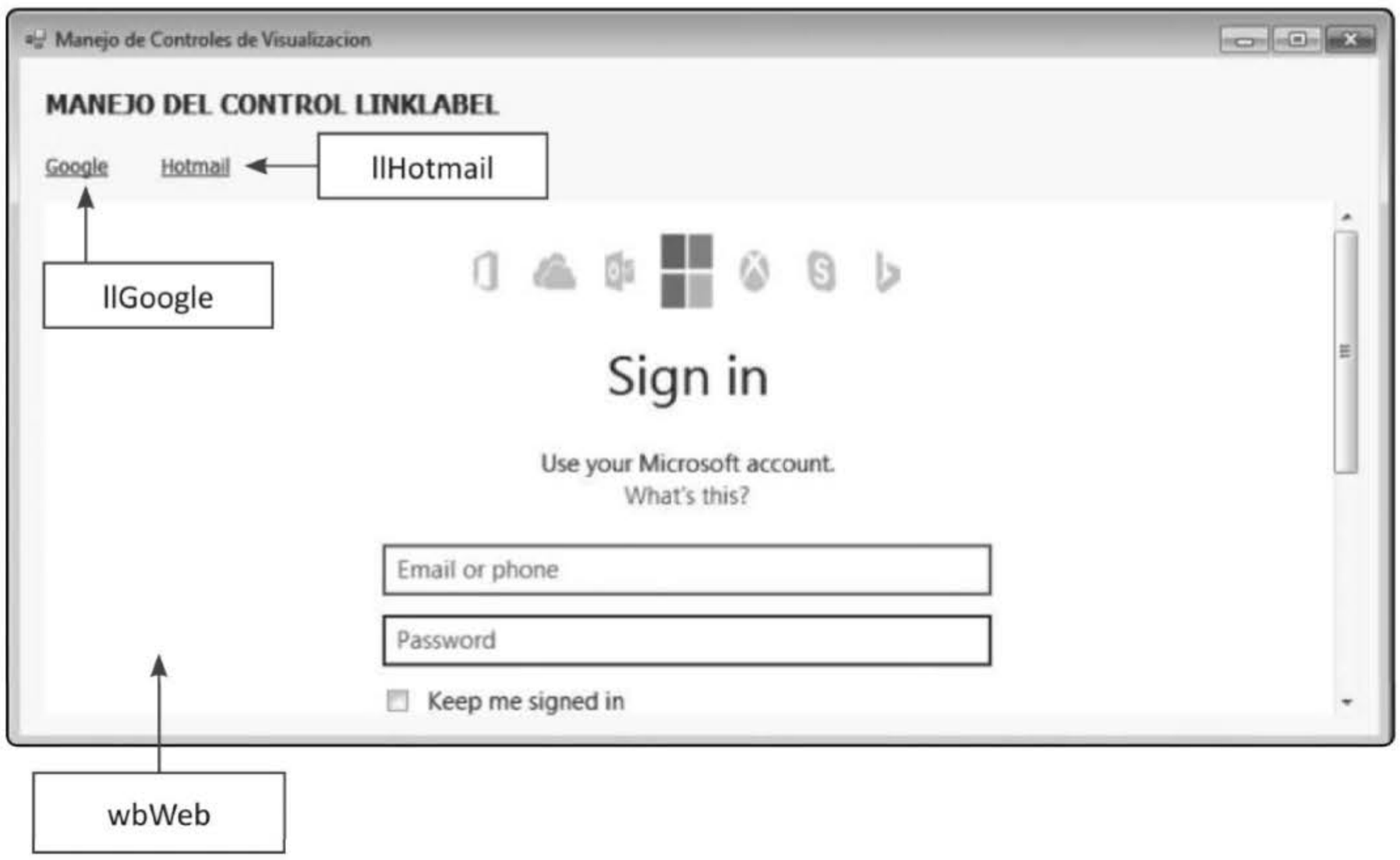
Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto LinkLabel. Por ejemplo, para el enlace «Más información de la clase LinkLabel» se asigna el nombre lnkInformacion.
ActiveLinkColor	Define el color del enlace activo.
Font	Permite definir la fuente, estilo, tamaño y efectos al contenido del objeto LinkLabel.
Forecolor	Permite asignar un color inicial al texto contenido dentro del objeto LinkLabel.
LinkColor	Define el color de visita del enlace.
Text	Permite asignar un contenido al objeto LinkLabel; este será visible por el usuario para su respectivo enlace.

Ahora, haga doble clic el código sobre el control para asignar la siguiente url: https://msdn.microsoft.com.

```
Private Sub lnkInformacion_LinkClicked(sender As Object,
                                       e As LinkLabelLinkClickedEventArgs)
                                        Handles lnkInformacion.LinkClicked
        System.Diagnostics.Process.Start("https://msdn.microsoft.com")
End Sub
```

El método Start permite definir el enlace a ejecutar por parte del objeto LinkLabel, de tal forma que carga un navegador web y muestra dicho enlace.

Ahora, una aplicación que permite mostrar el contenido de una web en un control **WebBrowser** usando el control LinkLabel:

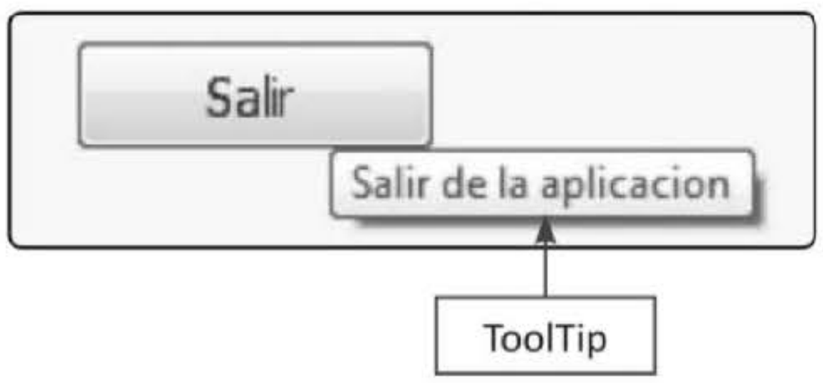


Código de solución:

```
Public Class frmLinkLabel
    Private Sub llGoogle_LinkClicked(…) Handles llGoogle.LinkClicked
        wbWeb.Navigate("http://www.google.com")
    End Sub

    Private Sub llHotmail_LinkClicked(…) Handles llHotmail.LinkClicked
        wbWeb.Navigate("http://www.hotmail.com")
    End Sub
End Class
```

2.5.3 Control ToolTip

Representa al objeto de la clase ToolTip, el cual permite mostrar un texto informativo de un determinado control con solo posicionar el puntero del mouse encima de dicho control.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ToolTip. Por ejemplo, para todo tipo de mensaje ToolTip dentro del formulario, se puede nombrar como ttMensaje.

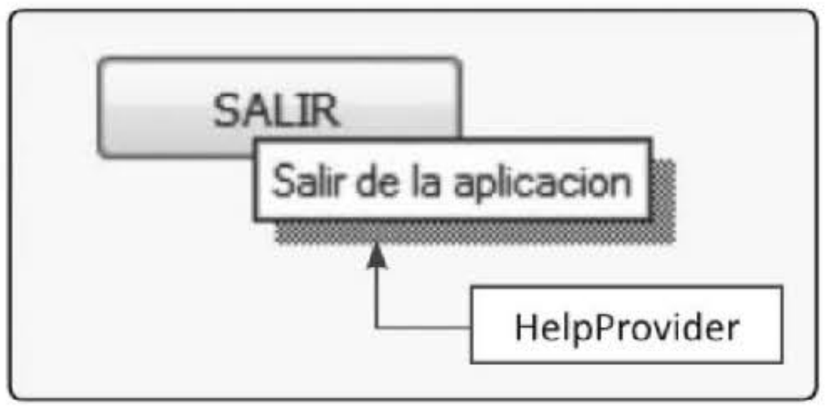
Ahora, el código que permite mostrar un mensaje ToolTip en un botón.

```
Private Sub frmPrueba_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        ttMensaje.SetToolTip(btnSalir, "Salir de la aplicacion")
        ttMensaje.SetToolTip(btnProcesar, "Muestra resultados de la aplicación")
End Sub
```

La asignación de mensajes para cualquier control debe realizarse en el evento load del formulario; asimismo, debe considerar que un solo objeto de tipo ToolTip puede asignar mensajes a varios controles dentro de un mismo formulario. El método SetToolTip es el que define el mensaje especificando primero el nombre del objeto a quien le asignara el mensaje. Al final, la especificación del mensaje se debe realizar mediante comillas.

2.5.4 Control HelpProvider

Representa al objeto de la clase HelpProvider, el cual permite mostrar un texto informativo desde un determinado control con solo posicionar el puntero del mouse encima de dicho control y haciendo clic en la tecla <F1>.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto HelpProvider. Por ejemplo, para todo tipo de mensaje HelpProvider dentro del formulario se puede nombrar como hpMensaje.

Ahora, el código que permite mostrar un mensaje HelpProvider en un determinado botón.

```
Private Sub frmPrueba_Load(sender As Object, e As EventArgs) Handles
MyBase.Load

        hpMensaje.SetHelpString(btnSalir, "Salir de la aplicación")

End Sub
```

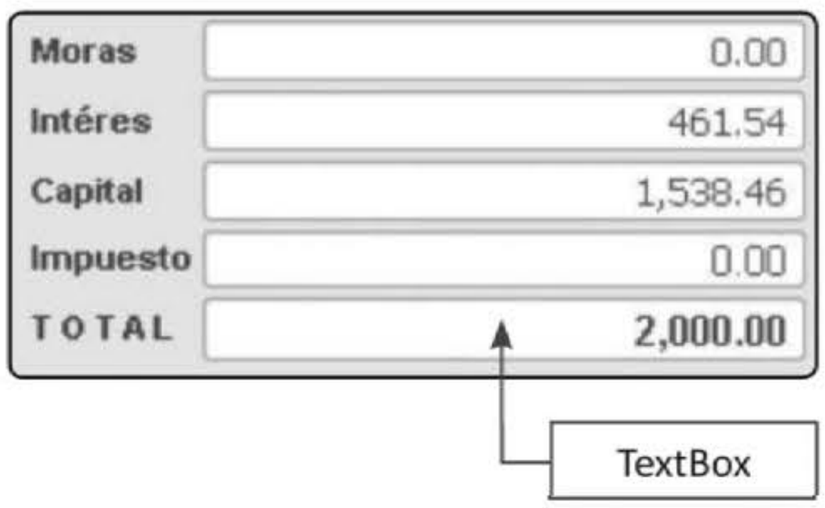
La asignación de mensajes se realiza mediante el método SetHelpString, haciendo referencia al control; también se debe considerar que varios controles pueden definir sus mensajes mediante un solo HelpProvider y que para observar el mensaje se debe ubicar el puntero del mouse sobre el control y hacer clic en la tecla <F1>.

2.6 Controles de ingreso de información

Estos controles permiten ingresar valores a una aplicación Visual Basic, de tal forma que el usuario podrá ingresar datos de cualquier tipo que la aplicación tendrá que procesar.

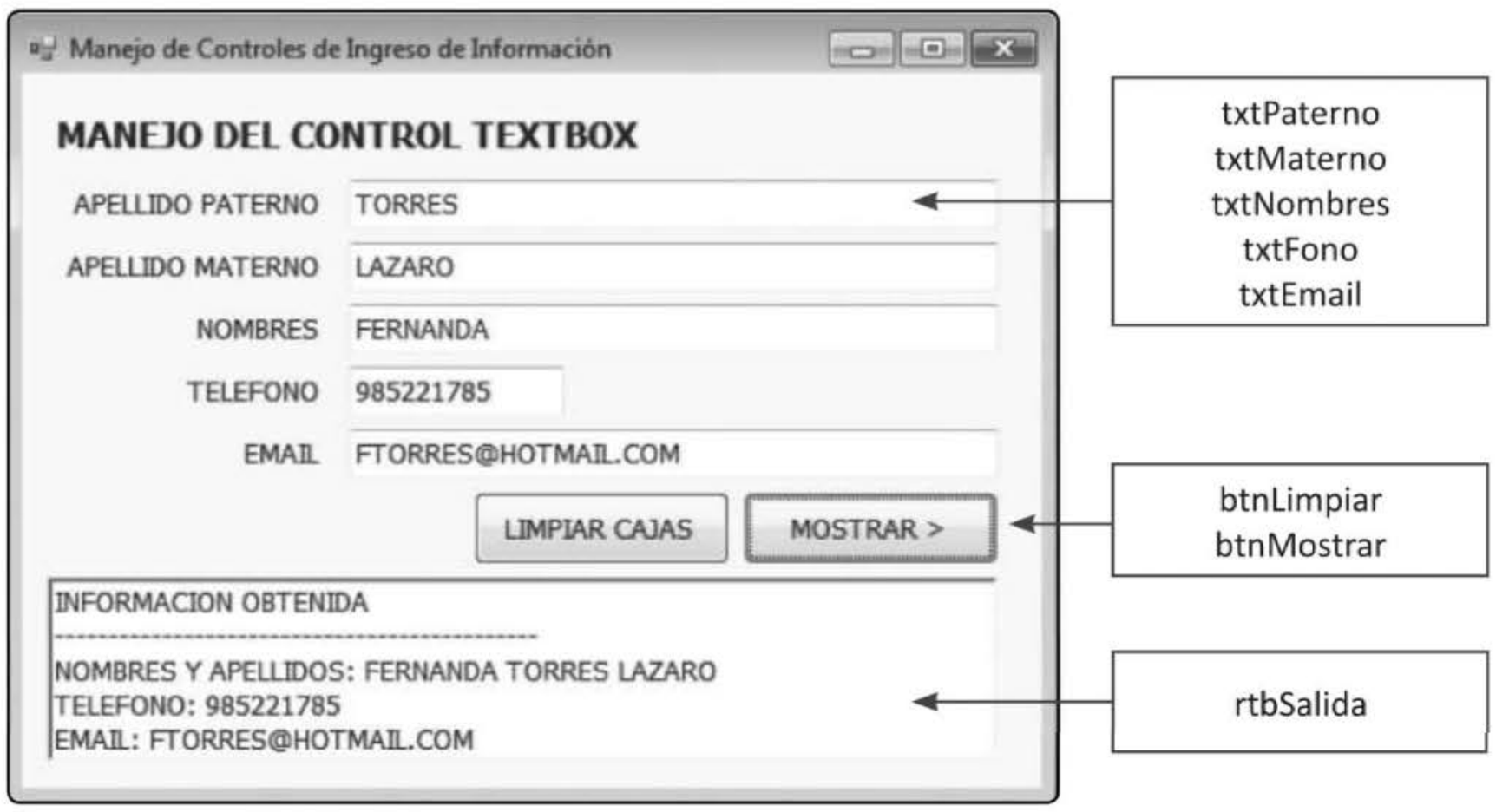
2.6.1 Control TextBox

Representa al objeto de la clase TextBox, el cual permite ingresar textos dentro de un marco de una línea. Es el único control donde el usuario podrá ingresar un determinado valor requerido por la aplicación; hay que tener en cuenta que dicho valor siempre será de tipo texto.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto TextBox1. La mayoría de los objetos Text colocados dentro de un formulario deben tener un nombre, ya que el valor ingresado servirá en el código de programación. Por ejemplo, para un ingreso de un nombre de usuario podría llamarse txtNombre.
Enabled	Permite habilitar o inhabilitar el ingreso de valor en un objeto Text. Se habilita para el ingreso de valor y se inhabilita para mostrar alguna información; por ejemplo, la fecha actual.
MaxLength	Define la longitud máxima de caracteres que puede contener un cuadro de texto; por ejemplo, si se solicita el DNI de una persona se asignará 8 y en el caso de ingresar una edad se asignará 2.
Multiline	Permite activar la escritura en varias líneas dentro del mismo objeto TextBox; esto servirá para ingresar valores extensos, como un comentario.
PasswordChar	Permite asignar un carácter enmascarado al texto ingresado. Su trabajo es similar a las claves solicitadas en los correos electrónicos. Por ejemplo, si se necesita mostrar asteriscos solo asigne un asterisco a la propiedad PasswordChar.
ReadOnly	Permite bloquear la modificación del texto contenido en el objeto TextBox.
ScrollBars	Permite asignar barras de desplazamientos cuando el objeto TextBox tiene asignado true en la propiedad MultiLine.
Text	Permite asignar un valor al control TextBox; no se recomienda su ingreso ya que es el usuario el que debe ingresar un valor en tiempo de ejecución.
TextAlign	Permite alinear el texto contenido en un objeto TextBox; se podría alienar a la derecha cuando se imprima valores monetarios.

Aquí se ve cómo usar el control TextBox mediante una aplicación básica de ingreso de valores y la muestra de los mismos en un control RichTextBox, tal como está en la siguiente ventana:



Tener en cuenta:

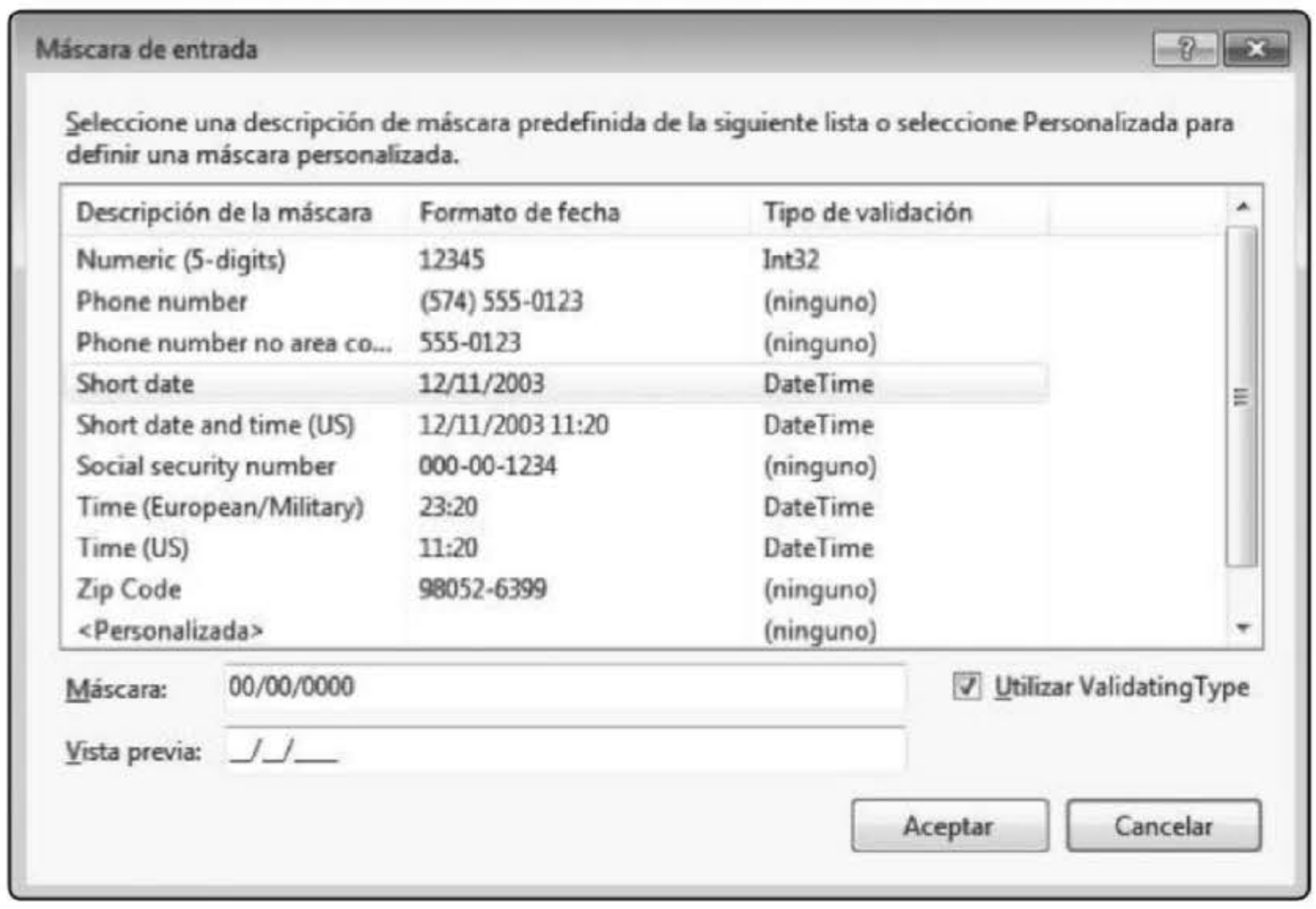
- En el valor del teléfono solo permitirá ingresar nueve valores.
- Para mostrar la información obtenida use el botón **Mostrar**.
- La información resultante debe mostrarse en un control RichTextBox.
- El botón **Limpiar cajas** limpiará todos los controles de tipo TextBox usando la estructura repetitiva For Each.

Código de solución:

```
Public Class frmTextBox
      Private Sub frmTextBox_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
        'Asignar una longitud de nueve caracteres al teléfono
        txtFono.MaxLength = 9
    End Sub

    Private Sub btnMostrar_Click(…) Handles btnMostrar.Click
      'Enviar información al control RichTextBox
      rtbSalida.AppendText("INFORMACION OBTENIDA" & vbNewLine)
      rtbSalida.AppendText("---------------------------" & vbNewLine)
      rtbSalida.AppendText("NOMBRES Y APELLIDOS: " & txtNombres.Text & " " &
                           txtPaterno.Text & " " & txtMaterno.Text & vbNewLine)
      rtbSalida.AppendText("TELEFONO: " & txtFono.Text & vbNewLine)
      rtbSalida.AppendText("EMAIL: " & txtEmail.Text)
    End Sub

    Private Sub btnLimpiar_Click(…) Handles btnLimpiar.Click
      'Recorriendo por todos los controles del formulario
      For Each misObjetos In Me.Controls
         'Identificar si los controles son TextBox
         If TypeOf misObjetos Is TextBox Then
               misObjetos.Text = ""
         End If
      Next
      txtPaterno.Focus()
    End Sub
End Class
```

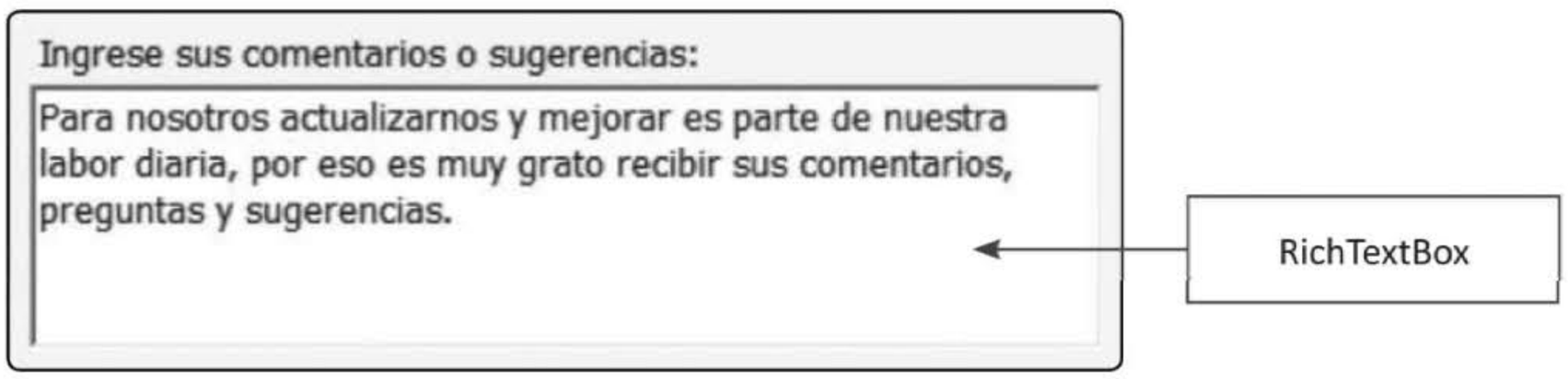
2.6.2 Control MaskedTextBox

Representa al objeto de la clase MaskedTextBox, el cual permite ingresar textos en un determinado formato; por ejemplo, el ingreso de una fecha, hora o valores específicamente numéricos.

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto MaskedTextBox. Como recomendación, se asignarán los nombres mediante las iniciales txt, por su parecido al control TextBox.
Mask	Permite definir un formato de entrada, el cual solo permitirá ingresar valores que cumplan con la especificación de máscara.  Desde la ventana de máscara de entrada seleccione un determinado formato; el valor mostrado en **Vista previa** es el mismo valor que el usuario visualizará, además se debe considerar que la activación de **Utilizar ValidatingType** permite validar el valor ingresado por el usuario, bloqueando los caracteres que no cumplen con el formato.

2.6.3 Control RichTextBox

Representa al objeto de la clase RichTextBox; es considerado como el control de texto enriquecido dentro del cual se puede escribir y editar un texto. Asimismo, se puede aplicar cualquier tipo de formato como el cambio de fuente, asignación de color, etc. Para poder mantener el contenido textual con el formato enriquecido se debe grabar dicha información en un archivo con extensión **RTF**.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto RichTextBox. Si, por ejemplo, se necesita un ingreso de comentarios o sugerencias se podría asignar el nombre rtbComentarios.

2.7 Controles de acción

Estos controles permiten accionar una línea de proceso implementada en la aplicación; es decir, a partir de aquí se podrá ver cómo reacciona una aplicación frente a los datos ingresados por el usuario.

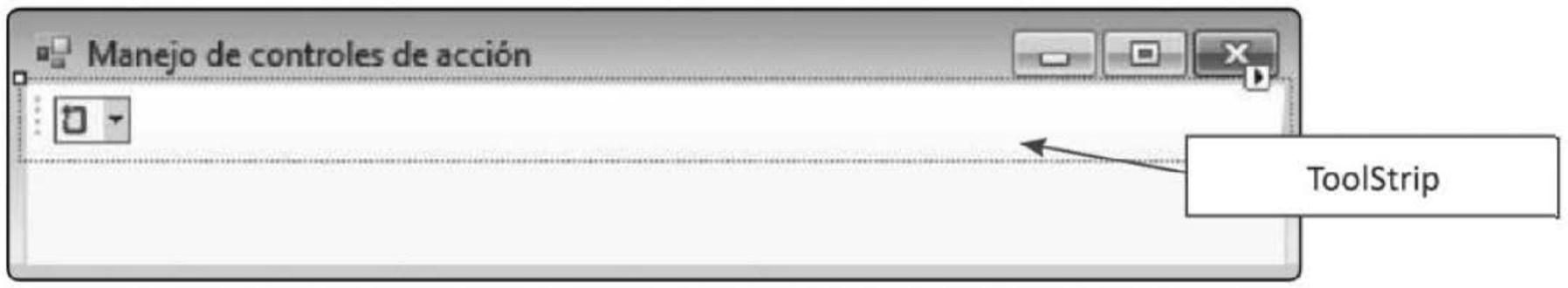
2.7.1 Control Button

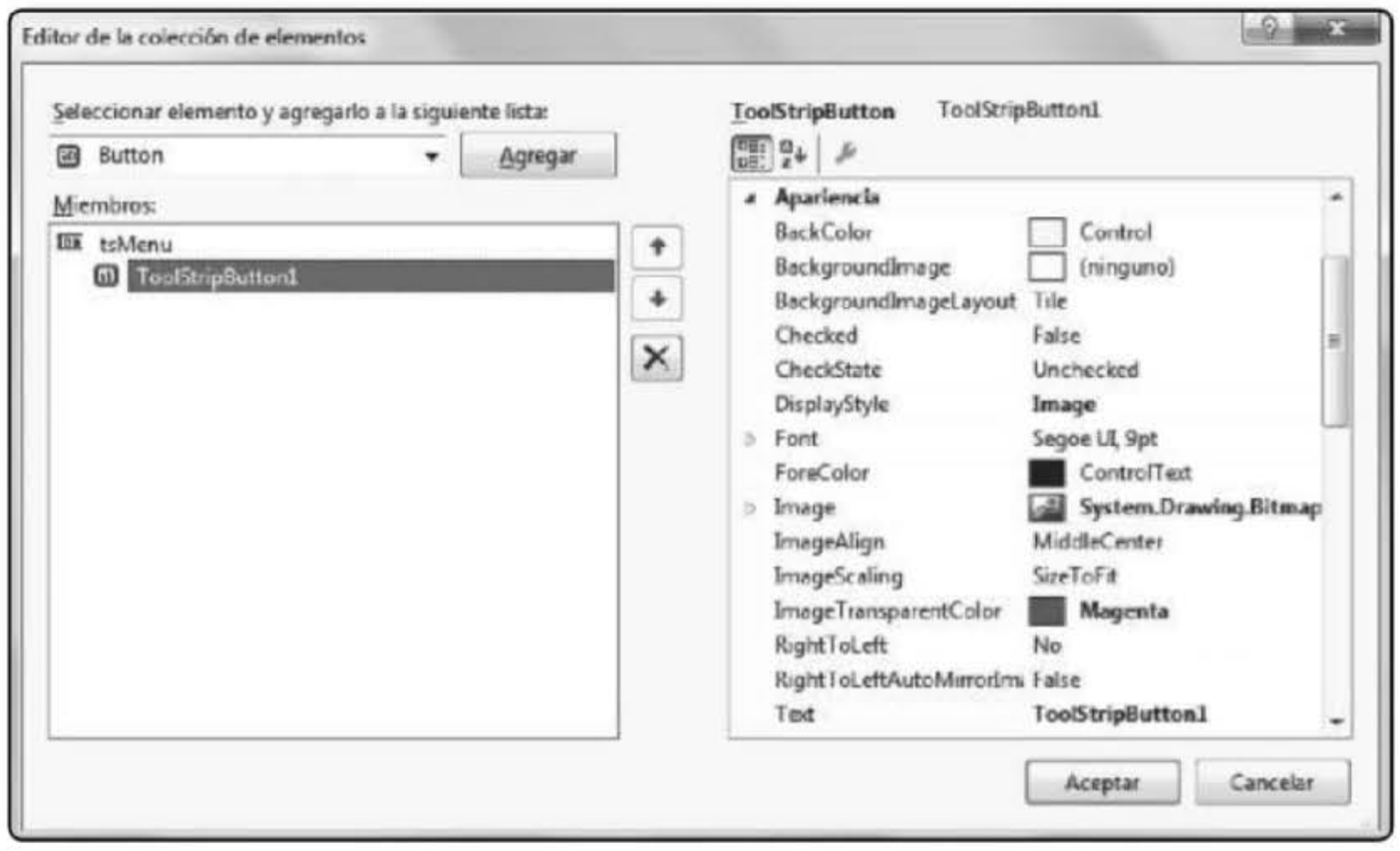
Representa al objeto de la clase Button, el cual permite realizar una acción según el código implementado por el programador. Se debe tener en cuenta que los botones son los más adecuados de usar al momento de mostrar resultados en una aplicación.

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto Button. Por ejemplo, para un botón de registro podría llamar btnRegistrar.
Enabled	Permite habilitar o inhabilitar al botón; esto podría habilitarse por algún criterio dentro de la aplicación.
Image	Permite asignar una imagen dentro del botón.
ImageAlign	Permite alinear la imagen dentro del botón.
Text	Permite asignar un título al botón.
TextAlign	Permite alinear el texto colocado en el título del botón.
Visible	Permite mostrar y ocultar el botón, este evento se puede controlar mediante código en el proceso colocando **btnSalir.Visible = True** para mostrar y **btnSalir.Visible = False** para ocultar el botón.

2.7.2 Control ToolStrip

Representa al objeto de la clase ToolStrip, el cual permite crear barra de herramientas administrables en tiempo de diseño. A partir de esta barra de herramientas, el usuario podrá controlar todos los procesos que contiene su aplicación.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ToolStrip. En este caso, son importantes los botones y opciones que se coloquen dentro de la barra; no será necesario modificar el nombre del control ToolStrip.
Items	Permite añadir un elemento de tipo Button, Label, Combobox al control TooStrip.  Para comenzar a agregar elementos al control ToolStrip, se debe seleccionar un elemento desde la lista y con el botón **Agregar**, añadirlo a la lista de miembros; luego, modifique las siguientes propiedades: • (**Name**): Para asignar un nombre al control contenido, por ejemplo, tsMostrarLista. • **DisplayStyle**: Para modificar la forma de visualización del elemento por Image, se muestra una imagen en el control e Image and Text muestra la imagen con un texto descriptivo del botón. • **Text**: Define el contenido que mostrará el control. • **Image**: Define la imagen que mostrará el control.

Agregar botones al ToolStrip

- Seleccione la barra ToolStrip.
- Seleccione Button desde .

Mostrar botón y texto al mismo tiempo

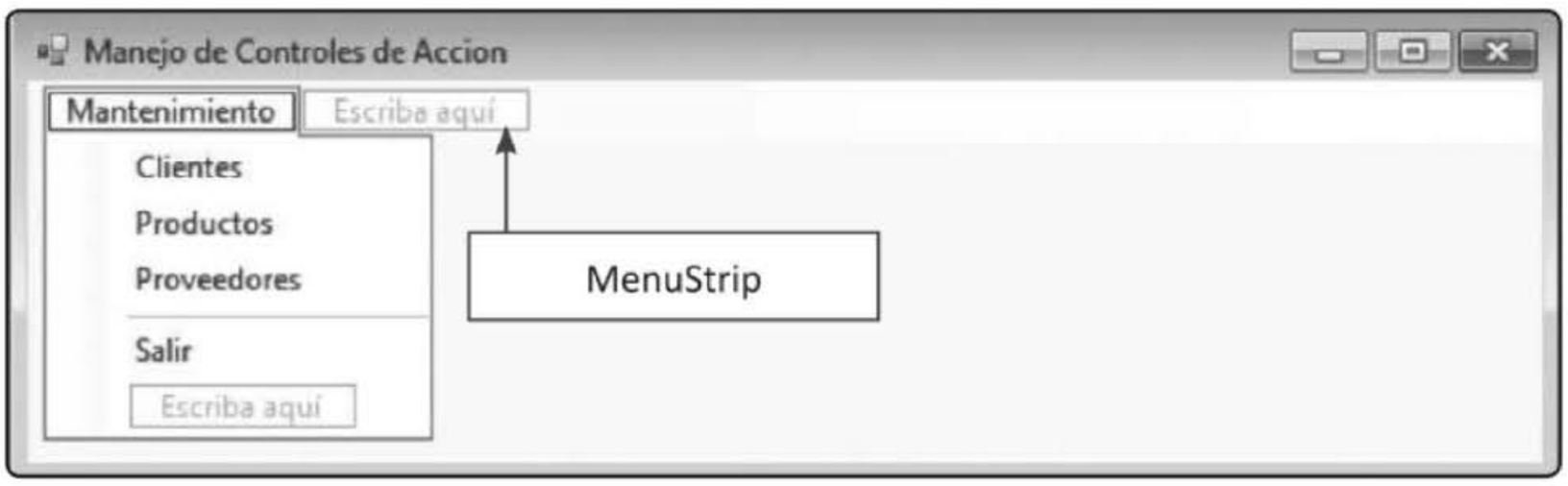
- Seleccione una opción del MenuStrip.
- Desde la ventana de propiedades seleccione DisplayStyle.
- Seleccione **Image and Text**.

Eliminar una opción del ToolStrip

- Seleccione una opción del ToolStrip.
- Haga clic en la tecla Suprimir.

2.7.3 Control MenuStrip

Representa al objeto de la clase MenuStrip, el cual permite crear barra de menú administrable en tiempo de diseño. A partir de esta barra se expone la funcionalidad de una aplicación agrupando bloques de opciones que el usuario final puede seleccionar.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto MenuStrip. Para este caso, interesan las opciones que se coloquen dentro de la barra de menú y estos asignen directamente un nombre a partir de la descripción que se le asigne a cada opción. Por ahora no será necesario modificar el nombre del control MenuStrip.

Agregar opciones al MenuStrip

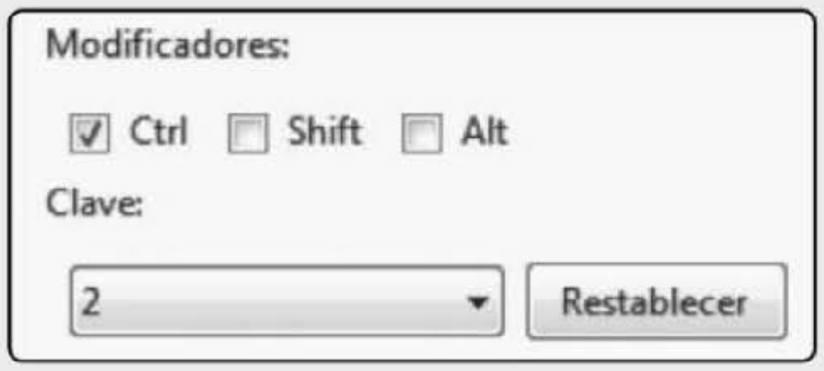
- Seleccione la barra MenuStrip
- Asigne una descripción de las opciones en **Escriba aquí**. Asimismo, si necesitamos mostrar el texto de la opción con una letra subrayada, asigne el símbolo & antes de la letra a subrayar; por ejemplo, Mantenimiento se escribiría Mante&nimiento.

Asignar un ícono a las opciones del MenuStrip

- Seleccione una opción del MenuStrip.
- Desde la ventana de propiedades seleccione **Image**.
- Seleccione **Importar** desde Recurso local.
- Selección una imagen desde su unidad de disco.
- Finalmente, seleccione el botón **Aceptar**.

Asignar un método de acceso rápido al MenuStrip

- Seleccione una opción del MenuStrip.
- Desde la ventana de propiedades seleccione **ShortcutKeys**.
- A partir de la siguiente, seleccione la combinación de tecla para el acceso rápido de la opción del menú, por ejemplo, active Ctrl y la clave 2 para una combinación del <Ctrl+2>, en orden de activar una opción de menú.



Tenga en cuenta que el botón **Restablecer** reinicia la selección actual y anula toda actividad realizada en esta ventana.

Eliminar una opción del MenuStrip

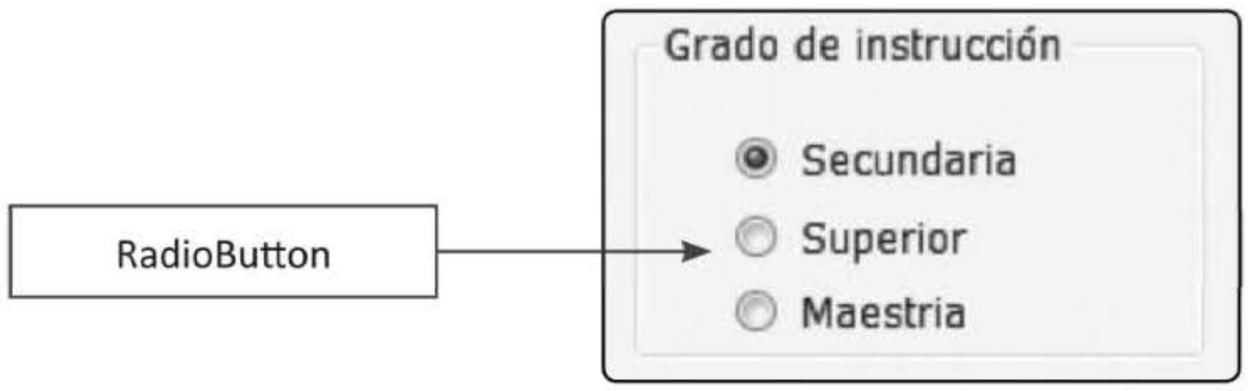
- Seleccione una opción del MenuStrip.
- Presione la tecla Suprimir.

2.8 Controles de opción simple y múltiple

Estos controles permiten seleccionar una o más opciones dentro de un mismo control. Para poder usar toda su potencialidad se les debe integrar a la estructuras de decisión como If o Select Case.

2.8.1 Control RadioButton

Representa al objeto de la clase RadioButton, el cual muestra un texto, imagen o ambos, desde los cuales se puede seleccionar una u otra opción; esto hace que al seleccionar uno de ellos los demás se desenfoquen de la selección del usuario. Asimismo, se debe tener en cuenta que cuando se colocan varios RadioButton se genera un grupo según el control; por ejemplo, colocar RadioButton dentro de un formulario hace que este último sea su contenedor. Para agregar un nuevo grupo de opciones dentro del mismo formulario se deben usar controles de agrupación como los GroupBox o Panel.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto RadioButton. Por ejemplo, para especificar un grado de instrucción de un empleado se puede usar el nombre rbSuperior.
Checked	Establece el modo activo del control Radiobutton al iniciar la aplicación.
Enabled	Establece el bloqueo o habilitación del control RadioButton; si se encuentra bloqueado, no responderá a ningún evento solicitado por el usuario.
ImageAlign	Establece una alineación a la imagen asignada al control Radiobutton.
Text	Permite establecer el texto que se mostrará al usuario.
Visible	Permite mostrar y ocultar el control RadioButton; este evento se puede controlar mediante código en el proceso colocando rbDolares. Visible = True para mostrar y rbDolares.Visible = False para ocultar el RadioButton.

Cuando se necesita determinar si una opción se encuentra activa o no, se puede usar el siguiente código:

```
If rbA.Checked Then
   MessageBox.Show("Opción A seleccionada")
   Else
   MessageBox.Show("Otra opción seleccionada")
End If
```

La propiedad Checked evalúa con un valor estándar en True al no especificarlo en una estructura condicional.

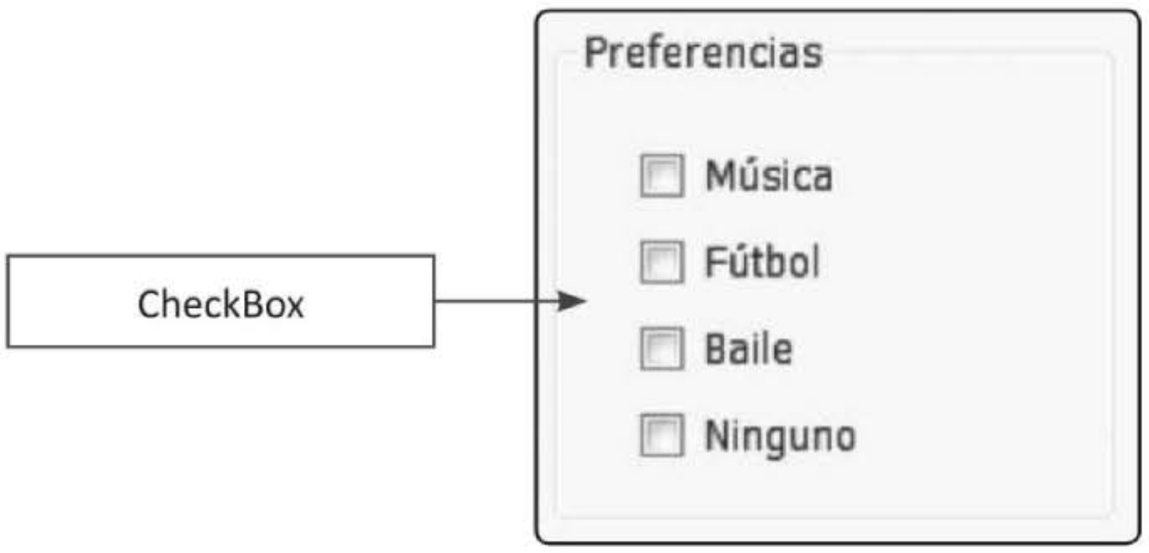
Ahora, un caso básico del uso de los controles RadioButton; esta aplicación modificará el idioma de los textos solicitados, tal como se muestra en la siguiente ventana:

Código de solución:

```
Public Class frmRadioButton
    Private Sub rbEspañol_CheckedChanged(…) Handles rbEspañol.CheckedChanged
        lblUsuario.Text = "USUARIO"
        lblClave.Text = "CLAVE"
    End Sub

    Private Sub rbIngles_CheckedChanged(…) Handles rbIngles.CheckedChanged
        lblUsuario.Text = "USER"
        lblClave.Text = "PASSWORD"
    End Sub
End Class
```

2.8.2 Control CheckBox

Representa al objeto de la clase CheckBox; este control es también llamado casilla, el cual indica si una opción está activa o inactiva. El uso del control CheckBox se da mayormente para mostrar grupos de opciones desde el cual un usuario podrá seleccionar una o más casillas según le convenga. Se recomienda que este control se encuentre agrupado dentro de un control GroupBox para generar un orden entre todas las casillas.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto CheckBox. Por ejemplo, para especificar un tipo de preferencia se puede usar el nombre chkMusica.
Checked	Establece si el control se encuentra activado. También puede especificar como activa una de las casillas frente a los demás controles del mismo tipo.
Enabled	Establece el bloqueo o habilitación del control CheckBox; si se encuentra bloqueado no responderá a ningún evento solicitado por el usuario.
Text	Permite establecer el texto que se mostrara al usuario.
Visible	Permite mostrar y ocultar el control CheckBox.

Si se necesita verificar la selección de un control CheckBox se puede usar el siguiente código:

```
If chkA.Checked = True Then
    MessageBox.Show("Check A seleccionada")
    Else
    MessageBox.Show("Otro check seleccionado")
End If
```

La propiedad Checked devuelve true o false dependiendo de la selección del control. Si se encuentra activo entonces la propiedad Checked es igual a True, caso contrario su valor es False.

A continuación, un caso práctico básico que use los controles Checkbox, de tal forma que un usuario pueda seleccionar sus preferencias y luego mostrar separadamente las activas de las inactivas, como se muestra en la siguiente imagen:

Código de solución:

```
Public Class frmCheckBox
    Private Sub btnMostrar_Click(…) Handles btnMostrar.Click
     'Limpiar la caja de salida
     txtPreferencias.Clear()

     'Verificando las preferencias seleccionadas
    txtPreferencias.AppendText("Preferencias activas: " & vbNewLine)
     txtPreferencias.AppendText("---------------------" & vbNewLine)
     For Each misObjetos In Me.Controls
         'Identificar si los controles son CheckBox
         If TypeOf misObjetos Is CheckBox Then
             If misObjetos.Checked Then
              txtPreferencias.AppendText(misObjetos.Text & vbNewLine)
```

```
                End If
            End If
        Next

        'Verificando las preferencias no seleccionadas
    txtPreferencias.AppendText(vbNewLine & "Preferencias NO activas: " & vbNewLine)
     txtPreferencias.AppendText("----------------------" & vbNewLine)
     For Each misObjetos In Me.Controls
         'Identificar si los controles son CheckBox
         If TypeOf misObjetos Is CheckBox Then
             If misObjetos.Checked = False Then
              txtPreferencias.AppendText(misObjetos.Text & vbNewLine)
             End If
         End If
     Next
 End Sub
End Class
```

Para poder identificar los controles de tipo CheckBox se usa la estructura repetitiva For Each, la cual recorre por todos los controles colocados en el formulario. Luego, se identifica al control con la sentencia Is CheckBox, y para el encuentro de los chequeos activos se le debe buscar mediante la sentencia **misObjetos.Checked"** donde misObjetos representa a todos los controles de tipo CheckBox.

Cuando se agrega las preferencias seleccionadas al control TextBox como respuesta, se usa la cláusula AppendText para añadir al final de la línea un texto nuevo y la función vbNewLine para realizar un cambio de línea.

2.8.3 Control ComboBox

Representa al objeto de la clase ComboBox, el cual permite mostrar una lista desplegable de valores que se contrae una vez seleccionada una opción.

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ComboBox. Por ejemplo, un cuadro combinado para listar países podría llamarse cboPais.

DropDownStyle	Permite modificar el estilo que muestra el cuadro combinado; si, por ejemplo, se asigna Simple, se mostrarán los valores sin contraerse. DropDown es el valor estándar y DropDownList no permite editar dentro del control.
Items	Permite asignar valores dentro del cuadro combinado.
Sorted	Permite ordenar los valores mostrados dentro del cuadro combinado.
Text	Permite asignar un texto de entrada al cuadro combinado y, una vez que el usuario selecciona una opción del objeto, este valor se pierde.

Hay que tener en cuenta que la propiedad Items permite llenar de opción el cuadro combinado, pero se recomienda, como desarrolladores, realizarlo mediante código, tal como se muestra a continuación:

```
Private Sub Form_Load(sender As Object, e As EventArgs) Handles MyBase.Load
   cboPais.Items.Add("Peru")
   cboPais.Items.Add("Argentina")
   cboPais.Items.Add("Mexico")
   cboPais.Items.Add("Ecuador")
   cboPais.Items.Add("Colombia")
End Sub
```

Tal como se observa, el código de asignación de opciones para el cuadro combinado debe realizarse en el evento Load del formulario, ya que, al cargar el formulario, el control debe precargar sus opciones.

A continuación, un caso práctico sobre el uso del control ComboBox. Una empresa de venta de electrodomésticos necesita determinar el precio de venta a sus clientes mediante una aplicación, tal como se muestra aquí:

Tener en cuenta los siguientes criterios:

- Los productos deberán estar precargados en el control cuadro combinado al iniciar la aplicación.

- Mostrar el precio de lista, el porcentaje de ganancia (12.7 % del precio de lista) y el precio de venta al público, el cual resulta del precio de lista aumentando la ganancia.
- Los precios de los productos se muestran en la siguiente tabla:

Descripción del producto	Precio unitario
Lavadora	$1500.00
Refrigeradora	$4500.00
Televisión	$3500.00
Radiograbadora	$2100.00

- Código de solución:

```
Public Class frmComboBox
     Private Sub frmComboBox_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
         'Agregando productos al cuadro combinado
         cboProducto.Items.Add("Lavadora")
         cboProducto.Items.Add("Refrigeradora")
         cboProducto.Items.Add("Television")
         cboProducto.Items.Add("Radiograbadora")
     End Sub

     Private Sub cboProducto_SelectedIndexChanged(…) Handles …
         'Asignando el precio del producto
         Dim precio As Double
         Select Case cboProducto.Text
             Case "Lavadora" : precio = 1500
             Case "Refrigeradora" : precio = 4500
             Case "Television" : precio = 3500
             Case "Radiograbadora" : precio = 2100
         End Select
         lblLista.Text = precio.ToString("C")

         'Calculando la ganancia desde el precio de lista
         Dim ganancia As Double = precio * (17.5 / 100)
         lblGanancia.Text = ganancia.ToString("C")

         'Calculando el previo de venta
         Dim venta As Double = precio + ganancia
         lblVenta.Text = venta.ToString("C")
     End Sub
End Class
```

Para poder mostrar los productos precargados en el cuadro combinado se debe codificar en el evento load del formulario. Asimismo, para mostrar los resultados esperados por la aplicación se debe codificar en el evento **SelectedIndexChanged** del control ComboBox, ya que, al momento de seleccionar un determinado producto, deberá mostrar los datos solicitados.

Para la asignación de precios de los productos se usa la estructura condicional múltiple Select Case; mediante el uso de la variable precio será asignado al control correspondiente. Para la impresión del símbolo de moneda se usa la sentencia **ToString(«C»)**.

A continuación, un segundo caso que permite mostrar el uso del evento **SelectedIndexChanged** del control Combobox. Para lo cual implementaremos una aplicación que, al seleccionar un determinado país, muestre las ciudades según le corresponda, tal como se muestra en la siguiente imagen:

Tener en cuenta los siguientes criterios:

- Al iniciar la aplicación se deberá mostrar precargados los países de Perú, Chile, Colombia y México.
- Al seleccionar un país, deberá cargar las ciudades en el control correspondiente.
- El botón **Mostrar** permitirá mostrar qué país y ciudad se han seleccionado desde los cuadros comandos.

Código de solución:

```
Public Class frmComboBox
      Private Sub frmComboBox_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
         'Agregando países al cuadro combinado
         cboPais.Items.Add("Perú")
         cboPais.Items.Add("Chile")
         cboPais.Items.Add("Colombia")
         cboPais.Items.Add("Mexico")
     End Sub
```

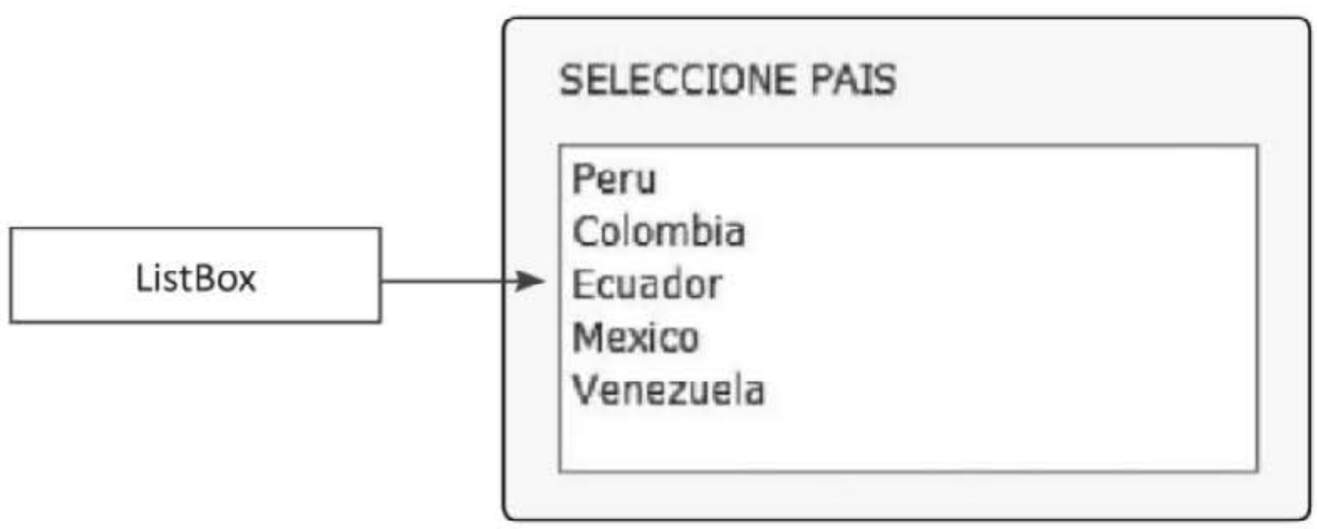
```
        Private Sub cboPais_SelectedIndexChanged(…) Handles cboPais.
SelectedIndexChanged
        'Asignando ciudades dependiendo del país
        Select Case cboPais.Text
            Case "Peru"
                cboCiudad.Items.Clear()
                cboCiudad.Items.Add("Lima")
                cboCiudad.Items.Add("Cuzco")
                cboCiudad.Items.Add("Arequipa")
                cboCiudad.Items.Add("Tacna")
            Case "Chile"
                cboCiudad.Items.Clear()
                cboCiudad.Items.Add("Santiago")
                cboCiudad.Items.Add("Talca")
                cboCiudad.Items.Add("Valparaíso")
                cboCiudad.Items.Add("Viña del Mar")
            Case "Colombia"
                cboCiudad.Items.Clear()
                cboCiudad.Items.Add("Cali")
                cboCiudad.Items.Add("Barranquilla")
                cboCiudad.Items.Add("Bogota")
                cboCiudad.Items.Add("Medellín")
            Case "México"
                cboCiudad.Items.Clear()
                cboCiudad.Items.Add("Guadalajara")
                cboCiudad.Items.Add("Puebla")
                cboCiudad.Items.Add("Monterrey")
                cboCiudad.Items.Add("Acapulco")
        End Select
    End Sub

    Private Sub btnMostrar_Click(…) Handles btnMostrar.Click
        'Obteniendo información desde los cuadros combinados
        lblPais.Text = cboPais.SelectedItem
        lblCiudad.Text = cboCiudad.SelectedItem
    End Sub
End Class
```

2.9 Controles de muestra de información

Estos controles permitirán mostrar información resultante desde un proceso de la aplicación. Esta sección de muestra de información dependerá del programador, pues él decidirá en qué control desea mostrar la información proveniente de la lógica de la aplicación.

2.9.1 Control ListBox

Representa al objeto de la clase ListBox, el cual permite mostrar una lista de valores al usuario, se usa principalmente para listar elementos o para mostrar información resultante en una aplicación. Sus propiedades son muy parecidas al control ComboBox.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ListBox. Por ejemplo, para una lista de países podría llamarse lstPaises.
BorderStyle	Define un tipo de borde al control ListBox.
SelectionMode	Define la forma de selección de los elementos contenidos en el control ListBox. • **One**: Selecciona de uno en uno las opciones. • **MultiSimple**: Selecciona un grupo adyacente de opciones. • **MultiExtended**: Selecciona elementos continuos con ayuda de la tecla <Ctrl>.
Items	Permite asignar valores dentro de la lista.

Hay que tener en cuenta que la propiedad Ítems permite llenar de opción el cuadro de lista, pero se recomienda, como desarrolladores, realizarlos mediante código, tal como se muestra:

```
Private Sub Form_Load(sender As Object, e As EventArgs) Handles MyBase.Load
   lstPaises.Items.Add("Peru")
   lstPaises.Items.Add("Colombia")
   lstPaises.Items.Add("Ecuador")
   lstPaises.Items.Add("Mexico")
   lstPaises.Items.Add("Venezuela")
End Sub
```

Tal como se observa, el código de asignación de opciones para el cuadro de lista debe realizarse en el evento Load del formulario; ya que, al cargar este, el control debe precargar sus opciones.

A continuación, un caso práctico sobre el uso del control ListBox:

Tener en cuenta los siguientes criterios:

- Debe agregar un nombre de país a la lista verificando que no se repita; si se repite, mostrar un mensaje de error. Asimismo, se debe validar el nombre del país en mayúscula, minúscula o la combinación de ellos.
- Cuando se agregue un país a la lista se limpia el control de ingreso para un nuevo valor.
- Debe permitirse agregar el nombre del país con la tecla **Enter**.
- Realice las siguientes operaciones básicas:
  - Eliminar un elemento: Permite eliminar un elemento de la lista previamente seleccionado. En caso no se seleccione un elemento, mostrar un mensaje de error.
  - Borrar lista: Elimina todos los elementos agregados a la lista.
  - Ordenar ascendente: Ordena todos los elementos de forma ascendente.
  - Ordenar descendente: Ordena todos los elementos de forma descendente.
  - Total de elementos: Determina el total de elementos agregados a la lista.

Código de solución:

```
Public Class frmListBox
 Dim listaPaises As New ArrayList
 Private Sub btnAgregar_Click(…) Handles btnAgregar.Click
   If Not String.IsNullOrWhiteSpace(txtPais.Text) Then
     'Verificando la duplicidad del nombre del país
```

```
        For Each p In listaPaises
            If p.ToString.ToUpper = txtPais.Text.ToUpper Then
            MessageBox.Show("El valor ingresado ya se encuentra en la lista..!!","VB")
                txtPais.Clear()
                txtPais.Focus()
                Return
            End If
        Next

        'Enviando el valor a la lista
        listaPaises.Add(txtPais.Text)
        lstPaises.Items.Add(txtPais.Text)
    End If

    'Limpiando y enfocando al control para otro ingreso de país
    txtPais.Clear()
    txtPais.Focus()
End Sub

Private Sub btnEliminar_Click(…) Handles btnEliminar.Click
    'Verificar que el usuario seleccione un elemento de la lista
    If (lstPaises.SelectedItem Is Nothing) Then
    MessageBox.Show("Debe seleccionar un elemento de la lista..!!!", "VB")
    Else
        'Eliminando un elemento de la lista
        lstPaises.Items.Remove(lstPaises.SelectedItem)
    End If
End Sub

Private Sub btnBorrar_Click(sender As Object, e As EventArgs) Handles
btnBorrar.Click
    lstPaises.Items.Clear()
End Sub

Private Sub btnAscendente_Click(…) Handles btnAscendente.Click
    'Enviando los países a un ArrayList
    listaPaises.Clear()
    For Each p In lstPaises.Items
        listaPaises.Add(p)
    Next

    'Aplicando un ordenamiento ascendente
    Dim paises = From p In listaPaises
                 Select p
                 Order By p.ToString Ascending

    'Limpiar la lista para mostrar los valores ordenados
    lstPaises.Items.Clear()
    For Each p In paises
        lstPaises.Items.Add(p)
    Next
End Sub
```

```
Private Sub btnDescendente_Click(…) Handles btnDescendente.Click
  'Enviando los países a un ArrayList
  listaPaises.Clear()
  For Each p In lstPaises.Items
      listaPaises.Add(p)
  Next

  'Aplicando un ordenamiento descendente
  Dim paises = From p In listaPaises
               Select p
               Order By p.ToString Descending

  'Limpiar la lista para mostrar los valores ordenados
  lstPaises.Items.Clear()
  For Each p In paises
          lstPaises.Items.Add(p)
  Next
End Sub

 Private Sub btnTotal_Click(sender As Object, e As EventArgs) Handles
btnTotal.Click
    MessageBox.Show("El total de paises registrados: " & lstPaises.
Items.Count, "VB")
 End Sub
End Class
```

2.9.2 Control ListView

Representa el objeto de la clase ListView, el cual muestra información matricial, tal como se muestra en Excel o en una tabla de Word.

	Cuota #	Crédito #	Valor cuota	Interés	Préstamo Vivo	Mora	Vence	Saldo cuota
✔	001	C-54	1,000.00	230.77	12,000.00	0.00	06/06/2014	1,000.00
✔	002	C-54	1,000.00	230.77	11,000.00	0.00	13/06/2014	1,000.00
	003	C-54	1,000.00	230.77	10,000.00	0.00	20/06/2014	1,000.00
	004	C-54	1,000.00	230.77	9,000.00	0.00	27/06/2014	1,000.00
	005	C-54	1,000.00	230.77	8,000.00	0.00	04/07/2014	1,000.00
	006	C-54	1,000.00	230.77	7,000.00	0.00	11/07/2014	1,000.00
	007	C-54	1,000.00	230.77	6,000.00	0.00	18/07/2014	1,000.00
	008	C-54	1,000.00	230.77	5,000.00	0.00	25/07/2014	1,000.00

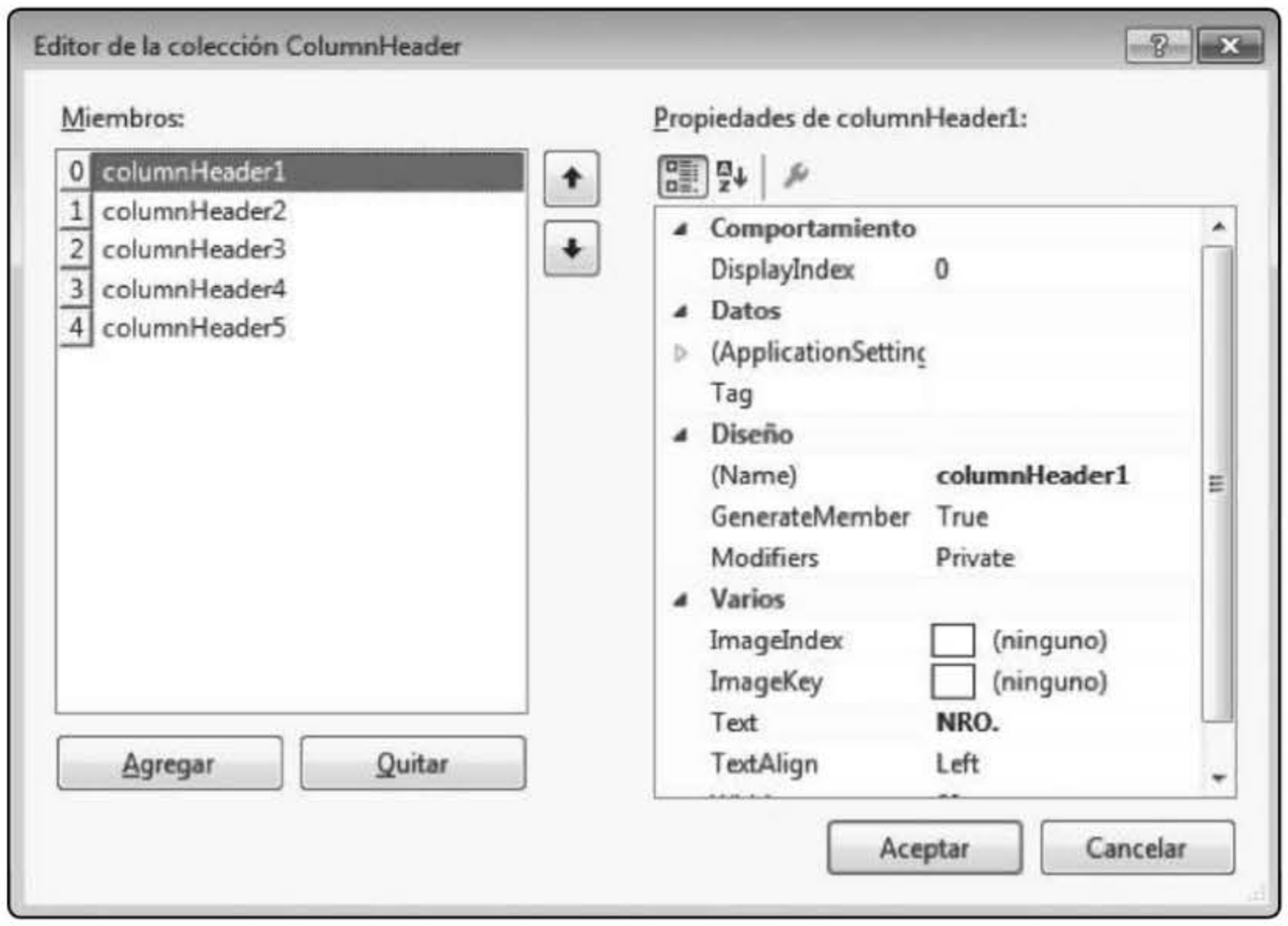
ListView

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ListView. Por ejemplo, para un registro de compra de productos podría llamarse lvRegistro.
GridLines	Asigna una rejilla al control ListView.
View	Asigna una determinada vista al objeto ListView. Para todos nuestros casos usaremos la vista Details (Detalles).
Columns	Permite asignar columnas al objeto ListView, de tal forma que se asigna una cabecera adecuada para el usuario.

A continuación, un caso de cómo implementar el siguiente ListView, el cual permitirá presentar el número, descripción, precio, cantidad y subtotal de una venta de productos.

NRO.	PRODUCTO	PRECIO	CANTIDAD	SUBTOTAL ($)

- Seleccione el objeto ListView desde el formulario.
- Modifique la propiedad **View > Details**.
- Modifique la propiedad **GridLines > True**.
- Modifique la propiedad **Columns** > y agregue los textos mostrados en la cabecera



Para agregar las cabeceras, primero debe crearse cinco columnas, para esto haga clic cinco veces sobre el botón **Agregar**; luego, seleccione uno a uno los miembros (columnHeader1) y en la propiedad **Text** mostrada en la misma ventana, modifique la cabecera. Cuando termine con todas las columnas haga clic en el botón **Aceptar**.

Podemos implementar la misma aplicación mediante el siguiente código:

```
Public Class frmListView
    Private Sub frmListView_Load(…) Handles MyBase.Load
        lvFactura.View = View.Details
        lvFactura.GridLines = True
        lvFactura.Columns.Add("NRO", 60)
        lvFactura.Columns.Add("PRODUCTO", 150)
        lvFactura.Columns.Add("PRECIO", 100)
        lvFactura.Columns.Add("CANTIDAD", 100)
        lvFactura.Columns.Add("SUBTOTAL($)", 120)
    End Sub
End Class
```

Para que el control ListView muestre una rejilla, use la sentencia **lvFactura.GridLines=True**; para visualizar en forma de columnas se usa la sentencia **lvFactura.View=View.Details**; y, finalmente, para agregar los encabezados se usa la sentencia **lvFactura.Columns.Add(«Titulo», Ancho de la Columna)**.

2.10 Controles contenedores de otros controles

Estos controles permitirán mostrar información resultante desde un proceso de la aplicación. Esta sección de muestra de información dependerá del programador, pues él decidirá en qué control desea mostrar la información proveniente de la lógica de la aplicación.

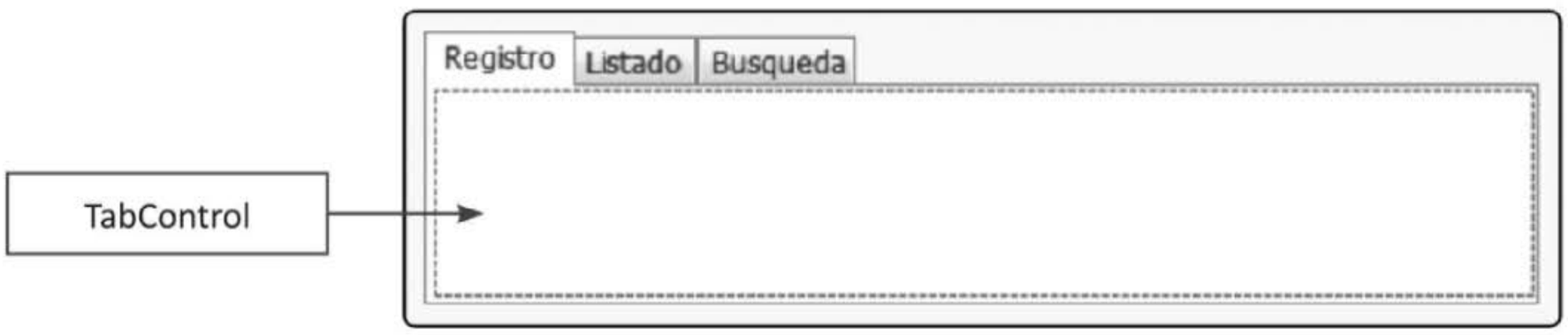
2.10.1 Control GroupBox

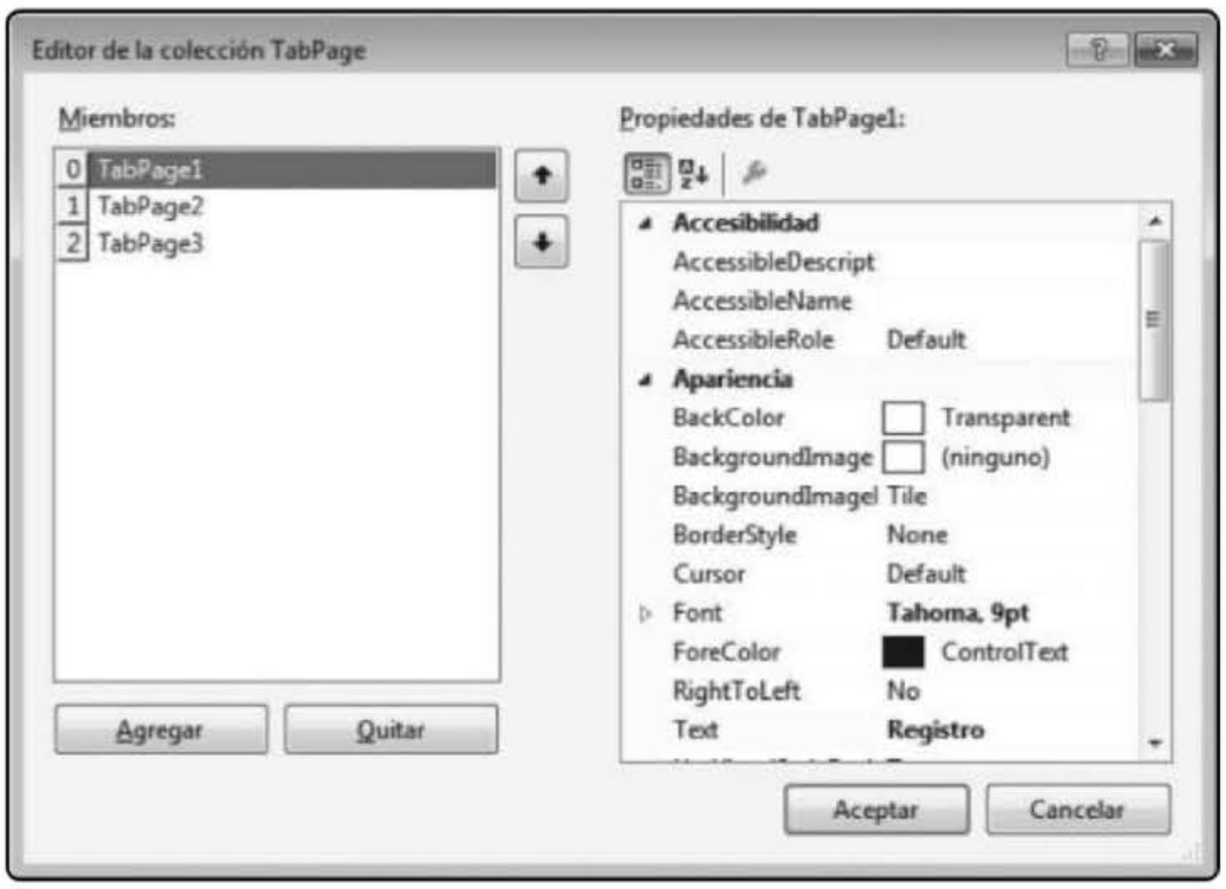
Representa al objeto de la clase GroupBox, el cual representa a un contenedor de controles.

Propiedad	Descripción
Text	Permite asignar un título al control GroupBox.
Font	Define la fuente que pueden tener los controles contenidos dentro del control GroupBox.

2.10.2 Control TabControl

Representa al objeto de la clase TabControl, el cual contiene un conjunto de páginas llamados Tab o Fichas; cada ficha puede contener un conjunto de controles que al final todos están agregados al formulario y, por lo tanto, los controles colocados en las fichas deben tener diferentes nombres.



Propiedad	Descripción
TabPages	Representa el conjunto de fichas que puede contener un TabControl.  • Para añadir una ficha al control haga clic en el botón **Agregar**. • Para eliminar una ficha del control, seleccione un TabPage de la lista y haga clic en el botón **Quitar**. • Para modificar el título de cada ficha, seleccione un TabPage de la lista y cambie el nombre desde la propiedad **Text**.

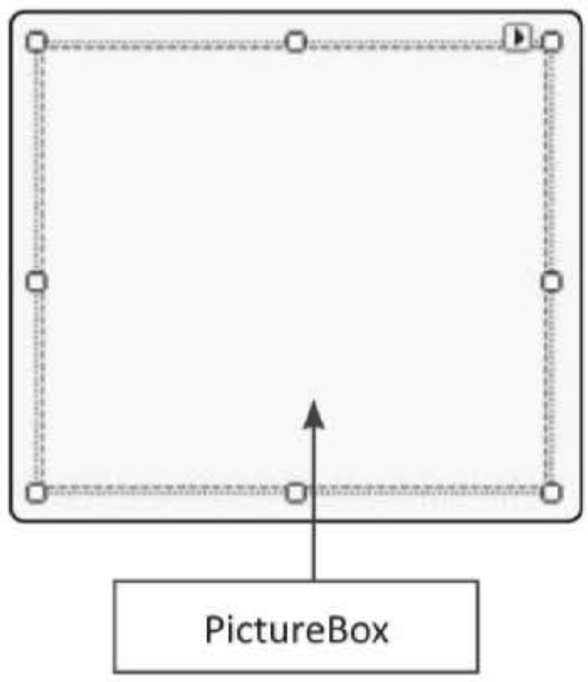
Propiedad	Descripción
Alignment	Permite alinear las fichas en uno de los extremos del TabControl: • Top: Las fichas se muestran en la parte superior. • Bottom: Las fichas se muestran en la parte inferior. • Left: Las fichas se muestra en la parte izquierda del TabControl. • Right: Las fichas se muestran en la parte derecha del TabControl.

2.11 Controles de manejo de imágenes

El control más resaltado de esta categoría es PictureBox, el cual permite mostrar una imagen en un marco; a partir de aquí se puede crear muchas aplicaciones que usen una imagen como elemento del proceso. Por ejemplo, mostrar un banner, la imagen de un producto, etc.

2.11.1 Control PictureBox

Representa al objeto de la clase PictureBox, el cual permite mostrar una lista de valores al usuario. Se usa principalmente para listar elementos seleccionados, sus propiedades son muy parecidas al control ComboBox.



Si se desea asignar una imagen seleccionada desde una ventana de dialogo, se usa el siguiente código:

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto PictureBox. Por ejemplo, para una imagen de los empleados puede llamarse pbEmpleado.
BorderStyle	Permite asignar un marco al control, ya que en el tiempo de ejecución no se muestra marco alguno.
Image	Permite asignar una imagen al control, seleccionando tres puntos **> Recurso local > Importar.**
SizeMode	Permite especificar cómo la imagen se adaptará al contenido del control PictureBox: • Normal: La imagen se muestra al tamaño real en el PictureBox. • StretchImage: La imagen se autoajusta al tamaño del PictureBox. • AutoSize: La imagen se ajusta al marco del control PictureBox. • CenterImage: La imagen se muestra al centro del PictureBox. • Zoom: La imagen se autoajusta al tamaño del PictureBox pero no pierde la proporción en alto y ancho de la imagen.

Para asignar una imagen seleccionada desde una ventana de dialogo, se usa el siguiente código:

```
Dim op As New OpenFileDialog
op.Filter = "imagenes|*.jpg"
If op.ShowDialog = DialogResult.OK Then
   pbFoto.Image = Image.FromFile(op.FileName)
End If
```

El objeto de la clase OpenFileDialog permite mostrar el cuadro de diálogo de apertura de archivo; para esto se debe especificar el tipo de archivo con la propiedad Filter donde *.jpg especifica que solo se pueden abrir imágenes de tipo jpg. Luego, se asigna la imagen seleccionada desde el cuadro de dialogo mediante la sentencia **Image.FromFile(op.FileName)**, el cual obtiene la dirección exacta de la imagen.

2.12 Controles de manejo de tiempos

Estos controles permitirán controlar las fechas dentro de una aplicación mostrando controles que permitirán seleccionar una fecha a partir de un calendario ya establecido sin necesidad de ingresar los valores; esto podrá ayudar, por ejemplo, a mostrar un reporte a partir de una fecha indicada.

2.12.1 Control DateTimePicker

Representa al objeto de la clase DateTimePicker, el cual permite mostrar una lista desplegable con una fecha en forma de texto y un calendario que muestra la selección del día actual. Este control permite al usuario seleccionar una fecha y lo devuelve validado.

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto DateTimePicker. Por ejemplo, para seleccionar un fecha de registro podría llamarse dtFechaRegistro.
Format	Define la forma de mostrar la fecha en el control: • Long: Muestra la fecha en formato largo; es decir, se visualiza el día en texto, mes en letra, día y año. • Short: Muestra la fecha en formato corto; es decir, se visualiza día, mes y año en números y separados por una barra. • Time: Muestra la hora actual del sistema.

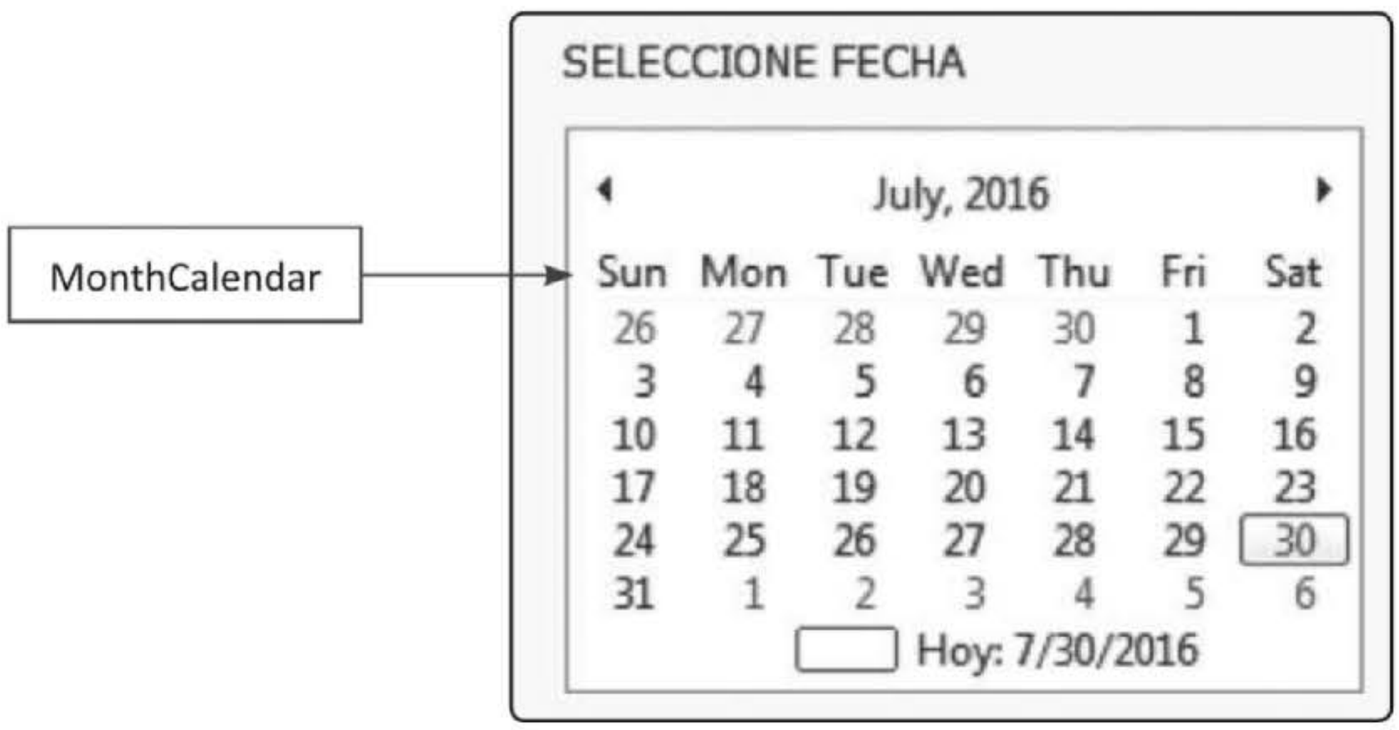
Ahora, una aplicación básica de selección de una fecha a partir de un control DateTimePicker de forma que, al seleccionar una fecha, se muestre la selección en un control Label.

El código para mostrar la información se muestra a continuación:

```
Public Class frmdateTimePicker
    Private Sub dtFecha_ValueChanged(…) Handles dtFecha.ValueChanged
        lblFecha.Text = dtFecha.Value
    End Sub
End Class
```

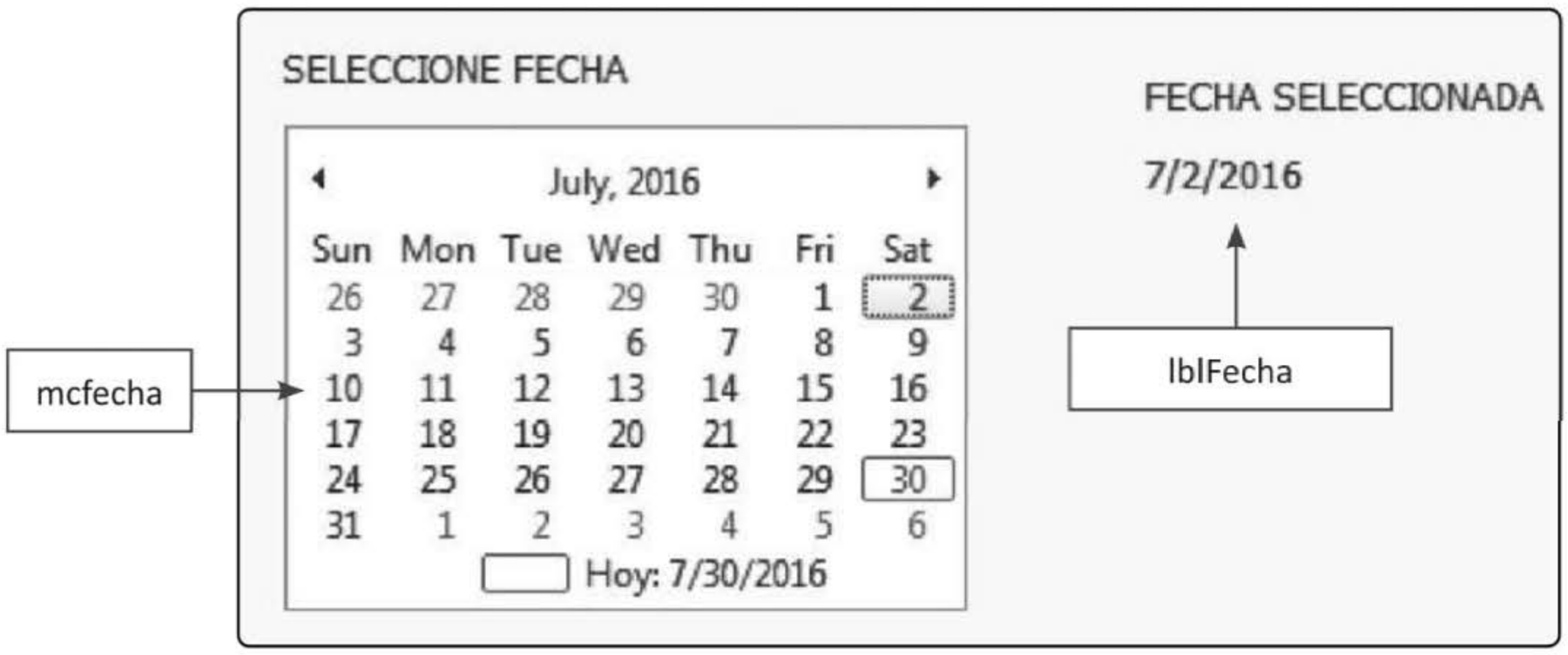
2.12.2 Control MonthCalendar

Representa al objeto de la clase MonthCalendar, el cual permite seleccionar una fecha a partir de la presentación del mes actual. A diferencia del control DateTimePicker, el mes no se contrae y siempre se muestra.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto MonthCalendar. Por ejemplo, se le podría llamar mcFecha.

A continuación, una aplicación básica de selección de una fecha a partir de un control MonthCalendar, de forma que, al seleccionar una fecha, se muestre la selección en un control Label.



El código para mostrar la información se muestra a continuación:

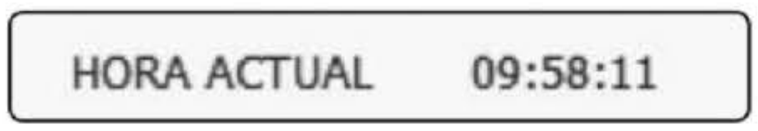
```
Public Class frmMonthCalendar
    Private Sub mcFecha_DateChanged(…) Handles mcFecha.DateChanged
        lblFecha.Text = mcFecha.SelectionStart
    End Sub
End Class
```

2.12.3 Control Timer

Representa al objeto de la clase Timer, trabaja como temporalizador de procesos en la cual se puede realizar diferentes actividades en un intervalo de tiempo especificado en milisegundos.

Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto Timer. Se podría asignar el nombre tHora.
Interval	Define el intervalo en milisegundos; por defecto se muestra 100 pero debemos modificar a 1000, ya que esto representa a 1 segundo.

A continuación, el código que permite mostrar la hora actual como un reloj digital:



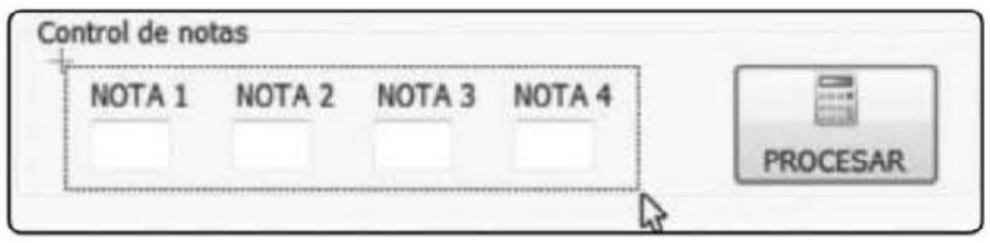

```
Public Class frmTimer
    Private Sub tHora_Tick(sender As Object, e As EventArgs) Handles
tHora.Tick
        lblHora.Text = TimeString
    End Sub

    Private Sub frmTimer_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        tHora.Start()
    End Sub
End Class
```

2.13 Operaciones básicas con los formularios

Ahora, algunas operaciones básicas que se realizan cuando se agregan controles al Windows Forms.

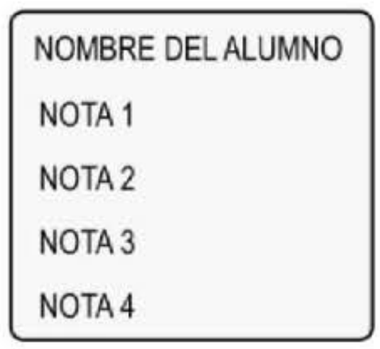
2.13.1 Seleccionar los controles en el formulario

La selección de uno o más controles permite administrar dichos objetos como copiar, mover, eliminar, alinear o modificar alguna propiedad en común. Se presentan los siguientes casos:

Un solo control	Haga click directamente en el control
Dos o más controles	Haga <Ctrl + Clic> en los controles
Dos o más controles adyacentes	Arrastre un marco de selección sobre los controles 
Todos los controles	Haga clic en <Ctrl + a> Ir a **Editar > Seleccionar todo**

2.13.2 Alinear los controles en el formulario

Visual cuenta con una barra de diseño, la cual presenta un conjunto de botones de alineación, como se muestra a continuación:

Tipo	Descripción
Alinear lados izquierdos	Permite alinear los controles al lado izquierdo. 
Alinear centros	Permite alinear los controles al centro tomando como referencia uno de ellos.

Tipo	Descripción
Alinear lados derechos	Permite alinear los controles al lado derecho. NOMBRE DEL ALUMNO NOTA 1 NOTA 2 NOTA 3 NOTA 4
Alinear lados	Permite alinear los controles en los datos superiores, puntos medios y lados inferiores. NOMBRE DEL ALUMNO NOTA 1 NOTA 2 NOTA 3 NOTA 4

2.13.3 Igualar los espacios entre los controles en el formulario

Igualar los espacios entre los controles conlleva a una buena distribución de los controles en el formulario.

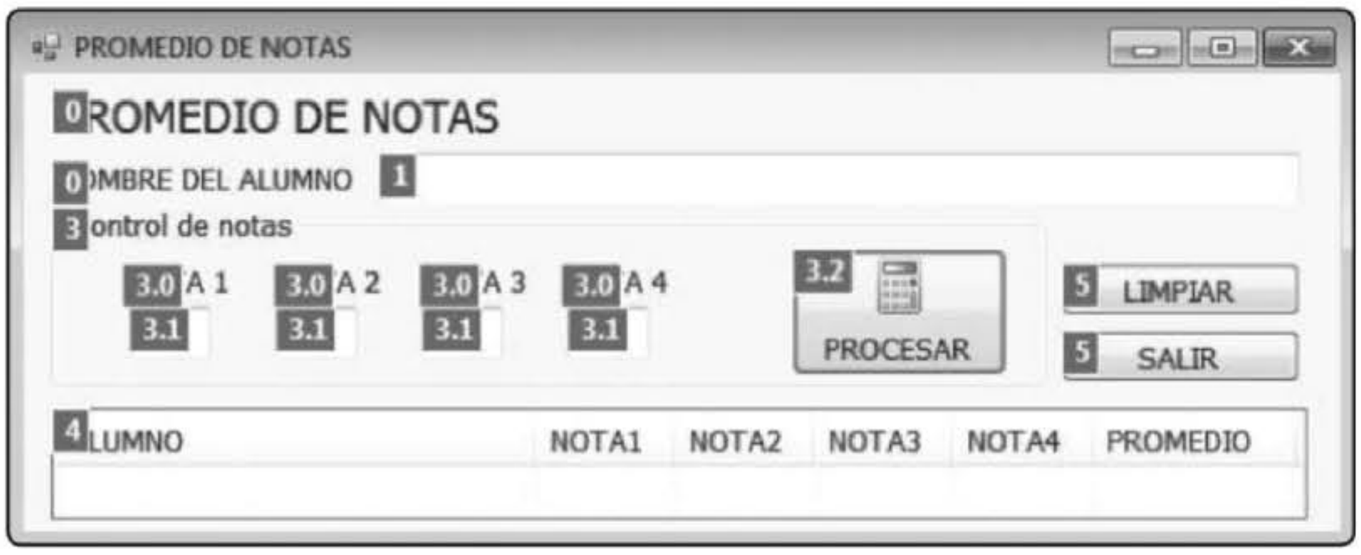
Tipo	Descripción
Igualar espaciado horizontal	Permite igualar el espacio entre los controles de forma horizontal, tal como se muestra en la siguiente imagen: Distribución manual: NOMBRE DEL ALUMNO NOTA 1 NOTA 2 NOTA 3 NOTA 4 Distribución espaciada de forma horizontal: NOMBRE DEL ALUMNO NOTA 1 NOTA 2 NOTA 3 NOTA 4
Igual espaciado vertical	Permite igualar el espacio entre los controles de forma vertical, tal como se muestra en la siguiente imagen: NOMBRE DEL ALUMNO NOTA 1 NOTA 2 NOTA 3 NOTA 4

2.13.4 Orden de tabulación de los controles en el formulario

Cuando se ingresa valores en los controles, habitualmente se usa la tecla Tab para saltar al siguiente control; dicho salto tiene un orden que se genera al momento de colocar los controles en el formulario. Este orden suele perderse cuando se realizan copias de elementos en el mismo formulario.

Para poder definir nuevamente un orden se pueden realizar manualmente mediante los siguientes pasos:

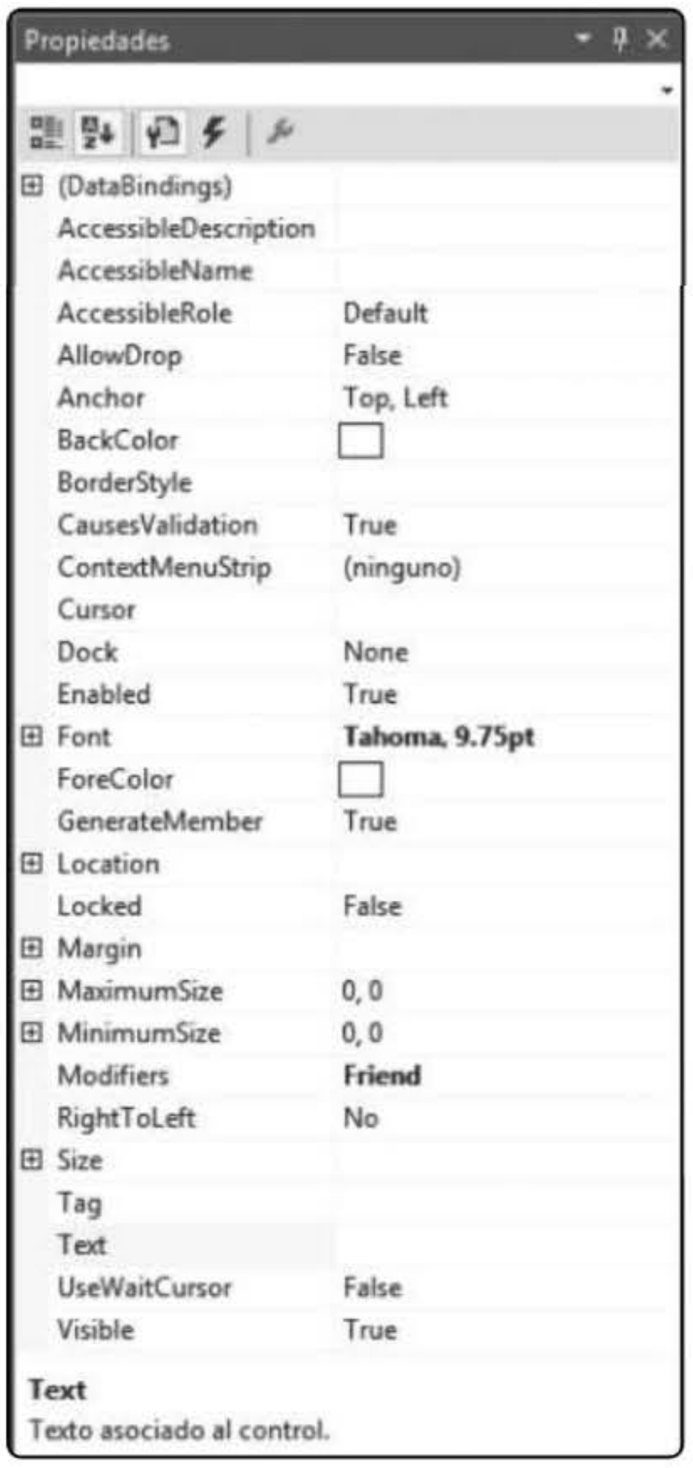
- Ubíquese en un formulario con controles diseñados
- Desde el menú **Ver > Orden de tabulación**



- Para dar un orden, será cuestión de realizar un clic en los controles dependiendo del orden que desea tabular.
- Para finalizar el orden de tabulación nuevamente seleccione **Ver > Orden de tabulación**
- Para probar el orden debemos ejecutar la aplicación con F5.

2.13.5 Establecer propiedades comunes de los controles

Existen algunos controles que tienen propiedades comunes; por ejemplo, un control Label y un TextBox tienen en común las siguientes propiedades:



Como se puede ver, el panel de propiedades solo lista las que tienen en común ambos controles y que, al seleccionar una de ellas, será aplicada a todos los controles seleccionados. No se olvide que para seleccionar uno o más controles al mismo tiempo puede usar la tecla <Ctrl> sobre los controles.

2.14 Eventos de los controles

Un evento es considerado como una ocurrencia del control; todos los controles sin excepción poseen eventos y estos pueden ser controlados mediante un código que el programador puede implementar. El evento más conocido entre los controles es el clic, el cual ejecuta una serie de sentencias con hacer clic del mouse en un determinado control.

2.14.1 Visualizar el código según el evento predeterminado de un control

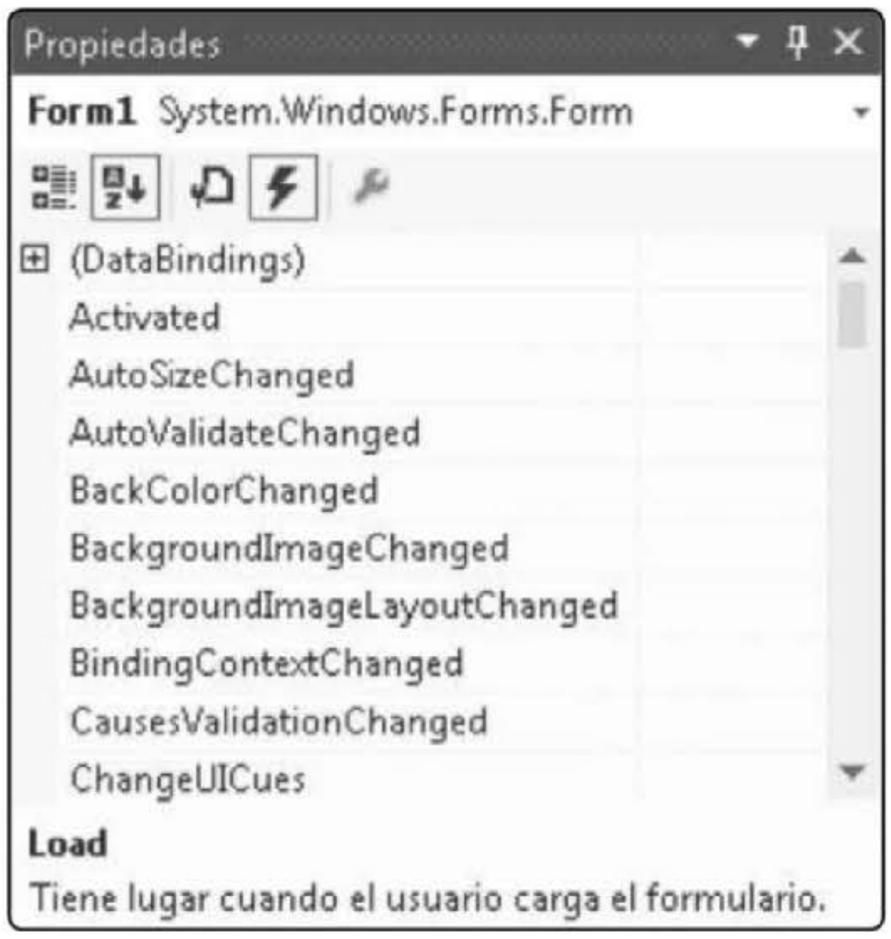
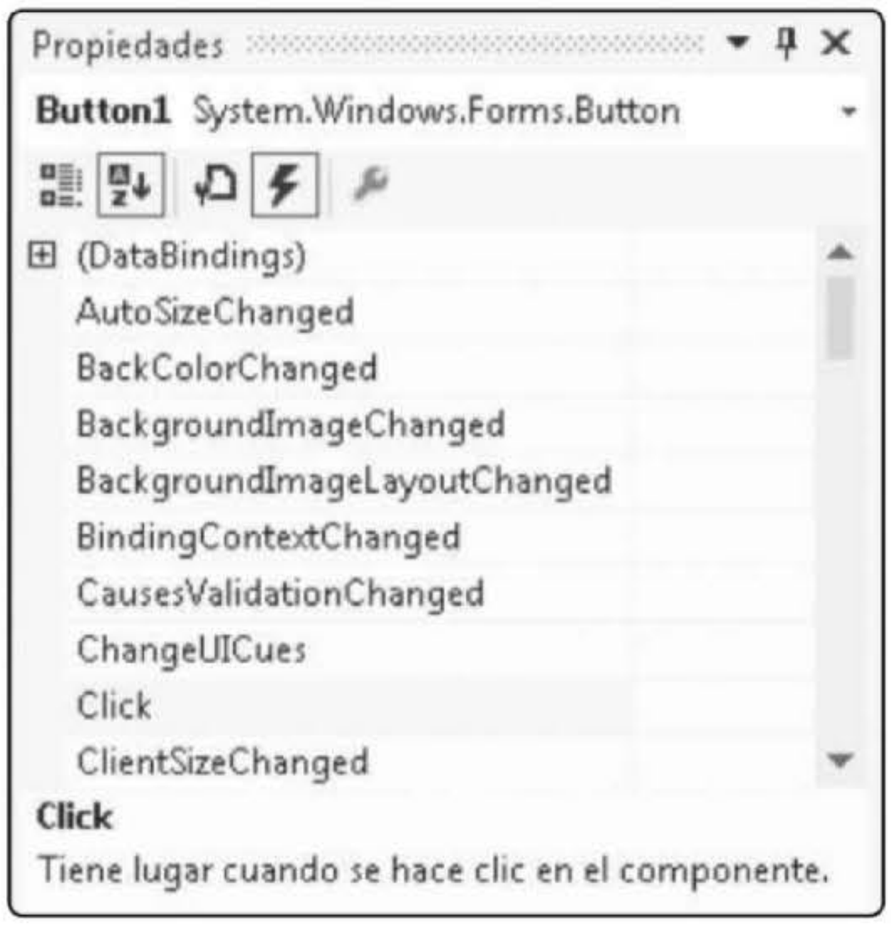
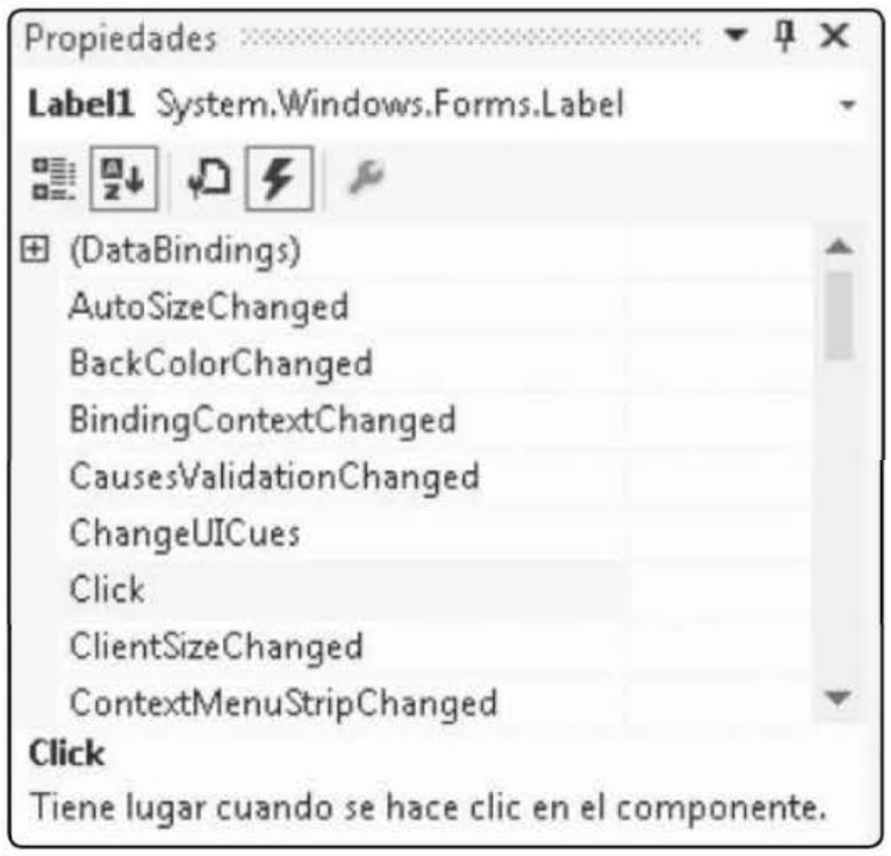
Todo control dentro del formulario tiene un evento predeterminado que se puede visualizar haciendo doble clic sobre dicho control.

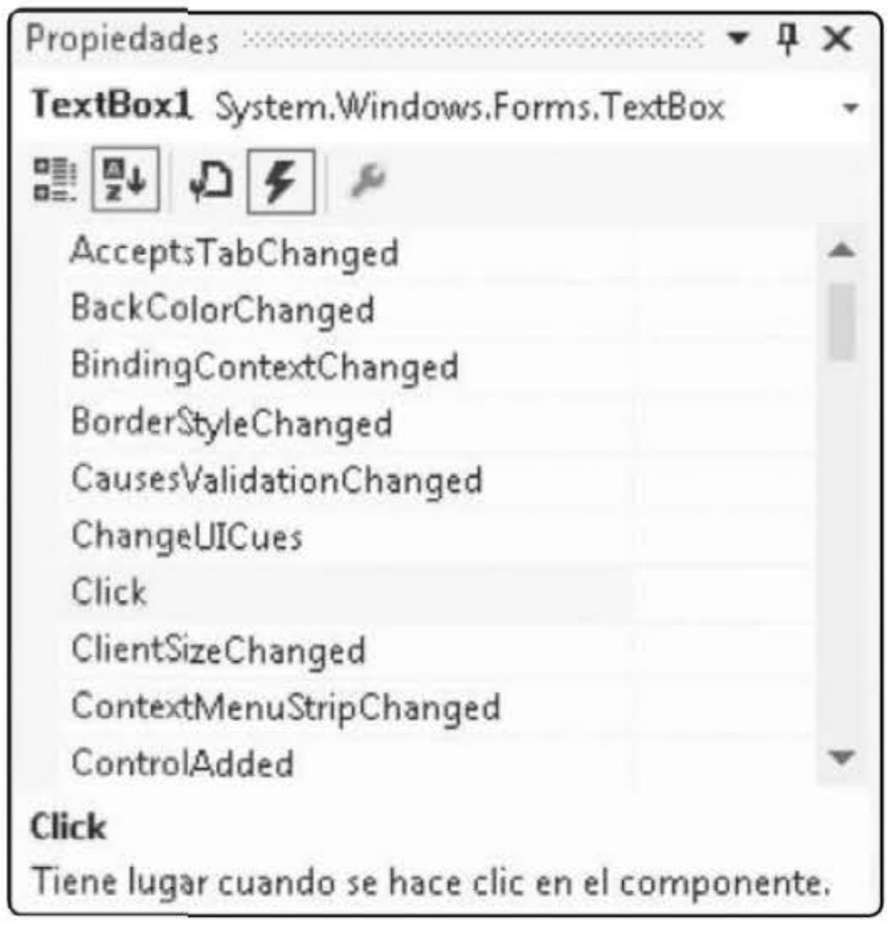
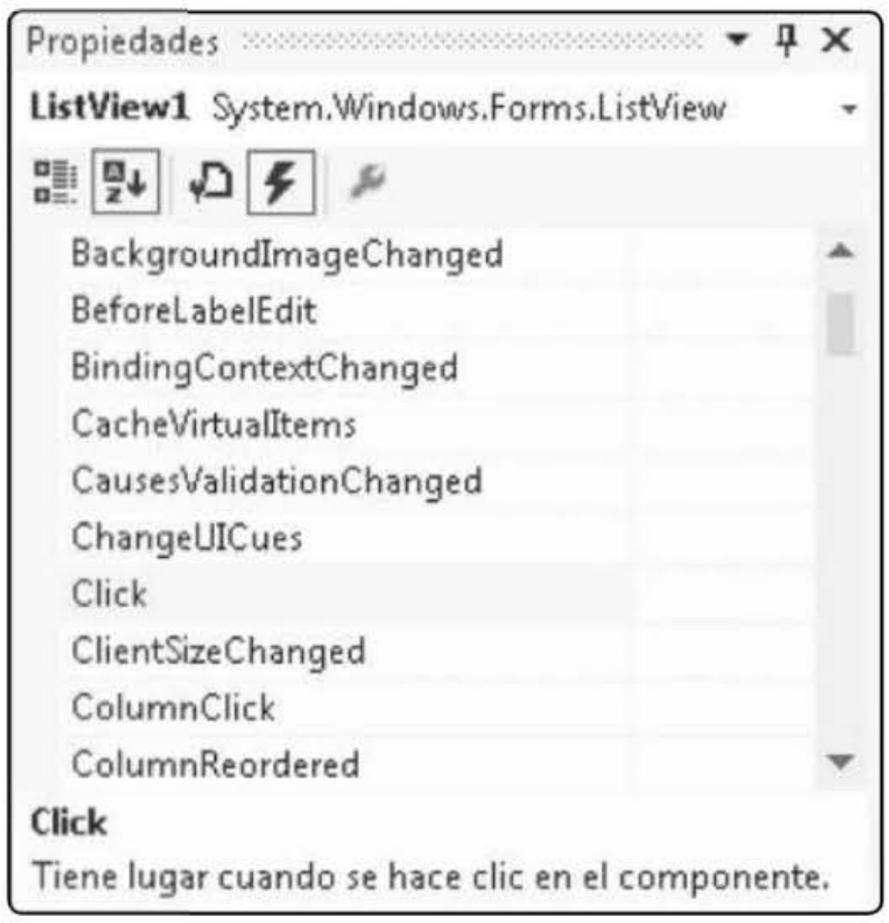
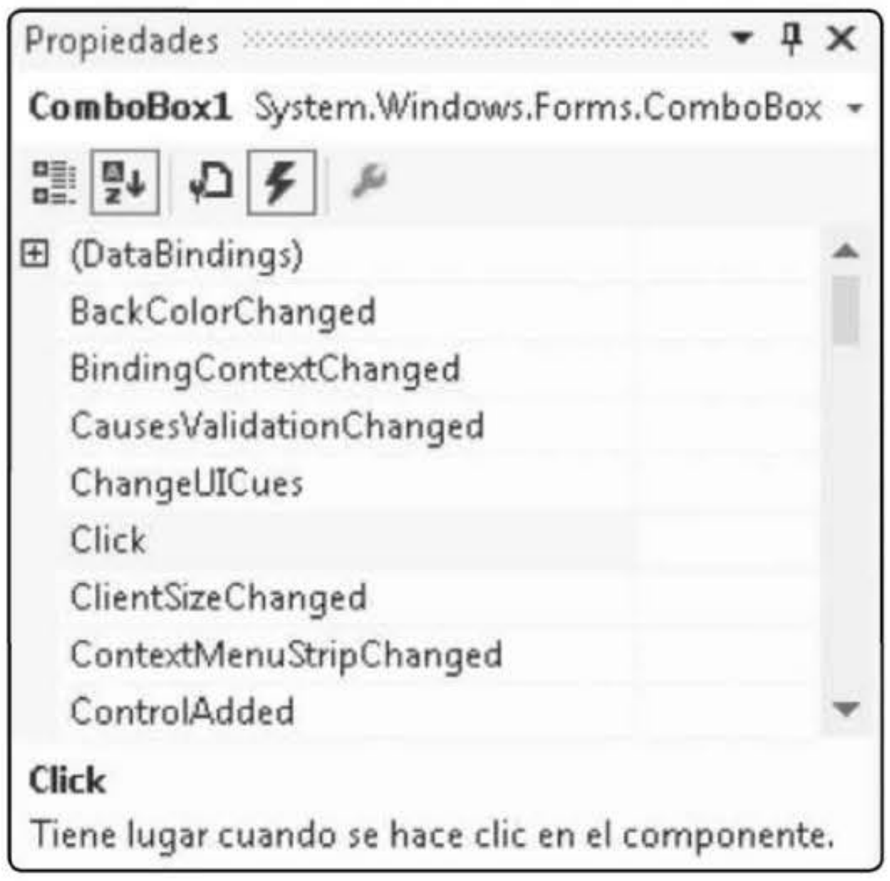
Control	Evento predeterminado
Form	`Private Sub frmPromedio_Load()` `End Sub`
Button	`Private Sub btnProcesar_Click()` `End Sub`
Label	`Private Sub Label6_Click()` `End Sub`
TextBox	`Private Sub txtAlumno_TextChanged()` `End Sub`
ListView	`Private Sub ListView1_SelectedIndexChanged()` `End Sub`
ComboBox	`Private Sub ComboBox1_SelectedIndexChanged()` `End Sub`

2.14.2 Seleccionar un evento para un determinado control

Realizar los siguientes pasos:

- Seleccione el control.
- Desde la ventana de propiedades seleccione eventos .

Form	
Button	
Label	

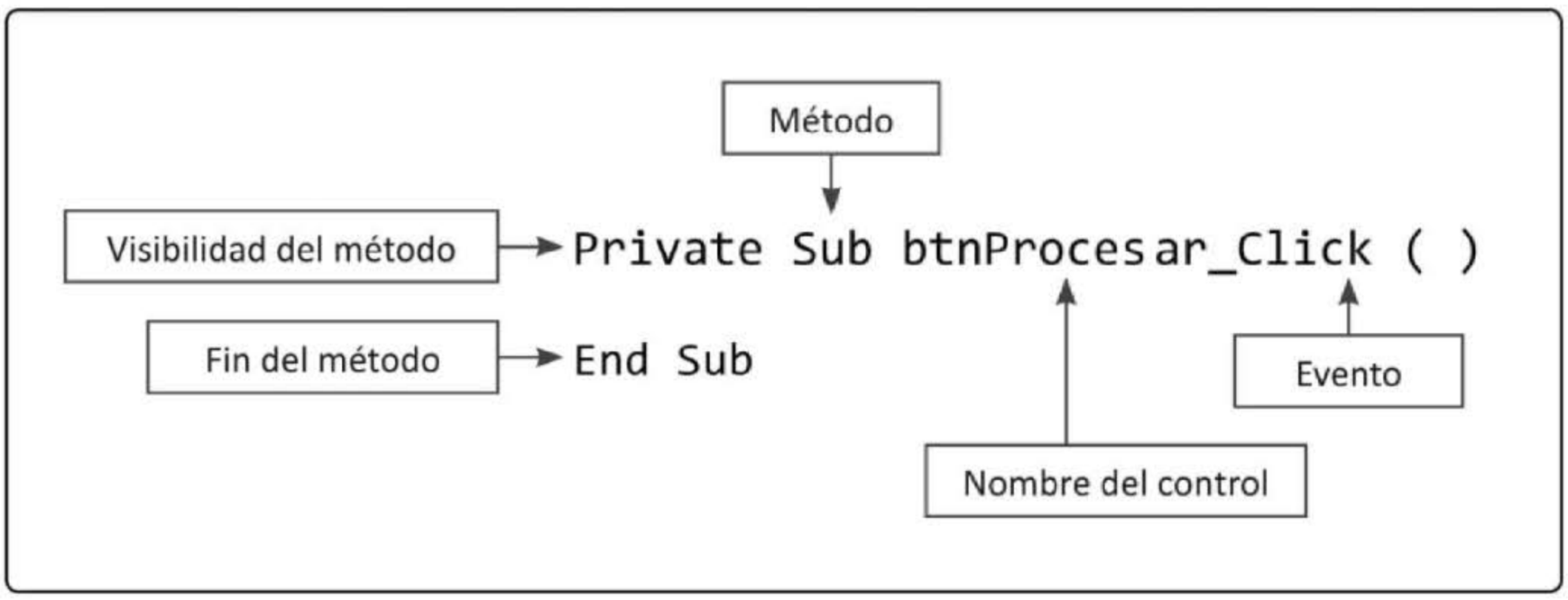
TextBox	
ListView	
ComboBox	

Para colocar código en uno de los eventos, primero debe asegurarse de conocer dicho evento, luego ubicarlo en la sección de eventos del control y para acceder a su ventana de código haga doble clic sobre un determinado evento.

Si por algún motivo se equivoca al seleccionar un evento, su eliminación se realizará desde la ventana de código, seleccionando todo el método y haciendo clic en la tecla Suprimir.

2.14.3 Partes del método generado por el evento del control

Aquí, los elementos que componen un método generado a partir de la selección de un evento.



Todos los eventos generan un método de visibilidad: Privado (Private), puesto que este método solo será accesible dentro del formulario, ya que todo formulario es considerado como una clase. La cláusula Sub indica que el método no devuelve valor alguno a quien lo invoque. El término **btnProcesar_Click()** representa al nombre asignado al control, además muestra el evento seleccionado. Y la cláusula End Sub indica el fin del método.

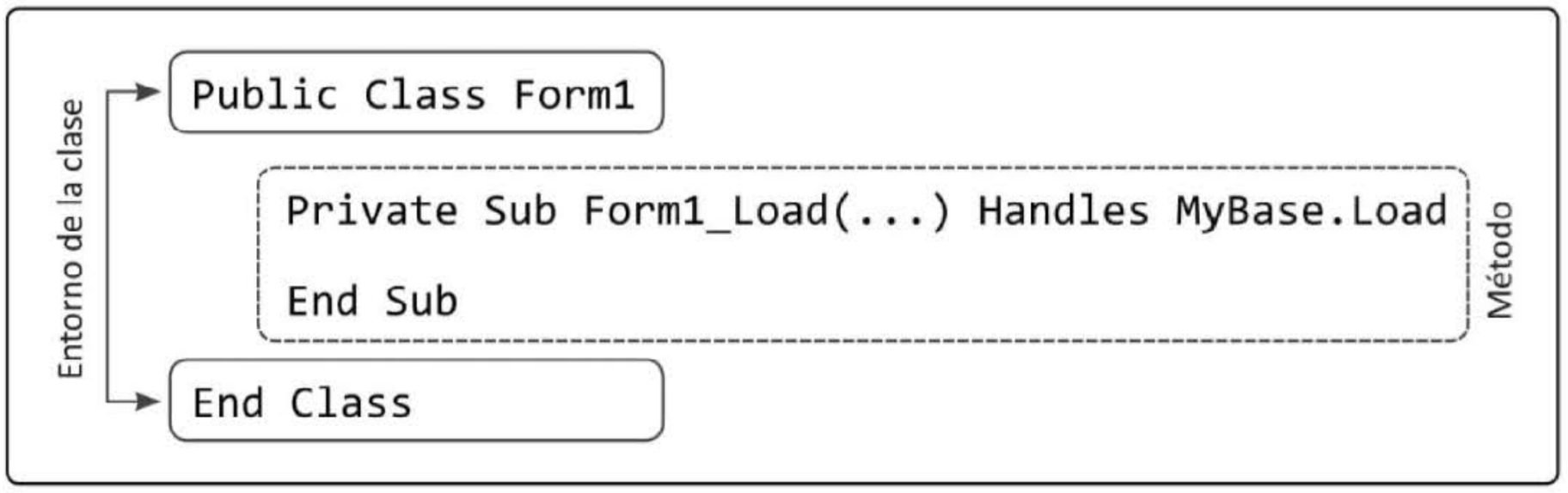
2.15 Editor de código

Todo código que se implementa en una aplicación Windows Forms se debe desarrollar en métodos que son provenientes de los eventos seleccionados desde un determinado control.

Para tener acceso al editor de código se puede realizar las siguientes actividades:

- Haga doble clic sobre el control.
- Presione la tecla <F7> desde el formulario.
- Seleccione un control, desde la ventana de propiedades seleccione **Eventos** y haga doble clic sobre un evento.

Inicialmente, el editor de código se muestra de la siguiente manera:



Public Class Form1 representa a la clase principal de la aplicación, dentro de él se podrá implementar todos los métodos posibles que necesite nuestra aplicación siguiendo el patrón según Visual Basic. Adicionalmente se puede mencionar que es posible eliminar los métodos sin dañar la estructura original de la aplicación.

2.16 Desarrollo de aplicaciones tipo Windows Forms

Ahora implementaremos aplicaciones que, usando elementos visuales, nos permitan desarrollar entornos de trabajo profesionales haciendo uso de los controles visuales de tipo Windows Forms.

2.16.1 Crear un nuevo proyecto de tipo Windows Forms

- Siga la ruta **Archivo > Nuevo proyecto.**
- Seleccione la plantilla **Visual Basic > Windows > Aplicación de Windows Forms.**

- Coloque un nombre según la notación de los formularios y seleccione **Aceptar**.

2.16.2 Guardar el proyecto de tipo Windows Forms

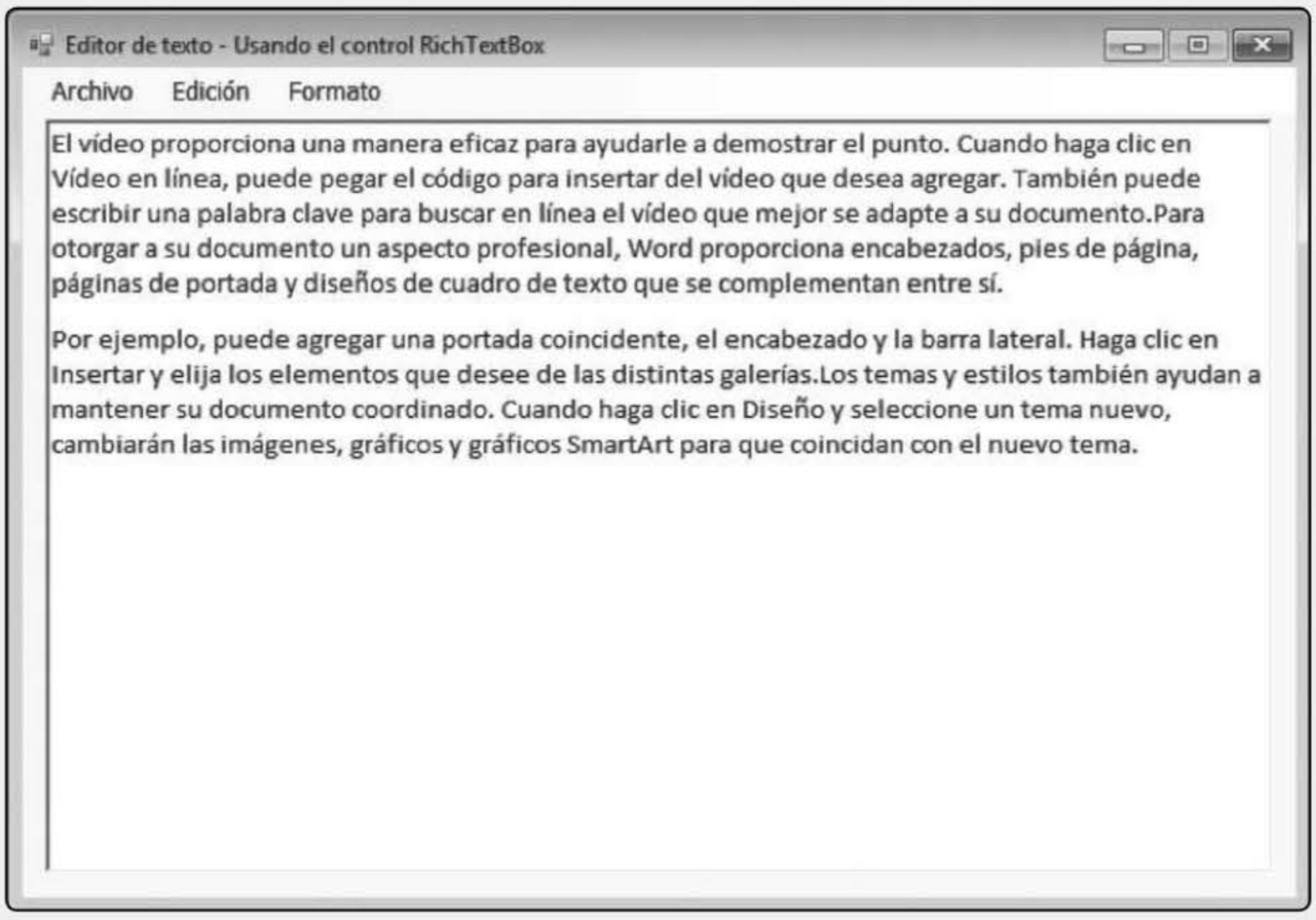
Realizar los siguientes pasos:

- Siga la ruta **Archivo > Guardar todo** (<Ctrl+Shift+S>).
- Asigne un nombre y una ubicación.

- Haga clic en el botón **Guardar**.

Caso desarrollado 1 Manejo del control RichTextBox

Aplicación que permita implementar un editor de texto básico usando el control RichTextBox.



2.16.2 Guardar el proyecto de tipo Windows Forms

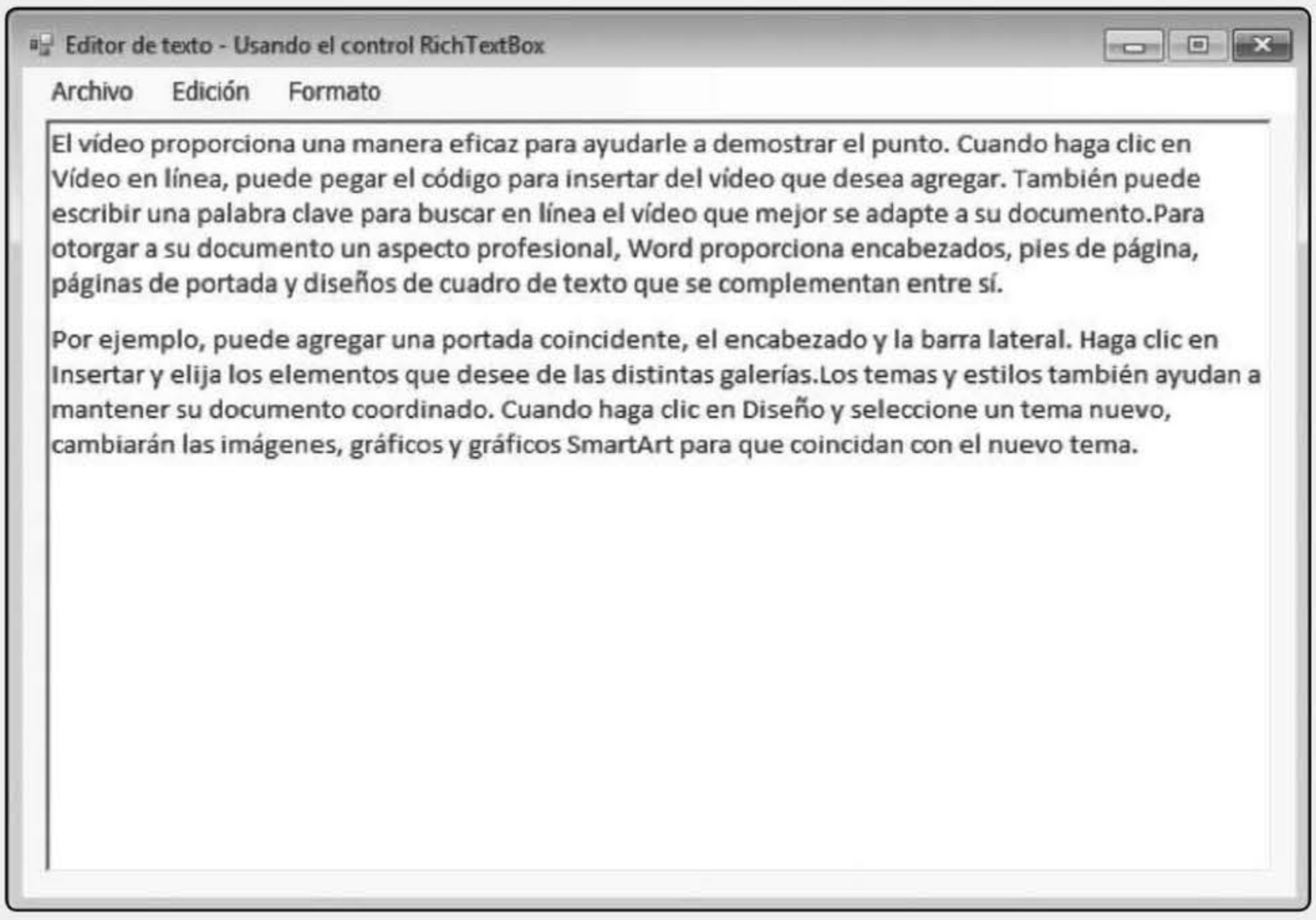
Realizar los siguientes pasos:

- Siga la ruta **Archivo > Guardar todo** (<Ctrl+Shift+S>).
- Asigne un nombre y una ubicación.

- Haga clic en el botón **Guardar**.

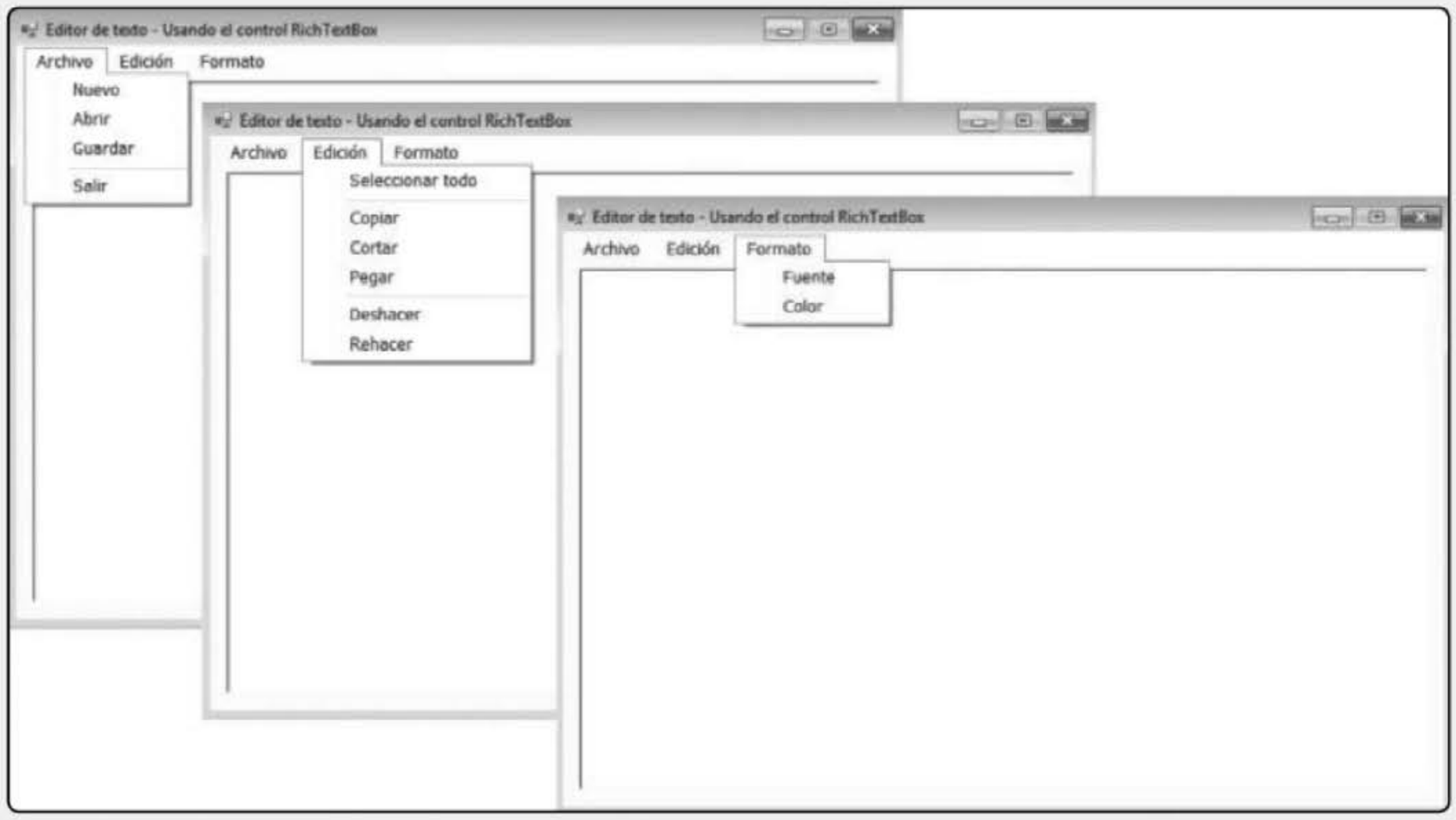
Caso desarrollado 1 Manejo del control RichTextBox

Aplicación que permita implementar un editor de texto básico usando el control RichTextBox.



Tener en cuenta los siguientes aspectos:

- Implemente un menú de opciones utilizando el control **MenuStrip**, de tal forma que se muestre de la siguiente manera:



- La ficha **Archivo** debe contar con las siguientes opciones de menú:

Nuevo	Permite limpiar el contenido del control RichTextBox.
Abrir	Permite abrir un documento de tipo RTF.
Guardar	Permite un documento de RTF.
Salir	Permite salir de la aplicación.

- La ficha **Edición** debe contar con las siguientes opciones de menú:

Seleccionar todo	Permite seleccionar todo el contenido del control RichTextBox.
Copiar	Permite copiar un bloque de texto previamente seleccionado.
Cortar	Permite cortar o mover un bloque de texto previamente seleccionado.
Pegar	Permite pegar un bloque de texto previamente copiado o cortado.
Deshacer	Permite deshacer el último cambio realizado.
Rehacer	Permite rehacer el último cambio realizado.

- La ficha **Formato** debe contar con las siguientes opciones de menú

Fuente	Permite modificar tipo de letra, tamaño y estilo de un determinado texto seleccionado.
Color	Permite modificar el color de un determinado texto seleccionado.

Solución:

- Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) Text	frmEditor Editor de texto
RichTextBox1	(Name)	rtbEditor

- Agregar los siguientes controles al formulario:

MenuStrip	El cual tiene como misión implementar el menú de opciones solicitadas por el caso. Aquí no será necesario asignar nombres a las opciones pues, automáticamente, se genera un nombre de acuerdo al texto establecido en cada opción, que por ahora será suficiente para nuestra aplicación.
ColorDialog	Permitirá mostrar el cuadro de dialogo de color, el cual permitirá seleccionar y asignar un color a un texto seleccionado. Asígnele el nombre cdColor.
FontDialog	Permitirá mostrar el cuadro de dialogo de fuente, el cual permitirá seleccionar y aplicar un tipo de letra, tamaño y un estilo a un texto seleccionado. Asígnele el nombre fdFuente.
OpenFileDialog	Permitirá mostrar el cuadro de dialogo de apertura de documento, el cual permitirá seleccionar un archivo de tipo RTF y mostrará toda su información en el control RichTextBox. Asígnele el nombre opArchivo.
SaveFileDialog	Permitirá mostrar el cuadro de diálogo de grabación de documento, para que se pueda asignar un nombre de archivo a un determinado texto colocado en el control RichTextBox. Asígnele el nombre svArchivo.

- El script de la solución es:

```
Public Class frmEditor
    Private Sub SeleccionarTodoToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                Handles SeleccionarTodoToolStripMenuItem.Click
        rtbEditor.SelectAll()
    End Sub

    Private Sub CopiarToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles CopiarToolStripMenuItem.Click
        rtbEditor.Copy()
    End Sub

    Private Sub CortarToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles CortarToolStripMenuItem.Click
        rtbEditor.Cut()
    End Sub

    Private Sub PegarToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles PegarToolStripMenuItem.Click
        rtbEditor.Paste()
    End Sub

    Private Sub DeshacerToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles DeshacerToolStripMenuItem.Click
        rtbEditor.Undo()
    End Sub

    Private Sub RehacerToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles RehacerToolStripMenuItem.Click
        rtbEditor.Redo()
    End Sub

    Private Sub ColorToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles ColorToolStripMenuItem.Click
        cdColor.ShowDialog()
        rtbEditor.SelectionColor = cdColor.Color
    End Sub

    Private Sub FuenteToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles FuenteToolStripMenuItem.Click
        fdFuente.ShowDialog()
        rtbEditor.SelectionFont = fdFuente.Font
    End Sub

    Private Sub AbrirToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles AbrirToolStripMenuItem.Click
        opArchivo.ShowDialog()
        rtbEditor.LoadFile(opArchivo.FileName, RichTextBoxStreamType.RichText)
    End Sub

    Private Sub GuardarToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles GuardarToolStripMenuItem.Click
```

```
        svArchivo.Filter = "Archivo de Texto Enriquecido|*.rtf"
        svArchivo.ShowDialog()
        rtbEditor.SaveFile(svArchivo.FileName, RichTextBoxStreamType.RichText)
    End Sub

    Private Sub SalirToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles SalirToolStripMenuItem.Click
        Me.Close()
    End Sub

    Private Sub NuevoToolStripMenuItem_Click(sender As Object, e As EventArgs)
                                        Handles NuevoToolStripMenuItem.Click
        rtbEditor.Clear()
    End Sub
End Class
```

Caso desarrollado 2 Cálculo de sueldos

Aplicación que permita determinar el sueldo básico, descuento y neto que debe percibir un empleado de acuerdo a las horas de trabajo y a un costo por hora.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar el nombre del empleado, horas trabajadas y costo por hora.
- El sueldo básico se obtiene del producto entre las horas trabajadas y el costo por hora.

- El descuento se obtiene aplicando el 12% al sueldo básico.
- El neto se obtiene de la diferencia entre el sueldo básico y el descuento.
- Mostrar los montos con el formato de moneda.
- El botón **Aceptar** es el encargado de mostrar la información resultante y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Mostrar el resumen de pago en un control ListBox.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmSueldos btnAceptar btnCancelar PAGO DE EMPLEADOS
RichTextBox1	(Name)	rtbEditor
Label1	Text	PAGO DE EMPLEADOS
Label2	Text	NOMBRE DEL EMPLEADO
Label3	Text	HORAS TRABAJADAS
Label4	Text	COSTO POR HORA
TextBox1	(Name)	txtEmpleado
TextBox2	(Name)	txtHoras
TextBox3	(Name)	txtCosto
Button1	(Name) Text	btnCancelar CANCELAR
Button2	(Name) Text	btnAceptar ACEPTAR
GroupBox1	Text	Resumen de pago
List1	(Name)	lstR

El script de la solución es:

```
Public Class frmSueldos
    Private Sub btnAceptar_Click(sender As Object, e As EventArgs)
        'OBTENIENDO VALORES
        Dim empleado As String = txtEmpleado.Text
        Dim hTrabajadas As Integer = Integer.Parse(txtHoras.Text)
        Dim costoHora As Double = Double.Parse(txtCosto.Text)

        'REALIZANDO CALCULOS
        Dim basico As Double = hTrabajadas * costoHora
        Dim descuento As Double = basico * 0.12
        Dim neto As Double = basico - descuento

        'IMPRIMIR RESULTADOS
        lstR.Items.Clear()
        lstR.Items.Add("EMPLEADO :" + empleado)
        lstR.Items.Add("HORAS TRABAJADAS :" + hTrabajadas.ToString())
        lstR.Items.Add("COSTO HORA :" + costoHora.ToString("C"))
        lstR.Items.Add("----------------------------------------")
        lstR.Items.Add("BASICO :" + basico.ToString("C"))
        lstR.Items.Add("DESCUENTO :" + descuento.ToString("C"))
        lstR.Items.Add("NETO :" + neto.ToString("C"))
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtEmpleado.Clear()
        txtCosto.Clear()
        txtHoras.Clear()
        lstR.Items.Clear()
        txtEmpleado.Focus()
    End Sub
End Class
```

Caso desarrollado 3 Venta de helados

Aplicación que permita determinar el monto, descuento y total a pagar que debe realizar un cliente por la compra de helados de un mismo tipo.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar el nombre del cliente y la cantidad de helados a comprar.
- El precio del producto es un valor constante que asciende a $1.50.
- El monto se calcula del producto entre el precio y la cantidad comprada de helados.
- El descuento se obtiene aplicando el 7.5% del monto.
- El total a pagar se determina de la diferencia entre el monto y el descuento.
- Mostrar los montos con el formato de moneda.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmHelado btnProcesar btnSalir VENTA DE HELADOS
Label1	Text	VENTA DE HELADOS
Label2	Text	NOMBRE DEL CLIENTE
Label3	Text	CANTIDAD DE HELADOS
TextBox1	(Name)	txtCliente
TextBox2	(Name)	txtCantidad
Button1	(Name) Image ImageAlign Text TextAlign	btnProcesar (Seleccione una imagen) MiddleLeft PROCESAR MiddleRight
Button2	(Name) Text	btnSalir SALIR
Button3	(Name) Text	btnLimpiar LIMPIAR
GroupBox1	Text	Resumen de venta
Label4	Text	CANT.
Label5	Text	COSTO
Label6	Text	MONTO
Label7	Text	DESCUENTO

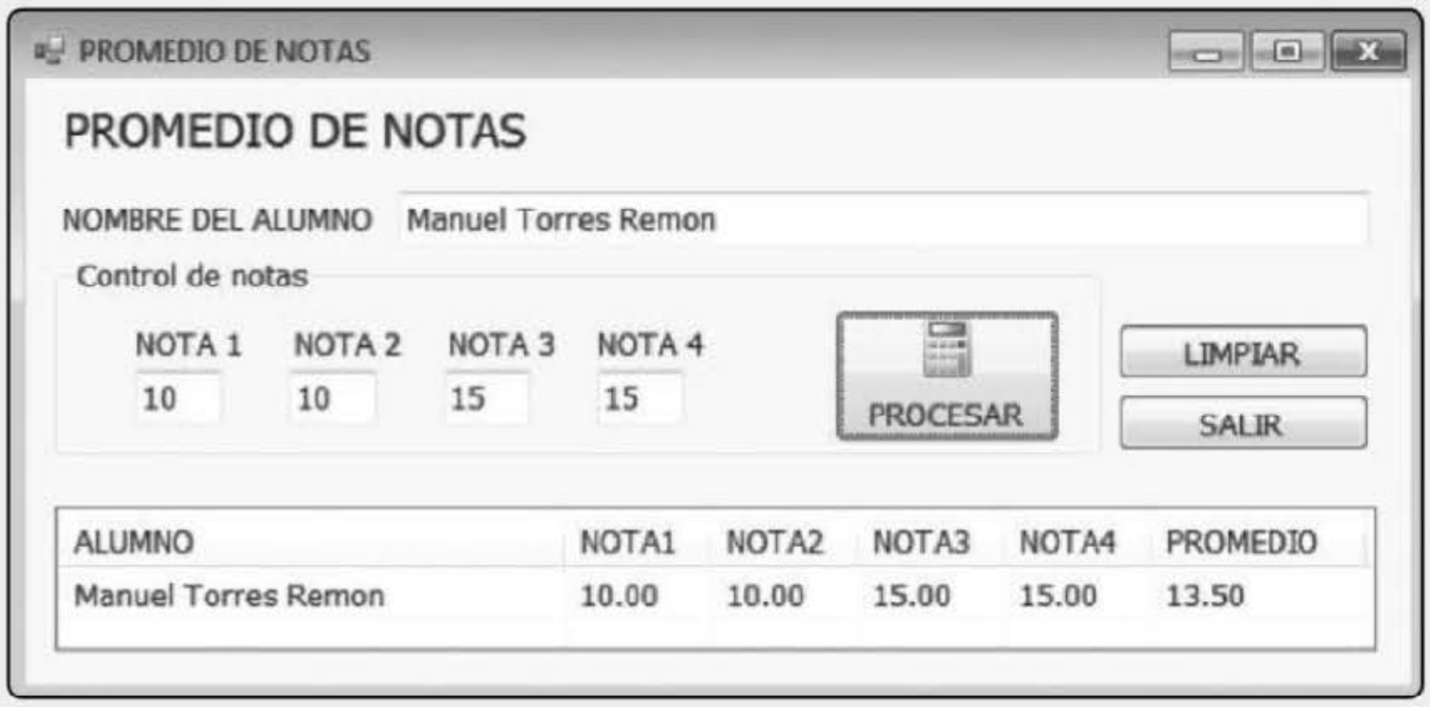
Label8	Text	TOTAL A PAGAR
Label9	(Name)	lblCantidad
Label10	(Name)	lblCosto
Label11	(Name)	lblMonto
Label12	(Name)	lblDescuento
Label13	(Name)	lblTotal

El script de la solución es:

```
Public Class frmHelado
    Const precio As Double = 1.5
    Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
        'CAPTURANDO LOS DATOS
        Dim cliente As String = txtCliente.Text
        Dim cantidad As Integer = Integer.Parse(txtCantidad.Text)

        'REALIZANDO CALCULOS
        Dim monto As Double = cantidad * precio
        Dim descuento As Double = monto * (7.5 / 100)
        Dim total = monto - descuento

        'IMPRIMIR RESULTADOS
        lblCantidad.Text = cantidad.ToString()
        lblCosto.Text = precio.ToString("C")
        lblMonto.Text = monto.ToString("C")
        lblDescuento.Text = descuento.ToString("C")
        lblTotal.Text = total.ToString("C")
    End Sub

    Private Sub btnLimpiar_Click(sender As Object, e As EventArgs)
        txtCliente.Clear()
        txtCantidad.Clear()
        lblCantidad.Text = ""
        lblCosto.Text = ""
        lblMonto.Text = ""
        lblDescuento.Text = ""
        lblTotal.Text = ""
        txtCliente.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs)
        Me.Close()
    End Sub
End Class
```

Caso desarrollado 4 Promedio de notas

Aplicación que permita determinar el promedio de un alumno dependiendo del ingreso de cuatro notas, para lo cual cada evaluación tiene un peso correspondiente.



Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar el nombre del alumno y sus cuatro notas.
- Los pesos de las notas son 10 %, 20 %, 30 % y 40 % según el orden de ingreso.
- El promedio de notas resulta del valor porcentual de cada nota según el peso.
- Mostrar los resultados en un formato numérico de dos decimales.
- Use el control ListView para mostrar los resultados.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmPromedio btnProcesar btnSalir PROMEDIO DE NOTAS
Label1	Text	PROMEDIO DE NOTAS
Label2	Text	NOMBRE DEL ALUMNO
Label3	Text	NOTA 1
Label4	Text	NOTA 2
Label5	Text	NOTA 3
Label6	Text	NOTA 4

TextBox1	(Name)	txtAlumno
TextBox2	(Name)	txtN1
TextBox3	(Name)	txtN2
TextBox4	(Name)	txtN3
TextBox5	(Name)	txtN4
Button1	(Name) Image ImageAlign Text TextAlign	btnProcesar (Seleccione una imagen) TopCenter PROCESAR BottomCenter
Button2	(Name) Text	btnSalir SALIR
Button3	(Name) Text	btnLimpiar LIMPIAR
GroupBox1	Text	Control de notas
ListView	(Name) GridLine Columns View	IvR True Agregue 6 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmPromedio
    Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
        'CAPTURANDO DATOS
        Dim alumno As String = txtAlumno.Text
        Dim n1 As Integer = Integer.Parse(txtN1.Text)
        Dim n2 As Integer = Integer.Parse(txtN2.Text)
        Dim n3 As Integer = Integer.Parse(txtN3.Text)
        Dim n4 As Integer = Integer.Parse(txtN4.Text)

        'REALIZANDO CALCULOS
        Dim promedio As Double = n1 * 0.1 + n2 * 0.2 + n3 * 0.3 + n4 * 0.4

        'IMPRIMIR RESULTADOS
        Dim fila As New ListViewItem(alumno)
        fila.SubItems.Add(n1.ToString("0.00"))
        fila.SubItems.Add(n2.ToString("0.00"))
        fila.SubItems.Add(n3.ToString("0.00"))
        fila.SubItems.Add(n4.ToString("0.00"))
```

```
        fila.SubItems.Add(promedio.ToString("0.00"))
        lvR.Items.Add(fila)
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs)
        Me.Close()
    End Sub

    Private Sub btnLimpiar_Click(sender As Object, e As EventArgs)
        txtAlumno.Clear()
        txtN1.Clear()
        txtN2.Clear()
        txtN3.Clear()
        txtN4.Clear()
        lvR.Items.Clear()
        txtAlumno.Focus()
    End Sub
End Class
```

Caso desarrollado 5 Hospedaje

Aplicación que permita determinar el monto total recaudado por hospedaje en un día determinado a un solo tipo de habitación.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Al iniciar la aplicación se deberá mostrar la fecha actual.
- Debe ingresar el nombre del cliente y la cantidad de días a hospedarse.

- El costo por día es un valor constante que asciende a $35.50.
- El monto resultado del producto entre los días de hospedaje y el costo por día.
- El monto total recaudado resulta del monto acumulado.
- Mostrar los montos con el formato de moneda.
- Los montos resultantes deberán mostrarse en el control ListView.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmHospedaje btnRegistrar btnSalir HOSPEDAJE – CONTROL DE PAGOS
Label1	Text	HOSPEDAJE – CONTROL DE PAGOS
Label2	Text	CLIENTE
Label3	Text	DÍAS
Label4	Text	FECHA
Label5	(Name)	lblFecha
Label6	Text	TOTAL RECAUDADO $
Label7	(Name)	lblTotal
TextBox1	(Name)	txtCliente
TextBox2	(Name)	txtDias
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnSalir SALIR
Button3	(Name) Text	btnAnular ANULAR
ListView	(Name) GridLine Columns View	lvR True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmHospedaje
    Const costo As Double = 35.5
    Private Sub frmHospedaje_Load(sender As Object, e As EventArgs)
        lblFecha.Text = DateAndTime.Now.ToShortDateString()
        lblTotal.Text = "0.00"
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'CAPTURANDO DATOS
        Dim cliente As String = txtCliente.Text
        Dim dias As Integer = Integer.Parse(txtDias.Text)

        'CALCULANDO MONTOS
        Dim monto As Double = costo * dias

        'IMPRIMIR
        Dim fila As New ListViewItem(cliente)
        fila.SubItems.Add(dias.ToString())
        fila.SubItems.Add(costo.ToString("C"))
        fila.SubItems.Add(monto.ToString("C"))
        lvR.Items.Add(fila)
        lblTotal.Text = (Double.Parse(lblTotal.Text) + monto).ToString("0.00")
    End Sub

    Private Sub btnAnular_Click(sender As Object, e As EventArgs)
        txtCliente.Clear()
        txtDias.Clear()
        txtCliente.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs)
        Me.Close()
    End Sub
End Class
```

Capítulo 3

Fundamentos de programación

Capacidad terminal:

Reconocer la forma y el uso de las variables en una aplicación, usando operadores y conversiones. Uso la clase String como formato de salida de información de nuestras aplicaciones.

Contenido

- Componentes de un código de aplicación
- Las variables
- Operadores
- Métodos matemáticos de .NET Framework
- Conversiones de tipo
- Clase Strings
- Enumeraciones
- La clase MessageBox
- Desarrollo de aplicaciones secuenciales

3.1 Componentes de un código de aplicación

Cuando se implementa una aplicación en Visual Basic 2015 se hace uso de diferentes elementos que permitirán desarrollar de la mejor manera un proceso de desarrollo; para esto, se debe conocer sus patrones y formatos para usarlos correctamente. Todo esto referido a la forma de declarar una variable, asignarle un valor o imprimir su valor.

3.1.1 Comentarios en el código fuente

Un código fuente puede contener diversos elementos para lograr un proceso de desarrollo; uno de ellos son los comentarios, los cuales son considerados como sentencias que no serán interpretadas por el compilador. Se puede colocar un comentario en las siguientes situaciones:

- Cuando se quiere colocar el nombre del autor del código.
- Cuando se quiere indicar al usuario sobre el funcionamiento de un bloque de código.
- Cuando se necesita anular una o más sentencias de código sin eliminarlos del editor de código.

A continuación, el formato para la asignación de comentarios:

Primera forma: Comentario de línea

- Ubique el cursor en una línea de código desde el editor.
- Digite un apostrofe simple «'» o coloque la cláusula REM.
- Escribir su comentario.

Segunda forma: Comentario de múltiple líneas

- Seleccione un bloque de líneas de código.
- Presionar las teclas <Ctrl + K>.
- Presionar las teclas <Ctrl + C>.

Por ejemplo, el siguiente código permite determinar el subtotal a pagar por la venta de un determinado producto, usando comentarios de línea y múltiples líneas.

```
COMENTARIO DE MÚLTIPLES LÍNEAS  'Método que permite devolver los cálculos
                                'del proceso de venta
                                Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
                                    'CAPTURANDO LOS DATOS
                                    Dim cliente As String = txtCliente.Text
                                    Dim cantidad As Integer = Integer.Parse(txtCantidad.Text)

                                    'REALIZANDO CALCULOS
                                    Dim monto As Double = cantidad * precio
                                    Dim descuento As Double = monto * (7.5 / 100)
                                    Dim total = monto - descuento

COMENTARIO DE LÍNEA             'IMPRIMIR RESULTADOS
                                    lblCantidad.Text = cantidad.ToString()
                                    lblCosto.Text = precio.ToString("C")
                                    lblMonto.Text = monto.ToString("C")
                                    lblDescuento.Text = descuento.ToString("C")
                                    lblTotal.Text = total.ToString("C")
                                End Sub
                                REM Fin de la implementación del método
```

3.1.2 Tipos de datos

Un tipo de datos hace referencia al tipo de valor y al tamaño que puede contener un determinado elemento de programación. Estos pueden ser:

Elemento	Uso del tipo
Variable	Dim variable As TipoDatos, por ejemplo: `Dim n As Integer`
Constante	Const constante As TipoDatos, por ejemplo: `Const PI As Double = 3.1416`

Argumento de procedimiento	Sub NombreProcedimiento(ByVal parámetro As TipoDatos), por ejemplo:
	`Sub calculaDescuento(ByVal monto as Double)`
	`  …`
	`End Sub`

Argumento de función	Function NombreFuncion(ByVal parámetro As TipoDatos) As TipoDatos, por ejemplo:
	`Function calculaDescuento(ByVal monto as Double) As Double`
	`  …`
	`  Return valor`
	`End Sub`

Los tipos de datos definen la capacidad de datos que puede recibir un determinado elemento de programación como lo son las variables u otros tipos, tal como se muestra en la tabla anterior. A continuación, las capacidades que presenta cada uno de estos tipos, ya que de esto dependerá la decisión del programador en usar uno u otro tipo:

A. Tipo de datos numérico entero

Un valor numérico entero se caracteriza por no presentar elementos decimales.

Tipo	Almacenamiento	Capacidad	System de .net
Byte	1 byte	0 a 255	Byte

SByte	1 byte	-128 a 127	SByte
Short	2 bytes	-32768 a +32767	Short
UShort	2 bytes	0 a 65535	UShort
Integer	4 bytes	-2147483648 a 2147483647	Int32
UInteger	4 bytes	-2147483648 a 2147483647	UInt32
Long	8 bytes	-9223372036854775808 a 9223372036854775807	Int64
ULong	8 bytes	0 a 18446744073709551615	UInt64

Por ejemplo, podemos mencionar las siguientes variables que podrían declararse como datos numéricos enteros según su condición:

Descripción	Variable
Cantidad de productos seleccionados en una compra.	Cantidad
Registrar la edad de los participantes a un evento.	Edad
Registrar el año de nacimiento de un estudiante.	AñoNacimiento

B. Tipo de datos numérico-decimal

Un valor numérico decimal se caracteriza por incluir todos los valores números enteros más los valores decimales o fraccionarios.

Tipo	Almacenamiento	Capacidad	System de .net
Double	8 bytes	±4.940656458412465E−324 a ±1.79769313486232E+308	Double
Decimal	16 bytes	−7.9228162514264337593543950335 a +7.9228162514264337593543950335	Decimal
Single	4 bytes	1,401298E-45 a 3,4028235E+38	Single

Por ejemplo, podemos mencionar las siguientes variables que podrían declararse como datos numéricos decimales según su condición:

Descripción	Variable
El precio de un determinado producto	Precio
El promedio ponderado de un estudiante	Promedio
Monto a pagar producto de una deuda por parte de un cliente	Monto

C. Tipo de datos carácter

Un valor de tipo carácter se caracteriza por almacenar un solo carácter de cualquier tipo, ya sea letra, numero o símbolo.

Tipo	Tamaño	Capacidad	System de .net
Char	2 bytes	0 a 65535	Char

Por ejemplo, se puede mencionar las siguientes variables que podrían declararse como datos de tipo carácter según su condición:

Descripción	Variable
El estado civil de los empleados en una empresa por ejemplo Soltero(s), Casado (c), Divorciado (d), etc.	EstadoCivil
La categoría de los estudiantes en un proceso de categorización.	Categoría
El sexo de los empleados en el registro del nuevo personal en una empresa.	Sexo

D. Tipo de datos lógica

Un valor de tipo lógica se caracteriza por solo contener valores True y False. Son usados en valores que solo pueden contener dos valores; por ejemplo, en el tipo de sexo.

Tipo	Tamaño	Capacidad	System de .net
Boolean	1 byte	True o False	Boolean

Por ejemplo, se puede mencionar las siguientes variables que podrían declararse como datos lógicos según su condición:

Descripción	Variable
El tipo de sexo de los empleados nuevos en una empresa Masculino (True) y Femenino (False).	Sexo
La condición de un alumno al rendir un examen Aprobado (True) y Desaprobado (False).	Condición

E. Tipo de datos cadena de caracteres

Un valor de tipo cadena de caracteres se caracteriza por contener un conjunto de caracteres en los cuales se puede incluir toda clase de símbolos, inclusive cualquier tipo de valor numérico.

Tipo	Tamaño	Capacidad	System de .NET
String	Variable	0 a 2.000 millones de caracteres Unicode aprox.	String

Por ejemplo, se puede mencionar las siguientes variables que podrían declararse como datos tipo cadena según su condición:

Descripción	Variable
Descripción de un producto en una tienda	Descripción
Nombre completo de un empleado	Nombres
Numero de documento de identidad de una persona	DNI

F. Tipo de datos fecha y hora

Un valor de tipo fecha y hora se caracteriza por contener fechas y horas en todos sus formatos.

Tipo	Tamaño	Capacidad	System de .NET
DateTime	8 bytes	0:00:00 (medianoche) del 1 de enero de 0001 a 11:59:59 p.m. del 31 de diciembre de 9999.	DateTime

Por ejemplo, se puede mencionar las siguientes variables que podrían declararse como datos tipo fecha según su condición:

Descripción	Variable
Fecha de nacimiento de un nuevo empleado	fechaNacimiento
Fecha de vencimiento de un producto en tienda	fechaVencimiento
Hora de ingreso en el control de entrada del personal	horaEntrada

3.1.3 Identificadores

Un identificador es considerado como un símbolo léxico que permite asignar un nombre a un elemento de programación. En este caso se usará para la identificación de variables, parámetros, constantes, nombres de clase, nombre de métodos, etc.

Antes de crear un identificador deberá tener en cuenta los siguientes aspectos:

- Siempre debe empezar con una letra. El símbolo aceptado para el inicio de un identificador es el guion bajo «_».

- A partir del segundo carácter puede usar letras, números y guion bajo.
- No debe contener espacios en blanco.
- No debe contener caracteres especiales.
- No debe ser una palabra reservada del lenguaje Basic.
- En Visual Basic no distingue las mayúsculas de las minúsculas.
- Puede contener 511 como longitud máxima de caracteres.

En el siguiente proceso de negocio haremos una lista de identificadores válidos y no validos:

Una tienda de prendas de vestir promociona la venta de polos importados a solo $14.50 la unidad y en las tallas S,M,L. Se le contrata para implementar una aplicación que permita controlar dicha venta, de tal manera que se pueda saber el número de polos vendidos por cada talla, monto acumulado de la venta por cada talla y el monto total correspondiente a la venta de todos los polos.

Valores de entrada y salida	Identificadores válidos	Identificadores no válidos
Cantidad de polos	cantidad, cantidadPolos, cPolos	cantidad polos, cantidad%polos
Talla del polo	talla, tallaPolo, tPolo	Talla polo, #talla, @talla
Subtotal de la venta	subtotal, subtotalVenta, sVenta	Subtotal venta, +subtotal_
Total de polos por cada talla	totalS, totalM, totalL	Total S, Total M, Total L
Monto total por cada talla	montoS, montoM, montoL	montoS, Monto M, Monto L
Monto total	total, montoTotal	1234Monto total

3.1.4 Palabras reservadas

Una palabra reservada no puede ser considerada como un elemento identificador en una aplicación. A continuación, una lista de palabras reservadas:

AddHandler	AddressOf	Alias	And	AndAlso
As	Boolean	ByRef	Byte	ByVal
Call	Case	Catch	CBool	CByte
CChar	CDate	CDec	CDbl	Char
CInt	Class	CLng	CObj	Const
Continue	CSByte	CShort	CSng	CStr
CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim

DirectCast	Do	Double	Each	Else
ElseIf	End	EndIf	Enum	Erase
Error	Event	Exit	False	Finally
For	Friend	Function	Get	GetType
GetXMLNamespace	Global	GoSub	GoTo	Handles
If	If()	Implements	In	Inherits
Integer	Interface	Is	IsNot	Let
Lib	Like	Long	Loop	Me
Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Narrowing	New	Next
Not	Nothing	NotInheritable	NotOverridable	Object
Of	On	Operator	Option	Optional
Or	OrElse	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property	Protected
Public	RaiseEvent	ReadOnly	ReDim	REM
RemoveHandler	Resume	Return	SByte	Select
Set	Shadows	Shared	Short	Single
Static	Step	Stop	String	Structure
Sub	SyncLock	Then	Throw	To
True	Try	TryCast	TypeOf	Variant
Wend	UInteger	ULong	UShort	Using
When	While	Widening	With	WithEvents
WriteOnly	Xor			

3.1.5 Literales

Representa un valor que se puede ingresar en la ejecución de una aplicación; dicho valor dependerá del tipo de dato declarado. A continuación, una tabla con los posibles literales de los tipos de datos más usados:

Literales enteros: Son valores que por su tipo solo pueden ser numéricos sin fracción decimal.

Tipo de datos	Literal forzado	Literal personalizado
Short	S (123S)	Dim nShort As Short = 123
Integer	I (123I)	Dim nInteger As Integer = 12345
Long	L (123L)	Dim nLong As Long = 1234567

Literales reales: Son valores que por su tipo pueden ser números entero o fraccionarios.

Tipo de datos	Literal forzado	Literal personalizado
Decimal	D (123.45D)	Dim nDecimal As Decimal = 123.45
Single	F (1234.56F)	Dim nSingle As Single = 1234.56
Double	R (12345.67R)	Dim nDouble As Double = 12345.67

Literales de tipo carácter: Son valores que por su tipo solo representan a un carácter de cualquier tipo. Puede ser letra, numero o símbolo.

Tipo de datos	Literal forzado	Literal personalizado
Char	C ("L".C)	Dim letra As Char = "L"

Literal booleano: Son valores que por su tipo solo aceptan los valores True y False.

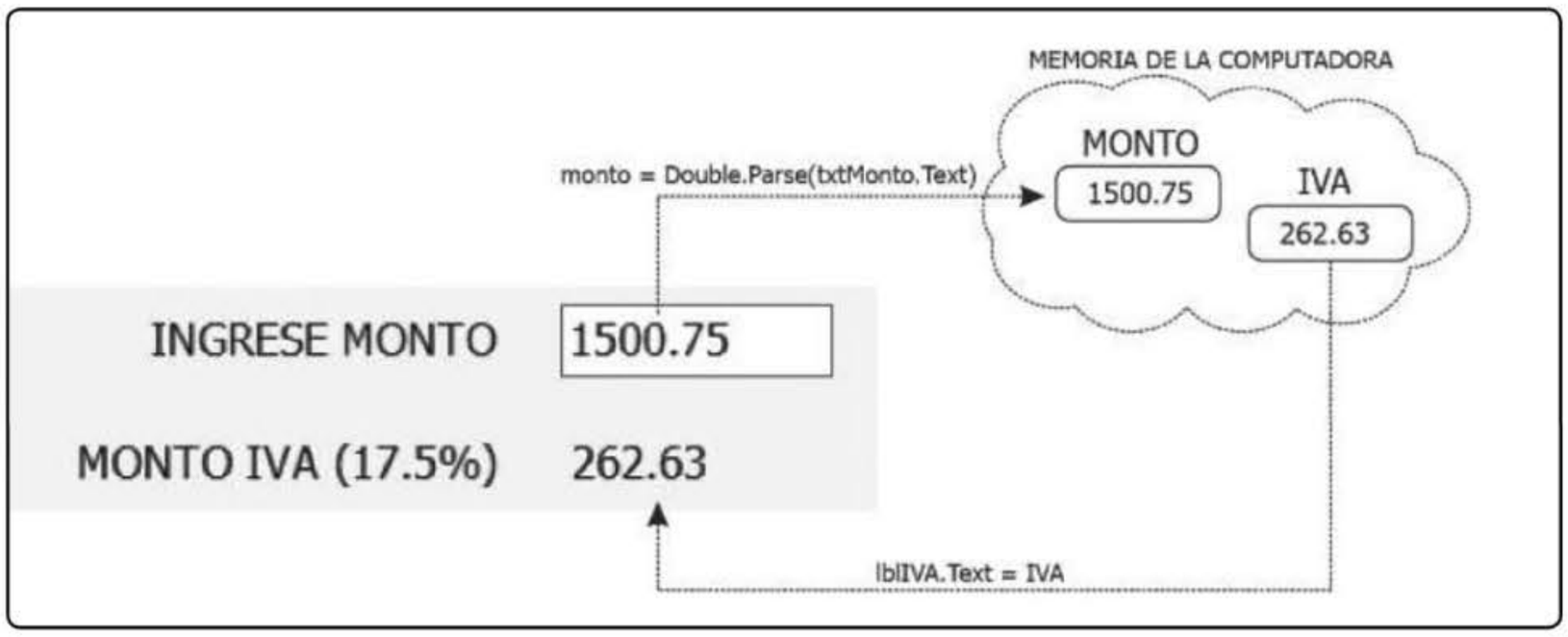
Tipo de datos	Literal forzado	Literal personalizado
Boolean	–	Dim opción As Boolean = True

Literales de cadena: Son valores que por su tipo aceptan todo tipo de carácter ya sea letra, numero, símbolos o la combinación de ellos.

Tipo de datos	Literal forzado	Literal personalizado
String	–	Dim descripcion As String = "Radio 500W"

3.2 Las variables

En Visual Basic 2015 todos los elementos de programación deben ser declarados. Una variable es un espacio de memoria; esta separación de memoria se realiza cuando se declara la variable. El valor contenido en la variable es el literal de un determinado tipo de datos. A continuación, un gráfico para analizar:



Como se observa en el gráfico, existe un entorno en el cual se ingresa un monto en una caja de texto llamada txtMonto, una vez ingresado el valor es enviado a la memoria para ser procesado, por ejemplo, para calcular el 17.5 correspondiente al IVA. Una vez que la fórmula haga su trabajo, almacenará el resultado en la variable IVA también almacenado en la memoria y es justamente desde allí que se obtendrá el resultado que es mostrado mediante el control label (lblIVA). El código que se usó para completar el proceso es el siguiente:

```
Dim monto As Double
monto = Double.Parse(txtMonto.Text)
Dim IVA As  Double
IVA = 0.175 * monto
lblIVA.Text = IVA
```

Formato de declaración de variable:

Dim NombreVariable As TipoDatos

Donde:

- **Dim:** Palabra reservada que marca el punto de inicio de la declaración de variables.
- **NombreVariable:** Representa el nombre de la variable, se recomienda que el nombre asignado tenga relación con su contenido.
- **TipoDatos:** Representa el tipo y capacidad que puede tener una variable en una aplicación. Estos pueden ser enteros, decimales, caracteres, cadenas, lógicos, fecha y hora, etc.

A continuación, un proceso de negocio a partir del cual se puede declarar las variables necesarias:

Una tienda de prendas de vestir promociona la venta de polos importados a solo $14.50 la unidad y en las tallas S,M,L. Se le contrata para implementar una aplicación que permita controlar dicha venta, de tal manera que se pueda saber el número de polos vendidos por cada talla, monto acumulado de la venta por cada talla y el monto total correspondiente a la venta de todos los polos.

Valores de entrada y salida	Declaración de variable
Cantidad de polos	Dim cantidad As Integer
Talla del polo	Dim talla As Double
Subtotal de la venta	Dim subtotal As Double
Total de polos por cada talla	Dim totalS As Integer, totalM As Integer, totalL As Integer
Monto total por cada talla	Dim montoS As Double, montoM As Double, montoL As Double
Monto total	Dim montoTotal As Double

Se podría usar resúmenes al momento de declarar un grupo de variables, de tal forma que el caso anterior podría resolverse de la siguiente manera:

```
Dim cantidad As Integer, totalS As Integer, totalM As Integer, totalL As Integer
Dim talla As Double, subtotal As Double, montoTotal As Double
Dim montoS As Double, montoM As Double, montoL As Double
```

Finalmente, se podría declarar de forma múltiple así:

```
Dim cantidad, totalS, totalM, totalL As Integer
Dim talla, subtotal, montoTotal As Double
Dim montoS, montoM, montoL As Double
```

3.2.1 Literales de tipo identificador

Al realizar la declaración de variables se puede usar algunos literales implementados solo para el lenguaje Visual Basic. A continuación, una lista de los literales y un ejemplo básico para su implementación:

Tipo de datos	Literal	Implementación
Integer	%	Dim n As Integer; es análogo a Dim n%
Long	&	Dim n As Long; es análogo a Dim n&
Single	!	Dim n As Single; es análogo a Dim n!
Decimal	@	Dim subtotal As Decimal; es análogo a Dim subtotal@
Double	#	Dim monto As Double; es análogo a Dim monto#
String	$	Dim descripción As String; es análogo a Dim descripcion$

3.2.2 Niveles de declaración

El ámbito de una variable es la sección o zona de una aplicación en la que una o más variables son definidas; estas pueden estar definidas como locales y globales.

Un ámbito local implica la declaración de una variable en un bloque de código y esta solo será referenciada dentro de la misma; es decir, el valor que contenga dicha variable se perderá cuando el bloque finalice. Un ámbito global implica la declaración de una variable generalizada, es decir, dicha variable podrá ser usada en todos los bloques de una misma aplicación.

Ahora, el código de una aplicación Windows Forms, donde se puede declarar variables locales y globales:

```
Public Class frmCompra

    Dim conteo As Integer                                    AMBITO GLOBAL
    Private Sub frmCompra_Load(...) Handles MyBase.Load

    End Sub

    Private Sub btnProcesar_Click(...) Handles btnProcesar.Click
        Dim monto As Double
                                                             AMBITO LOCAL
    End Sub
End Class
```

3.2.3 Declaración de variables de tipo variante

Este tipo de declaración se da cuando no se tiene claro el valor (literal) que puede tener una variable.

Formato:

Dim nombreVariable

Dim nombreVariable **As Object**

Ahora, algunos casos de implementación de variables variante

- Declarar y asignar un valor de tipo cadena
  - Dim descripcion = "Impresora Epson C32"
- Declarar y asignar el precio de un determinado producto
  - Dim precio As Object = 650.75

En la primera declaración la variable Descripción se declara como String solo por recibir una cadena de caracteres. Por otro lado, en la segunda declaración la variable Precio es de tipo Object, el cual se considera como variante entre los tipos de datos.

3.3 Operadores

Las sentencias de código que contiene una aplicación la componen los operadores, los cuales se representan mediante símbolos, normalmente se usa para implementar sentencias o expresiones.

3.3.1 Operador de asignación:

Este operador tiene una doble misión en Visual Basic pues puede representar la asignación de un valor a una variable y en una estructura condicional puede representar la comparación entre dos variables.

Operador	Descripción	Ejemplo
=	Permite asignar un determinado valor a una variable; el elemento situado a la derecha asigna valor a la izquierda.	**'Asignación simple** Dim n As Integer n = 10; **'Asignacion directa** Dim n As Integer = 10 **'Asignacion para expresión simple** Dim suma As Double; suma = 10 + 12.5 + 15 **'Asignacion para expresión directa** Dim suma As Double = 10 + 12.5 + 15

3.3.2 Operadores aritméticos:

Estos operadores nos permitirán componer una expresión dentro de una determinada situación de la aplicación.

Operador	Descripción	Ejemplo
+	Representa la suma entre uno o más valores. Cuando los valores son de tipo cadena el operador + se comporta como un elemento de concatenación.	'Declarando variables Dim num1, num2, suma As Integer 'Asignando valores num1 = 10 num2 = 20 'Expresión de suma suma = num1 + num2
-	Representa la resta de dos valores en una expresión.	'Declarando variables Dim num1, num2, resta As Integer 'Asignando valores num1 = 20 num2 = 10 'Expresión de resta resta = num1 - num2

*	Representa la multiplicación entre dos o más valores.	'Declarando variables Dim num1, num2, producto As Integer 'Asignando valores num1 = 5 num2 = 6 'Expresión de multiplicación producto = num1 * num2
/	Representa la división real entre dos valores numéricos en una expresión.	'Declarando variables Dim descuento As Double 'Asignando valores descuento = 1000 * (12 / 100)
MOD	Representa el resto de la división entre dos valores en una expresión.	'Declarando variables Dim num1, num2, resto As Integer 'Asignando valores num1=5 num2=2 'Expresión de resto resto = num1 MOD num2
\	Representa la división entera entre dos valores numéricos en una expresión.	'Declarando variables Dim n1, n2, promedio As Integer 'Asignando valores n1=5 n2=2 'Expresión de promedio promedio = n1 \ n2
^	Representa la potencia de un valor y un elemento base.	'Declarando variables Dim n, base, potencia As Integer 'Asignando valores n = 5 base = 2 'Expresión de potencia potencia = n ^ base
&	Representa la concatenación entre los elementos de programación.	'Declarando variables Dim nombres, apellidos As String 'Asignando valores nombres = "Fernanda Ximena" apellidos = "Torres Lázaro" 'Expresión de concatenación Dim empleado As String empleado = nombres & Strings.Space(1) & apellidos

3.3.3 Prioridad de los operadores aritméticos

Los operadores aritméticos permiten componer una expresión que permitirá devolver un valor de tipo numérico o real, para esto, como en todo proceso matemático, se debe tener en cuenta el orden de prioridad de los operadores que se verán a continuación:

Operador	Orden de prioridad
()	Los paréntesis son los operadores de más alto nivel de prioridad dentro de una expresión.
^	Potencia o elevado
*, /	Multiplicación y división real
\	División entera
MOD	Resto de la división
+, -	Suma y Resta

Aquí se ve cómo representar una expresión aritmética en un código Visual Basic:

Expresión aritmetica	Código Visual Basic
$r = \frac{3}{4} x^2y + 2xy$	Dim r As Double r = (3/4) * (x^2*y) + (2*x*y)
$v = 2\pi r - \pi r$	Dim v As Double v = (2*3.1416*r) - (3.1416*r)
$r = \frac{4}{x^2 - 2x} - \frac{2}{x - 2}$	Dim r As Double r = (4/(x*x-2*x)) – 2/(x-2)

3.4 Métodos matemáticos de .NET Framework

Existen métodos matemáticos implementados en el Framework de Visual Studio, el cual nos permitirá desarrollar expresiones trigonométricas, logarítmicas y otras funciones matemáticas comunes. A continuación, los principales métodos que se incluyen dentro de la clase Math, el cual proviene del espacio de nombres System:

Método	Descripción	Ejemplo
Abs	Método que determina el valor absoluto de un valor.	Dim n As Integer = -10 Dim valorAbsoluto As Integer = Math.Abs(n) 'Resultado > 10

3.3.3 Prioridad de los operadores aritméticos

Los operadores aritméticos permiten componer una expresión que permitirá devolver un valor de tipo numérico o real, para esto, como en todo proceso matemático, se debe tener en cuenta el orden de prioridad de los operadores que se verán a continuación:

Operador	Orden de prioridad
()	Los paréntesis son los operadores de más alto nivel de prioridad dentro de una expresión.
^	Potencia o elevado
*, /	Multiplicación y división real
\	División entera
MOD	Resto de la división
+, -	Suma y Resta

Aquí se ve cómo representar una expresión aritmética en un código Visual Basic:

Expresión aritmetica	Código Visual Basic
$r = \frac{3}{4} x^2y + 2xy$	Dim r As Double r = (3/4) * (x^2*y) + (2*x*y)
$v = 2\pi r - \pi r$	Dim v As Double v = (2*3.1416*r) - (3.1416*r)
$r = \frac{4}{x^2 - 2x} - \frac{2}{x - 2}$	Dim r As Double r = (4/(x*x-2*x)) – 2/(x-2)

3.4 Métodos matemáticos de .NET Framework

Existen métodos matemáticos implementados en el Framework de Visual Studio, el cual nos permitirá desarrollar expresiones trigonométricas, logarítmicas y otras funciones matemáticas comunes. A continuación, los principales métodos que se incluyen dentro de la clase Math, el cual proviene del espacio de nombres System:

Método	Descripción	Ejemplo
Abs	Método que determina el valor absoluto de un valor.	Dim n As Integer = -10 Dim valorAbsoluto As Integer = Math.Abs(n) 'Resultado > 10

Ceiling	Método que determina el valor más pequeño que es mayor o igual que el número decimal especificado.	Dim n As Double = 2.1 Dim valorCeiling As Integer = Math.Ceiling(n) 'Resultado > 3
Floor	Método que determina el valor más grande menor o igual que el número decimal especificado	Dim n As Double = 2.7; Dim valorFloor As Integer = Math.Floor(n) 'Resultado > 2
Max	Determina el mayor valor entre dos valores del mismo tipo.	Dim n1 As Integer=10, n2 As Integer = 20 Dim mayor As Integer = Math.Max(n1, n2) 'Resultado > 2
Min	Determina el menor valor entre dos valores del mismo tipo.	Dim n1 As Integer=10, n2 As Integer = 20 Dim menor As Integer = Math.Min(n1, n2) 'Resultado > 2
Pow	Determina el valor de la potencia entre un número y su base.	'Elevando un numero al cuadrado Dim n As Integer = 2; Dim potencia As Double = Math.Pow(n,2) 'Raíz al cubo de un numero Dim n As Integer = 3 Dim raizC As Double = Math.Pow(n , 1/3) 'Resultado > 1.44224957030741
Round	Determina el redondeo de un valor al integral más próximo.	Dim n As Double = 2.675 Dim valorR As Double = Math.Round(n,2) 'Resultado > 2.68
Sqrt	Determina la raíz cuadrada de un valor.	Dim n As Integer = 2 Dim raizC As Double = Math.Sqrt(n) 'Respuesta > 1.4142135623731
E	Representa la base logarítmica natural, especificada para la constante e.	Dim e As Double = Math.E 'Respuesta > 2.71828182845905
PI	Representa el valor de PI.	Dim e As Double = Math.PI 'Respuesta > 3.14159265358979

Aquí se ve cómo representar una expresión aritmética con código Visual Basic, usando métodos de la clase Math:

Expresión aritmetica	Código C#
$r = \frac{3}{4} x^2y + 2xy$	Dim r As Double r = (3/4) * (Math.Pow(x,2)*y) + (2*x*y)
$v = 2\pi r - \pi r$	Dim v As Double v = (2*Math.PI*x) - (Math.PI * r)

3.5 Conversiones de tipo

Las conversiones de tipos se refieren a una conversión realizada a un valor previamente declarado como un tipo de datos. Aquí se detalla cuándo será necesario realizar una conversión:

- Si se declara una variable como entero y el valor entero viene representado como cadena.
- Cuando necesite concatenar o imprimir un valor numérico en una cadena de texto.
- Cuando dos valores numéricos necesiten concatenarse y no sumarse.
- Cuando el resultado de una expresión es diferente al tipo de datos declarado.
- Cuando la función envía un valor de retorno en un tipo de datos especifico.

Para todos estos casos existen categorías de conversiones que veremos a continuación:

3.5.1 Conversión implícita

Una conversión implícita se caracteriza por no requerir formato particular. Aquí se ve cómo implementar una conversión implícita:

```
Dim valor As Integer = 123
Dim valorConvertido As Double = valor
```

El dato 123, asignado inicialmente a la variable Valor, es asignado a la variable valorConvertido de tipo Double, la cual pasa dicha información sin perder precisión en el dato. Aquí, una lista de conversiones numéricas implícitas:

Tipo original	Tipo destino
Byte	Byte, Short , Ushort, Integer, Uinteger, Long, Ulong, Single, Double o Decimal
Short	Short, Integer, Long, Single, Double o Decimal
Integer	Integer, Long, Single, Double o Decimal

Long	Long, Single, Double o Decimal
Char	Char o String
Single	Single, Double
Double	Double

El tipo original representa el tipo de dato inicial y el tipo destino representa al que se desea convertir, pero al encontrarse con la tabla se nota que existen conversiones implícitas y no se necesita de una función para su conversión.

3.5.2 Conversión explicita

Se llama así a las conversiones que se realiza con seguridad de tipos y con pérdida de valor. Cuando se necesite el valor perdido será mejor usar una función de conversión.

Las funciones de conversión tienen la misión de convertir un valor de una expresión en un determinado tipo de datos.

3.5.3 Conversión mediante palabras claves

Ahora, una tabla que muestra las principales conversiones mediante palabras claves:

Palabra	Descripción	Ejemplo

CDate	Convierte desde un tipo String al tipo Date.	`Dim fecha As String = "12/10/2017"`
		`Dim año As Integer = CDate(fecha).Year`
		`'Respuesta: 2017`

CDbl	Convierte desde cualquier tipo numérico al tipo Double; incluye también las cadenas de caracteres.	`Dim n As Integer = 123`
		`Dim numero As Double = CDbl(n)`
		`'Respuesta: 123`

CDec	Convierte desde cualquier tipo numérico al tipo Decimal; incluye también las cadenas de caracteres.	`Dim n As Integer = 123`
		`Dim numero As Decimal = CDec(n)`
		`'Respuesta: 123`

CInt	Convierte desde cualquier tipo numérico al tipo Integer; incluye también las cadenas de caracteres.	`Dim n As String = "123"`
		`Dim numero As Integer = CInt(n)`
		`'Respuesta: 123`

Palabra	Descripción	Ejemplo

CLng	Convierte desde cualquier tipo numérico al tipo Long; incluye también las cadenas de caracteres.	`Dim n As Integer = 123`
		`Dim numero As Long = CLng(n)`
		`'Respuesta: 123`

CStr	Convierte desde cualquier tipo al tipo String.	`Dim n As Integer = 123`
		`Dim numero As String = CStr(n)`
		`'Respuesta: 123`

CSng	Convierte desde cualquier tipo numérico al tipo Single; incluye también las cadenas de caracteres.	`Dim n As Integer = 123`
		`Dim numero As Single = CSng(n)`
		`'Respuesta: 123`

3.5.4 Conversión mediante la función CType

CType es una función que proporciona una serie completa de conversiones compatibles. Dicha función realiza conversiones comprobadas y siempre produce una excepción si no se admite la conversión.

Su formato es:

CType(Expresión o valor, TipoDatos)

Aquí se ve cómo hacer uso de la función CType

```
Dim n As Integer = 123
Dim numero As Double = CType(n, Double)
```

3.5.5 Conversión mediante el Método Parse

El Método Parse permite convertir un valor de tipo String a un tipo de datos distinto.

Método	Descripción	Ejemplo

Integer.
Parse
	Convierte desde un tipo cadena a un tipo Integer.	`Dim n As String = "123"`
		`Dim numero As Integer`
		`numero = Integer.Parse(n)`
		`'Respuesta: 123`

Double.
Parse
	Convierte desde un tipo cadena a un tipo Double.	`Dim n As String = "123.45"`
		`Dim numero As Double`
		`numero = Double.Parse(n)`
		`'Respuesta: 123.45`

Date.Parse	Convierte desde un tipo cadena a un tipo Date.	`Dim f As String = "15/05/2017"`
		`Dim fecha As Date`
		`fecha = Date.Parse(f)`
		`'Respuesta: 15/05/2017`

Boolean.
Parse
	Convierte desde un tipo cadena a un tipo Boolean.	`Dim condicion As String = "False"`
		`Dim valor As Boolean`
		`valor = Boolean.Parse(condicion)`
		`'Respuesta: False`

3.6 Clase Strings

Representa a una cadena de caracteres. Puede contener todo tipo de valores como textos, números, símbolos, etc. Internamente, dicha cadena se almacena de forma secuencial en objetos de tipo Char, es decir, una cadena de caracteres es una composición de elementos Char. La clase Strings contiene métodos que permiten manejar información de una cadena.

Métodos de la clase String

Método	Descripción

LCase	Convierte una cadena de caracteres en minúsculas.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim minúsculas As String = LCase(cadena)`
	'Resultado: visual basic 2015

UCase	Convierte una cadena de caracteres en mayúsculas.
	`Dim cadena As String = "visual basic 2015"`
	`Dim mayúsculas As String = UCase(cadena)`
	'Resultado: VISUAL BASIC 2015

Método	Descripción

Left	Obtiene una cantidad de caracteres desde el lado izquierdo de la cadena.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim izquierda As String = Strings.Left(cadena, 6)`
	'Resultado: VISUAL

Len	Obtiene el número total de caracteres de una cadena.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim total As Integer = Strings.Len(cadena)`
	'Resultado: 17

LTrim	Permite eliminar los espacios en blanco del lado izquierdo de una cadena de caracteres.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim letras As String = Strings.LTrim(cadena)`
	'Resultado: VISUAL BASIC 2015

RTrim	Permite eliminar los espacios en blanco del lado derecho de una cadena de caracteres.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim letras As String = Strings.RTrim(cadena)`
	'Resultado: VISUAL BASIC 2015

Trim	Permite eliminar los espacios en blanco, tanto del lado izquierdo como el derecho, de una cadena de caracteres.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim letras As String = Strings.Trim(cadena)`
	'Resultado: VISUAL BASIC 2015

Right	Obtiene una cantidad de caracteres desde el lado derecho de la cadena.
	`Dim cadena As String = "VISUAL BASIC 2015"`
	`Dim derecha As String = Strings.Right(cadena, 4)`
	'Resultado: 2015

Space	Permita asignar un determinado número de espacios en blanco.
	`Dim nombres As String = "FERNANDA"`
	`Dim apellidos As String = "TORRES LAZARO"`
	`Dim cliente As String = nombres + Strings.Space(1)+apellidos`
	'Resultado: FERNANDA TORRES LAZARO

ToString	Retorna una cadena de caracteres a partir de un valor distinto al tipo String.
	`Dim n As Integer = 123`
	`Dim texto As String = n.ToString`

3.7 Enumeraciones

Es un conjunto de valores bajo un mismo nombre, como si fuera una matriz de elementos, cuya característica principal es que sus valores son considerados como constantes. Una enumeración se declara fuera de la clase para que sea accesible en toda la aplicación. A continuación, algunas características que presentan las enumeraciones:

- Los elementos de una enumeración se almacenan en un orden inicial de posición cero y el último elemento tiene la posición n-1.
- Normalmente, la declaración de una numeración es de tipo numérica entera.
- La enumeración puede ser usada en cualquier parte de la aplicación.
- Todos los valores contenidos en la enumeración son considerados como constantes.

Declaración de la enumeración

Los elementos que contiene una enumeración son almacenados en una posición correlativa iniciado en cero.

```
Enum NombreEnumeracion As Integer
  Elemento1
  Elemento2
  ....
  ElementoN
End Num
```

También, se podría alterar el valor de su posición usando la siguiente declaración:

```
Enum NombreEnumeracion As Integer
  Elemento1 = ValorNumericoEntero
  Elemento2 = ValorNumericoEntero
  ....
  ElementoN = ValorNumericoEntero
End Num
```

Aquí, algunas opciones que se presenta en el uso de las enumeraciones:

Declarar una enumeración de productos:

```
Enum misProductos As Integer
 Lavadora
 Refrigeradora
 Licuadora
 Extractora
End Enum
```

Declarar una enumeración de productos con valor de índice personalizado:

```
Enum misProductos As Integer
 Lavadora = 1
 Refrigeradora = 3
 Licuadora = 2
 Extractora = 0
End Enum
```

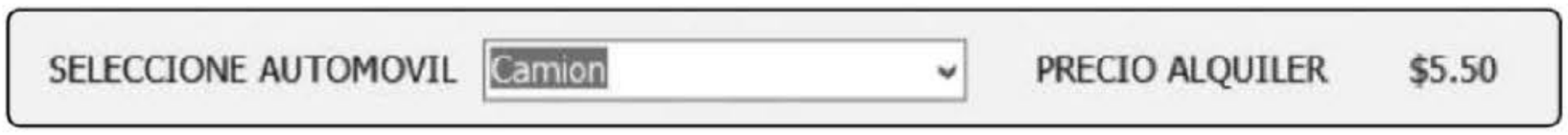
Mostrar los valores contenidos en una enumeración; para esto, se usa la estructura repetitiva For Each, para recorrer todos los elementos de la enumeración; para obtener su contenido, se usa en método GetNames:

```
For Each a In System.Enum.GetNames(GetType(tiposAutos))
    MessageBox.Show(a.ToString)
Next
```

Mostrar el índice de todos los valores contenido en la enumeración; para esto, se usa la estructura repetitiva For Each para recorrer por todos los elementos y para obtener su índice se usa en método GetValues:

```
For Each a In System.Enum.GetValues(GetType(tiposAutos))
   MessageBox.Show(a)
Next
```

Aquí, un caso práctico en el uso de las enumeraciones. El objetivo es llenar el cuadro combinado de tipos de automóviles y que al seleccionar uno de ellos muestre el precio de alquiler:



El código de solución:

```
Public Class frmEnumeracion
    Private Sub frmEnumeracion_Load(…) Handles MyBase.Load
        'Llenar el cuadro combinado desde la enumeración
        For Each a In System.Enum.GetNames(GetType(tiposAutos))
            cboTipo.Items.Add(a.ToString)
        Next
    End Sub

    Private Sub cboTipo_SelectedIndexChanged(…) Handles cboTipo.Selecte-
dIndexChanged
        'Capturando el tipo de auto desde el cuadro combinado
        Dim tipo As String = cboTipo.SelectedItem

        'Determinar el precio según el tipo de auto
        Dim precio As Double = 0
        If tipo = "Auto" Then precio = 3.5
        If tipo = "Camioneta" Then precio = 4
        If tipo = "Camion" Then precio = 5.5
        If tipo = "Omnibus" Then precio = 10
        If tipo = "Motocicleta" Then precio = 2

        'Mostrar el precio en el control Label
        lblPrecio.Text = precio.ToString("C")
    End Sub
End Class

'Declaración de la enumeración
Enum tiposAutos As Integer
    Auto
    Camioneta
    Camion
    Omnibus
    Motocicleta
End Enum
```

3.8 La clase MessageBox

Representa una ventana de salida mensaje o cuadro de dialogo de tipo mensaje, dicha ventana se caracteriza por bloquear el proceso de una aplicación hasta que el usuario no cierre el mensaje, por esta característica es llamada ventana modal. A continuación, se verá en qué ocasión se puede usar una ventana de dialogo de mensaje, por ejemplo:

- Mostrar un mensaje al usuario, el cual puede ser informativo, de error o de advertencia.
- Enviar una pregunta al usuario y que este responda por medio de botones dentro del mensaje.

Mensaje simple

Las características de un mensaje simple es que muestra solo un bloque de texto; aquí se debe tener en cuenta que no se muestra título de la ventana y que los botones del cuadro son de forma predeterminada, para este caso solo muestra el botón **Aceptar**. Su formato es:

```
MessageBox.Show("Mensaje básico")
```

Mensaje con título en la ventana

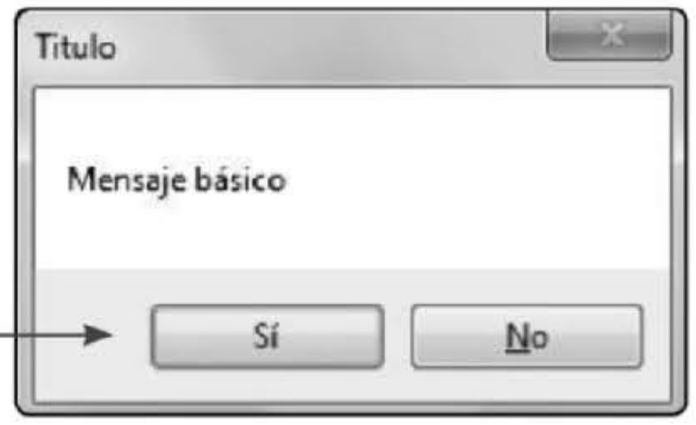
Un cuadro de mensaje con título de ventana nos indica de dónde proviene el mensaje, de tal forma que se recomienda que todo cuadro de mensaje debe tener un título identificativo. Su formato es:

```
MessageBox.Show("Mensaje básico", "TituloVentana")
```

Mensaje con botones

La especificación de botones dentro del cuadro de mensaje es muy importante, ya que a través de este medio el usuario podrá determinar ciertas acciones con respecto al cuadro de mensaje. Por ejemplo, se puede tomar decisiones de salida de la aplicación o aceptar el registro de un empleado. Su formato es:

```
MessageBox.Show("Mensaje básico","Titulo",
                MessageBoxButtons.Boton)
```



Mensaje con botones e íconos

Los íconos dentro de un cuadro de mensaje solo nos ayudan a una mejor presentación del cuadro.

Asimismo, se incorpora un sonido por cada tipo de ícono seleccionado. Su formato es:

```
MessageBox.Show("Mensaje básico","Titulo",
MessageBoxButtons.Boton,
MessageBoxIcon.Icono);
```

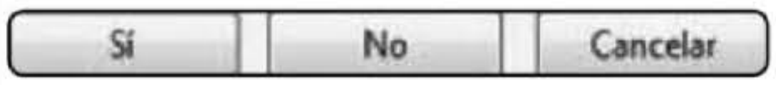
Mensaje con respuesta de usuario

```
Dim r As DialogResult = MessageBox.Show("Mensaje", "Titulo",
                        MessageBoxButtons.Boton,
 MessageBoxIcon.Icono)
```

Donde:

- **Mensaje**: Es el mensaje o comentario que se muestra al usuario.
- **Título**: Es un texto que se muestra en la barra de título del cuadro de dialogo de mensaje.

- **MessageBoxButtons**: Representa a un conjunto de botones permitidos en el cuadro de dialogo de mensaje. Aquí, la lista de botones permitidos:

Botón	Descripción	Imagen
OK	Representa al botón **Aceptar**, el cual también es considerado como el botón predeterminado en el cuadro de dialogo de mensaje.	
OKCancel	Representa a los botones **Aceptar** y **Cancelar** dentro del cuadro de dialogo de mensaje.	
YesNo	Representa a los botones **Sí** y **No** dentro del cuadro de dialogo de mensaje. Para representar la selección de un botón: • `DialogResult.Yes` • `DialogResult.No`	
YesNoCancel	Representa a los botones **Sí**, **No** y **Cancelar**, del cuadro de dialogo de mensaje.	

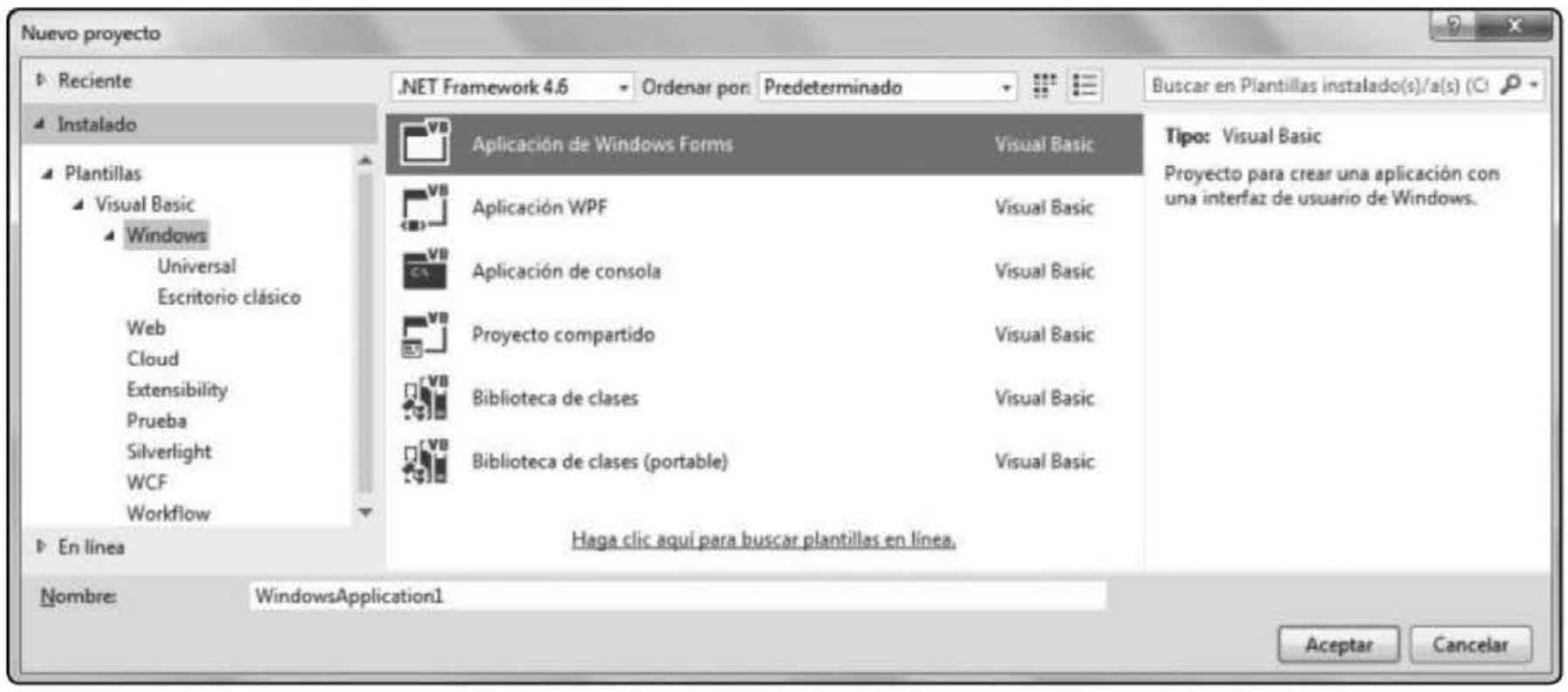
- **MessageBoxIcon**: Representa a un conjunto de iconos permitidos en el cuadro de diálogo de mensaje. Aquí, la lista de íconos permitidos:

Icono	Descripción	Imagen
Error	Se muestra un símbolo que consiste en una X blanca en un círculo con fondo rojo, dentro del cuadro de diálogo de mensaje.	
Exclamation	Se muestra un símbolo que consiste en un signo de exclamación en un triángulo con fondo amarillo, dentro del cuadro de diálogo de mensaje.	
Information	Se muestra un símbolo que consiste en una letra «i» minúscula en un círculo dentro del cuadro de dialogo de mensaje.	
None	No muestra ningún símbolo.	
Question	Se muestra un símbolo que consiste en un signo de interrogación en un círculo dentro del cuadro de dialogo de mensaje.	

3.9 Desarrollo de aplicaciones secuenciales

Una aplicación secuencial se caracteriza por seguir una serie de sentencias una a continuación de la otra como su nombre lo indica en secuencia, de forma top-down (arriba-abajo), donde cada sentencia es leída e interpretada una a continuación de la otra:

- **Archivo > Nuevo proyecto**
- Seleccione la plantilla **Visual Basic > Windows > Aplicación de Windows Forms**



- Coloque un nombre y seleccione **Aceptar**.

Caso desarrollado 1 Declaración de variable básica - Pago de vendedores

Una empresa de venta de tarjetas de crédito paga a sus vendedores un sueldo básico mensual de $500.00. Asimismo, el sueldo bruto del vendedor es igual al sueldo básico más una comisión, esta comisión se regula bajo el 12% del monto total vendido en un determinado mes. Por otro lado, todo vendedor se somete a un descuento de ley que asciende al 9.75%. Implemente una aplicación que permita calcular el monto de comisión, sueldo bruto, descuento y el sueldo neto de un vendedor en un determinado mes.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Usar declaración de variables básicas.
- Debe ingresar el nombre del vendedor y el monto vendido en el mes.
- El monto básico asciende a $300.00, el cual debe ser declarado como constante.
- El monto de comisión resulta del 12 % del monto vendido.
- El monto bruto resulta del acumulado entre la comisión y el monto básico.
- El monto de descuento resulta del 9.75 % del monto bruto.
- El monto neto resulta de la diferencia entre el monto bruto y el descuento.
- Mostrar todos los montos con el formato de moneda.
- El botón **Registrar** es el encargado de mostrar la información resultante y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón **Salir**.
- Mostrar el total de vendedores registrados usando variables globales.
- El registro del pago de vendedores deberá mostrarse en el control ListView.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmPagoVendedores btnAceptar btnCancelar PAGO DE VENDEDORES
Label1	Text	PAGO DE VENDEDORES
Label2	Text	VENDEDOR
Label3	Text	MONTO VENDIDO
Label4	Text	TOTAL DE VENDEDORES REGISTRADOS
Label5	(Name)	lblTotal
TextBox1	(Name)	txtVendedor

TextBox2	(Name)	txtMonto
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvPago True

El script de la solución es:

```
Public Class frmPagoVendedores
    'Declarando n como variable GLOBAL
    Dim n As Integer

    'Declarar a básico como constante
    Const básico As Double = 500

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'Generar numero de vendedores
        n = n + 1

        'Capturando los datos del formulario
        Dim vendedor As String = txtVendedor.Text
        Dim montoVendido As Double = txtMonto.Text

        'Realizando los cálculos
        Dim comisión As Double = montoVendido * 12 / 100
        Dim bruto As Double = básico + comisión
        Dim descuento As Double = bruto * 9.75 / 100
        Dim neto As Double = bruto - descuento

        'Imprimir los resultados
        Dim fila As New ListViewItem(n.ToString())
        fila.SubItems.Add(vendedor)
        fila.SubItems.Add(montoVendido.ToString("0.00"))
        fila.SubItems.Add(comision.ToString("0.00"))
        fila.SubItems.Add(bruto.ToString("0.00"))
        fila.SubItems.Add(descuento.ToString("0.00"))
        fila.SubItems.Add(neto.ToString("0.00"))
        lvPago.Items.Add(fila)

        'Mostrando el total de vendedores registrados
        lblTotal.Text = lvPago.Items.Count.ToString("0.00")

        'Limpiando los controles
        btnCancelar_Click(sender, e)
    End Sub
```

```
    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtVendedor.Clear()
        txtMonto.Clear()
        txtVendedor.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Ahora, una breve explicación de los códigos mostrados:

La variable N tiene la misión de controlar cuántos vendedores se van registrando; es por ese motivo que se declara como global. Asimismo, la constante básico se declara en la sección global con un valor inicial de 500.

En el botón **Registrar**, se inicia por aumentar el valor inicial de N en uno, ya que se va a registrar un nuevo vendedor. Para imprimir los resultados dentro del control ListView, se usa el siguiente código:

```
Dim fila As New ListViewItem(n.ToString())
fila.SubItems.Add(vendedor)
fila.SubItems.Add(montoVendido.ToString("0.00"))
fila.SubItems.Add(comision.ToString("0.00"))
fila.SubItems.Add(bruto.ToString("0.00"))
fila.SubItems.Add(descuento.ToString("0.00"))
fila.SubItems.Add(neto.ToString("0.00"))
lvPago.Items.Add(fila)
```

Se declara el objeto fila, el cual se encargará de enviar la información línea por línea en el control ListView. Al declarar el objeto, se envía como parámetro del método constructor el valor de n; hay que tener en cuenta que todo valor numérico debe ser convertido en cadena para su envío al objeto fila. El método SubItems.Add permite agregar los valores a imprimir en el control ListView. Se debe tener en cuenta el orden, ya que secuencialmente define la forma de visualización de los datos en columnas. Finalmente, todos los datos han sido enviados al objeto fila, que deberá ser agregado al control ListView mediante el método Items.Add.

La sentencia **btnCancelar_Click(sender, e)** ejecuta el contenido implementando dentro del botón **Cancelar**, el cual contiene codigo que permite limpiar todos los controles de ingreso.

Caso desarrollado 2 — Declaración por literales de tipo identificador - Venta de productos

Una tienda de venta de productos de seguridad necesita controlar la venta de su nuevo producto (cámara DOMO), cuyo costo unitario de promoción es de $ 65.50. Asimismo, ofrece una oferta del 8.65 % sobre el importe de la compra. Adicionalmente, la tienda obsequia un USB de 8GB por cada cinco productos adquiridos como una promoción por nuevo producto. Implemente una aplicación que determine el monto de la compra, descuento y el importe total a pagar por la compra de cierta cantidad de productos.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Al iniciar la aplicación se debe mostrar la fecha actual del registro de venta, usando el evento load del formulario.
- Se debe ingresar solo la cantidad comprada del nuevo producto por cada cliente.
- El precio del producto es un valor constante que asciende a $65.50.
- Mostrar la imagen del producto en promoción de forma estática usando el control PictureBox.
- El importe de compra resulta del producto entre la cantidad seleccionada y el precio constante del producto.
- El descuento se obtiene aplicando el 8.65 % al importe de compra.
- El importe total se obtiene de la diferencia entre el importe de compra y el descuento.
- Mostrar todos los montos con el formato de moneda.
- El botón **Registrar** es el encargado de mostrar la información en el control ListView y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Mostrar el control de la ventas en un control ListView.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmVenta btnAceptar btnCancelar CONTROL DE VENTA – NUEVO PRODUCTO
Label1	Text	CONTROL DE VENTA – NUEVO PRODUCTO
Label2	Text	CANTIDAD
Label3	Text	FECHA DE REGISTRO
Label4	Text	lblFecha
Label4	(Name)	$ 65.50
TextBox1	(Name)	txtCantidad
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
PictureBox1	Image SizeMode	domo.jpg StretchImage
ListView1	(Name)	lvVenta

El script de la solución es:

```
Public Class frmVenta
    'Declarando el precio como constante
    Const precio@ = 65.5
    'Declarando el contador de ventas como GLOBAL
    Dim n As Integer

    Private Sub frmVenta_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Mostrar la fecha actual
        lblFecha.Text = DateAndTime.Now.ToShortDateString()
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'Aumentando el contador de ventas
        n = n + 1
```

```
        'Capturando la cantidad
        Dim cantidad% = txtCantidad.Text

        'Realizando cálculos
        Dim importeCompra@ = cantidad * precio
        Dim descuento@ = importeCompra * 8.65 / 100
        Dim importeTotal@ = importeCompra - descuento
        Dim totalObsequios% = cantidad \ 5

        'Imprimir resultados
        Dim fila As New ListViewItem(n.ToString)
        fila.SubItems.Add(cantidad.ToString("0.00"))
        fila.SubItems.Add(importeCompra.ToString("0.00"))
        fila.SubItems.Add(descuento.ToString("0.00"))
        fila.SubItems.Add(importeTotal.ToString("0.00"))
        fila.SubItems.Add(totalObsequios.ToString("0.00"))
        lvVenta.Items.Add(fila)

        'Limpiando el control para un nuevo ingreso
        btnCancelar_Click(sender, e)
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtCantidad.Clear()
        txtCantidad.Focus()
    End Sub
End Class
```

A continuación, una breve explicación de los códigos mostrados:

Se inicia por realizar una declaración global para la constante Const precio@=65.5 usando literal @, que indica que la variable es de tipo Decimal, ya que el caso indica dicha constante. En el evento load del formulario se muestra la fecha actual con la sentencia **lblFecha.Text = DateAndTime.Now.ToShortDateString()** donde Now captura la fecha actual y ToShortDateString muestra la fecha en formato corto y convertido en cadena para su impresión en el control lblFecha.

Caso desarrollado 3 Declaración variante - Cajero automático

Una nueva entidad de ahorro y crédito ubicada en la ciudad de Lima desea implementar un cajero automático inteligente, de tal forma que, al solicitar un retiro de dinero, nos devuelva billetes de mayor a menor corte. Implemente una aplicación para que a partir de un monto de retiro nos devuelva una cantidad de billetes de $100.00, $50.00, $20.00, $10.00, $5.00, $2.00 y $1.00.

Por ejemplo, si el monto de retiro es $3688.00 puede descomponerse en 36 billetes de $100, más 1 billete de $50, más 1 billete de $20, más 1 billete de $10, más 1 billete de $5, más 1 billete de $2 y más 1 billete de $1.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar el monto de retiro para procesar la cantidad de billetes correspondientes a billetes de 100, 50, 20, 10, 5, 2 y 1.
- Asociar el botón **Procesar** a la tecla <Enter>.
- Mostrar la cantidad de billetes en un control ListBox.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmCajero btnProcesar btnCancelar CAJERO AUTOMATICO - SIMULACION DE RETIRO

Label1	Text	CAJERO AUTOMATICO - SIMULACION DE RETIRO
Label2	Text	MONTO DE RETIRO
TextBox1	(Name)	txtMonto
Button1	(Name) Text	btnProcesar PROCESAR
Button2	(Name) Text	btnSalir SALIR
Button3	(Name) Text	btnCancelar CANCELAR
List1	(Name)	lstR

El script de la solución es:

```
Public Class frmCajero
    Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
        'Capturando el monto
        Dim monto As Object = txtMonto.Text

        'Determinando los billetes
        Dim b100 As Object = monto \ 100
        Dim b50 As Object = (monto Mod 100) \ 50
        Dim b20 As Object = ((monto Mod 100) Mod 50) \ 20
        Dim b10 As Object = (((monto Mod 100) Mod 50) Mod 20) \ 10

       Dim b5 As Object = ((((monto Mod 100) Mod 50) Mod 20) Mod 10) \ 5
       Dim b2 As Object = (((((monto Mod 100) Mod 50) Mod 20) Mod 10) Mod 5) \ 2
       Dim b1 As Object = (((((monto Mod 100) Mod 50) Mod 20) Mod 10) Mod 5) Mod 2

        'Imprimir los resultados
        lstR.Items.Clear()
        lstR.Items.Add("** RESUMEN DE BILLETES **")
        lstR.Items.Add("BILLETES DE $100: " & b100.ToString)
        lstR.Items.Add("BILLETES DE $50: " & b50.ToString)
        lstR.Items.Add("BILLETES DE $20: " & b20.ToString)
        lstR.Items.Add("BILLETES DE $10: " & b10.ToString)
        lstR.Items.Add("BILLETES DE $5: " & b5.ToString)
        lstR.Items.Add("BILLETES DE $2: " & b2.ToString)
        lstR.Items.Add("BILLETES DE $1: " & b1.ToString)
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtMonto.Clear()
        lstR.Items.Clear()
        txtMonto.Focus()
    End Sub
```

```
    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then Me.Close()

    End Sub
End Class
```

A continuación, una breve explicación de los códigos mostrados:

En el botón **Procesar** se usa la declaración como tipo Object, el cual permite adaptar el contenido al tipo de datos. Si el monto ingresado es un valor decimal dicha variable lo declarará de la misma manera; se debe tener en cuenta que este tipo de datos se adapta al tipo del valor contenido.

Caso desarrollado 4 Conversión por palabra clave - Temperatura

Un centro médico desea implementar consultas vía telefónica sobre el estado de los pacientes, para lo cual solicita la temperatura en grados centígrados de cada paciente y tiene que realizar la conversión en Fahrenheit, Rankine y Kelvin respectivamente, ya que los manuales de asistencia se encuentran en estos tres tipos de estados. Implemente una aplicación que, ingresada la cantidad en grado centígrado, lo muestre en grados Fahrenheit (F), grados Kelvin (K) y grados Rankine(R). Se deben usar las siguientes fórmulas:

Fahrenheit = 9 x Centígrados / 5 + 32
Kelvin = Rankine – 187
Rankine = Centígrados + 460

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar la temperatura en grados centígrados.
- El botón **Calcular** es el encargado de mostrar la información resultante de las conversiones y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Mostrar el resumen de las conversiones en un control ListBox.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmTemperatura btnCalcular btnCancelar CONVERSIÓN DE TEMPERATURA
Label1	Text	CONVERSIÓN DE TEMPERATURA
Label2	Text	GRADOS CENTIGRADOS
TextBox1	(Name)	txtCentigrados
Button1	(Name) Text	btnCalcular CALCULAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
List1	(Name)	lstR

El script de la solución es:

```
Public Class frmTemperatura
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
        'Capturando los grados centígrados
        Dim centígrados As Double
        centígrados = CDbl(txtCentigrados.Text)

        'Calculando los grados
        Dim fahrenheit As Double, rankine As Double, kelvin As Double
        fahrenheit = (9 * centígrados) / 5 + 32
```

```
        rankine = centígrados + 460
        kelvin = rankine - 187

        'Imprimir resultados
        lstR.Items.Clear()
        lstR.Items.Add("**RESUMEN DE TEMPERATURAS**")
        lstR.Items.Add("GRADOS CENTIGRADOS: " & CStr(centígrados))
        lstR.Items.Add("----------------------------------------")
        lstR.Items.Add("GRADOS FAHRENHEIT: " & CStr(fahrenheit))
        lstR.Items.Add("GRADOS RANKINE: " & CStr(rankine))
        lstR.Items.Add("GRADOS KELVIN: " & CStr(kelvin))
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtCentigrados.Clear()
        lstR.Items.Clear()
        txtCentigrados.Focus()
    End Sub

Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then Me.Close()
    End Sub
End Class
```

A continuación, una breve explicación de los códigos mostrados:

En el botón **Calcular** se declara la variable Centígrados para registrar los grados ingresados por el usuario. En vista que este valor es ingresado en la caja de texto y debe ser enviado como valor a la variable Centígrados, se generaría un error si se envía directamente, ya que habría una anomalía en la compatibilidad de datos por ser cadena y doublé; por ese motivo, se usa la función de conversión CDBL, el cual convierte un valor de cadena que ingresamos por la caja de texto en un valor numérico de tipo double.

Caso desarrollado 5 Conversión usando CType - Donación

Un policlínico de la región ha recibido una donación especial en el mes de diciembre, para lo cual el administrador ha decidido realizar una repartición en porcentajes según el nivel de asistencias de las áreas, tal como se muestra en la siguiente lista:

- 30 % del total para el área de Medicina General.
- 25 % del total para el área de Ginecología.
- 50 % del monto asignado a Medicina General para el área de Odontología.
- Y el resto de la donación al área de Pediatría.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar el monto asignado a la donación.
- El botón **Calcular** es el encargado de mostrar la información resultante y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla Enter.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Mostrar los montos asignados a las áreas en controles de tipo Label.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmDonacion btnCalcular btnCancelar CONTROL DE REPARTICIÓN DE DONACIÓN
Label1	Text	CONTROL DE REPARTICIÓN DE DONACIÓN
Label2	Text	MONTO DONADO
Label3	Text	MEDICINA GENERAL
Label4	Text	GINECOLOGÍA
Label5	Text	PEDIATRÍA
Label6	(Name)	lblMedicina
Label7	(Name)	lblGinecologia
Label8	(Name)	lblOdontologia
Label9	(Name)	lblPediatria
TextBox1	(Name)	txtDonacion
Button1	(Name) Text	btnCalcular CALCULAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
GroupBox1	Text	Repartición

El script de la solución es:

```
Public Class frmDonacion
    Private Sub btnCalcular_Click(sender As Object, e As EventArgs)
        'Obtener valores
        Dim donación As Double = CType(txtDonacion.Text, Double)

        'Realizar cálculos
        Dim medicina As Double = 30 / 100 * donación
```

```
        Dim ginecologia As Double = 25 / 100 * donación
        Dim odontologia As Double = 50 / 100 * medicina
        Dim pediatria As Double = donación - (medicina + ginecologia + odontologia)

        'Imprimir resultados
        lblMedicina.Text = "$ " & CType(medicina, String)
        lblGinecologia.Text = "$ " & CType(ginecologia, String)
        lblOdontologia.Text = "$ " & CType(odontologia, String)
        lblPediatria.Text = "$ " & CType(pediatria, String)
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtDonacion.Clear()
        lblMedicina.Text = ""
        lblGinecologia.Text = ""
        lblOdontologia.Text = ""
        lblPediatria.Text = ""
        txtDonacion.Focus()
    End Sub

     Private Sub btnSalir_Click(sender As Object, e As EventArgs)
Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then Me.Close()
    End Sub
End Class
```

A continuación, una breve explicación de los códigos mostrados:

Dentro del botón **Calcular** se declara la variable donación encargará de registrar el monto de donación registrada por el usuario, dicho valor será ingresado en un caja de texto y esta debe ser parseado de alguna forma para evitar errores de compatibilidad; para este caso, se usa la función CType, el cual permite convertir un valor en un tipo de datos especifico, los parámetros indican que primero se debe colocar el control que tiene el valor a convertir y luego se especifica el tipo de datos.

Caso desarrollado 6 Método Parse - Distancia

Una tienda de venta de automóviles de diferentes marcas ha realizado una prueba de las distancias que puede recorrer sus autos; esta prueba se realiza en tres tramos, el primer tramo se registra en kilómetros, mientras que el segundo tramo se registra en pies, y el último tramo se registra en millas. Implemente una aplicación que determine la longitud total recorrida por cada automóvil en prueba, en metros y en yardas. Considere los siguientes factores de conversión:

1 metro = 3.2808 pies
1 yarda = 3 pies
1 kilómetro = 1000 metros
1milla = 1609 metros

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar la marca del automóvil, así como los tramos uno, dos y tres, teniendo en cuenta que cada tramo tiene diferentes medidas de longitud.
- El botón **Procesar** es el encargado de mostrar la información en el control ListView y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla <Enter>.

- El botón **Procesar** es el encargado de mostrar la información en el control ListView y, a la vez, debe ser considerado como predeterminado para poder activarlo con la tecla <Enter>.

- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmDistancia btnProcesar btnCancelar CONTROL DE DISTANCIAS
Label1	Text	CONTROL DE DISTANCIAS
Label2	Text	MARCA DE AUTOMÓVIL
Label3	Text	TRAMO 1 - KM
Label4	Text	TRAMO 2 - PIES
Label5	Text	TRAMO 3 - MILLAS
GroupBox1	Text	Distancia recorrida
TextBox1	(Name)	txtMarca
TextBox2	(Name)	txtKM
TextBox3	(Name)	txtPies
TextBox4	(Name)	txtMillas
Button1	(Name) Text	btnProcesar PROCESAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLine Columns View	lvControl True Agregue 3 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmDistancia
    Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
        'Capturando información
        Dim marca As String = txtMarca.Text
        Dim km As Double = Double.Parse(txtKM.Text)
        Dim pies As Double = Double.Parse(txtPies.Text)
        Dim millas As Double = Double.Parse(txtMillas.Text)

        'Realizando cálculos
        Dim metros As Double = km * 1000 + pies / 3.2808 + millas * 1609
        Dim yardas As Double = (metros * 3.2808) / 3

        'Imprimir resultados
        Dim fila As New ListViewItem(marca)
        fila.SubItems.Add(metros.ToString("0.00"))
        fila.SubItems.Add(yardas.ToString("0.00"))
        lvControl.Items.Add(fila)
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtMarca.Clear()
        txtKM.Clear()
        txtMillas.Clear()
        txtPies.Clear()
        txtMarca.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then Me.Close()
    End Sub
End Class
```

Caso desarrollado 7 Manejo de cadenas - Generación de códigos y correos corporativos

Una empresa de diseño gráfico profesional recluta empleados cada año con el fin de ampliar el recurso humano en la empresa y, en base al informe del área de recursos humanos, reciben información de los empleados nuevos, quienes, según política de la empresa, antes de iniciar sus labores deben contar con un código de usuario para el registro de asistencia y otros procesos dentro de la organización, así como su correo institucional.

Implemente una aplicación que permita registrar los nombres, apellido paterno, apellido materno, categoría y su fecha de nacimiento de los nuevos empleados y genere su nuevo código y correo corporativo. Asimismo, tenga en cuenta que la información de los nuevos empleados proviene del área de recursos humanos.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe ingresar el nombre completo, apellido paterno, apellido materno, categoría y fecha de nacimiento del empleado.
- Las categorías de los empleados se dan en base al monto de sueldo que son A, B, C y D.
- La fecha de nacimiento debe seleccionarse a partir del control DateTimePicker.
- El nuevo código del empleado se encuentra formado por la primera letra de su nombre y las cuatro primeras de su apellido paterno.
- El correo electrónico del empleado se encuentra formado por la primera letra de su nombre, las cuatro primeras letras de su apellido paterno, guion bajo, los dos últimos dígitos de su año de nacimiento y, finalmente, «@miempresa.com.pe».
- El botón **Registrar** es el encargado de enviar la información de los nuevos empleados al control ListView y a la vez debe ser considerado como predeterminado para poder activarlo con la tecla Enter.

- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla Esc.
- El registro de los nuevos empleados debe mostrarse en un control ListView.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmRegistroEmpleado btnRegistrar btnCancelar CONTROL DE REGISTRO DE NUEVOS EMPLEADOS
Label1	Text	CONTROL DE REGISTRO DE NUEVOS EMPLEADOS
Label2	Text	NOMBRES
Label3	Text	PATERNO
Label4	Text	MATERNO
Label5	Text	CATEGORÍA
Label6	Text	FECHA NAC.
TextBox1	(Name)	txtNombres
TextBox2	(Name)	txtPaterno
TextBox3	(Name)	txtMaterno
ComboBox1	(Name)	cboCategoria
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
DateTimePicker1	(Name)	dtFecha
ListView1	(Name) GridLine Columns View	lvEmpleados True Agregue 3 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmRegistroEmpleado
    Private Sub frmRegistroEmpleado_Load(sender As Object, e As EventArgs)
        cboCategoria.Items.Add("A")
        cboCategoria.Items.Add("B")
        cboCategoria.Items.Add("C")
        cboCategoria.Items.Add("D")
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'Recuperación de la información
        Dim nombre As String = txtNombres.Text
        Dim paterno As String = txtPaterno.Text
        Dim materno As String = txtMaterno.Text
        Dim categoria As String = cboCategoria.Text
        Dim fechaNac As DateTime = dtFecha.Value

        'Realizar los cálculos
       Dim correo As String = Strings.Left(nombre, 1) & Strings.Left(paterno, 4)
                           & "_" & Strings.Right(CType(fechaNac.Year, String), 2)
                                 & "@miempresa.com.pe"
       Dim codigo As String = Strings.Left(nombre, 1) & Strings.Left(paterno, 4)

        'Imprimir los resultados
       Dim fila As New ListViewItem(nombre & " " & paterno & " " & materno)
        fila.SubItems.Add(Strings.UCase(codigo))
        fila.SubItems.Add(Strings.LCase(correo))
        lvEmpleados.Items.Add(fila)
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtNombres.Clear()
        txtPaterno.Clear()
        txtMaterno.Clear()
        cboCategoria.SelectedIndex = -1
        txtNombres.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then Me.Close()
    End Sub
End Class
```

Capítulo 4

Estructuras de decisión

Capacidad terminal:

- Implementar aplicaciones que permitan al usuario decidir por una sentencia de condición adecuada según el proceso de negocio.
- Reconocer los formatos de las principales sentencias de decisión para Visual Basic 2015.
- Implementar aplicaciones que permitan reconocer la forma de trabajar de cada una de las sentencias expuestas en el capítulo.

Contenido

- Operadores para estructuras de decisión
- Estructura condicional If
- Sentencia de condición múltiple simple Choose
- Sentencia de condición múltiple
- Desarrollo de aplicaciones condicionales

4.1 Operadores para estructuras de decisión

4.1.1 Operadores de relación

Los operadores de relación permiten implementar sentencias de tipo condicional en la cual dentro de una condición se debe comparar una variable y un valor. A continuación, los operadores válidos para una condición:

Operador	Definición	Ejemplo

>	Mayor que	Determinar si el precio de un producto es superior a $500.00.
		`Dim precio As Double = 1500`
		`If (precio > 500) Then`

>=	Mayor o igual que	Determinar si el pago de un trabajador es superior o igual a $ 2500.00.
		`Dim pago As Double = 3500`
		`If (pago >= 2500) Then`

<	Menor que	Determinar si el precio de un producto es inferior a $500.00.
		`Dim precio As Double = 1500`
		`If (precio < 500) Then`

<=	Menor o igual que	Determinar si el pago de un trabajador es inferior o igual a $2500.00.
		`Dim pago As Double = 3500`
		`If (pago <= 2500) Then`

=	Igual	Comparar si la categoría de un empleado es Operario.
		`Dim categoria As String = "Jefe"`
		`If (categoria = "Operario") Then`

<>	Diferente o no es igual	Determinar si la condición de un alumno es Aprobado.
		`Dim condicion As String = "Aprobado"`
		`If (condicion <> "Desaprobado") Then`

4.1.2 Operadores lógicos

Los operadores lógicos cumplen un papel importante en la implementación de la condición en una estructura selectiva pues permitirá unir dos o más condiciones; de esta forma, la condición contará con más criterios de evaluación. Hay que tener en cuenta que el resultado adecuado dependerá del orden de prioridad de los operadores usados en la condición.

Operador	Definición	Ejemplo

And	Representa la Y lógica, la cual evalúa que ambas condiciones sean de tipo true (verdadero) para realizar una determinada acción.	Determinar si los trabajadores de una empresa perciben un ingreso entre $2000.00 y $2500.00.
		`Dim ingreso As Double = 5500`
		`If (ingreso >= 2000 And ingreso <= 2500) Then`

Or	Representa la O lógica, la cual ejecuta una determinada acción solo si una de las condiciones es de tipo true (verdadero).	Determinar si los trabajadores de una empresa tienen la categoría Jefe o Gerente.
		`Dim categoria As String = "Jefe"`
		`If (categoria = "Jefe" Or`
		`  categoria = "Gerente") Then`

Not	Representa la negación del resultado de una condición.	Determinar si los empleados de una empresa no son de tipo Jefe ni Gerente.
		`Dim categoria As String = "Jefe"`
		`If (Not (categoria = "Jefe" Or`
		`  categoria = "Gerente")) Then`

4.2 Estructura condicional If

Una aplicación de tipo secuencial se caracteriza por seguir una sentencia después de otra. Cuando se implementa una estructura de decisión las sentencias se ejecutan dependiendo del valor resultante en la condicional.

```
① Dim tipoFondo As String
  Dim basico As Double = 2500

②

③ tipoFondo = "Integra"
  Dim descuento As Double = 0

  If tipoFondo = "Integra" Then          True
     descuento = 0.125 * basico
False Else
     descuento = 0
  End If
     ④
  MessageBox.Show("El descuento asignado es: " & descuento)
```

En el código se puede observar que en el punto uno, dos, tres y cuatro son sentencias secuenciales y cuando llega a la condicional If se especifican dos caminos que puede tomar dependiendo del valor resultante de la condición; si es true se hace un descuento del 12.5 % al monto básico, pero si la condición resulta false al descuento se le asigna directamente el valor cero.

4.2.1 Condición If simple

Se le denomina simple cuando usa una sola sección de la sentencia condicionada; si la condición evaluada resulta True (verdadero) ejecuta una o más sentencias especificadas en ese bloque, caso contrario no ejecuta sentencia alguna y continúa con la siguiente instrucción de la aplicación llamada instrucción posterior.

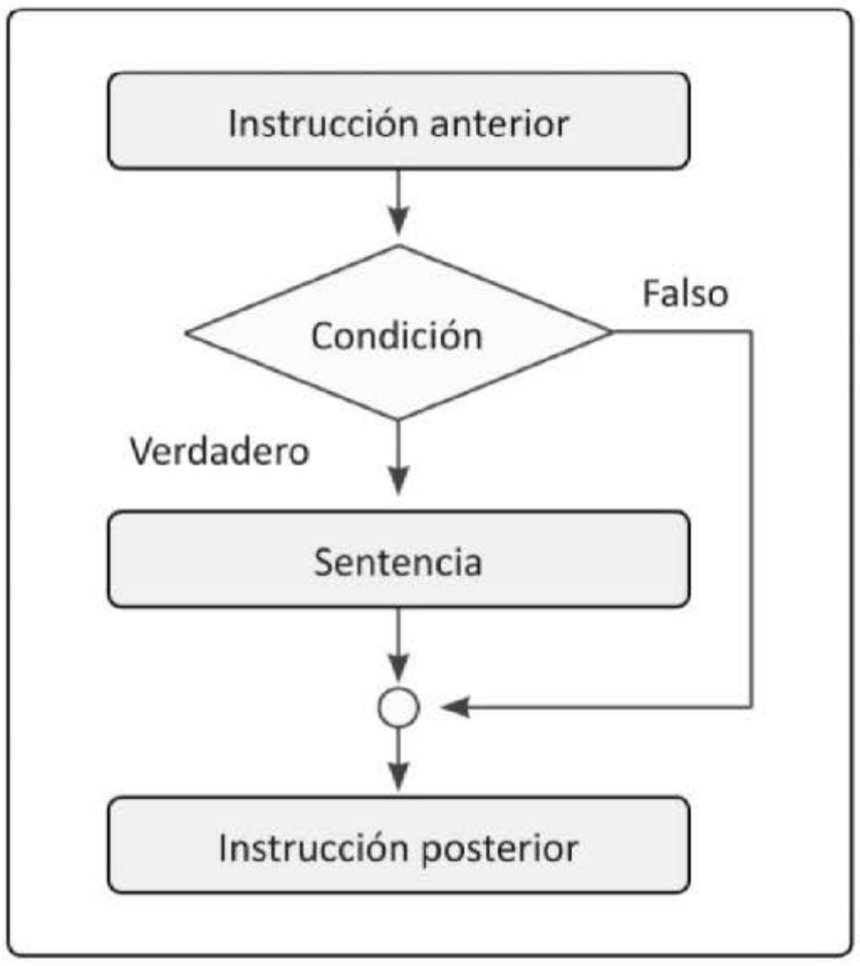
Ver los siguientes formatos:

Formato 1:

```
If (condicion) Then sentencia
```

Formato 2:

```
If (condicion) Then
sentencia1
sentencia2
End if
```



Donde:

- **Condición**: Es la expresión que determina las acciones a realizar si el resultado de esta devuelve true (verdadero). Asimismo, se debe considerar que los operadores lógicos y relacionales nos permitirán implementar la condición.
- **Sentencia**: Es la expresión que se ejecutará si y solo si la condición evaluada es verdadera. Se debe tener en cuenta de que si se trata de una sola sentencia, no será necesario el uso de la cláusula End If, como se observa en el formato uno. En cambio, si implementa más de dos sentencias, obligatoriamente debe finalizar con la cláusula End If, como se visualiza en el formato dos.

Implementación de sentencias If simple

- Determinar el nombre de la categoría de un producto según la siguiente tabla:

Categoría	Descripción
A	Lácteos
B	Bebidas
C	Abarrotes
D	Golosinas

```
Dim categoria, descripcion As String
categoria = "A"

If (categoria = "A") Then descripcion = "Lácteos"
If (categoria = "B") Then descripcion = "Bebidas"
If (categoria = "C") Then descripcion = "Abarrotes"
If (categoria = "D") Then descripcion = "Golosinas"
```

- Aplicar la siguiente tabla de descuentos con respecto al pago bruto de los trabajadores en una empresa:

Pago bruto	Descuento
Menor o igual a $750.00	7.50 %
Superior a $750.00 e inferior a $1400.00	12.50 %
Superior a $1400.00 e inferior a $2100.00	14.50 %
Superior a $2100.00	16.50 %

```
Dim sueldo, descuento As Double
sueldo = 1750.0

If (sueldo <= 750) Then descuento = sueldo * (7.5/100)
If (sueldo > 750 And sueldo <= 1400) Then descuento = sueldo * (12.5 / 100)
If (sueldo > 1400 And sueldo <= 2100) Then descuento = sueldo * (14.5/100)
If (sueldo > 2100) Then descuento = sueldo * (16.5 / 100)
```

- Aplicar la siguiente tabla de asignación de costo, hora y descuentos con respecto a la categoría de los empleados en una empresa:

Categoría	Costo hora	Descuento
Jefe	$50.00	18 %
Administrativo	$30.00	15 %
Apoyo	$20.00	12 %
Operario	$7.50	10 %

```
Dim categoria As String
        Dim costoHora, descuento As Double

        If (categoria = "Jefe") Then
              costoHora = 50
              descuento = 0.18
        End If
        If (categoria = "Administrativo") Then
              costoHora = 30
              descuento = 0.15
        End If
        If (categoria = "Apoyo") Then
              costoHora = 20
              descuento = 0.12
        End If
        If (categoria = "Operario") Then
              costoHora = 7.5
              descuento = 0.1
End If
```

Consideraciones finales de las sentencias If simple

- Para implementar una condición en una sentencia If, use la combinación de variables y operadores.
- Por regla general, toda implementación de condición debe tener por lo menos una sentencia de acción, la cual puede ser una expresión u otra implementación de If.
- Solo si la condición resulta verdadera ejecutará las sentencias implementadas.
- Si la condición resulta falsa no ejecutará ninguna sentencia y la aplicación continuará a la siguiente sentencia especificada después del If.
- Cuando hay una sola sentencia no será necesario el uso de la cláusula final End If.
- Cuando se necesite realizar una comparación múltiple puede colocar varios bloques If uno dentro de otro, generando un anidamiento entre sentencias del mismo tipo.

4.2.2 Condición If doble

Se le denomina así cuando se usa ambas secciones de la sentencia selectiva If. Cuando la condición evaluada resulta verdadera, se ejecuta sentencias implementadas en dicha sección; en cambio, si la condición resulta falsa se ejecuta otro bloque de sentencias. Se debe tener en cuenta que cuando se ejecuta un bloque de sentencias de una sección ya no se ejecuta el otro bloque y la aplicación continúa con el código especificado.

Ver los siguientes formatos:

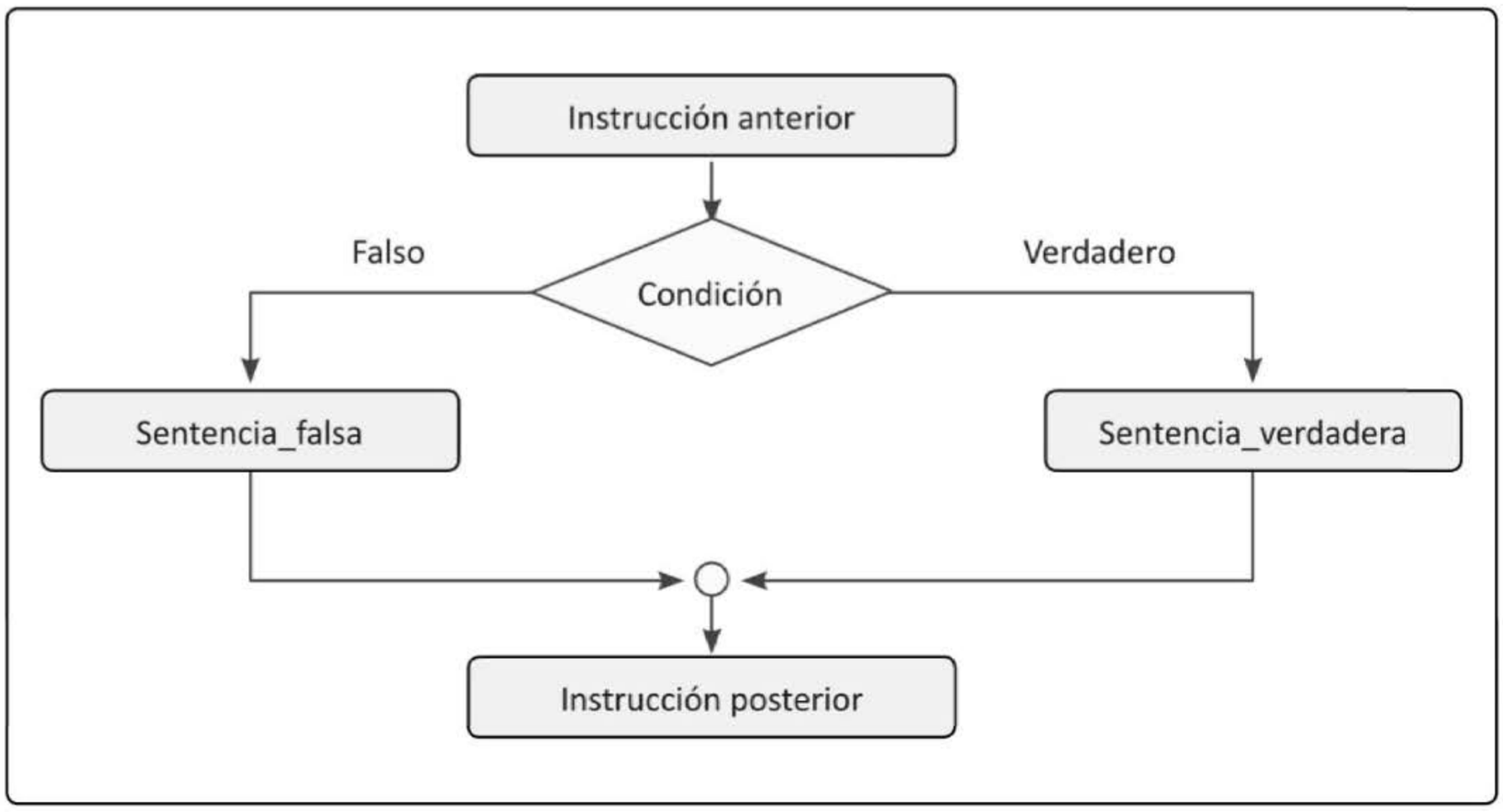
Primer formato:

```
If (condición) Then sentencia_verdadera Else sentencia_falsa
```

Segundo formato:

```
if (condición) Then
   sentencia_verdadera1
   sentencia_verdadera2
else
   sentencia_falsa1
   sentencia_falsa2
End If
```

Gráficamente:



Donde:

- **Condición**: Es una expresión que determina qué acciones se realizarán en una aplicación. Asimismo, se debe considerar que es aquí donde se usan los operadores lógicos y relacionales.
- **Sentencia_verdadera**: Son sentencias que se ejecutarán solo si la condición evaluada resulta verdadera.
- **Sentencia_falsa**: Son sentencias que se ejecutarán solo si la condición evaluada resulta falsa o simplemente no cumple con la condición.

Implementación de sentencias If doble

- Determinar los siguientes mensajes según la determinación del promedio de un determinado alumno, según la siguiente tabla:

Condición	Mensaje
APROBADO	Promovido
DESAPROBADO	Recuperación

```
Dim condición, mensaje As String

If (condición = "APROBADO") Then
        mensaje="Promovido"
        else
        mensaje="Recuperación"
```

- Aplicar descuentos de tipo AFP al importe de sueldo usando la siguiente tabla:

Tipo de AFP	Descuento 1	Descuento 2
Integra	10.0 %	2.0 %
Prima	11.0 %	1.75 %

```
Dim sueldo, descuento1, descuento2 As Double
Dim tipoAFP As String

If (tipoAFP = "Integra") Then
        descuento1 = sueldo * (10/100)
        descuento2 = sueldo * (2/100)
Else
        descuento1 = sueldo * (11/100)
descuento2 = sueldo * (1.75/100)
End If
```

Consideraciones finales de las sentencias If doble

- Para implementar una condición use la combinación de variables y operadores.
- Por regla general, toda implementación de condición debe tener sentencias de acción tanto para el bloque verdadero como para el falso.
- Solo si la condición resulta verdadera se ejecutará las sentencias implementadas en esa sección.
- Si la condición resulta falsa se ejecutará las sentencias implementadas en esa sección, también es llamada sentencias por defecto.

4.2.3 Condición If doblemente enlazada

Se le denomina así a la evaluación sucesiva que puede realizarse en una sentencia If. Si la condición evaluada resulta verdadera entonces se ejecuta las sentencias de dicho bloque y termina el ciclo selectivo, es decir, ya no realiza más comparaciones. Caso contrario seguirá evaluando la siguiente condición hasta encontrar el valor verdadero.

Ver los siguientes formatos:

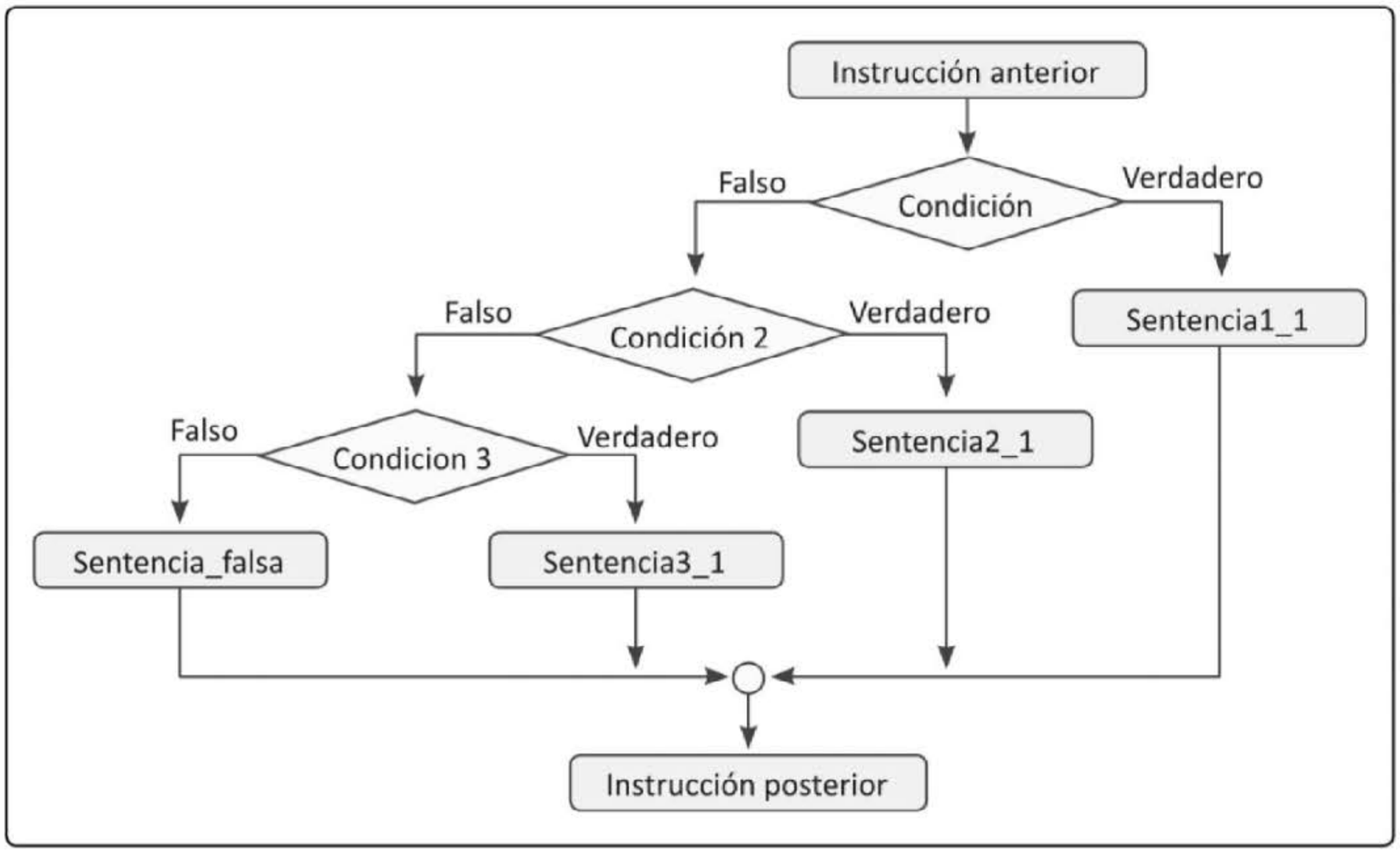
Primera forma:

```
If (condicion1) Then
   sentencia1_1
   Else If (condicion2) Then
   sentencia2_1
   Else If (condicion3) Then
   sentencia3_1
   Else
   sentencia_falsa
End If
```

Segunda forma:

```
If (condicion1) Then
    sentencia1_1
    sentencia1_2
   else if (condicion2) Then
    sentencia2_1
    sentencia2_2
   else if (condicion3) Then
    sentencia3_1
    sentencia3_2
   else
    sentenciaFalsa1
    sentenciaFalsa2
End If
```

Gráficamente:



Donde:

- **Condicion1, Condicion2, Condicion3**: Determina la cantidad de evaluaciones por la que puede pasar una variable, es decir, las posibles opciones que puede contener.
- **Sentencia**: Representa la o las expresiones que se ejecutarán única y exclusivamente si la condición evaluada es verdadera, caso contrario seguirá evaluando hasta que la condición sea verdadera.
- **Sentencia_falsa**: Representa las sentencias que se ejecutarán cuando ninguna de las condiciones resultara verdadera; son también llamadas sentencias ejecutadas por defecto.

Implementación de sentencias If doblemente enlazada

- Determinar el nombre de la categoría de un empleado según la siguiente tabla:

Categoría	Descripción
A	Jefe
B	Administrativo
C	Apoyo
D	Operario

```
Dim categoria, descripcion As String

if (categoria = "A") Then
        descripción = "Jefe"
        else if (categoria = "B") Then
                descripción = "Adiministrativo"
        else if (categoria = "C") Then
                descripción = "Apoyo"
        else if (categoria = "D") Then
                descripción = "Operario"
End If
```

- Aplicar la siguiente tabla de descuentos con respecto al sueldo de los empleados en una empresa:

Sueldo	Porcentaje de descuento
Menor o igual a $750.00	7.50
Superior a $750.00 e inferior a $1400.00	12.50
Superior a $1400.00 e inferior a $2100.00	14.50
Superior a $2100.00	16.50

```
Dim sueldo, descuento As Double

if (sueldo <= 750) Then
        descuento = sueldo * (7.5/100)
        else if (sueldo > 750 And sueldo <= 1400) Then
                descuento = sueldo * (12.5/100)
        else if (sueldo > 1400 And sueldo <= 2100) Then
                descuento = sueldo * (14.5/100)
        else if (sueldo > 2100) Then
                descuento = sueldo * (16.5/100)
End If
```

- Aplicar la siguiente tabla de asignación de costo hora y descuentos con respecto a la categoría de los empleados en una empresa:

Categoría	Costo hora	Descuento
Jefe	$50.00	18 %
Administrativo	$30.00	15 %
Apoyo	$20.00	12 %
Operario	$7.50	10 %

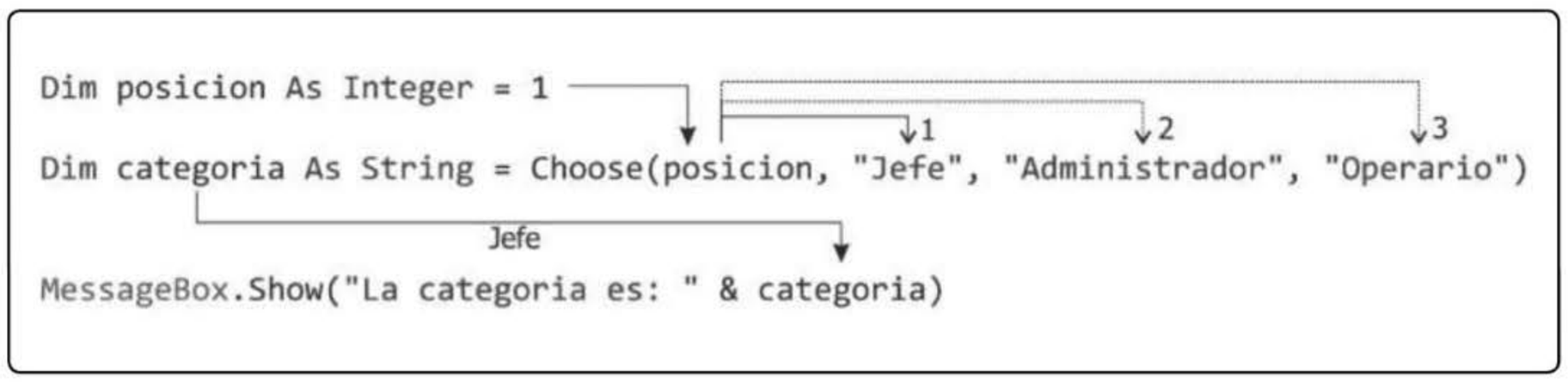
```
Dim categoria As String
        Dim costoHora, descuento As Double

        If (categoria = "Jefe") Then
             costoHora = 50
             descuento = 0.18
        ElseIf (categoria = "Administrativo") Then
             costoHora = 30
             descuento = 0.15
        ElseIf (categoria = "Apoyo") Then
             costoHora = 20
             descuento = 0.12
        ElseIf (categoria = "Operario") Then
             costoHora = 7.5
             descuento = 0.1
End If
```

Consideraciones finales de las sentencias If doblemente enlazada

- Para implementar una condición use la combinación de variables y operadores.
- Por regla general, toda implementación de condición debe tener sentencias de acción.
- Solo si la condición resulta verdadera se ejecutarán las sentencias implementadas en esa sección y terminará el proceso condicional.
- Si la condición resulta falsa se buscará la siguiente condición.
- Cuando ninguno de los valores cumple con la especificación de la condición se ejecutarán las sentencias especificadas en el bloque Else; en caso no exista implementación, se ejecutarán las sentencias próximas a la sentencia If.

4.3 Sentencia de condición múltiple simple Choose

Devuelve un valor a partir de una lista predeterminada de valores; estos se almacenan en una posición inicial de uno.



La variable Posición inicia en uno con el objetivo de obtener el cargo Jefe, la variable Categoría recibirá la evaluación realizada por la sentencia Choose, la cual evalúa el valor de la variable Posición y, de acuerdo con dicho valor, devuelve un valor según el índice.

Presenta el siguiente formato:

```
Variable = Choose(Índice, valor1, valor2, …, valorN)
```

A continuación, una aplicación básica para poder entender el manejo de la sentencia Choose, donde al seleccionar un determinado cargo nos muestra cuál es el costo por hora según el cargo:

Código de solución:

```
Public Class frmChoose
    Private Sub frmChoose_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        With cboCargo.Items
            .Add("Jefe")
            .Add("Administrador")
            .Add("Operario")
            .Add("Servicio")
        End With
    End Sub

    Private Sub cboCargo_SelectedIndexChanged(…) Handles cboCargo.
SelectedIndexChanged
        Dim costo As Double = Choose(cboCargo.SelectedIndex + 1, 50, 30, 10, 8)
        lblCosto.Text = costo.ToString("C")
    End Sub
End Class
```

Se inicia colocando el código en el evento load del formulario para poder llenar el cuadro combinado con los cargos de los empleados. Luego, en el evento SelectedIndexChanged del cuadro combinado se obtiene el costo en base al índice del elemento seleccionado, pero el índice obtenido desde el cuadro combinado siempre inicia en cero y se le aumenta su valor en uno: **cboCargo.SelectedIndex+1**. Finalmente, se debe entender que el Jefe tiene un costo por hora de $50.00, Administrador $30.00, Operario $10.00 y personal de Servicio $8.00.

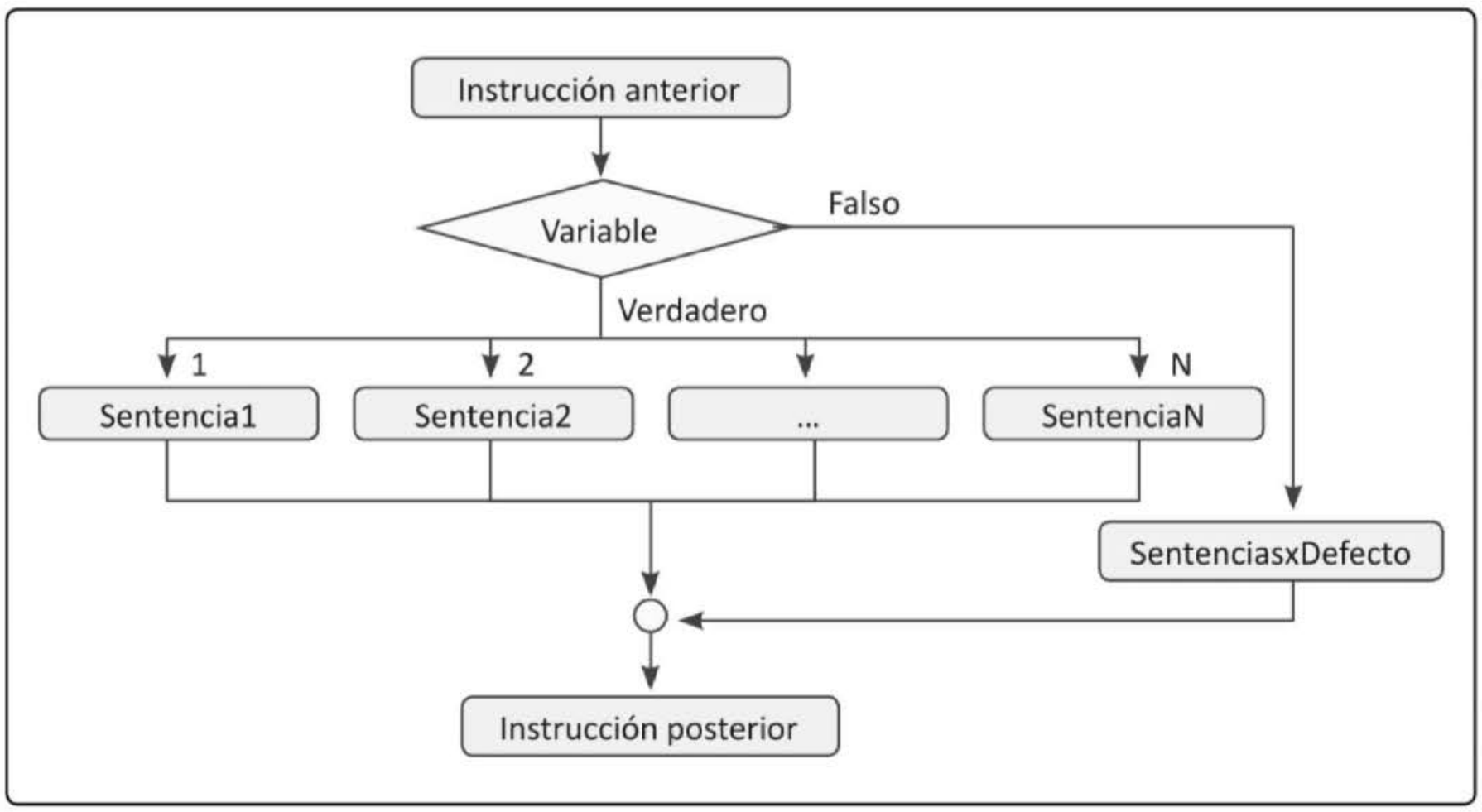
4.4 Sentencia de condición múltiple

Permite realizar una serie de comparaciones hasta encontrar una opción correspondiente a la variable evaluada; tiene un trabajo similar a lo especificado en la sentencia If doblemente enlazada.

Ver el siguiente formato:

```
Select Case opcion
   Case valor1
     Sentencia para la opcion1
   Case valor2
     Sentencia para la opcion2
   Case else
     Sentencia por defecto
End Select
```

Gráficamente:



Donde:

- **Select Case opción**: Opción que representa el dato que evaluará la estructura Select Case. Cuando este valor se encuentre entre las opciones del Case, se ejecutarán las sentencias especificadas en dicho caso.
- **Case valor**: Representa uno de los valores que puede tener la opción evaluada. Si esto sucede se ejecutan las sentencias especificadas dentro de este bloque.

- **Case Else**: Las sentencias que se especifiquen en este bloque se ejecutarán solo si la opción evaluada no se encuentra entre los valores. Es también llamado bloque por defecto.

Especificación de valor	Especificación select case

Case valor_numérico	`Dim n As Integer`
	`Select Case n`
	`  Case 1 : Sentencias`
	`  Case 2 : Sentencias`
	`  Case 3 : Sentencias`
	`End Select`

Case valor_cadena	`Dim cadena As String`
	`Select Case cadena`
	`  Case "opcion1" : Sentencias`
	`  Case "opcion2" : Sentencias`
	`  Case "opcion3" : Sentencias`
	`End Select`

Case Else	`Select Case opción`
	`  Case "opcion1" : Sentencias opcion1`
	`  Case "opcion2" : Sentencias opcion2`
	`  Case "opcion3" : Sentencias opcion3`
	`  Case Else : Sentencias por defecto`
	`End Select`

Implementación de la sentencia Select Case

Determinar el nombre de la categoría de un empleado según la siguiente tabla:

Categoría	Descripción
A	Jefe
B	Administrativo
C	Apoyo
D	Operario

```
Dim categoria, descripción As String
Select Case categoria
            Case "A" : descripción = "Jefe"
            Case "B" : descripción = "Administrativo"
            Case "C" : descripción = "Apoyo"
            Case "D" : descripción = "Operario"
End Select
```

- Aplicar la siguiente tabla de asignación de costo, hora y descuentos con respecto a la categoría de los empleados en una empresa:

Categoría	Costo hora	Descuento
Jefe	$50.00	18 %
Administrativo	$30.00	15 %
Apoyo	$20.00	12 %
Operario	$7.50	10 %

```
Dim categoria As String
Dim costoHora, descuento As Double

Select Case categoria
       Case "Jefe"
              costoHora = 50
              descuento = 0.18
       Case "Administrativo"
              costoHora = 30
              descuento = 0.15
       Case "Apoyo"
              costoHora = 20
              descuento = 0.12
      Case "Operario"
              costoHora = 7.5
              descuento = 0.1
End Select
```

Consideraciones finales de las sentencias de condición múltiple:

- Los valores más comunes para la búsqueda pueden ser de tipo Int y String.
- Por regla general, toda implementación de Case debe tener por lo menos una sentencia de acción.
- Solo si el valor buscado se encuentra en la lista de casos se ejecutará las sentencias implementadas en esa sección y terminará el proceso Select Case.
- Si no encuentra el valor buscado se seguirá buscando en los casos siguientes.
- En un bloque de sentencias se puede hacer referencia a estructuras como If, For o Select Case.
- Cuando ningún valor cumple con la especificación buscada se ejecutará el caso Else, conocido como caso por defecto.

4.5 Desarrollo de aplicaciones condicionales

Ahora se desarrollarán aplicaciones que hagan uso de las diferentes sentencias de condición como If y Select Case; para todos los casos se creará un proyecto de la siguiente manera:

- Siga la ruta **Archivo > Nuevo proyecto**.
- Seleccione la plantilla **Visual Basic > Windows > Aplicación de Windows Forms**.

- Escriba un nombre y seleccione **Aceptar**.

Caso desarrollado 1 If simple - Registro de venta de gasolina

Una empresa de distribución de gasolina desea controlar la venta que realizan sus vendedores en un día, para lo cual desean implementar una aplicación que registre el tipo de gasolina y el monto que el cliente solicita. La aplicación deberá determinar el costo según el tipo de gasolina, así como la cantidad de galones que corresponde dicho importe.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Usar declaración de variables globales.
- Debe seleccionar el tipo de gasolina y el importe que el usuario solicita.
- Los costos de los tipos de gasolina se dan bajo la siguiente tasa:

Tipo de gasolina	Costo $
84 Octanos	$10.65
90 Octanos	$12.75
95 Octanos	$14.55

- La cantidad de galones se obtiene a partir del importe del cliente y el costo según el tipo de gasolina.
- Mostrar todos los montos con el formato de moneda.
- El botón **Resumen** es el encargado de mostrar la información del costo por galón y la cantidad de galones.
- El botón **Registrar** es el encargado de mostrar información al control ListView como el número de venta, tipo de gasolina, costo según el tipo, importe del usuario y la cantidad de galones resultantes producto de la venta.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón **Salir**.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) CancelButton Text	frmVenta btnCancelar CONTROL DE VENTA DE GASOLINA – ESTACION 01
Label1	Text	CONTROL DE VENTA DE GASOLINA – ESTACION 01
Label2	Text	TIPO DE GASOLINA
Label3	Text	IMPORTE $

Control	Propiedad	Valor
Label4	Text	FECHA
Label5	(Name)	lblFecha
ComboBox1	(Name)	cboTipo
TextBox1	(Name)	txtImporte
Button1	(Name) Text	btnResumen RESUMEN
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnRegistrar REGISTRAR
Button4	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLine Columns View	lvVenta True Agregue 5 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmEstacion
    Dim n As Integer
    Dim tipo As String
    Dim importe, costo, galones As Double

   Private Sub frmEstacion_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Llenando los tipos de gasolinas
        cboTipo.Items.Add("84 Octanos")
        cboTipo.Items.Add("90 Octanos")
        cboTipo.Items.Add("95 Octanos")

        'Mostrando la fecha
        lblFecha.Text = DateTime.Now.ToShortDateString()
    End Sub

    Private Sub btnResumen_Click(sender As Object, e As EventArgs)
        'Capturar los datos
        tipo = cboTipo.Text
        importe = Double.Parse(txtImporte.Text)
```

```
        'Realizar cálculos
        If tipo = "84 Octanos" Then costo = 10.65
        If tipo = "90 Octanos" Then costo = 12.75
        If tipo = "95 Octanos" Then costo = 14.55

        galones = importe / costo
        lblGalones.Text = galones.ToString("0.00")

        'Realizar impresiones
        lblCosto.Text = costo.ToString("0.00")
        lblGalones.Text = galones.ToString("0.00")
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        n += 1
        Dim fila As New ListViewItem(n.ToString())
        fila.SubItems.Add(tipo)
        fila.SubItems.Add(costo.ToString("0.00"))
        fila.SubItems.Add(importe.ToString("0.00"))
        fila.SubItems.Add(galones.ToString("0.00"))
        lvVenta.Items.Add(fila)
        btnCancelar_Click(sender, e)
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        cboTipo.SelectedIndex = -1
        txtImporte.Clear()
        lblCosto.Text = ""
        lblGalones.Text = ""
        cboTipo.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

A continuación, una breve explicación de los códigos mostrados:

En el evento load del formulario se codifica para poder llenar el cuadro combinado de los diferentes tipos de gasolinas.

```
cboTipo.Items.Add("84 Octanos")
cboTipo.Items.Add("90 Octanos")
cboTipo.Items.Add("95 Octanos")
```

Otra forma de realizar el mismo código podría ser usando la sentencia With; esta sentencia permite reducir el código repetido que, para este caso, se manifestó en la sentencia **cboTipo.Items**, la cual se repite en varias líneas de sentencia y podría ser resumida con el siguiente código:

```
With cboTipo.Items
     .Add("84 Octanos")
     .Add("90 Octanos")
     .Add("95 Octanos")
End With
```

Asimismo, en el mismo evento se muestra la fecha actual desde el sistema mediante la sentencia `DateTime.Now.ToShortDateString()`.

En el botón **Resumen** se hace uso de la sentencia If de tipo simple, ya que solo se evaluará el tipo de gasolina por un solo tipo y se le asignará el costo directamente, tal como se muestra en el siguiente código:

```
If tipo = "84 Octanos" Then costo = 10.65
If tipo = "90 Octanos" Then costo = 12.75
If tipo = "95 Octanos" Then costo = 14.55
```

Lo que se debe tener en cuenta es que el resultado de la evaluación es correcto, pero el desempeño de la aplicación no es una buena práctica, ya que el compilador evalúa todas las sentencias selectivas; por ejemplo, si el tipo fuera 84 octanos, el costo sería 10.65 y seguirá evaluando la segunda y tercera condicional a pesar de ya haber encontrado su condición correcta.

Finalmente, se imprimen los valores calculados en los controles lblCosto y lblGalones de la siguiente manera:

```
lblCosto.Text = costo.ToString("0.00")
lblGalones.Text = galones.ToString("0.00")
```

En esta ocasión no se usará el parámetro C en el método ToString, ya que solo queremos mostrar a dos decimales el valor resultante en el costo y en la cantidad de galones calculados.

En el botón **Registrar**, la misión es enviar toda la información ingresada y calculada al control ListView, creando el objeto fila de clase ListViewItem. Se debe tener en cuenta que la variable N es global y que dentro de este botón aumenta por cada registro. Finalmente, se invoca al código implementado en el botón **Cancelar** mediante la sentencia **btnCancelar_Click(sender, e)**.

Caso desarrollado 2 If doble - Control de contratos de publicidad

Una empresa de publicidad necesita crear una aplicación, la cual permita controlar los contratos que un determinado cliente realiza por el servicio de publicidad por medio televisivo y radial.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Usar declaración de variables globales para la numeración de los contratos y el total neto a pagar.
- Debe seleccionar el medio publicitario y la cantidad de contratos.
- Los costos por medio publicitario se basan en la siguiente tabla:

Tipo de publicidad	Costo $
Televisión	$45000.00
Radioemisora	$25000.00

- Los descuentos se basan en la cantidad de contratos registrados, tal como se muestra en la siguiente tabla:

Número de contratos	Descuento
MENOR O IGUAL A 5	10.00 %
SUPERIOR A 5	17.50 %

- Mostrar todos los montos con el formato de moneda.
- El botón **Registrar** es el encargado de mostrar la información del número de control, medio publicitario, número de contrato, subtotal, descuento, neto y total a pagar por los servicios de publicidad y a la vez debe ser asociado a la tecla <Enter>.

- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón **Salir**.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	FrmPublicidad btnRegistrar btnSalir CONTROL DE CONTRATOS DE PUBLICIDAD
Label1	Text	CONTROL DE CONTRATOS DE PUBLICIDAD
Label2	Text	MEDIO PUBLICITARIO
Label3	Text	CANTIDAD DE CONTRATOS
Label4	Text	TOTAL A PAGAR
Label5	(Name)	lblTotal
ComboBox1	(Name)	cboMedio
TextBox1	(Name)	txtCantidad
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLine Columns View	lvContratos True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmPublicidad
    Dim n As Integer
    Dim total As Double
     Private Sub frmPublicidad_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
        'Llenando los medios de publicidad
        cboMedio.Items.Add("Televisión")
        cboMedio.Items.Add("Radioemisora")
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'Capturando valores del formulario
        Dim medio As String = cboMedio.Text
        Dim contratos As Integer = Integer.Parse(txtCantidad.Text)

        'Realizando calculos
        Dim costo As Double
        If medio = "Television" Then
            costo = 45000
        Else
            costo = 25000
        End If

        Dim descuento As Double
        If contratos <= 5 Then
            descuento = 10 / 100 * costo
        Else
            descuento = 17.5 / 100 * costo
        End If
        Dim neto As Double = costo - descuento

        'Imprimir resultados
        n += 1
        Dim fila As New ListViewItem(n.ToString("000"))
        fila.SubItems.Add(medio)
        fila.SubItems.Add(contratos.ToString("0.00"))
        fila.SubItems.Add(costo.ToString("0.00"))
        fila.SubItems.Add(descuento.ToString("0.00"))
        fila.SubItems.Add(neto.ToString("0.00"))
        lvContratos.Items.Add(fila)

        total += Double.Parse(neto)
        lblTotal.Text = "$" + total.ToString("0.00")
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        cboMedio.SelectedIndex = -1
```

```
        txtCantidad.Clear()
        cboMedio.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Una breve explicación de los códigos mostrados:

En el evento load del formulario se coloca el código que permite llenar el cuadro combinado con los elementos solicitados en la aplicación:

```
cboMedio.Items.Add("Televisión")
cboMedio.Items.Add("Radioemisora")
```

Asimismo, se podría resumir el código usando la sentencia With de la siguiente manera:

```
With cboMedio.Items
      .Add("Televisión")
      .Add("Radioemisora")
End With
```

Si se usa la declaración por enumeración primero debe definirse fuera de la clase y luego en el evento load recuperar la información con la estructura repetitiva For Each, tal como se muestra en el siguiente código:

```
Public Class frmPublicidad
     Private Sub frmPublicidad_Load(sender As Object, e As EventArgs)
Handles MyBase.Load
        For Each m In System.Enum.GetValues(GetType(Medios))
            cboMedio.Items.Add(m.ToString)
        Next
    End Sub
End Class

Enum Medios As Integer
    Television
    Radioemisora
End Enum
```

En el botón **Registrar** se inicia por evaluar el tipo de medio seleccionado desde el cuadro combinado, usando la sentencia If de tipo doble:

```
Dim costo As Double
If medio = "Television" Then
   costo = 45000
   Else
   costo = 25000
End If
```

Donde la variable Costo recibirá uno de los dos valores asignados dentro de la sentencia selectiva; cuando se evalúa el valor de la variable Medio se realiza con Televisión y se le asigna el monto de 45000 si es True, mientras que cuando es False se refiere al tipo Radioemisora y se le asigna el costo a 25000.

Para poder determinar el descuento, también se realiza una evaluación de tipo selectiva doble usando, para este caso, el número de contratos registrados, tal como se muestra en el siguiente código:

```
Dim descuento As Double
If contratos <= 5 Then
   descuento = 10 / 100 * costo
   Else
   descuento = 17.5 / 100 * costo
End If
```

Entonces, se debe entender que si la cantidad de contratos es menor o igual a 5 se aplica un descuento del 10 %, que es análogo a 10/100 sobre el costo. Caso contrario, es decir, si los contratos son superiores a 5 entonces se aplica el 17.5 % al costo; este último es tomado por defecto porque no existe una nueva evaluación sobre los contratos.

Caso desarrollado 3 — If doblemente enlazado - Simulador de pensión para estudiantes

Una institución particular necesita implementar una aplicación que permita simular el neto a pagar que debe realizar un estudiante según la categoría y su promedio ponderado del ciclo anterior.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe seleccionar la categoría del estudiante y el promedio ponderado.
- Al seleccionar una categoría deberá mostrarse el monto asignado de pensión según la siguiente tabla:

Categoría	Pensión $
PROMOVIDO	$ 950.00
TRASLADO EXTERNO	$ 1075.00
CAMBIO DE TURNO	$ 1010.00

- Los descuentos se basan en el promedio ponderado del ciclo anterior, tal como se muestra en la siguiente tabla:

Promedio ponderado	Descuento
Menor a 12.99	0.00 %
Entre 13 y 15.99	10.50 %
Entre 16 y 17.99	12.50 %
Entre 18 y 20	17.50 %

- El monto neto resultado de la diferencia entre la pensión y el descuento.
- Mostrar todos los montos con el formato de moneda.
- El botón **Simular** es el encargado de mostrar el monto de pensión, descuento, neto a pagar y, a la vez, debe ser asociado a la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón **Salir**.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	FrmPension btnSimular btnSalir SIMULADOR DE PENSION PARA ESTUDIANTES REGULARES
Label1	Text	SIMULADOR DE PENSION PARA ESTUDIANTES REGULARES
Label2	Text	CATEGORÍA
Label3	Text	PROMEDIO PONDERADO
Label4	Text	TOTAL A PAGAR
Label5	(Name)	lblPension
Label6	Text	MONTO $
Label7	(Name)	lblMonto
Label8	Text	DESCUENTO $
Label9	(Name)	lblDescuento
Label10	Text	NETO A PAGAR $
Label11	(Name)	lblNeto
ComboBox1	(Name)	cboCategoria
TextBox1	(Name)	txtPromedio

Control	Propiedad	Valor
Button1	(Name) Text	btnSimular SIMULAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR

El script de la solución es:

```
Public Class frmPension
    Private Sub frmPension_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Llenar las categorias
        cboCategoria.Items.Add("PROMOVIDO")
        cboCategoria.Items.Add("TRASLADO EXTERNO")
        cboCategoria.Items.Add("CAMBIO DE TURNO")
    End Sub

    Private Sub cboCategoria_SelectedIndexChanged(sender As Object, e As EventArgs)
        'Capturar la categoria
        Dim categoria As String = cboCategoria.Text

        'Determinar el monto de pensión
        Dim pension As Double
        If categoria = "PROMOVIDO" Then
            pensión = 950
        ElseIf categoria = "TRASLADO EXTERNO"
            pension = 1075
        Else
            pension = 1010
        End If
        lblPension.Text = pension.ToString("0.00")
    End Sub

    Private Sub btnSimular_Click(sender As Object, e As EventArgs)
        'Capturar datos del formulario
        Dim pension As Double = Double.Parse(lblPension.Text)
        Dim promedio As Double = Double.Parse(txtPromedio.Text)

        Dim descuento As Double
        If promedio <= 12.99 Then
            descuento = 0
        ElseIf descuento <= 15.99
            descuento = 10 / 100 * pensión
        ElseIf descuento <= 17.99
            descuento = 12.5 / 100 * pensión
```

```
        ElseIf descuento <= 20
            descuento = 17.5 / 100 * pensión
        End If

        Dim neto As Double = pension - descuento

        'Imprimir resultados
        lblMonto.Text = pension.ToString("C")
        lblDescuento.Text = descuento.ToString("C")
        lblNeto.Text = neto.ToString("C")
    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        cboCategoria.SelectedIndex = -1
        txtPromedio.Clear()
        lblMonto.Text = ""
        lblDescuento.Text = ""
        lblNeto.Text = ""
        cboCategoria.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)
        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Una breve explicación de los códigos mostrados:

Colocamos código en el evento SelectedIndexChanged del cuadro combinado para que al seleccionar un elemento de la lista nos muestre automáticamente el monto de la pensión según la categoría. Asimismo, se usa la sentencia If doblemente enlazada para asignar los montos de pensión, ya que contamos con varias categorías, como se muestra en el siguiente código:

```
Dim pension As Double
If categoria = "PROMOVIDO" Then
   pensión = 950
   ElseIf categoria = "TRASLADO EXTERNO"
   pension = 1075
   Else
   pension = 1010
End If
```

La primera evaluación lo realizamos con la categoría Promovido; si resulta true entonces la pensión será asignado con 950; en caso el resultado sea false, pasaría a una segunda evaluación pero esta vez para la segunda opción de las categorías. En caso hubieran más categorías se extendería el uso de la cláusula ElseIf dentro del bloque. Finalmente, el valor por defecto hace referencia a la última opción de las categorías.

En el botón **Simular** se implementa el uso de la sentencia If doblemente enlazado para determinar el descuento según el promedio del alumno, como se muestra en el siguiente código:

```
Dim descuento As Double
If promedio <= 12.99 Then
   descuento = 0
   ElseIf descuento <= 15.99
   descuento = 10 / 100 * pensión
   ElseIf descuento <= 17.99
   descuento = 12.5 / 100 * pensión
   ElseIf descuento <= 20
   descuento = 17.5 / 100 * pensión
End If
```

En la primera evaluación se especifica al promedio como menor o igual a 12.99, es decir, todos los valores menores a 13; inclusive se incluye el valor cero y negativo, pero se asume que cuando se registre el promedio no se ingresarán dichos valores. En la segunda evaluación se debe tener especial cuidado porque ya no se hace referencia al valor inicial, solo se especifica el tope como menor o igual a 15.99; para llegar a esta evaluación se asume que todos los valores inferiores a 13 ya fueron especificados en la primera evaluación o, mejor dicho, la segunda evaluación solo especificaría a valores mayores o igual a 13 e inferiores a 15.99. De la misma manera se aplica a las demás opciones de la condición.

Caso desarrollado 4 Select Case - Control de venta de productos por temporada

Una tienda por departamento realiza una venta de ciertos productos para damas por temporada y necesita una aplicación que permita generar la venta, de tal forma que nos muestre el importe, descuento y neto a pagar por la compra de cierta cantidad de prendas.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Debe seleccionar la prenda e ingresar la cantidad de prendas a comprar.
- Las prendas de temporada han sido seleccionadas por el administrador del local asignando el precio según la siguiente tabla:

Prenda	Costo unitario $
Pantalon jean	$60.00
Sacos de cuero	$180.00
Falda recta	$50.00
Blusa de seda	$35.00
Pantalón de vestir	$120.00

- Los descuentos de temporada se basan en el importe obtenido por la compra de cierta cantidad de productos, tal como se muestra en la siguiente tabla:

Importe de compra	Descuento
Inferior a $100.00	2.00 %
Entre $100.00 y $200.00	5.00 %
Entre $200.00 y $300.00	10.00 %
Superior a $300.00	20.00 %

- El monto neto a pagar resulta de la diferencia entre el importe y el descuento.
- Mostrar todos los montos con el formato de moneda.
- El botón **Vender** es el encargado de mostrar el nombre de la prenda, cantidad, precio, subtotal y a la vez debe ser asociado a la tecla <Enter>.
- El botón **Limpiar** es el encargado de limpiar el contenido de los controles y a la vez debe ser asociado a la tecla <Esc>.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón **Salir**.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmTemporada btnVender btnLimpiar CONTROL DE VENTA DE PRODUCTOS DE TEMPORADA
Label1	Text	CONTROL DE VENTA DE PRODUCTOS DE TEMPORADA
Label2	Text	SELECCIONE PRENDA
Label3	Text	CANTIDAD
Label4	Text	IMPORTE $
Label5	(Name)	lblImporte
Label6	Text	DESCUENTO $

Label7	(Name)	lblDescuento
Label8	Text	NETO A PAGAR $
Label9	(Name)	lblNeto
ComboBox1	(Name)	cboPrenda
TextBox1	(Name)	txtCantidad
Button1	(Name) Text	btnVender VENDER
Button2	(Name) Text	btnLimpiar LIMPIAR
Button3	(Name) Text	BtnSalir SALIR
ListView1	(Name) GridLine Columns View	lvVenta True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmTemporada
    Dim importe As Double
   Private Sub frmTemporada_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        cboPrenda.Items.Add("PANTALON JEAN")
        cboPrenda.Items.Add("SACOS DE CUERO")
        cboPrenda.Items.Add("FALDA RECTA")
        cboPrenda.Items.Add("BLUSA DE SEDA")
        cboPrenda.Items.Add("PANTALON DE VESTIR")
    End Sub

    Private Sub btnVender_Click(sender As Object, e As EventArgs)
        'Capturar los datos del formulario
        Dim prenda As String = cboPrenda.Text
        Dim cantidad As Integer = Integer.Parse(txtCantidad.Text)

        'Asignar el precio a los productos
        Dim precio As Double
        Select Case prenda
            Case "PANTALON JEAN" : precio = 60
            Case "SACOS DE CUERO" : precio = 180
            Case "FALDA RECTA" : precio = 50
            Case "BLUSA DE SEDA" : precio = 35
```

```
            Case "PANTALON DE VESTIR" : precio = 120
        End Select

        'Calculando el subtotal
        Dim subtotal As Double = precio * cantidad

        'Hallando el acumulado de subtotal
        importe += subtotal

        'Calcular el descuento de la venta
        Dim descuento As Double
        Select Case importe
            Case Is < 100 : descuento = 0.02 * importe
            Case 100 To 200 : descuento = 0.05 * importe
            Case 200 To 300 : descuento = 0.1 * importe
            Case Is > 300 : descuento = 0.2 * importe
        End Select
        Dim neto As Double = importe - descuento

        'Imprimir resultados
        Dim fila As New ListViewItem(prenda)
        fila.SubItems.Add(cantidad.ToString("0.00"))
        fila.SubItems.Add(precio.ToString("0.00"))
        fila.SubItems.Add(subtotal.ToString("0.00"))
        lvVenta.Items.Add(fila)

        lblImporte.Text = importe.ToString("0.00")
        lblDescuento.Text = descuento.ToString("0.00")
        lblNeto.Text = neto.ToString("0.00")
    End Sub

    Private Sub btnLimpiar_Click(sender As Object, e As EventArgs)
        cboPrenda.SelectedIndex = -1
        txtCantidad.Clear()
        lvVenta.Items.Clear()
        lblImporte.Text = ""
        lblDescuento.Text = ""
        lblNeto.Text = ""
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

A continuación, una breve explicación de los códigos mostrados:

Dentro del botón **Vender** implementamos la sentencia selectiva múltiple Select Case para determinar el precio del producto, como se visualiza en el siguiente código:

```
Dim precio As Double
Select Case prenda
    Case "PANTALON JEAN" : precio = 60
    Case "SACOS DE CUERO" : precio = 180
    Case "FALDA RECTA" : precio = 50
    Case "BLUSA DE SEDA" : precio = 35
    Case "PANTALON DE VESTIR" : precio = 120
End Select
```

Como es la variable Prenda la que contiene la información del producto seleccionado por el usuario, es justamente este valor que se especifica en la estructura Select Case y se le asignará un determinado precio según sea el caso, y una vez encontrado un producto automáticamente finaliza la sentencia. En caso no se cumpliera con ninguno de los casos el valor Precio tendría el valor cero asignado automáticamente.

La sentencia Select Case es simplemente una variación simple de la sentencia If doblemente enlazada, por lo que el código también podría haberse implementado de la siguiente manera:

```
Dim precio As Double
If prenda = "PANTALON JEAN" Then
   precio = 60
   ElseIf "SACOS DE CUERO" Then
   precio = 180
   ElseIf "FALDA RECTA" Then
   precio = 50
   ElseIf "BLUSA DE SEDA" Then
   precio = 35
   ElseIf "PANTALON DE VESTIR" Then
   precio = 120
End If
```

Para determinar el descuento se debe evaluar la variable Importe usando la sentencia Select Case de la siguiente manera:

```
Dim descuento As Double
Select Case importe
    Case Is < 100 : descuento = 0.02 * importe
    Case 100 To 200 : descuento = 0.05 * importe
    Case 200 To 300 : descuento = 0.1 * importe
    Case Is > 300 : descuento = 0.2 * importe
End Select
```

En el primer caso se especifica Is <100 para indicar que el importe sea menor a 100, y aquí se debe indicar que no será necesario digitar la cláusula Is porque automáticamente Visual lo asignará; en el segundo caso se evalúa un rango cerrado entre 100 y 200, es decir, el importe debe incluir el valor 100 hasta un tope de 200 para aplicar un descuento del 5 %, de la misma manera se aplicó a los demás casos.

Capítulo 5

Sentencias repetitivas

Capacidad terminal:

Desarrollar aplicaciones que involucren estructuras de repetición como For, While y la integración de ellos en una misma aplicación.

Contenido

- Los contadores
- Operaciones sobre los contadores
- Los acumuladores
- Operaciones sobre los acumuladores
- Sentencia While-End While
- Sentencia Do-Loop While
- Sentencia For
- Desarrollo de aplicaciones repetitivas

5.1 Los contadores

Como su nombre lo indica, realizan conteos de un determinado suceso. La característica principal de los contadores es que siempre inician en cero y cuando ocurre algún suceso aumenta o disminuye en uno su valor; esto se podría dar gracias a las sentencias If o Select Case.

En qué casos se hace uso de los contadores:

- Cuando alguien espera a ser atendidos en un centro bancario.
- Cuando emiten tickets para la entrada de un evento deportivo.
- Cuando alguien cuenta la cantidad de billetes del mismo valor.
- Cuando se verifica el consumo de energía en una vivienda.

Se puede llegar a la conclusión de que un contador es de tipo entero para que pueda aumentar o disminuir en uno su valor actual. Para un mejor control de los datos contados se puede usar sentencias de repetición como For o While en todos sus formatos.

Asimismo, el uso de los contadores se pueden dar de dos formas:

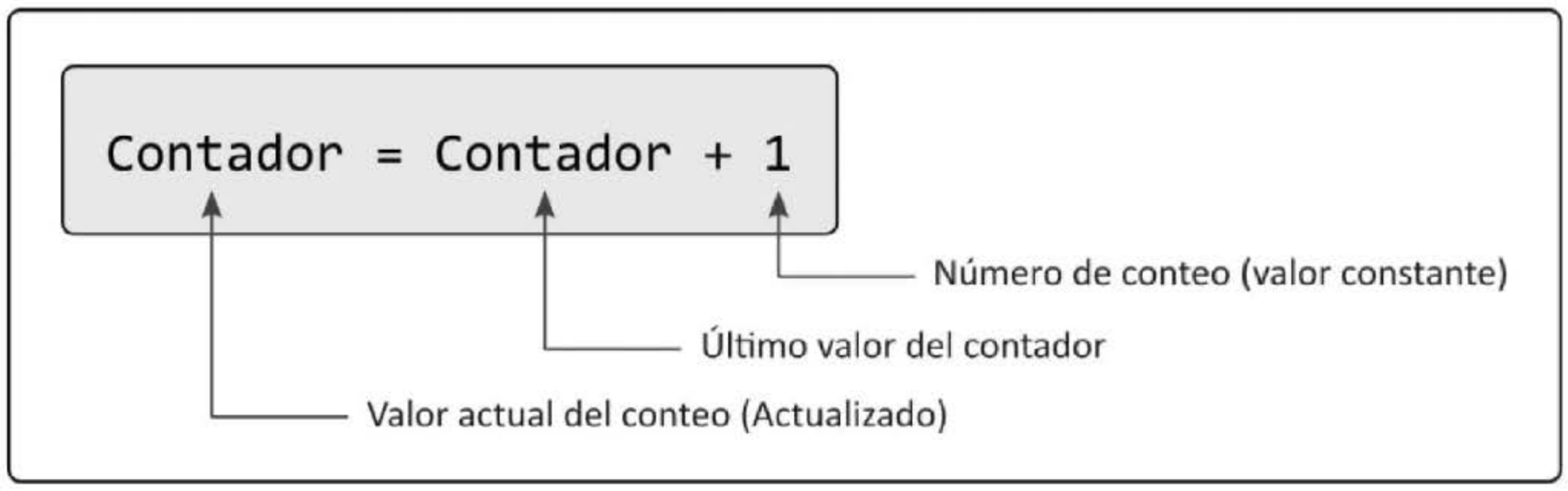
- **Fuera de una estructura repetitiva**: Dicho contador debe declararse de forma global para que el aumento o disminución de su valor se dé solo cuando ocurra una llamada al método. Por ejemplo, se podría aplicar conteos con un botón de registro, haciendo que dicho botón aumente o disminuya a la variable global.
- **Dentro de una estructura repetitiva**: Dicho contador aumenta o disminuye su valor según el ciclo de repeticiones implementado. Por ejemplo, si se cuenta con una población de mil personas y se quiere contabilizar el total de personas mayores y menores de edad; claro está que para diferenciar a las personas por edad se necesitará de la estructura de decisión If.

Finalmente, aquí están los diferentes formatos que puede tener un contador en Visual Basic:

Formato	Descripción
Contador = Contador + 1	Contador simple, el cual permite actualizar la variable contadora con un valor constante; este valor es definido por el usuario.
Contador + = 1	Contador con operador complejo, el cual permite actualizar a la variable contadora con un valor constante definido por el usuario.

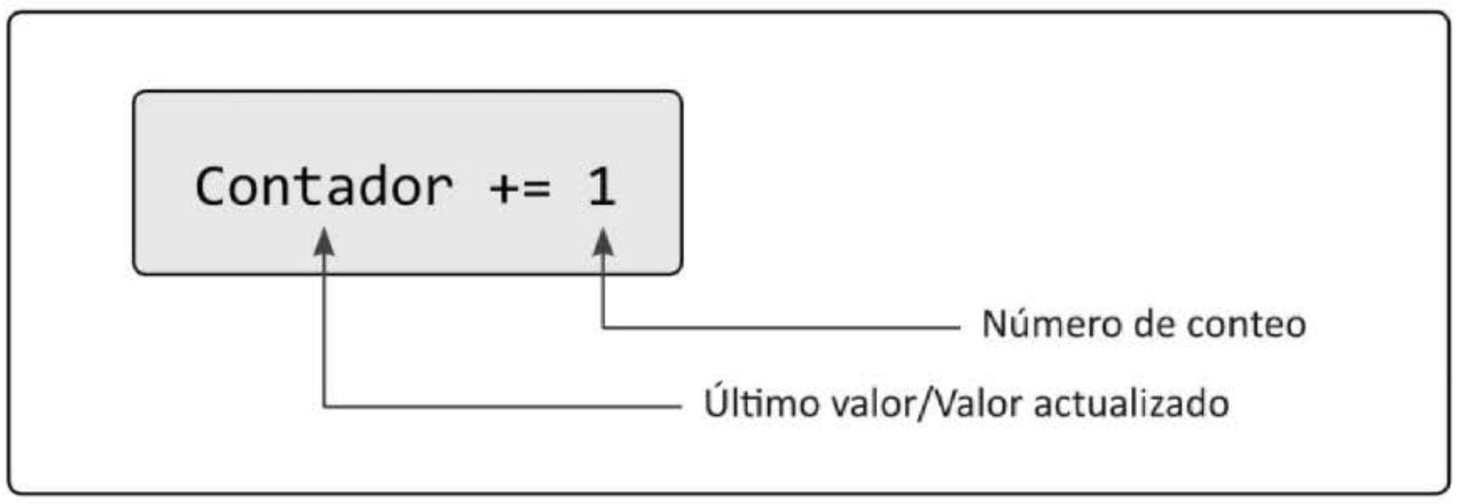
Se analiza la sentencia de conteo:

Se definirá específicamente las partes de la sentencia de conteo simple.



Donde:

- **Numero de conteo**: Es un valor constante que define la forma del conteo. Si es +1, entonces será un contador ascendente, y si es –1, será un contador descendente.
- **Último valor del contador**: Representa el ultimo valor obtenido en el conteo.
- **Valor actual del conteo**: Representa la actualización del valor contenido en el contador; este será utilizado en la misma sentencia como «ultimo valor del contador» en una segunda o más veces que ocurra un conteo.



Donde:

- **Numero de conteo**: Es un valor constante que define la forma del conteo. Si es + = 1 entonces será un contador ascendente, y si es – =1 será un contador descendente.
- **Último valor/Valor actualizado**: Tiene una doble representación, pues en primer lugar representa el último valor del conteo, así como también la actualización del valor contenido en el contador.

5.2 Operaciones sobre los contadores

5.2.1 Inicialización de valores

Visual Basic es menos estricto con respecto a las inicializaciones que se puede realizar a los contadores, pues solo con declararlos, ya sea en forma local o global, automáticamente son inicializados con un valor estándar según el tipo de datos.

Ver las siguientes inicializaciones:

Especificación de valor	Especificación select case
Inicializar la variable C en valor cero.	`Dim c As Integer = 0`
Inicializar el número de factura en 1000.	`Dim factura As Long = 1000`

5.2.2 Operación de incremento:

Una operación de incremento se realiza siempre de forma ascendente, es decir, el valor constante debe ser positivo con un valor constante.

A continuación, una implementación que permita contabilizar números aleatorios de tal forma que, al encontrar el número 9, muestre la cantidad de números generados para encontrar dicho número.

```
Dim aleatorio As New Random()
Dim c As Integer = 0
Dim n As Integer

Do
    n = aleatorio.Next(1, 50)
    MessageBox.Show("El valor aleatorio de N es: " & n.ToString())
    c += 1
Loop While n <> 9

MessageBox.Show("La cantidad de números generados es: " & c.ToString())
```

Se analiza el código mostrado:

- Se crea el objeto aleatorio de la clase Random, el cual permitirá crear números aleatorios en nuestra aplicación.
- Se declara la variable C con un valor de inicio en cero, el cual permitirá contabilizar el número de valores aleatorios generados.
- Se declara la variable N, la cual tiene la misión de registrar los valores aleatorios generados desde el objeto aleatorio.
- Se inicia el ciclo de repeticiones colocando la cláusula Do, y todas las sentencias colocadas dentro de su entorno se repetirán mientras que la condición sea verdadera.
- Se asigna los valores aleatorios generados entre 1 y 50 a la variable N y esta se encargará de mostrarlo al usuario.
- Para realizar la impresión de dicho valor asignado a la variable N se usa la sentencia **MessageBox.Show**. No olvidar asignar ToString a la variable N.
- Antes de finalizar el ciclo de repeticiones la variable contadora C debe aumentar en un valor; la idea principal es que cuente cuántos números se van generando hasta el momento.
- La finalización del ciclo de repeticiones se da cuando la condición es falsa; mientras que la variable N sea diferente a nueve se seguirán ejecutando las sentencias del ciclo.
- Finalmente, se imprime la variable C, la cual contiene la información de la cantidad de números generados aleatoriamente hasta que el número aleatorio sea nueve.

5.2.3 Operación de decremento:

Esta operación realiza un conteo a partir de un valor numérico superior hacia uno inferior.

A continuación, una implementación que permite mostrar cinco números generados aleatoriamente entre el rango 1 y 100; use conteo inverso en el contador.

```
Dim aleatorio As New Random()
Dim i As Integer = 5
Dim n As Integer
Do
    n = aleatorio.Next(1, 100)
    MessageBox.Show("El valor aleatorio de N: " & n.ToString())
    i -= 1
Loop While (i >= 1)
```

Analizaremos el código mostrado:

- Se crea el objeto aleatorio de la clase Random, el cual nos permitirá crear números aleatorios en nuestra aplicación.
- Se declara la variable I con el valor inicial cinco, porque el caso necesita mostrar los cinco primeros valores.
- Se declara la variable N, la cual tiene la misión de guardar el valor aleatorio generado por el objeto aleatorio de clase Random.
- Se envía la información del valor aleatorio mediante la sentencia **MessageBox.Show**.
- La variable I decrementa su valor en uno para contabilizar cuantos números aleatorios se están mostrando. Recordar que I inició en 5 y debe acabar cuando el ciclo de repeticiones finalice.
- Finalmente, la condición especificada en el Loop While determina que, mientras que I sea mayor o igual a 1, el ciclo de repeticiones continuará.

5.3 Los acumuladores

Los acumuladores son usados para totalizar valores, de tal forma que un valor actualiza a otro en valores no constantes.

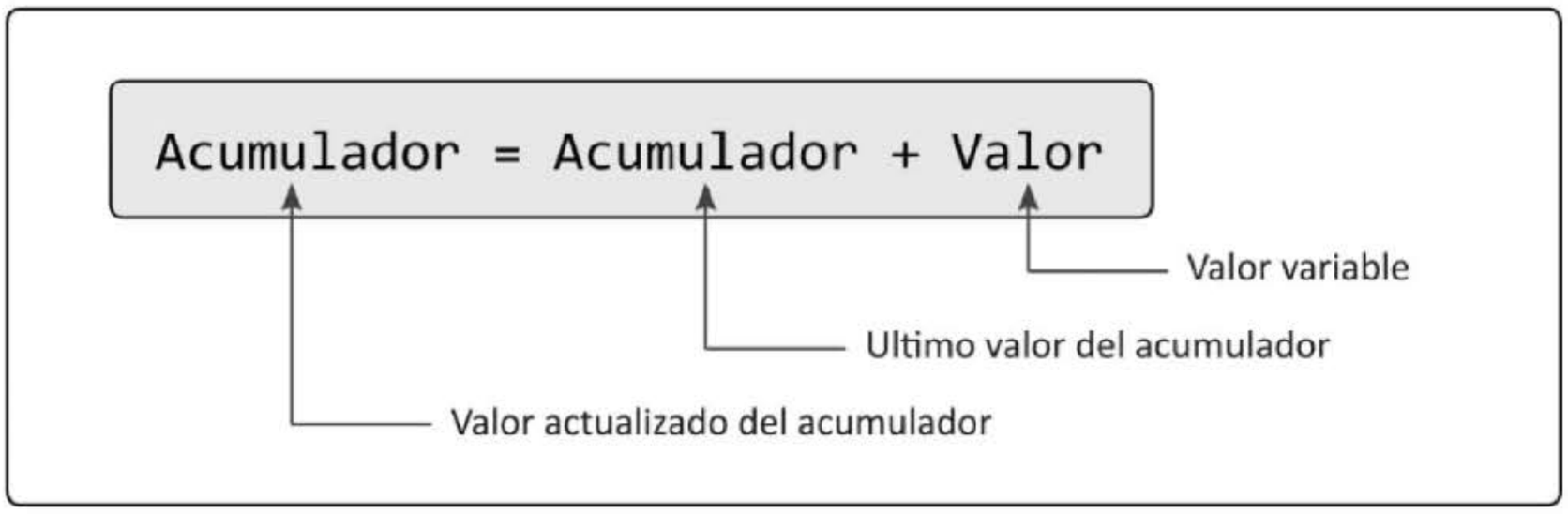
Asimismo, el uso de los acumuladores se puede dar de dos formas:

- **Fuera de una estructura repetitiva**: Donde la acumulación se puede realizar mediante acción de un método. Por ejemplo, un botón de registro de valores podría realizar una acumulación de valores declarados como global.
- **Dentro de una estructura repetitiva**: Es la encargada de acumular valores no constantes por una determinada condición en un ciclo de repeticiones específico.

A continuación, su formato:

Formato	Descripción
acumulador = acumulador + valor	Acumulador simple, donde Valor es un dato no constante que puede variar dependiendo de la regla del negocio de la aplicación y la variable Acumulador usa el actual valor y a la vez lo actualiza.
acumulador + = valor	Acumulador con operador complejo, el cual permite actualizar a la variable acumuladora con un valor no constante.

Gráficamente:



Donde:

- **Valor variable**: Es un dato numérico no constante que permite asignar valores a la variable acumuladora.
- **Ultimo valor del acumulador**: Contiene el ultimo valor acumulado.
- **Valor actual del acumulador**: Es el valor actualizado del acumulador, es decir, es el valor resultante de la acumulación.

5.4 Operaciones sobre los acumuladores

5.4.1 Inicialización de valores

Si se tiene en cuenta que una variable acumuladora debe ser numérica, entonces es necesario pensar en la forma de inicialización que debe tener dicha variable, de tal forma que se puede visualizar los siguientes casos:

Casos	Código Basic
Inicializando el acumulador suma con el valor cero.	`Dim suma As Double = 0`
Inicializando el acumulador monto con el valor inicial 1000; asumiendo que se realizan pagos sucesivos de una cuenta a partir de dicho monto.	`Dim factura As Long = 1000`

5.4.2 Operación de acumulación en incremento:

Permite aumentar un valor a la variable acumuladora, haciendo que se comporte como un acumulador de incremento positivo.

Ver la siguiente implementación para acumular 5 números aleatorios de dos cifras:

```
Dim aleatorio As New Random()
Dim s As Integer = 0
For i = 1 To 5
    Dim n As Integer = aleatorio.Next(1, 99)
    MessageBox.Show("El valor aleatorio de N: " & n.ToString())
    s += n
Next
MessageBox.Show("El valor acumulado es: " & s.ToString())
```

Se analiza el código mostrado:

- Se crea el objeto aleatorio de la clase Random, el cual permitirá crear números aleatorios en nuestra aplicación.
- Se declara la variable S, la cual tiene la misión de acumular los valores aleatorios generados.
- Se iniciamos el ciclo de repeticiones usando la sentencia For, en la cual se especifica el inicio en uno y finaliza en cinco.
- Dentro del bloque For se declara la variable N, la cual tiene la misión de guardar el valor aleatorio generado desde el objeto de la clase Random.
- Se imprime la variable N mediante la sentencia **MessageBox.Show**, la cual contiene el valor aleatorio generado.
- Se aplica la acumulación mediante la sentencia S + = N, en la cual N representa al valor aleatorio y la variable S acumula dichos valores.
- Finalmente, se imprime el valor acumulado en la variable S.

5.4.3 Operación de acumulación en decremento

Esta operación realiza la acumulación inversa, se puede usar con montos o valores que sucesivamente decrementan, como el pago de una deuda o el monto de retiro de una cuenta.

Si se cuenta con un monto inicial de $5000.00 y se realizan 4 retiros simultáneos de montos aleatorios entre $10.00 y $50.00, mostrar el nuevo monto a partir de los retiros.

```
Dim aleatorio As New Random()
Dim monto As Double = 5000
Dim i As Integer = 1
Dim retiro As Double
Do
    retiro = aleatorio.Next(10, 50)
    MessageBox.Show("El monto a retirar es :" & retiro)
    monto -= retiro
    i += 1
Loop While (i <= 4)
MessageBox.Show("El monto actual es: " & monto.ToString("C"))
```

Se analiza el código mostrado:

- Se crea el objeto aleatorio de la clase Random, el cual permitirá crear montos aleatorios en nuestra aplicación.
- Se declara la variable Monto, el cual se inicia en 5000 como monto inicial.
- Se declara la variable I con un valor inicial de uno; su función principal es de contabilizar el ciclo de repeticiones.
- La condición del ciclo de repeticiones indica que el valor I debe llegar hasta 4, por lo tanto, todas las sentencias dentro del ciclo se repetirán; desde allí se puede ver que la variable Monto disminuirá según el valor aleatorio de variable Retiro.
- Finalmente, se imprime el valor actual del monto después de cuatro disminuciones consecutivas.

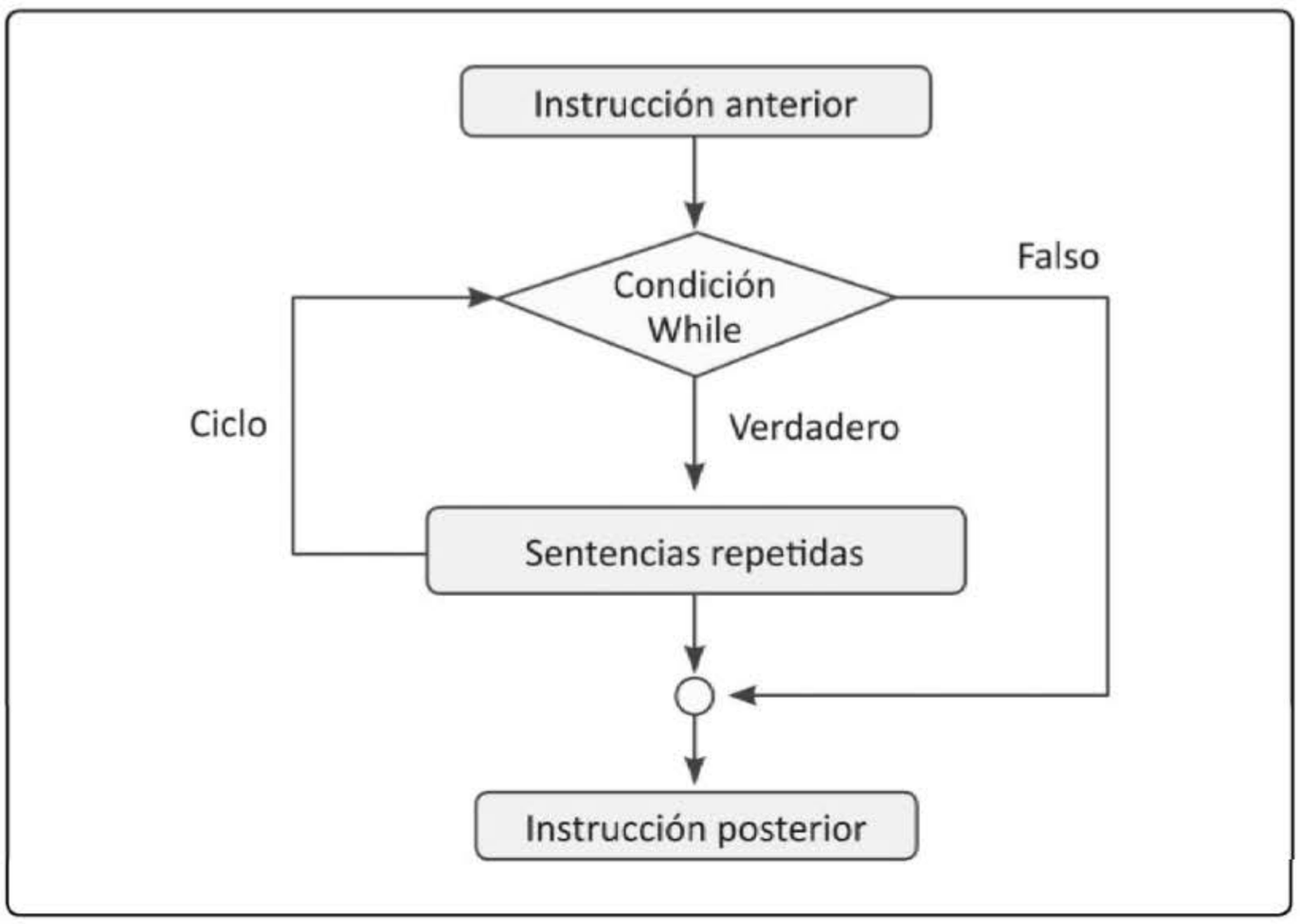
5.5 Sentencia While - End While

Permite ejecutar un conjunto de sentencias en un ciclo de repeticiones especificadas en la condición While, de tal forma que si la condición resulta verdadera se ejecutan las sentencias, caso contrario finaliza el ciclo de repeticiones.

A continuación, su formato:

```
while (condición)
   'Sentencias repetidas
End While
```

Gráficamente:



Donde:

- **Condición While**: Determina la cantidad de ciclo de repeticiones que va realizar la estructura.
- **Sentencias repetidas**: Representa a las sentencias que se repetirán durante el ciclo de repeticiones While; se debe tener en cuenta que dentro del bloque While se puede colocar cualquier tipo de sentencia como secuencial, condicional o repetitiva.

A continuación, algunos códigos que implementan el uso de la instrucción While… End While

Listar 10 números aleatorios usando la estructura While.

```
Dim i As Integer = 1
While (i <= 10)
    Dim Rnd As New Random()
    Dim n As Integer = Rnd.Next(1, 100)
    MessageBox.Show(n.ToString())
    i += 1
End While
```

Se analiza el código mostrado:

- Se inicia la variable I en uno para que el ciclo de repeticiones tenga un punto de inicio.
- Se implementa la sentencia While donde se condiciona que dicha variable I sea menor o igual a diez; con esto se marca el punto máximo que puede tomar el ciclo de repeticiones.
- Se crea el objeto RND de la clase Random, el cual nos permitirá generar números aleatorios.
- Se declara la variable N con el primer valor generado por el objeto RND; de tal forma que Next obtiene el siguiente valor aleatorio entre un rango de 1 y 100.
- Para imprimir cada número generado se usa la sentencia **MessageBox.Show**; asignar ToString a la variable N para que pueda considerarse como un valor de cadena.
- Antes de finalizar el ciclo de repeticiones de While, se debe tener en cuenta que la variable I es la encargada de controlar las repeticiones, por lo tanto, dicha variable debe aumentar su valor en uno hasta que la condición especificada en el While resulte falsa.

Listar los divisores de un numero entero.

```
Dim n As Integer = 8
Dim i As Integer = 1

While (i <= n)
   If (n Mod i = 0) Then
      MessageBox.Show(i.ToString())
   End If
   i += 1
End While
```

Se analiza el código mostrado:

- Se inicializa la variable N en ocho para obtener de ella sus divisores.
- Se inicializa la variable I en uno para que el ciclo de repeticiones tenga un punto de partida.
- Se implementa el ciclo de repeticiones While condicionando que la variable I sea siempre inferior o igual a N; con esto se puede llegar a la conclusión de que la cantidad de ciclos de repeticiones se dará desde el valor uno hasta ocho, ya que es el valor asignado a la variable N.
- Se condiciona la variable N para encontrar los divisores (N Mod i = 0) donde la función Mod retorna el residuo entre la variable N y el valor I.
- Se muestra el mensaje con los divisores, los cuales se encuentran en la variable I; solo única y exclusivamente si la condición anterior resulta verdadera.
- Antes de finalizar el ciclo de repeticiones de While, se debe tener en cuenta que la variable I es la encargada de controlar las repeticiones, por lo tanto, dicha variable debe aumentar su valor en uno hasta que la condición especificada en el While resulte falso.

Listar los 10 primeros números pares de una serie máximo permitida por los números enteros.

```
Dim i As Integer = 1
Dim cp As Integer = 0

While (i <= Integer.MaxValue)
   If (i Mod 2 = 0) Then
      MessageBox.Show(i.ToString())
      cp += 1
   End If

   If (cp = 10) Then Return
      i += 1
End While
```

Se analiza el código mostrado:

Se inicializa la variable I en uno para que el ciclo de repeticiones tenga un punto de partida.

- Se inicializa la variable CP en cero, la cual tiene la misión de contabilizar la cantidad de pares que se van obteniendo.
- Se implementa el ciclo de repeticiones While, condicionando que la variable I sea menor o igual a un número máximo permitido por la clase Integer (Integer.MaxValue).
- Se evalúa cada valor que puede tener la variable I, comparando si dicho valor es par; esto se obtiene usando la función Mod, el cual devuelve el valor residuo entre la variable I y el número dos, ya que cuando el residuo devuelve el valor cero se considera que el valor I es par.
- Se contabiliza el par encontrado y se almacena dicho conteo en la variable CP, usando la teoría vista en los contadores (CP + = 1).
- Se verifica la cantidad de pares encontrados con la sentencia (CP = 10); si la variable CP llega ser igual a 10, entonces se ejecuta la sentencia Return que permite terminar el ciclo de repeticiones.
- Finalmente, se aumenta en uno el valor de I para que el ciclo de repeticiones continúe, mientras que la condición lo permita.

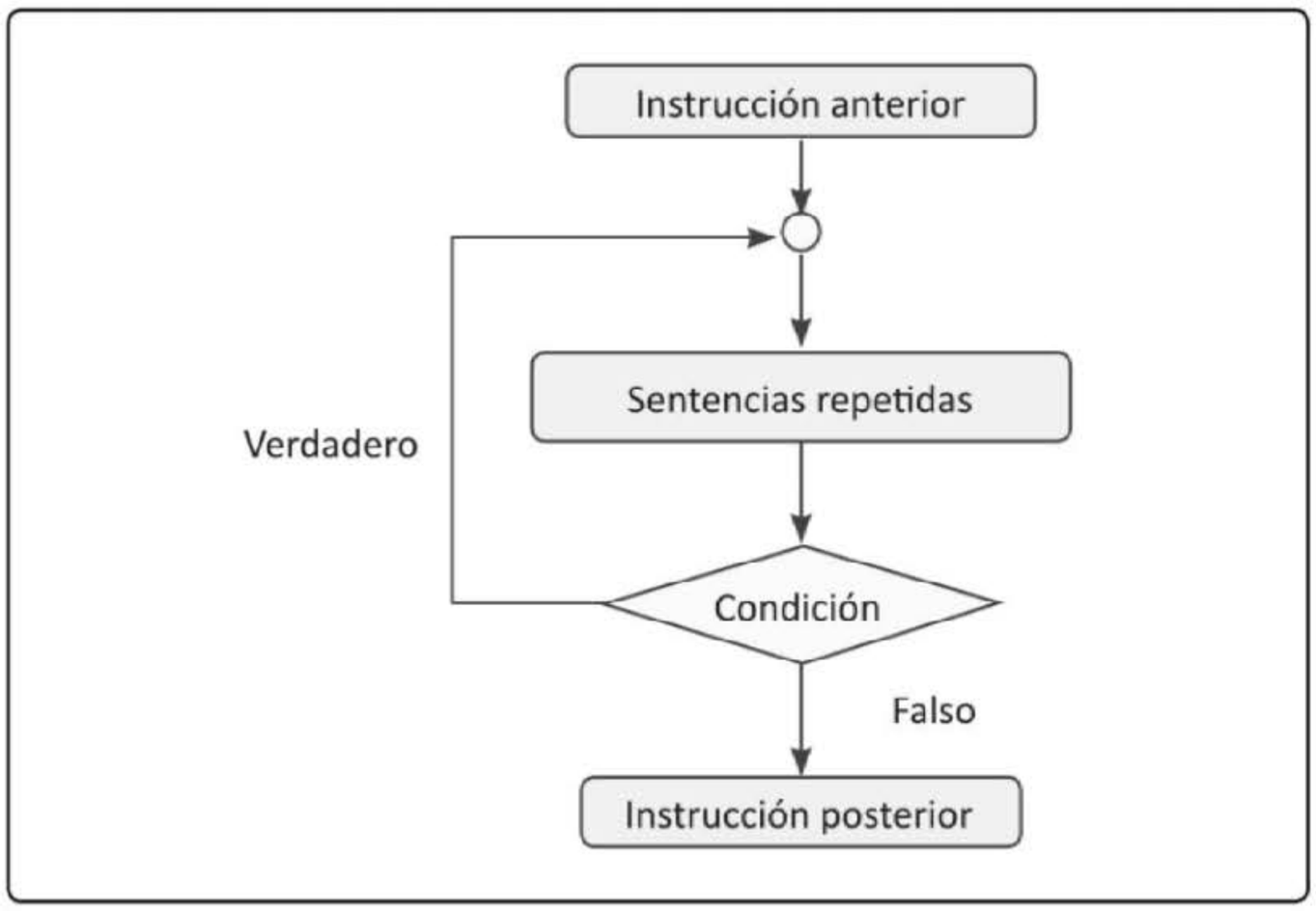
5.6 Sentencia Do - Loop While

La diferencia con While-End While radica en la forma cómo ingresa el primer valor al ciclo de repeticiones, pues cuando se usa Do-Loop While entra con un primer valor no evaluado, ya que según el formato este se evalúa al final de la sentencia. El ciclo de repeticiones seguirá solo cuando la condición especificada sea siempre verdadera, caso contrario finaliza el ciclo de repeticiones y la aplicación seguirá con la sentencia posterior.

A continuación, su formato:

```
Do
{
  'Sentencias repetidas
} Loop while (Condición)
```

Gráficamente:



Donde:

- **Sentencias repetidas**: Representa las sentencias que se ejecutarán mientras dure el ciclo de repeticiones, es decir, mientras que la condición resulte verdadera dichas sentencias se ejecutarán.
- **Condición**: Determina el ciclo de repeticiones, para este formato la condición se evalúa al final y si la condición siempre es verdadera se ejecutarán las sentencias repetidas, caso contrario, si la condición resulta falsa, el ciclo de repeticiones finalizará.

A continuación, algunos códigos que implementan el uso de la instrucción Do Loop… While

Listar 10 números aleatorios usando la estructura Do Loop While.

```
Dim i As Integer = 1
Do
    Dim Rnd As New Random()
    Dim n As Integer = Rnd.Next(1, 100)
    MessageBox.Show(n.ToString())
    i += 1
Loop While (i <= 10)
```

Se analiza el código mostrado:

- Se inicializa la variable I en uno para que el ciclo de repeticiones tenga un punto de partida.
- Se inicias el ciclo de repetición colocando la sentencia Do.

- Se declara el objeto RND de la clase Random, el cual permitirá generar números aleatorios.
- Se declara la variable N y registra un numero aleatorio generado desde el objeto RND y el método Next, los cuales tienen un rango de generación entre 1 y 100, es decir, el número aleatorio deberá encontrarse en dicho rango.
- Dicho número se muestra en pantalla mediante la sentencia **MessageBox.Show**.
- Antes de finalizar el ciclo de repeticiones se debe aumentar el valor de la variable I para que la condición de la sentencia Do While pueda finalizar; dicha condición debe ser verdadera para continuar en el ciclo.
- La condición indica que el valor I siempre debe ser inferior o igual para que genere ciclo de repeticiones; cuando I sea superior a 10 el ciclo finalizará.

Listar los divisores de un determinado número entero.

```
Dim n As Integer = 8
Dim i As Integer = 1

Do
    If (n Mod i = 0) Then
      MessageBox.Show(i.ToString())
    End If
    i += 1
Loop While (i <= n)
```

Se analiza el código mostrado:

- Se declara la variable N con un valor inicial de ocho; es a partir de este número del cual obtendremos sus divisores.
- Se inicializa la variable I en uno para que el ciclo de repeticiones tenga un punto de partida.
- Se inicia el ciclo de repetición colocando la sentencia Do.
- Se compara si al dividir la variable con el valor de I su resto (Mod) es cero, si es así entonces nos encontramos con un valor divisor y dicho valor se encuentra en la variable I.
- Dentro del ámbito de la sentencia If se imprime el valor de I, el cual representa el divisor del valor N solo si la condición resulta verdadera.
- Terminando la condición If se debe aumentar en un valor a la variable I para buscar los demás divisores del número N.
- Para finalizar, se condiciona que la variable I sea siempre inferior a N para generar ciclos de repeticiones controlados por dicha condición.

Listar los 10 primeros números pares de una serie máximo permitida por los números enteros.

```
Dim i As Integer = 1
Dim cp As Integer = 0
Do
    If (i Mod 2 = 0) Then
        MessageBox.Show(i.ToString())
        cp += 1
    End If

    If (cp = 10) Then Return
    i += 1
Loop While (i <= Integer.MaxValue)
```

Se analiza el código mostrado:

- Se inicializa la variable I en uno para que el ciclo de repeticiones tenga un punto de partida.
- Se declara la variable CP con un valor inicial de cero; su trabajo es contabilizar la cantidad de pares encontrados dentro del ciclo de repeticiones.
- Se inicia el ciclo de repeticiones con la sentencia Do.
- Dentro del bloque Do While, se compara si el resto de dividir la variable I con el valor dos es igual a cero; si resulta verdadero la condición se imprimirá el valor de I con la sentencia **MessageBox.Show**; allí mismo se contabiliza el par encontrado usando la sentencia CP + = 1.
- Luego, se compara si la variable contadora CP ya llego al máximo límite permitido: diez; si es así, se aplica la sentencia Return que permitirá finalizar la sentencia repetitiva.
- El aumento del valor en uno a la variable I se realiza para que el ciclo de repeticiones continúe su ciclo hasta que la variable CP sea igual a 10.

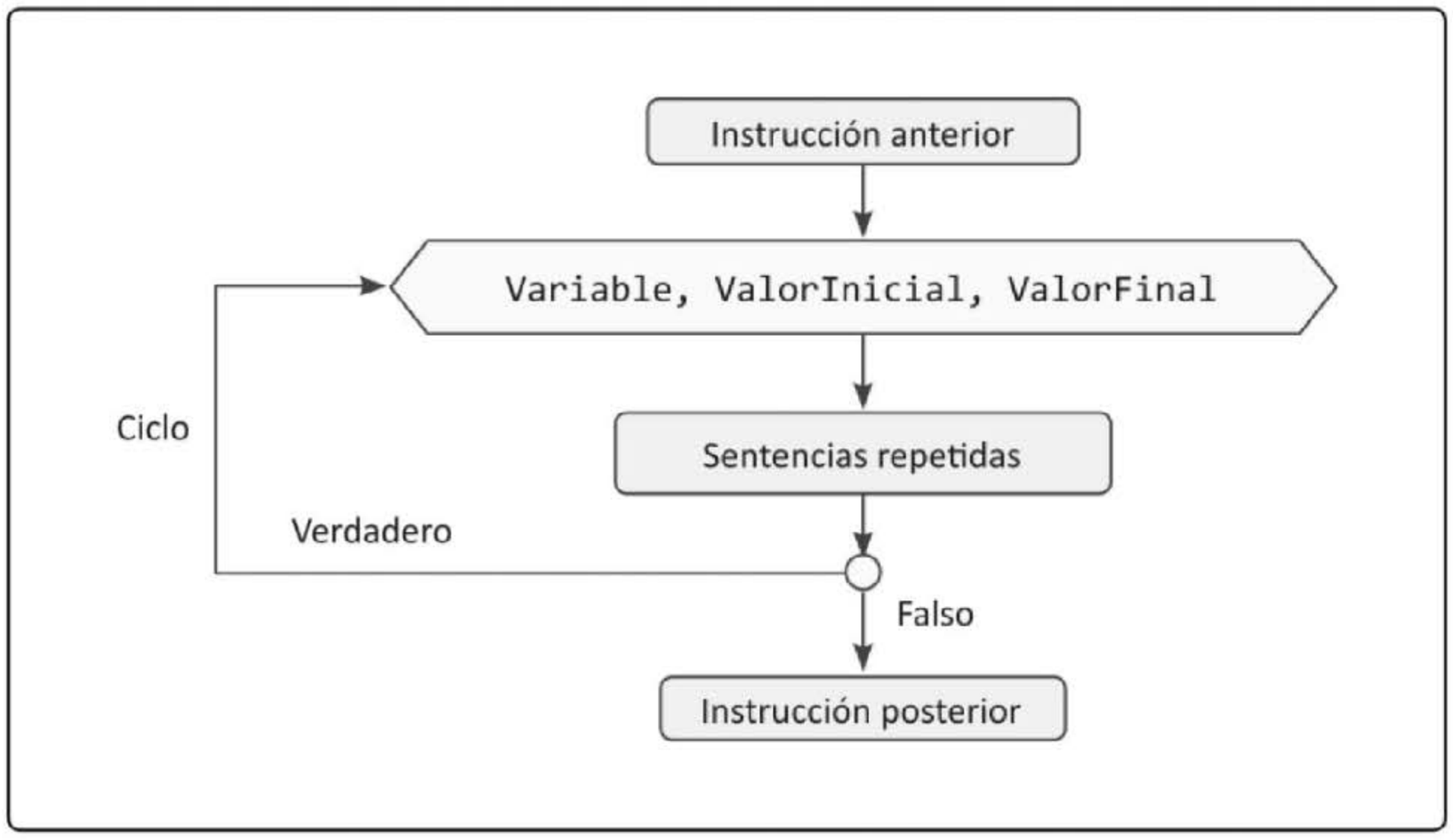
5.7 Sentencia For

Es la más usada dentro de las aplicaciones por su forma sencilla de implementar; su característica principal es que en una sola instrucción define todo el ciclo de repeticiones.

A continuación, su formato:

```
For variable = ValorInicial To ValorFinal
    'Sentencias repetidas
Next
```

Gráficamente:



Donde:

- **Variable**: Tiene la responsabilidad de contabilizar el número de vueltas del ciclo de repeticiones; normalmente se declara como I o J.
- **ValorInicial**: Es el valor de inicio para el ciclo de repeticiones; hay que tener en cuenta que dicho valor debe ser inferior al valor final si el ciclo se desarrollara en forma ascendente.
- **ValorFinal**: Es el valor que define el tope que tiene el ciclo de repeticiones, solo se debe tener cuenta que si el valor final es inferior al valor inicial, entonces el ciclo se desarrollará en forma descendente.
- **Sentencias repetidas**: Son las sentencias que se repetirán mientras el ciclo de repeticiones se mantenga.

A continuación, algunos códigos que implementan el uso de la sentencia For

Listar 10 números aleatorios usando la estructura For.

```
For i = 1 To 10
    Dim Rnd As New Random()
    Dim n As Integer = Rnd.Next(1, 100)
    MessageBox.Show(n.ToString())
Next
```

Listar los divisores de un determinado número entero.

```
Dim n As Integer = 9

For i = 1 To n
   If (n Mod i = 0) Then
      MessageBox.Show(i.ToString())
   End If
Next
```

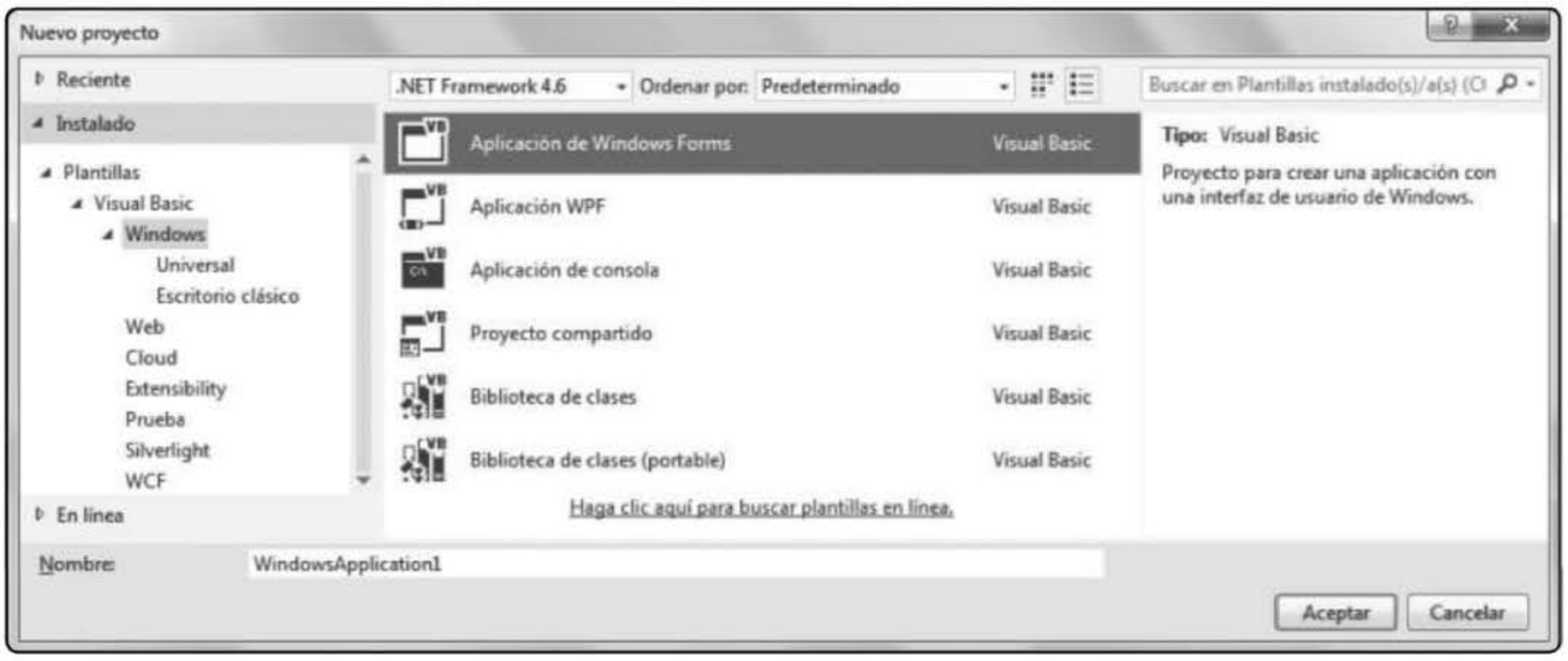
Listar los 10 primeros números pares de una serie máximo permitida por los números enteros.

```
Dim cp As Integer = 0

For i = 1 To Integer.MaxValue
   If (i Mod 2 = 0) Then
      MessageBox.Show(i.ToString())
      cp += 1
   End If
   If (cp = 10) Then Return
Next
```

5.8 Desarrollo de aplicaciones repetitivas

El desarrollo de aplicaciones repetitivas conlleva a usar las sentencias While y For en sus diferentes formas sin dejar de lado las sentencias condicionales; lo que se verá en estos casos es la integración de ambas sentencias. Para todos los casos se crea un proyecto de la siguiente manera:

- Siga la ruta **Archivo > Nuevo proyecto**.
- Seleccione la plantilla **Visual Basic > Windows > Aplicación de Windows Forms**.



- Escriba un nombre y seleccione **Aceptar**.

Caso desarrollado 1 Contadores - Control de vehículos en estacionamiento

La junta directiva de una urbanización ha decidido implementar un estacionamiento de vehículos ante la necesidad de sus vecinos de guardar sus automóviles o motocicletas. Se desea crear una aplicación que permita registrar el tipo de automóvil, número de placa, fecha y hora de ingreso del auto. Al final deberá contabilizar el total de automóviles, camiones, camionetas, motocicletas y ómnibus que ingresaron al estacionamiento.

Aspectos a considerar:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Usar declaración de variables globales para determinar los totales por cada tipo de vehículo.
- La fecha debe ser mostrada al iniciar la aplicación.
- La hora mostrada en la aplicación debe mantenerse interactiva, es decir, mostrarse como un reloj digital.
- Debe seleccionar un tipo de auto (automóvil, camión, camioneta, motocicleta y ómnibus), e ingresar el número de placa para poder registrarlo.
- El botón **Registrar** es el encargado de mostrar información al control ListView como tipo de auto, número de placa, fecha y hora exacta de ingreso al estacionamiento y, a la vez, debe estar asociado a la tecla <Enter>.
- Implementar la salida de la aplicación usando un cuadro de mensaje que se encuentre asociado al botón <Esc>.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmEstacionamiento btnRegistrar btnSalir CONTROL DE VEHICULOS - ESTACIONAMIENTO
Label1	Text	CONTROL DE VEHICULOS - ESTACIONAMIENTO
Label2	Text	TIPO DE AUTO
Label3	Text	N.o PLACA
Label4	Text	FECHA
Label5	(Name)	lblFecha
Label6	Text	HORA
Label7	(Name)	lblHora
Label8	Text	ESTADÍSTICA
ComboBox1	(Name)	cboTipo
TextBox1	(Name)	txtPlaca
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnSalir SALIR
ListBox1	(Name)	lstE
ListView1	(Name) GridLine Columns View	lvRegistro True Agregue 4 columnas y defina sus cabeceras, como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmEstacionamiento
    'Definición de variable GLOBALES
    Dim cAutomovil, cCamion, cCamioneta, cMotocicleta, cOmnibus As Integer

    Private Sub tHora_Tick(sender As Object, e As EventArgs) Handles tHora.Tick
        'Asignar la hora actual
        lblHora.Text = TimeString
    End Sub

    Private Sub frmEstacionamiento_Load(sender As Object, e As EventArgs)
        'Llenar de tipos de vehículos
        cboTipo.Items.Add("Automovil")
        cboTipo.Items.Add("Camion")
        cboTipo.Items.Add("Camioneta")
        cboTipo.Items.Add("Motocicleta")
        cboTipo.Items.Add("Omnibus")

        'Mostrando la fecha
        lblFecha.Text = DateString

        'Activando la hora
        tHora.Enabled = True
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'Capturar los valores
        Dim tipo As String = cboTipo.Text
        Dim placa As String = txtPlaca.Text
        Dim fecha As String = lblFecha.Text
        Dim hora As String = lblHora.Text

        'Contabilizar
        Select Case tipo
            Case "Automovil" : cAutomovil += 1
            Case "Camion" : cCamion += 1
            Case "Camioneta" : cCamioneta += 1
            Case "Motocicleta" : cMotocicleta += 1
            Case "Omnibus" : cOmnibus += 1
        End Select

        'Imprimir en la lista
        Dim fila As New ListViewItem(tipo)
        fila.SubItems.Add(placa)
        fila.SubItems.Add(fecha)
        fila.SubItems.Add(hora)
        lvRegistro.Items.Add(fila)
```

```
        'Imprimir conteos
        lstE.Items.Clear()
        lstE.Items.Add("Total de AUTOMOVIL: " & cAutomovil)
        lstE.Items.Add("Total de CAMION: " & cCamion)
        lstE.Items.Add("Total de CAMIONETA: " & cCamioneta)
        lstE.Items.Add("Total de MOTOCICLETA: " & cMotocicleta)
        lstE.Items.Add("Total de OMNIBUS: " & cOmnibus)
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Consideraciones:

- La definición de las variables globales para cAutomovil, cCamion, cCamioneta, cMotocicleta, cOmnibus conlleva al uso de dichas variables en cualquier parte de la aplicación. Considerar que si dichas variables son declaradas en el botón **Registrar** no permitirían el conteo, ya que, al ser declaradas, se inicializa su valor en cero y no se realizaría el conteo. En cambio, de forma global se declara una sola vez y el botón **Registrar** solo se encargaría de aumentar su valor.
- Para asignar un reloj digital en el formulario es necesario incorporarle el control Timer, el cual se encuentra en el cuadro de herramientas; en caso no lo encuentre rápidamente, escribir Timer en la parte de Búsqueda desde el cuadro de herramientas. Tener cuenta que el control Timer no es un objeto visible, por lo tanto, no se visualizará en el formulario sino debajo de él. Luego, cambiar de nombre al control por tHora y hacer clic sobre él para asignar el siguiente código:

```
Private Sub tHora_Tick(sender As Object, e As EventArgs) Handles tHora.Tick
        lblHora.Text = TimeString
End Sub
```

TimeString permite asignar la hora actual al control lblHora; hasta aquí solo se ha asignado la hora pero aún no se digitaliza, para esto colocar el siguiente código en el evento Load del formulario:

```
Private Sub frmEstacionamiento_Load(sender As Object, e As EventArgs)
        tHora.Enabled = True
End Sub
```

Para lograr que el cuadro combinado muestre los diferentes tipos de vehículos y, además, la fecha actual se muestre al iniciar la aplicación, asignar el siguiente código en el evento Load del formulario:

```
Private Sub frmEstacionamiento_Load(sender As Object, e As EventArgs)
        cboTipo.Items.Add("Automovil")
        cboTipo.Items.Add("Camion")
        cboTipo.Items.Add("Camioneta")
        cboTipo.Items.Add("Motocicleta")
        cboTipo.Items.Add("Omnibus")

        'Mostrando la fecha
        lblFecha.Text = DateString
End Sub
```

En el botón **Registrar** se captura todos los datos mostrados en el formulario con el siguiente código:

```
Dim tipo As String = cboTipo.Text
Dim placa As String = txtPlaca.Text
Dim fecha As String = lblFecha.Text
Dim hora As String = lblHora.Text
```

La idea principal es tener todos los datos en forma de cadena y así poder imprimirlo en el control ListView. Luego, pasar a realizar los conteos, para esto se debe comparar qué valor se encuentra en la variable Tipo. Por lo tanto, usar la sentencia selectiva múltiple Select Case para dicha evaluación, tal como se muestra en el siguiente código:

```
Select Case tipo
        Case "Automovil" : cAutomovil += 1
        Case "Camion" : cCamion += 1
        Case "Camioneta" : cCamioneta += 1
        Case "Motocicleta" : cMotocicleta += 1
        Case "Omnibus" : cOmnibus += 1
End Select
```

Según sea el caso, aumentará el contador solo si el valor de la variable tipo es igual a una de las opciones especificadas; seguidamente, se lleva toda la información capturada al control ListView; tal como se muestra en el siguiente código:

```
Dim fila As New ListViewItem(tipo)
fila.SubItems.Add(placa)
fila.SubItems.Add(fecha)
fila.SubItems.Add(hora)
lvRegistro.Items.Add(fila)
```

Se crea el objeto fila de la clase ListViewItem para poder enviar información fila por fila; finalmente, enviamos la información de los conteos al control ListBox. Siempre se debe iniciar limpiando el control ListBox, ya que es necesario mostrar información actualizada en dicho control y el símbolo & permite unir dos elementos en programación, tal como se muestra en el siguiente código:

```
lstE.Items.Clear()
lstE.Items.Add("Total de AUTOMOVIL: " & cAutomovil)
lstE.Items.Add("Total de CAMION: " & cCamion)
lstE.Items.Add("Total de CAMIONETA: " & cCamioneta)
lstE.Items.Add("Total de MOTOCICLETA: " & cMotocicleta)
lstE.Items.Add("Total de OMNIBUS: " & cOmnibus)
```

Caso desarrollado 2 Acumuladores - Control de entradas de concierto

Una empresa de eventos y espectáculos necesita una aplicación que permita controlar la venta de entradas de un determinado concierto, para lo cual se debe registrar el tipo de entrada y la cantidad de entradas solicitadas por los usuarios.

Al final se debe obtener una lista de entradas con su respectiva cantidad, costo y subtotal, además de mostrar las estadísticas de ventas y el monto total acumulado por cada tipo de entrada.

Considerar los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Use declaración de variables globales para los montos acumulados según el tipo de entrada.
- Debe seleccionar el tipo de entrada y la cantidad que los usuarios solicitan.
- Los costos de los tipos de entrada se dan bajo la siguiente tasa:

Tipo de entrada	Costo $
Occidente	120.00
Oriente	90.00
Norte	40.00
VIP	220.00

- El costo debe mostrarse al seleccionar un tipo de entrada desde un cuadro combinado.
- Mostrar todos los montos con el formato de moneda.
- Mostrar el total recaudado del registro, el cual resulta del monto acumulado de los subtotales.
- El botón **Comprar** es el encargado de mostrar información al control ListView, como el tipo de entrada, cantidad comprada, costo por tipo de entrada y el subtotal de la venta. Asimismo, debe mostrar el monto total acumulado de los subtotales y las estadísticas solicitadas por el caso. Dicho botón debe estar asociado a la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón <Esc>.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmEntrada btnComprar btnSalir CONTROL DE ENTRADAS - CONCIERTO
Label1	Text	CONTROL DE ENTRADAS – CONCIERTO

Control	Propiedad	Valor
Label2	Text	TIPO DE ENTRADA
Label3	Text	COSTO $
Label4	Text	CANTIDAD
Label5	Text	TOTAL RECAUDADO $
Label6	(Name)	lblRecaudado
Label7	Text	ESTADÍSTICA
ComboBox1	(Name)	cboTipo
TextBox1	(Name)	txtCantidad
Button1	(Name) Text	btnComprar COMPRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListBox1	(Name)	lstE
ListView1	(Name) GridLine Columns View	lvVenta True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmEntrada
    'Definición de variables GLOBALES
    Dim precio, recaudado As Double
    Dim aOccidente, aOriente, aNorte, aVIP As Double

    Private Sub frmEntrada_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Llenando los tipos de entrada
        cboTipo.Items.Add("Occidente")
        cboTipo.Items.Add("Oriente")
        cboTipo.Items.Add("Norte")
        cboTipo.Items.Add("VIP")
    End Sub
```

```
Private Sub cboTipo_SelectedIndexChanged(sender As Object, e As EventArgs)
    'Asignando el precio según el tipo de entrada
    Select Case cboTipo.Text
        Case "Occidente" : precio = 120
        Case "Oriente" : precio = 90
        Case "Norte" : precio = 40
        Case "VIP" : precio = 220
    End Select
    'Mostrando el precio
    lblCosto.Text = precio.ToString("0.00")
End Sub

Private Sub btnComprar_Click(sender As Object, e As EventArgs)
    'Capturar los datos
    Dim costo As Double = Double.Parse(lblCosto.Text)
    Dim cantidad As Integer = Integer.Parse(txtCantidad.Text)

    'Calcular el monto
    Dim subtotal As Double = costo * cantidad

    'Imprimir en la lista
    Dim fila As New ListViewItem(cboTipo.Text)
    fila.SubItems.Add(cantidad.ToString("0.00"))
    fila.SubItems.Add(costo.ToString("0.00"))
    fila.SubItems.Add(subtotal.ToString("0.00"))
    lvEntradas.Items.Add(fila)

    'Calcular e imprimir lo recaudado
    recaudado += subtotal
    lblRecaudado.Text = recaudado.ToString("0.00")

    'Calcular monto acumulado por tipo e imprimirlos
    Select Case cboTipo.Text
        Case "Occidente" : aOccidente += subtotal
        Case "Oriente" : aOriente += subtotal
        Case "Norte" : aNorte += subtotal
        Case "VIP" : aVIP += subtotal
    End Select

    lstE.Items.Clear()
    lstE.Items.Add("MONTO TOTAL ACUMULADO POR OCCIDENTE:"& aOccidente.ToString("C"))
    lstE.Items.Add("MONTO TOTAL ACUMULADO POR ORIENTE: " & aOriente.ToString("C"))
    lstE.Items.Add("MONTO TOTAL ACUMULADO POR NORTE: " & aNorte.ToString("C"))
    lstE.Items.Add("MONTO TOTAL ACUMULADO POR VIP: " & aVIP.ToString("C"))
End Sub

Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
    cboTipo.SelectedIndex = -1
    lblCosto.Text = ""
```

```
        txtCantidad.Clear()
        cboTipo.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Consideraciones:

- La variable Recaudado debe ser global porque necesita acumular los montos resultantes de los subtotales. Asimismo, se debe declarar de forma global los acumuladores según el tipo de entrada, tal como se muestra en el siguiente código:

```
Dim recaudado As Double
Dim aOccidente, aOriente, aNorte, aVIP As Double
```

- Al iniciar la aplicación se debe mostrar los diferentes tipos en el control cboTipo, por lo tanto, se debe colocar el siguiente código en el evento load del formulario:

```
Private Sub frmEntrada_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Llenando los tipos de entrada
        cboTipo.Items.Add("Occidente")
        cboTipo.Items.Add("Oriente")
        cboTipo.Items.Add("Norte")
        cboTipo.Items.Add("VIP")
End Sub
```

- Para mostrar el costo según el tipo de entrada se debe programar en el control cboTipo, para esto, solo hacer doble clic sobre el control y coloque el siguiente código:

```
Private Sub cboTipo_SelectedIndexChanged(sender As Object, e As EventArgs)
        'Asignando el precio según el tipo de entrada
        Dim precio As Double
```

```
        Select Case cboTipo.Text
            Case "Occidente" : precio = 120
            Case "Oriente" : precio = 90
            Case "Norte" : precio = 40
            Case "VIP" : precio = 220
        End Select
        'Mostrando el precio
        lblCosto.Text = precio.ToString("0.00")
End Sub
```

- Para mostrar toda la información referente a los cálculos y a las estadísticas, se tiene que realizar en el botón btnComprar, de tal forma que se obtenga los valores ingresados por el usuario, como el costo según el tipo de entrada y la cantidad de boletos adquiridos por el cliente:

```
Dim costo As Double = Double.Parse(lblCosto.Text)
Dim cantidad As Integer = Integer.Parse(txtCantidad.Text)
```

Luego, realizar el cálculo del subtotal, el cual resulta de multiplicar el costo de la entrada por la cantidad solicitada por el cliente `Dim subtotal As Double = costo * cantidad`.

Seguidamente, realizar la impresión de los montos en el control ListView; se debe crear un objeto de la clase ListViewItem y asignar como primer valor de la lista al tipo de entrada, tal como se muestra en el siguiente código:

```
Dim fila As New ListViewItem(cboTipo.Text)
fila.SubItems.Add(cantidad.ToString("0.00"))
fila.SubItems.Add(costo.ToString("0.00"))
fila.SubItems.Add(subtotal.ToString("0.00"))
lvEntradas.Items.Add(fila)
```

Ahora, es necesario realizar la acumulación de los subtotales en la variable global recaudado. Recordar que la variable Subtotal proviene del Producto entre Costo y Cantidad y que dicho valor debe ser acumulado en la variable Recaudado y es justamente este último valor el cual debe imprimirse, tal como se muestra en el siguiente código:

```
recaudado += subtotal
lblRecaudado.Text = recaudado.ToString("0.00")
```

Para realizar las acumulaciones por tipo de entrada, determinar el tipo de entrada mediante la sentencia Select Case, de tal forma que por cada tipo el acumulador aumente su valor, tal como se muestra en el siguiente código:

```
Select Case cboTipo.Text
            Case "Occidente" : aOccidente += subtotal
            Case "Oriente" : aOriente += subtotal
            Case "Norte" : aNorte += subtotal
            Case "VIP" : aVIP += subtotal
        End Select
```

Finalmente, imprimir los valores acumuladores en el control ListBox, tal como se muestra en el siguiente código:

```
lstE.Items.Clear()
lstE.Items.Add("MONTO TOTAL ACUMULADO POR OCCIDENTE:"& aOccidente.ToString("C"))
lstE.Items.Add("MONTO TOTAL ACUMULADO POR ORIENTE: " & aOriente.ToString("C"))
lstE.Items.Add("MONTO TOTAL ACUMULADO POR NORTE: " & aNorte.ToString("C"))
lstE.Items.Add("MONTO TOTAL ACUMULADO POR VIP: " & aVIP.ToString("C"))
```

Se debe tener en cuenta que para mostrar la información resultante, el control ListBox debe iniciar limpio y cada vez que se imprima un valor numérico debe asignarle la función ToString para especificar el formato de moneda. Para esto se coloca el parámetro C en la función ToString.

Caso desarrollado 3 While… End While - Financiamiento de vehículo Simulador

Una empresa de venta de automóviles ofrece una serie de facilidades para la adquisición de nuevos vehículos. Es así que necesitan una aplicación que permita ingresar el monto real del vehículo, el cual será financiado; la aplicación mostrará los tipos de financiamiento, el interés mensual, la cuota inicial y el número de letras de todos los tipos de financiamiento que ofrece la empresa. Luego, al seleccionar un tipo de financiamiento deberá mostrar el número de letras, las fechas de pago y los montos mensuales en caso el cliente acepte el financiamiento.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Ingresar el monto a financiar, tenga en cuenta que dicho valor genera todos los resultados en la aplicación.
- Los porcentajes, cuota inicial y número de letras se dan bajo la siguiente tabla:

Tipo de financiamiento	Interés mensual	Cuota inicial	Número de letras
Personal	5 %	60 %	2
Banco	10 %	50 %	4
Fondo Colectivo	15 %	35 %	6

- Mostrar todos los montos con el formato de moneda.
- El botón **Simular** es el encargado de mostrar información al control ListView como todos los tipos de financiamiento, los montos correspondientes al interés mensual, cuota inicial y número de letras según el tipo de financiamiento. Dicho botón debe estar asociado a la tecla <Enter>.
- En un cuadro combinado mostrar los tipos de financiamiento y que al seleccionar uno de ellos muestre en una lista el número de cuota, fecha y monto según el tipo de financiamiento mostrado en la lista anterior.

- Con respecto a las fechas, se dan de acuerdo al número de cuotas que le corresponde al tipo de financiamiento. Además, debe mostrar las fechas próximas de pago usando la función DateAdd.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón <Esc>.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmFinanciamiento btnSimular btnSalir FINANCIAMIENTO DE VEHICULO - SIMULADOR
Label1	Text	FINANCIAMIENTO DE VEHICULO – SIMULADOR
Label2	Text	MONTO A FINANCIAR
Label3	Text	MONTOS SEGÚN TIPO DE FINANCIAMIENTO
Label4	Text	TIPO DE FINANCIAMIENTO
TextBox1	(Name)	txtMonto
ListView1	(Name) GridLine Columns View	lvMontos True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details
ListView2	(Name) GridLine Columns View	lvCuotas True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details
ComboBox1	(Name)	cboTipo
Button1	(Name) Text	btnSimular SIMULAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR

El script de la solución es:

```
Public Class frmFinanciamiento
    Private Sub frmFinanciamiento_Load(sender As Object, e As EventArgs)
        'Llenando el combo de tipos de financiamiento
        cboTipo.Items.Add("Personal")
        cboTipo.Items.Add("Banco")
        cboTipo.Items.Add("Fondo Colectivo")

    End Sub

    Private Sub btnSimular_Click(sender As Object, e As EventArgs)
        'Capturar el monto
        Dim monto As Double = Double.Parse(txtMonto.Text)

        'Calculando el interés
        Dim iPersonal As Double = monto * 0.05
        Dim iBanco As Double = monto * 0.1
        Dim iFondoColectivo As Double = monto * 0.15

        'Calculando la cuota inicial
        Dim ciPersonal As Double = (monto + iPersonal) * 0.6
        Dim ciBanco As Double = (monto + iBanco) * 0.5
        Dim ciFondoColectivo As Double = (monto + iFondoColectivo) * 0.35

        'Calculando el numero de letras
        Dim lPersonal As Integer = 2
        Dim lBanco As Integer = 4
        Dim lFondoColectivo As Integer = 6

        'Imprimir en la lista
        Dim fila1 As New ListViewItem("Personal")
        fila1.SubItems.Add(iPersonal.ToString("C"))
        fila1.SubItems.Add(ciPersonal.ToString("C"))
        fila1.SubItems.Add(lPersonal.ToString("0.00"))
        lvMontos.Items.Add(fila1)

        Dim fila2 As New ListViewItem("Banco")
        fila2.SubItems.Add(iBanco.ToString("C"))
        fila2.SubItems.Add(ciBanco.ToString("C"))
        fila2.SubItems.Add(lBanco.ToString("0.00"))
        lvMontos.Items.Add(fila2)

        Dim fila3 As New ListViewItem("Fondo Colectivo")
        fila3.SubItems.Add(iFondoColectivo.ToString("C"))
        fila3.SubItems.Add(ciFondoColectivo.ToString("C"))
        fila3.SubItems.Add(lFondoColectivo.ToString("0.00"))
        lvMontos.Items.Add(fila3)
    End Sub
```

```
    Private Sub cboTipo_SelectedIndexChanged(sender As Object, e As EventArgs)
        Dim letras As Integer
        Dim monto As Double = Double.Parse(txtMonto.Text)
        Dim interes, cuotaInicial As Double
        Dim i As Integer = 0
        While i < lvMontos.Items.Count
            If lvMontos.Items.Item(i).SubItems(0).Text = cboTipo.Text Then
                letras = lvMontos.Items.Item(i).SubItems(3).Text
                interes = lvMontos.Items.Item(i).SubItems(1).Text
                cuotaInicial = lvMontos.Items.Item(i).SubItems(2).Text
            End If
            i += 1
        End While

        Dim n As Integer = 1
        Dim montoMensual As Double = (monto + interes - cuotaInicial) / letras
        Dim fechaActual As DateTime = DateTime.Now.ToShortDateString()
        lvCuotas.Items.Clear()
        While n <= letras
            Dim fecha As DateTime = DateAdd(DateInterval.Month, n, fechaActual)
            Dim fila As New ListViewItem(n.ToString())
            fila.SubItems.Add(fecha.ToShortDateString())
            fila.SubItems.Add(montoMensual.ToString("C"))
            lvCuotas.Items.Add(fila)
            n += 1
        End While

    End Sub

    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtMonto.Clear()
        cboTipo.SelectedIndex = -1
        lvCuotas.Items.Clear()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Consideraciones:

- Al iniciar la aplicación, los tipos de financiamiento deberán estar agregados en el Combobox, para lo cual se debe programar en el evento Load del formulario (haga doble clic sobre el fondo del formulario), tal como se muestra en el siguiente código:

```
Private Sub frmFinanciamiento_Load(sender As Object, e As EventArgs)
        cboTipo.Items.Add("Personal")
        cboTipo.Items.Add("Banco")
        cboTipo.Items.Add("Fondo Colectivo")
End Sub
```

- Para iniciar la simulación, programar en el botón btnSimular e iniciar capturando el monto a financiar Dim monto As Double = Double.Parse(txtMonto.Text). Luego, se debe realizar los cálculos del interés por cada tipo de financiamiento, tal como se muestra en el siguiente código:

```
Dim iPersonal As Double = monto * 0.05
Dim iBanco As Double = monto * 0.1
Dim iFondoColectivo As Double = monto * 0.15
```

- Se debe tener en cuenta que los resultados del interés se deben al monto a financiar capturado. Luego, calcular la cuota inicial por cada tipo de financiamiento, tal como se muestra en el siguiente código:

```
Dim ciPersonal As Double = (monto + iPersonal) * 0.6
Dim ciBanco As Double = (monto + iBanco) * 0.5
Dim ciFondoColectivo As Double = (monto + iFondoColectivo) * 0.35
```

- Para determinar el número de letras se implementa el siguiente código:

```
Dim lPersonal As Integer = 2
Dim lBanco As Integer = 4
Dim lFondoColectivo As Integer = 6
```

- Luego, para imprimir los resultados tanto del interés como de la cuota inicial y las letras, crear tres objetos de la clase ListViewItem, tal como se muestra en el siguiente código:

```
Dim fila1 As New ListViewItem("Personal")
fila1.SubItems.Add(iPersonal.ToString("C"))
```

```
fila1.SubItems.Add(ciPersonal.ToString("C"))
fila1.SubItems.Add(lPersonal.ToString("0.00"))
lvMontos.Items.Add(fila1)

Dim fila2 As New ListViewItem("Banco")
fila2.SubItems.Add(iBanco.ToString("C"))
fila2.SubItems.Add(ciBanco.ToString("C"))
fila2.SubItems.Add(lBanco.ToString("0.00"))
lvMontos.Items.Add(fila2)

Dim fila3 As New ListViewItem("Fondo Colectivo")
fila3.SubItems.Add(iFondoColectivo.ToString("C"))
fila3.SubItems.Add(ciFondoColectivo.ToString("C"))
fila3.SubItems.Add(lFondoColectivo.ToString("0.00"))
lvMontos.Items.Add(fila3)
```

- Uno de los requerimientos de la aplicación indica que al seleccionar un tipo de financiamiento este debe mostrar información como el número de letras, las fecha de los posibles pagos y los montos mensuales que debe realizar un determinado cliente. Para esto, implementar el siguiente código en el evento SelectedIndexChanged del control cboTipo; para llegar a dicho evento solo hacer clic sobre dicho control e iniciar capturando el monto ingresado, tal como se muestra en el siguiente código:

```
Dim monto As Double = Double.Parse(txtMonto.Text)
```

Ahora, la idea es encontrar el tipo de financiamiento seleccionado por el usuario en el control ListView. Recorrer por todas las filas del control ListView lvMontos, para esto usar la sentencia While i < lvMontos.Items.Count y desde aquí generar ciclos repetitivos hasta que la variable I, que inició en cero, sea mayor al número de fila que contiene el control; la propiedad Count permite determinar el número total de filas del control ListView.

Dentro del ciclo de repeticiones, buscar qué fila contiene el valor del tipo de financiamiento seleccionado; para lograrlo se debe especificar una búsqueda solo por la primera columna de la lista, usando la siguiente sentencia:

```
lvMontos.Items.Item(i).SubItems(0).Text
```

I representa la posición de la fila.

Cero representa a la primera columna de la lista, la cual corresponde a los tipos de financiamiento.

Entonces, usando la condición If, verificar si el tipo seleccionado se encuentra en la primera columna de la lista con la sentencia If lvMontos.Items.Item(i).SubItems(0). Text = cboTipo.Text Then; una vez encontrado el tipo se debe capturar todos sus valores usando el siguiente código:

```
        letras = lvMontos.Items.Item(i).SubItems(3).Text
        interes = lvMontos.Items.Item(i).SubItems(1).Text
        cuotaInicial = lvMontos.Items.Item(i).SubItems(2).Text
```

Para obtener el número de letras se hace referencia a la posición 3 de la lista, mientras que el monto de interés se encuentra en la posición 1 y la cuota inicial desde la posición 2. Ver la siguiente especificación que se asigna en el control ListView.

TIPO DE FINANCIAMIENTO	INTERES MENSUAL	CUOTA INICIAL	NUM. LETRAS
Personal	$2,500.00	$31,500.00	2.00
Banco	$5,000.00	$27,500.00	4.00
Fondo Colectivo	$7,500.00	$20,125.00	6.00

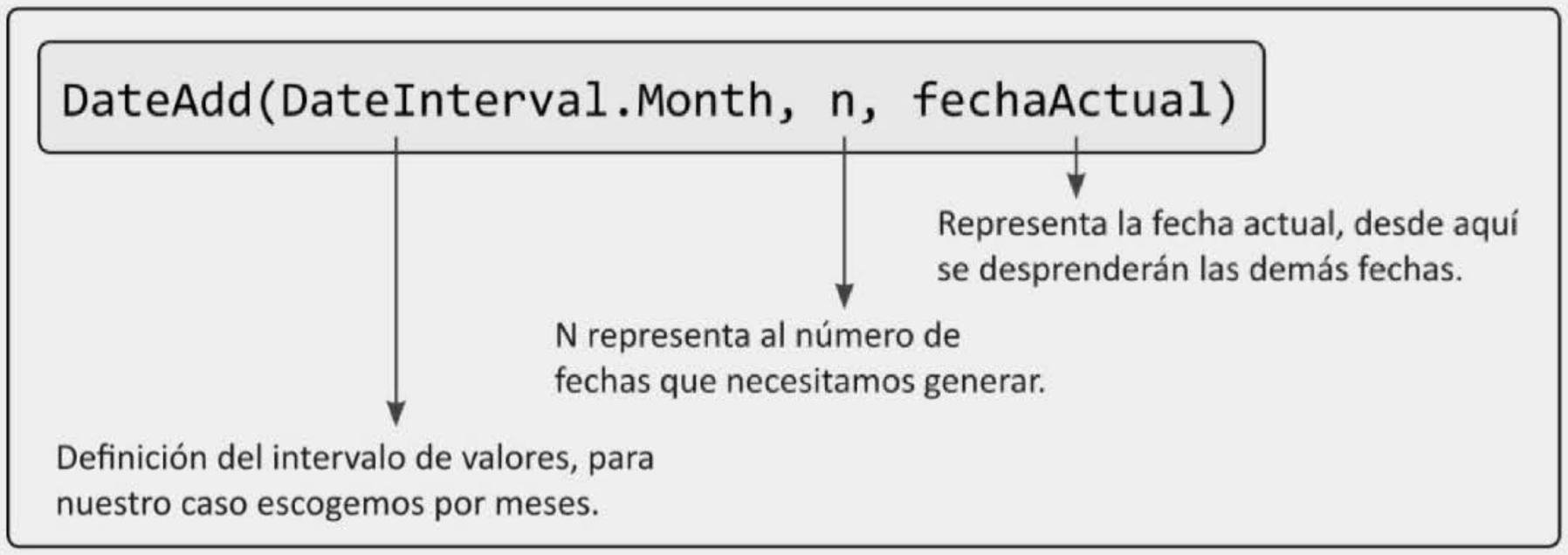
SubItems(0) SubItems(1) SubItems(2) SubItems(3)

El monto mensual resulta de sumar el monto financiado y el interés obtenido menos la cuota inicial dividido entre el número de letras según el tipo de financiamiento `Dim montoMensual As Double = (monto + interés - cuotaInicial) / letras`.

- Para determinar las fechas posibles de pago se debe iniciar por capturar la fecha actual con la sentencia Dim fechaActual As DateTime = DateTime.Now. ToShortDateString(). Luego, se iniciará el ciclo de repeticiones de tal forma que genere ciclos de acuerdo al número de letras, tal como se muestra el siguiente código:

```
While n <= letras
        Dim fecha As DateTime = DateAdd(DateInterval.Month, n, fechaActual)
        Dim fila As New ListViewItem(n.ToString())
        fila.SubItems.Add(fecha.ToShortDateString())
        fila.SubItems.Add(montoMensual.ToString("C"))
        lvCuotas.Items.Add(fila)
        n += 1
End While
```

- Para obtener las fechas posteriores a la fecha de pago usar la función DateAdd con un intervalo de un mes, para que las fechas se muestren por cada mes según la cantidad de letras. La función DateAdd cuenta con los siguientes parámetros:



No olvidar que se deben crear fechas tanto número de letras sea seleccionado; por lo tanto, la declaración de la fecha debe realizarse dentro del ciclo de repeticiones.

Caso desarrollado 4 Do… Loop While - Control de registro de nuevos libros

Un centro de información adquiere libros nuevos y usados a fin de año, de diferentes entidades, en forma de donación; en visto de esto, necesitan una aplicación que permita controlar el registro de dichos libros como el título, número de páginas, editorial, categoría, costo y año de edición del libro. Al final, mostrar algunas estadísticas solicitadas por la administración como el número total de libros adquiridos según la categoría, determinar cuál es el título del libro cuyo costo sea mayor a los demás, cuál es el título del libro cuyo número de páginas es menor a las demás y, finalmente, determinar el costo acumulado según la editorial.

Considerar los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Mostrar todos los montos con el formato de moneda.
- El botón **Registrar** es el encargado de mostrar información al control ListView como el título, editorial, categoría, año de edición, número de páginas y el costo del libro registrado; además de mostrar las estadísticas solicitadas por el caso. Dicho botón debe estar asociado a la tecla <Enter>.
- El botón **Cancelar** es el encargado de limpiar el contenido de los controles, para un nuevo registro de datos del libro.
- Implementar la salida de la aplicación usando un cuadro de mensaje y que se encuentre asociado al botón <Esc>.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmEditorial btnRegistrar btnSalir CONTROL DE REGISTRO DE NUEVOS LIBROS
Label1	Text	CONTROL DE REGISTRO DE NUEVOS LIBROS
Label2	Text	TITULO
Label3	Text	NRO. PÁGINAS
Label4	Text	EDITORIAL
Label5	Text	CATEGORÍA
Label6	(Name)	COSTO $
Label7	Text	AÑO EDICIÓN
TextBox1	(Name)	txtTitulo
TextBox2	(Name)	txtPaginas
TextBox3	(Name)	txtCosto
TextBox4	(Name)	txtAño

Control	Propiedad	Valor
ComboBox1	(Name)	cboEditorial
ComboBox2	(Name)	cboCategoria
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLine Columns View	lvLibros True Agregue 6 columnas y defina sus cabeceras como se muestra en el modelo. Details
ListBox1	(Name)	lstE

El script de la solución es:

```
Public Class frmEditorial
    Private Sub frmEditorial_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Llenar las editoriales
        With cboEditorial.Items
            .Add("Montena")
            .Add("Planeta")
            .Add("ViaMagna")
            .Add("Macro")
        End With

        'Llenar las categorias
        With cboCategoria.Items
            .Add("Superación personal")
            .Add("Literatura")
            .Add("Ciencias")
            .Add("Informática")
        End With
    End Sub

    Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
        'Obtener los datos del formulario
        Dim titulo As String = txtTitulo.Text
        Dim paginas As Integer = Integer.Parse(txtPaginas.Text)
```

```
        Dim editorial As String = cboEditorial.Text
        Dim categoria As String = cboCategoria.Text
        Dim costo As Double = Double.Parse(txtCosto.Text)
        Dim edición As Integer = Integer.Parse(txtAño.Text)

        'Enviar los datos del libro a la lista
        Dim fila As New ListViewItem(titulo)
        fila.SubItems.Add(editorial)
        fila.SubItems.Add(categoria)
        fila.SubItems.Add(edicion.ToString())
        fila.SubItems.Add(paginas.ToString())
        fila.SubItems.Add(costo.ToString("0.00"))
        lvLibros.Items.Add(fila)

        'Calculando las estadísticas
        'Determinar el numero de libros por categoría
        Dim i As Integer = 0
        Dim cSuperacion, cLiteratura, cCiencias, cInformatica As Integer
        Do
            Dim cat As String = lvLibros.Items.Item(i).SubItems(2).Text
            Select Case cat
                Case "Superación personal" : cSuperacion += 1
                Case "Literatura" : cLiteratura += 1
                Case "Ciencias" : cCiencias += 1
                Case "Informática" : cInformatica += 1
            End Select
            i += 1
        Loop While i < lvLibros.Items.Count

        'Determinar el libro con mayor costo
        i = 0
        Dim pos As Integer
        Dim mayor As Double = 0
        Do
            Dim cos As String = lvLibros.Items.Item(i).SubItems(5).Text
            If cos > mayor Then
                mayor = cos
                pos = i
            End If
            i += 1
         Loop While i < lvLibros.Items.Count
        Dim libroCostoMayor As String = lvLibros.Items.Item(pos).SubItems(0).Text

         'Determinar el libro con menor numero de paginas
         i = 0
         Dim menor As Integer = Integer.MaxValue
         Do
```

```
        Dim pag As Integer = lvLibros.Items.Item(i).SubItems(4).Text
        If pag < menor Then
            menor = pag
            pos = i
        End If
        i += 1
    Loop While i < lvLibros.Items.Count
    Dim libroMenorPagina As String = lvLibros.Items.Item(pos).SubItems(0).Text

    'Determinar los costos registrados por editorial
    i = 0
    Dim cMontena, cPlaneta, cVia, cMacro As Double
    Do
     Dim edi As String = lvLibros.Items.Item(i).SubItems(1).Text
     Select Case ediCase "Montena":cMontena+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
     Case "Planeta":cPlaneta+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
     Case "ViaMagna":cVia+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
      Case "Macro":cMacro+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
     End Select
        i += 1
    Loop While i < lvLibros.Items.Count

    'Imprimir las estadísticas
    With lstE.Items
        .Clear()
        .Add("** ESTADISTICA DE REGISTRO DE LIBROS **")
        .Add("NUMERO DE LIBROS POR CATEGORIA")
        .Add("SUPERACION PERSONAL: " & cSuperacion.ToString("0.00"))
        .Add("LITERATURA:" & cLiteratura.ToString("0.00"))
        .Add("CIENCIAS:" & cCiencias.ToString("0.00"))
        .Add("INFORMÁTICA:" & cInformatica.ToString("0.00"))
        .Add("-------------------------------------------------")
        .Add("LIBRO DE MAYOR COSTO:" & libroCostoMayor)
        .Add("-------------------------------------------------")
        .Add("LIBRO CON MENOR NUMERO DE PÁGINAS:" & libroMenorPagina)
        .Add("-------------------------------------------------")
        .Add("COSTO ACUMULADO MONTENA:" & cMontena.ToString("C"))
        .Add("COSTO ACUMULADO PLANETA:" & cPlaneta.ToString("C"))
        .Add("COSTO ACUMULADO VIA MAGNA:" & cVia.ToString("C"))
        .Add("COSTO ACUMULADO MACRO:" & cMacro.ToString("C"))
    End With

    'Limpiando los controles para un nuevo registro
    btnCancelar_Click(sender, e)
End Sub
```

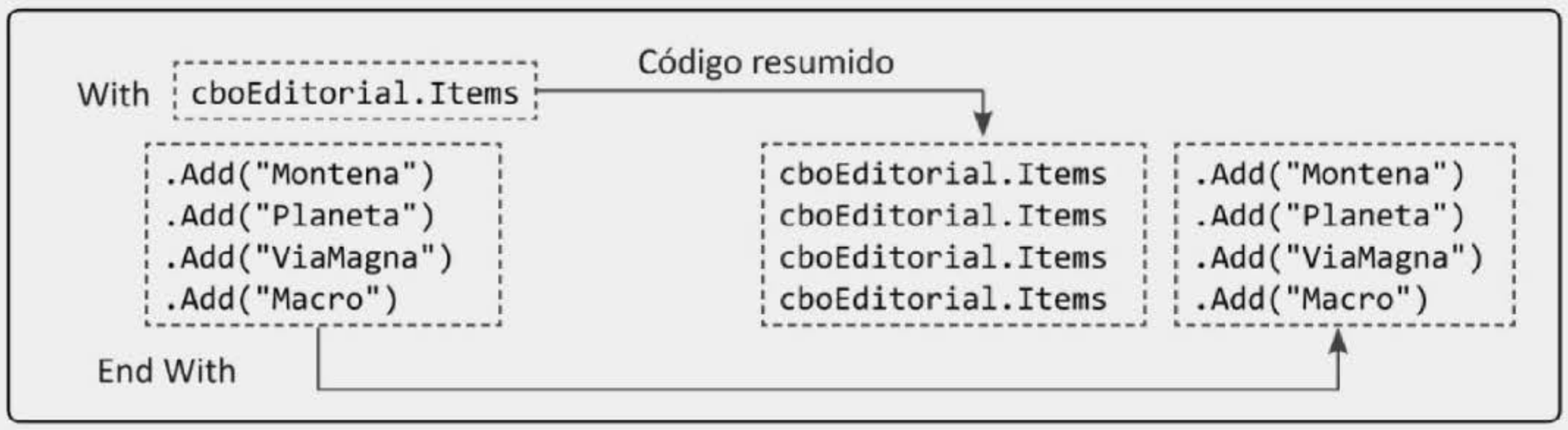
```
    Private Sub btnCancelar_Click(sender As Object, e As EventArgs)
        txtTitulo.Clear()
        txtPaginas.Clear()
        cboEditorial.SelectedIndex = -1
        cboCategoria.SelectedIndex = -1
        txtCosto.Clear()
        txtAño.Clear()
        txtTitulo.Focus()
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub
End Class
```

Consideraciones:

- Al iniciar la aplicación se deben mostrar los cuadros combinados de las editoriales y categorías llenas; para eso usamos una nueva sentencia llamada With, que permite resumir el código repetido en una determinada sentencia. El código debe implementarse en el evento load del formulario, tal como se muestra en el siguiente código:

```
Private Sub frmEditorial_Load(sender As Object, e As EventArgs)
        'Llenar las editoriales
        With cboEditorial.Items
            .Add("Montena")
            .Add("Planeta")
            .Add("ViaMagna")
            .Add("Macro")
        End With

        'Llenar las categorias
        With cboCategoria.Items
            .Add("Superación personal")
            .Add("Literatura")
            .Add("Ciencias")
            .Add("Informática")
        End With
End Sub
```

A continuación, la diferencia entre la sentencia completa y el uso de With:



- Desde el botón **Registrar** se puede obtener todos los datos ingresados del nuevo libro usando el siguiente código:

```
Dim titulo As String = txtTitulo.Text
Dim paginas As Integer = Integer.Parse(txtPaginas.Text)
Dim editorial As String = cboEditorial.Text
Dim categoria As String = cboCategoria.Text
Dim costo As Double = Double.Parse(txtCosto.Text)
Dim edición As Integer = Integer.Parse(txtAño.Text)
```

Luego, se envían los datos capturados al control ListView para la muestra al usuario, tal como se muestra en el siguiente código:

```
Dim fila As New ListViewItem(titulo)
fila.SubItems.Add(editorial)
fila.SubItems.Add(categoria)
fila.SubItems.Add(edicion.ToString())
fila.SubItems.Add(paginas.ToString())
fila.SubItems.Add(costo.ToString("0.00"))
lvLibros.Items.Add(fila)
```

- Ahora se realizan los cálculos de las estadísticas y se determina el total de libros por cada categoría colocando el siguiente código:

```
Dim i As Integer = 0
Dim cSuperacion, cLiteratura, cCiencias, cInformatica As Integer
Do
        Dim cat As String = lvLibros.Items.Item(i).SubItems(2).Text
        Select Case cat
                Case "Superación personal" : cSuperacion += 1
                Case "Literatura" : cLiteratura += 1
                Case "Ciencias" : cCiencias += 1
                Case "Informática" : cInformatica += 1
        End Select
        i += 1
Loop While i < lvLibros.Items.Count
```

La idea es recorrer por todos los elementos de la lista usando la sentencia repetitiva Do...Loop While. La condición de repetición se da mientras que la variable I (inicializada en cero, para obtener el primer valor de la lista) sea menor al número total de registros de la lista; esto es porque la primera fila inicia en cero. Dentro del ciclo de repeticiones se captura la categoría (Cat) de cada fila, de modo que se pueda evaluar y una vez encontrado aumentar sus contadores; que para esta ocasión no se necesita que sean variables globales, solo deben ser declaradas antes del ciclo de repeticiones. No olvidar que se trata de una sentencia de repetición condicionada, por lo tanto, la variable I siempre debe aumentar su valor en uno por cada ciclo de repeticiones.

- Ahora, se determinará el libro con mayor costo, colocando el siguiente código:

```
i = 0
Dim pos As Integer
Dim mayor As Double = 0
Do
        Dim cos As String = lvLibros.Items.Item(i).SubItems(5).Text
        If cos > mayor Then
                mayor = cos
                pos = i
        End If
        i += 1
Loop While i < lvLibros.Items.Count
Dim libroCostoMayor As String = lvLibros.Items.Item(pos).SubItems(0).Text
```

Para cualquier tipo de búsqueda de valores, ya sea numérico o textual dentro de una lista (ListView), se debe recorrer por cada fila con una estructura de repetición, siempre desde la posición cero (i=0) hasta la última fila de lista (lvLibros.Items. Count). Para obtener el mayor costo se debe usar la variable mayor, la cual tiene la misión de guardar el mayor costo encontrado en la lista; esta debe ser inicializada en cero para que el primer costo encontrado sea el mayor elemento y se compare con los demás. La variable Pos tiene la misión de obtener el valor de la posición del costo mayor encontrado, de tal forma que al mostrar el título del libro con mayor costo, sea la variable Pos la encargada de determinar en qué posición de fila se encuentra el libro con mayor costo.

- Ahora, determinar el libro con menor número de páginas, colocando el siguiente código:

```
i = 0
Dim menor As Integer = Integer.MaxValue
Do
        Dim pag As Integer = lvLibros.Items.Item(i).SubItems(4).Text
        If pag < menor Then
                menor = pag
                pos = i
        End If
        i += 1
Loop While i < lvLibros.Items.Count
Dim libroMenorPagina As String = lvLibros.Items.Item(pos).SubItems(0).Text
```

Se inicia por declarar la variable menor, la cual tendrá como misión obtener el menor valor asignado al número de páginas de un determinado libro; aquí se aplica la inversa, al inicializar la variable menor pues el valor cero es siempre menor a cualquier otro valor; por lo tanto, se debe inicializar con el valor más alto que pueda tener el tipo de datos entero, para eso usamos la sentencia Integer.MaxValue. Luego, se recorre todas las filas de la lista usando la sentencia Do…Loop While y, dentro del ciclo de repeticiones, recuperamos el número de páginas de cada fila en la variable Pag; esta se ubica en la columna cuatro por eso hacemos referencia a SubItems(4). Luego, verificar si dicho valor es menor al valor registrado en la variable Menor, de acuerdo a esto se actualiza el valor en la variable y lo más importante es capturar la posición de dicho valor; para eso se usa la variable Pos, que será la que nos brinde la posición del libro con el menor número de páginas, a través del siguiente código lvLibros.Items.Item(pos).SubItems(0).Text.

- Seguidamente, se determina el costo acumulado según el tipo de editorial, colocando el siguiente código:

```
i = 0
Dim cMontena, cPlaneta, cVia, cMacro As Double
Do
  Dim edi As String = lvLibros.Items.Item(i).SubItems(1).Text
  Select Case edi
   Case "Montena":cMontena+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
   Case "Planeta":cPlaneta+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
   Case "ViaMagna":cVia+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
    Case "Macro":cMacro+=Double.Parse(lvLibros.Items.Item(i).SubItems(5).Text)
  End Select
   i += 1
Loop While i < lvLibros.Items.Count
```

Como en los códigos anteriores, se recorre por todas las filas de la lista usando la sentencia Do…Loop While. Dentro del ciclo se captura el nombre de la editorial para que sea evaluada y, de acuerdo a esto, se acumulen los valores en las respectivas variables de acumulación según el tipo de editorial.

- Finalmente, realiza la impresión de los resultados en el control ListBox usando la sentencia With, la cual nos permite resumir código, como este caso a la sentencia lstE.Items, que se repetía en varias sentencias. Y para invocar a la limpieza de los controles se usa la sentencia **btnCancelar_Click(sender, e)**; el cual hace referencia al botón **Cancelar**, que limpia los controles y devuelve el foco al título del libro para un nuevo registro.

Caso desarrollado 5 For - Control de pensión de estudiantes

Una institución de educación superior desea tener el control de las pensiones que realizan sus estudiantes. Según la categoría y el promedio ponderado del alumno, se determinan el monto de la pensión, el descuento y el monto de la nueva pensión con descuento.

Tener en cuenta los siguientes aspectos:

- Implementar un formulario que contemple los datos solicitados en la aplicación.
- Usar la declaración de variables globales para la generación del código de registro del estudiante.
- El código de registro del estudiante se debe realizar de forma automática con el formato 0001.
- Se debe registrar el nombre, categoría y promedio ponderado del estudiante.
- Los montos de pensión se dan mediante la categoría según la siguiente tabla:

Categoría	Monto de pensión $
A	1500.00
B	1000.00
C	900.00
D	750.00

- Los montos de descuento se aplican basándose en el promedio ponderado según la siguiente tabla:

Promedio ponderado	Descuento %
Inferior a 13.99	0
Entre 14 y 15.99	10
Entre 16 y 17.99	12
Entre 18 y 20	15

- La nueva pensión se determina a partir de la diferencia entre el monto de pensión y el de descuento.
- El botón **Procesar** es el encargado de mostrar la información al control ListView como el código, nombre del estudiante, categoría, pensión, descuento y nueva pensión. Dicho botón debe estar asociado a la tecla <Enter>.
- El botón **Estadísticas** es el encargado de mostrar el monto acumulado según la nueva pensión de cada una de las categorías de los estudiantes. Asimismo, el total de estudiantes por categoría y, finalmente, el monto acumulado de las pensiones de todos los estudiantes.
- El botón **Limpiar** es el encargado de limpiar el contenido de los controles.
- Implementar la salida de la aplicación usando un cuadro de mensaje y hacer que este se encuentre asociado a la tecla <Esc>.

Solución:

Asignar las propiedades a los controles de la siguiente manera:

Control	Propiedad	Valor
Form1	(Name) AcceptButton CancelButton Text	frmPension btnProcesar btnSalir CONTROL DE PENSIÓN DE ESTUDIANTES
Label1	Text	CONTROL DE PENSIÓN DE ESTUDIANTES
Label2	Text	CÓDIGO
Label3	(Name)	lblCodigo
Label4	Text	ESTUDIANTE
Label5	Text	CATEGORÍA
Label6	Text	PROMEDIO PONDERADO
TextBox1	(Name)	txtEstudiante
TextBox2	(Name)	txtPromedio
ComboBox1	(Name)	cboCategoria
Button1	(Name) Text	btnProcesar PROCESAR
Button2	(Name) Text	btnLimpiar LIMPIAR
Button3	(Name) Text	btnEstadistica ESTADISTICAS
Button4	(Name) Text	btnSalir SALIR
ListBox1	(Name)	lstE
ListView1	(Name) GridLine Columns View	lvPension True Agregue 6 columnas y defina sus cabeceras como se muestra en el modelo. Details

El script de la solución es:

```
Public Class frmPension
    Dim n As Integer
    Private Sub frmPension_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Llenando las categorias
        With cboCategoria.Items
            .Add("A")
            .Add("B")
            .Add("C")
            .Add("D")
        End With

        'Generando el numero de registro de estudiante
        n += 1
        lblCodigo.Text = n.ToString("0000")
    End Sub

    Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
        'Capturando los datos
        Dim codigo As Integer = lblCodigo.Text
        Dim estudiante As String = txtEstudiante.Text
        Dim categoria As String = cboCategoria.Text
        Dim promedio As Double = Double.Parse(txtPromedio.Text)

        'Determinar la pensión del estudiante
        Dim pension As Double
        Select Case categoria
            Case "A" : pension = 1500
            Case "B" : pension = 1000
            Case "C" : pension = 900
            Case "D" : pension = 750
        End Select

        'Calculando el descuento
        Dim descuento As Double
        Select Case promedio
            Case 18 To 20 : descuento = 0.15 * pension
            Case 16 To 17.99 : descuento = 0.12 * pension
            Case 14 To 15.99 : descuento = 0.1 * pension
            Case Is < 13.99 : descuento = 0
        End Select

        'Calculando la nueva pension
        Dim nuevaPension As Double = pension - descuento

        'Imprimir los datos del estudiante y sus pagos
        Dim fila As New ListViewItem(codigo)
        fila.SubItems.Add(estudiante)
        fila.SubItems.Add(categoria)
        fila.SubItems.Add(pension.ToString("0.00"))
```

```
        fila.SubItems.Add(descuento.ToString("0.00"))
        fila.SubItems.Add(nuevaPension.ToString("0.00"))
        lvPension.Items.Add(fila)

        'Generando el numero de registro de estudiante
        n += 1
        lblCodigo.Text = n.ToString("0000")
    End Sub

    Private Sub btnEstadistica_Click(sender As Object, e As EventArgs)
        Dim i As Integer
        'Determinar el monto acumulado de pensiones por categoría de alumno
        Dim aA, aB, aC, aD As Double
        For i = 0 To lvPension.Items.Count - 1
           Dim cat As String = lvPension.Items.Item(i).SubItems(2).Text
            Select Case cat
              Case "A" : aA += lvPension.Items.Item(i).SubItems(5).Text
              Case "B" : aB += lvPension.Items.Item(i).SubItems(5).Text
              Case "C" : aC += lvPension.Items.Item(i).SubItems(5).Text
              Case "D" : aD += lvPension.Items.Item(i).SubItems(5).Text
            End Select
        Next

        'Total de alumnos por categoría
        Dim tA, tB, tC, tD As Integer
        For i = 0 To lvPension.Items.Count - 1
           Dim cat As String = lvPension.Items.Item(i).SubItems(2).Text
            Select Case cat
                Case "A" : tA += 1
                Case "B" : tB += 1
                Case "C" : tC += 1
                Case "D" : tD += 1
            End Select
        Next

        'Monto total acumulado de nuevas pensiones
        Dim aPension As Double
        For i = 0 To lvPension.Items.Count - 1
           Dim pen As Double = lvPension.Items.Item(i).SubItems(5).Text
            aPension += pen
        Next

        'Imprimir los resultados
        lstE.Items.Clear()
        lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. A: " & aA.ToString("C"))
        lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. B: " & aB.ToString("C"))
        lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. C: " & aC.ToString("C"))
        lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. D: " & aD.ToString("C"))
        lstE.Items.Add("-----------------------------------------")
    lstE.Items.Add("TOTAL DE ALUMNOS - CAT. A: " & tA.ToString("0.00"))
    lstE.Items.Add("TOTAL DE ALUMNOS - CAT. B: " & tB.ToString("0.00"))
    lstE.Items.Add("TOTAL DE ALUMNOS - CAT. C: " & tC.ToString("0.00"))
```

```
        lstE.Items.Add("TOTAL DE ALUMNOS - CAT. D: " & tD.ToString("0.00"))
        lstE.Items.Add("-----------------------------------------")
        lstE.Items.Add("MONTO ACUMULADO DE PENSIONES: " & aPension.ToString("C"))
    End Sub

    Private Sub btnSalir_Click(sender As Object, e As EventArgs) Handles
btnSalir.Click
        'Mostrando el mensaje y capturando la respuesta del usuario
        Dim r = MessageBox.Show("Esta seguro de salir?", "Salir",
                                MessageBoxButtons.YesNo, MessageBoxIcon.Error)

        'Evaluando la respuesta del usuario
        If (r = DialogResult.Yes) Then
            Me.Close()
        End If
    End Sub

    Private Sub btnLimpiar_Click(sender As Object, e As EventArgs)
        txtEstudiante.Clear()
        cboCategoria.SelectedIndex = -1
        txtPromedio.Clear()
        txtEstudiante.Focus()
        lstE.Items.Clear()
    End Sub
End Class
```

Consideraciones:

- Al iniciar la aplicación, mostrar las categorías de los alumnos, para esto se agrega el código en el evento Load del formulario; tal como se muestra a continuación:

```
Private Sub frmPension_Load(sender As Object, e As EventArgs) Handles
MyBase.Load
        'Llenando las categorias
        With cboCategoria.Items
            .Add("A")
            .Add("B")
            .Add("C")
            .Add("D")
        End With

        'Generando el numero de registro de estudiante
        n += 1
        lblCodigo.Text = n.ToString("0000")
    End Sub
```

En el código se usa la sentencia With, la cual permite resumir el código; luego, aumentar el contador N en un valor para generar el primer código autogenerado en la aplicación. Este será impreso en el control lblCodigo con el formato de cuatro ceros a la izquierda, usando la sentencia ToString("0000").

- Desde el botón **Procesar**, realizar todos los cálculos usando el siguiente código:

```
Private Sub btnProcesar_Click(sender As Object, e As EventArgs)
        Dim codigo As Integer = lblCodigo.Text
        Dim estudiante As String = txtEstudiante.Text
        Dim categoria As String = cboCategoria.Text
        Dim promedio As Double = Double.Parse(txtPromedio.Text)
```

Se empieza por capturar todos los elementos ingresados en el formulario; luego, asignar un monto de pensión según el tipo de categoría, tal como se muestra en el siguiente código:

```
Dim pension As Double
        Select Case categoria
            Case "A" : pension = 1500
            Case "B" : pension = 1000
            Case "C" : pension = 900
            Case "D" : pension = 750
End Select
```

Luego, calcular el descuento basado en el promedio ponderado del alumno; para esto usar la sentencia Select Case. Asimismo, una vez obtenido el descuento calcular la nueva pensión haciendo una diferencia entre la pensión normal y el descuento encontrado:

```
Dim descuento As Double
        Select Case promedio
            Case 18 To 20 : descuento = 0.15 * pension
            Case 16 To 17.99 : descuento = 0.12 * pension
            Case 14 To 15.99 : descuento = 0.1 * pension
            Case Is < 13.99 : descuento = 0
End Select

Dim nuevaPension As Double = pension - descuento
```

Finalmente, realizar la impresión de los valores usando la ListViewItem y aumentar en uno el valor N, para que al final del registro se autogenere el código del estudiante:

```
Dim fila As New ListViewItem(codigo)
fila.SubItems.Add(estudiante)
fila.SubItems.Add(categoria)
fila.SubItems.Add(pension.ToString("0.00"))
fila.SubItems.Add(descuento.ToString("0.00"))
fila.SubItems.Add(nuevaPension.ToString("0.00"))
lvPension.Items.Add(fila)

n += 1
lblCodigo.Text = n.ToString("0000")
```

- Desde el botón **Estadísticas** realizar la impresión de los conteos y acumulaciones, usando el siguiente código:

```
Private Sub btnEstadistica_Click(sender As Object, e As EventArgs)
        Dim i As Integer

    Dim aA, aB, aC, aD As Double
    For i = 0 To lvPension.Items.Count - 1
        Dim cat As String = lvPension.Items.Item(i).SubItems(2).Text
        Select Case cat
            Case "A" : aA += lvPension.Items.Item(i).SubItems(5).Text
            Case "B" : aB += lvPension.Items.Item(i).SubItems(5).Text
            Case "C" : aC += lvPension.Items.Item(i).SubItems(5).Text
            Case "D" : aD += lvPension.Items.Item(i).SubItems(5).Text
        End Select
    Next
```

Para realizar las acumulaciones de valor que se encuentran en la sexta columna (Nueva Pensión) del control ListView, usando la sentencia repetitiva For, iniciando desde la posición cero hasta la última posición del elemento agregado, la sentencia lvPension.Items.Count devuelve el total de elementos en la lista pero la posición del último elemento se encuentra en una posición anterior, ya que su primer registro se encuentra en la posición cero; es así que en la sentencia For disminuimos en uno la sentencia **lvPension.Items.Count-1**. No olvidar que las variables de acumulación deben encontrarse fuera de la sentencia y que por cada ciclo de repetición se deben capturar el valor de la categoría Cat para poder evaluarlo y así determinar que por

cada tipo existe una acumulación. Finalmente, considerar que la sentencia **lvPension.Items.Item(i).SubItems(5).Text** hace referencia a la posición 5 del control ListView, el cual corresponde a las nuevas pensiones el cual queremos acumular.

Ahora, realizar el conteo de alumnos por categoría con el siguiente código:

```
Dim tA,tB, tC, tD As Integer
        For i = 0 To lvPension.Items.Count - 1
            Dim cat As String = lvPension.Items.Item(i).SubItems(2).Text
            Select Case cat
                Case "A" : tA += 1
                Case "B" : tB += 1
                Case "C" : tC += 1
                Case "D" : tD += 1
        End Select
Next
```

Iniciar declarando las variables de conteo (tA, tB, tC, tD) fuera del ciclo de repeticiones, luego realizar el recorrido por toda la lista usando la sentencia For desde la posición cero hasta la posición del último elemento agregado a la lista (lvPension.Items.Count-1); seguidamente, capturar la categoría por cada ciclo para poder compararlo y así asignar el contador a las variables correspondientes.

Ahora, calcular el monto acumulado de pensiones de la misma forma que los procesos anteriores; se debe recorrer la lista y obtener el valor de la nueva pensión `(lvPension.Items.Item(i).SubItems(5).Text)` para poder acumularlo; la variable aPension tiene la misión de acumular dichos valores.

```
Dim aPension As Double
        For i = 0 To lvPension.Items.Count - 1
            Dim pen As Double = lvPension.Items.Item(i).SubItems(5).Text
            aPension += pen
        Next
```

Finalmente, realizar la impresión de todos los valores de conteo y acumulación. No olvidar que siempre se debe iniciar con la limpieza del control y luego la impresión de valores, tal como se muestra en el siguiente código:

```
lstE.Items.Clear()
lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. A: " & aA.ToString("C"))
lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. B: " & aB.ToString("C"))
lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. C: " & aC.ToString("C"))
lstE.Items.Add("MONTO ACUMULADO DE PENSION-CAT. D: " & aD.ToString("C"))
lstE.Items.Add("-----------------------------------------------")
lstE.Items.Add("TOTAL DE ALUMNOS - CAT. A: " & tA.ToString("0.00"))
lstE.Items.Add("TOTAL DE ALUMNOS - CAT. B: " & tB.ToString("0.00"))
lstE.Items.Add("TOTAL DE ALUMNOS - CAT. C: " & tC.ToString("0.00"))
lstE.Items.Add("TOTAL DE ALUMNOS - CAT. D: " & tD.ToString("0.00"))
lstE.Items.Add("-----------------------------------------------")
lstE.Items.Add("MONTO ACUMULADO DE PENSIONES: " & aPension.ToString("C"))
```

Capítulo 6

Programación modular

Capacidad terminal:

Desarrollar aplicaciones con la metodología modular en donde su programa se dividirá en bloques haciendo una aplicación robusta y de calidad.

Contenido

- Introducción a la programación modular
- Subprocedimientos
- Procedimientos de función
- Procedimiento de propiedad
- Miembros de la biblioteca en tiempo de ejecución
- Microsoft Visual Basic Information
- Funciones de conversión
- Función lambda
- Casos desarrollados

6.1 Introducción a la programación modular

Conforme se avanza en los códigos se nota que estos crecen dependiendo de la dificultad encontrada; es así que llegará un momento en que será tedioso buscar un proceso dentro de un bloque de código. La programación conocida hasta el momento se llama lineal, en la cual el código sigue una secuencia de instrucciones, uno a continuación del otro, pero, ¿qué hacer si es necesario un código previamente colocado? Lo usual es copiar dicho código, generando así más líneas de código que en algún momento resultarán incontrolables.

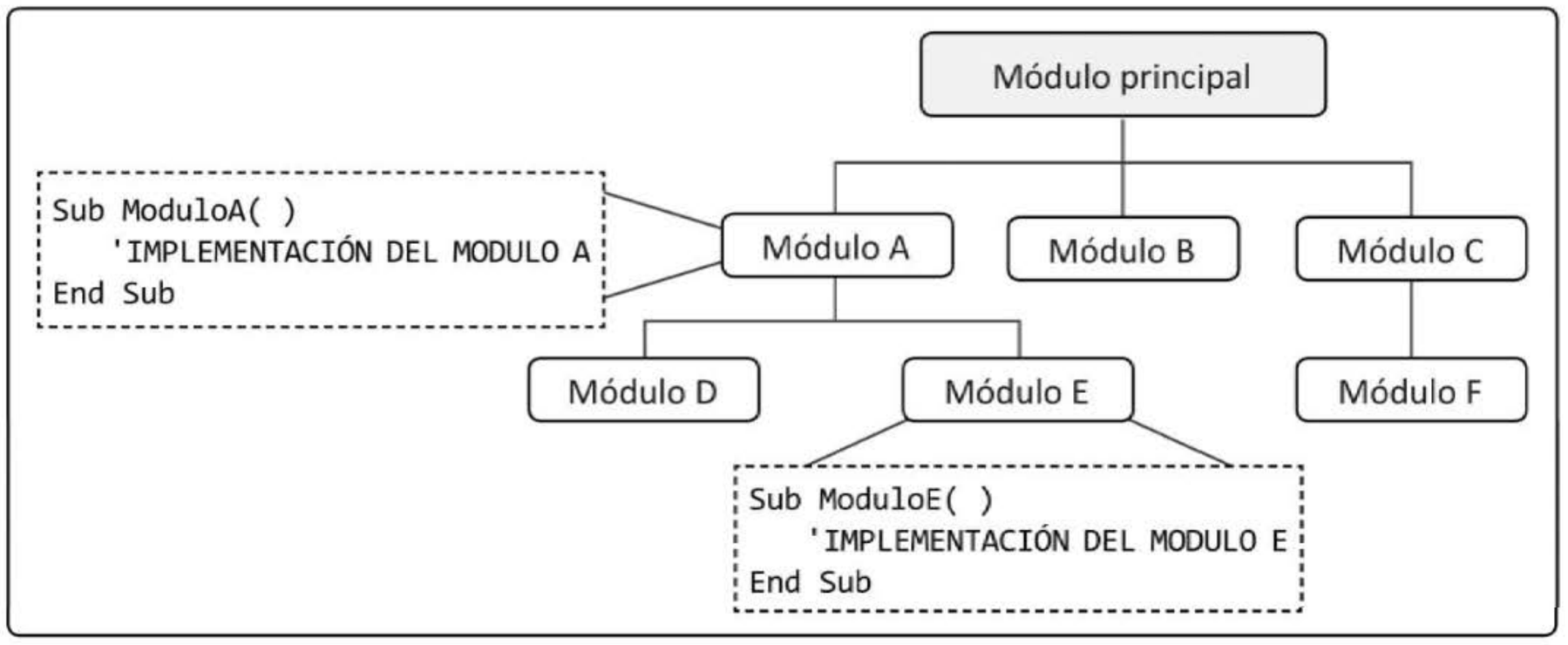
Además, la idea en la programación actual es usar menos código y esto se logra reutilizándolo en toda una aplicación; esto permitirá resolver un proceso complejo, ya que se dividirá en partes controlables, usando la clásica técnica de «divide y vencerás». La subdivisión del código en bloques genera un mejor control de él y además permite corregir errores rápidamente, ya que no sería necesario revisar todo el código sino solo el modulo donde se ocasionó el error. Asimismo, cada módulo representa una actividad en una aplicación que, integrada, cumple con el objetivo trazado en el proceso de desarrollo de la aplicación.

El tema de la integración hace pensar que la programación modular no es solo cuestión de dividir el código; para esto sirve la habilidad de detectar qué bloques de código pueden servir como módulo, ya que, al final, los módulos solo dividen al código que antes se encontraba unido pero que, gracias a ellos, se volverán a unir mediante la llamada a dichos módulos.

Toda aplicación desarrollada en lenguajes de programación modernos se encuentra dividida en módulos y, así, es necesario acostumbrarse a programar, ya que facilitará el desarrollo de la misma. En este capítulo se estudiará cómo implementar métodos en Visual Basic, así como su aplicación en casos reales.

6.1.1 Composición de un módulo

Como se había visto, un módulo es un bloque de código que tiene un objetivo específico en el desarrollo de una aplicación, por lo tanto, tendría una composición, como se muestra en la siguiente imagen:



Si se considera que un proceso de desarrollo se dividirá en bloques de código generando módulos, se los debe tener organizados, de forma que será necesario un módulo principal que controle las llamadas a los demás módulos y genere algo parecido a una organización, generando así un medio abstracto de tener controlada una aplicación; desde aquí se pueden mencionar dos términos que se usará en todo el desarrollo de módulos: el primero es la implementación, la cual hace referencia al código que contiene un módulo y que fue desarrollada para un objetivo específico en la aplicación; y la segunda es la llamada del modulo, la cual mediante su invocación permitirá ejecutar el código contenido en el modulo.

A continuación, una comparación entre un código completo y la composición de bloques de código, ambos con el mismo objetivo:

```
Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)

        'CAPTURANDO DATOS
        Dim cliente As String = txtCliente.Text
        Dim dias As Integer = Integer.Parse(txtDias.Text)

        'CALCULANDO MONTOS
        Dim monto As Double = costo * dias

        'IMPRIMIR
        Dim fila As New ListViewItem(cliente)
        fila.SubItems.Add(dias.ToString())
        fila.SubItems.Add(costo.ToString("C"))
        fila.SubItems.Add(monto.ToString("C"))
        lvR.Items.Add(fila)
        lblTotal.Text = (Double.Parse(lblTotal.Text) + monto).ToString("0.00")

End Sub
```

En el código anterior se observa que el botón **Registrar** realiza tres actividades específicas. La primera es obtener los valores desde la caja de texto del cliente y el número de días. La segunda es calcular el monto según el costo y el número de días. Por último, la tercera es realizar la impresión de los resultados en el control ListView.

A continuación, el mismo código dividido en módulos.

```
Private Sub btnRegistrar_Click(sender As Object, e As EventArgs)
    'CAPTURANDO DATOS
    capturaDatos ──────►  Dim cliente As String = txtCliente.Text
                          Dim dias As Integer = Integer.Parse(txtDias.Text)

    'CALCULANDO MONTOS
    daterminarCalculos ─►  Dim monto As Double = costo * dias

    'IMPRIMIR
    imprimirResultados ─►  Dim fila As New ListViewItem(cliente)
                           fila.SubItems.Add(dias.ToString())
                           fila.SubItems.Add(costo.ToString("C"))
                           fila.SubItems.Add(monto.ToString("C"))
                           lvR.Items.Add(fila)
                           lblTotal.Text = (Double.Parse(lblTotal.Text) +
                                                        monto).ToString("0.00")
End Sub
```

Para este caso se divide en tres módulos. El primero, capturaDatos, permite obtener los valores como el nombre del cliente y la cantidad de días. El segundo módulo, determinarCalculos, se encarga de realizar el cálculo del monto. Y el tercer módulo, imprimirResultados, permite mostrar los resultados en el control ListView.

6.1.2 Descomposición modular

Se le llama así a la determinación que toma el programador frente al desarrollo de aplicaciones usando programación modular. Es determinante que para descomponer un código se analice la complejidad de la aplicación, asimismo determinar si dicho modulo será reutilizable, ya que la descomposición debe ayudar a mantener una aplicación compleja en una aplicación controlada y eficiente en el tiempo.

En el análisis de un proceso a desarrollar se debe tener en cuenta que la cantidad de módulos deben ser los necesarios para una aplicación controlada sin ocasionar problemas de efectividad al momento de implementarlos. Por eso, la sugerencia sería analizar el proceso y encontrar qué actividades componen a una aplicación y si estos se convertirán en módulos.

Algunas consideraciones necesarias al implementar módulos:

- La cantidad de líneas de código que se implementara en un módulo.
- El nivel de anidamiento debe tener un máximo de 7 niveles.
- Cada módulo tiene un objetivo específico en la aplicación, es decir, siempre tiene una labor que servirá a quien lo invoque.

A continuación, la aplicación de la descomposición en módulos a partir de un caso:

Realizar una aplicación que permita determinar el monto, descuento y total a pagar que debe realizar un cliente por la compra de helados de un mismo tipo.

Tener en cuenta los siguientes aspectos:

- Debe ingresar el nombre del cliente y la cantidad de helados a comprar.
- El precio del producto se da según la siguiente tabla:

Tipo de helado	Precio
Friorico	$3.50
Tartufo	$5.00
Sandwich	$2.50

- El monto se calcula del producto entre el precio y la cantidad comprada de helados.
- El descuento se obtiene aplicando el 7.5 % al monto.
- El total a pagar se determina de la diferencia entre el monto y el descuento.

Para realizar la descomposición se debe tener en cuenta el orden de las actividades que necesita la aplicación:

- Capturar los datos del formulario como el nombre del cliente, tipo de helado y la cantidad seleccionada por el usuario.
- Determinar el precio del helado según lo seleccionado por el usuario.
- Calcular el subtotal de la compra el cual resulta del producto entre el precio y la cantidad comprada.
- Calcular el descuento según el subtotal.
- Calcular en monto neto a pagar por el cliente, el cual resulta de la diferencia entre el subtotal y el descuento.
- Imprimir los resultados.

La GUI propuesta podría ser de la siguiente manera:

A continuación, el código implementado en el botón **Registrar**:

```
Public Class frmVenta

    Private Sub btnRegistrar_Click() Handles btnRegistrar.Click
        'Capturando los valores del formulario
        Dim cliente As String = txtCliente.Text
        Dim helado As String = cboHelado.Text
        Dim cantidad As Integer = Integer.Parse(txtCantidad.Text)

        'Realizando cálculos
        Dim precio As Double = asignaPrecio(helado)
        Dim subtotal As Double = calculaSubtotal(cantidad, precio)
        Dim descuento As Double = calculaDescuento(subtotal)
        Dim neto As Double = calculaNeto(subtotal, descuento)

        'Imprimir
        imprimir(subtotal, descuento, neto)
    End Sub

    'Implementación de Modulos
    Function asignaPrecio(ByVal helado) As Double
        Select Case helado
            Case "Friorico" : Return 3.5
            Case "Tartufo" : Return 5.0
            Case "Sandwich" : Return 2.5
            Case Else : Return 0
        End Select
    End Function

    Function calculaSubtotal(ByVal cantidad As Integer,
                             ByVal precio As Double) As Double
        Return cantidad * precio
    End Function

    Function calculaDescuento(ByVal subtotal As Double) As Double
        Return subtotal * 0.15
    End Function

    Function calculaNeto(ByVal subtotal As Double,
                         ByVal descuento As Double) As Double
        Return subtotal - descuento
    End Function

    Sub imprimir(subtotal As Double, descuento As Double, neto As Double)
        Dim fila As New ListViewItem(txtCliente.Text)
        fila.SubItems.Add(cboHelado.Text)
        fila.SubItems.Add(txtCantidad.Text)
        fila.SubItems.Add(subtotal.ToString("C"))
        fila.SubItems.Add(descuento.ToString("C"))
        fila.SubItems.Add(neto.ToString("C"))
        lvVenta.Items.Add(fila)
    End Sub
End Class
```

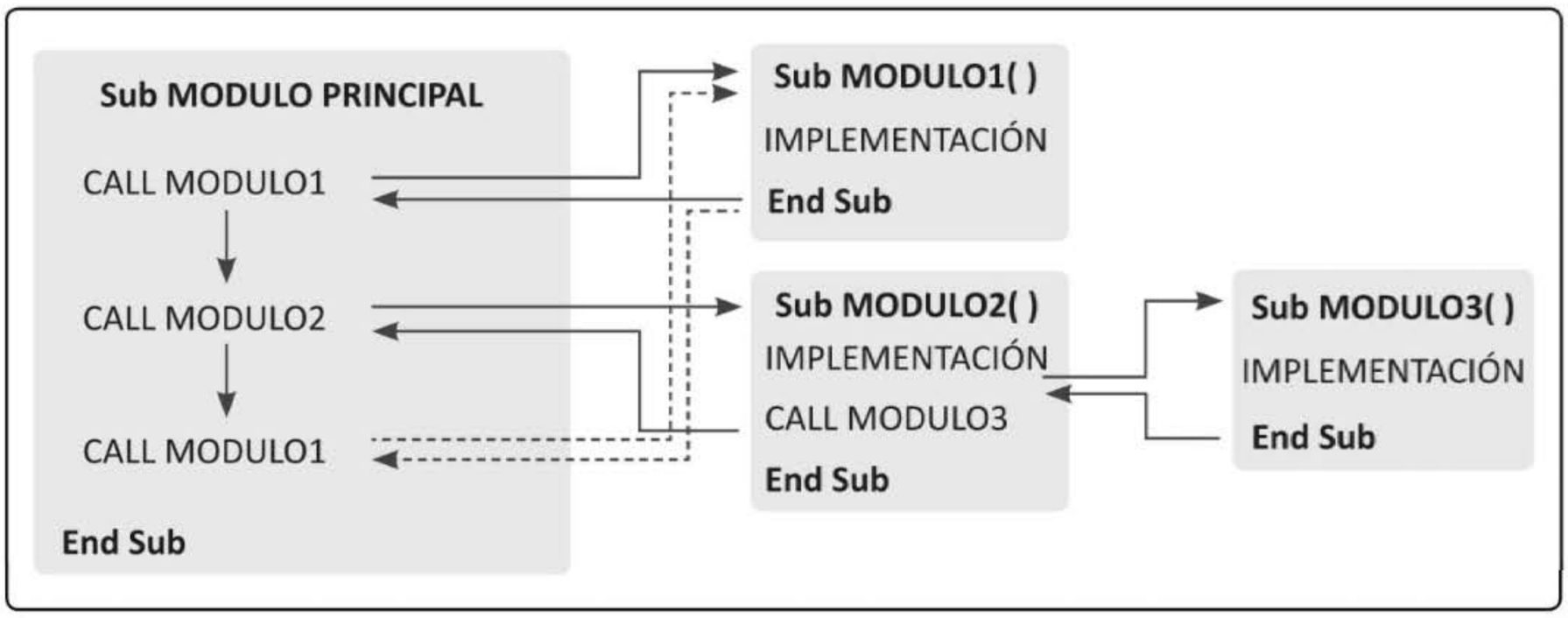
Para desarrollar el caso propuesto, implementar los módulos:

Nombre del módulo	Descripción
asignaPrecio	Módulo que permite asignar el precio a un determinado tipo de helado, para esto necesita como parámetro el tipo de helado, y es a partir de allí que se podrá asignar un precio mediante la sentencia Select Case.
calculaSubtotal	Módulo que permite calcular el monto subtotal, para lo cual necesita como parámetros el precio del producto y la cantidad comprada por el cliente.
calculaDescuento	Módulo que permite calcular el monto de descuento, para lo cual solo necesita el monto subtotal.
calculaNeto	Módulo que permite calcular el monto neto que pagará el cliente, para lo cual necesita como parámetro el monto subtotal y el descuento.
Imprimir	Módulo que permite imprimir los valores resultantes de la aplicación, se necesita como parámetro los montos de subtotal, descuento y neto, ya que los demás valores a imprimir se encuentran en los controles del formulario.

6.1.3 Llamada a un módulo

Cuando se coloca un código dentro de un módulo se le llama implementación, el cual tiene un objetivo específico en la aplicación; asimismo, se debe tener en cuenta que algunos módulos necesitarán de parámetros de entrada para poder usarlo dentro de su código. Si todo es correcto hasta aquí, usar el módulo, el cual se le debe llamar llamadas al módulo, el cual lo invoca para que genere una actividad dentro de una aplicación.

Se debe tener en cuenta que, al invocar a un módulo, se le está cediendo el control del flujo de la aplicación y que cuando termine devuelve el control a quien lo invocó.



Desde el módulo principal se inician las llamadas a los módulos; al invocar al Modulo1 este buscará su implementación y ejecutará los códigos encontrados en dicho módulo, una vez finalizado devuelve el control al módulo principal para continuar con la llamada al Modulo2; de la misma forma se busca el código implementado en el Modulo2 para ejecutarlo, pero allí mismo se hace una llamada al Modulo3. Entonces, una vez finalizada la ejecución del código del Modulo3 devuelve el control al Modulo2 para, finalmente, devolver el control al módulo principal.

Finalmente, se debe tener en cuenta que en el Modulo principal se invoca nuevamente al Modulo1, ya que las llamadas a los módulos no tiene límite, es decir, se puede llamar a los módulos las veces que sean necesarias y en cualquier lugar de la aplicación.

6.1.4 Tipos de módulos

Existen varios tipos de módulos, los cuales son implementados de diferentes formas por los lenguajes de programación. Para este caso:

- Subprocedimiento

```
Sub calculaPromedio()
    Dim promedio As Double = (n1 + n2 + n3 + n4) / 4
End Sub
```

- Procedimiento de función

```
Function calculaPromedio()
    Return (n1 + n2 + n3 + n4) / 4
End Function
```

- Procedimiento de propiedad

```
ReadOnly Property promedio As Double
    Get
        Return (n1 + n2 + n3 + n4) / 4
    End Get
End Property
```

6.2 Subprocedimientos

Son módulos que ejecutan una determinada acción en la aplicación, pero no devuelven valor de respuesta a quien lo invoca; también son conocidos como módulos sin valor de retorno. Se debe considerar que existen dos formar de implementar subprocedimientos:

Tipo	Descripción
Subprocedimientos sin parámetros	Se le llama así cuando el procedimiento no necesita de valores para su funcionamiento, en cambio usará valores globales que le permitirá modificar o no a dichos valores; se puede nombrar como ejemplo al subprocedimiento de limpiar controles o mostrar mensajes.
Subprocedimientos con parámetros	Se le llama así cuando el procedimiento necesita de valores de entrada, los cuales serán usados en la implementación de su código; se puede nombrar como ejemplo al subprocedimiento de imprimir, que tendría como parámetros los valores que necesita mostrar al usuario.

6.2.1 Implementación de un subprocedimiento sin parametros

```
Sub NombreProcedimiento()
  'implementación del subprocedimiento
End Sub
```

Sub representa la cláusula de inicio del subprocedimiento, el nombre del procedimiento tiene la misma regla que se hace al declarar una variable; en la implementación del subprocedimiento se coloca el código que tendrá alguna actividad en la aplicación y, finalmente, End Sub indica el fin del subprocedimiento. El siguiente subprocedimiento permite limpiar los controles de un formulario:

```
Sub LimpiaControles()
  txtCliente.Text = ""
  cboTipo.SelectedIndex = -1
  txtCliente.Focus()
End Sub
```

La caja de texto txtCliente permite ingresar el nombre del cliente y para limpiarlo se hace referencia a la propiedad Text asignando un juego de comillas vacías para su limpieza. El cuadro combinado usa la propiedad SelectedIndex=-1 para posicionar en un elemento vacío del cuadro y, finalmente, se activa el foco a la caja de texto para poder ubicar el cursor del mouse en dicha caja de texto.

Cuando se realiza la llamada al subprocedimiento LimpiaControles se realiza de la siguiente manera:

Call LimpiaControles(), o directamente, colocar LimpiaControles().

6.2.2 Implementación de un subprocedimiento con parámetros

```
Sub NombreProcedimiento(Byval parámetro As TipoDatos)
  'implementación del procedimiento
End Sub
```

La cláusula Sub indica el inicio del subprocedimiento, seguidamente se asigna un nombre el cual tiene las mismas reglas que tienen los nombres de variables, luego se definen los parámetros como si se declarara una variable, a excepción de que no se coloca la cláusula Dim sino, Byval, el cual representa a un valor. El siguiente subprocedimiento permite calcular el monto a partir de un precio y una cantidad ingresada como parámetro:

```
Sub calculaMonto(ByVal precio As Double, ByVal cantidad As Integer)
  Dim monto As Double
  monto = precio * cantidad
End Sub
```

Los parámetros Precio y Cantidad deben especificarse con su propio tipo de datos para cada uno además de colocar la clausula Byval para indicar que el parámetro es un valor y no una referencia. Luego, en la implementación del subprocedimiento, se declara la variable monto que tendrá la misión de almacenar dicho valor desde el producto entre precio y cantidad.

Cuando es necesario llamar al subprocedimiento calculaMonto se usa el siguiente código:

Dim precio As Double = 100

Dim cantidad As Integer = 10

CallcalculaMonto(precio,cantidad), o directamente colocar calculaMonto(precio,cantidad).

6.3 Procedimiento de función

Son módulos que ejecutan una determinada acción en la aplicación y que devuelve un solo resultado a quien lo invocó; a partir de aquí se debe tener en cuenta que también se debe definir el tipo de datos del valor de salida; es aquí donde se diferencia del subprocecimiento que no devuelve valor.

También, considerar que existen dos formas de implementar funciones:

Tipo	Descripción
Función sin parámetros	Se le llama así cuando la función no necesita de valores de entrada para su funcionamiento, sino que usa valores globales que le permite modificar o no a dichos valores, para, finalmente, devolver siempre un solo valor. Por ejemplo, se puede implementar una función que devuelva la fecha actual.
Función con parámetros	Se le llama así cuando la función necesita de valores de entrada, los cuales son usados en la implementación de su código; se puede nombrar como ejemplo una función que calcula el monto total a pagar por una venta en la cual necesite como parámetros el precio y la cantidad de productos comprados.

6.3.1 Implementación de una función sin parámetros

```
Function Nombrefuncion() As TipoDatos
  'Implementación de la función
  Return valor_retorno
End Function
```

Function representa la cláusula de inicio de la función, cuyo nombre tiene la misma regla que se hace al declarar una variable; en la implementación de la función se coloca el código que tendrá alguna actividad en la aplicación y al final de la implementación siempre se coloca la cláusula Return para especificar cuál es el valor resultante de la función. End function indica el fin de la implementación de la función. La siguiente función permite mostrar la fecha actual del sistema:

```
Function muestraFecha() As DateTime
  Dim fechaActual As DateTime
  fechaActual = DateTime.Now
  Return fechaActual
End Function
```

La función muestraFecha tiene como tipo de datos de salida a un DateTime porque el resultado esperado es la fecha actual. En la implementación declaramos la variable fechaActual del mismo tipo de datos de la salida, ya que recibe la fecha actual del sistema, esto se muestra gracias a la función DateTime.Now que, finalmente, es devuelto mediante la cláusula Return.

Cuando se necesite realizar la llamada a la función muestraFecha, se podrían usar los siguientes códigos:

- Asignar el valor de la función a una variable, Dim Fecha As DateTime = muestraFechaActual().
- Asignar el valor de la función a un mensaje, MessageBox.Show(muestraFechaActual()).
- Asignar el valor de la función a un control Label, lblFecha.Text = muestraFechaActual().

6.3.2 Implementación de una función con parámetros

```
Function NombreFuncion(Byval parámetro As TipoDatos) As TipoDatos
  'Implementación de la función
End Sub
```

La cláusula Function indica el inicio de la función, seguidamente se asigna un nombre, el cual tiene las mismas reglas que tienen los nombres de variables, luego, se definen los parámetros como si se declarara una variable a excepción de que no se coloca la cláusula Dim sino, Byval, el cual representa a un valor; no hay límite en la cantidad de parámetros. Finalmente, se debe especificar el tipo de datos del valor de salida. La siguiente función permite calcular el monto a partir de un precio y una cantidad ingresada como parámetro:

```
Function calculaMonto(ByVal precio As Double, ByVal cantidad As Integer) As Double
   Dim monto As Double
   monto = precio * cantidad
   Return monto
End Sub
```

Los parámetros Precio y Cantidad deben especificarse con su propio tipo de datos para cada uno, además de colocar la cláusula Byval para indicar que el parámetro es un valor y no una referencia. Luego, en la implementación de la función se declara la variable Monto que tendrá la misión de almacenar dicho valor desde el precio y cantidad de productos adquiridos. Finalmente, se especifica la cláusula Return para devolver el valor monto como resultado de la función.

Cuando se realiza la llamada a la función calculaMonto se efectúa de la siguiente manera:

- Dim precio As Double = 100
- Dim cantidad As Integer = 10
- Dim monto As Double = calculaMonto(precio,cantidad)

6.4 Procedimiento de propiedad

Son módulos que ejecutan una determinada acción en la aplicación y que devuelve un solo resultado a quien lo invocó usando el método GET. Su implementación se adecúa al uso de clases y objetos en una aplicación.

Implementación de una propiedad

```
Property ReadOnly NombrePropiedad(Parámetros) As TipoDatos
  Get
    Return valor_retorno
  End Get
End Property
```

Property representa la cláusula de inicio de la propiedad, ReadOnly asegura que los valores que ingresan no sean modificados; el nombre de la propiedad tiene la misma regla que se hace al declarar una variable, dentro de Ge y End Get se debe colocar la implementación que tendrá alguna actividad en la aplicación y que para devolver un resultado debe usar la cláusula Return. La siguiente propiedad permite mostrar el cálculo del monto a pagar por la compra de un producto a un determinado precio y una cantidad:

```
Public ReadOnly Property calculaMonto(ByVal precio As Double,
                                      ByVal cantidad As Integer) As Double
  Get
    Return precio * cantidad
  End Get
End Property
```

La propiedad calculaMonto debe tener especificación ReadOnly ya que no recibe valores Set.. End Set, solo devuelve un resultado mediante Get..End Get. Los parámetros son especificados como se realiza en las funciones o procedimientos; asimismo, se especifica el valor de retorno con la clausula Return.

Cuando se necesite realizar la llamada a la propiedad calculaMonto, se podría usar el siguiente código:

```
Dim precio As Double = 20
Dim cantidad As Integer = 10
Dim monto As Double = calculaMonto(precio, cantidad)
MessageBox.Show(monto.ToString())
```

6.5 Miembros de la biblioteca en tiempo de ejecución

El espacio de nombres Microsoft.VisualBasic contiene las clases, los módulos, las constantes y las enumeraciones que conforman la biblioteca en tiempo de ejecución de Visual Basic. Estos miembros de la biblioteca proporcionan los procedimientos, las propiedades y los valores de las constantes que puede utilizar en el código. Cada módulo y clase representa una categoría o funcionalidad particular.

6.5.1 Funciones de cadena

1. **Asc**: Devuelve un valor numérico que corresponde al código ASCII de un determinado carácter.

Determinar el código ASCII de un carácter enviado como parámetro:

```
Dim codigo As Integer
codigo = Asc("A")
```

El valor contenido en la variable código es 65. En el siguiente código se visualiza todo los código ASCII de cada uno de los caracteres contenidos en una cadena:

```
Dim texto As String = "ABC"
For i = 0 To texto.Length - 1
  Dim codigo As Integer = Asc(texto(i))
  MessageBox.Show(codigo)
Next
```

Los valores que toma la variable Código son 65, 55 y 67. Se debe tener en cuenta que una cadena en realidad es un arreglo de caracteres, los cuales se puede recorrer usando una estructura de repetición y para acceder a cada carácter se usa el formato del arreglo texto(0), texto(1) y texto(2), ya que el primer carácter ocupa la primera posición, es decir, la posición cero. Es por esa razón que la estructura de repetición inicia en cero y termina en el tamaño total de la cadena menos uno.

2. **Chr**: Devuelve un carácter asociado a un valor numérico correspondiente a un código ASCII.

Mostrar el carácter asociado a un numero enviado como parámetro:

```
Dim caracter As String
caracter = Chr(65)
```

El carácter contenido en la variable es la letra A en mayúsculas. En el siguiente código se mostrarán las letras del abecedario en mayúsculas:

```
For i = 65 To 90
  Dim caracter As String
  caracter = Chr(i)
  MessageBox.Show(caracter)
Next
```

Los valores que toma la variable caracter son desde la letra "A" hasta la "Z" ya que los codigos ASCII inicia del 65 para la letra A y 90 para la letra Z.

3. **Format**: Asigna un determinado formato a los valores resultantes en unas aplicaciones. Contamos con las siguientes variaciones:
   - Formato moneda (C, c): Permite mostrar un valor numérico en formato de moneda.

Mostrar un valor decimal en el formato de moneda.

```
Dim monto As Double
monto = 2590.48
Dim fMoneda As String = Format(monto, "C")
```

El valor contenido en la variable fMoneda es $2,590.48; el formato de moneda asigna un separador de miles y dos decimales expresados en una cadena de caracteres; se especifica esto porque dicho valor no puede ser considerado como un valor numérico de aquí en adelante.

   - Formato estándar (N, n): Permite mostrar un valor numérico con separador de miles.

Mostrar un valor decimal en el formato de moneda.

```
Dim valor As Double
valor = 2590.48
Dim fEstandar As String = Format(valor, "N")
```

El valor contenido en la variable fEstandar es 2,590.48; el formato asigna un separador de miles que se puede usar para representar grandes cantidades, ya que no es considerado como valor monetario.

- Formato Percent: Permite mostrar un valor numérico multiplicado por 100 y con un operador de porcentaje.

Mostrar la asignación de 20.00 % en un valor de tipo texto:

```
Dim valor As Double
valor = 0.2
Dim fPercent As String = Format(valor, "Percent")
```

El valor contenido en la variable fPercent es 20.00%; el formato asigna un separador de decimales, dos decimales en la expresión, además del símbolo de porcentaje junto al ultimo carácter.

- Formato P (P, p): Permite mostrar un valor numérico multiplicado por 100 y con un operador separado de porcentaje.

Mostrar la asignación de 120.00 % en un valor de tipo texto:

```
Dim valor As Double
valor = 1.2
Dim fP As String = Format(valor, "P")
```

El valor contenido en la variable fP es 120.00 %; el formato asigna un separador de decimales, dos decimales en la expresión además del símbolo de porcentaje separado al ultimo carácter.

- Formato científico: Permite mostrar un valor numérico en notación científica.

Mostrar un valor numérico expresado en un valor científico:

```
Dim valor As Double
valor = 1234567
Dim fCientifico As String = Format(valor, "Scientific")
```

El valor contenido en la variable fCientifico es 1.23E + 06.

- Formato E (E, e): Permite mostrar un valor numérico en notación científica estándar con seis dígitos.

Mostrar un valor numérico expresado en un valor científico:

```
Dim valor As Double
valor = 1234567
Dim fE As String = Format(valor, "E")
```

El valor contenido en la variable fE es 1.234567E + 006.

- Formato de fecha general (G): Muestra la fecha y la hora correspondiente a un valor de tipo fecha.

Mostrar la fecha actual con el formato de fecha general:

```
Dim fechaActual As DateTime
fechaActual = DateTime.Now.ToString()
MessageBox.Show("La fecha actual es: " & Format(fechaActual, "G"))
```

El valor contenido en la variable fechaActual es Sábado, 23 de setiembre de 2017 09:00:00.

- Formato de fecha mediana (D): Muestra solo la fecha en formato mediano correspondiente a un valor de tipo fecha.

Mostrar la fecha actual con el formato mediano:

```
Dim fechaActual As DateTime
fechaActual = DateTime.Now.ToString()
MessageBox.Show("La fecha actual es: " & Format(fechaActual, "G"))
```

El valor contenido en la variable fechaActual es Sábado, 23 de setiembre de 2017 .

- Formato de fecha corta (d): Muestra solo la fecha en formato corto correspondiente a un valor de tipo fecha.

Mostrar la fecha actual con el formato corto:

```
Dim fechaActual As DateTime
fechaActual = DateTime.Now.ToString()
MessageBox.Show("La fecha actual es: " & Format(fechaActual, "d"))
```

El valor contenido en la variable fechaActual es 23/09/2017.

- Formato de hora larga (T): Muestra solo la hora en formato largo, el cual corresponde a un valor de tipo fecha.

Mostrar la hora actual con el formato largo:

```
Dim horaActual As DateTime
horaActual = DateTime.Now.ToString()
MessageBox.Show("La hora actual es: " & Format(horaActual, "T"))
```

El valor contenido en la variable horaActual es 09:59:20.

- Formato de hora corta (t): Muestra solo la hora en formato corto el cual corresponde a un valor de tipo fecha.

Mostrar la hora actual con el formato corto:

```
Dim horaActual As DateTime
horaActual = DateTime.Now.ToString()
MessageBox.Show("La hora actual es: " & Format(horaActual, "t"))
```

El valor contenido en la variable horaActual es 09:59.

- Formato definido por el usuario para valores de tipo numérico:

Mostrar el valor numérico sin ningún formato:

```
Dim valor As Double
valor = 500
Dim formato As String = Format(valor, "")
```

El valor contenido en la variable formato es 500. En cambio, si se especifica un solo cero, se estaría especificando un valor redondeado por exceso, por ejemplo:

```
Dim valor As Double
valor = 0.6
Dim formato As String = Format(valor, "0")
```

El valor contenido en la variable formato es 1. En cambio, si se especifica un solo cero mas un punto y dos decimales, se especificaría un valor tipo moneda sin símbolo, por ejemplo:

```
Dim valor As Double
valor = 500
Dim formato As String = Format(valor, "0.00")
```

El valor contenido en la variable formato es 500.00 y si fuera un decimal como 0.8 el resultado sería 0.80. Si se especifica un formato de numeral (# el cual representa a cualquier valor numérico) y una coma (,) como especificador de miles; se verá un número con separador de miles pero sin decimales, por ejemplo:

```
Dim valor As Double
valor = 1578
Dim formato As String = Format(valor, "#,##0")
```

El valor contenido en la variable formato es 1,578 y si fuera un valor decimal como 1578.68 el resultado sería 1,579. Ahora, si la intención es especificar con el formato y símbolo de moneda, se puede usar el siguiente código:

```
Dim valor As Double
valor = 5500.7
Dim formato As String = Format(valor, "$#,##.00")
```

El valor contenido en la variable Formato es $5500.70; para valores porcentuales se puede usar el siguiente código:

```
Dim valor As Double
valor = 1.25
Dim formato As String = Format(valor, "0%")
```

El valor contenido en la variable formato es 125 % y si fuera un valor decimal como 1.255 el resultado sería 1.26%. Finalmente, si se necesita especificar en el valor porcentual un factor decimal se puede usar el siguiente código:

```
Dim valor As Double
valor = 1.255
Dim formato As String = Format(valor, "0.00%")
```

El valor contenido en la variable formato es 125.50 %.

4. **FormatCurrency**: Permite devolver un valor en un formato tipo moneda con el símbolo especificado en la configuración regional de sus sistemas operativos.

```
Dim monto As Double = 9857
MessageBox.Show("El monto es: " & FormatCurrency(monto, 2))
```

Se declara y asigna un valor a la variable Monto, el cual es impreso en formato de moneda utilizando la función FormatCurrency. Es necesario tener en cuenta que, en los parámetros de la función, primero se especifica la variable que contiene un valor numérico y luego el numero de decimales para el valor. Asimismo, el resultado mostrará el valor numérico expresado en moneda, es decir, con el símbolo de moneda y el separador de miles. El resultado es $9,857.00.

5. **FormatDateTime**: Especifica diferentes tipos de salida para un valor de tipo fecha y hora.

```
Dim fecha As DateTime = "04/04/17"
MessageBox.Show("La fecha es: " & FormatDateTime(fecha, DateFormat.ShortDate))
```

Se declara la variable fecha de tipo DateTime con un valor predeterminado que permitirá imprimir en diferentes formatos, luego en la impresión se usa el formato de fecha y hora, especificando primero la variable Fecha y luego el formato de salida, el cual se podría usar:

- DateFormat.GeneralDate: Muestra una fecha corta y si hubiera una hora lo muestra como hora larga.
- DateFormat.LongDate: Muestra una fecha con el formato de fecha larga.
- DateFormat.ShortDate: Muestra una fecha con el formato de fecha corta.
- DateFormat.LongTime: Muestra una hora con el formato de hora larga.
- DateFormat.ShortTime: Muestra una hora con el formato de 24 horas (hh:mm).

6. **FormatNumber**: Permite devolver un valor numérico en una expresión numérica con separadores de miles y decimales.

```
Dim cantidad As Integer = 2967
MessageBox.Show("El valor es: " & FormatNumber(cantidad, 2))
```

Se declara la variable Cantidad de tipo entero con un valor predeterminado de cuatro cifras para poder agregar el símbolo de miles en su impresión, luego, en la impresión se usa el formato de número especificando dos decimales, el resultado es 2,967.00.

7. **FormatPercent**: Permite mostrar un valor decimal multiplicado por cien adicionando el símbolo de porcentaje.

```
Dim porcentaje As Double = 0.25
MessageBox.Show("El valor porcentual es: " & FormatPercent(porcentaje))
```

Se declara la variable Porcentaje de tipo double con un valor 0.25 para mostrar 25 % en la impresión usando la función formatPercent; se debe tener en cuenta que el resultado se mostrará en dos decimales y con el símbolo %; el resultado es 25.00 %.

8. **GetChar**: Permite devolver un carácter desde una cadena de caracteres, para lo cual se especifica el índice de cada carácter a partir de la posición 1.

```
Dim cadena As String = "Visual Basic"
For i = 1 To cadena.Length
    MessageBox.Show(GetChar(cadena, i))
Next
```

Se declara la variable Cadena de tipo String con un valor inicial de Visual Basic; para recorrer por todos los caracteres se usa la sentencia For desde la posición 1 hasta la ultima posición de la cadena, usando la función Length. Finalmente, se muestran los caracteres uno a uno usando la función GetChar y en la especificación de obtención de carácter se colocan I para el cambio de posición en el carácter.

9. **Join**: Permite unir dos o más palabras en un sola cadena de caracteres.

```
Dim productos() As String = {"Lavadora", "Refrigeradora", "Licuadora"}
Dim listaProductos As String = Join(productos, "-")
MessageBox.Show("La lista de productos: " & listaProductos)
```

Se declara un arreglo de productos tipo String, el cual permitirá almacenar varios productos en una matriz; desde aquí se declara la variable listaProductos de tipo cadena, el cual contendrá todos los productos separados por un guion. En la función Join solo se puede especificar la variable de tipo matriz y el separador entre los valores, el valor por defecto es un espacio en blanco.

10. **LCase**: Devuelve una cadena de caracteres en minúsculas.

```
Dim correo As String = "MTORRES@GMAIL.COM"
MessageBox.Show("El correo es: " & LCase(correo))
```

Se inicializa la variable Correo con un email en mayúsculas para mostrarlo en minúsculas usando la función LCase; esta función no necesita parámetros adicionales, solo especificar qué palabra se convertirá en minúsculas.

11. **Left**: Devuelve caracteres del lado izquierdo de una cadena de caracteres.

```
Dim cadena As String = "Visual Basic"
MessageBox.Show(Strings.Left(cadena, 6))
```

Desde la cadena se devuelve la palabra Visual usando la función Left, en donde la especificación 6 indica la cantidad de caracteres del lado izquierdo que se desea obtener.

12. **Len**: Determina el numero total de caracteres que puede contener un valor de tipo String.

```
Dim cadena As String = "Visual Basic"
MessageBox.Show(Len(cadena))
```

El código devuelve como resultado el valor 12 pues determina que dicho valor es el número total de caracteres incluidos en la cadena. Se debe tener en cuenta que los espacios en blanco también son considerados caracteres.

13. **Ltrim**: Devuelve la copia de una cadena de caracteres obviando los espacios en blanco que pudieran estar en el lado izquierdo de la cadena.

```
Dim cadena As String = "    Visual Basic"
MessageBox.Show("La cadena es: " & LTrim(cadena))
```

El valor devuelto es la misma cadena pero sin espacios en blanco desde el lado izquierdo.

14. **Mid:** Devuelve una cadena a partir de una posición inicial y una cantidad de caracteres.

```
Dim cadena As String = "Visual Basic"
MessageBox.Show("La nueva cadena es: " & Mid(cadena, 8, 5))
```

El valor devuelto es Basic, ya que dicho valor inicia desde la posición 8 y desde allí queremos obtener 5 caracteres.

15. **Replace**: Permite buscar y reemplazar un valor por otro en una cadena de caracteres; la función termina los reemplazos al final de la cadena.

```
Dim cadena As String = "Visual Basic"
MessageBox.Show("La nueva cadena es: " & Replace(cadena, "a", "A"))
```

El valor devuelto es “VisuAl BAsic” .

16. **Right**: Devuelve caracteres del lado derecho de una cadena de caracteres.

```
Dim cadena As String = "Visual Basic"
MessageBox.Show(Strings.Rigtht(cadena, 5))
```

Desde la cadena se devuelve la palabra Basic usando la función Right, en donde la especificación 5 indica la cantidad de caracteres del lado derecho se desea obtener.

17. **Rtrim**: Devuelve la copia de una cadena de caracteres obviando los espacios en blanco que pudieran estar en el lado derecho de la cadena.

```
Dim cadena As String = "Visual Basic     "
MessageBox.Show("La cadena es: " & RTrim(cadena))
```

El valor devuelto es la misma cadena, pero sin espacios en blanco desde el lado derecho.

18. **Space**: Permite asignar un numero determinado de espacios en blanco en una combinación de caracteres.

```
Dim cadena1 As String = "Visual"
Dim cadena2 As String = "Basic"
MessageBox.Show("La nueva cadena es: " & cadena1 & Space(1) & cadena2)
```

El valor resultado es Visual Basic, ya que las cadenas han sido concatenadas mediante el operador &, además de agregar un espacio entre ellos usando la función space, con un parámetro de uno, el cual representa la cantidad de espacios en blanco.

19. **Split**: Devuelve un bloque de caracteres a partir de una cadena. El criterio de separación son los espacios en blanco; estos valores deberán colocarse en una matriz de caracteres.

```
Dim cadena As String = "Visual Basic 2015"
Dim palabras() As String = Split(cadena)
For i = 0 To palabras.Count - 1
   MessageBox.Show("Posicion " & i & ": " & palabras(i))
Next
```

Los valores devueltos son: Posición 0: Visual, Posición 1: Basic y Posición 2: 2015. Como vemos la cadena de caracteres formo un conjunto de elementos los cuales son guardados en una matriz llamada palabras; los cuales pueden ser vistas recorriendo la matriz mediante la estructura repetitiva For.

20. **StrConv**: Permite realizar diversas conversiones a una cadena de caracteres. A continuación, una lista de especificaciones que se puede realizar.
    - **VbStrConv.None**: No realiza ninguna conversión sobre la cadena.
    - **VbStrConv.UpperCase**: Convierte la cadena de caracteres en mayúsculas.
    - **VbStrConv.LowerCase**: Convierte la cadena de caracteres en minúsculas.
    - **VbStrConv.ProperCase**: Convierte la primera letra de cada palabra de la cadena a mayúsculas.

```
Dim cadena As String = "Visual Basic"
Dim nuevaCadena As String = StrConv(cadena, VbStrConv.Uppercase)
MessageBox.Show("La nueva cadena es: " & nuevaCadena)
```

El resultado de la cadena es VISUAL BASIC porque se especifica Uppercase en el parámetro de la función StrConv.

21. **StrReverse**: Devuelve una cadena con los caracteres invertidos.

```
Dim cadena As String = "Visual Basic 2015"
Dim reversa As String = StrReverse(cadena)
MessageBox.Show("La nueva cadena es: " & reversa)
```

El valor devuelto es "5102 cisaB lausiV" que resulta de la inversión de la cadena Visual Basic 2015.

22. **Trim**: Devuelve la copia de una cadena de caracteres obviando los espacios en blanco que pudieran estar en el lado derecho o izquierdo de la cadena.

```
Dim cadena As String = "   Visual Basic 2015    "
MessageBox.Show("La cadena es: " & Trim(cadena))
```

El valor devuelto es la misma cadena inicializada en la variable Cadena, pero sin espacios en blanco tanto en el lado izquierdo como derecho.

23. **UCase**: Devuelve una cadena de caracteres en mayúsculas.

```
Dim codigo As String = "pro-123"
MessageBox.Show("El código es: " & UCase(codigo))
```

El valor devuelto es PRO-123 ya que la función UCase convierte todos los caracteres en mayúsculas.

6.5.2 Funciones de fecha y hora

Para trabajar con las funciones de tipo fecha y hora se deben considerar los siguientes intervalos:

Intervalo	Valor simplificado	Referencia
DateInterval.Day	"d"	Día
DateInterval.DayOfYear	"y"	Día
DateInterval.Hour	"h"	Hora
DateInterval.Minute	"n"	Minuto
DateInterval.Month	"m"	Mes
DateInterval.Quarter	"q"	Trimestre
DateInterval.Second	"s"	Segundo
DateInterval.Weekday	"w"	Semana
DateInterval.WeekOfYear	"ww"	Semana del calendario
DateInterval.Year	"yyyy"	Año

1. **DateAdd**: Devuelve un valor numérico que corresponde al código ASCII de un determinado carácter.

```
Dim fechaActual As DateTime = DateTime.Now
For i = 1 To 5
  MessageBox.Show(DateAdd(DateInterval.Month, i, fechaActual).ToShortDateString)
Next
```

El resultado del código es mostrar cinco fechas posteriores según el mes, para esto declaramos la variable FechaActual, que se inicializa con la fecha actual, luego la estructura repetitiva tendrá la misión de devolver cinco valores. Si hoy fuera 05/12/2017 el resultado sería:

```
06/12/2017
07/12/2017
08/12/2017
09/12/2017
10/12/2017
```

2. **DateDiff**: Devuelve un valor numérico que corresponde al código ASCII de un determinado carácter.

```
Dim fecha1 As Date = "02/01/2017"
Dim fecha2 As Date = "02/20/2017"
Dim totalDias As Integer = DateDiff(DateInterval.Day, fecha1, fecha2)
MessageBox.Show("Total de dias es: " & totalDias)
```

El resultado es 19 ya que entre la dos fechas existe una diferencia entre 20 y 1; esto se logra especificando el tipo de intervalo DateInterval.Day en el parámetro de la función DateDiff. Debe tener en cuenta que en una diferencia se podría dar el caso de mostrar valores negativos, por lo tanto, la fecha inicial siempre debe ser mayor que la segunda.

3. **DateString**: Devuelve la fecha actual con el formato mm/dd/aaaa convertido en cadena; es decir, con esta función no se necesita convertir del tipo fecha a cadena.

```
MessageBox.Show("La fecha actual es: " & DateString)
```

El resultado del código es mostrar la fecha actual con el formato String.

4. **Day**: Devuelve un valor numérico que representa al numero de día en el mes de una determinada fecha.

```
Dim diaActual As Integer = DateAndTime.Day(Now)
MessageBox.Show("El dia actual es: " & diaActual)
```

El valor resultante dependerá del día en que se encuentre ya que la fecha evaluada es la fecha actual; especificada con la clausula Now.

5. **Hour**: Devuelve un valor numérico que representa la hora entre los valores 0 a 23, en una determinada hora.

```
Dim horaActual As Integer = DateAndTime.Hour(Now)
MessageBox.Show("El hora actual es: " & horaActual)
```

El valor resultante dependerá de la hora en que se encuentre ya que la hora evaluada es la actual, especificada con la clausula Now.

6. **Minute**: Devuelve un valor numérico que representa a los minutos entre los valores 0 a 59 en una determinada hora.

```
Dim minutoActual As Integer = DateAndTime.Minute(Now)
MessageBox.Show("El minuto actual es: " & minutoActual)
```

El valor resultante dependerá de la hora en que se encuentre ya que la hora evaluada es la actual, especificada con la clausula Now.

7. **Month**: Devuelve un valor numérico que representa al numero de mes entre los valores de 1 a 12 de una determinada fecha.

```
Dim mesActual As Integer = Month(Now)
MessageBox.Show("El mes actual es: " & mesActual)
```

El valor resultante dependerá de la fecha en que se encuentre, es a partir de la fecha actual que se captura el numero de mes.

8. **MonthName**: Devuelve un valor numérico que representa al numero de día en el mes de una determinada fecha.

```
Dim mesActual As String = MonthName(Month(Now))
MessageBox.Show("El mes actual es: " & mesActual)
```

El valor resultante dependerá de la fecha en que se encuentre, es a partir de la fecha actual que se captura el nombre del mes en ingles.

9. **Now**: Devuelve un valor de tipo DateTime que representa la fecha y la hora actual.

```
Dim fechaActual As DateTime = Now
MessageBox.Show("La fecha actual es: " & fechaActual)
```

El valor resultante dependerá de la fecha y la hora en que se encuentre; se debe tener en cuenta que este valor es obtenido de la fecha del sistema operativo.

10. **Second**: Devuelve un valor de tipo entero que representa a los segundos de una determinar hora.

```
Dim segundoActual As Integer = Second(Now)
MessageBox.Show("El segundo actual es: " & segundoActual)
```

El valor resultante dependerá de la hora en que se encuentre, debe tener en cuenta que este valor es obtenido de la hora mostrada desde el sistema operativo.

11. **TimeString**: Devuelve un valor de tipo string que representa a la hora actual; se debe considerar que dicha hora no necesita ser parseada por que la función ya lo expresa en el tipo String.

```
Dim horaActual As DateTime = TimeString
MessageBox.Show("La hora actual es: " & horaActual)
```

El valor resultante dependerá de la hora en que se encuentre; debe tener en cuenta que este valor es obtenido de la fecha mostrada en el sistema operativo.

12. **Today**: Devuelve un valor de tipo DateTime que representa la fecha actual.

```
Dim fechaActual As Date = Today
MessageBox.Show("La fecha actual es: " & fechaActual)
```

El valor resultante dependerá de la fecha en que se encuentre; debe tener en cuenta que este valor es obtenido de la fecha del sistema operativo.

13. **WeekDay**: Devuelve un valor de tipo entero que representa el numero día entre 1 y 7 de una determinada fecha; se debe tener en cuenta que se cuenta como primer día al domingo.

```
Dim diaActual As Integer = Weekday(Now)
MessageBox.Show("El dia actual es: " & diaActual)
```

El valor resultante dependerá de la fecha en que se encuentre; se debe tener en cuenta que este valor es obtenido de la fecha mostrado en el sistema operativo.

14. **WeekdayName**: Devuelve un valor de tipo DateTime que representa la fecha y la hora actual.

```
Dim diaActual As String = WeekdayName(Weekday(Now))
MessageBox.Show("El dia actual es: " & diaActual)
```

El valor resultante dependerá de la fecha en que se encuentre, el nombre de la semana obtenido se mostrará en el idioma configurado en el sistema operativo.

15. **Year**: Devuelve un valor de tipo entero que representa al año de una determinada fecha.

```
Dim añoActual As Integer = Year(Now)
MessageBox.Show("el año actual es: " & añoActual)
```

El valor resultante dependerá de la fecha en que se encuentre; se debe tener en cuenta que este valor es obtenido de la fecha mostrada por el sistema operativo.

6.5.3 Funciones matemáticas

1. **Abs**: Devuelve el valor absoluto de un determinado valor numérico.

```
Dim n As Integer = -7
MessageBox.Show("El valor de N es: " & Math.Abs(n))
```

El valor resultante es siete positivo; se debe tener en cuenta que para usar la función debe, primero, colocar la cláusula Math por pertenecer al grupo de funciones matemáticas.

2. **Ceiling**: Devuelve el valor entero redondeado a partir de un valor decimal.

```
Dim numero As Double = 5.26767
MessageBox.Show("El valor es: " & Math.Ceiling(numero))
```

El valor resultante es seis, ya que es el valor redondeado del numero decimal 5.26767.

3. **Exp**: Devuelve el valor e que representa la base de los logaritmos naturales.

```
MessageBox.Show("El valor es: " & Math.Exp(1))
```

El valor resultante es 2.718281828459O5 que es valor de e.

4. **Floor**: Devuelve el valor entero sin aplicar ningún tipo de redondeo.

```
Dim numero As Double = 5.6767
MessageBox.Show("El valor es: " & Math.Floor(numero))
```

El valor resultante es cinco, ya que es valor entero redondeado sin aplicar exceso ni defecto.

5. **Log**: Devuelve el valor logaritmo natural de un determinado valor numérico.

```
Dim numero As Integer = 100
MessageBox.Show("El logaritmo base 10 es: " & Math.Log(numero))
```

El valor resultante es 4.61, que es el resultado del logaritmo natural de 100.

6. **Log10**: Devuelve el valor logaritmo en base 10 de un valor numérico.

```
Dim numero As Integer = 100
MessageBox.Show("El logaritmo base 10 es: " & Math.Log10(numero))
```

El valor resultante es dos ya que ese es el resultado del logaritmo de 100 en base 10.

7. **Max**: Devuelve el mayor valor obtenido de la comparación de dos números.

```
Dim n1 As Integer = 7
Dim n2 As Integer = 14
MessageBox.Show("El menor valor es: " & Math.Max(n1,n2))
```

El valor resultante es catorce ya que es el mayor valor entre 7 y 14.

8. **Min**: Devuelve el menor valor obtenido de la comparación de dos números.

```
Dim n1 As Integer = 7
Dim n2 As Integer = 14
MessageBox.Show("El menor valor es: " & Math.Min(n1,n2))
```

El valor resultante es siete ya que es el menor valor entre 7 y 14.

9. **Pow**: Devuelve el valor elevado de un numero a una determinada base.

```
Dim n As Double = 8
MessageBox.Show("El valor " & n & " al cubo es :" & Math.Pow(n, 3))
```

El valor resultante es 512; se debe tener en cuenta que la base de la potencia puede ser un valor directo o por medio de una variable, pero siempre debe ser un valor numérico entero. A continuación, un segundo caso donde se muestra la raíz cubica de un valor:

```
Dim n As Double = 8
MessageBox.Show("El raíz cubica de " & n & " es :" & Math.Pow(n, 1 / 3))
```

La especificación de la raíz siempre será expresada por el valor uno sobre el valor de la raíz.

10. **Round**: Devuelve el valor redondeado a un numero determinado de decimales. Además, se debe considerar que el redondeo se realiza al exceso del valor.

```
Dim n As Double = 4.58
MessageBox.Show("El valor redondeado es: " & Math.Round(n, 1))
```

El valor resultante es 4.6, ya que en los parámetros de la función Round se especifica el valor 1 en el factor de redondeo.

11. **Sqrt**: Devuelve el valor raíz cuadrada de un valor numérico.

```
Dim n As Double = 4
MessageBox.Show("La raíz cuadrada es: " & Math.Sqrt(n))
```

El valor resultante es dos ya que ese es el resultado de la raíz cuadrada de cuatro; se debe tener en cuenta que devuelve el valor de la raíz cuadrada de cualquier tipo de numero.

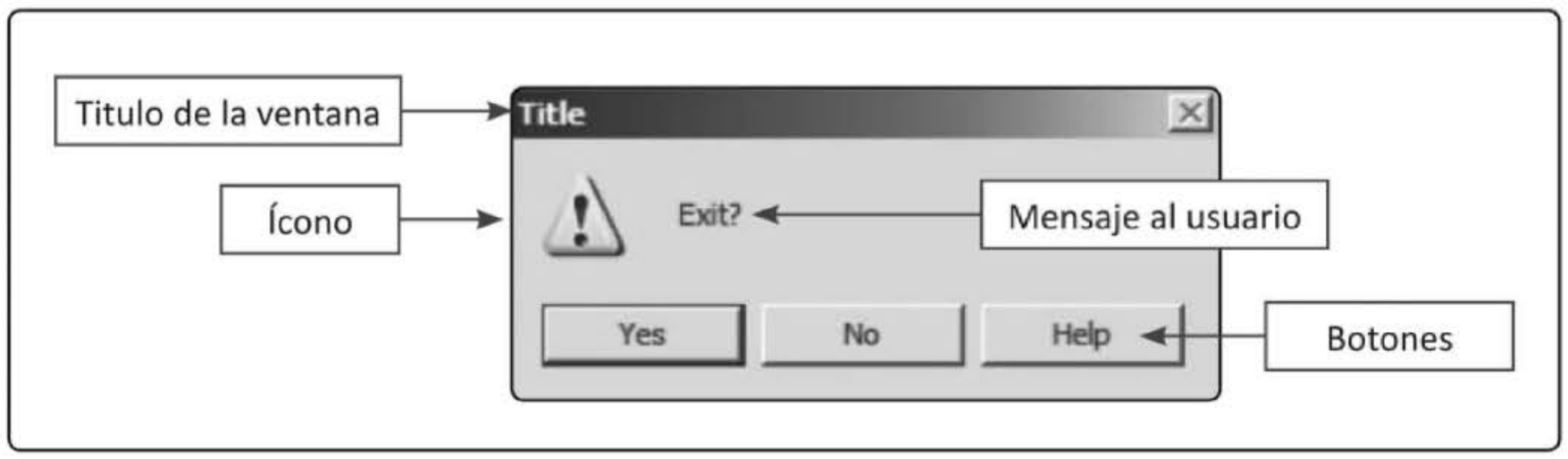
12. **Truncate**: Devuelve la parte entera de un valor numérico decimal como un double o single sin especificaciones de decimales.

```
Dim n As Double = 20.856
MessageBox.Show("El valor truncado es: " & Math.Truncate(n))
```

El valor es veinte ya que la función Truncate solo obtiene la parte entera del valor numérico.

6.5.4 Funciones interaction

1. **MsgBox**: Muestra un mensaje mediante un cuadro de dialogo, asimismo devuelve un numero entero de acuerdo al botón seleccionado. A continuación, las partes de un cuadro de mensaje.



Los valores devueltos por los botones del cuadro de mensaje son los siguientes:

Constante	Valor
OK (Aceptar)	1
Cancel (Cancelar)	2
Abort (Anular)	3
Retry (Reintentar)	4
Ignore (Omitir)	5
Sí	6
No	7

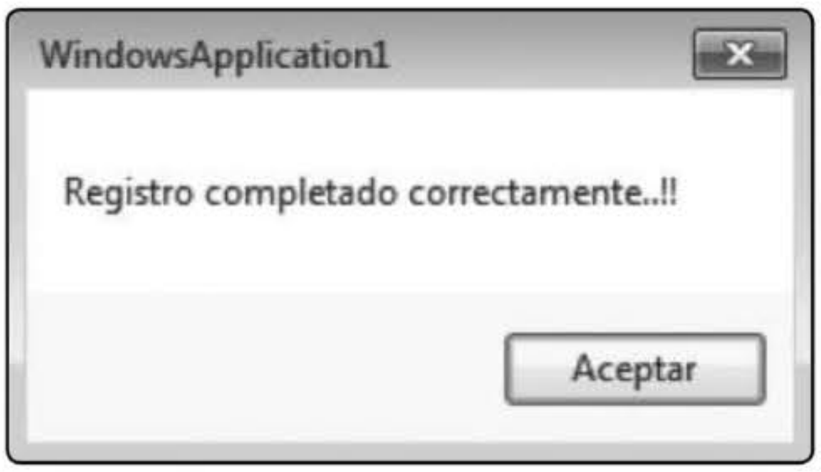
A continuación, los valores que representan a los botones y a los íconos en el cuadro de mensaje:

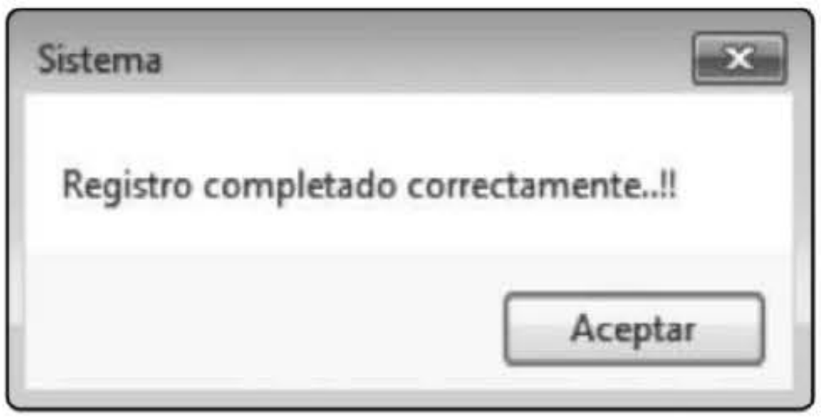
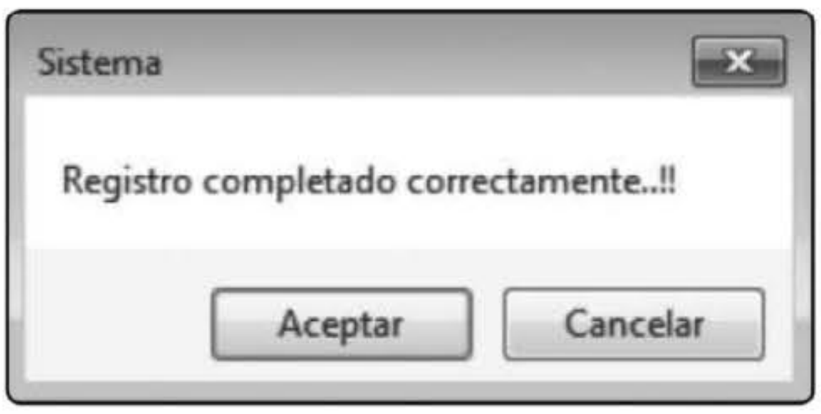
Miembro	Valor	Descripción
OKOnly	0	Muestra solo el botón **Aceptar**.
OKCancel	1	Muestra los botones **Aceptar** y **Cancelar**.
AbortRetryIgnore	2	Muestra los botones **Anular**, **Reintentar** y **Omitir**.
YesNoCancel	3	Muestra los botones **Sí**, **No** y **Cancelar**.
YesNo	4	Muestra los botones **Sí** y **No**.
RetryCancel	5	Muestra los botones **Reintentar** y **Cancelar**.
Critical	16	Muestra el ícono Mensaje crítico.
Question	32	Muestra el ícono Consulta de advertencia.
Exclamation	48	Muestra el ícono Mensaje de advertencia.
Information	64	Muestra el ícono Mensaje de información.

Los valores de 0 a 5 definen el tipo de botón, mientras que los demás números muestran un tipo de ícono en el cuadro de mensaje. El formato para el cuadro de mensaje se muestra a continuación:

```
Msgbox("Mensaje al usuario")
Msgbox("Mensaje al usuario",,"Titulo de la ventana")
Msgbox("Mensaje al usuario", MsgBoxStyle , "Titulo de la ventana")
Dim valor As Integer = MsgBox("Mensaje", MsgBoxStyle)
```

A continuación, una lista de posibilidades de uso de la función MsgBox

Mostrar un mensaje que permita indicar al usuario que el registro de datos se ha completado correctamente de manera simple. 	MsgBox("Registro completado correctamente..!!")

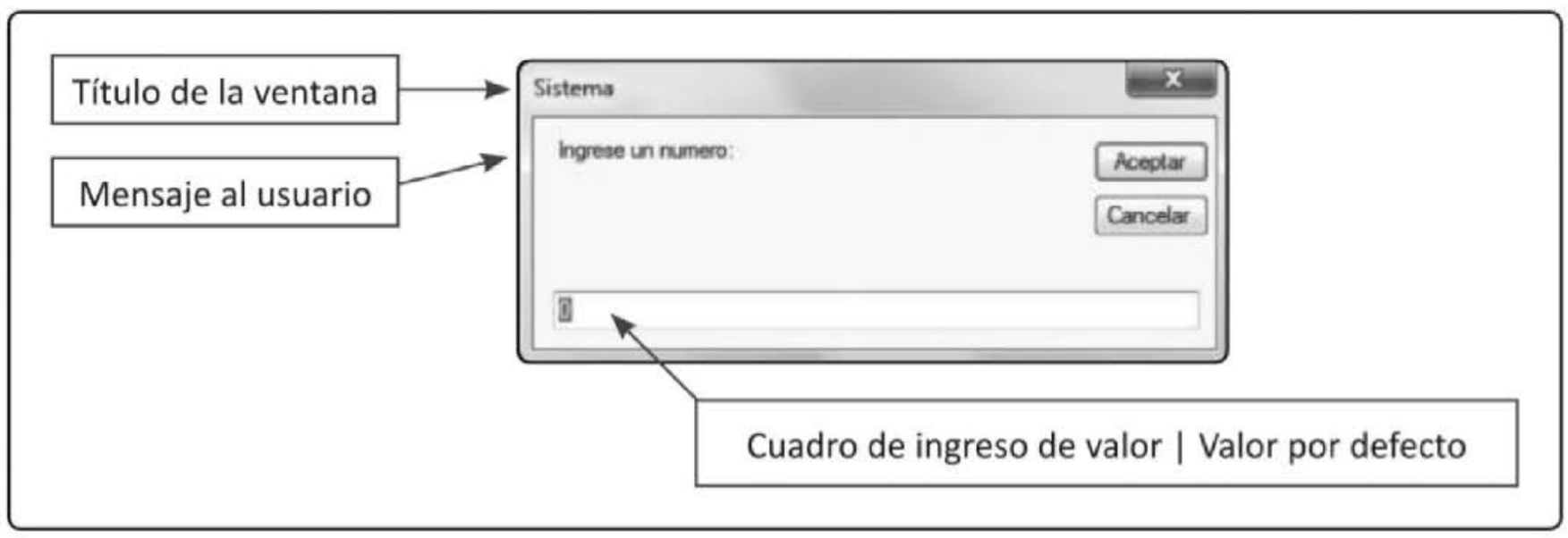
Mostrar un mensaje que permita indicar al usuario que el registro de datos se ha completado correctamente, además de mostrar un título adecuado en la ventana de dialogo del mensaje. 	**MsgBox("Registro completado correctamente..!!",,** **"Sistema")**
Mostrar un mensaje que permita indicar al usuario que el registro de datos se ha completado correctamente, mostrando un título y los botones **Aceptar** y **Cancelar**. 	**MsgBox("Registro completado correctamente..!!",** **MsgBoxStyle.OkCancel,** **"Sistema")** **MsgBox("Registro completado correctamente..!!",** **1,** **"Sistema")**
Mostrar un mensaje que permita indicar al usuario que el registro de datos se ha completado correctamente asignando un título, los botones **Aceptar** y **Cancelar**, además del el ícono Question. 	**MsgBox("Registro completado correctamente..!!",** **MsgBoxStyle.** **OkCancel+MessageBoxIcon.Question,** **"Sistema")** **MsgBox("Registro completado correctamente..!!",** **1 + 32,** **"Sistema")**

Mostrar un mensaje tipo de salida que permita al usuario seleccionar una opción de salida en una aplicación.
 	
	Dim r As Integer = MsgBox("Esta seguro de salir",
	MsgBoxStyle.YesNo)
	If r = vbYes Then
	Me.Close()
	End If

2. **InputBox**: Muestra un mensaje en un cuadro de diálogo; normalmente se usa para ingresar algún valor a la aplicación, el cual debe ser asignado a una variable para que pueda tomarse como parte de la aplicación. La función InputBox tiene el siguiente formato:

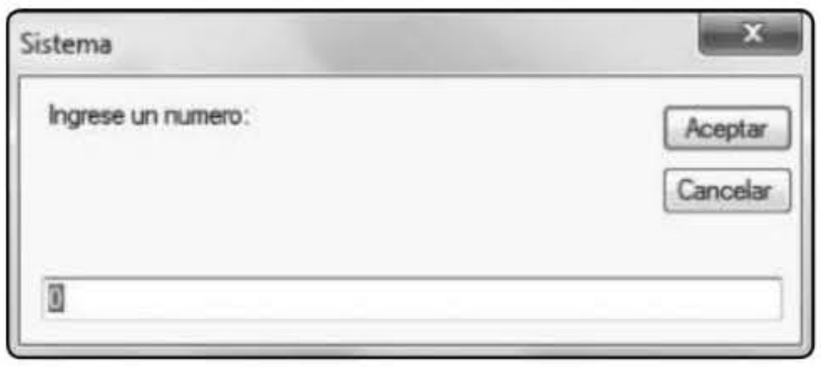
```
Dim variable As Tipo = InputBox("Mensaje","Titulo",ValorxDefecto,
                                                    PosicionX, PosicionY)
```

A continuación, los elementos que compone una ventana de dialogo InputBox:



Donde la variable declarada debe ser del mismo tipo que el valor que se ingresará por la ventana InputBox. Entre los parámetros de la función InputBox se tiene al mensaje, el cual muestra un texto explicativo al usuario; el titulo representa al texto que se mostrará en el titulo de la ventana InputBox; valor por defecto se refiere a uno que se asignará a la variable solo si el usuario no registra valor en el cuadro de diálogo InputBox, valor que debe ser del mismo tipo. Esta ventana de diálogo también se puede direccionar a una determinada posición mediante los parámetros posicionX y posicionY. Para un funcionamiento simple, se recomienda usar solo los parámetros mensaje, título y valor por defecto.

A continuación, algunas funcionalidades que se le puede dar a la función InputBox:

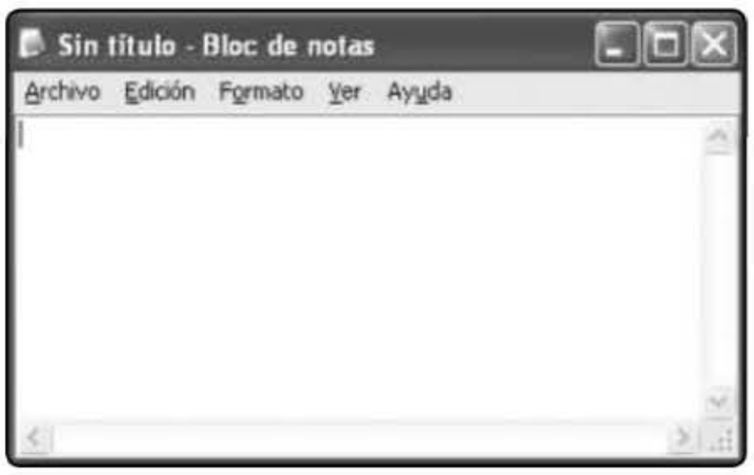
Obtener un valor de tipo numérico a partir de una función de tipo InputBox, además deberá mostrar el valor por defecto cero cuando el usuario no registre valor. 	Dim numero As Integer = InputBox("Ingrese un numero: ", "Sistema", 0)

Obtener un valor de tipo cadena a partir de una función de tipo InputBox. 	Dim descripción As String = InputBox("Ingrese descripción:", "Sistema")

3. **Shell**: Permite ejecutar una aplicación y devuelve un valor entero que contiene el identificador de proceso del programa si aún se está ejecutando. La función Shell tiene el siguiente formato:

```
Dim valor As Integer = Shell("ruta\aplicacion.exe")
```

A continuación, algunas funcionalidades que se le puede dar a la función Shell:

Implementar el código para mostrar la calculadora de windows mediante la función Shell. 	Dim calculadora As Integer **calculadora=Shell("C:\Windows\ system 32\calc.exe"**
Implementar el código para mostrar la calculadora de windows mediante la función Shell. 	Dim bloc As Integer **bloc = Shell("C:\Windows\system32\ notepad.exe",** **Style:=AppWinStyle. MaximizedFocus)**

6.6 Microsoft Visual Basic Information

1. **IsDate**: Devuelve un valor de tipo boleano a partir de la evaluación de un valor de tipo fecha u hora válida. Devuelve True si la fecha u hora evaluada es valida, caso contrario devuelve Falso.

   A continuación, el código para verificar que un valor de tipo fecha es valido o no, mediante un cuadro de diálogo de mensaje:

```
Dim fecha As String = "30/02/2017"
If IsDate(fecha) Then
   MsgBox("Fecha valida")
   Else
   MsgBox("Fecha NO valida")
End If
```

   Se inicia por declarar la variable fecha de tipo cadena y aloja un valor de tipo fecha no válida, pues el 30 de febrero no existe. En la condición se usa la función IsDate para evaluar a dicho valor, si el resultado evaluado en la condición es True, entonces mostrará el mensaje Fecha válida, caso contrario mostrará el mensaje Fecha NO válida.

2. **IsNumeric**: Devuelve un valor de tipo boleano a partir de la evaluación de un valor de tipo numérico valido. Devuelve True si el valor numérico es valido, caso contrario devuelve False.

   A continuación, el código para verificar si el valor de tipo numérico es válido o no, mediante un cuadro de dialogo de mensaje:

```
Dim numero As String = "1020"
If IsNumeric(numero) Then
   MsgBox("Numero válido")
   Else
   MsgBox("Numero NO válido")
End If
```

   Se inicia por declarar la variable número de tipo cadena, el cual aloja el valor 1020, el cual es válido como numero; para evaluarlo se usa la función IsNumeric y, mediante la función MsgBox, se muestra los mensajes según el valor. Para este caso se mostrará Numero valido.

6.7 Funciones de conversión

Son funciones que permiten convertir una expresión en un tipo de datos especificado por el usuario. Se debe tener en cuenta que estas funciones se compilan en una misma línea, es decir, el código de conversión forma parte del código que evalúa la expresión.

Existen funciones de conversión propias de Visual Basic, así como funciones de conversión de Net Framework, los cuales realizan la misma tarea pero no necesariamente devuelven el mismo resultado. Se recomienda usar función de conversión de Visual Basic ya que son mas cortas y además son procesados de manera eficaz por el compilador.

A continuación, una lista de funciones de conversión mas usados:

Función	Tipo de datos devuelto	Intervalo de valores del argumento expression
CDate	Date	Convierte un valor de tipo cadena en una de tipo fecha u hora. Dim cadena As String = "01/02/2017" Dim fecha As Date = CDate(cadena) MsgBox("El tipo de datos es: " & fecha.GetType().ToString()) Se inicia por declarar una variable de tipo cadena que contiene una fecha válida, luego pasar dicha información a una variable fecha de tipo Date; para esto se debe convertir la fecha de tipo cadena en un valor de tipo fecha usando la función CDate. Finalmente, realizar la impresión de la variable fecha usando la función GetType, el cual muestra el tipo de datos.
CDbl	Double	Convierte un valor en un tipo de doble precisión. Dim numero As Integer = 20 Dim valorDecimal As Double = CDbl(numero) MsgBox("El tipo de datos es: " & valorDecimal.GetType().ToString()) Se inicia por declarar la variable número de tipo entero con un valor inicial de 20, luego este pasará a la variable valorDecimal de tipo Double para lo cual se realiza una conversión con la función CDbl. Finalmente, se imprime el valor mediante la función GetType para mostrar el tipo de datos actual de la variable ValorDecimal.
CInt	Integer	Convierte un valor en un tipo entero. Dim numero As Integer = CInt(txtNumero.Text) MsgBox("El tipo de datos es: " & numero.GetType().ToString()) Se inicia por declarar la variable numero de tipo entero, al cual se le asigna como valor inicial el valor ingresado en la caja de texto “txtNumero”, para esto se debe convertir dicho valor en un entero usando la función CInt. Finalmente, se imprime la variable número especificando su tipo de datos con la función GetType.

CLng	Long	Convierte un valor en un tipo entero largo. Dim cantidad As Long = CLng(txtCantidad.Text) MsgBox("El tipo de datos es: " & cantidad.GetType().ToString()) Se inicia por declarar la variable cantidad de tipo entero largo, al cual se le asigna como valor inicial el valor ingresado en la caja de texto "txtCantidad", para esto se debe convertir dicho valor en un entero usando la función CLng. Finalmente, se imprime la variable cantidad especificando su tipo de datos con la función GetType.
CStr	String	Convierte un valor en un tipo cadena. Dim numero As Integer = 20 Dim cadena As String = CStr(numero) MsgBox("El tipo de datos es: " & cadena.GetType().ToString()) Se inicia por declarar la variable número de tipo entero con un valor inicial de 20, luego este pasará a la variable cadena de tipo String, para lo cual se realiza una conversión con la función CStr. Finalmente, se imprime el valor mediante la función GetType para mostrar el tipo de datos actual de la variable cadena.

6.8 Función lambda

A partir de la versión 2005 del Visual Studio se introdujo el concepto de métodos anónimos y es así como algunos llaman a las expresiones lambda. Una expresión lambda tiene un trabajo similar a los métodos con valor de retorno, lo que lo diferencia es la forma de implementarlo, ya que permite compactar el código de un determinado proceso.

Finalmente, se puede decir que una expresión lambda es simplemente una función o procedimiento sin nombre que solo puede contar con una sentencia. En otros lenguajes de programación, la función lambda es considerado como una función anónima.

A continuación, tres formatos para la implementación de una función lambda:

```
Dim variable = Función(parámetro) Sentencia

Dim variable = Función(parámetro)
                Sentencias
                Return valor
               End Función

variable = (Función(parámetro As TipoDatos) Sentencia)(Parámetro)
```

Donde:

- **Dim variable**: Almacena el resultado de la función lambda que luego será invocada especificando sus parámetros.
- **Variable**: Almacena el resultado de la función lambda; asimismo deberá especificarse los parámetros que pudiera tener la función.
- **Return valor**: Es el valor de devolución que tiene la función lambda; esta sentencia solo se especificará si la función lambda especifica su finalización con End Function.

Aquí, un caso desarrollado básico sobre el uso de la función lambda. Se necesita obtener el 12.5 % por concepto de IVA a partir de un monto usando lambda para dicho calculo, cuyo diseño se muestra a continuación:

Código de solución:

```
Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click
   Dim monto As Double = Double.Parse(txtMonto.Text)
   Dim IVA = Function(m) m * 0.125
   lblIVA.Text = "$" & IVA(monto)
End Sub
```

Se inicia por capturar el monto ingresado mediante el cuadro de texto “txtMonto”; luego se implementa la función lambda IVA, la cual tiene la misión de obtener el valor monto por medio del parámetro m que, dentro de la función, multiplica con 0.125, el cual representa el 12.5 %. Finalmente, se invoca a la función lambda enviando como parámetro a la variable Monto, de tal forma que la llamada sería IVA(monto).

También, se podría usar el siguiente codigo:

```
Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click
   Dim monto As Double = Double.Parse(txtMonto.Text)
   Dim IVA = Function(m)
                Return m * 0.125
             End Function
   lblIVA.Text = "$" & IVA(monto)
End Sub
```

Se debe tener en cuenta que si la implementación lambda es una sola sentencia, entonces se puede realizar en una sola línea, mientras que si hay más sentencias se debe usar la cláusula Return para devolver un resultado y finalizar con la cláusula End Function.

Otra forma de implementar una función lambda puede ser haciéndolo en la misma llamada, tal como se puede observar en el siguiente codigo:

```
Private Sub btnCalcular_Click(sender As Object, e As EventArgs) Handles
btnCalcular.Click
    Dim monto As Double = Double.Parse(txtMonto.Text)
    lblIVA.Text = "$" & (Function(m As Double) m * 0.125)(monto)
End Sub
```

Como se observa en el código, el valor del monto es enviado como parámetro en la misma implementación y todo el resultado es directamente mostrar en el control lblIVA.

Caso desarrollado 1 Subprocedimiento sin parámetros - Fábrica de vidrio

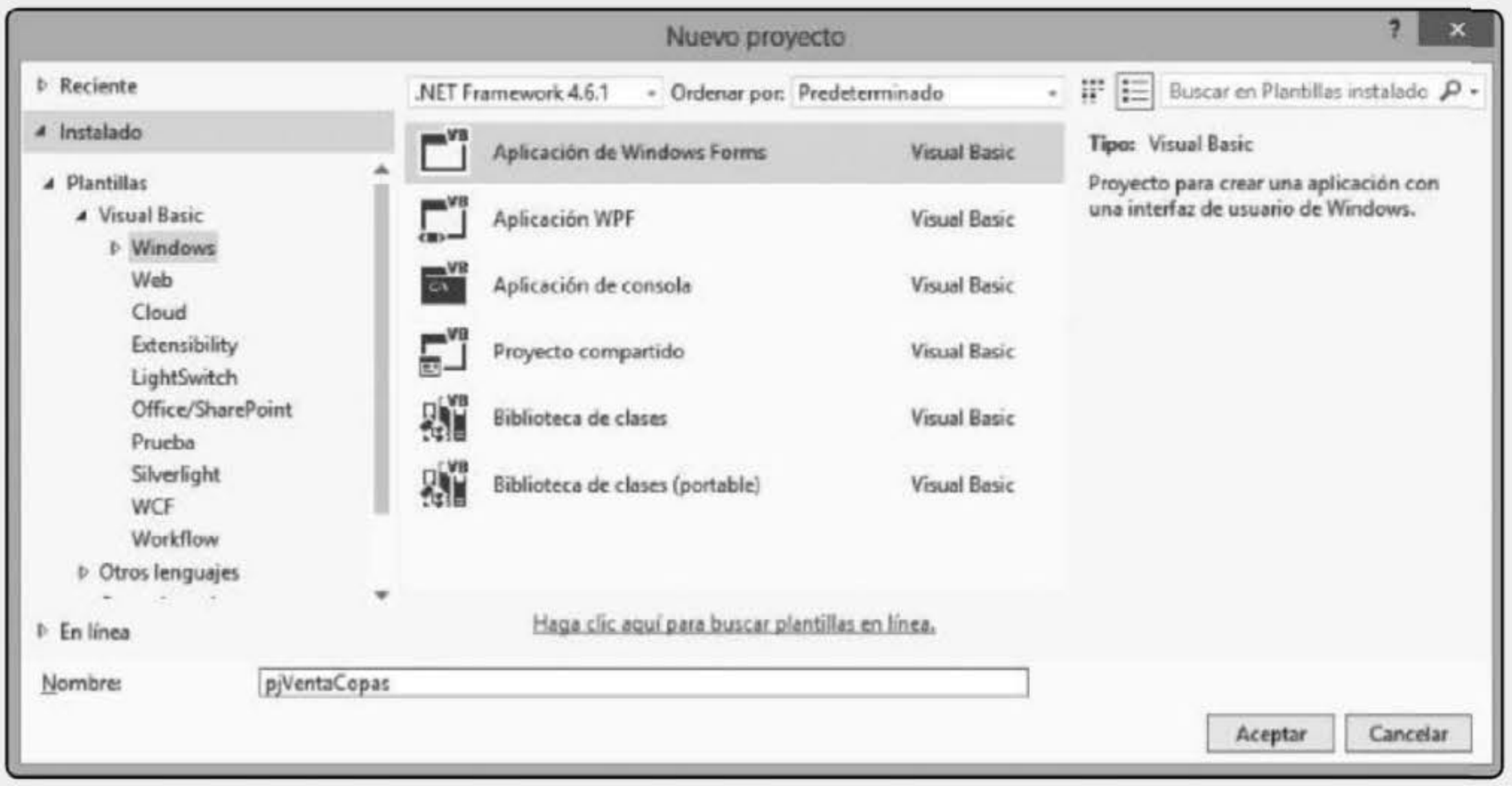
Objetivo	Controlar la venta de copas en un 100 % producida por una fábrica de vidrio.
Referencias	Usar subprocedimientos sin parámetros.
Descripción del caso	Una fábrica de vidrio ha decidido producir dos tipos de copas que serán puestas a la venta según su presentación. La copa sencilla tendrá un precio de introducción de $100.00 y otras talladas a $250.00 por caja de 12 unidades cada una. Desarrollar una aplicación en Visual Basic 2015 que permita controlar las ventas de las copas. **Tipo de copa** / **Costo por caja** Copa sencilla / $100.00 Copa tallada / $250.00 El vendedor del área de ventas es el encargado del registro de venta, para lo cual selecciona un tipo de copa, registra la cantidad y debe mostrar el registro en una lista, donde al final se muestre un cuadro de resumen donde figure el monto acumulado por cada vendedor.
Tener en cuenta	• Use subprocedimientos sin parámetros. • Al iniciar la aplicación se debe mostrar la fecha y la hora de manera automática. • El cuadro combinado de los vendedores debe mostrar los nombres de los vendedores de forma automática al iniciar la aplicación. Debe considerar el uso de enumeraciones para su definición.

	• El cuadro combinado de los tipos de copa debe mostrarse al iniciar la aplicación. Asimismo, se debe considerar que al seleccionar un tipo de copa se debe mostrar el precio según el tipo. • El botón **Cancelar** limpia los controles para un nuevo registro y el foco de ingreso debe ubicarse en el cuadro combinado del vendedor. • El botón **Registrar** envía la información ingresada además del precio y el monto neto de la venta en el control ListView. Luego, se deberá calcular el monto acumulado de todos los netos (Total venta) y, finalmente, mostrar en un cuadro de resumen los montos acumulados según el nombre del vendedor en un control ListBox. • Finalmente, se debe implementar un método de validación de ingreso de datos que al no seleccionar un vendedor, tipo de copa o ingresar la cantidad, emita un mensaje de error al usuario.
Formulario propuesto	

Solución:

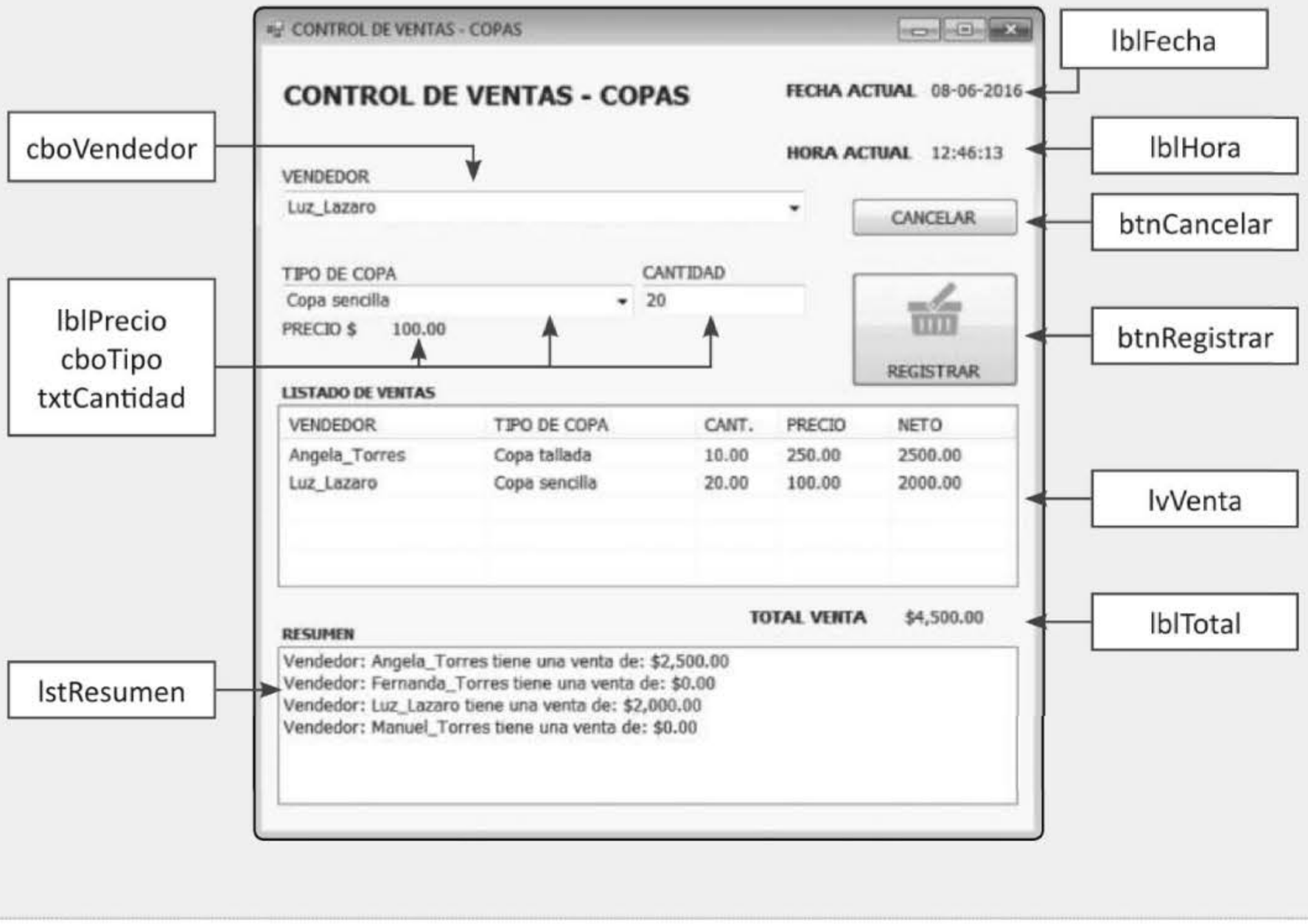
1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.
   - Seleccionar Aplicación de Windows Forms.

- Asignar el nombre pjVentaCopas, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmVentaCopa.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).



4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmVentaCopa CONTROL DE VENTA
Label1	Text	CONTROL DE VENTAS - COPAS
Label2	Text	FECHA ACTUAL
Label3	Text	HORA ACTUAL
Label4	(Name) Text	lblFecha lblFecha
Label5	(Name) Text	lblHora lblHora
Label6	Text	VENDEDOR
Label7	Text	TIPO DE COPA
Label8	Text	CANTIDAD
Label9	Text	PRECIO $
Label10	Text	lblPrecio lblPrecio
Label11	Text	LISTADO DE VENTAS
Label12	Text	TOTAL VENTA
Label13	(Name) Text	lblTotal lblTotal
Label14	Text	RESUMEN
ComboBox1	(Name)	cboVendedor
ComboBox2	(Name)	cboTipo
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
ListBox1	(Name)	lstResumen
ListView1	(Name) GridLine Columns View	lvVenta True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details
TextBox1	(Name)	txtCantidad

Código fuente:

```
Public Class frmVentaCopa
    'Definición de variables GLOBALES
    Dim precio As Double
    Dim neto As Double
    Dim vendedor, tipoCopa As String
    Dim cantidad As Integer
    Dim valida As Boolean
    Private Sub frmVentaCopa_Load(…) Handles MyBase.Load
        Call llenaVendedor()
        Call LlenaTipoCopa()
        Call muestraFecha()
        Call muestraHora()
    End Sub
    Private Sub btnRegistrar_Click(…) Handles btnRegistrar.Click
        Call validaControles()

        If valida = True Then
            Call capturaDatos()
            Call asignaPrecio()
            Call calculaNeto()
            Call imprimir()
            Call resumen()
            Call limpiarControles()
        End If
    End Sub
     Private Sub cboTipo_SelectedIndexChanged(…) Handles cboTipo.Selec-
tedIndexChanged
        Call asignaPrecio()
    End Sub
    Private Sub btnCancelar_Click(…) Handles btnCancelar.Click
        Call limpiarControles()
    End Sub

    'SUBPROCEDIMIENTOS SIN PARAMETROS
    'Metodo que llena los nombres de los vendedores al cuadro combinado
    Sub llenaVendedor()
        cboVendedor.Text = "(Seleccione vendedor)"
        For Each v In System.Enum.GetNames(GetType(Vendedores))
            cboVendedor.Items.Add(v.ToString)
        Next
    End Sub

    'Metodo que llena los tipos de copas en el cuadro combinado
    Sub LlenaTipoCopa()
        cboTipo.Text = "(Seleccione un producto)"
        With cboTipo.Items
            .Add("Copa sencilla")
            .Add("Copa tallada")
        End With
    End Sub
```

```
'Metodo que muestra la fecha actual
Sub muestraFecha()
    lblFecha.Text = DateString
End Sub

'Metodo que muestra la hora actual
Sub muestraHora()
    lblHora.Text = TimeString
End Sub

'Metodo que asigna el precio segun el tipo de copa
Sub asignaPrecio()
    If cboTipo.Text = "Copa sencilla" Then
        precio = 100
    Else
        precio = 250
    End If
    lblPrecio.Text = precio.ToString("0.00")
End Sub

'Metodo que captura los datos ingresados
Sub capturaDatos()
    vendedor = cboVendedor.Text
    tipoCopa = cboTipo.SelectedItem
    cantidad = Integer.Parse(txtCantidad.Text)
End Sub

'Metodo para calcular el neto
Sub calculaNeto()
    neto = cantidad * precio
End Sub

'Metodo que permite imprimir los resultados
Sub imprimir()
    Dim fila As New ListViewItem(vendedor)
    fila.SubItems.Add(tipoCopa)
    fila.SubItems.Add(cantidad.ToString("0.00"))
    fila.SubItems.Add(precio.ToString("0.00"))
    fila.SubItems.Add(neto.ToString("0.00"))
    lvVenta.Items.Add(fila)
    Call calculaTotalVenta()
    Call resumen()
End Sub

'Metodo que calcula el acumulado de netos
Sub calculaTotalVenta()
    Dim total As Double
    For i = 0 To lvVenta.Items.Count - 1
      total += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    Next
    lblTotal.Text = total.ToString("C")
End Sub
```

```
'Metodo que muestra el resumen de la venta
Sub resumen()
    lstResumen.Items.Clear()
    Dim totalV1 = 0, totalV2 = 0, totalV3 = 0, totalV4 = 0
For i = 0 To lvVenta.Items.Count - 1
If lvVenta.Items.Item(i).SubItems(0).Text=Vendedores.Angela_Torres.ToString Then
    totalV1 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    ElseIf lvVenta.Items.Item(i).SubItems(0).Text =
                                        Vendedores.Fernanda_Torres.ToString Then
    totalV2 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    ElseIf lvVenta.Items.Item(i).SubItems(0).Text =
                                            Vendedores.Luz_Lazaro.ToString Then
    totalV3 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    ElseIf lvVenta.Items.Item(i).SubItems(0).Text =
                                          Vendedores.Manuel_Torres.ToString Then
    totalV4 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
 End If
Next
   lstResumen.Items.Add("Vendedor: " & Vendedores.Angela_Torres.ToString &
                        " tiene una venta de: " & totalV1.ToString("C"))
  lstResumen.Items.Add("Vendedor: " & Vendedores.Fernanda_Torres.ToString &
                        " tiene una venta de: " & totalV2.ToString("C"))
   lstResumen.Items.Add("Vendedor: " & Vendedores.Luz_Lazaro.ToString &
                        " tiene una venta de: " & totalV3.ToString("C"))
  lstResumen.Items.Add("Vendedor: " & Vendedores.Manuel_Torres.ToString &
                        " tiene una venta de: " & totalV4.ToString("C"))
End Sub

'Metodo que valida los valores ingresados en el controles
Sub validaControles()
    If cboVendedor.SelectedIndex = -1 Then
      cboVendedor.Focus()
     MessageBox.Show("Debe seleccionar un vendedor de la lista", "Sistema",
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
    ElseIf cboTipo.SelectedIndex = -1 Then
      cboTipo.Focus()
    MessageBox.Show("Debe seleccionar un tipo de la lista", "Sistema",
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
    ElseIf String.IsNullOrWhiteSpace(txtCantidad.Text) Or
           Not IsNumeric(txtCantidad.Text) Then
      txtCantidad.Clear()
      txtCantidad.Focus()
      MessageBox.Show("Debe ingresar un valor valida para la cantidad","Sistema",
                        MessageBoxButtons.OK, MessageBoxIcon.Error)
     Else
         valida = True
     End If
End Sub

'Metodo que permite limpiar todos los controles
Sub limpiarControles()
    cboVendedor.SelectedIndex = -1
```

```
        cboTipo.SelectedIndex = -1
        cboVendedor.Text = "(Seleccione vendedor)"
        cboTipo.Text = "(Seleccione un producto)"
        txtCantidad.Clear()
        cboVendedor.Focus()
    End Sub
End Class

'Definición de la enumeración de vendedores
Enum Vendedores As Integer
    Fernanda_Torres
    Angela_Torres
    Manuel_Torres
    Luz_Lazaro
End Enum
```

A continuación, la explicación de algunos procesos implementados en el codigo fuente:

Iniciar la aplicación declarando las variables globales como neto, que serán la encargadas de almacenar el cálculo del neto mediante un subprocedimiento y, asimismo, mostrarlo dentro del subprocedimiento Imprimir. Las variables Vendedor, tipoCopa y Cantidad son valores capturados mediante un subprocedimiento que serán impresos mediante el subprocedimiento Imprimir. Finalmente, la variable Válida tiene la misión de devolver el valor True para determinar que todos los controles están correctamente llenos.

En el evento load se invocan a los subprocedimientos que llevan los cuadros combinados como llenaVendedor y llenaTipoCopa, asimismo, para mostrar la fecha y la hora con los subprocedimientos, muestraFecha y muestraHora respectivamente.

En el botón **Registrar** se inicia invocando el subprocedimiento validaControles, el cual emite un valor para la variable global válida si esta es True, entonces la aplicación deberá capturar los datos, asignar un precio al producto seleccionado, calcular el monto neto, imprimir los resultados en el control ListView, mostrar los montos acumulados en el control ListBox y limpiar los controles con los subprocedimientos respectivos.

Al seleccionar un determinado tipo de copa se debe mostrar el precio del mismo, para esto hacer doble clic sobre el cuadro combinado cboTipo, el cual accede al evento SelectionIndexChanged y dentro solo invocamos el subprocedimiento asignaPrecio para mostrar el precio en el control lblPrecio. Mientras tanto, el botón **Cancelar** invoca al subprocedimiento limpiarControles.

La implementación del subprocedimiento llenaVendedor inicia colocando directamente el texto Seleccione vendedor para que al iniciar la aplicación o limpiar los controles siempre muestre dicho texto. Aquí se debe tener en cuenta que para llenar los nombres de los vendedores se debe usar una enumeración y esta debe implementarse al final de la clase con el siguiente codigo:

```
Enum Vendedores As Integer
    Fernanda_Torres
    Angela_Torres
    Manuel_Torres
    Luz_Lazaro
End Enum
```

Donde Vendedores define el nombre de la enumeración y los nombres de los vendedores en forma consecutiva representan el orden de aparición en el control combobox. Asimismo, para recorrer por todos los nombres de los vendedores, se usa la estructura repetitiva For Each, tal como se muestra en el siguiente código:

```
For Each v In System.Enum.GetNames(GetType(Vendedores))
    cboVendedor.Items.Add(v.ToString)
Next
```

La variable V se encarga de representar todos los valores de la enumeración y que para acceder a los nombres se debe usar el método GetNames de la enumeración Vendedores, en este caso se usa la función GetType para identificar qué es una enumeración. Luego, se envía todos los valores del recorrido de la enumeración al cuadro combinado usando el método Add, colocando la variable V y, obligatoriamente, el método ToString para su obtención.

En el subprocedimiento LlenaTipoCopa se usa la sentencia With para suprimir el código repetido. A continuación, una comparación entre otro posible código para la misma actividad:

```
With cboTipo.Items
     .Add("Copa sencilla")
     .Add("Copa tallada")
End With
```

```
cboTipo.Items.Add("Copa sencilla")
cboTipo.Items.Add("Copa tallada")
```

En el subprocedimiento asignaPrecio se compara el valor seleccionado en el cuadro combinado cboTipo y, de acuerdo a la selección, se asigna a la variable global Precio

un monto. Asimismo, se imprime en el control lblPrecio, esto servirá para mostrar el precio al momento de seleccionar un producto desde el cuadro combinado del tipo de copa.

El subprocedimiento calculaNeto determina el monto neto desde el producto entre la cantidad y el precio capturados en el subprocedimiento capturaDatos. El subprocedimiento Imprimir se encarga de enviar toda la información ingresada y calculada en el control ListView; aquí se debe considerar que muchas de estas variables fueron declaradas como global justamente para imprimir su resultado en este subprocedimiento. Hay que tener en cuenta al momento de imprimir los valores también se debe mostrar el total neto, para esto se invoca al subprocedimiento calculaTotalVenta; de la misma manera, se invoca al subprocedimiento resumen para mostrar la información de los totales por vendedores en el control ListBox.

El subprocedimiento calcularTotalVenta tiene la misión de recorrer por todos los montos netos que se encuentra en el control ListView, para esto se usa la estructura For desde 0 hasta la ultima fila llena del control (lvVenta.Items.Count-1), el cual cuenta el total de filas llenas y se le resta uno por que la posición inicial de las filas inicia en cero. Y, para obtener los valores desde el control ListView, se usa lvVenta.Items.Item(i).SubItems(4).Text donde Item(i) representa a todas las filas y SubItems(4), a la quinta columna del control, el cual representa a los netos. Finalmente, se imprime en el mismo subprocedimiento el total en el control lblTotal.

El subprocedimiento Resumen tiene la misión de calcular el monto acumulado por cada vendedor, para se debe recorrer por todas las filas del control ListView e identificarlos por el nombre del vendedor. Hay que tener en cuenta que los nombres see encuentran en la columna cero y que los montos netos están en la columna cuatro del control ListView. Las variables totalV1, totalV2, totalV3 y totalV4 acumularán los montos según los vendedores, para lograr identificar el vendedor usamos la sentencia If.

Finalmente, el subprocedimiento validaControles evalúa control por control para determinar si el usuario seleccionó correctamente los valores y si ingresó un valor correcto en la cantidad. Para determinar si el usuario ha seleccionado un elemento de los cuadros combinados se usala sentencia SelectedIndex = –1, donde menos uno representa a un valor vacío no válido y, en el caso de la cantidad, se evalúa, en primer lugar, que la caja no se encuentre vacía o se haya ingresado un valor nulo con la sentencia String.IsNullOrWhiteSpace; asimismo se determina si solo se ingresó un valor numérico con la sentencia IsNumeric. Una vez que todos los controles fueron evaluados la cláusula Else envía un valor True para la variable Válida, indicando que todos los controles se encuentran correctamente seleccionados.

Caso desarrollado 2 Subprocedimiento con parámetros - Proforma de autos

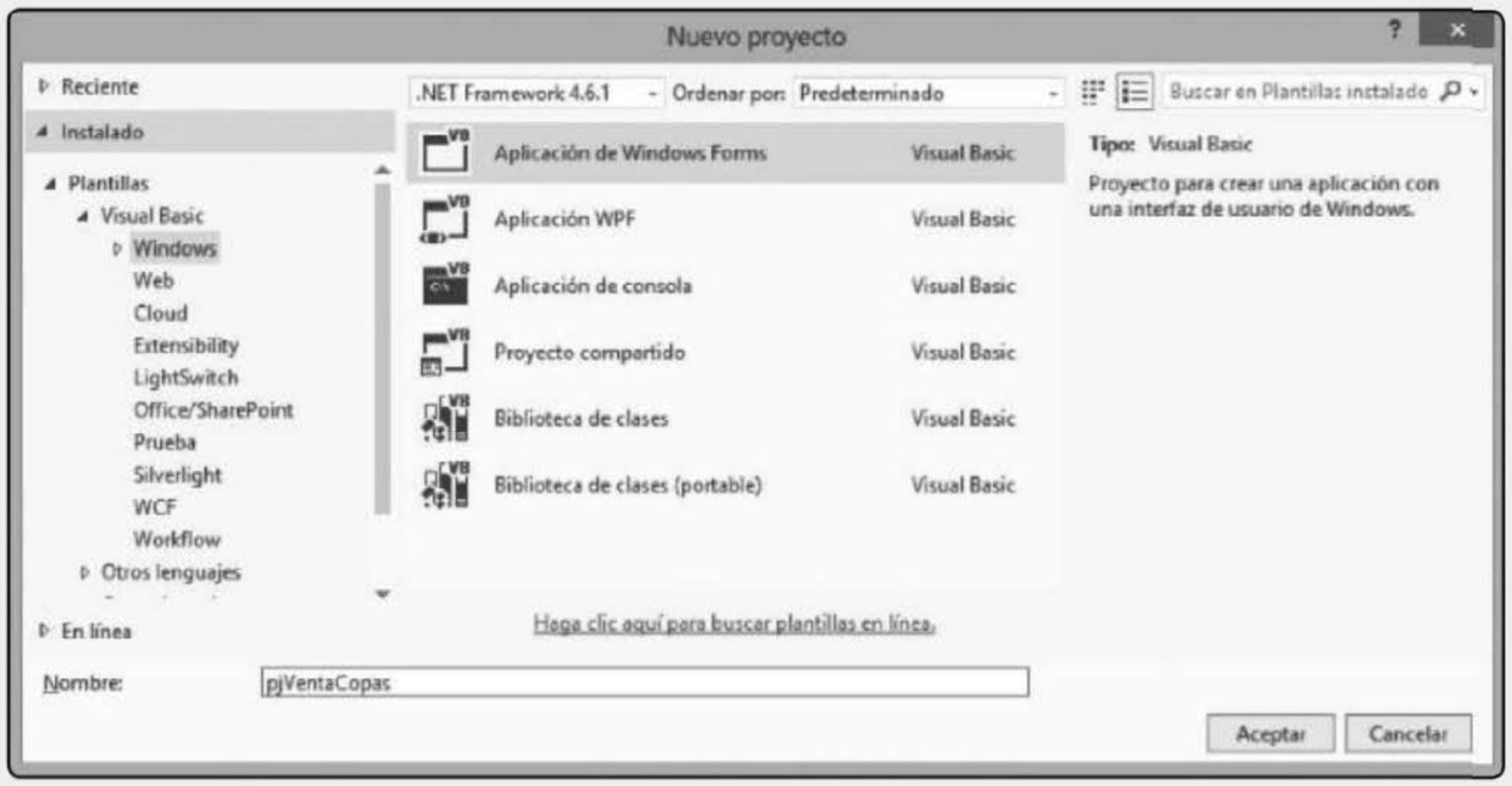
<table>
<tr><td>Objetivo</td><td>Tener el control de las proformas de servicios.</td></tr>
<tr><td>Referencias</td><td>Usar Subprocedimientos con parámetros.</td></tr>
<tr><td>Descripcion del caso</td><td>Una fábrica de autos y camiones cuenta con tres tipos de talleres, los cuales ofrecen servicios a sus clientes en fabricación de motores, carrocerías y montaje de autos. En el taller de motores se tardan cinco horas en fabricar un motor de un auto y diez horas el de un camión; en el taller de carrocería se tardan 2 horas en fabricar una carrocería de auto y 5 horas en la carrocería de un camión. Finalmente, en el taller de montaje se invierte 12 horas en montar un auto y 72 en montar un camión. Los montos por cada servicio de muestra en la siguiente tabla:
<table>
<tr><th>Tipo de servicio</th><th>Auto</th><th>Camión</th></tr>
<tr><td>Fabricación de motores</td><td>$500.00</td><td>$1000.00</td></tr>
<tr><td>Fabricación de carrocerías</td><td>$1200.00</td><td>$3500.00</td></tr>
<tr><td>Montaje</td><td>$350.00</td><td>$750.00</td></tr>
</table>
Implementar una aplicación que permita registrar los servicios que solicite el cliente, asimismo la aplicación debe mostrar todos los servicios en una lista.</td></tr>
<tr><td>Tener en cuenta</td><td>• Use subprocedimientos con parámetros.
• Al iniciar la aplicación se debe mostrar la fecha de manera automática, el numero de proforma autogenerado y la hora digitalizada. Asimismo, deberá estar precargados los nombres de los técnicos que registrarán las proformas, de la misma manera para los tipos de servicios y los tipos de auto, tal como se muestra en la siguiente lista:
<table>
<tr><th>Listado de técnicos</th><th>Código</th></tr>
<tr><td>Rocio Oviedo Cárdenas</td><td>EMP0001</td></tr>
<tr><td>Guadalupe Acosta Ferrer</td><td>EMP0002</td></tr>
<tr><td>Geidi Rengifo Reategui</td><td>EMP0003</td></tr>
<tr><td>Saray Pacheco Sandoval</td><td>EMP0004</td></tr>
</table>
<table>
<tr><th>Listado de tipo de auto</th></tr>
<tr><td>Automovil</td></tr>
<tr><td>Camión</td></tr>
</table></td></tr>
</table>

	• Al seleccionar un técnico automáticamente se debe visualizar su código. • Para seleccionar la cantidad de servicios se debe usar el control NumericUpDown. • Implementar un botón de registro, el cual permita enviar la información a un control ListView que muestre el tipo de servicio, costo, cantidad e importe. Asimismo, mostrar el subtotal (monto acumulado de importes), descuento y neto. • El subtotal resulta del producto entre el costo por tipo y la cantidad de minutos registrados. • Por cada registro, se debe mostrar el importe a pagar, el cual resulta del acumulado del monto subtotal. • Finalmente, mediante un cuadro combinado(Combobox) deberá seleccionar un tipo de llamada para mostrar los montos acumulados según el tipo de llamada.
Formulario propuesto	

Solución:

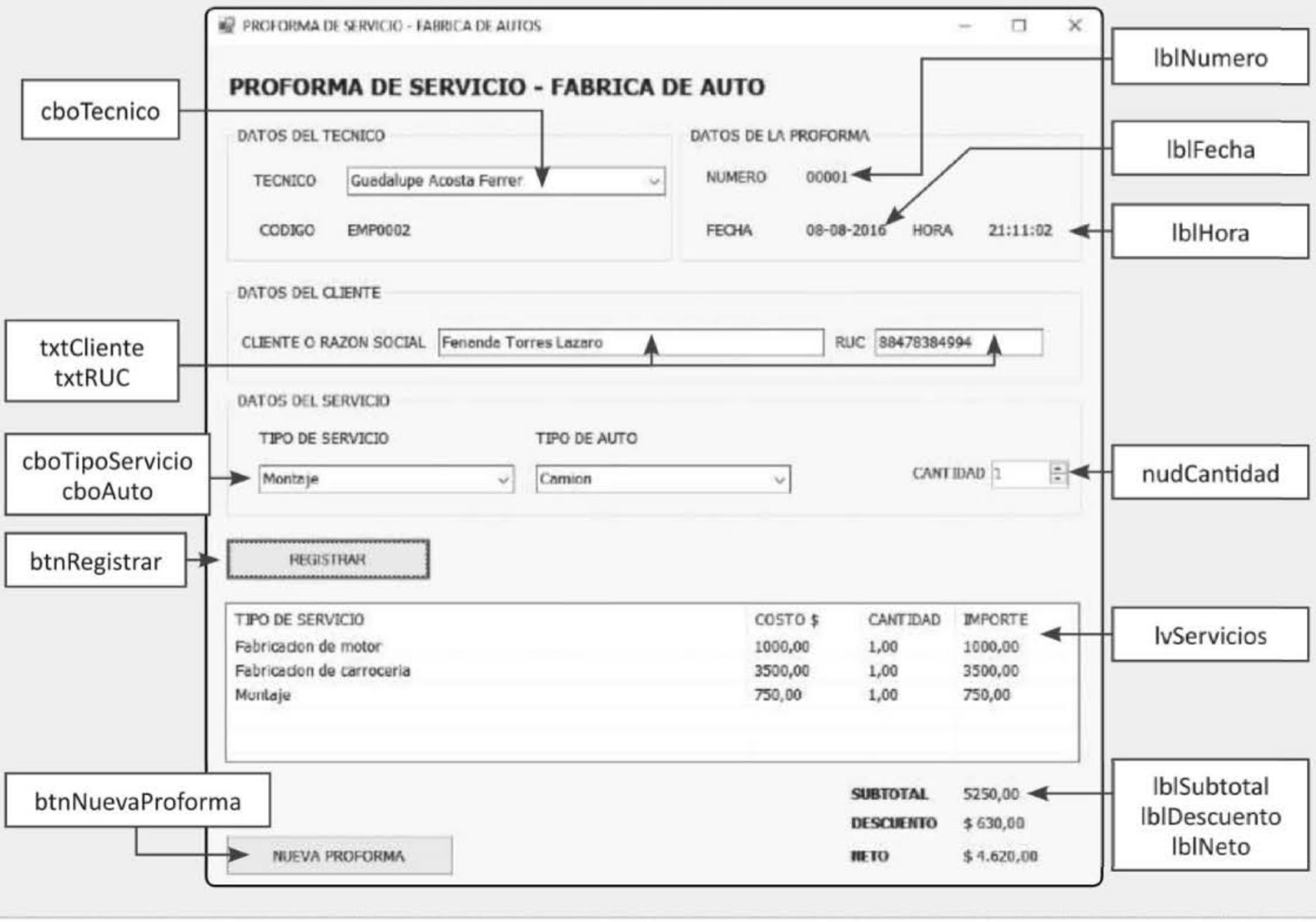
1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.
   - Seleccionar Aplicación de Windows Forms.

- Asignar el nombre pjVentaCopas, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmFabricaAutos.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).



4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmFabricaAutos PROFORMA DE SERVICIO – FABRICA DE AUTO
Label1	Text	PROFORMA DE SERVICIO - FABRICA DE AUTO
GroupBox1	Text	DATOS DEL TECNICO
GroupBox2	Text	DATOS DE LA PROFORMA
GroupBox3	Text	DATOS DEL CLIENTE
GroupBox4	Text	DATOS DEL SERVICIO
Label3	Text	TECNICO
Label4	Text	CODIGO
Label5	Text	NUMERO
Label6	Text	FECHA
Label7	Text	HORA
Label8	Text	CLIENTE O RAZON SOCIAL
Label9	Text	RUC
Label10	Text	TIPO DE SERVICIO
Label11	Text	TIPO DE AUTO
Label12	Text	CANTIDAD
Label13	(Name) Text	lblCodigo lblCodigo
Label14	(Name) Text	lblNumero lblNumero
Label15	(Name) Text	lblFecha lblFecha
Label16	(Name) Text	lblHora lblHora
Label17	Text	SUBTOTAL
Label18	(Name) Text	lblSubtotal lblSubtotal
Label19	Text	DESCUENTO
Label20	(Name) Text	lblDescuento lblDescuento
Label21	Text	NETO

Control	Propiedad	Valor
Label22	(Name) Text	lblNeto lblNeto
ComboBox1	(Name)	cboTecnico
ComboBox2	(Name)	cboTipoServicio
ComboBox3	(Name)	cboAuto
NumericUpDown	(Name)	nudCantidad
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnNuevaProforma NUEVA PROFORMA
ListView1	(Name) GridLine Columns View	lvServicios True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details
TextBox1	(Name)	txtCliente
TextBox2	(Name)	txtRUC

Código fuente:

```
Public Class frmFabricaAutos
    Dim numero As Integer
    Dim costo As Double, importe As Double
    Dim valida As Boolean
    Private Sub frmFabricaAutos_Load(…) Handles MyBase.Load
        llenaTecnicos()
        asignaNumeroProforma()
        muestraFecha()
        tHora.Start()
        llenaTipoServicio()
        llenaTipoAuto()
    End Sub
    Private Sub cboTecnico_SelectedIndexChanged(…) Handles cboTecnico.
SelectedIndexChanged
        muestraCodigo(cboTecnico.Text)
    End Sub
   Private Sub tHora_Tick(sender As Object, e As EventArgs) Handles tHora.Tick
        lblHora.Text = TimeString
    End Sub
    Private Sub btnRegistrar_Click(…) Handles btnRegistrar.Click
        validacion()
```

```
        If valida = True Then
            asignaCosto(cboTipoServicio.SelectedIndex, cboAuto.SelectedIndex)
            calculaImporte(costo, nudCantidad.Value)
            imprimir()
            calculaSubtotal()
            calculaDescuento(lblSubtotal.Text)
            calculaNeto(lblSubtotal.Text, lblDescuento.Text)
        End If
    End Sub
    Private Sub lvServicios_MouseDoubleClick(…) Handles lvServicios.MouseDoubleClick
        lvServicios.SelectedItems(0).Remove()
        calculaSubtotal()
        calculaDescuento(lblSubtotal.Text)
        calculaNeto(lblSubtotal.Text, lblDescuento.Text)
    End Sub
    Private Sub btnNuevaProforma_Click(…) Handles btnNuevaProforma.Click
        limpiaControles()
        asignaNumeroProforma()
    End Sub

    'IMPLEMENTACION DE SUBPROCEDIMIENTOS SIN PARAMETROS
    'Llena nombre de Tecnicos
    Sub llenaTecnicos()
        With cboTecnico.Items
            .Add("Rocio Oviedo Cárdenas")
            .Add("Guadalupe Acosta Ferrer")
            .Add("Geidi Rengifo Reategui")
            .Add("Saray Pacheco Sandoval")
        End With
    End Sub

    'Llena tipo de autos
    Sub llenaTipoAuto()
        With cboAuto.Items
            .Add("Automovil")
            .Add("Camion")
        End With
    End Sub

    'Asigna numero de proforma
    Sub asignaNumeroProforma()
        numero += 1
        lblNumero.Text = numero.ToString("00000")
    End Sub

    'Muestra la fecha actual
    Sub muestraFecha()
        lblFecha.Text = DateString
    End Sub

    'Llena de tipos de servicio
    Sub llenaTipoServicio()
        With cboTipoServicio.Items
```

```
            .Add("Fabricacion de motor")
            .Add("Fabricacion de carroceria")
            .Add("Montaje")
        End With
    End Sub

    'Imprimir en la lista
    Sub imprimir()
        Dim fila As New ListViewItem(cboTipoServicio.Text)
        fila.SubItems.Add(costo.ToString("0.00"))
        fila.SubItems.Add(nudCantidad.Value.ToString("0.00"))
        fila.SubItems.Add(importe.ToString("0.00"))
        lvServicios.Items.Add(fila)
    End Sub

    'Calcular el subtotal
    Sub calculaSubtotal()
        Dim subtotal As Double
        For i = 0 To lvServicios.Items.Count - 1
          subtotal += Double.Parse(lvServicios.Items.Item(i).SubItems(3).Text)
        Next
        lblSubtotal.Text = subtotal.ToString("0.00")
    End Sub

    'Validacion de datos
    Sub validacion()
        If cboTecnico.SelectedIndex = -1 Then
            MessageBox.Show("Debe seleccionar un Técnico")
            cboTecnico.Focus()
        ElseIf String.IsNullOrWhiteSpace(txtCliente.Text) Then
            MessageBox.Show("Debe ingresar el nombre del cliente")
            txtCliente.Clear()
            txtCliente.Focus()
        ElseIf String.IsNullOrWhiteSpace(txtRUC.Text) Then
            MessageBox.Show("Debe ingresar el RUC del cliente")
            txtRUC.Clear()
            txtRUC.Focus()
        ElseIf cboTipoServicio.SelectedIndex = -1 Then
            MessageBox.Show("Debe seleccionar un tipo de servicio")
            cboTipoServicio.Focus()
        ElseIf cboAuto.SelectedIndex = -1 Then
            MessageBox.Show("Debe seleccionar un tipo de auto")
            cboAuto.Focus()
        Else
            valida = True
        End If
    End Sub

    Sub limpiaControles()
        cboTecnico.SelectedIndex = -1
        lblCodigo.Text = ""
        txtCliente.Clear()
```

```
        txtRUC.Clear()
        cboTipoServicio.SelectedIndex = -1
        cboAuto.SelectedIndex = -1
        nudCantidad.Value = 1
        lvServicios.Items.Clear()
        lblSubtotal.Text = "0.00"
        lblDescuento.Text = "0.00"
        lblNeto.Text = "0.00"
        cboTecnico.Focus()
    End Sub

    'IMPLEMENTACION DE SUBPROCEDIMIENTOS CON PARAMETROS
    'Muestra codigo del técnico seleccionado
    Sub muestraCodigo(ByVal tecnico As String)
        Select Case tecnico
        Case "Rocio Oviedo Cárdenas":lblCodigo.Text = "EMP" & (1).ToString("0000")
        Case "Guadalupe Acosta Ferrer":lblCodigo.Text = "EMP" & (2).ToString("0000")
        Case "Geidi Rengifo Reategui":lblCodigo.Text = "EMP" & (3).ToString("0000")
        Case "Saray Pacheco Sandoval":lblCodigo.Text = "EMP" & (4).ToString("0000")
        End Select
    End Sub

    'Determina el costo por servicio
    Sub asignaCosto(ByVal servicio As Integer, ByVal auto As Integer)
        If auto.Equals("Automovil") Then
            Select Case servicio
                Case 0 : costo = 500
                Case 1 : costo = 1200
                Case 2 : costo = 350
            End Select
        Else
            Select Case servicio
                Case 0 : costo = 1000
                Case 1 : costo = 3500
                Case 2 : costo = 750
            End Select
        End If
    End Sub

    'Determina el importe por servicio
    Sub calculaImporte(ByVal costo As Double, ByVal cantidad As Integer)
        importe = costo * cantidad
    End Sub

    'Calcular el descuento
    Sub calculaDescuento(ByVal subtotal As Double)
        lblDescuento.Text = (subtotal * 0.12).ToString("C")
    End Sub

    'Calcular en neto a pagar
    Sub calculaNeto(ByVal subtotal As Double, ByVal descuento As Double)
        lblNeto.Text = (subtotal - descuento).ToString("C")
    End Sub
End Class
```

A continuación, la explicación de algunos procesos implementados en el código fuente:

Para mostrar el reloj digitalizado en el formulario, primero se debe agregar el control Timer al formulario y asignarle un nombre. Para este caso se le asigna tHora, luego, en su propiedad Intervalo, asignar 1000, ya que representa milisegundos. Después, se debe asignar la hora actual al control que visualizará dicha hora usando la sentencia lblHora.Text = TimeString; este código debe ser colocado en el evento Tick del control tHora. Para llegar al evento solo es cuestión de hacer doble clic sobre él.

El uso de los subprocedimientos con parámetros permite enviar un valor por el parámetro para que sea usado dentro del subprocedimiento, por ejemplo, si se observa la invocación al subprocedimiento muestraCodigo(cboTecnico.Text), cboTecnico representa al valor del parámetro, es decir, el valor seleccionado desde el cuadro combinado cboTecnico será tomado como un parámetro. Es necesario tener en cuenta y estar muy atentos a la cantidad de parámetros especificados y, especialmente, al orden de envío de los valores.

Un tema muy importante en la aplicación es la validación de datos ingresados en los controles, para esto está preparado un subprocedimiento llamado Validación, el cual verifica que todos los controles contengan valores adecuados.

```
Sub validacion()
        If cboTecnico.SelectedIndex = -1 Then
            MessageBox.Show("Debe seleccionar un Técnico")
            cboTecnico.Focus()
        ElseIf String.IsNullOrWhiteSpace(txtCliente.Text) Then
            MessageBox.Show("Debe ingresar el nombre del cliente")
            txtCliente.Clear()
            txtCliente.Focus()
        ElseIf String.IsNullOrWhiteSpace(txtRUC.Text) Then
            MessageBox.Show("Debe ingresar el RUC del cliente")
            txtRUC.Clear()
            txtRUC.Focus()
        ElseIf cboTipoServicio.SelectedIndex = -1 Then
            MessageBox.Show("Debe seleccionar un tipo de servicio")
            cboTipoServicio.Focus()
        ElseIf cboAuto.SelectedIndex = -1 Then
            MessageBox.Show("Debe seleccionar un tipo de auto")
            cboAuto.Focus()
        Else
            valida = True
        End If
    End Sub
```

Para verificar que el usuario seleccione un elemento del cuadro combinado se usa la propiedad SelectedIndex= -1, donde menos uno indica el valor nulo de los elementos, ya que desde la posición cero inició el índice de los elementos contenidos. Asimismo, se debe evaluar el contenido de las cajas de texto, para esto se debe evaluar el contenido vacío y valores nulos, para esto se usa la función IsNullOrWhiteSpace y como parámetro se especifica el control a evaluar. No olvidar que la variable Válida contiene el valor True, el cual indicará que todos los controles han sido evaluados correctamente.

Una de las especificaciones de la aplicación indica que si el usuario desea eliminar un servicio desde la lista de servicios registrados, lo debe realizar haciendo doble clic en el mismo registro, para esto se debe acceder al evento MouseDoubleClick del control lvServicio; entonces, primero seleccionar el control, en el panel de propiedades seleccionar **Eventos**, en la lista seleccionar MouseDoubleClick y colocar el siguiente código `lvServicios.SelectedItems(0).Remove()`, donde cero representa la fila seleccionada y Remove es el método que permite eliminarla. Finalmente, se debe recalcular todos los montos, para esto se invoca a los subprocedimientos calculaSubtotal, calculaDescuento y calculaNeto.

En el siguiente subprocedimiento se genera y se imprime un número de proforma de forma automática:

```
Sub asignaNumeroProforma()
        numero += 1
        lblNumero.Text = numero.ToString("00000")
    End Sub
```

Aquí la variable Número debe ser global para que el aumento en uno se realice por cada invocación de subprocedimiento; luego se imprime en su control respecto con cuatro ceros a la izquierda; se debe tener en cuenta que la cantidad de ceros colocados en el método ToString serán los ceros a la izquierda del numero generados, contando desde el mismo numero.

Para calcular el subtotal se debe recorrer por toda la lista de servicios registrados, especialmente por la columna donde se muestran los montos de tipo importe.

A continuación, un código que da solución a lo propuesto.

```
Dim subtotal As Double
For i = 0 To lvServicios.Items.Count - 1
   subtotal += Double.Parse(lvServicios.Items.Item(i).SubItems(3).Text)
Next
```

Se inicia por declarar la variable Subtotal que permitirá obtener el monto acumulado de importes. Para recorrer por todas las filas de servicios registrados se usa una sentencia repetitiva For y se inicia desde la fila cero hasta la ultima posición, la cual es obtenida mediante el total de filas menos uno. Y, finalmente, para hacer referencia a los importes se usa la sentencia `Double.Parse(lvServicios.Items.Item(i).SubItems(3).Text)`, donde i representa la posición de todas las filas de la lista, y 3 representa la columna de los importes.

En la limpieza de los controles se debe considerar los siguientes aspectos:

SelectedIndex = -1	Esta propiedad con la asignación -1 ubica el cursor en un lugar vacío de los elementos del control y solo es aplicable para cuadros combinados(ComboBox).
Clear()	Método que permite limpiar el contenido de un control de tipo TextBox.
nudCantidad.Value=1	Permite inicializar el valor del control NumericUpDonw en uno.
Text = "0.00"	La asignación de 0.00 se realiza para que se muestre directamente dicho valor e indique que no hay monto a mostrar. Esto sucede en la limpieza de controles, al cancelar todo el ingreso, al registrar una nueva proforma o al iniciar la aplicación.
Focus()	Envía el cursor del mouse a un determinado control.

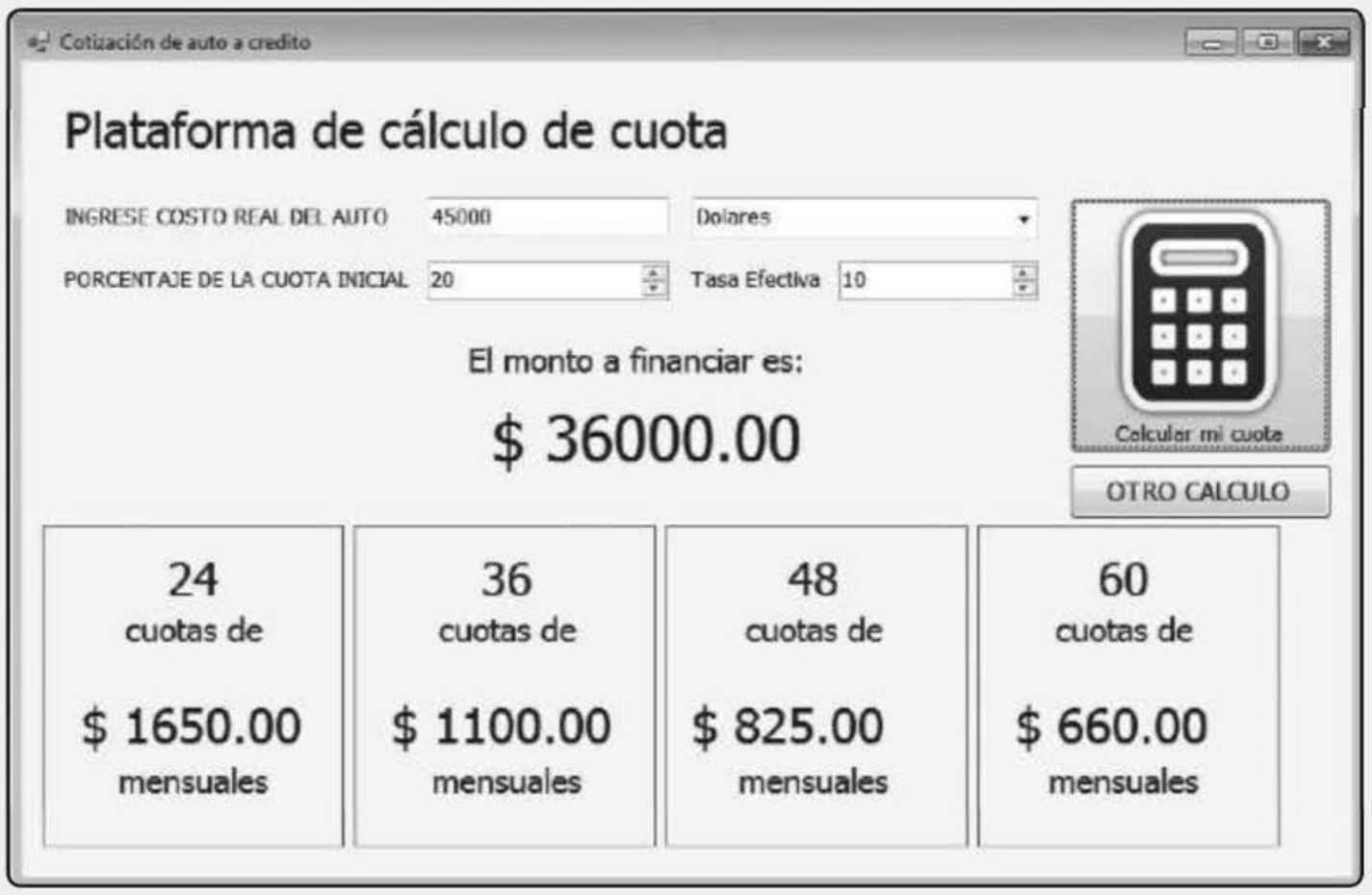
Para mostrar un numero entero con ceros a la izquierda se puede usar la sentencia (1). ToString("0000"), la cual mostrará 0001, donde el valor 1 ocupará siempre la posición de derecha a izquierda. Se debe mencionar que para realizar una comparación en una sentencia If de tipo cadena, puede usar dos formatos. Ver el siguiente caso:

```
If auto.Equals("Automovil") Then
   Select Case servicio
      Case 0 : costo = 500
      Case 1 : costo = 1200
      Case 2 : costo = 350
   End Select
End If
```

```
If auto ="Automovil" Then
   Select Case servicio
      Case 0 : costo = 500
      Case 1 : costo = 1200
      Case 2 : costo = 350
   End Select
End If
```

Finalmente, no olvidar que para imprimir en formato de moneda se puede usar el método ToString usando como parámetro el valor C, este imprimirá en formato configurado en su sistema operativo (subtotal * 0.12).ToString("C"); en este caso, por ejemplo, el resultado que se obtenga entre el subtotal y su 12 % será mostrado en formato de moneda.

Caso desarrollado 3 — Procedimiento de función sin parámetros - Control de cotización

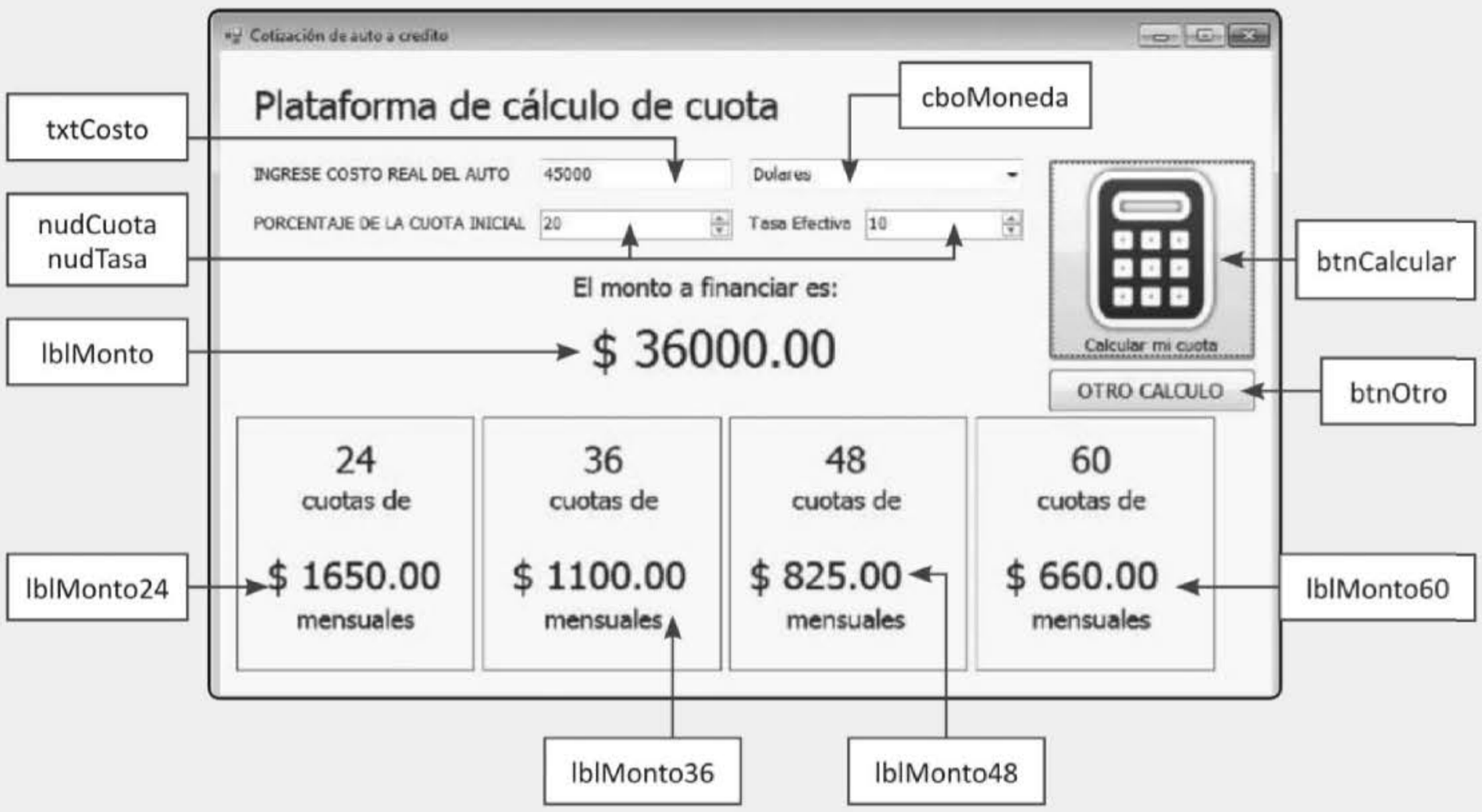
Objetivo	Tener el control de las cotizaciones para la venta de autos al crédito.
Referencias	Usar subprocedimientos con parámetros.
Descripción del caso	Una entidad bancaria ha implementado una plataforma para cotizar la venta al crédito de automóviles, para el cual un encargado del banco deberá ingresar el costo real del auto, el tipo de moneda, asignar un porcentaje para la cuota inicial, una asignación en porcentaje de la tasa efectiva y, de acuerdo a esa información, emitirá el monto que el banco debe financiar y los montos mensuales que podría realizar el cliente según la cantidad de meses preestablecidos con que cuenta la entidad.
Tener en cuenta	• Usar funciones sin parámetros. • Al iniciar la aplicación se debe mostrar precargado los tipos de monedas, como soles y dólares respectivamente. • Para seleccionar el porcentaje de la cuota inicial y la tasa efectiva usar el control NumericUpDown respectivamente. • El monto a financiar resulta del restar el monto del porcentaje de la cuota inicial al costo real del auto. • Hay que tener en cuenta que al seleccionar un tipo de moneda todos los montos expuestos en la aplicación deberán mostrar el símbolo correspondiente. • Finalmente, validar los controles de ingreso de costo y la selección del tipo de moneda; en caso hubiera algún error mostrar un mensaje usando funciones sin parámetros.
Formulario propuesto	

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto.**
   - Seleccionar la plantilla Visual Basic.
   - Seleccionar Aplicación de Windows Forms.
   - Asignar el nombre pjCotizacionAuto.

2. Cambiar el nombre del formulario.
   - Seleccione el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmCotizacion.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).



4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmCotizacion Cotización de auto a crédito
Label1	Text	Plataforma de cálculo de cuota
Label2	Text	INGRESE COSTO REAL DEL AUTO
Label3	Text	PORCENTAJE DE LA CUOTA INICIAL
Label4	Text	TASA EFECTIVA
Label5	Text	EL MONTO A FINANCIAR ES:

Label6	(Name) Text	lblMonto 0.00
Label7	Text	24
Label8	Text	cuotas de
Label9	Text	36
Label10	Text	cuotas de
Label11	Text	48
Label12	Text	cuotas de
Label13	Text	60
Label14	Text	cuotas de
Label15	Text	lblMonto24 0.00
Label16	(Name) Text	lblMonto36 0.00
Label17	(Name) Text	lblMonto48 0.00
Label18	(Name) Text	lblMonto60 0.00
ComboBox1	(Name)	cboMoneda
TextBox1	(Name)	txtCosto
NumericUpDonw1	(Name) Value	nudCuota 20
NumericUpDonw2	(Name) Value	nudTasa 10
Button1	(Name) Text	btnCalcular CALCULAR MI CUOTA
Button2	(Name) Text	btnOtro OTRO CALCULO

Código fuente:

```
Public Class frmCotizacion

    Private Sub frmCotizacion_Load(…) Handles MyBase.Load
        llenaMoneda()
        cboMoneda.SelectedIndex = 0
    End Sub
```

```
    Private Sub btnCalcular_Click(…) Handles btnCalcular.Click
        If valida() = "" Then
            Dim monto As Double = calculaMontoFinanciado()
            imprimir(monto, getMoneda())
        Else
            MessageBox.Show("El error se encuentra en " & valida())
        End If
    End Sub

    Private Sub btnOtro_Click(sender As Object, e As EventArgs) Handles btnOtro.Click
        limpiaControles()
    End Sub

    'Implementación de subProcedimientos sin parámetros
    Sub llenaMoneda()
        With cboMoneda.Items
            .Add("Soles")
            .Add("Dolares")
        End With
    End Sub

    Sub limpiaControles()
        txtCosto.Clear()
        cboMoneda.SelectedIndex = 0
        nudCuota.Value = 20
        nudTasa.Value = 10
        lblMonto.Text = "0.00"
        lblMonto24.Text = "0.00"
        lblMonto36.Text = "0.00"
        lblMonto48.Text = "0.00"
        lblMonto60.Text = "0.00"
        txtCosto.Focus()
    End Sub

    'Implementación de subProcedimientos con parámetros
    Sub imprimir(ByVal monto As Double, ByVal moneda As String)
        If moneda = "Dolares" Then
            lblMonto.Text = "$ " & monto.ToString("0.00")
            lblMonto24.Text = "$ " & calculaCuota24().ToString("0.00")
            lblMonto36.Text = "$ " & calculaCuota36().ToString("0.00")
            lblMonto48.Text = "$ " & calculaCuota48().ToString("0.00")
            lblMonto60.Text = "$ " & calculaCuota60().ToString("0.00")
        Else
            lblMonto.Text = "S/. " & monto.ToString("0.00")
            lblMonto24.Text = "S/. " & calculaCuota24().ToString("0.00")
            lblMonto36.Text = "S/. " & calculaCuota36().ToString("0.00")
            lblMonto48.Text = "S/. " & calculaCuota48().ToString("0.00")
            lblMonto60.Text = "S/. " & calculaCuota60().ToString("0.00")
        End If
    End Sub

    'Implementación de funciones sin parámetros
    Function getCosto() As Double
        Return Double.Parse(txtCosto.Text)
    End Function
    Function getMoneda() As String
        Return cboMoneda.SelectedItem
    End Function
```

```
    Function getCuota() As Integer
        Return nudCuota.Value
    End Function
    Function getTasa() As Integer
        Return nudTasa.Value
    End Function

    Function calculaMontoFinanciado()
        'Obtener los valores ingresados
        Dim costo As Double = getCosto()
        Dim cuotaInicial As Double = getCuota()

        'Calcular y retornar el monto
        Return costo - costo * (cuotaInicial / 100)
    End Function

    Function calculaCuota24() As Double
        Dim monto As Double = calculaMontoFinanciado()
        Return (monto + (monto * (getTasa() / 100))) / 24
    End Function

    Function calculaCuota36() As Double
        Dim monto As Double = calculaMontoFinanciado()
        Return (monto + (monto * (getTasa() / 100))) / 36
    End Function

    Function calculaCuota48() As Double
        Dim monto As Double = calculaMontoFinanciado()
        Return (monto + (monto * (getTasa() / 100))) / 48
    End Function

    Function calculaCuota60() As Double
        Dim monto As Double = calculaMontoFinanciado()
        Return (monto + (monto * (getTasa() / 100))) / 60
    End Function

    Function valida() As String
        If String.IsNullOrWhiteSpace(txtCosto.Text) Or
           Not IsNumeric(txtCosto.Text) Then
            txtCosto.Clear()
            txtCosto.Focus()
            Return "costo del auto"
        ElseIf cboMoneda.SelectedIndex = -1 Then
            cboMoneda.Focus()
            Return "tipo de moneda"
        Else
            Return ""
        End If
    End Function
End Class
```

A continuación, la explicación de algunos procesos implementados en el codigo fuente:

Al iniciar la aplicación el selector de tipo de moneda debe mostrar el primer elemento del cuadro, para esto se usa la sentencia cboMoneda.SelectedIndex = 0, donde el valor 0 representa al primer elemento; todo esto debe ser especificado en el evento load del formulario.

La función de validación tiene por objetivo verificar que en los controles se hayan ingresado o seleccionado los valores correctos. Pero como se sabe, toda función tiene una devolución, por tanto, esta función devolverá una porción de texto en la cual indicará dónde se ocasionó el error, tal como se muestra en el siguiente código:

```
Function valida() As String
        If String.IsNullOrWhiteSpace(txtCosto.Text) Or
           Not IsNumeric(txtCosto.Text) Then
            txtCosto.Clear()
            txtCosto.Focus()
            Return "costo del auto"
        ElseIf cboMoneda.SelectedIndex = -1 Then
            cboMoneda.Focus()
            Return "tipo de moneda"
        Else
            Return ""
        End If
End Function
```

Hay un valor de retorno especificado por cada evaluación y como valor por defecto devuelve una cadena vacía, la cual se verifica al momento de mostrar los resultados desde el botón **Calcular**, como se muestra en el siguiente código:

```
If valida() = "" Then
   Dim monto As Double = calculaMontoFinanciado()
   imprimir(monto, getMoneda())
   Else
   MessageBox.Show("El error se encuentra en " & valida())
End If
```

Si la función válida retorna a un valor vacío, indica que todos los controles evaluados son correctos y se procederá a mostrar los resultados que la aplicación necesita. Pero si es diferente de vacío implica que viene una cadena que será mostrada desde el MessageBox, unido al texto El error se encuentran en...

Se han implementado funciones que permiten capturar los valores que fueron ingresados o seleccionados desde los controles del formulario. Ver la siguiente implementación:

```
Function getCosto() As Double
   Return Double.Parse(txtCosto.Text)
End Function
```

La función getCosto permite obtener el valor registrado en el control txtCosto que podrá ser invocado en cualquier momento de la aplicación; hay que tener en cuenta que es una buena práctica realizar este tipo de funciones, especialmente cuando se trabaja con aplicaciones orientadas a objetos. Solo se debe tener en cuenta que su forma de invocación es distinta al de los subprocedimientos; las funciones tienen las características de ser invocados dentro de una expresión o sentencia.

Caso desarrollado 4 — Procedimiento de función con parámetros - Fabricación de mesas

Objetivo

Tener el control de los pedidos para la fabricación de mesas.

Referencias

Usar funciones con parámetros.

Descripción del caso

Una empresa de fabricación de mesas necesita implementar una aplicación que permita estimar los tiempos y costos para la fabricación de cierta cantidad de mesas solicitadas por sus clientes, para lo cual cuenta con los siguientes criterios en precios y tiempos estimados:

Modelo	Tiempo corte(h)	Tiempo montaje(h)	Tiempo pintado(h)	Costo unitario($)
Tipo A pintado	1	8	4	$70.00
Tipo A sin pintar	1	8	0	$60.00
Tipo B pintado	2	4	3	$50.00
Tipo B sin pintar	2	4	0	$40.00
Tipo C pintado	4	2	4	$20.00
Tipo C sin pintar	4	2	0	$150.00

La aplicación debe mostrar el numero telefónico del empleado encargado de registrar. A continuación, asignación:

Empleado	Teléfono
Juan Miranda	580-3934
Carlos Ramos	580-3935
Jorge García	580-3936
Miguel Robles	580-3937

Al seleccionar un tipo de mesa y la cantidad respectiva la aplicación nos debe mostrar en una lista los montos subtotales por cada pedido, asimismo deberá determinar el tiempo estimado por cada tipo de mesa.

Tener en cuenta	• Usar funciones con parámetros. • Al iniciar la aplicación se debe mostrar precargados los nombres de los empleados y tipos de mesa respectivamente. • Al seleccionar un empleado desde el control cuadro combinado se debe mostrar el numero telefónico asignado a cada empleado. • Usar el control CheckBox para indicar si la mesa será pintada o no. • El botón **Registrar** envía la información del pedido, asimismo muestra el precio asignado y el subtotal calculado. Con el mismo botón se debe mostrar el detalle de la fabricación en el cual describe el detalle de las horas calculadas para la fabricación de una determina mesa. Finalmente, mostrar cuál es el neto a pagar por todos los pedidos realizados. • El botón **Eliminar** permite eliminar un elemento de la lista de pedido asi mismo se debe actualizar el monto neto a pagar y el detalle de la fabricación. • El botón **Limpiar** permite eliminar todos los elementos de la lista de pedido, asimismo deberá limpiar los datos mostrados en el detalle de la fabricación. • Finalmente, validar los controles de selección de empleado, tipo de mesa y cantidad; en caso hubiera algún error mostrar un mensaje usando funciones sin parámetros.
Formulario propuesto	

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.

- Seleccionar la plantilla Visual Basic.
- Seleccionar Aplicación de Windows Forms.
- Asignar el nombre pjFabricaMesas.

2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmFabricacion.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).

4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmFabricacion Estimación de tiempo y costo
Label1	Text	ESTIMACION DE TIEMPO Y COSTO – FABRICACION DE MESAS
Label2	Text	EMPLEADO
Label3	Text	TELEFONO
Label4	(Name) Text	lblTelefono lblTelefono
Label5	Text	TIPO DE MESA

Control	Propiedad	Valor
Label6	Text	CANTIDAD
Label7	Text	NETO A PAGAR
Label8	Text	lblNeto lblNeto
GroupBox1	Text	Datos del empleado registrador
GroupBox2	Text	Datos de la fabricación
GroupBox3	Text	Detalle de la fabricación
Combobox1	(Name)	cboEmpleado
Combobox2	(Name)	cboTipo
CheckBox1	(Name) Text	chkPintado PINTADO
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnLimpiar LIMPIAR
Button3	(Name) Text	btnEliminar ELIMINAR
ListView1	(Name) GridLine Columns View	lvPedido True Agregue 5 columnas y defina sus cabeceras como se muestra en el modelo. Details
Listbox1	(Name)	lstDetalle
Listbox2	(Name)	lstResumen

Código fuente:

```
Public Class frmFabricacion

    Private Sub frmFabricacion_Load(…) Handles MyBase.Load
        llenaEmpleado()
        llenaTipoMesa()
    End Sub

    Private Sub btnRegistrar_Click(…) Handles btnRegistrar.Click
        If valida() = "" Then
            Dim mesa As String = getTipoMesa()
            Dim cantidad As Integer = getCantidad()
            Dim pintado As String = getPintado()
            Dim precio As Double = determinaPrecio(mesa, pintado)
```

```
        Dim subtotal As Double = calculaSubtotal(cantidad, precio)

        imprimir(mesa, pintado, cantidad, precio, subtotal)
        lblNeto.Text = calculaNeto().ToString("C")

        imprimirDetalle(mesa, pintado, cantidad)

        With lstResumen.Items
         .Add("Mesa: " & mesa &
              " - Pintada: " & pintado &
          " - Total tiempo de Fabric.:" & calculaTiempo(mesa,pintado,cantidad))
        End With
    Else
        MessageBox.Show("El error se encuentra en " & valida())
    End If
End Sub

Private Sub cboEmpleado_SelectedIndexChanged(…)
    lblTelefono.Text = muestraFono(cboEmpleado.SelectedItem)
End Sub

Private Sub btnEliminar_Click(…) Handles btnEliminar.Click
    Dim posicion As Integer = lvPedido.SelectedItems(0).Index
    lvPedido.Items.RemoveAt(posicion)
    lstResumen.Items.RemoveAt(posicion)
    lblNeto.Text = calculaNeto().ToString("C")
    lstDetalle.Items.Clear()
End Sub

Private Sub btnLimpiar_Click(…) Handles btnLimpiar.Click
    lvPedido.Items.Clear()
    limpiarControles()
    lblNeto.Text = "0.00"
End Sub

'*** SUBPROCEDIMIENTOS SIN PARAMETROS ****
'Subprocedimiento que llena los nombres de los empleados
Sub llenaEmpleado()
    With cboEmpleado.Items
        .Add("Juan Miranda")
        .Add("Carlos Ramos")
        .Add("Jorge Garcia")
        .Add("Miguel Robles")
    End With
End Sub

'Subprocedimiento que llena los tipos de mesa
Sub llenaTipoMesa()
    With cboTipo.Items
        .Add("Tipo A")
        .Add("Tipo B")
        .Add("Tipo C")
    End With
End Sub
```

```
    'Subprocedimiento que limpiar los controles
    Sub limpiarControles()
        lstDetalle.Items.Clear()
        lstResumen.Items.Clear()
    End Sub

    '*** SUBPROCEDIMIENTOS CON PARAMETROS ****
    'Subprocedimiento que imprime valores en la lista
    Sub imprimir(mesa As String, pintado As String, cantidad As Integer,
                 precio As Double, subtotal As Double)
        Dim fila As New ListViewItem(mesa)
        fila.SubItems.Add(pintado)
        fila.SubItems.Add(cantidad.ToString("0.00"))
        fila.SubItems.Add(precio.ToString("0.00"))
        fila.SubItems.Add(subtotal.ToString("0.00"))
        lvPedido.Items.Add(fila)
    End Sub

    'Subprocedimiento que muestra los detalles del pedido
    Sub imprimirDetalle(ByVal mesa As String, ByVal pintado As String,
                        ByVal cantidad As Integer)
        lstDetalle.Items.Clear()
        If pintado = "SI" Then
         With lstDetalle.Items
           .Add("Mesa: " & mesa)
           .Add("Pintada: " & pintado)
           .Add("Cantidad: " & cantidad)
           .Add("-------------------------------------------------------")
           .Add("Tiempo de corte:" & calculaTiempoCorte(mesa) * cantidad & " Horas")
           .Add("Tiempo de montaje:" & calculaTiempoMontaje(mesa)*cantidad & " Horas")
           .Add("Tiempo de pintado:" & calculaTiempoPintura(mesa)*cantidad & " Horas")
         End With
        Else
         With lstDetalle.Items
           .Add("Mesa: " & mesa)
           .Add("Pintada: " & pintado)
           .Add("Cantidad: " & cantidad)
           .Add("-------------------------------------------------------")
           .Add("Tiempo de corte: " & calculaTiempoCorte(mesa) * cantidad & " Horas")
           .Add("Tiempo de montaje:" & calculaTiempoMontaje(mesa)*cantidad & " Horas")
           .Add("Tiempo de pintado: " & 0 & " Horas")
         End With
        End If
    End Sub

    '**** FUNCIONES SIN PARAMETROS ****
    'Función que captura el nombre del empleado
    Function getEmpleado() As String
        Return cboEmpleado.SelectedItem
    End Function

    'Función que captura el tipo de mesa seleccionada
    Function getTipoMesa() As String
        Return cboTipo.SelectedItem
    End Function
```

```
    'Subprocedimiento que limpiar los controles
    Sub limpiarControles()
        lstDetalle.Items.Clear()
        lstResumen.Items.Clear()
    End Sub

    '*** SUBPROCEDIMIENTOS CON PARAMETROS ****
    'Subprocedimiento que imprime valores en la lista
    Sub imprimir(mesa As String, pintado As String, cantidad As Integer,
                 precio As Double, subtotal As Double)
        Dim fila As New ListViewItem(mesa)
        fila.SubItems.Add(pintado)
        fila.SubItems.Add(cantidad.ToString("0.00"))
        fila.SubItems.Add(precio.ToString("0.00"))
        fila.SubItems.Add(subtotal.ToString("0.00"))
        lvPedido.Items.Add(fila)
    End Sub

    'Subprocedimiento que muestra los detalles del pedido
    Sub imprimirDetalle(ByVal mesa As String, ByVal pintado As String,
                        ByVal cantidad As Integer)
        lstDetalle.Items.Clear()
        If pintado = "SI" Then
         With lstDetalle.Items
           .Add("Mesa: " & mesa)
           .Add("Pintada: " & pintado)
           .Add("Cantidad: " & cantidad)
           .Add("-------------------------------------------------------")
           .Add("Tiempo de corte:" & calculaTiempoCorte(mesa) * cantidad & " Horas")
           .Add("Tiempo de montaje:" & calculaTiempoMontaje(mesa)*cantidad & " Horas")
           .Add("Tiempo de pintado:" & calculaTiempoPintura(mesa)*cantidad & " Horas")
         End With
        Else
         With lstDetalle.Items
           .Add("Mesa: " & mesa)
           .Add("Pintada: " & pintado)
           .Add("Cantidad: " & cantidad)
           .Add("-------------------------------------------------------")
           .Add("Tiempo de corte: " & calculaTiempoCorte(mesa) * cantidad & " Horas")
           .Add("Tiempo de montaje:" & calculaTiempoMontaje(mesa)*cantidad & " Horas")
           .Add("Tiempo de pintado: " & 0 & " Horas")
         End With
        End If
    End Sub

    '**** FUNCIONES SIN PARAMETROS ****
    'Función que captura el nombre del empleado
    Function getEmpleado() As String
        Return cboEmpleado.SelectedItem
    End Function

    'Función que captura el tipo de mesa seleccionada
    Function getTipoMesa() As String
        Return cboTipo.SelectedItem
    End Function
```

```
'Función que captura la cantidad ingresada
Function getCantidad() As Integer
    Return Integer.Parse(txtCantidad.Text)
End Function

'Función que determina el pintado de la mesa
Function getPintado() As String
    If chkPintado.Checked Then
        Return "SI"
    Else
        Return "NO"
    End If
End Function

'Función que determina el monto neto a pagar
Function calculaNeto() As Double
    Dim neto As Double
    For i = 0 To lvPedido.Items.Count - 1
       neto += Double.Parse(lvPedido.Items.Item(i).SubItems(4).Text)
    Next
    Return neto
End Function

'Función que valida el ingreso de valores en el formulario
Function valida() As String
    If cboEmpleado.SelectedIndex = -1 Then
        cboEmpleado.Focus()
        Return "nombre del empleado"
    ElseIf cboTipo.SelectedIndex = -1 Then
        cboTipo.Focus()
        Return "tipo de mesa"
    ElseIf String.IsNullOrWhiteSpace(txtCantidad.Text) Or
           Not IsNumeric(txtCantidad.Text) Then
        txtCantidad.Clear()
        txtCantidad.Focus()
        Return "cantidad"
    Else
        Return ""
    End If
End Function

'***** FUNCIONES CON PARAMETROS *****
'Función que muestra el telefono segun el empleado
Function muestraFono(ByVal empleado As String) As String
    Select Case empleado
        Case "Juan Miranda" : Return "580-3934"
        Case "Carlos Ramos" : Return "580-3935"
        Case "Jorge Garcia" : Return "580-3936"
        Case "Miguel Robles" : Return "580-3937"
    End Select
    Return "000-0000"
End Function
```

```
'Función que determina el precio segun el pedido seleccionado
Function determinaPrecio(ByVal mesa As String, ByVal pintado As String) As Double
    If pintado = "SI" Then
        Select Case mesa
            Case "Tipo A" : Return 70
            Case "Tipo B" : Return 50
            Case "Tipo C" : Return 20
        End Select
    Else
        Select Case mesa
            Case "Tipo A" : Return 60
            Case "Tipo B" : Return 40
            Case "Tipo C" : Return 15
        End Select
    End If
    Return 0
End Function

'Función que determina el tiempo de corte
Function calculaTiempoCorte(ByVal mesa As String) As Integer
    Select Case mesa
        Case "Tipo A" : Return 1
        Case "Tipo B" : Return 2
        Case "Tipo C" : Return 4
    End Select
    Return 0
End Function

'Función que determina el tiempo de montaje
Function calculaTiempoMontaje(ByVal mesa As String) As Integer
    Select Case mesa
        Case "Tipo A" : Return 8
        Case "Tipo B" : Return 4
        Case "Tipo C" : Return 2
    End Select
    Return 0
End Function

'Función que determina el tiempo de pintado
Function calculaTiempoPintura(ByVal mesa As String) As Integer
    Select Case mesa
        Case "Tipo A" : Return 4
        Case "Tipo B" : Return 3
        Case "Tipo C" : Return 4
    End Select
    Return 0
End Function

'Función que determina el tiempo total del pedido
Function calculaTiempo(ByVal mesa As String, ByVal pintado As String,
                       ByVal cantidad As Integer)
    If pintado = "SI" Then
        Select Case mesa
```

```
            Case "Tipo A" : Return calculaTiempoCorte(mesa) * cantidad +
                                   calculaTiempoMontaje(mesa) * cantidad +
                                   calculaTiempoPintura(mesa) * cantidad
            Case "Tipo B" : Return calculaTiempoCorte(mesa) * cantidad +
                                   calculaTiempoMontaje(mesa) * cantidad +
                                   calculaTiempoPintura(mesa) * cantidad
            Case "Tipo C" : Return calculaTiempoCorte(mesa) * cantidad +
                                   calculaTiempoMontaje(mesa) * cantidad +
                                   calculaTiempoPintura(mesa) * cantidad
        End Select
        Else
            Select Case mesa
             Case "Tipo A" : Return calculaTiempoCorte(mesa) * cantidad +
                                    calculaTiempoMontaje(mesa) * cantidad
             Case "Tipo B" : Return calculaTiempoCorte(mesa) * cantidad +
                                    calculaTiempoMontaje(mesa) * cantidad
             Case "Tipo C" : Return calculaTiempoCorte(mesa) * cantidad +
                                    calculaTiempoMontaje(mesa) * cantidad
            End Select
        End If
        Return 0
    End Function

    'Función que calcula el subtotal segun la cantidad y el precio
    Function calculaSubtotal(ByVal cantidad As Integer, ByVal precio As Double)
        Return cantidad * precio
    End Function
End Class
```

Caso desarrollado 5 — Procedimiento de Propiedad - Control de pago por consumo

Objetivo	Controla el simulador de pago por consumo.
Referencias	Usar procedimiento de propiedad.
Descripción del caso	Una empresa de suministro de energía desea implementar una aplicación que permita simular el pago que debe realizar un usuario de acuerdo a la lectura de su medidor de energía. Los datos que necesita la aplicación es el tipo de conexión que tiene el usuario, la lectura anterior y actual de su medidor. La aplicación debe mostrar la siguiente asignación:

Tipo de conexión	Tarifa	Cargo fijo
Monofásica	$0.45	$5.00
Bifásica	$1.20	$10.00
Trifásica	$2.50	$25.00

La tarifa se aplica de acuerdo al tipo de conexión, este monto se multiplica con el consumo que se registre en las lecturas del medidor. En el caso del cargo fijo se aplica por tipo de conexión y se le asigna directamente al monto a pagar por el usuario.

Tener en cuenta	• Usar procedimientos de propiedad. • Al iniciar la aplicación se debe mostrar precargados los tipos de conexión. • Al seleccionar un tipo de conexión se debe mostrar la tarifa y monto de cargo fijo según el tipo de conexión. • El botón **Simular** muestra la información de la tarifa, cargo fijo, consumo actual según las lecturas, el subtotal a pagar, el monto asignado a IVA y el total a pagar por el usuario. El monto asignado al IVA asciende al 18 % del monto subtotal y el total a pagar es la diferencia entre el subtotal y el IVA. • Finalmente, validar los controles tipo de conexión, lectura anterior y lectura actual; en caso hubiera algún error, mostrar un mensaje usando procedimientos de propiedad.
Formulario propuesto	

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.
   - Seleccione Aplicación de Windows Forms.
   - Asigne el nombre pjEnergiaElectrica.
2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmReciboPago.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).

4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmReciboPago Simulación de pago por consumo
Label1	Text	SIMULADOR DE PAGO POR CONSUMO
Label2	Text	TIPO DE CONEXIÓN
Label3	Text	TARIFA
Label4	(Name) Text	lblTarifa 0.00
Label5	Text	CARGO FIJO
Label6	Text	LblCargoFijo 0.00
Label7	Text	LECTURA ANTERIOR
Label8	Text	LECTURA ACTUAL
GroupBox1	Text	Datos de la conexión
GroupBox2	Text	Datos del consumo
Combobox1	(Name)	cboTipoConexion
TextBox1	(Name)	txtLecturaAnt
TextBox2	(Name)	txtLecturaAct
Button1	(Name) Text	btnSimular SIMULAR
Listbox1	(Name)	lstResumen

Código fuente:

```
Public Class frmReciboPago
    Private Sub frmReciboPago_Load(…) Handles MyBase.Load
        LlenaTipoConexion()
    End Sub

    Private Sub btnSimular_Click(…) Handles btnSimular.Click
        If valida() = "" Then
            imprimir()
        Else
            MessageBox.Show("El error se encuentra en " & valida())
        End If
    End Sub

    Private Sub cboTipoConexion_SelectedIndexChanged(…)
       lblTarifa.Text = determinaTarifa(getTipoConexion()).ToString("C")
      lblCargoFijo.Text = determinaCargoFijo(getTipoConexion()).ToString("C")
    End Sub

    '** SUBPROCEDIMIENTOS SIN PARAMETROS **
    'Subprocedimiento que permite llenar los tipos de conexiones
    Sub LlenaTipoConexion()
        With cboTipoConexion.Items
            .Add("Monofasica")
            .Add("Bifasica")
            .Add("Trifasica")
        End With
    End Sub

    'Subprocedimiento que permite imprimir los resultados
    Sub imprimir()
        Dim tarifa As Double = determinaTarifa(getTipoConexion())
        Dim cargoFijo As Double = determinaCargoFijo(getTipoConexion())
      Dim consumo As Double=calculaConsumo(getLecturaAnterior(),getLecturaActual())
        Dim subtotal As Double = calculaSubtotal(cargoFijo, consumo)
        Dim IVA As Double = calculaIVA(subtotal)
        Dim total As Double = calculaTotal(IVA, subtotal)

        With lstResumen.Items
            .Add("Tarifa: " & tarifa.ToString("C"))
            .Add("Cargo fijo: " & cargoFijo.ToString("C"))
            .Add("Consumo actual: " & consumo.ToString())
            .Add("-------------------------------------------------")
            .Add("Subtotal: " & subtotal.ToString("C"))
            .Add("IVA: " & IVA.ToString("C"))
            .Add("TOTAL A PAGAR: " & total.ToString("C"))
        End With
    End Sub
```

```
'** PROCEDIMIENTO DE PROPIEDAD **
'Procedimiento de propiedad que captura el tipo de conexión
ReadOnly Property getTipoConexion As String
    Get
        Return cboTipoConexion.SelectedItem
    End Get
End Property

'Procedimiento de propiedad que captura la lectura anterior
ReadOnly Property getLecturaAnterior As Integer
    Get
        Return Integer.Parse(txtLecturaAnt.Text)
    End Get
End Property

'Procedimiento de propiedad que captura la lectura actual
ReadOnly Property getLecturaActual As Integer
    Get
        Return Integer.Parse(txtLecturaAct.Text)
    End Get
End Property

'Procedimiento de propiedad que determina la tarifa segun el tipo de conexion
ReadOnly Property determinaTarifa(ByVal tipoConexion As String) As Double
        Get
            Select Case tipoConexion
                Case "Monofasica" : Return 0.45
                Case "Bifasica" : Return 1.2
                Case "Trifasica" : Return 2.5
            End Select
            Return 0
        End Get
    End Property

    'Procedimiento de propiedad que determina el cargo fijo segun el tipo de conexion
     ReadOnly Property determinaCargoFijo(ByVal tipoConexion As String)
As Double
        Get
            Select Case tipoConexion
                Case "Monofasica" : Return 5
                Case "Bifasica" : Return 10
                Case "Trifasica" : Return 25
            End Select
            Return 0
        End Get
    End Property

    'Procedimiento de propiedad que calcula el valor del consumo segun las lecturas
    ReadOnly Property calculaConsumo(ByVal lecturaAnt As Integer,
```

```
                                        ByVal lecturaAct As Integer) As Integer
        Get
            Return lecturaAct - lecturaAnt
        End Get
    End Property

    'Procedimiento de propiedad que determina el costo por consumo
    ReadOnly Property calculaCostoxConsumo(ByVal lectura As Integer,
                                           ByVal tarifa As Double) As Double
        Get
            Return lectura * tarifa
        End Get
    End Property

    'Procedimiento de propiedad que calcula el subtotal a pagar
    ReadOnly Property calculaSubtotal(ByVal cargoFijo As Double,
                                      ByVal cargoEnergia As Double) As Double
        Get
            Return cargoEnergia + cargoFijo
        End Get
    End Property

    'Procedimiento de propiedad que calcula el impuesto IVA
    ReadOnly Property calculaIVA(ByVal subtotal As Double) As Double
        Get
            Return subtotal * 0.18
        End Get
    End Property

    'Procedimiento de propiedad que calcula el total a pagar
    ReadOnly Property calculaTotal(ByVal IVA As Double,
                                   ByVal subtotal As Double) As Double
        Get
            Return subtotal + IVA
        End Get
    End Property

    'Procedimiento de propiedad que valida el ingreso de valores en el formulario
    ReadOnly Property valida() As String
        Get
            If cboTipoConexion.SelectedIndex = -1 Then
            cboTipoConexion.Focus()
            Return "tipo de conexion"
        ElseIf txtLecturaAct.Text < txtLecturaAnt.Text Then
            txtLecturaAct.Clear()
            txtLecturaAnt.Clear()
            txtLecturaAct.Focus()
            Return "La lectura ya que la lectura anterior debe ser
                    inferior a la actual..!!"
        ElseIf String.IsNullOrWhiteSpace(txtLecturaAnt.Text) Or
               Not IsNumeric(txtLecturaAnt.Text) Then
            txtLecturaAnt.Clear()
```

```
            txtLecturaAnt.Focus()
            Return "lectura anterior"
        ElseIf String.IsNullOrWhiteSpace(txtLecturaAct.Text) Or
               Not IsNumeric(txtLecturaAct.Text) Then
            txtLecturaAct.Clear()
            txtLecturaAct.Focus()
            Return "lectura actual"
        Else
            Return ""
        End If
    End Get
End Property
End Class
```

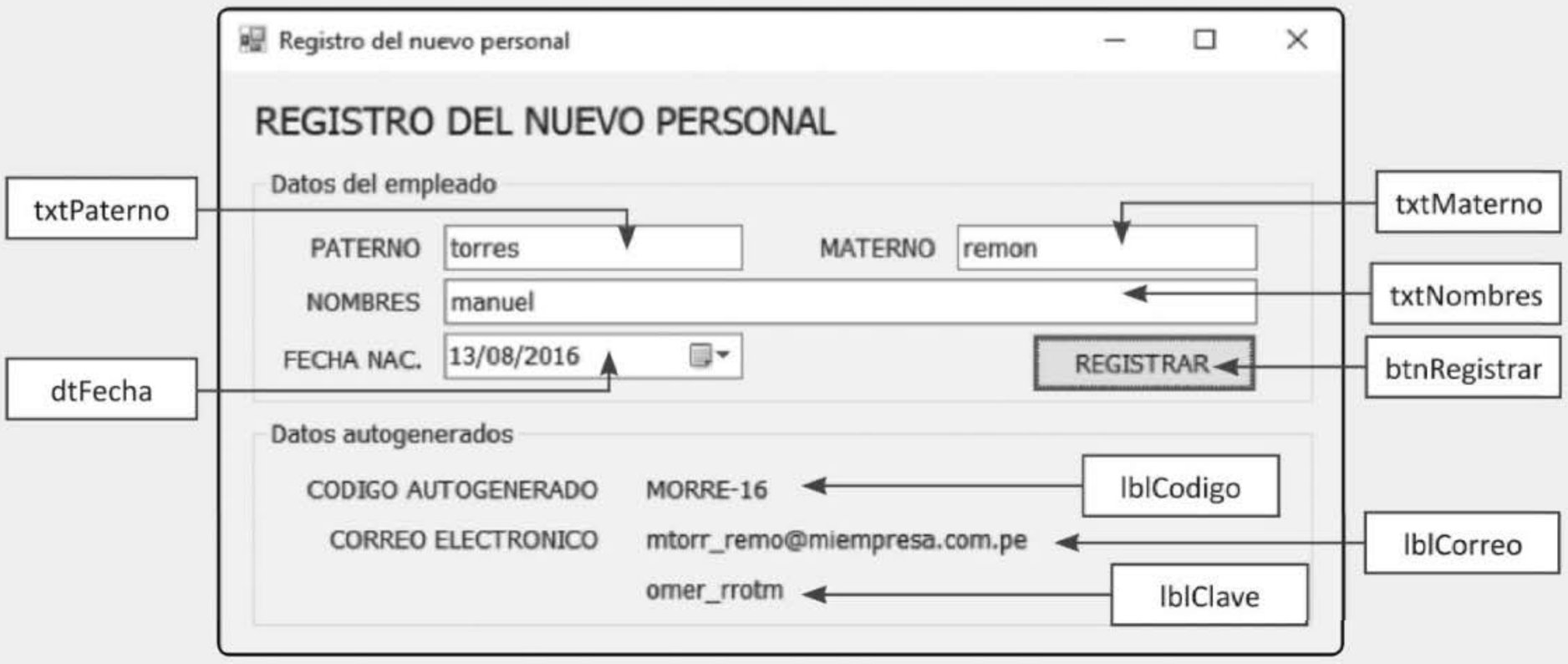
Caso desarrollado 6 Funciones de cadena - Registro de empleado

Objetivo	Controlar el registro de los empleados.
Referencias	Usar funciones de tipo cadena.
Descripción del caso	Una empresa necesita una aplicación que permite controlar el registro de los nuevos empleados para determinar su código de usuario en el sistema, así como su correo electrónico corporativo. Con respecto del código, se obtiene a partir de la primera letra de su nombre, la segunda y tercera letra de su apellido paterno, y las dos primeras letras del apellido materno; al final, agregar los dos últimos números del año de su nacimiento. Con respecto de su correo electrónico, está formado por la primera letra de su nombre, las cuatro primera letras de su apellido paterno, los cuatro primeras letras de su apellido materno y al final agregar el texto «@miempresa.com.pe». Finalmente, se debe mostrar la clave autogenerada para el acceso a su correo electrónico, la cual está formada por el nombre del correo con los caracteres invertidos.
Tener en cuenta	• Usar funciones de tipo cadena. • La fecha de nacimiento debe ser seleccionado a partir del control DateTimePicker. • El botón **Registrar** permite determinar el codigo del empleado, correo electrónico y la clave del correo. • Finalmente, validar los controles del apellido paterno, materno y nombres; en caso hubiera algún error, mostrar un mensaje usando procedimientos de propiedad.

Formulario propuesto

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.
   - Seleccionar Aplicación de Windows Forms.
   - Asignar el nombre pjRegistroEmpleado.
2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmRegistro.
3. Diseñar la GUI (Interfaz Gráfica de Usuario).



4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmRegistro Registro del nuevo personal
Label1	Text	REGISTRO DEL NUEVO PERSONAL
Label2	Text	PATERNO
Label3	Text	MATERNO
Label4	Text	NOMBRES
Label5	Text	FECHA NAC.
Label6	Text	CODIGO AUTOGENERADO
Label7	(Name) Text	lblCodigo lblCodigo
Label8	Text	CORREO ELECTRONICO
Label9	(Name) Text	lblCorreo lblCorreo
Label10	(Name) Text	lblClave lblClave
GroupBox1	Text	Datos del empleado
GroupBox2	Text	Datos autogenerados
TextBox1	(Name)	txtPaterno
TextBox2	(Name)	txtMaterno
TextBox3	(Name)	txtNombres
Button1	(Name) Text	btnRegistrar REGISTRAR
DateTimePicker1	(Name)	dtFecha

Código fuente:

```
Public Class frmRegistro
    Private Sub btnRegistrar_Click(…) Handles btnRegistrar.Click
        If valida() = "" Then
            imprimir()
        Else
            MessageBox.Show("El error se encuentra en " & valida())
        End If
    End Sub
    '**FUNCIONES**
    'Función que captura y retorna el apellido paterno del empleado
```

```
    Function getPaterno() As String
        Return txtPaterno.Text
    End Function

    'Función que captura y retorna el apellido materno del empleado
    Function getMaterno() As String
        Return txtMaterno.Text
    End Function

    'Función que captura y retorna el nombre del empleado
    Function getNombres() As String
        Return txtNombres.Text
    End Function

    'Función que captura y retorna la fecha de nacimiento del empleado
    Function getFechaNac() As String
        Return dtFecha.Value.ToShortDateString
    End Function

    'Función que determina el codigo de usuario del empleado
    Function determinaCodigo() As String
        Dim año As String = Strings.Right(getFechaNac(), 2)
        Dim paterno As String = Strings.Mid(getPaterno(), 2, 2)
        Dim materno As String = Strings.Left(getMaterno(), 2)
        Dim nombre As String = Strings.Left(getNombres(), 1)

      Dim codigo As String = Strings.UCase(nombre & paterno & materno & "-" & año)

        Return codigo
    End Function

    'Función que determina el correo electrónico del empleado
    Function determinaCorreo() As String
        Dim nombre As String = Strings.Left(getNombres(), 1)
        Dim paterno As String = Strings.Left(getPaterno(), 4)
        Dim materno As String = Strings.Left(getMaterno(), 4)

        Dim correo As String = Strings.LCase(nombre & paterno & "_" &
                                             materno & "@miempresa.com.pe")
        Return correo
    End Function

    'Función que determina la clave segun su correo del empleado
    Function determinaClaveCorreo(ByVal correo As String) As String
        Dim letras As String = ""
        For i = 1 To correo.Count
            If GetChar(correo, i) <> "@" Then
                letras += GetChar(correo, i)
            Else
                Exit For
            End If
        Next
        Return Strings.StrReverse(letras)
    End Function

    'Subprocedimiento que permite imprimir los resultados
    Sub imprimir()
        lblCodigo.Text = determinaCodigo()
```

```
        lblCorreo.Text = determinaCorreo()
        lblClave.Text = determinaClaveCorreo(determinaCorreo())
    End Sub

    'Función que valida los controles del formulario
    Function valida() As String
        If String.IsNullOrWhiteSpace(txtPaterno.Text) Then
            txtPaterno.Clear()
            txtPaterno.Focus()
            Return "paterno del personal"
        ElseIf String.IsNullOrWhiteSpace(txtMaterno.Text) Then
            txtMaterno.Clear()
            txtMaterno.Focus()
            Return "materno del personal"
        ElseIf String.IsNullOrWhiteSpace(txtNombres.Text) Then
            txtNombres.Clear()
            txtNombres.Focus()
            Return "nombre del personal"
        Else
            Return ""
        End If
    End Function
End Class
```

Caso desarrollado 7 Funciones de fecha - Simulador de pago por préstamo

Objetivo	Simular los pagos de un préstamo en una entidad bancaria.
Referencias	Usar funciones de tipo fecha para las fechas de pago.
Descripción del caso	Una entidad bancaria realiza prestamos personales a sus clientes por un determinado numero de periodos, a una determinada tasa de interés que puede ser mensual o anual y la especificación de la primera fecha de pago. La aplicación deberá mostrar la información simulada de los números de cuotas y, fechas de pago a partir de la primera fecha, el monto de la deuda disminuida cuota a cuota, el monto de interés y el pago mensual que debe realizar el cliente si acepta la cotización. Usar la siguientes fórmulas para realizar los cálculos: • Fórmula para el pago mensual. $$\frac{Tasa\ interés \times monto\ préstamo}{1-(1+Tasa\ interés)^{-periodo}}$$ • Fórmula para la conversión de tasa anual a mensual. $$(1+Tasa\ interés)^{\frac{1}{12}}-1$$

	• Fórmula para amortización. $$\frac{\textit{Monto préstamo}}{\textit{Periodo}}$$ Donde, la tasa de interés debe ser expresada en el valor decimal del porcentaje; por ejemplo, el 1 % debe ser expresado en 1/100 o 0.01. Finalmente, se debe mostrar en una lista las cuotas, fechas de pago, monto de la deuda reducida cuota a cuota, el monto de amortización, el monto de interés y el pago mensual.
Tener en cuenta	• Usar funciones de tipo fecha. • Al iniciar la aplicación debe mostrarse precargados los tipos de tasa (mensual y anual). • La primera fecha de pago debe ser seleccionada a partir del control DateTimePicker. • Para el periodo y el porcentaje de tasa debe usarse el control NumericUpDown respectivamente. • El botón **Simular** permite determinar mostrar todos los datos solicitados por la aplicación. • Finalmente, validar los controles del monto de préstamo y la selección del tipo de tasa; en caso hubiera algún error, mostrar un mensaje usando procedimientos de propiedad.
Formulario propuesto	

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.
   - Seleccionar Aplicación de Windows Forms.
   - Asignar el nombre pjPrestamo.

2. Cambiamos el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmSimulador.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).

4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmSimulador simulador de pago por préstamo
Label1	Text	SIMULADOR DE PAGO POR PRESTAMO
Label2	Text	MONTO DE PRESTAMO
Label3	Text	PERIODO
Label4	Text	TASA
Label5	Text	1RA FECHA DE PAGO
GroupBox1	Text	Datos del préstamo
TextBox1	(Name)	txtMonto
NumericUpDown1	(Name)	nudPeriodo
NumericUpDown2	(Name)	nudTasa
Button1	(Name) Text	btnSimular SIMULAR
DateTimePicker1	(Name)	DtFecha
ListView1	(Name)	lvCuotas

Código fuente:

```
Public Class frmSimulador
    Private Sub frmSimulador_Load(…) Handles MyBase.Load
        llenaTasa()
    End Sub
    Private Sub btnSimular_Click(…) Handles btnSimular.Click
        If valida() = "" Then
            imprimirCuotas()
        Else
            MessageBox.Show("El error se encuentra en " & valida())
        End If
    End Sub

    'Subprocedimiento que permite llenar los tipos de tasas
    Sub llenaTasa()
        With cboTasa.Items
            .Add("Mensual")
            .Add("Anual")
        End With
    End Sub

    'Función que captura el monto de préstamo
    Function getMonto() As Double
        Return Double.Parse(txtMonto.Text)
    End Function

    'Función que captura el numero de periodos del préstamo
    Function getPeriodo() As Integer
        Return nudPeriodo.Value
    End Function

    'Función que captura el tipo de tasa [Mensual|Anual] del préstamo
    Function getTipoTasa() As String
        Return cboTasa.SelectedItem
    End Function

    'Función que captura el porcentaje de interés
    Function getTasa() As Integer
        Return nudTasa.Value
    End Function

    'Función que captura la fecha del primer pago
    Function getFecha() As DateTime
        Return dtFecha.Value.ToShortDateString
    End Function

    'Función que determina el monto de pago mensual
    Function calculaPagoMensual(ByVal monto As Double, ByVal tasa As Double,
                                ByVal periodo As Integer) As Double
        Return (tasa * monto) / (1 - (1 + tasa) ^ -periodo)
    End Function
```

```
    'Función que determina el monto de la tasa segun el tipo de tasa
    Function determinaTasa(ByVal tipoTasa As String) As Double
        If tipoTasa = "Mensual" Then
            Return getTasa() / 100
        ElseIf tipoTasa = "Anual" Then
            Return ((1 + getTasa() / 100) ^ (1 / 12)) - 1
        Else
            Return 0
        End If
    End Function

    'Subprocedimiento que imprime en la lista los resultados
    Sub imprimirCuotas()
        lvCuotas.Items.Clear()
        Dim deuda As Double = getMonto()
        Dim amortizacion As Double = getMonto() / getPeriodo()
        For i = 0 To getPeriodo()
           Dim pagoMensual As Double = Math.Round(calculaPagoMensual(getMonto(),
                                                  determinaTasa(getTipoTasa()),
                                                                getPeriodo()), 2)
            Dim interes As Double = pagoMensual - amortizacion
            Dim fila As New ListViewItem(i)
            fila.SubItems.Add(DateAdd(DateInterval.Month, i,
                                      getFecha()).ToShortDateString())
            fila.SubItems.Add(deuda.ToString("C"))
            fila.SubItems.Add(amortizacion.ToString("C"))
            fila.SubItems.Add(interes.ToString("C"))
            fila.SubItems.Add(pagoMensual.ToString("C"))
            lvCuotas.Items.Add(fila)
            deuda -= amortizacion
        Next
    End Sub

    'Función que valida los controles
    Function valida() As String
        If String.IsNullOrWhiteSpace(txtMonto.Text) Or
           Not IsNumeric(txtMonto.Text) Then
            txtMonto.Clear()
            txtMonto.Focus()
            Return "monto de préstamo"
        ElseIf cboTasa.SelectedIndex = -1 Then
            cboTasa.Focus()
            Return "tipo de tasa"
        Else
            Return ""
        End If
    End Function
End Class
```

Caso desarrollado 8 Uso del InputBox - Casa de cambio

Objetivo	Controlar el tipo de cambio en soles y dólares.
Referencias	Usar la sentencia InputBox.
Descripción del caso	Una casa de cambio ha implementado una aplicación que permite mostrar el monto según el tipo de moneda. La aplicación deberá permitir el ingreso de un monto y un tipo de moneda; al calcular, debe mostrar el monto del tipo de cambio mostrado de forma estática y con valores preestablecidos. Asimismo, implementar un módulo de actualización de dichos montos. Usar la sentencia InputBox para la actualización de monto tanto en la compra como en la venta. La fórmula para el cambio de tipo venta se realiza mediante una división entre el monto y el valor de la venta. Mientras que la compra es el producto del monto y el valor expresado en la venta.
Tener en cuenta	• Usar la sentencia Inputbox para la actualización de los montos de la compra y venta. • Al iniciar la aplicación debe mostrarse precargados los tipos de moneda (soles y dólares). • Los montos de compra y venta inicialmente deben mostrar los valores S/. 2.35 y S/. 2.40 respectivamente. • Se debe implementar dos botones de actualización para los montos de compra y venta; haciendo que las variables globales se actualicen. • El botón **Calcular** permite determinar y mostrar los datos solicitados por la aplicación. • Finalmente, validar los controles del monto y la selección del tipo de cambio; en caso hubiera algún error, mostrar un mensaje usando procedimientos de propiedad.
Formulario propuesto	

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.

- Seleccionar Aplicación de Windows Forms.
- Asignar el nombre pjCasaCambio.

2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmCambio.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).

4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmCambio Casa de cambio
Label1	Text	CASA DE CAMBIO
Label2	Text	MONTO
Label3	Text	MONEDA
Label4	Text	COMPRA
Label5	Text	lblCompra 0.00
Label6	Text	VENTA
Label7	Text	lblVenta 0.00
Label8	Text	MONTO
Label9	Text	lblMonto 0.00

Control	Propiedad	Valor
Label10	Text	MONEDA
Label11	Text	lblMoneda lblMoneda
Label12	Text	CAMBIO
Label13	Text	lblCambio 0.00
GroupBox1	Text	Modulo de ingreso de datos
GroupBox2	Text	Modulo de actualización
TextBox1	(Name)	txtMonto
Combobox1	(Name)	cboMoneda
Button1	(Name) Text	btnCalcular CALCULAR
Button2	(Name) Text	btnActualizaCompra ACTUALIZA
Button3	(Name) Text	btnActualizaVenta ACTUALIZA

Código fuente:

```
Public Class frmCambio
    'Asignación de variables GLOBALES
    Dim compra As Double = 2.35, venta As Double = 2.4
    Private Sub frmCambio_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        llenaMoneda()
        asignaMontoCompra()
    End Sub

    Private Sub btnCalcular_Click(…) Handles btnCalcular.Click
        If valida() = "" Then
        Dim monto As Double = getMonto()
        Dim moneda As String = getMoneda()
        Dim cambio As Double = calculaMonto(moneda, monto)

        lblMoneda.Text = moneda
        If moneda = "Dolar" Then
            lblMonto.Text = "$" & monto.ToString("0.00")
            lblCambio.Text = "S/. " & cambio.ToString("0.00")
        Else
            lblMonto.Text = "S/." & monto.ToString("0.00")
```

```
            lblCambio.Text = "$. " & cambio.ToString("0.00")
        End If
    Else
        MessageBox.Show("El error se encuentra en " & valida())
    End If
End Sub

Private Sub btnActualizaCompra_Click(…) Handles btnActualizaCompra.Click
    compra = InputBox("Ingrese monto de compra: ")
    asignaMontoCompra()
End Sub

Private Sub btnActualizaVenta_Click(…) Handles btnActualizaVenta.Click
    venta = InputBox("Ingrese monto de venta: ")
    asignaMontoCompra()
End Sub

'Subprocedimiento que llena los tipos de monedas
Sub llenaMoneda()
    With cboMoneda.Items
        .Add("Soles")
        .Add("Dolar")
    End With
End Sub

'Función que captura los montos
Function getMonto() As Double
    Return Double.Parse(txtMonto.Text)
End Function

'Función que captura el tipo de moneda
Function getMoneda() As String
    Return cboMoneda.SelectedItem
End Function

'Subprocedimiento que muestra los valores globales
Sub asignaMontoCompra()
    lblCompra.Text = "S/." & compra.ToString("0.00")
    lblVenta.Text = "S/." & venta.ToString("0.00")
End Sub

'Función que calcula el monto segun el tipo de moneda
Function calculaMonto(ByVal moneda As String, ByVal monto As Double) As Double
    If moneda = "Dolar" Then
        Return monto * venta
    ElseIf moneda = "Soles" Then
        Return monto / compra
    Else
        Return 0
    End If
End Function
```

```
    'Función que valida el ingreso de valores en el formulario
    Function valida() As String
        If String.IsNullOrWhiteSpace(txtMonto.Text) Or
           Not IsNumeric(txtMonto.Text) Then
            txtMonto.Clear()
            txtMonto.Focus()
            Return "monto"
        ElseIf cboMoneda.SelectedIndex = -1 Then
            cboMoneda.Focus()
            Return "tipo de moneda"
        Else
            Return ""
        End If
    End Function
End Class
```

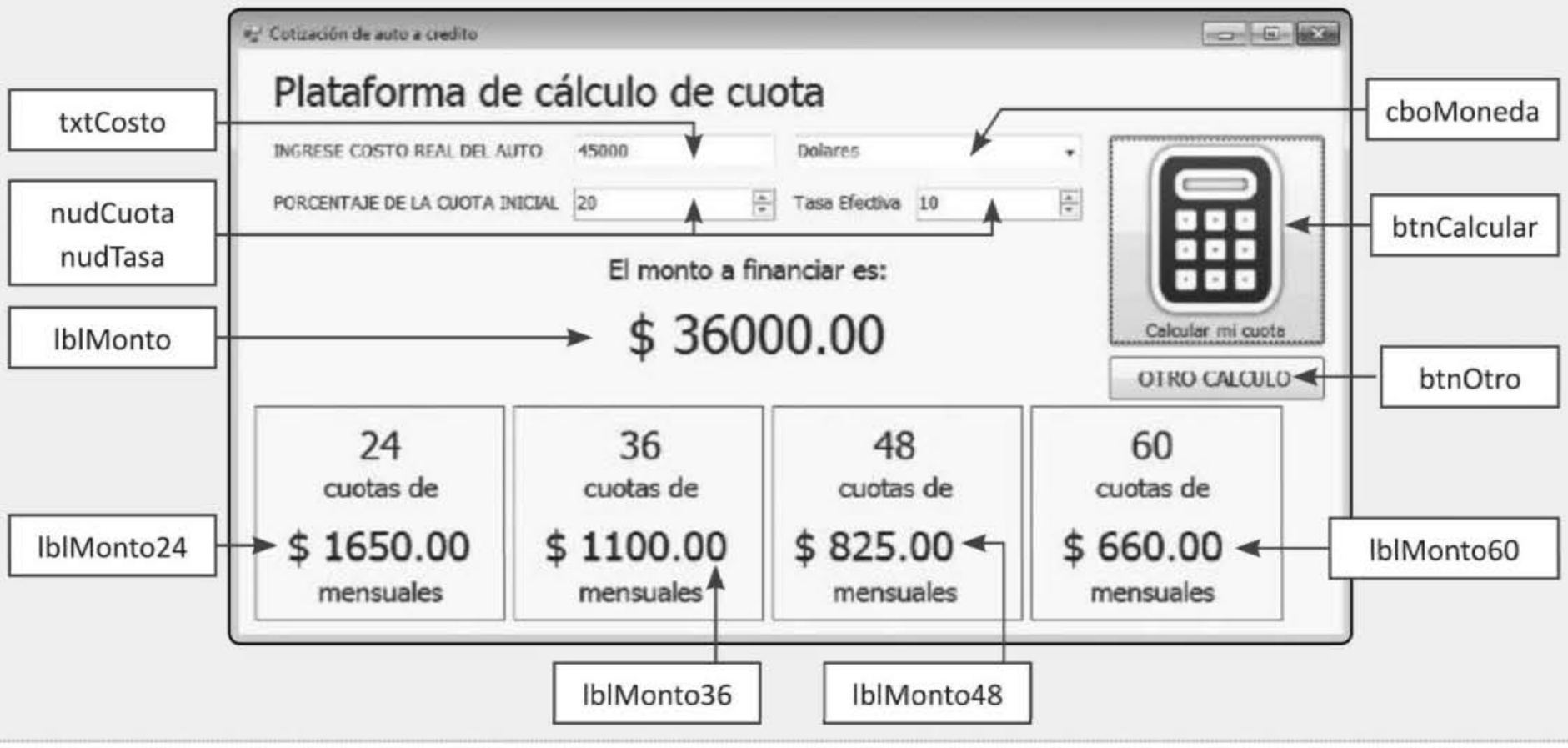
Caso desarrollado 9 Función lambda - Control de cotizaciones

Objetivo	Tener el control de las cotizaciones para la venta de autos al crédito.
Referencias	Usar función lambda.
Descripción del caso	Una entidad bancaria ha implementado una plataforma para cotizar la venta al crédito de automóviles, para lo cual un encargado del banco deberá ingresar el costo real del auto y el tipo de moneda, asignar un porcentaje para la cuota inicial y un porcentaje de la tasa efectiva. De acuerdo con esa información, emitirá el monto que el banco debe financiar y los montos mensuales que podría realizar el cliente según la cantidad de meses preestablecidos con que cuenta la entidad.
Tener en cuenta	• Usar funciones lambda de function y sub. • Al iniciar la aplicación se debe mostrar precargado los tipos de monedas como soles y dólares respectivamente. • Para seleccionar el porcentaje de la cuota inicial y la tasa efectiva usar el control NumericUpDown respectivamente. • El monto a financiar resulta del restar el monto del porcentaje de la cuota inicial al costo real del auto. • Hay que tener en cuenta que al seleccionar un tipo de moneda todos los montos expuestos en la aplicación deberán mostrar el símbolo correspondiente. • Finalmente, validar los controles de ingreso de costo y la selección del tipo de moneda; en caso hubiera algún error, mostrar un mensaje usando funciones sin parámetros.

Formulario propuesto

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto**.
   - Seleccionar la plantilla Visual Basic.
   - Seleccionar Aplicación de Windows Forms.
   - Asignar el nombre pjCotizacionAuto.
2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmCotizacion.
3. Diseñar la GUI (Interfaz Gráfica de Usuario).



4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmCotizacion Cotización de auto a crédito
Label1	Text	Plataforma de cálculo de cuota
Label2	Text	INGRESE COSTO REAL DEL AUTO
Label3	Text	PORCENTAJE DE LA CUOTA INICIAL
Label4	Text	TASA EFECTIVA
Label5	Text	EL MONTO A FINANCIAR ES:
Label6	(Name) Text	lblMonto 0.00
Label7	Text	24
Label8	Text	cuotas de
Label9	Text	36
Label10	Text	cuotas de
Label11	Text	48
Label12	Text	cuotas de
Label13	Text	60
Label14	Text	cuotas de
Label15	Text	lblMonto24 0.00
Label16	(Name) Text	lblMonto36 0.00
Label17	(Name) Text	lblMonto48 0.00
Label18	(Name) Text	lblMonto60 0.00
ComboBox1	(Name)	cboMoneda
TextBox1	(Name)	txtCosto
NumericUpDonw1	(Name) Value	nudCuota 20
NumericUpDonw2	(Name) Value	nudTasa 10
Button1	(Name) Text	btnCalcular CALCULAR MI CUOTA
Button2	(Name) Text	btnOtro OTRO CALCULO

Código fuente:

```
Public Class frmCotizacion
    Private Sub frmCotizacion_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        'Metodo Lambda del tipo Sub
        Dim llenaMoneda = Sub()
                              With cboMoneda.Items
                                  .Add("Soles")
                                  .Add("Dolares")
                              End With
                          End Sub

        'Invocación del método Lambda del tipo Sub
        llenaMoneda()
        cboMoneda.SelectedIndex = 0
    End Sub

    Private Sub btnCalcular_Click(…) Handles btnCalcular.Click
        If valida() = "" Then
            'Método Lambda del tipo Sub
            Dim imprimir = Sub(monto As Double, moneda As String)
                               If moneda = "Dolares" Then
                                   lblMonto.Text = "$ " & monto.ToString("0.00")
                                lblMonto24.Text = "$ " & calculaCuota24().ToString("0.00")
                                lblMonto36.Text = "$ " & calculaCuota36().ToString("0.00")
                                lblMonto48.Text = "$ " & calculaCuota48().ToString("0.00")
                                lblMonto60.Text = "$ " & calculaCuota60().ToString("0.00")
                                   Else
                                    lblMonto.Text = "S/. " & monto.ToString("0.00")
                                lblMonto24.Text = "S/. " & calculaCuota24().ToString("0.00")
                                lblMonto36.Text = "S/. " & calculaCuota36().ToString("0.00")
                                lblMonto48.Text = "S/. " & calculaCuota48().ToString("0.00")
                                lblMonto60.Text = "S/. " & calculaCuota60().ToString("0.00")
                                   End If
                               End Sub
            'Invocación del metodo Lambda del tipo Sub
            imprimir(calculaMontoFinanciado(), getMoneda())
        Else
            MessageBox.Show("El error se encuentra en " & valida())
        End If
    End Sub

    Private Sub btnOtro_Click(sender As Object, e As EventArgs) Handles btnOtro.Click
        limpiaControles()
    End Sub

    Sub limpiaControles()
        txtCosto.Clear()
        cboMoneda.SelectedIndex = 0
        nudCuota.Value = 20
```

```
        nudTasa.Value = 10
        lblMonto.Text = "0.00"
        lblMonto24.Text = "0.00"
        lblMonto36.Text = "0.00"
        lblMonto48.Text = "0.00"
        lblMonto60.Text = "0.00"
        txtCosto.Focus()
    End Sub

    'Implementación de funciones lambda
    Dim getCosto As Func(Of Double) = Function() Double.Parse(txtCosto.Text)
    Dim getMoneda As Func(Of String) = Function() cboMoneda.SelectedItem
    Dim getCuota As Func(Of Integer) = Function() nudCuota.Value
    Dim getTasa As Func(Of Integer) = Function() nudTasa.Value

    'Función lambda que calcula el monto de financiamiento
    Dim calculaMontoFinanciado As Func(Of Double) = Function()
                                                        'Obtener los valores ingresados
                                                    Dim costo As Double = getCosto()
                                                Dim cuotaInicial As Double = getCuota()

                                                    'Calcular y retornar el monto
                                                Return costo - costo * (cuotaInicial / 100)
                                                End Function

    'Función lambda que calcula el monto a pagar en 24 cuotas
    Dim calculaCuota24 As Func(Of Double) = Function()
                                                Dim monto As Double = calculaMontoFinanciado()
                                                Return (monto + (monto * (getTasa() / 100))) / 24
                                            End Function

    'Función lambda que calcula el monto a pagar en 36 cuotas
    Dim calculaCuota36 As Func(Of Double) = Function()
                                                Dim monto As Double = calculaMontoFinanciado()
                                                Return (monto + (monto * (getTasa() / 100))) / 36
                                            End Function

    'Función lambda que calcula el monto a pagar en 48 cuotas
    Dim calculaCuota48 As Func(Of Double) = Function()
                                                Dim monto As Double = calculaMontoFinanciado()
                                                Return (monto + (monto * (getTasa() / 100))) / 48
                                            End Function

    'Función lambda que calcula el monto a pagar en 60 cuotas
    Dim calculaCuota60 As Func(Of Double) = Function()
                                                Dim monto As Double = calculaMontoFinanciado()
                                                Return (monto + (monto * (getTasa() / 100))) / 60
                                            End Function

    'Función lambda que valida los controles del formulario
    Dim valida As Func(Of String) = Function()
```

```
                If String.IsNullOrWhiteSpace(txtCosto.Text) Or
                    Not IsNumeric(txtCosto.Text) Then
                        txtCosto.Clear()
                        txtCosto.Focus()
                        Return "costo del auto"
                ElseIf cboMoneda.SelectedIndex = -1 Then
                        cboMoneda.Focus()
                        Return "tipo de moneda"
                    Else
                        Return ""
                    End If
                End Function
End Class
```

Capítulo 7

Introducción a la programación orientada a objetos

Capacidad terminal:

Implementar aplicaciones que permitan el uso de clase y objeto.

Contenido

- Introducción a la programación orientada a objetos
- Los atributos de la clase
- Los objetos

7.1 Introducción a la programación orientada a objetos

El capítulo anterior dedicamos todo un tema a las funciones y subprocedimientos que se pueden implementar en Visual Basic. Justamente ese es el punto de partida para entender cómo trabaja la programación orientada a objetos, la cual es considerada como una evolución de la programación basada en métodos como procedimientos o funciones, incluyendo las funciones lambda.

Se debe tener en cuenta que las funciones o procedimientos se implementan dentro de la aplicación para su uso y que alguna modificación en una de ellas altera a todos las sentencias que lo invocan; es aquí donde la programación orientada a objetos se diferencia de los métodos, ya que una de sus características es que agrupa bloques de código con funcionalidades comunes que se guardarán en un archivo de tipo clase y, asimismo, que para usarlo se declararán objetos que permitirán instanciar a dicha clase para acceder a toda su funcionalidad, haciendo que el código de la aplicación se encuentre descargado de su lógica. Dentro de este marco se puede indicar que la programación orientada a objetos pretende agrupar un código en una aplicación, el cual es conocido como encapsulamiento.

Como se mencionó anteriormente, se pretende implementar una clase con funcionalidades comunes para la aplicación y para usar dichas funciones se debe crear un objeto, pero, ¿por qué un objeto? La idea es llegar a las funciones o procedimientos implementados dentro de la clase que de ahora en adelante se les va a llamar métodos y que para llegar a ellos se debe realizar mediante los objetos. Ver la siguiente imagen:

En un aula de clase se pueden encontrar muchos objetos y, si se observa mejor, hay objetos como las sillas, mesas, libros, etc., pero hay algo en común entre todos los objetos que se debe considerar para el desarrollo de las aplicaciones, aparte de mencionar que la programación orientada a objetos coge objetos del mundo real para plasmarlos en clases. Los elementos comunes entre todos los objetos son los atributos y métodos, donde los atributos son las características que cuenta el objeto mientras que los métodos son la funcionalidad que este objeto tiene. A continuación, una lista de elementos encontrados en la imagen anterior:

Objeto	Propiedades	Métodos
Libro	Título Número de paginas Autor Año de edición	Agregar un libro Eliminar un libro Revisar un libro Modificar datos del libro
Silla	Alto Ancho Precio Color	Cambiar de color Modificar su precio
Monitor	Marca Modelo Precio	Encender el monitor Apagar el monitor Asignar un precio

Se debe mencionar que, a pesar de que tienen muchas propiedades y métodos comunes, los objetos son diferentes entre sí; por ejemplo, si se cuenta con diez libros, cada uno tendrá un título, número de páginas, autor, año de edición, otros elementos diferentes y es justamente esa diferencia la que hace que la programación orientada a objetos sea la técnica más adecuada para implementar aplicaciones.

7.1.1 Características de la programación orientada a objetos

Las principales características que presenta la programación orientada a objetos son:

- **Abstracción**: Es considerada como la acción de separar las cualidades que puede presentar un objeto del mundo real; esto quiere decir que uno no debe preocuparse en forma detallada, sino más bien obtener los atributos necesarios de un objeto. Se podría citar como ejemplo la venta de un producto en una tienda comercial, pues el cliente solo debe escogerlo del mostrador, llevarlo al área de cobranzas para realizar su pago y llevarlo a casa. Los elementos abstraídos del caso expuesto son los siguientes:
  - No es importante cómo llego el producto a la tienda comercial.

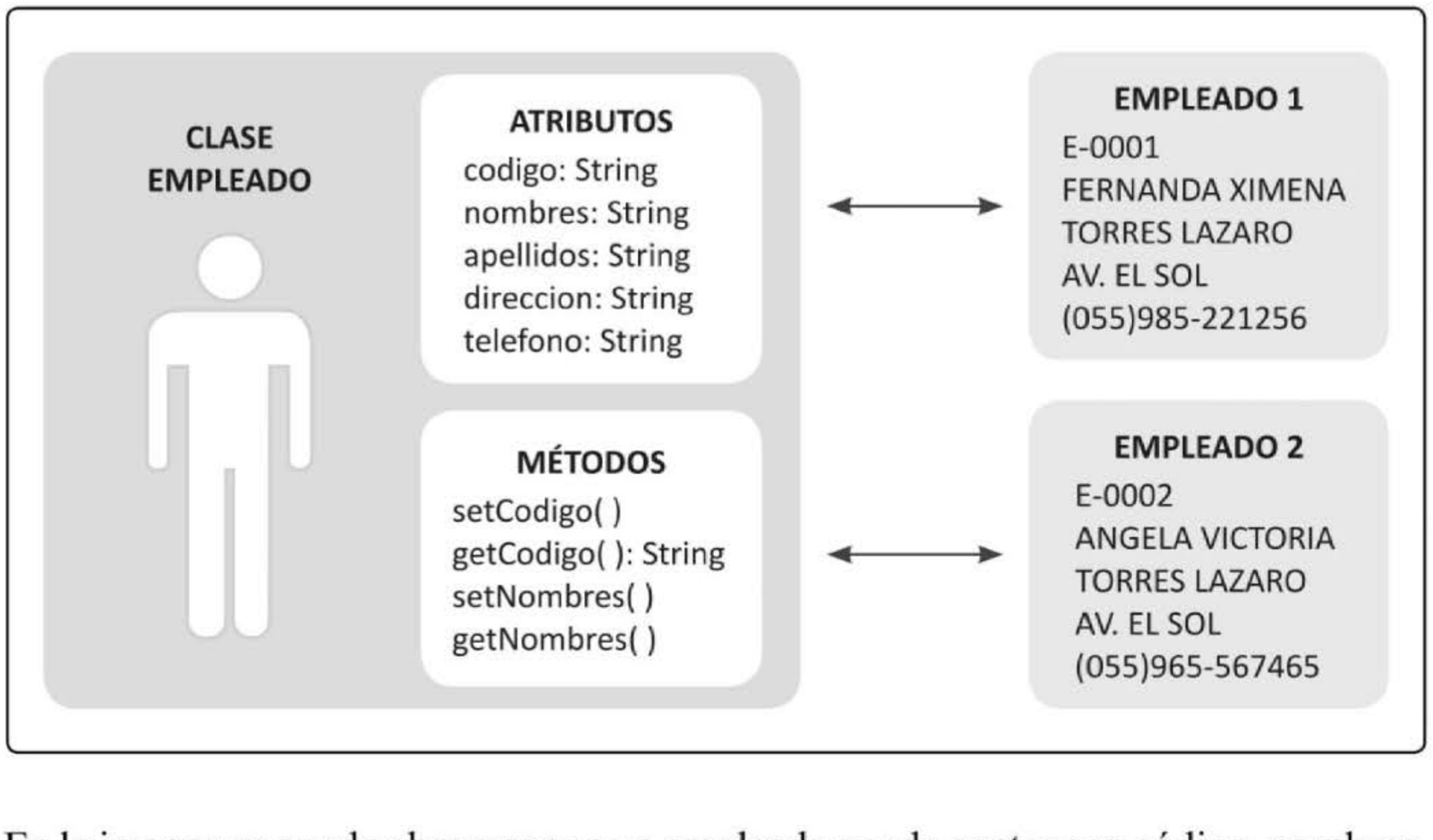
- No son importantes los insumos que usaron para la elaboración del producto.
- No son importantes los impuestos que pagaron las empresas elaboradoras en el producto a comprar.

A estos elementos se les llama abstracción, pues, hay elementos que sí son necesarios para la compra de dicho producto, como los siguientes:

- El registro de la fecha de vencimiento del producto.
- El costo de venta del producto.
- El nombre de la empresa elaboradora.

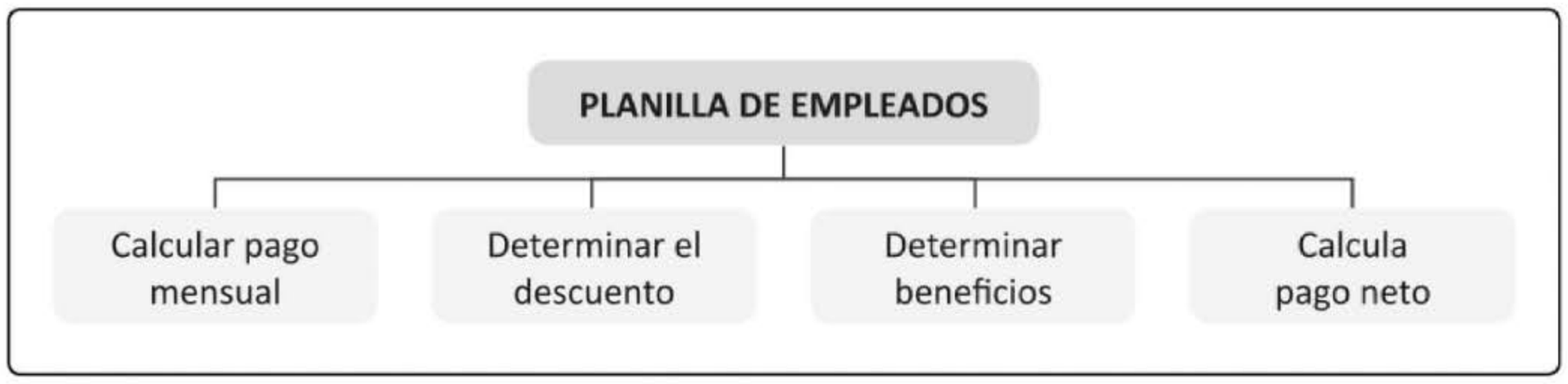
El punto principal de la abstracción es justamente abstraer los atributos y métodos de los objetos y agruparlos en una determinada clase.

- **Encapsulamiento**: Es un nivel de protección de los elementos que compone una clase, pues no es recomendable que los atributos de una clase sean accesibles directamente por el objeto. Encapsular implica envolver al atributo de tal forma que mediante un método se envíe y reciba valor de sus objetos. El encapsulamiento trabaja con un nivel de visibilidad de elementos públicos, privados y protegidos que se estudiarán en este capítulo.



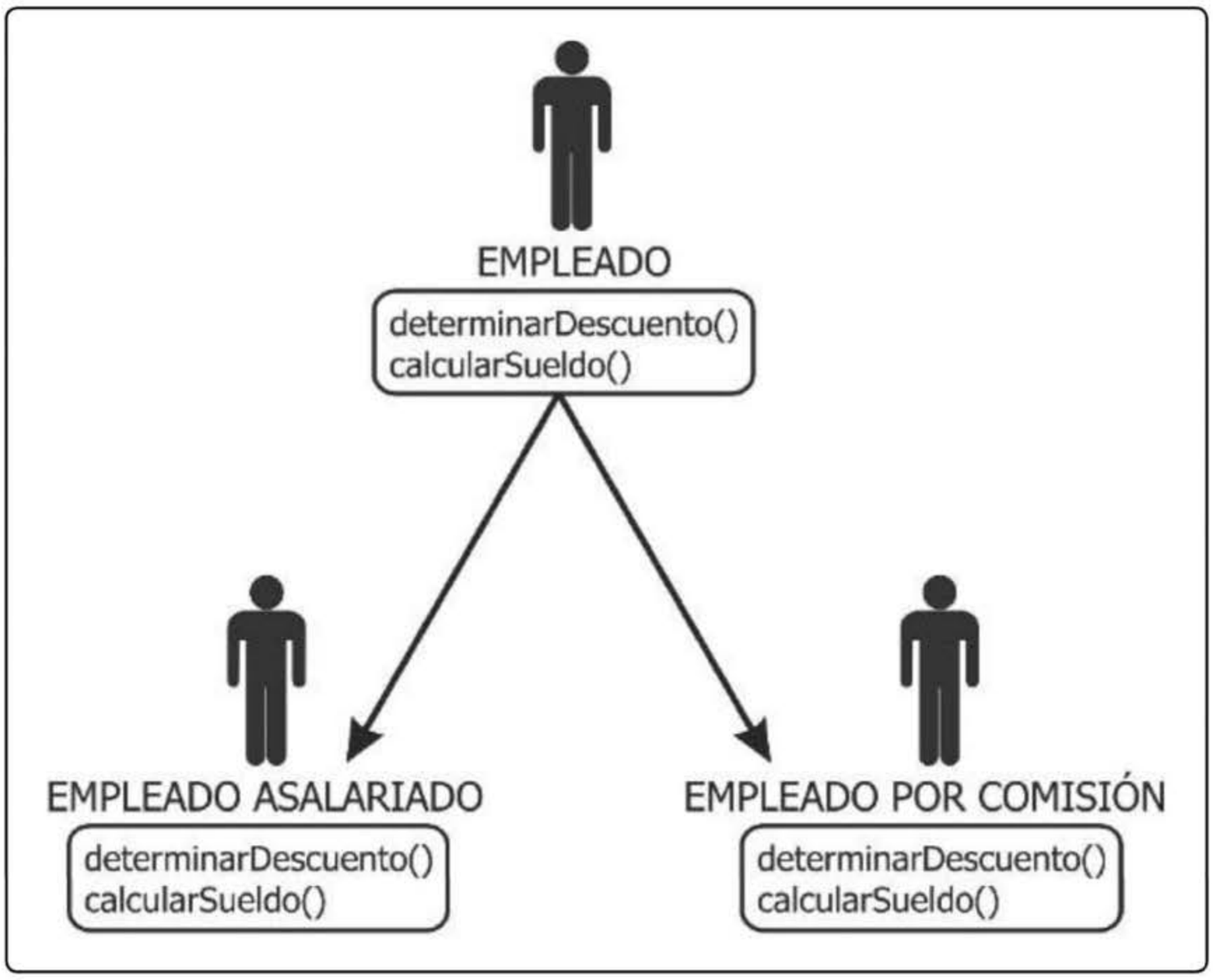
En la imagen se puede observar que un empleado puede contar con código, nombres, apellidos, dirección y teléfono como atributos; estos serán encapsulados mediante métodos que asignen y devuelvan valor como son Get y Set. El método Set asigna un valor al atributo y Get devuelve el valor a quien lo invoque.

- **Modularidad**: Se denomina así a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

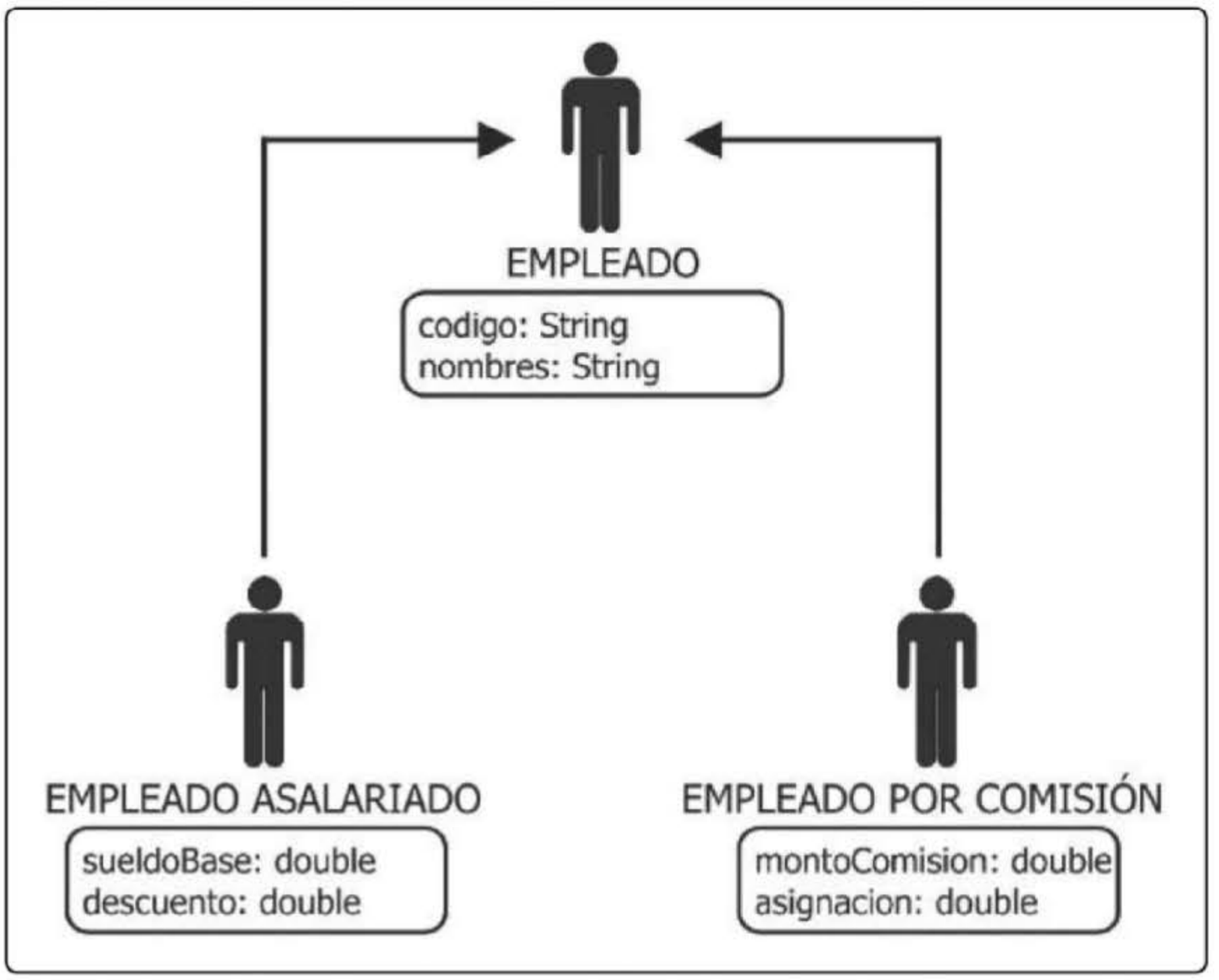


En la imagen anterior se puede observar que para un proceso de planilla de empleados necesitan muchas actividades como calcular el pago mensual, descuento, beneficios y el pago neto de un empleado, los cuales se implementarán en módulos visualmente separados pero unidos a la vez para el objetivo del proceso principal.

- **Polimorfismo**: Es uno de los pilares de la programación orientada a objetos, el cual permite implementar métodos con el mismo nombre pero de diferente accionar. Se podría nombrar la forma de calcular el sueldo de un empleado, el cual se diferencia por el tipo de empleado que puede presentar una empresa, en la cual uno tiene un salario fijo y otro gana por comisión. En ambos casos se debe calcular el sueldo pero se realiza de diferente forma.



- **Herencia**: Es una de las principales características que presenta la programación orientada a objetos, en la cual una clase hereda cualidades a otras clases, de tal forma que una clase llamada Base reúne características comunes entre otras clases y las implementa dentro de ella, para luego heredarlas a las clases derivadas o hijas.



En conclusión, cuando se ejecuta un programa orientado a objetos, ocurren tres acciones:

1. Se crean los objetos cuando se necesitan.
2. Los mensajes se envían desde un objeto y se reciben en otro.
3. Se borran los objetos cuando ya no son necesarios y se recupera la memoria ocupada por ellos.

Entonces se concluye que la programación orientada a objetos estructura el programa en objetos que cooperan entre sí. Cada objeto es instancia de alguna clase y las clases están relacionadas casi siempre mediante herencias.

7.1.2 La clase

Una clase es considerada como una plantilla que especifica atributos y comportamientos de un determinado tipo de objeto, ya sea físico o abstracto. Finalmente, una clase Visual Basic puede ser considerado como un nuevo tipo de datos y que a partir de él se podrán crear objetos.

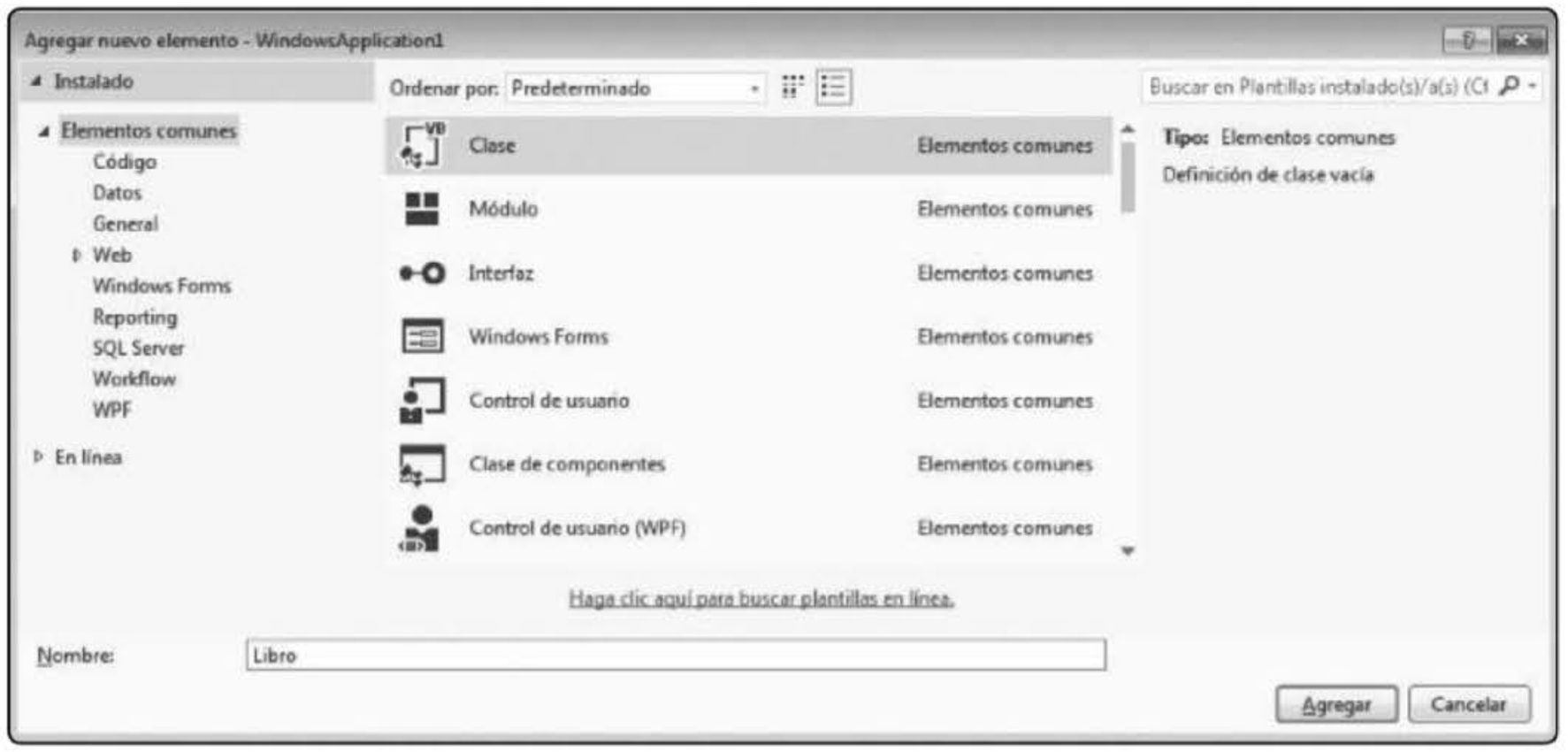
Clase
Atributo1: tipo Atributo2: tipo Atributo3: tipo Atributo4: tipo
Método1() Método2() Método3() Método4()

Hay que tener en cuenta que en Visual Studio 2015 existen muchas formas de crear una clase.

Por ejemplo, se necesita implementar la clase Libro en un proceso de negocio registro de nuevos libros en una biblioteca:

Primera forma: desde el menú de opciones.

- Dentro de una aplicación Windows Form de Visual Basic
- Seleccionar el menú **Proyecto > Agregar clase…**



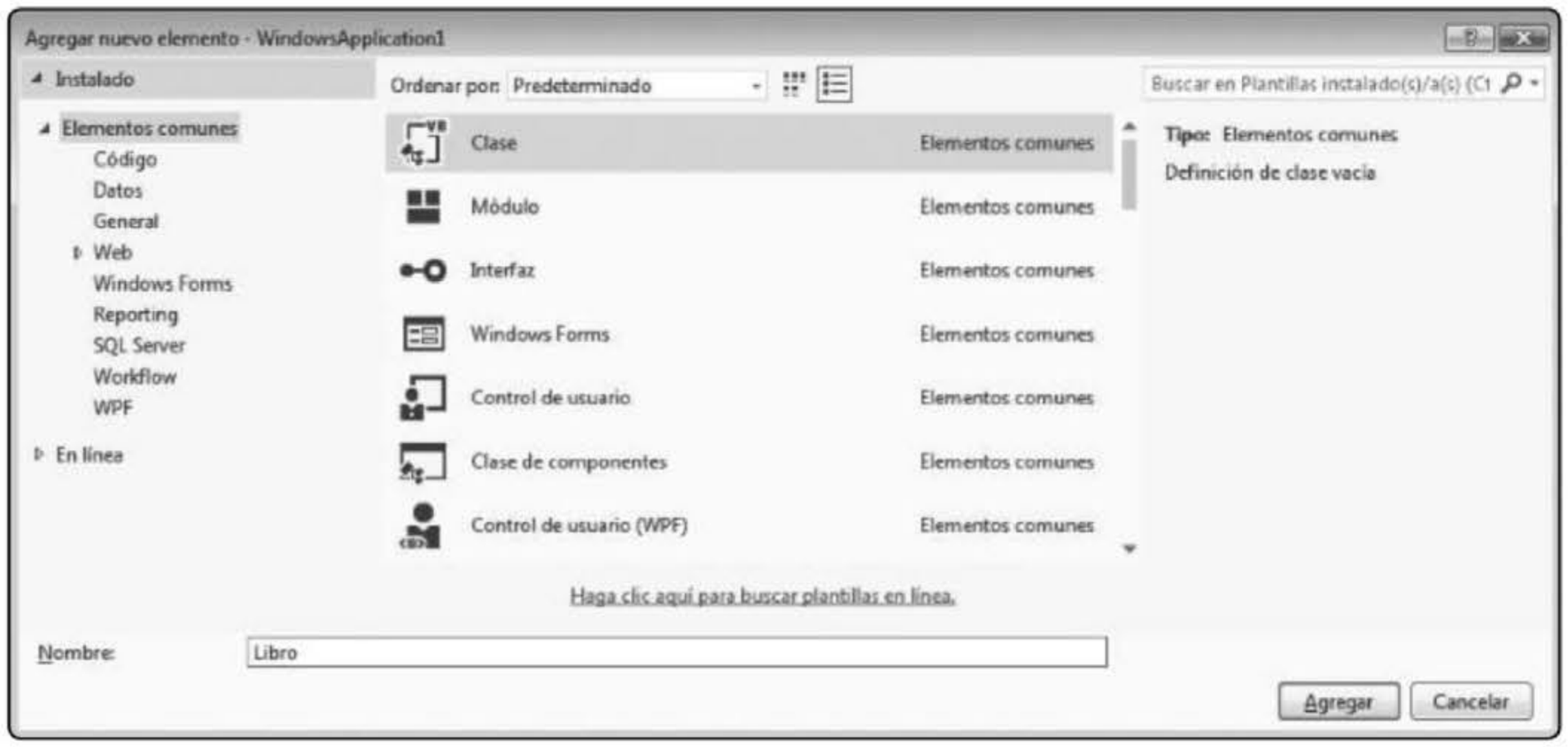
- De la lista de elementos asegurarse que se encuentre seleccionado la opción Clase, asignarle un nombre, por ejemplo, Libro, y luego agregarlo al proyecto con el botón **Agregar**. El código siguiente indica la implementación inicial de la clase:

```
Public Class Libro
End Class
```

Como se puede ver, la clase siempre nace pública y cuenta con la cláusula Class, que indica su denominación de clase y siempre finaliza con la sentencia End Class.

Segunda forma: Desde la ventana Explorador de soluciones.

- Dentro de una aplicación Windows Form de Visual Basic.
- Haga clic derecho sobre el proyecto desde la ventana Explorador de Soluciones.
- Seleccione **Agregar > Clase**

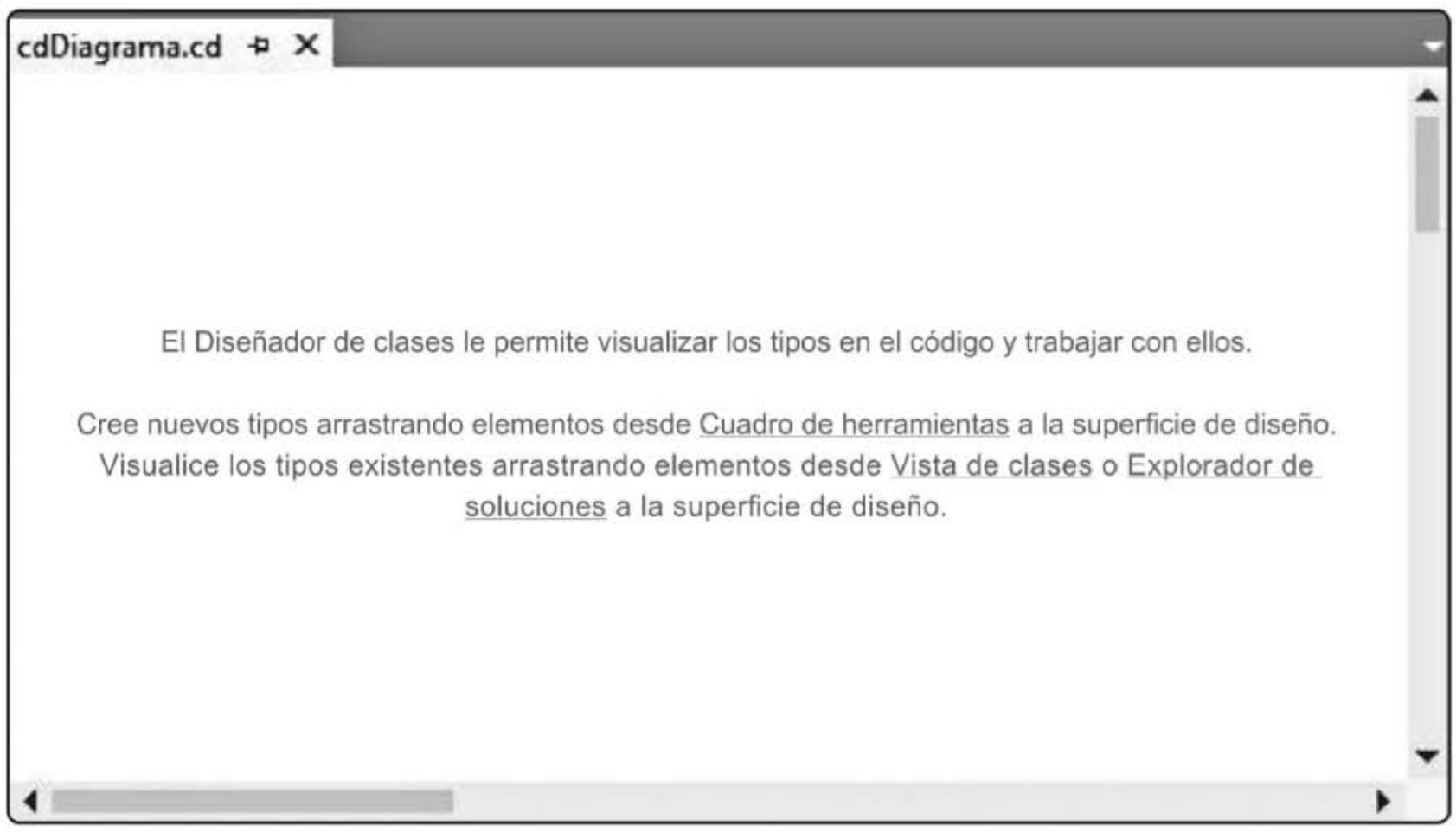


- De la misma manera asigne el nombre de la clase y presione el botón **Agregar**.

Tercera forma: mediante un diagrama de clase.

- Dentro de una aplicación Windows Form de Visual Basic.
- Haga clic derecho sobre el proyecto desde la ventana Explorador de soluciones.
- Seleccione **Agregar > Nuevo elemento**.
- Seleccione la categoría **General > Diagrama de clase**.

- Asigne un nombre al diagrama, por ejemplo, cdDiagrama y presione el botón **Agregar**.



- Aparece un entorno a partir del cual debe agregar la clase de la siguiente manera: haga clic derecho en el fondo del Diseñador de clase, seleccione **Agregar > Clase** y asigne un nombre, como se muestra en la siguiente imagen:

- Sobre la forma de controlar la clase mediante la implementación gráfica, pues, se debe entender que esta tercera forma solo es visual para la creación de la clase ya que, internamente, tiene código para llegar a él. Haga doble clic sobre la clase desde el diseñador.

Y el código mostrado es exactamente igual al de la primera forma, tal como se muestra en el siguiente código:

```
Public Class Libro
End Class
```

7.1.3 Nivel de visibilidad

Se le llama así a la forma de controlar el acceso a ciertos elementos implementados en una clase, ya que se debe administrar correctamente qué elementos serán públicos, privados o protegidos.

Para controlar el acceso a los miembros de una clase se usan especificadores o modificadores de acceso que se anteponen a las declaraciones de los miembros a controlar. Los especificadores de acceso se muestran en la siguiente tabla:

Visibilidad	Acceso a los miembros		
	Desde la misma clase	**Desde una clase heredada**	**Desde el exterior de la clase**
Public	Sí	Sí	Sí
Private	Sí		
Protected	Sí	Sí	
Sin especificación	Sí		

- Los elementos con visibilidad pública podrán ser usados por otras clases sin restricciones.
- Los elementos con visibilidad privada podrán ser accedidos solo en la clase donde se implementó.
- Los elementos con visibilidad protected solo podrán ser accedidos por las clases que implementan herencia.

Finalmente, si se implementan miembros sin especificación, estos solo podrán ser accedidos en la clase que lo implementó, igual que se especificara privado.

7.2 Los atributos de clase

Los atributos representan la característica que presenta una determinada clase, que deben definirse con un nivel de abstracción controlada por el usuario. Los atributos se encuentran asociados a clases y objetos, ya que es mediante ellos que se puede acceder a los miembros.

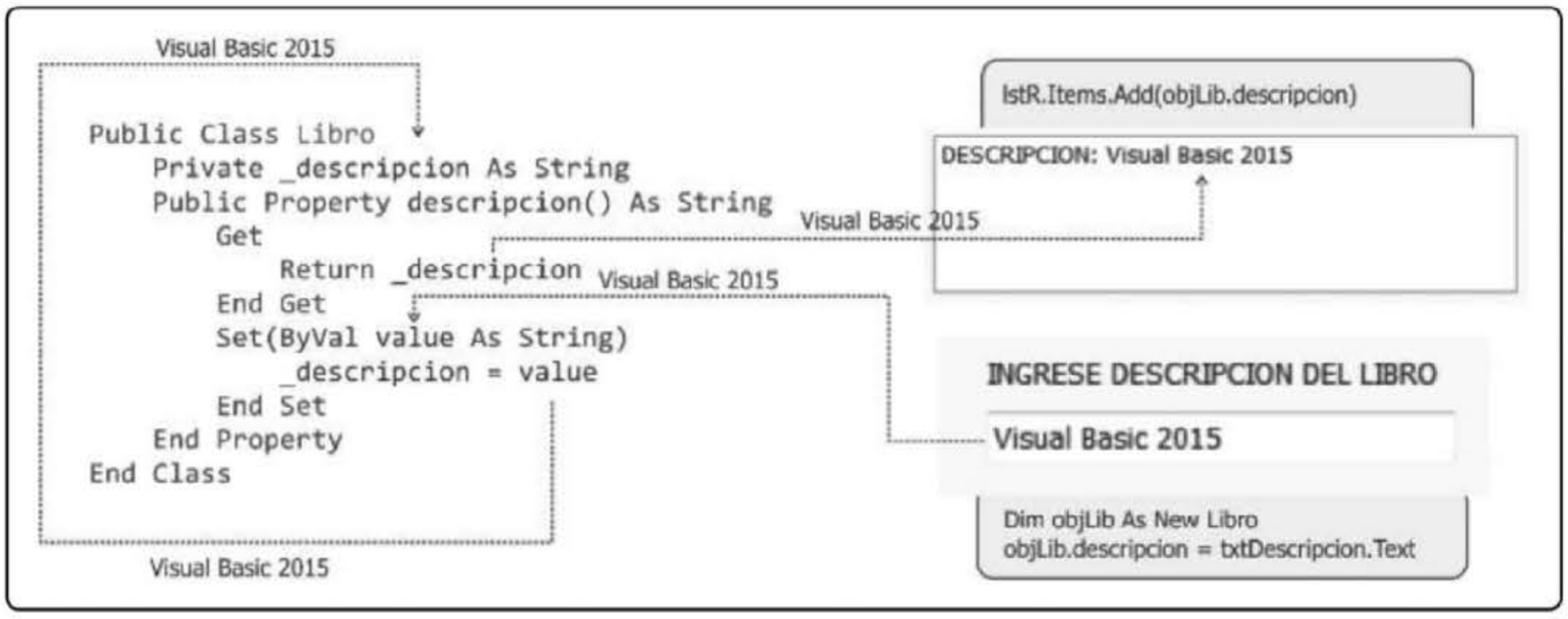
A continuación, la manera de agregar atributos en una clase implementada en Visual Studio 2015:

```
Public Class Libro
    Private _descripcion As String
    Public Property descripcion() As String
        Get
            Return _descripcion
        End Get
        Set(ByVal value As String)
            _descripcion = value
        End Set
    End Property
    Private _precio As String
    Public Property precio() As String
        Get
            Return _precio
        End Get
        Set(ByVal value As String)
            _precio = value
        End Set
    End Property
End Class
```

Se especifica un atributo privado, el cual podrá ser referenciado mediante un objeto por medio de los métodos Get y Set; la definición inicial se le debe a Set, ya que es el encargado de asignar un valor al atributo privado, mientras que Get envía el valor del atributo privado asignado por Set. Ver el siguiente caso:

Si existe la clase Pago y se necesita especificar un atributo llamado numPago de tipo entero, su implementación en la clase seria de la siguiente manera:

```
Public Class Libro
    Private _descripcion As String
    Public Property descripcion() As String
        Get
            Return _descripcion
        End Get
        Set(ByVal value As String)
            _descripcion = value
        End Set
    End Property
End Class
```



Al crear un objeto de la clase Libro se puede enviar el valor por medio de dicho objeto. Este valor es recibido por Set y asignado al atributo privado, para que cuando alguien solicite su valor Get lo devuelva tal como se muestra en la imagen.

7.3 Los objetos

Los objetos para la programación representan entidades del mundo real que se encuentran asociadas a una determinada clase, por medio de la cual se puede tener acceso a los atributos y métodos definidos dentro de la misma. Todo esto dependerá de la visibilidad que implemente a sus métodos o atributos la clase.

Si un objeto representa un ejemplar de una clase si se asume que el objeto pertenece a una clase, entonces dicho objeto tendrá acceso a todos los elementos que compone la clase.

7.3.1 Creación de un objeto:

```
Dim objeto As New Clase
```

La creación de un objeto es muy similar a la declaración de variables; se inicia por la cláusula Dim seguido por el nombre del objeto, el cual tiene la misma característica de las variables de no usar espacios en blanco, caracteres especiales o palabras reservadas, seguidas de la cláusula As New y, finalmente, el nombre de la clase.

Si se cuenta con la clase Libro, la forma de crear un objeto de la clase sería de la siguiente manera:

```
Dim objLib As New Libro
```

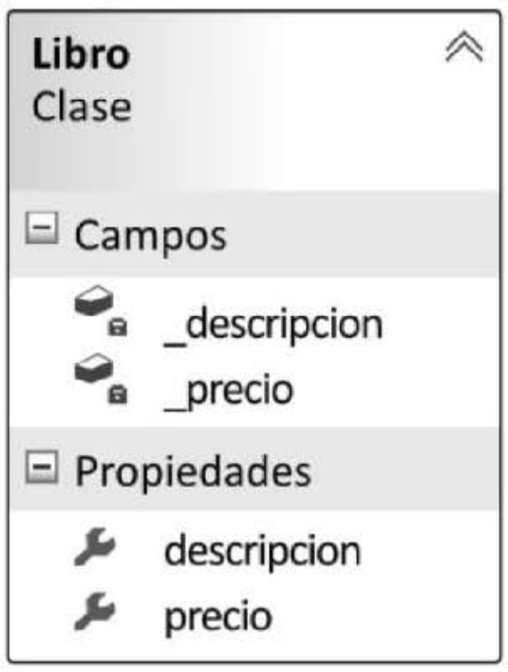
Donde objLib es el nombre del objeto y Libro es la especificación de la clase. Hay que tener en cuenta que se pueden crear N objetos de la misma clase sin ninguna restricción, como, por ejemplo:

```
Dim objLib1 As New Libro
Dim objLib2 As New Libro
Dim objLib3 As New Libro
Dim objLib4 As New Libro
```

7.3.2 Enviando información a los atributos de la clase por medio del objeto

Una vez creada la clase, se puede acceder a todos los miembros públicos declarados dentro de ella. Ver el caso de cómo asignar valores a sus atributos.

Si existe una clase Libro con la siguiente estructura de clase:



- **Enviando valores directos**: Los valores pueden ser asignados directamente a los atributos públicos de la clase Libro, todo esto mediante el objeto, como se muestra en el siguiente código:

```
Dim objLib As New Libro
objLib.descripcion = "Visual Basic 2015"
objLib.precio = 60.25
```

- **Enviando valores indirectos**: Los valores pueden ser asignados a variables locales y estas luego podrán ser asignadas a los atributos públicos de la clase Libro mediante el objeto, como se muestra en el siguiente código:

```
Dim descripción As String = "Visual Basic 2015"
Dim precio As Double= 60.25

Dim objLib As New Libro
objLib.descripcion = descripcion
objLib.precio = precio
```

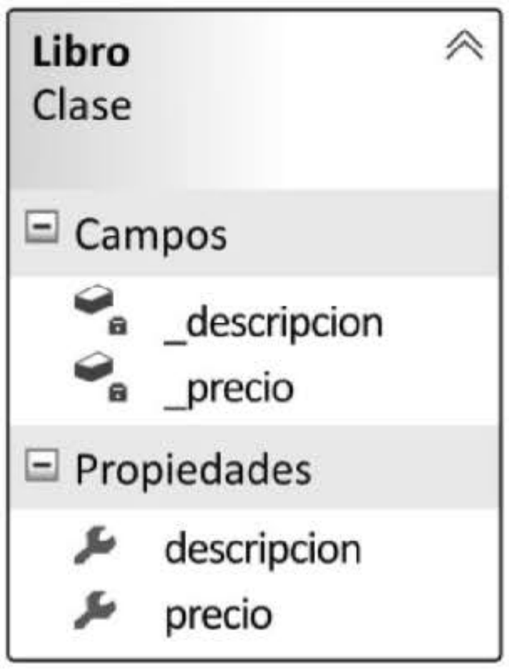
- **Enviando valores calculados**: No necesariamente puede enviarse valores directos, también pueden ser valores que provienen de algún calculo, eso dependerá mucho del proceso de negocio que se está implementando.

```
Dim descripcion As String = "Visual Basic 2015"

Dim objLib As New Libro
objLib.descripcion = descripcion
objLib.precio = 60.25 + (60.25 * 0.05)
```

7.3.3 Usando la información de los atributos de clase por medio del objeto

Una vez que los valores han llegado a la clase por medio de la propiedad Set, estos pueden ser recuperados para miles de acciones como realizar impresiones, cálculos, etc.

Si existe la clase Libro con la siguiente estructura de clase y se asume que la clase tiene valores asignados mediante un objeto:



- **Imprimir los valores directamente**: Es decir, se solicita el valor de la clase mediante el objeto.

```
Dim objLib As New Libro
objLib.descripcion = "Visual Basic 2015"
objLib.precio = 60.25

MessageBox.Show("La descripción del libro es: : "+objLib.descripcion)
lblDescripcion.Text = objLib.descripcion
lstSalida.Items.Add(objLib.precio.ToString("C"))
```

- **Imprimir los valores indirectamente**: Es decir, se solicita el valor de la clase mediante el objeto y se envía a variables. A partir de aquí se puede imprimirlos.

```
Dim objLib As New Libro
objLib.descripcion = "Visual Basic"
objLib.precio = 60.25
```

```
Dim descripción As String = objLib.descricion
Dim precio As Double= objLib.precio

MessageBox.Show("La descripción del libro es: : " & descripcion)
lblDescripcion.Text = descripcion
lstSalida.Items.Add(precio.ToString("C"))
```

- **Realizar cálculos**: Habitualmente se usan los valores de los atributos para realizar cálculos o usarlos dentro de las estructuras como If, For, While.

```
Dim objLib As New Libro
objLib.descripcion = "Visual Basic"
objLib.precio = 60.25

Dim subtotal As Double = objLib.precio * 10;
```

Caso desarrollado 1 Manejo de clase y objeto - Fábrica de vidrio

Objetivo	Controlar la venta de copas en un 100% producida por una fábrica de vidrio.
Referencias	Usar clase y objeto
Descripción del caso	Una fábrica de vidrio ha decidido producir dos tipos de copas que serán puestas a la venta según su presentación. La copa sencilla tendrá un precio de introducción de $100.00 y otras talladas a $250.00 por caja de 12 unidades cada una. Desarrollar una aplicación en Visual Basic 2015 que permita controlar las ventas de la copas.

Tipo de copa	Costo por caja
Copa sencilla	$100.00
Copa tallada	$250.00

El vendedor del área de ventas es el encargado del registro de venta, para lo cual selecciona un tipo de copa, registra la cantidad y debe mostrar el registro en una lista. Al final se debe mostrar un cuadro de resumen donde figure el monto acumulado por cada vendedor.

Tener en cuenta	• Implementar la clase Venta, la cual debe contar con los atributos vendedor, tipo de copa y cantidad solicitada para la venta. • Al iniciar la aplicación se debe mostrar la fecha y la hora de manera automática. • El cuadro combinado de los vendedores debe mostrar los nombres de los vendedores de forma automática al iniciar la aplicación. Debe considerarse el uso de enumeraciones para su definición.

	• El cuadro combinado de los tipos de copa debe mostrar los tipos de copas al iniciar la aplicación. Asimismo, se debe considerar que al seleccionar un tipo de copa se debe mostrar el precio según el tipo. • El botón **Cancelar** limpia los controles para un nuevo registro y el foco de ingreso debe ubicarse en el cuadro combinado del vendedor. • El botón **Registrar** crea un objeto de la clase Venta llamada objV, mediante el cual envía la información ingresada en el formulario y esta sea mostrada en un control ListView, además del precio y el monto neto de la venta implementada en funciones dentro de la clase Venta. Luego, se deberá calcular el monto acumulado de todos los netos (Total venta) y, finalmente, mostrar en un cuadro de resumen los montos acumulados según el nombre del vendedor en un control ListBox. • Finalmente, se debe implementar un método de validación de ingreso de datos que, al no seleccionar un vendedor, tipo de copa o ingresar la cantidad, emita un mensaje de error al usuario.
Formulario propuesto	

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015.
   - Seguir la ruta **Archivo > Nuevo proyecto.**
   - Seleccionar la plantilla Visual Basic.

- Seleccionar Aplicación de Windows Forms.
- Asignar el nombre pjVentaCopas.

2. Cambiar el nombre del formulario.
   - Seleccionar el formulario Form1.vb desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo asignar el nombre frmVentaCopa.

3. Diseñar la GUI (Interfaz Gráfica de Usuario).

4. Modificar las propiedades de los controles.

Control	Propiedad	Valor
Form1	(Name) Text	frmVentaCopa CONTROL DE VENTA
Label1	Text	CONTROL DE VENTAS - COPAS
Label2	Text	FECHA ACTUAL
Label3	Text	HORA ACTUAL
Label4	(Name) Text	lblFecha lblFecha
Label5	(Name) Text	lblHora lblHora
Label6	Text	VENDEDOR
Label7	Text	TIPO DE COPA

Label8	Text	CANTIDAD
Label9	Text	PRECIO $
Label10	Text	lblPrecio lblPrecio
Label11	Text	LISTADO DE VENTAS
Label12	Text	TOTAL VENTA
Label13	(Name) Text	lblTotal lblTotal
Label14	Text	RESUMEN
ComboBox1	(Name)	cboVendedor
ComboBox2	(Name)	cboTipo
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
ListBox1	(Name)	lstResumen
ListView1	(Name) GridLine Columns View	lvVenta True Agregue 4 columnas y defina sus cabeceras como se muestra en el modelo. Details
TextBox1	(Name)	txtCantidad

Código fuente de la clase Venta:

```
Public Class Venta
    'Implementación de los atributos
    Private _vendedor As String
    Public Property vendedor() As String
        Get
            Return _vendedor
        End Get
        Set(ByVal value As String)
            _vendedor = value
        End Set
    End Property
    Private _tipoCopa As String
    Public Property tipoCopa() As String
        Get
            Return _tipoCopa
        End Get
```

```
        Set(ByVal value As String)
            _tipoCopa = value
        End Set
    End Property
    Private _cantidad As String
    Public Property cantidad() As String
        Get
            Return _cantidad
        End Get
        Set(ByVal value As String)
            _cantidad = value
        End Set
    End Property

    'Implementación de los metodos
    'Método que asigna el precio según el tipo de copa
    Function asignaPrecio() As Double
        Dim precio As Double
        If tipoCopa = "Copa sencilla" Then
            precio = 100
        Else
            precio = 250
        End If
        Return precio
    End Function

    'Método para calcular el neto
    Function calculaNeto() As Double
        Return cantidad * asignaPrecio()
    End Function
End Class
```

Código fuente del formulario:

```
Public Class frmVentaCopa

    Private Sub frmVentaCopa_Load(…) Handles MyBase.Load
        Call llenaVendedor()
        Call LlenaTipoCopa()
        Call muestraFecha()
        Call muestraHora()
    End Sub

    Private Sub btnRegistrar_Click(…) Handles btnRegistrar.Click
        If valida() = "" Then
            Dim objv As New Venta
            objv.vendedor = cboVendedor.SelectedItem
            objv.tipoCopa = cboTipo.SelectedItem
            objv.cantidad = Integer.Parse(txtCantidad.Text)
```

```
        Dim precio As Double = objv.asignaPrecio
        Dim neto As Double = objv.calculaNeto

        Call imprimir(objv.vendedor, objv.tipoCopa, objv.cantidad, precio, neto)
        Call resumen()
        Call limpiarControles()
    Else
        MessageBox.Show("El error se encuentra en " & valida())
    End If
End Sub

Private Sub cboTipo_SelectedIndexChanged(…) Handles cboTipo.SelectedIndexChanged
    Dim objV As New Venta
    objV.tipoCopa = cboTipo.SelectedItem
    lblPrecio.Text = objV.asignaPrecio().ToString("C")
End Sub
Private Sub btnCancelar_Click(…) Handles btnCancelar.Click
    Call limpiarControles()
End Sub

'Metodo que llena los nombres de los vendedores al cuadro combinado
Sub llenaVendedor()
    cboVendedor.Text = "(Seleccione vendedor)"
    For Each v In System.Enum.GetNames(GetType(Vendedores))
        cboVendedor.Items.Add(v.ToString)
    Next
End Sub

'Metodo que llena los tipos de copas en el cuadro combinado
Sub LlenaTipoCopa()
    cboTipo.Text = "(Seleccione un producto)"
    With cboTipo.Items
        .Add("Copa sencilla")
        .Add("Copa tallada")
    End With
End Sub

'Metodo que muestra la fecha actual
Sub muestraFecha()
    lblFecha.Text = DateString
End Sub

'Metodo que muestra la hora actual
Sub muestraHora()
    lblHora.Text = TimeString
End Sub

'Metodo que permite imprimir los resultados
Sub imprimir(ByVal vendedor As String, ByVal tipoCopa As String,
             ByVal cantidad As Integer, ByVal precio As Double,
             ByVal neto As Double)
    Dim fila As New ListViewItem(vendedor)
```

```
        fila.SubItems.Add(tipoCopa)
        fila.SubItems.Add(cantidad.ToString("0.00"))
        fila.SubItems.Add(precio.ToString("0.00"))
        fila.SubItems.Add(neto.ToString("0.00"))
        lvVenta.Items.Add(fila)
        Call calculaTotalVenta()
        Call resumen()
    End Sub

    'Metodo que calcula el acumulado de netos
    Sub calculaTotalVenta()
        Dim total As Double
        For i = 0 To lvVenta.Items.Count - 1
            total += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
        Next
        lblTotal.Text = total.ToString("C")
    End Sub

    'Metodo que muestra el resumen de la venta
Sub resumen()
   lstResumen.Items.Clear()
   Dim totalV1 = 0, totalV2 = 0, totalV3 = 0, totalV4 = 0
   For i = 0 To lvVenta.Items.Count - 1
   If lvVenta.Items.Item(i).SubItems(0).Text = Vendedores.Angela_Torres.ToString Then
       totalV1 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    ElseIf lvVenta.Items.Item(i).SubItems(0).Text=
                                        Vendedores.Fernanda_Torres.ToString Then
       totalV2 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    ElseIf lvVenta.Items.Item(i).SubItems(0).Text =
                                          Vendedores.Luz_Lazaro.ToString Then
       totalV3 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
    ElseIf lvVenta.Items.Item(i).SubItems(0).Text =
                                        Vendedores.Manuel_Torres.ToString Then
       totalV4 += Double.Parse(lvVenta.Items.Item(i).SubItems(4).Text)
     End If
Next
lstResumen.Items.Add("Vendedor: " & Vendedores.Angela_Torres.ToString &
                     " tiene una venta de: " & totalV1.ToString("C"))
lstResumen.Items.Add("Vendedor: " & Vendedores.Fernanda_Torres.ToString &
                     " tiene una venta de: " & totalV2.ToString("C"))
 lstResumen.Items.Add("Vendedor: " & Vendedores.Luz_Lazaro.ToString &
                     " tiene una venta de: " & totalV3.ToString("C"))
  lstResumen.Items.Add("Vendedor: " & Vendedores.Manuel_Torres.ToString &
                     " tiene una venta de: " & totalV4.ToString("C"))
End Sub

    'Metodo que valida los valores ingresados en el controles
    Function valida() As String
        If cboVendedor.SelectedIndex = -1 Then
            cboVendedor.Focus()
            Return "nombre del vendedor"
        ElseIf cboTipo.SelectedIndex = -1 Then
```

```
            cboTipo.Focus()
            Return "tipo de copa"
        ElseIf String.IsNullOrWhiteSpace(txtCantidad.Text) Or
               Not IsNumeric(txtCantidad.Text) Then
            txtCantidad.Clear()
            txtCantidad.Focus()
            Return "cantidad solicitada"
        Else
            Return ""
        End If
    End Function

    'Metodo que permite limpiar todos los controles
    Sub limpiarControles()
        cboVendedor.SelectedIndex = -1
        cboTipo.SelectedIndex = -1
        cboVendedor.Text = "(Seleccione vendedor)"
        cboTipo.Text = "(Seleccione un producto)"
        txtCantidad.Clear()
        cboVendedor.Focus()
    End Sub
End Class

'Definición de la enumeración de vendedores
Enum Vendedores As Integer
    Fernanda_Torres
    Angela_Torres
    Manuel_Torres
    Luz_Lazaro
End Enum
```

Referencias bibliográficas

Johanes, L. (1996). *Fundamentos de programación. Algoritmos y estructura de datos*. Madrid: McGraw-Hill.

Rodríguez, M. (1991). *Metodologia de la programación. A través de pseudocódigo.* Madrid: McGraw-Hill.

Webs:

<https://msdn.microsoft.com/en-us/library/dd831853.aspx>

<https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx>

<https://msdn.microsoft.com/en-us/library/2x7h1hfk.aspx>

Impreso en los talleres gráficos de

EDITORIAL MACRO®

Surquillo

Mayo 2017