EDITORIAL MACRO®

Incluye versión digital + código de descuento

DESARROLLO DE APLICACIONES CON VISUAL C# 2015

Manuel Torres Remon

– Fundamentos de programación – Estructuras condicionales – Estructuras repetitivas
– Programación modular – Programación orientada a objetos

DESARROLLO DE APLICACIONES CON

VISUAL C#

2015



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

EDITORIAL MACRO®

Desarrollo de aplicaciones con Visual C#
Autor: Manuel Torres Remon

© Derechos de autor registrados:
Empresa Editora Macro EIRL

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

Jefe de edición:
Cynthia Arestegui Baca

Coordinación de edición:
Magaly Ramon Quiroz

Diseño de portada:
Rudy Herrera

Corrección de estilo:
Yadira Cabello Villanueva

Diagramación:
Lucero Monzón Morán
José Eduardo Siesquén Aquije

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: enero de 2016
Tiraje: 1000 ejemplares

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

ISBN N.° 978-612-304-325-4
Hecho el depósito legal en la Biblioteca Nacional del Perú N.° 2015- 19002
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

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

Biografía

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 varias capacitaciones en las instituciones más importantes de Lima, Perú.

Recibió su formación 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 lo puede demostrar ampliamente en las diferentes instituciones que labora.

Actualmente, se desempeña como docente de tecnología en los institutos Manuel Arévalo Cáceres, Cibertec y Unimaster en donde imparte los cursos de tecnología, especialmente en las materias de Programación, Base de Datos y Análisis de Sistemas.

Ha publicado libros como *Programación orientada a objetos con Visual Basic 2012, Programación Transact con SQL Server 2012*, *Desarrollo de aplicaciones con JAVA*, *JCreator*, *JDeveloper NetBeans*, *Fundamentos de programación con Visual Basic 2012*, *Desarrollo de aplicaciones web con PHP*, *Diseño web con HTML5 y CSS3*, *Aplicaciones VBA con Excel* y *Guía práctica Macros, aplicaciones VBA con Excel*.

Agradecimientos

Agradecimientos

Todo libro escrito es un sacrificio que tiene que ser valorado y esto solo se dará si usted aprecia este material, tal como lo aprecio yo mismo. Escuché alguna vez la frase «La lectura de un libro enriquece de conocimientos y empobrece la ignorancia»; pero, hoy en día, un libro impreso es difícil de leer pues son pocas las personas que cargan un libro en la mano; pues han sido reemplazados por los dispositivos móviles, así es que mi primer agradecimiento es para usted, amigo lector, por adquirir este material que me ha hecho sacrificar tiempo y esfuerzo.

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

Finalmente, agradezco enormemente a las personas sacrificadas como lo son mis pequeñas hijas Ángela Victoria y Fernanda Ximena Torres Lázaro, a mi esposa Luz, y, finalmente, a mi querida suegra Victoria Menor Bautista que siempre tiene un comentario bueno sobre mi persona y eso me motiva de gran manera a seguir adelante.

Índice

Índice

Introducción 11

CAPÍTULO 1. Introducción a los algoritmos

- Introducción 15

1.1 Algoritmo 16
1.2 Características de los algoritmos 21
1.3 Lenguajes de programación 21
1.4 Casos desarrollados 22
Caso desarrollado 1: promedios de nota 22
Caso desarrollado 2: conversión de medidas 24
Caso desarrollado 3: capital de empresa 26
Caso desarrollado 4: control de venta de productos 28
Preguntas frecuentes *30*
Para recordar *32*

CAPÍTULO 2. Introducción al Visual Studio - Proceso de instalación

- Introducción 35

2.1 Plataforma .Net 37
2.2 .NET Framework 4.6 40
2.3 Principios del Common Language Runtime 42
2.4 Biblioteca de clases .NET 44
2.5 Versiones de la plataforma .NET 45
2.6 Proceso de instalación del Visual Studio 45
2.7 Barra de herramientas estándar 48
2.8 Cuadro de herramientas 49
2.9 Explorador de soluciones 50
2.10 Ventana de propiedades 52
2.11 Soluciones para C# 53
2.12 Proyectos para C# 57
2.13 Windows Forms 58
2.14 Controles visuales 62
2.15 Notación para los nombres de los controles 62
2.16 Control Form 64

2.17 Control Label 66
2.18 Control TextBox 68
2.19 Control Button 70
2.20 Control ComboBox 71
2.21 Control ListBox 72
2.22 Control ListView 73
Preguntas frecuentes *75*
Para recordar *76*

CAPÍTULO 3. Las variables y los operadores

▪ Introducción 81
3.1 Estructura de una aplicación de plataforma en Visual C# 81
3.2 Eventos de los controles 82
3.3 Comentarios en el código fuente 86
3.4 Tipos de datos para Visual C# 88
3.5 Identificadores 90
3.6 Palabras reservadas 91
3.7 Literales 91
3.8 Las variables 92
3.9 Ámbito de las variables en C# 93
3.10 Operadores 96
3.11 Prioridad de los operadores aritméticos 97
3.12 Métodos matemáticos de .NET Framework 98
3.13 Conversiones de tipos en C# 99
3.14 Conversión implícita 99
3.15 Conversión explícita 100
3.16 Conversión definida por el usuario 100
3.17 Conversión mediante el método parse 101
3.18 Conversión mediante la clase Convert 101
3.19 Conversión implícita de tipos 102
3.20 Clase String 103
3.21 La clase MessageBox 105
3.22 Casos desarrollados 107
Caso desarrollado 1: pago de empleados 107
Caso desarrollado 2: control de medidas 112
Caso desarrollado 3: control de área y volumen 116
Para recordar *120*
Preguntas frecuentes *121*

CAPÍTULO 4. Estructuras de decisión

- Introducción 125

4.1 Operadores de relación 126
4.2 Operadores lógicos 128
4.3 Instrucción if simple 128
4.4 Instrucción if doble 131
4.5 Instrucción if doblemente enlazada 133
4.6 Instrucción condicional múltiple 137
4.7 Casos desarrollados 140
Caso desarrollado 1: if simple - pago de empleados 140
Caso desarrollado 2: if simple - control de registro de documentos 148
Caso desarrollado 3: if doble - promedio de notas 154
Caso desarrollado 4: if doble - venta de productos 161
Caso desarrollado 5: if doblemente enlazado - control de registro de multas de transito ... 169
Caso desarrollado 6: if doblemente enlazado - registro de postulantes 175
Caso desarrollado 7: switch - control de registro de llamadas 181
Caso desarrollado 8: switch - control de registro de estacionamiento 188

Para recordar 194

Preguntas frecuentes 195

CAPÍTULO 5. Estructuras de repetición

- Introducción 199

5.1 Los contadores 200
5.2 Operaciones sobre los contadores 202
5.3 Los acumuladores 203
5.4 Operaciones sobre los acumuladores 204
5.5 Bucle While 205
5.6 Bucle do while 208
5.7 Bucle For 210
5.8 Casos desarrollados 212
Caso desarrollado 1: contadores - control de registro de participantes 212
Caso desarrollado 2: acumuladores - planilla de sueldos 221
Caso desarrollado 3: for - seguros de vida 228
Caso desarrollado 4: while - control de facturas 235
Caso desarrollado 5: do while - control de registro alquiler de habitaciones 242
Caso desarrollado 6: integración - control de venta de boletos 250

Para recordar 257

Preguntas frecuentes 257

CAPÍTULO 6. Programación modular

- Introducción.....261

6.1 Ámbito de las variables.....262
6.2 Programación modular.....263
6.3 Método sin valor de retorno sin parámatros.....263
6.4 Método sin valor de retorno con parámetros.....266
6.5 Método con valor de retorno sin parámetros.....268
6.6 Método con valor de retorno con parámetros.....269
6.7 Expresiones lambda.....271
6.8 Casos desarrollados.....274
Caso desarrollado 1: control de registro de llamadas - método sin valor de retorno sin parámetros.....274
Caso desarrollado 2: control de registro de llamadas - método sin valor de retorno con parámetros.....287
Caso desarrollado 3: control de registro de llamadas - método con valor de retorno sin parámetros.....292
Caso desarrollado 4: control de registro de llamadas - método con valor de retorno con parámetros.....298
Caso desarrollado 5: integración de métodos - control de pago de empleados.....303
Caso desarrollado 6: lambda - control de registro de libros.....317
Para recordar.....*325*
Preguntas frecuentes.....*326*

CAPÍTULO 7. Programación orientada a objetos

- Introducción.....329

7.1 Características de la programación orientada a objetos.....330
7.2 La clase.....333
7.3 Nivel de visibilidad.....337
7.4 Los atributos de clase.....338
7.5 Los objetos.....340
7.5.1 Creación de un objeto.....341
7.5.2 Enviando información a los atributos de la clase por medio del objeto.....341
7.5.3 Usando la información de los atributos de clase por medio del objeto.....343
7.6 Método constructor.....344
7.7 Casos desarrollados.....345
Caso desarrollado 1: clase objeto - boleta de venta.....345
Caso desarrollado 2: clase objeto - planilla de empleados.....357

Bibliografía.....365

CAPÍTULO 6. Programación modular

- Introducción.....261

6.1 Ámbito de las variables.....262
6.2 Programación modular.....263
6.3 Método sin valor de retorno sin parámatros.....263
6.4 Método sin valor de retorno con parámetros.....266
6.5 Método con valor de retorno sin parámetros.....268
6.6 Método con valor de retorno con parámetros.....269
6.7 Expresiones lambda.....271
6.8 Casos desarrollados.....274
Caso desarrollado 1: control de registro de llamadas - método sin valor de retorno sin parámetros.....274
Caso desarrollado 2: control de registro de llamadas - método sin valor de retorno con parámetros.....287
Caso desarrollado 3: control de registro de llamadas - método con valor de retorno sin parámetros.....292
Caso desarrollado 4: control de registro de llamadas - método con valor de retorno con parámetros.....298
Caso desarrollado 5: integración de métodos - control de pago de empleados.....303
Caso desarrollado 6: lambda - control de registro de libros.....317
Para recordar.....*325*
Preguntas frecuentes.....*326*

CAPÍTULO 7. Programación orientada a objetos

- Introducción.....329

7.1 Características de la programación orientada a objetos.....330
7.2 La clase.....333
7.3 Nivel de visibilidad.....337
7.4 Los atributos de clase.....338
7.5 Los objetos.....340
7.5.1 Creación de un objeto.....341
7.5.2 Enviando información a los atributos de la clase por medio del objeto.....341
7.5.3 Usando la información de los atributos de clase por medio del objeto.....343
7.6 Método constructor.....344
7.7 Casos desarrollados.....345
Caso desarrollado 1: clase objeto - boleta de venta.....345
Caso desarrollado 2: clase objeto - planilla de empleados.....357

Bibliografía.....365

Introducción

El libro *Desarrollo de aplicaciones con Visual C#* le permitirá crear aplicaciones de plataforma usando como plataforma de trabajo el Visual Studio 2015.

Para cumplir con dicho objetivo, se dividió el libro en siete capítulos que son distribuidos de la siguiente manera: en el primer capítulo, se presenta una breve introducción a los algoritmos y se interpretan los mismos de manera correcta.

En el capítulo dos, se hace una introducción al Visual Studio 2015, así también, se describen el proceso de instalación y los primeros pasos en el entorno. En el capítulo tres, se hace referencia al uso de los operadores y variables en Visual C# y se explica su uso con casos desarrollados.

En el capítulo cuatro, se explica detalladamente el uso de las estructuras de decisión tanto simples, dobles, doblemente enlazadas y múltiples con casos desarrollados. En el capítulo cinco, se hace referencia al uso de las estructuras repetitivas como for, while y do while; como una media importante en el desarrollo de aplicaciones.

En el capítulo seis, se implementan aplicaciones usando la programación modular y un solo ejercicio desarrollado será expresado en cuatro formas de trabajo usando el método sin valor de retorno sin parámetros, el método sin valor de retorno con parámetro, el método con valor de retorno sin parámetro y el método con valor de retorno con parámetro.

Finalmente, en el capítulo siete, se hace uso de la programación orientada a objetos haciendo que las aplicaciones usen clase objeto para la solución de su proceso de negocio.

El libro está dirigido a personas que requieren conocer los conceptos básicos de la programación en C# mediante el uso de Visual Studio 2015. No es necesario tener conocimientos previos en el desarrollo de aplicaciones, solo muchas ganas de aprender el lenguaje.

Estudiantes de computación e informática, ingeniería de sistemas y profesionales de toda especialidad podrán desarrollar aplicaciones de forma guiada mediante los casos desarrollados escritos en este material.

Capítulo 1

Introducción a los algoritmos

Capacidad terminal:

Conocer la terminología usada en la solución de problemas usando algoritmos, así como las etapas para la resolución de problemas usando aplicaciones tipo consola con Visual C#.

Contenido:

- Introducción
- Algoritmo
- Características de los algoritmos
- Lenguajes de programación
- Casos desarrollados
    - o Caso desarrollado 1: promedio de notas
    - o Caso desarrollado 2: conversión de medidas
    - o Caso desarrollado 3: capital de empresa
    - o Caso desarrollado 4: control de venta de productos
- Preguntas frecuentes
- Para recordar

Introducción

La computadora, en la actualidad, es considerada una herramienta indispensable, ya que permite realizar muchas actividades en corto tiempo y de manera eficaz. Técnicamente, es considerada como un dispositivo electrónico capaz de ejecutar instrucciones o sentencias solicitadas por un determinado usuario.

Ha pasado de ser un simple aparato electrónico que permitía registrar información o realizar cálculos complejos a convertirse en un dispositivo indispensable para la humanidad. Permite desde redactar una carta simple hasta controlar una operación médica compleja.

Entonces, se puede decir que la computadora, en la actualidad, es considerada una herramienta de trabajo indispensable en todo sector; ya que permite optimizar y agilizar todo tipo de tarea. También es usada como entretenimiento, ya que permite ejecutar juegos tanto de plataforma como online, así como visualizar videos, escuchar música, entre otros que seguirán entreteniendo. Asimismo, se puede decir que es considerada un medio de comunicación, ya que gracias al correo electrónico, redes sociales, entre otros, nos podemos comunicar con el mundo. Finalmente, puede ser considerada como el más grande repositorio de información que existe, dado que gracias a la Internet podemos tener acceso a cientos de miles de información.

Por estas razones, se puede decir que las computadoras y la tecnología han evolucionado paralelamente de manera rápida por no decir abrumadora, hasta podemos decir que en algún momento será difícil manejar tanta información, es notable que nos está simplificando muchas de nuestras tareas. El uso de aparatos móviles, tabletas y la computadora misma será de vital importancia para nuestro futuro.

Finalmente, se concluye que la computadora resulta ser una herramienta importante en el proceso de solución de nuestro problema. Para poder llegar a solucionar problemas debemos comportarnos como programadores ya no como un usuario más de la computadora. He aquí el objetivo de este libro, involucrarlo a usted en el mundo de la programación mediante el lenguaje C#.

1.1 Algoritmo

Es considerado como un conjunto de métodos que permiten resolver un problema particular. Mediante una serie de pasos ordenados, bien definidos y especialmente finitos, describe el proceso que se sigue para dar solución a medida.

La palabra «algoritmo» deriva de la traducción al latín de la palabra árabe *alkhowarismi*, que hace referencia a un reconocido matemático, astrónomo y geógrafo árabe Al-Khwarizmi considerado como el padre del algoritmo junto al matemático griego Euclides que inventó un método para determinar el máximo común divisor de dos números.

Los algoritmos tienen por finalidad solucionar problemas. Un problema es considerado como una cuestión que se plantea para hallar un dato desconocido a partir de información conocida, es aquí donde el programador debe analizar el concepto, ya que es el encargado capaz de determinar con qué datos se cuenta para encontrar el nuevo valor que normalmente se denomina resultado.

Para dar solución a un determinado problema hay que enfocarse en el diseño correcto de un algoritmo. Véase el siguiente enfoque:



Un problema debe ser claro y expresar lo que realmente se quiere como objetivo, ya que de otra manera no se podrá diseñar el algoritmo y, en consecuencia, no se podrá implementar en un programa, que se realiza usando el lenguaje de programación C#.

A continuación, se presenta un caso completo que refleja el enfoque de solución al problema usando Visual C# 2015 como entorno de trabajo.

Problema:

> Una tienda de abarrotes tiene problemas para controlar la venta de productos. El proceso inicia cuando un cliente solicita un determinado producto, el vendedor verifica el precio del producto y, de acuerdo a la cantidad seleccionada, calcula el monto que se debe cancelar. Se pide diseñar un algoritmo que determine el monto a pagar por la compra de cierta cantidad de un producto.

Algoritmo:

```
Inicio
 Leer precio, cantidad
 monto ← precio x cantidad
 Escribir monto
Fin
```

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Ejercicio01
{
 class Program
 {
   static void Main(string[] args)
   {
     Console.WriteLine(" ** CALCULADORA DE MONTOS ** ");
     Console.WriteLine(" ");

     double precio;
     int cantidad;

     Console.Write("Ingrese el precio del producto $: ");
     precio = double.Parse(Console.ReadLine());
     Console.Write("Ingrese la cantidad seleccionada: ");
     cantidad = int.Parse(Console.ReadLine());

     double monto = precio*cantidad;

     Console.WriteLine("---------------------------------------");
     Console.WriteLine("MONTO A CANCELAR $: "+monto.ToString("0.00"));

     Console.ReadKey();
    }
  }
 }
```

Resultado:

```
 ** CALCULADORA DE MONTOS **

Ingrese el precio del producto $: 7.50
Ingrese la cantidad seleccionada: 10
---------------------------------------
MONTO A CANCELAR $: 75.00
```

Ahora, se analizará la solución presentada:

En el problema:
Aquí, se debe determinar qué valor se requiere como resultado, en este caso es el monto. Para ello, se necesitan ciertos valores como el precio del producto y la cantidad comprada por el usuario. Por lo tanto, antes de diseñar el algoritmo, se deben identificar dichos valores. Esto se realiza mediante las siguientes cuestiones:

- **¿Cuál es el resultado esperado del problema?**, la respuesta a esta pregunta serán los valores resultantes del problema; también es considerado como valores de salida.

- **¿Qué datos se necesitan para obtener el resultado esperado?**, la respuesta que se proponga a esta pregunta permitirá determinar los valores necesarios para dar solución al problema; también es considerado como valores de entrada.

E/S	Valores	Descripción
Entrada	Precio	Representa el precio del producto seleccionado por el cliente.
	Cantidad	Representa la cantidad de productos seleccionados por el cliente.
Salida	Monto	Representa el resultado esperado, el cual resulta de multiplicar el precio y la cantidad.

En el algoritmo:

Un algoritmo puede representarse de dos maneras: la primera, mediante un texto simple llamado pseudocódigo, y la otra, mediante un gráfico llamado diagrama de flujo. Ambas técnicas son iguales, lo que varía es la forma; ya que el contenido sigue siendo el mismo. Para este caso, se usarán los pseudocódigos por ser muy parecidos al código C#, el principal objetivo. De igual manera, se mostrarán ambas soluciones:

Pseudocódigo	Diagrama de flujo
Inicio **Leer** precio, cantidad monto ← precio x cantidad **Escribir** monto **Fin**	

La característica principal de los pseudocódigos es la simplicidad al presentar los valores tanto de entrada como de salida; pero, como se indicó antes, estos valores deben ser detectados correctamente. De otro modo, el pseudocódigo cae en anomalías, de igual manera, el diagrama de flujo y, peor aún, el código en C#.

Hay algunas palabras dentro del pseudocódigo que pueden ser tomadas como un estándar. Por ejemplo, para representar los valores de entrada, se usa la sentencia LEER o CAPTURAR, mientras que para los valores de salida, usa la sentencia ESCRIBIR o IMPRIMIR. Punto aparte es el trato de las operaciones de cálculo, los cuales se representan de la siguiente manera:

Resultado ← Expresión

Resultado guardará el valor obtenido en una determinada expresión, por ejemplo:

Sentencia	Descripción
n ← 0	El valor cero es asignado a la variable n.
b ← a	El valor que tenga "A" será asignado a "B".
x ← (a+b)/2	X registra el valor obtenido de la expresión (a+b)/2.

En el programa:

Los códigos mostrados fueron realizados en una aplicación de consola desde Visual Studio 2015. Para crear una aplicación de consola, siga los siguientes pasos:

1. Diríjase a **Archivo > Nuevo > Proyecto**.
2. Seleccione la plantilla **Visual C# > Aplicación de consola**.
3. Asigne un nombre y una ubicación a la aplicación.
4. Visual C# muestra un código inicial, tal como se muestra a continuación:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Ejercicio01
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Aquí, agregue el código. Ahora, se explicará brevemente cada línea de sentencia implementada:

Sentencia	Descripción
`Console.WriteLine("**CALCULADORA DE MONTOS**");` `Console.WriteLine(" ");`	La primera sentencia permite mostrar el mensaje "Calculadora de montos". La segunda línea muestra una línea en blanco. Ambas sentencias hacen cambio de línea de forma automática.
`double precio;`	Declarando la variable precio del tipo double por tratarse de un valor decimal.
`int cantidad;`	Declarando la variable cantidad del tipo int por ser un valor entero.
`Console.Write("Ingrese el precio del producto $: ");`	Mostrar el mensaje "Ingrese el precio del producto"; solo debe tener en cuenta que la sentencia no permite el cambio de línea, es decir: Write: Muestra el mensaje. WriteLine: Muestra el mensaje pero cambiando de línea.
`precio = double.Parse(Console.ReadLine());`	Capturando el precio del producto ingresando y asignándolo a la variable precio, previamente declarado.
`Console.Write("Ingrese la cantidad seleccionada: ");`	Mostrar el mensaje "Ingrese la cantidad seleccionada".
`cantidad = int.Parse(Console.ReadLine());`	Capturando la cantidad registrada y enviándola a la variable cantidad, previamente declarada.
`double monto = precio*cantidad;`	Calculando el monto de la venta, para lo cual se multiplica el precio y la cantidad y el resultado del producto es asignado a la variable monto, declarado en la misma línea.
`Console.WriteLine("-------------------------");`	Mostrando una línea de separación en la impresión de los resultados.
`Console.WriteLine("MONTO A CANCELAR $: "` `+monto.ToString("0.00"));`	Mostrando el resultado de la aplicación empezando con el mensaje "Monto a cancelar $". Seguidamente, se hace la impresión del monto, pero como dicho monto es un valor double, se debe convertir a cadena mediante la función ToString. El "0.00" es para imprimir a dos decimales el resultado.

5. Finalmente, ejecute el programa con la tecla **F5** y agregue los valores de entrada.

1.2 Características de los algoritmos

Para que un algoritmo sea considerado como tal, debe cumplir ciertos aspectos mencionados a continuación:

- Todo algoritmo tiene un inicio y un fin; esto permitirá determinar los límites de la aplicación.
- Un algoritmo debe ser preciso, al probar los resultados siempre debe dar los esperados por el usuario.
- Un algoritmo debe tener un orden de realización de pasos.
- Un algoritmo debe ser legible, es decir, lo descrito debe ser claro de forma que pueda entenderse fácilmente.
- Todo algoritmo tiene tres partes importantes: entrada, proceso y salida. La entrada representa los valores necesarios para la solución del problema; el proceso, la fórmula de solución y la salida, el resultado de la aplicación.



1.3 Lenguajes de programación

Se debe recordar que nuestro objetivo es implementar una aplicación C# en Visual Studio; para lograrlo se pasa el diseño del algoritmo en un programa de computadora, para lo cual se usa un lenguaje de programación que permita interpretar los códigos de manera que los usuarios puedan usar y reutilizar la aplicación.

C# es considerado un nuevo lenguaje de programación diseñado para crear un conjunto de aplicaciones que se ejecutan en la plataforma de trabajo .NET Framework. Es reconocido como una evolución de lenguaje C y C++; además de ser sencillo de codificar, moderno y principalmente está orientado a objetos.

El código desarrollado mediante el lenguaje C# se compila como código administrado, es decir, todo código usa el Common Language Runtime; el cual es considerado como un conjunto de servicios que incluyen inteoperatibilidad entre aplicaciones; al respecto se debe entender que dentro del código C# se pueden usar sentencias, por ejemplo, de Basic.

Finalmente, se debe estar preguntando por qué elegir C# como nuestro lenguaje para desarrollar aplicaciones. La respuesta está en los principales beneficios que se encuentran en este lenguaje como la capacidad orientada a objetos, el cual le permite crear todo tipo de aplicaciones. También se puede mencionar que proporciona la capacidad de generar componentes de sistema duraderos en virtud de las siguientes características:

- Gran robustez, gracias a la recolección de elementos no utilizados, es decir, recicla las variables declaradas pero no usadas, liberando así memoria de la computadora.
- Interacción entre otros lenguajes de programación, es decir, entre plataformas distintas.
- Interoperabilidad por medio de los servicios de COM y .NET Framework con un acceso limitado a los controles basado estrictamente en bibliotecas, esto es, toda aplicación desarrollado en Visual C# deberá contener librerías necesarias para la ejecución de la aplicación.
- Acceso a datos XML mediante clases basadas en tecnología Web; esto permite enviar información entre aplicaciones de otros lenguajes.

1.4 Casos desarrollados

Caso desarrollado 1 : promedios de nota

Implemente una aplicación de consola en Visual C# 2015 que permita dar solución al siguiente caso:

Problema:

Un docente necesita tener el control de los promedios de notas de un grupo determinado de alumnos, para lo cual necesita una aplicación que dada tres notas, permita resolver el promedio de los mismos.

Determinando las entradas y salidas:

Descripción	Tipo	Nombre de la variable
Nombre del alumno	Entrada	Nombre
Tres notas	Entrada	Nota1, Nota2, Nota3
Promedio de notas	Salida	Promedio

Algoritmo:

```
Inicio
 Leer alumno, nota1, nota2, nota3
 promedio ← (nota1+ nota2+ nota3)/3
 Escribir promedio
Fin
```

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ejercicioDesarrollado01
{
 class Program
 {
   static void Main(string[] args)
   {
     Console.WriteLine(" ** PROMEDIO DE NOTAS ** ");
     Console.WriteLine(" ");

     string alumno;
     int nota1, nota2, nota3;

     Console.Write("Ingrese nombre del alumno: ");
     alumno = Console.ReadLine();
     Console.Write("Ingrese la nota 1: ");
     nota1 = int.Parse(Console.ReadLine());
     Console.Write("Ingrese la nota 2: ");
     nota2 = int.Parse(Console.ReadLine());
     Console.Write("Ingrese la nota 3: ");
     nota3 = int.Parse(Console.ReadLine());

     double promedio = (nota1+nota2+nota3)/3.0;

     Console.WriteLine("---------------------------");
     Console.WriteLine("EL PROMEDIO: "+promedio.ToString("0.00"));

     Console.ReadKey();
    }
 }
}
```

Resultado:

```
 ** PROMEDIO DE NOTAS **

Ingrese nombre del alumno: Manuel Torres
Ingrese la nota 1: 10
Ingrese la nota 2: 12
Ingrese la nota 3: 15
---------------------------
EL PROMEDIO: 12.33
```

Caso desarrollado 2 : conversión de medidas

Implemente una aplicación de consola en Visual C# 2015 que permita dar solución al siguiente caso:

Problema:

El área de proyectos de una empresa textil necesita controlar las medidas solicitadas por sus clientes. Estos proporcionan la medida de sus telas en metros y la empresa necesita convertirlas en centímetros, pies, pulgadas y yardas. Implemente una aplicación que permita convertir la cantidad dada en metros por un determinado cliente a sus equivalentes en centímetros, pulgadas, pies y yardas. Teniendo la siguiente tabla de conversiones:

1 metro = 100 centímetros
1 pie = 12 pulgadas
1 yarda = 3 pies
1 pulgada = 2.54 centímetros

Determinando las entradas y salidas:

Descripción	Tipo	Nombre de la variable
Metros	Entrada	Metros
Centímetros, pulgadas, pies y yardas	Salida	Centímetros, pulgadas, pies y yardas

Algoritmo:

```
Inicio
 Leer metros
 centimetros ← metros x 100
 pulgadas ← centimetros / 2.54
 pies ← pulgadas /12
 yardas ← pies /3
 Escribir centimetros, pulgadas, pies, yardas
Fin
```

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ejercicioDesarrollado02
{
 class Program
 {
   static void Main(string[] args)
   {
     Console.WriteLine(" ** CONVERSION DE MEDIDAS - EMP. TEXTIL ** ");
     Console.WriteLine(" ");

     double metros;

     Console.Write("Ingrese cantidad de metros: ");
     metros = double.Parse(Console.ReadLine());

     double centimetros = metros * 100.0;
     double pulgadas = centimetros / 2.54;
     double pies = pulgadas / 12.0;
     double yardas = pies / 3.0;

     Console.WriteLine("-----------------------------------");
     Console.WriteLine("CENTIMETROS: "+centimetros.ToString("0.00"));
     Console.WriteLine("PULGADAS: "+pulgadas.ToString("0.00"));
     Console.WriteLine("PIES: "+pies.ToString("0.00"));
     Console.WriteLine("YARDAS: "+yardas.ToString("0.00"));

     Console.ReadKey();
   }
 }
}
```

Resultado:

```
 ** CONVERSION DE MEDIDAS - EMP. TEXTIL **
Ingrese cantidad de metros: 250.50
-----------------------------------
CENTIMETROS: 25050.00
PULGADAS: 9862.20
PIES: 821.85
YARDAS: 273.95
```

Caso desarrollado 3 : capital de empresa

Implemente una aplicación de consola en Visual C# 2015 que permita dar solución al siguiente caso:

Problema:

Una sociedad de hermanos decide formar una empresa en la ciudad de Lima, para lo cual acuerdan realizar un aporte económico entre los tres socios formando así el capital necesario para la formación de la empresa. Diseñe una aplicación que determine el monto total del capital formado y el porcentaje de dicho capital por cada uno de los socios.

Determinando las entradas y salidas:

Descripción	Tipo	Nombre de la variable
Tres montos	Entrada	Monto1, Monto2, Monto3
Monto capital	Salida	Capital
Porcentajes de los tres socios	Salida	pSocio1, pSocio2, pSocio3

Algoritmo:

```
Inicio
 Leer monto1, monto2, monto3
 capital ← monto1+ monto2+ monto3
 pSocio1 ← (monto1 x 100) / capital
 pSocio2 ← (monto2 x 100) / capital
 pSocio3 ← (monto3 x 100) / capital
 Escribir capital, pSocio1, pSocio2, pSocio3
Fin
```

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ejercicioDesarrollado03
{
 class Program
 {
   static void Main(string[] args)
   {
     Console.WriteLine(" ** CAPITAL DE LA EMPRESA ** ");
     Console.WriteLine(" ");
```

```
        double monto1, monto2, monto3;

        Console.Write("Ingrese monto socio 01: $ ");
        monto1 = double.Parse(Console.ReadLine());
        Console.Write("Ingrese monto socio 02: $ ");
        monto2 = double.Parse(Console.ReadLine());
        Console.Write("Ingrese monto socio 03: $ ");
        monto3 = double.Parse(Console.ReadLine());

        double capital = monto1 + monto2 + monto3;
        double pSocio1 = (monto1 * 100) / capital;
        double pSocio2 = (monto2 * 100) / capital;
        double pSocio3 = (monto3 * 100) / capital;
        Console.WriteLine("-----------------------------------");
        Console.WriteLine("CAPITAL: $ "+capital.ToString("0.00"));
        Console.WriteLine("PORCENTAJE SOCIO 1: "+pSocio1.ToString("0.00"));
        Console.WriteLine("PORCENTAJE SOCIO 2: "+pSocio2.ToString("0.00"));
        Console.WriteLine("PORCENTAJE SOCIO 3: "+pSocio3.ToString("0.00"));

        Console.ReadKey();
      }
  }
}
```

Resultado:

```
 ** CAPITAL DE LA EMPRESA **

Ingrese monto socio 01: $ 15000
Ingrese monto socio 02: $ 12500
Ingrese monto socio 03: $ 35000
-----------------------------------
CAPITAL: $ 62500.00
PORCENTAJE SOCIO 1: 24.00
PORCENTAJE SOCIO 2: 20.00
PORCENTAJE SOCIO 3: 56.00
```

Caso desarrollado 4 : control de venta de productos

Implemente una aplicación de consola en Visual C# 2015 que permita dar solución al siguiente caso:

Problema:

Una tienda ha puesto en oferta la venta de un producto con un 11 % de descuento sobre el importe de la compra. Diseñe un algoritmo que determine el importe de la compra, el importe del descuento y el importe a pagar por la compra de cierta cantidad de unidades del producto.

Determinando las entradas y salidas:

Descripción	Tipo	Nombre de la variable
Precio y cantidad de productos	Entrada	Precio, cantidad
Importe, descuento e importe total a pagar	Salida	Importe, descuento, total

Algoritmo:

```
Inicio
 Leer precio, cantidad
 importe ← cantidad x precio
 descuento ← 11/100 x importe
 total ← importe – descuento
 Escribir importe, descuento, total
Fin
```

Programa:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ejercicioDesarrollado04
{
 class Program
 {
   static void Main(string[] args)
   {
     Console.WriteLine(" ** CONTROL DE VENTA DE PRODUCTOS ** ");
     Console.WriteLine(" ");
```

```
        double precio;
        int cantidad;

        Console.Write("Ingrese precio del producto: $ ");
        precio = double.Parse(Console.ReadLine());
        Console.Write("Ingrese cantidad comprada: ");
        cantidad = int.Parse(Console.ReadLine());

        double importe = cantidad * precio;
        double descuento = 11/100.0 * importe;
        double total = importe - descuento;

        Console.WriteLine("-----------------------------------");
        Console.WriteLine("IMPORTE TOTAL: $ "+importe.ToString("0.00"));
        Console.WriteLine("DESCUENTO    : $ "+descuento.ToString("0.00"));
        Console.WriteLine("TOTAL A PAGAR: $ "+total.ToString("0.00"));

        Console.ReadKey();
      }
  }
}
```

Resultado:

```
** CONTROL DE VENTA DE PRODUCTOS **

Ingrese precio del producto: $ 25.70
Ingrese cantidad comprada: 10
-----------------------------------
IMPORTE TOTAL: $ 257.00
DESCUENTO    : $ 28.27
TOTAL A PAGAR: $ 228.73
```

Preguntas frecuentes

➤ **¿Cómo comprobar que los pseudocódigos son correctos?**

La comprobación de un pseudocódigo es opcional, ya que, conforme va desarrollando aplicaciones, estas se volverán más sencillas. Pero se puede realizar una comprobación del código mediante una prueba de escritorio, llamada así porque simula el comportamiento que puede tomar algoritmo. Su objetivo principal es detectar posibles errores, omisiones o, simplemente, mejorar el algoritmo. A continuación, se muestra cómo realizar la prueba de escritorio del siguiente problema:

Una tienda de abarrotes tiene inconvenientes para controlar la venta de productos; el proceso inicia cuando un cliente solicita un determinado producto, el vendedor verifica el precio del producto y de acuerdo a la cantidad seleccionada calcula el monto que se debe cancelar. Se pide diseñar un algoritmo que determine el monto a pagar por la compra de cierta cantidad de un producto.

El pseudocódigo propuesto para la solución del problema es:

```
Inicio
    Leer precio, cantidad
    monto ← precio x cantidad
    Escribir monto
Fin
```

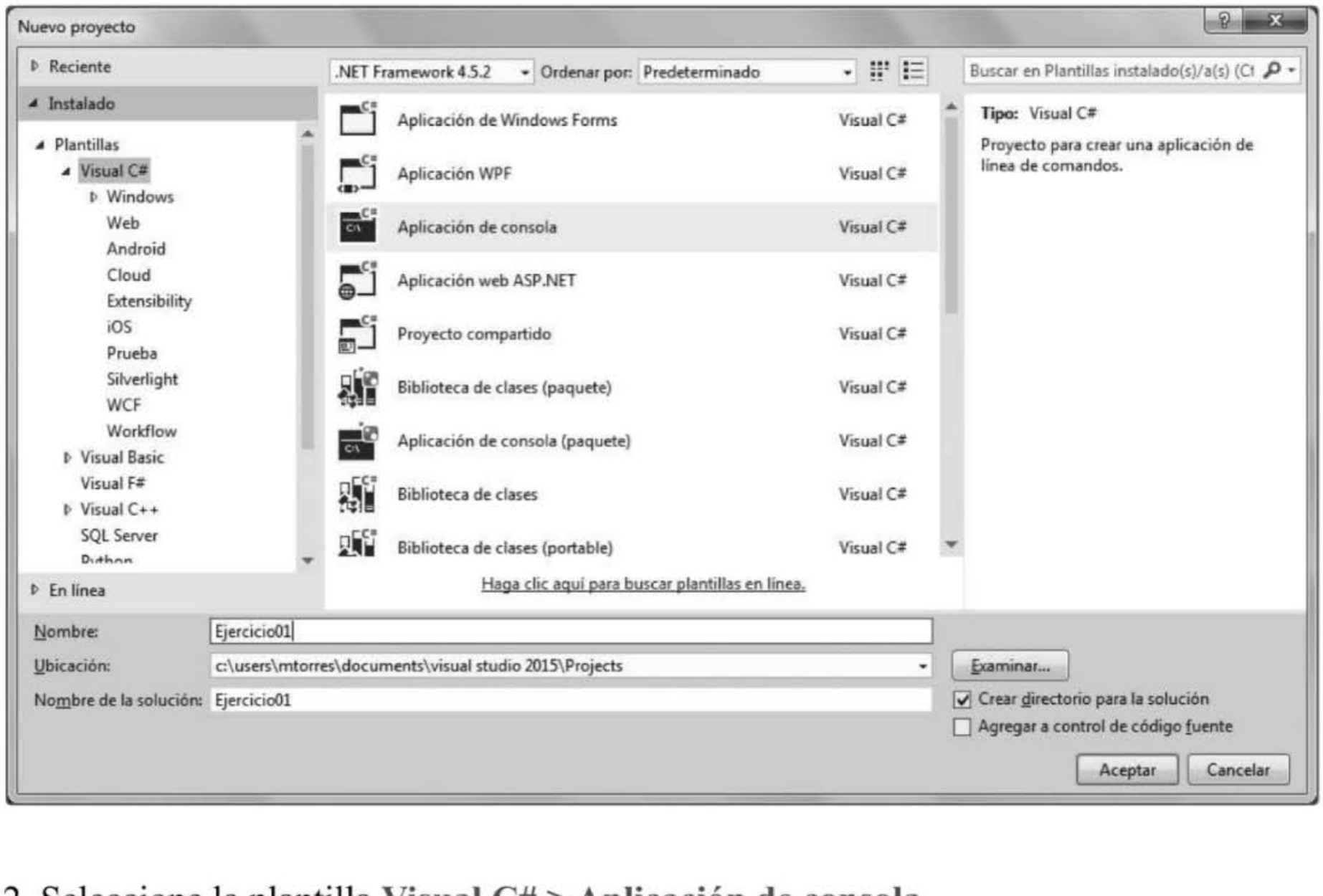
Se realiza la prueba de escritorio:

Precio	Cantidad	Monto	Resultado
7.50	10	75.00	75.00
10.00	20	200.00	200.00

➤ ¿Cómo crear un proyecto para C# de tipo consola en Visual Studio 2015?

Desde el entorno de Visual Studio 2015.

1. Diríjase a **Archivo > Nuevo > Proyecto**.



2. Seleccione la plantilla **Visual C# > Aplicación de consola**.
3. Asigne un nombre a la aplicación, se sugiere que dicho nombre sea una palabra única. También puede designar una ubicación para el proyecto seleccionando el botón **Examinar**.
4. Finalmente, si todo es realizado correctamente, presione el botón **Aceptar.**

Para recordar

- Los algoritmos están considerados como métodos que permiten resolver un problema particular de manera sencilla, precisa y rápida.
- Al desarrollar un problema primero se debe entender cuál es el objetivo del problema y qué necesita para cumplir con dicho objetivo; luego, preparar el algoritmo de solución; y, finalmente, implementarlo en un programa, por ejemplo, C# con Visual Studio 2015.



- Entre las características principales de los algoritmos, se puede mencionar que siempre tienen un punto de inicio y un fin. También un orden de realización de pasos; se debe tener en cuenta que la ejecución de una aplicación siempre se realiza de arriba hacia abajo y, finalmente, que el algoritmo no se desarrolla para una sola prueba, debe aplicarse para todo tipo de prueba y en todas dará el mismo resultado, por ejemplo, un algoritmo que permite calcular el promedio de notas siempre devolverá el promedio de notas.
- Los lenguajes de programación permiten interpretar los códigos que el programador le proporciona siguiendo un determinado patrón llamado reglas. Estos están formados por reglas y símbolos que definen la estructura de sus componentes. La característica principal es editar, compilar y ejecutar; editar es digitar el código, compilar es comprobar las sentencias y ejecutar es poner en marcha la aplicación.

Capítulo 2

Introducción al Visual Studio - Proceso de instalación

Capacidad terminal:

Conocer la forma adecuada de realizar la instalación de Visual Studio y reconocer todos los elementos que componen el entorno de Visual Studio.

Contenido:

- Introducción
- Plataforma .Net
- NET Framework 4.6
- Principios del Common Language Runtime
- Biblioteca de clases .Net
- Versión de la plataforma .Net
- Proceso de instalación del Visual Studio
- Barra de herramientas estándar
- Cuadro de herramientas
- Explorador de soluciones
- Ventana de propiedades
- Soluciones para C#
- Proyectos para C#
- Windows Forms
- Controles visuales
- Notación para los nombres de los controles
- Control Form
- Control Label
- Control TextBox
- Control Button
- Control ComboBox
- Control ListBox
- Control ListView
- Preguntas frecuentes
- Para recordar

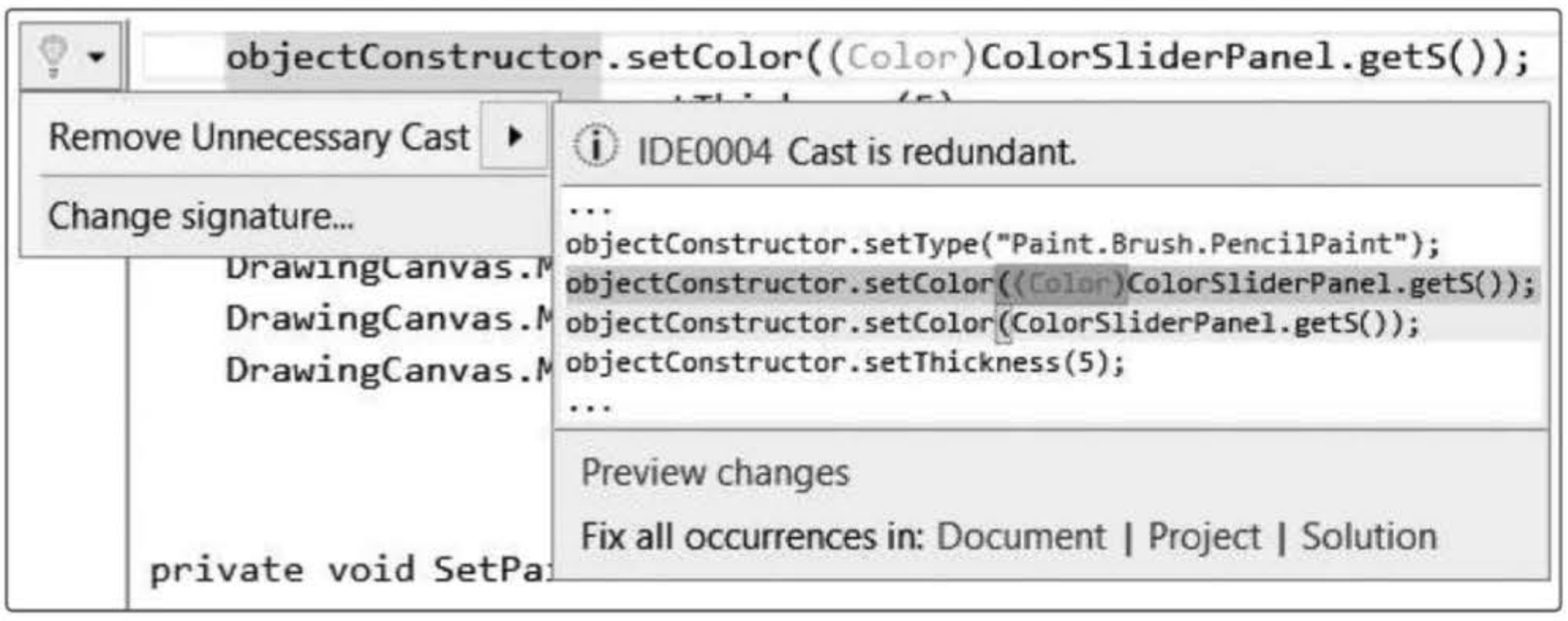
Introducción

Visual Studio 2015 es considerado un entorno de desarrollo completo e integrado que permite implementar aplicaciones para la plataforma Windows, Android y iOS; así como aplicaciones web de manera profesional.

Visual Studio 2015 es considerado como un ambiente de desarrollo integrado (IDE Integrated Development Environment), el cual permite al programador crear aplicaciones de desarrollo en un ambiente totalmente amigable y, a su vez, profesional.

Los IDE tienen como característica contar con un editor de código que permite codificar contando con un autocompletado de código llamado Intellisense. En Visual Studio 2015, el Intellisense permite disminuir el número de errores mientras escribe un determinado código. Esta funcionalidad solo se encuentra activa cuando se encuentra en la vista de código de la aplicación y su principal ayuda al programador es seguir los vínculos que requiere un código.

Entre las novedades de la versión 2015 de Visual Studio está el manejo de las herramientas de codificación eficaz porque permite navegar por código completo, así como comprender el concepto del código mediante cuadros de diálogo y refactorizar el código de manera rápida y sencilla.



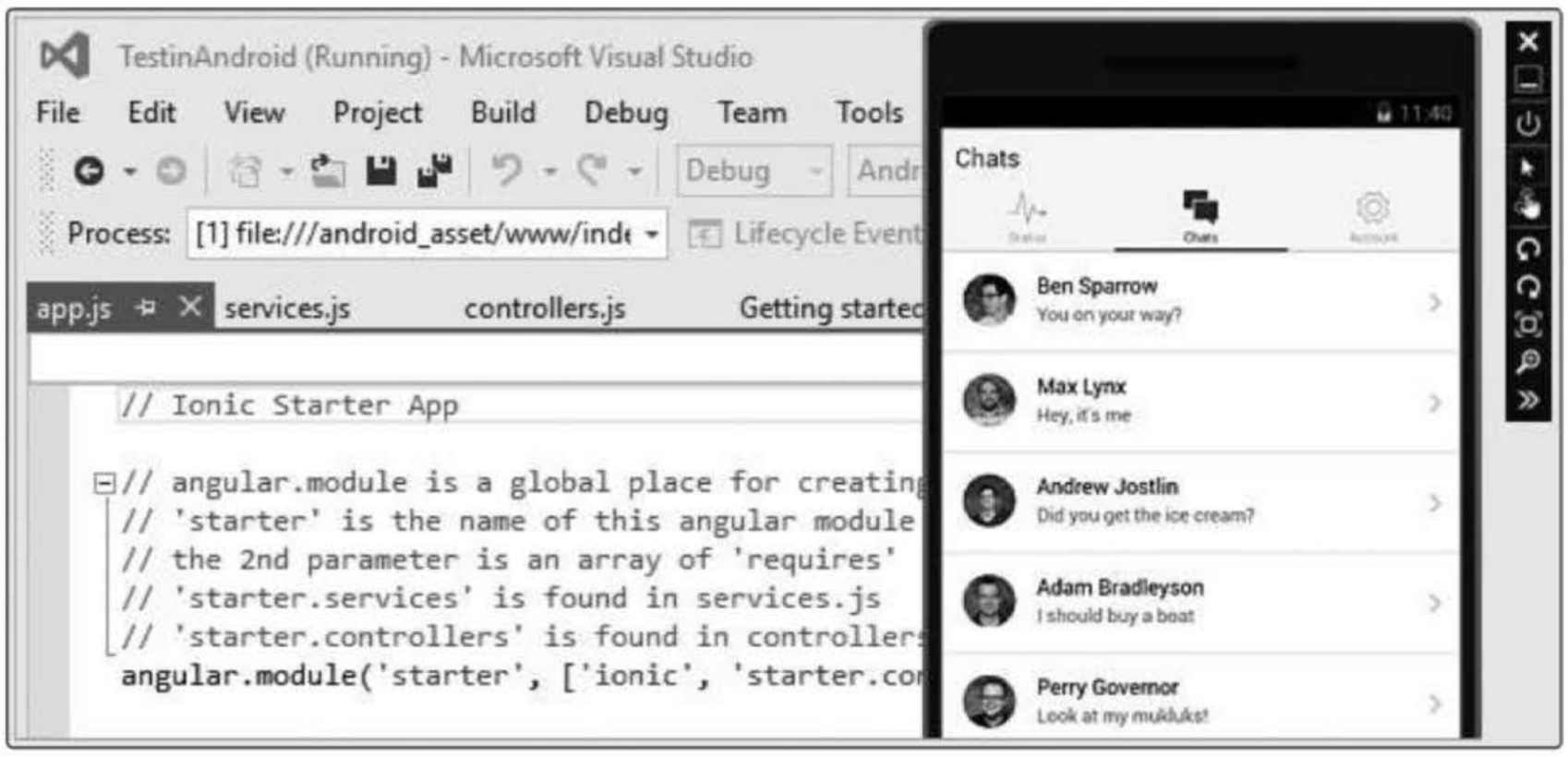
<https://www.visualstudio.com/products/visual-studio-community-vs>

Con respecto a la depuración, permite encontrar errores comunes que se pueden presentar en el código de una aplicación implementada en cualquier lenguaje de programación del Visual Studio 2015. Asimismo, diagnostica errores de rendimiento que pueden ser corregidos al instante por el programador.

```
41              ImageListSource.IsSourceGrouped = false;
42              await LoadImages();
43          }
44
45          private async Task LoadImages()
46          {
47              progressBar.Visibility = Visibility.Visible;
48              imgSelectedImage.Visibility = Visibility.Collapsed;
49              m_images = new ConcurrentBag<ImageItem>();
50
51              var folder = KnownFolders.PicturesLibrary;
52              await GetImagesFromCloud();
53              await LoadImagesFromDisk(folder);
54              ImageListSource.Source = from image in
55                  m_images orderby image.Folder.Name select image;
56
57              progressBar.Visibility = Visibility.Collapsed;
58              imgSelectedImage.Visibility = Visibility.Visible;
59          } ≤2,046ms elapsed
60
61          private async Task GetImagesFromCloud()
```

<https://www.visualstudio.com/products/visual-studio-community-vs>

Con respecto a las aplicaciones, amplía las posibilidades de desarrollo en más dispositivos como aplicaciones de escritorio, plataforma universal de Windows, Android y iOS de manera sencilla y profesional.



<https://www.visualstudio.com/products/visual-studio-community-vs>

Con respecto a las aplicaciones web, proporciona un desarrollo muy mejorado en ASP .NET, Node.js y JavaScript; de esta forma garantiza la implementación de aplicaciones modernas haciendo uso de código HTML5, CSS3 y JavaScript.

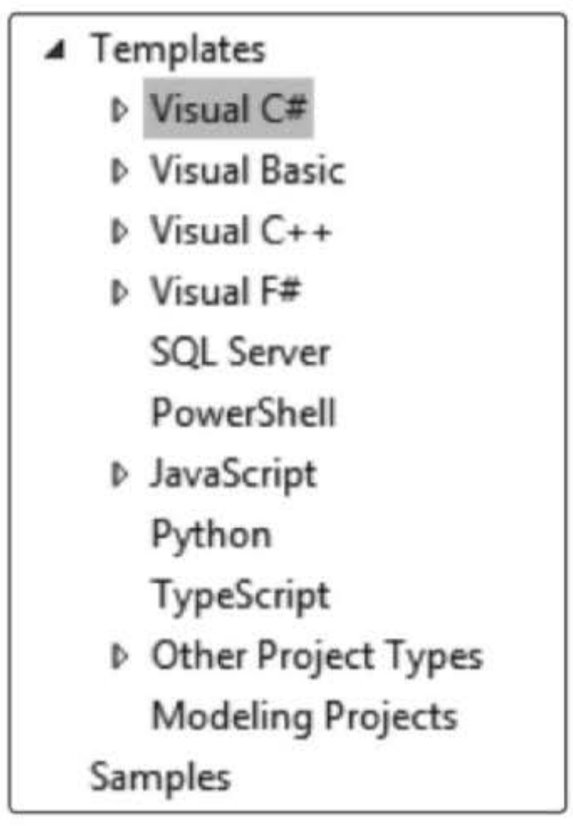
```
        ViewBag.Title = "Home Page";
    }

<div id="myCarousel" class="carousel slide" data-ride="carousel"
    <ol class="carousel-indicators">
        <li dat                                    ata-slide-to="0" class="active
        <li dat                                    ata-slide-to="1"></li>
        <li dat                                    ata-slide-to="2"></li>
        <li dat                                    ata-slide-to="3"></li>
    </ol>
    <div class=                                    istbox">
        <div cl
            <im                                    Banners-01.png" alt="ASP.NET"
            <div class="container">
                <div class="carousel-caption">
                    <p>
```

carousel
carousel-caption
carousel-control
carousel-indicators
carousel-inner
checkbox
checkbox-inline
clearfix
close

<https://www.visualstudio.com/products/visual-studio-community-vs>

Finalmente, se debe mencionar que Visual Studio presenta compatibilidad con varios lenguajes de programación. De manera que si usted es programador de otro lenguaje, siempre encontrará en Visual Studio uno compatible como C#, Visual Basic, C++, JavaScript entre los lenguajes principales.

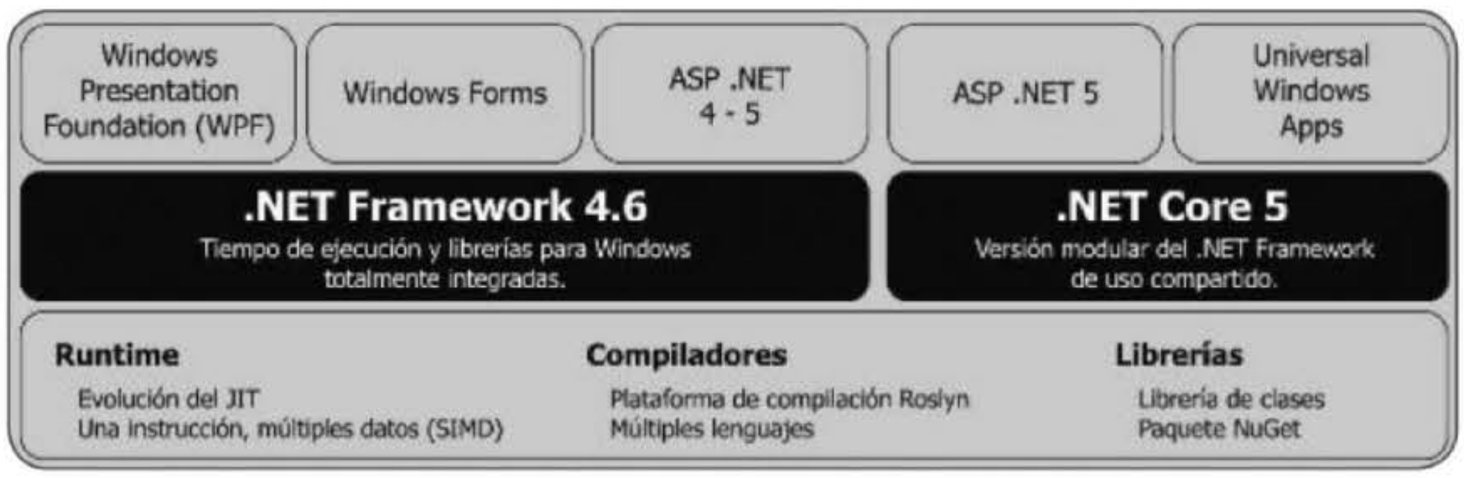


2.1 Plataforma .Net

Es una tecnología característica de Microsoft que permite realizar la compilación, depuración y ejecución de todo tipo de aplicaciones. Esta plataforma proporciona un entorno de desarrollo orientado a objetos; asimismo, proporciona un entorno de ejecución coherente porque permite eliminar problemas de rendimiento entre los diferentes entornos de trabajo que presenta Visual. También esta plataforma está pensada en el programador, ya que le permite desarrollar aplicaciones en diferentes espacios basados en tecnologías de plataforma, web o móviles.

La plataforma .Net cuenta con los componentes CLR (Common Language Runtime) y la biblioteca de clases. El Common Language Runtime es el componente principal de la plataforma .NET y es considerado como un agente que administra el código en tiempo de ejecución y proporciona servicios centrales, y entre sus características principales está administrar la memoria en tiempo de ejecución, administrar subprocesos y controlar la comunicación remota.

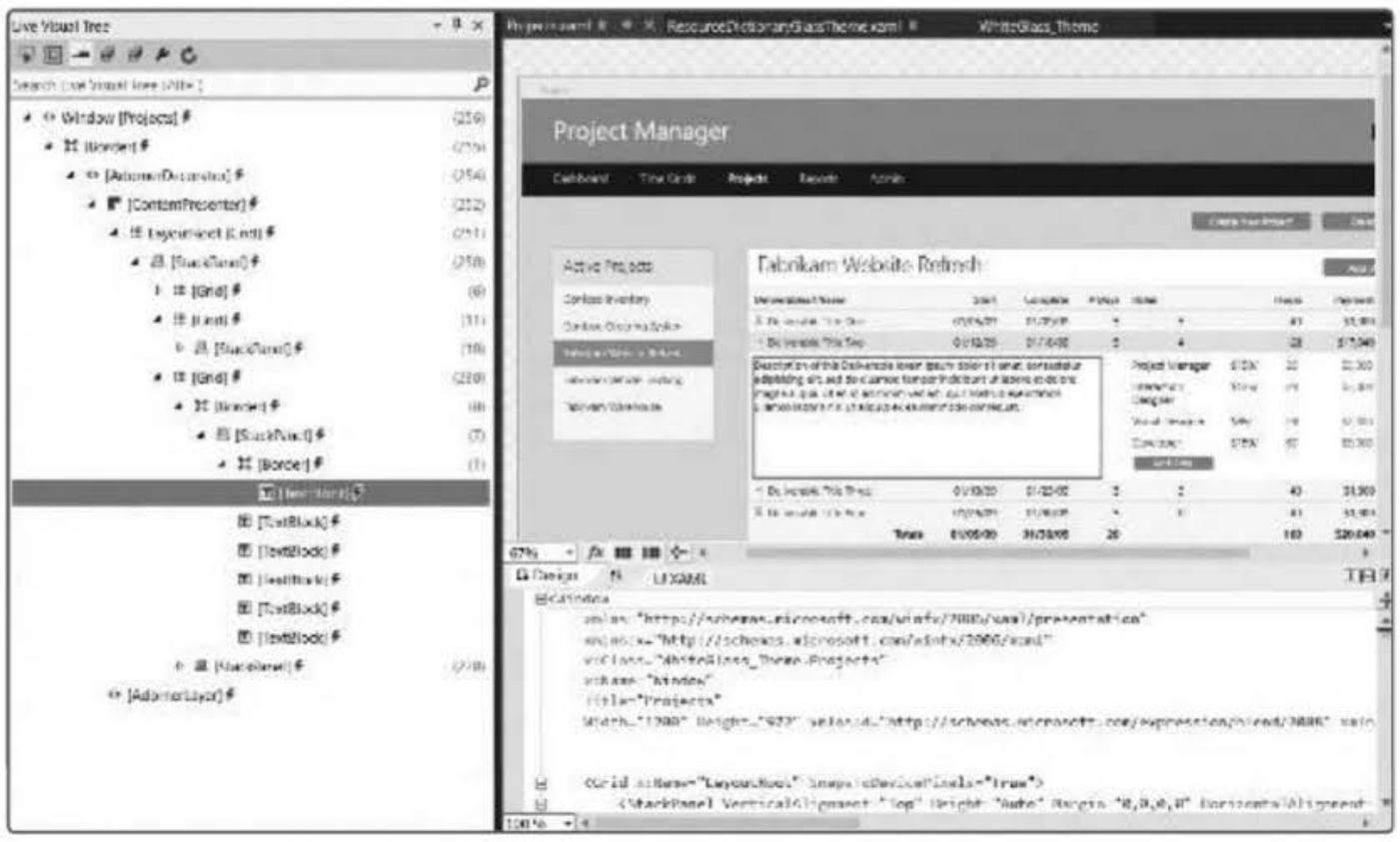
Mientras que la biblioteca de clases es considerada como una gran colección de elementos orientada a objetos de tipos totalmente reutilizables, de forma que se pueden desarrollar aplicaciones básicas con interfaz gráfica de usuario (GUI), de línea de comandos hasta aplicaciones web profesionales con ASP.NET. La siguiente imagen muestra el contexto general del Framework .Net 4.6.



Arquitectura .NET Framework 4.6

- **Windows Presentation Foundation (WPF)**

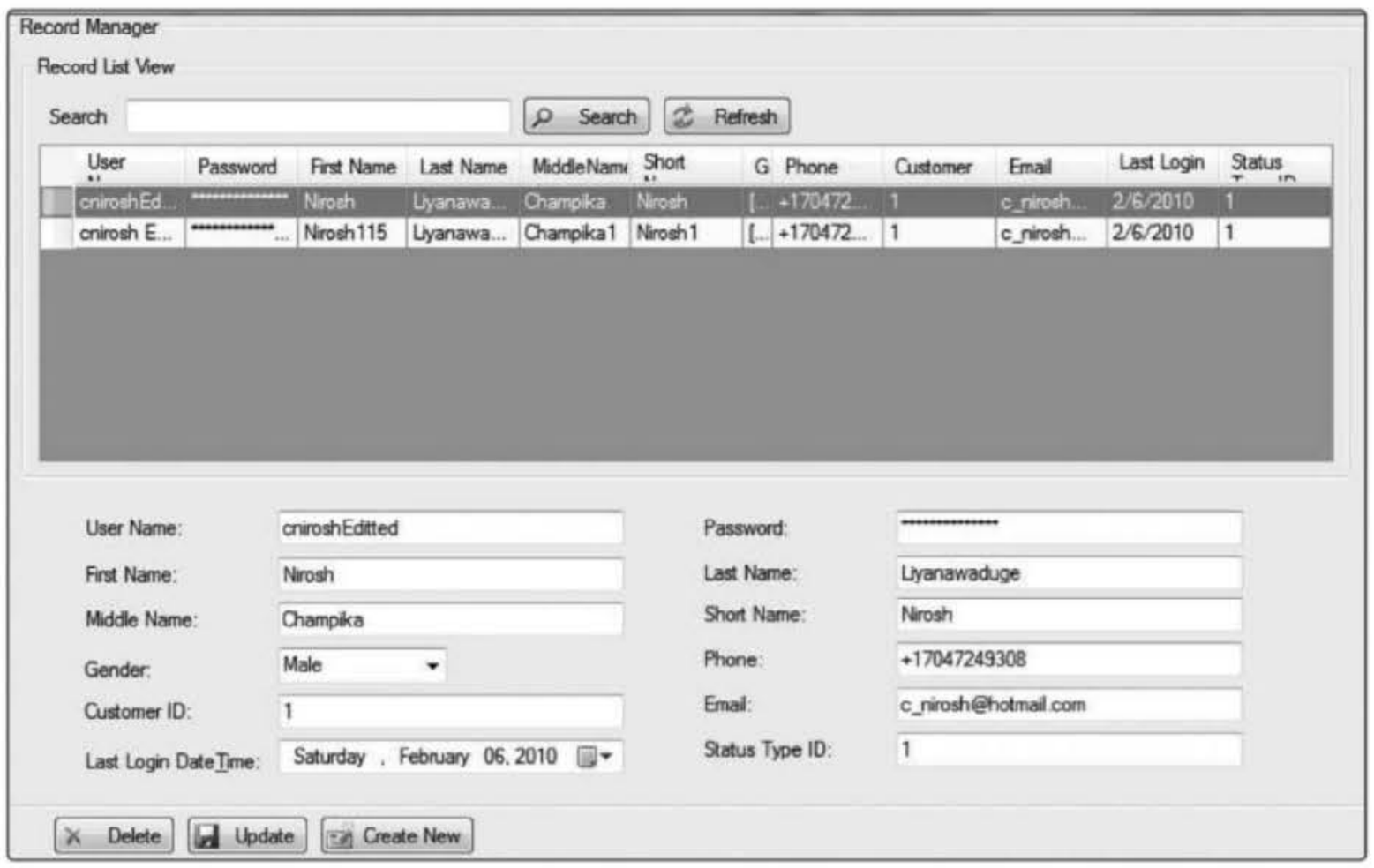
  Para la versión Visual Studio 2015, WPF proporciona a los desarrolladores un modelo de programación unificado para crear modernas aplicaciones de escritorio empresariales en Windows.



<http://blogs.msdn.com/cfs-filesystemfile.ashx/__key/communityserver-blogs-components-weblogfiles/00-00-01-29-92/1488.2014FallMoment-_2D00_-WPF-vNext-VisualDiagnostics.png>

- **Windows Forms**

  Representa la unidad básica de una aplicación en Visual C#, ya que a partir de su uso se pueden desarrollar aplicaciones de plataforma solo con agregar controles y asignar ciertas propiedades; a esto se denomina GUI o Interfaz Gráfica de Usuario, el cual permite interactuar al usuario con la misma aplicación. Visual Studio se caracteriza por proporcionar un entorno de desarrollo integrado (también llamado IDE), el cual cuenta con un conjunto completo de controles escrito con .NET Framework.



<http://www.codeproject.com/Articles/128280/Architecture-Guide-Windows-Forms-Generics-Auto-Map>

- **ASP .NET 5**

  También es conocido como ASP .Net vNext, el cual es considerado un framework exclusivamente web de Microsoft. Una de las características principales de la versión 5 es que se pueden ejecutar aplicaciones en diferentes plataformas como Windows, Linux y Mac. Además, es de código abierto.



<http://somostechies.com/bienvenido-asp-net-vnext/>

- **Universal Windows Apps**

  Con la aparición del Windows 10, la UWA (Universal Windows Apps) mejora la experiencia al programador. Al crear una aplicación bajo la plataforma, UWA garantiza ejecutarlo en diferentes dispositivos como un teléfono móvil, una aplicación de escritorio o un xBox.

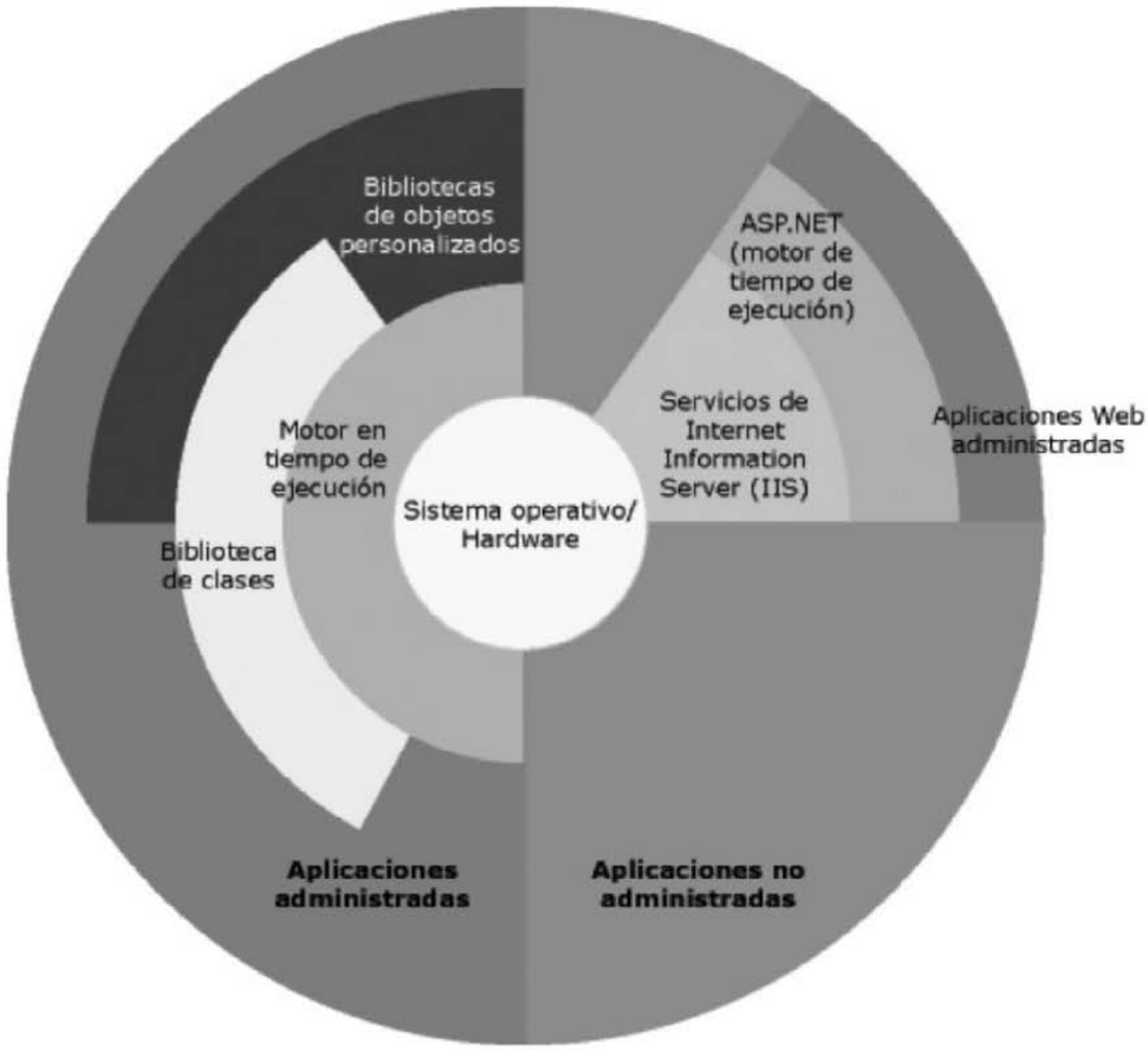


<https://msdn.microsoft.com/en-us/library/dn975273.aspx>

2.2 .NET Framework 4.6

La versión 2015 de Visual Studio cuenta con .NET Framework 4.6, el cual ofrece más de 150 API nuevas y 50 API modificadas y actualizadas, con la finalidad de aceptar más escenarios. Entre las principales características para esta nueva versión del .NET Framework, se tiene:

- Aumento en la productividad con respecto al manejo de colecciones con IReadOnlyCollection<T>, el cual representa una colección de elementos fuertemente tipada y de solo lectura.
- ASP.NET ofrece una plataforma mucho más eficiente para compilar modernas aplicaciones basadas en la nube.
- Las aplicaciones implementadas para la Tienda Windows escritas en C# ahora presentan una mejora en el tiempo de ejecución, ya que usa el .NET Nativo, el cual compila las aplicaciones a código nativo; además, agrega un compilador Just-In-Time de 64-bit.
- Hay mejoras en los compiladores para C# y Visual Basic, ya que aceleran los tiempos de manera significativa y proporcionan API completas de análisis de código.
- En los lenguajes C# y Visual Basic, se hicieron muchas mejoras para que el desarrollador experimente un ambiente más intuitivo, práctico y principalmente productivo.



<https://msdn.microsoft.com/es-es/library/zw4w595w(v=vs.110).aspx>

El .Net Framework llama aplicaciones administradas a todo código desarrollado por un compilador de lenguaje orientado al tiempo de ejecución; dicho código presenta beneficios frente a los demás, por ejemplo, la integración con otros lenguajes, controlar las excepciones generadas, mejorar la compatibilidad entre las diferentes versiones del .NET la depuración.

La **biblioteca de clases** de .NET Framework es una biblioteca completa de clases. Contiene todas las clases que un desarrollador de aplicaciones puede imaginar, donde la lógica y su imaginación será su límite. También presenta una lista completa de interfaces y tipos de valor, los cuales proporcionan un acceso controlado a la funcionalidad del sistema. Se podría decir que es la base sobre la que se compila todo tipo de aplicación, así como componentes y controles de .NET Framework.

Resulta muy importante hablar del **motor en tiempo de ejecución**, ya que toda aplicación que desarrolle dentro de Visual Studio 2015 deberá ejecutarse en algún momento. Es aquí donde entra el motor, ya que permite iniciar la ejecución de la aplicación. Para lograrlo, ejecuta el proceso de crear dominios en el proceso y carga el código de usuario justamente en dicho dominio.

- **.NET Core 5**

  Es considerada como una versión modular del .NET Framework, lo que significa que trabaja bajo su plataforma como un subconjunto. Está diseñada especialmente para que las aplicaciones sean portables entre muchas plataformas. Asimismo, proporciona una funcionalidad clave para implementar las características necesarias de una aplicación y reutilizar este código independientemente del destino de la plataforma.

  Esto no quiere decir que las demás versiones del .NET Framework no permitían la portabilidad, sino más bien que era administrado de otra manera. Lo que diferencia la nueva versión es que cuenta con una funcionalidad compartida para las tareas clave, por ejemplo, leer archivos locales. Su característica principal se nota al hacer uso de herramientas de terceros, por ejemplo, iOS o Android.

  En la conferencia para desarrolladores Build2 2015 en el Moscone Center, San Francisco, Microsoft anunció el lanzamiento de la vista previa completa del .NET Core para las plataformas Linux y Mac. Con esto, Microsoft cumple su promesa de traer algunas de las características principales de su plataforma .NET a otras plataformas y especialmente de código abierto.



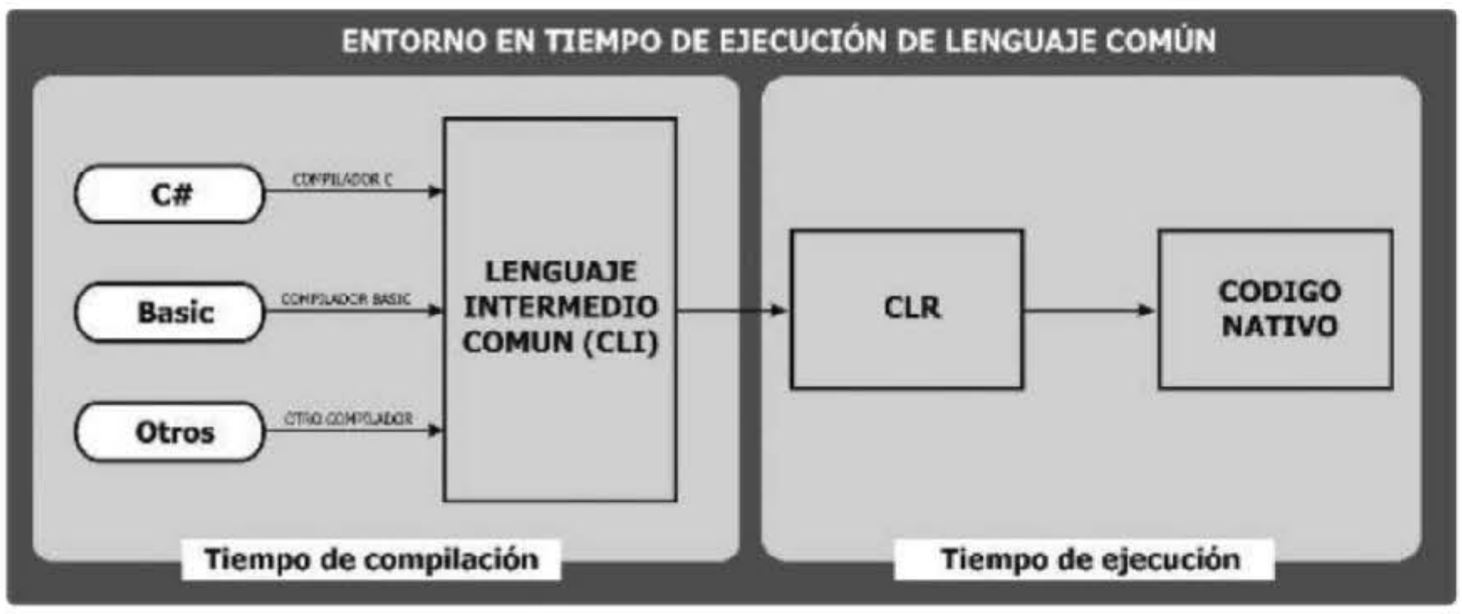
2.3 Principios del Common Language Runtime

Cuando se ejecuta una aplicación, el .NET Framework proporciona un entorno que facilita el proceso de desarrollo; dicho entorno lo realiza en 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 el CLR ,se encuentran:

- 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 administrar automáticamente dichos objetos así como su referencia y liberación cuando los objetos no son utilizados.
- CLR aplica la interoperatibilidad ya que permite al programador usar sus componentes COM o DLL desarrollados en versiones anteriores.

- Compilación JIT permite ejecutar todo el código administrado en lenguaje máquina nativo del sistema en el que se ejecuta la aplicación; mientras tanto administra la memoria mejorando enormemente su rendimiento.
- El runtime se puede alojar dentro de servidores como SQL Server o IIS (Internet Information Services) y, dentro de ellos, aplica las mismas características anteriormente especificadas.

La siguiente imagen muestra la forma de trabajo del entorno en tiempo de ejecución de lenguaje común más conocido como CLR:



Cuando el compilador interpreta el código desde un lenguaje de programación, genera un código intermedio antes de generar un código nativo. Se debe tener en cuenta que el código nativo es aquel código que puede ser interpretado por el procesador. A este código, se le llama código administrado y es generado por el CLI (Common Intermediate Language) o MSIL (Microsoft Language Intermediate). 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.

Las versiones del CLR fueron evolucionando conforme el .NET Framework lo hacía. A continuación, una tabla que muestra dichas versiones:

.NET Framework	CLR
1.0	1.0
1.1	1.1
2.0	2.0
3.0	2.0
4.0	4.0
4.5	4.0
4.6	4.0

2.4 Biblioteca de clases .NET

Es considerada una biblioteca completa de clases, interfaces y tipos de valor, que proporciona acceso a la funcionalidad de una aplicación. Se puede decir que es la base fundamental sobre la que se compila todo tipo de aplicación del .NET Framework sin importar la plataforma de trabajo. A continuación una lista de los espacios de nombres y categorías contenidas dentro de la biblioteca de clases más usadas:

Espacio de nombres	Descripción
System	Es el principal espacio de nombres usado 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 todas las funciones matemáticas, trigonométricas, logarítmicas y muchas más funciones. • **Random:** Representa un generador de números aleatorios. • **String:** Representa una serie de caracteres de tipo Unicode. • **Type:** Representa todo tipo de declaración en una aplicación.
System.Collections	Contiene una serie de clases que permiten manejar información masiva como si se tratase de un arreglo, con la diferencia de que estos datos son llamados colecciones como Arrayist, pilas, colas y diccionarios.
System.Configuration	Representa la información administrada por un archivo de configuración; para nuestro 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 puede 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.
Microsoft.VisualBasic	Representa las clases que permiten el uso de funcionalidades desde el compilador Visual Basic, por ejemplo, el uso de InputBox del Visual Basic.

2.5 Versiones de la plataforma .NET

En el año 1996, Microsoft contaba con cuatro productos independientes, a saber, Visual Basic, Visual C++, Visual FoxPro y Visual Sourcesafe. Esto creaba desarrolladores de diferentes líneas a pesar de usar un producto del mismo proveedor. El producto que permitía una integración entre todos era el Visual Sourcesafe. Pero no resultaba productivo que un programador nativo de una línea lo integre a los demás. En 1997, decide integrar los productos mencionados en uno solo que llamó Visual Studio 97, el cual fue mostrarlo al mundo en febrero del mismo año. De allí en adelante, han surgido diferentes versiones que se verá a continuación:

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

2.6 Proceso de instalación del Visual Studio

En Visual Studio se puede desarrollar todo tipo de aplicaciones en diferentes plataformas como se mencionó anteriormente. Existe una versión gratuita del software llamada Visual Studio Community, la cual promueve el desarrollo de aplicaciones sin fines de lucro por parte del desarrollador.

- **Link de descarga**

  La URL para descargar Visual Studio 2015 es https://www.visualstudio.com/downloads/download-visual-studio-vs.

➢ Requisitos

Para la instalación de Visual Studio 2015 se divide en secciones la línea de requisitos, tómese en cuenta que dichos requisitos representan lo mínimo que se deben 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	• Windows 8.1 para aplicaciones de la Tienda Windows, Windows Phone y Universal Apps.

➢ Proceso de instalación

Se inicia el proceso de instalación de Visual Studio 2015 y se explican 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. Véanse las siguientes imágenes que representan los pasos para la instalación:



1. **Iniciando el proceso de instalación**

   En este paso, se observa que solicita la ubicación dentro de la unidad de disco, el cual puede modificar mediante el botón **Examinar** representado por el botón tres puntos (...). Seguidamente, se debe seleccionar el tipo de instalación, en este caso, seleccione **Típica** para una instalación estándar con todas las aplicaciones que ofrece Visual Studio 2015. Finalmente, presione el botón **Instalar** para iniciar el proceso de instalación.



2. Proceso de instalación

En este paso, solo queda esperar el proceso de instalación, ya que el asistente se encuentra cargando los archivos necesarios para la aplicación.



3. Selección del entorno y color de tema

Una vez finalizada la instalación, el asistente pregunta en qué lenguaje de programación se realizarán las aplicaciones. De forma estándar, aparece **General**, pero se recomienda que elija su lenguaje de desarrollo. Luego se puede cambiar el entorno desde la aplicación misma (se encuentra en la sección Preguntas frecuentes de este capítulo). Seguidamente, puede modificar el tema de color del entorno pero, como se mencionó antes, esto puede ser modificado dentro de la aplicación (se encuentra en la sección Preguntas frecuentes de este capítulo). Finalmente, haga clic en el botón **Iniciar Visual Studio** para comenzar con la aplicación.

4. Pantalla inicial

El aspecto de la pantalla no ha cambiado con respecto a su versión anterior, como se puede observar en la siguiente imagen:



Un entorno adecuado de aplicación debe contar con las siguientes opciones dentro de la pantalla principal de Visual Studio 2015:

- Barra de herramientas estándar
- Cuadro de herramientas
- Explorador de soluciones
- Ventana de propiedades

2.7 Barra de herramientas estándar

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



Descripción de los principales botones:

Ícono	Herramienta	Descripción
	Navegar atrás Navegar adelante	En el editor de código, el botón de navegador desplaza el cursor por las líneas de código anteriormente seleccionadas. En otro entorno, el botón de navegador lo desplaza entre dicho entorno.
	Nuevo	Presenta dos opciones. Permite crear un nuevo proyecto o 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 quieren guardar todos los componentes del proyecto, ya sea Windows Forms, códigos, clases, etc.
	Iniciar o Ejecutar	Permite ejecutar una aplicación. Otra forma de ejecutar una aplicación es presionando la tecla F5.
	Comentario	Permite asignar y quitar un comentario a una línea de código, solo cuando se encuentra en el editor de código.

2.8 Cuadro de herramientas

Es uno de los principales paneles que debe considerar activar al momento de desarrollar cualquier tipo de aplicación, pues, Visual Studio presenta herramientas de mucha ayuda para el programador creativo y profesional. Finalmente, se debe tener en cuenta que el cuadro de herramientas muestra controles según el entorno de desarrollo seleccionado. En este caso, se trabajará con controles para aplicaciones de escritorio.

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



Entre sus principales fichas, se tiene:

Categoría	Descripción
Todos los 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 apliaciones 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 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.

2.9 Explorador de soluciones

Mediante el uso del Explorador de soluciones, se pueden administrar los elementos de una solución o un proyecto, y se puede examinar el código. En función del elemento especificado, el tipo de proyecto determina los comandos de administración disponibles.

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



Entre sus principales botones, se tiene:

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 se está trabajando en un proyecto de tipo Windows Forms.
	Permite actualizar los elementos mostrados en el Explorador de soluciones.

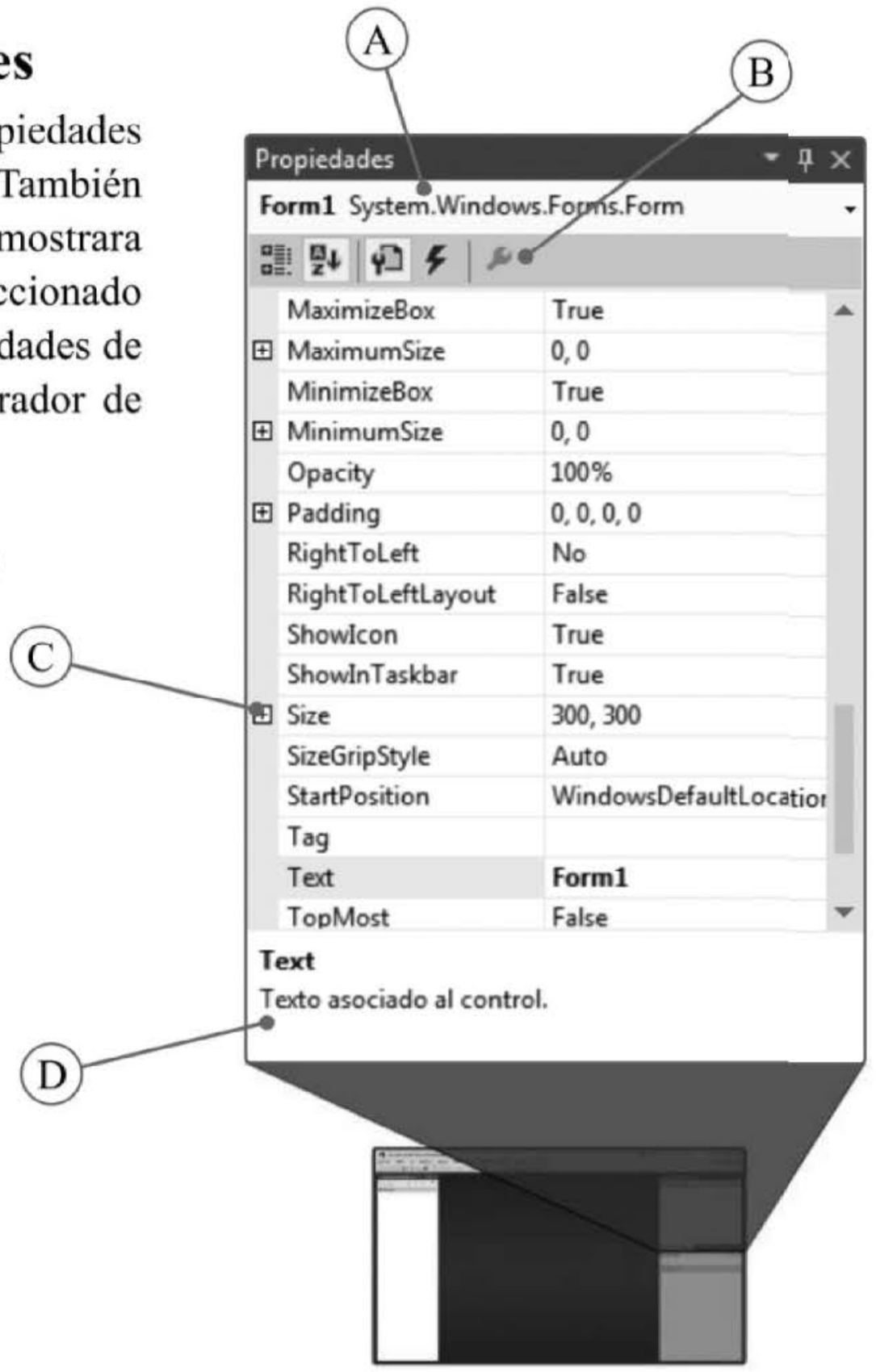
Los archivos que contiene el Explorador de soluciones:

Botón	Descripción
	Es el nombre de la solución actual. Entre paréntesis siempre se muestra la cantidad de proyectos que contiene la solución. El valor por defecto es uno, ya que al crear una solución automáticamente tiene un proyecto.
	Es el nombre del proyecto actual. A partir de aquí, se controlan todos los archivos contenidos.
Properties	Muestra las propiedades del proyecto actual.
Referencias	Muestra todas las referencias activas para un determinado proyecto.
App.config	Muestra el contenido del archivo de configuraciones.
Form1.cs	Muestra el contenido del formulario perteneciente al proyecto.
Program.cs	Muestra el código inicial del proyecto. Desde aquí se puede configurar el formulario de inicio.

2.10 Ventana de propiedades

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

Acceso: **Ver > Ventana propiedades**
Tecla de acceso: **F4**



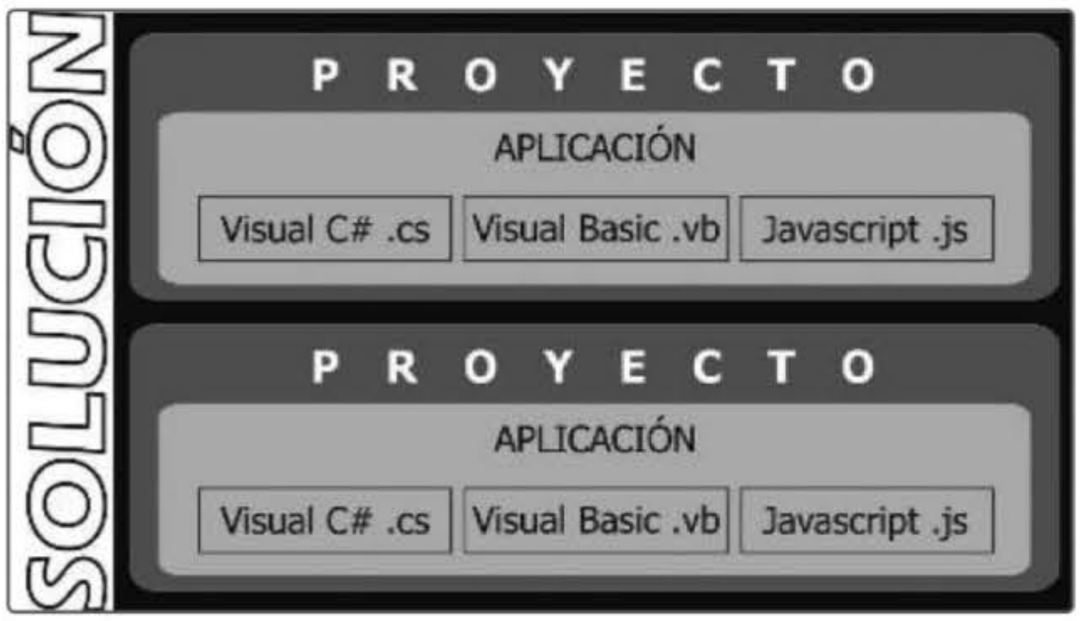
Partes de la ventana de propiedades:

	Partes	Descripción
A	Form1 System.Windows.Forms.Form	Form1 representa el nombre del objeto seleccionado. System.Windows.Forms.Form representa la clase de la cual se muestra el objeto.
B		• Muestra las propiedades en forma de categorías. • Muestra las propiedades ordenadas de forma ascendente. • Muestra las propiedades de un determinado objeto. • Muestra 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 clic sobre el nombre de la propiedad. Finalmente, el valor de la propiedad aparece al lado derecho, lo cual puede ser personalizado por el usuario. También puede modificarse dando doble clic sobre el valor.
D		Al pie de la ventana de propiedades, se muestra un diálogo que indica algunas características de la propiedad seleccionada.

2.11 Soluciones para C#

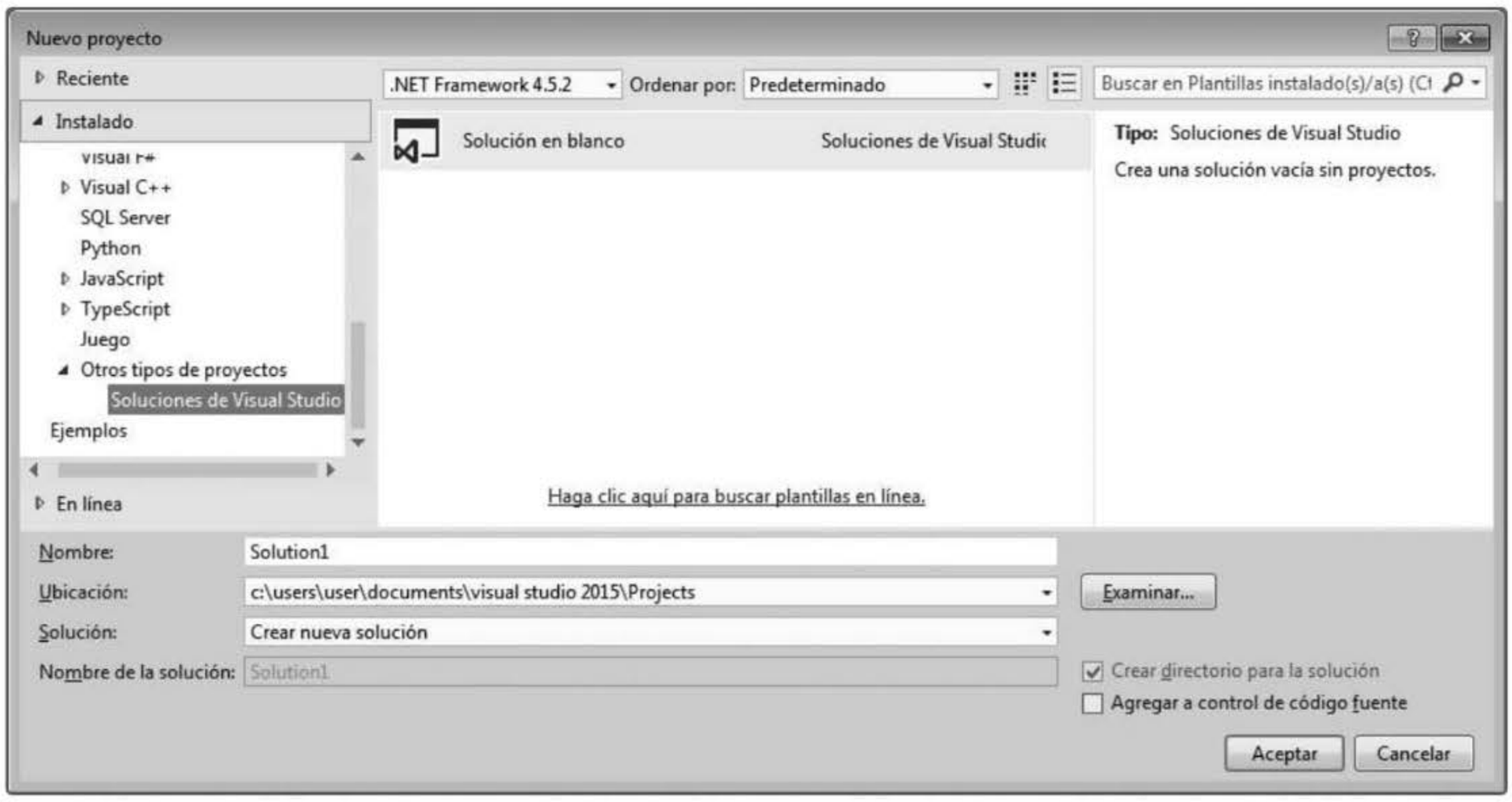
Una solución podría ser denominada como un espacio de trabajo ya que contiene un conjunto de proyectos en un solo archivo. La extensión asociada a este archivo es .sln, además de contener las preferencias y la información de configuración de toda la aplicación y el usuario mismo.

Véase la relación que existe entre un proyecto y un conjunto de proyectos:



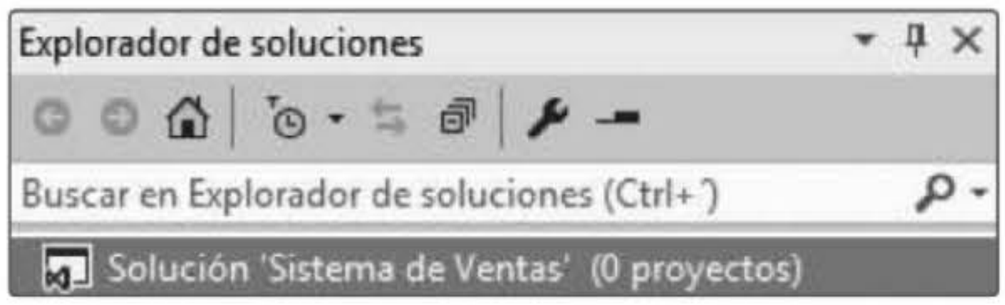
- **Crear una nueva solución**

1. Seleccione **Archivo > Nuevo > Proyecto**.
2. Seleccione **Otros tipos de proyectos > Soluciones de Visual Studio**.



3. Seguidamente, asigne un nombre y una ubicación a la solución. Si desea modificar la ubicación, presione el botón **Examinar**.
4. Finalmente, presione **Aceptar**.

En la ventana **Explorador de soluciones**, se debe mostrar de la siguiente manera:



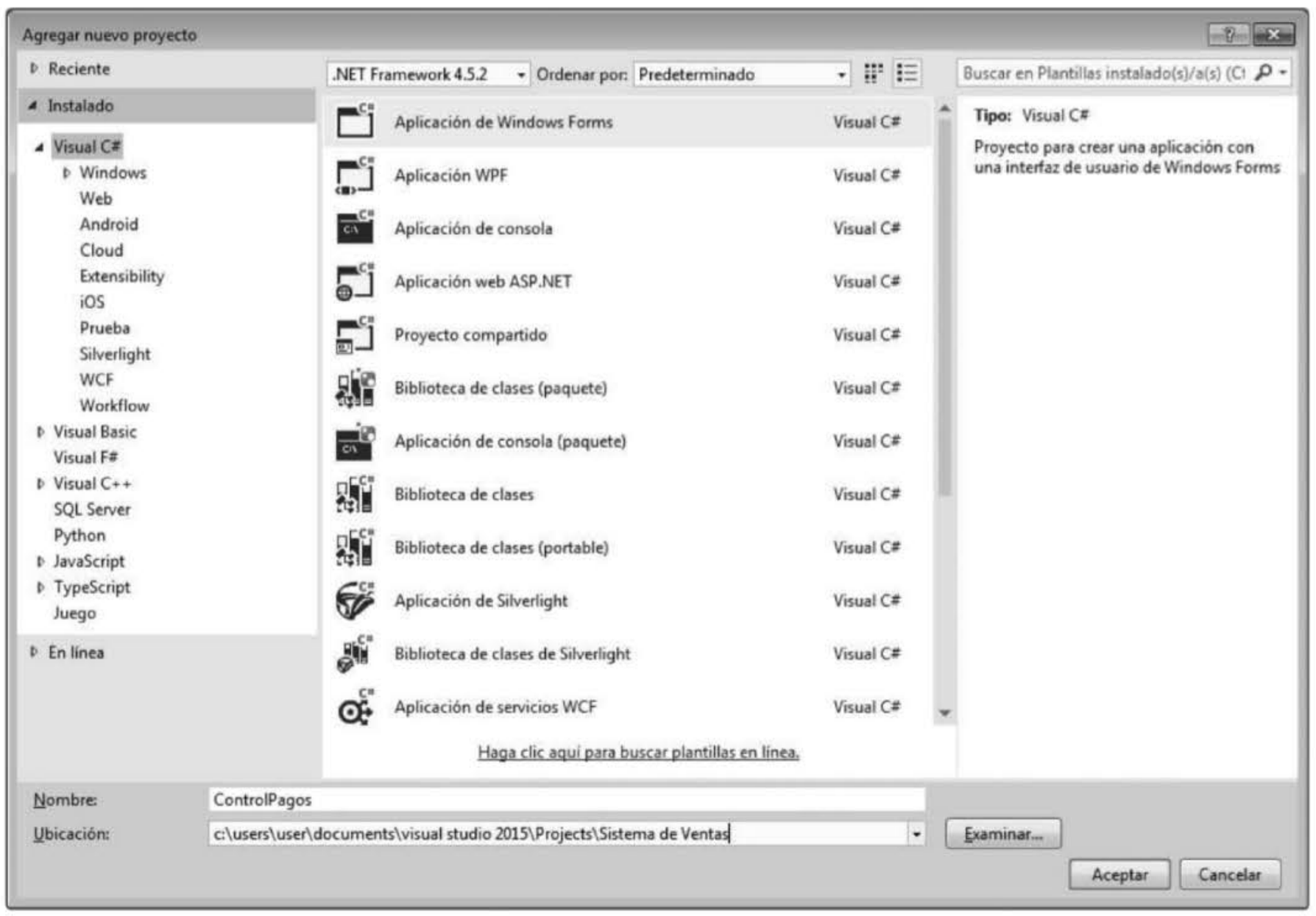
Como se observa, entre paréntesis se indica cero proyectos, ya que aún no se agregó ningún proyecto.

- **Agregar proyectos a la solución**

Véanse las diversas posibilidades para agregar un proyecto a una solución:

Primera forma:

1. Haga clic derecho sobre la solución desde el **Explorador de soluciones**.
2. Seleccione **Agregar > Nuevo > Proyecto.**



3. Asigne un nombre adecuado al proyecto y presione el botón **Aceptar**. No es necesario especificar la ruta del archivo, pues el archivo se registra en el mismo lugar que la solución.

Segunda forma:

1. Desde el menú seleccione **Archivo > Nuevo > Proyecto.**
2. Asigne un nombre al proyecto y presione el botón **Aceptar.**

Supóngase que se agregan dos proyectos a la solución inicial, el primero llamado **Controlpagos** y el segundo llamado **ControlAsistencia**, al final.



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 modificada a petición del usuario.
- Para detectar el proyecto de inicio, visualice los proyectos desde el Explorador de soluciones. Uno de los nombres del proyecto se encuentra en negrita, ese es el proyecto de inicio, como se muestra en la imagen anterior.
- Debe tener en cuenta que el último proyecto agregado a la solución siempre será el proyecto de inicio.

➢ **Establecer el proyecto de inicio**

Para modificar la asignación del proyecto de inicio, realice los siguientes pasos:

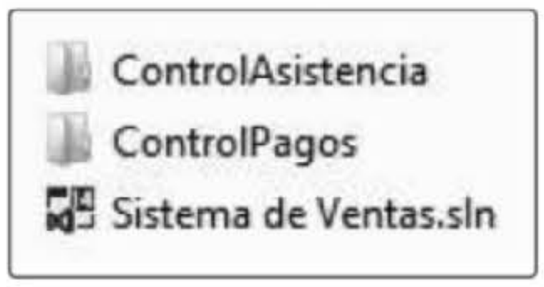
1. Haga clic derecho sobre un determinado proyecto.
2. Seleccione **Establecer como proyecto de inicio.**

➢ **Guardar la solución**

Se debe tener en cuenta que al guardar una solución, en realidad, se están guardando todos los elementos contenidos en la solución. Para guardar una solución, realice los siguientes pasos:

1. Seleccione **Archivo > Guardar todo**.
2. Especifique un nombre y una ubicación para la solución.

Considere que los proyectos y soluciones se registran dentro de una carpeta por defecto llamada **Projects**, de la carpeta **Visual Studio 2015**, ubicada en **Mis documentos** de Windows. En Windows, la solución y sus proyectos se visualizan tal como se muestra en la siguiente imagen:



➢ Cerrar la solución

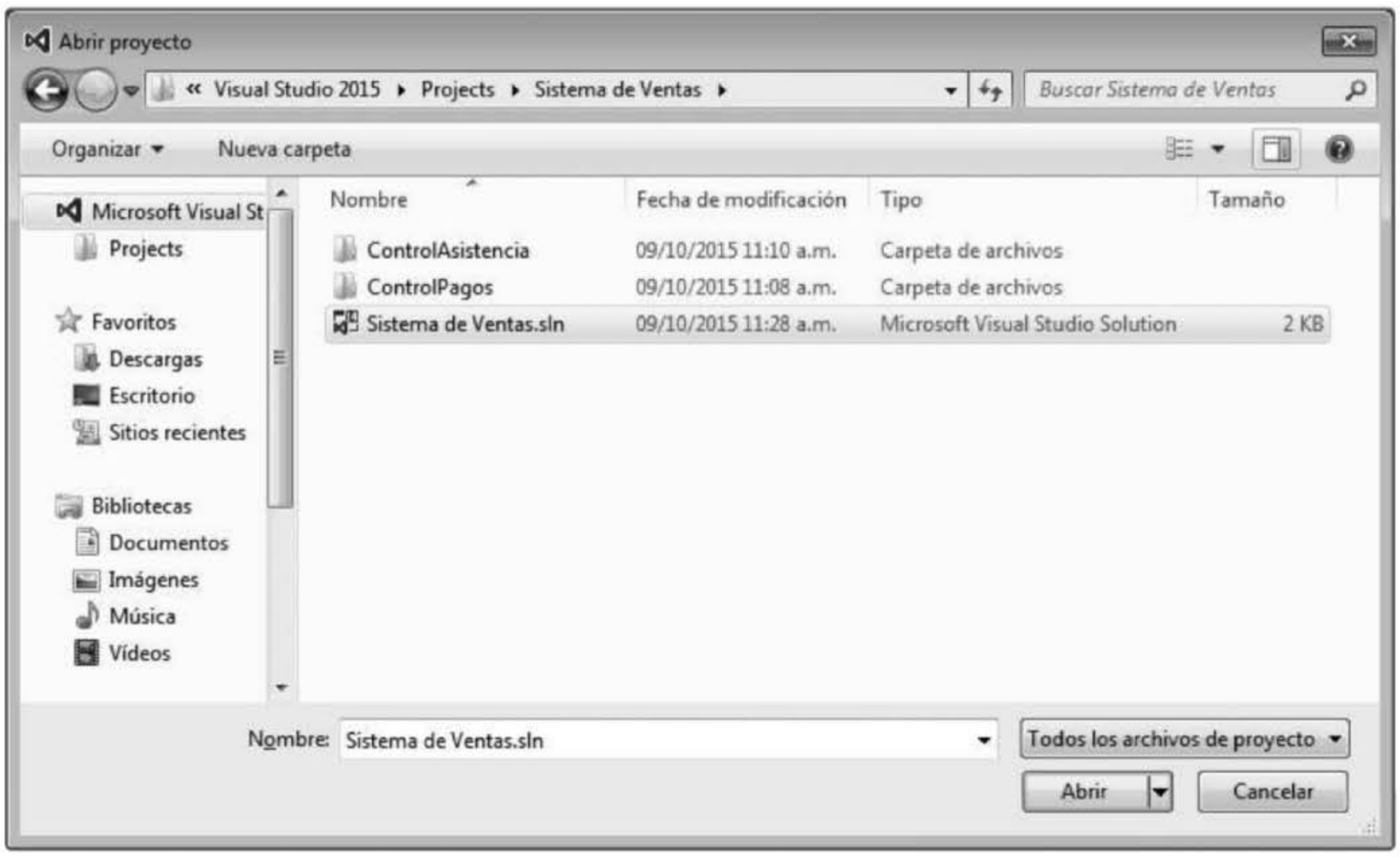
Al cerrar una solución, automáticamente todos los proyectos se cierran; ya que se encuentran contenidos en la solución. Para cerrar una solución, realice los siguientes pasos:

1. Seleccione **Archivo > Cerrar solución**.
2. Si se solicita guardar los cambios, se graban todos los elementos contenidos en la solución con los nombres asignados hasta el momento. Esto ayuda a no especificar un nombre por cada elemento contenido en la solución al momento de guardar.

➢ Abrir una solución

Al abrir una solución permitirá mostrar todos los elementos contenidos dentro de él, no será necesario especificar que archivo quiere visualizar puesto que se cargaran todos los archivos; para lograrlo realice los siguientes pasos:

1. Seleccione **Archivo > Abrir > Proyecto o Solución**.
2. Ubique la carpeta donde guarda la solución y seleccione el archivo de tipo .sln; tal como se muestra en la siguiente imagen:



3. Finalmente, seleccione el botón **Abrir**.

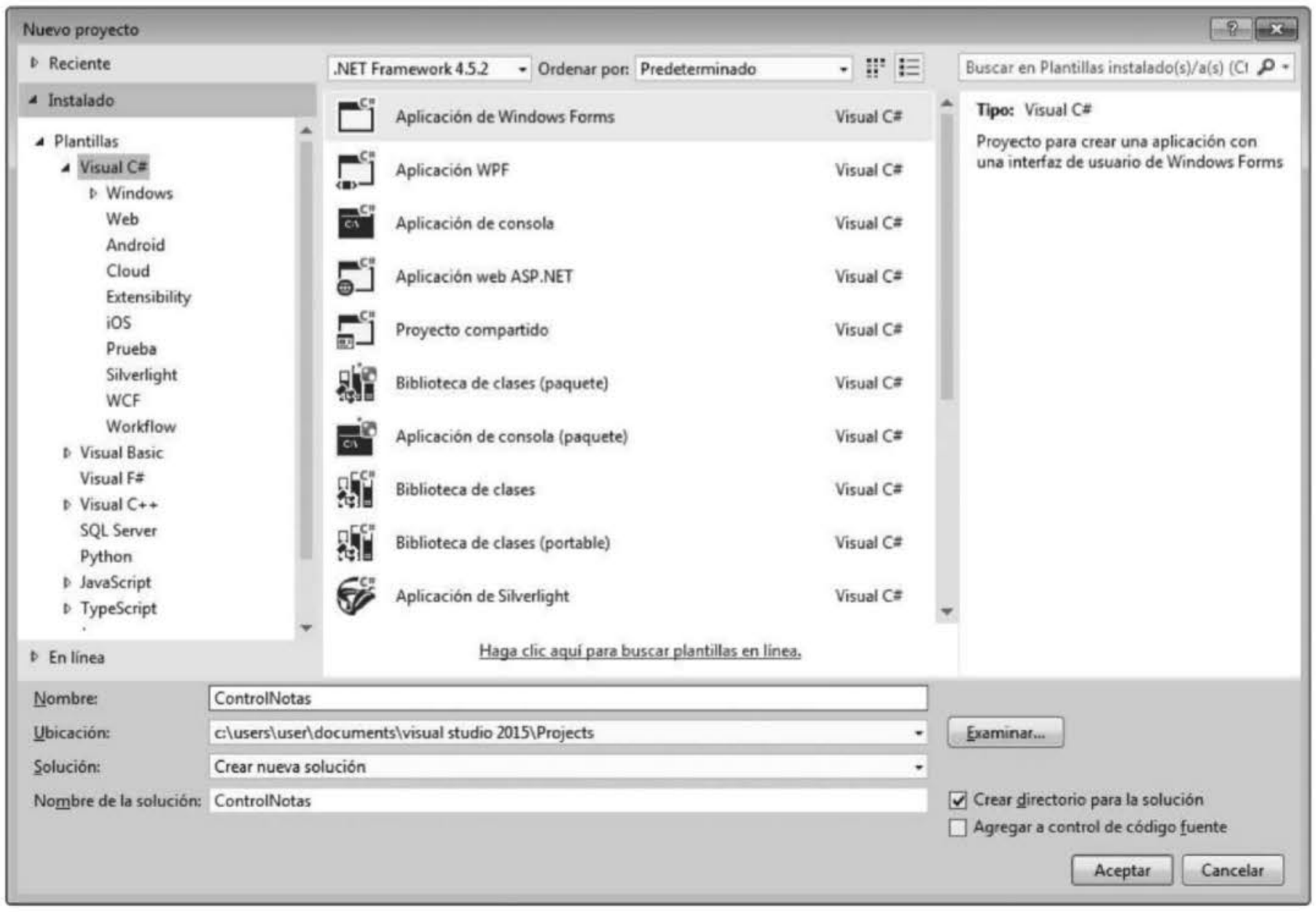
2.12 Proyectos para C#

Un proyecto puede contener todos los archivos de código fuente, iconos, imágenes, archivos de datos y cualquier otro elemento que se compilará en un programa ejecutable o sitio web.

➢ Crear un nuevo proyecto

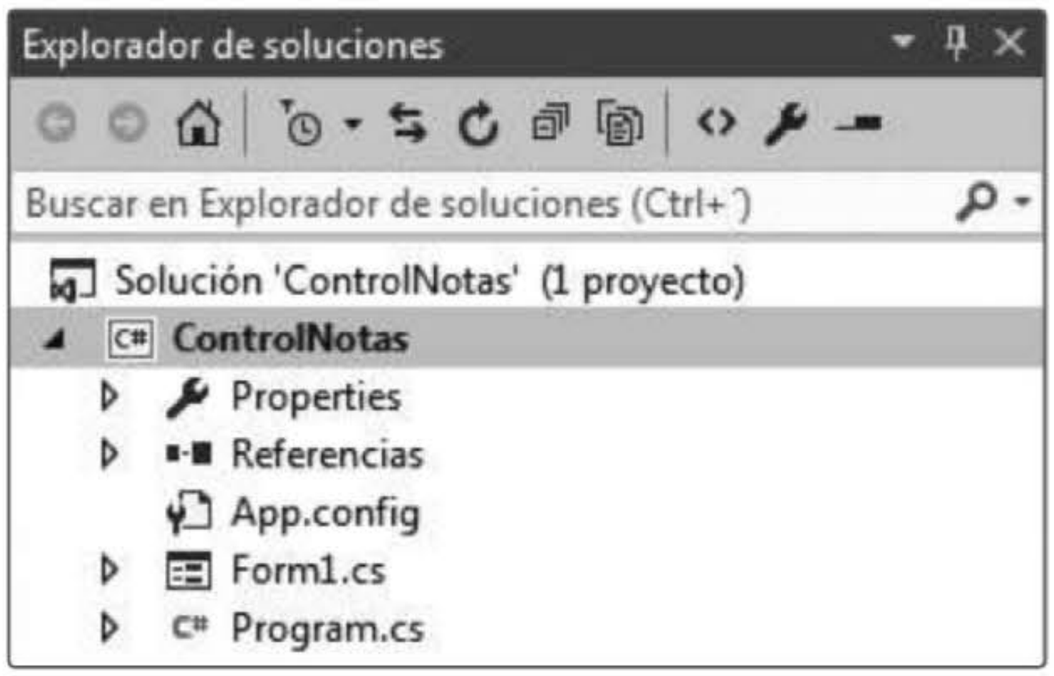
Cuando se crea un nuevo proyecto, automáticamente, se crea una solución de inicio por defecto, cuyo nombre es el mismo que se le asigna al proyecto. Para crear un nuevo proyecto, realice los siguientes pasos:

1. Seleccione **Archivo > nuevo > Proyecto**.
2. Especifique un nombre y una ubicación para el proyecto.



3. Seguidamente, presione el botón **Aceptar** para crear el proyecto.

4. Visualice el **Explorador de soluciones** para determinar los nombres que se asignaron tanto a la solución como al proyecto.



➤ **Guardar el proyecto**

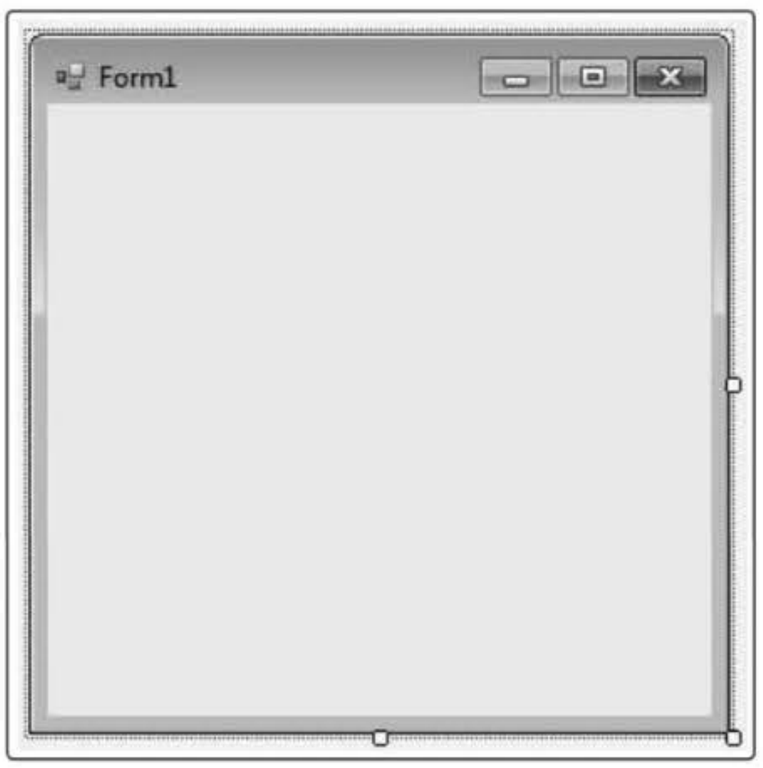
Se debe tener en cuenta que al guardar un proyecto, en realidad se están guardando todos los elementos contenidos en el mismo. Para guardar un proyecto, realice los siguientes pasos:

1. Seleccione **Archivo > Guardar todo**.
2. Especifique un nombre y una ubicación para el proyecto.

Considere que los proyectos se registran dentro de una carpeta por defecto llamada Projects, de la carpeta Visual Studio 2015, ubicada en Mis documentos de Windows.

2.13 Windows Forms

Cuando se crea un proyecto, de forma predeterminada, contiene un archivo Form1.cs. Este permite agregar controles a partir del cuadro de herramientas y, así, implementar una aplicación de plataforma o también llamada aplicaciones de escritorio mediante formularios de Windows.

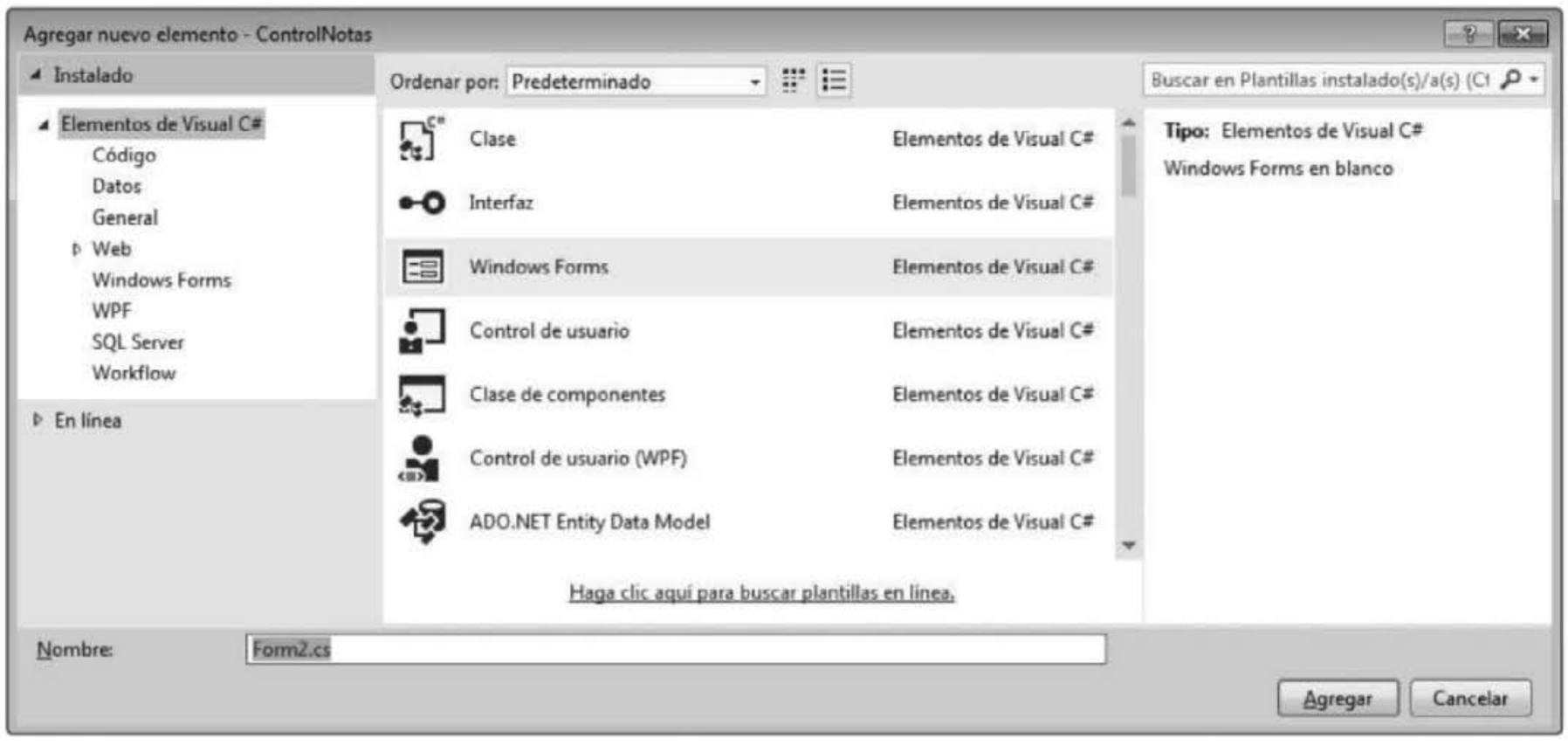


➢ Agregar un formulario al proyecto

Los formularios son entornos que permiten al usuario interactuar con la aplicación. Así, la cantidad de formularios que puede contener un proyecto es ilimitada. Por lo tanto, se pueden agregar o eliminar los formularios en cualquier momento. Para agregar un formulario, realice los siguientes pasos:

Primera forma:

1. Haga clic derecho sobre el **proyecto > Agregar > Windows Forms**.
2. Asigne un nombre al formulario y presione el botón **Agregar**.



Segunda forma:

1. Desde el menú principal, seleccione **Proyecto > Agregar Windows Forms**.
2. Asigne un nombre al formulario y presione el botón **Agregar**.

➢ Modificar el nombre del formulario

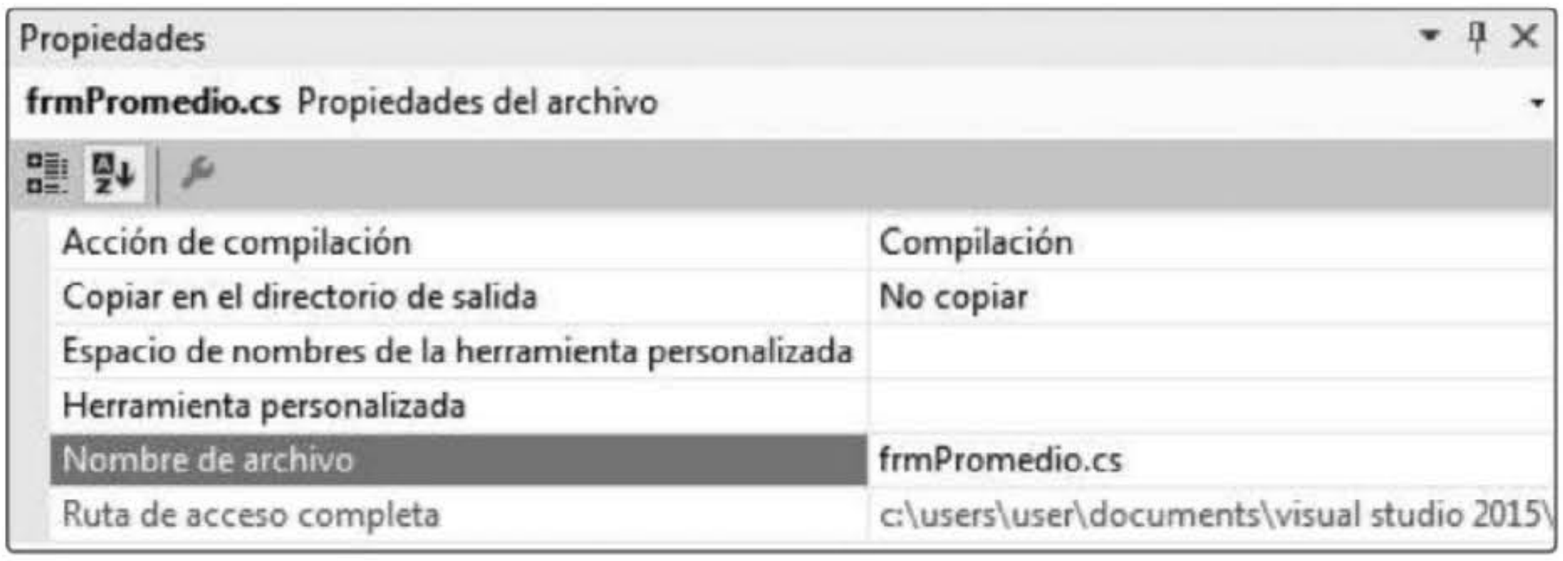
Se debe tener en cuenta que un formulario, dentro de un proyecto, representa dos cosas: la primera, un archivo, pues al guardar el proyecto, el formulario se registra en un archivo con extension .cs; la segunda, un objeto de la clase Form.

El formulario es uno de los controles que puede tener un nombre como archivo y otro nombre como control dentro de un mismo proyecto. Se recomienda que ambos nombres sean iguales pero la decisión es del programador.

Pasos para modificar el nombre del archivo Form1.cs:

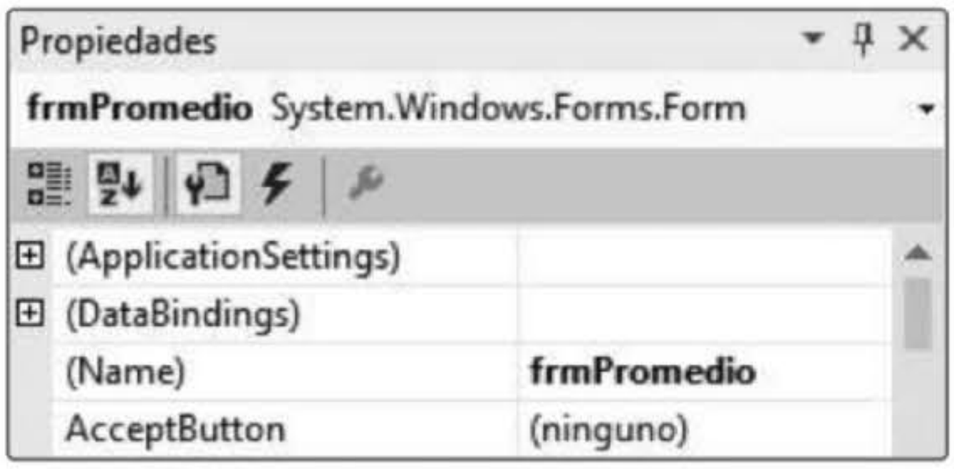
1. Seleccione el formulario desde el Explorador de soluciones.

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



Pasos para modificar el nombre del objeto Form1:

1. Haga clic dentro del contenedor del formulario.
2. Ubíquese en la propiedad **(Name)**, desde el panel de propiedades.
3. Asigne un nombre, tal como se muestra en la siguiente imagen:



➢ **Eliminar un formulario**

Cuando se elimina un formulario, también se eliminan todos los controles agregados. Pasos para eliminar el formulario Form1.cs:

- Seleccione el formulario desde el Explorador de soluciones.
- Haga clic derecho sobre el formulario y seleccione **Eliminar**.

➢ **Como establecer un formulario inicio dentro de un proyecto**

Cuando se ejecuta una aplicación de tipo Windows Forms, uno de los formularios es el que se muestra al usuario; por lo tanto, se debe tener el control de qué formulario es el ejecutado al presionar **F5** en Visual. Para modificar el formulario de inicio, realice los siguientes pasos:

1. Haga doble clic sobre el archivo **Program.cs**, desde el Explorador de soluciones.

2. Se muestra el siguiente código fuente del archivo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ControlNotas
{
    static class Program
    {
        /// <summary>
        /// Punto de entrada principal para la aplicación.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmPromedio());
        }
    }
}
```

3. Para cambiar el formulario de inicio, cambie el nombre del formulario en la siguiente sección de código:

```
Application.Run(new frmRegistro());
```

4. Finalmente, pruebe el cambio presionando **F5** para ejecutar la aplicación.

2.14 Controles visuales

Los controles visuales permiten crear entornos de desarrollo gráfico de forma profesional. Se debe tener en cuenta que toda aplicación está enfocada 100 % al usuario y no a la comodidad del programador; así, al momento de usar dichos controles, considérese punto.

En este punto, se aprende a colocar controles dentro de un formulario y a asignar las propiedades necesarias para una aplicación de escritorio.



<http://www.incanatoit.com/2014/05/sistema-de-almacen-en-visual-basic-net.html>

2.15 Notación para los nombres de los controles

Todo elemento que contiene un proyecto tiene un nombre y este puede ser modificado a conveniencia del programador. Es así, que existe una notación estándar que permite diferenciar el control, dependiendo de la clase de donde proviene. Véase la tabla de los prefijos más conocidos para los principales controles:

Ficha: **Controles comunes**

Control	Prefijo	Descripción
Form	frm	Windows Forms, es considerado como un contenedor de controles. Aquí, el programador diseña un entorno enfocado en el usuario final.
Button	btn	Botón de comando, se encarga de ejecutar un determinado proceso en una aplicación, como calcular, eliminar, listar, etc.
CheckBox	chk	Botón de chequeo, permite asignar cajas de chequeo que permiten al usuario seleccionar múltiples opciones en un mismo formulario.

ComboBox	cbo	Cuadro combinado, permite asignar varios valores en un solo control, los cuales pueden ser seleccionados por el usuario a partir de una selección desplegable.
DateTimePicker	dtp	Control de fecha y hora, permite al usuario seleccionar en un solo elemento una fecha y una hora.
Label	lbl	Etiqueta, se considera como texto estático que permite al programador mostrar al usuario información que no puede 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 ve en Excel.
MaskedTextBox	mtb	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. Se puede usar para el ingreso de fechas u horas en un formato adecuado.
MonthCalendar	mc	Mes de calendario, permite seleccionar una fecha y ocupar 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 puede ser seleccionado.
TextBox	txt	Cuadro de texto, permite ingresar un valor de cualquier tipo; los cuales son administrados por el programador.

Ficha: **Contenedores**

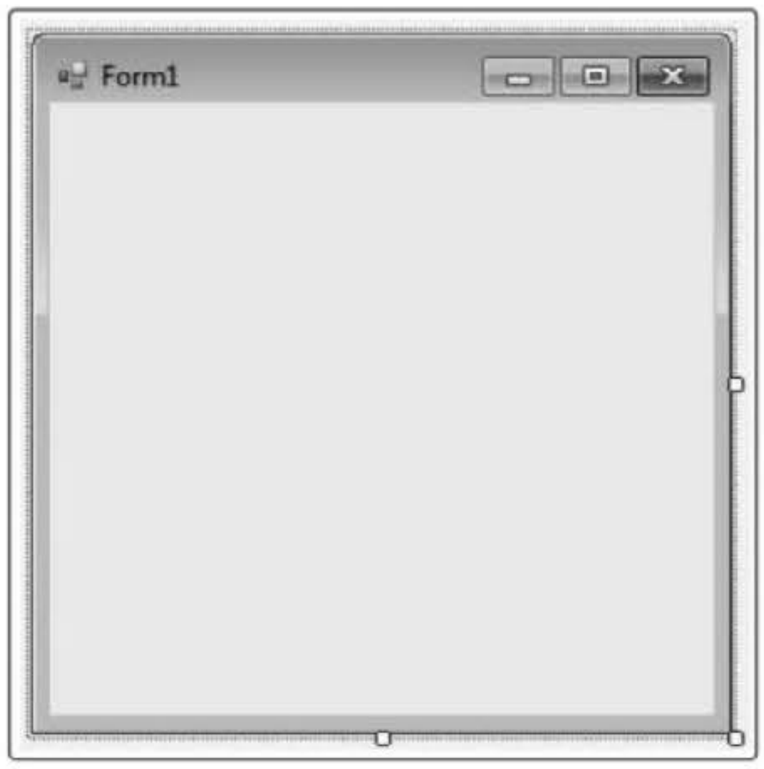
Control	Prefijo	Descripción
GroupBox	gb	Cuadro de grupo, permite agrupar controles dentro de un formulario.
TabControl	tc	Control tabulador, permite agrupar controles dentro de fichas.

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.

2.16 Control Form

Representa el objeto de la clase Form. Permite diseñar un formulario que interactúa con el usuario.



Propiedad	Descripción
(Name)	Permite asignar un nombre del control Form.
BackColor	Permite asignar un color de fondo al formulario.
BackgroundImage	Permite asignar una imagen de fondo al formulario.
BackgroundImageLayout	Permite definir cómo se muestra 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 ejecutar la aplicación.
Text	Determina el título del formulario.
WindowState	Determina el estado inicial del formulario al ejecutar la aplicación (maximizar, restaurar o minimizar).

Véase las siguientes cuestiones presentadas en un control de tipo formulario:

- **Asignar el nombre frmListado**
  1. Haga clic en el fondo del formulario.
  2. Modifique la propiedad **(Name) > frmListado**



- **Asignar el color rojo de fondo**
  1. Haga clic en el fondo del formulario.
  2. Seleccione la propiedad **BackColor > Personalizado.**
  3. Seleccione el color **Red** (rojo).



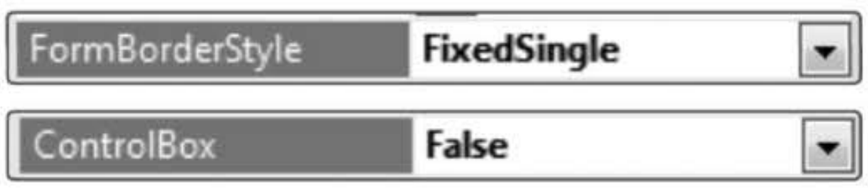
- **Asignar una imagen de fondo**
  1. Haga clic en el fondo del formulario.
  2. Seleccione la propiedad **BackgroungImage**.
  3. Seleccione **… > Recurso local > Importar**.
  4. Seleccione la imagen.
  5. Presione **Aceptar**.



- **Desactivar los botones de control**
  1. Haga clic en el fondo del formulario.
  2. Seleccione la propiedad **ControlBox > False**.



- **Cómo bloquear al formulario para que no cambie en alto y ancho**
  1. Haga clic en el fondo del formulario.
  2. Seleccione la propiedad **FormBorderStyle > FixedSingle**.
  3. Seleccione la propiedad **ControlBox > False**.



➢ **Asignar la posición central de la pantalla al ejecutar una aplicación**

1. Haga clic en el fondo del formulario.
2. Seleccione la propiedad **StartPosition > CenterScreen**.



➢ **Asignar el título Listado de empleados**

1. Haga clic en el fondo del formulario.
2. Seleccione la propiedad **Text** y asigne el texto **Listado de empleados**.



➢ **Asignar el estado de maximizado al ejecutar la aplicación**

1. Haga clic en el fondo del formulario.
2. Seleccione la propiedad **WindowState > Maximized**.



➢ **Cómo colocar controles dentro del formulario**

Primera forma:

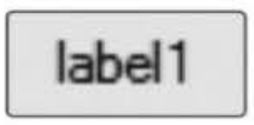
Arrastre un control desde el cuadro de herramientas hacia el formulario.

Segunda forma:

Haga doble clic sobre un control desde el cuadro de herramientas.

2.17 Control Label

Representa al objeto de la clase Label, la cual permite colocar textos estáticos dentro de un formulario.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto Label. Se debe tener en cuenta que si el objeto es un texto informativo para el usuario, no es 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 permita mostrar el sueldo como resultado de un proceso.
Autosize	Permite autoajustar el marco que contiene al texto del objeto Label. Si este se activa, el marco se autoajusta al contenido.

BorderStyle	Permite establecer un estilo de borde al objeto Label. Se tiene 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 del objeto Label dentro del formulario. Si se intenta realizar una de estas acciones, aparece un candado al costado del objeto Label.
Text	Permite asignar un contenido al objeto Label, este será visible por el usuario.
TextAlign	Permite asignar una alineación al texto contenido dentro del marco del objeto Label. Debe tener en cuenta que esta propiedad será visible solo si la propiedad AutoSize es False.

Véanse las siguientes cuestiones presentadas en un control de tipo Label:

- **Asignar el nombre lblSueldo**
  1. Seleccione el objeto **Label** desde el formulario.
  2. Modifique la propiedad **(Name) > lblSueldo**.



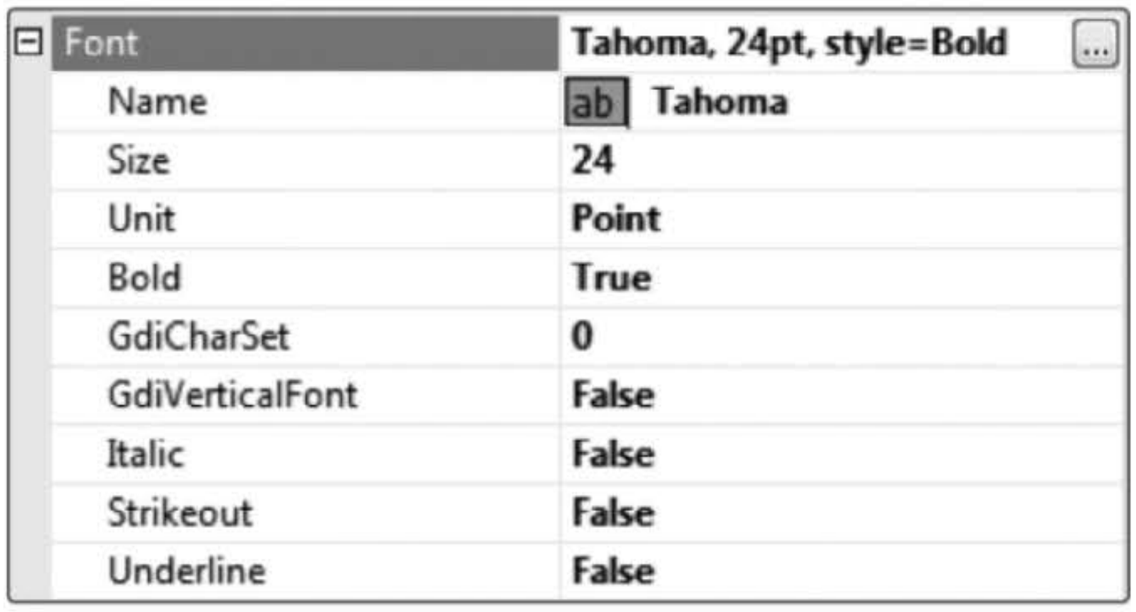
- **Asignar el color azul al texto**
  1. Seleccione el objeto **Label** desde el formulario
  2. Modifique la propiedad **Forecolor > Personalizado > Blue**.

- **Asignar un borde simple al marco del texto**
  1. Seleccione el objeto **Label** desde el formulario.
  2. Modifique la propiedad **BorderStyle > FixedSingle**.

- **Asignar el texto CONTROL DE SUELDO DE EMPLEADOS**
  1. Seleccione el objeto **Label** desde el formulario.
  2. Modifique la propiedad **Text** y coloque el texto **CONTROL DE SUELDO DE EMPLEADOS**.

➤ **Asignar el tipo de letra Tahoma con tamaño 24 y negrita**

1. Seleccione el objeto **Label** desde el formulario.
2. Modifique la propiedad **Font** > ...



2.18 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 puede ingresar un determinado valor requerido por la aplicación.



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 el ingreso de un nombre de usuario puede 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 como, por ejemplo, la fecha actual.
MaxLength	Define una longitud en caracteres como, por ejemplo, si se solicita el DNI de una persona se asigna 8 y, en el caso de ingresar una edad, se asigna 2.
Multiline	Permite activar la escritura en varias líneas dentro del mismo objeto TextBox. Sirve para ingresar valores extensos como un comentario.
PasswordChar	Permite asignar un carácter enmascarado al texto ingresado. Es similar a lo empleado en las claves solicitadas en los correos electrónicos.
ReadOnly	Permite bloquear la modificación del texto contenido en el objeto TextBox.
ScrollBars	Permite asignar barras de desplazamiento cuando el objeto TextBox permita ingresar varias líneas.
Text	Permite asignar un valor al control TextBox. No se recomienda su ingreso ya que es el usuario el que debe ingresar un valor.
TextAlign	Permite alinear el texto contenido en un objeto TextBox.

Véase las siguientes cuestiones presentadas en un control de tipo TextBox:

- **Asignar el nombre txtNombres para ingresar el nombre del usuario**
  1. Seleccione el objeto **TextBox** desde el formulario.
  2. Modifique la propiedad **(Name) > txtNombres**.



- **Asignar una longitud de 8 caracteres**
  1. Seleccione el objeto **TextBox** desde el formulario.
  2. Modifique la propiedad **MaxLength > 8**.



- **Asignar la máscara asterisco al objeto TextBox**
  1. Seleccione el objeto **TextBox** desde el formulario.
  2. Modifique la propiedad **PasswordChar > ***.



- **Permitir el ingreso de texto en varias líneas y asignar la barra de desplazamiento vertical**
  1. eleccione el objeto **TextBox** desde el formulario.
  2. Modifique la propiedad **Multiline > True**.
  3. Modifique la propiedad **ScrollBars > Vertical**.



- **Mostrar el valor 18 % de solo lectura**
  1. Seleccione el objeto **TextBox** desde el formulario
  2. Modifique la propiedad **Text > 18 %**.
  3. Modifique la propiedad **ReadOnly > True**.



2.19 Control Button

Representa al objeto de la clase Button, el cual permite realizar un proceso según el código implementado.



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto Button. Por ejemplo, para un botón de registro puede llamarse btnRegistrar.
Enabled	Permite habilitar o inhabilitar el botón, según 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.

Véase las siguientes cuestiones presentadas en un control de tipo Button:

- **Asignar el nombre btnRegistrar al botón**

  1. Seleccione el objeto Button desde el formulario.
  2. Modifique la propiedad **(Name) > btnRegistrar**.



- **Asignar el texto REGISTRO al botón**

  1. Seleccione el objeto **Button** desde el formulario.
  2. Modifique la propiedad **Text > REGISTRO**.



- **Asignar el texto SALIR y asignar una imagen con las alineaciones respectivas**



1. Seleccione el objeto **Button** desde el formulario.
2. Modifique la propiedad **Text > SALIR**.
3. Modifique la propiedad **Image > … > Recurso local > Importar**.
4. Modifique la propiedad **ImageAlign > MiddleLeft**.
5. Modifique la propiedad **TextAlign > MiddleRight**.

Image	System.Drawing...
ImageAlign	**MiddleLeft**

Text	**SALIR**
TextAlign	**MiddleRight**

2.20 Control ComboBox

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



Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ComboBox. Por ejemplo, para un cuadro combinado para listar países, puede llamarse cboPais.
DropDownStyle	Permite modificar el estilo que muestra el cuadro combinado. Si se asigna Simple, se muestran los valores sin contraerse. DropDown es el valor estándar y DropDownList no permite editar dentro del objeto.
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.

Véase las siguientes cuestiones presentadas en un control de tipo ComboBox:

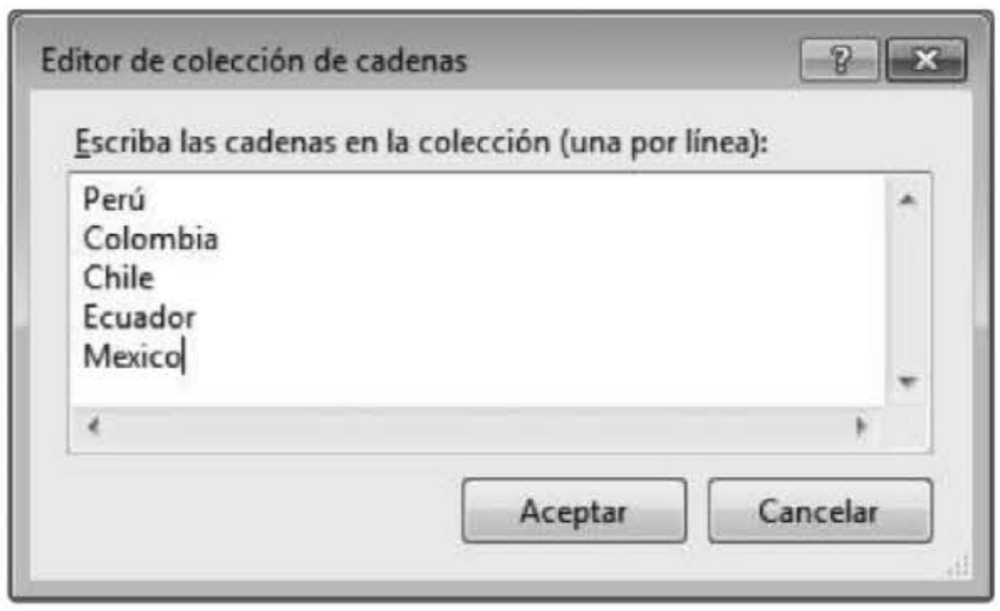
- **Asignar el nombre cboPais**

1. Seleccione el objeto **ComboBox** desde el formulario.
2. Modifique la propiedad **(Name) > cboPais**.

(Name)	**cboPais**

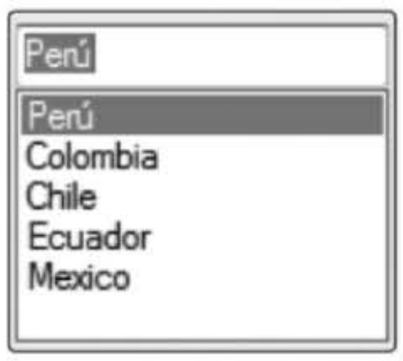
➤ **Asignar los siguientes países al cuadro combinado Perú - Colombia - Chile - Ecuador - México**

1. Seleccione el objeto **ComboBox** desde el formulario.
2. Modifique la propiedad **Items > … > Digite los países > Aceptar**.



➤ **Mostrar los países con el estilo de combo Simple**

1. Seleccione el objeto **ComboBox** desde el formulario.
2. Modifique la propiedad **DropDownStyle > Simple**.



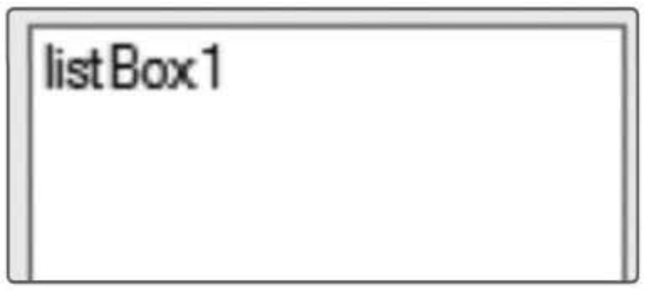
➤ **Mostrar el texto inicial (Seleccione un país)**

1. Seleccione el objeto **ComboBox** desde el formulario
2. Modifique la propiedad **Text > Seleccione un país**.



2.21 Control ListBox

Representa al objeto de la clase ListBox, el cual permite mostrar una lista de valores al usuario.



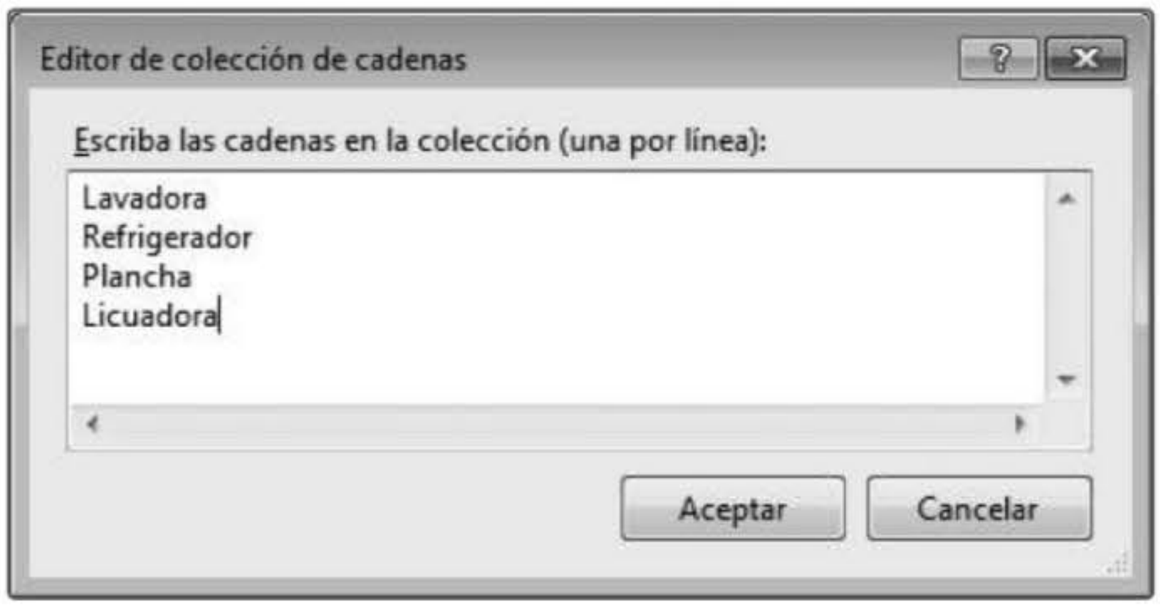
Propiedad	Descripción
(Name)	Permite asignar un nombre al objeto ListBox. Por ejemplo para una lista de productos podría llamarse lstProductos.
Items	Permite asignar valores dentro de la lista.

Véase las siguientes cuestiones presentadas en un control de tipo ComboBox:

- **Asignar el nombre lstProductos**
  1. Seleccione el objeto **ListBox** desde el formulario.
  2. Modifique la propiedad **(Name) > lstProductos**.



- **Asignar los siguientes productos a la lista Lavadora – Refrigerador – Plancha - Licuadora**
  1. Seleccione el objeto **ListBox** desde el formulario.
  2. Modifique la propiedad **Items > … > Digite los productos > Aceptar**.



2.22 Control ListView

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



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

Véase las siguientes cuestiones presentadas en un control de tipo Button:

- **Asignar el nombre lvRegistro al ListView**

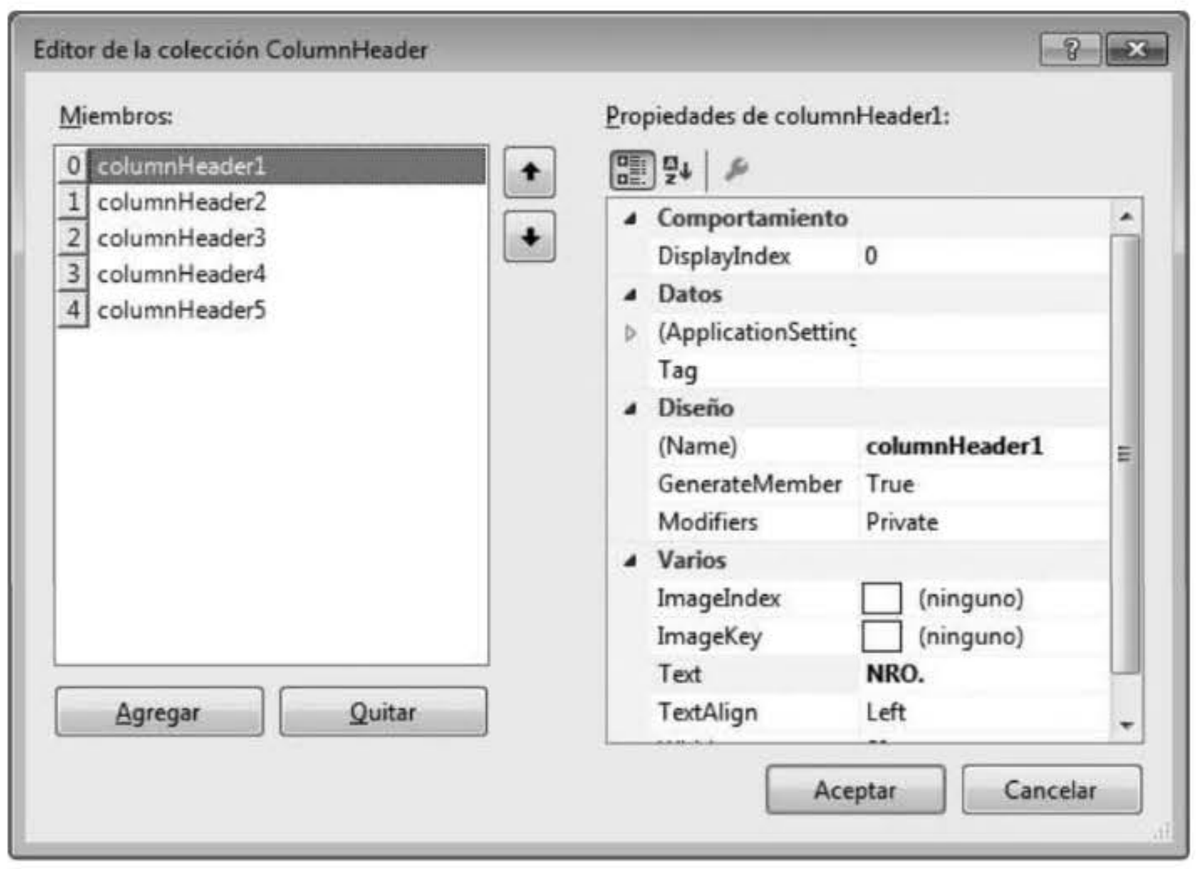
1. Seleccione el objeto **ListView** desde el formulario
2. Modifique la propiedad **(Name) > lvregistro**



- **Implementar el siguiente ListView**



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

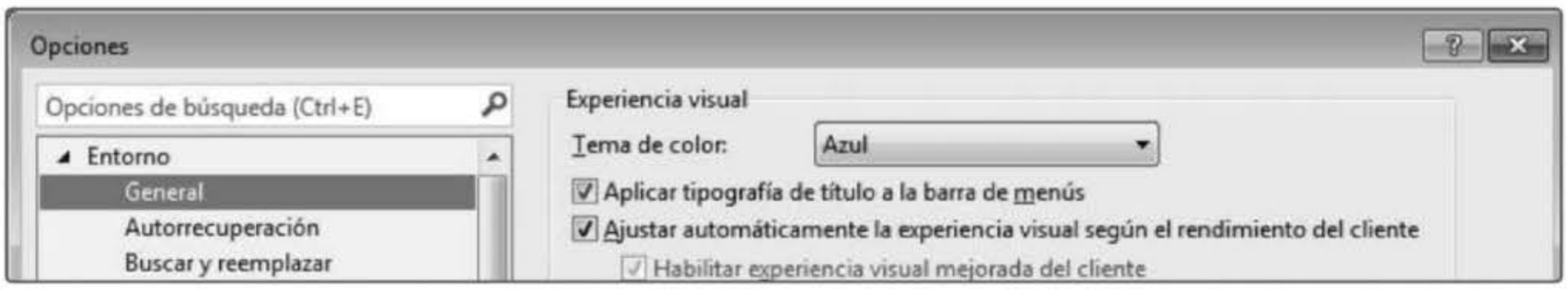


5. Para agregar las cabeceras, primero deben crearse cinco columnas. Para esto, presione cinco veces sobre el botón **Agregar**. Seleccione uno a uno los miembros mostrados y, en la propiedad **Text** mostrada en la misma ventana, modifique la cabecera. Cuando termine con todas las columnas, presione el botón **Aceptar**.

Preguntas frecuentes

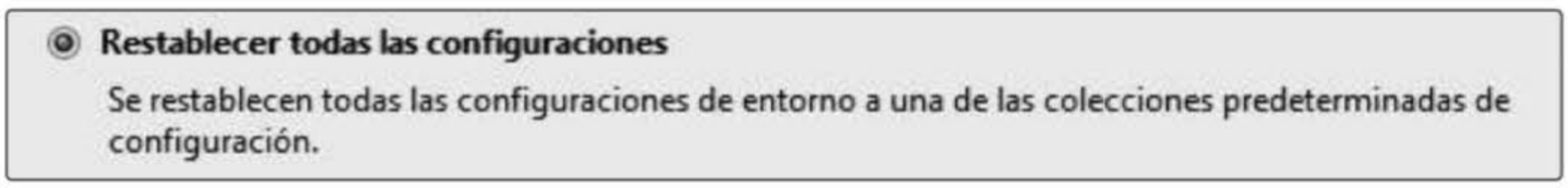
➢ **¿Cómo se puede modificar el color del tema para el entorno principal?**

- **Herramientas > Opciones**.
- Seleccione el color de su preferencia desde la opción **Tema de color**.

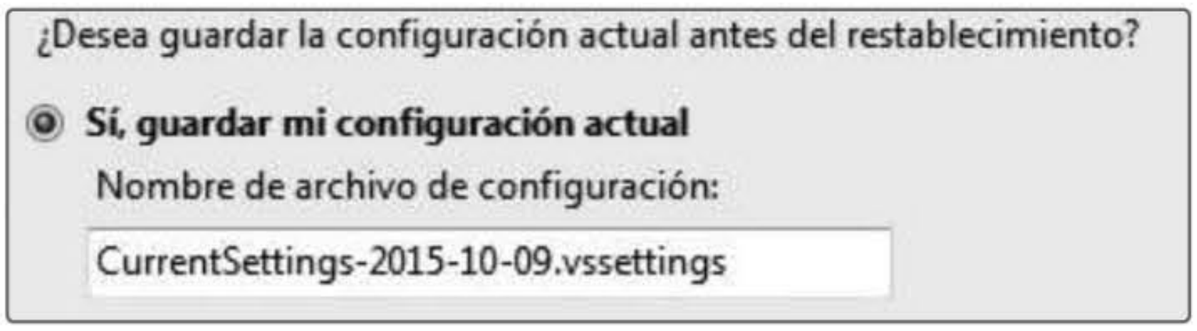


➢ **¿Cómo se puede modificar el lenguaje de programación predeterminado en Visual Studio 2015?**

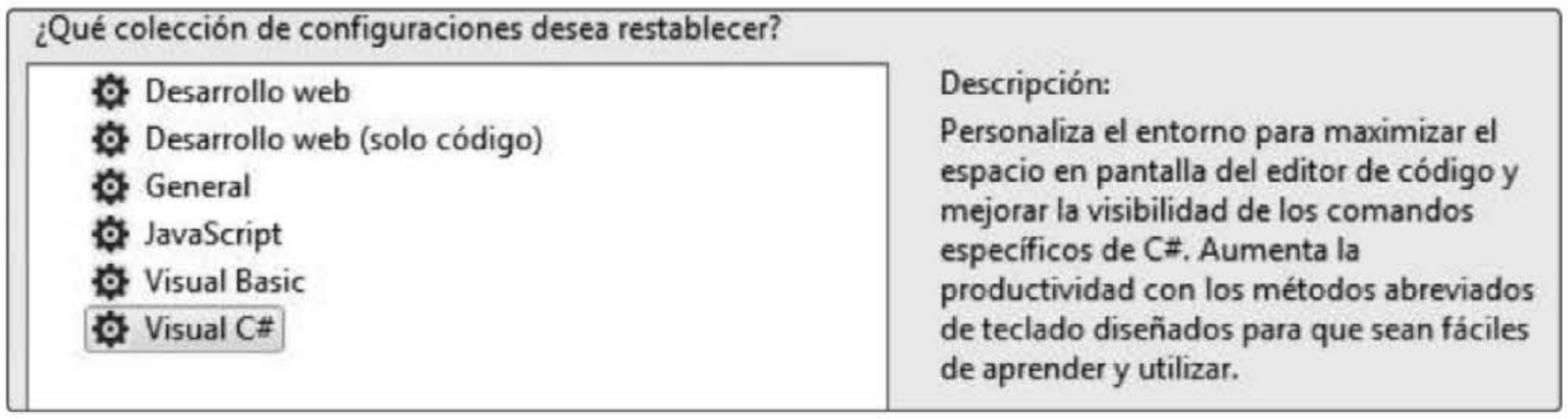
- **Herramientas > Importar y exportar configuraciones…**
- En la primera ventana, seleccione **Restablecer todas las configuraciones**.



- En la segunda ventana, seleccione **Sí, guardar mi configuración actual**.



- En la tercera ventana, seleccione su lenguaje de programación.



- Finalmente, presione el botón **Finalizar**.

Para recordar

- Visual Studio 2015 es considerado como un entorno de desarrollo completo e integrado que permite implementar aplicaciones en las principales plataformas como son Windows, Android y iOS.
- La plataforma .NET proporciona un entorno de desarrollo orientada al trabajo de objetos; que mejora en rendimiento frente a los diferentes dispositivos donde puede ejecutar una aplicación, sin importar en qué lenguaje lo desarrolle.
- La plataforma .NET cuenta con un CLR y una biblioteca completa de clases que lo hace diferente a las demás plataformas de desarrollo.
- Windows Forms permite desarrollar aplicaciones de plataforma y es la mejor manera de iniciar con el proceso de aprendizaje de Visual Studio 2015.
- UWA (Universal Windows App) permite desarrollar aplicaciones universales, las cuales pueden ser ejecutadas en cualquier dispositivo sin ningún problema.
- NET Framework no permitía la portabilidad, sino más bien que era administrado de otra manera. Lo que diferencia la nueva versión es que cuenta con una funcionalidad compartida para las tareas clave como, por ejemplo, leer archivos locales. Su característica principal se puede notar al hacer uso de herramientas de terceros como IOS o Android.
- La biblioteca de clase de .NET es una biblioteca completa de clases, interfaces y tipos de valor, que proporcionan acceso a la funcionalidad de una aplicación.

- Las versiones del Visual Studio son:

  Visual Studio 97
  Visual Studio 6.0
  Visual Studio .NET
  Visual Studio .NET 2003
  Visual Studio .NET 2005
  Visual Studio 2008
  Visual Studio 2010
  Visual Studio 2012
  Visual Studio 2013
  Visual Studio 2015

- La URL para descargar Visual Studio 2015 es
  https://www.visualstudio.com/downloads/download-visual-studio-vs.
- La pantalla principal del Visual Studio debe contar con la barra de herramientas estándar, Cuadro de herramientas (ALT + CTRL + X), Explorador de soluciones (CTRL + ALT + L) y la ventana de propiedades (F4).
- Una solución para Visual Studio es un contenedor de proyectos.
- Un proyecto es una aplicación que permite realizar un proceso de negocio determinado. Entre los lenguajes más importantes que presenta Visual Studio están C# y Visual Basic.

Capítulo 3

Las variables y los operadores

Capacidad terminal:

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

Contenido:

- Introducción
- Estructura de una aplicación de plataforma en Visual C#
- Eventos de los controles
- Comentarios en el código fuente
- Tipos de datos para Visual C#
- Identificadores
- Palabras reservadas
- Literales
- Las variables
- Ámbito de las variables en C#
- Operadores
- Prioridad de los operadores aritméticos
- Métodos matemáticos de.Net Framework
- Conversiones de tipo en C#
- Conversión implícita
- Conversión explícita
- Conversión definida por el usuario
- Conversión mediante el método parse
- Conversión mediante la clase Convert
- Conversión implícita de tipos
- Clase String
- La clase MessageBox
- Casos desarrollados
  - o Caso desarrollado 1: pago de empleados
  - o Caso desarrollado 2: control de medidas
  - o Caso desarrollado 3: control de área y volumen
- Para recordar
- Preguntas frecuentes

Introducción

Hasta el momento se ha tratado todo lo referente al proyecto que se puede implementar en Visual Studio. Antes de realizar aplicaciones con C# dentro de Visual Studio, es necesario entender algunas notaciones básicas como la declaración de variables, operadores, implementar expresiones y manejar los cuadros de mensaje en las aplicaciones.

Una de las principales actividades al momento de realizar código es la correcta declaración de variables, se debe tener en cuenta que primero se necesita identificar cuántas variables serán necesarias para la aplicación, así como definir el valor que contendrá dicha variable.

Asimismo, se deben reconocer correctamente los operadores que usaremos en Visual C#. Ya que en cada sentencia siempre se usa operadores, primero se debe entender su concepto y la forma de usarlo en una aplicación.

Finalmente, se verá cómo implementar mensajes dentro de una aplicación, pues es muy importante informar al usuario sobre algún evento en la aplicación o solicitar alguna información.

3.1 Estructura de una aplicación de plataforma en Visual C#

Se debe tener en cuenta que los códigos a desarrollar están enfocados en aplicaciones de plataforma, por lo tanto, se debe conocer la estructura de dicha aplicación.

Cuando se crea un proyecto para Visual C#, automáticamente se agrega un formulario por defecto llamado "form1"; es aquí donde se deben agregar los controles necesarios para la aplicación. Pero se debe tener en cuenta que no solo consiste en diseñar un entorno, llamado interfaz gráfica de usuario (GUI), sino que se debe agregar un código para que su funcionamiento sea completo. Entonces, para crear una aplicación de plataforma en Visual C#, se deben tener en cuenta los siguientes pasos:

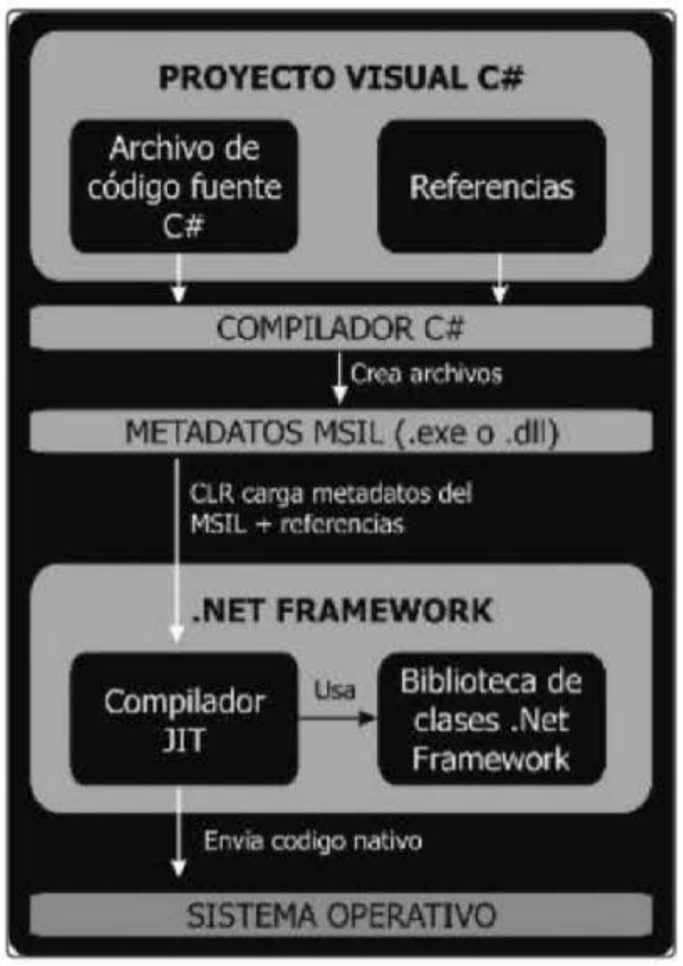
1. Crear un nuevo proyecto o solución en Visual Studio 2015:

- Seleccione **Archivo > Nuevo > Proyecto**.
- Seleccione la plantilla **Visual C#**.
- Asigne una ubicación y un nombre al proyecto.

2. Diseñar la interfaz gráfica de usuario usando un Windows Form:

- Asigne un nombre adecuado al formulario.
- Agregue los controles desde el cuadro de herramientas hacia el formulario.
- Mediante las propiedades, modifique dichos controles hasta tener una interfaz adecuada al caso a desarrollar.
- Agregue un código a los botones.
- Ejecute la aplicación.

Para que una aplicación se considere correctamente compilada y ejecutada, debe pasar por las siguientes etapas:



3.2 Eventos de los controles

Un evento es considerado como una ocurrencia del control. Se debe tener en cuenta que todos los controles tienen eventos y que pueden ser manipulados por el programador; es decir, se puede programar dentro de un evento para que se accione justamente cuando ocurra dicho evento.

Véase una lista de los principales eventos que pueden tener algunos controles:

➢ Control Button

Evento	Descripción
Click	Es el evento que por defecto tiene el botón; ya que todos los botones suelen accionarse por el clic de un mouse y estos ejecutan o realizan una tarea específica.

El código mostrado en editor se presenta de la siguiente manera:

```
private void btnProcesar_Click(object sender, EventArgs e)
{
   //Código a implementar
}
```

La visibilidad del método es privado. Por esto, las variables declaradas, así como los objetos usados dentro del botón; no pueden ser referencias fuera del mismo. Void es un tipo de método que tiene como característica no devolver un valor específico, por eso el término vacío.

btnProcesar_Click es la unión del nombre asignado al objeto, así como la especificación del evento seleccionado. Es decir, el botón procesar acciona algún código al hacer clic. Los parámetros que presenta entre paréntesis tienen la siguiente funcionalidad:

- **Sender:** Representa una referencia al objeto que lanza el evento actual.
- **EventArgs:** Representa la clase base de donde proviene el evento. En el caso de un botón, el clic del mouse lo puede accionar.

La pareja de llaves representa el inicio y fin del código a implementar. Este código es ejecutado al momento de accionar el evento haciendo clic sobre el botón **Procesar**.

➢ Control ComboBox

Evento	Descripción
SelectedIndexChanged	Es el evento por defecto que tiene el cuadro combinado. Este evento permite accionar un código con solo escoger un elemento contenido en la lista.

El código mostrado en el editor de código sería de la siguiente manera:

```
private void cboProductos_SelectedIndexChanged(object sender, EventArgs e)
{
  //Código a implementar

}
```

cboProductos_SelectedIndexChanged especifica el nombre del control y su evento. Se debe tener en cuenta que por cada selección de un elemento en el cuadro combinado se acciona el código programado dentro del evento.

Cuando los eventos de los objetos no son los predeterminados, se necesita realizar los siguientes pasos:

- Seleccione el objeto desde el formulario.
- Seleccione el botón **Eventos** desde la ventana de propiedades  .
- Ubique el evento y haga doble clic sobre el mismo.

➢ Control ListView

Evento	Descripción
MouseDoubleClick	Es el evento que responde al hacer doble clic sobre una de las filas mostradas en el ListView.

El código mostrado en editor es de la siguiente manera:

```
private void lvRegistro_MouseDoubleClick(object sender, MouseEventArgs e)
{
  //Código a implementar
}
```

lvregistros_MouseDoubleClick es el evento que permite accionar el código programado en el evento cuando se hace doble clic sobre una fila mostrada en el objeto ListView.

➢ Editor de código

En una aplicación Windows Forms se debe tener acceso al código para poder administrar de la mejor manera los controles del formulario. Se puede ingresar al editor de código de varias maneras:

Primera forma: Evento predeterminado

Haciendo doble clic sobre cualquier control contenido en el formulario
Por ejemplo, si se realiza en un botón, el editor de código muestra el siguiente código:

```
PRIVATE VOID BUTTON1_CLICK(OBJECT SENDER, EVENTARGS E)
{
            //CÓDIGO A IMPLEMENTAR
}
```

Segunda forma: Evento personalizado

Haciendo clic sobre cualquier control contenido en el formulario:

- Seleccione **Eventos** de la ventana de propiedades.
- Haga doble clic en el evento seleccionado.
- Por ejemplo, si se realiza en un Listview con el evento MouseDoubleClick, el editor de código muestra el siguiente código:

```
PRIVATE VOID LVREGISTRO_MOUSEDOUBLECLICK(OBJECT SENDER, MOUSEEVENTARGS E)
{
  //CÓDIGO A IMPLEMENTAR
}
```

El código completo dentro del editor se muestra a continuación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjEjercicio01
{
 public partial class frmPago : Form
 {
   public frmPago()
   {
      InitializeComponent();
   }
   private void Button1_Click(object sender, eventargs e)
   {
      //código a implementar
   }
   private void ListView1_MouseDoubleClick(object sender,mouseeventargs e)
   {
//código a implementar
   }
 }
}
```

Como se observa, el editor de código muestra la estructura básica de una aplicación de tipo Windows Form. Se recomienda no modificar las librerías necesarias para la aplicación. Conforme se usan más funciones, se agregan más librerías. La siguiente lista son las librerías mínimas necesarias para la aplicación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

Donde:

- namespace pjEjercicio01 define un ámbito que representa a un conjunto de objetos pertenecientes al proyecto pjEjercicio01.
- public partial class frmPago : Form representa la declaración de la clase frmPago, la cual hereda atributos y métodos de la clase Form.
- public frmPago() representa el método constructor de la clase frmPago, el cual tiene como misión invocar el método InitializeComponent(), que permite mostrar todos los controles añadidos al formulario.

3.3 Comentarios en el código fuente

Cuando se ejecuta una aplicación en Visual, el compilador reconoce las sentencias especificadas en el editor de código y las convierte en un código entendible por la computadora. Al agregar comentarios en la aplicación, estas NO son interpretadas por el compilador.

Un comentario es una sentencia no reconocida por el compilador y esta se puede colocar en el código en estas situaciones:

- Colocar el nombre del autor del código.
- Colocar un mensaje sobre las sentencias implementadas.
- Anular una o más sentencias de código.

Véase el formato para la asignación de comentarios:

Primera forma: Comentario de una sola línea

- Ubique el cursor en una línea de código desde el editor.
- Digite dos barras inclinadas (//).

Segunda forma: Comentario de varias líneas

- Ubique el cursor en una línea de código desde el editor.
- Digite una barra y un asterisco (/*) para dar inicio al bloque de comentarios.
- Digite un asterisco y una barra (*/) para finalizar el bloque de comentarios.

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

```
/*
   Autor: Lic. Manuel Torres R.
   Método que permite determinar el monto subtotal de una
   determinada venta de producto.
*/
private void btnProcesar_Click(object sender, EventArgs e)
{
   //Asignando los valores del producto
   string producto = "Agua Mineral";
   double precio = 2.50;
   int cantidad = 10;

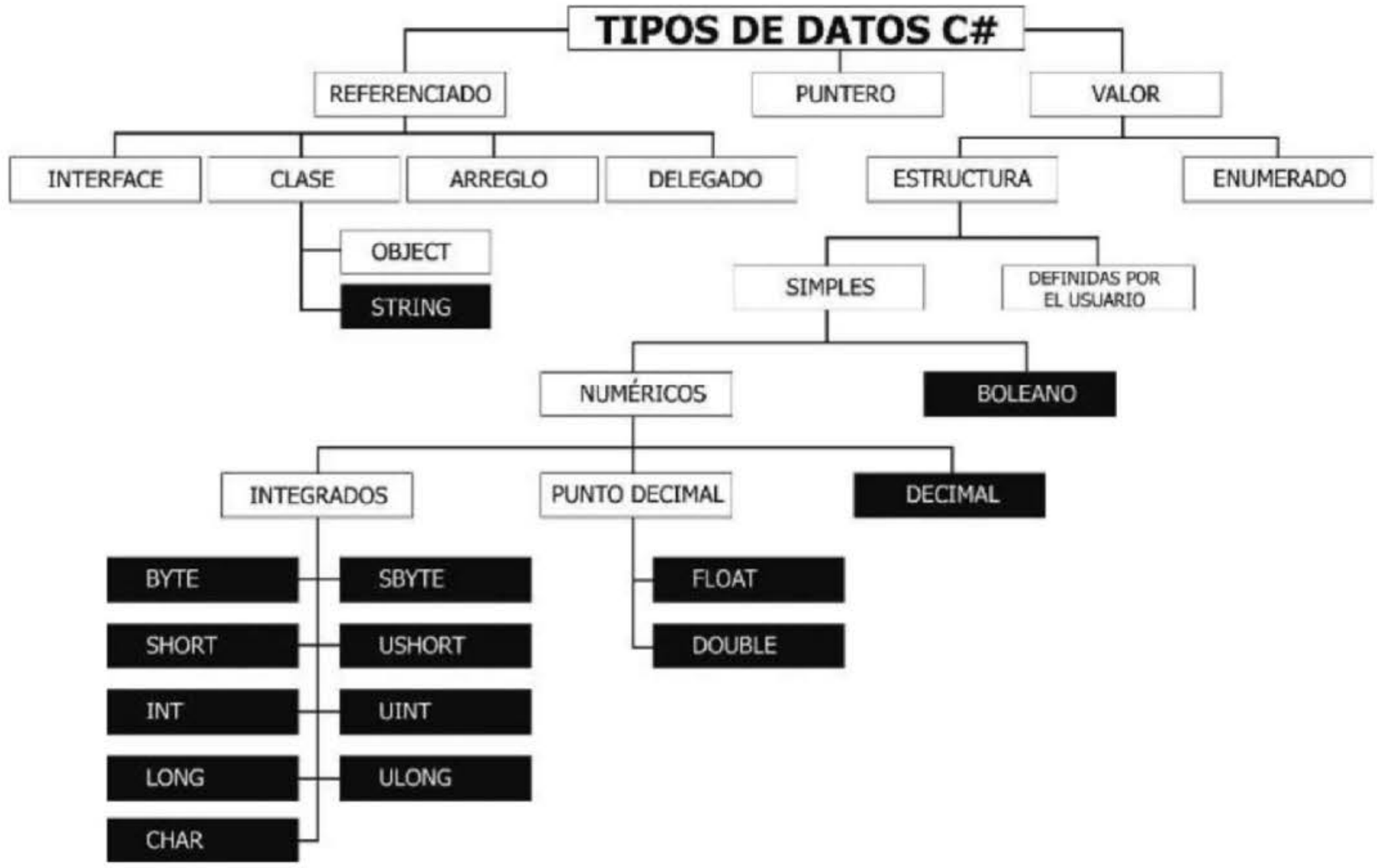
   //Realizando los calculos
   double subtotal = precio * cantidad;

   //Realizando la impresion
   MessageBox.Show("PRODUCTO   : " + producto);
   MessageBox.Show("SUBTOTAL   : $ " + subtotal.ToString("0.00"));
}
```

3.4 Tipos de datos para Visual C#

C# es considerado un lenguaje de programación estrictamente tipado, esto implica que se declaren todas las variables que se piensa usar, al igual que los objetos que se crean.

Véase su clasificación a continuación:



Capacidad de los tipos numéricos:

Tipo	Tamaño	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
int	4 bytes	-2147483648 a 2147483647	Int32
uint	4 bytes	0 a 4294967295	UInt32
long	8 bytes	-9223372036854775808 a 9223372036854775807	Int64
ulong	8 bytes	0 a 18446744073709551615	UInt64

Capacidad de los tipos numéricos decimales:

Tipo	Tamaño	Capacidad	System de .NET
float	4 bytes	±1.401298E−45 a ±3.402823E+38	Single
double	8 bytes	±4.94065645841246E−324 a ±1.79769313486232E+308	Double
decimal	16 bytes	−7.9228162514264337593543950335 a +7.9228162514264337593543950335	Decimal

Capacidad de los tipos caracteres:

Tipo	Tamaño	Capacidad	System de .NET
char	2 bytes	'\u0000' a '\uFFFF'	Char

Capacidad de los tipos boolean:

Tipo	Tamaño	Capacidad	System de .NET
bool	1 byte	true o false	Boolean

Capacidad del tipo de cadena de caracteres:

Tipo	Tamaño	Capacidad	System de .NET
string	Variable	El permitido por la memoria	String

Capacidad del tipo de fecha:

Tipo	Tamaño	Capacidad	System de .NET
DateTime	4 bytes	{00:00:00AM} equivale a {12:00:00AM}, medianoche. {00:00:00PM} equivale a {12:00:00PM}, mediodía. {00:00:00} a {11:59:59} equivale a {12:00:00AM} a {11:59:59AM} {12:00:00} a {23:59:59} equivale a {12:00:00PM} a {11:59:59PM}	DateTime

3.5 Identificadores

Son considerados como símbolos léxicos que permiten asignar un nombre a una entidad. En este caso, se usa para la identificación de los diversos elementos que puede tener una aplicación en Visual C# como variables, atributos de clase, constantes, métodos, clases, estructuras, etc.

Antes de crear un identificador, se deben considerar 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, se pueden usar letras, números, guion bajo.
- No debe contener espacios en blanco.
- No debe contener caracteres especiales.
- No debe ser una palabra reservada del lenguaje.
- Considere que C# distingue entre mayúsculas y minúsculas, es decir, no son iguales, a pesar de sonar idéntico, por eso se debe considerar Case Sensity.
- Puede contener 511 como longitud máxima de caracteres.

En el siguiente proceso de negocio, se hará una lista de identificadores válidos y no validos:

«Una tienda comercial ha puesto en oferta la venta de un producto ofreciendo un 10 % de descuento sobre el importe de la compra. Se pide determinar el importe de la compra, el importe del descuento y el importe a pagar por la compra de cierta cantidad de productos del mismo tipo».

Valores de entrada y salida	Posibles identificadores (válido)	Identificadores no válidos
Nombre del producto	producto, nomProducto	@producto_
Cantidad comprada	cantidad, unidades, cant	_ cantidad,
Precio del producto	precio, precio_producto, costo	precio del producto
Importe de compra	importeCompra, iCompra	iCompr@
Importe de descuento	importeDescuento, iDescuento	10 % Descuento
Importe a pagar	importePagar, iPagar	Importe a Pagar

3.6 Palabras reservadas

Son identificadores previamente definidos y reservados por el lenguaje los cuales tienen un significado especial para el compilador. Estos no deben usarse como identificadores tanto en variables como en objetos.

Veamos su clasificación a continuación:

abstract	continue	extern	in	Object	return	this	using
as	decimal	false	int	operador	sbyte	throw	virtual
base	default	finally	interface	out	sealed	true	void
bool	delegate	fixed	internal	override	short	try	volatile
break	do	float	is	params	sizeof	typeof	while
byte	double	for	lock	private	stackalloc	uint	
case	else	foreach	Long	protected	static	ulong	
catch	enum	goto	namespace	public	string	unchecked	
char	Event	if	new	readonly	struct	unsafe	
checked	explicit	implicit	null	ref	switch	ushort	

3.7 Literales

Representa un determinado valor según el tipo de dato. Véase la siguiente tabla con los posibles literales de los tipos de datos más usados:

Tipo de datos	Literal	Literal personalizado
int	10	10
double	23.5D	23.5
decimal	780.50M	780
long	120L	120
char	'A'	'A'
String	"Visual C#"	"Visual C#"
bool	true	true
DateTime	"10/10/2015"	"10/10/2015"

Se debe tener especial cuidado en el uso del tipo decimal pues soporta literales con valor entero y decimal pero agregando la letra M mayúscula al final; de otra forma no acepta dichos valores. Con respecto al literal DateTime, no se puede asignar directamente un valor fecha; ya que obligatoriamente debe encerrarse entre comillas. Por lo tanto, se debe parsear su contenido, por ejemplo:

```
DateTime fechaNac = DateTime.Parse("10/10/2015");
```

Si se trata con cadena de caracteres, entonces se debe considerar hacer referencia a caracteres especiales.

Véase una lista de literales válidos para el tipo String:

Literal	Descripción
`\' apostrofe`	Representa la comilla simple en una cadena de caracteres.
`\" Comillas`	Representa las comillas dobles en una cadena de caracteres.
`\\ Backslash`	Representa la barra inclinada en una cadena de caracteres.
`\n Nueva línea`	Representa una nueva línea en una cadena de caracteres.
`\t Tabulación Horizontal`	Representa una tabulacion en una cadena de caracteres.

3.8 Las variables

Como se mencionó, C# es un lenguaje tipado y eso significa que todo elemento que se use en el código fuente debe ser declarado de forma obligatoria, respetando las reglas de los identificadores y literales vistos en este capítulo.

Una variable es un espacio de memoria designado a solicitud del programador. 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.

Formato de declaración de variable:

TIPODATO VARIABLE;

Donde:

- **TIPODATO:** Representa un tipo de datos. Antes de asignar un tipo, primero se debe pensar en el valor que almacena la variable y revisar la tabla de tipo de datos aceptados por C#.
- **VARIABLE;:** Representa el nombre de la variable. Si se necesita especificar más de una variable, se pueden separar por medio de comas.

Véase un proceso de negocio en el cual se pueden declarar las variables que se necesita:

«Una editorial vende un libro cuyo costo unitario es $ 120.50. Como oferta, la editorial ofrece un descuento fijo del 12.5 % del importe de la compra. Adicionalmente, la tienda obsequia 1 calendario por cada docena de productos adquiridos. Declare las variables necesarias para que determine el importe de la compra, el importe del descuento, el total de calendarios y el importe neto a pagar por la compra de cierta cantidad de libros».

Valores de entrada y salida	Declaración de variable
Nombre del libro	string libro;
Cantidad comprada	int cantidad;
Precio unitario	double pUnitario;
Importe de compra	double iCompra;
Importe de descuento	double iDescuento;
Total de calendarios	int totalCalendarios;
Importe neto a pagar	double iNeto;

Se puede resumir la declaración en los siguientes grupos:

```
string libro;
int cantidad, totalCalendarios;
double pUnitario, iCompra, iDescuento, iNeto;
```

3.9 Ámbito de las variables en C#

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, globales y estáticas.

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 contiene dicha variable se pierde cuando el bloque finaliza. Véase el código de una aplicación Windows Forms donde se pueden declarar variables locales:

```
using System;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjEjercicio01
{
 public partial class frmPago : Form
 {
```

```
    public frmPago()
    {
       InitializeComponent();
    }

    private void Button1_Click(object sender, eventargs e)
    {
       //Declaracion de variables locales en un objeto de tipo Button
    }

    void metodo1()
    {
       //Declaracion de variables locales en un metodo sin valor de retorno
    }

    double metodo2()
    {
       //Declaracion de variables locales en un metodo con valor de retorno
    }
  }
}
```

Una variable se considera global cuando se define fuera de los bloques y ocupa una posición en la memoria de manera fija. Su característica principal es que su valor puede ser modificado en cualquier bloque de código en cualquier tiempo.

Véase el código de una aplicación Windows Forms donde se pueden declarar variables globales:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjEjercicio01
{
 public partial class frmPago : Form
 {
   //Declaracion de variables GLOBALES
   public frmPago()
   {
      InitializeComponent();
   }
 }
}
```

Finalmente, las variables declaradas estáticas son valores permanentes y fijos dentro de la memoria. Estas pueden ser declaradas en un bloque de la aplicación o en la sección de declaración de globales.

Véase el código de una aplicación Windows Forms donde se pueden declarar variables estáticas:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjEjercicio01
{
 public partial class frmPago : Form
 {
   //Declaracion de variables estaticas en la sección GLOBAL
   public frmPago()
   {
     InitializeComponent();
   }

   private void Button1_Click(object sender, eventargs e)
   {
      //Declaracion de variables estaticas en un objeto de tipo Button
   }

   void metodo1()
   {
     //Declaracion de variables estaticas en un metodo sin valor de retorno
   }

   double metodo2()
   {
     //Declaracion de variables estaticas en un metodo con valor de retorno
   }
 }
}
```

3.10 Operadores

Son símbolos que representan un enlace entre cada uno de los argumentos que necesita una determinada operación. Normalmente, se usa para implementar expresiones.

➢ Operador de asignación

Operador	Descripción	Ejemplo

=	Permite asignar un determinado valor a una variable. Se debe considerar que dicho valor debe corresponder a un literal aceptado por el tipo de datos.	**Ejemplo 1: Asignación simple**
		int n;
		n = 10;
		
		Ejemplo 2: Asignación directa
		int n = 10;
		
		Ejemplo 3: Asignación para expresión simple
		double suma;
		suma = 10 + 12.5 + 15;
		
		Ejemplo 4: Asignación para expresión directa
		double suma = 10 + 12.5 + 15;

➢ Operadores aritméticos

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
		int num1, num2, suma;
		
		//Asignando valores
		num1=10;
		num2=20;
		
		//Expresión de suma
		suma = num1 + num2;

-	Representa la resta de dos valores en una expresión.	//Declarando variables
		int num1, num2, resta;
		
		//Asignando valores
		num1=20;
		num2=10;
		
		//Expresión de resta
		resta = num1 - num2;

*	Representa la multiplicación entre dos o más valores.	//Declarando variables
		int num1, num2, producto;
		
		//Asignando valores
		num1=5;
		num2=6;
		
		//Expresión de multiplicación
		producto = num1 * num2;

/	Representa la división entre dos valores en una expresión. Si los valores a dividir son números enteros, el resultado de la división será otro valor entero. Si uno de los elementos de la división es decimal, el resultado será decimal. Por ejemplo, representar el 5 % se puede realizar de las siguientes formas: 0.05, 5.0/100 o 5.0/100.0.	//Declarando variables
		int monto, letras, cuota;
		
		//Asignando valores
		monto = 1000.00;
		letras = 3;
		
		//Expresión de división
		cuota = monto / letras;

%	Representa el resto de la división entre dos valores en una expresión.	//Declarando variables
		int num1, num2, resto;
		
		//Asignando valores
		num1=5;
		num2=2;
		
		//Expresión de resto
		resto = num1 % num2;

3.11 Prioridad de los operadores aritméticos

Los operadores aritméticos permiten componer una expresión que permite 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 aritmética.
* /	Multiplicación y división
%	Resto de la división
+ -	Suma y resta

Se debe tener en cuenta que los paréntesis tienen la más alta prioridad en una expresión. Por lo tanto, se puede determinar el orden de la expresión agregando o quitando paréntesis según el valor esperado por el usuario.

Véase como representar una expresión aritmética en un código C#:

Expresión aritmética	Código C#
$r = \frac{3}{4}x^2y + 2xy$	double r; r = (3/4) * (x*x*y) + (2*x*y);
$v = 2\pi r - \pi r$	double v; v = (2*3.1416*r) - (3.1416*r);
$r = \frac{4}{x^2 - 2x} - \frac{2}{x - 2}$	double r; r = (4/(x*x-2*x)) – 2/(x-2);

3.12 Métodos matemáticos de .NET Framework

Existen métodos matemáticos implementados en el Framework de Visual Studio que permiten desarrollar expresiones trigonométricas, logarítmicas y otras funciones matemáticas comunes. La clase se llama Math, el cual proviene del espacio de nombres System.

A continuación, se presentan los principales métodos de la clase Math:

Método	Descripción	Ejemplo
Abs	Método que determina el valor absoluto de un valor.	int n=-10; int valorAbsoluto = Math.Abs(n);

Ceiling	Método que determina el valor más pequeño que es mayor o igual que el número decimal especificado.	double n = 2.1;
		int valorCeiling = (int) Math.Ceiling(n);
		//Resultado > 3

Floor	Método que determina el valor más grande menor o igual que el número decimal especificado	double n = 2.7;
		int valorFloor = (int) Math.Floor(n);
		//Resultado > 2

Max	Determina el mayor valor entre dos valores del mismo tipo.	int n1=10, n2=20; int mayor = Math.Max(n1, n2);
Min	Determina el menor valor entre dos valores del mismo tipo.	int n1=10, n2=20; int menor = Math.Min(n1, n2);

Pow	Determina el valor de la potencia entre un número y su base.	//Elevando un numero al cuadrado
		int n=2;
		int potencia = (int)Math.Pow(n,2);
		
		//Raiz al cubo de un numero
		int n=3;
		double raizC= Math.Pow(n , 1.0/3);
		//Respuesta > 1.44224957030741

Round	Determina el redondeo de un valor al integral más próximo.	double n = 2.675;
		double valorR = Math.Round(n,2);
		//Respuesta > 2.68

Sqrt	Determina la raíz cuadrada de un valor.	double n = 2;
		double raizC = Math.Sqrt(n);
		//Respuesta > 1.4142135623731

E	Representa la base logarítmica natural, especificada para la constante e.	double e = Math.E; //Respuesta > 2.71828182845905
PI	Representa el valor de PI.	double e = Math.PI; //Respuesta > 3.14159265358979

Véase cómo representar una expresión aritmética con código C# usando métodos de la clase Math:

Expresión aritmética	Código C#
$r = \frac{3}{4}x^2y + 2xy$	double r; r = (3/4) * (Math.pow(x,2)*y) + (2*x*y);
$v = 2\pi r - \pi r$	double v; v = (2*Math.PI*x) - (Math.PI * r);

3.13 Conversiones de tipos en C#

Las conversiones de tipos se refieren a una conversión realizada a un valor previamente declarado como un tipo de datos.

Véase cuándo es necesario realizar una conversión:

- Si se declara una variable como entera y el valor entero viene representado como cadena.
- Cuando se necesita concatenar o imprimir un valor numérico en una cadena de texto.
- Cuando dos valores numéricos necesitan 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 específico.

Para todos estos casos, existen categorías de conversiones que se verán a continuación:

3.14 Conversión implícita

Se llama así a toda conversión que se realiza con seguridad de tipos y sin pérdida de valor.

```
int n =  32767;  //Tipo original
long nLargo = n;  //Tipo destino

//Resultado nLargo = 32767
```

El valor 32767 asignado inicialmente a la variable n es asignado a la variable nLargo de tipo long, el cual pasa dicha información sin perder precisión en el dato.

Véase una lista de conversiones numéricas implícitas:

Tipo original	Tipo destino
Sbyte	short , int, long, float, double o decimal
Byte	short , ushort, int, uint, long, ulong, float, double o decimal
short	int , long, float, double o decimal
ushort	int , uint, long, ulong, float, double o decimal
Int	long , float, double o decimal
Uint	long , ulong, float, double o decimal
Long	float , double o decimal
char	ushort , int, uint, long, ulong, float, double o decimal
float	Double
ulong	float , double o decimal

El tipo original representa el tipo de dato inicial y el tipo destino representa el tipo al cual se desea convertir. Se observa que existen conversiones implícitas y no necesitan de una función para su conversión.

3.15 Conversión explícita

Se llama así a toda conversión que se realiza con seguridad de tipos y con pérdida de valor. Cuando se necesita el valor perdido, es más conveniente usar una función de conversión.

```
double valor =  16.78;  //Tipo original
int valorEntero = (int) valor;  //Tipo destino

//Resultado valorEntero > 16
```

El valor 16.78 asignado inicialmente a la variable valor es asignado a la variable valorEntero de tipo int. Por ser de tipo inferior, el valor pasa perdiendo la precisión de los valores decimales en el dato.

Véase una lista de conversiones numéricas explícitas:

Tipo original	Tipo destino
sbyte	byte , ushort, uint, ulong o char
byte	Sbyte o char
short	sbyte , byte, ushort, uint, ulong o char
ushort	sbyte , byte, short o char
int	sbyte , byte, short, ushort, uint, ulong,o char
uint	sbyte , byte, short, ushort, int o char
Long	sbyte , byte, short, ushort, int, uint, ulong o char
ulong	sbyte , byte, short, ushort, int, uint, long o char
char	sbyte , byte o short
float	sbyte , byte, short, ushort, int, uint, long, ulong, char o decimal
double	sbyte , byte, short, ushort, int, uint, long, ulong, char, float o decimal
decimal	sbyte , byte, short, ushort, int, uint, long, ulong, char, float o double

3.16 Conversión definida por el usuario

Conversión en la que el programador decide qué tipo de datos desea convertir. Se debe tener en cuenta que el programador conoce la lista de conversiones implícitas y explícitas, y que, además, reconoce el tipo original y el destino del dato.

3.17 Conversión mediante el método parse

Véase la tabla que muestra las principales conversiones:

Método Parse	Descripción	Ejemplo
Int.Parse()	Convierte el valor de una cadena en un valor numérico de tipo entero.	string n = "20"; int numero = int.Parse(n);
Int32.Parse()	Convierte el valor de una cadena en un valor numérico de tipo entero de 32 bits.	string n = "20"; Int32 numero = Int32.Parse(n);
Int64.Parse()	Convierte el valor de una cadena en un valor numérico de tipo entero de 64 bits.	string n = "20"; Int64 numero = Int64.Parse(n);
Double.Parse()	Convierte el valor de una cadena en un valor numérico de tipo double.	string n = "14.5"; double numero = Double.Parse(n);
DateTime.parse()	Convierte el valor de una cadena en un valor de tipo fecha.	string valor = "10/10/2015"; DateTime fecha = DateTime.Parse(valor);

3.18 Conversión mediante la clase Convert

Convert es una clase que proporciona una serie completa de métodos para las conversiones compatibles. La clase Convert realiza conversiones comprobadas y siempre produce una excepción si no se admite la conversión.

Se puede pasar el valor que se desea convertir a uno de los métodos adecuados de la clase Convert e inicializar el valor devuelto en una nueva variable.

Véase la tabla de las principales conversiones de la clase

Tipo	Método	Descripción

decimal	Convert.ToDecimal()	Convierte cualquier valor de tipo numérico a tipo decimal. Por ejemplo:
		
		int n = 1500;
		decimal valorDecimal = Convert.ToDecimal (n);

double	Convert.ToDouble()	Convierte cualquier valor de tipo numérico a tipo double. Por ejemplo:
		
		int n = 1500;
		double valorDoble = Convert.ToDouble (n);

short	Convert.ToInt16()	Convierte cualquier valor de tipo numérico a tipo short. Por ejemplo:
		
		int n = 50;
		short valorCorto = Convert.ToInt16 (n);

int	Convert.ToInt32()	Convierte cualquier valor de tipo numérico a tipo int. Por ejemplo:
		
		double n = 1500;
		int valorEntero= Convert.ToInt32(n);

long	Convert.ToInt64()	Convierte cualquier valor de tipo numérico a tipo long. Por ejemplo:
		
		int n = 1500;
		long valorLargo = Convert.ToLong(n);

3.19 Conversión implícita de tipos

En Visual C# existe un tipo de datos implícito que asigna un tipo de datos con solo recibir un determinado valor. Es decir, el compilador deduce el tipo de datos de la variable a partir de un valor de inicialización. Se recomienda su uso cuando el programador no tiene claro qué tipo de datos debe asignar a una variable. Su formato es:

var nombre_variable = valor;

Donde:

- **var**: Es el nombre de la función de conversión implícita de tipos.
- **nombre_variable**: Representa el nombre de la variable, para este tipo de asignación solo se puede dar valor a una sola variable.
- **valor**: Representa el valor asignado a la variable así mismo define el tipo de datos de la variable.

Véase un proceso de negocio en el que se puede usar la conversión implícita de tipos:

«Una empresa de confección de prendas paga a sus operarios un sueldo básico mensual de $ 450.00. El sueldo bruto es igual al sueldo básico más una comisión, que es igual al 12 % del monto total producido. Por ley, todo operario se somete a un descuento del 7.2 %. Determine las variables y sus tipos para calcular la comisión, el sueldo bruto, el descuento y el sueldo neto de un determinado operario".

```
//Asignando valores iniciales
var nombre = "Marisol Diaz";
var basico = 450;
var montoProducido = 10500;

//Realizando los calculos del problema
var comision = (12.0 / 100) * montoProducido;
var sueldoBruto = basico + comision;
var descuento = sueldoBruto * (7.2/100);
var sueldoNeto = sueldoBruto - descuento;
```

```
//Impriendo resultados solicitados
MessageBox.Show("Comision: " + comision);
MessageBox.Show("Sueldo Bruto: " + sueldoBruto);
MessageBox.Show("Descuento: " + descuento);
MessageBox.Show("Sueldo Neto: " + sueldoNeto);
```

3.20 Clase String

Representa una cadena de caracteres que 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.

Se debe tener en cuenta que la palabra string es el alias de String, por lo tanto, es equivalente hacer referencia a ambos al momento de declarar una variable.

➢ Propiedades de la clase String

Propiedad	Descripción

Length	Determina el número total de caracteres contenidos en una cadena. Véase cómo determinar el total de caracteres de una cadena:
	
	`string cadena = "Visual C#"; ;`
	`MessageBox.Show(cadena.Length.ToString());`
	
	Ahora véase cómo mostrar carácter por carácter una cadena:
	
	`string cadena = "Visual C#"; ;`
	`for (int i = 0; i < s.Length; i++)`
	`  MessageBox.Show(cadena[i].ToString());`

➢ Métodos de la clase String

Método	Descripción

Concat	Método que permite unir o concatenar dos o más cadenas de caracteres. Véase cómo unir tres palabras e imprimirlas en una sola instrucción.
	
	`string cadena1 = "Visual";`
	`string cadena2 = "Studio";`
	`string cadena3 = "2015";`
	
	`MessageBox.Show(String.Concat(cadena1, " ",`
	`cadena2, " ", cadena3));`

<table>
<tr><td>Equals</td><td>Determina si una cadena tiene el valor especificado. Por ejemplo, cómo comprobar que la categoría de un producto sea “Lacteos”.

<pre>string producto = "Leche sin lactosa";
string categoria = "Lacteos";
if (categoria.Equals("Lacteos"))
      MessageBox.Show("El producto es un Lacteo");</pre></td></tr>
<tr><td>StartsWith</td><td>Determinar los caracteres de inicio de una cadena de caracteres. Por ejemplo, si se quiere buscar a los empleados cuya letra inicial sea “M”.

<pre>string empleado = "Marisol Diaz Zambrano";
if (empleado.StartsWith("M"))
   MessageBox.Show("El empleado inicia con M");</pre></td></tr>
<tr><td>SubString</td><td>Recupera una subcadena de una cadena de caracteres. Por ejemplo, si se tiene el siguiente código de empleado “EMP0001” y solo se quiere recuperar la parte numérica.

<pre>string codigo = "EMP0001";
int n = int.Parse(codigo.Substring(3, 4));
MessageBox.Show("La parte numerica del código es: "+n);</pre></td></tr>
<tr><td>ToLower</td><td>Devuelve una copia de la cadena en minúsculas. Por ejemplo, si se quiere convertir el correo electrónico registrado por un usuario en minúsculas.

<pre>string correo = "MDIAZ@GMAIL.COM";
string correoValido = correo.ToLower(); MessageBox.Show("El
correo es: "+correoValido);</pre></td></tr>
<tr><td>ToString</td><td>Asigna el formato de salida tipo cadena a una variable. Aquí se puede especificar un determinado formato de salida.
<table>
<tr><th>Formato</th><th>Descripción</th><th>Ejemplo</th></tr>
<tr><td>C</td><td>Moneda</td><td>double sueldo = 1000;
MessageBox.Show(sueldo.ToString("C"));

//Resultado > $ 1000.00</td></tr>
<tr><td>D</td><td>Decimal</td><td>double sueldo = 10;
MessageBox.Show(sueldo.ToString("D5"));

//Resultado > 00010</td></tr>
<tr><td>F</td><td>Punto fijo</td><td>double porcentaje = 12;
MessageBox.Show(porcentaje.ToString("F2"));

//Resultado > 12.00</td></tr>
</table></td></tr>
<tr><td>ToUpper</td><td>Devuelve una copia de la cadena en mayúsculas. Por ejemplo, si se quiere convertir a mayúscula el apellido paterno de un usuario.

<pre>string paterno = "zamora";
string paternoMayuscula = paterno.ToUpper();
MessageBox.Show("Paterno: "+paternoMayuscula);</pre></td></tr>
<tr><td>Trim</td><td>Elimina los caracteres en blanco del lado derecho e izquierdo de una cadena de caracteres. Por ejemplo, si se quieren eliminar los espacios en blanco de una cadena:

<pre>string cadena = "   RESUMEN DE VENTA   ";
MessageBox.Show("Cadena: "+cadena.Trim());</pre></td></tr>
</table>

3.21 La clase MessageBox

Representa una ventana de salida; se conoce también como el 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 se llama ventana modal.

Véase cuándo usar una ventana de dialogo de mensaje:

- 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.

Formato básico para un mensaje simple

```
MessageBox.Show("Mensaje simple al usuario");
```

Formato para un mensaje con título en la ventana

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

Formato para un mensaje con botones

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

Formato para un mensaje con botones e iconos

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

Formato para obtener una respuesta del usuario

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

Donde:

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

Véase la lista de botones permitidos:

Botón	Descripción	Imagen
OK	Representa al botón Aceptar, que es considerado como 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, se tiene: 1. DialogResult.Yes 2. DialogResult.No	
YesNoCancel	Representa a los botones Si, No y Cancelar del cuadro de dialogo de mensaje.	

- **MessageBoxIcon**: Conjunto de iconos permitidos en el cuadro de dialogo de mensaje.

Véase la lista de iconos permitidos:

Ícono	Descripción	Imagen
Error	Se muestra un símbolo que consiste en una x blanca en un círculo con fondo rojo dentro del cuadro de dialogo de mensaje.	
Exclamation	Se muestra un símbolo que consiste en un signo de exclamación en un triángulo con fondo amarillo dentro del cuadro de dialogo 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.	

3.22 Casos desarrollados

Caso desarrollado 1 : pago de empleados

Una empresa manufacturera de implementos deportivos de Perú necesita una aplicación que le permita calcular el pago mensual que les corresponde a sus empleados. El pago se inicia con el sueldo básico, el cual se calcula en base al número de horas trabajadas y una tarifa horaria que varía según el empleado. Adicionalmente, al sueldo básico se le aplica una bonificación especial del 20 % con lo que se obtiene el sueldo bruto. A este último, finalmente se debe calcular el sueldo neto el cual aplica un descuento de ley correspondiente al 12 % del sueldo bruto.

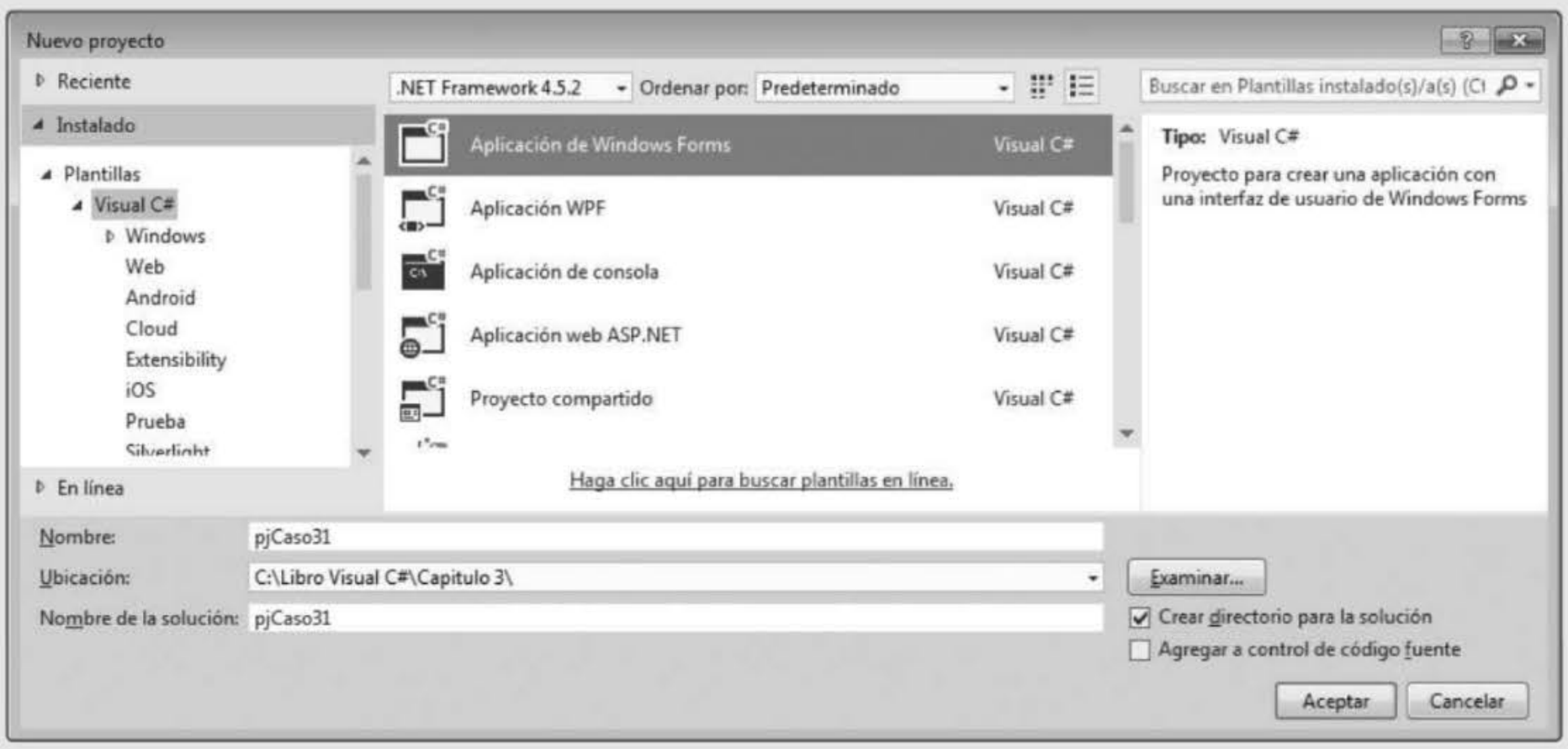
Se tiene en cuenta los siguientes aspectos:

- Diseñe la interfaz gráfica de usuario adecuada para el caso.
- Muestre los datos del empleado, horas trabajadas y costo por hora.
- Muestre el sueldo básico, el sueldo bruto, monto de descuento y el sueldo neto de un empleado.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

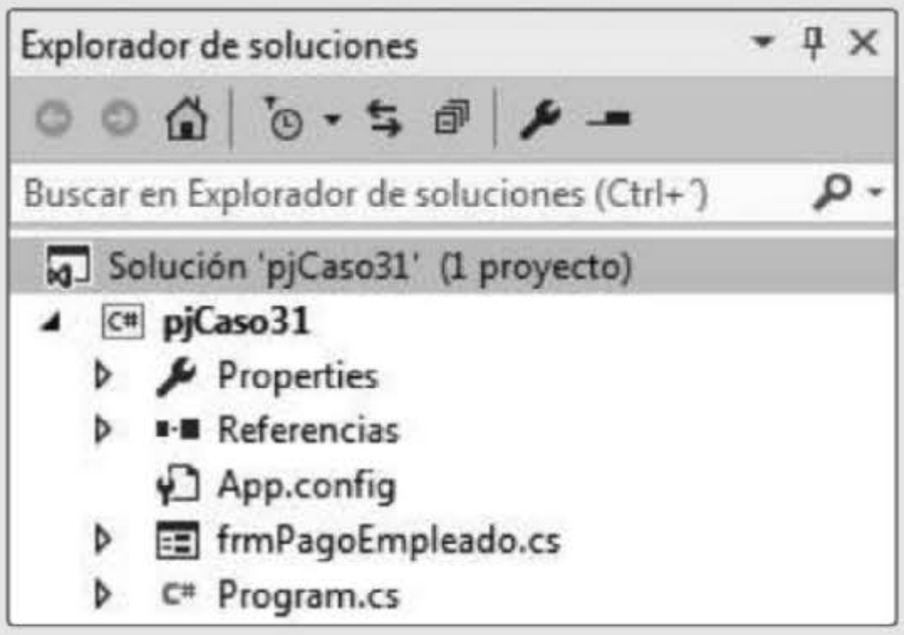
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso31**, tal como se muestra en la siguiente imagen:



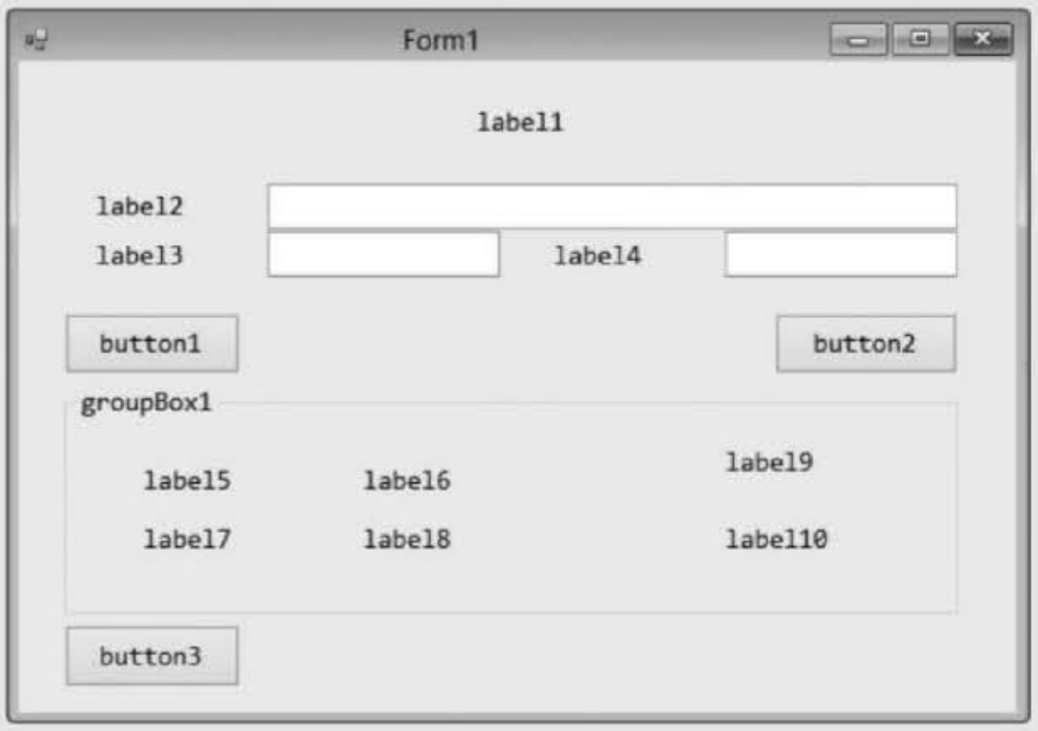
   - Finalmente, presione el botón **Aceptar**.

2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la ventana de propiedades, seleccione **Nombre de archivo** y asigne el nombre **frmPagoEmpleado.cs**.
- La ventana **Explorador de soluciones** debe mostrarse de la siguiente manera:



3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



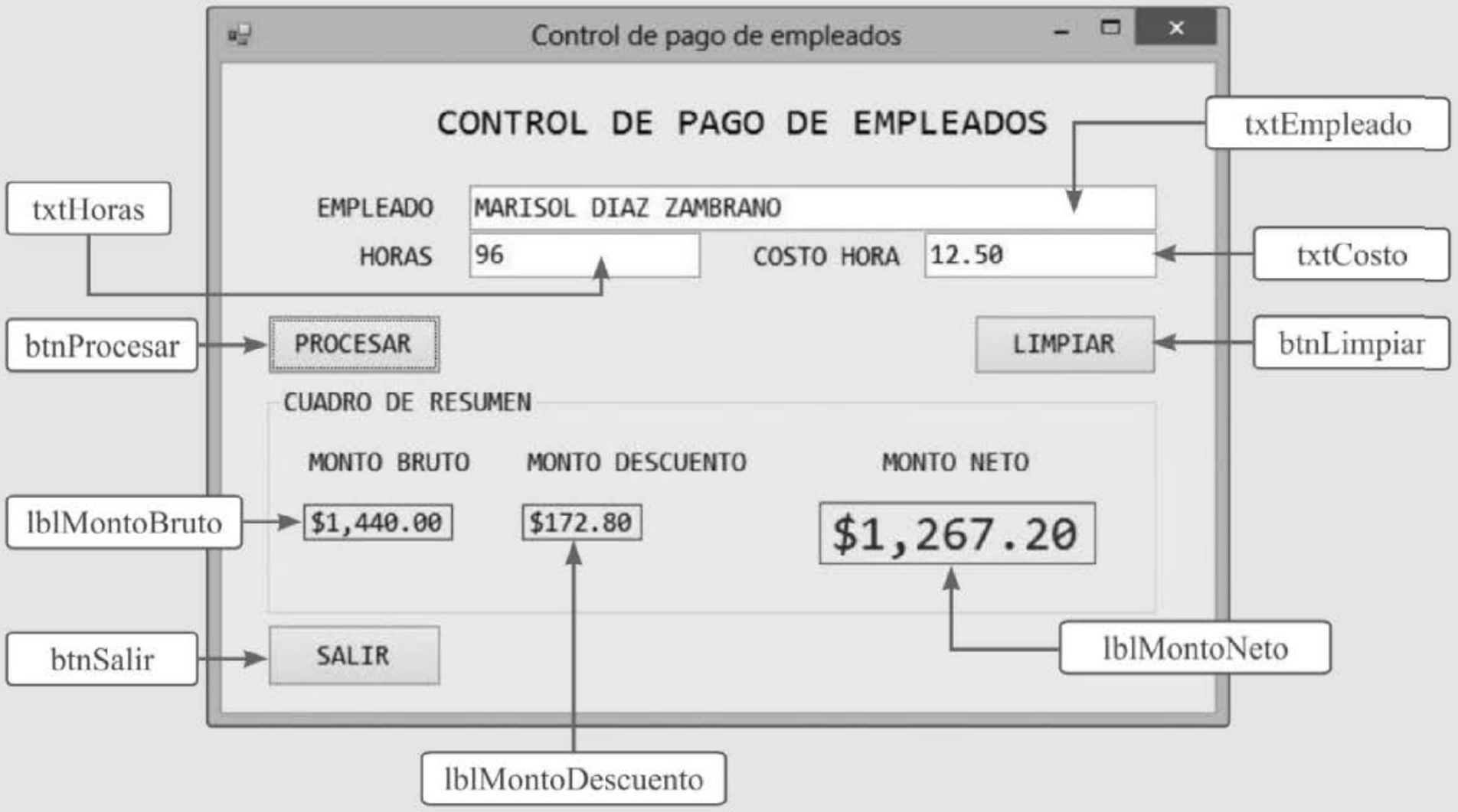
Estadísticas de los controles usados en el frmPagoEmpleado:

Clase	N.º	Controles
Form	01	form1
Label	10	label1, label2, label3, label4, label5, label6, label7, label8, label9, label10.
TextBox	03	textbox1, textbox2, textbox3
Button	03	button1, button2, button3
GroupBox	01	groupbox1
TOTAL	18	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmPagoEmpleado Control de pago de empleados
Label1	Text	CONTROL DE PAGO DE EMPLEADOS
Label2	Text	EMPLEADO
Label3	Text	HORAS
Label4	Text	COSTO HORA
Label5	Text	MONTO BRUTO
Label6	Text	MONTO DESCUENTO
Label7	(Name) Text BorderStyle	lblMontoBruto Dejar vacio FixedSingle
Label8	(Name) Text BorderStyle	LblMontoBruto Dejar vacio FixedSingle
Label9	Text	MONTO NETO
Label10	(Name) Text BorderStyle	lblMontoBruto Dejar vacio FixedSingle
Textbox1	(Name)	txtEmpleado
Textbox2	(Name)	txtHoras
Textbox3	(Name)	txtCosto
Button1	(Name) Text	bntProcesar PROCESAR
Button2	(Name) Text	btnLimpiar LIMPIAR
Button3	(Name) Text	btnSalir SALIR
GroupBox1	Text	CUADRO DE RESUMEN

Finalmente, la interfaz queda de la siguiente forma:



5. Agregar código fuente al botón Procesar.
   - Haga doble clic sobre el botón **Procesar**.

```
private void btnProcesar_Click(object sender, EventArgs e)
{
    //Recuperando la informacion registrada
    string empleado = txtEmpleado.Text;
    int horas = int.Parse(txtHoras.Text);
    double costo = double.Parse(txtCosto.Text);

    //Realizando los calculos
    double sueldoBasico = horas * costo;
    double bonificacion = 20.0/100 * sueldoBasico;
    double sueldoBruto = sueldoBasico + bonificacion;
    double descuento = sueldoBruto * 12.0/100;
    double sueldoNeto = sueldoBruto - descuento;

    //Imprimiendo resultados
    lblMontoBruto.Text = sueldoBruto.ToString("C");
    lblMontoDescuento.Text = descuento.ToString("C");
    lblMontoNeto.Text = sueldoNeto.ToString("C");
}
```

6. Agregar código fuente al botón Limpiar:

   - Haga doble clic sobre el botón **Limpiar**.

```
private void btnLimpiar_Click(object sender, EventArgs e)
{
    txtEmpleado.Clear();
    txtHoras.Clear();
    txtCosto.Clear();
    lblMontoBruto.Text = "";
    lblMontoDescuento.Text = "";
    lblMontoNeto.Text = "";
    txtEmpleado.Focus();
}
```

7. Agregar código fuente al botón Salir:

   - Haga doble clic sobre el botón **Salir**.

```
private void btnSalir_Click(object sender, EventArgs e)
{
    DialogResult r = MessageBox.Show("Esta seguro de seguro?",
                                     "Control de pago",
                                     MessageBoxButtons.YesNo,
                                     MessageBoxIcon.Information);
    if (r == DialogResult.Yes)
         this.Close();
}
```

8. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 2 : control de medidas

La empresa ManRud dedicada al rubro de sastrería para varones necesita ampliar su línea de negocio a otros países, los cuales solicitan propuestas en diversos sistemas de medida. Por esto, se debe implementar una aplicación que permita convertir una cantidad dada en metros a sus equivalentes en centímetros, pulgadas, pies y yardas. Para esto, se debe tomar en cuenta la siguiente tabla de conversión:

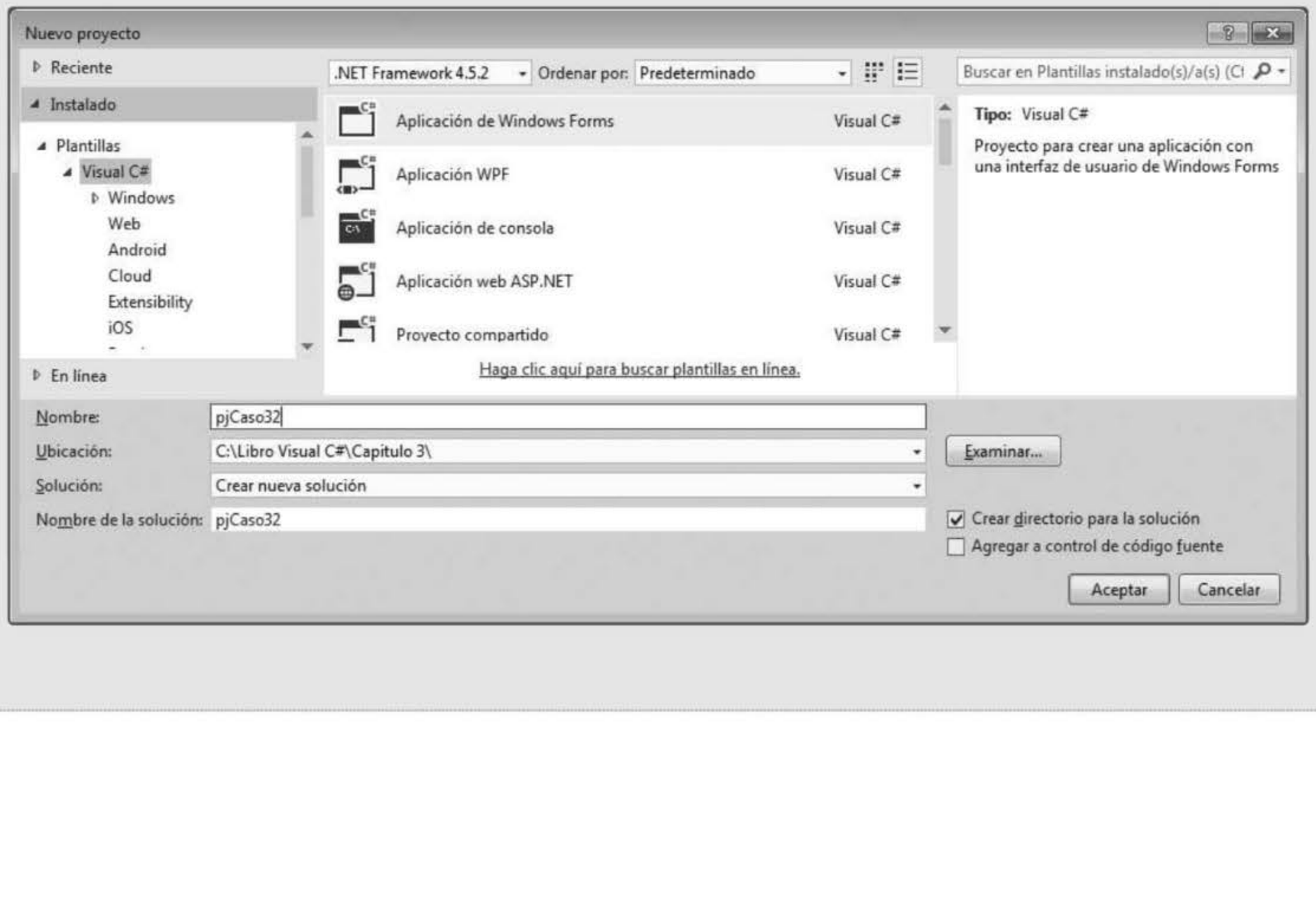
1 metro = 100 centímetros
1 pie = 12 pulgadas
1 yarda = 3 pies
1 pulgada = 2.54 centímetros

Se tiene en cuenta los siguientes aspectos:

- Diseñe la interfaz gráfica de usuario adecuada para el caso.
- Muestre los datos resultantes en un control cuadro de lista (ListBox).

Solución:

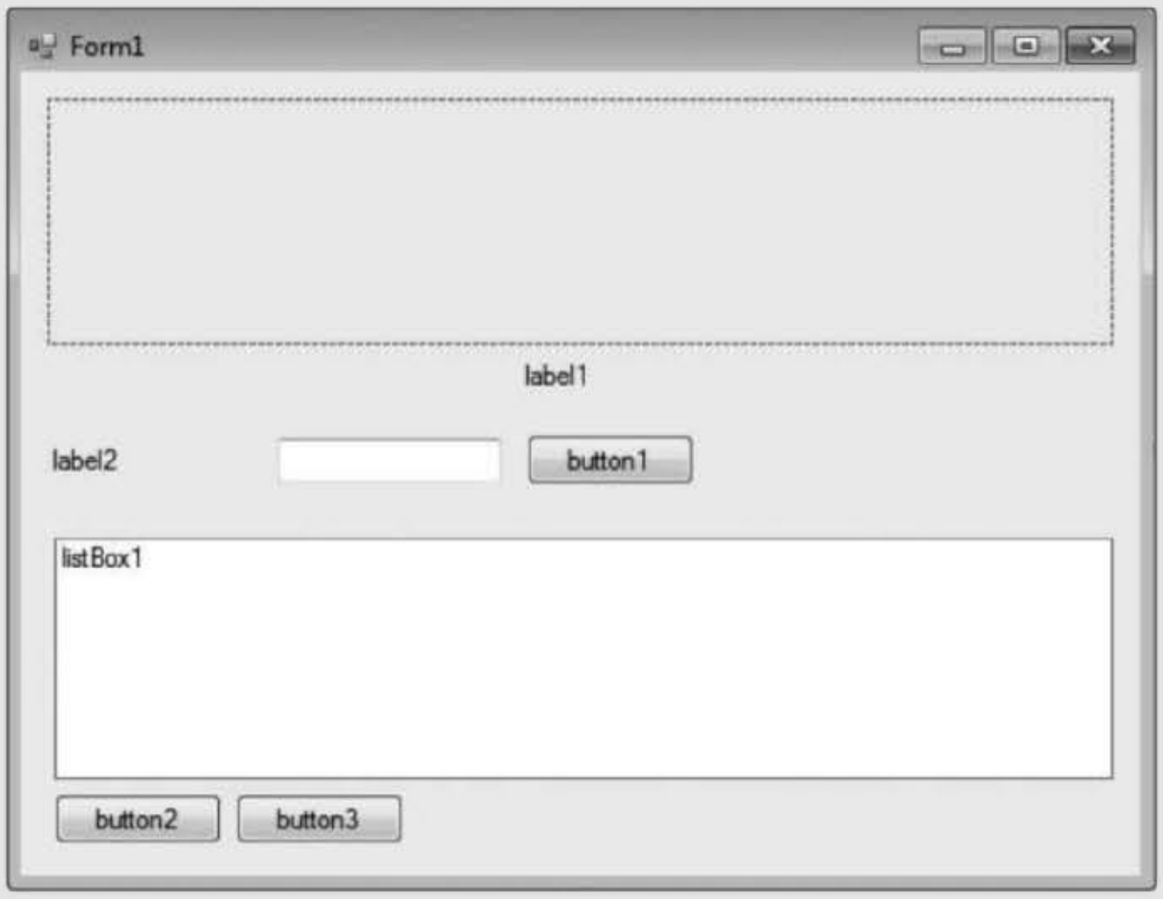
1. Crear un nuevo proyecto en Visual Studio 2015:
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso32**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la propiedad **Nombre de archivo**, asigne el nombre **frmMedidas**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmMedidas:

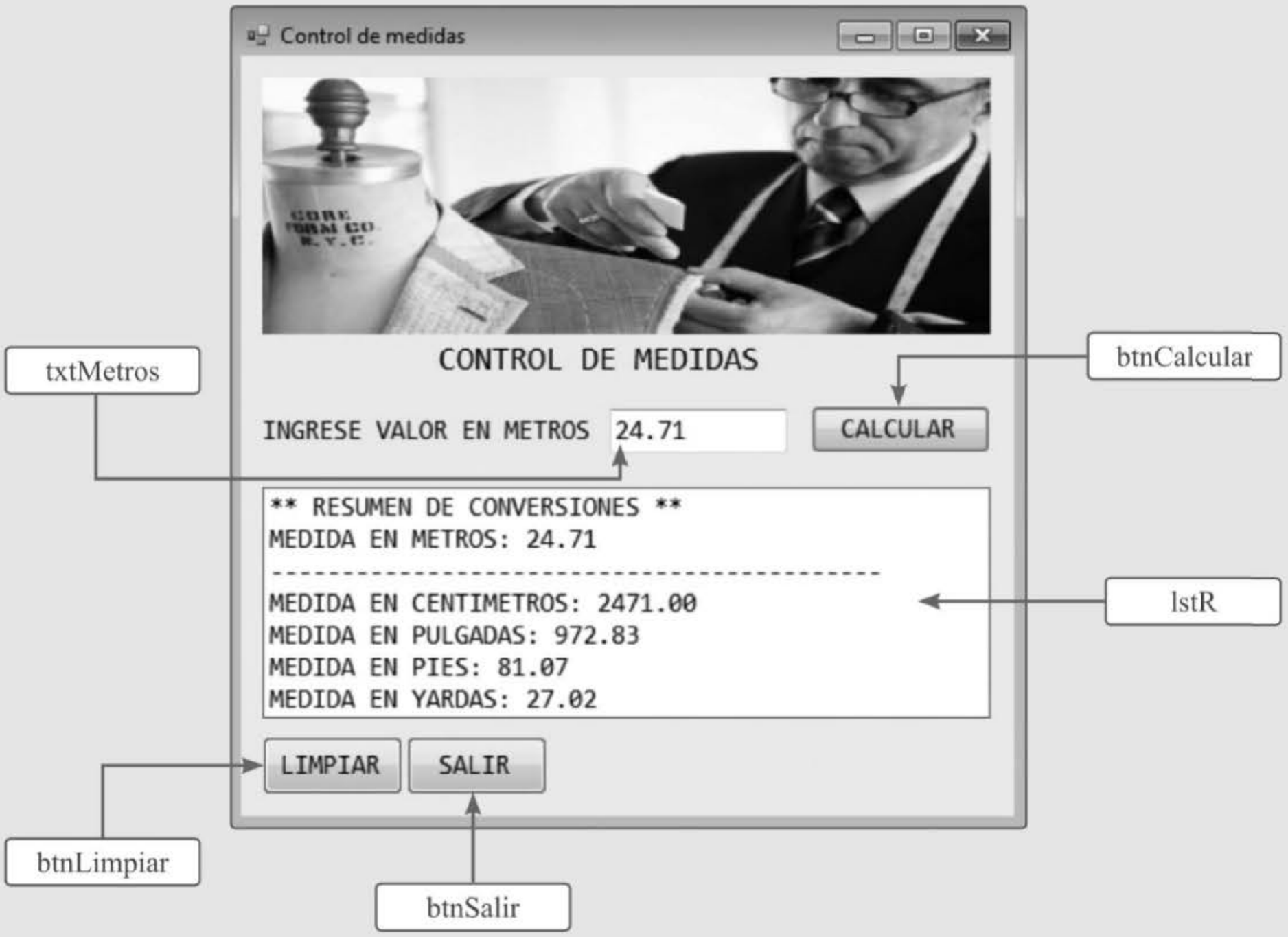
Clase	N.º	Controles
Form	01	Form1
PictureBox	01	PictureBox1
Label	02	Label1, Label2
TextBox	01	TextBox1
Button	03	Button1, Button2, Button3
TOTAL	08	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmMedidas Control de medidas
PictureBox	BackgroundImage BackgroundImageLayout	Seleccione una imagen Stretch
Label1	Text	CONTROL DE MEDIDAS
Label2	Text	INGRESE VALOR EN METROS
Textbox1	(Name)	txtMetros

Button1	(Name) Text	bntCalcular CALCULAR
Button2	(Name) Text	btnLimpiar LIMPIAR
Button3	(Name) Text	btnSalir SALIR
ListBox1	(Name)	lstR

Finalmente, la interfaz queda de la siguiente forma:



5. Agregar código fuente al botón Calcular:
   - Haga doble clic sobre el botón **Calcular**.

```
private void btnCalcular_Click(object sender, EventArgs e)
{
    //Capturando el valor en metros
    double metros = double.Parse(txtMetros.Text);

    //Realizando conversiones
    double centimetros = metros * 100;
    double pulgadas = centimetros / 2.54;
    double pies = pulgadas / 12;
    double yardas = pies / 3;
```

```
   //Mostrando los resultados de la conversión
   lstR.Items.Add("** RESUMEN DE CONVERSIONES **");
   lstR.Items.Add("MEDIDA EN METROS: " + metros.ToString("0.00"));
   lstR.Items.Add("----------------------------------------");
   lstR.Items.Add("MEDIDA EN CENTIMETROS : "+ centimetros.ToString("0.00"));
   lstR.Items.Add("MEDIDA EN PULGADAS: " + pulgadas.ToString("0.00"));
   lstR.Items.Add("MEDIDA EN PIES: " + pies.ToString("0.00"));
   lstR.Items.Add("MEDIDA EN YARDAS: " + yardas.ToString("0.00"));
}
```

6. Agregar código fuente al botón Limpiar:

- Presione doble clic sobre el botón **Limpiar.**

```
private void btnLimpiar_Click(object sender, EventArgs e)
{
   txtMetros.Clear();
   lstR.Items.Clear();
   txtMetros.Focus();
}
```

7. Agregar código fuente al botón Salir:

- Haga doble clic sobre el botón **Salir**.

```
private void btnSalir_Click(object sender, EventArgs e)
{
   DialogResult r = MessageBox.Show("Esta seguro de salir?","Medidas",
                                    MessageBoxButtons.YesNo,
                                       MessageBoxIcon.Information);
   if (r == DialogResult.Yes) this.Close();
}
```

8. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 3 : control de área y volumen

Kimberly-Clark del Perú, empresa líder en productos de consumo para el cuidado e higiene personal y familiar, necesita una aplicación que permita calcular el área total y el volumen de los nuevos rollos de papel higiénico que lanzará al mercado. Implemente una aplicación que calcule el área total y el volumen de un rollo de papel a partir de una altura y un diámetro. Para esto, se deben tomar en cuenta las siguientes fórmulas:



$$\text{Área} = 2\pi r(r+h)$$
$$\text{Volumen} = \pi r^2 h$$

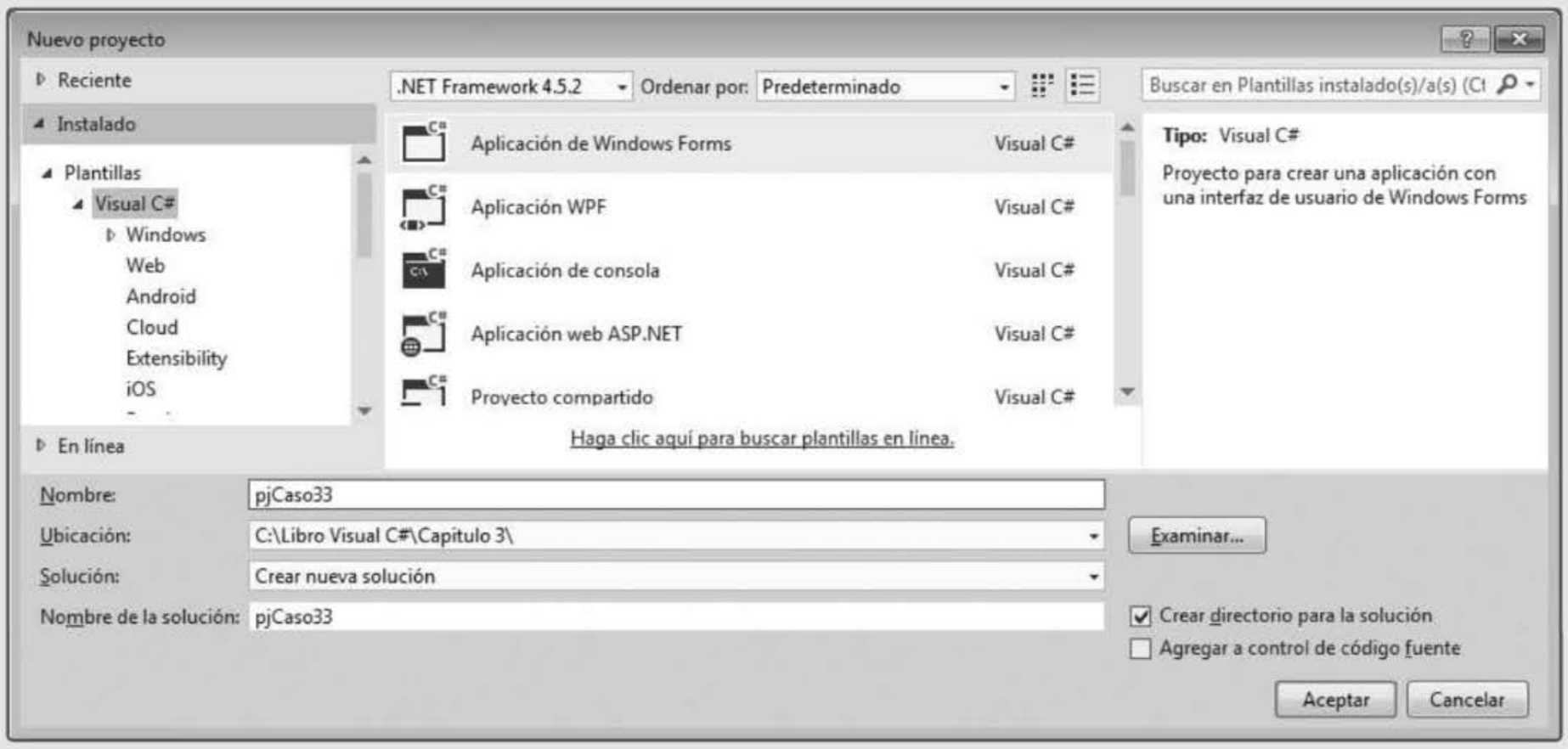
Se tiene en cuenta los siguientes aspectos:

- Diseñar la interfaz gráfica de usuario adecuada para el caso.
- Mostrar los resultados de la aplicación en un cuadro de texto.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

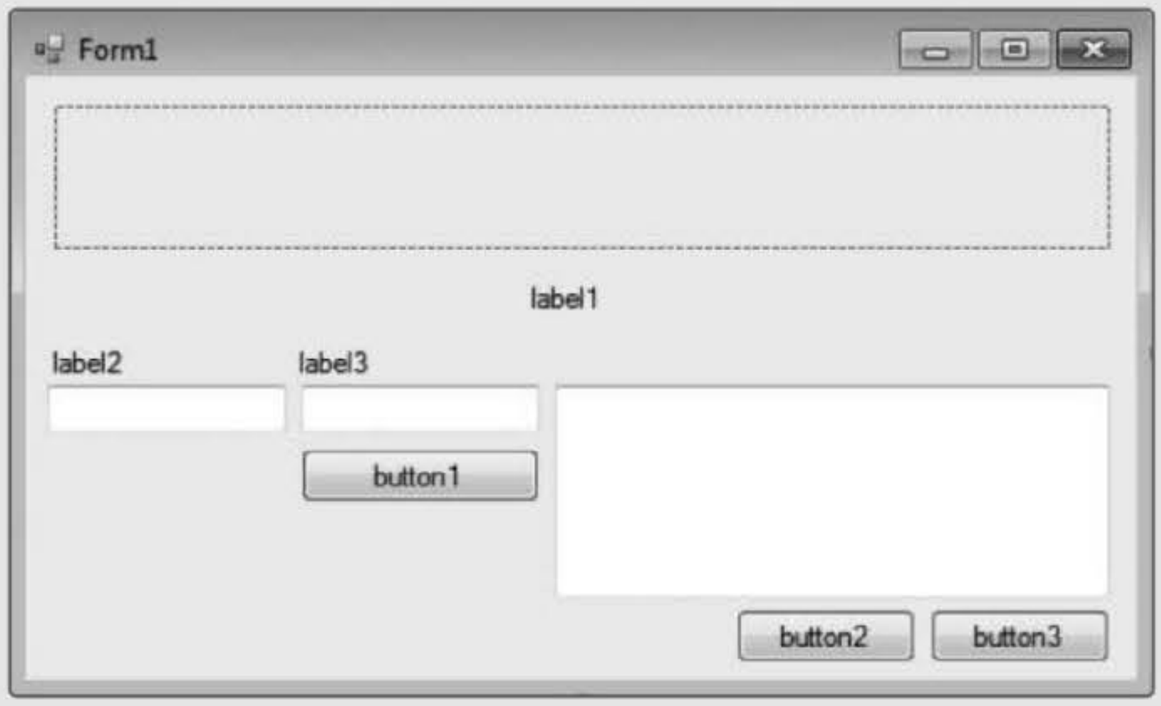
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso33**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la propiedad **Nombre de archivo**, asigne el nombre **frmRollo**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmRollo:

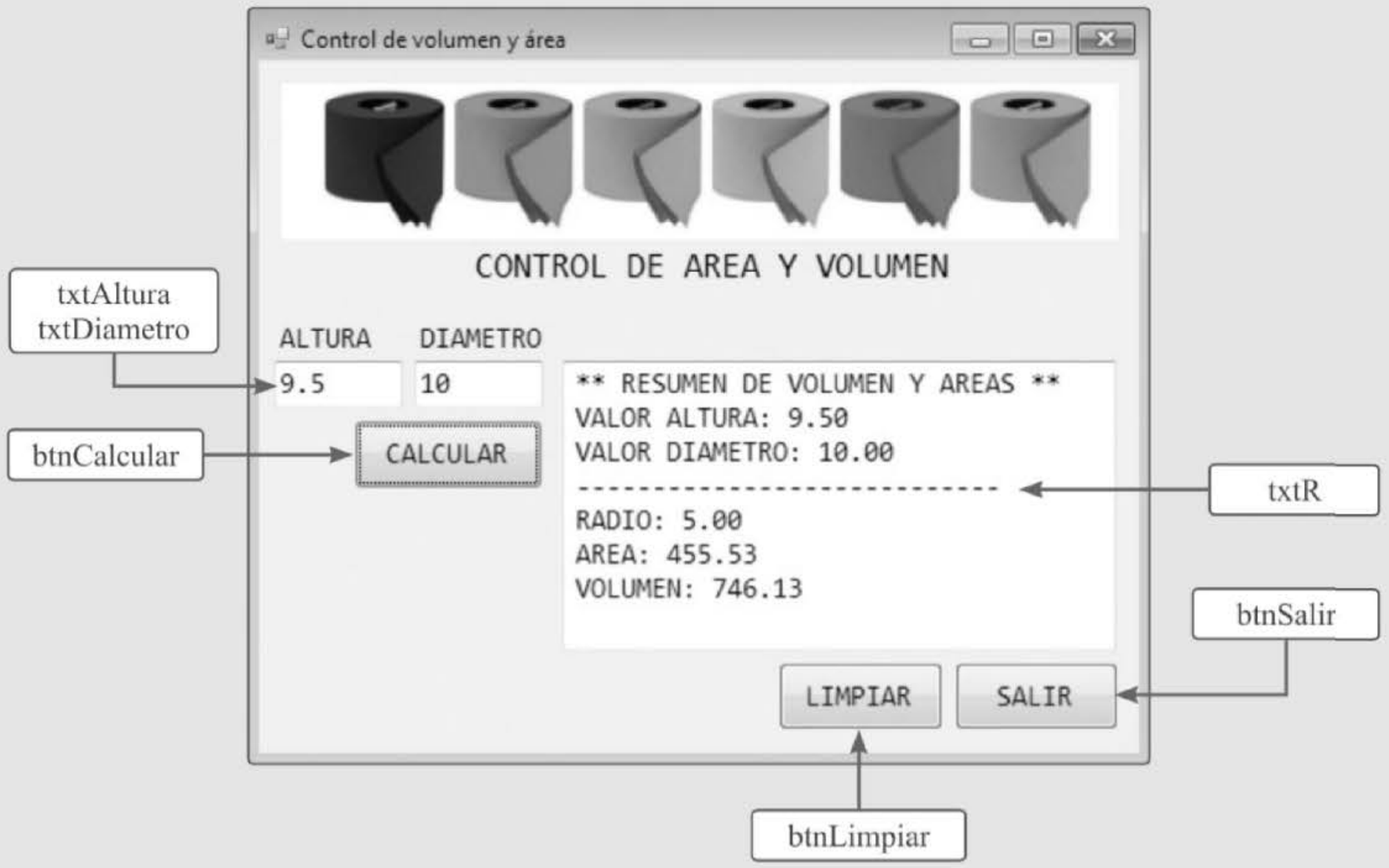
Clase	N.º	Controles
Form	01	Form1
PictureBox	01	PictureBox1
Label	03	Label1, Label2, Label3
TextBox	03	TextBox1, TextBox2, TextBox3
Button	03	Button1, Button2, Button3
TOTAL	11	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmRollo Control de volumen y área
PictureBox	BackgroundImage BackgroundImageLayout	Seleccione una imagen Stretch
Label1	Text	CONTROL DE AREA Y VOLUMEN
Label2	Text	ALTURA
Label3	Text	DIAMETRO
Textbox1	Text	txtAltura
Textbox2	Text	txtDiametro

Textbox3	(Name) MultiLine	txtR True
Button1	(Name) Text	bntCalcular CALCULAR
Button2	(Name) Text	btnLimpiar LIMPIAR
Button3	(Name) Text	btnSalir SALIR

Finalmente, la interfaz queda de la siguiente forma:



5. Agregar código fuente al botón Calcular:
   - Haga doble clic sobre el botón **Calcular**.

```
private void btnCalcular_Click(object sender, EventArgs e)
{
    //Capturando los valores
    double altura = double.Parse(txtAltura.Text);
    double diametro = double.Parse(txtDiametro.Text);

    //Realizando los calculos
    double radio = diametro / 2;
    double area = (2 * Math.PI * radio) * (radio + altura);
    double volumen = Math.PI * Math.Pow(radio, 2) * altura;
```

```
//Mostrando los resultados en el cuadro de texto
txtR.Text = "** RESUMEN DE VOLUMEN Y AREAS **";
txtR.Text += "\r\nVALOR ALTURA: " + altura.ToString("0.00");
txtR.Text += "\r\nVALOR DIAMETRO: " + diametro.ToString("0.00");
txtR.Text += "\r\n----------------------------";
txtR.Text += "\r\nRADIO: " + radio.ToString("0.00");
txtR.Text += "\r\nAREA: " + area.ToString("0.00");
txtR.Text += "\r\nVOLUMEN: " + volumen.ToString("0.00");
```

6. Agregar código fuente al botón Limpiar:

   - Haga doble clic sobre el botón **Limpiar**.

```
private void btnLimpiar_Click(object sender, EventArgs e)
{
    txtAltura.Clear();
    txtDiametro.Clear();
    txtR.Clear();
    txtAltura.Focus();
}
```

7. Agregar código fuente al botón Salir:

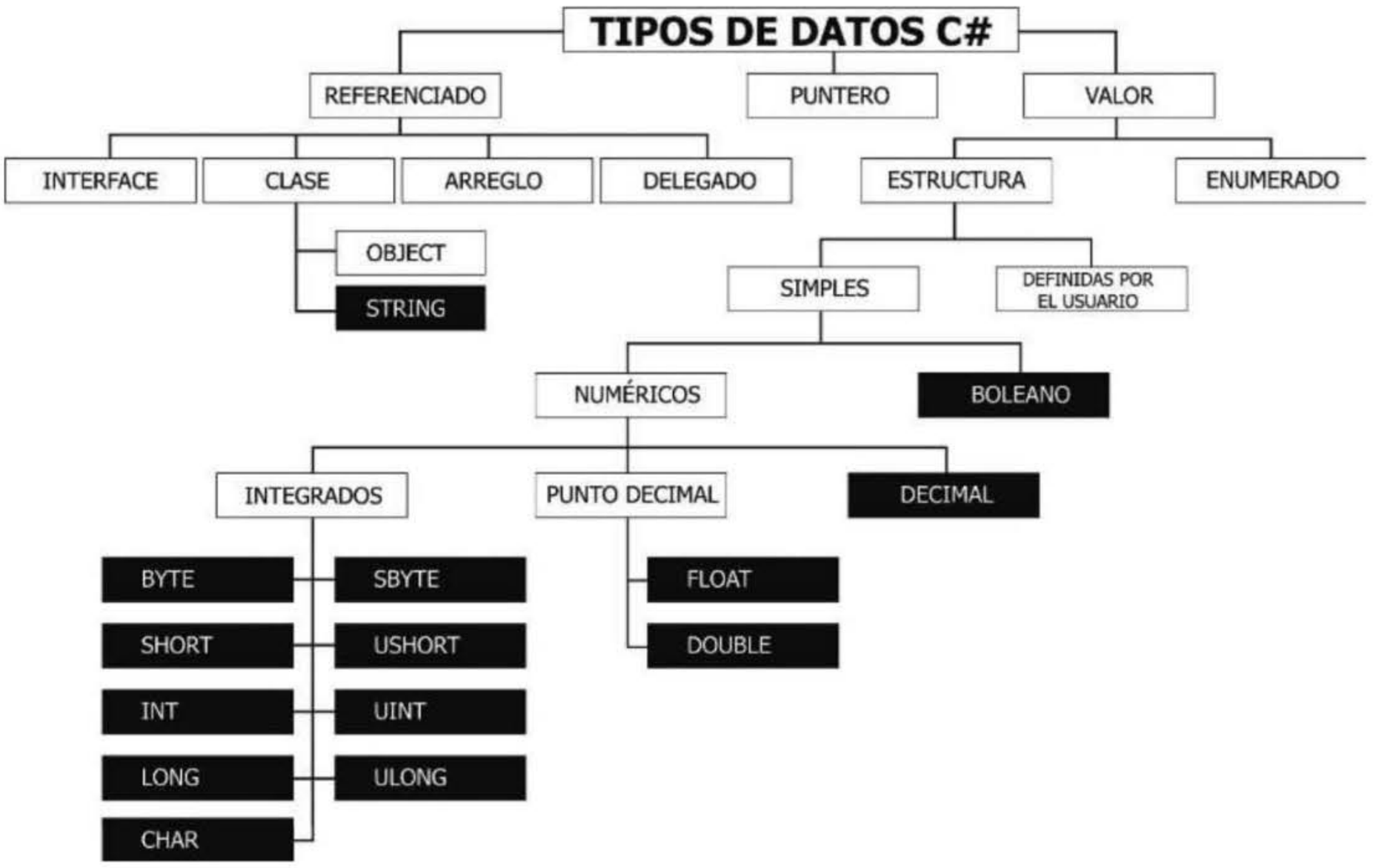
   - Haga doble clic sobre el botón **Salir**.

```
private void btnSalir_Click(object sender, EventArgs e)
{
   DialogResult r = MessageBox.Show("Esta seguro de salir?", "Medidas",
                                    MessageBoxButtons.YesNo,
                                    MessageBoxIcon.Information);
   if (r == DialogResult.Yes) this.Close();
}
```

8. Finalmente, presione **F5** para probar la aplicación.

Para recordar

- Para crear un nuevo proyecto, realice los siguientes pasos:
  - Seleccione **Archivo > Nuevo > Proyecto**.
  - Seleccione la plantilla **Visual C#**.
  - Asigne una ubicación y un nombre al proyecto.
- Cuando un evento es considerado una ocurrencia del control, se debe tener en cuenta que todos los controles tienen eventos y que pueden ser manipulados por el programador; es decir, se puede programar dentro de un evento que se acciona justamente cuando ocurre dicho evento.
- En una aplicación, el editor de código presenta un entorno para el desarrollo de código fuente y para acceder a este, debe presionar doble en el control donde necesite un desarrollo.
- Los comentarios en el código sirven para recordar alguna información referente al código o simplemente para anular el código en la aplicación, ya que los comentarios son obviados por el compilador.
- Los tipos de datos permiten definir la capacidad de una variable en el desarrollo de una aplicación. Se tiene el siguiente esquema:



- Las palabras reservadas en Visual C# no pueden ser usadas como identificadores para ningún elemento en el desarrollo de una aplicación.
- Una variable es un espacio de memoria que permite almacenar algún tipo de información, dicho valor tendrá un rango de capacidad definida por el tipo de datos.
- El operador (=) representa la asignación de una valor a una variable, mientras que la doble igualdad representa la comparación entre dos valores o variables.
- El operador (+) tiene doble funcionalidad: si los valores son números, el operador realiza la suma de dichos valores y si ambos son cadenas, los concatena.
- Las conversiones de tipo representa un punto importante dentro de la codificación en una aplicación, se distinguen las implícitas, explícitas, por parseo y por medio de la clase convert.
- La clase string representa una cadena de caracteres, puede contener todo tipo de valores como textos, números, símbolos, etc.
- La clase MessageBox permite mostrar un cuadro de mensaje al usuario.

Preguntas frecuentes

➢ **¿Cómo cambiar el tamaño y tipo de letra de la fuente que se presenta en el editor de código?**

- Seleccione **Herramientas > Opciones**.
- Seleccione **Entorno > Fuentes y colores**.
- Realice la modificación de la fuente y el tamaño del texto.
- Haga clic en **Aceptar**.

➢ **Cómo asignar comentarios a un bloque seleccionado de código?**

- Desde la ventana de código, seleccione una porción de código.
- Presione **CTRL + E**, luego presione la letra **C**.

  Y para quitar el comentario de un bloque de código:
- Seleccione el código y presione **CTRL + E**, luego la letra **U**.

➢ **¿Cómo saber cuáles son las palabras reservadas del lenguaje C#?**

- Visite la siguiente URL: https://msdn.microsoft.com/es-pe/library/x53a06bb.aspx

➢ **¿Cómo especificar correctamente el formato de una cadena?**

- Visite la siguiente URL: https://msdn.microsoft.com/es-es/library/dwhawy9k(v=-vs.110).aspx

Capítulo 4

Estructuras de decisión

Capacidad terminal:

Implementar aplicaciones que usen sentencias condicionales, asimismo el resultado de todas las aplicaciones las mostrara en una lista de datos de manera profesional.

Contenido:

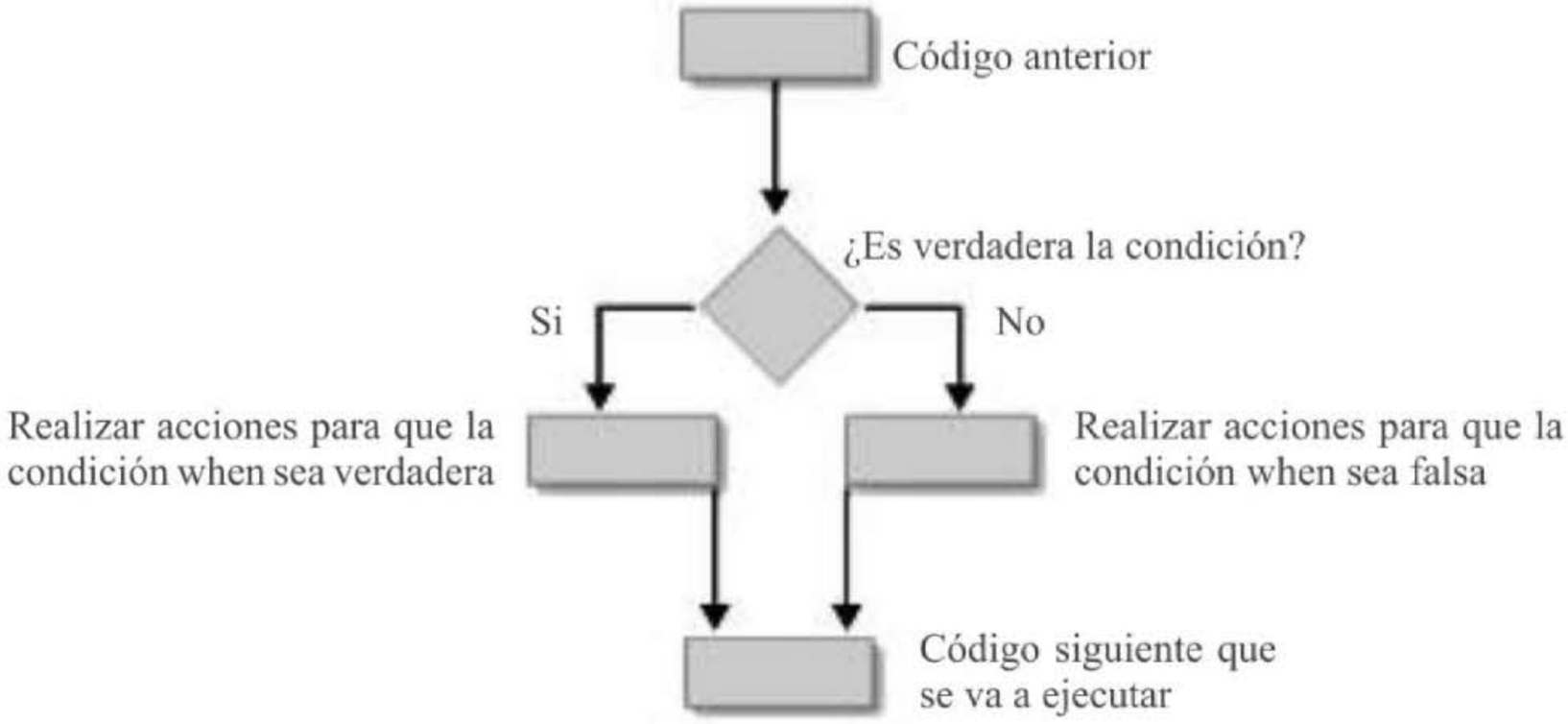
- Introducción
- Operadores de relación
- Operadores lógicos
- Instrucción if simple
- Instrucción if doble
- Instrucción if doblemente enlazada
- Instrucción condicional múltiple
- Casos desarrollados
  - o Caso desarrollado 1: if simple - pago de empleados
  - o Caso desarrollado 2: if simple - control de registro de documentos
  - o Caso desarrollado 3: if doble - promedio de notas
  - o Caso desarrollado 4: if doble - venta de productos
  - o Caso desarrollado 5: if doblemente enlazado - control de registro de multas de transito
  - o Caso desarrollado 6: if doblemente enlazado - registro de postulantes
  - o Caso desarrollado 7: switch - registro de registro de llamadas
  - o Caso desarrollado 8: switch - control de registro de estacionamiento

- Para recordar
- Preguntas frecuentes

Introducción

Cuando se implementa una aplicación en C#, se debe considerar que para finalizar una expresión o línea de código, debe asignarse punto y coma, el cual forma una sentencia. El conjunto de sentencias uno a continuación del otro es considerada como estructuras secuenciales.

Asimismo, una estructura secuencial se caracteriza por realizar operaciones en forma secuencial de arriba hacia abajo. Pero no todos los casos pueden resolverse empleando dicha estructura, ya que en algún momento se debe tomar una decisión, allí se aplican las sentencias selectivas o también llamadas sentencias condicionales. En la vida diaria de un programador se pueden encontrar varias sentencias condicionales como ¿En qué lenguaje de programación desarrollar?, ¿qué sistema de gestor de base de datos usar en la aplicación?, ¿qué tipo de aplicación desarrollar?, etc. La siguiente imagen demuestra el punto de quiebre de la sentencia secuencial, ya que en determinado momento aparecerán las sentencias condicionales.



Las estructuras condicionales se pueden considerar como simples, dobles, enlazadas y doblemente enlazadas

4.1 Operadores de relación

El uso de una estructura condicional tiene como base fundamental la implementación de la condición; ya que es por donde la estructura fluye durante el proceso de la aplicación.

Véase la lista operadores válidos para C#:

Operador	Definición	Ejemplo

>	Mayor que	Determinar si el sueldo de un empleado es superior a $ 4500.00
		
		double sueldo = 4500;
		if (sueldo > 4500)
		
		Determinar si un candidato a una beca es mayor de edad, asumiendo que una persona es considerada mayor de edad cuando su edad es superior o igual a 18.
		
		int edad = 23;
		if (edad >17)

> =	Mayor o Igual que	Determinar si el sueldo de un empleado es superior a $ 4500.00
		
		double sueldo = 5000;
		if (sueldo > = 4500)
		
		Determinar si un candidato a una beca es mayor de edad, asumiendo que una persona es considerada mayor de edad cuando su edad es superior o igual a 18.
		
		int edad = 23;
		if (edad > = 18)

<	Menor que	Determinar si el precio de un producto es inferior a $ 20.00
		
		double precio = 50;
		if (precio < 20)
		
		Determinar si el promedio de nota de un alumno se considera como desaprobado; es decir, dicho promedio debe ser menor a 12.5.
		
		double promedio = 17.10;
		if (promedio < 12.50)

< =	Menor o Igual que	Determinar si el precio de un producto es inferior a $ 20.00
		
		double precio = 50;
		if (precio < = 19.99)
		
		Determinar si el promedio de nota de un alumno se considera como desaprobado; es decir, dicho promedio debe ser menor a 12.5.
		
		double promedio = 17.10;
		if (promedio < = 12.49)

= =	Igual	Comparar si la categoría de un producto en una tienda es considerada como bebidas. double categoria = "Bebidas"; if (categoría == "Bebidas") Determinar si la cantidad de productos comprados en una tienda resulta 20. int cantidad = 20; if (cantidad == 20)
! =	Diferente o no es igual	Determinar si la categoría de un empleado es diferente a la descripción "Operario". string categoría = "Operario"; if (categoría ! = "Operario")
?	Condicional	Permite devolver uno de dos valores especificados en una sola sentencia condicional. Determinar si la nota obtenida por un alumno en una evaluación es aprobatoria o no, se debe tener en cuenta que la nota mínima aprobatoria es 12.5. double nota = 20; string condicion = nota < 12.5 ? "Desaprobado": "Aprobado";

4.2 Operadores lógicos

Los operadores lógicos cumplen un papel importante en la implementación de la condición en una estructura selectiva, pues permiten unir dos o más condiciones; de esta forma la condición cuenta con más criterios de evaluación. Se debe tener en cuenta que el resultado adecuado depende del orden de prioridad de los operadores usados en la condición.

Operador	Definición	Ejemplo
&&	Representa la Y lógica, que evalúa que ambas condiciones sean de tipo true para realizar un determinada acción.	Determinar si los empleados de una empresa ganan entre $ 1000.00 y $ 1500.00 double sueldo = 1000; if (sueldo>=1000 && sueldo<=1500)
||	Representa la O lógica, que ejecuta una determinada acción solo si una de las condiciones es de tipo true.	Determinar si los empleados de una empresa tienen la categoría "Jefe" o "Gerente". string categoria = "Jefe"; if (categoria =="Jefe" || categoria == "Gerente")
!	Representa la negación del resultado de una condición.	Determinar si los empleados de una empresa no son de tipo Jefe ni Gerente. string categoria = "Jefe"; if (!(categoria =="Jefe" || categoria == "Gerente"))

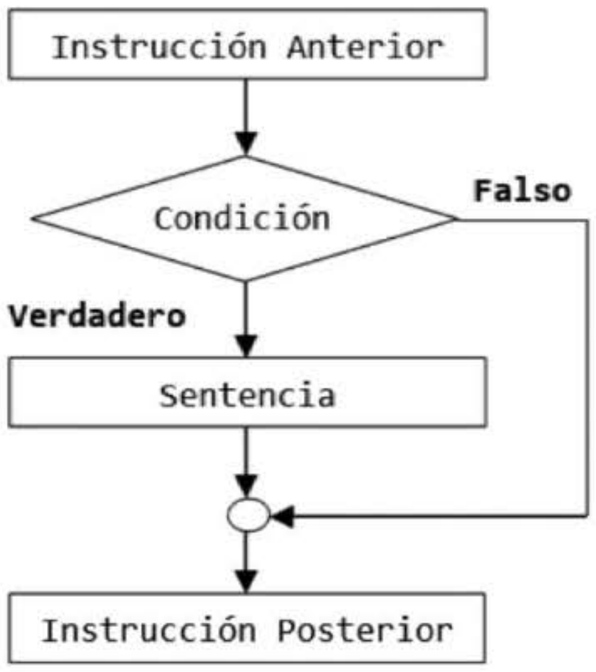
4.3 Instrucción if simple

Se le denomina simple cuando se usa solo una sección de la sentencia condicionada; si la condición evaluada resulta true, ejecuta una o más sentencias, caso contrario, no ejecuta nada y continúa con la siguiente sentencia de la aplicación.

Véanse los siguientes formatos:

```
Formato 1:
If (condicion) sentencia;

Formato 2:
If (condicion) {
sentencia1;
sentencia2;
}
```



Donde:

- **Condición:** Es la expresión que determina las acciones a realizar si el resultado de esta devuelve true o verdadero. Asimismo, se debe considerar que es aquí donde se usan los operadores lógicos y relacionales.
- **Sentencia:** Es la expresión que se ejecuta si y solo si la condición evaluada es verdadera. Se debe tener en cuenta que si se trata de una sola sentencia, no es necesario el uso de la pareja de llaves como se observa en el formato 1. Si se implementan más de dos sentencias, obligatoriamente deben estar encerradas entre llaves como se visualiza en el formato 2.

Implementación de sentencias If simple:

1. Determinar el nombre de la categoría de un empleado según la siguiente tabla:

Categoría	Descripción
A	Jefe
B	Administrativo
C	Apoyo
D	Operario

```
string categoria, descripcion;
categoria = "A";

if (categoria =="A") descripción = "Jefe";
if (categoria =="B") descripción = "Adiministrativo";
if (categoria =="C") descripción = "Apoyo";
if (categoria =="D") descripción = "Operario";
```

2. 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

```
double sueldo, descuento;
sueldo = 1750.00;

if (sueldo <= 750)                                descuento = sueldo * (7.5/100);
if (sueldo > 750 && sueldo <= 1400)  descuento = sueldo * (12.5/100);
if (sueldo > 1400 && sueldo <= 2100) descuento = sueldo * (14.5/100);
if (sueldo > 2100)                                descuento = sueldo * (16.5/100);
```

3. 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 %

```
string categoria;
double costoHora, descuento;

if (categoria  == "Jefe")    {
      costoHora = 50;
      descuento = 0.18;
}
if (categoria  == "Administrativo")  {
      costoHora = 30;
      descuento = 0.15;
}
if (categoria  == "Apoyo")   {
      costoHora = 20;
      descuento = 0.12;
}
if (categoria  == "Operario")        {
      costoHora = 7.5;
      descuento = 0.1;
}
```

Consideraciones finales de las sentencias If simple:

- 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, ejecuta las sentencias implementadas.
- Si la condición resulta falsa, no ejecuta ninguna sentencia y la aplicación continua a la siguiente sentencia especificada después del if.
- Cuando hay una sola sentencia, no es necesario el uso de llaves.
- Cuando se especifican varias sentencias, es necesario el uso de las llaves, ya que estas delimitan el inicio y fin de las sentencias.
- Si se realiza una comparación múltiple, se usan varios bloques if.

4.4 Instrucción if doble

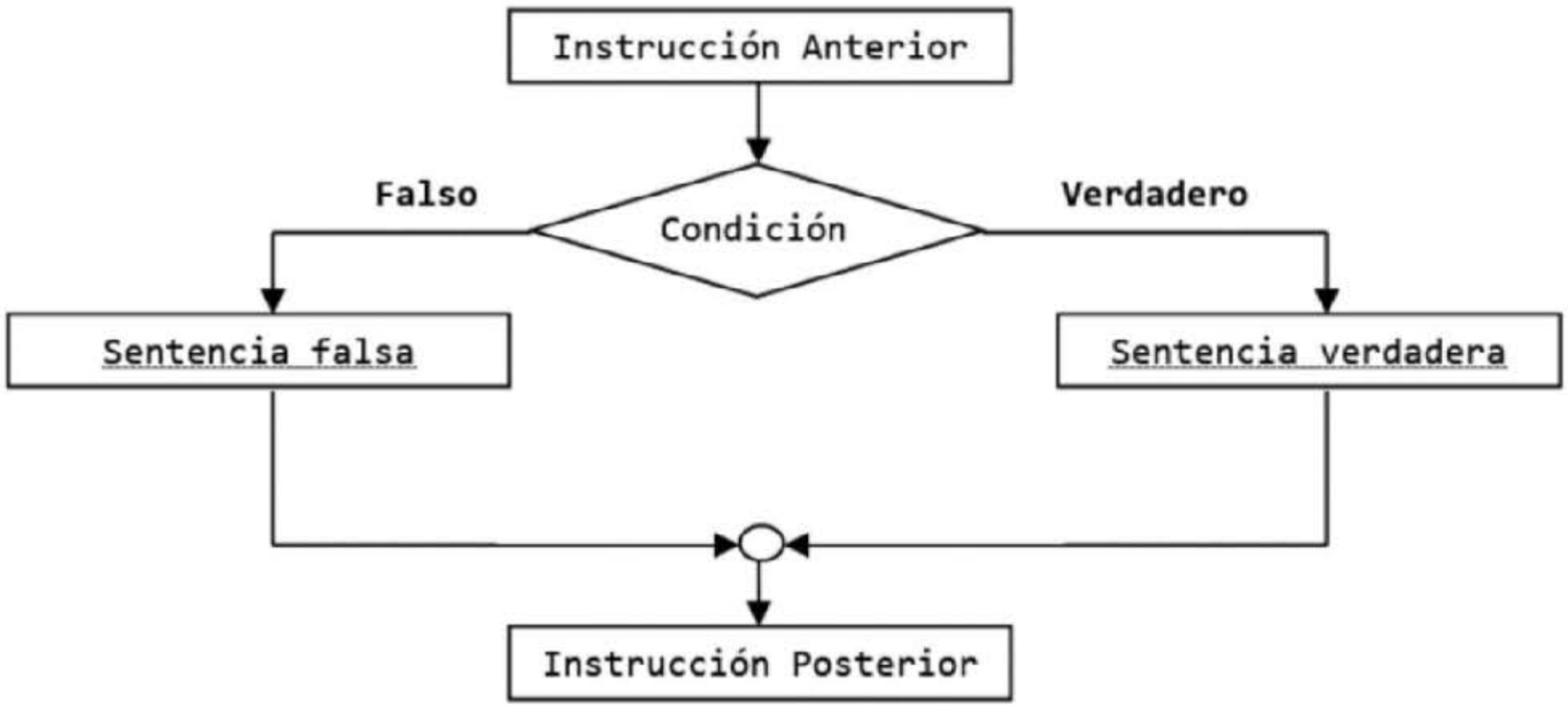
Se le denomina así cuando se usan ambas secciones de la sentencia selectiva if. Cuando la condición evaluada resulta true o verdadero, ejecuta sentencias implementadas en dicha sección; en cambio, si la condición resulta false o falso, ejecuta otro bloque de sentencias. Se debe tener en cuenta que cuando se ejecuta un bloque de sentencias de una sección, ya no se ejecuta el otro bloque y la aplicación continúa con el código especificado.

Véanse los siguientes formatos:

```
Primer formato:
if (condicion) sentencia_verdadera; else sentencia_falsa;

Segundo formato:
if (condicion) {
   sentencia_verdadera1;
   sentencia_verdadera2;
  }
else {
   sentencia_falsa1;
   sentencia_falsa2;
}
```

Gráficamente:



Donde:

- **Condición**: Es la expresión que determina las acciones a realizar si el resultado de esta devuelve true o verdadero. Asimismo, aquí se usan los operadores lógicos y relacionales.
- **Sentencia_verdadera**: Sentencia que se ejecuta solo si la condición evaluada resulta verdadera.
- **Sentencia_falsa**: Sentencia que se ejecuta solo si la condición evaluada resulta falsa.

Implementación de sentencias If doble:

1. Determinar los siguientes mensajes según la realización de los pagos de cuotas en una universidad, según la siguiente tabla:

Condición	Mensaje
PAGO	"Estimado estudiante usted se encuentre al día, en sus pagos".
NO PAGO	"Estimado estudiante recuerde que el retraso en sus cuotas genera moras".

```
string condicion, mensaje;

if (condicion == "PAGO")
      mensaje="Estimado estudiante usted se encuentre al día, en sus pagos";
      else
      ensaje="Estimado estudiante recuerde que el retraso en
          sus cuotas genera moras";
```

2. 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 %

```
double sueldo, descuento1, descuento2;
string tipoAFP;

if (tipoAFP =="Integra"){
      descuento1 = sueldo * (10.0/100);
      descuento2 = sueldo * (2.0/100);
}
else
{
      descuento1 = sueldo * (11.0/100);
      descuento2 = sueldo * (1.75/100);
}
```

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 el falso.
- Solo si la condición resulta verdadera, se ejecutan las sentencias implementadas en esa sección.
- Si la condición resulta falsa, se ejecutan las sentencias implementadas en esa sección. También es llamada sentencia por defecto.
- Cuando hay una sola sentencia en cualquier sección, no es necesario el uso de llaves.
- Cuando se especifican varias sentencias en cualquier sección; es necesario el uso de las llaves, ya que estas delimitan el inicio y fin de dichas sentencias.

4.5 Instrucción if doblemente enlazada

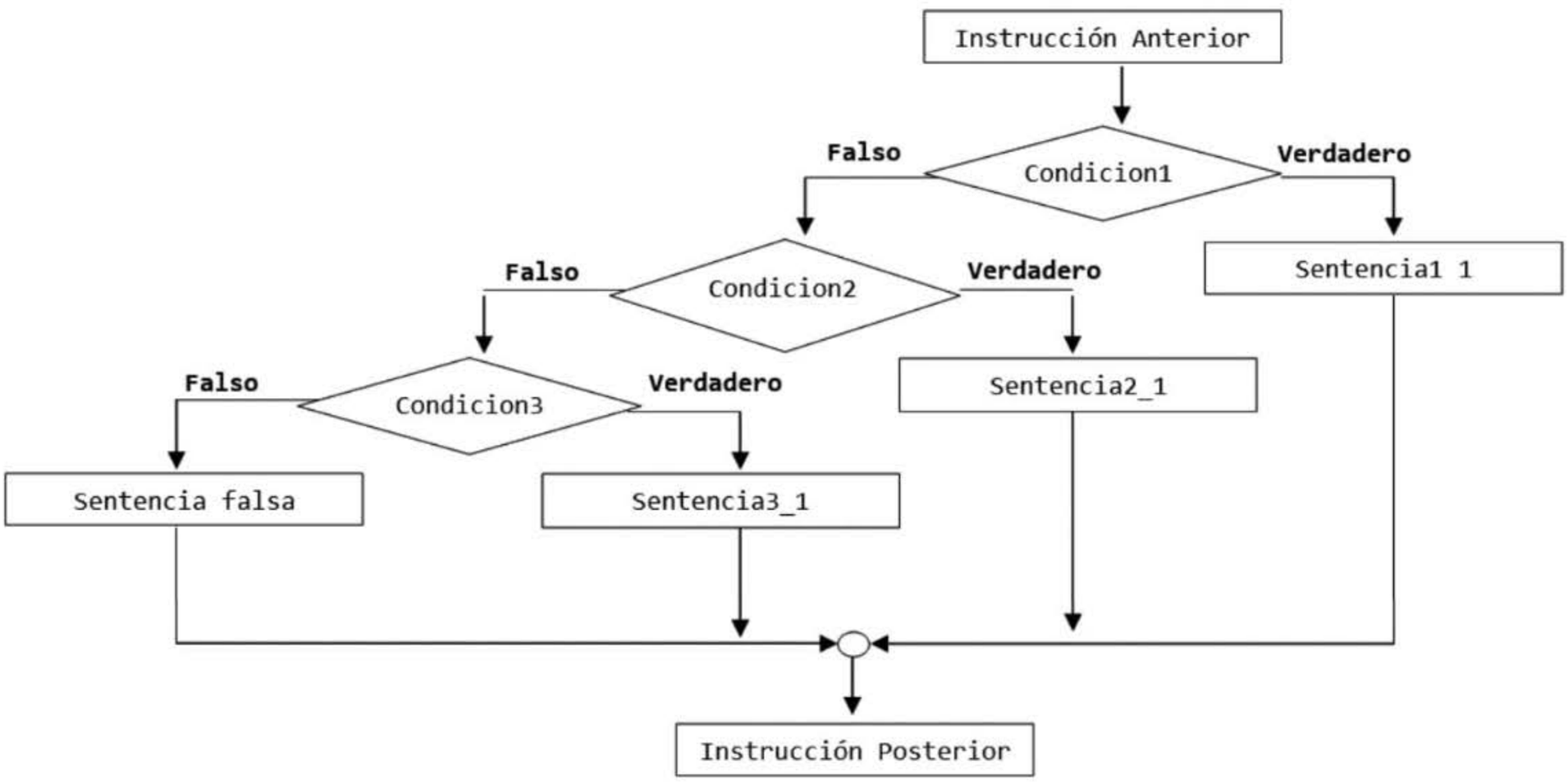
Se denomina así a la evaluación sucesiva que puede realizarse en una sentencia if. Si la condición evaluada resulta verdadera, entonces se ejecutan las sentencias de dicho bloque y termina el ciclo selectivo; es decir, ya no realiza más comparaciones. Caso contrario, sigue evaluando la siguiente condición hasta encontrar el valor verdadero.

Véanse los siguientes formatos:

```
Primera forma:
if (condicion1)
   sentencia1_1;
   else if (condicion2)
   sentencia2_1;
   else if (condicion3)
   sentencia3_1;
   else
   sentencia_falsa;

Segunda forma:
if (condicion1){
    sentencia1_1;
    sentencia1_2;
   }
   else if (condicion2) {
    sentencia2_1;
    sentencia2_2;
   }
   else if (condicion3) {
    sentencia3_1;
    sentencia3_2;
   }
   else{
    sentenciaFalsa1;
    sentenciaFalsa2;
   }
```

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 sentencias que se ejecuta única y exclusivamente si la condición evaluada es verdadera, caso contrario, sigue evaluando hasta que la condición sea verdadera.
- **Sentencia_falsa**: Representa las sentencias que se ejecutan cuando ninguna de las condiciones resulta verdadera; es también llamada sentencia ejecutada por defecto.

Implementación de sentencias If doblemente enlazada

1. Determinar el nombre de la categoría de un empleado según la siguiente tabla:

Categoría	Descripción
A	Jefe
B	Administrativo
C	Apoyo
D	Operario

```
string categoria, descripcion;

if (categoria =="A")
      descripción = "Jefe";
      else if (categoria =="B")
               descripción = "Adiministrativo";
      else if (categoria =="C")
               descripción = "Apoyo";
      else if (categoria =="D")
               descripción = "Operario";
```

2. 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

```
double sueldo, descuento;

if (sueldo <= 750)
      descuento = sueldo * (7.5/100);
      else if (sueldo > 750 && sueldo <= 1400)
               descuento = sueldo * (12.5/100);
      else if (sueldo > 1400 && sueldo <= 2100)
               descuento = sueldo * (14.5/100);
      else if (sueldo > 2100)
               descuento = sueldo * (16.5/100);
```

3. 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 %

```
string categoria;
double costoHora, descuento;

if (categoria  == "Jefe")    {
      costoHora = 50;
      descuento = 0.18;
      } else if (categoria  == "Administrativo") {
               costoHora = 30;
               descuento = 0.15;
      } else if (categoria  == "Apoyo") {
               costoHora = 20;
               descuento = 0.12;
      } else if (categoria  == "Operario")   {
               costoHora = 7.5;
               descuento = 0.1;
      }
```

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, ejecutan las sentencias implementadas en esa sección y termina el proceso condicional.

- Si la condición resulta falsa, se busca la siguiente condición.
- Cuando hay una sola sentencia, no es necesario el uso de llaves.
- Cuando se especifican varias sentencias, es necesario el uso de las llaves, ya que estas delimitan el inicio y fin de las sentencias.
- Cuando ninguno de los valores cumple con la especificación de la condición se ejecutan las sentencias especificadas en el bloque "else"; si no existe implementación, se ejecutan las sentencias próximas a la sentencia if.

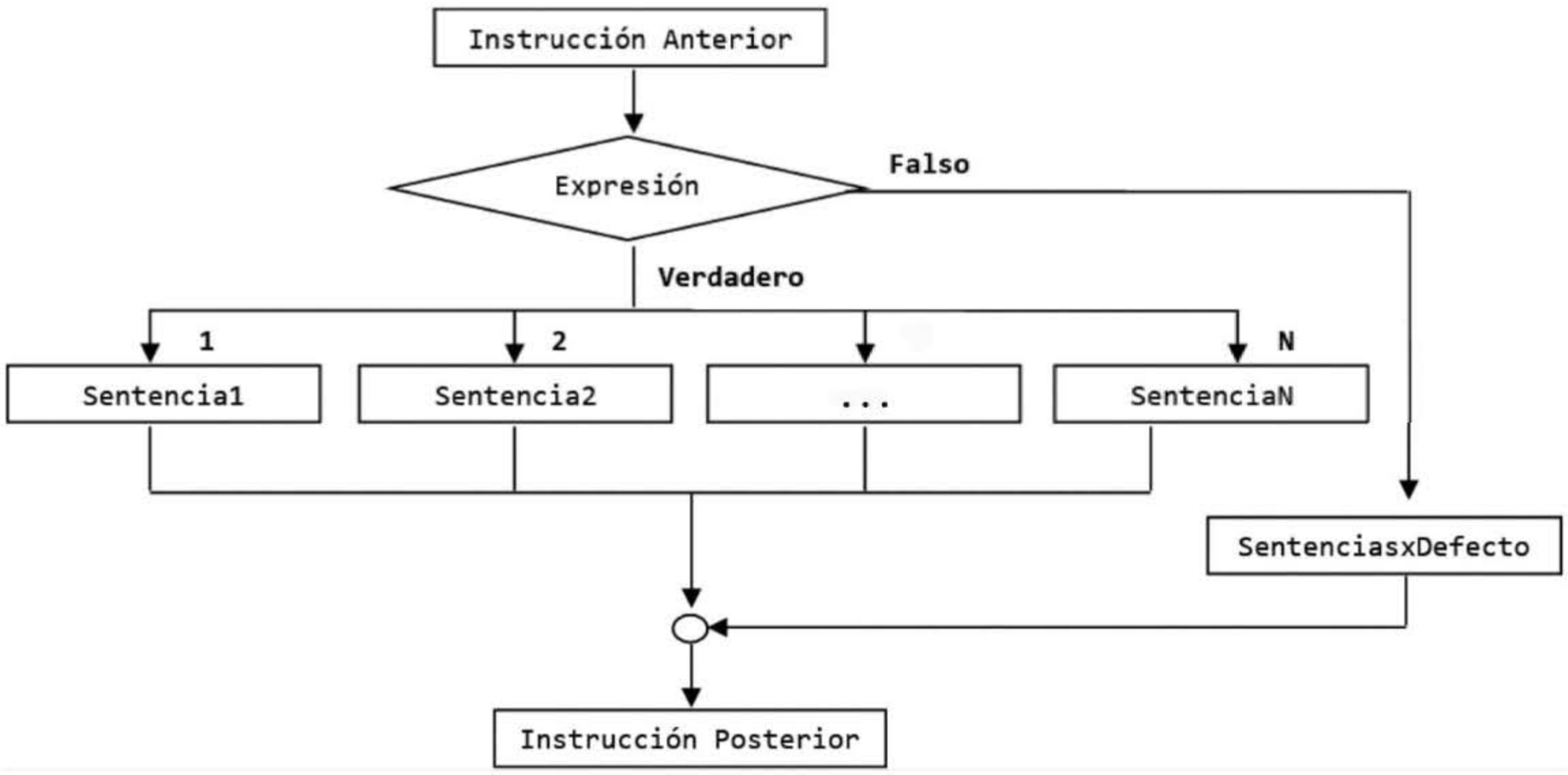
4.6 Instrucción condicional múltiple

La finalidad de la instrucción es implementar sentencias condicionales por opción multiple; es muy parecido al trabajo que realiza la condicional enlazada del if.

Véase el siguiente formato:

```
switch(expresion){
 case valor: sentencia1; break;
 case valor: sentencia2; break;
 case valor: sentencia3; break;
 default: sentenciaxdefecto; break;
}
```

Gráficamente:



Donde:

- **Expresión**: Es el dato que evalúa la estructura switch, cuando el valor de la expresión se encuentra entre las opciones del switch, se ejecutan las sentencias especificadas en dicho caso.
- **break**: Especifica el término del bloque de sentencias en un caso; No es necesario asignar llaves a cada bloque de sentencias. El bloque de sentencias termina solo si encuentra un break; quiere decir que es obligatoria en todos sus ámbitos, inclusive en el último caso.
- **case valor**: Representa una de las posibles opciones que puede tomar la expresión evaluada, aquí se debe considerar la forma en que se especifica un valor. Se tiene:

Especificación de valor	Especificación SWITCH

case valor numérico	`int n;`
	`switch (n){`
	`  case 1: break;`
	`}`

case "valor cadena"	`string cadena;`
	`switch (cadena){`
	`  case "valor": break;`
	`}`

- **default**: Las sentencias especificadas en esta sección son ejecutadas cuando la expresión evaluada no toma ninguna de los valores especificados, se debe tener en cuenta que al final del bloque de sentencias debe asignar un break.

Implementación de la estructura switch:

- Determinar el nombre de la categoría de un empleado según la siguiente tabla:

Categoría	Descripción
A	Jefe
B	Administrativo
C	Apoyo
D	Operario

```
string categoria, descripcion;

switch(CATEGORIA){
      case "A": descripción="Jefe"; break;
      case "B": descripción="Administrativo"; break;
      case "C": descripción="Apoyo"; break;
      case "D": descripción="Operario"; break;
}
```

- 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 %

```
string categoria;
double costoHora, descuento;

switch(categoria){
      case "Jefe":
                  costoHora = 50;
                  descuento = 0.18;
                  break;
case "Administrativo":
                  costoHora = 30;
                  descuento = 0.15;
                  break;
case "Apoyo":
                  costoHora = 20;
                  descuento = 0.12;
                  break;
case "Operario":
                  costoHora = 7.5;
                  descuento = 0.1;
                  break;
}
```

Consideraciones finales de las sentencias If doblemente enlazada:

- Los valores aceptados para la búsqueda solo pueden ser de tipo int, char, string o bool.
- Para implementar una condición, use la combinación de variables y operadores.
- Por regla general, toda implementación de case debe tener sentencias de acción.
- Solo si el valor buscado se encuentra en la lista de casos, se ejecutan las sentencias implementadas en esa sección y termina el proceso gracias a la sentencia break.
- Si no se encuentra el valor, se busca en los siguientes casos.
- Un bloque de sentencias no necesita llaves; solo debe asignar llaves si dentro del bloque especifica sentencias como if, for o switch.
- Cuando ninguno de los valores cumple con la especificación de la condición, se ejecuta el caso default, el cual es conocido como caso por defecto.

4.7 Casos desarrollados

Caso desarrollado 1 : if simple - pago de empleados

La empresa de consultoría de proyectos Project-Perú ha decidido ampliar su línea de negocio, por lo cual ha contratado más personal para así poder lograr los objetivos trazados por la gerencia. Por lo tanto, se necesita una aplicación que permita controlar el pago de sus empleados según su categoría; mostrando el sueldo asignado, descuento y el importe neto a recibir.

Se tiene en cuenta:

- Al iniciar la aplicación, se debe mostrar la fecha actual de registro; además de mostrar las categorías de los empleados en un cuadro combinado.
- Usar la siguiente tabla de asignación de sueldos según la categoría del empleado:

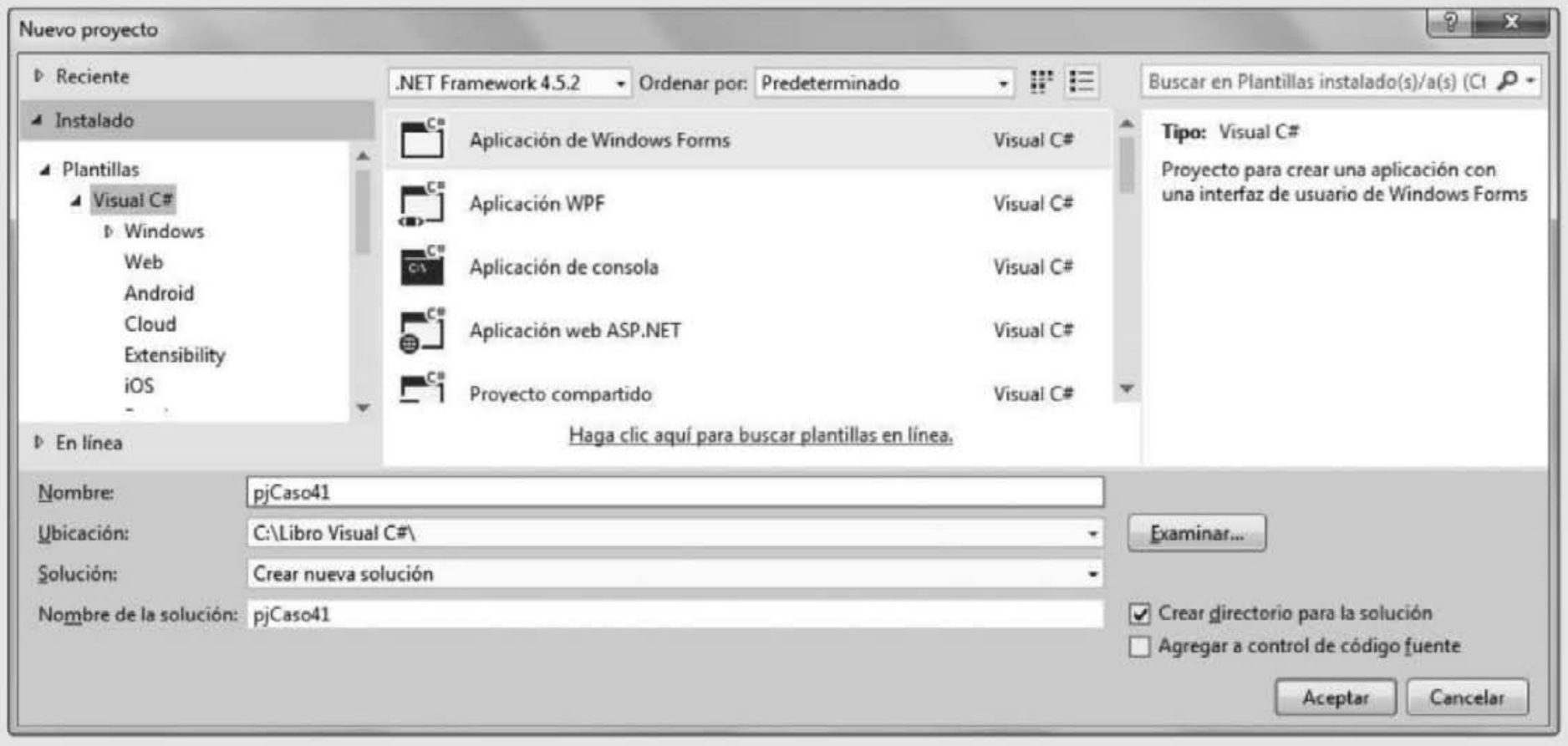
Categoría	Sueldo
Jefe	$ 3500.00
Administrativo	$ 2500.00
Técnico	$ 1700.00
Operario	$ 1000.00

- El descuento aplicado representa el 12.5 % del sueldo solo si dicho sueldo es mayor a $ 2000.00; caso contrario, no se le aplica descuento.
- El importe neto resulta de la diferencia entre el sueldo y el descuento aplicado.
- Todos los resultados deben ser mostrados en un cuadro de lista.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#.**
   - Seleccione **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso41**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

   - Seleccione el formulario Form1.cs desde el Explorador de soluciones.
   - Desde la propiedad Nombre de archivo, asigne el nombre frmPago.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:

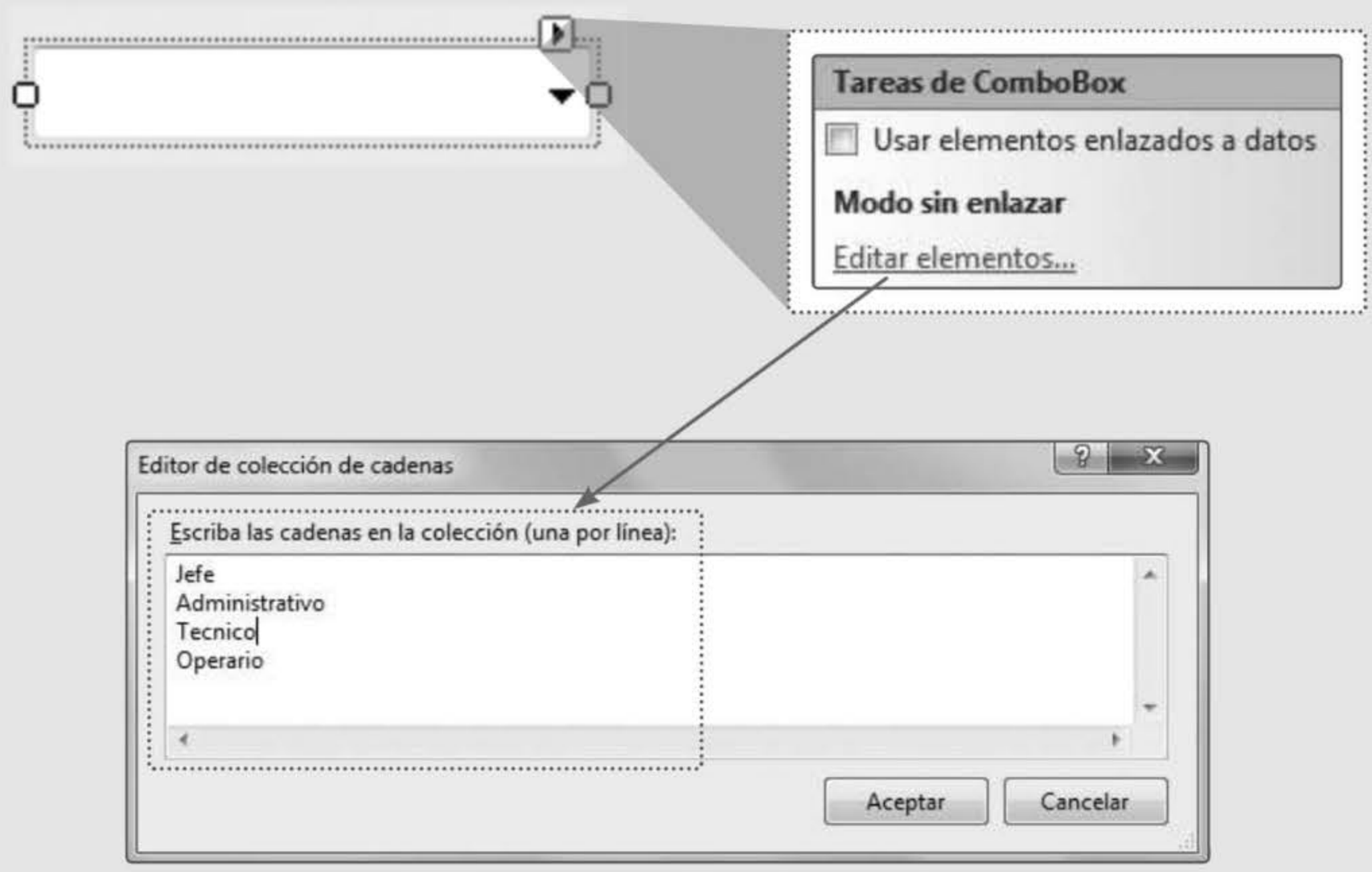


Estadísticas de los controles usados en el frmPago:

Clase	N.° elementos	Controles
Form	01	Form1
Label	07	Label1, Label2, Label3, Label4, Label5, Label6, Label7
TextBox	01	TextBox1
ComboBox	01	ComboBox1
Button	03	Button1, Button2, Button3
ListView	01	ListView1
TOTAL	14	

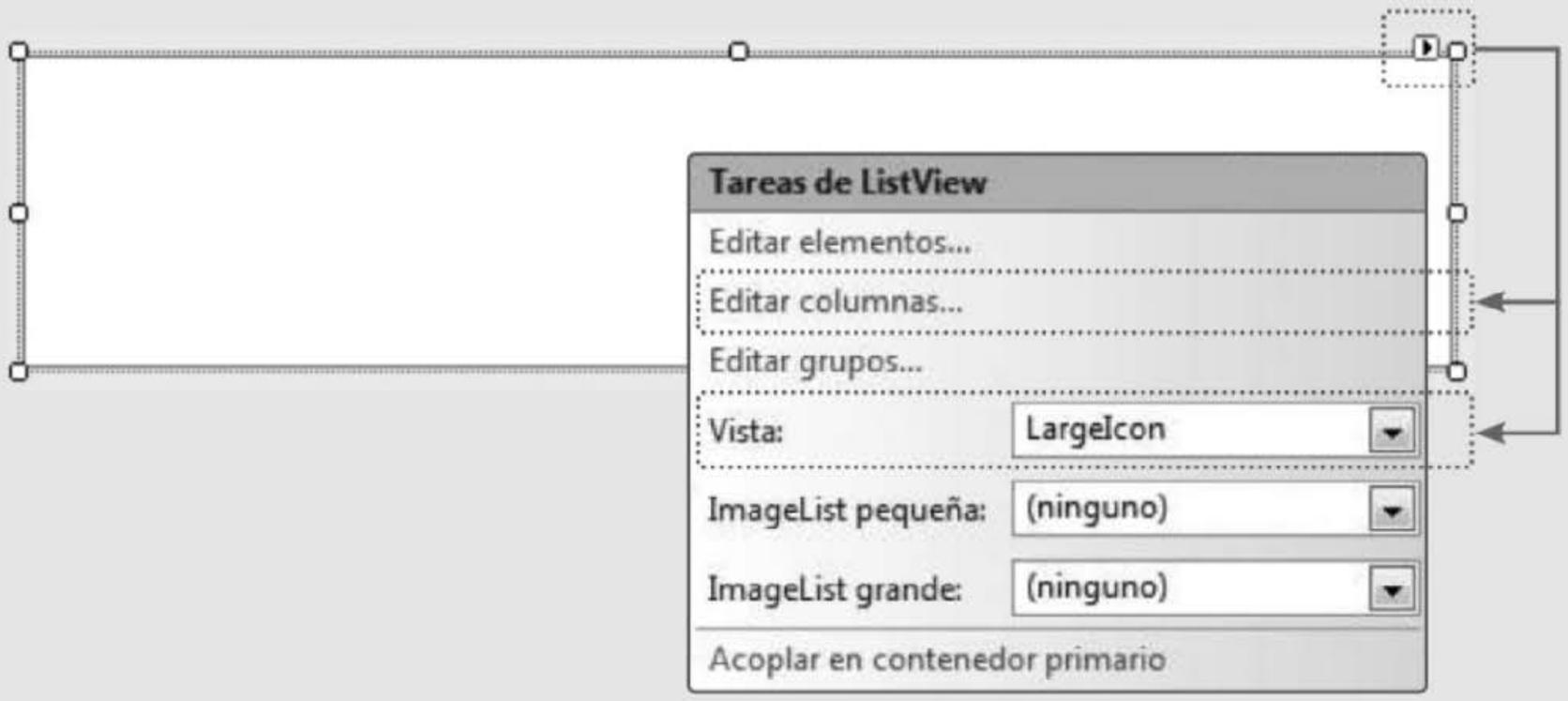
Configurando el cuadro combinado (ComboBox):

- Seleccione el cuadro combinado (ComboBox1).
- Seleccione la pestaña superior derecha del objeto > **Editar elementos…**
- Coloque las categorías de los empleados uno debajo del otro

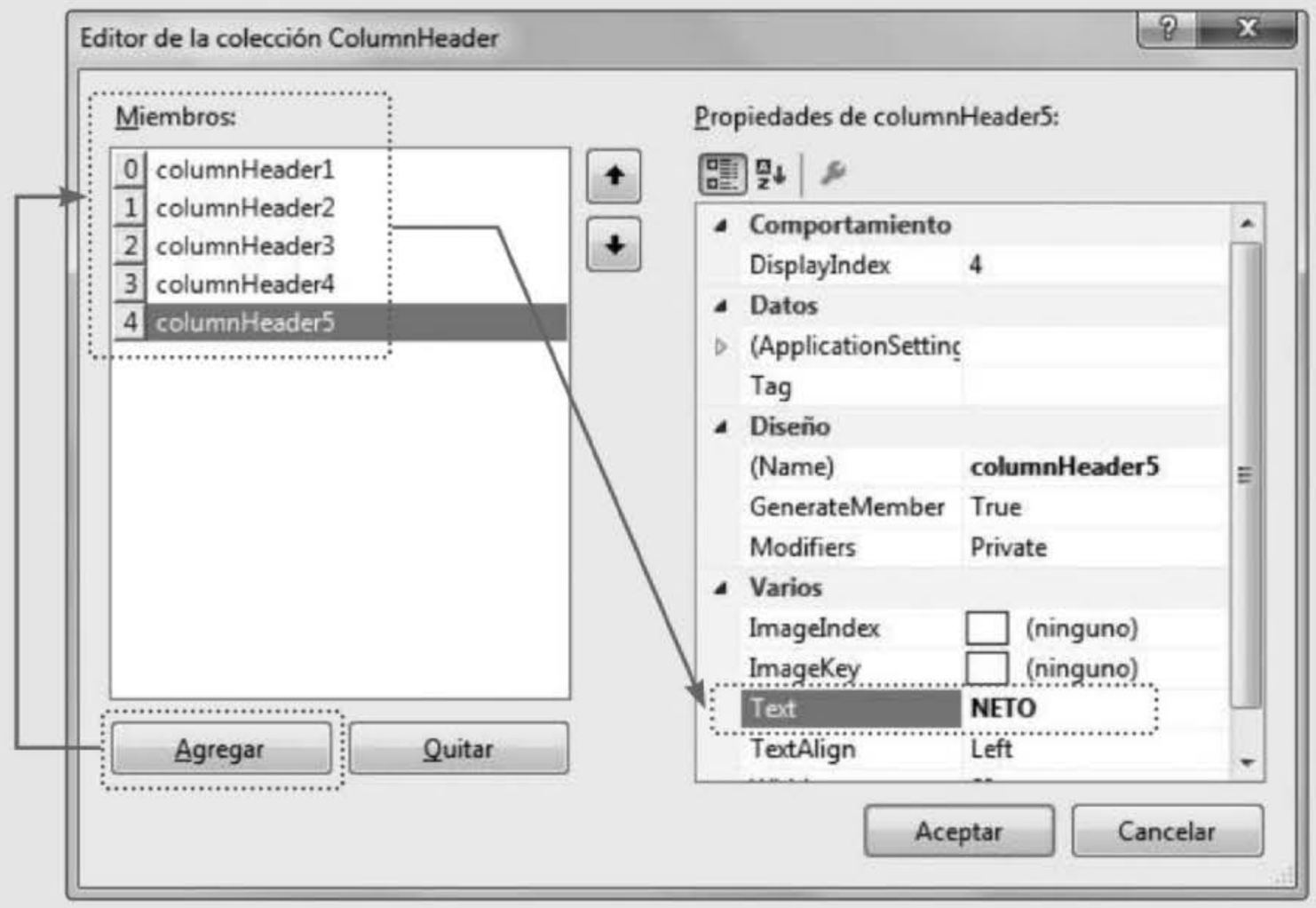


Configurando el ListView:

- Seleccione el objeto **ListView1**.
- Seleccione la pestaña superior derecha.
- Seleccione **Details**, desde el cuadro **Vista**.
- Luego, presione **Editar columnas…** para definir las columnas del cuadro de lista.



- Ahora, asigne los títulos de la lista, para lo cual debe agregar 5 columnas con el botón **Agregar**.

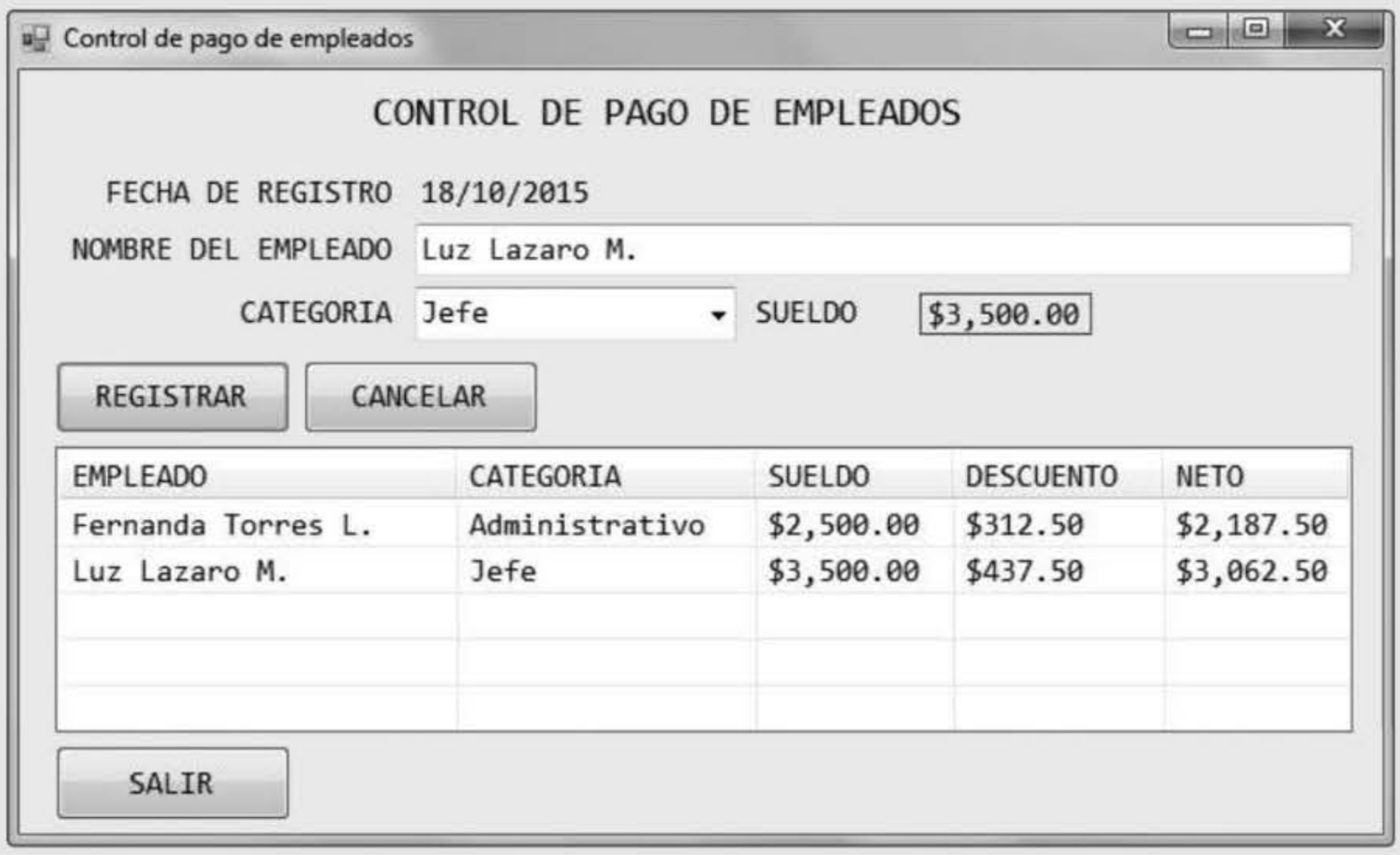


- Para modificar el título, seleccione cada **columnHeader** y modifique su propiedad Text; no es necesario asignar un nombre (Name).
- Cuando finalice de asignar los títulos, presione el botón **Aceptar**.

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmPago Control de pago de empleados
Label1	Text	CONTROL DE PAGO DE EMPLEADOS
Label2	Text	FECHA DE REGISTRO
Label3	(Name) Text	lblFecha Dejar vacio
Label4	Text	NOMBRE DEL EMPLEADO
Label5	Text	CATEGORÍA
Label6	Text	SUELDO
Label7	(Name) Text	lblSueldo Dejar vacio
Textbox1	(Name)	txtEmpleado
ComboBox1	(Name) Text	cboCategoria (Seleccione)
Button1	(Name) Text	bntRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvPagos True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso41
{
    public partial class frmPago : Form
    {
        double sueldo = 0;
        public frmPago()
        {
            InitializeComponent();
        }

        private void frmPago_Load(object sender, EventArgs e)
        {
            lblFecha.Text = DateTime.Today.Date.ToString("d");
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los valores
            string empleado = txtEmpleado.Text;
            string categoria = cboCategoria.Text;

            //Realizando calculos
            double descuento=0;
            if (sueldo > 2000) descuento = sueldo * (12.5/100);

            double neto = sueldo - descuento;

            //Imprimiendo los resultados en la lista
            ListViewItem fila = new ListViewItem(empleado);
            fila.SubItems.Add(categoria);
            fila.SubItems.Add(sueldo.ToString("C"));
            fila.SubItems.Add(descuento.ToString("C"));
            fila.SubItems.Add(neto.ToString("C"));
            lvPagos.Items.Add(fila);

            //Limpiando los controles
            btnCancelar_Click(sender, e);
        }
```

```
        private void cboCategoria_SelectedIndexChanged(object sender, Even-
tArgs e)
        {
            //Capturando la categoria seleccionada
            string categoria=cboCategoria.Text;

            //Asignando el sueldo segun la categoria
            if (categoria == "Jefe") sueldo = 3500;
            if (categoria == "Administrativo") sueldo = 2500;
            if (categoria == "Tecnico") sueldo = 1700;
            if (categoria == "Operario") sueldo = 1000;

            //Enviando el sueldo obtenido a la impresion
            lblSueldo.Text = sueldo.ToString("C");
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            cboCategoria.Text = "(Seleccione)";
            txtEmpleado.Clear();
            txtEmpleado.Focus();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
          DialogResult r = MessageBox.Show("Esta seguro de salir?","Pago",
                                           MessageBoxButtons.YesNo,
                                           MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Mostrar la hora actual en el formulario**

  Se debe tener en cuenta que al ejecutar la aplicación, la fecha ya debe mostrarse en el formulario; por lo tanto, la programación debe hacerla en el evento del load del formulario. Esto indica que al cargar el formulario, se muestra le fecha. Para llegar al evento load, solo basta hacer doble clic en un lugar vacío del fondo del formulario. Inicialmente, se muestra de la siguiente manera:

```
private void frmPago_Load(object sender, EventArgs e)
{
     //Codigo del evento load del formulario
}
```

Dentro del evento load se hace referencia a DateTime.Today.Date.ToString("d"). DateTime contiene una propiedad llamada "Date", que captura la fecha y la hora actual del sistema; para mostrar solo la fecha se asigna la letra "d" dentro del metodo ToString; si se necesita visualizar la fecha y la hora, se debe dejar vacío el metodo ToString.

- **Asignar el sueldo como variable GLOBAL**

  La idea principal es mantener la información que resulta del sueldo del empleado, la cual puede ser usada tanto al seleccionar una categoría del empleado como en los cálculos de sus montos. Esta debe ser declarada después de la línea:

```
public partial class frmPago : Form
     {
            double sueldo = 0;
       }
```

- **Mostrar el sueldo del empleado al seleccionar una categoría**

  Al seleccionar una categoría automáticamente, debe mostrarse el sueldo asignado según el criterio de la aplicación. Para colocar el código dentro del cuadro combinado, solo se debe hacer doble clic sobre dicho control, de forma que muestre el siguiente código:

```
private void cboCategoria_SelectedIndexChanged(object sender, EventArgs e)
     {
            //Código del evento selectedIndexChanged del cuadro combinado
 }
```

- **Enviar los valores al ListView**

  La información resultante de la aplicación debe ser mostrada en el objeto ListView, por lo tanto, a continuación se analiza su código:

```
ListViewItem fila = new ListViewItem(empleado);
fila.SubItems.Add(categoria);
fila.SubItems.Add(sueldo.ToString("C"));
fila.SubItems.Add(descuento.ToString("C"));
fila.SubItems.Add(neto.ToString("C"));
lvPagos.Items.Add(fila);
```

Para agregar información al cuadro de lista, se necesita la clase "ListViewItem" la cual representa un elemento del ListView. Los valores a enviar deben ser estrictamente ordenados, es decir, deben estar en el mismo orden definido en la cabecera. El objeto fila representa a los valores que se muestran en el ListView. Por lo tanto, toda la

información a mostrar debe ser enviada a dicho objeto mediante el método "SubItems.Add". Una vez llena de todos los valores, es momento de enviar el objeto ListView mediante la sentencia lvPagos.Items.Add(fila).

➤ **Invocar el código del objeto btnCancelar**

Después de registrar el pago de un empleado, la aplicación debe permitir limpiar los controles para volver a enfocar en el cuadro de ingreso de nombre del empleado, para lo cual, se cuenta con el botón Cancelar, que casualmente realiza las mismas actividades. Entonces, se debe invocar su código. Por lo tanto, al finalizar el registro del pago de empleado, se debe colocar el siguiente código btnCancelar_Click(sender, e); el cual invoca a todo el código implementado en el botón Cancelar.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 2 : if simple - control de registro de documentos

La notaría Garcia-DosFuentes cuenta con el proceso de control de documentos manual, para lo cual, un empleado registra todo tipo de documentos recibidos. Se necesita una aplicación que permita controlar dichos documentos de forma automática. Se deben mostrar los documentos en una lista con el número de documento, fecha de emisión, empresa emisora, años de registro y la condición del documento.

Se tiene en cuenta lo siguiente:

- Para la selección de la fecha, usar el control **DateTimePicker**.
- Para calcular los años de registro, considerar la fecha de emisión del documento con respecto al año actual.
- Usar la siguiente tabla para determinar la condición del documento:

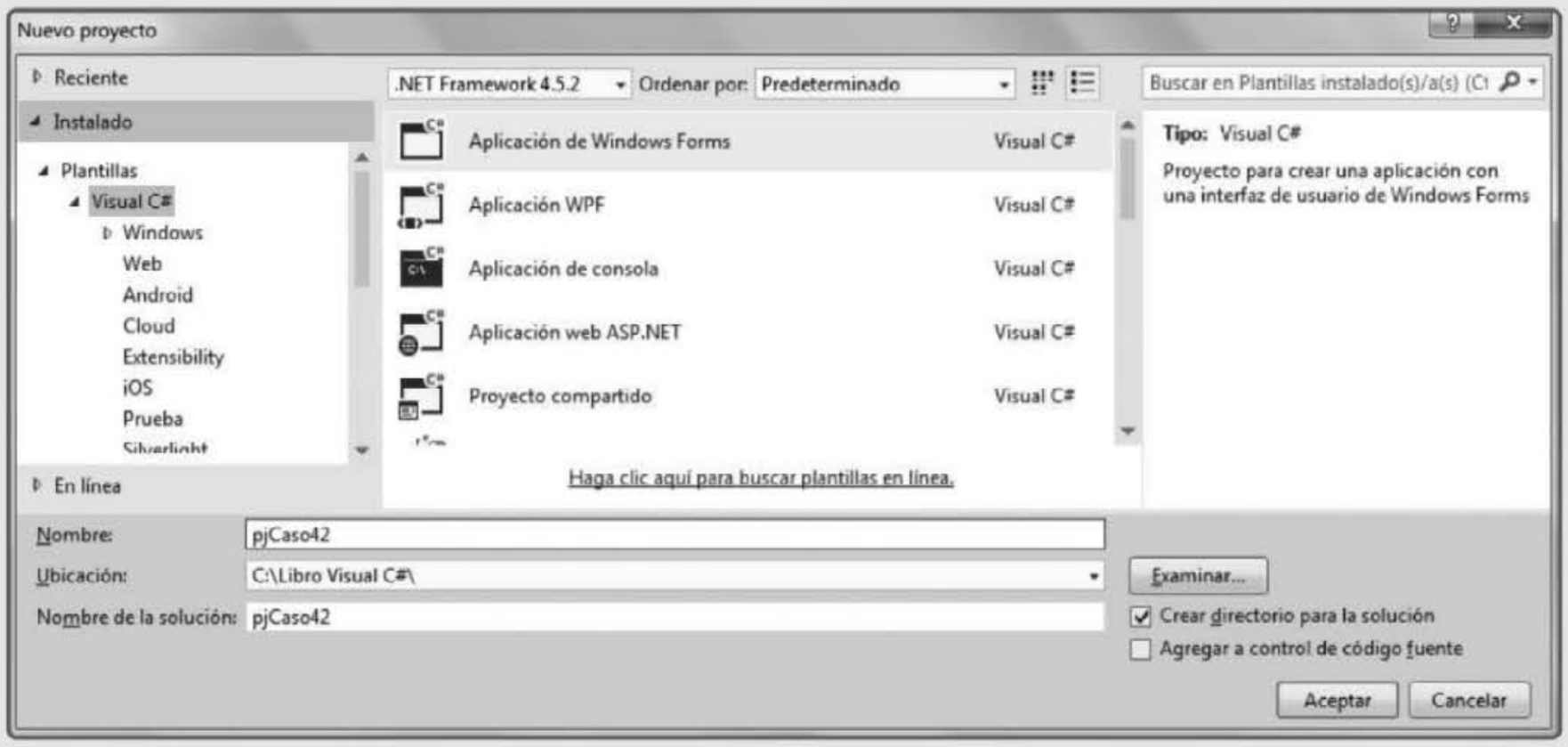
Años de emisión	Condición
Menor o igual a 5 años	Habilitado
Mayor a 5 años	Inhabilitado

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.

- Seleccione **Aplicación de Windows Form**.
- Asigne el nombre **pjCaso42**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario Form1.cs, desde el Explorador de soluciones.
- Desde la propiedad "Nombre de archivo" asigne el nombre frmRegistro.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmRegistro:

Clase	N.º elementos	Controles
Form	01	Form1
Label	04	Label1, Label2, Label3, Label4
TextBox	02	TextBox1, TextBox2
DateTimePicker	01	DateTimePicker1
Button	03	Button1, Button2, Button3
ListView	01	ListView1
TOTAL	12	

4. Modificar las propiedades de los controles

Control	Propiedad	Valor
Form1	(Name) Text	frmRegistro Control de registro de documentos
Label1	Text	CONTROL DE REGISTRO DE DOCUMENTOS
Label2	Text	NUMERO DE DOC.
Label3	Text	FECHA DE EMISION
Label4	Text	EMPRESA EMISORA
Textbox1	(Name)	txtNumero
Textbox2	(Name)	txtEmpresa
DateTimePicker	(Name) Format	dtFecha Short
Button1	(Name) Text	bntRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvRegistro True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso42
{
    public partial class frmRegistro : Form
    {
        public frmRegistro()
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los valores
            int numero = int.Parse(txtNumero.Text);
            DateTime fecha = dtFecha.Value;
```

```
            string empresa = txtEmpresa.Text;

            //Calculando los años de emision
            int años = DateTime.Today.Date.Year-fecha.Year;

            //Determinando la condicion
            string condicion="";
            if (años <= 5) condicion = "HABILITADO";
            if (años > 5) condicion = "INHABILITADO";

            //Imprimiendo los resultados
            ListViewItem fila = new ListViewItem(numero.ToString());
            fila.SubItems.Add(fecha.ToString("d"));
            fila.SubItems.Add(empresa);
            fila.SubItems.Add(años.ToString());
            fila.SubItems.Add(condicion);
            lvRegistro.Items.Add(fila);

            //Limpiando los controles
            btnCancelar_Click(sender,e);
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            txtNumero.Clear();
            dtFecha.Value = DateTime.Today.Date;
            txtEmpresa.Clear();
            txtNumero.Focus();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Documentos",
                                             MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Capturando la fecha desde el control DateTimePicker**

  El control DateTimePicker permite seleccionar una fecha determinada, por tanto, la variable de captura debe ser del tipo DateTime y, para obtener la fecha seleccionada se usa la propiedad Value del objeto DateTimePicker; tal como se muestra en la siguiente sentencia:

```
DateTime fecha = dtFecha.Value;
```

➤ **Calculando los años**

Primero, se debe capturar el año actual, para lo cual se usa la sentencia DateTime. Today.Date.Year; mientras que para obtener el año seleccionado, se usa la propiedad directamente desde la variable fecha, tal como se muestra en el siguiente código:

```
int años = DateTime.Today.Date.Year-fecha.Year
```

➤ **Determinando la condición del documento**

La variable condición almacena el resultado de la evaluación de los años. Así, esta variable se declara fuera de la sentencia If y se inicializa con el valor doble comillas vacías; pues, de otra manera, el compilador emite un error, ya que no es preciso el valor asignado en la condición if, tal como se muestra en el siguiente código:

```
string condicion="";
if (años <= 5) condicion = "HABILITADO";
if (años > 5) condicion = "INHABILITADO";
```

➤ **Enviar los valores al ListView**

La información resultante de la aplicación debe ser mostrada en el objeto ListView, por lo tanto, se analiza el siguiente código:

```
ListViewItem fila = new ListViewItem(numero.ToString());
fila.SubItems.Add(fecha.ToString("d"));
fila.SubItems.Add(empresa);
fila.SubItems.Add(años.ToString());
fila.SubItems.Add(condicion);
lvRegistro.Items.Add(fila);
```

Los datos que son enviados a la lista deben ser de tipo string, por lo cual, el número de documentos debe ser parseado, de la misma manera los años calculados y finalmente la fecha de registro. No se debe olvidar que el parámetro "d" emite solo la fecha.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 3 : if doble - promedio de notas

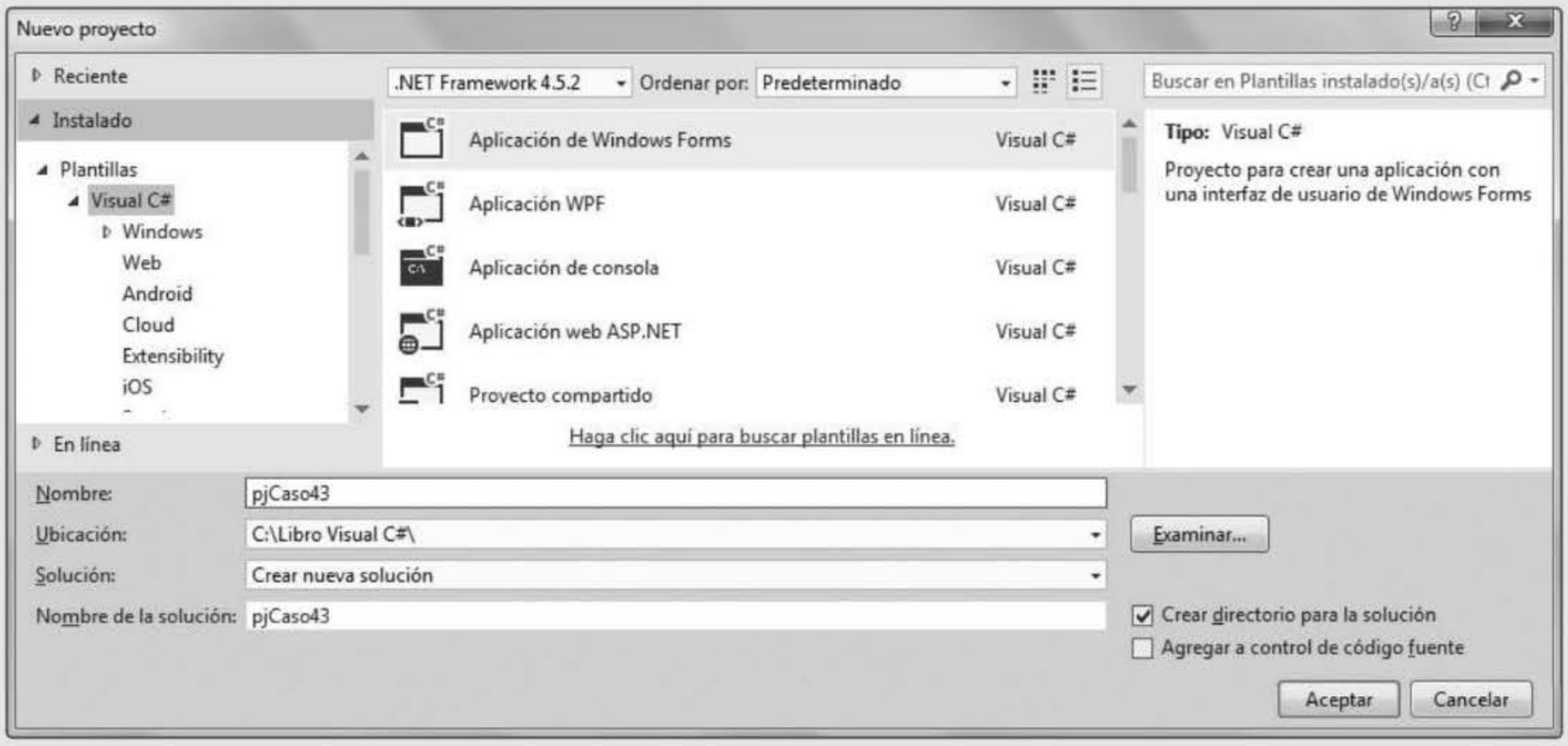
El docente del curso de programación en la universidad necesita tener un control del registro de notas de sus alumnos. El docente registra cuatro evaluaciones y los promedia eliminando la nota baja que el alumno haya obtenido; es decir, se promedian las tres notas más altas. Implemente, una aplicación que permita mostrar la nota menor, el promedio y la condición de los alumnos. La condición considera aprobados a los alumnos cuyo promedio sea mayor o igual a 12.5, caso contrario, se considera desaprobado.

Se tiene en cuenta lo siguiente:

- Validar el ingreso de los valores en los controles, de forma que se use el control Error Provider para mostrar mensajes referentes al error, especialmente, en los controles de ingreso de datos como el TextBox.
- Mostrar el registro de los alumnos en un control ListView.

Solución:

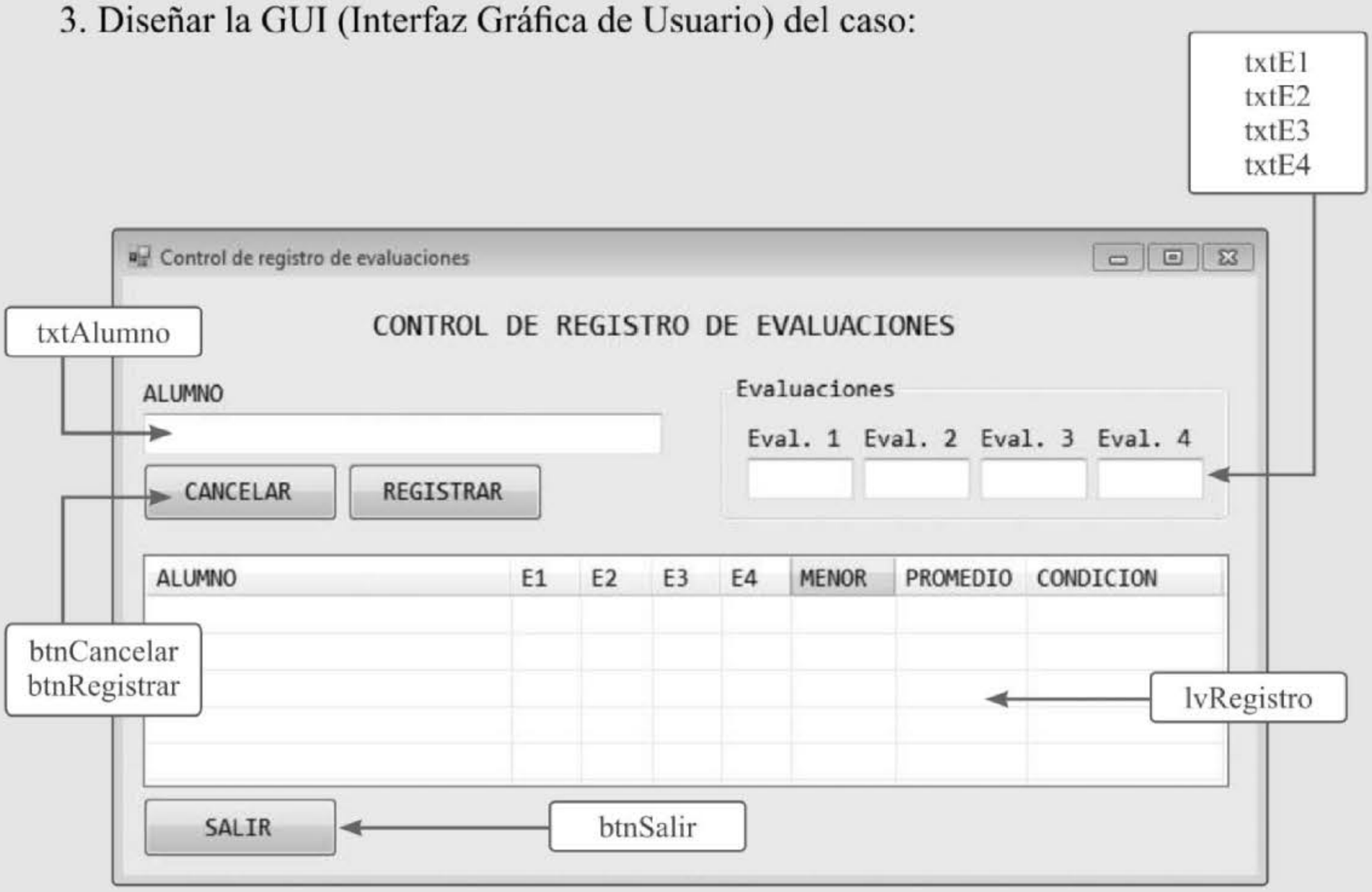
1. Crear un nuevo proyecto en Visual Studio 2015:
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Selección **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso43**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la propiedad **Nombre de archivo**, asigne el nombre **frmEvaluacion**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



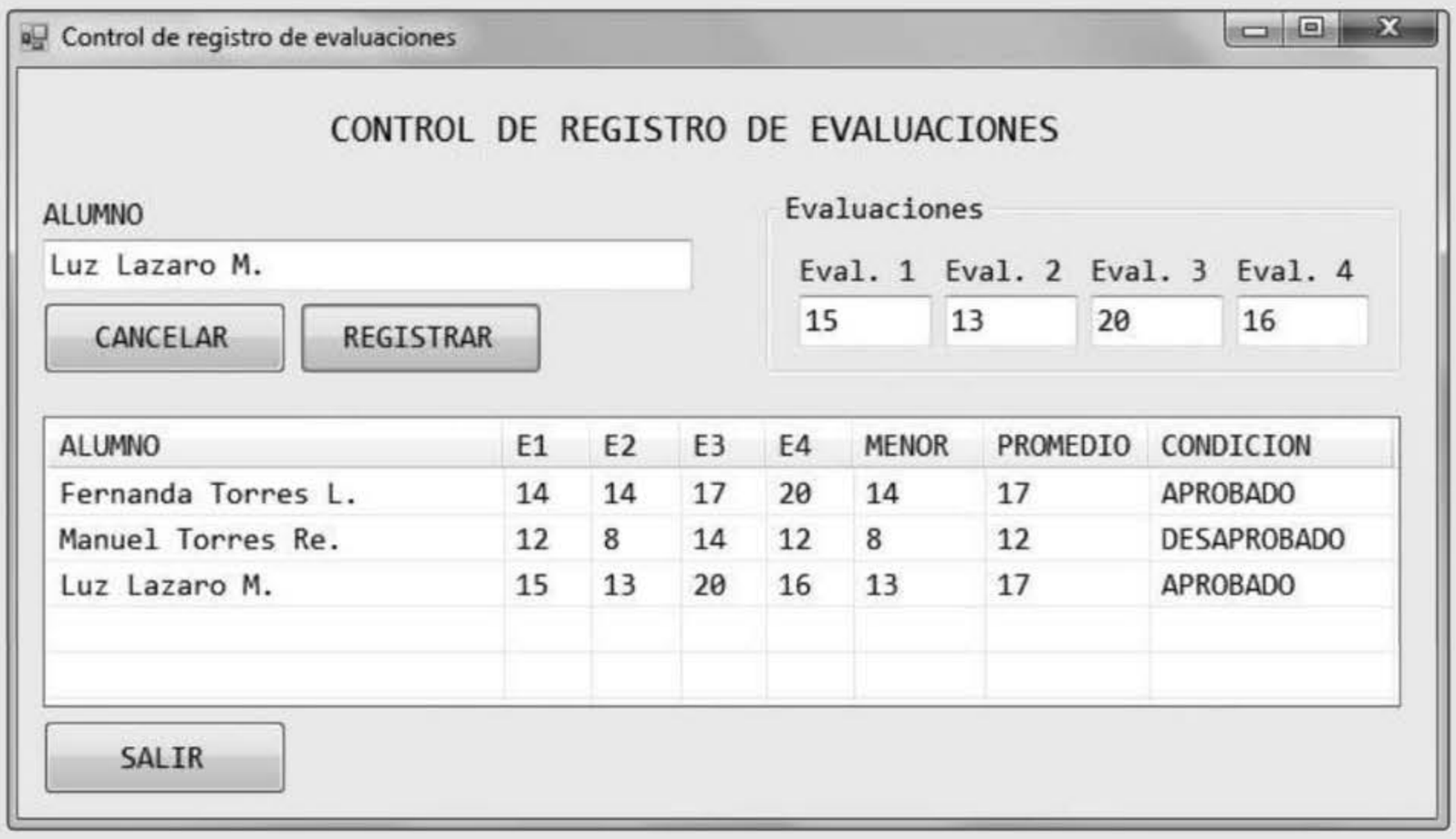
Estadísticas de los controles usados en el frmEvaluacion:

Clase	N.º elementos	Controles
Form	01	Form1
Label	06	Label1, Label2, Label3, Label4, Label5, Label6
TextBox	05	TextBox1, TextBox2, TextBox3, TextBox4, TextBox5
GroupBox	01	GroupBox1
Button	03	Button1, Button2, Button3
ListView	01	ListView1
ErrorProvider	01	ErrorProvider1
TOTAL	18	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmRegistro Control de registro de evaluaciones
Label1	Text	CONTROL DE REGISTRO DE EVALUACIONES
Label2	Text	ALUMNO
Label3	Text	Eval. 1
Label4	Text	Eval. 2
Label5	Text	Eval. 3
Label6	Text	Eval. 4
Textbox1	(Name)	txtNumero
Textbox2	(Name)	txtE1
Textbox3	(Name)	txtE2
Textbox4	(Name)	txtE3
Textbox5	(Name)	txtE4
GroupBox1	Text	Evaluaciones
Button1	(Name) Text	bntRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvRegistro True
ErrorProvider1	(Name)	epError

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso43
{
    public partial class frmEvaluacion : Form
    {
        public frmEvaluacion()
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los valores
            string alumno = txtAlumno.Text;
            int e1 = int.Parse(txtE1.Text);
            int e2 = int.Parse(txtE2.Text);
            int e3 = int.Parse(txtE3.Text);
            int e4 = int.Parse(txtE4.Text);

            //Determinar la menor nota
            int menor;
            if (e1 < e2) menor = e1; else menor = e2;
            if (e3<menor) menor = e3;
            if (e3<menor) menor = e4;

            //Calculando el promedio
            double promedio = (e1 + e2 + e3 + e4 - menor) / 3;

            //Determinado la condicion
            string condicion;
            if (promedio <= 12.49)
                condicion = "DESAPROBADO";
            else
                condicion = "APROBADO";

            //Imprimiendo los resultados
            ListViewItem fila = new ListViewItem(alumno);
            fila.SubItems.Add(e1.ToString());
            fila.SubItems.Add(e2.ToString());
```

```
            fila.SubItems.Add(e3.ToString());
            fila.SubItems.Add(e4.ToString());
            fila.SubItems.Add(menor.ToString());
            fila.SubItems.Add(promedio.ToString());
            fila.SubItems.Add(condicion);
            lvRegistro.Items.Add(fila);

            //Limpiando los controles
            btnCancelar_Click(sender,e);
        }

        private void txtAlumno_Validated(object sender, EventArgs e)
        {
            if (txtAlumno.Text.Trim() == "") {
                epError.SetError(txtAlumno, "Ingrese nombre del alumno...!!!");
             txtAlumno.Focus();
                } else {
                    epError.Clear();
                }
        }

        private void txtE1_Validated(object sender, EventArgs e)
        {
            if (txtE1.Text.Trim() == "")
            {
                epError.SetError(txtE1, "Ingrese evaluacion 1 ...!");
                txtE1.Focus();
            }
            else
            {
                epError.Clear();
            }
        }

        private void txtE2_Validated(object sender, EventArgs e)
        {
            if (txtE2.Text.Trim() == "")
            {
                epError.SetError(txtE2, "Ingrese evaluacion 2 ...!");
                txtE2.Focus();
            }
            else
            {
                epError.Clear();
            }
        }

        private void txtE3_Validated(object sender, EventArgs e)
        {
            if (txtE3.Text.Trim() == "")
            {
                epError.SetError(txtE3, "Ingrese evaluacion 3 ...!");
```

```
            txtE3.Focus();
        }
        else
        {
            epError.Clear();
        }
    }

    private void txtE4_Validated(object sender, EventArgs e)
    {
        if (txtE4.Text.Trim() == "")
        {
            epError.SetError(txtE4, "Ingrese evaluacion 4 ...!");
            txtE4.Focus();
        }
        else
        {
            epError.Clear();
        }
    }

    private void btnCancelar_Click(object sender, EventArgs e)
    {
        txtAlumno.Clear();
            txtE1.Clear();
            txtE2.Clear();
            txtE3.Clear();
            txtE4.Clear();
            txtAlumno.Focus();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                        "Promedio",
                                        MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();

        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Determinar la menor nota de cuatro registradas**

  De las cuatro notas registradas, exclusivamente una de ellas es la menor. Para determinar esto, se deben realizar evaluaciones sucesivas entre los valores, además, contar con una variable adicional llamada menor, que será la encargada de recibir el menor

valor de las notas. Se empieza comparando las dos primeras notas, si e1 es menor que e2 entonces la variable menor almacena e1, caso contrario la almacena e2. El resultado de esta evaluación se compara con la tercera evaluación, de aquí se obtiene un nuevo valor para la variable menor solo si la e3 es inferior. Finalmente, se compara con la cuarta evaluación, es decir, si e4 es inferior al valor contenido en la variable menor, esta actualizará su valor con dicha variable.

Asúmanse las siguientes evaluaciones:

E1 = 17 E2 = 20 E3 = 15 E4 = 18

CÓDIGO DE COMPARACIÓN	PRUEBA DE VALOR
`int menor;`	`0`
`if (e1 < e2) menor = e1; else menor = e2;`	`(17<20) (V) → menor = 17`
`if (e3<menor) menor = e3;`	`(15<17) (V) → menor = 15`
`if (e4<menor) menor = e4;`	`(18<15) (F)`

Observar que la variable menor almacena el valor 15 el cual representa al menor valor registrado. Véase un segundo bloque de evaluaciones:

E1 = 20 E2 = 10 E3 = 18 E4 = 16

CÓDIGO DE COMPARACIÓN	PRUEBA DE VALOR
`int menor;`	`0`
`if (e1 < e2) menor = e1; else menor = e2;`	`(20<10) (F) → menor = 10`
`if (e3<menor) menor = e3;`	`(18<10) (F)`
`if (e4<menor) menor = e4;`	`(16<10) (F)`

Como se observa, la variable menor almacena el menor valor encontrado, en este caso, 10; pero las demás evaluaciones no fueron inferiores a 10, por lo tanto, el menor mantiene su valor entre las evaluaciones.

- **Mostrar un mensaje de error al dejar vacío un control de texto**

  Cuando el usuario selecciona un cuadro de texto (TextBox) y no llena valor alguno, la aplicación debe mostrarle un mensaje justamente en dicho control. Para esto, se debe agregar al formulario un control llamado Error Provider, el cual permite definir un mensaje de error. Para esto, se usará el evento Validated de los controles textuales.

Para seleccionar el evento **Validated**, primero seleccione el control desde el formulario. En propiedades, cambie a la vista de eventos  , seleccione **Validated** y haga doble clic para generar el código.

```
private void txtAlumno_Validated(object sender, EventArgs e)
        {
      //Código de validacion
       }
```

En el código de validación, verificar que la caja de texto no se encuentre vacía, mediante la condición txtAlumno.Text.Trim()==""". La función Trim tiene la misión de quitar los espacios en blanco de la cadena ingresada. Una vez verificado que se encuentra vacío, se usa el método SetError del control Error Provider de la siguiente manera:

```
epError.SetError(txtAlumno, "Ingrese nombre del alumno...!!!");
```

El primer parámetro del método SetError es definir a qué control se desea validar, el segundo define el mensaje que se muestra al usuario cuando el control de encuentre vacío. Si no se encuentra vacío el control, se aplica la sentencia epError.Clear(), que evita mostrar mensajes de error al usuario cuando este último registra un contenido en el control.

6. Finalmente, presione **F5** para probar la aplicación

Caso desarrollado 4 : if doble - venta de productos

Una feria de libros instalada en el centro de la ciudad requiere controlar las ventas de sus tres mejores libros para lo cual necesita implementar una aplicación que dado el nombre de un artículo, cantidad comprada y su forma de pago (Contado, Tarjeta) permita registrar la venta si se sabe que se paga al contado. Si se paga al contado, se le aplica un descuento del 5 % respecto al subtotal; si se paga con tarjeta, se le aplica un recargo del 10 % también respecto al subtotal, En cualquiera de los casos, se debe mostrar el descuento, recargo y el precio final.

Se tiene en cuenta lo siguiente:

- Validar el ingreso de los valores en los controles para mostrar mensajes personalizados según el error, por ejemplo, en caso de los cuadros combinados, validar que el usuario haya seleccionado algún elemento; mientras que en el cuadro de texto, valide que el valor registrado sea numérico.

- Los precios de los tres productos se muestran en la siguiente tabla:

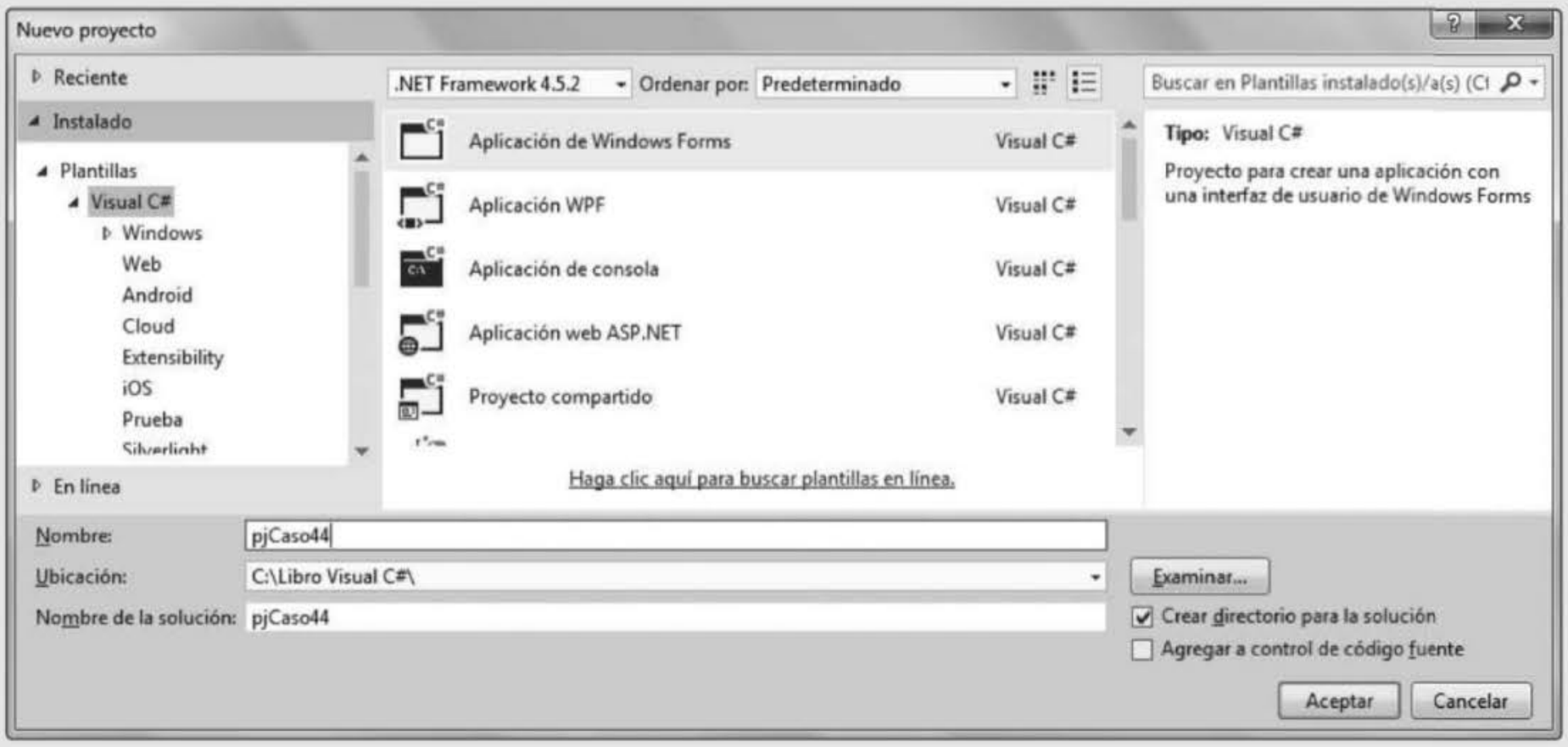
DESCRIPCIÓN DEL PRODUCTO	PRECIO UNITARIO
Colección Escolar	$ 240.00
Colección PreUniversitaria	$ 150.00
Colección Profesional	$ 350.00

- Mostrar el registro de la venta en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

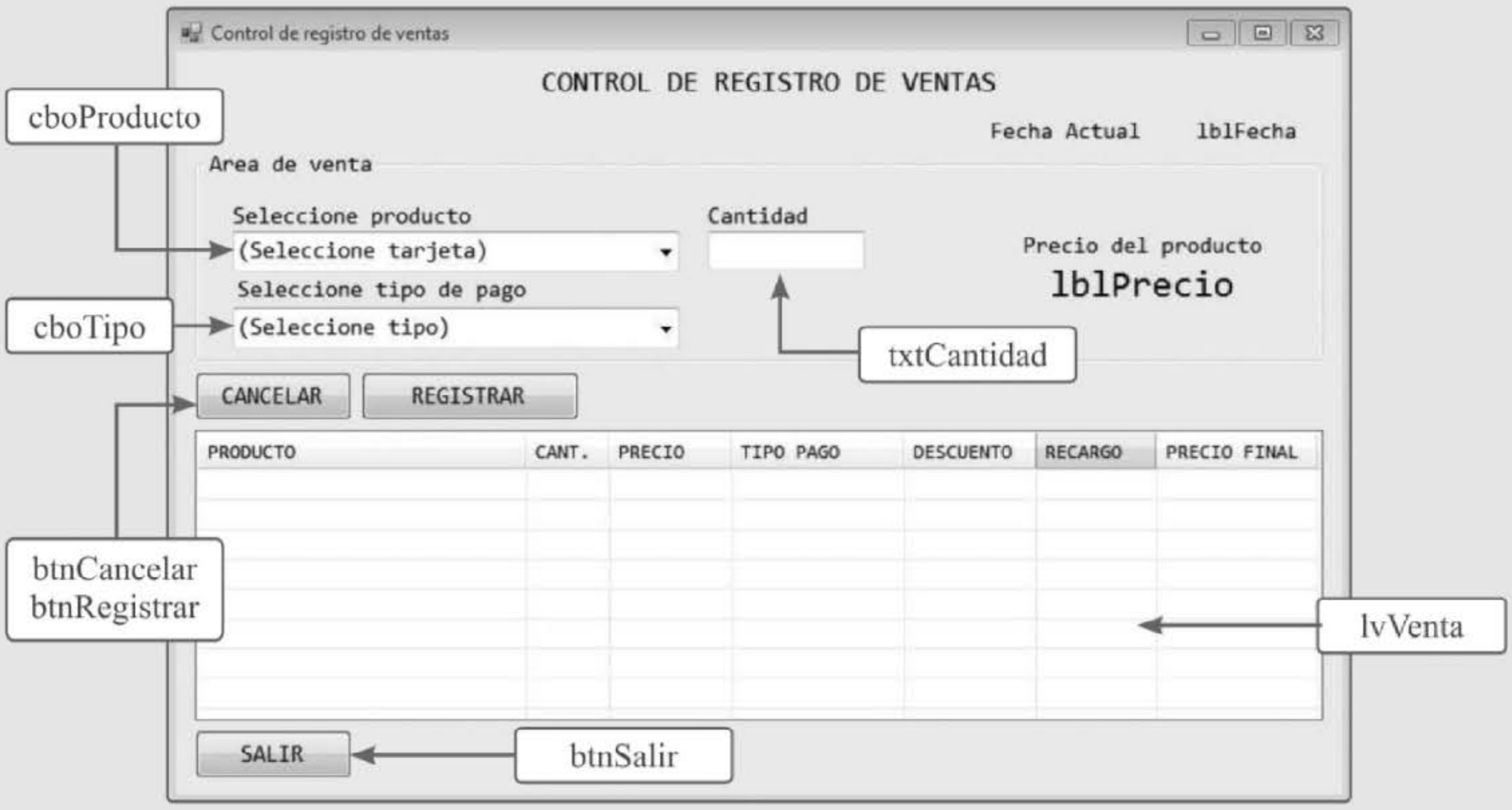
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso44**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

   - Seleccione el formulario **Form1.cs**, desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmVenta**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmVenta:

Clase	N.º elementos	Controles
Form	01	Form1
Label	08	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8
TextBox	01	TextBox1
GroupBox	01	GroupBox1
Button	03	Button1, Button2, Button3
ListView	01	ListView1
ComboBox	02	ComboBox1, ComboBox2
TOTAL	17	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmVenta Control de registro de ventas
Label1	Text	CONTROL DE REGISTRO DE VENTAS
Label2	Text	Fecha Actual
Label3	(Name)	lblFecha
Label4	Text	Seleccione producto
Label5	Text	Cantidad
Label6	Text	Seleccione tipo de pago

Label7	Text	Precio del producto
Label8	(Name)	lblPrecio
Textbox1	(Name)	txtCantidad
ComboBox1	(Name)	cboProducto
ComboBox2	(Name)	cboTipo
GroupBox1	Text	Area de venta
Button1	(Name) Text	bntRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvVenta True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.VisualBasic;
```

```
namespace pjCaso44
{
    public partial class frmVenta : Form
    {
        double precio = 0;

        public frmVenta()
        {
            InitializeComponent();
        }

        private void frmVenta_Load(object sender, EventArgs e)
        {
            lblFecha.Text = DateTime.Today.Date.ToString("d");
            lblPrecio.Text = (0).ToString("C");
        }

        private void cboProducto_SelectedIndexChanged(object sender, EventArgs e)
        {
            string producto = cboProducto.Text;

            if (producto.Equals("Colección Escolar")) precio = 250;
            if (producto.Equals("Colección PreUniversitaria")) precio = 150;
            if (producto.Equals("Colección Profesional")) precio = 350;

            lblPrecio.Text = precio.ToString("C");
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Validando
            if (cboProducto.SelectedIndex == -1)
                MessageBox.Show("Debe seleccionar un producto..!!!"); else
                if (!Information.IsNumeric(txtCantidad.Text))
                MessageBox.Show("Debe ingresar una cantidad..!!!");
            else
                if (cboTipo.SelectedIndex == -1)
                MessageBox.Show("Debe seleccionar un tipo..!!!");
            else
            {
                //Capturando datos
                string producto = cboProducto.Text;
                int cantidad = Convert.ToInt32(txtCantidad.Text);
                string tipo = cboTipo.Text;

                //Procesar calculos
                double subtotal = cantidad * precio;

                double descuento = 0, recargo = 0;
                if (tipo.Equals("Contado"))
```

```
            descuento = 0.05 * subtotal;
        else
            recargo = 0.1 * subtotal;

        double precioFinal = subtotal - descuento + recargo;

        //Impresion de resultados
        ListViewItem fila = new ListViewItem(producto);
        fila.SubItems.Add(cantidad.ToString());
        fila.SubItems.Add(precio.ToString("C"));
        fila.SubItems.Add(tipo);
        fila.SubItems.Add(descuento.ToString("C"));
        fila.SubItems.Add(recargo.ToString("C"));
        fila.SubItems.Add(precioFinal.ToString("C"));

        lvVenta.Items.Add(fila);
        btnCancelar_Click(sender,e);
    }
}

private void btnCancelar_Click(object sender, EventArgs e)
{
    cboProducto.Text = "(Seleccione producto)";
    cboTipo.Text = "(Seleccione tipo)";
    txtCantidad.Clear();
    lblPrecio.Text = (0).ToString("C");
    cboProducto.Focus();
}

private void btnSalir_Click(object sender, EventArgs e)
{
    DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                     "Venta",
                                MessageBoxButtons.YesNo,
                                    MessageBoxIcon.Exclamation);
    if (r == DialogResult.Yes) this.Close();
}
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Mantener el valor del precio de los productos**

  El precio obtenido de la selección del producto debe mantenerse, ya que otros procesos lo utilizan. Por ejemplo, al seleccionar un producto, el valor del precio es asignado al control lblPrecio; mientras que al presionar el botón Registrar, se necesita el precio del producto para calcular el subtotal. Para lograr todo lo mencionado, se debe declarar la variable precio de forma global.

```
double precio = 0;
```

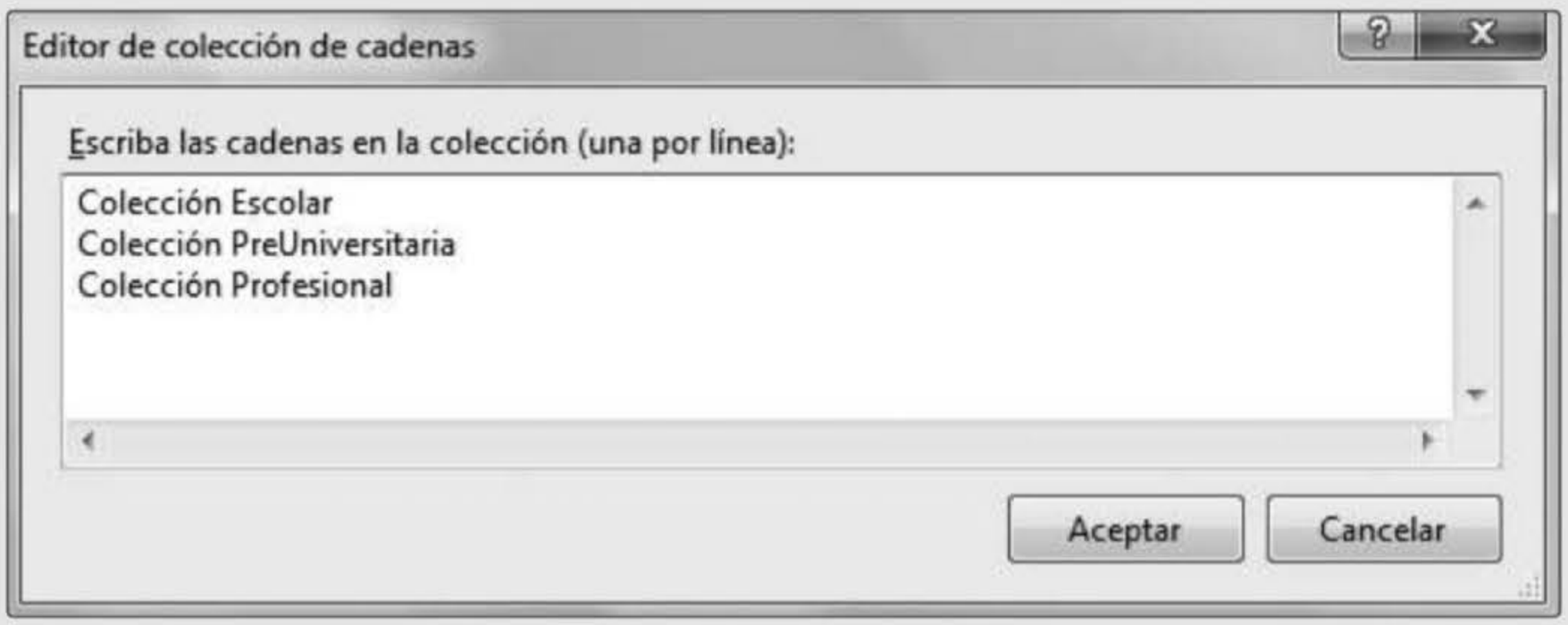
- **Mostrar la fecha actual y el precio inicial del producto**

  Al iniciar el formulario, se debe mostrar la información de la fecha actual y el valor $0.00. Para la fecha actual, se usa la sentencia DateTime.Today.Date.ToString("d"), Today representa la fecha y la hora al día actual, mientras que Date obtiene solo la fecha y el parametro "d" muestra la fecha corta del sistema.

  El precio del producto seleccionado inicialmente tiene el valor cero, con el formato $ 0.00 porque aún no selecciona un producto, por tanto, se debe formatear al número cero con la función ToString("C") donde el parámetro "C" representa el valor monetario.

- **Llenar el cuadro combinado de productos**

  El control ComboBox cuenta con una propiedad llamada Items, la cual muestra una ventana de ingreso de datos donde se registran los valores que se muestran en el cuadro combinado, tal como se muestra en la siguiente imagen:



  La otra posibilidad de agregar valores al control ComboBox colocando código fuente en el load del formulario, tal como sigue:

```
cboProducto.Items.Add("Colección Escolar");
cboProducto.Items.Add("Colección PreUniversitaria");
cboProducto.Items.Add("Colección Profesional");
```

➢ **Validar el ingreso de valores vacíos y no numéricos**

Cuando el usuario registra algún valor dentro de los controles, se debe tener seguridad de que estos sean los correctos, ya que dicha información se almacena en variables que declaran un tipo de datos que debe ser respetado. Por tanto, se empezará validando los cuadros combinados de la siguiente manera:

```
if (cboProducto.SelectedIndex == -1)
     MessageBox.Show("Debe seleccionar un producto..!!!");
```

La propiedad "SelectedIndex" determina la posición del elemento seleccionado en el cuadro combinado, cuando no se selecciona ningún elemento del cuadro, el valor es menos uno. Ahora, para validar un valor no numérico, se procede de la siguiente manera:

```
if (!Information.IsNumeric(txtCantidad.Text))
     MessageBox.Show("Debe ingresar una cantidad..!!!");
```

Se debe usar una función que determine si el valor ingresado es o no numérico, para lo cual se usa la sentencia Information.IsNumeric. Dicha función no es reconocida directamente por el compilador. Para esto, se necesita agregar una referencia al proyecto llamada "Microsoft.VisualBasic" y, además, colocar en las librerías del formulario "Using Microsoft.VisualBasic". La función IsNumeric determina si el número es numérico, pero en este caso se necesita saber si es NO numérico. Por lo tanto, se niega la expresión con el operador signo de admiración final (!).

Para agregar una referencia al proyecto, haga clic derecho en el proyecto > seleccione **Agregar > Referencia …** , y active el check **Microsoft VisualBasic**.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 5 : if doblemente enlazado - control de registro de multas de transito

La policía de tránsito del Perú es una institución conocida como policía de transporte o policía de tráfico, considerada la encargada de regular el orden y hacer cumplir las normas de tránsito o transitales establecidas para los distintos medios de transporte, ya sean trenes, buses o automóviles particulares. Se necesita implementar una aplicación que permita controlar las multas aplicadas a los vehículos que exceden el límite de velocidad permitido.

Se tiene en cuenta:

- La aplicación debe mostrar automáticamente la fecha y hora del registro de la multa.
- La tasa de multas por velocidad asignada se muestra en la siguiente tabla:

Velocidad (km/h)	Multa
Hasta 70	$ 0.00
Hasta 90	$ 120.00
Hasta 100	$ 240.00
Mayor a 100	$ 350.00

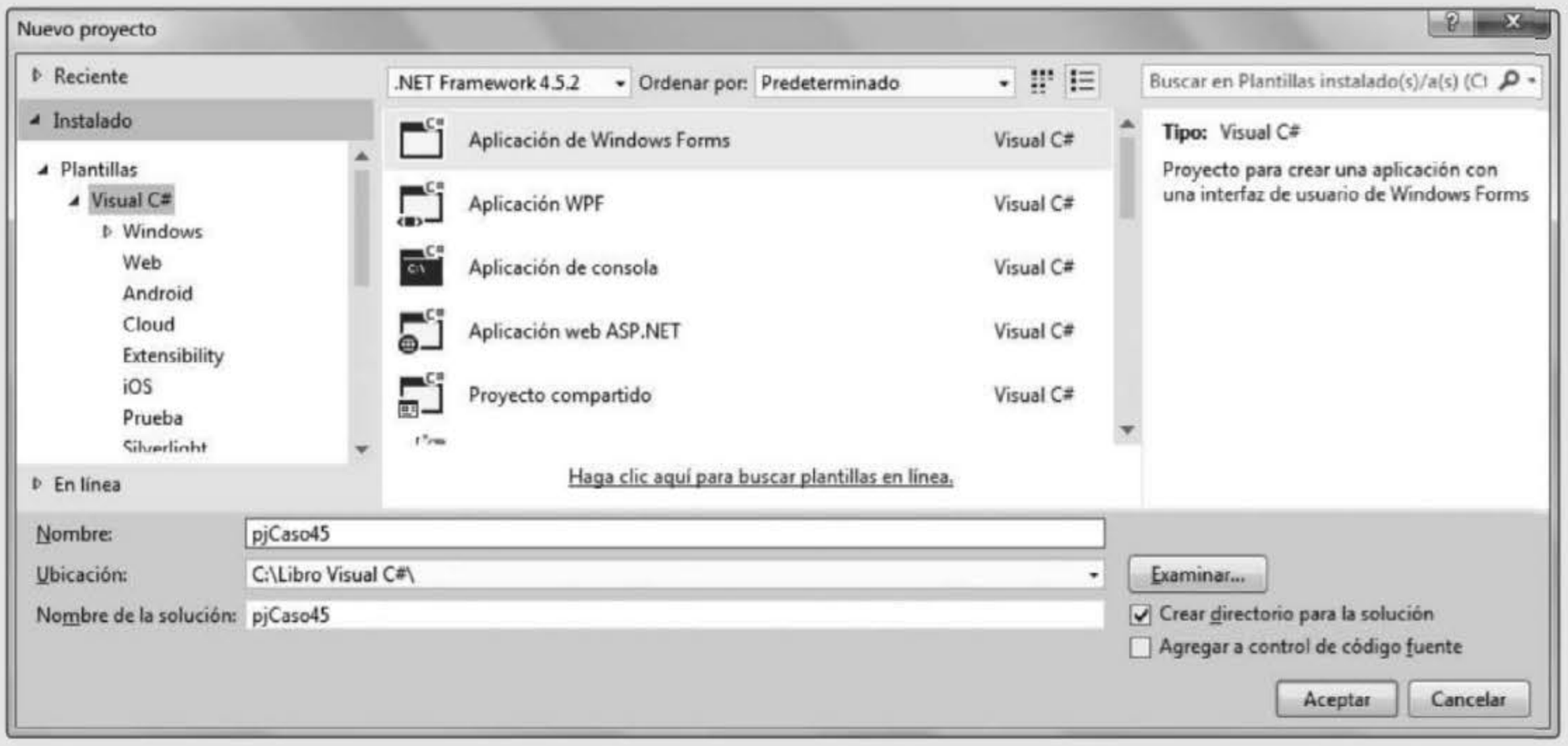
- Implementar el botón Eliminar que permite eliminar el registro de una multa registrada y mostrada en el control ListView; para eliminar, primero se debe seleccionar la multa desde el control ListView luego presionar el botón Eliminar.
- Mostrar los datos del registro de las multas como el número de placa, fecha, hora, velocidad y multa asignada en un control ListView.

Solución:

1. Crear nuevo proyecto en Visual Studio 2015:

- Seleccione **Archivo > Nuevo > Proyecto**
- Seleccione la plantilla **Visual C#**
- Seleccione **Aplicación de Windows Form**

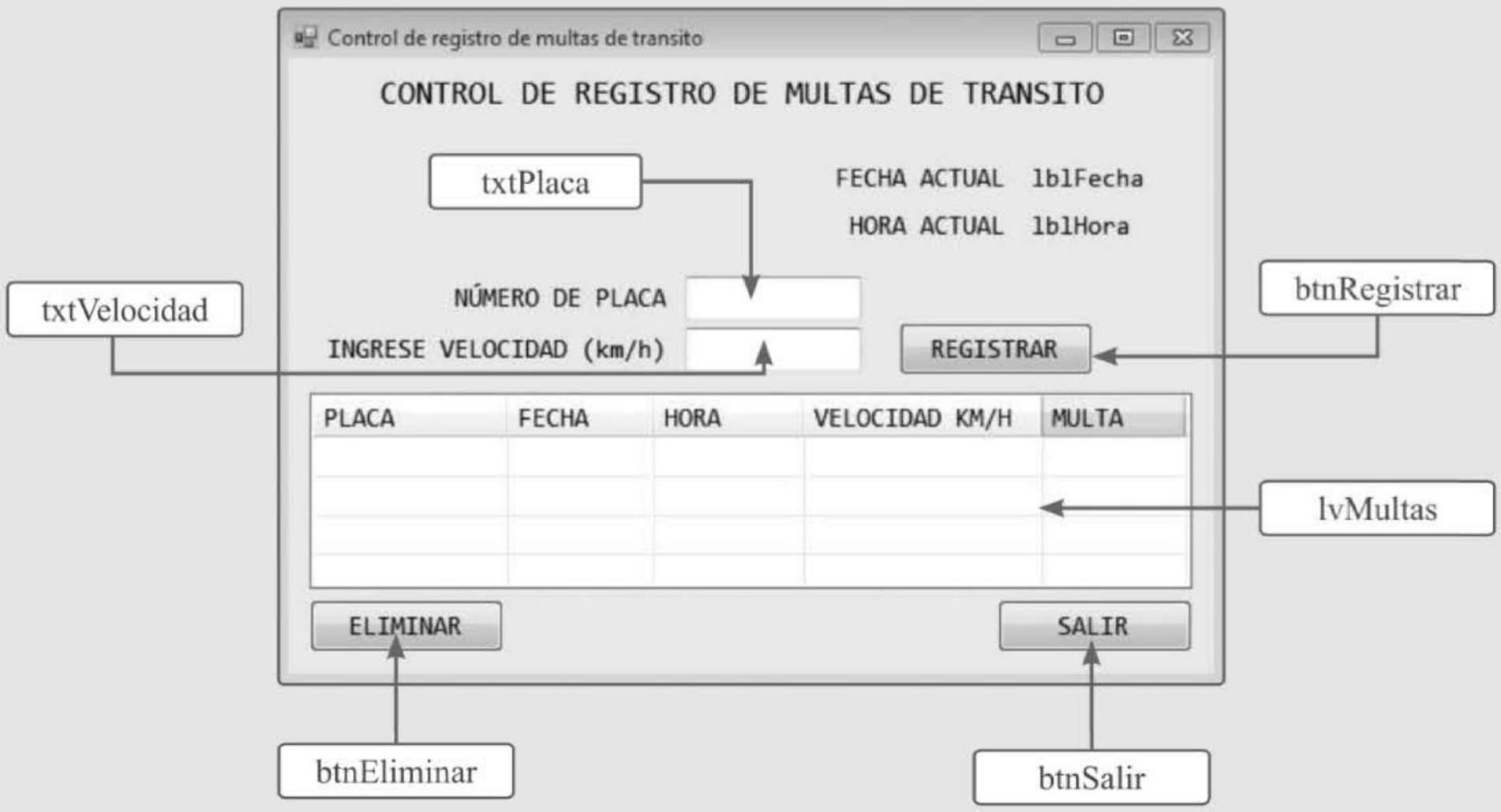
- Asigne el nombre **pjCaso45**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la propiedad **Nombre de archivo**, asigne el nombre **frmMultas**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmMultas:

Clase	N.º elementos	Controles
Form	01	Form1
Label	07	Label1, Label2, Label3, Label4, Label5, Label6, Label7
TextBox	02	Textbox1, Textbox2
Button	03	Button1, Button2, Button3
ListView	01	ListView1
TOTAL	14	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmMultas Control de registro de multas de transito
Label1	Text	CONTROL DE REGISTRO DE MULTAS DE TRANSITO
Label2	Text	FECHA ACTUAL
Label3	(Name)	lblFecha
Label4	Text	HORA ACTUAL
Label5	(Name)	lblHora
Label6	Text	NUMERO DE PLACA
Label7	Text	INGRESE VELOCIDAD (km/h)
Textbox1	(Name)	txtPlaca
Textbox2	(Name)	txtVelocidad
Button1	(Name) Text	bntRegistrar REGISTRAR
Button2	(Name) Text	btnEliminar ELIMINAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvMultas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso45
{
    public partial class frmMultas : Form
    {
        ListViewItem item;
        public frmMultas()
        {
            InitializeComponent();
        }

        private void frmMultas_Load(object sender, EventArgs e)
        {
            lblFecha.Text = DateTime.Today.Date.ToShortDateString();
            lblHora.Text = DateTime.Now.ToShortTimeString();
```

```
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los datos
            string placa = txtPlaca.Text;
            double velocidad = double.Parse(txtVelocidad.Text);
            DateTime fecha = DateTime.Parse(lblFecha.Text);
            DateTime hora = DateTime.Parse(lblHora.Text);
            //Procesando
            double multa=0;
            if (velocidad <= 70)
                multa = 0;
            else if (velocidad > 70 && velocidad <= 90)
                multa = 120;
            else if (velocidad > 90 && velocidad <= 100)
                multa = 240;
            else if (velocidad > 100)
                multa = 350;

            //Imprimiendo los resultados
            ListViewItem fila = new ListViewItem(placa);
            fila.SubItems.Add(lblFecha.Text);
            fila.SubItems.Add(lblHora.Text);
            fila.SubItems.Add(velocidad.ToString("0.00"));
            fila.SubItems.Add(multa.ToString("C"));
            lvMultas.Items.Add(fila);
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            if (item != null)
            {
                lvMultas.Items.Remove(item);
                MessageBox.Show("Multa eliminada correctamente...!!!");
            }else
            {
               MessageBox.Show("Debe seleccionar una multa de la lista");
            }
        }

        private void lvMultas_MouseClick(object sender, MouseEventArgs e)
        {
            item = lvMultas.GetItemAt(e.X, e.Y);
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Mostrar la fecha y la hora al iniciar el formulario**

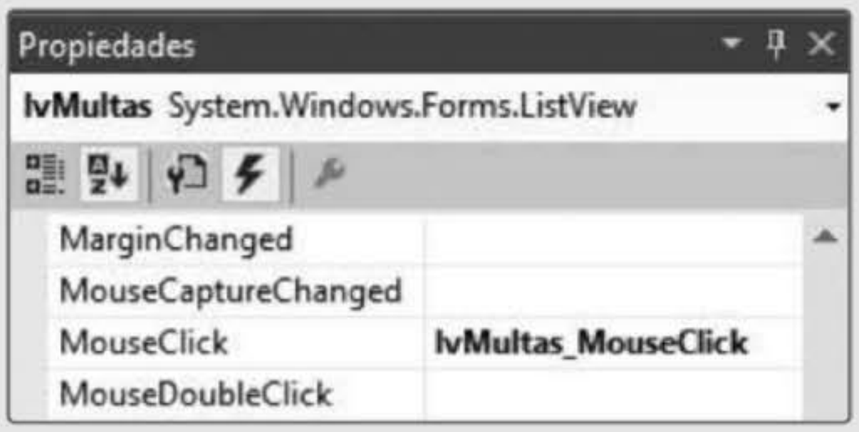
  La fecha y la hora del registro deben mostrarse al iniciar el formulario, es así que para mostrar la fecha se usa una sentencia de la clase DateTime "DateTime.Today.Date.ToShortDateString();" la cual muestra la fecha actual con el formato corto, mientras que "DateTime.Now.ToShortTimeString();" muestra la hora actual con formato corto.

```
lblFecha.text = DateTime.Today.Date.ToShortDateString();
lblHora.text = DateTime.Now.ToShortTimeString();
```

- **Eliminar una fila de un ListView**

  El botón eliminar se encarga de eliminar un registro de multa, de forma que primero se debe seleccionar una fila y capturar la posición del elemento seleccionado, para lograrlo se deben realizar los siguientes pasos:

  - Seleccione el control **ListView**.
  - En la ventana de propiedades cambie a vista **Eventos** y seleccione **MouseClick**.



  - Al hacer doble clic en dicho evento, se muestra el siguiente código:

```
private void lvMultas_MouseClick(object sender, MouseEventArgs e)
            {
                //Codigo procesado cuando se presione clic en el ListView
}
```

  - Luego, se debe agregar el código: item = lvMultas.GetItemAt(e.X, e.Y); donde eX representa la posición en X del elemento seleccionado, eY representa la posición en Y del elemento seleccionado. El método GetItemAt captura las posiciones obtenidas en e.X y e.Y. Como la variable Item guarda la posición del elemento seleccionado, es desde aquí que se debe eliminar el elemento, usando el botón **Eliminar**; para esto, se debe declarar dicha variable como global ListViewItem item;.

- Finalmente, se aplica el método de eliminación de fila con el siguiente código:

```
if (item != null)
{
   lvMultas.Items.Remove(item);
   MessageBox.Show("Multa eliminada correctamente...!!!");
} else{
   MessageBox.Show("Debe seleccionar una multa de la lista");
}
```

Se verifica que la variable item contiene la información del elemento seleccionado. Si contiene la posición, es decir, si es diferente de nulo (item != null); entonces, elimina el elemento mediante el método Remove del control ListView. Si, en caso contrario, el item contiene un valor nulo (item = null), entonces, se muestra un mensaje que indica al usuario que "debe seleccionar una multa de la lista".

6. Finalmente, presione **F5** para probar la aplicación

Caso desarrollado 6 : if doblemente enlazado -registro de postulantes

Una universidad particular realiza 02 exámenes de admisión para sus diferentes carreras profesionales, por tal motivo, al área de sistemas de la universidad se le ha encomendado la tarea de implementar una aplicación que permita registrar el nombre del postulante, carrera profesional a la cual postula y el puntaje que obtuvo en el examen de admisión. A partir de este punto, se debe registrar la condición del postulante con el texto "Ingresa" o "No ingresa" dependiendo del puntaje obtenido y la carrera profesional seleccionada; además, debe registrar una observación en la cual muestre cuántos puntos le faltan al estudiante solo si no ingresa a la carrera solicitada; y muestra cero si alcanza el puntaje.

Se tiene en cuenta:

- La aplicación debe mostrar automáticamente la hora del registro del postulante como si fuera un reloj digital.
- Los puntajes mínimos para el ingreso a la carrera profesional se presentan a continuación:

Carrera profesional	Puntaje mínimo
Ingeniería de Sistemas	100
Ingeniería de Software	120
Ingeniería mecánica	140
Ingeniería Industrial	160

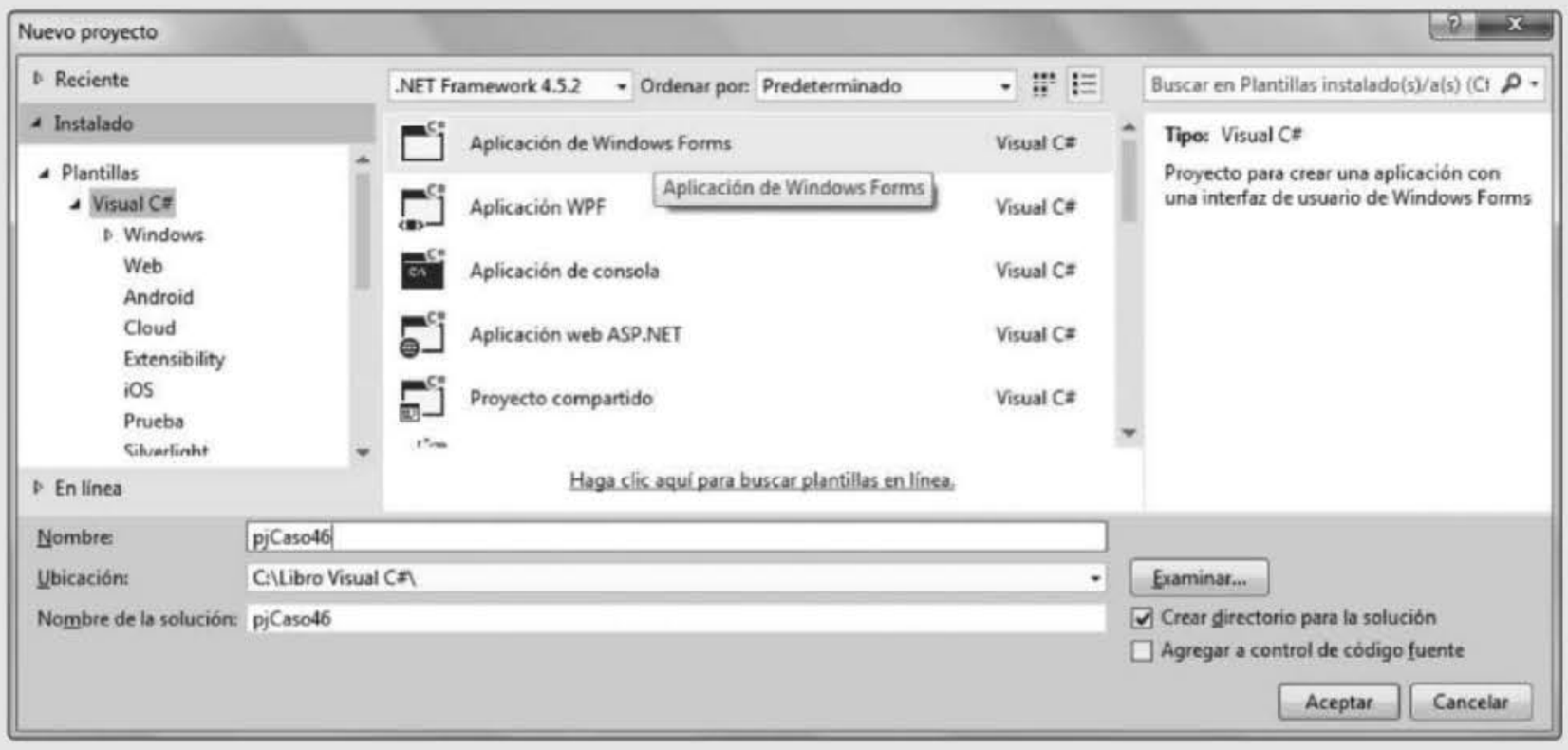
- Implementar los botones registrar postulante, cancelar registro y salir mediante la herramienta ToolStrip.

- Mostrar los datos del registro de los postulantes como el nombre del postulante, carrera profesional seleccionada, puntaje obtenido, condición y observación obtenido a partir del puntaje en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

- Seleccione **Archivo > Nuevo > Proyecto**.
- Seleccione la plantilla **Visual C#**.
- Seleccione **Aplicación de Windows Form**.
- Asigne el nombre **pjCaso46**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones.**
- Desde la propiedad **Nombre de archivo**, asigne el nombre **frmPostulantes**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmPostulantes:

Clase	N.º elementos	Controles
Form	01	Form1
Label	04	Label1, Label2, Label3, Label4
TextBox	02	Textbox1, Textbox2
ComboBox	01	ComboBox1
ToolStrip	01	ToolStrip1
ListView	01	ListView1
TOTAL	10	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmPostulantes Control de registro de postulantes
Label1	Text	CONTROL DE REGISTRO DE POSTULANTES
Label2	Text	POSTULANTE
Label3	(Name)	CARRERA PROFESIONAL
Label4	Text	PUNTAJE OBTENIDO
ComboBox1	(Name)	cboCarrera
Textbox1	(Name)	txtPostulante
Textbox2	(Name)	txtPuntaje
ToolStrip	(Name)	ToolStrip1
ListView1	(Name) GridLines	lvPostulantes True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso46
{
    public partial class frmPostulantes : Form
    {
        public frmPostulantes()
        {
            InitializeComponent();
            tHora.Enabled = true;
        }

        private void tsRegistrar_Click(object sender, EventArgs e)
        {
            //Capturar datos del formulario
            string postulante = txtPostulante.Text;
            string carrera = cboCarrera.Text;
            double puntaje = Double.Parse(txtPuntaje.Text);

            //Determinar
            string condicion="INGRESA";
            double puntos=0;
            if (carrera== "Ingeniería de Sistemas")
            {
                if (puntaje < 100)
                {
                    condicion = "NO INGRESA";
                    puntos = 100-puntaje;
                }
            }else if (carrera== "Ingeniería de Software")
            {
                if (puntaje < 120)
                {
                    condicion = "NO INGRESA";
                    puntos = 120 - puntaje;
                }
            } else if (carrera == "Ingeniería Mecánica")
            {
                if (puntaje < 140)
                {
                    condicion = "NO INGRESA";
                    puntos = 140 - puntaje;
                }
```

```
        } else if (carrera == "Ingeniería Industrial")
        {
            if (puntaje < 160)
            {
                condicion = "NO INGRESA";
                puntos = 160 - puntaje;
            }
        }

            //Mostrando resultados
            ListViewItem fila = new ListViewItem(postulante);
            fila.SubItems.Add(carrera);
            fila.SubItems.Add(puntaje.ToString("0.00"));
            fila.SubItems.Add(condicion);
            fila.SubItems.Add("Falta "+puntos.ToString("0.00"));
            lvPostulantes.Items.Add(fila);

            //Limpiando los controles
            tsCancelar_Click(sender, e);
        }

        private void tHora_Tick(object sender, EventArgs e)
        {
            this.Text = "Control de registro de postulantes - ";
            this.Text += DateTime.Now.ToString("hh:mm:ss");
        }

        private void tsCancelar_Click(object sender, EventArgs e)
        {
            txtPostulante.Clear();
            cboCarrera.SelectedIndex = -1;
            txtPuntaje.Clear();
            txtPostulante.Focus();
        }

        private void tsSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Postulantes",
                                             MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }
    }
}
```

➢ Mostrar la hora activa en el título del formulario

La idea principal es hacer que en la barra de título del formulario aparezca un texto y la hora actual como si fuera un reloj digital. Para esto, se deben realizar los siguientes pasos:

- En el cuadro de herramientas, busque **Timer**.
- Arrastre un control timer al formulario, en la parte inferior del formulario, aparece el nombre **Timer1**.
- Seleccione desde esta barra el control **Timer1** y cambie de nombre en las propiedades por **tHora**.
- Al hacer doble clic sobre el control **tHora**, aparece el siguiente código:

```
private void tHora_Tick(object sender, EventArgs e)
            {
                    //Código a implementar
}
```

- Coloque el siguiente código dentro del metodo Tick del timer: }

```
this.Text = "Control de registro de postulantes - ";
this.Text += DateTime.Now.ToString("hh:mm:ss");
```

- La primera sentencia define el primer texto a mostrar en el título del formulario, el segundo es la hora digital que acompaña al texto del título. Para esto, se debe concatenar el texto con el digital del tiempo, por tanto, se usa el operador += como si fuera un acumulador. Luego, para mostrar la hora actual, se usa DateTime.Now y se adicionan los parámetros de impresión para mostrar estrictamente la hora, minuto y segundo de la hora actual.
- Y finalmente, para que el reloj se muestre digitalizado, coloque el siguiente código **"tHora.Enabled=true;"** debajo de la llamada al "InitializeComponent", ubicado en el método constructor de la clase frmPostulantes.

➢ Agregar un barra ToolStrip al formulario

La barra ToolStrip permite ejecutar varios procesos de la aplicación desde un mismo control. Véase cómo implementar dicho control:

- Seleccione el control **ToolStrip** del cuadro de herramientas.
- Al agregar un control al formulario, automáticamente este se posiciona en la parte superior del formulario y tiene el siguiente aspecto:



- Se necesita agregar tres botones, eso se realiza haciendo clic sobre el mismo botón que se presenta en el **ToolStrip**



- Desde la ventana de propiedades, seleccione **Text** y agregue un texto, como aún no se muestra dicho texto, modifique la propiedad **Display Style** y asigne **ImageAndText**, finalmente también asigne un nombre (Name), por ejemplo tsRegistrar, tsCancelar y tsSalir.
- Para colocar el código dentro de dichos botones, haga doble clic sobre cada uno, como si se tratase de un Button.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 7 : switch - control de registro de llamadas

Un call center ubicado en el centro de la ciudad de Lima registra diariamente las llamadas que realizan los terminalistas a los clientes. Para esto, se necesita una aplicación que permita controlar el registro de llamadas haciendo que ingrese el número de teléfono al cual se realizado la llamada, el tipo de llamada y la cantidad total de minutos utilizados por el terminalista. La aplicación debe mostrar un reporte de importe, descuento y neto a pagar por cada registro de llamada.

Se tiene en cuenta lo siguiente:

- La aplicación debe solicitar el registro del número teléfono con un máximo de 15 caracteres, así mismo debe bloquear el ingreso de texto tanto en el número de teléfono como en la cantidad de minutos.
- La asignación de tarifa por minuto y descuentos se realizan mediante la siguiente tabla:

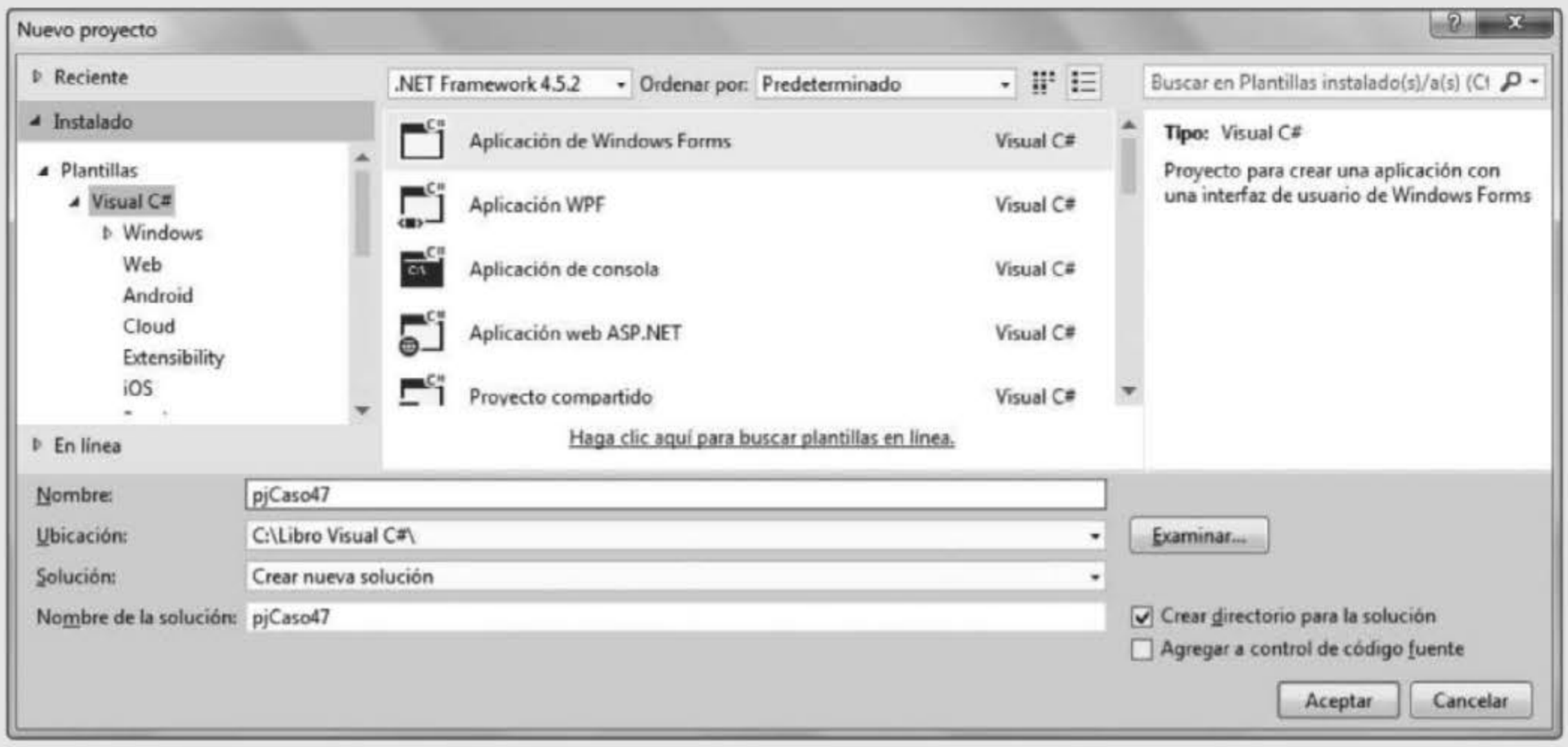
Tipo de llamada	Tarifa por minuto	Descuento
Fijo nacional	$ 0.25	5.0 %
Fijo internacional	$ 1.75	7.0 %
Movil nacional	$ 1.25	9.0 %
Movil internacional	$ 2.50	12.0 %
ToolStrip	01	ToolStrip1
ListView	01	ListView1
TOTAL	10	

- Se deben implementar los botones registrar llamada, cancelar registro y salir mediante la herramienta **ToolStrip** y ubicarlo en la parte inferior de formulario.
- Se deben mostrar los datos del registro de llamadas como el número de teléfono, tipo de llamada, minutos, importe, descuento y neto a pagar en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

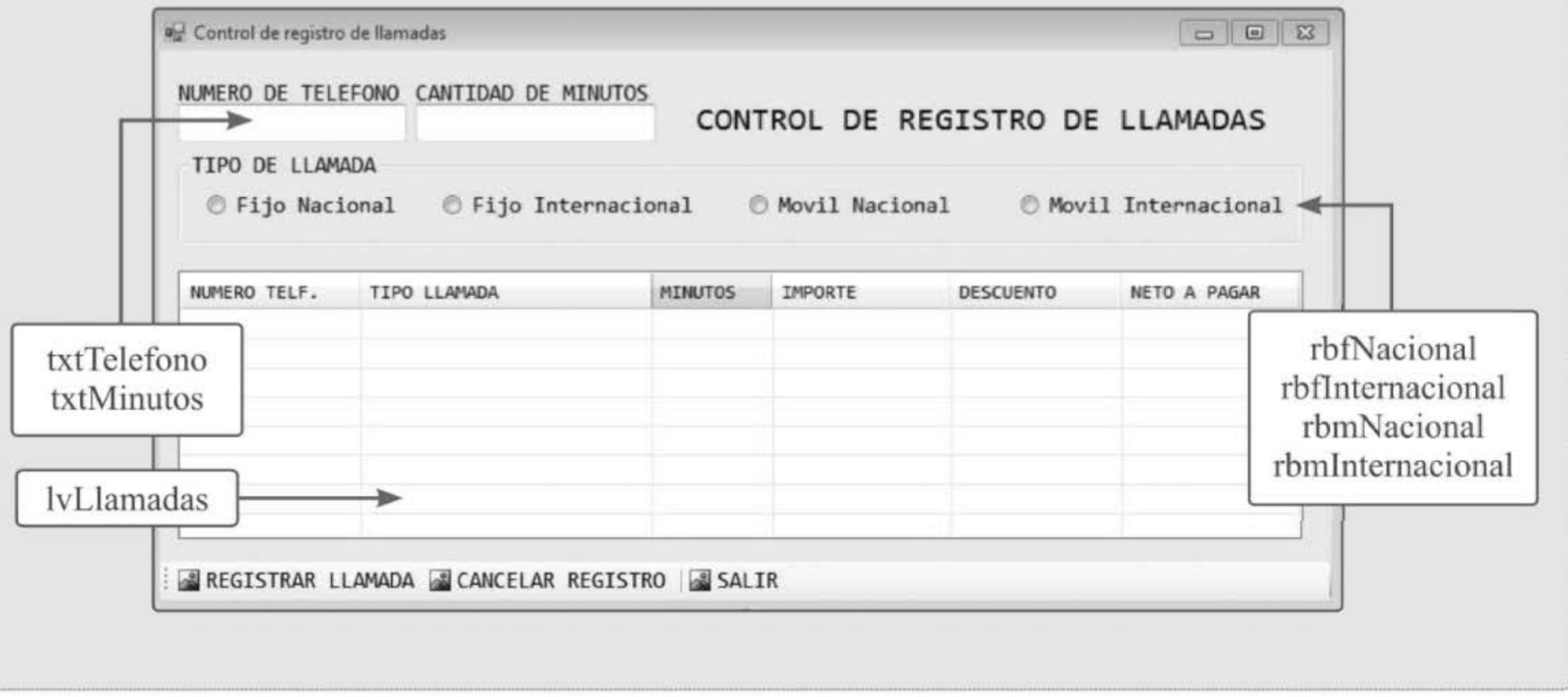
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Selección **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso47**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmLlamadas**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmLlamadas:

Clase	N.º elementos	Controles
Form	01	Form1
Label	03	Label1, Label2, Label3
TextBox	02	Textbox1, Textbox2
GroupBox	01	GroupBox1
RadioButton	04	RadioButton1, RadioButton2, RadioButton3, RadioButton4
ToolStrip	01	ToolStrip1
ListView	01	ListView1
TOTAL	13	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmLlamadas Control de registro de llamadas
Label1	Text	CONTROL DE REGISTRO DE LLAMADAS
Label2	Text	NUMERO DE TELEFONO
Label3	Text	CANTIDAD DE MINUTOS
GroupBox1	Text	TIPO DE LLAMADA
RadioButton1	(Name) Text	rbfNacional Fijo Nacional
RadioButton2	(Name) Text	rbfInternacional Fijo Internacional
RadioButton3	(Name) Text	rbmNacional Movil Nacional
RadioButton4	(Name) Text	rbmInternacional Movil Internacional
Textbox1	(Name) MaxLenght	txtTelefono 15
Textbox2	(Name)	txtMinutos
ToolStrip	(Name) Dock	ToolStrip1 Bottom
ListView1	(Name) GridLines	lvLlamadas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso47
{
    public partial class frmLlamadas : Form
    {
        public frmLlamadas()
        {
            InitializeComponent();
        }

        private void tsRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los datos
            string telefono = txtNumero.Text;
            int minutos = int.Parse(txtMinutos.Text);

            //Determinando el tipo de llamada
            string tipo="";
            if (rbfNacional.Checked == true) tipo = "Fijo Nacional";
```

```
if (rbfInternacional.Checked == true) tipo = "Fijo Internacional";
if (rbmNacional.Checked == true) tipo = "Movil Nacional";
if (rbmInternacional.Checked == true) tipo = "Movil Internacional";

//Validando si NO selecciona un tipo
if (tipo == "")
{
MessageBox.Show("Debe seleccionar el tipo de llamada..!!",
                    "Llamadas");
    return; //Expulsa del metodo
}


//Asignando una tarifa segun el tipo de llamada
double tarifa=0;
switch (tipo)
{
    case "Fijo Nacional": tarifa = 0.25; break;
    case "Fijo Internacional": tarifa = 1.75; break;
    case "Movil Nacional": tarifa = 1.25; break;
    case "Movil Internacional": tarifa = 2.50; break;
}

//Realizando los calculos de los importes
double importe = tarifa * minutos;

//Determinando el descuento segun los minutos
double descuento;
switch (tipo)
{
    case "Fijo Nacional":
                descuento = 5.0/100 * importe; break;
    case "Fijo Internacional":
                descuento = 7.0 / 100 * importe; break;
    case "Movil Nacional":
                descuento = 9.0 / 100 * importe; break;
    case "Movil Internacional":
                descuento = 12.0 / 100 * importe; break;
    default:
                descuento = 0; break;
}

//Calculando el neto
double neto = importe - descuento;

//Realizando impresiones
ListViewItem fila = new ListViewItem(telefono);
fila.SubItems.Add(tipo);
fila.SubItems.Add(minutos.ToString());
fila.SubItems.Add(importe.ToString("C"));
fila.SubItems.Add(descuento.ToString("C"));
```

```
            fila.SubItems.Add(neto.ToString("C"));
            lvLlamadas.Items.Add(fila);

            //Limpiando los controles
            tsCancelar_Click(sender,e);
        }

        private void tsCancelar_Click(object sender, EventArgs e)
        {
            txtNumero.Clear();
            txtMinutos.Clear();
            rbfNacional.Checked = true;
            txtNumero.Focus();
        }

        private void tsSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Llamadas",
                                        MessageBoxButtons.YesNo,
                                        MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }

        private void txtMinutos_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!(char.IsNumber(e.KeyChar)) && (e.KeyChar != (char)Keys.Back))
            {
                MessageBox.Show("Debe ingresar solo numeros","Llamadas");
                e.Handled = true;
                return;
            }
        }

        private void txtNumero_KeyPress(object sender, KeyPressEventArgs e)
        {
            if (!(char.IsNumber(e.KeyChar)) && (e.KeyChar != (char)Keys.Back))
            {
                MessageBox.Show("Debe ingresar solo numeros", "Llamadas");
                e.Handled = true;
                return;
            }
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Determinar si un control RadioButton se encuentra seleccionado**

  Se debe tener en cuenta que los controles RadioButton trabajan siempre en grupo, por eso, siempre que se agreguen dichos controles al formulario, debe hacerse con un contenedor como el Groupbox. Si es así, se debe determinar qué botón ha sido seleccionado, para lograrlo se usa la propiedad "Checked", la cual evalúa si el control se encuentra marcado por el usuario y gracias a la sentencia If puede evaluarse. Considérese que true determina el activo y false el inactivo, tal como se muestra en el siguiente código:

```
if (rbfNacional.Checked == true) tipo = "Fijo Nacional";
```

  La condición evalúa si el control rbfNacional se encuentra activo, si es así, asigna a la variable tipo el valor "Fijo Nacional".

- **Bloquear el ingreso de caracteres no numéricos en el cuadro de texto**

  Los controles de texto permiten agregar información de todo tipo, pero se pueden controlar mediante una serie de sentencias y así bloquear el texto ingresado y solo permitir el acceso de números como se muestra en el siguiente código:

```
if (!(char.IsNumber(e.KeyChar)) && (e.KeyChar != (char)Keys.Back))
{
     MessageBox.Show("Debe ingresar solo numeros","Llamadas");
     e.Handled = true;
     return;
}
```

  Esto se encarga de bloquear todo tipo de caracteres de tipo cadena, pero se debe agregar dicho código en un evento llamado "KeyPress" (presión de tecla) del control cuadro de texto para visualizar que dicho evento realice los siguientes pasos:

  - Seleccionar el control cuadro de texto
  - En la ventana de propiedades, seleccione **Eventos**.
  - Busque y seleccione el evento **KeyPress** y haga doble clic para visualizar su código, tal como se muestra en el siguiente código:

```
private void txtNumero_KeyPress(object sender, KeyPressEventArgs e)
{
     //Asignar codigo para el evento keypress
 }
```

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 8 : switch - control de registro de estacionamiento

Una empresa de estacionamiento necesita una aplicación que permita controlar el registro de los vehículos que diariamente ingresan al estacionamiento. Los datos que necesita registrar el personal de control son el número de placa del auto, la hora de inicio y la hora de salida del vehículo.

Se tiene en cuenta lo siguiente:

- La tarifa por hora que aplica la empresa se da según el día de la semana. En la siguiente tabla se presentan las tasas que rigen:

Días	Costo por hora
Domingo	$ 2.00
Lunes a jueves	$ 4.00
Viernes y sábado	$ 7.00

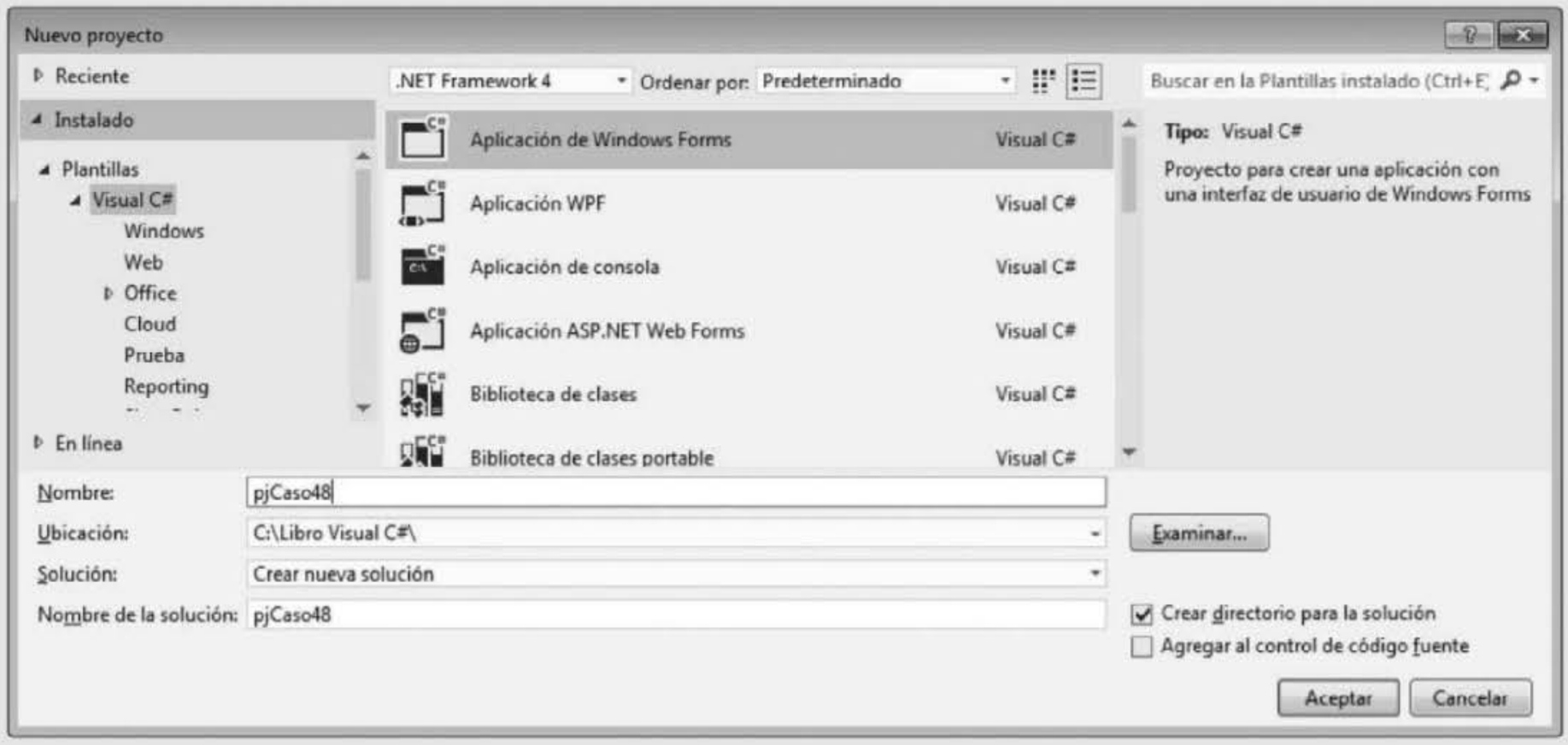
- Al iniciar la aplicación, se debe mostrar la fecha actual, así como el costo por hora según el día.
- La cantidad de horas que el vehículo se encuentra en el estacionamiento resulta de la diferencia entre la hora de salida y la de inicio.
- El importe resulta del producto de la cantidad de hora de estacionamiento y el costo por día.
- El botón registrar muestra el número de placa, fecha, hora de inicio, hora final, cantidad de horas de estacionamiento, la tarifa según el criterio anterior y el importe a pagar por el cliente en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso48**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la propiedad **Nombre de archivo**, asigne el nombre **frmEstacionamiento**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmEstacionamiento:

Clase	N.º elementos	Controles
Form	01	Form1
Label	03	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8
TextBox	02	Textbox1
GroupBox	01	GroupBox1
MaskedTextBox	02	MaskedTextBox1, MaskedTextBox2
Button	03	Button1, Button2, Button3
ListView	01	ListView1
TOTAL	13	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmEstacionamiento Control de registro de estacionamiento
Label1	Text	CONTROL DE REGISTRO DE ESTACIONAMIENTO
Label2	Text	PLACA
Label3	Text	COSTO POR DÍA $
Label4	(Name) Text	lblCosto lblCosto
Label5	Text	FECHA
Label6	(Name) Text	lblFecha lblFecha
Label7	Text	HORA INICIO
Label8	Text	HORA SALIDA
GroupBox1	Text	FECHA Y HORA
TextBox1	(Name)	txtPlaca
MaskedTextBox1	(Name) Mask	txtHoraInicio Time (US)
MaskedTextBox1	(Name) Mask	txtHoraFin Time (US)
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnCancelar CANCELAR
Button3	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvRegistro True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace pjCaso48
{
    public partial class frmEstacionamiento : Form
    {
        string dia;

        public frmEstacionamiento()
        {
            InitializeComponent();
        }

        private void frmEstacionamiento_Load(object sender, EventArgs e)
        {
            //Mostrando la fecha actual
            lblFecha.Text = DateTime.Now.ToShortDateString();

            //Determinar el dia
```

```
            DateTime fecha = DateTime.Parse(lblFecha.Text);
            dia = fecha.ToString("dddd");

            double costo=0;
            switch(dia){
                case "Sunday": costo = 2; break;
                case "Monday":
                case "Tuesday":
                case "Wednesday":
                case "Thursday":
                    costo = 4; break;
                case "Friday":
                case "Saturday":
                    costo = 7; break;
            }

            lblCosto.Text = costo.ToString("0.00");
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los datos del formulario
            string placa = txtPlaca.Text;
            double costo =  double.Parse(lblCosto.Text);
            DateTime fecha = DateTime.Parse(lblFecha.Text);
            DateTime horaInicio = DateTime.Parse(txtHoraInicio.Text);
            DateTime horaFin = DateTime.Parse(txtHorafin.Text);

            //Calcular la hora
            TimeSpan hora = horaFin - horaInicio;

            //Calcular el importe
            double importe = costo * (hora.TotalHours);

            ListViewItem fila = new ListViewItem(placa);
            fila.SubItems.Add(fecha.ToString("d"));
            fila.SubItems.Add(horaInicio.ToString("t"));
            fila.SubItems.Add(horaFin.ToString("t"));
            fila.SubItems.Add(hora.TotalHours.ToString());
            fila.SubItems.Add(costo.ToString("C"));
            fila.SubItems.Add(importe.ToString("C"));
            lvRegistro.Items.Add(fila);
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            txtPlaca.Clear();
            txtHoraInicio.Clear();
            txtHorafin.Clear();
            txtPlaca.Focus();
        }
```

```
        private void btnSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Estacionamiento",
                                             MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Determinar el día actual**

  Primero se debe capturar la fecha actual para determinar en el día de la semana actual, luego separar el día de la fecha, lo que se realiza con el siguiente código:

```
DateTime fecha = DateTime.Parse(lblFecha.Text);
dia = fecha.ToString("dddd");
```

Cuando se referencia a un día de la fecha de usa "dddd" que representa al nombre del día de la semana. Luego, en base al día obtenido, se usa la estructura switch para obtener el costo por día como se muestra en el siguiente código:

```
double costo=0;
switch(dia){
     case "Sunday": costo = 2; break;
     case "Monday":
      case "Tuesday":
     case "Wednesday":
     case "Thursday":
            costo = 4; break;
     case "Friday":
     case "Saturday":
            costo = 7; break;
 }
```

- **Determinar las horas de estacionamiento**

  Cuando se captura el valor de las horas, se debe asegurar que estos sean de tipo DateTime, ya que para calcular la cantidad de horas de diferencia ambas deben ser de dicho tipo. Luego, se usa la siguiente sentencia:

```
TimeSpan hora = horaFin - horaInicio;
```

TimeSpan es el tipo para intervalor de horas, mediante el cual la resta resulta en formato hora. Por ejemplo, si la cantidad de horas es 4 TimeSpan, devuelve 04:00:00; con esto se entiende que dicho valor no puede ser operado aún, porque se trata de una especificación del tiempo. Lo que se debe hacer es obtener el total de hora con la siguiente expresión "hora.TotalHours" que resulta beneficioso por la precisión que cuenta dicha propiedad.

6. Finalmente, presione **F5** para probar la aplicación.

Para recordar

- Los operadores de relación como mayor (>), menor (<) o igual (==) permiten comparar una expresión binaria. Es decir, si entre dos elementos hay un tercer elemento de comparación, se usa un operador lógico and (&&) u or (||), el cual permitirá unir dos o más expresiones condicionales.
- La sentencia condicional if simple permite ejecutar un bloque de sentencias, solo si la condición evaluada es verdadera, caso contrario no ejecuta acción alguna y continúa con la siguiente instrucción.
- La sentencia condicional if doble, permite ejecutar sentencias dependiendo del resultado de la condicion, si es verdadero ejecuta sentencias del lado verdadero si la condicional resulta falsa ejecutar sentencias especificadas en el lado falso.
- La sentencia condicional if doblemente enlazada permite ejecutar sentencias según la respuesta de la condición. La forma de ejecutar las sentencias se debe a una selección tipo múltiple, lo que significa que cuando la condición resulta verdadera, ejecuta las sentencias especificadas y allí mismo finaliza la sentencia, es decir, ya no evalúa las demás condiciones.
- La forma de trabajo de la sentencia condicional swtich es similar al if doblemente enlazado. Aunque los casos en una instrucción switch solo pueden ser números, caracteres y cadenas de caracteres.

Preguntas frecuentes

➤ **¿Qué se debe considerar al implementar una condición?**

Primero, se deben entender correctamente las reglas de negocio que propone el caso a desarrollar; si estas no son claras, probablemente la aplicación no devuelva los valores adecuados.

Luego, se debe conocer la forma de uso de los operadores lógicos y relacionales, ya que a partir de ellos se pueden implementar condiciones correctas en la aplicación. Un error común al implementar una condición se da justamente en el mal uso de estos operadores.

➤ **¿Qué tipos de operadores se deben usar en una expresión condicional?**

Existen dos tipos de operadores que se pueden considerar sumamente importantes para la implementación de las condiciones. Se tiene:

Operador	Definición
>	Mayor que
>=	Mayor o Igual que
<	Menor que
<=	Menor o Igual que
==	Igual
!=	Diferente o no es igual
?	Condicional

Operador	Definición
&&	Representa la Y lógica, la cual evalúa que ambas condiciones sean de tipo true para realizar una determinada acción.
||	Representa la O lógica, la cual ejecuta una determinada acción solo si una de las condiciones es de tipo true.
!	Representa la negación del resultado de una condición.

Capítulo 5

Estructuras de repetición

Capacidad terminal:

Desarrollar aplicaciones que involucren estructuras de repetición como for, while y la integración de ellos en una misma aplicación.

Contenido:

- Introducción
- Los contadores
- Operaciones sobre los contadores
- Los acumuladores
- Operaciones sobre los acumuladores
- Bucle While
- Bucle do while
- Bucle for
- Casos desarrollados
  - o Caso desarrollado 1: contadores - control de registro de participantes
  - o Caso desarrollado 2: acumuladores - planilla de sueldos
  - o Caso desarrollado 3: for - seguros de vida
  - o Caso desarrollado 4: while - control de facturas
  - o Caso desarrollado 5: do while - control de registro de alquiler de habitaciones
  - o Caso desarrollado 6: integración - control de venta de boletos

- Para recordar
- Preguntas frecuentes

Introducción

Una estructura de repetición, también llamada lazo o bucle, permite generar una serie de repeticiones de un bloque de sentencias específicas. Dicho ciclo de repeticiones debe terminar en algún momento, tal es así que, cuando se ejecuta una estructura de repeticiones, el procesador de la computadora se preocupa de prestar atención a dicho servicios.

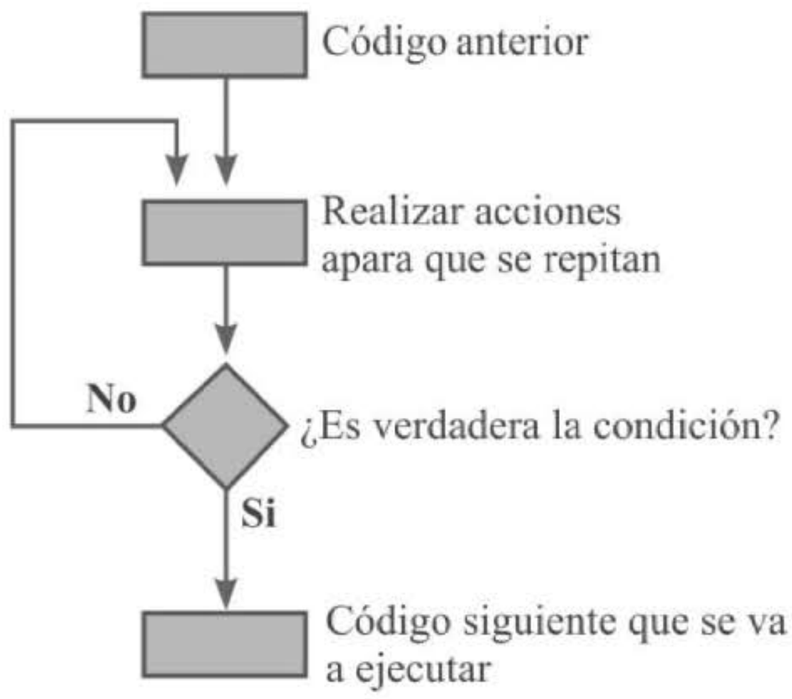
De este modo, si el ciclo de repeticiones no tiene una terminación adecuada, el procesador termina por detenerlo. Para un buen uso de la estructura de repetición, se debe tener bien claro el inicio y fin del ciclo o bucle.

Ahora, ¿cuándo es necesario usar dicha estructura? La respuesta se puede obtener de acuerdo a las siguientes situaciones:

- Cuando se necesita almacenar muchos valores con una sola sentencia de programación.
- Cuando se necesita realizar sumas sucesivas.
- Cuando se necesita realizar conteos sucesivos.
- Cuando se necesita generar números aleatorios.

Al implementar una estructura de repetición, se debe tener en cuenta que las repeticiones o ciclos no se dan automáticamente, sino más bien se condicionan los ciclos, es decir, se repiten las sentencias las veces que la condición lo permita.

Véase la siguiente imagen:



El código anterior representa cualquier sentencia implementada en la aplicación. Las acciones a realizar son especificadas dentro de la implementación de la estructura de repetición y lo más importante es definir correctamente la condición, pues gracias a esta, el bucle procesa correctamente.

Para nuestro caso, se usan las estructuras repetitivas for (para), while (mientras), do while (hacer mientras). Primero, se estudiará la forma de trabajo de los contadores y acumuladores y la importancia que tienen ellos en la implementación de las estructuras repetitivas.

5.1 Los contadores

Un contador es una variable de tipo entero cuyo valor incrementa o decrementa en un valor constante, cuando ocurre algún suceso; es decir, el uso del contador se debe a una condición en la aplicación.

Para que los contadores trabajen de manera eficaz en una aplicación, se deben integrar a estructuras condicionales como If, Switch y estructuras repetitivas como For o While.

Asimismo, el uso de los contadores se puede dar de dos formas:

- **Fuera de una estructura repetitiva:** Si se quiere usar un contador sin estructuras repetitivas, dicha variable debe ser declarada de forma global y así actualizar su valor durante las llamadas.
- **Dentro de una estructura repetitiva**, esta se encargará de actualizar el valor del contador por cada vez que cumple una determinada condición.

Finalmente, para redondear la idea de los contadores, se puede analizar la forma de trabajo que tienen las maquinas contadoras de dinero, usadas mayormente en las entidades bancarias. Estos contadores de billetes cuentan con una velocidad de 1000 billetes por minuto y, como se puede ver en la imagen, cuenta con un contador externo que permite visualizar cómo se actualiza el valor del contador.



<http://www.multipapel.com/productos/733501A01.GIF

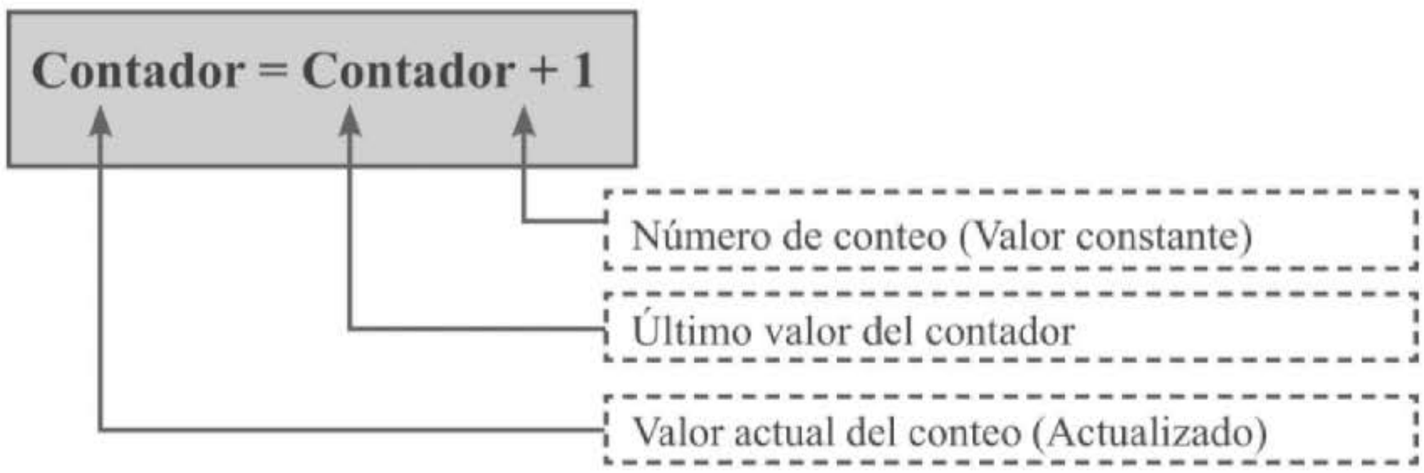
Véase su formato:

Formato	Descripción
Contador = Contador + 1	Contador simple, el cual permite actualizar la variable contadora con un valor constante, valor que es definido por el usuario. Si asigna uno, el conteo se realiza de uno en uno; en caso de asignar dos, entonces el conteo se altera y se realiza de dos en dos.
Contador++	Autoincremento, el cual permite actualizar a la variable contadora estrictamente en una unidad.
Contador+=1	Contador con operador complejo, el cual permite actualizar a la variable contadora con un valor constante definido por el usuario.

Analizando la sentencia contadora:

Se debe considerar, que para realizar un conteo se cuenta con tres distintas formas de realizarlo pero la función es la misma. Es decir, las siguientes sentencias tienen el mismo objetivo:

```
C++;
C+=1;
C = C+1;
```



Donde:

- **Número de conteo:** Determina la forma del conteo, ya que puede ser en ascendente o descendente.
- **Último valor del contador:** Representa el último valor obtenido en el conteo.
- **Valor actual del conteo:** Representa la actualización del valor contenido en el contador.

5.2 Operaciones sobre los contadores

➤ Inicialización

Cuando se declara una variable, ya sea de tipo local o global, como tipo entero en C#, automáticamente se le asigna el valor inicial cero. Por otra parte, el desarrollador puede asignar un valor inicial de acuerdo a su conveniencia.

Véanse las siguientes inicializaciones:

Descripción del caso	Código
Inicializar la variable c en valor cero.	int c=0;
Inicializar el número de factura en 1000.	int nFactura = 1000;

➤ 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.

Véase el caso de una implementación que permite contabilizar la cantidad de números generados aleatoriamente hasta encontrar el número 9:

```
Random aleatorio = new Random();
int c = 0,n;
do
{
   n = aleatorio.Next(1, 10);
   MessageBox.Show("El valor de N: " + n);
   c++;
} while (n != 9);
MessageBox.Show("La cantidad de numeros generados es: " + c);
```

➤ Operación de decremento:

Esta operación realiza un conteo a partir de un valor numérico superior hacia uno inferior.

Véase el caso de una implementación que permite mostrar tres números generados aleatoriamente entre 1 y 100; para lo cual, se realiza un conteo inverso en el ciclo de repeticiones:

```
Random aleatorio = new Random();
int i=3,n;
do
{
   n = aleatorio.Next(1, 100);
   MessageBox.Show("El valor de N: " + n);
   i--;
} while (i>=1);
```

5.3 Los acumuladores

Un acumulador o totalizador es una variable cuyo valor se incrementa o decrementa en un valor variable. Un acumulador suele utilizarse para acumular resultados producidos en las iteraciones de un ciclo de repeticiones.

Asimismo, el uso de los acumuladores se puede dar de dos formas:

- **Fuera de una estructura repetitiva:** Si se quiere usar un acumulador sin estructuras repetitivas, se debe tener en cuenta que dicha variable debe ser declarada de forma global y así actualizar su valor durante las llamadas.
- **Dentro de una estructura repetitiva:** Se actualiza el valor del acumulador por cada vez que cumple una determinada condición.

Finalmente, para redondear la idea de los acumuladores, se pueden mencionar las acumulaciones en monedas que realizan justamente las máquinas tragamonedas que cuando se realiza una jugada ganadora, el acumulador de coins aumenta, mientras que si se pierden turnos, se realiza un acumulador negativo de coins.

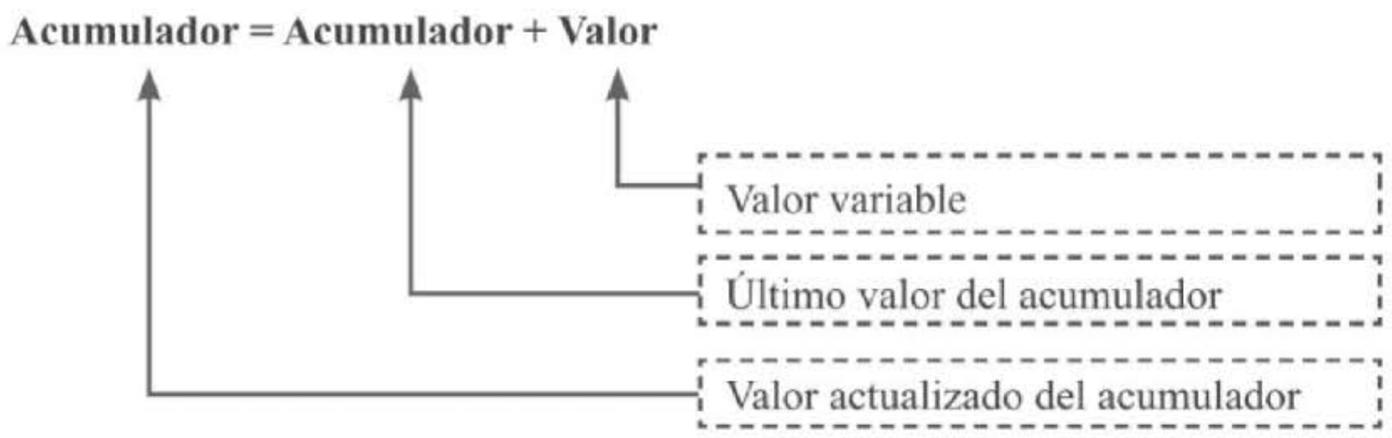


<https://fornaughtdotnet.files.wordpress.com/2011/08/slot-machine-payout.jpg>

Véase su formato:

Formato	Descripción
acumulador = acumulador + valor	Acumulador simple, el cual permite actualizar la variable acumuladora con un valor que no es constante.
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 valor no constante que permite asignar un nuevo valor a la variable acumuladora.
- **Último valor del acumulador:** Es la invocación al último valor del acumulador.
- **Valor actual del acumulador:** Es la asignación del valor final de la variable acumuladora.

5.4 Operaciones sobre los acumuladores

➢ Inicialización

Toda variable acumuladora debe ser inicializada con un valor, dicho valor debe ser definido por el desarrollador. Normalmente, se inicializa con cero.

Descripción	Código
Inicializando el acumulador suma con el valor cero.	double suma = 0;
Inicializando el acumulador coins con el valor inicial 1000; asumiendo que la máquina tragamonedas inicia con una coins de 1000 y el usuario acumulará a partir de este monto.	double coins = 1000;

➢ Operación de incremento

Permite aumentar en valor a la variable acumuladora, de forma que acumula de forma ascendente. Véase algunas referencias a la operación de incremento:

Véase la siguiente implementación para acumular de 5 números aleatorios de dos cifras:

```
Random aleatorio = new Random();
int s = 0;
for (int i = 1; i <= 5; i++)
{
    int n = aleatorio.Next(1, 99);
    MessageBox.Show("El valor de N: " + n);
    s += n;
}
MessageBox.Show("El valor acumulado es: "+s);
```

- **Operación de decremento**

Esta operación realiza la acumulación inversa y, probablemente, el resultado sea negativo en algún momento.

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.

```
Random aleatorio = new Random();
double monto = 5000;
int i=1,retiro;
do
{
   retiro = aleatorio.Next(10, 500);
   MessageBox.Show("El monto a retirar es :" + retiro);
   monto -=retiro;
   i++;
} while (i<=4);
MessageBox.Show("El monto actual es: "+monto.ToString("C"));
```

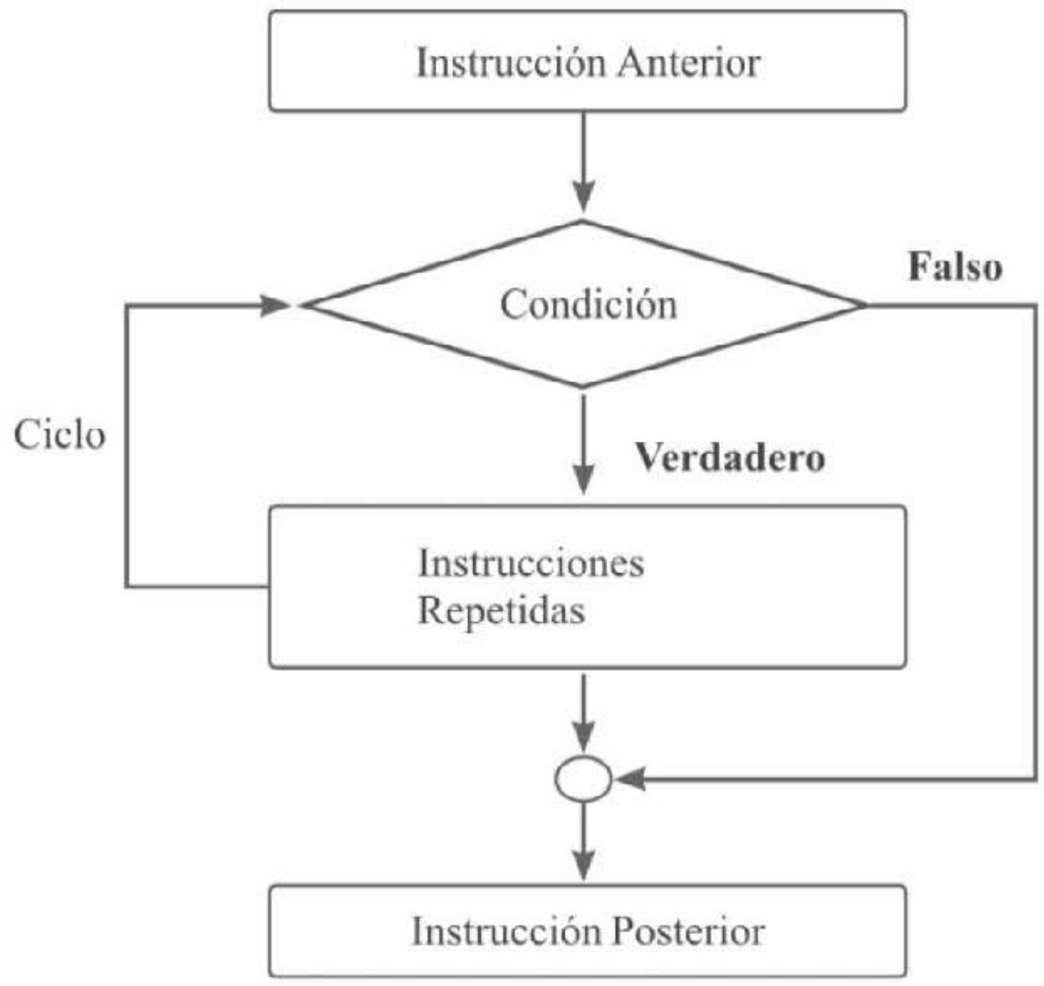
5.5 Bucle While

Permite ejecutar un conjunto de instrucciones un número determinado de veces mientras que la condición especificada en la instrucción While sea verdadera. En caso sea falsa la condición, el ciclo de repeticiones iniciado por While finaliza.

Véase su formato:

```
while (condición){
  //Sentencias repetidas
}
```

Gráficamente:



Donde:

- **Condición:** Se especifica la condición del ciclo de repeticiones. Mientras que dicha condición sea verdadera, el ciclo se continúa ejecutando. Si la condición resulta falso, el ciclo de repeticiones finaliza y continúa con la siguiente instrucción de la aplicación.
- **Instrucciones repetidas:** Se especifican las sentencias que se repiten durante el ciclo While, se debe tener en cuenta que aquí se puede especificar cualquier tipo de sentencia como secuencial, condicional o repetitivas.

Véase algunos códigos que implementan el uso de la instrucción While.

- Listar 10 números aleatorios usando la estructura While.

```
int i =1;
while (i <= 10) {
  Random rnd = new Random();
  int n = rnd.Next(1,100);
  MessageBox.Show(n.ToString());
  i++;
}
```

El ciclo de repetición con While inicia con 1 y la condición indica que mientras que dicho número sea menor a 10 seguirá repitiendo el bucle. Rnd es un objeto de la clase Random el cual permite generar números enteros aleatorios; esta generación es registrada en la variable n; se le asigna 1,100 para generar números entre dicho rango. No olvidar que el ciclo de repeticiones continúa solo si la variable condicional aumenta en una unidad antes de finalizar el While.

- Listar los divisores de un determinado número entero

```
int n =9;
int i = 1;
while (i <= n) {
  if (n%i==0)
    MessageBox.Show(i.ToString());
    i++;
}
```

La variable n tiene el número del cual se quieren obtener los divisores. Luego, el ciclo de repeticiones inicia en uno y culmina cuando se llega al número especificado en n. Dentro del bucle While, se compara si dicho número es divisor de n, si es verdad se imprime su divisor que se encuentra en la variable "i".

- Listar los 10 primeros números pares de una serie máxima permitida por los números enteros.

```
int i = 1;
int cp = 0;
while (i <= int.MaxValue)
{
   if (i % 2 == 0)
   {
     MessageBox.Show(i.ToString());
     cp++;
   }
   if (cp == 10)
     break;
     i++;
}
```

La variable i tiene la misión de ayudar a la generación de repeticiones con While debe iniciarse en el valor uno. El valor límite es dado en la condición While especificando el máximo valor que pueden tener los números enteros, usando la función int.MaxValue. La variable cp es la encargada de contabilizar la cantidad de pares encontrados; es así que dentro del ciclo While se compara si cp=10 para determinar si ya se tienen los 10 pares y finaliza el ciclo de repeticiones generado por While usando la sentencia break.

5.6 Bucle do while

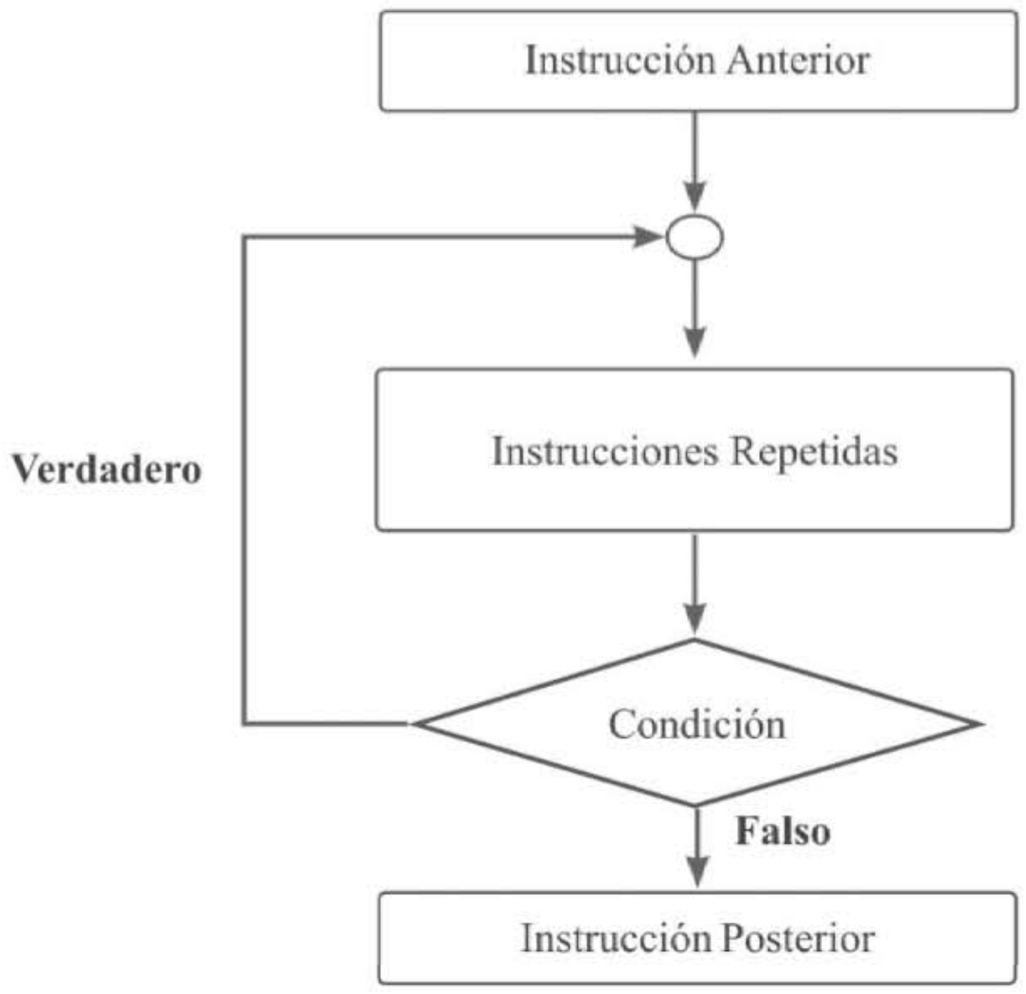
Este bucle permite repetir un conjunto de instrucciones un número indefinido de veces, todo depende de la condición establecida; eso no excluye que pueda generar ciclos fijos pero se adapta mejor en ciclos no conocidos.

Por ejemplo, en un sorteo de números en el cual se fija un número como base, mientras no se encuentre dicho número, el bucle seguirá funcionando hasta encontrarlo y finalizará solo cuando el número sea el buscado; caso contrario, seguirá funcionando el bucle.

Véase su formato:

```
do
{
  //Sentencias repetidas
} while (Condicion);
```

Gráficamente:



Donde:

- **Instrucciones repetidas:** Se especifican las sentencias que se repiten durante el ciclo do while. El ciclo empieza sin condición, por lo tanto, las sentencias se ejecutan por lo menos una vez.
- **Condición:** Se especifica la condición del ciclo de repeticiones. Mientras que dicha condición sea verdadera, el ciclo do while continua ejecutando. Si la condición resulta falso, el ciclo de repeticiones finaliza y continúa con la siguiente instrucción de la aplicación.

Véase algunos códigos que implementan el uso de la instrucción While:

- Listar 10 números aleatorios usando la estructura do while.

```
int i = 1;
do
{
    Random rnd = new Random();
    int n = rnd.Next(1, 100);
    MessageBox.Show(n.ToString());
    i++;
} while (i <= 10);
```

- Listar los divisores de un determinado número entero.

```
int n = 9;
int i = 1;
do
{
  if (n % i == 0)
    MessageBox.Show(i.ToString());
    i++;
} while (i <= n);
```

- Listar los 10 primeros números pares de una serie máxima permitida por los números enteros.

```
int i = 1;
int cp = 0;
do
{
  if (i % 2 == 0)
```

```
    {
      MessageBox.Show(i.ToString());
      cp++;
    }
    if (cp == 10) break;
    i++;
  } while (i <= int.MaxValue);
```

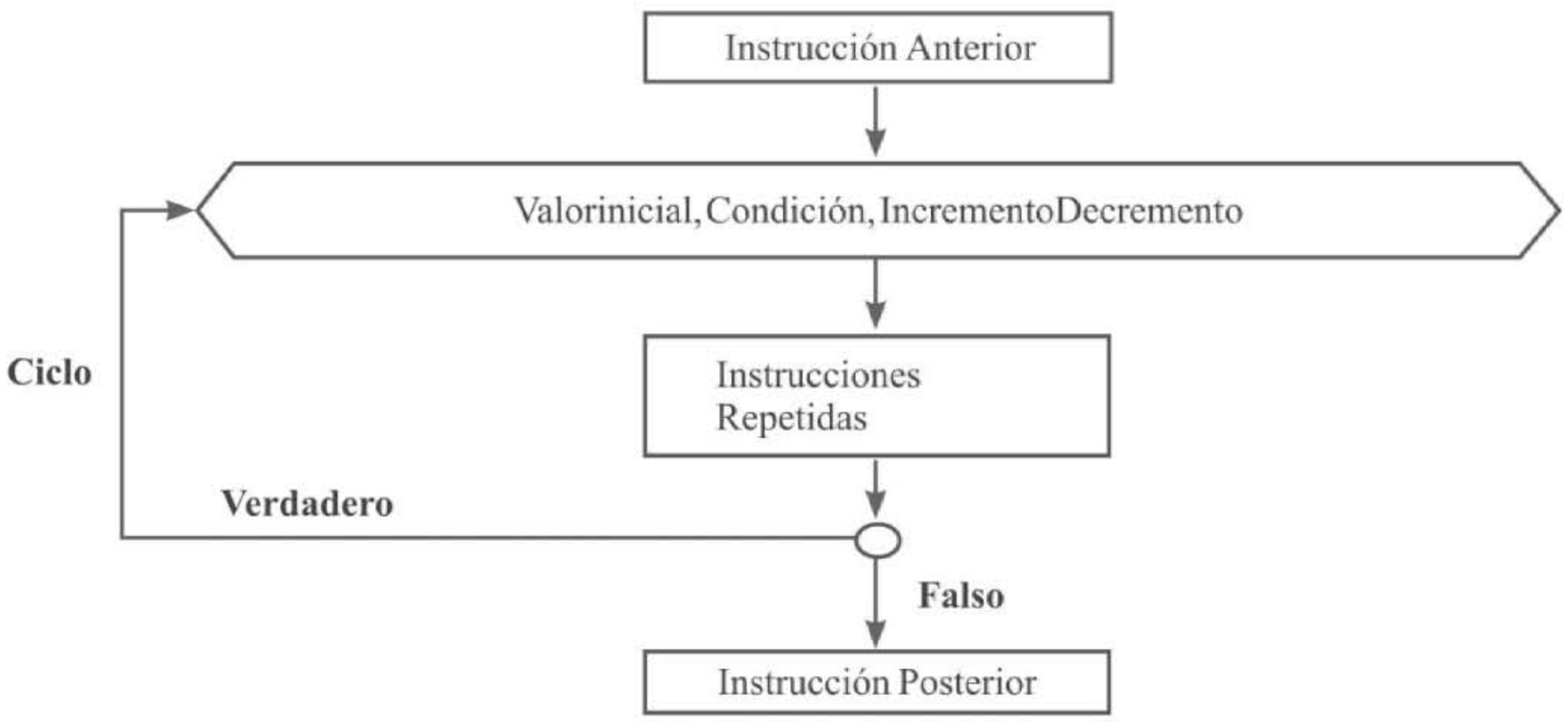
5.7 Bucle For

Es el más usado dentro de las aplicaciones por su forma sencilla de implementar; la característica principal es que en una sola instrucción define todo el ciclo de repeticiones.

Véase su formato:

```
for(valorInicial; condición; IncrementoDecremento){
   //Sentencias repetidas;
}
```

Gráficamente:



Donde:

- **ValorInicial:** Representa el valor de inicial del ciclo de repeticiones for; se debe tener en cuenta que no siempre inicia en uno o cero, esta decisión lo tomara el desarrollador.

- **Condición:** Es la expresión que valida el bucle de la estructura For. Si la condición resulta verdadera el ciclo de repeticiones continua; caso contrario, finaliza el proceso.
- **IncrementoDecremento:** Define la secuencia en que el ciclo de repeticiones ejecuta los procesos, esta puede ser de forma ascendente o descendente.

Véanse algunos códigos que implementan el uso de la instrucción while:

- Listar 10 números aleatorios usando la estructura for.

```
for(int i=0;i<=10;i++)
{
  Random rnd = new Random();
  int n = rnd.Next(1, 100);
  MessageBox.Show(n.ToString());
}
```

- Listar los divisores de un determinado número entero.

```
int n = 9;
for(int i=1;i<=n;i++)
{
  if (n % i == 0) MessageBox.Show(i.ToString());
  i++;
}
```

- Listar los 10 primeros números pares de una serie máxima permitida por los números enteros.

```
int cp = 0;
for(int i=1; i<=int.MaxValue; i++)
{
  if (i % 2 == 0)
  {
    MessageBox.Show(i.ToString());
    cp++;
  }
  if (cp == 10) break;
}
```

5.8 Casos desarrollados

Caso desarrollado 1 : contadores - control de registro de participantes

La compañía minera Casapalca está comprometida con el desarrollo económico del país y la generación de fuentes de trabajo para sus colaboradores, que impulsen su desarrollo personal y profesional. Para seguir sus objetivos, la empresa ha preparado una capacitación para sus empleados con cargo de jefes, operarios, practicantes y administrativos.

Se tiene en cuenta lo siguiente:

- Al iniciar la aplicación se debe mostrar un número generado de manera automática, con el formato "0000" de forma correlativa, asimismo, la fecha y hora del registro; este último debe mostrarse en forma digital usando el control Timer.
- Los cargos de los empleados a capacitarse se deben encontrar en un control ComboBox, estos se muestran en la siguiente tabla:

Cargo	Descripción
Jefe	Empleado asignado como jefe de operaciones de la empresa.
Operario	Empleado de planta de la empresa.
Administrativo	Empleado que realiza labores administrativas en la empresa.
Practicante	Considerado como empleado temporal que tiene como misión asistir al personal administrativo.

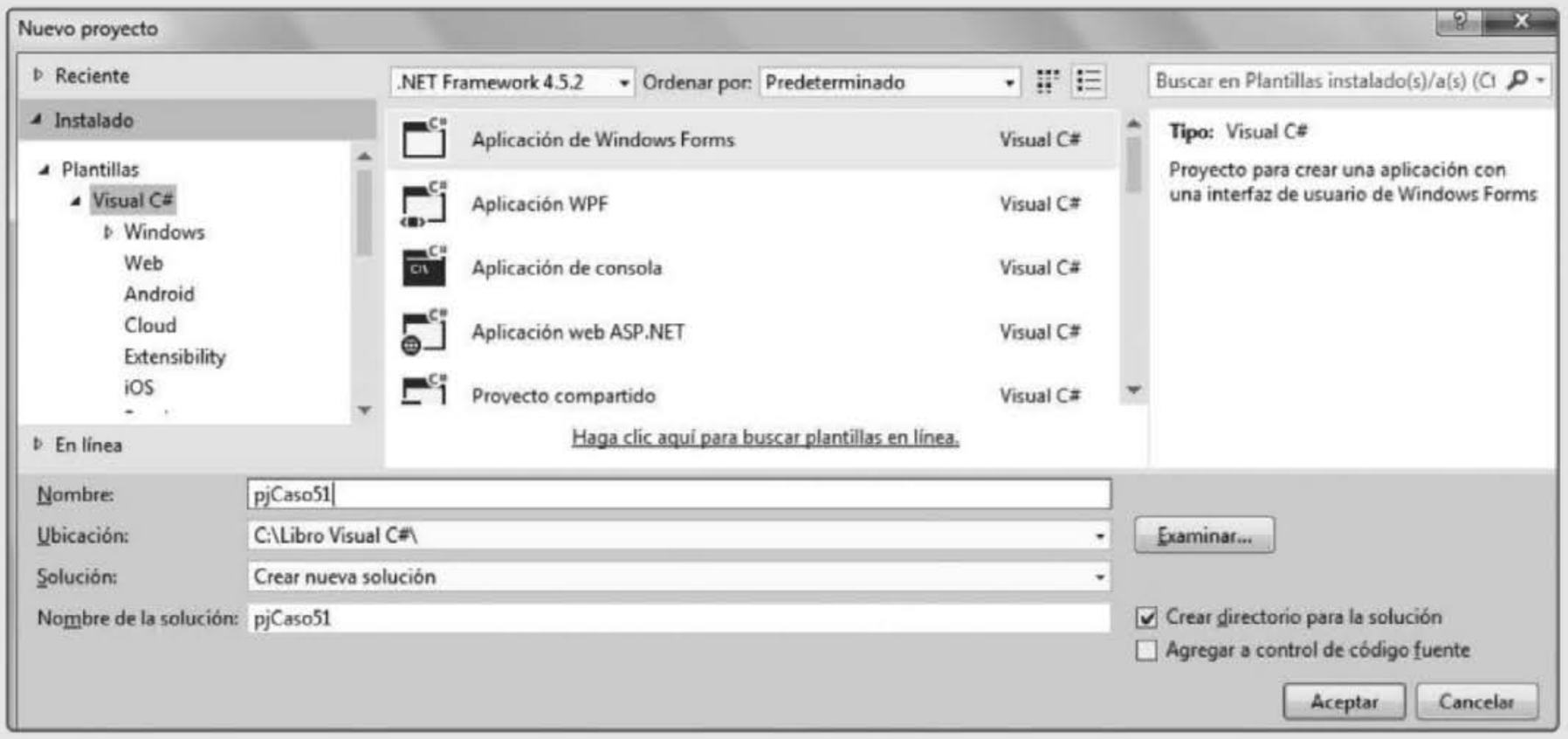
- Los datos deben ser mostrados en un control ListView, entre ellos, el número de registro, nombre del empleado, cargo, fecha y hora de registro.
- Finalmente, se deben mostrar las estadísticas de los registros como el total de jefes, operarios, administrativos y practicantes registrados y mostrarlo en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

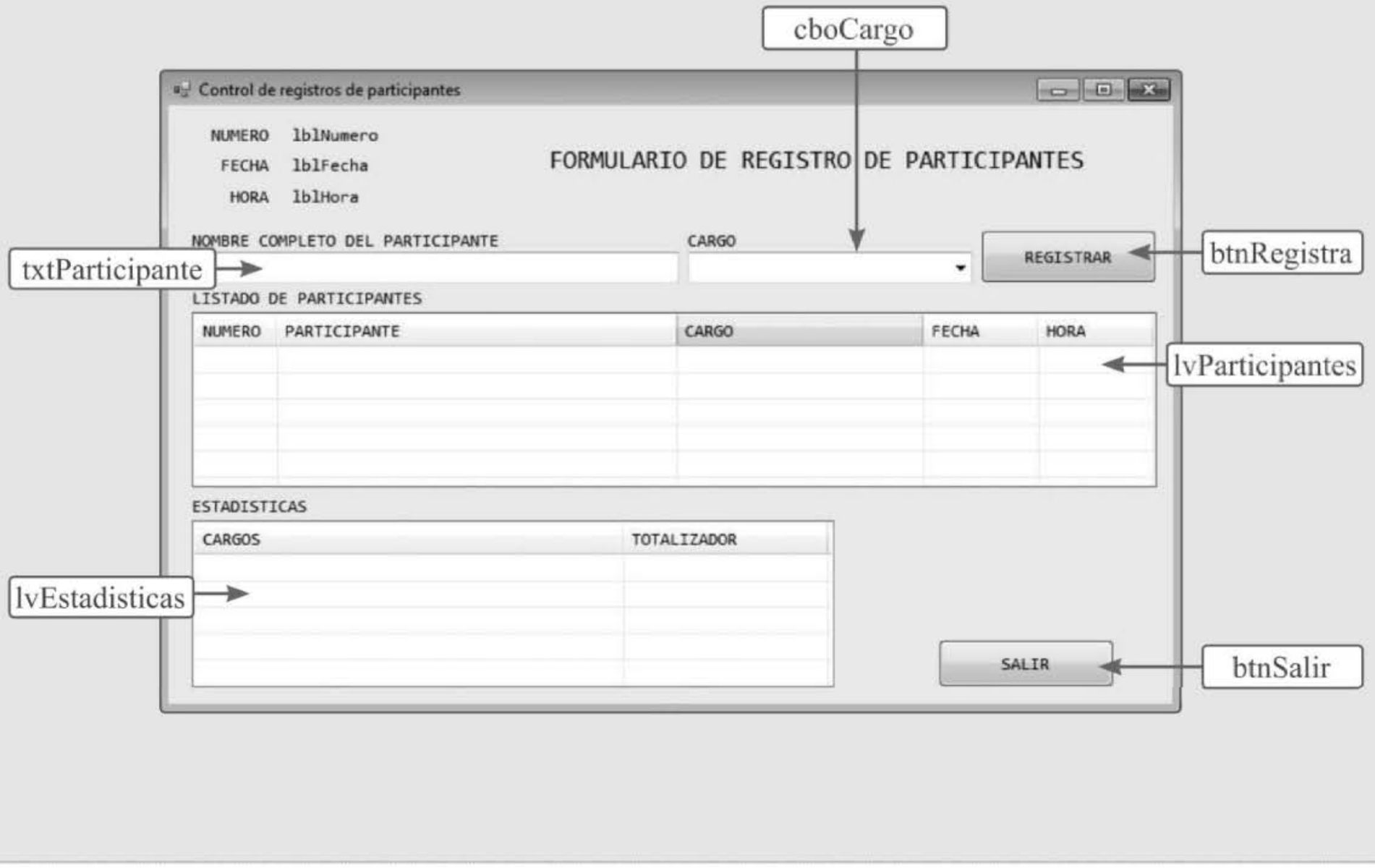
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso51**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmParticipantes**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



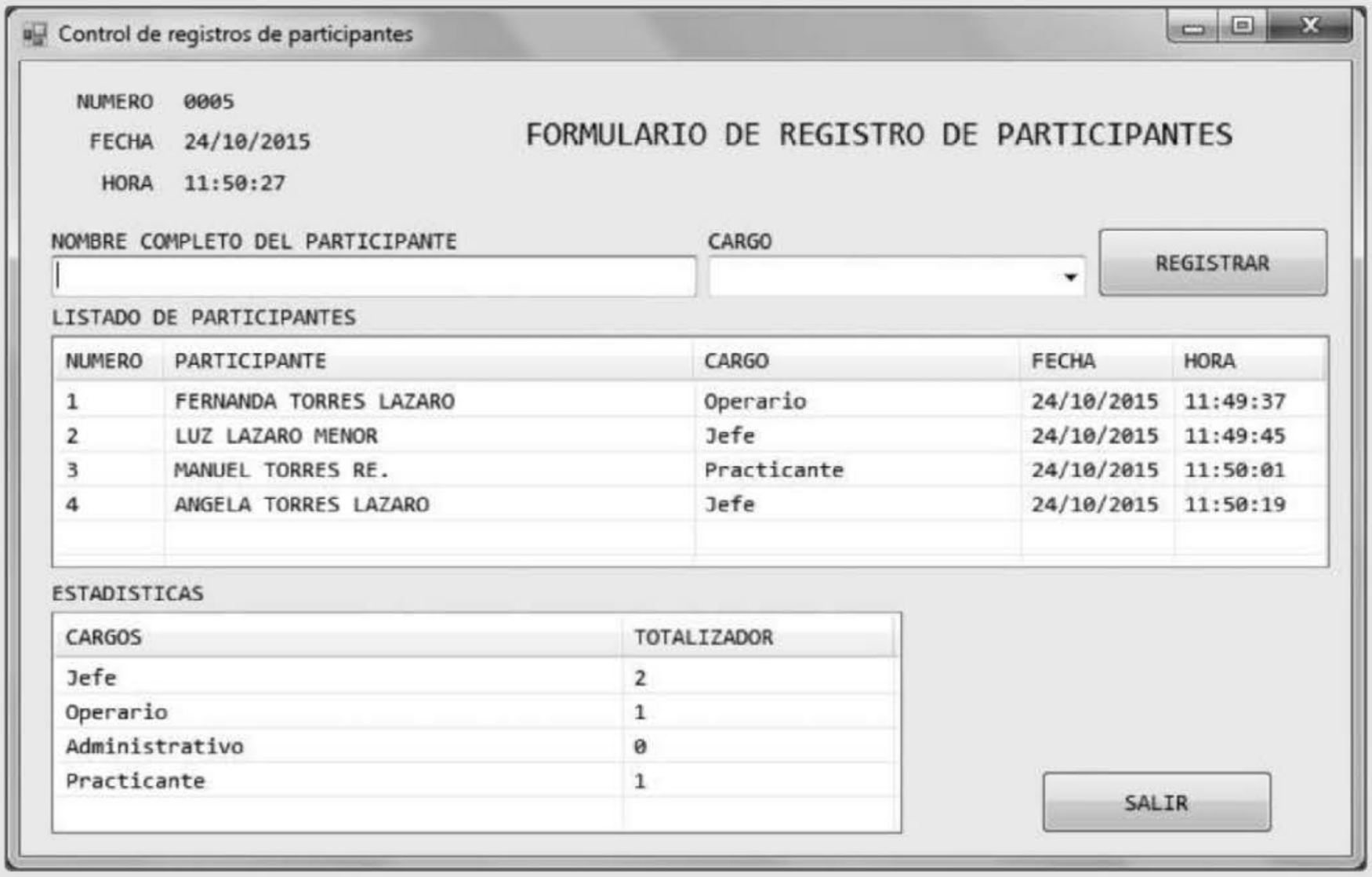
Estadísticas de los controles usados en el frmParticipantes:

Clase	N.º elementos	Controles
Form	01	Form1
Label	11	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8, Label9, Label10, Label11
TextBox	01	Textbox1
ComboBox	01	ComboBox1
Button	02	Button1, Button2
ListView	02	ListView1, ListView2
TOTAL	18	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmParticipantes Control de registro de participantes
Label1	Text	CONTROL DE REGISTRO DE PARTICIPANTES
Label2	Text	NUMERO
Label3	Text	FECHA
Label4	Text	HORA
Label5	(Name) Text	lblNumero lblNumero
Label6	(Name) Text	lblFecha lblFecha
Label7	(Name) Text	lblHora lblHora
Label8	Text	NOMBRE COMPLETO DEL PARTICIPANTE
Label9	Text	CARGO
Label10	Text	LISTADO DE PARTICIPANTES
Label11	Text	ESTADISTICAS
Textbox1	(Name)	txtParticipante
ComboBox1	(Name)	cboCargo
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvParticipantes True
ListView2	(Name) GridLines	lvEstadisticas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso51
{
    public partial class frmParticipantes : Form
    {
        int num;
        int cJefe, cOperario, cAdministrativo, cPracticante;

        public frmParticipantes()
        {
            InitializeComponent();
            tHora.Enabled = true;
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
```

```
        DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                         "Participantes",
                                        MessageBoxButtons.YesNo,
                                        MessageBoxIcon.Exclamation);
        if (r == DialogResult.Yes) this.Close();
    }

    private void frmParticipantes_Load(object sender, EventArgs e)
    {
        num++;
        lblFecha.Text = DateTime.Now.Date.ToString("d");
        lblNumero.Text = num.ToString("D4");
    }

    private void tHora_Tick(object sender, EventArgs e)
    {
        lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
    }

    private void btnRegistrar_Click(object sender, EventArgs e)
    {
        //Capturando los datos
        DateTime fecha, hora;
        string participante, cargo;
        int numero;
        participante = txtParticipante.Text;
        numero = int.Parse(lblNumero.Text);
        fecha = DateTime.Parse(lblFecha.Text);
        hora = DateTime.Parse(lblHora.Text);
        cargo = cboCargo.Text;

        //Contabilizar la cantidad segun los cargos
        switch (cargo) {
            case "Jefe": cJefe++; break;
            case "Operario": cOperario++; break;
            case "Administrativo": cAdministrativo++; break;
            case "Practicante": cPracticante++; break;
        }

        //Imprimiendo el registro
        ListViewItem fila = new ListViewItem(numero.ToString());
        fila.SubItems.Add(participante);
        fila.SubItems.Add(cargo);
        fila.SubItems.Add(fecha.ToString("d"));
        fila.SubItems.Add(hora.ToString("hh:mm:ss"));
        lvParticipantes.Items.Add(fila);

        //Imprimiendo las estadisticas
        lvEstadisticas.Items.Clear();
        string[] elementosFila = new string[2];
        ListViewItem row;

        //Añadimos la primera fila al lvEstadisticas
        elementosFila[0] = "Jefe";
        elementosFila[1] = cJefe.ToString();
        row = new ListViewItem(elementosFila);
        lvEstadisticas.Items.Add(row);
```

```
            //Añadimos la segunda fila al lvEstadisticas
            elementosFila[0] = "Operario";
            elementosFila[1] = cOperario.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Añadimos la tercera fila al lvEstadisticas
            elementosFila[0] = "Administrativo";
            elementosFila[1] = cAdministrativo.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Añadimos la cuarta fila al lvEstadisticas
            elementosFila[0] = "Practicante";
            elementosFila[1] = cPracticante.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Mostrando el nuevo numero de registro
            num++;
            lblNumero.Text = num.ToString("D4");

            //Limpiando los controles
            txtParticipante.Clear();
            cboCargo.SelectedIndex = -1;
            txtParticipante.Focus();
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Generar un número automático de registro**

  Implica que se debe declarar una variable global de tipo entero, la cual se auto-incrementa al registrar un participante en la aplicación. Los pasos para generar el código en la aplicación son los siguientes:

  - Declare un número entero de manera global con el siguiente código > **int num**.
  - En el evento **Load** del formulario, coloque el siguiente código:

```
private void frmParticipantes_Load(object sender, EventArgs e)
        {
                num++;
                lblNumero.Text = num.ToString("D4");
        }
```

El valor num aumenta en uno, puesto que cuando se declara en la sección de globales automáticamente se inicializa con el valor cero. Luego, se asigna al lblNumero. Text el valor del número, se determina qué aplicación necesita mostrar con ceros a la izquierda, para lo cual se agrega el parámetro D4 al método ToString, esto indica 4 ceros a la izquierda.

- Finalmente, se debe repetir el código dentro del botón **REGISTRAR**; pues cuando un participante se registra, el numero autogenerado debe aumentar.

➢ **Mostrar la fecha actual al inicio del formulario**

Una regla del proceso es mostrar la fecha actual en el formulario, para lo cual se deben realizar los siguientes pasos:

- Colocar el siguiente código dentro del evento **Load** del formulario:

```
private void frmParticipantes_Load(object sender, EventArgs e)
        {
                lblFecha.Text = DateTime.Now.Date.ToString("d");
        }
```

➢ **Mostrar la hora activa en el label de la aplicación**

La idea es hacer que en el control Label (lblHora) se muestre la hora actual como si fuera un reloj digital. Para lo cual, se deben realizar los siguientes pasos:

- Busque en el cuadro de herramientas **Timer**.
- Arrastre un control timer al formulario, en la parte inferior del formulario aparece el nombre **Timer1**.
- Seleccione desde esta barra el control Timer1, y cambie de nombre en las propiedades por **tHora**.
- Haga doble clic sobre el control **tHora**, aparecera el siguiente código:

```
private void tHora_Tick(object sender, EventArgs e)
        {
                lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
        }
```

- Y finalmente, para que el reloj se muestre digitalizado coloque el siguiente código **tHora.Enabled=true**; debajo de la llamada al **InitializeComponent** ubicado en el método constructor de la clase frmPostulantes.

➤ **Contabilizar según los cargos**

Lo primero que se debe de realizar es verificar que tipo de cargo se ha seleccionado para poder aumentar las variables contadoras. Se deben realizar los siguientes pasos:

- Inicialice las variables de conteo en la sección **GLOBALES**:

```
int cJefe, cOperario, cAdministrativo, cPracticante;
```

- Compruebe el tipo de cargo dentro del botón **REGISTRAR** y colocar el siguiente código:

```
switch (cargo) {
            case "Jefe": cJefe++; break;
            case "Operario": cOperario++; break;
            case "Administrativo": cAdministrativo++; break;
            case "Practicante": cPracticante++; break;
        }
```

La estructura switch se encarga de verificar el tipo de cargo y en cada uno de los casos se aumenta el contador solo si cumple con una de estas especificaciones. No olvidar que cada case debe contar con la sentencia break, incluso el último case.

➤ **Imprimir valores directamente en la lista de estadísticas**

Ya se visto como imprimir valores en un control listview pero aquí se presenta un caso particular, pues se deben imprimir los mismos datos cada vez que se registra un nuevo participante; lo único que debe modificarse son los valores de conteo ya que deben mostrar los conteos actualizados. Veánse los siguientes pasos:

```
//Imprimiendo las estadisticas
        lvEstadisticas.Items.Clear();
        string[] elementosFila = new string[2];
        ListViewItem row;
```

Se debe empezar por limpiar el control ListView de estadística, ya que por cada registro se actualizan los datos. Luego, se define un arreglo llamado elementosFila que tiene la misión de guardar la información de las estadísticas fila por fila; se le asigna dos entre corchetes, por el número de columnas que cuenta estadísticas que son el tipo de cargo y el total de conteos. El objeto row es el enlace entre los datos a mostrar y el control ListView. Ahora, se agrega la primera fila de estadística:

```
//Añadimos la primera fila al lvEstadisticas
elementosFila[0] = "Jefe";
elementosFila[1] = cJefe.ToString();
row = new ListViewItem(elementosFila);
lvEstadisticas.Items.Add(row);
```

Se debe enviar información a la columna primera columna (0) y se le asignas el texto "Jefe", en la segunda columna se agrega el valor obtenido del conteo llamado cJefe; una vez guardado en el arreglo "elementosFila", es pasado al objeto "row" para que pueda ser visto en el control ListView con la sentencia lvEstadisticas. Items.Add(row). De la misma manera, se realiza para la impresión de los contadores de Operario, Administrativo y Practicante.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 2 : acumuladores - planilla de sueldos

La empresa de ventas de autopartes Partes-Car S.A. ubicada en la ciudad de Lima, ha experimentado un crecimiento en ventas para lo cual se vio en la necesidad de aumentar el personal. En vista de eso, necesitan una aplicación que permita controlar la planilla de sueldos de sus empleados por locales. Para lo cual, se requiere registrar la fecha de registro de la planilla, DNI, nombre completo y categoría del empleado; este último determina el sueldo de cada empleado. Finalmente, se necesita reportar el total empleados por categoría y, además, el monto acumulado por cada tipo de empleado.

Se tiene en cuenta lo siguiente:

- Al iniciar la aplicación, se debe mostrar la fecha de manera automática.
- La asignación de sueldos se realiza mediante la categoría del empleado como se muestra en la siguiente tabla:

Categoría	Sueldo
Jefe	$ 5000.00
Supervisor	$ 3500.00
Vendedor	$ 1500.00
Administrador	$ 7 000.00

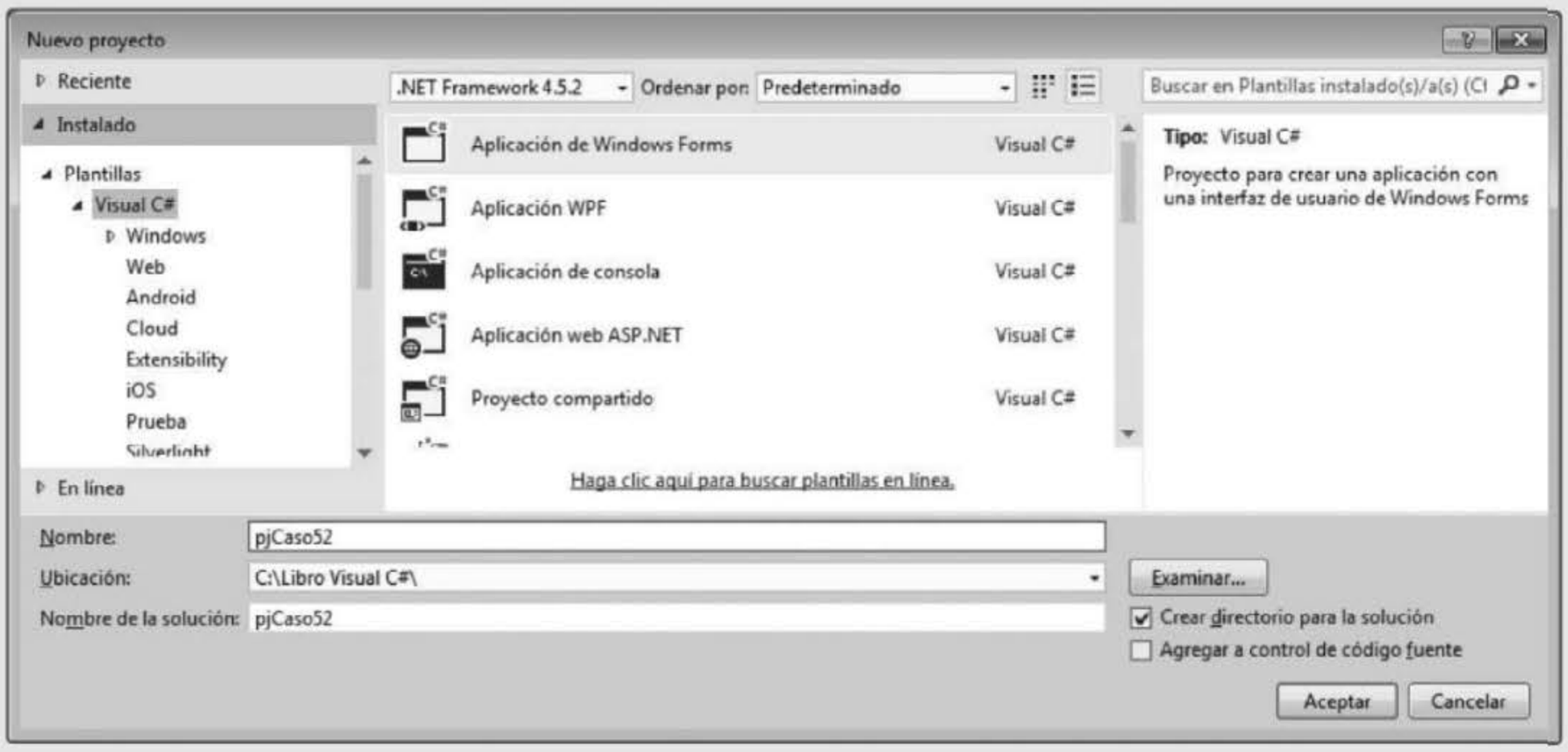
- El botón **REGISTRAR** envía los datos del empleado y el sueldo a un control ListView.
- Finalmente, se debe mostrar las estadísticas como el total de empleados por categoría y los montos acumulados por cada tipo de categoría en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso52**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmPlanilla**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmPlanilla:

Clase	N.o elementos	Controles
Form	01	Form1
Label	09	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8, Label9
TextBox	01	Textbox1, Textbox2
ComboBox	01	ComboBox1
Button	02	Button1, Button2, Button3
ListView	02	ListView1, ListView2
TOTAL	16	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmPlanilla Planilla de sueldos
Label1	Text	PLANILLA DE SUELDOS
Label2	Text	FECHA
Label3	(Name) Text	lblFecha lblFecha
Label4	Text	DNI
Label5	Text	EMPLEADO
Label6	(Name) Text	lblHora lblHora
Label7	(Name) Text	lblSueldo lblSueldo
Label8	Text	LISTADO DE EMPLEADOS
Label9	Text	ESTADISTICAS
Textbox1	(Name)	txtDNI
Textbox2	(Name)	txtEmpleado
ComboBox1	(Name)	cboCategoria
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnSalir SALIR
Button3	(Name) Text	btnCancelar CANCELAR
ListView1	(Name) GridLines	lvEmpleados True
ListView2	(Name) GridLines	lvEstadisticas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso52
{
    public partial class frmPlanilla : Form
    {
        double sueldo;
        int cJefe, cSupervisor, cVendedor, cAdministrador;
        double aJefe, aSupervisor, aVendedor, aAdministrador;

        public frmPlanilla()
        {
```

```
        InitializeComponent();
    }

    private void frmPlanilla_Load(object sender, EventArgs e)
    {
        lblFecha.Text = DateTime.Now.ToString("d");
    }

    private void cboCategoria_SelectedIndexChanged(…)
    {
        //Capturar la categoria del empleado
        string categoria = cboCategoria.Text;

        switch (categoria)
        {
            case "Jefe": sueldo = 5000; break;
            case "Supervisor": sueldo = 3500; break;
            case "Vendedor": sueldo = 1500; break;
            case "Administrador": sueldo = 7000; break;
        }
        lblSueldo.Text = sueldo.ToString("C");
    }

    private void btnRegistrar_Click(object sender, EventArgs e)
    {
        //Capturando los datos
        string dni = txtDNI.Text;
        string empleado = txtEmpleado.Text;
        string categoria = cboCategoria.Text;

        //Conteos y acumulaciones
        switch (categoria)
        {
            case "Jefe":
                cJefe++;
                aJefe += sueldo;
                break;
            case "Supervisor":
                cSupervisor++;
                aSupervisor += sueldo;
                break;
            case "Vendedor":
                cVendedor++;
                aVendedor += sueldo;
                break;
            case "Administrador":
                cAdministrador++;
                aAdministrador += sueldo;
                break;
        }

        //Impresion de planilla
        ListViewItem fila = new ListViewItem(dni);
        fila.SubItems.Add(empleado);
        fila.SubItems.Add(categoria);
        fila.SubItems.Add(sueldo.ToString("C"));
        lvEmpleados.Items.Add(fila);
```

```
            //Imprimiendo las estadisticas
            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[3];
            ListViewItem row;

            //Impresion de los datos del Jefe
            elementosFila[0] = "Jefe";
            elementosFila[1] = cJefe.ToString();
            elementosFila[2] = aJefe.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Impresion de los datos del Supervisor
            elementosFila[0] = "Supervisor";
            elementosFila[1] = cSupervisor.ToString();
            elementosFila[2] = aSupervisor.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Impresion de los datos del Vendedor
            elementosFila[0] = "Vendedor";
            elementosFila[1] = cVendedor.ToString();
            elementosFila[2] = aVendedor.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Impresion de los datos del Administrador
            elementosFila[0] = "Administrador";
            elementosFila[1] = cAdministrador.ToString();
            elementosFila[2] = aAdministrador.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Limpiando los controles
            txtDNI.Clear();
            txtEmpleado.Clear();
            cboCategoria.SelectedIndex = -1;
            lblSueldo.Text = (0).ToString("C");
            txtEmpleado.Focus();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Planilla",
                                        MessageBoxButtons.YesNo,
                                            MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Declaración global de los contadores y acumuladores**

  Las variables de conteos y acumulaciones deben ser declaradas como globales. Dichas variables se inicializan automáticamente en cero. La sección global se ubica dentro del bloque "public partial class". El código se muestra de la siguiente manera:

```
public partial class frmPlanilla : Form
        {
         double sueldo;
         int cJefe, cSupervisor, cVendedor, cAdministrador;
           double aJefe, aSupervisor, aVendedor, aAdministrador;
}
```

- **Mostrar el sueldo según la categoría seleccionada del empleado**

  La idea es seleccionar una categoría y mostrar el sueldo en el control lblSueldo. Para esto, se debe abrir el evento "SelectedIndexChanged" del cuadro combinado y colocar el siguiente código:

```
private void cboCategoria_SelectedIndexChanged(…)
        {
                //Capturar la categoria del empleado
             string categoria = cboCategoria.Text;

                switch (categoria)
        {
            case "Jefe": sueldo = 5000; break;
            case "Supervisor": sueldo = 3500; break;
            case "Vendedor": sueldo = 1500; break;
            case "Administrador": sueldo = 7000; break;
        }
        lblSueldo.Text = sueldo.ToString("C");
        }
```

  Primero, se captura la categoría seleccionada; luego, se compara dicho valor con las categorías establecidas en el caso y se asigna el monto de los sueldos según la categoría. Finalmente, se imprime el resultado en el control lblSueldo.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 3 : for - seguros de vida

La compañía de seguros Buena Vida S. A. C. cuenta con más de 400 vendedores de seguro en el Perú; los cuales desean una aplicación que les permita controlar el registro de las proformas de seguro que realizan con un determinado cliente. Esta proforma se realiza de forma corporativa; por tal razón, la proforma solicita la razón social de la empresa, el tipo de seguro y la cantidad de empleados a los que se les asignará el seguro. Téngase en cuenta que una empresa puede registrar distintos tipos de seguros para todos sus empleados.

Se tiene en cuenta lo siguiente:

- La tasa por tipo de seguro se muestra en la siguiente tabla:

Tipo de seguro	Costo unitario
Clásica	$ 50.00
Platino	$ 80.00
Oro	$ 150.00

- Además, para todos los tipos de seguro existe un asignación de descuento como se muestra en la siguiente tabla:

Tipo de seguro	Condición	Costo adicional por empleado
Clásica	Mayor a 10 empleados	$ 10.00
Platino	Mayor a 9 empleados	$ 8.00
Oro	Mayor a 6 empleados	$ 15.00

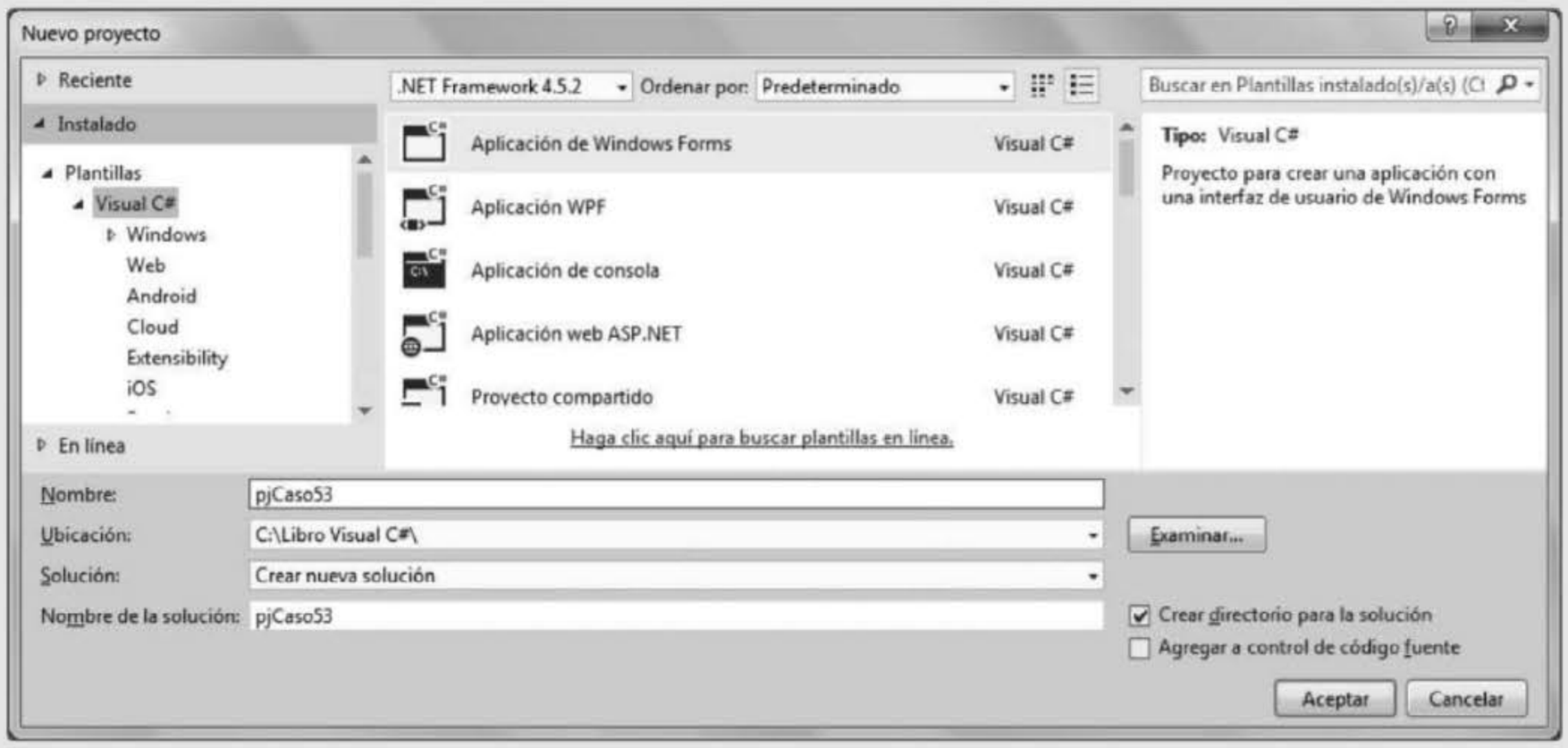
- El botón **REGISTRAR** envía los datos del tipo de seguro, cantidad de empleados por tipo y el pago mensual obtenido en un control ListView.
- Finalmente, se deben mostrar estadísticas como el total de personas aseguradas y el monto total a cancelar por la empresa solicitante en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

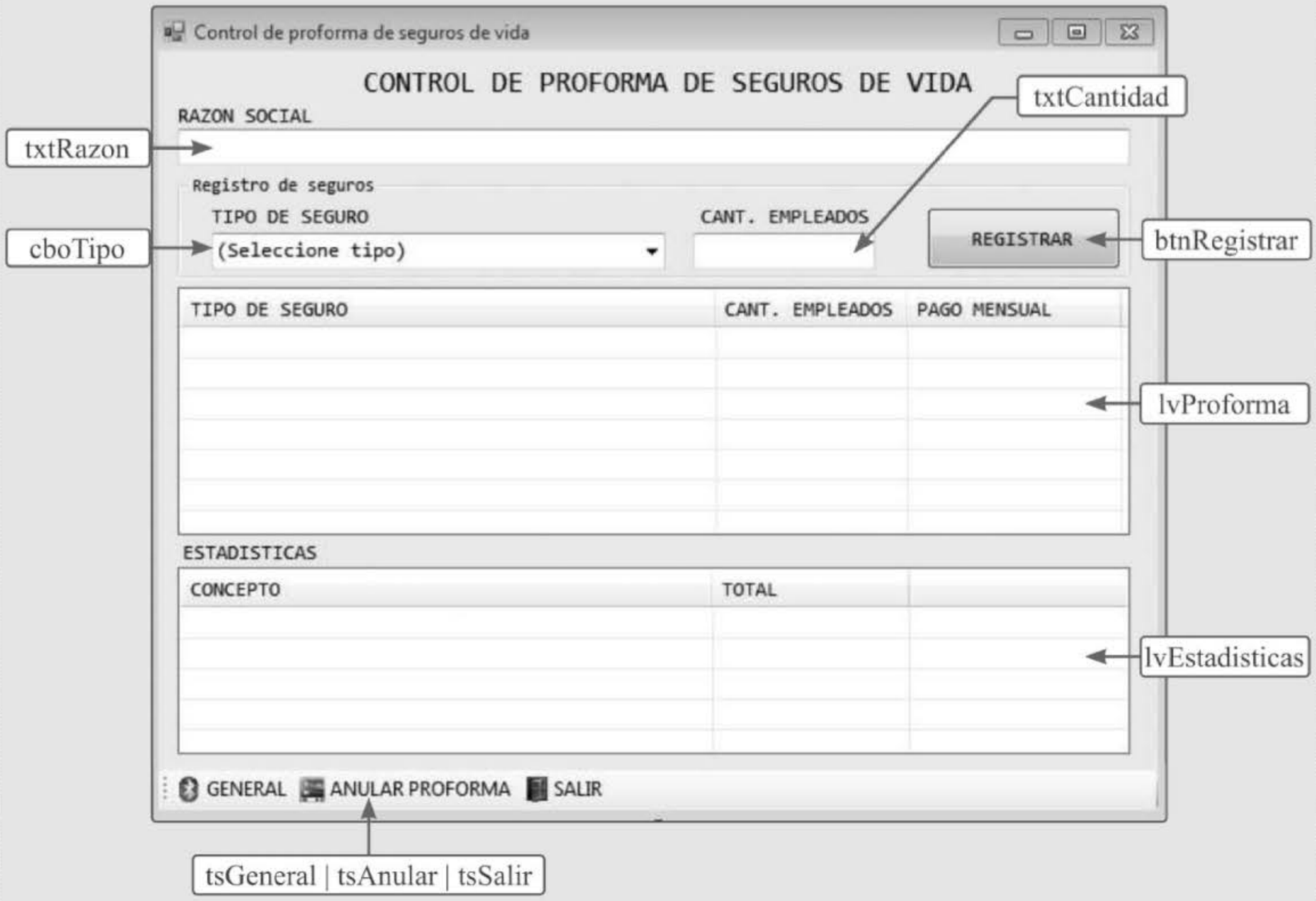
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Selección **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso53**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo** asigne el nombre **frmProformas**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmProformas:

Clase	N.º elementos	Controles
Form	01	Form1
Label	05	Label1, Label2, Label3, Label4, Label5
TextBox	02	Textbox1, Textbox2
ComboBox	01	ComboBox1
Button	01	Button1
ListView	02	ListView1, ListView2
ToolStrip	01	ToolStrip1
TOTAL	13	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmProformas Planilla de sueldos
Label1	Text	CONTROL DE PROFORMA DE SEGUROS DE VIDA
Label2	Text	RAZON SOCIAL
Label3	Text	TIPO DE SEGURO
Label4	Text	CANT. EMPLEADOS
Label5	Text	ESTADISTICAS
Textbox1	(Name)	txtRazonSocial
Textbox2	(Name)	txtCantidad
ComboBox1	(Name)	cboTipo
GroupBox1	Text	Registro de seguros
ToolStrip1	Dock	Botton
Button1	(Name) Text	btnRegistrar REGISTRAR
ListView1	(Name) GridLines	lvProforma True
ListView2	(Name) GridLines	lvEstadisticas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso53
{
    public partial class frmProformas : Form
    {
        public frmProformas()
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
```

```
            //Capturando los datos
            string razon = txtRazonSocial.Text;
            string tipo = cboTipo.Text;
            int cantidad = int.Parse(txtCantidad.Text);

            //Calculando el pago mensual por tipo de seguro
            double pagoMensual=0;
            switch (tipo)
            {
                case "Inversion Clasica":
                    if (cantidad <= 10)
                        pagoMensual = 50 * cantidad;
                    else
                        pagoMensual = (50* cantidad)+(10*(cantidad-10));
                    break;
                case "Inversion Platino":
                    if (cantidad <= 8)
                        pagoMensual = 80 * cantidad;
                    else
                        pagoMensual = (80 * cantidad) + (8 * (cantidad - 8));
                    break;
                case "Inversion Oro":
                    if (cantidad <= 5)
                        pagoMensual = 150 * cantidad;
                    else
                        pagoMensual = (150 *cantidad) + (15 * (cantidad - 5));
                    break;
            }

            //Imprimiendo el detalle de la proforma
            ListViewItem fila = new ListViewItem(tipo);
            fila.SubItems.Add(cantidad.ToString());
            fila.SubItems.Add(pagoMensual.ToString("0.00"));
            lvProforma.Items.Add(fila);

        }

        private void tsGeneral_Click(object sender, EventArgs e)
        {
            //Determinar el total de personas aseguradas
            int totalAsegurados = 0;
            for (int i = 0; i < lvProforma.Items.Count; i++)
            {
              if (lvProforma.Items[i].SubItems[0].Text != "")
              totalAsegurados+=int.Parse(lvProforma.Items[i].SubItems[1].Text);
            }

            //Determinar el monto total acumulado a cancelar
            double total = 0;
            for (int i = 0; i < lvProforma.Items.Count; i++)
            {
```

```
            if (lvProforma.Items[i].SubItems[0].Text != "")
            total += double.Parse(lvProforma.Items[i].SubItems[2].Text);
        }

        //Impresion de las estadisticas
        lvEstadisticas.Items.Clear();
        string[] elementosFila = new string[2];
        ListViewItem row;

            elementosFila[0] = "Total de personas aseguradas";
            elementosFila[1] = totalAsegurados.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto total a cancelar";
            elementosFila[1] = total.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        private void tsAnular_Click(object sender, EventArgs e)
        {
            DialogResult r=MessageBox.Show("Esta seguro de anular la proforma?",
                                           "Seguros",
                                           MessageBoxButtons.YesNo,
                                           MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes)
            {
                txtRazonSocial.Clear();
                cboTipo.Text = "(Seleccione tipo)";
                txtCantidad.Clear();
                txtRazonSocial.Focus();
                lvProforma.Items.Clear();
                lvEstadisticas.Items.Clear();
            }
        }

        private void tsSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Seguros",
                                             MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Exclamation);
                if (r == DialogResult.Yes) this.Close();
            }
        }
    }
```

Consideraciones al agregar el código de la aplicación:

- **Determinar el total de personas aseguradas**

  Para contabilizar el total de personas aseguradas, se debe recorrer los datos registrados en el control ListView. Para comenzar, se debe recorrer el total de registros que presenta el ListView con la sentencia "for(int i=0;i<lvProforma.Items.Count;i++)"; todos los registros en un ListView inician en la posición 0es asi que la estructura for inicializa en cero su ciclo de repeticiones y el límite lo determina la función Items. Count-1. Otras opciones que se pueden presentar para el mismo juego de sentencias son de la siguiente manera:

```
int totalAsegurados = 0;
int n = lvProforma.Items.Count;
  for (int i = 0; i < n; i++)
  {
    int cantidad = int.Parse(lvProforma.Items[i].SubItems[1].Text);
    totalAsegurados += cantidad;
}
```

  La variable "n" tiene la misión de guardar el número de registros que tiene el control ListView y asi se evita que dicha sentencia se implemente dentro de la estructura for. Para obtener el acumulado de cantidades, se agrega una variable "cantidad" la cual tiene por misión obtener información desde la columna "1", que pertenece a las cantidades registradas, para que, en la siguiente instrucción, se acumule dentro de la variable totalAsegurados.

- **Determinar el monto acumulado desde el control ListView**

```
double total = 0;
 for (int i = 0; i < lvProforma.Items.Count; i++)
 {
   if (lvProforma.Items[i].SubItems[0].Text != "")
   total += double.Parse(lvProforma.Items[i].SubItems[2].Text);
 }
```

  Declare la variable "total", la cual es responsable de almacenar el monto total acumulado; recorra todas las filas del ListView con la estructura For, recuerde que el total de elementos de un ListView se determina mediante la sentencia lvProforma. Items.Count.

En lvProforma.Items[i].SubItems[2].Text, el numero dos representa la tercera columna del control ListView de la proforma el cual muestra los montos, no olvide que todos los valores mostrados en el control ListView son valores de tipo texto, por tanto, si lo necesita para acumular valores, primero debe parsearlo.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 4 : while - control de facturas

La empresa SYSTEM SAC ha desarrollado un software de ventas el cual se viene ofreciendo a todo el Perú en diferentes tipos de licencia. La venta del software es gestionada por vendedores capacitados los cuales necesitan registrar las ventas realizadas mediante una aplicación de control de registro de facturas, el cual solicita el nombre completo del vendedor, número de factura, tipo de licencia y unidades de software vendido.

Debemos tener en cuenta:

- Al iniciar la aplicación se debe mostrar la fecha de manera automática.
- El registro de la factura debe mostrarse en un cuadro de lista, con los datos propios de la venta además del monto subtotal por cada factura.
- Asimismo, deberá mostrarse el monto total registrado de las facturas (esto representa al monto acumulado de subtotales).
- Finalmente, mostrar las estadísticas del registro como el total de unidades y el monto total acumulado por cada tipo de licencia.
- El costo de venta de las licencias se muestra en la siguiente tabla:

Tipo de licencia	Monto de venta por unidad
Cobre	$700.00
Bronce	$900.00
Silver	$1400.00
Gold	$2500.00

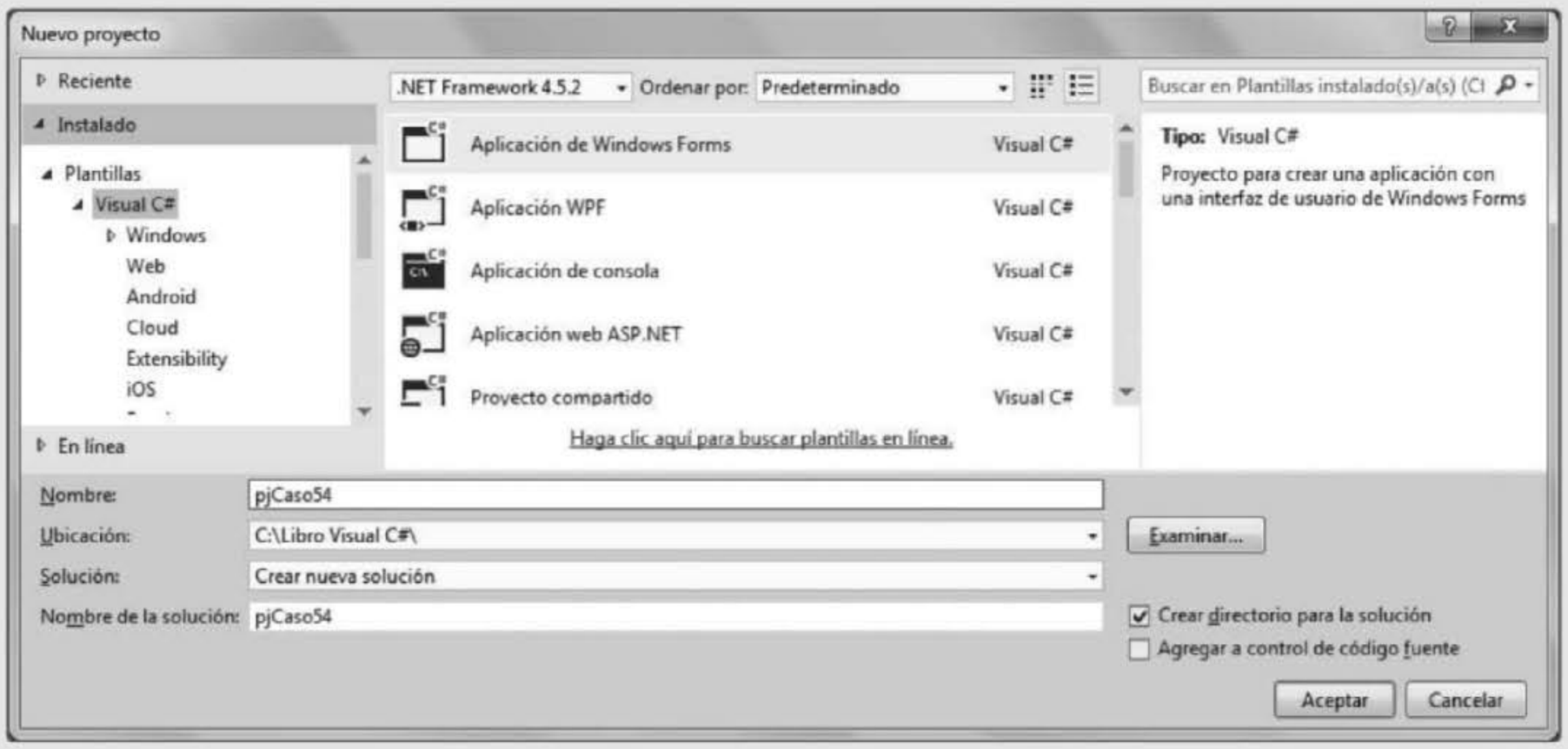
- El botón **REGISTRAR** ítem envía los datos a un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

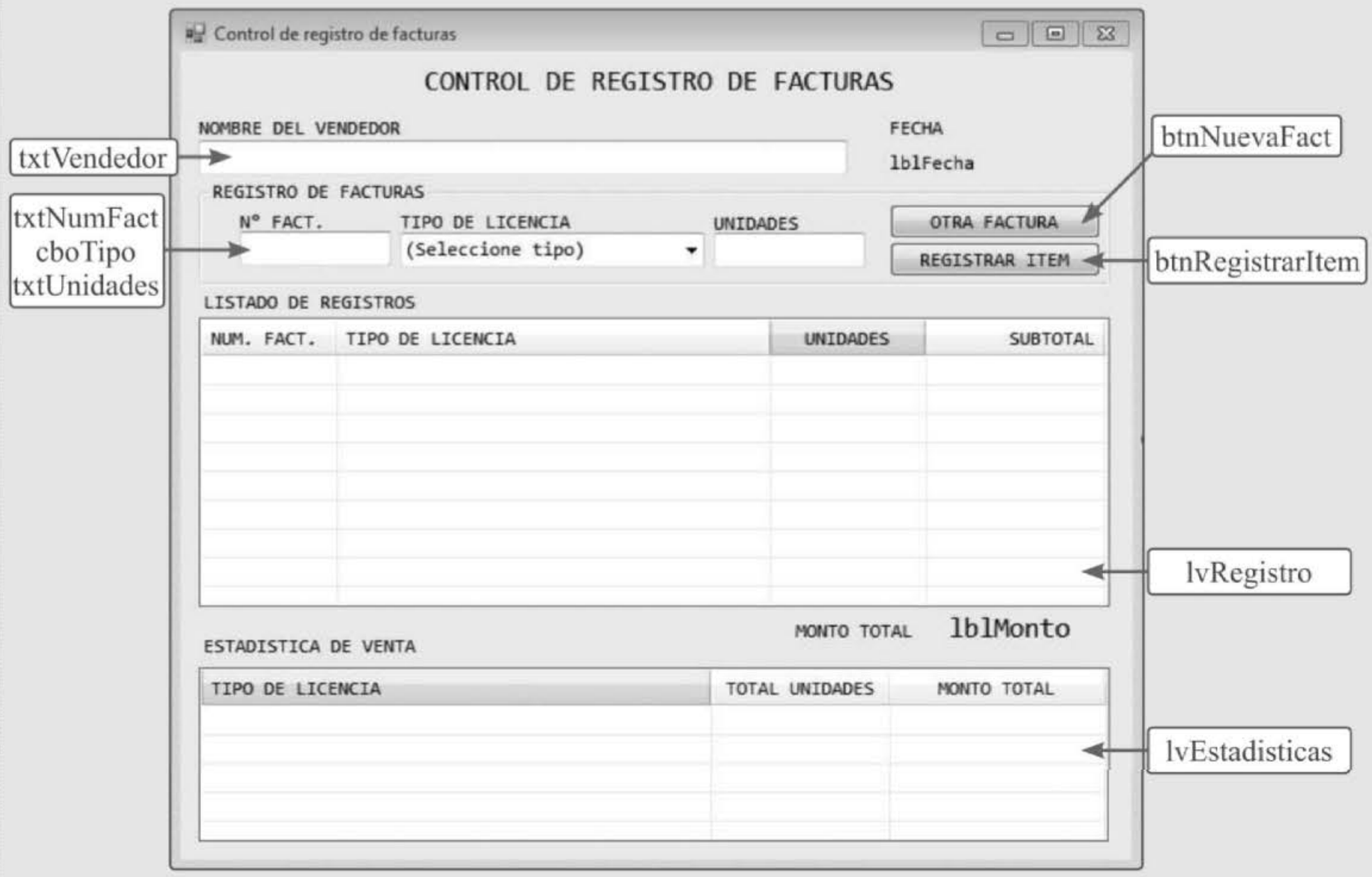
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso54**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmFacturas**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



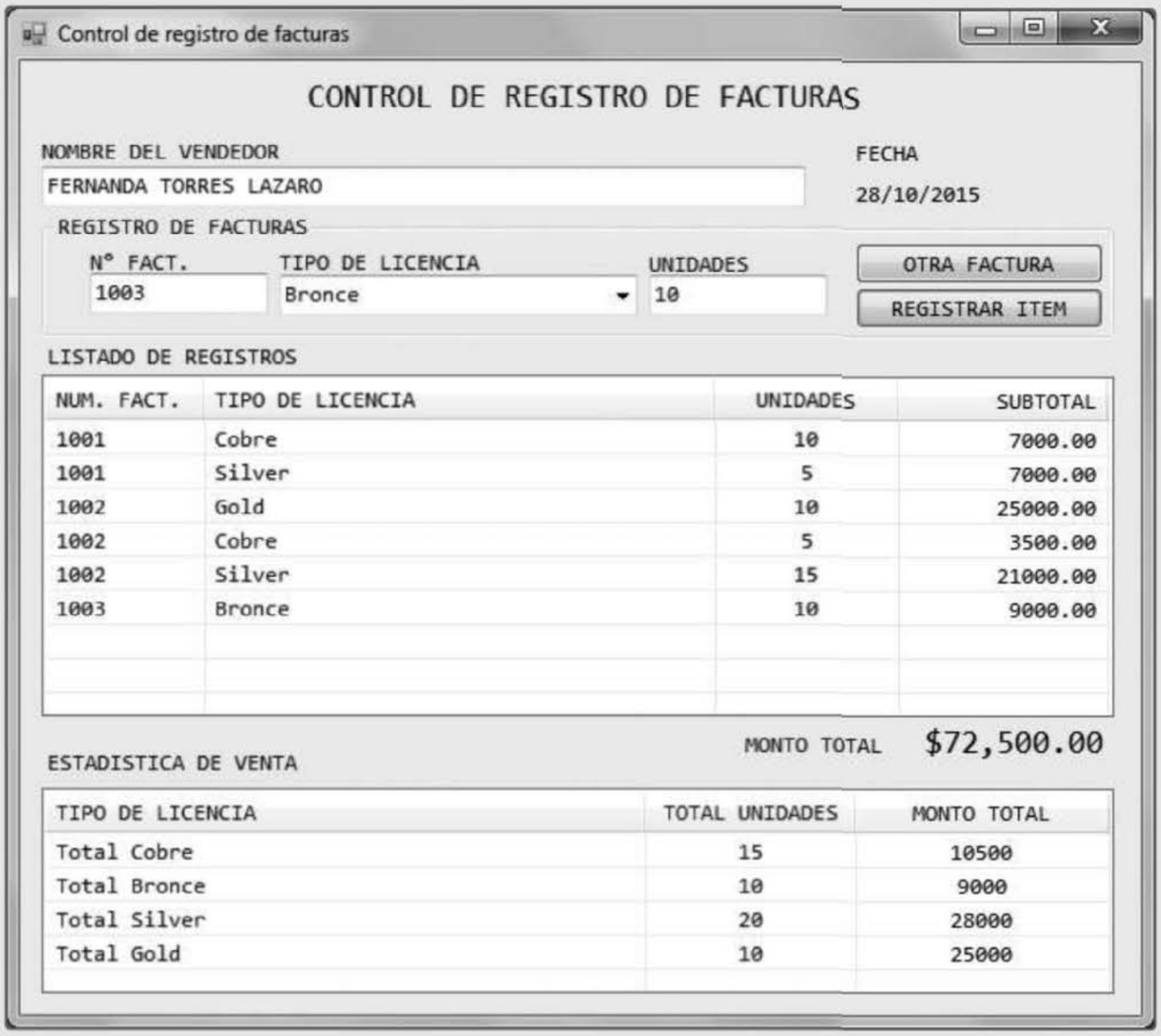
Estadísticas de los controles usados en el frmFacturas:

Clase	N.º elementos	Controles
Form	01	Form1
Label	11	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8, Label9, Label10, Label11
TextBox	03	Textbox1, Textbox2, Textbox3
ComboBox	01	ComboBox1
Button	02	Button1, Button2
ListView	02	ListView1, ListView2
GroupBox	01	GroupBox1
TOTAL	21	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmFacturas Control de registro de facturas
Label1	Text	CONTROL DE REGISTRO DE FACTURAS
Label2	Text	NOMBRE DEL VENDEDOR
Label3	Text	FECHA
Label4	(Name) Text	lblFecha lblFecha
Label5	Text	Nº FACT
Label6	Text	TIPO DE LICENCIA
Label7	Text	UNIDADES
Label8	Text	LISTADO DE REGISTROS
Label9	Text	ESTADISTICA DE VENTA
Label10	Text	MONTO TOTAL
Label11	(Name) Text	lblMonto lblMonto
Textbox1	(Name)	txtVendedor
Textbox2	(Name)	txtNumFact
Textbox3	(Name)	txtUnidades
ComboBox1	(Name)	cboTipo
Button1	(Name) Text	btnRegistrarItem REGISTRAR ITEM
Button2	(Name) Text	btnNuevaFact OTRA FACTURA
GroupBox1	Text	REGISTRO DE FACTURAS
ListView1	(Name) GridLines	lvRegistro True
ListView2	(Name) GridLines	lvEstadisticas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso54
{
    public partial class frmFacturas : Form
    {
        public frmFacturas()
        {
            InitializeComponent();
        }
```

```
private void frmFacturas_Load(object sender, EventArgs e)
{
    lblFecha.Text = DateTime.Now.ToShortDateString();
}

private void btnRegistrarItem_Click(object sender, EventArgs e)
{
    int i;

    //Capturando los datos
    int numFact = int.Parse(txtNumFact.Text);
    string licencia = cboTipo.Text;
    int unidades = int.Parse(txtUnidades.Text);

    //Determinar el precio de las licencias
    double precio = 0;
    switch (licencia)
    {
        case "Cobre": precio = 700; break;
        case "Bronce": precio = 900; break;
        case "Silver": precio = 1400; break;
        case "Gold": precio = 2500; break;
    }
    //Calculando el subtotal
    double subtotal = unidades * precio;

    //Enviando la informacion a la lista de registros
    ListViewItem fila = new ListViewItem(numFact.ToString());
    fila.SubItems.Add(licencia);
    fila.SubItems.Add(unidades.ToString());
    fila.SubItems.Add(subtotal.ToString("0.00"));
    lvRegistro.Items.Add(fila);

    //Calculando el monto total acumulado
    double mAcumulado = 0;
    i = 0;
    while(i < lvRegistro.Items.Count)
    {
mAcumulado += double.Parse(lvRegistro.Items[i].SubItems[3].Text);
        i++;
    }
    lblMonto.Text =mAcumulado.ToString("C");
    //Calculando las estadisticas
    int cCobre = 0, cBronce = 0, cSilver = 0, cGold = 0;
    double tCobre = 0, tBronce = 0, tSilver = 0, tGold = 0;

    i = 0;
    while(i < lvRegistro.Items.Count)
    {
        if (lvRegistro.Items[i].SubItems[1].Text == "Cobre")
        {
     cCobre += int.Parse(lvRegistro.Items[i].SubItems[2].Text);
   tCobre += double.Parse(lvRegistro.Items[i].SubItems[3].Text);
        }
```

```
            else if (lvRegistro.Items[i].SubItems[1].Text == "Bronce")
              {
            cBronce+=int.Parse(lvRegistro.Items[i].SubItems[2].Text);
          tBronce+=double.Parse(lvRegistro.Items[i].SubItems[3].Text);
              }
            else if (lvRegistro.Items[i].SubItems[1].Text == "Silver")
              {
            cSilver+=int.Parse(lvRegistro.Items[i].SubItems[2].Text);
          tSilver+=double.Parse(lvRegistro.Items[i].SubItems[3].Text);
              }
              else if (lvRegistro.Items[i].SubItems[1].Text == "Gold")
              {
               cGold+=int.Parse(lvRegistro.Items[i].SubItems[2].Text);
           tGold+=double.Parse(lvRegistro.Items[i].SubItems[3].Text);
              }
              i++;
          }

          //Imprimiendo las estadisticas
          lvEstadisticas.Items.Clear();
          string[] elementosFila = new string[3];
          ListViewItem row;

          elementosFila[0] = "Total Cobre";
          elementosFila[1] = cCobre.ToString();
          elementosFila[2] = tCobre.ToString();
          row = new ListViewItem(elementosFila);
          lvEstadisticas.Items.Add(row);
          elementosFila[0] = "Total Bronce";
          elementosFila[1] = cBronce.ToString();
          elementosFila[2] = tBronce.ToString();
          row = new ListViewItem(elementosFila);
          lvEstadisticas.Items.Add(row);

          elementosFila[0] = "Total Silver";
          elementosFila[1] = cSilver.ToString();
          elementosFila[2] = tSilver.ToString();
          row = new ListViewItem(elementosFila);
          lvEstadisticas.Items.Add(row);

          elementosFila[0] = "Total Gold";
          elementosFila[1] = cGold.ToString();
          elementosFila[2] = tGold.ToString();
          row = new ListViewItem(elementosFila);
          lvEstadisticas.Items.Add(row);
      }

          private void btnNuevaFact_Click(object sender, EventArgs e)
          {
              txtNumFact.Clear();
              txtUnidades.Clear();
              cboTipo.Text = "(Seleccione tipo)";
              txtNumFact.Focus();
          }
      }
}
```

Consideraciones al agregar el código de la aplicación:

- **Mostrar la fecha actual al iniciar la aplicación**

  Para mostrar la fecha al iniciar la aplicación, debe programar en el evento "load" del formulario. Por lo tanto, haga doble clic sobre el fondo del formulario y obtendrá acceso al load del formulario. Con respecto al código para mostrar la fecha actual, debe considerar el método ToShortDateString el cual muestra la fecha en formato corto, es decir, día, mes y año de dos cifras, el método "Now" invoca a la fecha y hora obtenida desde el sistema.

- **Calcular el monto total acumulado**

  Realice el cálculo del monto total acumulado usando la estructura repetida While. Si se sabe que el primer registro del ListView tiene la posición cero es a partir de esta posición que se debe capturar el valor de los subtotales. Estos valores se encuentran en la columna 3 de la lista, por lo tanto, se debe realizar dicha especificación para poder obtener sus valores.

```
double mAcumulado = 0;
          i = 0;
        while(i < lvRegistro.Items.Count)
        {
        mAcumulado += double.Parse(lvRegistro.Items[i].SubItems[3].Text);
            i++;
        }
  lblMonto.Text =mAcumulado.ToString("C");
```

Entonces, la variable i tendrá la misión de contabilizar los ciclos de repeticiones de la estructura While, preguntando en la condición si dicha variable es inferior al total de elementos registrados en el control ListView(lvRegistro). Mientras que esta condición sea verdadera, se acumularán los valores de la columna 3 (subtotal) en la variable mAcumulado. Al finalizar el ciclo de repeticiones, la variable mAcumulado será impresa en el objeto lblMonto. Recuerde que el parámetro C asignado al ToString permite mostrar el resultado con el símbolo de la moneda según la configuración regional de su sistema operativo.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 5 : do while - control de registro alquiler de habitaciones

Una de las empresas hoteleras que operan en el centro de la ciudad de Lima, necesita una aplicación que permita controlar el registro de los alquileres de sus habitaciones que se realiza diariamente; esto debido a la gran demanda de turistas en el mes actual. La aplicación deberá mostrar un formulario de registro de alquileres con la fecha actual y la hora en forma digital. Además de poder seleccionar el tipo de habitación y los días que el usuario solicito en alquiler, la aplicación deberá registrar todos los alquileres mostrándolos en una lista con los datos del tipo, costo por día de la habitación, días alquilados, subtotal, descuento y pago neto que realizan sus clientes. Al final, se debe mostrar una estadística de la cantidad de días y neto acumulados por cada tipo de habitación.

Se tiene en cuenta lo siguiente:

- Al iniciar la aplicación, se debe mostrar la fecha y la hora digitalizada de manera automática.
- El precio de las habitaciones varía según el tipo, tal como se muestra en la siguiente tabla:

Tipo de habitación	Costo por día
Simple	$ 30.00
Doble	$ 50.00
Matrimonial	$ 120.00
Presidencial	$ 300.00

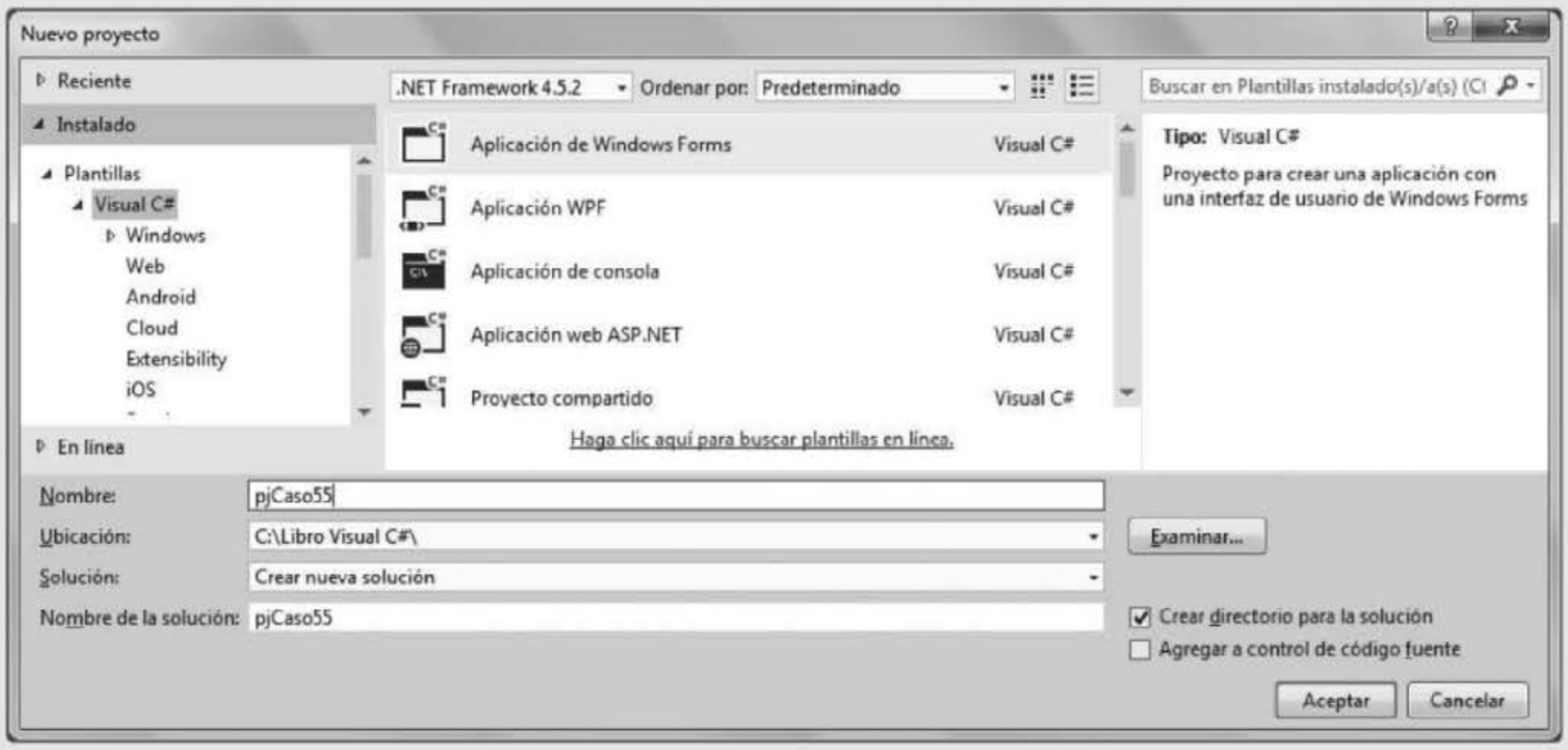
- El botón **REGISTRAR** envía los datos de la habitación y la cantidad de días alquilados a un control ListView.
- Finalmente, se deben mostrar las estadísticas de forma automática al registrar una habitación, como el total de habitaciones simples, dobles, matrimoniales y presidenciales, además de los montos recaudados por cada tipo de habitación en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Selección **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso55**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmAlquiler**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



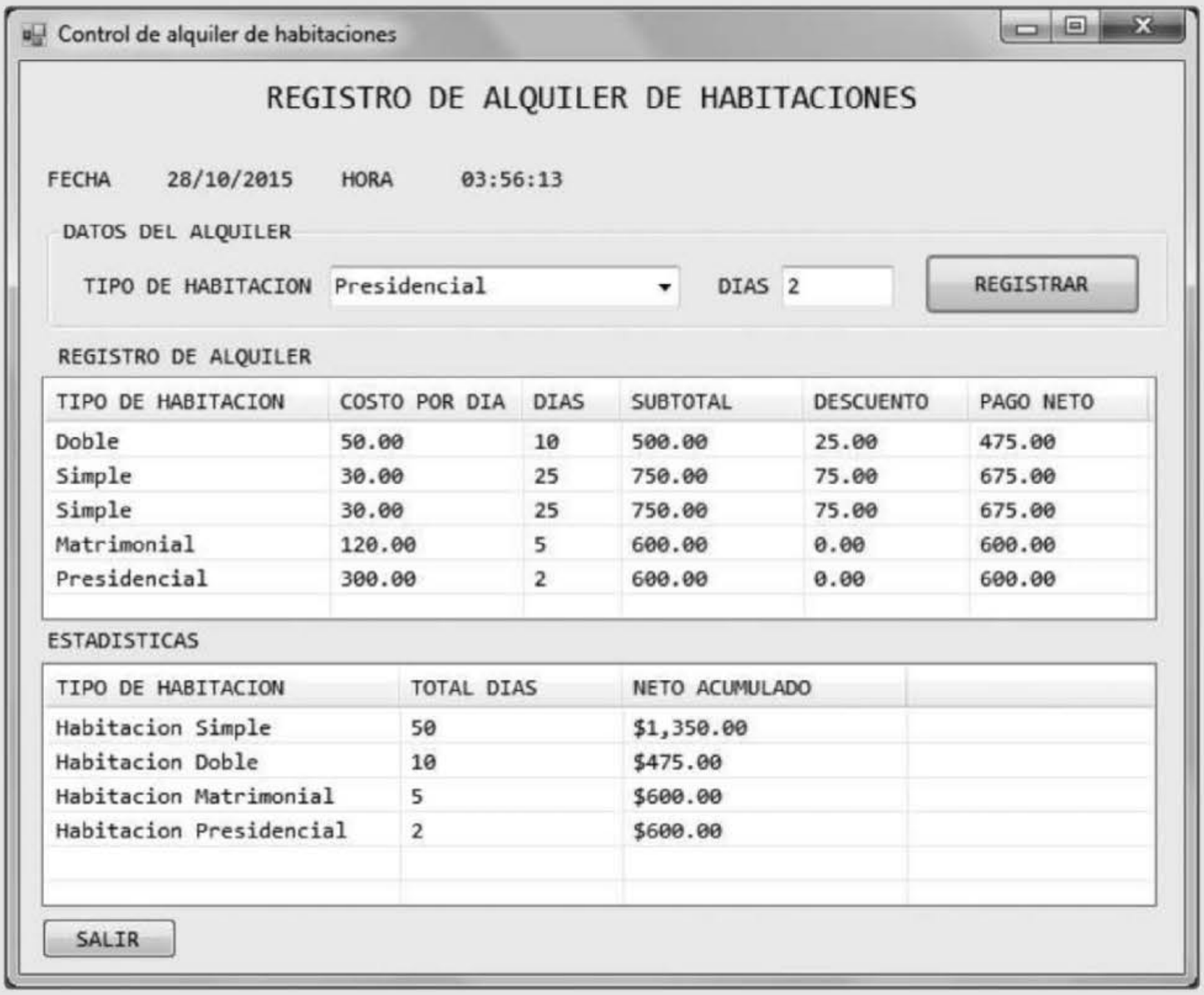
Estadísticas de los controles usados en el frmAlquiler:

Clase	N.º elementos	Controles
Form	01	Form1
Label	09	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8, Label9
TextBox	01	Textbox1
ComboBox	01	ComboBox1
Button	02	Button1, Button2
ListView	02	ListView1, ListView2
GroupBox	01	GroupBox1
TOTAL	17	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmAlquiler Registro de alquiler de habitaciones
Label1	Text	REGISTRO DE ALQUILER DE HABITACIONES
Label2	Text	FECHA
Label3	(Name) Text	lblFecha lblFecha
Label4	Text	HORA
Label5	(Name) Text	lblHora lblHora
Label6	Text	TIPO DE HABITACION
Label7	Text	DIAS
Label8	Text	REGISTRO DE ALQUILER
Label9	Text	ESTADISTICAS
GroupBox1	Text	DATOS DEL ALQUILER
Textbox1	(Name)	txtDias
ComboBox1	(Name)	cboTipo
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnSalir SALIR
ListView1	(Name) GridLines	lvAlquiler True
ListView2	(Name) GridLines	lvEstadisticas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso55
{
    public partial class frmAlquiler : Form
    {
        public frmAlquiler()
        {
            InitializeComponent();
            tHora.Enabled = true;
        }
```

```
private void frmAlquiler_Load(object sender, EventArgs e)
{
    lblFecha.Text = DateTime.Now.ToShortDateString();
}

private void tHora_Tick(object sender, EventArgs e)
{
    lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
}

private void btnRegistrar_Click(object sender, EventArgs e)
{
    //Capturar los datos
    string habitacion = cboTipo.Text;
    int dias = int.Parse(txtDias.Text);

    //Asignacion de costo por habitacion
    double costo = 0;
    switch (habitacion)
    {
        case "Simple": costo = 30; break;
        case "Doble": costo = 50; break;
        case "Matrimonial": costo = 120; break;
        case "Presidencial": costo = 300; break;
    }

    //Realizando los calculos
    double subtotal = costo * dias;

    double descuento = 0;
    if (dias <= 5)
        descuento = 0;
    else if (dias > 5 && dias <= 10)
        descuento = 0.05 * subtotal;
    else if (dias > 10 && dias <= 30)
        descuento = 0.1 * subtotal;
    else if (dias > 30)
        descuento = 0.2 * subtotal;

    double neto = subtotal - descuento;

    //Enviar los datos al registro de alquiler
    ListViewItem fila = new ListViewItem(habitacion);
    fila.SubItems.Add(costo.ToString("0.00"));
    fila.SubItems.Add(dias.ToString());
    fila.SubItems.Add(subtotal.ToString("0.00"));
    fila.SubItems.Add(descuento.ToString("0.00"));
    fila.SubItems.Add(neto.ToString("0.00"));
    lvAlquiler.Items.Add(fila);
```

```
//Determinar las estadisticas
int i = 0;
int cSimple = 0, cDoble = 0, cMatrimonial = 0, cPresidencial = 0;
double tSimple = 0, tDoble = 0, tMatrimonial = 0, tPresidencial = 0;

 do {
     string tipo = lvAlquiler.Items[i].SubItems[0].Text;
     switch (tipo)
     {
        case "Simple":
         cSimple+=int.Parse(lvAlquiler.Items[i].SubItems[2].Text);
         tSimple+=double.Parse(lvAlquiler.Items[i].SubItems[5].Text);
        break;
        case "Doble":
         cDoble+=int.Parse(lvAlquiler.Items[i].SubItems[2].Text);
         tDoble+= double.Parse(lvAlquiler.Items[i].SubItems[5].Text);
        break;

        case "Matrimonial":
         cMatrimonial+=int.Parse(lvAlquiler.Items[i].SubItems[2].Text);
    tMatrimonial+=double.Parse(lvAlquiler.Items[i].SubItems[5].Text);
         break;
       case "Presidencial":
       cPresidencial+=int.Parse(lvAlquiler.Items[i].SubItems[2].Text);
    tPresidencial+=double.Parse(lvAlquiler.Items[i].SubItems[5].Text);
       break;
        }
        i++;
    } while (i<lvAlquiler.Items.Count);

    //Imprimiendo las estadisticas
    lvEstadisticas.Items.Clear();
    string[] elementosFila = new string[3];
    ListViewItem row;

    elementosFila[0] = "Habitacion Simple";
    elementosFila[1] = cSimple.ToString();
    elementosFila[2] = tSimple.ToString("C");
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);

    elementosFila[0] = "Habitacion Doble";
    elementosFila[1] = cDoble.ToString();
    elementosFila[2] = tDoble.ToString("C");
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);
```

```
            elementosFila[0] = "Habitacion Matrimonial";
            elementosFila[1] = cMatrimonial.ToString();
            elementosFila[2] = tMatrimonial.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Habitacion Presidencial";
            elementosFila[1] = cPresidencial.ToString();
            elementosFila[2] = tPresidencial.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                "Seguros",
                                MessageBoxButtons.YesNo,
                                MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Mostrar la hora digitalizada**

  Busque el control Timer desde el cuadro de herramientas de Visual, incorpore uno de ellos al formulario y en las propiedades cámbiele el nombre a "tHora". Para que se pueda habilitar la digitalización, debe usar el método constructor del formulario, tal como se muestra en el siguiente código:

```
public frmAlquiler()
        {
            InitializeComponent();
            tHora.Enabled = true;
        }
```

True, permitirá habilitar la digitalización de la hora al ejecutar la aplicación. Además tenga en cuenta que la hora actual debe ser asignada al control "lblHora" para ello, debe programar dentro del evento Tick del control tHora (Timer), para esto, solo basta hacer doble clic para agregar el siguiente código:

```
private void tHora_Tick(object sender, EventArgs e)
        {
            lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
        }
```

Los parámetros "hh:mm:ss" especifican que la hora se muestre en dicho formato; de otra manera se mostraran con formato de milisegundos.

- **Determinar las acumulaciones por tipo de habitación**

  En este proceso se usa la estructura do..while; esta se encarga de recorrer por todos los registros. Por tal razón, por cada ciclo de repeticiones, se captura el tipo de habitación en la variable tipo, para que luego se compare con la estructura switch. Una vez comparados los tipos de habitaciones, se acumula en sus respectivas variables acumuladores. No debe olvidar declarar estas variables antes de entrar al ciclo do.. while.

```
case "Simple":
            cSimple+=int.Parse(lvAlquiler.Items[i].SubItems[2].Text);
            tSimple+=double.Parse(lvAlquiler.Items[i].SubItems[5].Text);
            break;
```

Analice solo cuando el tipo de habitación sea "simple". La variable "cSimple" permite acumular el total de días del tipo de habitación simple, el cual se encuentra en la columna 2 de la lista. Mientras que la variable "tSimple" permite acumular los pagos netos, los cuales se ubican en la posición 5 de la columna. De la misma manera, se comparan a los demás tipos de habitaciones.

6. Finalmente, presione **F5** para probar la aplicación.

Caso desarrollado 6 : integración - control de venta de boletos

La empresa Cine-Word es una cadena global de multicines líder en el mercado de exhibición cinematográfica el cual ha decidido instalarse en la ciudad de Lima. Ha empezado por ofrecer la venta de sus boletos de cine de forma corporativa. Para lo cual, se necesita una aplicación que según la edad del cliente, se le asigne una categoría y, a partir de allí, se apliquen los precios y descuentos según la siguiente tabla:

Se tiene en cuenta lo siguiente:

- Al iniciar la aplicación, el cuadro combinado de las edades deben encontrarse precargados.
- El precio de las habitaciones varía según el tipo, tal como se muestra en la siguiente tabla:

Edad	Categoría	Descuento	Precio unitario
03 - 12	Niño	20 %	$ 10.00
13 - 17	Joven I	10 %	$ 15.00
18 - 50	Joven II	5 %	$ 25.00
51 - 65	Adulto I	10 %	$ 15.00
>65	Adulto II	20 %	$ 10.00

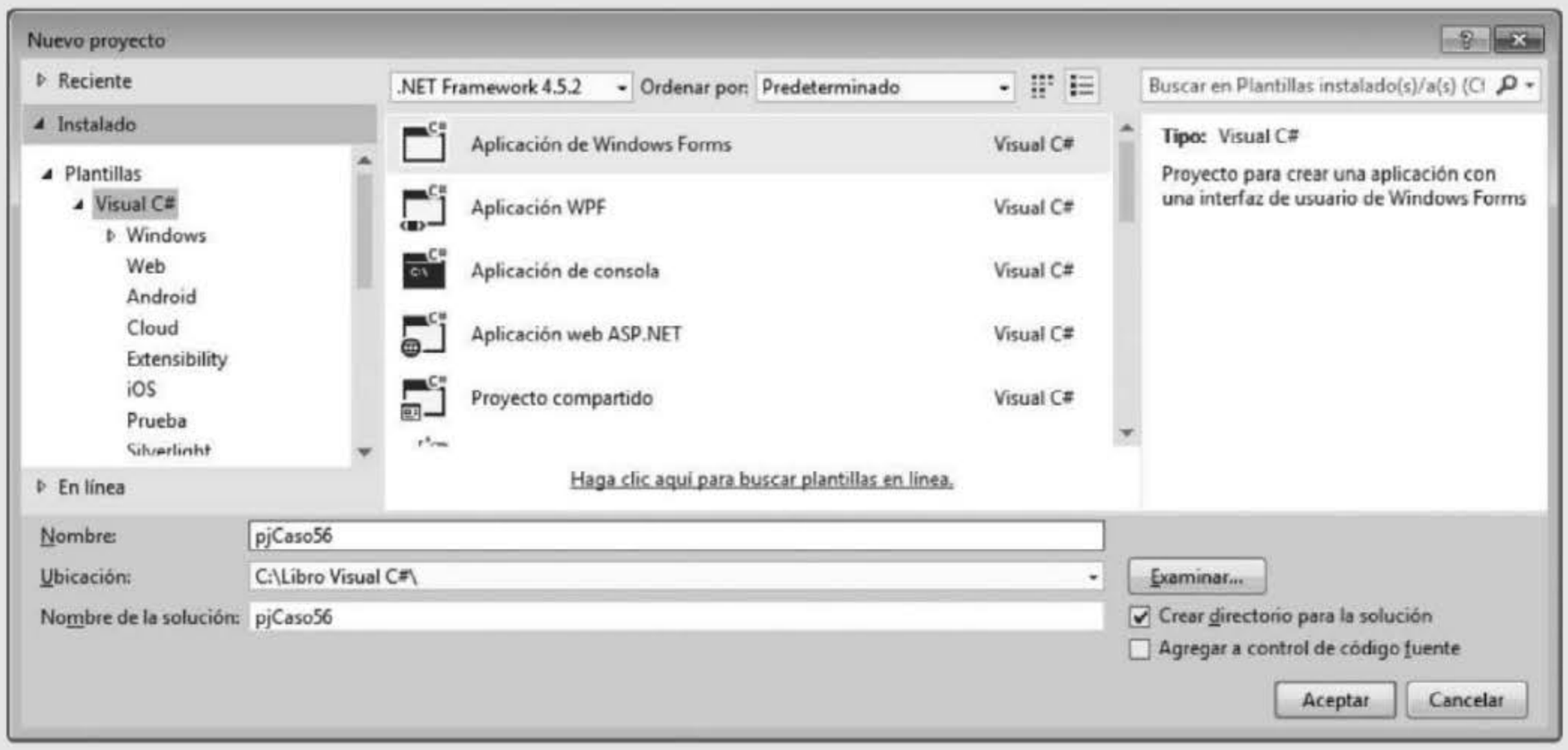
- Al seleccionar una edad desde el cuadro combinado, debe mostrar automáticamente el precio y la categoría asignada.
- El botón **REGISTRAR** envía los datos de la venta como la categoría, precio unitario, cantidad de boletos, subtotal (precio por la cantidad seleccionada), descuento e importe a pagar en un control ListView.
- Finalmente, mediante un botón, se deben mostrar las estadísticas el monto total acumulado de los subtotales, monto total que la empresa deja de percibir por aplicar descuento, los montos acumulados por cada tipo de categoría en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

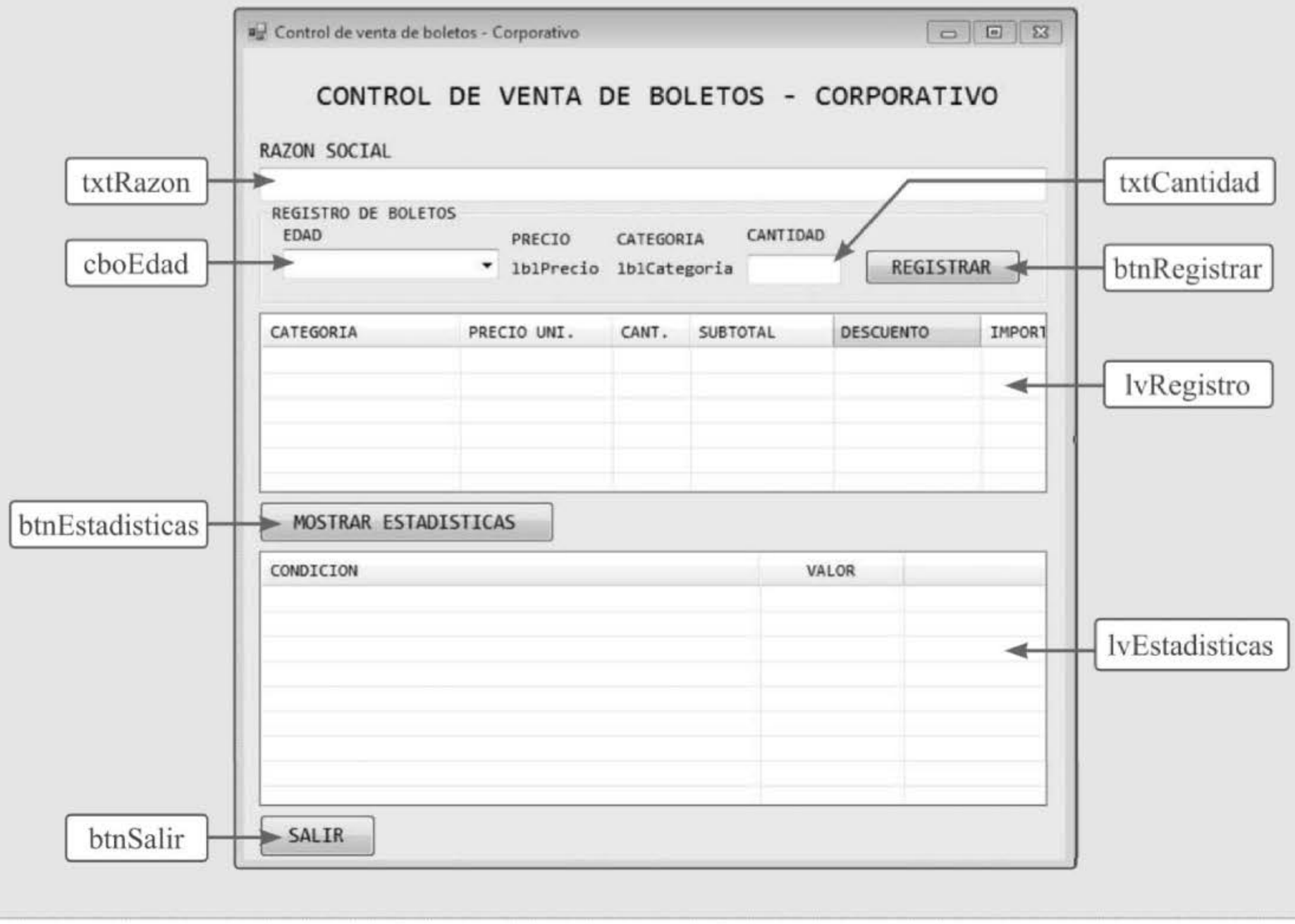
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Selección **Aplicación de Windows Form**.

- Asigne el nombre **pjCaso56**, tal como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs**, desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo**, asigne el nombre **frmCine**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmCine:

Clase	N.º elementos	Controles
Form	01	Form1
Label	08	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8
TextBox	02	Textbox1, Textbox2
ComboBox	01	ComboBox1
Button	02	Button1, Button2
ListView	02	ListView1, ListView2
GroupBox	01	GroupBox1
TOTAL	17	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmCine Control de venta de boletas - Corporativo
Label1	Text	CONTROL DE VENTA DE BOLETOS - CORPORATIVO
Label2	Text	RAZON SOCIAL
Label3	Text	EDAD
Label4	Text	PRECIO
Label5	(Name) Text	lblPrecio lblPrecio
Label6	Text	CATEGORIA
Label7	(Name) Text	lblCategoria lblCategoria
Label8	Text	CANTIDAD
Textbox1	(Name)	txtRazon
Textbox2	(Name)	txtCantidad
GroupBox1	Text	REGISTRO DE BOLETOS
ComboBox1	(Name)	cboEdad
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnEstadisticas MOSTRAR ESTADÍSTICAS
ListView1	(Name) GridLines	lvRegistro True
ListView2	(Name) GridLines	lvEstadisticas True

Finalmente, el formulario debe tener el aspecto siguiente:



5. El código completo de la aplicación es:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso56
{
    public partial class frmCine : Form
    {
        double precio = 0;
        string categoria ="";
        public frmCine()
```

```
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los datos necesarios
            categoria = lblCategoria.Text;
            int cantidad = int.Parse(txtCantidad.Text);

            //Realizando los calculos
            double subtotal = precio * cantidad;
            double descuento = 0;
            switch (categoria)
            {
                case "Niño": descuento = 20.0 / 100 * subtotal; break;
                case "Joven I": descuento = 10.0 / 100 * subtotal; break;
                case "Joven II": descuento = 5.0 / 100 * subtotal; break;
                case "Adulto I": descuento = 10.0 / 100 * subtotal; break;
                case "Adulto II": descuento = 20.0 / 100 * subtotal; break;
            }
            double importe = subtotal - descuento;

            //Imprimir en la lista
            ListViewItem fila = new ListViewItem(categoria);
            fila.SubItems.Add(precio.ToString("0.00"));
            fila.SubItems.Add(cantidad.ToString());
            fila.SubItems.Add(subtotal.ToString("0.00"));
            fila.SubItems.Add(descuento.ToString("0.00"));
            fila.SubItems.Add(importe.ToString("0.00"));
            lvRegistro.Items.Add(fila);

            lvEstadisticas.Items.Clear();
        }

        private void cboEdad_SelectedIndexChanged(object sender, EventArgs e)
        {
            //Capturando la edad seleccionada
            int edad = cboEdad.SelectedIndex;

            //Asignando el precio y categoria segun la edad seleccionada
            switch (edad)
            {
                case 0: precio = 10; categoria = "Niño"; break;
                case 1: precio = 15; categoria = "Joven I"; break;
                case 2: precio = 25; categoria = "Joven II"; break;
                case 3: precio = 15; categoria = "Adulto I"; break;
                case 4: precio = 10; categoria = "Adulto II"; break;
            }
```

```
        //Mostrando el precio y la categoria
        lblPrecio.Text = precio.ToString("C");
        lblCategoria.Text = categoria;
    }

    private void btnEstadisticas_Click(object sender, EventArgs e)
    {
        lvEstadisticas.Items.Clear();

        //Hallar el monto total sin descuento
        double tSubtotal = 0;
        int i;
        for (i = 0; i < lvRegistro.Items.Count; i++)
        {
            tSubtotal += double.Parse(lvRegistro.Items[i].SubItems[3].Text);
        }

        //Hallar el monto total que la empresa no percibe
        //los descuentos realizados
        double tDescuento = 0;
        i = 0;
        while (i < lvRegistro.Items.Count)
        {
            tDescuento += double.Parse(lvRegistro.Items[i].SubItems[4].Text);
            i++;
        }

        //Hallar el monto total acumulado por categoria
        double aNiño=0, aJovenI=0, aJovenII=0, aAdultoI=0, aAdultoII=0;
        i = 0;
        do
        {
            string categoria = lvRegistro.Items[i].SubItems[0].Text;
            switch (categoria)
            {
              case "Niño":
                aNiño+=double.Parse(lvRegistro.Items[i].SubItems[5].Text);
                break;
              case "Joven I":
                aJovenI+=double.Parse(lvRegistro.Items[i].SubItems[5].Text);
                break;
              case "Joven II":
                aJovenII+=double.Parse(lvRegistro.Items[i].SubItems[5].Text);
                break;
              case "Adulto I":
                aAdultoI+=double.Parse(lvRegistro.Items[i].SubItems[5].Text);
                break;
               case "Adulto II":
                aAdultoII+=double.Parse(lvRegistro.Items[i].SubItems[5].Text);
                break;
            }
            i++;
        } while (i < lvRegistro.Items.Count);
```

```
            //Imprimiendo las estadisticas
            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[2];
            ListViewItem row;

            elementosFila[0] = "Monto total sin descuento";
            elementosFila[1] = tSubtotal.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto total que la empresa no percibe";
            elementosFila[1] = tDescuento.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto acumulado por categoria Niño";
            elementosFila[1] = aNiño.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto acumulado por categoria Joven I";
            elementosFila[1] = aJovenI.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto acumulado por categoria Joven II";
            elementosFila[1] = aJovenII.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto acumulado por categoria Adulto I";
            elementosFila[1] = aAdultoI.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

           elementosFila[0] = "Monto acumulado por categoria Adulto II";
            elementosFila[1] = aAdultoII.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }
    }
}
```

6. Finalmente, presione **F5** para probar la aplicación.

Para recordar

- Un contador es una variable de tipo entero cuyo valor incrementa o decrementa en un valor constante, esto se produce solo cuando ocurre algún suceso; es decir, el uso del contador se debe a una condición en la aplicación.
- La inicialización de un contador se realiza principalmente cuando es declarada de forma local, si el contador es declarado como global, automáticamente recibe un valor inicial dependiendo del tipo de datos declarado.
- Un acumulador es una variable cuyo valor se incrementa o decrementa en un valor variable. Un acumulador suele utilizarse para acumular resultados producidos en las iteraciones de un ciclo de repeticiones.
- El ciclo While permite ejecutar un conjunto de instrucciones un número determinado de veces; mientras que la condición especificada en la instrucción While sea verdadera, en caso sea falsa, la condición el ciclo de repeticiones iniciada por While finaliza.
- El ciclo Do While permite repetir un conjunto de instrucciones un número indefinido de veces, todo depende de la condición establecida.
- El ciclo For permite repetir un conjunto de instrucciones un número definido de veces su implementación es la más usada por los desarrolladores.

Preguntas frecuentes

➢ **¿Qué tipo de estructura repetitiva se debe usar en la aplicación?**

Todas las estructuras repetitivas cumplen el mismo objetivo de recorrer por un ciclo de repeticiones que pueden ser controladas por la aplicación y, así, realizar tareas específicas; lo que varía entre ellos es la forma de implementarlo. La sugerencia es optar siempre la estructura que sea más cómoda de usar, es decir, podría optar por la sentencia For, por ser la más simple y sencilla de implementar.

➢ **¿Cuál es la diferencia entre un contador y un acumulador?**

Los contadores tienen la misión de realizar conteos, normalmente, de uno en uno. Su tarea principal es determinar un número determinado de elementos. Mientras que los acumuladores suman valores de forma sucesiva, de tal forma que su valor no resulta constante sino, más bien, controlado por un valor que es variable por cada ciclo de repeticiones que le dan una estructura repetitiva. Normalmente, toda variable declarada como contador o acumulador siempre debe declararse como variable global.

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
- Ámbito de las variables
- Programación modular
- Método sin valor de retorno sin parámatros
- Método sin valor de retorno con parámetros
- Método con valor de retorno sin parámetros
- Método con valor de retorno con parámetros
- Expresiones lambda
- Casos desarrollados
  - o Caso desarrollado 1: control de registro de llamadas - método sin valor de retorno sin parámetros
  - o Caso desarrollado 2: control de registro de llamadas - método sin valor de retorno con parámetros
  - o Caso desarrollado 3: control de registro de llamadas - método con valor de retorno sin parámetros
  - o Caso desarrollado 4: control de registro de llamadas - método con valor de retorno con parámetros
  - o Caso desarrollado 5: integración de métodos - control de pago de empleados
  - o Caso desarrollado 6: lambda - control de registro de libros
- Para recordar
- Preguntas frecuentes

Introducción

La programación actual tiene como método principal para resolver problemas la subdivisión en porciones pequeñas de código, llamado subprogramas. Se ha de comprender que para resolver un problema particular se debe encontrar todos los procesos necesarios para dicha solución, solo así se encuentra lo modular de la aplicación.

La idea principal es que una aplicación use dichos bloques de código para un propósito específico sin preocuparse de su implementación, es como la caja negra del avión: entra y sale información pero no se sabe cómo lo hace.

Esto no deja de lado despreocuparse de la implementación de los módulos de una aplicación, la labor de los programadores será codificar correctamente los módulos de manera que la aplicación use dichos módulos y le devuelva la respuesta esperada.

Imagine que necesita una aplicación que permita calcular el monto neto y el descuento que debe recibir un empleado por laborar un mes en su centro de trabajo. Para ello, debe registrar las horas que trabaja y el costo por hora. En este proceso de negocio los posibles módulos pueden ser:

Módulos	Descripción
getEmpleado	Permite capturar el nombre del empleado.
getHoras	Permite capturar las horas trabajadas por el empleado.
getPagoHora	Permite capturar el pago por hora del empleado.
calculaImporte	Permite calcular el importe de pago del empleado el cual resulta del producto entre las horas y el pago por hora.
calculaDescuento	Permite calcular el monto de descuento aplicado al importe generado al empleado.
calculaNeto	Permite calcular el monto neto a partir del importe y el descuento del empleado.

Un módulo dentro de la programación ofrece un conjunto de servicios a toda la aplicación, es decir, se puede interactuar entre todos sus procesos en cualquier tiempo. Asimismo, promueve la descomposición de un programa general en un subprograma independiente que visualmente está separado pero internamente sigue unido.

Por otro lado, los módulos son también conocidos como rutina, subprograma o subrutina, el cual permite resolver un proceso de negocio mediante la separación de código.

6.1 Ámbito de las variables

Una variable resulta ser uno de los elementos o miembros más importante para una aplicación C# ya que usa la política de tipado donde se menciona que todo lo que se usa dentro de la aplicación debe ser declarado obligatoriamente.

Si se centra en el proyecto de aplicaciones se debe considerar que las variables son declaradas en dos lugares específicos, se explican en la siguiente tabla:

Ámbito	Descripción
Local	Las variables declaradas como locales solo pueden ser usadas dentro del proceso declarado. Una vez finalizado el proceso la variable local se destruye.
Global	Las variables declaradas como globales pueden ser accedidas desde cualquier proceso; ya que al ser consideradas globales su valor podrá ser usado en cualquier parte de la aplicación.

La declaración entre las variables locales y globales no presenta diferencias, por lo tanto, la declaración se representa de la siguiente manera:

```
Tipo_Datos nombreVariable;
```

➢ **Características de las variables locales:**

- Solo son utilizadas dentro del contexto donde se declaran.
- El valor asignado solo puede ser modificado dentro de su contexto.
- Cuando se aplique contadores o acumuladores estas variables deben tener un valor inicial obligatoriamente; si es numérico entero inicialícelo con cero (0), si es una cadena de caracteres use doble comilla y si fuera un valor real inicialícelo con cero punto cero (0.0).

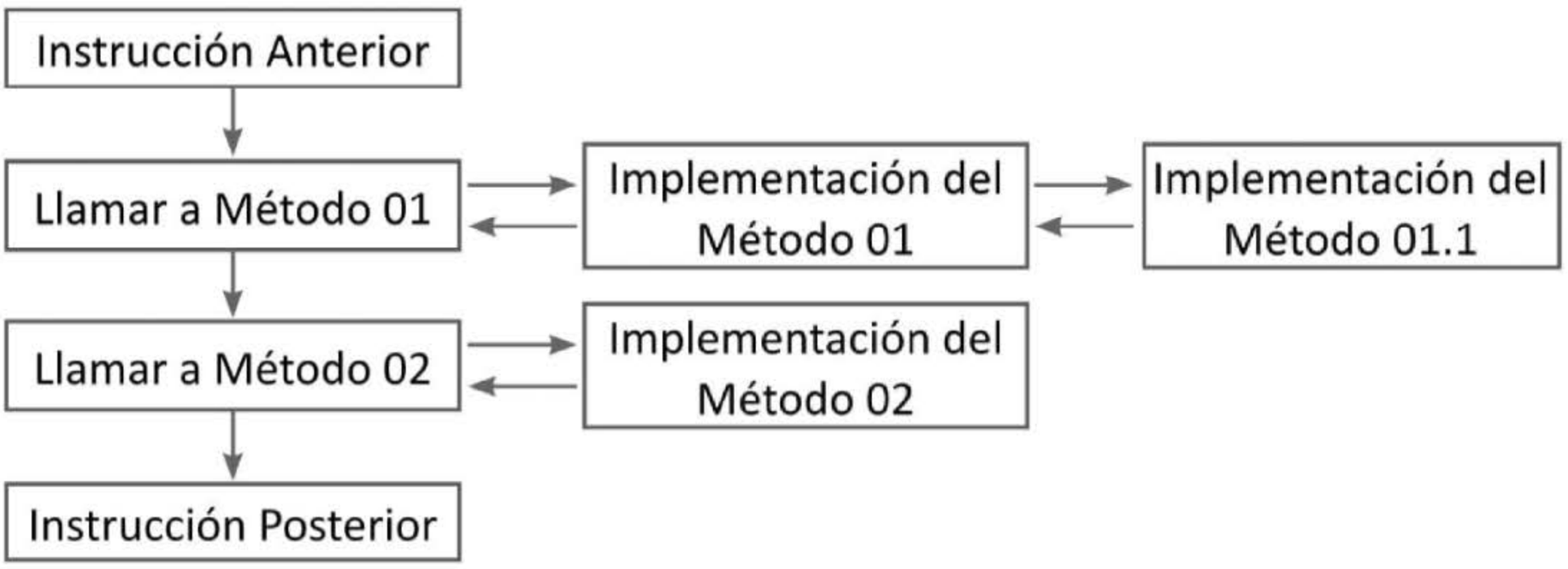
➢ **Características de las variables globales:**

- Pueden ser usadas en todos los procesos de la aplicación pero no fuera de ella, es decir, en otra aplicación.
- El valor asignado a una variable global puede ser modificado en cualquier momento.
- Toda variable global tiene asignada un valor inicial automáticamente, es decir, no será necesario inicializarlo.
- Solo cuando finaliza la aplicación las variables globales perderán su valor, es decir, cuando la aplicación se encuentra ejecutada la variable global siempre tendrá un valor.

6.2 Programación modular

Se denomina modular a la acción de descomponer una aplicación en porciones pequeñas llamadas módulos. Estos contienen un conjunto de sentencias que tienen un sentido lógico para la aplicación. Se recomienda siempre analizar cuántos módulos puede tener una aplicación antes de comenzar a implementarlos.

La programación modular es parte importante de la programación orientada a objetos, ya que la descomposición de una aplicación en porciones pequeñas se viene aplicando desde el hecho de que una aplicación Visual C# ejecuta toda su aplicación en módulos que son llamados métodos, por ser un lenguaje orientado a objetos y porque la aplicación se compone de clases. Veáse una gráfica de invocaciones entre métodos.



Finalmente, se estudiarán cuatro tipos de métodos que permitirán mejorar la implementación de las aplicaciones, estos se muestran a continuación:

- Método sin valor de retorno sin parámetros
- Método sin valor de retorno con parámetros
- Método con valor de retorno sin parámetros
- Método con valor de retorno con parámetros

6.3 Método sin valor de retorno sin parámatros

Tiene la finalidad de realizar una determinada tarea sin devolver algún valor a quien lo invoca asimismo no recibe ninguna variable como parámetro.

Si se usa este tipo de método se debe tener en claro el uso de las variables locales y globales, ya que al no recibir parámetros estos serán obtenidos desde las variables globales.

Formato:

```
void metodo(){
    //Declaraciones de variables locales
    //Implementación del método
}
```

Donde:

- **Void:** Es la palabra reservada que indica que el método no devolverá valor alguno, es decir, "método sin valor de retorno".
- **metodo():** Es el nombre que se le asignará al método, se recomienda que siempre inicie con un verbo para así diferenciarlo de las variables locales y globales. Note que entre los paréntesis que presenta no debe especificar nada, por tratarse de una especificación sin parámetros.

Véanse algunas implementaciones básicas para el uso de los métodos sin valor de retorno sin parámetros:

- Implementar el cálculo del promedio de cuatro notas declaradas como global de un determinado alumno.

```
public partial class frmPrueba: Form
{
   //Seccion de declaracion de variables GLOBALES
   int nota1, nota2, nota3, nota4;
   double promedio;

   public frmPrueba()
   {
      InitializeComponent();
   }

   //Implementacion del metodo
   void calculaPromedio()
   {
     Promedio = (nota1+nota2+nota3+nota4)/4.0;
   }
}
```

- Implementar el conteo de valores según la categoría (A, B, C) de un trabajador.

```
public partial class frmPrueba: Form
{
   //Seccion de declaracion de variables GLOBALES
   String categoria;
   int cA, cB, cC;

   public frmPrueba()
   {
      InitializeComponent();
   }

   //Implementacion del metodo
   void asignaConteos()
   {
     switch(categoria){
            case "A": cA++; break;
            case "B": cB++; break;
            case "C": cC++; break;
}
   }
}
```

- Implementar la impresión de los valores de una aplicación de pagos, el cual cuenta con salario, descuento y neto.

```
public partial class frmPrueba: Form
{
   //Seccion de declaracion de variables GLOBALES
   String empleado;
   double salario, descuento, neto;

   public frmPrueba()
   {
      InitializeComponent();
   }

   //Implementacion del metodo
   void imprimeValores()
   {
     ListViewItem fila=new ListViewItem(empleado)
     fila.subItems(salario.ToString("C"));
     fila.subItems(descuento.ToString("C"));
     fila.subItems(neto.ToString("C"));
     lvReporte.Items.Add(fila);
   }
}
```

6.4 Método sin valor de retorno con parámetros

Tiene la finalidad de realizar una determinada tarea sin devolver algún valor a quien lo invoca pero declara parámetros de entrada.

Por los parámetros se pasarán los valores que necesita el método para su realización.

Formato:

```
void metodo(PARAMETRO){
    //Declaraciones de variables locales
    //Implementación del método
}
```

Donde:

- **Void:** Es la palabra reservada que indica que el método no devolverá valor alguno, es decir, "método sin valor de retorno".
- **metodo(PARAMETRO):** Es el nombre que se le asignará al método, la definición de los parámetros se realiza como la declaración de una variable común.

Veánse algunas implementaciones básicas para el uso de los métodos sin valor de retorno con parámetros:

- Implementar el cálculo del promedio de cuatro notas de un determinado alumno.

```
public partial class frmPrueba: Form
{
   //Seccion de declaracion de variables GLOBALES
   double promedio;

   public frmPrueba()
   {
      InitializeComponent();
   }

   //Implementacion del metodo
   void calculaPromedio(int n1, int n2, int n3, int n4)
   {
     promedio = (n1+n2+n3+n4)/4.0;
   }
}
```

- Implementar el conteo de valores según la categoría (A, B, C) de un trabajador.

```
public partial class frmPrueba: Form
{
   //Seccion de declaracion de variables GLOBALES
   int cA, cB, cC;

   public frmPrueba()
   {
      InitializeComponent();
   }

   //Implementacion del metodo
   void asignaConteos(string categoria)
   {
     switch(categoria){
             case "A": cA++; break;
             case "B": cB++; break;
             case "C": cC++; break;
}
   }
}
```

- Implementar la impresión de los valores de una aplicación de pagos, el cual cuenta con salario, descuento y neto.

```
public partial class frmPrueba: Form
{
   public frmPrueba()
   {
      InitializeComponent();
   }

   //Implementacion del metodo
   void imprimeValores(string empleado, double salario,
                               double descuento, double neto)
   {
      ListViewItem fila=new ListViewItem(empleado)
      fila.subItems(salario.ToString("C"));
      fila.subItems(descuento.ToString("C"));
      fila.subItems(neto.ToString("C"));
      lvReporte.Items.Add(fila);
   }
}
```

6.5 Método con valor de retorno sin parámetros

Tiene la finalidad de realizar una determinada tarea devolviendo un solo valor a quien lo invoca, pero no contiene parámetros de entrada.

Los valores a usar dentro del método deben ser declarados como globales para su asignación y uso.

Formato:

```
tipo metodo(){
    //Declaraciones de variables locales
    //Implementación del método
}
```

Donde:

- **Tipo:** Es la definición del tipo de datos de salida que tiene el método, es decir, como este método siempre devolverá un valor se debe definir de qué tipo será dicha salida.
- **metodo():** Es el nombre que se le asignará al método, como se puede notar no se definen parámetros, por lo tanto, se usan las variables globales.

Veánse algunas implementaciones básicas para el uso de los métodos con valor de retorno sin parámetros:

- Implementar el cálculo del promedio de cuatro notas declaradas como global de un determinado alumno.

```
public partial class frmPrueba: Form
{
   //Seccion de declaracion de variables GLOBALES
   int nota1, nota2, nota3, nota4;

   public frmPrueba()
   {
      InitializeComponent();
   }

   private void btnCalcular_Click(object sender, EventArgs e)
   {
     nota1=10;
     nota2=14;
     nota3=15;
     nota4=20;
     double promedio = calculaPromedio();
```

```
    }

    //Implementacion del metodo
    double calculaPromedio()
    {
      return (nota1+nota2+nota3+nota4)/4.0;
    }
}
```

- Implementar la asignación de costo de producto sabiendo que el producto A=20, B=30 y C=50.

```
public partial class frmPrueba: Form
{
    //Seccion de declaracion de variables GLOBALES
    string producto;

    public frmPrueba()
    {
       InitializeComponent();
    }

    private void btnCalcular_Click(object sender, EventArgs e)
    {
      producto ="A";
      double costo = asignaCosto();
    }

    //Implementacion del metodo
    double asignaCosto()
    {
      switch(producto){
              case "A": return 20; break;
              case "B": return 30; break;
              case "C": return 50; break;
}
    }
}
```

6.6 Método con valor de retorno con parámetros

Tiene la finalidad de realizar una determinada tarea devolviendo un solo valor a quien lo invoca, asimismo define parámetros de entrada los cuales pueden ser usados dentro de las sentencias del método.

Formato:

```
tipo metodo(PARAMETROS){
    //Declaraciones de variables locales
    //Implementación del método
}
```

Donde:

- **tipo:** Es la definición del tipo de datos de salida que tiene el método, es decir, como este método siempre devolverá un valor se debe definir de qué tipo será dicha salida.
- **metodo(PARAMETRO):** Es el nombre que se le asignará al método, los parámetros de entrada se definen como una declaración de variables simples.

Veánse algunas implementaciones básicas para el uso de los métodos con valor de retorno con parámetros:

- Implementar el cálculo del promedio de cuatro notas de un determinado alumno.

```
public partial class frmPrueba: Form
{
   public frmPrueba()
   {
      InitializeComponent();
   }

   private void btnCalcular_Click(object sender, EventArgs e)
   {
     int n1=10;
     int n2=20;
     int n3=14;
     int n4=15;

     double promedio = calculaPromedio(n1,n2,n3,n4);
   }

   //Implementacion del metodo
   double calculaPromedio(int n1, int n2, int n3, int n4)
   {
     return (n1+n2+n3+n4)/4.0;
   }
}
```

- Implementar la asignación de costo de producto sabiendo que el producto A=20, B=30 y C=50.

```
public partial class frmPrueba: Form
{
   public frmPrueba()
   {
      InitializeComponent();
   }

   private void btnCalcular_Click(object sender, EventArgs e)
   {
     producto ="A";
     double costo = asignaCosto(producto);
   }

   //Implementacion del metodo
   double asignaCosto(string producto)
   {
     switch(producto){
            case "A": return 20; break;
            case "B": return 30; break;
            case "C": return 50; break;
   }
      }
   }
```

6.7 Expresiones lambda

A partir de la versión 2005 del Visual Studio se introdujo el concepto de métodos anónimos, de esta manera, algunos llaman a las expresiones lambda. Una expresión lambda tiene un trabajo similar a los métodos con valor de retorno, la diferencia es la forma de implementarlo, ya que permite compactar el código de un determinado proceso.

Formato:

```
Func <tEntrada, tSalida>  NombreExpresionLambda = (aEntrada, aSalida) =>
variable = valor;

Func <tEntrada, tSalida>  NombreExpresionLambda = (aEntrada, aSalida) =>{
  //Sentencias
 return valor;
};
```

Donde:

- **Func:** Es la palabra reservada que indica el inicio de la expresión lambda con operadores estándar.
- **tEntrada:** Representa el tipo de datos del parámetro de entrada que puede tener la expresión lambda, este tipo se define dependiendo de la cantidad de parámetros que necesita la expresión. Si la expresión no necesita parámetros, entonces no se especificará el tipo "tEntrada".
- **tSalida:** La característica de la función es devolver un valor a quien lo invoca, es así que la expresión lambda también especifica el tipo de datos de su valor resultante. tSalida representa el tipo de datos del valor resultante de la expresión lambda. Definir siempre a pesar de no tener valores de entrada.
- **NombreExpresionLambda:** Es el nombre asignado a la expresión lambda y a partir de cual es invocado por los procesos dentro de la aplicación.
- **(aEntrada, aSalida):** Representa a los parámetros de entrada y salida que necesita la expresión lambda. Debe ser la misma cantidad de tipos de datos especificados en tEntrada y tSalida.
- **return valor:** Es la representación del valor resultante de la expresión lambda, se debe tener en cuenta que dicho valor sea del mismo tipo que lo especificado en tSalida.

Veáse una comparación de una función convencional con una expresión lambda:

- Expresión lambda que permita calcular el promedio de tres notas

```
Func<int, int, int, double> calculaPromedio = (n1, n2, n3) =>
       {
          return (n1 + n2 + n3) / 3.0;
};
```

La forma de invocar a la expresión lambda podría ser de la siguiente manera:

```
MessageBox.Show("El promedio es: "+calculaPromedio(10,12,20));
```

- Expresión lambda que permita mostrar la fecha actual

```
Func<string> mostrarFecha = () => DateTime.Now.ToShortDateString();
```

La forma de invocacion de la expresion podría ser de la siguiente manera:

```
MessageBox.Show("La fecha actual es: "+mostrarFecha());
```

- Expresión lambda que permita calcular el sueldo neto de un trabajador a partir de un sueldo base y un descuento.

```
Func<string, double> calculaBase = (categoria) =>
        {
                double montoBase=0;
                switch (categoria)
                {
                    case "A": montoBase = 2500; break;
                    case "B": montoBase = 2000; break;
                    case "C": montoBase = 1500; break;
                    case "D": montoBase = 1000; break;
                }
               return montoBase;
};
```

La forma de invocación de la expresión podría ser de la siguiente manera:

```
MessageBox.Show("El sueldo base es: "+calculaBase("B"));
```

6.8 Casos desarrollados

Caso desarrollado 1 : control de registro de llamadas - método sin valor de retorno sin parámetros

La empresa Bussines-Vent, que presta servicios de call center en la ciudad de Lima, realiza el seguimiento de una cartera de clientes a las cuales se realizó la venta de tarjetas de crédito solicitado por una determinada empresa. El call center necesita una aplicación que permita tener el control de las llamadas que realizan sus terminalistas (operador de llamadas), de forma que registre el tipo de llamada realizada, el horario y la cantidad de minutos. La aplicación deberá registrar las llamadas mostrando el costo por minuto y el costo por llamada.

Se debe tener en cuenta:

- Usar el método sin valor de retorno y sin parámetros.
- Al iniciar la aplicación se debe mostrar la fecha de manera automática.
- Asimismo, la aplicación debe mostrar la hora digitalizada de manera automática.
- El tipo de llamada y el horario deben ser seleccionados desde un cuadro combinado.
- Al seleccionar un tipo de llamada automáticamente se debe mostrar el costo según la tabla de costo por minuto.
- La asignación de los costos y descuentos por tipo se muestran en la siguiente tabla:

Tipo de llamada	Costo por minuto	Descuento
Local nacional	$ 0.20	30 %
Local internacional	$ 0.50	20 %
Móvil nacional	$ 1.20	10 %
Móvil internacional	$ 2.20	30 %

El descuento se aplica al importe calculado (costo por minuto por la cantidad de minutos llamados) según el tipo de llamada.

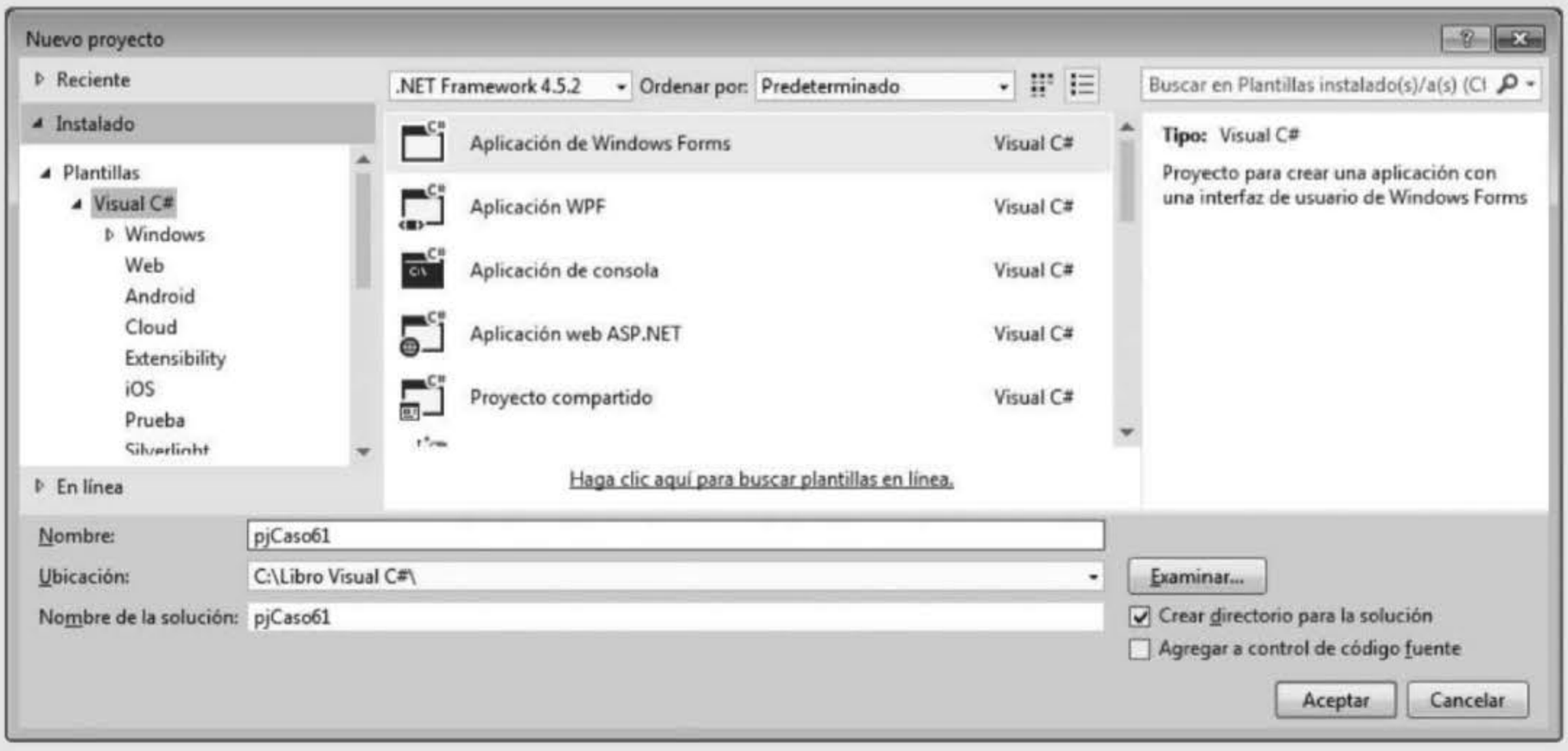
- El costo por llamada resulta de la diferencia del importe de pago y el descuento aplicado.
- El botón **REGISTRAR** envía los datos de las llamadas a un control ListView que muestre el tipo de llamada, horario de llamada, minutos, costo por minuto y el costo por llamada.

- Finalmente, mediante el botón mostrar las estadísticas como el número de llamadas registradas entre 10 y 30 minutos, el costo acumulado por tipo local nacional e internacional, móvil nacional e internacional, monto mayor según el costo de llamada, tipo de llamada con costo de llamada mayor, horario con costo de llamada mayor en un control ListView.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

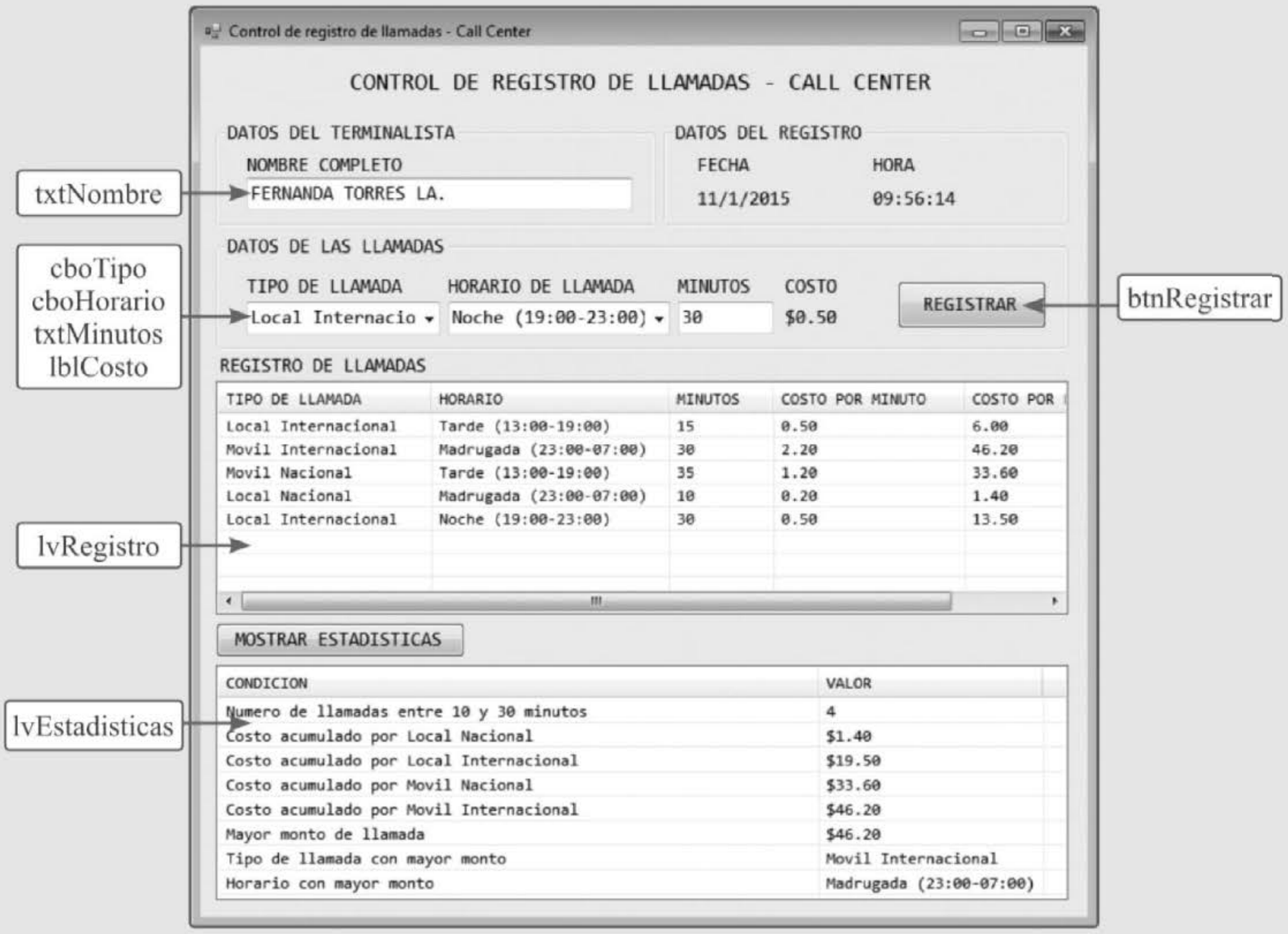
   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso61** como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo** asigne el nombre **frmLlamadas**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmLlamadas:

Clase	N.º elementos	Controles
Form	01	Form1
Label	12	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8, Label9, Label10, Label11, Label12
TextBox	01	Textbox1, Textbox2
ComboBox	02	ComboBox1, ComboBox2
Button	02	Button1, Button2
GroupBox	03	GroupBox1, GroupBox2, GroupBox3
ListView	02	ListView1, ListView2
TOTAL	23	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmLlamadas Control de registro de llamadas – Call Center
Label1	Text	CONTROL DE REGISTRO DE LLAMADAS – CALL CENTER
Label2	Text	NOMBRE COMPLETO
Label3	Text	FECHA
Label4	(Name) Text	lblFecha lblFecha
Label5	Text	HORA
Label6	(Name) Text	lblHora lblHora
Label7	Text	TIPO DE LLAMADA
Label8	Text	HORARIO DE LLAMADA
Label9	Text	MINUTOS
Label10	Text	COSTO
Label11	(Name) Text	lblCosto lblCosto
Label12	Text	REGISTRO DE LLAMADAS
Textbox1	(Name)	txtNombre
Textbox2	(Name)	txtMinutos
ComboBox1	(Name) Items	cboTipo Local Nacional, Local Internacional, Movil Nacional, Movil Internacional
ComboBox2	(Name) Items	cboHorario Diurno (07:00-13:00), Tarde (13:00-19:00), Noche (19:00-23:00), Madrugada (23:00-07:00)
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnEstadisticas MOSTRAR ESTADISTICAS
GroupBox1	Text	DATOS DEL TERMINALISTA
GroupBox2	Text	DATOS DEL REGISTRO
GroupBox3	Text	DATOS DE LAS LLAMADAS
ListView1	(Name) GridLines	lvRegistro True
ListView2	(Name) GridLines	lvEstadisticas True

CÓDIGO: usando métodos sin valor de retorno sin parámetro

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso61
{
    public partial class frmLlamadas : Form
    {
        //Declaracion de variables GLOBALES
        string tipo;
        string horario;
        int minutos;
        double costoMinuto;
        double costoLlamada;

        double mayorMonto;
        string horarioMayor;
        string tipoMayor;

        //Contadores y acumuladores
        int cLlamadas;
        double aLocNac, aLocInt, aMovNac, aMovInt;

        public frmLlamadas()
        {
            InitializeComponent();
            tHora.Enabled = true;
        }

        private void frmLlamadas_Load(object sender, EventArgs e)
        {
            //Mostrar la fecha actual
            lblFecha.Text = DateTime.Now.ToShortDateString();
         }

       private void cboTipo_SelectedIndexChanged(object sender, EventArgs e)
        {
            //Asignar el costo por minuto
            asignaCostoxMinuto();
            lblCosto.Text = costoMinuto.ToString("C");
        }
        private void tHora_Tick(object sender, EventArgs e)
```

```
        {
            //Mostrar la hora
            lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
        }
        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los datos
            horario = cboHorario.Text;
            minutos = int.Parse(txtMinutos.Text);

            //Determinar el costo por minuto
            asignaCostoxMinuto();

            //Determinar el costo por llamada
            asignaCostoxLlamada();

            //Imprimir el registro de llamadas
            imprimirRegistro();

            lvEstadisticas.Items.Clear();
        }

        private void btnEstadisticas_Click(object sender, EventArgs e)
        {
            //Enviar los datos a la lista de estadisticas
            imprimirEstadisticas();
        }

        //Metodo que permite asignar el costo por minuto según el tipo de llamada
        void asignaCostoxMinuto()
        {
            //Capturando el tipo de llamada desde el cuadro combinado
            tipo = cboTipo.Text;

            //Asignando el costo por minuto según el tipo de llamada
            switch (tipo)
            {
                case "Local Nacional": costoMinuto = 0.20; break;
                case "Local Internacional": costoMinuto = 0.50; break;
                case "Movil Nacional": costoMinuto = 1.20; break;
                case "Movil Internacional": costoMinuto = 2.20; break;
            }
        }

        //Metodo que permite asignar por llamada según el horario
        void asignaCostoxLlamada()
        {
            //Variables locales
            double importe = costoMinuto * minutos;
```

```
        double descuento = 0;

        //Asignando el descuento según el horario
        switch (horario)
        {
                case "Diurno (07:00-13:00)": descuento = importe * 0.3; break;
                case "Tarde (13:00-19:00)": descuento = importe * 0.2; break;
                case "Noche (19:00-23:00)": descuento = importe * 0.1; break;
                case "Madrugada (23:00-07:00)": descuento = importe * 0.3; break;
        }
            costoLlamada = importe - descuento;
        }

        //Metodo que permite imprimir los valores en la lista de registro
        void imprimirRegistro()
        {
            ListViewItem fila = new ListViewItem(tipo);
            fila.SubItems.Add(horario);
            fila.SubItems.Add(minutos.ToString());
            fila.SubItems.Add(costoMinuto.ToString("0.00"));
            fila.SubItems.Add(costoLlamada.ToString("0.00"));
            lvRegistro.Items.Add(fila);
        }

       //Metodo que permite mostrar los valores GLOBALES para la estadistica
        void imprimirEstadisticas()
        {
            //Contar el numero de llamadas entre 10 y 30 minutos
            numeroLlamadas();

            //Monto acumulado del costo por llamada por tipo
            costoAcumuladoxtipo();

            //Mayor costo por llamada, que tipo y horario
            mayorMontoLlamada();

            //Enviando los resultados a la lista de Estadisticas
            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[2];
            ListViewItem row;

            elementosFila[0] = "Numero de llamadas entre 10 y 30 minutos";
            elementosFila[1] = cLlamadas.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Local Nacional";
            elementosFila[1] = aLocNac.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
```

```
            elementosFila[0] = "Costo acumulado por Local Internacional";
            elementosFila[1] = aLocInt.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Movil Nacional";
            elementosFila[1] = aMovNac.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Movil Internacional";
            elementosFila[1] = aMovInt.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Mayor monto de llamada";
            elementosFila[1] = mayorMonto.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Tipo de llamada con mayor monto";
            elementosFila[1] = tipoMayor;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Horario con mayor monto";
            elementosFila[1] = horarioMayor;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        //Determinar el numero de llamadas entre 10 y 30 minutos
        void numeroLlamadas()
        {

            //Inicializar el contador de llamadas
            cLlamadas = 0;

            //Recorremos por todo los registros de la lista
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
                //Capturamos los minutos
                int minutos = int.Parse(lvRegistro.Items[i].SubItems[2].Text);
                //Comparamos si los mintos se encuentran en el rango de 10 y 30
                if (minutos >= 10 && minutos <= 30) cLlamadas++;
            }
        }

        //Metodo que determina el total acumulado del costo por llamada
        void costoAcumuladoxtipo()
        {
```

```
        //Inicializar las variables acumuladoras en cero
        aLocNac=0; aLocInt=0; aMovNac=0; aMovInt=0;

        //Recorriendo por todos los registros
        for (int i = 0; i < lvRegistro.Items.Count; i++)
        {
            //Capturando el tipo de llamada
            string t = lvRegistro.Items[i].SubItems[0].Text;

           //Condicionar el tipo de llamadas para realizar la acumulacion
             if (t == "Local Nacional")
            aLocNac+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
             else if (t == "Local Internacional")
            aLocInt+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
             else if (t == "Movil Nacional")
            aMovNac+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
             else if (t == "Movil Internacional")
            aMovInt+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
        }
    }

    //Metodo que determina el mayor monto de llamada
    void mayorMontoLlamada()
    {
     //Inicializar la variable local posicion
     int posicion = 0;

    //Inicializar la variable mayor con el primer costo de los registros
    mayorMonto = double.Parse(lvRegistro.Items[0].SubItems[4].Text);

     //Recorrer por todos los registros
     for (int i = 0; i < lvRegistro.Items.Count; i++)
     {

     //Si uno de los costos es mayor que el valor asignado a la variable
     //mayor entonces hemos encontrado el mayor de los elementos
     if (double.Parse(lvRegistro.Items[i].SubItems[4].Text) > mayorMonto)
     {
      mayorMonto = double.Parse(lvRegistro.Items[i].SubItems[4].Text);
      posicion = i;
     }

            tipoMayor = lvRegistro.Items[posicion].SubItems[0].Text;
            horarioMayor = lvRegistro.Items[posicion].SubItems[1].Text;
        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- Asignar el costo por minuto según el tipo de llamada por metodo sin valores de retorno sin parámetro.

Para iniciar se debe capturar el tipo de llamada seleccionada por el usuario, por ser una cadena de caracteres no necesitará ser parseado ya que la variable que recibirá el tipo debe ser de tipo string. La variable "tipo" debe ser declarada como global porque dicho valor será usado dentro de otros métodos, por ejemplo, en la impresión.

```
void asignaCostoxMinuto()
        {
            tipo = cboTipo.Text;
     switch (tipo)
            {
                case "Local Nacional": costoMinuto = 0.20; break;
                case "Local Internacional": costoMinuto = 0.50; break;
                case "Movil Nacional": costoMinuto = 1.20; break;
                case "Movil Internacional": costoMinuto = 2.20; break;
            }
        }
```

Finalmente, luego de una serie de comparaciones del valor de la variable "tipo", asigne un monto a la variable costoMinuto que será usada en otros procesos, por eso, es necesario declararlo en la sección global.

- Asignar el costo por tipo de llamada usando método sin valor de retorno sin parámetros.

Inicie calculando el importe que se genera por el producto del costo por minuto y la cantidad de llamadas por minutos ya que gracias a este valor podrá calcular el descuento.

Inicie el monto de descuento en cero ya que al entrar a la evaluación en la estructura switch recién se le dará un valor; si no lo inicializa genera un error de inicialización. Finalmente, calcule el costo por llamada según el importe y el descuento.

```
        void asignaCostoxLlamada()
        {

            //Variables locales
            double importe = costoMinuto * minutos;
            double descuento = 0;
```

```
            //Asignando el descuento según el horario
            switch (horario)
            {
                case "Diurno (07:00-13:00)": descuento = importe * 0.3; break;
                case "Tarde (13:00-19:00)": descuento = importe * 0.2; break;
                case "Noche (19:00-23:00)": descuento = importe * 0.1; break;
                case "Madrugada (23:00-07:00)": descuento = importe * 0.3; break;
            }
            costoLlamada = importe - descuento;
        }
```

- **Imprimir las estadísticas**

  La impresión de las estadísticas se realiza con muchos procesos internos, así también se invoca a otros métodos, específicamente uno para cada proceso. Empiece por determinar el número de llamadas, calcular el costo acumulado por tipo, determinar el mayor monto por llamada y, finalmente, enviarlo a la lista de estadísticas.

```
void numeroLlamadas()
       {
     cLlamadas = 0;
     for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
                int minutos = int.Parse(lvRegistro.Items[i].SubItems[2].Text);
                if (minutos >= 10 && minutos <= 30) cLlamadas++;
            }
        }
```

El método "numeroLlamadas" se encarga de contar cuántas llamadas se encuentran entre 10 y 30 minutos; para esto debe recorrer por la columna donde se muestran los minutos. En la siguiente imagen dicha columna es la número 2 según el registro de llamadas.

REGISTRO DE LLAMADAS

TIPO DE LLAMADA	HORARIO	MINUTOS	COSTO POR MINUTO	COSTO POR LLA
Local Internacional	Tarde (13:00-19:00)	20	0.50	8.00
Movil Nacional	Madrugada (23:00-07:00)	100	1.20	84.00
Local Internacional	Noche (19:00-23:00)	40	0.50	18.00
Movil Nacional	Diurno (07:00-13:00)	30	1.20	25.20
Local Nacional	Tarde (13:00-19:00)	5	0.20	0.80
0	1	2	3	4

Para determinar la última posición de los minutos use la instrucción lvRegistro.Items.Count, ahora capture el valor del primer minuto usando la sentencia **lvRegistro.Items[i].SubItems[2].Text**, donde la variable i representa a todos los demás minutos mostrados en la lista.

```
void costoAcumuladoxtipo()
        {
            aLocNac=0; aLocInt=0; aMovNac=0; aMovInt=0;
     for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
                string t = lvRegistro.Items[i].SubItems[0].Text;
          if (t == "Local Nacional")
                    aLocNac+=double.Parse(lvRegistro.Items[i].SubItems[4].
Text);
                else if (t == "Local Internacional")
                    aLocInt+=double.Parse(lvRegistro.Items[i].SubItems[4].
Text);
                else if (t == "Movil Nacional")
                    aMovNac+=double.Parse(lvRegistro.Items[i].SubItems[4].
Text);
                else if (t == "Movil Internacional")
                    aMovInt+=double.Parse(lvRegistro.Items[i].SubItems[4].
Text);
            }
        }
```

Para determinar los costos acumulados según el tipo, debe inicializar las variables acumuladoras en el valor cero puesto que cuando se invoque al método este volverá a contabilizar los valores, por ello, este paso es sumanente necesario. Luego, recorra por todos los valores del registro, siempre desde la posición cero hasta el último elemento de la lista (lvRegistro). Esta vez tener en cuenta que lo que necesita evaluar es el tipo de llamada, por lo tanto, debe capturar el valor de la posición cero, y a la vez evaluarlo en una sentencia switch para determinar los acumuladores según el tipo. Hay que considerar que los valores a acumular se encuentran en la columna cuatro.

```
void mayorMontoLlamada()
       {
     int posicion = 0;
     mayorMonto = double.Parse(lvRegistro.Items[0].SubItems[4].Text);

     for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
```

```
if (double.Parse(lvRegistro.Items[i].SubItems[4].Text) > mayorMonto)
                {
                   mayorMonto = double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                   posicion = i;
                }
               }

               tipoMayor = lvRegistro.Items[posicion].SubItems[0].Text;
               horarioMayor = lvRegistro.Items[posicion].SubItems[1].Text;
           }
```

Inicie declarando la variable local "posicion" que tendrá por misión determinar la posición en que se encuentra el mayor valor de la lista. Asimismo, declare la variable local mayorMonto que inicialmente captura el primer valor de la lista asumiendo que es el mayor valor porque aún no se han comparado los demás valores. Dicha comparación se debe realizar desde los demás valores de la lista obtenidos con **lvRegistro.Items[i].SubItems[4]**.Text con el valor declarado en "mayorMonto", si alguno de los elementos supera al valor inicial entonces será el nuevo mayor valor. Una vez encontrado el mayor valor capture la posición donde se encontró dicho valor, esta posición está registrada en la variable "i", por tal razon, se le asigna a la variable posición.

Finalmente, la variable "tipoMayor" obtiene el tipo de llamada que tiene el mayor costo por llamada desde la sentencia **lvRegistro.Items[posicion].SubItems[0]**.Text donde posición tiene el valor de i que se encontró al mayor elemento y como se quiere mostrar qué tipo de llamada era se usa el valor de la columna cero.

Caso desarrollado 2 : control de registro de llamadas - método sin valor de retorno con parámetros

La empresa Bussines-Vent, que presta servicios de call center en la ciudad de Lima, realiza el seguimiento de una cartera de clientes a los cuales se realizó la venta de tarjetas de crédito solicitado por una determinada empresa. El call center necesita una aplicación que permita tener el control de las llamadas que realizan sus terminalistas (operador de llamadas), de forma que registre el tipo de llamada realizada, el horario y la cantidad de minutos. La aplicación deberá registrar las llamadas mostrando el costo por minuto y el costo por llamada.

Se debe tener en cuenta:

- Usar el método sin valor de retorno con especificación de parámetros.
- Al iniciar la aplicación se debe mostrar la fecha de manera automática.
- Asimismo, la aplicación debe mostrar la hora digitalizada de manera automática.
- El tipo de llamada y el horario deben ser seleccionados desde un cuadro combinado.
- Al seleccionar un tipo de llamada automáticamente se debe mostrar el costo según la tabla de costo por minuto.
- La asignación de los costos y descuentos por tipo se muestran en la siguiente tabla:

Tipo de llamada	Costo por minuto	Descuento
Local nacional	$ 0.20	30 %
Local internacional	$ 0.50	20 %
Movil nacional	$ 1.20	10 %
Movil internacional	$ 2.20	30 %

El descuento se aplica al importe calculado (costo por minuto por la cantida de minutos llamados) según el tipo de llamada.

- El costo por llamada resulta de la diferencia del importe de pago y el descuento aplicado.
- El botón **REGISTRAR** envía los datos de las llamadas a un control ListView que muestre el tipo de llamada, horario de llamada, minutos, costo por minuto y el costo por llamada.

- Finalmente, mediante un botón mostrar las estadísticas como el número de llamadas registradas entre 10 y 30 minutos, el costo acumulado por tipo local nacional e internacional, móvil nacional e internacional, monto mayor según el costo de llamada, tipo de llamada con costo de llamada mayor, horario con costo de llamada mayor en un control ListView.

CÓDIGO: usando métodos sin valor de retorno con parámetro

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso61
{
    public partial class frmLlamadasSinValorRetornoConParametros : Form
    {
        //** Declaracion de variables GLOBALES***//
        double costoMinuto;
        double costoLlamada;
        //** Fin de la declaracion de GLOBALES **//

        public frmLlamadasSinValorRetornoConParametros()
        {
            InitializeComponent();
            tHora.Enabled = true;
        }

        private void frmLlamadasSinValorRetornocONParametros_Load(…)
        {
            //Mostrando la fecha actual
            lblFecha.Text = DateTime.Now.ToShortDateString();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los datos
            string tipo = cboTipo.Text;
            string horario = cboHorario.Text;
            int minutos = int.Parse(txtMinutos.Text);

            //Determinar el costo por minuto
            asignaCostoxMinuto(tipo);
```

```
        //Determinar el costo por llamada
        asignaCostoxLlamada(horario, minutos);

        //Imprimir el registro de llamadas
        imprimirRegistro(tipo, horario, minutos);

        lvEstadisticas.Items.Clear();
    }

    private void cboTipo_SelectedIndexChanged(object sender, EventArgs e)
    {
        //Asignar el costo por minuto
        asignaCostoxMinuto(cboTipo.Text);
        lblCosto.Text = costoMinuto.ToString("C");
    }

    private void tHora_Tick(object sender, EventArgs e)
    {
        lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
    }

    private void btnEstadisticas_Click(object sender, EventArgs e)
    {
        //Determinar el numero de llamadas entre 10 y 30
        int cLlamadas = 0;
        for (int i = 0; i < lvRegistro.Items.Count; i++)
        {
          int minutos = int.Parse(lvRegistro.Items[i].SubItems[2].Text);
            if (minutos >= 10 && minutos <= 30) cLlamadas++;
        }

        //Determinar los valores acumulador por tipo de llamada
        double aLocNac = 0,aLocInt = 0, aMovNac = 0,aMovInt = 0;
        for (int i = 0; i < lvRegistro.Items.Count; i++)
        {
            //Capturando el tipo de llamada
            string t = lvRegistro.Items[i].SubItems[0].Text;
            if (t == "Local Nacional")
                aLocNac+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
            else if (t == "Local Internacional")
                aLocInt+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
            else if (t == "Movil Nacional")
                aMovNac+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
            else if (t == "Movil Internacional")
                aMovInt+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
        }
```

```
            //Determinar el mayor monto de llamada
            double mayorMonto=double.Parse(lvRegistro.Items[0].SubItems[4].Text);
            int posicion = 0;
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
              if (double.Parse(lvRegistro.Items[i].SubItems[4].Text)>mayorMonto)
              {
                    mayorMonto = double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                    posicion = i;
              }
            }

            string tipoMayor = lvRegistro.Items[posicion].SubItems[0].Text;
            string horarioMayor = lvRegistro.Items[posicion].SubItems[1].Text;

            //Mostrar los resultados en la lista de estadisticas
            imprimirEstadisticas(cLlamadas, aLocNac, aLocInt, aMovNac,
                                 aMovInt, mayorMonto, tipoMayor, horarioMayor);
        }

        //Asignacion de costo por minuto segun el tipo
        void asignaCostoxMinuto(string tipo)
        {
            switch (tipo)
            {
                case "Local Nacional": costoMinuto = 0.20; break;
                case "Local Internacional": costoMinuto = 0.50; break;
                case "Movil Nacional": costoMinuto = 1.20; break;
                case "Movil Internacional": costoMinuto = 2.20; break;
            }
        }

        //Asignar el costo por llamada segun el horario
        void asignaCostoxLlamada(string horario, int minutos)
        {
            //Calculando el importe
            double importe = costoMinuto * minutos;

            //Determinado el descuento segun el horario
            double descuento = 0;
            switch (horario)
            {
              case "Diurno (07:00-13:00)": descuento = importe * 0.3; break;
              case "Tarde (13:00-19:00)": descuento = importe * 0.2; break;
              case "Noche (19:00-23:00)": descuento = importe * 0.1; break;
              case "Madrugada (23:00-07:00)": descuento = importe * 0.3; break;
            }
            costoLlamada = importe - descuento;
        }
```

```
//Enviando informacion a la lista lvRegistro
void imprimirRegistro(string tipo, string horario, int minutos)
{
    ListViewItem fila = new ListViewItem(tipo);
    fila.SubItems.Add(horario);
    fila.SubItems.Add(minutos.ToString());
    fila.SubItems.Add(costoMinuto.ToString("0.00"));
    fila.SubItems.Add(costoLlamada.ToString("0.00"));
    lvRegistro.Items.Add(fila);
}

//Enviando la informacion a la lista lvEstadisticas
void imprimirEstadisticas(int cLlamadas, double aLocNac, double aLocInt,
                          double aMovNac, double aMovInt,
                          double mayorMonto,
                          string tipoMayor, string horarioMayor)
{
    //Enviando los resultados
    lvEstadisticas.Items.Clear();
    string[] elementosFila = new string[2];
    ListViewItem row;

    elementosFila[0] = "Numero de llamadas entre 10 y 30 minutos";
    elementosFila[1] = cLlamadas.ToString();
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);

    elementosFila[0] = "Costo acumulado por Local Nacional";
    elementosFila[1] = aLocNac.ToString("C");
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);

    elementosFila[0] = "Costo acumulado por Local Internacional";
    elementosFila[1] = aLocInt.ToString("C");
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);

    elementosFila[0] = "Costo acumulado por Movil Nacional";
    elementosFila[1] = aMovNac.ToString("C");
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);

    elementosFila[0] = "Costo acumulado por Movil Internacional";
    elementosFila[1] = aMovInt.ToString("C");
    row = new ListViewItem(elementosFila);
    lvEstadisticas.Items.Add(row);
```

```
            elementosFila[0] = "Mayor monto de llamada";
            elementosFila[1] = mayorMonto.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Tipo de llamada con mayor monto";
            elementosFila[1] = tipoMayor;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Horario con mayor monto";
            elementosFila[1] = horarioMayor;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }
    }
}
```

Caso desarrollado 3 : control de registro de llamadas - método con valor de retorno sin parámetros

La empresa Bussines-Vent, que presta servicios de call center en la ciudad de Lima, realiza el seguimiento de una cartera de clientes a los cuales se realizó la venta de tarjetas de crédito solicitado por una determinada empresa. El call center necesita una aplicación que permita tener el control de las llamadas que realizan sus terminalistas (operador de llamadas), de forma que registre el tipo de llamada realizada, el horario y la cantidad de minutos. La aplicación deberá registrar las llamadas mostrando el costo por minuto y el costo por llamada.

Se debe tener en cuenta:

- Usar el método con valor de retorno y sin especificación de parámetros.
- Al iniciar la aplicación se debe mostrar la fecha de manera automatica.
- Asimismo, la aplicación debe mostrar la hora digitalizada de manera automática.
- El tipo de llamada y el horario deben ser seleccionados desde un cuadro combinado.
- Al seleccionar un tipo de llamada automáticamente se debe mostrar el costo según la tabla de costo por minuto.

- La asignación de los costos y descuentos por tipo se muestran en la siguiente tabla:

Tipo de llamada	Costo por minuto	Descuento
Local nacional	$ 0.20	30 %
Local internacional	$ 0.50	20 %
Movil nacional	$ 1.20	10 %
Movil internacional	$ 2.20	30 %

El descuento se aplica al importe calculado (costo por minuto por la cantida de minutos llamados) según el tipo de llamada.

- El costo por llamada resulta de la diferencia del importe de pago y el descuento aplicado.
- El botón **REGISTRAR** envía los datos de las llamadas a un control ListView que muestre el tipo de llamada, horario de llamada, minutos, costo por minuto y el costo por llamada.
- Finalmente, mediante un botón mostrar las estadísticas como el número de llamadas registradas entre 10 y 30 minutos, el costo acumulado por tipo local nacional e internacional, móvil nacional e internacional, monto mayor según el costo de llamada, tipo de llamada con costo de llamada mayor, horario con costo de llamada mayor en un control ListView.

CÓDIGO: usando métodos con valor de retorno sin parámetro

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso61
{
    public partial class frmLlamadasConValorRetornoSinParametros : Form
    {
        public frmLlamadasConValorRetornoSinParametros()
        {
            InitializeComponent();
        }
```

```
        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Imprimir el registro de llamadas
            ListViewItem fila = new ListViewItem(getTipo());
            fila.SubItems.Add(getHorario());
            fila.SubItems.Add(getMinutos().ToString());
            fila.SubItems.Add(asignaCostoxMinuto().ToString("0.00"));
            fila.SubItems.Add(asignaCostoxLlamada().ToString("0.00"));
            lvRegistro.Items.Add(fila);

            lvEstadisticas.Items.Clear();
        }

        private void frmLlamadasConValorRetornoSinParametros_Load(…)
        {
            //Mostrando la fecha actual
            lblFecha.Text = DateTime.Now.ToShortDateString();
        }

        private void cboTipo_SelectedIndexChanged(object sender, EventArgs e)
        {
            //Asignar el costo por minuto
            lblCosto.Text = asignaCostoxMinuto().ToString("C");
        }

        private void btnEstadisticas_Click(object sender, EventArgs e)
        {
            //Determinar el total acumulado del costo por llamada por tipo
            double aLocNac = 0,aLocInt = 0,aMovNac = 0,aMovInt = 0;
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
                //Capturando el tipo de llamada
                string t = lvRegistro.Items[i].SubItems[0].Text;
                if (t == "Local Nacional")
                    aLocNac+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                else if (t == "Local Internacional")
                    aLocInt+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                else if (t == "Movil Nacional")
                    aMovNac+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                else if (t == "Movil Internacional")
                    aMovInt+=double.Parse(lvRegistro.Items[i].SubItems[4].Text);
            }

            //Enviando los resultados
            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[2];
            ListViewItem row;

            elementosFila[0] = "Numero de llamadas entre 10 y 30 minutos";
            elementosFila[1] = numeroLlamadas().ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
```

```
            elementosFila[0] = "Costo acumulado por Local Nacional";
            elementosFila[1] = aLocNac.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Local Internacional";
            elementosFila[1] = aLocInt.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Movil Nacional";
            elementosFila[1] = aMovNac.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Movil Internacional";
            elementosFila[1] = aMovInt.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Determinar el mayor monto de llamada
            double mayorMonto=double.Parse(lvRegistro.Items[0].SubItems[4].Text);
            int posicion = 0;
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
              if (double.Parse(lvRegistro.Items[i].SubItems[4].Text)>mayorMonto)
              {
                mayorMonto = double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                posicion = i;
              }
            }

            elementosFila[0] = "Mayor monto de llamada";
            elementosFila[1] = mayorMonto.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Tipo de llamada con mayor monto";
            elementosFila[1] = lvRegistro.Items[posicion].SubItems[0].Text;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Horario con mayor monto";
            elementosFila[1] = lvRegistro.Items[posicion].SubItems[1].Text;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        private void tHora_Tick(object sender, EventArgs e)
        {
```

```
        lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
    }

    //Capturando los valores del formulario
    string getTipo()
    {
        return cboTipo.Text;
    }

    string getHorario()
    {
        return cboHorario.Text;
    }

    int getMinutos()
    {
        return int.Parse(txtMinutos.Text);
    }

    //Asignacion de costo por minuto segun el tipo
    double asignaCostoxMinuto()
    {
        double costoMinuto=0;
        switch (cboTipo.Text)
        {
            case "Local Nacional": costoMinuto = 0.20; break;
            case "Local Internacional": costoMinuto = 0.50; break;
            case "Movil Nacional": costoMinuto = 1.20; break;
            case "Movil Internacional": costoMinuto = 2.20; break;
        }
        return costoMinuto;
    }

    //Asignar el costo por llamada segun el horario
    double asignaCostoxLlamada()

    {
        //Calculando el importe
        double importe = asignaCostoxMinuto() * getMinutos();

        //Determinado el descuento segun el horario
        double descuento = 0;
        switch (getHorario())
        {
           case "Diurno (07:00-13:00)": descuento = importe * 0.3; break;
           case "Tarde (13:00-19:00)": descuento = importe * 0.2; break;
           case "Noche (19:00-23:00)": descuento = importe * 0.1; break;
           case "Madrugada (23:00-07:00)": descuento = importe * 0.3; break;
        }
```

```
        return importe - descuento;
    }

    //Determinar el numero de llamadas entre 10 y 30 minutos
    int numeroLlamadas()
    {
        int cLlamadas = 0;
        for (int i = 0; i < lvRegistro.Items.Count; i++)
        {
          int minutos = int.Parse(lvRegistro.Items[i].SubItems[2].Text);
            if (minutos >= 10 && minutos <= 30) cLlamadas++;
        }
        return cLlamadas;
    }
}
}
```

Caso desarrollado 4 : control de registro de llamadas - método con valor de retorno con parámetros

La empresa Bussines-Vent, que presta servicios de call center en la ciudad de Lima, realiza el seguimiento de una cartera de clientes a los cuales se le realizó la venta de tarjetas de crédito solicitado por una determinada empresa. El call center necesita una aplicación que permita tener el control de las llamadas que realizan sus terminalistas (operador de llamadas), de forma que registre el tipo de llamada realizada, el horario y la cantidad de minutos. La aplicación deberá registrar las llamadas mostrando el costo por minuto y el costo por llamada.

Se debe tener en cuenta:

- Usar el método con valor de retorno y con especificación de parámetros.
- Al iniciar la aplicación se debe mostrar la fecha de manera automatica.
- Asimismo, la aplicación debe mostrar la hora digitializada de manera automática.
- El tipo de llamada y el horario deben ser seleccionados desde un cuadro combinado.
- Al seleccionar un tipo de llamada automáticamente se debe mostrar el costo según la tabla de costo por minuto.
- La asignación de los costos y descuentos por tipo se muestran en la siguiente tabla:

Tipo de llamada	Costo por minuto	Descuento
Local nacional	$ 0.20	30 %
Local internacional	$ 0.50	20 %
Móvil nacional	$ 1.20	10 %
Móvil internacional	$ 2.20	30 %

El descuento se aplica al importe calculado (costo por minuto por la cantida de minutos llamados) según el tipo de llamada.

- El costo por llamada resulta de la diferencia del importe de pago y el descuento aplicado.
- El botón **REGISTRAR** envía los datos de las llamadas a un control ListView que muestre el tipo de llamada, horario de llamada, minutos, costo por minuto y el costo por llamada.

- Finalmente, mediante un botón mostrar las estadísticas como el número de llamadas registradas entre 10 y 30 minutos, el costo acumulado por tipo local nacional e internacional, móvil nacional e internacional, monto mayor según el costo de llamada, tipo de llamada con costo de llamada mayor, horario con costo de llamada mayor en un control ListView.

CÓDIGO: usando métodos con valor de retorno con parmetro

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso61
{
    public partial class frmLlamadasConValorRetornoConParametros : Form
    {
        public frmLlamadasConValorRetornoConParametros()
        {
            InitializeComponent();
        }

        private void frmLlamadasConValorRetornoConParametros_Load(…)
        {
            lblFecha.Text = DateTime.Now.ToShortDateString();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            string tipo = cboTipo.Text;
            string horario = cboHorario.Text;
            int minutos = int.Parse(txtMinutos.Text);

            ListViewItem fila = new ListViewItem(tipo);
            fila.SubItems.Add(horario);
            fila.SubItems.Add(minutos.ToString());
            fila.SubItems.Add(asignaCostoxMinuto(tipo).ToString("0.00"));
            fila.SubItems.Add(asignaCostoxLlamada(horario).ToString("0.00"));
            lvRegistro.Items.Add(fila);

            lvEstadisticas.Items.Clear();
        }
        private void tHora_Tick(object sender, EventArgs e)
```

```
        {
            lblHora.Text = DateTime.Now.ToString("hh:mm:ss");
        }

        private void cboTipo_SelectedIndexChanged(object sender, EventArgs e)
        {
            lblCosto.Text = asignaCostoxMinuto(cboTipo.Text).ToString("C");
        }

        private void btnEstadisticas_Click(object sender, EventArgs e)
        {
            int cLlamadas = 0;
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
              int minutos = int.Parse(lvRegistro.Items[i].SubItems[2].Text);
                if (minutos >= 10 && minutos <= 30) cLlamadas++;
            }

            double aLocNac = 0, aLocInt = 0, aMovNac = 0, aMovInt = 0;
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
                string t = lvRegistro.Items[i].SubItems[0].Text;
                if (t == "Local Nacional")
                    aLocNac += double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                else if (t == "Local Internacional")
                    aLocInt += double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                else if (t == "Movil Nacional")
                    aMovNac += double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                else if (t == "Movil Internacional")
                    aMovInt += double.Parse(lvRegistro.Items[i].SubItems[4].Text);
            }

            //Enviando los resultados
            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[2];
            ListViewItem row;

            elementosFila[0] = "Numero de llamadas entre 10 y 30 minutos";
            elementosFila[1] = cLlamadas.ToString();
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Local Nacional";
            elementosFila[1] = aLocNac.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
```

```
            elementosFila[0] = "Costo acumulado por Local Internacional";
            elementosFila[1] = aLocInt.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Movil Nacional";
            elementosFila[1] = aMovNac.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Costo acumulado por Movil Internacional";
            elementosFila[1] = aMovInt.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            //Determinar el mayor monto de llamada
            double mayorMonto=double.Parse(lvRegistro.Items[0].SubItems[4].Text);
            int posicion = 0;
            for (int i = 0; i < lvRegistro.Items.Count; i++)
            {
              if (double.Parse(lvRegistro.Items[i].SubItems[4].Text)>mayorMonto)
              {
                 mayorMonto = double.Parse(lvRegistro.Items[i].SubItems[4].Text);
                 posicion = i;
              }
            }

            elementosFila[0] = "Mayor monto de llamada";
            elementosFila[1] = mayorMonto.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Tipo de llamada con mayor monto";
            elementosFila[1] = lvRegistro.Items[posicion].SubItems[0].Text;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Horario con mayor monto";
            elementosFila[1] = lvRegistro.Items[posicion].SubItems[1].Text;
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        //IMPLEMENTACION DE LOS METODOS
        //Asignacion de costo por minuto segun el tipo
        double asignaCostoxMinuto(string tipo)
        {
```

```
        double costoMinuto = 0;
        switch (tipo)
        {
            case "Local Nacional": costoMinuto = 0.20; break;
            case "Local Internacional": costoMinuto = 0.50; break;
            case "Movil Nacional": costoMinuto = 1.20; break;
            case "Movil Internacional": costoMinuto = 2.20; break;
        }
        return costoMinuto;
    }

    //Asignar el costo por llamada segun el horario
    double asignaCostoxLlamada(string horario)
    {
double importe=asignaCostoxMinuto(cboTipo.Text)*int.Parse(txtMinutos.Text);

        double descuento = 0;
        switch (horario)
        {
            case "Diurno (07:00-13:00)": descuento = importe * 0.3; break;
            case "Tarde (13:00-19:00)": descuento = importe * 0.2; break;
            case "Noche (19:00-23:00)": descuento = importe * 0.1; break;
            case "Madrugada (23:00-07:00)": descuento = importe * 0.3; break;
        }
        return importe - descuento;
    }
}
```

Caso desarrollado 5: integración de métodos - control de pago de empleados

Marketing S. A. C. es una empresa cooperativa de ahorro y crédito dedicada a brindar productos financieros y contribuir con el crecimiento de sus clientes, en consecuencia, necesita una aplicacion que permita controlar el pago que realiza a sus empleados así como el monto por movilidad, asignación a favor por el número de hijos, el descuento de ley y el neto que la empresa necesita pagar a sus empleados.

Se debe tener en cuenta:

- Usar los métodos necesarios para la implementación de la aplicación.
- La asignación del sueldo base se procesa de acuerdo a la siguiente tabla:

Área/Condición	Personal	Comisión
Ventas	$ 2500.00	$ 500.00
Marketing	$ 1800.00	$ 1100.00
Logística	$ 3500.00	$ 2500.00
Préstamo	$ 1500.00	$ 900.00

- La asignación por concepto de movilidad se procesar de acuerdo al area y se aplica sobre el sueldo base, como se muestra en la siguiente tabla:

Área/Condición	Personal	Comisión
Ventas	10 %	5 %
Marketing	20 %	10 %
Logística	30 %	15 %
Préstamo	25 %	12.5 %

- Calcular el tiempo de servicio según la fecha de ingreso que se registra para un determinado empleado.
- Calcular una asignación adicional por la cantidad de hijos que registra el empleado, de forma que por cada hijo se le aumente $ 20.00.
- Calcular un descuento del 17 % sobre el sueldo base del empleado.
- Calcular el monto neto que recibe el empleado, de acuerdo al sueldo base, asignación de movilidad, asignación por número de hijos y descuento, usando la siguiente fórmula:

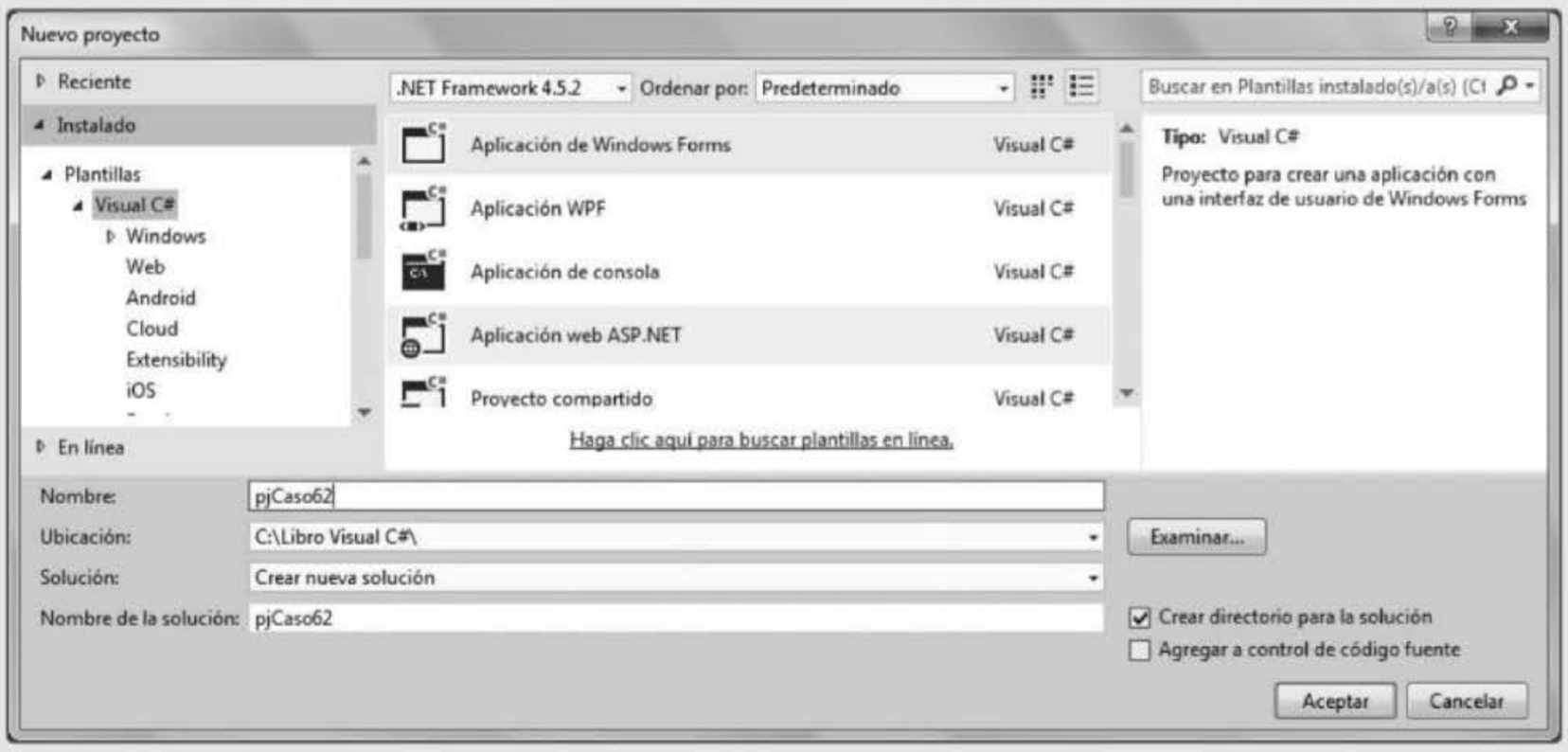
$$\text{Neto} = \text{sueldo Base} + \text{movilidad} + \text{asignación} - \text{descuento}$$

- Validar el nombre del empleado de forma que no permita registrar al mismo empleado en la lista de empleados.
- Implementar el botón "Consulta 1"que permita determinar el total de empleados por área.
- Implementar el botón "Consulta 2" que permita determinar el monto acumulado de los netos según la condición del empleado.
- Implementar el botón "Consulta 3" que permita determinar el monto acumulado de los netos según el área.
- Finalmente, validar los valores del formulario, de forma que al registrar a un empleado no permita registrar espacios vacíos en el nombre del empleado, determinar que el número de hijos sea numérico, determinar que el área y la condicion sean seleccionados para un determinado empleado.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.
   - Asigne el nombre **pjCaso62** como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo** asigne el nombre **frmPlanilla**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en frmPlanilla:

Clase	N.º elementos	Controles
Form	01	Form1
Label	06	Label1, Label2, Label3, Label4, Label5, Label6,
TextBox	02	Textbox1, Textbox2
ComboBox	02	ComboBox1, ComboBox2
Button	05	Button1, Button2, Button3, Button4, Button5
DateTimePicker	01	DateTimePicker1
GroupBox	02	GroupBox1, GroupBox2
ListView	02	ListView1, ListView2
TOTAL	21	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmPlanilla Control de pago de empleados
Label1	Text	CONTROL DE PAGO DE EMPLEADOS
Label2	Text	EMPLEADO
Label3	Text	NUM. HIJOS
Label4	Text	AREA
Label5	Text	CONDICIÓN
Label6	Text	FECHA DE ING.
DateTimePicker1	(Name) Format	dtFechaIng Short
Textbox1	(Name)	txtEmpleado
Textbox2	(Name)	txtHijos

ComboBox1	(Name) Items	cboArea Ventas, Marketing, Logistica, Prestamo
ComboBox2	(Name) Items	cboCondicion Personal, Comision
Button1	(Name) Text	btnRegistrar REGISTRAR
Button2	(Name) Text	btnConsulta1 CONSULTA 1
Button3	(Name) Text	btnConsulta2 CONSULTA 2
Button4	(Name) Text	btnConsulta3 CONSULTA 3
Button5	(Name) Text	btnSalir SALIR
GroupBox1	Text	DATOS DEL EMPLEADO
GroupBox2	Text	DATOS PLANILLA
ListView1	(Name) GridLines	lvEmpleado True
ListView2	(Name) GridLines	lvEstadisticas True

5. Código completo de la aplicación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Microsoft.VisualBasic;

namespace pjCaso62
{
    public partial class frmPlanilla : Form
    {
      //Declaracion de variables GLOBALES
      double tVentas, tMarketing, tLogistica, tPrestamo;
      double aVentas, aMarketing, aLogistica, aPrestamo;
      double aPersonal, aComision;

      public frmPlanilla()
      {
        InitializeComponent();
```

```
}
private void btnRegistrar_Click(object sender, EventArgs e)
{
  if (valida() == "")
  {
    //Captura de datos
    string empleado = getEmpleado();
    if (validaEmpleado() == false)
    {
      int hijos = getHijos();
      string area = getArea();
      string condicion = getCondicion();

      //Realizando calculos
      int años = calculaTiempoServicio();
      double sueldoBase = asignaSueldoBase(area, condicion);
      double movilidad = calculaMovilidad(sueldoBase);
      double asignacion = calculaAsignacion(hijos);
      double descuento = calculaDescuento(sueldoBase);
      double neto = calculaNeto(sueldoBase,movilidad,asignacion,descuento);

      //Imprimir en la lista
     imprimir(años, sueldoBase, movilidad, asignacion, descuento, neto);
    }
    else
     {
       MessageBox.Show("El empleado ya se encuentra registra","Planilla");
       return;
     }
    }
    else
    {
        MessageBox.Show("El error se encuentra en "+valida(),"Planilla",
                        MessageBoxButtons.OK,MessageBoxIcon.Error);
    }
}

private void btnConsulta1_Click(object sender, EventArgs e)
{
       totalEmpleadosxArea();

       lvEstadisticas.Items.Clear();
       string[] elementosFila = new string[2];
       ListViewItem row;

       elementosFila[0] = "Total de Personal del area de Ventas";
       elementosFila[1] = tVentas.ToString("0.00");
       row = new ListViewItem(elementosFila);
       lvEstadisticas.Items.Add(row);

       elementosFila[0] = "Total de Personal del area de Marketing";
       elementosFila[1] = tMarketing.ToString("0.00");
```

```
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Total de Personal del area de Logistica";
            elementosFila[1] = tLogistica.ToString("0.00");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Total de Personal del area de Prestamos";
            elementosFila[1] = tPrestamo.ToString("0.00");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        private void btnConsulta3_Click(object sender, EventArgs e)
        {
            montoAcumuladoxArea();

            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[2];
            ListViewItem row;

            elementosFila[0] = "Monto neto acumulado del area de Ventas";
            elementosFila[1] = aVentas.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto neto acumulado del area de Marketing";
            elementosFila[1] = aMarketing.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto neto acumulado del area de Logistica";
            elementosFila[1] = aLogistica.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto neto acumulado del area de Prestamos";
            elementosFila[1] = aPrestamo.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        private void btnConsulta2_Click(object sender, EventArgs e)
        {
          montoAcumuladorxCondicion();

          lvEstadisticas.Items.Clear();
          string[] elementosFila = new string[2];
          ListViewItem row;
```

```
            elementosFila[0] = "Monto neto acumulado por empleados tipo Personal";
            elementosFila[1] = aPersonal.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);

            elementosFila[0] = "Monto neto acumulado por empleados tipo Comision";
            elementosFila[1] = aComision.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            DialogResult r = MessageBox.Show("Esta seguro de salir?",
                                             "Planilla",
                                             MessageBoxButtons.YesNo,
                                             MessageBoxIcon.Exclamation);
            if (r == DialogResult.Yes) this.Close();
        }

        //Metodos de captura de valor
        string getEmpleado()
        {
            return txtEmpleado.Text;
        }

        int getHijos()
        {
            return int.Parse(txtHijos.Text);
        }

        string getArea()
        {
            return cboArea.Text;
        }

        string getCondicion()
        {
            return cboCondicion.Text;
        }

        DateTime getFecha()
        {
            return dtFechaIng.Value;
        }

        //Metodos para calcular
        //Metodo que determina el sueldo
        double asignaSueldoBase(string area, string condicion)
        {
            switch (area)
            {
```

```
            case "Ventas":
                if (condicion == "Personal")
                    return 2500;
                else
                    return 500;
            case "Marketing":
                if (condicion == "Personal")
                    return 1800;
                else
                    return 1100;
            case "Logistica":
                if (condicion == "Personal")
                    return 3500;
                else
                    return 2500;
            case "Prestamo":
                if (condicion  == "Personal")
                    return 1500;
                else
                    return 900;
        }
        return 0;
    }

    //Calcular Años de servicio
    int calculaTiempoServicio()
    {
        return DateTime.Now.Year - getFecha().Year;
    }

    //Calcular la asignacion de movilidad
    double calculaMovilidad(double sueldoBase)
    {
        switch (getArea())
        {
            case "Ventas":
                if (getCondicion() == "Personal")
                    return 10.0/100 * sueldoBase;
                else
                    return 5.0 / 100 * sueldoBase;
            case "Marketing":
                if (getCondicion() == "Personal")
                    return 20.0 / 100 * sueldoBase;
                else
                    return 10.0 / 100 * sueldoBase;

            case "Logistica":
                if (getCondicion() == "Personal")
                    return 30.0 / 100 * sueldoBase;
                else
                    return 15.0 / 100 * sueldoBase;
            case "Prestamo":
                if (getCondicion() == "Personal")
                    return 25.0 / 100 * sueldoBase;
```

```
                else
                    return 12.5 / 100 * sueldoBase;
            }
            return 0;
        }

        //Metodo para calcular la asignacion por numero de hijos
        double calculaAsignacion(int hijos)
        {
            return hijos * 20;
        }

        double calculaDescuento(double sueldoBase)
        {
            return 17.0 / 100 * sueldoBase;
        }

        double calculaNeto(double sueldoBase, double movilidad,
                           double asignacion, double descuento)
        {
            return sueldoBase+movilidad+asignacion-descuento;
        }

        //Metodo para imprimir
        void imprimir(int tiempo, double sueldoBase, double movilidad,
                      double asignacion, double descuento, double neto)
        {
            ListViewItem fila = new ListViewItem(getEmpleado());
            fila.SubItems.Add(getHijos().ToString());
            fila.SubItems.Add(getArea());
            fila.SubItems.Add(getCondicion());
            fila.SubItems.Add(tiempo.ToString());
            fila.SubItems.Add(sueldoBase.ToString("0.00"));
            fila.SubItems.Add(movilidad.ToString("0.00"));
            fila.SubItems.Add(asignacion.ToString("0.00"));
            fila.SubItems.Add(descuento.ToString());
            fila.SubItems.Add(neto.ToString("0.00"));
            lvEmpleados.Items.Add(fila);
        }

        //Metodos para las estadisticas
        //CONSULTA 1: Total de Empleados por Areas
        public void totalEmpleadosxArea()
        {
            tVentas = 0;
            tMarketing = 0;
            tLogistica = 0;
            tPrestamo = 0;

            for (int i=0; i < lvEmpleados.Items.Count; i++)
            {
                string area = lvEmpleados.Items[i].SubItems[2].Text;
                switch (area)
                {
```

```
                    case "Ventas": tVentas++; break;
                    case "Marketing": tMarketing++; break;
                    case "Logistica": tLogistica++; break;
                    case "Prestamo": tPrestamo++; break;
                }
            }
        }

        //CONSULTA 2: Monto acumulado de netos por condicion
        public void montoAcumuladorxCondicion()
        {
          aPersonal = 0;
          aComision = 0;
          for (int i = 0; i < lvEmpleados.Items.Count; i++)
          {
            string condicion= lvEmpleados.Items[i].SubItems[3].Text;
            switch (condicion)
            {
              case "Personal":
                aPersonal+=double.Parse(lvEmpleados.Items[i].SubItems[9].Text);
                break;
              case "Comision":
                aComision += double.Parse(lvEmpleados.Items[i].SubItems[9].Text);
                break;
            }
          }
        }

        //CONSULTA 3: Monto acumulado de netos por area
        public void montoAcumuladoxArea()
        {
            aVentas = 0;
            aMarketing= 0;
            aLogistica = 0;
            aPrestamo = 0;

            for (int i = 0; i < lvEmpleados.Items.Count; i++)
            {
                string area = lvEmpleados.Items[i].SubItems[2].Text;
                switch (area)
                {
                  case "Ventas":
                    aVentas+=double.Parse(lvEmpleados.Items[i].SubItems[9].Text);
                    break;
                  case "Marketing":
                    aMarketing+=double.Parse(lvEmpleados.Items[i].SubItems[9].Text);
                    break;
                  case "Logistica":
                    aLogistica+=double.Parse(lvEmpleados.Items[i].SubItems[9].Text);
                    break;
                  case "Prestamo":
                    aPrestamo+=double.Parse(lvEmpleados.Items[i].SubItems[9].Text);
                    break;
                }
            }
        }
```

```
//Metodo que valida el registro duplicado del empleado
bool validaEmpleado()
{
    bool estado=false;
    for (int i = 0; i < lvEmpleados.Items.Count; i++)
    {
        string empleado = lvEmpleados.Items[i].SubItems[0].Text;

        if (getEmpleado() == empleado) estado = true;
    }
    return estado;
}
//Metodo de validacion de datos
string valida()
{
    if (txtEmpleado.Text.Trim().Length == 0)
    {
        txtEmpleado.Focus();
        return "nombre del empleado";
    }
    else if (!Information.IsNumeric(txtHijos.Text))
    {
        txtHijos.Text = "";
        txtHijos.Focus();
        return "numero de hijos; es un valor numero..!!";
    }
    else if (cboArea.SelectedIndex == -1)
    {
        cboArea.Focus();
        return "area del empleado";
    }
    else if (cboCondicion.SelectedIndex == -1)
    {
        cboCondicion.Focus();
        return "condicion del empleado";
    }
    return "";
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Asignar variables globales para los contadores y acumuladores de la aplicación**

  En vista de que la aplicación integra todos los métodos vistos anteriormente, considere que en toda aplicación solo deben declararse variables globales a los contadores y acumuladores.

```
public partial class frmPlanilla : Form
    {
double tVentas, tMarketing, tLogistica, tPrestamo;
double aVentas, aMarketing, aLogistica, aPrestamo;
                    double aPersonal, aComision;
    }
```

Donde las variables que inician con "T" son usadas para determinar el total de los empleados según el área. Mientras que las variables que inician con "A" son usadas para acumular los montos netos según el área y, finalmente, "aPersonal" determina el monto acumulado de los montos netos según la condición del empleado, de la misma forma trabajará la variable "aComision".

- **Validar el ingreso de valores del formulario**

Ahora que conoce cómo implementar métodos, es necesario aplicar la metodología de validación de datos, es decir, verificar que los valores ingresados por el usuario cumplen los requerimientos básicos para la aplicación, por ejemplo, no dejar vacía la casilla de ingreso del empleado, no procesar si el usuario ingresa un texto en el número de hijos y en el caso de los cuadros combinados no escoger ningún elemento de la lista. Se estudiarán los códigos implementados en el método "Valida".

```
string valida()
        {
            if (txtEmpleado.Text.Trim().Length == 0)
            {
                txtEmpleado.Focus();
                return "nombre del empleado";
            }
            else if (!Information.IsNumeric(txtHijos.Text))
            {
                txtHijos.Text = "";
                txtHijos.Focus();
                return "numero de hijos; es un valor numero..!!";
            }
            else if (cboArea.SelectedIndex == -1)
            {
                cboArea.Focus();
                return "area del empleado";
            }
            else if (cboCondicion.SelectedIndex == -1)
            {
                cboCondicion.Focus();
                return "condicion del empleado";
            }
            return "";
        }
```

Inicie por verificar si la caja de texto no se encuentra vacía, el método "trim" permite eliminar los espacios en blanco que pudiera ingresarse en la caja de texto, mientras que "length" determina la cantidad de caracteres digitados en el caja de texto, entonces se puede decir que evalúa si la longitud ingresada quitanto los espacios en blanco es igual a cero, entonces se encuentra con un error de ingreso. Para lo cual se le envía el foco al objeto txtEmpleado para que el usuario pueda ingresar un valor y a la vez se envía como respuesta la palabra "nombre del empleado" para que se una a un texto preparado en un mensaje solo cuando ocurra dicho error. Caso contrario, sigue evaluando al siguiente control.

En el control txtHijos debe validar que el valor ingresado sea un valor numérico entero, por tanto, se usa el método "IsNumeric" el cual permite evaluar si el valor es numérico o no. Esta función viene del ensamblado "Information", y para que funcione correctamente debe agregar la referencia Microsoft.VisualBasic al proyecto (clic derecho en el proyecto > Agregar referencia > Activar el check "Microsoft.VisualBasic") y, finalmente, en el proyecto declarar la librería "Using Microsoft.VisualBasic".

Para evaluar el cuadro combinado debe especificar si la propiedad "SelectedIndex" es igual a -1, ya que los elementos que compone un cuadro combinado se almacenan desde la posición 0 hacia adelante, el -1 indica que no ha seleccionado ningún elemento de la lista.

No olvide que para indicar que todo es correcto este método no debe devolver ningún valor, por tanto, debe especificar "return con comillas vacias" para entender que todo lo ingresado es correcto.

Ahora se usará el método dentro del botón **Registrar** de forma que si la función valida devuelve vacío, entonces el proceso debe continuar, como se muestra en el siguiente código.

```
private void btnRegistrar_Click(object sender, EventArgs e)
      {
        if (valida() == "")
        {
          //Sentencias si la validación de los objetos es correcto
        }
        else
        {
          MessageBox.Show("El error se encuentra en "+valida(),"Planilla",
                          MessageBoxButtons.OK,MessageBoxIcon.Error);
        }
    }
```

Si la función valida devuelve algún valor, indica que hubo un error en algún objeto es por eso que se implementa un mensaje que permita concatenar la palabra "El error se encuentra en..." y el mensaje que devuelve el método "valida".

- **Validar el ingreso duplicado del nombre del empleado**

  Como se trata de una aplicación de registro de planilla en un determinado mes, no debe permitirse el ingreso duplicado del empleado, es decir, el nombre que intente registrar no debe encontrarse en la lista, si es así, debe mostrar un mensaje de error y anular el ingreso, el código de validación de nombre de usuario se muestra a continuación:

```
bool validaEmpleado()
        {
            bool estado=false;
            for (int i = 0; i < lvEmpleados.Items.Count; i++)
            {
                string empleado = lvEmpleados.Items[i].SubItems[0].Text;

                if (getEmpleado() == empleado) estado = true;
            }
            return estado;
        }
```

Opte por asignar un valor de salida booleana del método (true=si hay duplicado | false=-no hay duplicado), dentro del método se declara la variable local "estado" con el valor false, de forma que si no encuentra duplicado dicho valor será la salida del método. Por otro lado, debe recorrer todos los registros del empleado para comparar si el valor ingresado ya se encuentra dentro de la lista, por eso, la estructura for recorre desde la posición 0 de la lista hasta el último registro (LvEmpleados.Items.Count-1), debe tener en cuenta que los nombres de los elementos se encuentran registrados en la columna 0 de la lista, así que para compararlos se necesita un if que verifique "lvEmpleados. Items[i].SubItem[0]" con el nombre del empleado capturado desde el método getEmpleado, si esto coincide entonces el estado cambia a "true" porque se ha encontrado una duplicidad de nombres.

- **Calcular el tiempo de servicio del empleado**

  Primero obtenga el año actual, use la función DateTime.Now.Year, donde Now representa la fecha y la hora actual, mientras que Year obtiene solo el año de dicha fecha. Y para obtener el año de la fecha ingresado se debe tener en cuenta que dicha variable debe ser de tipo DateTime, ya que de otro tipo de datos no se puede obtener dicho año.

Caso desarrollado 6 : lambda - control de registro de libros

La empresa editora Books-Now, dedicada a la distribución de libros a nivel nacional, ha decidido implementar un aplicación que permita registrar los libros de una adquisición realizada, para ello debe generar un número de registro, registrar el título, categoría y costo del libro para determinar el descuento y el precio de venta de dicho libro.

Se debe tener en cuenta:

- Usar los métodos necesarios para la aplicación.
- Al iniciar la aplicación se debe mostrar el número autogenerado usanda la expresión lambda.
- La asignación del precio real del libro se da al momento de registrar el costo del mismo.
- Para obtener el descuento se debe implementar una expresión lambda que aplica descuentos según la siguiente tabla:

Categoría	Descuento
Gestión	10 %
Ingeniería	12 %
Programación	20 %
Base de datos	15 %

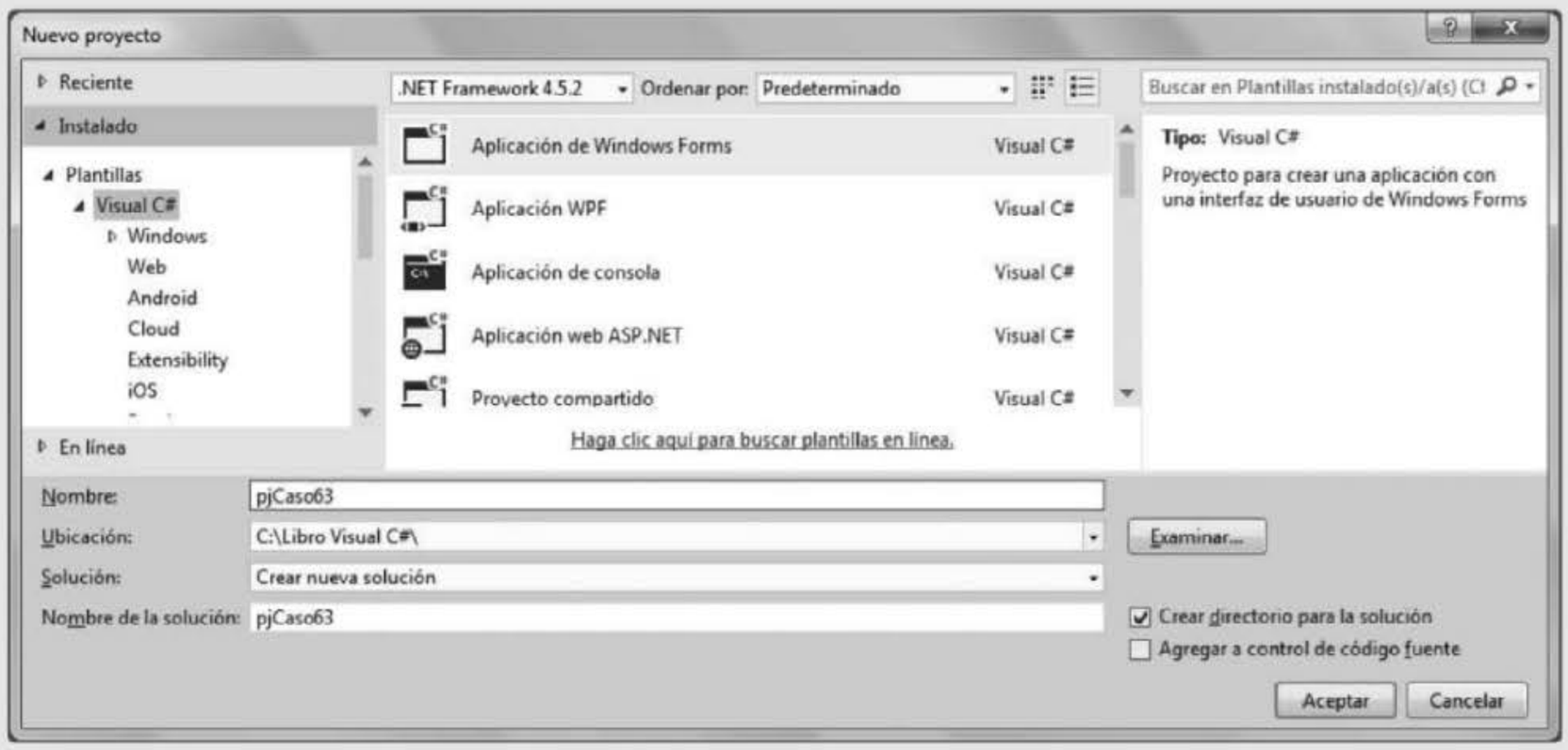
- El precio de venta resulta de la diferencia entre el precio real y el descuento, por lo tanto, debe implementarse en una expresión lambda.
- Con respecto a las estadísticas debe mostrar el total de montos acumulados de descuento y quien es el libro con precio de venta más caro.

Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

   - Seleccione **Archivo > Nuevo > Proyecto**.
   - Seleccione la plantilla **Visual C#**.
   - Seleccione **Aplicación de Windows Form**.

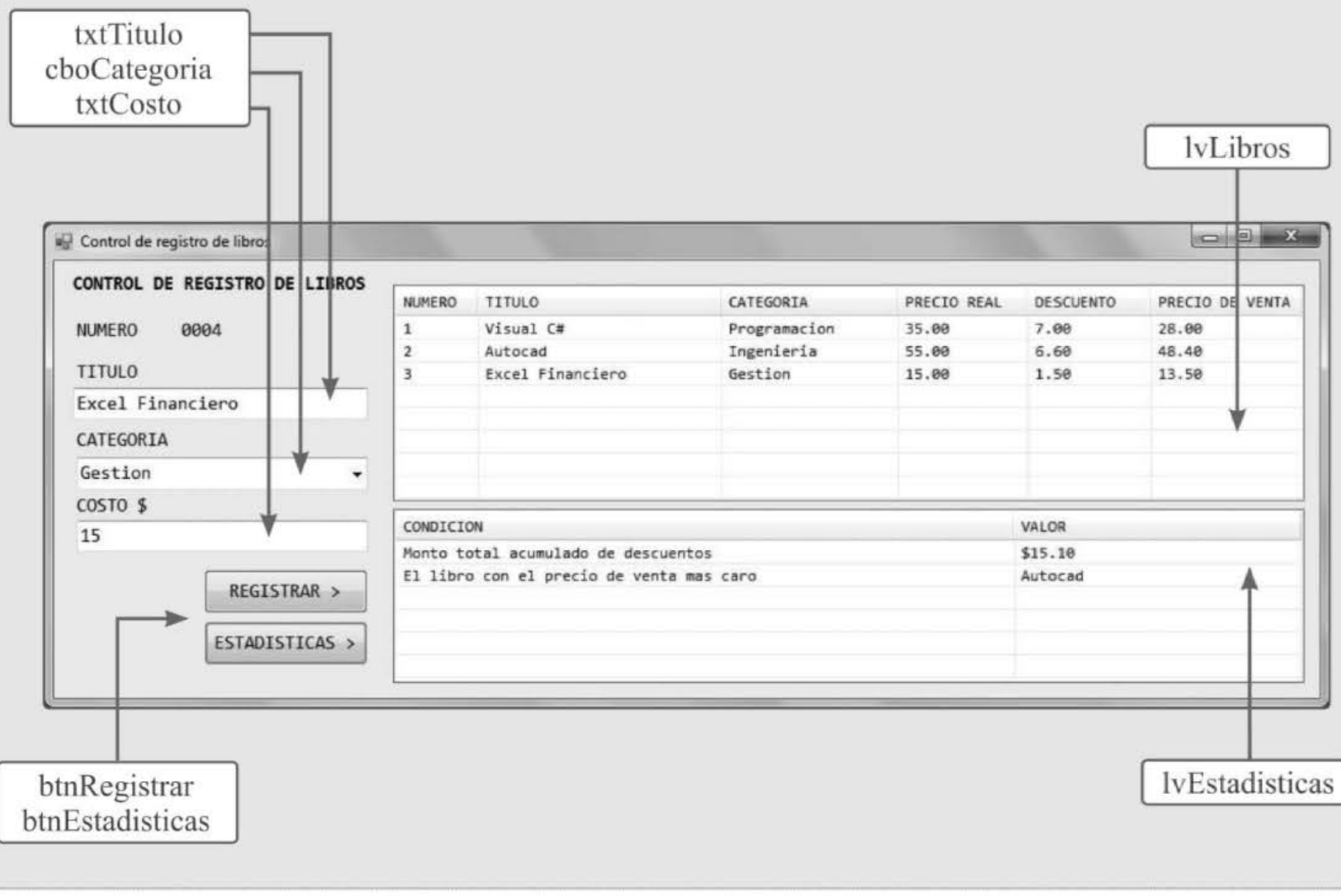
- Asigne el nombre **pjCaso63** como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
- Desde la propiedad **Nombre de archivo** asigne el nombre **frmLibros**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmLibros:

Clase	N.º elementos	Controles
Form	01	Form1
Label	05	Label1, Label2, Label3, Label4, Label5
TextBox	02	Textbox1, Textbox2
ComboBox	01	ComboBox1
Button	02	Button1, Button2
ListView	02	ListView1, ListView2
TOTAL	13	

4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmLibros Control de registro de libros
Label1	Text	CONTROL DE REGISTRO DE LIBROS
Label2	Text	NUMERO
Label3	(Name) Text	lblNumero lblNumero
Label4	text	CATEGORIA
Label5	Text	COSTO $
Textbox1	(Name)	txtTitulo
Textbox2	(Name)	txtCosto
ComboBox1	(Name) Items	cboCategoria Gestion, Ingenieria, Programacion, Base de datos
Button1	(Name) Text	btnRegistrar REGISTRAR >
Button2	(Name) Text	Btnestadisticas ESTADISTICAS >
ListView1	(Name) GridLines	lvLibros True
ListView2	(Name) GridLines	lvEstadisticas True

5. Código completo de la aplicación:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso63
{
    public partial class frmLibros : Form
    {
        static int contador;

        public frmLibros()
        {
            InitializeComponent();
        }

        private void frmLibros_Load(object sender, EventArgs e)
        {
            lblNumero.Text =generaNumero();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
                //Capturando los datos del formulario
                double costo = getCosto();
                string categoria = getCategoria();
                double descuento = asignaDescuento(categoria, costo);
                double precioVenta = calculaPrecioVenta(costo, descuento);

                //Enviando a la impresion
                imprimirRegistro(descuento, precioVenta);

                lblNumero.Text = generaNumero();
            }
            else
                MessageBox.Show("El error se encuentra en "+valida());
        }

        private void btnEstadisticas_Click(object sender, EventArgs e)
        {
```

```
        double totalDescuentos = calculaTotalDescuentos();
        string libroAlto = libroMasAlto();

        imprimirEstadisticas(totalDescuentos,libroAlto);
    }

    //Implementacion de expresiones Lambda
    Func<string> generaNumero= () =>
      {
          contador++;
          return contador.ToString("0000");
      };

    //Metodos que capturan los valores
    int getNumero()
    {
        return int.Parse(lblNumero.Text);
    }
    string getTitulo()
    {
        return txtTitulo.Text;
    }

    double getCosto()
    {
        return double.Parse(txtCosto.Text);
    }

    string getCategoria()
    {
        return cboCategoria.Text;
    }

    Func<string, double, double> asignaDescuento = (categoria, costo) =>
    {
        double descuento = 0;
        switch (categoria)
        {
            case "Gestion": descuento = 10.0/100 * costo; break;
            case "Ingenieria": descuento = 12.0 / 100 * costo; break;
            case "Programacion": descuento = 20.0 / 100 * costo; break;
            case "Base de datos": descuento = 15.0 / 100 * costo; break;
        }
        return descuento;
    };

    Func<double, double, double> calculaPrecioVenta =
                                        (costo,descuento) =>costo-descuento;
```

```
        //Calculando las estadisticas
        //Monto total de descuentos
        double calculaTotalDescuentos()
        {
            double total = 0;
            for (int i = 0; i < lvLibros.Items.Count; i++)
            {
                total += double.Parse(lvLibros.Items[i].SubItems[4].Text);
            }
            return total;
        }

        //Libro con el precio total mas alto
        string libroMasAlto()
        {
           double mayor = double.Parse(lvLibros.Items[0].SubItems[5].Text);
            int posicion=0;
            for (int i = 0; i < lvLibros.Items.Count; i++)
            {
               if (double.Parse(lvLibros.Items[i].SubItems[5].Text)> mayor)
                {
                    posicion = i;
                }
            }
            return lvLibros.Items[posicion].SubItems[1].Text;
        }

        //Imprimir el registro de ventas
        void imprimirRegistro(double descuento, double precioVenta)
        {
            ListViewItem fila = new ListViewItem(getNumero().ToString());
            fila.SubItems.Add(getTitulo());
            fila.SubItems.Add(getCategoria());
            fila.SubItems.Add(getCosto().ToString("0.00"));
            fila.SubItems.Add(descuento.ToString("0.00"));
            fila.SubItems.Add(precioVenta.ToString("0.00"));
            lvLibros.Items.Add(fila);
        }

        //Imprimir estadisticas
        void imprimirEstadisticas(double totalDescuentos, string LibroAlto)
        {
            lvEstadisticas.Items.Clear();
            string[] elementosFila = new string[2];
            ListViewItem row;

            elementosFila[0] = "Monto total acumulado de descuentos";
            elementosFila[1] = totalDescuentos.ToString("C");
            row = new ListViewItem(elementosFila);
            lvEstadisticas.Items.Add(row);
```

```
        elementosFila[0] = "El libro con el precio de venta mas caro";
        elementosFila[1] = LibroAlto;
        row = new ListViewItem(elementosFila);
        lvEstadisticas.Items.Add(row);
    }

    //Metodo de validacion de ingreso de datos
    string valida()
    {
        if (txtTitulo.Text.Trim().Length == 0)
        {
            txtTitulo.Focus();
            return "titulo del libro";
        } else if (cboCategoria.SelectedIndex == -1)
        {
            cboCategoria.Focus();
            return "categoria del libro";
        }else if (txtCosto.Text.Trim().Length == 0)
        {
            txtCosto.Focus();
            return "costo del libro";
        }
        return "";
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Implementar la expresión lambda para generar un número de manera automática**

  La expresión lambda inicia con la palabra reservada "Func" seguido se indica el tipo de datos de salida, se le ha declarado como string para que el número generado tenga el siguiente aspecto "0001", como la expresión no necesita parámetros se deja vacío los paréntesis.

```
Func<string> generaNumero= () =>
        {
            contador++;
            return contador.ToString("0000");
        };
```

La variable contador es la encargada de generar los números consecutivos, por ello, debe declararse en la sección global de la aplicación, la diferencia con otros tipos de variables es que esta variable debe declararse de static, ya que de otra manera genera un error por el uso de la expresión lambda. Finalmente, la expresión debe enviar un valor resultante mediante la sentencia return. Los parámetros "0000" se espefican para colocar ceros a la izquierda en la generación del número.

- **Implementar la expresión lambda para asignar un descuento según la categoría**

  La asignación de descuentos se realiza sobre el costo y la evaluación de la categoría, por tal motivo la expresión lambda tiene dos parámetros de entrada y uno de salida es así que en su especificación se realiza "string, double, double", string para la categoría, double para el costo y el último double para el descuento resultante de la expresión.

```
Func<string, double, double> asignaDescuento = (categoria, costo) =>
        {
            double descuento = 0;
            switch (categoria)
            {
                case "Gestion": descuento = 10.0/100 * costo; break;
                case "Ingenieria": descuento = 12.0 / 100 * costo; break;
                case "Programacion": descuento = 20.0 / 100 * costo; break;
                case "Base de datos": descuento = 15.0 / 100 * costo; break;
            }
            return descuento;
        };
```

- **Implementar la expresión lambda para calcular el precio de venta**

  Es la expresión más corta que se puede implementar en lambda ya que solo será cuestión de restar el descuento al costo real de la siguiente forma:

```
Func<double,double,double>calculaPrecioVenta=(costo,descuento)=>costo-descuento;
```

Como nota el primer double representa el costo, el segundo representa al descuento y último el resultado de la función. Por ser una expresión directa y corta no necesita llaves ni la instrucción return.

Para recordar

- El ámbito de las variables hace referencia al lugar donde se declaran las variables, de acuerdo al ámbito puede acceder a la informacion registrada en una variable, se tiene las variables locales y globales:

Ámbito	Descripción
Local	Las variables declaradas como locales solo pueden ser usadas dentro del proceso declarado. Una vez finalizado el proceso la variable local se destruye.
Global	Las variables declaradas como globales pueden ser accedidas desde cualquier proceso ya que al ser consideradas globales su valor podrá ser usado en cualquier parte de la aplicación.

- La programación modular es considerada como un paradigma de la programación el cual consiste en dividir una aplicación en módulos, subprogramas o porciones de código con el fin de hacerlo más legible y manejable. Se presenta históricamente como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos de lo que esta puede resolver.

- Método sin valor de retorno sin parámetros, se define como un método que no recibe parámetros para su funcionamiento, normalmente opera con variables globales. El formato de implementación es como sigue:

```
void metodo(){
    //Declaraciones de variables locales
    //Implementación del método
}
```

- Método sin valor de retorno con parámetros, se define como un método que recibe parámetros para su funcionamiento, es decir, dichos parámetros serán usados dentro del método, es posible que dicesteho método llene de valores a variables globales ya que aún no devuelve valor alguno.

```
void metodo(PARAMETRO){
    //Declaraciones de variables locales
    //Implementación del método
 }
```

- Método con valor de retorno sin parámetros, se define como un método devuelve un valor a quien lo invoque pero probablemente use variables globales dentro de su método ya que no cuenta con parámetros. Estos métodos se caracterizan porque siempre se especifica al inicio el tipo de datos de su valor de salida.

```
tipo metodo(){
    //Declaraciones de variables locales
    //Implementación del método
}
```

- Método con valor de retorno con parámetros, se define como un método que cuenta con parámetros que le servirá en su implementación y asimismo podrá devolver información a quien lo invoque.

```
tipo metodo(PARAMETROS){
    //Declaraciones de variables locales
    //Implementación del método
}
```

- Las expresiones lambda son consideradas como métodos ligeros ya que implementan métodos cortos tanto para el desarrollador como para el compilador. Son también llamados métodos anónimos.

```
Func <tEntrada, tSalida>  NombreExpresionLambda =
(aEntrada, aSalida) => variable = valor;
```

Preguntas frecuentes

➤ **¿Cuál es la diferencia entre una variable local y global?**

Una variable local es creada siempre dentro de un proceso, cuando este finaliza la variable también finaliza y se podrá reutilizar en otro proceso, mientras que una variable global mantiene su valor durante la vida de la aplicación, asi se tiene que son globales los contadores y acumuladores.

➤ **¿Qué método utilizar en mis aplicaciones?**

El uso de los tipos de métodos se debe al análisis que se realice al caso, a partir de las actividades que realizará en el problema determinará qué métodos necesitará, se recomienda primero hacer una lista de actividades y estas serán condiseradas métodos, ahora, con respecto al tipo se puede mencionar que conforme avance su nivel de programación se dará cuenta qué tipo de método será necesario, lo que sí debe tener en cuenta es que en una aplicación solo serán globales las variables contadoras y acumuladores.

Capítulo 7

Programación orientada a objetos

Capacidad terminal:

Implementar aplicaciones que permitan el uso de clase y objetos.

Contenido:

- Introducción
- Características de la programación orientada a objetos
- La clase
- Nivel de visibilidad
- Los atributos de clase
- Los objetos
  - o Creación de un objeto
  - o Enviando información a los atributos de la clase por medio del objeto
  - o Usando la información de los atributos de clase por medio del objeto
- Método constructor
- Casos desarrollados
  - o Caso desarrollado 1: clase objeto - boleta de venta
  - o Caso desarrollado 2: clase objeto - planilla de empleados

Introducción

Al mirar con atención alrededor, se pueden encontrar muchos objetos fáciles de detectar y que pueden ser seleccionados y manipulados según la necesidad. De la misma manera sucede con la programación orientada a objetos, el cual usa elementos visuales que en realidad son objetos de una determinada clase, así se puede encontrar el diseño de un formulario como un contenedor de objetos. A continuación, se tiene el siguiente formulario para analizar cuáles son las clases que encontramos:



Para empezar, se encuentra la clase Form el cual lo presenta mediante el objeto form1 y tiene la misión de contener otros objetos que componen un prototipo de la aplicación. Seguidamente, todos los elementos de tipo etiqueta pertenecen a la clase Label, mientras que las cajas de texto pertenecen a la clase TextBox, los cuadros combinados pertenecen a la clase ComboBox, los botones de acciones pertenecen a la clase Button y los cuadros de lista pertenecen a la clase ListView. De esta manera, las clases son representadas en una aplicación mediante objetos que a su vez forman una aplicación visual. En este capítulo se ira más allá de la implementación de formularios o controles pues, gracias a Visual Studio el uso de los objetos se realiza arrastrando los controles desde el cuadro de herramientas; mientras que se implementan clases que instanciarán a los objetos para usarlo dentro de la aplicación, todo esto mediante código.

La programación orientada a objetos reúne un conjunto de técnicas para obtener calidad interna como medio para obtener calidad externa (reutilización y extensibilidad). Se basa de dividir un programa en pequeñas unidades lógicas de código. A estas pequeñas unidades lógicas de código se les llama objetos.

El software se organiza como una colección de objetos que contienen tanto estructura como comportamiento. Por lo tanto, un programa orientado a objetos es una colección de

clases. Necesitará una función principal que cree objetos y comience la ejecución mediante la invocación de sus funciones miembro.

Los lenguajes de programación orientados a objetos tratan a las aplicaciones como conjuntos de objetos que se ayudan entre sí para realizar acciones. Permiten que los programas sean más fáciles de escribir, mantener y reutilizar en el tiempo.

7.1 Características de la programación orientada a objetos

Las principales características que presenta la programación orientada a objetos son las siguientes:

➢ **Abstracción**

Considerada como la acción de separar las cualidades que puede presentar un objeto del mundo real, es decir, no se debe preocupar en forma detallada, sino más bien obtener los atributos necesarios de un objeto. Se puede citar como ejemplo la venta de un producto en una tienda comercial, el cliente solo debe escogerlo desde mostrador y llevarlo al área de cobranzas para realizar su pago y luego llevarlo a casa. ¿Qué elementos son abstraídos del caso expuesto?

- No se debe preocupar cómo llegó el producto a la tienda comercial.
- No se debe preocupar por los insumos que usaron para la elaboración del producto.
- No se debe preocupar por los impuestos que pagaron las empresas elaboradoras en el producto que se compró.

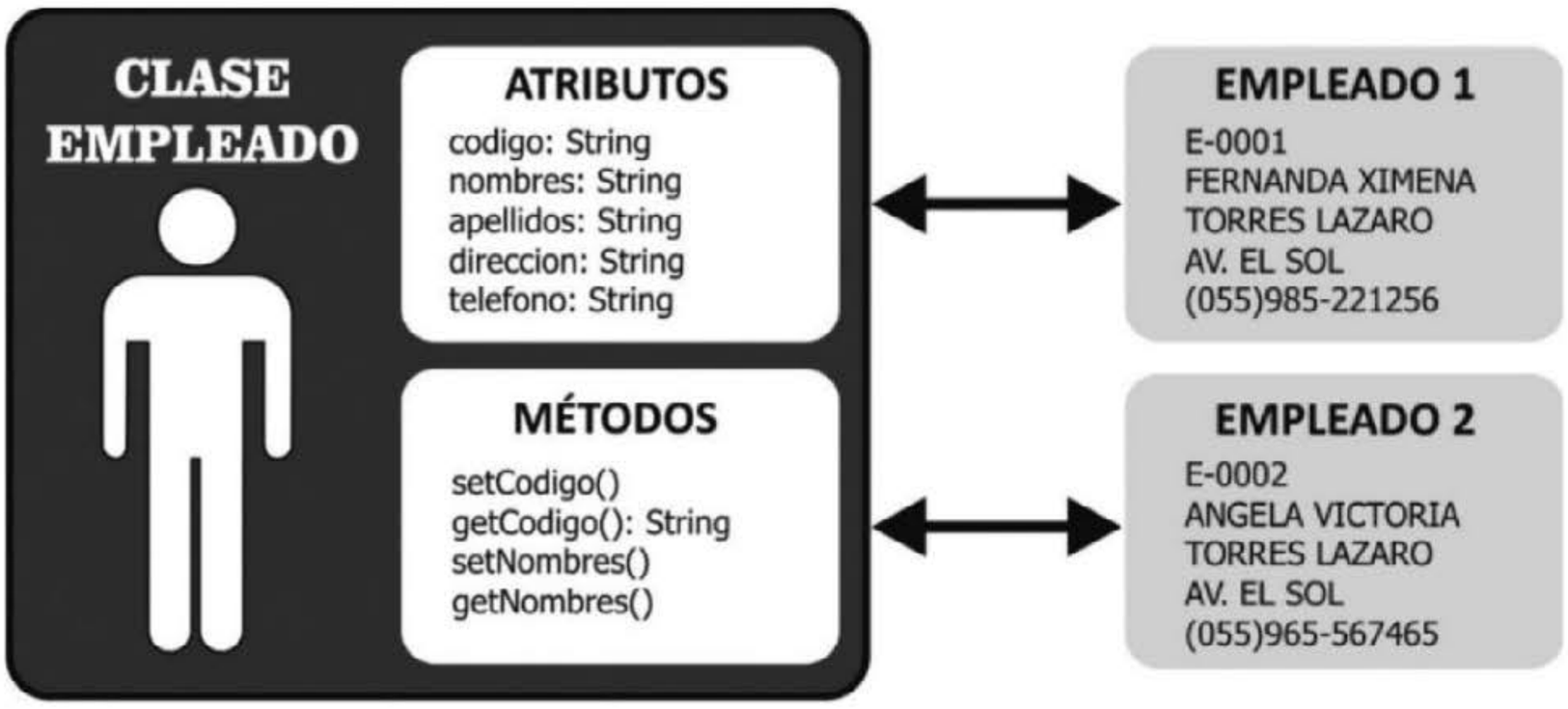
A estos elementos se les llama abstracción ya que hay elementos que sí son necesarios para la compra de dicho producto, se pueden nombrar los siguientes:

- 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**

Nivel de protección de los elementos que compone una clase ya que no es recomendado 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**

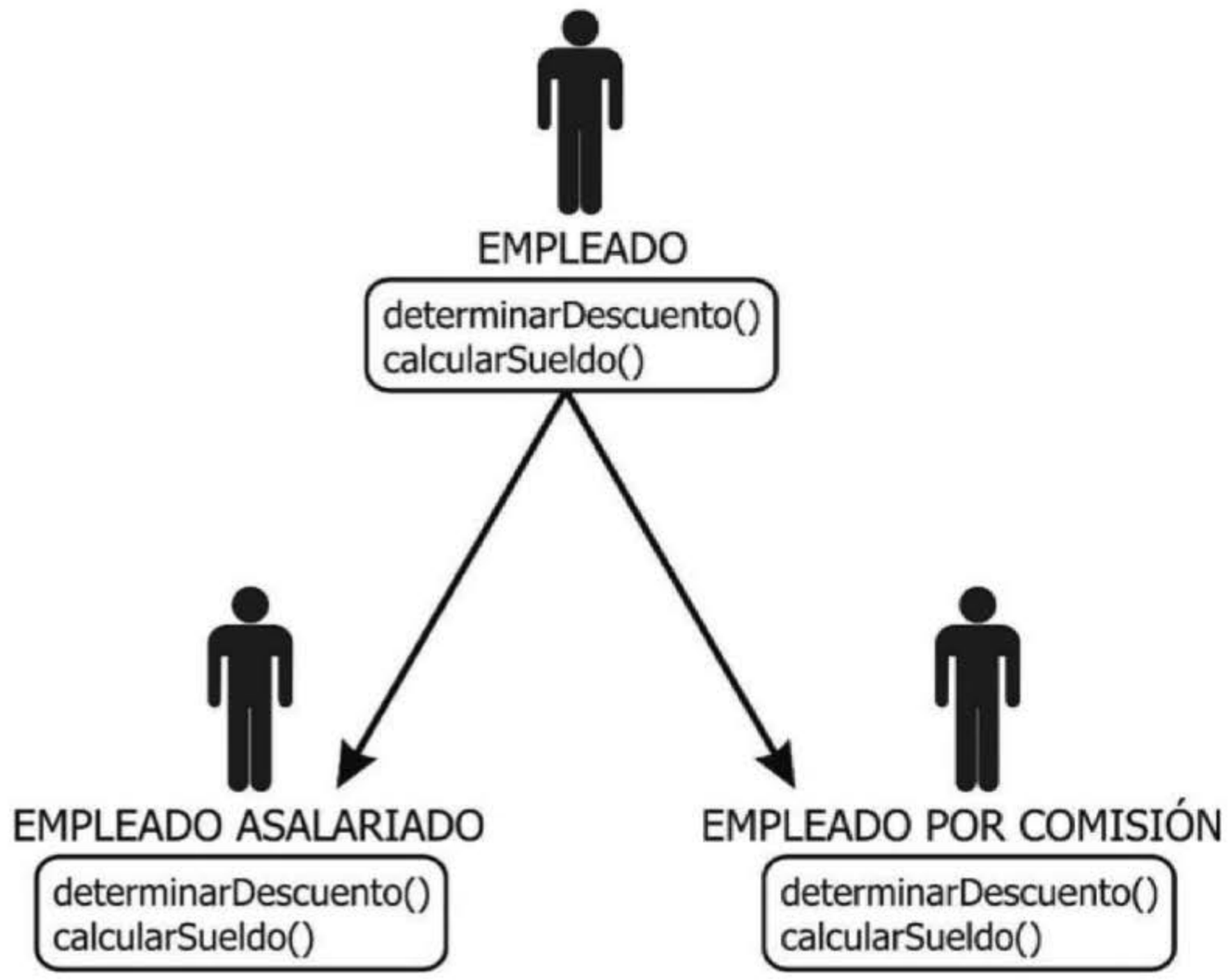
Se denomina a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos). Cada uno debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.



En la imagen anterior, se observa que para un proceso de planilla de empleados es necesario muchas actividades como calcular el pago mensual, determinar el descuento, los beneficios y el pago neto de un empleado, que se implementarán en módulos visualmente separados pero unidos a la vez para el objetivo del proceso principal.

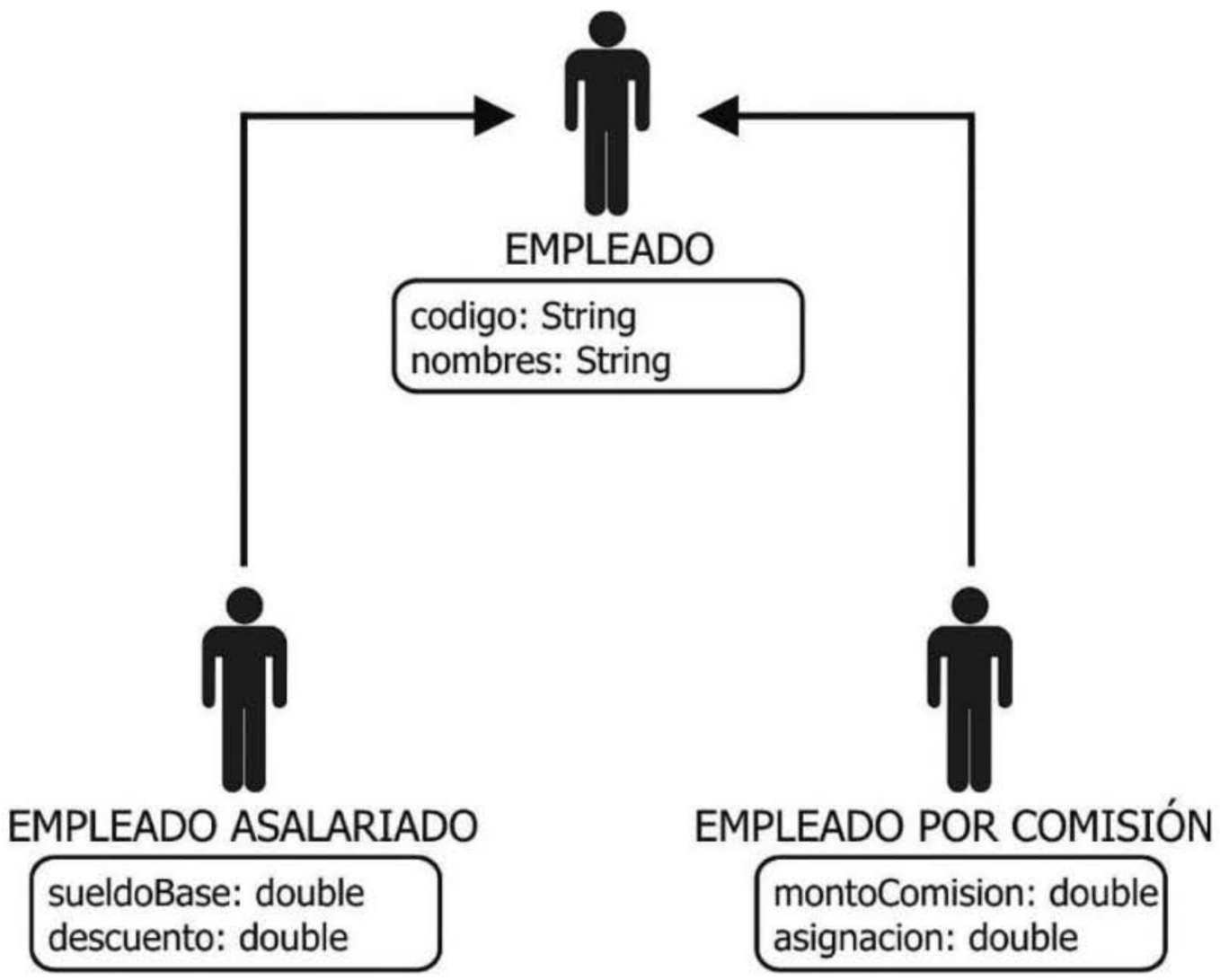
- **Polimorfismo**

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 puede nombrar la forma de calcular el sueldo de un empleado el cual se diferencia por el tipo de empleado que puede presentar una empresa, donde uno tiene un salario fijo y el otro gana por comisión, en ambos casos se debe calcular el sueldo pero se realizará 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. Una clase llamada base reúne características comunes entre otras clases y las implementa dentro de ella para luego heredarlas a las clase derivadas o hijas.



Conclusión:
Cuando se ejecuta un programa orientado a objetos, ocurren tres acciones:

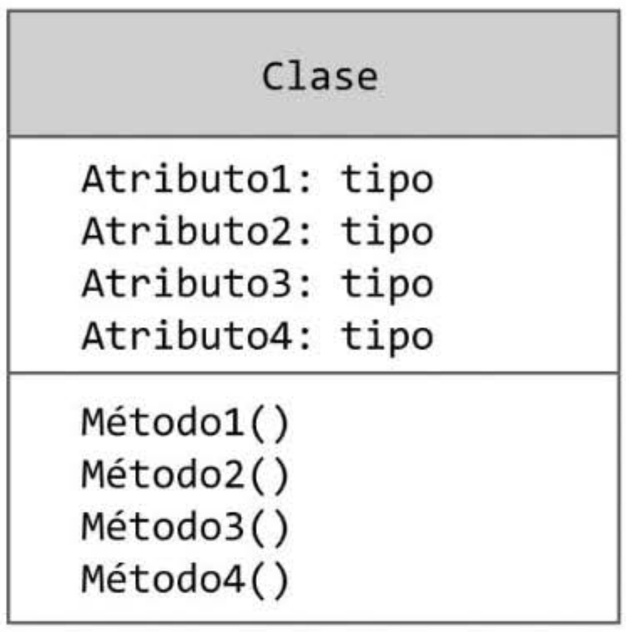
1. Se crean los objetos cuando se necesitan.
2. Los mensajes se envían desde un objeto y se reciben en otro.
3. Se borran los objetos cuando ya no son necesarios y se recupera la memoria ocupada por ellos.

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.

7.2 La clase

Para definir un concepto particular de clase primero se debe entender que la programación modular fue el rompimiento del código en porciones pequeñas de código los cuales trabajaban en integración para cumplir un determinado objetivo. La clase tiene la misión de reunir justamente esos bloques de códigos para ser usados en los diferentes objetos que se asocien a él.

Una clase es considerada como una plantilla que especifica atributos y comportamiento de un determinado tipo de objeto ya sea físico o abstracto. Finalmente, una clase en Visual C# será considerado como un nuevo tipo de datos y a partir de él se podrán crear objetos.



Se debe tener en cuenta que en Visual Studio 2015 existen muchas formas de crear una clase, supóngase que se necesita implementar la clase producto en un proceso de negocio de venta de productos:

Primera forma: Desde el menú de opciones

Dentro de una aplicación Windows Form de Visual C#:

- Seleccione el menú **Proyecto > Agregar clase**.



- La ventana aparece seleccionada el elemento Clase, solo debe asignarle el nombre a la clase, en este caso, asigne Producto, de forma que el código generado para la implementación de la clase será de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pjCasoEjemplo
{
    class Producto
    {
            //Implementacion de la clase
    }
}
```

Segunda forma: Desde la ventana Explorador de soluciones

Dentro de una aplicación Windows Form de Visual C#:

- Haga clic derecho sobre el proyecto desde la ventana **Explorador de Soluciones**.
- Seleccione **Agregar > Clase**.

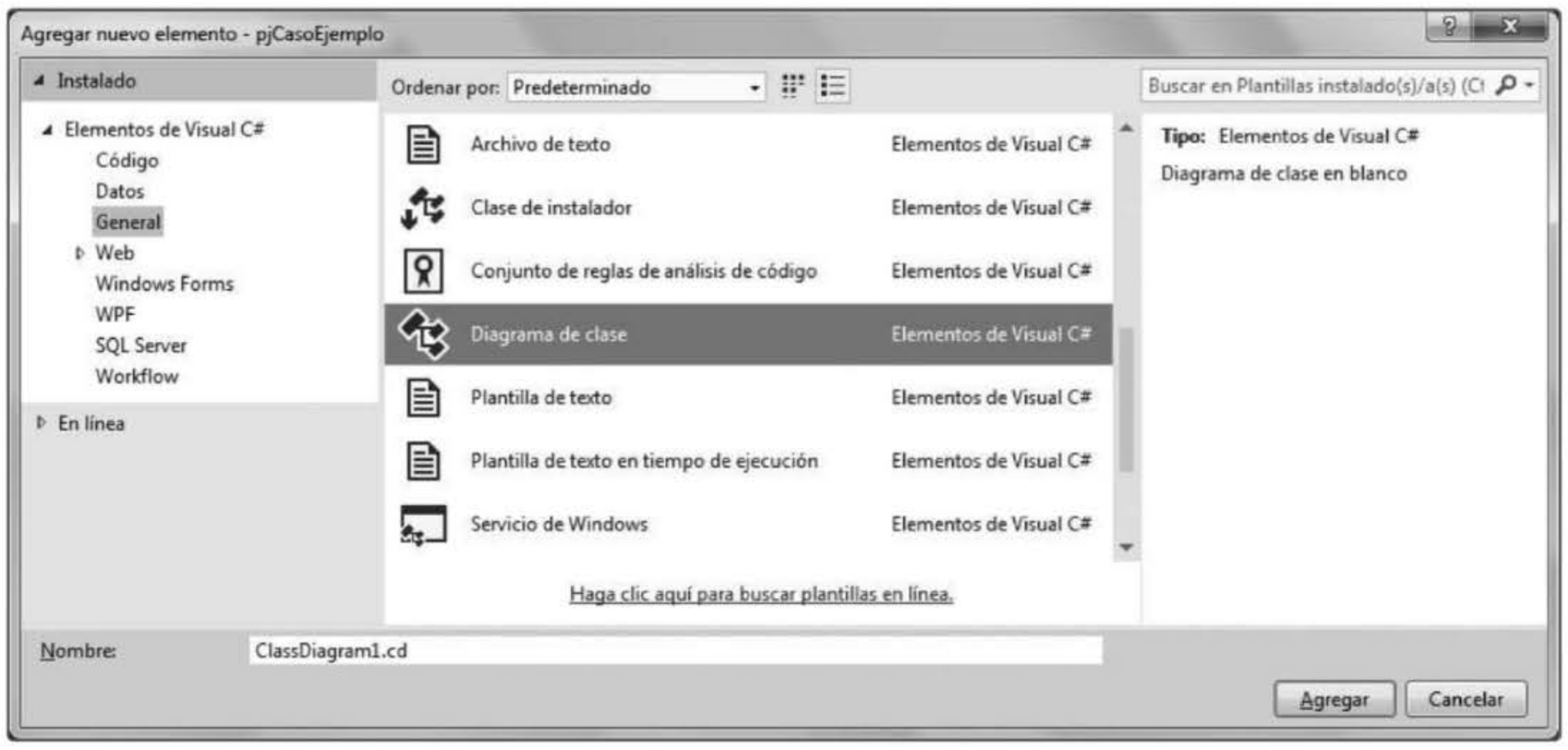


- De la misma manera, asigne el nombre de la clase y presione el botón **Agregar**.

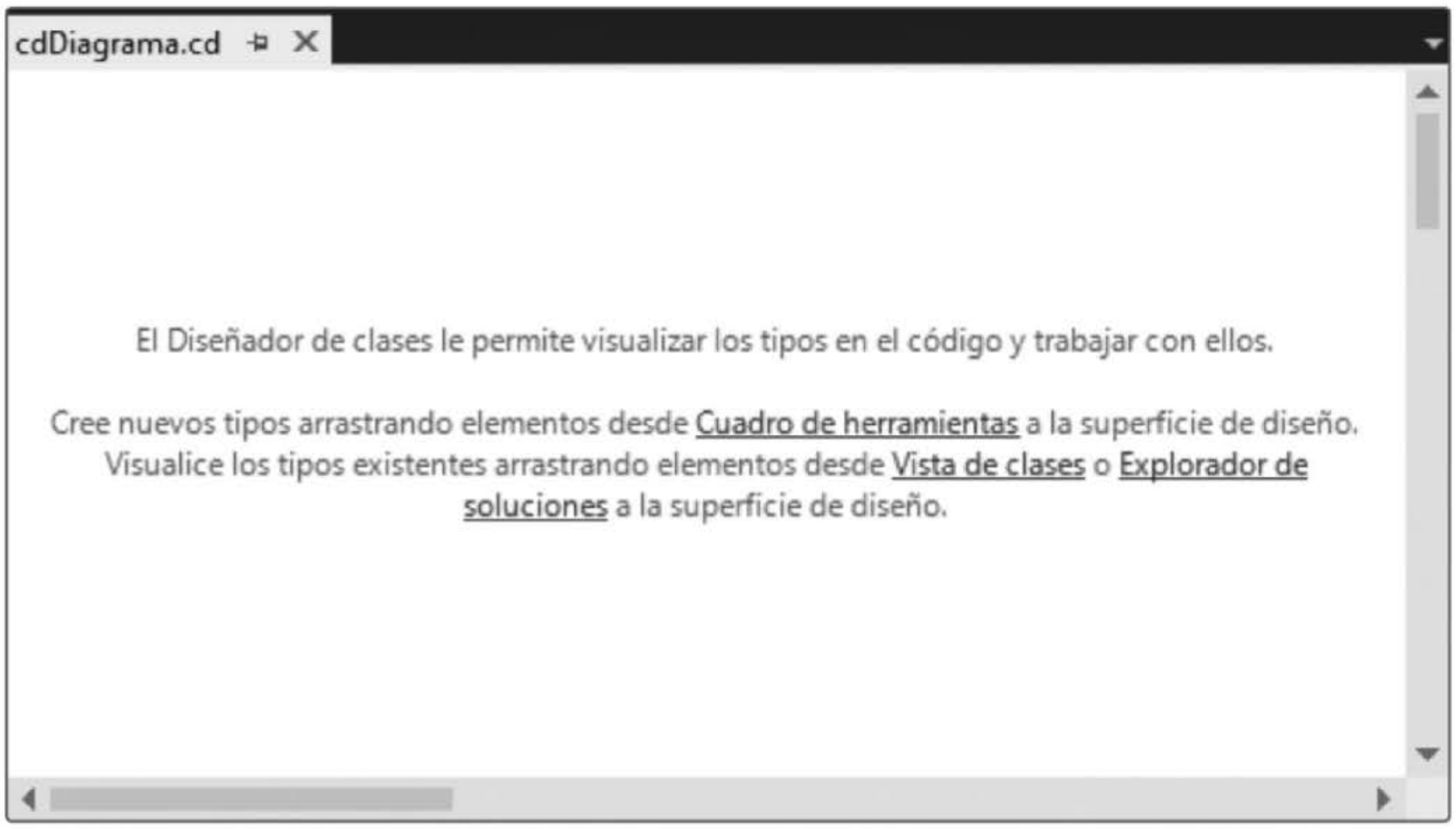
Tercera forma: Mediante un diagrama de clase

Dentro de una aplicación Windows Form de Visual C#:

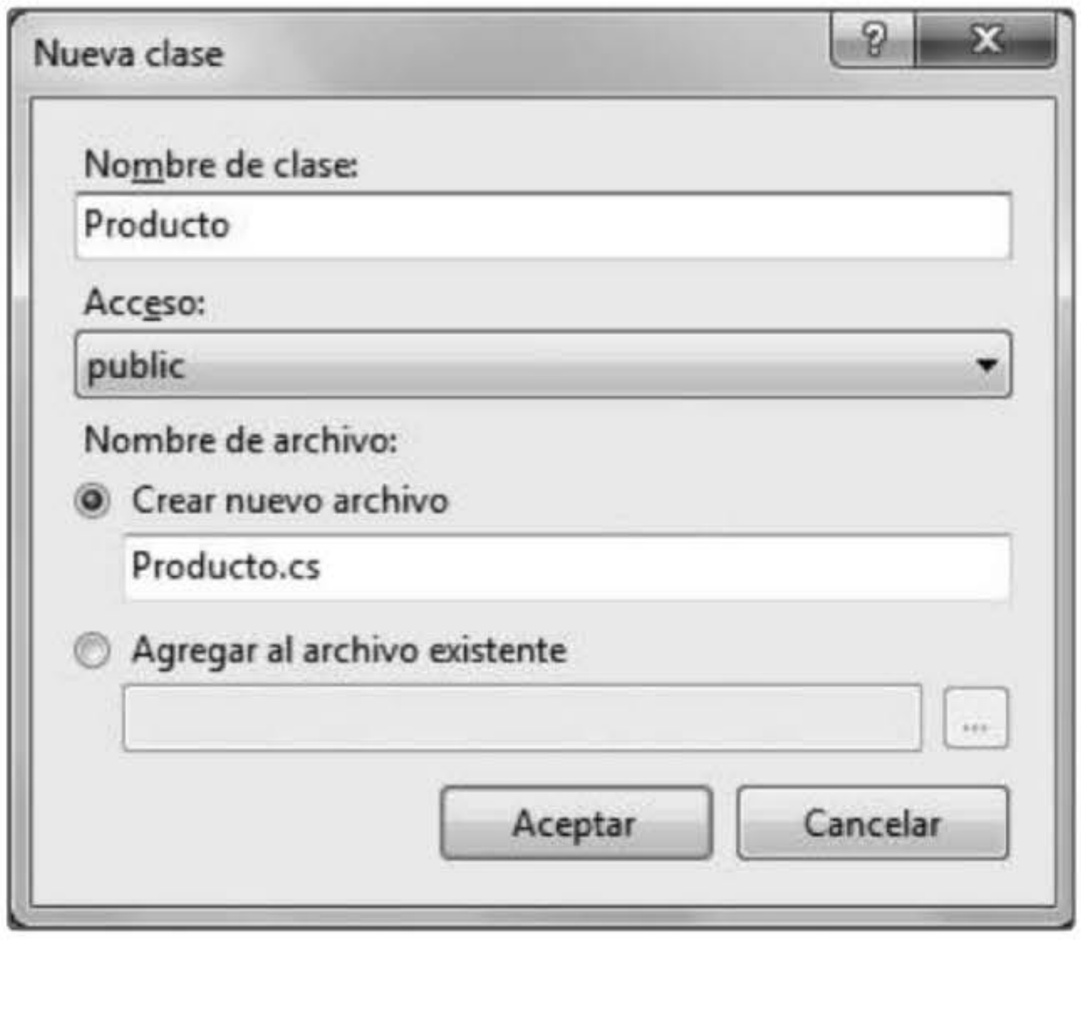
- Haga clic derecho sobre el proyecto desde la ventana **Explorador de soluciones**.
- Seleccione **Agregar > Nuevo elemento**.
- Seleccione la categoría **General > Diagrama de clase**.



- Asigne un nombre al diagrama, por ejemplo, cdDiagrama y presione el botón **Agregar**.



- Observe que aparece un entorno a partir del cual podrá 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:



- Seguro se preguntará cómo controlar la clase mediante la implementación gráfica, 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 mostrado en la primera forma como se muestra en el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace pjCasoEjemplo
{
    public class Producto
    {
            //Implementacion de la clase
    }
}
```

7.3 Nivel de visibilidad

Una clase puede controlar qué partes de un programa pueden acceder a los miembros de su clase: variables miembro y métodos miembro. Una clase bien diseñada impide el acceso directo a sus variables miembro proporcionando a cambio un conjunto de métodos de acceso que sirven como intermediarios entre las variables miembro y el mundo exterior. Esto permite controlar el uso correcto de las variables miembro, pues los métodos de acceso pueden actuar como filtros que prueben los datos que se pretenden ingresar a las variables miembro. Por otra parte, algunos métodos de la clase pueden ser necesarios solo desde el interior de la clase por lo que deben quedar restringidos solo para uso interno.

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	SI	SI	SI
Private	SI		
Protected	SI	SI	
Sin especificacion	SI		

La visibilidad de los miembros de una clase dependerá de la asociación que existe entre ellas; es decir, mediante los objetos se puede acceder o no a los miembros de una clase.

Los miembros especificados con visibilidad pública podrán ser usados por otras clases sin restricciones. Mientras que los miembros con visibilidad privada podrán ser accedidos solo en la clase donde se implementó, los miembros especificados como protected solo podrán ser accedidos por las clases heredadas. Y, finalmente, si se implementa miembros sin especificación, estos solo podrán ser accedidos solo en la clase que lo implementó como si se especificará privado al miembro.

7.4 Los atributos de clase

Los atributos representan la característica que presenta una determinada clase, estas deben definirse con un nivel de abstracción controlada por el usuario. Los atributos se encuentran asociados a clases y objetos ya que mediante ellos se accede a los miembros. A continuación, se ve cómo agregar atributos en una clase implementada en Visual Studio 2015:

Primera definición:

```
private int _atributo;
public int atributo
{
    get
    {
        return _atributo;
    }
    set
    {
        _atributo = value;
    }
}
```

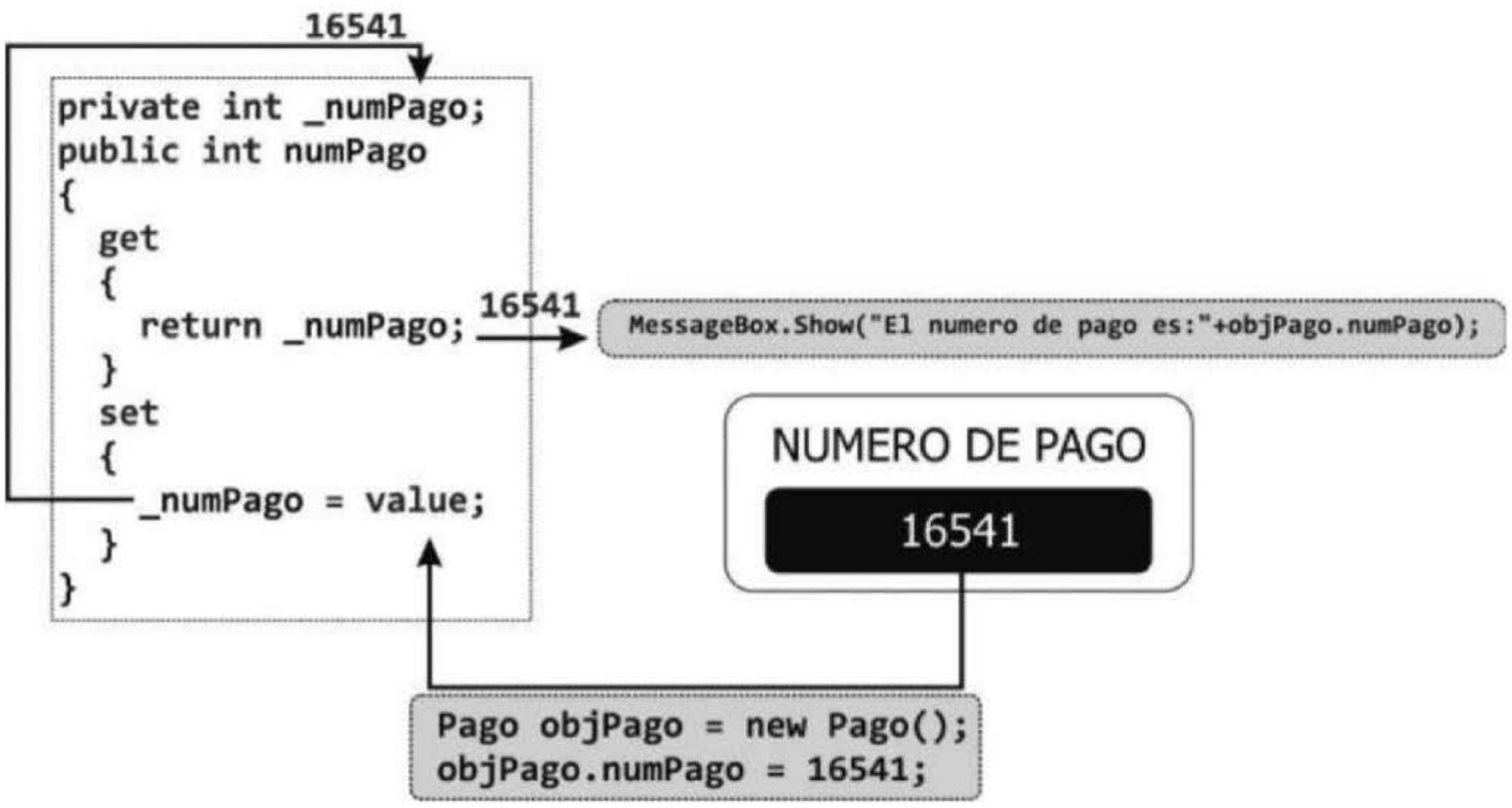
Se especifica un atributo privado que puede ser referenciado mediante un objeto por medio de los métodos get y set, la definición inicial se le debe al método set; ya que es el encargado de asignar un valor al atributo privado, mientras que get envía el valor del atributo privado asignado por set. Véase el siguiente caso:

Si tiene la clase Pago y necesita especificar un atributo llamado “numPago” de tipo entero, su implementación en la clase sería de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pjClaseEjemplo
{
```

```
public class Pago
{
    private int _numPago;
    public int numPago
    {
        get
        {
            return _numPago;
        }
        set
        {
            _numPago = value;
        }
    }
}
```



Al crear un objeto de la clase pago, 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 como se muestra en la imagen.

Segunda definición:

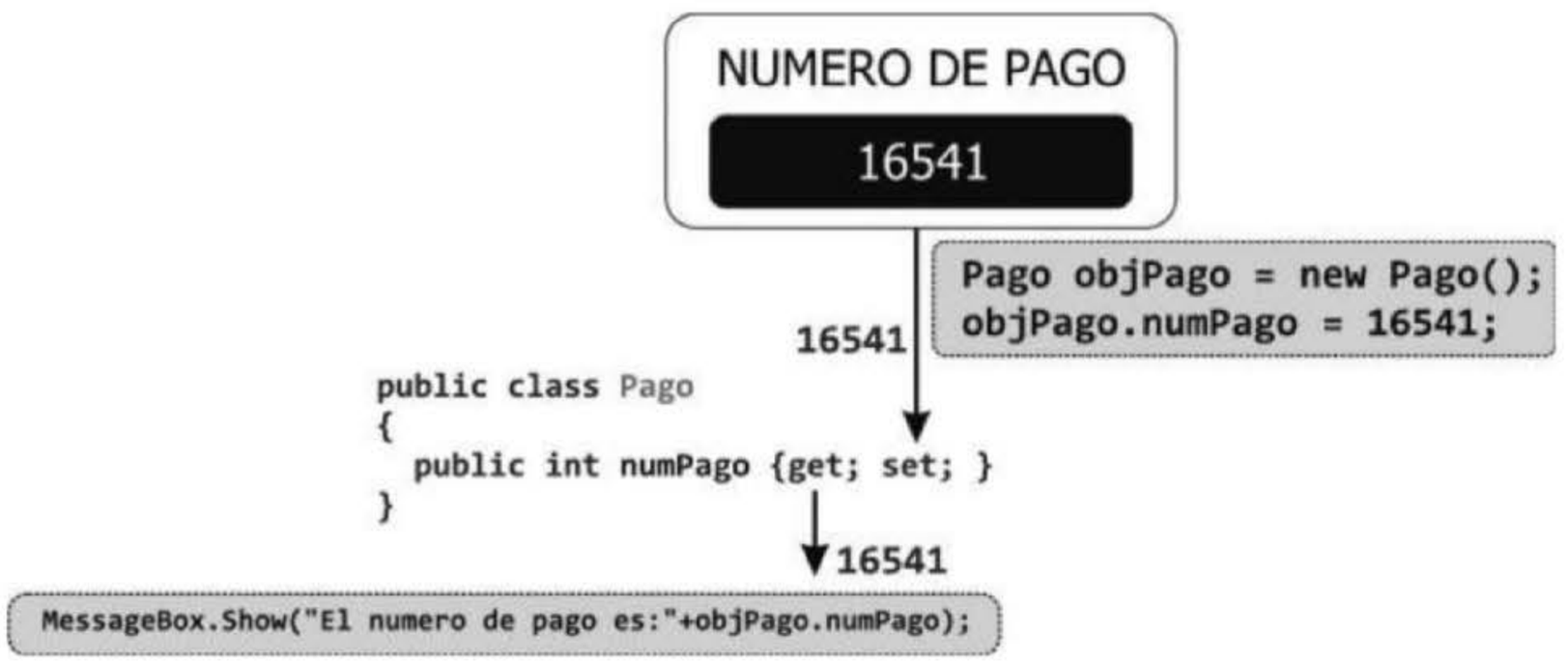
```
public int numPago { get; set; }
```

Se especifica de forma resumida las propiedades get y set de forma que en una sola línea se defina todo. Veáse el siguiente caso:

Si tiene la clase Pago y necesita especificar un atributo llamado "numPago" de tipo entero, su implementación en la clase sería de la siguiente manera:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pjClaseEjemplo
{
    public class Pago
    {
        public int numPago { get; set; }
    }
}
```



Al igual que la primera definición, el valor llega a la especificación set y cuando se requiera el valor será "get" quien devuelva la información, en este formato no será necesario especificar el atributo privado.

7.5 Los objetos

Dentro de la programación representan entidades del mundo real que se encuentran asociadas a una determinada clase. Por medio de estos se puede tener acceso a los atributos y métodos definidos dentro del mismo; todo esto dependerá de la visibilidad que implemente a sus métodos o atributos la clase.

Un objeto representa un ejemplar de una clase si se asume que el objeto pertenece a una clase, entonces se puede decir que dicho objeto tendrá acceso a todos los elementos que compone la clase.

7.5.1 Creación de un objeto

```
Clase objeto = new clase();
```

Primero, se especifica el nombre de la clase, seguidamente, el nombre del objeto el cual es el encargado de representar a la clase, luego el operador new que se encarga de crear el objeto y finalmente se hace referencia al método constructor de la clase.

Si cuenta con la clase Empleado y necesita crear un objeto de dicha clase, puede usar los siguientes códigos:

Primera forma:
Empleado objEmp = new Empleado();

Segunda forma:
Empleado objEmp;
objEmp = new Empleado();

Ambas formas cumplen con el objetivo de crear el objeto de la clase Empleado. La diferencia radica solo en la cantidad de líneas usadas y, además, que en la primera forma se crea el objeto directamente, mientras que en la segunda se declara primero el objeto y luego crea el objeto. Este último podría usarse cuando se necesita usar una clase forma global en una aplicación.

7.5.2 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 la clase, véase el caso de cómo asignar valores a sus atributos:

Si tiene la clase Empleado con la siguiente estructura de clase:



- **Enviando valores directos**

  Los valores pueden ser asignados directamente a los atributos públicos de la clase empleado mediante el objeto objE.

```
Empleado objE = new Empleado();
objE.codigo = "E001";
objE.nombres= "Maria";
objE.paterno = "Zamora";
objE.numeroHijos = 3;
objE.sueldoBasico = 3500.00;
```

- **Enviando valores indirectos**

  Los valores pueden ser asignados a variables locales y estas luego podrán ser asignadas a los atributos públicos de la clase Empleado mediante el objeto objE.

```
string código = "E001";
string nombres = "Maria";
string paterno = "Zamora";
int numeroHijos = 3;
double sueldoBasico = 3500.00;

Empleado objE = new Empleado();
objE.codigo = código;
objE.nombres= nombres;
objE.paterno = paterno;
objE.numeroHijos = numeroHijos;
objE.sueldoBasico = sueldoBasico;
```

- **Enviando valores calculados**

  No necesariamente puede enviarse valores directos, también pueden ser valores que provienen de algún cálculo, eso dependerá mucho del proceso de negocio que está implementando.

```
double básico = 2500;
Empleado objE = new Empleado();
objE.codigo = "E001";
objE.nombres= "Maria";
objE.paterno = "Zamora";
objE.numeroHijos = 3;
objE.sueldoBasico = basico+(50/100.0 * basico);
```

7.5.3 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 tiene la clase Empleado con la siguiente estructura de clase y asumiendo que tiene valores asignados mediante un objeto llamado objE:



- **Imprimir los valores directamente**

  Es decir, se solicita el valor de la clase mediante el objeto.

```
MessageBox.Show("El código es: "+objE.codigo);
lblNombres.Text = objE.nombres;
lstR.Items.Add(objE.numeroHijos)
```

- **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 imprimir.

```
String cod = objE.codigo;
String nom = objE.nombres;
Int numHijos = objE.numHijos;

MessageBox.Show("El código es: "+ cod);
lblNombres.Text = nom;
lstR.Items.Add(numHijos)
```

- **Realizar cálculos**

  Habitualmente se usan los valores de los atributos para realizar cálculos o usarlos dentro de las estructuras como if, for, while.

```
double descuento = objE.sueldoBasico * 0.15;

if (objE.numeroHijos > 3)
```

7.6 Método constructor

Un constructor es considerado como un método que permite inicializar los valores que contiene una clase por medio del objeto, por tal razón, cuando se crea un objeto se hace referencia al método constructor. Este método solo se ejecuta cuando algún elemento instancia a la clase, es decir, crea un objeto.

Formato de un constructor vacío:

```
public MetodoConstructor()
{
}
```

Formato de un constructor con especificación de parámetros:

```
public MetodoConstructor(tipo parámetro)
{
  this.atributo  = parámetro;
}
```

Se debe tener en cuenta que todo constructor debe tener visibilidad pública de forma obligatoria, además, una característica especial es que dicho método debe ser nombrado con el mismo nombre de la clase.

Si se tiene la clase Pago e implementa su método constructor, el código podría ser de la siguiente manera:

```
public class Pago
    {
        private int numPago;
        private DateTime fecha;
        private string deudor;

        public Pago(int numPago, DateTime fecha, string deudor){
            this.numPago = numPago;
            this.fecha = fecha;
            this.deudor = deudor;
        }
    }
```

La creación del objeto a partir de la implementación del método constructor varía de la siguiente manera:

```
//Asignando valor a las variables
int numero = 1;
string f = "01/08/2008";
DateTime fecha = Convert.ToDateTime(f);
string deudor = "Fernanda Torres";

//Creando el objeto
Pago objp = new Pago(numero,fecha,deudor);
```

7.7 Casos desarrollados

Caso desarrollado 1 : clase objeto - boleta de venta

La empresa de venta de juegos Oferta-Top, dedicada a la distribución de artículos para diversión, ha decidido implementar un aplicación que permita controlar las ventas que realizan sobre sus productos, para ello se debe generar un boleta de venta que registre el nombre del cliente, la dirección, la fecha de la compra, el número de documento de identidad del cliente. Además, seleccionar un producto e ingresar una cantidad para determinar el importe a pagar por la compra de ciertos productos que ofrece la empresa.

Se tiene en cuenta lo siguiente:

- Implemente la clase Boleta que contenga los atributos número de boleta, nombre, dirección, DNI del cliente, fecha de registro, descripción del producto y la cantidad comprada. Además, la clase debe implementar los métodos "determinaPrecio" el cual se encarga de asignar un precio a un determinado producto y el método "calculaImporte" el cual se encarga de calcular el monto de importe por la compra de un producto.
- Al iniciar la aplicación se debe mostrar el número autogenerado de la boleta con el formato "00000" usando la expresión lambda. Asimismo, mostrar la fecha de registro de la boleta de manera automática.
- Los precios de los productos se encuentran en la siguiente tabla:

Descripción del producto	Precio unitario
PS4 + 1 MANDO DS4	$ 2049.00
PS4 + 2 MANDO DS4	$ 1899.00
PS3 (500GB)	$ 1349.00
MANDO PS4/DS4	$ 219.00
MANDO PS3/DS4	$ 199.00

- Al seleccionar un producto desde el cuadro combinado, automáticamente debe mostrar el precio del producto.

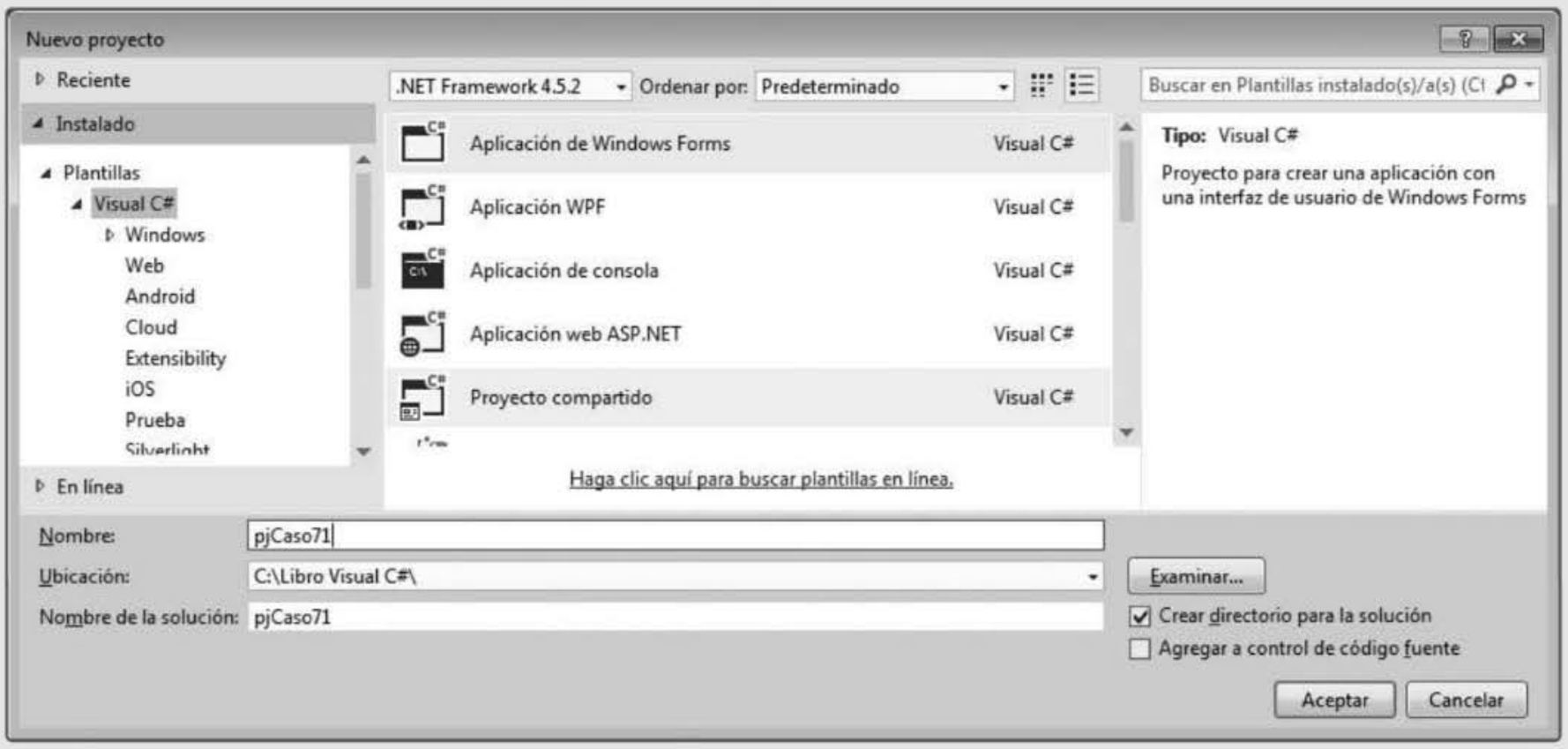
- Mostrar el monto total acumulado de los importes.
- Mostrar un resumen de registro de boletas el cual imprimirá todas las boletas registradas en la aplicación así como la fecha de dicho registro, el total de productos comprados y el monto total.
- El botón **Añadir** tiene la misión de agregar un producto en la lista de detalle según el número de boleta que se encuentre registrando.
- El botón **Registrar boleta** tiene la misión de registrar la información importante que tiene el registro de la boleta como los números de las boletas, las fechas de su registro, el total de productos comprados y el monto total acumulado por cada boleta registrada.
- También se puede dar la posibilidad de eliminar algún producto de la lista, esto se dará al hacer doble clic sobre un producto registrado y la aplicación deberá preguntar como se muestra a continuación:



Solución:

1. Crear un nuevo proyecto en Visual Studio 2015:

- Seleccione **Archivo > Nuevo > Proyecto**.
- Seleccione la plantilla Visual C#.
- Seleccione **Aplicación de Windows Form**.
- Asigne el nombre **pjCaso71** como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:
   - Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones**.
   - Desde la propiedad **Nombre de archivo** asigne el nombre **frmBoleta**.

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



Estadísticas de los controles usados en el frmBoleta:

ClaseW	N.º elementos	Controles
Form	01	Form1
Label	15	Label1, Label2, Label3, Label4, Label5, Label6, Label7, Label8, Label9, Label10, Label11, Label12, Label13, Label14, Label15
TextBox	06	Textbox1, Textbox2, Textbox3, Textbox4, Textbox5, Textbox6
ComboBox	01	ComboBox1
Button	03	Button1, Button2, Button3
ListView	02	ListView1, ListView2
TOTAL	28	

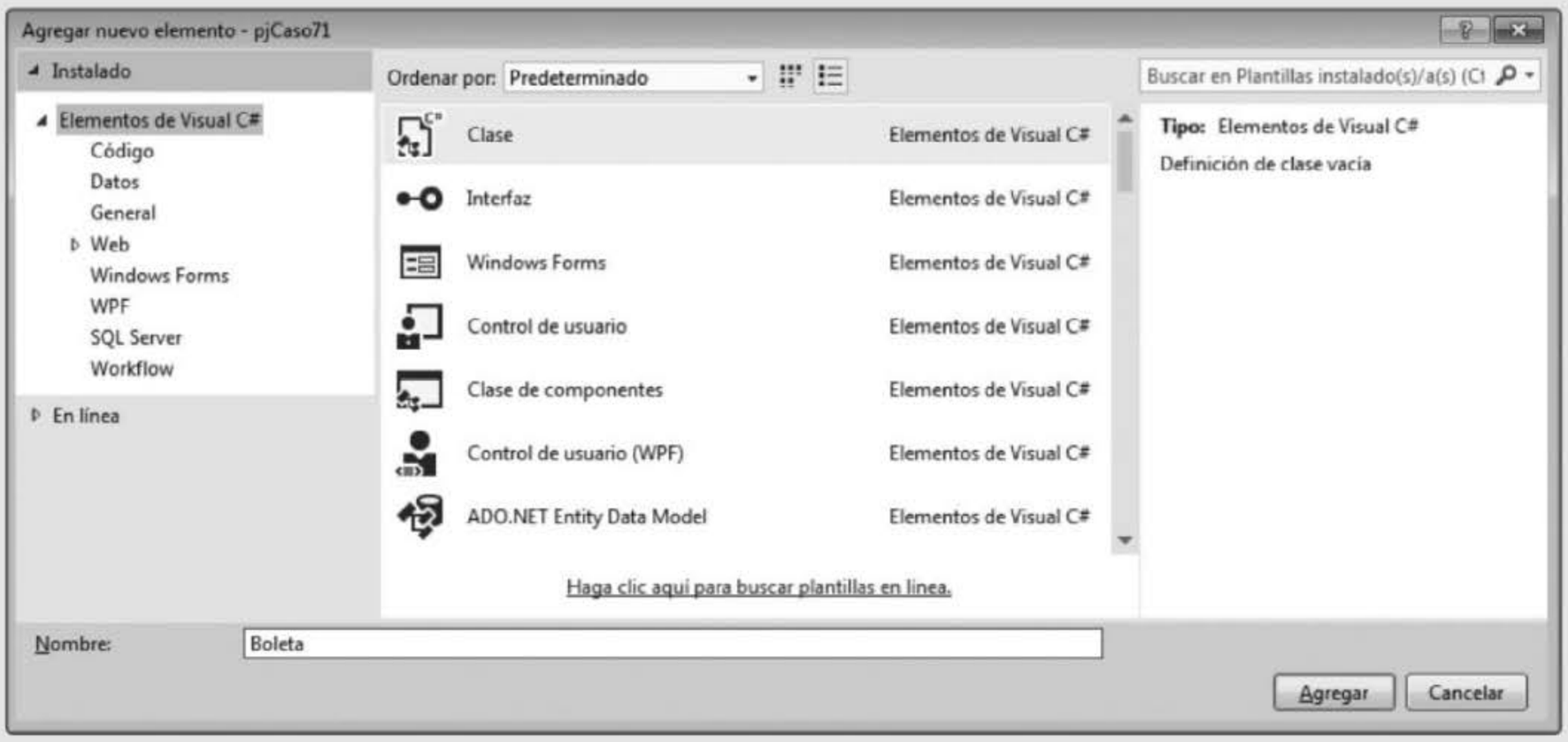
4. Modificar las propiedades de los controles:

Control	Propiedad	Valor
Form1	(Name) Text	frmBoleta Control de venta
Label1	Text	OFERTA-TOP
Label2	Text	R.U.C. 510054511012
Label3	Text	BOLETA DE VENTA
Label4	text	2015 -
Label5	Text	SEÑOR
Label6	Text	DIRECCIÓN
Label7	Text	FECHA
Label8	Text	DNI
Label9	Text	PRODUCTO
Label10	Text	PRECIO
Label11	Text	CANTIDAD
Label12	Text	RESUMEN DE REGISTRO DE BOLETAS
Label13	Text	TOTAL
Label14	(Name) Text	lblNumero lblNumero
Label15	(Name) Text	lblTotal lblTotal
Textbox1	(Name)	txtCliente
Textbox2	(Name)	txtDireccion
Textbox3	(Name)	txtFecha
Textbox4	(Name)	txtDNI
Textbox5	(Name)	txtPrecio
Textbox6	(Name)	txtCantidad
ComboBox1	(Name) Items	cboProducto PS4 + 1 MANDO DS4 PS4 + 2 MANDO DS4 PS3 (500GB) MANDO PS4/DS4 MANDO PS3/DS4
Button1	(Name) Text	btnAñadir AÑADIR
Button2	(Name) Text	btnRegistrar REGISTRAR BOLETA
Button3	(Name) Text	btnSalir SALIR

ListView1	(Name) GridLines	lvDetalle True
ListView2	(Name) GridLines	lvEstadisticas True

5. Agregar la clase Boleta al proyecto:

- Haga clic derecho sobre el **Proyecto > Agregar > Clase.**
- Coloque el nombre **Boleta** y, finalmente, presione el botón **Agregar** como se muestra en la siguiente imagen:



6. Agregar el siguiente código de la clase Boleta:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pjCaso71
{
    public class Boleta
    {
        public int numero { get; set; }
        public string cliente { get; set; }
        public string direccion { get; set; }
        public DateTime fecha { get; set; }
        public string dni { get; set; }
        public string producto { get; set; }
        public int cantidad { get; set; }
```

```
        //Metodo para determinar el precio del producto
        public double determinaPrecio()
        {
            switch (producto)
            {
                case "PS4 + 1 MANDO DS4": return 2049;
                case "PS4 + 2 MANDO DS4": return 1899;
                case "PS3 (500GB)": return 1349;
                case "MANDO PS4/DS4": return 219;
                case "MANDO PS3/DS4": return 199;
            }
            return 0;
        }

        //Metodo para determinar el importe
        public double calculaImporte()
        {
            return cantidad * determinaPrecio();
        }
    }
}
```

Consideraciones al agregar el código de la clase BOLETA:

- **Clase publica Boleta**

  El proyecto se compone de dos clases: la clase Boleta y la clase frmBoleta, la idea es que la clase frmBoleta use los atributos y métodos implementados en la clase Boletapor ello, dicha clase debe ser pública de forma obligatoria.

```
    public class Boleta
    {
    //Implementación de la clase
}
```

- **Declarar los atributos de la clase boleta**

  Los atributos de la clase boleta se declaran de la siguiente manera:

```
public int numero { get; set; }
public string cliente { get; set; }
public string direccion { get; set; }
public DateTime fecha { get; set; }
public string dni { get; set; }
public string producto { get; set; }
public int cantidad { get; set; }
```

Donde public representa la visibilidad que tiene el atributo frente a las demás clases, int representa el tipo de datos del atributo, numero es el nombre del atributo, get representa el método de devolución de valor, mientras que set representa la asignación de valor del atributo. De esta forma, todos los atributos deben tener la misma especificación ya que serán usados en el formulario como parte de la aplicación.

- **Método de clase que determina el precio de un producto**

  Antes de analizar el código de asignación de precio se debe previamente tener el valor del producto, ya que el método evalúa directamente el atributo "producto" porque asume que se encuentra lleno de producto. Luego, comparar dicho valor con la sentencia switch y asignar un precio según el producto seleccionado.

```
public double determinaPrecio()
 {
     switch (producto)
     {
        case "PS4 + 1 MANDO DS4": return 2049;
        case "PS4 + 2 MANDO DS4": return 1899;
        case "PS3 (500GB)": return 1349;
        case "MANDO PS4/DS4": return 219;
        case "MANDO PS3/DS4": return 199;
     }
     return 0;
  }
```

7. Código del formulario:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso71
{
    public partial class frmBoleta : Form
    {
        //Variables GLOBALES
        static int n;
        ListViewItem item;

        //Objeto de la clase Boleta
        Boleta objB = new Boleta();
```

```
        public frmBoleta()
        {
            InitializeComponent();
        }

        private void frmBoleta_Load(object sender, EventArgs e)
        {
            lblNumero.Text = generaNumero();
            txtFecha.Text = DateTime.Now.ToShortDateString();
        }

        private void cboProducto_SelectedIndexChanged(object sender, EventArgs e)
        {
            objB.producto = cboProducto.Text;
            txtPrecio.Text = objB.determinaPrecio().ToString("C");
        }
        private void btnAñadir_Click(object sender, EventArgs e)
        {

            if (valida() == "")
            {
                //Capturar los datos
                capturaDatos();

                //Determinar los calculos de la aplicacion
                double precio = objB.determinaPrecio();
                double importe = objB.calculaImporte();

                //Imprimir el detalle de la venta
                imprimirDetalle(precio, importe);

                //Imprimir el total acumulado
                lblTotal.Text = determinaTotal().ToString("C");
            }
            else
                MessageBox.Show("El error se encuentra en "+valida());
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            ListViewItem fila = new
            ListViewItem("2015-"+(int.Parse(lblNumero.Text).ToString("00000")));
            fila.SubItems.Add(txtFecha.Text);
            fila.SubItems.Add(totalCantidad().ToString("0.00"));
            fila.SubItems.Add(acumuladoImportes().ToString("C"));
            lvEstadisticas.Items.Add(fila);
            limpiarControles();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
```

```
        DialogResult r = MessageBox.Show("Esta seguro de salir?", "Boleta",
                                         MessageBoxButtons.YesNo,
                                         MessageBoxIcon.Information);
        if (r == DialogResult.Yes) this.Close();
    }

    //Metodo que genera un numero aleatorio usando lambda
    Func<string> generaNumero = () =>
    {
        n++;
        return n.ToString("00000");
    };

    //Capturar los datos del formulario
    void capturaDatos()
    {
        objB.numero = int.Parse(lblNumero.Text);
        objB.cliente = txtCliente.Text;
        objB.direccion = txtDireccion.Text;
        objB.fecha = DateTime.Parse(txtFecha.Text);
        objB.dni = txtDni.Text;
        objB.producto = cboProducto.Text;
        objB.cantidad = int.Parse(txtCantidad.Text);
    }

    void imprimirDetalle(double precio, double importe)
    {
        ListViewItem fila = new ListViewItem(objB.cantidad.ToString());
        fila.SubItems.Add(objB.producto);
        fila.SubItems.Add(precio.ToString("0.00"));
        fila.SubItems.Add(importe.ToString("0.00"));
        lvDetalle.Items.Add(fila);
    }

    //Metodo que calcula el monto acumulado de importes
    double determinaTotal()
    {
        double total=0;
        for(int i = 0; i < lvDetalle.Items.Count; i++)
        {
            total +=double.Parse(lvDetalle.Items[i].SubItems[3].Text);
        }
        return total;
    }

    //Total de productos por boleta
    int totalCantidad()
    {
        int total = 0;
        for (int i = 0; i < lvDetalle.Items.Count; i++)
        {
```

```
            total += int.Parse(lvDetalle.Items[i].SubItems[0].Text);
        }
        return total;
    }

    //Monto acumulado de los importes por boleta
    double acumuladoImportes()
    {
        double acumulado = 0;
        for (int i = 0; i < lvDetalle.Items.Count; i++)
        {
            acumulado += double.Parse(lvDetalle.Items[i].SubItems[3].Text);
        }
        return acumulado;
    }

    //Validar el ingreso de datos
    string valida()
    {
        if (txtCliente.Text.Trim().Length == 0)
        {
            txtCliente.Focus();
            return "nombre del cliente";
        }
        else if (txtDireccion.Text.Trim().Length == 0)
        {
            txtDireccion.Focus();
            return "direccion del cliente";
        }
        else if (txtDni.Text.Trim().Length == 0)
        {
            txtDni.Focus();
            return "DNI del cliente";
        }

        else if (cboProducto.SelectedIndex == -1)
        {
            cboProducto.Focus();
            return "descripcion del producto";
        }
        else if (txtCantidad.Text.Trim().Length == 0)
        {
            txtCantidad.Focus();
            return "cantidad comprada";
        }
        return "";
    }
    void limpiarControles()
    {
        lblNumero.Text = generaNumero();
        txtCliente.Clear();
```

```
            txtDireccion.Clear();
            txtDni.Clear();
            cboProducto.Text = "(Seleccione)";
            txtPrecio.Clear();
            txtCantidad.Clear();
            lvDetalle.Items.Clear();
        }

        private void lvDetalle_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            item = lvDetalle.GetItemAt(e.X, e.Y);
            string producto = lvDetalle.Items[item.Index].SubItems[1].Text;
            DialogResult r=MessageBox.Show("Esta seguro de eliminar el producto
                                            > "+producto,
                                            "Boleta",
                                            MessageBoxButtons.YesNo,
                                            MessageBoxIcon.Information);
            if (r == DialogResult.Yes)
            {
                lvDetalle.Items.Remove(item);
                lblTotal.Text=acumuladoImportes().ToString("C");
                MessageBox.Show("Detalle eliminado correctamente...!!!");
            }

        }
    }
}
```

Consideraciones al agregar el código de la aplicación:

- **Implementar la expresión lambda para generar un número de manera automática**

  La expresión lambda inicia con la palabra reservada "Func", seguido se indica el tipo de datos de salida, se le ha declarado como string para que el número generado tenga el siguiente aspecto "0001" como la expresión no necesita parámetros se deja vacío los paréntesis.

```
Func<string> generaNumero= () =>
        {
            contador++;
            return contador.ToString("0000");
        };
```

La variable contador es la encargada de generar los números consecutivos, por ello, debe declararse en la sección global de la aplicación. Se diferencia con otros tipos de variables en que debe declararse de static, ya que de otra manera genera un error por el uso de la expresión lambda. Finalmente, la expresión debe enviar un valor resultante mediante la sentencia return. Los parámetros "0000" se especifican para colocar ceros a la izquierda en la generación del número.

- **Implementar la expresión lambda para asignar un descuento según la categoría**

  La asignación de descuentos se realiza sobre el costo y la evaluación de la categoría, por tal motivo, la expresión lambda tiene dos parámetros de entrada y uno de salida, en su especificación se realiza "string, double, double", string para la categoría, double para el costo y el último double para el descuento resultante de la expresión.

```
Func<string, double, double> asignaDescuento = (categoria, costo) =>
        {
            double descuento = 0;
            switch (categoria)
            {
                case "Gestion": descuento = 10.0/100 * costo; break;
                case "Ingenieria": descuento = 12.0 / 100 * costo; break;
                case "Programacion": descuento = 20.0 / 100 * costo; break;
                case "Base de datos": descuento = 15.0 / 100 * costo; break;
            }
            return descuento;
        };
```

- **Implementar la expresión lambda para calcular el precio de venta**

  Es la expresión más corta que se pueda implementar en lambda ya que solo será cuestión de restar el descuento al costo real de la siguiente forma:

```
Func<double,double,double>calculaPrecioVenta=(costo,descuento)=>costo-descuento;
```

  El primer double representa el costo, el segundo el descuento y ultimo el resultado de la función. Por ser una expresión directa y corta, no necesita llaves ni la instrucción return.

Caso desarrollado 2 : clase objeto - planilla de empleados

La empresa de capacitación World-Tech S. A., dedicada al servicio de cursos online y presencial en diferentes áreas, necesita una aplicación que permita controlar el pago de la planilla de sus empleados. Para ello, se debe registrar el nombre del empleado, fecha de ingreso a la empresa y el cargo; de acuerdo a estos se deben mostrar los montos correspondientes al ingreso del empleado como el sueldo básico, gratificación y comisión, además, el total de descuento a que se somete el empleado como la AFP y el monto por ahorro en la cooperativa de la misma empresa, finalmente, mostrar la aportación al seguro social y los montos acumulados de los ingresos, descuentos y aportaciones.

Coordinador
Jefe
Capacitador
Asistente

Se tiene en cuenta lo siguiente:

- Al iniciar la aplicación se debe mostrar la fecha actual y el mes consultado de la planilla el cual dependerá de la fecha actual, este último debe mostrarse en letras.
- El sueldo básico depende de la siguiente tabla:

Cargo	Sueldo básico
Coordinador	$ 2000.00
Jefe	$ 4000.00
Capacitador	$ 2500.00
Asistente	$ 1200.00

- La gratificación del empleado dependerá del mes consultado y se aumentará un monto de gratificación como se muestra en la siguiente tabla:

Mes	Sueldo básico
Abril	$ 400.00
Julio	$ 450.00
Diciembre	doble de su sueldo básico

- La comisión del empleado solo es aplicable a los empleados con cargo asistente y coordinador, los montos de beneficio por comisión es como sigue:

Cargo	Monto por comisión
Asistente	$ 100.00
Coordinador	$ 250.00

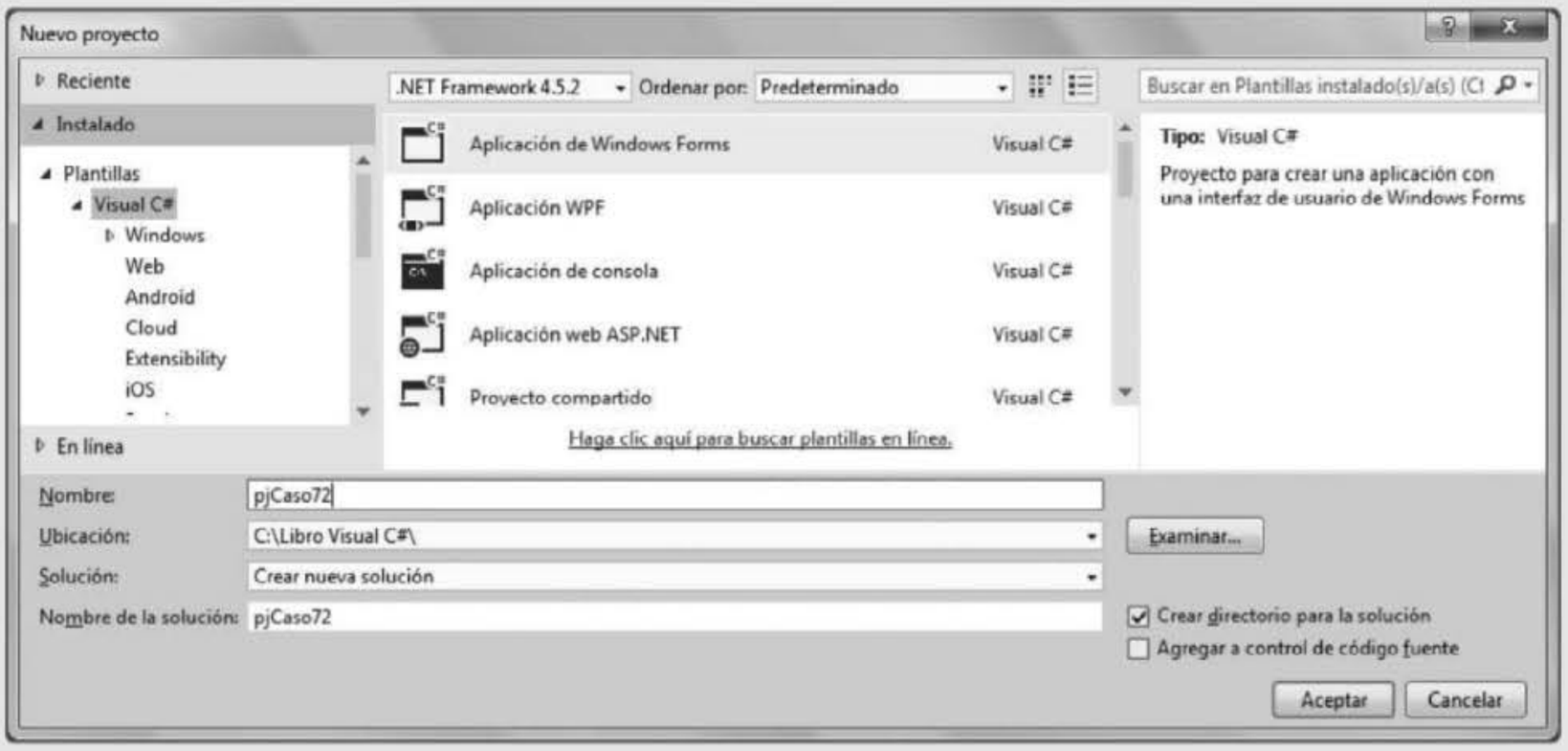
- El monto de la AFP del empleado representa al 12 % del sueldo básico.
- El monto asignado por ahorro en la cooperativa solo será aplicable a los empleados cuyo cargo sea jefe o capacitador, el descuento se realizará al sueldo básico como se muestra en la siguiente tabla:

Cargo	Porcentaje cooperativa
Jefe	5.00 %
Capacitador	2.00 %

- El monto por seguro social del empleado corresponde al 5 % del sueldo básico a todos los empleados de la empresa.
- Implemente la clase Planilla que permita especificar los atributos empleado, cargo, fecha de ingreso y años que labora el empleado. Además de implementar los métodos calcular años de servicio, determinar el mes consultado, el monto básico, el monto de gratificación, el monto de comisión, el monto de descuento AFP, el monto de descuento por ahorro de cooperativa, el monto de aportación al seguro social, calcular los montos totales de ingreso, descuento, aportaciones y, finalmente, el monto neto del empleado.

1. Crear un nuevo proyecto en Visual Studio 2015:

- Seleccione **Archivo > Nuevo > Proyecto**.
- Seleccione la plantilla **Visual C#**.
- Seleccione **Aplicación de Windows Form**.
- Asigne el nombre **pjCaso72** como se muestra en la siguiente imagen:



2. Cambiar el nombre del formulario:

- Seleccione el formulario **Form1.cs** desde el **Explorador de soluciones.**
- Desde la propiedad **Nombre de archivo** asigne el nombre **frmPlanilla.**

3. Diseñar la GUI (Interfaz Gráfica de Usuario) del caso:



4. Código de la clase:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace pjCaso72
{
```

```
public class Planilla
{
    //Atributos
    public string empleado { get; set; }
    public string cargo { get; set; }
    public DateTime fechaIngreso { get; set; }
    public int años { get; set; }

    //Metodos
    public int añosServicio()
    {
        return DateTime.Now.Year - fechaIngreso.Year;
    }

    public string mesConsultado()
    {
        int mes = DateTime.Now.Month; ;
        switch (mes){
            case 1: return “Enero”;
            case 2: return “Febrero”;
            case 3: return “Marzo”;
            case 4: return “Abril”;
            case 5: return “Mayo”;
            case 6: return “Junio”;
            case 7: return “Julio”;
            case 8: return “Agosto”;
            case 9: return “Setiembre”;
            case 10: return “Octubre”;
            case 11: return “Noviembre”;
            case 12: return “Diciembre”;
        }
        return “”;
    }

    //Determinar el sueldo basico
    public double DeterminaBasico()
    {
        switch (cargo)
        {
            case “Coordinador”: return 2000;
            case “Jefe”: return 4000;
            case “Capacitador”: return 2500;
            case “Asistente”: return 1200;
        }
        return 0;
    }

    //Determinar el monto de gratificacion
    public double determinaGratificacion()
    {
        if (mesConsultado() == “Abril”)
            return 400;
```

```
        else if (mesConsultado() == "Julio")
            return 450;
        else if (mesConsultado() == "Diciembre")
            return DeterminaBasico()*2;
        return 0;
    }

    //Determina la comision
    public double determinaComision()
    {
        if (cargo == "Asistente")
            return 100;
        else if (cargo == "Coordinador")
            return 200;
        return 0;
    }

    //Determinar el descuento AFP
    public double determinaDescuentoAFP()
    {
        return 0.12 * DeterminaBasico();
    }

    //Determinar el descuento por Cooperativa
    public double determinaDescuentoCooperativa()
    {
        if (cargo == "Jefe")
            return 5.0 / 100 * DeterminaBasico();
        else if (cargo == "Capacitador")
            return 2.0 / 100 * DeterminaBasico();
        return 0;
    }

    //Determinar aportaciones por seguro social
    public double determinaAportacionSeguro()
    {
        return 0.05 * DeterminaBasico();
    }

    //Calculando los totales
    public double calculaTotalIngresos()
    {
        return DeterminaBasico() + determinaGratificacion()
                                 + determinaComision();
    }
    public double calculaTotalDescuentos()
    {
        return determinaDescuentoAFP()+determinaDescuentoCooperativa();
    }
    public double calculaTotalAportaciones()
    {
        return determinaAportacionSeguro();
```

```
        }

        //Determinar el total neto
        public double determinaNeto()
        {
            return calculaTotalIngresos() - calculaTotalDescuentos()
                                          - calculaTotalAportaciones();
        }
    }
}
```

5. Código del formulario:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace pjCaso72
{
    public partial class frmPlanilla : Form
    {
        public frmPlanilla()
        {
            InitializeComponent();
        }

        private void frmPlanilla_Load(object sender, EventArgs e)
        {
            mostrarFecha();
            mostrarMesActual();
        }

        private void btnProcesar_Click(object sender, EventArgs e)
        {
            //Capturando los datos
            string empleado = txtEmpleado.Text;
            string cargo = cboCargo.Text;
            DateTime fechaIngreso = dtFechaIngreso.Value;
            int años = int.Parse(txtAños.Text);

            //Objeto de la clase Planilla
            Planilla objP = new Planilla();
```

```
            objP.empleado = empleado;
            objP.cargo = cargo;
            objP.fechaIngreso = fechaIngreso;
            objP.años = años;

            //Realizando la impresion de valores
            txtMes.Text = objP.mesConsultado();
            lblBasico.Text = objP.DeterminaBasico().ToString("C");
            lblGratificacion.Text = objP.determinaGratificacion().ToString("C");
            lblComision.Text = objP.determinaComision().ToString("C");
            lblAFP.Text = objP.determinaDescuentoAFP().ToString("C");
            lblCooperativa.Text = objP.determinaDescuentoCooperativa().ToString("C");
            lblSeguro.Text = objP.determinaAportacionSeguro().ToString("C");

            lblTotalIngresos.Text = objP.calculaTotalIngresos().ToString("C");
            lblTotalDescuentos.Text = objP.calculaTotalDescuentos().ToString("C");
            lblTotalAportaciones.Text = objP.calculaTotalAportaciones().ToString("C");

            lblTotal.Text = objP.determinaNeto().ToString("C");
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            txtEmpleado.Clear();
            dtFechaIngreso.Value = DateTime.Now;
            cboCargo.Text = "(Seleccione cargo)";
            txtMes.Clear();
            txtAños.Clear();

            lblBasico.Text = (0).ToString("C");
            lblGratificacion.Text = (0).ToString("C");
            lblComision.Text = (0).ToString("C");
            lblAFP.Text = (0).ToString("C");
            lblCooperativa.Text = (0).ToString("C");
            lblSeguro.Text = (0).ToString("C");

            lblTotalAportaciones.Text = (0).ToString("C");
            lblTotalDescuentos.Text = (0).ToString("C");
            lblTotalIngresos.Text = (0).ToString("C");
            lblTotal.Text = (0).ToString("C");

            txtEmpleado.Focus();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {

            DialogResult r = MessageBox.Show("Esta seguro de salir?", "Planilla",
                                        MessageBoxButtons.YesNo,
                                        MessageBoxIcon.Information);
            if (r == DialogResult.Yes) this.Close();
```

```
        }
        //IMPLEMENTACION DE METODOS
        //1. Mostrar la fecha Actual
        void mostrarFecha()
        {
            lblFechaActual.Text = DateTime.Now.ToShortDateString();
        }
        void mostrarMesActual()
        {
            txtMes.Text = DateTime.Now.Month.ToString();
        }

        private void dtFechaIngreso_ValueChanged(object sender, EventArgs e)
        {
            txtAños.Text = (DateTime.Now.Year
                                    - dtFechaIngreso.Value.Year).ToString();
        }
    }
}
```

Bibliografía

Bibliografía

- Johanes, L. (2010). *Programación C - Metodologia, algoritmos y estructura de datos*. EE. UU.: McGraw Hill.

- Microsoft (2015). *Visual Studio 2015* [Recurso electrónico]. Disponible en https://msdn.microsoft.com/es-es/library/bb386063.aspx [Consulta: 01 de diciembre de 2015]

Impreso en los talleres gráficos de

EDITORIAL MACRO®

Surquillo