PROGRAMACIÓN ORIENTADA A OBJETOS CON

VISUAL C# 2015

Y ADO.NET 4.6

EDITORIAL
MACRO®

EDITORIAL
MACRO®

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

Programación orientada a objetos con Visual C# (2015) y ADO.NET 4.6
Autor: Manuel Torres Remon

© Derechos de autor registrados:
Empresa Editora Macro EIRL

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

Coordinación de edición:
Magaly Ramon Quiroz

Diseño de portada:
Rudy Herrera Torres

Corrección de estilo:
Sonia Obregon Dionisio

Diagramación:
Julissa Ventocilla Fernández

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

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

Primera edición e-book: julio 2016

Disponible en: macrobibliotecasenlinea.com

ISBN N.° 978-612-304-372-8
ISBN e-book N.° 978-612-304-421-3

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

Manuel Torres Remon

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

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

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

EDITORIAL MACRO®

EDITORIAL
MACRO®

Dedicatoria

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

EDITORIAL MACRO®

EDITORIAL
MACRO®

Agradecimientos

Todo libro escrito es un sacrificio que tiene que ser valorado y esto solo se dará si usted aprecia este material, como lo aprecio yo. Escuché, alguna vez, la frase «La lectura de un libro enriquece de conocimientos y empobrece la ignorancia», pienso que es cierto, aunque hoy un libro impreso es difícil de leer, pues son pocas las personas que cargan un libro en la mano y muchos de ellos han sido reemplazados por los dispositivos móviles. Por eso, mi primer agradecimiento es para usted, amigo lector, por adquirir este material, que 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 *Programación orientada a objetos con Visual C# y ADO.NET 4.6*.

Finalmente, agradezco de gran manera por el apoyo en los casos desarrollados a Lucy Pacheco, estudiante del curso taller de programación del Instituto Manuel Arévalo Cáceres.

EDITORIAL MACRO®

EDITORIAL
MACRO®

Índice

Introducción 13

CAPÍTULO 1: Programación orientada a objetos 15

1.1 Introducción a la programación orientada a objetos 17
1.2 Características de la programación orientada a objetos 18
1.3 La clase 21
1.4 Nivel de visibilidad 25
1.5 Atributos de clase 26
1.6 Métodos definidos por el usuario 27
Caso desarrollado 1: Visibilidad pública - Control de promedio de notas 28
1.7 Métodos get y set 32
Caso desarrollado 2: Métodos get y set - Venta de productos 35
1.8 Métodos de clase 41
Caso desarrollado 3: Método de clase - Control de pago de empleados 42
1.9 Método constructor 46
1.10 Los objetos 49
1.10.1 Enviando información a los atributos de la clase por medio del objeto 50
1.10.2 Usando los valores de clase 51
Caso desarrollado 4: Método constructor - Control de facturas de vendedores 54
1.11 Herencia de clases 58
Caso desarrollado 5: Herencia - Venta de productos al contado y crédito 59
1.12 Métodos polifórmicos 71
1.12.1 Modificadores del polimorfismo 71
Caso desarrollado 6: Polimorfismo - Control de evaluaciones 72

CAPÍTULO 2: Colecciones 79

2.1 Las colecciones en .Net Framework 81
2.2 Estructura foreach 81
2.3 Clases de colecciones 84
2.3.1 Espacio de nombre System.Collection 84
2.3.2 Espacio de nombre System.Collections.Specialized 84
2.4 Clase ArrayList 85
2.5 Clase List<> 88
2.6 Clase Stack 91

EDITORIAL MACRO®

2.7 Clase Queue 93
Caso desarrollado 1: ArrayList - Control de registro de personal 96
Caso desarrollado 2: List - Control de registro de productos 102
Caso desarrollado 3: Stack - Control de registro de alumnos 109
Caso desarrollado 4: Queue - Control de registro de libros 114

CAPÍTULO 3: Serialización 123

3.1 Serialización 125
3.2 Ventajas de la serialización 126
3.3 Clase SaveFileDialog 126
3.4 Clase OpenFileDialog 128
3.5 Clase FileStream 129
Caso desarrollado 1: Serialización - Manejo de cadena 130
Caso desarrollado 2: Serialización de datos - Registro de cliente 133
Caso desarrollado 3: Serialización de datos e imagen - Registro de cliente con imagen ... 136
3.6 Clase XMLSerializer 141
Caso desarrollado 4: Serialización XML - Registro de producto 142
Caso desarrollado 5: Serialización XML - Registro de productos usando arreglos 145
Caso desarrollado 6: Serialización XML - Obtener registros de un archivo XML 150

CAPÍTULO 4: ADO.NET 4.6 155

4.1 Script de la base de datos de origen 157
4.2 Diagrama de base de datos Contrato 169
4.3 Introducción al ADO.NET 4.6 170
4.4 Novedades de ADO.NET 4.6 171
4.5 Arquitectura de ADO.NET 172
4.6 Componentes de ADO.NET 173
4.7 Proveedores de datos 173
4.8 Objetos principales de los proveedores de datos 174
4.9 Espacio de nombre System.Data.SqlClient 175
4.10 Espacio de nombre System.Data 176
4.11 Clase SQLConnection 176
4.12 Definición de la cadena de conexión 179
4.13 Clase SQLDataAdapter 180
4.14 ConfigurationManager 184
4.15 Clase DataSet 186
4.16 Clase DataTable 188
4.17 Clase DataView 190
4.18 Instrucción Using 192
4.19 Clase SqlCommand 193

EDITORIAL MACRO®

4.20 Casos desarrollados 196
Caso desarrollado 1: Uso de asistente - Listado de clientes 196
Caso desarrollado 2: Cadena de conexión directa - Listado de clientes 204
Caso desarrollado 3: Usando clase - Listado de clientes 207
Caso desarrollado 4: Manejo de dataset - Listado de contratistas 211
Caso desarrollado 5: Manejo de datatable - Listado de fichas de devolución 215
Caso desarrollado 6: Manejo de dataview - Listado de equipos 219
Caso desarrollado 7: Listado de fichas de devolución por contratista 222
Caso desarrollado 8: Listado de equipos por estado y tipo 227
Caso desarrollado 9: Filtrado de contratos por cliente 232
Caso desarrollado 10: Búsqueda de datos de cliente 236
Caso desarrollado 11: Búsqueda de fichas de reclamo por año 241
Caso desarrollado 12: Mantenimiento de registros de equipo 245
Caso desarrollado 13: Mantenimiento de registro de contratista 256

CAPÍTULO 5: TableAdapter - LinQ to SQL 273

5.1 TableAdapter 275
Caso desarrollado 1: Usando tableadapter y consulta simple - Listado de contratistas 276
Caso desarrollado 2: Usando tableadapter y procedimiento almacenado - Listado de equipos 285
Caso desarrollado 3: Usando tableadapter y procedimiento almacenado - Listado de equipos por estado y tipo 293
5.2 LinQ 298
5.3 Implementación de una consulta con LINQ 300
5.4 Operaciones básicas de una consulta LinQ to SQL 300
5.5 Clase DataContext 302
Caso desarrollado 4: LinQ to SQL - Listado de contratista 303
Caso desarrollado 5: LinQ to SQL con procedimientos almacenados - Listado de equipos 306
Caso desarrollado 6: LinQ to SQL - Listado de fichas de reclamo por cliente 308
Caso desarrollado 7: LinQ to SQL - Listado de fichas de reclamo por año 311
Caso desarrollado 8: LinQ to SQL - Registro del nuevo equipo 314

CAPÍTULO 6: Transacciones - Ado Entity 319

6.1 Transacciones 321
6.2 Clase SqlTransaction 321
6.3 Método beginTransaction 322
Caso desarrollado 1: Mantenimiento de fichas de devolución 323
6.4 Ado Entity 333
Caso desarrollado 2: Ado Entity - Listado de equipo 334
Caso desarrollado 3: Ado Entity - Listado de equipo por estado y tipo 338
Caso desarrollado 4: Ado Entity - Mantenimiento de equipos 341

EDITORIAL MACRO®

CAPÍTULO 7: Reportes 347

7.1 Reportes con Crystal 349
7.2 Instalación de Crystal Report para Visual Studio 2015 349
7.3 Diseño de un informe con Crystal Report 351
7.4 Secciones del reporte 355
7.5 Modificar el diseño del reporte 356
Caso desarrollado 1: Crystal Básico - Reporte de contratos 357
Caso desarrollado 2: Crystal Report - Reporte de contratos por cliente 361
Caso desarrollado 3: Crystal Report - Reporte de contratos por rango de años 366

CAPÍTULO 8: Servicios WCF 373

8.1 WCF 375
8.2 Características del WCF 375
8.3 Implementación de un servicio WCF 378
Caso desarrollado 1: WCF - Listado de contratistas 384
Caso desarrollado 2: WCF - Listado de fichas de devolución por contratista 388
Caso desarrollado 3: WCF - Mantenimiento de equipos 394

CAPÍTULO 9: Programación en N-Capas 407

9.1 Arquitectura en capas 409
9.2 Aplicaciones distribuidas 410
9.3 Creación de una solución en N-Capas 412
Caso desarrollado 1: Mantenimiento de contratistas en N-Capas 418
Caso desarrollado 2: Mantenimiento de clientes con imágenes en N-Capas 441
Caso desarrollado 3: Reporte de equipos con Crystal Report en N-Capas 476
Caso desarrollado 4: Reporte de equipos por estado y tipo con Crystal Report en N-Capas 482
Caso desarrollado 5: Reporte total de contratos por años con gráfico en N-Capas 490

Bibliografía 495

EDITORIAL MACRO®

Introducción

El libro *Programación orientada a objetos con Visual C# y ADO.NET 4.6* permitirá crear aplicaciones de plataforma cliente-servidor usando como plataforma de trabajo al Visual Studio 2015 y SQL Server como servidor de datos.

Para cumplir con dicho objetivo, se ha dividido el libro en nueve capítulos. En el primer capítulo se hace un estudio de las aplicaciones orientadas a objetos, se crean clases y se usan en las aplicaciones, asimismo se aplica la herencia y el polimorfismo.

En el capítulo dos se emplean las clases que permiten registrar información masiva como las colecciones, las cuales pueden ser administradas gracias a los métodos que ofrecen las clases de tipo Colección.

En el capítulo tres se emplean las clases que permiten la serialización de diferentes tipos de archivos. A partir de ellas se pueden crear aplicaciones que conviertan un texto en un archivo binario, manejar imágenes y guardarlas en archivos de texto, además poder crear y leer archivos de tipo XML.

En el capítulo cuatro se implementan aplicaciones cliente-servidor con ADO.NET. En esta parte se implementan métodos para conectarse a la base de datos, así como se preparan métodos que permiten obtener información y realizar un mantenimiento del mismo.

En el capítulo cinco se implementan aplicaciones de consultas a través de la tecnología TableAdapter. Asimismo se realizan consultas usando LinQ to SQL y finalmente se registra información en la base de datos a partir de un entorno Linq to SQL.

En el capítulo seis se implementan aplicaciones de consulta y mantenimiento de registros usando la tecnología ADO Entity. En el capítulo siete se desarrollan aplicaciones que permiten reportar información mediante Crystal Report.

En el capítulo ocho se aprende a crear servicios que permiten obtener y realizar un mantenimiento de registros, que luego serán empleados por una aplicación.

EDITORIAL MACRO®

Finalmente, en el capítulo nueve se implementan aplicaciones de mantenimiento de registros usando la tecnología N-Capas.

Este libro está dirigido a todas las personas que requieren implementar aplicaciones cliente-servidor usando Visual C# 2015 y SQL Server, como estudiantes de la carrera profesional de Computación e Informática, Ingeniería de Sistemas y profesionales de otras especialidades, quienes podrán desarrollar aplicaciones de forma guiada mediante los casos desarrollados escritos detalladamente en este material. Es necesario tener conocimientos previos en el desarrollo de aplicaciones con Visual C#.

EDITORIAL
MACRO®

1

Capítulo

Programación orientada a objetos

Capacidad terminal:

Implementar aplicaciones de plataforma que permitan el uso de clase, objetos y todos los elementos que componen a una clase.

Contenido

EDITORIAL MACRO®

- Introducción a la programación orientada a objetos
- Características de la programación orientada a objetos
- La clase
- Nivel de visibilidad
- Atributos de clase
- Métodos definidos por el usuario
  Caso desarrollado 1: Visibilidad pública - Control de promedio de notas
- Métodos get y set
  Caso desarrollado 2: Métodos get y set - Venta de productos
- Métodos de clase
  Caso desarrollado 3: Métodos de clase - Control de pago de empleados
- Método constructor
- Los objetos
  Caso desarrollado 4: Método constructor - Control de facturas de vendedores
- Herencia de clases
  Caso desarrollado 5: Herencia - Venta de productos al contado y crédito
- Métodos polifórmicos
  Caso desarrollado 6: Polimorfismo - Control de evaluaciones

EDITORIAL
MACRO®

1.1 Introducción a la programación orientada a objetos

La programación orientada a objetos es conocida como un modelo de programación. Esto lleva a pensar que no representa a un lenguaje específico, sino a una forma de implementar los procesos dentro de una organización.

La característica de la programación orientada a objetos es tener la misión de llevar elementos del mundo real a un nivel de código que represente lo mismo. Así, por ejemplo, los empleados de una empresa por características comunes tienen un código que los identifica: nombres y apellidos, categoría de empleado, entre otras propias del empleado.

Analizando un poco más allá de los empleados, se puede encontrar que ellos se distinguen por las distintas labores que realizan o los horarios de trabajo. Es así que se comenzará a pasar los elementos del mundo real en un código orientado a objetos. Sin embargo, resulta importante especificar que existen diferencias entre las palabras «programación orientada a objetos» y un «lenguaje de programación orientado a objetos».

Mientras que la programación orientada a objetos es reconocida como un modelo de programación que convierte entidades del mundo real en objetos, el lenguaje orientado a objetos es considerado como un lenguaje de programación que permite desarrollar aplicaciones orientadas a objetos. En virtud de este contraste, en estos primeros capítulos buscará usar correctamente la metodología de la programación orientada a objetos, tomando como entorno de trabajo a Visual Studio 2015.

Este tipo de programación surge como una alternativa moderna para el desarrollo de aplicaciones, muy distinta a los usos anteriores de una programación estructurada o de una programación modular que permitía descomponer un código completo.

Por otro lado, se trata de una programación que define una serie de conceptos y técnicas que permiten representar eventos o entidades del mundo real basados en objetos —ahora vistos en clases, objetos, métodos, propiedades, herencia, encapsulación y polimorfismo— generando así una aplicación duradera en el tiempo y sujeta a cambios que no perjudican el accionar de la misma.

EDITORIAL MACRO®

1.2 Características de la programación orientada a objetos

Las principales características que presenta la programación orientada a objetos son:

- **Abstracción**

  Se puede definir como una operación que separa las características comunes de un objeto para considerarlas separadamente. Asimismo, se podría decir que la abstracción denota las características principales que distinguen a un objeto con respecto a los demás.

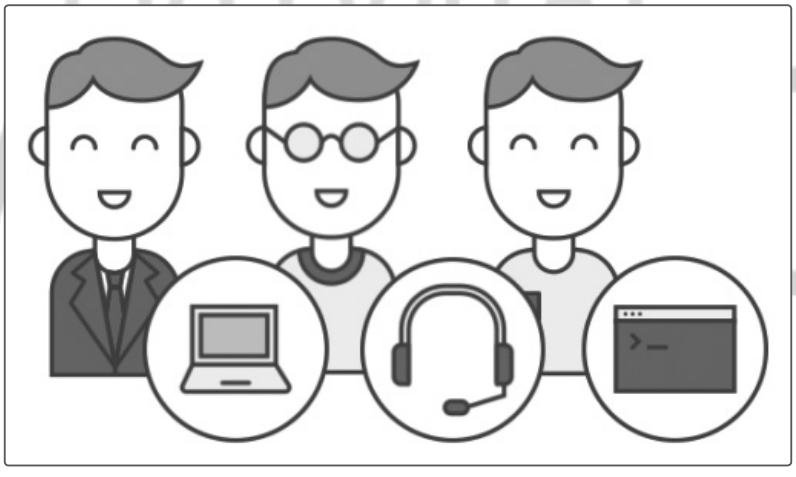
  La idea principal es abstraer los métodos y los datos comunes entre un conjunto de objetos y almacenarlos en una clase. Es de ese modo que los lenguajes de programación orientados a objetos gestionan las abstracciones.

- **Encapsulamiento**

  Un objeto está separado del medio que lo rodea por algún tipo de envoltura. Esta separación es la que determina la unidad del objeto, es decir, lo convierte en algo independiente.

  La envoltura oculta los detalles relacionados con la construcción interior del objeto. Esto significa que solo se conoce del objeto aquello que es revelado por sus métodos públicos.

  Asimismo, se puede decir que encapsular es la propiedad que tienen los objetos de ocultar detalles internos, de tal manera que permiten asegurar que el contenido de la información de un objeto esté oculta al mundo exterior.



Fuente: <http://4.bp.blogspot.com/-lxrSqh5eg0Y/VOruIY1JUjI/AAAAAAAAAC4/l_ArsIatflA/s1600/ArticuloAtlassian1.png>

Si se analiza el proceso de servicio técnico que suelen tener las empresas, se puede ver que si una persona necesita un mantenimiento de su computadora portátil, esta realiza una llamada al personal de servicio técnico. Este genera a su vez una ficha de mantenimiento que será enviada al área técnica para resolver el problema que presenta el cliente.

Entonces, los métodos encapsulados que se pueden encontrar son los siguientes:

Cliente:

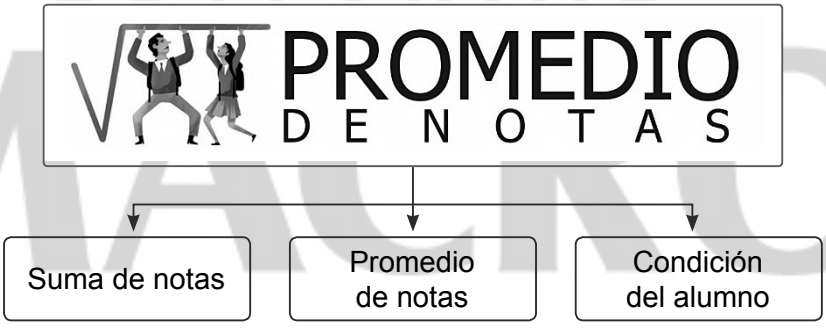
- Enciende la computadora.
- Usa la computadora.
- Reinicia la computadora.
- Apaga la computadora.

Técnico:

- Analiza la falla.
- Diagnostica.
- Repara.
- Realiza pruebas.

- **Modularidad**

  Se denomina modularidad a la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes.

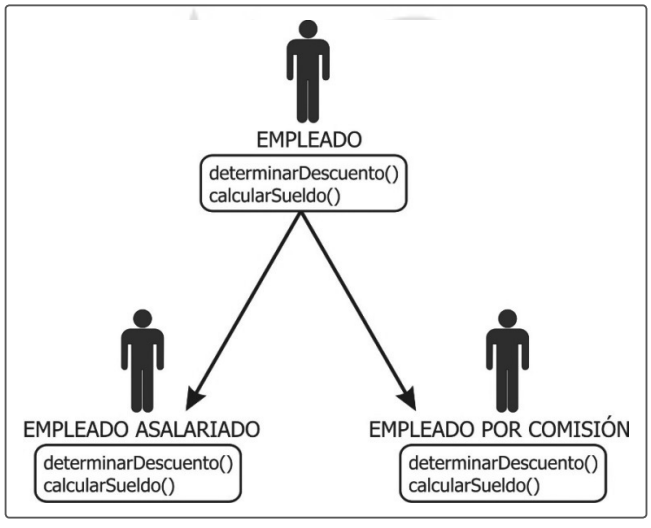


En la imagen anterior, se puede observar que para un proceso de promedio de notas se necesita otros procesos como la suma de todas las notas, el promedio mismo y la condición del alumno, luego de su promedio. Esto determinará si el alumno se encuentra aprobado o desaprobado.

EDITORIAL MACRO®

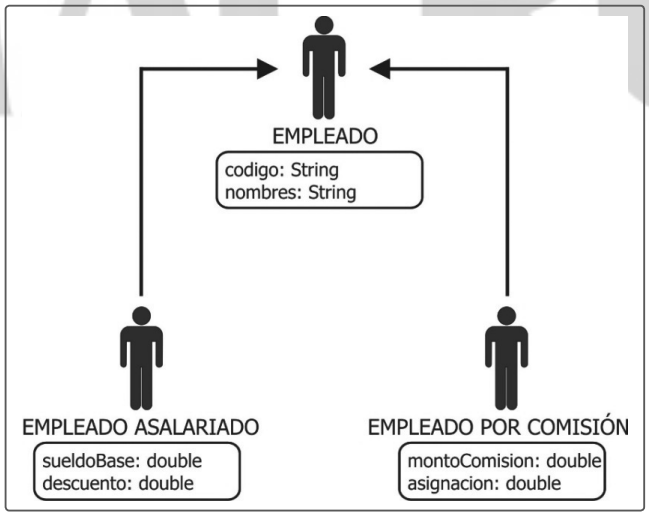
- **Polimorfismo**

  Es uno de los pilares de la programación orientada a objetos el cual permite implementar métodos con el mismo nombre pero de diferente accionar, podríamos nombrar la forma de calcular el sueldo de un empleado el cual se diferencia por el tipo de empleado que puede presentar una empresa, en la cual un tipo de empleado tiene un salario fijo y 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, de tal forma que una clase llamada base puede reunir características comunes entre otras clases y las implementa dentro de esta para luego heredarlas a las clases derivadas o hijas.

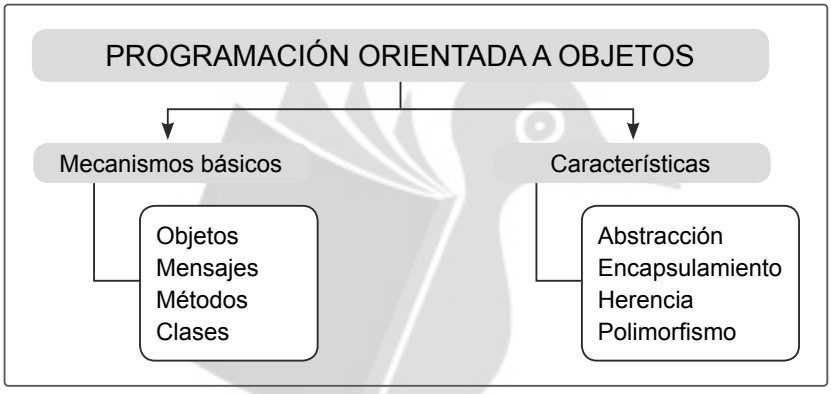


EDITORIAL MACRO®

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.

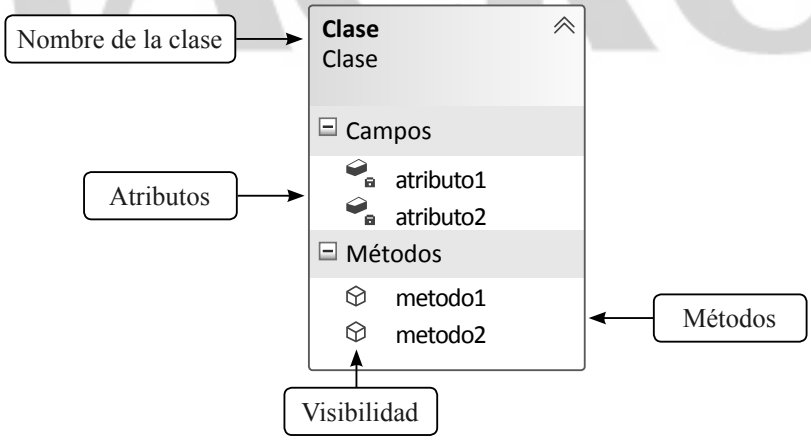


Finalmente, se podría decir que la programación orientada a objetos usa mecanismos básicos para su implementación, los cuales interactúan entre sí, como las clases, los objetos, los mensajes y los métodos.

Asimismo, presenta características muy importantes como la abstracción, el encapsulamiento, la herencia y el polimorfismo.

1.3 La clase

Una clase es considerada como una plantilla que especifica atributos y comportamiento de un determinado tipo de objeto, ya sea físico o abstracto. Una clase en Visual C# se visualiza de la siguiente manera:

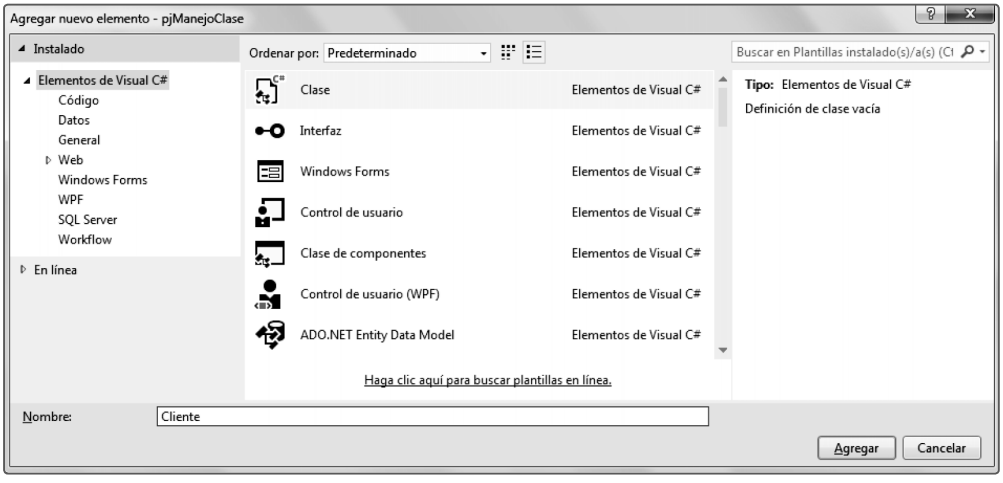


EDITORIAL MACRO®

Hay que tener en cuenta que en Visual Studio 2015 existen muchas formas de crear una clase, por ejemplo, si se necesita implementar la clase Cliente en un proceso de negocio de venta, se puede usar las siguientes formas:

Primera forma: Desde el menú de opciones

- Dentro de una aplicación Windows Form de Visual C#, seleccione el menú **Proyecto > Agregar clase**.

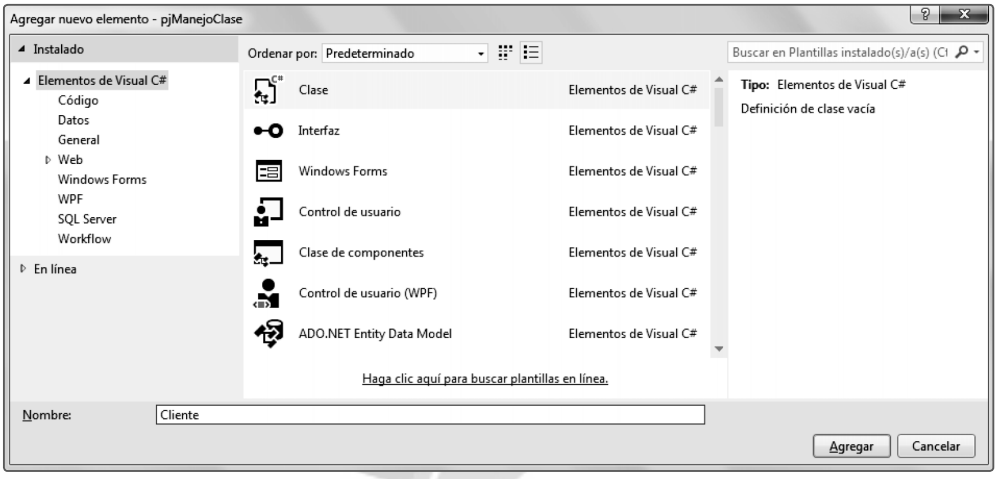


- Seleccione **Elementos de Visual C# > Clase**.
- Asigne un nombre a la clase, por ejemplo, Cliente.
- El código que se genera inicialmente es el siguiente:

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

namespace pjManejoClase
{
    class Cliente
    {

    }
}
```

Segunda forma: Desde la ventana Explorador de soluciones

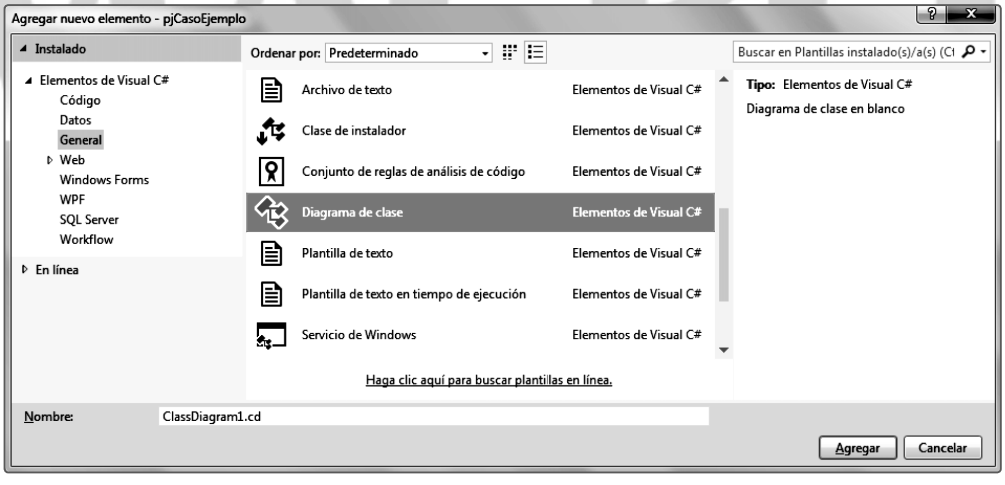
- Dentro de una aplicación Windows Form de Visual C#, presione clic derecho sobre el proyecto desde la ventana Explorador de soluciones.
- Seleccione **Agregar** > **Clase**.



- De la misma manera, se debe asignar el nombre de la clase, por ejemplo Cliente y presione el botón **Agregar**.

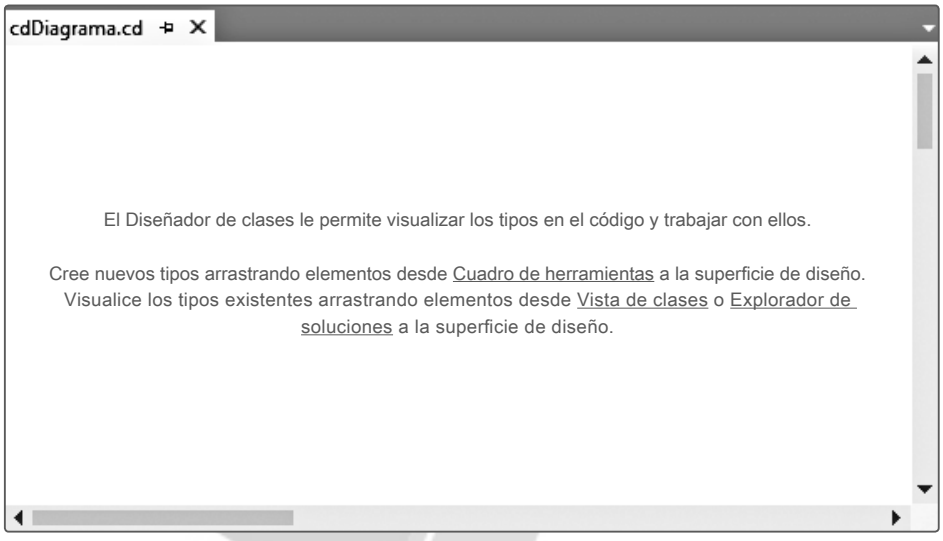
Tercera forma: Mediante un diagrama de clase

- 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**.

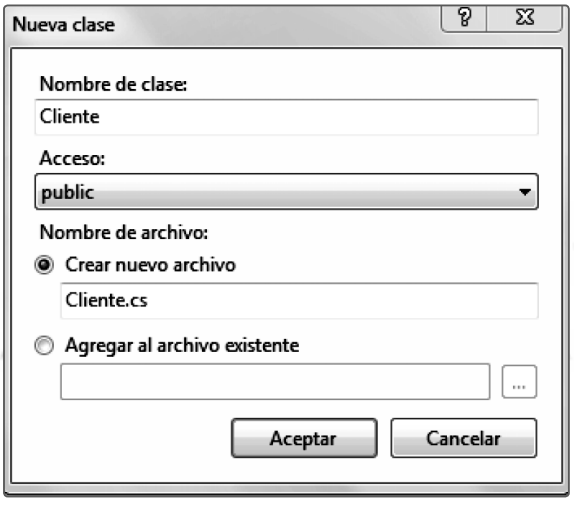


EDITORIAL MACRO®

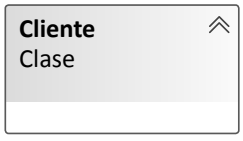
- Asigne un nombre al diagrama, por ejemplo, cdDiagrama y presione el botón **Agregar**.



- Como se puede notar, aparece un entorno a partir del cual se podrá agregar la clase de la siguiente manera: presione clic derecho en el fondo del **Diseñador de clase** y seleccione **Agregar** > **Clase**. Luego, asigne un nombre, como se muestra en la siguiente imagen:



- Una vez agregada la clase Cliente, se puede visualizar el código de la clase presionando doble clic sobre la clase.



Y el código mostrado es exactamente igual al mostrado en la primera forma, lo único que lo diferencia es que la clase se presenta como pública, ya que, al crearla, se selecciona el acceso público, como se muestra en el siguiente código:

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

namespace pjManejoClase
{
    public class Cliente
    {

    }
}
```

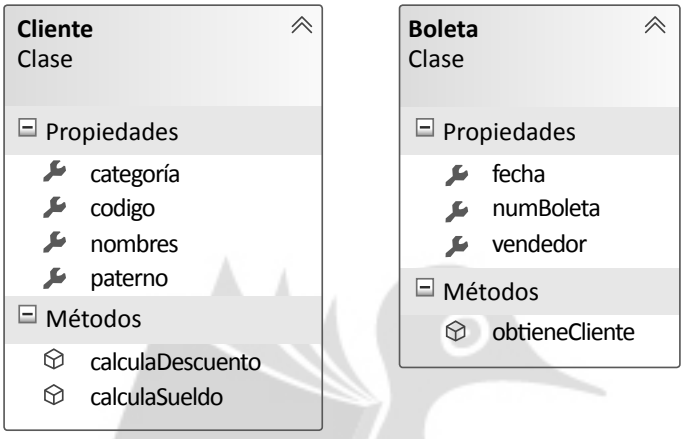
1.4 Nivel de visibilidad

Para controlar el acceso a los miembros de una clase, los cuales pueden ser considerados como los atributos y métodos de la clase, se usan especificadores o modificadores de acceso que se anteponen a las declaraciones de los miembros a controlar. Los especificadores de acceso pueden ser:

Visibilidad	Acceso a los miembros		
	Desde la misma clase	Desde una clase heredada	Desde el exterior de la clase
Public	SÍ	SÍ	SÍ
Private	SÍ	NO	NO
Protected	SÍ	SÍ	NO
Sin especificación	SÍ	NO	NO

EDITORIAL MACRO®

Si se cuenta con las clases Cliente-Boleta, se presenta las siguientes cuestiones:



- Si los atributos categoría, código, nombres y paterno de clase cliente son privados, ¿qué métodos de ambas clases pueden usar dichos atributos?

  Solo podrán ser accesibles por los métodos calculaDescuento y calculaSueldo. El método obtieneCliente no podrá acceder a ningún atributo de la clase Cliente.

- Si los atributos categoría, código, nombres y paterno de clase cliente son públicos, ¿qué métodos de ambas clases pueden usar dichos atributos?

  Todos los métodos podrán visualizar a los atributos de la clase Cliente.

1.5 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 puede acceder a los miembros. Observe cómo agregar atributos en una clase implementada en Visual Studio 2015:

```
Visibilidad Tipo_Datos NombreAtributo;
```

Donde:

- **Visibilidad**: Representa la forma de acceso sobre los atributos.
- **Tipo de datos**: Representa el tipo de datos del atributo.
- **Atributo**: Representa el nombre del atributo.

EDITORIAL MACRO®

Si se necesita declarar los atributos código, nombres, paterno y categoría de la clase Cliente de forma privada, entonces el código sería:

```
private string codigo;
private string nombres;
private string paterno;
private string categoria;
```

También se puede implementar el código de la siguiente manera:

```
string codigo;
string nombres;
string paterno;
string categoria;
```

Si se necesita declarar los atributos código, nombres, paterno y categoría de la clase Cliente de forma pública, entonces el código sería:

```
public string codigo;
public string nombres;
public string paterno;
public string categoria;
```

1.6 Métodos definidos por el usuario

Representa la lógica de negocio, la cual permite realizar un proceso determinado que cumplirá en parte con el objetivo de la aplicación. Toda implementación de clase cuenta con un método nativo llamado método constructor, el cual se analizará más adelante.

Los métodos pueden tener asignado cualquier tipo de visibilidad, lo que se debe tener en cuenta es que se asignará como privado a métodos que permitan realizar un proceso específico que solo servirá dentro de la clase, mientras que los asignados como público serán visibles desde cualquier clase.

EDITORIAL MACRO®

Formato:

Existen cuatro formatos para la implementación de un método:

1. Método sin valor de retorno sin parámetros.

```
visibilidad void nombreMetodo(){
 //Implementación del método
}
```

2. Método sin valor de retorno con parámetros.

```
visibilidad void nombreMetodo(Parámetro){
 //Implementación del método
}
```

3. Método con valor de retorno sin parámetros.

```
visibilidad tipoDatos nombreMetodo(){
 //Implementación del método
}
```

4. Método con valor de retorno con parámetros.

```
visibilidad tipoDatos nombreMetodo(Parámetro){
 //Implementación del método
}
```

EDITORIAL MACRO®

Caso desarrollado 1 : Visibilidad pública - Control de promedio de notas

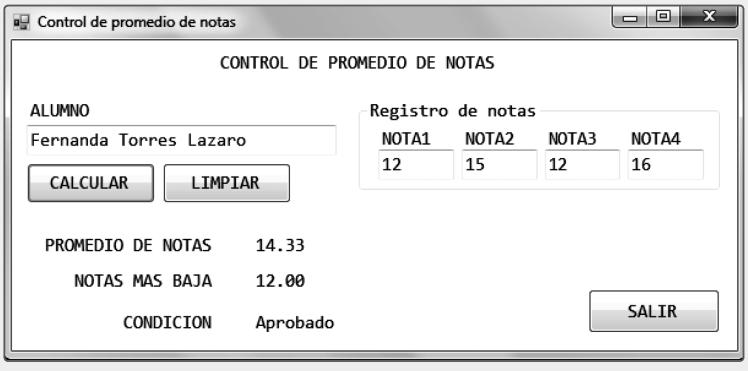
Implementar una aplicación que permita controlar el promedio de notas de un alumno en un determinado curso.

Se tiene en cuenta lo siguiente:

- Implementar la clase Promedio, la cual incluya los atributos del alumno, como su nombre y sus cuatro notas con visibilidad pública.

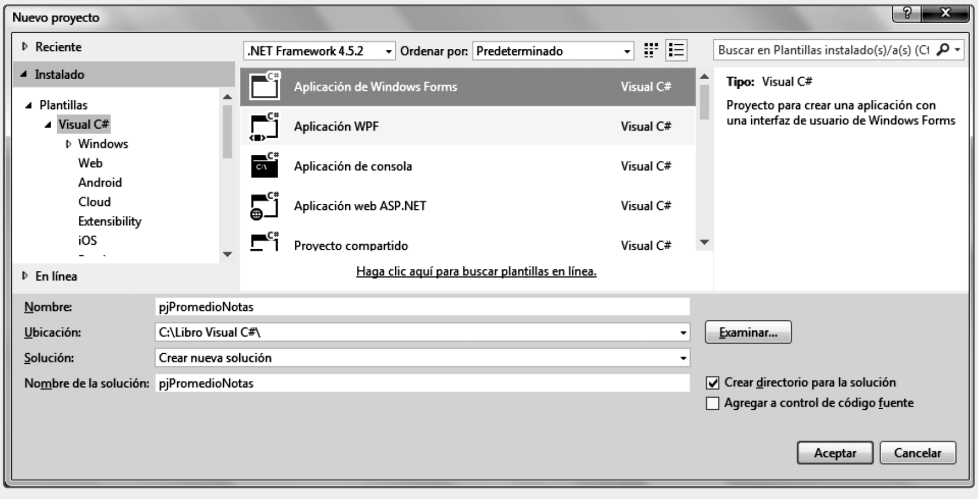
- Implementar un método que permita determinar la nota más baja que tiene el alumno.
- Implementar un método que permita determinar el promedio de notas eliminando la más baja de sus cuatro notas.
- Implementar un método que permita determinar la condición de un alumno, la cual, en base a su promedio, debe mostrar los mensajes de aprobado (mayor a 12.5), recuperación (mayor a 10, pero menor o igual a 12.5) y desaprobado (menor o igual a 10).

GUI propuesta:



Solución:

1. Haga clic en **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjPromedioNotas.



2. Agregue al proyecto la clase Promedio e implemente el siguiente código:

Clase Promedio:

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

namespace pjPromedioNotas
{
    public class Promedio
    {
        //Atributos públicos de la clase
        public string alumno;
        public int nota1;
        public int nota2;
        public int nota3;
        public int nota4;

        //Metodos de la clase
        public int masBaja()
        {
            int menor=int.MaxValue;
            if (nota1 < nota2) menor = nota1; else menor = nota2;
            if (nota3 < menor) menor = nota3;
            if (nota4 < menor) menor = nota4;
            return menor;
        }
        public double calculaPromedio()
        {
            return (nota1+nota2+nota3+nota4-masBaja()) / 3.0;
        }
        public string asignaCondicion()
        {
            double promedio = calculaPromedio();
            if (promedio <= 10)
                return "Desaprobado";
            else if (promedio > 10 && promedio <= 12.5)
                return "Recuperación";
            else
                return "Aprobado";
        }
    }
}
```

Observaciones en el código:

La clase Promedio contiene atributos que permiten obtener información como el nombre del alumno y sus cuatro notas. Adicionalmente se implementan los siguientes métodos:

- **masBaja**

  Método que determina la menor nota que obtiene el alumno a partir de sus cuatro notas. En este método se usa la función MaxValue de la clase int, la cual obtiene el número de más alto perteneciente a los números enteros. Luego, este valor es asignado a la variable «menor», que tiene por misión guardar el valor correspondiente a la menor nota. Luego de una serie de comparaciones, se obtiene el menor valor, el cual es devuelto con la sentencia return.

- **calculaPromedio**

  Método que tiene la misión de calcular el promedio entre las cuatro notas y la menor nota obtenida.

- **asignaCondicion**

  Método que tiene la misión de determinar si con el promedio obtenido el alumno se encuentra aprobado (mayor a 12.5), desaprobado (menor o igual a 10) o debe dar examen de recuperación (entre 11 y 12.5).

3. Implemente el siguiente código en la clase frmPromedio:

```
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 pjPromedioNotas
{
    public partial class frmPromedio : Form
    {
        public frmPromedio()
        {
            InitializeComponent();
        }
```

```
        private void btnCalcular_Click(object sender, EventArgs e)
        {
            //Objeto de la clase Promedio
            Promedio objP = new Promedio();

            //Enviando los valores a la clase
            objP.alumno = txtAlumno.Text;
            objP.nota1 = int.Parse(txtNota1.Text);
            objP.nota2 = int.Parse(txtNota2.Text);
            objP.nota3 = int.Parse(txtNota3.Text);
            objP.nota4 = int.Parse(txtNota4.Text);

            //Imprimiendo los valores
            lblPromedio.Text = objP.calculaPromedio().ToString("0.00");
            lblBaja.Text = objP.masBaja().ToString("0.00");
            lblCondicion.Text = objP.asignaCondicion();
        }
    }
}
```

Observaciones en el código:

En el botón btnCalcular se debe colocar el código fuente que permitirá devolver los resultados que se esperan de la aplicación. Se empieza por crear un objeto de la clase Promedio, que permitirá tener acceso a los atributos públicos. Es justamente por este modo que se enviará la información necesaria a la clase para determinar los cálculos correspondientes. Cuando se imprimen los resultados, es por este mismo objeto que se puede obtener dichos valores. Para este caso, no se necesita enviar parámetros por los métodos, porque previamente dichos valores ya fueron enviados a la clase por medio de la asignación directa a sus atributos públicos.

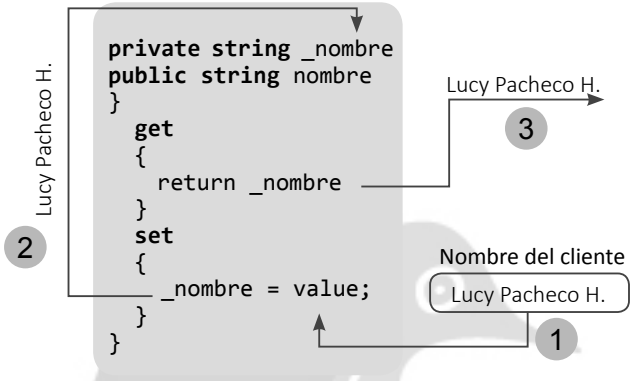
MACRO

1.7 Métodos get y set

Son métodos que permiten controlar el acceso a la información contenida en la clase. Se debe tener en cuenta que los atributos de la clase deben ser privados y la información desde el exterior debe llenar de valores a sus atributos. Entonces, se usará métodos que permitan asignar y devolver valores a dichos atributos.

- El método set permite asignar un valor al atributo privado desde el exterior, de tal forma que inicializa y actualiza el valor de dicho atributo.
- El método get permite devolver el valor almacenado en el atributo privado, de tal forma, que al invocar a la propiedad, este devolverá el valor recepcionado.

Finalmente, observe cómo enviar información a una propiedad de la clase. Para este ejemplo, se usará la clase Venta, que cuenta con el atributo Nombre de un cliente.



Para iniciar la explicación, se va a suponer que se tiene la clase Venta, la cual cuenta con el atributo privado _nombre que pertenece al nombre de un determinado cliente. Entonces, para enviar el nombre del cliente, se debe realizar el siguiente código:

```
Venta objV = new Venta
objV.nombre ="Lucy Pacheco H";
```

Cuando se le envía el valor, la propiedad encarga al set a inicializar el atributo privado con el valor que proviene del exterior y es así que dicho atributo privado pasa a tener un valor asignado. Luego, cuando el valor quiere ser devuelto, se podría usar el siguiente código:

```
LblNombre.Text = objV.nombre;
```

La propiedad determina que get debe enviar dicho valor a quien lo invoca, de tal forma que devuelve el valor al control lblNombre.

Para implementar los métodos get y set, se necesita una clase, y dentro del entorno de código en Visual C# se puede realizar lo siguiente:

- Colocar el término «prop» y presionar dos veces la tecla **Tab**, de tal forma que aparezca el siguiente código:

```
public int MyProperty { get; set; }
```

EDITORIAL MACRO®

- Colocar el término «propfull» y presionar dos veces la tecla **Tab**, de tal forma que aparezca el siguiente código:

```
private int myVar;

public int MyProperty
{
    get { return myVar; }
    set { myVar = value; }
        }
```

En ambos casos se debe modificar el tipo de datos del atributo y el nombre del mismo, es decir, los valores a modificar son MyProperty y myVar, de tal forma que se puede tener la siguiente implementación de propiedades:

- Usando el método prop:

```
public class Factura
    {
        public int numeroFactura { get; set; }
        public DateTime fechaRegistro { get; set; }
        public String  cliente { get; set; }
        public double monto{ get; set; }
    }
```

- Usando el método propfull:

```
public class Factura
    {
        private int _numeroFactura;
        public int numeroFactura
        {
            get { return _numeroFactura; }
            set { _numeroFactura = value; }
        }
        private DateTime _fechaRegistro;
        public DateTime fechaRegistro
        {
            get { return _fechaRegistro; }
            set { _fechaRegistro = value; }
        }
        private string _cliente;
        public string cliente
```

```
        {
            get { return _cliente; }
            set { _cliente = value; }
        }
        private double _monto;
        public double monto
        {
            get { return _monto; }
            set { _monto = value; }
        }
    }
```

Caso desarrollado 2 : Métodos get y set - Venta de productos

Una tienda de venta de aparatos informáticos ha decidido implementar una aplicación que permita controlar la venta de sus productos. Se requiere registrar los datos del cliente, fecha y hora de la venta, así como especificar el producto y la cantidad a adquirir. Se pide obtener el precio del producto seleccionado así como el subtotal, el descuento y el neto a pagar por la compra de ciertos productos.

Se tiene en cuenta lo siguiente:

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

Producto	Precio
Mouse	$20.00
Teclado	$35.00
Impresora	$350.00
Monitor	$550.00
Parlantes	$50.00

- El subtotal es calculado en base al precio del producto y la cantidad seleccionada.
- El descuento aplicado se muestra en la siguiente tabla:

Subtotal	Descuento
Menor o igual a 300	5 %
Entre 301 y 500	10 %
Mayor a 500	12.5 %

GUI Propuesta:

Clase Venta:

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

namespace pjVenta
{
    public class Venta
    {
        //Atributos y metodos GET y SET
        private string _producto;
        public string producto
        {
            get { return _producto; }
            set { _producto = value; }
        }

        private int _cantidad;
        public int cantidad
        {
            get { return _cantidad; }
            set { _cantidad = value; }
        }

        //Metodos
        //Asignando el precio a los productos
        public double asignaPrecio()
        {
```

```
            switch (producto)
            {
                case "Mouse": return 20;
                case "Teclado": return 35;
                case "Impresora": return 350;
                case "Monitor": return 550;
                case "Parlantes": return 50;
            }
            return 0;
        }

        //Calculando el subtotal
        public double calculaSubtotal()
        {
            return asignaPrecio() * cantidad;
        }

        //Calculando el descuento
        public double calculaDescuento()
        {
            double subtotal = calculaSubtotal();
            if (subtotal <= 300)
                return 5.0 / 100 * subtotal;
                else if (subtotal > 300 && subtotal<=500)
                return 10.0 / 100 * subtotal;
                else
                return 12.5/100 * subtotal;
        }

        //Calculando en neto
        public double calculaNeto()
        {
            return calculaSubtotal() - calculaDescuento();
        }
    }
}
```

Observaciones en el código:

Dentro de la clase Venta, se debe implementar los métodos get y set para los atributos producto y cantidad. Recordar que esto se logra con definir la propiedad para cada atributo usando las palabras prop o propfull.

Asimismo, se define el método asignaPrecio, el cual permite asignar un precio a un determinado producto. Aquí se usa la estructura switch comparando el posible producto seleccionado y desde allí se devolverá directamente el precio del producto. Para todos los casos, no será necesario especificar la sentencia break, ya que se está devolviendo un valor directo.

El método calculaSubtotal permite obtener el valor subtotal entre el precio y la cantidad por producto. Se debe tener en cuenta que el precio del producto se obtiene del método asignaPrecio. El método calculaDescuento permite asignar un monto de descuento según el valor obtenido en el subtotal y siguiendo la tabla de descuentos según el caso. Finalmente, se implementa el método calculaNeto, el cual se obtiene de la diferencia entre el subtotal y el descuento.

Clase frmVenta:

```
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 System.Collections;

namespace pjVenta
{
    public partial class frmVenta : Form
    {
        //Inicialización del arreglo de productos
        static string[] productos = { "Teclado", "Impresora",
                                      "Monitor", "Mouse",
                                      "Parlantes" };
        //Creando el objeto de la clase ArrayList
        ArrayList aProductos = new ArrayList(productos);

        //Objeto de la clase venta
        Venta objV = new Venta();

        //Acumulador de totales
        double total;

        public frmVenta()
        {
            InitializeComponent();
        }

        private void frmVenta_Load(object sender, EventArgs e)
        {
            limpiaControles();
            llenaProductos();
            muestraFecha();
            muestraHora();
        }
```

```
//Llenando los productos en el cuadro combinado
void llenaProductos()
{
    foreach(string p in aProductos)
    {
        cboProducto.Items.Add(p);
    }
}

//Método que muestra la fecha actual
void muestraFecha()
{
    lblFecha.Text = DateTime.Now.ToShortDateString();
}

//Método que muestra la hora actual
void muestraHora()
{
    lblHora.Text = DateTime.Now.ToShortTimeString();
}

private void btnRegistrar_Click(object sender, EventArgs e)
{
    //Enviando los valores a la clase
    objV.producto = cboProducto.Text;
    objV.cantidad = int.Parse(txtCantidad.Text);

    //Imprimiendo las respuestas
    ListViewItem fila = new ListViewItem(objV.producto);
    fila.SubItems.Add(objV.cantidad.ToString());
    fila.SubItems.Add(objV.asignaPrecio().ToString("C"));
    fila.SubItems.Add(objV.calculaSubtotal().ToString("C"));
    fila.SubItems.Add(objV.calculaDescuento().ToString("C"));
    fila.SubItems.Add(objV.calculaNeto().ToString("C"));
    lvVenta.Items.Add(fila);

    //Calculando el total de netos
    total += objV.calculaNeto();
    lblTotal.Text = total.ToString("C");
}

//Método que limpia los controles del formulario
void limpiaControles()
{
    txtCliente.Clear();
    cboProducto.Text = "(Seleccione un producto)";
    txtCantidad.Clear();
    lblPrecio.Text = "";
    txtCliente.Focus();
}
```

```
            private void cboProducto_SelectedIndexChanged(object sender,
            EventArgs e)
        {
            objV.producto = cboProducto.Text;
            lblPrecio.Text = objV.asignaPrecio().ToString("C");
        }
    }
}
```

Observaciones en el código:

Para poblar el cuadro combinado de productos, se preparó un arreglo declarado de forma global con los productos que se ofertan en la tienda:

```
static string[] productos = { "Teclado", "Impresora",
                              "Monitor", "Mouse",
                              "Parlantes" };
```

El arreglo productos debe ser declarado de tipo cadena y asignado de productos encerrados entre llaves. Esto crea una matriz de productos, en donde el orden especificado es el mismo que se mostrará en el cuadro combinado. No olvidar que debe ser declarado como «static» para que no se modifique los valores iniciales. Ahí mismo, en la línea global, se debe declarar un objeto de la clase ArrayList, el cual tendrá la misión de almacenar los productos especificados en el arreglo de productos. Es a partir de aquí que se enviará todos los valores al cuadro combinados.

Asimismo, se debe declarar un objeto de la clase Venta llamado objV, que permitirá tener acceso a los atributos y métodos públicos de la clase Venta; la variable total, la cual tendrá la misión de acumular los montos netos y determinar el total a pagar por toda la venta.

Se implementan los siguientes métodos:

- **llenaProductos**: Encargado de llenar los productos en el cuadro combinado, usando la estructura repetitiva foreach y el objeto de tipo ArrayList llamado aProductos.
- **muestraFecha**: Encargado de imprimir la fecha en el control label al iniciar la aplicación.
- **muestraHora**: Encargado de imprimir la hora en el control label al iniciar la aplicación.

- **LimpiaControles**: Encargado de limpiar los controles que se presentan en el formulario, así como devolver el foco a la caja de texto del cliente.

En el botón Registrar se envía los valores obtenidos desde los controles del formulario como el producto y la cantidad. Luego, estos valores serán enviados al control ListView y allí se especificará los métodos que devuelven los montos como el subtotal, el descuento y el neto.

Finalmente, se imprime el monto total a pagar por toda la venta. Este valor se acumulará en la variable global «total», la cual se obtiene desde el cálculo del monto neto a partir del método calculaNeto.

1.8 Métodos de clase

Una clase también puede ayudar a implementar métodos que permitan operar directamente dentro de una aplicación. Estos métodos podrán ser accedidos desde cualquier aplicación. La plantilla de clase que contenga métodos de clase es:

```
public class Nombre_Clase
{
   //Método constructor privado
   private nombre_constructor()
   {
   }

   //Método estático
   public static double nombreMetodo(atributo)
   {
      return valor;
   }
}
```

Entonces, para implementar la clase se necesita:

- Implementar el método constructor en privado para que al invocar a la clase no genere inicialización alguna.
- Los métodos que van a ser implementados deben especificar obligatoriamente la cláusula estática.

Caso desarrollado 3 : Método de clase - Control de pago de empleados

Una empresa de auditoría necesita contratar personal por hora para los tres últimos meses del año. Además, necesita una aplicación que permita controlar los pagos de dichos empleados, para lo cual ingresará el nombre del empleado, la categoría según su contrato y las horas trabajadas. Lo que se quiere obtener es el monto bruto, el descuento y el neto a pagar.

Se tiene en cuenta lo siguiente:

- Las categorías de los empleados deben ser registradas mediante un objeto de tipo ArrayList. Además, el monto por hora asignado se debe al tipo de categoría del empleado como se muestra en la siguiente tabla:

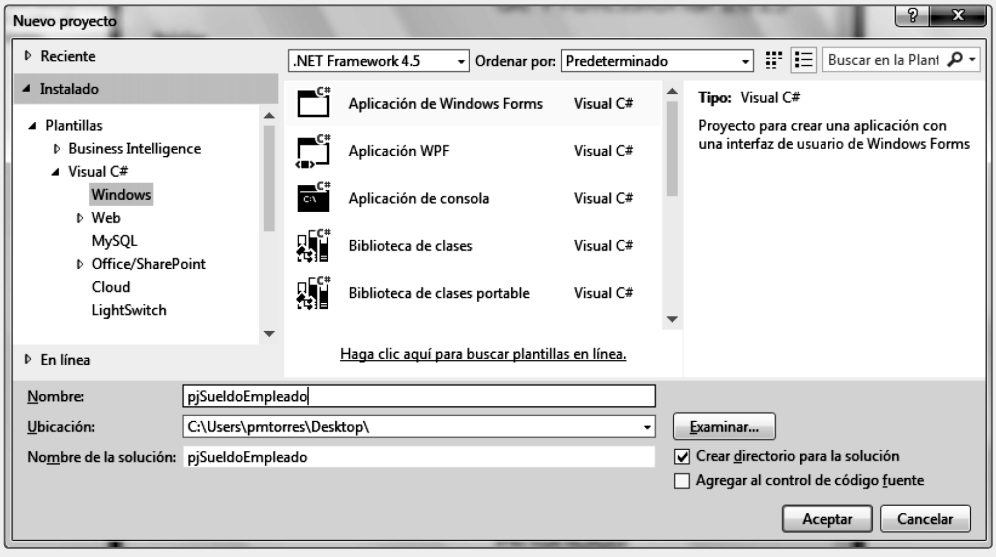
Categoría	Monto
CAS	$15.00
CAP	$25.00

- El cálculo de descuento resulta de aplicar el 12 % al monto bruto.
- El neto a pagar resulta de la diferencia entre el monto bruto y el descuento aplicado.

GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjSueldoEmpleado.



2. Agregue al proyecto la clase Empleado e implemente el siguiente código:

Clase Empleado:

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

namespace pjSueldoEmpleado
{
    public class Empleado
    {
```

```
        //Método constructor privado
        private Empleado()
        {
        }
        //Método que determina la hora según la categoría
        public static double asignaCostoHora(string categoria)
        {
            switch (categoria)
            {
                case "CAS": return 15;
                case "CAP": return 25;
            }
            return 0;
        }

        //Método que determina el monto bruto
        public static double calculaBruto(int horas, double costo)
        {
            return costo * horas;
        }

        //Método que determina el descuento
        public static double calculaDescuento(double bruto)
        {
            return 0.12 * bruto;
        }
        //Método que calcula el neto
        public static double calculaNeto(double bruto, double descuento)
        {
            return bruto - descuento;
        }
    }
}
```

Observaciones en el código:

- Debe especificar que el método constructor debe ser privado. Además, no debe presentar parámetros y no debe tener implementación.
- Implemente el método asignaCostoHora que recibe como parámetro la categoría del empleado. Este será enviado desde el cuadro combinado del formulario.
- Implemente el método calculaBruto con los parámetros horas y el costo, que provienen de cajas de texto del formulario para determinar el monto bruto.
- Implemente el método calculaDescuento con los parámetros bruto para determinar el 12 %.

- Implemente el método calculaNeto con los parámetros bruto y descuento, que permiten aplicar la diferencia entre el bruto y el descuento.

3. Implemente el siguiente código en la clase frmSueldoEmpleado:

```
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 pjSueldoEmpleado
{
    public partial class frmSueldo : Form
    {
        public frmSueldo()
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los valores del formulario
            string nombre = txtEmpleado.Text;
            string categoria = cboCategoria.Text;
            int horas = int.Parse(txtHoras.Text);

            //Capturando los montos obtenidos desde la clase
            double costo = Empleado.asignaCostoHora(categoria);
            double bruto = Empleado.calculaBruto(horas, costo);
            double descuento = Empleado.calculaDescuento(bruto);
            double neto = Empleado.calculaNeto(bruto,descuento);

            //Imprimiendo los resultados
            ListViewItem fila = new ListViewItem(nombre);
            fila.SubItems.Add(categoria);
            fila.SubItems.Add(horas.ToString());
            fila.SubItems.Add(costo.ToString("C"));
            fila.SubItems.Add(bruto.ToString("C"));
            fila.SubItems.Add(descuento.ToString("C"));
            fila.SubItems.Add(neto.ToString("C"));
            lvPago.Items.Add(fila);
        }
    }
}
```

Observaciones en el código:

- Las variables nombre, categoría y horas reciben los valores de los controles del formulario. Estos valores serán enviados a los parámetros de los métodos estáticos.
- Para la invocación de los métodos de clase, no será necesario crear un objeto de la clase Empleado.
- Para realizar los cálculos, se debe invocar a los métodos por medio de la referencia de clase. Para lo cual, se hace referencia directamente a la clase Empleado, de esta forma se puede llegar directamente a los métodos de clase y no olvidarse que también se debe colocar los parámetros según la implementación realizada en la clase.

1.9 Método constructor

En Visual C#, cuando se construye un objeto de una determinada clase, resulta necesaria la inicialización de sus atributos. Para esto se tiene dos posibilidades: la primera hace referencia a los métodos get y set; y la segunda, al método constructor. El método constructor es un método especial que pertenece a la clase. Esta posee unas características especiales que se verán a continuación:

- Todo método constructor recibe el mismo nombre de su clase.
- Su misión es asignar valores, por eso no devuelve valor alguno, pero tampoco es de tipo void (método sin valor de retorno).
- Se pueden implementar varios métodos constructores dentro de una misma clase. A esto se le llama sobrecarga de métodos y estos se diferencian por los parámetros que albergan. Si existen varios métodos constructores al momento de crear un objeto, solo se usará uno de estos métodos.
- La implementación del método constructor dependerá del usuario, ya que si no se especifica serán métodos set, los cuales asignarán un valor inicial a los atributos.
- Se debe tener en cuenta que todo constructor debe tener visibilidad pública de forma obligatoria.

Para definir los constructores se emplea la siguiente sintaxis:

- Formato de un constructor vacío.

```
public MetodoConstructor()
{
}
```

EDITORIAL MACRO®

- Formato de un constructor con especificación de parámetros.

```
public MetodoConstructor(tipo parámetro)
{
  this.atributo  = parámetro;
}
```

- Formato de constructores sobrecargados.

```
public MetodoConstructor(tipo parámetro)
{
  this.atributo  = parámetro;
}
public MetodoConstructor(tipo parámetro,tipo parámetro)
{
  this.atributo  = parámetro;
}
public MetodoConstructor(tipo parámetro,tipo parámetro,tipo parámetro)
{
  this.atributo  = parámetro;
}
```

Si se tiene la clase Pago y se implementa el método constructor, se tiene las siguientes posibilidades de implementación:

```
//1. Constructor vacío
public class Pago
    {
        private int numPago;
        private DateTime fecha;
        private string deudor;

        public Pago(){

        }
    }

//2. Constructor con un solo parámetro
public class Pago
    {
        private int numPago;
        private DateTime fecha;
        private string deudor;

        public Pago(int numPago){
```

```
            this.numPago = numPago;
        }
    }

//3. Constructor con dos parámetros
public class Pago
    {
        private int numPago;
        private DateTime fecha;
        private string deudor;

        public Pago(int numPago, DateTime fecha){
            this.numPago = numPago;
            this.fecha = fecha;
        }
    }

//4. Constructor con tres parámetros
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;
        }
    }
```

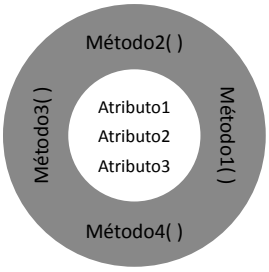
EDITORIAL MACRO®

La creación del objeto a partir de la implementación de los métodos constructores varía según los parámetros especificados:

```
//1. Objeto si el Constructor es vacio
Pago objP = new Pago();

//2. Constructor con un solo parámetro
int nPago = 10;
Pago objP = new Pago(nPago);

//3. Constructor con dos parámetros
int nPago = 10;
DateTime fechaPago = DateTime.Parse(DateTime.Now.ToString());
Pago objP = new Pago(nPago,fechaPago);

//4. Constructor con tres parámetros
int nPago = 10;
DateTime fechaPago = DateTime.Parse(DateTime.Now.ToString());
string deudor = "Juan Perez";
Pago objP = new Pago(nPago,fechaPago,deudor);
```

1.10 Los objetos

Un objeto representa un ejemplar de una clase. Es considerado como una unidad dentro de una aplicación, ya que presenta un comportamiento particular de una clase.



Se debe tener en cuenta que gracias a los objetos se puede acceder a los miembros de una determinada clase como sus atributos y métodos. Entre las características de los objetos se tiene:

- **Atributos**

  Representa la información que necesita la clase y que se usará dentro de los métodos implementados en la clase.

- **Métodos**

  Representa el comportamiento que puede tener una clase, es decir, son todas las acciones que el objeto puede realizar por sí mismo. Es a partir de aquí que una aplicación puede tener acceso a la lógica del negocio.

- **Identidad**

  Si se entiende que puede haber muchos objetos para una misma clase, entonces se debe tener en cuenta que cada objeto es totalmente independiente de los demás. Es así que se recomienda que, al crear un objeto, se le asigne un nombre representativo y único. Por ejemplo, el objeto de la clase Cliente podría llamarse objC.

  Para crear objetos, se emplea la siguiente sintaxis:

  ➢ **Formato 1**

```
Clase Objeto;
Objeto = new MetodoConstructor();
```

  Por ejemplo, si se tiene una clase llamada Cliente, la cual cuenta con un método constructor sin parámetros, el objeto se crearía de la siguiente manera:

  Cliente objC;

  objC = new Cliente();

  Entonces, primero se declara objC del tipo Cliente, esto se realiza como si se estuviera declarando una variable. Seguidamente, se debe crear el objeto con el operador new y haciendo referencia al método constructor de la clase.

  ➢ **Formato 2**

```
Clase Objeto = new MetodoConstructor();
```

EDITORIAL MACRO®

Por ejemplo, si se tiene una clase llamada Cliente, que cuenta con un método constructor sin parámetros, el objeto se crearía de la siguiente manera:

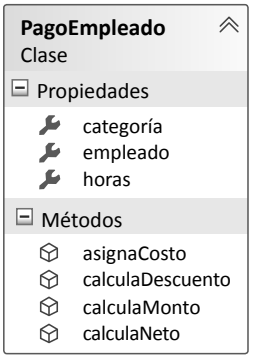
Cliente objC = new Cliente();

La idea es crear el objeto en una sola línea haciendo referencia a la clase y al mismo tiempo a su método constructor.

1.10.1 Enviando información a los atributos de la clase por medio del objeto

Cuando se crean los objetos, se puede tener acceso a todos los elementos públicos de la clase, es así que se llenará de información a los atributos de la siguiente manera:

Si se tiene la clase pagoEmpleado con la siguiente estructura de clase:



- **Enviando valores directos**

  Los valores pueden ser asignados directamente a los atributos de la clase pagoEmpleado, todo esto mediante el objeto objE.

```
//Creando el objeto objE de la clase pagoEmpleado
PagoEmpleado objE = new PagoEmpleado();

//Enviando los valores
objE.categoria = "Jefe";
objE.empleado = "Fernanda Torres La.";
objE.horas = 48;
```

- **Enviando valores indirectos**

  Los valores pueden ser asignados a variables locales y estas luego podrán ser asignadas a los atributos de la clase PagoEmpleado mediante el objeto objE.

```
//Declarando variables
string categoria = "Jefe";
string empleado = "Fernanda Torres La.";
int horas = 48;

//Creando el objeto objE de la clase pagoEmpleado
PagoEmpleado objE = new PagoEmpleado();

//Enviando los valores
objE.categoria = categoria;
objE.empleado = empleado;
objE.horas = horas;
```

MACRO®

- **Enviando valores calculados**

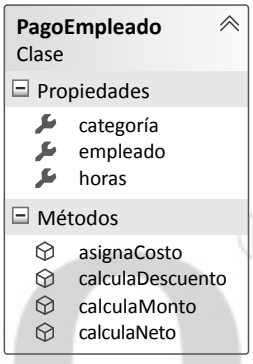
  Un campo calculado puede ser enviado como un valor o como una expresión a los parámetros, ya que esta al final siempre tiene un resultado, solo se debe tener cuidado con el tipo de datos.

```
//Declarando variables
string categoria = "Jefe";
string empleado = "Fernanda Torres La.";
int díasTrabajados = 6;
int horasDias = 6;

//Creando el objeto objE de la clase pagoEmpleado
Empleado objE = new Empleado();

//Enviando los valores
objE.categoria = categoria;
objE.empleado = empleado;
objE.horas = diastrabajados * horasDias;
```

1.10.2 Usando los valores de clase

Toda vez que los valores ingresan a la clase por medio de los objetos, se podrán usar y reutilizar tantas veces se necesite. Lo que se verá ahora son cuáles son las posibilidades de su uso.

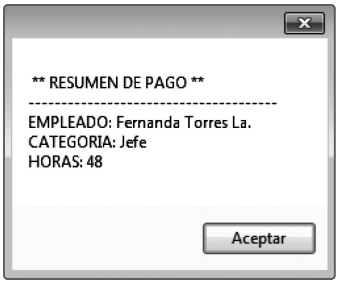
Si se tiene la clase PagoEmpleado que cuenta con la siguiente estructura:



- **Imprimir los valores directamente**

  Se enviará los valores por medio de controles Windows Form.

  - En un cuadro de mensaje (MessageBox).



EDITORIAL MACRO®

```
//Declarando variables
string categoría = "Jefe";
string empleado = "Fernanda Torres La.";
int horas = 48;

//Creando el objeto de la clase pagoEmpleado
PagoEmpleado objE = new PagoEmpleado();

//Enviando los valores
objE.empleado = empleado;
objE.categoria = categoría;
objE.horas = horas;

//Mostrando la información con MessageBox
MessageBox.Show(" ** RESUMEN DE PAGO ** \n" +
                "--------------------------------------" +
                "\nEMPLEADO: " + objE.empleado+
                "\nCATEGORIA: " + objE.categoria+
                "\nHORAS: " + objE.horas);
```

- En un cuadro de lista (ListBox).

```
 ** RESUMEN DE PAGO **
-----------------------------------------
EMPLEADO: Fernanda Torres La.
CATEGORIA: Jefe
HORA: 48
```

```
//Declarando variables
string categoría = "Jefe";
string empleado = "Fernanda Torres La.";
int horas = 48;

//Creando el objeto de la clase pagoEmpleado
PagoEmpleado objE = new PagoEmpleado();

//Enviando los valores
objE.empleado = empleado;
objE.categoria = categoría;
objE.horas = horas;

//Mostrando la información en un cuadro de lista
lstResumen.Items.Add(" ** RESUMEN DE PAGO ** ");
lstResumen.Items.Add("-----------------------------------------");
lstResumen.Items.Add("EMPLEADO: "+objE.empleado);
lstResumen.Items.Add("CATEGORIA: "+objE.categoria);
lstResumen.Items.Add("HORA: "+objE.horas);
```

- En un control ListView.

EMPLEADO	CATEGORIA	HORAS
Fernanda Torres La.	Jefe	48

```
//Declarando variables
string categoría = "Jefe";
string empleado = "Fernanda Torres La.";
int horas = 48;

//Creando el objeto de la clase pagoEmpleado
PagoEmpleado objE = new PagoEmpleado();

//Enviando los valores
objE.empleado = empleado;
objE.categoria = categoría;
objE.horas = horas;

//Mostrando la información en el Listview
ListViewItem fila = new ListViewItem(objE.empleado);
fila.SubItems.Add(objE.categoria);
fila.SubItems.Add(objE.horas.ToString());
lvResumen.Items.Add(fila);
```

- **Realizar cálculos**

  Los datos obtenidos desde la clase, ya sea atributos o métodos, se pueden usar en la implementación de cálculos y es así que se cumplirá con la encapsulación.

EMPLEADO	CATEGORIA	HORAS	CONDICION
Fernanda Torres La.	Jefe	48	Sobretiempo

```
//Declarando variables
string categoría = "Jefe";
string empleado = "Fernanda Torres La.";
int horas = 48;

//Creando el objeto de la clase pagoEmpleado
PagoEmpleado objE = new PagoEmpleado();

//Enviando los valores
objE.empleado = empleado;
objE.categoria = categoría;
objE.horas = horas;

//Determinando si hay hora de sobretiempo
string condicion = "";
if (objE.horas > 45)
   condicion = "Sobretiempo";
   else
   condicion = "Tiempo correcto";

//Mostrando la información en el Listview
ListViewItem fila = new ListViewItem(objE.empleado);
fila.SubItems.Add(objE.categoria);
fila.SubItems.Add(objE.horas.ToString());
fila.SubItems.Add(condicion);
lvResumen.Items.Add(fila);
```

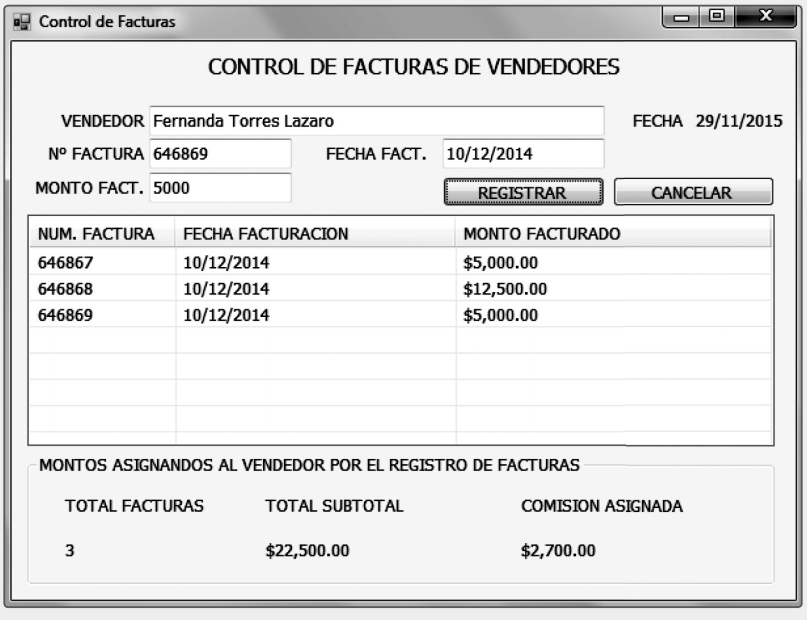
Caso desarrollado 4 : Método constructor - Control de facturas de vendedores

Implementar una aplicación que permita controlar el registro de las facturas que realizan los vendedores en una tienda comercial. Para lo cual, se deberá registrar el nombre del vendedor, el número de factura, la fecha de la facturación y el monto facturado.

Se tiene en cuenta lo siguiente:

- Mostrar automáticamente la fecha de registro de la factura.
- Los datos del registro deberán mostrarse en un cuadro de lista.
- Implementar un método constructor que reciba como parámetros los datos del registro, como el número de la factura, la fecha y el monto facturado.
- Al final, determinar el total de facturas registradas, el monto acumulado y la comisión asignada al vendedor por la venta realizada.

GUI Propuesto:



Solución:

1. Haga clic en **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjControlFacturas.

2. Agregue al proyecto la clase Factura e implemente el siguiente código:

Clase Factura:

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

namespace pjControlFacturas
{
    public class Factura
    {
        //Declaración de atributos privados
        private int _numFactura;
        private DateTime _fechaFact;
        private double _montoFact;

        static private double acumulado;
        static private int n;

        //Método constructor con parámetros
       public Factura(int numFactura, DateTime fechaFact,double montoFact )
        {
            this._numFactura = numFactura;
            this._fechaFact = fechaFact;
            this._montoFact = montoFact;
            acumulado += montoFact;
            n++;
        }

        //Metodos GET"s
        public int numFactura
        {
            get { return _numFactura; }
        }
        public DateTime fechaFact
        {
            get { return _fechaFact; }
        }
        public double montoFact
        {
            get { return _montoFact; }
        }
```

```
        public int totalFacturas()
        {
            return n;
        }
        public double calculaTotalSubtotal()
        {
            return acumulado;
        }
        public double calculaComision()
        {
            return 0.12 * calculaTotalSubtotal();
        }
    }
}
```

Observaciones en el código:

En la clase Factura se declaran como atributos privados a los elementos del control de factura, como el número de la factura, la fecha y el monto facturado. Este tipo de declaración se debe a que los valores que vendrán del formulario serán recepcionados por el método constructor. En la implementación del constructor se asignan los valores privados con los valores recibidos por los parámetros del método constructor. Además, se totalizan los montos facturados en una variable estática llamada «acumulado». Eso indica que, al crear el objeto de la clase Factura, automáticamente se acumulará los montos facturados y, de la misma forma, trabajará la variable *n* que tiene la misión de determinar cuántos objetos de la clase Factura se han creado.

En la misma clase se implementan los métodos get de los tres atributos privados de la clase Factura, solo se necesita este método, ya que el constructor se encarga de asignar valor a dichos atributos. Entonces, se llega a la conclusión de que no se debe implementar los métodos set por el trabajo que realiza el constructor.

También se implementa el método totalfacturas, que es el encargado de devolver el total de facturas registradas. Esta variable se incrementa cada vez que se hace referencia al método constructor, es decir, al crear un objeto de la clase Factura. De la misma manera, trabaja el método calculaTotalSubtotal que permite devolver el acumulado de los montos facturados enviados como parámetro del método constructor.

Finalmente, el método calculaComision tiene la misión de determinar el monto de comisión que se le asigna al vendedor por la venta realizada.

Clase frmFactura:

```
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 pjControlFacturas
{
    public partial class frmFacturas : Form
    {
        public frmFacturas()
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Capturando los valores del formulario
            int numFactura = int.Parse(txtNumFact.Text);
            DateTime fechaFact = DateTime.Parse(txtFechaFact.Text);
            double montoFact = double.Parse(txtMontoFact.Text);

            //Objeto de la clase Factura
            Factura objF = new Factura(numFactura,fechaFact,montoFact);

            //Imprimiendo en la lista
            ListViewItem fila = new ListViewItem(objF.numFactura.ToString());
            fila.SubItems.Add(objF.fechaFact.ToShortDateString());
            fila.SubItems.Add(objF.montoFact.ToString("C"));
            lvFacturas.Items.Add(fila);

            //Mostrando los montos
            lblTotalFact.Text = objF.totalFacturas().ToString();
            lblTotalSub.Text = objF.calculaTotalSubtotal().ToString("C");
            lblComision.Text = objF.calculaComision().ToString("C");
        }

        private void frmFacturas_Load(object sender, EventArgs e)
        {
            lblFecha.Text = muestraFecha();
        }

        //Función lambda que muestra la fecha actual
       Func<String> muestraFecha = () => DateTime.Now.ToShortDateString();
    }
}
```

Se debe iniciar por implementar el método lambda, llamado muestraFecha, de tal forma que dicho método devuelva la fecha corta, usando la función ToShortDateString. Esta función será invocada en el evento load del formulario. Se debe recordar que toda función lambda siempre devuelve valor y esta será asignada al control lblFecha.

En el botón Registrar se inicia capturando los valores desde el formulario y justamente estos valores se enviarán por los parámetros del método constructor. Cuando se crea el objeto de la clase Factura, se especifica en los parámetros los valores obtenidos desde el formulario, solo se debe considerar que los tipos de datos deben ser compatibles. Luego, se envía los valores al control ListView mediante el objeto de la clase ListViewItem.

1.11 Herencia de clases

También es conocida como generalización. Permite agrupar valores comunes de una determinada entidad en una sola, formando así una nueva entidad genérica del cual se podrán instanciar todas las clases que deseen hacerlo, pero esta vez no será una asociación normal sino una generalizada o heredada.

«La herencia permite crear nuevas clases a partir de clases existentes. La herencia puede simplificar el diseño de la aplicación proporcionando una estructura de relaciones entre las distintas clases» (Fuente: https://msdn.microsoft.com/es-es/library/yet634fk(v=vs.90).aspx). Esta nueva clase es llamada «derivada» y posee todas las características de la clase heredada.

Entonces, en una relación de herencia existe un tipo de clase llamada derivada, el cual se desprende de otro llamado base o clase padre, de tal forma que el espacio de declaración del tipo derivado contiene implícitamente todos los miembros de tipo no constructor del tipo base.

Consideraciones generales de la herencia:

- De forma predeterminada, todas las clases pueden ser heredables. Las clases pueden heredar de otras clases del proyecto o de clases en otros ensamblados a los que hace referencia el proyecto.
- Para evitar la exposición de elementos restringidos en una clase base, el tipo de acceso de una clase derivada debe ser igual o más restrictivo que el de su clase base. Por ejemplo, una clase Public no puede heredar una clase Friend o Private y una clase Friend no puede heredar una clase Private.

EDITORIAL MACRO®

Caso desarrollado 5 : Herencia - Venta de productos al contado y crédito

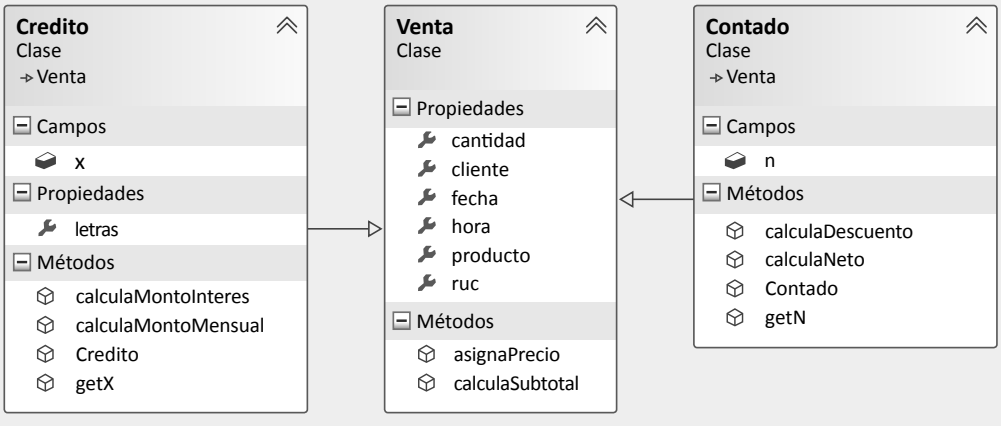
Implementar una aplicación que permita gestionar la venta de productos en una tienda comercial de artefactos electrónicos. Tanto para la venta al contado o crédito, se debe solicitar el nombre o razón social del cliente, número de RUC, así como la fecha de la venta y la hora de registro.

Se tiene en cuenta lo siguiente:

- Implementar la clase Venta, que contenga los atributos nombre del cliente, número de RUC, fecha y hora de registro, nombre del producto y la cantidad comprada.
- Implementar el método asignaPrecio dentro de la clase Venta, que a partir del nombre del producto le asigne un precio.
- Los precios de los productos se basan en la siguiente tabla:

Descripción del producto	Precio
Lavadora	$1500.00
Refrigeradora	$3500.00
Licuadora	$500.00
Extractora	$150.00
Radiograbadora	$750.00
DVD	$100.00
Blu Ray	$250.00

- Implementar el método calculaSubtotal, el cual permitirá calcular el monto subtotal entre el precio del producto y la cantidad seleccionada por el cliente.
- Implementar las siguiente clases:



- Implementar un formulario inicial. A partir del tipo de venta se debe seleccionar un tipo de formulario como se muestra en la siguiente imagen:

GUI Propuesta para la venta de productos al contado:

Se tiene en cuenta lo siguiente:

- La clase Contado es considerada como clase derivada y debe heredar de la clase Venta.
- Declarar la variable *n* que permite determinar la cantidad de productos que el cliente está comprando. Este valor debe ser devuelto por un método llamada getN. Usar el método constructor para dicho conteo.

- Implementar el método calculaDescuento, que a partir del monto subtotal enviado como parámetro se pueda determinar el monto de descuento, el cual se basa en la siguiente tabla:

Monto Subtotal	Porcentaje de Descuento
Menor a $1000.00	2 %
Entre $1000.00 y $3000.00	5 %
Mayor a $3000.00	12 %

GUI Propuesta para la venta de productos al crédito:

Se tiene en cuenta lo siguiente:

- La clase Crédito es considerada como clase derivada y debe heredar de la clase Venta.
- Se debe declarar la variable *x* que permite determinar la cantidad de productos que el cliente está comprando. Este valor debe ser devuelto por un método llamado getX. Use el método constructor para dicho conteo.
- Por tratarse de una venta al crédito, declarar el atributo letras de tipo entero y asignar sus métodos get y set.

- Mostrar el monto mensual según la cantidad de letras seleccionadas por el cliente, además de mostrar el monto total a pagar.
- Para calcular el monto mensual del crédito, se debe basar en la siguiente tabla:

Monto subtotal	Porcentaje de descuento
Menor a $1000.00	2 %
Entre $1000.00 y $3000.00	5 %
Mayor a $3000.00	12 %

Solución:

1. Haga clic en **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjVentaHeredada.
2. Agregue al proyecto la clase Venta e implemente el siguiente código:

Clase base: Venta

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace pjVentaHeredada
{
    public class Venta
    {
        //Atributos
        public string cliente { get; set; }
        public string ruc { get; set; }
        public DateTime fecha { get; set; }
        public DateTime hora { get; set; }
        public string producto { get; set; }
        public int cantidad { get; set; }

        //Metodos de la clase Venta
        public double asignaPrecio()
        {
            switch (producto)
            {
                case "Lavadora": return 1500;
                case "Refrigeradora": return 3500;
                case "Licuadora": return 500;
                case "Extractora": return 150;
                case "Radiograbadora": return 750;
                case "DVD": return 100;
```

```
                case "BluRay": return 250;
            }
            return 0;
        }

        //Método que calcula el subtotal
        public double calculaSubtotal()
        {
            return asignaPrecio() *cantidad;
        }
    }
}
```

La clase base Venta cuenta con los atributos que serán heredados a las clases Contado y Crédito. Asimismo, se implementa el método asignaPrecio que permite asignar un precio a un determinado producto. El método calculaSubtotal permite calcular el subtotal a pagar por un determinado producto.

Clase derivada: Contado

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

namespace pjVentaHeredada
{
    public class Contado:Venta
    {
        public static int n;
        public Contado()
        {
            n++;
        }

        public int getN()
        {
            return n;
        }

        //Metodos de la clase contado
        public double calculaDescuento(double subtotal)
        {
            if (subtotal < 1000)
                return 2.0 / 100 * subtotal;
            else if (subtotal >= 1000 && subtotal <= 3000)
                return 5.0 / 100 * subtotal;
            else
```

```
                return 12.0 / 100 * subtotal;
        }
        public double calculaNeto(double subtotal, double descuento)
        {
            return subtotal - descuento;
        }
    }
}
```

Se inicia la clase Contado heredando de la clase Venta mediante el código Contado:Venta. Con esta sentencia, se hace que la clase Venta herede todos sus atributos y métodos a la clase Contado. Asimismo, se declara la variable *n* que permitirá contar la cantidad de objetos creados de la clase Contado. Este aumento se realizará en el método constructor de la clase Contado.

Por otro lado, se debe implementar los métodos getN, que son los encargados de devolver el valor de N. El método calculaDescuento determina el monto de descuento según el monto subtotal recibido como parámetro y, finalmente, calculaNeto determina el monto neto a pagar por la compra al contado de un producto.

Clase derivada: Crédito

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

namespace pjVentaHeredada
{
    public class Credito:Venta
    {
        public static int x;
        public Credito()
        {
            x++;
        }

        public int getX()
        {
            return x;
        }

        //Atributos de la clase Crédito
        public int letras { get; set; }

        //Metodos de la clase Crédito
        public double calculaMontoInteres()
```

```
        {
            switch (letras)
            {
                case 3: return 5.0 / 100 * calculaSubtotal();
                case 6: return 10.0 / 100 * calculaSubtotal();
                case 9: return 15.0 / 100 * calculaSubtotal();
                case 12: return 25.0 / 100 * calculaSubtotal();
            }
            return 0;
        }

        public double calculaMontoMensual()
        {
            return (calculaSubtotal()+calculaMontoInteres())/letras;
        }
    }
}
```

De la misma forma que la clase Contado, la clase Crédito hereda de la clase Venta con la sentencia Credito:Venta. Asimismo, la variable *x* tiene la misión de contar el total de objetos creados de la clase Crédito. También se declara la propiedad letras, ya que es el valor que se usará para los cálculos implementados en los métodos posteriores de la clase Crédito.

También se implementa los métodos calculaMontoInteres, que tienen la misión de determinar el monto de interés que le corresponde pagar al cliente según la cantidad de letras seleccionadas por el cliente. El método calculaMontoMensual determina el pago que debe realizar el cliente mensualmente según la cantidad de letras seleccionadas.

Clase base: frmPrincipal

```
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 pjVentaHeredada
{
    public partial class frmPrincipal : Form
    {
        public frmPrincipal()
```

```
        {
            InitializeComponent();
        }

        private void btnCredito_Click(object sender, EventArgs e)
        {
            frmCredito frmCre = new frmCredito();
            frmCre.Show();
        }

        private void btnContado_Click(object sender, EventArgs e)
        {
            frmContado frmC = new frmContado();
            frmC.Show();
        }
    }
}
```

Clase base: frmContado

```
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 System.Collections;

namespace pjVentaHeredada
{
    public partial class frmContado : Form
    {
        //Inicialización del arreglo de productos
         static string[] productos = { "Lavadora","Refrigeradora","Licuadora",
        "Extractora","Radiograbadora","DVD","BluRay"};

        //Creando el objeto de la clase ArrayList
        ArrayList aProductos = new ArrayList(productos);

        //Variable acumuladora de totales
        double tSubtotal = 0;

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

```
private void btnAdquirir_Click(object sender, EventArgs e)
{
    //Objeto de la clase Contado
    Contado objC = new Contado();

    //Datos del cliente
    objC.cliente = txtCliente.Text;
    objC.ruc = txtRuc.Text;
    objC.fecha =DateTime.Parse(lblFecha.Text);
    objC.hora = DateTime.Parse(lblHora.Text);

    //Datos del producto
    objC.producto = cboProducto.Text;
    objC.cantidad = int.Parse(txtCantidad.Text);

    //Imprimiendo en la lista
    ListViewItem fila = new ListViewItem(objC.getN().ToString());
    fila.SubItems.Add(objC.producto);
    fila.SubItems.Add(objC.cantidad.ToString());
    fila.SubItems.Add(objC.asignaPrecio().ToString("C"));
    fila.SubItems.Add(objC.calculaSubtotal().ToString());
    lvDetalle.Items.Add(fila);

    listado(objC);
}

private void frmContado_Load(object sender, EventArgs e)
{
    cboProducto.DataSource = aProductos;
    mostrarFecha();
    mostrarHora();
}
void mostrarFecha()
{
    lblFecha.Text = DateTime.Now.ToShortDateString();
}
void mostrarHora()
{
    lblHora.Text = DateTime.Now.ToShortTimeString();
}
void listado(Contado objC)
{
    tSubtotal += objC.calculaSubtotal();
    lstResumen.Items.Clear();
    lstResumen.Items.Add("** RESUMEN DE VENTA **");
  lstResumen.Items.Add("-------------------------------------");
    lstResumen.Items.Add("CLIENTE: " + objC.cliente);
    lstResumen.Items.Add("RUC: " + objC.ruc);
    lstResumen.Items.Add("FECHA: " + objC.fecha);
    lstResumen.Items.Add("HORA: " + objC.hora);
  lstResumen.Items.Add("-------------------------------------");
```

```
            lstResumen.Items.Add("SUBTOTAL: " + tSubtotal.ToString("C"));

                    double descuento = objC.calculaDescuento(tSubtotal);
            double neto = objC.calculaNeto(tSubtotal, descuento);

           lstResumen.Items.Add("DESCUENTO: " + descuento.ToString("C"));
            lstResumen.Items.Add("NETO: " + neto.ToString("C"));

            //Hallar el monto total sin descuento
            lblNeto.Text = neto.ToString("C");
        }
    }
}
```

En el formulario frmContado se inicializa el arreglo «productos» con la descripción de los productos que se listarán en el cuadro combinado. Estos productos serán enviados a la colección ArrayList llamada aProductos y es desde aquí que se enviará al control cuadro combinado.

Para mostrar el monto total acumulado, se declarará la variable tSubtotal, que será la encargada de acumular los montos resultantes del subtotal.

En el evento load del formulario se debe llenar el cuadro combinado de productos con la sentencia cboProducto.DataSource=aProductos, donde aProductos representa al objeto ArrayList que contiene la descripción de los productos, además de mostrar la fecha y la hora actual. Se debe tener en cuenta que el formato corto de la hora se muestra con ToShortDateString y la hora corta con ToShortTimeString.

El botón Adquirir permite registrar los datos ingresados en el formulario contado, es por eso que se inicia creando un objeto de la clase Contado. Después, se envía los datos mediante el objeto que luego será impreso en el control ListView. Asimismo, se implementa el método listado que tiene la misión de mostrar el resumen de la venta. Los datos de la venta, así como los montos subtotal, descuento y neto son calculados en la clase Contado.

Clase base: frmCredito

```
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 System.Collections;
namespace pjVentaHeredada
```

```
{
    public partial class frmCredito : Form
    {
        static int[] letras = { 3,6,9,12 };
       static string[] productos = { "Lavadora","Refrigeradora","Licuadora",
                                     "Extractora","Radiograbadora","DVD","BluRay"};

        //Declaración de los ArrayList
        ArrayList aProductos = new ArrayList(productos);
        ArrayList aLetras = new ArrayList(letras);

        double tSubtotal = 0;

        public frmCredito()
        {
            InitializeComponent();
        }

        private void frmCredito_Load(object sender, EventArgs e)
        {
            cboLetras.DataSource = aLetras;
            cboProducto.DataSource = aProductos;
            mostrarFecha();
            mostrarHora();
        }

        void mostrarFecha()
        {
            lblFecha.Text = DateTime.Now.ToShortDateString();
        }
        void mostrarHora()
        {
            lblHora.Text = DateTime.Now.ToShortTimeString();
        }

        private void btnAdquirir_Click(object sender, EventArgs e)
        {
            //Objeto de la clase Crédito
            Crédito objCr = new Crédito();

            //Datos del cliente
            objCr.cliente = txtCliente.Text;
            objCr.ruc = txtRuc.Text;
            objCr.fecha = DateTime.Parse(lblFecha.Text);
            objCr.hora = DateTime.Parse(lblHora.Text);

            //Datos del producto
            objCr.producto = cboProducto.Text;
            objCr.cantidad = int.Parse(txtCantidad.Text);

            //Imprimiendo en la lista
           ListViewItem fila = new ListViewItem(objCr.getX().ToString());
            fila.SubItems.Add(objCr.producto);
```

```
            fila.SubItems.Add(objCr.cantidad.ToString());
            fila.SubItems.Add(objCr.asignaPrecio().ToString("C"));
            fila.SubItems.Add(objCr.calculaSubtotal().ToString());
            lvDetalle.Items.Add(fila);
            tSubtotal += objCr.calculaSubtotal();
            lblMonto.Text = tSubtotal.ToString("0.00");
        }
        void montoLetras(int le)
        {
            double montoMensual = double.Parse(lblMonto.Text) / le;

            lvResumen.Items.Clear();
            for(int i=1;i<=le; i++)
            {
                ListViewItem fila = new ListViewItem(i.ToString());
                fila.SubItems.Add(montoMensual.ToString("C"));
                lvResumen.Items.Add(fila);
            }
        }

        private void btnMostrar_Click(object sender, EventArgs e)
        {
            int letras = int.Parse(cboLetras.Text);
            switch (letras)
            {
                case 3: montoLetras(3); break;
                case 6: montoLetras(6); break;
                case 9: montoLetras(9); break;
                case 12: montoLetras(12); break;
            }
        }
    }
}
```

En el formulario frmCredito se inicia declarando dos arreglos: el primero será para el número de letras del crédito y el segundo para los productos que vende la tienda comercial. Asimismo, debe declararse dos objetos de tipo ArrayList para cada uno de los arreglos. Adicionalmente, se declara la variable tSubtotal para acumular los montos subtotales productos de la venta.

En el evento load del formulario se debe cargar las letras y los productos en sus respectivos cuadros combinados, además de mostrar la fecha y la hora actual por medio de métodos.

El método montoLetras tiene la misión de determinar el monto mensual a pagar según la cantidad de letras seleccionadas. Estos datos serán impresos en un cuadro de lista como un resumen de la venta al crédito. En el botón Mostrar se invocará al método montoLetras y así poder mostrar los montos según la selección de letras del cliente.

1.12 Métodos polifórmicos

Representa las muchas formas que puede tener un método de una clase cuando estas se encuentran heredadas. Es decir, los métodos tendrán el mismo nombre, pero realizarán tareas distintas. También se debe considerar que la única forma de diferenciarlos será por la cantidad parámetros que contenga dicho método.

Por otro lado, la clase base debe contener los métodos originales, mientras que las clases derivadas deben contener los métodos polifórmicos.

1.12.1 Modificadores del polimorfismo

De acuerdo al concepto inicial, si un método polifórmico tiene el mismo nombre pero diferente implementación, entonces se debe usar modificadores que permitan especificar cuál es el método polifórmico y quién lo sobreescribe. Se tiene dos modificadores:

- **Modificador virtual**

  Se utiliza para especificar qué método será modificado desde una clase derivada. Por ejemplo, se observa que la implementación de un método virtual calcula el sueldo de un trabajador, que cuenta con sus horas trabajadas y un costo hora.

```
public virtual double calculaSueldo()
{
    return horasTrabajadas * costoHora;
}
```

La implementación de un miembro virtual puede reemplazarse por un miembro de reemplazo de una clase derivada. Consideraciones generales del método virtual:

- Al invocar a un método virtual, se verifica si existe un método que lo sobrescribirá en la clase derivada.
- Cuando se invoca a un método sobrescrito, se busca al método de mayor grado de derivación, es decir, el cual se encuentra en la clase base.
- El nombre del método sobrescrito debe ser igual al método original de la clase base.
- Se debe tener en cuenta que todo método implementado en cualquier clase no es virtual, ya que esta se especifica cuando hay una actividad polifórmica.
- Se debe tener en cuenta que no se puede sobrescribir un método no especificado como virtual.
- No se puede usar el modificador virtual cuando el método especificado es static, abstract, private u override.

EDITORIAL MACRO®

- **Modificador override**

  El modificador override es necesario para ampliar o modificar la implementación virtual de un método heredado. Consideraciones generales del modificador override:

  - Un método override proporciona una nueva forma de implementación de un método virtual que se encuentra en una clase base. Como requisito, el método de la clase base debe tener el mismo nombre del método override.
  - Una declaración override no puede cambiar la visibilidad del método virtual. Tanto el método sobrescrito como el método virtual deben tener el mismo nivel de visibilidad.
  - A continuación se puede visualizar el método calculaSueldo sobrescrito:

```
public Override double calculaSueldo()
{
    return horasTrabajadas * costoHora;
}
```

Caso desarrollado 6 : Polimorfismo - Control de evaluaciones

Un docente del curso de Programación I necesita una aplicación que permita controlar las notas de sus alumnos, de tal forma que pueda registrar el nombre completo del estudiante, tres evaluaciones y una evaluación actitudinal. La aplicación debe mostrar el promedio de las evaluaciones y una condición que muestre si el alumno está aprobado o no.

Asimismo, se debe mostrar en un cuadro estadístico la suma de promedios, el promedio más alto, el promedio más bajo, el total de aprobados y el total de desaprobados.

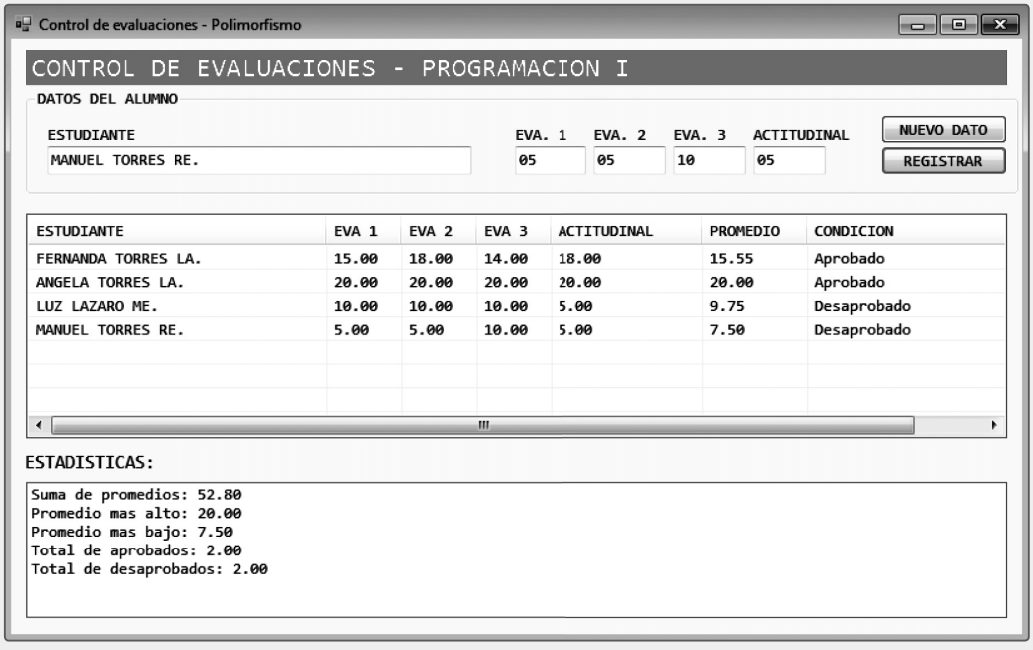
Se tiene en cuenta lo siguiente:

- Implementar una clase base llamada Promedio que registre los datos del alumno como su nombre, sus tres evaluaciones y su nota actitudinal.

El promedio de un alumno de pregrado tendrá la siguiente formula: 15 % de la evaluación práctica 1,20 % de la evaluación práctica 2,25 % de la evaluación práctica 3,30 % de la evaluación y 10 % de la nota actitudinal. En tanto, el alumno regular tendrá como promedio la media aritmética de sus notas.

Al tener dos tipos de alumnos, se necesita crear una clase llamada promedio que contenga todos los atributos especificados en el problema y otra clase llamada promedioProgramacion.

Se debe considerar, además, que se debe validar los datos registrados en los controles del formulario, de tal manera que envíe un mensaje indicando al usuario dónde se generó el error y además ubicar el foco en dicho control.



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjPolimorfismo.
2. Agregue al proyecto la clase Promedio e implemente el siguiente código:

- **Clase base: Promedio**

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

namespace pjPolimorfismo
{
    public class Promedio
    {
        //Atributos
        public string estudiante { get; set; }
        public int evaluacion1 { get; set; }
        public int evaluacion2 { get; set; }
        public int evaluacion3 { get; set; }
```

```
        public int actitudinal { get; set; }

        //Método para calcular el promedio
        public virtual double calculaPromedio()
        {
         return (evaluacion1 + evaluacion2 + evaluacion3 + actitudinal) / 4;
        }

        //Método para determinar la condicion del estudiante
        public string determinaCondicion()
        {
            if (calculaPromedio() < 12.5)
                return "Desaprobado";
            else
                return "Aprobado";
        }
    }
}
```

La clase promedio representa la clase base de la aplicación, por lo tanto todos los atributos declarados en esta sección serán heredados a las clases derivadas. Asimismo, se implementa el método calculaPromedio que tiene la misión de determinar el promedio de notas del alumno según las cuatro evaluaciones. Este método debe especificar que se trata de un método virtual y que podrá ser sobrescrito por otro método en las clases derivadas y así se aplicará el polimorfismo.

También se implementa el método determinaCondicion que permite mostrar el mensaje de aprobado o desaprobado según el promedio del alumno.

- **Clase derivada: PromedioProgramacion**

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

namespace pjPolimorfismo
{
  public class PromedioProgramacion: Promedio
  {
    //Método para calcular el promedio de programación
    public override double calculaPromedio()
    {
      return evaluacion1*0.15 + evaluacion2*0.3 + evaluacion3*0.5 +actitudinal*0.05;
    }
  }
}
```

La clase PromedioProgramacion representa a la clase derivada de la aplicación y tiene la misión de heredar los atributos de la clase Promedio y sobrescribir el método calculaPromedio, ya que en el curso de programación el promedio se calcula por porcentajes en cada evaluación y no por una división simple como en la clase base.

- **Clase: frmEvaluacion**

```
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 pjPolimorfismo
{
    public partial class frmProgramacion : Form
    {
        public frmProgramacion()
        {
            InitializeComponent();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Objeto de la clase Promedio Programación
            PromedioProgramacion objP = new PromedioProgramacion();

            //Enviando los valores a la clase
            objP.estudiante = txtEstudiante.Text;
            objP.evaluacion1 = int.Parse(txtEva1.Text);
            objP.evaluacion2 = int.Parse(txtEva2.Text);
            objP.evaluacion3 = int.Parse(txtEva3.Text);
            objP.actitudinal = int.Parse(txtActitudinal.Text);

            //Calculando el promedio
            double promedio = objP.calculaPromedio();
            string condicion = objP.determinaCondicion();

            //Imprimir en la lista
            ListViewItem fila = new ListViewItem(objP.estudiante);
            fila.SubItems.Add(objP.evaluacion1.ToString("0.00"));
            fila.SubItems.Add(objP.evaluacion2.ToString("0.00"));
            fila.SubItems.Add(objP.evaluacion3.ToString("0.00"));
            fila.SubItems.Add(objP.actitudinal.ToString("0.00"));
            fila.SubItems.Add(objP.calculaPromedio().ToString("0.00"));
```

```
            fila.SubItems.Add(objP.determinaCondicion());
            lvEvaluaciones.Items.Add(fila);

            //Imprimir las estadísticas
            estadísticas();
        }

        //Método que determina la suma de todos los promedios
        double sumaPromedios()
        {
            double suma = 0;
            for (int i = 0; i < lvEvaluaciones.Items.Count; i++)
            {
              suma += double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text);
            }
            return suma;
        }

        //Método que determina cual es el promedio mas alto
        double promedioMasAlto()
        {
            double mayor = 0;
            for (int i = 0; i < lvEvaluaciones.Items.Count; i++)
            {
             if (double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text) > mayor)
                {
                mayor = double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text);
                }
            }
            return mayor;
        }

        //Método que determina cual es el promedio mas bajo
        double promedioMasBajo()
        {
            double menor = int.MaxValue;
            for (int i = 0; i < lvEvaluaciones.Items.Count; i++)
            {
             if (double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text) < menor)
                {
                menor = double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text);
                }
            }
            return menor;
        }

        //Método que determina el total de aprobados
        int totalAprobados()
        {
            int cAprobados = 0;
```

```
            for (int i = 0; i < lvEvaluaciones.Items.Count; i++)
            {
              if (double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text) >10)
                {
                    cAprobados++;
                }
            }
            return cAprobados;
        }

        //Método que determina el total de desaprobados
        int totalDesaprobados()
        {
            int cDesaprobados = 0;
            for (int i = 0; i < lvEvaluaciones.Items.Count; i++)
            {
             if (double.Parse(lvEvaluaciones.Items[i].SubItems[5].Text) <= 10)
                {
                    cDesaprobados++;
                }
            }
            return cDesaprobados;
        }

    void estadísticas()
    {
      lstR.Items.Clear();
     lstR.Items.Add("Suma de promedios:" + sumaPromedios().ToString("0.00"));
     lstR.Items.Add("Promedio mas alto:" + promedioMasAlto().ToString("0.00"));
     lstR.Items.Add("Promedio mas bajo:" + promedioMasBajo().ToString("0.00"));
     lstR.Items.Add("Total de aprobados:" + totalAprobados().ToString("0.00"));
     lstR.Items.Add("Total de desaprobados:"+totalDesaprobados().ToString("0.00"));
    }
  }
}
```

En el botón **Registrar** se crea un objeto de la clase PromedioProgramacion, ya que siendo una clase derivada de la clase promedio hereda todos sus atributos y métodos. Es así que se puede enviar todos los valores registrados en el formulario, como el nombre del estudiante y sus cuatro evaluaciones. Luego, se imprimen los valores del estudiante en un control ListView, obteniendo el promedio y la condición a partir de los métodos implementados en las clases Promedio y PromedioProgramacion.

Asimismo, se implementa los siguientes métodos:

- **SumaPromedios**

  Tiene la misión de recorrer por todos los promedios mostrados en el control ListView y acumular los valores para determinar la suma acumulada de promedios. Hay que tener en cuenta que Item[i] representa a las filas, mientras que SubItems[5] representa a la columna seis del control ListView y este corresponde al promedio de notas de los alumnos.

- **PromedioMasAlto**

  Tiene la misión de determinar el promedio más alto encontrado en los promedios de los alumnos. Para esto, se declara la variable mayor con un valor inicial cero, ya que cualquier promedio sería mayor que cero y así poder encontrar el mayor de los promedios recorriendo todos los promedios desde el control ListView. Se debe recordar que SubItems[5] representa a la sexta columna del control, el cual contiene todos los promedios.

- **PromedioMasBajo**

  Hace el mismo trabajo que el método promedioMasAlto, la diferencia radica en que la variable que encontrará no debe inicializarse en cero, ya que ese valor siempre será menor que todos los promedios. Por eso, se inicializa con el valor más alto que tiene el tipo de datos int, el cual se especifica con la sentencia int.MaxValue.

EDITORIAL MACRO®

2

Capítulo

Colecciones

Capacidad terminal:

Implementar aplicaciones de mantenimiento de registros usando colecciones, como listas, pilas, colas y arrayList, reconociendo su método constructor, así como sus principales métodos.

EDITORIAL MACRO®

Contenido

- Las colecciones en .Net Framework
- Estructura foreach
- Clases de colecciones
- Clase ArrayList
- Clase List<>
- Clase Stack
- Clase Queue

Caso desarrollado 1: ArrayList - Control de registro de personal
Caso desarrollado 2: List - Control de registro de productos
Caso desarrollado 3: Stack - Control de registro de alumnos
Caso desarrollado 4: Queue - Control de registro de libros

EDITORIAL
MACRO®

2.1 Las colecciones en .Net Framework

Net Framework 4.5.2 x Net Framework 4.6.1 ofrece clases especializadas para almacenamiento y recuperación de datos. En Visual existen dos formas de crear agrupamiento de información: la primera es mediante la creación de arreglo de objetos y la segunda mediante la creación de colecciones como ArrayList, Stack, Queue, HashTable y List.

La diferencia que existe entre un arreglo y una colección es que el arreglo contiene información de un mismo tipo con un tamaño fijo de elementos. Así, se puede mencionar que en un arreglo de números solo se puede almacenar números, mientras que las colecciones permiten un manejo de información variable. Por eso, son llamados dinámicos, ya que no tienen un tamaño específico. Estos aumentan o disminuyen de tamaño según la necesidad de la información.

Para Visual, una colección es una clase especializada que cuenta con una serie de métodos listos para la manipulación de la información contenida. Entonces, se llega a la conclusión que para un mejor manejo de la información se necesita los siguientes elementos:

- Seleccionar una clase especializada de colección.
- Usar una estructura de repetición que permita recorrer por la información almacenada en la colección.
- Una clase que contenga la información de los elementos a almacenar en la colección.
- Validar la información que contiene la colección.
- Un control ListView para mostrar toda la información de la colección.

2.2 Estructura foreach

La estructura repetitiva foreach será la encargada de recorrer por las colecciones, ya que su estructura así lo permite. Su misión es repetir un grupo de instrucciones por cada elemento de una determinada colección.

Se debe tener en cuenta que las sentencias especificadas en el ciclo foreach se ejecutarán para cada elemento encontrado en la colección, es decir, cuando ya no haya elementos, el ciclo foreach se detendrá. También existe la posibilidad de salir del ciclo foreach sin necesidad de esperar encontrar el último elemento. Para esto, se puede usar la instrucción break.

Formato:

```
foreach (tipoDatos elemento in colección)
{
    //Cuerpo del bucle;
}
```

EDITORIAL MACRO®

Donde:

- **tipoDatos:** Es el tipo de información que contiene la colección. Si la colección contiene información variable, entonces se debe especificar el nombre de la clase que contiene toda la especificación de los tipos de datos.
- **Elemento:** Es aquel elemento que permitirá evaluar, obtener o analizar los datos contenidos en la colección.
- **Colección:** Es el nombre del objeto de colección, el cual contiene la información que se necesita recorrer. Esta colección debe incluir un método llamado GetEnumerator que devuelva un tipo, que a su vez contiene la siguiente información:
  - Cuenta con una propiedad Current que devuelva el elemento actual de la colección.
  - Cuenta con un método MoveNext de tipo booleano, que permite incrementar el contador de elementos. Este devolverá true si detecta que aún quedan elementos en la colección.

Uso de la estructura foreach:

- Mostrando los productos registrados en la colección List.

```
//Objeto de la colección List
List<string> productos = new List<string>();

//Llenando de productos la colección
productos.Add("Impresora");
productos.Add("Mouse");
productos.Add("Teclado");
productos.Add("Parlantes");

//Mostrando los registros mediante un mensaje
foreach (string p in productos)
{
    MessageBox.Show(p);
}
```

El objeto de la clase List es llamado «productos». La idea es llenarlo de productos para luego imprimirlo. El método Add permite llenar de elementos la colección previamente creada. Para mostrar la información, se usa la estructura repetitiva foreach, la cual hace referencia al tipo de dato String, ya que solo se está almacenando información de productos y no sus demás elementos, como precio, stock u otros. La variable *p* representa a un elemento que permitirá tener la información contenida en la colección y finalmente se especifica la colección de elementos «productos», el cual representa a todos los productos registrados. No se debe olvidar que para que funcione correctamente la clase List necesita la librería using System.Collections.

- Mostrando los alumnos registrados en la colección ArrayList

```
//Objeto de la colección ArrayList
ArrayList alumnos = new ArrayList();

//Llenando de alumnos la colección
alumnos.Add("Lucy Pacheco Hernández");
alumnos.Add("Carla Gallegos Silva");
alumnos.Add("Heidi Rengifo Reátegui");

//Mostrando los registros mediante un mensaje
foreach (string a in alumnos)
{
   MessageBox.Show(a);
}
```

El objeto de la clase ArrayList es llamado «alumnos», el cual se llenará de información de alumnos con el método Add. Para mostrar la información, se usa la estructura repetitiva foreach que hace referencia al tipo de dato String, ya que solo se está almacenando información de los nombres de los alumnos. La variable *a* representa a un elemento que permitirá tener la información contenida en la colección de alumnos y, finalmente, se especifica la colección de elementos «alumnos», el cual representa a todos los nombres de los alumnos registrados.

- Mostrando los datos del libro y sus precios registrados en la colección HashTable:

```
//Objeto de la colección HashTable
Hashtable libros = new Hashtable();

//Llenando de libros la colección
libros.Add("Programación Orientada a Objetos",50);
libros.Add("Visual C# 2015",120);
libros.Add("Aplicaciones web con PHP",90);

//Mostrando los registros mediante un mensaje
foreach (string li in libros.Keys)
{
   MessageBox.Show(li);
   MessageBox.Show(libros[li].ToString());
}
```

Se creó un objeto de la clase HashTable llamado «libros», el cual permitirá almacenar información de los libros, así como sus respectivos precios. La propiedad Add se encargará de registrar dichos valores. Lo que se debe tener en cuenta es que para especificar el precio, se debe colocar una coma después del nombre del libro.

Para mostrar la información contenida, se usa la estructura foreach, allí se define como string a toda la información contenida en la colección y en la especificación se coloca libros.Keys para hacer referencia a todos los elementos de la colección. Finalmente, para imprimir el nombre del libro, solo se especifica el nombre asignado en el foreach, mientras que para el precio se usa la expresión libros[li].ToString().

2.3 Clases de colecciones

Las colecciones NET Framework 4.5.2 x Net Framework 4.6.1 tienen una característica especial entre ellos, que se basa en la numeración de su índice base, el tipo de elemento que registra y el valor de los elementos clave o llave.

Se definen sus clases por medio de los siguientes Namespace:

- System.Collections.
- System.Collections.Specialized.

2.3.1 Espacio de nombre System.Collection

El espacio de nombres System.Collections contiene clases que definen varias colecciones de objetos, como listas, colas, tablas hash y diccionarios.

Las clases que comúnmente utilizan son las siguientes:

Clase	Descripción
ArrayList	Representa una colección de elementos a un arreglo cuyo tamaño aumenta dinámicamente cuando es necesario por la aplicación.
HashTable	Representa una colección de pares de clave y valor que se organizan por código hash de la clave.
Queue	Representa una colección de objetos de tipo primero en entrar, primero en salir (FIFO); también conocidos como registro en cola.
Stack	Representa una colección simple no genérica de objetos últimos en entrar, primero en salir (LIFO); también conocido como registro en pila.

2.3.2 Espacio de nombre System.Collections.Specialized

El espacio de nombres System.Collections.Specialized proporciona clases de colección especializadas y con establecimiento inflexible de tipos, como colecciones de diccionarios híbridos, las cuales se usarán en este capítulo.

Las clases que comúnmente utilizan son las siguientes:

Clase	Descripción
HybridDictionary	Representa una colección de elementos, la cual usa ListDictionary mientras la colección es pequeña. A continuación, cambia a Hashtable cuando la colección aumenta por dicha ambigüedad se le llama colección híbrida.

2.4 Clase ArrayList

Es una clase que permite almacenar información de acuerdo a un índice numérico y su capacidad de almacenamiento es de acuerdo a cómo se agregan los elementos en el ArrayList.

Se debe considerar los siguientes aspectos al usar la colección ArrayList:

- Para tener acceso a los elementos, se utiliza un índice de tipo entero.
- El índice de la colección ArrayList siempre empieza en cero.
- Es posible registrar valores null como valor válido dentro del ArrayList.
- Es posible añadir valores duplicados, a menos que se valide dicho proceso.

Método constructor:

Método	Descripción
ArrayList()	Define una nueva instancia de la clase ArrayList e inicializa su tamaño con un valor predeterminado. Por ejemplo, si se tiene una colección de productos, se podría definir de la siguiente manera: `ArrayList aProductos = new ArrayList();`

Propiedades:

Propiedad	Descripción
Count	Devuelve el número total de elementos registrados en la colección ArrayList. Por ejemplo, determinar el tamaño total de elementos registrados en un colección de productos:

```
//Creando la colección
ArrayList aProductos = new ArrayList();

//Ingresando valores
aProductos.Add("Lavadora");
aProductos.Add("Refrigeradora");
aProductos.Add("Tostadora");
aProductos.Add("Extractora");

//Determinar el total de elementos de la colección
int total = aProductos.Count;
MessageBox.Show("El total de productos es: "+total.ToString());
```

Ítem	Devuelve el valor que se encuentra en el índice especificado. Por ejemplo, mostrar todos los productos registrados en la colección:

```
//Creando la colección
ArrayList aProductos = new ArrayList();

//Ingresando valores
aProductos.Add("Lavadora");
aProductos.Add("Refrigeradora");
aProductos.Add("Tostadora");
aProductos.Add("Extractora");

//Mostrar los productos
for (int i = 0; i < aProductos.Count; i++)
{
  MessageBox.Show(aProductos[i].ToString());
}
```

Métodos:

Propiedad	Descripción
Add	Método que permite agregar un nuevo elemento a la colección ArrayList. Se debe tener en cuenta que siempre se agregará al final de la colección. Por ejemplo, si se necesita agregar productos a una colección, se podría usar el siguiente código:

```
//Creando la colección
ArrayList aProductos = new ArrayList();

//Agregando valores
aProductos.Add("DVD");
aProductos.Add("BlueRay");
aProductos.Add("Televisor");
```

Clear	Método que permite eliminar todo el contenido de la colección ArrayList. Por ejemplo, si se necesita eliminar el contenido de productos, se puede usar el siguiente código:

```
aProductos.Clear();
```

Insert	Método que permite insertar un elemento en la colección en una posición determinada. Por ejemplo, se inserta un nuevo producto en la posición 1 de la colección.

```
aProductos.Insert(1,"Extractora");
```

Remove

Método que permite eliminar la aparición de un determinado objeto registrado en la colección ArrayList. Por ejemplo, si se cuenta con cierta cantidad de productos registrados y se necesita eliminar un determinado producto, entonces el código sería como sigue:

```
//Creando la colección
ArrayList aProductos = new ArrayList();

//Ingresando valores
aProductos.Add("Lavadora");
aProductos.Add("Refrigeradora");
aProductos.Add("Tostadora");
aProductos.Add("Extractora");

//Eliminando un determinado producto
foreach(string p in aProductos)
{
    if (p == "Tostadora")
    {
       aProductos.Remove(p);
       break;
    }
}

//Mostrar los productos
for (int i = 0; i < aProductos.Count; i++)
{
   MessageBox.Show(aProductos[i].ToString());
}
```

RemoveAt

Método que permite eliminar un elemento de la colección en una determinada posición según su índice.

```
//Creando la colección
ArrayList aProductos = new ArrayList();

//Ingresando valores
aProductos.Add("Lavadora");
aProductos.Add("Refrigeradora");
aProductos.Add("Tostadora");
aProductos.Add("Extractora");

//Eliminando un determinado producto
aProductos.RemoveAt(1);

//Mostrar los productos
for (int i = 0; i < aProductos.Count; i++)
{
  MessageBox.Show(aProductos[i].ToString());
}
```

Sort	Método que permite ordenar de forma ascendente los elementos de una colección, se debe tener en cuenta que, al ordenar los elementos, cambian de posición según el orden.
	`//Creando la colección`
	`ArrayList aProductos = new ArrayList();`
	
	`//Ingresando valores`
	`aProductos.Add("Lavadora");`
	`aProductos.Add("Refrigeradora");`
	`aProductos.Add("Tostadora");`
	`aProductos.Add("Extractora");`
	
	`//Ordenando en forma ascendente`
	`aProductos.Sort();`
	
	`//Mostrar los productos`
	`for (int i = 0; i < aProductos.Count; i++)`
	`{`
	`  MessageBox.Show(aProductos[i].ToString());`
	`}`

2.5 Clase List<>

Es una clase que permite almacenar una lista de objetos fuertemente tipado, siempre controlado desde un índice numérico y su capacidad de almacenamiento es de acuerdo a cómo se agregan los elementos en el List.

La librería necesaria para su uso es System.Collections.

Método constructor:

Método	Descripción
List<>()	Define una nueva instancia de la clase List e inicializa su tamaño con un valor predeterminado. Por ejemplo, si se tiene una colección de libros, se podría definir de la siguiente manera: `List<Libro> aLibros = new List<Libro>();`

EDITORIAL MACRO®

Propiedades:

Propiedad	Descripción

Count	Devuelve el número total de elementos registrados en la colección List. Por ejemplo, determinar el tamaño total de elementos registrados en una colección de libros.
	`//Creando la colección`
	`List<Libro> aLibros = new List<Libro>();`
	
	`//Ingresando valores`
	`aLibros.Add("Visual C# Fundamentos");`
	`aLibros.Add("Visual C# Orientada a Objetos");`
	`aLibros.Add("Visual ASP con C#");`
	
	`//Determinar el total de elementos de la colección`
	`int total = aLibros.Count;`
	`MessageBox.Show("El total de libros es: "+total.ToString());`

Item	Devuelve el valor que se encuentra en el índice especificado. Por ejemplo, mostrar todos los libros registrados en la colección.
	`//Creando la colección`
	`List<Libro> aLibros = new List<Libro>();`
	
	`//Ingresando valores`
	`aLibros.Add("Visual C# Fundamentos");`
	`aLibros.Add("Visual C# Orientada a Objetos");`
	`aLibros.Add("Visual ASP con C#");`
	
	`//Mostrar los productos`
	`int total = aLibros.Count;`
	`for (int i = 0; i <total; i++)`
	`{`
	`  MessageBox.Show(aLibros[i].ToString());`
	`}`

Métodos:

Método	Descripción

Add	Método que permite agregar un nuevo elemento a la colección List. Se debe tener en cuenta que siempre se agregará al final de la colección. Por ejemplo, si se necesita agregar libros a una colección, se podría usar el siguiente código:
	`//Creando la colección`
	`List<Libro> aLibros = new List<Libro>();`
	
	`//Ingresando valores`
	`aLibros.Add("Visual C# Fundamentos");`
	`aLibros.Add("Visual C# Orientada a Objetos");`
	`aLibros.Add("Visual ASP con C#");`

MACRO®

Clear	Método que permite eliminar todo el contenido de la colección List. Por ejemplo, si se necesita eliminar el contenido de libros, se puede usar el siguiente código: `aLibros.Clear();`
Insert	Método que permite insertar un elemento en la colección en una posición determinada. Por ejemplo, se insertará una nueva descripción de libro en la posición 2 de la colección. `aLibros.Insert(2,"PHP fundamentos");`

Remove	Método que permite eliminar la aparición de un determinado objeto registrado en la colección List. Por ejemplo, si se cuenta con cierta cantidad de libros registrados y se necesita eliminar un determinado libro, entonces el código sería como sigue:
	`//Creando la colección`
	`List<Libro> aLibros = new List<Libro>();`
	
	`//Ingresando valores`
	`aLibros.Add("Visual C# Fundamentos");`
	`aLibros.Add("Visual C# Orientada a Objetos");`
	`aLibros.Add("Visual ASP con C#");`
	
	`//Eliminando un determinado libro`
	`foreach(string li in aLibros)`
	`{`
	`  if (li == "Visual C# Orientada a Objetos")`
	`  {`
	`  aLibros.Remove(li);`
	`  break;`
	`  }`
	`}`
	
	`//Mostrar los productos`
	`for (int i = 0; i < aLibros.Count; i++)`
	`{`
	`  MessageBox.Show(aLibros[i].ToString());`
	`}`

RemoveAt	Método que permite eliminar un elemento de la colección en una determinada posición según su índice. Por ejemplo, se eliminará el libro que se encuentra en la posición 2 de la colección.
	`//Creando la colección`
	`List<Libro> aLibros = new List<Libro>();`
	
	`//Ingresando valores`
	`aLibros.Add("Visual C# Fundamentos");`
	`aLibros.Add("Visual C# Orientada a Objetos");`
	`aLibros.Add("Visual ASP con C#");`
	
	`//Eliminando un determinado producto`
	`aLibros.RemoveAt(2);`
	
	`//Mostrar los Libros`
	`for (int i = 0; i < aLibros.Count; i++)`
	`{`
	`  MessageBox.Show(aLibros[i].ToString());`
	`}`

2.6 Clase Stack

Es una clase que permite almacenar un conjunto de información en el orden último en entrar primero en salir, también conocida como pila de elementos o simplemente LIFO (Last Int, First Out). La librería necesaria para su uso es System.Collections.Generic.

Método constructor:

Método	Descripción
Stack<>()	Define una nueva instancia de la clase Stack e inicializa su tamaño con un valor predeterminado. Por ejemplo, si se tiene una colección de productos, se podría definir de la siguiente manera: `Stack<Producto> aProductos = new List<Producto>();`

Propiedad

Propiedad	Descripción

Count	Devuelve el número total de elementos registrados en la colección Stack. Por ejemplo, determinar el tamaño total de productos registrados:
	`Stack<Producto> aProductos = new Stack<Producto>();`
	`Producto objP = new Producto();`
	
	`//Creando la colección`
	`objP.descripcion = "Lavadora";`
	`aProductos.Push(objP);`
	
	`objP.descripcion = "Refrigeradora";`
	`aProductos.Push(objP);`
	
	`//Determinar el total de elementos de la colección`
	`int total = aProductos.Count;`
	`MessageBox.Show("El total de productos es: "+total.ToString());`

Métodos:

Método	Descripción

Push	Método que permite agregar un nuevo elemento a la colección Stack. Se debe tener en cuenta que siempre se agregará al inicio de la colección. Por ejemplo, si se necesita agregar productos a una pila, se podría usar el siguiente código:
	`Stack<Producto> aProductos = new Stack<Producto>();`
	`Producto objP = new Producto();`
	
	`//Creando la colección`
	`objP.descripcion = "Lavadora";`
	`aProductos.Push(objP);`
	
	`objP.descripcion = "Refrigeradora";`
	`aProductos.Push(objP);`

Clear	Método que permite eliminar todo el contenido de la colección Stack. Por ejemplo, si se necesita eliminar el contenido de la colección de productos, se puede usar el siguiente código: `aProductos.Clear();`

Peek	Método que devuelve el valor situado al principio de la pila. La forma de devolución es de lectura, ya que dicho valor aún no se eliminará.:
	`//Objeto de la clase Stack`
	`Stack<Producto> aProductos = new Stack<Producto>();`
	`Producto objP = new Producto();`
	
	`//Agregando elementos a la colección`
	`objP.descripcion = "Lavadora";`
	`aProductos.Push(objP);`
	
	`objP.descripcion = "Refrigeradora";`
	`aProductos.Push(objP);`
	
	`//Mostrando el producto de la pila`
	`MessageBox.Show("El producto de la pila es: "+aProductos.Peek().descripcion);`

Pop	Método que permite eliminar la aparición de un determinado objeto ubicado al principio de la colección Stack. Por ejemplo, si se cuenta con cierta cantidad de productos registrados y se necesita eliminar un elemento de la pila, entonces el código sería como sigue:
	`//Objeto de la clase Stack`
	`Stack<Producto> aProductos = new Stack<Producto>();`
	`Producto objP = new Producto();`
	
	`//Agregando elementos a la colección`
	`objP.descripcion = "Lavadora";`
	`aProductos.Push(objP);`
	
	`objP.descripcion = "Refrigeradora";`
	`aProductos.Push(objP);`
	
	`//Eliminando el producto de la pila`
	`aProductos.Pop();`
	
	`//Mostrando el producto de la pila`
	`MessageBox.Show("El producto de la pila es: " + aProductos.Peek().descripcion);`

2.7 Clase Queue

Es una clase que permite almacenar un conjunto de información en el orden primero en entrar, primero en salir, también conocida como cola de elementos o simplemente FIFO (First Last Int, First Out). La librería necesaria para su uso es System.Collections.

Método constructor:

Propiedad	Descripción
Queue<>()	Define una nueva instancia de la clase Queue e inicializa su tamaño con un valor predeterminado. Por ejemplo, si se tiene una colección de participantes a un evento, se podría definir de la siguiente manera: `Queue<Participante> aParticipantes = new Queue<Participante>();`

Propiedad:

Propiedad	Descripción
Count	Devuelve el número total de elementos registrados en la colección Queue. Por ejemplo, determinar el tamaño total de participantes registrados:

```
//Objeto de la clase Queue
Queue<Participante> aParticipantes = new Queue<Participante>();
Participante objP = new Participante();

//Agregando elementos a la colección
objP.nombres = "Maria Zamora Mejia";
aParticipantes.Enqueue(objP);

objP.nombres = "Heidi Rengifo Reategui";
aParticipantes.Enqueue(objP);

//Determinar el total de elementos de la colección
int total = aParticipantes.Count;
MessageBox.Show("El total de participantes es: "+total.ToString());
```

Métodos:

Propiedad	Descripción
Enqueue	Método que permite agregar un nuevo elemento a la colección Queue. Se debe tener en cuenta que siempre se agregará al final de la colección. Por ejemplo, si se necesita agregar participantes a una cola, se podría usar el siguiente código:

```
//Objeto de la clase Queue
Queue<Participante> aParticipantes = new Queue<Participante>();
Participante objP = new Participante();

//Agregando elementos a la colección
objP.nombres = "Maria Zamora Mejia";
aParticipantes.Enqueue(objP);

objP.nombres = "Heidi Rengifo Reátegui";
aParticipantes.Enqueue(objP);
```

MACRO®

Clear	Método que permite eliminar todo el contenido de la colección Queue. Por ejemplo, si se necesita eliminar el contenido de la colección de participantes, se puede usar el siguiente código: `aParticipantes.Clear();`

Peek	Método que devuelve el valor situado al principio de la cola. La forma de devolución es de lectura, ya que dicho valor no se eliminará.
	`//Objeto de la clase Queue`
	`Queue<Participante> aParticipantes = new Queue<Participante>();`
	`Participante objP = new Participante();`
	
	`//Agregando elementos a la colección`
	`objP.nombres = "Maria Zamora Mejia";`
	`aParticipantes.Enqueue(objP);`
	
	`objP.nombres = "Heidi Rengifo Reátegui";`
	`aParticipantes.Enqueue(objP);`
	
	`//Mostrando el participante de la cola`
	`MessageBox.Show("El participante de turno es:"+aParticipantes.`
	`Peek().nombres);`

Dequeue	Método que permite eliminar la aparición de un determinado objeto ubicado al principio de la colección Queue. Por ejemplo, si se cuenta con cierta cantidad de participantes registrados y se necesita despachar a un participante de la cola, entonces el código sería como sigue:
	`//Objeto de la clase Queue`
	`Queue<Participante> aParticipantes = new Queue<Participante>();`
	`Participante objP = new Participante();`
	
	`//Agregando elementos a la colección`
	`objP.nombres = "Maria Zamora Mejia";`
	`aParticipantes.Enqueue(objP);`
	
	`objP.nombres = "Heidi Rengifo Reátegui";`
	`aParticipantes.Enqueue(objP);`
	
	`//Eliminando el participante de la cola`
	`aParticipantes.Dequeue();`
	
	`//Mostrando el participante de la cola`
	`MessageBox.Show("El participante de turno es:" +aParticipantes.`
	`Peek().nombres);`

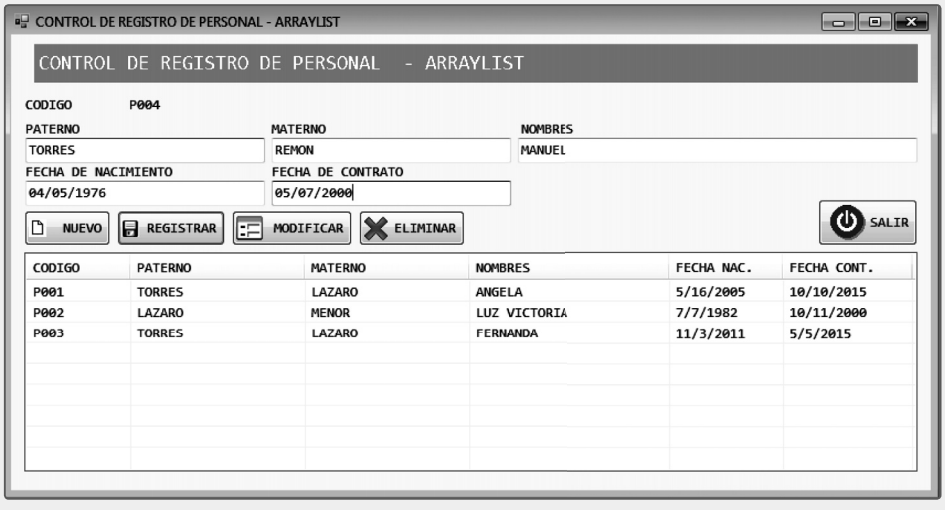
Caso desarrollado 1 : ArrayList - Control de registro de personal

Implementar una aplicación que permita registrar los datos de un nuevo personal como parte del Sistema de Control de Personal. Los datos que se necesitan registrar son nombres, apellidos, fecha de nacimiento y fecha de contratación. Usar la clase ArrayList.

Se tiene en cuenta lo siguiente:

- El código del personal a registrar debe ser automático y correlativo, de tal forma que tenga el siguiente formato «P001», donde la letra *p* es constante y el número uno representa el total de personal registrado.
- Implementar botones para el proceso de mantenimiento de registros de personal como nuevo, registrar, modificar y eliminar.
  - **Botón Nuevo:** Tiene la misión de limpiar los controles y mostrar el nuevo código de personal.
  - **Botón Registrar:** Tiene la misión de registrar los nuevos valores del personal al objeto ArrayList, que luego serán mostrados en un cuadro de lista.
  - **Botón Modificar:** Tiene la misión de actualizar los datos del empleado. Para esto, los controles deben mostrar la información del personal obtenida desde el control ListView. Para lograr mostrar la información, se debe presionar doble clic sobre el código del empleado mostrado en la lista.
  - **Botón Eliminar:** Tiene la misión de eliminar un determinado registro. Para lograr eliminar un personal, se debe mostrar toda su información en los controles. Para esto, se deberá presionar doble clic sobre el código del personal mostrado en el control ListView.

Formulario propuesto:



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjControlPersonal.
2. Agregue al proyecto la clase Personal e implemente el siguiente código:

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

namespace pjControlPersonal
{
    public class Personal
    {
        public string codigo { get; set; }
        public string paterno { get; set; }
        public string materno { get; set; }
        public string nombres { get; set; }
        public DateTime fechaNac { get; set; }
        public DateTime fechaCon { get; set; }
    }
}
```

- **Clase Personal**

  La clase personal declara los atributos propios del empleado para que puedan ser enviados al objeto ArrayList.

- **Clase frmControlPersonal**

```
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;

//Libreria para colecciones
using System.Collections;

namespace pjControlPersonal
{
    public partial class frmControlPersonal : Form
    {
        //Objeto de la clase ArrayList
        ArrayList aPersonal = new ArrayList();
```

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

        private void frmControlPersonal_Load(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo();
        }

        private void btnNuevo_Click(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            try
            {
                //Objeto de la clase Personal
                Personal objP = new Personal();

                //Enviando información a la clase
                objP.codigo = lblCodigo.Text;
                objP.paterno = txtPaterno.Text;
                objP.materno = txtMaterno.Text;
                objP.nombres = txtNombres.Text;
                objP.fechaNac = DateTime.Parse(txtFechaNac.Text);
                objP.fechaCon = DateTime.Parse(txtFechaCon.Text);

                //Agregando al ArrayList
                aPersonal.Add(objP);

                listado();
                limpiarControles();
                lblCodigo.Text = generaCodigo();
            }
            catch (Exception)
            {
                MessageBox.Show("Ocurrió un error al registrar");
            }
        }

        private void btnModificar_Click(object sender, EventArgs e)
        {
            try
            {
                foreach (Personal p in aPersonal)
                {
                    if (p.codigo == lblCodigo.Text)
                    {
```

```
                    p.paterno = txtPaterno.Text;
                    p.materno = txtMaterno.Text;
                    p.nombres = txtNombres.Text;
                    p.fechaNac = DateTime.Parse(txtFechaNac.Text);
                    p.fechaCon = DateTime.Parse(txtFechaCon.Text);
                    break;
                }
            }
            listado();
            generaCodigo();
            limpiarControles();
        }
        catch (Exception)
        {
            MessageBox.Show("Ocurrió un error al modificar");
        }
    }

    private void btnEliminar_Click(object sender, EventArgs e)
    {
        try
        {
            foreach (Personal p in aPersonal)
            {
                if (p.codigo == lblCodigo.Text)
                {
                    aPersonal.Remove(p);
                    break;
                }
            }
            listado();
        }
        catch (Exception)
        {
            MessageBox.Show("Ocurrió un error al eliminar");
        }

    }

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

    private void lvPersonal_MouseDoubleClick(object sender, MouseEventArgs e)
    {
        //Obteniendo la posición del elemento seleccionado
        ListViewItem elemento = lvPersonal.GetItemAt(e.X, e.Y);
```

```
            //Verificando que halla elemento seleccionado
            if (elemento != null)
            {
                lblCodigo.Text = elemento.Text;
                txtPaterno.Text = elemento.SubItems[1].Text;
                txtMaterno.Text = elemento.SubItems[2].Text;
                txtNombres.Text = elemento.SubItems[3].Text;
                txtFechaNac.Text = elemento.SubItems[4].Text;
                txtFechaCon.Text = elemento.SubItems[5].Text;
            }
        }

        //Métodos personalizados
        public string generaCodigo()
        {
            string cod="";
            if (aPersonal.Count < 1)
                return "P001";
            else
                foreach (Personal p in aPersonal)
                    cod=p.codigo.ToString();
           return "P"+(int.Parse(cod.Substring(1, 3))+1).ToString("000");
        }
        void listado()

        {
            lvPersonal.Items.Clear();
            foreach (Personal p in aPersonal)
            {
                ListViewItem fila = new ListViewItem(p.codigo);
                fila.SubItems.Add(p.paterno);
                fila.SubItems.Add(p.materno);
                fila.SubItems.Add(p.nombres);
                fila.SubItems.Add(p.fechaNac.ToShortDateString());
                fila.SubItems.Add(p.fechaCon.ToShortDateString());
                lvPersonal.Items.Add(fila);
            }
        }

        void limpiarControles()
        {
            txtPaterno.Clear();
            txtMaterno.Clear();
            txtNombres.Clear();
            txtFechaNac.Clear();
            txtFechaCon.Clear();
            txtPaterno.Focus();
        }
    }
}
```

El formulario control de personal iniciará por declarar el objeto de tipo ArrayList llamado aPersonal, el cual contendrá con todos los registros de los empleados. Se debe declarar de forma global, ya que muchos métodos usarán sus valores.

En el evento load del formulario se hace referencia al método generaCodigo el cual se asigna a lblCodigo. Este método genera el nuevo código del personal como lo precisa el caso, mientras que el botón nuevo permitirá generar nuevamente un código nuevo del personal.

El botón **Registrar** creará un objeto de la clase personal para poder enviar toda la información correspondiente a un personal, que luego será enviada al objeto ArrayList mediante la sentencia aPersonal.add(objP), donde objP representa a todos los valores de un determinado personal y aPersonal representa el objeto de la colección ArrayList.

El botón **Modificar** recorrerá por todos los registros de la colección y se compara el código de la lista (p.codigo) con el código a modificar, el cual se encuentra en el control lblCodigo. Si ambos son iguales, entonces se procesa la actualización. No olvidar colocar la sentencia break dentro de la sentencia if, ya que una vez actualizado el registro se debe finalizar la sentencia foreach, ya que no pueden haber dos códigos de personal iguales. No olvidar que por cada cambio en los registros se debe mostrar la actualización en el control ListView, para eso se invoca al método listado.

En botón **Eliminar** recorrerá por todos los registros de la colección y se compara el código de la lista con el código a eliminar, el cual se encuentra en el control lblCodigo. Para eliminar dicho registro, se usa el método Remove(p), donde *p* representa al registro encontrado. Una vez eliminado el registro se tiene que finalizar el ciclo foerach usando la sentencia break.

Para devolver la información desde el control ListView a los controles del formulario, el usuario deberá presionar doble clic sobre el código del personal. Para lograr esto, se debe seleccionar el control Listview y en la ventana de propiedades seleccionar Eventos. Luego, se debe buscar el evento MouseDobleClick y presionando doble clic sobre dicho evento se debe colocar el siguiente código `ListViewItem elemento = lvPersonal.GetItemAt(e.X, e.Y)`. La variable «elemento» contiene el código, del cual se presionó doble clic. Es a partir de aquí que se puede obtener los demás valores del registro usando la sentencia `elemento.SubItems[1]`.Text, donde 1 representa la columna de donde se obtendrá el valor, ya que la posición cero es el código del personal y uno representa a su apellido paterno.

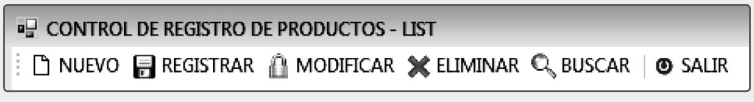
El método generaCodigo tiene la misión de componer el código del personal a registrar. Para lo cual, primero se compara que el total de personal registrado sea menor a uno, eso quiere decir que no hay personal registrado y que el código generado debe ir automáticamente con el formato P001, y que en caso haya registro de personal, se obtendrá el último código para aumentar en uno y generar el nuevo código. Para esto, se usa un foreach que recorra todos los registros y se captura el último código en la última vuelta del ciclo foreach. Luego, se obtiene los tres caracteres de la derecha con la función Substring(1,3), donde uno representa la posición del texto y tres la cantidad de caracteres que se necesita obtener desde la posición uno y luego esto se compone en un nuevo código.

Caso desarrollado 2 : List - Control de registro de productos

Implementar una aplicación que permita registrar los datos de un nuevo producto como parte del sistema de control de ventas. Los datos que se necesitan registrar son el código, descripción, categoría, precio, stock actual y stock mínimo de un determinado producto. Usar la clase List.

Se tiene en cuenta lo siguiente:

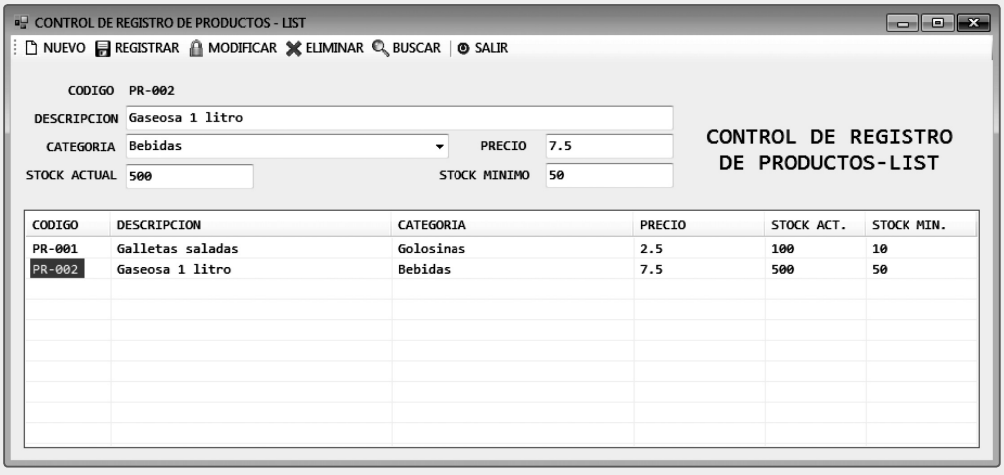
- Implementar la clase Producto, la cual tenga como atributos el código, la descripción, la el categoría, el precio, el stock actual y el stock mínimo de un determinado producto.
- Usar el control ToolStrip para los botones de mantenimiento, de tal forma que se muestre de la siguiente manera:



  Asigne los siguientes nombres: tsNuevo, tsRegistrar, tsModificar, tsEliminar, tsBuscar y tsSalir.

- El código del producto debe ser generado automáticamente con el siguiente formato PR-001, donde PR es un valor constante y el número 1 es consecutivo según el número de productos registrados.

GUI propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjControlProductos.
2. Agregue al proyecto la clase Producto e implemente el siguiente código:

- **Clase Producto**

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

namespace pjControlProductos
{
    public class Producto
    {
        public string codigo { get; set; }
        public string descripcion { get; set; }
        public string categoria { get; set; }
        public double precio { get; set; }
        public int stockActual { get; set; }
        public int stockMinimo { get; set; }
    }
}
```

La clase producto declara los atributos que permiten describir el producto, como su código, descripción, categoría, precio, stock actual y stock mínimo.

- **Clase frmControlProductos**

```
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;
using System.Collections;
using Microsoft.VisualBasic;

namespace pjControlProductos
{
    public partial class frmControlProductos : Form
    {
        //Objeto de la clase List
        List<Producto> aProductos = new List<Producto>();

        public frmControlProductos()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo();
        }

        private void tsNuevo_Click(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo();
        }

        private void tsRegistrar_Click(object sender, EventArgs e)
        {
                //Objeto de la clase Producto
                Producto objP = new Producto();

                //Enviando los valores del producto
                objP.codigo = lblCodigo.Text;
                objP.descripcion = txtDescripcion.Text;
                objP.categoria = cboCategoria.Text;
                objP.precio = double.Parse(txtPrecio.Text);
                objP.stockActual = int.Parse(txtStockActual.Text);
                objP.stockMinimo = int.Parse(txtStockMinimo.Text);

                //Validando la existencia del producto
                foreach (Producto p in aProductos)
                {
```

```
                if (p.descripcion == objP.descripcion)
                {
                MessageBox.Show("Producto ya se encuentra registrado");
                    return;
                }
            }

            //Agregando el producto a la colección List
            aProductos.Add(objP);

            listado();
            limpiarControles();
            lblCodigo.Text = generaCodigo();
        }

        private void tsModificar_Click(object sender, EventArgs e)
        {
            try
            {
                foreach (Producto p in aProductos)
                {
                    if (p.codigo == lblCodigo.Text)
                    {
                        p.descripcion = txtDescripcion.Text;
                        p.categoria = cboCategoria.Text;
                        p.precio = double.Parse(txtPrecio.Text);
                        p.stockActual = int.Parse(txtStockActual.Text);
                        p.stockMinimo = int.Parse(txtStockMinimo.Text);
                        break;
                    }
                }
                listado();
                generaCodigo();
                limpiarControles();
            }
            catch (Exception)
            {
                MessageBox.Show("Ocurrió un error al modificar");
            }

        }

        private void tsEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                foreach (Producto p in aProductos)
                {
                    if (p.codigo == lblCodigo.Text)
                    {
                        aProductos.Remove(p);
```

```
                    break;
                }
            }
            listado();
        }
        catch (Exception)
        {
            MessageBox.Show("Ocurrió un error al eliminar");
        }

    }

    private void tsBuscar_Click(object sender, EventArgs e)
    {
       string codigo = Interaction.InputBox("Ingrese codigo de producto");

        foreach (Producto p in aProductos)
        {
            if (p.codigo == codigo)
            {
                txtDescripcion.Text = p.descripcion;
                cboCategoria.Text = p.categoria;
                txtPrecio.Text = p.precio.ToString();
                txtStockActual.Text = p.stockActual.ToString();
                txtStockMinimo.Text = p.stockMinimo.ToString();
                break;
            }
        }
    }

    private void tsSalir_Click(object sender, EventArgs e)
    {
      DialogResult r = MessageBox.Show("Esta seguro de salir ? ", "Salir",
                       MessageBoxButtons.YesNo, MessageBoxIcon.Error);
        if (r == DialogResult.Yes)

            this.Close();
    }

    //Metodos personalizados
    public string generaCodigo()
    {
        string cod = "";
        if (aProductos.Count < 1)
            return "PR-001";
        else
            foreach (Producto p in aProductos)
                cod = p.codigo.ToString();
     return "PR-" + (int.Parse(cod.Substring(3, 3)) + 1).ToString("000");
    }
```

```
        void listado()
        {
            lvProductos.Items.Clear();
            foreach (Producto p in aProductos)
            {
                ListViewItem fila = new ListViewItem(p.codigo);
                fila.SubItems.Add(p.descripcion);
                fila.SubItems.Add(p.categoria);
                fila.SubItems.Add(p.precio.ToString());
                fila.SubItems.Add(p.stockActual.ToString());
                fila.SubItems.Add(p.stockMinimo.ToString());
                lvProductos.Items.Add(fila);
            }
        }

        void limpiarControles()
        {
            txtDescripcion.Clear();
            cboCategoria.SelectedIndex = -1;
            txtPrecio.Clear();
            txtStockActual.Clear();
            txtStockMinimo.Clear();
        }

       private void lvProductos_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            //Obteniendo la posición del elemento seleccionado
            ListViewItem elemento = lvProductos.GetItemAt(e.X, e.Y);

            //Verificando que halla elemento seleccionado
            if (elemento != null)
            {
                lblCodigo.Text = elemento.Text;
                txtDescripcion.Text = elemento.SubItems[1].Text;
                cboCategoria.Text = elemento.SubItems[2].Text;
                txtPrecio.Text = elemento.SubItems[3].Text;
                txtStockActual.Text = elemento.SubItems[4].Text;
                txtStockMinimo.Text = elemento.SubItems[5].Text;
            }
        }
    }
}
```

Se inicia declarando el objeto de la colección List llamada aProductos, el cual registrará todos los productos en la aplicación. Este objeto está declarado en la sección global, ya que existen varios métodos que usarán la colección de productos.

En el evento load se asigna el método generaCodigo al control lblCodigo, el cual mostrará el nuevo código generado. El método generaCodigo devuelve el código nuevo a partir del total de productos registrados. Si el total es inferior a uno, entonces el código inicial será PR-001 por tratarse del primer producto, caso contrario se debe obtener el último código registrado y obtener su parte numérica del código y aumentarlo en uno para generar el nuevo código. Para obtener solo la parte numérica del código, se usa el método Substring(3,3), el 3 de la izquierda indica la posición de donde se quiere capturar y el segundo 3 indica la cantidad de caracteres a capturar. Luego de esto, se pasa a número para aumentar en uno su valor.

El ToolStrip **nuevo** permite generar un nuevo código de producto. El botón registrar crea un objeto de la clase Producto y allí se enviarán todos los datos del producto registrado. Asimismo, valida la existencia del código para no repetir su ingreso. Para esto, se usa el control foreach para que recorra por todos los productos y comparar si el código a registrar ya que se encuentra en la colección; si esto ocurre se enviar el mensaje «Producto ya se encuentra registrado». Después de la validación enviamos el producto a la colección aProductos.

El ToolStrip **modificar** permite actualizar los datos del producto en la colección. Para esto, se usa la estructura foreach para recorrer por los productos y buscar el producto a modificar. Una vez encontrado dicho producto se envían todos los valores de los controles al objeto «p», que representan a la colección aProductos en el foreach. Para mostrar el resultado de la actualización, no se debe olvidar de invocar al método listado.

El ToolStrip **eliminar** permite eliminar el producto de la colección. Para esto, se debe recorrer por todos los registros usando la estructura foreach. Una vez encontrado se usa el método Remove(p) donde *p* representa el registro encontrado en la colección. No olvidarse de colocar la sentencia break para no seguir buscando el producto, ya que este se encuentra eliminado.

El ToolStrip **buscar** solicita el código del producto a buscar para poder mostrar toda su información en los controles. La solicitud del código se hace con la sentencia Interaction.InputBox. Para que esta función sea aceptada, se necesita la librería Using Microsoft.VisualBasic.

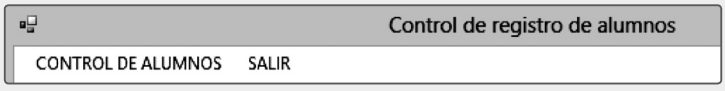
Para devolver la información desde el control ListView a los controles del formulario, el usuario deberá presionar doble clic sobre el código del producto. Para lograr esto, se debe seleccionar el control Listview y en la ventana de propiedades seleccionar Eventos. Luego, se debe buscar el evento MouseDobleClick y presionando doble clic sobre dicho evento colocar el siguiente código: `ListViewItem elemento = lvProductos.GetItemAt(e.X, e.Y).` La variable «elemento» contiene el código del cual se presionó doble clic, es a partir de aquí que se puede obtener los demás valores del registro usando la sentencia `elemento.SubItems[1].Text`; donde 1 representa la columna de donde obtendrá el valor, ya que la posición cero es el código del producto y uno representa a la descripción del producto.

Caso desarrollado 3 : Stack - Control de registro de alumnos

Implementar una aplicación que permita registrar los datos de los alumnos en una institución educativa, los cuales cuentan con código, apellido paterno, materno, nombres, categoría y fecha de nacimiento. Estos datos deberán ser registrados en la clase Stack.

Se tiene en cuenta lo siguiente:

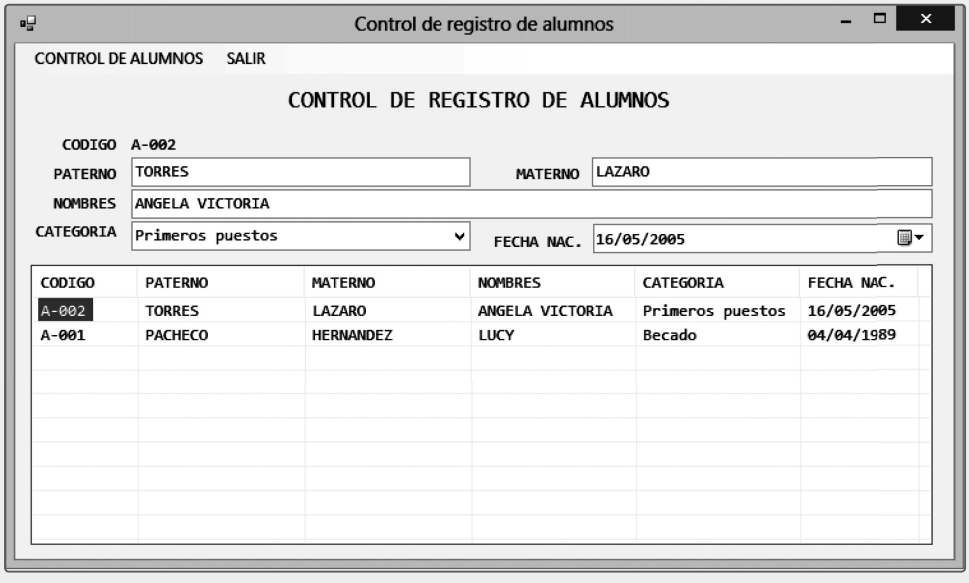
- Implementar la clase Alumno, la cual tenga como atributos el código, paterno, materno, nombres, categoría y fecha de nacimiento del alumno.
- Usar el control MenuStrip para los botones de mantenimiento, de tal forma que se muestre de la siguiente manera:



Agregar las opciones del menú como nuevo, grabar, modificar y eliminar.

- El código del alumno debe ser generado automáticamente con el siguiente formato A-001, donde «A-» es un valor constante y el numero 1 es consecutivo según el número de alumnos registrados.
- Las categorías de los alumnos son Primeros puestos y Becado, los cuales deben estar precargados en el control cuadro combinado.

GUI Propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjControl_Alumnos.
2. Agregue al proyecto la clase Alumno e implemente el siguiente código:

- **Clase Alumno**

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

namespace pjControl_Alumnos
{
    public class Alumno
    {
        public string codigo { get; set; }
        public string paterno { get; set; }
        public string materno { get; set; }
        public string nombres { get; set; }
        public string categoria { get; set; }
        public DateTime fechaNac { get; set; }
    }
}
```

La clase Alumno declara los atributos necesarios para el registro de un alumno, como el código, paterno, materno, nombres, categoría y fecha de nacimiento.

- **Clase frmControlAlumnos**

```
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 pjControl_Alumnos
{
    public partial class frmControlAlumnos : Form
    {
        //Objeto de la clase Stack
        Stack<Alumno> aAlumnos = new Stack<Alumno>();
```

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

        private void Form1_Load(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo();
        }

       private void NUEVOToolStripMenuItem_Click(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo();
        }

       private void GRABARToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Objeto de la clase Alumno
            Alumno objA = new Alumno();

            //Enviando los valores del alumno
            objA.codigo = lblCodigo.Text;
            objA.paterno= txtPaterno.Text;
            objA.materno = txtMaterno.Text;
            objA.nombres = txtNombres.Text;
            objA.categoria = cboCategoria.Text;
            objA.fechaNac = dtFechaNac.Value;

            //Validando la existencia del alumno
            foreach (Alumno a in aAlumnos)
            {
            string nombreAlumno = objA.nombres+" "+objA.paterno+" "+objA.materno;

              if (a.nombres+" "+a.paterno+" "+a.materno  == nombreAlumno)
                {
                    MessageBox.Show("Alumno ya se encuentra registrado");
                     return;
                }
            }

            //Agregando el alumno a la colección Queue
            aAlumnos.Push(objA);

            listado();
            limpiarControles();
            lblCodigo.Text = generaCodigo();
        }
```

```
private void MODIFICARToolStripMenuItem_Click(object sender, EventArgs e)
 {
     try
     {
         foreach (Alumno a in aAlumnos)
         {
             if (a.codigo == lblCodigo.Text)
             {
                 a.paterno= txtPaterno.Text;
                 a.materno = txtMaterno.Text;
                 a.nombres = txtNombres.Text;
                 a.categoria = cboCategoria.Text;
                 a.fechaNac = dtFechaNac.Value;
                 break;
             }
         }
         listado();
         generaCodigo();
         limpiarControles();
     }
     catch (Exception)
     {
         MessageBox.Show("Ocurrió un error al modificar");
     }
 }

private void ELIMINARToolStripMenuItem_Click(object sender, EventArgs e)
 {
     try
     {
         aAlumnos.Pop();
         listado();
         lblCodigo.Text = generaCodigo();
     }
     catch (Exception)
     {
         MessageBox.Show("Ocurrió un error al eliminar");
     }
 }

 //Metodos personalizados
 public string generaCodigo()
 {
     string cod = "";
     if (aAlumnos.Count < 1)
         return "A-001";
     else
```

```
                cod = aAlumnos.Peek().codigo.ToString();
            return "A-" + (int.Parse(cod.Substring(2, 3)) + 1).ToString("000");
        }

        void listado()
        {
            lvAlumnos.Items.Clear();
            foreach (Alumno a in aAlumnos)
            {
                ListViewItem fila = new ListViewItem(a.codigo);
                fila.SubItems.Add(a.paterno);
                fila.SubItems.Add(a.materno);
                fila.SubItems.Add(a.nombres);
                fila.SubItems.Add(a.categoria);
                fila.SubItems.Add(a.fechaNac.ToShortDateString());
                lvAlumnos.Items.Add(fila);
            }
        }

        void limpiarControles()
        {
            txtPaterno.Clear();
            txtMaterno.Clear();
            txtNombres.Clear();
            cboCategoria.SelectedIndex=-1;
            txtPaterno.Focus();
        }

      private void lvAlumnos_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            //Obteniendo la posición del elemento seleccionado
            ListViewItem elemento = lvAlumnos.GetItemAt(e.X, e.Y);

            //Verificando que halla elemento seleccionado
            if (elemento != null)
            {
                lblCodigo.Text = elemento.Text;
                txtPaterno.Text = elemento.SubItems[1].Text;
                txtMaterno.Text = elemento.SubItems[2].Text;
                txtNombres.Text = elemento.SubItems[3].Text;
                cboCategoria.Text = elemento.SubItems[4].Text;
              dtFechaNac.Value = DateTime.Parse(elemento.SubItems[5].Text);
            }

        }
    }
}
```

Al iniciar el formulario de control de alumnos, se declara en forma global el objeto de colección aAlumnos de la clase Stack con la siguiente sentencia `Stack<Alumno> aAlumnos = new Stack<Alumno>()`, donde Alumno es la referencia a la clase de donde se obtendrán los datos.

En el evento load del formulario se asigna el método generaCodigo al control lblCodigo. Este método genera un nuevo código de alumno, dentro del método se verifica si existe registro de alumnos, si no fuera así se asigna directamente el código A-001, caso contrario se captura el último código registrado. Para esto, se usa la función Peek que permite obtener el último valor registrado en la colección.

En la opción de menú grabar se crea un objeto de la clase Alumno para poder enviar los valores del alumno a registrar. Asimismo, se valida la existencia del alumno por medio del nombre completo del alumno. Para esto, se recorre todos los registros de los alumnos mediante la sentencia foreach. En la variable nombreAlumno se concatena el nombre y los apellidos del alumno para compararlo con los datos registrados en la colección. Si esto ocurre se le envía el mensaje «Alumno ya se encuentra registrado» y se aplica la sentencia return, el cual permite finalizar todo el proceso, es decir, no solo lo expulsa del ciclo de repeticiones, sino también del mismo proceso de grabar. Si después de la evaluación del nombre completo de alumno no se encontrara en la lista, entonces se agregará a la colección. En este caso se usa el método Push por tratarse de una colección Stack.

En la opción de menú modificar se inicia recorriendo los registros de la colección de alumnos. Si el código obtenido de la colección es igual al mostrado en el control lblCodigo, entonces se procederá a la modificación de los valores del registro actual. Tener en cuenta que se debe colocar la sentencia break dentro de la actualización, ya que se debe finalizar el ciclo de repeticiones.

La opción del menú eliminar permite sacar un elemento de la colección por medio del método Pop, lo que cumple con la regla último el llegar primero en salir.

MACRO

Caso desarrollado 4 : Queue - Control de registro de libros

Implementar una aplicación que permita registrar los datos de los libros en una biblioteca, los cuales cuentan con código, título, autor, año de edición y costo del libro.

Estos datos deberán ser registrados en la clase Queue.

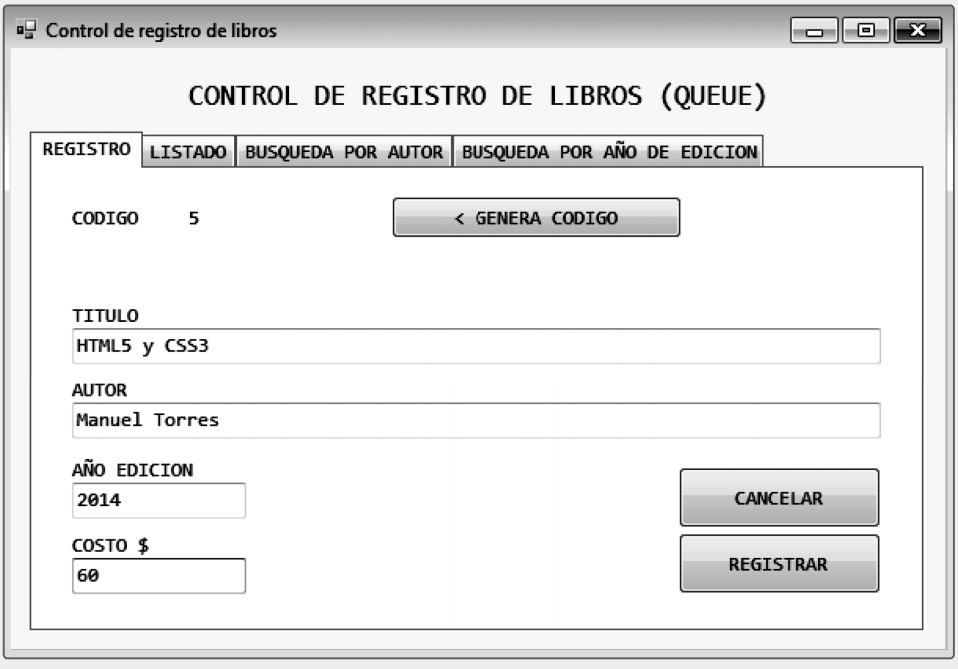
Se tiene en cuenta lo siguiente:

- Implementar la clase Libro, la cual tenga como atributos el código, título, autor, año de edición y costo del libro.
- Usar el control TabControl para los botones de mantenimiento, de tal forma que se muestre de la siguiente manera:

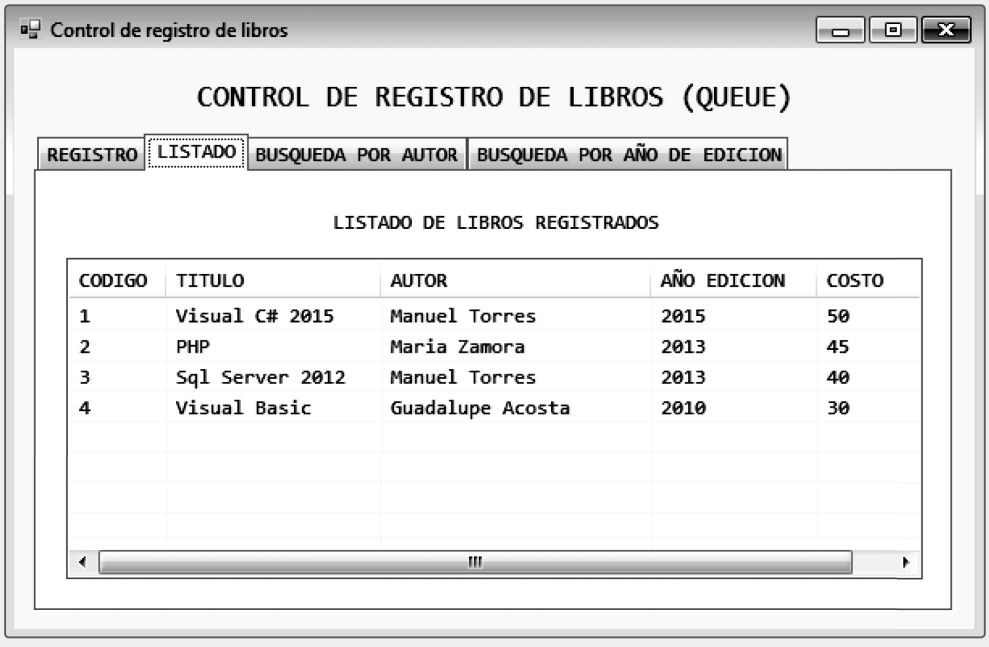
Agregar las fichas registro, listado, búsqueda por autor y búsqueda por año de edición.

- El contenido de las fichas varía según el título del mismo de la siguiente manera:

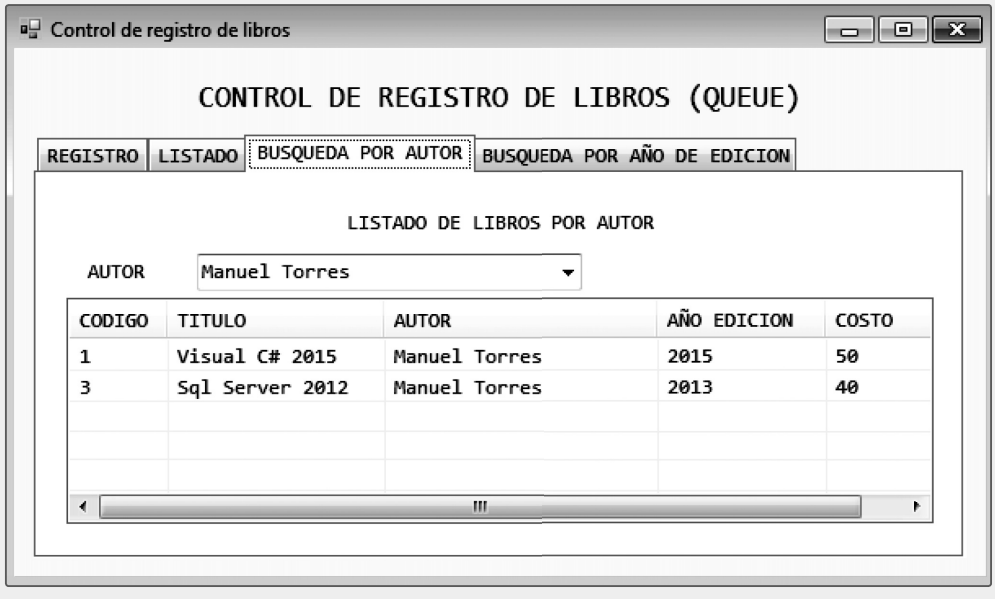
La ficha REGISTRO permite grabar los datos del nuevo en la colección Queue con el botón registrar. El botón GENERA CÓDIGO permite mostrar un nuevo código de libro. Hay que tener en cuenta que dicho código dependerá de la cantidad de libros registrados.



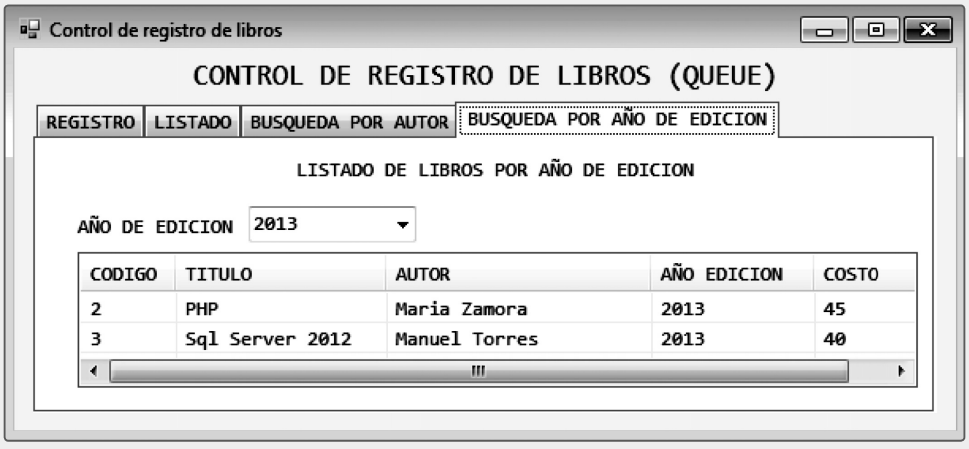
- La ficha LISTADO permite mostrar todos los registros de libros contenidos en la colección Queue, como se muestra en la siguiente ventana:



- La ficha BÚSQUEDA POR AUTOR muestra un cuadro combinado con todos los nombre de los autores registrados en la primera ficha, pero se debe tener en cuenta que no debe mostrar nombres de autores repetidos. Al seleccionar un autor del cuadro combinado, se deberá mostrar la lista de libros registrados con dicho autor.



- La ficha BÚSQUEDA POR AÑO DE EDICIÓN muestra un cuadro combinado con todos los años de edición registrados en la primera ficha, pero se debe tener en cuenta que no debe mostrar años repetidos. Al seleccionar un año del cuadro combinado, se deberá mostrar la lista de libros registrados con dicho año de edición.



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjControlLibros.
2. Agregue al proyecto la clase Libro e implemente el siguiente código:

- **Clase Libro:**

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

namespace pjControlLibros
{
    public class Libro
    {
        public int codigo { get; set; }
        public string titulo { get; set; }
        public string autor { get; set; }
        public int añoEdicion { get; set; }
        public double costo { get; set; }
    }
}
```

La clase Libro contiene los atributos pertenecientes al registro de un determinado libro, como código, título, autor, año de edición y costo.

- **Clase frmControlLibro:**

```
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 pjControlLibros
{
    public partial class frmControlLibros : Form
    {
        //Objeto de la clase Queue
        Queue<Libro> aLibros = new Queue<Libro>();

        public frmControlLibros()
        {
            InitializeComponent();
        }

        private void frmControlLibros_Load(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo().ToString();
        }

        private void btnGenera_Click(object sender, EventArgs e)
        {
            lblCodigo.Text = generaCodigo().ToString();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            //Objeto de la clase Libro
            Libro objL = new Libro();

            //Enviando los valores del libro
            objL.codigo = int.Parse(lblCodigo.Text);
            objL.titulo = txtTitulo.Text;
            objL.autor = txtAutor.Text;
            objL.añoEdicion = int.Parse(txtAño.Text);
            objL.costo = double.Parse(txtCosto.Text);

            //Validando la existencia del libro
            foreach (Libro li in aLibros)
            {
                if (li.titulo == objL.titulo)
                {
                    MessageBox.Show("Libro ya se encuentra registrado");
                    return;
                }
            }
```

```
        //Agregando el libro a la colección Queue
        aLibros.Enqueue(objL);

        listado();
        limpiarControles();
        lblCodigo.Text = generaCodigo().ToString();

        llenaAutores();
        llenaAños();
    }

    void llenaAutores()
    {
        var autores = (from Libro a in aLibros
                       select new { a.autor }).Distinct().ToList();

        cboAutor.DataSource = autores;
        cboAutor.DisplayMember = "autor";
    }

    void llenaAños()
    {
        var años = (from Libro a in aLibros
                    select new { a.añoEdicion }).Distinct().ToList();

        cboAño.DataSource = años;
        cboAño.DisplayMember = "añoEdicion";
    }

    //Metodos personalizados
    public int generaCodigo()
    {
        int cod = 0;
        if (aLibros.Count < 1)
            return 1;
        else
            foreach (Libro li in aLibros)
                cod = li.codigo;
        return cod+1;
    }

    void listado()
    {
        lvLibros.Items.Clear();
        foreach (Libro li in aLibros)
        {
           ListViewItem fila = new ListViewItem(li.codigo.ToString());
            fila.SubItems.Add(li.titulo);
            fila.SubItems.Add(li.autor);
            fila.SubItems.Add(li.añoEdicion.ToString());
            fila.SubItems.Add(li.costo.ToString());
```

```
                lvLibros.Items.Add(fila);
            }
        }

        void limpiarControles()
        {
            txtTitulo.Clear();
            txtAutor.Clear();
            txtAño.Clear();
            txtCosto.Clear();
            txtTitulo.Focus();
        }

        private void cboAutor_SelectedIndexChanged(object sender, EventArgs e)
        {
            lvLibrosAutor.Items.Clear();
            foreach (Libro li in aLibros)
            {
                if (li.autor == cboAutor.Text)
                {
                 ListViewItem fila = new ListViewItem(li.codigo.ToString());
                    fila.SubItems.Add(li.titulo);
                    fila.SubItems.Add(li.autor);
                    fila.SubItems.Add(li.añoEdicion.ToString());
                    fila.SubItems.Add(li.costo.ToString());
                    lvLibrosAutor.Items.Add(fila);
                }
            }
        }

        private void cboAño_SelectedIndexChanged(object sender, EventArgs e)
        {
            lvLibrosxAño.Items.Clear();
            foreach (Libro li in aLibros)
            {
                if (li.añoEdicion.ToString() == cboAño.Text)
                {
                 ListViewItem fila = new ListViewItem(li.codigo.ToString());
                    fila.SubItems.Add(li.titulo);
                    fila.SubItems.Add(li.autor);
                    fila.SubItems.Add(li.añoEdicion.ToString());
                    fila.SubItems.Add(li.costo.ToString());
                    lvLibrosxAño.Items.Add(fila);
                }
            }
        }

    }
}
```

Se inicia declarando en forma global el objeto de la colección Queue aLibros, el cual permite almacenar todos los libros registrados en la aplicación. En el evento load del formulario se asigna el método generaCodigo al control lblCodigo, ya que al iniciar la aplicación deberá mostrar un código autogenerado de libro. El método generaCodigo inicia comparando la cantidad de registros almacenados en la colección. Si no hay registros, el código automático será uno, caso contrario se recorre por todos los registros hasta obtener el último código de libro registrado y se devuelve el mismo código aumentado en uno.

El botón GENERA CÓDIGO muestra el código autogenerado en el control lblCodigo. El botón Registrar envía la información desde los controles a la clase Libro. Asimismo, se valida que el título del libro no se haya registrado antes. Si es así, se le envía el mensaje «Libro ya se encuentra registrado», caso contrario se agrega los datos del libro en la colección Queue con el método Enqueue, que permite agregar el registro como una cola de elementos. Asimismo, es importante resaltar que las fichas «búsqueda por autor» y «Búsqueda por año de edición» deben llenar sus cuadros combinados.

El método llenaAutores tiene la misión de llenar con los datos del autor en el cuadro combinado en la ficha «búsqueda por autor». Se inicia creando una variable llamada autores que guarda solo la información de los autores filtrados, es decir, sin repetir valores. El método distinct hace que los valores a.autor sean únicos dentro de la consulta LinQ. Luego, para asignar dichos datos al cuadro combinado, se debe usar la propiedad DataSource del cuadro combinado y aquí se especifica la variable que contiene los datos filtrados de los autores y finalmente se debe especificar el nombre de la columna a mostrar en el cuadro combinado usando la propiedad DisplayMember, que se usa para visualizar los datos. El valor «autor» representa el nombre del atributo que se le asignó en la clase Libro. De la misma forma, se trata con los años de edición de los libros.

MACRO

EDITORIAL
MACRO®

3

Capítulo

Serialización

Capacidad terminal:

Implementar aplicaciones de tipo colección con archivos binarios y XML.

Contenido

- Serialización
- Ventajas de la serialización
- Clase SaveFileDialog
- Clase OpenFileDialog
- Clase FileStream
  Caso Desarrollado 1: Serialización - Manejo de cadena
  Caso Desarrollado 2: Serialización de datos - Registro de cliente
  Caso Desarrollado 3: Serialización de datos e imagen - Registro de cliente con imagen
- Clase XMLSerializer
  Caso Desarrollado 4: Serialización XML - Registro de producto
  Caso Desarrollado 5: Serialización XML - Registro de productos usando arreglos
  Caso Desarrollado 6: Serialización XML - Obtener registros de un archivo XML

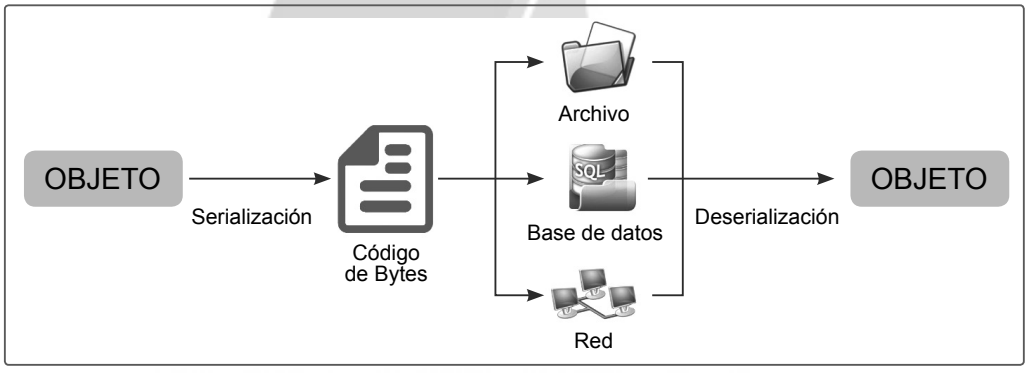
EDITORIAL MACRO®

EDITORIAL
MACRO®

3.1 Serialización

El término serialización es considerado como el proceso de convertir un objeto particular en una secuencia de bytes, que después podrán ser convertidos en otros objetos. Un ejemplo que se puede mencionar es el hecho de que una base de datos necesite almacenar información de una imagen. Esta imagen previamente debe serializarse para poder almacenarlo en la base de datos y así se gana en espacio en disco y se protege la información de la imagen, ya que esta queda registrada de forma permanente.

Como indica su teoría, es a partir de un objeto que se inicia el proceso de serialización, pasando por una conversión a bytes y finalmente caer en un elemento destino que puede ser una base de datos, un archivo de texto o simplemente se guarda en la memoria de la computadora.



Cuando se implementa una serialización, el Framework de .NET puede usar tres formatos como destino el Binario, Soap y Xml.

- **La serialización binaria**

  Consiste en convertir la información a bytes y se utiliza comúnmente cuando la información es transferida por una red hacia un sistema destino, el cual recibe dicha información y realiza el proceso inverso de la serialización llamada DeSerialización para reconstruir el objeto que fue transferido. El espacio de nombre es System. Runtime.Serialization.

- **La serialización Soap**

  Consiste en convertir los datos en un «documento estándar», en el cual se incluirá, además de los datos a serializar, una serie de información adicional que será utilizada por el sistema destino para construir el objeto original. Esta serialización es usada comúnmente con servicios web. El espacio de nombre es System.Runtime. Serialization.

EDITORIAL MACRO®

- **La serialización Xml**

  Consiste en transformar la información en un documento Xml que será interpretado por el sistema destino. El espacio de nombre es System.Xml.Serialization.

3.2 Ventajas de la serialización

La serialización da un soporte necesario al desarrollador de poder manipular información contenida dentro de un objeto, guardarlo y abrirlo cuando lo necesite. Entonces, la serialización permite:

- Guardar el estado de un objeto y volver a crearlo.
- Proporcionar almacenamiento de objetos e intercambio de datos.
- Enviar un objeto a una aplicación remota por medio de los servicios web.
- Mantener información relevante para el usuario entre aplicaciones.
- Facilitar la comunicación de datos. Si la información se transfiere en binario o XML, cualquier aplicación podría escribir un documento de texto plano con los datos que estaban manejando y otra aplicación podría recibir esta información para trabajar con ella.
- La migración de datos. Si se tiene que mover los datos de una base de datos a otra, sería muy sencillo si las dos trabajan, por ejemplo, en formato XML.

3.3 Clase SaveFileDialog

Es una clase que permite mostrar una ventana de diálogo de grabación de archivo. Es a partir de aquí que se puede guardar algún tipo de archivo especificado por el usuario.

Método constructor:

SaveFileDialog()	Método que permite inicializar una instancia de la clase SaveFileDialog. Por ejemplo, si se necesita crear un objeto de la clase SaveFileDialog, se puede usar los siguientes códigos:
	`SaveFileDialog sv = new SaveFileDialog();`
	O también:
	`SaveFileDialog sv;`
	`sv = new SaveFileDialog();`

Propiedades:

FileName	Permite obtener el nombre del archivo asignado por el usuario y la extensión asignada a dicho archivo. Por ejemplo, si se necesita obtener el nombre y el tipo de archivo asignado a un archivo, se puede usar el siguiente código: `string ruta =  sv.FileName;` Donde sv es el objeto de la clase SaveFileDialog y se asume que el usuario ya registró el nombre y seleccionó un tipo de archivo.

Filter	Establece el tipo de archivo. El formato a seguir es primero el texto que visualizará el usuario y luego la extensión del mismo. Por ejemplo, si se necesita definir la extensión txt, se podría usar el siguiente código:
	`SaveFileDialog sv = new SaveFileDialog();`
	`sv.filter = "Archivo de texto|*.txt";`
	Para separar el texto que visualiza el usuario del tipo de archivo, se usa el símbolo barra vertical | (alt+124).

Método:

ShowDialog	Permite mostrar el objeto de cuadro de diálogo mediante la condición de selección de botón. Por ejemplo, si se necesita mostrar el cuadro de diálogo, el código podría ser de la siguiente manera:
	`SaveFileDialog sv = new SaveFileDialog();`
	`sv.filter = "Archivo de texto|*.txt";`
	`if (sv.ShowDialog() == DialogResult.OK){`
	`  //Implementación de código`
	`}`

3.4 Clase OpenFileDialog

Es una clase que permite mostrar una ventana de diálogo de apertura de archivo. Es a partir de aquí que se podrá abrir y mostrar la información de un archivo especificado por el usuario.

Método constructor:

OpenFileDialog()	Método que permite inicializar una instancia de la clase OpenFileDialog. Por ejemplo, si se necesita crear un objeto de la clase OpenFileDialog, se puede usar los siguientes códigos:
	`OpenFileDialog sv = new OpenFileDialog();`
	O también:
	`OpenFileDialog op;`
	`op = new OpenFileDialog();`

Propiedades:

FileName	Permite obtener el nombre del archivo asignado por el usuario y obtener la extensión asignada a dicho archivo. Por ejemplo, si se necesita obtener el nombre y el tipo de archivo asignado a un archivo, se puede usar el siguiente código: `string ruta =  op.FileName;` Donde op es el objeto de la clase OpenFileDialog y se asume que el usuario ya registró el nombre y seleccionó un tipo de archivo.

Filter	Establece el tipo de archivo. El formato a seguir es primero el texto que visualizará el usuario y luego la extensión del mismo. Por ejemplo, si se necesita definir la extensión txt, se podría usar el siguiente código:
	`OpenFileDialog op = new OpenFileDialog();`
	`op.filter = "Archivo de texto|*.txt";`
	Para separar el texto que visualiza el usuario del tipo de archivo, se usa el símbolo barra vertical | (alt+124).

Método:

ShowDialog	Permite mostrar el objeto de cuadro de diálogo mediante la condición de selección de botón. Por ejemplo, si se necesita mostrar el cuadro de diálogo, el código podría ser de la siguiente manera:
	`OpenFileDialog op = new OpenFileDialog();`
	`op.filter = "Archivo de texto|*.txt";`
	`if (op.ShowDialog() == DialogResult.OK){`
	`  //Implementación de código`
	`}`

3.5 Clase FileStream

La clase FileStream permite leer, escribir, abrir y cerrar archivos en un sistema de archivos, y manipular otros manejadores relacionados con archivos del sistema operativo, incluyendo entrada estándar y salida estándar.

Método constructor:

FileStream (Archivo, FileMode)	Permite inicializar una nueva instancia de la clase FileStream, especificando el nombre del archivo y el modo de acceso al archivo. Observe algunas opciones:
	Abrir el archivo:
	`FileStream fs = new FileStream("c:\archivo.txt",FileMode.Open);`
	Modo de creación del archivo:
	`FileStream fs = new FileStream("c:\archivo.txt",FileMode.Create);`

Métodos:

Close	Método que permite cerrar la secuencia FileStream actual y libera todos los recursos asociados a esta. Por ejemplo: `fs.Close();`
Dispose	Método que libera todos los recursos utilizados por la secuencia FileStream. Por ejemplo: `fs.Dispose();`
Flush	Método que permite borrar los búferes de la secuencia FileStream y hace que todos los datos almacenados en los búferes se escriban en el archivo. `Fs.Flush();`

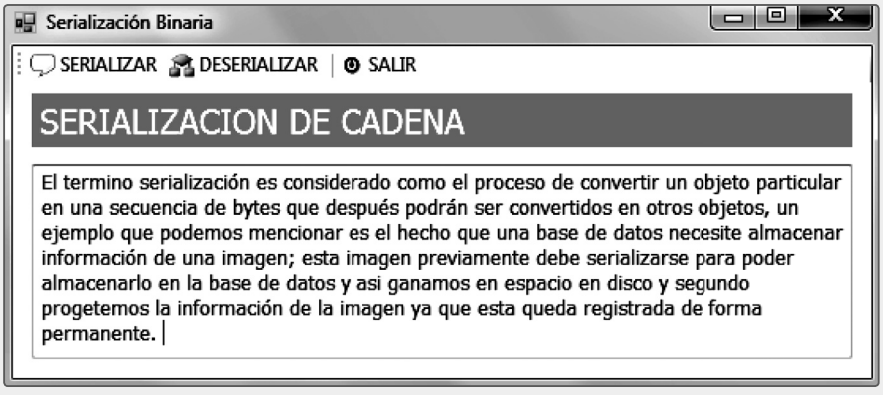
Caso desarrollado 1 : Serialización - Manejo de cadena

Implementar una aplicación que permita ingresar una cadena de texto y, por medio de la serialización, convierte la cadena en un archivo binario.

Se tiene en cuenta lo siguiente:

- Serializar implica grabar el archivo con extensión binaria.
- Deserializar implica invertir la serialización, es decir, el archivo de extensión binaria se convertirá en archivo de texto.

GUI Propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjSerializaciónTexto.

Clase: frmSerializa

```
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;

//Librerias necesarias para la Serialización
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace pjSerializaciónTexto
{
    public partial class frmSerializa : Form
    {
        public frmSerializa()
        {
            InitializeComponent();
        }

        private void tsSerializar_Click(object sender, EventArgs e)
        {
            //Objeto de la clase SaveFileDialog
            SaveFileDialog sv = new SaveFileDialog();

            //Definiendo el tipo de archivo
            sv .Filter = "Archivo binario|*.bin";
```

```
            //Mostrando el cuadro de dialogo de grabacion
            if (sv.ShowDialog() == DialogResult.OK)
            {
                //Creando el archivo
                using (FileStream fs = new
                                FileStream(sv.FileName, FileMode.Create))
                {
                    //Convirtiendo en binario
                    BinaryFormatter bf = new BinaryFormatter();
                    bf.Serialize(fs, txtCadena.Text);
                }
            }
        }

        private void tsDeserializar_Click(object sender, EventArgs e)
        {
            //Objeto de la clase OpenFileDialog
            OpenFileDialog op = new OpenFileDialog();

            //Definiendo el tipo de archivo
            op.Filter = "Archivo binario|*.bin";

            //Mostrando el cuadro de dialogo de apertura
            if (op.ShowDialog() == DialogResult.OK)
            {
                //Abriendo el archivo
                using (FileStream fs = new
                                 FileStream(op.FileName, FileMode.Open))
                {
                    //Convirtiendo el binario en archivo de texto
                    BinaryFormatter bf = new BinaryFormatter();
                    txtCadena.Text = bf.Deserialize(fs).ToString();
                }
            }
        }
    }
}
```

Se debe tener en cuenta que las librerías a usar en la aplicación son IO para temas de grabación y apertura de archivos y Runtime.Serialization.Formatters.Binary para la serialización binaria de los datos.

En el botón Serializar se usa la clase SaveFileDialog. Esto permite mostrar un cuadro de diálogo para la grabación de un determinado tipo de archivo. Para especificar los tipos, se usa la sentencia sv.Filter que define el tipo de datos que tendrá el archivo. En este caso se usa *.bin. y luego sea crea el archivo mediante la clase FileStrem y para convertir a binario el archivo se usa la clase BinaryFormatter mediante el método serialize.

En el botón Deserializar se usa la clase OpenFileDialog, el cual permite mostrar un cuadro de diálogo de apertura de archivo, especificando en este caso el tipo de archivo *.bin en la propiedad Filter. Luego, se abre el archivo mediante el uso de la clase FileStrem y se aplica el método Deserialize proveniente de la clase BinaryFormatter para convertir el texto serializado en un texto legible.

Caso desarrollado 2 : Serialización de datos - Registro de cliente

Implementar una aplicación que permita registrar y mostrar los datos de un cliente. Estos datos se guardarán en un archivo binario usando serialización.

Se tiene en cuenta lo siguiente:

- Implementar la clase Cliente que contenga el nombre completo, apellido paterno, materno, teléfono y correo electrónico de un cliente.
- Implementar un botón de registro de datos de cliente, que permita serializar los datos ingresados.
- Implementar un botón de mostrar los datos del cliente, que permita deserializar el archivo y muestre los datos en sus controles originales.

GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjSerializaciónDatos.
2. Agregue al proyecto la clase Cliente e implemente el siguiente código:

Script de la clase Cliente

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.Serialization;

namespace pjSerializaciónDatos
{
    [Serializable()]
    public class Cliente
    {
```

```
        public string nombres { get; set; }
        public string paterno { get; set; }
        public string materno { get; set; }
        public string telefono { get; set; }
        public string email { get; set; }
    }
}
```

La clase Cliente contiene todos los atributos que pertenecen al registro de un determinado cliente. Se debe tener en cuenta que la serialización binaria se realizará en esta clase. Por lo tanto, antes de la implementación de la clase, se debe colocar [Serializable()], esto hará que los atributos de la clase Cliente sean serializables.

Clase: frmRegistroCliente

```
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;
//Librerias
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;

namespace pjSerializaciónDatos
{
    public partial class frmRegistroCliente : Form
    {
        public frmRegistroCliente()
        {
            InitializeComponent();
        }

        private void tsRegistrar_Click(object sender, EventArgs e)
        {
            try
            {
                //Objeto de clase cliente
                Cliente objC = new Cliente();
                objC.nombres = txtNombres.Text;
                objC.paterno = txtPaterno.Text;
                objC.materno = txtMaterno.Text;
                objC.telefono= txtFono.Text;
                objC.email = txtEmail.Text;
```

```
            //Objeto de la clase SaveFileDialog
            SaveFileDialog sv = new SaveFileDialog();

            //Definiendo el tipo de datos
            sv.Filter = "Archivo binario|*.bin";

            if (sv.ShowDialog() == DialogResult.OK)
            {
                //Creando el archivo binario
                using (FileStream fs = new
                            FileStream(sv.FileName, FileMode.Create))
                {
                    //Serializando el archivo
                    BinaryFormatter bf = new BinaryFormatter();
                    bf.Serialize(fs, objC);
                }
            }
        }
        catch (Exception)
        {
            MessageBox.Show("Error al intentar registrar el
                                                archivo de cliente");
        }

    }

    private void tsMostrar_Click(object sender, EventArgs e)
    {
        try
        {
            //Objeto de la clase OpenFileDialog
            OpenFileDialog op = new OpenFileDialog();

            //Definiendo el tipo de archivo
            op.Filter = "Archivo binario|*.bin";

            if (op.ShowDialog() == DialogResult.OK)
            {
                //Abriendo el archivo binario
                using (FileStream fs = new
                              FileStream(op.FileName, FileMode.Open))
                {
                    //Deserializando
                    BinaryFormatter bf = new BinaryFormatter();
                    Cliente objC = (Cliente)(bf.Deserialize(fs));

                 //Enviando los valores deserializados a los controles
                    txtNombres.Text = objC.nombres;
                    txtPaterno.Text = objC.paterno;
                    txtMaterno.Text = objC.materno;
                    txtFono.Text = objC.telefono;
```

```
                        txtEmail.Text = objC.email;
                    }
                }
            }
            catch (Exception) {
                MessageBox.Show("Error al intentar abrir el archivo");
            }
        }

        private void tsSalir_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

Se empieza por definir las librerías System.IO para la apertura y creación de archivo. System.Runtime.Serialization.Formatters.Binary se usa para la clase BinaryFormatter, el cual permitirá convertir los datos del cliente en un archivo bin.

En el botón Registrar se crea un objeto de la clase Cliente y se envía todos los valores del cliente ingresado en los controles del formulario. Luego, se crea un objeto de la clase SaveFileDialog para poder grabar el archivo, no olvidarse de definir el filtro del tipo de archivo como *.bin. Después, se crea el archivo con la clase FileStream y finalmente se convierte los datos en un archivo binario con la clase BinaryFormatter mediante el método serialize, el cual especifica el archivo y el objeto de la clase que se serializará.

En el botón Mostrar se crea un objeto de la clase OpenFileDialog para poder abrir el archivo de tipo bin. Luego, se abre el archivo usando la clase FileStream y se deserializa la clase Cliente usando la clase BinaryFormatter mediante el método deserialize. Finalmente, se descargará los valores enviados al objeto de la clase Cliente a los controles del formulario.

Caso desarrollado 3 : Serialización de datos e imagen - Registro de cliente con imagen

Implementar una aplicación que permita registrar y mostrar los datos de un cliente adicionando su imagen. Estos datos se guardarán en un archivo binario usando serialización.

Se tiene en cuenta lo siguiente:

- Implementar la clase Cliente que contenga el nombre completo, apellido paterno, materno, teléfono, correo electrónico y la imagen de un cliente.

- Implementar el control ToolStrip para mostrar los botones Registrar cliente, Mostrar cliente y Salir. El botón Registrar grabará los datos del cliente, incluido su imagen en un archivo binario. En tanto, mostrar cliente permitirá seleccionar un archivo binario y cargará los datos del cliente en los controles del formulario, también mostrará su imagen correspondiente.

GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjSerializaciónDatos.
2. Agregue al proyecto la clase Cliente e implemente el siguiente código:

Script de la clase Cliente:

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

using System.Drawing;
using System.Runtime.Serialization;

namespace pjSerializaciónDatos
{
    [Serializable()]
    public class Cliente
    {
        public string nombres { get; set; }
        public string paterno { get; set; }
        public string materno { get; set; }
        public string telefono { get; set; }
        public string email { get; set; }
        public Bitmap foto { get; set; }
    }
}
```

La clase Cliente usa la librería System.Drawing para el tipo de datos Bitmap, que usa la imagen del cliente y System.Runtime.Serialization para serializar la clase Cliente mediante la sentencia [Serializable()], colocada antes de la implementación de la clase Cliente. No se debe olvidar que la imagen del cliente debe ser declarada de tipo Bitmap.

Clase: frmRegistroCliente

```
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;
//Librerias
using System.IO;
using System.Drawing;
using System.Runtime.Serialization.Formatters.Binary;

namespace pjSerializaciónDatos
{
    public partial class frmRegistroCliente : Form
    {
        public frmRegistroCliente()
        {
            InitializeComponent();
        }

        private void tsRegistrar_Click(object sender, EventArgs e)
        {
            try
            {
                //Objeto de clase cliente
                Cliente objC = new Cliente();
                objC.nombres = txtNombres.Text;
                objC.paterno = txtPaterno.Text;
                objC.materno = txtMaterno.Text;
                objC.telefono= txtFono.Text;
                objC.email = txtEmail.Text;
                objC.foto = (Bitmap)(pbFoto.Image);

                //Objeto de la clase SaveFileDialog
                SaveFileDialog sv = new SaveFileDialog();

                //Definiendo el tipo de datos
                sv.Filter = "Archivo binario|*.bin";

                if (sv.ShowDialog() == DialogResult.OK)
                {
                    //Creando el archivo binario
                    using (FileStream fs = new
```

```
                                FileStream(sv.FileName, FileMode.Create))
                {
                    //Serializando el archivo
                    BinaryFormatter bf = new BinaryFormatter();
                    bf.Serialize(fs, objC);
                }
            }
        }
        catch (Exception)
        {
            MessageBox.Show("Error al intentar registrar el
                                                archivo de cliente");
        }
    }

    private void tsMostrar_Click(object sender, EventArgs e)
    {
        try
        {
            //Objeto de la clase OpenFileDialog
            OpenFileDialog op = new OpenFileDialog();

            //Definiendo el tipo de archivo
            op.Filter = "Archivo binario|*.bin";

            if (op.ShowDialog() == DialogResult.OK)
            {
                //Abriendo el archivo binario
                using (FileStream fs = new
                                FileStream(op.FileName, FileMode.Open))
                {
                    //Deserializando
                    BinaryFormatter bf = new BinaryFormatter();
                    Cliente objC = (Cliente)(bf.Deserialize(fs));

                    //Enviando la deSerialización a los controles
                    txtNombres.Text = objC.nombres;
                    txtPaterno.Text = objC.paterno;
                    txtMaterno.Text = objC.materno;
                    txtFono.Text = objC.telefono;
                    txtEmail.Text = objC.email;
                    pbFoto.Image = objC.foto;
                   pbFoto.SizeMode = PictureBoxSizeMode.StretchImage;
                }
            }
        }
        catch (Exception) {
            MessageBox.Show("Error al intentar abrir el archivo");
        }
    }

    private void tsSalir_Click(object sender, EventArgs e)
    {
        this.Close();
```

```
        }

        private void btnExaminar_Click(object sender, EventArgs e)
        {
            try
            {
                OpenFileDialog op = new OpenFileDialog();
                op.Filter = "Imagenes JPG|*.jpg";

                if (op.ShowDialog() == DialogResult.OK)
                {
                    pbFoto.Image = Image.FromFile(op.FileName);
                    pbFoto.SizeMode = PictureBoxSizeMode.StretchImage;
                }
            }
            catch (Exception) {
                MessageBox.Show("Error al cargar la imagen");
            }
        }
    }
}
```

Se inicia declarando las librerías System.IO para poder crear y abrir archivos. System. Drawing permite manejar las clases que ayudarán a hacer uso de la imagen y System. Runtime.Serialization.formatters.Binary para convertir los datos cliente y su imagen a un archivo binario.

En el botón Registrar se crea el objeto de la clase Cliente para enviar toda la información referente al cliente, en el caso de la imagen se usa la sentencia objC.foto = (Bitmap)(pbFoto.Image), donde pbfoto.Image representa al control PictureBox y (Bitmap) aplica un casting a la imagen para registrarlo en el atributo foto de la clase Cliente. Se especifica el objeto de la clase SavefileDialog para poder grabar el archivo binario. Para crear el archivo binario, se usa la clase FileStream y finalmente se usa la clase BinaryFormatter. Se convierte los datos obtenidos en un archivo binario.

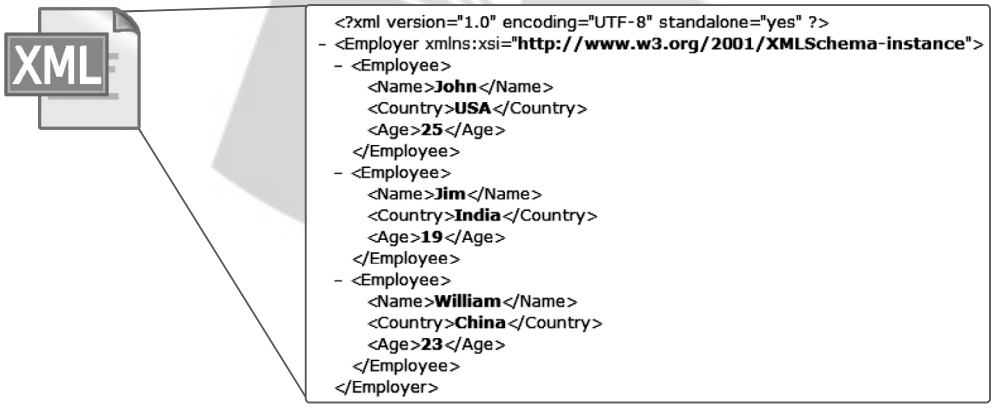
En el botón Mostrar se crea un objeto de la clase OpenFileDialog para poder mostrar el cuadro de diálogo de apertura de archivo y usando la clase FileStream se abre el archivo de tipo bin. Finalmente, la clase BinaryFormatter obtiene la información de los clientes mediante el método deserialize para poder devolver los valores a los controles del formulario. En el caso de la imagen, se usa la sentencia pbFoto.Image = objC.foto, donde la propiedad Image asigna al control PictureBox la imagen desde la clase Cliente.

El botón Examinar permitirá asignar una imagen desde cualquier unidad al control PictureBox. Para esto, se usa la clase OpenFileDialog para abrir el archivo de tipo imagen especificado en el filter de la clase y la sentencia Image.FromFile permite asignar la imagen obtenida en el control PictureBox.

3.6 Clase XMLSerializer

La serialización XML es el proceso mediante el cual los campos y las propiedades públicas de un campo se convierten a un formato de tipo XML para efectos de almacenamiento o transporte. La deserialización vuelve a crear el objeto en su estado original a partir de los resultados XML.

La serialización puede considerarse como una manera de guardar el estado de un objeto en una secuencia o un búfer. Para transferir datos entre los objetos y XML, es necesaria una asignación de las construcciones del lenguaje de programación al esquema XML y viceversa.



Método constructor:

XmlSerializer(Type)	Método que permite inicializar una nueva instancia de la clase de XmlSerializer, que puede serializar objetos del tipo especificado en documentos XML, y deserializa documentos XML en objetos del tipo especificado.

Métodos:

Deserialize(Stream)	Deserializa el documento XML contenido en stream especificado.
Finalize	Permite que un objeto intente liberar recursos y realizar otras operaciones de limpieza antes de ser reclamado por la recolección de elementos no utilizados. (Se hereda de Object).
Serialize(Stream, Object)	Serializa Object especificado y escribe el documento XML en un archivo con stream especificado.

EDITORIAL MACRO®

Caso desarrollado 4 : Serialización XML - Registro de producto

Implementar una aplicación que permita registrar los datos de un producto en un archivo llamado producto.xml. La aplicación deberá solicitar los datos de un determinado producto.

Se tiene en cuenta lo siguiente:

- Implementar la clase Producto de la cual se puede registrar la descripción, la categoría, el precio, la fecha de vencimiento y la cantidad.
- La categoría de los productos son golosinas, bebidas, lácteos y abarrotes.
- Implementar el control ToolStrip para los botones Grabar XML, Mostrar XML y Salir. El botón Grabar mostrará la ventana de grabación de archivo para XML; y el botón Mostrar permitirá abrir el archivo XML y mostrar los datos del producto en sus controles.
- El posible código del archivo XML del producto sería:

```
▼<Producto xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <descripcion>Caramelo de licor</descripcion>
   <categoria>Golosina</categoria>
   <precio>0.5</precio>
   <fecha>2015-12-12T00:00:00</fecha>
   <cantidad>100</cantidad>
 </Producto>
```

GUI Propuesto:

Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjSerializaciónXML.

2. Agregue al proyecto la clase Producto e implemente el siguiente código:

Script de la clase Producto:

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

namespace pjSerializaciónXML
{
    public class Producto
    {
        public string descripcion { get; set; }
        public string categoria { get; set; }
        public double precio{ get; set; }
        public DateTime fecha { get; set; }
        public int cantidad { get; set; }
    }
}
```

La clase Producto contiene la especificación de los atributos del producto a registrar en el archivo XML. Se debe tener en cuenta que los nombres de los atributos van a componer las etiquetas del archivo XML.

Clase frmSerializaciónXML

```
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;

using System.Collections;
using System.IO;
using System.Xml.Serialization;

namespace pjSerializaciónXML
{
    public partial class frmRegistroXML : Form
    {

        //Inicialización del arreglo de categorías
        static string[] categoria = { "Bebida", "Golosina",
                                      "Lacteo", "Abarrote"};
```

```
        //Creando el objeto de la clase ArrayList
        ArrayList aCategorias = new ArrayList(categoria);


        public frmRegistroXML()
        {
            InitializeComponent();
        }

        private void frmRegistroXML_Load(object sender, EventArgs e)
        {
            foreach (string c in aCategorias)
            {
                cboCategoria.Items.Add(c);
            }

        }

        private void tsGrabar_Click(object sender, EventArgs e)
        {
            SaveFileDialog sv = new SaveFileDialog();
            sv.Filter = "Archivo XML|*.xml";

            if (sv.ShowDialog() == DialogResult.OK)
            {
                Producto objP = new Producto();
                objP.descripcion = txtDescripcion.Text;
                objP.categoria = cboCategoria.Text;
                objP.precio = double.Parse(txtPrecio.Text);
                objP.fecha= DateTime.Parse(txtFecha.Text);
                objP.cantidad = int.Parse(txtCantidad.Text);

                using (FileStream fs = new
                                FileStream(sv.FileName, FileMode.Create))
                {
                 XmlSerializer xml = new XmlSerializer(typeof(Producto));
                    xml.Serialize(fs, objP);
                }
            }

        }

        private void tsMostrar_Click(object sender, EventArgs e)
        {
            OpenFileDialog op = new OpenFileDialog();
            op.Filter = "Archivo XML|*.xml";
            if (op.ShowDialog() == DialogResult.OK)
            {
                using (FileStream fs =
                              new FileStream(op.FileName, FileMode.Open))
                {
                 XmlSerializer xml = new XmlSerializer(typeof(Producto));
```

```
                    Producto objP = (Producto)xml.Deserialize(fs);
                    txtDescripcion.Text = objP.descripcion;
                    cboCategoria.Text = objP.categoria;
                    txtPrecio.Text = objP.precio.ToString();
                    txtFecha.Text = objP.fecha.ToShortDateString();
                    txtCantidad.Text = objP.cantidad.ToString();
                }
            }
        }
    }
}
```

Use las librerías System.Collections para almacenar las categorías de los productos en un ArrayList, System.IO para aperturar y crear un archivo en este caso XML y System. XML.Serialiation que permite generar el archivo XML.

En la sección de declaración global se declara la variable categoría, que se inicializa con las categorías de los productos que serán enviados al ArrayList. Asimismo, se declara el objeto aCategorias de la clase ArrayList para luego ser enviado al cuadro combinado que puede visualizarse en el evento load de la aplicación.

En el botón Grabar se crea un objeto de la clase SaveFileDialog para mostrar el cuadro de diálogo de grabación de archivo, el cual especifica como tipo a *.xml en el filter del objeto. Se crea un objeto de clase Producto para enviar los valores registrados en los controles del formulario. Mediante el objeto de la clase FileStream, se creará el archivo de tipo XML y finalmente el método serialize de la clase XmlSerializer creará el archivo XML con etiquetas referentes a los atributos de la clase Producto.

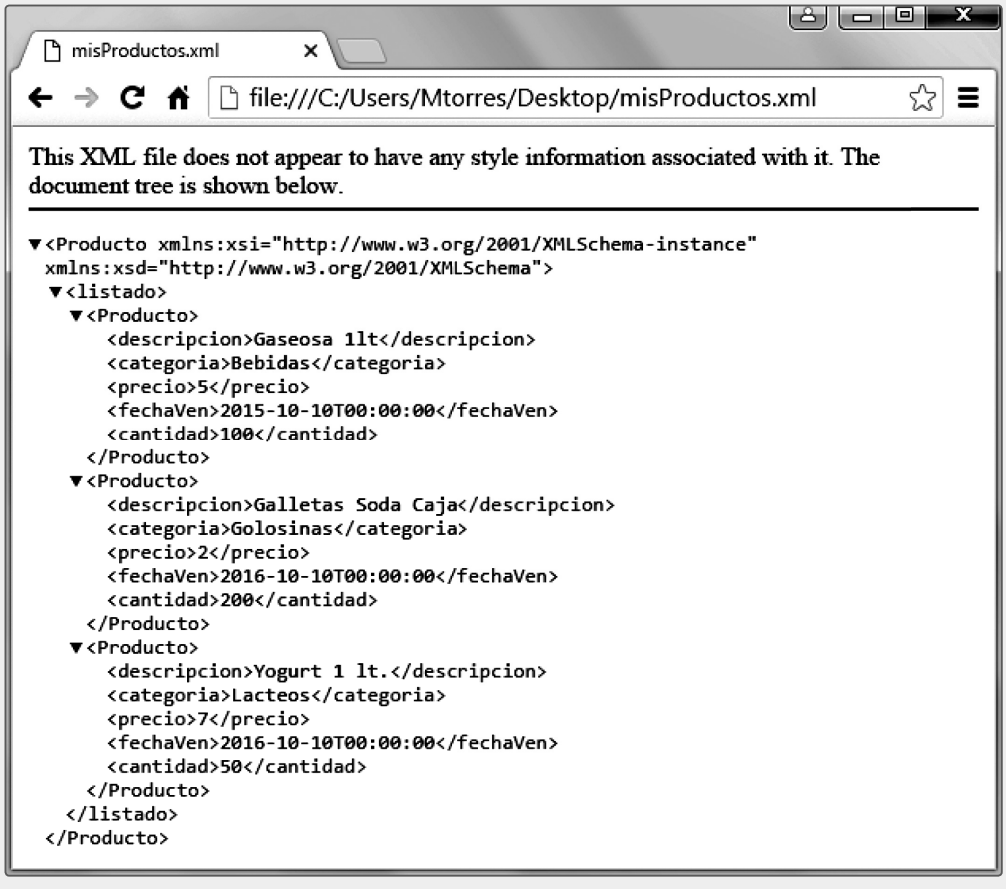
En el botón Mostrar se crea un objeto de la clase OpenFileDialog para mostrar el cuadro de diálogo de apertura de archivo, el cual especifica como tipo a *.xml en el filter del objeto. La clase FileStream permitirá abrir el archivo XML y es la clase XmlSerializer por medio del método deserialize, que convierte el archivo XML en un archivo legible y que serán enviados a sus controles del formulario.

Caso desarrollado 5 : Serialización XML - Registro de productos usando arreglos

Implementar una aplicación que permita registrar los datos de varios productos en un archivo llamado misProductos.xml. La aplicación deberá solicitar los datos de un determinado producto, luego almacenarlos en un arreglo y a partir de este enviarlo a un archivo XML.

Se tiene en cuenta lo siguiente:

- Implementar la clase Producto, de la cual se puede registrar la descripción, la categoría, el precio, la fecha de vencimiento y la cantidad.
- Implementar la clase arregloProductos, la cual permita registrar la colección de productos en un archivo XML.
- La categoría de los productos son golosinas, bebidas, lácteos y abarrotes.
- Implementar un botón de agregar, que permita registrar los datos del producto en un control ListView.
- Implementar un botón eliminar producto, que permita eliminar un determinado producto desde el control ListView. Para esto, se debe solicitar el nombre del producto a eliminar.
- Implementar un botón registrar productos en XML, que permita registrar los datos de todos los productos expuestos en el control ListView y enviarlo a un archivo XML.
- El posible código del archivo XML de los productos sería:




```
▼<Producto xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  ▼<listado>
    ▼<Producto>
       <descripcion>Gaseosa 1lt</descripcion>
       <categoria>Bebidas</categoria>
       <precio>5</precio>
       <fechaVen>2015-10-10T00:00:00</fechaVen>
       <cantidad>100</cantidad>
     </Producto>
    ▼<Producto>
       <descripcion>Galletas Soda Caja</descripcion>
       <categoria>Golosinas</categoria>
       <precio>2</precio>
       <fechaVen>2016-10-10T00:00:00</fechaVen>
       <cantidad>200</cantidad>
     </Producto>
    ▼<Producto>
       <descripcion>Yogurt 1 lt.</descripcion>
       <categoria>Lacteos</categoria>
       <precio>7</precio>
       <fechaVen>2016-10-10T00:00:00</fechaVen>
       <cantidad>50</cantidad>
     </Producto>
   </listado>
 </Producto>
```

GUI Propuesto:

Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjProductosXML.
2. Agregue al proyecto la clase Producto e implemente el siguiente código:

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

namespace pjProductosXML
{
    public class Producto
    {
        public string descripcion { get; set; }
        public string categoria { get; set; }
        public double precio { get; set; }
        public DateTime fechaVen { get; set; }
        public int cantidad { get; set; }
    }
}
```

La clase Producto contiene todos los atributos correspondientes a la descripción del producto a registrar.

Clase ArregloProductos

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

namespace pjProductosXML
{
    [XmlRoot("Producto")]
    public class ArregloProductos
    {
        [XmlArrayItem(typeof(Producto))]
        public List<Producto> listado = new List<Producto>();
    }
}
```

La clase arregloProductos contiene una colección que permite asociarse a la clase Producto para lograr registrar varios productos en una misma colección. Se debe agregar la librería System.Xml.Serialization para el uso de la sentencia XmlRoot.

Clase frmProducto

```
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 System.IO;
using System.Xml.Serialization;
using Microsoft.VisualBasic;

namespace pjProductosXML
{
    public partial class frmProductos : Form
    {
        //Objeto de la clase ArregloProductos
        ArregloProductos aProductos = new ArregloProductos();

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

```
private void btnAgregar_Click(object sender, EventArgs e)
{
    //Enviando los valores a la clase Producto
    Producto objP = new Producto();
    objP.descripcion = txtDescripcion.Text;
    objP.categoria = cboCategoria.Text;
    objP.precio = double.Parse(txtPrecio.Text);
    objP.fechaVen = DateTime.Parse(txtFechaVen.Text);
    objP.cantidad = int.Parse(txtCantidad.Text);

    //Enviando el producto al arreglo
    aProductos.listado.Add(objP);

    //Listando los registros
    listado();
}

//Método que permite listar los productos
void listado()
{
    lvProductos.Items.Clear();

    foreach (Producto p in aProductos.listado)
    {
        ListViewItem fila = new ListViewItem(p.descripcion);
        fila.SubItems.Add(p.categoria);
        fila.SubItems.Add(p.precio.ToString());
        fila.SubItems.Add(p.fechaVen.ToShortDateString());
        fila.SubItems.Add(p.cantidad.ToString());
        lvProductos.Items.Add(fila);
    }
}

private void btnEliminar_Click(object sender, EventArgs e)
{
  //Capturando el nombre del producto a eliminar
string descripcion = Interaction.InputBox("Ingrese nombre del producto");

  //Buscando el producto para eliminarlo
  foreach (Producto p in aProductos.listado)
  {
        if (p.descripcion == descripcion)
        {
            aProductos.listado.Remove(p);
            break;
        }
  }
    listado();
}
```

```
        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            SaveFileDialog sv = new SaveFileDialog();
            sv.Filter="Archivo XML|*.xml";
            if (sv.ShowDialog()==DialogResult.OK)
            {
             using (FileStream fs = new FileStream(sv.FileName,FileMode.Create))
                {
               XmlSerializer xml = new XmlSerializer(typeof(ArregloProductos));
                   xml.Serialize(fs,aProductos);
                }
            }
        }
    }
}
```

El botón Agregar permitirá enviar el registro de un producto al control ListView. El método listado se encarga de mostrar los registros guardados en la colección aProductos.

El botón Eliminar permite eliminar un registro de producto por medio de la descripción del producto. Para solicitar el nombre del producto, se usa la sentencia Interaction. InputBox, que necesita agregar la referencia Microsoft.VisualBasic (clic derecho en el proyecto > Agregar Referencia).

El botón Registrar enviará los registros a la clase XmlSerializer guardados en la colección arregloProductos.

Caso desarrollado 6 : Serialización XML - Obtener registros de un archivo XML

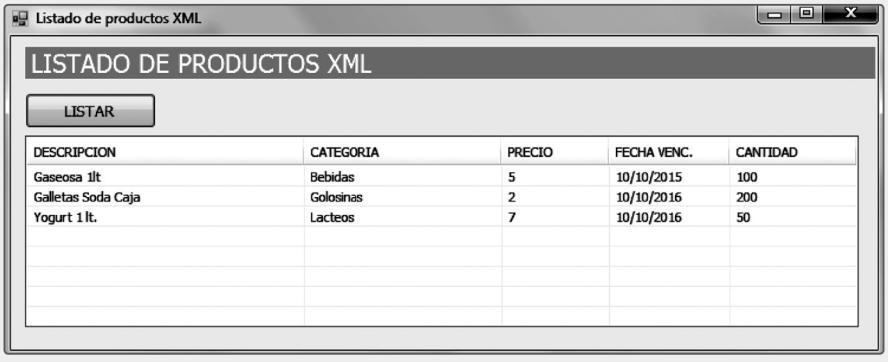
Implementar una aplicación que permita obtener los datos de los productos registrados en un archivo XML y mostrarlos en un control ListView.

Se tiene en cuenta lo siguiente:

- Implementar la clase Producto con los atributos descripción, categoría, precio, fecha de vencimiento y cantidad.
- Implementar la clase arregloProductos que permita obtener información de los productos provenientes de un archivo XML.

- Implementar un botón de listado que permita mostrar los registros de los productos almacenados en un archivo XML en un control ListView.

GUI propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo** > **Proyecto** > **Visual C#** > **Aplicación de Windows Forms** y asigne el nombre pjProductosXML_Abrir.
2. Agregue al proyecto la clase Producto e implemente el siguiente código:

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

namespace pjProductosXML_Abrir
{
    public class Producto
    {
        public string descripcion { get; set; }
        public string categoria { get; set; }
        public double precio { get; set; }
        public DateTime fechaVen { get; set; }
        public int cantidad { get; set; }
    }
}
```

La clase Producto declara los atributos pertenecientes a la descripción del producto a obtener del archivo XML.

Clase ArregloProductos

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

namespace pjProductosXML_Abrir
{
    [XmlRoot("Producto")]
    public class ArregloProductos
    {
        [XmlArrayItem(typeof(Producto))]
        public List<Producto> listado = new List<Producto>();
    }
}
```

La clase ArregloProductos define la colección de productos que vendrán desde el archivo XML.

Clase frmProducto

```
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 System.Xml.Serialization;
using System.IO;

namespace pjProductosXML_Abrir
{
 public partial class frmListadoProductos : Form
 {
   public frmListadoProductos()
   {
     InitializeComponent();
   }
```

```
    private void btnListar_Click(object sender, EventArgs e)
    {
      lvProductos.Items.Clear();
      OpenFileDialog open = new OpenFileDialog();
      open.Filter = "Archivo XML|*.xml";

      if (open.ShowDialog() == DialogResult.OK)
      {
       using (FileStream file = new FileStream(open.FileName, FileMode.Open))
        {
          XmlSerializer xml = new XmlSerializer(typeof(ArregloProductos));
           ArregloProductos pro = (ArregloProductos)xml.Deserialize(file);


           foreach (Producto p in pro.listado)
           {
             ListViewItem fila = new ListViewItem(p.descripcion);
             fila.SubItems.Add(p.categoria);
             fila.SubItems.Add(p.precio.ToString());
             fila.SubItems.Add(p.fechaVen.ToShortDateString());
             fila.SubItems.Add(p.cantidad.ToString());
             lvProductos.Items.Add(fila);
           }
          }
       }
     }
   }
 }
```

MACRO®

EDITORIAL
MACRO®

4

Capítulo

ADO.NET 4.6

Capacidad terminal:

Implementar aplicaciones de conexión a base de datos desde SQL Server con la tecnología ADO.NET y desarrollar aplicaciones de consulta, búsqueda y mantenimiento de registros.

EDITORIAL MACRO®

Contenido

- Script de la base de datos de origen
- Diagrama de base de datos Contrato
- Introducción al ADO.NET 4.6
- Novedades de ADO.NET 4.6
- Arquitectura de ADO.NET
- Componentes de ADO.NET
- Proveedores de datos
- Objetos principales de los proveedores de datos
- Espacio de nombres System.Data.SqlClient
- Espacio de nombre System.Data
- Clase SqlConnection
- Definición de la cadena de conexión
- Clase SqlDataAdapter
- Configuration Manager
- Clase DataSet
- Clase DataTable
- Clase Dataview
- Instrucción Using
- Clase SqlCommand
- Casos desarrollados

EDITORIAL
MACRO®

4.1 Script de la base de datos de origen

Para todas las aplicaciones cliente-servidor que se desarrollará de este capitulo en adelante, se usará la base de datos contrato, para lo cual se deberá ejecutar las siguientes sentencias en SQL Server, sin importar la versión del mismo:

- Script de la base de datos.

```
--DEFINIENDO EL FORMATO DE LA FECHA
SET DATEFORMAT DMY

---Verificando la existencia de la BD CONTRATO
IF DB_ID("CONTRATO") IS NOT NULL
BEGIN
        USE MASTER
        DROP DATABASE CONTRATO
END
GO

--Creando la BD CONTRATO
CREATE DATABASE CONTRATO
GO

--Activando la BD CONTRATO
USE CONTRATO
GO
```

EDITORIAL

- Script de la implementación de tablas.

```
CREATE TABLE CONTRATISTA(
        IDE_CON        CHAR(6)NOT NULL,
        NOM_CON        VARCHAR(30) NOT NULL,
        PAT_CON        VARCHAR(20) NOT NULL,
        MAT_CON        VARCHAR(20) NOT NULL,
        FON_CON        VARCHAR(15)    NOT NULL,
        EMA_CON        VARCHAR(50) NOT NULL)
GO

CREATE TABLE CLIENTE(
        IDE_CLICHAR(6)NOT NULL,
        NOM_CLIVARCHAR(30) NOT NULL,
        PAT_CLIVARCHAR(20) NOT NULL,
        MAT_CLIVARCHAR(50) NOT NULL,
        FON_CLIVARCHAR(15) NOT NULL,
        DNI_CLICHAR(8)NOT NULL,
        DIR_CLIVARCHAR(50)    NULL)
GO
```

```
CREATE TABLE EQUIPO(
        IDE_EQU        CHAR(6)NOT NULL,
        COD_TIP_EQU           CHAR(6)NOT NULL,
        DESC_EQU              VARCHAR(50) NOT NULL,
        PREC_EQU              MONEY          NULL,
        COD_EST        CHAR(6)NULL)
GO
CREATE TABLE CONTRATOALQUILER(
        COD_CONCHAR(6)NOT NULL,
        IDE_CLICHAR(6)NOT NULL,
        IDE_CONCHAR(6)NOT NULL,
        FIN_CONDATE           NOT NULL,
        FFI_CONDATE           NOT NULL,
        TIP_CONVARCHAR(50) NOT NULL)
GO
CREATE TABLE ESTADO_EQUIPO(
        COD_ESTCHAR(6)        NOT NULL,
        DES_ESTVARCHAR(100)   NOT NULL)
GO
CREATE TABLE DETALLE_CONTRATO_ALQUILER(
        COD_CONCHAR(6)NOT NULL,
        IDE_EQUCHAR(6)NOT NULL,
        UNI_CONINT            NOT NULL,
        DIA_CONINT            NOT NULL)
GO
CREATE TABLE TIPOMANTENIMIENTO(
        COD_TIP_MAN           CHAR(6)        NOT NULL,
        DES_TIP_MAN           VARCHAR(100)   NOT NULL)
GO
CREATE TABLE CONTRATOMANTENIMIENTO(
        COD_CONCHAR(6) NOT NULL,
        IDE_CLICHAR(6) NOT NULL,
        IDE_CONCHAR(6) NOT NULL,
        FEC_CONDATE     NOT NULL)
GO
CREATE TABLE DETALLE_CONTRATO_MANTENIMIENTO(
        COD_CONCHAR(6) NOT NULL,
        IDE_EQUCHAR(6) NOT NULL,
        COD_EQU_MAN    CHAR(6) NOT NULL)
GO
CREATE TABLE FICHADEDAÑOS(
        COD_FICCHAR(6)NOT NULL,
        IDE_CONCHAR(6)NOT NULL,
        IDE_CLICHAR(6)NOT NULL,
        FFI_FICDATE           NOT NULL)
GO
CREATE TABLE DETALLE_FICHA_DAÑO(
        COD_FICCHAR(6) NOT NULL,
        IDE_EQUCHAR(6) NOT NULL,
        COD_TIP_DAÑ    CHAR(6) NOT NULL,
        DES_DETVARCHAR(100) NOT NULL)
GO
```

```
CREATE TABLE TIPO_DAÑO(
       COD_TIP_DAÑ    CHAR(6) NOT NULL,
       DES_TIP_DAÑ    VARCHAR(100) NOT NULL)
GO
CREATE TABLE FICHARECLAMO(
       COD_FIC CHAR(6) NOT NULL,
       IDE_CON CHAR(6) NOT NULL,
       IDE_CLI CHAR(6) NOT NULL,
       FFI_FIC DATE           NOT NULL)
GO
CREATE TABLE DETALLE_FICHA_RECLAMO(
       COD_FIC CHAR(6) NOT NULL,
       IDE_EQU CHAR(6) NOT NULL,
       COD_TIP_REC    CHAR(6) NOT NULL,
       DES_DET VARCHAR(100) NOT NULL)
GO
CREATE TABLE TIPO_RECLAMO(
       COD_TIP_REC    CHAR(6)        NOT NULL,
       DESCTIP VARCHAR(100)   NOT NULL)
GO
CREATE TABLE TIPO_EQUIPO(
       COD_TIP_EQU    CHAR(6)        NOT NULL,
       DES_TIP VARCHAR(100)   NOT NULL)
GO
CREATE TABLE FICHA_DEVOLUCION(
       COD_FIC CHAR(6) NOT NULL,
       IDE_CON CHAR(6) NOT NULL,
       IDE_CLI CHAR(6) NOT NULL,
       IDE_EQU CHAR(6) NOT NULL,
       FDE_FIC DATE     NOT NULL,
       MOR_FIC MONEY   NULL)
GO
```

- Script de la implementación de las llaves primarias y foráneas.

```
--AGREGANDO LLAVES PRIMARIAS
ALTER TABLE CONTRATISTA ADD PRIMARY KEY (IDE_CON)
ALTER TABLE CLIENTE ADD PRIMARY KEY(IDE_CLI)
ALTER TABLE EQUIPO ADD PRIMARY KEY(IDE_EQU)
ALTER TABLE ESTADO_EQUIPO ADD PRIMARY KEY (COD_EST)
ALTER TABLE CONTRATOALQUILER ADD PRIMARY KEY (COD_CON)
ALTER TABLE DETALLE_CONTRATO_ALQUILER ADD PRIMARY KEY (COD_CON, IDE_EQU)
ALTER TABLE TIPOMANTENIMIENTO ADD PRIMARY KEY (COD_TIP_MAN)
ALTER TABLE CONTRATOMANTENIMIENTO   ADD PRIMARY KEY (COD_CON)
ALTER TABLE DETALLE_CONTRATO_MANTENIMIENTO ADD PRIMARY KEY(COD_CON,IDE_EQU)
ALTER TABLE FICHADEDAÑOS ADD PRIMARY KEY(COD_FIC)
ALTER TABLE DETALLE_FICHA_DAÑO ADD PRIMARY KEY(COD_FIC, IDE_EQU)
ALTER TABLE TIPO_DAÑO ADD PRIMARY KEY (COD_TIP_DAÑ)
```

```
ALTER TABLE FICHARECLAMO ADD PRIMARY KEY (COD_FIC)
ALTER TABLE DETALLE_FICHA_RECLAMO ADD PRIMARY KEY (COD_FIC, IDE_EQU)
ALTER TABLE TIPO_RECLAMO ADD PRIMARY KEY (COD_TIP_REC)
ALTER TABLE TIPO_EQUIPO ADD PRIMARY KEY(COD_TIP_EQU)
ALTER TABLE FICHA_DEVOLUCION ADD PRIMARY KEY (COD_FIC)

--AGREGANDO LLAVES FORANEAS
ALTER TABLE DETALLE_CONTRATO_ALQUILER
  ADD FOREIGN KEY (COD_CON) REFERENCES CONTRATOALQUILER
ALTER TABLE DETALLE_CONTRATO_ALQUILER
  ADD FOREIGN KEY(IDE_EQU) REFERENCES EQUIPO

ALTER TABLE DETALLE_CONTRATO_MANTENIMIENTO
  ADD FOREIGN KEY (COD_CON) REFERENCES CONTRATOMANTENIMIENTO

ALTER TABLE DETALLE_FICHA_DAÑO
  ADD FOREIGN KEY (COD_FIC) REFERENCES FICHADEDAÑOS

ALTER TABLE DETALLE_FICHA_RECLAMO
  ADD FOREIGN KEY (COD_FIC) REFERENCES FICHARECLAMO

ALTER TABLE CONTRATOALQUILER
  ADD FOREIGN KEY (IDE_CLI) REFERENCES CLIENTE
ALTER TABLE CONTRATOALQUILER
  ADD FOREIGN KEY (IDE_CON) REFERENCES CONTRATISTA

ALTER TABLE CONTRATOMANTENIMIENTO
  ADD FOREIGN KEY(IDE_CLI) REFERENCES CLIENTE
ALTER TABLE CONTRATOMANTENIMIENTO
  ADD FOREIGN KEY(IDE_CON) REFERENCES CONTRATISTA

ALTER TABLE DETALLE_CONTRATO_MANTENIMIENTO
  ADD FOREIGN KEY(COD_EQU_MAN) REFERENCES TIPOMANTENIMIENTO

ALTER TABLE FICHADEDAÑOS
  ADD FOREIGN KEY(IDE_CLI) REFERENCES CLIENTE
ALTER TABLE FICHADEDAÑOS
  ADD FOREIGN KEY(IDE_CON) REFERENCES CONTRATISTA

ALTER TABLE FICHARECLAMO
  ADD FOREIGN KEY(IDE_CLI) REFERENCES CLIENTE
ALTER TABLE FICHARECLAMO
  ADD FOREIGN KEY(IDE_CON) REFERENCES CONTRATISTA

ALTER TABLE DETALLE_CONTRATO_MANTENIMIENTO
  ADD FOREIGN KEY(IDE_EQU) REFERENCES EQUIPO

ALTER TABLE DETALLE_FICHA_DAÑO
  ADD FOREIGN KEY(IDE_EQU) REFERENCES EQUIPO
ALTER TABLE DETALLE_FICHA_DAÑO
  ADD FOREIGN KEY (COD_TIP_DAÑ) REFERENCES TIPO_DAÑO
```

```
ALTER TABLE DETALLE_FICHA_RECLAMO
  ADD FOREIGN KEY(IDE_EQU) REFERENCES EQUIPO
ALTER TABLE DETALLE_FICHA_RECLAMO
  ADD FOREIGN KEY (COD_TIP_REC) REFERENCES TIPO_RECLAMO

ALTER TABLE EQUIPO
  ADD FOREIGN KEY(COD_TIP_EQU) REFERENCES TIPO_EQUIPO
ALTER TABLE EQUIPO
  ADD FOREIGN KEY (COD_EST) REFERENCES ESTADO_EQUIPO

ALTER TABLE FICHA_DEVOLUCION
  ADD FOREIGN KEY (IDE_EQU) REFERENCES EQUIPO
ALTER TABLE FICHA_DEVOLUCION
  ADD FOREIGN KEY (IDE_CLI) REFERENCES CLIENTE
ALTER TABLE FICHA_DEVOLUCION
  ADD FOREIGN KEY (IDE_CON) REFERENCES CONTRATISTA
```

- Script del registro de datos a las tablas.

```
--REGISTROS DE LA TABLA CLIENTES
INSERT INTO CLIENTE
  VALUES ("CL0001","MARIA", "PEREZ", "TORRES",
          "5440855", "12345678", "AV.UNIVERSITARIA #582")
INSERT INTO CLIENTE
  VALUES ("CL0002","JUNIOR", "FERNANDEZ", "LOPEZ",
          "5656236", "98652856", "AV.TUPAC AMARU #123")
INSERT INTO CLIENTE
  VALUES ("CL0003","ALEX", "CARDENAS", "ZARRASCO",
          "1235821", "45698712", "AV.PIMENTEL #582")
INSERT INTO CLIENTE
  VALUES ("CL0004","JUAN", "MEZQUITA", "ROSAS",
          "5023169", "14702589", "AV.OLIVOS #562")
INSERT INTO CLIENTE
  VALUES ("CL0005","ERICK", "GONZALES", "SHULLER",
          "8520365", "85236974", "AV.GAMARRA #982")
INSERT INTO CLIENTE
  VALUES ("CL0006","JOSE", "GALVEZ", "RETAMOZO",
          "9850352", "96785410", "AV.JOSE OLAYA #982")
INSERT INTO CLIENTE
  VALUES ("CL0007","CARLOS", "JARA", "SALAS",
          "1230124", "23605895", "AV.BRASIL #897")
INSERT INTO CLIENTE
  VALUES ("CL0008","EDWARD", "RAMIREZ", "ROJAS",
          "5500002", "00213589", "AV.ARGENTINA #132")
INSERT INTO CLIENTE
  VALUES ("CL0009","MANUEL", "GUIZADO", "GAMARRA",
          "5232155", "32569874", "AV.BOLIVIA #942")
INSERT INTO CLIENTE
  VALUES ("CL0010","ALBERTO", "FARFAN", "PRADO",
          "8567412", "98657452", "AV.CHILE #528")
```

```
--REGISTRO DE LA TABLA TIPO DE EQUIPO
INSERT INTO TIPO_EQUIPO VALUES("TIP001", "MONITOR")
INSERT INTO TIPO_EQUIPO VALUES("TIP002", "MOUSE")
INSERT INTO TIPO_EQUIPO VALUES("TIP003", "PARLANTES")
INSERT INTO TIPO_EQUIPO VALUES("TIP004", "TECLADO")
INSERT INTO TIPO_EQUIPO VALUES("TIP005", "CPU")
INSERT INTO TIPO_EQUIPO VALUES("TIP006", "LAPTOP")
INSERT INTO TIPO_EQUIPO VALUES("TIP007", "AUDIFONOS")
INSERT INTO TIPO_EQUIPO VALUES("TIP008", "PROYECTOR")
INSERT INTO TIPO_EQUIPO VALUES("TIP009", "IMPRESORA")
INSERT INTO TIPO_EQUIPO VALUES("TIP010", "FOTOCOPIADORA")

--REGISTRO DE LA TABLA ESTADO EQUIPO
INSERT INTO ESTADO_EQUIPO VALUES("0","DISPONIBLE")
INSERT INTO ESTADO_EQUIPO VALUES("1","OCUPADO")

--REGISTRO DE LA TABLA EQUIPO
INSERT INTO EQUIPO VALUES("EQ0001","TIP001","MONITOR DELL 1707F",200,"0")
INSERT INTO EQUIPO VALUES("EQ0002","TIP002","MOUSE HP",15,"0")
INSERT INTO EQUIPO VALUES("EQ0003","TIP003","PARLANTES LG",50,"1")
INSERT INTO EQUIPO VALUES("EQ0004","TIP004","TECLADO GENIUS",40,"0")
INSERT INTO EQUIPO VALUES("EQ0005","TIP005","CPU ASUS",250,"0")
INSERT INTO EQUIPO VALUES("EQ0006","TIP006","LAPTOP TOSHIBA",800,"1")
INSERT INTO EQUIPO VALUES("EQ0007","TIP007","AUDIFONOS SONNY",30,"0")
INSERT INTO EQUIPO VALUES("EQ0008","TIP008","PROYECTOR CANON",500,"1")
INSERT INTO EQUIPO VALUES("EQ0009","TIP009","IMPRESORA EPSON",200,"0")
INSERT INTO EQUIPO VALUES("EQ0010","TIP010","FOTOCOPIADORA RICOH",750,"0")

--REGISTRO DE LA TABLA CONTRATISTA
INSERT INTO CONTRATISTA
 VALUES ("CON001", "PEDRO", "LOPEZ", "SANCHEZ",
         "983645364","PLOPEZ@HOTMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON002", "LUIGUI", "JARAMILLO", "ORTIZ",
         "98346545","LJARAMILLO@HOTMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON003", "OSCAR", "ZAMORA", "ROSAS",
         "946566755","OZAMORA@HOTMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON004", "DICK", "GALVEZ", "CARBAJAL",
         "998765436","DGALVEZ@GMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON005", "RICHARD", "VELEZMORO", "SO",
         "945678899","RVELEZMORO@HOTMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON006", "NELSON", "MEJIA", "MEJIA",
         "923456788","NMEJIA@HOTMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON007", "TOMAS", "GONZALES", "SANCHEZ",
         "46578006","TGONZALES@GMAIL.COM")
INSERT INTO CONTRATISTA
```

```
 VALUES ("CON008", "CRISTHIAN", "HUARAC", "TORRES",
         "5976534","CHUARAC@HOTMAIL.COM")
INSERT INTO CONTRATISTA
 VALUES ("CON009", "JORGE", "CASIMIRO", "SOTO",
         "34567898","JCASIMIRO@HOTMAIL.COM")
 INSERT INTO CONTRATISTA
 VALUES ("CON010", "KEVIN", "CARLOS", "TARAZONA",
         "967895448","KCARLOS@GMAIL.COM")

--REGISTRO DE LA TABLA CONTRATO DE ALQUILER
INSERT INTO CONTRATOALQUILER
VALUES ("ABC001","CL0001","CON001","20/05/2006","01/05/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC002","CL0002","CON002","21/06/2005","02/05/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC003","CL0003","CON003","22/07/2005", "03/01/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC004","CL0004","CON004","23/09/2005", "04/02/2009","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC005","CL0005","CON005","24/05/2004", "12/03/2009","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC006","CL0006","CON006","25/04/2003", "13/02/2009","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC007","CL0007","CON007","26/03/2003", "21/10/2009","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC008","CL0008","CON008","27/02/2006", "25/11/2009","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC009","CL0009","CON009","28/01/2007", "25/12/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES ("ABC010","CL0010","CON010","29/08/2009", "05/10/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES("ABC011","CL0001","CON002","29/04/2010","05/05/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES("ABC012", "CL0001","CON002","30/04/2010","15/05/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES("ABC013", "CL0001","CON003","19/05/2010","05/06/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES("ABC014", "CL0007","CON003","09/06/2010","11/06/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES("ABC015", "CL0007","CON006","13/06/2010","05/07/2010","PRESTAMO")
INSERT INTO CONTRATOALQUILER
VALUES("ABC016","CL0007","CON006","22/07/2010","05/08/2010","PRESTAMO")

--REGISTRO DE LA TABLA DETALLE_CONTRATO_ALQUILER
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC001", "EQ0001", "20", "30")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC002", "EQ0002", "5", "21")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC003", "EQ0003", "8", "12")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC004", "EQ0004", "10", "18")
INSERT INTO DETALLE_CONTRATO_ALQUILER
```

```
  VALUES ("ABC005", "EQ0005", "12", "28")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC006", "EQ0006", "3", "29")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC007", "EQ0007", "6", "6")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC008", "EQ0008", "9", "15")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC009", "EQ0009", "7", "19")
INSERT INTO DETALLE_CONTRATO_ALQUILER
  VALUES ("ABC010", "EQ0010", "16", "31")

--REGISTRO DE LA TABLA CONTRATO MANTENIMIENTO
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC001", "CL0001", "CON001", "12/09/2010")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC002", "CL0002", "CON003", "02/05/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC003", "CL0002", "CON002", "19/05/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC004", "CL0001", "CON001", "14/08/2010")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC005", "CL0002", "CON002", "20/05/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC006", "CL0001", "CON001", "05/05/2010")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC007", "CL0001", "CON002", "17/03/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC008", "CL0003", "CON001", "12/05/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC009", "CL0001", "CON003", "29/03/2010")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC010", "CL0003", "CON001", "25/05/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC011", "CL0001", "CON002", "20/12/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC012", "CL0003", "CON001", "20/07/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC013", "CL0002", "CON002", "21/06/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC014", "CL0003", "CON003", "22/07/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC015", "CL0005", "CON005", "23/05/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC016", "CL0006", "CON006", "25/04/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC017", "CL0007", "CON007", "26/03/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC018", "CL0008", "CON008", "27/02/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC019", "CL0009", "CON009", "28/01/2009")
INSERT INTO CONTRATOMANTENIMIENTO
  VALUES ("ABC020", "CL0010", "CON010", "29/08/2008")
```

```
--REGISTRO DE LA TABLA TIPO MANTENIMIENTO
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN001", "LIMPIEZA")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN002", "CAMBIO DE PIEZAS")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN003", "CAMBIO DE PIEZAS")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN004", "LIMPIEZA")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN005", "LIMPIEZA")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN006", "LIMPIEZA")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN007", "CAMBIO DE PIEZAS")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN008", "LIMPIEZA")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN009", "LIMPIEZA")
INSERT INTO TIPOMANTENIMIENTO VALUES ("MAN010", "CAMBIO DE PIEZAS")

--REGISTRO DE LA TABLA DETALLE CONTRATO MANTENIMIENTO
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC011", "EQ0001", "MAN001")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC012", "EQ0002", "MAN002")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC013", "EQ0003", "MAN003")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC014", "EQ0004", "MAN004")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC015", "EQ0005", "MAN005")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC016", "EQ0006", "MAN006")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC017", "EQ0007", "MAN007")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC019", "EQ0008", "MAN008")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC019", "EQ0009", "MAN009")
INSERT INTO DETALLE_CONTRATO_MANTENIMIENTO
  VALUES ("ABC020", "EQ0010", "MAN010")

--REGISTRO DE LA TABLA FICHA DE DAÑO
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0001", "CON001", "CL0001", "25/05/2009")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0002", "CON002", "CL0002", "11/06/2009")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0003", "CON003", "CL0003", "18/11/2009")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0004", "CON004", "CL0004", "12/07/2009")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0005", "CON005", "CL0005", "24/02/2003")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0006", "CON006", "CL0006", "11/08/2001")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0007", "CON007", "CL0007", "02/05/2005")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0008", "CON008", "CL0008", "26/08/2006")
```

```
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0009", "CON009", "CL0009", "08/11/2008")
INSERT INTO FICHADEDAÑOS
  VALUES ("FI0010", "CON010", "CL0010", "31/12/2007")

--REGISTRO DE LA TABLA TIPO DE DAÑO
INSERT INTO TIPO_DAÑO VALUES ("X1", "NO PRENDE")
INSERT INTO TIPO_DAÑO VALUES ("X2", "SE APAGA SOLO")
INSERT INTO TIPO_DAÑO VALUES ("X3", "SE DETIENE")

--REGISTRO DE LA TABLA DETALLE FICHA DE DAÑOS
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0001", "EQ0001", "X1", "LE CAYÓ AGUA")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0002", "EQ0002", "X2", "SE CAYÓ")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0003", "EQ0003", "X3", "ESTA MAL PROGRAMADO")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0004", "EQ0004", "X1", "LE CAYÓ AGUA")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0005", "EQ0005", "X2", "SE CAYÓ")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0006", "EQ0006", "X3", "ESTA MAL PROGRAMADO")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0007", "EQ0007", "X1", "LE CAYÓ AGUA")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0008", "EQ0008", "X2", "SE CAYÓ")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0009", "EQ0009", "X3", "ESTA MAL PROGRAMADO")
INSERT INTO DETALLE_FICHA_DAÑO
  VALUES ("FI0010", "EQ0010", "X1", "LE CAYÓ AGUA")

--REGISTRO DE LA TABLA FICHA DE RECLAMO
INSERT INTO FICHARECLAMO
  VALUES ("FI0011", "CON001", "CL0001", "25/08/2009")
INSERT INTO FICHARECLAMO
  VALUES ("FI0012", "CON002", "CL0002", "13/02/2001")
INSERT INTO FICHARECLAMO
  VALUES ("FI0013", "CON003", "CL0003", "16/09/2007")
INSERT INTO FICHARECLAMO
  VALUES ("FI0014", "CON004", "CL0004", "23/08/2005")
INSERT INTO FICHARECLAMO
  VALUES ("FI0015", "CON005", "CL0005", "31/12/2008")
INSERT INTO FICHARECLAMO
  VALUES ("FI0016", "CON006", "CL0006", "26/07/2006")
INSERT INTO FICHARECLAMO
  VALUES ("FI0017", "CON007", "CL0007", "18/03/2003")
INSERT INTO FICHARECLAMO
  VALUES ("FI0018", "CON008", "CL0008", "01/01/2005")
INSERT INTO FICHARECLAMO
  VALUES ("FI0019", "CON009", "CL0009", "14/02/2008")
INSERT INTO FICHARECLAMO
  VALUES ("FI0020", "CON010", "CL0010", "20/05/2007")
```

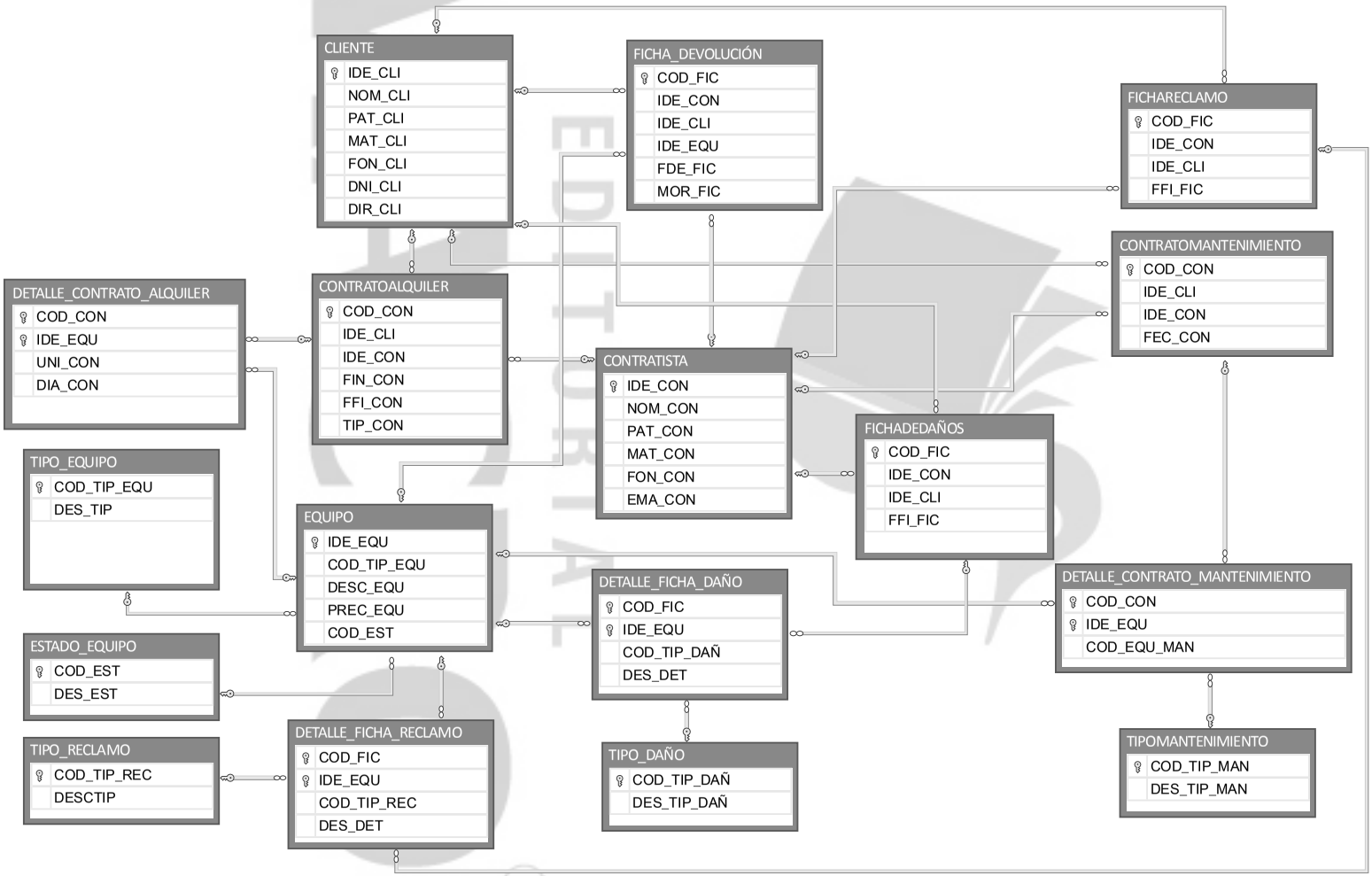
```
--REGISTRO DE LA TABLA TIPO DE RECLAMO
INSERT INTO TIPO_RECLAMO VALUES ("Y1", "DEVOLUCION DE DINERO")
INSERT INTO TIPO_RECLAMO VALUES ("Y2", "EXIGE REPARAR PRODUCTO")

--REGISTRO DE LA TABLA DETALLE FICHA DE RECLAMO
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0011", "EQ0001", "Y1", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0012", "EQ0002", "Y2", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0013", "EQ0003", "Y1", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0014", "EQ0004", "Y2", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0015", "EQ0005", "Y2", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0016", "EQ0006", "Y2", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0017", "EQ0007", "Y1", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0018", "EQ0008", "Y1", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0019", "EQ0009", "Y2", "NO PRENDE")
INSERT INTO DETALLE_FICHA_RECLAMO
  VALUES ("FI0020", "EQ0010", "Y1", "NO PRENDE")

--REGISTRO DE LA TABLA FICHA DE DEVOLUCION
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0010", "CON001", "CL0001", "EQ0002", "25/05/2009", 70)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0011", "CON001", "CL0010", "EQ0001", "15/08/2009", 50)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0012", "CON002", "CL0009", "EQ0002", "05/06/2009", 100)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0013", "CON002", "CL0001", "EQ0001", "12/05/2009", 70)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0014", "CON001", "CL0009", "EQ0008", "11/07/2009", 10)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0015", "CON003", "CL0007", "EQ0003", "14/05/2009", 100)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0016", "CON003", "CL0005", "EQ0001", "18/08/2009", 30)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0017", "CON001", "CL0003", "EQ0002", "22/08/2009", 50)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0018", "CON003", "CL0001", "EQ0004", "29/06/2009", 10)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0019", "CON001", "CL0002", "EQ0001", "20/08/2009", 100)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0020", "CON004", "CL0001", "EQ0005", "05/08/2009", 20)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0021", "CON001", "CL0002", "EQ0001", "25/08/2009", 30)
INSERT INTO FICHA_DEVOLUCION
```

```
  VALUES ("FI0022", "CON002", "CL0002", "EQ0002", "12/08/2007", 50)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0023", "CON003", "CL0003", "EQ0003", "26/12/2009", 70)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0024", "CON004", "CL0004", "EQ0004", "16/11/2009", 80)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0025", "CON005", "CL0005", "EQ0005", "28/05/2010", 85)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0026", "CON006", "CL0001", "EQ0006", "31/01/2010", 120)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0027", "CON007", "CL0007", "EQ0007", "24/08/2010", 45)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0028", "CON008", "CL0008", "EQ0008", "19/09/2009", 20)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0029", "CON009", "CL0009", "EQ0009", "03/12/2010", 10)
INSERT INTO FICHA_DEVOLUCION
  VALUES ("FI0030", "CON010", "CL0010", "EQ0010", "05/04/2010", 150)
GO
```

- Script de las consultas realizadas a las tablas.

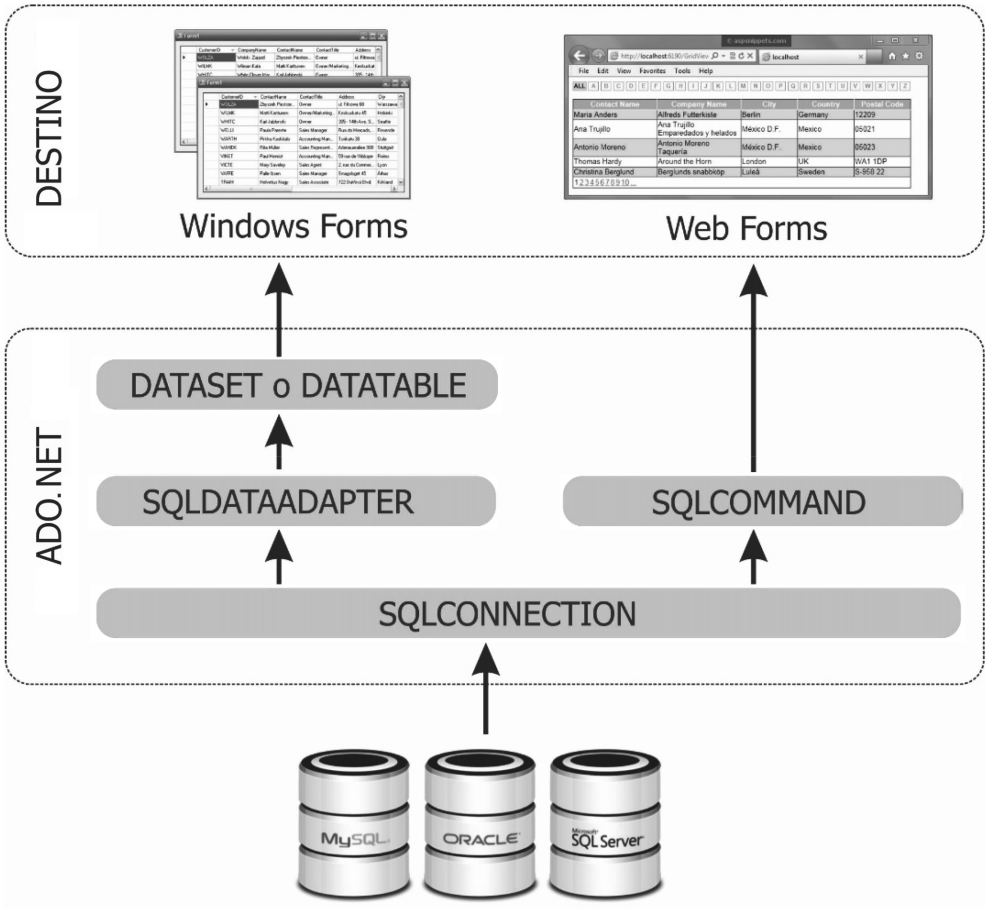
```
SELECT * FROM CONTRATISTA
SELECT * FROM CLIENTE
SELECT * FROM CONTRATOALQUILER
SELECT * FROM CONTRATOMANTENIMIENTO
SELECT * FROM DETALLE_CONTRATO_ALQUILER
SELECT * FROM DETALLE_CONTRATO_MANTENIMIENTO
SELECT * FROM DETALLE_FICHA_DAÑO
SELECT * FROM DETALLE_FICHA_RECLAMO
SELECT * FROM EQUIPO
SELECT * FROM ESTADO_EQUIPO
SELECT * FROM FICHA_DEVOLUCION
SELECT * FROM FICHADEDAÑOS
SELECT * FROM FICHARECLAMO
SELECT * FROM TIPO_DAÑO
SELECT * FROM TIPO_EQUIPO
SELECT * FROM TIPO_RECLAMO
```

4.2 Diagrama de base de datos Contrato



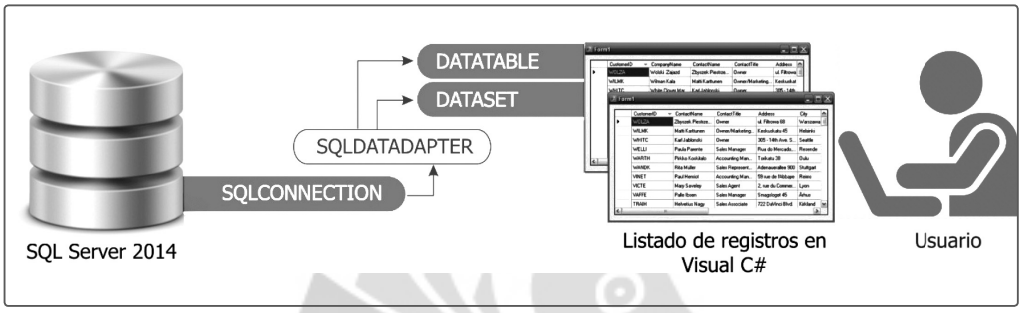
4.3 Introducción al ADO.NET 4.6

ADO.NET (ActiveX Data Objects) representa un conjunto de objetos de acceso a datos, que ofrecen servicios de conectividad para aplicaciones de desarrollo en Visual C#. Asimismo, constituye una parte integral de Framework, ya que proporciona acceso a datos relacionales y de aplicaciones.

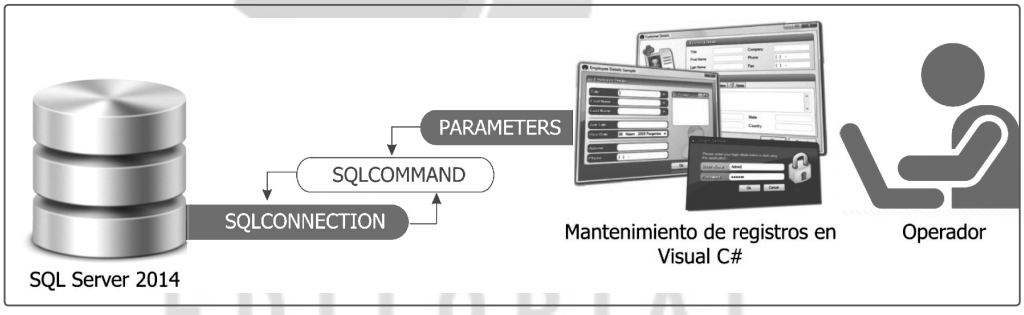


ADO.NET permite tener un acceso sencillo y confiable a orígenes de datos como SQL Server, así como a orígenes de datos expuestos mediante OLE DB y ODBC. Es a partir de esto que se puede tener acceso a casi todos los sistemas de gestión de base de datos como Oracle, Access o Excel. Las aplicaciones de desarrollo que necesitan mostrar información de dichos gestores pueden utilizar las clases ADO.NET para conectar a estos orígenes y así permitir recuperar, registrar, modificar o eliminar dicha información.

Observe cómo se usan las clases para ADO.NET con el fin de listar información desde un gestor de base de datos como SQL Server:



Observe cómo se usan las clases para ADO.NET con el objetivo de manipular la información (registrar, modificar o eliminar) desde un gestor de base de datos como SQL Server:



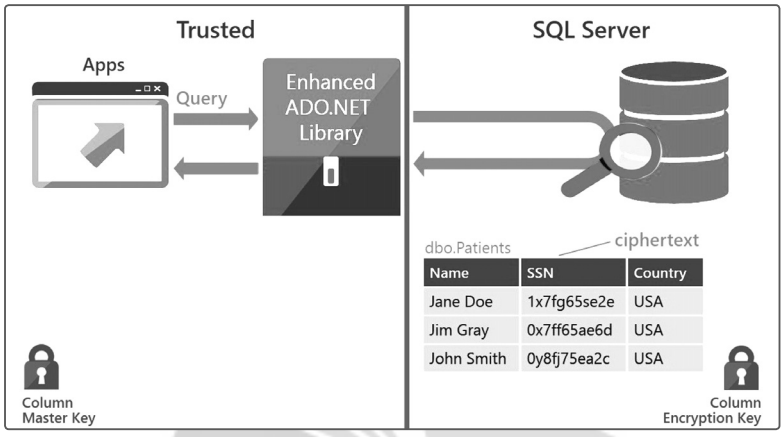
4.4 Novedades de ADO.NET 4.6

ADO.NET ahora es compatible con la característica Always Encrypted disponible en SQL Server 2016. Con Always Encrypted, SQL Server puede realizar operaciones en los datos cifrados y, lo mejor de todo, es que la clave de cifrado reside, junto con la aplicación, en el entorno de confianza del cliente y no en el servidor.

Always Encrypted protege los datos de la máquina cliente para que los administradores de bases de datos no tengan acceso a los datos de texto sin formato. El cifrado y descifrado de datos ocurre de forma transparente en el nivel de controlador, lo que minimiza los cambios que deben realizarse en las aplicaciones existentes.

La característica principal del Always Encrypted está basada en tecnología de Microsoft Research y ayuda a proteger los datos en reposo y en movimiento. Al utilizar Always Encrypted, SQL Server puede desempeñar operaciones en datos encriptados y lo principal es que la llave de encriptación reside en la aplicación en el ambiente de confianza de la máquina cliente.

EDITORIAL MACRO®

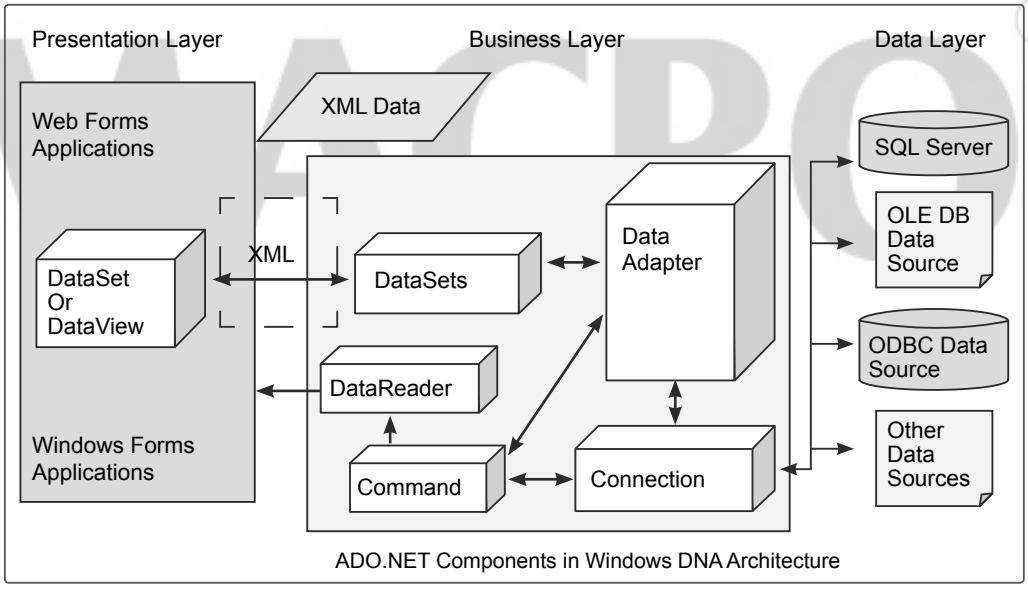


Fuente: http://mscorp.blob.core.windows.net/mscorpmedia/2015/05/Always_Encrypted-graphic.jpg

4.5 Arquitectura de ADO.NET

ADO.NET aprovecha al máximo la eficacia de los archivos de tipo XML para proporcionar un excelente acceso a datos sin mantener una conexión a la base de datos abierta. Ambas tecnologías convergen en el objeto DataSet, el cual obtiene información proveniente del archivo XML dejando de lado el proveedor de datos y concentrando más en la información que se presentará al cliente.

Asimismo, se debe tener en cuenta que los componentes de ADO.NET están diseñados para separar el acceso a datos de la manipulación de la información. Es decir, existe una clase especializada para la conexión al servidor y otras para la administración de la información.



Fuente: <http://www.c-sharpcorner.com/uploadfile/mahesh/ado.netfromwindowsdnaperspective11302005001902am/ado.netfromwindowsdnaperspective.aspx>

4.6 Componentes de ADO.NET

Los componentes de ADO.NET han sido diseñados para acceder a la base de datos del servidor, así como manipular la información contenida en dicha base de datos. Entonces los componentes son:

- **Proveedores de datos**

  Los cuales proporcionan clases especializadas para el acceso y manipulación de la información desde una base de datos. Se tiene las clases SqlConnection, SqlCommand, SqlDataAdapter y DataReader.

- **DataSet**

  El cual recibe el modelo relacional de la base de datos, desconectándose así de la base de datos y manteniendo información en memoria.

4.7 Proveedores de datos

Los proveedores de datos .NET Framework sirven para conectarse a una determinada base de datos, que podría ser SQL Server, Oracle, etc., y permiten ejecutar comandos y recuperar resultados. Asimismo, son considerados ligeros, de manera que crean un nivel mínimo entre el origen de datos y el código, con lo que aumenta el rendimiento sin sacrificar funcionalidad.

Los principales proveedores son:

Framework	Descripción
SQL Server	Proporciona acceso a datos a SQL Server de cualquier versión. Se usa el espacio de trabajo: `System.Data.SqlClient`
OLE DB	Proporciona acceso a datos para orígenes de tipo OLE DB, como un archivo de texto y un correo electrónico. Se usa el espacio de trabajo: `System.Data.OleDb`
ODBC	Proporciona acceso a datos que se exponen mediante ODBC como el mismo SQL Server, Oracle, FoxPro y Access. Se usa el espacio de trabajo: `System.Data.Odbc`
Oracle	Proporciona acceso a orígenes de datos de Oracle. Se usa el espacio de trabajo: `System.Data.OracleClient`
EntityClient	Proporciona acceso a datos para las aplicaciones de Entity Data Model (EDM). Se usa el espacio de trabajo: `System.Data.EntityClient`

4.8 Objetos principales de los proveedores de datos

Existen cuatro clases, las que compone un entorno de desarrollo a nivel cliente-servidor. Cada una de estas clases, tiene un objetivo específico dentro de este desarrollo.

- **Objeto Connection**

  Establece una conexión a un origen de datos determinado. Observe el objeto de la clase SQLConnection, que permita conectarse a una base de datos en Sql Server:

```
SqlConnection cn = new SqlConnection("server=192.168.1.10;
                                database=contratos; uid=sa;pwd=sql");
```

- **Objeto DataAdapter**

  Llena un DataSet o DataTable y realiza las actualizaciones necesarias en el origen de datos. Observe el código que permite llenar un DataSet desde una conexión activa:

```
DataTable dt = new DataTable();
SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM CLIENTE",cn);
Da.fill(dt);
```

- **Objeto Command**

  Ejecuta un comando en un origen de datos. Es a partir de este que se puede obtener resultados escalados, inserciones, actualizaciones o eliminación de registros. Observe el código para agregar un nuevo registro de cliente sin especificación de parámetros:

```
SqlCommand cmd = new SqlCommand("SP_NUEVOCLIENTE",cn);
cmd.ExecuteNonQuery();
```

- **Objeto DataReader**

  Lee un flujo de datos de solo avance y solo lectura desde un origen de datos. Esto permitirá llenar controles como cuadros combinados o listas desde una conexión activa. Observe el código sobre cómo llenar un cuadro combinado desde la tabla Cliente:

```
SqlCommand cmd = new SqlCommand("SELECT * FROM CLIENTE",cn);
cn.Open();
SqlDataReader dr = cmd.ExecuteReader();
while (dr.Read())
{
   cboCliente.Items.Add(dr[1]);
}
```

EDITORIAL MACRO®

ADO.NET adiciona clases a los cuatro anteriores solo como un control de datos mucho más especializado, ya que el proveedor de datos así lo permite.

- **Transaction**

  Incluye comandos en las transacciones que se realizan en el origen de datos. Desde aquí se puede controlar la información que genera los procesos y así poder asegurar la información, así como anularla.

- **Parameter**

  Define los parámetros de entrada, salida y valores devueltos para los comandos y los procedimientos almacenados. Desde aquí se puede enviar los parámetros que se necesitan para el registro, la actualización o la eliminación de datos.

- **Exception**

  Se devuelve cuando se detecta un error en el origen de datos. En caso de que el error se detecte en el cliente, los proveedores de datos .NET Framework generan una excepción de .NET Framework. La clase base para todos los objetos Exception es DbException.

4.9 Espacio de nombre System.Data.SqlClient

Describe una colección de clases que se utiliza para obtener acceso a una base de datos de SQL Server en el espacio administrado. Observe las principales clases que componen el espacio de trabajo:

- **SqlCommand**

  Representa un procedimiento almacenado o una instrucción de Transact-SQL que se ejecuta en una base de datos de SQL Server como una sentencia escalada inserción, actualización o eliminación.

- **SqlCommandBuilder**

  Representa un procedimiento almacenado o una instrucción de Transact-SQL que se ejecuta de forma desconectada al SQL Server.

- **SqlConnection**

  Representa una conexión abierta a una base de datos de SQL Server.

- **SqlDataAdapter**

  Representa un conjunto de comandos de datos y una conexión de base de datos que se utilizan para rellenar un DataSet.

- **SqlDataReader**

  Proporciona una forma de leer una secuencia de filas solo hacia adelante en una base de datos de SQL Server.

EDITORIAL MACRO®

- **SqlException**

  Representa la excepción que se produce cuando SQL Server devuelve una advertencia o un error.

- **SqlTransaction**

  Representa una transacción de Transact-SQL que se realiza en una base de datos de SQL Server.

4.10 Espacio de nombre System.Data

El espacio de nombres System.Data proporciona acceso a las clases que representan la arquitectura de ADO.NET. Desde aquí se podrán compilar componentes que administran eficazmente los datos de varios orígenes de datos. Observe las principales clases que componen el espacio de trabajo:

- **DataColumn**

  Representa una columna especificada en un DataTable.

- **DataRow**

  Representa una fila de datos especificada en un DataTable.

- **DataSet**

  Representa una caché de memoria interna de datos recuperados de un origen de datos.

- **DataTable**

  Representa una tabla almacenada en un espacio de memoria desde un origen de datos.

- **DataView**

  Representa una vista personalizada que se puede enlazar a datos de un DataTable para ordenarlos, buscarlos, etc.

4.11 Clase SQLConnection

Contiene métodos y propiedades responsables de la conexión a una base de datos efectiva, además de representar la interacción entre las sentencias enviadas al servidor, así como las que se reciben.

Un objeto de la clase SqlConnection guarda datos del servidor SQL, al cual se desea conectar. Luego, podrán ser abiertos para ejecutar sentencias o cerrarlas cuando se termine un proceso.

EDITORIAL MACRO®

Método constructor:

Formato	Descripción

SqlConnection()	Inicializa un objeto de la clase SqlConnection vacía, de tal forma que, mediante propiedades, pueda configurarse la conexión al servidor de datos. El siguiente código muestra la configuración vacía:
	`SqlConnection cn = new SqlConnection();`
	Y mediante la propiedad ConnectionString se puede configurar la conexión a una base de datos contrato del Sql Server, por ejemplo:
	`cn.ConnectionString = "server=USER-PC; database=contrato; integrated security=SSPI";`

SqlConnection(cadena)	Inicializa con una cadena de conexión directamente a un objeto de la clase SqlConnection. El siguiente código muestra la configuración a la base de datos contrato del Sql Server: `SqlConnection cn = new SqlConnection("server=USER-PC; database=contrato; integrated security=SSPI");`

En ambas especificaciones se necesita especificar los siguientes parámetros:

- **Nombre del servidor**

  Aquí se especifica el nombre del servidor al cual se desea conectar. Para este caso, se concentrará en el servidor Sql Server. Observe una lista de posibles especificaciones:

Especificación	Descripción

`Data Source = (local);`
`Data Source = .;`
`Server = (local);`
`Server = .;`
	Especificando un nombre de servidor por defecto.

`Data Source = 192.168.1.10;` `Server = 192.168.1.10;`	Especificando un nombre del servidor con número de IP.
`Data Source = SOCRATES;` `Server = SOCRATES;`	Especificando un nombre de servidor configurado a nivel de red.

Hay que tener en cuenta que estas especificaciones no son sensibles a mayúsculas o minúsculas, es decir, dará igual especificarlo en mayúsculas o minúsculas.

EDITORIAL MACRO®

- **Nombre de la base de datos**

  Aquí se especifica el nombre de la base de datos de donde provienen los datos. Hay que tener en cuenta que la base de datos debe encontrarse dentro del servidor especificado en el nombre del servidor.

Especificación	Descripción
`Initial Catalog = Contrato;`	Especificando la base de datos con el catálogo inicial.
`Database = Contrato;`	Especificando la base de datos con Database.

- **Especificación de seguridad**

  Aquí se especifica el modo de seguridad que se tiene frente al servidor, ya que desde allí se obtendrá las credenciales de acceso a la base de datos.

Especificación	Descripción
`Integrated Security = SSPI;` `Integrated Security = True;`	Especificación para el modo de autenticación Windows.
`User Id=sa; Password=sql;` `Uid=sa; Pwd=sql;`	Especificación para el modo de autenticación Sql Server.

Propiedad:

Propiedad	Descripción

ConnectionString	Establece la cadena de conexión usada para abrir una base de datos en Sql Server.
	`SqlConnection cn = new SqlConnection();`
	`cn.ConnectionString = "server=USER-PC;`
	`database=contrato;`
	`integrated security=SSPI";`

Métodos:

Método	Descripción

Close	Cierra la conexión mantenida con el servidor de base de datos.
	`SqlConnection cn = new SqlConnection();`
	`cn.ConnectionString = "server=USER-PC;`
	`database=contrato;`
	`integrated security=SSPI";`
	`cn.close();`

Dispose	Libera los recursos usados en una conexión de base de datos.
	`SqlConnection cn = new SqlConnection();`
	`cn.ConnectionString = "server=USER-PC;`
	`database=contrato;`
	`integrated security=SSPI";`
	`cn.dispose();`

Open	Abre la conexión especificada en la propiedad ConnectionString.
	`SqlConnection cn = new SqlConnection();`
	`cn.ConnectionString = "server=USER-PC;`
	`database=contrato;`
	`integrated security=SSPI";`
	`cn.open();`

4.12 Definición de la cadena de conexión

Observe las diferentes formas de implementar la cadena de conexión en una aplicación. En todos los casos se deben colocar las siguientes librerías:

```
using System.Data;
using System.Data.SqlClient;
```

- **En la misma aplicación:**

```
SqlConnection cn = new SqlConnection();
cn.ConnectionString = "server=USER-PC;
                       database=contrato;
                       integrated security=SSPI";

SqlCommand cmd = new SqlCommand("select * from cliente",cn);
cn.Open();
```

Se inicia creando el objeto de la clase SqlConnection llamado cn. Luego, se le asigna el parámetro de conexión mediante la propiedad ConnectionString.

- **En una clase:**

```
public class Conexion
{
   public SqlConnection getConecta()
   {
     SqlConnection cn = new SqlConnection("server=USER-PC;
                                          database=contrato;
                                          integrated security=SSPI");
     return cn;
   }
}
```

Se implementa el método getConecta con el tipo de salida SqlConnection, ya que el método debe devolver la conexión al objeto que lo invoque.

Luego, se debe referenciar mediante el siguiente código:

```
Conexion objcon = new Conexion();
SqlConnection cn = objcon.getConecta();

SqlCommand cmd = new SqlCommand("select * from cliente",cn);
cn.Open();
```

Se inicia declarando el objeto de la clase Conexión. Luego, se crea e inicializa el objeto cn de la clase SqlConnection con la invocación al método getConecta de la clase Conexión.

- **En un archivo de configuración:**

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="cn" connectionString="server=USER-PC;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

4.13 Clase SQLDataAdapter

Contiene propiedades y métodos que permiten crear un puente entre las sentencias SQL que necesita una aplicación y SQL Server. Normalmente, se usa para recuperar información o realizar inserciones, actualizaciones o eliminaciones sobre los registros de una tabla de base de datos.

Para recuperar información, se usará el método fill para llenar a un objeto de la clase DataSet o DataTable. Mientras que para grabar, actualizar o eliminar, se debe usar un objeto de la clase SqlCommand.

Métodos constructores:

Formato	Descripción

SqlDataAdapter()	Inicializa un objeto de la clase SqlDataAdapter en vacío y que mediante sus propiedades se podrá gestionar la ejecución de sentencias SQL a partir de una conexión activa.
	`SqlDataAdapter da = new SqlDataAdapter();`
	Por ejemplo, si se necesita listar los registros de los clientes en un control DataGridView, el código sería como sigue:
	`//Creando la cadena de conexión`
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	`//Creando el objeto de la clase SQLDataAdapter`
	`SqlDataAdapter da = new SqlDataAdapter();`
	`da.SelectCommand = new SqlCommand();`
	`da.SelectCommand.Connection = cn;`
	`da.SelectCommand.CommandText = "SP_LISTACLIENTE";`
	`//Enviando la información al DataTable`
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	`dgClientes.DataSource = dt;`

SqlDataAdapter (Sentencia, Cadena)	Inicializa un objeto de la clase SqlDataAdapter tomando como parámetro la sentencia SQL, la cual puede ser una sentencia SELECT o un procedimiento almacenado y la cadena de conexión.
	`SqlDataAdapter da = new SqlDataAdapter("sentencia",cadena);`
	Por ejemplo, si se necesita listar los registros de los clientes en un control DataGridView, el código sería como sigue:
	`//Creando la cadena de conexión`
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	`//Creando el objeto de la clase SQLDataAdapter`
	`SqlDataAdapter da = new`
	`SqlDataAdapter("SP_LISTACLIENTE",cn);`
	`//Enviando la informacion al DataTable`
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	`dgClientes.DataSource = dt;`

SqlDataAdapter (SqlCommand)	Inicializa un objeto de la clase SqlDataAdapter tomando como parámetro un objeto de la clase SQLCommand.
	`SqlDataAdapter da = new SqlDataAdapter(cmd);`
	Por ejemplo, si se necesita listar los registros de los clientes en un control DataGridView, el código sería como sigue:
	`//Creando la cadena de conexión`
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	`//Creando el objeto de la clase cmd`
	`SqlCommand cmd = new SqlCommand();`
	`cmd.Connection = cn;`
	`cmd.CommandText = "SP_LISTACLIENTE";`
	`//Creando el objeto de la clase SQLDataAdapter`
	`SqlDataAdapter da = new SqlDataAdapter(cmd);`
	`//Enviando la informacion al DataTable`
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	`dgClientes.DataSource = dt;`

Principales propiedades:

Propiedad	Descripción

DeleteCommand	Establece una sentencia SQL o procedimiento almacenado para eliminar un registro de una tabla en SQL Server.
	`//Creando la cadena de conexión`
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	`//Creando el objeto de la clase cmd`
	`SqlCommand cmd = new SqlCommand();`
	`cmd.Connection = cn;`
	`cmd.CommandText = "SP_ELIMINACLIENTE";`
	`//Agregando parámetros`
	`cmd.Parameters.Add("@cod",SqlDbType.Char).Value = codigo;`
	`da.DeleteCommand = cmd;`

InsertCommand	Establece una sentencia SQL o procedimiento almacenado para agregar un registro de una tabla en SQL Server.
	
	`//Creando la cadena de conexión`
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	
	`//Creando el objeto de la clase cmd`
	`SqlCommand cmd = new SqlCommand();`
	`cmd.Connection = cn;`
	`cmd.CommandText = "SP_AGREGACLIENTE";`
	
	`//Agregando parámetros`
	`cmd.Parameters.Add("@cod",SqlDbType.Char).Value = codigo;`
	`cmd.Parameters.Add("@nom",SqlDbType.Char).Value = nombres;`
	`cmd.Parameters.Add("@dir",SqlDbType.Char).Value = direccion;`
	`cmd.Parameters.Add("@tel",SqlDbType.Char).Value = telefono;`
	
	`da.InsertCommand = cmd;`

SelectCommand	Establece una sentencia SQL o procedimiento almacenado para listar uno o más registros de una tabla en SQL Server.
	
	`//Creando la cadena de conexión`
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	
	`//Creando el objeto de la clase cmd`
	`SqlCommand cmd = new SqlCommand();`
	`cmd.Connection = cn;`
	`cmd.CommandText = "SP_LISTACLIENTE";`
	
	`//Creando el objeto de la clase SQLDataAdapter`
	`SqlDataAdapter da = new SqlDataAdapter();`
	`da.SelectCommand(cmd);`
	
	`//Enviando la informacion al DataTable`
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	`dgClientes.DataSource = dt;`

UpdateCommand	Establece una sentencia SQL o procedimiento almacenado para actualizar un registro de una tabla en SQL Server.

```
//Creando la cadena de conexión
SqlConnection cn = new SqlConnection("server=.;
                          database=contrato;
                          integrated security=SSPI");

//Creando el objeto de la clase cmd
SqlCommand cmd = new SqlCommand();
cmd.Connection = cn;
cmd.CommandText = "SP_ACTUALIZACLIENTE";

//Agregando parámetros
cmd.Parameters.Add("@cod",SqlDbType.Char).Value = codigo;
cmd.Parameters.Add("@nom",SqlDbType.Char).Value = nombres;
cmd.Parameters.Add("@dir",SqlDbType.Char).Value = direccion;
cmd.Parameters.Add("@tel",SqlDbType.Char).Value = telefono;

da.UpdateCommand = cmd;
```

Métodos:

Dispose	Permite liberar los recursos usados en la implementación del SqlDataAdapter. La sentencia para liberar el objeto SqlDataAdapter da.dispose();
Fill	Permite agregar filas a un objeto datatable o dataset. Rellenar un datatable da.fill(dt). Rellenar un DataSet da.fill(ds,"Tabla").
ToString	Permite devolver el nombre del objeto de la clase SqlDataAdapter.

4.14 ConfigurationManager

Proporciona acceso a los archivos de configuración para las aplicaciones cliente. Se debe tener en cuenta que se debe agregar la librería al proyecto realizando los siguientes pasos:

1. Clic derecho sobre el proyecto.
2. Seleccione **Agregar** > **Referencia**.
3. Active el check System.Configuration.

Librería:

System.Configuration	La librería se especificará solo en la clase donde se implementa la cadena de conexión.

MACRO®

Propiedad:

ConnectionStrings	Especifica la conexión desde un pool de conexión implementado en el archivo de configuraciones app.config. `ConfigurationManager.ConnectionStrings["cn"].ConnectionString` Donde «cn» es el nombre de la cadena de conexión especificada en el archivo app.config.

Observe una implementación completa hacia la base de datos CONTRATO en SQL Server:

Pasos:

1. En el archivo de configuración app.config agregue el siguiente código:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

Considere que la cadena de conexión se especifica en ConnectionString y que este debe tener un nombre especificado en «name»; es este mismo nombre el cual se hará referencia en el método de conexión.

2. Luego, prepare el método de conexión con el siguiente código:

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

//Librerias ADO NET
using System.Data.SqlClient;
using System.Configuration;

namespace pjConexion
{
```

```
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            sqlconnection cn = new sqlconnection(configurationmanager.
            connectionstrings["cn"].connectionstring);
            return cn;
        }
    }
}
```

Tener en cuenta que primero debe agregar la referencia System.Configuration al proyecto. Para esto, presione clic derecho sobre el proyecto y luego **Agregar** > **Referencia**. Después, en la clase donde se implementa el método, agregue la librería using System.Configuration. Finalmente, se crea un objeto de la clase SqlConnection, en la cual se especifica en la propiedad connectionStrings.

4.15 Clase DataSet

Un objeto de la clase DataSet representa un conjunto de datos residente en memoria, el cual proporciona un modelo de programación relacional totalmente independiente del origen de datos. Eso quiere decir que el modo de trabajo sería desconectado de la data.

Método constructor:

DataSet	Inicializa el objeto de la clase DataSet en vacío y que a partir de un objeto SqlDataAdapter permite rellenar el DataSet vacío. Por ejemplo, si se quiere rellenar el dataset de clientes, sería de la siguiente manera:

```
//Creando la cadena de conexión
SqlConnection cn = new SqlConnection("server=.;
                        database=contrato;
                        integrated security=SSPI");

//Creando el objeto de la clase SQLDataAdapter
SqlDataAdapter da = new
                SqlDataAdapter("SP_LISTACLIENTE",cn);

//Enviando la información al DataSet
DataSet ds = new DataSet();
da.Fill(ds, "Clientes");
```

Propiedad:

Tables	Aquí se especifica el nombre de la tabla del cual necesita mostrar sus registros. Se debe tener en cuenta que dicho nombre fue asignado por la propiedad fill del objeto SQLDataAdapter. Por ejemplo, si se quiere rellenar el dataset de clientes, sería de la siguiente manera:

```
//Creando la cadena de conexión
SqlConnection cn = new SqlConnection("server=.;
                            database=contrato;
                            integrated security=SSPI");

//Creando el objeto de la clase SQLDataAdapter
SqlDataAdapter da = new
                  SqlDataAdapter("SP_LISTACLIENTE",cn);

//Enviando la informacion al DataSet
DataSet ds = new DataSet();
da.Fill(ds, "Clientes");
dgClientes.DataSource = ds.Tables["Clientes"];
```

Métodos:

Dispose	Libera todos los recursos asignados al objeto de la clase DataSet. El código de liberación es ds.dispose().
ToString	Devuelve el nombre del objeto al cual pertenece el DataSet.

Observe una implementación completa para mostrar los registros de los clientes:

Pasos:

1. Agregue las siguientes librerías:

```
using System.Data;
using System.Data.SqlClient;
```

2. Implemente un método de conexión:

```
SqlConnection getConecta()
{
    SqlConnection cn = new SqlConnection("server=.;
                                         database=contrato;
                                         integrated security=SSPI");
    return cn;
}
```

EDITORIAL MACRO®

3. Implemente un método que liste los registros de los clientes. Aquí se podría tener dos posibilidades de listado: la primera sería especificar la consulta directamente y la otra por medio de un procedimiento almacenado:

Consulta:

```
DataSet listaClientes()
{
   SqlConnection cn = getConecta();
   SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM CLIENTE",cn);
   DataSet ds = new DataSet();
   da.Fill(ds, "CLIENTES");
   return ds;
}
```

Procedimiento almacenado:

```
DataSet listaClientes()
{
   SqlConnection cn = getConecta();
   SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTE",cn);
   DataSet ds = new DataSet();
   da.Fill(ds, "CLIENTES");
   return ds;
}
```

4. Finalmente, asigne el resultado al control DataGridView.

```
dgClientes.DataSource = listaClientes().Tables["CLIENTES"];
```

4.16 Clase DataTable

Un objeto de la clase DataTable representa una determinada tabla en memoria obtenida desde un relleno realizado con el objeto de la clase SqlDataAdapter, de modo que se pueda interactuar con ella y enviarlo a controles que permitan visualizar sus registros. La diferencia que tiene con el DataSet es que no necesita una asignación de nombre.

Método constructor:

DataTable	Inicializa el objeto de la clase DataTable en vacío y que a partir de un objeto SqlDataAdapter permite rellenar el DataTable vacío. Por ejemplo, si se quiere rellenar el DataTable de clientes, sería de la siguiente manera:

```
//Creando la cadena de conexión
SqlConnection cn = new SqlConnection("server=.;
                            database=contrato;
                            integrated security=SSPI");

//Creando el objeto de la clase SQLDataAdapter
SqlDataAdapter da = new
                  SqlDataAdapter("SP_LISTACLIENTE",cn);

//Enviando la informacion al DataTable
DataTable dt = new DataTable();
da.Fill(dt);
```

Métodos:

Dispose	Libera todos los recursos asignados al objeto de la clase DataTable. El código de liberación es dt.dispose().
ToString	Devuelve el nombre del objeto al cual pertenece el DataTable.

Observe una implementación completa para mostrar los registros de los clientes:

Pasos:

1. Agregue las siguientes librerías:

```
using System.Data;
using System.Data.SqlClient;
```

2. Implemente un método de conexión.

```
SqlConnection getConecta()
{
    SqlConnection cn = new SqlConnection("server=.;
                                         database=contrato;
                                         integrated security=SSPI");
    return cn;
}
```

EDITORIAL MACRO®

3. Implemente un método que liste los registros de los clientes. Aquí se podría tener dos posibilidades de listado: la primera sería especificar la consulta directamente y la otra por medio de un procedimiento almacenado.

Consulta:

```
DataTable listaClientes()
{
   SqlConnection cn = getConecta();
   SqlDataAdapter da = new SqlDataAdapter("SELECT * FROM CLIENTE",cn);
   DataTable dt = new DataTable();
   da.Fill(dt);
   return dt;
}
```

Procedimiento almacenado:

```
DataTable listaClientes()
{
   SqlConnection cn = getConecta();
   SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTE",cn);
   DataTable dt = new DataTable();
   da.Fill(dt);
   return dt;
}
```

4. Finalmente, asigne el resultado al control DataGridView.

```
dgClientes.DataSource = listaClientes();
```

4.17 Clase DataView

Un objeto de tipo DataView permite representar a un subconjunto de datos obtenidos a partir de un DataSet o DataTable. Por medio de propiedades y métodos, se puede administrar la información contenida en un DataView.

EDITORIAL MACRO®

Métodos constructores:

DataView()	Inicializa con un valor vacío el objeto de la clase DataView.
	`DataView dv = new DataView();`
	Si se necesita asociar un DataSet con un DataView, se usa el siguiente código:
	`DataSet ds = new DataSet();`
	`da.Fill(ds);`
	`DataView dv = new DataView();`
	`dv.Table = ds.Tables[0];`
	`dgClientes.DataSource = dv;`

DataView (DataTable)	Inicializa con un conjunto de datos de tipo DataTable el objeto de la clase DataView.
	`DataView dv = new DataView(DataTable);`
	Si se necesita asociar un DataTable con un DataView, se usa el siguiente codigo:
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	`DataView dv = new DataView(dt);`
	`dgClientes.DataSource = dv;`

EDITORIAL ®

Propiedades:

Count	Devuelve el número total de registros almacenados en un DataView solo después de aplicar la propiedad RowFilter. Si se quiere mostrar el total de clientes asociados a un objeto DataView, se podría usar el siguiente código:
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	`DataView dv = new DataView(dt);`
	`dgClientes.DataSource = dv;`
	`MessageBox.Show("Total de registros: "+dv.Count.ToString());`

RowFilter	Establece una expresión que resultará como filtro de registros contenidos en un DataView. Por ejemplo, si se quiere mostrar el registro del cliente, cuyo código es CL0002, el código sería como sigue:
	`DataTable dt = new DataTable();`
	`da.Fill(dt);`
	
	`DataView dv = new DataView(dt);`
	`dv.RowFilter = "IDE_CLI ="CL0002"";`
	`dgClientes.DataSource = dv;`

Sort	Permite ordenar los registros contenidos en un objeto DataView. Este se especificará según las columnas contenidas en el DataView. Por ejemplo, si se quiere mostrar los registros de los clientes ordenados en forma ascendente por el apellido paterno del cliente, el código es como sigue:
	`DataSet ds = new DataSet();`
	`da.Fill(ds);`
	
	`DataView dv = new DataView();`
	`dv.Table = ds.Tables[0];`
	`dv.Sort = "PAT_CLI";`
	`dgClientes.DataSource = dv;`

4.18 Instrucción Using

En C#, la palabra clave Using tiene dos formas de uso: la primera es considerada como directiva, ya que permite definir las librerías en una aplicación, y la segunda se considera como definición de ámbito de un determinado objeto, el cual lo destruye al finalizar el ámbito de Using.

Observe las siguientes posibilidades de uso de Using:

Importando espacio de nombres	En este caso, Using actúa como directiva del compilador, haciendo que la aplicación use tipos definidos en el espacio de nombre, sin necesidad de especificarlo directamente en el código. Por ejemplo, para usar las clases ADO.NET, se necesita la siguiente directiva:
	`Using System.Data.SqlClient;`
	
	También se puede usar en la definición de un alias sobre un espacio de nombre. Por ejemplo, si se necesita usar la librería VisualBasic, se podría definirlo de la siguiente forma:
	`using mv = Microsoft.VisualBasic;`
	
	La librería Microsoft.VisualBasic tiene el alias «mv», esto es para invocar a sus métodos de forma directa. Observe cómo invocar al método InputBox:
	`string codigo = mv.Interaction.InputBox("Ingrese codigo: ");`

Liberación de recursos	En este caso, Using actúa como liberador de recursos al finalizar su ámbito. En un bloque Using se puede usar todo tipo de objetos y estos serán liberados cuando el ámbito del using finalice.
	`using (SqlConnection cn = new SqlConnection("server=MTORRES-PC;`
	`database=contrato;`
	`integrated security=SSPI"))`
	`{`
	`//Objeto de la clase SqlCommand`
	`SqlCommand cmd = new SqlCommand();`
	`cmd.Connection = cn;`
	`cmd.CommandText = "SP_LISTACLIENTE";`
	`//Creando el objeto de la clase SQLDataAdapter`
	`SqlDataAdapter da = new SqlDataAdapter();`
	`da.SelectCommand = cmd;`
	`//Enviando la informacion al Datase`
	`DataSet ds = new DataSet();`
	`da.Fill(ds);`
	`DataView dv = new DataView();`
	`dv.Table = ds.Tables[0];`
	`dgClientes.DataSource = dv;`
	`}`

4.19 Clase SqlCommand

Representa un procedimiento almacenado o una instrucción de Transact-SQL, que se ejecuta en una base de datos de SQL Server.

Métodos constructores:

SqlCommand()	Inicializa con un valor vacío el objeto de la clase SqlCommand y a partir de sus propiedades se especifica el tipo de comando y la conexión. Por ejemplo, si se crea el objeto de la clase SqlCommand vacía, el código sería: `SqlCommand cmd = new SqlCommand();`

SqlCommand (String)	Inicializa el objeto con una sentencia SQL o procedimiento almacenado, por ejemplo, si se crea el objeto de la clase SqlCommand en base a una sentencia de SQL:
	`String cadena = "SELECT * FROM CLIENTE";`
	`SqlCommand cmd = new SqlCommand(cmd);`

EDITORIAL MACRO®

SqlCommand (String, SqlConnection)	Inicializa el objeto con una sentencia SQL Server o procedimiento almacenado y además especifica la cadena de conexión. Por ejemplo, si se crea el objeto de la clase SqlCommand en base a una sentencia SQL y una conexión especifica: `SqlCommand cmd = new SqlCommand("SELECT * FROM CLIENTE",cn);`

SqlCommand (String, SqlConnection, SqlTransaction)	Inicializa el objeto con una sentencia SQL Server, una conexión y un objeto de transacción. Por ejemplo, si se crea el objeto de la clase SqlCommand en base a una cadena de conexión y un objeto de transacción:
	`SqlTransaction tr= cn.BeginTransaction(Isolation.Serializable);`
	`String cadena = "SELECT * FROM CLIENTE";`
	`SqlCommand cmd = new SqlCommand(cmd,cn,tr);`

Propiedades:

CommandText	Permite asignar la sentencia SQL o nombre del procedimiento almacenado al objeto de clase SqlCommand. Se puede especificar una sentencia SQL de la siguiente manera:
	`SqlCommand cmd = new SqlCommand();`
	`cmd.CommandText="SELECT * FROM CLIENTE";`
	O también se puede especificar un procedimiento almacenado de la siguiente manera:
	`SqlCommand cmd = new SqlCommand();`
	`Cmd.CommandText="SP_LISTACLIENTE";`

CommandType	Permite especificar el tipo de sentencia que debe interpretar el objeto de la clase SqlCommand.
	`SqlCommand cmd = new SqlCommand();`
	`cmd.CommandText="SP_LISTACLIENTE";`
	`cmd.Commandtype = CommandType.StoreProcedure;`

Connection	Permite especificar de qué cadena de conexión proviene la sentencia asociada al objeto de la clase SqlCommand.
	`SqlConnection cn = new SqlConnection("server=.;`
	`database=contrato;`
	`integrated security=SSPI");`
	`SqlCommand cmd = new SqlCommand();`
	`cmd.Connection =cn;`
	`cmd.CommandText="SP_LISTACLIENTE";`
	`cmd.Commandtype = CommandType.StoreProcedure;`

Parameters	Permite especificar los parámetros de entrada de una determinada sentencia o procedimiento almacenado. Esta propiedad se usará cuando la consulta presenta uno o más criterios.
	`SqlCommand cmd = new SqlCommand("SP_NUEVOCLIENTE",cn);`
	`cmd.Commandtype = CommandType.StoreProcedure;`
	
	`cmd.Parameters.Add("@cod",SqlDbType.Char).Value = codigo;`
	`cmd.Parameters.Add("@nom",SqlDbType.Char).Value = nombres;`
	`cmd.Parameters.Add("@dir",SqlDbType.Char).Value = direccion;`
	`cmd.Parameters.Add("@tel",SqlDbType.Char).Value = telefono;`

Transaction	Especifica la transacción que puede tener una sentencia SQL o procedimiento almacenado.
	`SqlTransaction tr= cn.BeginTransaction(Isolation.Serializable);`
	`String cadena = "SELECT * FROM CLIENTE";`
	`SqlCommand cmd = new SqlCommand(cmd,cn);`
	`cmd.Transaction = tr;`

Métodos:

Dispose	Permite liberar los recursos usados al crear el objeto de la clase SqlCommand. `cmd.dispose();`

ExecuteNonQuery	Permite especificar que la sentencia SQL o procedimiento almacenado está ejecutando sentencia del lado servidor como una inserción, actualización o eliminación.
	`SqlCommand cmd = new SqlCommand("SP_NUEVOCLIENTE",cn);`
	`cmd.Commandtype = CommandType.StoreProcedure;`
	
	`cmd.Parameters.Add("@cod",SqlDbType.Char).Value = codigo;`
	`cmd.Parameters.Add("@nom",SqlDbType.Char).Value = nombres;`
	`cmd.Parameters.Add("@dir",SqlDbType.Char).Value = direccion;`
	`cmd.Parameters.Add("@tel",SqlDbType.Char).Value = telefono;`
	
	`cmd.ExecuteNonQuery();`
	
	Se debe tener en cuenta que es obligatorio especificar la apertura de la conexión con el siguiente código cn.open().

ExecuteScalar	Permite especificar que se obtendrá solo el primer elemento de una sentencia SQL o procedimiento almacenado de tipo consulta. Esto podrá ayudar al recuperar un solo valor desde el servidor como, por ejemplo, el último código de cliente o el total de clientes.
	`SqlCommand cmd = new SqlCommand("SP_ULTIMOCLIENTE",cn);`
	`cmd.Commandtype = CommandType.StoreProcedure;`
	`string codigo = cmd.ExecuteScalar();`
	
	Se debe tener en cuenta que es obligatorio especificar la apertura de la conexión con el siguiente código cn.open().

4.20 Casos desarrollados

Caso desarrollado 1 : Uso de asistente - Listado de clientes

Implementar una aplicación que permita listar los datos de los clientes desde la base de datos Contrato. Al final, muestre el total de clientes mostrados.

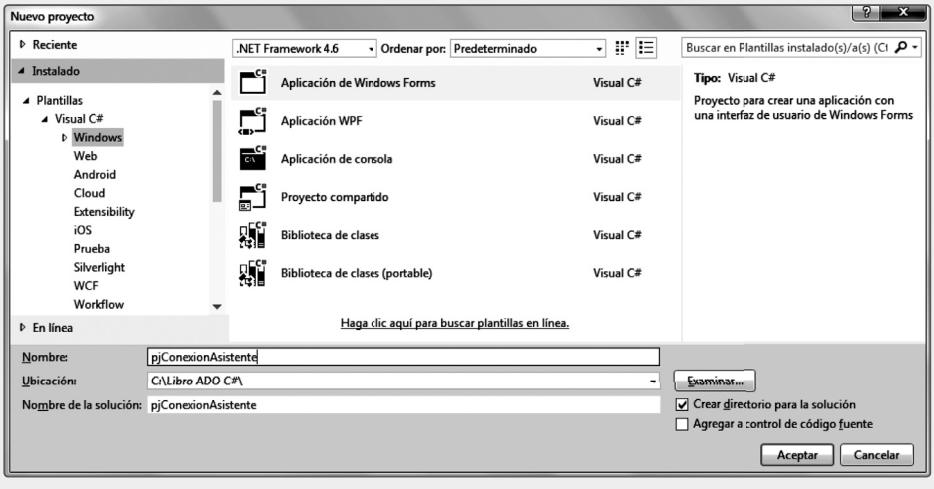
Se tiene en cuenta lo siguiente:

- Usar el asistente de configuración de origen de datos para la conexión y la administración de los datos.
- Usar el control DataGridView para listar los clientes.
- Mostrar el total de clientes en un control Label.

GUI propuesto:

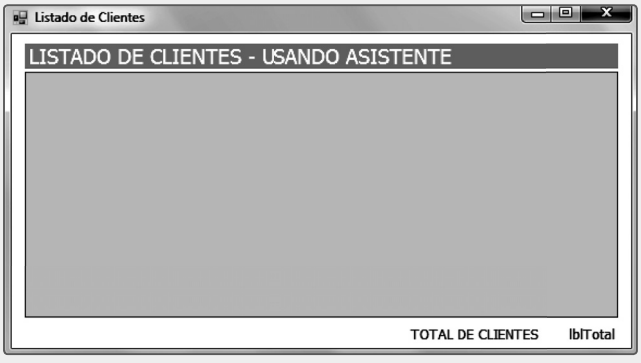
Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjConexionAsistente.

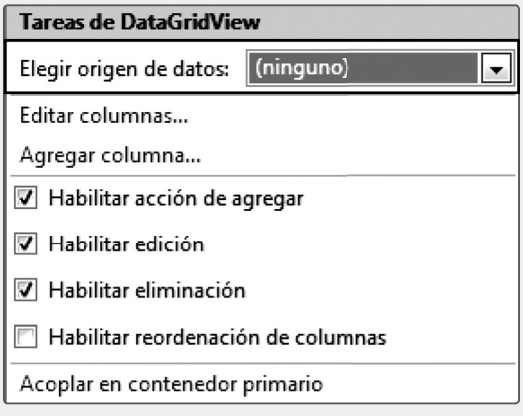


2. Cambie el nombre al formulario por frmListadoClientes. Coloque los objetos necesarios para la aplicación y asigne las siguientes propiedades:

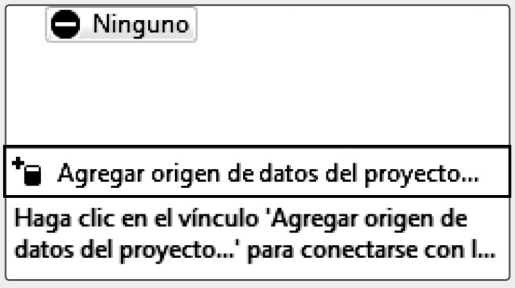
Control	Propiedad
Form1	**Text** Listado de clientes
Label1	**Text** LISTADO DE CLIENTES - USANDO ASISTENTE
Label2	**Text** TOTAL DE CLIENTES
Label3	**(Name)** lblTotal **Text** dejar vacío
DataGridView1	**(Name)** dgClientes



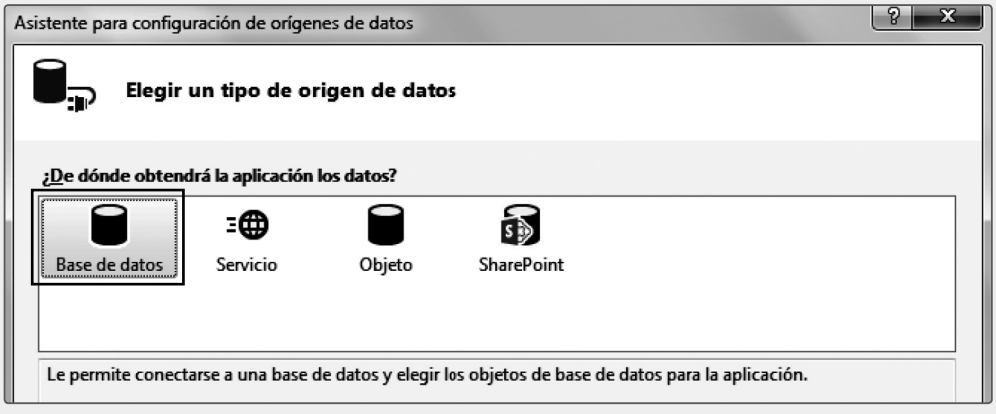
3. Seleccione **Tareas de DataGridView** desde la pestaña superior derecha que se presenta en el control DataGriView.



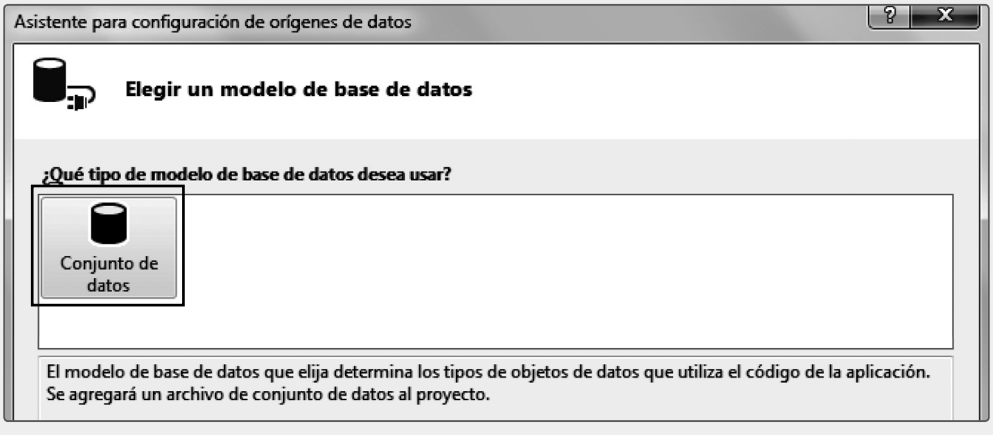
4. Luego, seleccione la opción **Elegir origen de datos** > **Agregar origen de datos del proyecto**



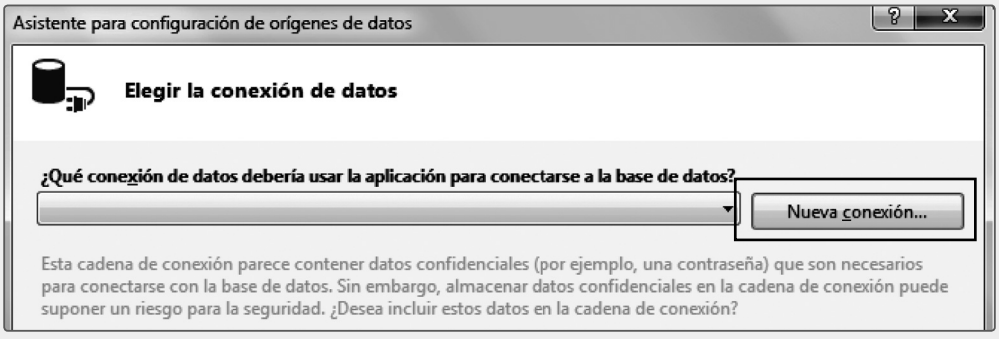
5. Ahora, seleccione el origen de los datos que necesita mostrar en el control. A continuación, seleccione **Base de datos**.



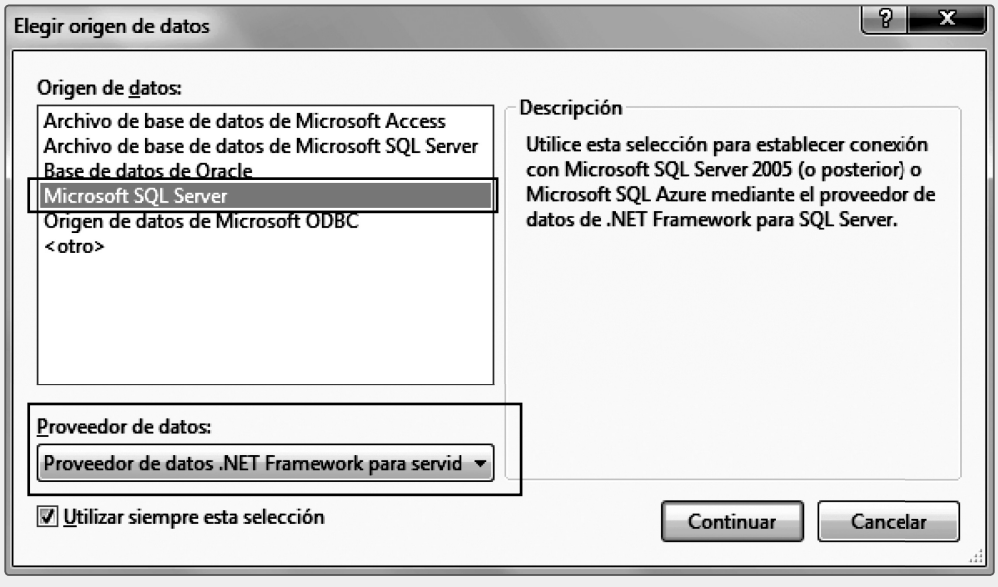
6. El resultado de los datos debe almacenarse en un repositorio, por tal motivo seleccione **Conjunto de datos** de la ventana **Elegir un modelo de base de datos**.



7. Ahora, configure la cadena de conexión a SQL Server, para esto inicie por seleccionar **Nueva conexión.**

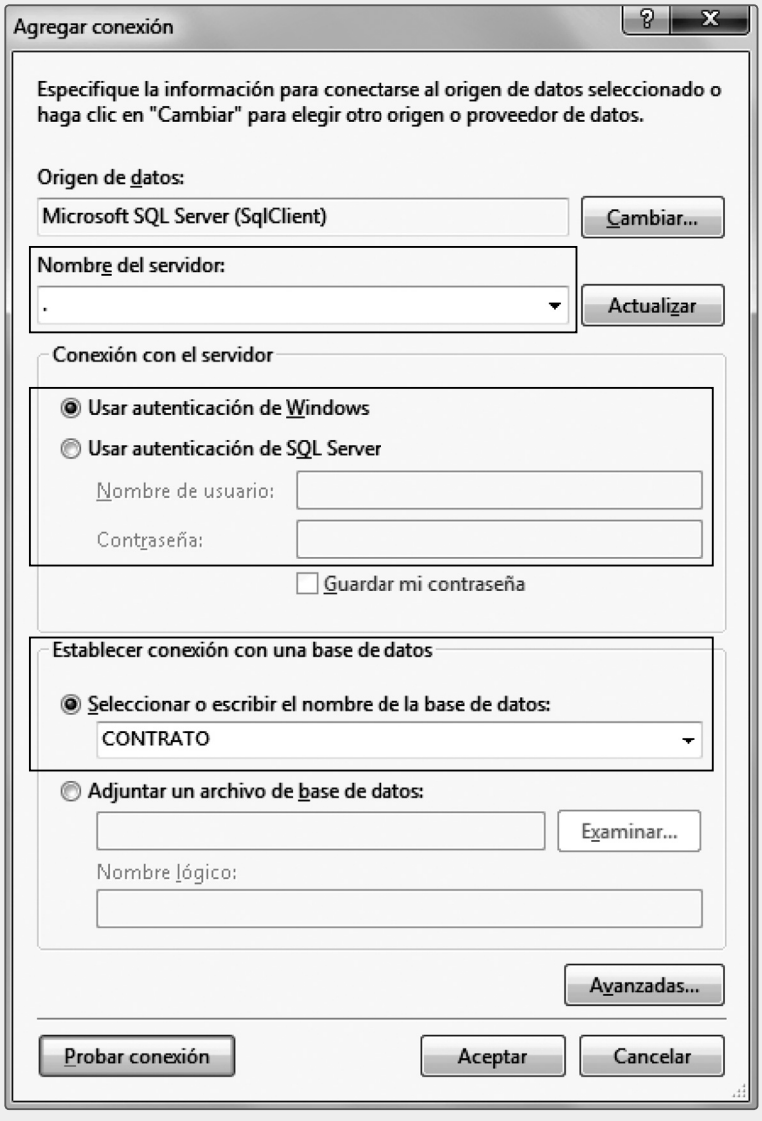


8. Aparece una lista de orígenes a los cuales puede conectarse. Para este caso, seleccione **Microsoft SQL Server**. Debe notar que el proveedor de datos mostrará **Proveedor de datos.NET Framework para servidor SQL Server**.



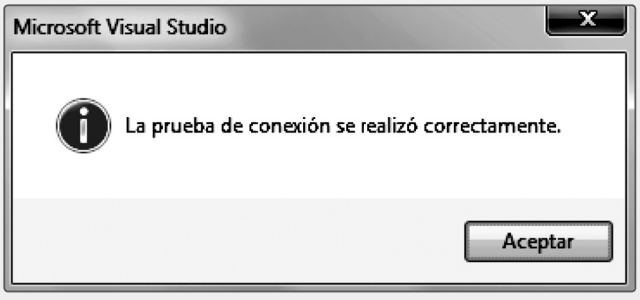
9. Ahora, debe realizar la conexión a la base de datos Contrato. Para esto, coloque el nombre del servidor. Aquí se tiene varias posibilidades como, por ejemplo, colocar el nombre de la misma computadora si es que esta se encuentra configurada como servidor o colocar el número de IP de una máquina conectada en red.

Para este caso, se debe usar el punto como referencia a que la computadora es el servidor de datos, ya que aquí se encuentra instalado SQL Server:

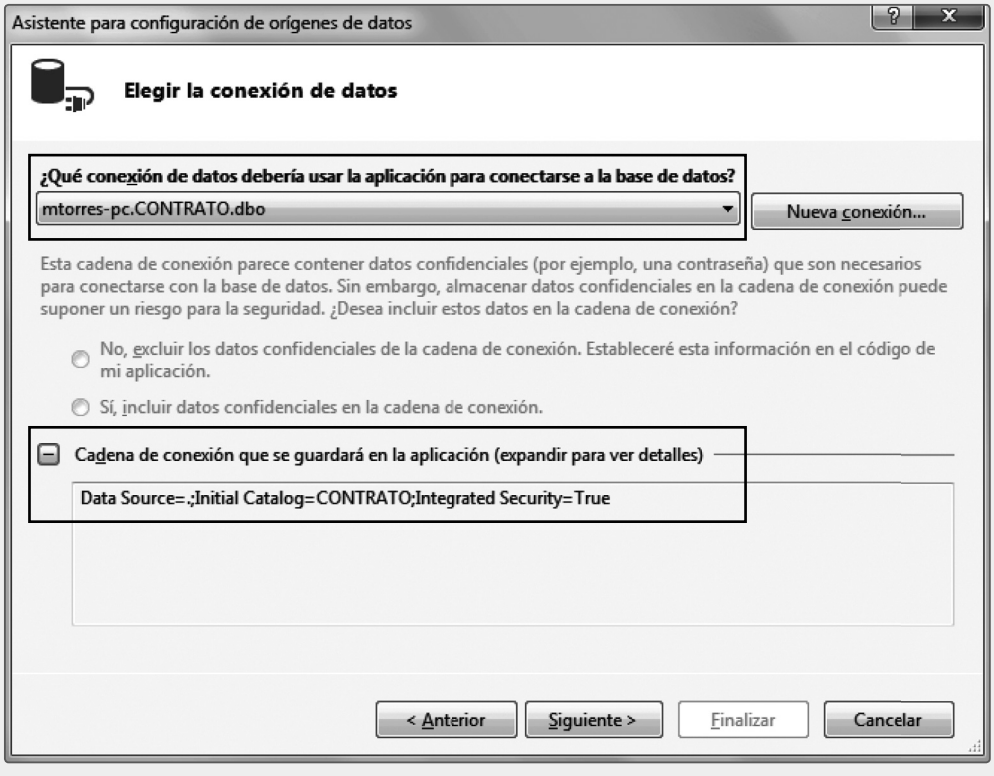


10. Luego, configure la forma de conexión al servidor. Aquí se presentan dos opciones. **Usar autentificación de Windows** para lo cual no se necesita especificar ni usuario ni clave, ya que el solo hecho de encontrarse en Windows es suficiente para el acceso.

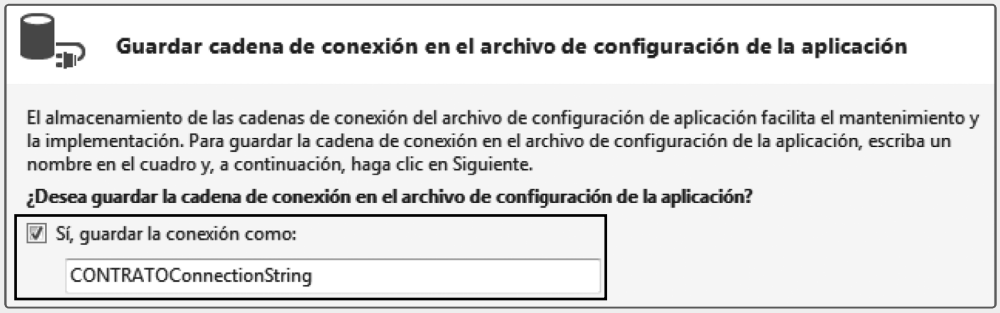
El otro es seleccionar **Usar autentificación de SQL Server**, para lo cual se debe especificar el nombre de usuario y la clave asignada para el acceso de los datos en el servidor SQL Server. En cualquier situación, se puede probar la conexión usando el botón Probar conexión. Si todo es correcto, se mostrará la siguiente imagen:



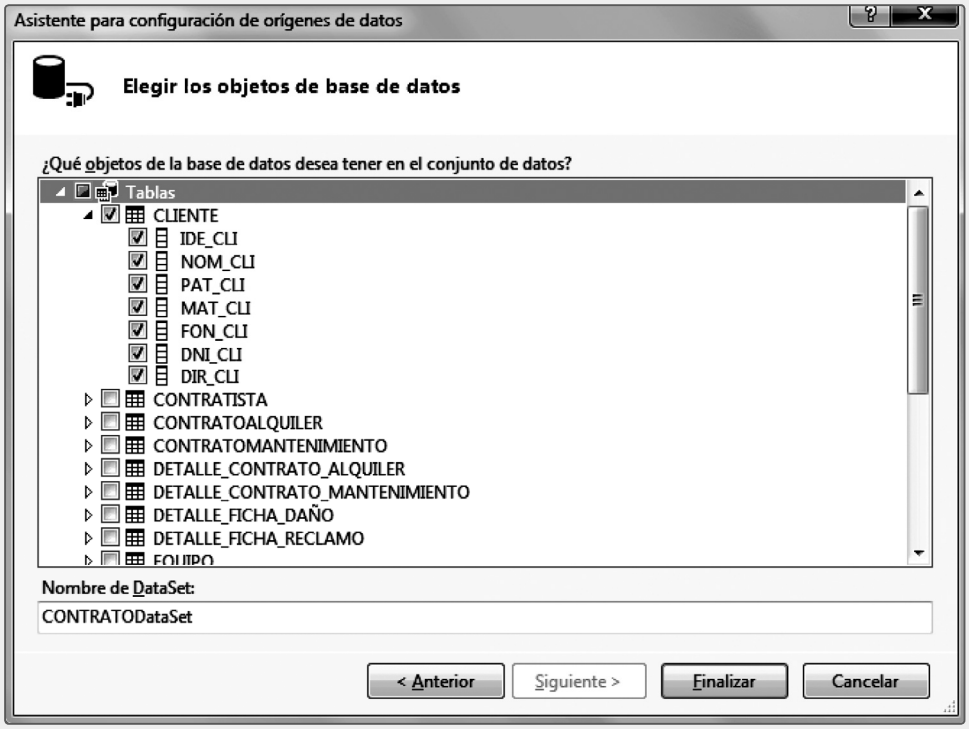
11. Una vez terminada la configuración, se debe volver a la imagen anterior, pero esta vez mostrando los datos de la conexión, como se muestra en la siguiente imagen:



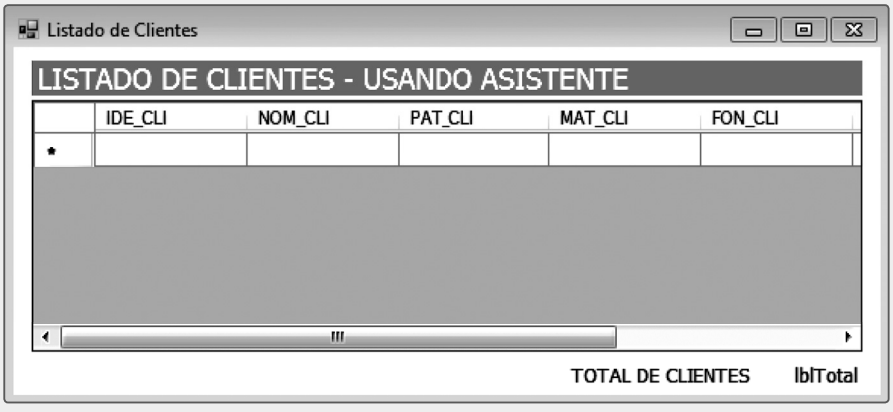
12. La sección que indica «¿Qué conexión de datos debería usar la aplicación para conectarse a la base de datos?» debe mostrar el nombre del servidor y el nombre de la base de datos. Además, se puede visualizar la cadena de conexión al expandir el detalle y notar que Data Source hace referencia al nombre del servidor, Initial Catalog hace referencia a la base de datos e Integrated Security hace referencia a la seguridad del servidor SQL Server. Luego, siguiendo el asistente pide ¿Desea guardar la cadena de conexión en el archivo de configuración de la aplicación? Solo debe dejar activado el check que indica **Sí, guardar la conexión como** y presionar clic **Siguiente**.



13. Finalmente, debe seleccionar **Tablas** y desplegar hasta mostrar las tablas contenidas y seleccionar **CLIENTE**. Como se notará, se activan los check en todos los atributos de la tabla CLIENTE. Esto indica que el listado mostrará todas las columnas de la tabla CLIENTE. Presionar clic en **Finalizar** para terminar el proceso.



Al finalizar, el proceso el datagridview debe mostrar el nombre de las columnas obtenidas desde la tabla seleccionada en el asistente. Esto indica que la conexión se ha configurado correctamente.



14. Para mostrar el total de clientes registrados en la tabla, necesita agregar código en el evento load del formulario. Para lograrlo, debe presionar doble clic en la barra de titulo del formulario y colocar el siguiente código:

```
private void frmListadoClientes_Load(object sender, EventArgs e)
{
    this.cLIENTETableAdapter.Fill(this.cONTRATODataSet.CLIENTE);
    lblTotal.Text = dgClientes.RowCount.ToString();
}
```

Finalmente, al ejecutar la aplicación, el formulario debe mostrarse de la siguiente forma:

Como sugerencia, se recomienda revisar el archivo app.config desde el Explorador de Soluciones, pues, aquí se muestra cómo el asistente hizo la configuración de la cadena de conexión:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="pjConexionAsistente.Properties.Settings.
        CONTRATOConnectionString"
        connectionString="Data Source=.;Initial Catalog=CONTRATO;Integrated
        Security=True"
        providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
</configuration>
```

Es necesario conocer el etiquetado <ConnectionStrings>, ya que aquí se está configurando la cadena de conexión al servidor SQL Server, donde «name» permite asignar un nombre a la cadena de conexión, y «connectionString» permite definir la cadena de conexión hacia el servidor local, especificar la base de datos y definir la autenticación al SQL Server.

Caso desarrollado 2 : Cadena de conexión directa - Listado de clientes

Implementar una aplicación que permita listar los datos de los clientes desde la base de datos Contrato. Al final, presentar el total de clientes mostrados.

Se tiene en cuenta lo siguiente:

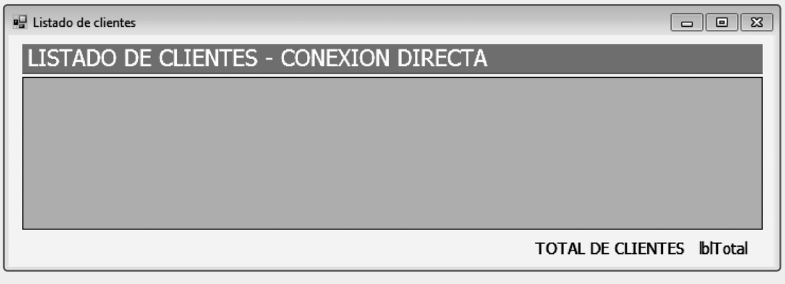
- La cadena de conexión debe implementarse en el mismo formulario.
- Implementar en SQL Server un procedimiento almacenado que permita listar los datos de los clientes.
- Usar el control DataGridView para listar los clientes.
- Mostrar el total de clientes en un control Label.

GUI propuesto:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjConexionDirecta.
2. Cambie el nombre al formulario por frmConexionDirecta, coloque los objetos necesarios para la aplicación y asigne las siguientes propiedades:

Control	Propiedad	
Form1	**Text**	Listado de clientes
Label1	**Text**	LISTADO DE CLIENTES - CONEXIÓN DIRECTA
Label2	**Text**	TOTAL DE CLIENTES
Label3	**(Name)**	lblTotal
	Text	dejar vacío
DataGridView1	**(Name)**	dgClientes



3. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTACLIENTE")IS NOT NULL
        DROP PROC SP_LISTACLIENTE
GO
CREATE PROC SP_LISTACLIENTE
AS
        SELECT *
                FROM CLIENTE
GO
```

El código completo 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;
//Libreria ADO NET
using System.Data.SqlClient;

namespace pjConexionDirecta
{
    public partial class frmConexionDirecta : Form
    {
        public frmConexionDirecta()
        {
            InitializeComponent();
        }

        private void frmConexionDirecta_Load(object sender, EventArgs e)
        {
            dgClientes.DataSource = listaClientes();
            lblTotal.Text = listaClientes().Rows.Count.ToString();
        }

        //Método de conexión
        SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection("server=.;
                                                 database=contrato;
                                              integrated security=SSPI");
            return cn;
        }

        //Método que lista los clientes
        DataTable listaClientes()
```

```
        {
            SqlConnection cn = getConecta();
            SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTE",cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }
    }
}
```

Caso desarrollado 3 : Usando clase - Listado de clientes

Implementar una aplicación Windows Forms que permita listar los datos de los clientes en un control DataGridView desde la base de datos Contrato.

Se tiene en cuenta lo siguiente:

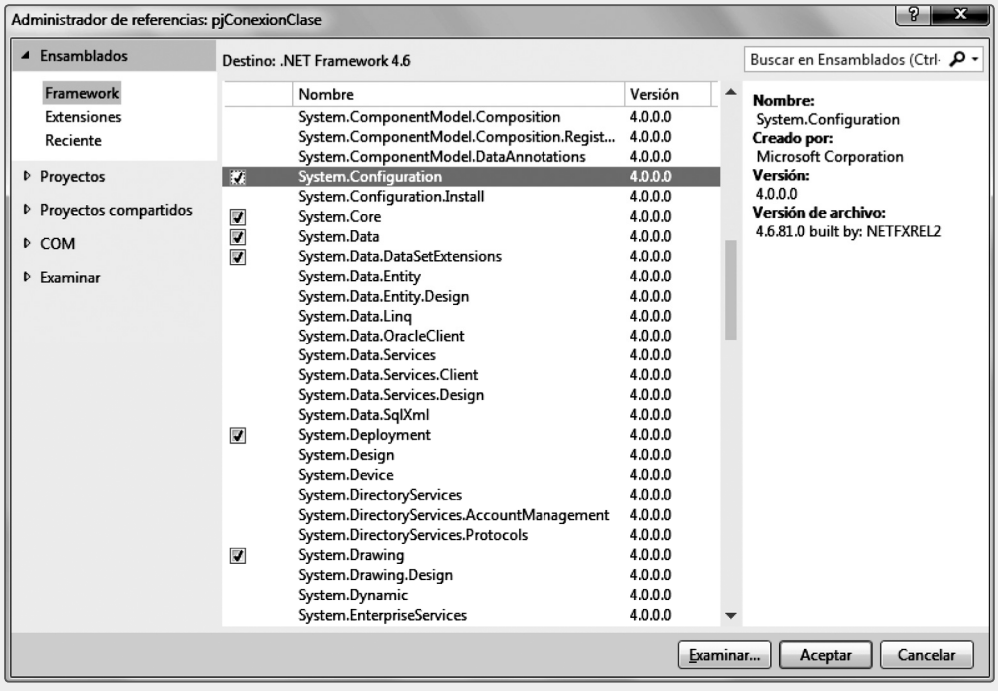
- Implementar en SQL Server un procedimiento almacenado SP_LISTACLIENTE que permita listar los datos de los clientes.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar el método listaClientes que permita asociarse al procedimiento almacenado SP_LISTACLIENTE.
- Usar el control DataGridView para listar los clientes.
- Mostrar el total de clientes en un control Label.

GUI propuesto:

IDE_CLI	NOM_CLI	PAT_CLI	MAT_CLI	FON_CLI	DNI_
CL0001	MARIA	PEREZ	TORRES	5440855	12345
CL0002	JUNIOR	FERNANDEZ	LOPEZ	5656236	98652
CL0003	ALEX	CARDENAS	ZARRASCO	1235821	45698
CL0004	JUAN	MEZQUITA	ROSAS	5023169	14702
CL0005	ERICK	GONZALES	SHULLER	8520365	85236
CL0006	JOSE	GALVEZ	RETAMOZO	9850352	96785
CL0007	CARLOS	JARA	SALAS	1230124	23605
CL0008	EDWARD	RAMIREZ	ROJAS	5500002	00213
CL0009	MANUEL	GUIZADO	GAMARRA	5232155	32569
CL0010	ALBERTO	FARFAN	PRADO	8567412	98657

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjConexionClase.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto y luego en **Agregar** > **Referencia**. Después, active el check en **System. Configuration**, como se muestra en la siguiente imagen:



3. Cambie el nombre al formulario por frmConexionClase, coloque los objetos necesarios para la aplicación y asigne las siguientes propiedades:

Control	Propiedad
Form1	**Text** Listado de clientes
Label1	**Text** LISTADO DE CLIENTES - CLASE
Label2	**Text** TOTAL DE CLIENTES
Label3	**(Name)** lblTotal **Text** dejar vacío
DataGridView1	**(Name)** dgClientes

4. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTACLIENTE")IS NOT NULL
        DROP PROC SP_LISTACLIENTE
GO
CREATE PROC SP_LISTACLIENTE
AS
        SELECT *
                FROM CLIENTE
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

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

//Librerias ADO NET
using System.Data.SqlClient;
using System.Configuration;

namespace pjConexionClase
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager.
            ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

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

//Librerias ADO NET
using System.Data;
using System.Data.SqlClient;

namespace pjConexionClase
{
    public class LogicaNegocio
    {
        //Objeto de la clase Conexion
        Conexion objCon = new Conexion();

        //Variable de la clase SqlConnection
        SqlConnection cn;

        //Método que lista los clientes
        public DataTable listaClientes()
        {
            cn = objCon.getConecta();
            SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTE",cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }
    }
}
```

8. El código del formulario frmConexionClase completo 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 pjConexionClase
{
    public partial class frmConexionClase : Form
    {
```

```
        //Objeto de la clase LogicaNegocio
        LogicaNegocio objL = new LogicaNegocio();

        public frmConexionClase()
        {
            InitializeComponent();
        }

        private void frmConexionClase_Load(object sender, EventArgs e)
        {
            dgClientes.DataSource = objL.listaClientes();
            lblTotal.Text = objL.listaClientes().Rows.ToString();
        }
    }
}
```

Caso desarrollado 4 : Manejo de dataset - Listado de contratistas

Implementar una aplicación Windows Forms que permita listar los datos de los contratistas en un control DataGridView desde la base de datos Contrato.

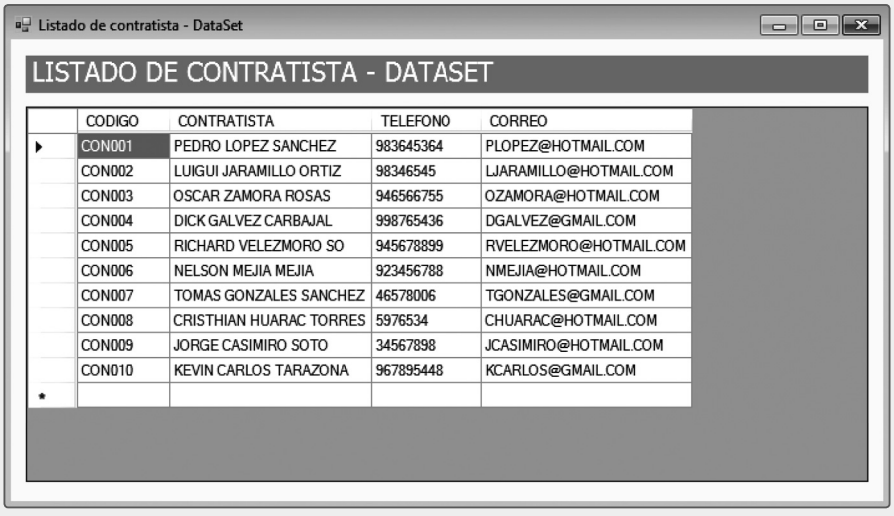
Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado SP_LISTACONTRATISTA que permita listar los datos completos de los contratistas.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, que permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar el método listaContratista, que permita asociarse al procedimiento almacenado SP_LISTACONTRATISTA usando DATASET.
- La tabla Contratista cuenta con la siguiente estructura:

CONTRATISTA

Column Name	Condensed Type
IDE_CON	char(6)
NOM_CON	varchar(30)
PAT_CON	varchar(20)
MAT_CON	varchar(20)
FON_CON	varchar(15)
EMA_CON	varchar(50)

GUI propuesto:



CODIGO	CONTRATISTA	TELEFONO	CORREO
CON001	PEDRO LOPEZ SANCHEZ	983645364	PLOPEZ@HOTMAIL.COM
CON002	LUIGUI JARAMILLO ORTIZ	98346545	LJARAMILLO@HOTMAIL.COM
CON003	OSCAR ZAMORA ROSAS	946566755	OZAMORA@HOTMAIL.COM
CON004	DICK GALVEZ CARBAJAL	998765436	DGALVEZ@GMAIL.COM
CON005	RICHARD VELEZMORO SO	945678899	RVELEZMORO@HOTMAIL.COM
CON006	NELSON MEJIA MEJIA	923456788	NMEJIA@HOTMAIL.COM
CON007	TOMAS GONZALES SANCHEZ	46578006	TGONZALES@GMAIL.COM
CON008	CRISTHIAN HUARAC TORRES	5976534	CHUARAC@HOTMAIL.COM
CON009	JORGE CASIMIRO SOTO	34567898	JCASIMIRO@HOTMAIL.COM
CON010	KEVIN CARLOS TARAZONA	967895448	KCARLOS@GMAIL.COM

Solución:

1. Haga clic en **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjContratistaDataSet.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto y luego en **Agregar** > **Referencia**. Después, active el check en System.Configuration.
3. Cambie el nombre al formulario por frmListadoContratista y asigne los nombres necesarios a los controles.
4. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTACONTRATISTA")IS NOT NULL
       DROP PROC SP_LISTACONTRATISTA
GO
CREATE PROC SP_LISTACONTRATISTA
AS
       SELECT C.IDE_CON AS CODIGO,
              C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
              C.FON_CON AS TELEFONO,
              C.EMA_CON AS CORREO
              FROM CONTRATISTA C
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjContratistaDataSet
{
  public class Conexion
  {
    public SqlConnection getConecta()
    {
      SqlConnection cn = new SqlConnection(ConfigurationManager
                                .ConnectionStrings["cn"].ConnectionString);
      return cn;
    }
  }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjContratistaDataSet
{
    public class LogicaNegocio
    {
        //Definición GLOBAL
        Conexion objCon = new Conexion();
        SqlConnection cn = new SqlConnection();

        //Método que lista los contratistas
        public DataSet listaContratistas()
        {
            cn = objCon.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
            DataSet ds = new DataSet();
            da.Fill(ds,"CONTRATISTA");
            return ds;
        }
    }
}
```

8. El código del formulario frmListadoContratista completo 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.Windows.Forms;

namespace pjContratistaDataSet
{
  public partial class frmListadoContratista : Form
  {
    //Definicion GLOBAL
    LogicaNegocio objL = new LogicaNegocio();

    public frmListadoContratista()
    {
```

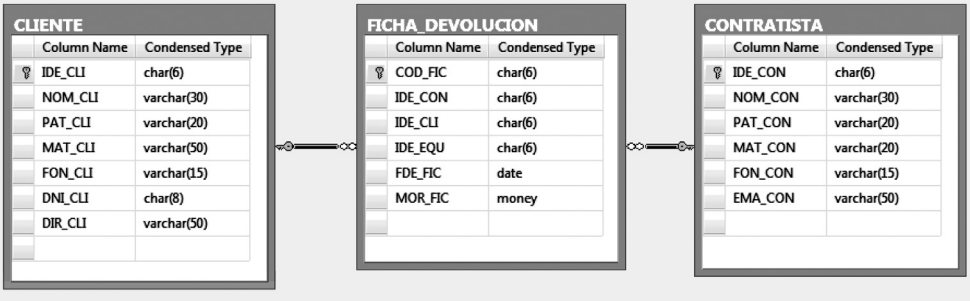
```
        InitializeComponent();
    }

    private void frmListadoContratista_Load(object sender, EventArgs e)
    {
        listaContratistas();
    }

    //Método que invoca al listado de contratistas desde DataSet
    void listaContratistas()
    {
    dgContratistas.DataSource=objL.listaContratistas().Tables["CONTRATISTA"];
    }
  }
}
```

Caso desarrollado 5 : Manejo de datatable - Listado de fichas de devolución

Implementar una aplicación Windows Forms que permita listar los datos de las fichas de devolución en un control DataGridView desde la base de datos Contrato.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado SP_LISTAFICHADEVOLUCION que permita listar los datos de las fichas de devolución. Tener en cuenta que también debe obtener información de la tabla cliente y contratista.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar el método listaFichasDevolucion, que permita asociarse al procedimiento almacenado SP_SP_LISTAFICHADEVOLUCION usando DATATABLE.
- La tabla ficha de devolución cuenta con la siguiente relación:



GUI propuesto:

CODIGO	CONTRATISTA	CLIENTE	EQUIPO	FE
FI0010	PEDRO LOPEZ SANCHEZ	MARIA PEREZ TORRES	MOUSE HP	5/2
FI0011	PEDRO LOPEZ SANCHEZ	ALBERTO FARFAN PRADO	MONITOR DELL 1707F	8/
FI0012	LUIGUI JARAMILLO ORTIZ	MANUEL GUIZADO GAMARRA	MOUSE HP	6/
FI0013	LUIGUI JARAMILLO ORTIZ	MARIA PEREZ TORRES	MONITOR DELL 1707F	5/
FI0014	PEDRO LOPEZ SANCHEZ	MANUEL GUIZADO GAMARRA	PROYECTOR CANON	7/
FI0015	OSCAR ZAMORA ROSAS	CARLOS JARA SALAS	PARLANTES LG	5/
FI0016	OSCAR ZAMORA ROSAS	ERICK GONZALES SHULLER	MONITOR DELL 1707F	8/
FI0017	PEDRO LOPEZ SANCHEZ	ALEX CARDENAS ZARRASCO	MOUSE HP	8/
FI0018	OSCAR ZAMORA ROSAS	MARIA PEREZ TORRES	TECLADO GENIUS	6/
FI0019	PEDRO LOPEZ SANCHEZ	JUNIOR FERNANDEZ LOPEZ	MONITOR DELL 1707F	8/

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjFichaDevolucionDataTable.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto y luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.
3. Cambie el nombre al formulario por frmFichaDevolucion y asigne los nombres adecuados a los controles del formulario.
4. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTAFICHADEVOLUCION")IS NOT NULL
        DROP PROC SP_LISTAFICHADEVOLUCION
GO
CREATE PROC SP_LISTAFICHADEVOLUCION
AS
        SELECT FD.COD_FIC AS CODIGO,
                C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
                CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
                E.DESC_EQU AS EQUIPO,
                FD.FDE_FIC AS FECHA
                FROM FICHA_DEVOLUCION FD
                JOIN CONTRATISTA C ON FD.IDE_CON=C.IDE_CON
                JOIN CLIENTE CL ON FD.IDE_CLI=CL.IDE_CLI
                JOIN EQUIPO E ON FD.IDE_EQU=E.IDE_EQU
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjFichaDevolucionDataTable
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;
```

```
namespace pjFichaDevolucionDataTable
{
  public class LogicaNegocio
  {
     //Definición GLOBAL
     Conexion objCon = new Conexion();
     SqlConnection cn = new SqlConnection();

     //Método que lista las fichas de devolución
     public DataTable listaFichasDevolucion()
     {
        cn = objCon.getConecta();
       SqlDataAdapter da = new SqlDataAdapter("SP_LISTAFICHADEVOLUCION", cn);
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
     }
  }
}
```

8. El código del formulario frmFichaDevolucion completo se muestra a continuación:

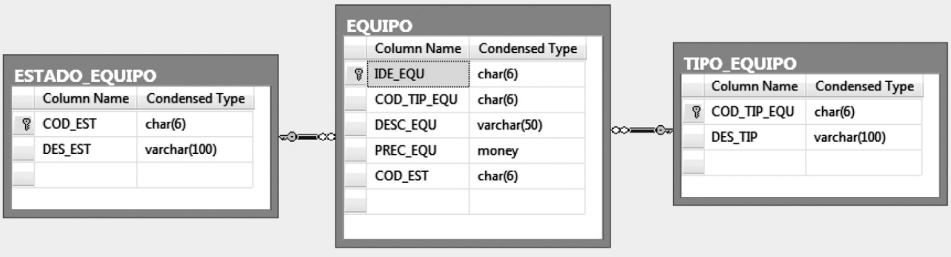
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace pjFichaDevolucionDataTable
{
    public partial class frmFichaDevolucion : Form
    {
        //Definición GLOBAL
        LogicaNegocio objL = new LogicaNegocio();

        public frmFichaDevolucion()
        {
            InitializeComponent();
        }

        private void frmFichaDevolucion_Load(object sender, EventArgs e)
        {
            listaFichasDevolucion();
        }
        void listaFichasDevolucion()
        {
            dgFichas.DataSource = objL.listaFichasDevolucion();
        }
    }
}
```

Caso desarrollado 6 : Manejo de dataview - Listado de equipos

Implementar una aplicación Windows Forms que permita listar los datos de los equipos en un control DataGridView desde la base de datos Contrato.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado SP_LISTAEQUIPO que permita listar los datos de los equipos, además de mostrar la descripción del estado y del tipo de equipo.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar el método listaEquipos que permita asociarse al procedimiento almacenado SP_LISTAEQUIPO usando la clase DataView.
- La tabla equipo cuenta con las siguientes relaciones:



GUI propuesto:

CODIGO	TIPO_EQUIPO	DESCRIPCION	PRECIO	ESTADO
EQ0001	MONITOR	MONITOR DELL 1707F	200.0000	DISPONIBLE
EQ0002	MOUSE	MOUSE HP	15.0000	DISPONIBLE
EQ0003	PARLANTES	PARLANTES LG	50.0000	OCUPADO
EQ0004	TECLADO	TECLADO GENIUS	40.0000	DISPONIBLE
EQ0005	CPU	CPU ASUS	250.0000	DISPONIBLE
EQ0006	LAPTOP	LAPTOP TOSHIBA	800.0000	OCUPADO
EQ0007	AUDIFONOS	AUDIFONOS SONNY	30.0000	DISPONIBLE
EQ0008	PROYECTOR	PROYECTOR CANON	500.0000	OCUPADO
EQ0009	IMPRESORA	IMPRESORA EPSON	200.0000	DISPONIBLE
EQ0010	FOTOCOPIADORA	FOTOCOPIADORA RICOH	750.0000	DISPONIBLE

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjEquipoDataView.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.
3. Cambie el nombre al formulario por frmEquiposDataView y asigne las propiedades necesarias a los controles del formulario.
4. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTAEQUIPO")IS NOT NULL
        DROP PROC SP_LISTAEQUIPO
GO
CREATE PROC SP_LISTAEQUIPO
AS
        SELECT E.IDE_EQU AS CODIGO,
               TE.DES_TIP AS TIPO_EQUIPO,
               E.DESC_EQU AS DESCRIPCION,
               E.PREC_EQU AS PRECIO,
               EE.DES_EST AS ESTADO
               FROM EQUIPO E
               JOIN TIPO_EQUIPO TE ON TE.COD_TIP_EQU=E.COD_TIP_EQU
               JOIN ESTADO_EQUIPO EE ON EE.COD_EST=E.COD_EST
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjEquipoDataView
{
    class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                    .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjEquipoDataView
{
    class LogicaNegocio
    {
        //Definicion GLOBAL
        Conexion objCon = new Conexion();
        SqlConnection cn = new SqlConnection();

        //Método que lista los equipos usando DataView
        public DataView listaEquipos()
        {
            cn = objCon.getConecta();
            SqlDataAdapter da = new SqlDataAdapter("SP_LISTAEQUIPO", cn);
            DataSet ds = new DataSet();
            da.Fill(ds,"EQUIPOS");
            DataView dv = new DataView();
            dv.Table = ds.Tables[0];
            return dv;
        }
    }
}
```

8. El código del formulario frmEquiposDataView completo 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.Windows.Forms;

namespace pjEquipoDataView
{
    public partial class frmEquiposDataView : Form
    {
        //Definicion GLOBAL
        LogicaNegocio objL = new LogicaNegocio();

        public frmEquiposDataView()
        {
            InitializeComponent();
        }

        private void frmEquiposDataView_Load(object sender, EventArgs e)
        {
            listaEquipos();
        }
        void listaEquipos()
        {
            dgEquipos.DataSource = objL.listaEquipos();
        }
    }
}
```

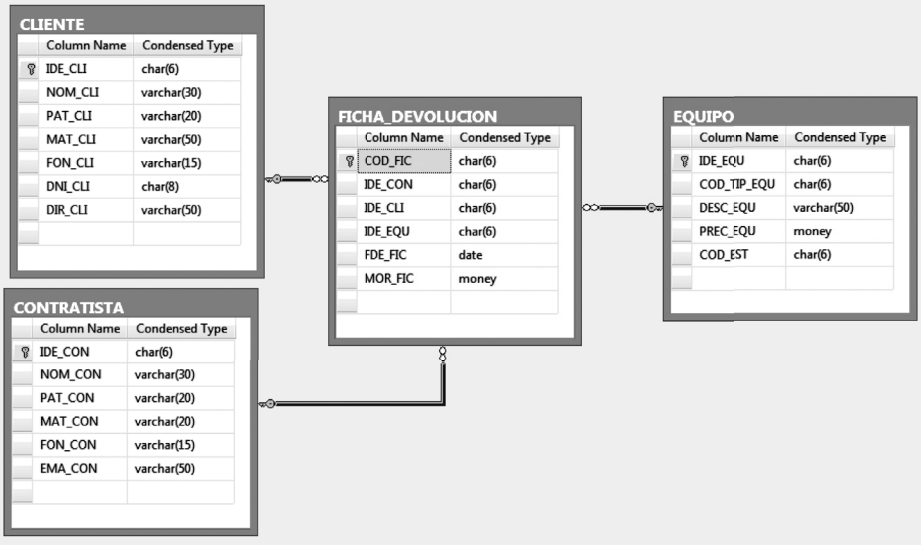
Caso desarrollado 7 : Listado de fichas de devolución por contratista

Implementar una aplicación Windows Forms que permita listar las fichas de devolución según el contratista desde la base de datos Contrato.

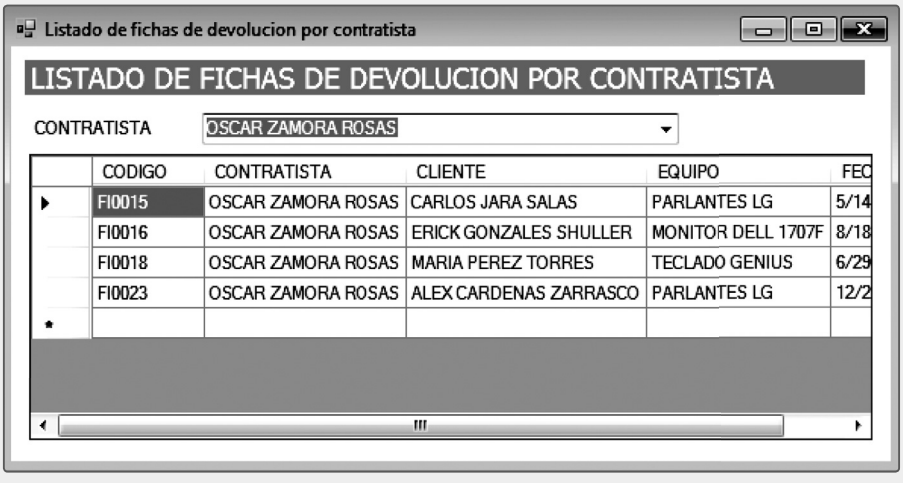
Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado SP_CONTRATISTA, que permita listar los datos de los contratistas, y otro procedimiento almacenado SP_LISTAFICHADEVOLUCIONXCONTRATISTA, que permita listar las fichas de devolución por contratista.
- Configurar la cadena de conexión en el archivo app.config.

- Crear la clase Conexión e implementar el método getConecta que permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar los métodos listaContratistas y listaFichasxContratista para que se asocien a los procedimientos almacenados.
- Las relaciones de las tablas se muestran a continuación:



GUI propuesto:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjFichasDevolucionxContratista.

2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System.Configuration.
3. Cambie el nombre al formulario por frmFichasDevolucionxContratista y asigne los nombres necesarios a los controles del formulario.
4. En SQL Server implemente los siguientes procedimientos almacenados:

```
--PROCEDIMIENTO QUE LISTA LOS CONTRATISTAS QUE SE MOSTRARAN EN EL
--CUADRO COMBINADO
IF OBJECT_ID("SP_CONTRATISTA")IS NOT NULL
        DROP PROC SP_CONTRATISTA
GO
CREATE PROC SP_CONTRATISTA
AS
        SELECT C.IDE_CON AS CODIGO,
                C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA
                FROM CONTRATISTA C
GO

--PROCEDIMIENTO ALMACENADO PARA LISTAR LAS FICHAS DE DEVOLUCION POR
--CONTRATISTA
IF OBJECT_ID("SP_LISTAFICHADEVOLUCIONXCONTRATISTA")IS NOT NULL
        DROP PROC SP_LISTAFICHADEVOLUCIONXCONTRATISTA
GO
CREATE PROC SP_LISTAFICHADEVOLUCIONXCONTRATISTA(@CON CHAR(6))
AS
        SELECT FD.COD_FIC AS CODIGO,
                C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
                CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
                E.DESC_EQU AS EQUIPO,
                FD.FDE_FIC AS FECHA
                FROM FICHA_DEVOLUCION FD
                JOIN CONTRATISTA C ON FD.IDE_CON=C.IDE_CON
                JOIN CLIENTE CL ON FD.IDE_CLI=CL.IDE_CLI
                JOIN EQUIPO E ON FD.IDE_EQU=E.IDE_EQU
                WHERE FD.IDE_CON=@CON
GO
--PRUEBA: SP_LISTAFICHADEVOLUCIONXCONTRATISTA "CON003"
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
```

```
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace frmFichasDevolucionxContratista
{
    public class conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace frmFichasDevolucionxContratista
{
    public class LogicaNegocio
    {
        //Definición GLOBAL
        conexión objCon = new conexión();
        SqlConnection cn = new SqlConnection();
```

```
        //Método que lista los contratistas
        public DataTable listaContratistas()
        {
            cn = objCon.getConecta();
            SqlDataAdapter da = new SqlDataAdapter("SP_CONTRATISTA", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

      //Método que lista las fichas por contratista
      public DataTable listaFichasxcontratista(string contratista)
      {
         cn = objCon.getConecta();
         cn.Open();
         SqlDataAdapter da=new
                    SqlDataAdapter("SP_LISTAFICHADEVOLUCIONXCONTRATISTA",cn);
         da.SelectCommand.CommandType = CommandType.StoredProcedure;
        da.SelectCommand.Parameters.Add("CON",SqlDbType.Char).Value=contratista;

         DataTable dt = new DataTable();
         da.Fill(dt);
         return dt;
      }
    }
}
```

8. El código del formulario frmFichasDevolucionxContratista completo 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.Windows.Forms;

namespace frmFichasDevolucionxContratista
{
  public partial class frmFichasDevolucionxContratista : Form
  {
    LogicaNegocio objL = new LogicaNegocio();
    public frmFichasDevolucionxContratista()
    {
      InitializeComponent();
    }
```

```
    private void frmFichasDevolucionxContratista_Load(object sender, EventArgs e)
    {
      llenaContratista();
    }

    private void cboContratista_SelectionChangeCommitted(…)
    {
      string contratista = cboContratista.SelectedValue.ToString();
     dgFichasDevolucion.DataSource = objL.listaFichasxcontratista(contratista);
    }

    void llenaContratista(){
            cboContratista.DataSource = objL.listaContratistas();
            cboContratista.DisplayMember = "CONTRATISTA";
            cboContratista.ValueMember = "CODIGO";
    }

  }
}
```

Debemos tener en cuenta que el cuadro combinado (cboContratista) muestra los datos de los contratistas y que al seleccionar uno de ellos se listara sus fichas de devolucion. Para este caso debemos acceder al evento SelectionChangeCommited, para esto primero seleccione el cuadro combinado y desde el panel propiedades seleccione el botón evento, luego busque el evento SelectionChangeCommited y presione doble clic para acceder al método con el mismo nombre.

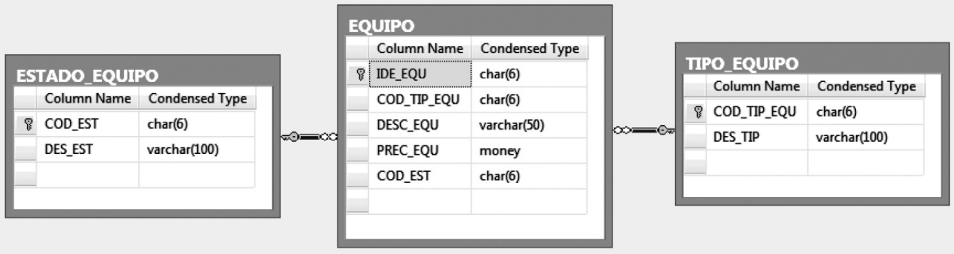
Caso desarrollado 8 : Listado de equipos por estado y tipo

Implementar una aplicación Windows Forms que permita listar la información de los equipos según el estado y tipo de equipo seleccionado a partir de cuadros combinados.

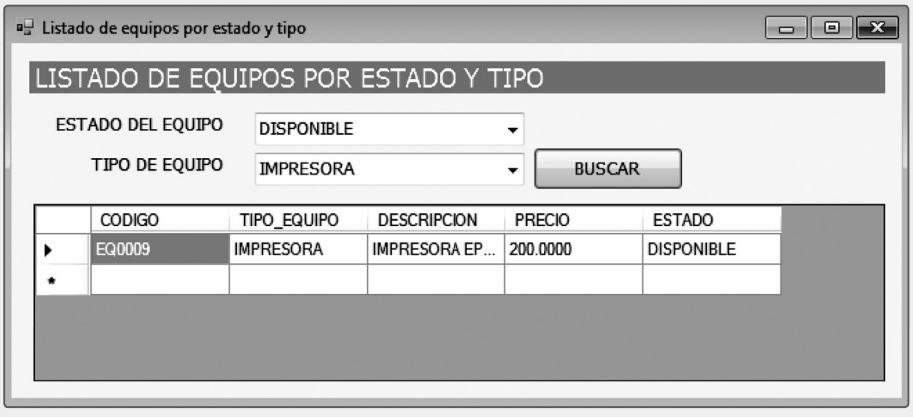
Se tiene en cuenta lo siguiente:

- Implementar en SQL Server procedimientos almacenados que permitan listar los datos de los estados, tipos y equipos.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar métodos necesarios para el llenado de los cuadros combinados del estado y tipo, además del listado de equipos según el estado y tipo.

- La relación de las tablas se muestra a continuación:



GUI propuesto:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjEquiposxTipoxEstado.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.
3. Cambie el nombre al formulario por frmEquipoxTipoxEstado.
4. En SQL Server implemente los siguientes procedimientos almacenados:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS TIPOS DE EQUIPOS
IF OBJECT_ID("SP_TIPOEQUIPO")IS NOT NULL
        DROP PROC SP_TIPOEQUIPO
GO
CREATE PROC SP_TIPOEQUIPO
AS
        SELECT T.COD_TIP_EQU AS CODIGO,
```

```
                T.DES_TIP AS TIPO
                FROM TIPO_EQUIPO T
GO


IF OBJECT_ID("SP_ESTADOEQUIPO")IS NOT NULL
        DROP PROC SP_ESTADOEQUIPO
GO
CREATE PROC SP_ESTADOEQUIPO
AS
        SELECT E.COD_EST AS CODIGO,
                E.DES_EST AS ESTADO
                FROM ESTADO_EQUIPO E
GO

IF OBJECT_ID("SP_EQUIPOSXESTADOXTIPO")IS NOT NULL
        DROP PROC SP_EQUIPOSXESTADOXTIPO
GO
CREATE PROC SP_EQUIPOSXESTADOXTIPO(@EST CHAR(6),@TIP CHAR(6))
AS
        SELECT E.IDE_EQU AS CODIGO,
                TE.DES_TIP AS TIPO_EQUIPO,
                E.DESC_EQU AS DESCRIPCION,
                E.PREC_EQU AS PRECIO,
                EE.DES_EST AS ESTADO
                FROM EQUIPO E
                JOIN TIPO_EQUIPO TE ON TE.COD_TIP_EQU=E.COD_TIP_EQU
                JOIN ESTADO_EQUIPO EE ON EE.COD_EST=E.COD_EST
                WHERE E.COD_EST=@EST AND E.COD_TIP_EQU=@TIP
GO
--PRUEBA: SP_EQUIPOSXESTADOXTIPO "0","TIP001"
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjEquiposxTipoxEstado
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjEquiposxTipoxEstado
{
    public class LogicaNegocio
    {
      //Definición GLOBAL
      Conexion objCon = new Conexion();
      SqlConnection cn = new SqlConnection();

      //Método que lista los tipos de equipos
      public DataTable listaTipo()
      {
        cn = objCon.getConecta();
        SqlDataAdapter da = new SqlDataAdapter("SP_TIPOEQUIPO", cn);
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
      }
```

```
      //Metodos que lista los estados de los equipos
      public DataTable listaEstado()
      {
        cn = objCon.getConecta();
        SqlDataAdapter da = new SqlDataAdapter("SP_ESTADOEQUIPO", cn);
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
      }


      //Metodos que lista los equipos según el tipo y estado
      public DataTable listaEquiposxTipoxEstado(string estado, string tipo)
      {
        cn = objCon.getConecta();
        cn.Open();
       SqlDataAdapter da = new SqlDataAdapter("SP_EQUIPOSXESTADOXTIPO", cn);
        da.SelectCommand.CommandType = CommandType.StoredProcedure;
       da.SelectCommand.Parameters.Add("EST", SqlDbType.Char).Value = estado;
       da.SelectCommand.Parameters.Add("TIP", SqlDbType.Char).Value = tipo;

        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
      }
    }
}
```

8. El código del formulario frmEquipoxTipoxEstado completo 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.Windows.Forms;

namespace pjEquiposxTipoxEstado
{
    public partial class frmEquipoxTipoxEstado : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmEquipoxTipoxEstado()
        {
```

```
            InitializeComponent();
        }

        private void frmEquipoxTipoxEstado_Load(object sender, EventArgs e)
        {
            llenaEstado();
            llenaTipoEquipo();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            string estado = cboEstado.SelectedValue.ToString();
            string tipo = cboTipo.SelectedValue.ToString();

          dgEquipos.DataSource = objL.listaEquiposxTipoxEstado(estado,tipo);

        }

        void llenaEstado()
        {
            cboEstado.DataSource = objL.listaEstado();
            cboEstado.DisplayMember = "ESTADO";
            cboEstado.ValueMember = "CODIGO";
        }

        void llenaTipoEquipo()
        {
            cboTipo.DataSource = objL.listaTipo();
            cboTipo.DisplayMember = "TIPO";
            cboTipo.ValueMember = "CODIGO";
        }
    }
}
```

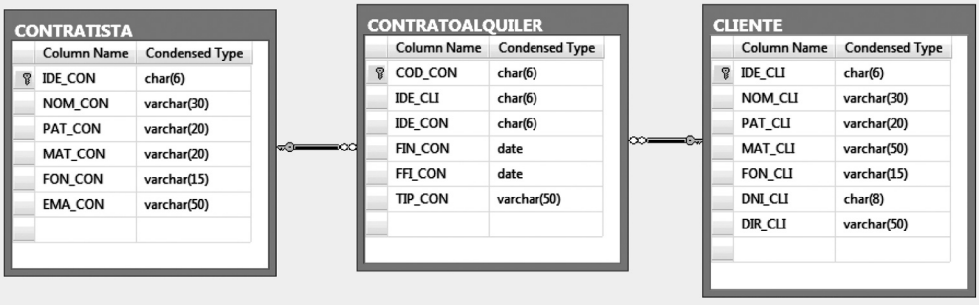
Caso desarrollado 9 : Filtrado de contratos por cliente

Implementar una aplicación Windows Forms que permita mostrar los datos de los contratos según el ingreso del nombre letra por letra del cliente.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado que filtra los datos de los contratos con la inicial del nombre del cliente. Se debe tomar en consideración que se muestra el nombre completo de cliente, así como de los contratistas.

- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar un método que permita asociarse al procedimiento almacenado que filtra los contratos de los clientes.
- La relación de las tablas se muestra a continuación:



GUI propuesto:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjContratosxCliente.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.
3. Cambie el nombre al formulario por frmContratosxCliente.

4. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTACONTRATOS")IS NOT NULL
        DROP PROC SP_LISTACONTRATOS
GO
CREATE PROC SP_LISTACONTRATOS(@LETRA VARCHAR(30))
AS
  SELECT CA.COD_CON AS CODIGO,
        C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE,
        CO.NOM_CON+SPACE(1)+CO.PAT_CON+SPACE(1)+CO.MAT_CON AS CONTRATISTA,
        CA.FIN_CON AS FECHAINICIO,
        CA.FFI_CON AS FECHAFIN,
        CA.TIP_CON AS TIPO
        FROM CONTRATOALQUILER CA
        JOIN CLIENTE C ON CA.IDE_CLI=C.IDE_CLI
        JOIN CONTRATISTA CO ON CA.IDE_CON=CO.IDE_CON
        WHERE C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI LIKE @LETRA+"%"
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
```

```
namespace pjContratosxCliente
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                    .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }

    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjContratosxCliente
{
  public class LogicaNegocio
  {
    //Definición GLOBAL
    Conexion objCon = new Conexion();
    SqlConnection cn = new SqlConnection();

    //Metodos que lista los contratos por cliente
    public DataTable listaContratosxCliente(string letra)
    {
      cn = objCon.getConecta();
      cn.Open();
      SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATOS", cn);
      da.SelectCommand.CommandType = CommandType.StoredProcedure;
     da.SelectCommand.Parameters.Add("LETRA", SqlDbType.VarChar).Value = letra;

      DataTable dt = new DataTable();
      da.Fill(dt);
      return dt;
    }
  }
}
```

8. El código del formulario frmContratosxCliente completo 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.Windows.Forms;

namespace pjContratosxCliente
{
    public partial class frmContratosxCliente : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmContratosxCliente()
        {
            InitializeComponent();
        }

        private void txtCliente_TextChanged(object sender, EventArgs e)
        {
        dgContratos.DataSource = objL.listaContratosxCliente(txtCliente.Text);
        }
    }
}
```

Caso desarrollado 10 : Búsqueda de datos de cliente

Implementar una aplicación Windows Forms que permita mostrar los datos de un determinado cliente mediante su código.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado que permita mostrar los datos de un cliente por su código.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.

- Crear la clase LogicaNegocio e implementar un método que permita mostrar los datos de un cliente por su código.
- Los datos de los clientes deberán mostrarse en controles Label como se muestra en la GUI propuesta.
- La tabla cliente cuenta con la siguiente estructura:

CLIENTE

Column Name	Condensed Type
IDE_CLI	char(6)
NOM_CLI	varchar(30)
PAT_CLI	varchar(20)
MAT_CLI	varchar(50)
FON_CLI	varchar(15)
DNI_CLI	char(8)
DIR_CLI	varchar(50)

GUI propuesto:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjBuscaCliente.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.

3. Cambie el nombre al formulario por frmBuscaCliente.
4. En SQL Server implemente el siguiente procedimiento almacenado:

```
IF OBJECT_ID("SP_LISTACLIENTE")IS NOT NULL
        DROP PROC SP_LISTACLIENTE
GO
CREATE PROC SP_LISTACLIENTE (@COD CHAR(6))
AS
        SELECT C.IDE_CLI AS CODIGO,
               C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE,
               C.FON_CLI AS TELEFONO,
               C.DNI_CLI AS DNI,
               C.DIR_CLI AS DIRECCION
               FROM CLIENTE C
               WHERE C.IDE_CLI=@COD
GO
--SP_LISTACLIENTE "CL0001"
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
```

```
namespace pjBuscaCliente
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager.
            ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjBuscaCliente
{
  public class LogicaNegocio
  {
    //Definición GLOBAL
    Conexion objCon = new Conexion();
    SqlConnection cn = new SqlConnection();

    //Método que lista los datos del cliente
    public DataTable listaCliente(string codigo)
    {
       cn = objCon.getConecta();
       cn.Open();
       SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTE", cn);
       da.SelectCommand.CommandType = CommandType.StoredProcedure;
       da.SelectCommand.Parameters.Add("COD", SqlDbType.VarChar).Value = codigo;

       DataTable dt = new DataTable();
       da.Fill(dt);
       return dt;
     }
   }
}
```

El código del formulario frmBuscaCliente completo se muestra a continuación:

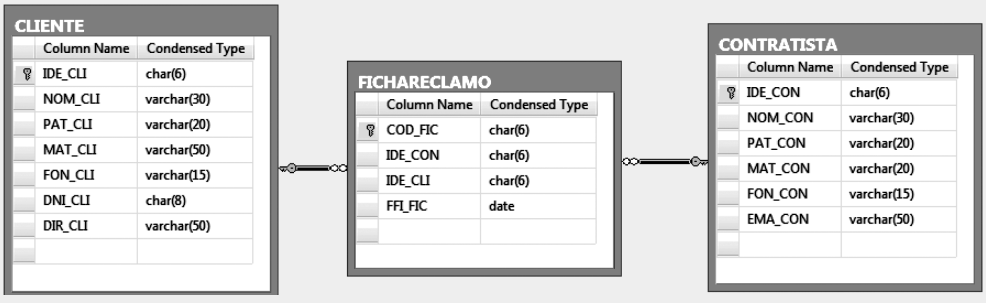
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace pjBuscaCliente
{
    public partial class frmBuscaCliente : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmBuscaCliente()
        {
            InitializeComponent();
        }

        private void btnMostrar_Click(object sender, EventArgs e)
        {
            try
            {
                string codigo = txtCodigo.Text;
                DataRow fila = objL.listaCliente(codigo).Rows[0];

                lblCodigo.Text = fila[0].ToString();
                lblCliente.Text = fila[1].ToString();
                lblFono.Text = fila[2].ToString();
                lblDNI.Text = fila[3].ToString();
                lblDireccion.Text = fila[4].ToString();
            }
            catch (Exception)
            {
                MessageBox.Show("El codigo de cliente no existe..!!");
                limpiaControles();
            }
        }

        void limpiaControles()
        {
            lblCodigo.Text = "";
            lblCliente.Text = "";
            lblFono.Text = "";
            lblDNI.Text = "";
            lblDireccion.Text = "";
        }

        private void frmBuscaCliente_Load(object sender, EventArgs e)
        {
            limpiaControles();
        }
    }
}
```

Caso desarrollado 11 : Búsqueda de fichas de reclamo por año

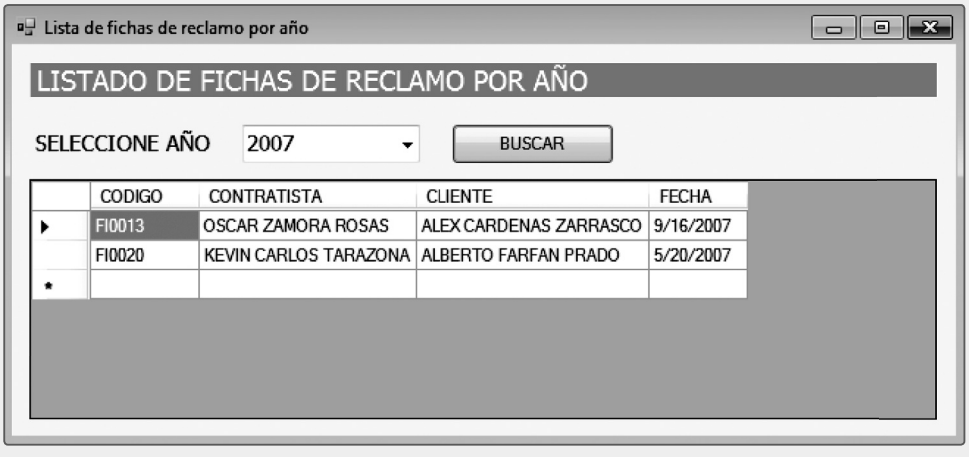
Implementar una aplicación Windows Forms que permita listar los datos de la fichas de reclamo en un determinado año.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server procedimientos almacenados que permitan mostrar los años en orden descendente y otro procedimiento almacenado que liste las fichas de reclamo por año.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar métodos que se asocien a los procedimientos almacenados. Se debe considerar que los años se mostrarán en un cuadro combinado.
- La relación de tablas se muestra a continuación:



GUI propuesto:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjFichasReclamoxAño.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.
3. Cambie el nombre al formulario por frmFichasReclamoxAño.
4. En SQL Server implemente los siguientes procedimientos almacenados:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS AÑOS
IF OBJECT_ID("SP_LISTAAÑOS")IS NOT NULL
        DROP PROC SP_LISTAAÑOS
GO
CREATE PROC SP_LISTAAÑOS
AS
        SELECT DISTINCT YEAR(F.FFI_FIC) AS AÑO
                FROM FICHARECLAMO F
                ORDER BY 1 DESC
GO

--PROCEDIMIENTO ALMACENADO QUE LISTE LAS FICHAS DE RECLAMO
IF OBJECT_ID("SP_LISTAFICHASRECLAMOXAÑO")IS NOT NULL
        DROP PROC SP_LISTAFICHASRECLAMOXAÑO
GO
CREATE PROC SP_LISTAFICHASRECLAMOXAÑO(@AÑO INT)
AS
        SELECT F.COD_FIC AS CODIGO,
                C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
                CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
                F.FFI_FIC AS FECHA
                FROM FICHARECLAMO F
                JOIN CONTRATISTA C ON C.IDE_CON = F.IDE_CON
                JOIN CLIENTE CL ON CL.IDE_CLI = F.IDE_CLI
                WHERE YEAR(F.FFI_FIC)=@AÑO
GO
--PRUEBA: SP_LISTAFICHASRECLAMOXAÑO 2008
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
```

```
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjFichasReclamoxAño
{
    class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                   .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjFichasReclamoxAño
{
    class LogicaNegocio
    {
      //Definicion GLOBAL
```

```
        Conexion objCon = new Conexion();
        SqlConnection cn = new SqlConnection();

        //Metodos que lista los años en forma descendente
        public DataTable listaAño()
        {
          cn = objCon.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTAAÑOS", cn);
          DataTable dt = new DataTable();
          da.Fill(dt);
          return dt;
        }

        //Metodos que lista los datos del cliente
        public DataTable listaFichasxAño(int año)
        {
          cn = objCon.getConecta();
          cn.Open();
         SqlDataAdapter da = new SqlDataAdapter("SP_LISTAFICHASRECLAMOXAÑO", cn);
          da.SelectCommand.CommandType = CommandType.StoredProcedure;
         da.SelectCommand.Parameters.Add("año", SqlDbType.Int).Value = año;

          DataTable dt = new DataTable();
          da.Fill(dt);
          return dt;
        }
      }
}
```

8. El código del formulario frmConexionClase completo 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.Windows.Forms;

namespace pjFichasReclamoxAño
{
    public partial class frmFichasReclamoxAño : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmFichasReclamoxAño()
        {
```

```
            InitializeComponent();
        }

        private void frmFichasReclamoxAño_Load(object sender, EventArgs e)
        {
            llenaAños();
        }

        void llenaAños()
        {
            cboAño.DataSource = objL.listaAño();
            cboAño.DisplayMember = "AÑO";
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            int año = int.Parse(cboAño.Text.ToString());
            dgFichas.DataSource = objL.listaFichasxAño(año);
        }
    }
}
```

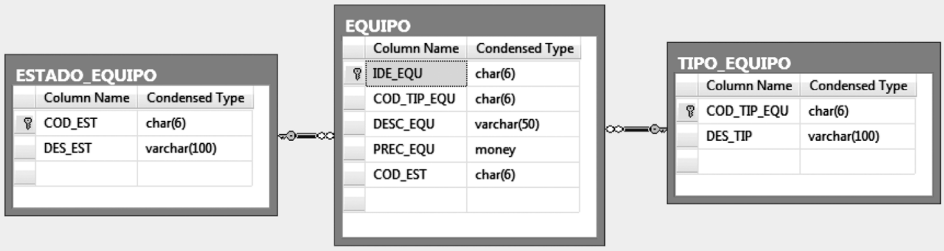
Caso desarrollado 12 : Mantenimiento de registros de equipo

Implementar una aplicación Windows Forms que permita realizar el mantenimiento de la tabla Equipo.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server procedimientos almacenados que permitan listar, insertar, modificar y eliminar registros de la tabla Equipo, además de implementar procedimientos almacenados que permitan listar los datos de los tipos y estados de los equipos.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar los métodos necesarios para asociar a todos los procedimientos almacenados necesarios para el mantenimiento de registros de la tabla Equipo.
- Usar el control ToolStrip para las opciones que presenta el mantenimiento de registro de equipos.

- Tomar en cuenta las siguiente relación de tablas:



GUI propuesto:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjMantenimientoEquipo.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System. Configuration.
3. Modifique el nombre del formulario a frmMantenimientoEquipo.

4. En SQL Server implemente los siguientes procedimientos almacenados:

```
--SP_LISTAEQUIPOS
IF OBJECT_ID("SP_LISTAEQUIPOS")IS NOT NULL
        DROP PROC SP_LISTAEQUIPOS
GO
CREATE PROC SP_LISTAEQUIPOS
AS
        SELECT E.IDE_EQU AS CODIGO,
               E.DESC_EQU AS DESCRIPCION,
               T.DES_TIP AS TIPO,
               E.PREC_EQU AS PRECIO,
               ES.DES_EST AS ESTADO
               FROM EQUIPO E
               JOIN TIPO_EQUIPO T ON E.COD_TIP_EQU=T.COD_TIP_EQU
               JOIN ESTADO_EQUIPO ES ON E.COD_EST=ES.COD_EST
GO

--SP_LISTATIPOEQUIPOS
IF OBJECT_ID("SP_LISTATIPOEQUIPOS")IS NOT NULL
        DROP PROC SP_LISTATIPOEQUIPOS
GO
CREATE PROC SP_LISTATIPOEQUIPOS
AS
        SELECT T.COD_TIP_EQU AS CODIGO,
               T.DES_TIP AS TIPO
               FROM TIPO_EQUIPO T
GO

--SP_LISTAESTADOS
IF OBJECT_ID("SP_LISTAESTADOS")IS NOT NULL
        DROP PROC SP_LISTAESTADOS
GO
CREATE PROC SP_LISTAESTADOS
AS
        SELECT ES.COD_EST AS CODIGO,
               ES.DES_EST AS ESTADO
               FROM ESTADO_EQUIPO ES
GO

--SP_ULTIMOEQUIPO
IF OBJECT_ID("SP_ULTIMOEQUIPO")IS NOT NULL
        DROP PROC SP_ULTIMOEQUIPO
GO
CREATE PROC SP_ULTIMOEQUIPO
AS
        SELECT TOP 1 E.IDE_EQU AS CODIGO
               FROM EQUIPO E
               ORDER BY 1 DESC
```

```
GO

--SP_NUEVOEQUIPO
IF OBJECT_ID("SP_NUEVOEQUIPO")IS NOT NULL
        DROP PROC SP_NUEVOEQUIPO
GO
CREATE PROC SP_NUEVOEQUIPO(@COD CHAR(6), @TIP CHAR(6),@DES VARCHAR(40),@
PRE MON-EY, @EST INT)
AS
        INSERT INTO EQUIPO VALUES (@COD,@TIP,@DES,@PRE,@EST)
GO

--SP_ACTUALIZAEQUIPO
IF OBJECT_ID("SP_ACTUALIZAEQUIPO")IS NOT NULL
        DROP PROC SP_ACTUALIZAEQUIPO
GO
CREATE PROC SP_ACTUALIZAEQUIPO(@COD CHAR(6), @TIP CHAR(6),@DES VARCHAR(40),@
PRE MONEY, @EST INT)
AS
        UPDATE EQUIPO
                SET COD_TIP_EQU=@TIP,DESC_EQU=@DES,PREC_EQU=@PRE,COD_EST=@EST
                WHERE IDE_EQU=@COD
GO

--SP_ELIMINAEQUIPO
IF OBJECT_ID("SP_ELIMINAEQUIPO")IS NOT NULL
        DROP PROC SP_ELIMINAEQUIPO
GO
CREATE PROC SP_ELIMINAEQUIPO(@COD CHAR(6))
AS
        DELETE EQUIPO WHERE IDE_EQU=@COD
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjMantenimientoEquipo
{
    class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                    .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjMantenimientoEquipo
{
    class LogicaNegocio
    {
        //Definicion GLOBAL
        Conexion objCon = new Conexion();
        SqlConnection cn = new SqlConnection();

        string mensaje;

        //Método que lista los equipos
        public DataTable listaEquipos()
        {
            cn = objCon.getConecta();
            SqlDataAdapter da = new SqlDataAdapter("SP_LISTAEQUIPOS", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }
```

```
        //Método que lista los estados
        public DataTable listaEstado()
        {
            cn = objCon.getConecta();
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTAESTADOS", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }
        //Método que lista los tipos
        public DataTable listaTipos()
        {
            cn = objCon.getConecta();
            SqlDataAdapter da = new SqlDataAdapter("SP_LISTATIPOEQUIPOS", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

        //Método que genere un nuevo codigo de Equipo
        public string generaCodigo()
        {
            cn = objCon.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_ULTIMOEQUIPO", cn);
            return "EQ" + (int.Parse(cmd.ExecuteScalar().ToString()
                                    .Substring(2, 4)) + 1).ToString("0000");
        }

        //Método que registra un nuevo equipo
        public string nuevoEquipo(string codigo, string tipo, string descripcion,
                                  double precio, string estado)
        {
            mensaje = "";
            cn = objCon.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_NUEVOEQUIPO", cn);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add("cod", SqlDbType.Char).Value = codigo;
            cmd.Parameters.Add("tip", SqlDbType.Char).Value = tipo;
            cmd.Parameters.Add("des", SqlDbType.VarChar).Value = descripcion;
            cmd.Parameters.Add("pre", SqlDbType.Money).Value = precio;
            cmd.Parameters.Add("est", SqlDbType.Char).Value = estado;

            try
            {
                int n = cmd.ExecuteNonQuery();
              mensaje = n.ToString() + " EQUIPO REGISTRADO CORRECTAMENTE";
            }
```

```
        catch (SqlException ex)
        {
            mensaje = ex.Message;
        }
        finally
        {
            cn.Close();
        }
        return mensaje;
    }

    //Método que actualiza los datos de un equipo
    public string actualizaEquipo(string codigo, string tipo,
                      string descripcion, double precio, string estado)
    {
        mensaje = "";
        cn = objCon.getConecta();
        cn.Open();
        SqlCommand cmd = new SqlCommand("SP_ACTUALIZAEQUIPO", cn);
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.Add("cod", SqlDbType.Char).Value = codigo;
        cmd.Parameters.Add("tip", SqlDbType.Char).Value = tipo;
        cmd.Parameters.Add("des", SqlDbType.VarChar).Value = descripcion;
        cmd.Parameters.Add("pre", SqlDbType.Money).Value = precio;
        cmd.Parameters.Add("est", SqlDbType.Char).Value = estado;

        try
        {
            int n = cmd.ExecuteNonQuery();
         mensaje = n.ToString() + " EQUIPO ACTUALIZADO CORRECTAMENTE";
        }
        catch (SqlException ex)
        {
            mensaje = ex.Message;
        }
        finally
        {
            cn.Close();
        }
        return mensaje;
    }

    //Método que elimina un registro de equipo
    public string eliminaEquipo(string codigo)
    {
        mensaje = "";
        cn = objCon.getConecta();
        cn.Open();
```

```
            SqlCommand cmd = new SqlCommand("SP_ELIMINAEQUIPO", cn);
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Parameters.Add("cod", SqlDbType.Char).Value = codigo;
            try
            {
                int n = cmd.ExecuteNonQuery();
              mensaje = n.ToString() + " EQUIPO ELIMINADO CORRECTAMENTE";
            }
            catch (SqlException ex)
            {
                mensaje = ex.Message;
            }
            finally
            {
                cn.Close();
            }
            return mensaje;
        }
    }
}
```

El código del formulario frmMantenimientoEquipo completo 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.Windows.Forms;
using System.Globalization;

namespace pjMantenimientoEquipo
{
    public partial class frmMantenimientoEquipo : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmMantenimientoEquipo()
        {
            InitializeComponent();
        }

       private void frmMantenimientoEquipo_Load(object sender, EventArgs e)
        {
            llenaTipo();
            llenaEstado();
```

```
            llenaEquipos();
            lblCodigo.Text = generacodigo();
        }

        private void tsNuevo_Click(object sender, EventArgs e)
        {
            lblCodigo.Text = generacodigo();
            limpiarControles();
        }


        private void tsGrabar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
                //Capturando los datos del formulario
                string codigo = lblCodigo.Text;
                string descripcion = txtDescripcion.Text;
                string tipo = cboTipo.SelectedValue.ToString();
                string estado = cboEstado.SelectedValue.ToString();
                double precio = double.Parse(txtPrecio.Text);

                //Grabando el nuevo equipo
              string mensaje = objL.nuevoEquipo(codigo, tipo, descripcion,
                                                  precio, estado);
                MessageBox.Show(mensaje);
                llenaEquipos();
            }
            else
                MessageBox.Show("El error se encuentra" + valida());
        }

        private void tsModificar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
                //Capturando los datos del formulario
                string codigo = lblCodigo.Text;
                string descripcion = txtDescripcion.Text;
                string tipo = cboTipo.SelectedValue.ToString();
                string estado = cboEstado.SelectedValue.ToString();
                double precio = double.Parse(txtPrecio.Text);

                //Modificando
                string mensaje = objL.actualizaEquipo(codigo, tipo, descripcion,
                                                    precio, estado);
                MessageBox.Show(mensaje);
                llenaEquipos();
            }
            else
                MessageBox.Show("El error se encuentra" + valida());
        }
```

```
        private void tsEliminar_Click(object sender, EventArgs e)
        {
            //Capturando los datos del formulario
            string codigo = lblCodigo.Text;

            //Eliminando
            string mensaje = objL.eliminaEquipo(codigo);
            MessageBox.Show(mensaje);
            llenaEquipos();
        }

        private void tsSalir_Click(object sender, EventArgs e)
        {
            this.Close();
        }

      private void dgEquipos_MouseDoubleClick(object sender, MouseEventArgs e)
        {
           lblCodigo.Text = dgEquipos.CurrentRow.Cells[0].Value.ToString();
           txtDescripcion.Text = dgEquipos.CurrentRow.Cells[1].Value.ToString();
           cboTipo.Text = dgEquipos.CurrentRow.Cells[2].Value.ToString();
           cboEstado.Text = dgEquipos.CurrentRow.Cells[4].Value.ToString();
           txtPrecio.Text = dgEquipos.CurrentRow.Cells[3].Value.ToString();
        }

        void llenaTipo()
        {
            cboTipo.DataSource = objL.listaTipos();
            cboTipo.DisplayMember = "TIPO";
            cboTipo.ValueMember = "CODIGO";
        }

        void llenaEstado()
        {
            cboEstado.DataSource = objL.listaEstado();
            cboEstado.DisplayMember = "ESTADO";
            cboEstado.ValueMember = "CODIGO";

        }
        void llenaEquipos()
        {
            dgEquipos.DataSource = objL.listaEquipos();
        }

        string generacodigo()
        {
            return objL.generaCodigo();
        }
```

```
        void limpiarControles()
        {
            txtDescripcion.Clear();
            cboEstado.SelectedIndex = -1;
            cboTipo.SelectedIndex = -1;
            txtPrecio.Clear();
            txtDescripcion.Focus();
        }
        string valida()
        {
            if (txtDescripcion.Text.Trim().Length == 0)
            {
                txtDescripcion.Clear();
                txtDescripcion.Focus();
                return " la descripcion del equipo";
            }
            else if (cboTipo.SelectedIndex == -1)
            {
                cboTipo.Focus();
                return " el tipo de equipo";
            }
            else if (cboEstado.SelectedIndex == -1)
            {
                cboEstado.Focus();
                return " el estado del equipo";
            }
            else if (txtPrecio.Text.Trim().Length == 0)
            {
                txtPrecio.Clear();
                txtPrecio.Focus();
                return " el precio del equipo";
            }
            else
                return "";
        }
      private void txtPrecio_KeyPress(object sender, KeyPressEventArgs e)
        {
        CultureInfo cc = System.Threading.Thread.CurrentThread.CurrentCulture;
           if (char.IsNumber(e.KeyChar) ||
              e.KeyChar.ToString() == cc.NumberFormat.NumberDecimalSeparator)
                e.Handled = false;
            else
                e.Handled = true;
        }
    }
}
```

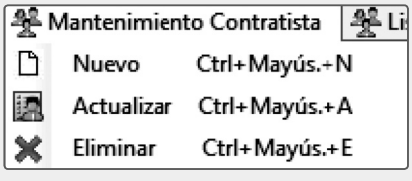
Caso desarrollado 13 : Mantenimiento de registro de contratista

Implementar una aplicación Windows Forms que permita realizar el mantenimiento de registro de la tabla Contratista.

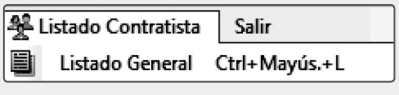
Se tiene en cuenta lo siguiente:

- Implementar en SQL Server procedimientos almacenados que permitan listar, insertar, modificar y eliminar un registro de contratista.
- Configurar la cadena de conexión en el archivo app.config.
- Crear la clase Conexión e implementar el método getConecta, el cual permita asociarse a la cadena de conexión especificada en el app.config.
- Crear la clase LogicaNegocio e implementar los métodos necesarios para el mantenimiento de registro de contratistas.
- Agregar al proyecto un formulario primario MDI para el menú de opciones de la aplicación.

El menú **Mantenimiento Contratista** presenta las siguientes opciones:



El menú **Listado Contratista** presenta las siguientes opciones:



GUI propuesto para las opciones del menú:

- Formulario de listado de contratistas.

- Formulario de registro del nuevo contratista.

- Formulario de actualización de datos del contratista.

- Formulario de eliminación de registro de contratista.

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjMantenimientoContratista.
2. Agregue la referencia System.Configuration, presionando clic derecho sobre el proyecto, luego en **Agregar** > **Referencia**. Después, active el check en System.Configuration.

3. Agregue cuatro formularios para el mantenimiento de registros y un formulario primario MDI.
4. En SQL Server implemente los siguientes procedimientos almacenados:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS CONTRATISTAS
IF OBJECT_ID("SP_LISTACONTRATISTA")IS NOT NULL
       DROP PROC SP_LISTACONTRATISTA
GO
CREATE PROC SP_LISTACONTRATISTA
AS
        SELECT C.IDE_CON AS CODIGO,
               C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
               C.FON_CON AS TELEFONO,
               C.EMA_CON AS CORREO
               FROM CONTRATISTA C
GO

--PROCEDIMIENTO ALMACENADO PARA EL ULTIMO CODIGO DE CONTRATISTA
IF OBJECT_ID("SP_ULTIMOCODIGO")IS NOT NULL
       DROP PROC SP_ULTIMOCODIGO
GO
CREATE PROC SP_ULTIMOCODIGO
AS
        SELECT TOP 1 C.IDE_CON
               FROM CONTRATISTA C
               ORDER BY C.IDE_CON DESC
GO

--PROCEDIMIENTO ALMACENADO PARA UN NUEVO CONTRATISTA
IF OBJECT_ID("SP_NUEVOCONTRATISTA")IS NOT NULL
       DROP PROC SP_NUEVOCONTRATISTA
GO
CREATE PROC SP_NUEVOCONTRATISTA(@COD CHAR(6),@NOM VARCHAR(30),
                                @PAT VARCHAR(20),@MAT VARCHAR(20),
                                @FON VARCHAR(15),@EMA VARCHAR(50))
AS
       INSERT INTO CONTRATISTA
              VALUES (@COD,@NOM,@PAT,@MAT,@FON,@EMA)
GO

--PROCEDIMIENTO ALMACENADO QUE ACTUALIZA LOS DATOS DEL CONTRATISTA
IF OBJECT_ID("SP_ACTUALIZACONTRATISTA")IS NOT NULL
       DROP PROC SP_ACTUALIZACONTRATISTA
GO
CREATE PROC SP_ACTUALIZACONTRATISTA(@COD CHAR(6),@NOM VARCHAR(30),
                                    @PAT VARCHAR(20),@MAT VARCHAR(20),
                                    @FON VARCHAR(15),@EMA VARCHAR(50))
AS
       UPDATE CONTRATISTA
              SET NOM_CON=@NOM,PAT_CON=@PAT,MAT_CON=@MAT,
```

```
                  FON_CON=@FON,EMA_CON=@EMA
              WHERE IDE_CON=@COD
GO

--PROCEDIMIENTO ALMACENADO QUE PERMITE ELIMINAR UN REGISTRO DE CONTRATISTA
IF OBJECT_ID("SP_ELIMINACONTRATISTA")IS NOT NULL
       DROP PROC SP_ELIMINACONTRATISTA
GO
CREATE PROC SP_ELIMINACONTRATISTA(@COD CHAR(6))
AS
       DELETE CONTRATISTA
        WHERE IDE_CON = @COD
GO

--PROCEDIMIENTO ALMACENADO QUE MUESTRA LOS DATOS DE UN CONTRATISTA
IF OBJECT_ID("SP_BUSCACONTRATISTA")IS NOT NULL
       DROP PROC SP_BUSCACONTRATISTA
GO
CREATE PROC SP_BUSCACONTRATISTA(@COD CHAR(6))
AS
        SELECT C.IDE_CON AS CODIGO,
              C.NOM_CON AS NOMBRE,
              C.PAT_CON AS PATERNO,
              C.MAT_CON AS MATERNO,
              C.FON_CON AS TELEFONO,
              C.EMA_CON AS CORREO
              FROM CONTRATISTA C
              WHERE IDE_CON = @COD
GO
```

5. Configure la cadena de conexión en el archivo app.config desde el Explorador de Soluciones.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="server=.;
                                     database=contrato;
                                     integrated security=SSPI"/>
  </connectionStrings>
</configuration>
```

6. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;

namespace pjMantenimientoContratista
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

7. Agregue la clase LogicaNegocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace pjMantenimientoContratista
{
    public class LogicaNegocio
    {
        //Definicion GLOBAL
        Conexion objCon = new Conexion();
        SqlConnection cn = new SqlConnection();

        string mensaje;

        //Método que lista los contratistas
        public DataTable listaContratistas()
```

```
        {
            cn = objCon.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

        //Método que genere un nuevo codigo de contratista
        public string generaCodigo()
        {
            cn = objCon.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_ULTIMOCODIGO", cn);
            return "CON" + (int.Parse(cmd.ExecuteScalar().ToString()
                                    .Substring(3, 3)) + 1).ToString("000");
        }

        //Método que registra un nuevo contratista
        public string nuevoContratista(Contratista objC)
        {
            mensaje = "";
            cn = objCon.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_NUEVOCONTRATISTA", cn);
            cmd.CommandType = CommandType.StoredProcedure;
           cmd.Parameters.Add("COD", SqlDbType.VarChar).Value = objC.codigo;
           cmd.Parameters.Add("NOM", SqlDbType.VarChar).Value = objC.nombres;
           cmd.Parameters.Add("PAT", SqlDbType.VarChar).Value = objC.paterno;
           cmd.Parameters.Add("MAT", SqlDbType.VarChar).Value = objC.materno;
           cmd.Parameters.Add("FON", SqlDbType.VarChar).Value = objC.telefono;
           cmd.Parameters.Add("EMA", SqlDbType.VarChar).Value = objC.correo;

            try
            {
                int n = cmd.ExecuteNonQuery();
             mensaje = n.ToString() + " CONTRATISTA REGISTRADO CORRECTAMENTE";
            }
            catch (SqlException ex)
            {
                mensaje = ex.Message;
            }
            finally
            {
                cn.Close();
            }
            return mensaje;
        }
```

```
//Método que busca los datos del contratista
public DataTable buscaContratista(Contratista objC)
{
    cn = objCon.getConecta();
    cn.Open();
  SqlDataAdapter da = new SqlDataAdapter("SP_BUSCACONTRATISTA", cn);
    da.SelectCommand.CommandType = CommandType.StoredProcedure;
   da.SelectCommand.Parameters.Add("COD", SqlDbType.VarChar).Value =
                                                          objC.codigo;

    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
}

//Método que actualiza los datos del contratista
public string actualizaContratista(Contratista objC)
{
    mensaje = "";
    cn = objCon.getConecta();
    cn.Open();
   SqlCommand cmd = new SqlCommand("SP_ACTUALIZACONTRATISTA", cn);
   cmd.CommandType = CommandType.StoredProcedure;
   cmd.Parameters.Add("COD", SqlDbType.VarChar).Value = objC.codigo;
   cmd.Parameters.Add("NOM", SqlDbType.VarChar).Value = objC.nombres;
   cmd.Parameters.Add("PAT", SqlDbType.VarChar).Value = objC.paterno;
   cmd.Parameters.Add("MAT", SqlDbType.VarChar).Value = objC.materno;
   cmd.Parameters.Add("FON", SqlDbType.VarChar).Value = objC.telefono;
   cmd.Parameters.Add("EMA", SqlDbType.VarChar).Value = objC.correo;

    try
    {
       int n = cmd.ExecuteNonQuery();
     mensaje = n.ToString() + " CONTRATISTA ACTUALIZADO CORRECTAMENTE";
    }
    catch (SqlException ex)
    {
        mensaje = ex.Message;
    }
    finally
    {
        cn.Close();
    }
    return mensaje;
}
```

```
        //Método que elimina un contratista
        public string eliminaEquipo(Contratista objC)
        {
            mensaje = "";
            cn = objCon.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_ELIMINACONTRATISTA", cn);
            cmd.CommandType = CommandType.StoredProcedure;
           cmd.Parameters.Add("cod", SqlDbType.Char).Value = objC.codigo;
            try
            {
                int n = cmd.ExecuteNonQuery();
              mensaje = n.ToString() + " CONTRATISTA ELIMINADO CORRECTAMENTE";
            }
            catch (SqlException ex)
            {
                mensaje = ex.Message;
            }
            finally
            {
                cn.Close();
            }
            return mensaje;
        }
    }
}
```

El código del formulario frmListadoContratista completo 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.Windows.Forms;

namespace pjMantenimientoContratista
{
    public partial class frmListadoContratista : Form
    {
        LogicaNegocio objL = new LogicaNegocio();
```

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

       private void frmListadoContratista_Load(object sender, EventArgs e)
        {
            llenaContratista();
        }
        void llenaContratista()
        {
            dgContratistas.DataSource = objL.listaContratistas();
        }
    }
}
```

El código del formulario frmNuevoContratista completo 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.Windows.Forms;

namespace pjMantenimientoContratista
{
    public partial class frmNuevoContratista : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmNuevoContratista()
        {
            InitializeComponent();
        }

        private void frmNuevoContratista_Load(object sender, EventArgs e)
        {
            generaCodigo();
        }

        void generaCodigo()
        {
            lblCodigo.Text = objL.generaCodigo();
        }
```

```
        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            Contratista objCo = new Contratista();

            //Enviando los valores al objeto de la clase Contratista
            objCo.codigo = lblCodigo.Text;
            objCo.nombres = txtNombres.Text;
            objCo.paterno = txtPaterno.Text;
            objCo.materno = txtMaterno.Text;
            objCo.telefono = txtFono.Text;
            objCo.correo = txtCorreo.Text;

            //Grabando el nuevo registro de contratista
            string mensaje = objL.nuevoContratista(objCo);
            MessageBox.Show(mensaje);
            limpiarControles();
        }

        private void btnAnular_Click(object sender, EventArgs e)
        {
            limpiarControles();
        }

        void limpiarControles()
        {
            generaCodigo();
            txtNombres.Clear();
            txtPaterno.Clear();
            txtMaterno.Clear();
            txtFono.Clear();
            txtCorreo.Clear();
            txtNombres.Focus();
        }

        private void btnListado_Click(object sender, EventArgs e)
        {
          frmListadoContratista objListado = new frmListadoContratista();
            objListado.Show();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

El código del formulario frmActualizaContratista completo 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.Windows.Forms;

namespace pjMantenimientoContratista
{
    public partial class frmActualizaContratista : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

        public frmActualizaContratista()
        {
            InitializeComponent();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            Contratista objCo = new Contratista();
            objCo.codigo = txtCodigo.Text;

            DataRow fila = objL.buscaContratista(objCo).Rows[0];

            txtNombres.Text = fila[1].ToString();
            txtPaterno.Text = fila[2].ToString();
            txtMaterno.Text = fila[3].ToString();
            txtFono.Text = fila[4].ToString();
            txtCorreo.Text = fila[5].ToString();

        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            Contratista objCo = new Contratista();
            objCo.codigo = txtCodigo.Text;
            objCo.nombres = txtNombres.Text;
            objCo.paterno = txtPaterno.Text;
            objCo.materno = txtMaterno.Text;
            objCo.telefono = txtFono.Text;
            objCo.correo = txtCorreo.Text;

            string mensaje = objL.actualizaContratista(objCo);
            MessageBox.Show(mensaje);
        }
```

```
        private void btnAnular_Click(object sender, EventArgs e)
        {
            limpiarControles();
        }
        void limpiarControles()
        {
            txtCodigo.Clear();
            txtNombres.Clear();
            txtPaterno.Clear();
            txtMaterno.Clear();
            txtFono.Clear();
            txtCorreo.Clear();
            txtCodigo.Focus();
        }

        private void btnListado_Click(object sender, EventArgs e)
        {
          frmListadoContratista objListado = new frmListadoContratista();
            objListado.Show();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

El código del formulario frmEliminaContratista completo 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.Windows.Forms;

namespace pjMantenimientoContratista
{
    public partial class frmEliminaContratista : Form
    {
        LogicaNegocio objL = new LogicaNegocio();

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

```
        private void btnListado_Click(object sender, EventArgs e)
        {
          frmListadoContratista objListado = new frmListadoContratista();
            objListado.Show();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            Contratista objCo = new Contratista();
            objCo.codigo = txtCodigo.Text;

            DataRow fila = objL.buscaContratista(objCo).Rows[0];

            lblNombres.Text = fila[1].ToString();
            lblPaterno.Text = fila[2].ToString();
            lblMaterno.Text = fila[3].ToString();
            lblTelefono.Text = fila[4].ToString();
            lblCorreo.Text = fila[5].ToString();
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            Contratista objCo = new Contratista();
            objCo.codigo = txtCodigo.Text;

            string mensaje = objL.eliminaEquipo(objCo);
            MessageBox.Show(mensaje);
            limpiarControles();
        }

        private void btnAnular_Click(object sender, EventArgs e)
        {
            limpiarControles();
        }

        private void btnSalir_Click(object sender, EventArgs e)
        {
            this.Close();
        }
        void limpiarControles()
        {
            txtCodigo.Clear();
            lblNombres.Text = "";
            lblPaterno.Text = "";
            lblMaterno.Text = "";
            lblTelefono.Text = "";
            lblCorreo.Text = "";
            txtCodigo.Focus();
        }

    }
}
```

El código del formulario primario MDI MenuPrincipal completo 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.Windows.Forms;

namespace pjMantenimientoContratista
{
    public partial class MenuPrincipal : Form
    {
        public MenuPrincipal()
        {
            InitializeComponent();
        }

       private void nuevoToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (Form formularios in MdiChildren)
            {
                formularios.Close();
            }
            frmNuevoContratista childForm = new frmNuevoContratista();
            childForm.MdiParent = this;
            childForm.Show();
        }

       private void actualizarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (Form formularios in MdiChildren)
            {
                formularios.Close();
            }
          frmActualizaContratista childForm = new frmActualizaContratista();
            childForm.MdiParent = this;
            childForm.Show();
        }

       private void eliminarToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (Form formularios in MdiChildren)
            {
                formularios.Close();
            }
           frmEliminaContratista childForm = new frmEliminaContratista();
            childForm.MdiParent = this;
```

```
            childForm.Show();
        }

    private void listadoGeneralToolStripMenuItem_Click(object sender,EventArgs e)
        {
            foreach (Form formularios in MdiChildren)
            {
                formularios.Close();
            }
           frmListadoContratista childForm = new frmListadoContratista();
            childForm.MdiParent = this;
            childForm.Show();
        }

       private void salirToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

EDITORIAL MACRO®

EDITORIAL
MACRO®

5
Capítulo

TableAdapter - LinQ to SQL

Capacidad terminal:

Implementar aplicaciones de consulta mediante el uso del TableAdapter y LinQ, además de poder agregar registros mediante el uso de clase de LinQ to SQL.

Contenido

EDITORIAL MACRO®

- TableAdapter
  Caso desarrollado 1: Usando TableAdapter y consulta simple - Listado de contratistas
  Caso desarrollado 2: Usando TableAdapter y procedimiento almacenado - Listado de equipos
  Caso desarrollado 3: Usando TableAdapter y procedimiento almacenado - Listado de equipos por estado y tipo
- LinQ
- Implementación de una consulta con LinQ
- Operaciones básicas de una consulta LinQ to SQL
- Clase DataContext
  Caso desarrollado 4: LinQ to SQL - Listado de contratistas
  Caso desarrollado 5: LinQ to SQL con procedimientos almacenados - Listado de equipos
  Caso desarrollado 6: LinQ to SQL - Listado de fichas de reclamo por cliente
  Caso desarrollado 7: LinQ to SQL - Listado de fichas de reclamo por año
  Caso desarrollado 8: LinQ to SQL - Registro del nuevo equipo

EDITORIAL
MACRO®

5.1 TableAdapter

Los TableAdapters tienen la finalidad de comunicar una aplicación con una base de datos. Desde aquí se podrá ejecutar sentencias o procedimientos almacenados, los cuales obtendrán los valores desde la base de datos formando nuevas tablas y desde aquí se podrán usar en las aplicaciones. En otras ocasiones se podrán utilizar para registrar información en las tablas, las cuales replicarán en la base de datos.

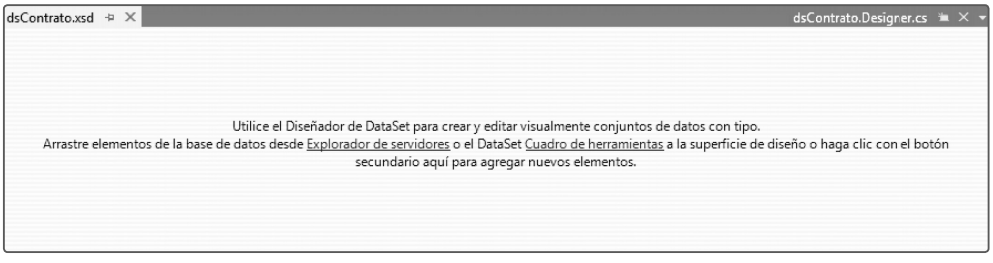
A veces, se puede asociar el término TableAdapter a DataAdapter, ya que ambos cuentan con un objeto de conexión integrado y pueden contener muchas consultas. Observe el código de configuración que realiza el TableAdapter:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
      <add name="pjTableAdapter.Properties.Settings. CONTRATOConnectionString"
          connectionString="Data Source=.;Initial Catalog=CONTRATO;
                            Integrated Security=True"
       providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
</configuration>
```

Figura 5.1 Código fuente del archivo app.config

Un TableAdapter presenta un entorno de diseño en el que se pueden incorporar las consultas o los procedimientos almacenados y desde el que se puede hacer referencia mediante objetos simples en las aplicaciones. Normalmente, contiene métodos como Update y Fill para capturar y actualizar datos en una base de datos.

Para crear un TableAdapter, se necesita un diseñador de DataSet, que debe ser agregado a la aplicación y mediante un asistente se agregará un TableAdapter. Haciendo las configuraciones necesarias, se llegará a conectar a la base de datos origen.



En el entorno solo se debe presionar clic derecho en el fondo para poder agregar un TableAdapter. Esto podrá realizarse directamente por una consulta o mediante un procedimiento almacenado, previamente implementado en SQL Server.

Dependiendo de la sentencia o procedimiento, se mostrará en el diseñador el resultado de la consulta como un objeto, tal como se muestra en la siguiente imagen:

Se debe considerar que se muestran las columnas como se especificaron en la consulta, además de mostrar el campo clave y los métodos de acceso como Fill y GetData(). Es a partir de estos métodos que se puede obtener la información contenida en el TableAdapter.

Caso desarrollado 1 : Usando TableAdapter y consulta simple - Listado de contratistas

Implementar una aplicación Windows Forms que permita listar los registros de los contratistas.

Se tiene en cuenta lo siguiente:

- Agregar un objeto DataSet al proyecto y agregar la tabla Contratista.
- El listado se debe realizar sobre el control DataGridView con los campos código, nombre completo del contratista, teléfono y correo electrónico del contratista.
- Tomar en cuenta la siguiente estructura de la tabla:

CONTRATISTA

Nombre de columna	Tipo comprimido
IDE_CON	char(6)
NOM_CON	varchar(30)
PAT_CON	varchar(20)
MAT_CON	varchar(20)
FON_CON	varchar(15)
EMA_CON	varchar(50)

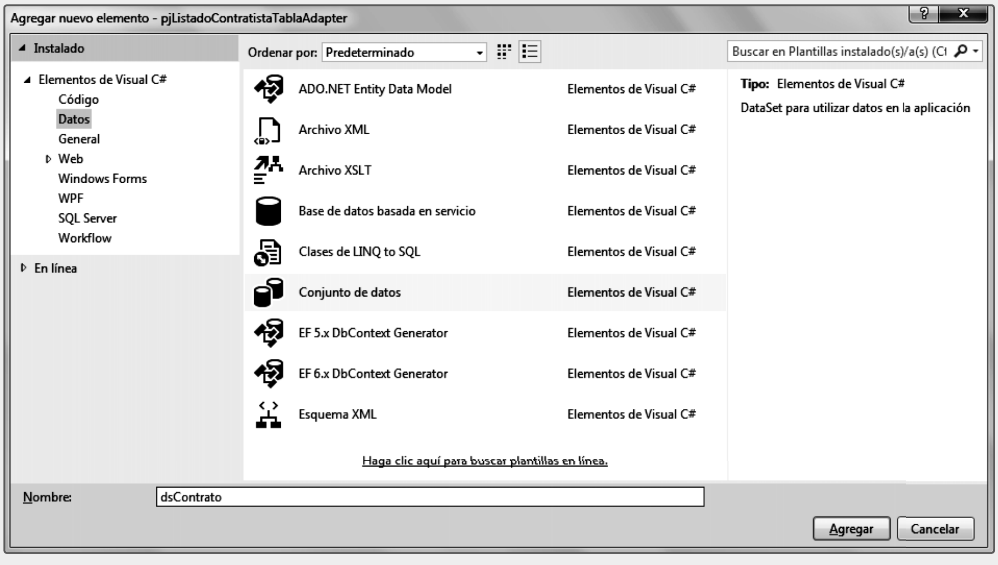
GUI propuesto:

CODIGO	CONTRATISTA	TELEFONO	CORREO
CON001	PEDRO LOPEZ SANCHEZ	983645364	PLOPEZ@HOTMAIL.COM
CON002	LUIGUI JARAMILLO ORTIZ	98346545	LJARAMILLO@HOTMAIL.COM
CON003	OSCAR ZAMORA ROSAS	946566755	OZAMORA@HOTMAIL.COM
CON004	DICK GALVEZ CARBAJAL	998765436	DGALVEZ@GMAIL.COM
CON005	RICHARD VELEZMORO SO	945678899	RVELEZMORO@HOTMAL.COM
CON006	NELSON MEJIA MEJIA	923456788	NMEJIA@HOTMAIL.COM
CON007	TOMAS GONZALES SANCHEZ	46578006	TGONZALES@GMAIL.COM
CON008	CRISTHIAN HUARAC TORRES	5976534	CHUARAC@HOTMAIL.COM
CON009	JORGE CASIMIRO SOTO	34567898	JCASIMIRO@HOTMAIL.COM

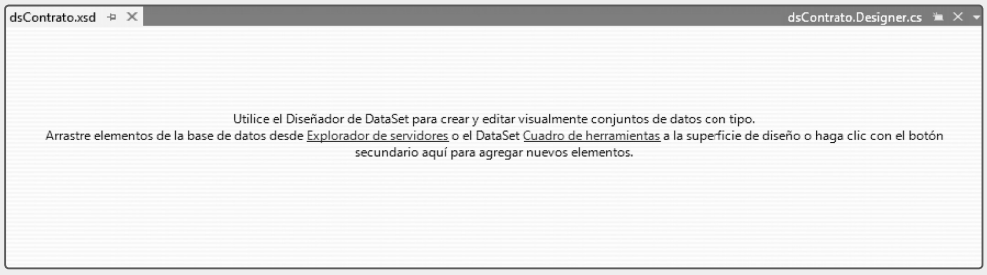
Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjListadoContratistaTableAdapter.

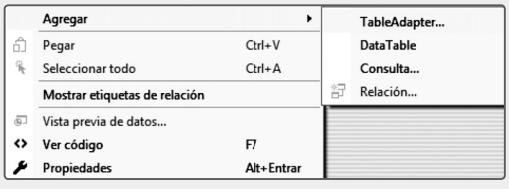
2. Haga clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento**. Seleccione el elemento **Datos** > **Conjunto de datos** y asigne un nombre. En este caso, se le asigna dsContrato por tratarse de obtener los datos de la base de datos Contrato.



Inicialmente, el entorno de DataSet se muestra vacío, ya que se debe agregar un elemento como una consulta o un procedimiento almacenado:

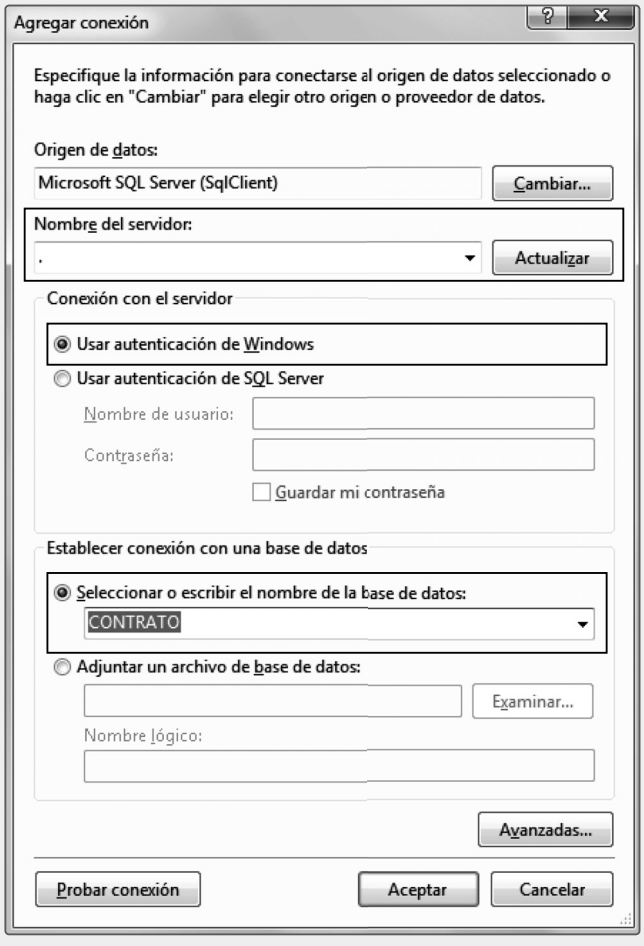


3. Agregue un TableAdapter al entorno, presionando clic derecho sobre el fondo del entorno y seleccione **Agregar > TableAdapter….**



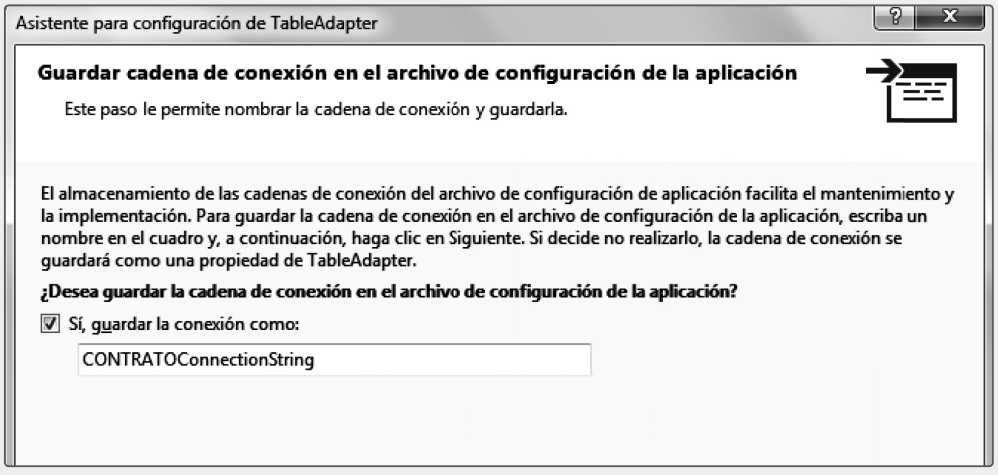
4. Elija la conexión de datos. Es decir, en este paso se debe seleccionar el servidor y la base de datos de donde se obtendrá los datos a mostrar:

5. Ahora, debe agregar una nueva conexión para configurar el acceso al servidor. Para esto, debe presionar el botón **Nueva conexión…** Si es la primera vez que agrega un TableAdapter, se muestra la ventana de selección de origen de datos, en la cual debe seleccionar **Microsoft SQL Server (SqlClient)**. Luego, seleccione un nuevo servidor, el tipo de autenticación y la base de datos desde la siguiente ventana:

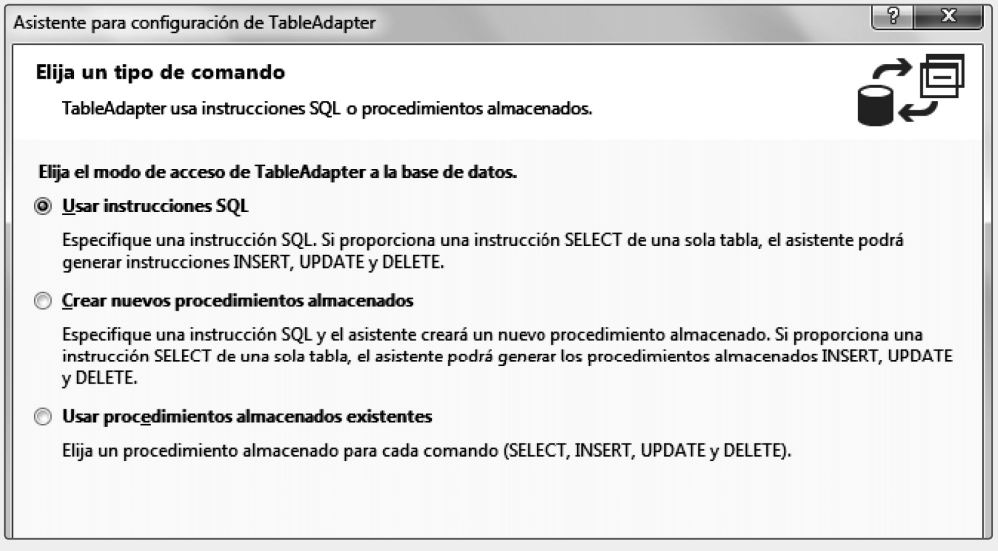


6. La asignación del nombre del servidor se puede dar de varias formas como, por ejemplo, colocando el nombre del servidor de red de donde provienen los datos, el nombre de la computadora o el número de IP. Para este caso, debe usar el punto como nombre por defecto del servidor actual. Como segundo paso, debe seleccionar el modo de acceso al servidor. Se presentan dos opciones: **Usar autenticación de Windows**, que es el valor estándar, y **Usar autenticación de SQL Server**, el cual necesita de usuario y clave. Finalmente, seleccione la base de datos. Si esta aparece en la lista, será un indicador de que todo lo seleccionado hasta el momento es lo correcto. Presione **Aceptar** para finalizar la configuración de la conexión.

7. Luego, el asistente pregunta si se quiere guardar la configuración realizada. Para lo cual, se dejará como se encuentra y solo se presiona el botón siguiente:



8. Luego, seleccione el tipo de comando que permitirá obtener la información a mostrar en el TableAdapter.



Seleccione un tipo de comando permitirá especificar la sentencia o procedimiento almacenado.

- **Usar instrucciones SQL**

  Desde aquí se puede colocar la sentencia SQL directamente. Sería recomendable que primero se pruebe dicha sentencia en el mismo entorno de SQL Server y luego se debe copiar dicha sentencia.

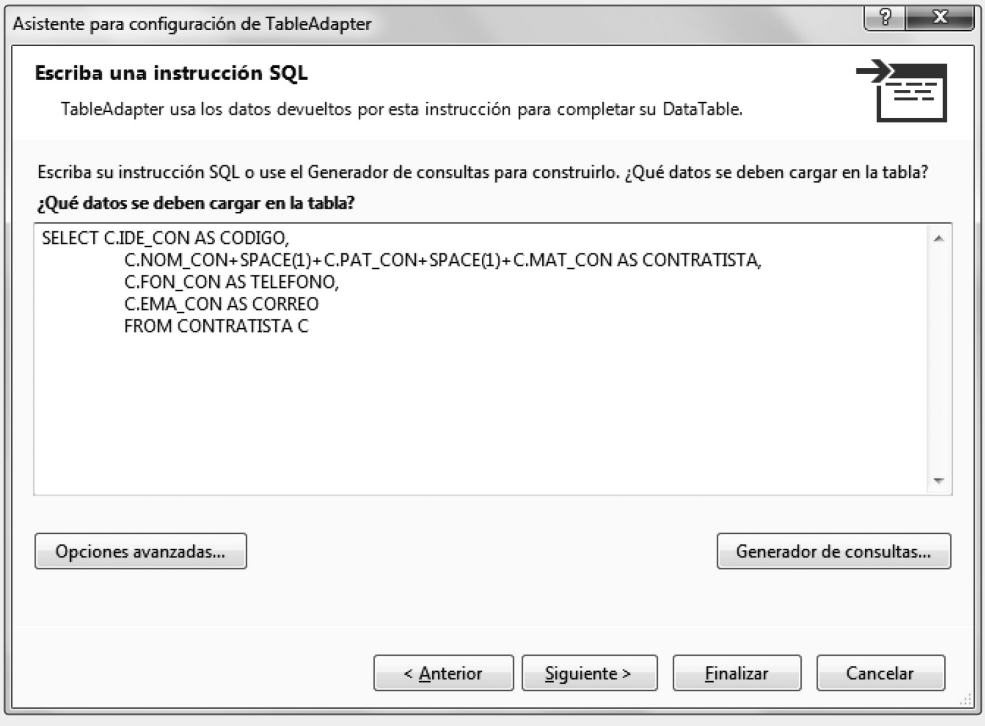
- **Crear nuevos procedimientos almacenados**

  Desde aquí se puede crear nuevos procedimientos almacenados a partir de una sentencia SQL.

- **Usar procedimientos almacenados existentes**

  Desde aquí se puede seleccionar los procedimientos almacenados, previamente implementados en el entorno SQL Server.

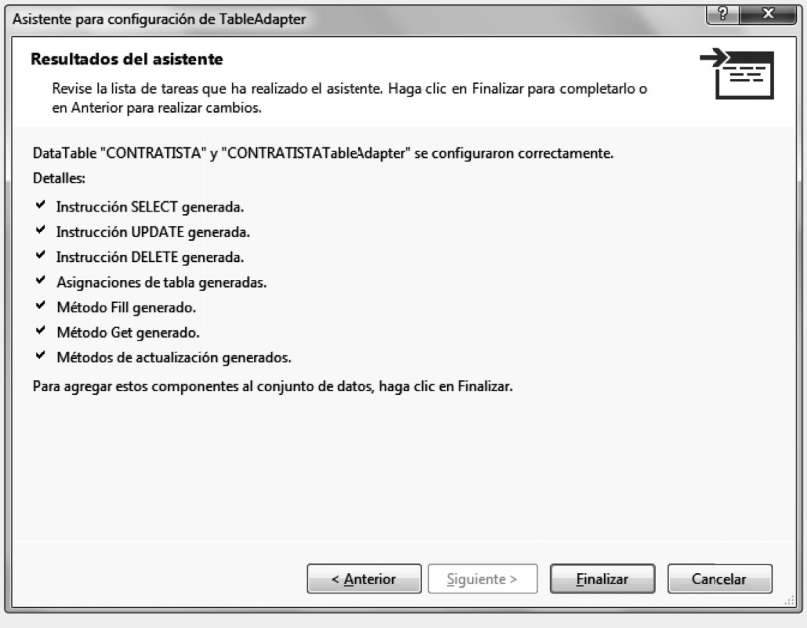
9. En este caso, seleccione **Usar instrucciones SQL** y coloque la sentencia que permitirá mostrar los datos del contratista, como se muestra en la siguiente imagen:



La variante más simple para la consulta de los contratistas puede ser solamente colocando SELECT * FROM CONTRATISTA.

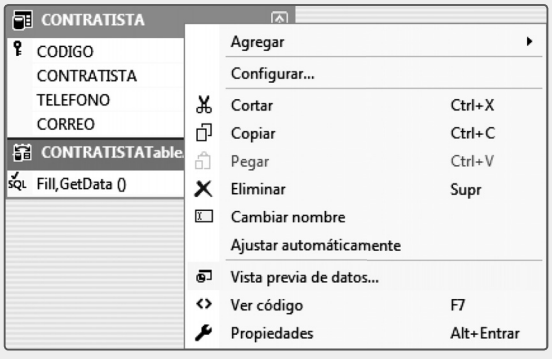
10. Luego, el asistente permitirá elegir los métodos que se generan por cada TableAdapter que genere. Para este caso, debe dejar como se muestra en la ventana y solo presionar el botón **Siguiente**.

Finalmente, se muestra la ventana de resultados, en la cual puede observar que todo lo seleccionado o colocado en el asistente fue correcto. Es así que todas las opciones deberán aparecer con un check.

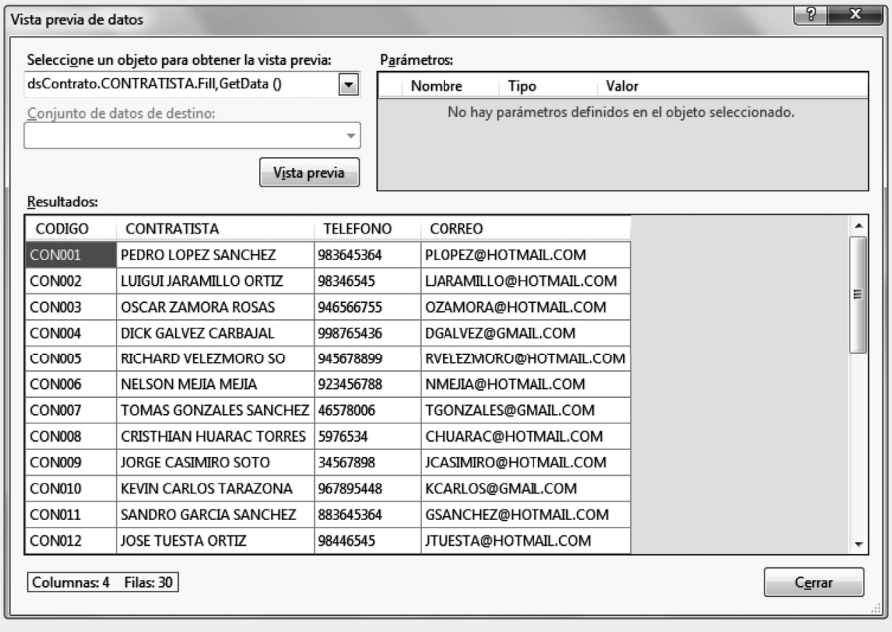


11. Cuando finaliza el asistente, su diseñador mostrará el siguiente objeto TableAdapter, indicando que todo es correcto y que puede usar dicho objeto en las aplicaciones:

12. Opcionalmente, se puede visualizar los datos que contiene dicho TableAdapter. Para esto, debe presionar clic derecho sobre el objeto TableAdapter > **Vista previa de datos…**, como se muestra en la siguiente imagen:



13. La ventana inicialmente no muestra los datos. Para esto, debe presionar el botón **Vista previa**, como se muestra en la siguiente ventana:



14. Modifique el nombre del formulario a frmListadoContratistas.
15. Coloque el siguiente código en el evento load 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 pjListadoContratistaTablaAdapter
{
    public partial class frmListadoContratistas : Form
    {
        public frmListadoContratistas()
        {
            InitializeComponent();
        }

       private void frmListadoContratistas_Load(object sender, EventArgs e)
        {
            llenaContratistas();
        }
        void llenaContratistas()
        {
            dsContratoTableAdapters.CONTRATISTATableAdapter da =
                        new dsContratoTableAdapters.CONTRATISTATableAdapter();
            dsContrato ds = new dsContrato();

            da.Fill(ds.CONTRATISTA);
            dgContratistas.DataSource = ds.Tables["CONTRATISTA"];
        }
    }
}
```

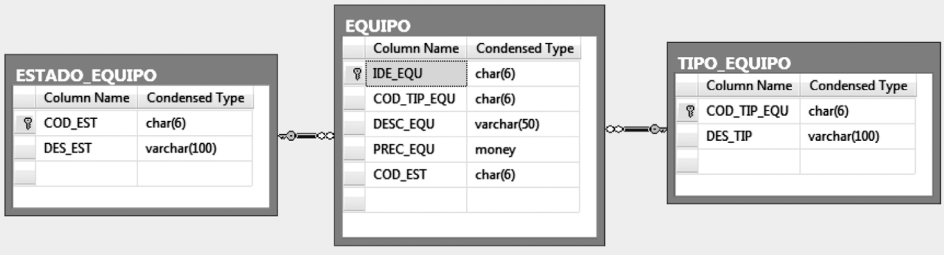
Inicie por crear un método que permita llenar los datos de los contratistas. Este método crea un objeto de la clase dsContratoTableAdapters.ContratistaTableAdapter. Con esto, se tendrá acceso al TableAdapter que contiene información de los contratistas. Luego, se crea un objeto del DataSet para poder tener acceso a todos los TableAdapters contenidos. Y como si se tratara de una DataAdapter, se debe rellenar la información mediante el método fill para finalmente enviar dicha información al control dgContratistas, siempre especificando el nombre de la tabla por tratarse de un DataSet

Caso desarrollado 2 : Usando tableadapter y procedimiento almacenado - Listado de equipos

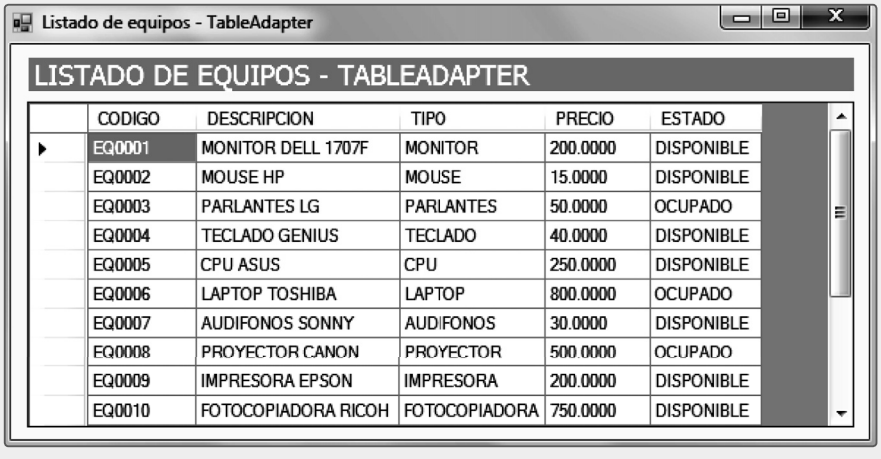
Implementar una aplicación Windows Forms que permita listar los registros de los equipos.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server un procedimiento almacenado que permita mostrar los datos de los equipos, como el código del equipo, la descripción, el precio del equipo, además de mostrar la descripción del tipo y el estado del mismo. Se debe considerar que la descripción del estado y tipo debe ser obtenida desde las tablas ESTADO_EQUIPO y TIPO_EQUIPO.
- Agregar un objeto DataSet al proyecto y agregar el procedimiento almacenado, que lista los datos del equipo.
- El listado se debe realizar sobre el control DataGridView.
- Tomar en cuenta la siguiente relación de tablas:



GUI propuesto:



CODIGO	DESCRIPCION	TIPO	PRECIO	ESTADO
EQ0001	MONITOR DELL 1707F	MONITOR	200.0000	DISPONIBLE
EQ0002	MOUSE HP	MOUSE	15.0000	DISPONIBLE
EQ0003	PARLANTES LG	PARLANTES	50.0000	OCUPADO
EQ0004	TECLADO GENIUS	TECLADO	40.0000	DISPONIBLE
EQ0005	CPU ASUS	CPU	250.0000	DISPONIBLE
EQ0006	LAPTOP TOSHIBA	LAPTOP	800.0000	OCUPADO
EQ0007	AUDIFONOS SONNY	AUDIFONOS	30.0000	DISPONIBLE
EQ0008	PROYECTOR CANON	PROYECTOR	500.0000	OCUPADO
EQ0009	IMPRESORA EPSON	IMPRESORA	200.0000	DISPONIBLE
EQ0010	FOTOCOPIADORA RICOH	FOTOCOPIADORA	750.0000	DISPONIBLE

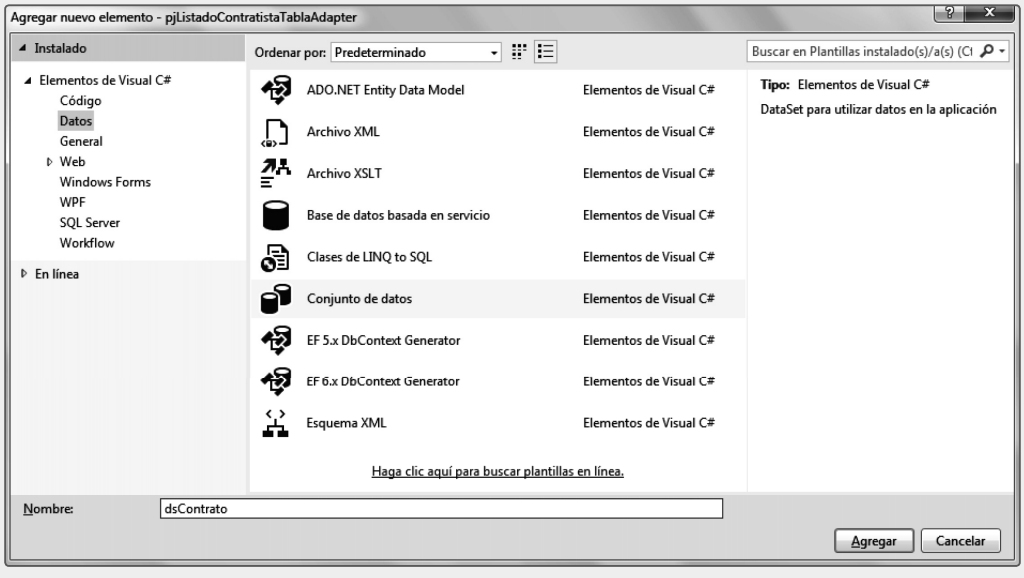
Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjListadoEquipoTableAdapter.

2. Implemente el siguiente procedimiento almacenado en SQL Server.

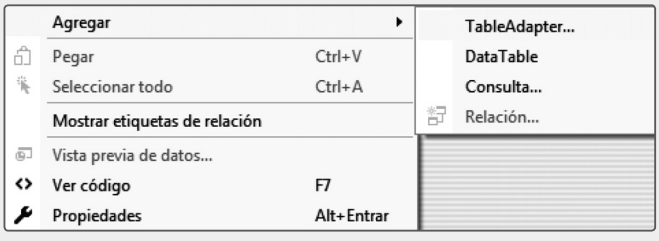
```
IF OBJECT_ID("SP_LISTAEQUIPOS")IS NOT NULL
       DROP PROC SP_LISTAEQUIPOS
GO
CREATE PROC SP_LISTAEQUIPOS
AS
       SELECT E.IDE_EQU AS CODIGO,
              E.DESC_EQU AS DESCRIPCION,
              T.DES_TIP AS TIPO,
              E.PREC_EQU AS PRECIO,
              ES.DES_EST AS ESTADO
              FROM EQUIPO E
              JOIN TIPO_EQUIPO T ON E.COD_TIP_EQU=T.COD_TIP_EQU
              JOIN ESTADO_EQUIPO ES ON E.COD_EST=ES.COD_EST
GO
```

3. Haga clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento**. Seleccione el elemento **Datos** > **Conjunto de datos** y asigne un nombre. En este caso, asigne el nombre dsContrato por tratarse de obtener los datos de la base de datos Contrato.

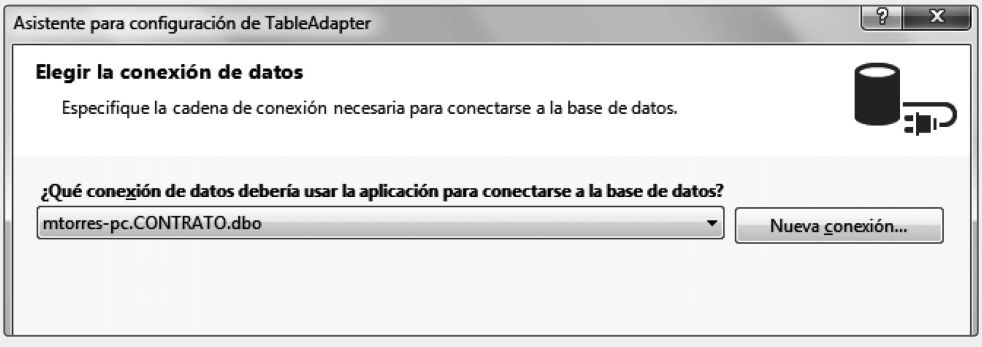


El diseñador de dataset se muestra vacío y es aquí donde se incorpora el procedimiento almacenado SP_LISTAEQUIPO.

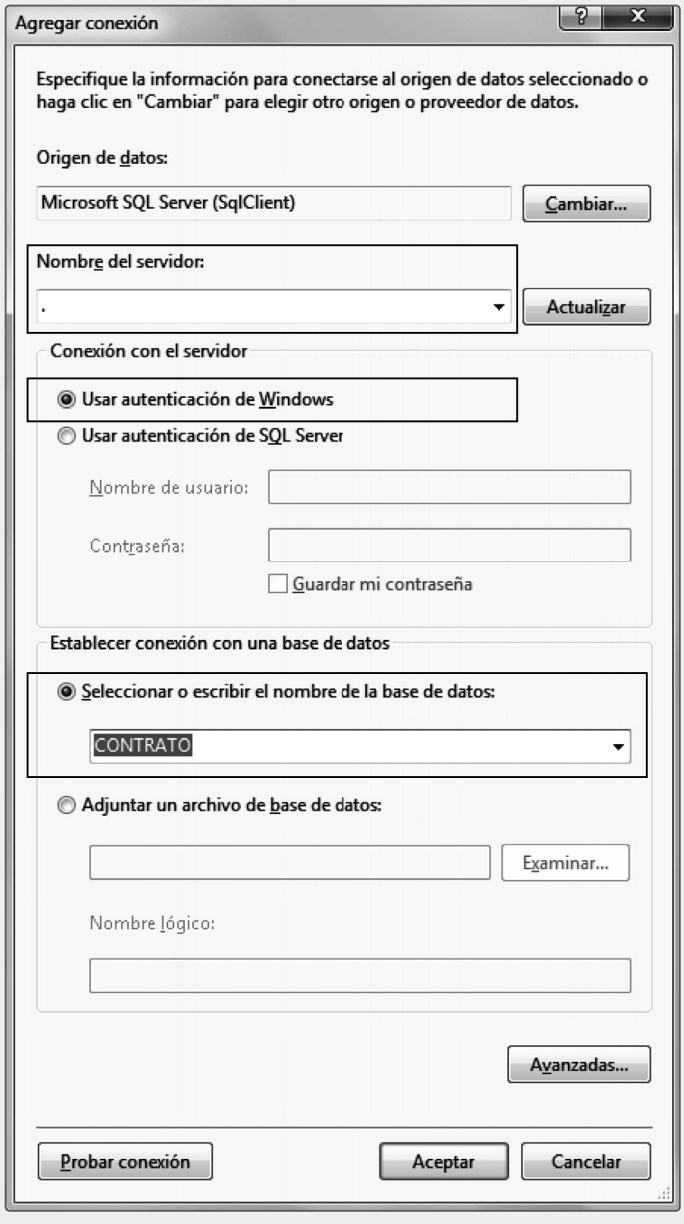
4. Agregue un TableAdapter al diseñador, presionando clic derecho sobre el fondo del entorno y seleccione **Agregar** > **TableAdapter…**



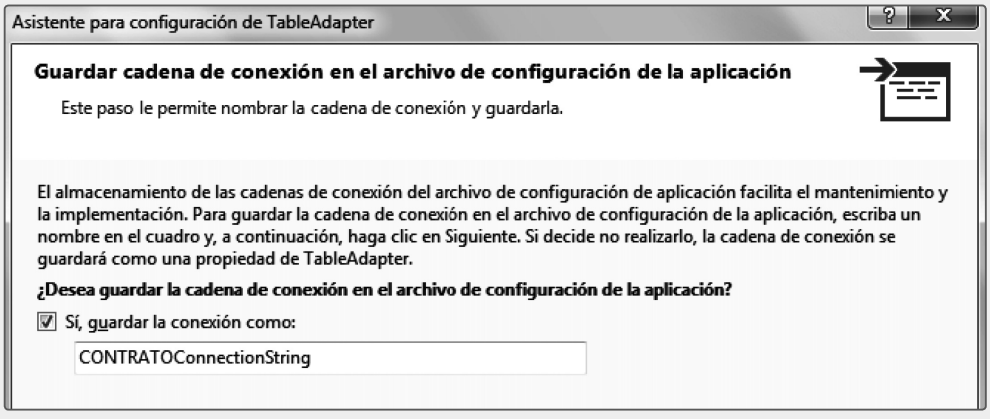
5. Elija la conexión de datos, es decir, en este paso se debe seleccionar el servidor y la base de datos, de donde obtendrá los datos a mostrar:



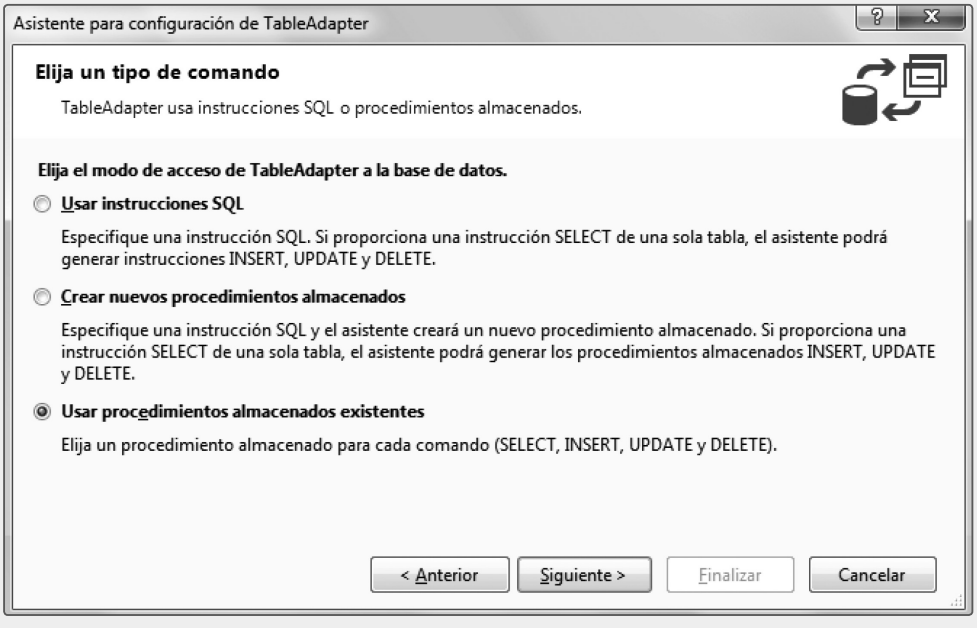
Se inicia por seleccionar el botón **Nueva conexión** para seleccionar el nombre del servidor, el tipo de autenticación y la base de datos desde la siguiente ventana:



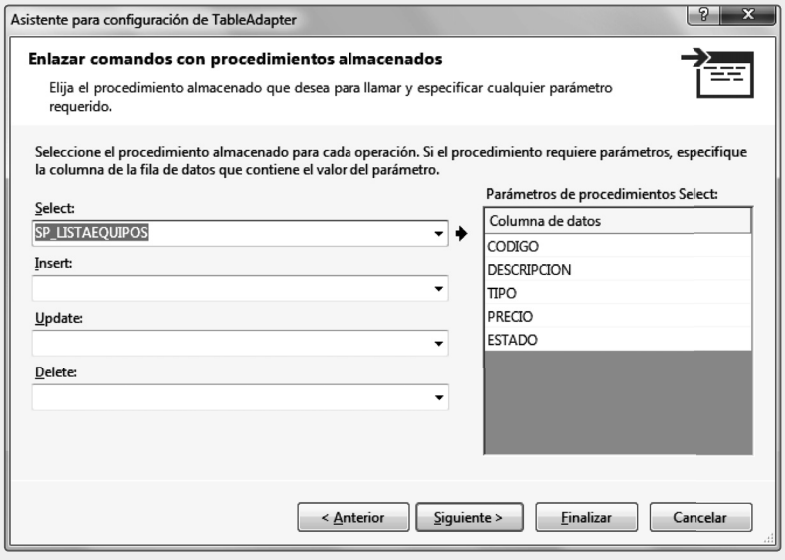
6. En la ventana siguiente se pide que se grabe la cadena de conexión en el archivo de configuración, es decir, la cadena quedará registrada en el archivo app.config. Aquí solo debe presionar el botón siguiente para continuar con el asistente:



7. No olvide que previamente se ha debido implementar el procedimiento almacenado SP_LISTAEQUIPO en SQL Server, ya que en este paso se debe seleccionar **Usar procedimientos almacenados existentes.**

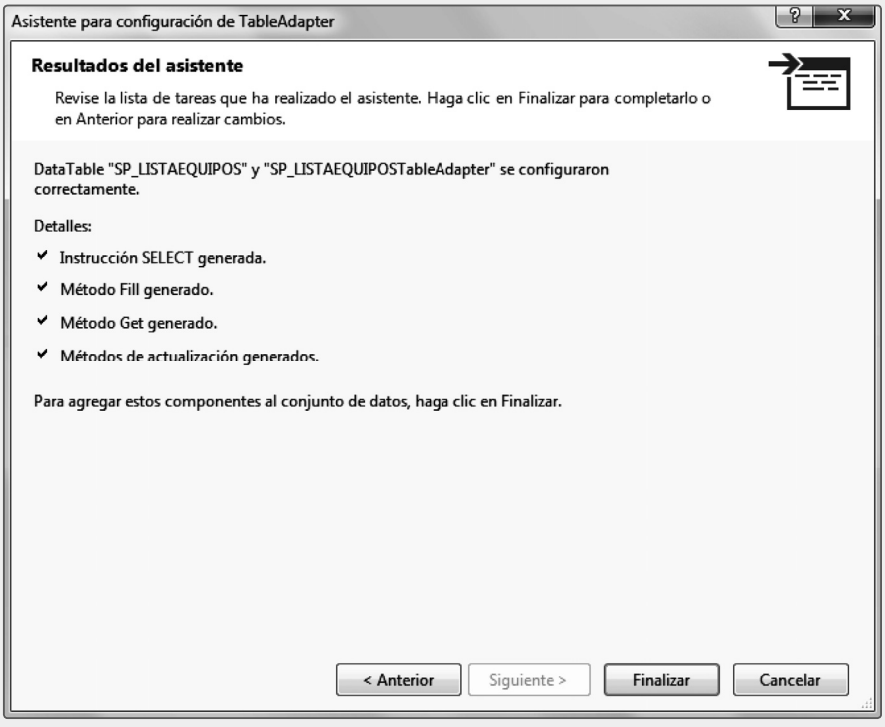


8. Si todo es correcto, entonces se debería mostrar una lista de procedimientos en la lista SELECT. Ahora, solo debe seleccionar el procedimiento almacenado SP_LISTAEQUIPO y como notará en el lado derecho se mostrarán las columnas que componen dicho procedimiento:

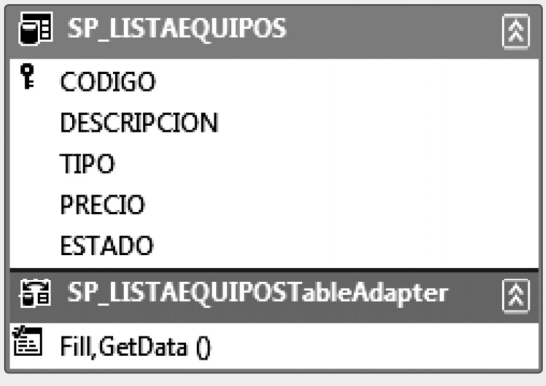


9. Luego, debe seleccionar los métodos del TableAdapter, para lo cual se dejará como se encuentra en la siguiente imagen:

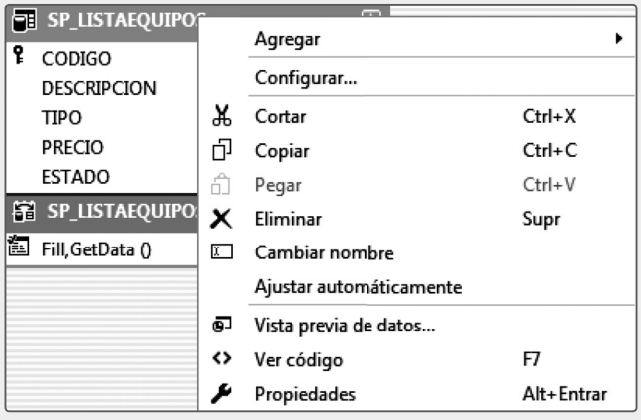
10. Finalmente, debe comprobar que todo lo seleccionado es correcto. Para esto, se mostrarán los check de todas las sentencias usadas en el asistente, como se muestra en la siguiente ventana. Ahora solo debe presionar **Finalizar** para terminar el proceso:



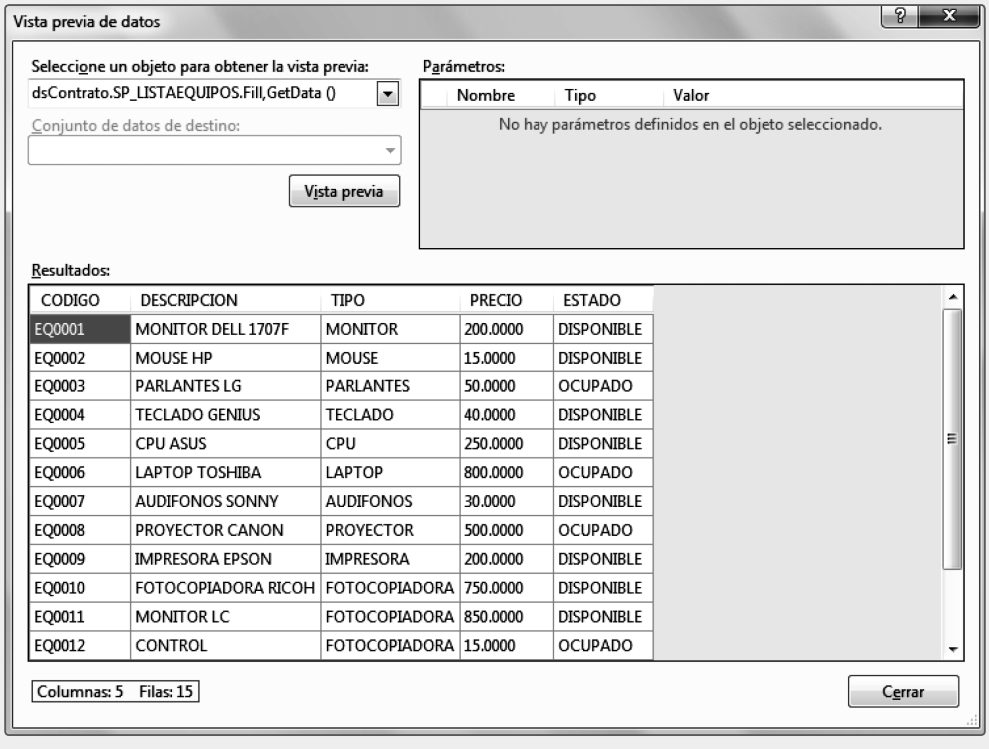
11. En el diseñador se mostrará el objeto TableAdapter que permite mostrar los datos de los equipos mediante el procedimiento almacenado SP_LISTAEQUIPOS. Debe tener en cuenta que el objeto TableAdapter recibe el mismo nombre que el procedimiento almacenado seleccionado en el asistente.



12. Opcionalmente, muestre los datos contenidos en el objeto SP_LISTAEQUIPOS. Para esto, presione clic derecho sobre dicho objeto y seleccione **Vista previa de datos...**



13. Ahora, solo presione el botón **Vista previa** y se mostrará los registros contenidos en el objeto SP_LISTAEQUIPO, como se muestra en la siguiente imagen:



CODIGO	DESCRIPCION	TIPO	PRECIO	ESTADO
EQ0001	MONITOR DELL 1707F	MONITOR	200.0000	DISPONIBLE
EQ0002	MOUSE HP	MOUSE	15.0000	DISPONIBLE
EQ0003	PARLANTES LG	PARLANTES	50.0000	OCUPADO
EQ0004	TECLADO GENIUS	TECLADO	40.0000	DISPONIBLE
EQ0005	CPU ASUS	CPU	250.0000	DISPONIBLE
EQ0006	LAPTOP TOSHIBA	LAPTOP	800.0000	OCUPADO
EQ0007	AUDIFONOS SONNY	AUDIFONOS	30.0000	DISPONIBLE
EQ0008	PROYECTOR CANON	PROYECTOR	500.0000	OCUPADO
EQ0009	IMPRESORA EPSON	IMPRESORA	200.0000	DISPONIBLE
EQ0010	FOTOCOPIADORA RICOH	FOTOCOPIADORA	750.0000	DISPONIBLE
EQ0011	MONITOR LC	FOTOCOPIADORA	850.0000	DISPONIBLE
EQ0012	CONTROL	FOTOCOPIADORA	15.0000	OCUPADO

14. Modifique el nombre del formulario a frmListadoEquipos.

15. Coloque el siguiente código en el evento load 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 pjListadoEquipoTableAdapter
{
    public partial class frmListadoEquipos : Form
    {
        public frmListadoEquipos()
        {
            InitializeComponent();
        }

        private void frmListadoEquipos_Load(object sender, EventArgs e)
        {
            dsContrato ds = new dsContrato();
            dsContratoTableAdapters.SP_LISTAEQUIPOSTableAdapter da=
                 new dsContratoTableAdapters.SP_LISTAEQUIPOSTableAdapter();
            da.Fill(ds.SP_LISTAEQUIPOS);

            dgEquipos.DataSource = ds.SP_LISTAEQUIPOS;
        }
    }
}
```

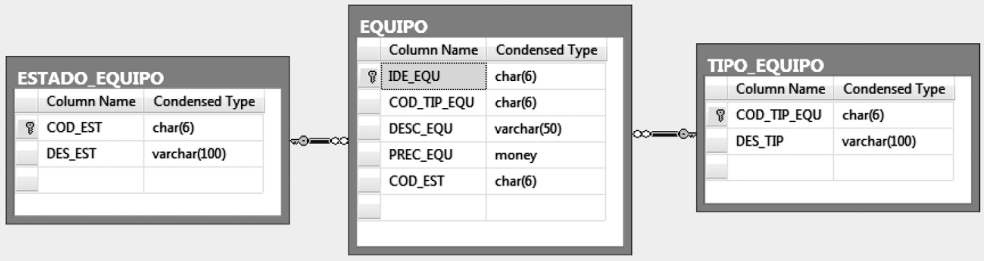
Caso desarrollado 3 : Usando TableAdapter y procedimiento almacenado - Listado de equipos por estado y tipo

Implementar una aplicación Windows Forms que permita listar los registros de los equipos según el tipo y el estado del mismo. Estos serán seleccionados por medio de controles de cuadro combinado.

Se tiene en cuenta lo siguiente:

- Implementar en SQL Server los siguientes procedimientos almacenados: listado de equipos (SP_EQUIPOS), listado de tipo de equipos (SP_TIPOEQUIPO), listado de estados de equipo (SP_ESTADOEQUIPO) y el listado de los equipos por estado y por tipo (SP_EQUIPOSXESTADOXTIPO).

- Agregar un objeto DataSet al proyecto y agregar todos los procedimientos almacenados.
- El listado de los equipos se debe mostrar en el control DataGridView, mientras que los estados y los tipos en un cuadro combinado.
- Tomar en cuenta la siguiente relación de tablas:



GUI propuesto:

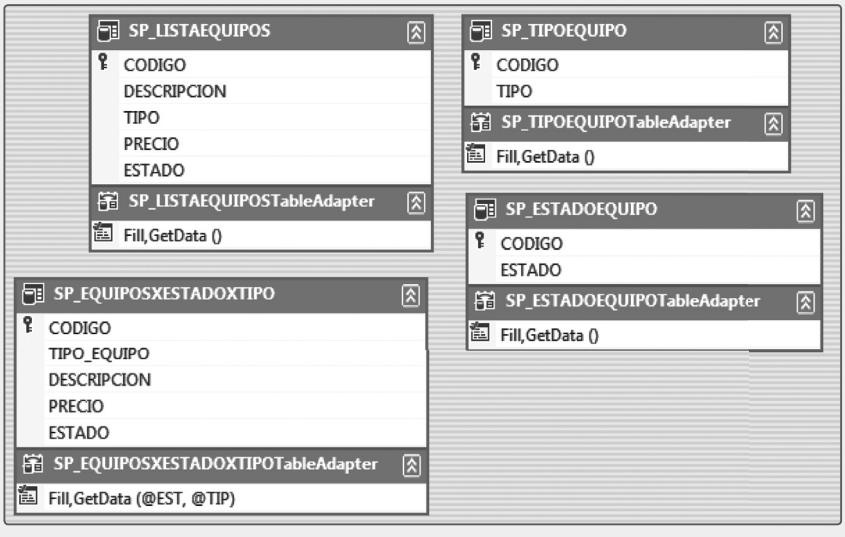
Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjBuscaEquipoTableAdapter.
2. Implemente los siguientes procedimientos almacenados en SQL Server.

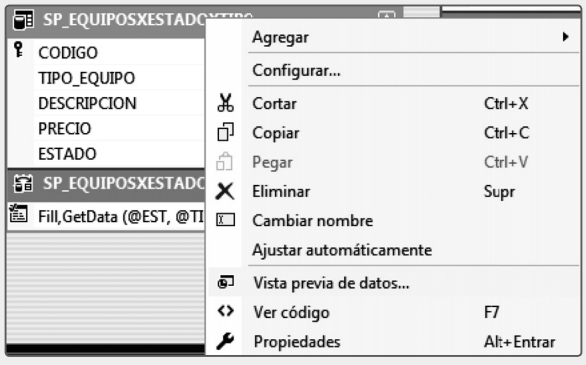
```
CREATE PROC SP_LISTAEQUIPOS
AS
       SELECT E.IDE_EQU AS CODIGO,
              E.DESC_EQU AS DESCRIPCION,
              T.DES_TIP AS TIPO,
              E.PREC_EQU AS PRECIO,
              ES.DES_EST AS ESTADO
              FROM EQUIPO E
              JOIN TIPO_EQUIPO T ON E.COD_TIP_EQU=T.COD_TIP_EQU
              JOIN ESTADO_EQUIPO ES ON E.COD_EST=ES.COD_EST
GO
```

```
CREATE PROC SP_TIPOEQUIPO
AS
        SELECT T.COD_TIP_EQU AS CODIGO,
               T.DES_TIP AS TIPO
               FROM TIPO_EQUIPO T
GO

CREATE PROC SP_ESTADOEQUIPO
AS
        SELECT E.COD_EST AS CODIGO,
               E.DES_EST AS ESTADO
               FROM ESTADO_EQUIPO E
GO

CREATE PROC SP_EQUIPOSXESTADOXTIPO(@EST CHAR(6),@TIP CHAR(6))
AS
        SELECT E.IDE_EQU AS CODIGO,
               TE.DES_TIP AS TIPO_EQUIPO,
               E.DESC_EQU AS DESCRIPCION,
               E.PREC_EQU AS PRECIO,
               EE.DES_EST AS ESTADO
               FROM EQUIPO E
               JOIN TIPO_EQUIPO TE ON TE.COD_TIP_EQU=E.COD_TIP_EQU
               JOIN ESTADO_EQUIPO EE ON EE.COD_EST=E.COD_EST
               WHERE E.COD_EST=@EST AND E.COD_TIP_EQU=@TIP
GO
```

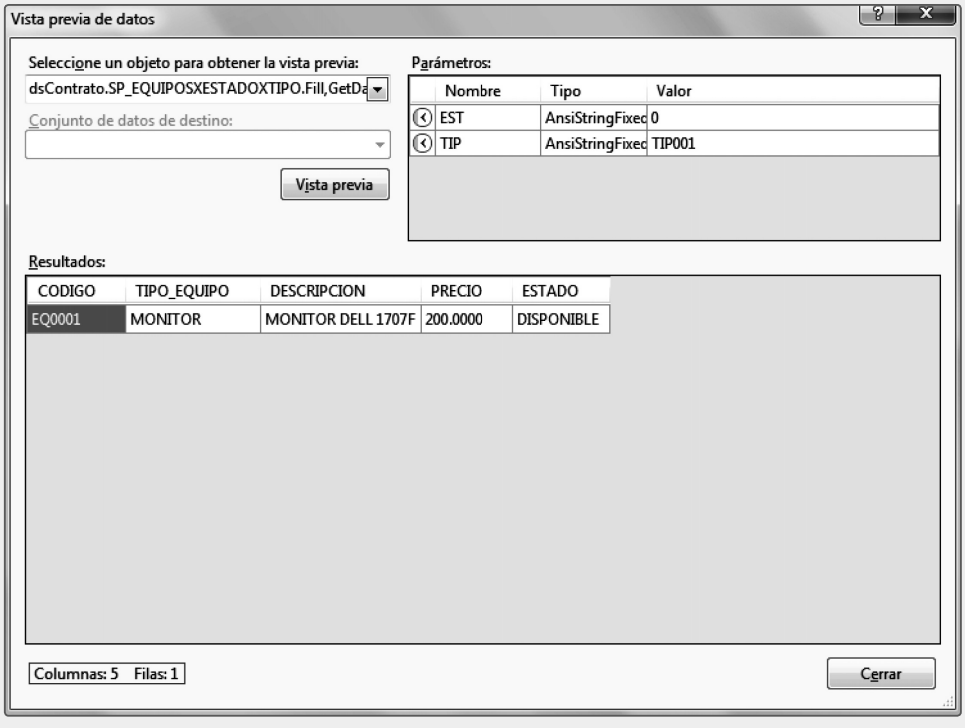
3. Agregue un objeto del tipo DataSet al proyecto llamado DSContrato, en el cual agregará todos los procedimientos almacenados al diseñador como se muestra en la siguiente imagen:



4. Opcionalmente, pruebe los datos que muestra el objeto SP_EQUIPOSXESTADOXTIPO. Para lo cual, presione clic derecho sobre el objeto y seleccione **Vista previa de datos...**



Opcionalmente, en la siguiente ventana, antes de presionar el botón **Vista previa**, se debe agregar dos parámetros, los cuales se encuentran en la parte superior derecha, dichos valores deben ser códigos como, por ejemplo, en el estado del equipo solo se tiene dos códigos como cero y uno, mientras que en el tipo se tiene TIP001, TIP002, TIP003, ... TIP010, etc.



5. Modifique el nombre del formulario a frmBuscaEquipo.
6. Coloque el siguiente código en el evento load 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 pjBuscaEquipoTableAdapter
{
    public partial class frmBuscaEquipo : Form
    {
        //GLOBAL
        dsContrato ds = new dsContrato();

        public frmBuscaEquipo()
        {
            InitializeComponent();
        }

        private void frmBuscaEquipo_Load(object sender, EventArgs e)
        {
            llenaEstado();
            llenaTipo();
            llenaEquipo();
        }

        void llenaEstado()
        {
            dsContratoTableAdapters.SP_ESTADOEQUIPOTableAdapter da =
                   new dsContratoTableAdapters.SP_ESTADOEQUIPOTableAdapter();
            da.Fill(ds.SP_ESTADOEQUIPO);
            cboEstado.DataSource = ds.SP_ESTADOEQUIPO;
            cboEstado.DisplayMember = "ESTADO";
            cboEstado.ValueMember = "CODIGO";

        }
        void llenaTipo()
        {
            dsContratoTableAdapters.SP_TIPOEQUIPOTableAdapter da =
                   new dsContratoTableAdapters.SP_TIPOEQUIPOTableAdapter();
            da.Fill(ds.SP_TIPOEQUIPO);
            cboTipo.DataSource = ds.SP_TIPOEQUIPO;
```

```
            cboTipo.DisplayMember = "TIPO";
            cboTipo.ValueMember = "CODIGO";

        }

        void llenaEquipo()
        {
            dsContratoTableAdapters.SP_LISTAEQUIPOSTableAdapter da =
                 new dsContratoTableAdapters.SP_LISTAEQUIPOSTableAdapter();
            da.Fill(ds.SP_LISTAEQUIPOS);
            dgEquipo.DataSource = ds.SP_LISTAEQUIPOS;
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            string estado = cboEstado.SelectedValue.ToString();
            string tipo = cboTipo.SelectedValue.ToString();

          dsContratoTableAdapters.SP_EQUIPOSXESTADOXTIPOTableAdapter da =
             new dsContratoTableAdapters.SP_EQUIPOSXESTADOXTIPOTableAdapter();
            da.Fill(ds.SP_EQUIPOSXESTADOXTIPO, estado, tipo);

            dgEquipo.DataSource = ds.SP_EQUIPOSXESTADOXTIPO;
        }
    }
}
```

EDITORIAL MACRO®

5.2 LinQ

Language-Integrated Query (LINQ) es considerado como un conjunto de tecnologías modernas, que consiste en integrar funciones de consulta en el lenguaje C#, solo basta tener un conjunto de elementos guardados a partir de un arreglo de datos o una tabla de base de datos para poder usar una sentencia LinQ.

Para este caso, se usará LinQ integrada a la tecnología ADO.NET, en dicha integración existen tres tecnologías:

- **LinQ to DataSet**

  Proporciona una capacidad de consultas mucho más dinámica sobre un objeto de tipo DataSet. Asimismo, este tipo de tecnología es considerado para aplicaciones de tipo desconectado a la base de datos.

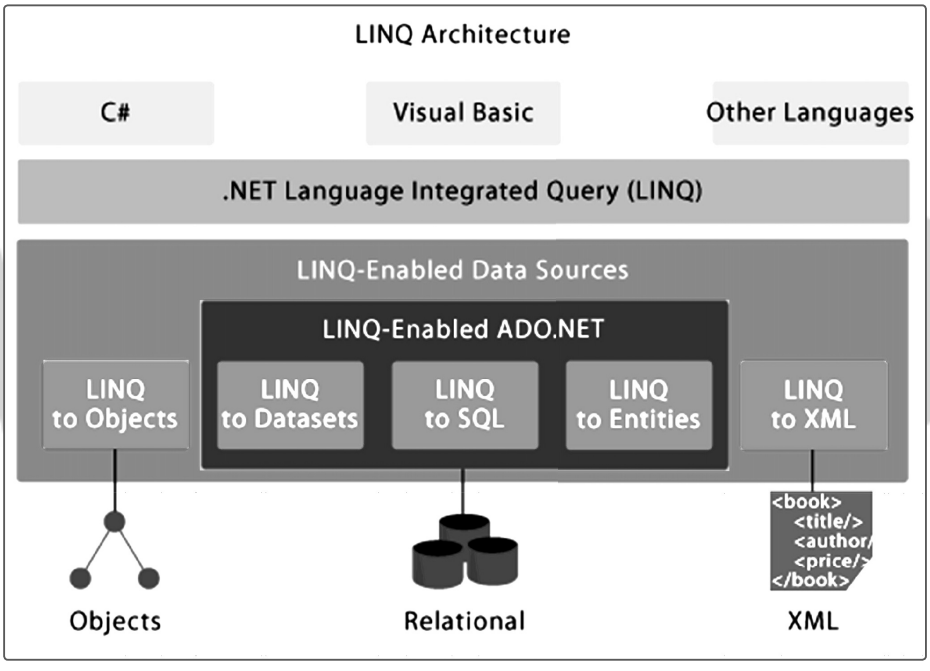
- **LinQ to SQL**

  Permite realizar una consulta directamente sobre un objeto de datos conectado a una base de datos. Asimismo, este tipo de tecnología es considerado como un modelo relacional de objetos, que permite ejecutar consultas convirtiendo el modelo de objeto en una sentencia entendible por SQL y se las envía para que los procese. Luego, SQL envía la respuesta, para lo cual LinQ to SQL los convierte nuevamente a objetos que pueden ser manipulados por el desarrollador de aplicaciones.

- **LinQ to Entities**

  Permite consultar información sobre objetos contenidos en un modelo de datos llamado Entity Data Model.

Para un programador que escribe consultas, la parte integrada en el lenguaje más visible de LINQ es la expresión de consulta. Dichas expresiones se escriben en la sintaxis de consulta declarativa. Al usar esta sintaxis, se pueden realizar incluso operaciones complejas de filtrado, ordenación y agrupamiento en orígenes de datos con código mínimo. Los mismos patrones de expresión de consulta básicos se usan para consultar y transformar datos de bases de datos SQL, conjuntos de datos de ADO.NET, secuencias y documentos XML, y colecciones de .NET. Observe a continuación la arquitectura que presenta LinQ:



Fuente: <http://www.codeproject.com/KB/linq/UnderstandingLINQ/UnderstandingLINQ.gif>

5.3 Implementación de una consulta con LINQ

Una consulta es una expresión que recupera datos de un origen de datos. Las consultas se expresan en un lenguaje de consultas dedicado. A lo largo del tiempo, se han ido desarrollando lenguajes diferentes para los distintos tipos de orígenes de datos, como SQL para las bases de datos relacionales y XQuery para XML. De esta manera, el desarrollador de aplicaciones debe aprender un nuevo lenguaje de consultas para cada tipo de origen de datos o formato de datos admitido.

LINQ simplifica esta situación al proporcionar un modelo coherente para trabajar con los datos de varios formatos y orígenes de datos. En una consulta LINQ siempre se trabaja con objetos.

Se utilizan los mismos modelos de codificación básicos para consultar y transformar los datos de documentos XML, las bases de datos SQL, los conjuntos de datos y las entidades de ADO.NET, las colecciones de .NET Framework y cualquier otro formato u origen de datos para el que haya un proveedor LINQ disponible.

5.4 Operaciones básicas de una consulta LinQ to SQL

Observe algunas instrucciones para la implementación de una consulta condicionada en LinQ to SQL.

Cláusula	Descripción

From	Especifica el origen de datos que se desea consultar, el cual toma el siguiente formato:
	From Alias **in** Colección
	Por ejemplo, si se quiere realizar una consulta a la tabla Clientes, el código LinQ to SQL sería de la siguiente forma:
	`var misClientes = from c in CLIENTE`
	`select c;`
	Donde, misClientes es una variable que contiene el resultado de la consulta. La variable *c* representa el alias de la colección y cliente es el nombre de la tabla de donde provienen los datos, previamente deben encontrarse en el entorno DataContext. Finalmente, select c representa la selección de todos los campos de la tabla Cliente.

Where	Permita filtrar la información que será enviada a la variable de tipo colección, el cual toma el siguiente formato:
	From Alias **in** Colección
	Where condición;
	Por ejemplo, si se quiere realizar una consulta del cliente con código CL0001, el código LinQ to SQL sería de la siguiente forma:
	`var misClientes = from c in CLIENTE`
	`where c.IDE_CLI = "CL0001"`
	`select c;`

Order by

Permite ordenar los registros en forma ascendente o descendente de una determinada columna del objeto LinQ. Observe cómo ordenar de forma descendente por el código del cliente:

```
var misClientes =  from c in CLIENTE
                   orderby c.IDE_CLI descending
                   select c;
```

Select

Permite seleccionar los registros de una consulta LinQ to SQL. Observe cómo mostrar ciertas columnas de la tabla Cliente.

```
var misClientes = from c in CLIENTE
                  select new
                  {
                     c.IDE_CLI,
                     c.NOM_CLI,
                     c.PAT_CLI,
                     c.MAT_CLI
                  };
```

O también se podría definir las cabeceras de las columnas mostradas de la siguiente manera:

```
var misClientes = from c in CLIENTE
                  select new
                  {
                     CODIGO = c.IDE_CLI,
                     NOMBRES = c.NOM_CLI,
                     PATERNO = c.PAT_CLI,
                     MATERNO = c.MAT_CLI
                  };
```

O también se podría concatenar las columnas, por ejemplo, unir el nombre y los apellidos de la siguiente manera:

```
var misClientes = from c in CLIENTE
                  select new
                  {
                     CODIGO = c.IDE_CLI,
                     CLIENTE = c.NOM_CLI+c.PAT_CLI+c.MAT_CLI,
                     CORREO  = c.COR_CLI
                  };
```

Join

Permite unir dos o más objetos pertenecientes al entorno DataSet. Observe cómo mostrar los datos de los clientes adicionando la descripción de su distrito, como se muestra en el siguiente código:

```
var misClientes = from c in CLIENTE
      join d in DISTRITO on c.COD_DIST equals d.COD_DIST
      select new
      {
        CODIGO = c.COD_CLI,
        CLIENTE = c.NOM_CLI+c.PAT_CLI+c.MAT_CLI,
        DISTRITO = d.DES_DIS,
        CORREO_ELECTRONICO = c.COR_CLI
      };
```

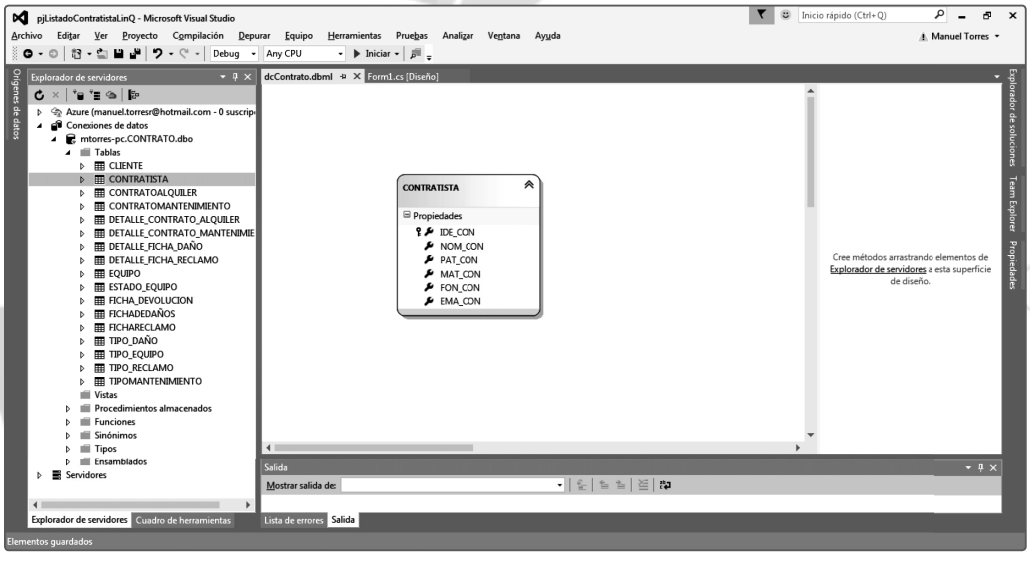
5.5 Clase DataContext

Representa el principal punto de entrada para el marco de trabajo de LINQ to SQL. DataContext es el origen de todas las entidades asignadas en una conexión de base de datos. Realiza un seguimiento de los cambios realizados en todas las entidades recuperadas y mantiene una «memoria caché de identidad», que garantiza que las entidades que se recuperan más de una vez se representen utilizando la misma instancia de objeto.

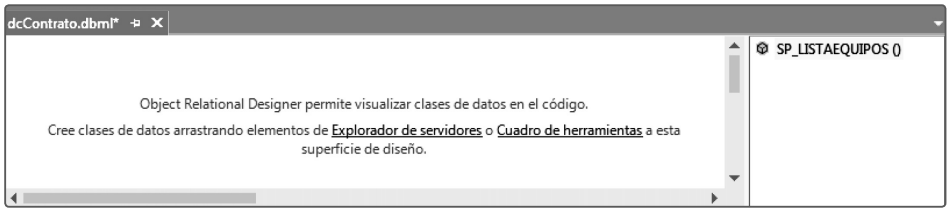
En general, una instancia de DataContext está diseñada para que dure una unidad de trabajo, sea cual sea la definición de ese término en la aplicación. DataContext es un objeto ligero cuya creación no es costosa. Una aplicación LINQ to SQL típica crea instancias de DataContext en el ámbito de métodos o como miembro de clases de duración corta que representan un conjunto lógico de operaciones de base de datos relacionados.

Para enviar una o más tablas al DataContext, se necesita la ventana Explorador de servidores, la cual se puede activar desde la opción ver > **Explorador de servidores**.

Luego, debe arrastrar la tabla desde la carpeta tablas del Explorador de servidores. Al final, el entorno se debe mostrar como la siguiente imagen:



En caso de usar procedimientos almacenados, existe un área destinada a dichos objetos como se muestra en la siguiente imagen:



Caso desarrollado 4 : LinQ to SQL - Listado de contratista

Implementar una aplicación que permita listar los datos de los contratistas en un control DataGridView, para lo cual se usará la clase LinQ to SQL.

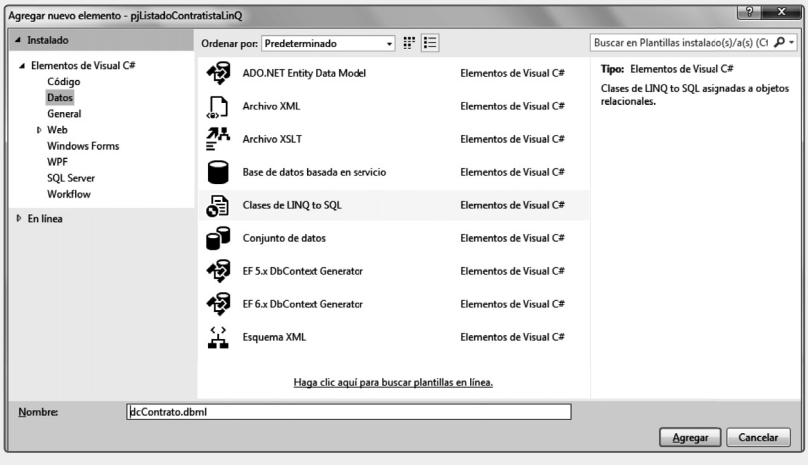
Se tiene en cuenta lo siguiente:

- Agregar al proyecto un archivo de tipo Clases de LinQ to SQL.
- Usar el Explorador de servidores para seleccionar la tabla Contratista.
- Usar LinQ para listar todos los registros de la tabla Contratista

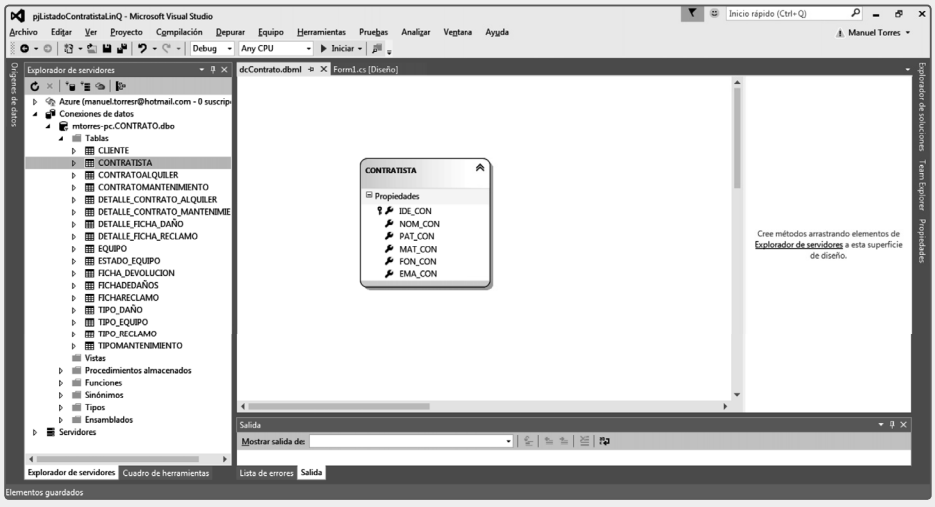
GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjListadoContratistaLinQ.
2. Agregue un archivo al proyecto presionando clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento** > **Datos** > **Clases de LinQ to SQL** y asigne el nombre dcContrato, como se muestra en la siguiente imagen:



3. Clases LinQ to SQL presenta un entorno de trabajo en la cual debe agregar la tabla, de la cual se quiere obtener sus datos. Para este caso, arrastre la tabla CONTRATISTA desde la ventana Explorador de servidores, como se muestra en la siguiente imagen:



4. Luego, coloque el siguiente código en el formulario frmListadoContratistas:

```
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 pjListadoContratistaLinQ
{
    public partial class frmListadoContratistas : Form
    {
        public frmListadoContratistas()
        {
            InitializeComponent();
        }

       private void frmListadoContratistas_Load(object sender, EventArgs e)
        {
            llenaContratista();
        }
        void llenaContratista()
        {
            dcContratoDataContext da = new dcContratoDataContext();
            var contratistas = from c in da.CONTRATISTA select c;
            dgContratistas.DataSource = contratistas;
        }
    }
}
```

Implemente el método llenaContratista para rellenar de información al control DataGridView. Dentro, se debe crear un objeto de la clase dcContratoDataContext que pertenece al entorno de desarrollo de las Clases de LinQ to SQL. Esto tiene como finalidad acceder a los elementos contenidos en el entorno.

Luego, se debe crear una variable LinQ para acceder a la información de los contratistas, que luego será asignada al control dgContratistas.

Se debe tener en cuenta que al colocar select c en la sentencia LinQ está aceptando que se muestren todas las columnas de la tabla Contratista. Una modificación al código podría ser de la siguiente manera:

```
void llenaContratista()
{
    dcContratoDataContext da = new dcContratoDataContext();
    var contratistas = from c in da.CONTRATISTA
                   select new
                   {
                      CODIGO = c.IDE_CON,
                      NOMBRES = c.NOM_CON,
                      PATERNO = c.PAT_CON,
                      MATERNO = c.MAT_CON,
                      TELEFONO = c.FON_CON,
                      CORREO = c.EMA_CON
                   };
    dgContratistas.DataSource = contratistas;
}
```

O podría unir su nombre completo en una sola línea, como se muestra en el siguiente código:

```
void llenaContratista()
{
    dcContratoDataContext da = new dcContratoDataContext();
    var contratistas = from c in da.CONTRATISTA
                   select new
                   {
                      CODIGO = c.IDE_CON,
                    CONTRATISTA = c.NOM_CON+" "+c.PAT_CON + " "+c.MAT_CON,
                      TELEFONO = c.FON_CON,
                      CORREO = c.EMA_CON
                   };
    dgContratistas.DataSource = contratistas;
}
```

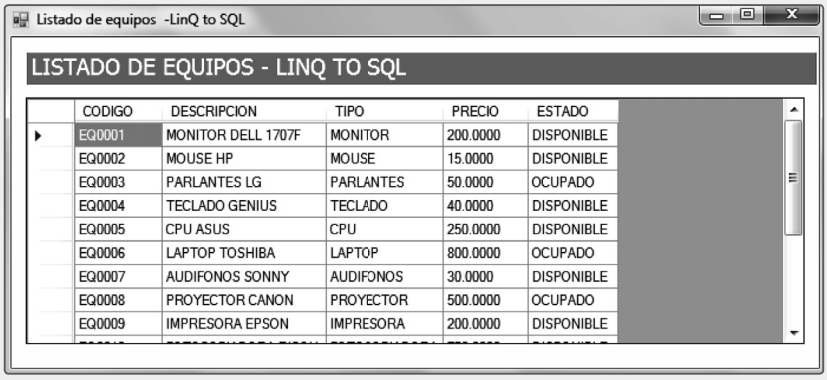
Caso desarrollado 5 : LinQ to SQL con procedimientos almacenados - Listado de equipos

Implementar una aplicación que permita mostrar los registros de los equipos, para lo cual debe usar la clase LinQ to SQL.

Se tiene en cuenta lo siguiente:

- Implementar el procedimiento almacenado SP_LISTAEQUIPOS, que permita mostrar los datos del equipo adicionando la descripción del tipo y el estado.
- Agregar al proyecto un archivo de tipo Clases de LinQ to SQL.
- Agregar el procedimiento almacenado desde el Explorador de servidores.

GUI Propuesta:



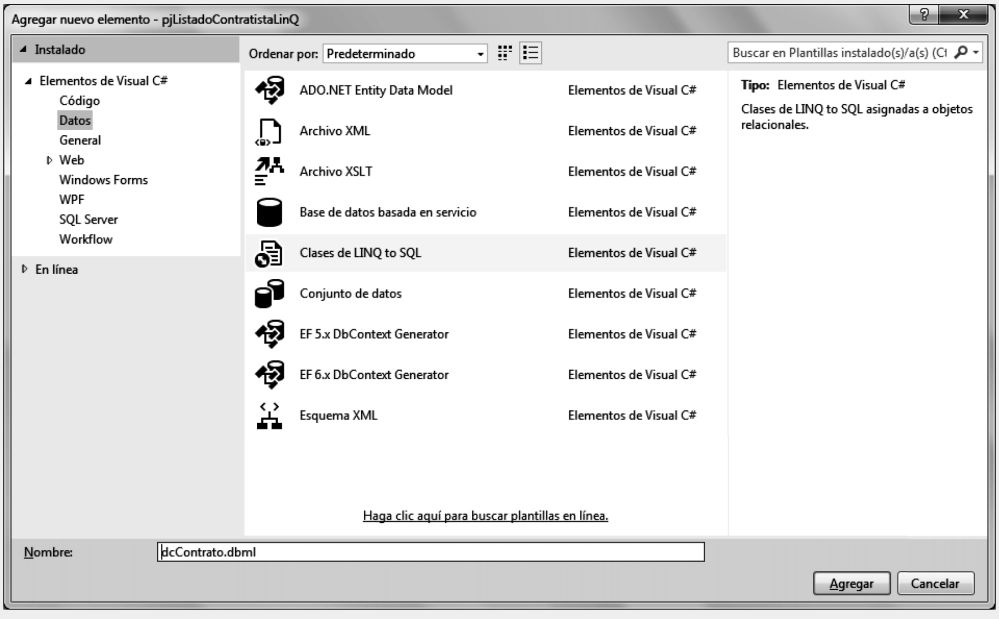
CODIGO	DESCRIPCION	TIPO	PRECIO	ESTADO
EQ0001	MONITOR DELL 1707F	MONITOR	200.0000	DISPONIBLE
EQ0002	MOUSE HP	MOUSE	15.0000	DISPONIBLE
EQ0003	PARLANTES LG	PARLANTES	50.0000	OCUPADO
EQ0004	TECLADO GENIUS	TECLADO	40.0000	DISPONIBLE
EQ0005	CPU ASUS	CPU	250.0000	DISPONIBLE
EQ0006	LAPTOP TOSHIBA	LAPTOP	800.0000	OCUPADO
EQ0007	AUDIFONOS SONNY	AUDIFONOS	30.0000	DISPONIBLE
EQ0008	PROYECTOR CANON	PROYECTOR	500.0000	OCUPADO
EQ0009	IMPRESORA EPSON	IMPRESORA	200.0000	DISPONIBLE

Solución:

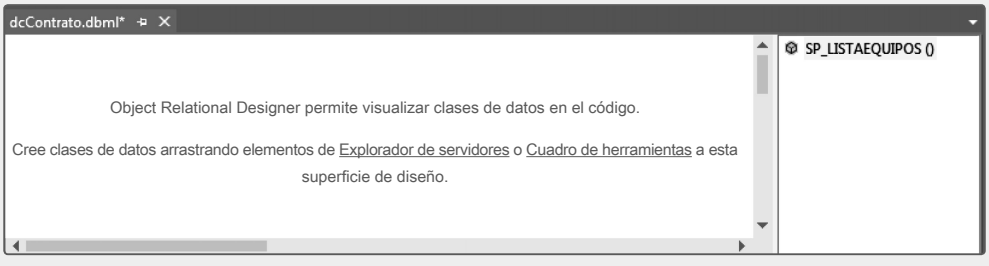
1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjListaEquiposLinQ.
2. Ejecute el siguiente procedimiento almacenado en SQL Server.

```
IF OBJECT_ID("SP_LISTAEQUIPOS")IS NOT NULL
       DROP PROC SP_LISTAEQUIPO
GO
CREATE PROC SP_LISTAEQUIPOS
AS
       SELECT E.IDE_EQU AS CODIGO,
              TE.DES_TIP AS TIPO_EQUIPO,
              E.DESC_EQU AS DESCRIPCION,
              E.PREC_EQU AS PRECIO,
              EE.DES_EST AS ESTADO
              FROM EQUIPO E
              JOIN TIPO_EQUIPO TE ON TE.COD_TIP_EQU=E.COD_TIP_EQU
              JOIN ESTADO_EQUIPO EE ON EE.COD_EST=E.COD_EST
GO
```

3. Agregue un archivo al proyecto presionando clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento** > **Datos** > **Clases de LinQ to SQL** y asigne el nombre dcContrato, como se muestra en la siguiente imagen:



4. Arrastre el procedimiento almacenado SP_LISTAEQUIPOS desde el Explorador de servidores al entorno que presenta Clases LinQ to SQL, como se muestra en la siguiente imagen:



5. Luego, coloque el siguiente código en el formulario frmListadoEquipos:

```
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 pjListaEquiposLinQ
{
    public partial class frmListadoEquipos : Form
    {
        public frmListadoEquipos()
        {
            InitializeComponent();
        }

        private void frmListadoEquipos_Load(object sender, EventArgs e)
        {
            llenaEquipos();
        }
        void llenaEquipos()
        {
            dcContratoDataContext db = new dcContratoDataContext();
            dgEquipos.DataSource = db.SP_LISTAEQUIPOS();
        }
    }
}
```

Implemente el método llenaEquipos, en el cual se debe crear un objeto de la clase dcContratoDataContext perteneciente al entorno Clases de LinQ to SQL. Por tratarse de un procedimiento almacenado implementado en SQL Server, se debe asignar directamente db.SP_LISTAEQUIPOS() al control dgEquipos sin necesidad de especificar las columnas o realizar algún cambio, ya que el procedimiento almacenado tiene su propia implementación.

Caso desarrollado 6 : LinQ to SQL - Listado de fichas de reclamo por cliente

Implementar una aplicación que permita mostrar las fichas de reclamo por cliente a partir de la selección de un cliente desde el control cuadro combinado. Se deberá mostrar todas las fichas de reclamo asociadas a dicho cliente, para lo cual debe usar la clase LinQ to SQL.

Se tiene en cuenta lo siguiente:

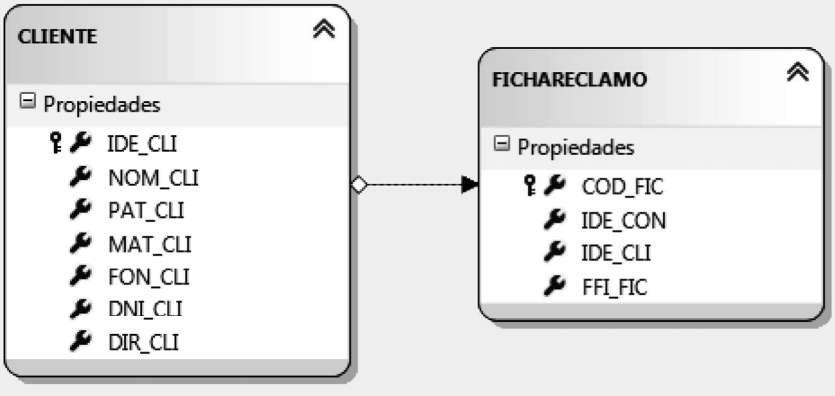
- Agregar al proyecto un archivo de tipo Clases de LinQ to SQL.
- Agregar las tablas Cliente y Ficha de Reclamo al entorno de Clases de LinQ to SQL.
- Asignar el nombre completo de los clientes (nombres y apellidos) en un cuadro combinado.

- Mostrar en una lista los datos de las fichas de reclamo como el código de la ficha, el código de contratista, el código de cliente y la fecha de registro de la ficha de reclamo.

GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjFichasReclamoxCliente.
2. Agregue un archivo al proyecto presionando clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento** > **Datos** > **Clases de LinQ to SQL** y asigne el nombre dcContrato.
3. Agregue las tablas Cliente y Ficha de reclamo al entorno Clases LinQ to SQL, como se muestra en la siguiente imagen:



4. Luego, coloque el siguiente código en el formulario frmFichasxCliente.

```
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 pjFichasReclamoxCliente
{
    public partial class frmFichasxCliente : Form
    {
        //Definición GLOBAL
        dcContratoDataContext db = new dcContratoDataContext();

        public frmFichasxCliente()
        {
            InitializeComponent();
        }

        private void frmFichasxCliente_Load(object sender, EventArgs e)
        {
            llenaClientes();
            llenaFichasReclamo();
        }

       //Método que llena los datos de los clientes en el cuadro combinado
        void llenaClientes()
        {
          var clientes = from c in db.CLIENTE
                         select new
                         {
                           codigo = c.IDE_CLI,
                           nombres = c.NOM_CLI + " " +c.PAT_CLI + " "+c.MAT_CLI
                         };
          cboCliente.DataSource = clientes;
          cboCliente.DisplayMember = "nombres";
          cboCliente.ValueMember = "codigo";
        }

        //Método que muestra las fichas de reclamo
        void llenaFichasReclamo()
        {
            var fichas = from f in db.FICHARECLAMO
                         select new
                         {
                             CODIGO = f.COD_FIC,
                             CONTRATISTA = f.IDE_CON,
                             CLIENTE = f.IDE_CLI,
```

```
                                FECHA = f.FFI_FIC
                            };
                dgFichas.DataSource = fichas;
            }

        private void cboCliente_SelectionChangeCommitted(object sender, EventArgs e)
        {
            var fichas = from f in db.FICHARECLAMO
                         where f.IDE_CLI == cboCliente.SelectedValue.ToString()
                             select new
                             {
                                CODIGO = f.COD_FIC,
                                CONTRATISTA = f.IDE_CON,
                                CLIENTE = f.IDE_CLI,
                                FECHA = f.FFI_FIC
                             };
            dgFichas.DataSource = fichas;
        }
    }
}
```

Debemos tener en cuenta que el cuadro combinado (cboCliente) muestra los datos de los clientes y que al seleccionar uno de ellos se listara sus fichas de reclamo. Para este caso debemos acceder al evento SelectionChangeCommited, para esto primero seleccione el cuadro combinado y desde el panel propiedades seleccione el botón evento, luego busque el evento SelectionChangeCommited y presione doble clic para acceder al método con el mismo nombre.

Caso desarrollado 7 : LinQ to SQL - Listado de fichas de reclamo por año

Implementar una aplicación que permita mostrar los registros de las fichas de reclamo en un determinado año, para lo cual se debe usar la clase LinQ to SQL.

Se tiene en cuenta lo siguiente:

- Implementar un procedimiento almacenado que permite listar los años en los que se registraron las fichas de reclamo de forma descendente sin repetirse.
- Implementar un procedimiento almacenado que permita listar los datos de la ficha de reclamo según el año de registro.
- Agregar al proyecto un archivo de tipo Clases de LinQ to SQL e incluir el procedimiento almacenado que muestra los años y la tabla ficha de reclamo desde el Explorador de servidores.
- Mediante un botón de búsqueda, mostrar los datos de la ficha como código, nombre completo del contratista, nombre completo del cliente y la fecha de registro de la ficha de reclamo. Usar la cláusula inner join para agregar los datos de los contratistas y el cliente en la consulta LinQ to SQL.

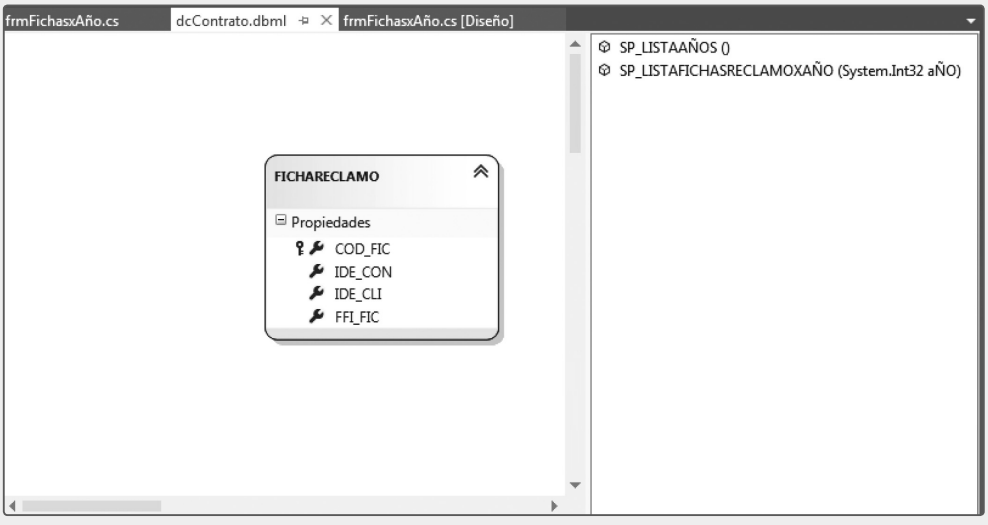
GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjFichasReclamoxAñoLinqToSQL.
2. Agregue un archivo al proyecto presionando clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento** > **Datos** > **Clases de LinQ to SQL** y asigne el nombre dcContrato.

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS AÑOS SEGUN LA FICHA DE RECLAMO
CREATE PROC SP_LISTAAÑOS
AS
        SELECT DISTINCT YEAR(F.FFI_FIC) AS AÑO
                FROM FICHARECLAMO F
                ORDER BY 1 DESC
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LAS FICHAS DE RECLAMO SEGUN EL AÑO
CREATE PROC SP_LISTAFICHASRECLAMOXAÑO(@AÑO INT)
AS
        SELECT F.COD_FIC AS CODIGO,
        C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
        CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
                F.FFI_FIC AS FECHA
                FROM FICHARECLAMO F
                JOIN CONTRATISTA C ON C.IDE_CON = F.IDE_CON
                JOIN CLIENTE CL ON CL.IDE_CLI = F.IDE_CLI
                WHERE YEAR(F.FFI_FIC)=@AÑO
GO
```

3. Agregue, al entorno Clases LinQ to SQL, la tabla ficha de reclamo, así como los procedimientos almacenados SP_LISTAAÑOS y SP_LISTAFICHASRECLAMOXAÑO desde el Explorador de servidores, como se muestra en la siguiente imagen:



4. Luego, coloque el siguiente código en el formulario frmFichasxAño:

```
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 pjFichasReclamoxAñoLinqToSQL
{
    public partial class frmFichasxAño : Form
    {
        dcContratoDataContext db = new dcContratoDataContext();

        public frmFichasxAño()
        {
            InitializeComponent();
        }

        private void frmFichasxAño_Load(object sender, EventArgs e)
        {
            llenaAños();
            llenaFichas();
        }
```

```
        //Método que llena los años en el cuadro combinado
        void llenaAños()
        {
            cboAño.DataSource = db.SP_LISTAAÑOS();
            cboAño.DisplayMember = "AÑO";
        }

        //Método que llena los datos de la ficha de reclamo en el DataGridView
        void llenaFichas()
        {
            var fichas = from f in db.FICHARECLAMO
                         select new
                         {
                             CODIGO = f.COD_FIC,
                             CONTRATISTA = f.IDE_CON,
                             CLIENTE = f.IDE_CLI,
                             FECHA = f.FFI_FIC
                         };
            dgFichas.DataSource = fichas;
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            int año = int.Parse(cboAño.Text);
            dgFichas.DataSource = db.SP_LISTAFICHASRECLAMOXAÑO(año);
        }
    }
}
```

Caso desarrollado 8 : LinQ to SQL - Registro del nuevo equipo

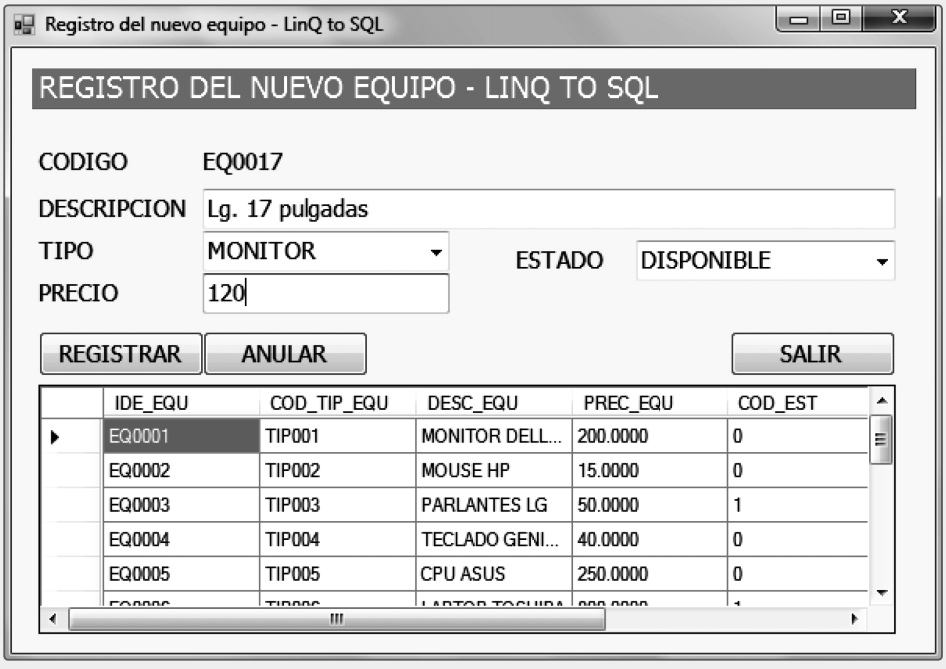
Implementar una aplicación que permita agregar un nuevo registro de datos en la tabla Equipo, para lo cual se debe usar la clase LinQ to SQL.

Se tiene en cuenta lo siguiente:

- Agregar al proyecto un archivo de tipo Clases de LinQ to SQL y agregar las tablas Equipo, tipo de equipo y estado de equipo.
- Mostrar el tipo y estado de equipo en cuadros combinados respectivamente.

- Autogenerar, de manera automática, el nuevo código del equipo, el cual se basa en el último código de equipo registrado más uno.
- Mostrar los datos de los equipos registrados en un control DataGridView.
- Mediante el botón registrar, agregar el registro de datos en la tabla Equipo.

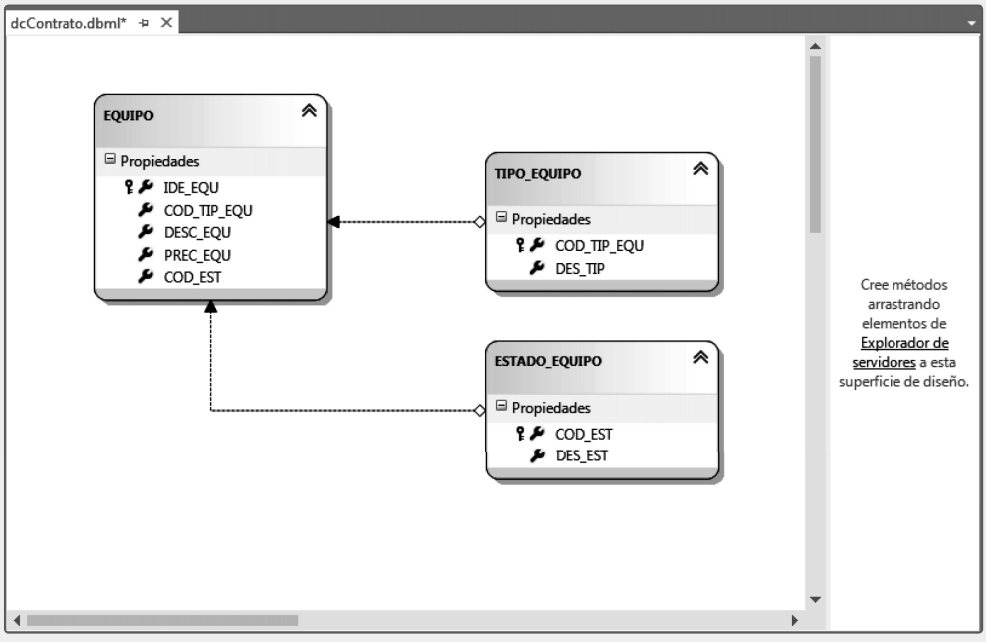
GUI Propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjMantenimientoLinQtoSQL.
2. Agregue un archivo al proyecto presionando clic derecho sobre el proyecto, luego en **Agregar** > **Nuevo elemento** > **Datos** > **Clases de LinQ to SQL** y asigne el nombre dcContrato.

3. Agregue las tablas Equipo, Tipo y estado de equipo al entorno Clases LinQ to SQL, arrastrando desde la ventana Explorador de servidores, como se muestra en la siguiente imagen:



4. Luego, colocar el siguiente código en el formulario frmMantenimientoEquipos:

```
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 pjMantenimientoLinQtoSQL
{
    public partial class frmMantenimientoEquipos : Form
    {
        //Definicion GLOBAL
        dcContratoDataContext db = new dcContratoDataContext();

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

```
private void frmMantenimientoEquipos_Load(object sender, EventArgs e)
 {
     llenaEstado();
     llenaTipo();
     lblCodigo.Text = determinaCodigo();
     llenaEquipos();
 }

 //Método que llena los estados en un cuadro combinado
 void llenaEstado()
 {
     var estado = from e in db.ESTADO_EQUIPO
                  select new
                  {
                      codigo = e.COD_EST,
                      estado = e.DES_EST
                  };
     cboEstado.DataSource= estado;
     cboEstado.DisplayMember = "ESTADO";
     cboEstado.ValueMember = "CODIGO";
 }

 //Método que llena los tipos de equipo en un cuadro combinado
 void llenaTipo()
 {
     var tipo = from t in db.TIPO_EQUIPO
                select new
                {
                    codigo = t.COD_TIP_EQU,
                    tipo = t.DES_TIP
                };
     cboTipo.DataSource = tipo;
     cboTipo.DisplayMember = "TIPO";
     cboTipo.ValueMember = "CODIGO";
 }

 //Método que devuelve el código autogenerado de equipo
 string determinaCodigo()
 {
     var equipos = (from e in db.EQUIPO
                    orderby e.IDE_EQU descending
                    select new { e.IDE_EQU }).Take(1);

     string codigo = "";
     foreach (var equ in equipos.ToList())
         codigo = equ.IDE_EQU;

  return "EQ"+(int.Parse(codigo.Substring(2, 4))+1).ToString("0000");
 }
```

```
        //Método que llena los datos de los equipos en un DataGridView
        void llenaEquipos()
        {
            var equipos = from e in db.EQUIPO
                          select e;
            dgEquipos.DataSource = equipos;
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            EQUIPO objE = new EQUIPO();

            objE.IDE_EQU = lblCodigo.Text;
            objE.COD_TIP_EQU = cboTipo.SelectedValue.ToString();
            objE.DESC_EQU = txtDescripcion.Text;
            objE.PREC_EQU = int.Parse(txtPrecio.Text);
            objE.COD_EST = cboEstado.SelectedValue.ToString();

            db.EQUIPO.InsertOnSubmit(objE);
            db.SubmitChanges();
            llenaEquipos();
        }
    }
}
```

En el botón registrar se debe crear un objeto de la clase Equipo, el cual fue incorporado en el entorno Clases LinQ to SQL. Mediante este objeto, se podrá enviar información del nuevo equipo. Luego, usar el método InsertOnSubmit para agregar el nuevo registro a la clase y finalmente grabar con el método SubmitChanges. Se debe tener en cuenta que db hace referencia a la clase DataContext de contrato que permite tener acceso a la tabla Equipo del LinQ to SQL.

MACRO

6

Capítulo

Transacciones - Ado Entity

Capacidad terminal:

Implementar aplicaciones cliente-servidor usando las transacciones. Asimismo, se podrá implementar aplicaciones de consultas y mantenimiento con el objeto relacional Ado Entity.

EDITORIAL MACRO®

Contenido

- Transacciones
- Clase SqlTransaction
- Método begin Transaction
  Caso desarrollado 1: Mantenimiento de fichas de devolución
- Ado Entity
  Caso desarrollado 2: Ado Entity - Listado de equipo
  Caso desarrollado 3: Ado Entity - Listado de equipo por estado y tipo
  Caso desarrollado 4: Ado Entity - Mantenimiento de equipos

EDITORIAL
MACRO®

6.1 Transacciones

Cuando se desarrolla una aplicación, se piensa en todos los procesos que se debe contar, así como la forma de usarlas en la aplicación, pero siempre se asume que todo será correcto. ¿Qué pasaría si se registra información a varias tablas desde una misma aplicación, y en la mitad del proceso ocurre un error? Se puede pensar que alguna información se ha registrado en la base de datos, así como que otra información se ha perdido. Pero para asegurar que la información llegue correcta solo cuando todos los procesos sean correctos, se usan las transacciones.

Entonces, se puede decir que una transacción aborda casos en que el estado resultante de la base de datos como grabación, actualización o eliminación depende del éxito total de una serie de operaciones especificadas por el desarrollador. Esto podría suceder debido a que las operaciones sucesivas pueden modificar el resultado de las operaciones anteriores. En estos casos, si se produce un error en una operación, el estado resultante puede ser indeterminado.

Las transacciones permiten agrupar una serie de métodos que garantizan la integridad del resultado en todas las tablas que se referencie. Si los métodos ejecutan correctamente todos sus procesos, entonces se estaría asegurando que la información llegue correctamente a la base de datos; caso contrario, la transacción se detiene y caso contrario, si la transacción se detiene, no alcanzaría a realizarse alguna grabación en la base de datos. A este último escenario se le denomina reversión, el cual implicaría la anulación de todo tipo de acción en la base de datos.

Finalmente, se puede decir que una transacción debe ajustarse a las propiedades ACID (atomicidad, coherencia, aislamiento y durabilidad) para poder garantizar la coherencia e integridad de sus datos. La mayoría de los sistemas de bases de datos relacionales, como Microsoft SQL Server, admiten transacciones al proporcionar funciones de bloqueo, registro y administración de transacciones, cada vez que una aplicación cliente realiza una operación de actualización, inserción o eliminación.

6.2 Clase SqlTransaction

Representa un proceso de transacción Transact-SQL que se realiza cuando se usa una base de datos de SQL Server. La librería que soporta la clase es el espacio de nombres System.Data.SqlClient.

La aplicación crea un objeto SqlTransaction mediante una llamada a BeginTransaction en el objeto SqlConnection. Todas las operaciones siguientes asociadas a la transacción (por ejemplo, la confirmación o la anulación de la transacción) se realizan en el objeto SqlTransaction.

EDITORIAL MACRO®

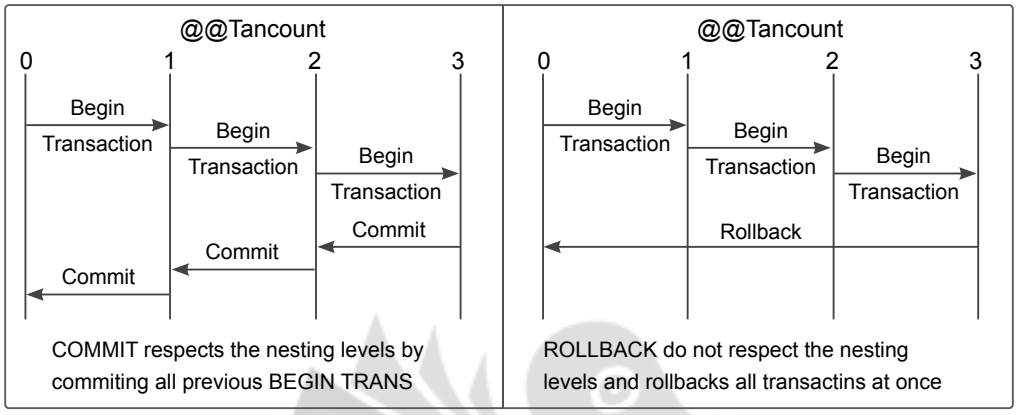


Figura 6.1 Forma de trabajo de commint y rollback on en un proceso de transacción.

Propiedad:

IsolationLevel	Especifica los niveles de aislamiento que puede tener el motor de base de datos. Toda vez que se trabaje con transacciones se debe especificar un IsolationLevel.

Métodos:

Commit()	Confirma la transacción y realiza todas las operaciones en la base de datos. `Transaccion.Commit();`
Dispose()	Libera todos los recursos que no se usará en la transacción. `Transaccion.Dispose();`
Rollback()	Deshace la transacción desde el punto de estado pendiente hasta el inicio de las operaciones, es decir, anula todo el lote de transacción. `Transaccion.Rollback();`

6.3 Método beginTransaction

Marca un punto de inicio en la aplicación desde el cual iniciará la transacción. Es en este punto donde volverá la transacción si es que ocurre algún error en el proceso, caso contrario asegura la información.

El código para marcar el inicio de la transacción:

```
SqlTransaction tr = cn.BeginTransaction(IsolationLevel.Serializable);
```

Y para referenciar la transacción en la sentencia de inserción, actualización o eliminación, se usa el siguiente código:

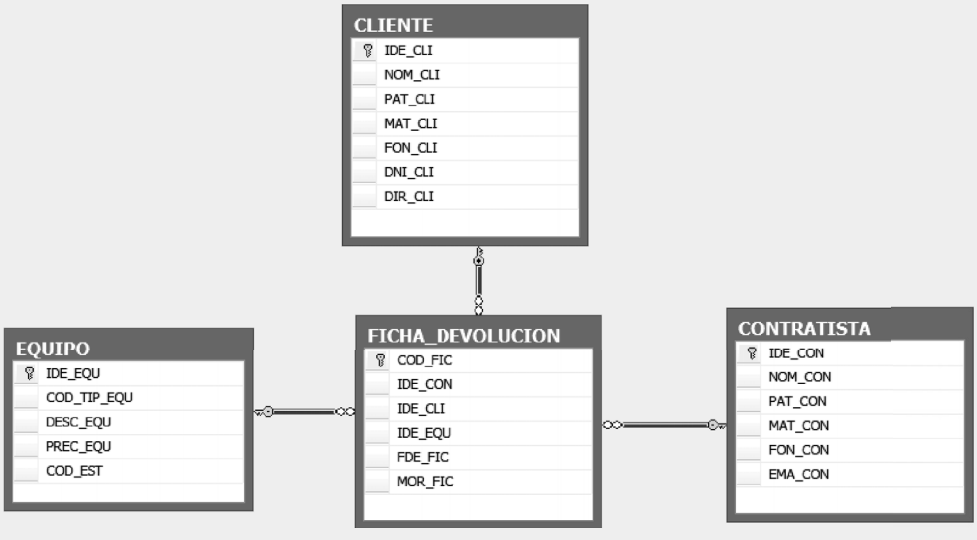
```
SqlCommand cmd = new SqlCommand("SP_NUEVAFICHADEVOLUCION", cn,tr);
```

Caso desarrollado 1 : Mantenimiento de fichas de devolución

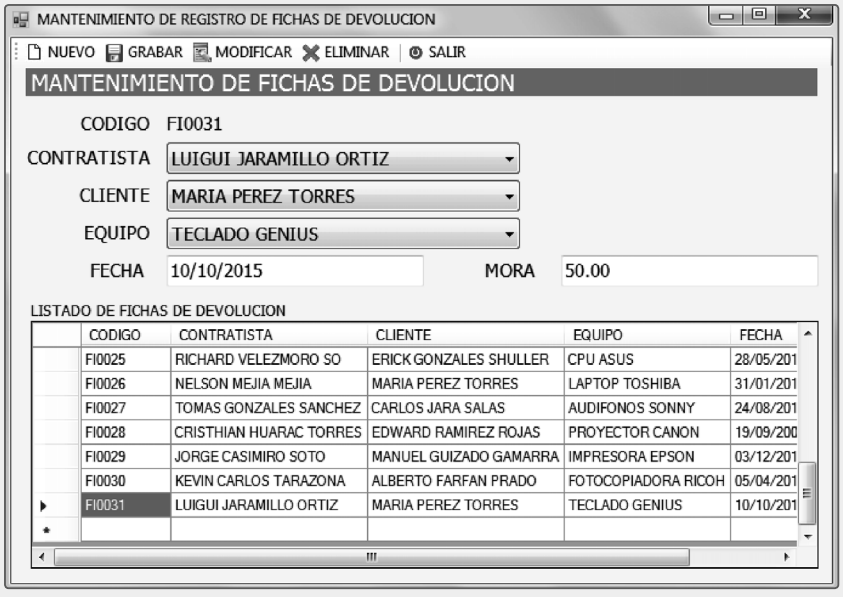
Implementar una aplicación que permita realizar el mantenimiento de los registros de la tabla ficha de devolución, de tal forma que se pueda listar, agregar, modificar y eliminar las fichas de devolución.

Se tiene en cuenta lo siguiente:

- Implementar procedimientos almacenados que permitan listar los datos del cliente, contratistas y equipos, además de determinar el último código de ficha, nueva ficha, modificar y eliminar ficha de devolución.
- Se debe autogenerar el código de la ficha de devolución con respecto al último registro realizado.
- Para registrar una ficha de devolución, se debe seleccionar un cliente, un contratista y un equipo a partir de cuadros combinados.
- Usar la siguiente relación de tablas:



GUI Propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjMantenimientoFichaDevolucion.
2. Configure la cadena de conexión en el archivo app.config.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.6" />
    </startup>
<connectionStrings>
  <add name="cn" connectionString="server=MTORRES-PC;
                                   database=contrato;
                                   integrated security=SSPI" />
</connectionStrings>
</configuration>
```

3. Agregue la referencia System.Configuration al proyecto.

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS DE LOS CONTRATISTAS
CREATE PROC SP_LISTACONTRATISTA
AS
       SELECT C.IDE_CON AS CODIGO,
              C.NOM_CON+SPACE(1)+C.PAT_CON+C.MAT_CON AS CONTRATISTA
              FROM CONTRATISTA C
GO
```

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS DE LOS CLIENTES
CREATE PROC SP_LISTACLIENTES
AS
        SELECT C.IDE_CLI AS CODIGO,
                C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE
                FROM CLIENTE C
GO
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS DE LOS EQUIPOS
CREATE PROC SP_LISTAEQUIPO
AS
        SELECT IDE_EQU AS CODIGO,
                DESC_EQU AS EQUIPO
                FROM EQUIPO E
GO
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS DE LAS FICHAS DE DEVOLUCION
CREATE PROC SP_LISTAFICHADEVOLUCION
AS
        SELECT FD.COD_FIC AS CODIGO,
        C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
        CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
                E.DESC_EQU AS EQUIPO,
                FD.FDE_FIC AS FECHA
                FROM FICHA_DEVOLUCION FD
                JOIN CONTRATISTA C ON FD.IDE_CON=C.IDE_CON
                JOIN CLIENTE CL ON FD.IDE_CLI=CL.IDE_CLI
                JOIN EQUIPO E ON FD.IDE_EQU=E.IDE_EQU
GO
--PROCEDIMIENTO ALMACENADO QUE MUESTRE EL ULTIMO CÓDIGO DE FICHA DE DEVOLUCION
CREATE PROC SP_ULTIMAFICHADEVOLUCION
AS
        SELECT TOP 1 FD.COD_FIC
                FROM FICHA_DEVOLUCION FD
                ORDER BY 1 DESC
GO
--PROCEDIMIENTO ALMACENADO QUE REGISTRA UNA FICHA DE DEVOLUCION
CREATE PROC SP_NUEVAFICHADEVOLUCION(@IFI CHAR(6), @ICO CHAR(6),
                                    @ICL CHAR(6), @IEQ CHAR(6),
                                    @FDE DATE, @MOR MONEY)
AS
        INSERT INTO FICHA_DEVOLUCION
                VALUES (@IFI,@ICO,@ICL,@IEQ,@FDE,@MOR)
GO
--PROCEDIMIENTO ALMACENADO QUE ACTUALIZA LOS DATOS DE UNA FICHA DE DEVOLUCION
CREATE PROC SP_ACTUALIZAFICHADEVOLUCION(@IFI CHAR(6), @ICO CHAR(6),
                                    @ICL CHAR(6), @IEQ CHAR(6),
                                    @FDE DATE, @MOR MONEY)
AS
        UPDATE FICHA_DEVOLUCION
                SET IDE_CON=@ICO,IDE_CLI=@ICL,IDE_EQU=@IEQ,FDE_FIC=@FDE,MOR_FIC=@MOR
                WHERE COD_FIC=@IFI
GO
--PROCEDIMIENTO ALMACENADO QUE ELIMINA UNA FICHA DE DEVOLUCION
CREATE PROC SP_ELIMINAFICHADEVOLUCION(@IFI CHAR(6))
AS
        DELETE FICHA_DEVOLUCION
                WHERE COD_FIC=@IFI
GO
```

4. Agregue la clase Conexión y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Configuration;
using System.Data.SqlClient;

namespace pjMantenimientoFichaDevolucion
{
    public class Conexion
    {
        public SqlConnection getConecta()
        {
            SqlConnection cn = new SqlConnection(ConfigurationManager
                                    .ConnectionStrings["cn"].ConnectionString);
            return cn;
        }
    }
}
```

5. Agregue la clase FichaDevolucion y coloque el siguiente código:

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

namespace pjMantenimientoFichaDevolucion
{
    public class FichaDevolucion
    {
        public string codigo { get; set; }
        public string contratista { get; set; }
        public string cliente { get; set; }
        public string equipo { get; set; }
        public DateTime fecha { get; set; }
        public double mora { get; set; }
    }
}
```

6. Agregue la clase LogicaNegocio y coloque el siguiente código:

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

namespace pjMantenimientoFichaDevolucion
{
    public class LogicaNegocio
    {
        //GLOBAL
        Conexion objCon = new Conexion();
        SqlConnection cn = new SqlConnection();

        //Método para listar los contratistas
        public DataTable listaContratistas()
        {
            cn = objCon.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

        //Método para listar los clientes
        public DataTable listaClientes()
        {
            cn = objCon.getConecta();
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTES", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

        //Método para listar los equipos
        public DataTable listaEquipos()
        {
            cn = objCon.getConecta();
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTAEQUIPO", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

        //Método para listar las fichas de devolución
        public DataTable listaFichasDevolucion()
        {
           cn = objCon.getConecta();
         SqlDataAdapter da = new SqlDataAdapter("SP_LISTAFICHADEVOLUCION", cn);
```

```
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }

        //Método para generar un nuevo codigo de ficha de devolución
        public string generaCodigo()
        {
            cn = objCon.getConecta();
            cn.Open();
          SqlCommand cmd = new SqlCommand("SP_ULTIMAFICHADEVOLUCION", cn);
            cmd.CommandType = CommandType.StoredProcedure;
            return "FI"+(int.Parse(cmd.ExecuteScalar().ToString()
                                        .Substring(2,4))+1).ToString("0000");
        }

    //Método para agregar una nueva ficha de devolución
    public int agregaFicha(FichaDevolucion objF)
    {
       cn = objCon.getConecta();
       cn.Open();

     using (SqlTransaction tr=cn.BeginTransaction(IsolationLevel.Serializable))
       {
        SqlCommand cmd = new SqlCommand("SP_NUEVAFICHADEVOLUCION", cn,tr);
          cmd.CommandType = CommandType.StoredProcedure;

          cmd.Parameters.Add("@ifi", SqlDbType.Char).Value = objF.codigo;
          cmd.Parameters.Add("@ico", SqlDbType.Char).Value = objF.contratista;
          cmd.Parameters.Add("@icl", SqlDbType.Char).Value = objF.cliente;
          cmd.Parameters.Add("@ieq", SqlDbType.Char).Value = objF.equipo;
          cmd.Parameters.Add("@fde", SqlDbType.Date).Value = objF.fecha;
          cmd.Parameters.Add("@mor", SqlDbType.Money).Value = objF.mora;

          try
          {
             int i = cmd.ExecuteNonQuery();
             tr.Commit();
             return i;
          }
          catch (SqlException ex)
          {
             tr.Rollback();
          }
       }
       return 0;
    }

    //Método para actualizar una  ficha de devolución
    public int actualizaFicha(FichaDevolucion objF)
    {
      cn = objCon.getConecta();
      cn.Open();
     using (SqlTransaction tr = cn.BeginTransaction(IsolationLevel.Serializable))
```

```
    {
      SqlCommand cmd = new SqlCommand("SP_ACTUALIZAFICHADEVOLUCION", cn, tr);
       cmd.CommandType = CommandType.StoredProcedure;

       cmd.Parameters.Add("@ifi", SqlDbType.Char).Value = objF.codigo;
       cmd.Parameters.Add("@ico", SqlDbType.Char).Value = objF.contratista;
       cmd.Parameters.Add("@icl", SqlDbType.Char).Value = objF.cliente;
       cmd.Parameters.Add("@ieq", SqlDbType.Char).Value = objF.equipo;
       cmd.Parameters.Add("@fde", SqlDbType.Date).Value = objF.fecha;
       cmd.Parameters.Add("@mor", SqlDbType.Money).Value = objF.mora;

       try
       {
          int i = cmd.ExecuteNonQuery();
          tr.Commit();
          return i;
       }
       catch (SqlException ex)
       {
          tr.Rollback();
       }
    }
    return 0;
  }

  //Método para eliminar una ficha de devolución
  public int eliminaFicha(FichaDevolucion objF)
  {
    cn = objCon.getConecta();
    cn.Open();

   using (SqlTransaction tr = cn.BeginTransaction(IsolationLevel.Serializable))
    {
      SqlCommand cmd = new SqlCommand("SP_ELIMINAFICHADEVOLUCION", cn, tr);
       cmd.CommandType = CommandType.StoredProcedure;

       cmd.Parameters.Add("@ifi", SqlDbType.Char).Value = objF.codigo;

       try
       {
          int i = cmd.ExecuteNonQuery();
          tr.Commit();
          return i;
       }
       catch (SqlException ex)
       {
          tr.Rollback();
       }
    }
    return 0;
   }
  }
}
```

7. Luego, coloque el siguiente código en el formulario frmMantenimientoEquipos:

```
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 pjMantenimientoFichaDevolucion
{
    public partial class frmMantenimiento : Form
    {
        //GLOBAL
        LogicaNegocio objL = new LogicaNegocio();

        public frmMantenimiento()
        {
            InitializeComponent();
        }

        private void frmMantenimiento_Load(object sender, EventArgs e)
        {
            generaCodigo();
            llenaCliente();
            llenaContratista();
            llenaEquipo();
            llenaFicha();
        }

        void generaCodigo()
        {
            lblCodigo.Text = objL.generaCodigo();
        }

        void llenaContratista()
        {
            cboContratista.DataSource = objL.listaContratistas();
            cboContratista.DisplayMember = "CONTRATISTA";
            cboContratista.ValueMember = "CODIGO";
        }
        void llenaCliente()
        {
            cboCliente.DataSource = objL.listaClientes();
            cboCliente.DisplayMember = "CLIENTE";
            cboCliente.ValueMember = "CODIGO";
        }
        void llenaFicha()
```

```
        {
            dgFichas.DataSource = objL.listaFichasDevolucion();
        }
        void llenaEquipo()
        {
            cboEquipo.DataSource = objL.listaEquipos();
            cboEquipo.DisplayMember = "EQUIPO";
            cboEquipo.ValueMember = "CODIGO";
        }

        private void tsNuevo_Click(object sender, EventArgs e)
        {
            generaCodigo();
            limpiaControles();
        }

        private void tsGrabar_Click(object sender, EventArgs e)
        {
          try
          {
            FichaDevolucion objF = new FichaDevolucion();
            objF.codigo = lblCodigo.Text;
            objF.contratista = cboContratista.SelectedValue.ToString();
            objF.cliente = cboCliente.SelectedValue.ToString();
            objF.equipo = cboEquipo.SelectedValue.ToString();
            objF.fecha = DateTime.Parse(txtFecha.Text);
            objF.mora = double.Parse(txtMora.Text);

            int i = objL.agregaFicha(objF);
          MessageBox.Show(i + " registro de ficha de devolución correcto..!!");
                llenaFicha();
          }
          catch (Exception)
          {
            MessageBox.Show("Ocurrió un error al grabar la ficha");
          }
        }

        private void tsModificar_Click(object sender, EventArgs e)
        {
          try
          {
                FichaDevolucion objF = new FichaDevolucion();
                objF.codigo = lblCodigo.Text;
                objF.contratista = cboContratista.SelectedValue.ToString();
                objF.cliente = cboCliente.SelectedValue.ToString();
                objF.equipo = cboEquipo.SelectedValue.ToString();
                objF.fecha = DateTime.Parse(txtFecha.Text);
                objF.mora = double.Parse(txtMora.Text);
```

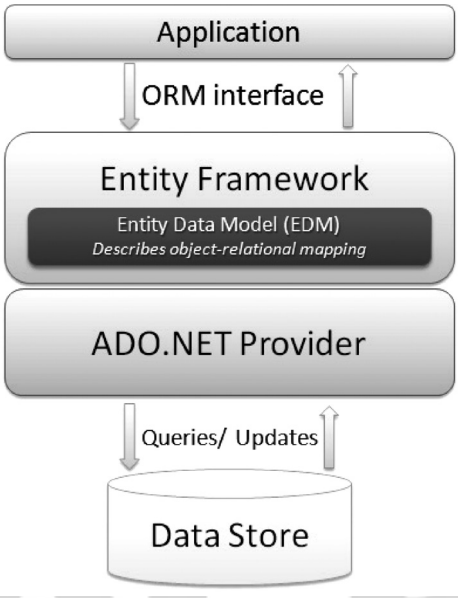
```
                int i = objL.actualizaFicha(objF);
                MessageBox.Show(i + " actualización de ficha de
                                                devolución correcto..!!");
                llenaFicha();
            }
            catch (Exception)
            {
               MessageBox.Show("Ocurrió un error al actualizar la ficha");
            }
        }

        private void tsEliminar_Click(object sender, EventArgs e)
        {
            try
            {
                FichaDevolucion objF = new FichaDevolucion();
                objF.codigo = lblCodigo.Text;

                int i = objL.eliminaFicha(objF);
                MessageBox.Show(i + " eliminación de ficha de devolución
                                                        correcto..!!");
                llenaFicha();
            }
            catch (Exception)
            {
               MessageBox.Show("Ocurrió un error al actualizar la ficha");
            }
        }

       private void dgFichas_MouseDoubleClick(object sender, MouseEventArgs e)
        {
          lblCodigo.Text = dgFichas.CurrentRow.Cells[0].Value.ToString();
          cboContratista.Text = dgFichas.CurrentRow.Cells[1].Value.ToString();
           cboCliente.Text = dgFichas.CurrentRow.Cells[2].Value.ToString();
          cboEquipo.Text = dgFichas.CurrentRow.Cells[3].Value.ToString();
          txtFecha.Text = DateTime.Parse(dgFichas.CurrentRow.Cells[4].Value
                                            .ToString()).ToShortDateString();
           txtMora.Text = double.Parse(dgFichas.CurrentRow.Cells[5].Value
                                            .ToString()).ToString("0.00");
        }

        void limpiaControles()
        {
            lblCodigo.Text = "";
            txtFecha.Clear();
            txtMora.Clear();
            cboContratista.Focus();
        }
    }
}
```

6.4 Ado Entity

Es un entorno objeto-relacional que permite a los desarrolladores de Visual C# trabajar con datos relacionales obtenidos desde SQL Server y usarlos como objetos en una aplicación. Una de las principales ventajas es que permite optimizar el código de acceso a datos que se realiza para conectarse a una base de datos para obtener información.

Se puede mencionar que:

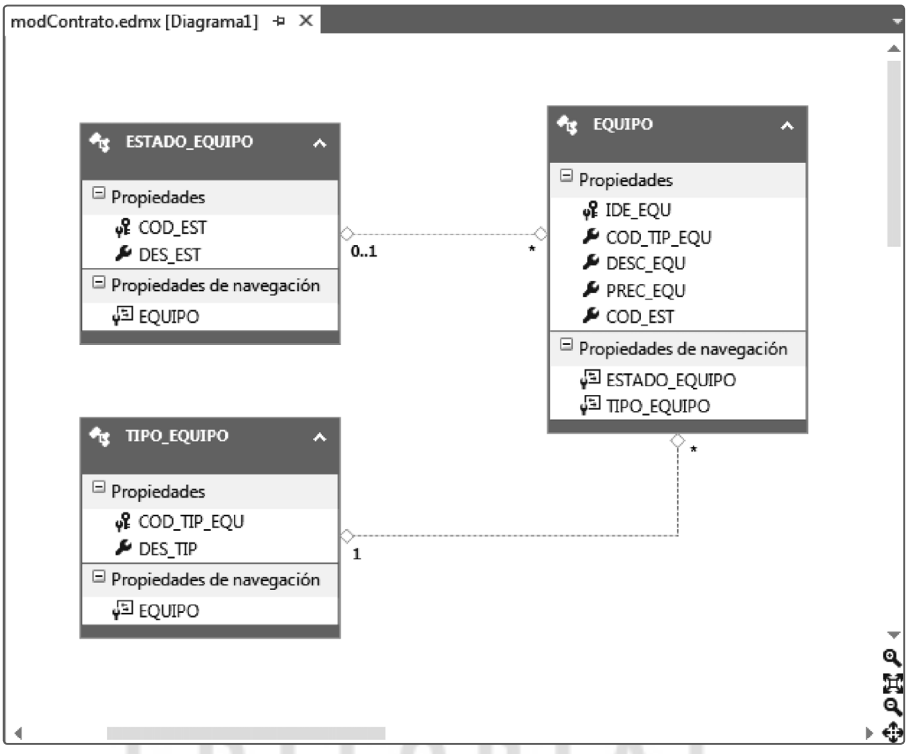
- Permite trabajar con los datos en forma de objetos. Esto permite no preocuparse de la conexión a la base o el manejo de las tablas en SQL.
- Genera un nivel de abstracción más elevado a la hora de trabajar con datos y permite tener aplicaciones con menos código.



Fuente: <https://i-msdn.sec.s-msft.com/en-us/data/aa937709.EF_AtAGlance_1c(en-us,MSDN.10).png>

Los ORM (mapa del objeto relacional) ayudan a mantener el diseño de la base de datos separado del diseño del dominio, lo cual hace que las aplicaciones de plataforma sean más simples de mantener y escalar.

A continuación, observe el diseñador del modelo relacional de Ado Entity asociado a las tablas equipo, estado y tipo de equipo, el cual, al ser empleado en la aplicación, hará que estos sean referenciados como objetos:



EDITORIAL

Caso desarrollado 2 : Ado Entity - Listado de equipo

Implementar una aplicación que permita listar los registros de los equipos usando Ado Entity.

Se tiene en cuenta lo siguiente:

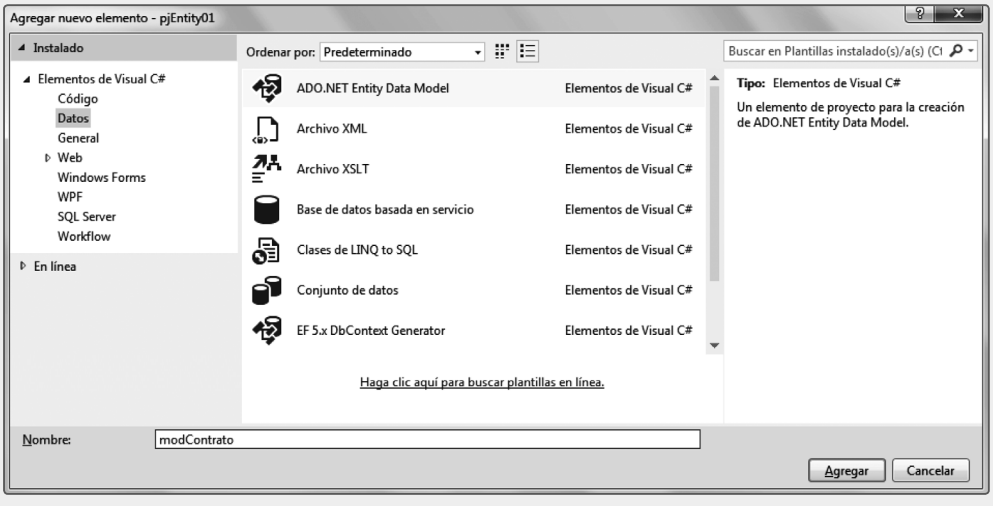
- Agregar un elemento de tipo Ado Entity al proyecto.
- Listar los datos de los equipos en un control DataGridView.

GUI Propuesta:

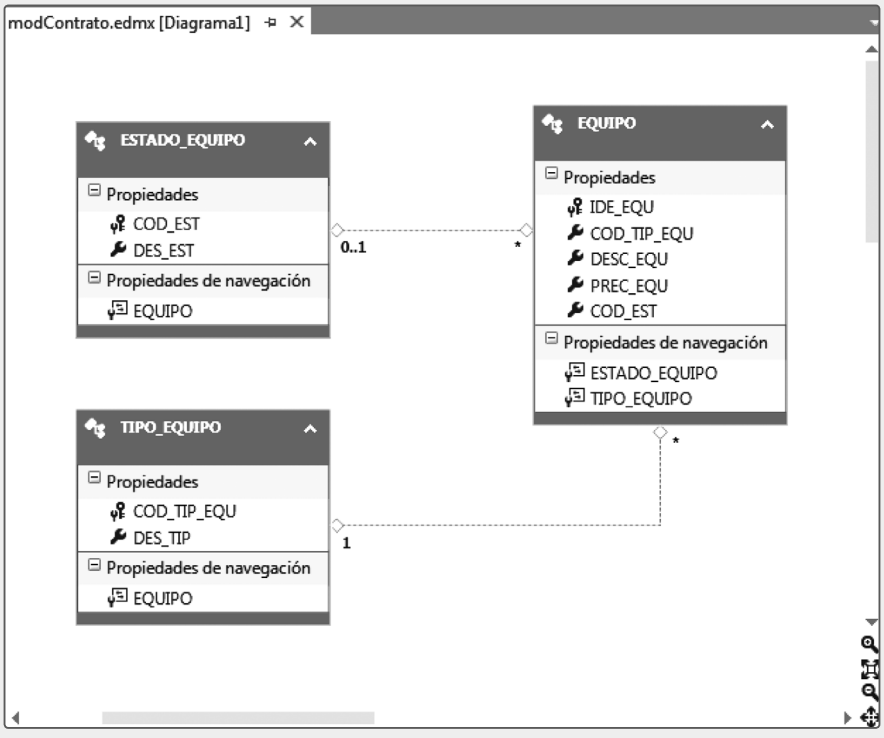
CODIGO	TIPO	DESCRIPCION	PRECIO	ESTADO
EQ0001	TIP001	MONITOR DELL 1707F	200.0000	0
EQ0002	TIP002	MOUSE HP	15.0000	0
EQ0003	TIP003	PARLANTES LG	50.0000	1
EQ0004	TIP004	TECLADO GENIUS	40.0000	0
EQ0005	TIP005	CPU ASUS	250.0000	0
EQ0006	TIP006	LAPTOP TOSHIBA	800.0000	1
EQ0007	TIP007	AUDIFONOS SONNY	30.0000	0
EQ0008	TIP008	PROYECTOR CANON	500.0000	1
EQ0009	TIP009	IMPRESORA EPSON	200.0000	0

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjEntity01.
2. Agregue un nuevo elemento al proyecto, seleccione la categoría Datos, luego ADO. NET Entity Data Model y asigne el nombre modContrato, como se muestra en la siguiente imagen:



3. Agregue las tablas equipo, estado y tipo, como se muestra en la siguiente imagen:



Si necesita agregar una tabla, presione clic derecho en el diseñador y seleccione **Actualizar modelo desde base de datos...** Luego, seleccione una tabla para agregarla al diseñador.

4. Verifique el código de conexión en el archivo app.config.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
    http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.
    ConfigFile.EntityFrameworkSection, EntityFramework, Version=5.0.0.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
```

```
  <connectionStrings>
    <add name="CONTRATOEntities" connectionString="metadata=res://*/
    modContrato.csdl|res://*/modContrato.ssdl|res://*/modContrato.
    msl;provider=System.Data.SqlClient;provider connection
    string=&quot;data source=.;initial catalog=CONTRATO;integrated
    security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
    providerName="System.Data.EntityClient" />
  </connectionStrings>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.
    LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="v11.0" />
      </parameters>
    </defaultConnectionFactory>
  </entityFramework>
</configuration>
```

5. Agregue el siguiente código en el formulario frmListaEquipo:

```
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 pjEntity01
{
    public partial class FrmListaEquipo : Form
    {
        //Definición GLOBAL
        CONTRATOEntities db = new CONTRATOEntities();

        public FrmListaEquipo()
        {
            InitializeComponent();
        }

        private void FrmListaEquipo_Load(object sender, EventArgs e)
        {
            //LinQ para listar los datos del Equipo
            var misContratos = from c in db.EQUIPO
                               select new
                               {
                                   CODIGO = c.IDE_EQU,
                                   TIPO = c.COD_TIP_EQU,
```

```
                                    DESCRIPCION = c.DESC_EQU,
                                    PRECIO = c.PREC_EQU,
                                    ESTADO = c.COD_EST

                                };

            //Enviando los datos al control DataGridView
            dgContrato.DataSource = misContratos.ToList();
        }
    }
}
```

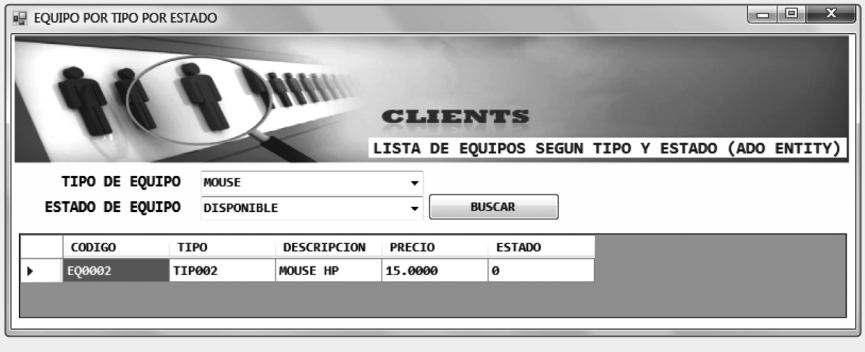
Caso desarrollado 3 : Ado Entity - Listado de equipo por estado y tipo

Implementar una aplicación que permita listar los registros de los equipos según el tipo y estado del mismo usando Ado Entity.

Se tiene en cuenta lo siguiente:

- Agregar un elemento de tipo Ado Entity al proyecto.
- Llenar los cuadros combinados con los tipos y estados de los equipos.
- Listar los datos de los equipos en un control DataGridView.

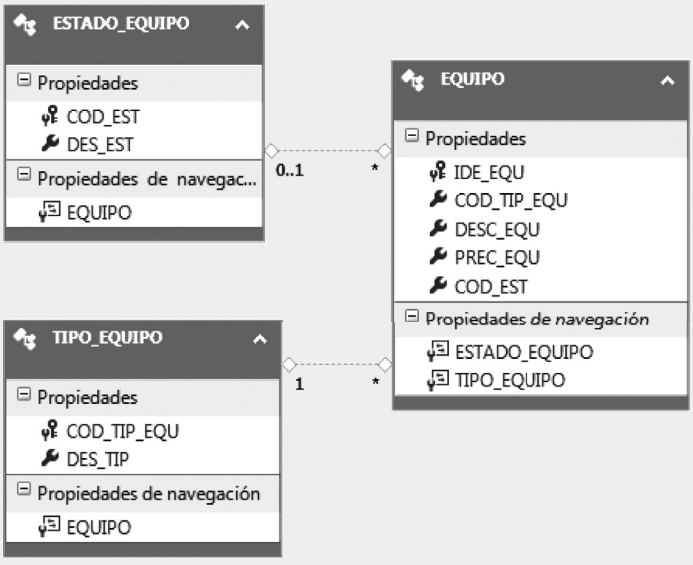
GUI Propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjEntity02.
2. Agregue un nuevo elemento al proyecto, seleccione la categoría Datos, luego ADO.NET Entity Data Model y asigne el nombre modContrato.

3. Agregue las tablas equipo, estado y tipo, como se muestra en la siguiente imagen:



Si se necesita agregar una tabla, presione clic derecho en el diseñador y seleccione **Actualizar modelo desde base de datos...** Luego, seleccione una tabla para agregarla al diseñador.

4. Verifique el código de conexión en el archivo app.config:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
    http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.
    ConfigFile.EntityFrameworkSection, EntityFramework, Version=5.0.0.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <connectionStrings>
    <add name="CONTRATOEntities" connectionString="metadata=res://*/
    modContrato.csdl|res://*/modContrato.ssdl|res://*/modContrato.
    msl;provider=System.Data.SqlClient;provider connection
    string=&quot;data source=.;initial catalog=CONTRATO;integrated
```

```
    security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
    providerName="System.Data.EntityClient" />
  </connectionStrings>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.
    LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="v11.0" />
      </parameters>
    </defaultConnectionFactory>
  </entityFramework>
</configuration>
```

5. Agregue el siguiente código en el formulario FrmEquipoXTipoXEstado:

```
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 PjEntity02
{
    public partial class FrmEquipoXTipoXEstado : Form
    {
        //GLOBAL
        CONTRATOEntities db = new CONTRATOEntities();

        public FrmEquipoXTipoXEstado()
        {
            InitializeComponent();
        }

       private void FrmEquipoXTipoXEstado_Load(object sender, EventArgs e)
        {
            llenarTipo();
            llenarEstado();
        }

        //Método que llena los tipos de equipos
        void llenarTipo()
        {
            cboTipo.DataSource = db.TIPO_EQUIPO.ToList();
            cboTipo.DisplayMember = "des_tip";
```

```
            cboTipo.ValueMember = "cod_tip_equ";
        }

        //Método que llena los estados de los equipos
        void llenarEstado()
        {
            cboEstado.DataSource = db.ESTADO_EQUIPO.ToList();
            cboEstado.DisplayMember = "des_est";
            cboEstado.ValueMember = "cod_est";
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            //Capturando información de los cuadros combinados
            string tipo = cboTipo.SelectedValue.ToString();
            string estado = cboEstado.SelectedValue.ToString();

            //Listando los equipos según el tipo y el estado
            var equipos = from eq in db.EQUIPO
                          where eq.COD_TIP_EQU==tipo && eq.COD_EST==estado
                           select new
                           {
                               CODIGO = eq.IDE_EQU,
                               TIPO = eq.COD_TIP_EQU,
                               DESCRIPCION = eq.DESC_EQU,
                               PRECIO =eq.PREC_EQU,
                               ESTADO = eq.COD_EST
                           };

            dgContrato.DataSource = equipos.ToList();
        }
    }
}
```

Caso desarrollado 4 : Ado Entity - Mantenimiento de equipos

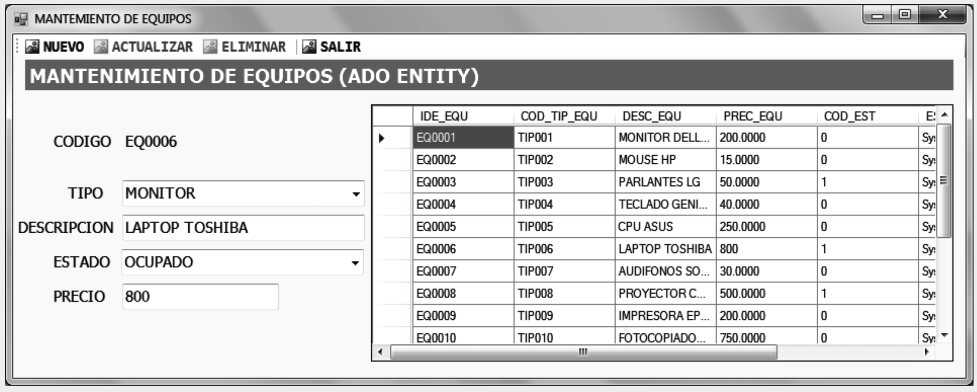
Implementar una aplicación que permita controlar el mantenimiento de registros de la tabla equipo, de tal forma que se pueda agregar, modificar y eliminar los datos usando Ado Entity.

Se tiene en cuenta lo siguiente:

- Agregar un elemento de tipo Ado Entity al proyecto y agregar las tablas equipo, tipo y estado.
- Autogenerar un nuevo código de equipo y que mediante el botón nuevo se muestre en el formulario.
- Llenar los cuadros combinados con los tipos y los estados de los equipos.

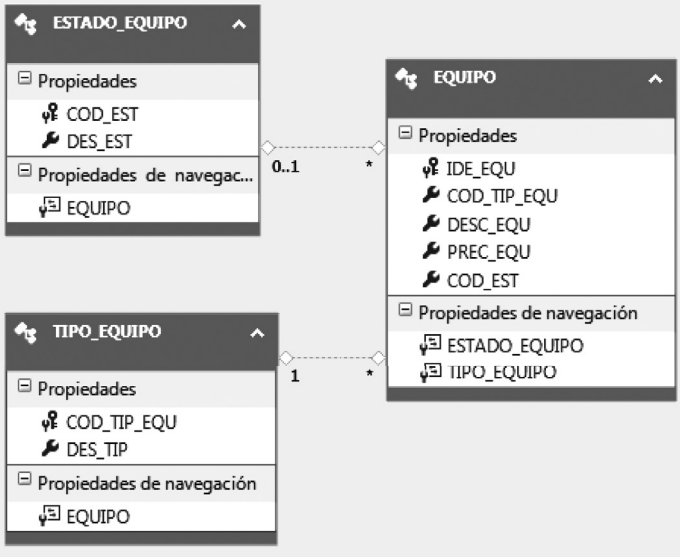
- Considerar que los botones deben estar habilitados dependiendo del proceso a desarrollar.
- Listar los datos de los equipos en un control DataGridView.

GUI Propuesta:



Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjEntity03.
2. Agregue un nuevo elemento al proyecto, seleccione la categoría Datos, luego ADO.NET Entity Data Model y asigne el nombre modContrato.
3. Agregue las tablas equipo, estado y tipo, como se muestra en la siguiente imagen:



Si necesita agregar una tabla, presione clic derecho en el diseñador y seleccione **Actualizar modelo desde base de datos...** Luego, seleccione una tabla para agregarla al diseñador.

4. Verifique el código de conexión en el archivo app.config.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit
    http://go.microsoft.com/fwlink/?LinkID=237468 -->
    <section name="entityFramework" type="System.Data.Entity.Internal.
    ConfigFile.EntityFrameworkSection, EntityFramework, Version=5.0.0.0,
    Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <connectionStrings>
    <add name="CONTRATOEntities" connectionString="metadata=res://*/
     modContrato.csdl|res://*/modContrato.ssdl|res://*/modContrato.
     msl;provider=System.Data.SqlClient;provider connection
     string=&quot;data source=.;initial catalog=CONTRATO;integrated
     security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
     providerName="System.Data.EntityClient" />
  </connectionStrings>
  <entityFramework>
    <defaultConnectionFactory type="System.Data.Entity.Infrastructure.
    LocalDbConnectionFactory, EntityFramework">
      <parameters>
        <parameter value="v11.0" />
      </parameters>
    </defaultConnectionFactory>
  </entityFramework>
</configuration>
```

5. Agregue el siguiente código en el formulario FrmMantenimientodeEquipo:

```
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 pjEntity03
{
    public partial class FrmMantenimientodeEquipo : Form
    {
```

```
CONTRATOEntities db = new CONTRATOEntities();
public FrmMantenimientodeEquipo()
{
    InitializeComponent();
}

private void FrmMantenimientodeEquipo_Load(object sender, EventArgs e)
{
    llenarTipo();
    llenarEstado();
    llenarEquipos();

    //Visibilidad de los botones
    tsGrabar.Visible = false;
    tsActualizar.Enabled = false;
    tsElimina.Enabled = false;
}
void llenarTipo()
{
    cboTipo.DataSource = db.TIPO_EQUIPO.ToList();
    cboTipo.DisplayMember = "des_tip";
    cboTipo.ValueMember = "cod_tip_equ";

}
void llenarEstado()
{
    cboEstado.DataSource = db.ESTADO_EQUIPO.ToList();
    cboEstado.DisplayMember = "des_est";
    cboEstado.ValueMember = "cod_est";
}


void llenarEquipos()
{
    dgContrato.DataSource = db.EQUIPO.ToList();
}

private void tsNuevo_Click(object sender, EventArgs e)
{
    lblCodigo.Text = generaCodigo();
    cboTipo.Focus();

    //visibilidad de los botones
    tsNuevo.Visible = false;
    tsGrabar.Visible = true;
    tsActualizar.Enabled= false;
    tsElimina.Enabled = false;
}

private void tsGrabar_Click(object sender, EventArgs e)
{
    //Enviando los datos del equipo
    EQUIPO regEquipo = new EQUIPO();
```

```
        regEquipo.IDE_EQU = lblCodigo.Text;
        regEquipo.COD_TIP_EQU = cboTipo.SelectedValue.ToString();
        regEquipo.DESC_EQU = txtDescripcion.Text;
        regEquipo.PREC_EQU = int.Parse(txtPrecio.Text);
        regEquipo.COD_EST = cboEstado.SelectedValue.ToString();

        //Agregando los datos a la tabla
        db.EQUIPO.Add(regEquipo);
        db.SaveChanges();

        llenarEquipos();

        //Visibilidad de los botones
        tsGrabar.Visible = false;
        tsNuevo.Visible = true;
        tsActualizar.Visible = true;
        tsElimina.Visible = true;
    }

    //capturar el código del equipo
    string generaCodigo()
    {
        var misEquipos = (from c in db.EQUIPO
                          orderby c.IDE_EQU descending
                          select new { c.IDE_EQU}).Take(1);
        string codigo = "";
        foreach (var equ in misEquipos.ToList())
            codigo = equ.IDE_EQU;

     return "EQ"+(int.Parse(codigo.Substring(2, 4)) + 1).ToString("0000");
    }

  private void tsActualizar_Click(object sender, EventArgs e)
  {
     //Buscar el equipo dentro de la tabla
      EQUIPO regEquipo = db.EQUIPO.Find(lblCodigo.Text);

      //Capturar las actualizaciones
      regEquipo.COD_TIP_EQU = cboTipo.SelectedValue.ToString();
      regEquipo.DESC_EQU = txtDescripcion.Text;
      regEquipo.PREC_EQU = int.Parse(txtPrecio.Text);
      regEquipo.COD_EST = cboEstado.SelectedValue.ToString();

      //Actualizando la informacion
      db.SaveChanges();
      llenarTipo();

      //Visibilidad de botones
      tsNuevo.Visible = true;
      tsGrabar.Visible = false;
      tsActualizar.Enabled = false;
      tsElimina.Enabled = false;
```

```
            llenarEquipos();
        }

        private void dgContrato_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            //Enviando la informacion a los controles
            lblCodigo.Text=dgContrato.CurrentRow.Cells[0].Value.ToString();
            cboTipo.SelectedValue = dgContrato.CurrentRow.Cells[1].Value
                                                        .ToString();
          txtDescripcion.Text = dgContrato.CurrentRow.Cells[2].Value.ToString();
           txtPrecio.Text = double.Parse(dgContrato.CurrentRow.Cells[3].Value
                                                  .ToString()).ToString("0");
            cboEstado.SelectedValue = dgContrato.CurrentRow
                                                   .Cells[4].Value.ToString();

            //Visibilidad de los botones
            tsNuevo.Visible = false;
            tsElimina.Enabled = true;
            tsActualizar.Enabled = true;
        }

        private void tsElimina_Click(object sender, EventArgs e)
        {
            try
            {
                //Buscando el equipo dentro de la tabla
                EQUIPO regEquipo = db.EQUIPO.Find(lblCodigo.Text);
                db.EQUIPO.Remove(regEquipo);
                db.SaveChanges();

                llenarEquipos();

                  //Visibilidad de botones
                  tsNuevo.Visible = true;
                  tsGrabar.Visible = false;
                  tsActualizar.Enabled = false;
                  tsElimina.Enabled = false;
              }
              catch (Exception)
            {
               MessageBox.Show("ocurrió un error al eliminar el equipo..!!!");
            }
        }

        private void tsSalir_Click(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

7

Capítulo

Reportes

Capacidad terminal:

Implementar aplicaciones que permitan mostrar información en forma de reporte usando Crystal Report.

Contenido

- Reportes con Crystal
- Instalación de Crystal Report para Visual Studio 2015
- Diseño de un informe con Crystal Report
- Secciones del reporte
- Modificar el diseño del reporte

  Caso desarrollado 1: Crystal Básico - Reporte de contratos
  Caso desarrollado 2: Crystal Report - Reporte de contratos por cliente
  Caso desarrollado 3: Crystal Report - Reporte de contratos por rango de años

EDITORIAL MACRO®

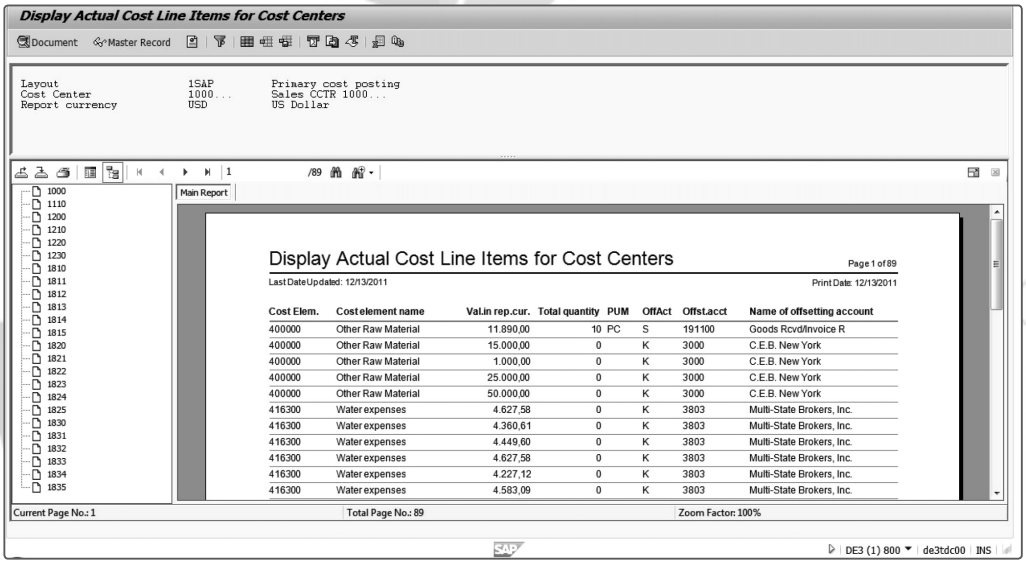
EDITORIAL
MACRO®

7.1 Reportes con Crystal

Crystal Reports para Visual Studio es la versión del reportador más usado en el desarrollo de aplicaciones. Visual Studio proporciona un conjunto de herramientas que permiten conectarse a una base de datos y obtener datos a partir de sus tablas. Crystal Report se encuentra disponible, como parte de la instalación, en la mayoría de las versiones de Microsoft Visual Studio.

Con Crystal Report se puede crear desde informes simples, agrupados e inclusive se puede incluir gráficos estadísticos. Esto mejora el diseño de las aplicaciones vistas en Visual Studio sin salir del mismo, es decir, la implementación de informes con Crystal se desarrollará dentro del marco de Visual Studio.

Cuando se tiene una aplicación en Visual Studio, se agrega un informe Crystal y gracias al asistente se puede diseñar el informe, que luego podrá ser visualizado en Windows Form con el control Crystal Report Viewer. Para implementar un informe, se debe tener un contenedor DataSet con las tablas o los procedimientos almacenados que se usará en el informe.



Fuente: <http://trustedbi.com/images/blog/CrystalReports/alv_run5.png>

7.2 Instalación de Crystal Report para Visual Studio 2015

Cada versión de Visual Studio tiene asociado un archivo de instalación de Crystal Report. Para este caso, se usa la versión Visual Studio 2015 y se busca la versión Crystal Report correspondiente.

URL de descarga: http://scn.sap.com/docs/DOC-7824

Pantalla de descarga:

Support Packs, Fixed Issues and Distribution File downloads

Fixes for each Support Pack are prioritized and released on or about end of each yearly quarter. All support packs are full builds of Crystal Reports for Visual Studio 2010/2012, thus it is not necessary to update incrementally. The most recent Support Pack in the below table is listed first.

To keep current and up to date with new releases as well as KBAs and more follow us on **Twitter**

Update: All Service Packs are cumulative so we are removing the links to previous patches and will be keeping 3 or 4 still active. Links still work if you have them. For installing and starting at the RTM build and progressing to the latest SP is not required.

Please note: To integrate "SAP Crystal Reports, developer version for Microsoft Visual Studio" into VS 2010 or 2012 (SP 7 and higher) or VS 2013 (SP 9 or higher) or VS 2015 RC (SP14) - VS 2015 (fully - SP 15), **you must run the Install Executable**. Running the MSI will not fully integrate Crystal Reports into VS. **MSI files by definition are for runtime distribution only**.

By default Windows 10 does not install the 3.5 framework, CR for VS still needs it. Select it by "Turn Windows feature on or off" and choose both options.

Note 2: SAP Crystal Reports, Developer Version for Visual Studio .NET **does NOT support Express Editions** of any version of Visual Studio .NET. VS.

Install Executable	Fixed Issues	MSI 32 Bit	MSI 64 Bit	MSM 32 Bit	ClickOnce 32/64	ClickOnce "Homesite"	WEB XML Deployment
Support Pack 15 (v.13.0.15.1840	SP15 Fixed Issues	32bit.msi	64bit.msi	13_0_15.msn	Clickonce32/64	clickonce32 clickonce64	crdbxml15.msi

Seleccionar **Support Pack 15** para iniciar la descarga de la versión adecuada para Visual Studio 2015. Luego de esto, ejecutar el archivo descargado y eso será todo. Después, entre al Visual Studio y se podrá crear aplicaciones usando Crystal Report.

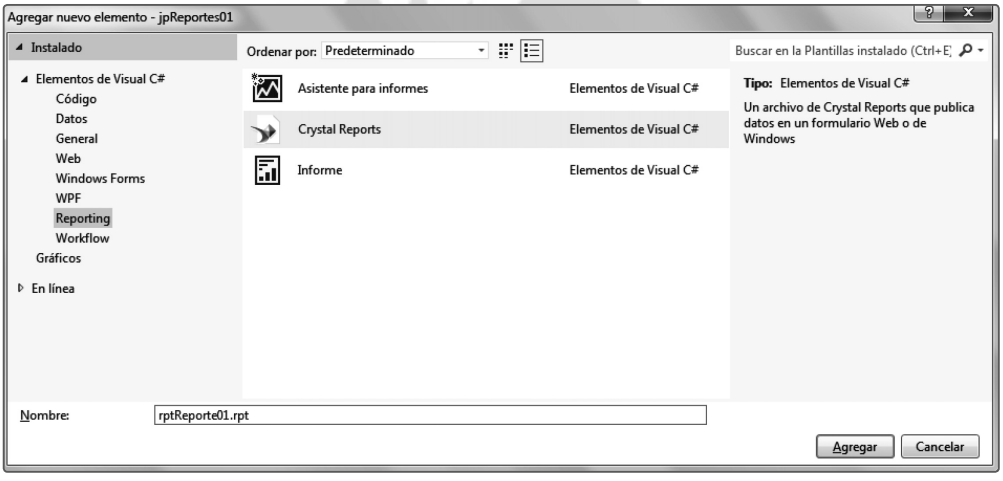
EDITORIAL MARCOMBO®

7.3 Diseño de un informe con Crystal Report

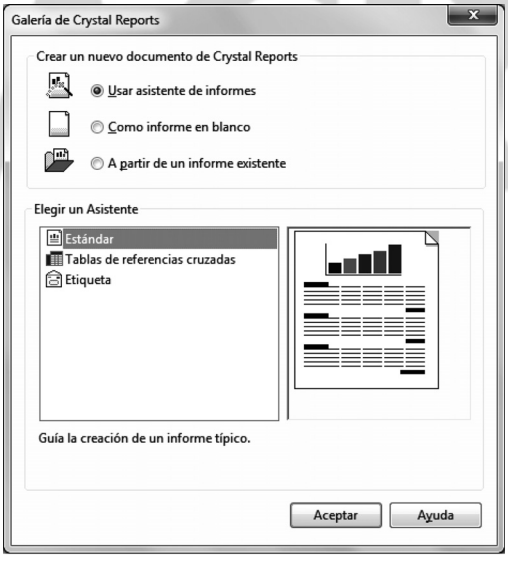
Los informes se crean en el diseñador de Crystal Report. El diseñador de Crystal Report se inicia automáticamente al añadir un objeto de Crystal Reports al proyecto o al hacer doble clic en un objeto de Crystal Reports existente en el proyecto.

Pasos:

1. Haga clic derecho sobre el proyecto y seleccione **Agregar** > **Nuevo elemento.**
2. Seleccione **Ficha Reporting** y luego **Crystal Reports**.

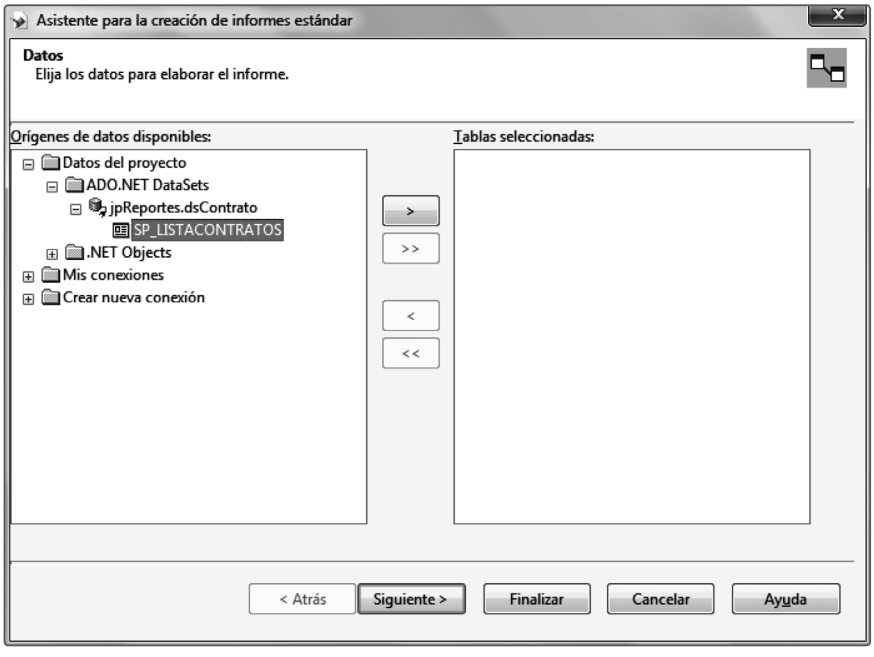


3. Asigne un nombre al reporte y presione el botón **Agregar**.
4. La pantalla inicial del asistente muestra cómo creará el reporte, para lo cual debe seleccionar **Usar asistente de informes**.



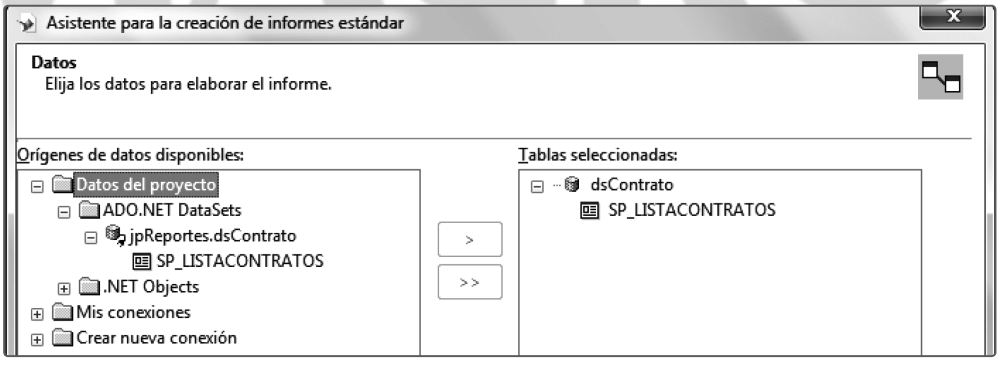
EDITORIAL MACRO®

5. Luego, debe seleccionar de dónde provienen los datos, para lo cual previamente ha debido agregar un elemento DataSet, que contiene el procedimiento almacenado que lista registros de una o más tablas.



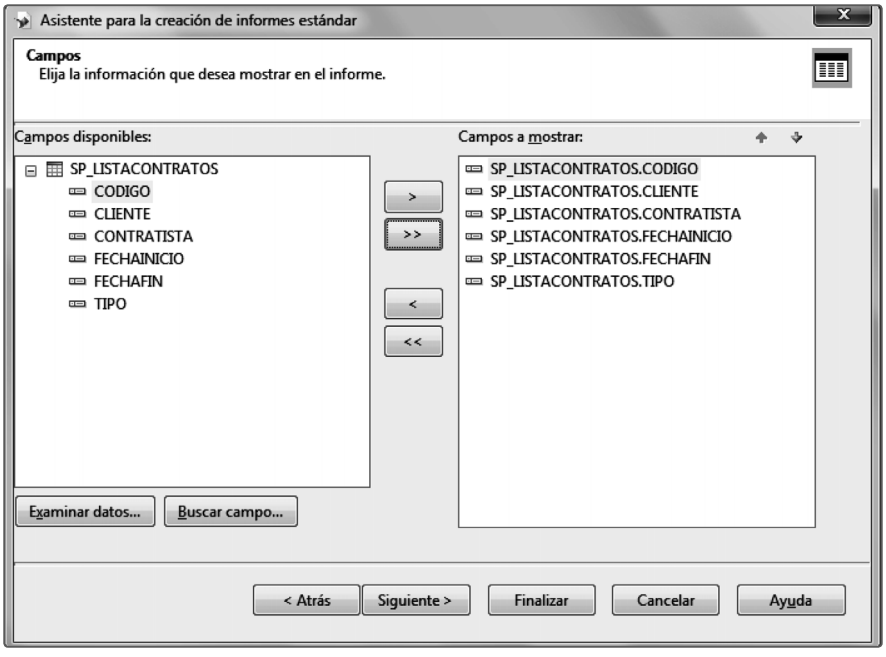
La imagen muestra que dentro de **Datos del proyecto** > **ADO.NET DataSets** se encuentra el elemento DataSet agregado al proyecto. Desde aquí podrá visualizar todas las tablas o los procedimientos agregados al DataSet, el cual proporciona información para el reporte.

Tiene que pasar los datos disponibles a las tablas seleccionadas, es decir, seleccione la tabla o el procedimiento almacenado y con el botón. Luego, debe seleccionar una tabla para el informe, de tal forma que la ventana anterior terminará de la siguiente manera:

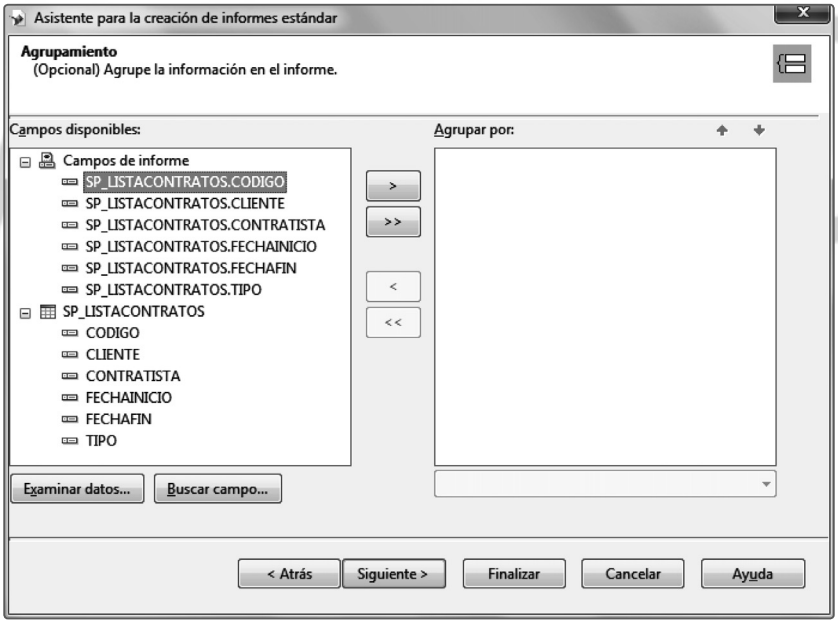


EDITORIAL MACRO®

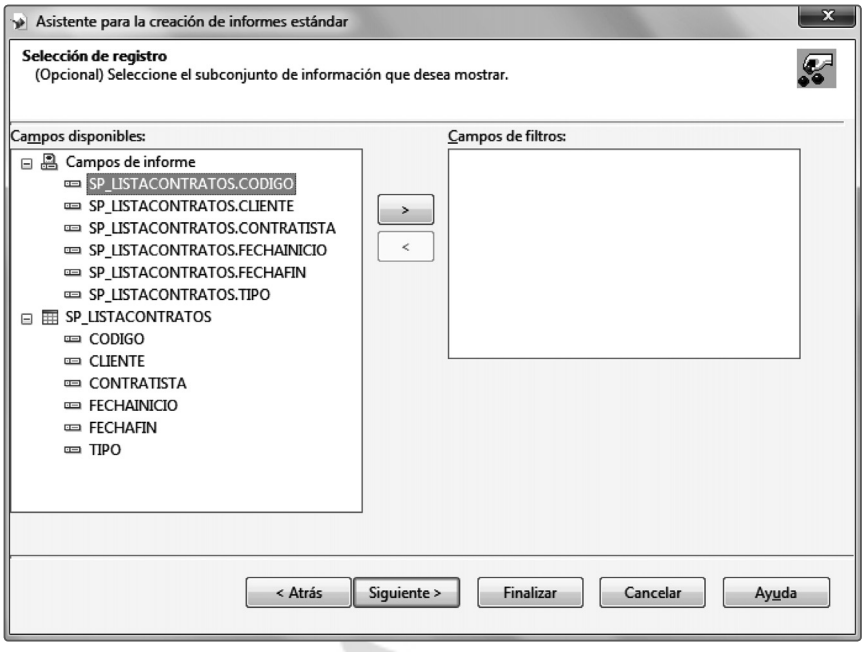
6. En la siguiente ventana, debe seleccionar los campos que componen el reporte. Para lo cual, seleccione los campos disponibles y envíelos a los campos a mostrar con el botón >. Para enviar todos los campos, use el botón >>.



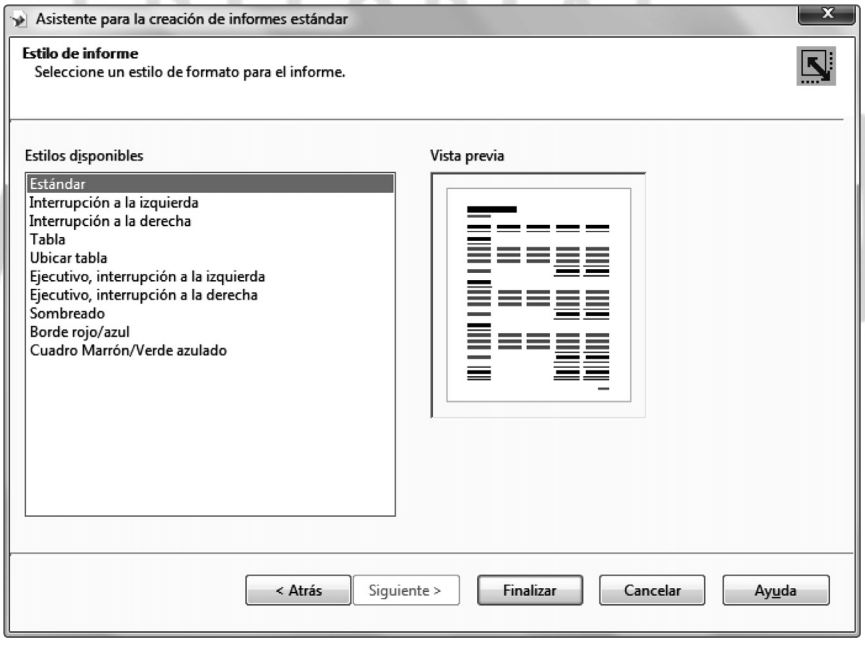
7. En la ventana de agrupamiento, debe seleccionar algún campo que permite generar un nivel de agrupamiento sobre las demás columnas. Para esto, debe pasar uno o más campos desde **Campos disponibles** a **Agrupar por**.



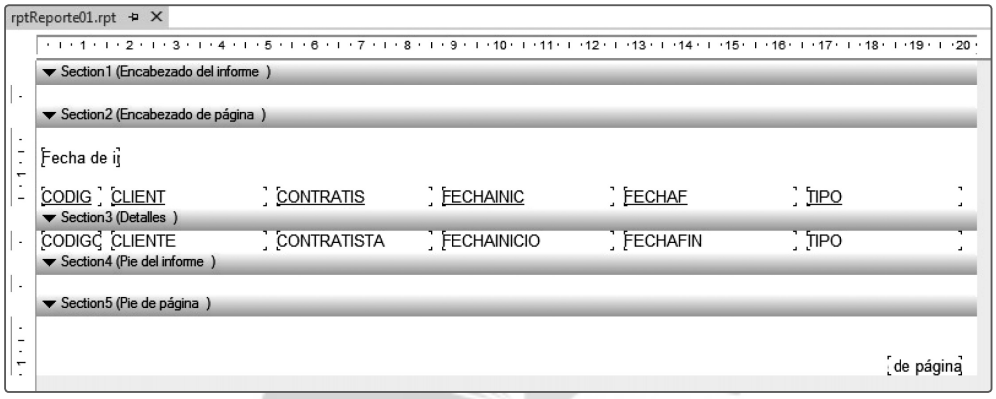
8. En la ventana de selección de registro es opcional la selección de los campos, ya que dependerá del desarrollador crear un subinforme dentro del reporte.



9. Finalmente, debe seleccionar un estilo de informe, el cual permitirá mostrar los datos del reporte en una plantilla predeterminada.



Inicialmente, el reporte se muestra de la siguiente forma:



Como se observa, el reporte se presentará de acuerdo al estilo seleccionado y es a partir de esto que se puede realizar algunos cambios, como agregar un texto, imagen o cambiar la fuente de los textos.

7.4 Secciones del reporte

El diseño de Crystal Report está dividido en secciones, las cuales se deben considerar al mejorar la plantilla inicial del reporte. Se cuenta con encabezados, pies y detalle. Cada sección tiene características especiales que se conocerá a continuación:

a. **Encabezado de informe:** Todos los objetos colocados en esta sección solo se podrán visualizar al inicio del reporte. Esta puede contener el título del informe o alguna información adicional, que desea que aparezca en esta sección.

   Se tiene en cuenta lo siguiente:

   - Los gráficos estadísticos y las tablas con referencia cruzada contienen los datos de todo el reporte.
   - Las fórmulas incorporadas solo se evalúan una vez, ya que esta sección solo se presenta una vez.

b. **Encabezado de página:** Todos los objetos colocados en esta sección se podrán visualizar en cada página, que compone el reporte. Debe tener en cuenta que la cantidad de páginas se genera de acuerdo a la información a mostrar.

   Esta sección puede contener campos como textos para títulos o encabezados de las columnas que se mostrarán en el detalle del reporte.

   Se tiene en cuenta lo siguiente:

   - No se puede incluir gráficos estadísticos ni tablas cruzadas.
   - Se pueden colocar fórmulas, pero se debe tener en cuenta que solo se evalúan una vez por página.

EDITORIAL MACRO®

c. **Detalle:** Todos los objetos colocados en esta sección se imprimen con cada registro obtenido desde la tabla de origen. Normalmente, se especifican las columnas que se desean mostrar en el reporte.

Se tiene en cuenta lo siguiente:

- No se puede incluir gráficos estadísticos ni tablas cruzadas.
- Se pueden colocar fórmulas, pero debe tener en cuenta que estas se evalúan por cada registro mostrado en esta sección.

d. **Pie de informe**: Todos los objetos colocados en esta sección se imprimen solo una vez al final del reporte. Normalmente, se usa para imprimir totales.

Se tiene en cuenta lo siguiente:

- Se puede incorporar gráficos estadísticos y tablas cruzadas.
- Se puede incorporar fórmulas, pero se debe tener en cuenta que solo se evalúan una vez.

e. **Pie de página:** Todos los objetos colocados en esta sección se imprimen en la parte inferior de cada página. Normalmente, esta sección refleja el número de página, así como el derecho de autor u otros elementos que se desee incorporar.

Se tiene en cuenta lo siguiente:

- No se puede incluir gráficos estadísticos ni tablas cruzadas.
- Se puede incorporar fórmulas, pero se debe tener en cuenta que solo se ejecutan por cada página del reporte.

7.5 Modificar el diseño del reporte

Observe algunas opciones para mejorar el reporte inicial, como cambiar el tipo de letra o agregar imágenes.

a. **Agregar un texto al reporte.**

1. Clic derecho sobre cualquier sección del reporte.
2. Seleccione **Insertar** > **Objeto de texto**.
3. Presione clic en la sección y coloque el texto.

b. **Modificar la fuente a los textos.**

1. Seleccione los marcos de los textos a modificar la fuente.
2. Presione clic derecho sobre uno de los elementos.
3. Seleccione **Dar formato a varios objetos**.

EDITORIAL MACRO®

c. Agregar una imagen.

1. Clic derecho sobre cualquier sección del reporte.
2. Seleccione insertar > **Imagen...**
3. Seleccione una imagen desde la unidad de disco.
4. Finalmente, presione un clic sobre la sección del reporte.

d. Agregar campos especiales.

1. Clic derecho sobre cualquier sección del reporte.
2. Seleccione insertar > **Campo especial.**
3. Seleccione alguna opción especial como la fecha, la hora y el número de página.

Caso desarrollado 1 : Crystal Básico - Reporte de contratos

Implementar una aplicación que permita mostrar los registros de los contratos en un reporte usando Crystal Report.

Implementar un procedimiento almacenado que permita mostrar los datos de los contratos, que incluya código, nombre completo del cliente, nombre completo del contratista, fecha de inicio, fecha de finalización y descripción del tipo.

GUI Propuesta:

COD	CLIENTE	CONTRATISTA	FECHAINICIO	FECHAFIN	TIPO
ABC001	MARIA PEREZ TORRES	PEDRO LOPEZ SANCHEZ	20/05/2006 12:0	01/05/2010 12:!	PRESTAMO
ABC002	JUNIOR FERNANDEZ LOPEZ	LUIGUI JARAMILLO ORTIZ	21/06/2005 12:0	02/05/2010 12:!	PRESTAMO
ABC003	ALEX CARDENAS ZARRASCO	OSCAR ZAMORA ROSAS	22/07/2005 12:0	03/01/2010 12:!	PRESTAMO
ABC004	JUAN MEZQUITA ROSAS	DICK GALVEZ CARBAJAL	23/09/2005 12:0	04/02/2009 12:!	PRESTAMO
ABC005	ERICK GONZALES SHULLER	RICHARD VELEZMORO SO	24/05/2004 12:0	12/03/2009 12:!	PRESTAMO
ABC006	JOSE GALVEZ RETAMOZO	NELSON MEJIA MEJIA	25/04/2003 12:0	13/02/2009 12:!	PRESTAMO
ABC007	CARLOS JARA SALAS	TOMAS GONZALES SANCHEZ	26/03/2003 12:0	21/10/2009 12:!	PRESTAMO
ABC008	EDWARD RAMIREZ ROJAS	CRISTHIAN HUARAC TORRES	27/02/2006 12:0	25/11/2009 12:!	PRESTAMO
ABC009	MANUEL GUIZADO GAMARRA	JORGE CASIMIRO SOTO	28/01/2007 12:0	25/12/2010 12:!	PRESTAMO

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjReportes.
2. Implemente el siguiente procedimiento almacenado en Sql Server:

```
--SP_LISTACONTRATOS
IF OBJECT_ID("SP_LISTACONTRATOS")IS NOT NULL
       DROP PROC SP_LISTACONTRATOS
GO
CREATE PROC SP_LISTACONTRATOS
AS
       SELECT CA.COD_CON AS CODIGO,
              C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE,
              CO.NOM_CON+SPACE(1)+CO.PAT_CON+SPACE(1)+CO.MAT_CON AS CONTRATISTA,
              CA.FIN_CON AS FECHAINICIO,
              CA.FFI_CON AS FECHAFIN,
              CA.TIP_CON AS TIPO
              FROM CONTRATOALQUILER CA
              JOIN CLIENTE C ON CA.IDE_CLI=C.IDE_CLI
              JOIN CONTRATISTA CO ON CA.IDE_CON=CO.IDE_CON
GO
```

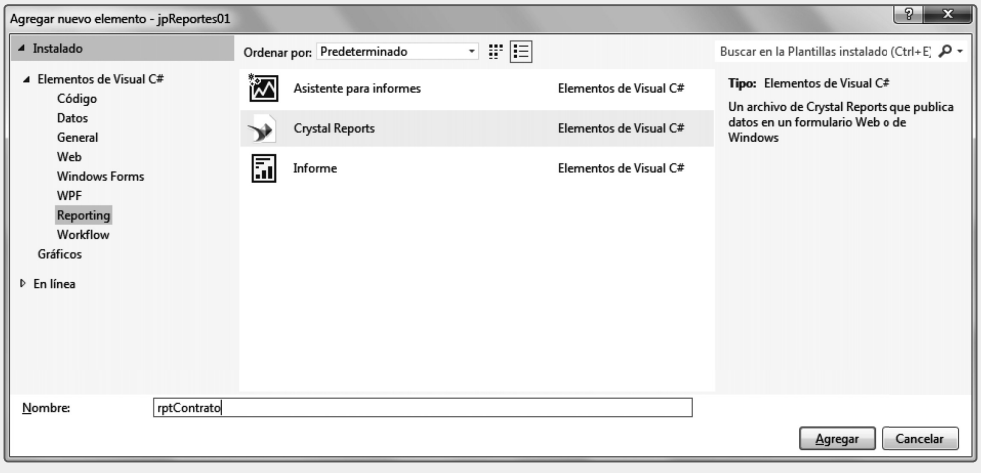
3. Agregue el elemento Dataset al proyecto y llámelo dsContrato. Luego, agregue el procedimiento almacenado SP_LISTACONTRATOS, como se muestra en la siguiente imagen:

4. Configure la cadena de conexión en el archivo app.config.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="jpReportes.Properties.Settings.CONTRATOConnectionString"
      connectionString="Data Source=.;Initial Catalog=CONTRATO;Integrated
      Security=True"
        providerName="System.Data.SqlClient" />
  </connectionStrings>
  <startup  useLegacyV2RuntimeActivationPolicy="true">
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
</configuration>
```

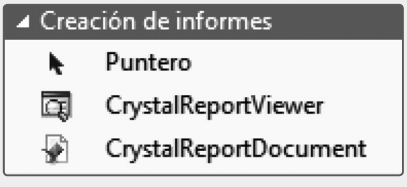
Se debe tener en cuenta que la cadena de conexión se ha configurado automáticamente por agregar un DataSet al proyecto, solo se debe agregar la instrucción «`useLegacyV2RuntimeActivationPolicy="true»` para ejecutar correctamente el reporte.

5. Agregue un nuevo elemento al proyecto del tipo Crystal Reports, ubicado en la sección Reporting y asigne el nombre rptContrato, como se muestra en la siguiente imagen:



6. Luego, diseñe el siguiente modelo de reporte desde el diseñador de reportes de Crystal Report:

7. Agregue el elemento Crystal Report Viewer desde el panel de la barra de herramientas, sección **Creación de informes** y asigne el nombre crvContrato.



8. Agregue la clase Conexión y coloque el siguiente código:

```
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 jpReportes
{
```

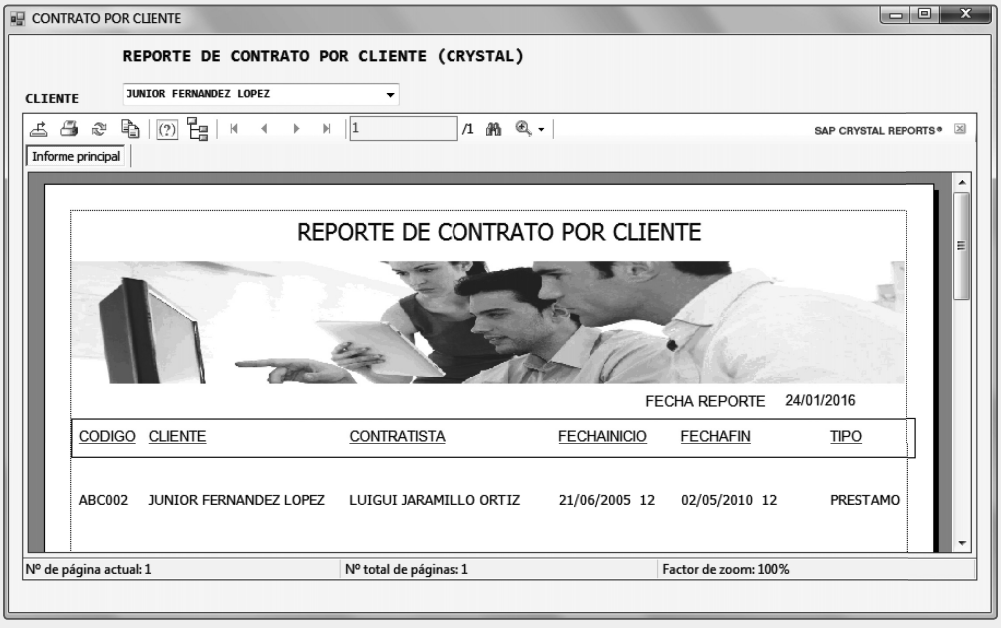
```
    public partial class frmContrato : Form
    {
      public frmContrato()
      {
            InitializeComponent();
      }

      private void frmContrato_Load(object sender, EventArgs e)
      {
            mostrarReporte();
      }

      //Metodo que visualiza el reporte
      void mostrarReporte()
      {
        //Creando un objeto del dsContrato
        dsContrato ds = new dsContrato();

       //Creando un objeto del objeto que contiene el procedimiento almacenado
        dsContratoTableAdapters.SP_LISTACONTRATOSTableAdapter da =
                  new dsContratoTableAdapters.SP_LISTACONTRATOSTableAdapter();

        //Poblando el DataSet
        da.Fill(ds.SP_LISTACONTRATOS);

        //Creando u objeto del reporte
        rptContrato reporte = new rptContrato();

        //Asignando datos al reporte
        reporte.SetDataSource(ds);

        //Enviando la informacion el visor de reporte
        crvContrato.ReportSource = reporte;
      }
    }
}
```

Caso desarrollado 2 : Crystal Report - Reporte de contratos por cliente

Implementar una aplicación que permita mostrar los registros de los contratos según un determinado cliente en un reporte usando Crystal Report.

Implementar procedimientos almacenados en SQL Server que permita mostrar los datos, los contratos según el cliente.

GUI Propuesta:



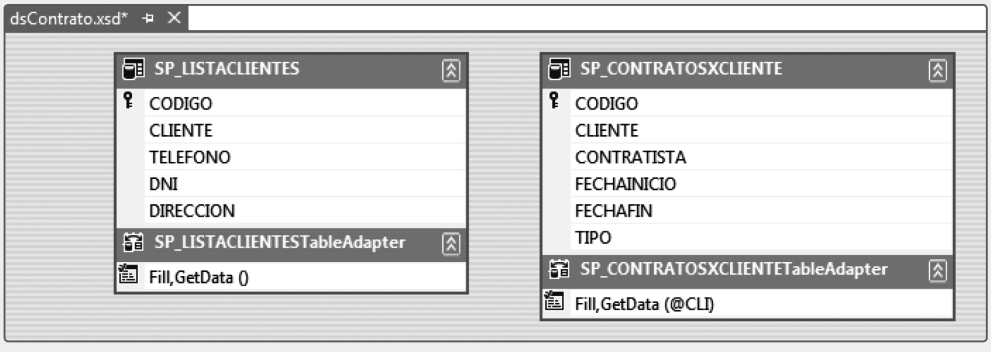
Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjReporte02.
2. Implemente los siguientes procedimientos almacenados en Sql Server:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS CLIENTES
IF OBJECT_ID("SP_LISTACLIENTES")IS NOT NULL
       DROP PROC SP_LISTACLIENTES
GO
CREATE PROC SP_LISTACLIENTES
AS
       SELECT C.IDE_CLI AS CODIGO,
              C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE,
              C.FON_CLI AS TELEFONO,
              C.DNI_CLI AS DNI,
              C.DIR_CLI AS DIRECCION
              FROM CLIENTE C
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS CONTRATOS POR CLIENTE
IF OBJECT_ID("SP_CONTRATOSXCLIENTE")IS NOT NULL
       DROP PROC SP_CONTRATOSXCLIENTE
GO
CREATE PROC SP_CONTRATOSXCLIENTE(@CLI CHAR(6))
AS
```

```
 SELECT CA.COD_CON AS CODIGO,
        C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE,
        CO.NOM_CON+SPACE(1)+CO.PAT_CON+SPACE(1)+CO.MAT_CON AS CONTRATISTA,
        CA.FIN_CON AS FECHAINICIO,
        CA.FFI_CON AS FECHAFIN,
        CA.TIP_CON AS TIPO
        FROM CONTRATOALQUILER CA
        JOIN CLIENTE C ON CA.IDE_CLI=C.IDE_CLI
        JOIN CONTRATISTA CO ON CA.IDE_CON=CO.IDE_CON
        WHERE CA.IDE_CLI=@CLI
GO
```

3. Agregue el elemento Dataset al proyecto y llámelo dsContrato. Agregue los procedimientos almacenados SP_LISTACLIENTES y SP_CONTRATOSCLIENTE, como se muestra en la siguiente imagen:

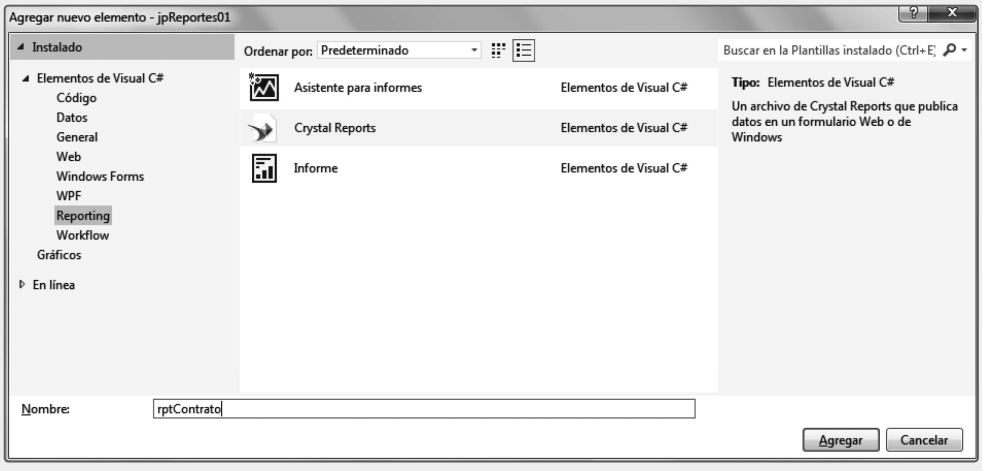


4. Configure la cadena de conexión en el archivo app.config.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
       <add name="pjReporte02.Properties.Settings.CONTRATOConnectionString"
      connectionString="Data Source=.;Initial Catalog=CONTRATO;Integrated
      Security=True"
           providerName="System.Data.SqlClient" />
    </connectionStrings>

    <!--Activando el reporte en la aplicacion -->
    <startup useLegacyV2RuntimeActivationPolicy="true">
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

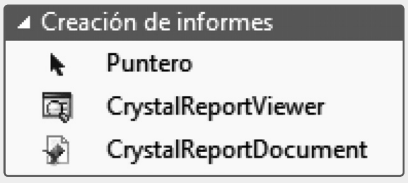
Se debe tener en cuenta que la cadena de conexión se ha configurado automáticamente por agregar un DataSet al proyecto, solo se debe agregar la instrucción `"useLegacy-V2RuntimeActivationPolicy="true"` para ejecutar correctamente el reporte.

5. Agregue un nuevo elemento al proyecto del tipo Crystal Reports, ubicado en la sección **Reporting**, y asigne el nombre rptContrato, como se muestra en la siguiente imagen:



6. Luego, diseñe el siguiente modelo de reporte desde el diseñador de reportes de Crystal Report:

7. Agregue el elemento Crystal Report Viewer desde el panel de la barra de herramientas sección **Creación de informes** y asigne el nombre crvContrato.



8. Agregue la clase Conexión y coloque el siguiente código:

```
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 pjReporte02
{
    public partial class FrmContratoxCliente : Form
    {
        //Objeto del esquema DataSet
        dsContrato ds = new dsContrato();

        public FrmContratoxCliente()
        {
            InitializeComponent();
        }

        private void FrmContratoxCliente_Load(object sender, EventArgs e)
        {
            //Objeto de la tabla ListaClientes para llenar el comboBox
            dsContratoTableAdapters.SP_LISTACLIENTESTableAdapter da =
                new dsContratoTableAdapters.SP_LISTACLIENTESTableAdapter();

            //Llenando el combobox
            da.Fill(ds.SP_LISTACLIENTES);
            cboClientes.DataSource = ds.SP_LISTACLIENTES;
            cboClientes.DisplayMember = "CLIENTE";
            cboClientes.ValueMember = "CODIGO";
        }
```

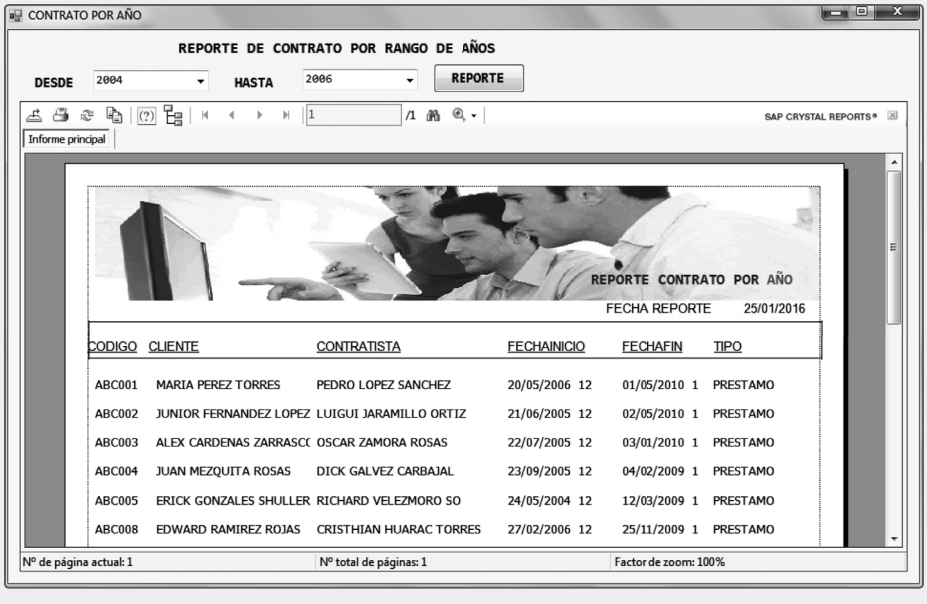
```
    private void cboClientes_SelectedIndexChanged(object sender, EventArgs e)
     {
       //Objeto de la tabla contrato por cliente
       dsContratoTableAdapters.SP_CONTRATOSXCLIENTETableAdapter da =
             new dsContratoTableAdapters.SP_CONTRATOSXCLIENTETableAdapter();
      da.Fill(ds.SP_CONTRATOSXCLIENTE,cboClientes.SelectedValue.ToString());

       //Llevando la informacion al reporte
       rptContrato reporte = new rptContrato();
       reporte.SetDataSource(ds);

       //Mostrando el informe
       crvContrato.ReportSource = reporte;
     }
    }
}
```

Caso desarrollado 3 : Crystal Report - Reporte de contratos por rango de años

Implementar una aplicación que permita mostrar los registros de los contratos según un rango de años en un reporte usando Crystal Report.

Implementar los procedimientos almacenados en SQL Server, que permita listar los años en forma ascendente y descendente, además de mostrar los contratos por un rango de años.

GUI Propuesta:

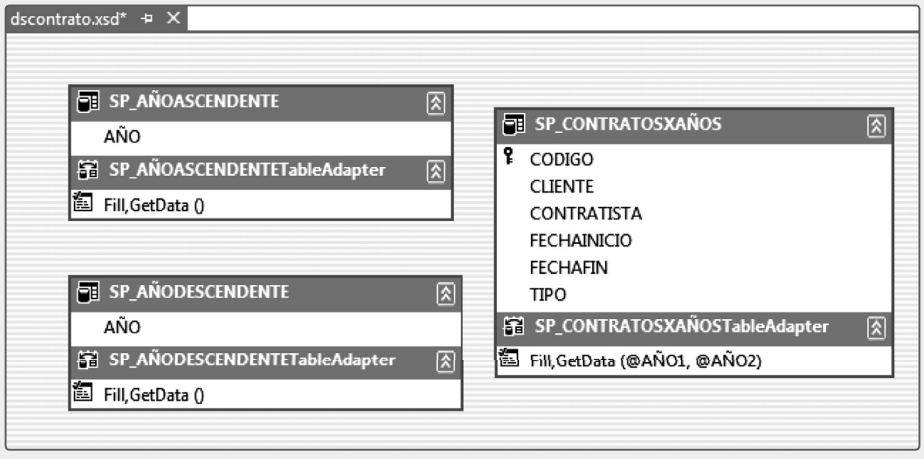


Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjReporte03.
2. Implemente los siguientes procedimientos almacenados en Sql Server:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS AÑOS EN FORMA ASCENDENTE
IF OBJECT_ID("SP_AÑOASCENDENTE")IS NOT NULL
       DROP PROC SP_AÑOASCENDENTE
GO
CREATE PROC SP_AÑOASCENDENTE
AS
       SELECT DISTINCT YEAR(C.FIN_CON) AS AÑO
              FROM CONTRATOALQUILER C
              ORDER BY 1 ASC
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS AÑOS EN FORMA DESCENDENTE
IF OBJECT_ID("SP_AÑODESCENDENTE")IS NOT NULL
       DROP PROC SP_AÑODESCENDENTE
GO
CREATE PROC SP_AÑODESCENDENTE
AS
       SELECT DISTINCT YEAR(C.FIN_CON) AS AÑO
              FROM CONTRATOALQUILER C
              ORDER BY 1 DESC
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS CONTRATOS POR AÑOS
IF OBJECT_ID("SP_CONTRATOSXAÑOS")IS NOT NULL
       DROP PROC SP_CONTRATOSXAÑOS
GO
CREATE PROC SP_CONTRATOSXAÑOS(@AÑO1 INT, @AÑO2 INT)
AS
  SELECT CA.COD_CON AS CODIGO,
       C.NOM_CLI+SPACE(1)+C.PAT_CLI+SPACE(1)+C.MAT_CLI AS CLIENTE,
       CO.NOM_CON+SPACE(1)+CO.PAT_CON+SPACE(1)+CO.MAT_CON AS CONTRATISTA,
       CA.FIN_CON AS FECHAINICIO,
       CA.FFI_CON AS FECHAFIN,
       CA.TIP_CON AS TIPO
       FROM CONTRATOALQUILER CA
       JOIN CLIENTE C ON CA.IDE_CLI=C.IDE_CLI
       JOIN CONTRATISTA CO ON CA.IDE_CON=CO.IDE_CON
       WHERE YEAR(CA.FIN_CON) BETWEEN @AÑO1 AND @AÑO2
GO
--PRUEBA: SP_CONTRATOSXAÑOS 1999,2005
```

3. Agregue el elemento Dataset al proyecto y llámelo xdsContrato. Luego, agregue los procedimientos almacenados SP_AÑODESCENDENTE, SP_AÑOASCENDENTE y SP_CONTRATOSXAÑOS, como se muestra en la siguiente imagen:

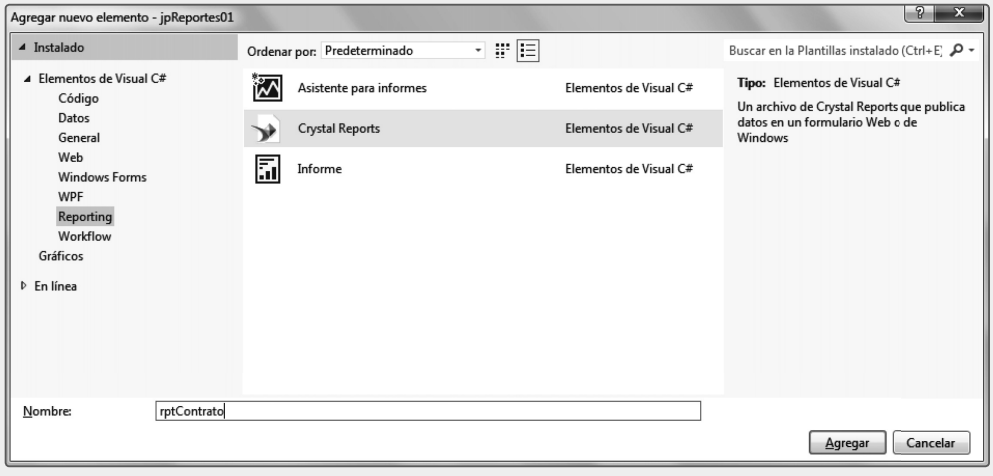


4. Configure la cadena de conexión en el archivo app.config.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
        <add name="jpReporte03.Properties.Settings.CONTRATOConnectionString"
       connectionString="Data Source=.;Initial Catalog=CONTRATO;Integrated
       Security=True"
            providerName="System.Data.SqlClient" />
    </connectionStrings>
    <startup  useLegacyV2RuntimeActivationPolicy="true">
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

Se debe tener en cuenta que la cadena de conexión se ha configurado automáticamente por agregar un DataSet al proyecto, solo se debe agregar la instrucción `"useLegacy-V2RuntimeActivationPolicy="true"` para ejecutar correctamente el reporte.

5. Agregue un nuevo elemento al proyecto del tipo Crystal Reports, ubicado en la sección **Reporting**, y asigne el nombre rptContrato, como se muestra en la siguiente imagen:



6. Luego, diseñe el siguiente modelo de reporte desde el diseñador de reportes de Crystal Report:

7. Agregue el elemento Crystal Report Viewer desde el panel de la barra de herramientas sección **Creación de informes** y asigne el nombre crvContrato.

8. Agregue la clase Conexión y coloque el siguiente código:

```
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 jpReporte03
{
    public partial class Form1 : Form
    {
        //Objeto del esquema DataSet
        dscontrato ds = new dscontrato();

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            //Objeto de la tabla ListaClientes para llenar el comboBox
            dscontratoTableAdapters.SP_AÑOASCENDENTETableAdapter da1 =
              new dscontratoTableAdapters.SP_AÑOASCENDENTETableAdapter();
            dscontratoTableAdapters.SP_AÑODESCENDENTETableAdapter da2 =
               new dscontratoTableAdapters.SP_AÑODESCENDENTETableAdapter();

            //Llenando el combobox
            da1.Fill(ds.SP_AÑOASCENDENTE);

            cboHasta.DataSource = ds.SP_AÑOASCENDENTE;
            cboHasta.DisplayMember = "AÑO";

            da2.Fill(ds.SP_AÑODESCENDENTE);
            cboDesde.DataSource = ds.SP_AÑODESCENDENTE;
```

```
            cboDesde.DisplayMember = "AÑO";
        }

        private void btnReporte_Click(object sender, EventArgs e)
        {

            int desde=int.Parse(cboDesde.Text);
            int hasta=int.Parse(cboHasta.Text);


            //Objeto de la tabla contrato por AÑO
            dscontratoTableAdapters.SP_CONTRATOSXAÑOSTableAdapter da =
               new dscontratoTableAdapters.SP_CONTRATOSXAÑOSTableAdapter();
            da.Fill(ds.SP_CONTRATOSXAÑOS, desde, hasta);

            //Llevando la informacion al reporte
            rptContrato reporte = new rptContrato();
            reporte.SetDataSource(ds);

            //Mostrando el informe
            crvContrato.ReportSource = reporte;
        }
    }
}
```

EDITORIAL MACRO®

EDITORIAL
MACRO®

8

Capítulo

Servicios WCF

Capacidad terminal:

Implementar aplicaciones de consultas y mantenimiento de registros a partir de un consumo de servicio WCF.

Contenido

- WCF
- Características del WCF
- Implementación de un servicio WCF
  Caso desarrollado 1: WCF - Listado de contratistas
  Caso desarrollado 2: WCF - Listado de fichas de devolución por contratista
  Caso desarrollado 3: WCF - Mantenimiento de equipos

EDITORIAL MACRO®

EDITORIAL
MACRO®

8.1 WCF

Windows Communication Foundation (WCF) es un marco de trabajo para la creación de aplicaciones orientadas a servicios. Con WCF, es posible enviar datos como mensajes asincrónicos de un extremo de servicio a otro. Un extremo de servicio puede formar parte de un servicio disponible continuamente hospedado por IIS (Internet Information Server), o puede ser un servicio hospedado en una aplicación. Un extremo puede ser un cliente de un servicio que solicita datos de un extremo de servicio. Los mensajes pueden ser tan simples como un carácter o una palabra enviados como XML, o tan complejos como un flujo de datos binarios. A continuación, se indican unos cuántos escenarios de ejemplo:

- Un servicio seguro para procesar transacciones comerciales.
- Un servicio que proporciona datos actualizados a otras personas, como un informe sobre tráfico u otro servicio de supervisión.
- Un servicio de chat que permite a dos personas comunicarse o intercambiar datos en tiempo real.
- Una aplicación de panel que sondea los datos de uno o varios servicios y los muestra en una presentación lógica.
- Exponer un flujo de trabajo implementado utilizando Windows Workflow Foundation como un servicio WCF.

8.2 Características del WCF

WCF incluye el siguiente conjunto de características:

- **Orientación a servicios**

  Como consecuencia del uso de los estándares de WS, WCF le permite crear aplicaciones orientadas a servicios.SOA. La arquitectura orientada a servicios es el uso de servicios web para enviar y recibir datos. Los servicios tienen la ventaja general de estar débilmente acoplados entre una aplicación y otra en lugar de incluirlos en el código. Una relación de acoplamiento débil implica que cualquier cliente creado en cualquier plataforma puede conectar con cualquier servicio siempre y cuando se cumplan los contratos esenciales.

- **Interoperabilidad**

  WCF implementa los estándares del sector moderno para la interoperabilidad de servicios web.

- **Varios patrones de mensajes**

  Los mensajes se intercambian mediante uno de los distintos patrones. El más común es el de solicitud/respuesta, en que un extremo solicita datos de otro extremo y el otro extremo responde.

EDITORIAL MACRO®

Existen otros patrones, como un mensaje unidireccional, en que un único extremo envía un mensaje sin esperar ninguna respuesta. Un patrón más complejo es el patrón de intercambio dúplex, donde dos extremos establecen una conexión y envían datos hacia delante y hacia atrás, similar a un programa de mensajería instantánea.

- **Metadatos de servicios**

  WCF admite la publicación de metadatos de servicios utilizando los formatos especificados en los estándares de la industria, como WSDL, Esquemas XML y WS-Policy. Estos metadatos pueden usarse para generar y configurar automáticamente clientes para el acceso a los servicios de WCF. Los metadatos se pueden publicar sobre HTTP y HTTPS, o utilizando el estándar intercambio de metadatos de servicios web. Para obtener más información, observe Metadatos.

- **Contratos de datos**

  Dado que WCF se basa en .NET Framework, también incluye métodos con código sencillo para proporcionar los contratos que se desea aplicar. Uno de los tipos de contrato universales es el contrato de datos. Básicamente, mientras se escribe el código del servicio usando Visual C# o Visual Basic, la forma más sencilla de controlar los datos consiste en crear clases que representan una entidad de datos con propiedades que pertenecen a la misma. WCF incluye un completo sistema para trabajar con datos de esta manera fácil. Cuando se han creado las clases que representan los datos, el servicio genera automáticamente los metadatos que permiten a los clientes ajustarse a los tipos de datos que se han diseñado.

- **Seguridad**

  Es posible cifrar los mensajes para proteger la privacidad, así como obligar a los usuarios a que se autentiquen antes de permitirles recibir mensajes. La seguridad puede implementarse utilizando estándares conocidos como SSL o WS-SecureConversation.

- **Varios transportes y codificaciones**

  Los mensajes pueden enviarse con cualquiera de los protocolos y codificaciones integrados. La combinación más frecuente de protocolo y codificación consiste en enviar mensajes SOAP codificados de texto utilizando el protocolo de transferencia de hipertexto (HTTP) usado en World Wide Web. WCF también le permite enviar mensajes sobre TCP, canalizaciones con nombre o MSMQ. Estos mensajes pueden codificarse como texto o utilizando un formato binario optimizado. Los datos binarios pueden enviarse de manera eficaz utilizando el estándar MTOM. Si ninguno de los transportes o las codificaciones proporcionados satisfacen las necesidades, se puede crear uno personalizado.

- **Mensajes confiables y en cola**

  WCF admite intercambio de mensajes confiables usando sesiones confiables implementadas sobre mensajería WS-Reliable y mediante MSMQ.

EDITORIAL MACRO®

- **Mensajes duraderos**

  Un mensaje duradero es aquel que nunca se pierde debido a una interrupción de la comunicación. Los mensajes que forman parte de un patrón de mensajes duraderos siempre se guardan en una base de datos. Si se produce una interrupción, la base de datos le permite reanudar el intercambio de mensajes cuando se restablezca la conexión. También puede crear un mensaje duradero utilizando Windows Workflow Foundation (WF).

- **Transacciones**

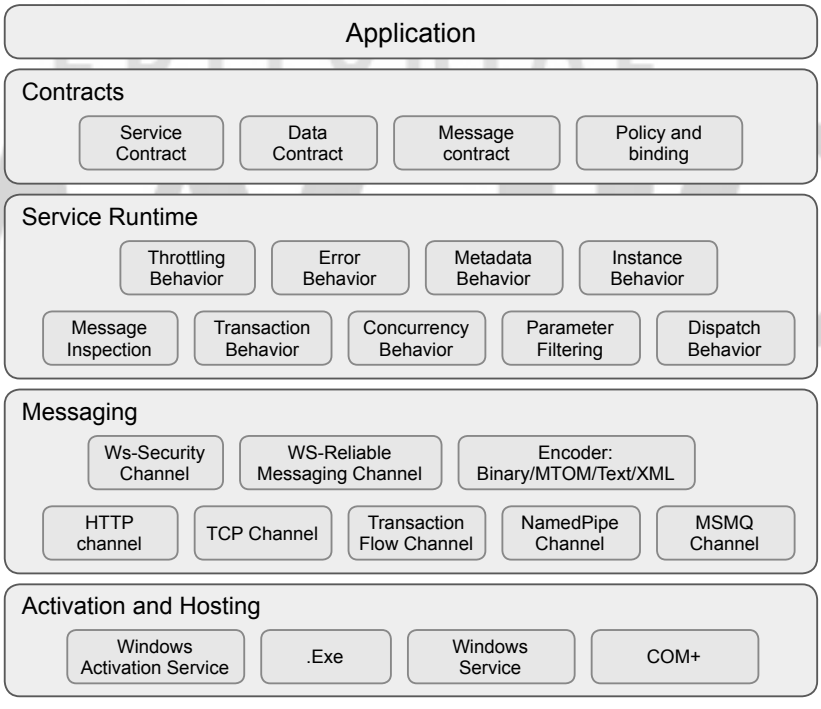
  WCF también admite las transacciones que usan uno de los tres modelos de transacción: las transacciones WS-Atomic, las API del espacio de nombres System.Transactions y Coordinador de transacciones distribuidas de Microsoft.

- **Compatibilidad con AJAX y REST**

  REST es un ejemplo de una tecnología de la Web 2.0 en evolución. WCF se puede configurar para procesar datos XML sin formato, que no se ajustan en un SOAP. WCF también se puede extender para admitir formatos XML concretos, como ATOM (un estándar popular de RSS), e incluso formatos no XML, como notación de objetos JavaScript (JSON).

- **Extensibilidad**

  La arquitectura de WCF tiene varios puntos de extensibilidad. Si se necesita una función adicional, existen una serie de puntos de entrada que le permiten personalizar el comportamiento de un servicio.

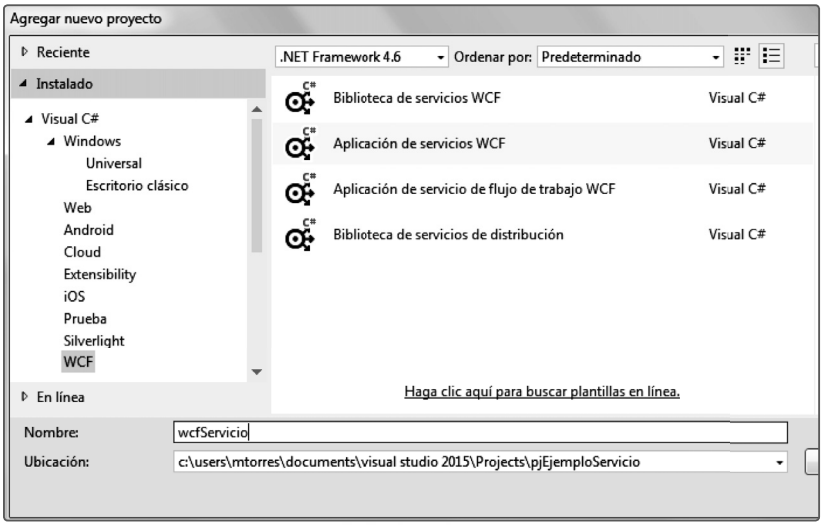


Fuente: <http://wcftutorial.net/Images/050000_WCF-Architecture.jpg>

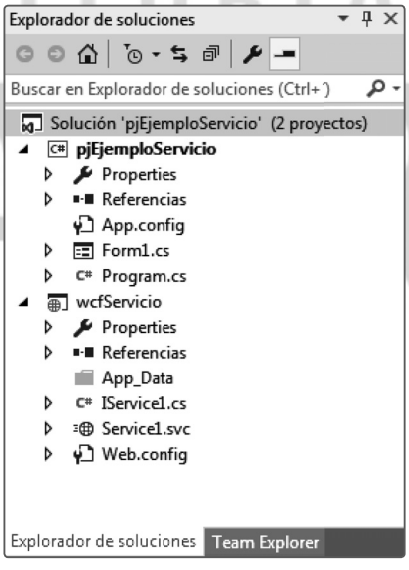
8.3 Implementación de un servicio WCF

Inicialmente, se tiene que crear un proyecto de tipo Windows Forms y luego realizar los siguientes pasos:

1. Haga clic en **Archivo** > **agregar** > **nuevo proyecto**, seleccione la categoría **WCF** > **Aplicación de servicios WCF** y asígnele un nombre al servicio.



Inicialmente, el Explorador de proyecto se mostrará de la siguiente manera:



Como puede notar, el servicio se agrega al proyecto inicial y, es a partir de aquí, que configurará el servicio en los archivos Iservice1 y Service1.

EDITORIAL MARO®

2. Agregue el siguiente código en el archivo Iservice1.cs.

```
//Librería para el uso del DataSet
using System.Data;

//Operación de servicio para el listado
[OperationContract]
DataSet LISTACONTRATISTA();
```

De modo, que el archivo IService1.cs contenga el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;

namespace wcfServicio
{
    // NOTA: puede usar el comando "Rename" del menú "Refactorizar"
    [ServiceContract]
    public interface IService1
    {

        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: agregue aquí sus operaciones de servicio
        [OperationContract]
        DataSet LISTACONTRATISTA();
    }


    // Utilice un contrato de dato
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
```

```
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

3. Seguidamente, agregue el siguiente código al archivo Service1.svc.

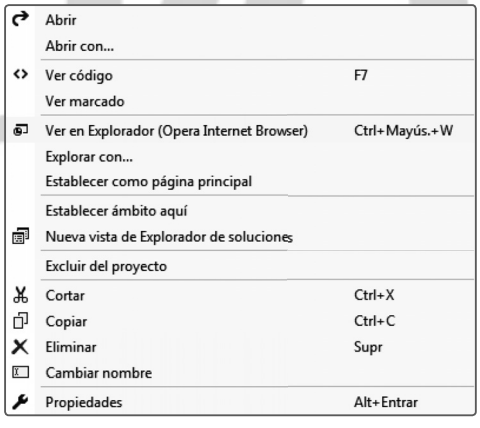
```
//Agregar las siguientes librerias
using System.Data;
using System.Data.SqlClient;

//Agregar la cadena de conexión a la base de datos
SqlConnection cn = new SqlConnection("server=.;database=CONTRATO;
                                      integrated security=SSPI");


//Implementar el método que lista registros
public DataSet LISTACONTRATISTA()
{
   SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
   DataSet ds = new DataSet();
   da.Fill(ds, "contratista");
   return ds;
}
```

De modo que el archivo Service1.svc contenga el siguiente código:

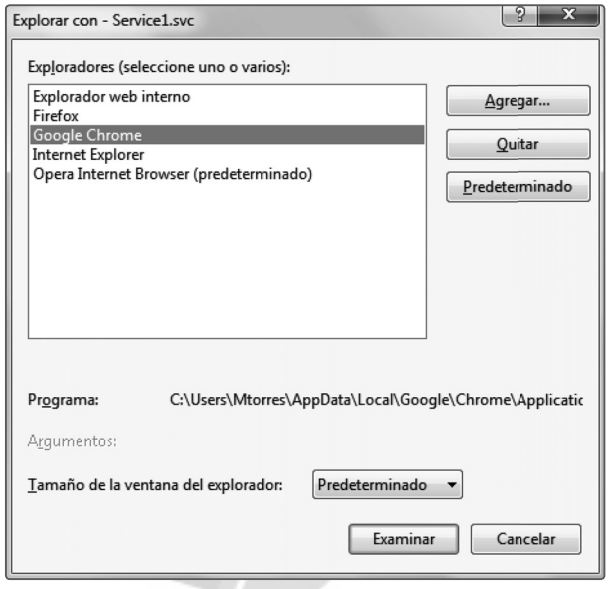
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

using System.Data;
using System.Data.SqlClient;
```

```
namespace WcfContrato
{
    public class Service1 : IService1
    {
       SqlConnection cn = new SqlConnection("SERVER=.;DATABASE=CONTRATO;
                                            INTEGRATED SECURITY=SSPI");


        public DataSet LISTACONTRATISTA()
        {
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
            DataSet ds = new DataSet();
            da.Fill(ds, "CONTRATISTA");
            return ds;
        }


        public string GetData(int value)
        {
            return string.Format("You entered: {0}", value);
        }

       public CompositeType GetDataUsingDataContract(CompositeType composite)
        {
            if (composite == null)
            {
                throw new ArgumentNullException("composite");
            }
            if (composite.BoolValue)
            {
                composite.StringValue += "Suffix";
            }
            return composite;
        }
    }
}
```

4. Luego, debe compilar la aplicación WCF para crear el servicio, para lo cual presione clic derecho sobre el archivo Service1.svc y seleccione **Ver en Explorador.**



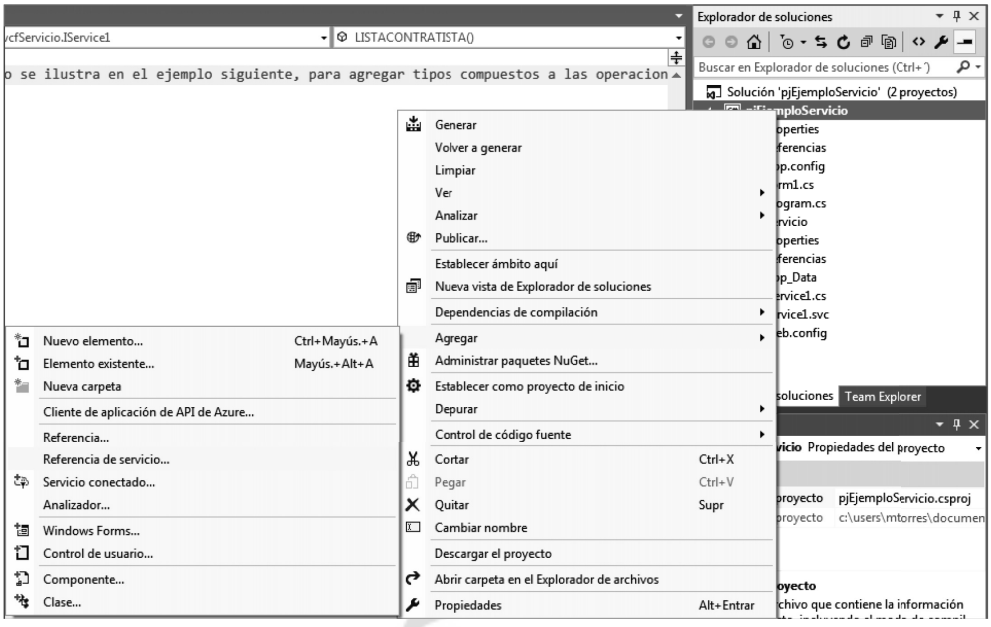
Si necesita modificar el tipo de Explorador web, seleccione de la ventana anterior **Explorar con...** y seleccione el explorador de su preferencia.



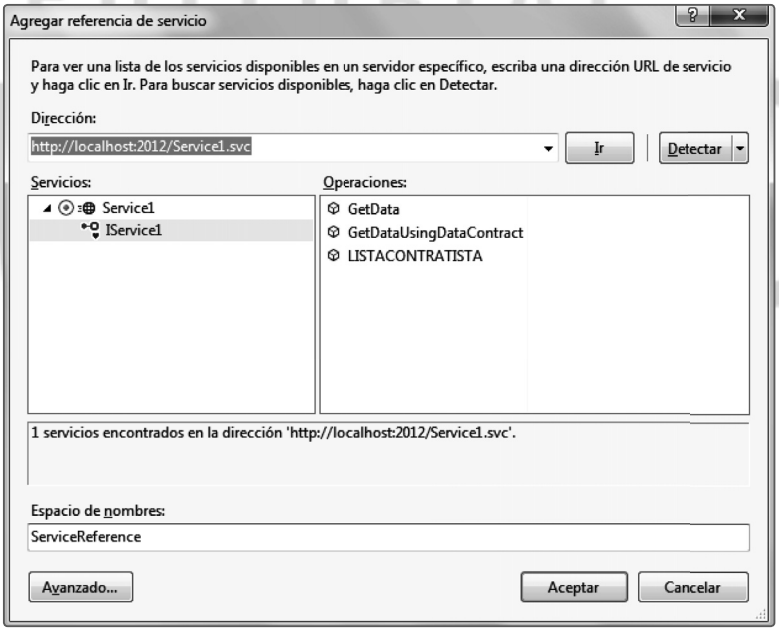
5. Si todo es correcto, se muestra el entorno web del servicio compilado.

Copie la URL que se muestra en el Explorador, en este caso «localhost:2012/Service1.svc».

6. Ahora, agregue el servicio al proyecto inicial. Para esto, presione clic derecho sobre el proyecto y seleccione **Agregar** > **Referencia de servicio...**



7. De la siguiente ventana, pegue el URL en la dirección y presione el botón **Ir**. Con esto, puede visualizar los servicios y las operaciones implementadas, que de ahora en adelante se puede usar.



8. Finalmente, debe agregar los siguientes códigos en un formulario para poder listar los registros desde el servicio.

```
//Referenciar al proyecto del servicio
using pjWCF01.ServiceReference1;

//Crear un objeto del servicio
Service1Client servicio = new Service1Client();

//Invocamos al método implementado en el servicio
dgContratista.DataSource=servicio.LISTACONTRATISTA().Tables["CONTRATISTA"];
```

Caso desarrollado 1 : WCF - Listado de contratistas

Implementar una aplicación que permita listar los datos de los contratistas usando servicios WCF.

Se tiene en cuenta lo siguiente:

- Implementar un procedimiento almacenado que permita mostrar los datos de los contratistas, como su código, nombre completo, teléfono y correo electrónico.
- Agregar un nuevo proyecto de servicio a la aplicación.

GUI Propuesta:

CODIGO	CONTRATISTA	TELEFONO	CORREO
CON001	PEDRO LOPEZ SANCHEZ	983645364	PLOPEZ@HOTMAIL.COM
CON002	LUIGUI JARAMILLO ORTIZ	98346545	LJARAMILLO@HOTMAIL.COM
CON003	OSCAR ZAMORA ROSAS	946566755	OZAMORA@HOTMAIL.COM
CON004	DICK GALVEZ CARBAJAL	998765436	DGALVEZ@GMAIL.COM
CON005	RICHARD VELEZMORO SO	945678899	RVELEZMORO@HOTMAIL.COM
CON006	NELSON MEJIA MEJIA	923456788	NMEJIA@HOTMAIL.COM
CON007	TOMAS GONZALES SANCHEZ	46578006	TGONZALES@GMAIL.COM

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjWCF1.

2. Implemente el siguiente procedimiento almacenado en Sql Server:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS DE LOS CONTRATISTAS
IF OBJECT_ID("SP_LISTACONTRATISTA")IS NOT NULL
        DROP PROC SP_LISTACONTRATISTA
GO
CREATE PROC SP_LISTACONTRATISTA
AS
        SELECT C.IDE_CON AS CODIGO,
               C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
               C.FON_CON AS TELEFONO,
               C.EMA_CON AS CORREO
               FROM CONTRATISTA C
GO
```

3. Agregue un nuevo proyecto de servicio WCF al proyecto, llámelo WCFContrato e implemente la siguiente operación de servicio al archivo IService1.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;

namespace WcfContrato
{
    // NOTA: puede usar el comando "Rename" del menú "Refactorizar"
    [ServiceContract]
    public interface IService1
    {

        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: agregue aquí sus operaciones de servicio
        [OperationContract]
        DataSet LISTACONTRATISTA();
    }


    // Utilice un contrato de datos
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
```

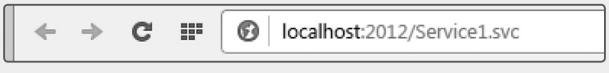
```
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

4. Implemente la siguiente operación en el archivo Service1.svc:

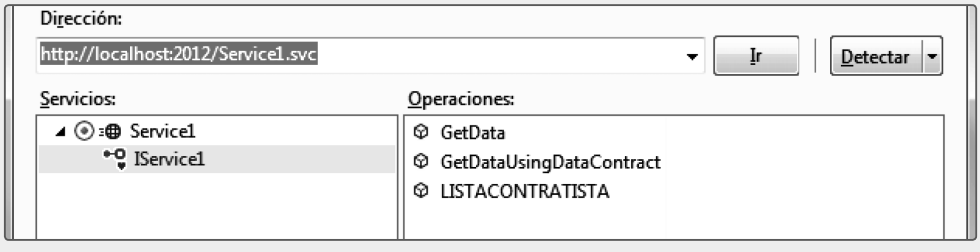
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;

//Librerias necesarias
using System.Data;
using System.Data.SqlClient;

namespace WcfContrato
{
    public class Service1 : IService1
    {
        //Cadena de conexion
        SqlConnection cn = new SqlConnection("SERVER=.;DATABASE=CONTRATO;
                                              INTEGRATED SECURITY=SSPI");

        //Método que lista los datos de los contratistas
        public DataSet LISTACONTRATISTA()
        {
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
```

```
            DataSet ds = new DataSet();
            da.Fill(ds, "CONTRATISTA");
            return ds;
        }

        public string GetData(int value)
        {
            return string.Format("You entered: {0}", value);
        }

        public CompositeType GetDataUsingDataContract(CompositeType composite)
        {
            if (composite == null)
            {
                throw new ArgumentNullException("composite");
            }
            if (composite.BoolValue)
            {
                composite.StringValue += "Suffix";
            }
            return composite;
        }
    }
}
```

5. Compile el archivo Service1.svc, presionando clic derecho sobre dicho archivo y seleccionando **Ver en explorador**, y del resultado copie la URL.



6. Agregue al proyecto una referencia de servicio y pegue la URL para visualizar los métodos implementados en el servicio.



El siguiente código se muestra dentro 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;
using pjWCF01.ServiceReference1;

namespace pjWCF01
{
  public partial class frmListaContratista : Form
  {

    Service1Client servicio = new Service1Client();

    public frmListaContratista()
    {
      InitializeComponent();
    }

    private void frmListaContratista_Load(object sender, EventArgs e)
    {
     dgContratista.DataSource=servicio.LISTACONTRATISTA().Tables["CONTRATISTA"];
    }
  }
}
```

Caso desarrollado 2 : WCF - Listado de fichas de devolución por contratista

Implementar una aplicación que permita listar los datos de los contratistas usando servicios WCF.

Se tiene en cuenta lo siguiente:

- Implementar un procedimiento almacenado que permita mostrar los datos de los contratistas, como su código, nombre completo, teléfono y correo electrónico.
- Agregar un nuevo proyecto de servicio a la aplicación.

GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjWCF2.
2. Implemente los siguientes procedimientos almacenados en Sql Server:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA TODAS LAS FICHAS DE DEVOLUCION
IF OBJECT_ID("SP_LISTAFICHADEVOLUCION")IS NOT NULL
       DROP PROC SP_LISTAFICHADEVOLUCION
GO
CREATE PROC SP_LISTAFICHADEVOLUCION
AS
       SELECT FD.COD_FIC AS CODIGO,
              C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
              CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
              E.DESC_EQU AS EQUIPO,
              FD.FDE_FIC AS FECHA
              FROM FICHA_DEVOLUCION FD
              JOIN CONTRATISTA C ON FD.IDE_CON=C.IDE_CON
              JOIN CLIENTE CL ON FD.IDE_CLI=CL.IDE_CLI
              JOIN EQUIPO E ON FD.IDE_EQU=E.IDE_EQU
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS DEL CONTRATISTA
IF OBJECT_ID("SP_CONTRATISTA")IS NOT NULL
```

```
        DROP PROC SP_CONTRATISTA
GO
CREATE PROC SP_CONTRATISTA
AS
        SELECT C.IDE_CON AS CODIGO,
               C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA
               FROM CONTRATISTA C
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LAS FICHAS SEGUN EL CONTRATISTA
IF OBJECT_ID("SP_LISTAFICHADEVOLUCIONXCONTRATISTA")IS NOT NULL
        DROP PROC SP_LISTAFICHADEVOLUCIONXCONTRATISTA
GO
CREATE PROC SP_LISTAFICHADEVOLUCIONXCONTRATISTA(@CON CHAR(6))
AS
        SELECT FD.COD_FIC AS CODIGO,
               C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
               CL.NOM_CLI+SPACE(1)+CL.PAT_CLI+SPACE(1)+CL.MAT_CLI AS CLIENTE,
               E.DESC_EQU AS EQUIPO,
               FD.FDE_FIC AS FECHA
               FROM FICHA_DEVOLUCION FD
               JOIN CONTRATISTA C ON FD.IDE_CON=C.IDE_CON
               JOIN CLIENTE CL ON FD.IDE_CLI=CL.IDE_CLI
               JOIN EQUIPO E ON FD.IDE_EQU=E.IDE_EQU
               WHERE FD.IDE_CON=@CON
GO
--PRUEBA: SP_LISTAFICHADEVOLUCIONXCONTRATISTA "CON003"
```

3. Agregue un nuevo proyecto de servicio WCF al proyecto, llámelo WCFContrato e implemente la siguiente operación de servicio al archivo IService1.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;

namespace WCFContrato
{
    // NOTA: puede usar el comando "Rename" del menú "Refactorizar"
    [ServiceContract]
    public interface IService1
    {
```

```
        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        // TODO: agregue aquí sus operaciones de servicio
        [OperationContract]
        DataSet LISTAFICHADEVOLUCION();

        [OperationContract]
        DataSet CONTRATISTA();

        [OperationContract]
        DataSet LISTAFICHADEVOLUCIONXCONTRATISTA(string contratista);
    }

    // Utilice un contrato de datos
    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }
}
```

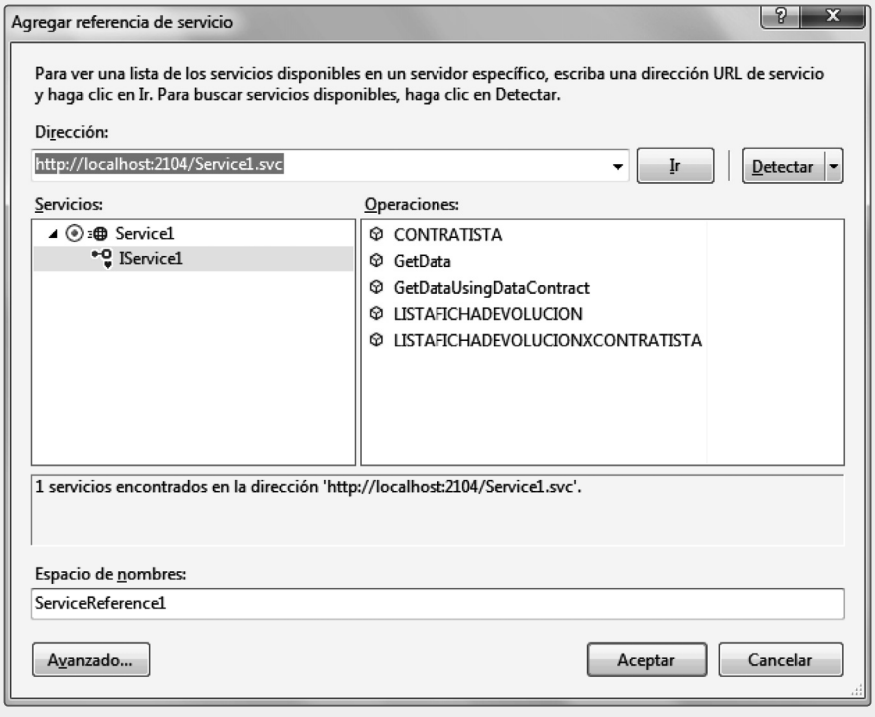
4. Implemente la siguiente operación en el archivo Service1.svc:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data.SqlClient;
using System.Data;
```

```
namespace WCFContrato
{
  public class Service1 : IService1
  {
    SqlConnection cn = new SqlConnection("SERVER=.;DATABASE=CONTRATO;
                                          INTEGRATED SECURITY=SSPI");

  public DataSet LISTAFICHADEVOLUCION()
  {
   SqlDataAdapter da = new SqlDataAdapter("SP_LISTAFICHADEVOLUCION", cn);
    DataSet ds = new DataSet();
    da.Fill(ds,"contrato");
    return ds;
  }

  public DataSet CONTRATISTA()
  {
    SqlDataAdapter da = new SqlDataAdapter("SP_CONTRATISTA", cn);
    DataSet ds = new DataSet();
    da.Fill(ds, "contratista");
    return ds;
  }

  public DataSet LISTAFICHADEVOLUCIONXCONTRATISTA(string contratista)
  {
 SqlDataAdapter da=new SqlDataAdapter("SP_LISTAFICHADEVOLUCIONXCONTRATISTA",cn);
  da.SelectCommand.CommandType = CommandType.StoredProcedure;
  da.SelectCommand.Parameters.Add("@CON", SqlDbType.Char).Value = contratista;
  DataSet ds = new DataSet();
  da.Fill(ds, "ficha");
  return ds;
  }

  public string GetData(int value)
  {
    return string.Format("You entered: {0}", value);
  }

  public CompositeType GetDataUsingDataContract(CompositeType composite)
  {
            if (composite == null)
            {
                throw new ArgumentNullException("composite");
            }
            if (composite.BoolValue)
            {
                composite.StringValue += "Suffix";
            }
            return composite;
  }
 }
}
```

5. Compile el archivo Service1.svc, presionando clic derecho sobre dicho archivo y seleccionando **Ver en explorador** y del resultado copie la URL.

6. Agregue al proyecto una referencia de servicio y pegue la URL para visualizar los métodos implementados en el servicio.



El siguiente código se muestra dentro 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;
using pjWCF02.ServiceReference1;

namespace pjWCF02
```

```
{
    public partial class frmListaFichaDevolucionxContratista : Form
    {
        Service1Client servicio = new Service1Client();
        public frmListaFichaDevolucionxContratista()
        {
            InitializeComponent();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
          dgFicha.DataSource = servicio.LISTAFICHADEVOLUCIONXCONTRATISTA(
                    cboContratista.SelectedValue.ToString()).Tables["FICHA"];
        }

        private void frmListaFichaDevolucionxContratista_Load(…)
        {
        dgFicha.DataSource=servicio.LISTAFICHADEVOLUCION().Tables["CONTRATO"];
          cboContratista.DataSource = servicio.CONTRATISTA()
                                                  .Tables["CONTRATISTA"];
          cboContratista.DisplayMember = "CONTRATISTA";
          cboContratista.ValueMember = "CODIGO";
        }
    }
}
```

EDITORIAL

Caso desarrollado 3 : WCF - Mantenimiento de equipos

Implementar una aplicación que permita realizar el mantenimiento de registros de la tabla Equipo usando servicios WCF.

Se tiene en cuenta lo siguiente:

- Implementar procedimientos almacenados en SQL Server que permitan listar, agregar, actualizar y eliminar los datos de los equipos, además de mostrar el último código de equipo registrado y listado de estados y tipos de equipos.
- Agregar un nuevo proyecto de servicio a la aplicación e implementar operaciones en el servicio para todo el mantenimiento de registros.
- Habilitar los botones nuevo, guardar, actualizar y eliminar de acuerdo a la selección del proceso. Por ejemplo, al seleccionar el botón nuevo, los botones nuevo, actualizar y eliminar se inhabilitan mientras que el botón grabar se habilita.

GUI Propuesta:

Solución:

1. Seleccione **Archivo** > **Nuevo Proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre al proyecto pjWCF3.
2. Implemente los siguientes procedimientos almacenados en Sql Server:

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS EQUIPOS
IF OBJECT_ID("SP_LISTAEQUIPOS")IS NOT NULL
        DROP PROC SP_LISTAEQUIPOS
GO
CREATE PROC SP_LISTAEQUIPOS
AS
        SELECT E.IDE_EQU AS CODIGO,
               E.DESC_EQU AS DESCRIPCION,
               T.DES_TIP AS TIPO,
               E.PREC_EQU AS PRECIO,
               ES.DES_EST AS ESTADO
               FROM EQUIPO E
               JOIN TIPO_EQUIPO T ON E.COD_TIP_EQU=T.COD_TIP_EQU
               JOIN ESTADO_EQUIPO ES ON E.COD_EST=ES.COD_EST
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS TIPOS DE EQUIPOS
IF OBJECT_ID("SP_LISTATIPOEQUIPOS")IS NOT NULL
        DROP PROC SP_LISTATIPOEQUIPOS
GO
CREATE PROC SP_LISTATIPOEQUIPOS
AS
```

```
        SELECT T.COD_TIP_EQU AS CODIGO,
               T.DES_TIP AS TIPO
               FROM TIPO_EQUIPO T
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS ESTADOS DE LOS EQUIPOS
IF OBJECT_ID("SP_LISTAESTADOS")IS NOT NULL
        DROP PROC SP_LISTAESTADOS
GO
CREATE PROC SP_LISTAESTADOS
AS
        SELECT ES.COD_EST AS CODIGO,
               ES.DES_EST AS ESTADO
               FROM ESTADO_EQUIPO ES
GO

--PROCEDIMIENTO ALMACENADO QUE MUESTRA EL ULTIMO CODIGO DE EQUIPO
IF OBJECT_ID("SP_ULTIMOEQUIPO")IS NOT NULL
        DROP PROC SP_ULTIMOEQUIPO
GO
CREATE PROC SP_ULTIMOEQUIPO
AS
        SELECT TOP 1 E.IDE_EQU AS CODIGO
               FROM EQUIPO E
               ORDER BY 1 DESC
GO

--PROCEDIMIENTO ALMACENADO QUE AGREGAR UN NUEVO EQUIPO
IF OBJECT_ID("SP_NUEVOEQUIPO")IS NOT NULL
        DROP PROC SP_NUEVOEQUIPO
GO
CREATE PROC SP_NUEVOEQUIPO(@COD CHAR(6), @TIP CHAR(6),@DES VARCHAR(40),@
PRE MONEY, @EST INT)
AS
        INSERT INTO EQUIPO VALUES (@COD,@TIP,@DES,@PRE,@EST)
GO

--PROCEDIMIENTO ALMACENADO QUE ACTUALIZA LOS DATOS DEL EQUIPO
IF OBJECT_ID("SP_ACTUALIZAEQUIPO")IS NOT NULL
        DROP PROC SP_ACTUALIZAEQUIPO
GO
CREATE  PROC  SP_ACTUALIZAEQUIPO(@COD  CHAR(6),  @TIP  CHAR(6),@DES  VAR-
CHAR(40),@PRE MONEY, @EST INT)
AS
        UPDATE EQUIPO
               SET COD_TIP_EQU=@TIP,DESC_EQU=@DES,PREC_EQU=@PRE,COD_EST=@EST
               WHERE IDE_EQU=@COD
GO

--PROCEDIMIENTO ALMACENADO QUE ELIMINA UN REGISTRO DE EQUIPO
IF OBJECT_ID("SP_ELIMINAEQUIPO")IS NOT NULL
        DROP PROC SP_ELIMINAEQUIPO
GO
CREATE PROC SP_ELIMINAEQUIPO(@COD CHAR(6))
AS
        DELETE EQUIPO WHERE IDE_EQU=@COD
GO
```

3. Agregue un nuevo proyecto de servicio WCF al proyecto, llámelo WCFMantenimiento e implemente la siguiente operación de servicio al archivo IService1.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data;

namespace WcfMantenimiento
{
    // NOTA: puede usar el comando "Rename"
    [ServiceContract]
    public interface IService1
    {

        [OperationContract]
        string GetData(int value);

        [OperationContract]
        CompositeType GetDataUsingDataContract(CompositeType composite);

        //Definiendo las operaciones del proyecto
        [OperationContract]
        DataSet LISTAEQUIPOS();

        [OperationContract]
        DataSet LISTATIPOEQUIPOS();

        [OperationContract]
        string ULTIMOEQUIPO();

        [OperationContract]
        DataSet LISTAESTADOS();

        [OperationContract]
        void NUEVOEQUIPO(Equipo objP);

        [OperationContract]
        void ACTUALIZAEQUIPO(Equipo objP);

        [OperationContract]
        void ELIMINAEQUIPO(Equipo objP);
    }

    [DataContract]
    public class CompositeType
    {
        bool boolValue = true;
        string stringValue = "Hello ";

        [DataMember]
        public bool BoolValue
        {
```

```
            get { return boolValue; }
            set { boolValue = value; }
        }

        [DataMember]
        public string StringValue
        {
            get { return stringValue; }
            set { stringValue = value; }
        }
    }

    //Definiendo el contrato de la clase EQUIPO
    [DataContract]
    public class Equipo
    {
        private string _codigo;
        [DataMember]
        public string codigo
        {
            get { return _codigo; }
            set { _codigo = value; }
        }

        private string _tipo;
        [DataMember]
        public string tipo
        {
            get { return _tipo; }
            set { _tipo = value; }
        }

        private string _descripcion;
        [DataMember]
        public string descripcion
        {
            get { return _descripcion; }
            set { _descripcion = value; }
        }

        private double _precio;
        [DataMember]
        public double precio
        {
            get { return _precio; }
            set { _precio = value; }
        }

        private int _estado;
        [DataMember]
        public int estado
        {
            get { return _estado; }
            set { _estado = value; }
        }
    }
}
```

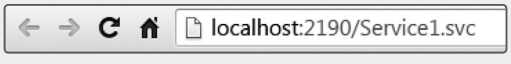
4. Implemente la siguiente operación en el archivo Service1.svc:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Text;
using System.Data.SqlClient;
using System.Data;

namespace WcfMantenimiento
{
    public class Service1 : IService1
    {
        SqlConnection cn = new SqlConnection("SERVER=.;DATABASE=CONTRATO;
                                              INTEGRATED SECURITY=SSPI");

        public DataSet LISTAEQUIPOS()
        {
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTAEQUIPOS", cn);
            DataSet ds = new DataSet();
            da.Fill(ds, "LISTA");
            return ds;
        }

        public DataSet LISTATIPOEQUIPOS()
        {
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTATIPOEQUIPOS", cn);
            DataSet ds = new DataSet();
            da.Fill(ds, "TIPO");
            return ds;
        }

        public DataSet LISTAESTADOS()
        {
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTAESTADOS", cn);
            DataSet ds = new DataSet();
            da.Fill(ds, "ESTADO");
            return ds;
        }

        public string ULTIMOEQUIPO()
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_ULTIMOEQUIPO", cn);
            cmd.CommandType = CommandType.StoredProcedure;
            return "EQ"+(int.Parse(cmd.ExecuteScalar().ToString()
                                        .Substring(2,4))+1).ToString("0000");
        }
```

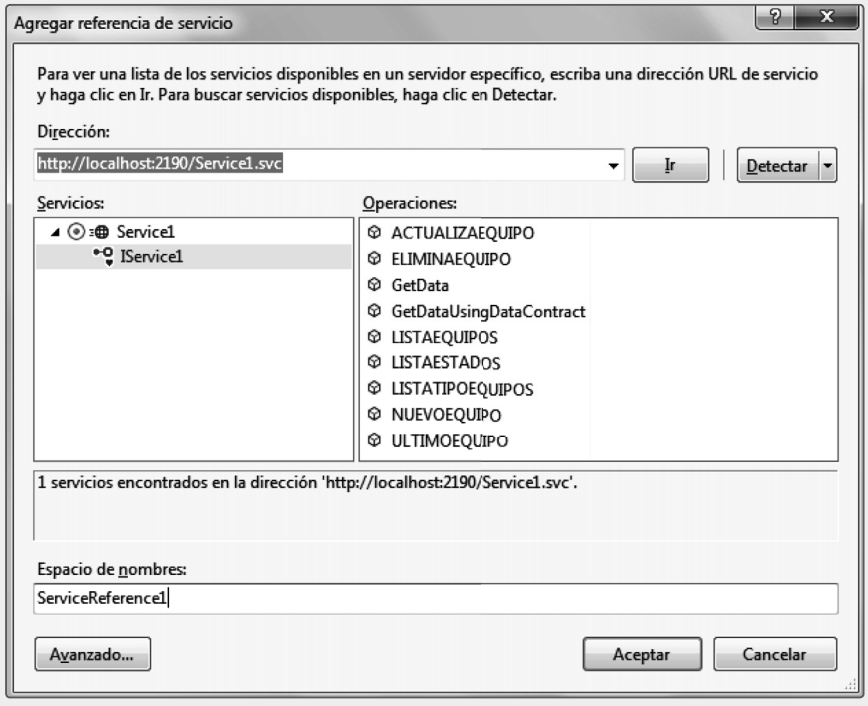
```
public void NUEVOEQUIPO(Equipo objP)
{
  cn.Open();
  SqlCommand cmd = new SqlCommand("SP_NUEVOEQUIPO", cn);
  cmd.CommandType = CommandType.StoredProcedure;

  cmd.Parameters.Add("@COD", SqlDbType.Char).Value = objP.codigo;
  cmd.Parameters.Add("@TIP", SqlDbType.Char).Value = objP.tipo;
 cmd.Parameters.Add("@DES", SqlDbType.VarChar).Value = objP.descripcion;
 cmd.Parameters.Add("@PRE", SqlDbType.Money).Value = objP.precio;
  cmd.Parameters.Add("@EST", SqlDbType.Int).Value = objP.estado;

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (Exception)
    {

    }
}

public void ACTUALIZAEQUIPO(Equipo objP)
{
  cn.Open();
  SqlCommand cmd = new SqlCommand("SP_ACTUALIZAEQUIPO", cn);
  cmd.CommandType = CommandType.StoredProcedure;

  cmd.Parameters.Add("@COD", SqlDbType.Char).Value = objP.codigo;
  cmd.Parameters.Add("@TIP", SqlDbType.Char).Value = objP.tipo;
 cmd.Parameters.Add("@DES", SqlDbType.VarChar).Value = objP.descripcion;
 cmd.Parameters.Add("@PRE", SqlDbType.Money).Value = objP.precio;
  cmd.Parameters.Add("@EST", SqlDbType.Int).Value = objP.estado;

    try
    {
        cmd.ExecuteNonQuery();
    }
    catch (Exception)
    {

    }
}
public void ELIMINAEQUIPO(Equipo objP)
```

```
        {
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_ELIMINAEQUIPO", cn);
            cmd.CommandType = CommandType.StoredProcedure;
          cmd.Parameters.Add("@COD", SqlDbType.Char).Value = objP.codigo;

            try
            {
                cmd.ExecuteNonQuery();
            }
            catch (Exception)
            {
            }
        }

        public string GetData(int value)
        {
            return string.Format("You entered: {0}", value);
        }

       public CompositeType GetDataUsingDataContract(CompositeType composite)
        {
            if (composite == null)
            {
                throw new ArgumentNullException("composite");
            }
            if (composite.BoolValue)
            {
                composite.StringValue += "Suffix";
            }
            return composite;
        }
    }
}
```

5. Compile el archivo Service1.svc, presionando clic derecho sobre dicho archivo y seleccionando **Ver en explorador**, y del resultado copie la URL.



6. Agregue al proyecto una referencia de servicio y pegue la URL para visualizar los métodos implementados en el servicio.



El siguiente código se muestra dentro 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;
using jpWCF03.ServiceReference1;

namespace jpWCF03
{
    public partial class frmMantenimientodeequipos : Form
    {
        Service1Client servicio = new Service1Client();

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

```
private void frmMantenimientodeequipos_Load(object sender, EventArgs e)
 {
     //Llenado de registros
     llenaEstado();
     llenaTipo();
     llenaEquipos();
     generaCodigo();

     //Habilitar botones
     tsGrabar.Enabled = false;
     tsActualizar.Enabled = false;
     tsEliminar.Enabled = false;
 }

 void llenaEstado() {
    cboEstado.DataSource = servicio.LISTAESTADOS().Tables["estado"];
     cboEstado.DisplayMember = "estado";
     cboEstado.ValueMember = "codigo";
 }

 void llenaTipo() {
    cboTipo.DataSource = servicio.LISTATIPOEQUIPOS().Tables["tipo"];
     cboTipo.DisplayMember = "tipo";
     cboTipo.ValueMember = "codigo";
 }

 void llenaEquipos() {
dgMantenimiento.DataSource=servicio.LISTAEQUIPOS().Tables["lista"];
 }

 void generaCodigo()
 {
     lblCodigo.Text = servicio.ULTIMOEQUIPO().ToString();
 }

 private void tsNuevo_Click(object sender, EventArgs e)
 {
     limpiarControles();

     generaCodigo();

     //Habilitar botones
     tsGrabar.Enabled = true;
     tsNuevo.Enabled = false;
     tsEliminar.Enabled = false;
     tsActualizar.Enabled = false;
 }

 private void tsGrabar_Click(object sender, EventArgs e)
 {
     Equipo objE = new Equipo();
```

```
            objE.codigo = lblCodigo.Text;
            objE.tipo = cboTipo.SelectedValue.ToString();
            objE.descripcion = txtDescripcion.Text;
            objE.precio = double.Parse(txtPrecio.Text);
            objE.estado = int.Parse(cboEstado.SelectedValue.ToString());

            servicio.NUEVOEQUIPO(objE);
            llenaEquipos();

            //Habilitar botones
            tsGrabar.Enabled = false;
            tsActualizar.Enabled = false;
            tsEliminar.Enabled = false;
            tsNuevo.Enabled = true;

            limpiarControles();
        }

        private void tsActualizar_Click(object sender, EventArgs e)
        {
            Equipo objE = new Equipo();
            objE.codigo = lblCodigo.Text;
            objE.tipo = cboTipo.SelectedValue.ToString();
            objE.descripcion = txtDescripcion.Text;
            objE.precio = double.Parse(txtPrecio.Text);
            objE.estado = int.Parse(cboEstado.SelectedValue.ToString());

            servicio.ACTUALIZAEQUIPO(objE);
            llenaEquipos();

            //Habilitar botones
            tsNuevo.Enabled = true;
            tsGrabar.Enabled = false;
            tsActualizar.Enabled = false;
            tsEliminar.Enabled = false;

            limpiarControles();
        }

        private void tsEliminar_Click(object sender, EventArgs e)
        {
            Equipo objE = new Equipo();
            objE.codigo = lblCodigo.Text;

            servicio.ELIMINAEQUIPO(objE);
            llenaEquipos();

            //Habilitar botones
            tsNuevo.Enabled = true;
            tsGrabar.Enabled = false;
            tsActualizar.Enabled = false;
            tsEliminar.Enabled = false;
```

```
            limpiarControles();
        }

        private void dgMantenimiento_MouseDoubleClick(…)
        {
        lblCodigo.Text = dgMantenimiento.CurrentRow.Cells[0].Value.ToString();
        cboTipo.Text = dgMantenimiento.CurrentRow.Cells[2].Value.ToString();
        txtDescripcion.Text=dgMantenimiento.CurrentRow.Cells[1].Value.ToString();
        txtPrecio.Text = dgMantenimiento.CurrentRow.Cells[3].Value.ToString();
        cboEstado.Text = dgMantenimiento.CurrentRow.Cells[4].Value.ToString();

           tsNuevo.Enabled = false;
           tsGrabar.Enabled = false;
           tsActualizar.Enabled = true;
           tsEliminar.Enabled = true;
        }

        void limpiarControles()
        {
            lblCodigo.Text = "";
            txtDescripcion.Clear();
            txtPrecio.Clear();
            cboTipo.Focus();
        }

        private void tsSalir_Click_1(object sender, EventArgs e)
        {
            this.Close();
        }
    }
}
```

MACRO

EDITORIAL
MACRO®

9

Capítulo

Programación en N-Capas

Capacidad terminal:

Implementar aplicaciones cliente servidor usando N-Capas de la programación.

Contenido

- Arquitectura en capas
- Aplicaciones distribuidas
- Creación de una solución en N-Capas
  Caso Desarrollado 1: Mantenimiento de contratistas en N-Capas
  Caso Desarrollado 2: Mantenimiento de clientes con imágenes en N-Capas
  Caso Desarrollado 3: Reporte de equipos con Crystal Report en N-Capas
  Caso Desarrollado 4: Reporte de equipos por estado y tipo con Crystal Report en N-Capas
  Caso Desarrollado 5: Reporte total de contratos por años con gráfico en N-Capas

EDITORIAL MACRO®

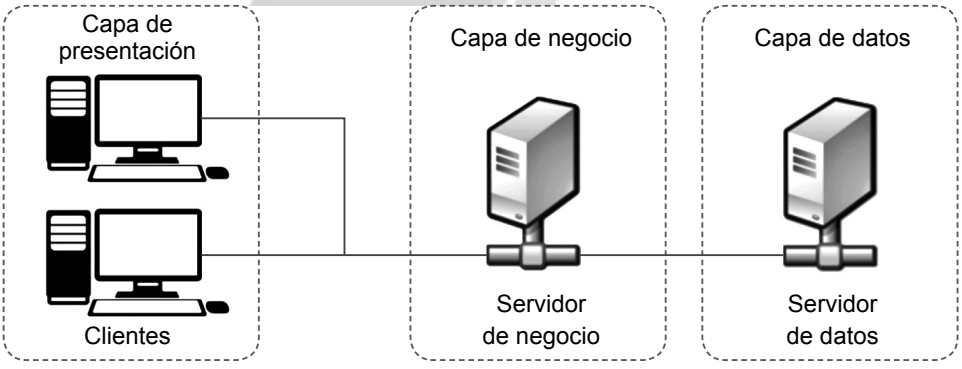
EDITORIAL
MACRO®

9.1 Arquitectura en capas

El desarrollo de una solución, usando la arquitectura en capas, tiene como objetivo separar el código, de tal forma que se lleve un orden de los elementos que componen la solución. El trabajo consiste en separar la lógica de negocio de la lógica que presenta el diseño.

El desarrollo de aplicaciones, a partir de este procedimiento, se lleva a cabo en varios niveles, los cuales permiten tener un mejor control de código, puesto que, al presentar un error, este solo se concentrará en la respectiva capa y no en todo el código.

A este tipo de arquitectura se le considera como arquitectura escalable, ya que permite ampliar la gama de necesidades que presentan los procesos de negocio. El diseño más utilizado actualmente es el diseño en tres niveles (o en tres capas):



a. **Capa de presentación:** Es la encargada de interactuar con el usuario. Es a partir de este punto donde el usuario podrá comunicarse con el sistema. Presenta las siguientes características:
   - Presenta el sistema en su totalidad al usuario mediante formularios.
   - Presenta información, al usuario, obtenida desde la base de datos.
   - Permite capturar información que el usuario necesita procesar en el sistema.
   - Valida la información que el usuario envía al sistema.
   - La capa de presentación muestra un aspecto amigable frente al usuario, ya que es la persona encargada de interactuar con él. El desarrollador de la aplicación debe diseñar interfaces de presentación pensadas en el usuario final.
   - La capa de presentación solo se comunica con la capa de negocio. Es decir, la capa de negocio presta servicios solicitados por la capa de presentación.

b. **Capa de negocio:** Es la encargada de negociar los procesos entre la capa de presentación y la capa de datos. Presenta las siguientes características:
   - La capa de negocio se encarga de recibir peticiones del usuario mediante la capa de presentación y es justamente por aquí donde le envía las respuestas.

EDITORIAL MACRO®

- La capa de negocio se caracteriza por no presentar implementación en sus métodos, solo presenta llamadas a métodos implementados en la capa de datos.
- La capa de negocio se comunica con la capa de presentación para recibir las peticiones de los usuarios, es decir, actúa como un servidor de peticiones.
- La capa de negocio se comunica con la capa de datos para obtener información solicitada desde la capa de presentación. Es decir, la capa de negocio es un tramitador de peticiones entre ambas capas.

c. **Capa de datos:** Es la encargada de conectarse al servidor de datos e implementar métodos que permitan obtener toda la información solicitada por el usuario. Presenta las siguientes características:

- La capa de datos puede hacer referencia a muchos gestores de base de datos haciendo un pool de conexiones.
- La capa de datos devuelve la respuesta a todas las peticiones realizadas desde la capa de negocio.

Finalmente, se puede mencionar que la implementación de las capas se puede realizar en una sola computadora, pero lo más común es ver a la capa de presentación en diferentes computadoras haciendo que muchos clientes hagan solicitudes cliente-servidor.

9.2 Aplicaciones distribuidas

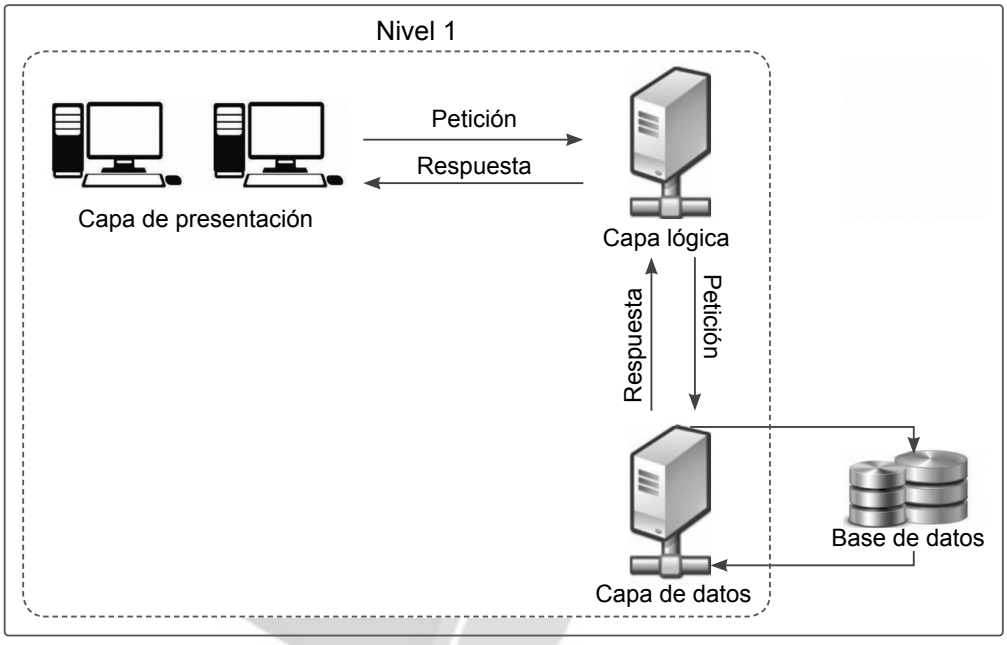
Si una aplicación usa la arquitectura N-Capas, en realidad solo está separando la lógica de negocio de toda la aplicación y las presenta sectorizadas, de tal forma que podría distribuir las capas en diferentes computadoras.

Aquí nace el término niveles, que hace referencia a como una aplicación que usa la arquitectura en capas es distribuida en una o más computadoras. Se puede decir que la aplicación de capas sectoriza la aplicación de forma lógica, mientras que los niveles distribuyen dichas capas de forma física. Entonces, se puede mencionar los siguientes casos:

- **Arquitectura de tres capas en un solo nivel**

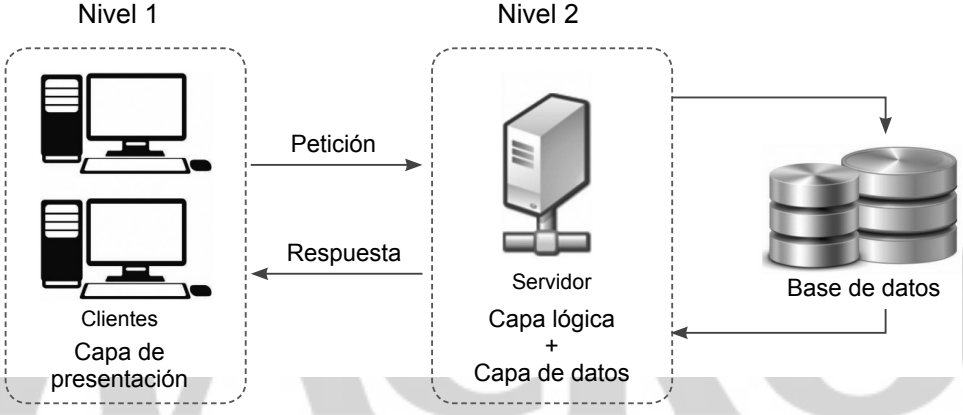
  Se dice así cuando todas las capas de una aplicación residen en una sola computadora.

EDITORIAL MACRO®



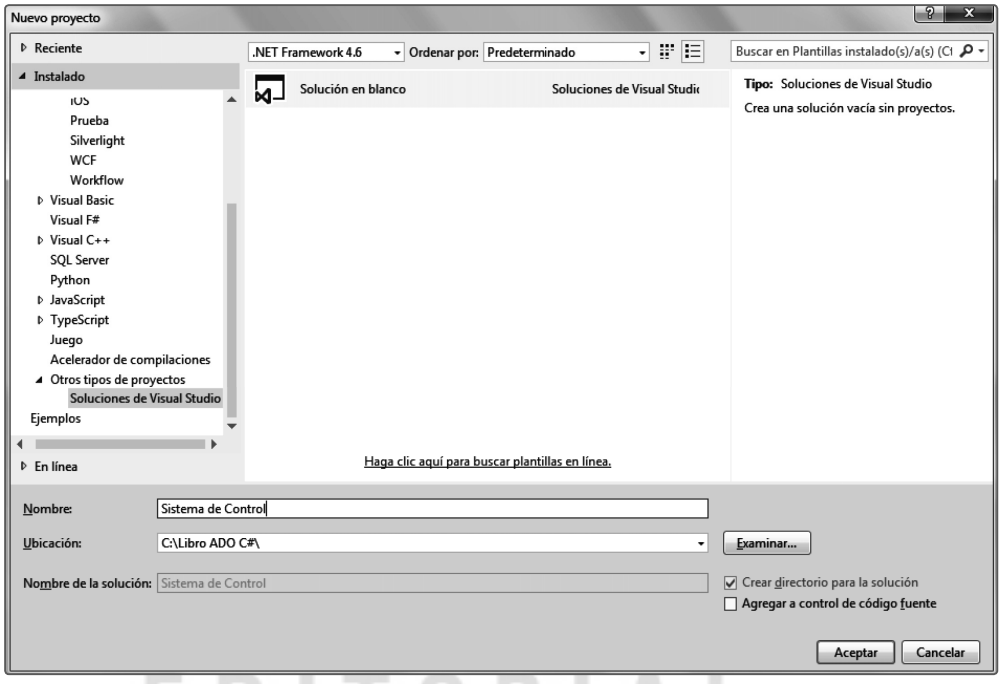
- **Arquitectura de tres capas de dos niveles**

  Se dice así cuando las capas se encuentran distribuidas en dos computadoras.

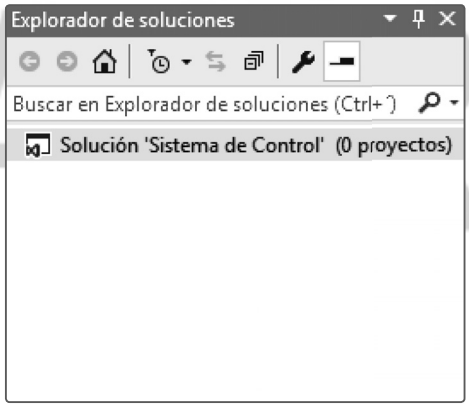


9.3 Creación de una solución en N-Capas

1. En Visual Studio 2015, seleccione **Archivo** > **Nuevo proyecto** > **Otros tipos de proyectos** > **Soluciones de Visual Studio** y asigne un nombre al proyecto, como se muestra en la siguiente imagen:



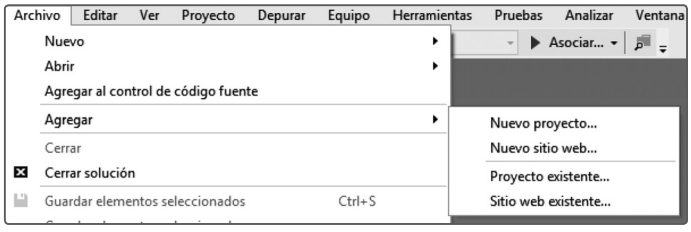
2. El Explorador de soluciones debe mostrarse de la siguiente manera:



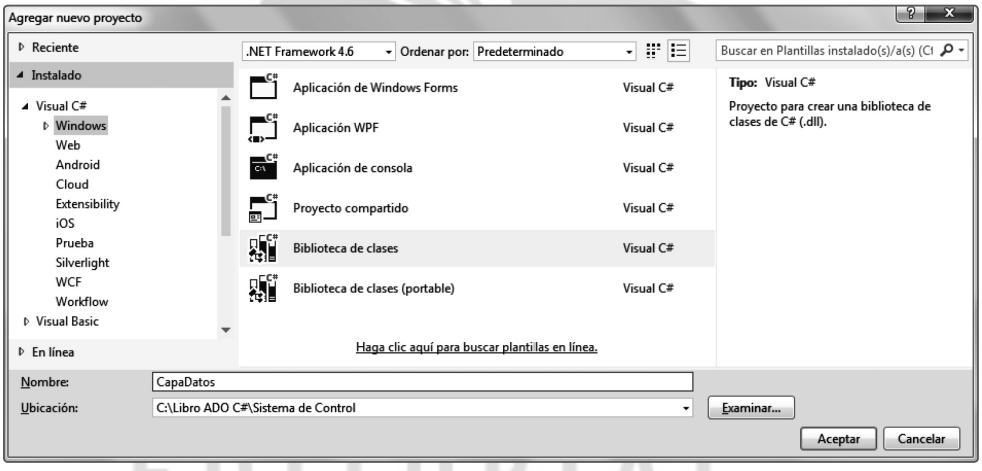
La solución presenta 0 proyectos y es así que se debe agregar proyectos dentro de la solución y estos serán considerados como las capas.

EDITORIAL MARO®

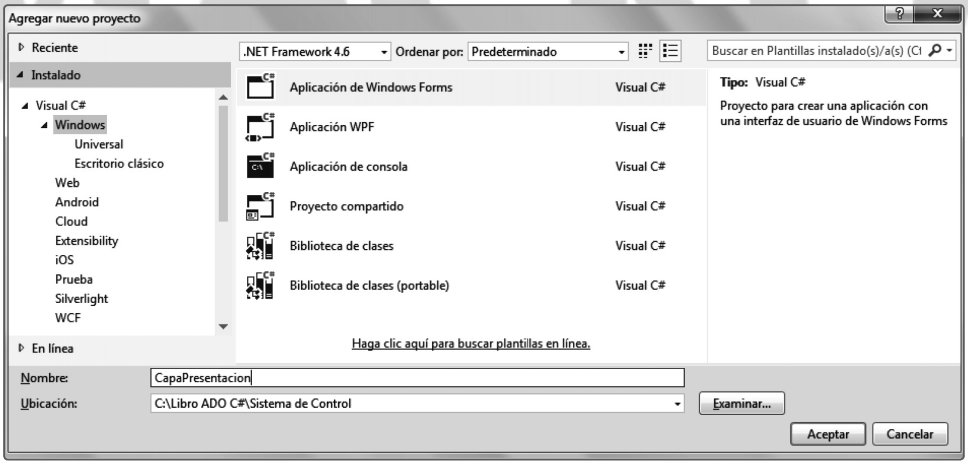
3. Para agregar una capa a la solución, seleccione **Archivo** > **Agregar** > **Nuevo proyecto**.



4. Luego, seleccione **Visual C#** > **Windows** > **Biblioteca de clases** y asigne el nombre a la capa, por ejemplo CapaDatos como se muestra en la siguiente imagen:

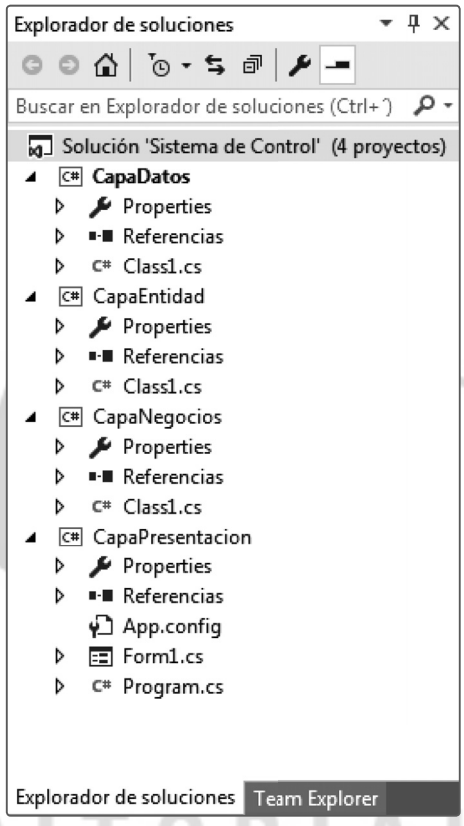


5. De la misma forma, agregue la capa negocios y la capa entidad.
6. Para agregar la capa de presentación, seleccione **Archivo** > **Agregar** > **Nuevo proyecto** > **Visual C#** > **Windows** > **Aplicación de Windows Forms** y asigne el nombre CapaPresentacion, como se muestra en la siguiente imagen:

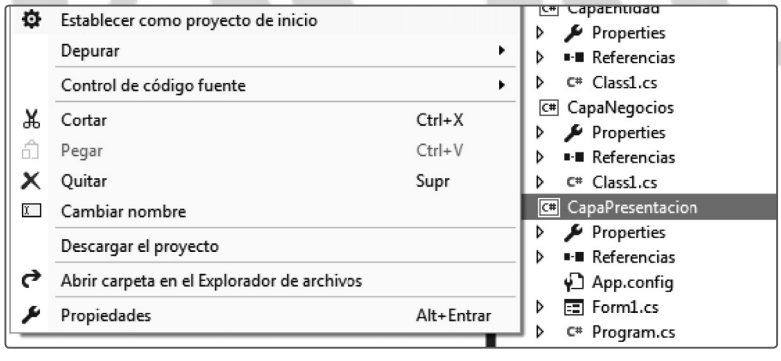


EDITORIAL MACRO®

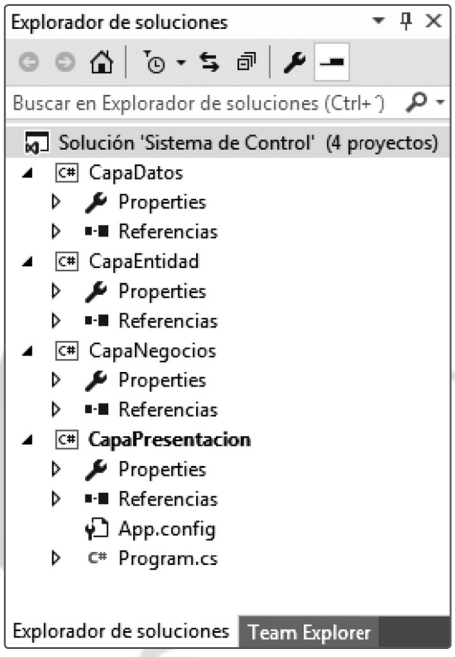
Inicialmente, el Explorador de soluciones muestra el siguiente aspecto:



7. Puede iniciar eliminando todos archivos Class1.cs de todas las capas, ya que será usted quien va a agregar las clases necesarias por capas. Luego, se tiene que establecer como proyecto de inicio a la capa de presentación, pues, es la que se presentará al ejecutar la aplicación. Para esto, presione clic derecho sobre la capa de presentación y seleccione **Establecer como proyecto de inicio**.



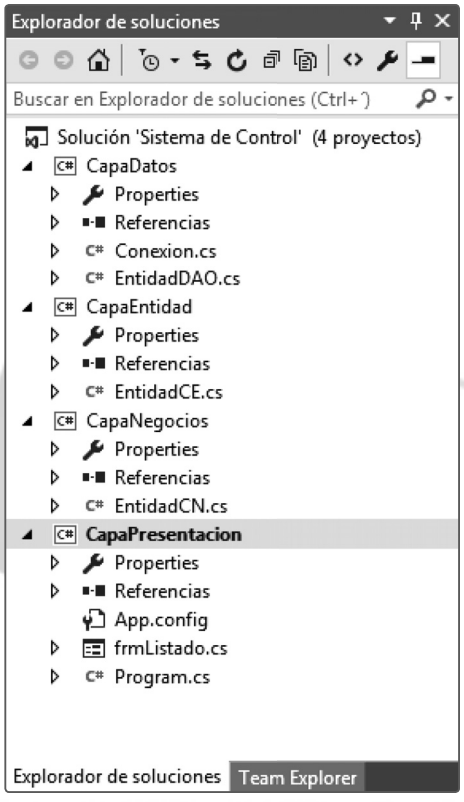
Finalmente, el explorador se presentará de la siguiente manera:



8. Luego, agregue las respectivas clases según la capa, de tal forma que tenga todo listo para la codificación. Las clases son como siguen:

Capa	Contenido	Descripción
Datos	Conexión.cs	Contiene un método que permite conectarse a la base de datos usando al archivo app.config.
	entidadDAO.cs	Contiene la implementación de todos los métodos que permiten realizar el listado y mantenimiento de registros de una determinada entidad.
Entidad	entidadCE	Contiene los atributos y los métodos get y set de la clase, a la cual se realizará el mantenimiento o consulta.
Lógica	entidadCN	Contiene la llamada a todos los métodos implementados en la clase EntidadDAO.
Presentación	App.config	Contiene la configuración de la cadena de conexión a la base de datos.
	frmListado	Representa a los formularios que se implementarán en el proyecto. La cantidad de formularios lo determinará el desarrollador.

Finalmente, el Explorador de soluciones debe mostrarse de la siguiente manera:

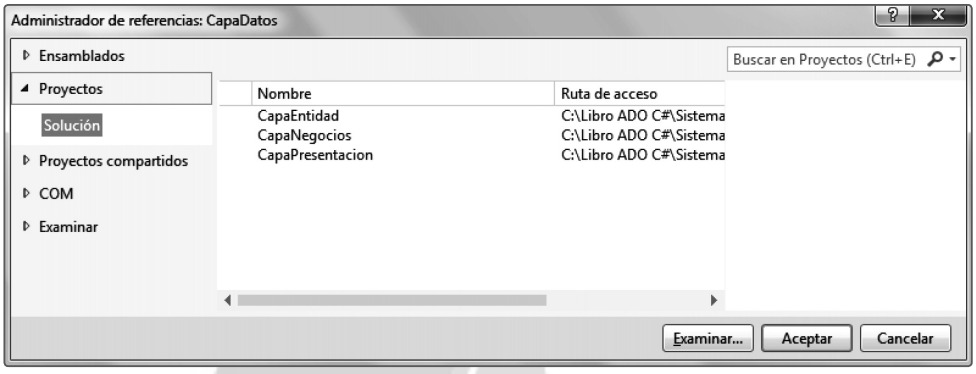


9. Ahora, debe crear una asociación entre las capas, ya que muchas de ellas necesitan compartir información.

Capas	Descripción
Datos a Entidad	En la capa datos se encuentran métodos, cuyo parámetro necesita hacer referencia a una determinada clase que se encuentra implementada en la clase Entidad.
Negocios a Datos	En la capa lógica se encuentran métodos sin implementación, los cuales necesitan asociarse a los métodos implementados en la clase Datos.
Negocios a Entidad	En la capa lógica se encuentran métodos sin implementación, los cuales hacen referencia a parámetros que necesitan la invocación de una clase desde la capa Entidad.
Presentación a Negocios	En la capa de presentación se encuentran los formularios, que usan los métodos que se encuentran referenciados en la capa lógica.
Presentación a Entidad	En la capa de presentación se encuentran los formularios, los cuales necesitan referenciar a las clases que se encuentran en la capa entidad.

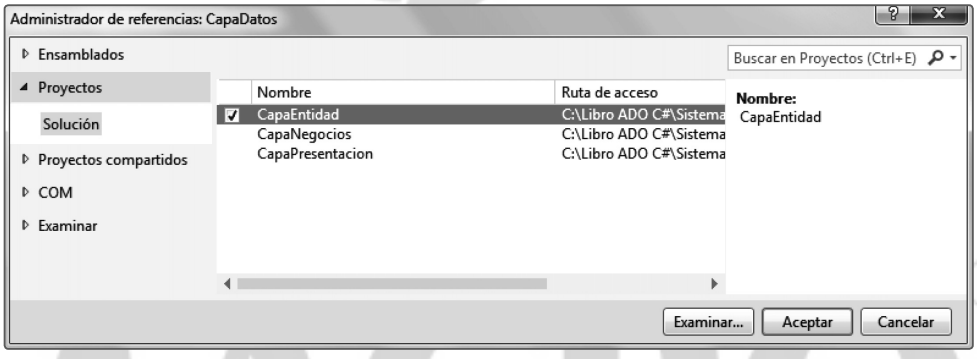
Para referenciar las capas, se debe seguir los siguientes pasos:

1. Haga clic derecho en la capa **Agregar** > **Referencia**.
2. Seleccione **Solución** y luego active el check de la capa a asociarse.

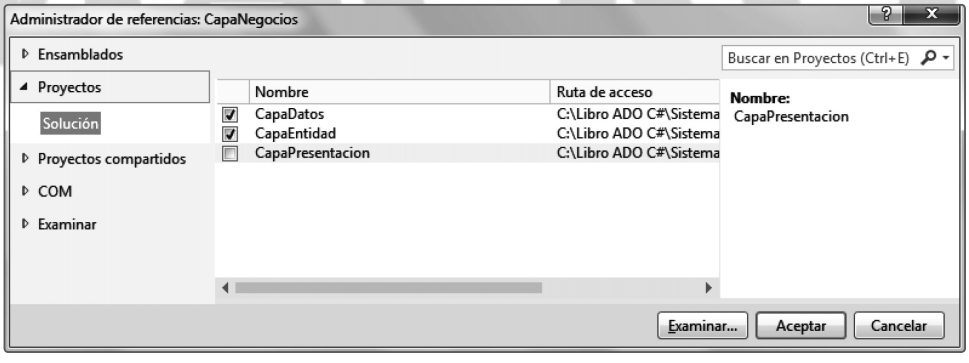


3. Finalmente, observe las referencias entre las diferentes capas:

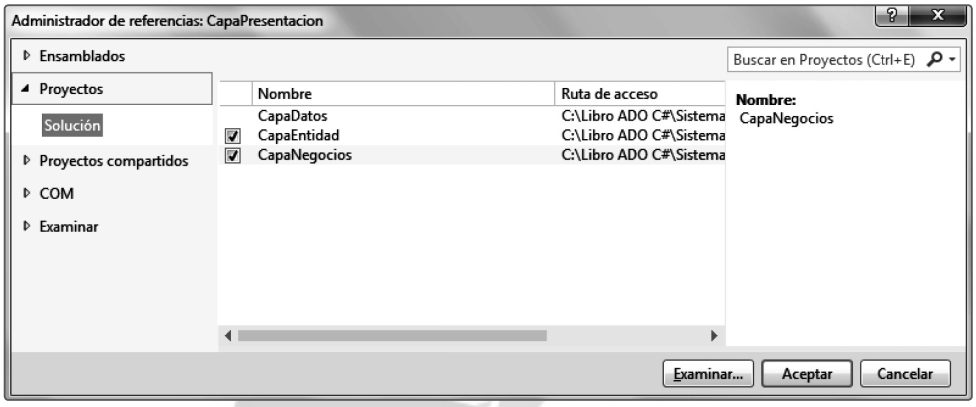
- Referencia de la capa Datos.



- Referencia de la capa Negocios.



- Referencia de la capa Presentación.



Caso desarrollado 1 : Mantenimiento de contratistas en N-Capas

Implementar una solución que permite realizar el mantenimiento de registros de la tabla Contratista usando la base de datos contrato.

Se tiene en cuenta lo siguiente:

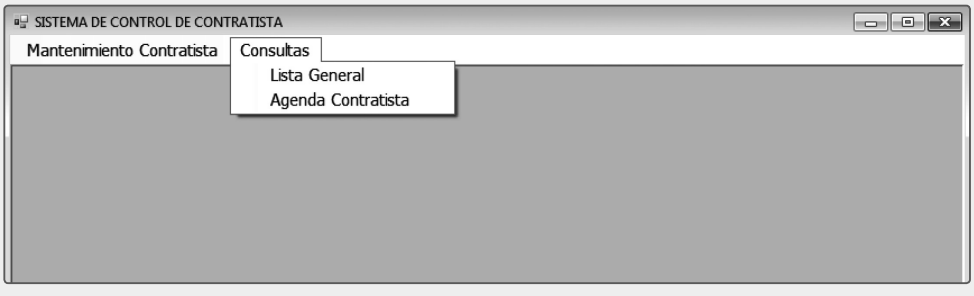
- Crear una solución en blanco.
- Implementar procedimientos almacenados para el proceso de mantenimiento de registros de los contratistas.
- Agregar tres proyectos de bibliotecas de clases y llámelos CapaDatos, CapaEntidad y CapaNegocio.
- Agregar un proyecto de Windows Forms y llámelo CapaPresentacion
- La distribución de clases en las capas de la solución es como sigue:

Capa	Clase	Descripción
Capa Datos	Conexión	Contiene un método que permite referenciar a la cadena de conexión especificada en el archivo app.config.
	ContratistaDAO	Contiene la implementación de los métodos que permiten realizar un mantenimiento de registros de la tabla Contratista.
Capa Entidad	ContratistaCE	Contiene los atributos y los métodos Get y Set de la clase Contratista.
Capa Negocios	ContratistaCN	Contiene la misma cantidad de métodos que tiene la clase ContratistaDAO, pero carece de implementación.

Capa Presentación	frmListaGeneral	Formulario que permite listar los registros de los contratistas.
	frmAgendaContratista	Formulario que permita mostrar los registros de los contratistas filtrando por su primera letra de su nombre.
	frmNuevoContratista	Formulario que permite agregar un nuevo registro a la tabla Contratista.
	frmActualizarContratista	Formulario que permite actualizar los datos de un determinado contratista previamente buscado desde el mismo formulario.
	frmBuscarContratista	Formulario que permite mostrar los datos de los contratistas por medio de la búsqueda del código de contratista.
	frmEliminarContratista	Formulario que permite eliminar un registro de contratista previamente buscado en el mismo formulario.
	frmMenuPrincipal	Formulario primario MDI que permite referenciar a todos los formularios de la solución.

- Las opciones que presenta el formulario frmMenuPrincipal se muestra de la siguiente manera:
  - Menú de opciones del mantenimiento de registros.

  - Menú de opciones de las consultas realizadas a la tabla contratista.



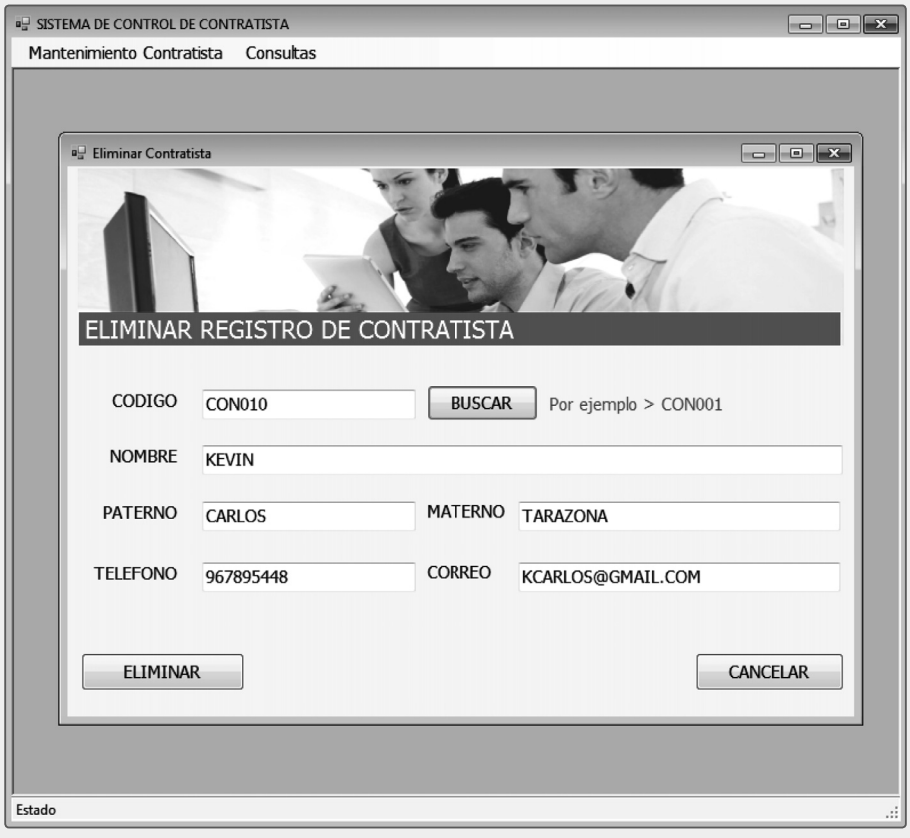
- Los formularios se deben presentar de la siguiente manera:
  - **Formulario de listado de contratistas:** El cual presenta los datos de los contratistas.

  - **Formulario de la agenda de contratistas:** Donde se filtrarán los datos a partir de la inicial del nombre del contratista. Esto se realizará presionando un botón.

- **Formulario de registro del nuevo contratista:** Al iniciar el formulario, se mostrará el código autogenerado del nuevo contratista.

- **Formulario de actualización de datos del contratista:** Se debe mostrar los datos de los contratistas en un control DataGridView y que, al presionar doble clic sobre uno de ellos los datos, se mostrarán en los controles para realizar las modificaciones correspondientes.

- **Formulario de eliminación de un determinado contratista:** Para encontrar al contratista, debe buscarlo por su código. Una vez mostrada la información, proceda a eliminar.



Solución:

1. Ejecute los siguientes procedimientos almacenados en SQL Server.

```
--PROCEDIMIENTO ALMACENADO QUE LISTA LOS CONTRATISTAS
IF OBJECT_ID("SP_LISTACONTRATISTA")IS NOT NULL
       DROP PROC SP_LISTACONTRATISTA
GO
CREATE PROC SP_LISTACONTRATISTA
AS
       SELECT C.IDE_CON AS CODIGO,
              C.NOM_CON AS NOMBRE,
              C.PAT_CON AS PATERNO,
              C.MAT_CON AS MATERNO,
              C.FON_CON AS TELEFONO,
              C.EMA_CON AS CORREO
              FROM CONTRATISTA C
GO
```

```
--PROCEDIMIENTO ALMACENADO QUE MUESTRA EL ULTIMO CÓDIGO DE CONTRATISTA
IF OBJECT_ID("SP_ULTIMOCONTRATISTA")IS NOT NULL
       DROP PROC SP_ULTIMOCONTRATISTA
GO
CREATE PROC SP_ULTIMOCONTRATISTA
AS
        SELECT TOP 1 C.IDE_CON
              FROM CONTRATISTA C
              ORDER BY 1 DESC
GO

--PROCEDIMIENTO ALMACENADO QUE AGREGA UN NUEVO CONTRATISTA
IF OBJECT_ID("SP_NUEVOCONTRATISTA")IS NOT NULL
       DROP PROC SP_NUEVOCONTRATISTA
GO
CREATE PROC SP_NUEVOCONTRATISTA(@COD CHAR(6),@NOM VARCHAR(30),
                                @PAT VARCHAR(20), @MAT VARCHAR(20),
                                @FON VARCHAR(15),@EMA VARCHAR(50))
AS
       INSERT INTO CONTRATISTA
              VALUES (@COD,@NOM,@PAT,@MAT,@FON,@EMA)
GO

--PROCEDIMIENTO ALMACENADO QUE ACTUALIZA LOS DATOS DEL CONTRATISTA
IF OBJECT_ID("SP_ACTUALIZACONTRATISTA")IS NOT NULL
       DROP PROC SP_ACTUALIZACONTRATISTA
GO
CREATE PROC SP_ACTUALIZACONTRATISTA(@COD CHAR(6),@NOM VARCHAR(30),
                                    @PAT VARCHAR(20), @MAT VARCHAR(20),
                                    @FON VARCHAR(15),@EMA VARCHAR(50))
AS
       UPDATE CONTRATISTA
              SET NOM_CON=@NOM,PAT_CON=@PAT,
                 MAT_CON=@MAT,FON_CON=@FON,EMA_CON=@EMA
              WHERE IDE_CON=@COD
GO

--PROCEDIMIENTO ALMACENADO QUE ELIMINA UN DETERMINADO CONTRATISTA
IF OBJECT_ID("SP_ELIMINACONTRATISTA")IS NOT NULL
       DROP PROC SP_ELIMINACONTRATISTA
GO
CREATE PROC SP_ELIMINACONTRATISTA(@COD CHAR(6))
AS
       DELETE CONTRATISTA
        WHERE IDE_CON = @COD
GO

--PROCEDIMIENTO ALMACENADO QUE BUSCA LOS DATOS DE UN CONTRATISTA
IF OBJECT_ID("SP_BUSCACONTRATISTA")IS NOT NULL
       DROP PROC SP_BUSCACONTRATISTA
GO
CREATE PROC SP_BUSCACONTRATISTA(@COD CHAR(6))
AS
```

```
        SELECT C.IDE_CON AS CODIGO,
               C.NOM_CON AS NOMBRE,
               C.PAT_CON AS PATERNO,
               C.MAT_CON AS MATERNO,
               C.FON_CON AS TELEFONO,
               C.EMA_CON AS CORREO
               FROM CONTRATISTA C
               WHERE IDE_CON = @COD
GO

--PROCEDIMIENTO ALMACENADO QUE FILTRA LOS DATOS DE LOS CONTRATISTAS
IF OBJECT_ID("SP_AGENDACONTRATISTA")IS NOT NULL
       DROP PROC SP_AGENDACONTRATISTA
GO
CREATE PROC SP_AGENDACONTRATISTA (@LETRA CHAR(1))
AS
       SELECT C.IDE_CON AS CODIGO,
              C.NOM_CON+SPACE(1)+C.PAT_CON+SPACE(1)+C.MAT_CON AS CONTRATISTA,
              C.FON_CON AS TELEFONO,
              C.EMA_CON AS CORREO
              FROM CONTRATISTA C
              WHERE C.NOM_CON LIKE @LETRA+"%"
GO
```

2. Cree una solución en blanco en Visual Studio 2015 y agregue tres proyectos del tipo biblioteca de clase a la solución y asigne los nombres CapaDatos, CapaEntidad y CapaNegocios.
3. Agregue a la solución un proyecto de tipo Windows Forms y asigne el nombre capaPresentacion.
4. Elimine los archivos Class1.cs de todas las capas.
5. A la capa de datos, agregue la referencia System.Configuration.
6. Desde la capa de presentación, modifique el archivo app.config, de tal forma que permita conectarse a la base de datos Contrato.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="SERVER=.;DATABASE=CONTRATO;
                                     INTEGRATED SECURITY=SSPI"/>
  </connectionStrings>
</configuration>
```

7. Agregue la clase ContratistaCE a la capa entidad, esta tendrá la misión de especificar todos los métodos get y set que componen la clase Contratista.

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

namespace CapaEntidad
{
   public class ContratistaCE
    {
        public string codigo { get; set; }
        public string nombre { get; set; }
        public string paterno { get; set; }
        public string materno { get; set; }
        public string telefono { get; set; }
        public string correo { get; set; }
    }
}
```

8. Agregue la clase Conexión a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Configuration;

namespace CapaDatos
{
  public  class Conexion
    {
      public SqlConnection getConecta()
      {
          SqlConnection cn = new SqlConnection(ConfigurationManager
                                 .ConnectionStrings["cn"].ConnectionString);
          return cn;
      }
    }
}
```

9. Agregue la clase ContratistaDAO a la capa de datos y coloque el siguiente código:

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

using CapaEntidad;
namespace CapaDatos
{
  public  class ContratistaDAO
    {
      SqlConnection cn;
      Conexion objCON = new Conexion();

      public DataTable LISTACONTRATISTA()
      {
          cn = objCON.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTACONTRATISTA", cn);
          DataTable dt = new DataTable();
          da.Fill(dt);
          return dt;

      }

      public string NUEVOCODIGO()
      {
          cn = objCON.getConecta();
          cn.Open();
          SqlCommand cmd = new SqlCommand("SP_ULTIMOCONTRATISTA", cn);
          SqlDataAdapter da = new SqlDataAdapter();
          da.SelectCommand = cmd;
          int codigo = int.Parse(cmd.ExecuteScalar().ToString()
                                                    .Substring(3, 3)) + 1;

          return "CON" +codigo.ToString("000");
      }

      public void ACTUALIZACONTRATISTA(ContratistaCE c)
      {
          cn = objCON.getConecta();
          cn.Open();
          SqlCommand cmd = new SqlCommand();
          cmd.CommandText = "SP_ACTUALIZACONTRATISTA";
          cmd.CommandType = CommandType.StoredProcedure;
          cmd.Connection = cn;

          cmd.Parameters.Add("@COD", SqlDbType.Char).Value = c.codigo;
         cmd.Parameters.Add("@NOM", SqlDbType.VarChar).Value = c.nombre;
        cmd.Parameters.Add("@PAT", SqlDbType.VarChar).Value = c.paterno;
        cmd.Parameters.Add("@MAT", SqlDbType.VarChar).Value = c.materno;
       cmd.Parameters.Add("@FON", SqlDbType.VarChar).Value = c.telefono;
```

```
        cmd.Parameters.Add("@EMA", SqlDbType.VarChar).Value = c.correo;

        try
        {
           cmd.ExecuteNonQuery();
        }
        catch (Exception)
        {
        }
    }

    public void NUEVOCONTRATISTA(ContratistaCE c)
    {
        cn = objCON.getConecta();
        cn.Open();
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "SP_NUEVOCONTRATISTA";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = cn;

        cmd.Parameters.Add("@COD", SqlDbType.Char).Value = c.codigo;
        cmd.Parameters.Add("@NOM", SqlDbType.VarChar).Value = c.nombre;
        cmd.Parameters.Add("@PAT", SqlDbType.VarChar).Value =c.paterno;
       cmd.Parameters.Add("@MAT", SqlDbType.VarChar).Value = c.materno;
      cmd.Parameters.Add("@FON", SqlDbType.VarChar).Value = c.telefono;
        cmd.Parameters.Add("@EMA", SqlDbType.VarChar).Value = c.correo;

        try
        {
            cmd.ExecuteNonQuery();
        }
        catch (Exception)
        {
        }
    }

    public void ELIMINACONTRATISTA(ContratistaCE C)
    {
        cn = objCON.getConecta();
        cn.Open();
        SqlCommand cmd = new SqlCommand();
        cmd.CommandText = "SP_ELIMINACONTRATISTA";
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Connection = cn;

        cmd.Parameters.Add("@COD", SqlDbType.Char).Value = C.codigo;
        try
        {
             cmd.ExecuteNonQuery();
        }
        catch (Exception)
        {
        }
    }

public DataTable BUSCACONTRATISTA(ContratistaCE C)
```

```
 {
    cn = objCON.getConecta();
    SqlDataAdapter da = new SqlDataAdapter("SP_BUSCACONTRATISTA", cn);
    da.SelectCommand.CommandType = CommandType.StoredProcedure;
    da.SelectCommand.Parameters.Add("@COD", SqlDbType.Char).Value = C.codigo;
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
 }

  public DataTable AGENDACONTRATISTA(string letra)
 {
    cn = objCON.getConecta();
    SqlDataAdapter da = new SqlDataAdapter("SP_AGENDACONTRATISTA", cn);
    da.SelectCommand.CommandType = CommandType.StoredProcedure;
    da.SelectCommand.Parameters.Add("@letra",SqlDbType.VarChar).Value=letra;
    DataTable dt = new DataTable();
    da.Fill(dt);
    return dt;
  }
 }
}
```

10. Agregue la clase ContratistaCN a la capa de negocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using CapaEntidad;
using CapaDatos;

namespace CapaNegocio
{
  public  class ContratistaCN
    {
      ContratistaDAO objDAO = new ContratistaDAO();

      public DataTable LISTACONTRATISTA()
      {
          return objDAO.LISTACONTRATISTA();
      }

      public string NUEVOCODIGO()
      {
          return objDAO.NUEVOCODIGO();
      }

      public void NUEVOCONTRATISTA(ContratistaCE c)
      {
          objDAO.NUEVOCONTRATISTA(c);
      }
```

```
        public void ACTUALIZACONTRATISTA(ContratistaCE c)
        {
            objDAO.ACTUALIZACONTRATISTA(c);
        }

        public void ELIMINACONTRATISTA(ContratistaCE c)
        {
            objDAO.ELIMINACONTRATISTA(c);
        }

        public DataTable BUSCACONTRATISTA(ContratistaCE c)
        {
             return objDAO.BUSCACONTRATISTA(c);
        }

        public DataTable AGENDACONTRATISTA(string letra)
        {
             return objDAO.AGENDACONTRATISTA(letra);
        }
    }
}
```

11. Observe el código del formulario frmListaGeneral:

```
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;
//Referencia a la capa de negocio
using CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmListaGeneral : Form
    {
        //Objeto de la clase ContratistaCN
        ContratistaCN objCO = new ContratistaCN();

        public frmListaGeneral()
        {
            InitializeComponent();
        }

        private void frmListaGeneral_Load(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCO.LISTACONTRATISTA();
        }
    }
}
```

12. Observe el código del formulario frmAgendaContratista:

```
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;

//Referencia a la capa de negocio y entidad
using CapaEntidad;
using CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmAgendaContratista : Form
    {
        //Objeto de la clase ContratistaCN
        ContratistaCN objCon = new ContratistaCN();

        public frmAgendaContratista()
        {
            InitializeComponent();
        }

       private void frmAgendaContratista_Load(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.LISTACONTRATISTA();
        }

        private void btnA_Click(object sender, EventArgs e)
        {
           dgContratista.DataSource = objCon.AGENDACONTRATISTA("A");
        }

        private void btnP_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("P");
        }

        private void btnB_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("B");
        }

        private void btnC_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("C");
        }

        private void btnD_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("D");
        }
```

```
        private void btnE_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("E");
        }

        private void btnF_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("F");
        }

        private void btnG_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("G");
        }

        private void btnH_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("H");
        }

        private void btnI_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("I");
        }

        private void btnJ_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("J");
        }

        private void btnK_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("K");
        }

        private void btnL_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("L");
        }

        private void btnM_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("M");
        }

        private void btnN_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("N");
        }

        private void btnO_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("O");
        }
```

```
        private void btnQ_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("Q");
        }

        private void btnR_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("R");
        }

        private void btnS_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("S");
        }

        private void btnT_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("T");
        }

        private void btnU_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("U");
        }

        private void btnV_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("V");
        }

        private void btnW_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("W");
        }

        private void btnX_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("X");
        }

        private void btnY_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("Y");
        }

        private void btnZ_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.AGENDACONTRATISTA("Z");
        }

        private void btnTodos_Click(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.LISTACONTRATISTA();
        }
    }
}
```

13. Observe el código del formulario frmNuevoContratista

```
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;

//Referencia a la capa de negocio y entidad
using CapaNegocio;
using CapaEntidad;

namespace CapasPresentacion
{
    public partial class frmNuevoContratista : Form
    {
        //Objeto de la clase ContratistaCN
        ContratistaCN objCon = new ContratistaCN();

        public frmNuevoContratista()
        {
            InitializeComponent();
        }

        private void frmNuevoContratista_Load(object sender, EventArgs e)
        {
            generarCodigo();

       }

       //Método que muestra el código autogenerado
        void generarCodigo()
        {
            lblCodigo.Text = objCon.NUEVOCODIGO();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
                ContratistaCE objCE = new ContratistaCE();
                objCE.codigo = lblCodigo.Text;
                objCE.nombre = txtNombre.Text;
                objCE.paterno = txtPaterno.Text;
                objCE.materno = txtMaterno.Text;
                objCE.telefono = txtTelefono.Text;
                objCE.correo = txtCorreo.Text;
```

```
                try
                {
                    objCon.NUEVOCONTRATISTA(objCE);
                    generarCodigo();
                    MessageBox.Show("CONTRATISTA REGISTRADO CORRECTAMENTE..!!",
                                    "REGISTRO", MessageBoxButtons.OK,
                                    MessageBoxIcon.Information);
                }
                catch (Exception)
                {

                }
            }
            else
                MessageBox.Show("El error se encuentra en "+valida());
        }

        //Metodo de validación de datos
        string valida() {
            if (txtNombre.Text.Trim().Length == 0) {
                return " nombre del contratista";
            }else if (txtPaterno.Text.Trim().Length == 0) {
                return " paterno del contratista";
            }else if (txtMaterno.Text.Trim().Length == 0) {
                return " materno del contratista";
            }else if (txtTelefono.Text.Trim().Length == 0) {
                return " telefono del contratista";
            }
            else if (txtCorreo.Text.Trim().Length == 0)
            {
                return " correo del contratista";
            }
            else {
                return "";
            }
        }
    }
}
```

14. Observe el código del formulario frmActualizarContratista:

```
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;

//Referencia a la capa de negocio y entidad
using CapaNegocio;
```

```
using CapaEntidad;

namespace CapasPresentacion
{
    public partial class frmActualizarContratista : Form
    {
        //Objeto de la clase ContratistaCN
        ContratistaCN objCon = new ContratistaCN();

        public frmActualizarContratista()
        {
            InitializeComponent();
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
                ContratistaCE objCE = new ContratistaCE();
                objCE.codigo = txtCodigo.Text;
                objCE.nombre = txtNombre.Text;
                objCE.paterno = txtPaterno.Text;
                objCE.materno = txtMaterno.Text;
                objCE.telefono = txtTelefono.Text;
                objCE.correo = txtCorreo.Text;

                try
                {
                    objCon.ACTUALIZACONTRATISTA(objCE);
                    MessageBox.Show("CONTRATISTA ACTUALIZA CORRECTAMENTE..!!",
                                    "ACTUALIZA", MessageBoxButtons.OK,
                                    MessageBoxIcon.Information);
                    dgContratista.DataSource = objCon.LISTACONTRATISTA();
                }
                catch (Exception)
                {
                }
            }
            else
                MessageBox.Show("El error se encuentra en " + valida());
        }

        private void frmActualizarContratista_Load(object sender, EventArgs e)
        {
            dgContratista.DataSource = objCon.LISTACONTRATISTA();
        }

    private void dgContratista_MouseDoubleClick(object sender, MouseEventArgs e)
    {
       txtCodigo.Text = dgContratista.CurrentRow.Cells[0].Value.ToString();
       txtNombre.Text = dgContratista.CurrentRow.Cells[1].Value.ToString();
       txtPaterno.Text = dgContratista.CurrentRow.Cells[2].Value.ToString();
       txtMaterno.Text = dgContratista.CurrentRow.Cells[3].Value.ToString();
       txtTelefono.Text = dgContratista.CurrentRow.Cells[4].Value.ToString();
```

```
        txtCorreo.Text = dgContratista.CurrentRow.Cells[5].Value.ToString();
    }

    //Metodo de validacion de datos
    string valida()
    {
            if (txtNombre.Text.Trim().Length == 0)
            {
                return " nombre del contratista";
            }
            else if (txtPaterno.Text.Trim().Length == 0)
            {
                return " paterno del contratista";
            }
            else if (txtMaterno.Text.Trim().Length == 0)
            {
                return " materno del contratista";
            }
            else if (txtTelefono.Text.Trim().Length == 0)
            {
                return " telefono del contratista";
            }
            else if (txtCorreo.Text.Trim().Length == 0)
            {
                return " correo del contratista";
            }
            else
            {
                return "";
            }
    }
    private void btnCancelar_Click(object sender, EventArgs e)
    {
       this.Close();
    }

    //Método que permite limpiar los controles de texto del formulario
    public static void limpiar(Form ofrm)
    {
            foreach (Control oControls in ofrm.Controls)
            {
                if (oControls is TextBox)
                {
                    oControls.Text = "";
                }
            }
    }
  }
}
```

15. Observe el código del formulario frmBuscarContratista:

```
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;

//Referencia a la capa de negocio y entidad
using CapaNegocio;
using CapaEntidad;

namespace CapasPresentacion
{
    public partial class frmBuscarContratista : Form
    {
        //Objeto de la clase ContratistaCN
        ContratistaCN objCon = new ContratistaCN();

        public frmBuscarContratista()
        {
            InitializeComponent();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            ContratistaCE objC = new ContratistaCE();
            objC.codigo = txtCodigo.Text;

            if (valida() == "")
            {
                DataRow fila = objCon.BUSCACONTRATISTA(objC).Rows[0];
                txtNombre.Text = fila[1].ToString();
                txtPaterno.Text = fila[2].ToString();
                txtMaterno.Text = fila[3].ToString();
                txtTelefono.Text = fila[4].ToString();
                txtCorreo.Text = fila[5].ToString();
            }
            else
                MessageBox.Show("El error se encuentra en " + valida());
        }

        //Método que valida el ingreso del código de contratista
        string valida()
        {
            if (txtCodigo.Text.Trim().Length == 0)
            {
```

```
                return " Código del contratista";
            }
            else
            {
                return "";
            }
        }
    }
}
```

16. Observe el código del formulario frmEliminarContratista:

```
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;

//Referencia a la capa de negocio y entidad
using CapaEntidad;
using CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmEliminarContratista : Form
    {
        //Objeto de la clase ContratistaCN
        ContratistaCN objCon = new ContratistaCN();

        public frmEliminarContratista()
        {
            InitializeComponent();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            ContratistaCE objC = new ContratistaCE();
            objC.codigo = txtCodigo.Text;

            if (valida() == "")
            {
                DataRow fila = objCon.BUSCACONTRATISTA(objC).Rows[0];
                txtNombre.Text = fila[1].ToString();
                txtPaterno.Text = fila[2].ToString();
                txtMaterno.Text = fila[3].ToString();
                txtTelefono.Text = fila[4].ToString();
```

```
                txtCorreo.Text = fila[5].ToString();
            }
            else
                MessageBox.Show("El error se encuentra en " + valida());
        }

        //Método que valida el ingreso del código de contratista
        string valida()
        {
            if (txtCodigo.Text.Trim().Length == 0)
            {
                return " Código del contratista";
            }
            else
            {
                return "";
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            ContratistaCE objC = new ContratistaCE();
            objC.codigo = txtCodigo.Text;

            try
            {
                objCon.ELIMINACONTRATISTA(objC);
               MessageBox.Show("CONTRATISTA ELIMINADO CORRECTAMENTE..!!",
                                "ELIMINADO", MessageBoxButtons.OK,
                                 MessageBoxIcon.Information);
            }
            catch (Exception)
            {
            }
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        //Método que limpia los controles del formulario
        public static void limpiar(Form ofrm)
        {
            foreach (Control oControls in ofrm.Controls)
            {
                if (oControls is TextBox)
                {
                    oControls.Text = "";
                }
            }
        }
    }
}
```

17. Observe el código del formulario frmMenuPrincipal:

```
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 CapasPresentacion
{
    public partial class frmMenuPrincipal : Form
    {

        public frmMenuPrincipal()
        {
            InitializeComponent();
        }

       private void CloseAllToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (Form childForm in MdiChildren)
            {
                childForm.Close();
            }
        }

        private void actualizarContratistaToolStripMenuItem_Click(…)
        {
          frmActualizarContratista childForm = new frmActualizarContratista();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void buscarContratistaToolStripMenuItem_Click(…)
        {
            frmBuscarContratista childForm = new frmBuscarContratista
                ();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void eliminarContratistaToolStripMenuItem_Click(…)
        {
           frmEliminarContratista childForm = new frmEliminarContratista();
            childForm.MdiParent = this;
```

```
            childForm.Show();
        }

        private void nuevoContratistaToolStripMenuItem_Click(…)
        {
            frmNuevoContratista childForm = new frmNuevoContratista();
            childForm.MdiParent = this;
            childForm.Show();
        }

       private void salirToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void listaGeneralToolStripMenuItem_Click(…)
        {
            frmListaGeneral childForm = new frmListaGeneral();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void agendaContratistaToolStripMenuItem1_Click()
        {
            frmAgendaContratista childForm = new frmAgendaContratista();
            childForm.MdiParent = this;
            childForm.Show();
        }
    }
}
```

Caso desarrollado 2 : Mantenimiento de clientes con imágenes en N-Capas

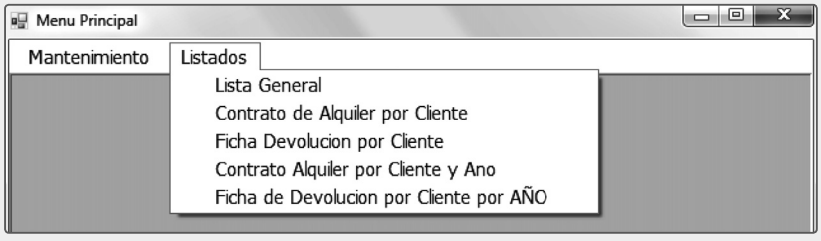
Implementar una solución que permite realizar el mantenimiento de registros de la tabla Cliente usando imágenes.

Se tiene en cuenta lo siguiente:

- Crear una solución en blanco.
- Implementar procedimientos almacenados para el proceso de mantenimiento de registros de los clientes con imágenes.
- Agregar tres proyectos de bibliotecas de clases y llámelos CapaDatos, CapaEntidad y CapaNegocio.
- Agregar un proyecto de Windows Forms y llámelo CapaPresentacion.

- Las opciones que presenta el formulario frmMenuPrincipal se muestran de la siguiente manera:
  - Menú de opciones del mantenimiento de registros.

  - Menú de opciones de las consultas realizadas a la tabla Contratista.



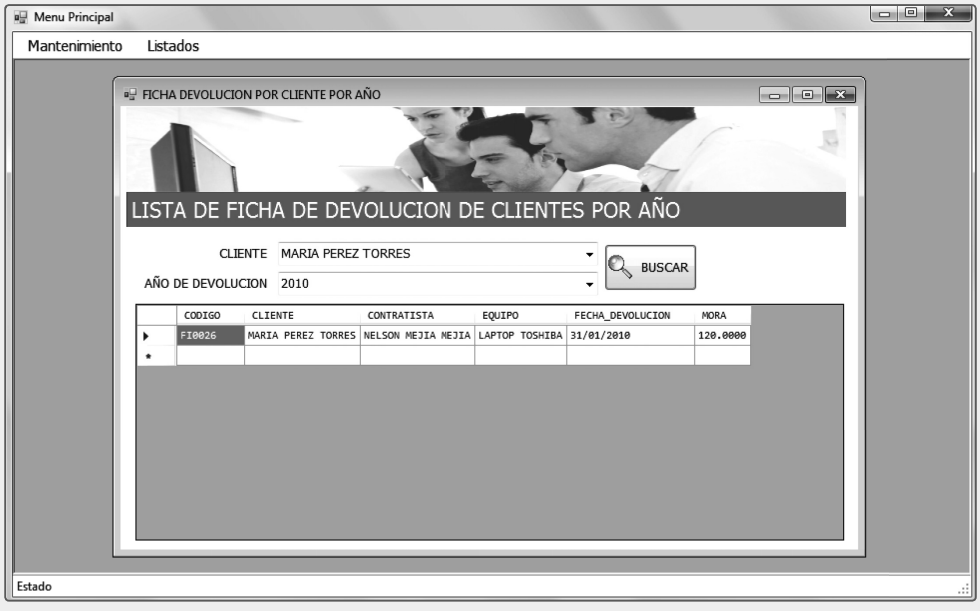
- Los formularios se deben presentar de la siguiente manera:
  - **Formulario de listado de clientes:** El cual presenta los datos de los clientes con sus respectivas imágenes.

- **Formulario de listado de contratos de alquiler por cliente:** Al seleccionar un determinado cliente, mostrará los contratos de alquiler asociados a este.

- **Formulario de listado de fichas de devolución por cliente:** En donde debe seleccionar un determinado cliente y se mostrará las fichas de devolución asociadas a este.

- **Formulario de listado de contratos por alquiler de cliente por año:** Al seleccionar un cliente y un año de fecha de inicio, se mostrarán los contratos asociados a ambos criterios.

- **Formulario de listado de fichas de devolución de clientes por año:** Al seleccionar un cliente en un determinado año, según la fecha de devolución, se mostrarán las fichas de devolución.



- **Formulario de registro del nuevo cliente:** Se inicia con un código autogenerado y se debe seleccionar una imagen, la cual corresponda al nuevo cliente mediante el botón **Examinar**.

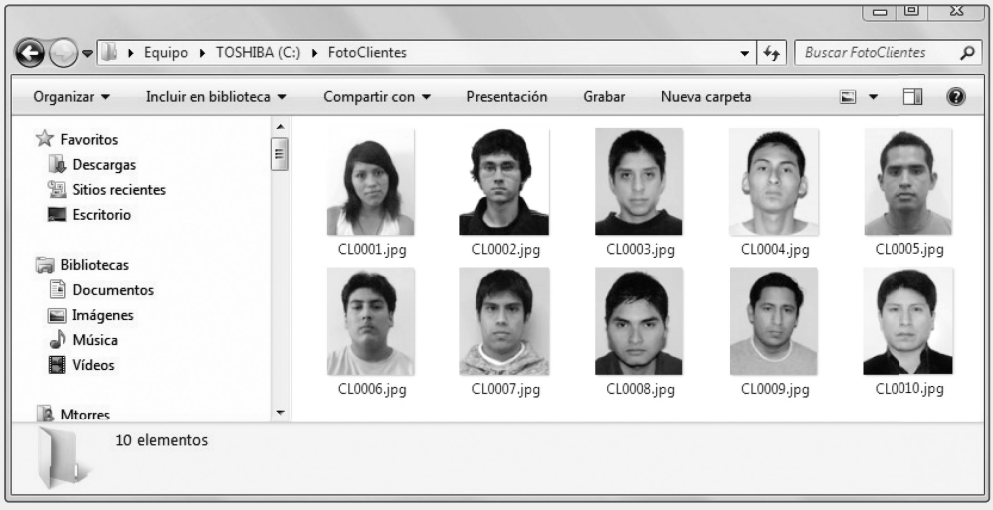
- **Formulario de actualización de datos del cliente:** Se inicia mostrando todos los datos de los clientes, que incluya sus imágenes. Luego, al presionar doble clic sobre uno de ellos, los datos se devuelven a los controles y se podrá realizar la modificación de sus datos.

- **Formulario de búsqueda de cliente:** Al colocar un código de cliente válido, se debe mostrar los datos de los clientes, que incluyan su imagen correspondiente.

- **Formulario de eliminación de un determinado cliente:** Para encontrar al cliente, debe buscarlo por su código. Una vez mostrada la información se procede a eliminar.

Solución:

1. Inicialmente, debe contar con una carpeta de imágenes de los clientes en la unidad C:, como se muestra en la siguiente imagen:



La base de datos contrato ha sufrido ciertos cambios con respecto a lo que se venía trabajando en todos los casos anteriores. Por lo tanto, se ha creado una base de datos paralela que contenga imágenes. El código es el siguiente:

```
-------------- BASE DE DATOS CONTRATO CON IMAGENES -------------------
SET DATEFORMAT DMY

---1. Verificando la existencia de la BD CONTRATO_IMA
IF DB_ID("CONTRATO_IMA") IS NOT NULL
BEGIN
        USE MASTER
        DROP DATABASE CONTRATO_IMA
END
GO

--2. Creando la BD CONTRATO_IMA en forma estandar
CREATE DATABASE CONTRATO_IMA
GO

--3. Activando la BD CONTRATO_IMA
USE CONTRATO_IMA
GO

--4. Creando la tabla cliente
CREATE TABLE CLIENTE
(
```

```
        IDE_CLI CHAR(6) NOT NULL,
        NOM_CLI VARCHAR(30)  NOT NULL,
        PAT_CLI VARCHAR(20)  NOT NULL,
        MAT_CLI VARCHAR(50)  NOT NULL,
        FON_CLI VARCHAR(15)  NOT NULL,
        DNI_CLI CHAR(8) NOT NULL,
        DIR_CLI VARCHAR(50)    NULL,
        FOT_CLI IMAGE          NULL
)
GO

--5. Implemenacion de las llaves primarias
ALTER TABLE CLIENTE
        ADD PRIMARY KEY(IDE_CLI)

--6. Insercion de registros
---- TABLA CLIENTE ----
INSERT INTO CLIENTE
        SELECT "CL0001","MARIA", "PEREZ", "TORRES", "5440855",
              "12345678", "AV.UNIVERSITARIA #582",
        BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0001.JPG",
                                                        SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
        SELECT "CL0002","JUNIOR", "FERNANDEZ", "LOPEZ", "5656236",
               "98652856", "AV.TUPAC AMARU #123",
        BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0002.JPG",
                                                        SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
        SELECT "CL0003","ALEX", "CARDENAS", "ZARRASCO", "1235821",
              "45698712", "AV.PIMENTEL #582",
        BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0003.JPG",
                                                        SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
        SELECT "CL0004","JUAN", "MEZQUITA", "ROSAS", "5023169",
              "14702589", "AV.OLIVOS #562",
        BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0004.JPG",
                                                        SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
        SELECT "CL0005","ERICK", "GONZALES", "SHULLER", "8520365",
               "85236974", "AV.GAMARRA #982",
        BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0005.JPG",
                                                        SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
        SELECT "CL0006","JOSE", "GALVEZ", "RETAMOZO", "9850352",
              "96785410", "AV.JOSE OLAYA #982",
        BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0006.JPG",
                                                        SINGLE_BLOB) AS CLI
```

```
GO
INSERT INTO CLIENTE
       SELECT "CL0007","CARLOS", "JARA", "SALAS", "1230124",
             "23605895", "AV.BRASIL #897",
       BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0007.JPG",
                                                    SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
       SELECT "CL0008","EDWARD", "RAMIREZ", "ROJAS", "5500002",
             "00213589", "AV.ARGENTINA #132",
       BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0008.JPG",
                                                    SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
       SELECT "CL0009","MANUEL", "GUIZADO", "GAMARRA", "5232155",
             "32569874", "AV.BOLIVIA #942",
       BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0009.JPG",
                                                    SINGLE_BLOB) AS CLI
GO
INSERT INTO CLIENTE
       SELECT "CL0010","ALBERTO", "FARFAN", "PRADO", "8567412",
             "98657452", "AV.CHILE #528",
       BULKCOLUMN FROM OPENROWSET (BULK "C:\fotoclientes\CL0010.JPG",
                                                    SINGLE_BLOB) AS CLI
GO

--7. Listado de Registros
SELECT * FROM CLIENTE
```

2. Ejecute los siguientes procedimientos almacenados en SQL Server.

```
--PROCEDIMIENTO ALMACENADO QUE REGISTRA UN NUEVO CLIENTE
IF OBJECT_ID("SP_NUEVOCLIENTE")IS NOT NULL
       DROP PROC SP_NUEVOCLIENTE
GO
CREATE PROC SP_NUEVOCLIENTE(@COD CHAR(6),@NOM VARCHAR(30),@PAT VARCHAR(20),
                            @MAT VARCHAR(50),@FON VARCHAR(15),@DNI CHAR(8),
                            @DIR VARCHAR(50),@FOT IMAGE)
AS
       INSERT INTO CLIENTE
              VALUES (@COD,@NOM,@PAT,@MAT,@FON,@DNI,@DIR,@FOT)
GO

--PROCEDIMIENTO ALMACENADO QUE MUESTRA EL ULTIMO CODIGO
IF OBJECT_ID("SP_ULTIMOCLIENTE")IS NOT NULL
       DROP PROC SP_ULTIMOCLIENTE
GO
CREATE PROC SP_ULTIMOCLIENTE
AS
```

```
        SELECT TOP 1 C.IDE_CLI
               FROM CLIENTE C
               ORDER BY C.IDE_CLI DESC
GO

--PROCEDIMIENTO ALMACENADO QUE ACTUALIZA LOS DATOS DEL CLIENTE
IF OBJECT_ID("SP_ACTUALIZACLIENTE")IS NOT NULL
        DROP PROC SP_ACTUALIZACLIENTE
GO
CREATE PROC SP_ACTUALIZACLIENTE(@COD CHAR(6),@NOM VARCHAR(30),
                                @PAT VARCHAR(20), @MAT VARCHAR(50),
                                @FON VARCHAR(15),@DNI CHAR(8),
                                     @DIR VARCHAR(50),@FOT IMAGE)
AS
        UPDATE CLIENTE
               SET NOM_CLI=@NOM,PAT_CLI=@PAT,MAT_CLI=@MAT,
               FON_CLI=@FON,DNI_CLI=@DNI,DIR_CLI=@DIR,
               FOT_CLI=@FOT
               WHERE IDE_CLI=@COD
GO

--PROCEDIMIENTO ALMACENADO QUE ELIMINA UN REGISTRO DE CLIENTE
IF OBJECT_ID("SP_ELIMINACLIENTE")IS NOT NULL
        DROP PROC SP_ELIMINACLIENTE
GO
CREATE PROC SP_ELIMINACLIENTE(@COD CHAR(6))
AS
        DELETE CLIENTE
               WHERE IDE_CLI=@COD
GO

--PROCEDIMIENTO ALMACENADO QUE BUSCA LOS DATOS DE UN CLIENTE
IF OBJECT_ID("SP_BUSCACLIENTE")IS NOT NULL
        DROP PROC SP_BUSCACLIENTE
GO
CREATE PROC SP_BUSCACLIENTE(@COD CHAR(6))
AS
        SELECT C.IDE_CLI AS CODIGO,
               C.NOM_CLI AS NOMBRES,
               C.PAT_CLI AS PATERNO,
               C.MAT_CLI AS MATERNO,
               C.FON_CLI AS FONO,
               C.DNI_CLI AS DNI,
               C.DIR_CLI AS DIRECCION,
               C.FOT_CLI AS FOTO
               FROM CLIENTE C
               WHERE C.IDE_CLI=@COD
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS CLIENTES
IF OBJECT_ID("SP_LISTAGENERAL")IS NOT NULL
        DROP PROC SP_LISTAGENERAL
```

```
GO
CREATE PROC SP_LISTAGENERAL
AS
        SELECT C.IDE_CLI AS CODIGO,
        C.NOM_CLI + SPACE(1)+ C.PAT_CLI + SPACE(1)+ C.MAT_CLI AS CLIENTE,
        C.FON_CLI AS FONO,
        C.DNI_CLI AS DNI,
        C.DIR_CLI AS DIRECCION,
        C.FOT_CLI AS FOTO
        FROM CLIENTE C
GO

--LISTADO DE CONTRATOS DE ALQUILER SEGÚN EL CÓDIGO DE CLIENTE
IF OBJECT_ID("CONTRATOALQUILERXCLIENTE")IS NOT NULL
        DROP PROC CONTRATOALQUILERXCLIENTE
GO
CREATE PROC CONTRATOALQUILERXCLIENTE(@CLI CHAR(6))
AS
        SELECT CA.COD_CON AS CODIGO,
        C.NOM_CLI + SPACE(1)+ C.PAT_CLI + SPACE(1)+ C.MAT_CLI AS CLIENTE,
        CO.NOM_CON + SPACE(1)+ CO.PAT_CON + SPACE(1)+ CO.MAT_CON AS CONTRATSTA,
        FIN_CON AS FECHA_INICIO,
        FFI_CON AS FECHA_FIN,
        TIP_CON AS TIPO
        FROM CONTRATOALQUILER CA
        JOIN CLIENTE C ON CA.IDE_CLI=C.IDE_CLI
        JOIN CONTRATISTA CO ON CA.IDE_CON=CO.IDE_CON
        WHERE C.IDE_CLI = @CLI
GO
--PRUEBA: CONTRATOALQUILERXCLIENTE "CL0001"

--LISTADO DE CLIENTE PARA EL CUADRO COMBINADO
IF OBJECT_ID("SP_LISTACLIENTE")IS NOT NULL
        DROP PROC SP_LISTACLIENTE
GO
CREATE PROC SP_LISTACLIENTE
AS
        SELECT C.IDE_CLI AS CODIGO,
                C.NOM_CLI + SPACE(1)+ C.PAT_CLI + SPACE(1)+ C.MAT_CLI AS
CLIENTE
                FROM CLIENTE C
GO

--PROCEDIMIENTO ALMACENADO QUE LISTA LOS DATOS ORIGINALES DEL CLIENTE
IF OBJECT_ID("SP_LISTACLIENTEORIGINAL")IS NOT NULL
        DROP PROC SP_LISTACLIENTEORIGINAL
GO
CREATE PROC SP_LISTACLIENTEORIGINAL
AS
        SELECT *
                FROM CLIENTE C
GO
```

```
--LISTADO DE FICHAS DE DEVOLUCION POR CÓDIGO DE CLIENTE
IF OBJECT_ID("FICHADEVOLUCIONXCLIENTE")IS NOT NULL
        DROP PROC FICHADEVOLUCIONXCLIENTE
GO
CREATE PROC FICHADEVOLUCIONXCLIENTE(@CLI CHAR(6))
AS
        SELECT FD.COD_FIC AS CODIGO,
        C.NOM_CLI + SPACE(1)+ C.PAT_CLI + SPACE(1)+ C.MAT_CLI AS CLIENTE,
        CO.NOM_CON + SPACE(1)+ CO.PAT_CON + SPACE(1)+ CO.MAT_CON AS CONTRATISTA,
        E.DESC_EQU AS EQUIPO,
        FD.FDE_FIC AS FECHA_DEVOLUCION,
        MOR_FIC AS MORA
        FROM FICHA_DEVOLUCION FD
        JOIN CLIENTE C ON FD.IDE_CLI=C.IDE_CLI
        JOIN CONTRATISTA CO ON FD.IDE_CON=CO.IDE_CON
        JOIN EQUIPO E ON E.IDE_EQU=FD.IDE_EQU
        WHERE C.IDE_CLI = @CLI
GO

--LISTADO DE AÑOS QUE SE MUESTRA EN EL CUADRO COMBINADO
IF OBJECT_ID("SP_LISTAAÑOSINICIO")IS NOT NULL
        DROP PROC SP_LISTAAÑOSINICIO
GO
CREATE PROC SP_LISTAAÑOSINICIO
AS
        SELECT DISTINCT YEAR(CA.FIN_CON) AS AÑO
                FROM CONTRATOALQUILER CA
GO

--LISTADO DE CONTRATOS DE ALQUILER SEGUN EL CODIGO CLIENTE Y UN AÑO
IF OBJECT_ID("CONTRATOALQUILERXCLIENTEXAÑO")IS NOT NULL
        DROP PROC CONTRATOALQUILERXCLIENTEXAÑO
GO
CREATE PROC CONTRATOALQUILERXCLIENTEXAÑO(@CLI CHAR(6),@AÑO INT)
AS
        SELECT CA.COD_CON AS CODIGO,
        C.NOM_CLI + SPACE(1)+ C.PAT_CLI + SPACE(1)+ C.MAT_CLI AS CLIENTE,
        CO.NOM_CON + SPACE(1)+ CO.PAT_CON + SPACE(1)+ CO.MAT_CON AS CONTRATISTA,
        FIN_CON AS FECHA_INICIO,
        FFI_CON AS FECHA_FIN,
        TIP_CON AS TIPO
        FROM CONTRATOALQUILER CA
        JOIN CLIENTE C ON CA.IDE_CLI=C.IDE_CLI
        JOIN CONTRATISTA CO ON CA.IDE_CON=CO.IDE_CON
        WHERE C.IDE_CLI = @CLI AND YEAR(CA.FIN_CON) = @AÑO
GO
--PRUEBA: CONTRATOALQUILERXCLIENTEXAÑO "CL0001",2010

--LISTADO DE FICHAS DEVOLUCION SEGUN EL CODIGO DE CLIENTE Y UN AÑO
IF OBJECT_ID("FICHADEVOLUCIONXCLIENTEXAÑO")IS NOT NULL
        DROP PROC FICHADEVOLUCIONXCLIENTEXAÑO
```

```
GO
CREATE PROC FICHADEVOLUCIONXCLIENTEXAÑO(@CLI CHAR(6),@AÑO INT)
AS
        SELECT FD.COD_FIC AS CODIGO,
        C.NOM_CLI + SPACE(1)+ C.PAT_CLI + SPACE(1)+ C.MAT_CLI AS CLIENTE,
        CO.NOM_CON + SPACE(1)+ CO.PAT_CON + SPACE(1)+ CO.MAT_CON AS CONTRATISTA,
        E.DESC_EQU AS EQUIPO,
        FD.FDE_FIC AS FECHA_DEVOLUCION,
        MOR_FIC AS MORA
        FROM FICHA_DEVOLUCION FD
        JOIN CLIENTE C ON FD.IDE_CLI=C.IDE_CLI
        JOIN CONTRATISTA CO ON FD.IDE_CON=CO.IDE_CON
        JOIN EQUIPO E ON E.IDE_EQU=FD.IDE_EQU
        WHERE C.IDE_CLI = @CLI AND YEAR(FD.FDE_FIC) = @AÑO
GO
--PRUEBA: FICHADEVOLUCIONXCLIENTEXAÑO "CL0001",2009
```

3. Cree una solución en blanco en Visual Studio 2015 y agregue tres proyectos del tipo biblioteca de clase a la solución y asigne los nombres CapaDatos, CapaEntidad y CapaNegocios.
4. Agregue a la solución un proyecto de tipo Windows Forms y asígnele el nombre capaPresentacion.
5. Elimine los archivos Class1.cs de todas las capas.
6. A la capa de datos, agregue la referencia System.Configuration.
7. Desde la capa de presentación, modifique el archivo app.config, de tal forma que permita conectarse a la base de datos CONTRATO_IMA.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <startup>
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
  <connectionStrings>
    <add name="cn" connectionString="SERVER=.;DATABASE=CONTRATO_IMA;
                                     INTEGRATED SECURITY=SSPI"/>
  </connectionStrings>
</configuration>
```

8. Agregue la clase ClienteCE a la capa entidad, esta tendrá la misión de especificar todos los métodos get y set que componen la clase Cliente:

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

namespace CapaEntidad
{
   public class ClienteCE
    {
        public string codigo{ get; set; }
        public string nombre { get; set; }
        public string paterno { get; set; }
        public string materno { get; set; }
        public string telefono { get; set; }
        public string dni { get; set; }
        public string direccion { get; set; }
        public object foto { get; set; }
    }
}
```

9. Agregue la clase Conexión a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Configuration;

namespace CapaDatos
{
  public  class Conexion
    {
      public SqlConnection getConecta()
      {
          SqlConnection cn = new SqlConnection(ConfigurationManager
                                  .ConnectionStrings["cn"].ConnectionString);
          return cn;

      }
    }
}
```

10. Agregue la clase ClienteDAO a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Data;
using CapaEntidad;

namespace CapaDatos
{
    public class ClienteDAO
    {
        SqlConnection cn;
        Conexion objC = new Conexion();

        public void NUEVOCLIENTE(ClienteCE c)
        {
            cn = objC.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "SP_NUEVOCLIENTE";
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Connection = cn;

            cmd.Parameters.Add("@COD", SqlDbType.Char).Value = c.codigo;
          cmd.Parameters.Add("@NOM", SqlDbType.VarChar).Value = c.nombre;
           cmd.Parameters.Add("@PAT", SqlDbType.VarChar).Value = c.paterno;
           cmd.Parameters.Add("@MAT", SqlDbType.VarChar).Value = c.materno;
          cmd.Parameters.Add("@FON", SqlDbType.VarChar).Value = c.telefono;
            cmd.Parameters.Add("@DNI", SqlDbType.VarChar).Value = c.dni;
         cmd.Parameters.Add("@DIR", SqlDbType.VarChar).Value = c.direccion;
            cmd.Parameters.Add("@FOT", SqlDbType.Binary).Value = c.foto;

            try
            {
                cmd.ExecuteNonQuery();
            }
            catch (Exception) { }
        }

        public string NuevoCodigo()
        {
            cn = objC.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand("SP_ULTIMOCLIENTE", cn);
            SqlDataAdapter da = new SqlDataAdapter();
            da.SelectCommand = cmd;
```

```
            string NuevoCodigo = "CL" + (int.Parse(cmd.ExecuteScalar()
                        .ToString().Substring(2, 4)) + 1).ToString("0000");
            return NuevoCodigo;
        }

        public void ACTUALIZACLIENTE(ClienteCE c)
        {
            cn = objC.getConecta();
            cn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.CommandText = "SP_ACTUALIZACLIENTE";
            cmd.CommandType = CommandType.StoredProcedure;
            cmd.Connection = cn;

            cmd.Parameters.Add("@COD", SqlDbType.Char).Value = c.codigo;
            cmd.Parameters.Add("@NOM", SqlDbType.VarChar).Value = c.nombre;
            cmd.Parameters.Add("@PAT", SqlDbType.VarChar).Value = c.paterno;
            cmd.Parameters.Add("@MAT", SqlDbType.VarChar).Value = c.materno;
            cmd.Parameters.Add("@FON", SqlDbType.VarChar).Value = c.telefono;
            cmd.Parameters.Add("@DNI", SqlDbType.VarChar).Value = c.dni;
            cmd.Parameters.Add("@DIR", SqlDbType.VarChar).Value = c.direccion;
            cmd.Parameters.Add("@FOT", SqlDbType.Binary).Value = c.foto;

            try
            {
                cmd.ExecuteNonQuery();
            }
            catch (Exception) { }
        }

        public void ELIMINACLIENTE(ClienteCE c)
        {
          cn = objC.getConecta();
          cn.Open();
          SqlCommand cmd = new SqlCommand();
          cmd.CommandText = "SP_ELIMINACLIENTE";
          cmd.CommandType = CommandType.StoredProcedure;
          cmd.Connection = cn;

          cmd.Parameters.Add("@COD", SqlDbType.Char).Value = c.codigo;
          try
          {
              cmd.ExecuteNonQuery();
          }
          catch (Exception) { }
        }

    public DataTable BUSCACLIENTE(ClienteCE c)
    {
        cn = objC.getConecta();
```

```
        SqlDataAdapter da = new SqlDataAdapter("SP_BUSCACLIENTE", cn);
        da.SelectCommand.CommandType = CommandType.StoredProcedure;
       da.SelectCommand.Parameters.Add("@COD", SqlDbType.Char).Value = c.codigo;
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
    }

    public DataTable LISTAGENERAL()
    {
            cn = objC.getConecta();
           SqlDataAdapter da = new SqlDataAdapter("SP_LISTAGENERAL", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
    }

    public DataTable CONTRATOALQUILERXCLIENTE(string cliente)
    {
       cn = objC.getConecta();
       SqlDataAdapter da = new SqlDataAdapter("CONTRATOALQUILERXCLIENTE", cn);
       da.SelectCommand.CommandType = CommandType.StoredProcedure;
       da.SelectCommand.Parameters.Add("@CLI", SqlDbType.Char).Value = cliente;
       DataTable dt = new DataTable();
       da.Fill(dt);
       return dt;
    }

    public DataTable LISTACLIENTE()
    {
       cn = objC.getConecta();
       SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTE", cn);
       DataTable dt = new DataTable();
       da.Fill(dt);
       return dt;
    }

    public DataTable LISTACLIENTEORIGINAL()
    {
       cn = objC.getConecta();
       SqlDataAdapter da = new SqlDataAdapter("SP_LISTACLIENTEORIGINAL", cn);
       DataTable dt = new DataTable();
       da.Fill(dt);
       return dt;
   }

   public DataTable FICHADEVOLUCIONXCLIENTE(string cliente)
   {
       cn = objC.getConecta();
       SqlDataAdapter da = new SqlDataAdapter("FICHADEVOLUCIONXCLIENTE", cn);
```

```
        da.SelectCommand.CommandType = CommandType.StoredProcedure;
        da.SelectCommand.Parameters.Add("@CLI", SqlDbType.Char).Value = cliente;
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
    }

    public DataTable LISTAAÑOSINICIO()
    {
        cn = objC.getConecta();
        SqlDataAdapter da = new SqlDataAdapter("SP_LISTAAÑOSINICIO", cn);
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
    }

    public DataTable CONTRATOALQUILERXCLIENTEXAÑO(string cliente, int AÑO)
    {
       cn = objC.getConecta();
      SqlDataAdapter da = new SqlDataAdapter("CONTRATOALQUILERXCLIENTEXAÑO", cn);
       da.SelectCommand.CommandType = CommandType.StoredProcedure;
      da.SelectCommand.Parameters.Add("@CLI", SqlDbType.Char).Value = cliente;
     da.SelectCommand.Parameters.Add("@AÑO", SqlDbType.Char).Value = AÑO;
       DataTable dt = new DataTable();
       da.Fill(dt);
       return dt;
    }

    public DataTable FICHADEVOLUCIONXCLIENTEXAÑO(string cliente, int AÑO)
    {
       cn = objC.getConecta();
     SqlDataAdapter da = new SqlDataAdapter("FICHADEVOLUCIONXCLIENTEXAÑO", cn);
       da.SelectCommand.CommandType = CommandType.StoredProcedure;
      da.SelectCommand.Parameters.Add("@CLI", SqlDbType.Char).Value = cliente;
     da.SelectCommand.Parameters.Add("@AÑO", SqlDbType.Char).Value = AÑO;
       DataTable dt = new DataTable();
       da.Fill(dt);
       return dt;
    }
  }
}
```

11. Agregue la clase ClienteCN a la capa de negocio y coloque el siguiente código:

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

using CapaDatos;
using CapaEntidad;

namespace CapaNegocio
{
   public class ClienteCN
    {
       ClienteDAO objDAO = new ClienteDAO();

       public DataTable LISTAGENERAL()
       {
           return objDAO.LISTAGENERAL();
       }
       public string NUEVOCODIGO()
       {
           return objDAO.NuevoCodigo();
       }
       public void NUEVOCLIENTE(ClienteCE c)
       {
           objDAO.NUEVOCLIENTE(c);
       }
       public void ACTUALIZACLIENTE(ClienteCE c)
       {
           objDAO.ACTUALIZACLIENTE(c);
       }
       public void ELIMINACLIENTE(ClienteCE c)
       {
           objDAO.ELIMINACLIENTE(c);
       }
       public DataTable BUSCACLIENTE(ClienteCE c)
       {
           return objDAO.BUSCACLIENTE(c);
       }
       public DataTable CONTRATOALQUILERXCLIENTE(string cliente)
       {
           return objDAO.CONTRATOALQUILERXCLIENTE(cliente);
       }
       public DataTable LISTACLIENTE()
       {
           return objDAO.LISTACLIENTE();
       }
       public DataTable FICHADEVOLUCIONXCLIENTE(string cliente)
```

```
        {
            return objDAO.FICHADEVOLUCIONXCLIENTE(cliente);
        }
        public DataTable LISTAAÑOSINICIO()
        {
            return objDAO.LISTAAÑOSINICIO();
        }
        public DataTable CONTRATOALQUILERXCLIENTEXAÑO(string cliente, int AÑO)
        {
            return objDAO.CONTRATOALQUILERXCLIENTEXAÑO(cliente, AÑO);
        }
        public DataTable FICHADEVOLUCIONXCLIENTEXAÑO(string cliente, int AÑO)
        {
            return objDAO.FICHADEVOLUCIONXCLIENTEXAÑO(cliente, AÑO);
        }
        public DataTable LISTACLIENTEORIGINAL()
        {
            return objDAO.LISTACLIENTEORIGINAL();
        }
    }
}
```

12. Observe el código del formulario frmListaGeneral:

```
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 CapaNegocio;
namespace CapasPresentacion
{
    public partial class frmListaGeneral : Form
    {
        ClienteCN objCon = new ClienteCN();

        public frmListaGeneral()
        {
            InitializeComponent();
        }

        private void frmListaGeneral_Load(object sender, EventArgs e)
        {
            dgCliente.DataSource = objCon.LISTAGENERAL();
        }
    }
}
```

13. Observe el código del formulario frmContratoAlquilerxCliente:

```
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 CapaEntidad;
using CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmContratoAlquilerXCliente : Form
    {
        ClienteCN objCon = new ClienteCN();

        public frmContratoAlquilerXCliente()
        {
            InitializeComponent();
        }

       private void frmContratoAlquilerXCliente_Load(object sender, EventArgs e)
        {
            llenarCliente();
        }
        void llenarCliente()
        {
            cboCliente.DataSource = objCon.LISTACLIENTE();
            cboCliente.DisplayMember = "cliente";
            cboCliente.ValueMember = "codigo";
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
          dgCliente.DataSource = objCon.CONTRATOALQUILERXCLIENTE(cboCliente.
                                                    SelectedValue.ToString());
        }
    }
}
```

14. Observe el código del formulario frmFichaDevolucionxCliente:

```
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 CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmFichaDevolucionxCliente : Form
    {
        ClienteCN objCon = new ClienteCN();

        public frmFichaDevolucionxCliente()
        {
            InitializeComponent();
        }

       private void frmFichaDevolucionxCliente_Load(object sender, EventArgs e)
        {
            llenarCliente();
        }
        void llenarCliente()
        {
            cboCliente.DataSource = objCon.LISTACLIENTE();
            cboCliente.DisplayMember = "cliente";
            cboCliente.ValueMember = "codigo";
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
          dgCliente.DataSource = objCon.FICHADEVOLUCIONXCLIENTE(cboCliente.
                                                  SelectedValue.ToString());
        }
    }
}
```

15. Observe el código del formulario frmContratoAlquilerxClientexAño:

```
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 CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmContratoAlquilerXClienteXAÑO : Form
    {
        ClienteCN objCon = new ClienteCN();
        public frmContratoAlquilerXClienteXAÑO()
        {
            InitializeComponent();
        }

        private void frmContratoAlquilerXClienteXAÑO_Load()
        {
            llenarCliente();
            llenarAÑOS();
        }
        void llenarCliente()
        {
            cboCliente.DataSource = objCon.LISTACLIENTE();
            cboCliente.DisplayMember = "cliente";
            cboCliente.ValueMember = "codigo";
        }
        void llenarAÑOS()
        {
            cboAÑO.DataSource = objCon.LISTAAÑOSINICIO();
            cboAÑO.DisplayMember = "AÑO";
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            dgCliente.DataSource = objCon.CONTRATOALQUILERXCLIENTEXAÑO(
                 cboCliente.SelectedValue.ToString(),int.Parse(cboAÑO.Text));
        }
    }
}
```

16. Observe el código del formulario frmFichaDevolucionxClientexAño:

```
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 CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmFichaDevolucionxClientxAÑO : Form
    {
        ClienteCN objCon = new ClienteCN();
        public frmFichaDevolucionxClientxAÑO()
        {
            InitializeComponent();
        }

        private void frmFichaDevolucionxClientxAÑO_Load(…)
        {
            llenarCliente();
            llenarAÑOS();
        }
        void llenarCliente()
        {
            cboCliente.DataSource = objCon.LISTACLIENTE();
            cboCliente.DisplayMember = "cliente";
            cboCliente.ValueMember = "codigo";
        }
        void llenarAÑOS()
        {
            cboAÑO.DataSource = objCon.LISTAAÑOSINICIO();
            cboAÑO.DisplayMember = "AÑO";
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            dgCliente.DataSource = objCon.FICHADEVOLUCIONXCLIENTEXAÑO(
                cboCliente.SelectedValue.ToString(), int.Parse(cboAÑO.Text));
        }

    }
}
```

17. Observe el código del formulario frmNuevoCliente:

```
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 System.IO;
using CapaEntidad;
using CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmNuevoCliente : Form
    {
        ClienteCN objC = new ClienteCN();
        OpenFileDialog OP = new OpenFileDialog();

        public frmNuevoCliente()
        {
            InitializeComponent();
        }

        private void frmNuevoCliente_Load(object sender, EventArgs e)
        {
            generarCodigo();
        }

        //Metodo que genera un nuevo codigo de cliente
        void generarCodigo()
        {
            lblCodigo.Text = objC.NUEVOCODIGO();
        }

        private void btnRegistrar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
              ClienteCE objCE = new ClienteCE();
                objCE.codigo = lblCodigo.Text;
                objCE.nombre = txtNombre.Text;
                objCE.paterno = txtPaterno.Text;
                objCE.materno = txtMaterno.Text;
                objCE.telefono = txtTelefono.Text;
                objCE.dni = txtDni.Text;
                objCE.direccion = txtDireccion.Text;
```

```
            PictureBox imagen = pbImagen;
            MemoryStream ms = new MemoryStream();
        imagen.Image.Save(ms,System.Drawing.Imaging.ImageFormat.Jpeg);

            objCE.foto = ms.GetBuffer();
            try
            {
                objC.NUEVOCLIENTE(objCE);
                generarCodigo();
              MessageBox.Show("CLIENTE REGISTRADO CORRECTAMENTE..!!",
                                "REGISTRO", MessageBoxButtons.OK,
                                 MessageBoxIcon.Information);
                limpiar(this);
                txtNombre.Focus();
            }
            catch (Exception) { }
        }
        else
            MessageBox.Show("El error se encuentra en " + valida());
    }

    //Metodo de validacion de datos
    string valida() {
        if (txtNombre.Text.Trim().Length == 0) {
            return " nombre del cliente";
        }else if (txtPaterno.Text.Trim().Length == 0) {
            return " paterno del cliente";
        }else if (txtMaterno.Text.Trim().Length == 0) {
            return " materno del cliente";
        }else if (txtTelefono.Text.Trim().Length == 0) {
            return " telefono del cliente";
        }
        else if (txtDni.Text.Trim().Length == 0)
        {
            return " DNI del cliente";
        }
        else if (txtDireccion.Text.Trim().Length == 0)
        {
            return " direccion del cliente";
        }
        else {
            return "";
        }
    }

    private void btnExaminar_Click(object sender, EventArgs e)
    {
        OP.Filter = "Archivo jpg|*.jpg";
        if(OP.ShowDialog()==DialogResult.OK)
        {
            pbImagen.Image = Image.FromFile(OP.FileName);
        }
```

```
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        //Metodo que limpia los controles del formulario
        public static void limpiar(Form ofrm)
        {
            foreach (Control oControls in ofrm.Controls)
            {
                if (oControls is TextBox)
                {
                    oControls.Text = "";
                }
            }
        }
    }
}
```

18. Observe el código del formulario frmActualizaCliente:

```
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 System.IO;
using CapaNegocio;
using CapaEntidad;

namespace CapasPresentacion
{
    public partial class frmActualizaCliente : Form
    {
        ClienteCN objC = new ClienteCN();
        OpenFileDialog OP = new OpenFileDialog();

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

```
        private void frmActualizaCliente_Load(object sender, EventArgs e)
        {
            dgCliente.DataSource = objC.LISTACLIENTEORIGINAL();
        }

        private void btnActualizar_Click(object sender, EventArgs e)
        {
            if (valida() == "")
            {
                ClienteCE objCE = new ClienteCE();
                objCE.codigo = txtCodigo.Text;
                objCE.nombre = txtNombre.Text;
                objCE.paterno = txtPaterno.Text;
                objCE.materno = txtMaterno.Text;
                objCE.telefono = txtTelefono.Text;
                objCE.dni = txtDni.Text;
                objCE.direccion = txtDireccion.Text;

                PictureBox imagen = pbImagen;
                MemoryStream ms = new MemoryStream();
            imagen.Image.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);

                objCE.foto = ms.GetBuffer();
                try
                {
                    objC.ACTUALIZACLIENTE(objCE);
                   MessageBox.Show("CLIENTE ACTUALIZA CORRECTAMENTE..!!",
                                    "ACTUALIZA", MessageBoxButtons.OK,
                                    MessageBoxIcon.Information);
                    dgCliente.DataSource = objC.LISTACLIENTEORIGINAL();
                    limpiar(this);
                    pbImagen.Image = null;
                    txtCodigo.Focus();
                }
                catch (Exception) { }
            }
            else
                MessageBox.Show("El error se encuentra en " + valida());
        }

        //Metodo de validacion de datos
        string valida()
        {
            if (txtNombre.Text.Trim().Length == 0)
            {
                return " nombre del cliente";
            }
            else if (txtPaterno.Text.Trim().Length == 0)
            {
                return " paterno del cliente";
```

```
            }
            else if (txtMaterno.Text.Trim().Length == 0)
            {
                return " materno del cliente";
            }
            else if (txtTelefono.Text.Trim().Length == 0)
            {
                return " telefono del cliente";
            }
            else if (txtDni.Text.Trim().Length == 0)
            {
                return " DNI del cliente";
            }
            else if (txtDireccion.Text.Trim().Length == 0)
            {
                return " direccion del cliente";
            }
            else
            {
                return "";
            }
        }

        private void btnExaminar_Click(object sender, EventArgs e)
        {
            OP.Filter = "Archivo jpg|*.jpg";
            if (OP.ShowDialog() == DialogResult.OK)
            {
                pbImagen.Image = Image.FromFile(OP.FileName);
            }
        }

        //Metodo para limpiar los controles del formulario
        public static void limpiar(Form ofrm)
        {
            foreach (Control oControls in ofrm.Controls)
            {
                if (oControls is TextBox)
                {
                    oControls.Text = "";
                }
            }
        }

      private void dgCliente_MouseDoubleClick(object sender, MouseEventArgs e)
        {
           txtCodigo.Text = dgCliente.CurrentRow.Cells[0].Value.ToString();
           txtNombre.Text = dgCliente.CurrentRow.Cells[1].Value.ToString();
           txtPaterno.Text = dgCliente.CurrentRow.Cells[2].Value.ToString();
```

```
            txtMaterno.Text = dgCliente.CurrentRow.Cells[3].Value.ToString();
            txtTelefono.Text = dgCliente.CurrentRow.Cells[4].Value.ToString();
            txtDni.Text = dgCliente.CurrentRow.Cells[5].Value.ToString();
           txtDireccion.Text = dgCliente.CurrentRow.Cells[6].Value.ToString();

            //Recuperando la imagen del cliente
            byte[] datos = new byte[0];
            datos = (byte[])dgCliente.CurrentRow.Cells[7].Value;

            //Mostrando la imagen del cliente
            MemoryStream ms = new MemoryStream(datos);
            Image imagen = Image.FromStream(ms);
            pbImagen.Image = imagen;
        }
    }
}
```

19. Observe el código del formulario frmBuscaCliente:

```
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 System.IO;
using CapaNegocio;
using CapaEntidad;

namespace CapasPresentacion
{
    public partial class frmBuscarCliente : Form
    {
        ClienteCN objCon = new ClienteCN();
        OpenFileDialog OP = new OpenFileDialog();
        public frmBuscarCliente()
        {
            InitializeComponent();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            ClienteCE objCl = new ClienteCE();
            objCl.codigo = txtCodigo.Text;
```

```
            if (valida() == "")
            {
                DataRow fila = objCon.BUSCACLIENTE(objCl).Rows[0];
                txtNombre.Text = fila[1].ToString();
                txtPaterno.Text = fila[2].ToString();
                txtMaterno.Text = fila[3].ToString();
                txtTelefono.Text = fila[4].ToString();
                txtDni.Text = fila[5].ToString();
                txtDireccion.Text = fila[6].ToString();

                byte[] datos = new byte[0];
                datos = (byte[])fila[7];

                MemoryStream ms = new MemoryStream(datos);
                Image imagen = Image.FromStream(ms);
                pbImagen.Image = imagen;
            }
            else
                MessageBox.Show("El error se encuentra en " + valida());
        }

        //Metodo de validacion de datos
        string valida()
        {
            if (txtCodigo.Text.Trim().Length == 0)
            {
                return " Codigo del contratista";
            }
            else
            {
                return "";
            }
        }

        private void btnExaminar_Click(object sender, EventArgs e)
        {
             OP.Filter = "Archivo jpg|*.jpg";
            if (OP.ShowDialog() == DialogResult.OK)
            {
                pbImagen.Image = Image.FromFile(OP.FileName);
            }
        }
    }
}
```

20. Observe el código del formulario frmEliminarCliente:

```
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 System.IO;
using CapaEntidad;
using CapaNegocio;

namespace CapasPresentacion
{
    public partial class frmEliminarCliente : Form
    {
        ClienteCN objCon = new ClienteCN();
        OpenFileDialog OP = new OpenFileDialog();

        public frmEliminarCliente()
        {
            InitializeComponent();
        }

        private void btnBuscar_Click(object sender, EventArgs e)
        {
            ClienteCE objCl = new ClienteCE();
            objCl.codigo = txtCodigo.Text;

            if (valida() == "")
            {
                DataRow fila = objCon.BUSCACLIENTE(objCl).Rows[0];
                txtNombre.Text = fila[1].ToString();
                txtPaterno.Text = fila[2].ToString();
                txtMaterno.Text = fila[3].ToString();
                txtTelefono.Text = fila[4].ToString();
                txtDni.Text = fila[5].ToString();
                txtDireccion.Text = fila[6].ToString();

                //Recuperando la imagen del cliente
                byte[] datos = new byte[0];
                datos = (byte[])fila[7];

                //Mostrando la imagen del cliente
                MemoryStream ms = new MemoryStream(datos);
                Image imagen = Image.FromStream(ms);
                pbImagen.Image = imagen;
            }
            else
                MessageBox.Show("El error se encuentra en " + valida());
        }

        //Método de validación de datos
        string valida()
        {
            if (txtCodigo.Text.Trim().Length == 0)
```

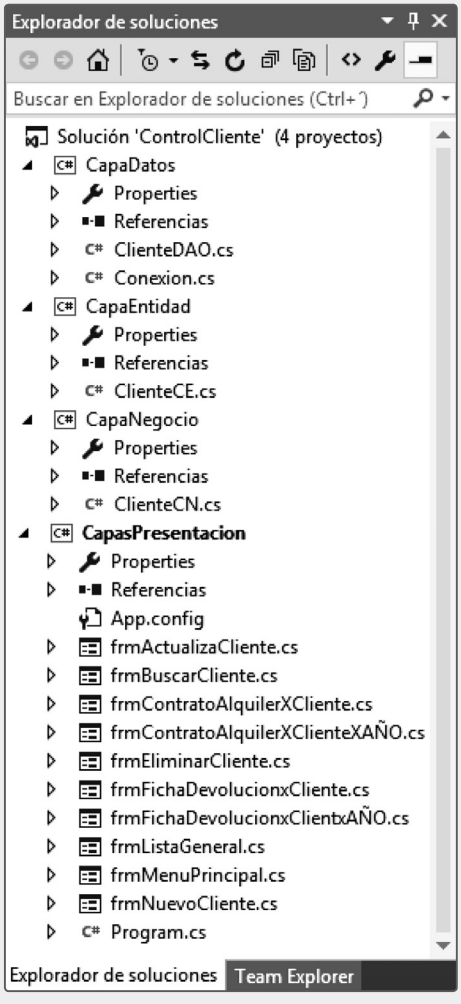
```
            {
                return " Codigo del CLIENTE";
            }
            else
            {
                return "";
            }
        }

        private void btnEliminar_Click(object sender, EventArgs e)
        {
            ClienteCE objCl = new  ClienteCE();
            objCl.codigo = txtCodigo.Text;
            try
            {
                objCon.ELIMINACLIENTE(objCl);
                MessageBox.Show("CLIENTE ELIMINADO CORRECTAMENTE..!!",
                                "ELIMINADO", MessageBoxButtons.OK,
                                MessageBoxIcon.Information);
                limpiar(this);
                pbImagen.Image = null;
                txtCodigo.Focus();
            }
            catch (Exception) { }
        }

        private void btnCancelar_Click(object sender, EventArgs e)
        {
            this.Close();
        }
        public static void limpiar(Form ofrm)
        {
            foreach (Control oControls in ofrm.Controls)
            {
                if (oControls is TextBox)
                {
                    oControls.Text = "";
                }
            }
        }

        private void btnExaminar_Click(object sender, EventArgs e)
        {
            OP.Filter = "Archivo jpg|*.jpg";
            if (OP.ShowDialog() == DialogResult.OK)
            {
                pbImagen.Image = Image.FromFile(OP.FileName);
            }
        }
    }
}
```

21. Observe el código del formulario frmMenuPrincipal:

```
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 CapasPresentacion
{
    public partial class frmMenuPrincipal : Form
    {
        public frmMenuPrincipal()
        {
            InitializeComponent();
        }

       private void CloseAllToolStripMenuItem_Click(object sender, EventArgs e)
        {
            foreach (Form childForm in MdiChildren)
            {
                childForm.Close();
            }
        }

        private void nuevoClienteToolStripMenuItem_Click(…)
        {
            frmNuevoCliente childForm = new frmNuevoCliente();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void actualizaClienteToolStripMenuItem_Click(…)
        {
            frmActualizaCliente childForm = new frmActualizaCliente();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void eliminaClienteToolStripMenuItem_Click(…)
        {
            frmEliminarCliente childForm = new frmEliminarCliente();
            childForm.MdiParent = this;
            childForm.Show();
        }
```

```
        private void buscaClienteToolStripMenuItem_Click(…)
        {
            frmBuscarCliente childForm = new frmBuscarCliente();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void listaGeneralToolStripMenuItem_Click(…)
        {
            frmListaGeneral childForm = new frmListaGeneral();
            childForm.MdiParent = this;
            childForm.Show();
        }

      private void contratoDeAlquilerPorClienteToolStripMenuItem_Click(…)
        {
            frmContratoAlquilerXCliente childForm =
                                        new  frmContratoAlquilerXCliente();
            childForm.MdiParent = this;
            childForm.Show();
        }

        private void fichaDevolucionPorClienteToolStripMenuItem_Click(…)
        {
            frmFichaDevolucionxCliente childForm =
                                          new frmFichaDevolucionxCliente();
            childForm.MdiParent = this;
            childForm.Show();
        }

       private void contratoAlquilerPorClienteYAnoToolStripMenuItem_Click(…)
        {
            frmContratoAlquilerXClienteXAÑO childForm =
                                     new   frmContratoAlquilerXClienteXAÑO();
            childForm.MdiParent = this;
            childForm.Show();
        }

       private void fichaDeDevolucionPorClientePorAnoToolStripMenuItem_Click(…)
        {
            frmFichaDevolucionxClientxAÑO childForm =
                                        new frmFichaDevolucionxClientxAÑO();
            childForm.MdiParent = this;
            childForm.Show();
        }
    }
}
```

Finalmente, el Explorador de soluciones para el proyecto de control de cliente se debe tener el siguiente aspecto:



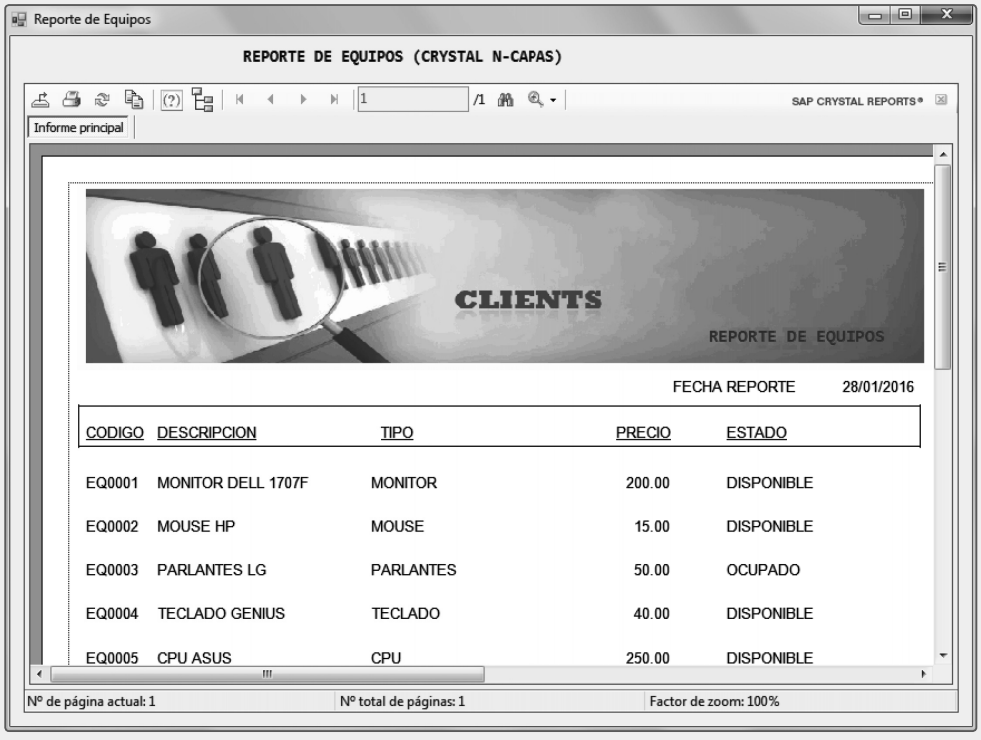
Caso desarrollado 3 : Reporte de equipos con Crystal Report en N-Capas

Implementar una solución que permite realizar el reporte de los equipos usando la tecnología de N-Capas.

Se tiene en cuenta lo siguiente:

- Crear una solución en blanco.
- Implementar un procedimiento almacenado en SQL Server que permita listar los datos de los equipos que se reportarán.
- Agregar tres proyectos de bibliotecas de clases y llamarlos CapaDatos, CapaEntidad y CapaNegocio.

- Agregar un proyecto de Windows Forms y llamarlo CapaPresentacion.
- El formulario de reporte de clientes se muestra de la siguiente manera:

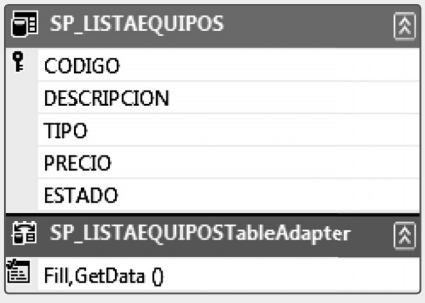


Solución:

1. Ejecute los siguientes procedimientos almacenados en SQL Server.

```
--SP_LISTAEQUIPOS
IF OBJECT_ID("SP_LISTAEQUIPOS")IS NOT NULL
       DROP PROC SP_LISTAEQUIPOS
GO
CREATE PROC SP_LISTAEQUIPOS
AS
       SELECT E.IDE_EQU AS CODIGO,
              E.DESC_EQU AS DESCRIPCION,
              T.DES_TIP AS TIPO,
              E.PREC_EQU AS PRECIO,
              ES.DES_EST AS ESTADO
              FROM EQUIPO E
              JOIN TIPO_EQUIPO T ON E.COD_TIP_EQU=T.COD_TIP_EQU
              JOIN ESTADO_EQUIPO ES ON E.COD_EST=ES.COD_EST
GO
```

2. Cree una solución en blanco en Visual Studio 2015, agregue tres proyectos del tipo biblioteca de clase a la solución y asigne los nombres CapaDatos, CapaEntidad y CapaNegocios.
3. Agregue a la solución un proyecto de tipo Windows Forms y asigne el nombre capaPresentacion.
4. Elimine los archivos Class1.cs de todas las capas.
5. A la capa de datos, agregue la referencia System.Configuration.
6. Agregue un elemento de tipo DATASET a la capa de presentación, asigne el nombre dsEquipo y agregue el procedimiento almacenado ejecutado en SQL Server, como se muestra en la siguiente imagen:



7. Agregue un elemento de tipo Crystal Report a la capa de presentación y llámelo rptEquipo. Luego, diseñe un reporte simple desde el Dataset dsEquipo.
8. Desde la capa de presentación, modifique el archivo app.config, de tal forma que permita conectarse a la base de datos Contrato:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
      <add name="cn" connectionString="SERVER=.;DATABASE=CONTRATO;
                                       INTEGRATED SECURITY=SSPI"/>
    </connectionStrings>
    <startup useLegacyV2RuntimeActivationPolicy="true">
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

9. Agregue la clase EquipoCE a la capa entidad, esta tendrá la misión de especificar todos los métodos get y set que componen la clase Equipo:

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

namespace CapaEntidad
{
  public  class EquipoCE
    {
        public string codigo { get; set; }
        public string tipo { get; set; }
        public string descripcion { get; set; }
        public double precio { get; set; }
        public string estado { get; set; }
    }
}
```

10. Agregue la clase Conexión a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Configuration;

namespace CapaDatos
{
  public  class Conexion
    {
      public SqlConnection getConecta()
      {
          SqlConnection cn = new SqlConnection(ConfigurationManager
                                  .ConnectionStrings["cn"].ConnectionString);
          return cn;
      }
    }
}
```

11. Agregue la clase EquipoDAO a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using CapaEntidad;

namespace CapaDatos
{
  public  class EquipoDAO
    {
      SqlConnection cn;
      Conexion objC = new Conexion();

      //Listando Equipos

      public DataTable listaEquipos()
      {
          cn = objC.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTAEQUIPOS", cn);
          DataTable dt = new DataTable();
          da.Fill(dt);
          return dt;
      }
    }
}
```

12. Agregue la clase EquipoCN a la capa de negocio y coloque el siguiente código:

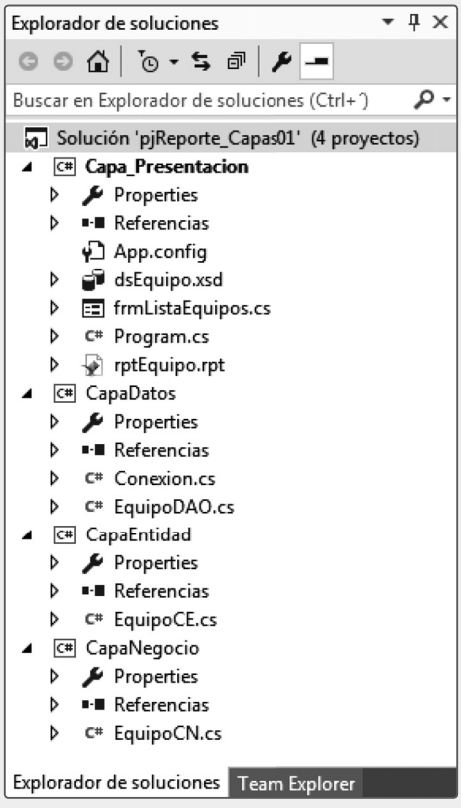
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Data;
using CapaEntidad;
using CapaDatos;

namespace CapaNegocio
{
  public  class EquipoCN
    {
      EquipoDAO objDAO = new EquipoDAO();

      public DataTable listaEquipos()
      {
          return objDAO.listaEquipos();
      }
    }
}
```

13. No olvide agregar al formulario el control Crystal Reporte Viewer y asigne el nombre crvEquipo.

14. Observe el código del formulario frmListaEquipos:

```
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 CapaNegocio;

namespace Capa_Presentacion
{
    public partial class frmListaEquipos : Form
    {
        //Creando enlace a la capa negocio
        EquipoCN objCon = new EquipoCN();

        public frmListaEquipos()
        {
            InitializeComponent();
        }

        private void frmListaEquipos_Load(object sender, EventArgs e)
        {

            rptEquipo reporte = new rptEquipo();

            //Asignando datos al reporte
            reporte.SetDataSource(objCon.listaEquipos());

            //Enviando la informacion al visor del reporte
            crvEquipo.ReportSource = reporte;
        }
    }
}
```

Finalmente, el Explorador de soluciones se debe mostrar de la siguiente manera al finalizar el proyecto.



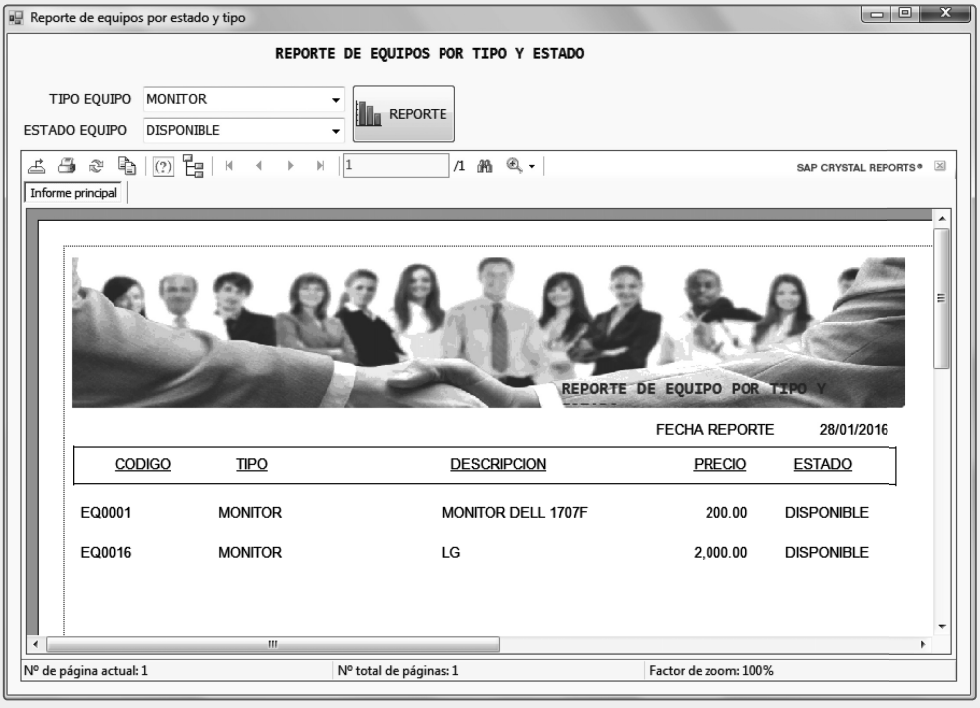
Caso desarrollado 4 : Reporte de equipos por estado y tipo con Crystal Report en N-Capas

Implementar una solución que permite realizar el reporte de los equipos según el estado y el tipo de equipo usando la tecnología de N-Capas.

Se tiene en cuenta lo siguiente:

- Crear una solución en blanco.
- Implementar procedimientos almacenados que permitan llenar los cuadros combinados del estado y el tipo de equipo, además de listar los equipos según el tipo y estado.
- Agregar tres proyectos de bibliotecas de clases y llámelos CapaDatos, CapaEntidad y CapaNegocio.

- Agregar un proyecto de Windows Forms y llámelo CapaPresentacion.
- El formulario de reporte de equipos, según el estado y tipo, se muestra de la siguiente manera:



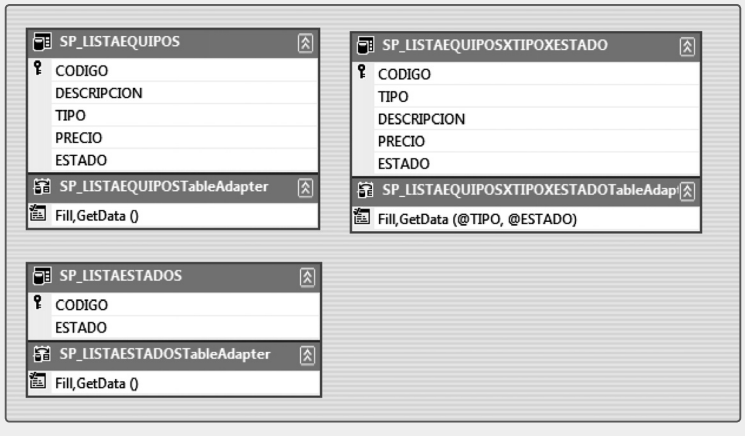
Solución:

1. Ejecute los siguientes procedimientos almacenados en SQL Server:

```
--SP_LISTATIPOEQUIPOS
IF OBJECT_ID("SP_LISTATIPOEQUIPOS")IS NOT NULL
       DROP PROC SP_LISTATIPOEQUIPOS
GO
CREATE PROC SP_LISTATIPOEQUIPOS
AS
       SELECT T.COD_TIP_EQU AS CODIGO,
              T.DES_TIP AS TIPO
              FROM TIPO_EQUIPO T
GO

--SP_LISTAESTADOS
IF OBJECT_ID("SP_LISTAESTADOS")IS NOT NULL
       DROP PROC SP_LISTAESTADOS
GO
CREATE PROC SP_LISTAESTADOS
AS
```

```
        SELECT ES.COD_EST AS CODIGO,
               ES.DES_EST AS ESTADO
               FROM ESTADO_EQUIPO ES
GO

--SP_LISTAEQUIPOSXTIPOXESTADO
IF OBJECT_ID("SP_LISTAEQUIPOSXTIPOXESTADO")IS NOT NULL
        DROP PROC SP_LISTAEQUIPOSXTIPOXESTADO
GO
CREATE PROC SP_LISTAEQUIPOSXTIPOXESTADO(@TIPO CHAR(6), @ESTADO INT)
AS
        SELECT E.IDE_EQU AS CODIGO,
               E.DESC_EQU AS DESCRIPCION,
               T.DES_TIP AS TIPO,
               E.PREC_EQU AS PRECIO,
               ES.DES_EST AS ESTADO
               FROM EQUIPO E
               JOIN TIPO_EQUIPO T ON E.COD_TIP_EQU=T.COD_TIP_EQU
               JOIN ESTADO_EQUIPO ES ON E.COD_EST=ES.COD_EST
               WHERE E.COD_TIP_EQU=@TIPO AND E.COD_EST=@ESTADO
GO
--PRUEBA:  SP_LISTAEQUIPOSXTIPOXESTADO "TIP002",0
```

2. Cree una solución en blanco en Visual Studio 2015, agregue tres proyectos del tipo biblioteca de clase a la solución y asigne los nombres CapaDatos, CapaEntidad y CapaNegocios.
3. Agregue a la solución un proyecto de tipo Windows Forms y asigne el nombre capaPresentacion.
4. Elimine los archivos Class1.cs de todas las capas.
5. A la capa de datos, agregue la referencia System.Configuration.
6. Agregue un elemento de tipo DATASET a la capa de presentación, asigne el nombre dsEquipo y agregue todos los procedimientos almacenados ejecutados en SQL Server, como se muestra en la siguiente imagen:



7. Agregue un elemento de tipo Crystal Report a la capa de presentación y llámelo rptEquipo. Luego, diseñe un reporte simple con el procedimiento almacenado SP_LISTAEQUIPOSXTIPOXESTADO.

8. Desde la capa de presentación, modifique el archivo app.config, de tal forma que permita conectarse a la base de datos Contrato:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
      <add name="cn" connectionString="SERVER=.;DATABASE=CONTRATO;
                                        INTEGRATED SECURITY=SSPI"/>
    </connectionStrings>
    <startup useLegacyV2RuntimeActivationPolicy="true">
       <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

9. Agregue la clase EquipoCE a la capa entidad, esta tendrá la misión de especificar todos los métodos get y set que componen la clase Equipo:

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

namespace CapaEntidad
{
  public  class EquipoCE
    {
        public string codigo { get; set; }
        public string tipo { get; set; }
        public string descripcion { get; set; }
        public double precio { get; set; }
        public string estado { get; set; }
    }
}
```

10. Agregue la clase Conexión a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Configuration;

namespace CapaDatos
{
  public  class Conexion
    {
      public SqlConnection getConecta()
      {
          SqlConnection cn = new SqlConnection(ConfigurationManager.
                                  ConnectionStrings["cn"].ConnectionString);
          return cn;
      }
    }
}
```

11. Agregue la clase EquipoDAO a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using CapaEntidad;

namespace CapaDatos
{
  public  class EquipoDAO
    {
      SqlConnection cn;
      Conexion objC = new Conexion();

     public DataTable listaTipoEquipos()
      {
          cn = objC.getConecta();
```

```
        SqlDataAdapter da = new SqlDataAdapter("SP_LISTATIPOEQUIPOS", cn);
         DataTable dt = new DataTable();
         da.Fill(dt);
         return dt;
      }

      public DataTable listaEstado()
      {
          cn = objC.getConecta();
          SqlDataAdapter da = new SqlDataAdapter("SP_LISTAESTADOS", cn);
          DataTable dt = new DataTable();
          da.Fill(dt);
          return dt;
      }

      public DataTable LISTAEQUIPOSXTIPOXESTADO(EquipoCE E)
      {
        cn = objC.getConecta();

       SqlDataAdapter da = new SqlDataAdapter("SP_LISTAEQUIPOSXTIPOXESTADO",cn);
        da.SelectCommand.CommandType = CommandType.StoredProcedure;
       da.SelectCommand.Parameters.Add("@TIPO", SqlDbType.Char).Value = E.tipo;
    da.SelectCommand.Parameters.Add("@ESTADO", SqlDbType.Char).Value = E.estado;
        DataTable dt = new DataTable();
        da.Fill(dt);
        return dt;
      }
    }
}
```

12. Agregue la clase EquipoCN a la capa de negocio y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Data;
using CapaEntidad;
using CapaDatos;

namespace CapaNegocio
{
  public  class EquipoCN
    {
      EquipoDAO objDAO = new EquipoDAO();

      public DataTable listaTipoEquipos()
```

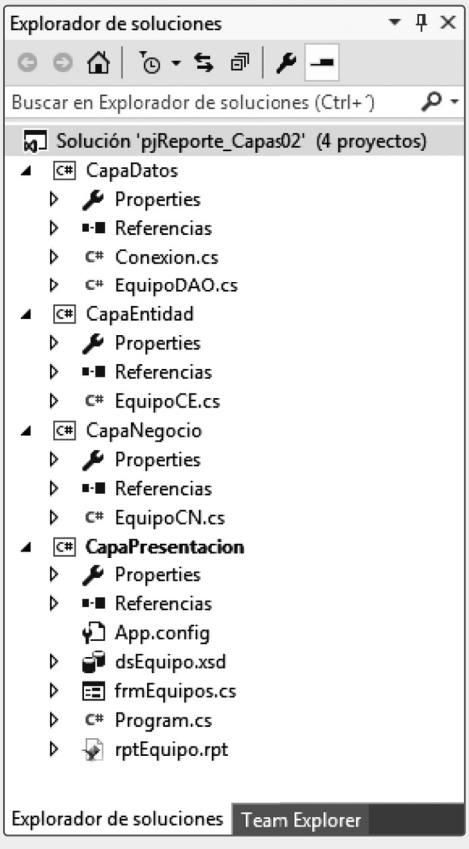
```
        {
            return objDAO.listaTipoEquipos();
        }
        public DataTable listaEstado()
        {
            return objDAO.listaEstado();
        }
        public DataTable LISTAEQUIPOSXTIPOXESTADO(EquipoCE E)
        {
            return objDAO.LISTAEQUIPOSXTIPOXESTADO(E);

        }
    }
}
```

13. No olvide agregar al formulario el control Crystal Report Viewer y asignar el nombre crvEquipoS.
14. Observe el código del formulario frmEQUIPOSXTIPOSXESTADO:

```
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 CapaEntidad;
using CapaNegocio;

namespace CapaPresentacion
{
    public partial class frmEQUIPOSXTIPOSXESTADO : Form
    {
        EquipoCN objCON = new EquipoCN();

        public frmEQUIPOSXTIPOSXESTADO()
        {
            InitializeComponent();
        }

       private void frmEQUIPOSXTIPOSXESTADO_Load(object sender, EventArgs e)
        {
            llenarTipo();
            llenarEstado();
        }
        void llenarTipo()
        {
            cboTipo.DataSource = objCON.listaTipoEquipos();
```

```
            cboTipo.DisplayMember = "TIPO";
            cboTipo.ValueMember = "CODIGO";
        }

        void llenarEstado()
        {
            cboEstado.DataSource = objCON.listaEstado();
            cboEstado.DisplayMember = "ESTADO";
            cboEstado.ValueMember = "CODIGO";
        }

        private void btnReporte_Click(object sender, EventArgs e)
        {
            EquipoCE objE = new EquipoCE();
            objE.tipo = cboTipo.SelectedValue.ToString();
            objE.estado = cboEstado.SelectedValue.ToString();

            rptEquipo reporte = new rptEquipo();
            reporte.SetDataSource(objCON.LISTAEQUIPOSXTIPOXESTADO(objE));
            crvEquipos.ReportSource = reporte;
        }
    }
}
```

Finalmente, el Explorador de soluciones se debe mostrar de la siguiente manera al finalizar el proyecto.

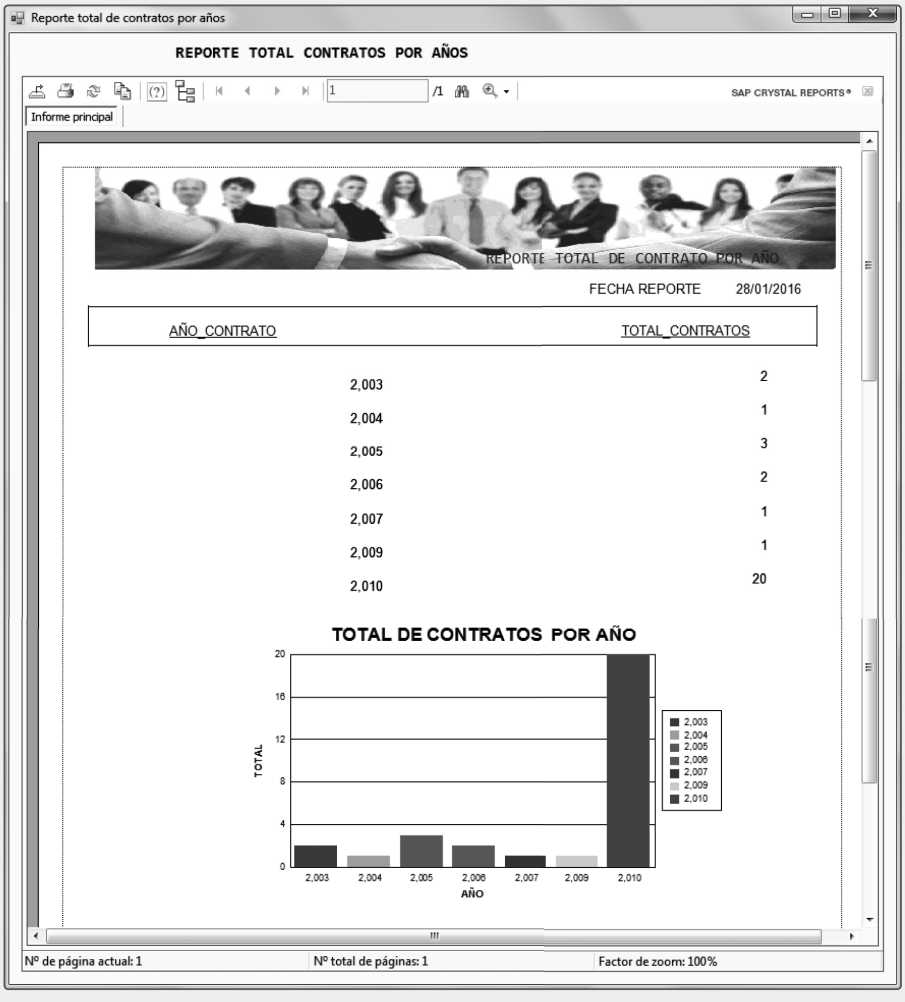


Caso desarrollado 5 : Reporte total de contratos por años con gráfico en N-Capas

Implementar una solución que permite realizar el reporte del total de contratos por años usando la tecnología de N-Capas.

Se tiene en cuenta lo siguiente:

- Crear una solución en blanco.
- Implementar un procedimiento almacenado que muestra los años de registro del contrato y el total de contratos registrados por años.
- Agregar tres proyectos de bibliotecas de clases y llamarlos CapaDatos, CapaEntidad y CapaNegocio.
- Agregar un proyecto de Windows Forms y llamarlo CapaPresentacion.
- El formulario del total de contratos por años se muestra de la siguiente manera:



Solución:

1. Ejecute el siguiente procedimiento almacenado en SQL Server.

```
IF OBJECT_ID("SP_CONTRATOSXAÑOS")IS NOT NULL
        DROP PROC SP_CONTRATOSXAÑOS
GO
CREATE PROC SP_CONTRATOSXAÑOS
AS
        SELECT YEAR(C.FIN_CON) AS AÑO_CONTRATO,
               COUNT(*) AS TOTAL_CONTRATOS
               FROM CONTRATOALQUILER C
               GROUP BY YEAR(C.FIN_CON)
GO
--PRUEBA: SP_CONTRATOSXAÑOS
```

2. Cree una solución en blanco en Visual Studio 2015 y agregue tres proyectos del tipo biblioteca de clase a la solución. Luego, asigne los nombres CapaDatos, CapaEntidad y CapaNegocios.
3. Agregue a la solución un proyecto de tipo Windows Forms y asigne el nombre capaPresentacion.
4. Elimine los archivos Class1.cs de todas las capas.
5. A la capa de datos, agregue la referencia System.Configuration.
6. Agregue un elemento de tipo DATASET a la capa de presentación, asigne el nombre dsTotal y agregue el procedimiento almacenado ejecutado en SQL Server, como se muestra en la siguiente imagen:

7. Agregue un elemento de tipo Crystal Report a la capa de presentación y llámelo rptTotal. Luego, diseñe un reporte simple con el procedimiento almacenado SP_CONTRATOSXAÑOS.
8. Desde la capa de presentación, modifique el archivo app.config, de tal forma que permita conectarse a la base de datos Contrato:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <configSections>
    </configSections>
    <connectionStrings>
      <add name="cn" connectionString="SERVER=.;DATABASE=CONTRATO;
                                       INTEGRATED SECURITY=SSPI"/>
```

```
    </connectionStrings>
    <startup useLegacyV2RuntimeActivationPolicy="true">
        <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
    </startup>
</configuration>
```

9. Agregue la clase Conexión a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Configuration;

namespace CapaDatos
{
  public  class Conexion
    {
      public SqlConnection getConecta()
      {
          SqlConnection cn = new SqlConnection(ConfigurationManager.
                                  ConnectionStrings["cn"].ConnectionString);
          return cn;
      }
    }
}
```

10. Agregue la clase ContratoDAO a la capa de datos y coloque el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data;
using System.Data.SqlClient;
using CapaEntidad;

namespace CapaDatos
{
  public  class ContratoDAO
    {
      SqlConnection cn;
      Conexion objC = new Conexion();
```

```
        public DataTable listaContratoxAÑO()
        {
            cn = objC.getConecta();
           SqlDataAdapter da = new SqlDataAdapter("SP_CONTRATOSXAÑOS", cn);
            DataTable dt = new DataTable();
            da.Fill(dt);
            return dt;
        }
    }
}
```

11. Agregue la clase ContratoCN a la capa de negocio y coloque el siguiente código:

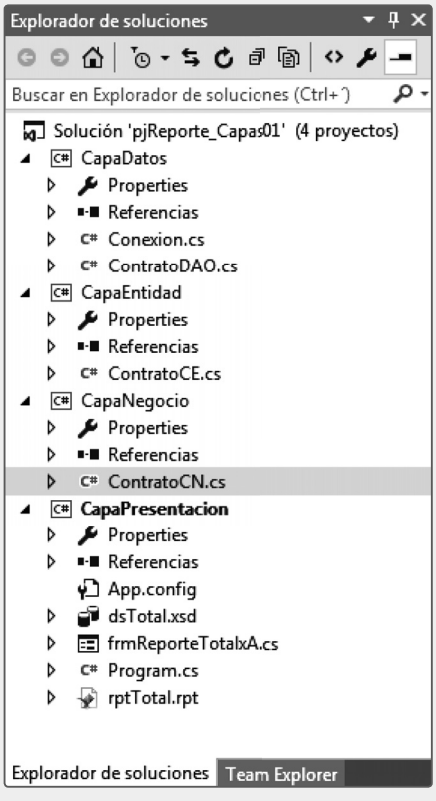
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Data.SqlClient;
using System.Data;
using CapaEntidad;
using CapaDatos;

namespace CapaNegocio
{
  public  class ContratoCN
    {
      ContratoDAO objDAO = new ContratoDAO();

      public DataTable listaContratoxAÑO()
      {
          return objDAO.listaContratoxAÑO();
      }
    }
}
```

12. No olvide agregar al formulario el control Crystal Report Viewer y asignar el nombre crvTotal.
13. Observe el código del formulario frmReporteTotalxA:

```
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 CapaNegocio;
using CapaEntidad;

namespace CapaPresentacion
{
    public partial class frmReporteTotalxA : Form
    {
        ContratoCN objCN = new ContratoCN();

        public frmReporteTotalxA()
        {
            InitializeComponent();
        }

        private void frmReporteTotalxA_Load(object sender, EventArgs e)
        {
            rptTotal reporte = new rptTotal();
            reporte.SetDataSource(objCN.listaContratoxAÑO());
            crvTotal.ReportSource = reporte;
        }
    }
}
```

14. Finalmente, el Explorador de soluciones se debe mostrar de la siguiente manera al finalizar el proyecto.



Bibliografía

Joyanes, L. & Zahonero, I. (2005). *Programación en C: metodología, algoritmos y estructura de datos*. 2.ª ed. Estados Unidos: McGraw- Hill.

Microsoft. https://msdn.microsoft.com/es-es/library/e80y5yhx(v=vs.110).aspx [Consultado: Febrero 2016]

EDITORIAL MACRO®

Impreso en los talleres gráficos de

EDITORIAL MACRO®

Surquillo