UNIDAD

DIPLOMATURA EN PROGRAMACION JAVA

Proyecto Ventas



# Manual de Ejercicios – Capítulo 6

## Laboratorio 9

Para crear el servicio WCF y el servidor que maneje la comunicación con el cliente se debe cambiar el proyecto del que se venía trabajando. Este nuevo diseño una clase dedicada a la comunicación con el servidor debe encaminará todas las operaciones que se deseen realizar para que estas se ejecuten en el servicio.

Por otra parte, el servicio deberá encargarse de todas las operaciones contra la base de datos. Por lo tanto éste se convierte en el modelo remoto del patrón de diseño MVC. Como se va a trabajar en una arquitectura orientada al servicio, se modifica levemente el patrón de diseño de manera que la vista no se actualizará automáticamente. Si bien la tecnología está preparada para ofrecer esto, excede de los límites de lo que se quiere mostrar como introducción al tema.

Para el nuevo diseño se cambia el cliente gráfico por uno nuevo llamado Aplicacion3CapasVentas dentro del proyecto Ciente3CapasVentas. La única razón por la que se realizará este cambio es para mantener conceptualmente separada la aplicación de dos capas de la de tres.

Esta clase crea una instancia de ClienteDelServicioVentas que será la encargada de comunicarse con el servicio. En el constructor de esta clase se genera el proxy para la comunicación con el servicio WCF y provee una propiedad que se puede acceder para tener una referencia de este.

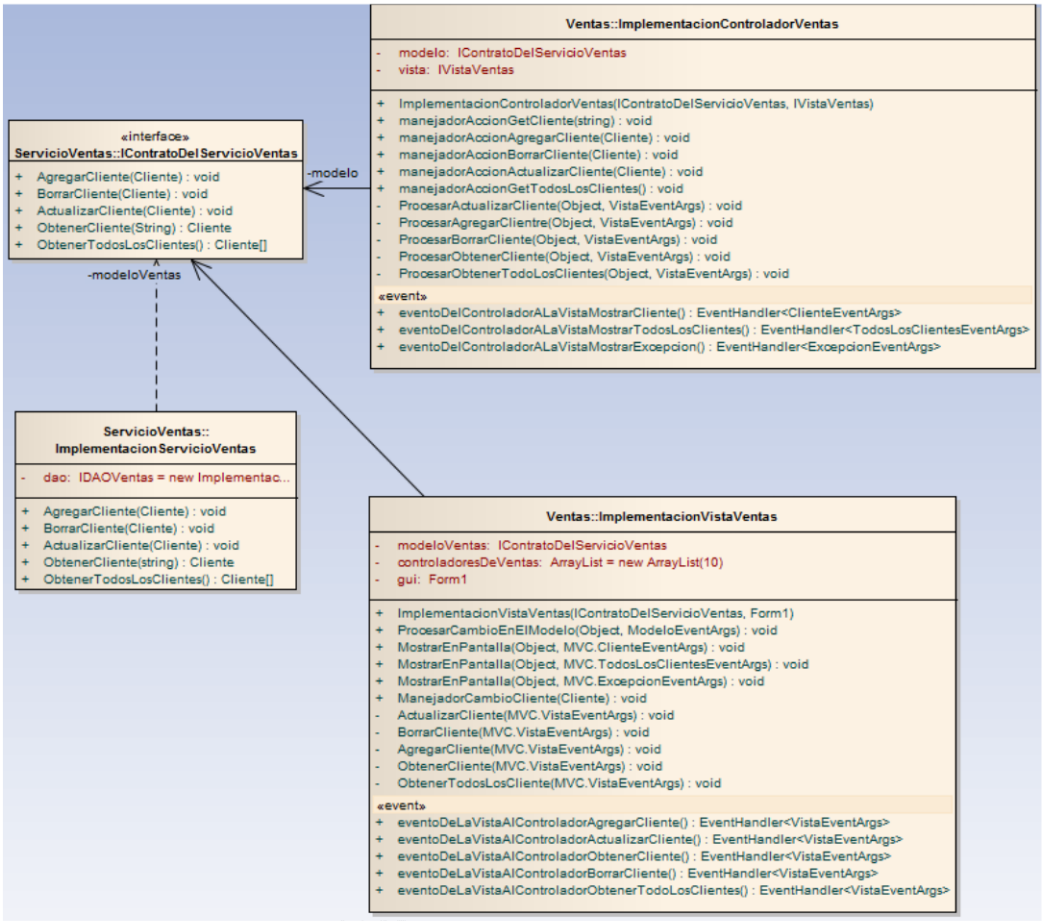
Cuando se crea el punto final se asigna la dirección que se estableció en el servidor porque es la prefijada para el servicio que se está brindando. Esta dirección está expuesta como otro punto final del otro lado en la clase Servidor que se encuentra dentro del proyecto ServidorDeServicio, la cual a su vez en la encargada de exponer el acceso al servicio.

Al entregar los datos del servicio utiliza un contrato de datos dentro de la clase Cliente ya que es la única entidad que se va a utilizar para el segmento. La única modificación respecto de la clase utilizada anteriormente son los atributos que definen el contrato de datos.

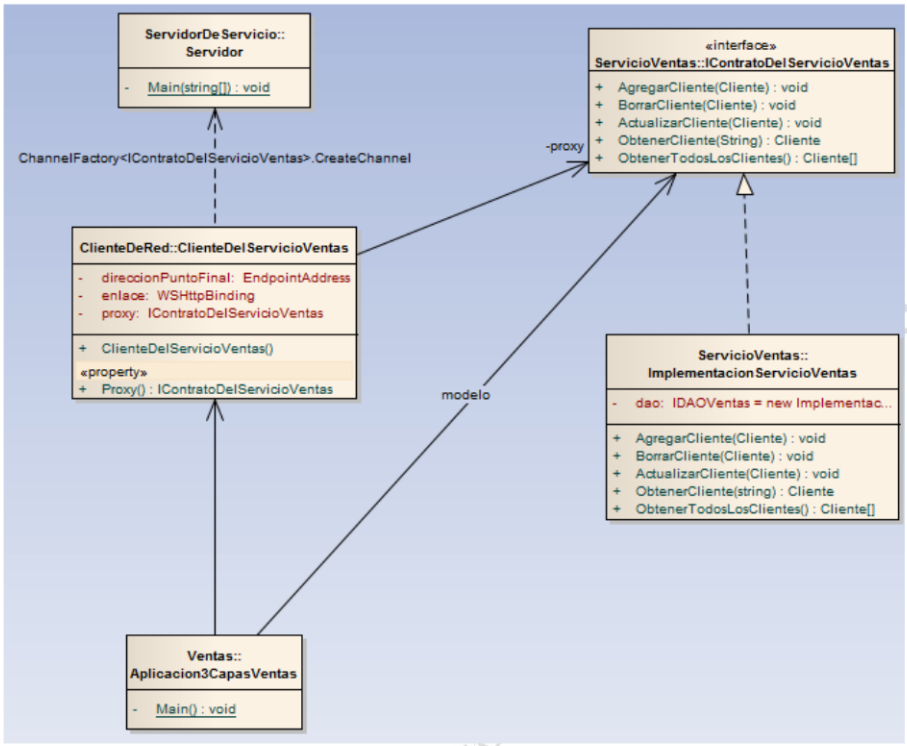
En este laboratorio no se tendrán en cuenta las excepciones porque se explicarán posteriormente.

El próximo actuar para la aplicación como si fuera el modelo, por lo tanto las operaciones remotas del servicio. Esto se consigue por una referencia a la interfaz que permite acceder a los métodos definidos en el proxy como si fueran locales.

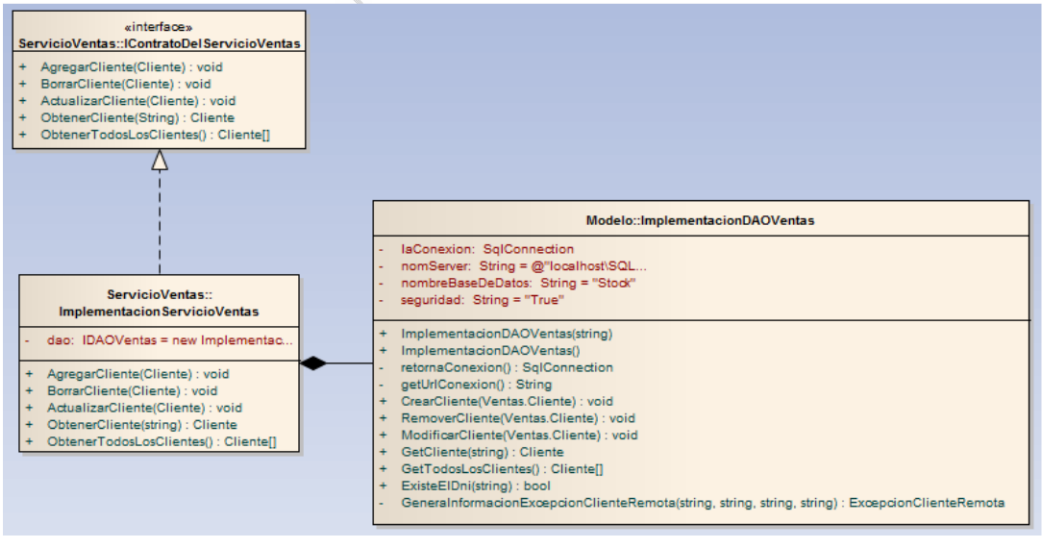
Por lo tanto el controlador y la vista accederán al proxy como lo hacían anteriormente con el modelo. Esto implica que las referencias deben ser cambiadas cuando se crea los objetos de manera que el modelo actual es el proxy. El siguiente diagrama muestra como queda conformada las relaciones entre las diferentes clases:




La clase que actúa como cliente de red debe comunicarse con el servidor creando un canal. De esta manera se tiene una referencia a las operaciones que expone el contrato. El siguiente diagrama muestra como queda la relación entre clases y cómo se crea el canal mencionado.




Del lado del servidor el servicio compone una referencia a ImplementacionDAOVentas para lograr el acceso a la base de datos. Por lo tanto los métodos que se utilizaron para invocar un objeto de este tipo en el modelo pasan a ser las operaciones del contrato del servicio que se implementan en la clase concreta ImplementacionServicioVentas. El siguiente diagrama refleja a lo explicado:



## Creación de un servicio y su servidor con WCF

A continuación se expondrán los pasos para crear un proyecto de librería WCF, cómo se definirá la interfaz del servicio que se quiere brindar, la cual es el contrato a implementar en la clase concreta y por último la clase que implementa el servicio, la cual es una copia de la clase ImplementacionModeloVentas sin la lista con los suscriptores de cambios ni la declaración del evento que notifica a la vista (recordar que en esta implementación la vista y el modelo se implementan es espacios diferentes de memoria y un evento no tiene sentido salvo que sea remoto y dicha explicación exceden los objetivos actuales).

## Creación del servicio

1. Crear un nuevo proyecto
2. Seleccionar WCF
3. Elegir la plantilla "WCF Service Library". Asignarle el nombre ServicioVentas y presionar "OK". Esto creará un proyecto que genera una dll para un servicio WCF.
4. Agregar una referencia al proyecto Entidades
5. Agregar una referencia al proyecto Modelo
6. Agregar una referencia al proyecto Excepciones
7. Cambiar el nombre de la interfaz IService1 a IContratoDelServicioVentas.
8. Borrar el contenido de la interfaz de ejemplo y toda la clase CompositeType.
9. Declarar las mismas operaciones que figuran en la interfaz IModeloVentas (proyecto MVC) con excepción del evento y resolver las referencias a las clases que estén en otro espacio de nombres.
10. Anteponer la declaración del atributo [OperationContract] u <OperationContract()> (C# o VB) para cada método. Esto indica en tiempo de ejecución a WCF que será un método remoto que se expondrá en el proxy.
11. El contenido de la interfaz deberá ser como el que se muestra a continuación:

**C#**

```
/**-------------------------------------------------------------
 * Agregar el Cliente al modelo Ventas
 */
[OperationContract]
void AgregarCliente(Cliente cliente);

/**-------------------------------------------------------------
 * Borrar el cliente del modelo Ventas
 */
[OperationContract]
void BorrarCliente(Cliente cliente);

/**-------------------------------------------------------------
 * Actualizar el cliente en el modelo Ventas
 */
[OperationContract]
void ActualizarCliente(Cliente cliente);

// Métodos del segmento Cliente para consulta del estado
/**-------------------------------------------------------------
```

```
        * Dado un dni, retorna el cliente del modelo
        */
       [OperationContract]
       Cliente ObtenerCliente(String id);


       /**-------------------------------------------------------------
        * Retorna todos los clientes en el modelo Ventas
        */
       [OperationContract]
       Cliente[] ObtenerTodosLosClientes();
```

## VB

```
' Agregar el Cliente al modelo Ventas
'
<OperationContract()>
Sub AgregarCliente(ByVal cliente As Cliente)

'-------------------------------------------------------------
' Borrar el cliente del modelo Ventas
'
<OperationContract()>
Sub BorrarCliente(ByVal cliente As Cliente)

'-------------------------------------------------------------
' Actualizar el cliente en el modelo Ventas
'
<OperationContract()>
Sub ActualizarCliente(ByVal cliente As Cliente)

' Métodos del segmento Cliente para consulta del estado
'-------------------------------------------------------------
' Dado un dni, retorna el cliente del modelo
'
<OperationContract()>
Function ObtenerCliente(ByVal id As String) As Cliente


'-------------------------------------------------------------
' Retorna todos los clientes en el modelo Ventas
'
<OperationContract()>
Function ObtenerTodosLosClientes() As Cliente()
```

12. Cambiar el nombre de la clase concreta Service1 a ImplementacionServicioVentas.
13. Borrar el contenido de la clase.
14. Crear la variable de instancia de referencia a IDAOVentas para crear una instancia en la declaración de ImplementacionDAOVentas.
15. Implementar la interfaz en la clase concreta y delegar la resolución de las operaciones a los métodos del DAO manejando apropiadamente las excepciones. En el método Agregar cliente se debe reemplazar el lanzamiento de la excepción cuando encuentra el DNI. Poner en su lugar una salida por consola del tipo Console.WriteLine("Cliente Existe."). Posteriormente se verá el manejo de excepciones

16. No debe haber ninguna línea de código que dispare el evento que informa el cambio en el modelo
17. Manejar apropiadamente los mensajes que salen por consola
18. El código debe verse como el siguiente:

## C#

```
namespace ServicioVentas
{
    public class ImplementacionServicioVentas : IContratoDelServicioVentas
    {
        private IDAOVentas dao = new ImplementacionDAOVentas();

        public void AgregarCliente(Cliente cliente)
        {
            try
            {
                if (dao.ExisteElDni(cliente.Id))
                {
                    Console.WriteLine("Cliente Existe.");
                }
                else
                {
                    dao.CrearCliente(cliente);
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("ImplementacionModeloVentas.AgregaCliente\n" +
                    e.Message);
                Console.WriteLine(e.StackTrace);
            }
        }

        public void BorrarCliente(Cliente cliente)
        {
            dao.RemoverCliente(cliente);
        }

        public void ActualizarCliente(Cliente cliente)
        {
            dao.ModificarCliente(cliente);
        }

        public Ventas.Cliente ObtenerCliente(string id)
        {
            return dao.GetCliente(id);
        }

        public Ventas.Cliente[] ObtenerTodosLosClientes()
        {
            return dao.GetTodosLosClientes();
        }
    }
}
```

## VB

```
Imports Modelo
Imports Entidades
Imports Excepciones
Imports System.Data.SqlClient

' NOTE: You can use the "Rename" command on the context menu to change the class
name "Service1" in both code and config file together.
Public Class ImplementacionServicioVentas
    Implements IContratoDelServicioVentas

    Private dao As IDAOVentas = New ImplementacionDAOVentas
    ''-------------------------------------------------------
    ' Agrega el Cliente al modelo
    ''
    Public Sub AgregarCliente(cliente As Cliente) _
       Implements IContratoDelServicioVentas.AgregarCliente
        Try
            If (dao.ExisteElDni(cliente.Id)) Then
                Console.WriteLine("Cliente Existe.")
            Else
                dao.CrearCliente(cliente)
            End If
        Catch e As Exception
            Console.WriteLine("ServicioVentas.agregaCliente\n" & e.Message)
            Console.WriteLine(e.StackTrace)
            Throw New ExcepcionVentas("ServicioVentas.agregaCliente\n" & e.Message)
        End Try
    End Sub

    ''-------------------------------------------------------------
    ' Borra al cliente del modelo
    ''
    Public Sub BorrarCliente(cliente As Cliente) _
       Implements IContratoDelServicioVentas.BorrarCliente
        Try
            If (Not dao.ExisteElDni(cliente.Id)) Then
                Throw New ExcepcionVentas("El DNI no existe")
            Else
                dao.RemoverCliente(cliente)
            End If
        Catch e As Exception
            Console.WriteLine(e.StackTrace)
            Throw New ExcepcionVentas("ServicioVentas.borrarCliente\n" & e.Message)
        End Try
    End Sub

    ''-------------------------------------------------------------
    ' Actualizar el cliente en el modelo
    ''
    Public Sub ActualizarCliente(cliente As Cliente) _
       Implements IContratoDelServicioVentas.ActualizarCliente
        Try
            If (Not dao.ExisteElDni(cliente.Id)) Then
                Throw New ExcepcionVentas("El DNI no existe")
            Else
                dao.ModificarCliente(cliente)
```

```
            End If

        Catch e As Exception
            Console.WriteLine(e.StackTrace)
            Throw New ExcepcionVentas( _
              "ServicioVentas.actualizarCliente\n" & e.Message)
        End Try
    End Sub

    ' Segmento de los métodos de consulta del estado del Cliente
    ''-------------------------------------------------------------
    ' Dado un dni, retorna el Cliente del modelo
    '
    Public Function ObtenerCliente(id As String) _
       As Cliente Implements IContratoDelServicioVentas.ObtenerCliente
        Try
            Dim cli As Cliente = Nothing

            cli = dao.GetCliente(id)

            If (cli Is Nothing) Then
                ' si la consulta falla
                Throw New ExcepcionVentas("Registro para " & id & " no se encontró")
            End If

            ' retorna cli
            Return cli
        Catch e As Exception
            Console.WriteLine(e.StackTrace)
            Throw New ExcepcionVentas("ServicioVentas.ObtenerCliente\n" & e.Message)
        End Try
    End Function

    Public Function ObtenerTodosLosClientes() As Cliente() _
       Implements IContratoDelServicioVentas.ObtenerTodosLosClientes
        Dim todosCli As Cliente() = Nothing
        Try
            todosCli = dao.GetTodosLosClientes()
        Catch e As SqlException
            todosCli = Nothing
            Console.WriteLine(e.StackTrace)
            Throw New ExcepcionVentas( _
              "ServicioVentas.getTodosLosClientes\n" & e.Message)
        End Try
        Return todosCli
    End Function
End Class
```



### Creación del contrato de datos

Para que los datos que se recuperan de la base de datos se puedan transmitir a través de la red se deben serializar. En WCF se arma un contrato de datos con este fin. Como en este caso sólo se transmitirán los datos de la clase Cliente, se define en ella el único contrato de datos a utilizar.

19. En el proyecto Entidades agregar una referencia a System.Runtime.Serialization
20. Agregar una referencia a System.ServiceModel

21. Declarar el atributo [DataContract] en C# o <DataContract()> en VB antecediendo la declaración de la clase.
22. Declarar el atributo [DataMember] en C# o <DataMember()> en VB antecediendo a cada propiedad de la clase.

### Creación del servidor

23. Crear un nuevo proyecto de consola. Llamarlo ServidorDeServicio
24. Agregar una referencia al proyecto ServicioVentas
25. Agregar una referencia a System.ServiceModel
26. Generar un objeto que maneje la dirección en dónde se prestará servicio con la siguiente declaración:

```
Uri direccionBase = new Uri("http://localhost:8888/");
```

27. Obtener el tipo de la instancia de la clase concreta que implementa servicio mediante la siguiente declaración:

**C#**

```
Type tipoDeLaInstancia =
        typeof(ImplementacionServicioVentas);
```

**VB**

```
Dim tipoDeLaInstancia As Type = GetType(ImplementacionServicioVentas)
```

28. Crear el servidor del servicio pasando el tipo de la instancia de la clase concreta que lo implementa y la dirección en dónde se la debe enlazar. Para eso se debe utilizar la clase ServiceHost y resolver la visibilidad del espacio de nombres System.ServiceModel:

**C#**

```
ServiceHost servidor = new ServiceHost(tipoDeLaInstancia, direccionBase);
```

**VB**

```
Dim servidor As New ServiceHost(tipoDeLaInstancia, direccionBase)
```

29. Definir un bloque `using` que maneje los recursos del servidor automáticamente de la siguiente manera

**C#**

```
using (servidor)
{
        :
        :
}
```

**VB**

```
Using servidor
        :
        :
End Using
```

30. Definir el tipo de contrato que utilizar al servicio. Esta declaración se va a utilizar para agregar el punto final al servidor:

**C#**

```
Type tipoDeContrato = typeof(IContratoDelServicioVentas);
```

**VB**

```
Dim tipoDeContrato As Type = _
       GetType(ServicioVentas.IContratoDelServicioVentas)
```

31. Crear el punto final en el servidor pasándole la dirección relativa con que se identifica servicio como una cadena:

**C#**

```
string direccionRelativa = "ServicioVentas";
servidor.AddServiceEndpoint(tipoDeContrato,
            new WSHttpBinding(), direccionRelativa);
```

**VB**

```
Dim direccionRelativa As String = "ServicioVentas"
servidor.AddServiceEndpoint( _
       tipoDeContrato, New WSHttpBinding(), direccionRelativa)
```

32. Abrir el servidor para que quede disponible la dirección de acceso al servicio y sacar por consola un mensaje que advierte que se está ejecutando el mismo

**C#**

```
servidor.Open();
Console.WriteLine("Servicio del Cliente de Ventas " +
       "ejecutándose. Presione <ENTER> para salir.");
```

**VB**

```
servidor.Open()
Console.WriteLine("Servicio del banco " +
       "ejecutándose. Presione <ENTER> para salir.")
```

33. En este punto se debe detener la ejecución del programa para que no se cierre, por lo tanto se utilizará una instrucción que lea la presión de la tecla Enter en el teclado. Cuando se lea dicho carácter, se debe permitir continuar con la ejecución para cerrar el programa:

**C#**

```
Console.ReadLine();
servidor.Close();
```

**VB**

```
Console.ReadLine()
servidor.Close()
```

### Creación del cliente de red

Se necesitará una aplicación que encamine la comunicación con el servidor del servicio. Con el fin de separar las ocupaciones de cada parte del código, esta clase se creará por separado en un nuevo proyecto del tipo librería, para desarrollar una dll que se incorpore en el proyecto que define la interacción entre el cliente gráfico y el modelo. Desde el punto de vista del cliente esta clase se comporta como el modelo, ya que expone las mismas operaciones que el mismo pero en realidad lo único que hace es encaminar el requerimiento al servicio mediante el servidor.

34. Crear un nuevo proyecto del tipo Class Library y llamarlo ClienteDeRed
35. Renombrar el archivo y la clase Class1.cs a ClienteDelServicioVentas.cs o ClienteDelServicioVentas.vb de manera que éste también sea el nombre de la nueva clase.
36. Agregar una referencia al proyecto ServicioVentas
37. Agregar una referencia a System.ServiceModel
38. Definir las siguientes variables de instancia y solucionar la visibilidad correspondiente a cada una en su espacio de nombres:

**C#**

```
private EndpointAddress direccionPuntoFinal;
private WSHttpBinding enlace;
private IContratoDelServicioVentas proxy;
```

**VB**

```
Private direccionPuntoFinal As EndpointAddress
Private enlace As WSHttpBinding
Private proxy As IContratoDelServicioVentas
```

39. En el constructor crear una nueva dirección de punto final de la siguiente manera:

**C#**

```
direccionPuntoFinal = new EndpointAddress(
      "http://localhost:8888/ServicioVentas");
```

**VB**

```
direccionPuntoFinal = New EndpointAddress( _
      "http://localhost:8888/ServicioVentas")
```

40. Crear un nuevo enlace el protocolo HTTP:

**C#**

```
enlace = new WSHttpBinding();
```

**VB**

```
enlace = New WSHttpBinding()
```

41. El canal de comunicación utilizando la dirección del punto final y el enlace creado con el protocolo HTTP.

**C#**

```
_proxy = ChannelFactory<IContratoDelServicioVentas>.CreateChannel(
       enlace, direccionPuntoFinal)
```

**VB**

```
_proxy = ChannelFactory(Of IContratoDelServicioVentas).CreateChannel( _
       enlace, direccionPuntoFinal)
```

42. Crear una propiedad para la variable _proxy

### Modificaciones en la vista

Revisar las modificaciones realizadas en el proyecto Ciente3CapasVentas. Se debe tener en cuenta que la abstracción del modelo está representada por el cliente de red, por lo tanto se cambiaron todas las referencias al mismo por referencias a un objeto del tipo ClienteDelServicioVentas. Las consecuencias de dicho cambio son las respectivas modificaciones en todos los lugares donde había una referencia el modelo en el proyecto de dos capas. Por lo tanto se modifica el código de formulario y del controlador de manera que pueda manejar una referencia a la interfaz del servicio que es la utilizada por el proxy para realizar la invocación de las operaciones.

### Compilación y Ejecución

Ejecutar los diferentes programas indicando en el Visual Studio que se arrancan múltiples proyectos a la vez. Verificar que el servidor del servicio se ejecute primero (deberá estar primero en la lista).

Al correr el programa se podrá observar que trabaja con la funcionalidad básica pero que no maneja ninguna condición de error.

## Laboratorio 10

El proyecto ventas no está completo sin el manejo de las condiciones de error, tanto de negocio como de infraestructura. Para poder manejarlas se debe declarar una excepción diferente a la manejada con ExcepcionVentas ya que esta sirve para procesar en el mismo espacio de memoria. Por lo tanto se creará una nueva excepción que se utilizará para entregar mensajes de condiciones anómalas a los clientes con el fin de mantener separadas las ocupaciones. La clase ExcepcionVentas se seguirá utilizando localmente en el espacio de memoria del servidor y brindará el detalle de lo ocurrido realizando salidas por consola. De esta manera podemos clasificar a los errores según el espacio de memoria en el cual se manejan:

- **Espacio de memoria del servidor del servicio**: maneja las excepciones que se producen en la base de datos y en el servidor
- **Espacio de memoria del cliente gráfico**: maneja las excepciones que se producen remotamente durante el procesamiento del servicio o los errores de comunicación.

Todo esto acarrea como consecuencia analizar cuándo se lanzará una excepción remota y cuando no. Las operaciones que involucran reglas de negocio fueron modificadas de manera que en lugar de lanzar ExcepcionVentas, ahora lo hagan con la excepción remota.

Para poder gestionar las excepciones remotas se deberán realizar las siguientes acciones:

### Creación de la clase que contiene la información de la excepción remota.

1. Seleccionar el proyecto Excepciones y crear la clase ExcepcionClienteRemota.
2. Agregar una referencia a System.Runtime.Serialization
3. Crear las siguientes variables de instancia:

**C#**

```
private string _mensajeDeError = "No Definido";
private String _dni;
private String _nombre;
private String _dir;
```

**VB**

```
Private _mensajeDeError As String
Private _dni As String
Private _nombre As String
Private _dir As String
```

4. A partir de las variables de instancia, crear una propiedad por cada una de las declaradas.
5. Esta clase se serializará, por lo tanto es un contrato de datos y se le deberán colocar los siguientes atributos:
   `DataContract`, antecediendo la declaración de la clase
   `DataMember`, antecediendo la declaración de cada propiedad

### Modificar el contrato del servicio para que maneje fallas

6. Agregar una referencia al proyecto Excepciones

7. Declarar el atributo [FaultContract(typeof(ExcepcionClienteRemota))] o <FaultContract(GetType(ExcepcionClienteRemota))> (C# o VB) antecediendo todas las operaciones de IContratoDelServicioVentas. Colocar el atributo entre OperationContract y la firma del método. Esto indica que cualquiera de los métodos puede lanzar una excepción remota y la información estará contenida en la clase ExcepcionClienteRemota.

### Modificar la gestión de la base de datos

Se trabajará sobre la clase ImplementacionDAOVentas. Separar conceptualmente el manejo de errores de la siguiente manera:

8. Agregar una referencia a System.ServiceModel al proyecto Modelo
9. Los errores de SQL se deberán seguir manejando localmente con ExcepcionVentas
10. Los errores de negocio se manejarán con la excepción remota. Para facilitar el desarrollo, se provee indicaciones en el código de la clase para realizar las modificaciones. Los lugares donde modificar el código está antecedidos por el siguiente comentario:

**C#**

```
//** -----------Excepción Remota-------------------------------
```

**VB**

```
'** -----------Excepción Remota-------------------------------
```

### Manejar excepciones en la implementación del servicio

Existen excepciones que se pueden generar en el servicio que sirvan en el cliente que lo invoque para cumplir alguna regla de negocio.

Si a esto se suma que se pueden atrapar excepciones que tienen como destino al que invocó una operación remota, se dan dos situaciones claras a resolver en la implementación del servicio:

- **Recibir una excepción que tiene que enviarse al cliente**: en este caso se deberá lanzar nuevamente la excepción
- **Generar una excepción para enviar al cliente**: se debe crear una nueva excepción remota

La clase dónde se deben realizar tales modificaciones es ImplementacionServicioVentas

11. Para facilitar el desarrollo, se provee indicaciones en el código de la clase para realizar las modificaciones. Los lugares donde modificar el código está antecedidos por el siguiente comentario:
    ```
    //** -----------Excepción Remota-------------------------------
    ```

### Modificar el cliente gráfico para que muestre el error

Cuando el cliente recibe la información de la excepción remota la debe atrapar para realizar acciones con ella. En el código del formulario se atrapar dicha excepción y mostrar la información pertinente al cliente.

12. Agregar una referencia al proyecto Excepciones.
13. Agregar una referencia a System.ServiceModel

14. Verificar que el controlador vuelve a lanzar las excepciones remotas que recibe para lleguen a la vista. Esto se debe a la modificación del patrón MVC clásico.
15. Para facilitar el desarrollo, se provee indicaciones en el código de la clase para realizar las modificaciones. Los lugares donde modificar el código está antecedidos por el siguiente comentario:

```
//** -----------Excepción Remota------------------------------
```