



DIPLOMATURA EN PROGRAMACION EN .NET
Proyecto Ventas

El Diseño del Proyecto Ventas

Ventas: Iteración 1

Actividades:

Durante la primera iteración del ciclo de desarrollo de software se realizará:

- El análisis de los requerimientos de Ventas: notas del consultor
- Análisis de Ventas: iteración 1 (descubrimiento de los objetos del dominio)
- Análisis arquitectónico: abstracción de los requerimientos
- Aplicación del patrón de diseño MVC: introducción
- Aplicación del patrón de diseño MVC: sección del cliente
- Ejercicio de la iteración 1: diseño de la arquitectura de Ventas

Análisis de los requerimientos de Ventas: notas del consultor

A continuación se presentan las notas de los requerimientos efectuados al consultor para la construcción de la aplicación. Se debe proveer los mismos como parte del pedido de desarrollo

Sentencia del problema

Crear una aplicación para el manejo de stock y clientes por parte de los agentes comerciales, que provea la funcionalidad lista en los siguientes casos de uso y requerimientos. La funcionalidad estará sujeta a las restricciones enumeradas.

Requerimientos

Los requerimientos son clasificados como casos de uso, requerimientos generales y restricciones.

- Casos de uso:
 - Caso de uso 1: crear un nuevo cliente
 - Caso de uso 2: borrar un cliente
 - Caso de uso 3: actualizar los campos de los clientes
 - Caso de uso 4: ver los detalles de los clientes
 - Caso de uso 5: ver la lista de clientes
 - Caso de uso 6: ver la cartera de clientes
 - Caso de uso 7: comprar mercadería de distribución sobre pedido de clientes
 - Caso de uso 8: vender mercadería de distribución sobre pedido de clientes
 - Caso de uso 9: ver los precios del stock
- Requerimientos generales:

Diplomatura en Programación en .Net

- La aplicación deberá ser capaz de soportar múltiples usuarios simultáneamente desde todas las locaciones de la compañía que se encuentran en la intranet
- Todos los casos de uso deberán ser utilizables a través de la interfaz gráfica
- Restricciones:
 - La aplicación debe presentar y actualizar la información almacenada en una base de datos MS SQL Server en la empresa
 - Toda la información presentada deberá reflejarse en tiempo real en el medio de almacenamiento, o sea, la información de la base de datos deberá ser actualizada en línea.
- Casos de prueba:
 - No fueron provistos por el consultor. Deberá considerarse la creación de los mismos si el tiempo lo permite.

Planes de construcción del proyecto: Iteración 2

Los siguientes planes de construcción “sugeridos” son utilizados para construir la aplicación Ventas. Los planes se irán construyendo a lo largo del taller por segmentos

Manejo de la complejidad

Para manejar la complejidad, el consultor recomendó la siguiente estrategia:

- La primera realización del software soporta sólo el segmento del cliente de los requerimientos (casos de uso del 1 al 5)

Como consecuencia, las interfaces creadas IModeloVentas, IVistaVentas e IControladorVentas contienen inicialmente sólo los métodos que soportan los requerimientos del segmento del cliente.
- La primera versión del software consiste en una serie de construcciones:
 - En cada construcción, se crearán los componentes que se agregarán a la construcción previa
 - Deberá verificar individualmente cada componente antes de integrarlo a la construcción previa
 - Se deberá verificar la integración de todos los componentes construidos y anexados a la versión previa antes de pasar a la próxima construcción
- La versión final del software soportará todos los requerimientos de Ventas

Diplomatura en Programación en .Net

Universidad Tecnológica Nacional – Derechos Reservados

Construcción 1

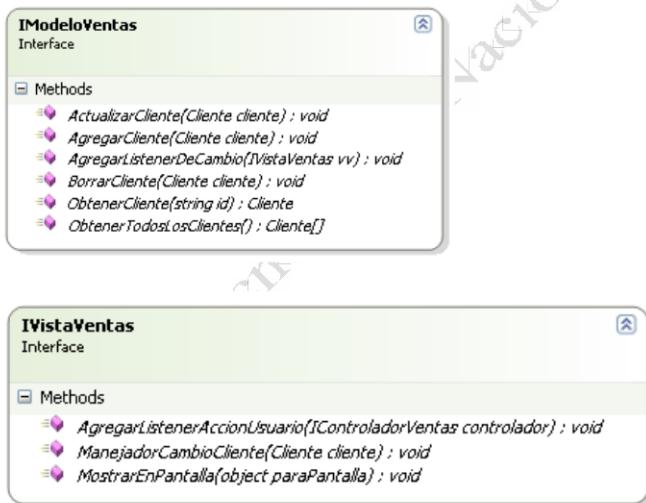
Establecer una línea base para el MVC de Ventas

A continuación se detallan el objetivo, propósito y tareas de la construcción 1

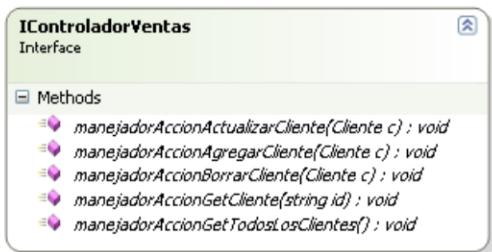
- Objetivo: crear y compilar las interfaces del dominio y todas las clases de soporte identificadas en las tareas de la iteración 1
- Propósito: validar el diseño de las interfaces y las clases
- Tareas:
 - Crear la infraestructura de soporte para el proyecto
 - Realizar el diseño en detalle requerido para las clases e interfaces
 - Escribir el código y compilarlo para las clases e interfaces
 - Verificar la compilación de las clases e interfaces

A continuación se muestran las clases de la construcción 1, comenzando con las interfaces ya generadas para el manejo de la complejidad

Proyecto MVC: clases del proyecto



Diplomatura en Programación en .Net



Descubriendo los objetos del dominio

Durante la fase de análisis para la primera iteración, el consultor identificó y documentó los siguientes objetos de dominio como “posibles”:

- Clase Cliente

Esta clase representa a un único cliente en el dominio de la aplicación

- Clase Stock

Esta clase representa un stock en particular en el dominio de la aplicación

Atributos posibles:

- ID_stock
- Precio

- Clase Orden de Compra:

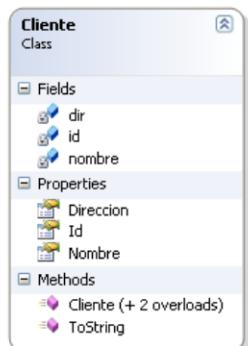
Esta clase representa la información acerca de la mercadería que un cliente posee en un único stock.

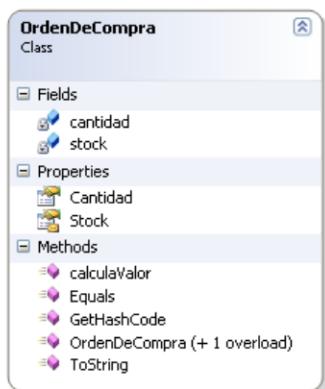
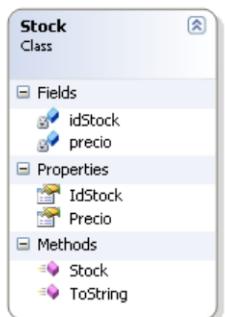
- Clase Cartera:

Esta clase representa la información acerca de todas las mercaderías que un cliente posee. Como consecuencia, esta puede contener un atributo que enlace la cartera con el cliente y un atributo que represente una colección de mercaderías.

Esta es la primera iteración del análisis. En las próximas iteraciones se continúa con el análisis para confirmar las clases identificadas como objetos del dominio requeridos para el diseño de la aplicación. Estas iteraciones de análisis junto con las iteraciones de diseño definen las clases con más detalles.

Proyecto Entidades: clases del proyecto





Manejo de anomalías

El consultor estableció que el manejo de anomalías se debe realizar mediante excepciones propias del proyecto. De esta manera, cualquier situación de error deberá manejarse como una excepción personalizada para ventas. La clase provista a tal fin se encuentra en un proyecto separada para una mejor gestión de la misma.

Proyecto Excepciones: clase del proyecto

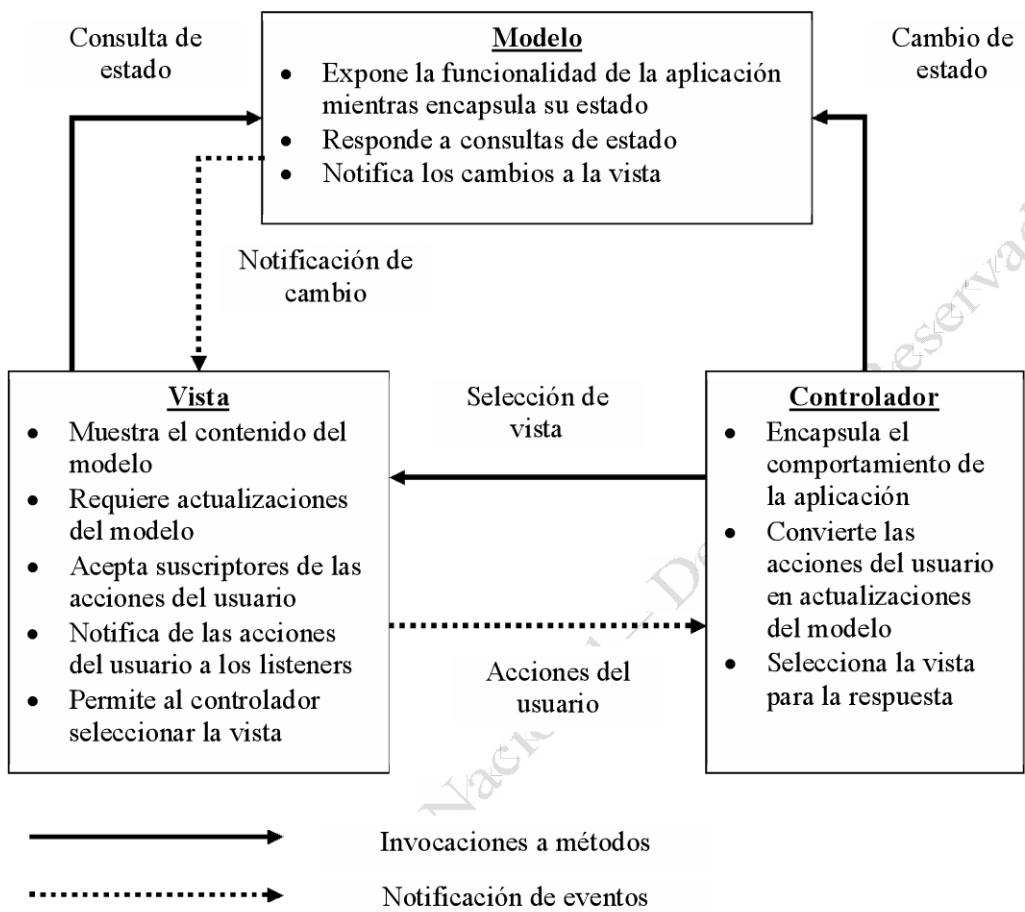


Análisis arquitectónico: abstracción de los requerimientos

Un análisis preliminar de los requerimientos de Ventas indica que la aplicación necesita lo siguiente:

- Una interfaz gráfica que permita:
 - Mostrar por pantalla la información almacenada en la empresa
 - Poder cambiar la información almacenada en la empresa
- Un componente de software asociado con los datos de la empresa que:
 - Contenga la lógica de negocio que valide los cambios requeridos a los datos de la empresa
 - Contenga el acceso a la lógica de negocio que brinde servicios a los requerimientos de información a partir de los datos de la empresa (transformación de datos en información)
- Un componente de software asociado con la interfaz gráfica que contenga la lógica de control específica de esta en la aplicación que incluya:
 - Validación de las acciones del usuario
 - Traslado de los requerimientos del usuario
 - Navegación inteligente para las vistas de usuario

Como se puede observar en la figura, el patrón MVC se ajusta perfectamente a los requerimientos arquitectónicos de la aplicación



Aplicando el patrón de diseño MVC

En el punto anterior se identificó al patrón de diseño MVC como una solución aplicable al proyecto. Esta sección contiene un procedimiento paso a paso que ayuda a aplicar el patrón de diseño MVC. La intención de este procedimiento es la de una guía introductoria. A medida que crezca la experiencia del desarrollador, se invita a modificar y adaptar el procedimiento de manera que mejor satisfaga a los requerimientos según su opinión. Otro elemento de ayuda es la aplicación Info dada sobre el patrón MVC explicada anteriormente. Utilizarlo como modelo

1. Delinear un diagrama de interacción entre los posibles participantes genéricos del modelo MVC para la solución. Asegurarse de utilizar los nombres específicos asignados a los participantes para la aplicación que se está desarrollando. Estos nombres estarán definidos como interfaces en los pasos siguientes de este procedimiento.

Diplomatura en Programación en .Net

Por ejemplo, si se está aplicando el patrón de diseño MVC a una aplicación de manejo de clientes, los participantes podrían estar representados por:

- i. IModeloCliente
 - ii. IVistaCliente
 - iii. IControladorCliente
2. Listar la información encapsulada por el modelo
 3. Listar la información que mostrará por pantalla la vista
 4. Definir las páginas que alberga la vista para mostrar los datos

Por lo general, una vista almacena múltiples páginas para mostrar los datos. Cada página para mostrar los datos consiste en información lógicamente relacionada y controles para ingresos del usuario. Las páginas que se muestran por pantalla al usuario en un determinado momento del tiempo son dependientes del contexto en el que se encuentra la aplicación. La tarea en este punto es poder definir las páginas necesarias para cumplir con la lista del punto anterior. Cada página deberá contener:

- i. Nombre de la página a mostrar por pantalla
- ii. Un bosquejo de cómo se muestra la información en pantalla y los controles que posee para los usuarios

Nota: en este punto del desarrollo de la aplicación, el foco está en el “qué” y no en el “cómo”. Como resultado, en cada página que se muestra por pantalla se define qué información se muestra y qué ingresos del usuario se pueden recoger. Posteriormente se cambiará el foco a como la información es mostrada por pantalla y como son tomados los ingresos del usuario. Esto se determinará en el diseño en detalle y las subsiguientes iteraciones de cada una de las interfaces gráficas

Luego de definir todas las páginas a mostrar por pantalla, documentar como se las navega y en que secuencia, la registración de acciones del usuario y los eventos que disparan los cambios en cada página

5. Definir los métodos soportados por la vista en términos de interfaces

- a. Poner un nombre a la interfaz

Por ejemplo, para la aplicación de manejo de clientes, se puede crear la siguiente interfaz

Diplomatura en Programación en .Net

C#

```
public interface IVistaVentas
```

VB

```
Public Interface IVistaVentas
```

- b. Especificar los métodos de utilidad para los requerimientos de registración de los suscriptores a las acciones del usuario que hagan los controladores.

Se necesita un método que controle todas las acciones del usuario que son manejadas por la vista. Se puede considerar el siguiente formato para especificar la registración de los listeners a las acciones del usuario:

C#

```
void AgregarListenerAccionUsuario(IControladorVentas  
controlador);
```

VB

```
Sub AgregarListenerAccionUsuario(ByVal controlador As _  
IControladorVentas)
```

- c. Especificar los métodos de utilidad para los comandos de la página a mostrar por pantalla solicitados por el controlador

Al menos se necesita uno de estos métodos para mostrar por pantalla todos los comandos que envíe el controlador. Por ejemplo, se utilizar el siguiente formato:

C#

```
void MostrarEnPantalla(Object paraPantalla);
```

VB

```
Sub MostrarEnPantalla(ByVal paraPantalla As Object)
```

- d. Especificar métodos de callback (llamada por retorno debido a una acción del programa o sistema operativo) de utilidad para los eventos de cambio de estado que son generados por el modelo

Por lo menos se necesita uno de estos métodos para manejar todos los cambios de estado. Por ejemplo, se puede utilizar el siguiente formato:

C#

```
void ManejadorCambioCliente(Cliente cliente);
```

VB

```
Sub ManejadorCambioCliente(ByVal cliente As Cliente)
```

Diplomatura en Programación en .Net

En muchas oportunidades, se necesitará más de un método para manejar los cambios de estado. En esos casos, se puede utilizar los siguientes formatos para los métodos de call back:

C#

```
void ManejadorDeCambioXXX(VistaCliente vc)
void ManejadorDeCambioYYY(VistaCliente vc)
```

VB

```
Sub ManejadorDeCambioXXX(ByVal vc As VistaCliente)
Sub ManejadorDeCambioYYY(ByVal vc As VistaCliente)
```

Nota: aún si la vista contiene una sola página para mostrar por pantalla, se deberá especificar un método de utilidad para mostrarla. Construir este tipo de infraestructura ayuda a acomodar futuras expansiones de la vista para contener muchas diferentes páginas a mostrar

6. Definir las utilidades o servicios soportados por el modelo en términos de la tecnología:

- a. Nombrar una interfaz

Por ejemplo, para una aplicación de manejo de clientes se deberá crear la siguiente interfaz:

C#

```
public interface IModeloVentas
```

VB

```
Public Interface IModeloVentas
```

- b. Especificar los métodos de servicio o utilidad para los requerimientos de registración de los suscriptores de cambio de estado del modelo para la vista

Por lo menos se necesita uno de estos métodos para manejar todos los cambios de estado. En esos casos, se puede utilizar los siguientes formatos para los métodos de call back:

C#

```
void AgregarListenerDeCambioXXX(VistaCliente vc)
void AgregarListenerDeCambioYYY(VistaCliente vc)
```

VB

```
Sub AgregarListenerDeCambioXXX(ByVal vc As VistaCliente)
Sub AgregarListenerDeCambioYYY(ByVal vc As VistaCliente)
```

Diplomatura en Programación en .Net

- c. Especificar los métodos de servicio o utilidad para los comandos de cambio de estado que genere el controlador.

Estos son esencialmente los métodos de negocio para la aplicación que contiene el modelo. Por ejemplo

C#
void AgregarCliente(Cliente cliente);
void BorrarCliente(Cliente cliente);

VB
Sub AgregarCliente(ByVal cliente As Cliente)
Sub BorrarCliente(ByVal cliente As Cliente)

- d. Especificar los métodos de servicio o utilidad para los requerimientos de cambio de estado del modelo por la vista

Estos son métodos de acceso (accessors) que retornan los datos encapsulados por el modelo. Por ejemplo:

C#
Cliente ObtenerCliente(String id);
Cliente[] ObtenerTodosLosClientes();

VB
Function ObtenerCliente(ByVal id As String)
Function ObtenerTodosLosClientes() As Cliente()

7. Definir los servicios soportados por el controlador en términos de la tecnología de .Net

- a. Nombrar una interfaz

Por ejemplo, para una aplicación de manejo de clientes se deberá crear la siguiente interfaz:

C#
public interface IControladorVentas

VB
Public Interface IControladorVentas

- b. Especificar métodos de callback (llamada por retorno debido a una acción del programa o sistema operativo) de utilidad para los eventos de acciones del usuario que son generados por la vista

Estos métodos manejan los eventos que se disparan por las acciones del usuario que se generan en la vista

Diplomatura en Programación en .Net

En esos casos, se puede utilizar los siguientes formatos para especificar los métodos de registro de suscriptores de acciones del usuario:

```
public void manejadorAccionXXX(EventoDeDatosXXX evento)
```

```
public void manejadorAccionYYY(EventoDeDatosYYY evento)
```

C#

```
void manejadorAccionXXX(DatosEvento evento);  
void manejadorAccionYYY(DatosEvento evento);
```

VB

```
Sub ManejadorAccionXXX(ByVal evento As DatosEvento)  
Sub ManejadorAccionYYY(ByVal evento As DatosEvento)
```

8. Revisar los participantes del patrón MVC

Hasta el momento se han definido los participantes del modelo MVC utilizando interfaces de .Net. La tarea en este punto es la de revisar estas interfaces nuevamente para verificar su aplicación a los casos de uso más relevantes identificados en la fase de análisis del proyecto.

- a. Verificar que el modelo provee un método de acceso (accessor) o propiedad para cada dato que se muestre en pantalla
- b. Verificar que el modelo provea un método de negocio para cada acción del usuario que genere un cambio de estado en el modelo
- c. Verificar que cada interfaz provee los servicios requeridos por las otras interfaces de los participantes del patrón MVC

9. Documentar todos los tipos de datos

En este punto, la tarea es identificar, corroborar y documentar todos los datos y sus tipos de todos los objetos participantes.

a. Identificar los objetos del dominio

Estos objetos fueron identificados previamente al análisis arquitectónico. Verificar que las interfaces utilicen los objetos del dominio. A través de examinar los métodos de las interfaces que utilizan los objetos del dominio, se puede clarificar los atributos y métodos que los mismos deberán poseer.

b. Identificar nuevos objetos

Examinar los parámetros de los métodos, los tipos retornados y las excepciones lanzadas por los métodos cuando se implementan las interfaces participantes del

Diplomatura en Programación en .Net

patrón MVC para descubrir nuevos objetos. Documentar estos descubrimientos y definir sus atributos y responsabilidades.

10. Verificar la integridad de los objetos del dominio, las interfaces de los participantes de MVC y los objetos soportados

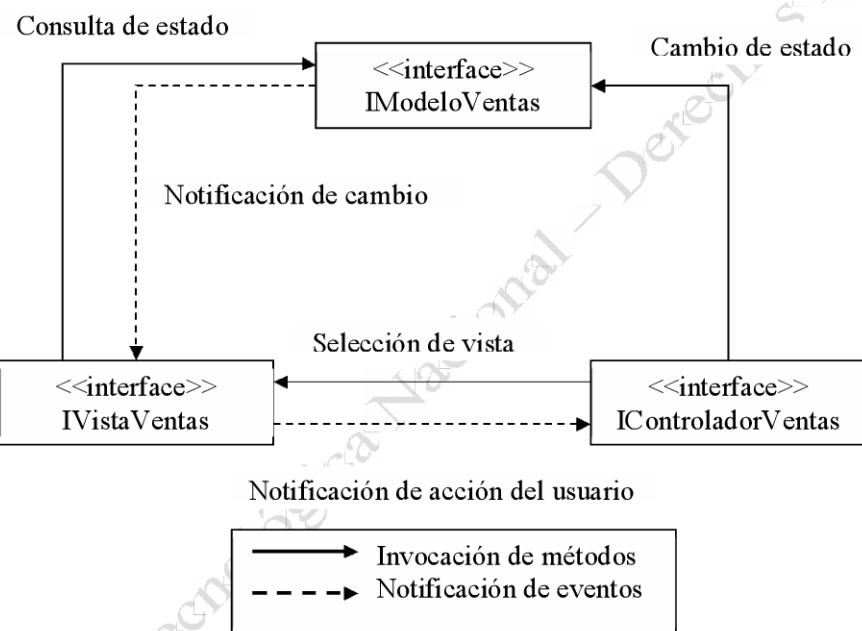
Esta etapa involucra crear los correspondientes archivos fuentes de código .Net (C# o VB) y verificar su correcta compilación. Se debe tener cuidado y en cuenta que no todo el dominio, los objetos de este o las clases soportadas, pueden estar definidos en este punto. Si se tiene conocimiento de las necesidades pero no se conocen los detalles, se pueden utilizar clases vacías o abstractas para definirlos hasta conocer a fondo sus atributos y responsabilidades. En caso de no conocer estos elementos, se debe tener en cuenta que el diseño sea lo suficientemente flexible (dentro de las posibilidades) para poder crecer. Una precaución que brinda mucha flexibilidad respecto del crecimiento es la utilización de interfaces.

Aplicando el modelo MVC

Segmento Cliente

Esta sección demuestra la aplicación del procedimiento de análisis para el segmento del cliente en la aplicación Ventas. No contiene el diseño MVC para el segmento Cartera o el segmento Stock de la aplicación. Este análisis, el del segmento Cartera y el del segmento Stock, se deja como ejercicio para realizar.

1. Bosquejar un diagrama de interacción como el que se muestra a continuación, pero utilizar sus propios nombres específicos para la aplicación a construir para cada uno de los participantes de MVC



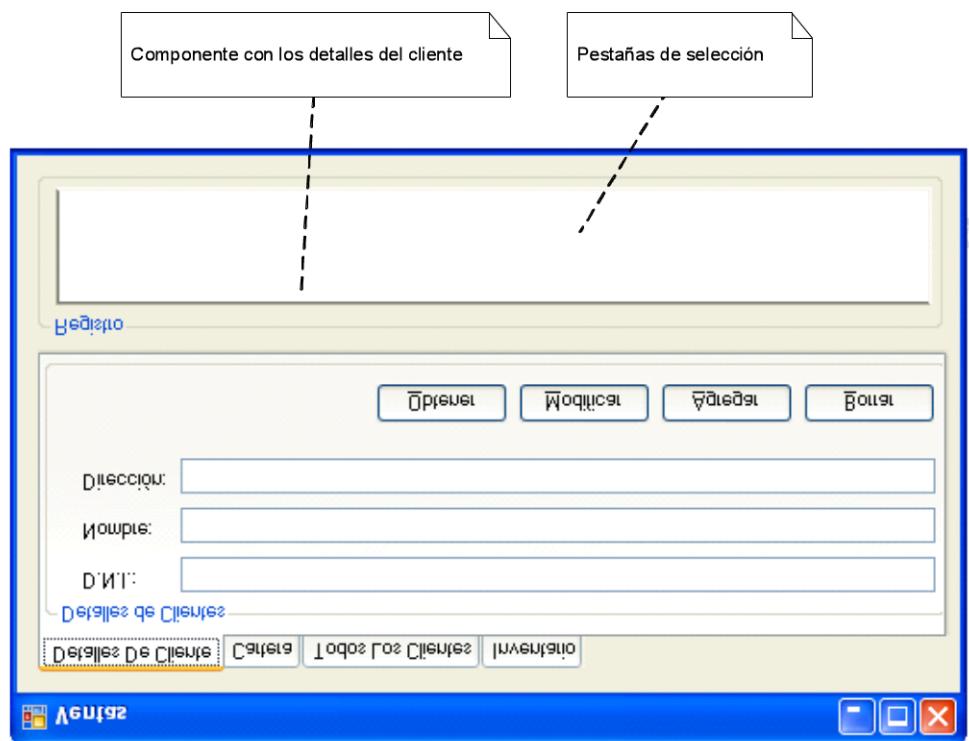
2. Listar la información encapsulada por el modelo
 - a. El modelo de ventas encapsula la siguiente información:
 - i. Identificación del cliente (DNI). Valor único
 - ii. Nombre del cliente
 - iii. Dirección del cliente
3. Listar la información mostrada por la vista
 - a. Listar la información del cliente

Diplomatura en Programación en .Net

- i. Identificación del cliente (DNI). Valor único
 - ii. Nombre del cliente
 - iii. Dirección del cliente
 - b. Listar todos los clientes
4. Definir las páginas que se mostrarán por pantalla manejadas por la vista
- a. Definir la página de información del cliente. Información a mostrar:
 - i. Identificación del cliente (DNI). Valor único
 - ii. Nombre del cliente
 - iii. Dirección del cliente
- Controles para el usuario:
- iv. Especificar los controles de selección como los siguientes botones:
 - 1. Agregar Cliente
 - 2. Obtener Cliente
 - 3. Actualizar Cliente
 - 4. Borrar Cliente

Esto se muestra en la figura

Diplomatura en Programación en .Net



- v. Mostrar en pantalla las pestañas de selección como los siguientes nombres (ver figura anterior):

1. Detalles De Clientes
2. Cartera
3. Todos los clientes
4. Inventario

- b. Definir la información de la página para todos los clientes

- i. Información a mostrar:

1. Lista de todos los clientes

Diplomatura en Programación en .Net

- ii. Especificar el control que mostrará en una grilla a todos los usuarios (la

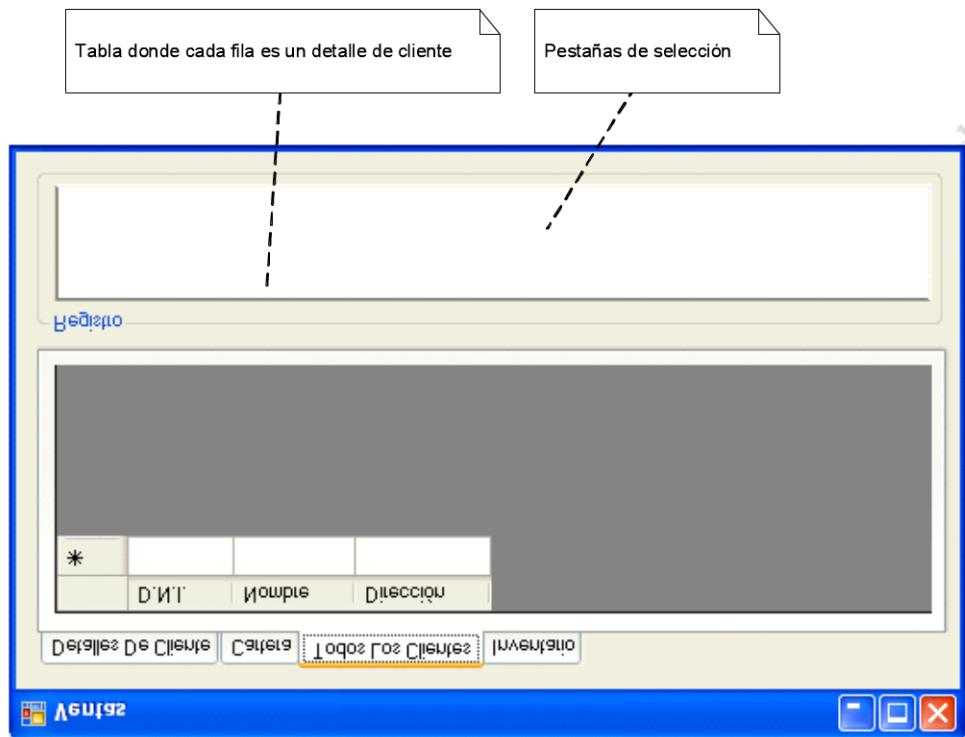


figura a continuación muestra la grilla):

5. Definir los métodos de servicio soportados por la vista como interfaces. Examinar el código que se muestra a continuación y observar los siguientes puntos:
- a. El nombre de la interfaz
 - b. Los métodos de servicio para los requerimientos de registro de los controladores de listeners para las acciones del usuario
 - c. Los métodos de servicio para los comandos de mostrar página por pantalla pedido por el controlador
 - d. Los métodos de servicio para los eventos de cambio de estado generados por el modelo

C#

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
```

Diplomatura en Programación en .Net

```
5
6 namespace Ventas
7 {
8     public interface IVistaVentas
9     {
10         //Métodos para registrar los listeners de acciones del usuario
11         /* agregar requirentes a la lista de objetos a ser notificados de
12          * las acciones del usuario ingresadas a través de una interfaz como la GUI
13          * Las acciones del usuario para el segmento del cliente son agregar,
14          * borrar, actualizar, obtener y obtener todos los clientes. Hay acciones
15          * del usuario similares para los segmentos de cartera y stock
16          */
17         void AgregarListenerAcciónUsuario(IControladorVentas controlador);
18
19         //Métodos de utilidad para mostrar en pantalla un requerimiento de
20         //selección del vendedor
21         /*
22          * muestra por pantalla la página especificada por el controlador
23          */
24         void MostrarEnPantalla(Object paraPantalla);
25
26         // iteracion 1 -- Métodos del segmento del Cliente para la vista de los vendedores
27         /*
28          * -----Métodos de callback para manejar la notificación de los cambios
29          * de estado del cliente para el modelo Ventas
30          */
31         void ManejadorCambioCliente(Cliente cliente);
32
33         // Segmento Cartera - Se completará en una iteración futura
34         // Agregar un método para manejar las notificaciones de cambio de cartera
35         // desde el modelo Ventas
36
37         // Segmento Stock - Se completará en una iteración futura
38         // Agregar un método para manejar las notificaciones de cambio de stock
39         // desde el modelo Ventas
40
41     }
42 }
```

VB

```
1. Imports EntidadesVB
2. Public Interface IVistaVentas
3. ' Métodos para registrar los suscriptores de acciones del usuario
4. ' agregar requirentes a la lista de objetos a ser notificados de
5. ' las acciones del usuario ingresadas a través de una interfaz como la GUI
6. ' Las acciones del usuario para el segmento del cliente son agregar,
7. ' borrar, actualizar, obtener y obtener todos los clientes. Hay acciones
8. ' del usuario similares para los segmentos de cartera y stock
9.
10. Sub AgregarListenerAcciónUsuario(ByVal controlador As IControladorVentas)

11. ' Métodos de utilidad para mostrar en pantalla un requerimiento de
12. ' selección del vendedor
13. ' -----
14. ' muestra por pantalla la información especificada por el controlador
15. '
16. Sub MostrarEnPantalla(ByVal paraPantalla As Object)

17. ' iteracion 1 -- Métodos del segmento del Cliente para la vista de los vendedores
18. ' -----
19. ' Métodos de callback para manejar la notificación de los cambios
20. ' de estado del cliente para el modelo de Ventas
```

Diplomatura en Programación en .Net

```
21.  
22. Sub ManejadorCambioCliente(ByVal cliente As Cliente)  
  
23. ' Segmento Cartera - Se completará en una iteración futura  
24. ' Agregar un método para manejar las notificaciones de cambio de cartera  
25. ' desde el modelo Ventas  
  
26. ' Segmento Stock - Se completará en una iteración futura  
27. ' Agregar un método para manejar las notificaciones de cambio de stock  
28. ' desde el modelo Ventas  
29. End Interface
```

6. Definir los servicios soportados por el modelo como interfaces de .Net. Examinar el código a continuación y observar lo siguiente:

- a. El nombre de la interfaz
- b. Los métodos de servicio para los requerimientos de registro de listeners de cambio de estado del modelo por la vista
- c. Los métodos de servicio para los comandos de cambio de estado generados por el controlador
- d. Los métodos de servicio para requerimientos de consulta de estado del modelo realizados por la vista

C#

```
1. using System;  
2. using System.Collections.Generic;  
3. using System.Linq;  
4. using System.Text;  
5.  
6. namespace Ventas  
7. {  
8.     public interface IModeloVentas  
9.     {  
10.         // Métodos para registrar listeners de cambio de estado para  
11.         // el modelo del Ventas  
12.         /* -----  
13.             * Agregar un observador a la lista de objetos que serán notificados  
14.             * cuando un objeto(Cliente, Cartera o Stock) del modelo de Ventas  
15.             * altera su estado  
16.         */  
17.         void AgregarListenerDeCambio(IVistaVentas vv);  
18.  
19.         // iteración 1 Métodos del segmento Cliente para el modelo de Ventas  
20.         // Métodos del segmento Cliente de cambio de estado  
21.         /**-----  
22.             * Agregar el Cliente al modelo Ventas  
23.             */  
24.         void AgregarCliente(Cliente cliente);  
25.  
26.         /**-----  
27.             * Borrar el cliente del modelo Ventas  
28.             */  
29.         void BorrarCliente(Cliente cliente);  
30.  
31.         /**-----
```

Diplomatura en Programación en .Net

```
32.          * Actualizar el cliente en el modelo Ventas
33.          */
34. void ActualizarCliente(Cliente cliente);
35.
36. // Métodos del segmento Cliente para consulta del estado
37. /*****
38.     * Dado un dni, retorna el cliente del modelo
39.     */
40. Cliente ObtenerCliente(String id);
41.
42.
43. /*****
44.     * Retorna todos los clientes en el modelo Ventas
45.     */
46. Cliente[] ObtenerTodosLosClientes();
47.
48. // Segmento Cartera - Se completará en una iteración futura
49. // Agregar métodos de cambio de estado al segmento Cartera
50. // Agregar métodos de consulta de estado al segmento Cartera
51.
52. // Segmento Stock - Se completará en una iteración futura
53. // Agregar métodos de cambio de estado al segmento Stock
54. // Agregar métodos de consulta de estado al segmento Stock
55. }
56. }
```

VB

```
1. Imports Entidades
2. Public Interface IModeloVentas
3. ' Métodos para registrar listeners de cambio de estado para
4. ' el modelo de Ventas
5. ' -----
6. ' Agregar un suscriptor a la lista de objetos que serán notificados
7. ' cuando un objeto(Cliente, Cartera o Stock) del modelo de Ventas
8. ' altera su estado
9.
10. Sub AgregarListenerDeCambio(ByVal vv As IVistaVentas)
11. ' iteración 1 Métodos del segmento Cliente para el modelo de Ventas
12. ' Métodos del segmento Cliente de cambio de estado
13. ' -----
14. ' Agregar el Cliente al modelo Ventas
15. '
16. Sub AgregarCliente(ByVal cliente As Cliente)
17. ' -----
18. ' Borrar el cliente del modelo Ventas
19. '
20. Sub BorrarCliente(ByVal cliente As Cliente)
21. ' -----
22. ' Actualizar el cliente en el modelo Ventas
23. '
24. Sub ActualizarCliente(ByVal cliente As Cliente)
25. ' Métodos del segmento Cliente para consulta del estado
26. ' -----
27. ' Dado un dni, retorna el cliente del modelo
28. '
29. Function ObtenerCliente(ByVal id As String) As Cliente
30. ' -----
31. ' Retorna todos los clientes en el modelo Ventas
32. '
33. Function ObtenerTodosLosClientes() As Cliente()
34. ' Segmento Cartera - Se completará en una iteración futura
```

Diplomatura en Programación en .Net

```
35. ' Agregar métodos de cambio de estado al segmento Cartera
36. ' Agregar métodos de consulta de estado al segmento Cartera
37. ' Segmento Stock - Se completará en una iteración futura
38. ' Agregar métodos de cambio de estado al segmento Stock
39. ' Agregar métodos de consulta de estado al segmento Stock
40. End Interface
```

7. Definir los servicios soportados por el controlador en como interfaces de .Net. Examinar el código a continuación y verificar lo siguiente:

- a. El nombre de la interfaz
- b. Los métodos de callback de servicio para los eventos de acciones del usuario generados por la vista

C#

```
1. using System;
2. using System.Collections.Generic;
3. using System.Text;
4.
5. namespace Ventas
6. {
7.     public interface IControladorVentas
8.     {
9.         //      métodos de call back para las acciones del usuario
10.        // -----
11.        /** Obtener el Cliente por el método manejador de la acción del usuario
12.         * llamado por la vista de ImplVistaVentas en respuesta al clic del botón
13.         * para obtener el cliente en la interfaz gráfica
14.         * o su equivalente en la interfaz del usuario
15.         * acción - indicar el tipo presentación en pantalla para el cliente
16.         * en la interfaz gráfica a través del método mostrarEnPantalla de VistaVentas
17.         */
18.        void manejadorAcciónGetCliente(String id);
19.
20.
21.        // -----
22.        /** Agregar el Cliente por el método manejador de la acción del usuario
23.         * llamado por ImplVistaVentas en respuesta al clic sobre el botón para
24.         * agregar cliente en la interfaz gráfica o su equivalente en la
25.         * interfaz del usuario
26.         * acción - agregar el (nuevo) cliente del modelo
27.         */
28.        void manejadorAcciónAgregarCliente(Cliente c);
29.
30.        // -----
31.        /** Borrar el Cliente por el método manejador de la acción del usuario
32.         * llamado por ImplVistaVentas en respuesta al clic sobre el botón para
33.         * borrar cliente en la interfaz gráfica o su equivalente en la
34.         * interfaz del usuario
35.         * acción - borrar el cliente del modelo
36.         */
37.        void manejadorAcciónBorrarCliente(Cliente c);
38.
39.        // -----
40.        /** Actualizar el Cliente por el método manejador de la acción del usuario
41.         * llamado por ImplVistaVentas en respuesta al clic sobre el botón para
42.         * actualizar cliente en la interfaz gráfica o su equivalente en la
43.         * interfaz del usuario
```

Diplomatura en Programación en .Net

```
44.         * acción - actualizar el cliente del modelo
45.         */
46.         void manejadorAccionActualizarCliente(Cliente c);
47.
48.
49.         // -----
50.         /** Obtener todos los Clientes por el método manejador de la acción del
51.          * usuario llamado por la vista de ImplVistaVentas en respuesta al clic del
52.          * botón para obtener todos los clientes en la interfaz gráfica
53.          * o su equivalente en la interfaz del usuario
54.          * acción - indicar el tipo presentación en pantalla para todos los clientes
55.          * en la interfaz gráfica a través del método mostrarEnPantalla de
      VistaVentas
56.          */
57.         void manejadorAccionGetTodosLosClientes();
58.
59.         // Segmento de Cartera - se realizará en una iteración futura
60.         // Agregar un método para manejar las notificaciones de las acciones del
61.         // usuario relacionadas con la cartera desde VistaVentas
62.         //Segmento de Stock - se realizará en una iteración futura
63.         //Agregar un método para manejar las notificaciones de las acciones del
64.         // usuario relacionadas con el stock desde VistaVentas
65.     }
66. }
```

8. Revisar los participantes de MVC

- a. Verificar que el modelo de ventas provee métodos de acceso para toda la información que se mostrará por pantalla en las páginas
- b. Verificar que el modelo de ventas provee métodos de negocio para cada acción que el usuario que genere un cambio de estado en el modelo
- c. Verificar que cada interfaz provee los servicios requeridos por los otros dos participantes del patrón MVC

9. Documentar todos los tipos de datos

- a. El objeto de dominio Cliente es requerido
- b. Un objeto nuevo, el ExcepcionVentas fue identificado

Se puede utilizar la clase ExcepcionVentas para reportar todas las condiciones de excepción relacionadas con Ventas, como por ejemplo las condiciones de excepción “cliente duplicado” o “registro no encontrado”

10. Verificar la integridad de los objetos del dominio, las interfaces participantes en MVC y los objetos soportados. Este paso involucra crear y compilar lo siguiente:

- a. MVC: IModeloVentas, IVistaVentas y IControladorVentas
- b. Clases del dominio: Cliente
- c. Clases de soporte: ExcepcionVentas

Laboratorio 3



Verificar los proyectos y clases que componen la solución ventas