Proyecto integrador

El proyecto integrador es un ejercicio desarrollado en varias fases que permite poner en práctica los conceptos tratados en clase.

El objetivo es desarrollar una aplicación del tipo ABM (altas­bajas­modificaciones), totalmente orientada a objetos y estructurada en capas.

Esta aplicación consta de un cliente de escritorio (Windows Forms) que gestiona la venta telefónica de productos.

El proceso de negocio comienza cuando un cliente solicita la compra de uno o más productos, supuestamente disponibles a través de un catálogo impreso. Los clientes pueden ser individuos particulares o empresas.

El operador telefónico (que será el usuario de la aplicación) deberá poder recuperar los datos del cliente, si éste ya ha comprado alguna vez o en su defecto podrá registrar los datos necesarios para efectuar la transacción.

Además, deberá lógicamente registrar los productos pedidos por el cliente accediendo a una lista de ellos. La lista podrá ser filtrada por categoría y subcategoría.

El I.V.A. se considerará el mismo para todos los productos.

El pedido se registrará en una lista de compras (carro de compras) editable.

Una vez confirmada la compra, se emitirá una factura.

Importante

El integrador que estamos realizando se encuentra en fase beta, con lo cual es posible que el mismo sufra modificaciones. Intentaremos que estas modificaciones no entorpezcan el desarrollo normal del ejercicio.

Fase 01

Introducción

En esta fase comenzaremos a crear algunos de los objetos que gestionará nuestra aplicación.

Estos objetos, a los que llamaremos entidades, funcionarán como si fuesen tipos de datos personalizados.

Los objetos a codificar por el momento serán Producto, Proveedor, ClienteIndividuo, ClienteEmpresa, Vendedor y Factura.

Habilidades y conocimientos aplicados en esta fase

● Crear soluciones y proyectos en Visual Studio.

● Comprender la arquitectura basada en capas.

● Crear clases.

● Codificar atributos de lectura/escritura como propiedades autoimplementadas.

● Codificar atributos de sólo lectura.

● Comprender el concepto de visibilidad.

● Crear constructores parametrizados.

● Crear referencias e importar espacios de nombres.

● Instanciar objetos.

● Uso básico de la consola.

Paso 1

Crear un nuevo proyecto del tipo consola. Visual Studio creará un proyecto y una solución.

Guardar todo y nombrar Consola al proyecto e Integrador a la solución.

Paso 2

Agregar a la solución un proyecto del tipo librería de clases y llamarlo Entidades.

Paso 3

Dentro del proyecto Entidades, crear las clases que se detallan a continuación y codificar sus atributos como propiedades autoimplementadas, con excepción de los atributos de sólo lectura, en los cuales deberá codificarse el método Get().

|  |  |  |
| --- | --- | --- |
| Clase: Producto | | |
| Propiedad | Tipo de dato | Acceso |
| Nombre | string | lectura/escritura |
| Descripción | string | lectura/escritura |
| Preciocosto | decimal | lectura/escritura |
| Margen | double | lectura/escritura |
| Iva | double | lectura/escritura |
| Preciobruto | decimal | sólo lectura |
| Precioventa | decimal | sólo lectura |
| Proveedor | string | lectura/escritura |
| Categoría | string | lectura/escritura |
| Subcategoría | string | lectura/escritura |

● El IVA y el margen de ganancia se considerará en forma decimal. Por ejemplo, 0.21 equivale a un IVA del 21%.

● PrecioBruto = PrecioCosto+ Margen

● PrecioVenta = PrecioBruto + IVA

● Verificar que tanto los atributos como las clases sean públicas.

|  |  |  |
| --- | --- | --- |
| Clase: Proveedor | | |
| Propiedad | Tipo de dato | Acceso |
| Nombre | string | lectura/escritura |
| Cuit | string | lectura/escritura |
| Contacto | string | lectura/escritura |
| Email | string | lectura/escritura |
| Teléfono | string | lectura/escritura |
| Dirección | string | lectura/escritura |

|  |  |  |
| --- | --- | --- |
| Clase: ClienteIndividuo | | |
| Propiedad | Tipo de dato | Acceso |
| Nombre | string | lectura/escritura |
| Apellido | string | lectura/escritura |
| Cuit | string | lectura/escritura |
| Email | string | lectura/escritura |
| Teléfono | string | lectura/escritura |
| Dirección | string | lectura/escritura |

|  |  |  |
| --- | --- | --- |
| Clase: Vendedor | | |
| Propiedad | Tipo de dato | Acceso |
| Nombre | string | lectura/escritura |
| Apellido | string | lectura/escritura |
| Dni | string | lectura/escritura |
| Email | string | lectura/escritura |
| Teléfono | string | lectura/escritura |
| Dirección | string | lectura/escritura |

|  |  |  |
| --- | --- | --- |
| Clase: ClienteEmpresa | | |
| Propiedad | Tipo de dato | Acceso |
| Nombre | string | lectura/escritura |
| Cuit | string | lectura/escritura |
| Contacto | string | lectura/escritura |
| Email | string | lectura/escritura |
| Teléfono | string | lectura/escritura |
| Dirección | string | lectura/escritura |

|  |  |  |
| --- | --- | --- |
| Clase: Factura | | |
| Propiedad | Tipo de dato | Acceso |
| Tipo | string | lectura/escritura |
| Numero | string | lectura/escritura |
| Fecha | date | lectura/escritura |
| Cliente | string | lectura/escritura |
| Dirección | string | lectura/escritura |
| Condicioniva | string | lectura/escritura |
| Condicionventa | string | lectura/escritura |
| Detalle | string | lectura/escritura |
| Total | decimal | lectura/escritura |

Paso 4

Crear constructores parametrizados para todas las clases, de tal forma que sea posible instanciarlas únicamente asignando valores a todos sus atributos.

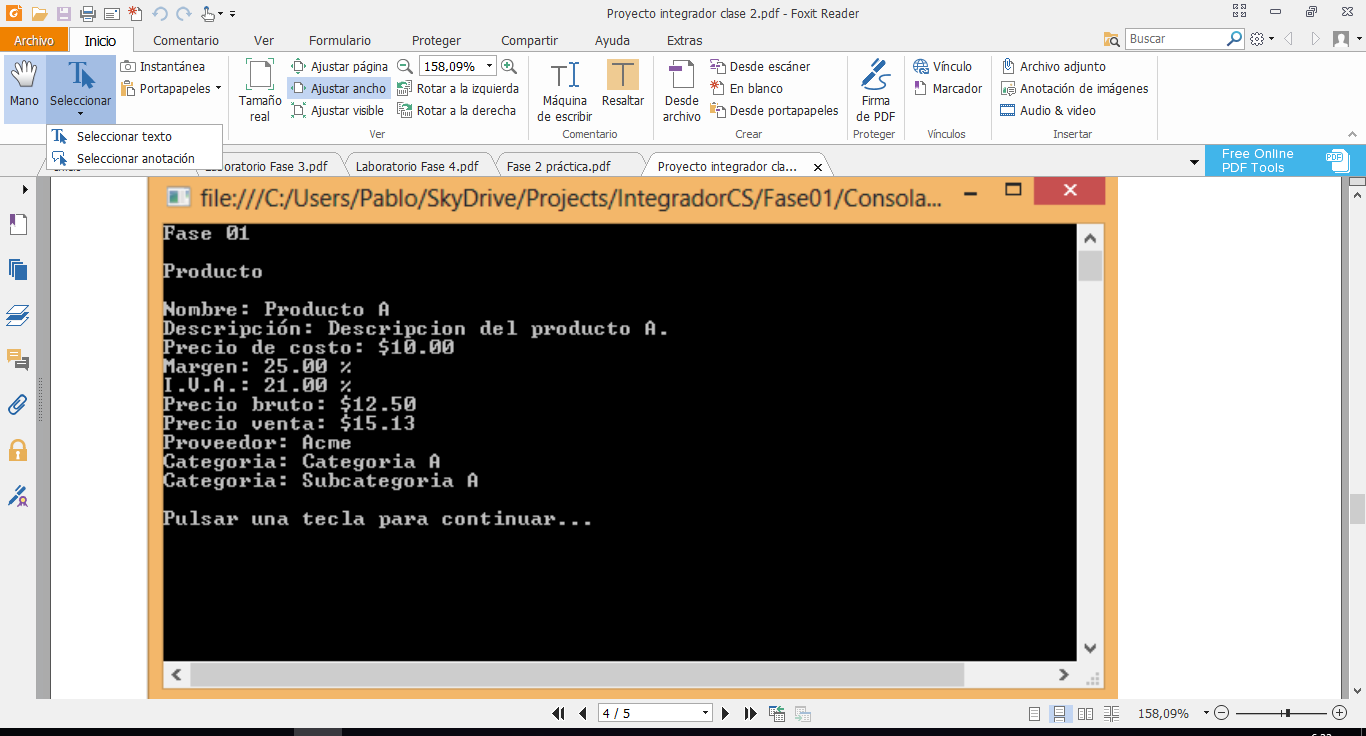
Paso 5

Crear una referencia desde Consola a Entidades. Importar el espacio de nombres Entidades en la consola.

Paso 6

Dentro del proyecto Consola, instanciar las clases, asignarles valores a sus atributos y mostrarlos por pantalla.

El resultado final debería ser similar al de la siguiente figura:



Fase 02

Introducción

Si analizamos las clases creadas en la fase anterior teniendo en cuenta el concepto de herencia, vemos que es posible reformular algunas de éstas.

Por ejemplo, ClienteIndividuo y Vendedor podrían tener en común los siguientes atributos: nombre, apellido, email, teléfono y dirección.

Podemos crear entonces una clase que contenga los atributos comunes y hacer que ambas hereden de ésta.

Por otro lado, impediremos que la clase Persona sea instanciada en forma directa.

Finalmente, si analizamos las clases Proveedor y ClienteEmpresa vemos que las mismas comparten exactamente los mismos atributos. Por lo tanto, no tiene sentido que existan 2 clases diferentes que representen los mismos datos. Por ello la unificaremos en una única clase llamada Empresa.

Habilidades y conocimientos aplicados en esta fase

● Comprender el concepto de herencia.

● Codificar relaciones de herencia.

● Comprender el concepto de clase abstracta.

● Codificar clases abstractas.

Paso 1

Crear en Entidades una nueva clase y llamarla Persona. Esta clase tendrá los siguientes atributos:

|  |  |  |
| --- | --- | --- |
| Clase: Persona | | |
| Propiedad | Tipo de dato | Acceso |
| nombre | String | lectura/escritura |
| apellido | String | lectura/escritura |
| email | String | lectura/escritura |
| telefono | String | lectura/escritura |
| direccion | String | lectura/escritura |

Paso 2

En las clases ClienteIndividuo y Vendedor realizar las modificaciones necesarias para que éstas hereden de la clase Persona. Esta última se declarará abstracta.

● La clase Persona necesitará un constructor por defecto para que las demás clases puedan heredar de ella.

Paso 3

Crear en Entidades una nueva clase y llamarla DocumentoComercial. Esta clase tendrá los siguientes atributos:

|  |  |  |
| --- | --- | --- |
| Clase: DocumentoComercial | | |
| Propiedad | Tipo de dato | Acceso |
| Numero | string | lectura/escritura |
| Fecha | Date | lectura/escritura |
| Cliente | String | lectura/escritura |
| Dirección | String | lectura/escritura |
| Condicioniva | String | lectura/escritura |
| Condicionventa | String | lectura/escritura |
| Detalle | String | lectura/escritura |
| Total | Decimal | lectura/escritura |

Paso 4

En las clases Factura y Remito (clase nueva) realizar las modificaciones necesarias para que éstas hereden de la clase DocumentoComercial. Esta última se declarará abstracta.

● La clase DocumentoComercial necesitará un constructor por defecto para que las demás clases puedan heredar de ella.

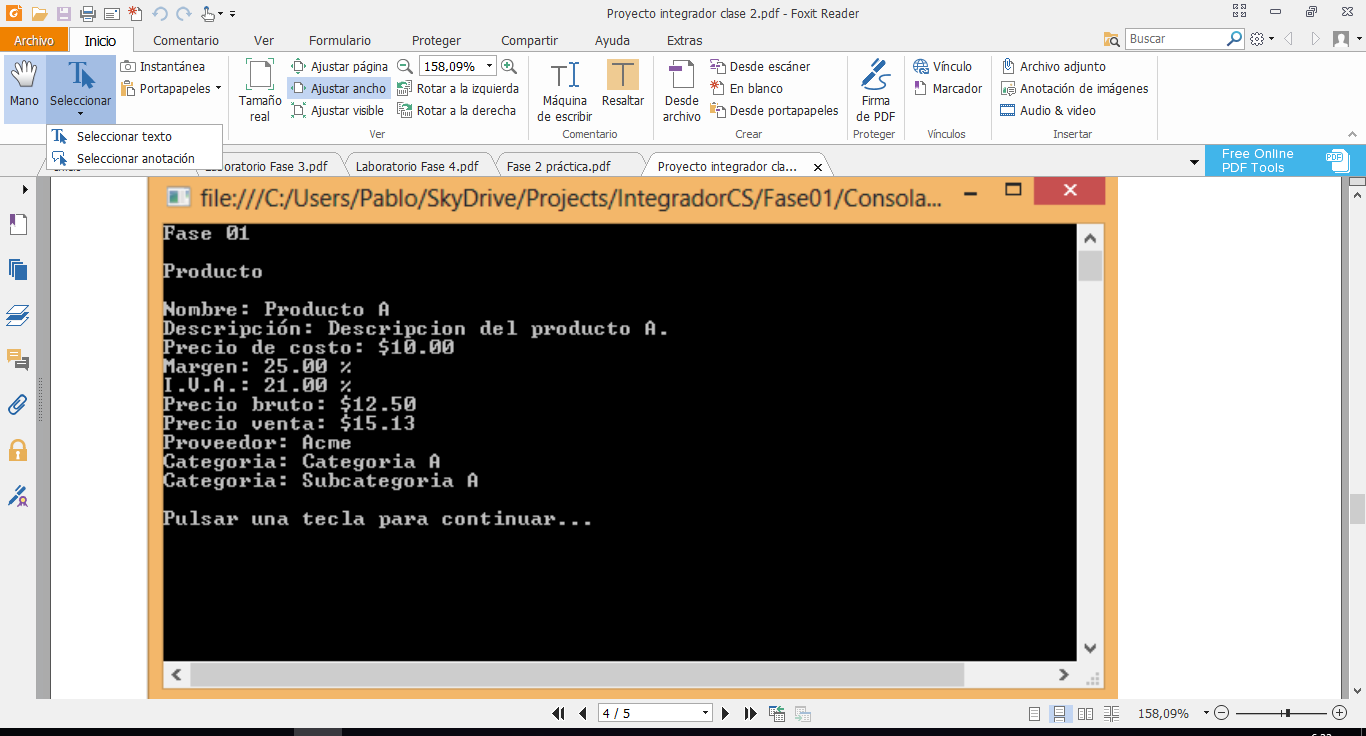
Paso 5

Unificar las clases Proveedor y ClienteEmpresa en una clase denominada Empresa.

Paso 6

En la consola, corregir el error que se ha producido al eliminar las clases Proveedor y ClienteEmpresa.

El resultado final debe ser idéntico al de la fase anterior:



Fase 03

Introducción

En la fase anterior comenzamos a codificar las clases que administrará nuestra aplicación. Ahora, comenzaremos a codificar las clases que contendrán los métodos que representarán las reglas de negocio.

Habilidades y conocimientos aplicados en esta fase

● Codificar métodos.

● Comprender el concepto de colección.

● Codificar funciones que devuelvan colecciones.

Paso 1

Agregar un nuevo proyecto del tipo librería de clases a la solución y llamarlo Negocio.

Paso 2

Establecer una referencia al proyecto Entidades.

Paso 3

Codificar la clase admClientes con los métodos que se detallan a continuación:

|  |  |  |
| --- | --- | --- |
| Clase: admClientes | | |
| Método | Parámetros | Salida |
| ListarClienteIndividuo() |  | lista de ClienteIndividuo |
| ListarClienteIndividuo() | CUIT (string) | ClienteIndividuo |
| ListarClienteEmpresa() |  | lista de Empresa |
| ListarClienteEmpresa() | CUIT (string) | Empresa |

● Debemos importar el espacio de nombres Entidades.

● Es probable que debamos agregar constructores por defecto a las clases ClienteIndividuo y Empresa.

Paso 4

Codificar la clase admProductos con los métodos que se detallan a continuación:

|  |  |  |
| --- | --- | --- |
| Clase: admProductos | | |
| Método | Parámetros | Salida |
| Listar() |  | lista de producto |
| Listar() | Categoria (string) | lista de Producto |
| Listar() | Categoria (string), Subcategoria (string) | lista de Producto |
| ListarDetalle() | Nombre (string) | Producto |

● Deberemos importar el espacio de nombres Entidades.

● Es probable que debamos agregar constructores por defecto a la clase Producto.

Paso 5

Codificar la clase admCompras con los métodos que se detallan a continuación:

|  |  |  |
| --- | --- | --- |
| Clase: admCompras | | |
| Método | Parámetros | Salida |
| Agregar() | item(producto),cantidad(entero) |  |
| Modificar() | cantidad(entero) |  |
| Eliminar() | nombre(string) |  |
| Confirmar() | Vendedor() |  |
| Cancelar() |  |  |

● Deberemos importar el espacio de nombres Entidades.

Paso 6

Codificar la clase admFacturas con los métodos que se detallan a continuación:

|  |  |  |
| --- | --- | --- |
| Clase: admFacturas | | |
| Método | Parámetros | Salida |
| Generar() | Productos(lista de producto),Cliente(ClienteIndividuo) |  |
| Generar() | Productos(lista de producto),Cliente(ClienteEmpresa) |  |

● Deberemos importar el espacio de nombres Entidades.

Fase 04

Introducción

Si analizamos las clases Empresa y Persona (junto con sus derivadas ClienteIndividuo y Vendedor) vemos que todas comparten un conjunto de atributos: email, dirección y teléfono.

Podemos entonces crear una clase, llamada por ejemplo DatosContacto, que reúna dichos atributos en una entidad. Más adelante sería posible en esta clase implementar métodos que permitiesen validar dichos atributos según las reglas que oportunamente se estableciesen.

Por otro lado, la clase Producto tiene dos atributos referidos a su categorización: Categoria y Subcategoria.

Estas propiedades están codificadas como string, es decir, podemos básicamente asignarles cualquier valor.

Sin embargo, en esta aplicación, las categorías y subcategorías estarán normalizadas, es decir, deberán poder asignarse desde un conjunto predefinido. Para ello es conveniente crear una nueva entidad, a la que llamaremos Categoria y codificar ambos atributos en forma consecuente.

Además, en la misma clase, tenemos un atributo Proveedor, también codificado como string. Sin embargo, ya sabemos que el proveedor es una empresa, y por lo tanto, codificaremos dicho atributo como tal.

Finalmente, revisemos la clase Factura. Ésta posee un atributo que representa el detalle de los productos.

Actualmente se encuentra codificado como string, pero evidentemente no es una buena idea codificar una lista de ítems en un string.

Para resolver este problema crearemos otra clase a la que llamaremos Item y se procederá a codificar el atributo Detalle como una colección de objetos Item.

Habilidades y conocimientos aplicados en esta fase

● Comprender las relaciones entre objetos.

● Codificar relaciones entre objetos.

Paso 1

Crear en Entidades una clase llamada DatosContacto con los atributos Email, Direccion y Telefono, todos del tipo string y codificados como propiedades autoimplementadas. Implementar su correspondiente constructor parametrizado.

Paso 2

Relacionar esta clase con las clases Empresa y Persona (y sus derivadas).

Paso 3

Realizar las modificaciones necesarias en la consola para corregir los errores que se han producido.

Paso 4

En el proyecto Entidades, crear la clase Categoria con su constructor parametrizado. Dicha clase tendrá por ahora un único atributo llamado Nombre del tipo string.

Paso 5

En la clase Producto, cambiar el tipo de dato de los atributos Categoria y Subcategoria a Categoría.

Paso 6

Realizar las modificaciones necesarias en la consola para corregir los errores que se han producido.

Paso 7

En la clase Producto, cambiar el tipo de dato del atributo Proveedor a Empresa.

Paso 8

Realizar las modificaciones necesarias en la consola para corregir los errores que se han producido.

Paso 9

En el proyecto Entidades, crear una clase denominada Item y codificar los siguientes atributos:

|  |  |  |
| --- | --- | --- |
| Clase: Item | | |
| Propiedad | Tipo de dato | Acceso |
| Cantidad | Entero | lectura/escritura |
| Descripción | String | lectura/escritura |
| Preciounitario | Decimal | lectura/escritura |
| Importe | Decimal | Lectura |

Codificar un constructor parametrizado.

Paso 10

En la clase Factura, codificar el atributo Detalle como lista de Item.

Paso 11

Realizar las modificaciones necesarias en la consola para corregir los errores que se han producido.

Paso 12

Para finalizar, podemos modificar el atributo Total de la clase Factura codificándolo como atributo de sólo lectura, ya que será la suma de los importes de cada item.

Paso 13

Realizar las modificaciones necesarias en la consola para corregir los errores que se han producido.

Paso 14

En el proyecto Negocio, adaptar la clase admProductos reemplazando los parámetros referidos a categorias y subcategorias definidos como string, por parámetros del tipo Categoria. Renombrar el método ListarDetalle() como Listar(), ya que ahora es posible sobrecargar el mismo.

Paso 15

Finalmente modificar los métodos en la clase admCompras() para que utilicen la clase Item como parámetros.

Fase 05

Introducción

Teniendo en cuenta el concepto de clase estática, podemos ver que todas las clases que hemos creado en la librería de clases Negocio pueden ser clases estáticas, ya que nunca necesitaremos más de una instancia de ellas para poder utilizarlas.

Por otro lado, si se asume que todos los productos compartirán el mismo valor de I.V.A., podemos declarar dicha propiedad como estática (o sea compartida por todas las instancias de la clase Producto).

Finalmente, aprovecharemos esta fase para agregar los métodos necesarios para gestionar las altas, bajas y modificaciones de clientes y productos.

Habilidades y conocimientos aplicados en esta fase:

● Comprender el concepto de clase estática.

● Codificar clases estáticas.

● Comprender el concepto de atributo estático.

● Codificar atributos estáticos.

Paso 1

En el proyecto Negocio, transformar todas las clases (y sus métodos) en clases estáticas.

Paso 2

En la clase Producto, declarar el atributo IVA estático.

Paso 3

Realizar las modificaciones necesarias en la consola para corregir los errores que se han producido.

Paso 5

Implementar las siguientes modificaciones:

|  |  |  |
| --- | --- | --- |
| Clase: admClientes | | |
| Método | Parámetros | Salida |
| Agregar() | Cliente(clienteIndividuo) |  |
| Modificar() | Cliente(clienteIndividuo) |  |
| Eliminar() | Cliente(clienteIndividuo) |  |
| Agregar() | Cliente(empresa) |  |
| Modificar() | Cliente(empresa) |  |
| Eliminar() | Cliente(empresa) |  |

|  |  |  |
| --- | --- | --- |
| Clase: admProductos | | |
| Método | Parámetros | Salida |
| Agregar() | Producto(Producto) |  |
| Modificar() | Producto(Producto) |  |
| Eliminar() | Producto(Producto) |  |

Fase 06

Introducción

Teniendo en cuenta el proceso de negocio de nuestra aplicación comenzaremos una primera aproximación a la interfaz gráfica, que será del tipo Windows Forms.

En las fases subsiguientes se irá ajustando en función de las necesidades de cada proceso.

Habilidades y conocimientos aplicados en esta fase:

● Diseñar una interfaz de usuario para un cliente Windows Forms.

Paso 1

Agregar un nuevo proyecto a la solución del tipo Windows Forms y llamarlo Presentacion.

Paso 2

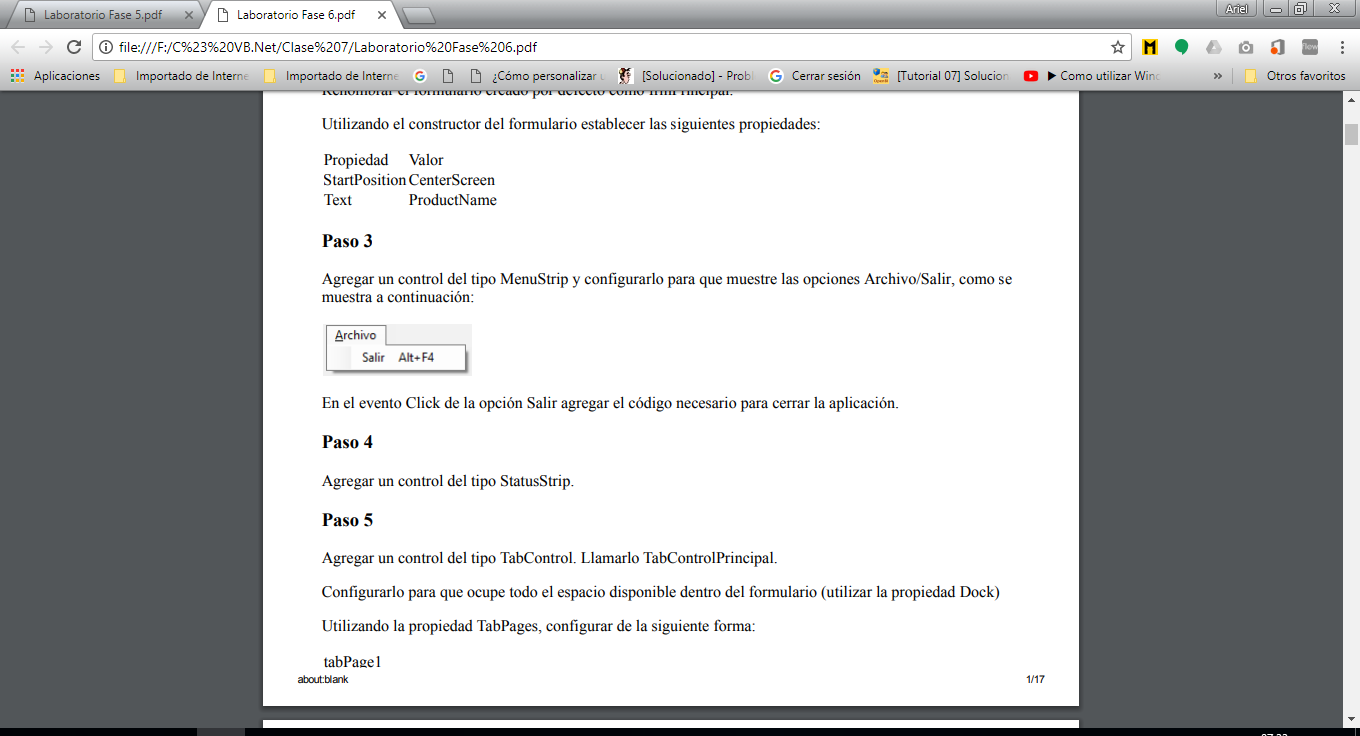
Renombrar el formulario creado por defecto como frmPrincipal.

Utilizando el constructor del formulario establecer las siguientes propiedades:

|  |  |
| --- | --- |
| Propiedad | Valor |
| StartPosition | CenterScreen |
| Text | ProductName |

Paso 3

Agregar un control del tipo MenuStrip y configurarlo para que muestre las opciones Archivo/Salir, como se muestra a continuación:



En el evento Click de la opción Salir agregar el código necesario para cerrar la aplicación.

Paso 4

Agregar un control del tipo StatusStrip.

Paso 5

Agregar un control del tipo TabControl. Llamarlo TabControlPrincipal.

Configurarlo para que ocupe todo el espacio disponible dentro del formulario (utilizar la propiedad Dock)

Utilizando la propiedad TabPages, configurar de la siguiente forma:

|  |  |
| --- | --- |
| tabPage1 |  |
| Name | tabPageVentas |
| Text | Ventas |
| tabPage2 |  |
| Name | tabPageProductos |
| Text | Catálogo de productos |
| tabPage3 |  |
| Name | tabPageClientes |
| Text | Clientes |
| tabPage4 |  |
| Name | tabPageProveedores |
| Text | Proveedores |
| tabPage5 |  |
| Name | tabPageFacturas |
| Text | Facturas |

Paso 6

Dentro de la pestaña tabPageVentas agregar un control del tipo toolStrip. Llamarlo toolStripVentas.

Agregar 3 toolStripButton(s) configurados como se indica a continuación:

|  |  |
| --- | --- |
| toolStripButton1 |  |
| Name | toolStripButtonAbrir |
| Text | Abrir |
| ToolTipText | Comienza una nueva transacción de venta. |
| DisplayStyle | Text |
| toolStripButton2 |  |
| Name | toolStripButtonCancelar |
| Text | Cancelar |
| ToolTipText | Cancela un transacción de venta en curso. |
| DisplayStyle | Text |
| Enabled | False |
| toolStripButton3 |  |
| Name | toolStripButtonCerrar |
| Text | Cerrar |
| ToolTipText | Confirma la transacción de venta. |
| DisplayStyle | Text |
| Enabled | False |

Paso 7

Dentro de la pestaña tabPageVentas agregar un control del tipo tabControl. Llamarlo tabControlVenta.

Utilizando la propiedad TabPages, configurar de la siguiente forma:

|  |  |
| --- | --- |
| tabPage1 |  |
| Name | tabPageSeleccionarCliente |
| Text | Ventas |
| tabPage2 |  |
| Name | tabPageConfigurarPedido |
| Text | Catálogo de productos |

Paso 8

Dentro de pestaña tabPageSeleccionarCliente agregar un control del tipo splitContainer y llamarlo

splitContainerSeleccionarCliente.

Paso 9

Dentro del panel splitContainer1.Panel1 del splitContainerSeleccionarCliente agregar un control del tipo tabControl y llamarlo tabControlSeleccionarCliente.

Configurarlo para que ocupe todo el espacio disponible dentro del formulario (utilizar la propiedad Dock)

Utilizando la propiedad tabPages, configurar de la siguiente forma:

|  |  |
| --- | --- |
| tabPage1 |  |
| Name | tabPageSeleccionarClienteIndividuo |
| Text | Particular |
| tabPage2 |  |
| Name | tabPageSeleccionarClienteEmpresa |
| Text | Empresa |

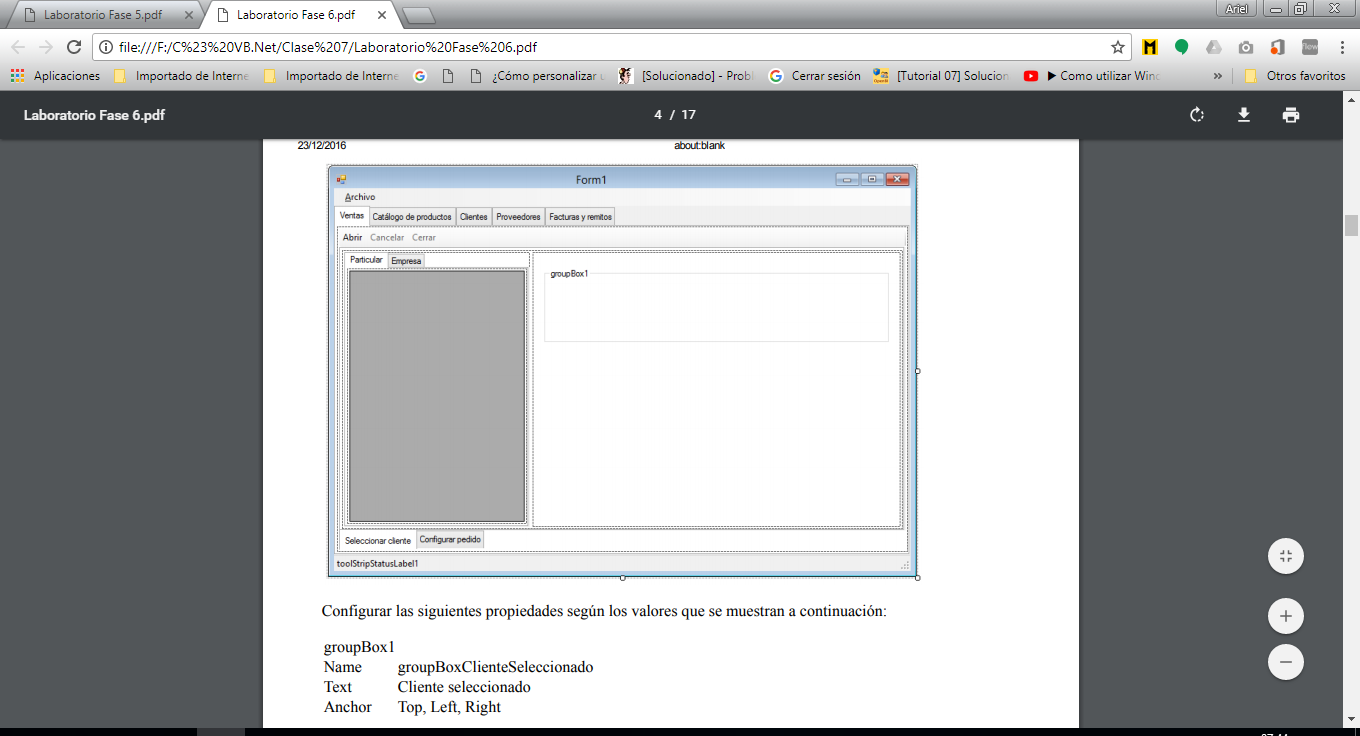
Asignar el valor “Bottom” a la propiedad Alignment del control tabControlSeleccionarCliente.

Paso 10

Dentro del tabPageSeleccionarClienteIndividuo y del tabPageSeleccionarClienteEmpresa, agregar 2 controles del tipo dataGridView. Llamarlos dataGridViewSeleccionarClienteIndividuo y dataGridViewSeleccionarClienteEmpresa respectivamente.

Paso 11

Dentro del panel splitContainer1.Panel2 del splitContainerSeleccionarCliente agregar un control del tipo groupBox en una posición similar a la que se muestra a continuación:



Configurar las siguientes propiedades según los valores que se muestran a continuación:

|  |  |
| --- | --- |
| groupBox1 |  |
| Name | groupBoxClienteSeleccionado |
| Text | Cliente seleccionado |
| Anchor | Top, Left, Right |

Paso 12

Dentro del control groupBoxClienteSelecionado agregar 2 controles del tipo label, uno debajo del otro, y llamarlos respectivamente labelNombreClienteSeleccionado y labelCUITClienteSeleccionado.

Asignar “Nombre” y “C.U.I.T.” a las propiedades Text a cada control según corresponda.

Paso 13

Debajo del control GroupBoxClienteSelecionado agregar otro control del tipo groupBox y configurarlo según las propiedades que se muestran a continuación

|  |  |
| --- | --- |
| groupBox2 |  |
| Name | groupBoxDatosContactoClienteSeleccionado |
| Text | Información de contacto |
| Anchor | Top, Left, Right |

Paso 14

Dentro del control groupBoxDatosContactoClienteSelecionado agregar 3 controles del tipo label, uno debajo del otro, y llamarlos respectivamente labelEmailClienteSeleccionado, labelTelefonoClienteSeleccionado y labelDireccionClienteSeccionado.

Asignar “Email”, “Teléfono” y “Dirección” a la propiedad Text de cada control según corresponda.

Paso 15

Debajo del control groupBoxDatosContactoClienteSelecionado agregar otro control del tipo groupBox y configurarlo según las propiedades que se muestran a continuación

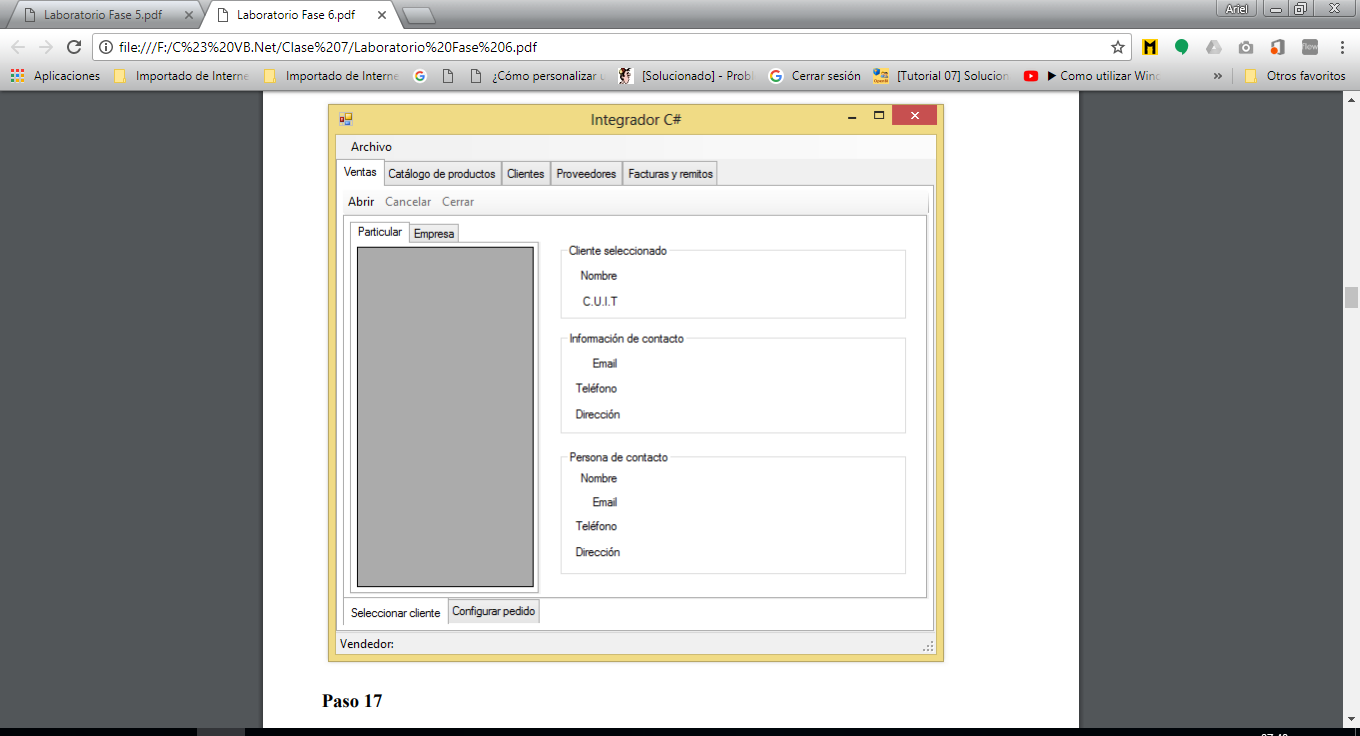
|  |  |
| --- | --- |
| groupBox3 |  |
| Name | groupBoxContactoContactoSeleccionado |
| Text | Persona de contacto |
| Anchor | Top, Left, Right |

Paso 16

Dentro del control groupBoxContactoClienteSelecionado agregar un control del tipo label y llamarlo labeContacto.

Asignar “Nombre” a la propiedad Text del control.

En este momento, nuestro formulario, en modo ejecución, debería ser similar a la de la siguiente figura:



Paso 17

Dentro de pestaña tabPageConfigurarPedido agregar un control del tipo splitContainer y llamarlo splitContainerConfigurarPedido.

Paso 18

Dentro del panel splitContainer1.Panel1 del splitContainerConfigurarPedido agregar un control del tipo panel y llamarlo panelCategorias.

Asignar el valor “Top” a su propiedad Dock.

Paso 19

Dentro del control panelCategorias agregar 2 controles del tipo comboBox, uno debajo del otro. Configurar las siguientes propiedades como se muestra a continuación:

|  |  |
| --- | --- |
| comboBox1 |  |
| Name | comboBoxCategoria |
| Anchor | Top, Left, Right |
| DropDownStyle | DropDownList |
| comboBox2 |  |
| Name | comboBoxSubcategoria |
| Anchor | Top, Left, Right |
| DropDownStyle | DropDownList |

Paso 20

Debajo del control panelCategorias, agregar un control del tipo dataGridView y nombrarlo dataGridViewPedidoProductos.

Establecer su propiedad Anchor en el valor Top, Bottom, Left, Right.

Paso 21

Dentro del panel splitContainer1.Panel2 del splitContainerConfigurarPedido agregar un control del tipo panel y llamarlo panelCondiciones.

Asignar el valor “Top” a su propiedad Dock.

Paso 22

Dentro del control panelCondiciones agregar dos controles del tipo label, uno debajo del otro. Nombrarlos labelCondicionIVA y labelCondicionVenta respectivamente.

Asignar los valores “Condición frente al I.V.A.” y “Condición de venta” a la propiedad Text según corresponda.

Paso 23

Al lado de cada una de los controles label agregados en el paso anterior, agregar 2 controles del tipo comboBox y configurarlos con los siguientes valores:

|  |  |
| --- | --- |
| comboBox1 |  |
| Name | comboBoxCondicionIVA |
| Anchor | Top, Left, Right |
| DropDownStyle | DropDownList |
| comboBox2 |  |
| Name | comboBoxCondicionVenta |
| Anchor | Top, Left, Right |
| DropDownStyle | DropDownList |

Establecer su propiedad Anchor en el valor Top, Bottom, Left, Right.

Paso 24

Dentro del panel splitContainer1.Panel2 del splitContainerConfigurarPedido agregar un control del tipo panel y llamarlo panelTotal.

Asignar el valor “Bottom” a su propiedad Dock.

Paso 25

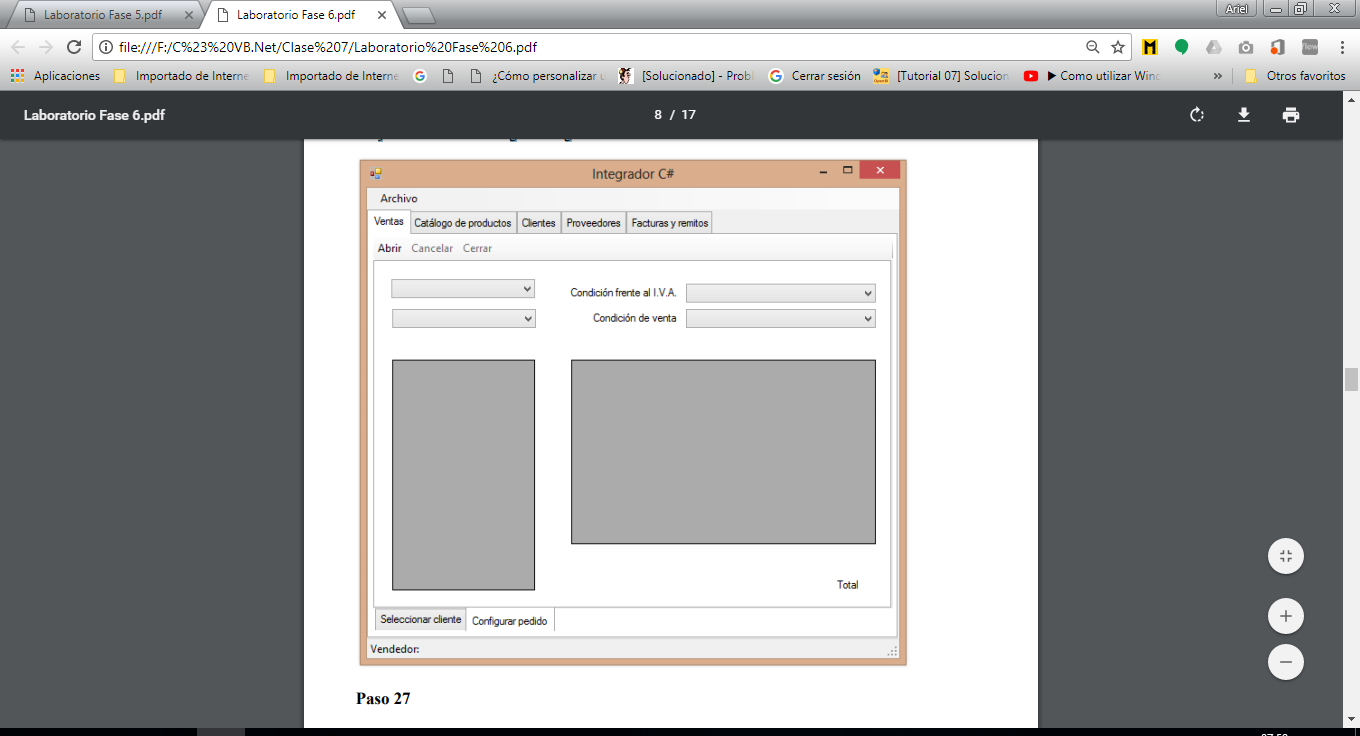
Dentro del control panelTotal agregar un control del tipo label y nombrarlo labelTotal. Asignar el valor “Total” a su propiedad Text y arrastralo hacia el lado derecho del panel.

Configurar el valor “Top, Right” a su propiedad Dock.

Paso 26

En la zona que ha quedado entre ambos paneles, agregar un control dataGridView. Nombrarlo dataGridViewPedido y configurar su propiedad Anchor con el valor “Top, Bottom, Left, Right”.

En estos momentos, el formulario con la pestaña “Configurar pedido” seleccionada debería verse en tiempo de ejecución similar a la siguiente figura:



Paso 27

En la pestaña tabPageProductos, agregar un control del tipo toolStrip. Llamarlo toolStripVentas.

Agregar 3 toolStripButton(s) configurados como se indica a continuación:

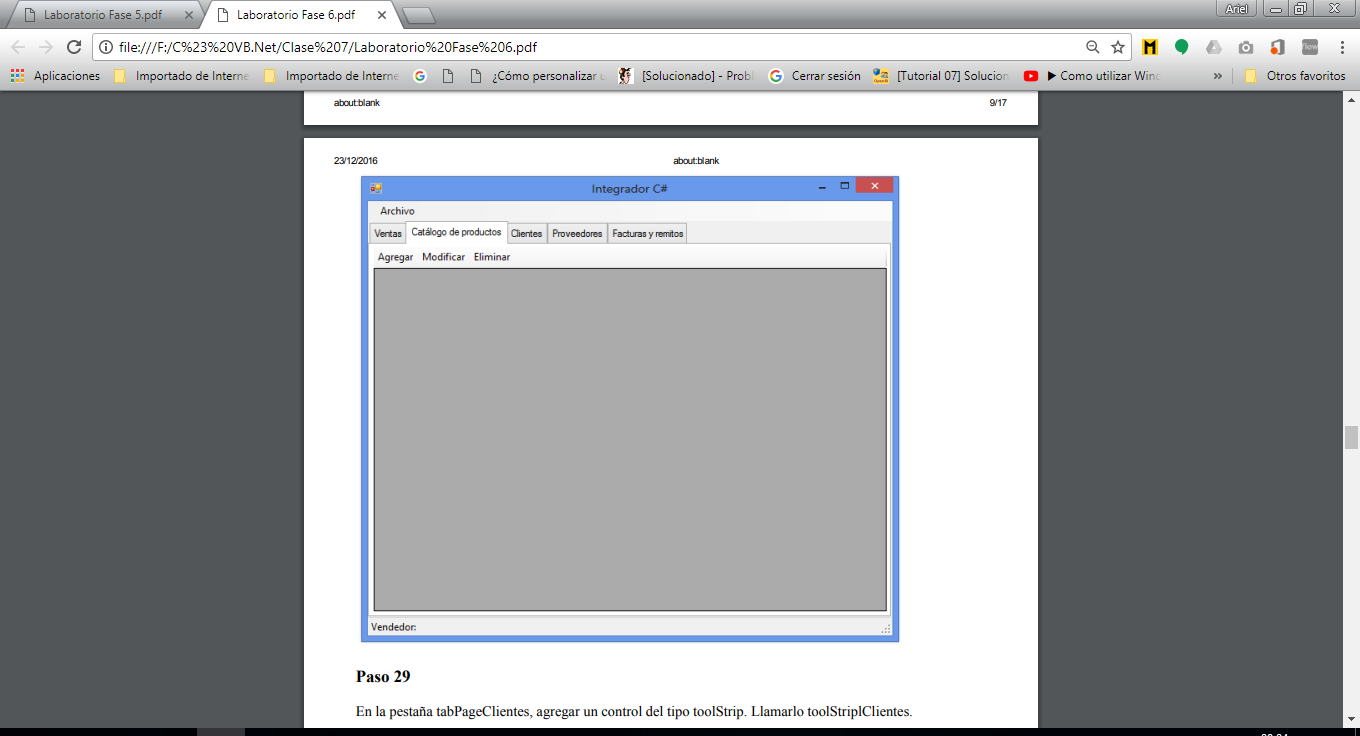
|  |  |
| --- | --- |
| toolStripButton1 |  |
| Name | toolStripButtonAgregarProducto |
| Text | Agregar |
| ToolTipText | Agrega un nuevo producto. |
| DisplayStyle | Text |
| toolStripButton2 |  |
| Name | toolStripButtonModificarProducto |
| Text | Modificar |
| ToolTipText | Modifica el producto seleccionado. |
| DisplayStyle | Text |
| toolStripButton3 |  |
| Name | toolStripButtonEliminarProducto |
| Text | Eliminar |
| ToolTipText | Elimina el producto seleccionado. |
| DisplayStyle | Text |

Paso 28

En la pestaña tabPageProductos, agregar un control del tipo dataGridView. Llamarlo dataGridViewProductos.

Configurar su propiedad Dock en el valor Fill.

En estos momentos, el formulario con la pestaña “Catálogo de productos” seleccionada debería verse en tiempo de ejecución similar a la siguiente figura:



Paso 29

En la pestaña tabPageClientes, agregar un control del tipo toolStrip. Llamarlo toolStriplClientes.

Agregar 3 toolStripButton(s) configurados como se indica a continuación:

|  |  |
| --- | --- |
| toolStripButton1 |  |
| Name | toolStripButtonAgregarCliente |
| Text | Agregar |
| ToolTipText | Agrega un nuevo cliente. |
| DisplayStyle | Text |
| toolStripButton2 |  |
| Name | toolStripButtonModificarCliente |
| Text | Modificar |
| ToolTipText | Modifica el cliente seleccionado. |
| DisplayStyle | Text |
| toolStripButton3 |  |
| Name | toolStripButtonEliminarCliente |
| Text | Eliminar |
| ToolTipText | Elimina el cliente seleccionado. |
| DisplayStyle | Text |

Paso 30

En la pestaña tabPageClientes, agregar un control del tipo tabControl. Llamarlo toolControlClientes.

Configurarlo para que ocupe todo el espacio disponible dentro del formulario (utilizar la propiedad Dock)

Utilizando la propiedad tabPages, configurar de la siguiente forma:

|  |  |
| --- | --- |
| tabPage1 |  |
| Name | tabPageClienteIndividuo |
| Text | Particulares |
| tabPage2 |  |
| Name | tabPageClienteEmpresa |
| Text | Empresas |

Paso 31

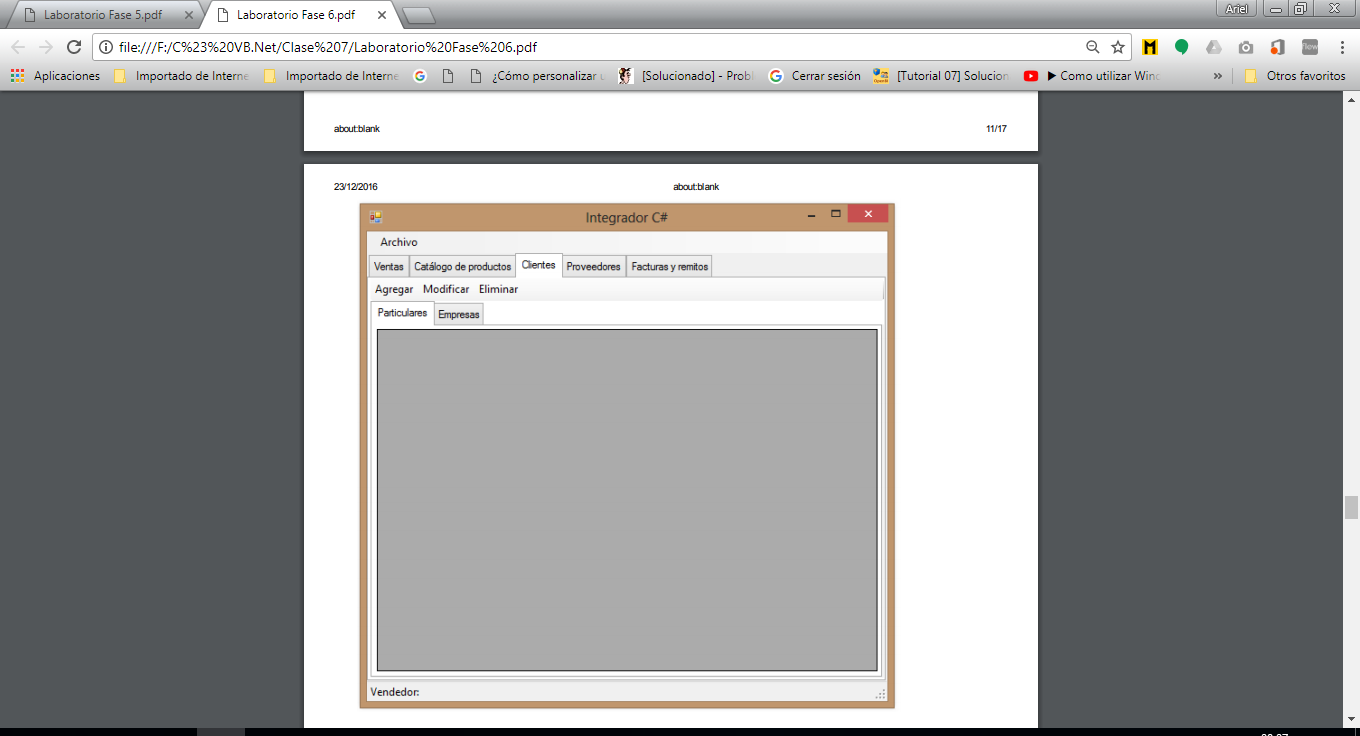
Dentro de la pestaña tabControlClientesIndividuo agregar un control dataGridView y nombrarlo dataGridViewClientesIndividuo.

Configurar su propiedad Dock con el valor Fill.

Dentro de la pestaña tabControlClientesEmpresa agregar un control dataGridView y nombrarlo dataGridViewClientesEmpresa.

Configurar su propiedad Dock con el valor Fill.

En estos momentos, el formulario con la pestaña “Clientes” seleccionada debería verse en tiempo de ejecución similar a la siguiente figura:



Paso 32

En la pestaña tabPageProveedores, agregar un control del tipo toolStrip. Llamarlo toolStriplProveedores.

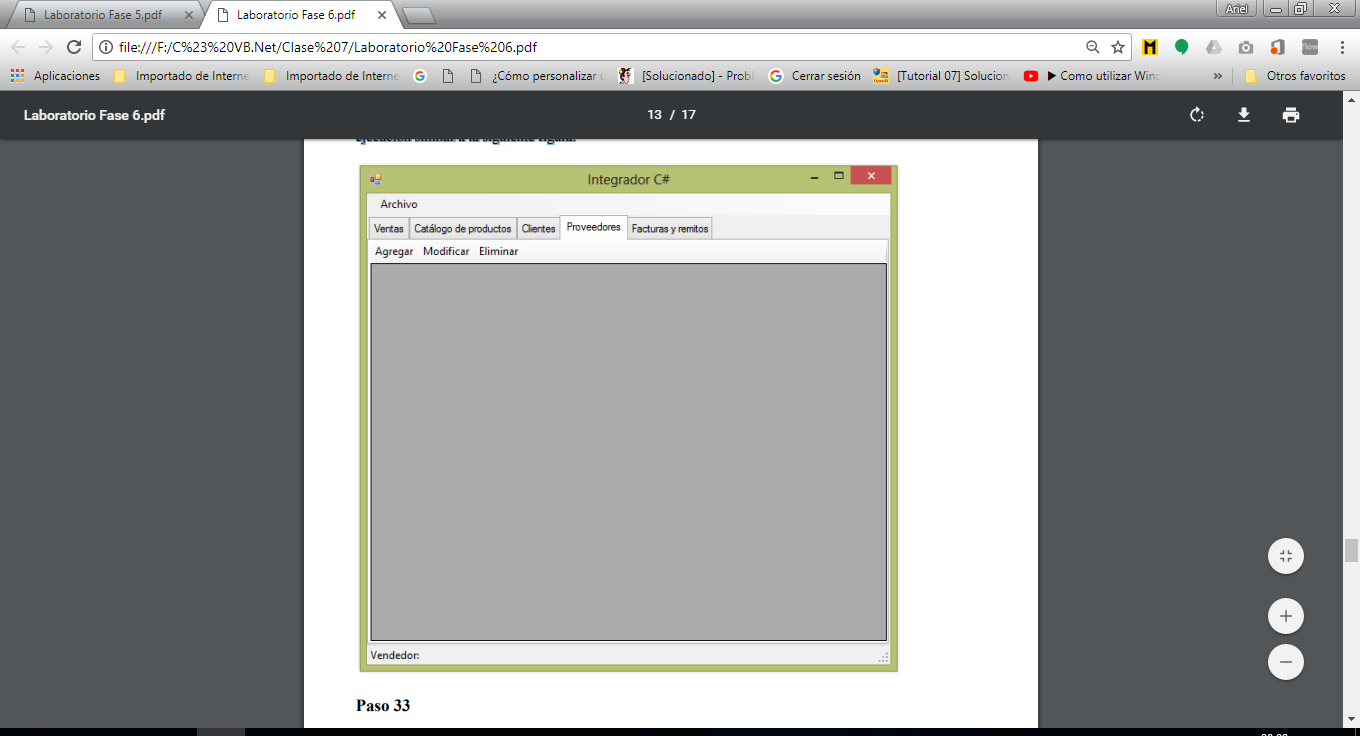
Agregar 3 toolStripButton(s) configurados como se indica a continuación:

|  |  |
| --- | --- |
| toolStripButton1 |  |
| Name | toolStripButtonAgregarProveedor |
| Text | Agregar |
| ToolTipText | Agrega un nuevo proveedor |
| DisplayStyle | Text |
| toolStripButton2 |  |
| Name | toolStripButtonModificarProveedor |
| Text | Modificar |
| ToolTipText | Modifica el proveedor seleccionado. |
| DisplayStyle | Text |
| toolStripButton3 |  |
| Name | toolStripButtonEliminarProveedor |
| Text | Eliminar |
| ToolTipText | Elimina el proveedor seleccionado. |
| DisplayStyle | Text |

Dentro de la pestaña tabControlProveedores agregar un control dataGridView y nombrarlo dataGridViewClientesProveedores.

Configurar su propiedad Dock con el valor Fill.

En estos momentos, el formulario con la pestaña “Clientes” seleccionada debería verse en tiempo de ejecución similar a la siguiente figura:



Paso 33

En la pestaña tabPageFacturas, agregar un control del tipo toolStripDropDownButton. Llamarlo toolStripDropDownButtonOrdenar.

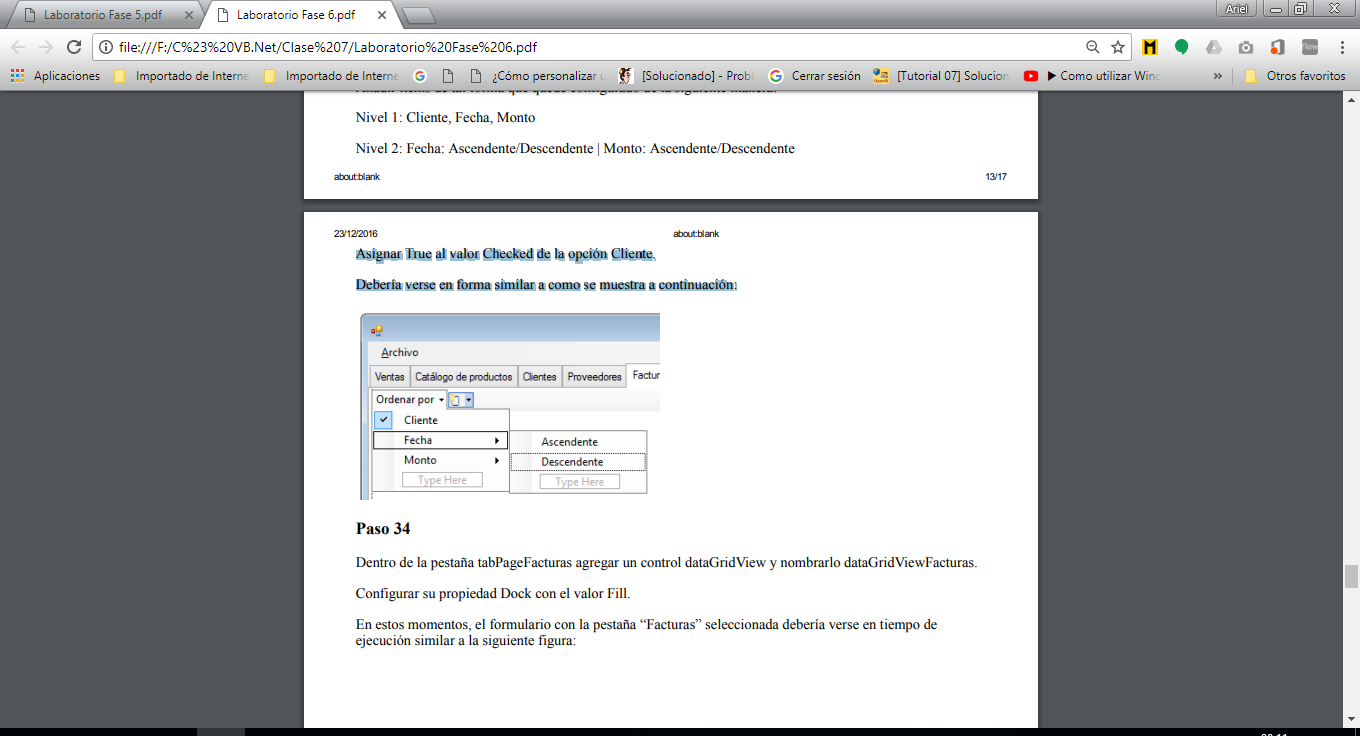
Añadir items de tal forma que quede configurado de la siguiente manera:

Nivel 1: Cliente, Fecha, Monto

Nivel 2: Fecha: Ascendente/Descendente | Monto: Ascendente/Descendente

Asignar True al valor Checked de la opción Cliente.

Debería verse en forma similar a como se muestra a continuación:

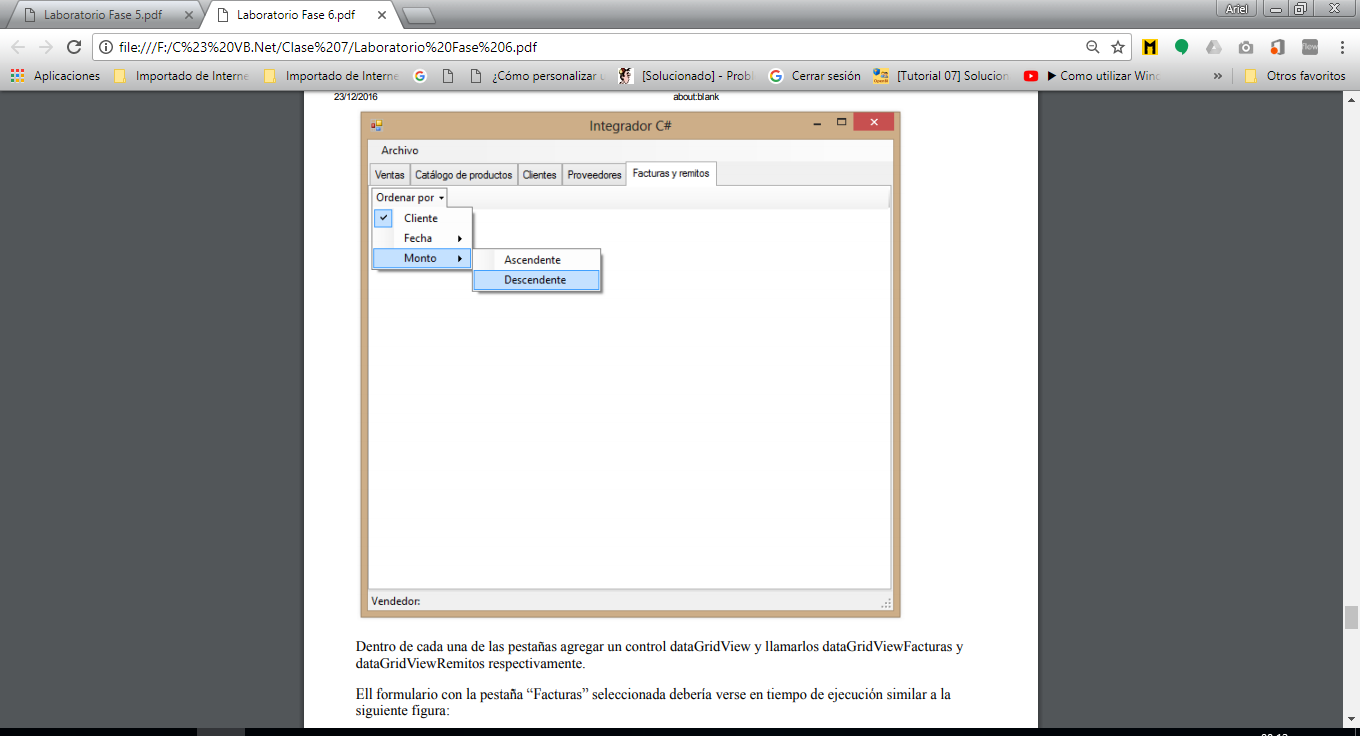


Paso 34

Dentro de la pestaña tabPageFacturas agregar un control dataGridView y nombrarlo dataGridViewFacturas.

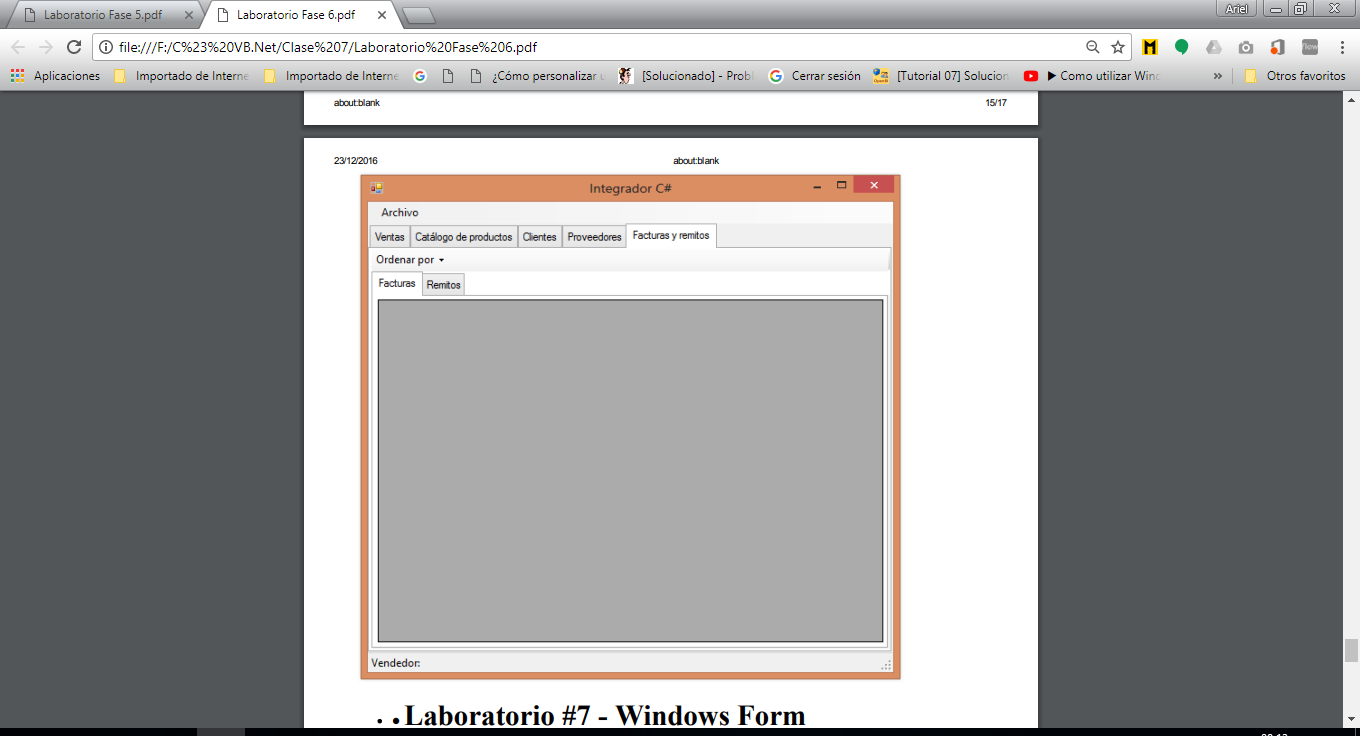
Configurar su propiedad Dock con el valor Fill.

En estos momentos, el formulario con la pestaña “Facturas” seleccionada debería verse en tiempo de ejecución similar a la siguiente figura:



Dentro de cada una de las pestañas agregar un control dataGridView y llamarlos dataGridViewFacturas y dataGridViewRemitos respectivamente.

El formulario con la pestaña “Facturas” seleccionada debería verse en tiempo de ejecución similar a la siguiente figura:



Fase 07

Introducción

Es esta fase procederemos a implementar validaciones en las propiedades de nuestras entidades.

Habilidades y conocimientos aplicados en esta fase

Implementar validaciones en valores de propiedades generando excepciones.

Paso 1

Implementar las validaciones que se indican a continuación:

|  |  |  |  |
| --- | --- | --- | --- |
| Clase: Persona |  |  |  |
| Propiedad | Obligatorio | Valor mínimo | Valor máximo |
| Nombre | Si | 1 | 50 |
| Apellido | Si | 1 | 50 |
| Clase: DatosContacto |  |  |  |
| Email | Si | 6 | 50 |
| Dirección | Si | 15 | 100 |
| Teléfono | No | 15 | 15 |
| Clase: ClienteIndividuo |  |  |  |
| Cuit | si | 11 | 11 |
| Clase: Empresa |  |  |  |
| Nombre | Si | 1 | 50 |
| Cuit | Si | 11 | 11 |
| Clase: Vendedor |  |  |  |
| Dni | no | 8 | 8 |
| Clase: Producto |  |  |  |
| Nombre | Si | 1 | 50 |
| Descripción | No | - | 500 |
| PrecioCosto | Si | 0.1 | - |
| Clase: Categoria |  |  |  |
| Nombre | si | 1 | 50 |
| Clase: DocumentoComercial |  |  |  |
| Numero | Si | 13 | 13 |
| Cliente | Si | 1 | 100 |
| Dirección | Si | 15 | 100 |
| Clase: Factura |  |  |  |
| Tipo | Si | 1 | 1 |
| Clase: Item |  |  |  |
| Cantidad | Si | 1 | - |
| Descripcion | Si | 1 | 50 |
| PrecioUnitario | Si | 0.1 | - |

Paso 2

Es probable que en la consola se hayan producido errores debido a valores inválidos en una o más propiedades de una o más clases. Corregir los valores erróneos y ejecutar nuevamente.

Paso 3

Controlar con la estructura Try/Catch las instrucciones que puedan generar excepciones. Probar asignando valores inválidos a algunas clases y ver que se muestre correctamente el mensaje de error.

Fase 8

Introducción

Esta fase es la más larga y compleja de todo el integrador. A fin de mejorar la comprensión del desarrollo la dividiremos en cuatros partes:

1. Desarrollo de la capa de datos

2. Comunicación de la capa de datos con la capa de negocio

3. Comunicación de la capa de datos con la capa de presentación

En caso de necesidad, consultar el código fuente disponible a fin de observar algunos detalles particulares en la implementación del código. Por supuesto, el código presentado como ejemplo es una propuesta entre otras posibles. Se podrán implementar modificaciones, propuestas alternativas, etc. Sin embargo, para llegar a eso, el punto de partida es comprender el código propuesto como ejemplo.

Habilidades y conocimientos aplicados en esta fase

1. Comprender la estructura de una base de datos y la relación de sus tablas con las entidades de una aplicación.

2. Codificar métodos utilizando las clases del modelo ADO conectado.

Desarrollo de la capa de datos

Paso 1

Ejecutar el script provisto a fin de crear la base de datos, que se llamará IntegradorNET.

Opcionalmente, se podrá diseñarla desde cero utilizando SQL Server Management Studio y la documentación disponible.

Paso 2

Agregar a la solución una librería de clases y llamarla Servidor.

Paso 3

En dicha librería crear una clase estática llamada BaseDatos.

Paso 4

Ir a las propiedades del proyecto Servidor y crear una entrada en Settings con el nombre StringConexion, asignando el valor adecuado para conectar con la base de datos.

Paso 5

En la clase BaseDatos, codificar una propiedad de sólo lectura que devuelva la string de conexión almacenada en los settings del proyecto. Nombrar esta propiedad StringConexion.

Paso 6

Agregar a la solución una librería de clases y llamarla Datos.

Paso 7

En las clases Categoria, Producto, ClienteIndividuo y Empresa agregar una propiedad autoimplementada del tipo entero llamada Id.

Sobrecargar los constructores existentes de tal forma que uno pida el valor del atributo Id y el otro no.

Estas modificaciones se deben a que los objetos guardarán su estado en tablas de bases de datos utilizando un atributo entero como identificador (clave primaria). Sin embargo, cuando el objeto todavía no fue almacenado, no tiene sentido asignar un valor al identificador, ya que todavía el motor de base de datos no le asignó una clave primaria.

Paso 8

En la clase Producto agregar un constructor sobrecargado que sólo pida el Id y el Nombre.

En la clase ClienteIndividuo agregar un constructor sobrecargado que pida solamente Id, Nombre, Apellido y CUIT.

En la clase Empresa agregar dos constructores sobrecargados: uno que pida solamente Id y Nombre y otro que pida Id, Nombre y CUIT.

Estas modificaciones se deben a que no siempre necesitaremos recuperar de la base de datos toda la información de cada objeto.

Paso 9

En el proyecto Datos crear las siguientes clases estáticas y codificar los métodos que se detallan a continuación. El orden en que se listan es el recomendado para el desarrollo, aunque no es indispensable respetarlo.

|  |  |  |
| --- | --- | --- |
| Clase / Método | Parámetros | Salida |
| dbProveedores |  |  |
| Listar() | - | Lista de Empresa |
| dbCategorias |  |  |
| Listar() | - | Lista de Categoria |
| ListarSubcategorías() | Id(entero) | Lista de Categoria |
| dbProductos |  |  |
| Listar() | - | Lista de Producto |
| Listar() | Categoria(entero),Subcategoria(entero) | Lista de Producto |
| Listar() | Id(entero) | Producto |
| dbClientes |  |  |
| ListarClientesIndividuo() | - | Lista de Cliente Individuo |
| ListarClientesIndividuo() | Id(entero) | Cliente Individuo |
| ListarClientesEmpresa() | - | Lista de Cliente Empresa |
| ListarClientesEmpresa() | Id(entero) | Empresa |
| dbFacturas |  |  |
| Generar() | - | - |
| dbRemitos |  |  |
| Generar() | - | - |

Tener en cuenta que ya existen en la base de datos procedimientos almacenados para realizar todas las operaciones de lectura, con excepción de la generación de facturas y remitos.

Comunicación de la capa de datos con la capa de negocios

Paso 1

En las clases admProvedores, admCategorias, admProductos, admClientes, admFacturas y admRemitos llamar a los métodos homónimos de las clases de datos.

En este punto es probable que estemos pensando qué sentido tiene la capa de Negocio, si básicamente estamos utilizándola como intermediaria con respecto a la capa de datos.

Uno de los motivos es que, si en el futuro tuviésemos que implementar esta aplicación utilizando otra base de datos (Oracle, MySQL, etc.) bastaría con crear una nueva librería de datos y/o recodificar la existente sin necesidad de escribir el resto del código.

Sin embargo, el motivo principal es poder implementar reglas de negocio que sean independientes de cómo manipulamos los datos.

Por ejemplo, podríamos impedir que se facture fuera de horario laboral. En este caso, el método admFacturas deberá controlar la hora a fin de impedir de ser necesario la facturación.

En este caso, implementaremos una regla de negocio sencilla: siempre que se factura, se debe emitir un remito.

Si mantenemos las clases admFacturas y admRemitos por separado, esta regla queda supedita a su implementación dentro de la capa de presentación. Sin embargo, si unificamos estas clases en otra llamada por ejemplo admDocumentosComerciales, invocaremos directamente los métodos de datos que generan las facturas y los remitos independizándonos de la capa de presentación.

Paso 2

Unificar la clases admFacturas y admRemitos en una llamada admDocumentosComerciales.

Implementar el método Generar() según se muestra a continuación:

|  |  |  |
| --- | --- | --- |
| Método | Parámetros | Salida |
| Generar() | pFactura(Factura),pRemito(Remito) | - |

Paso 3

Si bien podríamos haber creado tablas en la base de datos a tal efecto, se ha preferido implementar la gestión de posiciones frente al I.V.A. y las opciones de venta utilizando enumeraciones.

Para ello crearemos una nueva librería de clases y la llamaremos Enumeraciones.

Dentro de dicha librería crearemos una clase estática y la llamaremos también Enumeraciones.

En dicha clase implementar las enumeraciones que se muestran a continuación:

|  |  |
| --- | --- |
| Enumeración | Valores |
| CondicionesIVA | Consumidor Final |
|  | Exento |
|  | No Responsable |
|  | Responsable Inscripto |
|  | Responsable No Inscripto |
|  | Responsable Monotributo |
|  |  |
| CondicionesVenta | Contado |
|  | Cuenta Corriente |
|  |  |
| TiposFacturas | A |
|  | B |
|  | C |
|  | E |
|  | X |

Finalmente codificar los siguientes métodos:

|  |  |
| --- | --- |
| Método | Resultado |
| ListarCondicionesIVA() | Lista de CondicionesIVA |
| ListarCondicionesVenta() | Lista de CondicionesVenta |
| ListarTiposFacturas() | Lista de TiposFacturas |

Es probable que debamos “investigar” cómo se devuelve una lista de enumeraciones

Comunicación de la capa de datos con la capa de presentación

1. Cuando el formulario principal termina de construirse, todas las áreas de datos ya deberían estar cargadas con sus respectivos valores.

2. Para ello es necesario configurar previamente algunos controles, principalmente los dataGridView.

3. Por otro lado deberemos inicializar algunos valores necesarios para que todo funcione correctamente, por ejemplo, el valor del atributo IVA de la clase Producto.

Paso 1

Codificar un método que permita configurar los atributos de los controles del tipo dataGridView y llamarlo ConfigurarDataGridView.

En el cuerpo del método asignar al control dataGridView que se le pase como parámetro los siguientes valores:

|  |  |
| --- | --- |
| Propiedad | Valor |
| ReadOnly | True |
| MultiSelect | False |
| SelectionMode | FullRowSelect |
| AutoSizeColumnsMode | Fill |
| RowHeadersVisible | False |
| AllowUserToResizeRows | False |
| AlternatingRowsDefaultCellStyle.BackColor | Color a elección |

Los valores de las propiedades ReadOnly, MultiSelect y SelectionMode no tienen finalidad estética sino funcional.

Paso 2

Llamar al método ConfigurarDataGridView desde el constructor pasándole sucesivamente como parámetros los controles:

1. dataGridViewClientesIndividuo

2. dataGridViewClientesEmpresa

3. dataGridViewProductos

4. dataGridViewProveedores

5. dataGridViewSeleccionarClienteEmpresa

6. dataGridViewSeleccionarClienteIndividuo

7. dataGridViewPedido

8. dataGridViewPedidoProductos

Paso 3

En las propiedades del proyecto Presentación, crear un setting del tipo doble llamado IVA y asignarle el valor .21.

Paso 4

En el constructor, asignar el valor del setting recién creado al atributo estático IVA de la clase Producto.

Paso 5

Crear los siguientes procedimientos:

1. ListarProveedores()

2. ListarProductos()

3. ListarClientesIndividuo()

4. ListarClientesEmpresa()

5. ListarCategorias()

En el cuerpo de cada procedimiento llamar a la función de la capa de datos correspondiente:

1. admProveedores.Listar()

2. admProductos.Listar()

3. admClientes.ListarClienteIndividuo()

4. admCliente.ListarClienteEmpresa()

5. admCategorias.Listar(),

y asignarlas a la propiedad DataSource de sus respectivos DataGridView, excepto ListarCategorias(), que se asignará al mismo atributo pero del control comboBoxCategoria.

Además, en los controles del tipo comboBox es necesario definir, además del DataSource, el DisplayMember y ValueMember. Estos atributos representan cuál atributo se muestra y cuál se devuelve cómo valor.

El orden correcto de asignación es ValueMember, DisplayMember y Datasource. Los valores serán en este caso “Id”, “Nombre” y admCategorias.Listar() respectivamente.

Paso 6

En el constructor del formulario, llamar a los métodos definidos en el paso 4.

Paso 7

Por último, debajo de las llamadas a los procedimientos que acabamos de codificar, asignar las propiedades DataSource a los controles ComboBoxCondicionIVA y comboBoxCondicionVenta, utilizando las funciones ListarCondicionesIVA() y ListarCondicionesVenta() definidas con anterioridad.

Paso 8

Si ejecutamos en este momento el programa y observamos el resultado obtenido en cada DataGridView nos damos cuenta que aquellos objetos, que como valor de sus atributos tienen otros atributos, no se muestran correctamente. Concretamente esto sucede en la clase Producto.

Esto se puede solucionar creando atributos de sólo lectura. Codificar entonces en la clase Producto los siguientes atributos:

|  |  |
| --- | --- |
| Atributo | Salida |
| ViewProveedor | Nombre del Proveedor |
| ViewCategoria | Nombre de la Categoria |
| ViewSubCategoria | Nombre de la Subcategoria |

Paso 6

En las clases que devuelven objetos del tipo ClienteIndividuo, vemos que el nombre y el apellido se muestran por separado.

Crear entonces un atributo de sólo lectura en la clase Persona llamado NombreCompleto que devuelva el apellido separado con una coma del nombre. por ejemplo: “Pérez, Juan”.

Paso 7

Finalmente, en cada uno de los procedimientos creados en el paso 4, ajustaremos las columnas de cada control dataGridView a fin de lograr que:

1. Las columnas que muestran el identificador de cada objeto no se muestren.

2. Asimismo, ocultaremos todas las columnas que hayan sido reemplazadas por las propiedades de sólo lectura (nombre, apellido, proveedor, categoria, subcategoria).

3. Los textos de los encabezados que no sean correctos deben modificarse. Por ejemplo, “ViewProveedor” debe reemplazarse por “Proveedor”.

4. Los encabezados de las columnas que muestren números y sus celdas deberán estar alineadas hacia la derecha.

5. Las celdas que muestran valores monetarios o porcentajes deberán formatearse a tal efecto.

6. Finalmente, si una columna queda en el extremo derecho de la grilla, pero esa información es importante, deberá desplazarse hacia la izquierda.

Para implementar este paso, se recomienda consultar el código de ejemplo, más que nada por cuestiones de tiempo.

Paso 8

En este paso codificaremos lo necesario para el control comboBoxSubcategoria muestre las subcategorías correspondientes a la categoria seleccionada en el control comboBoxCategoria.

Para ello utilizaremos asociaremos un procedimiento al evento SelectedIndexChanged del comboBoxCategoria.

En dicho evento estableceremos los valores ValueMember, DisplayMember y DataSource del control comboBoxSubcategoria según los siguientes valores: “Id”, “Nombre” y admCategorias.ListarSubCategorias().

El valor de entrada del método admCategorias.ListarSubCategorias() se obtiene preguntando por el valor del atributo SelectedValue del control comboBoxCategoria.

Paso 9

A fin de lograr que el control DataGridViewPedidoProductos muestre los productos filtrados por categoria y subcategoría, debemos asignar el datasource y sus columnas en el evento SelectedIndexChange del control comboBoxSubcategoria.

Paso 10

En este paso comenzaremos a codificar la selección del producto. Para ello utilizaremos asociaremos un procedimiento al evento SelectedIndexChanged del comboBoxSubcategoria.

En dicho procedimiento asignaremos el resultado del método admProductos.Listar() pasándole como parámetros los valores de categoría y subcategoría a través del atributo SelectedValue, obtenidos de los respectivos controles comboBoxes.

Finalmente, procederemos a ajustar las columnas con un criterio similar al definido en el paso 7.

Paso 11

Ahora implementaremos lo necesario para el proceso de selección de clientes, tanto individuos como empresas.

Comenzaremos agregando en los procedimientos ya definidos ListarClientesIndividuo() y ListarClientesEmpresa() lo necesario para que los controles DataGridViewSeleccionarClienteIndividuo y dataGridViewSeleccionarClienteEmpresa sus respectivas propiedades DataSource. La mecánica es exactamente igual a la efectuada con anterioridad, incluyendo el ajuste de las columnas.

A continuación, asociaremos un procedimiento al evento DoubleClick de ambos dataGridView.

En estos procedimientos, procederemos a instanciar un objeto del tipo ClienteIndividuo o Empresasegún corresponda, invocando respectivamente a los métodosadmClientes.ListarClienteIndividuo() y admClientes.ListarClienteEmpresa() pasándoles como parámetro el Id del cliente seleccionado en sus respectivos controles dataGridView.

Una vez instanciado el objeto, simplemente se debe asignar a las propiedades de texto de las labels que se encuentran en los controles groupBoxClienteSeleccionado, groupBoxDatosContactoClienteSeleccionado y groupBoxContactoClienteSeleccionado sus respectivos valores.

Finalmente, como el control que muestra los datos de la persona de contacto sólo tiene sentido cuando se selecciona la pestaña “Empresa”, asociar un procedimiento al evento SelectedIndexChanged del control tabControlSeleccionarCliente y utlizando la propiedad Visible decidir cuándo debe mostrarse y cuándo no.

Aunque hay diversas maneras de implementar la funcionalidad comentada, es recomendable consultar el código de ejemplo y ver el uso que se le ha dado a la variable ClienteSeleccionado definida a nivel de clase.

Paso 12

En este paso comenzaremos a implementar la lógica asociada al “carrito de compras”.

Comenzaremos implementando la funcionalidad de soltar y arrastrar (drag & drop) entre los controles dataGridViewPedidoProductos y dataGridViewPedido.

Para ello primero debemos establecer qué hacer cuando el usuario pulsa el botón izquierdo del mouse sobre el control dataGridViewPedidoProductos.

A fin de obtener el valor del item seleccionado y al mismo tiempo tener un feedback visual, asociaremos un procedimiento al evento MouseDown de dicho dataGridView.

Para saber cómo extraer el valor de la celda “Id”, que es lo que utilizaremos como identificador, y generar el feedback visual consultar el código de ejemplo.

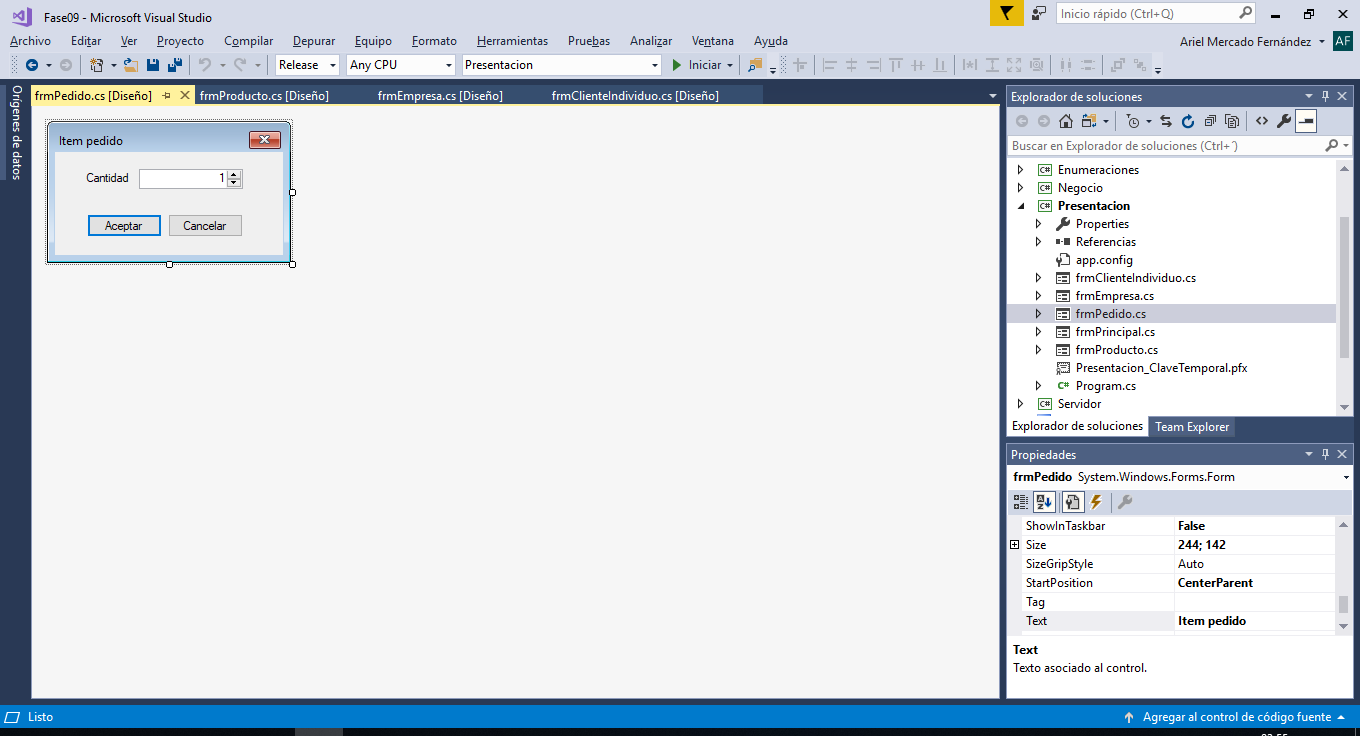
A continuación, debemos configurar al control dataGridViewPedido para que acepte el elemento que arrastraremos a él. Para ello asignar el valor verdadero al atributo AllowDrop.

Finalmente, deberemos asociar procedimientos a los eventos DragEnter y DragDrop.

El evento DragEnter se utiliza verificar que tipo de dato se está arrastrando sobre el control y para proveer feedback visual al usuario. Consultar el código de ejemplo para ver cómo se realizan estas operaciones.

Antes de continuar con el procedimiento asociado el evento DragDrop deberemos agregar a nuestro proyecto un pequeño formulario que nos servirá para que el usuario pueda seleccionar la cantidad de unidades a agregar al pedido y eventualmente modificarla.

Dicho formulario se llamará frmPedido y constará de una label, un dropDownList y dos controles del tipo Button. El diseño debería verse similar al siguiente:



Utilizar los siguientes controles:

labelCantidad

numericUpDownCantidad

buttonAceptar

buttonCancelar

Una vez creado el formulario, se procederá a llamarlo en forma modal desde el procedimiento asociado al evento DragDrop pero antes deberemos definir la lógica del carrito de compras, lo que haremos a continuación.

Paso 13

En este punto retomaremos la clase admCompras codificada con anterioridad y comenzaremos a definir sus operaciones.

Para ello será necesario declarar una variable del tipo lista de Item, que será como puede suponerse la lista de items a facturar.

Luego declararemos dos propiedades de sólo lectura: una que devuelve la lista de items y otra que devuelve el total de la compra: Items y Total respectivamente.

El método Agregar() pide como parámetro un Item. Simplemente lo agrega a la lista.

El método Modificar() pide como parámetro el identificador del item a modificar y la cantidad, que es el único atributo modificable.

El método Eliminar() solamente el identificador del item a eliminar. Pero, ¿a qué nos referimos con el identificador del Item?

Si miramos el código de ejemplo, vemos que en la clase Item se ha agregado un atributo de sólo lectura llamado Id definido del tipo Guid. Guid es un número único que es generado por el método NewGuid() de dicha clase. Será la forma de identificar unívocamente nuestro item en la lista, ya que no disponemos de un identificador generado por un motor de base de datos.

La implementación del método Modificar() y Eliminar() utiliza expresiones Lambda que se encuentran fuera del alcance de este curso, pero de fácil interpretación.

Paso 14

Ya estamos en condiciones de completar el formulario frmPedido.

Primero definiremos una enumeración a la que llamaremos Modos con los valores Agregar y Modificar.

Dado que el formulario será utilizado en ambas modalidades, utilizaremos esta enumeración como bandera para saber qué hacer cuando el usuario presione el botón “Aceptar”.

A continuación, definiremos un constructor que pedirá 2 parámetros: el Id del producto que fue arrojado al control dataGridViewPedido y la modalidad. En este caso la modalidad se asignará al valor “Agregar” ya que el formulario se invocará a través de este constructor con dicho propósito.

Luego sobrecargaremos el constructor pidiendo en este caso el Id del Item seleccionado en el control dataGridViewPedido. La selección del Item a modificar (y posteriormente a eliminar) se hará a través de controles del tipo contextMenuStrip que agregaremos posteriormente. La modalidad se asignará en este caso al valor “Modificar” ya que el formulario se invocará a través de este constructor con dicho propósito.

Finalmente capturaremos el evento Click del control buttonAceptar y según sea la modalidad de apertura, invocaremos al método admCompras.Agregar() ó admCompras.Modificar().

Paso 15

Regresamos ahora al procedimiento DragDrop del control dataGridViewPedido e implementamos el código necesario para invocar al formulario frmPedido con los parámetros adecuados para agregar un item a la compra.

Para saber cómo extraer el valor del Id del producto arrastrado sobre el control, consultar el código de ejemplo.

Paso 16

Ahora deberemos crear un procedimiento llamado ListarPedido que como su nombre indica mostrará el contenido de la compra en el control dataGridViewPedido.

Para ello asignar al dataGridView el valor de la propiedad Items de la clase admCompras a su propiedad DataSource y realizar los ajustes correspondientes a sus columnas con los criterios definidos en el paso 7.

Paso 17

Agregar al formulario principal un control del tipo contextMenuStrip. El mismo se posicionará automáticamente fuera del mismo. Nombrarlo contextMenuStripDataGridViewPedido.

Agregar al mismo dos elementos del tipo MenuItem a través de la propiedad Items. El primero se llamará toolStripMenuItemModificar y el otro toolStripMenuItemEliminar. Los valores de sus propiedades Text serán respectivamente “Modificar” y “Eliminar”. Asociar un procedimiento a sus respectivos eventos Click.

En el control DataGridViewPedido, asignar a la propiedad ContextMenu el nombre del menu de contexto que hemos agregado.

En el procedimiento que permite modificar un item invocaremos nuevamente al formulario frmPedido pero esta vez a través del constructor que solicita el Id del item y la modalidad, que será evidentemente “Modificar”. Para ver cómo obtener el Id del item seleccionado consultar el código de ejemplo.

En el procedimiento que permite eliminar, utilizando un MessageBox pedir confirmación al usuario y, en caso afirmativo, invocar al método admCompras.Eliminar() pasándole como parámetro el identificador del item a eliminar, como se ha hecho en el caso de la modificación.

Paso 18

Se implementará a continuación parte de la funcionalidad de los controles toolStripButtonAbrirVenta, toolStripButtonCancelarVenta y toolStripButtonCerrarVenta.

Para ello asociaremos un procedimiento a cada uno de sus eventos Click.

En dichos procedimientos invocaremos un procedimiento llamado HabilitarPedido() que se definirá a continuación pasándole un parámetro del tipo booleano.

Si el procedimiento HabilitarPedido() se le pasa el valor verdadero, deberá deshabilitar el botón toolStripButtonAbrirVenta y habilitar los botones toolStripButtonCancelarVenta y toolStripButtonCerrarVenta. Si se le pasa el valor falso, hará las operaciones inversas a las definidas.

Entonces, cuando se pulse el botón toolStripButtonAbrirVenta se invocará al procedimiento con el valor verdadero, mientras que cuando se pulsen los restantes botones se lo invocará con el valor falso.

Para el evento asociado a la cancelación de la venta se procederá además a invocar el método admCompras.Cancelar() e igualar la propiedad DataSource del control dataGridViewPedido al valor nulo.

Además, actualizará el control labelTotal asignando el valor del atributo admCompras.Total.

Para el evento asociado al cierre de la venta se procederá además a verificar que exista al menos un cliente seleccionado y un producto seleccionado. Caso contrario, se informará al usuario que no es posible cerrar la venta si cualquiera de éstas condiciones no se cumplen.

Por ahora finalizaremos el método invocando al método admCompras.Cancelar() e igualar la propiedad DataSource del control dataGridViewPedido al valor nulo y actualizaremos el valor del atributo Text del control labelTotal.

En la fase siguiente, cuando se codifique el proceso de negocio relacionado con la documentación comercial, se procederá a completar el cierre de la venta y reemplazaremos éstas 3 últimas líneas de código.

Fase 09

Introducción

En esta fase completaremos los métodos de datos usando ADO conectado y ADO desconectado.

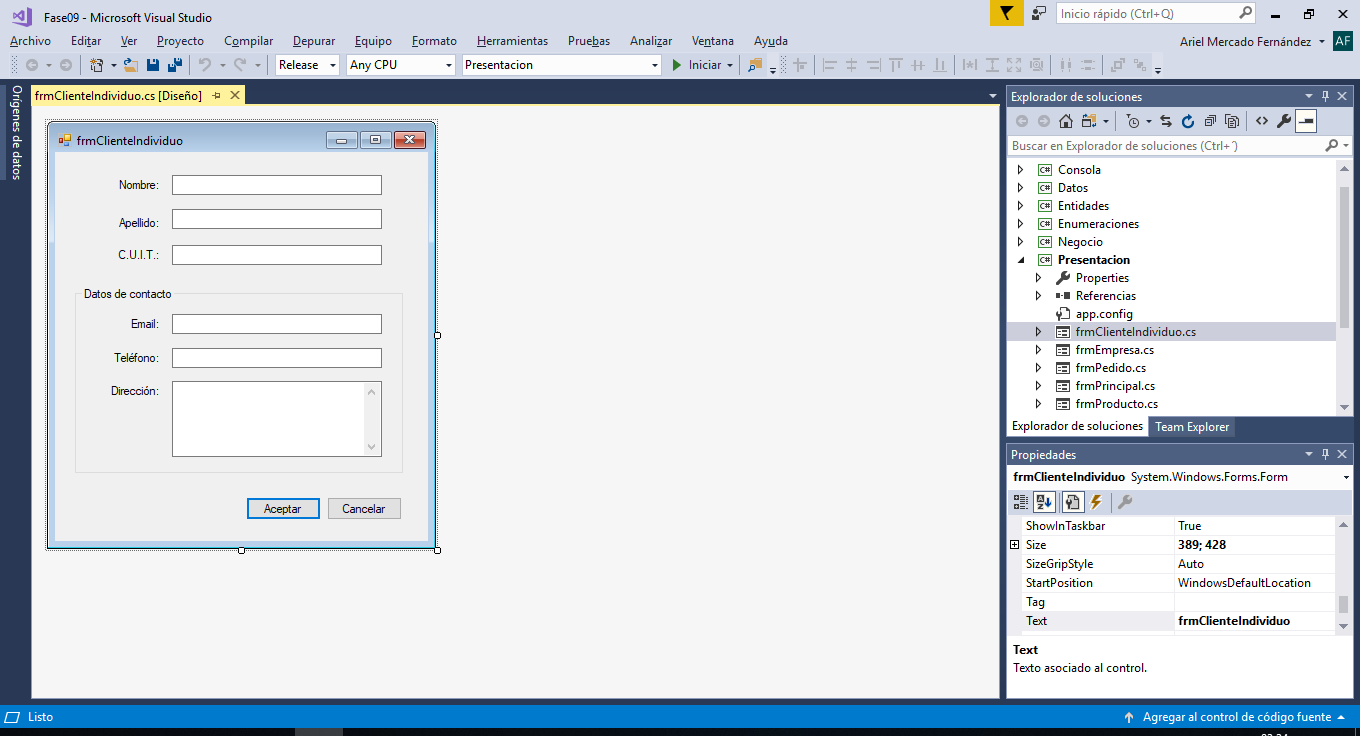
Habilidades y conocimientos aplicados en esta fase

Codificar métodos utilizando las clases del modelo ADO desconectado.

Métodos con ADO conectado

Paso 1

Crear un formulario para crear y modificar clientes individuo. Llamarlo frmClienteIndividuo. El diseño debería verse similar al siguiente:



Utilizar los siguientes controles:

textBoxNombre

textBoxApellido

textBoxCUIT

groupBoxDatosContacto

textBoxEmail

textBoxTelefono

textBoxDireccion

Paso 2

Codificar los siguientes métodos utilizando ADO conectado:

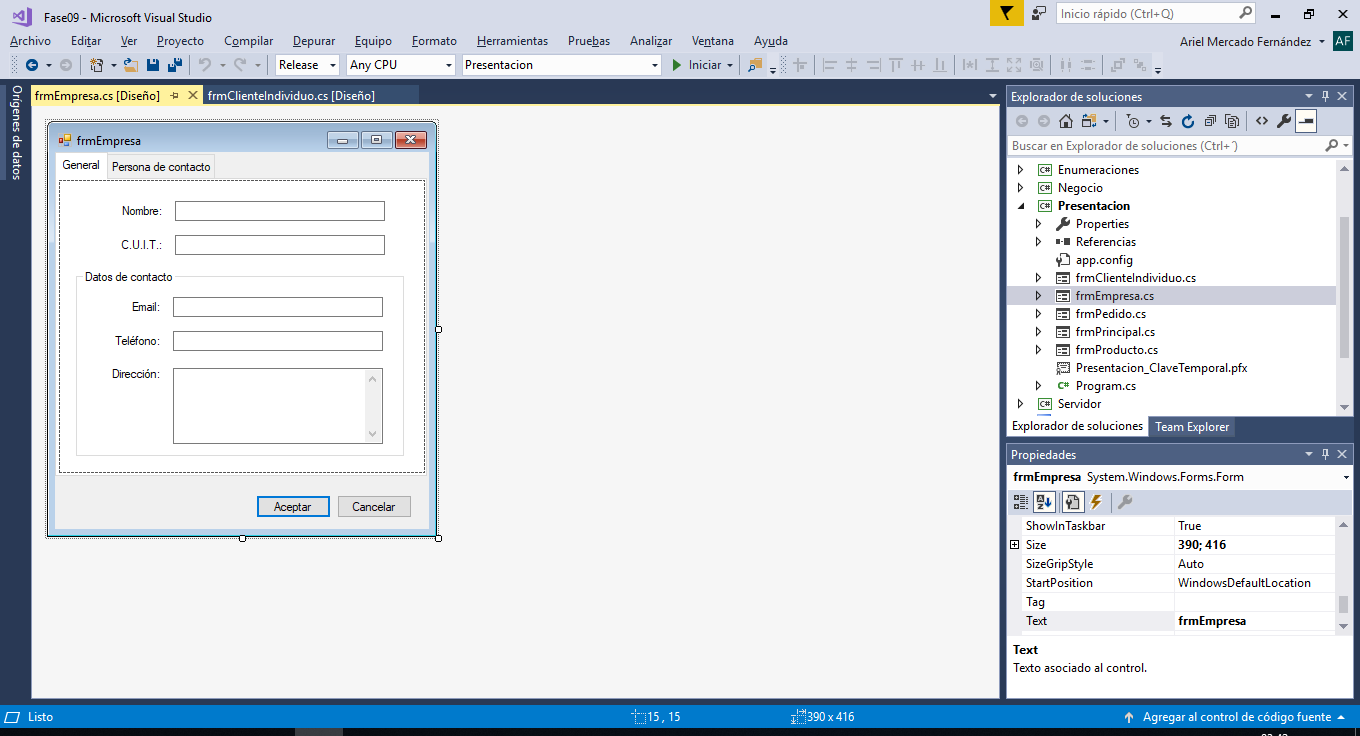
|  |  |  |
| --- | --- | --- |
| Clase / Método | Parámetros | Salida |
| dbClientes |  |  |
| Agregar() | Cliente ClienteIndividuo | - |
| Modificar() | Cliente ClienteIndividuo | - |
| Eliminar() | Cliente ClienteIndividuo | - |

Paso 3

Relacionarlos con el formulario y comprobar su funcionalidad.

Paso 4

Crear un formulario para crear y modificar empresas (clientes empresariales y proveedores). Llamarlo frmEmpresa. El diseño debería verse similar al siguiente:



Utilizar los siguientes controles:

tabControlEmpresa

tabPageGeneral

textBoxNombre

textBoxCUIT

groupBox1

textBoxEmail

textBoxTelefono

textBoxDireccion

tabPagePersonaContacto

textBoxNombreContacto

textBoxApellidoContacto

groupBox2

textBoxEmailContacto

textBoxTelefonoContacto

textBoxDireccionContacto

Paso 5

Codificar los siguientes métodos utilizando ADO conectado:

|  |  |  |
| --- | --- | --- |
| Clase / Método | Parámetros | Salida |
| dbClientes |  |  |
| Agregar() | Cliente ClienteIndividuo | - |
| Modificar() | Cliente ClienteIndividuo | - |
| Eliminar() | Cliente ClienteIndividuo | - |
| dbProveedores |  |  |
| Agregar() | Proveedor Empresa | - |
| Modificar() | Proveedor Empresa | - |
| Eliminar() | Proveedor Empresa | - |

Paso 6

Relacionarlos con el formulario y comprobar su funcionalidad.

Paso 7

Utilizando ADO conectado codificar los siguientes métodos:

|  |  |  |
| --- | --- | --- |
| Clase / Método | Parámetros | Salida |
| dbFacturas |  |  |
| Generar() | factura Factura | - |
| dbRemitos |  |  |
| Generar() | remito Remito | - |

Paso 8

Utilizando ADO desconectado con DataSets tipados codificar los siguientes métodos:

|  |  |  |
| --- | --- | --- |
| Clase / Método | Parámetros | Salida |
| dbFacturas |  |  |
| Listar() | orden | Lista de factura |
| ListarItems() | id(entero) | Lista de ítem |
| dbRemitos |  |  |
| Listar() | orden | Lista de remito |
| ListarItems() | id(entero) | Lista de ítem |

Paso 9

Codificar lo necesario en el formulario principal para visualizar la lista de facturas y remitos.

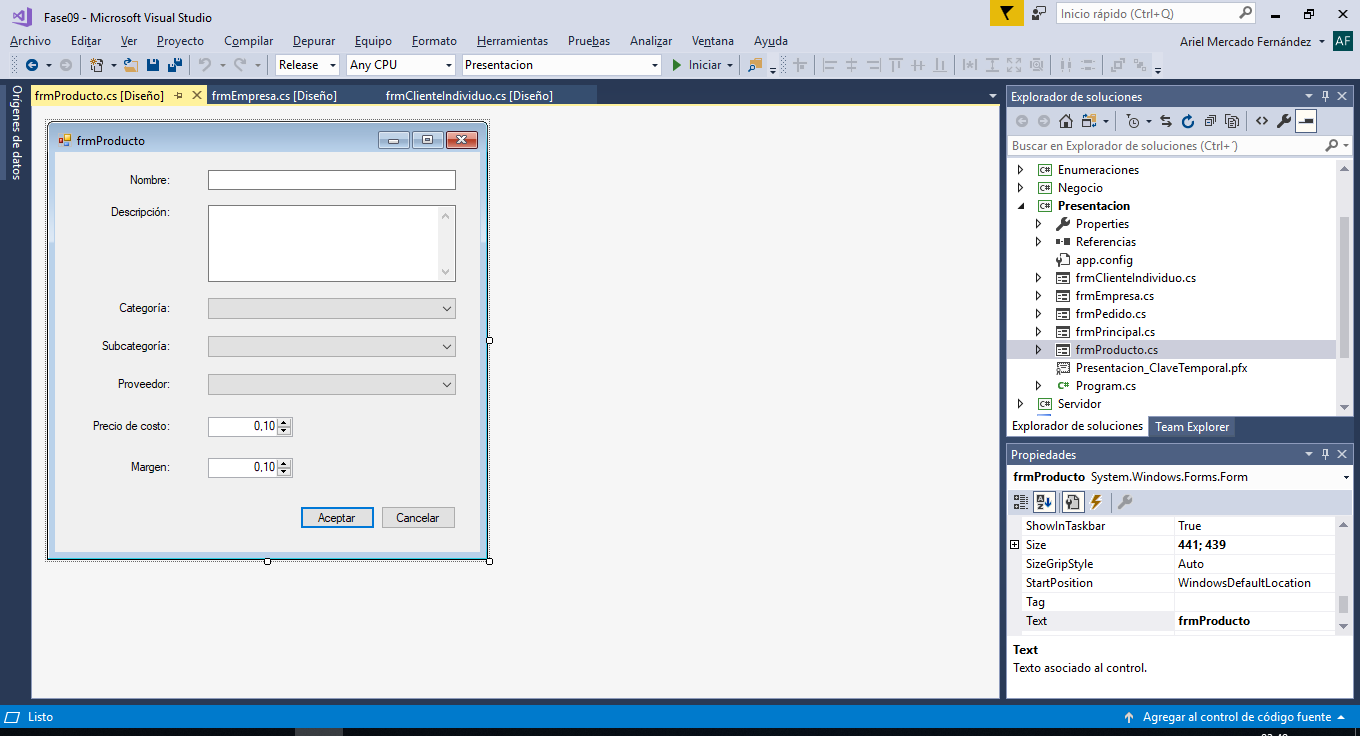
Paso 10

Utilizando ADO desconectado y DataSets tipados codificar los siguientes métodos:

|  |  |  |
| --- | --- | --- |
| Clase / Método | Parámetros | Salida |
| dbProductos |  |  |
| Agregar() | producto(Producto) | - |
| Modificar() | producto(Producto) | - |
| Eliminar() | id(entero) | - |

Paso 11

Crear un formulario para crear y modificar productos. Llamarlo frmProducto. El diseño debería verse similar al siguiente:



Utilizar los siguientes controles:

textBoxNombre

textBoxDescripcion

comboBoxCategoria

comboBoxSubcategoria

comboBoxProveedor

numericUpDownPrecioCosto

numericUpDownMargen

Paso 12

Relacionar los métodos de la clase admProductos con el formulario y comprobar su funcionalidad.