

2015

ABM de 1 capa en Visual Basic .NET

Programación Aplicada II.

Aprende a diseñar y construir un sistema de Alta, Baja y Modificación (ABM) de una entidad genérica.



1 Contenido

1) Objetivo del documento:	3
2) Alcances del documento:	3
3) Requerimientos:	3
4) Análisis:	3
5) Diseño:	4
5.1) Definición general de la solución:	4
5.2) Datos:	4
5.3) Interfaz gráfica de usuario:	4
5.4) Clases:	6
5.4.1) Métodos:	7
6) Construcción:.....	7
6.1) Base de Datos:.....	7
6.2) Solución y proyectos en Visual Studio .NET:.....	8
6.2.1) Interfaz Gráfica de Usuario:	10
6.2.2) Clase Producto:	12
6.2.2.1) Campos privados y Atributos públicos:	12
6.2.2.2) Métodos:	13
6.2.2.3) Constructores:	17
6.3. Programación de Botones y Procedimientos:	18
6.3.1) Botones Formulario Frm_Producto_Buscar	18
6.3.2) Procedimientos Formulario Frm_Producto_Buscar	19
6.3.3) Variable Formulario Frm_Producto_Alta	21
6.3.4) Botones Formulario Frm_Producto_Alta	21
6.3.5) Procedimientos Formulario Frm_Producto_Alta	22

1) Objetivo del documento:

Este documento, tiene como objetivo suplir el conocimiento suficiente para que el lector pueda diseñar y construir un ABM de cualquier entidad simple (atributos simples) sobre la plataforma de desarrollo Visual Basic .NET.

2) Alcances del documento:

- Comprensión de los requerimientos.
- Análisis, Diseño y modelado de la solución.
- Construcción de la solución en código fuente.

3) Requerimientos:

Los requerimientos son el pilar de todos los sistemas. Definen el comportamiento esperado de las aplicaciones desarrolladas a demanda. Es necesario comprenderlos para lograr cumplir con las expectativas que el cliente tiene sobre la solución informática que estamos ofreciendo. Mientras más específicos sean los requerimientos, menos ambigüedades se encontrarán en el momento de generar un diseño para la solución y menos re-trabajo vamos a tener como programadores.

Para este estudio, vamos a definir los siguientes requerimientos:

- Se requiere un sistema que registre, artículos para un negocio X.
- Todos los productos deben ser registrados almacenando los datos: nombre, rubro, precio, stock, fecha inicio de comercialización.
- El sistema debe permitir modificar cualquier dato existente de cualquier producto.
- El sistema debe contar con la funcionalidad de buscar los artículos por rubro y/o nombre.
- Se debe ofrecer la posibilidad de eliminar cualquier producto del sistema.

4) Análisis:

El cliente está solicitando un software que permita realizar altas, bajas, y modificaciones de una entidad determinada, cuyos atributos fueron suministrados en los requerimientos. El objetivo del cliente es tener información real referida a los productos que la empresa comercializa.

Plan:

- Diseñar la solución
- Construir la funcionalidad ALTA.
- Construir la funcionalidad MODIFICACIÓN.
- Construir la funcionalidad Baja.
- Testear integralmente.

5) Diseño:

El diseño de esta solución se plasma la generación de la información necesario que le suministre al programador, las instrucciones unívocas de qué debe hacer la aplicación y cómo debe estar construida.

5.1) Definición general de la solución:

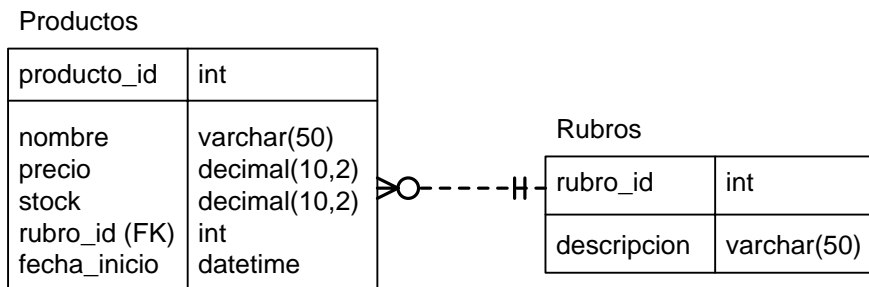
Dado el escenario que está definido por los requerimientos, se establece que la solución informática será construida con la plataforma tecnológica .NET en el lenguaje Visual Basic con Windows Forms como interfaz gráfica de usuario.

Los datos se almacenarán en una base de datos MS ACCESS.

La aplicación debe ser un archivo ejecutable que abra un menú donde pueda seleccionar la funcionalidad disponible en el sistema.

5.2) Datos:

Los datos serán almacenados en una base de datos que tiene la siguiente estructura:



5.3) Interfaz gráfica de usuario:

La interfaz de usuario debe ser un formulario de Windows que contenga los atributos que se desean almacenar y/o editar de la entidad producto.

Pantalla de alta de Producto:

Nuevo Producto

Código

1421

Nombre

Fideos 500 gr. Matarazzo

Precio \$

9.80

Stock

521

Rubro

Panadería

Verdulería

Juguetes

Bebidas

Pastas secas

Pastas frescas

Fecha Inicio

09/04/2011

Guardar

Actualizar

Eliminar

5.4) Clases:

Producto

Class

Fields

Properties

Methods

Codigo As Integer

FechaInicio As Date

Nombre As String

Precio As Double

Rubro As Integer

Stock As Double

Actualizar()

Buscar(nombre_par As String, rubro_par As Integer) As DataTable

Eliminar()

Insertar() As Integer

Modificar()

New()

New(codigo_par As Integer)

Obtener_Rubros() As DataTable

5.4.1) Métodos:

- New: Constructor vacío, se utiliza para instanciar la clase cuando se debe dar de alta a un artículo nuevo.
- New (codigo_par as Integer): Constructor con un parámetro, se utiliza para enviar un número entero que representa al código de un artículo. Con ese código, el sistema ubica un único registro en la base de datos, y construye una clase con los datos encontrados en la base de datos.
- Buscar: Es un método estático que devuelve un objeto "DataTable" utilizado para mostrar la información resultante en una grilla de la GUI. Este método recepciona los datos enviados por los parámetros con el fin de procesar la búsqueda.
- Insertar: Este método hace que todos los atributos de la clase instanciada, se almacenen en la base de datos generando un nuevo registro en la tabla "productos".
- Actualizar: Guarda en la base de datos los datos actuales de cada atributo de la instancia.
- Eliminar: Ejecuta la eliminación del registro seleccionado en la base de datos.
- Obtener_Rubros: Este método devuelve una tabla con todos los nombres y códigos disponibles obtenidos de la base de datos. Es utilizado para completar el dropdown list "cmb_rubros".

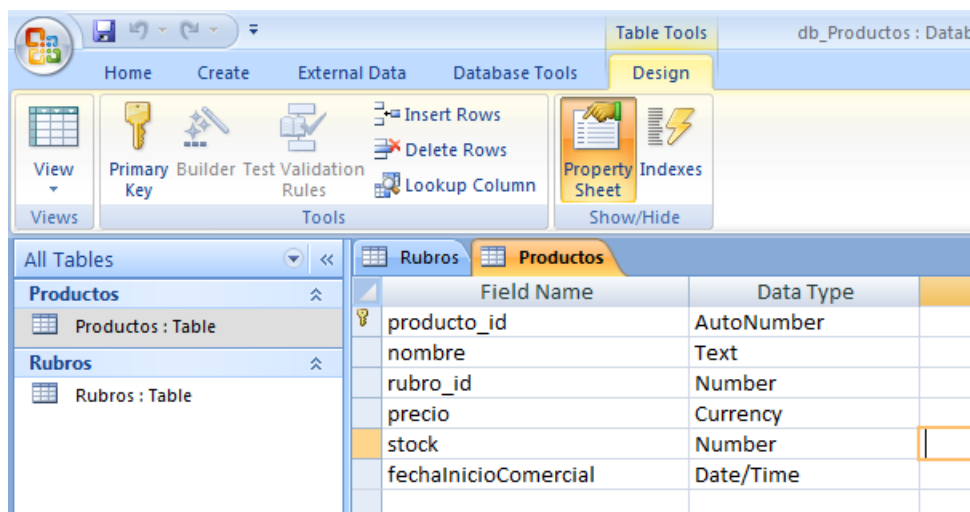
6) Construcción:

En la construcción, vamos a llevar a utilizar Visual Studio 2010 y Microsoft Access 2010.

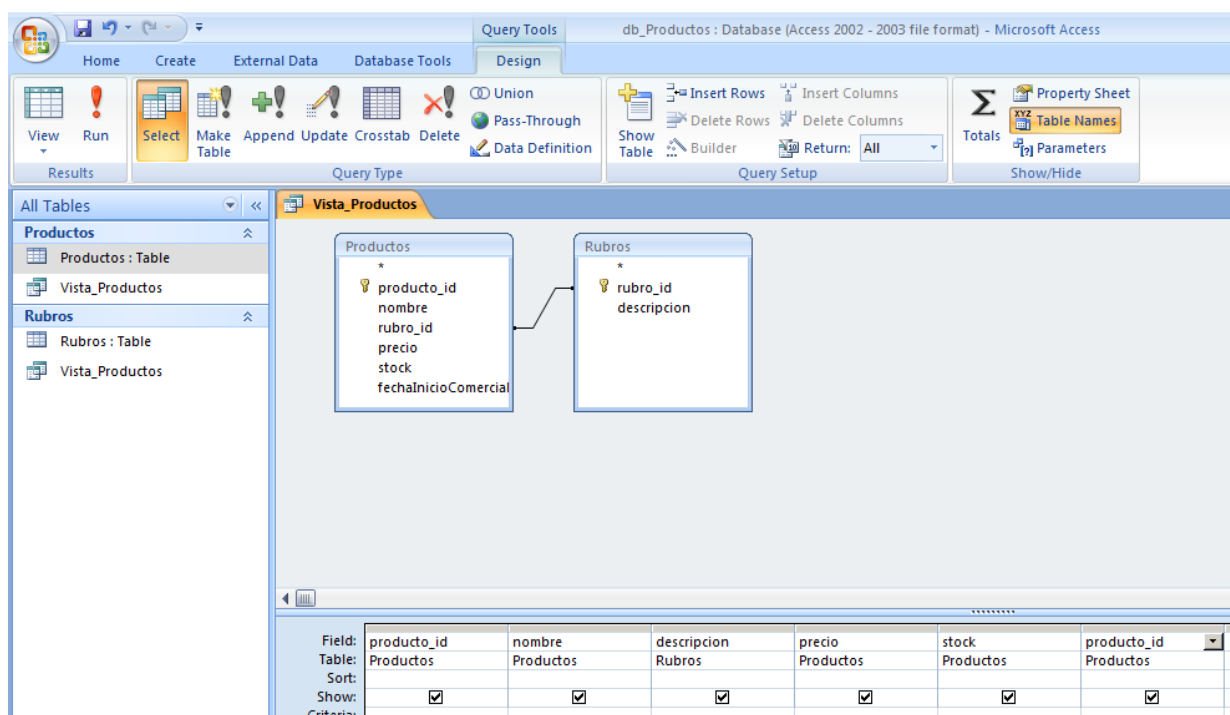
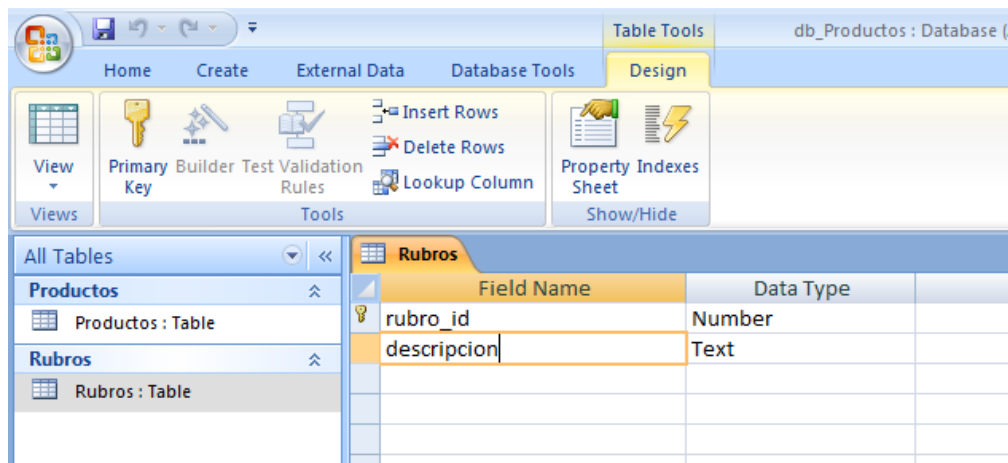
6.1) Base de Datos:

Para construir la sección que corresponde a la base de datos, vamos a seguir los siguientes ítems:

- Crear una base de datos llamada "db_productos.accdb".
- Crear una tabla llamada "Productos" según el diseño.
- Crear una tabla llamada "Rubros" según el diseño.
- Crear una vista llamada "Vista_Rubros" según el diseño.



Field Name	Data Type
producto_id	AutoNumber
nombre	Text
rubro_id	Number
precio	Currency
stock	Number
fechaInicioComercial	Date/Time



6.2) Solución y proyectos en Visual Studio .NET:

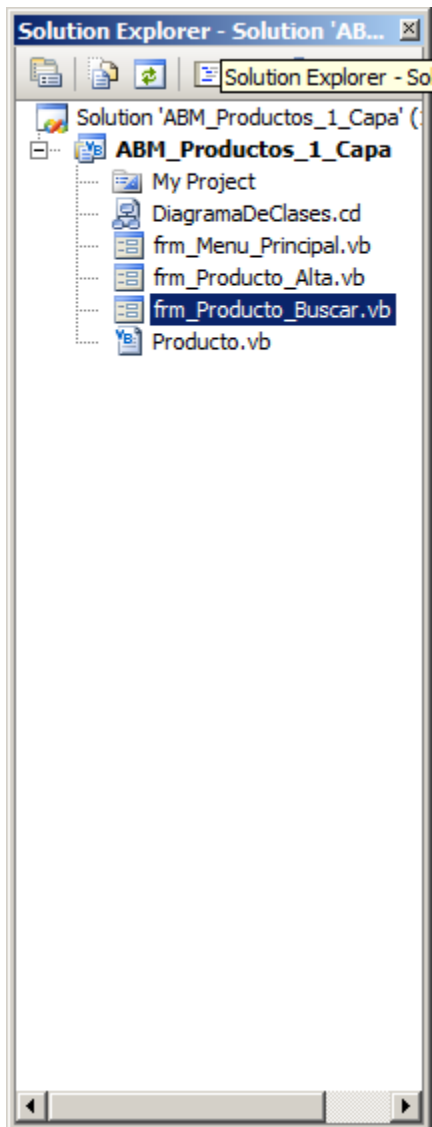
La sección de la construcción con la tecnología .net, se puede dividir en 2 partes:

- 1- Interfaz gráfica de usuario.
- 2- Código de procesamiento (reglas de negocio).

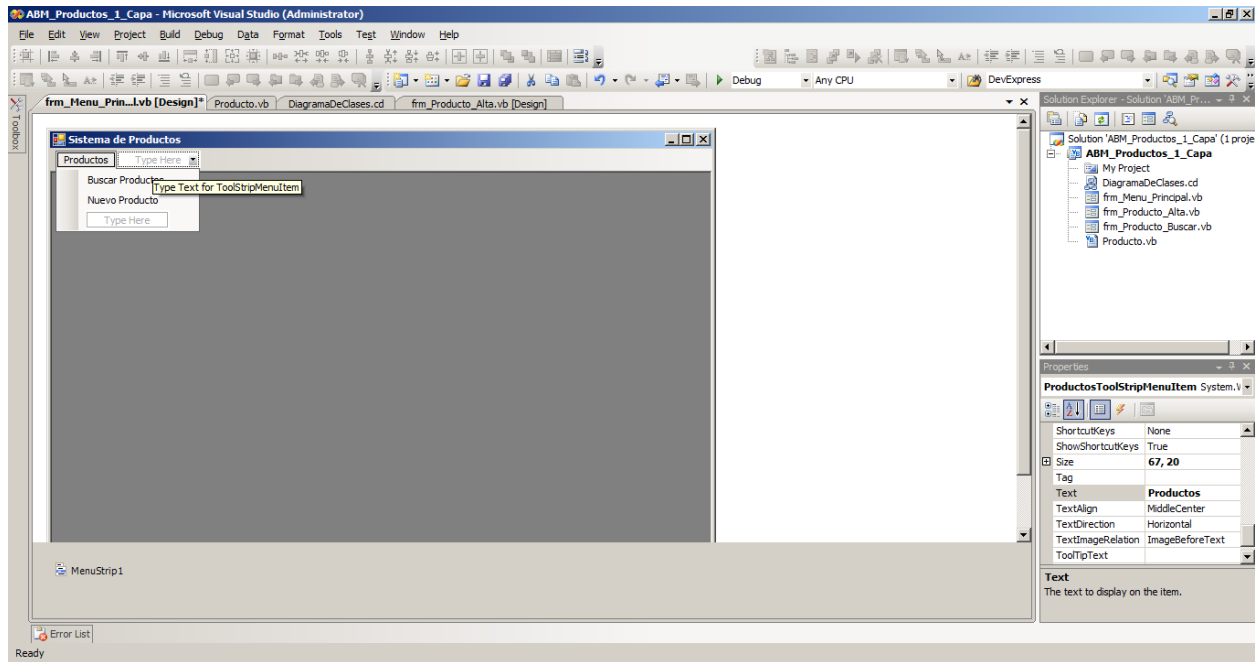
En la primera parte, se trabaja construyendo la pantalla que el usuario final utilizará una vez que el sistema está funcionando. Mientras que la otra parte, está oculta para el usuario final, pero es la parte que se encarga de procesar los datos y realiza las acciones que se esperan de la aplicación. Esta parte está compuesta solamente por código fuente. Esta segunda sección son las clases que serán llamadas por la interfaz gráfica para que los datos que sean procesados.

Vamos a seguir los siguientes pasos para trabajar con Visual Studio:

- Crear un proyecto en Visual Basic .NET llamado "ABM_Productos_1_Capa" de tipo "Windows Forms".
- Agregar un nuevo formulario con el nombre "frm_Menu_Principal".
- Agregar un nuevo formulario con el nombre "frm_Producto_Alta".
- Agregar un nuevo formulario con el nombre "frm_Producto_Buscar".
- Configurar el proyecto para que inicie con el formulario "frm_Menu_Principal"
- Crear una clase llamada Producto.



6.2.1) Interfaz Gráfica de Usuario:



En el evento click del primer ítem del menu (“Buscar Productos”) se debe agregar el siguiente código:

```
Dim frm = New frm_Producto_Buscar()  
frm.MdiParent = Me  
frm.Show()
```

En el evento click del segundo ítem del menu (“Nuevo Producto”) se debe agregar el siguiente código:

```
Dim frm = New frm_Producto_Alta()  
frm.MdiParent = Me  
frm.Show()
```

Luego se debe construir el formulario de Alta y Búsqueda:

Productos

Producto.vb DiagramaDeClases.cd frm_Producto_Alta.vb* frm_Produ

Nuevo Producto

Código

Nombre

Rubro

Precio \$

Stock

Fecha de Inicio 02/10/2012

Guardar Actualizar Eliminar

ABM_Productos_1_Capa - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Data Format Tools Test Window Help

frm_Producto_B...r.vb [Design]* frm_Menu_Principal.vb frm_Menu_Principal.vb [Design]

Buscar Productos

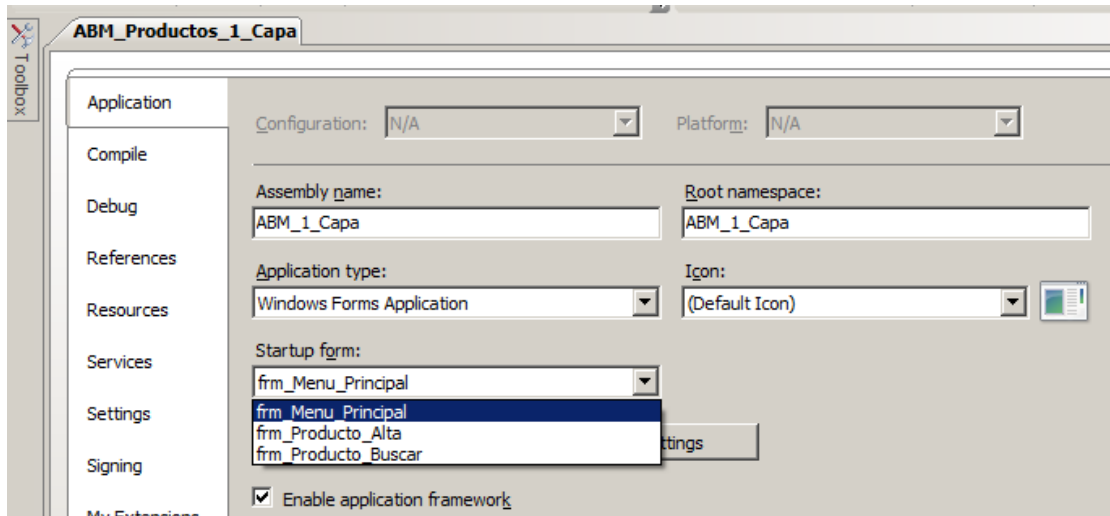
Filtros de Búsqueda

Nombre

Rubro

Buscar

Nuevo Editar Eliminar



6.2.2) Clase Producto:

Es necesario importar un namespace en la clase Producto:

```
Imports System.Data.OleDb
```

6.2.2.1) Campos privados y Atributos públicos:

```
Private _codigo As Integer
Private _nombre As String
Private _rubro As Integer
Private _stock As Double
Private _fechaInicio As DateTime
Private _precio As Double

Public Property Codigo() As Integer
    Get
        Return _codigo
    End Get
    Set(ByVal value As Integer)
        _codigo = value
    End Set
End Property

Public Property Nombre() As String
    Get
        Return _nombre
    End Get
    Set(ByVal value As String)
        _nombre = value
    End Set
End Property

Public Property Rubro() As Integer
```

```

        Get
            Return _rubro
        End Get
        Set(ByVal value As Integer)
            _rubro = value
        End Set
    End Property

    Public Property Precio() As Double
        Get
            Return _precio
        End Get
        Set(ByVal value As Double)
            _precio = value
        End Set
    End Property

    Public Property Stock() As Double
        Get
            Return _stock
        End Get
        Set(ByVal value As Double)
            _stock = value
        End Set
    End Property

    Public Property FechaInicio() As DateTime
        Get
            Return _fechaInicio
        End Get
        Set(ByVal value As DateTime)
            _fechaInicio = value
        End Set
    End Property

```

6.2.2.2) Métodos:

Método "Insertar()":

Este método debe tomar cada uno de los valores de los atributos de la instancia de la clase, y almacenarlos en la tabla correspondiente en la base de datos.

```

    Public Function Insertar() As Integer

        'Declaro una variable tipo String que contiene la ESTRUCTURA de la
        'sentencia SQL
        'La sentencia SQL INSERT es la que inserta un nuevo registro en la
        'tabla especificada.
        Dim cmd_insertar = "INSERT INTO Productos (nombre, rubro_id, precio,
        stock, fechaInicioComercial) "
        & " VALUES (@nombre, @rubro_id, @precio, @stock,
        @fechaInicioComercial)"
    End Function

```

```

        'Declaro una variable tipo OleDbConnetion y la contruyo enviando la
cadena de conexión.
        'En este caso, la cadena de conexión está configurada en las
"settings" de la aplicación
        Dim conexion = New OleDbConnection(My.Settings.My_DataBase)

        'Abro la conexión a la base de datos
conexion.Open()

        'Declaro una variable tipo OleDbCommand para ejecutar el comando
especificado anteriormente ("cmd_insertar")
        'Construyo la variable enviando la conexión ("cmd_insertar") y el
comando SQL ("conexion")
        Dim comando As New OleDbCommand(cmd_insertar, conexion)

        'Por medio de los siguientes pasos, se le especifica a cada parámetro
de la sentencia SQL que debe ser reemplazado por
        'los valores que se especifican en cada caso.

comando.Parameters.Add(New OleDbParameter("@nombre", Me.Nombre))
comando.Parameters.Add(New OleDbParameter("@rubro_id", Me.Rubro))
comando.Parameters.Add(New OleDbParameter("@precio", Me.Precio))
comando.Parameters.Add(New OleDbParameter("@stock", Me.Stock))
comando.Parameters.Add(New OleDbParameter("@fechaInicioComercial",
Me.FechaInicio))

        'Se ejecuta la sentencia SQL en la base de datos. Este método
devuelve un número entero que significa cuantos registros
        'han sido afectados por la sentencia SQL.
comando.ExecuteNonQuery()

        'Obtengo el ultimo ID generado automáticamente en la base de datos
Dim cmd_select = "select @@identity"
comando = New OleDbCommand(cmd_select, conexion)
Dim dt = New DataTable()
Dim da = New OleDbDataAdapter(comando)
da.Fill(dt)
Me.Codigo = Convert.ToInt32(dt.Rows(0)(0)) 'Obtengo el valor y se lo
asigno al atributo Codigo.

        'Cierro la conexión a la base de datos
conexion.Close()

        'Devuelvo el código generado automáticamente tras la inserción en la
tabla
        Return Me.Codigo

End Function

```

Método “Actualizar()”:

Ejecuta una sentencia SQL con el fin de actualizar los valores de un registro existente. Actualiza el registro con los valores de los atributos de la instancia generada.

```
'=====

Public Sub Actualizar()

    Dim cmd_update = "UPDATE Productos SET (nombre=@nombre,
rubro_id=@rubro_id, precio=@precio, stock=@stock,
fechaInicioComercial=@fechaInicioComercial) WHERE producto_id=@producto_id"

    Dim conexion = New OleDbConnection(My.Settings.My_DataBase)
    conexion.Open()

    Dim comando As New OleDbCommand(cmd_update, conexion)

    comando.Parameters.Add(New OleDbParameter("@nombre", Me.Nombre))
    comando.Parameters.Add(New OleDbParameter("@rubro_id", Me.Rubro))
    comando.Parameters.Add(New OleDbParameter("@precio", Me.Precio))
    comando.Parameters.Add(New OleDbParameter("@stock", Me.Stock))
    comando.Parameters.Add(New OleDbParameter("@fechaInicioComercial",
Me.FechaInicio))
    comando.Parameters.Add(New OleDbParameter("@producto_id", Me.Codigo))

    comando.ExecuteNonQuery()
    conexion.Close()

End Sub

'=====
```

Método “Buscar(nombre_buscado, rubro_buscado)”:

Este método devuelve un objeto DataTable, que es una tabla que contiene los datos de la sentencia SQL de tipo SELECT que permite recuperar datos de una base de datos. Se le envían parámetros para poder utilizarlos como filtros de la búsqueda.

```
'=====

Public Shared Function Buscar(ByVal nombre_par As String, ByVal rubro_par
As Integer) As DataTable

    Dim cmd_select = "SELECT * FROM Vista_Productos WHERE nombre LIKE @nombre "

    Dim busca_rubro = False

    If rubro_par <> 0 Then

        cmd_select += " AND rubro_id=@rubro_id"
```

```

        busca_rubro = True

    End If

    Dim conexion = New OleDbConnection(My.Settings.My_DataBase)
    conexion.Open()

    Dim comando As New OleDbCommand(cmd_select, conexion)

    comando.Parameters.Add(New OleDbParameter("@nombre", "%" & nombre_par
& "%"))

    If busca_rubro Then
        comando.Parameters.Add(New OleDbParameter("@rubro_id",
rubro_par))
    End If

    Dim dt = New DataTable()

    Dim da = New OleDbDataAdapter(comando)

    da.Fill(dt)

    Return dt

End Function

```

Método “Eliminar()”:

Al invocar este método, la clase debe eliminar el registro que corresponda a la instancia generada.

Se ejecuta una sentencia SQL de tipo DELETE con el fin de eliminar el registro cuya clave primaria sea el código del producto.

```

Public Sub Eliminar()

    Dim cmd_update = "DELETE Productos WHERE producto_id=@producto_id"

    Dim conexion = New OleDbConnection(My.Settings.My_DataBase)
    conexion.Open()

    Dim comando As New OleDbCommand(cmd_update, conexion)

    comando.Parameters.Add(New OleDbParameter("@producto_id", Me.Codigo))

    comando.ExecuteNonQuery()
    conexion.Close()

```


End Sub

Método "Obtener_Rubros()":

Este método debe devolver el listado de los rubros disponibles en la base de datos, servirá para alimentar a los dropdownlist. Debe devolver un código y un nombre por cada registro de rubros.

```
Public Shared Function Obtener_Rubros() As DataTable

    Dim cmd_select = "SELECT * FROM Rubros"

    Dim conexion = New OleDbConnection(My.Settings.My_DataBase)
    conexion.Open()

    Dim comando As New OleDbCommand(cmd_select, conexion)

    Dim dt = New DataTable()

    Dim da = New OleDbDataAdapter(comando)

    da.Fill(dt)

    Return dt

End Function
```

6.2.2.3) Constructores:

Constructor New() vacío:

Este constructor debe estar especificado porque será utilizado para construir una instancia de la clase Producto en el momento de guardar un producto nuevo.

```
Public Sub New()

End Sub
```

Constructor New(codigo_producto) con 1 parámetro (Integer) :

Este constructor debe ser invocado cuando se desea obtener una instancia de la clase producto que contenga todos los atributos con valores obtenidos de la base de dato, o sea, un producto existente.

```
'=====

Public Sub New(ByVal codigo_par As Integer)

    Dim cmd_select = "SELECT * FROM Productos WHERE producto_id =
@producto_id "

    Dim conexion = New OleDbConnection(My.Settings.My_DataBase)
    conexion.Open()

    Dim comando As New OleDbCommand(cmd_select, conexion)

    comando.Parameters.Add(New OleDbParameter("@producto_id",
codigo_par))

    Dim dt = New DataTable()

    Dim da = New OleDbDataAdapter(comando)

    da.Fill(dt)

    Me.Codigo = Convert.ToInt32(dt.Rows(0) ("producto_id"))
    Me.Nombre = Convert.ToString(dt.Rows(0) ("nombre"))
    Me.Rubro = Convert.ToInt32(dt.Rows(0) ("rubro_id"))
    Me.Precio = Convert.ToDouble(dt.Rows(0) ("precio"))
    Me.Stock = Convert.ToDouble(dt.Rows(0) ("stock"))
    Me.FechaInicio =
Convert.ToDateTime(dt.Rows(0) ("fechaInicioComercial"))

End Sub

'=====
```

6.3. Programación de Botones y Procedimientos:

6.3.1) Botones Formulario Frm_Producto_Buscar

Btn_Buscar:

```
Private Sub btn_buscar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_buscar.Click

    Try
        Buscar()
    Catch ex As Exception
```

```

        MessageBox.Show(ex.Message)
    End Try

End Sub

```

Btn_Nuevo:

```

Private Sub btn_Nuevo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Nuevo.Click

    Nuevo_Producto()

End Sub

```

Btn_Editar:

```

Private Sub btn_editar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_editar.Click

    Try

        Editar_Producto()

    Catch ex As Exception

        MessageBox.Show("Error al abrir el producto")

    End Try

End Sub

```

Btn_Eliminar:

```

Private Sub btn_Eliminar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Eliminar.Click

    Dim codigo =
Convert.ToInt32(DataGridView1.SelectedRows(0).Cells(0).Value)
    Dim producto_seleccionado As New Producto(codigo)
    producto_seleccionado.Eliminar()
    Buscar()

End Sub

```

6.3.2) Procedimientos Formulario Frm_Producto_Buscar

CONSTRUCTOR :

```
Public Sub New()  
  
    ' This call is required by the Windows Form Designer.  
    InitializeComponent()  
  
    Try  
  
        Prepara_Interfaz()  
  
    Catch ex As Exception  
  
        MessageBox.Show("No se pudo preparar la interfaz")  
  
    End Try  
  
End Sub
```

Procedimientos:

```
Private Sub frm_Producto_Buscar_Load(ByVal sender As System.Object, ByVal e  
As System.EventArgs) Handles MyBase.Load
```

```
    btn_buscar.PerformClick()  
End Sub
```

```
Sub Buscar()  
  
    Dim text_buscado = txt_nombre.Text  
    Dim rubro_id = Convert.ToInt32(cmb_rubro.SelectedValue)  
  
    DataGridView1.DataSource = Producto.Buscar(text_buscado, rubro_id)  
  
End Sub
```

```
Sub Prepara_Interfaz()  
  
    cmb_rubro.DataSource = Producto.Obtener_Rubros()  
    cmb_rubro.DisplayMember = "descripcion"  
    cmb_rubro.ValueMember = "rubro_id"  
  
End Sub
```

```
Sub Nuevo_Producto()  
  
    Dim frm = New frm_Producto_Alta()  
    frm.MdiParent = Me.MdiParent  
    frm.Show()
```

```
End Sub
```

```
Sub Editar_Producto()
```

```

        Dim codigo =
Convert.ToInt32(DataGridView1.SelectedRows(0).Cells(0).Value)

        Dim producto_seleccionado As New Producto(codigo)

        Dim frm = New frm_Producto_Alta(producto_seleccionado)
        frm.MdiParent = Me.MdiParent
        frm.Show()

End Sub

```

6.3.3) *Variable Formulario Frm_Producto_Alta*

Declaraciones:

```
Dim Producto_Actual As Producto
```

6.3.4) *Botones Formulario Frm_Producto_Alta*

btn_guardar:

```

Private Sub btn_guardar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_guardar.Click

    Try

        Guardar()

    Catch ex As Exception

        MessageBox.Show(ex.Message)

    End Try

End Sub

```

btn_Eliminar:

```

Private Sub btn_Eliminar_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btn_Eliminar.Click
    Try

        Eliminar()

    Catch ex As Exception

```

```

        MessageBox.Show(ex.Message)

    End Try
End Sub

```

6.3.5) *Procedimientos Formulario Frm_Producto_Alta*

CONSTRUCTORES :

```

Public Sub New()

    'Se debe llamar a este método para que se generen los controles
    automáticamente
    InitializeComponent()

    ' Cualquier accion debe escribirse a partir de esta línea
    Preparar_Interfaz()

End Sub

```

```

Public Sub New(ByRef producto_parametro As Producto)

    InitializeComponent()
    Preparar_Interfaz()
    Producto_Actual = producto_parametro
    Clase_A_Interfaz()

End Sub

```

Procedimientos:

```

Sub Preparar_Interfaz()

    cmb_rubro.DataSource = Producto.Obtener_Rubros()
    cmb_rubro.ValueMember = "rubro_id"
    cmb_rubro.DisplayMember = "descripcion"

End Sub

```

```

Sub Clase_A_Interfaz()

    txt_codigo.Text = Producto_Actual.Codigo.ToString()
    txt_nombre.Text = Producto_Actual.Nombre
    txt_precio.Text = Producto_Actual.Precio.ToString()
    txt_stock.Text = Producto_Actual.Stock.ToString()
    dtp_fechaInicio.Value = Producto_Actual.FechaInicio

End Sub

```

```

        cmb_rubro.SelectedValue = Producto_Actual.Rubro.ToString()

End Sub

Sub Interfaz_A_Clase()

    If (Producto_Actual Is Nothing) Then
        Producto_Actual = New Producto()
    End If

    Producto_Actual.Nombre = txt_nombre.Text
    Producto_Actual.Precio = Convert.ToDouble(txt_precio.Text)
    Producto_Actual.Rubro = Convert.ToInt32(cmb_rubro.SelectedValue)
    Producto_Actual.Stock = Convert.ToDouble(txt_stock.Text)
    Producto_Actual.FechaInicio = dtp_fechaInicio.Value

End Sub

Sub Guardar()

    Interfaz_A_Clase()
    If (Producto_Actual.Codigo = 0) Then
        Producto_Actual.Insertar()
    Else
        Producto_Actual.Actualizar()
    End If
    Clase_A_Interfaz()
    MessageBox.Show("El producto se ha registrado correctamente")

End Sub

Sub Eliminar()

    Producto_Actual.Eliminar()
    MessageBox.Show("El producto se ha eliminado")
    Me.Close()

End Sub

```