# Programación Orientada a Objetos

Material de trabajo

## 1. Los Fundamentos.

**Objetivo:** Llevar a la práctica los Conceptos Fundamentales de las Programación Orientada a Objetos.
**Palabras Clave:** Clase, objeto, instancia, mensaje, atributo, método, evento, excepción.
**Desarrollo:** La práctica se divide en tres partes.

- Comprobar que las formas son definiciones de clases en VB6.
- Agregar un módulo de clase y definirle atributos y métodos mutadores y accesadores a la clase.
- Definirle propiedades y eventos a las instancias de la clase. Manejar los eventos desde su contenedor.

### 1.1. Creación del proyecto y definición de opciones.

- Defina un directorio en el disco duro, póngale un nombre apropiado, por ejemplo "MisPracticasPOO.
- Dentro de ese directorio construya otro para este primer proyecto. Llámelo "Sesion1"
- Invoque Visual Basic 6 y construya un nuevo proyecto de tipo standard.exe.
- Llámelo "PrjVBSesion1.vbp" y guárdelo en el directorio anteriormente referido.
- En el menú Herramientas -> Opciones, pestaña Editor seleccione la opción "Requerir declaración de Variables" ( Esto hará que el compilador detecte las variables que no han sido explícitamente declaradas ).
- En la pestaña "General" seleccione la opción "Interrupción en todos los errores".
- En esa misma pestaña elimine la selección de "Compilar a Petición".
- Salga del cuadro de diálogo con el botón de "OK".
- Salve el proyecto para que asimismo salve las opciones seleccionadas.

### 1.2. Agregando una forma y un botón.

- Agregue una forma, llámela frmMiForma.
- Agregue un botón de comando, llámelo "cmdBoton1" ( en la propiedad Name ), y póngale el letrero Botón1 en la propiedad Caption.
- Accese el código correspondiente al clic del botón anterior ( cmdBoton1_Click( ) ). Esto lo puede hacer dando doble clic al botón cmdBoton1.
- Agregue ahí el siguiente código:

```
Dim x as new frmMiForma
x.show
```

- Ejecute su proyecto ( por medio del botón “Start” o “Iniciar” de la barra de herramientas ).
- ¿Qué ocurre al dar clic al botón Botón1?. Explique lo que observa:

- ¿Puede acceder a la forma original y darle de nuevo Clic al botón?, Explique lo que ocurre:

- Explique porqué no se pierden las formas aunque aparentemente el ámbito de los objetos ha fenecido.

- Pase el proyecto a modo diseño ( o sea deténgalo ).
- Modifique el código para que el mensaje show al objeto x diga

```
x.show vbModal
```

- Ejecútelo y explique lo que ocurre:

- Salve su proyecto.

1.3. Agregando un módulo de clase.

En este apartado agregaremos un módulo de clase. Esta es la declaración de clase típica en VB6. En esta ocasión lo haremos totalmente “a mano”.

- Inserte un módulo de clase al proyecto ( Proyecto->Agregar Módulo de Clase )
- Nombre la clase clsMIClase1
- Agregue el siguiente código al módulo de clase:

```
Option Explicit                 ' este ya debe estar

Private strNombre As String
Private intDato As Integer

Public Sub AsignaNombre(unNombre As String)
' Mutador de strNombre
  strNombre = unNombre
End Sub

Public Function entregaNombre() As String
' Accesador de strNombre
```

```
  entregaNombre = strNombre
End Function

Public Sub AsignaDato(unDato As Integer)
' Mutador de intDato
  intDato = unDato
End Sub

Public Function entregaDato() As Integer
' Accesador de intDato
  entregaDato = intDato
End Function
```

1.4. Instanciado objetos de tipo clsMiClase1

- Agregue un nuevo botón a su forma, llámelo cmdBoton2, con caption Botón2.
- En el eveto clic del cmdBoton2 agregue el siguiente código:

```
Dim x As New clsMIClase1

x.AsignaNombre "Un nombre apropiado para el objeto"
x.AsignaDato 100

MsgBox x.entregaNombre & "   " & x.entregaDato
```

- Ejecute el programa y observe lo que ocurre.
- Elimine el vbModal del x.show del código del cmdBoton1. Despliegue varias formas. Cada una de ellas puede generar una instancia de la clase clsMiClase1. Explique porqué es esto:

- Salve su proyecto.

1.5. Trabajando con propiedad.

A continuación haremos que las interfases de acceso y mutación a los atributos de clsMiClase1 sean más “profesionales”. Deseamos que el uso de los métodos de acceso y mutación sea del estilo:

```
Dim z as new clsMiClase1
z.nombre = “Este es el nombre del Objeto”
z.dato = 100

MsgBox z.nombre & “ “ & z.dato
```

Para ello:

- Agregue el siguiente código dentro del módulo clsMiClase1

```
Public Property Let nombre(unNombre As String)
' nuevo mutador
  strNombre = unNombre
End Property

Public Property Get nombre() As String
'nuevo accesador
  nombre = strNombre
End Property

Public Property Let dato(unDato As Integer)
'nuevo mutador
  intDato = unDato
End Property

Public Property Get dato() As Integer
' nuevo accesador
  dato = intDato
End Property
```

- Modifique el código dentro del clic del botón cmdBoton2

```
  Dim x As New clsMIClase1

'  x.AsignaNombre "Un nombre apropiado para el objeto"
'  x.AsignaDato 100
'
'  MsgBox x.entregaNombre & "   " & x.entregaDato

  x.nombre = "Nombre para el objeto"
  x.dato = 200
  MsgBox x.nombre & " " & x.dato
```

- Ejecute su programa, observe que desde el punto de vista del usuario no hay mayor impacto, ¿qué opina desde el punto de vista del desarrollador?

- Salve su proyecto.

1.6. Eventos y más eventos.

VisualBasic ofrece de manera sencilla la mecánica de eventos a disposición del programador, haciendo transparente la resolución de la ranura de código que ha de soportar la ejecución de cada uno de los evento para las clases definidas por el usuario. Probaremos esto a continuación.

- Para ello, en el módulo de clsMiClase1 agregaremos la siguiente línea ( adicionalmente a la declaración de los atributos de las instancias )

```
Private strNombre As String
Private intDato As Integer
Public Event negativo() ‘ la negrita es solamente para que se vea aquí
```

- Y modifique el código de la propiedad mutadora de intDato

```
Public Property Let dato(unDato As Integer)
'nuevo mutador
 If unDato < 0 Then    ' Si se le trata de volver negativo
   RaiseEvent negativo ' Levantando la bander del evento
 Else
   intDato = unDato
 End If
End Property
```

- En la cabecera del código de frmMiForma1 agregue lo siguiente:

```
Option Explicit    ' este ya debe estar
Private WithEvents w As clsMIClase1
```

Obsérvese que NO SE HA INSTANCIADO OBJETO ALGUNO, solamente se ha declarado la variable para referenciar el objeto con eventos. Sin embargo en el combo de los objetos contenidos en la forma ya aparece el objeto w con su evento.

- Dentro del código de la forma frmMiForma1 programe el evento

```
Sub w_Negativo( )
```

con el siguiente aviso

```
MsgBox "Intento de volver negativo al dato..."
```

- Y en el evento initialize de la forma frmMiForma1( o en el evento Load )

```
Set w = New clsMIClase1
```

- Agregue un tercer botón ( cmdBoton3 , con caption Botón3 ), con el siguiente código en su evento click:

```
w.nombre = "Tercer nombre"
w.dato = 50
MsgBox w.nombre & " " & w.dato
```

- Ejecute el programa, oprimiendo el Botón3.
- Modifique el código cambiando el 50 de w.dato por un –50
- Explique lo que ocurre:

1.7. Previniendo los errores.

Ahora hemos de prevenir el error que le puede ocurrir al tratar de asignar como número algo que no lo es. Para ello:

- Modifiquemos el código de botón 2 de la forma frmMiForma1 para que se la asignación de la propiedad dato se vea así:

```
x.dato = 2E+22
```

- Ejecute el proyecto y oprima el botón 2, ¿Qué ocurre?

- En Herramientas->Opciones Pestaña “General” seleccione la opción “Interrupción en Errores No Controlados”

- Modifique ahora el código del Botón 2 para que quede como a continuación:

```
Private Sub cmdBoton2_Click()
  On Error GoTo AquiCachaElError
  Dim x As New clsMIClase1

'  x.AsignaNombre "Un nombre apropiado para el objeto"
'  x.AsignaDato 100
'
'  MsgBox x.entregaNombre & "   " & x.entregaDato

  x.nombre = "Nombre para el objeto"
  x.dato = 2E+22
  MsgBox x.nombre & " " & x.dato
  Exit Sub
AquiCachaElError:
```

```
MsgBox "cmdBoton2_Click:Caché el error " & Err.Number & " con descripción " & Err.Description
End Sub
```

- Salve su proyecto y salga de VB.

**TAREA:** Agregue cajas de texto para que Botón3 asigne valores al nombre y dato de w provenientes de las cajas de texto. Probablemente requiera CNum( Str ) para convertir string a número. Elimine después esta instrucción previniendo el error y procesándolo.

## 2. Elementos tecnológicos adicionales.

**Objetivo:** Poner en práctica el uso de colecciones. Estudiar la manera en que se defnen las responsabilidades de objetos subordinados a partir de los requerimientos impuestos a objetos de nivel jerárquico mayor.
**Palabras Clave:** Colección. Diagrama de Secuencia. Requerimiento funcional.
**Desarrollo:** La práctica se divide en dos partes.

- Manejo de la funcionalidad que ofrece VB para las colecciones de objetos.
- Diseñar e implantar las clases para soportar los requerimientos impuestos por una necesidad de usuario. Esto se hará diseñando y programando las clases que se detecten a partir de los requerimientos del usuario.

### 2.1. El Modelo.

Se nos ha encargado la implantación de una pequeña aplicación para poder representar una contabilidad básica. Por lo pronto solamente nos interesan las "Reglas de Negocio". Por ello, se llevará a cabo sin persistencia ( sin manejo de bases de datos ) y sin interfaz gráfica al usuario. La funcionalidad que debe satisfacer la contabilidad está dada por el siguiente "script":

```
Sub main( )
   Dim conta as new clsContabilidad
   Dim p0 as new clsPolizaCont
   With conta
         .asigna  "Empre S.A.", 2001
         .altaCta 100100,"Bancos                ", "D"
         .altaCta 100200,"Inventario            ", "D"
         .altaCta 200100,"Proveedores           ", "A"
         .altaCta 300000,"Capital               ", "A"
         .altaCta 400100,"Ventas                ", "A"
         .altaCta 500100,"Costo de lo Vendido", "D"
   End With

   With p0
         .num = 1
         .nombre = "Constitución de la empresa"
         .fecha = #10-Jan-2001#
         .cargo 100100,1000.0
         .abono 300000,1000.0
   End With

   conta.inicide p0

   conta.balance

End Sub
```

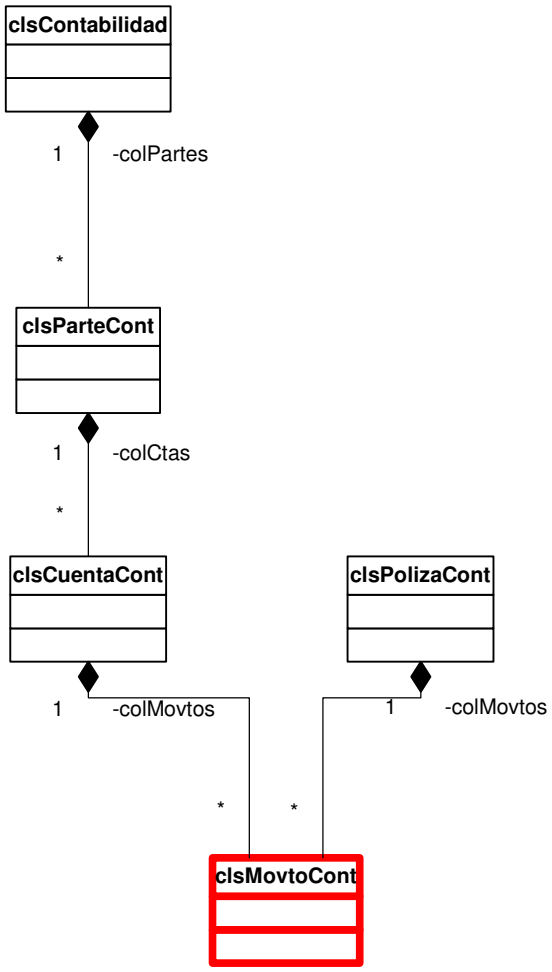
Complementado a esto, el contador explica que:

La contabilidad está constituida por 5 partes. Las partes tienen naturaleza, la cual es "Deudora" o "Acreedora" ( según se realice su saldo ).
Las partes tienen a su vez cuentas ( las que se requieran ). Las cuentas a su vez pueden ser "Deudoras" o "Acreedoras" ( también conforme se obtenga su saldo ). Las cuentas, en su interior tienen movimientos, cada uno de los cuales puede ser un cargo o un abono. Éstos provienen de los movimientos que traen las pólizas que se inciden en la contabilidad.

<table>
<tr><th>Elemento</th><th>Contiene</th><th>Puede ser</th><th>Observaciones</th></tr>
<tr><td>Contabilidad</td><td>Partes Contables</td><td></td><td>Hay 5 partes:<br>1 ... Activo , Deudora<br>2 ... Pasivo, Acreedora<br>3 ... Capital, Acreedora<br>4 ... Ingresos, Acreedora<br>5 ... Egresos, Deudora</td></tr>
<tr><td rowspan="2">Parte Contable</td><td rowspan="2">Atributos:<br>Identificador, nombre y naturaleza.<br>Contiene:<br>Cuentas Contables que tengan el sexto dígito igual a su identificador</td><td>Deudora</td><td>Se salda del lado del debe ( cargos ), como<br>Suma de saldos de sus cuentas deudoras – suma de los saldos de sus cuentas acreedoras.</td></tr>
<tr><td>Acreedora</td><td>Se salda del lado del haber ( abonos ), como<br>Suma de saldos de sus cuentas acreedoras – suma de los saldos de sus cuentas deudoras.</td></tr>
<tr><td rowspan="3">Cuentas</td><td rowspan="2">Atributos:<br>ID de la cuenta, nombre y naturaleza.<br>Contiene:<br>Cargos y Abonos.</td><td>Deudora</td><td>Se salda del lado del debe ( cargos ) como suma de montos de los cargos – suma de montos de los abonos.</td></tr>
<tr><td>Acreedora</td><td>Se salda del lado del haber ( abonos ) como suma de montos de los abonos – suma de montos de los cargos.</td></tr>
<tr><td colspan="3">Adicionalmente, se explica que la parte en que va cada cuenta está dada por el primer dígito ( el mas significativo del número de la cuenta. Este dígito es el identificador de la parte correspondiente.</td></tr>
<tr><td>Poliza Contable</td><td>Atributos:<br>NumeroDePoliza, descripción.<br>Contiene:<br>Movimientos contables de cargos y abonos</td><td></td><td>Para que sea válida debe:<br>1) Estar cuadrada: suma de montos de los cargos = suma de montos de los abonos.<br>2) Cada una de las cuantas referidas debe existir en la contabilidad, en</td></tr>
</table>

| | | | |
|---|---|---|---|
| | | | alguna de las partes. |
| Movimiento contable | Atributos: Naturaleza y monto | Cargo o abono. | |

La relación la vamos a representar con un diagrama de clases, en este diagrama, el rombo se refiere a "estar formado de..." o "tener a...". Así, clsContabilidad está formado de varias clsParteCont, éstas a su vez tienen a las cuentas y éstas a los movimientos contables. Igualmente la póliza contable tiene a su vez a los movimientos.




2.2. Construiremos primeramente la asociación de contabilidad hacia partes de contabilidad por medio de colección y con ello veremos los métodos que ofrecen las colecciones en VB6.

- Abra VisualBasic 6 y cree un nuevo proyecto standard.exe. Llámelo VbpConta00.vbp, lleve a cabo las modificaciones a las opciones para que

automáticamente se inserte el "Option Explicit" en cada módulo de código que se inserte en el proyecto, asimismo elimine la compilación bajo demanda y defina que la ejecución debe detenerse en todos los errores.

- Inserte un módulo de clase, llámelo clsParteCont, agréguele los atributos:

```
Option Explicit   ' este ya debe estar

Private intID As Integer
Private strNombre As String
Private strNat As String    ' naturaleza de la parte

Public Sub asigna(unId As Integer, unNombre As String, unaNat As String)
  intID = unId
  strNombre = unNombre
  strNat = unaNat
End Sub

Public Function strDespliega() As String
  strDespliega = CStr(intID) & " ... " & strNombre & " " & _
              IIf(strNat = "D", "Deudora", "Acreedora")
End Function
```

- Inserte asimismo el modulo de clase para la contabilidad, llámelo clsContabilidad, con el siguiente código, observe que la instanciación de las 5 partes de la contabilidad se lleva a cabo en el método initialize de la contabilidad, para que siempre ésta aparezca con las partes ya creadas.

```
Option Explicit   ' ya debe estar
Private colPartes As New Collection
Private strNombre As String
Private intEjercicio As Integer

Public Sub asigna(unNombre As String, unEjer As Integer)
  strNombre = unNombre
  intEjercicio = unEjer
End Sub

Public Sub balance()
' por lo pronto solamente despliega en la ventana de
' inmediato la lista de las partes.
  Dim x As clsParteCont
  Debug.Print strNombre & " Ejercicio " & intEjercicio
' recorramos la colección de partes, listando cada una de ellas
  For Each x In colPartes
```

```
    Debug.Print x.strDespliega()
  Next x
'
End Sub

Private Sub Class_Initialize()
  Dim x As clsParteCont

  Set x = New clsParteCont
  x.asigna 1, "Activo      ", "D"
  colPartes.Add x, "1"

  Set x = New clsParteCont
  x.asigna 2, "Pasivo      ", "A"
  colPartes.Add x, "2"

  Set x = New clsParteCont
  x.asigna 3, "Capital     ", "A"
  colPartes.Add x, "3"

  Set x = New clsParteCont
  x.asigna 4, "Ingresos    ", "A"
  colPartes.Add x, "4"

  Set x = New clsParteCont
  x.asigna 5, "Egresos     ", "D"
  colPartes.Add x, "5"

  Set x = Nothing

  End Sub
```

- Ahora vamos a insertar el "script" de prueba en un módulo de código puro. Para ello, insertemos dicho módulo a partir del menú Proyecto -> Agregar Módulo, pongámosle a este módulo el nombre moduloConta. Capturemos el script como a continuación se presenta, posteriormente le agragaremos más funcionalidad:

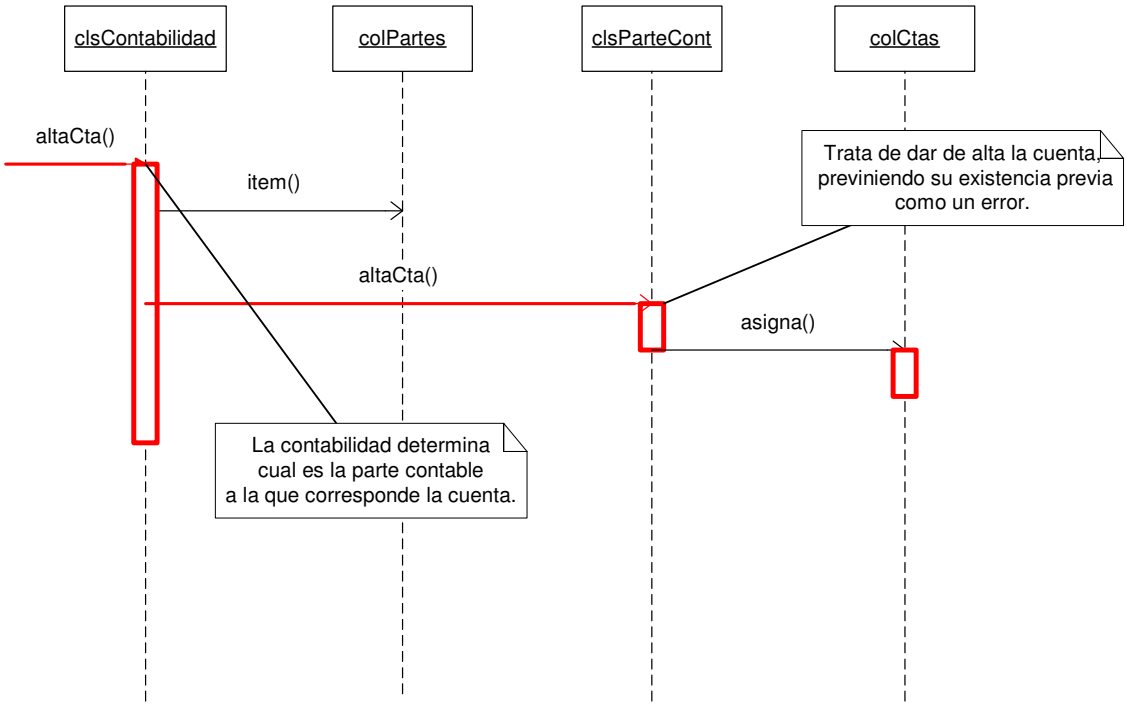
```
Sub main()
  Dim conta As New clsContabilidad
  Debug.Print " ============== " & Now & " ================="
  With conta
    .asigna "Empre S.A.", 2001
  End With
  conta.balance
End Sub
```

- Pongamos asimismo como objeto inical de la ejecución al procedimiento Sub Main. Para ello definámoslo como tal en el cuadro de diálogo de las Pripiedades del Proyecto: Proyecto->Propiedades del Proyecto, pestaña "General".

- Salve su proyecto. Para ejecutarlo habilite el despliegue de la ventana de inmediato. Con ello verá el resultado de la ejecución del "script". ¿ Porqué en esta etapa del desarrollo es conveniente trabajar con script y no con interfaz GUI?

2.3. Habilitando el alta de la cuenta.

Esto lo haremos primeramente con métodos inicialmente "dummies" y posteriormente los habilitaremos para que tengan la funcionalidad requerida. Pensemos inicialmente en el "***qué***", o lo que se requiere para posteriormente definir el "***cómo***" lo va a hacer para que de esa manera lo implantemos. Esta forma de trabajo será una constante a lo largo de nuestro trabajo. Ello nos permitirá encontrar de una manera sistemática la "huella" que tendrá nuestra aplicación desde el punto de vista del diseño. Para ello usaremos, además del diagrama de clases anteriormente mostrado, un diagrama de secuencias de aplicación de métodos entre los diferentes objetos que componen nuestro desarrollo. Así, para el método de altaCta de los objetos contabilidad ( método de instancia de la clase clsContabilidad ), el diagrama de secuencia puede ser el siguiente:




Este tipo de diagrama nos servirá para estructurar la secuencia de llamadas que debe llevarse a cabo para una tarea en particular. La idea básica es que cada objeto lleve a cabo la función que le corresponde a su ámbito para posteriormente solicitar a otro objeto que lleve a cabo otra(s) acciones para terminar con la tarea solicitada.

Aplicando este razonamiento al mensaje altaCta con los argumentos numeroDeCuenta, nombreDeLaCuenta y tipoDeCuenta que recibe contabilidad, podemos descomponerlo como sigue:

clsContabilidad recibe el mensaje con la solicitud de dar de alta una cuenta. El primer dígito del número de la cuenta codifica la parte de la contabilidad en que va la cuenta. Por ello debemos de obtenerlo, con ello sabremos cual es la parte a la que debemos solicitarle el alta de la cuenta. De existir, la contabilidad le solicita a la parte de contabilidad dar de lata la cuenta. Esta parte debe buscar, de entre su colección de cuentas si ya existe la que se desea dar de alta. En caso de que ya exista no debe darla de alta de nuevo. En caso de no existir entonces hay que crear el objeto y agregarlo a la colección de cuentas. Otra posibilidad consiste en aprovechar las características del lenguaje y de los elementos con los que se está implantando la combinación de estructura y funcionalidad requerida. En nuestro caso, las colecciones arrojan o levantan un error al tratar de dar de lata un objeto en una llave en que ya se encuentra otro. Podemos prever el error y atraparlo, decodificando su valor para estar seguros de que se debió precisamente a un alta en un índice ya ocupado. Haremos esto último. Para determinar el número del error

tenemos dos posibilidades: consultarlo en la ayuda del VB6 o bien provocarlo y examinarlo.

2.4. Definiendo el código para altaCta en clsContabilidad.

- Inserte el siguiente procedimiento en clsContabilidad:

```
Public Sub altaCta(numCta As Long, nombreCta As String,  _
                   tipoCta As String)
  Dim intParte As Integer
  Dim parte As clsParteCont
  intParte = numCta / 100000
  Set parte = colPartes.Item(CStr(intParte))
  Debug.Print "Cta:" & numCta & _
          " le corresponde la parte " & parte.strDespliega
  Set parte = Nothing
End Sub
```

- Salve el proyecto y pruebe el código
- Modifique el código del script para que trate de dar de alta una cuenta con un número de parte inválido, por ejemplo la cuenta 700000, ¿Qué ocurre?
- Prevenga el error en el procedimiento altaCta de clsContabilidad. No olvide fijar la opción de interrupción en error en
  Herramientas->Opciones, pestaña “General” a
  “Interrupción en errores no controlados”

2.5. Definiendo el método para altaCta en clsParteCont

Para ello, debemos programar el procedimiento hacia delante y tendremos que tomar una decisión importante de diseño: ¿Cómo vamos a notificar el hecho anormal de que ya exista una cuenta al tratar de darla de alta? ( Conviene aclarar que para nuestros fines las cuentas serán iguales si tienen el mismo número de cuenta ). Tenemos varias alternativas.

-Programar el método como una función con el valor de retorno indicando el resultado de la ejecución;
-Programarlo como un procedimiento con un argumento de retorno ( ByRef ) con el objetivo de entregar el valor de retorno con el resultado de la ejecución.
-Atrapar el error en algun nivel conveniente, quizá en el mismo clsContabilidad.
-Definir un evento y levantarlo al ocurrir el error.

En esta práctica tomaremos la última opción, así pues programemos el procedimiento hacia “adelante” y dejemos que VB levante el error cuando esto ocurra.

- Primeramente agregue el módulo de clase clsCtaCont para representar las cuentas, el código a continuación tiene los métodos requeridos.

```
Option Explicit ' ya debe estar
```

```
Private lngNumCta As Long
Private strNomCta As String
Private strTipoCta As String
Private colMovtos As New Collection

Public Sub asigna(numCta As Long, nombreCta As String, _
                  tipoCta As String)
  lngNumCta = numCta
  strNomCta = nombreCta
  strTipoCta = tipoCta
End Sub

Public Function strDespliega() As String
  strDespliega = "   " & CStr(lngNumCta) & " " & strNomCta & _
            IIf(strTipoCta = "A", "Acreedora", "Deudora")
End Function

Public Function saldo() As Currency
' por lo pronto un "Dummy"
  saldo = 0
End Function
```

- A continuación programemos la altaCta como un procedimiento en clsParteCont, declarando primeramente la colección para almacenar las cuentas en parte de contabilidad, en la parte general de la clase clsParteCont:

```
Private colCtas As New Collection
```

- Agregando las intrucciones requeridas en altaCta de clsParteCont

```
Public Sub despliegaCtas()
  Dim cta As clsCtaCont
   For Each cta In colCtas
  ' Esto no lo entrega, solo lo despliega
    Debug.Print cta.strDespliega
  Next cta
End Sub

Public Sub altaCta(numCta As Long, nombreCta As String, tipoCta As
String)
  Dim cta As New clsCtaCont
  cta.asigna numCta, nombreCta, tipoCta
  colCtas.Add cta, CStr(numCta)
  Set cta = Nothing
End Sub
```

- Ahora llamemos a este procedimiento desde clsContabilidad, en el método altaCta:

```
Public Sub altaCta(numCta As Long, nombreCta As String, _
                   tipoCta As String)
  Dim intParte As Integer
  Dim parte As clsParteCont
  On Error GoTo AltaCtaError
  intParte = numCta / 100000
  Set parte = colPartes.Item(CStr(intParte))
  Debug.Print "Cta:" & numCta & _
        " le corresponde la parte " & parte.strDespliega
  parte.altaCta numCta, nombreCta, tipoCta
  Set parte = Nothing
  Exit Sub
AltaCtaError:
  If Err.Number = 5 Then
    RaiseEvent CuentaInvalida(numCta)
  Else
    Debug.Print "Err.num = " & Err.Number & " descripcion: " &
Err.Description
  End If
End Sub
```

2.6. Detectando el error por llave duplicada

- Provoque y detecte el error debido a llave duplicada en la sección que atrapa los errores del método anterior. Prevenga su ocurrencia.

2.7. Modificando el método de balance para desplegar las cuentas de cada parte.

- Modifique el método de balance para que llame al método que despliega las cuentas de cada parte

2.8. Discuta con su instructor las alternativas para llevar a cabo el diseño y la implantación de la funcionalidad que resta.

TAREA: Termine la aplicación agregándole la funcionalidad impuesta por el script original

## 3. Construyendo una Biblioteca de Clases.

En esta práctica construiremos una biblioteca de clases en la modalidad de "Biblioteca de enlace dinámico" ( .dll). Haremos esto a partir de un proyecto que se le entregará para la contabilidad que se ha estado trabajando en la práctica 2. Usted partirá del código ya funcionando para las clases de contabilidad, parte de contabilidad, cuenta, póliza contable

y movimiento contable. Lo primero que hará es ejecutar la aplicación anterior y posteriormente creará un nuevo proyecto donde definirá la .dll para posteriormente agregar otro proyecto para probar la biblioteca. Esto lo hará agregando el script al proyecto de prueba, el script se ejecuta a partir del clic de un botón en una forma del proyecto de prueba.

3.1. Cargando y probando el proyecto base.

Se le proporcionará un directorio con los archivos resultado de la tarea de la práctica 2. Tal directorio tiene como nombre ***OriginalPrjSesion3VB6 .***

- Cópielo a un directorio de trabajo ( **PrjSesion3VB6** por ejemplo )
- Abra el proyecto ***prjConta00.vbp*** que se encuentra en él.
- Ejecútelo. La salida la manda a la ventana de inmediato. Trate de interpretar el script para saber qué es lo que hace e interprete la salida.

3.2. Construyendo el proyecto .dll y agregando los módulos de clase.

- En este apartado se construirá la biblioteca de clases para usarse con elnlace dinámico. Para ello debemos solicitar un nuevo proyecto de tipo .dll, agregarle los módulos correspondientes a las clases con las "Reglas de Negocio" y "generar" la .dll , solicite un nuevo proyecto en la modalidad "Nuevo" o "New", solicítelo de tipo .dll
- El proyecto tiene de manera automática un módulo de clase ( Class1 ) elimínelo con la opción Proyecto->Quitar Class1.cls
- Cámbiele el nombre al proyecto, defínalo como "ContaDll" y guárdelo en el mismo directorio PrjSesion3VB6.
- Agregue los módulos de clase al proyecto. Esto lo hará por medio de la opción "Proyecto->Agregar Módulo de Clase" pestaña "Existente" . Busque el directorio PrjSesion3VB6 y agregue los módulos
  - clsMovtoCont
  - clsPoliza
  - clsCtaCont
  - clsParteCont
  - clsContabilidad

- Modifique la propiedad de instanciación para los objetos de los clases clsContablidad y clsPoliza. Defina la propiedad de ambas como 5.- Multiuse
- Guarde el proyecto para la .dll ( ContaDll )
- Genere la .dll por medio de la opción "Archivo->Generar ContaDll.dll", guarde su .dll en el mismo directorio.

3.3. Agregando el proyecto de prueba y probando la biblioteca.

A continuación agregaremos el proyecto de prueba. Formaremos un grupo de proyectos con el proyecto de la .dll y el que ahora agregaremos.

- Por medio de "Archivo->Agregar Proyecto" Agregue un "standard.exe".
- Cámbiele el nombre a "ProytoPruebaDll"

- Estando sobre el proyecto en la ventana del explorador de proyectos asígnele al nuevo proyecto la propiedad de “Establecer como inicial”.
- Agregue un botón de comando a la forma ya existente.
- Agregue el módulo de código con el script ( moduloConta.bas )
- Invoque la rutina main a partir del clic del botón. Esto lo hace simplemente con la instrucción main dentro de la rutina del clic del botón:

```
Private Sub cmdBoton1_Click()
  main
End Sub
```

- Debemos “atar” o referenciar desde nuestro proyecto de prueba a la biblioteca de enlazado dnámico. Ello lo haremos accesando la facilidad ad-hoc que ofrece VB para estos fines. En “Proyecto->Referencias” debe aparecer la biblioteca como ContaDll, sin embargo quizá no tenga la asevereción de que se incluya en el proyecto. Póngale la aseveración a la cajita de selección.

- Salve su proyecto
- Pruebe su proyecto

Modifique el script para provocar errores en la operación del mismo y observe lo que ocurre.

TAREA: Se le solicita que proporcione la funcionalidad para llevar a cabo el cierre de la contabilidad. Lo que hace este proceso es definir la póliza de cierre e incidirla. La póliza de cierre debe tomar cada una de las cuentas de las partes 4 ( Ingresos ) y 5 ( Egresos ) que tengan saldo distinto de cero. Para cada una de ellas debe definir un movimiento contrario a su naturaleza para limpiar el saldo, agregando uno de la misma naturaleza a una cuenta de capital 300100 ,“Resultados del ejercicio anterior”,”Acreedora”. Observe que con este procedimiento, el efecto ya combinado de Ingresos y Egresos pasa a formar parte del capital, sin afectar el balance de la contabilidad.

## 4. Construyendo Controles Visuales.

En esta práctica le agregaremos la interfaz gráfica al usuario a nuestro proyecto de la contabilidad. La forma en que procederemos es en dos fases:

- Primero haremos la interfaz en base a formas independientes para cada una de la funcionalidad básica requerida, ( Alta de cuenta, Agregar Movimiento a póliza, incidir Póliza, Obtener Balance, etc. )

- Posteriormente para cada una de las formas independientes desarrollaremos un control visual con la misma funcionalidad.

En clase diseñaremos y desarrollaremos el control visual para el Alta de Cuenta, como tarea se quedan el diseño y desarrollo del resto de los controles.

4.1. Definiendo el grupo de proyectos.
- Construya o abra un grupo de proyectos similar al de la Práctica 3 "Construyendo una Biblioteca de Clases".
- Modifique las opciones del proyecto para que se detenga solamente en errores no controlados, que requiera la declaración explícita de todas las variables y que compile todo el proyecto al iniciar la ejecución.

4.2. Diseñando la secuencia de llamadas entre los objetos que conforman la interfaz visual general.

La interfaz visual que implantaremos tiene el siguiente aspecto:


El subgrupo de opciones de ***Contabilidad*** contiene las siguientes alternativas:

| | |
|---|---|
| Alta de Cuenta | Debe dar de lata una cuenta en la contabilidad. |
| Balance | Obtiene un reporte del Balance de la contabilidad. |
| Estado de resultados | Obtiene un reporte del estado de resultados. |

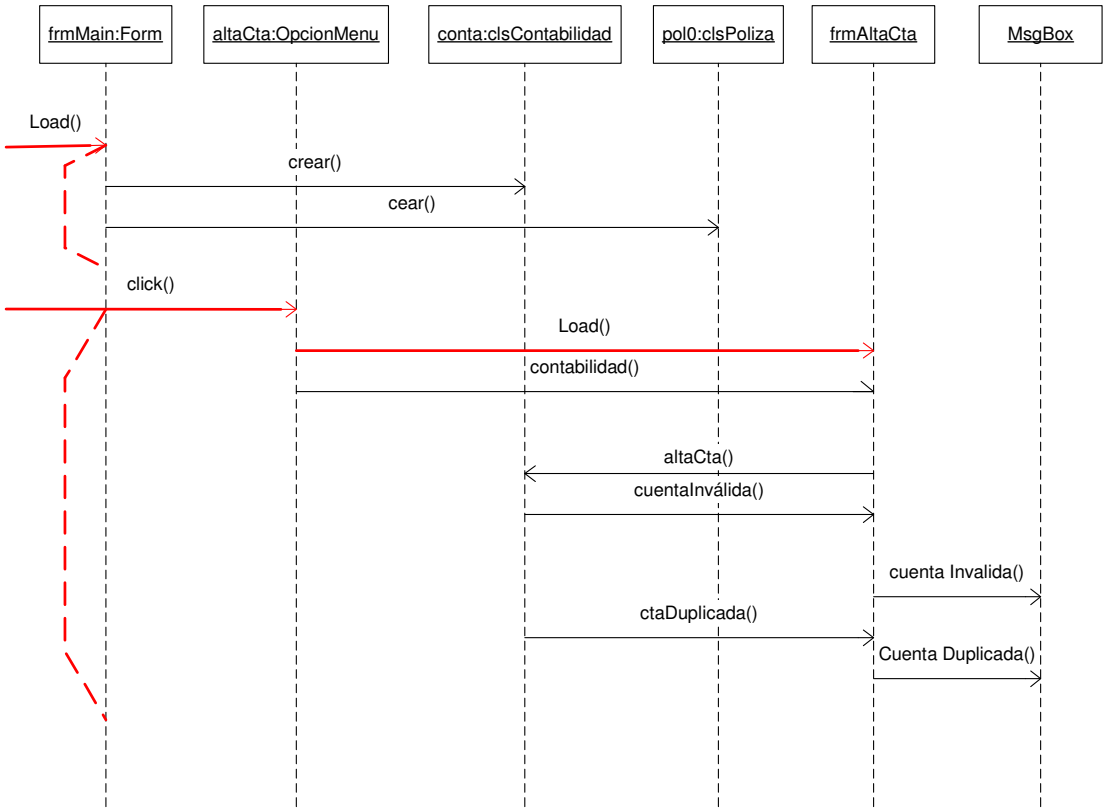
La opción de ***Póliza***:

| | |
|---|---|
| Nueva pólza | Limpia la póliza que se estuviese trabajando y permite capturar movimientos de cargo / abono. |
| Incidir | Incide la Póliza a la contabilidad. |

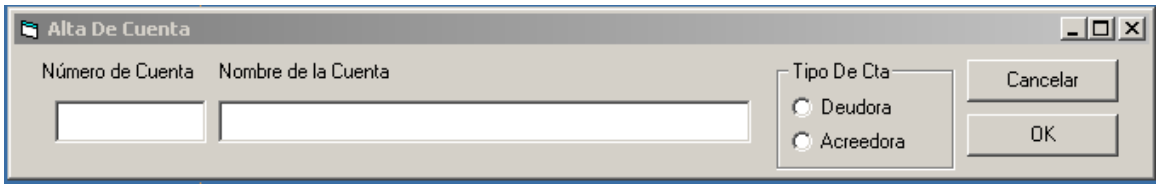
La opción de ***Salir*** termina la ejecución de la aplicación.

***Alta de Cuenta:***

La secuencia de mensajes que se utilizará es la siguiente:



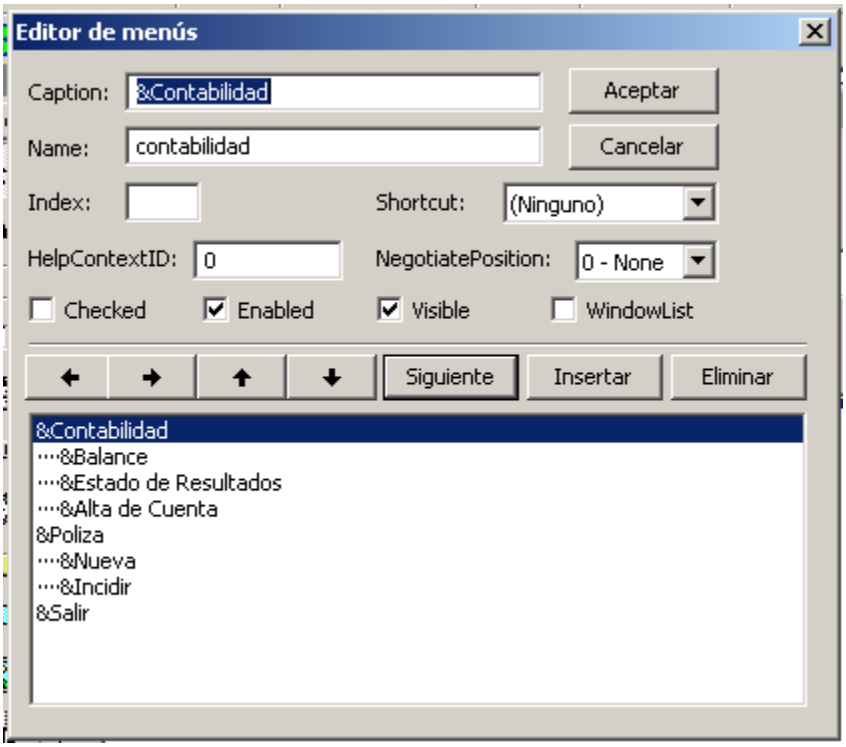

La forma que utilizaremos para el alta de la cuenta ( frmAltaCta ) tiene el siguiente layout:




El control de las situaciones de excepción debe ser “natural”.

4.3. Definiendo el menú:

- En la forma principal del proyecto ProytoPruebaDLL por medio de la alternativa que se accede con el botón derecho del ratón sobre la forma elegimos “Editor de menús” definimos las opciones de menú:




En la caja de texto "Name" de cada alternativa ponga el mismo letrero de la alternativa ( sin el & y sin los espacios de entre palabras ).

4.4. Definiendo la forma frmAltaCta.

- Agregue una forma al proyecto, nómbrela frmAlta
- Agregue los siguientes elementos gráficos con los siguientes nombres:

| | |
|---|---|
| lblNumCta | Caption = "Numero de Cuenta" |
| lblNomCta | Caption = "Nombre de la Cuenta" |
| txtNumCta | Text = "" |
| txtNombreCta | Text = "" |
| cmdBotonOK | Caption = "OK" |
| cmdBotonCancel | Caption = "Cancel" |
| frameTipoCta | Caption = "Tipo Cuenta" |
| optDeudora | Caption = "Deudora" ; Value = true |
| optAcreedora | Caption = "Acreedora" ; value = false |

- Salve su proyecto.

4.5. Agregando el código a la forma de Alta de Cuenta. Agregue el siguiente código a la forma:

```
Option Explicit
Dim WithEvents conta As contaDLL.clsContabilidad
```

```
Public Property Set contabilidad(unaConta As contaDLL.clsContabilidad)
  Set conta = unaConta
End Property
Private Sub cmdCancelar_Click()
  Unload Me
End Sub

Private Function blnEsNumCta(intNumDigitos As Integer, unaCadena As String) As
Boolean
  Dim resultado As Boolean
  Dim i As Integer
  resultado = True
  If Len(unaCadena) <> intNumDigitos Then
    resultado = False
  Else
   For i = 1 To intNumDigitos
     resultado = resultado And (Mid(unaCadena, i, 1) >= "0") _
                         And (Mid(unaCadena, i, 1) <= "9")
   Next i
  End If
  blnEsNumCta = resultado
End Function

Private Sub cmdOK_Click()
  Dim lngNumCta As Long
  If Not blnEsNumCta(6, txtNumCta) Or txtNombreCta = "" Then
    MsgBox "Numero de Cuenta o nombre de cuenta inválidos", vbCritical, "Alta de
Cuenta"
  Else
    MsgBox "Se procede a dar de alta la cuenta " & txtNumCta
    lngNumCta = CDbl(txtNumCta.Text)
    conta.altaCta lngNumCta, txtNombreCta.Text, IIf(optAcreedora, "A", "D")
    DoEvents
    Unload Me
  End If
End Sub

Private Sub conta_CuentaDuplicada(numCta As Long)
  MsgBox "Cuenta Duplicada " & txtNumCta.Text & "  " & txtNombreCta.Text
End Sub

Private Sub conta_CuentaInvalida(numDeCta As Long)
  MsgBox "Cuenta Inválida " & txtNumCta.Text & "  " & txtNombreCta.Text
End Sub
```

- Salve el proyecto

- Ejecute el proyecto para revisar la compilación de la forma frmAltaCta

4.6. Hilando la exhibición de la forma frmAltaCta a partir de la selección de la opción del menú.

- Agregue el siguiente código en la forma principal de su proyecto.

```
Option Explicit
Dim WithEvents conta As contaDLL.clsContabilidad
Dim pol0 As contaDLL.clsPoliza

Private Sub contabilidadAltaCta_Click()
  Dim x As New frmAltaCta
  Set x.contabilidad = conta
  x.Show vbModal
End Sub

Private Sub Form_Initialize()
  Set conta = New clsContabilidad
  Set pol0 = New clsPoliza
End Sub

Private Sub salir_Click()
  End
End Sub
```

- ¿Se puede declarar aquí la contabilidad sin los eventos?- justifique su respuesta.

- ¿Porqué se lanza frmAltaCta como modal?

- ¿Se puede lanar como no-modal? – justifique su respuesta.

- ¿Qué función tiene el DoEvents que aparece n la rutina del clic del botón OK?

4.7. Convirtiendo la forma frmAltaCta en un control .ocx. Lo que haremos a continuación es pasar la forma de lata de cuenta a un control visual de tipo ActiveX OCX. Para ello insertaremos otro proyecto al grupo de los que ya tenemos, por medio de “Copy & Paste” agregaremos tanto los controles que se

tienen en la frmCtaAlta como el código que los asiste. Modificaremos levemente el código para adecuarlo al control.

- Agregue un nuevo proyecto al grupo ya existente, del tipo Control ActiveX. Llámelo CtlAltaCta.
- Agregue, en caso de no tenerlo ya un "User Control", llámelo asimismo CtlAltaCta.
- Salve su proyecto.
- Copie el conjuto de controles de la frmAltaCta al user control.
- Copie el código de frmAltaCta a la página de código del user control.
- Modifique el código del user control para que quede conforme al siguiente:

```
Option Explicit

Dim WithEvents conta As contaDLL.clsContabilidad

Public Property Set contabilidad(unaConta As contaDLL.clsContabilidad)
  Set conta = unaConta
End Property

Private Sub cmdCancelar_Click()
  'Unload Me
End Sub
Private Function blnEsNumCta(intNumDigitos As Integer, unaCadena As String) As
Boolean
  Dim resultado As Boolean
  Dim i As Integer
  resultado = True
  If Len(unaCadena) <> intNumDigitos Then
    resultado = False
  Else
   For i = 1 To intNumDigitos
     resultado = resultado And (Mid(unaCadena, i, 1) >= "0") _
                        And (Mid(unaCadena, i, 1) <= "9")
   Next i
  End If
  blnEsNumCta = resultado
End Function

Private Sub cmdOK_Click()
  Dim lngNumCta As Long
  If Not blnEsNumCta(6, txtNumCta) Or txtNombreCta = "" Then
    MsgBox "Numero de Cuenta o nombre de cuenta inválidos", vbCritical, "Alta de
Cuenta"
  Else
    MsgBox "Se procede a dar de alta la cuenta " & txtNumCta
```

```
    lngNumCta = CDbl(txtNumCta.Text)
    conta.altaCta lngNumCta, txtNombreCta.Text, IIf(optAcreedora, "A", "D")
    DoEvents
    'Unload Me
  End If
End Sub

Private Sub conta_CuentaDuplicada(numCta As Long)
  MsgBox "Cuenta Duplicada " & txtNumCta.Text & "  " & txtNombreCta.Text
End Sub

Private Sub conta_CuentaInvalida(numDeCta As Long)
  MsgBox "Cuenta Inválida " & txtNumCta.Text & "  " & txtNombreCta.Text
End Sub
```

- Cierre tanto el user control ( parte visual ) como la página de código correspondiente ( de no ser así no podrá "operar" el control en su forma contenedor, es decir, en la forma principal del proyecto ).
- Cámbiese a la forma principal del proyecto de prueba. Observe que en la barra de herramientas visuales aparece el control que usted ha creado.
- Agregue el control a la forma principal. Deje el nombre que le asigna Visual Basic ( CtlAltaCta1 ).

- Modifique el código de la forma principal para que quede como:

```
Option Explicit
Dim WithEvents conta As contaDLL.clsContabilidad
Dim pol0 As contaDLL.clsPoliza

Private Sub contabilidadAltaCta_Click()
'  Dim x As New frmAltaCta
'  Set x.contabilidad = conta
'  x.Show vbModal
End Sub

Private Sub Form_Initialize()
  Set conta = New clsContabilidad
  Set pol0 = New clsPoliza
  Set CtlAltaCta1.contabilidad = conta
End Sub

Private Sub salir_Click()
  End
End Sub
```

- Salve su proyecto

- Ejecútelo y pruébelo

TAREA: Diseñe y construya los controles visuales para el resto de la funcionalidad requerida.