14. Programare orientată pe obiecte în Visual BASIC

14.1 Noţiuni generale de programare orientată pe obiecte

POO sau Programarea Orientată pe Obiecte are la bază noțiunea de obiect. Obiectul poate fi orice din lumea care ne înconjoară. El trebuie să fie tangibil și să interacționeze cu mediul în care există. De exemplu, un obiect ar putea fi ușa de la casă sau un proces cum este cel de testare a studenţilor, sau o relaţie ce caracterizează natura contactului între mai multe ființe sau ceva teoric, cum este un vector sau o matrice. Principial, orice subiect poate să fie un obiect.

14.2 Reprezentarea obiectelor

În vederea reprezentării, fiecare obiect primește un nume unic. El va avea un grup de atribute care îi definesc starea şi un grup de operaţii ce definesc acţiunile specifice obiectului în sine. De exemplu, atribute ar putea fi următoarele caracteristici: culoare, preţ, număr de locuri etc., iar operaţii ar putea fi: colorează, cumpără, pornește, oprește, încarcă etc. În faza de modelare a sistemului de programat este obişnuit ca obiectele să fie reprezentate grafic. O astfel de reprezentare apare în *Figura 74* pentru un obiect cu numele de Motor.

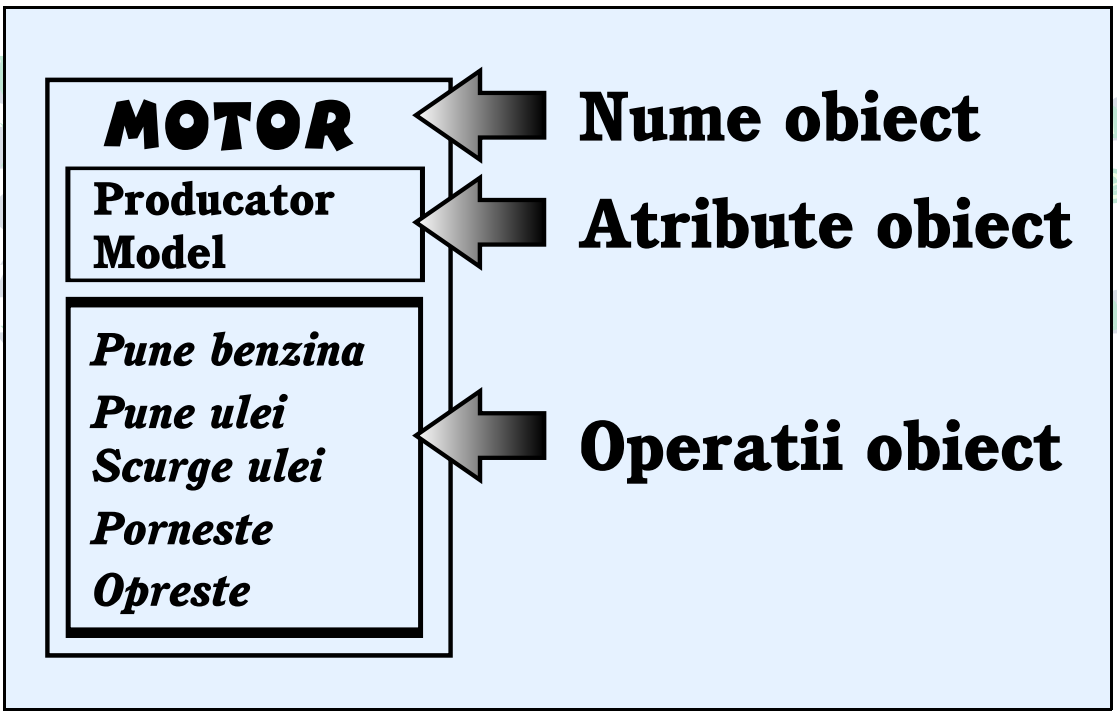

Figura 74 - Reprezentarea grafică a obiectului Motor.

Atributele sunt datele din obiect, acestea sunt manipulate prin operații. Atributele obiectului sunt singurele date pe care operațiile le vor accesa direct. În cazul unor implementări complexe se poate ca atributele să fie la rândul lor obiecte. Operațiile reprezintă acțiunile ce se pot realiza cu obiectul dat și sunt deseori numite și metode. Ele reprezintă locul în care au loc prelucrările și pot manipula direct atributele obiectului. Din punctul de vedere al programatorului VB ele sunt echivalentul procedurilor. În teoria generală a POO o colecție de obiecte similare se numește clasă. Toate obiectele clasei au aceleași atribute și metode, dar valorile atributelor diferă. Un obiect particular poartă denumirea de instanțiere a clasei. Din punctul de vedere al programatorului VB clasele permit definirea a noi tipuri de date.

14.2.1 Moştenirea

Clasele pot fi aranjate sub forma unei structuri ierarhice de moștenire. O clasă moștenește atributele și metodele părintelui. Acest proces stă la baza mecanismului de scriere a codului reutilizabil. În *Figura 75* se reprezintă grafic ierarhia de moștenirii pentru trei tipuri de motoare.

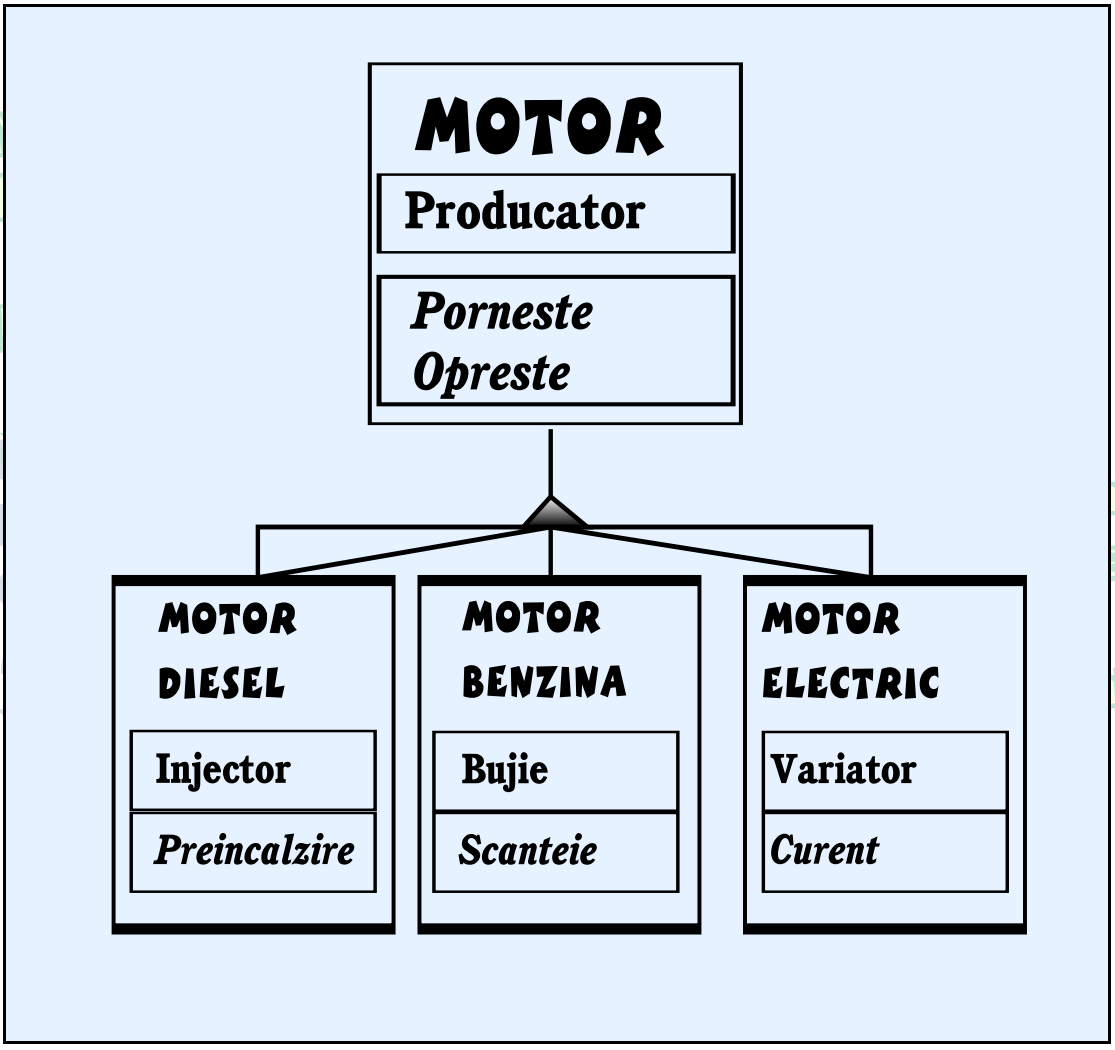

Figura 75 - Reprezentarea unei ierarhii de moșteniri.

Fiecare tip de motor moștenește caracteristicile de bază comune tuturor motoarelor, dar, dincolo de acestea, mai are atribute și operații specifice.

14.2.2 Încapsularea

După creare un obiect poate fi modificat doar cu ajutorul operațiilor definite la nivelul lui. În acest fel operațiile şi atributele sunt încapsulate într-o singură definiție, controlul modificării atributelor fiind mult simplificat. Apare în plus şi posibilitatea modificării reprezentării interne a obiectului cu păstrarea comportamentului extern. Marele avantaj al încapsulării, de care beneficiază şi obiectele VB, constă în accesul controlat la date. Din acest motiv nu se pot efectua transformări nedorite sau ilegale ale acestora.

14.2.3 Polimorfismul

Polimorfismul este proprietatea prin care operațiile cu acelaşi nume pot fi implementate în mai multe locuri. Singurele operații ale unui obiect sunt cele definite la nivelul lui sau cele moştenite de la un părinte sau de la un părinte al părintelui ş. a. m. d. Ca urmare a polimorfismului nu trebuie inventate noi nume pentru aceleaşi operații logice ce se fac însă cu date diferite. În cazul sistemelor orientate pe obiecte polimorfismul permite extinderea acestora împreună cu păstrarea naturală a expresiilor folosind scrierea infixată în locul apelurilor de procedură.

14.2.4 Principii de modelare cu obiecte

Modelarea prin obiecte s-a răspândit deoarece ea are o corespondență directă cu lumea reală. După determinare, modelul are o structură statică (foarte rar mai suferă modificări). Reprezentarea grafică a modelului este intuitivă, motiv pentru care el poate fi înțeles uşor de către persoane fără o pregătire specială. Un model de obiecte este definit prin:

! obiecte;
! atribute;
! operații;
! relații sau asocieri ce rezultă din moştenire sau agregare.

Există cazuri când un obiect este format din alte obiecte, acest proces de construire a unui obiect pe baza altor obiecte se numeşte agregare. Deşi şi în cazul agregării există ierarhie nu există moştenire. Componentele unui obiect agregat sunt stocate în atribute la momentul implementării, iar operațiile cu obiectele agregate pot fi descompuse deseori în operații la nivelul componentelor (sau măcar a unor componente).

Nu este obiectivul cărții prezentarea detaliată a acestor concepte, este însă nevoie de cunoaşterea sumară a termenilor deoarece noțiunea de model de obiecte va fi utilizat mult în capitolul de ActivaX Automation (o tehnologie software ce permite programarea unei aplicații server din una client pe baza modelului de obiecte expus de server).

14.3 Clase în Visual BASIC

În VB clasele sunt adăugate unui proicet realizând clic cu butonul drept al mouseului pe **Project Name** din **Project Explorer** de unde se va selecta **Add** şi apoi **Class Module**. Acelaşi lucru se obține şi prin selectarea lui **Add Class Module** din meniul **Project**. Spre deosebire de module este încetățenit ca numele modulelor de clasă să înceapă cu litera `C`. De exemplu, un astfel de nume ar putea fi **`CFTextIE`** ce va fi atribuit proprietății `Name` a clasei.

Pentru implementarea atributelor clasei, numite şi proprietăți, acestea vor fi precedate de cuvântul cheie **`Private`** urmate de un nume şi un tip de dată. Din nou, se obişnuieşte ca numele unui atribut să fie precedat de litera `m` având semnificația de membru de clasă.

Ca urmare a declaraţiei **Private** membri clasei nu vor putea fi accesaţi direct. Avantajul constă în creşterea siguranţei în vederea introducerii unor date valide. Operaţiile din cadrul clasei se implementează prin proceduri de tipul **Function** sau **Sub**.

14.3.1 Definirea membrilor dată

Când se atribuie o valoare unei proprietăţi din obiect, valoarea este stocată într-o variabilă internă care este accesibilă numai respectivului obiect. Un utilizator al obiectului poate manipula doar proprietăţile expuse. Astfel, datele specifice obiectului sunt protejate contra manipulărilor directe şi, eventual, greşite.

Primul pas este specificarea acestor variabile interne clasei care se mai numesc şi membri dată (atributele). De obicei, se declară câte o variabilă internă pentru fiecare dată de stocat în obiect respectând regulile:

- fiecare variabilă se declară folosind cuvântul cheie **Private** pentru a face variabilele disponibile numai în interiorul acelei clase;
- tradiţional, numele membrilor dată se prefixează cu **m** pentru a indica faptul că este vorba despre un membru al unei clase.

Exemple de membri dată sunt:

```
Private mcale As String
Private mtext As String
```


14.3.2 Definirea proprietăţilor clasei

Membri unei clase pot fi accesaţi direct dacă sunt **Public** şi nu pot fi accesaţi direct, ci doar prin intermediul instrucţiunilor **Property Get**, **Let** sau **Set**, dacă sunt **Private**. Din motive de consistenţă este permisă folosirea aceluiaşi nume de proprietate la înscrierea (**Let**) şi la citirea (**Get**) unui atribut.

Proprietatea Get

Property Get se foloseşte pentru întoarcerea valorii curente a unei proprietăţi. Ea este echivalentul unei funcţii care întoarce o valoare. Formatul general pentru această proprietate este:

```
Public Property Get numeproprietate( )
      [instructiuni]
      numeproprietate = mmembrudată
End Property
```

Folosirea lui **Public** face ca această procedură sau proprietate să fie vizibilă pentru procese externe. **numeproprietate** este numele proprietăţii, iar *mmembrudată* este variabila membru dată care stochează valoarea curentă a proprietăţii. Dacă dorim, se pot adăuga instrucţiuni în plus procedurii, deşi în cazul lui **Property Get** este destul de rar. Iată două exemple în continuare:

```
Property Get text() As String
    text = mtext
End Property
```

```
Property Get cale() As String
    cale = mcale
End Property
```

Proprietatea Let

Cu excepţia cazului în care doriţi ca proprietatea să poată fi doar citită, trebuie definită şi o procedură care permite atribuirea unei noi valori proprietăţii. **Property Let** se foloseşte pentru atribuirea unei valori unei proprietăţi (atribut) din clasă. Forma ei generală este:

```
Public Property Let numeproprietate(ValoareNouă)
    [instructiuni]
    mmembrudată = ValoareNouă
End Property
```

La fel ca şi mai înainte **numeproprietate** este numele proprietăţii, de asemenea, ***ValoareNou***ă este noua valoare pe care utilizatorul vrea s-o atribuie proprietăţii, iar *mmembrudat*ă este variabila membru dată a cărei valoare se va modifica. Şi aici pot fi adăugate instrucţiuni în plus. Acestea testează, de obicei, noua valoare înainte de a modifica membrul dată. Iată două exemple:


```
Property Let text(ByVal sirText As String)
    mtext = sirText
End Property

Property Let cale(ByVal sirCale As String)
    mcale = sirCale
End Property
```

Proprietatea Set

Property Let permite modificarea unui atribut care are un tip de dată simplu. Sunt cazuri când o proprietate este un obiect. După cum ştiţi deja obiectele nu pot primi direct o valoarea prin atribuire. În cazul unei referinţe la un obiect este obligatorie folosirea lui **Set**. **Property Set** se foloseşte în situaţia în care proprietatea de modificat este un obiect. Forma ei generală este:

```
Public Property Set numeproprietate(ValoareObiectNouă)
    [instructiuni]
    Set mmembrudatăobiect = ValoareObiectNouă
End Property
```

14.3.3 Definirea metodelor clasei

Orice obiect are şi un grup de metode care acţionează asupra obiectului. În acest scop, în cadrul clasei, trebuie să definim proceduri **Sub** sau **Function** de tipul **Public** pentru ca să fie vizibile extern.

14.3.4 Definirea evenimentelor clasei

Clasele definite de utilizator suportă lucrul cu evenimentele **Initialize** şi **Terminate**.

Evenimentul Initialize

Acesta apare atunci când se realizează instanţierea obiectului de tipul clasă, adică atunci când se declară o variabilă obiect de tipul clasei după cum urmează:

```
Dim objFile As CFTextIE

'Crearea unei noi instante a clasei CFTextIE
Set objFile = New CFTextIE
```

Pentru a folosi acest eveniment se adaugă clasei o procedură **Private** care foloseşte următorul format:

```
Private Sub Class_Initialize
     [codul de initializare se pune aici]
End Sub
```

Iată un exemplu:


```
Private Sub Class_Initialize()
    'Seteaza modul implicit al
    ' proprietatii de deschidere la scriere
    EvitaSuprascriere = True
    mesteDeschis = False
End Sub
```


Evenimentul Terminate

Acest eveniment apare atunci când toate referinţele la obiectul de tipul clasă sunt setate la **Nothing**. Se foloseşte atunci când clasa a alocat dinamic memorie şi aceasta trebuie eliberată. Pentru a folosi acest eveniment se scrie, în clasă, o procedură **Private** de forma:

```
Private Sub Class_Terminate
   [codul pentru terminare se pune aici]
End Sub
```

Iată un exemplu:

```
Private Sub Class_Terminate()
    'Daca fisierul este inca deschis se inchide
    ' inainte de terminare
    If mesteDeschis Then
        InchideFisier
    End If

    Set mActualizareCurenta = Nothing
    Set mUltimaActualizare = Nothing
End Sub
```

14.4 Crearea unei clase pentru manipularea fişierelor text

Clasa care urmează a fi creată pleacă de la următorul modul de cod:

```
Sub testIO()
    Dim cale As String
    Dim text As String
    Dim handle As Long

    cale = "C:\a\TEST1.TXT"

    'Inscriere in fisierul din cale
    handle = FreeFile
    Open cale For Append Access Write As handle
    Print #handle, "lina unu"
    Print #handle, "lina doi"
    Print #handle, "lina trei"
    Close #handle

    'Citirea liniilor din fisier
    handle = FreeFile
    Open cale For Input Access Read As handle
    Do Until EOF(handle)
        Line Input #handle, text
        Debug.Print text
    Loop
    Close #handle
End Sub
```


Manipularea unui fişier, în VB, se face cu ajutorul unui număr întreg ce identifică unic fişierul în cauză. Prima valoare numerică liberă ce ar putea fi folosită în acest scop este întoarsă de funcţia **`FreeFile`**. Pregătirea fişierului în vederea scrierii sau citirii se face prin deschiderea lui. În general, deschiderea presupune cel puţin verificarea existenţei fişierului şi a drepturilor de acces pe care le are utilizatorul cu privirea la fişier. Rezultatul deschiderii cu succes se manifestă prin posibilităţile de acces şi de identificare a fişierului de manipulat. În VB, funcţia **`Open`** este folosită pentru a deschide accesul la fişier şi pentru specificarea modului de acces la datele lui. Pentru scrierea într-un fişier secvenţa de parametrii a lui **`Open`** sunt **`cale For Append Access Write As handle`**. Din punctul de vedere al VB un fişier reprezintă un grup de date stocate pe disc sub un nume. Atunci când o aplicaţie accesează un fişier trebuie să facă presupuneri cu privire la organizarea datelor din fişier. **`cale`** este un şir ce specifică poziţia fişierului, mai sus valoarea lui este "**`C:\a\TEST1.TXT`**". Adică fişierul este pe discul **`C:`**, iar din rădăcina lui trebuie să existe un director cu numele **`a`** care conţine fişierul cu numele **`TEST1.TXT`**. Presupunând că fişierul va conţine doar text ASCII se poate folosi modul de lucru numit "secvenţial" (**`Append`**), iar accesul la acesta este cel pentru scriere (**`Write`**). În cazul în care fişierul nu există, el va fi creat automat. Dacă acesta există toate liniile noi vor fi adăugate la sfârşitul lui. Pentru adăugarea unei linii la fişier se foloseşte funcţia **`Print`**, iar terminarea lucrului cu fişierul se marchează prin închiderea lui folosind funcţia **`Close`**. Atunci când se

dorește citirea datelor din fișierul secvențial se folosește linia **`Open cale For Input Access Read As handle`**, iar citirea unei linii de text se face cu **`Line Input #handle, text`**. Textul unei linii se citește în variabila **`String`** cu numele **`text`** după care se afișează în fereastra imediată. Funcția **`EOF()`** întoarce valoarea booleană **`True`** atunci când s-a ajuns la sfârșitul de fișier.

Clasele care vor participa la aplicație sunt definite în continuare, iar rezultatele aplicației apar în *Figura 76*.

14.4.1 Clasa CFTextIE

Clasa trebuie să aibă proprietatea `Name` setată la valoarea **`CFTextIE`**. Ea conține la rândul ei alte clase. Acest mod de lucru poartă denumirea de **agregare** sau compunere promovând reutilizarea codului, una dintre cele mai apreciate facilități ale POO. Agregarea se face la nivelul proprietăților **`mActualizareCurenta`**, **`mUltimaActualizare`** în care se vor stoca obiecte din clasa **`CInfoF`**.

```
Option Explicit
Private mcale As String
Private mtext As String
Private mhandle As Long

Public EvitaSuprascriere As Boolean
Private mesteDeschis As Boolean
Private mModDeschidere As ModDeschidereFText
Private mActualizareCurenta As New CInfoF
Private mUltimaActualizare As New CInfoF

Public Enum ModDeschidereFText
    ftCitire
    ftScriere
End Enum

Property Get UltimaActualizare() As CInfoF
    Set UltimaActualizare = mUltimaActualizare
End Property

Property Get ActualizareCurenta() As CInfoF
    Set ActualizareCurenta = mActualizareCurenta
End Property

Property Let text(ByVal sirText As String)
    mtext = sirText
End Property

Property Get text() As String
    text = mtext
End Property
```

```
Property Let cale(ByVal sirCale As String)
    mcale = sirCale
End Property

Property Get cale() As String
    cale = mcale
End Property

Property Get EsteSfarsitdeFisier() As Boolean
    If mesteDeschis Then
        EsteSfarsitdeFisier = EOF(mhandle)
    End If
End Property

Property Let ModDeschidere(ModuldeDeschiderealFisieruli As _
                                              ModDeschidereFText)
    mModDeschidere = ModuldeDeschiderealFisieruli
End Property

Public Function DeschideFisier()
    If mesteDeschis Then
        InchideFisier
    End If
    mhandle = FreeFile
    Select Case mModDeschidere
        Case ftScriere
            If EvitaSuprascriere Then
                If ExistaFisier Then
                    MsgBox "Eroare, fisierul " & cale & _
                    " exista deja " & vbCrLf & _
                    "pentru a suprascrie setati" & _
                    " proprietatea" & vbCrLf & _
                    "EvitaSuprascriere la False"
                    Exit Function
                End If
            End If
            mUltimaActualizare.mcale = mcale
            mUltimaActualizare.ActualizareInfoFisier
            Open mcale For Append Access Write As mhandle
        Case ftCitire
            mUltimaActualizare.mcale = mcale
            mUltimaActualizare.ActualizareInfoFisier
            Open mcale For Input Access Read As mhandle
        Case Else
            Err.Raise 5, "Deschidere de fisier text",  _
                    "Nu exista modul specificat la deschidere"
    End Select
```

```
    mActualizareCurenta.mcale = mcale
    mesteDeschis = True
End Function

Public Sub InstrieOLinie()
    If mesteDeschis Then
        Print #mhandle, mtext
        mActualizareCurenta.ActualizareInfoFisier
    End If
End Sub

Public Sub CitesteOLinie()
    If mesteDeschis Then
        If Not EsteSfarsitdeFisier Then
            Line Input #mhandle, mtext
        End If
    End If
End Sub

Public Function ExistaFisier() As Boolean
    'Testeaza existenta unui fisier
    ' intr-un director pe baza lui mcale

    On Error Resume Next
    ExistaFisier = (Len(Dir$(mcale)) > 0)
    If Err.Number <> 0 Then
        ExistaFisier = False
    End If
End Function

Public Sub InchideFisier()
    Close mhandle
    mesteDeschis = False
    mActualizareCurenta.ActualizareInfoFisier
End Sub

Private Sub Class_Initialize()
    'Seteaza modul implicit al
    ' proprietatii de deschidere la scriere
    EvitaSuprascriere = True
    mesteDeschis = False
End Sub

Private Sub Class_Terminate()
    'Daca fisierul este inca deschis se inchide
    ' inainte de terminare
    If mesteDeschis Then
        InchideFisier
    End If
```

```
    Set mActualizareCurenta = Nothing
    Set mUltimaActualizare = Nothing
End Sub
```

14.4.2 Clasa CInfoF

Clasa trebuie să aibă proprietatea `Name` setată la valoarea **CInfoF**.

```
Option Explicit
Private mLungime As Long
Private mDataOraMod As Date
Public mcale As String

Public Sub ActualizareInfoFisier()
    'Lungimea fiserului in octeti
    mLungime = FileLen(mcale)
    'Ora si data ultimei modificari ale fisierului
    mDataOraMod = FileDateTime(mcale)
End Sub

Property Get LungimeFisier() As Long
  LungimeFisier = mLungime
End Property

Property Get OraDataModificare() As Date
  OraDataModificare = mDataOraMod
End Property
```


Modulul de cod din care se utilizează clasele este prezentat în continuare. Instanţierea obiectului **CFTextIE** se face în linia **Set obFTextIO = New CFTextIE**. O altă metodă de instanţiere este folosirea directă a lui **New** în declaraţia variabilei obiect (**Dim obFTextIO As New CFTextIE**) caz în care valoarea referinţei la obiect este atribuită direct variabilei **obFTextIO** fără a mai fi nevoie de **Set**.

```
Sub Main()
   Dim obFTextIO As CFTextIE

    'Crearea unei noi instante a clasei CFTextIE
    Set obFTextIO = New CFTextIE

    'Setarea proprietatii cale
    obFTextIO.cale = "C:\a\TEST3.TXT"

    'Se deschide fisierul,
    ' in modul de suprascriere,
    ' si pentru scriere de linii in el
    obFTextIO.ModDeschidere = ftScriere
    obFTextIO.EvitaSuprascriere = False
    obFTextIO.DeschideFisier
```

```
'Se inscriu trei linii
obFTextIO.text = "linia 1"
obFTextIO.InstrieOLinie
obFTextIO.text = "linia 2"
obFTextIO.InstrieOLinie
obFTextIO.text = "linia 3"
obFTextIO.InstrieOLinie
'Se inchide fisierul
obFTextIO.InchideFisier

Debug.Print "La ultima actualizare a fisierului avem"
Debug.Print obFTextIO.UltimaActualizare.LungimeFisier; _
            obFTextIO.UltimaActualizare.OraDataModificare
Debug.Print "Dupa actualizarea curenta avem"
Debug.Print obFTextIO.ActualizareCurenta.LungimeFisier; _
           obFTextIO.ActualizareCurenta.OraDataModificare

'Se deschide fisierul,
' in modul de citire
```


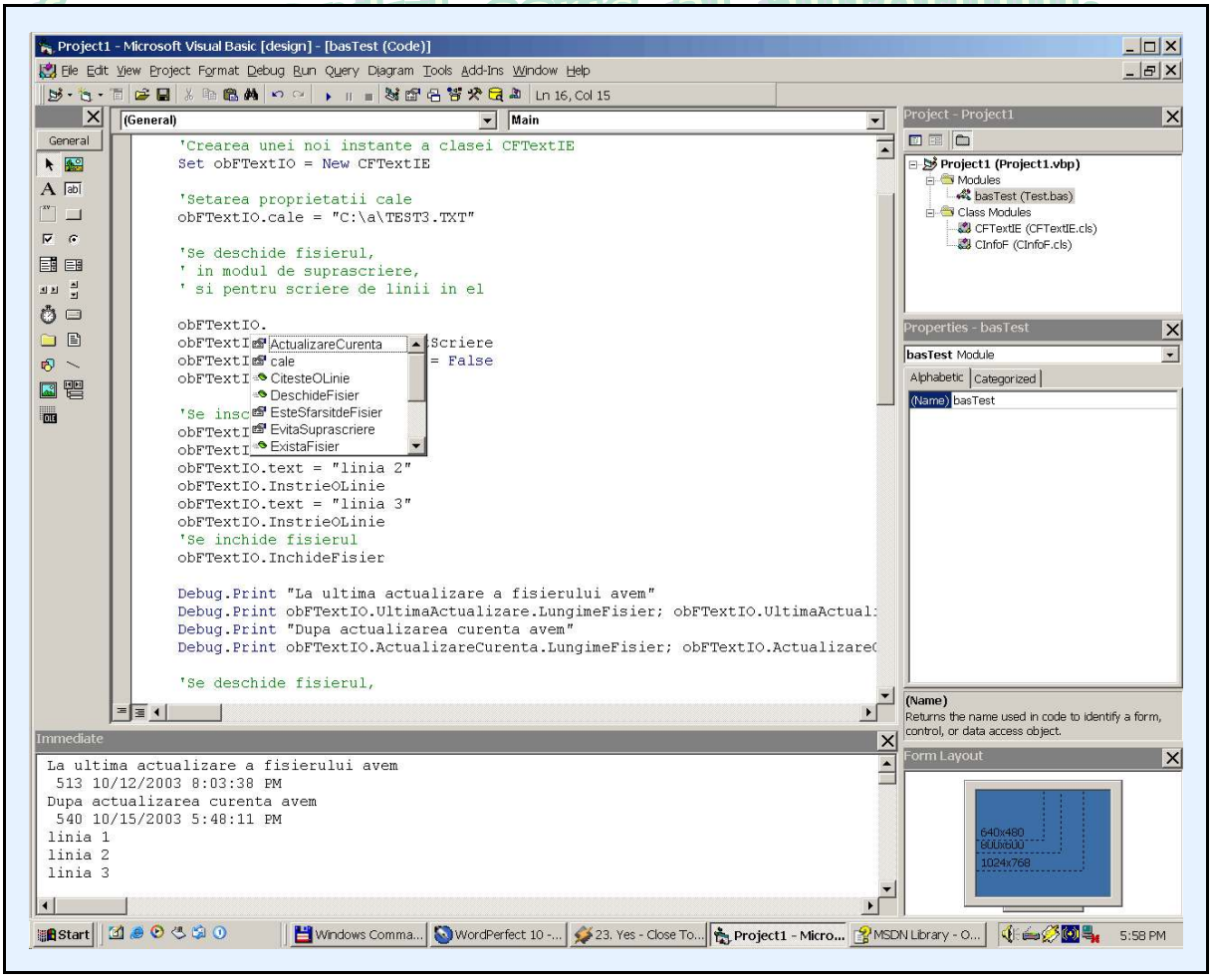

Figura 76 - Afişarea proprietăţilor şi a membrilor unei clase definite de utilizator în timpul utilizării ei.

```
    obFTextIO.ModDeschidere = ftCitire
    obFTextIO.DeschideFisier

    'Se citeste cate o linie
    ' pana se ajunge la sfarsitul fisierului
    Do While Not obFTextIO.EsteSfarsitdeFisier
        obFTextIO.CitesteOLinie
        Debug.Print obFTextIO.text
    Loop

    'Se inchide fisierul
    obFTextIO.InchideFisier

    'Se distruge instanta clasei
    Set obFTextIO = Nothing
End Sub
```

14.5 Implementarea unei clase cu moştenire de interfaţă

În general, **moştenirea** se poate defini la nivel de implementrare sau de interfaţă. Moştenirea de implementare asigură specializarea unor secvenţe de cod comune dintr-o clasă părinte prin dezvoltarea de cod particularizat la nivelul subclaselor. Moştenirea de interfaţă permite reutilizarea unei interfeţe, adică ea apare la nivelul proprietăţilor şi metodelor ce vor putea fi redefinite. Aceasta asigură păstrarea unei interfeţe atunci când codul din spatele ei se modifică. În VB moştenirea de interfaţă duce şi la polimorfism deoarece obligă la reutilizarea aceloraşi nume pentru proprietăţile şi metodele interfeţei. Pentru exemplul care urmează se vor crea două clase de primitive grafice una numită **`CCerc`** şi alta **`CDreptunghi`**. Proprietăţile cercului constau în coordonatele centrului şi rază, iar cele ale dreptunghiului în coordonatele colţului stânga-sus, lungime şi lăţime. La nivelul atributelor ce caracterizează cele două clase nu se găsesc elemente care, conceptual, să fie comune. Ambele obiecte sunt însă caracterizate prin arie şi posibilitatea desenării pe ecran. Aceste două elemente comune se folosesc pentru a crea o clasă de bază numită superclasă care va grupa metodele comune. În general, codul acestor metode este vid, iar numele lor este folosit în procesul de moştenire de interfaţă. Tradiţional, numele acestei clase generice de interfaţă începe cu litera **`I`** (de exemplu **`IFormaGeometrica`**). Implementarea următoare va calcula aria şi va realiza reprezentarea grafică a unor forme geometrice. Este clar că metoda folosită la desenarea cercului e distinctă de cea folosită la desenarea dreptunghiului, la fel se pune problema şi la calculul ariei celor două figuri geometrice. Polimorfismul apare ca urmare a acestor asemănări conceptuale care necesită implementări distincte.

Clasa de interfaţă se numeste **`IFormaGeometrica`**, apoi se mai definesc clasele **`CCerc`** pentru cerc şi **`CDreptunghi`** pentru dreptunghi. Pentru a specifica faptul că acestea implementează clase de interfaţă instrucţiunea **`Implements IFormaGeometrica`** este adăugată la începutul ambelor clase. Din acest moment este obligatoriu ca ele să redefinească metodele abstracte **`Aria()`** şi **`Deseneaza()`** cu metodele **`IFormaGeometrica_Aria()`**, **`IFormaGeometrica_Deseneaza()`**.

14.5.1 Clasa IFormaGeometrica

Clasa trebuie să aibă proprietatea `Name` setată la valoarea **`IFormaGeometrica`**.

```
Option Explicit
Public Function Aria() As Double
   'nu este cod
End Function

Public Sub Deseneaza()
   'nu este cod
End Sub
```

14.5.2 Clasa CCerc

Clasa trebuie să aibă proprietatea `Name` setată la valoarea **`CCerc`**. Ea mai trebuie obligatoriu să definească metodele **`Aria()`** şi **`Deseneaza()`**.

```
Option Explicit
Implements IFormaGeometrica
Private mX As Integer
Private mY As Integer
Private mRaza As Integer
Private mPicture As Object
' Specificarea definitiilor de interfata
Public Function IFormaGeometrica_Aria() As Double
    IFormaGeometrica_Aria = 3.14159 * mRaza ^ 2
End Function

Public Sub IFormaGeometrica_Deseneaza()
   mPicture.Circle (mX, mY), mRaza
End Sub

' Adaugarea proprietatilor clasei
Public Property Set PictureBox(ByVal pic As Object)
   Set mPicture = pic
End Property
Public Property Get X() As Integer
    X = mX
End Property
Public Property Let X(ByVal xVal As Integer)
    mX = xVal
End Property
Public Property Get Y() As Integer
    Y = mY
End Property
Public Property Let Y(ByVal yVal As Integer)
    mY = yVal
End Property
```

```
Public Property Get Raza() As Integer
    Raza = mRaza
End Property
Public Property Let Raza(ByVal r As Integer)
    mRaza = r
End Property
```

14.5.3 Clasa CDreptunghi

Clasa trebuie să aibă proprietatea `Name` setată la valoarea **`CDreptunghi`**. Ea mai trebuie obligatoriu să definească metodele **`Aria()`** şi **`Deseneaza()`**.

```
Option Explicit
Implements IFormaGeometrica
Private mX As Integer
Private mY As Integer
Private mLatime As Integer
Private mInaltime As Integer
Private mPicture As Object
' Definitiile de interfata
Public Function IFormaGeometrica_Aria() As Double
  IFormaGeometrica_Aria = CDbl(mLatime) * mInaltime
End Function

Public Sub IFormaGeometrica_Deseneaza()
  mPicture.Line (mX, mY)-Step(mLatime, mInaltime), , B
End Sub

' Adaugarea proprietatilor clasei
Public Property Set PictureBox(ByRef pic As Object)
  Set mPicture = pic
End Property

Public Property Get X() As Integer
  X = mX
End Property
Public Property Let X(ByVal xVal As Integer)
    mX = xVal
End Property
Public Property Get Y() As Integer
    Y = mY
End Property
Public Property Let Y(ByVal yVal As Integer)
    mY = yVal
End Property
Public Property Get Latime() As Integer
    Latime = mLatime
End Property
Public Property Let Latime(ByVal w As Integer)
    mLatime = w
```

```
End Property
Public Property Get Inaltime() As Integer
    Inaltime = mInaltime
End Property
Public Property Let Inaltime(ByVal h As Integer)
    mInaltime = h
End Property
```

14.5.4 Formularul de utilizare al claselor cu moştenire de interfaţă

Formularul din *Figura 77* are un singur control **PictureBox** care are proprietatea `Name` setată la valoarea **`pic1`**, iar codul corespunzător încărcării formularului este:

```
Private Sub Form_Load()
    Dim c As New CCerc
    Dim d As New CDreptunghi
    Dim referintaInterfata As IFormaGeometrica

    pic1.AutoRedraw = True
    Set referintaInterfata = c
    c.Raza = pic1.ScaleWidth / 4
    c.X = pic1.ScaleWidth / 2
    c.Y = pic1.ScaleHeight / 2
    Set c.PictureBox = Me.pic1
    referintaInterfata.Deseneaza
    Debug.Print "X=" & c.X; " Y=" & c.Y; ", R="; c.Raza
    Debug.Print "Aria cercului este:" & referintaInterfata.Aria
    Set referintaInterfata = d
    d.Inaltime = pic1.ScaleWidth / 4
    d.Latime = d.Inaltime
    d.X = pic1.ScaleWidth / 2
    d.Y = pic1.ScaleHeight / 2
    Set d.PictureBox = Me.pic1
    referintaInterfata.Deseneaza
    Debug.Print "X=" & d.X; " Y=" & d.Y; ", L="; d.Latime; "H =
" & d.Inaltime
            Debug.Print   "Aria   dreptunghiului   este:"   &
referintaInterfata.Aria
End Sub
```


Codul foloseşte variabilele obiect **`c`** şi **`d`** pentru **`CCerc`** şi **`CDreptunghi`** şi o referinţă numită **`referintaInterfata`** la interfaţa **`IFormaGeometrica`** pentru a ilustra polimorfismul în VB.

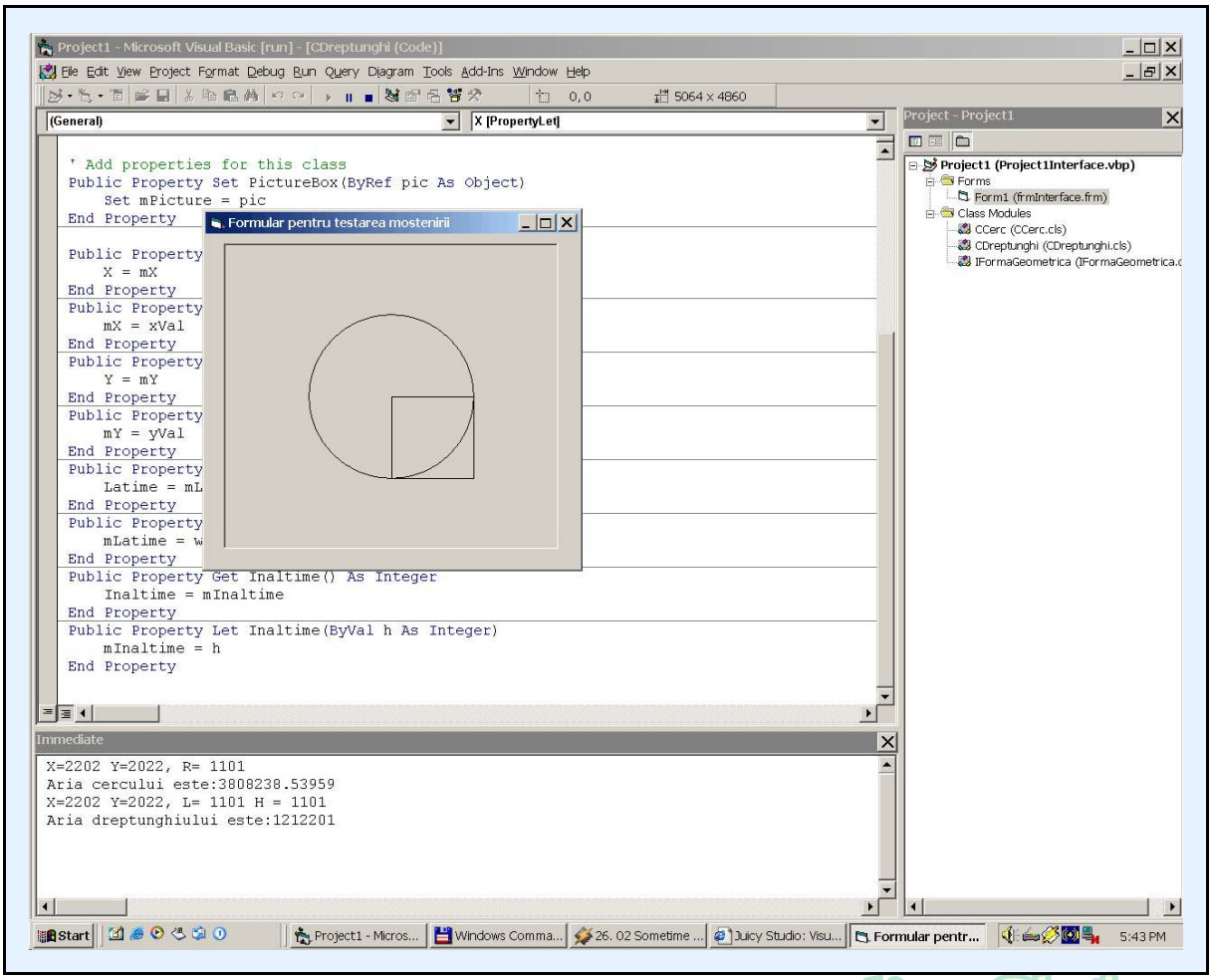

Figura 77 - Exemplificarea moştenirii la nivel de interfaţă în Visual BASIC.

14.6 Crearea unei clase pentru reprezentarea grafică a unor suprafeţe simple cu ascunderea feţelor invizibile

14.6.1 Clasa Class3DSurf

Clasa trebuie să aibă proprietatea `Name` setată la valoarea **`Class3DSurf`**. Ea foloseşte aceleaşi principii expuse în paragraful "Reprezentarea grafică a suprafeţelor" diferenţa esenţială fiind folosirea funcţiei API **`Polygon()`** pentru reprezentarea suprafeţei. Regiunea poligonală creată de **`Polygon`** are la bază un tablou de puncte de tipul **`POINTAPI`**. Windows închide automat poligonul conectând primul şi ultimul punct al regiunii poligonale. Marginea poligonului se afişează folosind setările curente pentru desenare, iar interiorul poligonului este umplut pe baza setărilor curente de umplere.

```
Option Explicit

Private Type punct3D
    X As Single
    Y As Single
```

```
    z As Single
End Type

Private Type POINTAPI
    X As Long
    Y As Long
End Type

Private mXmin As Single
Private mYmin As Single
Private mXmax As Single
Private mYmax As Single
Private mnx As Integer
Private mny As Integer
Private pts(1 To 4) As POINTAPI
Private mpuncte3d() As punct3D

Public s As Single
Public th As Single
Public Ascunde As Boolean

Private Declare Function Polygon Lib "gdi32" (ByVal hdc As Long,
lpPoint As POINTAPI, ByVal nCount As Long) As Long

'Creare tablou de valori
Public Sub SetareValori(ByVal x1 As Single, ByVal x2 As Single,
pasx As Single, _
ByVal y1 As Single, ByVal y2 As Single, ByVal pasy As Single,
ByRef zxy() As Single)

Dim X As Single
Dim Y As Single
Dim i As Integer
Dim j As Integer
Dim c As Long

    mnx = (x2 - x1) / pasx
    mny = (y2 - y1) / pasy
    ReDim mpuncte3d(mnx, mny)

    c = 0
    For i = 0 To mnx
        For j = 0 To mny
            mpuncte3d(i, j).X = zxy(c)
            mpuncte3d(i, j).Y = zxy(c + 1)
            mpuncte3d(i, j).z = zxy(c + 2)
            c = c + 3
        Next j
    Next i
```

```
End Sub

Private Sub Transformare3d2D()
Dim i As Integer
Dim j As Integer

    For i = 0 To mnx
        For j = 0 To mny
            mpuncte3d(i, j).X = mpuncte3d(i, j).X - _
                                mpuncte3d(i, j).Y * Cos(th) * s
            mpuncte3d(i, j).Y = mpuncte3d(i, j).z - _
                                mpuncte3d(i, j).Y * Sin(th) * s
            mpuncte3d(i, j).z = 0

        Next j
    Next i
End Sub

Private Sub Minmax()
    Dim i As Integer
    Dim j As Integer

    mXmin = mpuncte3d(0, 0).X
    mXmax = mpuncte3d(0, 0).X
    mYmin = mpuncte3d(0, 0).Y
    mYmax = mpuncte3d(0, 0).Y

    For i = 0 To mnx
        For j = 0 To mny
            If mpuncte3d(i, j).X > mXmax Then mXmax = _
                                              mpuncte3d(i, j).X
            If mpuncte3d(i, j).X < mXmin Then mXmin = _
                                              mpuncte3d(i, j).X
            If mpuncte3d(i, j).Y > mYmax Then mYmax = _
                                              mpuncte3d(i, j).Y
            If mpuncte3d(i, j).Y < mYmin Then mYmin = _
                                              mpuncte3d(i, j).Y
        Next j
    Next i
End Sub

Private Sub scalare2D(pic1 As PictureBox)
    Dim lx As Single
    Dim ly As Single
    Dim i As Integer
    Dim j As Integer

    ly = pic1.ScaleHeight
    lx = pic1.ScaleWidth
```

```
    For i = 0 To mnx
        For j = 0 To mny
            'fata
            mpuncte3d(i, j).X = scalare(mpuncte3d(i, j).X, _
                                        mXmin, mXmax) * lx
            mpuncte3d(i, j).Y = ly - _
                scalare(mpuncte3d(i, j).Y,  mYmin, mYmax) * ly
        Next j
    Next i
End Sub

Private Function scalare(X As Single, mXmin As Single, _
                                      mXmax As Single) As Single
    Dim d As Single
    d = mXmax - mXmin
    If d = 0 Then d = 1
    scalare = (X - mXmin) / d
End Function

Public Sub Grafic(ByVal pic1 As PictureBox)
    Dim i As Integer
    Dim j As Integer

    Call Transformare3d2D
    Call Minmax
    Call scalare2D(pic1)

    If Ascunde Then
        pic1.FillStyle = vbFSSolid
        pic1.FillColor = vbWhite
    Else
        pic1.FillStyle = vbFSTransparent
    End If

    For i = 0 To mnx - 1
        For j = 0 To mny - 1
            'Punctele unei fete
            pts(1).X = mpuncte3d(i, j).X
            pts(1).Y = mpuncte3d(i, j).Y
            pts(2).X = mpuncte3d(i + 1, j).X
            pts(2).Y = mpuncte3d(i + 1, j).Y
            pts(3).X = mpuncte3d(i + 1, j + 1).X
            pts(3).Y = mpuncte3d(i + 1, j + 1).Y
            pts(4).X = mpuncte3d(i, j + 1).X
            pts(4).Y = mpuncte3d(i, j + 1).Y

            Polygon pic1.hdc, pts(1), 4
        Next j
```

```
    Next i

End Sub
```

14.6.2 Formularul de reprezentare grafică

Formularul care folosește codul clasei de reprezentare grafică este prezentat în *Figura 78*, iar codul corespunzător lui urmează în continuare.

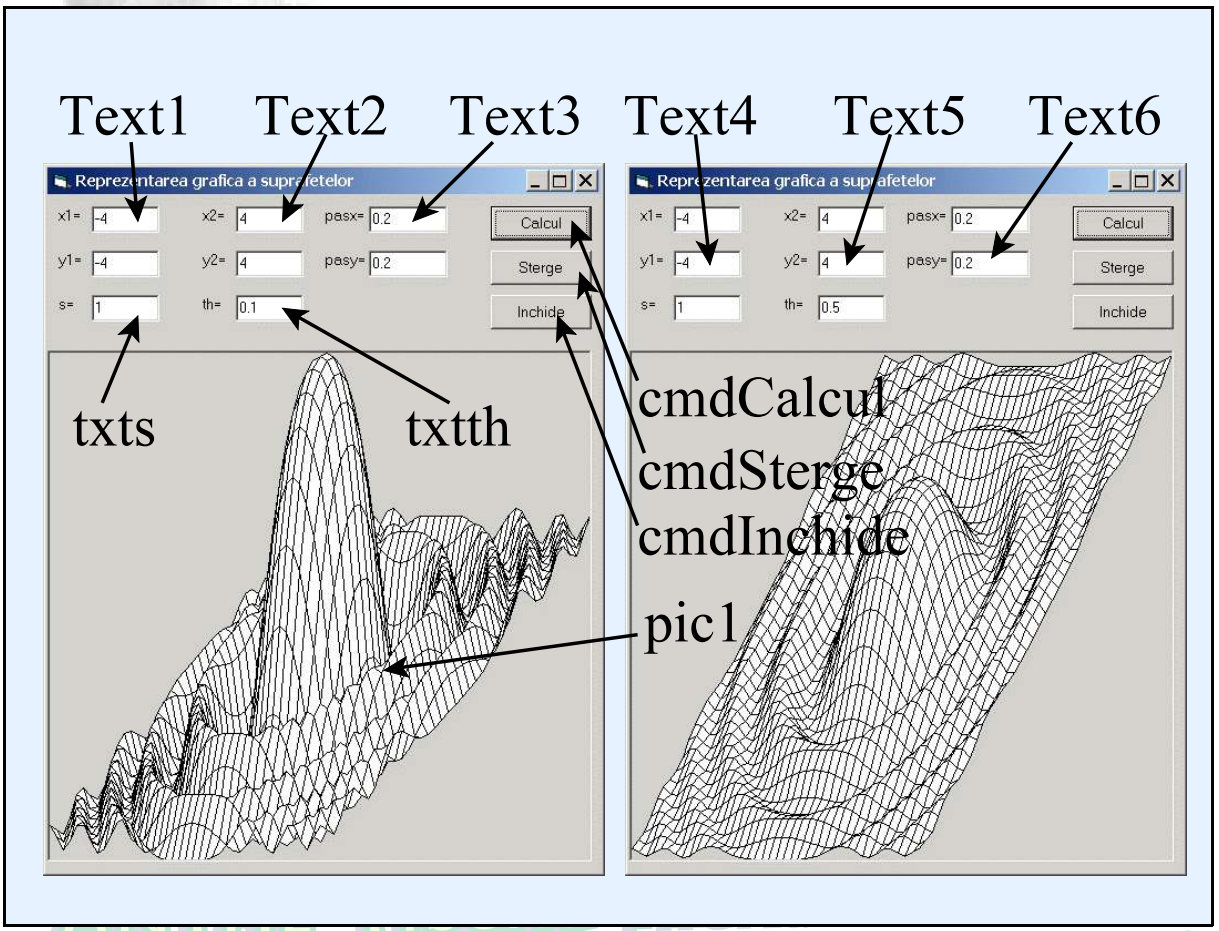

Figura 78 - Reprezentare grafică cu ascunderea fețelor invizibile.


```
Private Sub cmdCalcul_Click()
    Dim gr3d As Class3DSurf

    Dim nx As Integer
    Dim ny As Integer
    Dim i As Integer
    Dim j As Integer
    Dim x1 As Single
    Dim x2 As Single
    Dim X As Single
    Dim pasx As Single
    Dim y1 As Single
    Dim y2 As Single
```

```
    Dim Y As Single
    Dim pasy As Single
    Dim c As Long

    Dim zxy() As Single

    x1 = CSng(Text1.Text)
    x2 = CSng(Text2.Text)
    pasx = CSng(Text3.Text)
    nx = (x2 - x1) / pasx
    y1 = CSng(Text4.Text)
    y2 = CSng(Text5.Text)
    pasy = CSng(Text6.Text)
    ny = (y2 - y1) / pasy

    ReDim zxy(3 * (nx + 1) * (ny + 1))

    X = x1
    c = 0
    For i = 0 To nx
        Y = y1
        For j = 0 To ny
            zxy(c) = X
            zxy(c + 1) = Y
            zxy(c + 2) = f(X, Y)
            Y = Y + pasy
            c = c + 3
        Next j
        X = X + pasx
    Next i

    Set gr3d = New Class3DSurf
    gr3d.SetareValori x1, x2, pasx, y1, y2, pasy, zxy
    gr3d.s = CSng(txts)
    gr3d.th = CSng(txtth)
    gr3d.Ascunde = True
    gr3d.Grafic frmSuprafata.pic1
End Sub

Private Sub cmdInchide_Click()
    Unload Me
End Sub

Private Function f(X As Single, Y As Single) As Single
    f = Sin(1 + X * X + Y * Y) / (1 + X * X + Y * Y)
End Function
```

```
Private Sub cmdSterge_Click()
    frmSuprafata.pic1.Cls
End Sub
```