SIST. Y APLICACIONES INFORMÁTICAS
CURSO: 2009 - 2010
SEMANA:18
PROFESOR:Javier Sánchez Fernández
TEMAS: Programación Orientada a Objetos en Visual Basic.Net

1.- Realizar una clase de nombre Matrices que contenga los siguientes métodos:

1. Método constructor que define la dimensión de la matriz
2. Realizar un método rellenar, que rellene de manera aleatoria los elementos de la matriz creada.
3. Método escribir que muestra al usuario los datos contenidos en una matriz de tal forma que cada fila de la matriz se muestre en una linea de la pantalla de la consola.
4. Método Multimatriz que permita multiplicar dos matrices cuyas referencias son recibidas como argumentos. Igualmente se recibe como argumento la referencia de la matriz en la que se almacena el producto matricial.
5. Metodo SumarMatriz que permita sumar dos matrices que tengan igual numero de filas que de columnas.
6. Metodo Transponer que nos devuelve una matriz que tiene por filas las columnas de la matriz original, e.d. dado un elemento de la matriz original a_{ij} seria a_{ji} de la transpuesta.
7. Metodo Iguales, dos matrices son iguales si tienen el mismo numero de filas que de columnas y los elementos que ocupan la misma posición en ambas matrices son iguales

Posteriormente realizar un programa utilizando la clase, que solicite al usuario dos matrices, calcule su producto matricial y muestre el resultado.

Una matriz cuadrada es aquella que tiene igual numero de filas que de columnas, partiendo de la clase anterior, definir una clase de matrices cuadradas.

1. Dos matrices son simetricas si la transpuesta es igual a la original. Definir una operación en la que dada una matriz nos diga si es simétrica de la original
2. Calcular el determinante de una matriz cuadrada.

```
Interface Matriz
    Sub Trans(ByVal M As Matrices)
    Sub Sumar(ByVal M As Matrices, ByVal N As Matrices)
    Sub Multiplicar(ByVal M As Matrices, ByVal N As Matrices)

End Interface

Public Class Matrices
    Implements Matriz

    Private _filas, _columnas As Integer
    Private _m(,) As Integer

    Property filas()
        Get
            Return _filas
        End Get
        Set(ByVal Value)
            _filas = Value
        End Set
    End Property

    Property columnas()
        Get
            Return _columnas
        End Get
        Set(ByVal Value)
            _columnas = Value
        End Set
    End Property

    Property m(ByVal fil As Integer, ByVal col As Integer) As
Integer
        Get
            Return _m(fil, col)
        End Get
        Set(ByVal Value As Integer)
            _m(fil, col) = Value
        End Set
    End Property

    Public Sub New(ByVal fila As Integer, ByVal columna As
Integer)
        _filas = fila
        _columnas = columna
        ReDim _m(fila - 1, columna - 1)
    End Sub

    Public Sub Transpuesta(ByVal m1 As Matrices) Implements
Matriz.Trans
        Dim i, j As Integer

        For i = 0 To _filas - 1
```

```
            For j = 0 To _columnas - 1
                Me.m(i, j) = m1.m(j, i)
            Next
        Next
    End Sub

    Public Sub Sumar(ByVal m1 As Matrices, ByVal m2 As Matrices)
Implements Matriz.Sumar
        Dim i, j As Integer


        For i = 0 To m1.filas - 1
            For j = 0 To m1.columnas - 1
                Me.m(i, j) = m1.m(i, j) + m2.m(i, j)
            Next
        Next



    End Sub

    Public Sub Multiplicar(ByVal m1 As Matrices, ByVal m2 As
Matrices) Implements Matriz.Multiplicar
        Dim i, j, k, aux As Integer


        For i = 0 To m1.filas - 1
            For j = 0 To m2.columnas - 1
                aux = 0
                For k = 0 To m1.columnas - 1
                    aux += m1.m(i, k) * m2.m(k, j)
                Next
                Me.m(i, j) = aux
            Next
        Next

    End Sub

    Public Function iguales(ByVal m1 As Matrices) As Boolean
        ' Dos matrices son iguales si son iguales en filas y
columnas y ademas
        ' son iguales todos los elementos

        Dim i, j As Integer
        iguales = True
        If m1.filas = Me.filas And m1.columnas = Me.columnas Then
            For i = 0 To Me.filas - 1
                For j = 0 To Me.columnas - 1
                    If Me.m(i, j) <> m1.m(i, j) Then
                        Return False
                    End If
                Next
            Next
```

```
        Else : iguales = False
        End If
    End Function

    Public Sub escribir()
        Dim i, j As Integer
        For i = 0 To _filas - 1
            For j = 0 To _columnas - 1
                Console.Write("{0,6:d}", _m(i, j))
            Next
            Console.WriteLine()
        Next
    End Sub

    Public Sub rellenar()
        Dim i, j, alea, exp As Integer
        For i = 0 To _filas - 1
            For j = 0 To _columnas - 1
                alea = Int(Rnd() * 10)
                exp = Int(Rnd() * 2)
                _m(i, j) = (-1) ^ exp * alea
            Next
        Next
    End Sub
End Class


Public Class Matriz_cuadrada
    Inherits Matrices

    Public Sub New(ByVal filas As Integer)
        MyBase.New(filas, filas)
    End Sub

    Public Function Determinante() As Double

        Dim DimensionX As Integer

        Dim i As Integer, Det As Double
        Dim MenorM As Matriz_cuadrada


        If Me.columnas = 1 Then
            Determinante = Me.m(0, 0)
            Exit Function
        End If

        For i = 0 To Me.filas - 1
            MenorM = Me.HazMenorM(Me, i, 0)
            Det += ((-1) ^ i) * Me.m(i, 0) * MenorM.Determinante()

        Next
        Determinante = Det
```

```
    End Function


    Private Function HazMenorM(ByVal Mat As Matriz_cuadrada, ByVal
MenorX As Integer, ByVal MenorY As Integer) As Matriz_cuadrada

        Dim DimensionX As Integer
        Dim i As Integer, j As Integer

        Dim MenorM As Matriz_cuadrada
        DimensionX = Mat.filas - 1


        MenorM = New Matriz_cuadrada(DimensionX)
        For i = 0 To MenorX - 1
            For j = 0 To MenorY - 1
                MenorM.m(i, j) = Mat.m(i, j)
            Next
        Next

        For i = MenorX + 1 To Mat.filas - 1
            For j = 0 To MenorY - 1
                MenorM.m(i - 1, j) = Mat.m(i, j)
            Next
        Next

        For i = 0 To MenorX - 1
            For j = MenorY + 1 To Mat.filas - 1
                MenorM.m(i, j - 1) = Mat.m(i, j)
            Next
        Next

        For i = MenorX + 1 To Mat.filas - 1
            For j = MenorY + 1 To Mat.filas - 1
                MenorM.m(i - 1, j - 1) = Mat.m(i, j)
            Next
        Next

        Return MenorM
    End Function

    Public Function simetrica() As Boolean
        Dim mat As Matriz_cuadrada = New Matriz_cuadrada(Me.filas)
        mat.transpuesta(Me)
        Return Me.iguales(mat)
    End Function

    Public Sub adjunta(ByVal m1 As Matriz_cuadrada)
        'la matriz adjunta de una matriz cuadrada es la que se
obtiene al sustituir cada elemento de dicha
        'matriz por el determinante del menor de su simetrica
respecto a la diagonal principal
```

```
        Dim i, j As Integer
        Dim Adjunto As Matriz_cuadrada
        Dim res, det As Double

        det = m1.Determinante
        For i = 0 To m1.filas - 1
            For j = 0 To m1.columnas - 1
                Adjunto = m1.HazMenorM(m1, i, j)
                res = (-1) ^ (i + j) * Adjunto.Determinante
                Me.m(j, i) = res ' para respetar la definicion
intercambiamos i j
            Next
        Next
    End Sub
End Class
```

2.-Realizar una clase de nombre Alumno que cumpla las siguientes especificaciones:

1. El constructor admite como argumento el número de matricula del alumno y almacena ésta en una variable de tipo double

2. Contiene los siguientes métodos:

   - PonNota con dos argumentos de tipo double que corresponden a dos notas de un examen.El método almacena éstos en sendas variables de tipo double.

   - 2.2.-DameMedia retorna la media de las notas.

   - 2.3.-Muestra por pantalla ambas notas. Sustituyendo su valor por las siglas “N.P.” si la nota es menor que cero

   Realizar un programa que utilizando esta clase realice las siguientes tareas:

   1. Crea un array de 100 elementos de tipo Alumno
   2. Solicita al usuario los números de matricula de los alumnos y las notas; repite la operación hasta que el número de matricula sea menor que cero.
   3. Calcula la nota media de las medias de cada alumno y la muestra en pantalla
   4. Muestra de cada alumno sus dos notas y la nota media

```
Public Class Alumno
    Private np As Integer
    Private nota1 As Double
    Private nota2 As Double

    Public Sub New(ByVal np As Integer)
        Me.np = np
    End Sub

    Public Sub ponNota(ByVal nota1 As Double, ByVal nota2 As
Double)
        Me.nota1 = nota1
        Me.nota2 = nota2
    End Sub

    Public Function dameMedia()
        Return (nota1 + nota2) / 2
    End Function
```

```
    Public Sub imprime()
        If nota1 = 0 AndAlso nota2 > 0 Then
            Console.WriteLine("Alumno: {0} Notas N.P.,{1}", np,
nota2)
        ElseIf nota2 = 0 AndAlso nota1 > 0 Then
            Console.WriteLine("Alumno: {0} Notas:{1},N.P.", np,
nota2)
        ElseIf nota1 = 0 And nota2 = 0 Then
            Console.WriteLine("Alumno: {0} Notas: N.P.,N.P.", np)
        Else
            Console.WriteLine("Alumno: {0} Notas:{1},{2}", np,
nota1, nota2)
        End If

    End Sub
End Class


Module Module1

    Sub Main()

        Const MAX = 100
        Dim nota1, nota2, aux As Double
        Dim cont = 0, np = 1, i As Integer
        Dim ListaAlumno(MAX - 1) As Alumno
        Dim alum As Alumno

        Do While np > 0 AndAlso cont < MAX - 1
            Console.Write("Escribe el numero de la matricula
(negativo para terminar): ")
            np = Console.ReadLine()
            If np > 0 Then
                Console.Write("Introduce la nota del primer
cuatrimestre: ")
                nota1 = Console.ReadLine
                Console.Write("Introduce la nota del segundo
cuatrimestre: ")
                nota2 = Console.ReadLine
                alum = New Alumno(np)
                alum.ponNota(nota1, nota2)
                ListaAlumno(cont) = alum
                cont = +1


            End If
        Loop
        For i = 0 To cont
            aux += ListaAlumno(i).dameMedia()
        Next
```

```
        Console.WriteLine("La media de la clase es: {0}", aux /
(cont + 1))

        For i = 0 To cont
            ListaAlumno(i).imprime()
            Console.WriteLine("Media: {0} ",
ListaAlumno(i).dameMedia())
        Next

    End Sub

End Module
```

3.- Implementa la clase de los números naturales

```
Class nat
    Dim x As Integer

    Public Sub New()
        x = 0
    End Sub

    Public Sub New(ByVal val As Integer)
        If val > 0 Then
            x = val
        Else : x = 0
        End If
    End Sub

    Function suc() As nat
        Dim aux As nat = New nat(Me.x)

        aux.x += 1
        Return aux
    End Function

    Function dec() As nat
        Dim aux As nat = New nat(Me.x)

        If x > 0 Then
            aux.x -= 1
            Return aux
        End If
    End Function

    Function escero() As Boolean
        If x = 0 Then
            Return True
        Else : Return False
        End If
    End Function

    Public Function suma(ByVal num1 As nat) As nat
        Dim aux As nat = New nat(Me.x)

        While Not (num1.escero)
            aux = aux.suc()
            num1 = num1.dec()
        End While
        Return aux
    End Function

    Function producto(ByVal num As nat) As nat
        Dim aux As nat
        aux = New nat(Me.x * num.x)
        Return aux
```

```
    End Function

    Public Overrides Function tostring() As String
        Return x
    End Function
End Class

Module Module1

    Sub Main()
        Dim ent1, ent2, ent3 As nat

        Randomize()

        Do
            ent1 = New nat(Int(Rnd() * 100))
            ent2 = New nat(Int(Rnd() * 100))
            ent3 = ent1.suma(ent2)
            Console.WriteLine(ent1.ToString & "+" & ent2.ToString
& "=" & ent3.ToString)
            ent3 = ent1.producto(ent2)
            Console.WriteLine(ent1.ToString & "*" & ent2.ToString
& "=" & ent3.ToString)

            Console.ReadLine()
        Loop

    End Sub

End Module
```

4.- Implementa la clase de los números racionales

```
Imports System.Math

Public Class CRacionales
    Private _numerador As Integer
    Private _denominador As Integer


    Public Sub New()

    End Sub

    Public Sub New(ByVal num As Integer, ByVal den As Integer)

        _numerador = num
        _denominador = den
        Me.simplifica()
    End Sub

    Public Function numerador() As Integer
        Return _numerador
    End Function

    Public Sub numerador(ByVal n As Integer)
        _numerador = n
    End Sub

    Public Sub denominador(ByVal n As Integer)
        _denominador = n
    End Sub

    Public Function denominador() As Integer
        Return _denominador
    End Function

    Public Overrides Function ToString() As String
        Return (_numerador & "/" & _denominador)
    End Function

    Public Sub suma(ByVal R1 As CRacionales, ByVal R2 As
CRacionales)

        Me._numerador = R1.numerador * R2.denominador +
R2.numerador * R1.denominador
        Me._denominador = R1.denominador * R2.denominador

    End Sub

    Public Sub resta(ByVal R1 As CRacionales, ByVal R2 As
CRacionales)
```

```
        Me._numerador = R1.numerador * R2.denominador -
R2.numerador * R1.denominador
        Me._denominador = R1.denominador * R2.denominador

    End Sub

    Public Sub producto(ByVal R1 As CRacionales, ByVal R2 As
CRacionales)

        Me._numerador = R1.numerador * R2.numerador
        Me._denominador = R1.denominador * R2.denominador
    End Sub

    Public Sub division(ByVal R1 As CRacionales, ByVal R2 As
CRacionales)
        Me._numerador = R1.numerador * R2.denominador
        Me._denominador = R1.denominador * R2.numerador
    End Sub

    Public Function MCD(ByVal num1 As Integer, ByVal num2 As
Integer) As Integer
        If num2 = 0 Then
            Return num1
        Else
            Return MCD(num2, num1 Mod num2)
        End If
    End Function

    Public Sub simplifica()
        Dim divisor As Integer

        If Abs(numerador()) > Abs(denominador()) Then
            divisor = MCD(Abs(numerador()), Abs(denominador()))
        Else : divisor = MCD(Abs(denominador()), Abs(numerador()))

        End If

        If divisor <> 1 Then
            _denominador /= divisor
            _numerador /= divisor
        End If
    End Sub
End Class

Public Class CEnteros
    Inherits CRacionales

    Public Sub New()
        MyBase.New(0, 1)
    End Sub

    Public Sub New(ByVal num As Integer)
        MyBase.New(num, 1)
```

```
    End Sub

    Public Overrides Function tostring() As String
        'Si el denominador es distinto de 1 entonces ha resultado
        'que dividir dos numeros enteros nos ha devuelto un numero
racional
        'y deberemos llamar al procedimiento de la clase padre

        If Me.denominador <> 1 Then
            Return MyBase.ToString()
        Else
            Return (Me.numerador)
        End If
    End Function
```

5.- Se quiere modelar una agencia inmobiliaria para lo cual se deben tener en cuenta las siguientes entidades e informaciones asociadas:

a) Un inmueble viene dado por su ubicación y los metros cuadrados que ocupa. Hay dos tipos de inmuebles: superficies y construcciones. Los primeros tienen su precio por metro cuadrado. A su vez, hay dos tipos de superficies: solares y plazas de garaje. Los solares pueden estar en zona rústica o urbana. Las plazas de garaje pueden formar parte de un garaje público o de uno privado. Por su parte, las construcciones pueden ser nuevas o de segunda mano. Hay dos tipos de construcciones: viviendas y locales comerciales. Las viviendas tienen precio, número de habitaciones y piso. Los locales tienen precio por metro cuadrado.
b) La agencia que queremos modelar se dedica a la venta y alquiler de inmuebles. Ahora bien, solamente alquila plazas de garaje y locales comerciales, mientras que solo vende solares y viviendas.
c) La agencia inmobiliaria viene dada por sendas secuencias de alquileres y ventas. Para modelar estas secuencias empleamos un array junto con un contador que nos marca su nivel de ocupación. El array no tiene por qué estar ordenado.

Se pide:

- ❑ Definir la jerarquía de clases de forma que se cumplan los requisitos anteriores.
- ❑ Para cada una de las clases, definir su(s) constructora(s), métodos de acceso a cada uno de los atributos y un método muestra() que escriba, de la manera que consideres más conveniente, la información de la entidad de que se trate.
- ❑ Definir un método público precio() que calcula el precio que tiene una superficie.
- ❑ Definir un método añadeVentaInmueble(in), que añade el inmueble dado por el parámetro a la secuencia de inmuebles en venta de la agencia, siempre que no estuviera ya antes en venta.
- ❑ Definir un método añadeAlquilerInmueble(in), que añade el inmueble dado por el parámetro a la secuencia de inmuebles en alquiler de la agencia, siempre que no estuviera ya antes en alquiler.
- ❑ Definir métodos que respondan a los siguientes servicios:

    1. inmueblesVenta(p), que muestra los inmuebles con un precio de venta inferior al parámetro.
    2. localesSegundaMano(m), que muestra los locales comerciales de segunda mano con una superficie superior al parámetro.

3. solaresRusticos(), que averigua cuántos solares no urbanos están en venta.

```
Public Class inmueble
    Protected _Ubicacion As String
    Protected _Tamaño As Integer

    Public Sub New()

    End Sub

    Public Sub New(ByVal lugar As String, ByVal tam As Integer)
        _Ubicacion = lugar
        _Tamaño = tam
    End Sub

    Property ubicacion() As String
        Get
            Return _Ubicacion
        End Get
        Set(ByVal Value As String)
            _Ubicacion = Value
        End Set
    End Property

    Property tamaño() As Integer
        Get
            Return _Tamaño
        End Get
        Set(ByVal Value As Integer)
            _Tamaño = Value
        End Set
    End Property

End Class

Public Class superficie
    Inherits inmueble

    Protected _precio_m2 As Integer

    Public Sub New()

    End Sub

    Public Sub New(ByVal dir As String, ByVal sup As Integer,
ByVal pre As Integer)
        MyBase.New(dir, sup)
        _precio_m2 = pre
    End Sub

    Property precio_m2() As Integer
```

```
        Get
            Return _precio_m2
        End Get
        Set(ByVal Value As Integer)
            _precio_m2 = Value
        End Set
    End Property

    Public Function precio() As Integer
        Return _precio_m2 * MyBase.tamaño
    End Function



End Class

Public Class construccion
    Inherits inmueble

    Protected _nueva As Boolean

    Public Sub New()

    End Sub

    Public Sub New(ByVal dir As String, ByVal sup As Integer,
ByVal nue As Boolean)
        MyBase.New(dir, sup)
        _nueva = nue
    End Sub

    Property nueva() As String
        Get
            If _nueva Then
                Return "nueva"
            Else
                Return "Segunda mano"
            End If
        End Get
        Set(ByVal Value As String)
            If Value = "nueva" Then
                _nueva = True
            Else
                _nueva = False
            End If

        End Set
    End Property



End Class
```

```
Public Class Solar
    Inherits superficie

    Private _rustico As Boolean


    Public Sub New()

    End Sub

    Public Sub New(ByVal dir As String, ByVal sup As Integer,
ByVal prec As Integer, ByVal rus As Boolean)
        MyBase.New(dir, sup, prec)
        _rustico = rus
    End Sub

    Public Property rustico() As String
        Get
            If _rustico Then
                Return "rustico"
            Else : Return "urbano"
            End If
            Return _rustico
        End Get
        Set(ByVal Value As String)
            If Value = "rustico" Then
                _rustico = True
            Else
                _rustico = False
            End If

        End Set
    End Property


    Public Sub muestra()
        Console.WriteLine("EL solar es {4} tiene {0,6:d} m2, esta
ubicada " & _
                          "en {1} y su valor es {2,6:d}",
MyBase.tamaño, _
                          MyBase.ubicacion, precio, rustico())
    End Sub

    Public Function CreaSolar() As Solar

        Dim unsolar As Solar
        Dim sup, pre As Integer
        Dim dir, rust As String
        Dim nue As Boolean

        Console.Write("Direccion         : ")
        dir = Console.ReadLine()
        Console.Write("Superfice         : ")
```

```
        sup = Console.ReadLine()
        Console.Write("Precio            : ")
        pre = Console.ReadLine()
        Console.Write("Rustico      [s/n]: ")
        rust = Console.ReadLine
        If rust = "s" Or "S" Then
            nue = False
        Else
            nue = True
        End If
        unsolar = New Solar(dir, sup, pre, nue)
        Return unsolar
    End Function
End Class

Public Class plazagarage
    Inherits superficie

    Private _publico As Boolean

    Public Sub New()

    End Sub

    Public Sub New(ByVal dir As String, ByVal sup As Integer,
ByVal pre As Integer, ByVal pub As Boolean)
        MyBase.New(dir, sup, pre)
        _publico = pub
    End Sub

    Public Property publico() As String
        Get
            If _publico Then
                Return "Publico"
            Else
                Return "Privado"
            End If
        End Get
        Set(ByVal Value As String)
            If Value = "Publico" Then
                _publico = True
            Else
                _publico = False
            End If
        End Set
    End Property

    Public Sub muestra()
        Console.WriteLine("La plaza de garage esta ubicada {0}, "
& _
                          "tiene {1} m2 y esta situado en un
garage {2}", _
```

```
                            MyBase.ubicacion, MyBase.tamaño,
publico())
    End Sub

    Public Function creagarage() As plazagarage
        Dim ungarage As plazagarage
        Dim sup, piso, pre As Integer
        Dim dir, publico As String
        Dim pub As Boolean

        Console.Write("Direccion           : ")
        dir = Console.ReadLine()
        Console.Write("Superfice           : ")
        sup = Console.ReadLine()
        Console.Write("Precio              : ")
        pre = Console.ReadLine()
        Console.Write("Garaje Publico [s/n]: ")
        publico = Console.ReadLine
        If publico = "s" Or publico = "S" Then
            pub = False
        Else
            pub = True
        End If
        ungarage = New plazagarage(dir, sup, pre, pub)
        Return ungarage
    End Function

End Class

Public Class Vivienda
    Inherits construccion

    Private _precio As Integer
    Private _numhab As Integer
    Private _piso As Integer

    Public Sub New()

    End Sub
    Public Sub New(ByVal dir As String, ByVal sup As Integer, _
                   ByVal pre As Integer, ByVal hab As Integer, _
                   ByVal piso As Integer, ByVal nue As Boolean)

        MyBase.New(dir, sup, nue)
        _precio = pre
        _numhab = hab
        _piso = piso

    End Sub

    Property precio() As Integer
        Get
            Return _precio
```

```
        End Get
        Set(ByVal Value As Integer)
            _precio = Value
        End Set
    End Property

    Property numeroHabitaciones() As Integer
        Get
            Return _numhab
        End Get
        Set(ByVal Value As Integer)
            _numhab = Value
        End Set
    End Property

    Property piso() As Integer
        Get
            Return _piso
        End Get
        Set(ByVal Value As Integer)
            _piso = Value
        End Set
    End Property

    Public Sub muestra()
        Console.WriteLine("La vivienda es un {5} piso, es {3}
tiene {0,6:d} m2," & _
                          " tiene {4} habitaciones esta ubicada en
{1} y su valor es de {2,6:d} Euros", _
                          MyBase.tamaño, MyBase.ubicacion, precio,
MyBase.nueva, numeroHabitaciones, orden(piso))
    End Sub

    Private Function orden(ByVal num As Integer) As String
        Select Case num
            Case 1
                Return "Primer"
            Case 2
                Return "Segundo"
            Case 3
                Return "Tercer"
            Case 4
                Return "Cuarto"
            Case 5
                Return "Quinto"
            Case 6
                Return "Sexto"
            Case 7
                Return "Septimo"
            Case 8
                Return "Octavo"
            Case 9
                Return "Noveno"
```

```
            Case 10
                Return "Decimo"
            Case 11
                Return "Undecimo"
            Case 12
                Return "Duodecimo"
            Case 13
                Return ("Decimotercero")
            Case Else
                Return num.ToString
        End Select
    End Function

    Public Function CreaVivienda() As Vivienda
        Dim unavivienda As Vivienda
        Dim resp, sup, piso, pre, hab As Integer
        Dim dir, usada As String
        Dim nue As Boolean

        Console.Write("Direccion          : ")
        dir = Console.ReadLine()
        Console.Write("Superfice          : ")
        sup = Console.ReadLine()
        Console.Write("Piso               : ")
        piso = Console.ReadLine()
        Console.Write("Precio             : ")
        pre = Console.ReadLine()
        Console.Write("Habitaciones       : ")
        hab = Console.ReadLine()
        Console.Write("Segunda Mano [s/n]: ")
        usada = Console.ReadLine
        If usada = "s" Or usada = "S" Then
            nue = False
        Else
            nue = True
        End If
        unavivienda = New Vivienda(dir, sup, pre, hab, piso, nue)

        Return unavivienda
    End Function
End Class

Public Class localcomercial
    Inherits construccion

    Private precio As Integer

    Public Sub New()

    End Sub

    Public Sub New(ByVal dir As String, ByVal tam As Integer,
ByVal nue As Boolean, ByVal pre As Integer)
```

```
        MyBase.New(dir, tam, nue)
        precio = pre

    End Sub

    Public Sub muestra()
        Console.WriteLine("El local comercial tiene {0,6:d} m2" &
_
                          ", esta situado en {1} y su valor es
{2,6:d}", _
                          MyBase.tamaño, MyBase.ubicacion, precio)
    End Sub

    Public Function crealocalcomercial() As localcomercial
        Dim unlocalcomercial As localcomercial
        Dim resp, sup, pre As Integer
        Dim dir, usada As String
        Dim nue As Boolean

        Console.Write("Direccion          : ")
        dir = Console.ReadLine()
        Console.Write("Superfice          : ")
        sup = Console.ReadLine()
        Console.Write("Precio             : ")
        pre = Console.ReadLine()
        Console.Write("Rustico      [s/n]: ")
        usada = Console.ReadLine
        If usada = "s" Or "S" Then
            nue = False
        Else
            nue = True
        End If
        unlocalcomercial = New localcomercial(dir, sup, pre, nue)

        Return unlocalcomercial
    End Function
End Class

Public Class agencia

    'La clase agencia tenemos dos estructuras una con los
inmuebles en venta y otra con los inmuebles alquilados, los
locales alquilados pueden ser de dos tipos garages y locales
comerciales, cada uno lo almacenaremos en un array diferente. Los
locales en venta pueden ser viviendas y solares

    Structure enVenta
        Public contViviendas, contSolares As Integer
        Public viviendas() As Vivienda
        Public solares() As Solar
    End Structure

    Structure enAlquiler
```

```
        Public contGarajes, contLocales As Integer
        Public garajes() As plazagarage
        Public localesComerciales() As localcomercial
    End Structure

    Public ventas As enVenta
    Public alquileres As enAlquiler

    Public Function esta(ByVal viv As Vivienda) As Boolean
        Dim encontrada As Boolean = False
        Dim cont As Integer

        With ventas
            While cont < .contViviendas And Not encontrada
                If .viviendas(cont).ubicacion = viv.ubicacion And
_
                .viviendas(cont).tamaño = viv.tamaño And _
                .viviendas(cont).piso = viv.piso And _
                .viviendas(cont).numeroHabitaciones =
viv.numeroHabitaciones Then
                    encontrada = True
                Else
                    cont += 1
                End If

            End While
        End With
        Return encontrada
    End Function

    Public Function esta(ByVal sol As Solar) As Boolean
        Dim encontrada As Boolean = False
        Dim cont As Integer

        With ventas
            While cont < .contSolares And Not encontrada
                If .solares(cont).ubicacion = sol.ubicacion And _
                .solares(cont).tamaño = sol.tamaño Then
                    encontrada = True
                Else
                    cont += 1
                End If

            End While
        End With
        Return encontrada

    End Function

    Public Function esta(ByVal pla As plazagarage) As Boolean
        Dim encontrada As Boolean = False
        Dim cont As Integer
```

```
        With alquileres
            While cont < .contGarajes And Not encontrada
                If .garajes(cont).ubicacion = pla.ubicacion Then
                    encontrada = True
                Else
                    cont += 1
                End If

            End While
        End With
        Return encontrada
    End Function

    Public Function esta(ByVal loc As localcomercial) As Boolean
        Dim encontrada As Boolean = False
        Dim cont As Integer

        With alquileres
            While cont < .contLocales And Not encontrada
                If .localesComerciales(cont).ubicacion =
loc.ubicacion Then
                    encontrada = True
                Else
                    cont += 1
                End If

            End While
        End With
        Return encontrada
    End Function

    Public Sub lista(ByVal viv() As Vivienda)
        Dim i As Integer
        For i = 0 To viv.Length - 1
            viv(i).muestra()
            Console.WriteLine()
        Next
    End Sub

    Public Sub lista(ByVal loc() As localcomercial)
        Dim i As Integer
        For i = 0 To loc.Length - 1
            loc(i).muestra()
            Console.WriteLine()
        Next
    End Sub

    Public Sub lista(ByVal pla() As plazagarage)
        Dim i As Integer
        For i = 0 To pla.Length - 1
            pla(i).muestra()
            Console.WriteLine()
        Next
```

```
    End Sub

    Public Sub lista(ByVal sol() As Solar)
        Dim i As Integer
        For i = 0 To sol.Length - 1
            sol(i).muestra()
            Console.WriteLine()
        Next
    End Sub



    'Tenemos dos funciones para añadir un inmueble a la venta
    'Debo tener dos ya que no puedo pasarle un objeto de tipo
inmueble
    'y preguntar que tipo de objeto es: vivienda o solar


    Public Sub añadeVentaInmueble(ByVal unavivienda As Vivienda)

        If esta(unavivienda) Then
            Console.WriteLine("La vivienda {0} ya esta en la BD",
unavivienda.ubicacion)
        Else
            ReDim Preserve ventas.viviendas(ventas.contViviendas)
            ventas.viviendas(ventas.contViviendas) = unavivienda
            ventas.contViviendas += 1
        End If
    End Sub

    Public Sub añadeVentaInmueble(ByVal unsolar As Solar)
        If esta(unsolar) Then
            Console.WriteLine("La vivienda ya esta en la BD")
        Else
            ReDim Preserve ventas.solares(ventas.contSolares)
            ventas.solares(ventas.contSolares) = unsolar
            ventas.contSolares += 1
        End If
    End Sub

    'Tenemos dos funciones para añadir un inmueble al alquiler
    'Debo tener dos ya que no puedo pasarle un objeto de tipo
inmueble y preguntar
    'que tipo de objeto es: plaza o local comercial

    Public Sub añadeAlquilerInmueble(ByVal unlocalcomercial As
localcomercial)
        If esta(unlocalcomercial) Then
            Console.WriteLine("El local ya esta en la BD")
        Else
            ReDim Preserve
alquileres.localesComerciales(alquileres.contLocales)
```

```
            alquileres.localesComerciales(alquileres.contLocales)
= unlocalcomercial
            alquileres.contLocales += 1
        End If

    End Sub

    Public Sub añadeAlquilerInmueble(ByVal ungarage As
plazagarage)

        If esta(ungarage) Then
            Console.WriteLine("EL garage ya esta en la BD")
        Else
            ReDim Preserve
alquileres.garajes(alquileres.contGarajes)
            alquileres.garajes(alquileres.contGarajes) = ungarage
            alquileres.contGarajes += 1
        End If
    End Sub

    Public Sub inmueblesVenta(ByVal cantidad As Integer)

        Dim i, cont As Integer


        Console.WriteLine("Los inmuebles en venta que coinciden
con su peticion")

Console.WriteLine("_____________________________________________
_____")
        With ventas
            For i = 0 To .contViviendas - 1
                If .viviendas(i).precio < cantidad Then
                    .viviendas(i).muestra()
                    cont += 1
                End If
            Next
            Console.WriteLine("Hay {0} viviendas que coinciden con
su peticion", cont)
            For i = 0 To .contSolares - 1
                If .solares(i).precio < cantidad Then
                    .solares(i).muestra()
                End If
            Next
        End With
    End Sub

    Public Sub localesSegundaMano(ByVal sup As Integer)

        Dim i As Integer

        Console.WriteLine("los locales comerciales de 2ª mano con
una superficie menor a {0} son:", sup)
```

```
        With alquileres
            For i = 0 To .contLocales - 1
                If .localesComerciales(i).nueva = False And
.localesComerciales(i).tamaño < sup Then
                    .localesComerciales(i).muestra()

                End If
            Next
        End With

    End Sub

    Public Sub solaresRusticos()

        Dim i As Integer
        Console.WriteLine("Los locales rusticos que estan en venta
son los siguientes")

Console.WriteLine("_____________________________________________")
        With ventas
            For i = 0 To .contSolares
                If .solares(i).rustico = True Then
                    .solares(i).muestra()
                End If
            Next
        End With
    End Sub

End Class

Module Module1

    Sub Main()
        Dim res, res1 As Integer
        Dim unaagencia As agencia = New agencia()
        Dim micasa As Vivienda = New Vivienda()
        Dim migarage As plazagarage = New plazagarage()
        Dim milocal As localcomercial = New localcomercial()
        Dim misolar As Solar = New Solar()

        Dim micasa1 As Vivienda
        Dim migarage1 As plazagarage
        Dim milocal1 As localcomercial
        Dim misolar1 As Solar


        unaagencia.añadeVentaInmueble(micasa1)

        unaagencia.inmueblesVenta(150000)
        migarage = New plazagarage()
        milocal = New localcomercial()
        misolar = New Solar()
```

```
        Dim i As Integer



        Do
            Console.WriteLine("Agencia Nuevos Horizontes ")
            Console.WriteLine("_____________________________")
            Console.WriteLine("Venta                      ")
            Console.WriteLine("   1.-Viviendas            ")
            Console.WriteLine("   2.-Solares              ")
            Console.WriteLine("Alquiler                   ")
            Console.WriteLine("   3.-Locales Comerciales  ")
            Console.WriteLine("   4.-Plazas de garage     ")
            Console.WriteLine("5.-Listar Locales          ")
            Console.WriteLine("6.-Salir                   ")
            Console.Write("Introduzca Opcion [1/2]:       ")
            res = Console.ReadLine
            Select Case res
                Case 1
                    micasa = micasa.CreaVivienda
                    unaagencia.añadeVentaInmueble(micasa)
                Case 2
                    misolar = misolar.CreaSolar
                    unaagencia.añadeVentaInmueble(misolar)
                Case 3
                    milocal = milocal.crealocalcomercial
                    unaagencia.añadeAlquilerInmueble(milocal)
                Case 4
                    migarage = migarage.creagarage
                    unaagencia.añadeAlquilerInmueble(migarage)
                Case 5
                    Console.WriteLine("Elija la categoria de los
elementos a Listar")
                    Console.WriteLine("1.-Viviendas")
                    Console.WriteLine("2.-Solares")
                    Console.WriteLine("3.-Locales Comerciales")
                    Console.WriteLine("4.-Plazas de Garage")
                    Console.WriteLine("Elija opcion [1/4]")
                    res = Console.ReadLine
                    Select Case res
                        Case 1
                            Console.WriteLine("1.-Listar todos los
elementos")
                            Console.WriteLine("2.- Listar las
viviendas cuyo precio sea menor que:")
                            Console.WriteLine("Elija una opcion
[1/2]")
                            res = Console.ReadLine()
                            Select Case res
                                Case 1

unaagencia.lista(unaagencia.ventas.viviendas)
                                Case 2
```

```
                                    Console.WriteLine("Introduzca
una cantidad")
                                    res1 = Console.ReadLine

unaagencia.inmueblesVenta(res1)
                                End Select
                            Case 2

unaagencia.lista(unaagencia.ventas.solares)
                            Case 3

unaagencia.lista(unaagencia.alquileres.localesComerciales)
                            Case 4

unaagencia.lista(unaagencia.alquileres.garajes)
                        End Select

                    Case 6
                        Exit Do
                End Select
            Loop
        End Sub

End Module
```

6. Una operación desapila(p, n) que elimine los últimos n elementos apilados en una pila p si los hay.

```
  Public Sub desapilar(ByVal n As Integer)
            'Este procedimiento saca de una pila n elementos de
una pila si lo hay
            'Lo unico que tenemos que hacer es ir desapilando
elementos y decrementando
            'un contador hasta que el contador sea cero

            Dim pilaux As TipoPila = New TipoPila()
            Dim lon As Integer

            pilaux.copiar(Me)
            lon = pilaux.longitud

            Select Case lon
                Case Is < n
                    Console.WriteLine("La pila no tiene
suficientes elementos")
                Case Is = n
                    Me._pila.purgar()
                Case Is > n
                    While n <> 0
                        Me.desapilar()
                        n -= 1
                    End While
            End Select

        End Sub
```

7. Una operación inversa(p) que genere la inversa de una pila p.

```
        Public Function inversa() As tipoPila
 'Esta funcion nos devuelve una pila inversa.El algoritmo es
iterativo. Copiamos la lista en una variable local Pilaux y
creamos otra variable de tipo Pila que sera la que devolvamos.
Para crear la pila inversa solamente tenemos que apilar la cima de
la pila en la nueva pila

            Dim pilaux As TipoPila = New TipoPila()
            Dim pilaInv As TipoPila = New TipoPila()

            pilaux.copiar(Me)

            While Not pilaux.vacia
                pilaInv.apilar(pilaux.cima)
                pilaux.desapilar()
            End While
            Return pilaInv

        End Function
```

8. Una operación fondo(p) que calcule el elemento del fondo de una pila p.

```
Public Function fondo() As Object
'Esta función nos devuelve el ultimo elemento de la pila. El
algoritmo es iterativo. Como en las funciones anteriores copiamos
la pila, vamos desapilando esta copia hasta que la pila este vacia
y guardando el elemento de la cima. Cuando la pila esta vacia. El
fondo sera este último elemento leido

        Dim pilaux As TipoPila = New TipoPila()
        Dim fon As Object

        pilaux.copiar(Me)
        While Not pilaux.vacia
            fon = pilaux.cima
            pilaux.desapilar()
        End While
        Return fon
    End Function
```

9. Una expresión aritmética construida con los operadores aritméticos binarios '+', '-', '*', '/' y operandos de un único digito entre 0 y 9, se dice que está en forma postfija si es o bien un sólo operando o dos expresiones en forma postfija una detrás de otra seguidas inmediatamente de un operador. A continuación se muestra una expresión escrita en la notación infija habitual junto con su forma postfija:

   Forma infija: (3 + 5/9) * (6 - 9)
   Forma postfija: 3 5 9 / + 6 9 - *

   - Diseña un algoritmo iterativo que calcule el valor de una expresión dada en forma posfija.
   - Diseña un algoritmo que dada una expresión en forma infija genere su versión posfija.

```
'Diseña un programa que permita pasar de una expresion en forma
infija, es decir,escrita usando parentesis, para evitar
ambigüedades, y con los operandos entre los operandos, a otra en
forma postfija ,esto es, sin parentesis y con los operadores
'tras los operandos

Imports ClasePila
Imports ClaseArbolesBinarios

Module Module1

    Public Function esOperador(ByVal ch As Char) As Boolean
        If ch = "+" Or ch = "-" Or ch = "*" Or ch = "/" Or ch =
"(" Or ch = ")" Then
            Return True
        Else : Return False
        End If
    End Function

    Public Function prioridad(ByVal ch As Char) As Integer
        Select Case ch
            Case "+"c, "-"c
                Return 1
            Case "*"c, "/"c
                Return 2
            Case Else
                Return 0
        End Select
    End Function

    Public Sub postfija(ByVal inf As String, ByRef post() As Char)
'Nos fijamos en que cuando usamos parentesis abiertos siempre se
apila Cada vez que se ve un operando se apila, si es un parentesis
abierto se apila y si es cerrado se desapila, hasta encontrar uno
abierto. Tenemos que tener en cuenta si el operador que leemos
```

```
tiene mas prioridad que el que esta en la cima de la pila en ese
caso vamos desapilandolos y escribiendolos, y finalmente
insertamos el que hemos leido

        Dim i, j As Integer
        Dim pilaux As tipoPila = New tipoPila()
        Dim ch As Char

        pilaux.apilar("#"c)
        For i = 0 To inf.Length - 1
            ch = inf.Chars(i)
            If Not esOperador(ch) Then
                Console.Write(ch)
                post(j) = ch
                j += 1
            ElseIf ch = "("c Then
                pilaux.apilar(ch)
            Else
                If ch = ")"c Then
                    While pilaux.cima() <> "("c
                        Console.Write(pilaux.cima)
                        post(j) = pilaux.cima
                        j += 1
                        pilaux.desapilar()
                    End While
                    pilaux.desapilar() 'Desapilamos el parentesis
abierto
                Else
                    While prioridad(pilaux.cima) >= prioridad(ch)
                        Console.Write(pilaux.cima)
                        post(j) = pilaux.cima
                        j += 1
                        pilaux.desapilar()
                    End While
                    pilaux.apilar(ch)
                End If

            End If
        Next
        While pilaux.cima <> "#"c
            Console.Write(pilaux.cima)
            post(j) = pilaux.cima
            j = +1
            pilaux.desapilar()
        End While
    End Sub

    Public Function evaluar(ByVal post() As Char) As Double
'Programa que evalúa una expresión en forma postfija, suponiendo
que los operandos que contiene son digitos entre 0 y  9

        Dim i, op1, op2 As Integer
        Dim pila1 As tipoPila = New tipoPila()
```

```
        While post(i) <> Nothing
            If Not esOperador(post(i)) Then
                pila1.apilar(Val(post(i)))
            Else
                op1 = pila1.cima
                pila1.desapilar()
                op2 = pila1.cima
                pila1.desapilar()
                Select Case post(i)
                    Case "*"c
                        pila1.apilar(op2 * op1)
                    Case "+"c
                        pila1.apilar(op2 + op1)
                    Case "-"c
                        pila1.apilar(op2 - op1)
                    Case "/"c
                        pila1.apilar(op2 / op1)
                End Select
            End If
            i += 1
        End While
        Return pila1.cima
    End Function
```

11. Una operación que genere un arbol que sea la imagen especular de otro dado, esto es, un árbol en el que los hijos izquierdos pasen a ser derechos y los derechos, izquierdos.

```
    Public Sub Especular()
        Dim aux As ArbolesBinarios

        If Not Me.vacio() Then
            aux = Me.hd
            Me.der = Me.izq
            Me.izq = aux
            Me.hd.Especular()
            Me.hi.Especular()
        End If
    End Sub
```

12. Se tiene una expresión guardada en un árbol binario de la siguiente forma: los operandos en las hojas y los operadores en los nodos internos. Diseña un procedimiento que lea los valores para los operandos de la expresión y la evalúe

```
        Public Function CrearArbolExpresion(ByVal cadena As
String)
            Dim pilaux As tipoPila = New tipoPila()
            Dim ch As Char
            Dim i As Integer
            Dim arbolaux As ArbolesBinarios

        Do
            ch = cadena.Chars(i)
            If Char.IsLetterOrDigit(ch) Then
                pilaux.apilar(New ArbolesBinarios(ch))
            ElseIf ch = "+" Or ch = "-" Or ch = "*" Or ch = "/"
Then
                Dim resultado, izq, der As ArbolesBinarios
                der = pilaux.cima
                pilaux.desapilar()
                izq = pilaux.cima
                pilaux.desapilar()
                resultado = New ArbolesBinarios(ch, izq, der)
                pilaux.apilar(resultado)
            End If
            i += 1
        Loop While ch <> Nothing
        Return pilaux
    End Function

    Public Sub evaluarArbolExpresion(ByVal arb As ArbolesBinarios,
ByRef valor As Double)
        'Esta funcion dada una expresión introducida en un arbol
de la siguiente forma los operandos
        'en las ramas y los operadores en las hojas la evalua

        Dim val1, val2 As Double
        Dim ch As Char

        If Not arb.hi.vacio Then
            evaluarArbolExpresion(arb.hi, val1)
            evaluarArbolExpresion(arb.hd, val2)
            ch = arb.raiz
            Select Case ch
                Case "+"
                    valor = val1 + val2
                Case "-"
                    valor = val1 - val2
                Case "/"
                    valor = val1 / val2
                Case "*"
```

```
                    valor = val1 * val2
                End Select
            Else
                valor = Val(arb.raiz)
            End If
        End Sub
```