Fundamentos del lenguaje de programación

Programación Orientada a Objetos

Tecnológico Kinal
Instructor Edwin Tumax

Visual Studio

OBJETO

El concepto de objeto puede extenderse prácticamente a cualquier cosa, tanto real como ficticia o imaginaria. En el contexto de la programación Orientada a Objetos, se refiere a algo que posea estado, comportamiento e identidad. Su estado y comportamiento se definen en una clase común. Por tanto, una clase podra tener muchos objetos de su tipo, pero sin embargo un objeto solo podrá pertenecer a una clase.

CLASE

Se trata de realizar una abstracción denominada clase, esta permite la agrupación de objetos que comparten las mismas propiedades y comportamiento. No se debe olvidar que la clase es una generalización de objeto.

INSTANCIA

Una instancia es en si un objeto que se clasifica dentro de una determinada clase, se le denomina instancia para resaltar el conceptos de creación en un momento dado, que tiene un tiempo de vida y que pertenece a algo, o sea: la instancia de una clase.

CONCEPTOS INTRÍNSECOS DE POO

- Abstracción.
- Encapsulamiento.
- Herencia.
- Polimorfismo.

ABSTRACCIÓN

- La abstracción es una forma de trabajar con la complejidad que nos impone el mundo real. Una abstracción denota las caraterísticas esenciales de un objeto que lo distinguen de todos los demás tipos de objeto y proporciona así fronteras conceptuales nítidamente definidas respecto a la perspectiva del observador. Nos permite separar el comportamiento de la implementación. Será más importante saber qué se hace, y no cómo se hace.

ABSTRACCIÓN

Ejemplo:

Un sensor de temperatura.

- Sabemos que:
- Mide la temperatura y nos la dice
- Se puede calibrar

No sabemos

- Como mide la temperatura.
- De que esta hecho....

ENCAPSULAMIENTO

Consiste en reunir estados y métodos dentro de un entorno limitado, en otras palabras realizar encapsulación.

Se trata de que ninguna parte de un sistema complejo dependa de los detalles internos de otra parte. Es por tanto complementario de la abstracción. La principal forma de conseguir este objetivo es mediante la ocultación de la información. Todos aquellos aspectos de un objeto que no contribuyen a sus características esenciales (estructura e implementación de sus métodos) queda escondido.

```
Public Class Estrella
    Private X_coord as Long
    Private Y_coord as Long

    Public Sub Show()
    ' ...
    End Sub
    Public Sub Hide()
    ' ...
    Ens Sub
End Class
```

Propiedades internas

Métodos y funciones

- Propiedades: Manifiestan el Estado del Objeto.
- Métodos y Funciones: Manifiestan el comportamiento.

HERENCIA

Es la relación más característica de la OOP. Expresa en general una especialización o una generalización de una clase sobre la otra. Sirve para evitar definir las características comunes a un conjunto de clases multiples veces. También se denomina relación "es un/a".

La clase de la cual se hereda recibe el nombre de clase base o superclase. La clase que hereda se llama clase derivada o subclase. A través de esta relación la subclase comparte la estructura y/o el comportamiento de la superclase (herencia simple) o superclases (herencia múltiple).

HERENCIA

- En Visual Basic .NET no existe el concepto de herencia múltiple como en C++.
- Este concepto ha sido revocado por los lenguajes modernos como Java, C# y VB.NET.
- Para declarar que una clase hereda de otra se especifica el atributo: Inherits

Figura

Método Mostrar

Método Ocultar

Figura Plana

Función Área.

Figura Volumétrica

Función Volumen.

Triángulo

Función Perímetro

Función Altura

Cuadrado

Intersección.

Esfera.

```
Class Figura

    Public Sub Mostrar()

    End Sub
    Public Sub Ocultar()
       ...
    End Sub

End Class


Class FiguraPlana
  Inherits Figura
    Public Function Area as Float
    End Function
End Class
```

LLAMADO A MÉTODOS DE CLASES PADRES

- Variable “Me”: Invoca un método de la clase en la que se encuentra.
- Variable “MyBase”: Invoca un método de la clase del padre.

```
Class Cuenta

    Public Overridable Function Saldo() as Float

    End Sub

End Class


Class CuentaInversion
  Inherits Cuenta
    Public Overrides Function Saldo() as Float
     Saldo = MyBase.Saldo() + Intereses
    End Function
End Class
```

MÉTODOS ABSTRACTOS

- Los métodos abstractos solo se pueden declarar en clases que han sido declaradas como abstractas.
- Se declaran en Visual Basic .NET con el modificador MustOverride.
- No contiene implementación, es solo una declaración.
- Expresan funcionalidad. (El qué hace?)

CLASES ABSTRACTAS

- Una clase abstracta es una clase que al menos posee un método abstracto.
- Las clases abstractas no se pueden instanciar.
- En Visual Basic .NET se declaran con el atributo MustInherit.
- Cualquier clase no abstracta que herede de una clase abstracta deberá implementar los métodos abstractos de su clase padre.
- Si una clase abstracta hereda de otra clase abstracta puede no implementar sus métodos abstractos y adicionar mas.
- Una clase abstracta puede heredar de una no abstracta y adicionar métodos abstractos.

```
Public MustInherit Class Figura
    Private X_coord as Long
    Private Y_coord as Long

    Public Sub MustOverride Show()

    Public Sub MustOverride Hide()

End Class
```

```
Public Class Linea

    Inherits Figura
    Public Sub Override Show()
        ´ Dibujar la linea.
        Drawing.lineTo(X_coord,Y_coord,white)
    End sub
    Public Sub Override Hide()
        ´ Ocultar la linea.
        Drawing.lineTo(X_coord,Y_coord,black)
    End sub

End Class
```

MODIFICADORES DE VISIBILIDAD DE MÉTODOS.

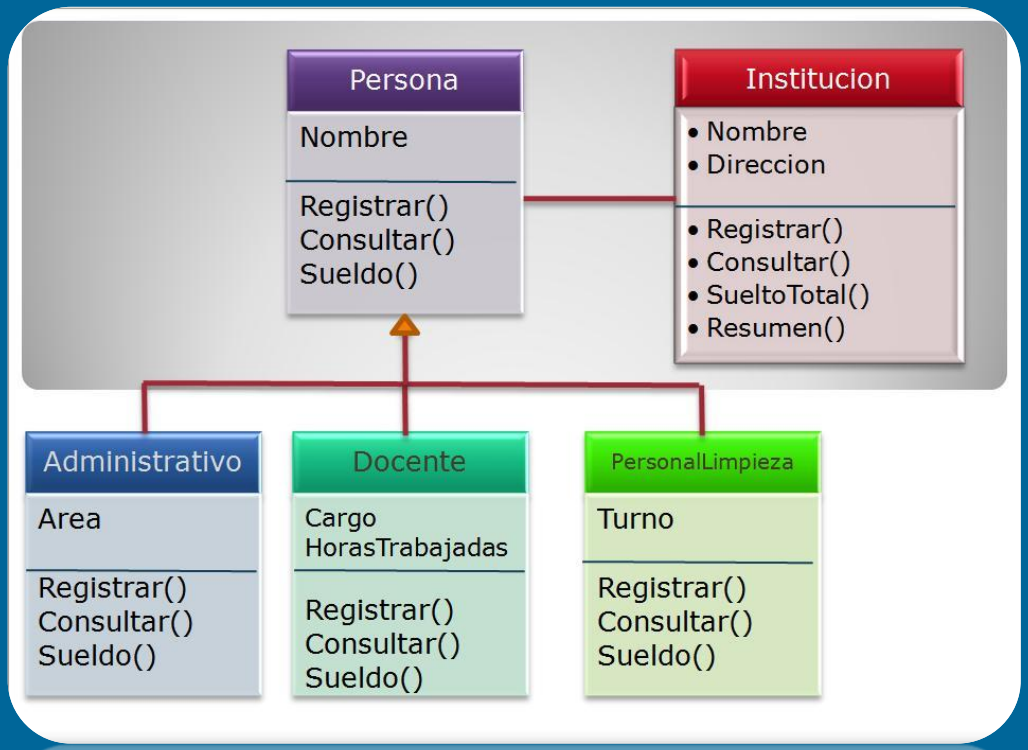
- Public: elementos que pueden ser accedidos desde cualquier contexto, dentro o fuera de la clase en que se han definido.
- Private: elementos que pueden ser accedidos solo desde dentro de la clase en que han sido definidos. Es lo considerado por defecto.
- Protected: elementos que pueden ser accedidos desde dentro de la clase en que es declarado y en las clases que deriven de la misma.
- Friend: elementos que pueden ser accedidos desde las clases que han sido definidas en el mismo paquete o proyecto.
- Protected Friend: elementos que solo pueden ser accedidos desde clases que deriven de esta y que estén definidas en el mismo proyecto.

MODIFICADORES DE VISIBILIDAD DE CLASES.

- ***Public:*** Es posible acceder a la clase desde cualquier ensamblado externo.
- ***Friend:*** Sólo es posible acceder a la clase desde el proyecto donde se declaró. *Es lo considerado por defecto.*

EJERCICIO

- Crear las siguientes clases en VB.NET



Fecha de entrega 9 de febrero de 2013

Visual Studio

CONSTRUCTORES

- Cuando se quiere trabajar con clases, es necesario crear objetos (instancias de las clases)
- Todas las clases tienen un constructor público por defecto sin parámetros, esto es debido a que todas heredan implicitamente de la clase System.Object.
- Para instanciar una clase es necesario invocar el constructor de la misma.
- El constructor de una clase en VB.NET se nombra como un Sub “New”

CONSTRUCTORES

```
Public Class Empleado
    Private Salario as Long
    Private Nombre as String
    Private UserName as String
    Private Password as String
  Public Sub New()
    Salario = 0
    Nombre = "Sin nombre"
    UserName = "anónimo"
    Password = "none"
  End Sub
  Public Function Login( aSystem as LoginSystem) as boolean
    Login = aSystem.Login(UserName & "sys", Password)
  End Function
End Class
```

```
Dim User1 as Empleado
User1 = New Empleado()
User1.Login(MySystem)
```

```
Dim User1 as New Empleado()
User1.Login(MySystem)
```

SOBRECARGA DE CONSTRUCTORES.

```
Public Class Empleado
    Private Salario as Long
    Private Nombre as String
    Private UserName as String
    Private Password as String
    Private GlobalCatalog as DBUsers
  Public Sub New()
    Salario = 0
    Nombre = "Sin nombre"
    UserName = "anónimo"
    Password = "none"
  End Sub
  Public Sub New(aUserName as String, aPassword as String)
    UserName = aUserName
    Password = aPassword
    GlobalCatalog.LookupUser(UserName, Password)
  End Sub
  Public Function Login(aSystem as LoginSystem)
    Login = aSystem.Login(UserName & "sys", Password)
  End Function
End Class
```

```
Dim User1 as New Empleado("guest", "123")
User1.Login(MySystem)
```

DESTRUCTORES

- Cuando una instancia se va a dejar de utilizar, debe destruirse para liberar los recursos que consume.
- En la plataforma .NET existe un proceso llamado Recolector de basura que mantiene una lista de las referencias creadas y referencias inválidas, cuando un proceso necesita memoria, el ambiente .NET verifica que hay memoria disponible para ese proceso, si no hay llama al recolector de basura para que libere los recursos no utilizados de todas las referencias inválidas.
- Dentro de la plataforma .NET no es necesario llamar el destructor de las clases, pero se pueden hacer operaciones antes de que la clase sea destruida por el Recolector de Basura.

FINALIZE METHOD

- Es invocado justo antes de que el Recolector de basura destruya la instancia de la clase.
- Puede ser de utilidad para realizar alguna acción antes de que el objeto sea destruido.
- Finalize es heredado de System.Object por lo que si se sobreescrbe debe declararse como mismo esta declarado el padre:

```
Protected Overrides Sub Finalize()
   ' clean up code goes here
End Sub
```

DISPOSE METHOD

- Objetos que hacen consumo exhaustivo de muchos recursos.
- Mantienen conexiones abiertas a bases de datos, archivos, etc.
- Es necerario liberar la memoria del objeto una vez que se deje de usar y no esperar a que el recolector de basura lo determine.
- Se crea un método en la clase llamado Dispose que cuando se invoca este método se liberan los recursos que consume este objeto.

```
Public Sub Dispose()
    ' clean up code goes here
End Sub
```

POLIMORFISMO

- Es una característica especial donde un identificador es declarado estáticamente de un tipo de clase y puede contener en tiempo de ejecución una instancia de otro tipo de clase derivada del tipo declarado.
- Cuando se invoca un método ***virtual*** a esta instancia, se invocará el método correcto según el tipo de objeto que esté dinámicamente instanciado.

POLIMORFISMO. MÉTODOS VIRTUALES

- Los métodos virtuales son declarados en Visual Basic .NET con el modificador Overridable.
- Los métodos abstractos siempre son virtuales.
- Para especificar que se quiere especializar o sobrescribir un método en una nueva clase se declara este en la clase hija de la misma forma que está declarado en la clase padre y se le coloca el modificador “Overrides”.

POLIMORFISMO, EJEMPLO:

```
MustInherit Public Class Shape
    Public MustOverride Sub Paint(g As Graphics, r As
Rectangle)
End Class

Public Class Ellipse
    Inherits Shape
    Public Overrides Sub Paint(g As Graphics, r As Rectangle)
        g.drawEllipse(r)
    End Sub
End Class

Public Class Box
    Inherits Shape
    Public Overrides Sub Paint(g As Graphics, r As Rectangle)
        g.drawRect(r)
    End Sub
End Class
```

POLIMORFISMO, EJEMPLO

```
Public Class ShapeList
  Private Shapes as ArrayList
  Public Sub AddShape(ByVal aShape as Shape)
    Shapes.Add(aShape)
  End sub
  Public Sub DrawAllShapes(g As Graphics, r As Rectangle)
    Dim i as Integer
    For i = 0 to Shapes.Count - 1
      Shapes(i).Paint(g,r)
    Next i
  End Sub
End Class

Dim Ellipse1 as New Ellipse()
Dim Box1 as New Box()
Dim Collection1 as New ShapeList()
Collection1.AddShape(Ellipse1)
Collection1.AddShape(Box1)
Collection1.DrawAllShapes(CurrentScreen, rect1)
```

EJERCICIO

- Definir una clase “Animal”
- Crear las clases “Animal carnívoro” y “Animal herbívoro” que hereden de la clase Animal.
- Definir las clases “vaca”, “perro”, “tiburón” que hereden la clase “Animal herbívoro” o “Animal carnívoro”.
- Clase Perro métodos “ladrar”, “caminar”, y “dormir”.
- Clase Tiburón métodos “nadar” y “dormir”.

SOBRECARGA DE MÉTODOS

- Los procedimientos funciones pueden declararse varias veces con el mismo nombre pero con diferentes parámetros.
- En tiempo de ejecución al invocar el método se hace el enlace con el método que corresponda según el tipo de parámetros que se estén pasando.

SOBRECARGA DE MÉTODOS

```
Public Class ShapeList
  Private Shapes as ArrayList
  Public Sub AddShape(ByVal aShape as Shape)
    Shapes.Add(aShape)
  End sub
  Public Overloads Sub DrawAllShapes(g As Graphics, r As
Rectangle)
    Dim i as Integer
    For i = 0 to Shapes.Count - 1
      Shapes(i).Paint(g,r)
    Next i
  End Sub
  Public Overloads Sub DrawAllShapes(r As Rectangle)
    Dim i as Integer
    For i = 0 to Shapes.Count - 1
      Shapes(i).Paint(defaultGraphics,r)
    Next i
  End Sub
End Class
```

INTERFACES.

- Una interface es un conjunto de definiciones de métodos y/o propiedades sin implementación.
- Las interfaces expresan funcionalidad (el que debe hacer un objeto?)
- La interface no es una clase abstracta aunque se le parezca.
- Las definiciones que se den de miembros de interfaces han de ser siempre públicas y no pueden incluir **override**, sí pueden dársele los modificadores como **Overridable** ó **MustOverride** y usar **Overrides** en redefiniciones que se les den en clases hijas de la clase que implemente la interfaz.

INTERFACES

```
Public Interface IShapeList
  Sub AddShape(ByVal aShape as Shape)
  Function CountShapes as Integer
  Sub ClearList()
  Sub Overloads DrawAllShapes(g As Graphics, r As Rectangle)
  Sub Overloads DrawAllShapes(r As Rectangle)
End Interface
```

Las clases pueden implementar una o varias interfaces al mismo tiempo y heredar de alguna clase. Por ejemplo:

```
Public Class ShapeList
  Implements IShapeList
  Sub AddShape(ByVal aShape as Shape) Implements IShapeList.AddShape
    Shapes.Add(aShape)
  End Sub
  ´ La clase debe implementar todos los métodos de la interface.
  ´ ...
End Interface
```

ELEMENTOS DE CLASE

- Elementos que pertenecen a la clase y no a la instancia que pueda crearse.
- No es necesario crear una instancia.
- No se puede acceder a un elemento (no se clase) desde un elemento de clase. A la inversa si.
- Los elementos de clase suelen llamarse estáticos o compartidos.
- En VB.net se identifican con el modificador Shared.

ELEMENTOS DE CLASE

```
Public Class Cuenta
  Public Shared Interes = 15
  Public Shared Function AplicarInteres(ByVal saldo as Float)
    AplicarInteres = saldo + (saldo * Interes / 100)
  End Function
  ´ ...

End Class
```

EVENTOS

- Eventos:

```
Public Class Employe
   Public Event EmployeEvent()
   Public Sub DoEvent()
     RaiseEvent EmployeEvent()
   End Sub
End Class
```

Asingnación Estática de Manejadores de eventos:

```
Public class Caller
   Public Employe1 WithEvents as new Employe()
   Protected Sub CallEvent() Handles Employe1.EmployeEvent
   End Sub
End class
```

EVENTOS, ASIGNACIÓN DINÁMICA

```
Public Class Employe
  Public Event EmployeEvent()
  Public Sub DoEvent()
    RaiseEvent EmployeEvent()
  End Sub
End Class

Public class Caller
  Public Employe1 WithEvents as new Employe()
  Protected Sub CallEvent()
    ´ Do someting
  End Sub
End class

Dim Caller1 as new Caller()
Dim Employe1 as new Employe()

AddHandler Employe1.EmployeEvent, AddressOf Caller1.CallEvent
```

DELEGADOS

```
Public Delegate Sub EmployeEventHandler(ByVal nState As Integer)

Public Delegate Function EmployeEventHandler(ByVal nState As EmployeState)
as boolean

Public Class Employe
   Public Event EmployeEvent as EmployeEventHandler

   Public Sub DoEvent()
     Result = EmployeEvent.Invoke(1);
   End Sub
End Class
```

- Asignacion de delegados:

```
Public Class Caller

   Public Sub CallEvent()
     Dim emp as new Employe
     AddHandler(emp.EmployEvent, new EmployeEventHandler(AddressOf Manager))
   End Sub

   Public Function Manager(ByVal nState As EmployeState) as boolean

   End Function
End Class
```

PROPIEDADES

```
Public class Persona
  Private Anio_de_nacimiento as integer
  Public Property Edad() as integer
    Get
      Return System.Now.GetYear() - Anio_de_nacimiento
    End Get
    Set (ByVal Value as Integer)
      Anio_de_nacimiento = System.Now.GetYear() - Value
    End Set
  End property
End Class
```

ESTRUCTURAS

```
'Structure to hold and manage a complex number
  Public Structure Complex
     Public Real As Double
     Public Imag As Double ' "Imaginary" part (coefficient of "i")
     ' Add another complex number to this one:
     Public Function Plus(ByVal Operand As Complex) As Complex
       Plus.Real = Operand.Real + Real
       Plus.Imag = Operand.Imag + Imag
     End Function
   ' Multiply this complex number by another one:
     Public Function Times(ByVal Operand As Complex) As Complex
       Times.Real = (Operand.Real * Real) - (Operand.Imag * Imag)
       Times.Imag = (Operand.Imag * Real) + (Operand.Real * Imag)
     End Function
   ' Invert this complex number:
     Public Function Reciprocal() As Complex
       Dim Denominator As Double = (Real * Real) + (Imag * Imag)
       If Denominator = 0 Then Throw New System.DivideByZeroException()
       Reciprocal.Real = Real / Denominator
       Reciprocal.Imag = -Imag / Denominator
     End Function
  End Structure ' Complex
```

VB → VB.NET

- Variant → Object
- Option Explicit es por defecto ON
- Set y Let no son soportados.
- Uso de paréntesis en llamadas a métodos.
- No hay propiedades por defecto.