Introducción a la programación orientada a objetos

Clases y objetos

Una clase es el código fuente que definen las características y el comportamiento de un objeto.
Un objeto es una instancia específica de dicha clase en tiempo de ejecución.
Las características o atributos de un objetos se codifican en .NET a través de propiedades.
El comportamiento a través de métodos, es decir, procedimientos o funciones.

Creación de clases

C#

```
class Producto
{
    //código de la clase
}
```

VB.NET

```
Public Class Producto

End Class
```

Instanciación de objetos

C#

```
Producto objProducto1 = new Producto();
Producto objProdcuto2 = new Producto();
```

VB.NET

```
Dim objProducto1 As New Producto
Dim objProducto2 As New Producto
```

Atributos de una clase

Los atributos de una clase siempre deben encapsularse, es decir, debe impedirse el acceso directo a lo mismos. Para ello se utilizan métodos, tanto para guardar como para recuperar diocho valor. En .NET se ha implementado esta estructura a través de propiedades.

Propiedades

Una propiedad es, en esencia, una variable privada del objeto al cual se accede a través de los métodos get y set.

El método get se ejecuta cuando se recupera el valor del atributo.
El método set se ejecuta cuando se guarda el valor del atributo.

C#

```
public class Alumno
{
    private string _Nombre = string.Empty;
    private string _Apellido = string.Empty;

    public string Nombre
    {
        get
        {
            return _Nombre;
        }
        set
        {
            _Nombre = value;
        }
    }

    public string Apellido
    {
        get
        {
            return _Apellido;
        }
        set
        {
            _Apellido = value;
        }
    }
}
```

VB.NET

```
Public Class Alumno

    Private _Nombre As String = String.Empty
    Private _Apellido As String = String.Empty

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

    Public Property Apellido As String
        Get
            Return _Apellido
        End Get
        Set(ByVal value As String)
            _Apellido = value
        End Set
    End Property

End Class
```

Dentro de cada método get/set es posible implementar la lógica que se desee.
Por ejemplo, en este ejemplo el atributo legajo se guarda suprimiendo los espacios que pudieran existir en ellos extremos izquierdo y derecho de value.
Asimismo, el atributo se devuelve siempre en minúsculas.

C#

```
public class Empleado
{
    private string _Legajo = string.Empty;

    public string Legajo
    {
        get
        {
            return _Legajo.ToLower();
        }
        set
        {
            _Legajo = value.Trim();
        }
    }

}
```

VB.NET

```vb
Public Class Empleado

    Private _Legajo As String = String.Empty

    Public Property Legajo As String
        Get
            Return _Legajo.ToLower
        End Get
        Set(ByVal value As String)
            _Legajo = value.Trim
        End Set
    End Property

End Class
```

Propiedades autoimplementadas

Cuando no se va a realizar ninguna operación en los métodos get o set se puede abreviar la codificación de la propiedad utilizando la forma autoimplementada.

C#

```csharp
public class Pais
{
    public string Nombre { get; set; }
    public int Poblacion { get; set; }
}
```

VB.NET

```vb
Public Class Pais

    Public Property Nombre As String
    Public Property Poblacion As Integer

End Class
```

Propiedades de sólo lectura

Las propiedades pueden ser de sólo lectura. Para ello se codifica exclusivamente el método get. Este tipo de propiedades frecuentemente se calculan a partir de otras propiedades y/o métodos.

C#

```
public class Persona
{
    public string Nombre { get; set; }
    public string Apellido { get; set; }
    public DateTime FechaNacimiento { get; set; }
    public int Edad
    {
        get
        {
            int _Edad = 0;
            _Edad=  DateTime.Now.Year - this.FechaNacimiento.Year;
            if (this.FechaNacimiento > DateTime.Now.AddYears(-_Edad))
            {
                _Edad = _Edad - 1;
            }
            return _Edad;
        }
    }
}
```

VB.NET

```
Public Class Persona

    Public Property Nombre As String
    Public Property Apellido As String
    Public Property FechaNacimiento As Date

    Public ReadOnly Property Edad As Integer
        Get
            Dim _Edad As Integer = 0
            _Edad = Date.Now.Year - Me.FechaNacimiento.Year
            If Me.FechaNacimiento > Date.Now.AddYears(-_Edad) Then
                _Edad = _Edad - 1
            End If
            Return _Edad
        End Get
    End Property

End Class
```

Propiedades de sólo escritura

Asimismo, pueden ser de sólo escritura. Para ello se codifica exclusivamente el método set. Este tipo de propiedad no es muy utilizado.

C#

```csharp
public class Usuario
{
    private string _Contraseña = string.Empty;
    public string Contraseña
    {
        set
        {
            _Contraseña = value;
            // Aqui podría invocarse a una función para validar la contraseña.
            // De cualquier forma es mejor reemplazar en este caso la
            //propiedad de sólo lectura por un método qyue reciba como parámetro
            //la contraseña.
        }
    }

}
```

VB.NET

```vbnet
Public Class Usuario

    Private _Contraseña As String = String.Empty

    Public WriteOnly Property Contraseña As String
        Set(ByVal value As String)
            _Contraseña = value
            ' Aqui podría invocarse a una función para validar la contraseña.
            ' De cualquier forma es mejor reemplazar en este caso la
            'propiedad de sólo lectura por un método qyue reciba como parámetro
            'la contraseña.
        End Set
    End Property

End Class
```

Modificadores de visibilidad

C#	VB.NET	Descripción
public	Public	Acceso irrestricto.
protected	Protected	Visible sólo desde la clase o sus derivadas.
internal	Friend	Acceso limitado al proyecto.
protected internal	Protected Friend	Visible sólo desde la clase, sus derivadas y el proyecto.
private	Private	Acceso limitado a la clase.

Los modificadores public/Public y private/Private son los únicos que se han utilizado hasta el momento.
Para más información consultar:

- C#: http://msdn.microsoft.com/en-us/library/wxh6fsc7(v=vs.100).aspx
- VB.NET: http://msdn.microsoft.com/en-us/library/76453kax(v=vs.100).aspx

Modificadores this/Me

Los modificadores this/me significan "ésta instancia de la clase". Su uso es opcional.

Sobrecarga de métodos

La sobrecarga de métodos es una operación muy frecuente en programación a objetos. Consiste en definir varias veces un mismo método cambiando la firma de sus parámetros.

Firma del parámetro

La firma del parámetro se puede definir como el tipo, la posición y la cantidad de pàrámetros en un método. La sobrecarga es posible mientras no se repita dicha firma.

El nombre de los parámetros no tiene relevancia para a la firma.

C#

```
public class Ventas
{
    public void Listar()
    {
        //...
    }
    public void Listar(string paramSucursal)
    {
        //...
    }
    public void Listar(int paramVendedor)
    {
        //...
    }
    public void Listar(string paramSucursal, int paramVendedor)
    {
        //...
    }
    public void Listar(string paramSucursal, int paramVendedor,
                       DateTime paramFechaInicio)
    {
        //...
    }
}
```

VB.NET

```vb
Public Class Ventas

    Public Sub Listar()
        '...
    End Sub

    Public Sub Listar(ByVal paramSucursal As String)
        '...
    End Sub

    Public Sub Listar(ByVal paramVendedor As Integer)
        '...
    End Sub

    Public Sub Listar(ByVal paramSucursal As String, ByVal paramVendedor As Integer)
        '...
    End Sub

    Public Sub Listar(ByVal paramSucursal As String, ByVal paramVendedor As Integer,
                      ByVal paramFechaInicio As Date)
        '...
    End Sub

End Class
```

Constructores

El constructor es un método que se ejecuta cuando se instancia una clase.

Constructor por defecto

El constructor, por defecto, no tiene parámetros ni instrucción alguna en el cuerpo del método.
Si se desea cambiar el comportamiento del constructor por defecto, el mismo debe declararse en forma explícita.

C#

```csharp
public class Direccion
{
    public Direccion()
    {
        //...
    }
}
```

VB.NET

```
Public Class Direccion

    Public Sub New()
        '...
    End Sub

End Class
```

Constructores sobrecargados

En diversas ocasiones es útil sobrecargar el contructor por defecto, por ejemplo, a fin de inicializar los atributos de una clase.

C#

```
public class Direccion
{
    public Direccion()
    {
        //...
    }
    public Direccion (string pCalle, string pAltura, string pCodigo)
    {
        this.Calle = pCalle;
        this.Altura = pAltura;
        this.Codigo = pCodigo;
    }

    public string Calle { get; set; }
    public string Altura { get; set; }
    public string Codigo { get; set; }
}
```

VB.NET

```
Public Class Direccion

    Public Sub New()
        '...
    End Sub

    Public Sub New(ByVal pCalle As String, ByVal pAltura As String,
                   ByVal pCodigo As String)
        Me.Calle = pCalle
        Me.Altura = pAltura
        Me.Codigo = pCodigo
    End Sub

    Public Property Calle As String
    Public Property Altura As String
    Public Property Codigo As String

End Class
```

Si se desea eliminar el contructor por defecto, debe sobrecargarse y al mismo tiempo se debe eliminar la declaración del constructor por defecto, si es que se ha hecho explícito.

C#

```
public class Circulo
{
    public Circulo (double pRadio)
    {
        this.Radio = pRadio;
    }

    double Radio { get; set; }
    double Superficie
    {
        get
        {
            return Math.PI * Math.Pow(this.Radio, 2);
        }
    }
}
```

VB.NET

```
Public Class Circulo

    Public Sub New(ByVal pRadio As Double)
        Me.Radio = pRadio
    End Sub

    Private Property Radio As Double

    Private ReadOnly Property Superficie As Double
        Get
            Return Math.PI * Math.Pow(Me.Radio, 2)
        End Get
    End Property
End Class
```