Unidad 2

DIPLOMATURA EN PROGRAMACION JAVA

Capítulo 2

Clases y Objetos en .Net

Clases y Objetos en .Net

Variables

Las variables en un lenguaje de programación es el medio por el cual se almacenan valores transitoriamente en un programa. Particularmente, en un lenguaje de programación orientado a objetos como los de .Net, las variables son directamente asociadas a los atributos de las clases, aunque se utilicen en otros lugares que no sean directamente atributos, como los parámetros de un método. Por lo tanto en estos lenguajes hablar de variables o atributos es lo mismo salvo que estén en un método.

Las variables se diferencian en un programa por sus nombres y su tipo. Estos siguen ciertas reglas de escritura a las que están circunscriptas los identificadores. La razón de esto es evitar ambigüedades entre posibles operadores e instrucciones y los nombres de los identificadores.

En .Net se puede hacer una primera clasificación de las variables según dos grupos:

- Tipos primitivos
- Tipos Referenciados

La principal diferencia entre estos dos grupos es que el primero son los tipos de datos prexistentes definidos en el lenguaje mientras que los segundos son aquellos que pueden almacenar las referencias que se obtienen al crear un objeto de cualquier clase que se haya definido.

Propiedades de las variables

Las variables en .Net poseen propiedades que permiten su control y manejo a lo largo de un programa. Las propiedades de una variable son las siguientes:

- Son espacios en memoria que almacenan datos
- Siempre tienen almacenado un valor desde su declaración
- Siempre deben ser declaradas por un tipo primitivo o referenciado
- Poseen visibilidad y alcance (lugares desde donde se pueden acceder y lugares desde los cuales no)
- Se pueden utilizar como parámetros de los métodos
- Se puede retornar el valor que almacenen desde un método

El manejo de estas propiedades es lo que garantiza el buen uso de las mismas, por lo tanto, si se puede asociar una variable con los atributos de una clase, es fundamental dominar el concepto para saber dónde utilizarlas y como.

Identificadores

Hay una regla muy simple para determinar si algo es un identificador:

> *Cuando en el código un programador debe decidir que nombre ponerle a un elemento, dicho elemento es un identificador*

Dentro de esta característica entran varios de los elementos antes mencionados, como ser, nombres de clases, variables, métodos, etc...

Hay otros elementos de .Net todavía no mencionados que son identificadores. Estos se irán descubriendo como tales a medida que se aprenda más del tema siguiendo la simple regla que se formuló.

Es preciso aclarar que cualquier palabra que el lenguaje defina como **reservada** o **clave**, no podrá ser utilizada como identificador.

Por otro lado, crear un identificador es asignarle un nombre a un elemento que permite definir un lenguaje de programación. Esta asignación de nombres debe seguir ciertas normas prestablecidas en el formato del mismo que variarán de lenguaje a lenguaje. En el caso de .Net, las reglas a seguir son las siguientes:

- El primer carácter de un identificador debe ser uno de los siguientes:
  - ✓ Una letra en mayúsculas (A~Z)
  - ✓ Una letra en minúsculas (a~z)
  - ✓ El carácter de subrayado (_)
- Del segundo carácter en adelante:
  - ✓ Cualquier elemento de los que sirve para el primer caracter
  - ✓ Caracteres numéricos (0~9)

Vale la pena mencionar que el espacio en blanco no es un carácter permitido, por lo tanto no debe utilizarse para nombrar un identificador.

Además existen ciertas normas estandarizadas para crear identificadores, que si bien no son obligatorias, han sido adoptadas por la mayoría de los programadores y seguirlas ayudan mucho a la legibilidad del código. Algunas de ellas son las siguientes

- Si es una clase, el nombre debe comenzar con mayúsculas
- Si es un método o una propiedad, el nombre debe comenzar con mayúsculas
- Las variables empiezan con minúsculas
- Si el nombre tiene más de una palabra, a partir de la segunda palabra separarlas comenzando con mayúsculas la primera letra de cada palabra a partir de la segunda que pertenece al identificador

No se debe olvidar el hecho que en .Net, C# a diferencia de Visual Basic, es un lenguaje sensible al caso (diferencia entre mayúsculas y minúsculas), por lo tanto si dos identificadores son iguales en su significado a la lectura pero se diferencia tan sólo en el caso de una letra, el lenguaje los considerará identificadores distintos.

Tipos primitivos y referenciados

Todo elemento que tenga las propiedades de una variable es un tipo. Por ejemplo, un objeto, que es un tipo referenciado, tiene las mismas propiedades de una variable.

Nota: los tipos primitivos en .Net exigen un análisis más profundo que excede el alcance del curso

Los tipos primitivos de .Net permiten clasificarlos según su uso en los siguientes grupos.

C#

- Tipos enteros
  - `sbyte`
  - `byte`
  - `short`
  - `ushort`
  - `int`
  - `uint`
  - `long`
  - `ulong`
- Tipos de punto flotante
  - `float`
  - `double`
  - `decimal`
- Tipo texto
  - `char`
  - `string`
- Tipo lógico
  - `bool`

VB

- Tipos enteros
  - `SByte`
  - `Byte`
  - `Short`
  - `UShort`
  - `Integer`
  - `UInteger`
  - `Long`
  - `ULong`
- Tipos de punto flotante
  - `Single`
  - `Double`
  - `Decimal`
- Tipo texto
  - `Char`

- `String`

➢ Tipo lógico
- `Boolean`

Las declaraciones de las variables en las clases se realizan con el siguiente formato UML:

[<modificador>] <*tipo primitivo*> <*identificador*> *[= valor inicial];*

En otras palabras:

➢ Si aparece "<>", quiere decir "elegir uno entre los posibles"
➢ Si aparece "[]", quiere decir que es opcional
➢ Si una palabra o símbolo aparece sin ninguna otra cosa, indica que ponerlo es obligatorio

Este formato debe leerse de la siguiente manera:

➢ **Modificador** (opcional): Es el modificador de visibilidad de la variable (indica como se lo puede utilizar y desde donde). Las posibilidades son:

C#

Miembros de	Accesibilidad predeterminada	Accesibilidades declaradas permitidas
enum	public	Ninguna
class	private	public
		protected
		internal
		private
		protected internal
interface	public	Ninguna
struct	private	public
		internal
		private

VB

Miembros de	Accesibilidad predeterminada	Accesibilidades declaradas permitidas
Enum	Public	Ninguna
Class	Public	Public
Module		Protected

		Friend
		Private
		Protected Friend
Interface	Public	Ninguna
Structure	Siempre hay que indicar el ámbito	Public
		Friend
		Private

- **Tipo**: Es obligatorio poner un tipo, pero se debe elegir uno de los posibles
- **Identificador**: Es obligatorio poner un identificador, se debe elegir cual nombre poner
- **Asignación inicial**: Existe la opción de elegir poner un "=" y un literal de asignación si se desea que la variable tenga un valor inicial
- **Fin de la declaración**: Siempre hay que finalizar la declaración con un ";" en C# que indica fin de línea de programa. En el caso de VB, cada línea se considera una instrucción y no existe un carácter especial para indicar el fin de línea. Si se desea utilizar más de una línea para una única instrucción, se debe poner separado por un caracter en blanco el símbolo de subrayado.

Ejemplos

C#

```
private int nroEmpleados = 10;
private int edad=20;
private float sueldoFijo;
private long documento;
```

VB

```
Private Dim nroEmpleados As Integer = 10
Private Dim edad As Integer = 20
Private Dim sueldoFijo As Single
Private Dim documento As Long
```

Nota: Cuando a una variable se le asigna un valor numérico dentro del código, dicho número se lo identifica como "literal de asignación" por ser caracteres escritos en un editor de texto. El compilador del lenguaje toma dichos caracteres y los convierte al valor binario que debe almacenar en la memoria para la variable con la mencionada asignación.

Tipos referenciados

Cada vez que se crea una clase se define un nuevo tipo, pero para poder hacer uso de él, se debe crear un objeto. Esto es similar a declarar una variable de un tipo primitivo, con la diferencia que la memoria requerida para la operación será determinada por los elementos declarados en dicha clase.

Por otro lado, se utiliza la definición de la clase como plantilla de la memoria a utilizar y en base a esto se realiza la creación del objeto.

En .Net cuando se declara un objeto se utiliza un operador diseñado para este fin: `new` en C# y `New` en VB. Este operador es quien toma la clase como molde y reserva la memoria para la creación del objeto.

Sin entrar en detalles acerca de la manera en que esto se realiza, el operador efectúa las operaciones necesarias en memoria y luego devuelve el valor de la dirección o referencia del lugar donde lo hizo, de ahí el nombre de las declaraciones como **tipo referenciados**.

Evidentemente el valor devuelto debe almacenarse en algún lugar para su posterior uso y ese lugar es una variable de tipo referencia. Como este tipo de variables almacenan la dirección de memoria donde se creo el objeto y como toda clase genera un nuevo tipo, es evidente que la declaración debe efectuarse con la clase que la define.

De esta manera, con el nombre de la clase seguido del identificador elegido se crea la variable referencia. Cuando esta deba almacenar el valor devuelto por el operador `new`, al identificador lo seguirá un operador de asignación "=", el operador `new` y por último en nombre de la clase que indica el tipo de objeto a crear.

El formato en UML es el siguiente:

C#

<nombre del tipo clase> <identificador>[= `new` nombre del tipo clase ()];

VB

`Dim` *<identificador>* `As` *<nombre del tipo clase> [= `New` nombre del tipo clase ()];*

Al igual que las variables, se pueden crear tipos referenciados sin necesidad de inicializarlos con un valor (referencia al objeto creado por `new`), pero se debe usar el operador posteriormente para crear un objeto y utilizarlo.

Ejemplo

C#

```
Empleado e = new Empleado();
Persona p = new Persona();
Persona p2;
```

VB

```
Dim e As Empleado = New Empleado()
Dim p As Persona = New Persona()
Dim p2 As Persona
```

Comentarios

En .Net, dentro del código, existen dos formas de poner comentarios en C# y una en VB. Tanto en C# como en VB se admiten comentarios de una línea, pero sólo el primero admite cometarios de varias líneas a la vez sin marcar de una en una.

C#

```
// La doble barra se usa si el comentario abarca una línea de
// comentario

/* Comienza el comentario
   La combinación barra – asterisco para comenzar y asterisco - barra
   para finalizar se utiliza el comentario abarca más de una línea
   Finaliza el comentario
 */
```

VB

```
' La comilla simple se usa si el comentario abarca una línea de
' comentario
```

Explorando los tipos primitivos

Tipos entero

Los tipos enteros se diferencian en su capacidad de almacenamiento como muestran las siguientes tablas.

C#

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
sbyte	System.SByte	1 byte	-128 a 127
byte	System.Byte	1 byte	0 a 255
short	System.Int16	2 bytes	-32,768 a 32,767
ushort	System.UInt16	2 bytes	0 a 65,535
int	System.Int32	4 bytes	-2,147,483,648 to 2,147,483,647
uint	System.UInt32	4 bytes	0 to 4,294,967,295
long	System.Int64	8 bytes	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
ulong	System.UInt64	8 bytes	0 a 18446744073709551615

VB

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
Byte	Byte	1 byte	0 a 255
Integer	Int32	4 bytes	-2.147.483.648 a 2.147.483.647
Long	Int64	8 bytes	-9223372036854775808 a 9223372036854775807

SByte	SByte	1 byte	-128 a 127 (con signo)
Short	Int16	2 bytes	-32.768 a 32.767 (con signo)
UInteger	UInt32	4 bytes	0 a 4.294.967.295 (sin signo)
ULong	UInt64	8 bytes	0 a 1844674407370955999
UShort	UInt16	2 bytes	0 a 65.535 (sin signo)

Ejemplo

C#

```
int i
int i = 5
```

VB

```
Dim i As Integer = 5
Dim j As Integer = 5
```

En el caso de C#, por defecto, se convierte a todo literal de asignación en un tipo entero `int`, por lo tanto cuando se asigna un literal a otro tipo entero diferente se debe especificar una conversión de tipo con un modificador (cast), como se indica a continuación.

C#

```
long p = 10L;
long n = (long)10;
short s = (short)5;
byte b = (byte)1;
```

Las formas utilizadas para p y n son análogas. La letra "L" utilizada en la asignación a la variable p es un camino corto que provee el lenguaje. El lenguaje permite la omisión de las conversiones de tipo en los primeros dos ejemplos debido a que `long` es de mayor tamaño que `int`. A esto se lo denomina **promoción automática**.

Sin embargo, VB no tiene el mismo comportamiento. Las variables simplemente se asignan y si la asignación es posible, se realiza. Sólo en algunos es necesaria la conversión de tipos y se realiza mediante funciones específicas diseñadas para tal fin.

VB

```
Dim p As Long = 10
Dim n As Long = 10
Dim s As Short = 5
Dim b As Byte = 1
```

Tipos punto flotante

Existen tres tipos primitivos de punto flotante y se diferencian por su capacidad de almacenamiento. Las variables de punto flotante en .Net pueden utilizar notación científica. Las siguientes tablas muestran los tipos.

C#

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
decimal	System.Decimal	16 bytes	$\pm 1.0 \times 10^{-28}$ a $\pm 7.9 \times 10^{28}$
float	System.Single	4 bytes	±1.5e−45 a ±3.4e38
double	System.Double	8 bytes	±5.0e−324 a ±1.7e308

VB

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
Decimal	Decimal	16 bytes	-7.9E+28 a 7.9E+28
Double	Double	8 bytes	±4.94065645841247E-324 a ±1.7E+308
Single	Single	4 bytes	-3.4E+38 a 3.4E+38

En C#, .Net toma la asignación de un literal numérico en punto flotante por defecto como un `double`, por lo tanto si es un `float` hay que especificarlo de una de las siguientes formas:

```
float f = 10.5F;
float g = (float)10.5;
```

Al igual que con los enteros, en VB este comportamiento no existe

```
Dim f As Single = 10.5F
Dim g As Single = 10.5
```

Tipo Texto

Este tipo se utiliza para almacenar un solo carácter. Tiene la capacidad de almacenar cualquier carácter en formato Unicode (formato estándar de cualquier carácter en .Net).

Se debe notar que sólo almacena un carácter, para almacenar una palabra o tan sólo más de un carácter, se debe utilizar otro tipo de datos que provee el lenguaje, el tipo `String`.

Otro punto importante es que cuando se asigna en el código un literal carácter, este debe ir rodeado de comillas simples en C# y comillas dobles en VB, como se muestra a continuación.

C#

```
char c = 'a';
```

VB

```
Dim c As Char = "a"
```

Las siguientes tablas muestran las diferentes posibilidades

C#

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
string	String	0 a 2.000 millones de caracteres Unicode	U+0000 a U+ffff por caracter
char	System.Char	2 bytes	U+0000 a U+ffff

VB

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
String	String	0 a 2.000 millones de caracteres Unicode	U+0000 a U+ffff por caracter
Char	Char	2 bytes	0 a 65535 (sin signo)

Tipo booleano

Este tipo sólo puede almacenar dos valores: `true` o `false` en C# y `True` o `False` en VB (que son literales que internamente se convierte en 0 y 1 respectivamente).
Los valores que almacenan son considerados palabras clave en el lenguaje y se deben utilizar para cualquier asignación a variables de este tipo. Generalmente es utilizado para la toma de decisiones.

Ejemplo

C#

```
bool varBool1 = true;
```

VB

```
Dim varBool1 As Boolean = True
```

C#

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
`bool`	System.Boolean	Según la plataforma	`true` o `false`

VB

Tipo	Estructura en el CLR	Asignación de memoria	Intervalo de valores
`Boolean`	Boolean	Según la plataforma	`True` o `False`

Métodos en .Net

Los métodos o, como también se los llama, “funciones”, son los elementos de una clase mediante los cuales se define una operación que un objeto de su tipo puede realizar. La declaración de un

método en .Net siempre se debe realizar dentro de una clase. En el caso de VB, los subprocesos (que se diferencian de los métodos por no definir un valor de retorno) se utilizan igual que dichos métodos. Estos pueden adoptar el siguiente formato:

C#

```
[<modificador>] <tipo retornado> <identificador>([lista de argumentos])
{
        [declaraciones y / o métodos a utilizar]
        [return [valor retornado] ;]
}
```

VB

```
[<modificador>] Sub <identificador> ( [lista de argumentos] )
                [declaraciones y / o métodos a utilizar]
End Sub

[<modificador>] Function <identificador>([lista de argumentos]) As <tipo>
        [declaraciones y / o métodos a utilizar]
        [Return [valor retornado]]
End Function
```

El formato quiere decir lo siguiente:

- Modificador: es el modificador de visibilidad del método (indica como se lo puede utilizar y desde donde). Las posibilidades se muestran en la siguiente tabla

VB	C#	Descripción del ámbito
Private	private	Accesible dentro del mismo módulo, clase o estructura.
Friend	internal	Accesible desde dentro del mismo proyecto, pero no desde fuera de él. **Valor por defecto**.
Protected	protected	Accesible desde dentro de la misma clase o desde una clase derivada de ella.
Protected Friend	protected internal	Accesible desde clases derivadas o desde dentro del mismo proyecto, o ambos.
Public	public	Accesible desde cualquier parte del mismo proyecto, desde otros proyectos que hagan referencia al proyecto, y desde un ensamblado generado a partir del proyecto.

- **Tipo retornado**: se debe elegir entre un tipo primitivo o uno referenciado. Si el método no devuelve ningún valor se puede indicar poniendo en este lugar la palabra clave `void`. No aplicable a los subprocesos de Visual Basic.
- **Identificador**: es el nombre que se le asignará al método y mediante el cual se lo invocará
- **Lista de argumentos**: en este lugar se indican los parámetros que recibirá el método. Los argumentos se declaran igual que las declaraciones de variables con la diferencia que no se pone el ";" final. Si el método posee varios argumentos, se deben separa unos de otros

con el operador de continuación de declaración (“,”). Si el método no recibe argumentos, se dejan sólo los paréntesis del mismo.

- **Comienzo del bloque de sentencias**: Obligatoriamente poner la llave de apertura de bloque de sentencias. No aplicable a Visual Basic.
- **Sentencias**: Opcionalmente agregar declaraciones e invocaciones a métodos. Existe la posibilidad de no poner nada a lo que se llama método de cuerpo vacío.
  Fin de ejecución de sentencias del método: Opcionalmente poner una sentencia `return` para terminar la ejecución del método. Si se indico que el método retorna un valor (recordar que puede ser `void`), este se debe poner a continuación en el caso de ser una función.
- **Fin del bloque de sentencias**: Poner obligatoriamente la llave de cierre en C# y la instrucción de fin de función o subproceso en Visual Basic.

Ejemplo

CS

```
public int calculaVolumen(int ancho, int largo, int alto){
            //bloque de sentencias
}
```

VB

```
Sub ProcesarInformacionDelDns(ByVal resultado As IAsyncResult)

End Sub


Public Function ProcesarInfo(ByVal a As Integer)

End Function
```

El método es público, devuelve un entero y recibe tres enteros como parámetros. El subproceso no retorna valor alguno.

Clases en .Net

Tienen los siguientes formatos según el lenguaje.

C#

```
<modificador> class <identificador>
{
        [declaraciones de variables, propiedades y métodos]
}
```

VB

```
<modificador> Class <identificador>
        [declaraciones de variables, subprocesos, propiedades y métodos]
End Class
```

Ejemplo

C#

```
public class Ejemplo
{
        private int var1;
        private int var2;

        public int Var1
        {
                get { return var1; }
                set { var1 = value; }
        }

        public int Metodo()
        {
                return 20;
        }
}
```

VB

```
Public Class Ejemplo
    Private _var1 As Integer
    Private _var2 As Integer

    Public Property Var1() As Integer
        Get
            Return _var1
        End Get
        Set(ByVal value As Integer)
            _var1 = value
        End Set
    End Property

    Public Function Metodo() As Integer
        Return 20
    End Function
End Class
```


No se debe olvidar que cuando se declara una clase se crea un nuevo tipo. En este caso dicho tipo es Ejemplo. Las clases son sólo moldes o plantillas para los objetos de su tipo que se creen. Se puede apreciar en ella la interfaz de la clase declarada con el modificador `public` o `Public` dependiendo del lenguaje

Constructores

Las clases tienen un método especial que se denomina constructor. Cuando se crea un objeto mediante la creación de una instancia de una clase, luego que se reserva en memoria los lugares de almacenamiento necesarios para el objeto, se ejecuta siempre el constructor.

Cómo es un método especial se diferencia de los otros por las siguientes dos características:

- Se llaman igual que la clase en C# y en `New` VB
- Nunca se le pone el valor retornado

Si la clase no posee uno, se ejecuta uno por defecto que el lenguaje provee más allá que no se especifique en el código.

Además, si se lo desea, se puede poner más de un constructor. La existencia de más de un constructor tiene su explicación en otra herramienta que proveen los lenguajes orientados a objetos llamada sobrecarga. Este tema se explicará posteriormente, por el momento alcanza con saber que para poner más de un constructor se debe cumplir al menos una de las siguientes reglas:

- La cantidad de los parámetros es diferente
- El tipo de los parámetros es diferente

Los constructores permiten que se les pase parámetros, pero cuando se declara un objeto se le deben pasar tantos parámetros como los que figure en alguno de sus constructores o la declaración será un error. De esta manera, si se le quiere poner a la clase anterior un constructor que reciba un entero, quedaría con el siguiente formato.

C#

```
public class Ejemplo
{
        private int var1;
        private int var2;

        public Ejemplo(int v)
        {
                var1 = v;
        }

        public int Var1
        {
                get { return var1; }
                set { var1 = value; }
        }

        public int Metodo()
        {
                return 20;
        }
}
```

VB

```
Public Class Ejemplo
    Private _var1 As Integer
    Private _var2 As Integer

    Public Sub New(v As Integer)
        _var1 = v
```

```
        End Sub

        Public Property Var1() As Integer
            Get
                Return _var1
            End Get
            Set(ByVal value As Integer)
                _var1 = value
            End Set
        End Property

        Public Function Metodo() As Integer
            Return 20
        End Function
    End Class
```

La declaración de un objeto de la clase sería:

C#

```
    Ejemplo obj = new Ejemplo(8);
```

VB

```
    Dim obj As New Ejemplo(8)
```

Una descripción general (sin entrar en detalles como la reserva de memoria) de lo que ocurre cuando se declara un objeto de una clase que tiene constructor, como en el ejemplo anterior, es la siguiente:

- Se reserva el espacio en memoria para el objeto
- Se le pasa el parámetro 8 al constructor
- Se empieza a ejecutar el constructor y se asigna el valor del parámetro a la variable `var1`
- Se termina la ejecución del constructor
- Se devuelve la referencia al objeto y se almacena en la variable `obj`

Uso de un objeto

Para utilizar los servicios que presta un objeto se invocan los elementos declarados en su interfaz pública.

En la clase `Ejemplo` el único elemento en dicha interfaz es la función `Metodo`. Para poder invocarlo se debe utilizar la notación de punto y la referencia almacenada en la variable `obj`, ya que esta indica el lugar de almacenamiento del mismo. El código en un programa se vería de la siguiente manera:

C#

```
    int aux;
    aux = obj.Metodo();
```

VB

```
Dim aux As Integer
aux = obj.Metodo
```

En este código se declara la variable entera `aux` y luego se almacena en ella el valor retornado por el método `Metodo`. Notar la diferencia de notación entre ambos lenguajes.

Ocultamiento de la información

Para proteger los datos que se almacenan dentro de un objeto, la clase utiliza la declaración de visibilidad `private` o `Private` (C# o VB) para que no se pueda acceder la variable, ya que esta declaración indica que la variable no pertenece a la interfaz de la clase. Cuando un elemento no pertenece a la interfaz no se puede acceder por notación de punto una vez creado un objeto, sólo lo podrán acceder aquellos elementos que pertenezcan a la clase.

La ventaja que ofrece .Net para acceder a los elementos privados de una clase es el uso de **propiedades** como elementos de los lenguajes. Se las pueden declarar para automatizar la creación de métodos de acceso y mutación de variables privadas. Las propiedades definen en C# métodos `get` y `set` (`Get` y `Set` en VB) en una notación abreviada para manejar la obtención y modificación de los valores de las variables privadas.
Los siguientes ejemplos muestran como manejar en una clase el ocultamiento de la información mediante propiedades.

Ejemplo

C#

```
public class Persona
{
        private String primerNombre;
        private String segundoNombre;
        private String apellido;
        private String documento;

        public Persona()
        {
        }
        public String PrimerNombre
        {
                get
                {
                        return primerNombre;
                }
                set
                {
                        primerNombre = value;
                }
        }
        public String SegundoNombre
        {
                get
                {
                        return segundoNombre;
```

```
            }
            set
            {
                segundoNombre = value;
            }
        }
        public String Apellido
        {
            get
            {
                return apellido;
            }
            set
            {
                apellido = value;
            }
        }
        public String Documento
        {
            get
            {
                return documento;
            }
            set
            {
                documento = value;
            }
        }
}
```

VB

```
Public Class Persona

    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String

    Public Sub New()

    End Sub
    Public Property PrimerNombre() As String
        Get
            Return _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property
    Public Property SegundoNombre() As String
        Get
            Return _segundoNombre
        End Get
        Set(ByVal value As String)
            _segundoNombre = 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
    Public Property Documento() As String
        Get
            Return _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property
End Class
```

Encapsulado

Como se mostró con anterioridad, en toda clase puede existir una parte pública y una privada. La primera define los elementos de la clase que son accesibles a través de su interfaz. Muchas veces se hace referencia a este hecho como "accesible por el mundo exterior o el universo". Esta frase puede simplificarse de la siguiente manera:

> *Los elementos públicos de una clase serán accesibles por notación de punto una vez creado un objeto de su tipo*

En cambio, la parte privada, define todo lo contrario. Los elementos declarados como privados en una clase no serán accesibles por ningún elemento salvo que este se encuentre en la misma clase.

Este último concepto es el que permite separar los servicios que brinda un objeto de la forma en que lo hace, por lo tanto, en otras palabras, un objeto debe verse como una serie de servicios que presta a través de sus interfaces y la forma en que lo hace se oculta del mundo exterior porque no es accesible.

Cuando una clase oculta una serie de operaciones (métodos y variables de la clase que éstos usan) declarándolos privados y los utiliza posteriormente para brindar un servicio, se dice que este servicio esta encapsulado dentro de la clase.

Otra forma común de denominar a las operaciones o métodos privados de una clase es **servicio privado**.

Se puede decir que el ocultamiento de la información y los métodos privados son en conjunto conocidos como "encapsulado", estén presente uno de ellos o ambos. Se debe notar en este punto que si los métodos se ocultan pero no así las variables de la clase, **no existe encapsulado**

porque un cambio de valor en una variable puede determinar un cambio de comportamiento en los métodos privados.

> *No existe encapsulado sin ocultamiento de la información. Sin embargo, al ocultamiento de la información muchas veces se la denomina encapsulado*

Un ejemplo claro de esto es cuando en una clase se oculta la información pero se quiere brindar la posibilidad de acceder a los datos almacenados, ya sea para guardar valores como para leerlos. En este caso se deben crear métodos públicos que cumplan ese rol.

Ejemplo

C#

```
public class Persona
{
	private String primerNombre;
	private String segundoNombre;
	private String apellido;
	private String documento;
	private String detalles;

	public Persona()
	{
	}
	public String PrimerNombre
	{
		get
		{
			return primerNombre;
		}
		set
		{
			primerNombre = value;
		}
	}
	public String SegundoNombre
	{
		get
		{
			return segundoNombre;
		}
		set
		{
			segundoNombre = value;
		}
	}
	public String Apellido
	{
		get
		{
			return apellido;
		}
```

```
            set
            {
                apellido = value;
            }
        }
        public String Documento
        {
            get
            {
                return documento;
            }
            set
            {
                documento = value;
            }
        }

        public String getDetalles()
        {
            armaDetalles();
            return detalles;
        }

        private void armaDetalles()
        {
            detalles = "Apellido: " + apellido+ ", Primer Nombre: " + primerNombre+
              ", Segundo Nombre: " + segundoNombre + ", Documento: " + documento;
        }
}
```

VB

```
Public Class Persona

    Private _primerNombre As String
    Private _segundoNombre As String
    Private _apellido As String
    Private _documento As String
    Private detalles As String

    Public Sub New()

    End Sub
    Public Property PrimerNombre() As String
        Get
            Return _primerNombre
        End Get
        Set(ByVal value As String)
            _primerNombre = value
        End Set
    End Property
    Public Property SegundoNombre() As String
        Get
            Return _segundoNombre
        End Get
        Set(ByVal value As String)
```

```
            _segundoNombre = 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
    Public Property Documento() As String
        Get
            Return _documento
        End Get
        Set(ByVal value As String)
            _documento = value
        End Set
    End Property
    Public Function getDetalles() As String
        armaDetalles()
        Return detalles
    End Function
    Private Sub armaDetalles()
        detalles = "Apellido: " + Apellido + ", Primer Nombre: " + PrimerNombre +
            ", Segundo Nombre: " + SegundoNombre + ", Documento: " + Documento

    End Sub
End Class
```


Strings

Las cadenas de caracteres o “strings” en .Net se manejan con una clase interna del lenguaje, por lo tanto, siempre será un tipo referenciado aunque los lenguajes permiten su manejo como un tipo primitivo.

Se permiten dos formas básicas de asignar una cadena de caracteres: cuando se crea el objeto y mediante el operador de asignación.

Ejemplo

C#

```
        string string1 = "Esta es una cadena creada por asignación.";
        Console.WriteLine(string1);

        // Creación de un string mediante un constructor que
        // repite el carater Unicode del primer argumento.
        string string3 = new String('c', 20);
        Console.WriteLine(string3);
        Console.ReadKey();
```

VB

```
        Dim string1 As String = "Esta es una cadena creada por asignación."
```

```
Console.WriteLine(string1)

' Creación de un string mediante un constructor que
' repite el carater Unicode del primer argumento.
Dim string3 As New String("c"c, 20)
Console.WriteLine(string3)
Console.ReadKey()
```

Comienzo de un programa en .Net

En .Net, el comienzo de un programa se coloca en un método. A diferencia de otros métodos escritos por el programador este tiene un nombre prestablecido: `Main` en C# y VB. Este método es el punto de entrada para el comienzo de un programa. En VB el método es un subproceso a diferencia de C# que sólo utiliza funciones

Nota: cuando se realizan aplicaciones con interfaces gráficas, VB permite omitir la declaración explícita de este subproceso, lo cual puede derivar en confusión. Sin embargo, el mismo existe a través de indicar como punto de entrada al formulario principal. Si se lo desea, se puede declarar explícitamente indicándole al proyecto que dicho subproceso es el punto de entrada a considerar. Sin embargo, la forma de realizarlo va más allá de los objetivos actuales a desarrollar.

En VB un programa puede poseer tanto módulos como clases y el programa puede comenzar en cualquiera de ellos. Sin embargo, en C# el método sólo puede estar dentro de una clase.

Nota: Si un programa es de consola en VB, se crea por defecto un módulo que posee el método `Main`. Sin embargo esto no es así cuando se crea una aplicación con interfaz gráfica

Como no esta definido que un programa deba comenzar en una clase en particular en C#, indica que puede haber muchas clases que posean el método `Main`, pero sólo puede haber un método de este tipo por clase.

Ejemplo

C#

```
class Program
{
    static void Main(string[] args)
    {
        string string1 = "Esta es una cadena creada por asignación.";
        Console.WriteLine(string1);

        // Creación de un string mediante un constructor que
        // repite el carater Unicode del primer argumento.
        string string3 = new String('c', 20);
        Console.WriteLine(string3);
        Console.ReadKey();
    }
```

VB

```
Module Module1

    Sub Main()
        Dim string1 As String = "Esta es una cadena creada por asignación."
        Console.WriteLine(string1)

        ' Creación de un string mediante un constructor que
        ' repite el carater Unicode del primer argumento.
        Dim string3 As New String("c"c, 20)
        Console.WriteLine(string3)
        Console.ReadKey()
    End Sub

End Module
```

Espacios de nombres

Una forma de organizar proyectos en .NET es mediante el uso de espacios de nombres. Estos son una forma de agrupar los nombres de los tipos (clases en C# y VB) reduciendo de la probabilidad de conflictos de nombres. Un espacio de nombres además puede contener otros espacios de nombres que tengan declarados en él sus propios tipos (a esto se lo conoce como anidamiento). El nombre completo de un tipo incluye la combinación de espacios de nombres que lo contienen.

El nombre completo de una clase se construye mediante la concatenación de los nombres de todos los espacios de nombres que lo contienen. Por ejemplo, el nombre completo de la clase Button es System.Windows.Forms.Button. La jerarquía de espacio de nombres ayuda a distinguir los tipos con el mismo nombre entre si. Por ejemplo, se puede definir una clase propia llamada Button que podrían estar en otro espacio de nombres. La única limitación es que no puede haber dos clases con el mismo nombre en un mismo espacio de nombres.

En ambos lenguajes, el espacio de nombre por defecto que se coloca en un proyecto coincide con el nombre de éste. Sin embargo, se pueden declarar otros espacios de nombre en un proyecto. Cabe destacar que en C# el espacio de nombre se declara siempre en forma explícita dentro de la definición de una nueva clase. Sin embargo, en VB esta declaración es implícita y se la puede observar en las propiedades de un proyecto. Por el momento, sólo se utilizará en forma básica las declaraciones de los espacios de nombres en los diferentes lenguajes.

Atención: si se declara explícitamente un espacio de nombres en VB, este estará anidado al espacio de nombre por defecto que se define con el nombre del proyecto.

Ejercicios

Los temas de los que tratan los ejercicios de este módulo son los siguientes:

- Clases en .Net
- Tipos básicos

Ocultamiento de la información