

IMPLEMENTACIÓN DE MÉTODOS EN C# .NET

Bruno López Takeyas
Instituto Tecnológico de Nuevo Laredo
Reforma Sur 2007, C.P. 88070, Nuevo Laredo, Tamps. México
<http://www.itnuevolaredo.edu.mx/takeyas>
E-mail: takeyas@itnuevolaredo.edu.mx

Resumen: *El presente documento tiene como objetivo crear conciencia en los programadores en C# .NET sobre el aprovechamiento de los recursos que proporciona el lenguaje para mejorar el estilo de programación y facilitar la depuración y corrección de los programas. Para este efecto se mencionan temas de suma importancia: Uso de métodos (procedimientos y funciones), manejo de variables (locales y globales), el envío de parámetros o argumentos (por valor y por referencia) y la recepción de valores; de tal forma que sean de utilidad al implementar programas orientados a objetos.*

1. Introducción

En los primeros paradigmas de programación de computadoras, las instrucciones se escribían y ejecutaban de manera secuencial o lineal; es decir, se codificaban las sentencias una después de la otra y seguían este patrón durante su ejecución. Sin embargo, este estilo provocaba programas muy extensos, poco legibles, mal organizados y por ende, complicados de depurar o corregir; a esto se le añade que en muchas ocasiones había necesidad de ejecutar un conjunto de instrucciones en varias ocasiones, lo cual provocaba escribirlo repetidamente en la codificación, ocasionando duplicidad de código y por ende más trabajo para el programador, ya que debía escribir varias veces el mismo código en el programa, revisarlo y provocando que los programas ocuparan más memoria y se tornaran difíciles de depurar.

Con el surgimiento del paradigma de la programación estructurada, se introduce la idea de organizar un programa de computadora en módulos, los cuales permiten organizarlo e identificar claramente la operación de los mismos. Cada módulo está identificado con un nombre, contiene

un conjunto de instrucciones que solamente se escriben una vez, pero pueden ser invocados las veces que sean necesarias; de tal forma, que ofrece al programador la facilidad de organizar sus programas de forma clara, precisa y fácil de depurar. En este paradigma de programación, estos módulos fueron conocidos con el nombre de subrutinas o subprogramas, los que actualmente en el paradigma de programación orientado a objetos se conocen con el nombre de métodos.

2. Definición de método

En la actualidad se conoce con el nombre de método a un conjunto de instrucciones que realiza una tarea específica y bien definida. Los métodos solamente se escriben una vez pero pueden ser invocados en múltiples ocasiones durante la ejecución de un programa. Esto le brinda al programador las siguientes ventajas:

- Facilita la separación de actividades en módulos debidamente identificados.
- Organiza de manera legible y fácil de entender a los programas.
- Facilita al programador la escritura de código.
- Facilita la depuración, corrección y mantenimiento de los programas.

Los métodos se clasifican en procedimientos y funciones (Fig. 1).

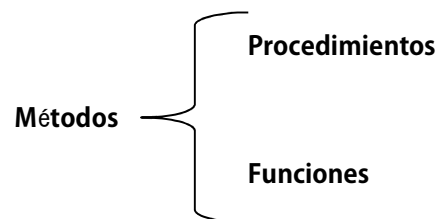


Fig. 1.- Tipos de métodos

En el paradigma orientado a objetos, los métodos representan las acciones realizadas por los objetos; por lo que se recomienda que se utilicen verbos para nombrarlos e identificarlos.

3. Procedimientos

Un procedimiento es un método que se compone de un conjunto de instrucciones para realizar un proceso, sin embargo, no devuelve algún resultado como producto de su operación; simplemente ejecuta las instrucciones que contiene sin informar el resultado obtenido. En C#, los procedimientos se identifican por su declaración de tipo *void*.

Por ejemplo, el siguiente método se encarga de imprimir en pantalla los datos de una persona; sin embargo, no devuelve valor alguno. Nótese que el método se declara de tipo *void* indicando que no devuelve valor; por lo tanto se trata de un procedimiento.

```
static void Imprimir()
{
    Console.WriteLine(Nombre);
    Console.WriteLine(Edad);
    Console.WriteLine(Sueldo);
}
```

La palabra *static* indica que el método es un miembro estático del programa; es decir, solamente se crea una vez cuando se ejecuta el programa y existe mientras se ejecute la aplicación (por el momento no se explica a detalle este concepto, el cual se trata en cursos de Programación Orientada a Objetos).

4. Funciones

Las funciones son métodos que ejecutan un conjunto de instrucciones e informan del resultado obtenido; es decir, devuelven el dato resultante de la ejecución. En C#, una función utiliza la sentencia *return()* para devolver el valor correspondiente. Enseguida se muestran algunos ejemplos de declaraciones de funciones:

```
static int Sumar() // Devuelve un valor
de tipo numérico entero
```

```
static double Calcular() // Devuelve un
valor de tipo numérico real
```

```
static string Comparar() // Devuelve un
valor de tipo cadena
```

Por ejemplo, el siguiente método calcula el área de una circunferencia aplicando la fórmula $A = \pi \text{Radio}^2$ y devuelve el resultado. Nótese que el método se declara de tipo *double*, indicando que devuelve un valor numérico real, por lo tanto se trata de una función.

```
static double CalcularArea()
{
    return(Math.PI*Math.Pow(Radio,2));
}
```

4.1.- Limitación de *return()*

Una limitante de una función es que la sentencia *return()* sólo devuelve un valor; esto restringe a que una función solamente pueda devolver un dato. Si se desea que la función devuelva más de un valor, entonces debe usarse otro mecanismo (por ejemplo el envío de parámetros por referencia ó el uso de parámetros de salida *out* en C# .NET).

5. Ámbito de las variables: Variables locales y globales

En el contexto de programación, se conoce como el ámbito a la disponibilidad que ofrece una variable dependiendo del lugar donde se declare. Las variables que se declaran dentro de un método o un bloque de sentencias se llaman variables locales mientras que las variables globales se conocen a través del programa entero y se pueden usar en cualquier segmento de código.

El valor de una variable local solamente se puede acceder dentro del segmento de código donde fue declarada dicha variable y no puede utilizarse en otra sección; en cambio una variable global puede accederse en cualquier parte del programa, manteniendo disponible su valor en todo momento.

Se pueden declarar variables globales declarándolas fuera de cualquier método (antes de *Main()*) y cualquier método puede acceder a ellas sin tener en cuenta en qué segmento de código esté dicha declaración.

6. Envío de parámetros a los métodos

Un método (procedimiento o función) puede recibir datos para realizar algunas acciones, los cuales se denominan parámetros o argumentos.

Existen dos formas de enviar datos a un método: por valor y por referencia (Fig. 2).

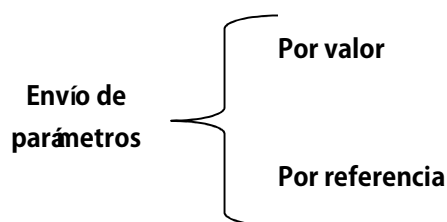


Fig. 2.- Tipos de envíos de parámetros

Cuando se invoca un método al que se le envía un parámetro por valor, se le manda una copia del valor de una variable o expresión, el cual es recibido por un parámetro declarado del mismo tipo de dato que el valor enviado. En cambio, si se le envía un parámetro por referencia, se le manda la referencia (dirección de memoria) de una variable.

Se pueden enviar varios datos a un método, sin embargo es necesario precisar que las variables receptoras deben estar declaradas en el orden indicado por el envío, también considerando la correspondencia con el tipo de dato.

Cuando se invoca un método se colocan entre paréntesis los parámetros enviados.

Es importante mencionar que los parámetros que reciben los datos enviados al método se consideran variables locales (independientemente si se hace por valor o por referencia) e incluso pueden tener el mismo nombre que la variable origen, sin embargo se trata de copias de los valores originales, por lo tanto, son variables diferentes.

6.1.- Envío de parámetros por valor

Cuando se envía un parámetro por valor, se hace una copia del valor de la variable enviada en el parámetro recibido, el cual actúa como una variable local, que lo recibe, lo manipula dentro del método y que desaparece y pierde su valor al terminar la ejecución de ese segmento de código.

Por ejemplo, al invocar (llamar) el método identificado con el nombre `Procesar`, se colocan entre paréntesis los parámetros `Nombre`, `Edad` y `Sueldo` separados por comas:

```
Procesar(Nombre, Edad, Sueldo);
```

La declaración del método se muestra enseguida:

```
void Procesar(string N, int E, double S)
{
    .....
}
```

En este caso, el parámetro `N` recibe una copia del valor de la variable `Nombre`, `E` recibe copia del valor de `Edad` y `S` recibe copia del valor de `Sueldo` (Fig. 3). Obsérvese que durante la llamada del método, se envían una cadena, un entero y un número real respectivamente, los cuales son recibidos en ese orden por los correspondientes parámetros; esto es, debe respetarse el orden de los tipos de datos enviados en la declaración de los parámetros.

| Memoria RAM | | |
|-------------|---------|----------|
| Dirección | Valor | Variable |
| FA31:B278 | "Pepe" | Nombre |
| ... | | |
| FA31:C45C | 18 | Edad |
| ... | | |
| FA31:D2A8 | 1500.50 | Sueldo |
| ... | | |
| FA31:E6A1 | "Pepe" | N |
| ... | | |
| FA31:E9A2 | 18 | E |
| ... | | |
| FA31:F3A8 | 1500.50 | S |
| | | |

Fig. 3.- Almacenamiento en memoria de los parámetros enviados por valor.

Ahora se considera el siguiente ejemplo: Declarar una variable global denominada "x" y se inicializa con el valor 5 ($x=5$). Dentro del programa principal se declara y se inicializa una variable local llamada "y" con el valor 13 ($y=13$). Cuando se hace la llamada a un `Metodo(y)` y se envía la variable "y", se hace por valor, es decir, se envía una copia del valor de la variable (13) que lo recibe otra variable local "a". En ese momento, se transfiere el control de la ejecución del programa hacia el método, activando su variable

local “a” y desactivando momentáneamente la variable local del programa principal “y” (la variable “x” permanece activa ya que se trata de una variable global). Dentro del método, se modifica el valor de su variable local “a”, se imprime (16) y se duplica el valor de la variable global “x” y se imprime (10). Cuando el método termina, el sistema desactiva su variable local “a”, regresa el control de la ejecución del programa al lugar donde se hizo la llamada, activa y recupera el valor de su propia variable local (y=13) y continúa con su operación. Al imprimir los valores, nos percatamos que la variable “x” modificó su valor por tratarse de una variable global que puede ser accedida en cualquier parte del programa, sin embargo, la variable “y” mantiene su valor original (no fué alterado), ya que por tratarse de una variable local, fue desactivada al llamar al Metodo() y reactivada con su valor original al retornar. Enseguida se muestra el código en C# de este ejemplo (Prog. 1):

```
class Program
{
    static int x = 5; // Variable global

    static void Main(string[] args)
    {
        int y = 13; // Variable local

        Console.WriteLine("\nx=" + x);

        // Llamada al método y envío por valor
        Metodo(y);

        Console.WriteLine("\nx=" + x);

        Console.WriteLine("\ny=" + y);

        Console.ReadKey();
    }

    // El parámetro "a" recibe el valor de "y"
    static void Metodo(int a)
    {
        a = a + 3;
        Console.WriteLine("\na=" + a);

        x = x * 2;
    }
}
```

Prog. 1.- Envío de parámetros por valor

Al ejecutar el programa anterior se produce la salida mostrada en la Fig. 4.

```
x=5
a=16
x=10
y=13
```

Fig.4.- Salida del programa de envío de parámetros por valor.

El Prog. 1 completo puede descargarse de:

<http://www.itnuevolaredo.edu.mx/takeyas/libroED/Prog7-1.rar>

Para monitorear los valores de las variables e identificar su ámbito, se recomienda ejecutar paso a paso por instrucciones este programa en Microsoft Visual C# 2010 ó 2012 oprimiendo repetidamente la tecla F11.

6.2.- Envío de parámetros por referencia

Todas las variables se almacenan en celdas de la memoria RAM (*Random Access Memory* por sus siglas en inglés), las cuales están identificadas por una dirección expresada en números hexadecimales (p. ejem FA4D:32CE).

Cuando se envía un parámetro por referencia, no se hace una copia de su valor y quien lo recibe, no contiene directamente el dato, sino una referencia (dirección de memoria) a él.

El envío de parámetros por referencia permite a un método cambiar el valor del parámetro y mantener vigente dicho cambio. Cuando una variable es enviada por referencia, el método recibe la referencia de la variable original y esto implica que los cambios realizados a esa variable dentro del método, afectan la variable original.

Para ilustrarlo mejor, consideremos el ejemplo de la sección anterior, pero ahora enviando la variable “y” por referencia: Se declara una variable global denominada “x” y se inicializa con el valor 5 (x=5). Dentro del programa principal se declara y se inicializa la variable “y” con el valor 13 (y=13), la cual se considera variable local al ser declarada dentro de un método. Cuando se hace la llamada del Metodo(ref y) y se envía la variable “y”, se hace por referencia, es decir, se envía la dirección de memoria (no el valor) de la variable “y”, que lo recibe

la variable local "a". Dentro del método, se modifica el valor de la variable "a", se imprime (16) y se duplica el valor de la variable global "x" (10) y se imprime. Cuando el método termina, el sistema desactiva su variable local "a", regresa el control de la ejecución del programa al lugar donde se hizo la llamada, activa y recupera los valores de su propia variable local (y=13) y continúa con su operación. Al imprimir los valores, nos percatamos que la variable "x" modificó su valor por tratarse de una variable global que puede ser accedida en cualquier parte del programa y la variable "y", aunque se trata de una variable local, también modificó su valor, ya que siendo la variable "a" una referencia de la variable "y", entonces al modificar "a" también se modifica "y". Esto se debe a que la variable "a" no recibe una copia del valor de la variable "y" sino la dirección de memoria donde está almacenado dicho valor (FA31:C45C). A este concepto se le conocía con el nombre de apuntador en lenguajes como Pascal, C y C++, ya que la variable "a" apunta al valor almacenado en la variable "y" (Fig. 5).

Memoria RAM

| <i>Dirección</i> | <i>Valor</i> | <i>Variable</i> |
|-------------------------|---------------------|------------------------|
| FA31:B278 | 5 | x |
| ... | | |
| FA31:C45C | 13 | y |
| ... | | |
| FA31:D2A8 | FA31:C45C | a |
| ... | | |




Fig. 5.- Almacenamiento en memoria de los parámetros enviados por referencia.

El siguiente código en C# ilustra este ejemplo (Prog. 2):

```
class Program
{
    static int x = 5; // Variable global

    static void Main(string[] args)
    {
        int y = 13; // Variable local

        Console.WriteLine("\nx=" + x);
```

```
// Llamada al método y envío por referencia
Metodo(ref y);

Console.WriteLine("\nx=" + x);

Console.WriteLine("\ny=" + y);

Console.ReadKey();
}

// El parámetro "a" recibe la ref. de "y"
static void Metodo(ref int a)
{
    a = a + 3;
    Console.WriteLine("\na=" + a);

    x = x * 2;
}
}
```

Prog. 2.- Envío de parámetros por referencia.

Al ejecutar el programa anterior se produce la salida mostrada en la Fig. 6.

```
x=5
a=16
x=10
y=16
```

Fig. 6.- Salida del programa de envío de parámetros por referencia.

El Prog. 2 completo puede descargarse de:

<http://www.itnuevolaredo.edu.mx/takeyas/libroED/Prog7-2.rar>

Para monitorear los valores de las variables e identificar su ámbito, se recomienda ejecutar paso a paso por instrucciones este programa en Microsoft Visual C# 2010 ó 2012 oprimiendo repetidamente la tecla F11.

6.3.- Parámetros de salida out en C#.NET

Un parámetro de salida en C# (*out*) es muy similar a un parámetro por referencia (*ref*), excepto que los parámetros por referencia deben ser inicializados antes de enviarse; sin embargo, el método debe asignarle un valor antes de devolverlo. Para utilizar un parámetro de salida, basta con anteponer la palabra *out* tanto en el parámetro enviado como en su declaración en el método.

Este tipo de parámetros son útiles cuando se requiere que una función devuelva más de un dato, ya que por definición, existe una restricción de que una función solamente devuelve un valor.

Para ilustrar mejor el uso de un parámetro de salida, tomaremos como ejemplo el programa anterior (Prog. 2) al que se le agrega una variable booleana como parámetro de salida para determinar si el parámetro enviado por referencia es par ó impar. Este código en C# muestra este ejemplo (Prog. 3):

```
class Program
{
    static int x = 5; // Variable global

    static void Main(string[] args)
    {
        int y = 13; // Variable local
        bool esImpar;

        Console.WriteLine("\nx=" + x);

        // Llamada al método y envío por referencia
        Metodo(ref y, out esImpar);

        Console.WriteLine("\nx=" + x);

        Console.WriteLine("\ny=" + y);

        if (esImpar)
            Console.WriteLine("\ny es un número
impar");
        else
            Console.WriteLine("\ny es un número
par");

        Console.ReadKey();
    }

    // El parámetro "a" recibe la referencia de
    "y" y el parámetro de salida sirve para
    determinar si el parámetro enviado es Impar
    static void Metodo(ref int a, out bool Impar)
    {
        a = a + 3;
        Console.WriteLine("\na=" + a);
    }
}
```

```
x = x * 2;

if (a % 2 != 0)
    Impar = true;
else
    Impar = false;
}
```

Prog.3.- Parámetro de salida en C#.

Al ejecutar el programa anterior se produce la salida mostrada en la Fig. 7.

```
x=5
a=16
x=10
y=16
y es un número par
```

Fig. 7.- Salida del programa de uso de parámetro de salida.

El Prog. 3 completo puede descargarse de:

<http://www.itnuevolaredo.edu.mx/takeyas/libroED/Prog7-3.rar>

Para monitorear los valores de las variables e identificar su ámbito, se recomienda ejecutar paso a paso por instrucciones este programa en Microsoft Visual C# 2010 ó 2012 oprimiendo repetidamente la tecla F11.

7. Recibiendo un valor de una función

Una vez que se invoca una función es necesario utilizar la variable capaz de recibir el valor devuelto por ésta, la cual debe ser del mismo tipo de la función. Por ejemplo, si se invoca el siguiente método

```
x = Procesar(a, b);
```

la variable "x" recibe el valor calculado por la función `Procesar()`, que acepta los parámetros "a" y "b" respectivamente.

Para ilustrarlo mejor, se considera el siguiente ejemplo: Se desea implementar una función que calcule el área de una circunferencia; para ello, la función recibe como parámetro el valor del radio, aplica la fórmula correspondiente y devuelve el resultado calculado. El programa principal solicita al usuario capturar el valor del radio de una circunferencia y almacenarlo en la variable local llamada "Radio", la cual es enviada como parámetro por valor a la función `CalcularArea()`, que recibe el parámetro en la variable local llamada "r" aplica la fórmula $Area = \pi r^2$ y devuelve el resultado calculado al programa principal quien lo recibe en la variable "Area". El siguiente código en C# ilustra este ejemplo (Prog. 4):

```
class Program
{
    static void Main(string[] args)
    {
        double Radio, Area;
        Console.WriteLine("Teclee el valor del radio:");
        Radio = double.Parse(Console.ReadLine());

        // La variable Area recibe el valor devuelto
        // por la función
        Area = CalcularArea(Radio);

        Console.WriteLine("Área = " + Area);
        Console.ReadKey();
    }

    static double CalcularArea(double r)
    {
        return (Math.PI * Math.Pow(r, 2));
    }
}
```

Prog. 4.- Función para calcular el área de una circunferencia

Al ejecutar el programa anterior se produce la salida mostrada en la Fig. 8.

```
Teclee el valor del radio: 2.3
Área = 16.61902513749
```

Fig. 8.- Salida del programa de uso de una función.

El Prog. 4 completo puede descargarse de:

<http://www.itnuevolaredo.edu.mx/takeyas/repasoFP/Prog4.rar>

Para monitorear los valores de las variables e identificar su ámbito, se recomienda ejecutar paso a paso por instrucciones este programa en Microsoft Visual C# 2010 ó 2012 oprimiendo repetidamente la tecla F11.

8. Aplicaciones prácticas del uso de métodos

Existen numerosas aplicaciones prácticas del uso de métodos para el desarrollo de sistemas computacionales. La implementación de métodos ayuda a organizar mejor el diseño de programas y facilitan su mantenimiento.

El Prog. 5 muestra una aplicación típica de un método al que se le envían parámetros: el ordenamiento de los datos de un arreglo. El programa principal solicita al usuario que capture la cantidad de celdas de un arreglo unidimensional y lo almacena en la variable local llamada "Tamaño", crea el arreglo local y lo llena con números enteros generados de manera aleatoria. Después invoca al procedimiento llamado `Ordenar()` y le envía como parámetros el "Arreglo" completo y su "Tamaño" (los cuales son recibidos por el arreglo "A" y la variable "T" respectivamente) para implementar un algoritmo que ordena de manera ascendente (de menor a mayor) los datos almacenados en el arreglo (en este momento no es importante explicar el método de ordenamiento). Obsérvese que se trata de un procedimiento, puesto que no devuelve valor; sin embargo, no es necesario devolver valor alguno, ya que cuando se envía un arreglo como parámetro a un método, automáticamente se hace por referencia; es decir, no se requiere implementar una función que devuelva el arreglo con los datos ordenados; sino que al hacer los cambios en el arreglo recibido dentro del método (arreglo "A"), éstos se reflejan de manera inmediata en el arreglo original (Arreglo) que fue generado en el programa principal. El siguiente código en C# muestra este ejemplo:

```
class Program
{
    static void Main(string[] args)
    {
```

```

// Declaración del tamaño del arreglo
int Tamaño;

// Declaración del arreglo
int [] Arreglo;

// Generar un número aleatorio
Random NumeroAleatorio = new
Random(int.MaxValue);

Console.Write("Teclee el tamaño del arreglo:
");
Tamaño = int.Parse(Console.ReadLine());

// Creación del arreglo
Arreglo = new int[Tamaño];

// Llena el Arreglo con números generados
aleatoriamente
Console.WriteLine("\nARREGLO
DESORDENADO\n");

for (int i = 0; i < Tamaño; i++)
{
    // Genera núm. aleatorio y lo almacena en
    la celda "i" del Arreglo
    Arreglo[i] = NumeroAleatorio.Next();

    // Imprime el núm. generado
    Console.WriteLine("{0:N0} ", Arreglo[i]);
}

Console.Write("\nOprima cualquier tecla para
ordenar el arreglo ...");
Console.ReadKey();

// Invoca el procedimiento "Ordenar" y le
envía el "Arreglo" y su "Tamaño"
Ordenar(Arreglo, Tamaño);

// Limpia la pantalla
Console.Clear();

// Imprime el arreglo
Console.WriteLine("ARREGLO ORDENADO\n");

for(int i=0; i<Tamaño; i++)
    Console.WriteLine("{0:N0} ", Arreglo[i]);

Console.Write("\nOprima cualquier tecla para
salir ...");
Console.ReadKey();
}

// Procedimiento para ordenar un arreglo
recibido como parámetro
static void Ordenar(int[] A, int T)
{

```

```

for(int p=0; p<T-1; p++)
    for(int e=p+1; e<T; e++)
        if (A[e] < A[p])
        {
            int temp=A[p];
            A[p] = A[e];
            A[e] = temp;
        }
}
}

```

Prog. 5.- Procedimiento para ordenar los datos de un arreglo

Al ejecutar el programa anterior se produce la salida mostrada en la Fig. 9.

```

Teclee el tamaño del arreglo: 6

ARREGLO DESORDENADO

1,559,595,546
1,755,192,844
1,649,316,172
1,198,642,031
442,452,829
1,200,195,955

Oprima cualquier tecla para ordenar el
arreglo ...

ARREGLO ORDENADO

442,452,829
1,198,642,031
1,200,195,955
1,559,595,546
1,649,316,172
1,755,192,844

Oprima cualquier tecla para salir

```

Fig. 9. Salida del programa de ordenamiento de un arreglo

El Prog. 5 completo puede descargarse de:

<http://www.itnuevolaredo.edu.mx/takeyas/repasoFP/Prog5.rar>

9. Conclusiones

El uso de métodos produce sistemas organizados en módulos que le facilitan al programador su legibilidad, depuración y mantenimiento, dando como consecuencia menor esfuerzo de diseño y programación; razón por la que es muy importante concientizar a estudiantes que vayan a cursar la materia de Programación Orientada a Objetos (POO) sobre la importancia de dominar el uso de métodos (procedimientos, funciones), envío de parámetros y recepción del valor de una función, ya que son un componente fundamental de los objetos porque representan sus acciones realizadas. Desde el inicio del curso de POO, se realiza diseño orientado a objetos en el que los métodos son la parte medular del comportamiento de los objetos, el cual, posteriormente se transforma en sistemas codificados en un lenguaje de programación como C# .NET.

10. Bibliografía

- Archer, Tom. **“A fondo C#”**. Editorial McGraw Hill. 2001.
- Ceballos, Francisco Javier. **“Enciclopedia de Microsoft Visual C#”**. Editorial Alfa Omega. 2010.
- Ceballos, Francisco Javier. **“Microsoft C#. Curso de Programación”**. Editorial Alfaomega. 2008.
- Ceballos, Francisco Javier. **“Microsoft C#. Lenguaje y aplicaciones”**. Editorial Alfaomega. 2008.
- Deitel & Deitel. **“Programming in C#”**. Editorial Prentice Hall.
- Ferguson, Jeff. **“La Biblia de C#”**. Editorial Anaya. 2003.
- López Takeyas, Bruno. **“Curso de Programación en C#”**. Filminas. Instituto Tecnológico de Nuevo Laredo. Consultado el 10 de diciembre de 2012 de <http://www.itnuevolaredo.edu.mx/takeyas/Apuntes/POO/index.htm>
- López Takeyas, Bruno. **“Estructuras de datos orientadas a objetos. Pseudocódigo y aplicaciones en C# .NET”**. Editorial Alfaomega. México. 2012.
- Miles, Rob. **“C# Development”**. Department of Computer Science. University of Hull. 2008-2009.
- Roque Hernández, Ramón. **“POO (Ingeniería – Tecnológico de Nuevo Laredo)”**. Filminas. Instituto Tecnológico de Nuevo Laredo. Consultado el 10 de diciembre de 2012 de <http://ramonroque.com/Materias/pooTec.htm>.

El autor



Bruno López Takeyas se tituló de Ingeniero en Sistemas Computacionales en el Instituto Tecnológico de Nuevo Laredo en 1993. Obtuvo el grado de Maestro en Ciencias de la Administración con especialidad en Sistemas en la Universidad Autónoma de Nuevo León en marzo del 2000.

Desde 1994 es profesor del Depto. de Sistemas y Computación del Instituto Tecnológico de Nuevo Laredo. También es profesor invitado de varias universidades públicas y privadas en sus programas de nivel maestría.

Ha impartido varias conferencias relacionadas con sistemas computacionales, las más recientes en el Instituto Tecnológico de Cancún, Instituto Tecnológico de Piedras Negras, Universidad Autónoma de Tamaulipas y para la Universidad Técnica de Machala, Ecuador. Es autor de los libros “Introducción a la ISC y al diseño de algoritmos” y “Estructuras de datos orientadas a objetos. Pseudocódigo y aplicaciones en C# .NET”.

Contacto:

Email: takeyas@itnuevolaredo.edu.mx

Web: <http://www.itnuevolaredo.edu.mx/takeyas>