ARTÍCULO: "IMPLEMENTACIÓN DE MÉTODOS EN C#.NET"

BRUNO LÓPEZ TAKEYAS

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 tomaran 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).

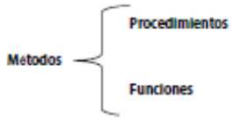
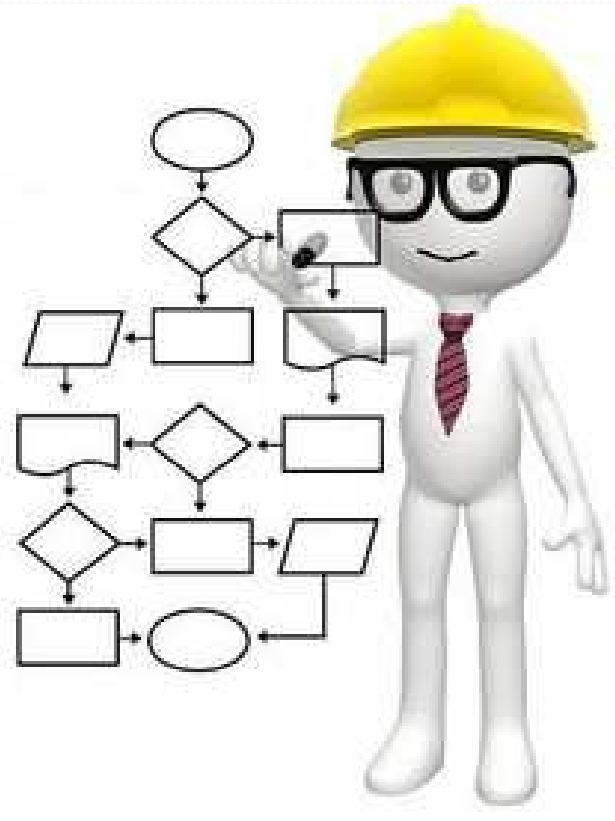


Fig. 1.- Tipos de métodos

ARTÍCULO:
"IMPLEMENTACIÓN DE MÉTODOS EN C#.NET"



1.- INTRODUCCIÓN

- Los primeros lenguajes de programación eran lineales o secuenciales
  - Muy extensos
  - Poco legibles
  - Mal organizados
  - Complicados de corregir o depurar
  - Duplicidad de código
- Programación estructurada: Usa subrutinas (métodos)

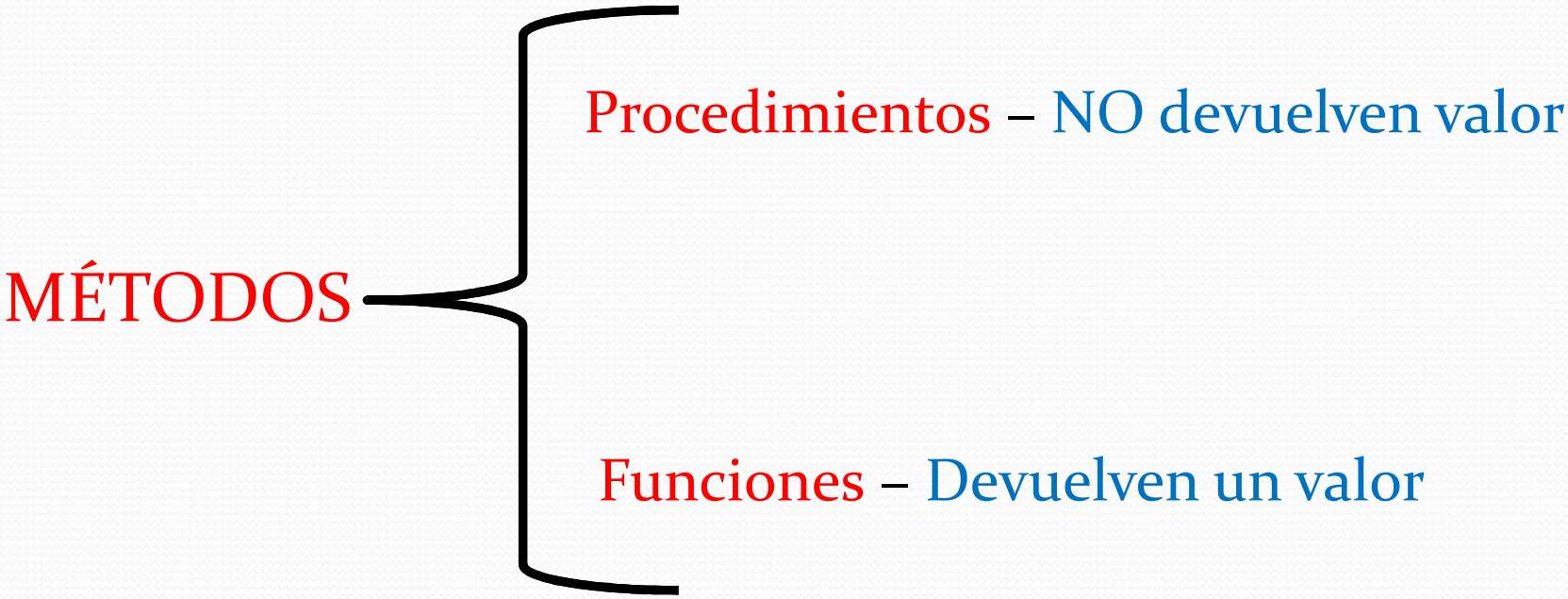
2.- DEFINICIÓN DE MÉTODO

- Es un conjunto de instrucciones que realiza una tarea específica y bien definida que se escribe solamente una vez pero puede invocarse muchas ocasiones.

- Antes se conocían con el nombre de "subrutinas" o simplemente "rutinas"

Métodos

- En C# las subrutinas se conocen como métodos y se clasifican en …



3.- PROCEDIMIENTOS

- Es un método que realiza una acción específica pero **NO** devuelve valor.

- Se declaran de tipo `void`

Sintaxis de los métodos

```
< tipoValorDevuelto > < nombreMétodo > (< parámetros >)
{
      < cuerpo >
}
```

void significa que NO devuelve valor (procedimiento)

- *Ejemplo:*

```
void Saludo( )
{
    Console.Write("Hola");
}
```

Ejemplo de un procedimiento

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

4.- FUNCIONES

- Son métodos que realizan alguna acción específica y bien definida e informan del resultado obtenido.
- Devuelven 1 valor.
- Utiliza la sentencia `return()` para devolver el valor deseado.
- Teóricamente una función **NO** puede devolver más de un valor.

Ejemplos 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
```

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

4.1.- Limitación de return()

- Teóricamente una función **NO** puede devolver más de un valor.
- Si se desea que una función devuelva más de un valor, entonces implemente:
  a) Envío de parámetros por referencia (`ref`).
  b) Parámetros de salida (`out`).

5.- ÁMBITO DE LAS VARIABLES: VARIABLES LOCALES Y GLOBALES

- Se conoce como el ámbito de una variable a su disponibilidad dependiendo de la ubicación de su declaración
- El ámbito de una variable define dónde puede usarse esa variable
- Una variable local declarada en un bloque de programa, sólamente puede ser usada en ese bloque
- El ámbito de una variable también aplica a los métodos y a los ciclos

Ámbito de las variables

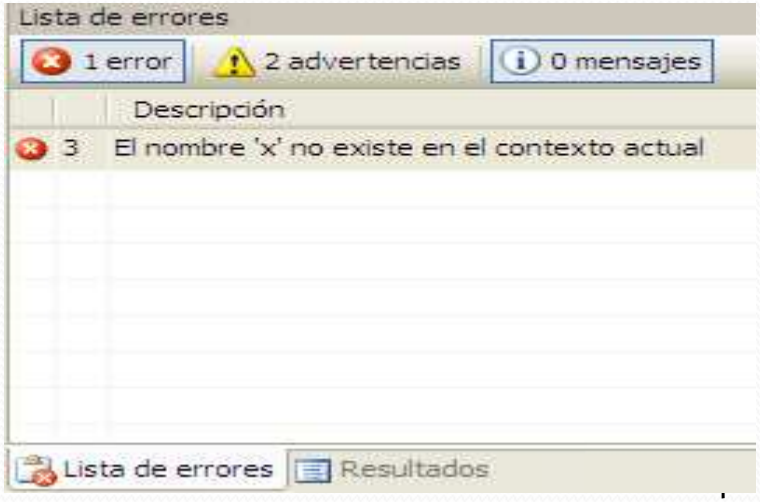
Ámbito de variables

Locales – Se declaran y utilizan dentro de un contexto específico. No se puede hacer referencia a ellas fuera de la sección de código donde se declara.

Globales – Se declaran fuera del cuerpo de cualquier método

Ámbito de variables en un ciclo for

```
for(int x = 1; x<=10; x++)
{
   Console.Write(x);
}

Console.Write(x);
```



Llamadas a los métodos

```
class Program
    {
        static void Main(string[] args)
        {
            Metodo(); // Se invoca (llamada)
        }

        static void Metodo( )
        {
                . . . // Codificación
        }
     }
```

Llamadas de procedimientos

```
class Program
    {
        static void Main(string[] args)
        {
            Procedimiento(); // Llamada
        }

        static void Procedimiento( )
        {
            Console.Write("Tec Laredo");
            return();// Fin del Procedimiento
        }
    }
```

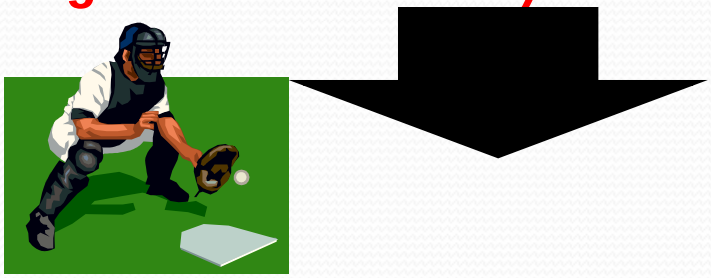
El Procedimiento NO devuelve valor

El uso de la sentencia return() es opcional

6.- ENVÍO DE PARÁMETROS A LOS MÉTODOS

- Entre los paréntesis se especifican una o mas variables (separadas por comas) con sus respectivos tipos de datos.
- Esas variables estarán accesibles dentro del método.

```
public void CambiarEstado( string nuevoestado )
{
    estado = nuevoestado;
}
```



- Al momento de invocar el método, se deben incluir esos valores en la llamada:

```
CambiarEstado("Apagado");
```

Firma de un método

- Se le llama la firma al conjunto de parámetros que recibe un método
- La firma de un método define:
  a) *La cantidad*
  b) *El orden*
  c) *Tipo de dato de cada uno de los parámetros*

Firma de un método (*cont.*)

- Debe coincidir la firma de los parámetros enviados con la firma definida en el método que los recibe.
- *Ejemplo:*
- *Si un método tiene una firma con 2 parámetros, entonces cuando se invoque deben enviarse 2 parámetros (respetando el orden de los datos enviados).*

Parámetros recibidos por los métodos

Parámetros

- Por valor – Se envía una copia del valor de la variable
- Por referencia – Se envía la dirección de la variable

6.1.- Envío de parámetros por valor

```
class Program
    {
        static void Main(string[] args)
        {
            int x=10;
            Metodo( x ); // Se envía el valor de x
            Console.Write("x="+x.ToString()); // x=10
        }
        static void Metodo(int y)
        {
            y+=5;
            Console.Write("y="+y.ToString()); // y=15
        }
     }
```

Envío de parámetros por valor

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

```
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;
  }
}
```

Descargar el ejemplo de parámetro por valor en

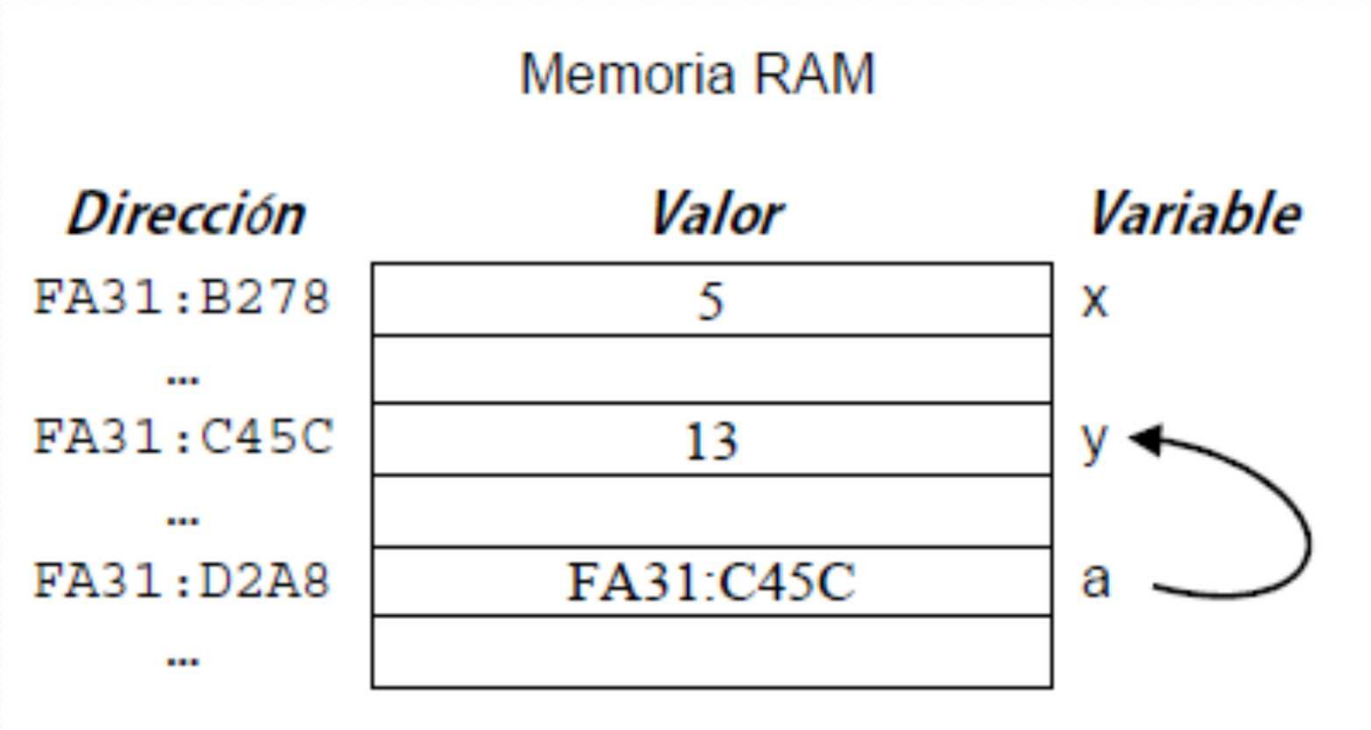
http://www.itnuevolaredo.edu.mx/Takeyas/libroED/Prog7-1.rar

6.2.- Envío de parámetros por referencia

```
static void Main(string[] args)
        {
            int x = 10;
            Metodo(ref x);  // Se envia la referencia de x
            Console.WriteLine("x=" + x); // x=15
            Console.ReadKey();
        }
        static void Metodo(ref int y)
        {
            y += 5;
            Console.WriteLine("\n\ny=" + y);  // y=15
        }
```

```
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;
  }
}
```

Envío de parámetros por referencia



Descargar el ejemplo de parámetro por referencia (ref) en

http://www.itnuevolaredo.edu.mx/Takeyas/libroED/Prog7-2.rar

6.3.- Parámetros de salida (out)

- Son muy parecidos a los parámetros por referencia (`ref`)
- El parámetro `ref` debe ser inicializado antes de enviarse
- El parámetro `out` no es necesario inicializarlo antes de enviarlo, sino que lo inicializa el método.

```
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;
 }
}
```

Descargar el ejemplo de parámetro de salida (out) en

http://www.itnuevolaredo.edu.mx/Takeyas/libroED/Prog7-3.rar

7.- RECIBIENDO EL VALOR DEVUELTO POR UNA FUNCIÓN

- El "Tipo de dato" del método NO es "void".
- Dentro del método debe haber una sentencia "return" con algún valor del tipo de dato del método.
- Ejemplo (Al declararlo):

```
public string ConsultarEstado()
{
    return estado;
}
```

- Al llamar al método (desde el programa):

```
string estado_actual = miCarro.ConsultarEstado();
```

32

Llamadas de métodos que retornan valor (funciones)

```
static void Main(string[ ] args)
        {
            double Radio = 10, Area;
            Area = Funcion(Radio);
            Console.WriteLine("Area=" + Area);
            Console.ReadKey();
        }

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

Parámetro enviado a la función

Variable receptora

Valor devuelto por la Función

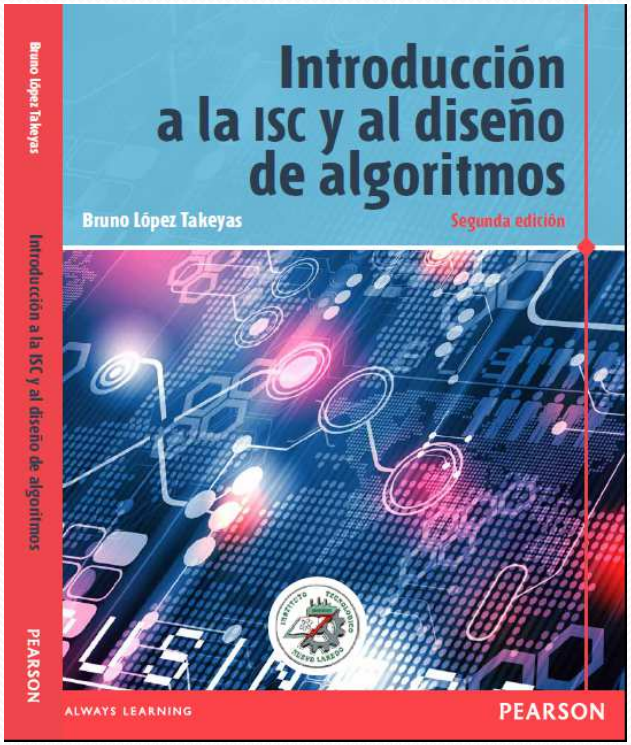
```
class Program
{
  static void Main(string[] args)
  {
    double Radio, Area;
    Console.Write("Teclee el valor del radio: ");
    Radio = double.Parse(Console.ReadLine());

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

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

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

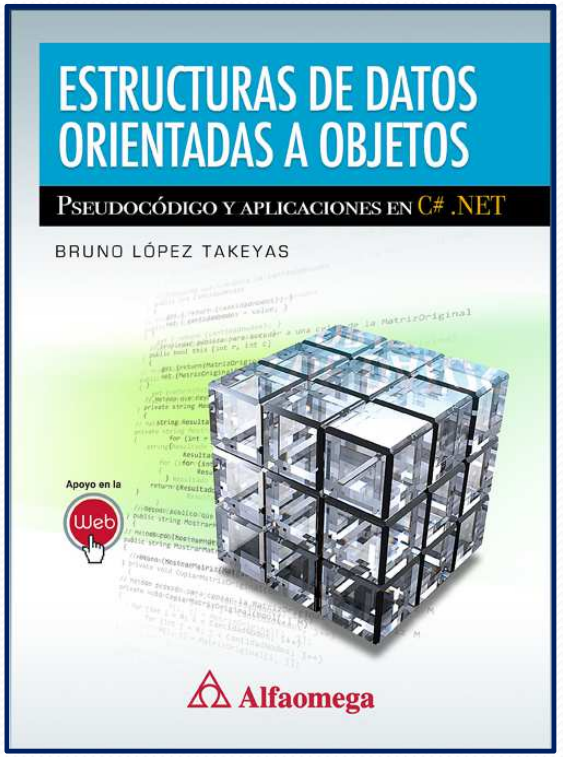
Descargar el ejemplo de recepción del valor devuelto por una función en

http://www.itnuevolaredo.edu.mx/Takeyas/RepasoFP/Prog4.rar

Otros títulos del autor

http://www.itnuevolaredo.edu.mx/Takeyas/Libro





bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas