
INTRODUCCIÓN
A LA INGENIERÍA EN SISTEMAS
COMPUTACIONALES Y AL
DISEÑO ORIENTADO A OBJETOS
Bruno López Takeyas
Pearson
FUNDAMENTOS DE
PROGRAMACIÓN
Capítulo 8.-Métodos
Bruno López Takeyas
Instituto Tecnológico de
Nuevo Laredo


CAPÍTULO 8
MÉTODOS
2

Preguntas detonadoras

- ¿Qué es un método?
- ¿Cuáles son los tipos de métodos? ¿En qué se parecen? ¿En qué difieren?
- ¿Cómo se envían datos a los métodos?
- ¿Cuándo se recomienda enviar parámetros por valor? ¿Cuándo por referencia? ¿Cuándo de salida?
- ¿Por qué son importantes los métodos para los objetos?
- ¿Puede haber métodos fuera de una clase?
- ¿A qué se refiere el ámbito de una variable?
- ¿Qué es una variable local? ¿y una global?
- ¿Qué es la firma de un método?

3

Métodos

- Contienen instrucciones para ejecutar al momento de ser invocados.
- Un método contiene:
  - Modificador de Acceso (Determina su visibilidad)
  - Tipo de dato (Devuelto al finalizar su ejecución)
  - Identificador (Nombre con el cual se invoca)
  - Parámetros (Cero o mas variables que recibe el método)

4

Métodos

- En C# las subrutinas se conocen como métodos, se codifican como parte de una clase y se clasifican en …

MÉTODOS
- Procedimientos – NO devuelven valor
- Funciones – Devuelven un valor

5

Ubicación de los métodos

- Pueden estar encapsulados dentro de una clase
- Pueden ser independientes (fuera de las clases)
- Independientemente de su ubicación, pueden interactuar

LlamadaTelefonica
- _intDuracion: int
+ Duracion { get; set; } : int + CalcularCosto() : int

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

static double Calcular(double dblR)
{
    return (Math.PI * dblR * dblRr);
}
```

6

Llamadas a los métodos

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

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

7

Llamadas de procedimientos

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

        static void HacerProcedimiento( )
        {
              Console.Write("Tec Laredo");
        }
     }
```

El procedimiento **NO** devuelve valor

8

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

9

Envío de parámetros por valor

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

La variable "y" recibe el valor de la variable "x"

10

Envío de parámetros por referencia

```
static void Main(string[] args)
{
  int x = 10; // Se declara e inicializa x
  HacerAlgo(ref x);  // Se envía la referencia de x
  Console.WriteLine("x=" + x); // x=15
}

static void HacerAlgo(ref int y)
{
  y += 5;
  Console.WriteLine("y=" + y);  // y=15
}
```

¡ Se modificó el valor de la variable x !

Parámetros por referencia (apuntador)

Memoria RAM

Dirección	*Valor*	*Variable*
FA31:B278	5	x
…		
FA31:C45C	13	y
…		
FA31:D2A8	FA31:C45C	a
…		

12

Parámetros de salida (*out*)

```
static void Main(string[] args)
{
  int x; //Se declara pero NO se inicializa la variable x
  HacerAlgo(out x); // Envía referencia de la variable x
  Console.WriteLine("x=" + x); // x=45
}

static void HacerAlgo(out int y)
{
  y = 45;
  Console.WriteLine("y=" + y);  // y=45
}
```

¡ Se modificó el valor de la variable x !

Envío de varios parámetros

```
static void Main(string[] args)
{
  int a=5; // Se declara e inicializa la variable a
  double b=3.2; // Se declara e inicializa la variable b
  string c; // Se declara pero NO se inicializa la variable c

  HacerAlgo(a, ref b, out c);
  Console.WriteLine("a=" + a); // a=5
  Console.WriteLine("b=" + b); // b=6.4
  Console.WriteLine("c=" + c); // c="Tec Laredo"
}

static void HacerAlgo(int x, ref double y, out string z)
{
   x+=8;
   y*=2;
   z="Tec Laredo";
}
```

Firma de un método

- Se conoce como la firma de un método al conjunto de parámetros que recibe y se forma de:
  - *Cantidad de parámetros recibidos*
  - *Orden de los parámetros*
  - *Tipos de datos*
- Deben coincidir los parámetros enviados con la firma del método

15

Sintaxis de los métodos

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

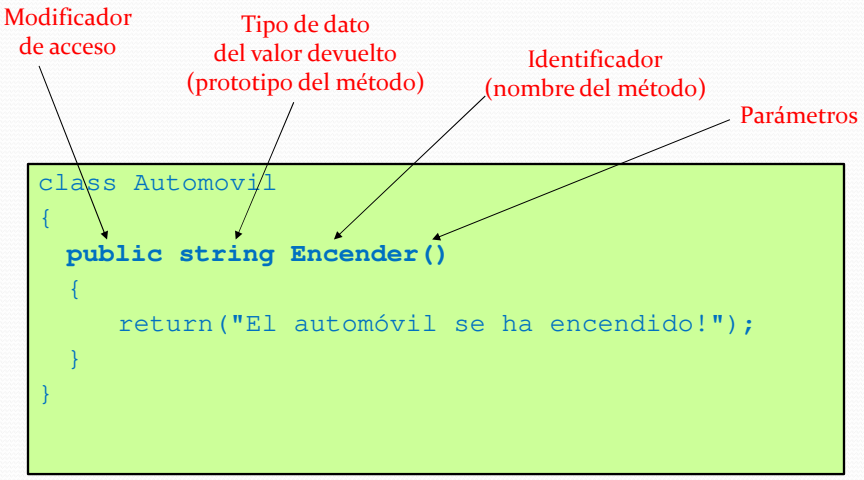
- *Ejemplo:*

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

void significa que NO devuelve valor (procedimiento)

16

Ejemplo de un método (en la clase)




```
class Automovil
{
  public string Encender()
  {
     return("El automóvil se ha encendido!");
  }
}
```

17

Ejemplo de un procedimiento independiente (no encapsulado)

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

18

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));
}
```

19

Métodos que retornan valores

- 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()
{
    string strEstado="Disponible";
    return strEstado;
}
```

- Al llamar al método (desde el programa):

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

20

20

Llamadas de métodos que retornan valor (funciones)

```
static void Main(string[ ] args)
{
  double dblRadio = 10, dblArea;
  dblArea = CalcularArea(dblRadio);
    Console.WriteLine("Area=" + dblArea);
}
 static double CalcularArea(double dblR)
 {
   return (Math.PI * Math.Pow(dblR, 2));
 }
```

Parámetro enviado a la función

Variable receptora

Valor devuelto por la función

21

Invocando al método (en el programa)

```
Carro miCarro = new Carro();
miCarro.Encender();
```

Nombre del objeto

Nombre del método

Parámetros

22

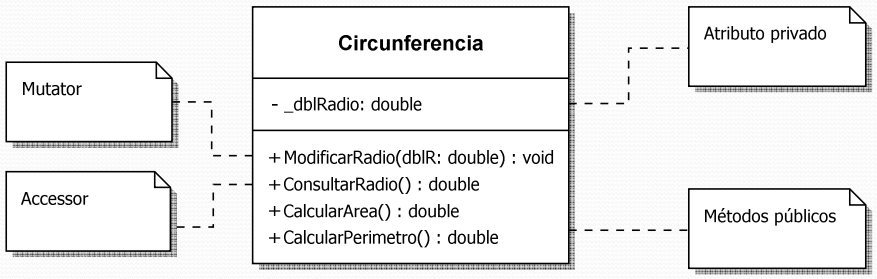
Invocando métodos

```
class Arbol
{
    public void Podar( )
    {
        . . .
    }
}
```

```
Arbol miFresno = new Arbol(); // Se crea el objeto
//Se invoca el método Podar() del objeto miFresno
miFresno.Podar();
```

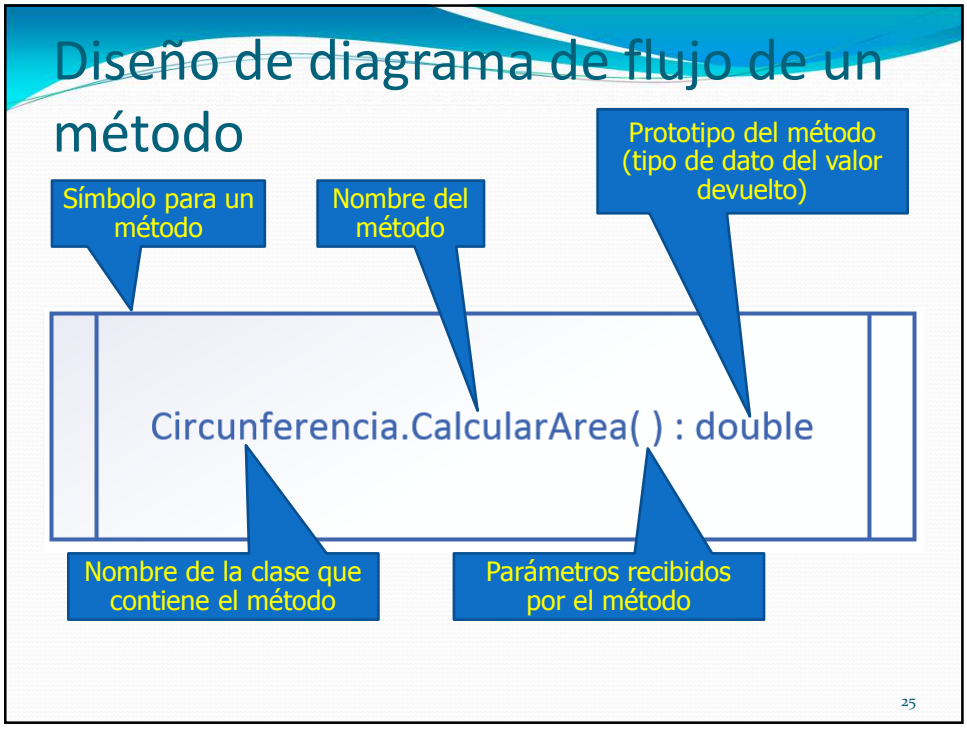
23

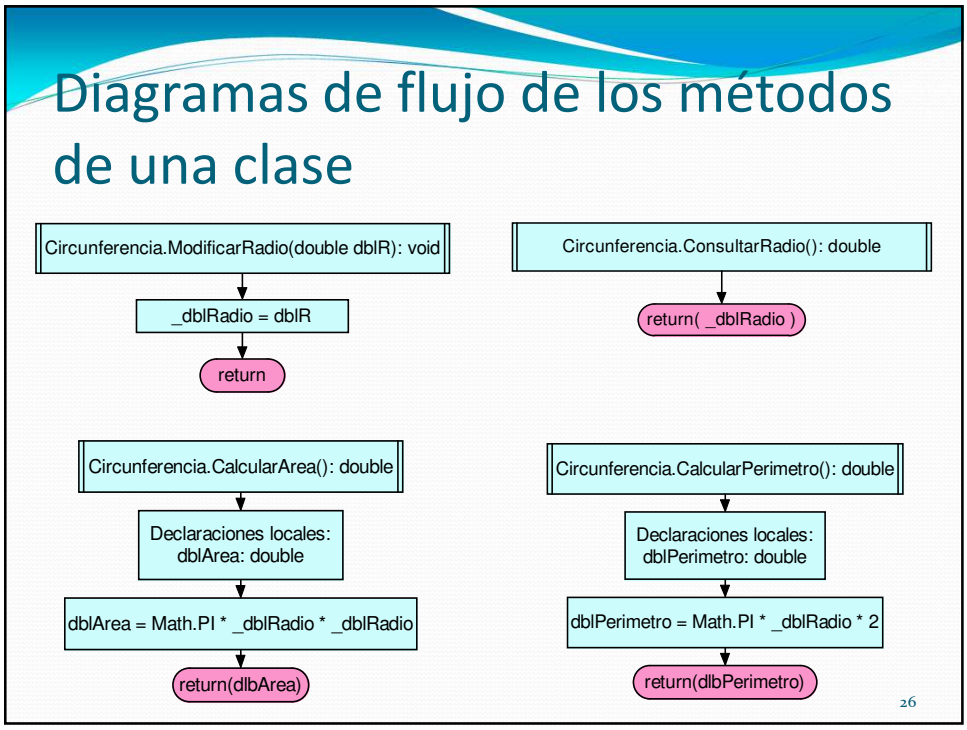
Uso de mutator y accessor



Al trabajar con objetos, primero deben introducirse los valores de sus atributos y después ejecutar las acciones invocando sus métodos.

24


Diseño de diagrama de flujo de un método
Símbolo para un método
Nombre del método
Prototipo del método (tipo de dato del valor devuelto)
Circunferencia.CalcularArea( ) : double
Nombre de la clase que contiene el método
Parámetros recibidos por el método
25


Diagramas de flujo de los métodos de una clase
Circunferencia.ModificarRadio(double dblR): void
_dblRadio = dblR
return
Circunferencia.ConsultarRadio(): double
return(_dblRadio)
Circunferencia.CalcularArea(): double
Declaraciones locales: dblArea: double
dblArea = Math.PI * _dblRadio * _dblRadio
return(dlbArea)
Circunferencia.CalcularPerimetro(): double
Declaraciones locales: dblPerimetro: double
dblPerimetro = Math.PI * _dblRadio * 2
return(dlbPerimetro)
26

Codificación de la clase

```
class Circunferencia
    {
        // Declaración del atributo privado
        private double _dblRadio;

        // Mutator
        public void ModificarRadio(double dblR)
        {
            _dblRadio = dblR;
        }

        // Accessor
        public double ConsultarRadio()
        {
            return (_dblRadio);
        }

        // Método público para calcular el área
        public double CalcularArea()
        {
            // Declaración de variable local
            double dblArea;

            dblArea=Math.PI * _dblRadio * _dblRadio;

            return (dblArea); // Devuelve el resultado
        }

        // Método público para calcular el perímetro
        public double CalcularPerimetro()
        {
            // Declaración de variable local
            double dblPerimetro;

            dblPerimetro=Math.PI * _dblRadio * 2;

            return (dblPerimetro);  // Devuelve el resultado
        }
    }
```

27

Ámbito de las variables

- El ámbito de una variable define dónde puede usarse esa variable (según dónde se declare)
- 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

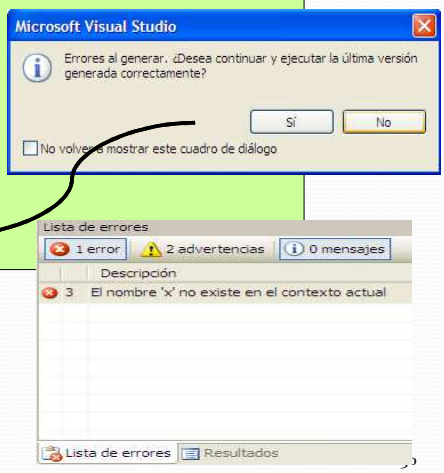
Locales – Se declaran y utilizan dentro de un contexto específico. No se puede hacer referencia a ellas fuera de la sección de código donde se declara.

Globales – Se declaran fuera del cuerpo de cualquier método

29

Ámbito de variables en un ciclo for

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

Console.Write(x);
```



Variables locales en métodos

- Son variables que se declaran dentro de un método
- Una variable local solamente puede ser utilizada por el método que la declara
- Los parámetros que recibe un método son variables locales
- Las variables declaradas dentro del método **`Main()`** son locales

Ejemplo de variables locales

```
class Program
{
   static void Main(string[] args)
   {
        int x=10; // Variable local
         . . .
   }
   static void HacerAlgo(int y) // Variable local
   {
         int r=5; // Variable local
         . . .
   }
}
```

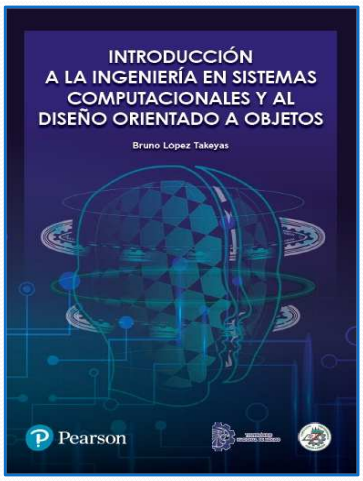
Variables globales

- Son variables que se declaran fuera de cualquier método (incluyendo **`Main()`**)
- Una variable global puede ser utilizada por cualquier método
- Se declaran antes del método **`Main()`**

Ejemplo de variables globales

```
class Program
{
    static int a=13; // Variables globales
    static double b=4.5;

    static void Main(string[] args)
    {
          int x=10; // Variable local
          . . .
    }
    static void HacerAlgo(int y) // Variable local
    {
          int r=5; // Variable local
          . . .
    }
}
```

LECTURA



Para reforzar este tema se recomienda la lectura de:

Capítulo 8.- Métodos

35

TAREA 5.1

Resolver la

Tarea 5.1.- Métodos

en MS Teams

Se contabilizará la tarea si se obtiene calificación aprobatoria

36

Captura e impresión de datos

Recuerde que NO es recomendable incluir sentencias para capturar o desplegar datos en pantalla dentro de un método de una clase, ya que se restringe el uso de dicha clase exclusivamente para la plataforma en la que se está trabajando y no se puede reutilizar en otras plataformas

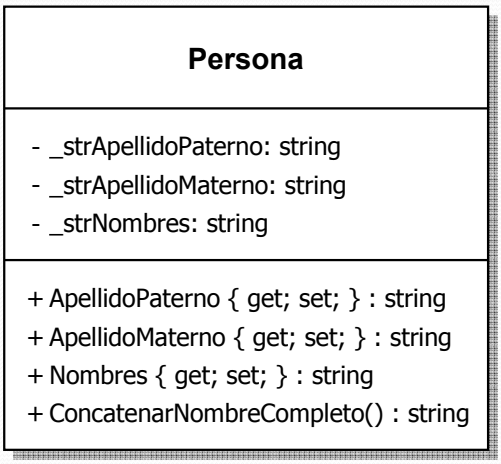
37

Concatenación de cadenas

- Es un proceso que consiste en "agregar" subcadenas al final de una cadena
- A una variable de tipo `string` se le pueden agregar datos o variables de tipo `string`.
- P. ejem.
  - *Capturar los apellidos y el nombre de una persona para concatenarlos e imprimir el nombre completo.*

38

Diseño de la clase



39

Codificación de la clase

```
class Persona
    {
        private string _strApellidoPaterno;

        public string ApellidoPaterno
        {
            get { return _strApellidoPaterno; }
            set { _strApellidoPaterno = value; }
        }

        private string _strApellidoMaterno;

        public string ApellidoMaterno
        {
            get { return _strApellidoMaterno; }
            set { _strApellidoMaterno = value; }
        }

        private string _strNombres;

        public string Nombres
        {
            get { return _strNombres; }
            set { _strNombres = value; }
        }

        public string ConcatenarNombreCompleto()
        {
            return (Nombres + " " + ApellidoPaterno + " " + ApellidoMaterno);
        }
    }
```

40

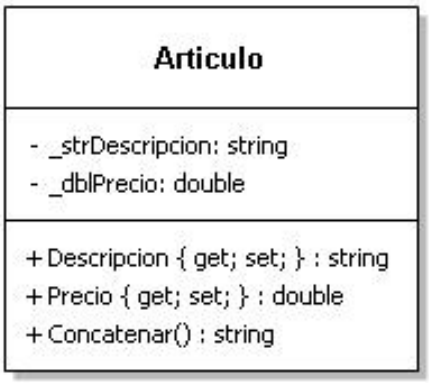
Codificación del método principal

```
static void Main(string[] args)
{
    Persona miPersona = new Persona();
    string strNombreCompleto;

    Console.WriteLine("Teclee el apellido paterno: ");
    miPersona.ApellidoPaterno = Console.ReadLine();

    Console.WriteLine("Teclee el apellido materno: ");
    miPersona.ApellidoMaterno = Console.ReadLine();

    Console.WriteLine("Teclee el nombre: ");
    miPersona.Nombres = Console.ReadLine();

    strNombreCompleto = miPersona.ConcatenarNombreCompleto();

    Console.WriteLine("Nombre completo = "+strNombreCompleto);
    Console.ReadKey();
}
```

41

¿Cómo concatenar datos que no son cadenas (*string*)?

- Suponga que desea concatenar la descripción y el precio de un artículo de un supermercado



42

Codificación de la clase

```
class Articulo
{
   private string _strDescripcion;
   public string Descripcion
   {
      get { return _strDescripcion; }
      set { _strDescripcion = value; }
   }

   private double _dblPrecio;
   public double Precio
   {
      get { return _dblPrecio; }
      set { _dblPrecio = value; }
   }

   public string Concatenar()
   {
      return ("El artículo " + Descripcion + " cuesta " + Precio.ToString("C"));
   }
}
```

43

CUESTIONARIO

Contestar el

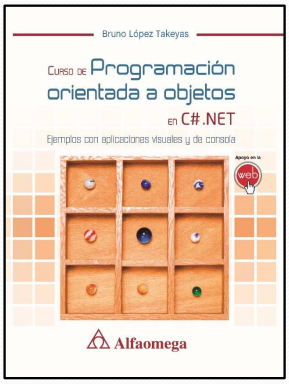
Cuestionario 05.- Métodos

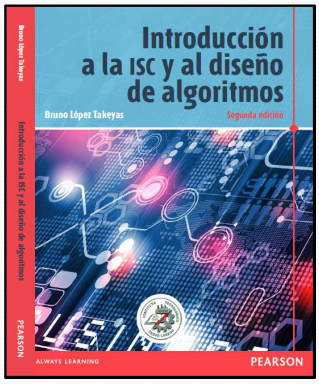
en la plataforma EaD IT Nuevo Laredo - Synesi

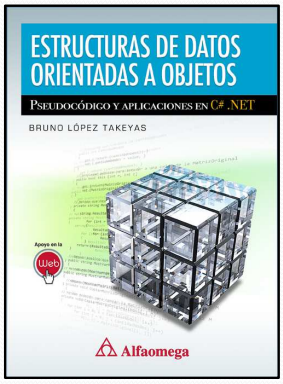
44

Otros libros del autor

https://nlaredo.tecnm.mx/takeyas/Libro







bruno.lt@nlaredo.tecnm.mx

Bruno López Takeyas