

# C# .Net

## Miembros estáticos

# Miembros estáticos (de la clase)

- Los miembros estáticos son miembros ligados a la clase (o tipo) como tal y no a los objetos (instancias) de la misma.
- Para definirlos basta con preceder la definición de ese miembro con la palabra reservada **static**

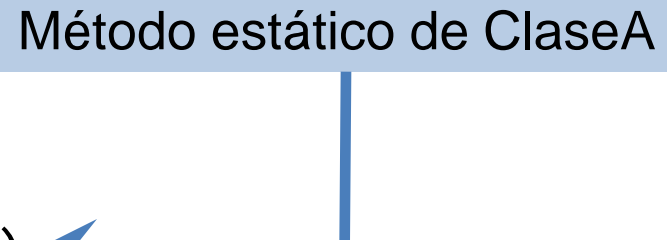
**Observe que el método **Main** que se utiliza como punto de entrada a un programa es un método estático**

# Miembros estáticos (de la clase)

Codifique la siguiente clase

Método estático de ClaseA

```
class ClaseA
{
    public static void Imprimir()
    {
        Console.Write("método estático de ClaseA");
    }
}
```

A blue rectangular callout box with the text "Método estático de ClaseA" is positioned above the code. A blue line extends from the bottom of the box, turning left to point at the "Imprimir()" method signature in the code block.

# Miembros estáticos (de la clase)

- La sintaxis a usar para acceder a un miembro de clase es `<nombreClase>.<miembro>`, por ejemplo:

`ClaseA.Imprimir()`

# Miembros estáticos (de la clase)

Codifique el siguiente programa y ejecute para comprobar su funcionamiento

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        ClaseA.Imprimir();
        Console.ReadKey(true);
    }
}
```

A screenshot of a code editor window with a black background. The text 'método estático de ClaseA' is written in a light blue, monospaced font. The editor has a vertical scrollbar on the right side, and the text is positioned near the top of the visible area.

método estático de ClaseA

# Miembros estáticos (de la clase)

Agregue las líneas sombreadas al método **Main** y verifique el mensaje del compilador

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        ClaseA.Imprimir();
        ClaseA a = new ClaseA();
        a.Imprimir();
        Console.ReadKey(true);
    }
}
```

**No se puede obtener acceso al miembro 'ClaseA.Imprimir()' con una referencia de instancia; califíquelo con un nombre de tipo en su lugar**

# Campos estáticos

- Un campo estático puede considerarse como una especie de variable global compartida por todas las instancias de la clase, o incluso por otros objetos si está marcado como público.

```
class ClaseA
```

```
{
```

```
    public static void Imprimir() {  
        Console.Write( "El valor de Y es: {0}", Y );
```

```
    }
```

```
    public int X;
```

```
    public static int Y;
```

```
}
```

**Modifique ClaseA**

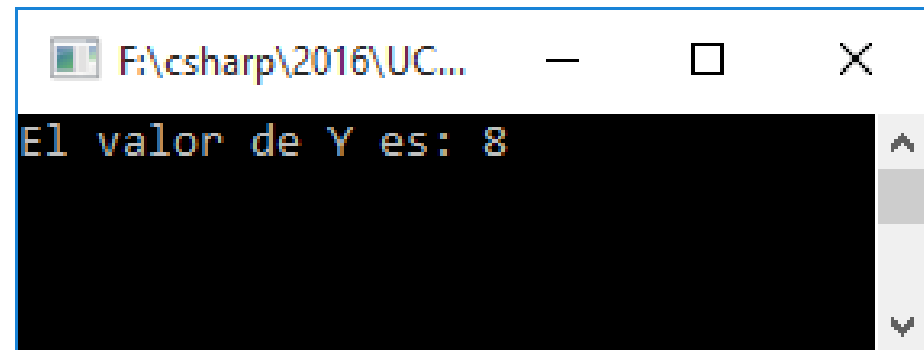
Variable de instancia

Variable de clase

# Campos estáticos

Modifique el método **Main** y ejecute

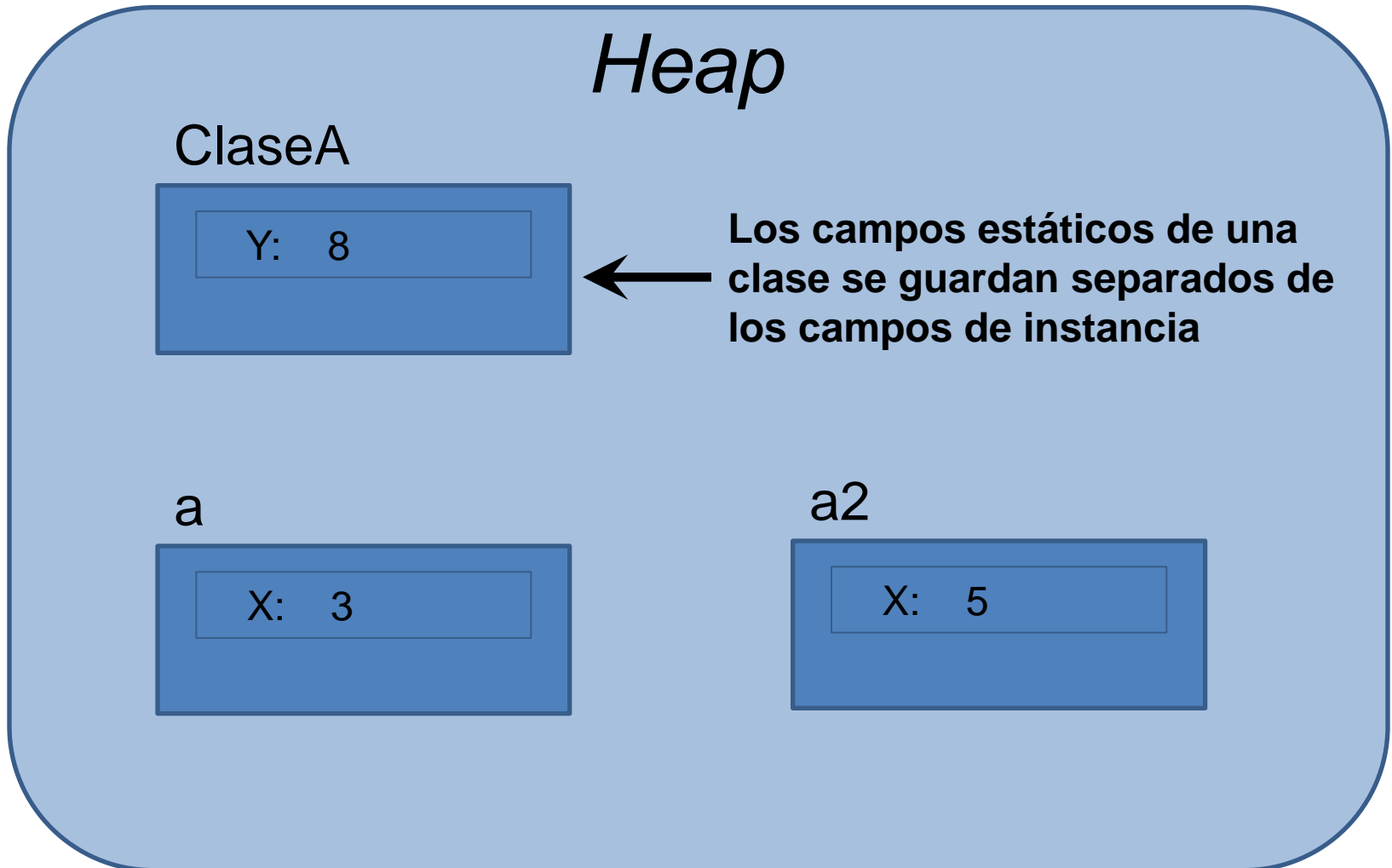
```
public static void Main(string[] args)
{
    ClaseA a = new ClaseA();
    ClaseA a2 = new ClaseA();
    a.X = 3;
    ClaseA.Y = a.X;
    a2.X = 5;
    ClaseA.Y += a2.X;
    ClaseA.Imprimir();
    Console.ReadKey(true);
}
```

A screenshot of a Windows console window. The title bar shows the file path 'F:\csharp\2016\UC...'. The console output displays the text 'El valor de Y es: 8' in a monospaced font. The window has standard Windows controls (minimize, maximize, close) in the title bar and a vertical scrollbar on the right side.

```
F:\csharp\2016\UC...
El valor de Y es: 8
```



# Campos estáticos



# Campos estáticos

- Modifique el método estático Imprimir y verifique el mensaje del compilador

```
class ClaseA
{
    public static void Imprimir() {
        Console.Write( "El valor de X es: {0}", X );
    }
    public int X;
    public static int Y;
}
```

Se requiere una referencia de objeto para el campo, método o propiedad no estáticos 'ClaseA.X'

**En los métodos estáticos sólo están visibles los campos y métodos estáticos de la clase**

# Miembros estáticos (de la clase)

- ¿Qué sentencias son incorrectas?

```
class programa{  
    public static void Main(){  
        ClaseA a= new ClaseA();  
        a.X = 2;  
        ClaseA.Y = 1;  
        a.Y = 3;  
        ClaseA.X = 5;  
        Console.ReadKey();  
    }  
}  
  
class ClaseA  
{  
    public int X;  
    public static int Y;  
}
```

**Y** es una variable estática, no está ligada a la instancia **a**

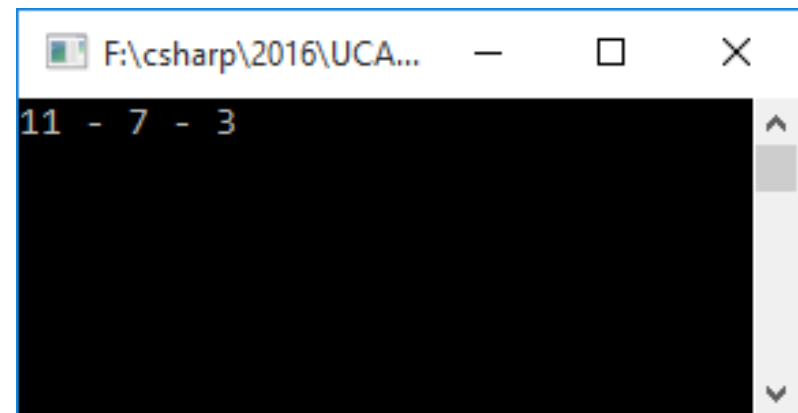
**X** es una variable de instancia no puede accederse desde la clase **ClaseA**

# Miembros estáticos (de la clase)

- ¿Cuál será la salida por Consola?

```
using System;
class programa{
    public static void Main(){
        ClaseA a1= new ClaseA(), a2=new ClaseA();
        a1.Incrementa(5);a1.Incrementa(6);
        a2.Incrementa(7);
        Console.WriteLine("{0} - {1} - {2}",
                           a1.X, a2.X, ClaseA.Y);
        Console.ReadKey();
    }
}
class ClaseA {
    public int X=0;
    public static int Y=0;
    public void Incrementa(int valor){
        X += valor;
        ClaseA.Y++;
    }
}
```

La variable de clase **Y** se está utilizando para contar cuántas veces es invocado el método de instancia **Incrementa()** por algún objeto sin importar cuál sea éste.



A screenshot of a Windows console window. The title bar shows the file path 'F:\csharp\2016\UCA...'. The console output displays the result of the program execution: '11 - 7 - 3'. The numbers are color-coded: '11' is blue, '7' is green, and '3' is red, matching the format string in the code. The rest of the console is black with white text.

# Miembros estáticos (de la clase)

```
class ClaseA {  
    public int X=0;  
    public static int Y=0;  
    public void Incrementa(int valor){  
        X += valor;  
        ClaseA.Y++;  
    }  
}
```

Los miembros estáticos pueden accederse desde la propia clase sin anteponer el nombre de la misma.

```
class ClaseA {  
    public int X=0;  
    public static int Y=0;  
    public void Incrementa(int valor){  
        X += valor;  
        Y++;  
    }  
}
```

Sin embargo, anteponer el nombre de la clase puede hacer el código más claro y fácil de leer.

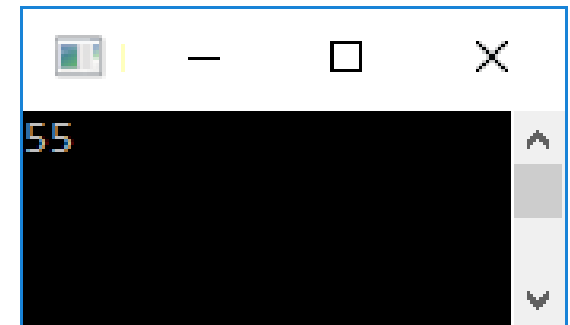
# Constructores estáticos

- Llevan el modificar **static**.
- No pueden tener parámetros ni modificadores de accesibilidad.
- No pueden sobrecargarse
- No pueden ser invocados explícitamente.
- Sólo son invocados por el sistema.

# Constructores estáticos - ejemplo

```
class ClaseA
{
    public static int Sum=0;
    static ClaseA()
    {
        for(int i = 1; i<=10; i++)
        {
            ClaseA.Sum += i;
        }
    }
}

class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine(ClaseA.Sum);
    }
}
```



# Constantes


Las constantes definidas en una clase son en realidad estáticas (aunque no se puede utilizar el modificador **static**)

```
class ClaseA
{
    public const double PI = 3.1416;
}
```

```
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine(ClaseA.PI);
        ClaseA a = new ClaseA();
        Console.WriteLine(a.PI);
        Console.ReadKey();
    }
}
```



OK



Error de compilación: No se puede obtener acceso al miembro 'ClaseA.PI' con una referencia de instancia; califíquelo con un nombre de tipo en su lugar



# Constantes

La asignación de una constante se resuelve en tiempo de compilación por eso sólo se pueden utilizar expresiones simples

```
class ClaseA
{
    public const double Y = 5 * 3 / 2.0;
    public const int SUM = Calcular();
    public const int[] vector = new int[] {1,2,3};

    private static int Calcular()
    {
        . . .
    }
}
```

OK

Error de compilación:  
La expresión que se  
asigne a 'ClaseA.SUM'  
debe ser constante

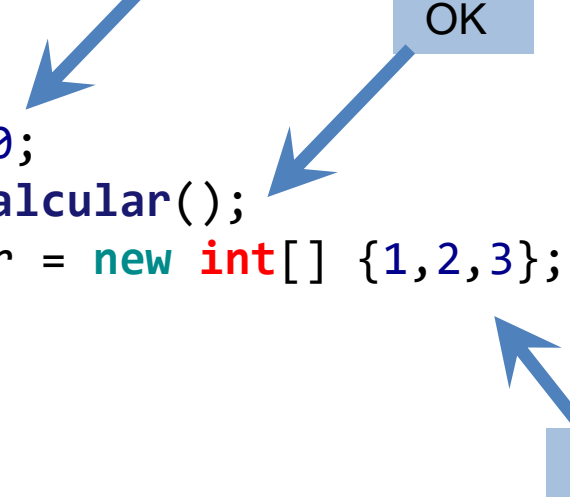
Un campo const de un tipo de referencia  
distinto de String sólo se puede  
inicializar con null

# Campos readonly

La asignación de los campos **readonly** se realiza en tiempo de ejecución, pueden ser estáticos o de instancia y sólo pueden modificarse en su declaración o en un constructor

```
class ClaseA
{
    public const double Y = 5 * 3 / 2.0;
    public static readonly int SUM = Calcular();
    public static readonly int[] vector = new int[] {1,2,3};

    private static int Calcular()
    {
        . . .
    }
}
```



# Clases estáticas

- Se puede aplicar la palabra clave **static** directamente a nivel de clase definiendo así una clase estática
- Las clases estáticas sólo pueden poseer miembros estáticos
- No es posible instanciar objetos de una clase estática.
- Ejemplos de clases estáticas: **Console**, **File**, **Directory**, **Math**, etc.

# Clases estáticas


```
static class TimeUtil
{
    public static void ImprimirHora()
    {
        Console.WriteLine("{0:hh:mm:ss}", DateTime.Now);
    }
    public static void ImprimirFecha()
    {
        Console.WriteLine("{0:dd/MM/yyyy}", DateTime.Today);
    }
}
```

Marcando a `TimeUtil` como estática nos beneficiamos del control que hace el compilador que nos avisa si intentamos instanciar un objeto `TimeUtil` por error

# Clases estáticas

Intento fallido de instanciar una clase estática

```
using System;  
class Program  
{  
    public static void Main(string[] args)  
    {  
        TimeUtil tu = new TimeUtil();  
        Console.ReadKey();  
    }  
}
```



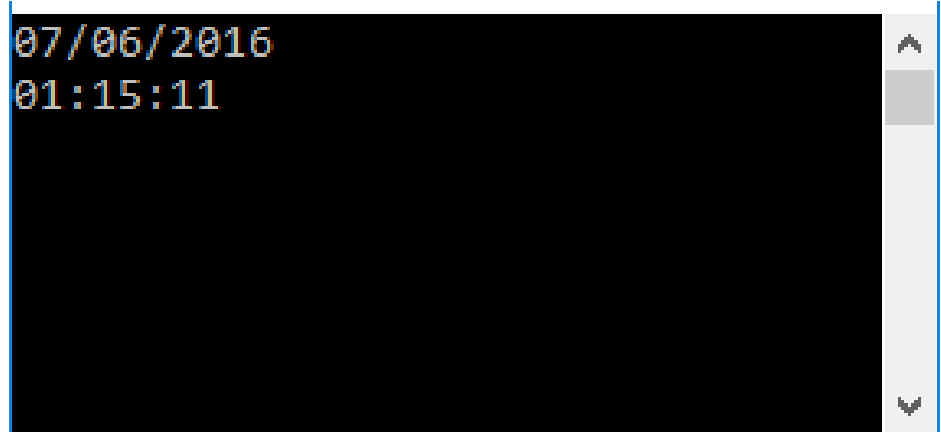
Esta línea produce dos errores de compilación:

- No se puede crear ninguna instancia de la clase estática 'TimeUtil'
- No se puede declarar una variable de tipo estático 'TimeUtil'

# Clases estáticas

Utilización correcta de la clase estática `TimeUtil`

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        TimeUtil.ImprimirFecha();
        TimeUtil.ImprimirHora();
        Console.ReadKey();
    }
}
```

A screenshot of a console window with a black background. It displays two lines of text in a monospaced font: the first line is '07/06/2016' and the second line is '01:15:11'. Both lines are rendered in a light blue or cyan color. The window has a standard scrollbar on the right side.

07/06/2016  
01:15:11

# Encapsulamiento

Acceso a miembros privados

# El rol del encapsulamiento

- El **encapsulamiento** es uno de los **pilares** de la **programación orientada a objetos**
- Es la capacidad del lenguaje para **ocultar detalles de implementación** hacia fuera del objeto
- En estrecha relación con la noción de encapsulamiento está la idea de **la protección de datos**. Idealmente, el estado de los objetos debería especificarse usando **campos privados**.



# Acceso a campos privados

- Implementar una clase `Cuadrado` con un campo privado de tipo `double` llamado *Lado*

```
class Cuadrado
{
    private double Lado;
}
```

- Sin cambiar su modificador de acceso y sin definir constructores **¿Qué mecanismo podríamos utilizar que nos permita asignar el campo `Lado` desde fuera de la clase `Cuadrado`?**

# Acceso a campos privados

- Defina el método **SetLado** que reciba como parámetro el valor para establecer el campo *Lado*.

```
public void SetLado (double value) {  
    Lado = value;  
}
```

- ¿Cómo podríamos obtener el valor del campo desde *Lado* desde fuera de la clase **Cuadrado**?

# Acceso a campos privados

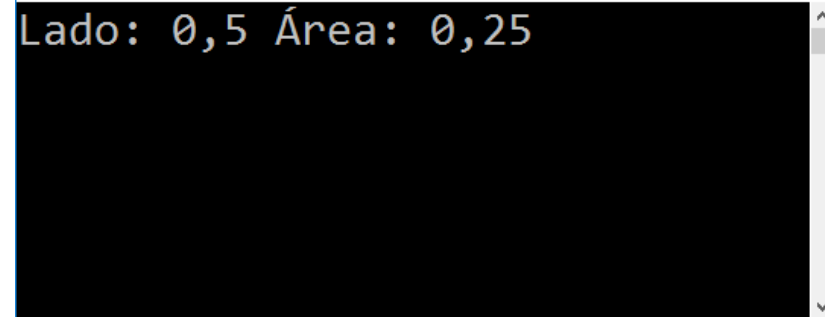
- Defina los método **GetLado** y **GetArea** para obtener el valor del lado y el área respectivamente.

```
public double GetLado(){  
    return Lado;  
}  
public double GetArea(){  
    return Lado * Lado;  
}
```

# Acceso a campos privados

Verifique su funcionamiento.

```
class Program
{
    public static void Main(string[] args)
    {
        Cuadrado c = new Cuadrado();
        c.SetLado(0.5);
        Console.WriteLine("Lado: {0} Área: {1}",
                           c.GetLado(),
                           c.GetArea());
        Console.ReadKey(true);
    }
}
```



Lado: 0,5 Área: 0,25

# Acceso a campos privados

La utilización de métodos *getters* y *setters* , como la ilustrada anteriormente, no está indicada en la **plataforma .NET**

**En su lugar utilice siempre  
PROPIEDADES**

# Propiedades

- Una **propiedad** es una mezcla entre el concepto de **campo** y el concepto de **método**.
- Externamente es accedida como si fuese un campo normal, pero internamente es posible asociar código a ejecutar en cada asignación o lectura de su valor
- Una propiedad no almacena datos, sino sólo se utiliza como si los almacenase

# Propiedades

## Definición de una propiedad – Sintaxis

```
modif_acceso tipoPropiedad nombrePropiedad
{
    get
    {
        <código de lectura>
    }
    set
    {
        <código de escritura>
    }
}
```

# Propiedades

- Dentro del bloque de código **set** se puede hacer referencia a un parámetro especial del mismo tipo de dato que la propiedad llamado **value**
- Dentro del código **get** se debe devolver siempre un objeto del tipo de dato de la propiedad.
- Una propiedad que sólo tenga el bloque **get** será una propiedad de **sólo lectura**.
- Una propiedad que sólo tenga el bloque **set** será una propiedad de **sólo escritura**



# Propiedades (Ejemplo)

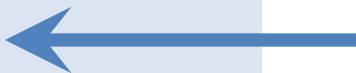
Modifique la clase `Cuadrado`

```
class Cuadrado  
{
```

```
    private double Lado;
```

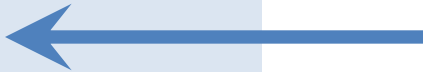
```
    public double Lado {  
        get { return Lado; }  
        set { Lado = value; }  
    }
```

Propiedad de  
lectura/escritura



```
    public double Area {  
        get { return Lado * Lado; }  
    }
```

Propiedad de  
sólo lectura

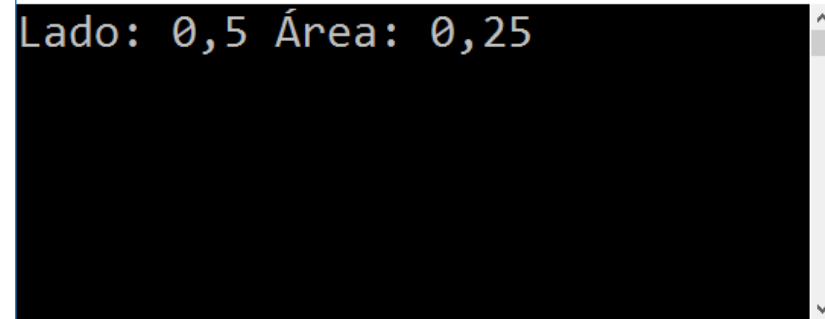


```
}
```

# Propiedades (Ejemplo)

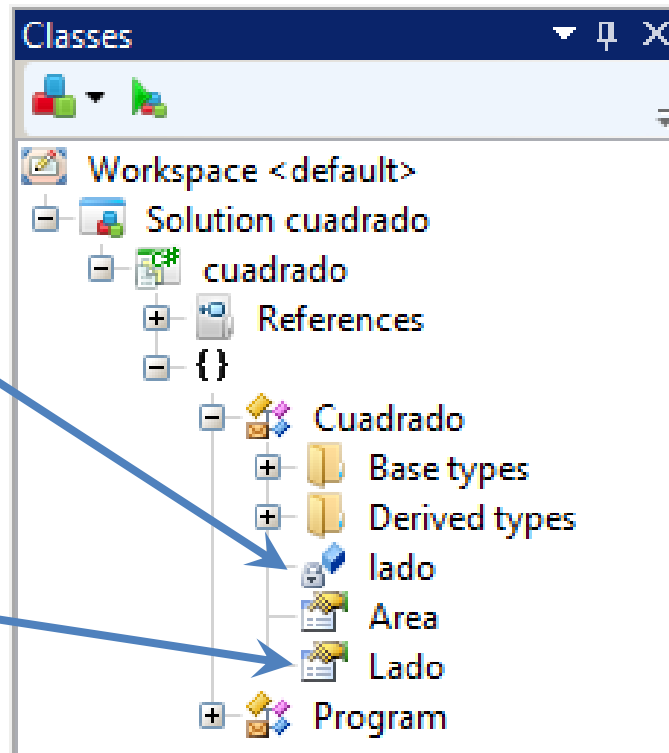
Modifique el método **Main**.

```
class Program
{
    public static void Main(string[] args)
    {
        Cuadrado c = new Cuadrado();
        c.Lado = 0.5;
        Console.WriteLine("Lado: {0} Área: {1}",
                          c.Lado, c.Area);
        Console.ReadKey(true);
    }
}
```



Lado: 0,5 Área: 0,25

# Propiedades



Prestando atención a los íconos podemos ver que las clases de la plataforma .NET nunca ofrecen campos públicos sino propiedades públicas, por ejemplo **Length** de un String, **Count** y **Capacity** de una colección, **Title** de Console, **Rank** de un arreglo, etc. son todas propiedades, no son campos.

# Propiedades (Ejemplo)

- Cambiamos la representación interna de la clase Cuadrado, sin embargo el programa anterior sigue funcionando de la misma manera

```
class Cuadrado{  
  
    private double area=0;  
  
    public double Lado {  
        get { return Math.Sqrt(area); }  
        set { area = value * value; }  
    }  
  
    public double Area {  
        get { return area; }  
    }  
}
```

# Propiedades

- Las propiedades permiten controlar el acceso a los campos privados ejecutando código convenientemente cada vez que se asigna o lee una propiedad
- **Ejercicio propuesto:** Modifique la clase cuadrado para que cualquier intento de asignar a la propiedad Lado un valor mayor que 100 no se lleve a cabo, asignando finalmente el valor 100 a esta propiedad.

# Propiedades implementadas automáticamente

- Con mucha frecuencia las propiedades están asociadas a campos privados de las clases

```
class Persona{  
    private string nombre;  
    public string Nombre {  
        get { return nombre; }  
        set { nombre = value; }  
    }  
}
```

# Propiedades implementadas automáticamente

- Para facilitar la tarea del programador C#3.0 introdujo las propiedades auto-implementadas
- Con ellas es posible declarar la propiedad sin declarar el campo asociado
- El compilador crea un campo oculto que asocia a la propiedad auto-implementada
- Debido a que no es posible acceder al campo oculto, sólo se implementan automáticamente propiedades de lectura/escritura

# Propiedades implementadas automáticamente

- La clase **Persona** puede re-escribirse de la siguiente manera:

```
class Persona{  
    public string Nombre{  
        get;  
        set;  
    }  
}
```



# Propiedades implementadas automáticamente

- Un buen hábito es no codificar campos públicos, sino hacer públicas las propiedades y privados todos los campos
- Usando las propiedades auto-implementadas puede hacerse fácilmente, sólo es necesario agregar `{get;set;}` al final del campo

```
class Persona{  
    public string Nombre;  
    public int Edad;  
    public int Dni;  
}
```

Clase con tres campos públicos

```
class Persona{  
    public string Nombre{get;set;}  
    public int Edad{get;set;}  
    public int Dni{get;set;}  
}
```

Clase con tres propiedades públicas

# Sintaxis de Inicialización de Objetos

- Para facilitar la inicialización de objetos, además de la utilización de constructores adecuados, C# provee una sintaxis específica para ello.
- Un inicializador consiste en una lista de elementos separada por comas encerradas entre llaves { } que sigue a la invocación del constructor
- Cada miembro de la lista mapea con un campo o propiedad pública del objeto al que le asigna un valor.

```
Persona p=new Persona(){Nombre="Juan",Edad=40,Dni=28765421};
```



También se podría invocar cualquier otro constructor que se haya definido en la clase Persona

# Ejemplo

Codifique las siguientes clases:

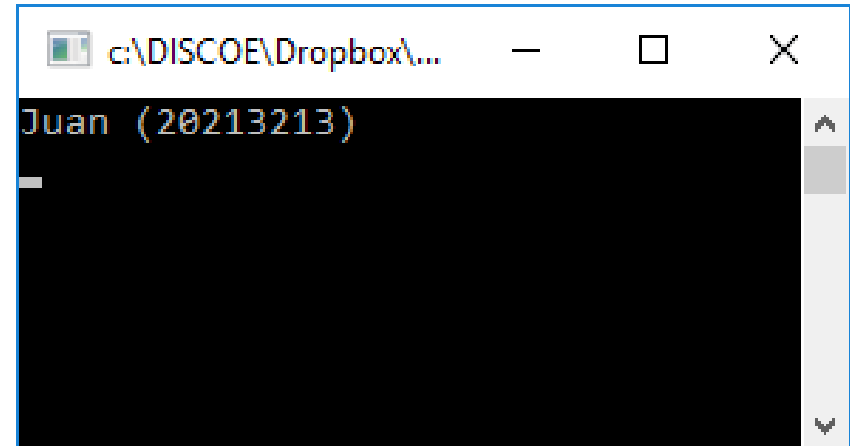
```
class Persona
{
    public string Nombre { get; set; }
    public int Dni { get; set; }
    public void imprimir() {
        Console.WriteLine("{0} ({1})", Nombre, Dni);
    }
}
```

```
class Familia
{
    public Persona Padre { get; set; }
    public Persona Madre { get; set; }
    public Persona Hijo { get; set; }
}
```

# Ejemplo

Codifique el siguiente programa:

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        Familia f=new Familia();
        f.Padre=new Persona() {Nombre="Juan",Dni=20213213};
        f.Madre=new Persona() {Dni=22123123, Nombre="María"};
        f.Hijo=new Persona() {Nombre="José",Dni=40111222};
        f.Padre.imprimir();
        Console.ReadKey(true);
    }
}
```



# Indizadores

Ahora se desea acceder a los miembros de una familia a través de un índice. De esta forma queremos que:

`f.Padre` se acceda por medio de `f[0]`

`f.Madre` se acceda por medio de `f[1]`

`f.Hijo` se acceda por medio de `f[2]`

# Indizadores

- Un **indizador** es una definición de cómo aplicar el operador (**[ ]**) a los objetos de una clase.
- A diferencia de los arreglos, los **índices** que se les pase entre corchetes **no están limitados a los enteros**, pudiéndose definir varios indizadores en una misma clase siempre y cuando cada uno tome un número o tipo de índices diferente (**sobrecarga**).

# Indizadores

## Sintaxis

```
TipoIndizador this[<indices>]  
{  
    get  
    {  
        <código Lectura>  
    }  
    set  
    {  
        <código Escritura>  
    }  
}
```

- Sintaxis similar a la sintaxis de las propiedades
- El nombre es siempre **this**
- En <índices> se indica cuáles son los índices que se pueden utilizar al acceder al indizador.
- Dentro del bloque **set** se utiliza el parámetro especial **value** del mismo tipo que el indizador

# Indizadores

Agregar el siguiente indizador de sólo lectura en la clase `Familia`

```
class Familia  
{
```

```
    . . .
```

```
    public Persona this[int i]  
    {  
        get {  
            if (i == 0) return this.Padre;  
            else if (i == 1) return this.Madre;  
            else if (i == 2) return this.Hijo;  
            else return null;  
        }  
    }  
}
```

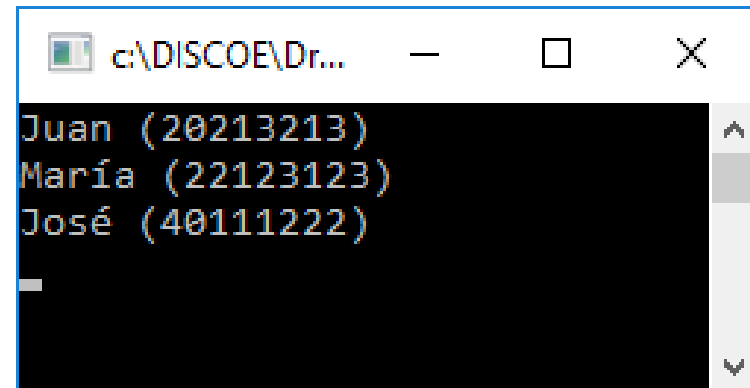
```
}
```



# Indizadores

Modifique el método **Main** de **Program** para verificar su funcionamiento.

```
public static void Main(string[] args)
{
    Familia f = new Familia();
    f.Padre = new Persona() { Nombre = "Juan", Dni = 20213213 };
    f.Madre = new Persona() { Dni = 22123123, Nombre = "María" };
    f.Hijo = new Persona() { Nombre = "José", Dni = 40111222 };
    for (int i = 0; i < 3; i++)
        f[i].imprimir();
    Console.ReadKey(true);
}
```



A screenshot of a Windows console window. The title bar shows the file path 'c:\DISCOE\Dr...'. The console output displays three lines of text: 'Juan (20213213)', 'María (22123123)', and 'José (40111222)'. Each line is printed on a new line, and there is a small cursor at the end of the last line.

# Indizadores

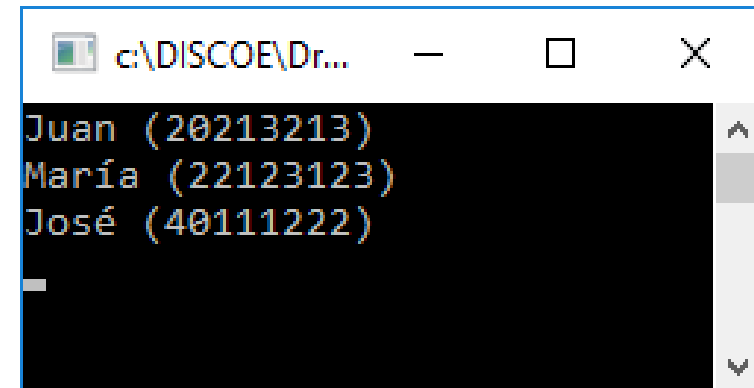
Agregar la parte set del indizador definido en la clase **Familia** para que sea de lectura/escritura

```
class Familia
{
    . . .
    public Persona this[int i] {
        get {
            if (i == 0) return this.Padre;
            else if (i == 1) return this.Madre;
            else if (i == 2) return this.Hijo;
            else return null;
        }
        set {
            if (i == 0) this.Padre = value;
            else if (i == 1) this.Madre = value;
            else if (i == 2) this.Hijo = value;
        }
    }
}
```

# Indizadores

Modifique el método **Main** de **Program** para verificar su funcionamiento.

```
public static void Main(string[] args)
{
    Familia f = new Familia();
    f.Padre = new Persona() { Nombre = "Juan", Dni = 20213213 };
    f[1] = new Persona() { Dni = 22123123, Nombre = "María" };
    f[2] = new Persona() { Nombre = "José", Dni = 40111222 };
    for (int i = 0; i < 3; i++)
        f[i].imprimir();
    Console.ReadKey(true);
}
```



A screenshot of a Windows console window. The title bar shows the file path 'c:\DISCOE\Dr...'. The console output displays three lines of text: 'Juan (20213213)', 'María (22123123)', and 'José (40111222)'. A cursor is visible on the line following the last output.

```
c:\DISCOE\Dr...
Juan (20213213)
María (22123123)
José (40111222)
_
```

# Indizadores

Agregar el siguiente indizador en la clase `Familia` con índice de tipo `string`

```
class Familia  
{
```

```
    . . .
```

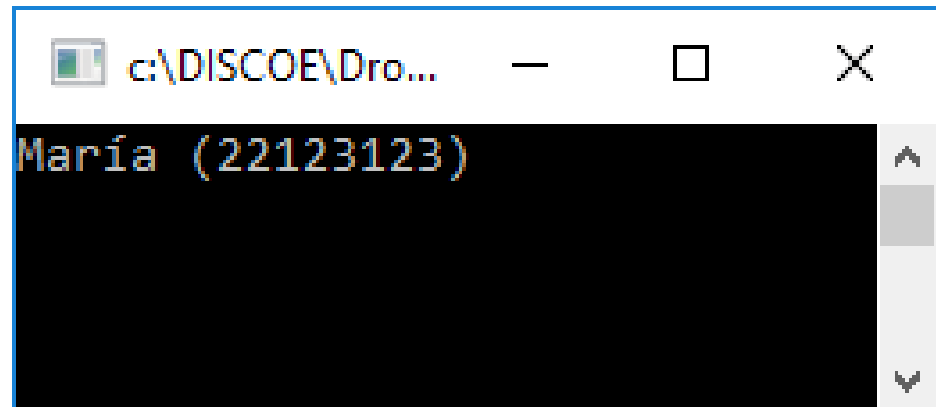
```
    public Persona this[string st]  
    {  
        get{  
            if (Padre.Nombre == st) return Padre;  
            else if (Madre.Nombre == st) return Madre;  
            else if (Hijo.Nombre == st) return Hijo;  
            else return null;  
        }  
    }  
}
```

¿Qué es lo que hace?

# Indizadores

Modifique el método **Main** de **Program** para verificar su funcionamiento.

```
public static void Main(string[] args)
{
    Familia f = new Familia();
    f.Padre = new Persona() { Nombre = "Juan", Dni = 20213213 };
    f[1] = new Persona() { Dni = 22123123, Nombre = "María" };
    f[2] = new Persona() { Nombre = "José", Dni = 40111222 };
    f["María"].imprimir();
    Console.ReadKey(true);
}
```



# Indizadores

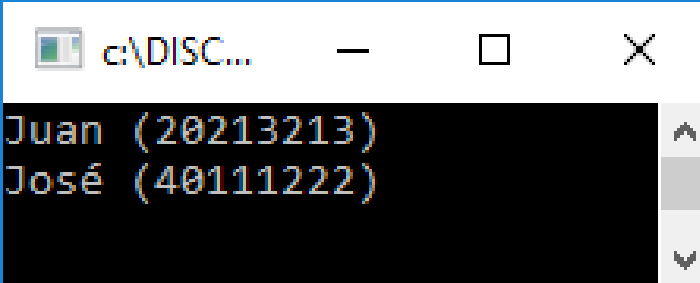
Agregar el siguiente indizador en la clase `Familia` con índice de tipo `char`, observe que devuelve algo de tipo `ArrayList`

```
public ArrayList this[char c]
{
    get{
        ArrayList result=new ArrayList();
        if (Padre.Nombre[0]==c) result.Add(Padre);
        if (Madre.Nombre[0]==c) result.Add(Madre);
        if (Hijo.Nombre[0]==c) result.Add(Hijo);
        return result;
    }
}
```

# Indizadores

Modifique el método **Main** de **Program** para verificar su funcionamiento.

```
using System;
using System.Collections;
class Program {
    public static void Main(string[] args) {
        Familia f = new Familia();
        f.Padre = new Persona() { Nombre = "Juan", Dni = 20213213 };
        f[1] = new Persona() { Dni = 22123123, Nombre = "María" };
        f[2] = new Persona() { Nombre = "José", Dni = 40111222 };
        ArrayList lista = f['J'];
        foreach(Persona p in lista) p.imprimir();
        Console.ReadKey(true);
    }
}
```



A screenshot of a Windows console window. The title bar shows a file icon and the text 'c:\DISC...'. The console output displays two lines: 'Juan (20213213)' and 'José (40111222)'. The text is in a monospaced font with some color coding (blue for names, red for DNI numbers). The window has standard Windows controls (minimize, maximize, close) in the top right corner.

# Indizadores con múltiples dimensiones

Ejemplo de clase con indizador de dos dimensiones

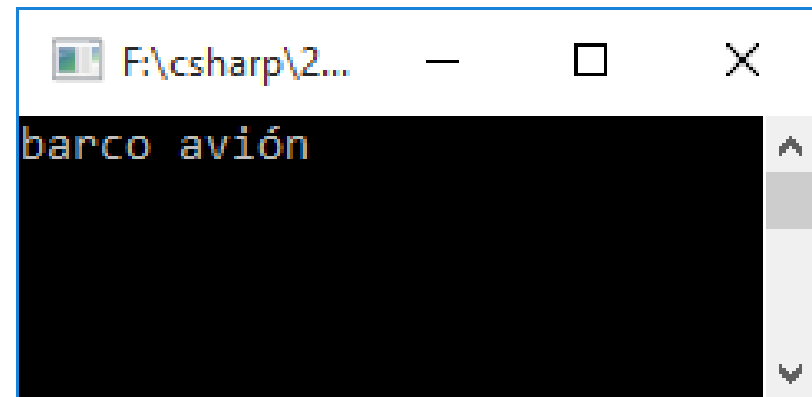
```
class Tablero {  
    Hashtable ht=new Hashtable();  
    public string this[char f, int c] {  
        set {  
            if (f >= 'A' && f <= 'J' && c >= 1 && c <= 10) {  
                ht[f.ToString()+c] = value;  
            }  
        }  
        get {  
            return ht[f.ToString()+c].ToString();  
        }  
    }  
}
```



# Indizadores con múltiples dimensiones

## Utilización de la clase **Tablero**

```
using System;
using System.Collections;
class Program
{
    public static void Main(string[] args)
    {
        Tablero t= new Tablero();
        t['H',3] = "barco";
        t['A',4] = "avión";
        Console.WriteLine("{0} {1}",t['H',3],t['A',4]);
        Console.ReadKey(true);
    }
}
```



A screenshot of a Windows console window. The title bar shows the file path 'F:\csharp\2...' and standard window controls (minimize, maximize, close). The console output displays the text 'barco avión' in a monospaced font, with 'barco' and 'avión' separated by a space. The text is rendered in a light color against a dark background.