POO

Programación Orientada a Objetos

Introducción

- Vayamos al grano, la teoría de la POO es idéntica a la vista en Java, por lo tanto, me voy a centrar en la parte práctica del tema.

Definición de Clases.C#

- Las clases permiten crear nuevos tipos de datos.
- Las clases se definen dentro de un namespace mediante la palabra clave **class seguida del** identificador de la clase.
- A diferencia de los demás tipos de datos las clases pueden **derivarse unas de otras.**
- Contienen datos, métodos y propiedades
- Una clase puede incluir otras clases.
- Las clases pueden implementar interfaces.
- También pueden incluir delegados y eventos.

Modificadores de acceso

- **public**
  - La clase o miembro es accesible en cualquier ámbito.
- **Protected**
  - Se aplica sólo a miembros de la clase. Indica que sólo es accesible desde la propia clase y desde las clases derivadas.
- **Private**
  - Se aplica a miembros de la clase. Un miembro privado sólo puede utilizarse en el interior de la clase donde se define, por lo que no es visible tampoco en clases derivadas.
- **Internal**
  - La clase o miembro sólo es visible en el proyecto (ensamblado) actual.
- **Internal/protected**
  - Visible en el proyecto (ensamblado) actual y también visible en las clases derivadas.

Constructores

- En C# existen unos métodos específicos para controlar la creación (constructores) y la eliminación de objetos (destructores).
- En C# si no se definen se heredan por defecto de System.Object.
- Los constructores tienen el mismo nombre de la clase y no devuelven ningún valor.
- Pueden tomar o no argumentos (constructor por defecto).

```
class MaClasse {
    public MaClasse()
    {
    // Codi del constructor per defecte
    }
    public MaClasse(int maEnt)
    {
    // Constructor con parámetro maEnt
    }
}
```

- Puede haber múltiples constructores en una clase (si cuenta con diferentes argumentos), pero cada clase sólo puede contar como máximo con un destructor.
- El constructor de la clase se ejecuta en el momento que se utiliza new para crear un nuevo objeto de esa clase.

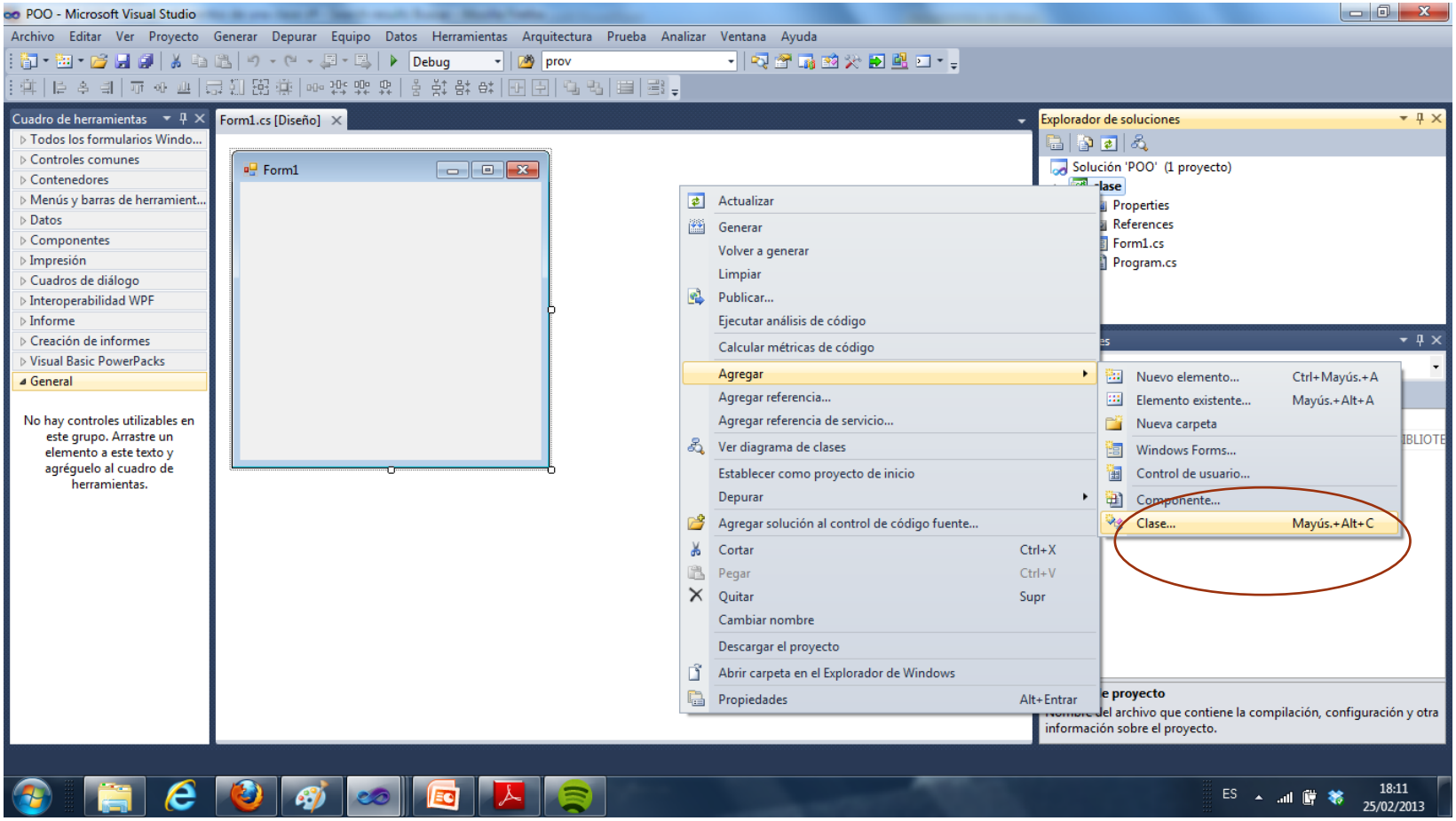
Destructor

- El **destructor** no retorna ningún valor y no tiene argumentos.

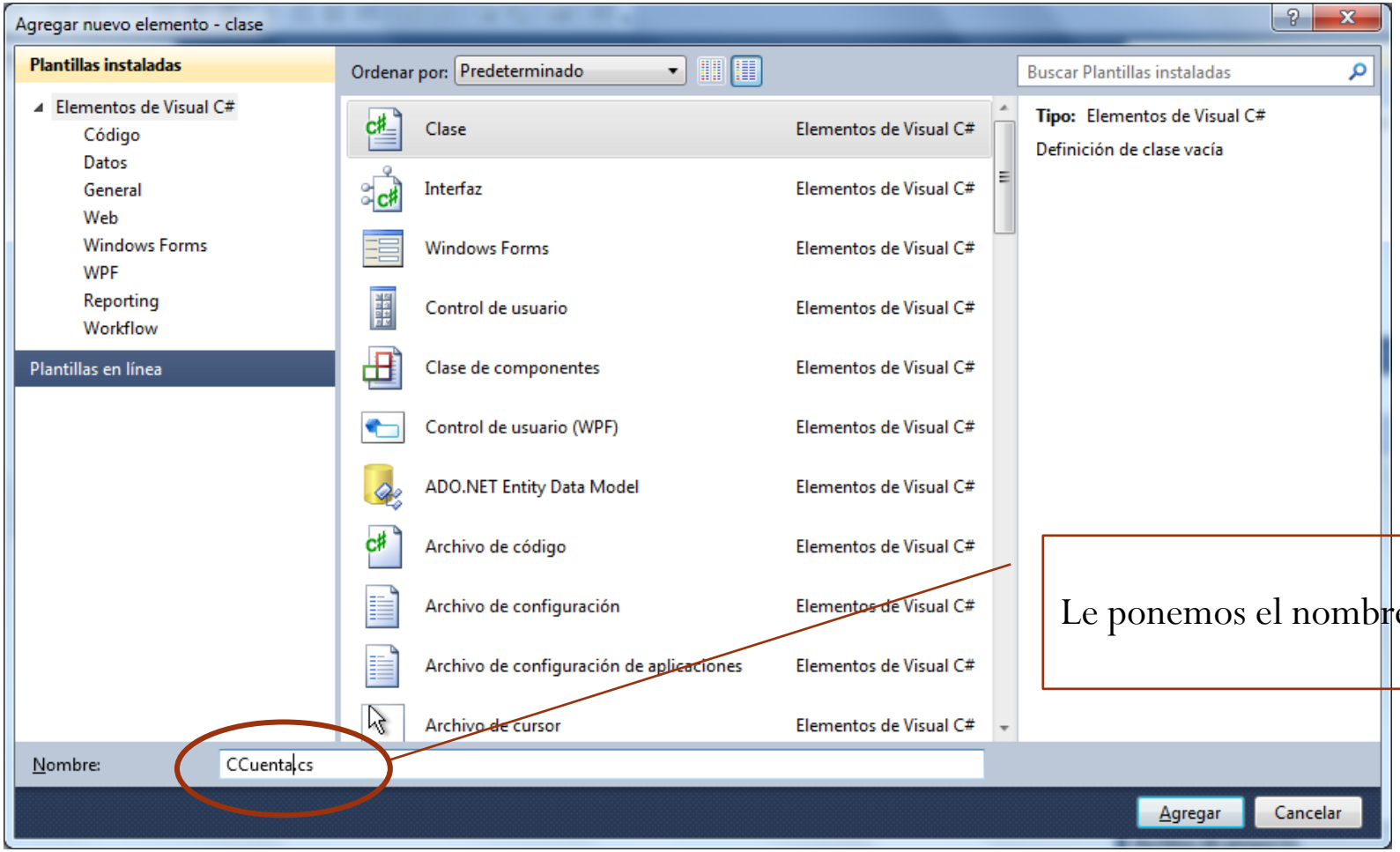
```
~MaClasse() {
// Códi del destructor.
}
```

- El destructor de la clase se ejecuta para cierto objeto cuando ya no hay ninguna referencia a ese objeto. De ello se ocupa el **recolector de basura (garbage** collector) de la plataforma .NET. (ya no existe el **delete en C#)**
- Para hacer referencia explícita dentro de la clase al objeto que llama al método usaremos **this.**

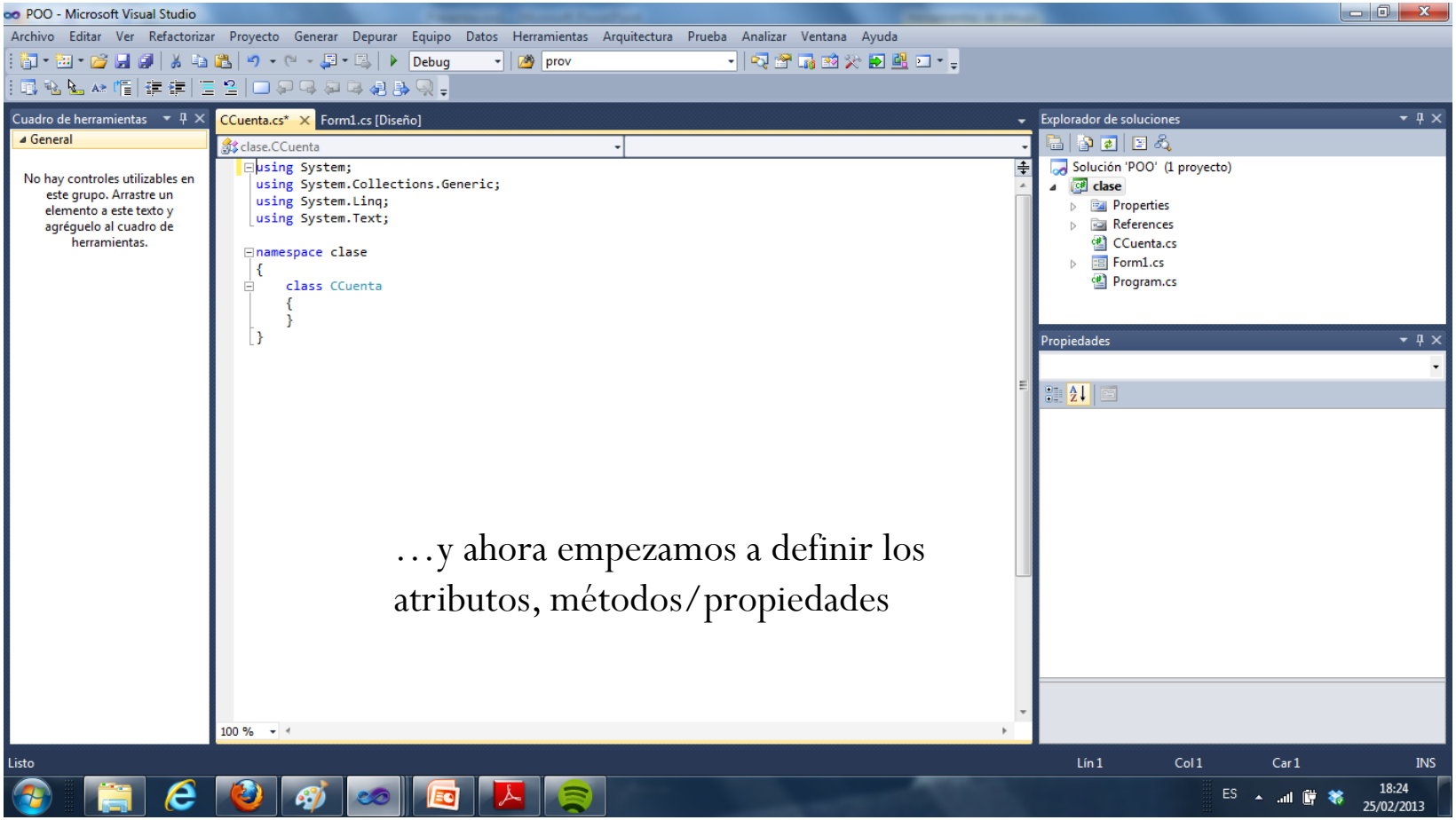
Crear un Clase en VS-C#



Crear un Clase en VS-C#



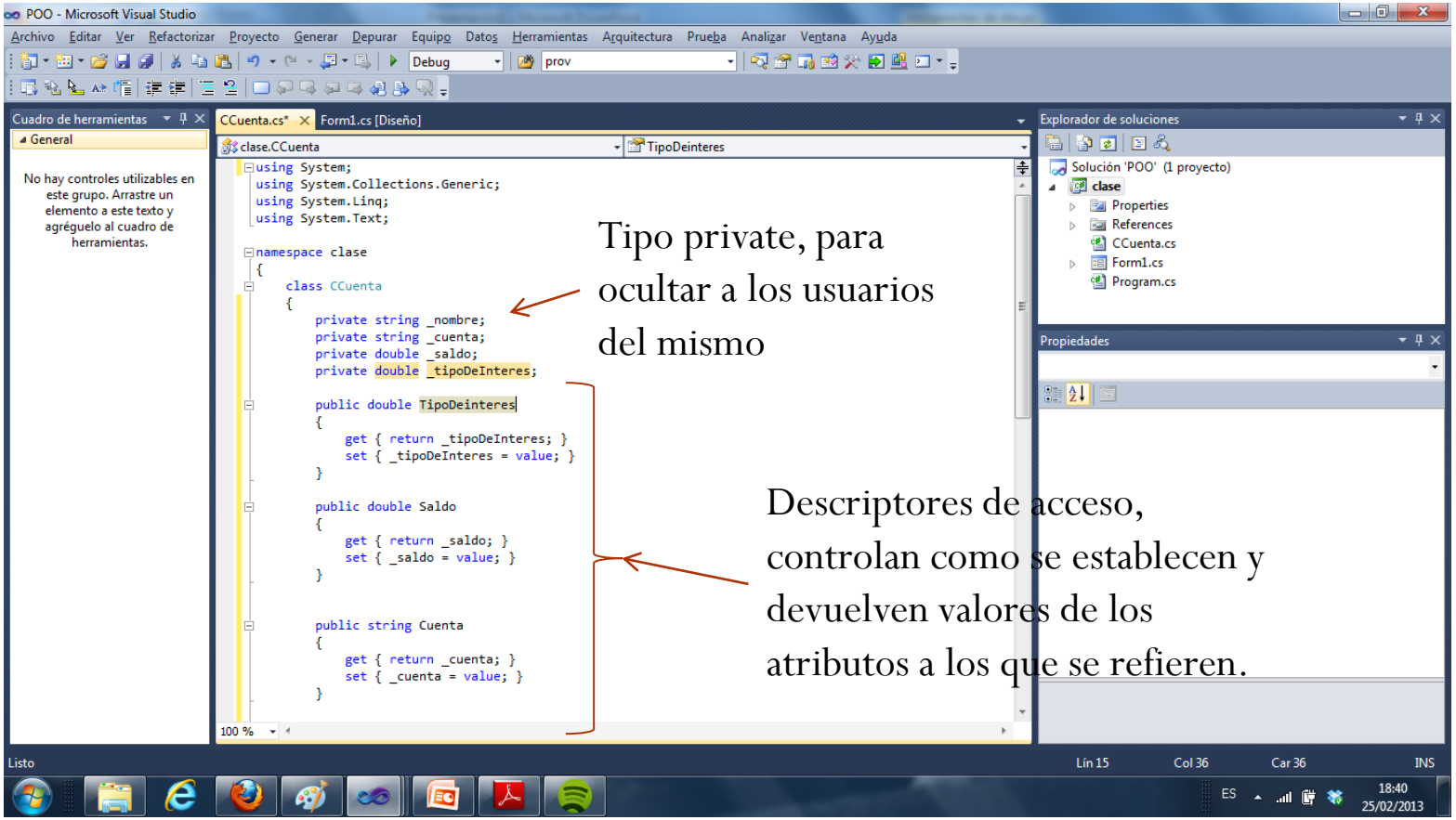
Crear un Clase en VS-C#



…y ahora empezamos a definir los atributos, métodos/propiedades

Crear un Clase en VS-C#

- **Atributos**: son las características individuales que diferencian un objeto de otro, pensemos en una clase Ccuenta.cs



Definir los Atributos y sus métodos get y set

- en Java

```java
public class Empleado {
    /* ATRIBUTOS */
    private String nombre;
    private double pagaHora;
    private double horas;

    /* 1º CONSTRUCTOR por defecto*/
    public Empleado() {
        nombre = "";
        pagaHora = 0;
        horas = 0;
    }
    /* 2º CONSTRUCTOR */
    public Empleado(String nombre, double pagaHora, double horas)
        this.nombre = nombre;
        this.pagaHora = pagaHora;
        this.horas = horas;
    }
    /* MÉTODOS GET Y SET */
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public double getPagaHora() {
        return pagaHora;
    }
    public void setPagaHora(double pagaHora) {
        this.pagaHora = pagaHora;
    }
    public double getHoras() {
        return horas;
    }
    public void setHoras(double horas) {
        this.horas = horas;
    }
```

en C#

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace clase
{
    class Empleado
    {
        /* ATRIBUTOS */
        private String nombre;
        private double pagaHora;
        private double horas;

        /* 1º CONSTRUCTOR por defecto*/
        public Empleado()
        {
            nombre = "";
            pagaHora = 0;
            horas = 0;
        }
        /* 2º CONSTRUCTOR */
        public Empleado(String nombre, double pagaHora, double horas)
        {
            this.nombre = nombre;
            this.pagaHora = pagaHora;
            this.horas = horas;
        }
    }
}
```

Hasta aquí todo igual…

Definir los Atributos y sus métodos get y set

- Donde hay cambios es en la hora de definir los gets y sets:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace clase
{
    class Empleado
    {

        /*get y set*/
        public String _nombre { get; set; }
        public double _pagaHora { get; set; }
        public double _horas { get; set; }

        /* 1º CONSTRUCTOR por defecto*/
        public Empleado()
        {
            _nombre = "";
            _pagaHora = 0;
            _horas = 0;
        }
        /* 2º CONSTRUCTOR */
        public Empleado(String nombre, double pagaHora, double horas)
        {
            this._nombre = nombre;
            this._pagaHora = pagaHora;
            this._horas = horas;
        }


    }
}
```

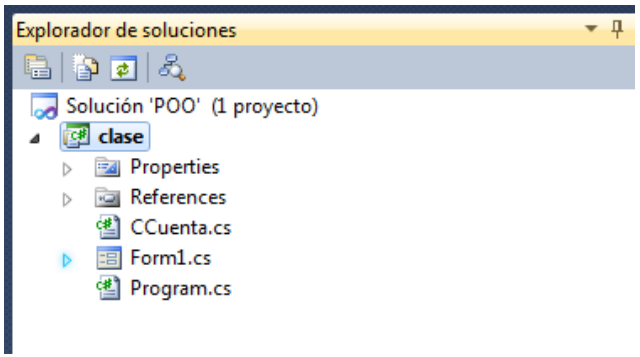
Aquí definimos los atribuots y los métodos get y set a la vez.
Si utilizáis el comando prop+2tabs lo genera automático, solo hay que cambiar el nombre dell atributo.

El nombre de los atributos/métodos debéis utilizar el estilo **caMel(la** primera letra de cada`palabra en mayúscula exceptuando la primera,ejemplo, contadorPrueba).
Aunque no quita que podéis ver ejemplo que eso no se cumple.

Hay otras maneras de crear las clases en c# pero podemos entrar en conflicto con las enseñanazas en Java.

Trabajar con Objetos en VS-C#

- Ya sabemos como se crea una clase ahora nos toca trabajar con ella… para ello tenemos la situación de la imagen un Form y la clase Ccuenta.cs
- En el método load del formulario vamos a generar el siguiente código




```
class CCuenta
{
   /* private string _nombre;
    private string _cuenta;
    private double _saldo;
    private double _tipoDeInteres;*/

    public string _nombre { get; set; }
    public double _saldo { get; set; }
    public double _tipoDeInteres { get; set; }
    public string _cuenta { get; set; }

    public CCuenta() { }

    public CCuenta(string nom, string cue, double sal, double tipo)
    {
        this._nombre=nom;
        this._cuenta=cue;
        this._saldo=sal;
        this._tipoDeInteres=tipo;
    }
```

```
private void Form1_Load(object sender, EventArgs e)
{
    CCuenta cuenta01 = new CCuenta();

    cuenta01._nombre = "Pepe";
    cuenta01._cuenta = "1234-44-5-00000000000000079798";
    cuenta01._tipoDeInteres = 2.5;
    cuenta01._saldo = 234.89;

    label1.Text = cuenta01._nombre;
    label2.Text = cuenta01._cuenta;
    label3.Text = cuenta01._saldo.ToString();
    label4.Text = cuenta01._tipoDeInteres.ToString();
```

Si os fijáis tanto para la lectura/Escritura si invoca al mismo método

…y hasta aquí lo convencional

Trabajar con Objetos en VS-C#

Repetir el ejemplo pero modificando la clase según el ejemplo

```
class CCuenta
{
   /* private string _nombre;
    private string _cuenta;
    private double _saldo;
    private double _tipoDeInteres;*/

    public string _nombre { get; set; }
    public double _saldo { get; set; }
    public double _tipoDeInteres { get; set; }
    public string _cuenta { get; set; }

    //public CCuenta() { }

    //public CCuenta(string nom, string cue, double sal, double tipo)
    //{
    //    this._nombre=nom;
    //    this._cuenta=cue;
    //    this._saldo=sal;
    //    this._tipoDeInteres=tipo;
    //}
```

- Y observar:
  - A pesar de no haber definido un constructor por Defecto en la clase me permite invocar la clase.
  - Pero en el momento que creemos un constructor con Parámetros nos obliga a definir éste.

Trabajar con Objetos en VS-C#

- De igual manera que habéis visto en Java tb aquí crearemos constructor con parámetros, pero además aquí hay una variante que según los "gurus" de la programación dicen por eso es mejor C# que Java, fijaros en la siguiente imagen

```
private void Form1_Load(object sender, EventArgs e)
{
    CCuenta cuenta01 = new CCuenta();

    cuenta01.Nombre = "Pepe";
    cuenta01.Cuenta = "1234-44-5-00000000000000079798";
    cuenta01.TipoDeinteres = 2.5;
    cuenta01.Saldo = 234.89;
    listBox1.Items.Add("---------Constructor por Defecto----------");
    listBox1.Items.Add(cuenta01.Nombre);
    listBox1.Items.Add(cuenta01.Cuenta);
    listBox1.Items.Add(cuenta01.Saldo);
    listBox1.Items.Add(cuenta01.TipoDeinteres);

    CCuenta cuenta02 = new CCuenta("Luis", "1003-55-5-00000000000869699", 6000, 3.5);
    listBox1.Items.Add("---------Constructor con Parámetros----------");
    listBox1.Items.Add(cuenta02.Nombre);
    listBox1.Items.Add(cuenta02.Cuenta);
    listBox1.Items.Add(cuenta02.Saldo);
    listBox1.Items.Add(cuenta02.TipoDeinteres);

    CCuenta cuenta03 = new CCuenta { Nombre = "Luis", Cuenta = "1003-55-5-00000000000869699" };
    listBox1.Items.Add("---------Constructor Personalizado----------");
    listBox1.Items.Add(cuenta03.Nombre);
    listBox1.Items.Add(cuenta03.Cuenta);

}
```

Este ya esta visto… sin novedad

Este nos creamos el constructor en la clase y le invocamos pasandole los valores… tb está claro,no?

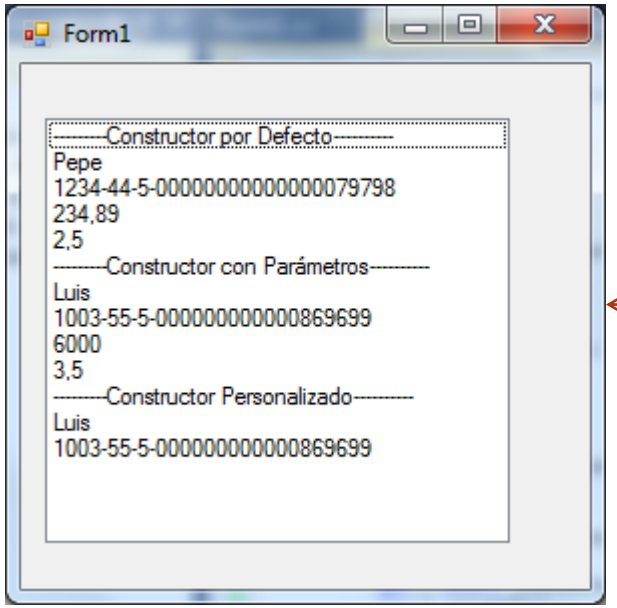
Este ya si que es diferente está compuesto por nombre de la variable(set) y valor, además encerrado entre llaves y no está definido en la clase, así nos podremos crear constructores a la carta

Veamos como está definida la clase….

Trabajar con Objetos en VS-C#

```
class CCuenta
{
    private string _nombre;
    private string _cuenta;
    private double _saldo;
    private double _tipoDeInteres;

    public CCuenta() { }

    public CCuenta(string nom, string cue, double sal, double tipo)
    {
        this.Nombre=nom;
        this.Cuenta=cue;
        this.Saldo=sal;
        this.TipoDeinteres=tipo;
    }

    public double TipoDeinteres
    {
        get { return _tipoDeInteres; }
        set { _tipoDeInteres = value; }
    }

    public double Saldo
    {
        get { return _saldo; }
        set { _saldo = value; }
    }


    public string Cuenta
    {
        get { return _cuenta; }
        set { _cuenta = value; }
    }


    public string Nombre
    {
        get { return _nombre; }
        set { _nombre = value; }
    }
```

Los 2 constructores, por defecto y por parámetros… y ni rastro del personalizado, si se ejecuta tendremos:



Observar que he sacado los datos en un listbox
lo mismo podría haberlo hecho en cualquier componente…

Trabajar con Array de Objetos

```
private void Form1_Load(object sender, EventArgs e)
{
    CCuenta cuenta01 = new CCuenta();

    cuenta01.Nombre = "Pepe";
    cuenta01.Cuenta = "1234-44-5-00000000000000079798";
    cuenta01.TipoDeinteres = 2.5;
    cuenta01.Saldo = 234.89;
    CCuenta cuenta02 = new CCuenta("Luis", "1003-55-5-00000000000869699", 6000, 3.5);
    CCuenta cuenta03 = new CCuenta { Nombre = "Luis", Cuenta = "1003-55-5-00000000000869699" };

   // ArrayList clientes = new ArrayList();
    List<CCuenta> clientes = new List<CCuenta>();

    clientes.Add(cuenta01);
    clientes.Add(cuenta02);
    clientes.Add(cuenta03);
    listBox1.Items.Add("--------------foreach------------------");
    foreach (CCuenta prime in clientes) // Loop through List with foreach
    {
        listBox1.Items.Add(prime.Nombre);
        listBox1.Items.Add(prime.Cuenta);
        listBox1.Items.Add(prime.Saldo);
        listBox1.Items.Add(prime.TipoDeinteres);
        listBox1.Items.Add("-----------------------------------------");

    }
    listBox1.Items.Add("--------------for--------------------");
    for (int i = 0; i < clientes.Count; i++) // Loop through List with for
    {
        listBox1.Items.Add(clientes[i].Nombre);
        listBox1.Items.Add(clientes[i].Cuenta);
        listBox1.Items.Add(clientes[i].Saldo);
        listBox1.Items.Add(clientes[i].TipoDeinteres);
        listBox1.Items.Add("-----------------------------------------");
    }

}
```

Para trabajar de una manera más cómoda con los objetos , podemos utilizar estructuras como los ArrayList o las List la diferencia está que a las listas le tienes que decir que tipos de datos va almacenar.

A la hora de Mostrar los datos solo tendremos que recorres la estructura con un foreach o un for…