C#

Programación101.net

Ese muchacho usa la extensión ReSharper

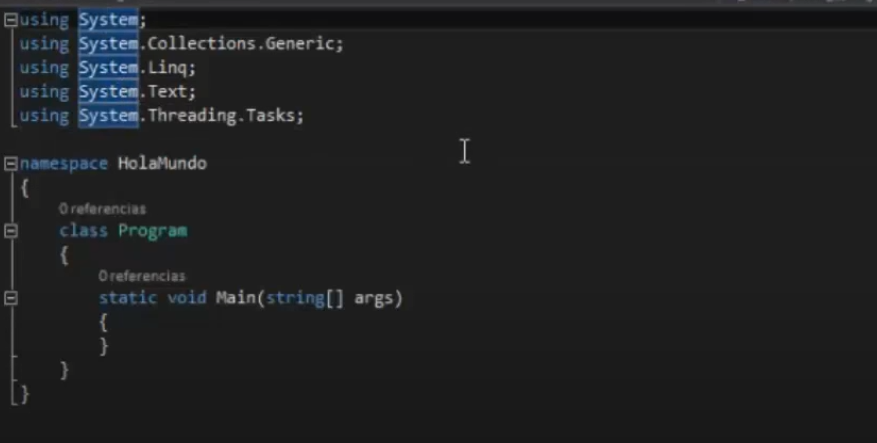
https://app.edutin.com/classroom?config=%7B%22certification\_id%22:0,%22clase\_id%22:%22409363117%22,%22curso\_id%22:%224294%22,%22idu%22:0,%22language\_id%22:%221%22,%22quiz\_id%22:0,%22source\_id%22:0,%22state%22:-1,%22status%22:false,%22tour%22:-1,%22type%22:null%7D

SQL

Post22Futi

Puerto= 5432

Dice que es un lenguaje orientado a objetos (o clases, serían sinónimos. No, no lo son).



Al abrir Visual Studio, elige crear un archivo “aplicación de consola” y así le aparece. A mi no.

El objeto/clase encapsula el código.

System sería una clase, y todo lo que está luego del . son métodos.

A la vez, HolaMundo es una clase creada por él, que ya viene con una clase.

El primer método siempre es el main.

Pone console.WriteLine (“Hola mundo”);

Al dar iniciar se abre una pantallita negra con el msj y se cierra al instante.

Console.ReadKey (); (o ReadLine, espera hasta que se toque una letra para finalizar). Estos 2 establecen una “pausa”.

Se ve que las clases, excepto system están todas con el mismo color.

Si se borra using system de arriba las cosas dejan de funcionar. Se podría arreglar si antes de cada método se lo pone. Ej: system. console.WriteLine (“Hola mundo”);

Comentarios: // o /\* \*/

Variables y tipos de datos

Tipos de datos se utilizan para declarar funciones y variables.

Variable de tipo byte, va de 0 a 255 byte edad= 34;

Entero es int, se pueden utilizar números muy grandes. Ralentizaría el procesamiento usar entero para variables mucho mas pequeñas.

Double, son valores flotantes, creo que con 2 decimales, esa es su gracia.

Flotante es float

Boolean. Solo 2 valores, true o false.

Caracteres es char. Solo uno puede ponerse.

String o cadena de texto.

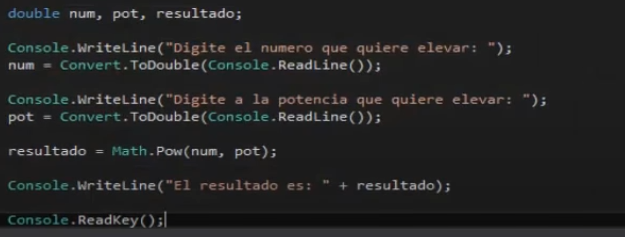
Cuando no se sabe el tipo de dato que se va a ingresar, es el Dynamic. El programa lo intenta adivinar.

Dynamic variable=23;

Operadores

+ - \* / También está el módulo %

Para usar potencias y raíces hay que usar la clase math



Primero declara las 3 variables como de tipo double.

Después pone las líneas de console para que las vea el usuario.

Después pone convert.ToDouble (ya que lo que se trata adentro de un console.ReadLine se convierte automáticamente a string, o sea, ya no era double.

Math.Pow (número, potencia) Ahí se hace la potencia.

Matg.sqrt es para buscar raíces cuadradas.



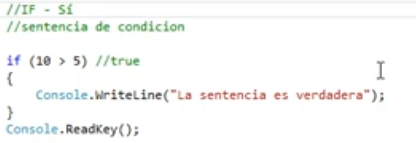
E+=1; ES IGUAL A E= E+1;

E\*=7; ES IGUAL A E= E\*7;

E++; ES IGUAL A E+1;

Sentencia If/Else

Sentencias de control o también llamadas “ciclos”, la cual nos permite realizar instrucciones dependiendo del cumplimiento de las condiciones que nosotros establezcamos previamente.



Switch

Es una estructura de control como el if. Gastaría mas recursos que la otra, por lo que se recomienda, de poderse usar el if.

Int valor=0;

Los case van con números.

Afuera del switch pone el console.ReadKey(); para que no se cierre la consola.

int valor = 0;

switch (valor)

{

case 1: //ahí adentro van las instrucciones. Luego la palabra break.

Console.WriteLine("Entro al caso 1");

break;

case 2:

Console.WriteLine("Entro al caso 2");

break;

case 3:

Console.WriteLine("Entro al caso 3");

break;

default:

Console.WriteLine("Entro al caso Default");

break;

}

Console.ReadKey();

Haciendo todo eso va a entrar al Default ya que el valor es 0 y no hay un case:0;

case 0 & 1: //ahí adentro van las instrucciones. Luego la palabra break.

Console.WriteLine("Entro al caso 1");

Eso si me lo tomó y entro a ese case.

También el la variable que determina a qué caso se entra podría ser un string y los case también lo serían.

While (Es un Ciclo)

While, do-while, for, foreach.

Repite el código la cantidad de veces que le digamos.

While (condición) {

Instrucciones

}

Incrementos y decrementos en variables:

Int num=0;

Num++; (incremento)

Num--; (decremento)

Console.ReadKey();

Do While

int numero = 0;

do

{

} while (condicion) //Que sería true o false.

El While primero evalúa la condición y después, si es true, ejecuta las instrucciones. El do-while, primero ejecuta y luego evalúa. O sea, que este ejecuta al menos una vez, por mas que al final, el while sea false.

For

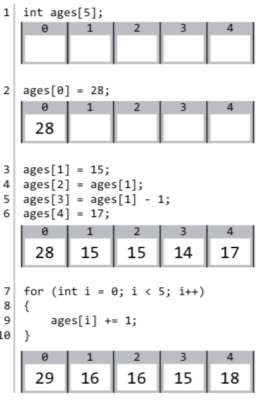
for (int i = 0; i < length; i++) //i<length es la condición.

{

}

Arrays(arreglos. A veces le dicen matrices)

Es una estructura de datos: la cual almacena múltiples valores para el mismo tipo de datos



Cada cajita tiene un valor y un índice.

La gran limitación del array es que si lo inicializas con 6 elementos, después no podes agregar un séptimo.  
Se podría crear un array mas grande y pasarle los valores que se pusieron en el array mas pequeño.

For each y arrayas multidimensionales

Foreach (tipo nombre in array). Es un bucle iterativo.

Foreach significa “para cada elemento de la lista”.

string[] listacompra = new string[4];

listacompra[0] = "leche";

listacompra[1] = "pan";

listacompra[2] = "cacao";

listacompra[3] = "maní";

foreach(string elemento in listacompra)

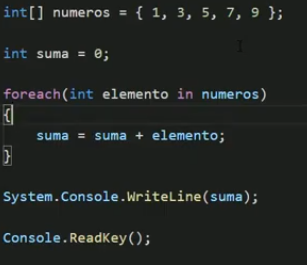
{

Console.WriteLine(elemento + ",");

}

Console.ReadKey();

La coma dentro de las comillas es lo que se pone cuando termina cada string. Se podría poner un punto, p ej.



Multidimensional es matrices.

Afirma que últimamente los arrays no están mucho en uso por el problema mencionado. Se reemplazaron por las listas.

Listas



List<T> es el tipo de datos Lista es el nombre. Lo otro la inicializa

En la T va el tipo de datos. Int, por ejemplo.

No hay que poner el número de elementos

Con el método Add agregamos elementos.

Lista.add(15); //dentro de () no va el índice.

Lista.add(4);

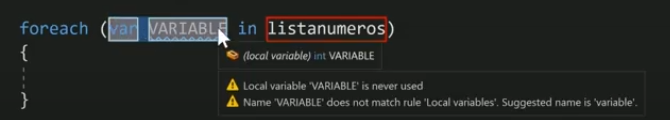
Lista.add(“15”); //Si fuera un string

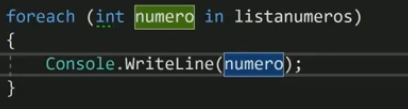
Imprimir uno

Console.writeline (lista[0]); //Ese si es el índice: sería el 15.

Console.writeline (lista.count); //eso va a devolver el número de elementos que tenemos.

Eliminar elementos:





Con solo ese foreach va a imprimir todos los números de la lista.

Lista.Remove (99); //Aquí no se indica la posición, sino el número a eliminar. Si hay 2 elementos iguales, solo borra el primero que aparece.

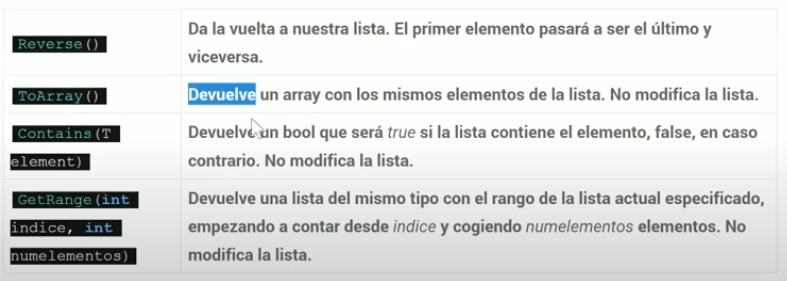
Lista.RemoveAt (1), // Aquí si elimina la posición o índice 1.

Lista.Clear(); //Borra todo.

Insertar elementos en una posición concreta:

Lista.insert (2,25); //Son 2 parámetros: (posición,valor).

Lista.reverse(); //Así la lista se da vuelta



If (lista.contains(30) { } //para eso se usa mucho.

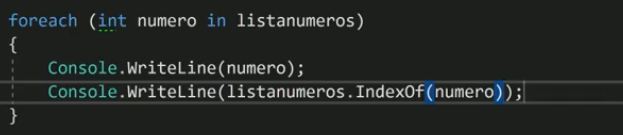
Recorrerla: imprime los elementos.

Es mas largo que el foreach. La ventaja es que tenemos acceso al número del elemento.

For (contador=0 ; contador<lista.count; contador++;) {

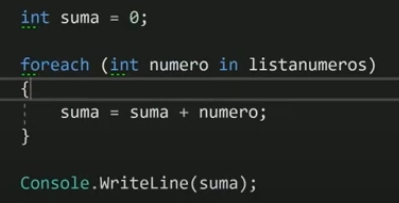
Console.WriteLine (lista[contador]);

}



Con el .IndexOf(numero) podemos averiguar el índice de los números.

Por mas que se pueda, el foreach no se usa cuando hay que saber la posición de los números. Si se lo puede usar, para sumar sus valores:



Repite que no se puede modificar una lista (añadir o modificar elementos) con un foreach.

Diccionario

Diccionarios, no hay índices, sino claves, del tipo de datos que queramos

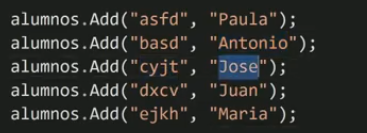


Dentro de <> hay 2 tipos de datos: El primero refiere a la clave y el segundo al valor.



Podría ser así.

Se agrega elementos de forma similar a una lista, con add. Pero siempre hay que agregar clave y valor. Podrían ser enteros también.

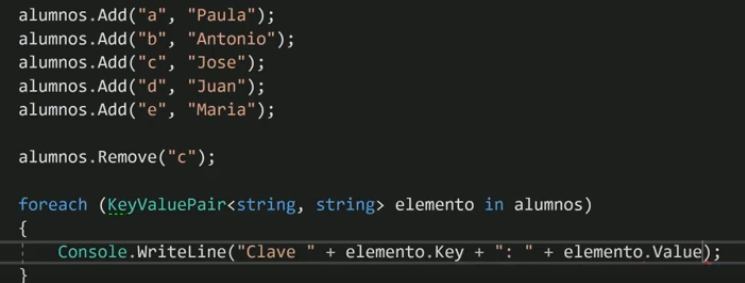


Así se imprime un elemento.

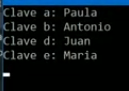


Con .remove se borra un elemento, indicando la clave.

Recorrer con foreach. Se usa KeyValuePair



Ahí se crea la variable elemento, que es a la cual después se le aplican los métodos para printear.



Dice que en vez de poner KeyValuePair<string,string> podemos poner solamente var. Lo cual es la inferencia de la variable que estamos utilizando. No recomiendo tanto usarlo, al menos durante el aprendizaje.

Programación orientada a objetos

La programación orientada a objetos (POO) se define como un paradigma de la programación, donde el código está organizado en unidades y se le llama clases, y a partir de estas clases, se crean objetos que se relacionan para conseguir el resultado deseado. Este paradigma es una forma especial de programar que trata de asemejar las cosas que expresamos en la vida real. Para entender la POO es necesario tener dos conceptos claros: clases y objetos.

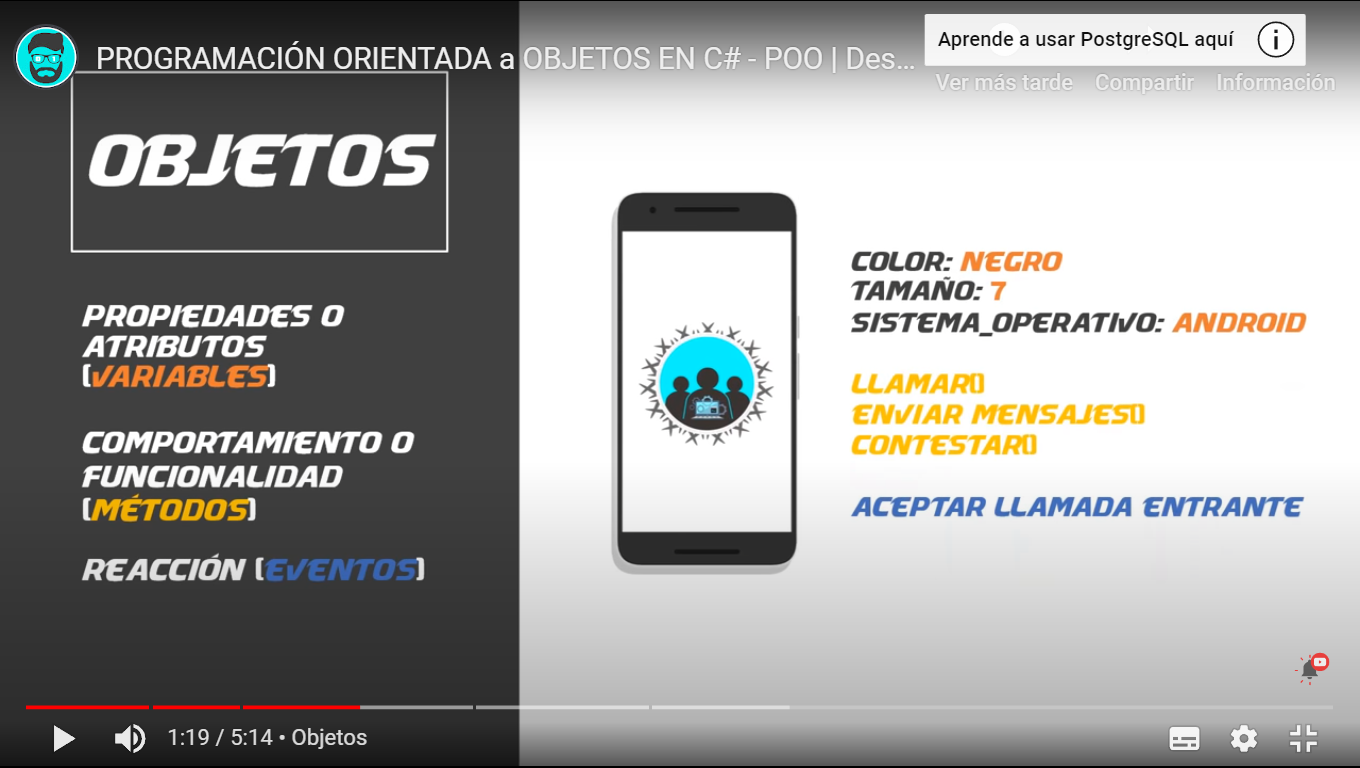
**Clases**

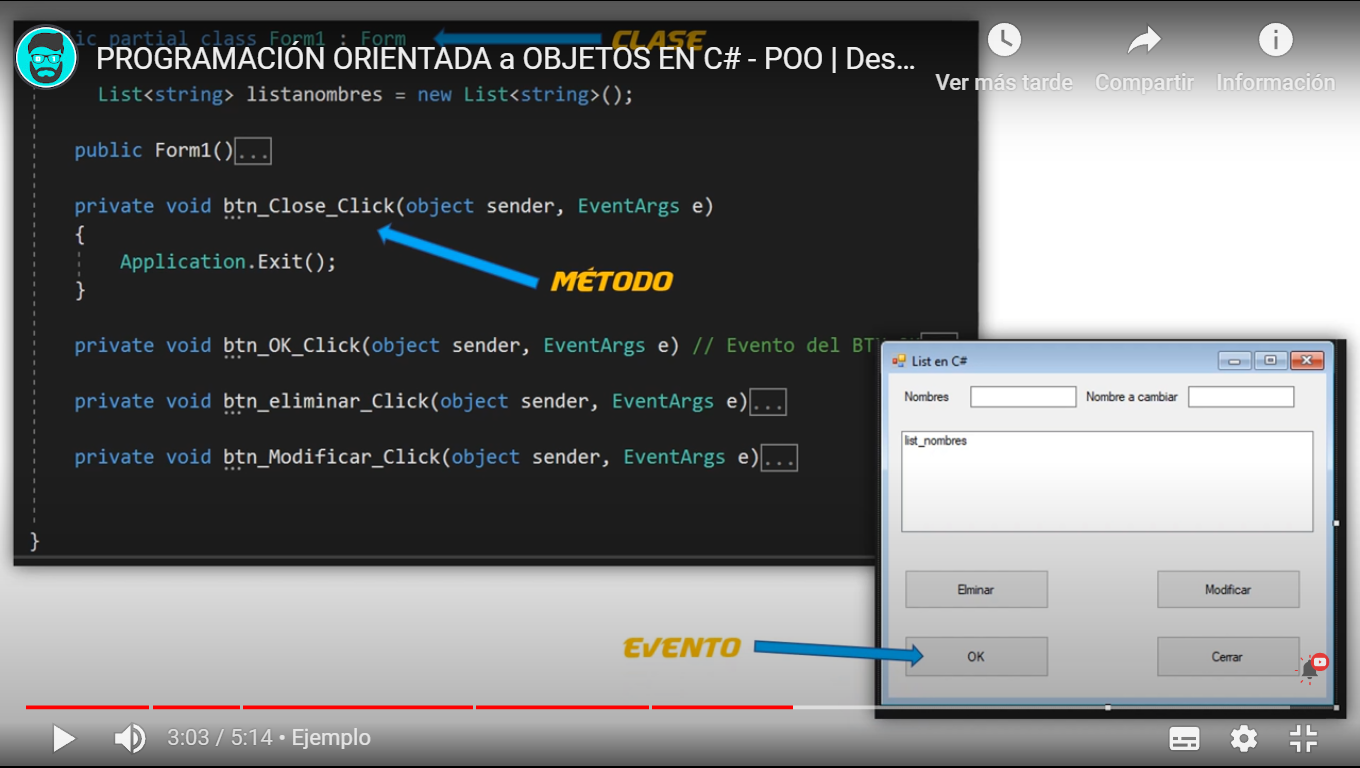
Define y estructura el comportamiento de un objeto. Diría yo que define y estructura los comportamientos posibles de un objeto.

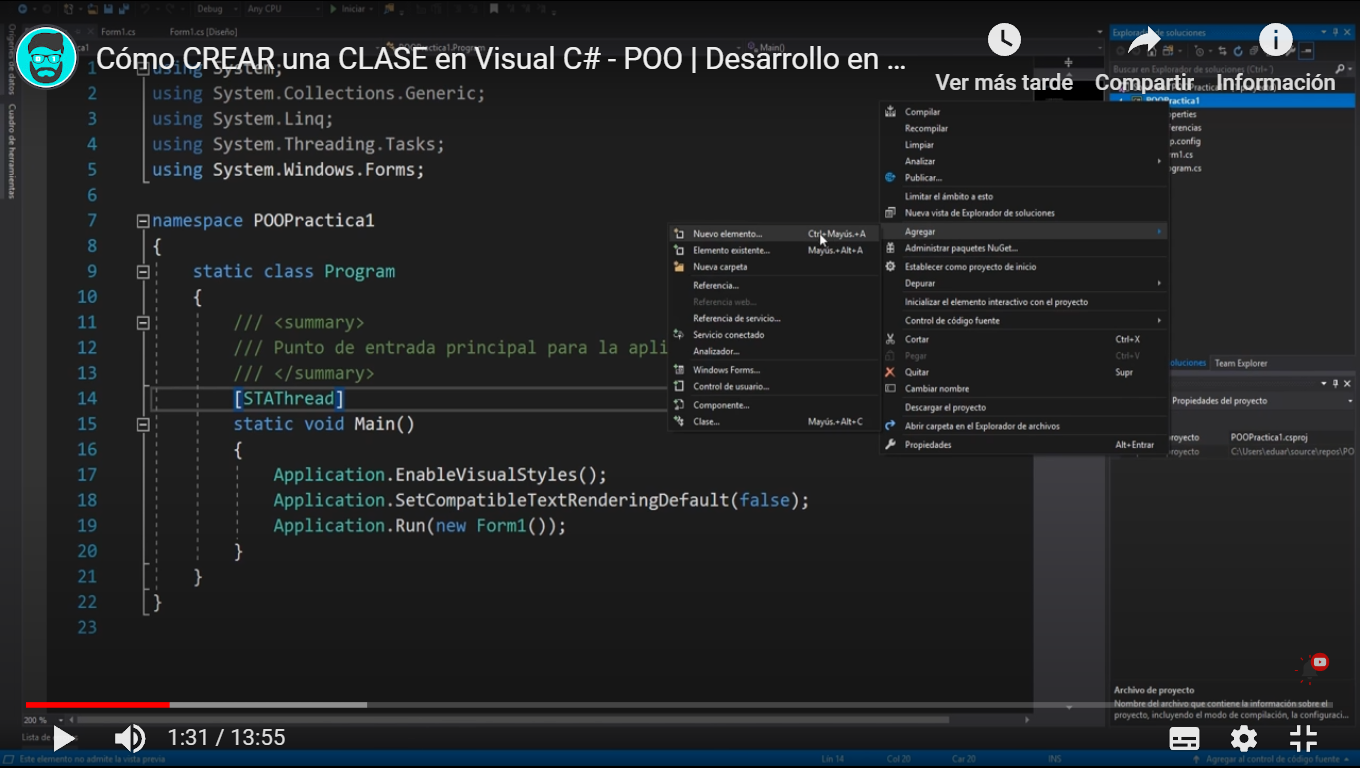
Describe un objeto y consta de una serie de métodos y datos que resumen las características del objeto. Las clases son reutilizables siempre y cuando no se hayan definido como estáticas, ya que se pueden instanciar, pero si es estática solo se podrá acceder a ella a través de la propia clase.

**Objetos**

Es una entidad que tiene un determinado estado, comportamiento e identidad, los objetos representan o son cosas. Cada objeto es una instancia de una clase.





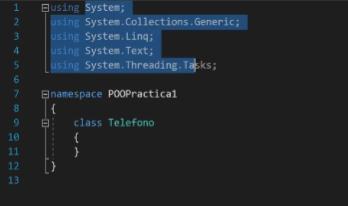


Se pone crear nuevo proyecto => Aplicación de Windows forms (.Net Framework) (para C#, obvio).

Eso te abre un form, el cual al hacerle click derecho te permite chusmearle el código.

Quizás haya que ir a Ver => Darle a explorador de soluciones. En su segundo renglón está el nombre del proyecto. Tocarlo. Tocar agregar, nuevo elemento: Nueva clase.

Se crean 2 páginas. La que primero se ejecuta es form1, la principal. Después está aquella a la que le dimos nombre. Sobre esta pongo el siguiente screenshot y es sobre la que primero trabaja.



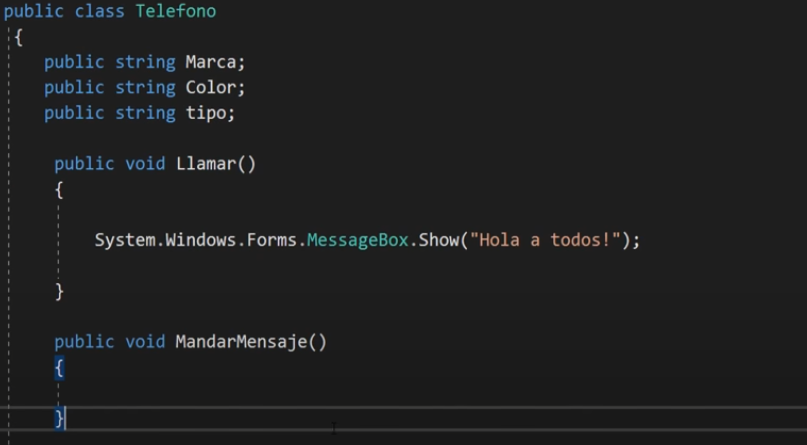
Los using son llamados a las librerías internas de C#.

El namespace es el espacio donde podemos encapsular las clases y el código.

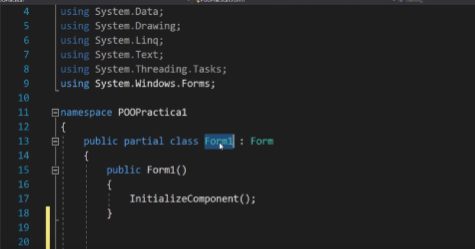
Luego se ve la clase, con sus llaves.

La clase tiene sus atributos y variables.

Pone public para que las variables se puedan utilizar fuera de la clase en que se crea.



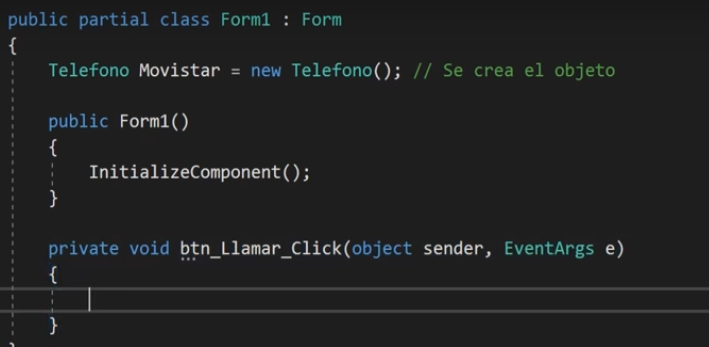
Eso es lo que hace. Agrega esos 2 métodos.

Luego va a form1:  


Public form1 es el método constructor o principal de nuestra clase. Constructor es el método que tiene el mismo nombre que la clase. No es tan importante como el main, pero es lo primero que se ejecuta dentro de esa clase.

Hace el llamado a teléfono.  
Teléfono Movistar= new Teléfono //Ahí crea el primer objeto.

Va a form1 Diseño, cuadro de herramientas y agrega un button. Al botón le pone un nombre y a la vez le pone aquello que tiene que verse. Cuando le da click, se crea automáticamente:



Dentro de las llaves pone Movistar y ahí le aparecen las opciones de método que ya había creado. Pone Movistar.Llamar ();

Como crear una clase

Program.cs es la clase principal. Ahí es donde está el main.

En cada una de las librerías de los using hay clases que c# puede usar.

Variables= Atributos o propiedades.

Una clase es como una plantilla para crear a partir de ella uno o varios objetos.

Nuevo elemento =>Crear clase.

Pone algunos string, y hace todo publico, incluso la clase.

Crea un método (llamar) con un mbox (Message box).

namespace Practica\_1

{

public class telefono

{

public string marca = "";

public string color = "";

public string tipo = "";

public void llamar ()

{

System.Windows.Forms.MessageBox.Show("¡Hola a todos!");

}

public void mandarMensaje ()

{

}

Luego va a form1:

Ahí crea

public partial class Form1 : Form

{

telefono Movistar = new telefono();

Y luego va al design y crea el botón.

Texto: Llamar y Name: btn\_Llamar

Luego. Dentro de el método pone esto:

private void btn\_Llamar\_Click(object sender, EventArgs e)

{

Movistar.llamar();

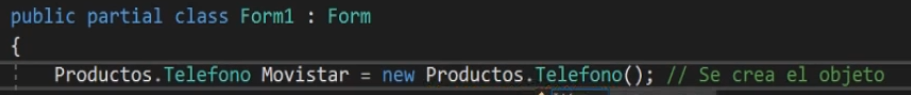
}

Eduardo de Tecnología Binaria, dio la clase.

Namespace: Ahí hay un conjunto de objetos relacionados, como las clases. Se pueden poner varios namespace. Uno destinado a productos, otro a la base de datos, etc.

Entonces, le cambia el namespace de la clase teléfono y luego va a form1 ahí hay muchas cosas que ya no funcionan.

1\_



Esa es una forma de arreglarlo, anteponiendo el llamado (Productos).

Namespace.Clase.Objeto sería:

Productos.Telefono.Movistar.

2\_También puede dejarse todo como está y poner en la parte de los using:

Using Productos;

Variables públicas y privadas

Si uno define un objeto como privado, no se puede modificar después el valor que tiene.

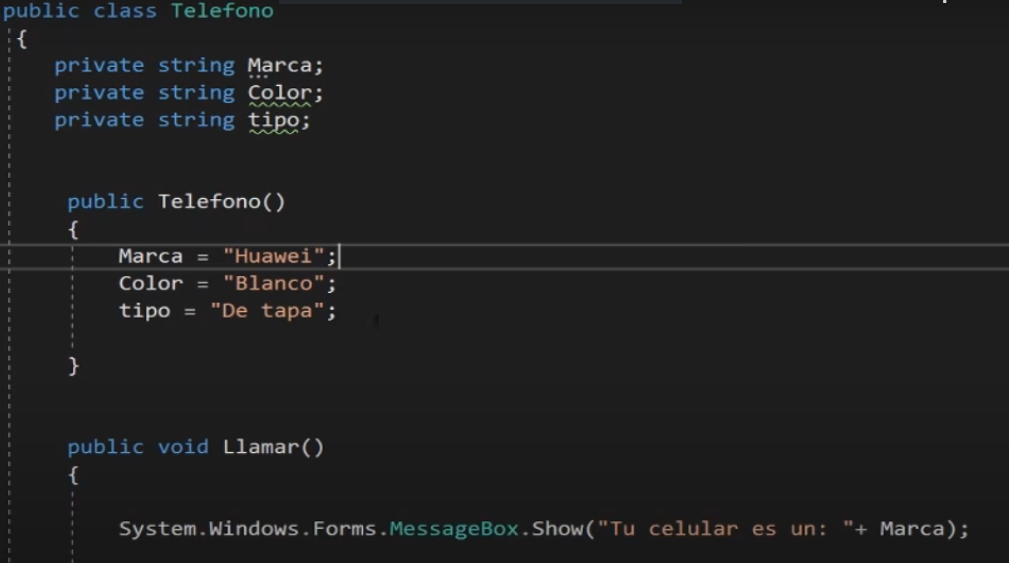
Usualmente, las variables dentro de una clase son privadas. Así, no se las puede llamar dentro de un método de otra clase. Solo dentro de la misma clase.

Método constructor

Tocando ctor se crea ese método. Se tiene que llamar igual que la clase. Es el primer método que se ejecuta, se hace automáticamente, no hay que llamarlo. Es el método por default.

Sirve para inicializar los atributos. Ahí manda los 3 strings que había hecho.

Este método no puede retornar datos.



Reescribe así. Y al iniciar, se printea eso (Huawei).

Constructores múltiples

Se tiene que llamar igual que la clase. Pero no pueden ser exactamente iguales los constructores. En el ejemplo, le pone como parámetro (string M) al segundo. Ya con eso no tira error.

public telefono(string M)

{

marca = M;//Aquí la Marca va a ser igual a lo que se le pase como parámetro.

color = "Blanco";

tipo = "Screentouch";

}

public partial class Form1 : Form

{

telefono Movistar = new telefono();

telefono Claro = new telefono("Samsung"); //Ahí pasa el parámetro a la M.

public Form1()

{

InitializeComponent();

}

private void btn\_Llamar\_Click(object sender, EventArgs e)

{

Movistar.llamar();

Claro.llamar();

}

}

Ahí, al tocar el botón ejecutaría las 2 líneas.

Métodos para retornar datos

Void. No retorna ningún valor.

En el constructor, el void está, pero implícito.

En vez del void pone el tipo de dato que va a retornar.

public string mandarMensaje (string N, string msj)

{

string Nombre = N;

string Mensaje = msj;

string enviandoMsj= "El mensaje: " + Mensaje + ", enviado por: " + Nombre;

return enviandoMsj;

}

Ahora usa este otro método, que es de tipo string.

Luego, en form1, desde el botón, donde se pasan los argumentos, hay que cambiar la forma de printear.

private void btn\_Llamar\_Click(object sender, EventArgs e)

{

MessageBox.Show(Movistar.mandarMensaje("Eduardo", "La puerca está en la pocilga"));

}

Cuando se usaba el método void llamar, se printeaba desde ahí.

Métodos y variables estáticas

Estático significa que no va a cambiar, tanto si es un método como si es una variable.

Agrega clase: Operaciones. Y crea ahí un par de variables.

Luego, desde form1 y su botón intenta llamar a la clase y no aparece nada. Luego redefine esas variables, anteponiendo static, y cuando llama ahora si aparecen.

class Operaciones

{

public static int N1 = 50;

public static int N2 = 100;

}

Así, queda el botón:

private void btn\_Llamar\_Click(object sender, EventArgs e)

{

MessageBox.Show(Operaciones.suma(55,66).ToString());

MessageBox.Show(Movistar.mandarMensaje("Eduardo", "La puerca está en la pocilga"));

MessageBox.Show("El crack de pablo pudo conseguir este número " +Operaciones.total.ToString());

}

Agrega el ToString() para que se pueda mostrar. Yo después agregué la tercera línea, para darle un uso a los ints creados en la clase operaciones. Muestro lo que allí hice:

class Operaciones

{

public static int NPrim = 50;

public static int NSec = 100;

public static int total= suma( NPrim, NSec);

public static int suma(int N1, int N2)

{

return N1 + N2;

}

}

Métodos Set y Get

Crea otra clase: Persona con 3 variables string.

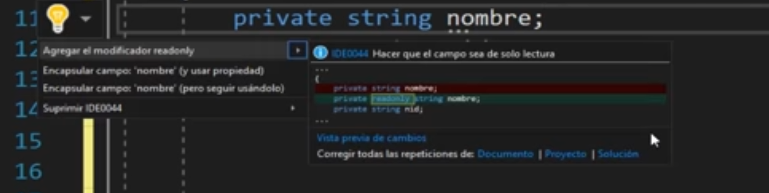
En el form1 crea el objeto = Lo inicializa

Persona P = new Persona();

Como son variables privadas no es tan simple poder acceder a ellas. Para eso se usa set y get.

Hay 3 formas:

Tocando ctrl+. En el mismo rengón donde se define la variable.

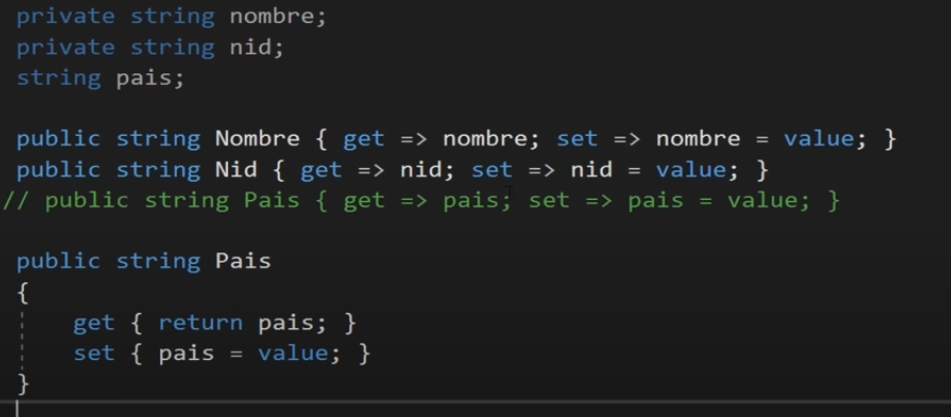


Y aparece eso. Toca encapsular campo (y usar propiedad). Tras eso, se genera:

public string Nombre1 { get => Nombre; set => Nombre = value; }

Eso es un método.

Ahora, si tocamos P, si nos va a aparecer la opción del nombre de esa variable.



Ahí comenta el tercer public string para poder hacerlo él, manualmente, en las últimas líneas del código.

Esta opción tiene la ventaja de que se pueden hacer modificaciones:

Esto va en el botón:

P.Pais = "Argentina"; //Este es el set, porque ahí se ingresa la info a la variable. Es lo que dice value..

La tercer manera es:

En la clase persona:

public string nid { get; set; }

En Botón:

P.nid = "4495";

MessageBox.Show(P.nid);

Creando archivos en C#

Aplicación de consola.

Dice que es necesario usar una librería:

Using.system.IO; //In & Out. Permite crear algunas variables.

TextWriter archivo;

Archivo= new StreamWriter(“archivo.txt”); //con esto lo que va a hacer es crear un archivo con cualquier extensión que se le indique.

String mensaje;

Mensaje= Console.Readline; // eso es para que se pasa el string por consola.

Archivo.WriteLine (mensaje); //con esto se graba el archivo en Archivo

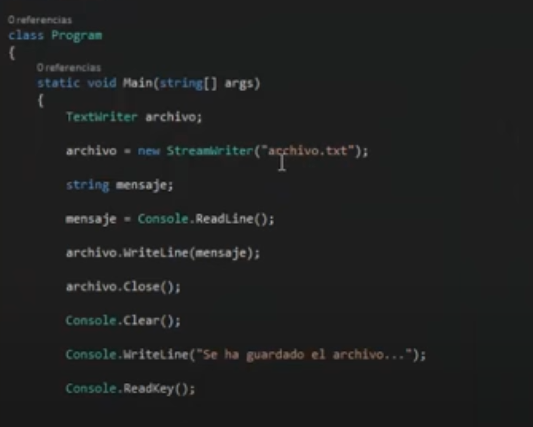
Archivo.Close(); //para que el archivo se cierre (se guarda).

Console.Clear ()//para “limpiar la pantalla”.

Console.WriteLine (“Se ha guardado el archivo”);

Console.ReadKey();

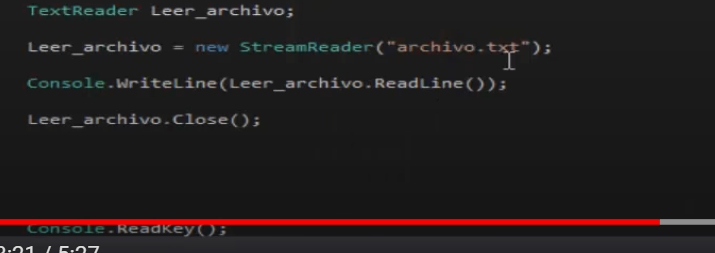
Archivos/bin/debug. Ahí los guarda por defecto.



Leyendo archivos

Se crea variable TextReader

Luego NewStreamReader.

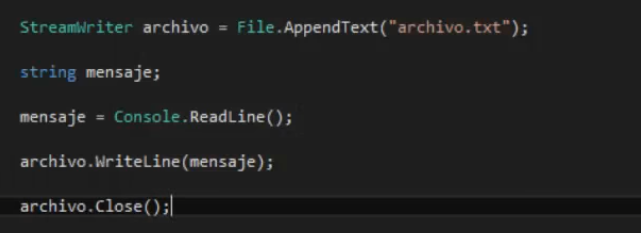
Luego un console.WriteLine para que imprima en consola lo que tiene el archivo.

Peeeero ReadLine, lee una sola línea. La primera.

Reemplaza el ReadLine por ReadToEnd

Agregar texto a un archivo ya creado

File.AppendText(); //Eso es para que no sobreescriba sino que sume el texto nuevo.



Hace solo eso. No muestra por consola lo que decía el archivo.