## Clase 2:

Clases propias: ejemplo -> “program”, o las que creamos nosotros, se crean cuando vamos a construir una app.

Clases predefinidas: Ya vienen con el propio lenguaje, son las “bibliotecas de clases” O “API”.

Ejemplo… using System adentro viene la clase “Console.”

Clase 3:

Identificadores: son nombres que se usan para identificar los elementos del programa.

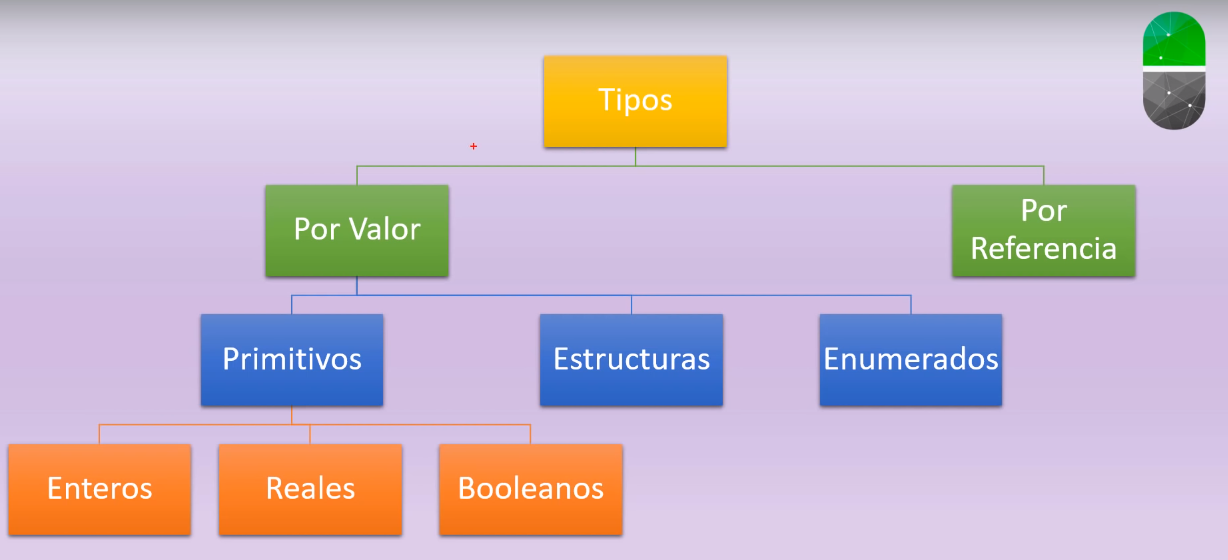
* Namespaces
* Clases
* Métodos
* Variables
* Constantes

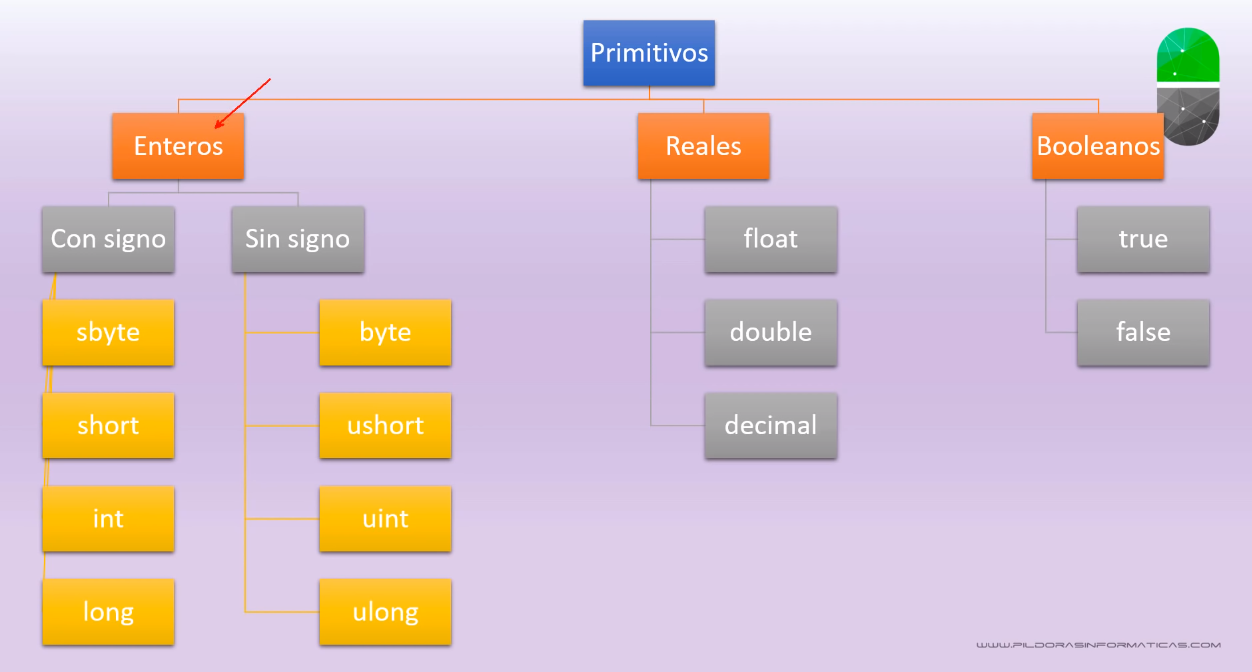
# Convenciones para nombrar identificadores:

* Solo se pueden usar letras (mayúsculas y minúsculas), números y guiones bajos.

Clase 4:

# Tipos de datos:





Sin signo:

* No hace referencia a nada, no es negativo ni positivo (maneja números entre -127 y 128). Ejemplo si uso “byte”, manejamos edades de personas podemos usar byte, ya que entraría en rango con este tipo de dato.

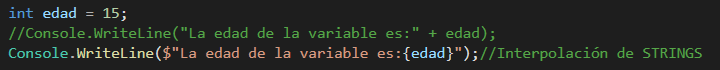
Reales:

* Decimal: se suele usar en C# a la hora de manejar transacciones bancarias como por ejemplo, no se necesita saber una cifra tan grande pero suele usarse ahí.



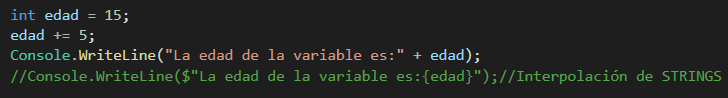
Clase 5:

# Interpolación de strings:



Incremento/decremento:

* Puede usarse adentro del método “WriteLine” de la clase “Console” o fuera.

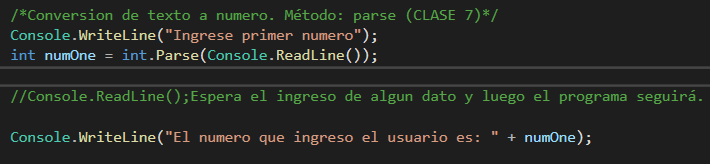


Clase 7:

# Conversión de texto a número:

Usaremos el método “parse”:

* Int.parse();
* Float.parse();
* Double.parse();
* Etc etc.



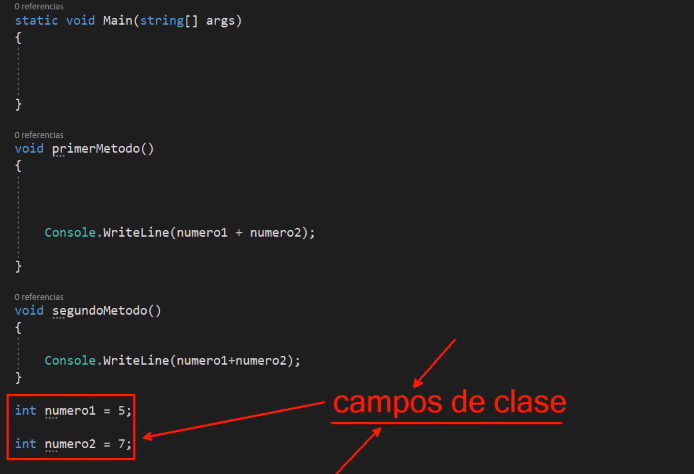
Clase 10 - 11: Métodos. (expression-bodied). “Lambda”



Clase 12:

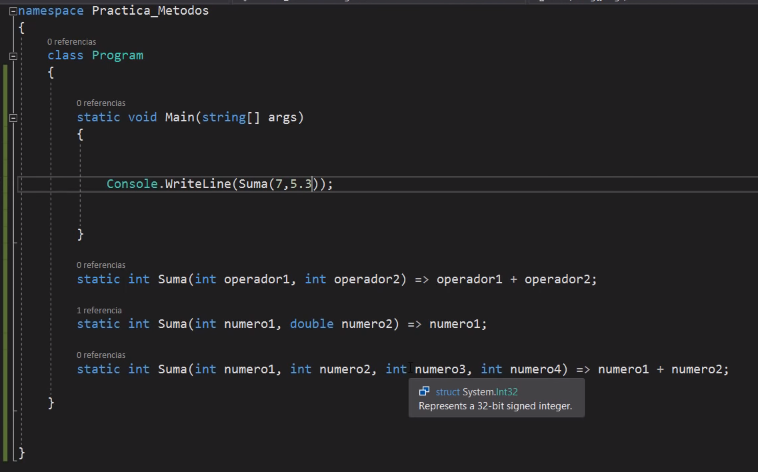
# Ámbito y sobrecarga de métodos:

Las variables/constantes que están declaradas en ámbito de clases se las denomina como “Campos” o “Campos de clase”.



# Sobrecarga de métodos:

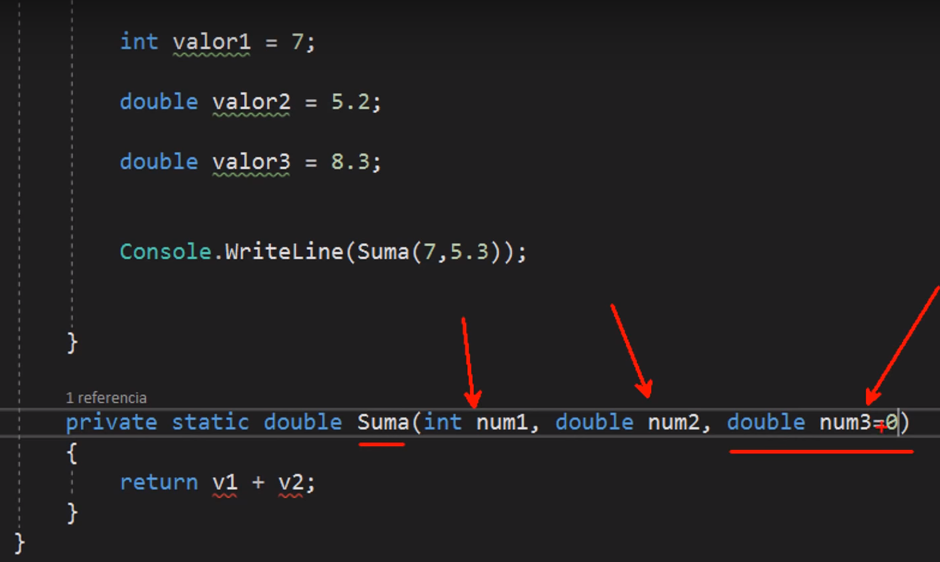
* Es cuando tienes un mismo método (mismo nombre) pero que contenga/reciban diferentes parámetros.

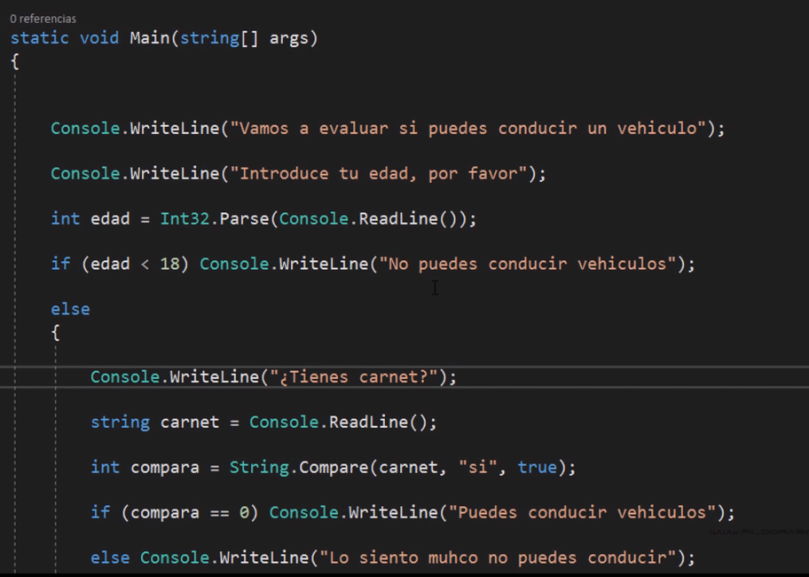


Clase 13: Métodos vs parámetros opcionales.

# Parámetros opcionales:

* Sirven para que puedan coexistir y trabajar en equipo a la hora de usar C# con otros lenguajes de programación. Algunos lenguajes no admiten sobrecargas y por eso se usan los parámetros opcionales.
* Ejemplo: La función Suma va a tener dos parámetros obligatorios y el tercero opcional. (se le asigna un valor).
* Esto hace que a la hora de la llamada podremos pasarle dos parámetros o tres.
* REGLA: los parámetros opcionales tienen que ir OBLIGATORIAMENTE después de/l los parámetro/s obligatorio/s.



Clase 16:

Clase 22:

# Excepciones: (try – catch)

SE USAN PARA QUE EL PROGRAMA NO CAIGA.

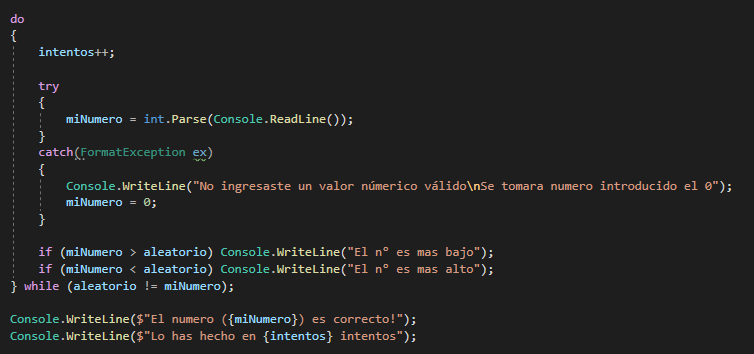
Son errores en tiempo de ejecución del programa que escapan al control del programador.

* Memoria corrupta.
* Desbordamiento de pila (stack).
* Sectores de disco duro defectuosos.
* Acceso a ficheros inexistentes.
* Conexiones a BBDD interrumpidas (Bases de Datos).

Try = intenta. (Metemos adentro las líneas que nos podrían llegar a ocasionar un error.

Ejemplo: a la hora de pedir un número, ingreso una cadena de caracteres.

Catch = Captura. (Código que se ejecuta si hay excepción).

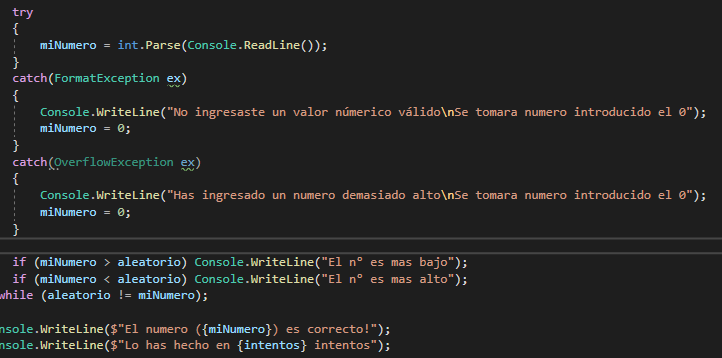


Clase 23:

# Captura de varias excepciones:

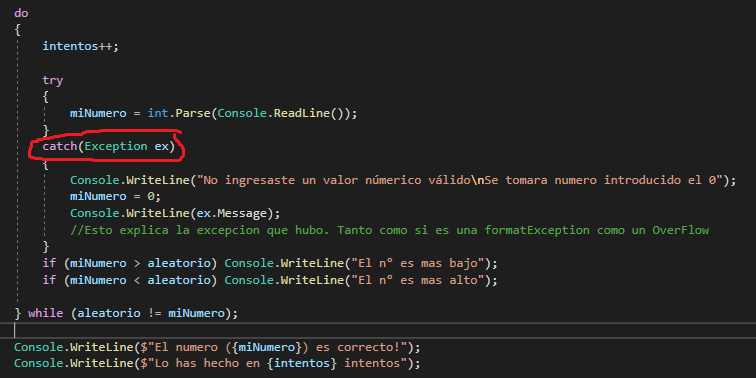
Pueden crearse varios Catch.

* En caso de un “FormatException + nombreVar” (es decir, que espere un número y el usuario ingrese una letra o string).
* Y también una excepción del tipo “OverFlowException + nombreVar” (es decir, que el usuario ingresa un número mucho más grande del tipo que espera y se produce un overflow).
* Se le asigna el nombre “e” o “ex” a la excepción por convención.



O también puede hacerse solamente llamando al objeto “Exception”.

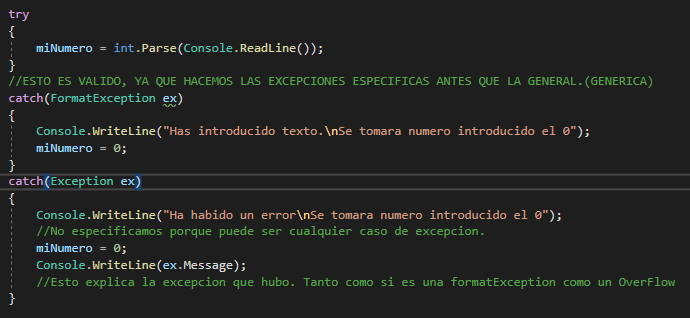
* Cuando se usa una captura ”Exception” (GENERAL) no tiene sentido usar las demás.
* PEEEEEEEEERO si puede ponerse una exception (ejemplo: FormatException) antes que la exception general. (Esto si lo toma como valido). (seria como un default???).



Clase 24:

# Conflictos con varias excepciones y captura con FILTROS:

* Primero van las excepciones específicas (FormatException, OverFlorException, etc).
* Luego van las generales. (Exception).



O puede usarse un método de la clase Exception, llamado .GetType();

* GetType() = Devuelve el tipo de exception que se ha lanzado.

Clase 25:

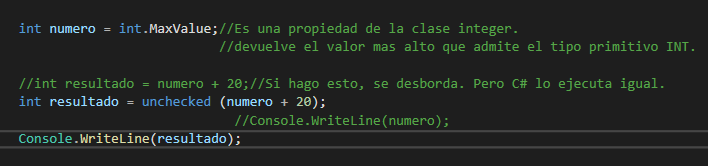
# Checked y unchecked:

# SOLAMENTE PUEDEN USARSE CON : int y long.

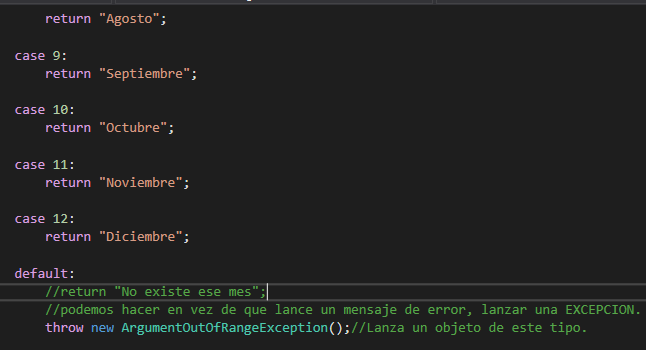
Checked puede activarse manualmente

* Click derecho en el proyecto.
* -Properties.
* -Build.
* -Advance.
* -Marcamos la opcion de checkear si hay desbordamiento.
* Y NOS MOSTRARIA CUANDO HAY UN OVERFLOW, ETC, ETC.

Unchecked puede usarse para que no checkee lo que encerramos entre paréntesis. (Lo contrario a checked).



# Lanzamientos de excepciones con “throw”:



Clase 27:

# POO: Programación orientada a objetos.

# ¿En qué consiste?

* Trasladar la naturaleza de los objetos de la vida real a código de programación.

# ¿Cuál es la naturaleza de un objeto de la vida real?

* Los objetos tienen un estado, un comportamiento (que puede hacer?) y unas propiedades.

Por ejemplo: un coche.

* ¿Cuál es el estado? Un coche puede estar: parado, circulando, aparcado, etc.
* ¿Qué propiedades tiene? Un coche tiene: color, un peso, un tamaño, etc.
* ¿Qué comportamiento tiene? Un coche puede: arrancar, frenar, acelerar, girar, etc.

------------------------------------------------------------------------------------------------------------------

# Ventajas:

* Programas divididos en “partes”,”trozos”,”módulos” o más bien llamado “clases”.
* Popularización.
* Esto hace muy utilizable al código.
* Si existe fallo en alguna línea de código, el programa continuara con su funcionamiento. Tratamiento de excepciones.
* Encapsulamiento.

------------------------------------------------------------------------------------------------------------------

# ¿Qué es una clase?

* Es un modelo, un molde, una plantilla donde se redactan las características comunes de un grupo de objetos.

# ¿Qué es un objeto?

Tiene:

* Propiedades/características/atributos.

Ejemplo: color, peso, ancho, alto, etc.

* Comportamientos. ¿Qué es capaz de hacer? //METODOS O FUNCIONES.

Ejemplo: arrancar, frenar, acelerar, etc.

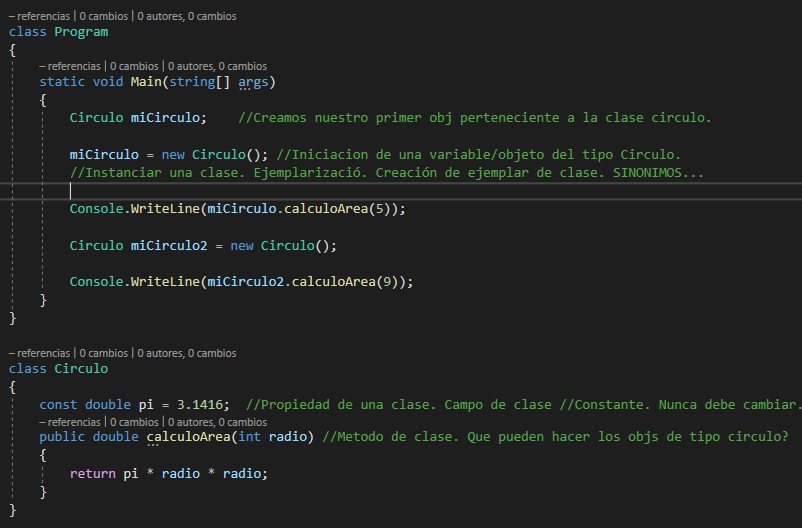
# Accediendo a propiedades de objeto desde código (pseudocodigo):

* Renault.color = “rojo”;
* Renault.peso = “1500”;
* Renault.altura = “900”;
* Renault.ancho = “3500”;

# Accediendo a comportamientos de objeto desde código (pseudocodigo): //MÉTODOS.

* Renault.arranca();
* Renault.frena();
* Renault.gira();
* Renault.acelera();

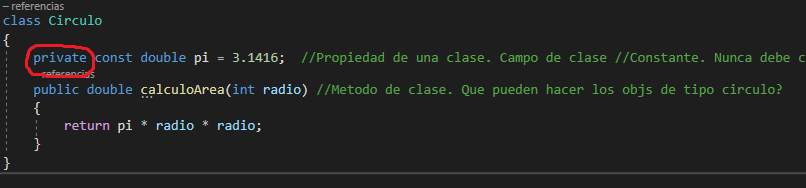
Clase 28:



Clase 29:

# Encapsulación y convenciones:

PALABRA RESERVADA PARA ENCAPSULAR DATOS USAMOS EL MODIFICADOR:

“private”: Solo es accesible desde la propia clase donde se encuentra.

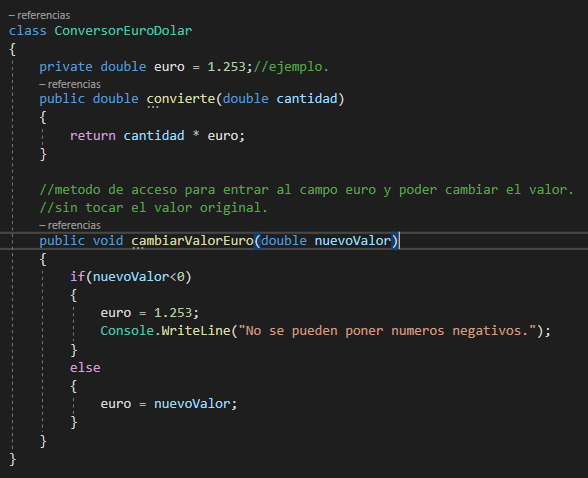
# Convenciones:

Los identificadores “public” deben comenzar con letra Mayuscula.

* Notación “PascalCase”.
* Ejemplo: public double CalculoArea();

Los identificadores que NO son “public” deben comenzar con letra minuscula.

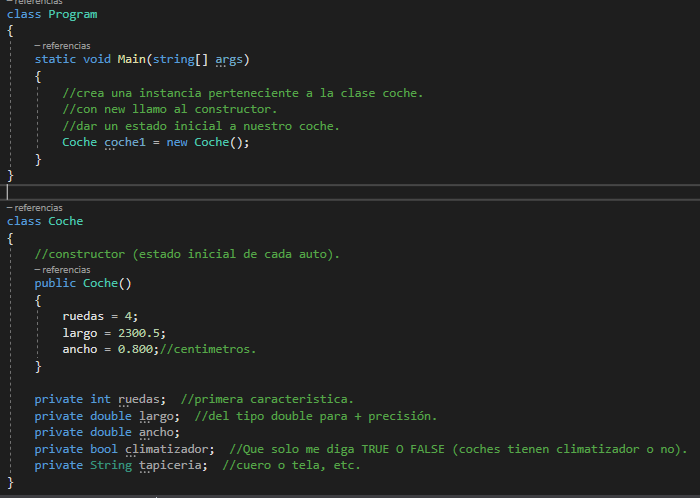
* Notación “camelCase”.
* Ejemplo : longitudRadio;



Clase 29:

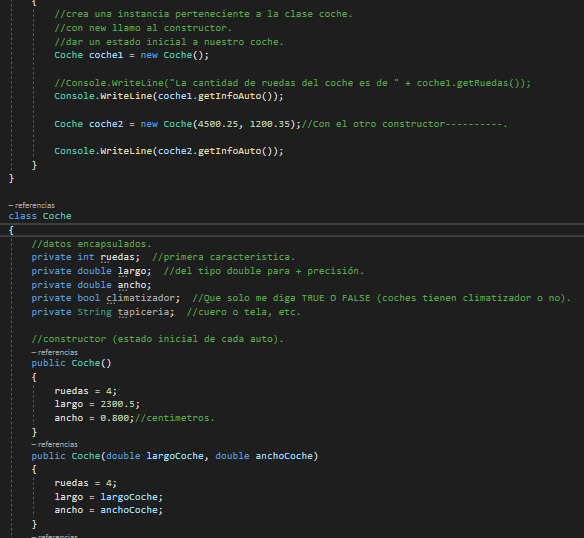
# Constructor:

* Un método que tiene como nombre **OBLIGATORIAMENTE** el mismo nombre que la clase.
* No podrá devolver nada.
* No será void.
* Si no se declara adentro de la clase se dice que hay un constructor por defecto. (del tipo public).



# Sobrecarga de constructores:

* 2 – 4 constructores que diferentes parámetros.
* Se produce en una clase cuando la clase tiene más de un constructor.



Clase 31:

# Getters y setters:

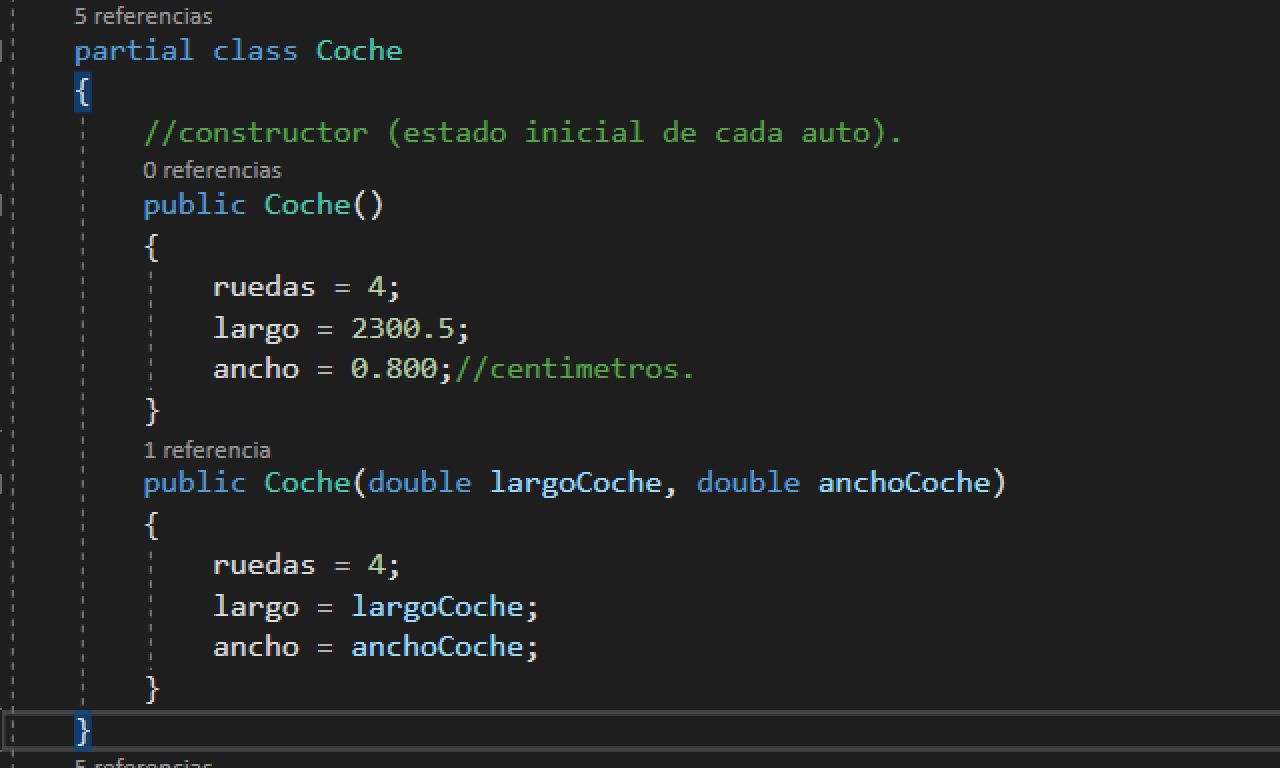
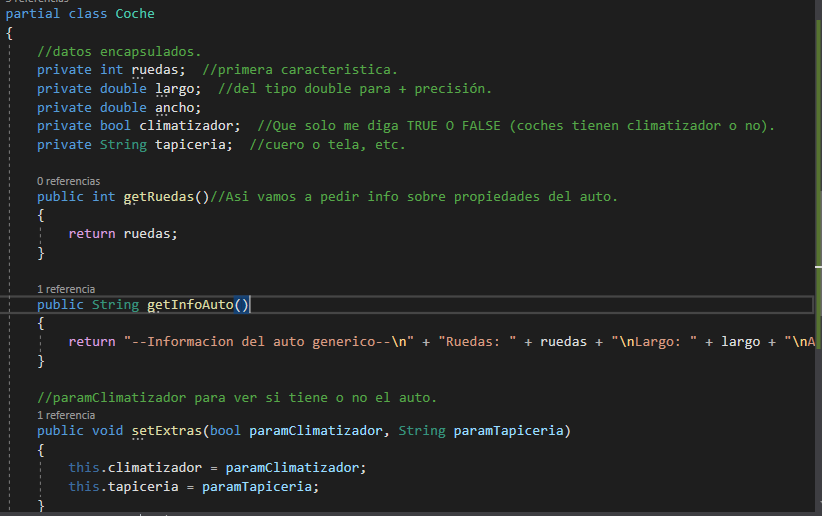
Getters: Establecer propiedades a los objetos.

* No devuelve nada.
* Su objetivo es establecer un objeto, dar valor. No devolver nada.

Setters:

Partial class Coche:

* Puedo usarlo para dividir una clase en trozos.
* Por ejemplo: una parte para los constructores y otra para los getters y setters.



Clase 32-33:

# Visual Studio y POO:

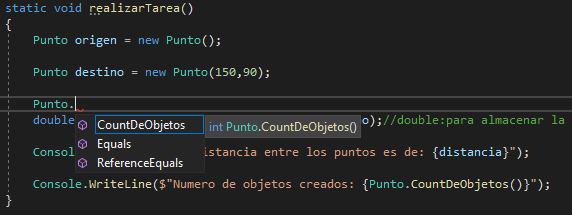
# Llamadas y clase Math:

Clase 34:

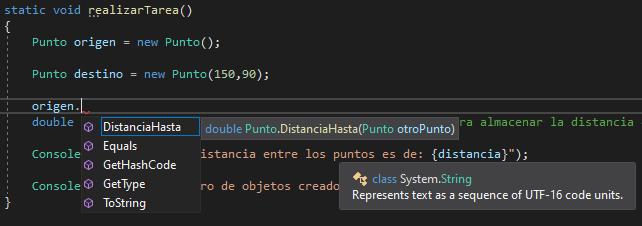
# Variables y método static:

Variables static: constante de clase. Pertenece a la propia clase.

* Ninguno de los obj o instancias que pertenezcan a la clase puedan actuar sobre esta variable estatica.
* Quien puede actuar sobre esta variable static? Solamente la dueña.
* Si es de este tipo solamente se puede entrar al método que creamos (también static) con el tipo de formato. En este caso “Punto”.



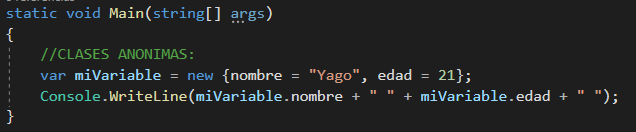
* En cambio si queremos acceder desde una variable del tipo Punto, no podemos.



Clase 35:

# Clases anónimas:

* Solo pueden contener campos públicos.
* Todos los campos deben estar iniciados. Ejemplo: {Nombre = “Yago”, edad = 21};
* Los campos no pueden ser static.
* No se pueden definir métodos.



Clase 36:

# Arrays:

* ¿Qué es? Estructura de datos que contiene una colección de valores del mismo tipo.
* ¿Para qué? Para almacenar valor que normalmente tienen alguna relación entre sí.
* SINTAXIS:

Declaracion: int [] miArray;

Iniciacion: miArray = new int[4]; Va a almacenar 4 valores.