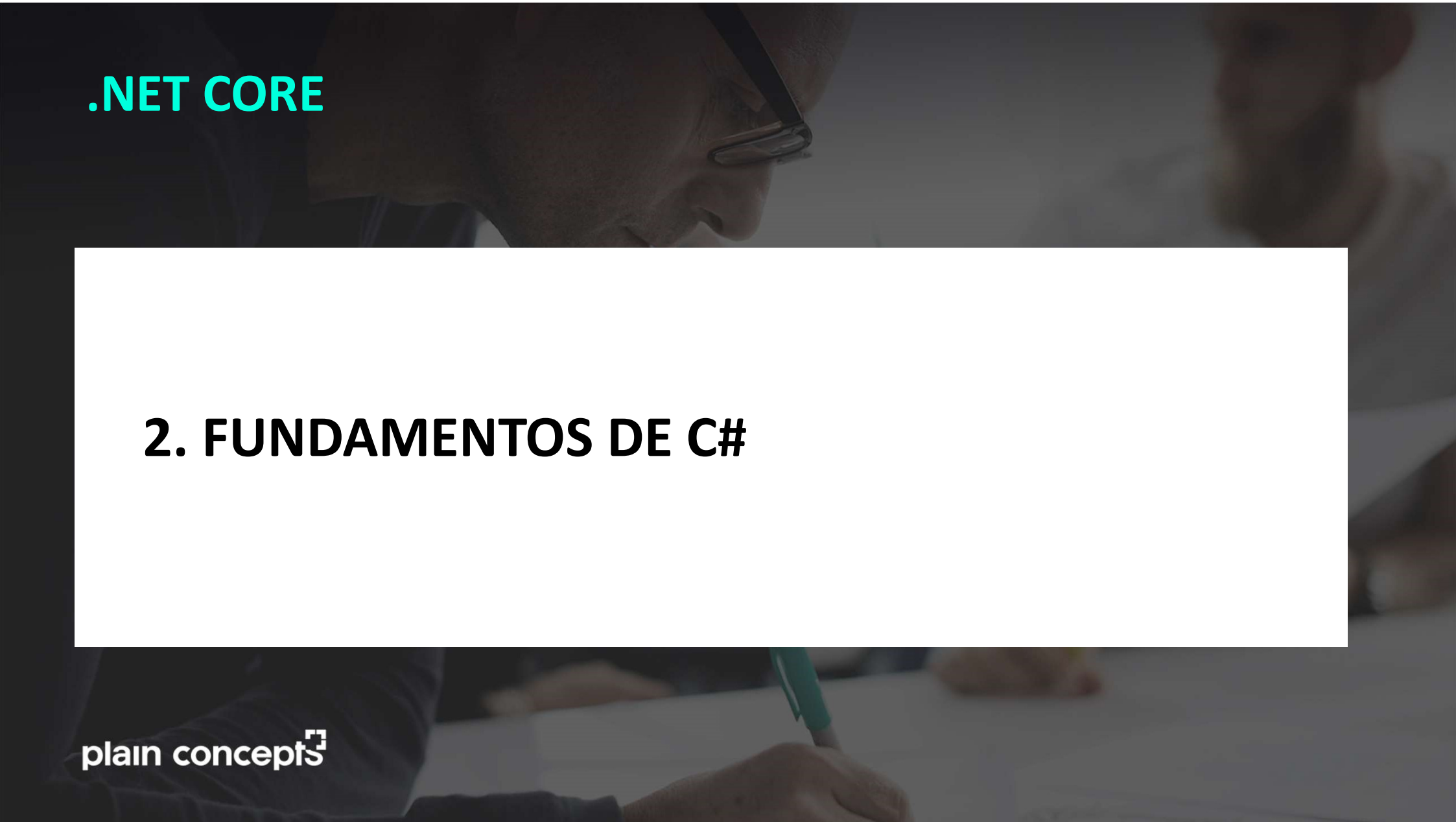
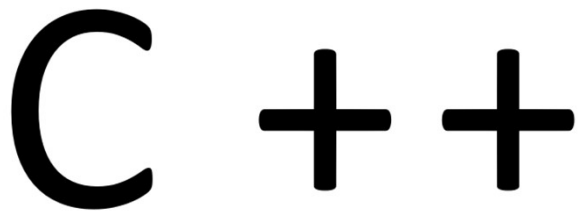

.NET CORE
2. FUNDAMENTOS DE C#
plain concepts

# El lenguaje C#

## Lenguaje C#

- Anders Hejlsberg, Scott Wiltamuth y Peter Golde fueron los principals inventores del lenguaje. Fue impulsado y utilizado por **Microsoft** desde **Junio del 2000** de manera oficial.
- Lenguaje basado en C/C++ que coge también aspectos de Java y Delphi. Al considerarlo una evolución decidieron llamarlo C++++ que equivale a C# (pronuncionado C Sharp)
- Actualmente vamos por la version 8 de C# que salió a principios de 2019 https://www.developerro.com/2019/02/19/csharp-8/
- Historia de C#: https://www.developerro.com/2019/02/05/historia-csharp/



# El lenguaje C#

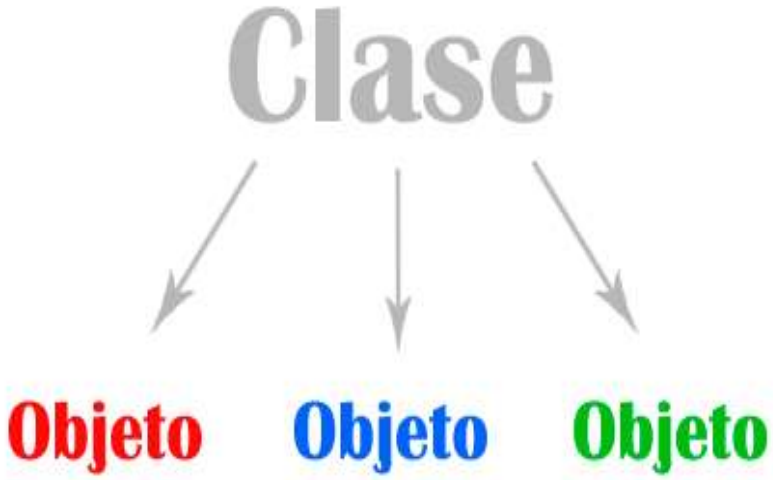
## ¡Hola mundo! en .NET Core

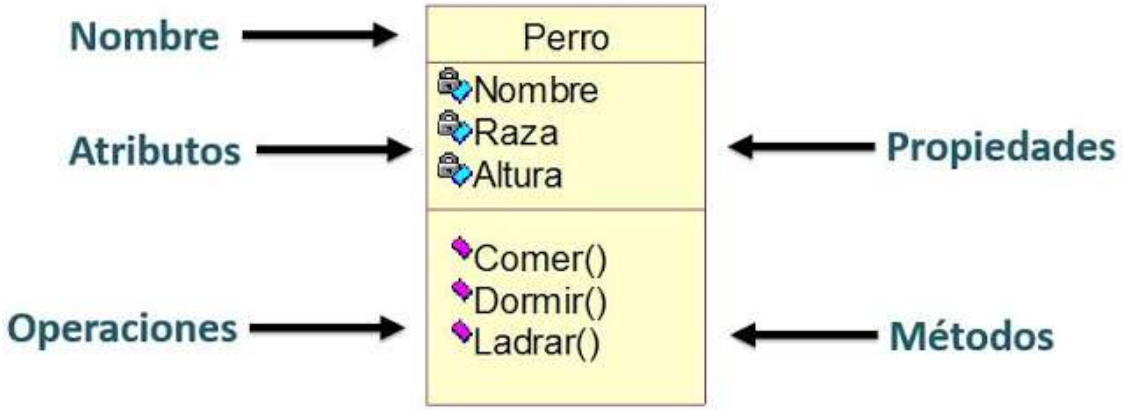
- La template del Proyecto de consola nos muestra como sería un ¡Hola Mundo!
- El método Main es el que abre la ejecución en todas las aplicaciones de .Net Core.
- Todo en C# son objetos.

# El lenguaje C#

## Clases y Objetos

- Una clase es la estructura fundamental de C#
- Combina estados (campos) y acciones (funciones)
- Una clase proporciona una definición para instancias creadas dinámicamente de la clase, también conocidas como objetos.



# El lenguaje C#

## Clases y Objetos

CLASE: Persona

```
2 references
public class Person
{
    2 references
    public string Name { get; set; }
    2 references
    public string Surname { get; set; }
    2 references
    public int Age { get; set; }

    1 reference
    public Person(string name, string surname, int age)
    {
        Name = name;
        Surname = surname;
        Age = age;
    }

    0 references
    public bool IsLegalAge()
    {
        return Age >= 18;
    }

    1 reference
    public string GetCompleteName()
    {
        return $"{Name} {Surname}";
    }
}
```

OBJETO: person1

```
0 references
static void Main(string[] args)
{
    var person1 = new Person("Juan", "Sanchez", 25);

    Console.WriteLine(person1.GetCompleteName());
    Console.ReadKey();
}
```

- La **Clase** representa **la estructura** del **Objeto.**
- El **Objeto** es el que **interactúa en el código.**
- El **Objeto** se crea a partir del **constructor** de la **Clase**
- El **Objeto** puede **cambiar el estado** de sus propiedades a lo largo del código.

# El lenguaje C#

## Herencia de clases

```
3 references
public class Person
{
    2 references
    public string Name { get; set; }
    2 references
    public string Surname { get; set; }
    3 references
    public DateTime Birthdate { get; set; }

    2 references
    public Person(string name, string surname, DateTime birthdate)
    {
        Name = name;
        Surname = surname;
        Birthdate = birthdate;
    }

    Public Functions
}
```

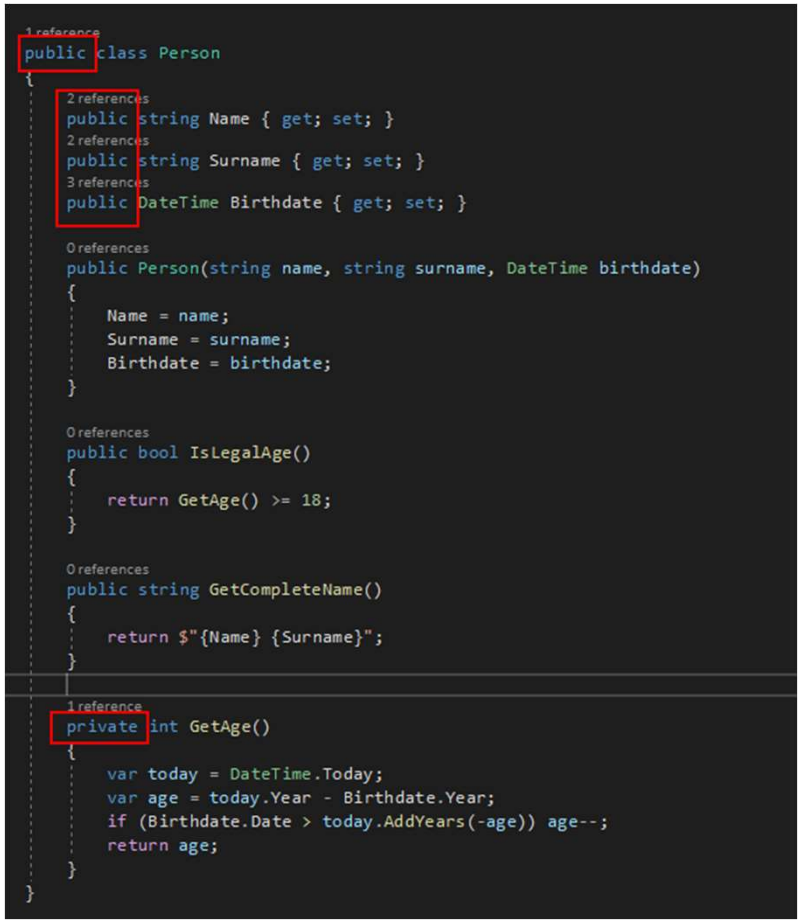
```
1 reference
public class Employee : Person
{
    0 references
    public int Salary { get; private set; }
    1 reference
    public int ContractType { get; set; }

    0 references
    public Employee(string name, string surname, DateTime birthdate, int contractType)
        : base(name, surname, birthdate)
    {
        ContractType = contractType;
    }

    0 references
    public int CalcularSueldo()
    {
        throw new NotImplementedException();
    }
}
```

```
1 reference
public class Customer : Person
{
    1 reference
    public string CustomerCode { get; set; }
    1 reference
    public int Category { get; set; }

    0 references
    public Customer(string name, string surname, DateTime birthdate, int category)
        : base(name, surname, birthdate)
    {
        CustomerCode = this.GenerateCode();
        Category = category;
    }

    1 reference
    public string GenerateCode()
    {
        throw new NotImplementedException();
    }
}
```

# El lenguaje C#

## Accesibilidad

- Cada miembro de una clase tiene asociada una accesibilidad
- No solo las clases tienen accesibilidad, los métodos y las propiedades también.
- Para que una clase sea vista por otras tiene que tener la accesibilidad de “public”.
- Estos son los valores que podemos dar por accesibilidad:

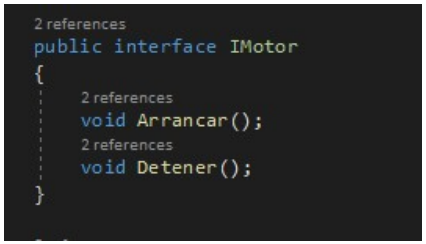


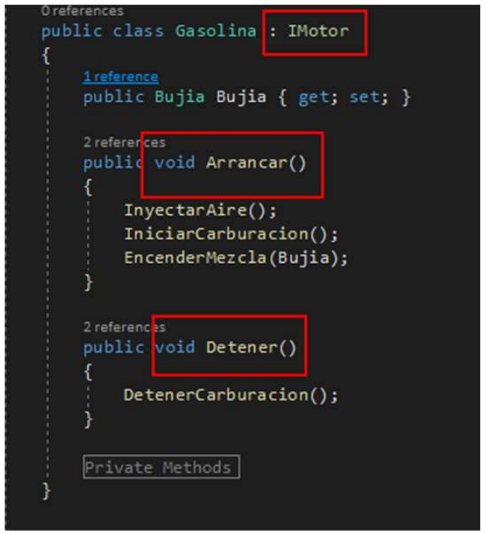

- ➢ public
  - Acceso no limitado
- ➢ protected
  - Acceso limitado a esta clase o a las clases derivadas de esta clase
- ➢ internal
  - Acceso limitado al ensamblado actual (.exe, .dll, etc.)
- ➢ protected internal
  - Acceso limitado a la clase contenedora, las clases derivadas de la clase contenedora, o bien las clases dentro del mismo ensamblado
- ➢ private
  - Acceso limitado a esta clase
- ➢ private protected
  - Acceso limitado a la clase contenedora o las clases derivadas del tipo contenedor con el mismo ensamblado
- ❑ Default
  - Por defecto la accesibilidad en C# es private!

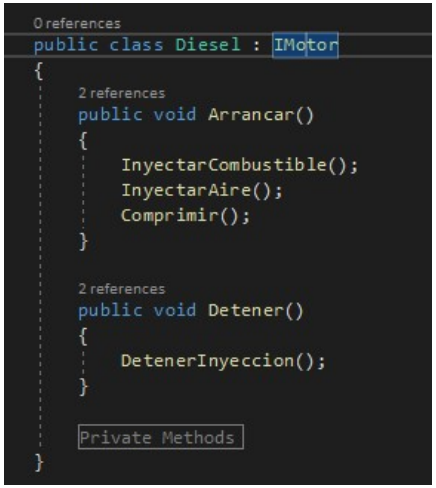
# El lenguaje C#

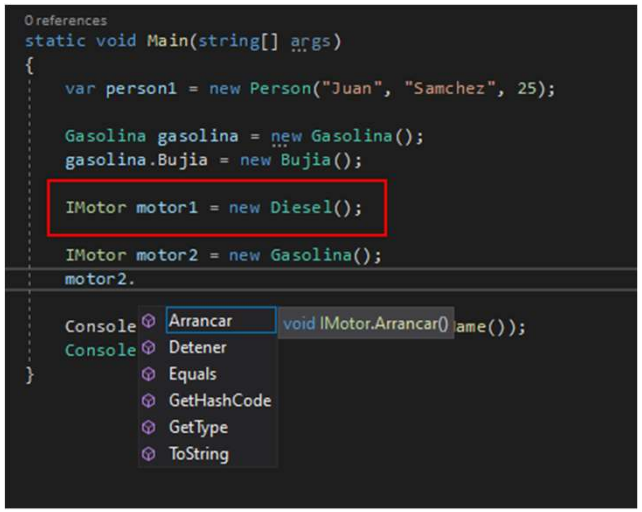
## Interfaces

- Un interfaz define un contrato que deben cumplir las clases que se lo asignan.
- Este contrato es una definición de la firma de métodos y propiedades que debe tener la Clase.
- Los nombres de los interfaces por nomenclatura suelen comenzar con la letra I.



# El lenguaje C#

## Métodos Estáticos

- En C# podemos definir métodos que se crean independientemente a la definición de objetos. Un método estático puede llamarse sin tener que crear un objeto de dicha clase. Un método estático tiene ciertas restricciones:
    - No puede acceder a los atributos de la clase (salvo que sean estáticos)
    - No puede utilizar el operador this, ya que este método se puede llamar sin tener que crear un objeto de la clase.
    - Puede llamar a otro método siempre y cuando sea estático.
    - Un método estático es lo más parecido a lo que son las funciones en los lenguajes estructurados (con la diferencia que se encuentra encapsulado en una clase)

```
0 references
static void Main(string[] args)
{
    var x = 10;
    var y = 3;
    Console.WriteLine($"El resultado de sumar {x} + {y} es: {CalculadoraEstatica.Sumar(x, y)}");
    Console.ReadKey();
}
```

```
0 references
public class CalculadoraEstatica
{
    0 references
    public static int Sumar(int x, int y)
    {
        return x + y;
    }
}
```

# El lenguaje C#

## Métodos de Extensión

- Los métodos de extensión permiten "agregar" métodos a los tipos existentes sin crear un nuevo tipo derivado, recompilar o modificar de otra manera el tipo original.
- Son un tipo especial de métodos estáticos, pero se les llama usando la sintaxis de método de instancia.
- El primer parámetro especifica en qué tipo funciona el método, y el parámetro está precedido del modificador this.
- Los métodos de extensión únicamente se encuentran dentro del ámbito cuando el espacio de nombres se importa explícitamente en el código fuente con una directiva using.

```
0 references
static void Main(string[] args)
{
    var phrase = "Estoy en el curso de NET Core.";
    var words = phrase.WordCount();
    Console.WriteLine($"Palabras que contiene la frase {phrase}: {words}");
    Console.ReadKey();
}
```

```
0 references
public static class StringExtensions
{
    1 reference
    public static int WordCount(this string str)
    {
        return str.Split(
            new char[] { ' ', '.', '?' },
            StringSplitOptions.RemoveEmptyEntries)
            .Length;
    }
}
```

# Demo

# Aplicación de Consola

plain concepts

# Ejercicios

## Crear aplicación de consola

plain concepts

# El lenguaje C#

## Atributos en C#

- Los atributos proporcionan una manera de asociar la información con el código de manera declarativa.
- También pueden proporcionar un elemento reutilizable que se puede aplicar a diversos destinos.
- Existen atributos ya definidos en C# y puedes crear tus propios atributos personalizados.

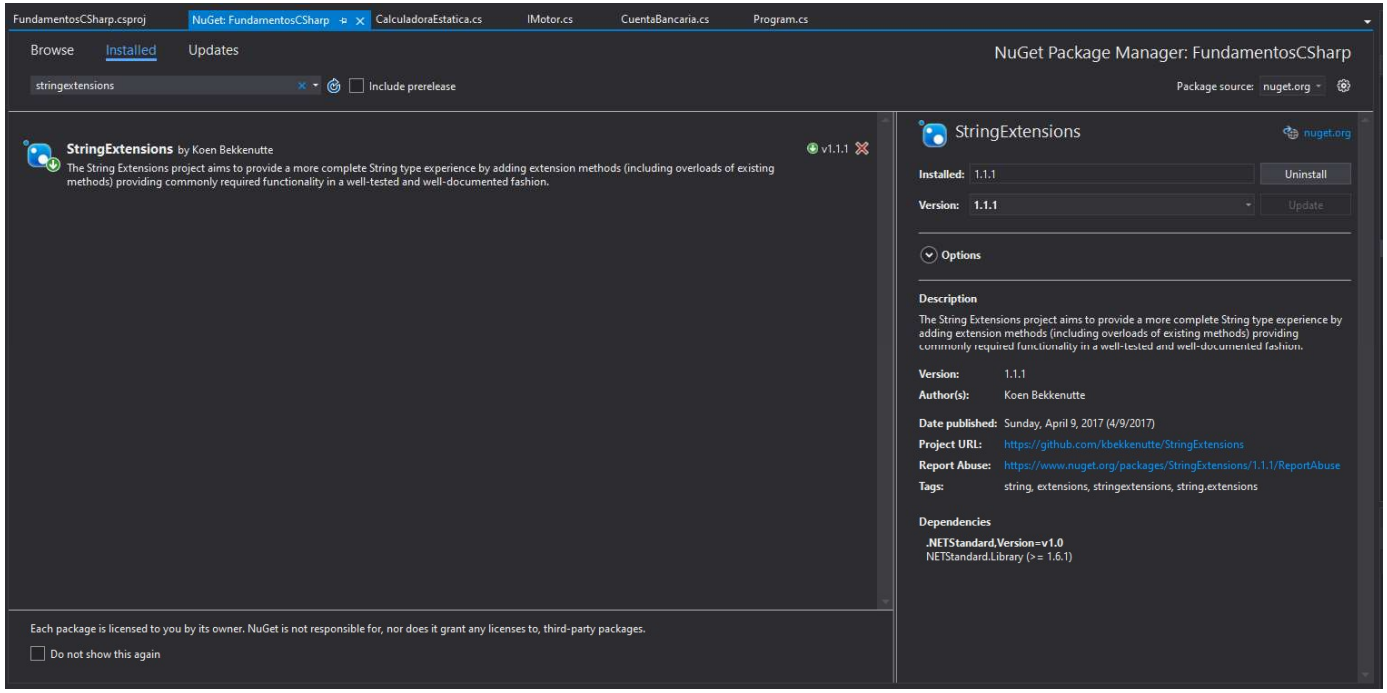
```
C#

[Obsolete("ThisClass is obsolete. Use ThisClass2 instead.")]
public class ThisClass
{

}
```

# El lenguaje C#

## Paquetes Nuget

- En .NET (incluido .NET Core), el mecanismo compatible con Microsoft para **compartir código** es **NuGet**, que define cómo se crean, hospedan y consumen paquetes en .NET, y ofrece las herramientas para cada uno de esos roles.

- Desde un punto de vista sencillo, un paquete NuGet es un archivo ZIP con la extensión .nupkg que contiene código compilado



# El lenguaje C#

## Async Task - Async/await

- El modelo de programación asincrónica de tareas (TAP) es una abstracción del código asincrónico.
- Si devolvemos un objeto Task en el método, podemos hacer que este sea async, con la palabra async.
  - Task<T>
- Si nuestro método es asíncrono, podemos usar await para esperar a funcionar que son asíncronas en el interior de nuestro método.