Bloque 2. Visual Basic .NET
UT 3. Confección de interfaces de usuario y creación de componentes visuales

TEMA 1

Tecnología .NET

1. Introducción 1
1.1. ¿Qué es el .NET Framework? 2
2. Primera aplicación con VB.NET 8
2.1. Primera aplicación de consola. 9
2.2. Primera aplicación de Windows. 15
2.3. Ficheros del proyecto. 18
3. Más conceptos de Visual Basic .NET. 18
4. Bases de la Programación Orientada a Objetos (POO) 23
4.1. Las clases 23
4.2. Los Objetos 23

1. Introducción

Hace muchos años las aplicaciones se creaban utilizando un mismo lenguaje para todas las tareas y para un sistema operativo concreto. Inicialmente las aplicaciones para Windows se realizaban en C y con llamadas directas al "núcleo" del Windows con la API. El trabajo era enorme por dos cosas: la complejidad del lenguaje y la necesidad de conocer todos los detalles del sistema operativo para poder programar sobre él.

Ahí nació Visual Basic, como una herramienta que permitía el desarrollo de las interfaces de una forma tremendamente sencilla para lo conocido hasta entonces y con un lenguaje conocido por entonces como el Basic. Se utilizaban los controles en formato VBX para diseñar las ventanas y luego un mecanismo de métodos y eventos para controlar el código.

La parte profesional se realizaba en el lenguaje "C++", un lenguaje orientado a objetos muy complejo y difícil de mantener. Por otro lado, Visual Basic seguía avanzando con los COM (Component Object Model): un modelo de objetos que permitía la reutilización de componentes independiente del lenguaje con el que estuviesen escritos, así transcurrieron las versiones 5 y 6 de Visual Basic que se convirtieron en los entornos más importantes del panorama Windows.

Con el tiempo además de los COM aparecieron otras técnicas complementarias para ampliar los lenguajes existentes. Pero esta heterogeneidad de tecnologías, muchas veces dictadas por las necesidades tecnológicas del momento, como Internet, hicieron que se produjeran multitud de servicios duplicados, creación de servicios exclusivamente para algunos lenguajes, poca reutilización de código, más complejidad,...

La solución definitiva se ha planteado con .NET que consta de una serie de servicios iguales en todos los lenguajes que mantienen la integridad con los desarrollos existentes y hace posible una interoperatividad entre los lenguajes desconocida hasta el momento. Esto es, podemos utilizar varios lenguajes diferentes (los permitidos .NET) y todos tendrán disponibles desde el mismo entorno de desarrollo hasta los controles y componentes de programación. Se acabó que Visual C tenga estos componente o Visual Fox Pro otros o Visual Basic otros distintos... todos parten de la tecnología .NET y comparten, por lo tanto, los mismos componentes y objetos. Podemos incluso escribir partes distintas de un mismo programa con varios lenguajes .NET.

Visual Basic .NET usa una jerarquía de clases que están incluidas en el .NET Framework, por tanto, conocer el .NET Framework nos ayudará a conocer al propio Visual Basic .NET, aunque también necesitaremos conocer la forma de usar y de hacer del VB.NET ya que, aunque en el fondo sea lo mismo, el aspecto sintáctico es diferente para cada uno de los lenguajes basados en .NET Framework.

Importante: Luego el verdadero núcleo de todo es .NET Framework un entorno con multitud de clases y objetos disponibles para trabajar con ellos. Sobre este núcleo tendremos los lenguajes de programación y uno de ellos es Visual Basic .NET. Por eso es muy importante conocer que es el .NET Framework

1.1. ¿Qué es el .NET Framework?

Mediante la plataforma .NET tenemos a nuestra disposición un conjunto de herramientas y tecnologías que permiten desarrollar aplicaciones destinadas a plataformas muy variadas:

- Aplicaciones Windows (aplicaciones de ventanas, aplicaciones de consola, servicios Windows).
- Aplicaciones y servicios web.
- Aplicaciones para smartphones.
- Aplicaciones orientadas a tabletas táctiles.
- Aplicaciones para sistemas embebidos.

Estos distintos tipos de aplicaciones pueden desarrollarse gracias a un elemento común: el **framework .NET**. Este framework es una solución de software que incluye varios componentes dedicados al desarrollo y ejecución de las aplicaciones. Lo facilita Microsoft con el sistema operativo Windows y está disponible para otros sistemas mediante soluciones de software de terceros, como Mono.

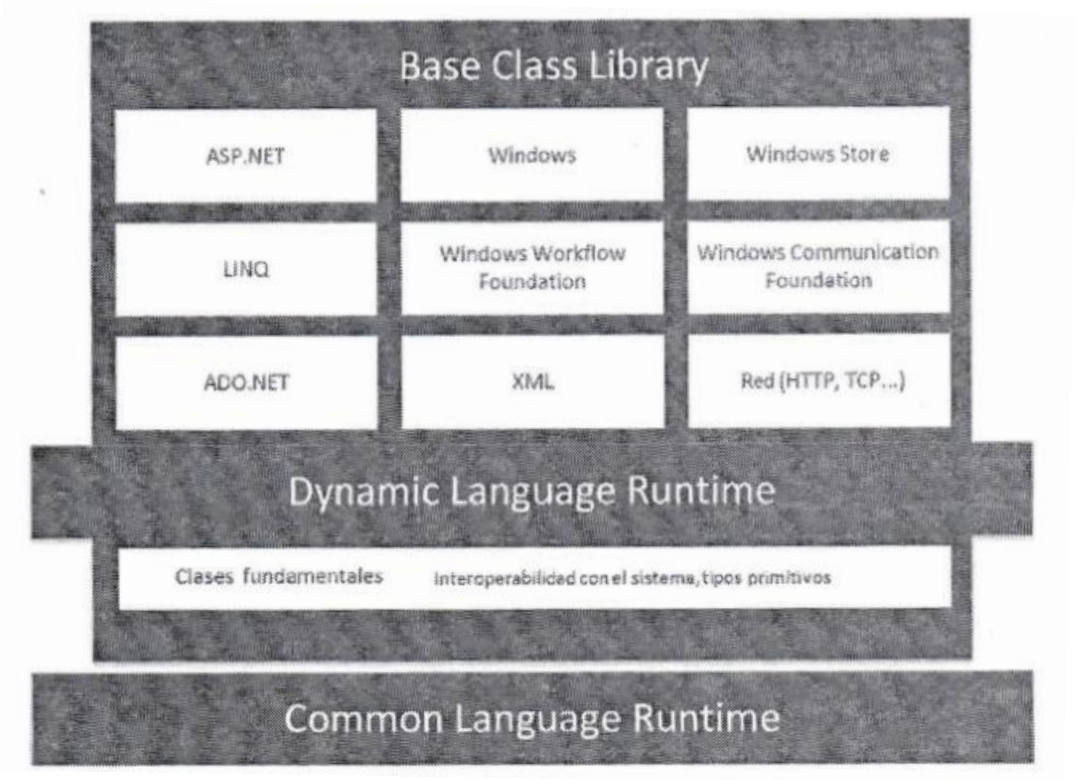
Los elementos que forman el núcleo del framework .NET son:

- El **Common Language Runtime (CLR)**
- El **Dynamic Language Runtime (DLR)**
- La **Base Class Library (biblioteca de clases básicas)**

El Common Language Runtime es un entorno que permite ejecutar código .NET y que asegura la gestión de la memoria. El código gestionado por el CLR se denomina generalmente código manejado.

El Dynamic Language Runtime es un complemento al Common Laguage Runtime, desarrollado en C#, que provee capacidades de ejecución de código dinámico. Gracias a él, lenguajes dinámicos como PHP o Python están soportados por el framework .NET.

La Base Class Library es un conjunto de clases que exponen funcionalidades habituales y que pueden utilizarse en cualquier tipo de aplicación. Está formada por varios bloques independientes que se han ido agregando progresivamente a lo largo de las ediciones del framework .NET.



Desde sus inicios Windows expone funcionalidades a los desarrolladores a través de interfaces de programación (en inglés **API**, *Application Programming Interface*) que les permiten crear aplicaciones. Con la complejidad que ha adquirido Windows a lo largo de los años, distintas API han ido apareciendo y evolucionando:

- API Windows (Win16, Win32 y Win64): estas API forman el núcleo de Windows y las utiliza directamente el sistema. Están destinadas a utilizarse al más bajo nivel, con el lenguaje C. Win16 apareció en Windows 3.1, mientras que Win32 se utiliza desde Windows 95 y Win64 está integrado en los sistemas de 64 bits desde Windows XP.
- MFC (Microsoft Foundation Classes) existe desde 1992 y encapsula las API Win16 y Win32 en una estructura orientada a objetos en C++.
- COM (Component Object Model) aparece poco después de las MFC para responder a una problemática de comunicación entre procesos. A continuación la librería ATL (Active Template Library) encapsuló las API COM para simplificar su uso.

La plataforma .NET apareció a principios de los años 2000 para dar respuesta al problema de la complejidad de estas API. Las unifica y las moderniza, permitiendo su uso con cualquier lenguaje que posea una implementación compatible.

La unificación de las API y de los modelos de desarrollo también ha tocado al mundo de la Web. Desde mediados de los años 90, Microsoft facilita del lado del servidor, el motor de script ASP (Active Server Page), que se dejó de usar en beneficio de un componente integrado en el framework .NET: ASP.NET. Este ha permitido mejorar el rendimiento de las aplicaciones web introduciendo código compilado y no interpretado. También ha mejorado su mantenibilidad remplazando la programación procedural típica de los lenguajes de script con un modelo de programación de eventos y orientado a objetos.

Desde su aparición en el año 2002, han aparecido 16 versiones diferentes de la plataforma .NET. A continuación se muestran de manera no exhaustiva las versiones a partir del año 2010.

Versión 4.0

Fecha de aparición: abril de 2010

IDE asociado: Visual Studio 2010

Funcionalidades asociadas:

- Introducción del DLR (Dynamic Language Runtime) para la interfaz con lenguajes dinámicos.
- Se incluyen las Parallel Extensions para gestionar el multithreading y la ejecución paralela de código.
- Mejora del rendimiento del Garbage Collector.

Versión 4.5

Fecha de aparición: agosto de 2012

IDE asociado: Visual Studio 2012

Funcionalidades asociadas:

- Soporte de aplicaciones para Windows 8
- Extensión de C# y VB.NET para incluir el asincronismo directamente en los lenguajes.
- Nueva API para la comunicación HTTP
- Mejora del rendimiento web con la minificación y el asincronismo.

Versión 4.5.1

Fecha de aparición: octubre de 2013

IDE asociado: Visual Studio 2013

Funcionalidades asociadas:

- Soporte de aplicaciones para Windows 8.1

Versión 4.5.2

Fecha de aparición: mayo de 2014

IDE asociado: Visual Studio 2013

Funcionalidades asociadas:

- Nuevas API para ASP.NET

Versión 4.6

Fecha de aparición: verano de 2015

IDE asociado: Visual Studio 2015

Funcionalidades asociadas:

- Unificación de las API Web en el framework MVC 6.
- .NET Core.
- Compilación de las aplicaciones Windows Store en código nativo con .NET Native.
- Nuevo compilador “Just-in-time” con mejor rendimiento: RyuJIT.
- Actualización de Windows Forms y WPF para el soporte a monitores con una mayor densidad de píxeles (High DPI).
- Soporte de nuevos algoritmos criptográficos.

Versión 4.6.1

Fecha de aparición: noviembre de 2015

IDE asociado: Visual Studio 2015

Funcionalidades asociadas:

- Mejoras en WPF (corrección ortográfica integrada, funcionalidades táctiles,…)-
- Mejora de rendimiento y de estabilidad.

Versión 4.6.2

Fecha de aparición: marzo de 2016

IDE asociado: Visual Studio 2015

Funcionalidades asociadas:

- Mejora de la criptografía.
- Mejora del soporte a pantallas con una gran densidad de píxeles.

Versión 4.7

Fecha de aparición: mayo de 2017

IDE asociado: Visual Studio 2017

Funcionalidades asociadas:

- Mejora de la criptografía.
- Mejora del protocolo de seguridad TLS.
- Nuevas API WPF dedicadas a la impresión.
- Mejora del soporte a monitores de gran densidad de píxeles.

Versión 4.7.1

Fecha de aparición: octubre de 2017

IDE asociado: Visual Studio 2017

Funcionalidades asociadas:

- Mejora de la accesibilidad.
- Compatibilidad con .NET Standard 2.0
- Compatibilidad con los generadores de configuración.
- Mejora del soporte a redes.

Versión 4.7.2

Fecha de aparición: abril de 2018

IDE asociado: Visual Studio 2017

Funcionalidades asociadas:

- Mejora de la criptografía.
- Mejora de la descompresión de archivos ZIP y API de colección adicionales.

Versión 4.8

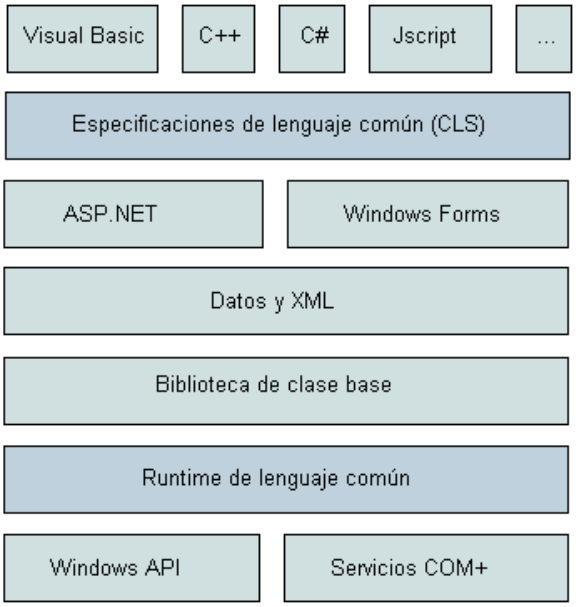
Fecha de aparición: abril de 2019

IDE asociado: Visual Studio 2019

Funcionalidades asociadas:

- Mejora para monitores de alta resolución.
- Mejora para la realización de actualizaciones.
- Mejoras de seguridad.

La mejor forma de comprender cómo funciona .NET es analizar las numerosas capas en las que se divide .NET Framework. Veamos la siguiente figura:



Como hemos comentado antes, .NET Framework constituye la base sobre la que se asienta .NET. Podemos agrupar en tres bloques el conjunto de herramientas y servicios:

- El **Common Language Runtime (CLR)**
- El **Dynamic Language Runtime (DLR)**
- La **Base Class Library (biblioteca de clases básicas)**

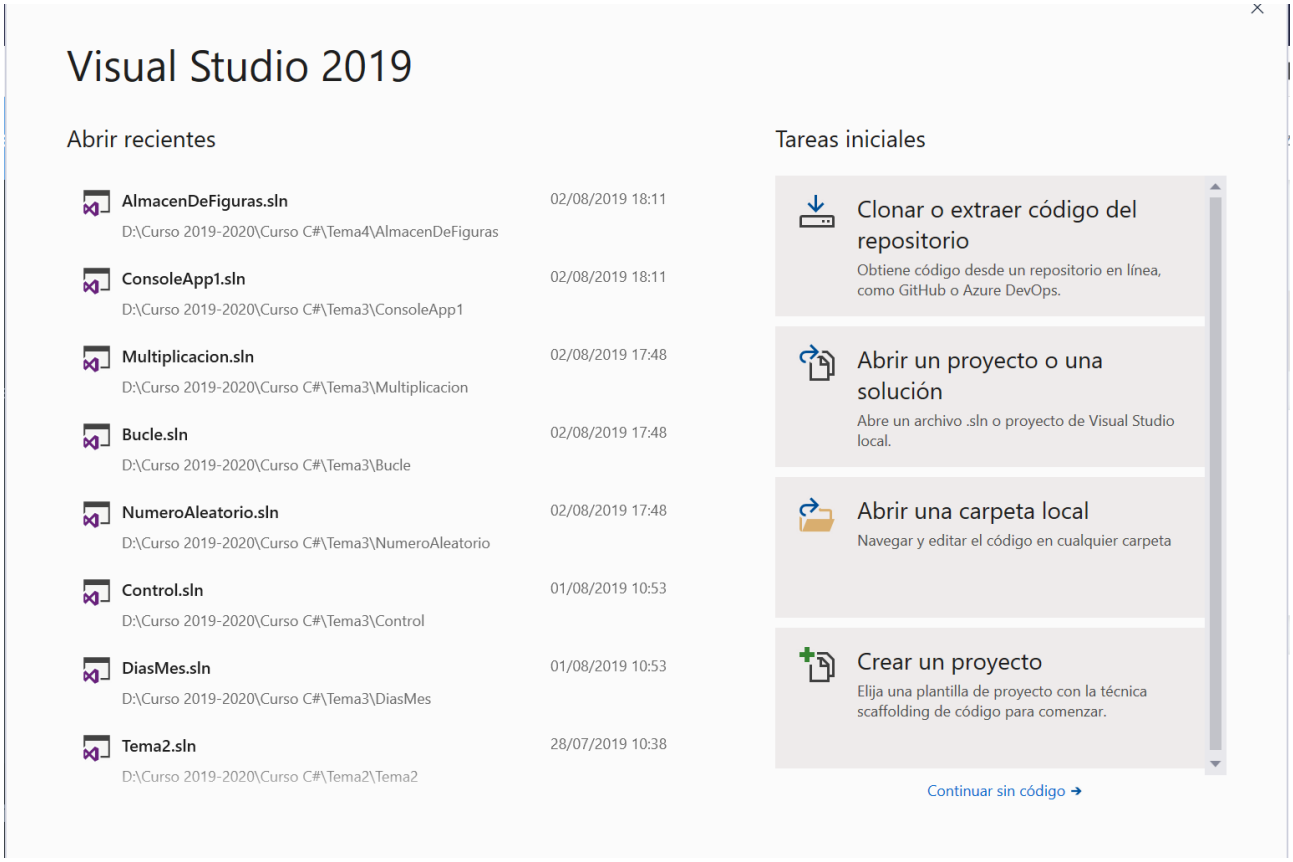
Estas son tres de las capas del esquema anterior. Simplemente esa jerarquía indica que en el nivel mas alto están los lenguajes que vamos a utilizar en nuestro desarrollo con .NET que pueden ser varios (en nuestro caso Visual Basic). Los dos principales lenguajes de Microsoft son C# y VB. Luego traduce esas instrucciones al estándar .NET para poder trabajar con esas instrucciones de una forma independiente al lenguaje con que se escribieron. El siguiente paso es la generación de formularios para Windows o web (ASP.NET). Luego el enlace con bases de datos si las hay. Por

fin llegamos a todos los objetos disponibles y por último el “runtime de lenguaje común” (CLR) que es el que va a ejecutar la aplicación.

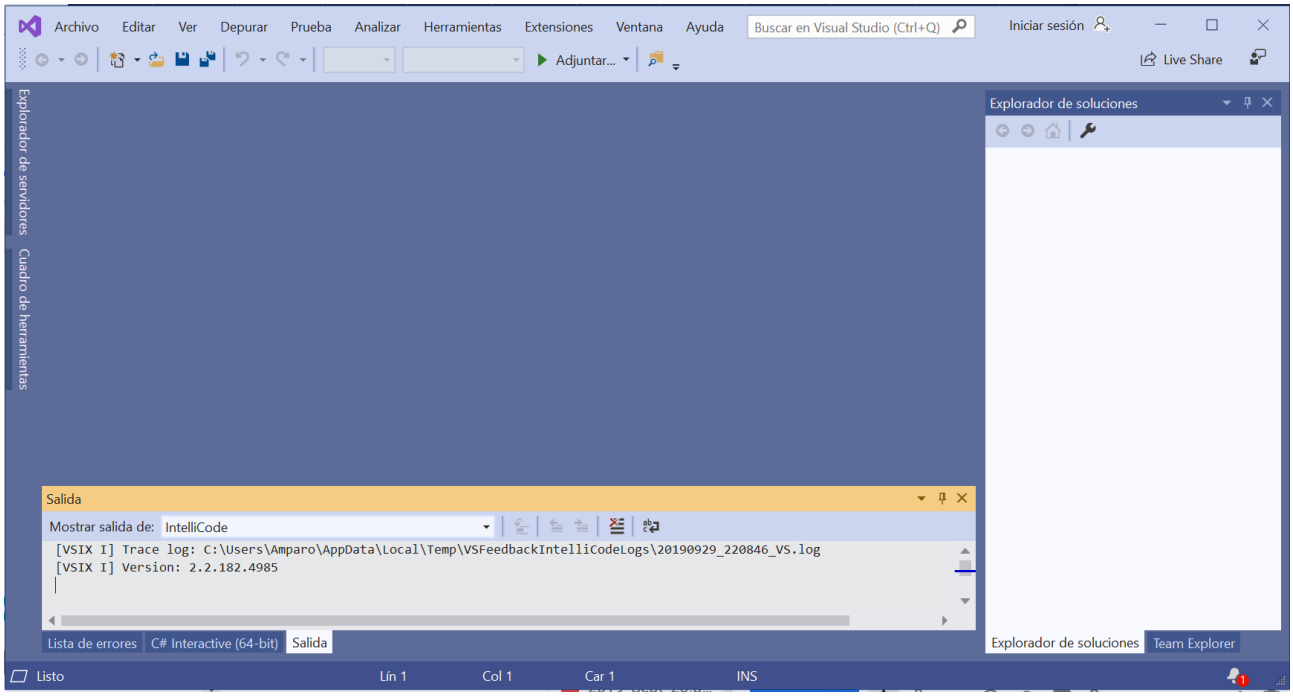
Nota: El CLR ejecuta el programa traduciéndolo a nuestro Windows. Entonces... si alguien hace una para Linux... ¿nuestro código se ejecutaría en Linux sin tocarlo? Pues sí, y de hecho existe en el proyecto "mono" que comenzó hace unos cuantos años. Lo puedes encontrar en el siguiente enlace: http://www.mono-project.com/

2. Primera aplicación con VB.NET

Vamos rápidamente a poner en marcha nuestro primer proyecto para ir conociendo el entorno de desarrollo integrado que llamaremos a partir de ahora IDE y que nos proporcionará ayuda ilimitada para nuestros programas. Nos vamos al menú inicio y ejecutamos el programa "Visual Studio 2019". La versión que instalamos en clase es la Enterprise:



Nada más abrir la aplicación nos aparece la pantalla de inicio que nos da distintas opciones. Entre ellas nos muestra una serie de atajos para abrir un proyecto reciente, crear uno nuevo o “Continuar sin código”. Si elegimos esta última opción nos muestra el entorno sin ningún código.

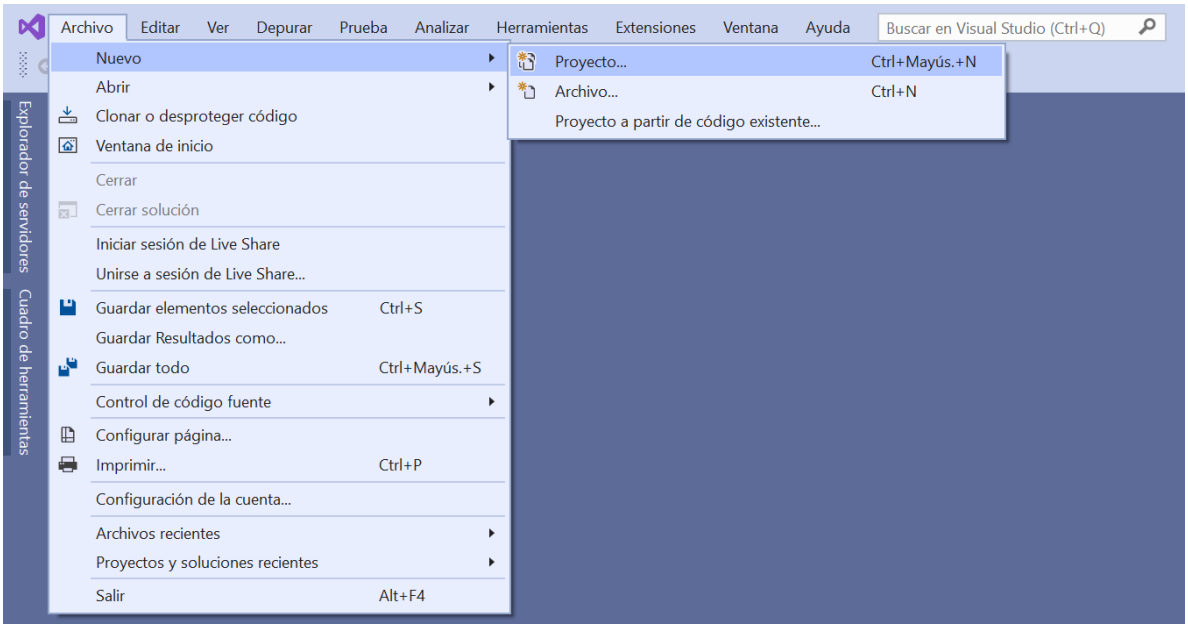


Visual Studio está constituido por tres tipos de elementos que se reparten en su zona de representación:

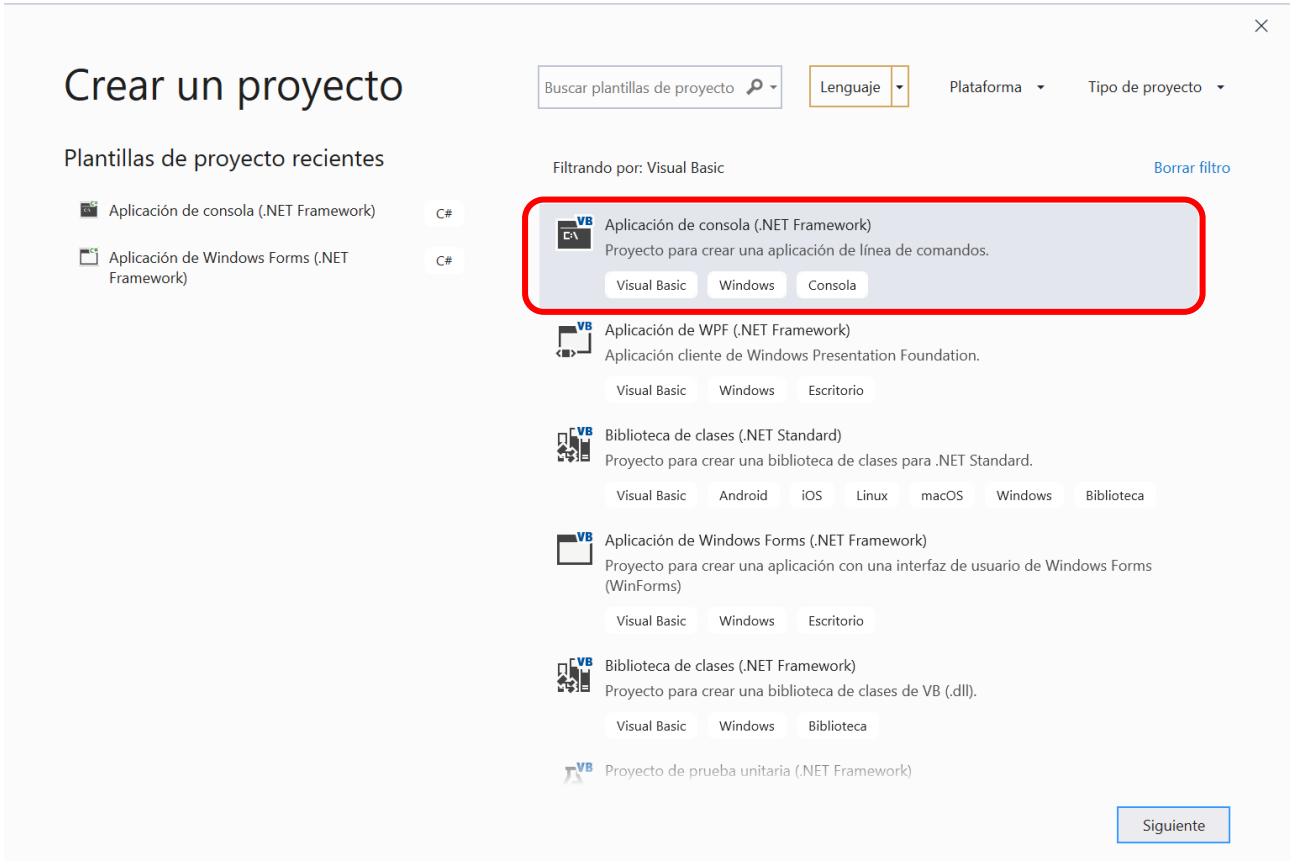
- La parte superior de la ventana principal está compuesta por una barra de menús y varias barras de herramientas.
- La parte central de la ventana principal contiene la zona de trabajo en la que es posible visualizar, entre otros, uno o varios diseñadores visuales así como ventanas de edición de código.
- Varias ventanas secundarias representan las distintas herramientas a disposición del desarrollador.

2.1. Primera aplicación de consola.

Sigamos con nuestro primer programa... para esto desde el menú de ARCHIVO pulsamos en la opción "Nuevo” y luego elegiremos “Proyecto":



Nos mostrará los diferentes tipos de proyectos que se pueden crear:

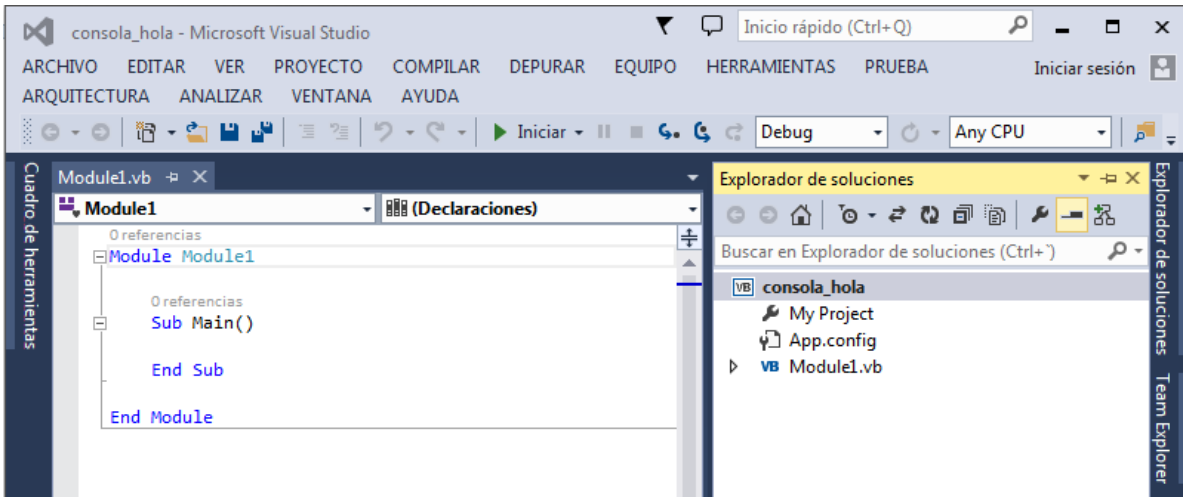


En la parte superior tenemos varios lenguajes a nuestra disposición pero nos centraremos en "Visual Basic". Tenemos los distintos proyectos disponibles. En este caso vamos a crear una "aplicación de consola". Le indicamos un nombre y la ubicación.

Vamos a crear una aplicación de consola. Sigue los pasos y luego veremos el resultado. Seleccionamos el icono que hemos marcado y le indicamos un nombre al proyecto para luego poder cargarlo o identificarlo.

El objetivo de este ejemplo es crear una básica aplicación de consola que se ejecutará en una ventana MS-DOS en lugar del habitual Windows.

Si nos fijamos en la pantalla central y en la derecha podemos ver varias cosas:



Por una parte a la derecha vemos en el explorador de soluciones que ha creado un fichero llamado Module1.vb, con el código necesario para empezar a escribir. Vemos también que ha creado un "procedimiento" **Sub Main**, que se utilizará como punto de entrada de nuestro ejecutable, también ha creado una "definición" llamada **Module Module1** con su respectivo **End Module**, que indica dónde termina la definición del módulo.

Un proyecto de consola se compone de un fichero "Module1.vb" y unas etiquetas "Module Module1" y "End Module". Dentro de estas etiquetas tenemos otras que pone "Sub Main" y "End Sub", que es donde pondremos las instrucciones que queremos que ejecute. Más adelante conoceremos el porqué de estos módulos y nombres...

Nota: Estamos creando una aplicación tipo consola, es decir, no se creará ninguna ventana gráfica, sino que el ejecutable que vamos a crear funciona desde una ventana de consola.

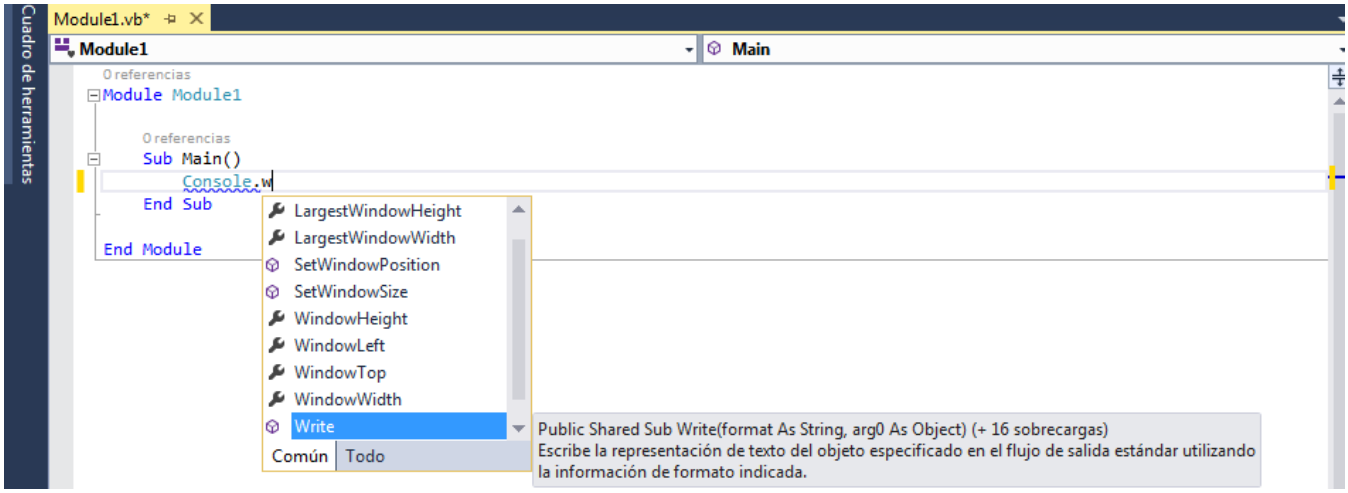
Lo que queremos hacer en este ejemplo es un muy sencillo programa que diga "Hola mundo .NET". Para mostrar un texto en la "consola" usaremos una función, método o instrucción,... como veremos más tarde. Todo esto es posible gracias a los assemblies o a las clases incluidas en el .NET Framework. De momento sigamos este ejemplo...

La instrucción que vamos a utilizar es **Console.Write** y se usa de la siguiente forma:

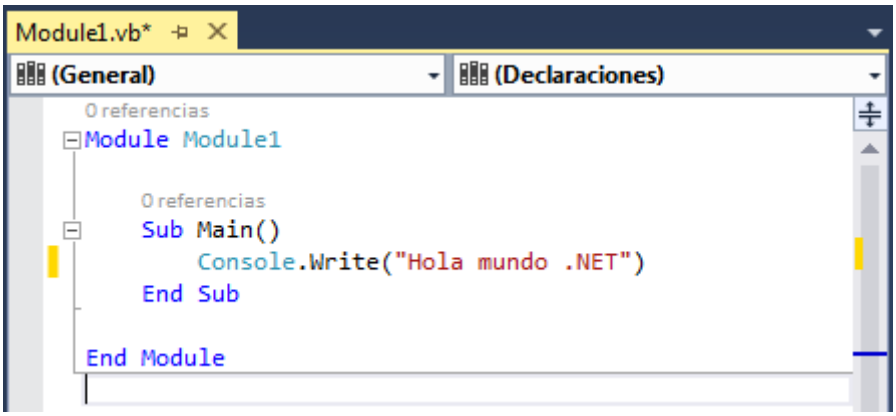
Console.Write("Hola mundo .NET")

Incluiremos dentro de paréntesis lo que queremos que se muestre en la consola, en este caso queremos mostrar un texto, el cual hay que incluirlo dentro de comillas dobles.

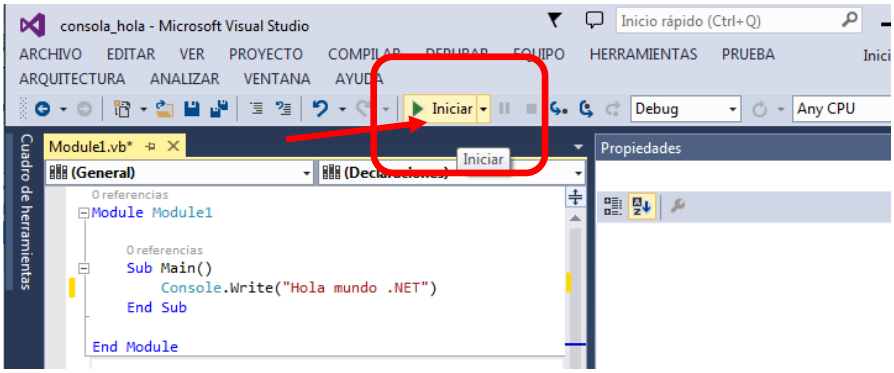
Ponemos esta instrucción entre el *Sub Main()* y el End Sub. Comprobaremos que cuando escribimos la palabra “Console” y el punto, se mostrarán las funciones que “Console” puede realizar, así como una pequeña ayuda, en modo de ToolTip.



Y terminamos de escribir la sentencia:



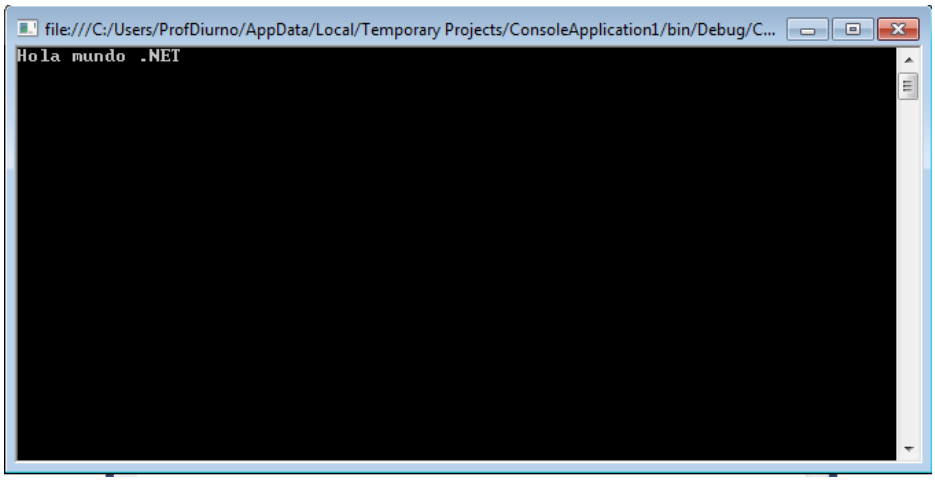
Bien, ya tenemos todo lo que necesitamos. Ahora tendremos que indicarle al "Entorno Integrado" (IDE) que compile el proyecto y lo ejecute. Después de este proceso de compilación, nos mostrará el texto en una consola de comando. Una vez escrito pulsamos en el botón que se muestra en esta pantalla:



Antes de pulsar para ejecutar la aplicación vamos a realizar un pequeño añadido. Si lo ejecutamos no veremos nada en pantalla porque se ejecutará muy rápidamente. Para ver el programa correctamente añadiremos debajo una instrucción para que haga una pausa y podamos comprobar que el programa funciona:

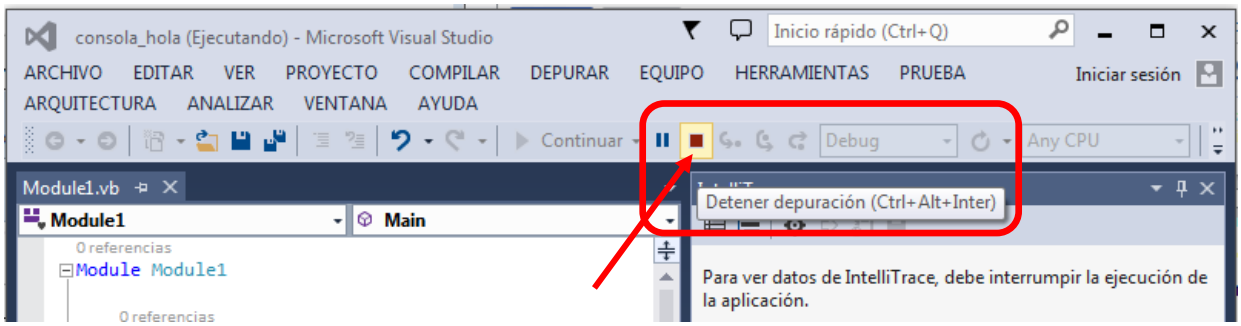
```
Module Module1

    Sub Main()
        Console.Write("Hola mundo .NET")
        Console.ReadLine()
    End Sub

End Module
```

Ahora sí, ejecutamos la aplicación y después de unos segundos nos aparecerá una pantalla de consola con:

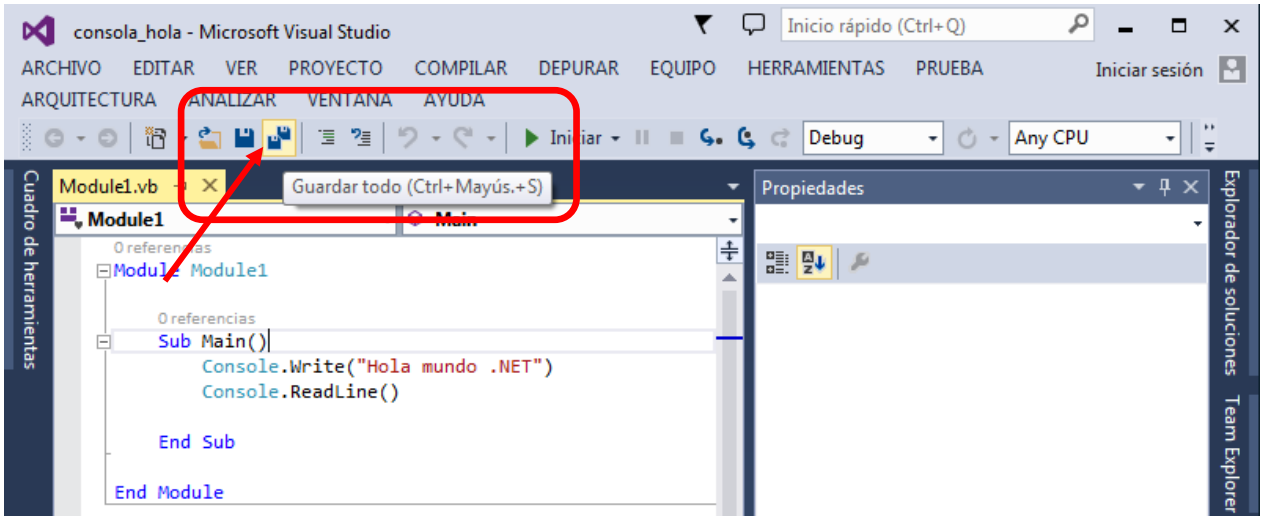


Al ejecutarse hemos visto una pantalla en modo texto con el mensaje que hemos escrito de "Hola Mundo .NET". Lo que hemos hecho ha sido ejecutar el programa desde nuestro entorno de desarrollo de una forma controlada o en el llamado “modo de depuración”. Para hacer el programa final tendríamos que compilarlo para generar el programa ejecutable, lógicamente con extensión .exe.

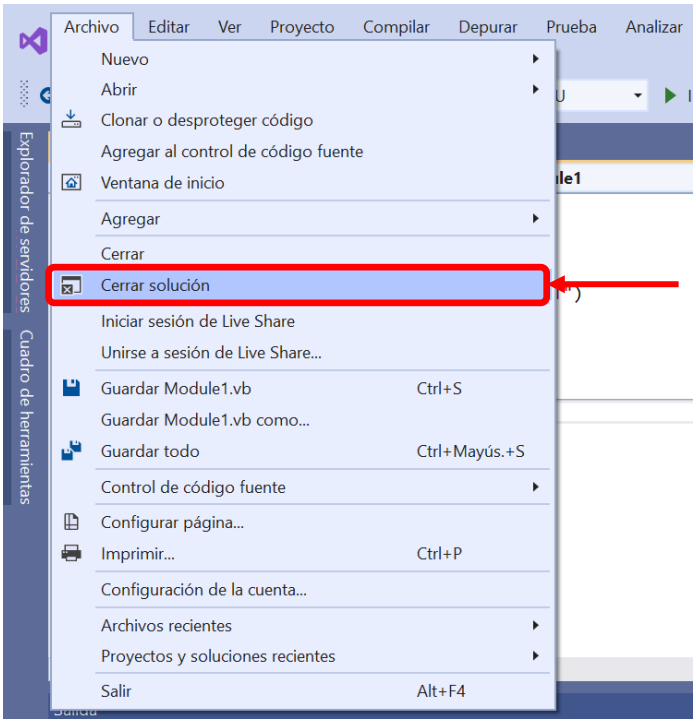
Con esto tenemos terminada nuestra primera aplicación de consola realizada en .NET. Ahora pulsaremos en el siguiente botón para detener la ejecución:



De esta forma volveremos a tener el control del IDE. Ahora guardaremos nuestro proyecto pulsando en el botón de Guardar.

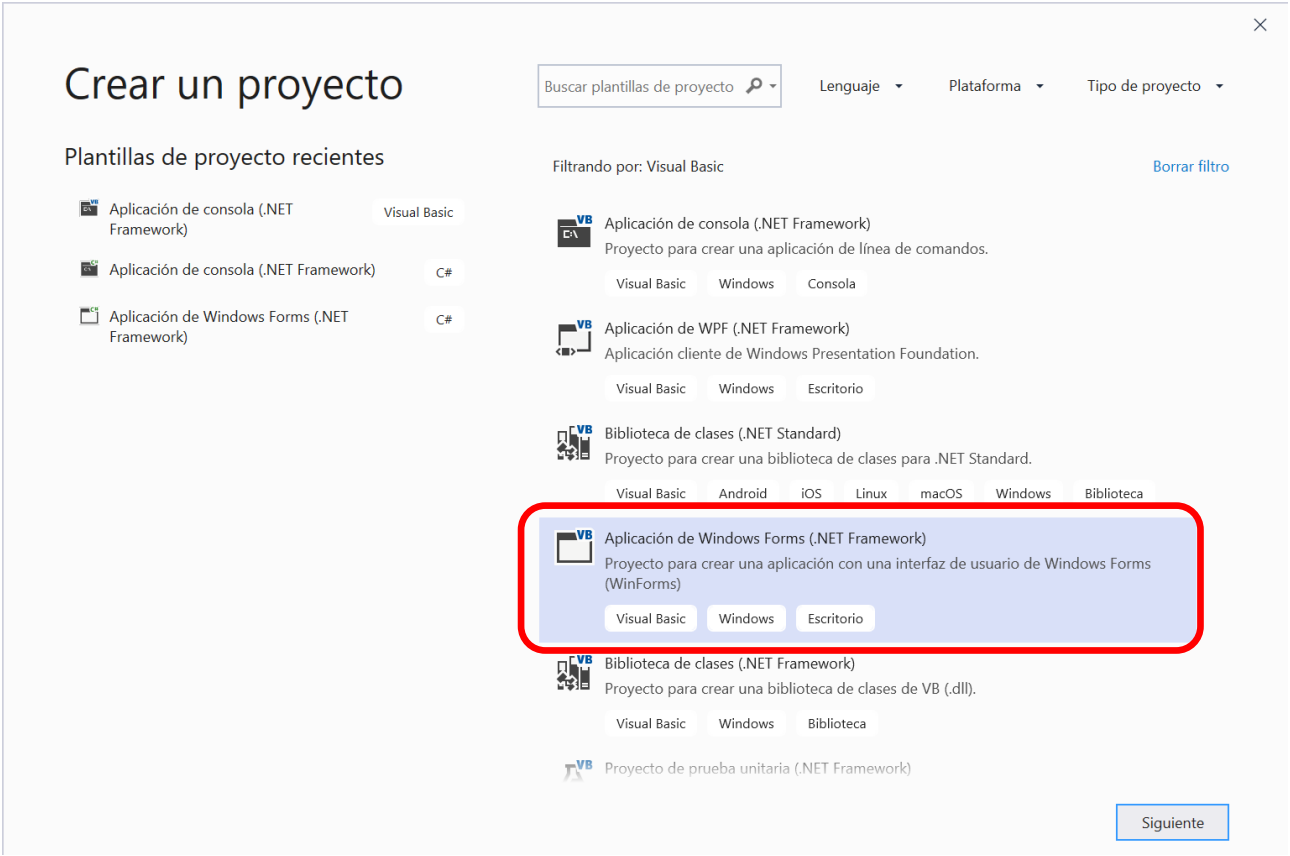


Finalmente cerramos el proyecto desde la opción de "Archivo" y luego "Cerrar solución":

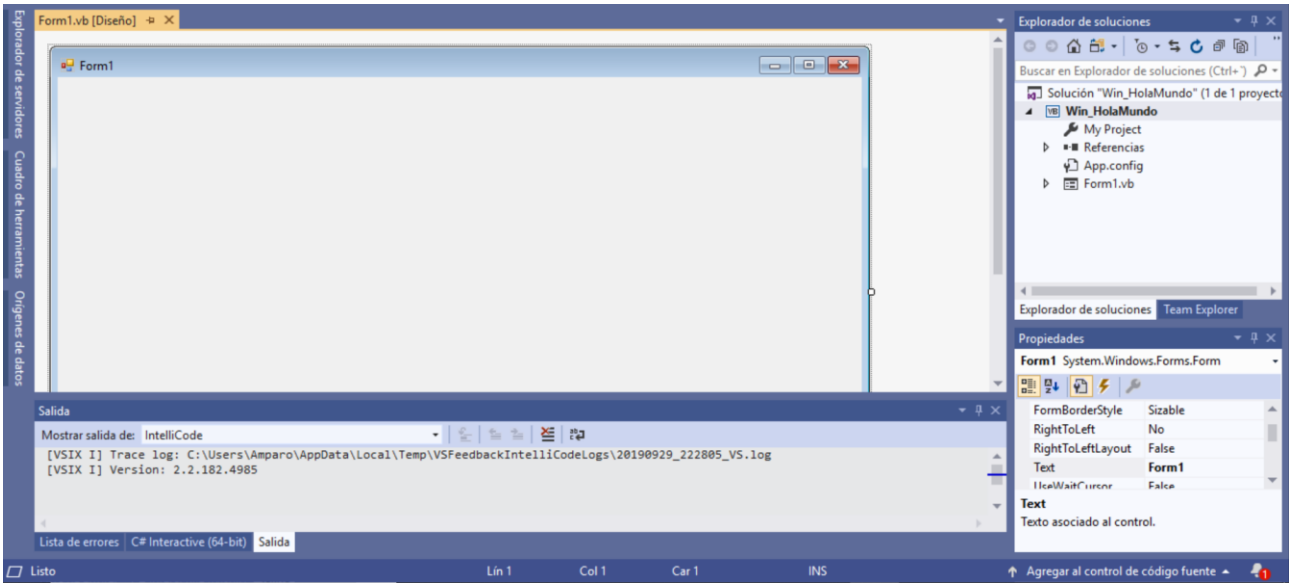


2.2. Primera aplicación de Windows.

Ahora vamos a elegir crear una aplicación Windows. Volvemos a indicar que queremos un "Nuevo proyecto", indicamos que sea de Windows Forms y le ponemos el nombre de Win_HolaMundo:

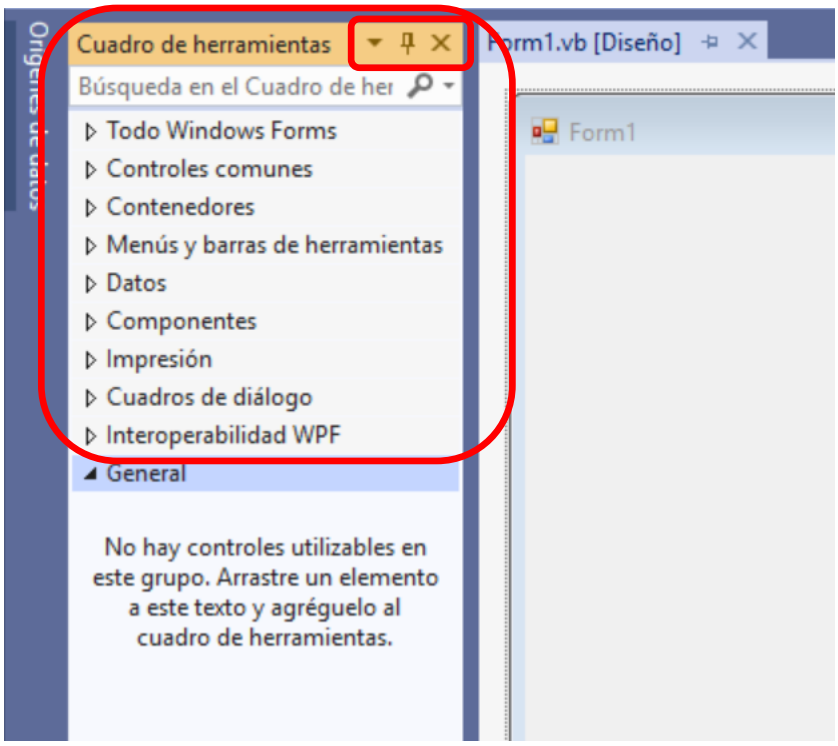


El aspecto del IDE ahora será distinto ya que ahora partimos de un formulario Windows:

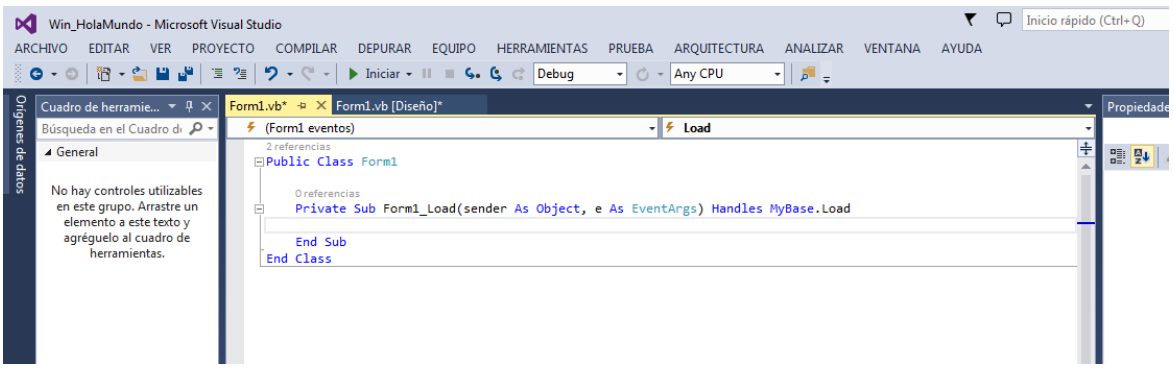


En la parte de la izquierda tenemos tres solapas verticales: “Explorador de servidores”, "Cuadro de herramientas" y "Orígenes de datos".

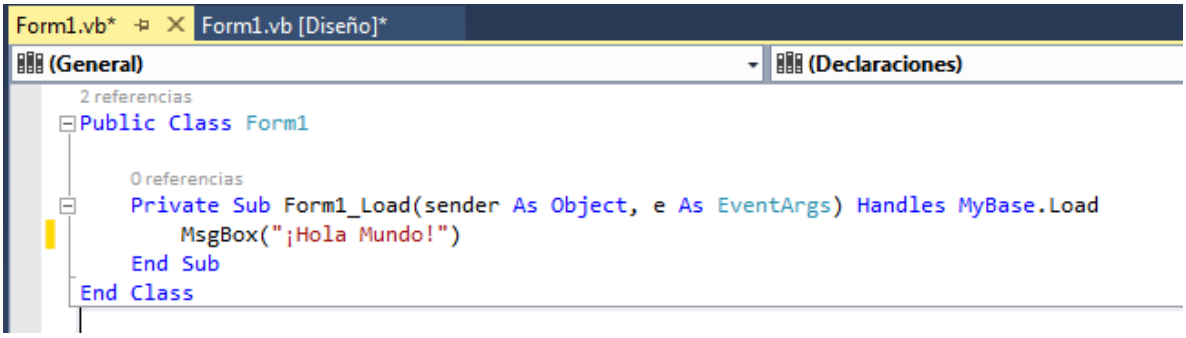
El “Cuadro de herramientas” contiene los controles de Windows que podremos utilizar en nuestras aplicaciones Windows. Haremos clic en esa pestaña y luego en la "chincheta" que aparece para que se quede fija en la pantalla:



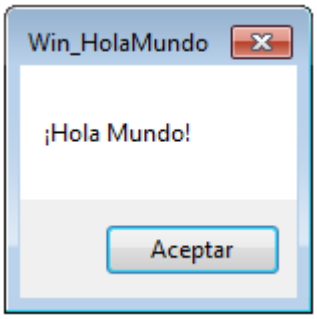
En nuestro entorno de desarrollo para la aplicación Windows, tenemos una serie de controles a la izquierda, en el centro un formulario para nuestra aplicación y a la derecha dos ventanas: el explorador de soluciones y la ventana de propiedades. Luego veremos con detalle todas estas partes, sigamos con nuestro ejemplo. Vamos a hacer doble clic en el centro del formulario que tiene de título "Form1" en el centro de la ventana. Nos mostrará una ventana con este aspecto:



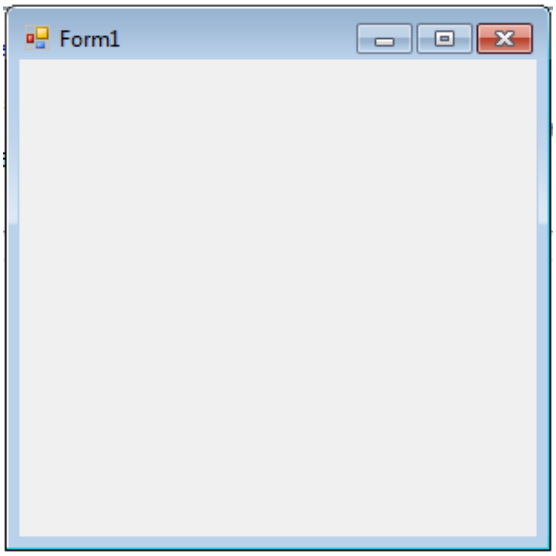
Si nos fijamos, veremos el cursor activo en el editor. Nos está indicando que es ahí donde debemos introducir el código cuando se produzca el evento "Load" de "Form1", es decir, cuando se cargue el formulario. Vamos a escribir ahora este código:



Ejecutaremos la aplicación como hicimos antes y nos mostrará el mensaje que le hemos indicado:



Al pulsar en "Aceptar" nos mostrará el formulario vacío, ya que no hemos puesto nada más que ese mensaje que nos ha mostrado.



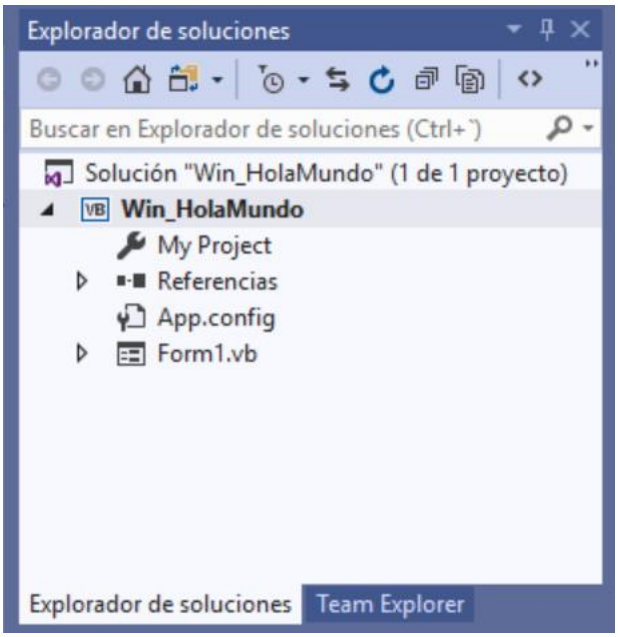
Paramos la ejecución con el botón que vimos antes y grabamos el proyecto.

Ya tenemos nuestra primera aplicación Windows, ahora veremos los ficheros que ha creado al generar el proyecto.

2.3. Ficheros del proyecto.

En Visual Basic .NET sólo existe un tipo de fichero de código que tiene la extensión .vb. En este tipo de fichero pueden coexistir distintos tipos de elementos, por ejemplo: un módulo de clase, un formulario, un módulo de código, un control, etc.

En nuestra aplicación anterior, un programa de Windows con un formulario, podíamos ver los ficheros de los que se compone la solución en la parte derecha:



Como ves, hay uno con extensión ".vb" que es el que contiene la definición y código del formulario, un fichero "My Project" con la definición del proyecto completo y un fichero "App.config" con la configuración del proyecto.

3. Más conceptos de Visual Basic .NET.

Sigamos con más conceptos básicos sobre .NET. Una aplicación .NET se compone de uno o más ensamblados, cada uno de los cuales estará formado por uno o más archivos. La aplicación, por tanto, puede ser tan simple como un sólo archivo .EXE o un conjunto de varios archivos de código y elementos externos.

Las aplicaciones .NET se ejecutan en lo que se llama "dominio de la aplicación" que es como un espacio donde se ejecuta nuestra aplicación y está aislada del resto de las aplicaciones que se están ejecutando en Windows. Perfecto para la seguridad y para que un fallo de aplicación no afecte a los demás...

Antes de continuar con otro sencillo ejemplo, vamos a conocer un poco sobre el entorno de desarrollo de Visual Basic .NET para que podamos configurar algunos aspectos. Por ejemplo,

para indicar cómo se comportará el compilador e intérprete sobre el código que escribamos o para configurar los ensamblados (assemblies) que utilizaremos en nuestras aplicaciones. (Recuerda que Visual Basic .NET utiliza una serie de bibliotecas (de clases) con las funciones que necesitemos en cada momento...)

VB.NET dispone entonces de distintas bibliotecas con funciones. Por ejemplo, para poder mostrar un texto en la consola necesitamos tener disponible la librería en la cual está declarada la clase **Console** y así podremos acceder a las funciones que dicha clase pone a nuestra disposición, (por ejemplo Write o Read). En este caso la librería o biblioteca en la que está la clase Console es **System**. System realmente es un Namespace o espacio de nombres, no es una librería o assembly.

Así que añadiremos distintas bibliotecas a nuestro proyecto para que éste pueda trabajar con bases de datos, gráficos, ...

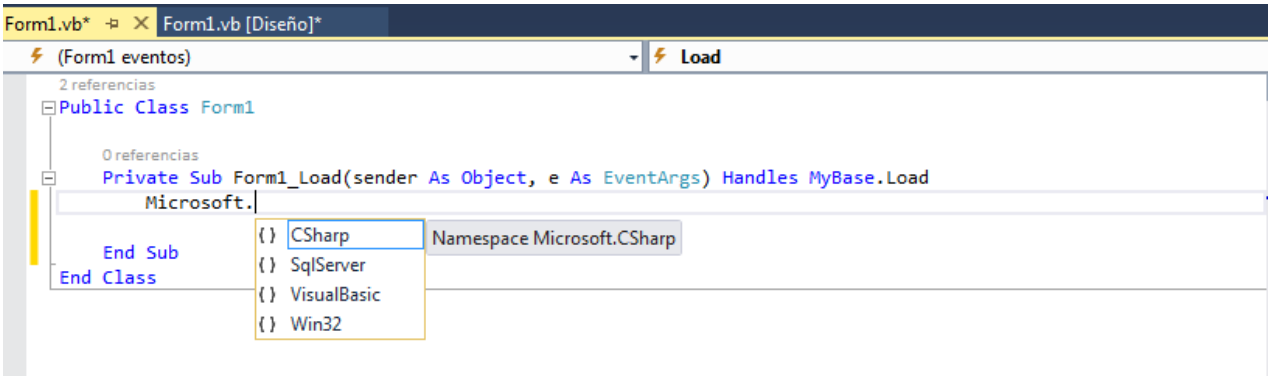
¿Qué es un Namespace (o espacio de nombres)?

Un **Namespace o espacio de nombres** es una forma de agrupar clases, funciones, tipos de datos, etc. que están relacionadas entre sí.

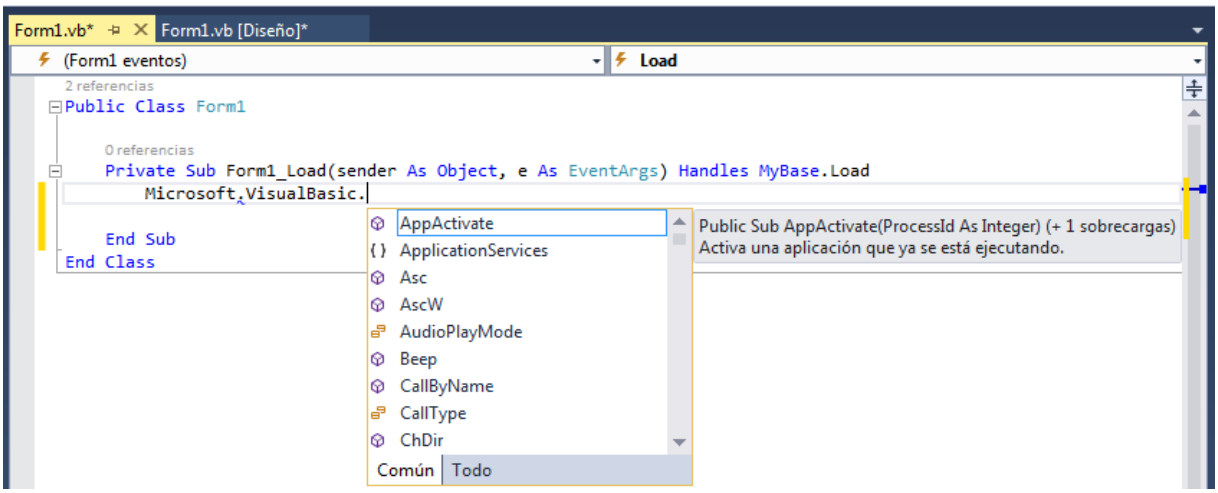
Por ejemplo, entre los Namespaces que podemos encontrar en el .NET Framework encontramos uno con funciones relacionadas con Visual Basic: **Microsoft.VisualBasic**.

Si nos fijamos, Microsoft y VisualBasic están separados por un punto, esto significa que **Microsoft** a su vez es un Namespace que contiene otros "espacios de nombres", tales como el mencionado **VisualBasic**, **CSharp** y **Win32** con el cual podemos acceder a eventos o manipular el registro del sistema...

Para saber que es lo que contiene un Namespace, simplemente escribe el nombre con un punto y nos mostrará una lista desplegable con los miembros que pertenecen a dicho espacio de nombres. Hagamos la prueba y en el código escribimos la palabra "Microsoft." (con el punto al final). Veremos que se despliegan los posibles Namespaces disponibles y si seguimos y escribimos, o seleccionamos VisualBasic con un punto al final, veremos otra vez los Namespaces de esta otra clase llamada VisualBasic.



Y si escribimos un punto después tendremos acceso a todos los comandos disponibles en este espacio de nombres:



Así que podemos utilizar como una especie de grupos de instrucciones y otros componentes añadiendo o utilizando "namespaces" a nuestros programas. De esta forma mantenemos los recursos de las aplicaciones en diferentes sitios reduciendo los conflictos.

Por regla general se agrupan en un Namespace funciones o clases que estén relacionadas entre sí. De esta forma, será más fácil saber que estamos trabajando con funciones de la misma familia.

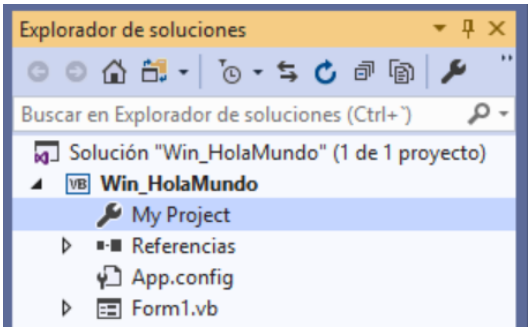
Hay espacios de nombres específicos, por ejemplo, para los gráficos y otros para trabajar con bases de datos, así los tenemos organizados de una forma más cómoda para localizarlo y referirnos a ellos en el código.

Pero de esto no debemos preocuparnos, ya que el IDE de Visual Studio .NET se encarga de "saber" en qué assembly está el Namespace que necesitamos.

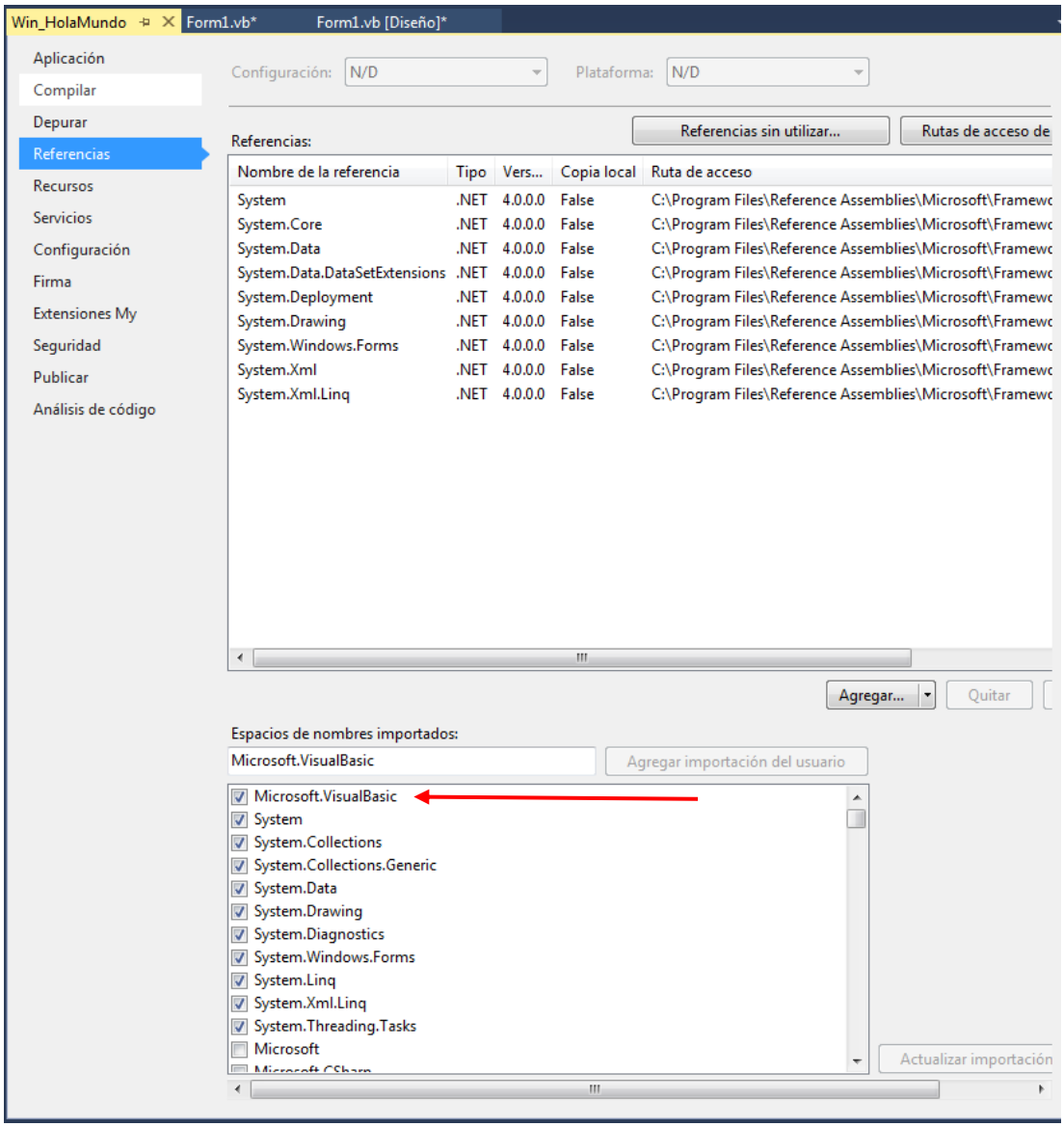
Los Namespaces más importantes, de los que luego heredan otros o pertenecen los demás son:

- System. Contiene clases fundamentales y clases base que definen los valores y tipos de datos de referencia, eventos y controladores de eventos, interfaces, atributos y excepciones de procesamiento comúnmente utilizados.
- Microsoft.VisualBasic. Contiene clases que admiten la compilación y generación de código mediante el lenguaje Basic .NET.

Ahora bien, si para poder utilizar las instrucciones de Visual Basic debo importar el espacio de nombres correspondiente... ¿porque me han funcionado los ejemplos anteriores? Muy sencillo, porque por defecto ya se encuentran incorporados a los proyectos los espacios de nombres más comunes. Hacemos doble clic en *My Project* en el explorador de soluciones:



Nos mostrará las propiedades del proyecto y, si nos fijamos en la opción "Referencias", veremos la siguiente información:



En la parte inferior, tenemos por defecto todos estos espacios de nombres importados a nuestro proyecto. Por lo tanto, no hace falta que los incluyamos con instrucciones especiales. Sin embargo, otros espacios de nombres, como los de acceso a bases de datos no lo están por defecto y más adelante tendremos que "importarlos".

¿Qué es un assembly (o ensamblado)?

> "Los ensamblados componen la unidad fundamental de implementación, control de versiones, reutilización, ámbito de activación y permisos de seguridad en una aplicación basada en .NET. Los ensamblados adoptan la forma de un archivo ejecutable (.exe) o un archivo de biblioteca de vínculos dinámicos (.dll), y constituyen unidades de creación de .NET Framework. Proporcionan a Common Language Runtime la información que necesita para estar al corriente de las implementaciones de tipos. Un ensamblado puede entenderse como una colección de tipos y recursos que forman una unidad lógica de funcionalidad y que se generan para trabajar conjuntamente"

Para que nos entendamos, podríamos decir que un assembly es una librería dinámica (DLL) o programa ejecutable en la cual pueden existir distintos espacios de nombres. Aunque esto es simplificar mucho, por ahora nos vale.

Un ensamblado o assembly puede estar formado por varios ficheros DLLs y EXEs, pero lo más importante es que todos los ensamblados contienen un manifiesto (o manifest). Este manifiesto es una tabla de contenido del Assembly que lo identifica y dice su versión y otros datos. Esta información elimina lo que en otras versiones se llamaba "infierno de las DLL" por los enormes problemas de versiones y conflictos que generaban.

4. Bases de la Programación Orientada a Objetos (POO)

En esta introducción general a .NET debemos comentar algunos conceptos básicos de esta tecnología y que iremos ampliando a lo largo de los siguientes temas. No van a ser explicaciones técnicas sino más bien una toma de contacto de la filosofía .NET. Todo lo que trataremos en .NET se basa en clases y objetos. Es un concepto muy sencillo pero que pudiera desconcertar al principio

Veamos qué son esos conceptos y la base de la programación orientada a objetos

4.1. Las clases

Todo lo que tiene .NET Framework son clases. Una clase no es ni más ni menos que código para crear objetos.

Cuando definimos una clase, realmente estamos definiendo dos cosas diferentes: los datos que dicha clase puede manipular o contener y la forma de acceder a esos datos.

Por ejemplo, si tenemos una clase de tipo Cliente, por un lado tendremos los datos de dicho cliente y por otro la forma de acceder o modificar esos datos. En el primer caso, los datos del Cliente, como por ejemplo el nombre, domicilio etc., estarán representados por una serie de campos o propiedades, mientras que la forma de modificar o acceder a esa información del Cliente se hará por medio de métodos. Esas propiedades o características y las acciones a realizar son las que definen a una clase.

Un coche tiene unas propiedades: color, marca, modelo, ... y unos métodos para trabajar con él: arrancar, frenar, cambiar de marcha. La definición de estas partes es lo que llamamos clase.

4.2. Los Objetos

Por un lado tenemos una clase que es la que define un "algo" con lo que podemos trabajar. Pero para que podamos trabajar con ese "algo", tendremos que poder convertirlo en "algo tangible", es decir, tendremos que tener la posibilidad de que exista. Aquí es cuando entran en juego los objetos, ya que un objeto es una clase que tiene información real. Por fin podemos crear un coche del cual ya tenemos su definición en la clase Coche

Digamos que la clase es la "plantilla" a partir de la cual podemos crear un objeto en la memoria. Por ejemplo, podemos tener varios objetos del tipo Cliente, uno por cada cliente que tengamos en nuestra cartera de clientes, pero la clase sólo será una.

En nuestros formularios: tenemos 10 botones y ... han sido creados a partir de la clase "Botón". Otro ejemplo es que tenemos una clase que se llama "Coches" donde describe qué es y cómo funciona un coche. Pues bien, podemos crear diferentes coches a partir de la clase "Coches", cada uno puede tener sus propias propiedades: color, ... pero funcionan todos igual.

Además existen objetos de más categoría que otros y esta relación se llama jerarquía de objetos. Por ejemplo, un objeto Coche puede tener a su vez varios objetos mas pequeños: motor, carrocería, ...

En .NET tenemos varias clases principales y debajo de ellas todas las instrucciones del lenguaje. Por ejemplo, habrá una clase para los formularios (ventanas de Windows) que a su vez tendrán otras clases dentro: botones, textos, imágenes. Otra clase sería la colección de funciones matemáticas que podemos utilizar. Para hacernos una idea, esta es la jerarquía de objetos para desarrollo en Web: