

INTRODUCCION A LA PROGRAMACION C#

4 - MÓDULO 1 : FUNDAMENTOS DE C#

UNIDAD: 4

MODULO: 1

PRESENTACIÓN: En esta unidad se explicarán las características principales del lenguaje C#.

OBJETIVOS

Que los participantes logren: Comprender los fundamentos del C#. Tener un primer contacto con la IDE Visual Studio. Hacer su primer programa en C#

TEMARIO

Origen y necesidad de un nuevo lenguaje	4
Características de C#.....	5
Escritura de aplicaciones.....	6
Aplicación básica ¡hola mundo!.....	7
Puntos de entrada	9
Compilación con Visual Studio	11
Compilación en línea de comandos.....	14

Origen y necesidad de un nuevo lenguaje

C# (leído en inglés “C Sharp”) es el lenguaje de propósito general diseñado específicamente por Microsoft para su plataforma .NET.

Sus principales creadores son Scott Wiltamuth y Anders Hejlsberg, éste último también conocido por haber sido el diseñador del lenguaje Turbo Pascal y la herramienta RAD Delphi.

Aunque es posible escribir código para la plataforma .NET en muchos otros lenguajes, C# es el único que ha sido diseñado específicamente para ser utilizado en ella, por lo que programarla usando C# es mucho más sencillo e intuitivo que hacerlo con cualquiera de los otros lenguajes ya que C# carece de elementos heredados innecesarios en .NET. Por esta razón, se suele decir que C# es el lenguaje nativo de .NET

La sintaxis y estructuración de C# es muy similar a la C++, ya que la intención de Microsoft con C# es facilitar la migración de códigos escritos en estos lenguajes a C# y facilitar su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son equiparables a los de Visual Basic.

Microsoft ha tenido que desarrollar un nuevo lenguaje que añadiese a las ya probadas virtudes de Java las modificaciones que Microsoft tenía pensado añadirle para mejorarlo aún más y hacerlo un lenguaje orientado al desarrollo de componentes.

En resumen, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo.

Características de C#

- Interoperabilidad. Es decir que puede acceder a código escrito en cualquier otro lenguaje soportado por .NET
- Es similar a Java. Cuando apareció se decía que era: 70% Java, 10% C++, 5% Visual Basic, 15% nuevo.
- Es compilado.
- Es Open Source !!! (Net Core).
- Es multiplataforma (Net Core).
- Es multidispositivo (Xamarin, Unity).
- Es joven y está evolucionando.

Version	Year	Key features introduced
1.0	Jan 2002	
1.2	Oct 2003	Modern, object-oriented, type safe, automatic memory management, versioning control
2.0	Sept 2005	Generics, partial classes, anonymous types, iterators, nullable types, static classes, delegate interface
3.0	Aug 2007	Implicit types, object and collection initializers, auto-implemented properties, extension methods, query and lambda expressions, expression trees, partial methods.
4.0	April 2010	Dynamic binding, named and optional arguments, Generic covariance and Contravariance, Embedded Interop types.
5.0	June 2013	Async methods, Caller Info Attributes
6.0	July 2015	Roslyn (compiler-as-a-service), exception filters, Await in catch/finally block, auto property initializer, string interpolation, nameof operator, dictionary initializer
7.0	2016	Tuples, pattern matching, record types, local functions, Async streams

Escritura de aplicaciones

Con C# podemos escribir aplicaciones de los siguientes tipos

- Aplicaciones de línea de comandos
- Aplicaciones Web
- Aplicaciones de Escritorio
- Servicios
- Mobile
- Juegos

Aplicación básica ¡hola mundo!

Básicamente una aplicación en C# puede verse como un conjunto de uno o más ficheros de código fuente con las instrucciones necesarias para que la aplicación funcione como se desea y que son pasados al compilador para que genere un ejecutable.

Cada uno de estos ficheros no es más que un fichero de texto plano escrito usando caracteres Unicode y siguiendo la sintaxis propia de C#.

Como primer contacto con el lenguaje, nada mejor que el típico programa de iniciación “¡Hola Mundo!” que lo único que hace al ejecutarse es mostrar por pantalla el mensaje ¡Hola Mundo!

Su código es:

```
1: class HolaMundo
2: {
3:     static void Main()
4:     {
5:         System.Console.WriteLine("¡Hola Mundo!");
6:     }
7: }
```

Todo el código escrito en C# se ha de escribir dentro de una definición de clase, y lo que en la línea 1: se dice es que se va a definir una clase (class) de nombre HolaMundo, cuya definición estará comprendida entre la llave de apertura de la línea 2: y su correspondiente llave de cierre en la línea 7.

Dentro de la definición de la clase (línea 3:) se define un método de nombre Main cuyo código es el indicado entre la llave de apertura de la línea 4: y su respectiva llave de cierre (línea 6:)

Un método no es más que un conjunto de instrucciones a las que se les asocia un nombre, de modo que para posteriormente ejecutarlas baste referenciarlas por su nombre en vez de tener que re escribirlas.

La partícula que antecede al nombre del método indica cuál es el tipo de valor que se devuelve tras la ejecución del método, y en este caso es void que significa que no se devuelve nada.

Por su parte, los paréntesis que se colocan tras el nombre del método indican cuáles son los parámetros éste toma, y como en este caso están vacíos ello significa que el método no toma parámetros.

Los parámetros de un método permiten variar el resultado de su ejecución según los valores que se les dé en cada llamada.

La palabra `static` que antecede a la declaración del tipo de valor devuelto es un modificador del significado de la declaración de método que indica que el método está asociado a la clase dentro de la que se define y no a los objetos que se creen a partir de ella.

`Main()` es lo que se denomina el punto de entrada de la aplicación, que no es más que el método por el que comenzará su ejecución. Necesita del modificador `static` para evitar que para llamarlo haya que crear algún objeto de la clase donde se haya definido.

Finalmente, la línea 5: contiene la instrucción con el código a ejecutar, que lo que se hace es solicitar la ejecución del método `WriteLine()` de la clase `Console` definida en el espacio de nombres `System` pasándole como parámetro la cadena de texto con el contenido ¡Hola Mundo!

Nótese que las cadenas de textos son secuencias de caracteres delimitadas por comillas dobles aunque dichas comillas no forman parte de la cadena.

Por su parte, un espacio de nombres puede considerarse que es algo similar para las clases a lo que un directorio es para los ficheros; es decir, es una forma de agruparlas.

El método `WriteLine()` se usará muy a menudo en los próximos temas, por lo que es conveniente señalar ahora que una forma de llamarlo que se utilizará en repetidas ocasiones consiste en pasarle un número indefinido de otros parámetros de cualquier tipo e incluir en el primero subcadenas de la forma `{i}` . Con ello se consigue que se muestre por la ventana de consola la cadena que se le pasa como primer parámetro pero sustituyéndole las subcadenas `{i}` por el valor convertido en cadena de texto del parámetro que ocupe la posición `i+2` en la llamada a `WriteLine()` .

Por ejemplo, la siguiente instrucción mostraría Tengo 5 años por pantalla si `x` valiese 5:

```
System.Console.WriteLine("Tengo {0} años", x);
```

Para indicar cómo convertir cada objeto en un cadena de texto basta redefinir su método `ToString()` , aunque esto es algo que no se verá hasta el Tema Clases.

Antes de seguir es importante resaltar que `C#` es sensible a las mayúsculas, lo que significa que no da igual la capitalización con la que se escriban los identificadores. Es decir, no es lo mismo escribir `Console` que `CONSOLE` o `CONSOLE` , y si se hace de alguna de las dos últimas formas el compilador producirá un error debido a que en el espacio de nombres `System` no existe ninguna clase con dichos nombres.

En este sentido, cabe señalar que un error común entre programadores acostumbrados a Java es llamar al punto de entrada `main` en vez de `Main`, lo que provoca un error al compilar ejecutables en tanto que el compilador no detectará ninguna definición de punto de entrada.

Puntos de entrada

Ya se ha dicho que el punto de entrada de una aplicación es un método de nombre `Main` que contendrá el código por donde se ha de iniciar la ejecución de la misma.

Hasta ahora sólo se ha visto una versión de `Main()` que no toma parámetros y tiene como tipo de retorno `void`, pero en realidad todas sus posibles versiones son:

```
static void Main()

static int Main()

static int Main(string[] args)

static void Main(string[] args)
```

Como se ve, hay versiones de `Main()` que devuelven un valor de tipo `int`. Un `int` no es más que un tipo de datos capaz de almacenar valor enteros comprendidos entre $-2.147\ 1\ 483.648$ y $2.147\ 1\ 483.647$, y el número devuelto por `Main()` sería interpretado como código de retorno de la aplicación.

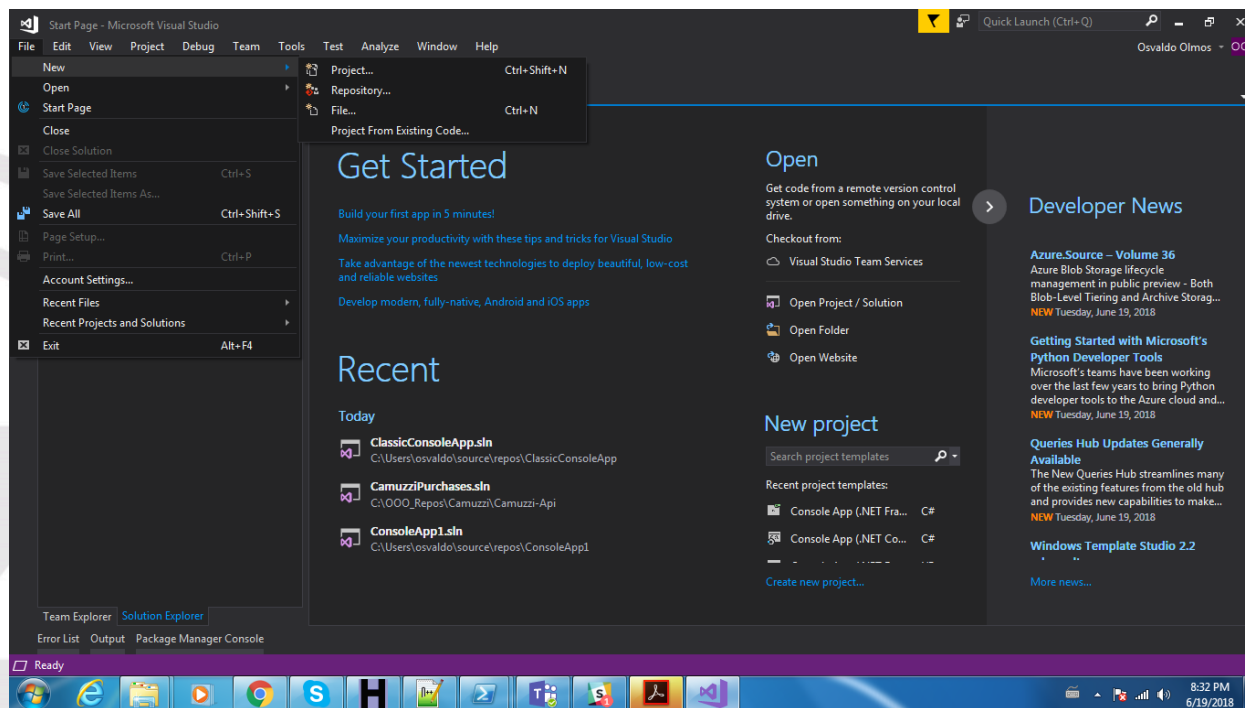
Éste valor suele usarse para indicar si la aplicación a terminado con éxito (generalmente valor 0) o no (valor según la causa de la terminación anormal), y en el Tema Métodos se explicará cómo devolver valores.

También hay versiones de `Main()` que toman un parámetro donde se almacenará la lista de argumentos con los que se llamó a la aplicación, por lo que sólo es útil usar estas versiones del punto de entrada si la aplicación va a utilizar dichos argumentos para algo.

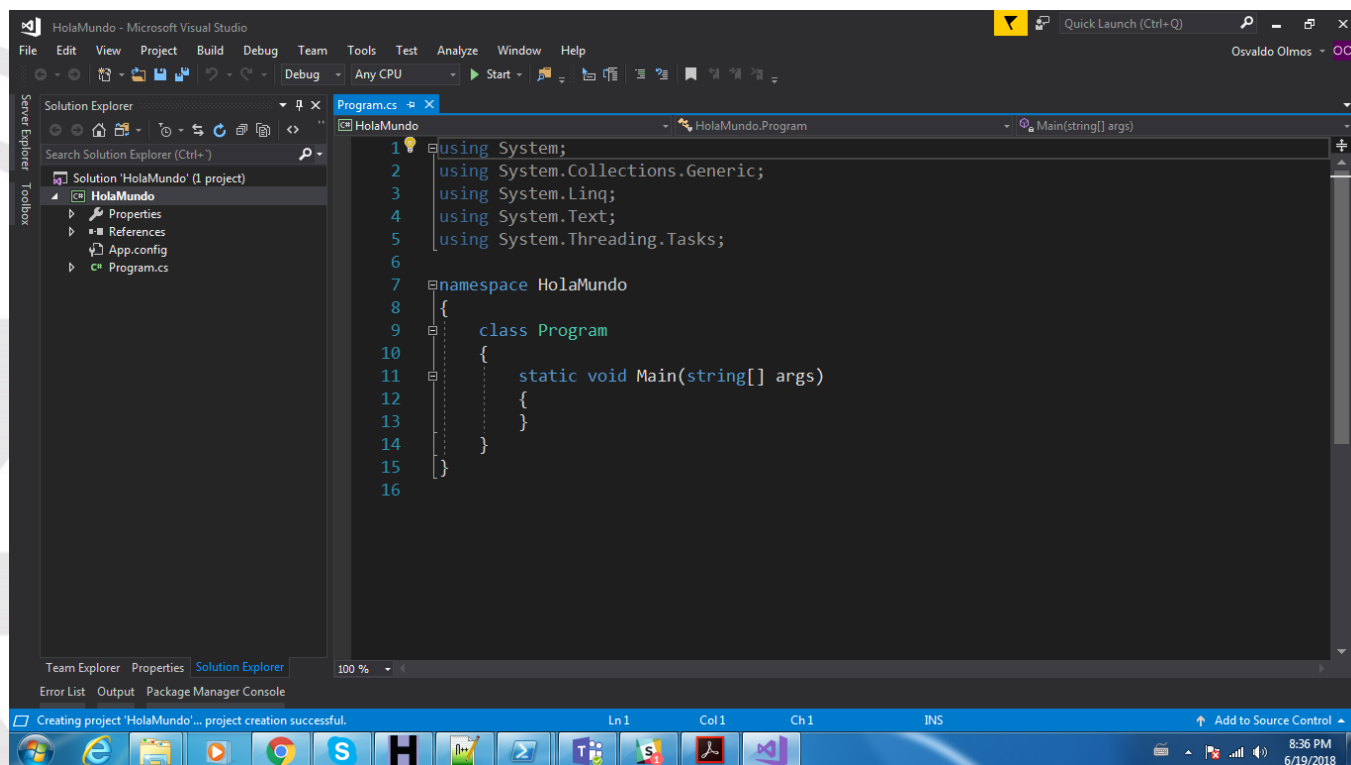
El tipo de este parámetro es `string[]`, lo que significa que es una tabla de cadenas de texto (en el Tema Campos se explicará detenidamente qué son las tablas y las cadenas), y su nombre -que es el que habrá de usarse dentro del código de `Main()` para hacerle referencia- es `args` en el ejemplo, aunque podría dársele cualquier otro.

El hello world en Visual Studio

Abrir el Visual Studio, ir a File ->New -> Project



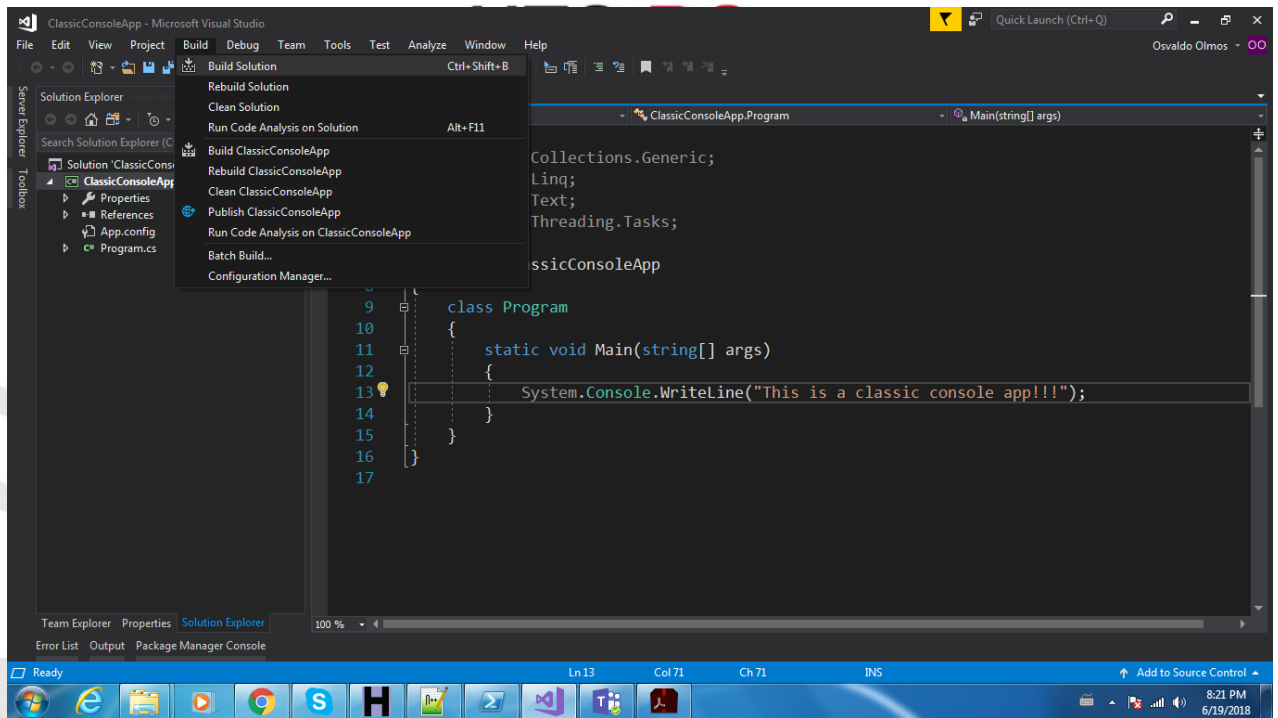
Ubicar la opción Console App (o aplicación de consola), darle un nombre (HolaMundo)



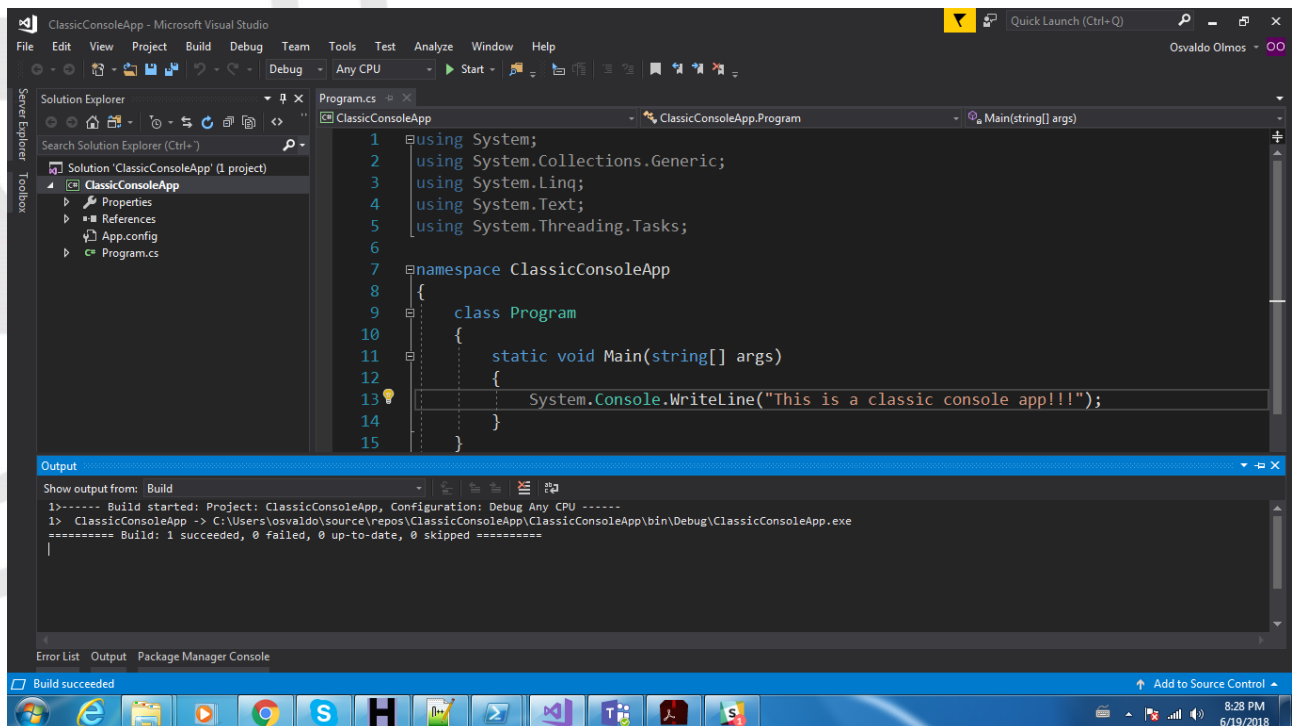
Y vamos a obtener una plantilla inicial con la clase Program.cs y el punto de entrada Main. Estamos listos para volcar el conocimiento de las dos secciones anteriores !!!

Compilación con Visual Studio

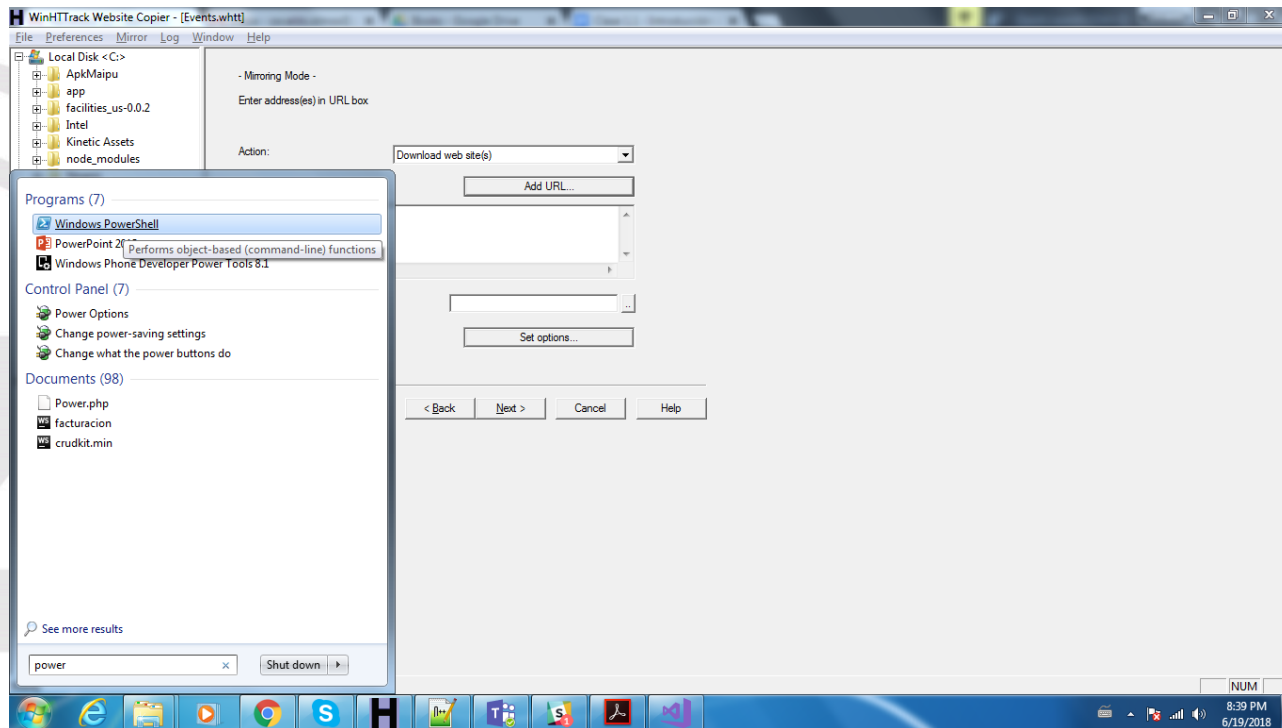
En la barra de menú ir a Build -> Build Solution



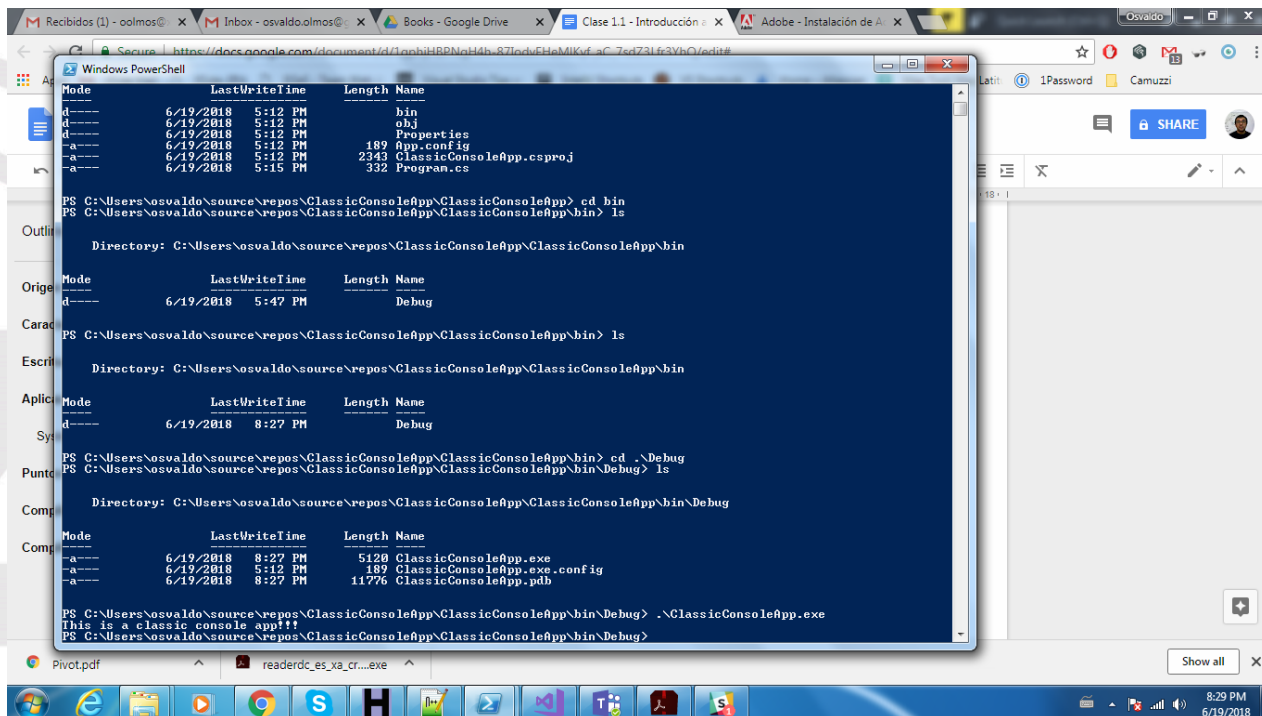
Una vez ejecutado, en la sección output, se puede ver el directorio destino del ejecutable



Para ejecutar el programa, abrir el power shell



Ir al directorio destino y ejecutar el exe



```

PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp> cd bin
PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin> ls

Directory: C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin

Mode                LastWriteTime         Length Name
----                -
d-----        6/19/2018   5:12 PM             bin
d-----        6/19/2018   5:12 PM             obj
d-----        6/19/2018   5:12 PM          Properties
-a-----        6/19/2018   5:12 PM             189 app.config
-a-----        6/19/2018   5:12 PM          2343 ClassicConsoleApp.csproj
-a-----        6/19/2018   5:15 PM             332 Program.cs

PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin> ls

Directory: C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin

Mode                LastWriteTime         Length Name
----                -
d-----        6/19/2018   5:47 PM             Debug

PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin> ls

Directory: C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin

Mode                LastWriteTime         Length Name
----                -
d-----        6/19/2018   8:27 PM             Debug

PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin> cd .\Debug
PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin\Debug> ls

Directory: C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin\Debug

Mode                LastWriteTime         Length Name
----                -
-a-----        6/19/2018   8:27 PM          5120 ClassicConsoleApp.exe
-a-----        6/19/2018   5:12 PM             189 ClassicConsoleApp.exe.config
-a-----        6/19/2018   8:27 PM         11776 ClassicConsoleApp.pdb

PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin\Debug> .\ClassicConsoleApp.exe
This is a classic console app!!!
PS C:\Users\nosvaldo\source\repos\ClassicConsoleApp\ClassicConsoleApp\bin\Debug>
  
```

Compilación en línea de comandos

El compilador del framework .NET incluido en el SDK para Windows es `csc.exe`, y es posible llamarlo desde cualquier directorio en tanto que al instalarlo se añade una referencia al mismo en el path.

Tras la compilación se obtendría un ejecutable llamado `HolaMundo.exe` cuya ejecución produciría la siguiente salida por la ventana de consola:

¡Hola Mundo!

BIBLIOGRAFÍA RECOMENDADA:

<https://docs.microsoft.com/en-us/visualstudio/ide/quickstart-ide-orientation>

