



Práctica 0:

Introducción a Visual Studio y C#

1. Objetivos

- Introducir el uso del entorno de desarrollo de aplicaciones **VisualStudio**.
- Introducir los conceptos de solución, proyecto e implementación de .NET.
- Implementación de nuestro primer programa usando lenguaje C#.
- Implementación de nuestra primera biblioteca usando lenguaje C#.
- Implementación de nuestra primera aplicación web .
- Familiarizarse con el método de entrega de las prácticas de la asignatura.

2. Requisitos técnicos

Sigue los pasos indicados, **respetar el uso de mayúsculas y minúsculas** así como el **nombre de las carpetas, archivos, clases y métodos** que se te indique que has de crear. Requisitos que tiene que cumplir este trabajo práctico para ser evaluado (si no se cumple alguno de los requisitos la calificación será **cero**) son:

- El archivo entregado se llama `hada-p0.zip` (**todo en minúsculas**).
- Al descomprimir el archivo `hada-p0.zip` se crea un directorio de nombre `hada-p0` (**todo en minúsculas**).
- Dentro del directorio `hada-p0` hay un archivo de nombre `hada-p0.sln`.
- Dentro del directorio `hada-p0` hay 4 directorios: `HadaConsola`, `HadaBiblioteca`, `HadaWeb` y `packages`.
- Los espacios de nombres, clases y métodos implementados, así como sus argumentos, se llaman como se indica en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).
- Los mensajes producidos siguen el formato especificado en el enunciado (**respetando en todo caso** el uso de mayúsculas y minúsculas).

Guía de Evaluación

Aunque esta práctica no cuenta para la nota final su **entrega es obligatoria** para comprobar que no hay ningún problema con vuestro usuario para las restantes entregas.

3. Entrega

Se entregará el directorio de la solución `hada-p0`, junto con todo su contenido, comprimido en un fichero llamado `hada-p0.zip`.

La entrega se realizará en <http://pracdlsi.dlsi.ua.es>, no se admite ningún otro método.

Fecha límite: 02/02/2020

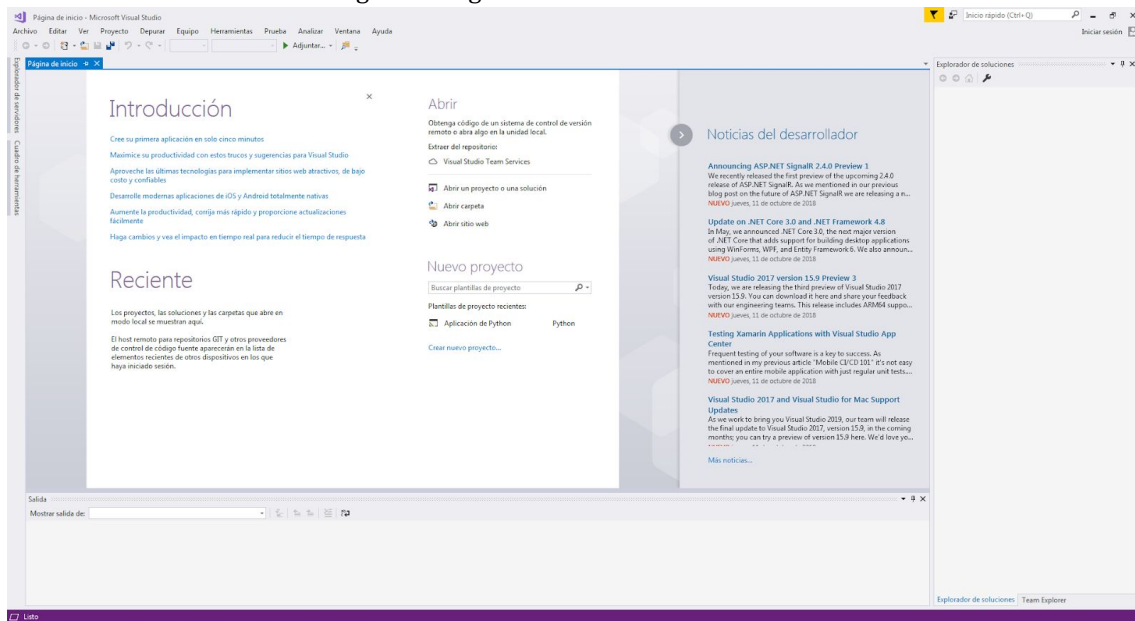
4. Enunciado

4.1. VisualStudio

VisualStudio es el entorno de desarrollo o IDE (Integrated Development Environment) que utilizaremos para la implementación de las prácticas de la asignatura. Por ello, esta primera sesión nos guiará en la creación de uno de los diversos tipos de proyectos que podemos seleccionar mediante este IDE.

El primer paso es abrir el entorno de desarrollo, para lo cual haremos: botón de inicio > Visual Studio 2019 > Visual Studio 2019. Una vez tenemos arrancado el entorno de desarrollo, procederemos a identificar los elementos existentes en éste con la ayuda de la siguiente figura.

Figura 1. Página de inicio de Visual Studio.



Si nos fijamos en la página de inicio (Start Page) observamos que podemos crear nuevos proyectos/sitios web y abrir los ya creados previamente. Asimismo, cuando ya hemos trabajado en varios proyectos en nuestra máquina, tendremos una lista de proyectos/sitios web más recientes. Ten en cuenta que una vez hayas utilizado el entorno durante varias sesiones de prácticas, serás capaz incluso de

modificar este acceso al IDE de forma personalizada, si lo deseas. Para las primeras veces que lo utilices, hemos preparado esta pequeña guía para que puedas acceder a las opciones apropiadas sin demasiado esfuerzo.

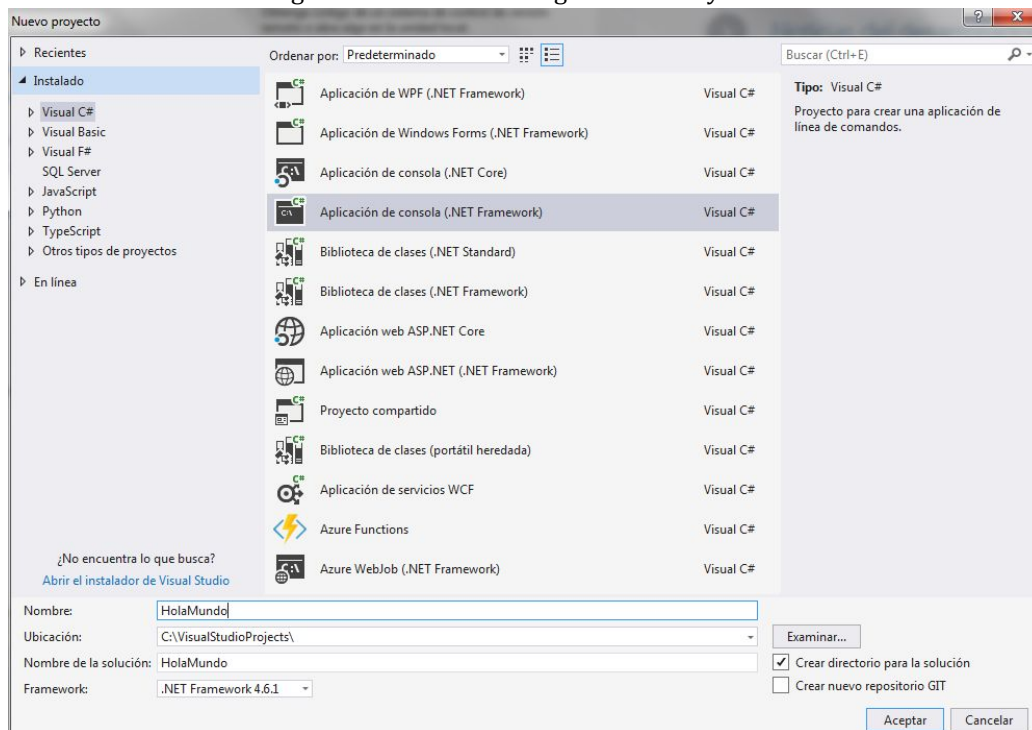
4.2. Proyectos e implementaciones .NET

Un **proyecto** es una colección de archivos de código fuente, recursos, referencias a otros proyectos, etc., que una vez *construido* (el proyecto) puede generar un archivo de código ejecutable o una biblioteca o una aplicación web,.

Para crear un nuevo proyecto, con el VisualStudio abierto, en el menú, elegimos Archivo > Nuevo > Proyecto. El cuadro de diálogo abierto (Figura 2) nos permite crear proyectos que se pueden implementar en varios lenguajes de programación (Visual Basic .NET, C#, Visual C++) hasta la naturaleza del propio proyecto.

Atendiendo al problema que debamos resolver, podemos optar por varios tipos de proyectos: la creación de un programa Windows, o bien de una biblioteca de clases (DLL - Dynamic Link Library), que más tarde podemos necesitar migrar a una aplicación web en función de las necesidades de nuestro proyecto.

Figura 2. Cuadro de diálogo Nuevo Proyecto.



Para cada tipo de proyecto, este IDE nos proporciona una **plantilla de proyecto**. Una plantilla contiene los archivos básicos y la configuración necesaria para un tipo de proyecto concreto, proveyéndonos de una estructura de ficheros que resuelven problemas tipo en el desarrollo software, como por ejemplo: Ficheros ejecutables, dll, ATL, Servicios, páginas asp, proyectos de instalación de sw, etc.



Fíjate en la Figura 2, estamos creando un proyecto para lenguaje *Visual C#*. De todas las posibles, elegimos la plantilla *Aplicación de consola (.NET Framework)*. Esta aplicación recibe el nombre de “HolaMundo”. Un detalle importante es decidir qué **implementación de .NET** utilizar en función de las necesidades del proyecto. Las implementaciones disponibles son:

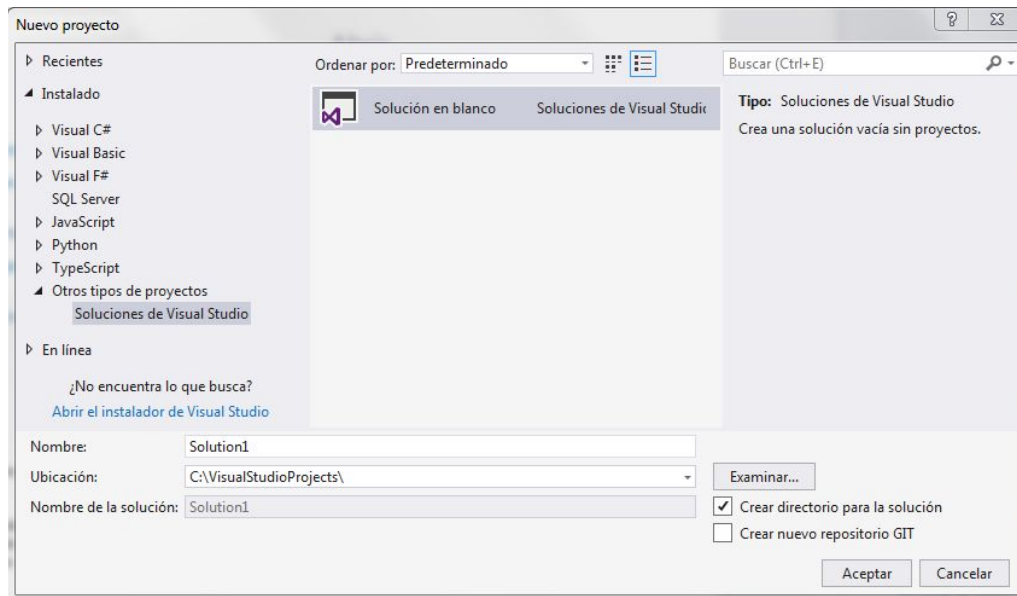
- **.NET Framework:** es la implementación de .NET original pensada para acceder a las API específicas de Windows y crear aplicaciones que se ejecutarán en Windows. Es la más estable para trabajar y para implementar software listo para entornos de producción. Nos permite utilizar tecnologías como ASP.NET Web Forms y ASP.NET Web Pages. *Importante: Será la que utilizemos durante todo el curso.*
- **.NET Core:** es una implementación multiplataforma (Windows, macOS y Linux) diseñada para controlar cargas de trabajo en el servidor y en la nube a escala.
- **Mono:** se usa principalmente cuando se requiere un entorno de ejecución pequeño (Android, Mac, iOS, tvOS y watchOS).
- **Plataforma universal de Windows (UWP):** se usa para implementar aplicaciones Windows táctiles y software para Internet de las cosas. Se ha diseñado para unificar los diferentes tipos de dispositivos de destino, como tabletas, teléfonos o la consola Xbox.

4.3. ¿Qué es una “solución”?

VisualStudio, como otros IDEs, estructura su funcionamiento en torno a las soluciones. Una **solución** no es más que un contenedor que permite organizar uno o más proyectos de código relacionados. Por ejemplo, una solución se compondrá de una biblioteca de clases y un proyecto de web correspondiente.

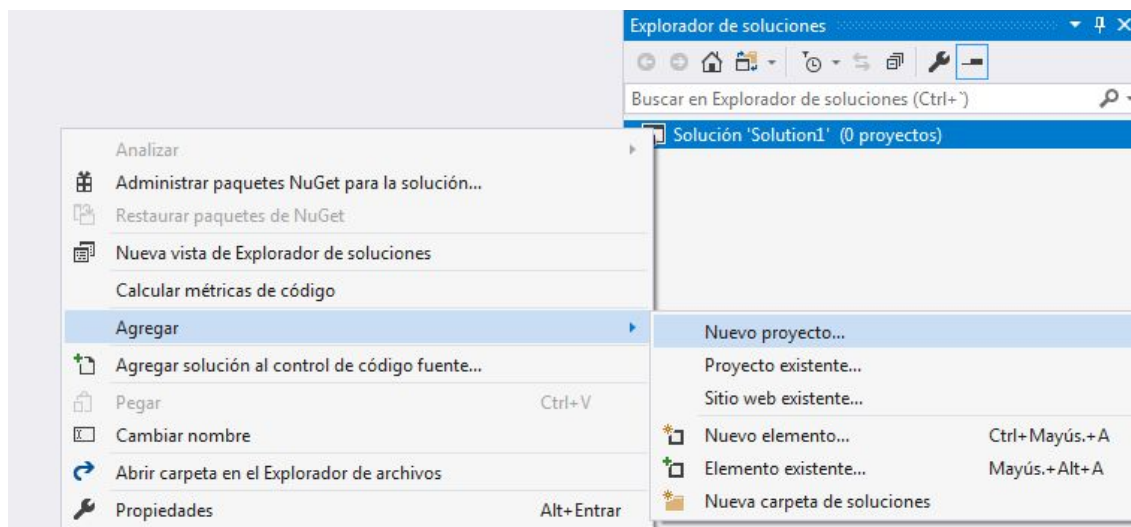
En otras palabras, una solución es uno de los tipos de proyecto que se pueden desarrollar son las “Soluciones de Visual Studio” que CONTIENE otros proyectos. Para crear una solución, con el VisualStudio abierto, desde el menú Archivo > Nuevo > Proyecto, abrimos el cuadro de diálogo de Nuevo Proyecto. Pero en este caso, seleccionamos la plantilla *solución en blanco* (Blank Solution en la Figura 3) la podemos encontrar en la categoría *Otro tipo de proyectos*.

Figura 3. Cuadro de diálogo Nuevo Proyecto, plantilla solución en blanco.



Para agregar un proyecto nuevo a la solución abierta o acabada de crear, selecciónala en el **Explorador de Soluciones** (Figura 4). Después, haz clic con el botón derecho sobre el nombre de la solución, Agregar > Nuevo Proyecto, así abrirás el cuadro de diálogo de *Agregar Nuevo Proyecto* (similar al de las Figura 2 y 3). En la misma solución puedes incluir proyectos de varios tipos, que pueden estar desarrollados en uno o varios lenguajes de programación.

Figura 4 Explorador de soluciones y opciones de agregación de un nuevo proyecto.



4.4. Ejercicio: Mi primer programa

Antes de lanzarnos a crear nuestro programa, necesitamos pensar los pasos necesarios a llevar a cabo:

1. Crear una SOLUCIÓN que actuará a modo de contenedor de programa/s que se llamará **hada-p0**.

2. Dentro de la solución hada-p0, crear un PROYECTO que desarrollaremos en lenguaje C# mediante la utilización de la PLANTILLA de proyecto que se denomina "Aplicación de Consola" utilizando la implementación *.NET FRAMEWORK*, que se llamará **HadaConsola**. Este proyecto aloja la lógica de nuestra aplicación.
3. Unos leves conocimientos de C# para implementar el famoso Hola Mundo.

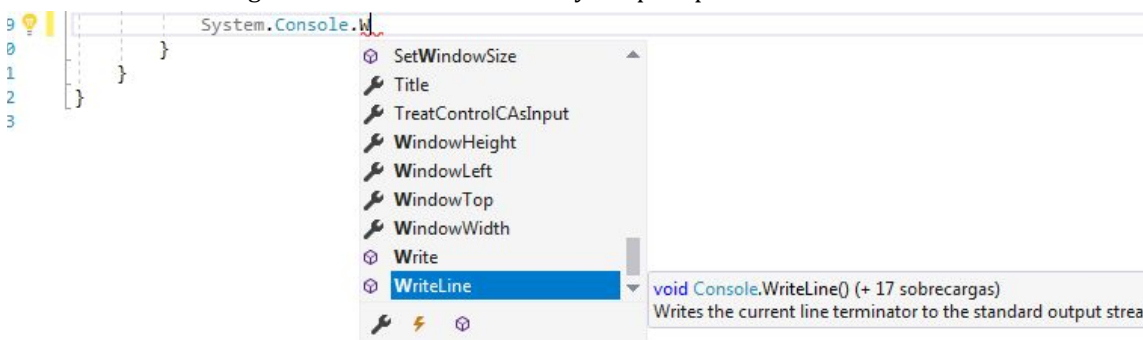
Una vez creado el proyecto HadaConsola en lenguaje C#, VisualStudio nos crea una serie de ficheros en diferentes directorios de nuestro sistema, con el objetivo de adelantarnos trabajo y ahorrarnos tareas poco productivas. Uno de esos archivos es el fichero *Program.cs*, que es el nombre por defecto que el IDE proporciona a los objetos creados de manera automática, es decir:

```
[Tipo Fichero][número]*\[extensión]
Ejemplos:
    Solution1.sln  [Solution][1].[sln]
    Class1.cs     [Class][1].[cs]
```

El primer paso para poder comenzar la implementación de nuestro programa consiste escribir código fuente C# en el fichero *Program.cs* usando el editor. El código debe llamar al método [System.Console.WriteLine\(\)](#) para mostrar la cadena literal "Hello world!" en la ventana de la consola (salida del programa). Por suerte para tí, VisualStudio crea un programa que hace justo eso.

Podrás comprobar cómo, ante la introducción de determinadas instrucciones de C# o cuando pasas el cursor sobre las mismas, el propio IDE detecta las palabras reservadas y si son un espacio de nombres o una clase o una función o una variable. Por ejemplo, observando la Figura 5, verás que te ofrece un menú contextual para la clase Console, dentro del espacio de nombres System, donde, al escribir la letra W, te muestra todos sus métodos (*SetWindowSize*, *Write*, *WriteLine*) y propiedades (*Title*, ..., *WindowWidth*) en una lista desplegable, junto con una ayuda contextual del elemento.

Figura 5. Menú contextual de ayuda para palabras reservadas

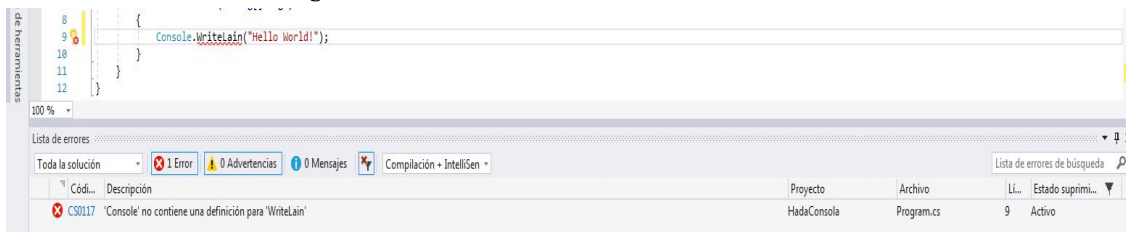


4.5. Mi primer error

Supongamos que has borrado la línea de código que ha autogenerado VisualStudio porque quieres practicar. Después has tecleado la primera y única línea del programa pero has escrito el nombre de función a tu manera. Como podemos observar en la Figura 6, este error aparece convenientemente subrayado en rojo

para que no se te olvide que está ahí. Desaparecerá cuando escribas correctamente el nombre del método que queremos emplear, y que en este caso es `WriteLine`.

Figura 6. Visualización de errores en VisualStudio



Como podemos observar en la Figura 6, el error también nos aparece en lo que se llama la “Ventana de Tareas” o “Ventana de errores”. Si hacemos doble click en la línea en la que se nos muestra el error, automáticamente el cursor se situará sobre la zona del código fuente en la que está éste, quedando el IDE a la espera de que modifiquemos esa línea para arreglarlo.


Si no aún así, no hubiéramos caído en cuál es el nombre correcto, podemos pulsar sobre el botón , y VisualStudio nos sugerirá diversas soluciones como puedes ver en la Figura 7, entre las que se encuentra cambiar `WriteLain` por `WriteLine`.


Figura 7. Visualización de ayuda para solucionar errores en VisualStudio



4.6. Generar y ejecutar un proyecto

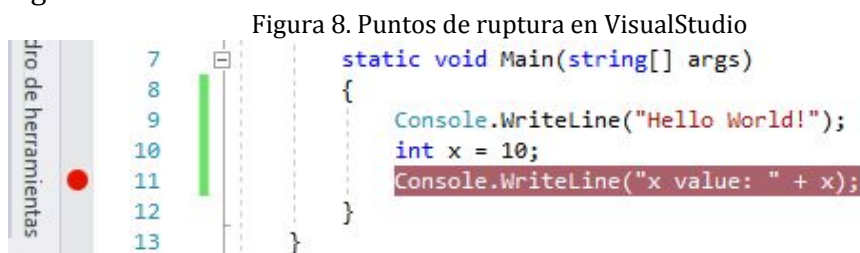
Ahora sin errores, podremos generar o compilar la solución y disfrutar de la visualización posterior de nuestro mensaje “Hello World!” en la consola MS-DOS de nuestro ordenador. Para ello, disponemos de tres opciones:


- Menú Compilar > Compilar solución: Esta opción realizará las tareas de compilación y linkado de los diferentes proyectos que compongan nuestra solución. Como nuestra solución únicamente está formada por una aplicación de consola, esta opción compilará el proyecto y generará un ejecutable sólo en caso de que el código fuente se encuentre libre de errores.
- Menú Compilar > Compilación por lotes: Si nuestra solución o proyecto son tan grandes que pueden llegar a comprometer los recursos de nuestra máquina (supongamos que desarrollamos un procesador de textos con C# y queremos compilarlo), tendremos la posibilidad de generarlos en batch o

- por lotes, es decir, cómo y cuando la máquina pueda, y a trozos
- Ejecutar: Pulsando la tecla Ctrl + F5 ejecuta el proyecto sin compilarlo si quiera o también puedes hacer clic en el botón .

4.7. Depurar un proyecto

Supongamos que, por alguna razón desconocida, no funciona nuestro programa "Hello World!" y necesitamos recurrir a una herramienta de depuración. Estas herramientas ejecutan las aplicaciones con el depurador adjunto. Al hacerlo, el depurador proporciona muchas formas de ver lo que hace el código mientras se ejecuta. Para ello, puedes establecer puntos de ruptura o "*break points*" en las líneas de tu programa que consideres sospechosas, como la línea 11 de la Figura 8.



Para iniciar la depuración, debes ejecutar normalmente, mediante el atajo F5 o el botón , y el entorno detendrá la ejecución del programa en todas y cada una de las líneas que estén marcadas con un punto de interrupción. Una vez detenida la ejecución, puedes comprobar los contenidos de las variables que te interesen y proceder a la corrección de los errores que se estén produciendo.

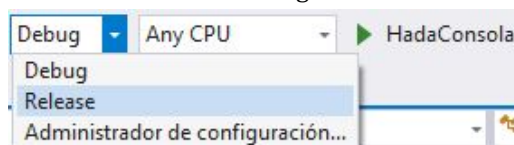
Una opción importante para la depuración de proyectos, independientemente de su tamaño, es el **administrador de configuración**. Disponemos de dos configuraciones estándar y predefinidas por el IDE: Debug y Release.

Cuando nos encontramos desarrollando un proyecto nos encontramos con la **configuración de Debug** seleccionada por defecto y solemos decir que estamos en debug o en modo depuración. Esta configuración *permite establecer puntos de ruptura* y seguir la ejecución de nuestro proyecto paso a paso, instrucción por instrucción, comprobando dónde está el error para poder solventarlo rápidamente. Esta configuración de depuración posee la característica de que utiliza librerías de Windows que no son las de producción, sino las de depuración.

Es por esta razón que una vez finalizada la fase de desarrollo y depuración de errores en tu proyecto, debes tomarte la molestia de cambiar de **configuración** a **modo Release**, para generar la versión definitiva de nuestro proyecto con las librerías de producción. Es decir, una vez que tu proyecto esté libre de errores y te encuentres en situación de generar un ejecutable, o una dll o cualquiera que sea la forma del entregable que estés construyendo, debes cambiar la

configuración de tu proyecto de modo Debug a modo Release, tal como podemos observar en la figura siguiente. Cuando pases al modo Release con la intención de generar el ejecutable definitivo de nuestro programa *no debemos dejar ningún punto de ruptura en la aplicación*.

Figura 9. Cambiar de modo Debug a modo Release en VisualStudio



4.8. Ejercicio: Mi primera librería

En lugar de guardar la lógica de tu aplicación en un proyecto de tipo aplicación de consola, vamos a crear una biblioteca. Esto te permitirá reutilizar tu "Hello World!" en otros proyectos. Dentro de la solución hada-p0, crea un PROYECTO en lenguaje *Visual C#* mediante la utilización de la PLANTILLA de proyecto que se denomina "Biblioteca de clases", que se llamará **HadaBiblioteca**.

Dentro de la librería hay una clase vacía de nombre `Class1.cs`, renómbrala a `MiNombre.cs`. Para ello, dentro del explorador de soluciones, pulsa con el botón derecho del ratón sobre la clase y selecciona "Cambiar Nombre".

Dentro de la clase `MiNombre`, añade una propiedad llamada `Nombre` de tipo `string` y un método para recuperar su valor.

```
public string Nombre { get; }
```

Crea el constructor de esta clase para que reciba una cadena como parámetro y actualice la propiedad `Nombre`.

```
public MiNombre(string nombre)
{
    Nombre = nombre;
}
```

Dentro de la clase `MiNombre`, implementa un método que se llame `Write` e imprima la propiedad `Nombre` tras la cadena "My name is ".

```
public void Write()
{
    Console.WriteLine("My name is {0}", Nombre);
}
```

Compila la biblioteca.

4.9. Ejercicio: Mi primera web con ASP .NET

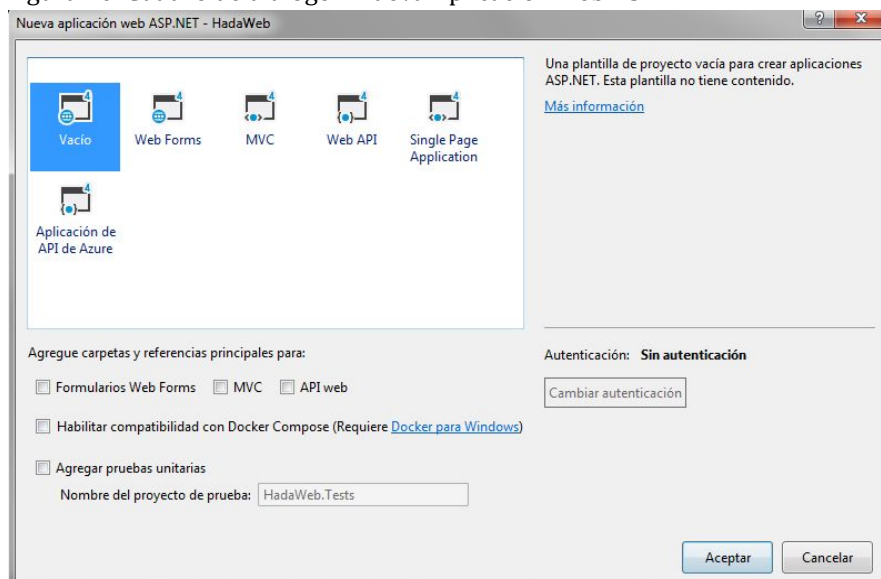
Ahora vamos a crear una web donde mostrar nuestro nombre. Para ello, dentro de la solución **hada-p0**:

1. Crea un PROYECTO en lenguaje *Visual C#* mediante la utilización de la PLANTILLA de proyecto que se denomina "Aplicación web ASP.NET (.NET

Framework)”, que se llamará **HadaWeb**.

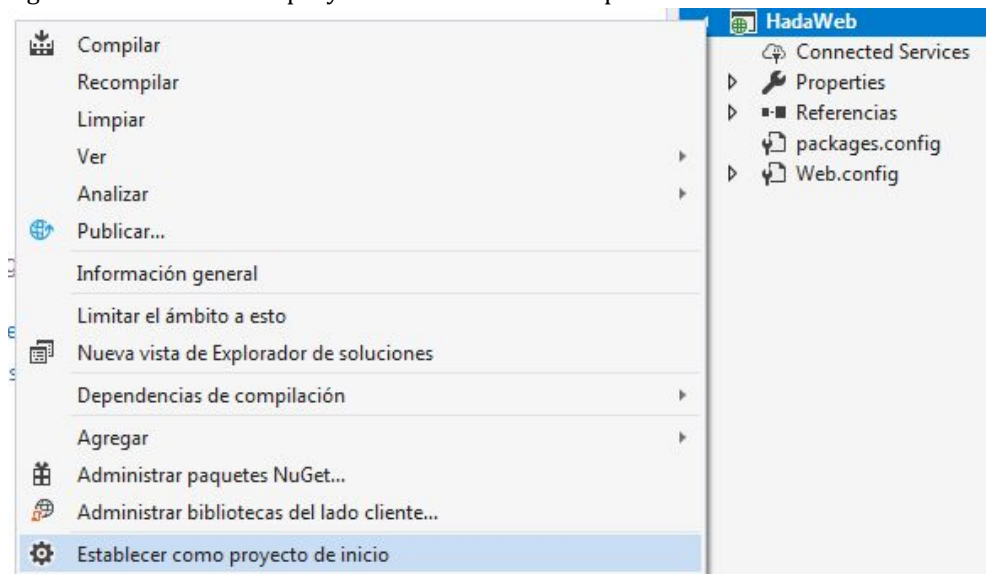
2. En el cuadro de diálogo “Nueva Aplicación web ASP.NET” selecciona la plantilla de proyecto “Vacío” y pulsa “Aceptar”.

Figura 10. Cuadro de diálogo “Nueva Aplicación web ASP.NET”



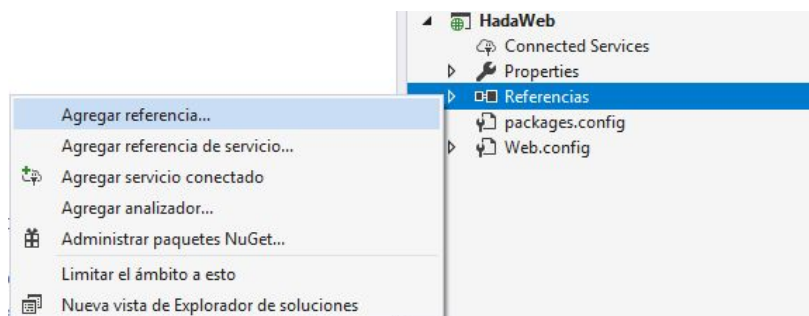
3. Establece el nuevo proyecto creado como proyecto de inicio de la aplicación: haz clic sobre el proyecto con el botón derecho y selecciona “Establecer como proyecto de inicio”.

Figura 11. Establecer un proyecto como inicio de la aplicación



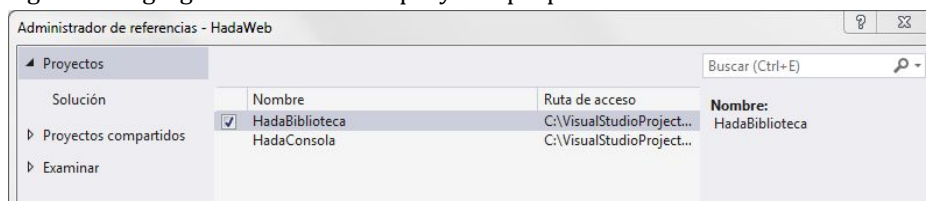
4. Añade la referencia a tu biblioteca de clases (**HadaBiblioteca**):
 - 4.1. Desde el explorador de soluciones, dentro en el proyecto **HadaWeb**, haz clic sobre el elemento *Referencias* con el botón derecho y elige *Agregar Referencia*.

Figura 12. Agregar referencia como una dependencia



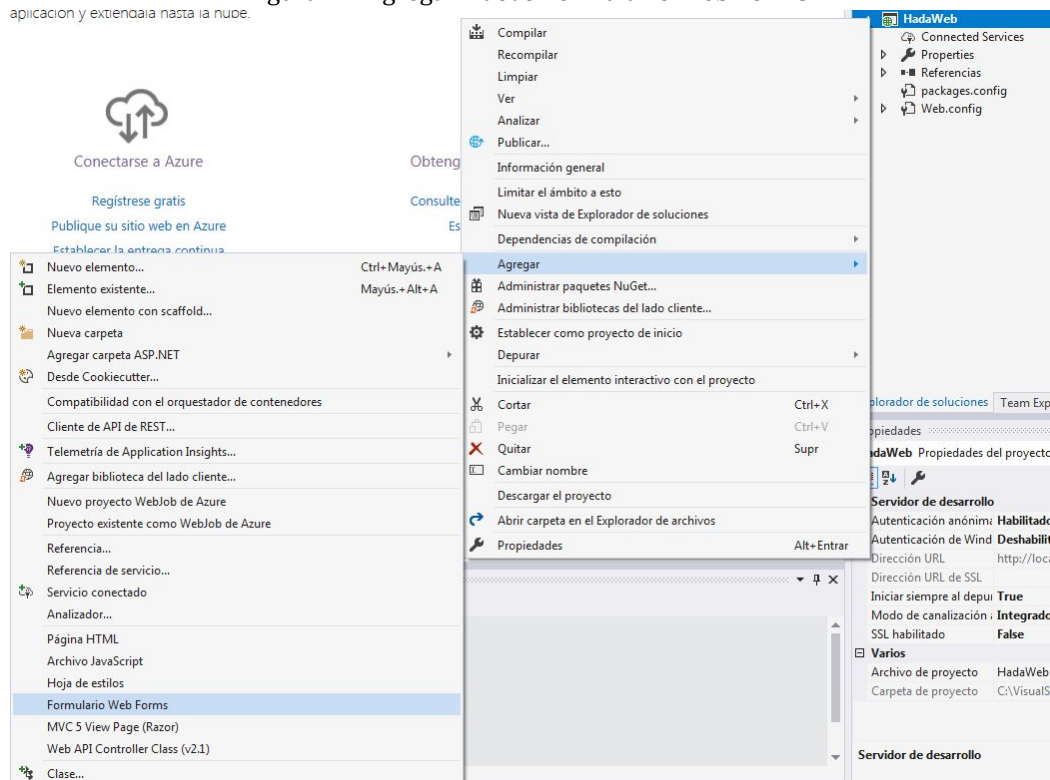
- 4.2. A continuación, en el cuadro de diálogo “Administrador de referencias”, desde la opción de Proyectos, selecciona tu biblioteca (**HadaBiblioteca**).

Figura 13. Agregar referencia a un proyecto propio



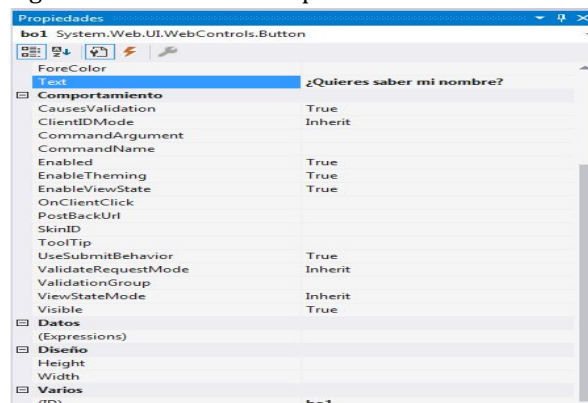
- 4.3. En el explorador de soluciones, dentro en el proyecto **HadaWeb**, haz click con el botón derecho y selecciona “Agregar nuevo elemento”. Escoge “Formulario Web Forms” (Figura 14) y dale el nombre “HadaWebForm”. El resultado es que se añade un nuevo fichero llamado HadaWebForm.aspx que contendrá una nueva página web del proyecto.

Figura 14. Agregar Nuevo Formulario Web Forms



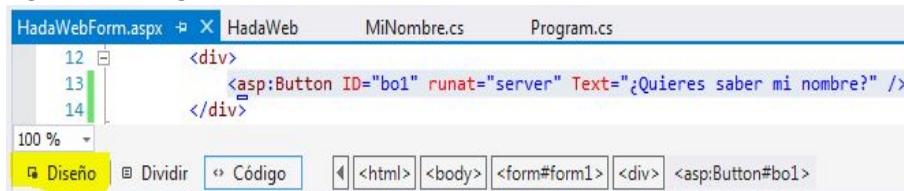
- 4.4. En la página que acabas de crear, añadirás un botón. Para ello usa el “Cuadro de herramientas” (puedes usar el atajo de teclado Control + Alt + X). Selecciona el botón y arrástralo hasta las etiquetas html `<div></div>`. Después cambiale el identificador a “bo1” y el texto a “¿Quieres saber mi nombre?”. Utiliza la ventana de propiedades, seleccionando el botón y editando las propiedades ID y Text. También puedes editar directamente la ventana de Propiedades, como en la Figura 15.

Figura 15. Ventana de Propiedades de un botón web



- 4.5. Ahora solo falta que añadas el siguiente código al evento por defecto del botón. Para ello, pulsa la pestaña “Diseño” en la parte inferior del editor (resaltado en amarillo en la Figura 16) y después pulsa dos veces sobre el botón que aparecerá.

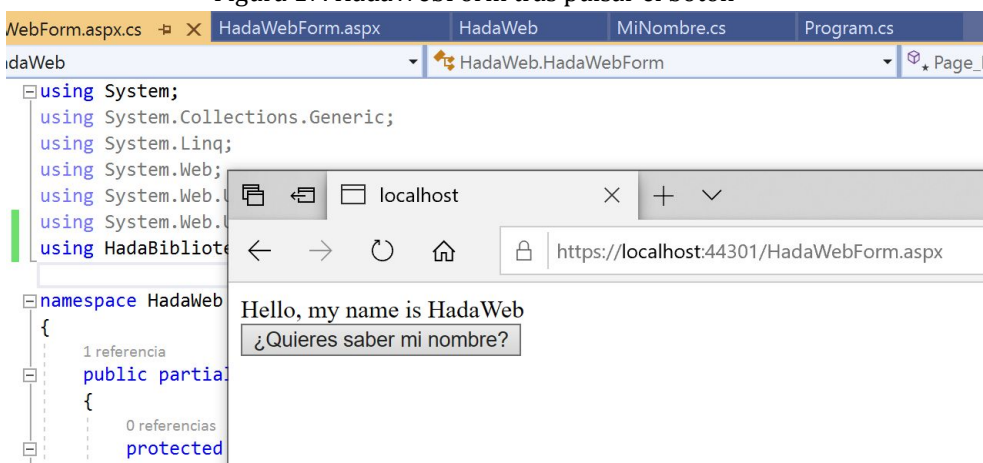
Figura 16. Código HTML resultante del botón



- 4.6. Se abrirá un nuevo archivo llamado **HadaWebForm.aspx.cs**. Este archivo también se corresponde con el formulario HadaWebForm de tu aplicación web. En el editor verás código C# que también se corresponde con el botón, en concreto en la línea 18 tienes el método llamado `bo1_Click` que gestiona la lógica del evento "Click" del botón `bo1`. Ahora la función está vacía por lo que debes escribir el siguiente código:


```
protected void bo1_Click(object sender, EventArgs e)
{
    MiNombre nombre = new MiNombre("HadaWeb");
    String message = "Hello, my name is " + nombre.Nombre;
    Response.Write(message);
}
```
- 4.7. Será necesario que incluyas un nuevo `using` a la biblioteca del ejercicio anterior, para que utilice la referencia creada en 4.2, y que compiles la solución para que encuentre la referencia a la clase.
- 4.8. Establece como página de inicio el formulario HadaWebForm (botón derecho sobre **HadaWebForm.aspx** > "Establecer como página de inicio")
- 4.9. Ejecuta el proyecto.
- 4.10. Es probable que el navegador considere tu web no es segura, añade una excepción para poder navegar.
- 4.11. Verás tu recién creada web con su botón. Si pulsas sobre él, verás el mensaje que has indicado, tal como se observa en la Figura 17.

Figura 17. HadaWebForm tras pulsar el botón





Referencias

Práctica basada en estos tutoriales, que podéis utilizar para profundizar:

- [1] <https://docs.microsoft.com/es-es/visualstudio/ide/quickstart-projects-solutions?view=vs-2017>
- [2] <https://docs.microsoft.com/es-es/visualstudio/ide/visual-studio-ide?view=vs-2017>
- [3] <https://docs.microsoft.com/es-es/visualstudio/debugger/getting-started-with-the-debugger?context=visualstudio%2Fdefault&contextView=vs-2017&view=vs-2017>
- [4] <https://docs.microsoft.com/es-es/dotnet/standard/choosing-core-framework-server>