Práctica 2

Entorno de programación Visual C#.

Introducción a la programación visual

Material: PC y Visual Studio 2013

Duración: 3 horas (1 hora clase de teoría + 2 horas prácticas)

Lugar: Laboratorios de prácticas (Laboratorio de Redes-Hardware)

La herramienta de desarrollo que utilizaremos para el desarrollo de las prácticas de la asignatura será el entorno de programación de Visual C#, Microsoft Visual Studio 2013, el cual permite la creación, compilación y ejecución de programas escritos en lenguaje Visual C#; tanto en modo consola como en modo gráfico. Se recuerda al alumnado que, para la mejor comprensión y asimilación de los conocimientos y habilidades desarrollados durante la práctica, deberá haberse estudiado previamente el material docente disponible.

Introducción:

La programación visual tiene, como principal objetivo, el diseño y el desarrollo de programas dotados de una interfaz hombre-máquina visual amigable, intuitiva y fácil de emplear.

Formalmente, la programación visual se apoya en los principios generales de la programación orientada a objetos, aunque es posible el desarrollo de programas de forma similar a la programación tradicional (programación imperativa en C, por ejemplo), teniendo en cuenta ciertas consideraciones de desarrollo, uso y comportamiento de los programas. Tanto durante el transcurso de las sucesivas sesiones prácticas, como en el desarrollo de los contenidos en las clases de teoría, se irán introduciendo los conocimientos necesarios para desarrollar una aplicación visual.

Para finalizar esta introducción, y aunque desarrollaremos más adelante el concepto en función de lo que sea necesario, un objeto es un tipo de variable “especial”, la cual, no

solamente puede almacenar un valor (o valores), de manera similar a como lo hace una variable de tipo entero por ejemplo en C (sea un entero simplemente o una colección de enteros *array*), sino que también puede tener funciones propias para operar con su valor o valores.

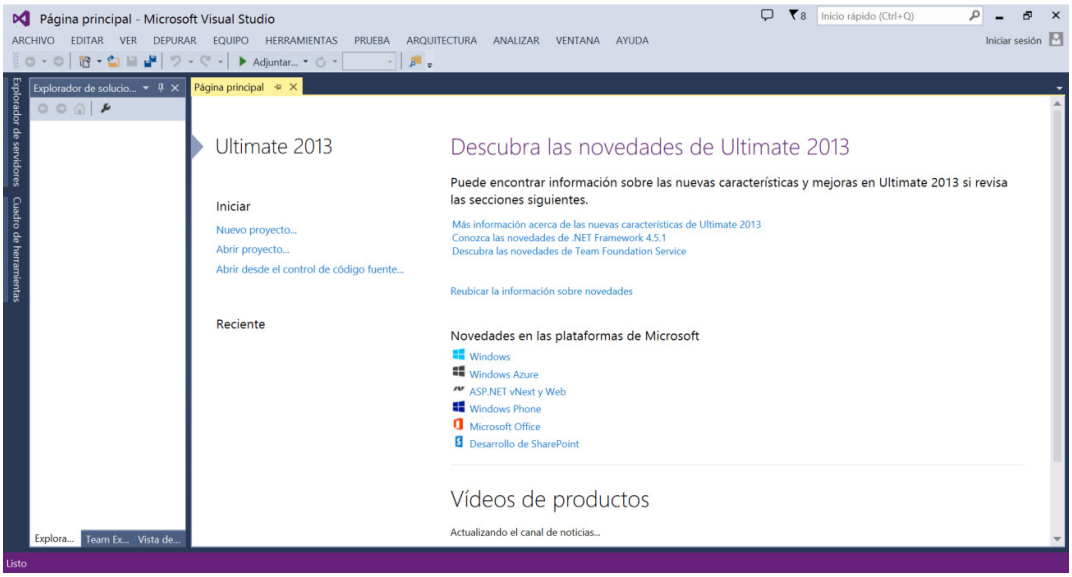
Desarrollo de la práctica:

1. Entorno de programación Visual C#.

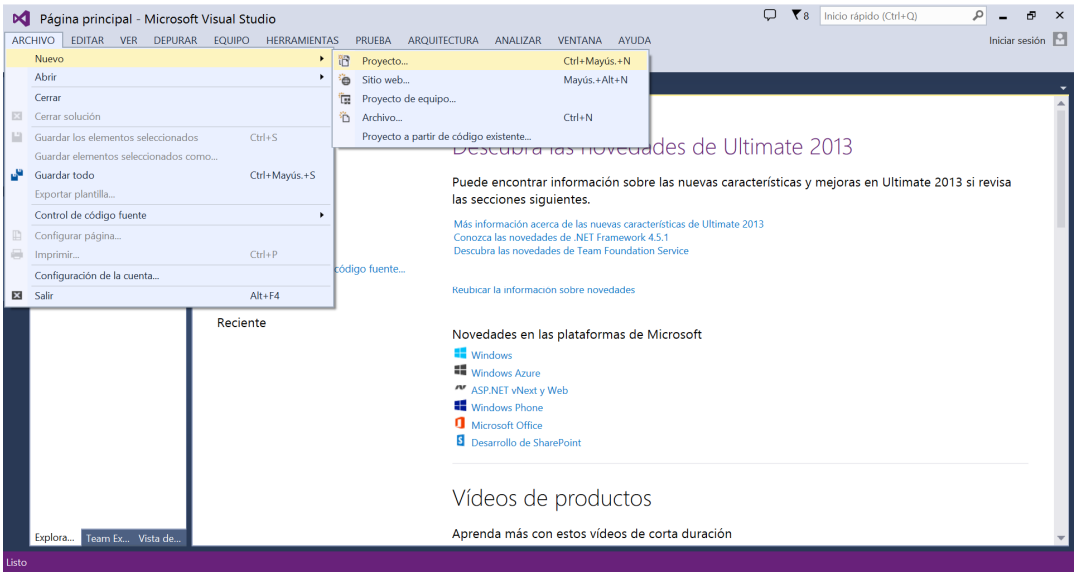
En esta primera parte de la práctica escribiremos un programa en Visual C# que cree una aplicación visual básica. Realmente, no tendremos que programar nada, la gran capacidad del entorno de programación nos permite crearla siguiendo unos sencillos pasos.

1) Al igual que en la práctica anterior, creamos una carpeta en el Escritorio de nuestro PC con un nombre en concreto, en este ejemplo le pondremos de nombre a dicha carpeta *Práctica 2*. De esta forma, podremos identificar dónde se encuentran los archivos que el entorno de programación generará para el programa que desarrollemos.

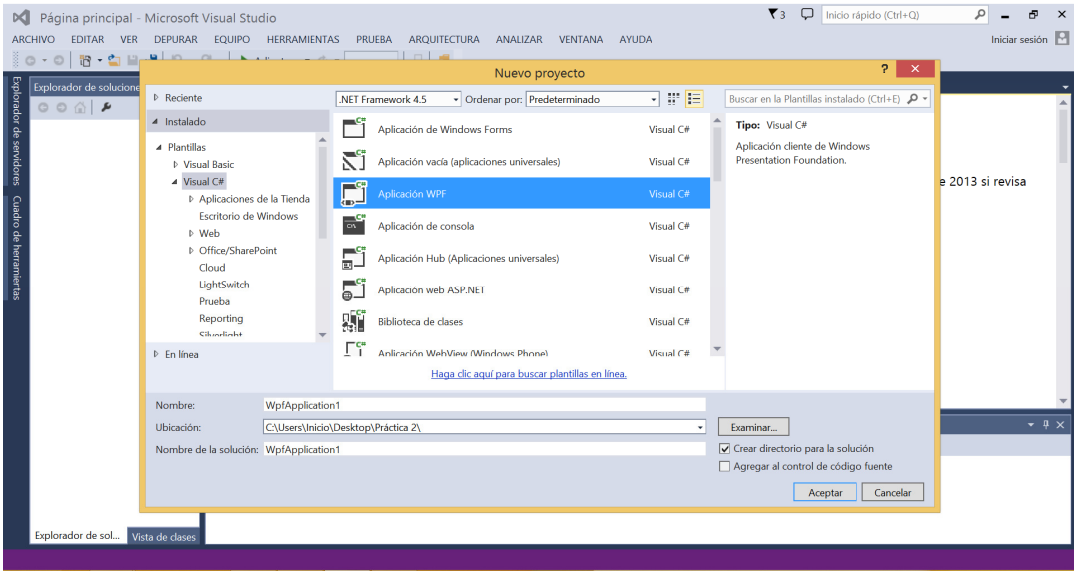
2) Ejecutamos el programa Visual Studio 2013, tras lo cual se nos mostrará la siguiente pantalla (o una similar, dependiendo de diversos factores).



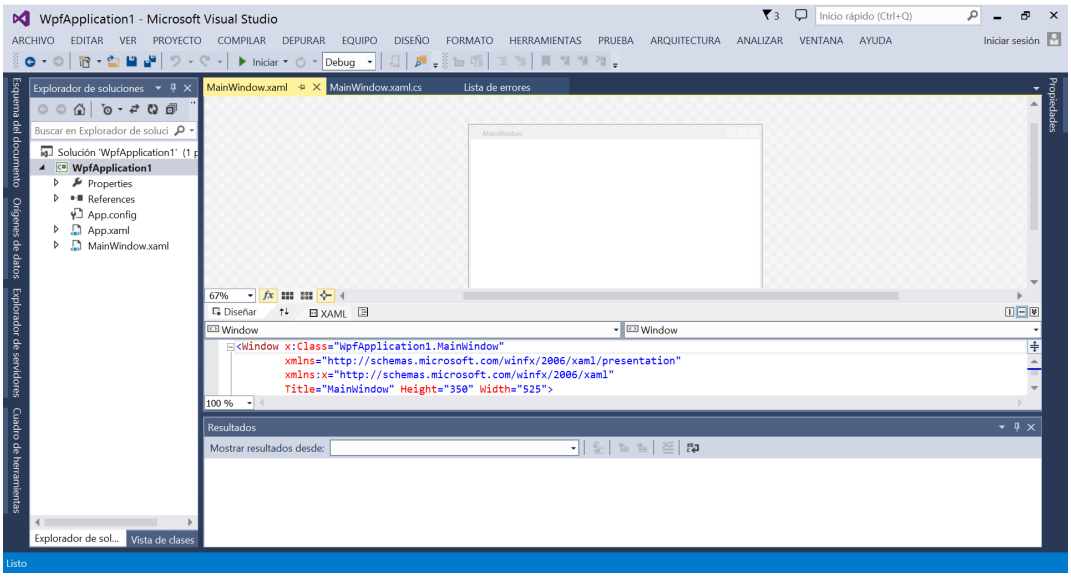
3) Seleccionaremos Archivo => Nuevo => Proyecto



4) En el menú de plantillas de la parte de la izquierda, que aparece en la ventana titulada como *Nuevo Proyecto*, comprobaremos que se encuentran seleccionadas las plantillas de Otros Lenguajes y dentro de ésta la pestaña correspondiente a Visual C#; elegimos *Aplicación WPF* como tipo de proyecto escribiendo, si queremos cambiar el nombre al proyecto *Hola_Visual* por ejemplo como nombre del proyecto que estamos creando (donde pone *Nombre*), en *Ubicación* indicaremos la carpeta donde queremos guardar nuestro proyecto (pulsando en *Examinar…* localizaremos la carpeta Práctica 2 que hemos creado en el escritorio en el paso 1) y, finalmente, pulsaremos Aceptar.



5) En la siguiente ventana que nos aparecerá veremos ya el entorno de trabajo listo para comenzar con el proyecto, es lo que se conoce como *vista de diseño*:



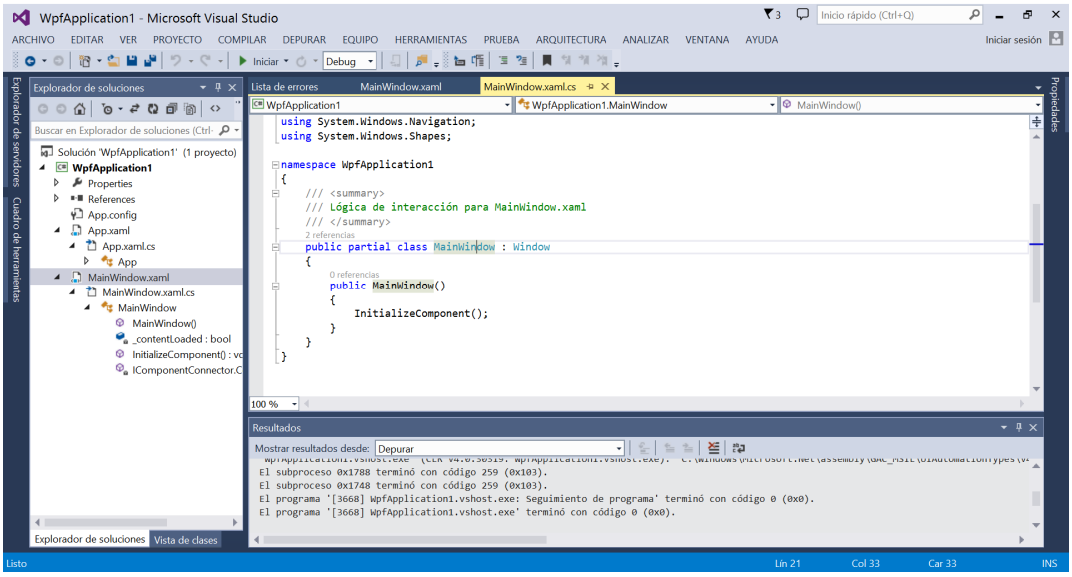
Como podemos ver en el explorador de soluciones, aparecen una serie de elementos y/o ficheros de código, los cuales iremos analizando y comentando según corresponda; mientras que, en la zona central aparece, con el nombre de *MainWindow.xaml*, una plantilla visual de una ventana clásica de Windows, la cual será el soporte de nuestra aplicación. Sin entrar en detalle en el explorador de soluciones, dicha ventana "visual" es la principal diferencia con los proyectos de consola que creamos en la Práctica 1. Dentro del explorador de soluciones se ubica un archivo denominado *MainWindow.xaml.cs* (archivo principal del proyecto), que también se encuentra disponible en una de las pestañas en la zona centra; el código que contiene es el siguiente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
```

```
        /// </summary>
        public partial class MainWindow : Window
        {
            public MainWindow()
            {
                InitializeComponent();
            }
        }
    }
```

En Visual c# el entorno lo crea por nosotros y escribe las líneas de código básicas que necesitamos para comenzar a trabajar. No entraremos a concretar qué significa cada línea de código simplemente diremos que, por el momento, contiene la función principal o *Main* (`MainWindow : Window`) y que lanza y/o ejecuta el *MainWindow();* mostrando nuestra ventana.

A continuación, pulsa en la pestaña denominada *MainWindow.xaml.cs*, para ver el código que hemos mostrado anteriormente. El entorno de programación mostrará lo siguiente:

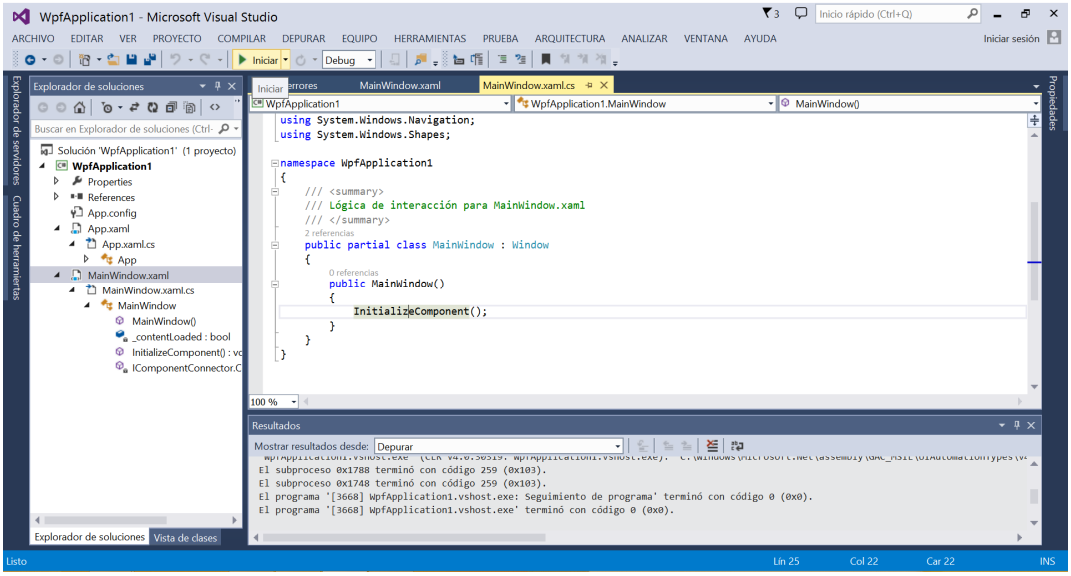


Este archivo (*MainWindow.xaml.cs*) contiene y contendrá la mayor parte del código que desarrollaremos para la aplicación visual. Por el momento, está prácticamente vacío, es el código del *MainWindow.xaml, es decir, el código que interactúa con la parte visual de nuestro programa*. Recuerda que, cuando programemos, añadiremos código sobre el que el propio entorno Microsoft Visual Studio nos ha creado por defecto, no lo borraremos pues podría impedir que nuestra aplicación se ejecutara correctamente.

Microsoft Visual Studio emplea nombres predefinidos para las ventanas, los componentes (clases, objetos y variables) y los eventos que emplearemos para programar. Se recomienda que no se modifiquen dichos nombres, pues podría ocasionar que el programa dejase de

funcionar. Por ejemplo, en el código mostrado en la ventana anterior se invoca a una función que se denomina *InitializeComponent();* la cual inicializa la ventana que se muestra al usuario. Aunque no veremos qué hace, el código se encuentra en el archivo *MainWindow.g.i.cs*.

Por el momento, simplemente hemos creado un proyecto visual vacío y hemos comentado algo de su código y estructura. Puedes ejecutar el proyecto y ver qué ocurre, pulsando sobre el botón Iniciar.

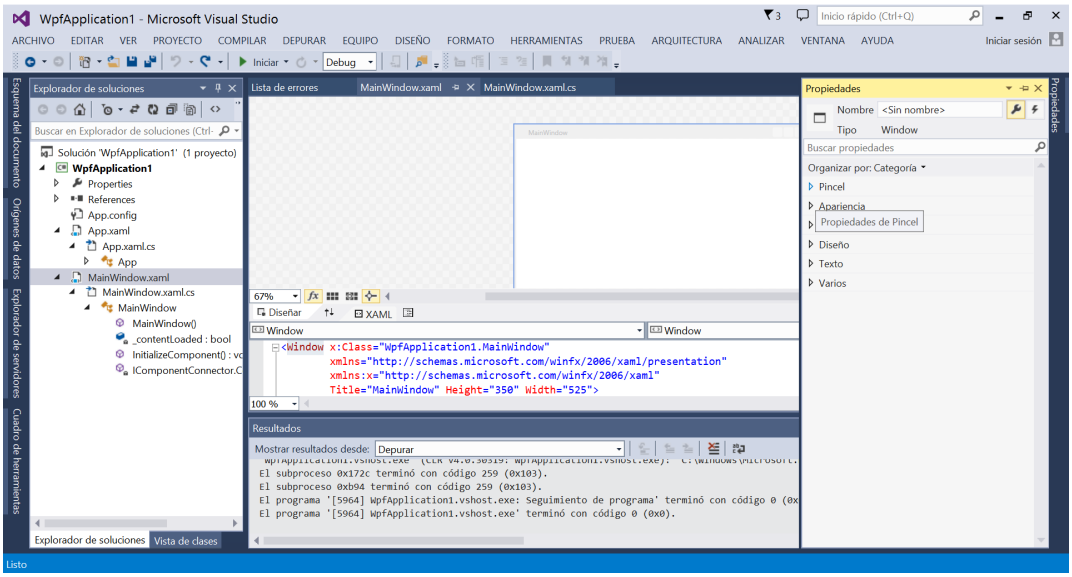


Ejercicio:

1) Ejecuta el programa que acabas de crear y comprueba el resultado. ¿Qué puedes hacer con la aplicación? Localiza el fichero ejecutable de la aplicación que acabas de crear (tendrá el nombre del proyecto y la extensión *exe*), ¿podrías copiarlo y ejecutarlo en otro ordenador? ¿Puedes copiar dicho ejecutable en tu escritorio y ejecutarlo?

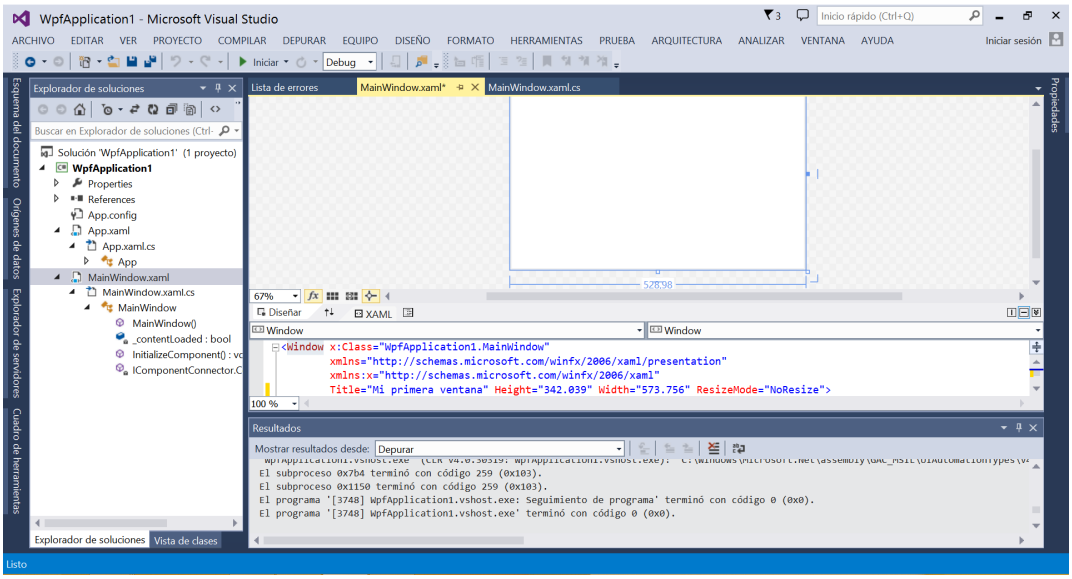
2. Mi primer objeto de Visual C#.

Volviendo al entorno de trabajo, en donde tenemos la pantalla que nos muestra nuestra ventana plantilla (*MainWindow.xaml*), pulsaremos con el botón izquierdo del ratón sobre ella hasta que aparezca rodeada de una línea de color azul, pulsando seguidamente sobre la pestaña de Propiedades que aparece en la zona de la derecha de la pantalla.



En la zona de la derecha nos aparecerá una ventana con las propiedades de la ventana principal de nuestra aplicación. Dicho de otro modo, la ventana que utilizaremos como plantilla es un objeto, y como comentamos en la introducción y en clase de teoría, tiene propiedades (de hecho, un montón de propiedades); el nombre de la ventana, por ejemplo, es una propiedad de la misma; podemos modificarlo dentro del menú *Común*, elemento *Title*, escribiendo el nombre que queremos darle. Tenemos también, por citar otro ejemplo, un campo que nos permite decidir si el usuario podrá cambiar las propiedades de la ventana (por ejemplo, si se permite el redimensionado de la ventana), etiquetado como *ResizeMode*, etc. No los estudiaremos todos, aunque puedes probar con alguno de ellos y comprobar el comportamiento que tiene sobre la ventana *MainWindow.xaml*, cambiado una de estas propiedades y ejecutando nuevamente la aplicación, tal y como se ha visto anteriormente.

Además, desde esta pantalla de trabajo principal, también podemos hacer que el tamaño de nuestra ventana sea mayor, cuando estamos sobre la vista gráfica del *MainWindow.xaml* (vista de diseño). Para ello, simplemente pincharemos en la parte derecha de la misma (en la mitad del cuadrado) y, manteniendo pulsado el ratón, podremos hacer más grande y/o más pequeña la ventana; de modo similar a como trabajaríamos con una imagen. Nos aparecen unos puntos cuadrados donde podemos pinchar y arrastrar:



Ejercicios:

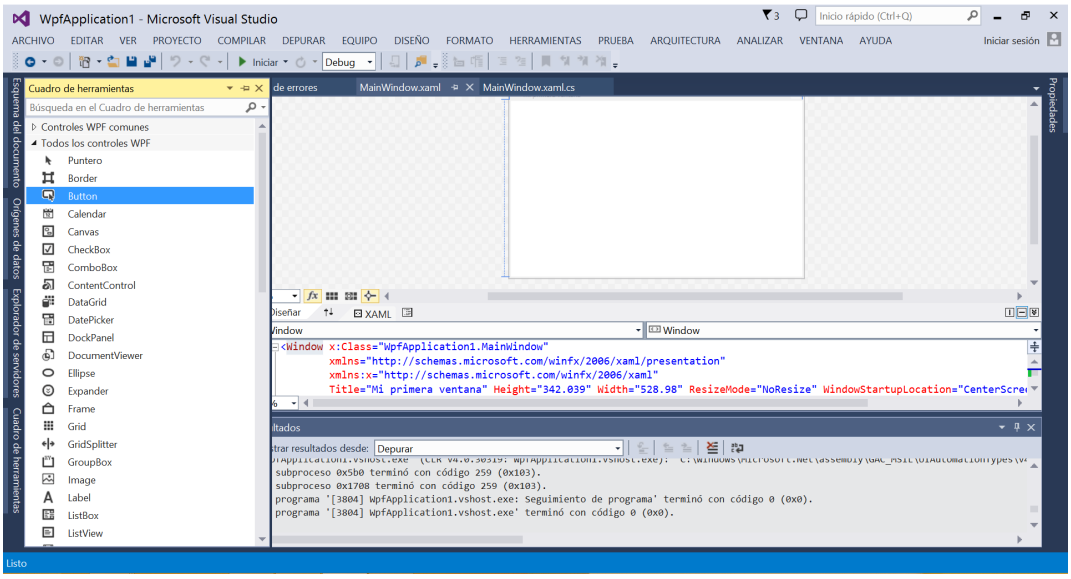
2) Cambia el nombre de la ventana, para que muestre tu nombre, y las propiedades de la misma para que, el usuario, no pueda cambiar las dimensiones.

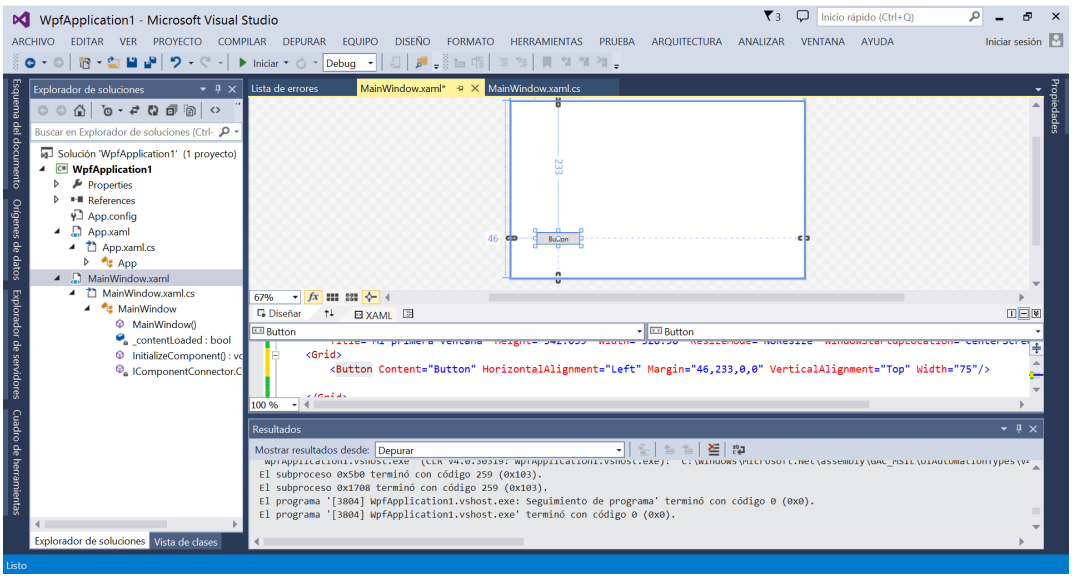
3) Busca el campo u opción, dentro de las propiedades de la ventana, que permita hacer que dicha ventana aparezca centrada en el monitor cada vez que ejecutamos el programa. ¿Qué otras alternativas ofrece dicho campo?

4) Elige un campo u opción de las propiedades a tu elección y prueba para qué se utiliza. **Nota**: no es necesario elegir uno complejo, se trata de que se elija uno tal que entiendas su comportamiento y efecto sobre la ventana. Puedes consultar en Internet.

3. Mi primer programa en Visual C#.

Lo primero que vamos a hacer en nuestro entorno es añadir un botón, elemento básico de toda aplicación visual. Nada más sencillo, buscamos el desplegable/pestaña *Cuadro de herramientas* y pulsamos sobre él (se encuentra en la zona izquierda de la pantalla, arrastrándolo hacia la vista de diseño de nuestra aplicación:

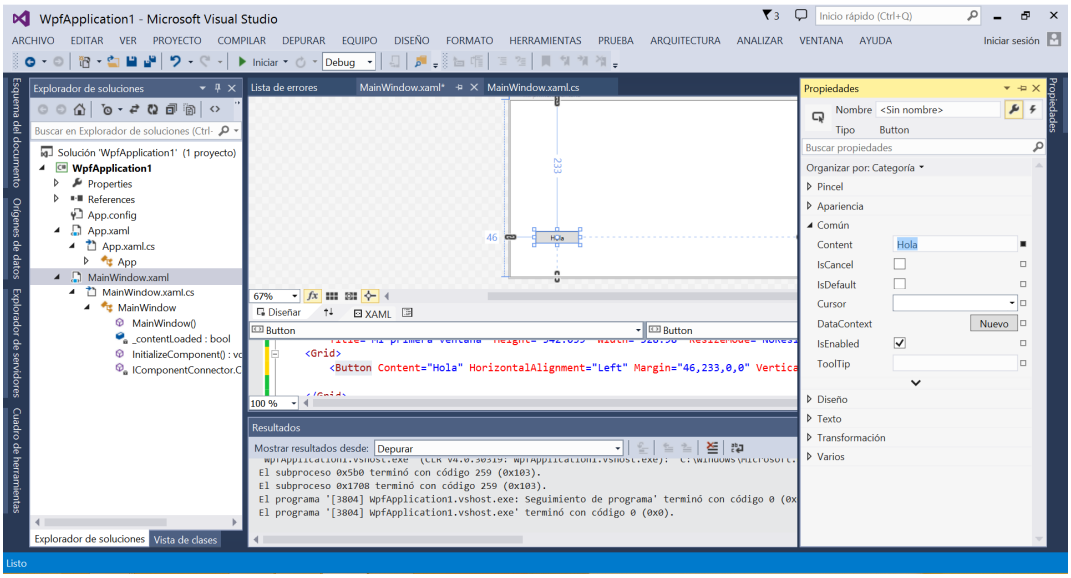




De igual forma que vimos que la ventana *MainWindow.xaml* que tenía propiedades, el botón que acabamos de crear también las tiene. Podemos verlas, haciendo click izquierdo sobre él y seleccionando la opción de Propiedades en la pestaña superior derecha. No las veremos todas, aunque se recomienda probar alguna de ellas, como la que permite cambiar el texto que se visualiza sobre el botón (campo o propiedad *Content*, dentro del menú Común). Recuerda que, de la misma forma que vimos las propiedades del *MainWindow.xaml*, el botón que acabamos de crear tiene las suyas propias, todo en Visual tiene propiedades.

Un botón es un componente que nos permite pulsar sobre él y desencadenar una acción o evento, abrir un archivo, imprimir un mensaje, encender un motor o abrir una válvula (aunque esto último no todo se verá este curso). Sea cual sea la acción a desencadenar tenemos que

programarla; prueba a ejecutar el programa nuevamente y comprueba qué sucede cuando pulsas en el botón.



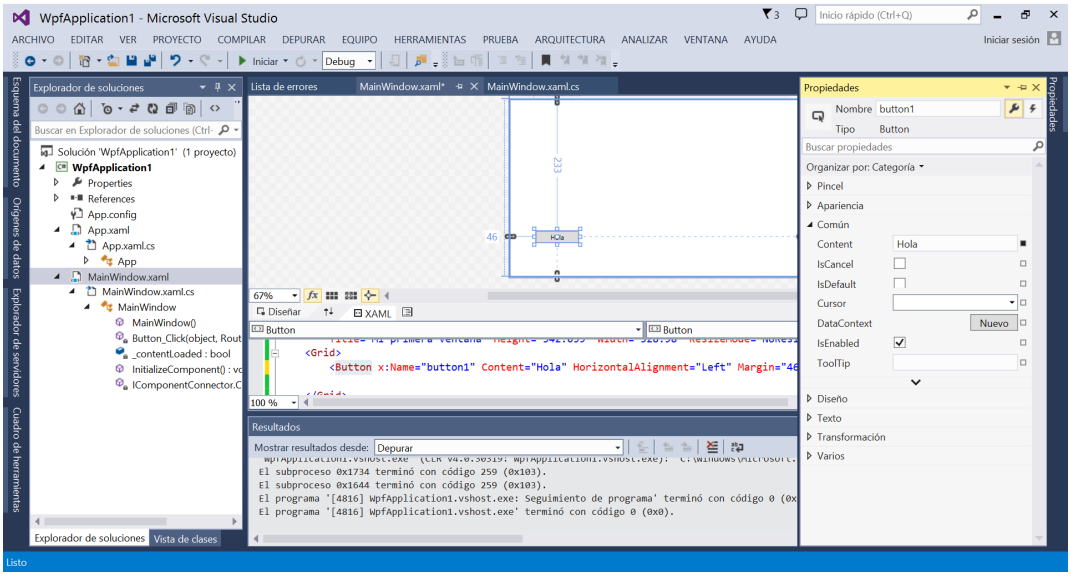
Para programar qué se debe hacer cuando se pulsa el botón en nuestra aplicación, haz doble click sobre el botón en la vista de diseño y verás que se abre la vista *MainWindow.xaml.cs*, para que escribas el código que quieres ejecutar cuando el botón es pulsado:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
```

```
        }
    }
}
```

Podemos ver que, el entorno, nos añade una función con un nombre bastante descriptivo, por lo que ubicaremos el código que queremos ejecutar cuando el botón es pulsado dentro de esta función (la función es Button_Click()). A partir de este punto, el código que podemos añadir en estas funciones es C#, con ciertas consideraciones para poder hacer referencia e interactuar con los elementos visuales de nuestra aplicación. Veremos algunos ejemplos más adelante.

Vamos a utilizar esta primera función, *Button1_Click*(), para acceder al componente botón que añadimos previamente y cambiarle el nombre cuando pulsemos sobre él. Para ello, el nombre por el cual se conocerá al citado botón en el programa será, por ejemplo, *button1*. Para ello, debemos volver a la vista de diseño, pulsar sobre el botón y abrir el menú de propiedades. El campo Nombre, en el menú de Propiedades, se encuentra vacío. Procederemos a escribir en dicho espacio *button1*, pulsando la tecla *enter* tras ello.



Ahora ya podemos volver al código del programa y escribir la instrucción que nos permitirá cambiar el nombre al botón, realmente la instrucción es el nombre que acabamos de darle al botón, seguido de punto más la propiedad del mismo que queremos modificar. El código que emplearemos será el siguiente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            button1.Content = "Adios";
        }
    }
}
```

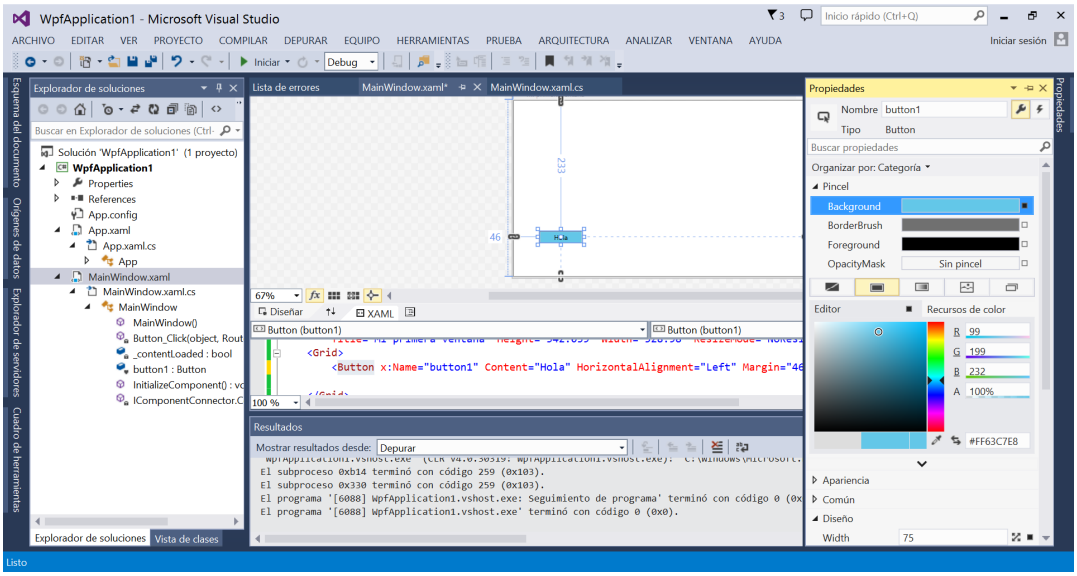
Como puede verse, los campos o propiedades de los botones se pueden cambiar durante la ejecución de la aplicación, por medio de eventos lanzados por el usuario, en este caso, la pulsación de un botón. La modificación de las propiedades de los componentes que agreguemos deberá ser consecuente con el tipo de la misma, es decir, el campo *Content* del botón es de tipo String en C# y sigue las reglas que se aplican a este tipo de variables, por lo que el texto introducido irá entre comillas dobles. En resumen, podemos modificar una propiedad de un componente mediante código, durante la ejecución del programa.

Ejercicio:

5) Haz que cuando se pulse el botón aparezca la frase “la nieve es blanca” sobre éste. ¿Qué problemas tienes? ¿Se te ocurre cómo solucionarlo? **Pista**: juega con el botón en la vista de diseño de la aplicación.

Puedes cambiar colores, sombras, estilos, etc., prácticamente todo lo que se nos ocurra en relación a un botón. Algunas de estas propiedades, como el tamaño, puedes cambiarlas tanto desde su menú de Propiedades como desde la vista de diseño de la aplicación, pinchando

sobre uno de sus laterales de la misma forma que hicimos con la ventana. Otra propiedad curiosa es el campo is*Enabled*, que puede tomar valores *True o False*; el cual permite habilitar o deshabilitar el botón, respectivamente.



Además de poder modificar sus propiedades, de la forma que hemos visto, algunos componentes tienen asociados métodos, que no son otra cosa sino funciones, como comentamos en clase de teoría. Veremos un ejemplo con otro componente más adelante, aunque comentamos aquí que no los estudiaremos todos, debido a la gran cantidad de propiedades y métodos que tiene cada uno de ellos.

Ejercicio:

6) Haz que, después de cambiar el texto del botón, tal y como se indicó en el ejercicio 2, se deshabilite el botón para impedir su uso. *Pista*: necesitas modificar el valor del campo o propiedad is*Enabled* del botón.

4. Objetos y eventos en Visual C#.

Vista esta primera introducción, podemos añadir algo más de teoría, mediante un pequeño resumen:

- Los componentes visuales que empleemos en nuestras aplicaciones son objetos, desde los botones que acabamos de ver hasta el propio *MainWindow.xaml* (ventana) que contiene o sustenta el resto de los objetos. Todos ellos tienen propiedades, a las que se puede acceder mediante el campo correspondiente, tanto desde la vista de diseño como mediante su programación. Es más, cada campo o propiedad tiene un formato o tipo de datos permitido, dependiendo de los valores que pueda tomar (enteros, cadenas de caracteres o String, float, double, booleanos, etc.).
- Los componentes se relacionan e interactúan mediante eventos: pulsar un botón, seleccionar una opción, ejecutar una acción automáticamente mediante un timer, etc. Iremos viendo distintos tipos de eventos a lo largo de las prácticas. Por el momento, sólo hemos visto el evento desencadenado cuando se pulsa un botón, creado automáticamente por el entorno de programación (función *Button_Click(object sender, RoutedEventArgs e)*, no comentaremos los atributos del evento.
- Además, podremos emplear el lenguaje C#, para completar nuestros programas.

Como ya hemos visto, programar de forma visual difiere de los programas tradicionales de consola; pero estos últimos se pueden traducir y/o modificar convenientemente. Ejemplo: crear una aplicación que, tras pulsar un botón nos aparezca por pantalla el mensaje “Hola mundo!!!”. Modifica el código a ejecutar cuando se pulse el botón de la siguiente manera:

```
private void Button1_Click(object sender, RoutedEventArgs e)
{
  MessageBox.Show("Hola mundo!!!");
}
```

Acabas de crear tu primer programa con mensaje por pantalla incluido. Normalmente, cuando ejecutas un programa visual éste se comunica contigo de esta forma, cuando tiene que avisar/comunicar algo que requiera tu atención.

Ejercicios:

7) ¿Serías capaz de mostrar, después de pulsar un botón, el mensaje “Hola mundo!!!” en un componente del tipo TextBox? **Pista**. Si lo necesitas, busca ayuda en Internet para aprender en relación a los TextBox de Visual C#. Debes seguir un desarrollo similar a cuando creaste un

botón en los pasos previos, le diste un nombre para que el programa lo reconociera y cambiaste una de sus propiedades al pulsar un botón.

8) La instrucción de visualización de mensajes y/o alertas (*MessageBox.Show()*) permite una gran cantidad de riqueza en la información mostrada en la ventana emergente generada (puedes consultar http://tech.pro/tutorial/611/csharp-dialogs-part-1-messagebox o http://www.wpf-tutorial.com/dialogs/the-messagebox/). Prueba alguna de las combinaciones que se muestran en el enlace dado en este ejercicio.

9) La instrucción *MessageBox.Show("This MessageBox has extra options.\n\nHello, world?", "My App", MessageBoxButton.YesNoCancel);*, ¿qué muestra por pantalla?

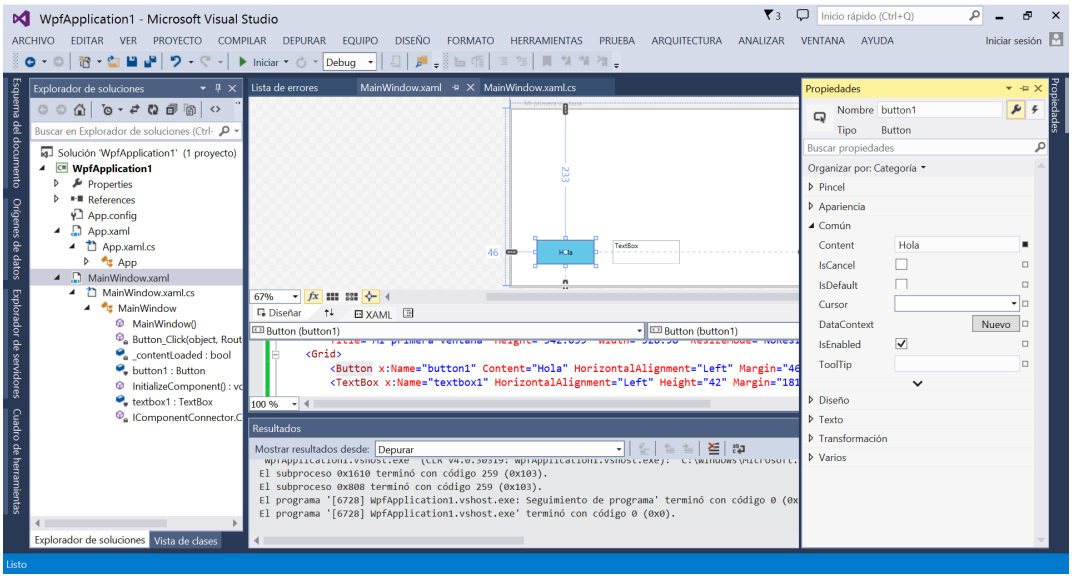
10) ¿Cuántas opciones tienes disponibles para el icono visual informativo (parámetro *MessageBoxIcon*) que se muestra al lado del mensaje en la instrucción *MessageBox.Show("Hello, world!", "My App", MessageBoxButton.OK, MessageBoxImage.Information);*? **Pista**. No copies y pegues la línea anterior, escríbela poco a poco y deja que el entorno de programación te muestre las opciones.

5. Variables en Visual C#.

Los programas funcionan mediante la manipulación de los datos almacenados en la memoria, las variables. En esta sección, veremos cómo utilizar variables, algunas de las cuales serán propias del lenguaje C#, mientras que otras son características del lenguaje C, en general. Por ejemplo, las variables de C char, *int, float y double* también se pueden emplear en C#; siguiendo los mismos principios para su declaración e inicialización que en C. Además, existen tipos nuevos que añaden potencia y flexibilidad al lenguaje, como iremos viendo en las sucesivas prácticas, según sea necesario.

Variables de tipo *String*

El primer nuevo tipo de variable, al cual echaremos un vistazo, es la variable de tipo cadena (*String* en C#). Las variables de cadena son siempre texto y serán tratadas de forma análoga a las cadenas de caracteres de C. Ejemplo: escribir un programa en Visual C# en el que, tomando el texto introducido en un componente cuadro de texto (TextBox), almacene el valor en una variable de tipo *String* para, finalmente, mostrar el texto en un cuadro de mensaje cuando pulsemos un botón. Lo primero de todo, dibujaremos en nuestro *MainWindow.xaml*, los componentes que necesitamos: un botón y un cuadro de texto.



El código necesario estará ubicado dentro del evento click del botón, según el enunciado del ejercicio.

```
private void button1_Click(object sender, EventArgs e)
{
     String cadena;

     cadena = textbox1.Text;
     MessageBox.Show(cadena,"My App", MessageBoxButton.OK,
                     MessageBoxImage.Information);
}
```

Las variables *String* en C# son muy potentes, permiten operaciones como la concatenación mediante el operador suma. Ejemplo:

```
private void button1_Click(object sender, EventArgs e)
{
  String cadena;

  cadena = "hola "+textbox1.Text;
  MessageBox.Show(cadena,"My App", MessageBoxButton.OK,
                  MessageBoxImage.Information);
}
```

El componente *TextBox* tiene, además de propiedades, métodos asociados, es decir, funciones, lo que nos permite realizar ciertas operaciones con ellos. Por ejemplo: la instrucción *textbox1.Clear()*, borra el contenido del componente cuando se ejecuta.

Ejercicios:

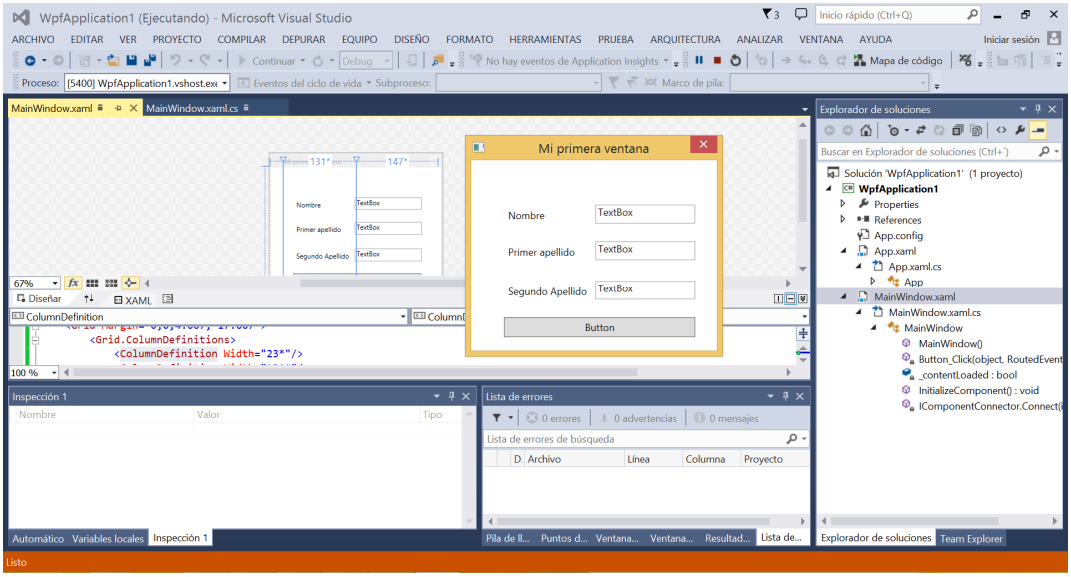
11) Repite el ejemplo anterior, pero en vez de emplear un *MessageBox* debes emplear el objeto/componente *Label*, es decir, modificarás el texto de dicho componente cuando se pulse un botón.

12) Ejecuta el siguiente código, bajo un evento click de botón y explica su funcionamiento:

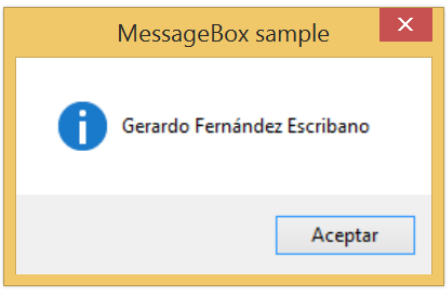
```
private void button1_Click(object sender, EventArgs e)
{
  String cadena;

  cadena = textbox1.Text;
  MessageBox.Show(cadena, "MessageBox sample", MessageBoxButton.OK,
                  MessageBoxImage.Information);

  cadena = "aprendiendo C#....";
  MessageBox.Show(cadena, "MessageBox sample", MessageBoxButton.OK,
                  MessageBoxImage.Information);}
```

Nota: analiza primero el código, pues puede ser que sea necesario algún componente más, además del botón.

10) Implementa un programa en el que, dada la entrada de datos que aparece en la ventana "Mi primera ventana":



Muestre, al pulsar sobre el botón, un mensaje informativo con tu nombre y tus dos apellidos concatenados de la forma:



Variables globales

Hasta ahora hemos visto cómo declarar variables locales dentro de cada evento (los cuales son funciones de C# en realidad). Por lo que, según lo aprendido hasta ahora en C#, dichas variables no son visibles entre eventos distintos (o funciones distintas). Entonces, ¿cómo podemos intercambiar valores entre eventos y/o funciones? La respuesta es la siguiente:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        int variable = 5;
        char caracter = 'c';
        String cadena = "la nieve es blanca";

        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

Es decir, dentro del fichero MainWindow.xaml.cs, se declararán (e inicializarán si fuera necesario), fuera de toda función o evento, preferentemente antes de la función que tiene el mismo nombre que el propio fichero, en este caso *MainWindow()*. Es más, podemos completar la explicación diciendo que dicha función es la función que será ejecutada en el momento en el que lancemos nuestra aplicación pues, en cierto modo, es la función principal del programa. También podemos utilizar esta función predefinida para la inicialización de variables (dicha función se conoce como *el constructor*).

Ejemplo: Escribe el siguiente código de Visual C#, dentro del archivo *MainWindow.xaml.cs*, añadiendo lo que falte. Deberás añadir los componentes y eventos que sean necesarios para

que tengas disponible el acceso a los mismos desde el código; según lo que hemos desarrollando a lo largo de la práctica.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        int variable = 5;
        char caracter = 'c';
        String cadena = "la nieve es blanca";

        public MainWindow()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            variable = variable + 1;
            label1.Content = variable.ToString();
        }
    }
}
```

Ejercicio:

11) Ejecuta el programa anterior y contesta cuál es la función de la instrucción *label1.Content = variable.ToString();*. **Nota**: las variables en C# también tienen ciertos métodos o funciones predefinidas a las que se puede acceder directamente, como se muestra en el ejercicio.

El ejercicio anterior muestra una diferencia sustancial entre la salida de texto por pantalla que empleábamos en C# y el empleo de C# para la visualización de información por pantalla; ya sea mediante mensajes interactivos (*Message.Show*) o mediante un componente *TextBox o Label*. Obligatoriamente, convertiremos el resultado a tipo *String* como se ha mostrado en el ejercicio anterior con la variable entera *variable*.

¿Ocurre lo mismo con la entrada de datos?

El elemento que utilizaremos para la entrada de datos es el componente *TextBox* que, recordemos, devuelve un valor de tipo *String*. ¿Cómo podemos convertir el valor recogido al aceptado por la variable que almacenará dicho valor? Con la función *Parse()*. Ejemplo:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        double variable = 5.7;
        int numero = 0;
        char caracter = 'c';
        String cadena = "la nieve es blanca";

        double suma_ocho(double dato)
        {
            return dato + 8.2;
        }

        public MainWindow()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            numero = int.Parse(textBox1.Text);
            label1.Content = numero.ToString();
        }
    }
}
```

Existen funciones homónimas tanto para las variables de tipo *float* como las de tipo *double*, *float.Parse(valor_o_variable_string)* y *double.Parse(valor_o_variable_string)* respectivamente. Además, habrá que tener cuidado, introduciendo exactamente el tipo de datos que se espera, de lo contrario el programa terminará; se produce un error en tiempo de ejecución.

6. Funciones en Visual C#.

De la misma forma que podemos declarar variables globales para compartir valores entre los eventos, podemos implementar nuestras propias funciones, escribiéndolas también dentro del código del archivo *MainWindow.xaml.cs*, pero fuera de las que ya haya predefinidas por el propio entorno de programación. Ejemplo: función *suma_ocho()*.

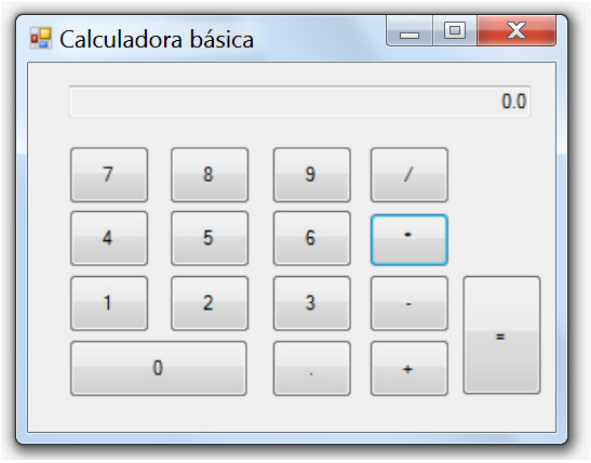
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace WpfApplication1
{
    /// <summary>
    /// Lógica de interacción para MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {

        int variable = 5;
        char caracter = 'c';
        String cadena = "la nieve es blanca";

        int suma_ocho(int dato)
        {
            return dato + 8;
        }

        public MainWindow()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, RoutedEventArgs e)
        {
            variable = suma_ocho(variable);
            label1.Content = variable.ToString();
        }
    }
}
```

Es decir, en la zona delimitada por *public partial class MainWindow : Window{*, y la llave que cierra dicho código (llave cerrada, *}*). Prueba el código anterior y comprueba qué es lo que aparece por pantalla.

Realmente, todo lo que programaremos en C# serán funciones, pues los propios eventos que añade el entorno de programación Visual Studio son funciones en realidad.

Ejercicio:

12) Escribe un programa que implemente una calculadora en lenguaje Visual C#. **Nota**: recuerda que el método *Clear()*, asociado a un componente de tipo *TextBox*, permite borrar su contenido. Ésta tendrá que realizar solamente operaciones básicas: suma, resta, multiplicación y división. Además de dichas operaciones y los números del 0 al 9, deberá incluir los símbolos de punto e igual. Los resultados se mostrarán mediante un componente *TextBox* (si lo consideras oportuno, añade un botón que limpie dicho componente *TextBox*). El aspecto visual deberá ser similar al siguiente:



No es necesario controlar excepciones, tales como límites de las operaciones o divisiones por cero, se comprobará su funcionamiento con valores que eviten dichas situaciones.