Future Developer Week

Javier San Juan Cervera

Contenido

Día 2: Controles visuales

- “¡Hola, mundo!” visual
- Eventos
- Funciones

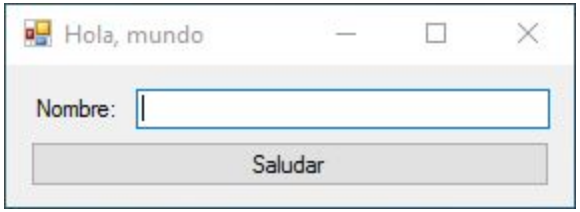
“¡Hola, mundo!” visual

Aplicaciones visuales

- En la última lección, creamos una aplicación en modo consola que saludaba al usuario por su nombre.
- Vamos a hacer una nueva versión de la aplicación para introducirnos en otro campo de la programación muy interesante: el desarrollo de aplicaciones visuales.
- Este tipo de aplicaciones, en lugar de interactuar con el usuario por medio de una consola de texto, utilizan componentes visuales como ventanas, menús y botones para realizar la interacción.

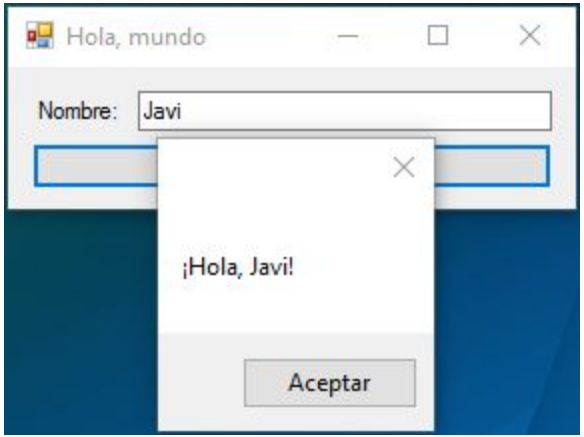
Aplicaciones visuales

Nuestra nueva aplicación “¡Hola, mundo!”, tendrá el siguiente aspecto:



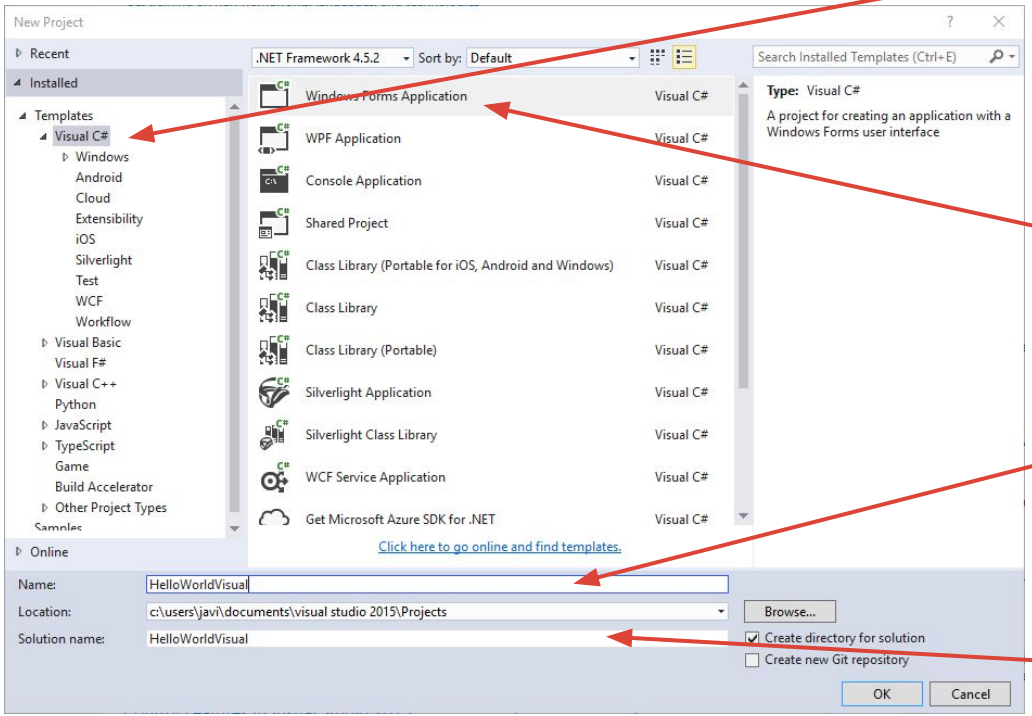
En la parte superior, se muestra una etiqueta y un campo de texto donde el usuario puede escribir su nombre. En la parte inferior, tenemos un botón que hará que se muestre un mensaje saludando al usuario:

Aplicaciones visuales



Pongámonos en marcha...

Creación del proyecto



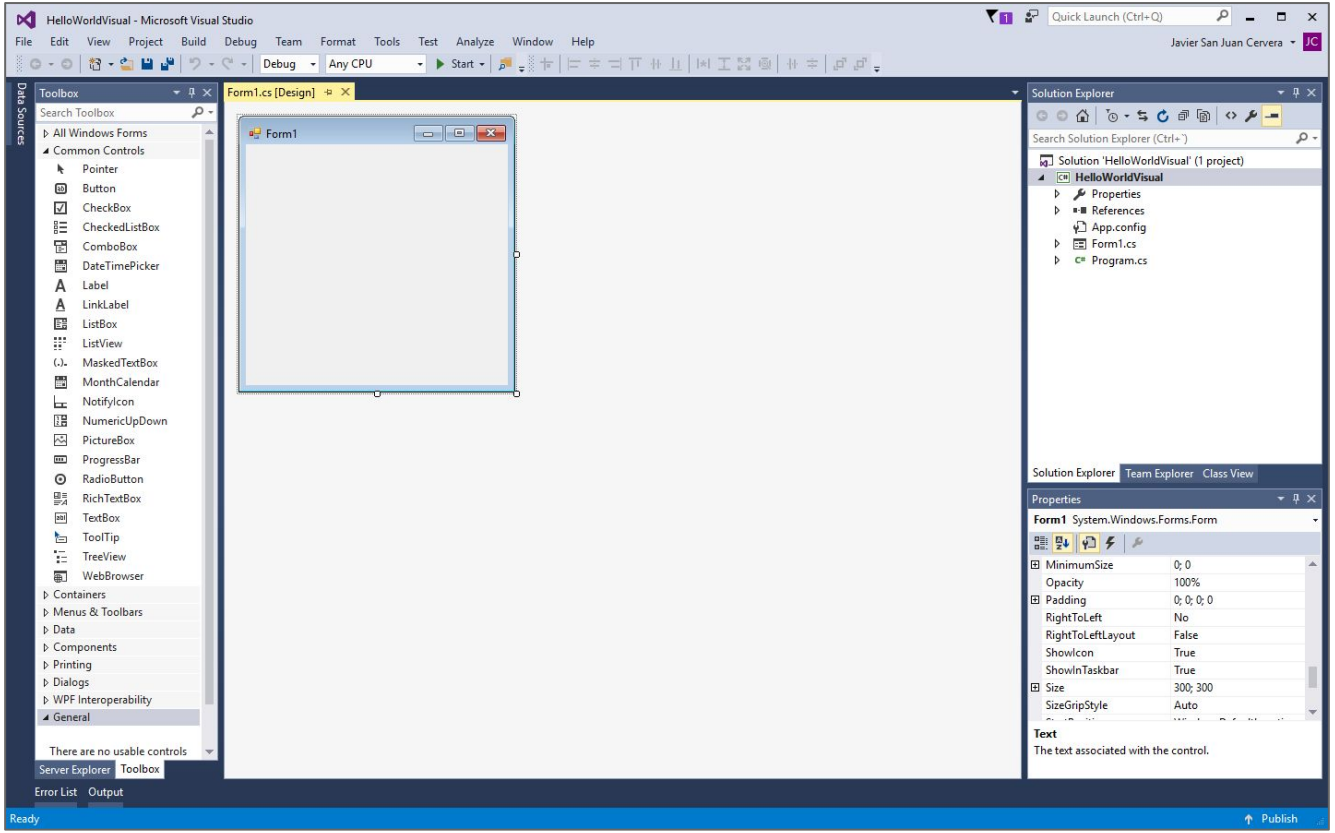
Lenguaje: Elegimos **Visual C#**

Tipo de aplicación: Esta vez escogemos **Windows Forms Application**

Nombre del proyecto

Ruta donde guardar el proyecto

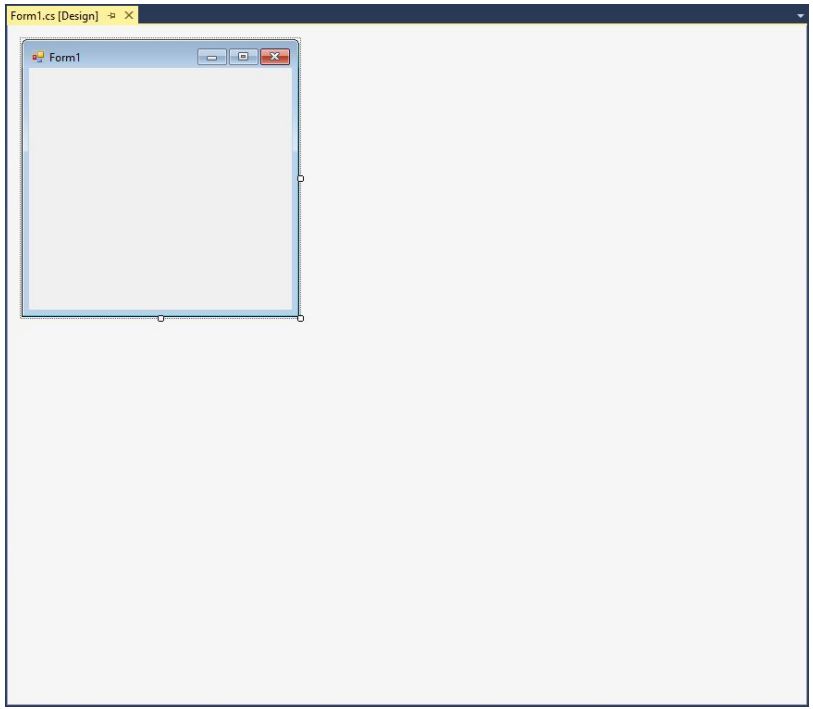
Creación del proyecto



Interfaz del entorno

- Al crear una aplicación visual, los paneles que se nos muestran en Visual Studio son diferentes a los de una aplicación de consola.
- Ya que en esta ocasión vamos a trabajar con las herramientas que nos ofrece el entorno de desarrollo, vamos a ver cada una de estas y la estructura general de un proyecto visual.

Interfaz del entorno



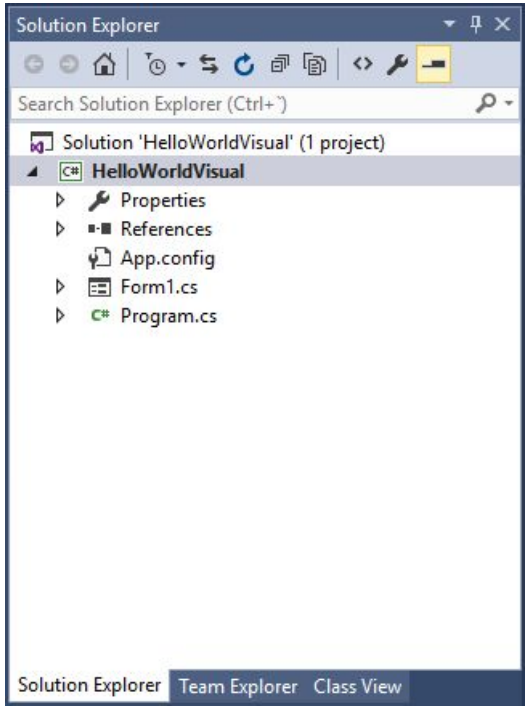
Editor:

- En el ejemplo de la consola, el editor mostraba el **código del programa**.
- En una aplicación visual, también se puede utilizar para editar el código, pero la pestaña abierta por defecto lleva la marca **[Design]**, indicando que la vamos a utilizar para **diseñar el aspecto** de nuestra ventana (llamada también **formulario**).
- Aquí añadiremos los botones, campos de texto, imágenes, etc, que contiene nuestro formulario.

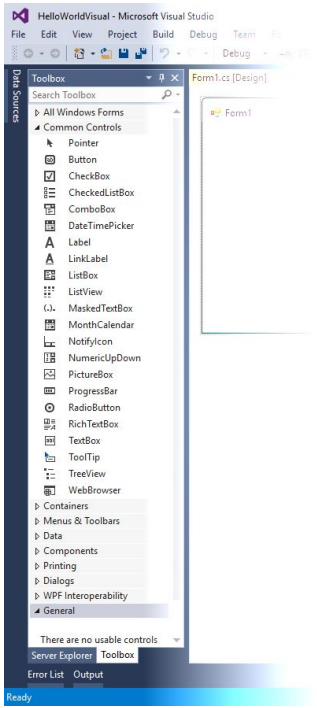
Interfaz del entorno

Solution Explorer (explorador de soluciones):

- Una solución contiene uno o varios proyectos, y éstos suelen contener varios archivos, carpetas, referencias, conexiones de datos…
- Los ficheros con extensión **.cs** representan los archivos de código escrito en lenguaje C#.
- Si no está visible en el entorno, podemos abrir el panel con la opción de menú **View / Solution Explorer**.



Interfaz del entorno



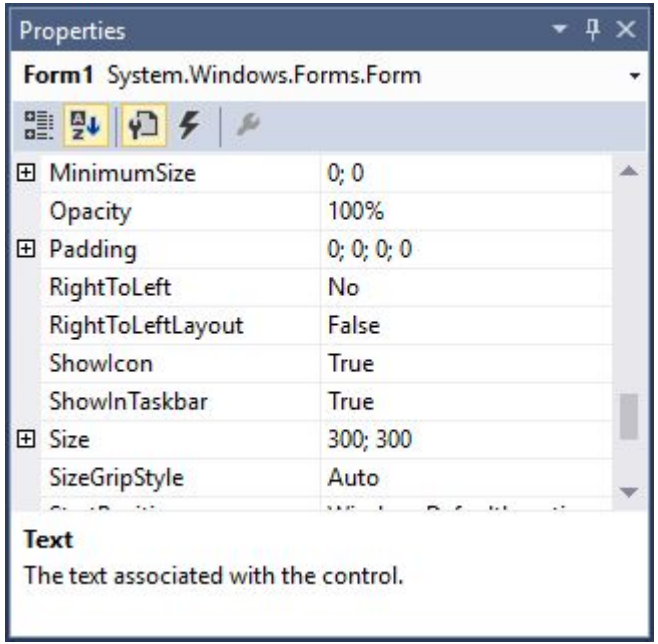
Toolbox (caja de herramientas):

- Contiene todos los controles que se pueden utilizar para crear nuestra interfaz de usuario.
- Podemos arrastrar los componentes desde la caja de herramientas a nuestro formulario.
- Si no está visible en el entorno, podemos abrir el panel con la opción de menú **View / Toolbox**.

Interfaz del entorno

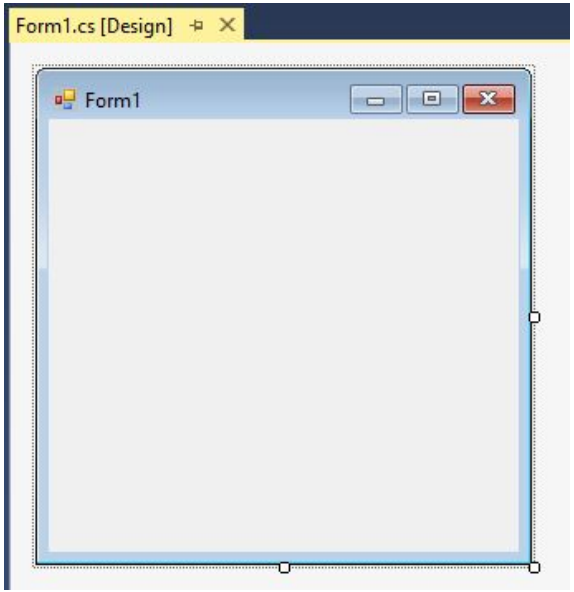
Properties (propiedades):

- Nos permite modificar las propiedades del control seleccionado en el formulario (o del propio formulario).
- Algunas de estas propiedades pueden ser el tamaño, texto, color del control, tipografía, etc.



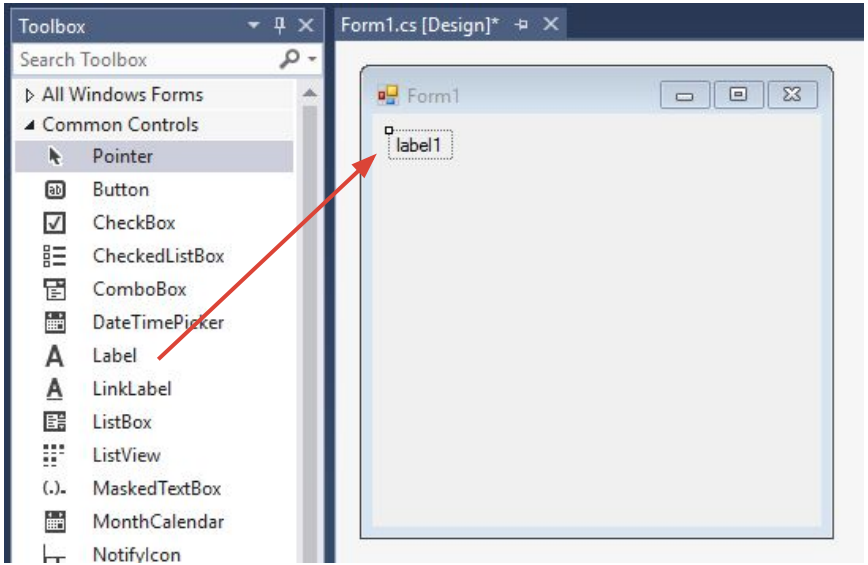
Interfaz de la aplicación

En la vista de diseño del editor, se nos muestra un formulario vacío:



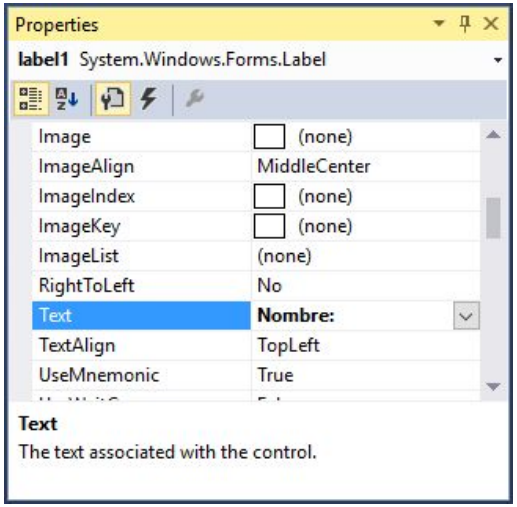
Interfaz de la aplicación

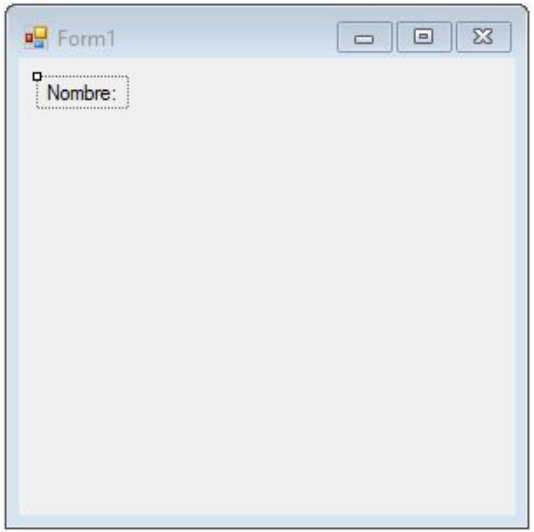
Desde la caja de herramientas (**Toolbox**), vamos a arrastrar al formulario una etiqueta (**Label**):



Interfaz de la aplicación

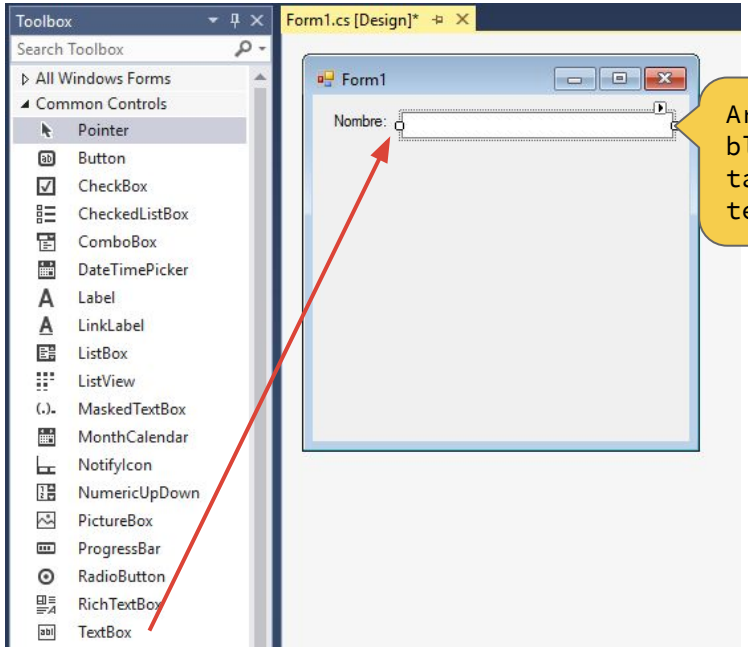
El **rectángulo punteado** alrededor de la etiqueta indica que es el control seleccionado, así que podemos cambiar sus propiedades con el panel **Properties**. Buscaremos la propiedad **Text** y cambiaremos su valor a **“Nombre:”**:





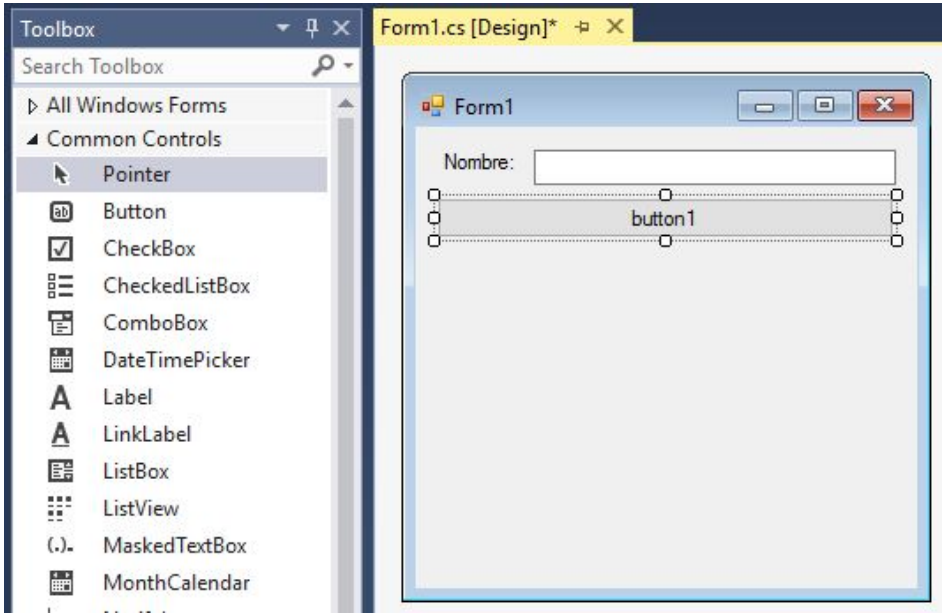
Interfaz de la aplicación

Ahora, desde el Toolbox arrastramos una caja de texto (**TextBox**) al formulario, justo a la derecha de la etiqueta:



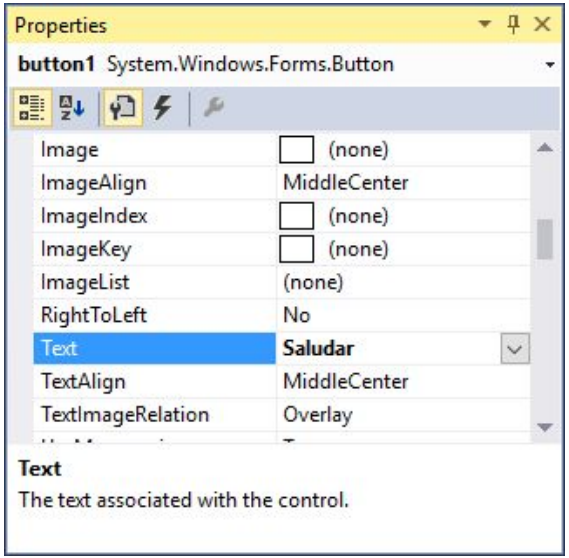
Interfaz de la aplicación

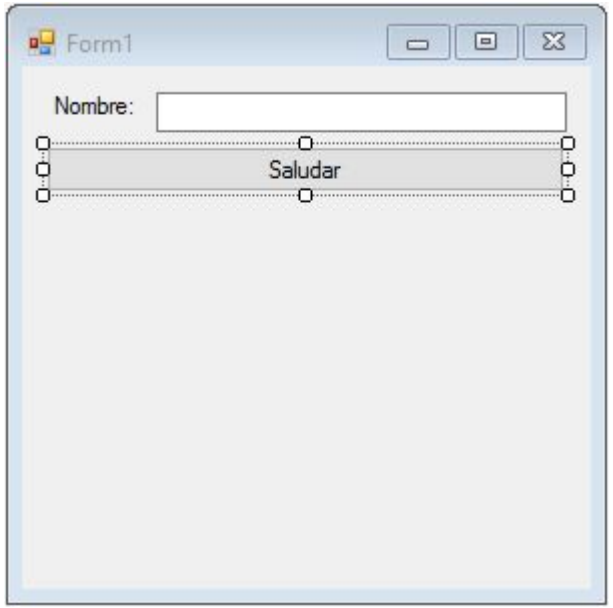
Después, colocamos un botón (**Button**) debajo de los otros dos controles, y lo agrandamos para que cubra toda la ventana en horizontal:



Interfaz de la aplicación

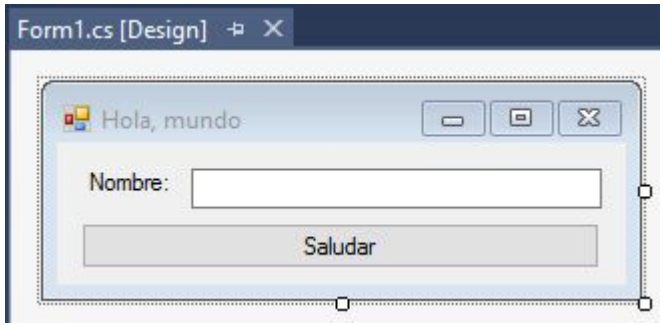
Cambiaremos en el panel **Properties** el valor de la propiedad **Text** del botón a **“Saludar”**:





Interfaz de la aplicación

Por último, seleccionamos el formulario haciendo **un solo click** sobre él (en la zona en la que no hay otros controles), reducimos su tamaño vertical para eliminar el espacio sobrante, y cambiamos su propiedad **Text** a **“Hola, mundo”**:



Eventos

Eventos

- Si ejecutamos la aplicación, veremos que se muestra la ventana con la interfaz, y que el programa continúa en ejecución hasta que se cierra la ventana, pero **no hará ninguna acción** más.
- Pero esto es lógico… ¡no hemos escrito ningún código!
- En la versión de consola, escribíamos una serie de instrucciones entre las llaves que se encontraban a continuación de la línea `static void Main(string[] args)`.
- En la versión gráfica, no escribimos código bajo esa línea (que está presente, podemos verla si hacemos doble click sobre el fichero **Program.cs** en el **Solution Explorer**, y veremos que ya viene con código escrito), sino que lo escribiremos en **respuesta a eventos**.

Eventos

- Cuando interactuamos con un control visual (por ejemplo, cuando **pulsamos un botón**), se produce un evento.
- Podemos escribir las sentencias que se deben ejecutar para responder a dicho evento.
- Cada control puede generar **varios eventos** (por ejemplo, al pulsarlo, arrastrar un archivo sobre él, mover el ratón…), aunque **cada tipo de control tiene un evento que se utiliza más habitualmente**.
- Si hacemos doble click en el editor sobre un control, se genera el evento más habitual para ese control.

Eventos

- Si hacemos **doble click** sobre el botón, se abrirá una nueva pestaña en el editor, con el título **Form1.cs**, pero sin la marca **[Design]**. Eso es porque ahora no estamos diseñando la interfaz para **Form1**, sino **definiendo su comportamiento mediante código**.
- Veremos la siguiente línea:

```
private void button1_Click(object sender, EventArgs e)
```

Eventos

- Esta línea representa el evento de hacer click sobre el botón llamado **button1** (que es un nombre que Visual Studio asignó al botón automáticamente al crearlo, podemos verlo o cambiarlo en la propiedad **(Name)** del botón en el panel Properties).
- Entre las llave de apertura y de cierre que se encuentran debajo, pondremos las sentencias que se deben ejecutar cuando se pulse el botón.

Eventos

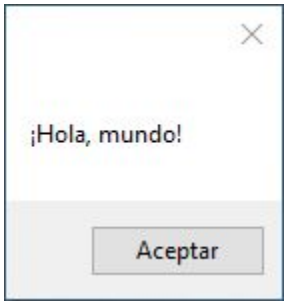
- Vamos a escribir la siguiente sentencia:

```
MessageBox.Show("¡Hola, mundo!");
```

- `MessageBox` es un tipo de control texto, botones, y un icono para dar una información al usuario.
- `Show` se utiliza para que el control se muestre, y la aplicación quedará a la espera de que pulsemos el botón **Aceptar** del control.

Eventos

Al ejecutar la aplicación y pulsar el botón, veremos el siguiente mensaje:



¿Cómo hacemos para que nos salude por nuestro nombre?

(NOTA: Si revisamos las propiedades del TextBox, veremos que su nombre es **textBox1**, y la propiedad que contiene el texto del botón se llama **Text**).

Funciones

Funciones

- Hemos estado escribiendo nuestro código entre las llaves que se encontraban debajo de unas determinadas líneas:

```
static void Main(string[] args)
private void button1_Click(object sender, EventArgs e)
```

- Éstas definen lo que se llama una **función**.
- Una función contiene una serie de **sentencias** que pueden ser **utilizadas en distintos momentos** en nuestro código. Por ejemplo, las sentencias de la función `button1_Click` son utilizadas cada vez que se pulsa sobre el botón.
- Hay que distinguir entre dos momentos:
  - Cuando **definimos** una función (se hace **una sola vez**).
  - Cuando **utilizamos** o **llamamos** a una función (se puede hacer **múltiples veces**).

Definición de funciones

Definición de funciones

- La definición de una función es la parte donde **indicamos ciertas características** de la misma, y la **lista de sentencias** que queremos que se ejecuten cuando la utilicemos.
- Las líneas que hemos mencionado en la diapositiva anterior comienzan la definición de la función. Tiene las siguientes partes:
  - Puede aparecer la palabra **private**.
  - Puede aparecer la palabra **static**.
  - Después, el **tipo** de valor que produce utilizar la función.
  - A continuación, el **nombre** que queremos darle a la función.
  - Tras éste, entre paréntesis y separados por comas, los **parámetros** de la función.
  - Por último, entre las llaves de apertura y cierre, las **sentencias**.

Definición de funciones

- Los dos primeros puntos los veremos más adelante, ahora sólo los mencionaremos de forma muy general.
- El primero, la palabra private, me permite definir la **visibilidad** de función. Puede ser **private**, **protected**, o **public**. Si no ponemos nada, será private. Permite indicar dónde puede ser utilizada la función.
- El segundo, permite definir el **contexto** de la función. Veremos el último día qué significa esto.

Definición de funciones

- El **tipo** se utiliza para indicar qué valor se produce o se **devuelve** al llamar a la función.
- Si ponemos **void**, no se produce ningún valor.
- Si ponemos otro tipo, se devolverá un valor de dicho tipo, y podremos utilizar la función **como una expresión**.
- Por ejemplo:

```
static void ImprimeHolaMundo()
{
	Console.WriteLine(“¡Hola, mundo!”);
}

static int DevuelveCinco()
{
	return 5;
}
```

Definición de funciones

- Para que la función devuelva un valor, se utiliza, como hemos visto en la diapositiva anterior, la sentencia **return**, seguida del valor o expresión a devolver.
- Cuando se ejecuta la sentencia return, la función **termina inmediatamente**. Es decir, si ponemos sentencias después, éstas no se ejecutarán.
- En las funciones void (es decir, que no devuelven valor), se puede utilizar return (sin poner seguido el valor) para terminar la ejecución del programa.

Definición de funciones

- Lo siguiente que aparece es el **nombre** de la función. Cuando queramos llamarla, nos referiremos a ella por su nombre.
- El nombre de la función tiene las mismas reglas que el nombre de una variable.

Definición de funciones

- Después, encontramos los **parámetros** entre paréntesis.
- Los parámetros son simplemente variables.
- Cuando llamamos a la función, le indicaremos qué valores tomarán esos parámetros.
- Si hay **varios** parámetros, se ponen **separados por coma**.
- Si **no hay** parámetros, se ponen **sólo los paréntesis**.

Definición de funciones

- Terminamos escribiendo la lista de sentencias que queremos que se ejecuten al llamar a la función. Estas sentencias pueden utilizar los parámetros como cualquier otra variable:

```
static int ObtenerMedia(int a, int b)
{
    int suma = a + b;
    return suma / 2;
}
```

- ¿Por qué esta función no devuelve los decimales de la media?
- ¿Cómo podemos solucionarlo?

Llamadas a funciones

Llamadas a funciones

- Cuando queramos ejecutar las sentencias contenidas en una función, debemos **llamarla**.
- En su versión más simple, la llamada a la función se realiza poniendo el nombre de la función seguido de los valores que queremos pasar a cada parámetro entre paréntesis y separados por coma:

```
ObtenerMedia(2, 4);
```

- Los argumentos (valores de los parámetros) se asignan por orden.

Llamadas a funciones

- En muchos casos, es necesario poner, antes del nombre de la función, y separado de éste por un **punto**, el **contexto** en el que se definió la misma:

```
Console.WriteLine(“¡Hola, mundo!”);
```

- En este caso, la función `WriteLine` está definida en la consola, que en C# se llama `Console`.
- El contexto podría ser, por ejemplo, un control (como un botón), en cuyo caso pondríamos su nombre antes del punto.
- Aunque ahondaremos más en esto al final de la semana, por el momento indicaremos el contexto si la función se definió en un fichero diferente al que estamos editando.