

Por Nacho Iborra

# Entornos de Desarrollo

## Bloque 3

### Tema 5: Introducción a los entornos gráficos. Windows Forms

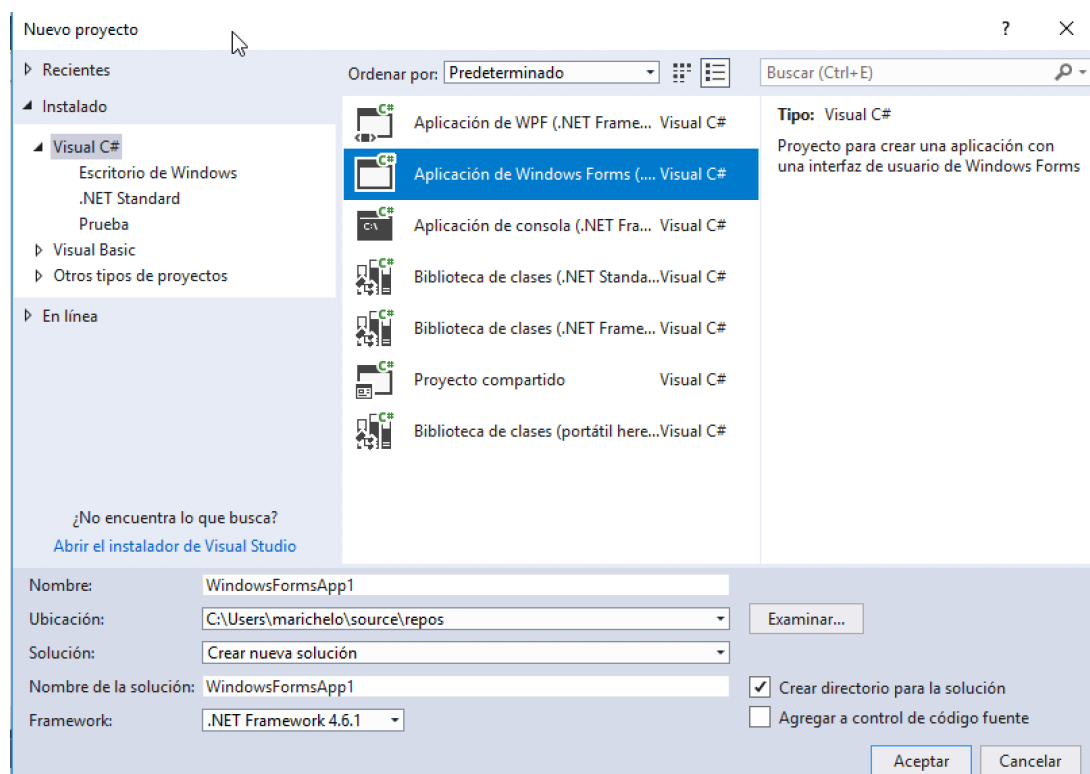
#### 3.5.1. Introducción a Windows Forms

Ahora que conocemos los conceptos básicos de programación en C# (estructuras de control, arrays, colecciones, entrada/salida, programación orientada a objetos...) vamos a ir un paso más allá. No solo vamos a trabajar con aplicaciones de consola que a veces son difíciles de utilizar, sino que vamos a ver como crear una aplicación gráfica basada en Windows.

La programación gráfica en C# consiste en desarrollar Interfaces gráficas de usuario (*Graphical User Interfaces* GUI). Estas aplicaciones se basan en formularios, o sea, en ventanas que tienen un conjunto de controles (menús, botones, campos de texto, listas...) Para desarrollar este tipo de aplicaciones con Visual Studio debemos usar **Windows Forms**, una librería integrada en Visual Studio.

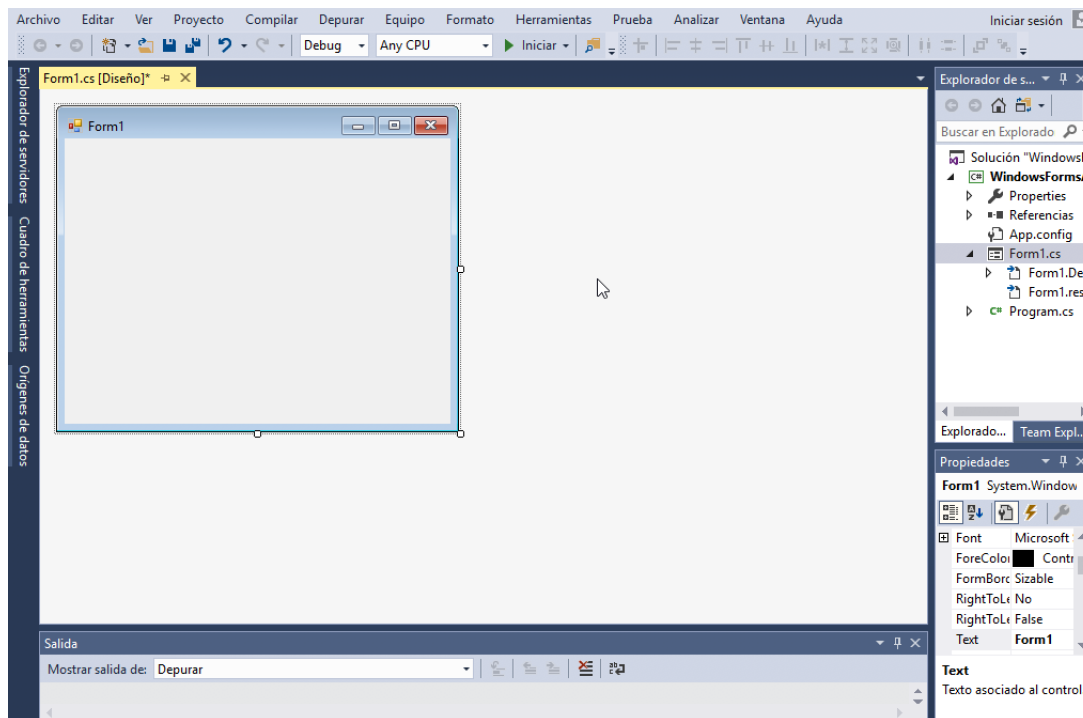
##### 3.5.1.1. Crear un proyecto

Para crear un proyecto de Windows solo necesitamos ir al menú Archivo > Nuevo > Proyecto y elegir Aplicación de Windows Forms.



Cuando creamos el proyecto se crea a su vez un programa principal llamado Program.cs además de un formulario controlado por la clase Form1.cs (aunque podemos renombrarla como cualquier clase en C# pulsando el botón derecho del ratón sobre ella y eligiendo la opción de renombrar)

En la zona de trabajo principal (a la izquierda de la parte central) podemos ver este formulario y editarlo con un interfaz gráfico como se puede ver en la imagen siguiente.

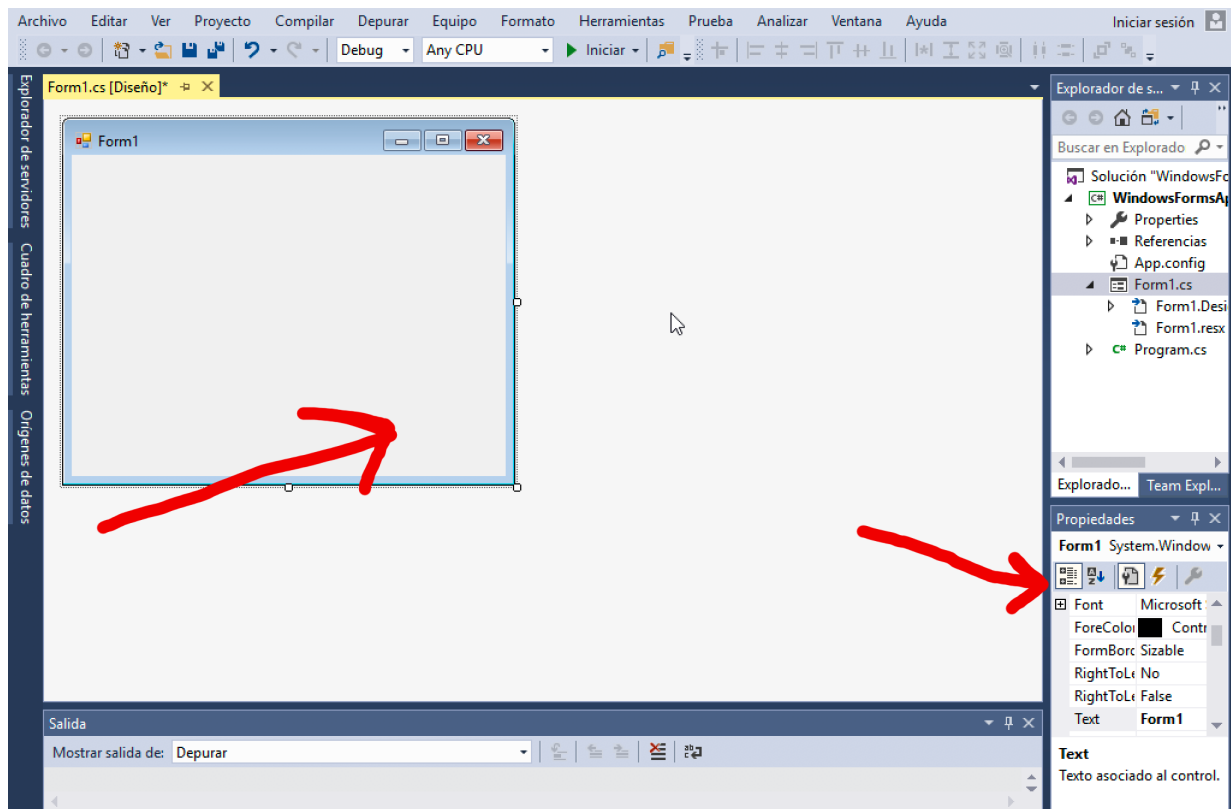


### 3.5.2. Controles y propiedades

Una vez que nuestro formulario principal está creado, podemos editarlo, añadirle controles como botones, etiquetas, menus... y cambiar algunas de sus propiedades como color, tipos de letra...

#### 3.5.2.1. Propiedades de formulario

Si pulsamos con el botón izquierdo en el área de trabajo del formulario principal, podemos ver sus propiedades en la parte inferior derecha (llamada la sección *Propiedades*)

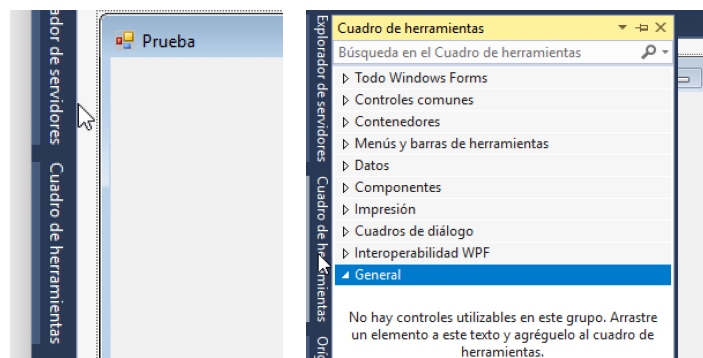


Algunas de las propiedades más interesantes son:

- **Text**, el que aparece en la barra superior del formulario
- **BackColor**, el color de fondo
- **BackgroundImage**, por si quisiéramos poner una imagen de fondo a nuestro formulario
- **ForeColor**, Color del texto
- **MaximumSize** y **MinimumSize** valores permitidos para la ventana
- **Size**, valor actual del tamaño que debe estar entre el máximo y mínimo definido con las propiedades anteriores
- **StartPosition**: Posición inicial en la pantalla
- **WindowState** para establecer si la ventana principal se iniciará minimizada, maximizada y normal.

### 3.5.2.2. Algunos controles típicos

Para editar los controles accedemos a las herramientas de control haciendo clic sobre la sección cuadro de herramientas en la esquina superior izquierda.



Alguno de los controles típicos en aplicaciones de Windows Forms son:

- **Button**: para definir botones de acción, como Aceptar, Cancelar, Guardar...
- **Checkbox**: para definir controles de marcar y desmarcar opciones
- **ComboBox**: para definir listas desplegables
- **Label**: para añadir texto explicativo junto a algunos controles
- **ListBox**: para mostrar un conjunto de elementos en una lista con un tamaño fijo (no una lista desplegable)
- **PictureBox**: para añadir imágenes a nuestra aplicación
- **RadioButton**: para definir un conjunto de opciones donde se puede elegir una sola de ellas al mismo tiempo
- **TextBox**: para añadir un texto tanto corto como multilinea

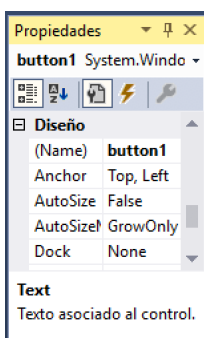
Existen además muchos más controles adicionales que se pueden usar en nuestra aplicación. Para usar alguno de estos controles se debe seleccionar en la caja de herramientas y arrastrarlo al formulario en el lugar donde queramos que aparezca. Al colocar nuevos controles, si ya existieran otros en el formulario aparecerán unas guías para ayudar a alinearlos con el resto de controles.

#### 3.5.2.2.1. Propiedades de los controles

Como hemos visto para el formulario principal, hay algunas propiedades interesantes también de los controles. Algunas de ellas son las mismas explicadas con anterioridad tales como: color de fondo, color del texto, texto, tamaño... Pero hay otras propiedades más específicas como: **Visible** (hace el control visible o no), **Enabled** (habilita o deshabilita el control)...

#### 3.5.2.2.2. Convenciones en los nombres

Existe una propiedad realmente importante que es **Name** que corresponde con el nombre. Es la primera propiedad que aparece en la lista de propiedades de diseño. Esta propiedad nos permite definir un nombre de variable para el control de forma que podamos referenciarlo desde nuestro código C#.



Aunque se puede usar cualquier nombre válido para los controles, hay algunas convenciones que se deben tener en cuenta. Dependiendo del tipo de control existen unos prefijos que se suelen poner al nombre del control. Algunos para los controles más utilizados serían:

- **btn** para botones
- **txt** para cuadros de texto
- **lbl** para etiquetas

- *chk* para el control *checkbox*
- *cmb* para listas desplegables
- *lst* para listas

### 3.5.2.3. Contenedores

Los contenedores son un conjunto especial de controles que ayudan a agrupar un subconjunto de controles y tratarlos como un todo: podemos moverlos como una unidad, dibujarles un borde... Algunos de los más utilizados son:

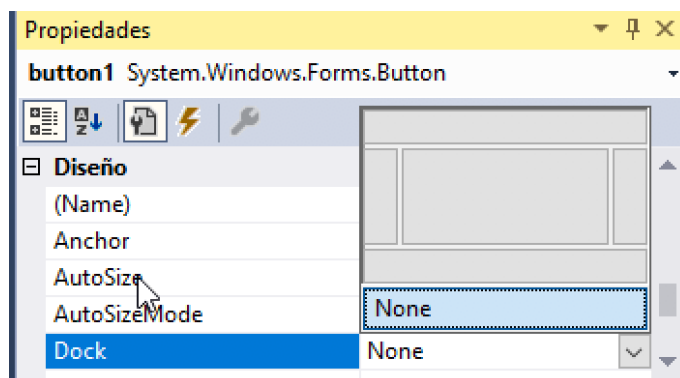
- **FlowLayoutPanel:** los controles situados dentro de este contenedor se colocan de izquierda a derecha y de arriba a abajo.
- **GroupBox:** este contenedor se puede usar para simplemente agrupar controles y colocarles un borde y/o título.
- **Panel:** un panel simple que agrupa controles y se puede mover como una unidad.
- **SplitContainer:** se crean dos zonas dentro del contenedor
- **TabControl:** para crear un conjunto de pestañas.
- ...

Estos contenedores tienen a su vez propiedades que son más o menos las mismas que los controles (colores, texto, tamaño...)

### 3.5.2.4. Colocar controles y contenedores. Propiedades *Dock* y *Anchor*

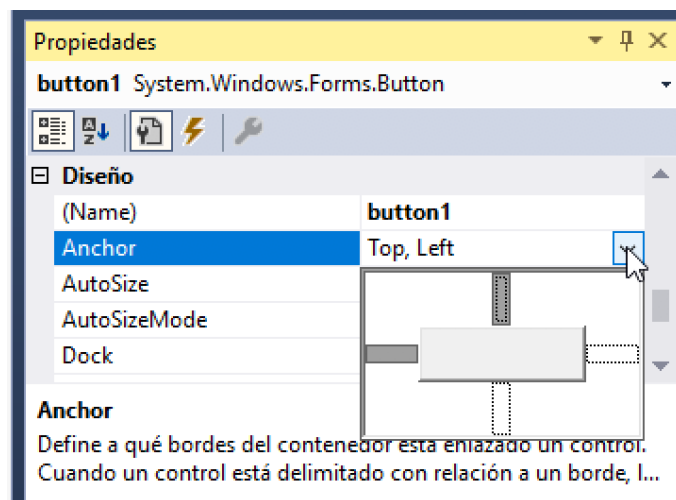
Si se coloca un contenedor o control dentro del formulario podemos encontrarnos problemas al redimensionar la ventana, puede darse el caso de que la posición o el tamaño de los controles no sean compatibles con el nuevo tamaño de la ventana. Para evitar este problema existen dos propiedades en los controles y contenedores.

**Dock:** Especifica en que parte de la ventana se añadirá el control o contenedor



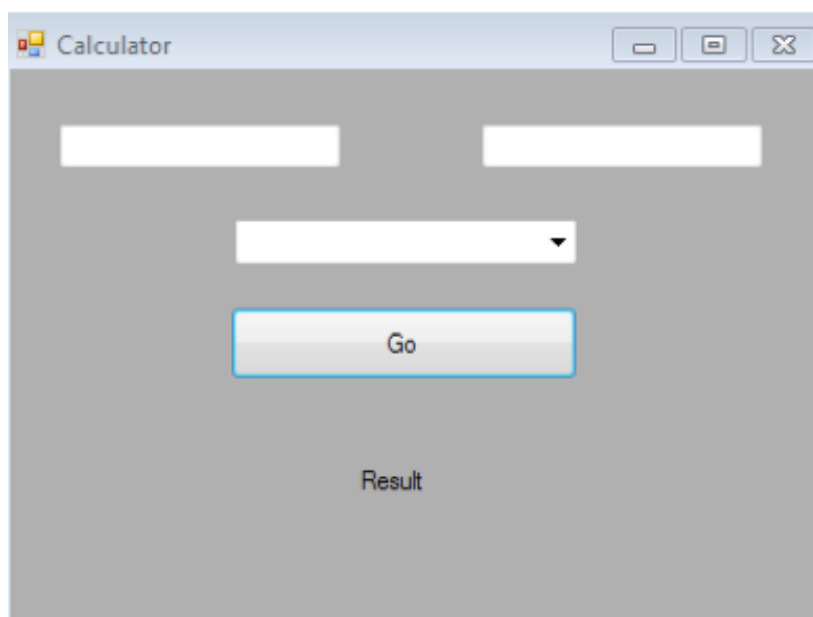
De esta forma si se redimensiona la ventana principal, el control o controlador se quedará fijado en la zona especificada.

**Anchor:** permite determinar los bordes para cada control. Con esto se consigue que el control o contenedor mantenga la misma distancia desde los bordes de referencia aunque se redimensione la ventana.



### 3.5.2.5. Un proyecto sencillo: Una calculadora

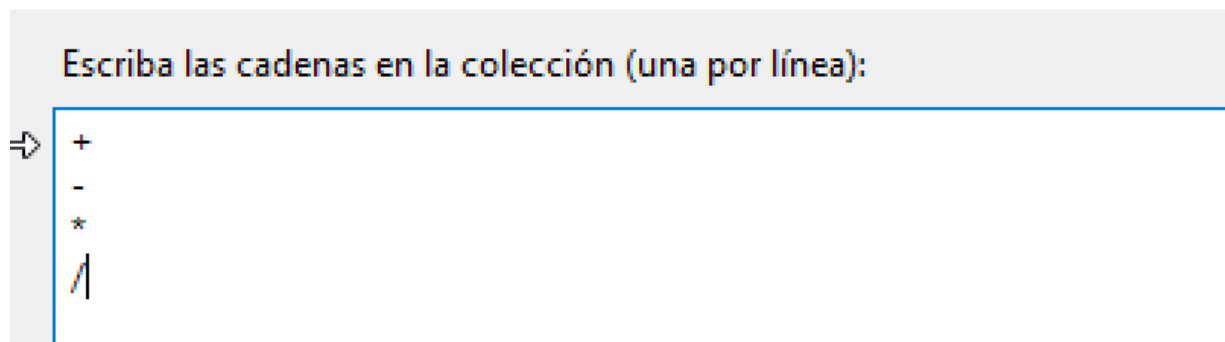
Vamos a crear una aplicación de Windows Form llamada *Calculator*. Usaremos dos cuadro de texto, una lista desplegable, un botón y una etiqueta y los colocamos en el formulario para que quede algo así:



Lo primero que debemos hacer es cambiar algunas propiedades:

- Establecemos el título del formulario como *Calculator* (text)
- Establecemos el color de fondo como *DarkGray*
- Establecemos el texto del botón como *Go*
- Establecemos el texto de la etiqueta como *Result* y el fuente y tamaño con el valor adecuado para que se vea bien.
- Establecemos la propiedad *Items* de la lista desplegable con las siguientes cadenas:

## Editor de colección de cadenas



Ahora establezcamos los nombres de los controles:

- El cuadro de texto de la izquierda será *txtNumber1*
- El cuadro de texto de la derecha será *txtNumber2*
- La lista desplegable será *cmbOperator*
- El botón será *btnGo*
- La etiqueta será *lblResult*

A continuación cambiemos las propiedades de anclaje (*Anchor*)

- Anclamos el cuadro de texto de la izquierda a los bordes de arriba y la izquierda.
- Anclamos el cuadro de texto de la derecha a los bordes de arriba y la derecha.
- Anclamos la lista desplegable y el botón a los bordes de arriba, la izquierda y la derecha.
- Anclamos la etiqueta a los bordes de abajo, la izquierda y la derecha.

Por último especificamos el tamaño del formulario:

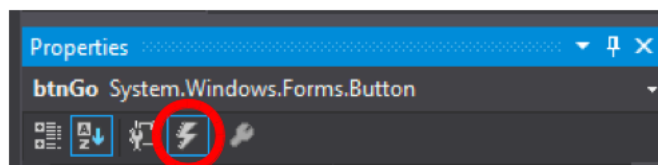
- Definimos el tamaño por defecto (inicial) a 400 x 300
- Definimos un tamaño mínimo de 300 x 200 y un máximo de 500 x 500

Prueba a ejecutar la aplicación y a redimensionar la ventana. De momento la aplicación todavía no hace nada, lo hará cuando se le añadan algunos eventos como se explica en el siguiente punto.

### 3.5.3. Eventos

Los eventos son lo que hace que una aplicación haga algo. Existen bloques de código que se ejecutan cuando algo ocurre. Por ejemplo, si se quiere guardar datos cuando se pulse un botón de **Guardar**, necesitamos definir el evento *click* de ese botón que ejecutará un método con las instrucciones para guardar los datos correspondientes.

Para definir eventos en cada control o contenedor (o formulario), hay una pestaña específica en el panel de propiedades (la pestaña del relámpago)



Si pulsamos sobre ella, podemos ver una lista de los eventos disponibles para el control seleccionado. Algunos de los más útiles son **Click** o **MouseClick** (se lanzan cuando se hace clic sobre el control), **KeyPress** (se lanza cuando se pulsa cualquier tecla en el control), **TextChanged** (para cuadros de texto por ejemplo se lanza cuando el texto de dentro del cuadro cambia).

#### 3.5.3.1. Código para un evento

Haciendo doble clic sobre un evento de la lista, se abre una página de código. Si nos fijamos en el nombre del fichero se llama igual que el formulario pero sin el sufijo (Diseño).



Aparece además un método que representa el evento que hemos seleccionado ( en este caso el hacer clic en el botón). Dentro de este método podemos añadir el código que queremos para ese evento. Incluso podemos referenciar a cualquier otro control de nuestra aplicación, siempre que le hayamos puesto un nombre. Todos los controles con sus nombres y propiedades automáticamente se almacenan en el fichero **Form.Designer.cs**. Este fichero se puede editar aunque no es necesario ya que todo lo que se modifica por la ventana de diseño se modifica en el método **InitializeComponent** que está dentro de este fichero.

Siempre que hagamos doble clic en un evento, Visual Studio añade una nueva línea en el **Form.Designer.cs** indicando la relación entre el control y el evento. En el ejemplo anterior se añadiría una línea como esta:

```
this.btnGo.Click += new System.EventHandler(this.btnGo_Click);
```

##### 3.5.3.1.1. Eventos por defecto

Cada control tiene un evento por defecto. Por ejemplo, el de un botón es el evento **Click**. De esta forma si hacemos doble clic sobre un botón o hacemos doble clic sobre el evento **Click** del botón accedemos al mismo lugar: a un método **Click** vacío que hay que rellenar con nuestro código.

##### 3.5.3.1.2. Eliminar un evento

¿Y si cometemos un error y no queremos implementar un evento que ya hemos creado? Necesitamos seguir los siguientes pasos:

1. Eliminar el método del evento del fichero **Form1.cs** (o el nombre que tenga en el proyecto).
2. Eliminar del fichero **Form.Designer.cs** la línea de código que une el control con el evento.

#### 3.5.3.2. Definir los eventos de la calculadora de ejemplo



Vamos a definir el evento **Click** del botón **Go** como se indica en el ejemplo anterior y dentro del método realizamos lo siguiente:

- Almacenamos los valores de los dos cuadros de texto en dos variables.
- Revisamos el valor seleccionado en la lista desplegable
- En función del valor seleccionado, realizamos la operación matemática correspondiente y mostramos por pantalla el resultado en la etiqueta.

El código sería algo así:

```
private void btnGo_Click(object sender, EventArgs e)
{
    int number1 = Convert.ToInt32(txtNumber1.Text);
    int number2 = Convert.ToInt32(txtNumber2.Text);
    string selOperator = (string)(cmbOperator.SelectedItem);
    switch (selOperator)
    {
        case "+":
            lblResult.Text = "" + (number1 + number2);
            break;
        case "-":
            lblResult.Text = "" + (number1 - number2);
            break;
        case "*":
            lblResult.Text = "" + (number1 * number2);
            break;
        case "/":
            lblResult.Text = "" + (number1 / number2);
            break;
    }
}
```

Pruébalo ahora y verás como funciona el evento que hemos realizado

### 3.5.3.3. Mostrar mensajes de error

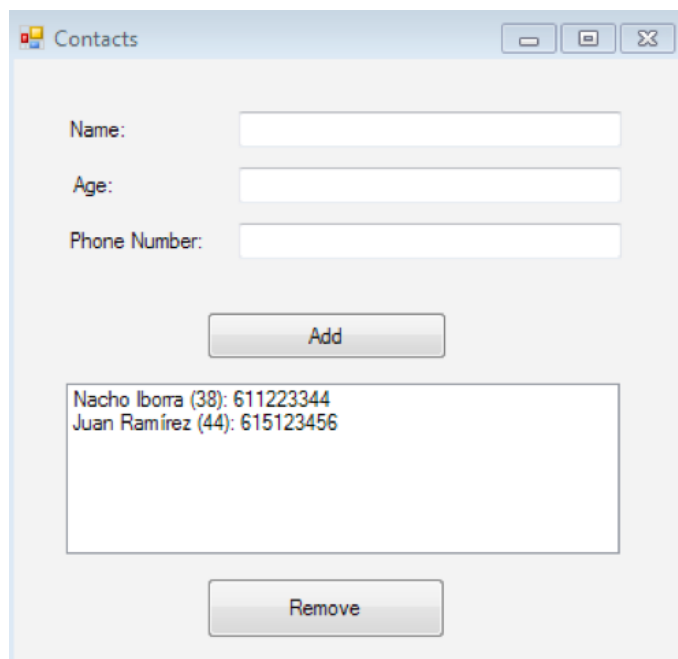
Para mostrar mensajes de alerta al usuario ya no se puede hacer mediante la consola (recordemos que no estamos en una aplicación de consola), necesitamos mostrar dicho mensaje en una ventana. Para ello podemos usar el método **MessageBox.Show**. Por ejemplo para mostrar un mensaje de error sería:

```
MessageBox.Show("No operator is selected");
```

El método **MessageBox.Show** está sobrecargado de forma que se puede usar con más parámetros si fuera necesario, especificando el mensaje, icono y otras propiedades. Puedes aprender sobre esto en [MessageBox.Show official documentation](#)

**Ejercicio propuesto:**

**3.5.3.1.** Para practicar con todos los conceptos aprendidos en este tema, crearemos una aplicación de Windows Forms llamada **Contacts**, con el siguiente aspecto:



La aplicación debe tener:

- Tres etiquetas y cuadros de texto en la sección superior.
- Un botón **Add** debajo de ellos
- Una lista debajo del botón **Add**
- Un botón **Eliminar** debajo de la lista

Siempre que se rellenen los 3 cuadros de texto y se haga clic sobre el botón **Add**, se creará un nuevo contacto (una clase dentro del proyecto con tres atributos: name (string), age (entero) y phone(string)). Cuando hagamos clic sobre el botón **Add**, se revisará si hay algún campo vacío (sacando un mensaje de error en su caso). Si todo es correcto un nuevo objeto **Contacto** se creará y se añadirá a la lista.

Para que los contactos se muestren en la lista como se ven en la imagen anterior, necesitarán sobrescribir el método **ToString** de la clase **Contact**, para que se muestre el nombre, la edad entre paréntesis y el número de teléfono.

Por último, si hacemos clic en el botón **Remove**, borraremos el ítem seleccionado en la lista (si hay alguno seleccionado).

Para finalizar el proyecto, almacena datos en un fichero de texto con un formato determinado (por ejemplo separando cada dato con ';'). Debemos reescribir este fichero siempre que añadamos o borremos de nuestra lista.