Programación Avanzada de entornos de escritorio

2021-2022

Escuela Politécnica Superior

Universidad de Alicante

Descripción del taller

- Crearemos una aplicación sencilla con Windows Forms

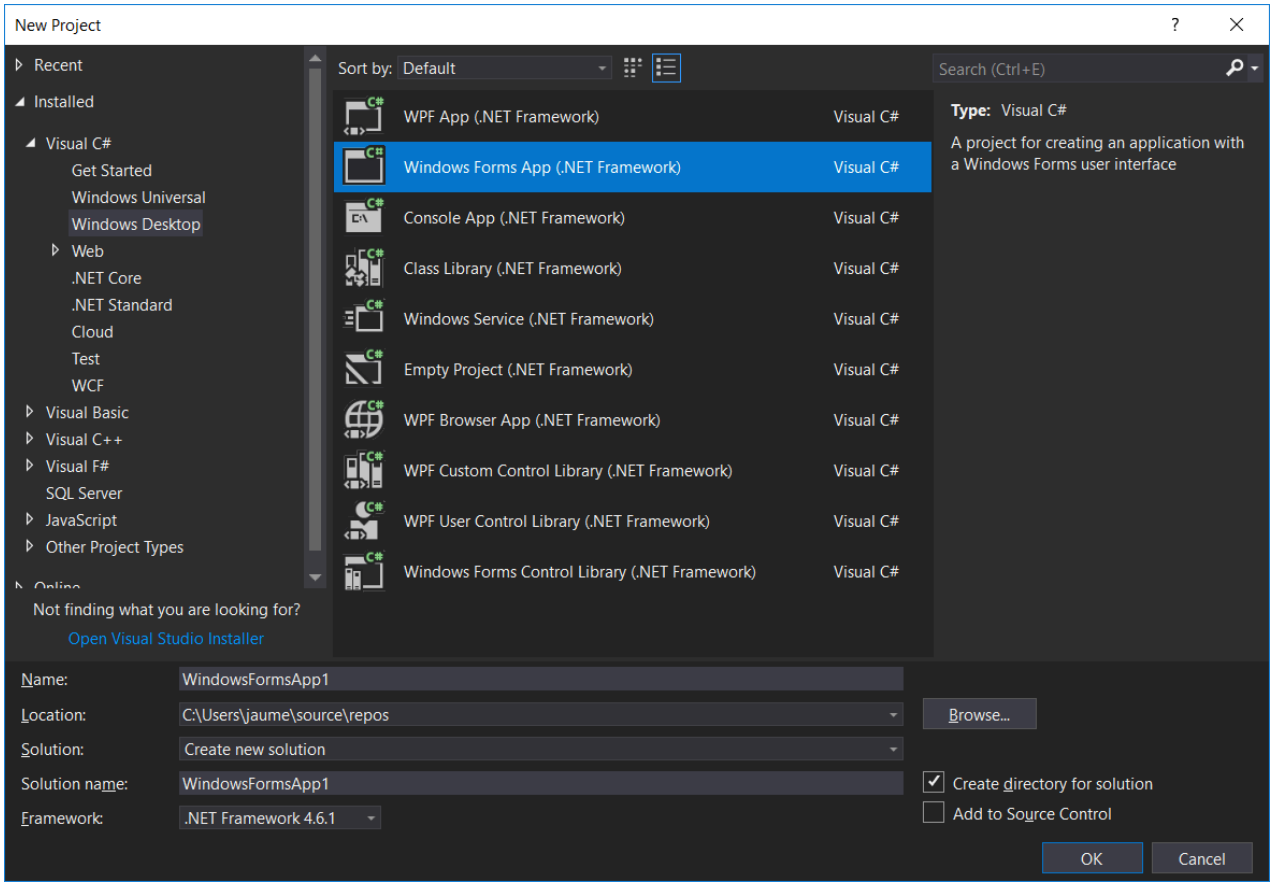
Índice

NOTA:

- Se ha subido a Moodle las versiones parciales de la aplicación realizada para optimizar el trabajo en clase

- El código fuente en ocasiones no es el más óptimo: en general se prima la pedagogía cada contexto a la optimización

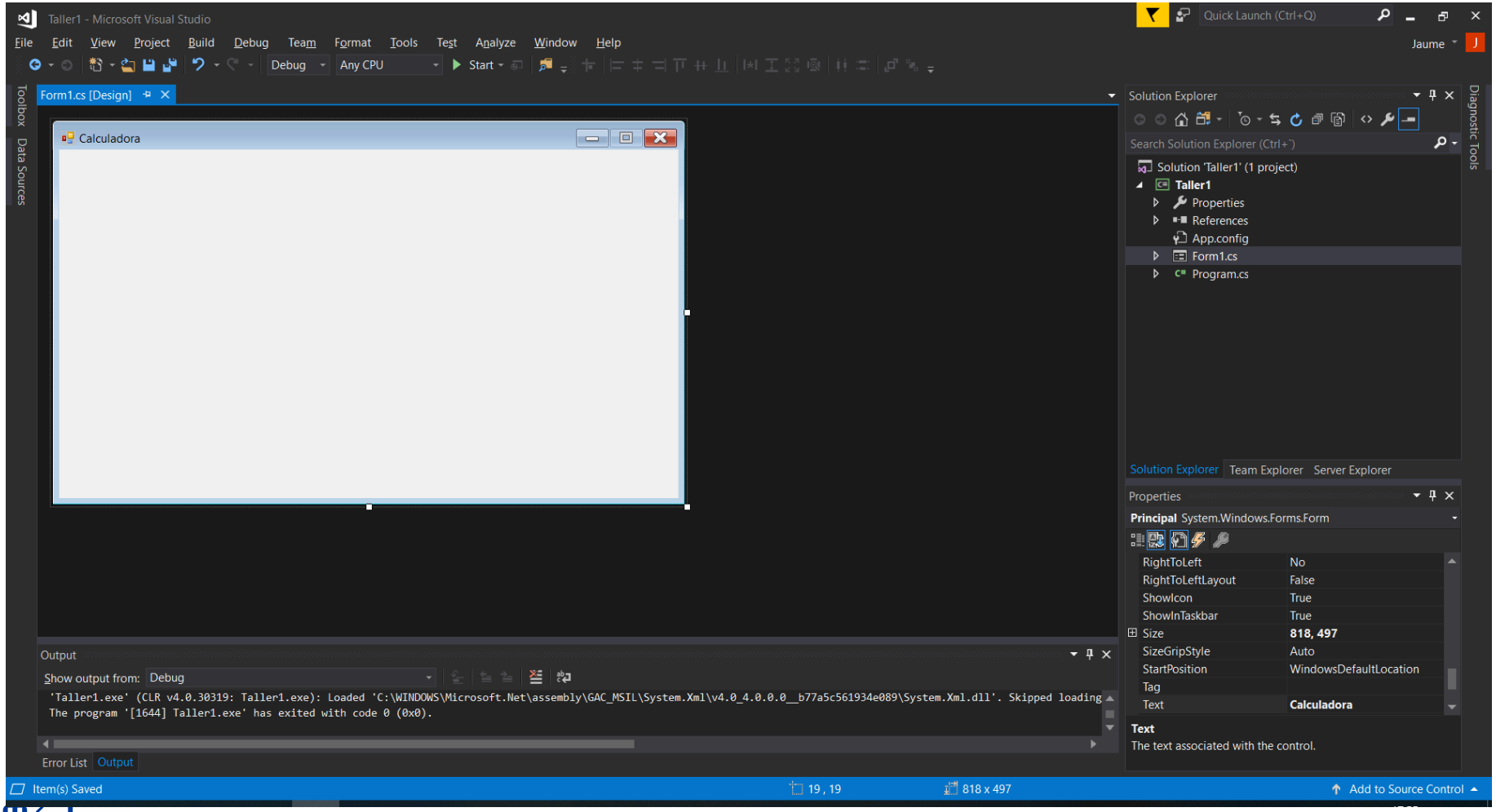
Crear el proyecto



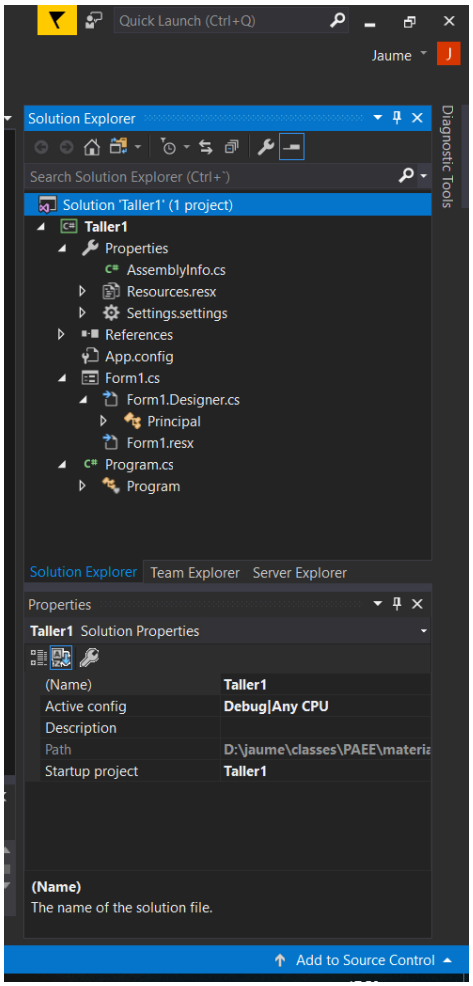
Diálogo de nuevo proyecto:

1. Seleccionamos proyecto de Visual C#, Windows Desktop, Windows Forms App.
2. Le damos un nombre al proyecto.
3. Escogemos su ubicación
4. Seleccionamos la versión del Framework.

El entorno del nuevo proyecto



El entorno del nuevo proyecto



- Ficheros del proyecto:
- 1.Propiedades
- 1.AssemblyInfo: Información de ensamblado
- 2.Recursos globales
- 3.Settings: Fichero de configuración de la aplicación.
- 2.Referencias: referencias a APIs que tiene el proyecto.
- 3.App.config: fichero XML en formato 'config' con directivas de configuración del programa
- 4.Form1.cs: el componente formulario por defecto.
- 1.Dos vistas: Diseño y Código (code behind)
- 2.Designer: inicializaciones y gestión de los controles del form.
- 3.Recursos locales
- 5.Clase Program.cs: programa principal.

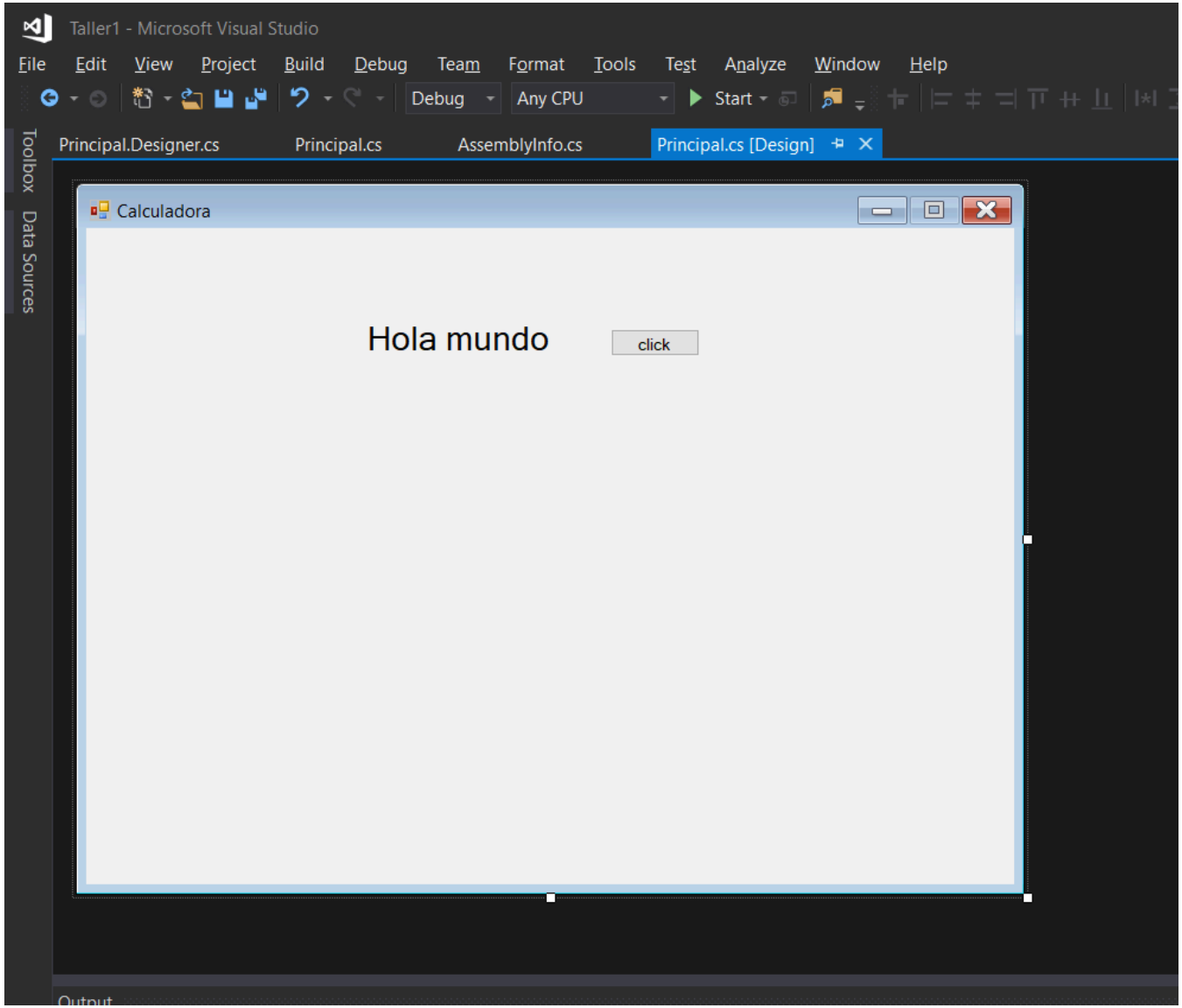
Personalizar el proyecto

- Se puede trabajar con los nombres que el entorno da a los objetos por defecto (form1, button1, etc.)
- o bien, se puede personalizar el código con nombres y valores por defecto más apropiados (**recomendable**)
- Por tanto, vamos a personalizar el proyecto con unos cambios iniciales:
  - Renombrar el fichero 'Form1.cs' por 'principal.cs'
  - Modificar las propiedades del formulario principal: name (Principal), StartPosition, Size (800, 600), Text (Calculadora), MaximizeBox(false), etc.
  - Otras personalizaciones: colores, efectos, icono, tipografía, etc.

“Hola mundo”

- Vamos a probar el funcionamiento de los controles en un formulario:
  - En este ejemplo no vamos a renombrar los objetos, pues son sólo para probar y no los usaremos en el resto del taller.
  - Añadimos una etiqueta (label) y le damos valor a su propiedad 'Text': “Hola mundo”.
  - Agregamos un botón al lado, y le ponemos ‘Click' como texto.
  - Hacemos doble click sobre el botón para acceder a editar el manejador del evento 'click' sobre él mismo.

“Hola mundo”


Taller1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Format Tools Test Analyze Window Help
Debug
Any CPU
Start
Principal.Designer.cs
Principal.cs
AssemblyInfo.cs
Principal.cs [Design]
Toolbox
Data Sources
Calculadora
Hola mundo
click
Output

“Hola mundo”

- Añadimos código tal que la clase 'Principal' quedará:

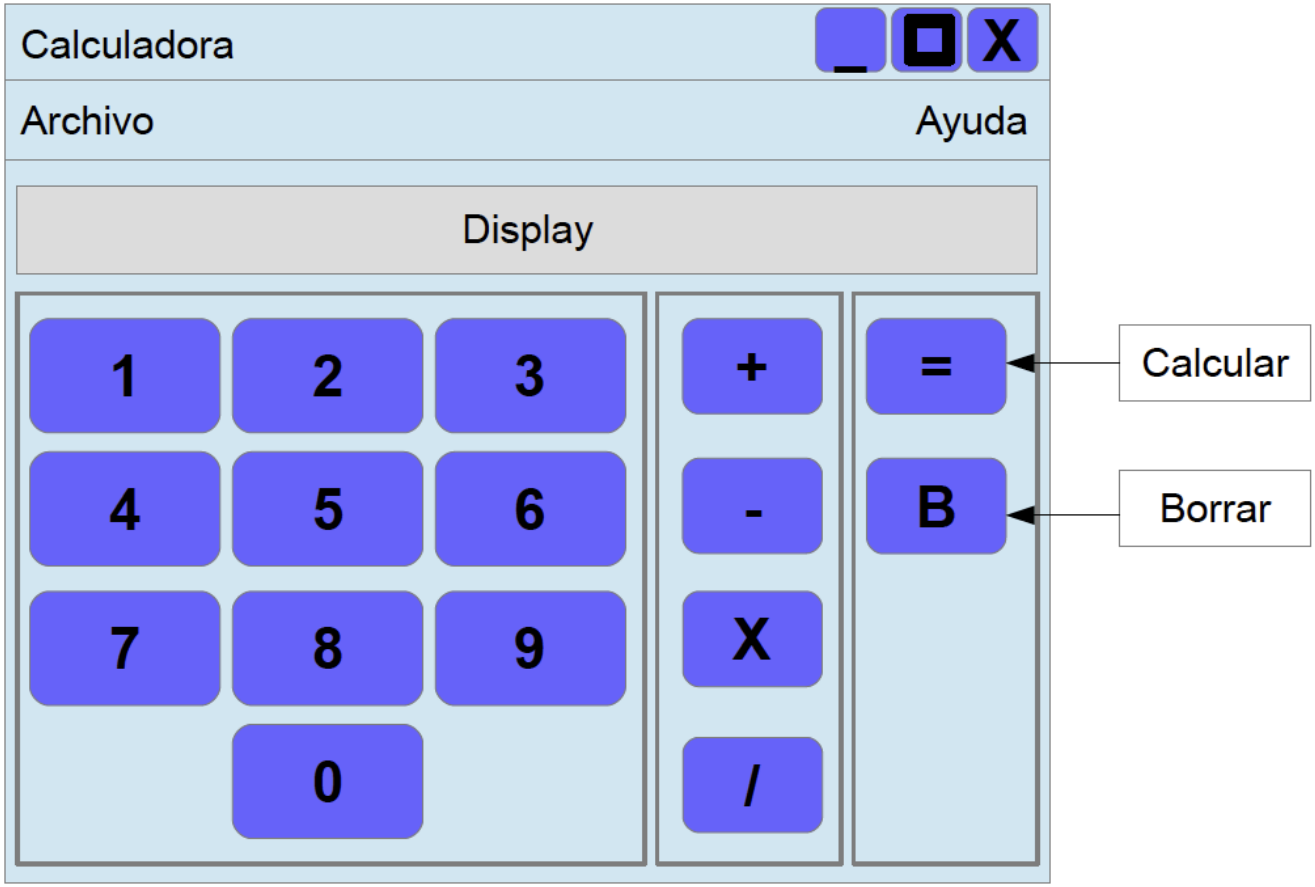
```
public partial class Principal : Form {
  private string lang = "es";
  public Principal() { ... }

  private void button1_Click(object sender,
EventArgs e) {
    if (lang=="es") {
      lang = "en";
      label1.Text = "Hello world!";
    } else {
      lang = "es";
      label1.Text = "Hola mundo!";
    }
  }
}
```

- Lo probamos ejecutando el proyecto

Creando el interfaz

- Vamos a diseñar el interfaz de la calculadora añadiendo al form principal los elementos necesarios:
  - Un menú con dos opciones: Archivo/Salir y Ayuda/Acerca de.
  - Una caja de texto o label que har. el papel de la pantalla o display
  - Botones para los números, operaciones, resultado y borrar.
  - Paneles para agrupar los botones relacionados.

Creando el interfaz



Añadiendo el menú

- Añadimos un 'MenuStrip', una barra de menú. La nombramos ‘menuPrincipal'.
- Le agregamos dos opciones principales 'ToolMenuStripItem': archivo y ayuda.
- A la opción ‘Archivo’:
  - La llamaremos ‘menuArchivo'
  - El texto será '&Archivo' el '&' es para indicar la tecla que lo activa (HotKey)
- La opción ‘Ayuda'
  - La llamaremos 'menuAyuda'. El texto ser. 'A&yuda'.
  - La alinearemos a la derecha.

Añadiendo el menú

- La opción 'Archivo' tendrá una opción de segundo nivel 'Salir':

  menuSalir, '&Salir'.

  - Haremos doble click para poder programar el manejador del evento click sobre esta opción.
  - El manejador simplemente saldrá de la aplicación:

```
private void menuSalir_Click(object sender,
EventArgs e) {
    Application.Exit();
}
```

  - Mejora: preguntar antes de salir: ¿Está seguro?.

Añadiendo el menú

- La opción 'Ayuda' tendrá una subopción 'Acerca de': menuAcercaDe, 'A&cerca de...'.
  - El manejador de la opción mostrará un diálogo modal con información del programa.

```
private void menuAcercaDe_Click(object sender,
EventArgs e) {
    MessageBox.Show(
    "Autor: Autor de la App\nVersión: 0.0.1",
    "Acerca de: Calculadora",
    MessageBoxButtons.OK,
    MessageBoxIcon.Information);
}
```

Añadimos los controles necesarios

- El display.
- El panel con los botones de los números.
- El panel con los botones de las operaciones aritméticas.
- El panel con los botones de acciones del programa: (=) para calcular el resultado, borrar.
- Para todos estos controles:
  - Renombrarlos: btn1, btn2, etc.
  - Ponerles las etiquetas correspondientes.

Añadimos los controles necesarios

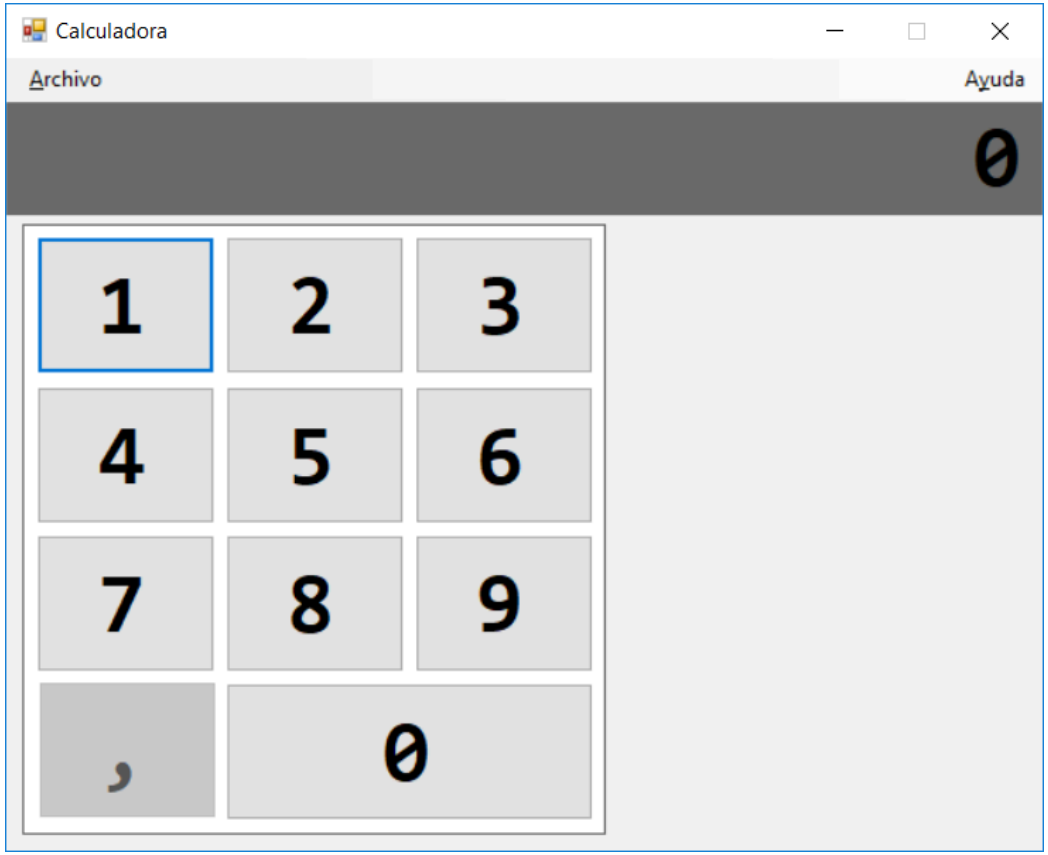
- Para añadir el display usaremos:
  - Un panel (color de fondo oscuro), que ocupe todo el ancho del form principal. Lo llamaremos pnlDisplay
  - Dentro del panel, insertaremos un label con las siguientes propiedades:
    - ➡ Nombre: lblDisplay, texto: 0
    - ➡ Tipografía: Console, bold, 36
    - ➡ Autosize: false
    - ➡ TextAlign: middle right, Anchor: right.

Añadimos los controles necesarios

- Incluimos un panel 'pnlNumeros' para ubicar los botones de los números.
- Añadimos los 10 botones para los números:
  - Los llamaremos: 'btn1', 'btn2', etc.
  - Les cambiaremos su texto: 1, 2, etc.
  - Los decoraremos con tipografía, tamaños y alineado adecuado para simular un teclado numérico.
  - El botón cero será un poco más grande.
  - Añadiremos un botón ',' (coma), llamado 'btnComa' para poder editar números decimales, que por ahora estará deshabilitado.

Añadimos los controles necesarios

- Aspecto del teclado numérico:

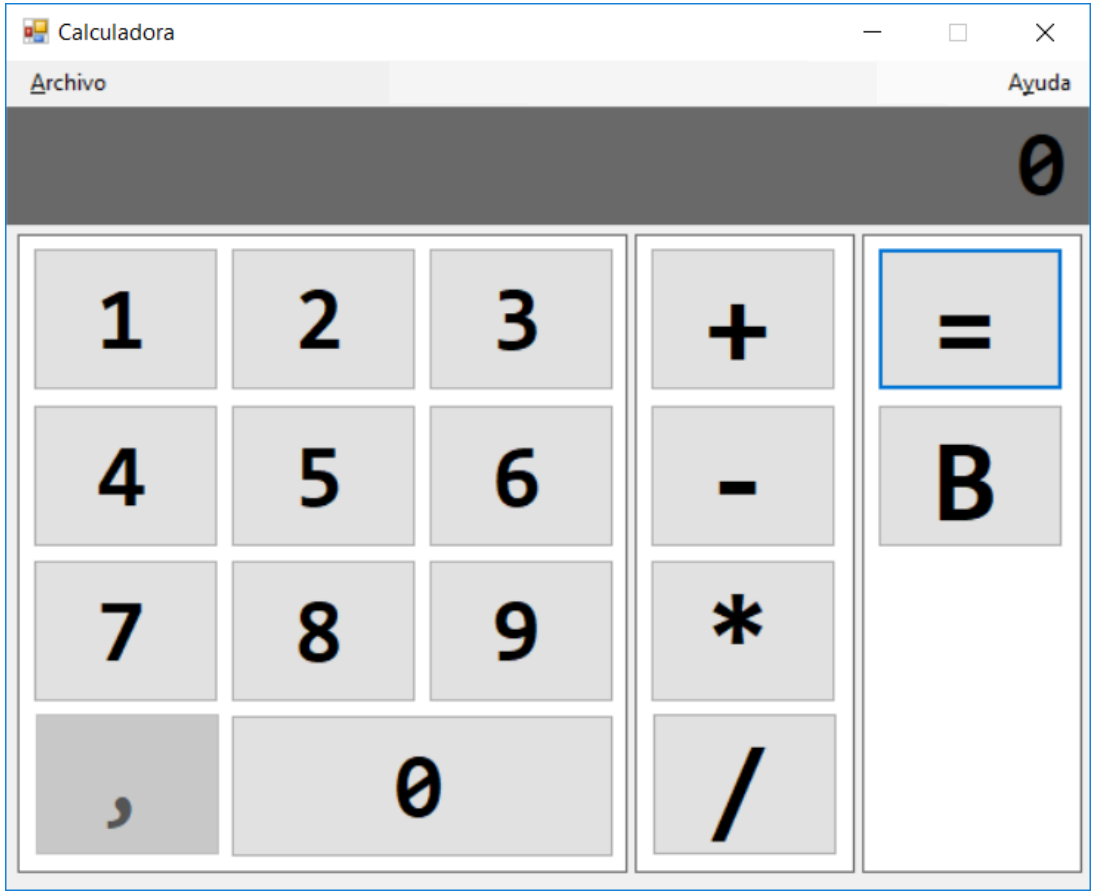


Añadimos los controles necesarios

- Con los demás botones del interfaz:
  - Añadiremos un panel: pnlOperadores
  - y dentro los cuatro botones de las 4 operaciones:
    - ➡ Llamados: btnSuma, btnResta, btnMultiplica, btnDivide
    - ➡ Textos: '+', '-', '*', '/'
  - Agregamos el panel 'pnlAcciones':
    - ➡ Con dos botones: btnCalcula, btnBorra
    - ➡ Textos: '=', 'B'.

Añadimos los controles necesarios

- Así quedará finalmente el interfaz:



Eventos y manejadores

- Los botones numéricos al recibir un click, modificarán el display añadiendo la cifra correspondiente. El evento será:

```
private void btn1_Click (object sender, EventArgs e)
{
    lblDisplay.Text += num;
}
```

- Los 10 botones harán exactamente lo mismo al ser pulsados:
  - Esto no es muy típico en una aplicación, porque cada botón suele tener un cometido muy específico y diferente de los demás.

Eventos y manejadores

- Mejoras:
  - Podríamos vincular el evento 'click' de los 10 botones al mismo manejador, o bien
  - Hacer que los manejadores de los botones numéricos invoquen a la misma función (**recomendable**)

Eventos y manejadores

- Por tanto, lo que haremos es crear 10 manejadores, uno por cada botón, que invoque a la misma función:

```
private void anyadirNumDisdplay(string num) {
    lblDisplay.Text += num;
}
private void btn1_Click(object sender, EventArgs e) {
    anyadirNumDisdplay("1");
}
private void btn2_Click(object sender, EventArgs e) {
    anyadirNumDisdplay("2");
}
```

Eventos y manejadores

- Un problema que tiene el código es que no elimina los ceros a la izquierda.

  Para solucionarlo, podemos modificar nuestra función común tal que:

```
private void anyadirNumDisdplay(string num) {
    string txt = lblDisplay.Text + num;
    int numero = Convert.ToInt32(txt);
    lblDisplay.Text = numero.ToString();
 }
```

  - Concatenamos el contenido del display con el nuevo número,

    convertimos la cadena a entero y finalmente la enviamos al display.

Eventos y manejadores

- Ahora tenemos otro problema, un número demasiado grande puede dar error al convertirlo, para ello vamos a capturar el posible error y si falla la conversión, no haremos nada.

```
private void anyadirNumDisdplay(string num) {
    string txt = lblDisplay.Text + num;
    int numero = 0;
    try {
        numero = Convert.ToInt32(txt);
        lblDisplay.Text = numero.ToString();
    } catch { /*nada*/ }
}
```

Gestión de las operaciones

- Declararemos e inicializaremos unas variables que necesitaremos:
  - Un enumerado para las operaciones

```
enum Operaciones {Suma, Resta, Producto,
Division};
```

  - Variables para los operandos, resultado y operación:

```
private int operando1, operando2, resultado;
private Operaciones operacion;
```

  - Inicializaremos las variables a cero en el constructor.

Gestión de las operaciones

- Los botones de operación al ser pulsados
  - Guardarán el contenido del display en el primer operando
  - Asignarán la operación que corresponda
  - Limpiarán el display

```
private void
btnSuma_Click (object sender, EventArgs e) {
   operando1 = Convert.ToInt32(lblDisplay.Text);
   operacion = Operaciones.Suma;
   lblDisplay.Text = "0";
}
```

Gestión de las operaciones

- El botón de cálculo (=):
  - Almacenará el contenido del display en el segundo operando.
  - Realizará la operación que corresponda según el valor de 'operación' y la almacenará en 'resultado'.
  - Mostrará el resultado en el display
  - Reiniciará los operandos a cero, preparándose para una nueva operación

Gestión de las operaciones

- Manejador del botón de cálculo (=):

```
private void
btnCalcula_Click(object sender, EventArgs e) {
   operando2 = Convert.ToInt32(lblDisplay.Text);
   switch (operación) {
      case Operaciones.Suma:
         resultado = operando1 + operando2;
         break;
         . . .
   }
   lblDisplay.Text = resultado.ToString();
   Operando1 = 0; operando2 = 0;
}
```

Gestión de las operaciones

- El botón borrar simplemente, reiniciará el display conservando la operación seleccionada

```
private void btnBorra_Click(object
sender, EventArgs e)
{
    lblDisplay.Text = "0";
}
```

Mejoras: división por cero

- Podemos comprobar justo antes de dividir si el divisor es cero y mostrar un aviso o error en ese caso.

```
case Operaciones.Division:
   if (operando2 != 0) {
      resultado = operando1 / operando2;
   } else {
      MessageBox.Show("No se puede dividir por cero!", "Error",
      MessageBoxButtons.OK, MessageBoxIcon.Error);
   }
   break;
```

Mejoras: división por cero

- Otro enfoque ser.a deshabilitar el botón '=' cuando se den

  las condiciones en las que se puede llegar a dividir por cero:

  - Al pulsar sobre '/', pues el segundo operando es 0.
  - Al estar en 'modo división' y tener un cero en el display.

Mejoras: desbordamiento

- Desbordamiento: Algunas operaciones pueden obtener un n.mero demasiado grande, que exceda el tamaño máximo, en ese caso, una solución es usar try/catch, capturar un posible error al realizar la operación y mostrar el error correspondiente.

```
try {
   switch (operación) {
      . . .
      case Operaciones.Producto:
         checked {
            resultado = operando1 * operando2;
         }
         break;
      . . .
   }
} catch (OverflowException ex) {
   MessageBox.Show("La operación ha causado un desbordamiento",
   "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
```

Ejercicios de ampliación

- Sencillos:
  - Mostrar dialogo ‘¿Está seguro?' antes de salir.
  - Añadir un botón para cambiar de signo el número actual del display (+/-).
  - Permitir trabajar con decimales.
  - Aumentar el rango máximo de los operadores (usando un tipo con mayor capacidad).

Ejercicios de ampliación

- No tan sencillos:
  - Agregar una 'status bar' o cuadro informativo donde se
  - pueda observar la operación en curso.
  - Incluir otra operación: potencia (^)
  - Añadir otra operación: raíz cuadrada.

    → ¡Esta operación sólo tiene un operando!