# Introducción a Xamarin.Forms

# SHARPULADORA

¿Quién no ha hecho un clásico “Hola mundo!” cuando inicia a trabajar con un nuevo lenguaje/framework? Pues bueno, tan clásico es el *hola mundo* como hacer una calculadora y para Xamarin.Forms no es la excepción. En esta guía te mostraré cómo usar algunos controles para crear tus apps… y entre ellas ¡tu propia calculadora!

# ¿Cómo funciona esta guía?

Esta guía emplea un formato interactivo, viene acompañada de código que puedes probar y modificar al momento que haces el ejercicio. El código se divide en tres o cuatro carpetas:

- **start** es el punto de inicio de la guía, contiene una solución preparada para comenzar a trabajar.
- **parts** son todos y cada uno de los archivos necesarios para completar la guía, usualmente son provistos como referencia para que si te llegas a atorar con algo los revises y si es necesario copies el código para seguir avanzando con la guía. Este documento está dividido en partes y a cada una de ellas le corresponde una subcarpeta.
- **final** es el resultado final, al que queremos llegar con la guía. Esta carpeta, al igual que la carpeta `parts` es incluida como referencia, para mostrar el producto final.
- **bonus** (opcional) muchas veces, al tratar de mostrar un concepto, es muy probable que se omitan algunas buenas prácticas, se evite el uso de librerías de terceros, etc. Por lo cual, esta carpeta contendrá una aplicación similar a la que se encuentra en `final` pero tomando en cuenta cosas que se pudieron haber omitido para favorecer el aprendizaje del tema tratado.

Como se menciona antes, esta guía está dividida en varias partes que componen los temas que se verán a lo mientras comienzas a desarrollar tu propia app. Es muy probable que en cada parte se vea una porción de código y posteriormente se haga referencia a uno o varios archivos en los que se hace uso de lo mencionado, estos son los archivos que vas a encontrar dentro de la carpeta `parts`.

Si tienes dudas sobre el funcionamiento de la guía, házmelas saber a feregrino@thatcsharpguy.com o @io_exception en Twitter.

# La Sharpuladora

# Introducción

Dentro de esta guía se asume que ya tienes un entendimiento básico de cómo es que funciona Xamarin, si no, te invito a que consultes Getting started with Xamarin. Mi post sobre qué necesitas para desarrollar con Xamarin o este otro sobre cómo ir de la consola a apps móviles. O cualquier otro recurso de confianza.

# Parte 1 - Familiarizándose con el código

Como todos los entornos de desarrollo, tanto Xamarin Studio como Visual Studio tienen su propia manera de gestionar los archivos que componen nuestras aplicaciones, pero en breve te cuento que en estos entornos se dividen los proyectos por plataforma, siendo así que como queremos que nuestra calculadora esté disponible para tres distintas tendremos tres proyectos diferentes más uno, que es el núcleo de nuestra aplicación. Escribí un post sobre esto, para conocer más visita ¿Cómo organiza mi código Visual Studio?.

Es momento de empezar a meterle mano al código, abre la solución `Sharpuladora.sln` que está dentro de la carpeta `start` y mira el panel del explorador de soluciones. Para esta parte no hay código que agregar, solo ver. Una vez que hayas explorado lo suficiente, avanza a la parte 2.

# Parte 2 - Las páginas

Las interfaces gráficas de las aplicaciones en Xamarin.Forms se compone de un conjunto de elementos del tipo `Page`, sin embargo, Forms nos ofrece algunas páginas pre-programadas que podemos usar para trabajar con nuestras apps. Puedes consultar el listado completo aquí.

La *Sharpuladora* únicamente consta de una sola pantalla, que creamos a partir de una `ContentPage` que es de las más sencillas de manejar.

# Parte 2.1 - Creando la página principal

Para crear una página nueva tenemos en realidad dos opciones: crearla en código C# o usar XAML para hacerlo. En este caso utilizaremos C#.

Crea una nueva carpeta llamada `Pages` dentro del proyecto `Sharpuladora` y dentro, crea una nueva clase de C# llamada `CalcPage` que derive de `ContentPage`.

## Código

Archivo `CalcPage.cs`

# Parte 2.2 - Estableciendo una página de inicio

Es importante establecer la pantalla de inicio de nuestra aplicación y para el caso de Xamarin.Forms se debe especificar a través de una propiedad. Dicha propiedad se llama `MainPage` y se encuentra en la clase `App` (dentro de `App.cs`).

En la solución dentro de `start` la propiedad `MainPage` es inicializada con una página creada ahí mismo, debemos cambiarla por una referencia a nuestra `CalcPage`:

```
MainPage = new CalcPage();
```

## Código

Archivo `App.cs`

# Parte 3 - El layout

Nuestra página actúa como un contenedor para los elementos de la interfaz, sin embargo, debemos agregarle un *layout* para indicarle la organización de dichos elementos. Al igual que con las páginas, Forms viene pre-cargado con una colección de *layouts* que puedes consultar en este enlace.

# Parte 3.1 - El Grid

Una calculadora sencilla no requiere de mucho pensar, los botones están ordenados en filas y columnas, lo cual parece un trabajo para un `Grid`. Un `Grid` está compuesto de filas y columnas, las cuales en Xamarin.Forms se especifican a través de `RowDefinition` o `ColumnDefinition`. Mira el siguiente código:

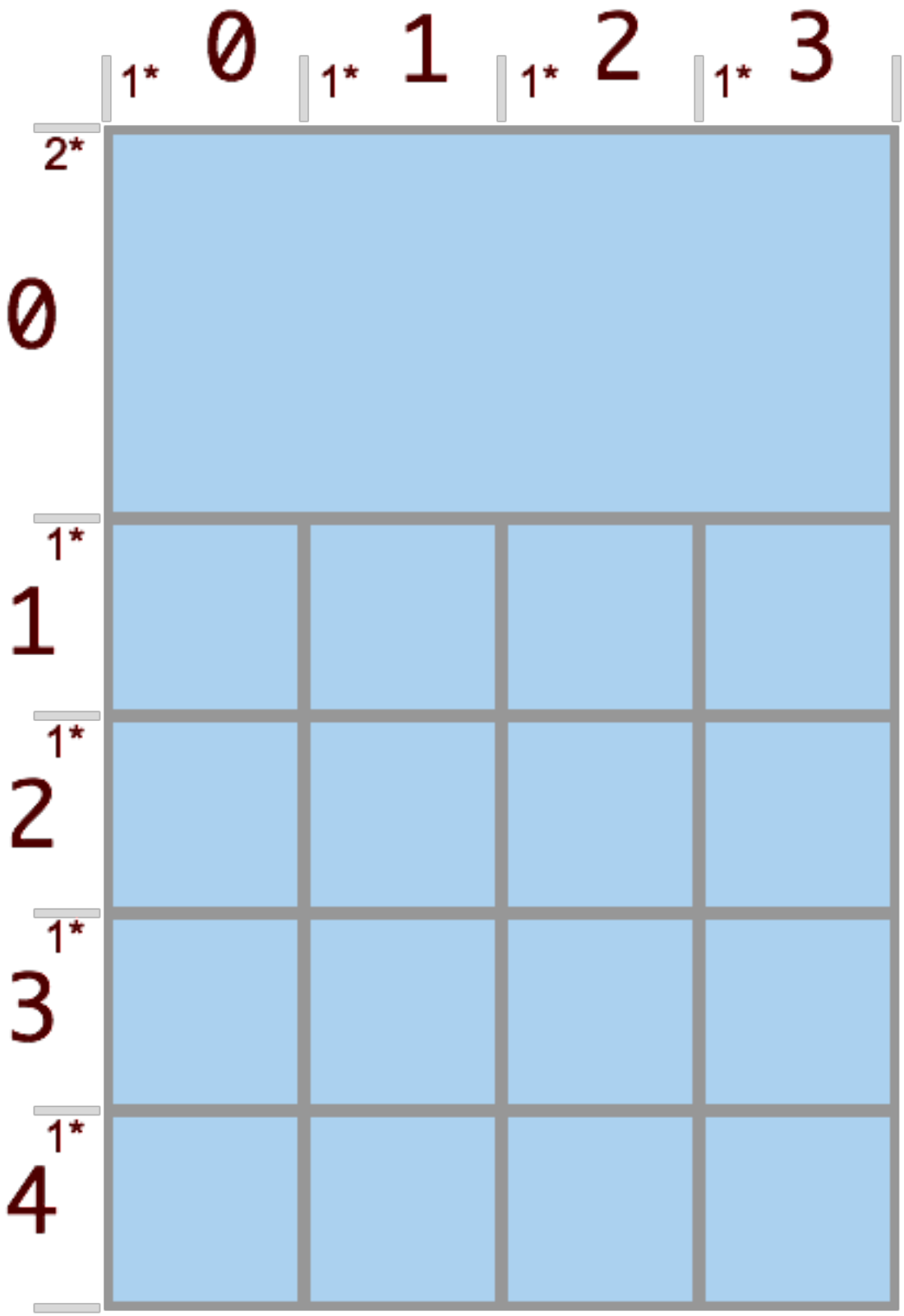
```
var layout = new Grid();

// Filas:
layout.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(2, GridUnitType.Star) });
layout.RowDefinitions.Add(new RowDefinition { Height = new
GridLength(1, GridUnitType.Star) });

// Columnas:
layout.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
layout.ColumnDefinitions.Add(new ColumnDefinition { Width = new
GridLength(1, GridUnitType.Star) });
```

En el código anterior se crea un nuevo `Grid` llamado `layout` y se le añaden dos columnas y dos filas. Algo importante a notar es que para las columnas se les establece la propiedad `Width` y para las filas a propiedad `Height`. El hecho de que se use `GridUnitType.Star` (o `*`) indica que el tamaño de las columnas será proporcional entre ellas.

Para el ejercicio, crea un grid con filas y columnas que representen el siguiente gráfico:


0
1
2
3
1*
1*
1*
1*
0
2*
1
1*
2
1*
3
1*
4
1*

Una vez creado el *layout* es importante especificarle a nuestra página que debe usarlo como contenido, esto se hace normalmente al final del constructor de la página con la siguiente línea de código:

```
Content = _layout;
```

### Código

Archivo `CalcPage.cs`

# Parte 4 - Los elementos de interfaz

Una vez creado el contenedor para nuestros elementos (nuestro *layout*), es momento de continuar con la creación de la interfaz. Usaremos botones para los dígitos y los operadores y un *label* para mostrar el resultado.

La inicialización de controles es bastante sencilla, por ejemplo, para crear el botón que será el '0' en nuestra calculadora tenemos que escribir el siguiente código:

```
_b0 = new Button { Text = "0" };
```

Una vez creado el control debemos añadirlo al *layout* contenedor para que se muestre en la pantalla. Recordemos que nuestro *layout* se compone de filas y columnas, así que debemos asignarle una fila y una columna en donde mostrarse a nuestro control, la clase `Grid` nos permite asignarlos con los métodos estáticos `SetColumn` y `SetRow`. En el código siguiente, se le asigna la columna 1 y la fila 4 al botón `_b0`:

```
Grid.SetColumn(_b0, 1);
Grid.SetRow(_b0, 4);
```

Para posteriormente añadirlo al *layout* con:

```
_gridLayout.Children.Add(_b0);
```

Para el caso de la etiqueta en donde se mostrará el resultado, haremos uso del método `SetColumnSpan` para indicarle a nuestro control que debe usar más de una columna dentro del *layout* que lo contiene. En este caso, le estamos indicando a `_resultDisplay` que debe ocupar 4 columnas:

```
Grid.SetColumnSpan(_resultDisplay, 4);
```

También existe `SetRowSpan` si lo que deseamos es que el control ocupe más de una fila.

### Código

Archivo `CalcPage.cs`

# parte 5 - Los eventos

En una calculadora como la que estamos haciendo se requiere de interacción de los usuarios y en Xamarin.Forms las interacciones son manejadas a través de eventos. Cada uno de los controles que añadimos a una página pueden tener uno o más eventos que son "lanzados" por interacciones con el usuario (o con otros elementos de nuestro programa), como programadores es necesario indicar qué eventos queremos manejar y cómo los vamos a manejar.

Para nuestra calculadora vamos a manejar el evento `Clicked` de los botones, dicho evento es lanzado cada vez que el usuario da "click" sobre el botón. Manejar eventos es cosa sencilla en C#, lo que tenemos que hacer es asignar (con el operador `+=`) un manejador al evento del control que deseamos. En el siguiente fragmento de código se especifica un manejador para el evento `Clicked` de `_b0`:

```
_b0.Clicked += OnNumericButtonClicked;
```

El manejador `OnNumericButtonClicked` no es más que un método que tiene como tipo de retorno `void` y recibe dos argumentos:

- `object sender` que es una referencia al control que lanzó el evento
- `EventArgs e` que son algunos argumentos extra sobre el evento lanzado

Como ejemplo acá abajo está la implementación del método para los botones del teclado numérico de nuestra calculadora. En la primera línea se ase un *cast* de `sender` a `Button` lo cual es válido porque como comenté: en `sender` viene el control que lanzó el evento.

```
void OnNumericButtonClicked(object sender, EventArgs e)
```

```
{
    Button botonClickeado = (Button)sender;
    if (primerNumero == null || primerNumero == "")
    {
        primerNumero = botonClickeado.Text;
    }
    else
    {
        segundoNumero = botonClickeado.Text;
    }
}
```

Los manejadores pueden ser compartidos entre muchos controles, así que no tienes que crear un manejador para cada control que uses en tu aplicación. Así que ahora es tu turno de crear toda la lógica de la calculadora usando los controles y los manejadores de eventos. Si te encuentras un poco perdido puedes echarle un ojo al archivo que te indico màs abajo para comenzar.

### Código

Archivo `CalcPage.cs`

# Parte 6 - Un pequeño truco

Podrás pensar que nuestra calculadora está (casi) terminada, pero no. Si la echas a andar a como la tenemos hasta ahora te darás cuenta que la calculadora no ocupa la pantalla entera... un pequeño bug. Para resolverlo haremos uso de otro tipo de *layout* que envuelva a nuestro *layout* original.

Para la ocasión haremos uso de un `RelativeLayout` que nos permite mayor flexibilidad a la hora de controlar cómo se muestran los elementos y por ende, requiere de mayor configuración. Reemplaza la asignación `Content = _layout;` por el siguiente fragmento de código:

```
// Trick to make our calculater fullscreen
var relativeLayout = new RelativeLayout();
relativeLayout.Children.Add(_layout, // <= Original layout
    Constraint.Constant(0),
    Constraint.Constant(0),
    Constraint.RelativeToParent(p => p.Width),
    Constraint.RelativeToParent(p => p.Height));
Content = relativeLayout;
```

# Siguientes pasos

Como probablemente te habrás dado cuenta, la aplicación no está completa, le falta implementar algunos manejadores y toda la lógica de la calculadora. Esa es tu misión, si decides aceptarla.

Recuerda que si llegas a tener dudas sobre lo visto en esta guía puedes contactarme vía correo o Twitter.