



# TEMA 10.

WINDOWS PRESENTATION FOUNDATION (WPF)

### 3. WPF - INTRODUCCIÓN

#### ■ ¿Qué es WPF?

*Windows Presentation Foundation (WPF)* es un marco para construir interfaces de usuario en *Windows*, destacando por su independencia de resolución y su motor gráfico basado en vectores.

Está diseñado para sacar partido del hardware gráfico moderno, ofreciendo una experiencia visualmente rica y fluida.

## 3. WPF - INTRODUCCIÓN

### ■ Características Clave de WPF

- Variedad de Herramientas de Desarrollo: Incluye el *Extensible Application Markup Language (XAML)*, controles, enlace de datos, y diseño de gráficos 2D y 3D.
- Animación y Estilos: Permite crear interfaces interactivas y estéticamente atractivas con animaciones, estilos, plantillas y tipografía avanzada.
- Integración Multimedia: Incorpora elementos multimedia, textos y documentos en tus aplicaciones.
- Compatibilidad con .NET: Forma parte del ecosistema .NET, lo que permite integrar fácilmente otras funciones y características de .NET en tus aplicaciones WPF.

*Para una comprensión más profunda y documentación técnica, visita la [documentación oficial de Microsoft sobre WPF \(https://learn.microsoft.com/es-es/dotnet/desktop/wpf/?view=netdesktop-8.0\)](https://learn.microsoft.com/es-es/dotnet/desktop/wpf/?view=netdesktop-8.0)*

## 3.1 - IMPLEMENTACIONES DE WPF

### ■ .NET Framework 4

- Compatibilidad: Funciona con *Visual Studio 2019* y *Visual Studio 2017*.
- Características: Parte de *.NET Framework*, considerado un componente esencial de Windows. WPF en esta versión viene integrado con el .NET Framework.
- Especificaciones: Ideal para aplicaciones solo de Windows.

### ■ .NET (Versión Abierta)

- Compatibilidad: Requiere *Visual Studio 2019* (versión 16.8 en adelante) y se ejecuta en .NET 5 o versiones superiores, incluyendo .NET Core 3.1.
- Características: Implementación de código abierto de WPF, disponible en GitHub. Aunque .NET admite múltiples plataformas, WPF se ejecuta exclusivamente en Windows.
- Consideraciones: El diseñador de XAML es un requisito clave para esta versión.

## 3.2 - XAML: EL CORAZÓN DE WPF

### ■ Introducción a XAML

- ***XAML (Extensible Application Markup Language)*** es un lenguaje de marcado basado en XML, utilizado para definir la interfaz gráfica de las aplicaciones en WPF. Es ideal para describir ventanas, diálogos, páginas, controles de usuario, y para organizarlos con formas y gráficos.
- ***XAML*** permite construir interfaces de usuario de manera declarativa y jerárquica, facilitando la gestión y el mantenimiento de proyectos complejos.

### ■ Árbol de Elementos en XAML

- La interfaz de usuario creada en ***XAML*** se organiza en una estructura jerárquica conocida como ***árbol de elementos***.
- Este enfoque ofrece una manera lógica y eficiente para construir y administrar interfaces complejas.

```
1  <Window x:Class="WpfDDI.MainWindow"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6      xmlns:local="clr-namespace:WpfDDI"
7      mc:Ignorable="d"
8      Title="MainWindow" Height="450" Width="800">
9      <Grid>
10         ...
11     </Grid>
12 </Window>
```

## 3.2.1 - XAML: SINTAXIS Y USO

### ■ Etiquetas y Atributos

- Etiquetas XAML: Las etiquetas en *XAML* definen los controles y elementos de la interfaz, cada uno con sus atributos específicos para su configuración.
- Atributos de Control: Estos atributos, definidos dentro de las etiquetas, especifican las propiedades de los controles, como tamaño, color, comportamiento, etc.

### ■ Propiedades y Elementos

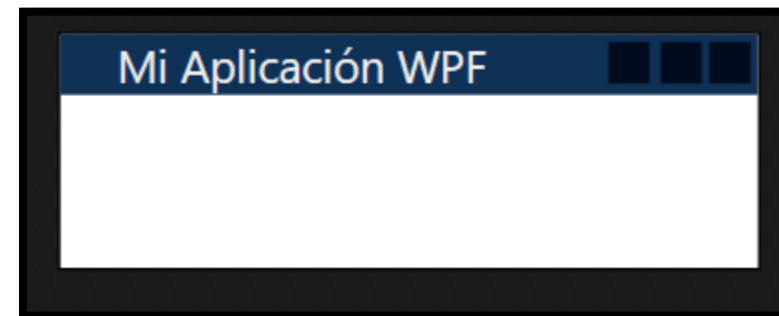
- Definición de Propiedades: Además de los atributos del control, ciertas propiedades pueden ser modificadas y no se encuentran en la etiqueta del elemento, sino definidas dentro de las etiquetas de apertura y cierre.
- Elemento Window: Es el elemento raíz en la mayoría de las aplicaciones WPF. Contiene el resto de los elementos y define propiedades como título, tamaño, y comportamiento de la ventana.

## 3.2.2 - XAML: PROPIEDADES ELEMENTO WINDOW

### ■ Propiedades Comunes del Elemento Window

- Title: Define el título de la ventana.
- WindowStartupLocation: Establece la posición inicial de la ventana.
- WindowState: Controla si la ventana se inicia maximizada, minimizada o en su tamaño normal.
- WindowStyle: Determina el estilo del borde de la ventana.
- Topmost: Establece si la ventana debe mostrarse por encima de todas las demás aplicaciones.

```
1 <Window x:Class="MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       Title="Mi Aplicación WPF"
5       WindowStartupLocation="CenterScreen"
6       WindowState="Normal"
7       WindowStyle="SingleBorderWindow"
8       Topmost="True" Height="62" Width="186">
9     <!-- El contenido de la ventana va aquí -->
10  </Window>
```

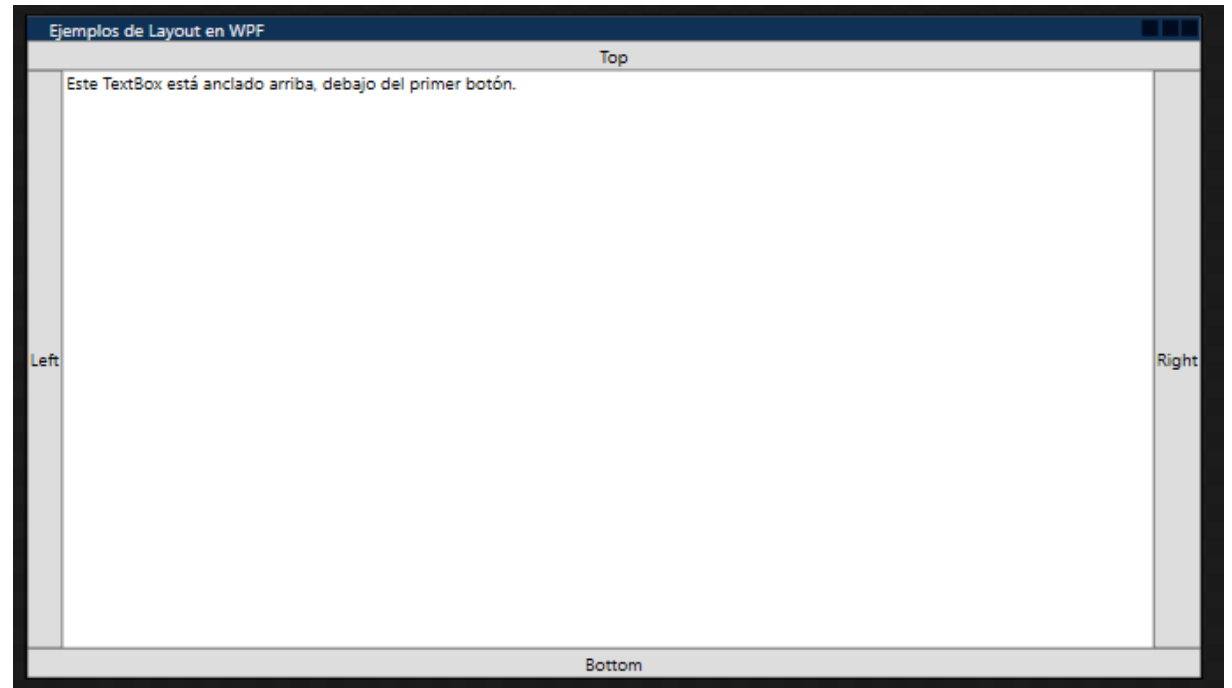




## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ DockPanel

- Permite organizar elementos secundarios de forma horizontal o vertical. Ideal para diseños donde algunos controles deben anclarse a los bordes de la ventana.



```
5  <DockPanel LastChildFill="True">
6      <Button DockPanel.Dock="Top" Content="Top" />
7      <Button DockPanel.Dock="Bottom" Content="Bottom" />
8      <Button DockPanel.Dock="Left" Content="Left" />
9      <Button DockPanel.Dock="Right" Content="Right" />
10     <TextBox DockPanel.Dock="Top" Text="Este TextBox está anclado arriba, debajo del primer botón." />
11 </DockPanel>
```

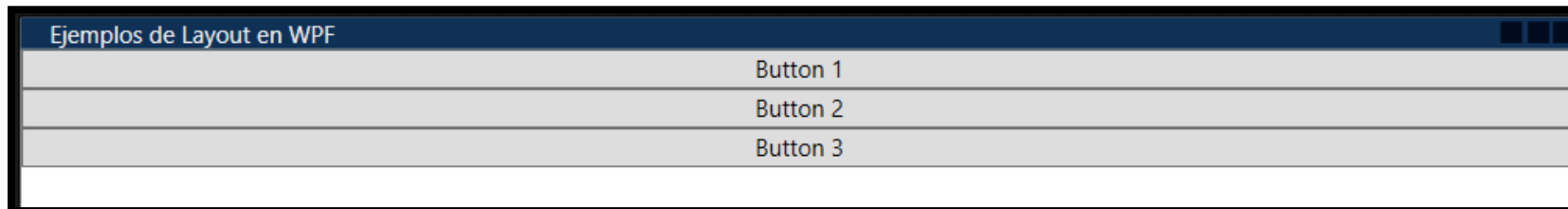


## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ StackPanel

- Organiza los elementos secundarios en una sola línea, ya sea horizontal o vertical. Útil para listas o grupos de botones.

```
5  <StackPanel Orientation="Vertical">
6      <Button Content="Button 1" />
7      <Button Content="Button 2" />
8      <Button Content="Button 3" />
9  </StackPanel>
```

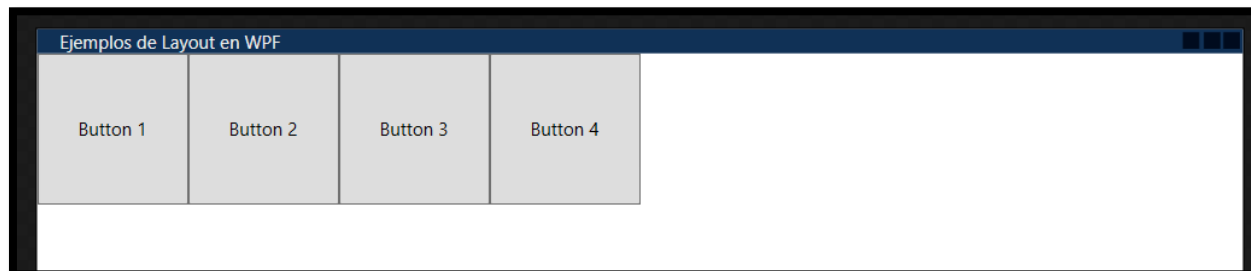


## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ WrapPanel

- Coloca los elementos secundarios secuencialmente de izquierda a derecha, moviéndose a la línea siguiente al alcanzar el borde del contenedor. Ideal para flujos de elementos como galerías de imágenes.

```
5  <WrapPanel ItemWidth="100" ItemHeight="100">
6      <Button Content="Button 1" />
7      <Button Content="Button 2" />
8      <Button Content="Button 3" />
9      <Button Content="Button 4" />
10 </WrapPanel>
```

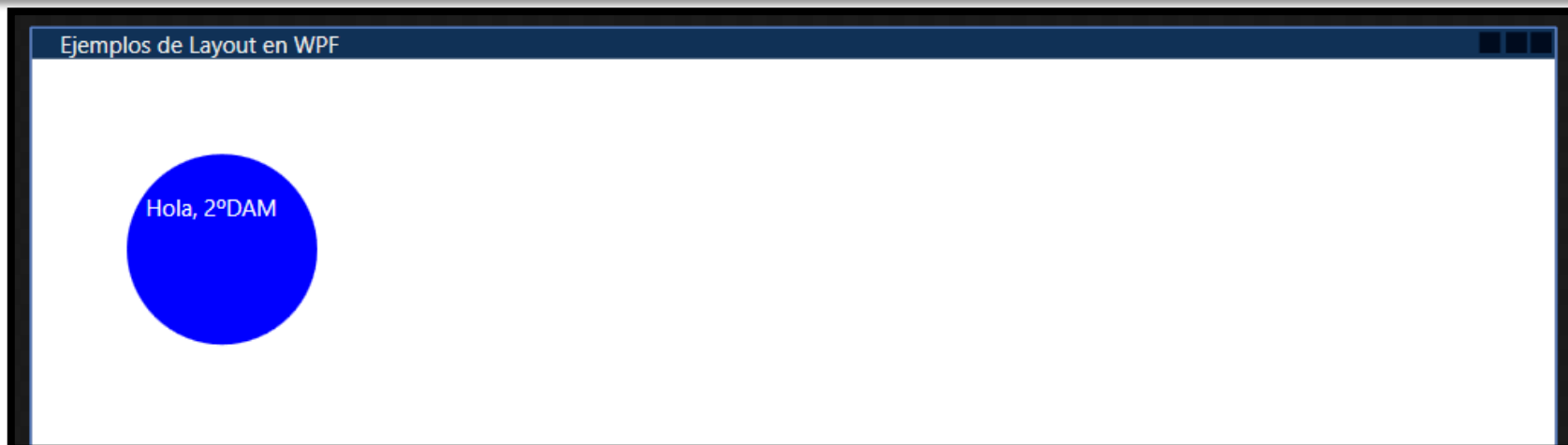


## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Canvas

- Define un área donde se pueden colocar elementos secundarios en coordenadas específicas. Perfecto para diseños precisos y personalizados.

```
5  <Canvas>
6      <Ellipse Canvas.Left="50" Canvas.Top="50" Width="100" Height="100" Fill="Blue" />
7      <TextBlock Canvas.Left="60" Canvas.Top="70" Foreground="White">Hola, 2ºDAM</TextBlock>
8  </Canvas>
```



## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Grid

- Crea una cuadrícula flexible con filas y columnas para organizar elementos. Es el más versátil para diseños complejos.

```
5  <Grid>
6      <Grid.RowDefinitions>
7          <RowDefinition Height="*" />
8          <RowDefinition Height="2*" />
9          <RowDefinition Height="*" />
10     </Grid.RowDefinitions>
11     <Grid.ColumnDefinitions>
12         <ColumnDefinition Width="Auto" />
13         <ColumnDefinition Width="*" />
14         <ColumnDefinition Width="Auto" />
15     </Grid.ColumnDefinitions>
16     <Button Grid.Row="0" Grid.Column="0" Content="R0, C0" />
17     <Button Grid.Row="1" Grid.Column="1" Content="R1, C1" />
18     <Button Grid.Row="2" Grid.Column="2" Content="R2, C2" />
19 </Grid>
```

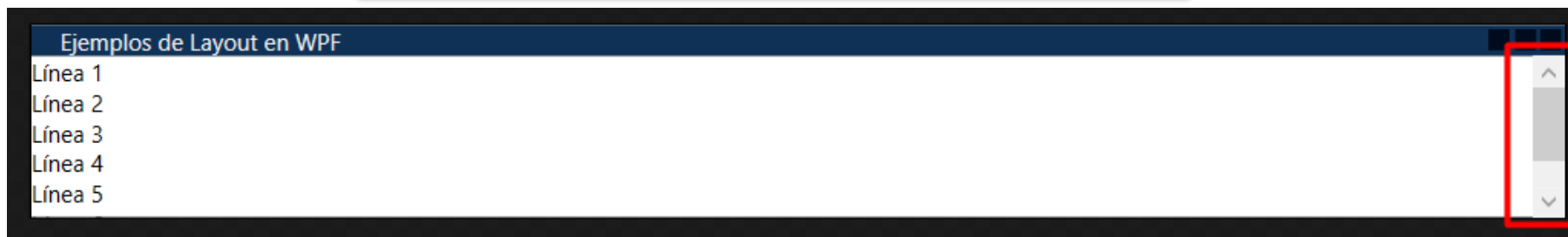


## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ ScrollView

- Presenta un área desplazable que puede contener otros elementos visuales. Importante para interfaces con contenido extenso.

```
5  <ScrollView VerticalScrollBarVisibility="Auto">
6      <StackPanel>
7          <TextBlock Text="Línea 1" />
8          <TextBlock Text="Línea 2" />
9          <TextBlock Text="Línea 3" />
10         <TextBlock Text="Línea 4" />
11         <TextBlock Text="Línea 5" />
12         <TextBlock Text="Línea 6" />
13         <TextBlock Text="Línea 7" />
14     </StackPanel>
15 </ScrollView>
```



## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Anidamiento de Contenedores en WPF

- En *WPF*, los contenedores se pueden anidar para crear diseños complejos y flexibles. Por ejemplo, un *StackPanel* puede contener varios *Grids*, o un *Grid* puede contener varios *Panels*.
- Se puede utilizar el anidamiento para segmentar diferentes áreas de la interfaz y para organizar los controles de manera lógica y eficiente.

```
6      <StackPanel>
7      |
8      |   <Grid>
9      |   |
10     |   |   <Button Content="Botón en Grid 1" />
11     |   |
12     |   |   </Grid>
13     |   |
14     |   |   <Grid>
15     |   |   |
16     |   |   |   <Button Content="Botón en Grid 2" />
17     |   |   |
18     |   |   |   </Grid>
19     |   |   </Grid>
20     |   </StackPanel>
```

## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Medidas en WPF

- Tres Formas de Medición: Los tamaños en WPF se pueden especificar en píxeles, porcentajes, o automáticamente (Auto).
- Esto permite adaptar la interfaz a diferentes tamaños de pantalla y resoluciones.

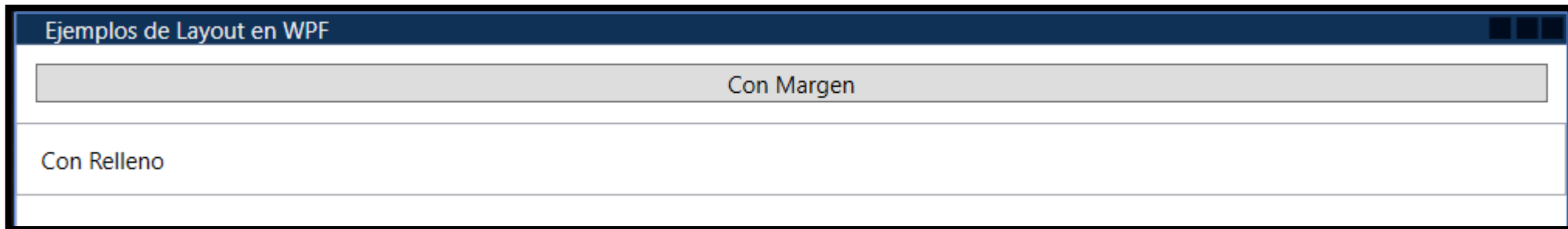
```
6  <StackPanel>
7      <Button Width="100" Height="50" Content="100x50 píxeles" />
8      <Button Width="Auto" Height="Auto" Content="Auto" />
9      <Button Width="1*" Height="1*" Content="1* (Porcentaje)" />
10 </StackPanel>
```



## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Margen y Relleno

- Margen (Margin): Define el espacio entre el elemento y sus elementos adyacentes, crucial para un diseño claro y bien espaciado.
- Relleno (Padding): Se refiere al espacio dentro del elemento, entre el borde y su contenido.

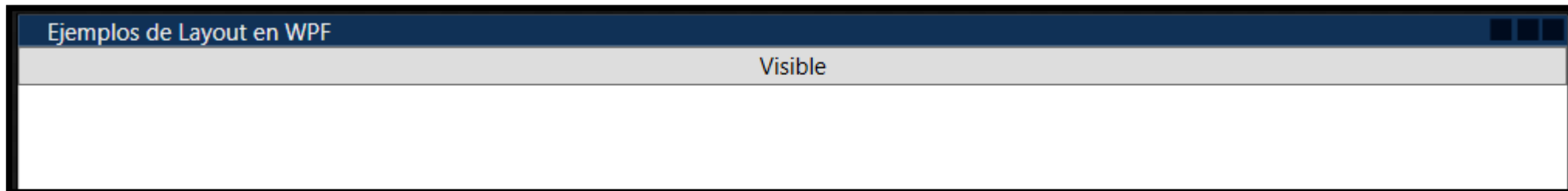


```
7  <StackPanel>
8      <Button Content="Con Margen" Margin="10" />
9      <TextBox Text="Con Relleno" Padding="10" />
10 </StackPanel>
```

## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Visibilidad

- Control de Visibilidad: La propiedad *Visibility* permite ocultar o mostrar elementos.
- *Hidden* mantiene el espacio del elemento aunque no sea visible, mientras que *Collapsed* elimina el elemento y su espacio.

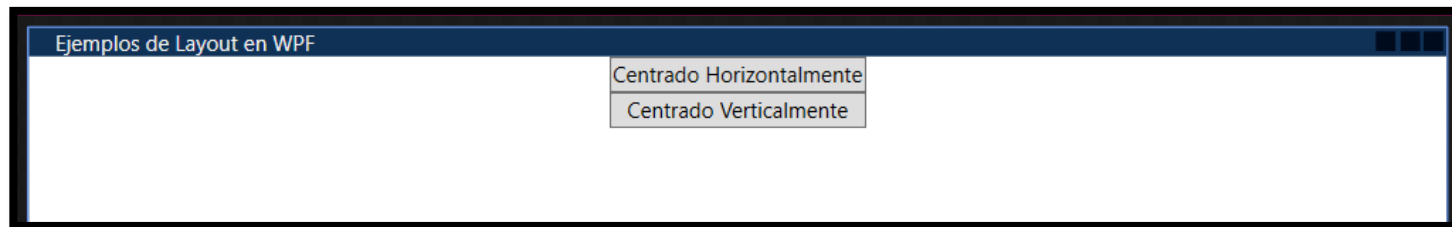


```
8      <StackPanel>
9          <Button Content="Visible" Visibility="Visible" />
10         <Button Content="Oculto" Visibility="Hidden" />
11         <Button Content="Colapsado" Visibility="Collapsed" />
12     </StackPanel>
```

## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Posiciones y Alineación

- Alineación Vertical y Horizontal: Las propiedades *VerticalAlignment* y *HorizontalAlignment* controlan cómo se posicionan y alinean los elementos dentro de su contenedor.



```
9      <StackPanel HorizontalAlignment="Center">
10          <Button Content="Centrado Horizontalmente" HorizontalAlignment="Center" />
11          <Button Content="Centrado Verticalmente" VerticalAlignment="Center" />
12      </StackPanel>
```

## 3.2.3 - XAML: CONTROLES DE LAYOUT EN WPF

### ■ Colores

- Selección de Colores: Se puede elegir colores predefinidos o especificar colores personalizados mediante códigos hexadecimales para los controles y elementos de la interfaz.



```
8  <StackPanel>
9      <TextBlock Text="Texto Azul" Foreground=■"Blue" />
10     <Rectangle Fill=■"Red" Height="50" />
11     <Button Content="Fondo Verde" Background=■"Green" />
12 </StackPanel>
13
```

## 3.2.4 - XAML: CONTROLES BÁSICOS EN WPF

### ■ Label

- **Función:** Representa la etiqueta de texto de un control.
- **Propiedades Comunes:**
  - **Content:** Texto a mostrar.
  - **FontSize:** Tamaño de la fuentes.
  - **FontStyle:** Estilo de la fuentes.
  - **Foreground:** Color del texto.

```
8  <Label Content="Etiqueta de Texto"  
9      FontSize="16"  
10     FontStyle="Italic"  
11     Foreground="Blue"/>
```

### Ejemplos de Layout en WPF

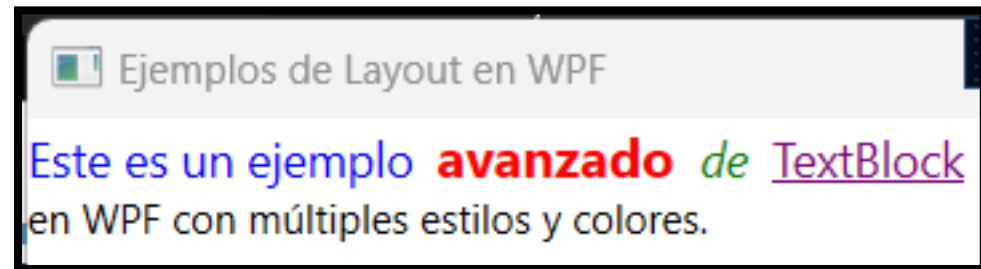
*Etiqueta de Texto*

## 3.2.4 - XAML: CONTROLES BÁSICOS EN WPF

### ■ TextBlock

- Función: Control para mostrar pequeñas cantidades de texto.
- Propiedades Comunes:
  - Text: Texto a mostrar.
  - TextTrimming: Comportamiento de recorte del texto.
  - TextWrapping: Ajuste del texto.

```
12 <TextBlock TextWrapping="Wrap">
13     <Run Text="Este es un ejemplo " Foreground="Blue" FontSize="14"/>
14     <Run Text="avanzado " FontWeight="Bold" Foreground="Red" FontSize="16"/>
15     <Run Text="de " FontStyle="Italic" Foreground="Green" FontSize="14"/>
16     <Run Text="TextBlock" TextDecorations="Underline" Foreground="Purple" FontSize="14"/>
17     <LineBreak/>
18     <Run Text="en WPF con múltiples estilos y colores." FontSize="12"/>
19 </TextBlock>
```

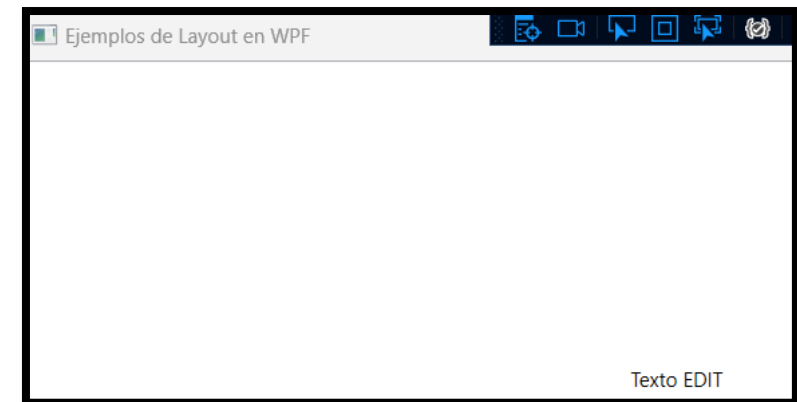


## 3.2.4 - XAML: CONTROLES BÁSICOS EN WPF

### ■ TextBox

- Función: Control para mostrar o editar texto.
- Propiedades Comunes:
  - Text: Texto a mostrar o editar.
  - CharacterCasing: Manejo de mayúsculas y minúsculas.
  - MaxLength: Número máximo de caracteres.
  - MinLines/MaxLines: Número mínimo/máximo de líneas visibles.
  - TextAlignment: Alineación del texto.

```
9      <TextBox Text="Texto Editable"  
10      CharacterCasing="Upper"  
11      MaxLength="10"  
12      MinLines="1"  
13      MaxLines="3"  
14      TextAlignment="Center"/>
```





## 3.2.4 - XAML: CONTROLES BÁSICOS EN WPF

### ■ PasswordBox

- Función: Control para manejar contraseñas.
- Propiedades Comunes:
  - PasswordChar: Carácter de enmascaramiento (usualmente '\*').
  - MaxLength: Número máximo de caracteres.

```
10 <PasswordBox PasswordChar="*"
11     MaxLength="8"/>
12
```



### Ejemplos de Layout en WPF

\*\*\*\*\*

## 3.2.4 - XAML: CONTROLES BÁSICOS EN WPF

### ■ Button

- Función: Control de botón que responde al evento de clic.
- Propiedades Comunes:
  - Content: Texto a mostrar.
  - HorizontalAlignment y VerticalAlignment: Controla la alineación horizontal y vertical del botón dentro de su contenedor.
  - Width y Height: Establece el ancho y alto del botón.
  - Background y Foreground: *Background* define el color de fondo del botón; *Foreground* define el color del texto o contenido del botón.
  - BorderBrush y BorderThickness: *BorderBrush* define el color del borde del botón; *BorderThickness* define el grosor del borde.
  - FontSize y FontWeight: *FontSize* define el tamaño de la fuerza del texto; *FontWeight* define el grosor de la fuerza del texto.
  - Style.Triggers y Trigger: *Style.Triggers* permite definir cambios en el estilo basados en ciertas condiciones; *Trigger* es una condición específica que, cuando se cumple, cambia el estilo según lo definido.

```
11 <Button Content="Botón Personalizado" HorizontalAlignment="Center" VerticalAlignment="Center" Width="200" Height="50">
12   <Button.Style>
13     <Style TargetType="Button">
14       <Setter Property="Background" Value="#LightGray"/>
15       <Setter Property="Foreground" Value="#Black"/>
16       <Setter Property="BorderBrush" Value="#Black"/>
17       <Setter Property="BorderThickness" Value="2"/>
18       <Setter Property="FontSize" Value="16"/>
19       <Setter Property="FontWeight" Value="Bold"/>
20     <Style.Triggers>
21       <Trigger Property="IsMouseOver" Value="True">
22         <Setter Property="Background" Value="#DarkGray"/>
23         <Setter Property="Foreground" Value="#White"/>
24       </Trigger>
25     </Style.Triggers>
26   </Style>
27 </Button.Style>
28 </Button>
```

Botón Personalizado

Botón Personalizado

## 3.2.5 - XAML: CONTROLES DE SELECCION EN WPF

### ■ CheckBox

- Función: Representa un control que el usuario puede marcar o desmarcar.
- Propiedades Comunes:
  - Content: Texto a mostrar.
  - IsChecked: Indica si está seleccionado.

```
12 ||| <CheckBox Content="Opción 1" IsChecked="True"/>
```

Ejemplos de Layout en WPF

☒ Opción 1

```
12 ||| <CheckBox Content="Opción 1" IsChecked="False"/>
```

Ejemplos de Layout en WPF

☐ Opción 1

## 3.2.5 - XAML: CONTROLES DE SELECCION EN WPF

### ■ RadioButton

- Función: Permite que un usuario seleccione una sola opción entre varias.
- Propiedades Comunes:
  - Content: Texto a mostrar en el botón.
  - GroupName: Especifica qué controles son mutuamente excluyentes.
  - IsChecked: Indica si está marcado.

```
12 <StackPanel>
13     <RadioButton Content="Opción A" GroupName="Grupo1" IsChecked="True"/>
14     <RadioButton Content="Opción B" GroupName="Grupo1"/>
15 </StackPanel>
```

### Ejemplos de Layout en WPF

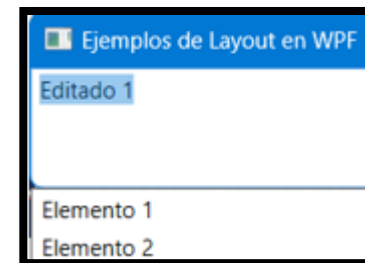
- ☒ Opción A
- ☐ Opción B

## 3.2.5 - XAML: CONTROLES DE SELECCION EN WPF

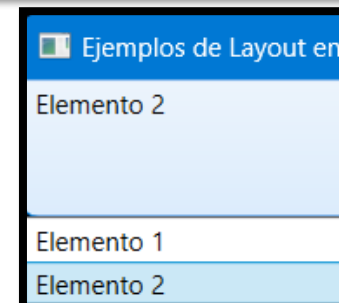
### ■ ComboBox

- Función: Control de selección con una lista desplegable.
- Propiedades Comunes:
  - ComboBoxItem: Elemento seleccionable dentro del ComboBox.
  - IsEditable: Habilita o deshabilita la edición del texto.
  - IsReadOnly: Establece un modo de solo selección.

```
13 <ComboBox IsEditable="True" IsReadOnly="False">
14     <ComboBoxItem Content="Elemento 1"/>
15     <ComboBoxItem Content="Elemento 2"/>
16 </ComboBox>
```



```
13 <ComboBox IsEditable="False" IsReadOnly="True">
14     <ComboBoxItem Content="Elemento 1"/>
15     <ComboBoxItem Content="Elemento 2"/>
16 </ComboBox>
```

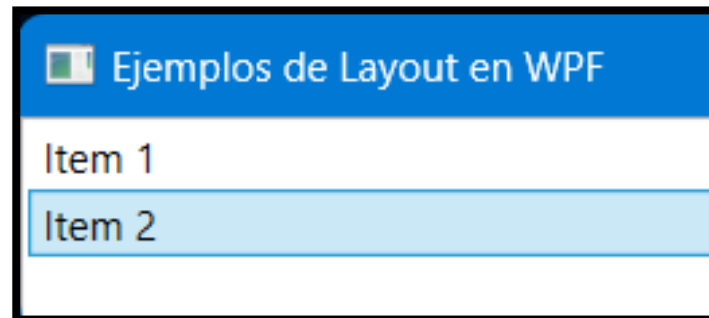


## 3.2.5 - XAML: CONTROLES DE SELECCION EN WPF

### ■ ListBox

- Función: Contiene una lista de elementos seleccionables.
- Propiedades Comunes:
  - ListBoxItem: Elemento seleccionable dentro de un ListBox.
  - SelectionMode: Determina cuántos elementos puede seleccionar un usuario.

```
14  <ListBox SelectionMode="Extended">
15      <ListBoxItem Content="Item 1"/>
16      <ListBoxItem Content="Item 2"/>
17  </ListBox>
```

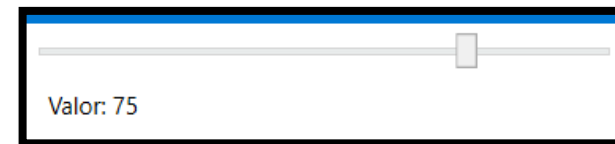


## 3.2.5 - XAML: CONTROLES DE SELECCION EN WPF

### ■ Slider

- Función: Permite seleccionar entre un rango de valores.
- Propiedades Comunes:
  - Value: Valor actual seleccionado.
  - Minimum y Maximum: Valores mínimo y máximo posible.
  - TickFrequency: Intervalo entre las marcas.
  - IsSnapToTickEnabled: Mueve automáticamente el cursor a la marca más cercana.

```
6 <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">
7 <!-- Slider -->
8 <Slider x:Name="mySlider" Minimum="0" Maximum="100" Value="50" TickFrequency="5" IsSnapToTickEnabled="True"
9 Orientation="Horizontal" Width="300"/>
10 <TextBlock Text="{Binding ElementName=mySlider, Path=Value, StringFormat='Valor: {0:F0}'}" Margin="10"/>
11 </StackPanel>
```



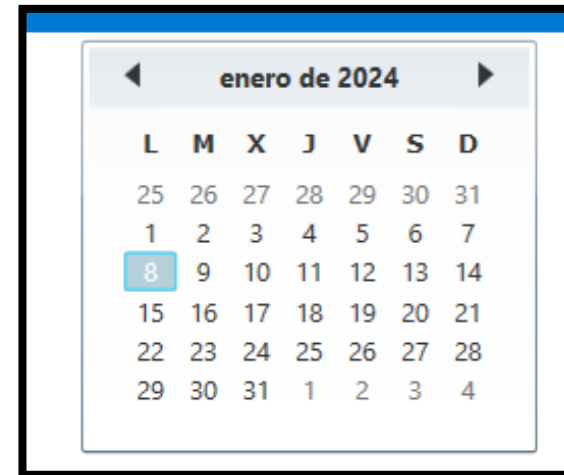


## 3.2.6 - XAML: PRESENTACIÓN Y SELECCIÓN DE FECHAS EN WPF

### ■ Calendar

- Función: Control que permite a los usuarios seleccionar una fecha mostrando un calendario.
- Propiedades Comunes:
  - SelectedDate: Fecha seleccionada actualmente.
  - DisplayMode: Indica si el calendario muestra un mes, año o década.
  - FirstDayOfWeek: Establece el día que se considera el inicio de la semana.
  - IsTodayHighlighted: Indica si la fecha actual está resaltada.

```
7 <Calendar SelectedDate="01/01/2024"  
8     DisplayMode="Month"  
9     FirstDayOfWeek="Monday"  
10    IsTodayHighlighted="True"/>
```

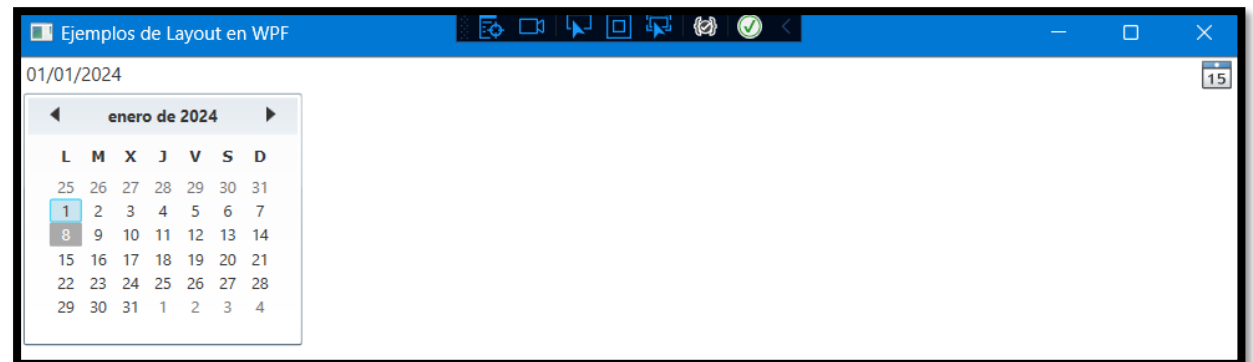


## 3.2.6 - XAML: PRESENTACIÓN Y SELECCIÓN DE FECHAS EN WPF

### ■ DatePicker

- Función: Control que facilita la selección de fechas por parte del usuario.
- Propiedades Comunes:
  - SelectedDate: Fecha seleccionada actualmente.
  - FirstDayOfWeek: Día que se considera el inicio de la semana.
  - IsTodayHighlighted: Si la fecha actual se resalta.
  - DisplayDateStart/End: Define el rango de fechas disponibles.

```
<DatePicker SelectedDate="01/01/2024"
             FirstDayOfWeek="Monday"
             IsTodayHighlighted="True"
             DisplayDateStart="2020-01-01"
             DisplayDateEnd="2024-12-31"/>
```

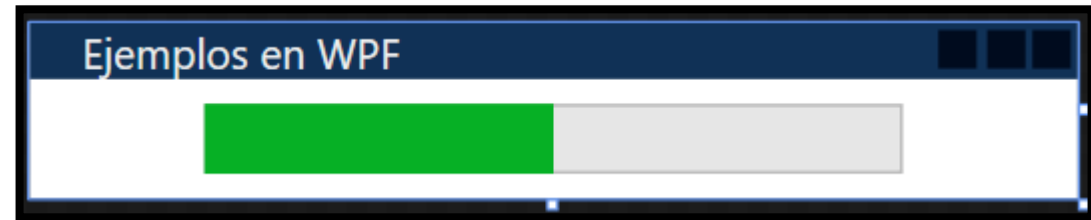


## 3.2.7 - XAML: CONTROLES DE INFORMACIÓN EN WPF

```
7 <ProgressBar Value="50" Minimum="0" Maximum="100" Height="20" Width="200" IsIndeterminate="False"/>
```

### ■ ProgressBar

- Función: Indica el progreso de una operación.
- Propiedades Comunes:
  - Value: Representa la magnitud actual del progreso.
  - Minimum y Maximum: Valores mínimo y máximo posibles.
  - IsIndeterminate: Muestra un progreso continuo genérico en lugar de valores reales.

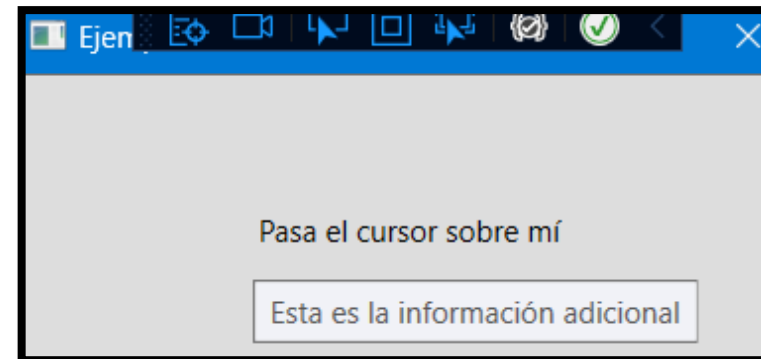


## 3.2.7 - XAML: CONTROLES DE INFORMACIÓN EN WPF

### ■ ToolTip

- Función: Crea una ventana emergente que muestra información para un elemento.
- Uso: Muy útil para proporcionar información adicional o instrucciones a los usuarios.

```
7 <Button Content="Pasa el cursor sobre mí">
8   <Button.ToolTip>
9     <ToolTip Content="Esta es la información adicional" />
10  </Button.ToolTip>
11 </Button>
```

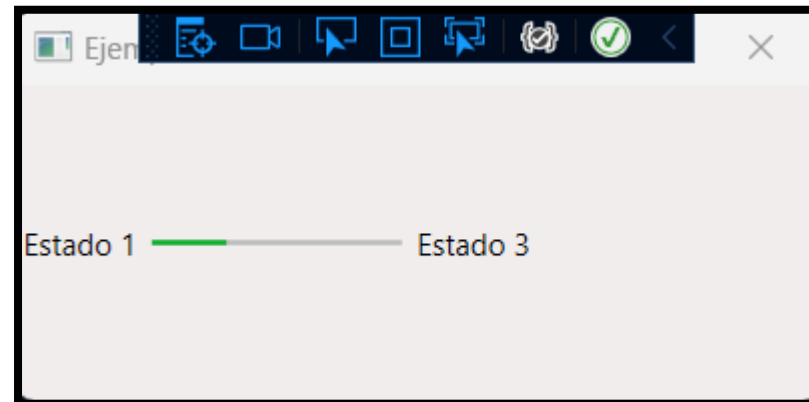


## 3.2.7 - XAML: CONTROLES DE INFORMACIÓN EN WPF

### ■ StatusBar

- Función: Muestra elementos e información en una barra horizontal en la ventana de una aplicación.
- Elementos: Puede contener varios *StatusBarItem* para mostrar diferentes tipos de información o controles.

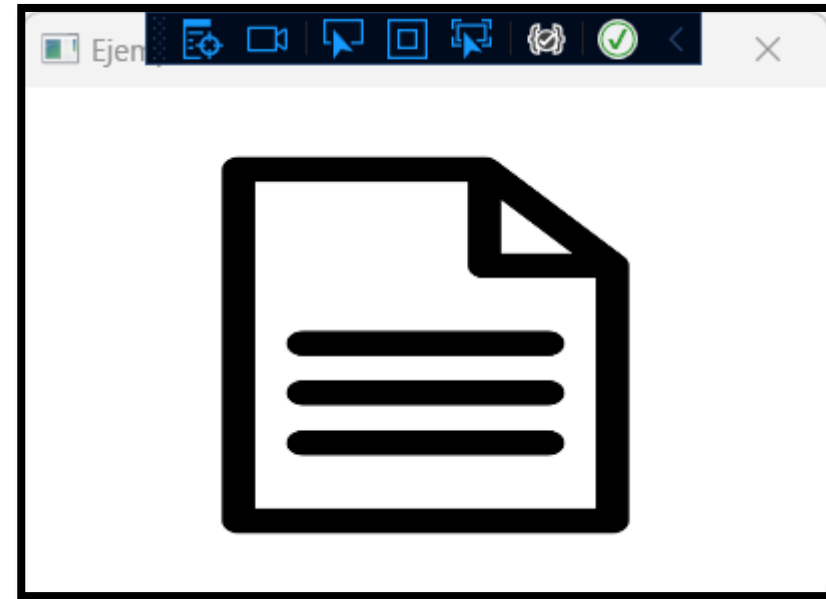
```
7 <StatusBar>
8   <StatusBarItem Content="Estado 1" />
9   <StatusBarItem>
10     <ProgressBar Value="30" Width="100"/>
11   </StatusBarItem>
12   <StatusBarItem Content="Estado 3" />
13 </StatusBar>
```



## 3.2.8 - XAML: CONTROLES DE IMÁGENES EN WPF

### ■ Image

- Función: Representa un control que muestra una imagen.
- Propiedades Comunes:
  - Source: Establece el origen de la imagen.
  - Stretch: Describe cómo se debe estirar una imagen para llenar el rectángulo de destino.



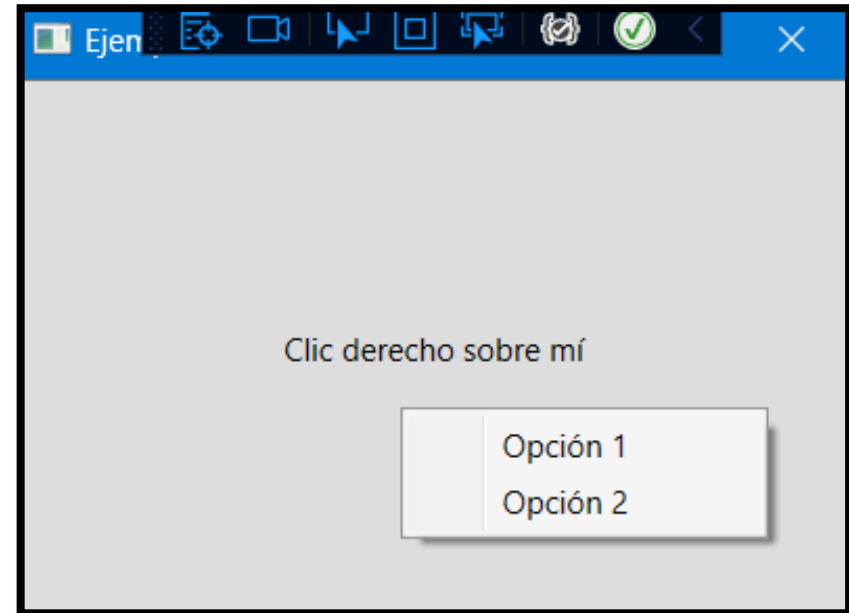
```
7 ||| <Image Source="C:\Users\dfere\Desktop\304579.png" Stretch="Fill" Width="200" Height="150"/>
```

## 3.2.9 - XAML: CONTROLES DE MENÚ EN WPF

### ■ ContextMenu

- Función: Representa un menú emergente que permite que un control exponga funcionalidad específica del contexto del control.
- Propiedades Comunes:
  - MenuItem: Representa un elemento seleccionable dentro de un menú.

```
7  <Button Content="Clic derecho sobre mí">
8      <Button.ContextMenu>
9          <ContextMenu>
10             <MenuItem Header="Opción 1"/>
11             <MenuItem Header="Opción 2"/>
12          </ContextMenu>
13      </Button.ContextMenu>
14 </Button>
```





## 3.2.9 - XAML: CONTROLES DE MENÚ EN WPF

### ■ ToolBar

- Función: Proporciona un contenedor para un grupo de comandos o controles.
- Propiedades Comunes:
  - ToolBarTray: Representa el contenedor que controla el diseño de una barra de herramientas.
  - Band: Indica dónde debe ubicarse la barra de herramientas dentro del ToolBarTray.

```
7  <ToolBarTray>
8      <ToolBar Band="1">
9          <Button Content="Botón 1"/>
10         <Separator/>
11         <Button Content="Botón 2"/>
12     </ToolBar>
13 </ToolBarTray>
```

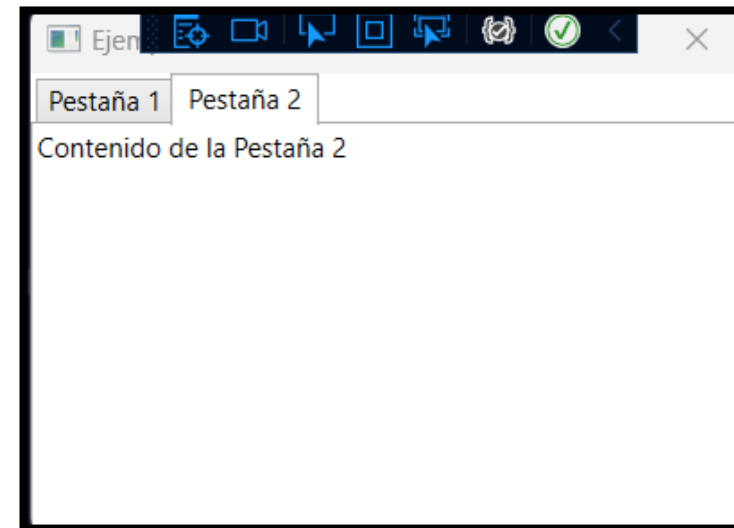


## 3.2.10 - XAML: CONTROLES DE NAVEGACIÓN EN WPF

### ■ TabControl

- **Función:** Permite la organización de contenido en pestañas, cada una conteniendo diferentes vistas o controles.
- **Propiedades Comunes:**
  - TabItem: Cada pestaña dentro del control.
  - IsSelected: Indica si una pestaña está seleccionada.

```
7 <TabControl>
8   <TabItem Header="Pestaña 1" IsSelected="True">
9     <TextBlock Text="Contenido de la Pestaña 1"/>
10  </TabItem>
11  <TabItem Header="Pestaña 2">
12    <TextBlock Text="Contenido de la Pestaña 2"/>
13  </TabItem>
14 </TabControl>
```

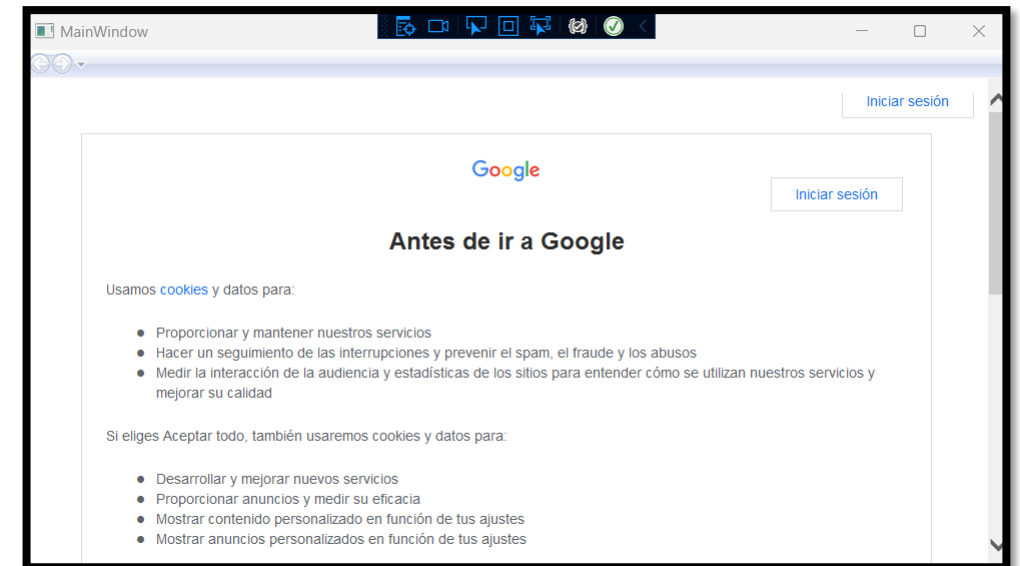


## 3.2.10 - XAML: CONTROLES DE NAVEGACIÓN EN WPF

### ■ Frame

- **Función:** Se utiliza para la navegación y la presentación de páginas dentro de una ventana.
- **Características:**
  - Permite cargar contenido dinámicamente.
  - Soporta navegación hacia adelante y hacia atrás como en un navegador web.

```
10 <Frame Source="http://google.com" NavigationUIVisibility="Visible"/>
```



## 3.2.11 - XAML: RECURSOS Y ESTILOS

- Los recursos, son objetos reutilizables en distintas partes de una aplicación.

- Tipos de Recursos en WPF:

- Recursos XAML
- Archivos de Datos de Recursos

- Un ejemplo de recursos en *XAML* son los estilos.

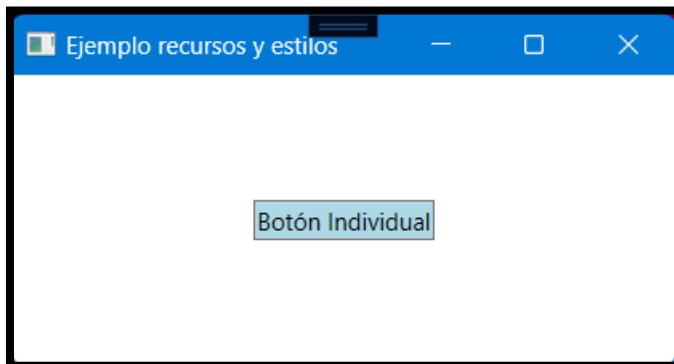
- Funcionalidad de los Estilos:

- Pueden definirse para diferentes ámbitos.
- Permiten su uso limitado a controles específicos bajo demanda.

## 3.2.11 - XAML: RECURSOS Y ESTILOS

### ■ Estilos - Control

- Se pueden definir estilos específicos para un solo control, afectando su apariencia y comportamiento.

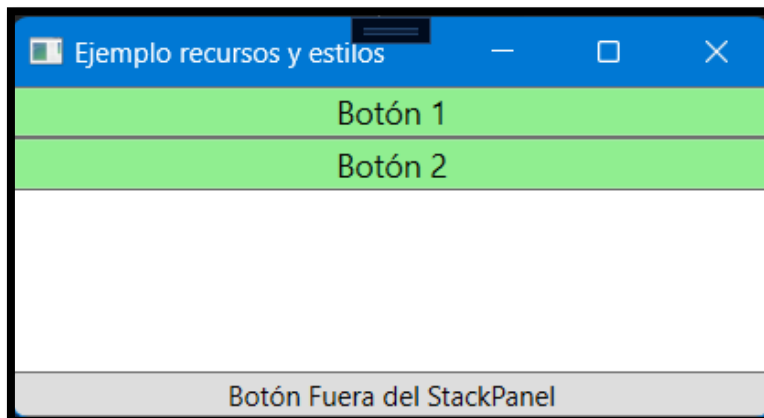


```
11 <Grid>
12   <Button Content="Botón Individual" HorizontalAlignment="Center" VerticalAlignment="Center">
13     <Button.Style>
14       <Style TargetType="Button">
15         <Setter Property="Background" Value="LightBlue"/>
16         <Setter Property="FontSize" Value="12"/>
17       </Style>
18     </Button.Style>
19   </Button>
20 </Grid>
```

## 3.2.11 - XAML: RECURSOS Y ESTILOS

### ■ Estilos - Controles Hijos

- Utilizando la sección 'Resources' de un control, se pueden definir estilos que afecten tanto al control como a sus controles hijos.

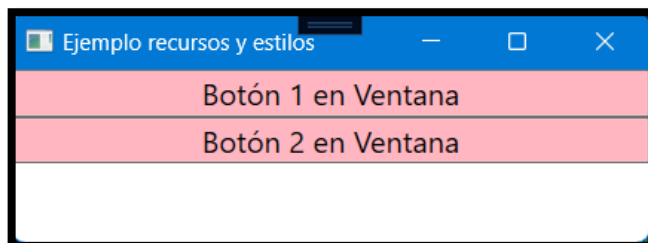


```
11 <Grid>
12 <!-- Contenedor principal -->
13 <StackPanel>
14 <!-- Contenedor de botones con estilos -->
15 <StackPanel.Resources>
16 <Style TargetType="Button">
17 <Setter Property="Background" Value="■"LightGreen"/>
18 <Setter Property="FontSize" Value="14"/>
19 </Style>
20 </StackPanel.Resources>
21
22 <Button Content="Botón 1"/>
23 <Button Content="Botón 2"/>
24 </StackPanel>
25
26 <Button Content="Botón Fuera del StackPanel" VerticalAlignment="Bottom"/>
27 </Grid>
```

## 3.2.11 - XAML: RECURSOS Y ESTILOS

### ■ Estilos - Ventana

- Definiendo estilos en '*Window.Resources*', se puede aplicar un estilo a todos los controles de una ventana específica.

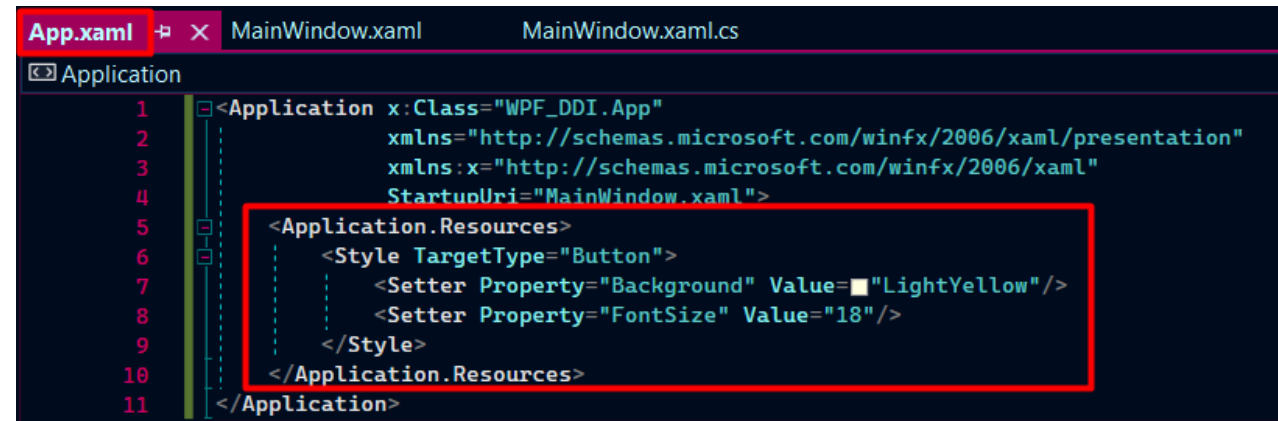


```
11 <Window.Resources>
12     <!-- Estilo que se aplicará a todos los botones dentro de esta ventana -->
13     <Style TargetType="Button">
14         <Setter Property="Background" Value="LightPink"/>
15         <Setter Property="FontSize" Value="16"/>
16     </Style>
17 </Window.Resources>
18
19 <StackPanel>
20     <!-- Botones dentro de la ventana que tendrán el estilo aplicado -->
21     <Button Content="Botón 1 en Ventana"/>
22     <Button Content="Botón 2 en Ventana"/>
23 </StackPanel>
```

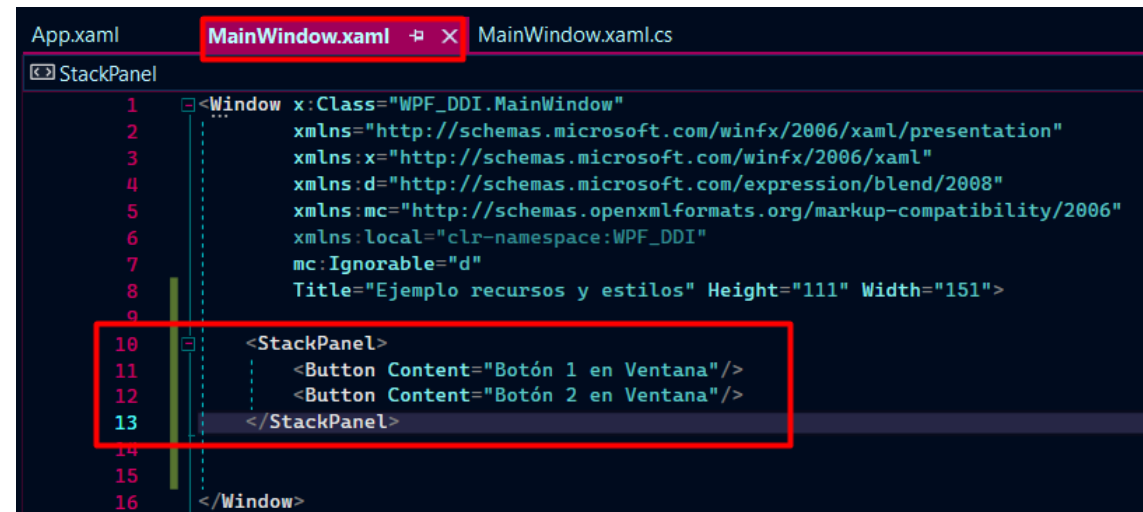
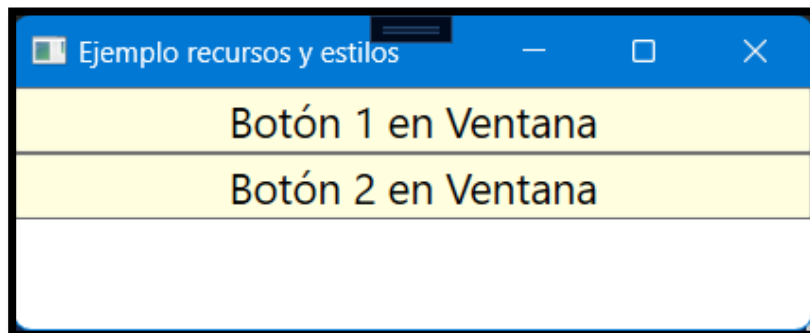
## 3.2.11 - XAML: RECURSOS Y ESTILOS

### ■ Estilos - Aplicación

- Para aplicar estilos a través de toda la aplicación, se definen en '*App.xaml*' usando la sección '*Application.Resources*'.



```
App.xaml  MainWindow.xaml  MainWindow.xaml.cs
Application
1  <Application x:Class="WPF_DDI.App"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      StartupUri="MainWindow.xaml">
5      <Application.Resources>
6          <Style TargetType="Button">
7              <Setter Property="Background" Value="LightYellow"/>
8              <Setter Property="FontSize" Value="18"/>
9          </Style>
10     </Application.Resources>
11 </Application>
```



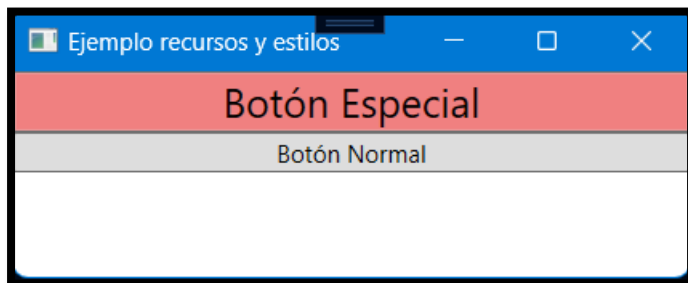
```
App.xaml  MainWindow.xaml  MainWindow.xaml.cs
StackPanel
1  <Window x:Class="WPF_DDI.MainWindow"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6      xmlns:local="clr-namespace:WPF_DDI"
7      mc:Ignorable="d"
8      Title="Ejemplo recursos y estilos" Height="111" Width="151">
9      <StackPanel>
10         <Button Content="Botón 1 en Ventana"/>
11         <Button Content="Botón 2 en Ventana"/>
12     </StackPanel>
13 </Window>
```



## 3.2.11 - XAML: RECURSOS Y ESTILOS

### ■ Estilos - Declaración Explícita

- Al establecer la propiedad *x:Key* en un estilo, se indica que solo se usará cuando se haga referencia explícita en un control específico.



```
10 <Window.Resources>
11     <Style x:Key="BotonEspecial" TargetType="Button">
12         <Setter Property="Background" Value="LightCoral"/>
13         <Setter Property="FontSize" Value="20"/>
14     </Style>
15 </Window.Resources>
16 <StackPanel>
17     <Button Content="Botón Especial" Style="{StaticResource BotonEspecial}"/>
18     <Button Content="Botón Normal"/>
19 </StackPanel>
20
```

## 3.2.11 - XAML: RECURSOS Y ESTILOS

### ■ Estilos - Fuentes Personalizadas

- Se pueden agregar fuentes personalizadas a la aplicación, creando una carpeta 'Fonts' en el proyecto y definiendo un estilo para ellas.



```
App.xaml  MainWindow.xaml  MainWindow.xaml.cs
Application
1  <Application x:Class="WPF_DDI.App"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:local="clr-namespace:WPF_DDI"
5      StartupUri="MainWindow.xaml">
6      <Application.Resources>
7          <Style TargetType="TextBlock">
8              <Setter Property="TextElement.FontFamily" Value="pack://application:,,,/Fonts/#FREEDOM"/>
9              <Setter Property="FontSize" Value="40"/>
10          </Style>
11      </Application.Resources>
12  </Application>
13
14
15
```

```
App.xaml  MainWindow.xaml  MainWindow.xaml.cs
Window
1  <Window x:Class="WPF_DDI.MainWindow"
2      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6      xmlns:local="clr-namespace:WPF_DDI"
7      mc:Ignorable="d"
8      Title="Ejemplo recursos y estilos" Height="111" Width="151">
9      <StackPanel Margin="10">
10          <TextBlock Text="Prueba"/>
11      </StackPanel>
12
13
14
15  </Window>
```

## 3.2.12 - XAML: EVENTOS

- Los **eventos** desempeñan un **papel fundamental** en el **desarrollo** de **aplicaciones WPF**.
  - Estos eventos **permiten** que nuestra **aplicación responda** a las **interacciones** del **usuario** de **manera eficiente**.
- 
- En **WPF**, la **mayoría** de los **elementos** de la **interfaz de usuario (UI)** están **impulsados** por **eventos**.
  - **Cada control**, incluida la **ventana principal (Window)**, **expone** una **serie** de **eventos** a los que **podemos suscribirnos**.
  - Esto **significa** que nuestra **aplicación** puede ser **notificada cuando** **ocurran eventos** y **tomar medidas** en **consecuencia**.

## 3.2.12 - XAML: EVENTOS

- Para manejar eventos en **WPF**, debemos **crear** un método en nuestro código C# (*archivo .cs*) con una **estructura específica**:
  - NombreDelMetodo: El nombre que le damos al método, que indica la acción que se llevará a cabo cuando ocurra el evento.
  - sender: Es el objeto que generó el evento, lo que nos permite identificar la fuente del evento.
  - EventArgs: Puede ser un objeto que contiene información adicional sobre el evento, dependiendo del tipo de evento.

```
private void NombreDelMetodo(object sender, EventArgs e)  
{  
    // Aquí colocamos el código que queremos ejecutar cuando el método entre en acción  
}
```

## 3.2.12 - XAML: EVENTOS

- Para que un **método creado** se **asocie** a un **evento** de un **control** en la **interfaz de usuario**, debemos **configurarlo** en el **archivo XAML** del **control**
  - **Control**: El nombre del control al que **deseamos asociar** el **evento**.
  - **Evento**: El nombre del **evento** que **queremos manejar** (por ejemplo, "**Click**" para el **evento** de clic de un botón).
  - **NombreDelMetodo**: El nombre del **método** que **hemos creado** para **manejar** el **evento**.

```
<Control ... Evento="NombreDelMetodo">
```

## 3.2.12 - XAML: EVENTOS

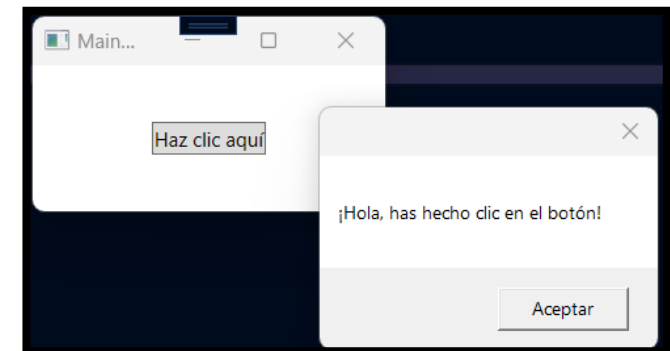
- Ejemplo 1:
  - Supongamos que **tenemos** un **botón** en nuestra aplicación y queremos **responder** cuando el usuario haga **clíc** en él.

- 1º Creamos un método en el código C#:

- 2º Asociamos este método al evento de clic del botón en el archivo XAML:

```
1 using System.Windows;
2
3 namespace WpfApp3
4 {
5     2 referencias
6     public partial class MainWindow : Window
7     {
8         0 referencias
9         public MainWindow()
10         {
11             InitializeComponent();
12         }
13
14         1 referencia
15         private void btnEjemplo_Click(object sender, RoutedEventArgs e)
16         {
17             MessageBox.Show("¡Hola, has hecho clic en el botón!");
18         }
19     }
20 }
```

```
<Window x:Class="WpfApp3.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="350" Width="525">
  <Grid>
    <Button Name="btnEjemplo" Content="Haz clic aquí" HorizontalAlignment="Center" VerticalAlignment="Center" Click="btnEjemplo_Click"/>
  </Grid>
</Window>
```



## 3.2.12 - XAML: EVENTOS

- **Ejemplo 2:**

- Supongamos que queremos utilizar el evento *MouseEnter* de un *Label*. En este ejemplo, cuando el usuario mueve el cursor sobre la etiqueta, el texto de la etiqueta cambiará.

- **1º Creamos un método en el código C#:**

```
private void lblEjemplo_MouseEnter(object sender, MouseEventArgs e)
{
    lblEjemplo.Content = "¡El cursor está aquí!";
}
```

- **2º Asociamos este método al evento de clic del botón en el archivo XAML:**

```
<Grid>
    <Label Name="lblEjemplo" Content="Pasa el cursor por aquí" HorizontalAlignment="Center" VerticalAlignment="Center" MouseEnter="lblEjemplo_MouseEnter"/>
</Grid>
```

Pasa el cursor por aquí



¡El cursor está aquí!

## 3.2.13 - XAML: DATA BINDING

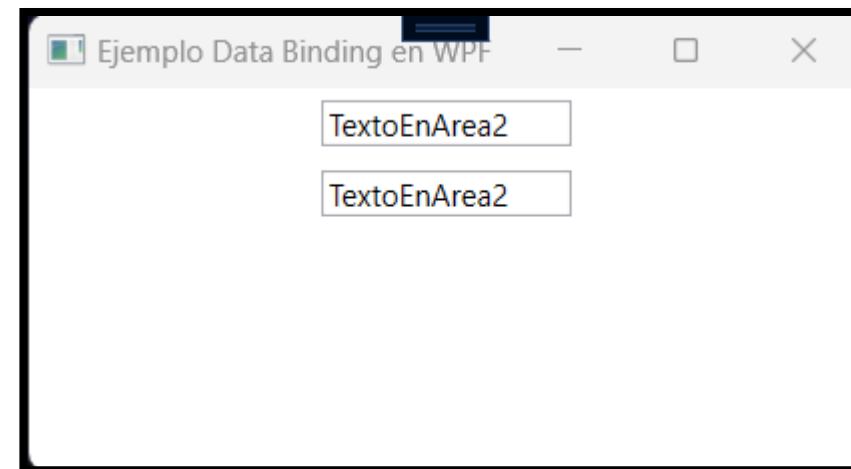
### ■ Data Binding en WPF

- El "**Data Binding**" es una técnica esencial en WPF que vincula la UI con fuentes de datos.
- Es ideal para reflejar automáticamente los cambios en la interfaz de usuario o la fuentes de datos.

### ■ Elementos clave del Data Binding:

- **ElementName:** Se refiere al nombre del control al que está enlazado.
- **Path:** Define la propiedad del elemento de datos que se enlaza.
- **Mode:** Establece la dirección del enlace, que puede ser *OneWay*, *TwoWay*, etc.

```
--  
11 <StackPanel HorizontalAlignment="Center">  
12     <TextBox Name="Area1" Width="100" Margin="5"  
13         Text="{Binding ElementName=Area2, Path=Text, Mode=OneWay}"/>  
14     <TextBox Name="Area2" Width="100" Margin="5" Text="Inicio"/>  
15 </StackPanel>  
--
```





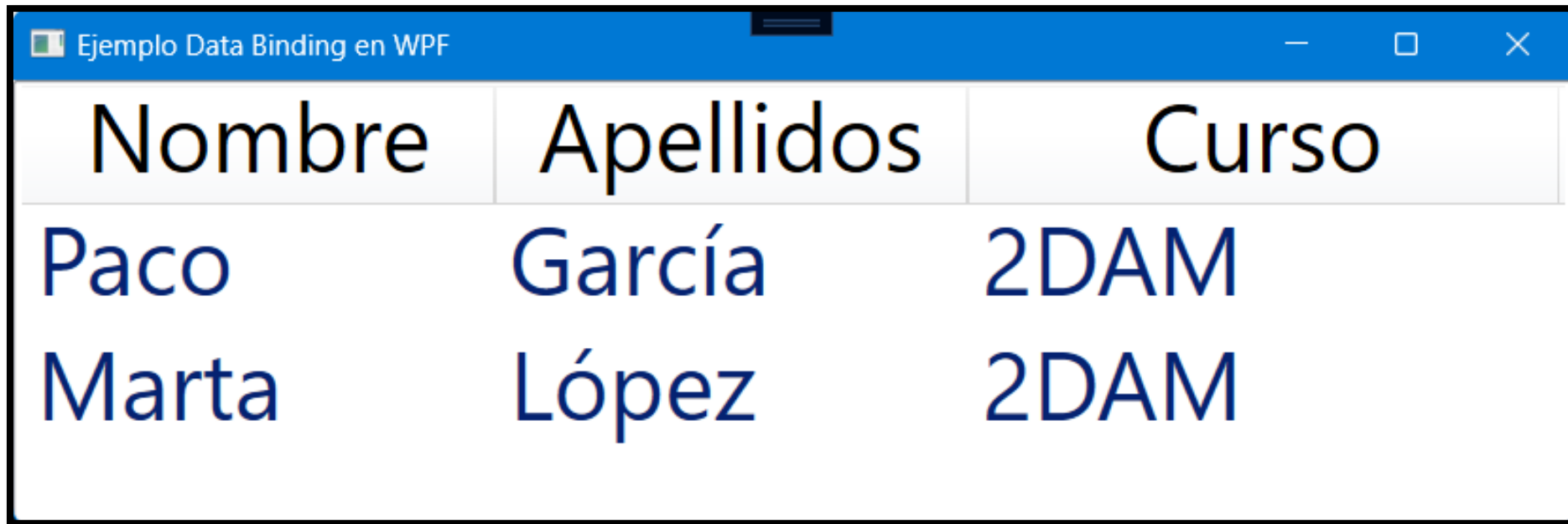
## 3.2.13 - XAML: DATA BINDING

- Data Binding - GridView en WPF
  - El *GridView* es un control de WPF que muestra datos en formato de tabla y es perfecto para listados detallados.

```
App.xaml | MainWindow.xaml | MainWindow.xaml.cs
Window
1 <Window x:Class="WPF_DDI.MainWindow"
2   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6   xmlns:local="clr-namespace:WPF_DDI"
7   mc:Ignorable="d"
8   Title="Ejemplo Data Binding en WPF" Height="111" Width="202">
9
10
11 <ListView Name="LstPersonas" FontSize="40">
12   <ListView.View>
13     <GridView ColumnHeaderToolTip="Personas">
14       <GridViewColumn Header="Nombre" Width="200" DisplayMemberBinding="{Binding Nombre}"/>
15       <GridViewColumn Header="Apellidos" Width="200" DisplayMemberBinding="{Binding Apellidos}"/>
16       <GridViewColumn Header="Curso" Width="250" DisplayMemberBinding="{Binding Curso}"/>
17     </GridView>
18   </ListView.View>
19 </ListView>
20
21
22 </Window>
23
```

```
App.xaml | MainWindow.xaml | MainWindow.xaml.cs | WPF-DDI
WPF-DDI
16 namespace WPF_DDI
17 {
18     4 referencias
19     public class Persona
20     {
21         2 referencias
22         public string Nombre { get; set; }
23         2 referencias
24         public string Apellidos { get; set; }
25         2 referencias
26         public string Curso { get; set; }
27     }
28
29     2 referencias
30     public partial class MainWindow : Window
31     {
32         ObservableCollection<Persona> ListaPersonas = new ObservableCollection<Persona>();
33
34         0 referencias
35         public MainWindow()
36         {
37             InitializeComponent();
38             ListaPersonas.Add(new Persona { Nombre = "Paco", Apellidos = "García", Curso = "2DAM" });
39             ListaPersonas.Add(new Persona { Nombre = "Marta", Apellidos = "López", Curso = "2DAM" });
40             LstPersonas.ItemsSource = ListaPersonas;
41         }
42     }
43 }
```

### 3.2.13 - XAML: DATA BINDING



The image shows a screenshot of a Windows Presentation Foundation (WPF) application window. The window's title bar is blue and contains the text 'Ejemplo Data Binding en WPF' along with standard minimize, maximize, and close buttons. The main content area of the window displays a table with three columns: 'Nombre', 'Apellidos', and 'Curso'. The table has two data rows. The first row contains the values 'Paco', 'García', and '2DAM'. The second row contains the values 'Marta', 'López', and '2DAM'. The text in the table is dark blue.

Nombre	Apellidos	Curso
Paco	García	2DAM
Marta	López	2DAM

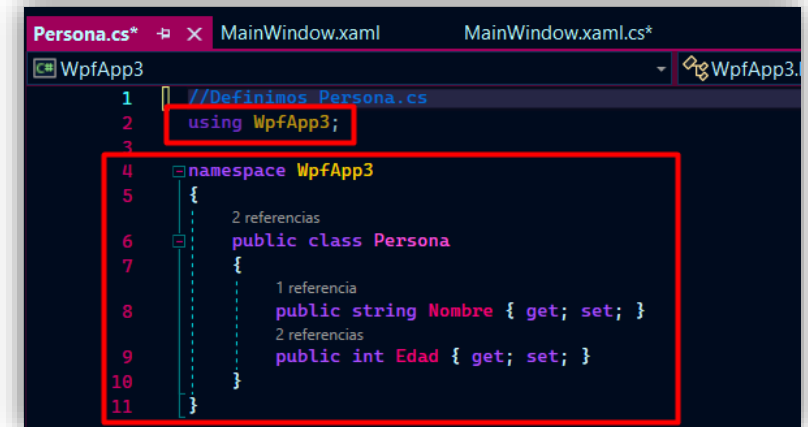
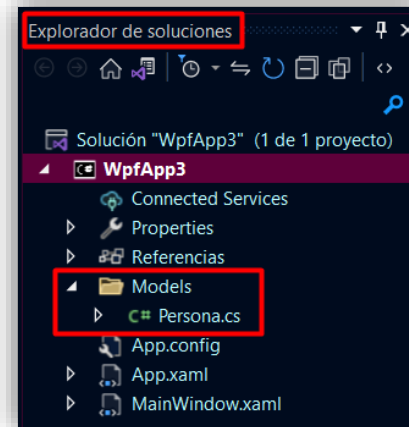
## 3.2.14 - XAML: DATA CONTEXT

- ¿Qué es?
  - Es la fuelle predeterminada para los enlaces (*bindings*) de datos en *WPF*.
- Herencia
  - Definido en *FrameworkElement*, heredado por controles de *UI*, incluyendo *Window*.
  - Heredable a través de la jerarquía de controles: si se establece en el nivel de la ventana, todos los controles hijos pueden utilizarlo.
- Flexibilidad
  - Cada control puede tener su propio *DataContext*, permitiendo diferentes fuentes de datos para diferentes partes de la *UI*.
  - Esto es útil para tener un contexto global y contextos locales más específicos.

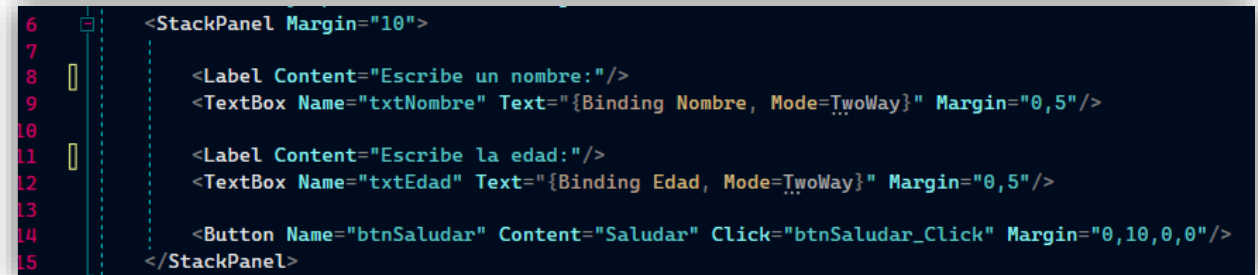
## 3.2.14 - XAML: DATA CONTEXT

### ■ Ejemplo práctico

- Primero, definimos la clase *Persona* que actuará como el modelo de datos y la situamos en una nueva carpeta dentro del proyecto llamada *Model* en un fichero llamado *Persona.cs*



- Luego, en el archivo XAML, configuramos los controles y enlazamos sus propiedades con el modelo de datos usando *Binding*.



- Aquí, por ejemplo, un *TextBox* se enlaza a la propiedad *Nombre* de la clase *Persona*

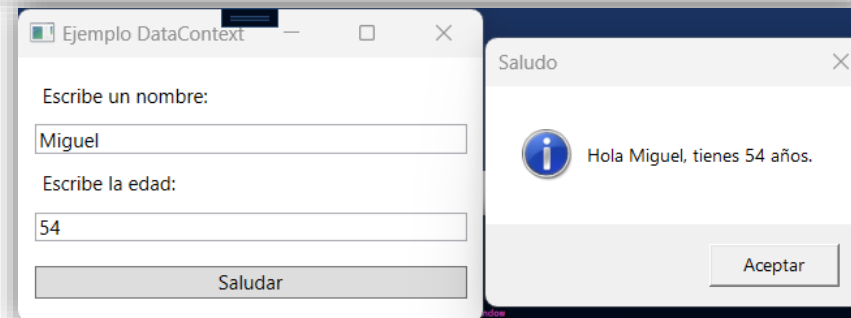
## 3.2.14 - XAML: DATA CONTEXT

### ■ Ejemplo práctico

- En el código que controla la lógica de **XAML**, por ahora, **MainWindow.xaml.cs**, configuramos el **DataContext** de la ventana para manejar el evento **Button** para mostrar un mensaje

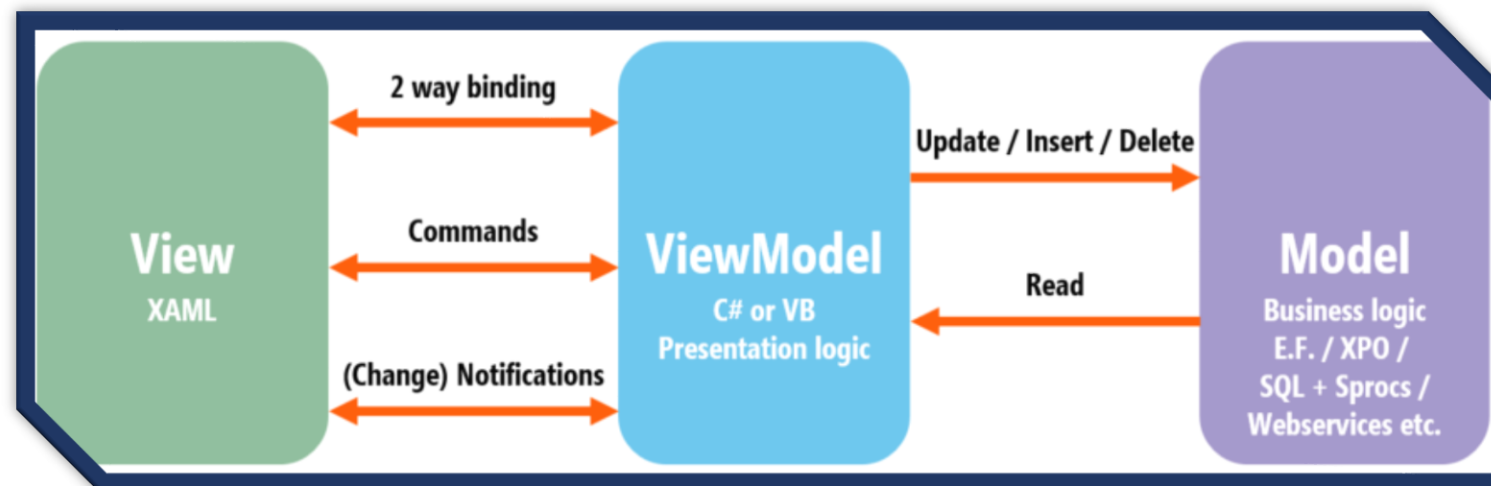
```
1 using System;
2 using System.Windows;
3 using WpfApp3;
4
5 namespace WpfApp3
6 {
7     2 referencias
8     public partial class MainWindow : Window
9     {
10         private Persona persona = new Persona();
11
12         0 referencias
13         public MainWindow()
14         {
15             InitializeComponent();
16             this.DataContext = persona;
17         }
18
19         1 referencia
20         private void btnSaludar_Click(object sender, RoutedEventArgs e)
21         {
22             if (int.TryParse(txtEdad.Text, out int edad))
23             {
24                 persona.Edad = edad;
25                 MessageBox.Show($"Hola {persona.Nombre}, tienes {persona.Edad} años.", "Saludo", MessageBoxButton.OK, MessageBoxImage.Information);
26             }
27             else
28             {
29                 MessageBox.Show("Por favor, ingresa una edad válida.", "Error", MessageBoxButton.OK, MessageBoxImage.Error);
30             }
31         }
32     }
33 }
```

- Podemos ver el resultado de la compilación



### 3.3.1 – MVVM: SEPARACIÓN Y CLARIDAD EN WPF

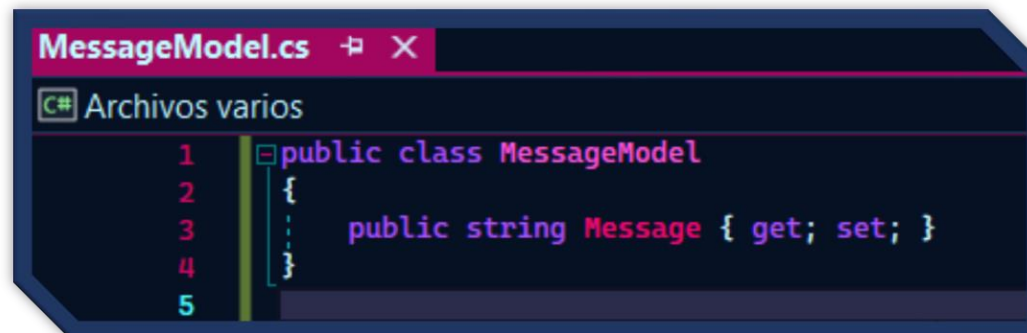
- El patrón **MVVM** es fundamental en el desarrollo de aplicaciones WPF, ofreciendo una separación clara entre: la **interfaz gráfica** (*View*), la **lógica de negocio** (*ViewModel*) y los **datos** (*Model*).
- Este patrón no solo facilita el **mantenimiento** y la **escalabilidad** de las aplicaciones, sino que también mejora la capacidad de **prueba** y la **limpieza** del código.



## 3.3.2 – MVVM: COMPONENTES DE MVVM

### ■ Modelo (Model)

- Representa la **lógica de negocio** y los **datos**.
- **No** tiene **conocimiento directo** de la vista o del ViewModel.
- Puede incluir clases que representan **datos reales** y la **lógica** para interactuar con estos **datos** (*por ejemplo, lógica de negocios, acceso a bases de datos, etc.*).

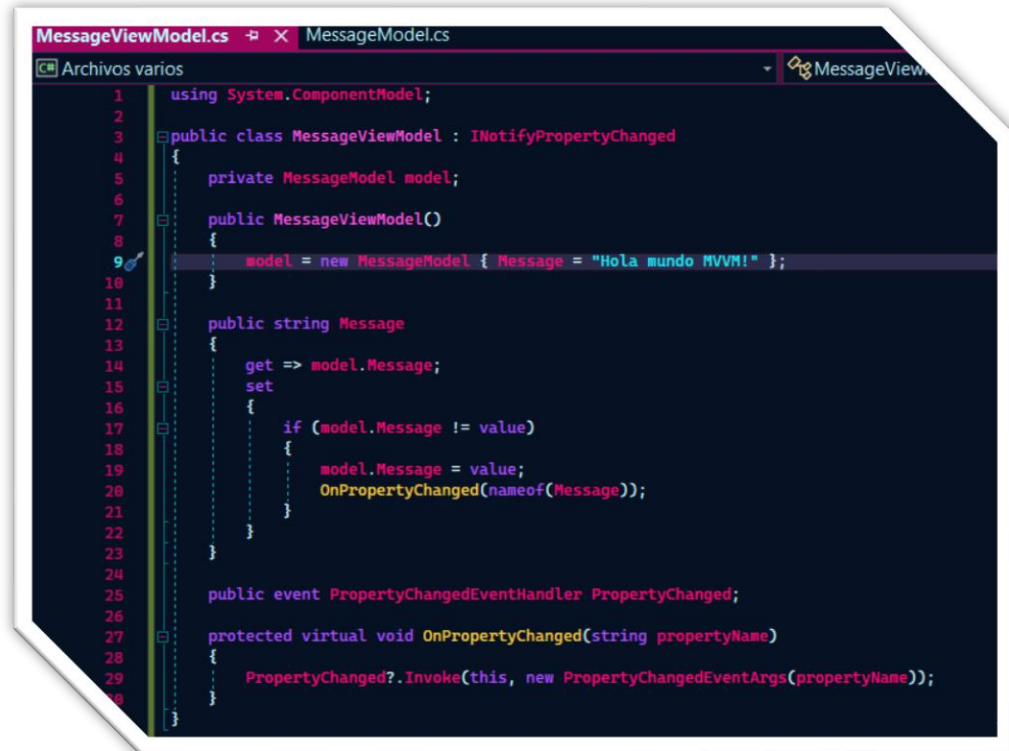


```
MessageModel.cs  + X
C# Archivos varios
1 public class MessageModel
2 {
3     public string Message { get; set; }
4 }
5
```

## 3.3.2 – MVVM: COMPONENTES DE MVVM

### ■ ViewModel:

- Es el **intermediario** entre la **Vista** y el **Modelo**.
- Contiene la **lógica de presentación** y los **comandos** que la vista puede usar para interactuar con los datos.
- Proporciona **propiedades** y **comandos** a los cuales la **Vista** puede **enlazar**.
- Es consciente de la **Vista** pero **no tiene conocimiento** directo sobre la **misma**, lo que facilita la **prueba unitaria** del *ViewModel*.



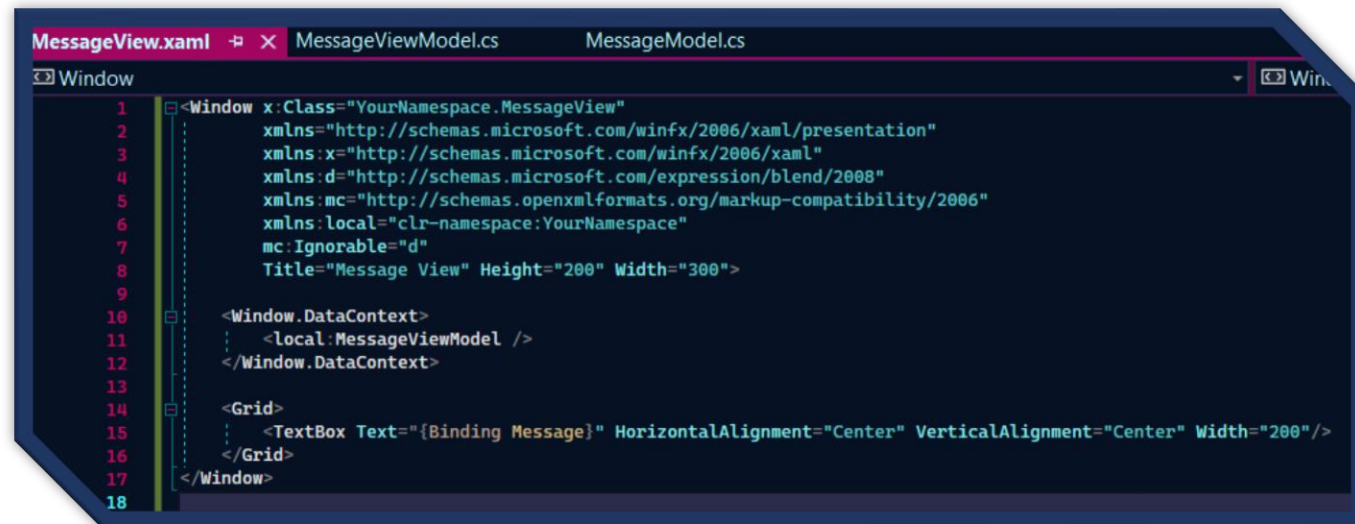
```
MessageViewModel.cs x MessageModel.cs
Archivos varios MessageView
1 using System.ComponentModel;
2
3 public class MessageViewModel : INotifyPropertyChanged
4 {
5     private MessageModel model;
6
7     public MessageViewModel()
8     {
9         model = new MessageModel { Message = "Hola mundo MVVM!" };
10    }
11
12    public string Message
13    {
14        get => model.Message;
15        set
16        {
17            if (model.Message != value)
18            {
19                model.Message = value;
20                OnPropertyChanged(nameof(Message));
21            }
22        }
23    }
24
25    public event PropertyChangedEventHandler PropertyChanged;
26
27    protected virtual void OnPropertyChanged(string propertyName)
28    {
29        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
30    }
31 }
```



## 3.3.2 – MVVM: COMPONENTES DE MVVM

### ■ Vista (View):

- Es la **representación visual** de los **datos** (*la interfaz de usuario*).
- En *WPF*, esto se **define** generalmente a través de **XAML**.
- La vista está diseñada para **no contener lógica** que maneje **datos** o **estados** de negocio.



The screenshot shows a code editor with three tabs: MessageView.xaml, MessageViewModel.cs, and MessageModel.cs. The MessageView.xaml tab is active, displaying XAML code for a WPF window. The code defines a window with a title bar, a data context, and a grid containing a text box.

```
1 <Window x:Class="YourNamespace.MessageView"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
5       xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
6       xmlns:local="clr-namespace:YourNamespace"
7       mc:Ignorable="d"
8       Title="Message View" Height="200" Width="300">
9
10    <Window.DataContext>
11        <local:MessageViewModel />
12    </Window.DataContext>
13
14    <Grid>
15        <TextBox Text="{Binding Message}" HorizontalAlignment="Center" VerticalAlignment="Center" Width="200"/>
16    </Grid>
17 </Window>
18
```

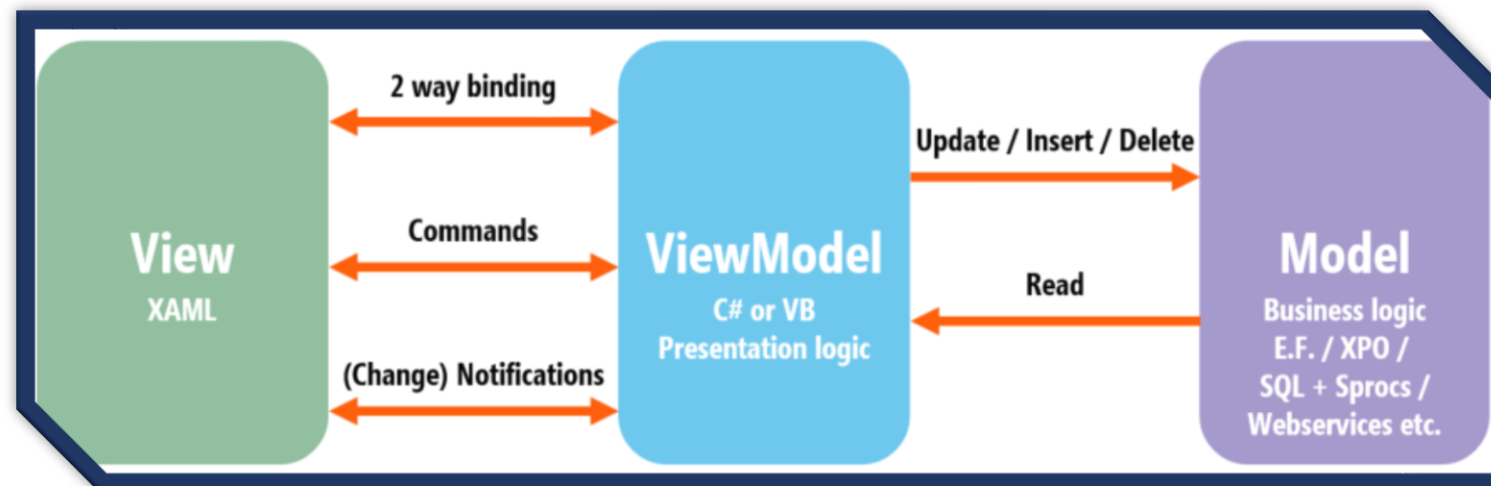
### 3.3.3 – MVVM: ENLACE (DATA BINDING)

- Enlace de Datos:

- En *MVVM*, la *Vista* se enlaza (binds) a *propiedades* y *comandos* en el *ViewModel*.
- Esto se hace a través de *Data Binding* en *WPF*, permitiendo una *sincronización automática* de la *UI* con el estado del *ViewModel*.
- Por ejemplo, si una *propiedad* en el *ViewModel* se actualiza, la *Vista* *refleja automáticamente* estos *cambios*.

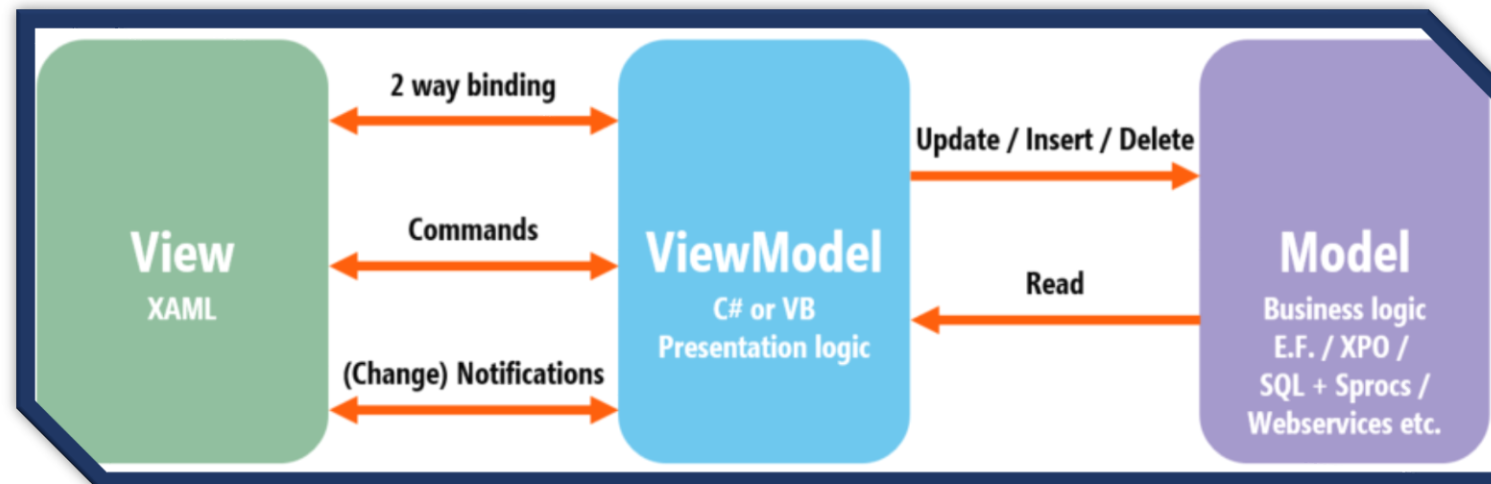
### 3.3.4 – MVVM: COMUNICACIÓN ENTRE COMPONENTES

- De *Model* a *ViewModel*:
  - El *ViewModel* puede suscribirse a cambios en el *Model*, o simplemente hacer uso del *Model* para obtener y manipular **datos**.



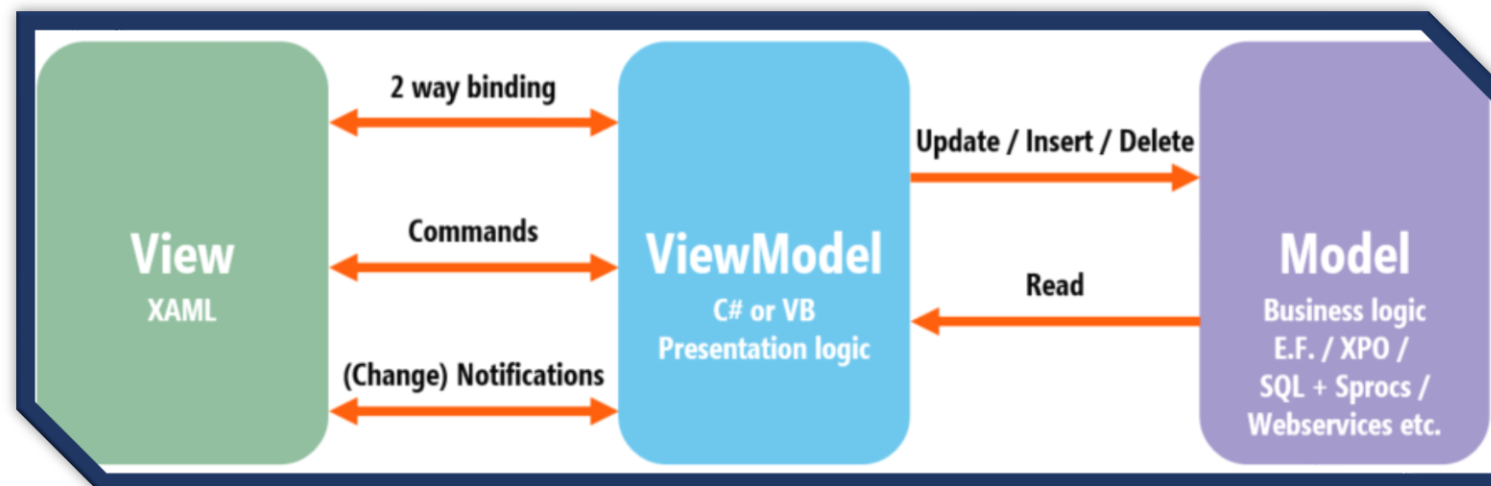
### 3.3.4 – MVVM: COMUNICACIÓN ENTRE COMPONENTES

- De ViewModel a Vista:
  - Se utiliza el enlace de datos.
  - La Vista se actualiza automáticamente cuando las propiedades del ViewModel cambian.
  - La Vista puede enlazar comandos del ViewModel a eventos de la interfaz, como clics de botones.



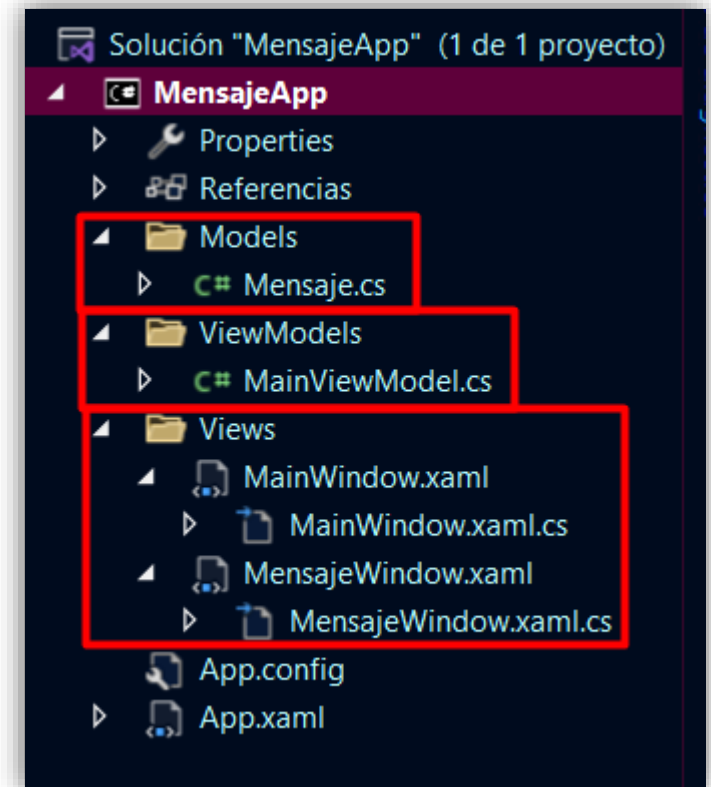
### 3.3.4 – MVVM: COMUNICACIÓN ENTRE COMPONENTES

- De Vista a ViewModel:
  - Se realiza a través de enlaces (*bindings*), especialmente en acciones como eventos de *click*, donde se pueden enlazar comandos.



## 3.3.5 – MVVM: EJEMPLO PRÁCTICO

- Definimos la estructura del proyecto:
  - Modelo (Models): *Mensaje.cs*
  - Vista (Views): *MainWindow.xaml + MainWindow.xaml.cs* y *MensajeWindow.xaml + MensajeWindow.xaml.cs*
  - ViewModel (ViewModels): *MainViewModel.cs*



## 3.3.5 – MVVM: EJEMPLO PRÁCTICO

- Definimos el [StartupURI](#) del fichero [MainWindow.xaml](#) dentro de [App.xaml](#)

App.xaml

```
<?xml version="1.0" encoding="utf-8" />
<Application x:Class="MensajeApp.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:MensajeApp"
    StartupUri="Views/MainWindow.xaml">
    <Application.Resources>
    </Application.Resources>
</Application>
```

- Definimos el [Model](#) en [Persona.cs](#)

```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6
7 namespace MensajeApp.Models
8 {
9     public class Mensaje
10     {
11         public string Contenido { get; set; }
12     }
13 }
14
15
```

## 3.3.5 – MVVM: EJEMPLO PRÁCTICO

■ Definimos el *ViewModel* en el fichero *MainViewModel.cs*

```
1 using MensajeApp.Models;
2 using System.ComponentModel;
3 using System.Runtime.CompilerServices;
4
5 namespace MensajeApp.ViewModels ←
6 {
7     3 referencias
8     public class MainViewModel : INotifyPropertyChanged
9     {
10         private Mensaje _mensaje;
11
12         3 referencias
13         public Mensaje Mensaje
14         {
15             get => _mensaje;
16             set
17             {
18                 _mensaje = value;
19                 OnPropertyChanged();
20             }
21         }
22
23         1 referencia
24         public MainViewModel()
25         {
26             Mensaje = new Mensaje();
27         }
28
29         public event PropertyChangedEventHandler PropertyChanged;
30
31         1 referencia
32         protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
33         {
34             PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
35         }
36     }
37 }
```



## 3.3.5 – MVVM: EJEMPLO PRÁCTICO

- Definimos la [View](#) en el fichero [MainWindow.xaml](#)

```
1 <Window x:Class="MensajeApp.Views.MainWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       Title="Ventana Principal" Height="200" Width="300">
5     <StackPanel>
6       <Button Content="Abrir Ventana de Mensaje" Click="AbrirMensaje_Click" />
7       <TextBlock x:Name="MensajeTextBlock" />
8     </StackPanel>
9 </Window>
10
```

- Definimos el [CodeBehind](#) de [MainWindow.xaml](#) en el fichero [MainWindow.xaml.cs](#)

```
1 using MensajeApp.ViewModels;
2 using System.Windows;
3
4 namespace MensajeApp.Views
5 {
6     2 referencias
7     public partial class MainWindow : Window
8     {
9         private MainViewModel _viewModel;
10
11         0 referencias
12         public MainWindow()
13         {
14             InitializeComponent();
15             _viewModel = new MainViewModel();
16             this.DataContext = _viewModel;
17         }
18
19         1 referencia
20         private void AbrirMensaje_Click(object sender, RoutedEventArgs e)
21         {
22             var mensajeWindow = new MensajeWindow(_viewModel.Mensaje);
23             mensajeWindow.ShowDialog();
24             MensajeTextBlock.Text = _viewModel.Mensaje.Contenido;
25         }
26     }
27 }
```

## 3.3.5 – MVVM: EJEMPLO PRÁCTICO

- Definimos la [View](#) en el fichero [MensajeWindow.xaml](#)

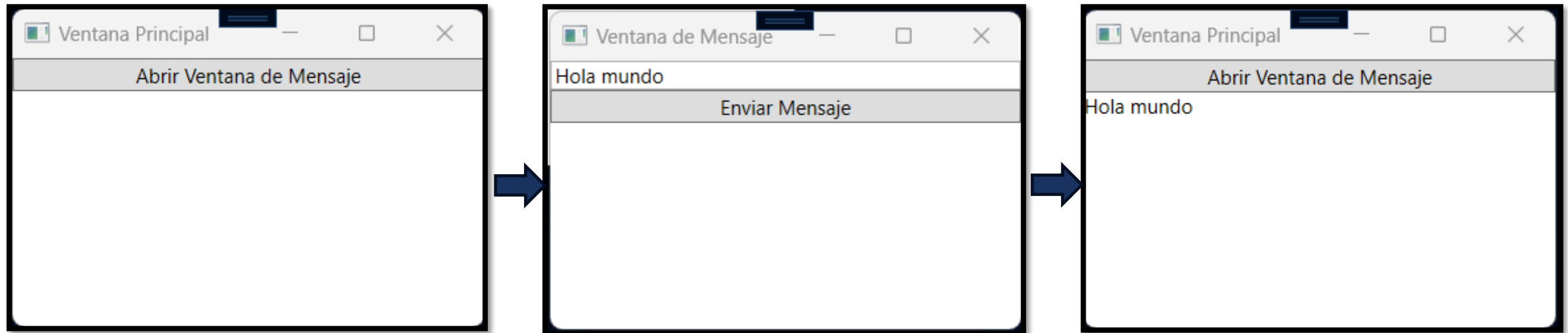
```
1 <Window x:Class="MensajeApp.Views.MensajeWindow"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       Title="Ventana de Mensaje" Height="200" Width="300">
5     <StackPanel>
6       <TextBox x:Name="MensajeTextBox" />
7       <Button Content="Enviar Mensaje" Click="EnviarMensaje_Click" />
8     </StackPanel>
9   </Window>
```

- Definimos el [CodeBehind](#) de [MensajeWindow.xaml](#) en el fichero [MensajeWindow.xaml.cs](#)

```
1 using MensajeApp.Models;
2 using System.Windows;
3
4 namespace MensajeApp.Views
5 {
6     public partial class MensajeWindow : Window
7     {
8         private Mensaje _mensaje;
9
10        public MensajeWindow(Mensaje mensaje)
11        {
12            InitializeComponent();
13            _mensaje = mensaje;
14        }
15
16        private void EnviarMensaje_Click(object sender, RoutedEventArgs e)
17        {
18            _mensaje.Contenido = MensajeTextBox.Text;
19            this.Close();
20        }
21    }
22 }
```

## 3.3.5 – MVVM: EJEMPLO PRÁCTICO

- Vemos la ejecución del caso práctico





GRACIAS POR  
VUESTRA  
ATENCIÓN