UD3 WPF y Componentes

Ciclo: Desarrollo de Aplicaciones Multiplataforma
Módulo: Desarrollo de interfaces.

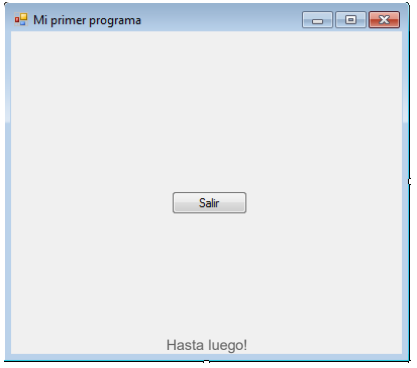
PROGRAMANDO EN WINDOWS

- Una aplicación para Windows diseñada para interaccionar con el usuario presentará una interfaz gráfica que mostrará todas las opciones que el usuario puede realizar.
- Dicha interfaz se basa fundamentalmente en dos tipos de objetos: ventanas (también llamadas formularios) y controles (botones, cajas de texto, menús, listas, etc.)

- Para realizar una aplicación que muestre una interfaz gráfica, se crean objetos que den lugar a ventanas y sobre esas ventanas se dibujan otros objetos llamados controles.
- Finalmente se escribe el código fuente relacionado con la función que tiene que realizar cada objeto de la interfaz.

- Una aplicación en Windows presenta todas las opciones posibles en uno o más formularios para que el usuario elija una de ellas.
- En la imagen, cuando el usuario haga clic sobre el botón “Salir”, en la caja de texto aparecerá el mensaje Hasta luego! Se dice en tonces que la programación es conducida por eventos y orientada a objetos.

```
<Button Content="Salir"
Name="btSalir"
Click="btSalir_Click" /
>
```



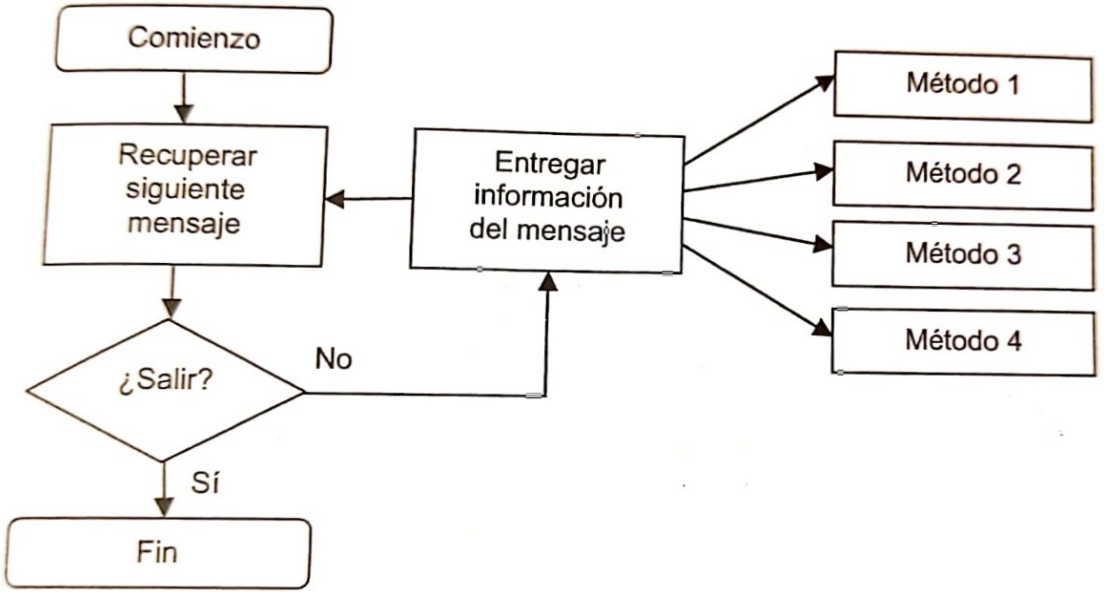

```
private void btSalir_Click(object sender,
RoutedEventArgs e)

etSaludo.Content = "Hasta luego!";
```

- El método btSalir_Click será puesto en ejecución en respuesta al evento Click del objeto identificado por btSalir (botón titulado “Salir").
- Esto significa que cuando el usuario haga clic en el objeto btSalir se ejecutará el método btSalir_Click.
- Esto es justamente lo que está indicando el código XAML Click="btSalir_Click" que se muestra un poco antes: el método btSalir_Click manejará el evento Click de btSalir.
- Esta forma de programar se denomina programación conducida por eventos y orientada a objetos.

Eventos

- Los eventos son mecanismos mediante los cuales los objetos (ventanas o controles) pueden notificar de la ocurrencia de sucesos.
- Un evento puede ser causado por una acción del usuario (por ejemplo, cuando pulsa una tecla), por el sistema (transcurrido un determinado tiempo) o indirectamente por el código (al cargar una ventana).

- Cómo actúa el bucle de mensajes mientras la aplicación está en ejecución:



Biblioteca WPF

- Para diseñar aplicaciones que utilicen interfaces gráficas (ventanas con controles, como etiquetas, cajas de texto, botones, barras de desplazamiento, etc.), .NET proporciona (además Windows Forms), otra biblioteca de clases denominada WPF.
- Proporciona la capacidad para programar una aplicación utilizando el lenguaje de marcado XAML para implementar su interfaz gráfica y los lenguajes de programación administrados, como C#, para escribir el código subyacente que implemente su comportamiento.
- Esta separación entre la apariencia y el comportamiento permite a los diseñadores implementar la apariencia de una aplicación al mismo tiempo que los programadores implementan su comportamiento.

XAML (eXtensible Application Markup Language: lenguaje de marcado extensible para aplicaciones o XML para Aplicaciones)

- Se utiliza para definir la estructura jerárquica de la interfaz de usuario, la cual se puede ver como un árbol de elementos (en nuestra aplicación, la raíz de este árbol es un objeto Window que tendrá como hijos los controles que añadamos).
- Árbol que normalmente guardaremos en un fichero en XAML, proceso que recibe el nombre de seriación; esto es, el fichero XAML es simplemente una seriación de objetos.

```
<Window x:Class="WPFAplicacion.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

Title="Ventana WPF" height="200" width="300" Name="VentanaWPF"> <Grid>

</Grid>

</Windows >
```

- Este código declara una ventana (elemento Window), denominada Ventana WPF, con unos atributos determinados (se corresponden con propiedades en la clase); algunos, como Title, Height, Width y Name, han sido definidos de forma explícita.
- El atributo xmlns (xml name space) define un espacio de nombres XML.

- En el ejemplo anterior, el atributo x:Class define la clase de código subyacente vinculada con este código XAML; lo cual quiere decir que este atributo le indica al procesador de XAML que deberá generar una clase MainWindow derivada del elemento raíz, Window.
- El atributo Title de Window no tiene prefijo de espacio de nombres, lo que indica, normalmente, que se corresponde con una propiedad del objeto .NET al que hace referencia el elemento XAML. Esto último sería equivalente al código siguiente:

  Ventana WPF.Title = "Ventana WPF" ;

- Si utilizamos XAML, cualquier herramienta de diseño capaz de procesar XML podrá deseriar el código XAML generado por Visual Studio, modificarlo y volverlo a seriar nuevamente, de modo que pueda volverse a cargar por Visual Studio.
- Así, mientras los programadores emplean Visual Studio para escribir el código subyacente, los diseñadores pueden utilizar otras herramientas.

Código subyacente

- Para responder a los eventos y manipular los objetos declarados en XAML, como etiquetas, cajas de texto, botones, se usa un fichero de código subyacente independiente.
- En nuestro caso, este fichero es MainWindow.xaml.cs:

```
namespace WpfAplicacion
{
   /// Lógica de interacción para MainWindow.xaml
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }
    }
}
```

```
public partial class MainWindow : System.Windows.Window, System.Windows.Markup. I Component Connector
{
     #line 4"..... \MainWindow.xaml"
     //...

     internal WPFAplicacion.MainWindow VentanaWPF;
     //…

     public void InitializeComponent()
     {
     //…
          System. Uri resourceLocater = new System. Uri(
             "/WPFAplicacion:component/mainwindow. Xaml", System, UriKind. Relative);
          //...
          #line 1."..\..\..\MainWindow.xaml"
          System.Windows. Application. LoadComponent(
             this, resourceLocater);
          //…
     }
     //…
}
```

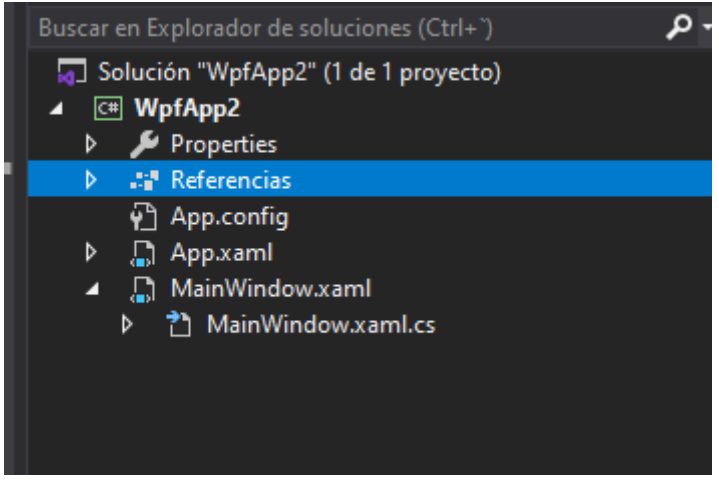
COMPILAR Y EJECUTAR LA APLICACIÓN

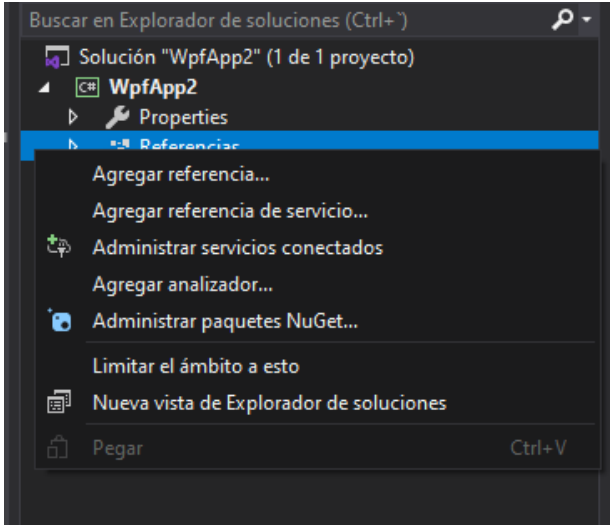
- Se puede ver en los ficheros de código subyacente de la aplicación las directrices using, que especifican los espacios de nombres a los que pertenecen las clases utilizadas por MainWindow y por App:

```
using System;

using System.Windows;
```

- La biblioteca de clases .NET está organizada en espacios de nombres que agrupan esas clases dispuestas según una estructura jerárquica.

- Estos espacios de nombres se corresponden con bibliotecas dinámicas del mismo nombre, a las que tendremos que hacer referencia cuando se compile una aplicación en la medida en la que sean necesarias.
- Esto se hace añadiendo dichas referencias al nodo References del proyecto si aún no están añadidas.
- Para ello, hay que pinchar con el botón secundario del ratón sobre este nodo y seleccionar Agregar referencia.





DISEÑO DE LA INTERFAZ GRÁFICA

- Nuestro siguiente paso consistirá en añadir a la ventana los elementos que tienen que formar parte de la interfaz gráfica.
- Este proceso requiere colocar en la misma los controles que nosotros creamos adecuados y añadir a la aplicación otras ventanas si fuera necesario mediante WPF

- XAML tiene un conjunto de reglas que asignan elementos (como Window) a clases o estructuras, atributos a propiedades o eventos, y espacios de nombres de XML a espacios de nombres del CLR.

```
<Grid>
<Button Content-"Hola Mundo" Name="btSaludo" />
</Grid>
```

Espacios de nombres XML

- El prefijo indica que el elemento existe en un espacio de nombres XAML diferente del espacio de nombres predeterminado

```
<Window x:Class="Conver Temps.MainWindow'
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  ...
  xmlns:local="clr-namespace: ConverTemps.Clases">
   <Window. Resources>
     <local:Conversor Double Brush x: Key="converGradosCaBrush
   </Window. Resources>
   <Grid>
   </Grid>
</Window>
```

- http://schemas.microsoft.com/winfx/2006/xaml/presentation es el espacio de nombres del núcleo de WPF. Abarca todas las clases de WPF, incluyendo las de los controles que utilizamos para construir las interfaces de usuario. Este espacio de nombres se declara sin prefijo, por lo que se convierte en el espacio de nombres predeterminado para todo el documento.
- http://schemas.microsoft.com/winfx/2006/xaml es el espacio de nombres de XAML. Incluye diversos elementos de XAML que permiten interpretar el documento de una u otra forma. Este espacio de nombres se asigna al prefijo x, lo cual significa que se puede hacer referencia al mismo colocando el prefijo antes del nombre del elemento (por ejemplo <x:id_elemento>).
- Los espacios de nombres permiten al analizador de XAML encontrar la clase correcta. Por ejemplo, cuando el analizador procesa los elementos Window y Grid busca en el espacio de nombres XAML predeterminado los correspondientes espacios de nombres .NET que definan las clases System.Windows.Window y System.Windows.Controls.Grid.

Propiedades como atributos

- En XAML se pueden utilizar atributos para especificar las propiedades de un objeto de una clase.

```
<Button Content="Hola mundo" Name="btSaludo" />
```

Propiedades como elementos

- Cuando el valor de la propiedad no se puede expresar como una cadena simple, en lugar de utilizar atributos, las propiedades se especifican como elementos hijo empleando la sintaxis: Elemento Padre.Nombre Propiedad.

```
(Button Content-"Hola Mundo" Name="btSaludo" >
  <Button. Background>
     (SolidColorBrush Color="Blue" />
  </Button.Background>
 </Button)
```

```
<Button Content="Hola Mundo" Name="btSaludo" >
  <Button.Background>
    <LinearGradientBrush StartPoint="0,0" EndPoint="0,1">
      <LinearGradientBrush.Gradient Stops >
        <GradientStop Offset="0" Color="#FF080000"/>
        <GradientStop Offset="0.60" Color="Blue"/>
        <GradientStop Offset="1" color="#FF050000" />
      </LinearGradientBrush.GradientStops)
    </LinearGradientBrush)
  </Button.Background)
</Button)
```

Propiedades de contenido

- Cualquier clase que se pueda utilizar como un elemento XAML puede designar a una de sus propiedades como propiedad de contenido.
- Esto, en la sintaxis de XAML, permite omitir la especificación explícita de esta propiedad, tomándose como valor para la misma el contenido de cualquier elemento secundario especificado.

```
<Button>

Hola Mundo

<Button.Name>btSaludo</Button.Name>

</Button)
```

Extensiones de marcado

- XAML a través de las extensiones de marcado ofrece una vía alternativa al control habitual de atributos o elementos del procesador de XAML de WPF, cediendo el procesamiento a una clase que decide durante la ejecución cómo establecer el valor de una propiedad.
- Las extensiones de marcado más comunes son las que admiten referencias de recursos (StaticResource y DynamicResource) y las que admiten el enlace de datos (Binding).

```
<Grid>
   <Grid.Resources>
      <SolidColorBrush x:Key="Color FondoBt" Color="Azure" />
   </Grid.Resources>
   <Button Content="Hola Mundo" Name="btSaludo"
       Background="{DynamicResource ColorFondoBt}" />
</Grid>
```

El valor de la propiedad Background del objeto Button se obtiene a partir del recurso ColorFondoBt.

- Las extensiones de marcado en WPF están diseñadas principalmente para proporcionar un medio de hacer referencia a otros objetos ya existentes, o referencias diferidas a los objetos que se evaluarán durante la ejecución.

```
<Button Content= "Hola Mundo" Name="btSaludo"
Background="{Binding Background, ElementName=btAceptar}" />
```

- Las extensiones de marcado permiten utilizar una sintaxis de propiedades como atributo" que, de no existir, no sería posible aplicar.
- Por ejemplo, un objeto Style es un tipo de referencia relativamente complejo que contiene varias propiedades, que se crea normalmente como un recurso y, a continuación, se hace referencia a él mediante una extensión de marcado que solicita el recurso….

```
<Grid>
      <Grid.Resources>
          <Style x:Key="BtFondoAzulClaro">
              <Setter Property="Control.Background" Value="LightBlue"/>
          </Style>
      </Grid.Resources>
      <Button Content="Hola Mundo" Name="btSaludo"
          Style="{StaticResource BtFondoAzulClaro}"/>
  </Grid>
```

Propiedades asociadas

- Se denominan así a las propiedades propias de un tipo que pueden ser establecidas, además de por un elemento del propio tipo, por cualquier otro elemento.
- La sintaxis es: TipoPrimario.NombrePropiedad.
- El propósito es permitir que los elementos secundarios informen a un elemento primario de los valores de una propiedad, aunque el elemento primario y los secundarios no posean esa propiedad como parte de las listas de miembros de su clase.

Un elemento secundario (Button) informa al primario (Grid) de cómo se va a presentar en la interfaz de usuario (en la fila 1 del grid)

```
<Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <Grid.RowDefinitions>
            <RowDefinition/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Button Grid.Row="1"
            Content="Haga clic aquí" Name="btSaludo" Height="23" Width="120"/>
    </Grid>
```

Propiedades de dependencia

- WPF proporciona un conjunto de servicios que se pueden utilizar para extender la funcionalidad de una propiedad CLR (una propiedad .NET con sus descriptores de acceso get y set respaldada por un campo).
- Este conjunto de servicios se conoce como sistema de propiedades de WPF. Una propiedad que esté respaldada por la clase DependencyProperty y registrada en este sistema se conoce como propiedad de dependencia, implementación alternativa al modelo estándar de respaldar la propiedad con un campo privado.

- El propósito de estas propiedades es proporcionar una manera de calcular el valor de una propiedad en función del valor de otras entradas.
- De esta forma, una propiedad de dependencia puede heredar su valor de un elemento padre.

- Una propiedad de dependencia, como Background, puede hacer referencia mediante un enlace de datos, definido por un objeto Binding, a un valor que será obtenido durante la ejecución (en el ejemplo, de la propiedad Background del elemento btAceptar).

```
<Button Content="Hola Mundo" Name="btSaludo"
Background="{Binding Background, ElementName=btAceptar)"/>
```

Crear un elemento

- La implementación de un elemento (una ventana o un control) abarca tanto su aspecto como su comportamiento, donde el aspecto define la apariencia del elemento para los usuarios y el comportamiento define como funciona ese elemento cuando los usuarios interactúan con él.
- En WPF, puede implementar el aspecto y comportamiento de un elemento utilizando el marcado XAML.
- Sin embargo, en general el aspecto de un elemento se implementa utilizando el marcado XAML y su comportamiento se implementa utilizando código subyacente, por ejemplo C#.

Controles más comunes

- Hay controles para cada uno de los elementos que ya hemos visto, más de una vez, en alguna ventana de la interfaz gráfica del sistema Windows, Linux u otro.
- A continuación vamos a ver de forma resumida una lista de los más comunes….

- Etiquetas. Se implementan a partir de la clase Label.

```
 <Grid>
<Label Content="This is a Label control." />
</Grid>
```

- Botones. Se implementan a partir de la clase Button.

```
<Button Background="Beige" Foreground="Blue" FontWeight="Bold" Click="Holaboton_Click"> Hola Botón</Button>
```

- Cajas de texto. Se implementan a partir de la clase TextBox las de una sola línea de texto, las de varias líneas y las de “palabra de paso”.

```
<TextBox Text="Hola mundo!" />
```

- Bloques de texto. Se implementan a partir de la clase TextBlock. Se trata de un control ligero, preparado específicamente para integrar pequeños fragmentos de contenido dinámico.

```
<Grid Margin="10">
<TextBox AcceptsReturn="True" TextWrapping="Wrap" />
</Grid>
```

- Bordes. Se implementan a partir de la clase Border. Dibuja alrededor de otro elemento un borde, un fondo o ambos.

```
<Border Background="GhostWhite" BorderBrush="Silver" BorderThickness="1" CornerRadius="8,8,3,3">
<StackPanel Margin="10">
<Button>Botón</Button>
</StackPanel>
</Border>
```

- Casillas de verificación. Se implementan a partir de la clase CheckBox.

```
<StackPanel Margin="10">
<Label FontWeight="Bold">Disponibilidad</Label>
<CheckBox>Mañanas</CheckBox>
<CheckBox>Tardes</CheckBox>
<CheckBox IsChecked="True">Fines de semana</CheckBox>
</StackPanel>
```

- Botones de opción. Se implementan a partir de la clase RadioButton.

```
<StackPanel Margin="10">
<Label FontWeight="Bold">Tipo de acceso</Label>
<RadioButton IsChecked="True">Ciclo Medio</RadioButton>
<RadioButton>Bachiller</RadioButton>
<RadioButton>Otros</RadioButton>
</StackPanel>
```

- Listas. Se implementan a partir de las clases ListBox y ComboBox.

```
<ListBox>
<ListBoxItem>Alicante</ListBoxItem>
<ListBoxItem>Castellón</ListBoxItem>
<ListBoxItem>Valencia</ListBoxItem>
</ListBox>
```

```
<ComboBox>
   <ComboBoxItem>Alicante</ComboBoxItem>
   <ComboBoxItem IsSelected="True">Castellón</ComboBoxItem>
   <ComboBoxItem>Valencia</ComboBoxItem>
</ComboBox>
```

- Barras de desplazamiento. Se implementan a partir de la clase ScrollBar.

```
<ScrollBar Name="McScroller" Orientation="Horizontal"
   Width ="250" Height="30"
   Margin="10,10,0,0"
Background=”LightSalmon”/>
```

- Estos controles se localizan en el espacio de nombres System.Windows.Controls y son objetos de subclases de la clase Control que a su vez se de riva de la clase System.Windows.FrameworkElement.
- La funcionalidad común de los controles disponibles en WPF está implementada en su mayoría en estas cuatro clases: UIElement, ContentElement, FrameworkElement y FrameworkContentElement, todas ellas derivadas indirectamente de DependencyObject

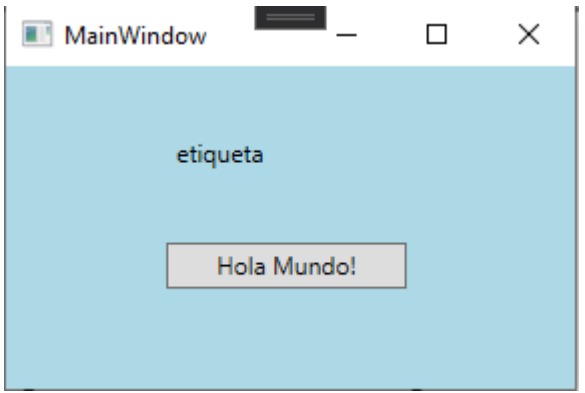
Paneles de diseño

- Un requisito fundamental de cualquier diseño es que la posición y el tamaño de los controles (elementos secundarios) puedan adaptarse a los cambios de tamaño del contenedor (elemento primario) y a la configuración de pantalla.
- Para ello, WPF proporciona un sistema de diseño que es invocado automáticamente cada vez que es necesario reorganizar los controles del contenedor: los controles indican a su contenedor qué ubicación y tamaño necesitan y el contenedor responde indicando a los controles de qué espacio dispone.

Canvas

- Es un contenedor que define un área en la que pueden colocarse explícitamente los elementos secundarios utilizando coordenadas relativas al área definida por este control.

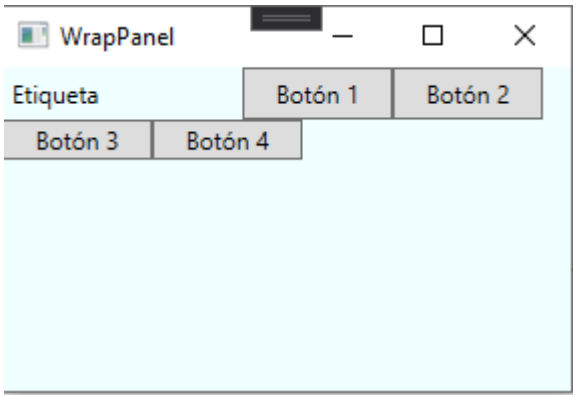
```
<Canvas Background="LightBlue">
        <Label Canvas.Left="80" Canvas.Top="30"
            Content="etiqueta" Name="labell" Width="120" Height="30"/>
        <Button Canvas.Left="80" Canvas.Bottom="50"
            Content="Hola Mundo!" Name="button" Height="23" Width="120"/>
</Canvas>
```


MainWindow
etiqueta
Hola Mundo!

WrapPanel

- Con este contenedor los elementos secundarios se sitúan por orden de izquierda a derecha y se ajustan a la línea siguiente cuando hay más controles de los que caben en la línea actual.

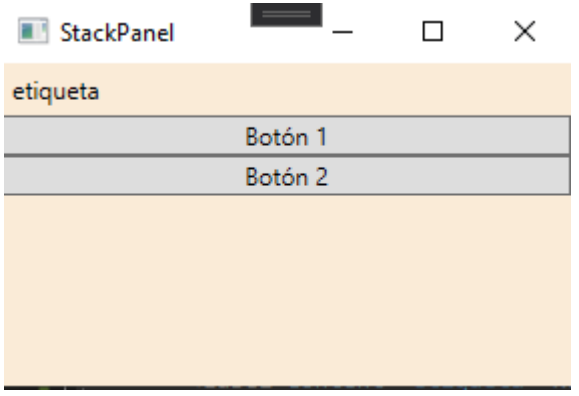
```
<WrapPanel Background="Azure">
       <Label Content="Etiqueta" Name="labell"  Width="120"/>
       <Button Content="Botón 1" Name="button1" Width="75"/>
       <Button Content="Botón 2" Name="button2" Width="75"/>
       <Button Content="Botón 3" Name="button3" Width="75"/>
       <Button Content="Botón 4" Name="button4" Width="75"/>
   </WrapPanel>
```



StackPanel

- Con este contenedor los elementos secundarios se apilan vertical u horizontalmente.

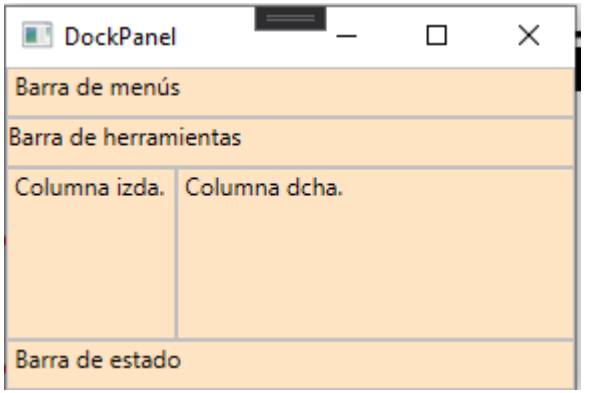
```
<StackPanel Orientation="Vertical" Background="AntiqueWhite">
      <Label Content="etiqueta" Name="labell"/>
      <Button Content="Botón 1" Name="buttonl"/>
      <Button Content="Botón 2" Name="button2"/>
</StackPanel>
```



DockPanel

- Con este contenedor los elementos secundarios se alinean con los bordes del mismo. Por defecto se alinean a la izquierda.

```
<DockPanel Background="Bisque">
      <Border DockPanel.Dock="Top"
          BorderBrush="Silver" BorderThickness="1" Height="25">
        <TextBlock Name="textBlockl" Text=" Barra de menús"/>
      </Border>
      <Border DockPanel.Dock="Top"
          BorderBrush="Silver" BorderThickness="1" Height="25">
        <TextBlock Name="textBlock2" Text="Barra de herramientas"/>
      </Border>
      <Border DockPanel.Dock="Bottom"
          BorderBrush="Silver" BorderThickness="1" Height="25">
          <TextBlock Name="textBlock3" Text=" Barra de estado"/>
      </Border>
      <Border DockPanel.Dock="left"
          BorderBrush="Silver" BorderThickness="1" Width="85">
          <TextBlock Name="textBlock4" Text=" Columna izda."/>
      </Border>
      <Border BorderBrush="Silver" BorderThickness="1">
          <TextBlock Name="textBlock5" Text=" Columna dcha."/>
      </Border>
  </DockPanel>
```



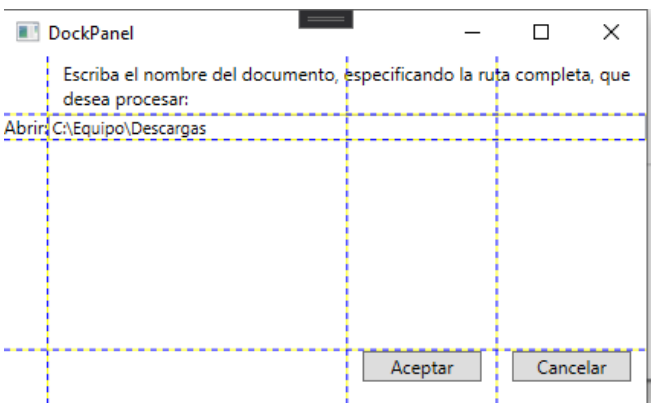
Grid

- Este contenedor define un área de cuadrícula flexible, compuesta por filas y columnas, para situar los elementos.
- Puede ser utilizado para lograr casi todo lo que se puede hacer con los otros contenedores, aunque, en algunos casos, no con la misma facilidad.
- Usando XAML, Grid permite definir filas (rows) y columnas (columns) en un panel y luego colocar los elementos secundarios haciendo referencia a los atributos Row y Column del Grid.
- Los controles pueden abarcar varias filas (RowSpan). columnas (ColumnSpan).

- Un Grid puede definir el alto de sus filas y el ancho de sus columnas de forma automática en función del tamaño del contenido (por ejemplo, Height="Auto" o Width="Auto').
- También puede especificar un tamaño proporcional al espacio restante disponible empleando, en XAML, el *, 2*,..., o bien especificar un valor absoluto en unidades independientes de dispositivo (1/96 de pulgada por unidad pixel: px) o no especificarlo, en cuyo caso se supone el valor *.

- La propiedad ColumnDefinitions del Grid define una colección de elementos ColumnDefinition, cada uno de los cuales define las propiedades específicas (por ejemplo Width) de la columna que representa y la propiedad RowDefinitions define una colección de elementos RowDefinition, cada uno de los cuales define las propiedades específicas (por ejemplo Height) de la fila que representa.

```
<Grid ShowGridLines="True">

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto"/>
        <ColumnDefinition Width="2*"/>
        <ColumnDefinition Width="*"/>
        <ColumnDefinition Width="*"/>
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="Auto"/>
        <RowDefinition Height="*"/>
        <RowDefinition Height="Auto"/>
    </Grid.RowDefinitions>
    <Image Grid.Row="0" Grid.Column="0" Source="libro.ico"/>
    <TextBlock Padding="10,3,10,3" Grid.Row="0" Grid.Column="1"
            Grid.ColumnSpan="3" TextWrapping="Wrap">
       Escriba el nombre del documento, especificando la ruta completa, que desea procesar:
    </TextBlock>
    <TextBlock Grid.Row="1" Grid.Column="0">Abrir:</TextBlock>
    <TextBox Grid.Row="1" Grid.Column="1" Grid.ColumnSpan="3"/>
    <Button Margin="10, 1, 10, 15"
            Grid.Row="3" Grid.Column="2">Aceptar</Button>
         <Button Margin="10, 1, 10, 15"
            Grid.Row="3" Grid.Column="3">Cancelar</Button>

</Grid>
```



- Combinando esta técnica con otros controles como ScrollViewer, GridSplitter. Viewbox, etc., puedes dar una gran flexibilidad a tus diseños
- Todos los paneles de diseño expuestos, excepto Canvas, ajustan la posición y el tamaño de los controles (elementos secundarios) cuando se cambia el tamaño del contenedor (elemento primario) o la configuración de pantalla.