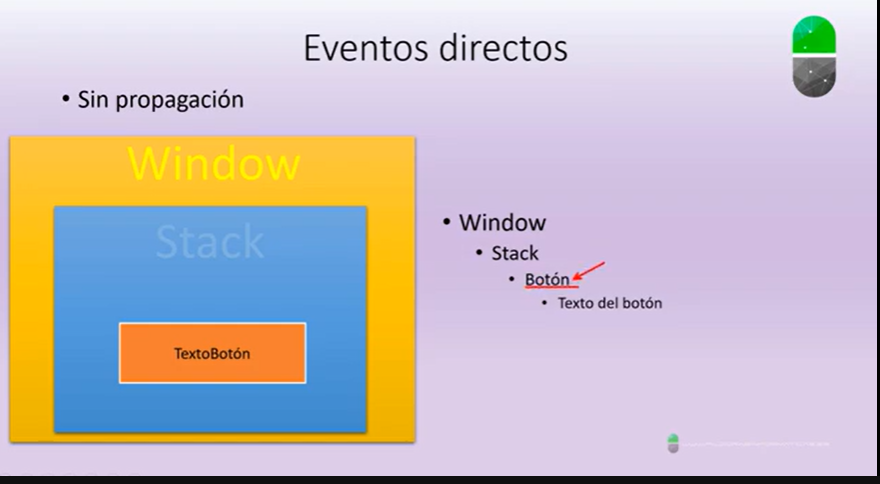
**ELEMENTOS - WPF**

**Eventos en Wpf:**

**Eventos Enrutados:**

Directos

Burbujas(Bubling)



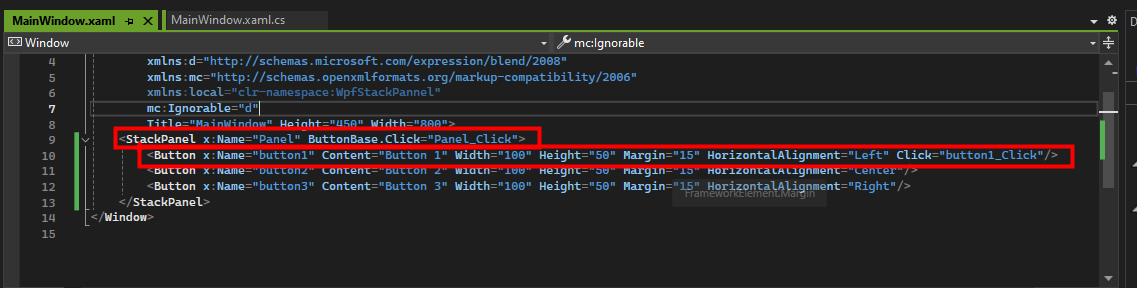
Tunelados(Tunneling)

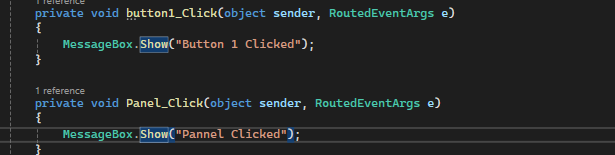


Entonces en nuestro código primero haremos el evento de tipo bubling:

De adentro hacia afuera -> primero al botono luego al panel:

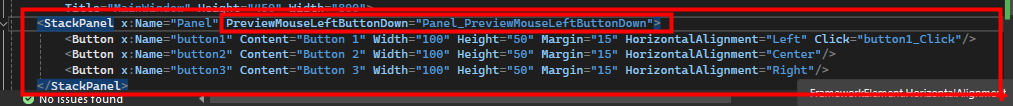
Creamos un evento de tipo bubling -> **ButonBase.click:**

****

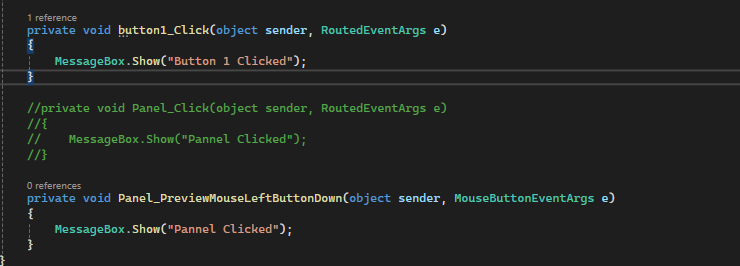
Y creamos estos eventos: 

Al darle click el botón 1: Debería primero salir Button1 clicked, y luego Pannel clicked

Ahora crearemos un evento de tipo **Tunneling**, que como vimos llevan por delante la palabra **preview**:

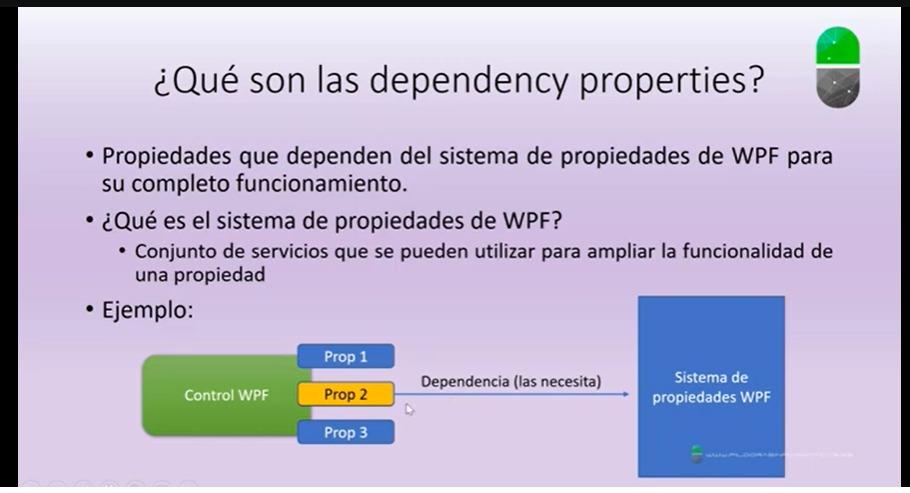
Entonces eliminamos el evento anterior de **ButonBase.Click** y ponemos ahora este evento: **PreviewMouseLeftButtonDown**

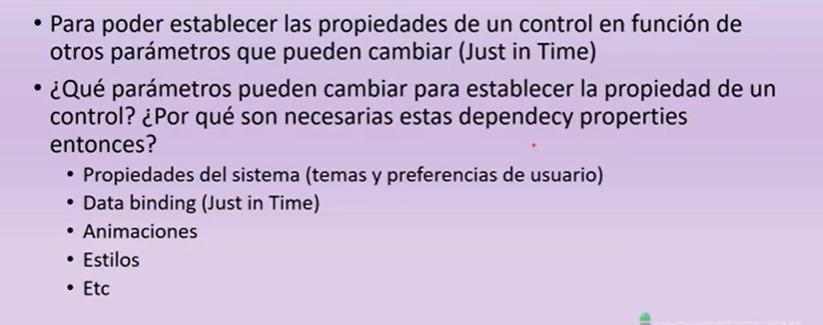
E implementamos:



Como vemos, no esta saliendo el mensaje de butoon 1 clicked, y es porque cuando se usa MessageBox, ya no desencadena el siguiente evento.

**DEPENDENCY PROPERTIES**

****

****

**Vamos a implementar un evento personalizado para mi control button:**



Además, se pueden usar **triggers** dentro de un Style para cambiar el valor de una propiedad en respuesta a un evento o condición específica, como cuando el usuario interactúa con el control (por ejemplo, pasando el mouse por encima).

En este caso, el Trigger se activa cuando se cumple una condición sobre la propiedad IsMouseOver:

 **Property="IsMouseOver"**: La propiedad que se está observando. IsMouseOver es una propiedad booleana de tipo Button, que es true cuando el mouse pasa por encima del botón y false cuando no lo está.

 **Value="True"**: El valor que se debe cumplir para que el Trigger se active. En este caso, el trigger se activa cuando el valor de IsMouseOver es true, es decir, cuando el mouse está sobre el botón.

 **Setter**: El Setter define qué propiedad cambiar cuando el trigger se activa. En este caso, cuando el mouse está sobre el botón (IsMouseOver = True), el Setter cambia la propiedad FontSize del botón a 25.

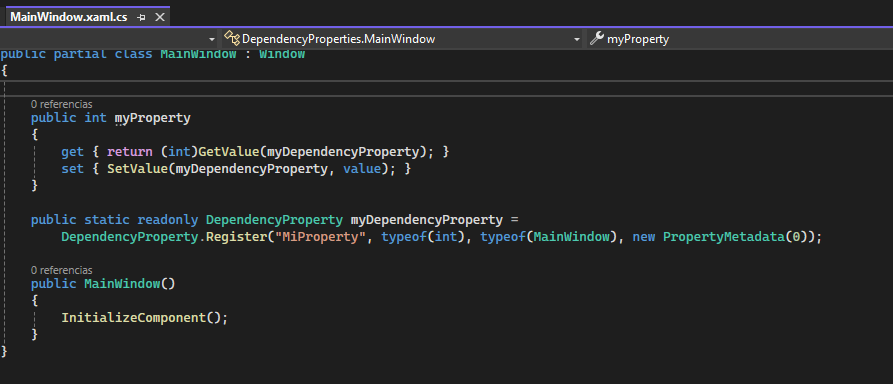
* **Property="FontSize"**: Se refiere a la propiedad FontSize del botón, que determina el tamaño de la fuente del texto dentro del botón.
* **Value="25"**: Es el nuevo valor que se le asigna a FontSize cuando el Trigger se activa. En este caso, el tamaño de la fuente se cambia a 25.

**Resumen:**

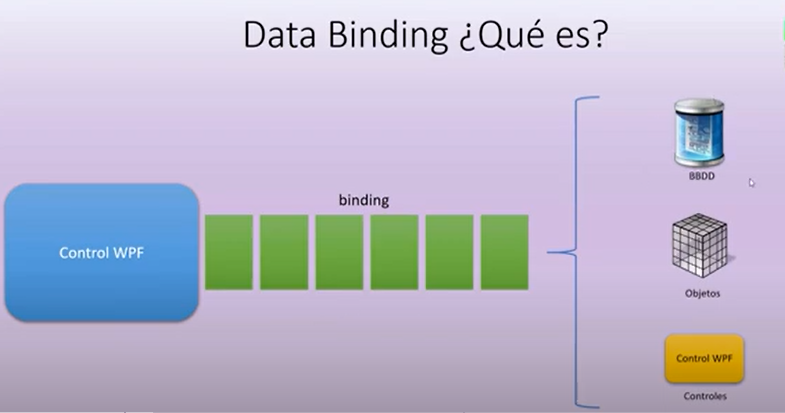
* **Style**: Define un conjunto de propiedades para un control (en este caso, un Button).
* **Trigger**: Permite cambiar propiedades de un control en respuesta a un cambio en otra propiedad (en este caso, cuando el mouse pasa sobre el botón).
* **Setter**: Cambia una propiedad del control cuando se activa el Trigger (en este caso, cambia el tamaño de la fuente del botón).

**Ahora vamos a crear una dependencyProperty popia:**

**Implementar una dependencyProperty**



**DATA BINDING**



El Data Binding en WPF (Windows Presentation Foundation) es una técnica que permite conectar propiedades de objetos en el código subyacente con propiedades de elementos de la interfaz de usuario, facilitando la sincronización automática de datos entre la lógica de la aplicación y su representación visual. Esta funcionalidad es esencial para desarrollar aplicaciones de manera eficiente, ya que permite separar claramente la lógica de negocio y la presentación.

### Características Clave del Data Binding

1. **Vinculación Unidireccional (One-Way Binding)**:
   * Los cambios en la fuente de datos se reflejan automáticamente en la propiedad de destino.
   * Ejemplo: Un TextBlock que muestra el nombre de usuario.
2. **Vinculación Bidireccional (Two-Way Binding)**:
   * Los cambios en la fuente de datos se reflejan en la propiedad de destino y viceversa.
   * Ejemplo: Un TextBox que permite editar el nombre de usuario y refleja los cambios en el modelo.
3. **Vinculación Unidireccional hacia la Fuente (One-Way to Source)**:
   * Los cambios en la propiedad de destino se reflejan en la fuente de datos, pero no al revés.
   * Ejemplo: Un TextBox que actualiza una propiedad en el modelo, pero no refleja cambios del modelo.
4. **Vinculación Unidireccional de un Solo Disparo (One-Time Binding)**:
   * La propiedad de destino se establece una sola vez cuando se crea la vinculación.
   * Ejemplo: Configurar un valor inicial en un TextBlock que no cambia.

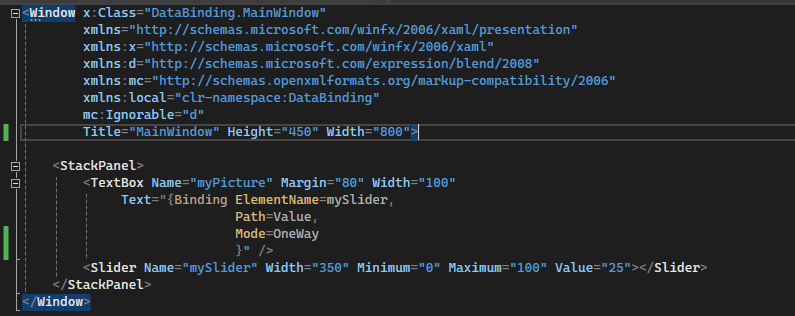
### Implementación del Data Binding

El Data Binding se define usualmente en XAML (eXtensible Application Markup Language) y se puede configurar de varias maneras.

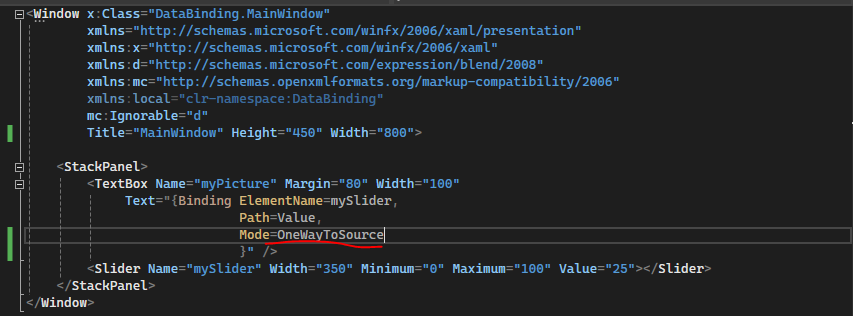
**Ahora vamos a implementar un dataBinding y creamos un nuevo proyecto**

utilizaremos un slider y un textBox

**utilizaremos oneWay** -> ya que enviaremos del source al target, es decir del slider al textBox



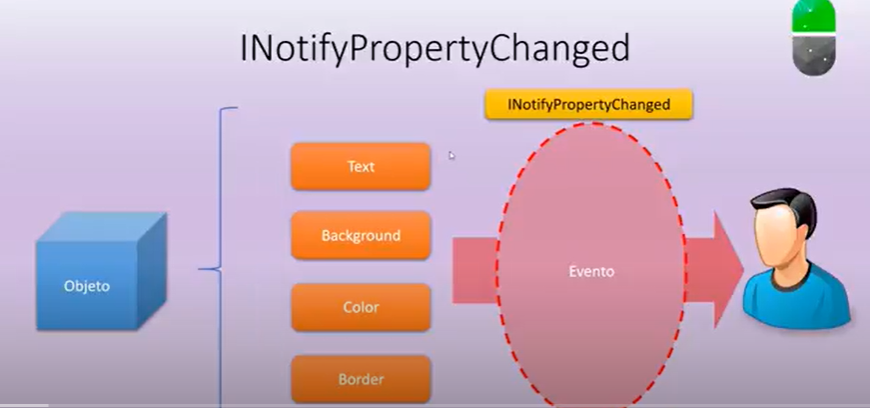
ahora que si usamos oneWayToSource, sera al contrario del target al source



Luego probamos con **two way**

**No olvidar presionar tab para que funcione**

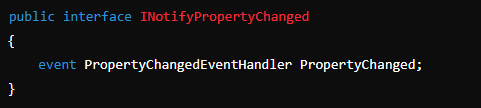
**INotifyPropertyChanged**



INotifyPropertyChanged es una interfaz en .NET que permite a las clases notificar a los clientes, típicamente enlaces de datos (bindings), que el valor de una propiedad ha cambiado. Esta interfaz es crucial para la implementación de data binding en aplicaciones WPF, ya que facilita la actualización automática de la interfaz de usuario cuando los datos subyacentes cambian.

### Cómo Funciona INotifyPropertyChanged

1. **Definición de la Interfaz**: INotifyPropertyChanged define un solo evento llamado PropertyChanged.

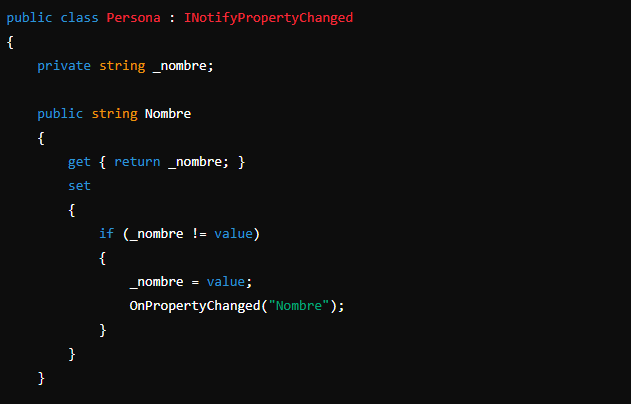


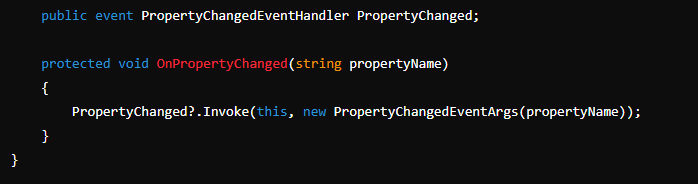
**2. Implementación de la Interfaz**: Para que una clase soporte notificaciones de cambios en propiedades, debe implementar la interfaz INotifyPropertyChanged y definir el evento PropertyChanged.

**3. Disparar el Evento PropertyChanged**: Cuando el valor de una propiedad cambia, la clase debe disparar el evento PropertyChanged, pasando el nombre de la propiedad que cambió.

### Ejemplo de Implementación

Supongamos que tienes una clase Persona que implementa INotifyPropertyChanged para su propiedad Nombre.





### Explicación del Ejemplo

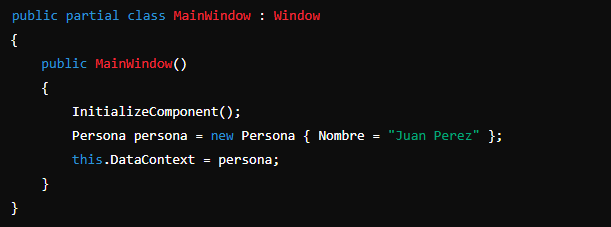
1. **Campo Privado**: La propiedad Nombre tiene un campo privado \_nombre que almacena su valor.
2. **Propiedad Pública**: La propiedad Nombre tiene un accesor get para devolver el valor y un accesor set para actualizar el valor.
3. **Comparación de Valores**: En el accesor set, se compara el nuevo valor con el valor actual. Si son diferentes, se actualiza el campo privado y se llama a OnPropertyChanged.
4. **Evento PropertyChanged**: El evento PropertyChanged se define de la interfaz INotifyPropertyChanged.
5. **Método OnPropertyChanged**: Este método dispara el evento PropertyChanged, pasando el nombre de la propiedad que cambió.

### Uso del Data Binding con INotifyPropertyChanged

Ahora, puedes vincular una instancia de Persona a la interfaz de usuario en WPF. Aquí hay un ejemplo de cómo hacerlo en XAML:



Y en el code-behind:

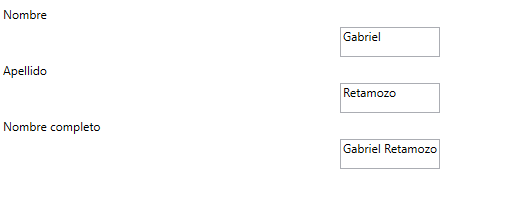


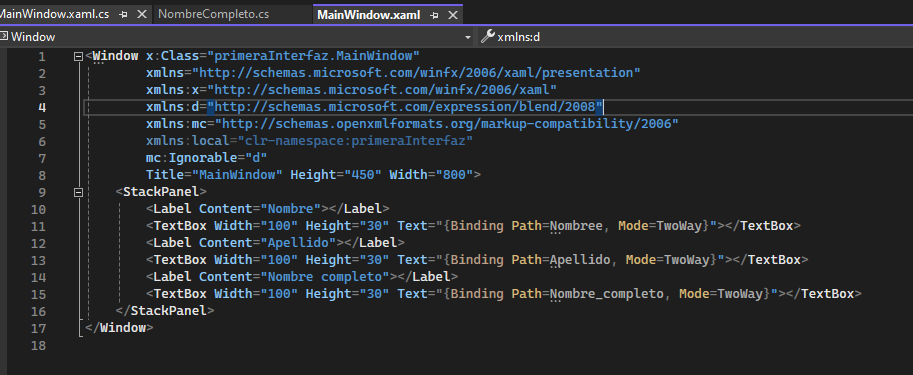
### Explicación del Uso en WPF

1. **DataContext**: El DataContext de la ventana principal se establece en una instancia de Persona.
2. **Binding**: El TextBox en XAML está vinculado a la propiedad Nombre de la instancia Persona.
3. **TwoWay Binding**: El modo de binding es TwoWay, lo que significa que los cambios en la propiedad Nombre se reflejarán automáticamente en el TextBox y viceversa.

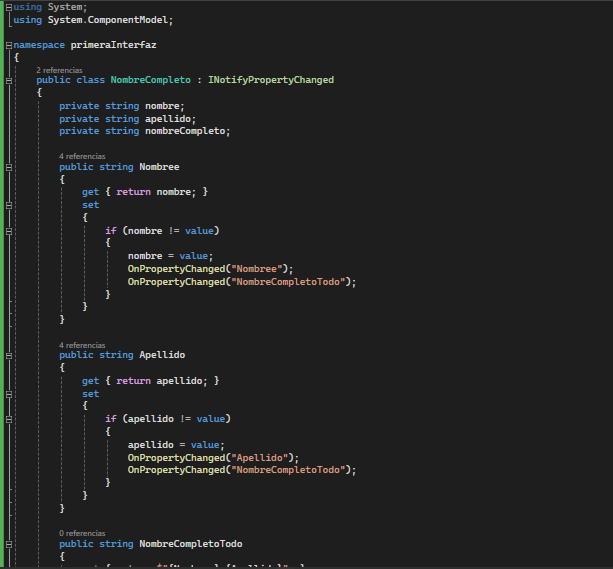
Implementar INotifyPropertyChanged es una práctica común y esencial para aprovechar al máximo las capacidades de data binding en WPF y otras tecnologías de .NET que soportan bindings de datos.

Implementamos el ejercicio:



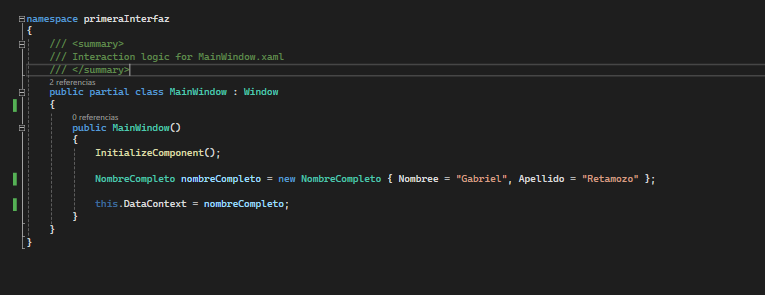


Creamos la clase NombreCompleto



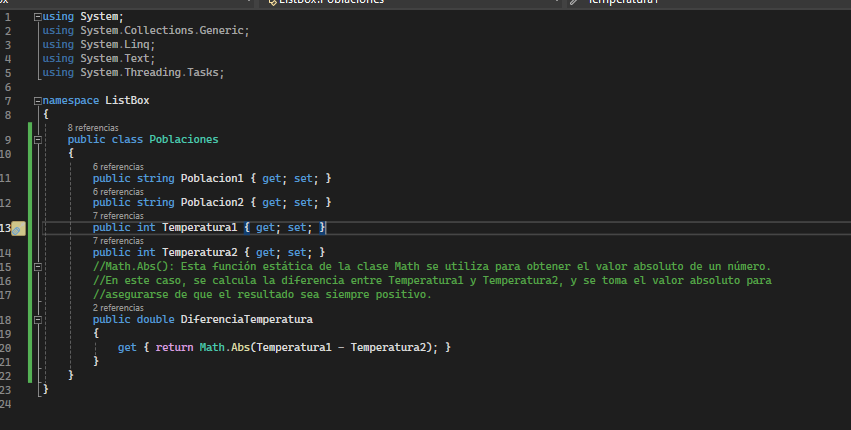


y por último colocamos esto en nuestra clase Main

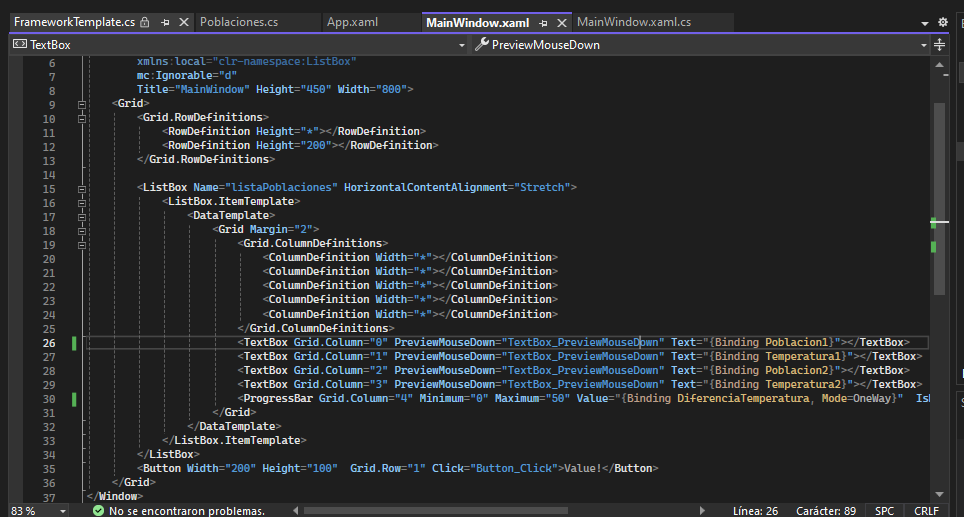


**Ejercicio LISTBOX con DataBinding**

1. Creamos la clase poblaciones



1. Creamos la estructur del grid



1. Creamos el codigo:

