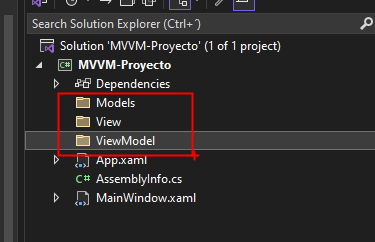
**PROYECTO MVVM - QUICK**

**Enlace: ->** <https://www.youtube.com/watch?v=CABv5xIDC08>

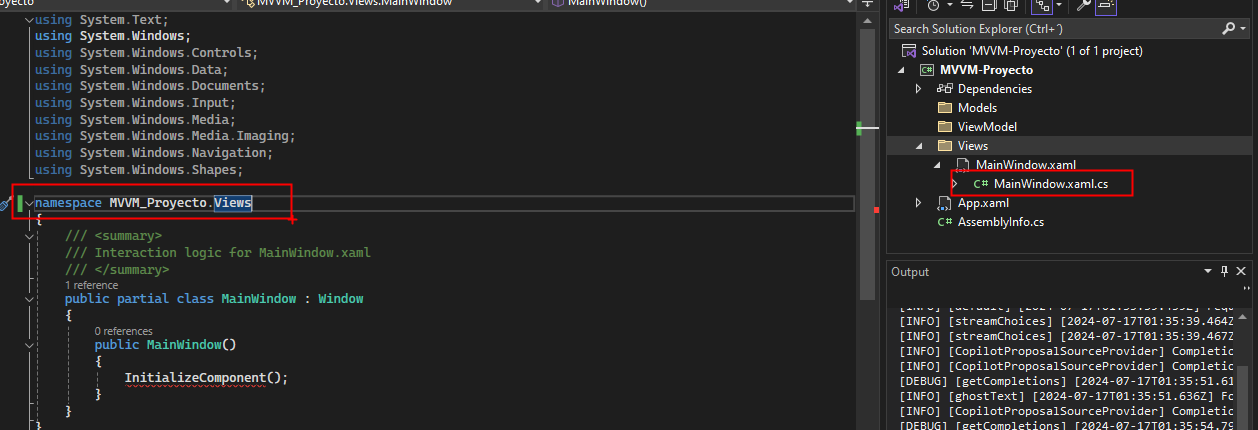
Creamos un Nuevo proyecto en WPF

Creamos las carpetas: **View**, **Model** y **ViewModel**

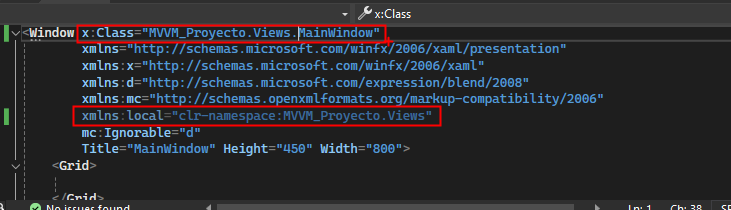


Y pasamos nuestro archivo **MainWindow.xaml** que representa mi vista a la carpeta **View**

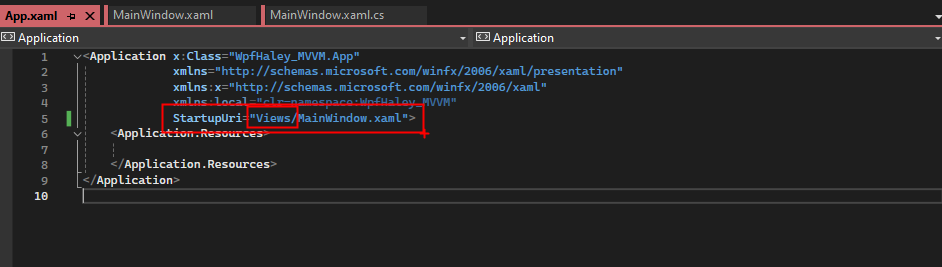
Y reajustamos el **nameSpace** de nuestro archivo **MainWindow.xaml.cs**



Luego tengo que modificar mi diseñador **Xaml:**

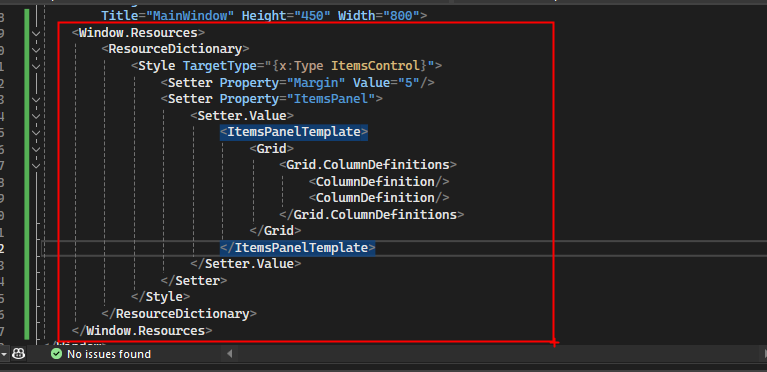


Y por ultimo en nuestro **App.xml** agregamos:



Ahora vamos a crear la interfaz – vista, colocando lo siguiente:

Eliminamos el **Grid** y ponemos un **Window.Resource:**

****

**Descripción Detallada**

**<Window.Resources>**

* **Descripción:** Define un bloque de recursos que están disponibles para la ventana actual.
* **Propósito:** Proporcionar un lugar donde definir recursos reutilizables como estilos, plantillas, y otros recursos que se pueden usar en cualquier lugar dentro del alcance de la ventana.

**<ResourceDictionary>**

* **Descripción:** Un contenedor para los recursos.
* **Propósito:** Organizar y gestionar los recursos de manera centralizada.

**<Style TargetType="{x:Type ItemsControl}">**

* **Descripción:** Define un estilo que se aplicará a todos los controles del tipo ItemsControl.
* **Propósito:** Aplicar propiedades y comportamientos a todos los ItemsControl en la ventana sin necesidad de configurar cada uno individualmente.

**<Setter Property="Margin" Value="5"/>**

* **Descripción:** Un Setter dentro del estilo que establece la propiedad Margin de todos los ItemsControl a 5.
* **Propósito:** Asegurarse de que todos los ItemsControl tengan un margen de 5 unidades alrededor.

**<Setter Property="ItemsPanel">**

 **Descripción:** Este Setter establece la propiedad ItemsPanel de los ItemsControl.

 **Propósito:** La propiedad ItemsPanel de un ItemsControl define el panel que se utiliza para organizar los ítems dentro del ItemsControl. En otras palabras, determina cómo se van a presentar los elementos contenidos en el ItemsControl.

* **ItemsPanel:** Es una propiedad de ItemsControl que define el panel que debe usarse para organizar los ítems.

**<Setter.Value>**

* **Descripción:** Contiene el valor que se va a asignar a la propiedad especificada en el Setter.
* **Propósito:** En este caso, establece la plantilla del panel de ítems.

**<ItemsPanelTemplate>**

* **Descripción:** Define la plantilla para el panel de ítems en un ItemsControl.
* **Propósito:** Personalizar el panel que organiza los elementos dentro del ItemsControl.

Este bloque de código XAML define un estilo que se aplicará a todos los ItemsControl en la ventana. Específicamente, establece un margen de 5 unidades alrededor de cada ItemsControl y define que el panel que organiza los ítems dentro del ItemsControl debe ser un Grid con dos columnas. Esto permite un diseño consistente y reutilizable para todos los ItemsControl en la ventana.

En resumen, estas líneas de código están configurando un estilo para todos los ItemsControl dentro del alcance del estilo, estableciendo un margen uniforme de 5 unidades y especificando cómo deben organizarse los ítems dentro del ItemsControl mediante la definición de un ItemsPanel.

**Propósito en el Contexto Global**

El código es parte de un recurso de estilo que se aplica automáticamente a todos los controles del tipo ItemsControl dentro del ámbito de la ventana. Esto significa que cualquier ItemsControl en la ventana tendrá automáticamente:

1. Un margen de 5 unidades alrededor.
2. Un ItemsPanel específico definido más adelante en el código.

Estos ajustes permiten una configuración uniforme y reutilizable de los ItemsControl, lo que facilita el mantenimiento y la consistencia del diseño de la interfaz de usuario.

Seguidamente debajo del **Window.Resource,** agregamos:

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<StackPanel Margin="10">

<ItemsControl>

<TextBlock Text="First Name"/>

<TextBox Grid.Column="1"/>

</ItemsControl>

<ItemsControl>

<TextBlock Text="Last Name"/>

<TextBox Grid.Column="1"/>

</ItemsControl>

<ItemsControl>

<TextBlock Text="Age"/>

<TextBox Grid.Column="1"/>

</ItemsControl>

<StackPanel Orientation="Horizontal" HorizontalAlignment="Right" Margin="5,20">

<Button Height="30" Width="120" Content="Add" Margin="5" HorizontalAlignment="Right"/>

<Button Height="30" Width="120" Content="Delete" Margin="5" HorizontalAlignment="Right"/>

</StackPanel>

</StackPanel>

<ListView x:Name="lstView" Grid.Row="1" Background="#FFEDE9C5">

<ListView.ItemTemplate>

<DataTemplate>

<WrapPanel>

<TextBlock Text="FirstName"/>

<TextBlock Text=" - " Foreground="Blue"/>

<TextBlock Text="LastName"/>

<TextBlock Text=" - " Foreground="Blue"/>

<TextBlock Text="Age"/>

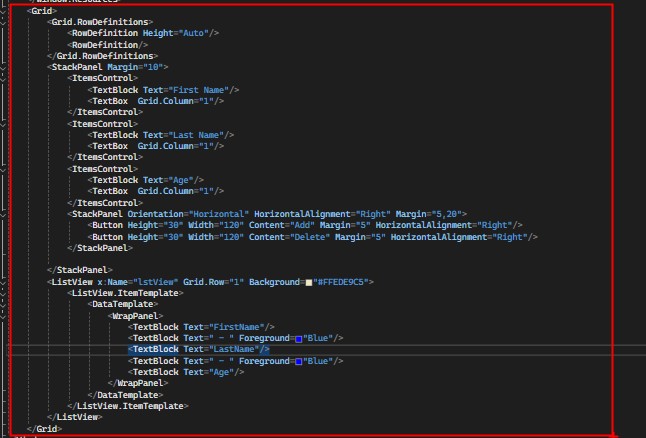
</WrapPanel>

</DataTemplate>

</ListView.ItemTemplate>

</ListView>

</Grid>

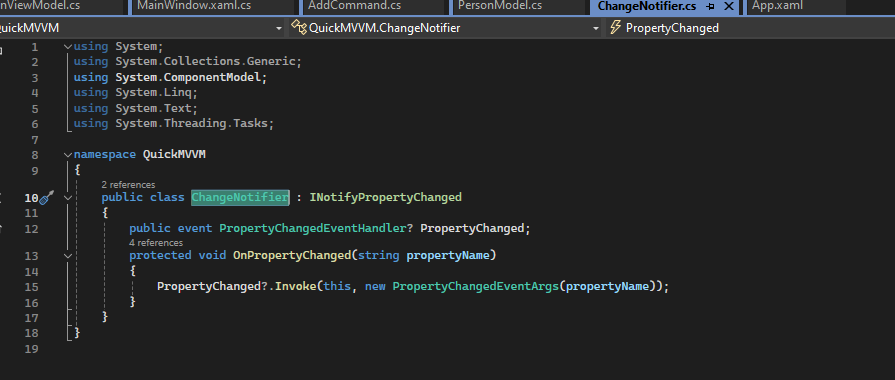


1. **ListView.ItemTemplate**: Define la plantilla que se utilizará para representar cada ítem en el ListView.
2. **DataTemplate**: Dentro de ItemTemplate, define la apariencia visual de cada ítem.
   * **WrapPanel**: Un panel que organiza sus hijos en una línea horizontal y luego en nuevas líneas según sea necesario.
     + **TextBlock Text="{Binding FirstName}"**: Muestra el valor de FirstName del objeto de datos vinculado.
     + **TextBlock Text=" - " Foreground="Blue"**: Muestra un guion con color azul entre FirstName y LastName.
     + **TextBlock Text="{Binding LastName}"**: Muestra el valor de LastName del objeto de datos vinculado.
     + **TextBlock Text=" - " Foreground="Blue"**: Muestra un guion con color azul entre LastName y Age.
     + **TextBlock Text="{Binding Age}"**: Muestra el valor de Age del objeto de datos vinculado.

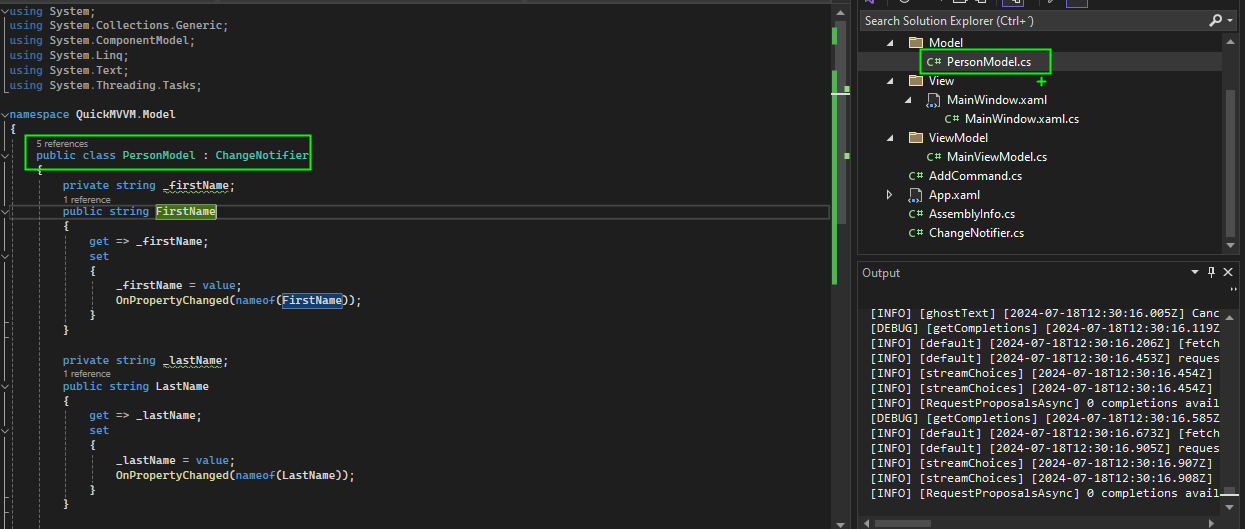
**Resumen**

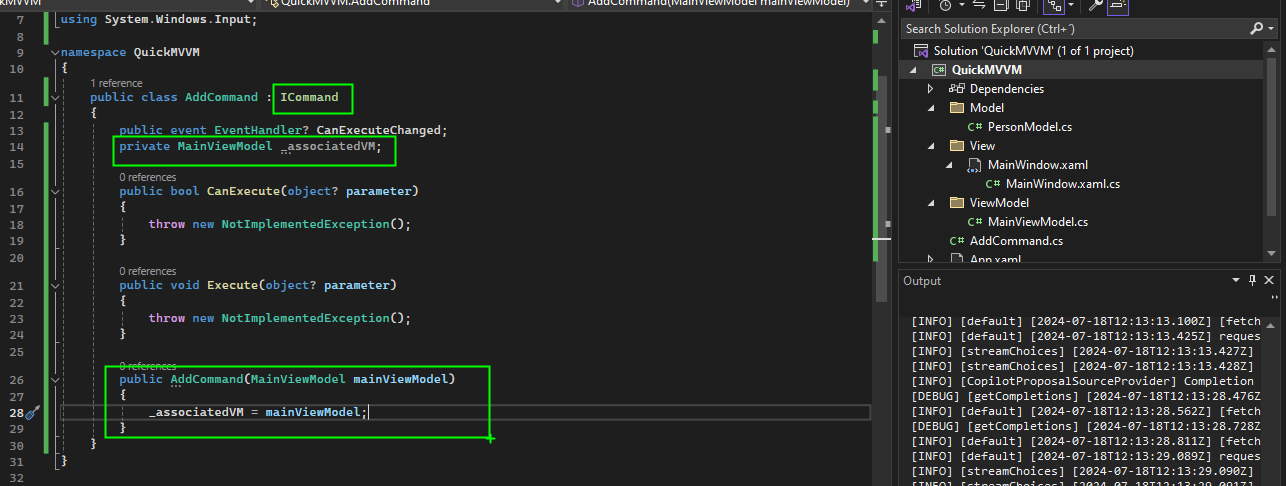
* **ItemTemplate**: Se utiliza para definir la plantilla de visualización para cada ítem en un ListView.
* **DataTemplate**: Define la estructura y el contenido visual de un ítem de datos específico dentro del ItemTemplate.

Creamos una clase **ChangeNotifier** que contendrá el **INotifyPropertyChanged**

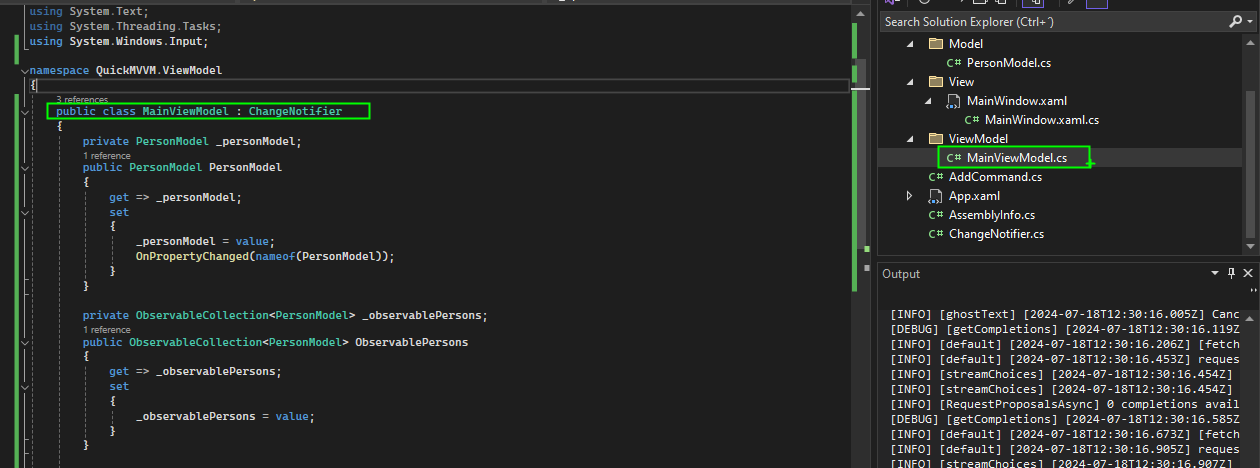


Ahora creamos nuestra clase de model dentro de la carpeta **Model** que se llamara **PersonModel** e implementara, al **ChangeNotifier**:

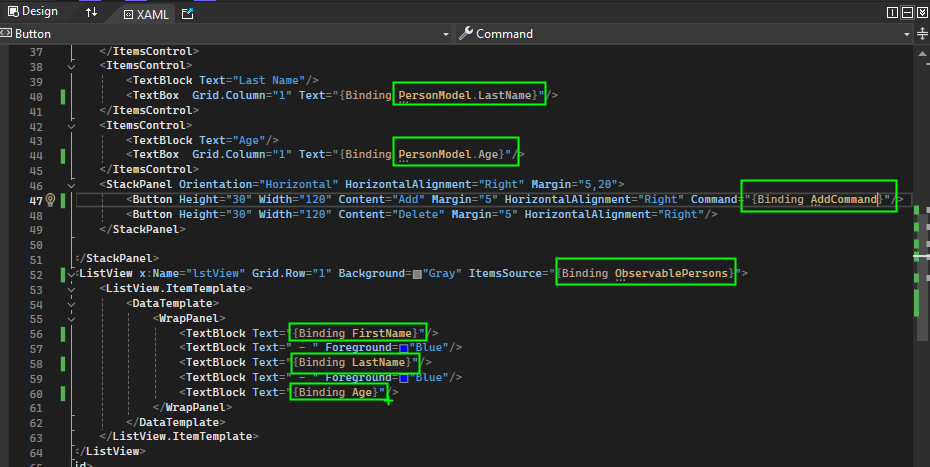


Ahora creamos la clase **ICommad** por fuera de la estructura de carpetas y ponemos por nombre **AddCommand**: 

Posteriormente crearemos nuestra clase **MainViewModel** dentro de la carpeta **ViewModel:**

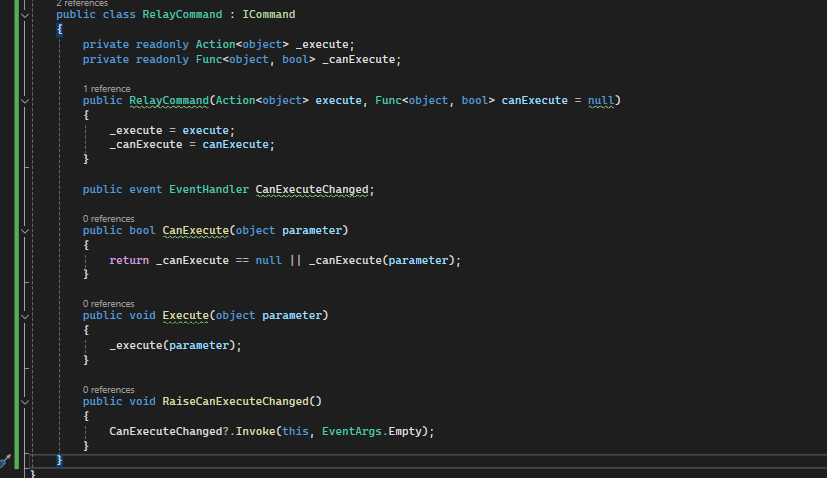
****

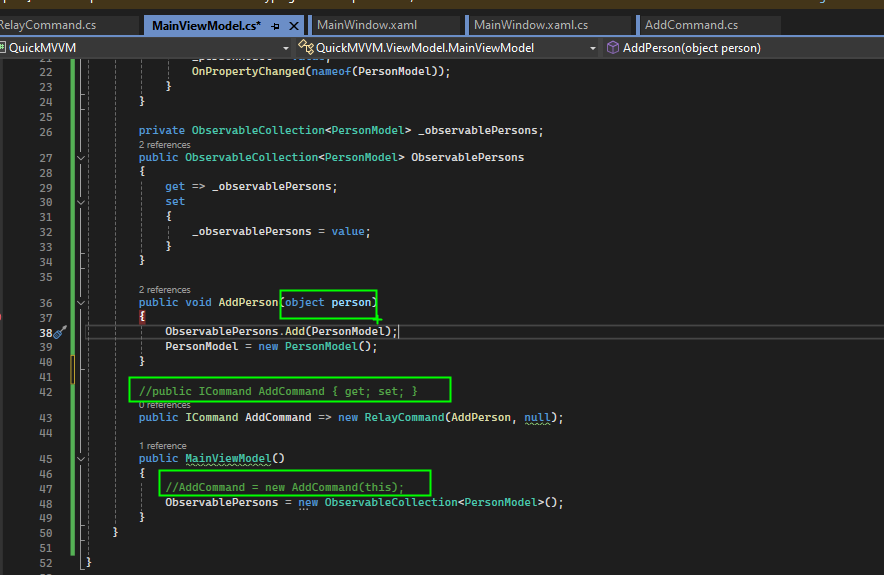
Seguidamente hacemos los bindings en la vista:

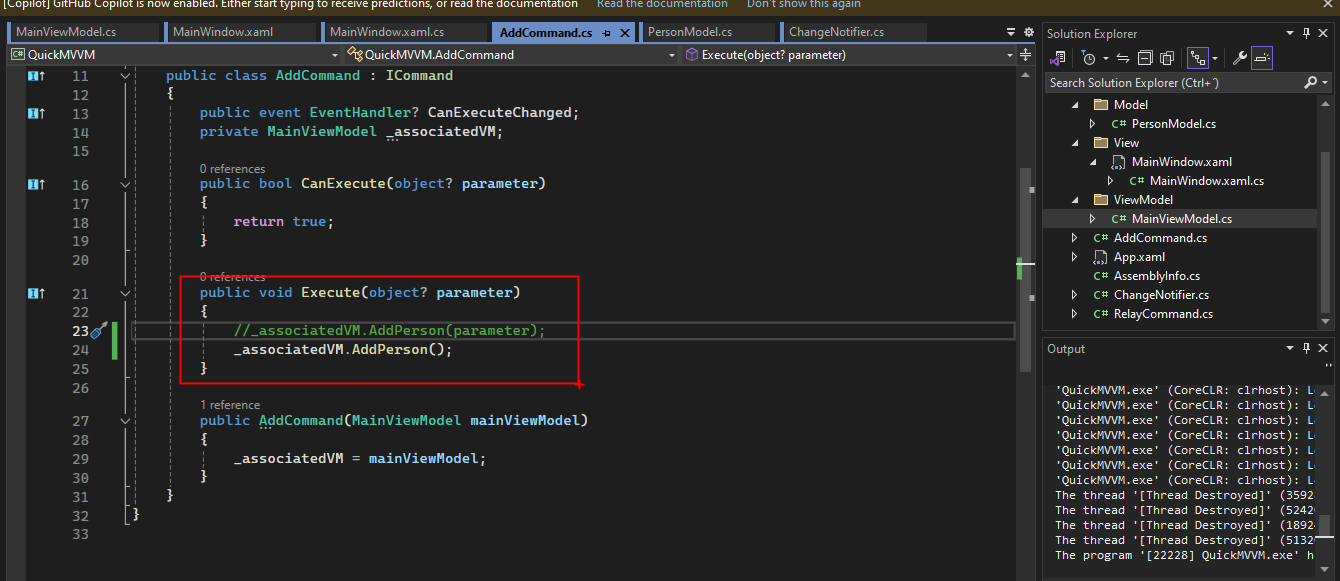


Y ejecutamos el proyecto.

Vamos a hacer un cambio vamos a ahora a crear un **RelayCommand** en vez del **AddCommand:**

****

Y en el **MainViewModel** hacemos estas modificaciones: 

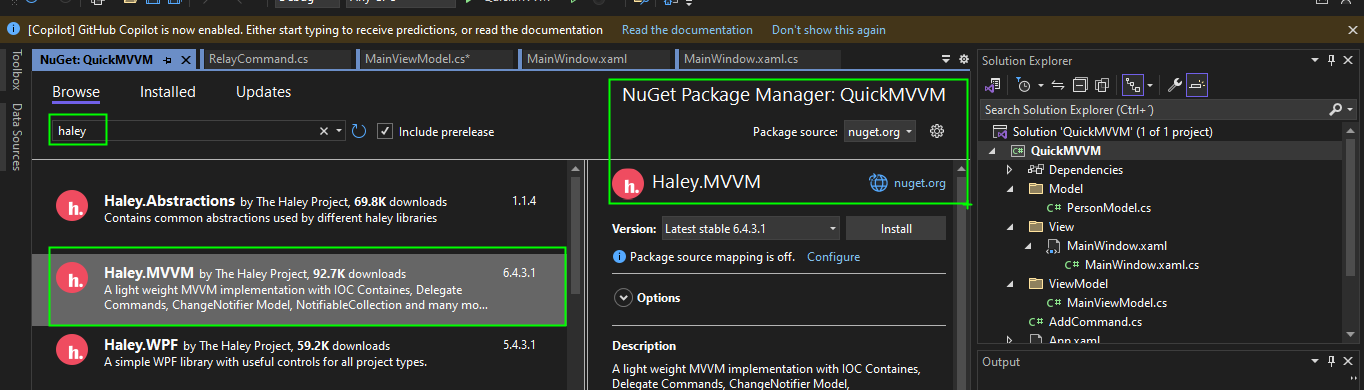


Y ejecutamos nuestro proyecto

Ahora cambiamos todo y probamos con **RelayCommand** en vez de solo **el get y set** del **AddCommand.**

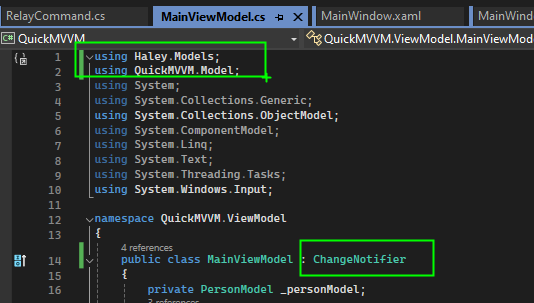
**Copiamos nuestro proyecto y empezamos uno nuevo:**

Ahora vamos a probar instalando una librería:

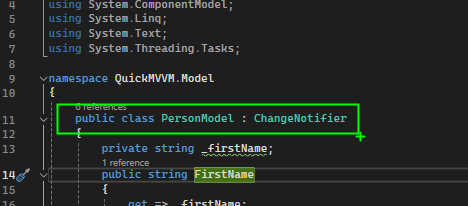


Ya que tiene mucha funcionalidad incluida como **INotifyPropertyChanged**, **ICommand**, **inyección** **de dependencias, etc**

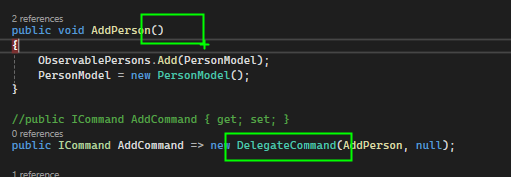
Ahora eliminamos nuestra clase **ChangeNotifier** e implementamos en nuestro **MainViewModel** el **ChangeNotifier** propio de la librería **Haley**

****

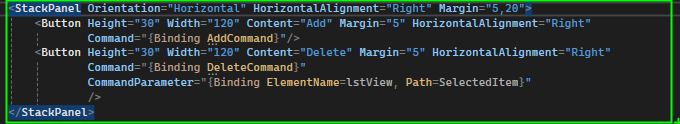
Lo mismo hacemos en nuestra clase **PersonModel**



Y cambiamos en nuestra clase **MainViewModel** a nuestro **RelayCommand** por un **DelegateCommand :**

****

Ahora agregamos el botón de **Delete:**

****

Y en el **listView** cambiamos:



 **Command="{Binding DeleteCommand}"**:

* **Descripción:** Enlaza el comando DeleteCommand desde el ViewModel a la acción del botón.
* **Propósito:** Permite que al hacer clic en el botón, se ejecute la lógica definida en el DeleteCommand en el ViewModel.

 **CommandParameter="{Binding ElementName=lstView, Path=SelectedItem}"**:

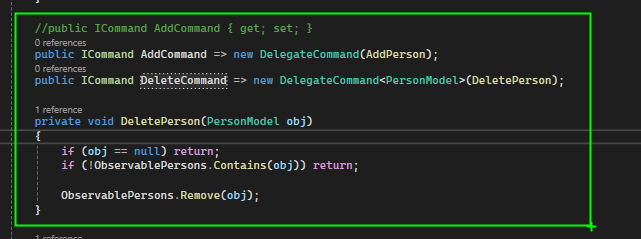
* **Descripción:** Define el parámetro que se pasará al comando cuando se ejecute.
* **Propósito:** El parámetro que se pasa al DeleteCommand es el ítem actualmente seleccionado en el ListView llamado lstView. Esto permite que el comando sepa qué ítem debe ser eliminado.

**Resumen**

* **Command="{Binding AddCommand}"**: Vincula el botón "Add" con el comando AddCommand del ViewModel, permitiendo agregar personas a la lista.
* **Command="{Binding DeleteCommand}" y CommandParameter="{Binding ElementName=lstView, Path=SelectedItem}"**: Vincula el botón "Delete" con el comando DeleteCommand del ViewModel y le pasa como parámetro el ítem seleccionado en el ListView llamado lstView, permitiendo eliminar personas de la lista.

Estos comandos y sus parámetros facilitan la interacción entre la interfaz de usuario y la lógica del ViewModel en una aplicación WPF siguiendo el patrón MVVM.

Y agregamos la funcionalidad de **Delete:**

****