

# **Pruebas unitarias. Personalización y persistencia del interfaz. Logging**



**Escuela Politécnica Superior  
Universidad de Alicante**

# Índice

- ▶ Pruebas unitarias
- ▶ Persistencia del estado del interfaz
  - ▶ Formulario de configuración
  - ▶ Cambios en el interfaz
- ▶ I18n
- ▶ Logging

# Stubs y pruebas unitarias

# **Stubs y pruebas unitarias**

- ▶ Es importante asegurar la corrección de cada parte del código a lo largo del ciclo de vida iterativo
  - evitando errores de regresión cuando hagamos cambios
- ▶ Podemos hacer prints y ver qué valores van teniendo los objetos
  - mejor lo podemos automatizar con pruebas unitarias: hacemos un programa que cada método hace lo que debe
  - esto lo aplicamos desde el primer momento a los subsistemas

# **Stubs y pruebas unitarias**

- ▶ Supongamos que queremos hacer una aplicación que lea o escriba en un dispositivo / sistema que no tenemos disponible o que aún no esté desarrollada
- ▶ No podemos esperar a tener esa disponibilidad para desarrollar el frontend
  - ▶ o queremos disponer de datos y comportamiento controlados
- ▶ Solución:
  - desacoplamos usando una interfaz (creamos un subsistema)
  - implementándola primero con clases con contenido “enlatado” denominadas Stubs
  - en la solución final se usará la implementación correcta mediante inyección de código o una factory

# Ejemplo

- ▶ Pantalla control sistema domótica
- ▶ Aunque dispusiéramos de la domótica implementada no vamos a hacer pruebas continuas de apertura y cierre de persianas, puertas, etc...
- ▶ Creamos un nuevo proyecto XAML Domotica y dentro un interfaz (botón derecho sobre proyecto > agregar nuevo elemento > Interfaz) IConectorSistemaDomotica que pondremos en el paquete domotica.io (en los ejemplos pone por error Domitica)

```
namespace Domitica.io
{
    /// <summary>
    /// Las persianas están por defecto bajadas (tanto por cierto apertura es 0)
    /// </summary>
    public interface IConectorSistemaDomitica
    {
        // devuelve el tanto por ciento entre 0 y 1 de apertura de la persiana
        double subirPersiana();

        // devuelve el tanto por ciento entre 0 y 1 de apertura de la persiana
        double bajarPersiana();
    }
}
```

# Stub

- ▶ Como no disponemos del hardware de domótica, vamos a implementar una clase que lo simule: un stub (siguiente diapositiva)

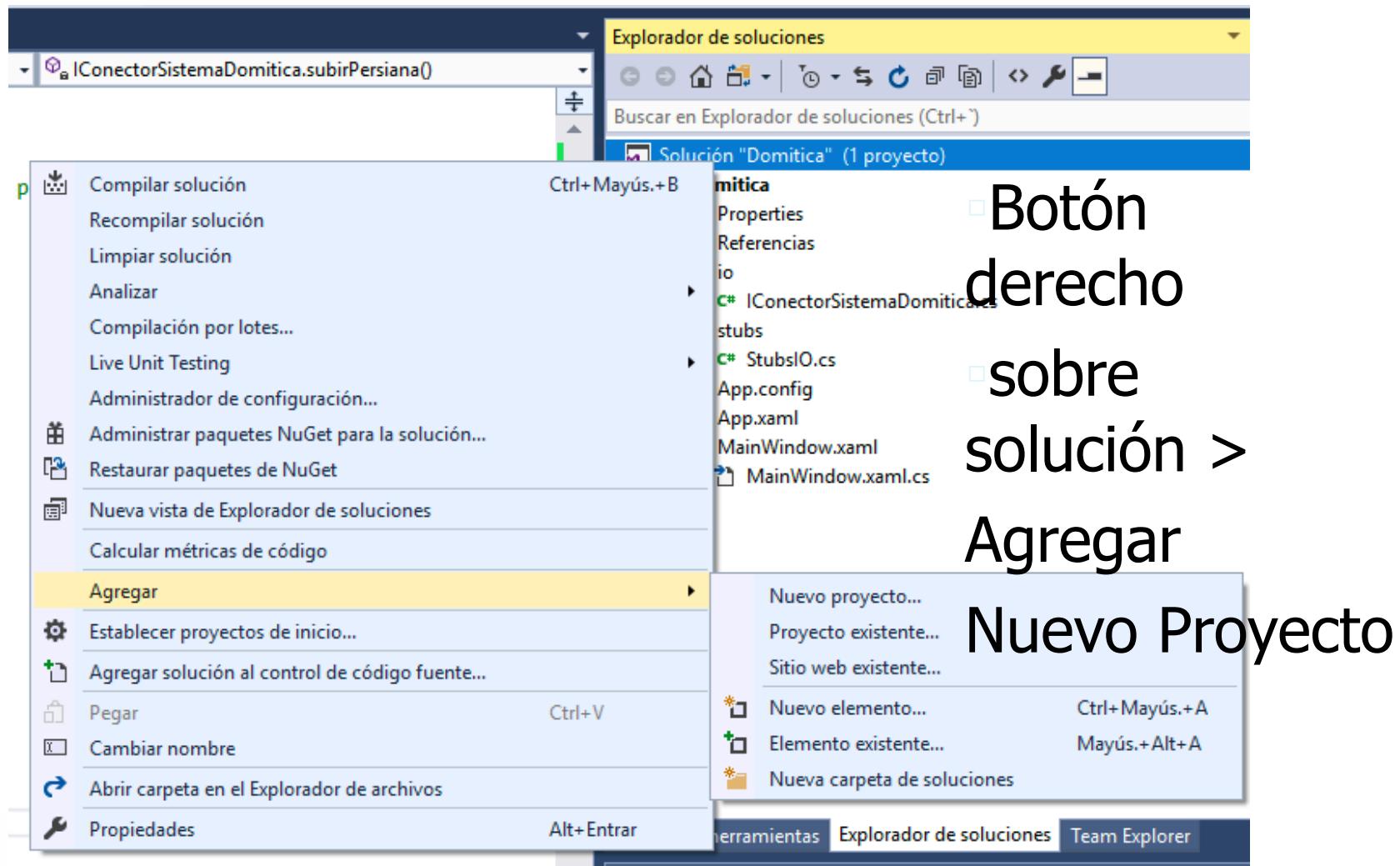
```
namespace Domitica.stubs
{
    /// <summary>
    /// Esta clase simula la subida y bajada de la persiana un 10% en cada petición
    /// </summary>
    public class ConecotorSistemaDomoticaStub : IConecotorSistemaDomotica
    {
        double posicionPersiana = 0;

        double IConecotorSistemaDomotica.bajarPersiana()
        {
            if (posicionPersiana <= 0.9)
            {
                posicionPersiana += 0.1;
            } else
            {
                posicionPersiana -= 0.1;
            }
            return posicionPersiana;
        }

        double IConecotorSistemaDomotica.subirPersiana()
        {
            if (posicionPersiana >= 0.1)
            {
                posicionPersiana -= 0.1;
            } else
            {
                posicionPersiana = 0;
            }
            return posicionPersiana;
        }
    }
}
```

Stub

# Creamos el proyecto de test unitario



## Agregar nuevo proyecto

? X

Recientes

Instalado

Visual C#

- Windows Universal
- Escritorio de Windows
- .NET Core
- .NET Standard
- Prueba**

Visual Basic

Visual C++

JavaScript

En línea

Ordenar por: Predeterminado



Buscar (Ctrl+E)



Proyecto de prueba de MSTest (.NET Core)

Visual C#

Tipo: Visual C#

Proyecto que contiene pruebas unitarias.



Proyecto de pruebas xUnit (.NET Core)

Visual C#



Proyecto de prueba unitaria (.NET Framework)

Visual C#

En VS 2019: C#, Windows, Prueba  
Proyecto de prueba unitaria (.NET Framework)

¿No encuentra lo que busca?

[Abrir el instalador de Visual Studio](#)

Nombre:

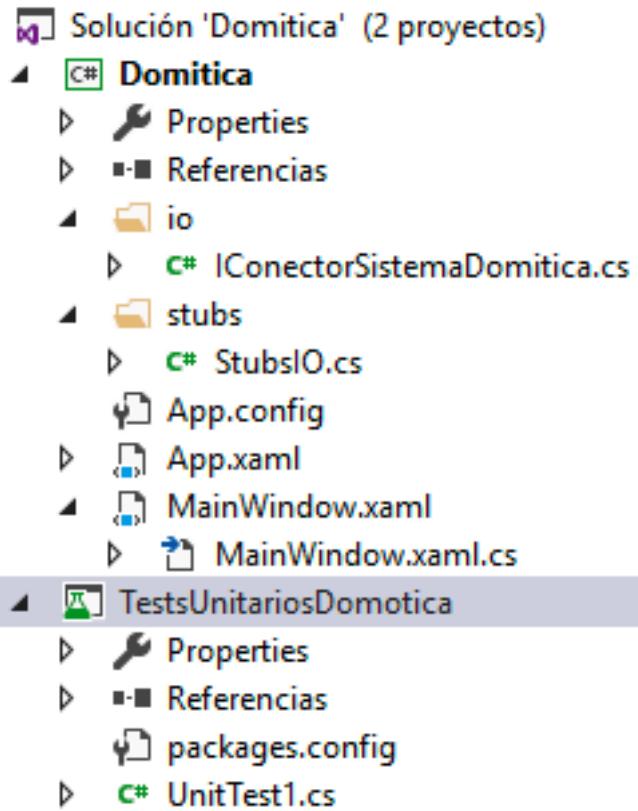
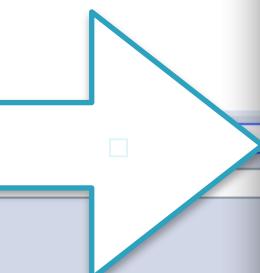
TestsUnitariosDomotica

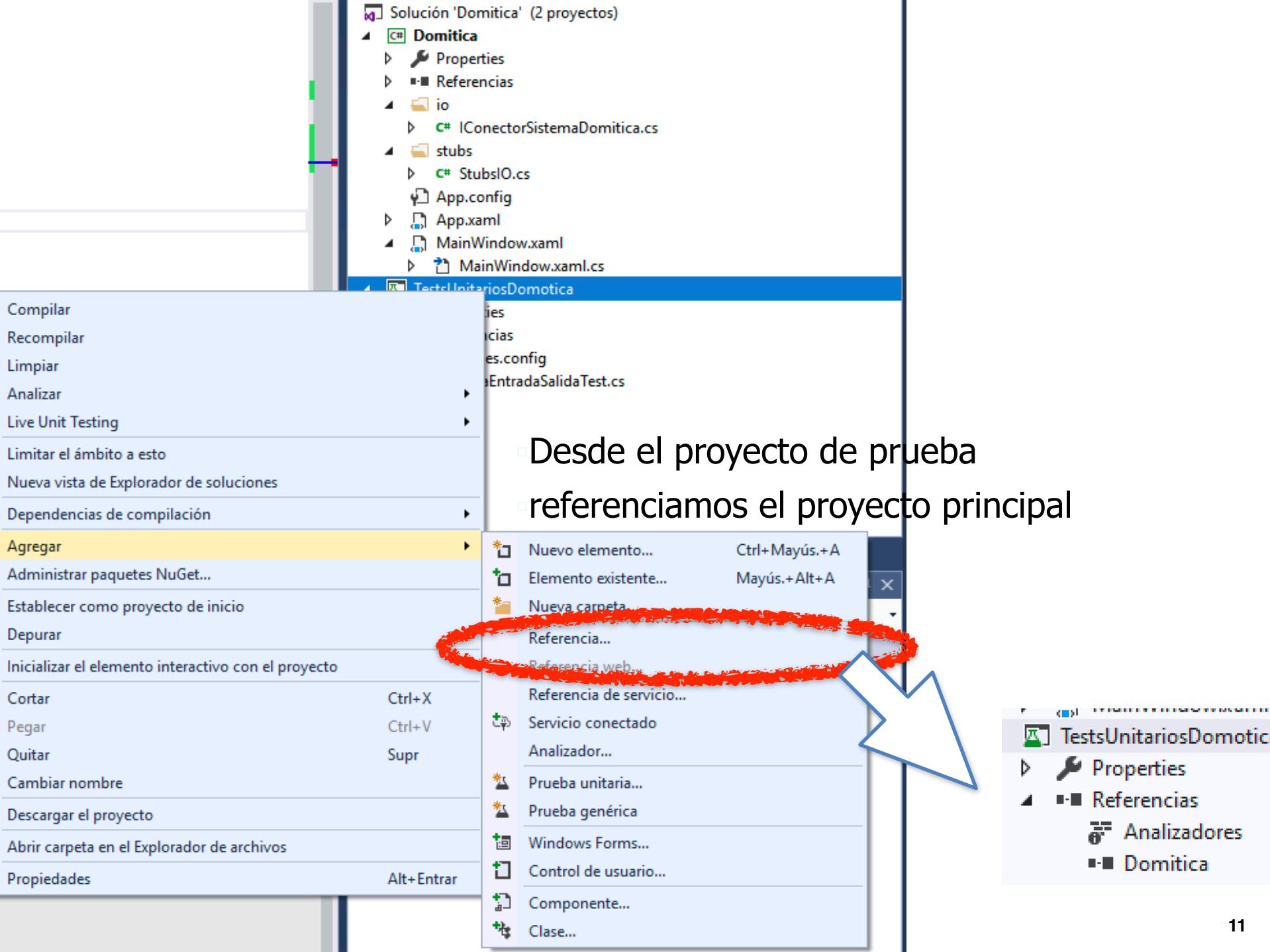
Ubicación:

C:\Users\drizo\source\repos\Domotica

Framework:

.NET Framework 4.6.1





■ Desde el proyecto de prueba  
referenciamos el proyecto principal

# Creamos la prueba

```
        posicionPersiar
    } else
    {
        posicionPersiar
    }
}
return posicionPersiar;
}

double IConektorSistema
{
    if (posicionPersiar
}
```

Explorad

Propieda

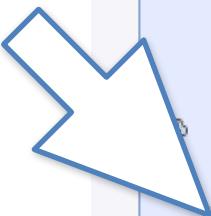
TestsUn

Archiv

Carpet

- \* Nuevo elemento... Ctrl+Mayús.+A
- + Elemento existente... Mayús.+Alt+A
- \* Nueva carpeta
- Cliente de API REST...
- Referencia...
- Referencia web...
- Referencia de servicio...
- Servicio conectado
- Analizador...
- \* Prueba unitaria...
- + Windows Forms...
- + Control de usuario (Windows Forms)...
- + Componente...
- + Clase...

- App.xaml
- MainWindow.xaml
- Compilar
- Recompilar
- Limpiar
- Análisis y limpieza de código
- Ejecutar pruebas
- Depurar pruebas
- Live Unit Testing
- Limitar el ámbito a esto
- Nueva vista de Explorador de soluciones
- Dependencias de compilación
- Agregar
- Administrar paquetes NuGet...
- Establecer como proyecto de inicio
- Depurar
- Inicializar el elemento interactivo con el proyecto
- Control de código fuente
- Cortar Ctrl+X
- Pegar Ctrl+V
- Quitar Supr
- Cambiar nombre
- Descargar el proyecto
- Cargar dependencias del proyecto
- Abrir carpeta en el Explorador de archivos
- Propiedades Alt+Entrar



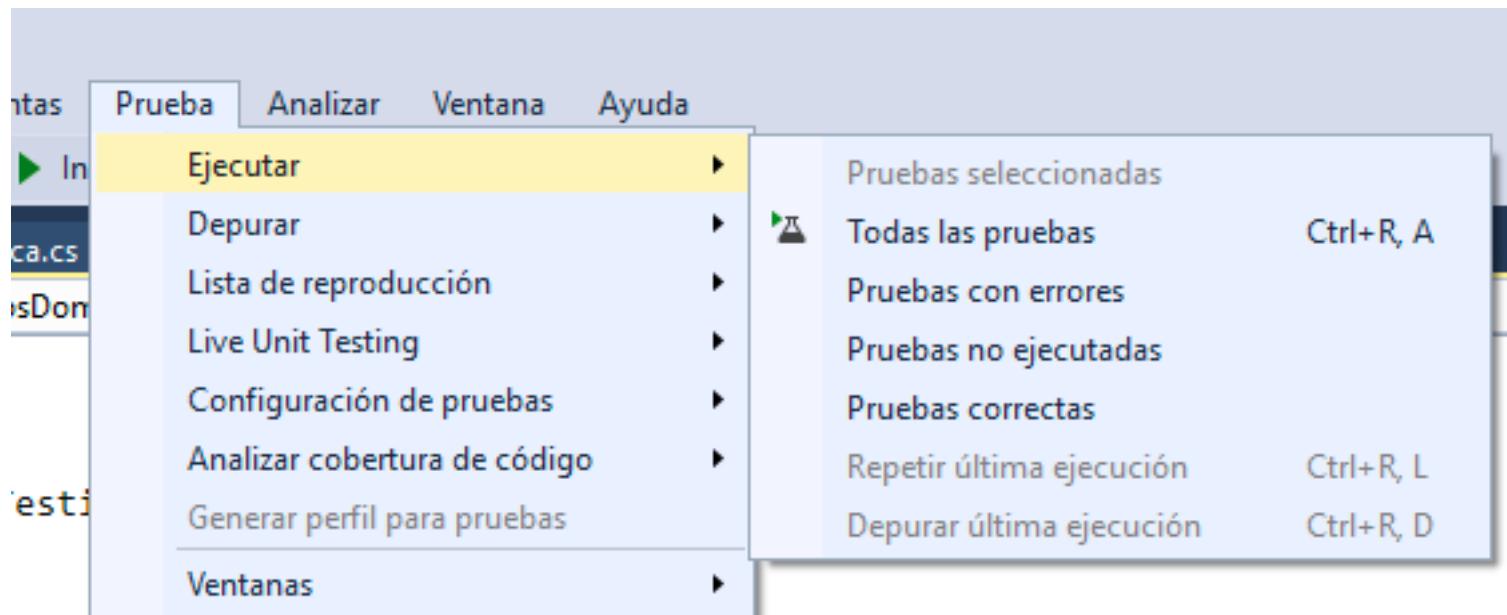
# Creamos la prueba

```
namespace TestsUnitariosDomotica
{
    [TestClass]
    public class SistemaEntradaSalidaTest
    {
        [TestMethod]
        public void TestSubida()
        {
            IConectorSistemaDomotica conector = new ConectorSistemaDomoticaStub();

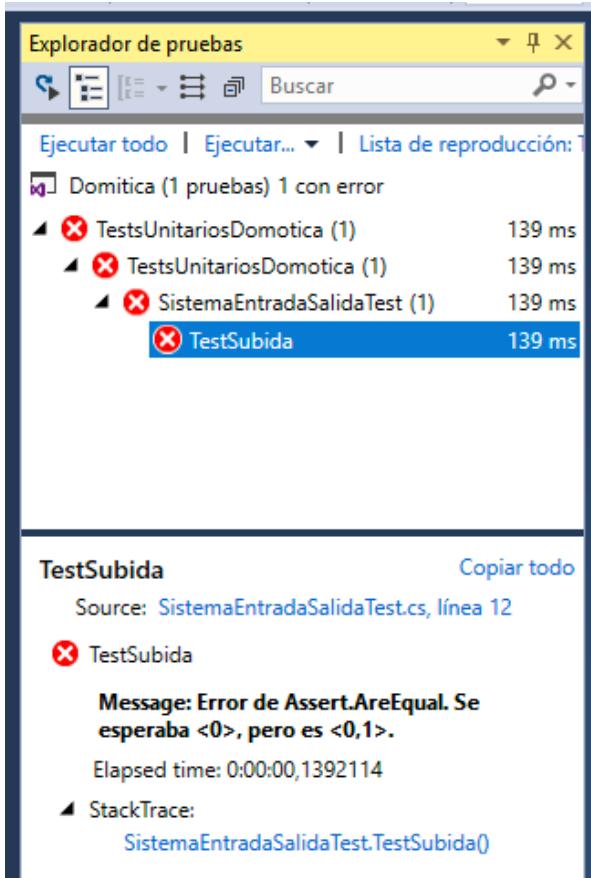
            // comprobamos el estado inicial que debe ser 0 (porque nosotros lo hemos establecido así)
            double posicion = conector.bajarPersiana(); // no podrá bajar más
            Assert.AreEqual(0.0, posicion);

            double nuevaPosicion = conector.subirPersiana();
            Assert.IsTrue(nuevaPosicion > posicion); // sólo puedo comprobar que ha subido, no el valor exacto
        }
    }
}
```

# Ejecutamos la prueba



# Prueba con errores



□ Teníamos mal el stub:  
estaba al revés  
(subía en lugar de bajar)

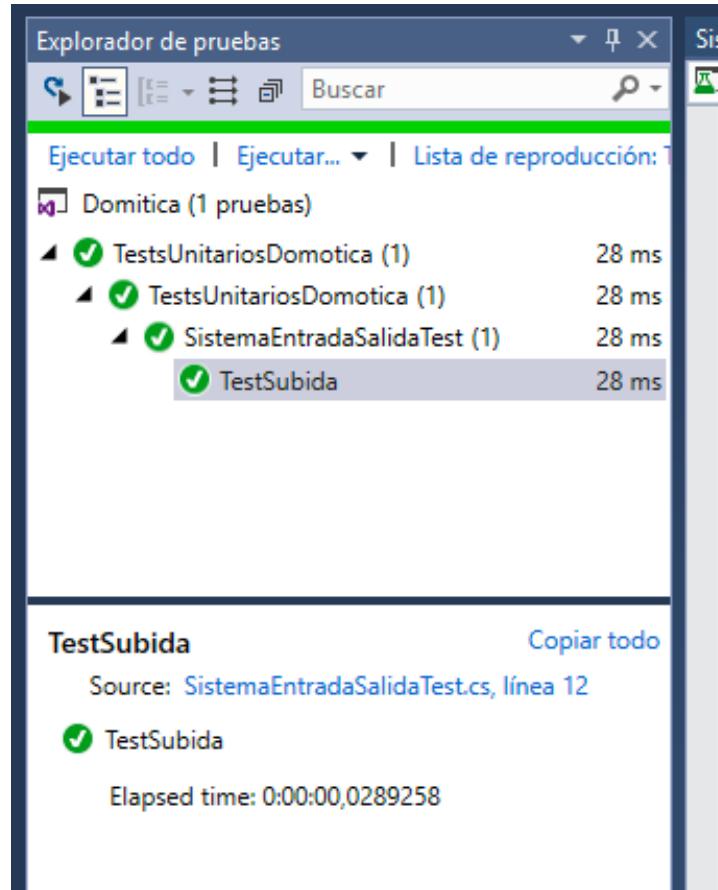
# Stub corregido

```
public class ConectorSistemaDomoticaStub : IConectorSistemaDomotica
{
    double posicionPersiana = 0;

    double IConectorSistemaDomotica.bajarPersiana()
    {
        if (posicionPersiana >= 0.1)
        {
            posicionPersiana -= 0.1;
        } else
        {
            posicionPersiana = 0;
        }
        return posicionPersiana;
    }

    double IConectorSistemaDomotica.subirPersiana()
    {
        if (posicionPersiana <= 0.9)
        {
            posicionPersiana += 0.1;
        } else
        {
            posicionPersiana = 1;
        }
        return posicionPersiana;
    }
}
```

# Prueba sin errores



# Actividad clase 1

- ▶ Crear una prueba unitaria que compruebe que la persiana sube y baja del todo en un número finito de pasos

# Actividad opcional

- ▶ Crear una pequeña interfaz gráfica que permita visualizar el estado de las ventanas del sistema domótico usando el Stub

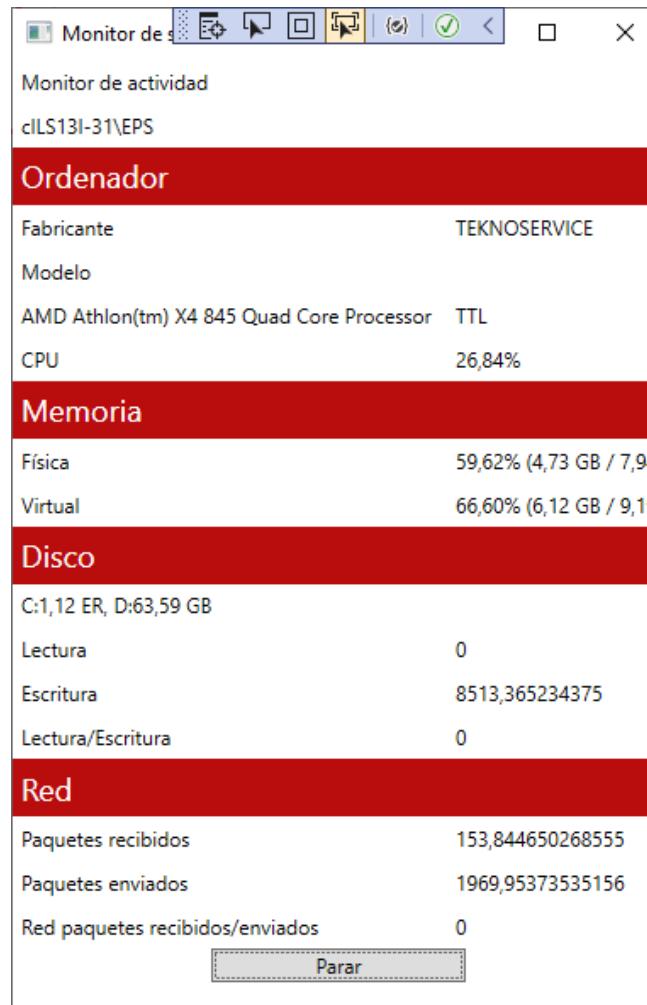
# Personalización (WPF)

# Estrategia

- ▶ Vamos a crear un objeto estático llamado Config (podría tener cualquier otro nombre) donde guardaremos las propiedades personalizadas
  - ▶ Para personalizar un formulario modificará sus valores
  - ▶ Para hacerlas persistentes las guardaremos en el registro de Windows
  - ▶ Para cargarlas de nuevo las leeremos del registro

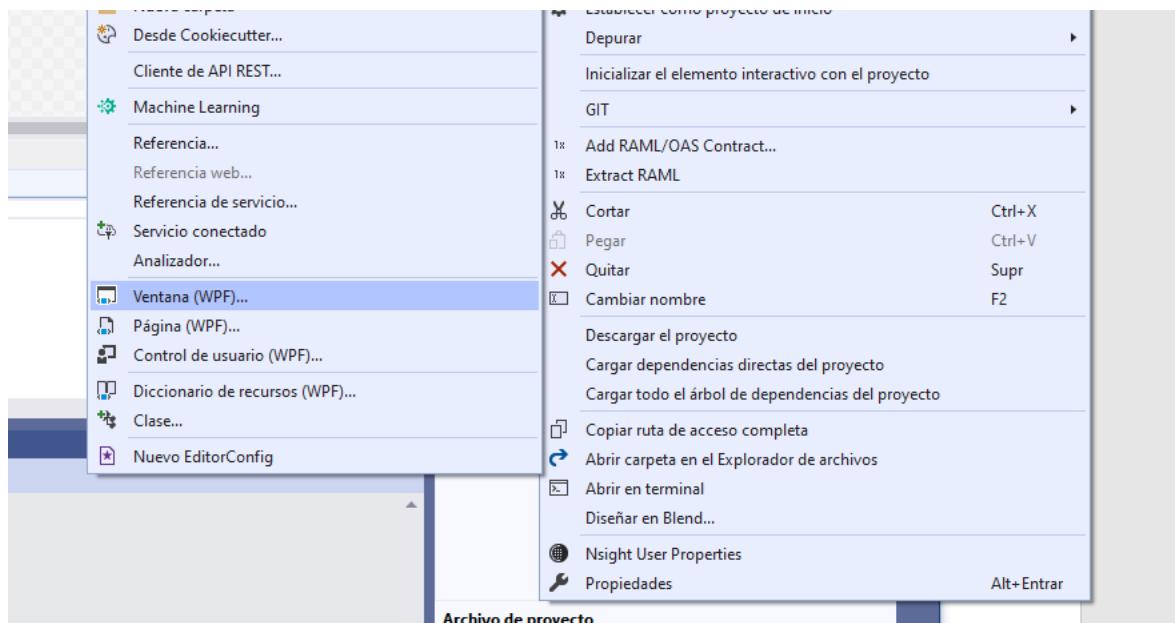
# Aplicación de ejemplo

- Partimos de la aplicación WPF del monitor de sistema (disponible en Moodle)



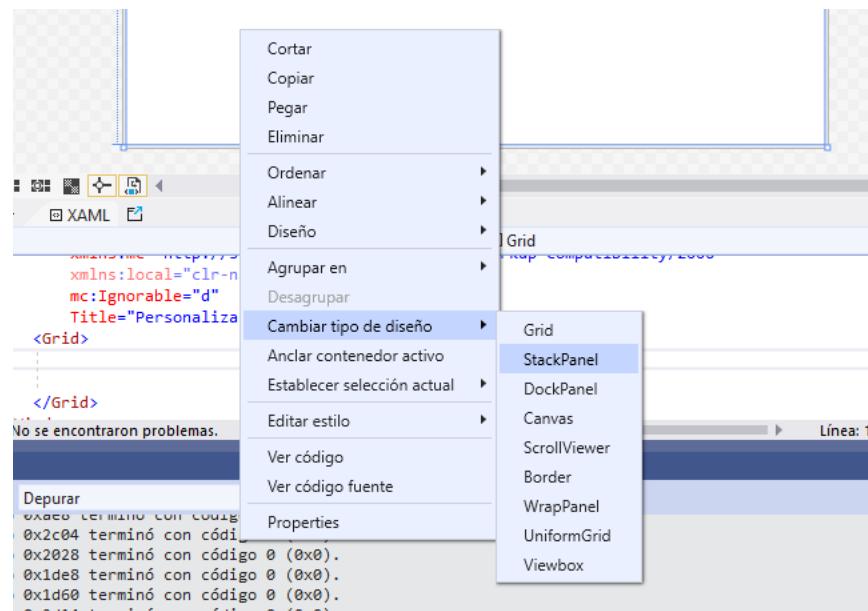
# Diálogo de configuración / personalización

- Crearemos una nueva ventana donde pondremos el diálogo con los datos de personalización.
  - Lo llamaremos WindowConfig. Cambiamos el título



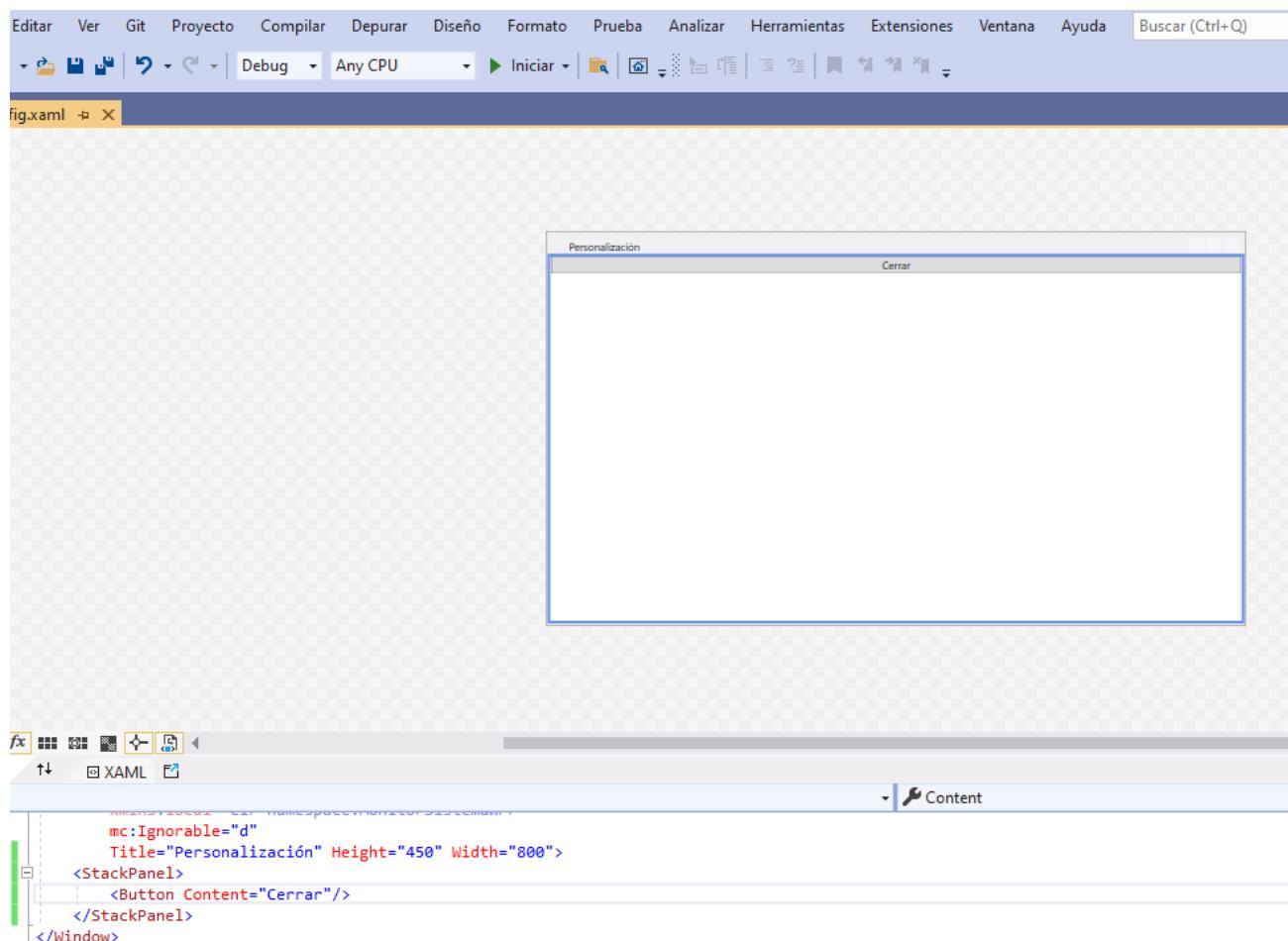
# Diálogo de configuración / personalización

- Cambiamos el Grid por un StackPanel



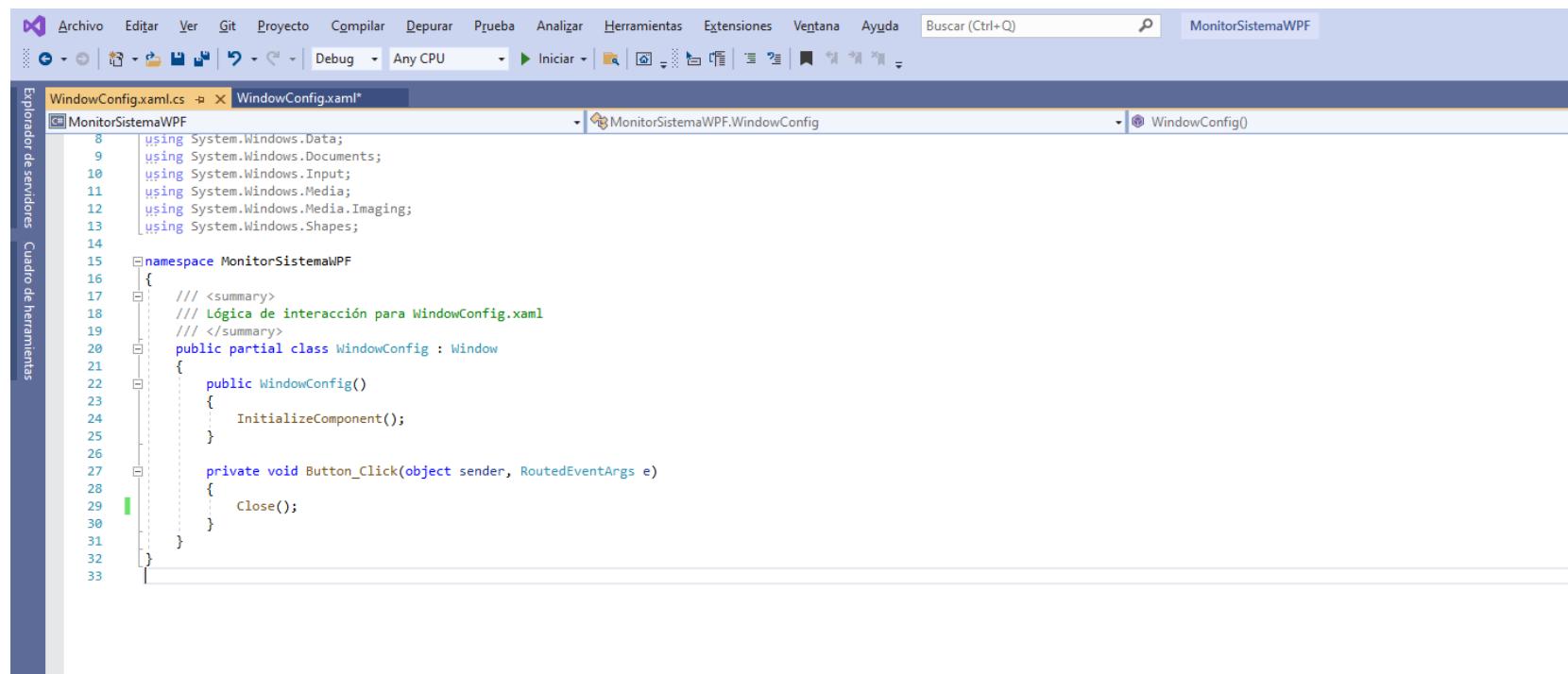
# Diálogo de configuración / personalización

- Añadimos un Botón “Cerrar” (luego agregaremos el resto de contenidos)



# Diálogo de configuración / personalización

- Creamos el código para que cuando el usuario pulse en “Cerrar” se cierre el diálogo

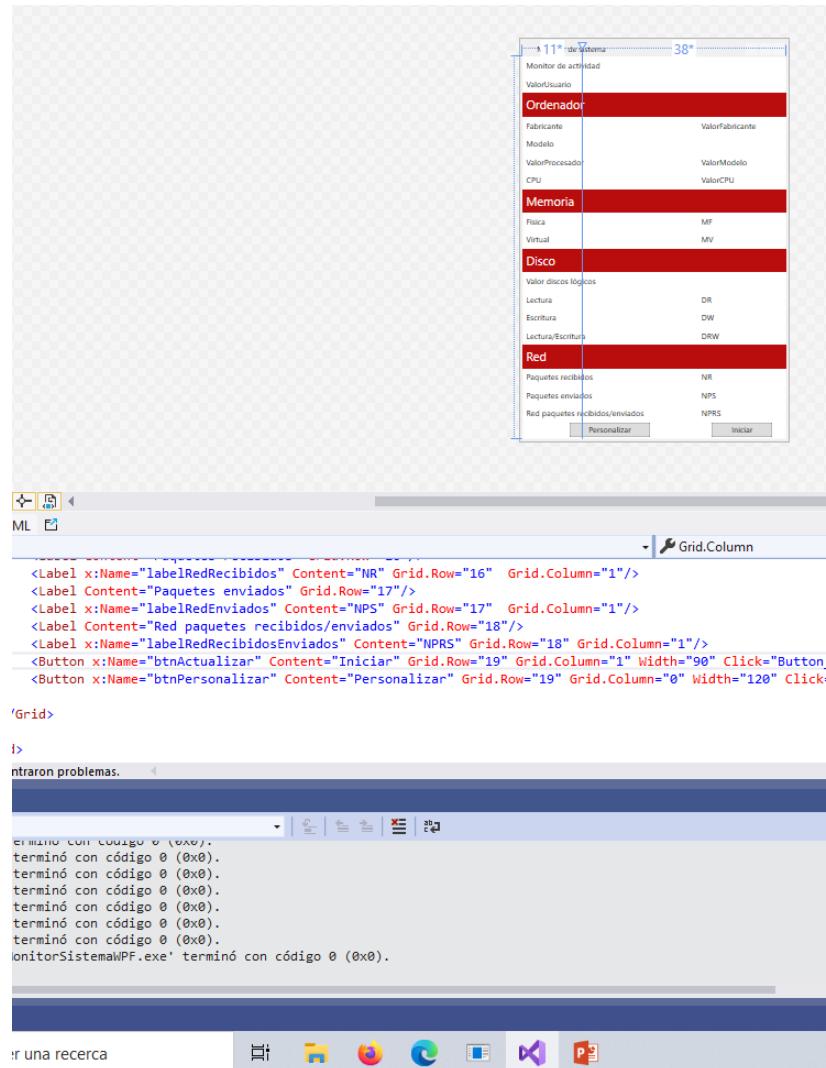


The screenshot shows the Microsoft Visual Studio IDE interface. The title bar reads "MonitorSistemaWPF". The menu bar includes Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda. The toolbar has various icons for file operations like Open, Save, and Build. The status bar at the bottom shows "MonitorSistemaWPF.WindowConfig" and "WindowConfig()". The main window displays two tabs: "WindowConfig.xaml.cs" and "WindowConfig.xaml". The code editor contains the following C# code:

```
8  using System.Windows.Data;
9  using System.Windows.Documents;
10 using System.Windows.Input;
11 using System.Windows.Media;
12 using System.Windows.Media.Imaging;
13 using System.Windows.Shapes;
14
15 namespace MonitorSistemaWPF
16 {
17     /// <summary>
18     /// Lógica de interacción para WindowConfig.xaml
19     /// </summary>
20     public partial class WindowConfig : Window
21     {
22         public WindowConfig()
23         {
24             InitializeComponent();
25         }
26
27         private void Button_Click(object sender, RoutedEventArgs e)
28         {
29             Close();
30         }
31     }
32 }
33
```

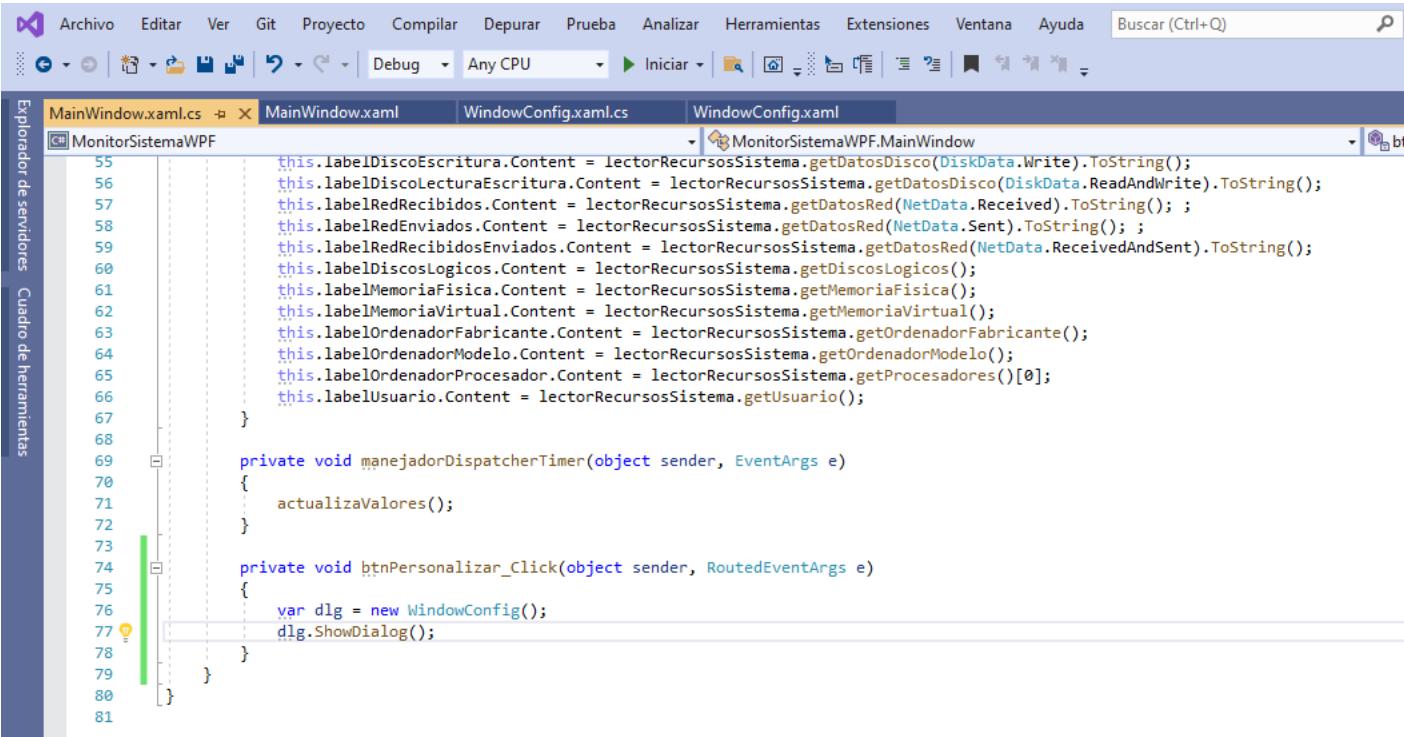
# Diálogo de configuración / personalización

- Incluimos un nuevo botón que muestre el diálogo de personalización



# Diálogo de configuración / personalización

- Añadimos el código para mostrar el diálogo (ventana modal)

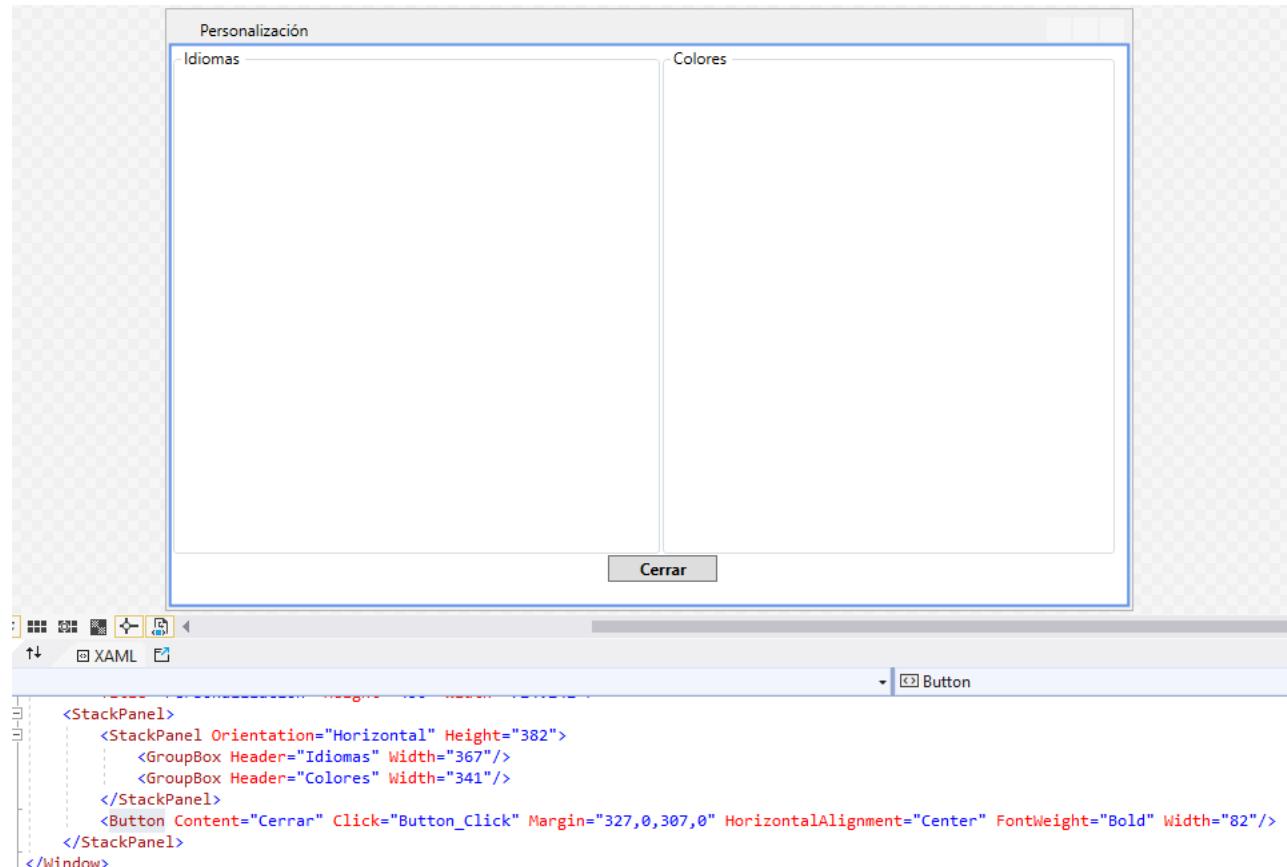


The screenshot shows the Microsoft Visual Studio IDE interface. The title bar displays "MonitorSistemaWPF". The menu bar includes Archivo, Editar, Ver, Git, Proyecto, Compilar, Depurar, Prueba, Analizar, Herramientas, Extensiones, Ventana, Ayuda, and Buscar (Ctrl+Q). The toolbar contains various icons for file operations like Open, Save, and Build. The solution explorer on the left shows files: MainWindow.xaml.cs, MainWindow.xaml, WindowConfig.xaml.cs, and WindowConfig.xaml. The main code editor window displays C# code for a class named "MonitorSistemaWPF.MainWindow". The code includes methods for updating labels with system resource data and handling a button click event to show a configuration dialog.

```
55     this.labelDiscoEscritura.Content = lectorRecursosSistema.getDatosDisco(DiskData.Write).ToString();
56     this.labelDiscoLecturaEscritura.Content = lectorRecursosSistema.getDatosDisco(DiskData.ReadAndWrite).ToString();
57     this.labelRedRecibidos.Content = lectorRecursosSistema.getDatosRed(NetData.Received).ToString(); ;
58     this.labelRedEnviados.Content = lectorRecursosSistema.getDatosRed(NetData.Sent).ToString(); ;
59     this.labelRedRecibidosEnviados.Content = lectorRecursosSistema.getDatosRed(NetData.ReceivedAndSent).ToString();
60     this.labelDiscosLogicos.Content = lectorRecursosSistema.getDiscosLogicos();
61     this.labelMemoriaFisica.Content = lectorRecursosSistema.getMemoriaFisica();
62     this.labelMemoriaVirtual.Content = lectorRecursosSistema.getMemoriaVirtual();
63     this.labelOrdenadorFabricante.Content = lectorRecursosSistema.getOrdenadorFabricante();
64     this.labelOrdenadorModelo.Content = lectorRecursosSistema.getOrdenadorModelo();
65     this.labelOrdenadorProcesador.Content = lectorRecursosSistema.getProcesadores()[0];
66     this.labelUsuario.Content = lectorRecursosSistema.getUsuario();
67 }
68
69 private void manejadorDispatcherTimer(object sender, EventArgs e)
70 {
71     actualizaValores();
72 }
73
74 private void btnPersonalizar_Click(object sender, RoutedEventArgs e)
75 {
76     var dlg = new WindowConfig();
77     dlg.ShowDialog();
78 }
79 }
80 }
81 }
```

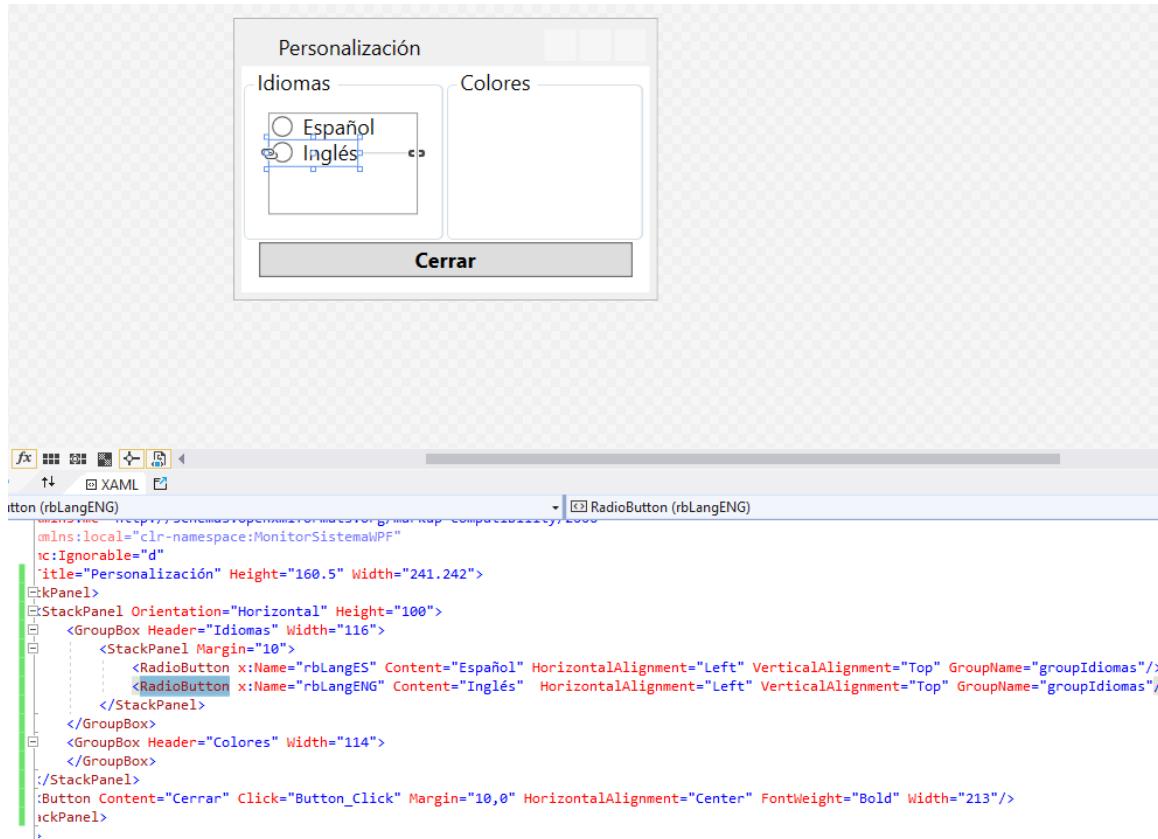
# Diálogo de personalización

- El diálogo será como éste usando un StackPanel con orientación horizontal y dos GroupBox



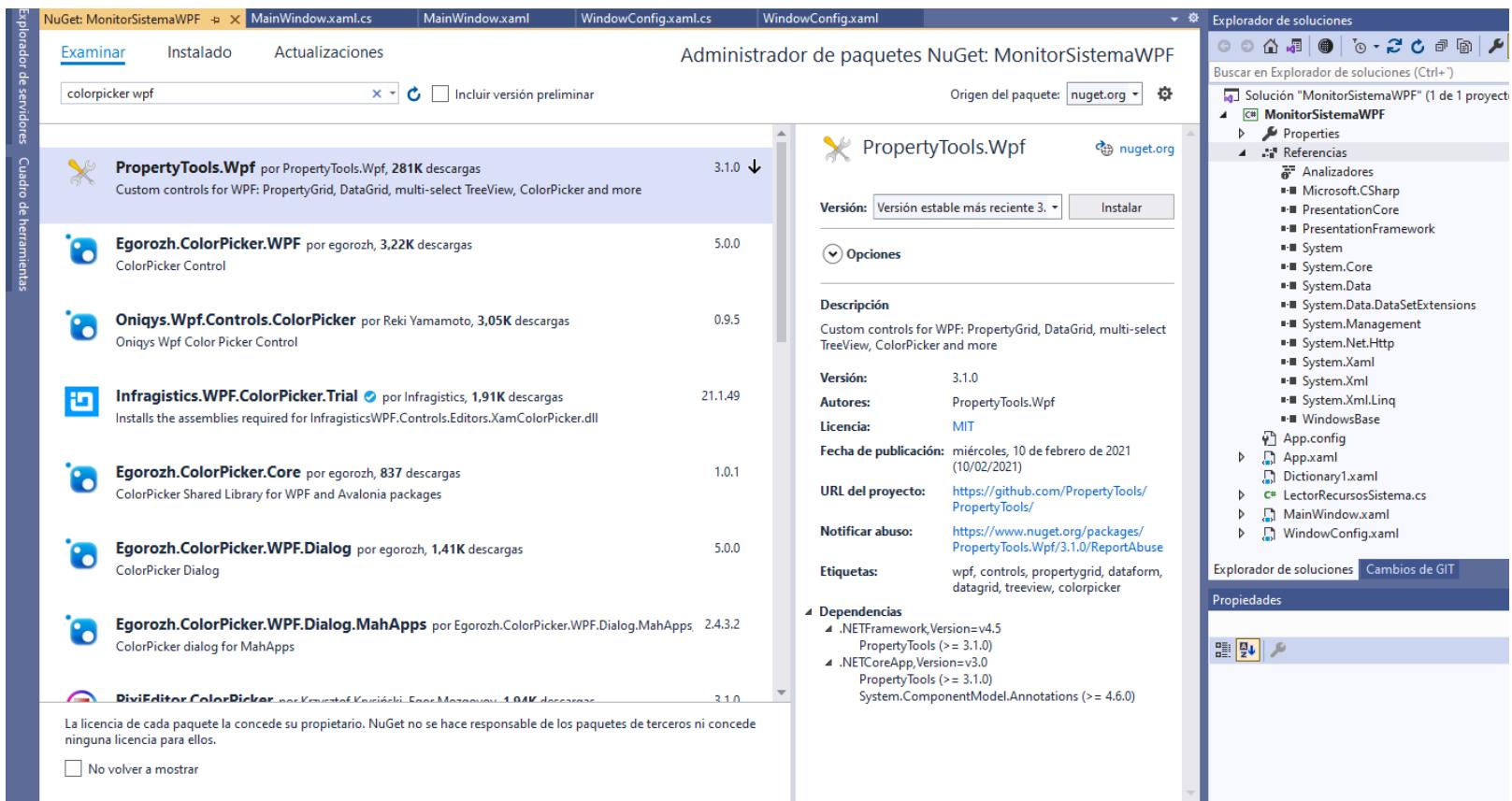
# Diálogo de personalización

- Dentro añadimos RadioButtons dentro, ambos con el mismo GroupName para que sean exclusivos, y alineados usando un StackPanel



# Diálogo de personalización

- Instalamos un ColorPicker de una librería a través de NuGet (WPF no lo incorpora como WinForms)



# Diálogo de personalización

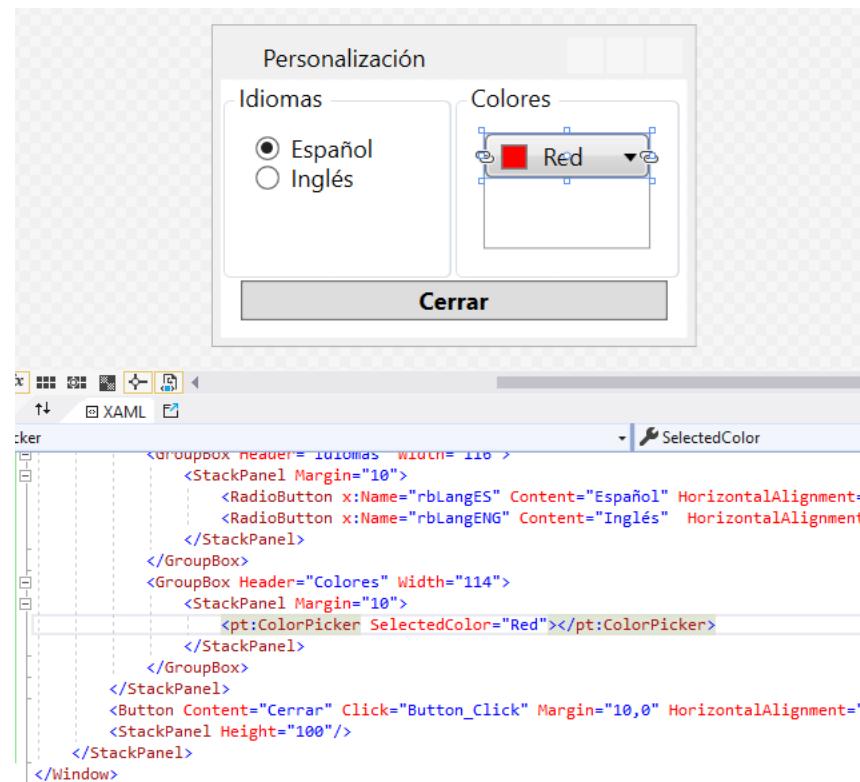
- Para usarlo deberemos agregar el namespace de la librería

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays XAML code for a user control named 'ColorPicker'. The code includes a StackPanel with various child elements like GroupBoxes and RadioButtons. A tooltip is visible, stating 'Agregar xmlns http://propertytools.org/wpf' with a warning 'XAML0414 Falta el espacio de nombres.' Below the code editor, the 'Lista de errores' (Error List) shows '0 de 2 Errores'. A detailed error message is shown in a callout box, indicating a missing namespace declaration for 'pt:ColorPicker'. The status bar at the bottom shows the search bar 'Escriu ací per fer una recerca' and several icons for file operations.

```
<StackPanel Orientation="Horizontal" Height="100">
    <GroupBox Header="Idiomas" Width="116">
        <StackPanel Margin="10">
            <RadioButton x:Name="rbLangES" Content="Español" HorizontalAlignment="Left" VerticalAlignment="Top" GroupName="groupIdoma" />
            <RadioButton x:Name="rbLangENG" Content="Inglés" HorizontalAlignment="Left" VerticalAlignment="Top" GroupName="groupIdoma" />
        </StackPanel>
    </GroupBox>
    <GroupBox Header="Colores" Width="114">
        <StackPanel Margin="10">
            <ColorPicker></ColorPicker>
        </StackPanel>
    </GroupBox>
    <Button Content="Cerrar" />
</StackPanel>
<StackPanel Height="100">
    <ColorPicker></ColorPicker>
</StackPanel>
```

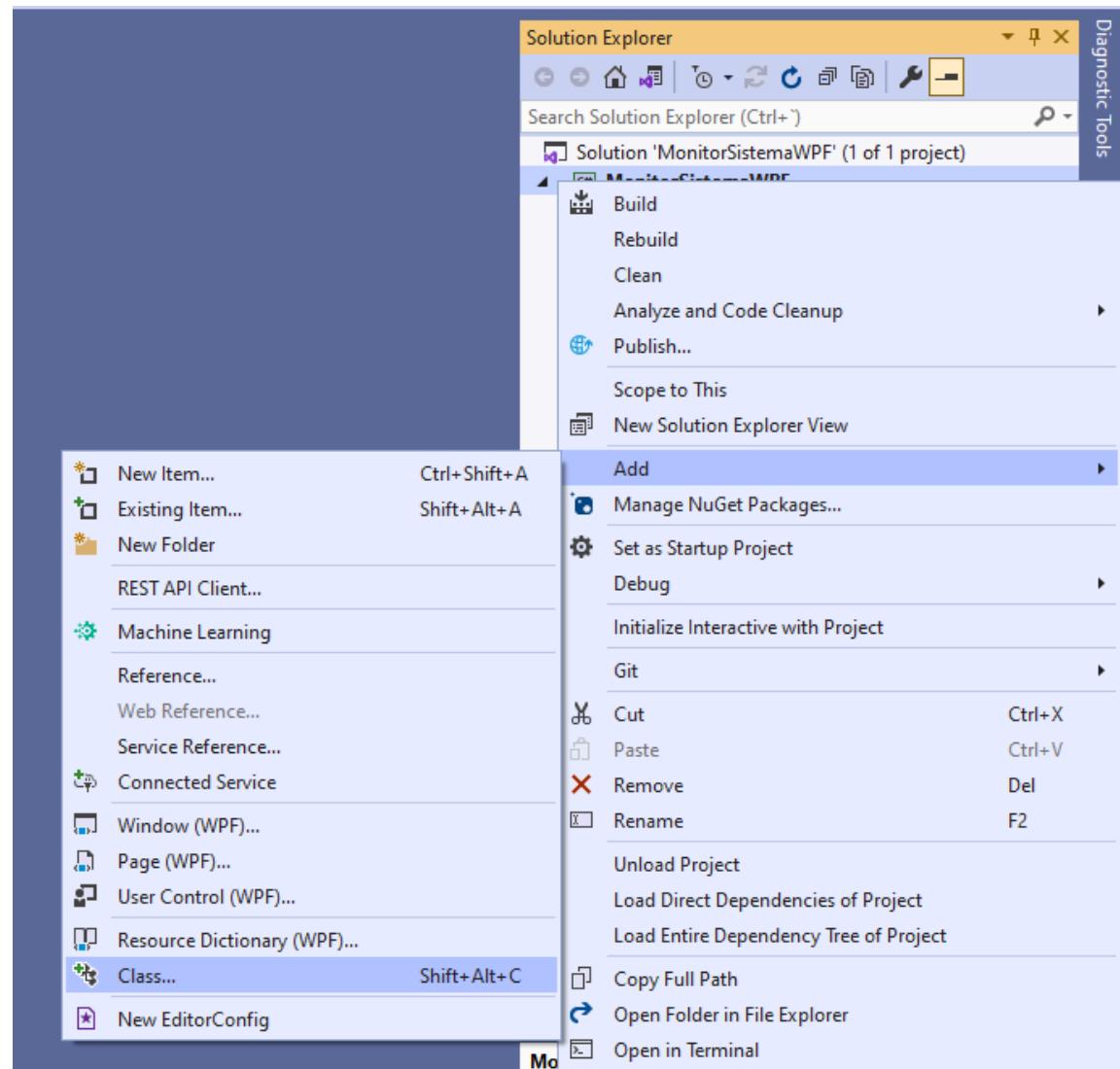
# Diálogo de personalización

- Ponemos la propiedad de color seleccionado al nuevo control



# Clase Config

- Vamos a crear una clase 'Config.cs' para almacenar la configuración de la interfaz:



# Clase Config

- La clase sigue el patrón Singleton porque queremos acceder a ella desde cualquier lugar y es un único objeto siempre

```
namespace MonitorSistemaWPF
{
    public class Config
    {
        public string lang { get; set; }
        public string color { get; set; }
        private static Config _instance;

        private Config() { }

        public static Config GetInstance()
        {
            if (_instance == null)
            {
                _instance = new Config();
            }
            return _instance;
        }
    }
}
```

# Persistencia de la configuración

Usaremos la clase  
System.Configuration.ConfigurationManager

Añadimos los métodos de las siguientes diapositivas a la clase Config.

Los métodos de lectura / escritura deberían estar en otra clase especializada, los dejamos en esta clase por conveniencia para la sesión de clase

[https://docs.microsoft.com/en-us/dotnet/api/  
system.configuration.configurationmanager?  
redirectedfrom=MSDN&view=windowsdesktop-5.0](https://docs.microsoft.com/en-us/dotnet/api/system.configuration.configurationmanager?redirectedfrom=MSDN&view=windowsdesktop-5.0)

# Persistencia de la configuración (lectura)

```
public void InitDefaults()
{
    lang = "es";
    color = "#FF0000";
}

public void Load()
{
    lang = Read("lang");
    color = Read("color");
}

private string Read(string key)
{
    return ConfigurationManager.AppSettings[key];
}
```

# Persistencia de la configuración (escritura)

```
private void Write(Configuration configFile, string key, string value)..  
{  
    var settings = configFile.AppSettings.Settings;  
  
    if (settings[key] == null)  
    {  
        settings.Add(key, value);  
    }  
    else  
    {  
        settings[key].Value = value;  
    }  
}  
  
public void Save()  
{  
    var configFile = ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None);  
    Write(configFile, "lang", lang);  
    Write(configFile, "color", color);  
    configFile.Save(ConfigurationSaveMode.Modified);  
    ConfigurationManager.RefreshSection(configFile.AppSettings.SectionInformation.Name);  
}
```

# Persistencia de la configuración

Comprobamos que funciona haciendo un test unitario

```
using NUnit.Framework;

namespace TestMonitorSistema
{
    public class Tests
    {
        [SetUp]
        public void Setup()
        {
        }

        [Test]
        public void TestConfig()
        {
            // 0. Cargamos la anterior configuración por si tenía algún valor
            MonitorSistemaWPF.Config.GetInstance().Load();
            string originalLang = MonitorSistemaWPF.Config.GetInstance().lang;
            string originalColor = MonitorSistemaWPF.Config.GetInstance().color;

            // 1. Reseteamos la configuración por defecto
            MonitorSistemaWPF.Config.GetInstance().InitDefaults();

            // 2. Comprobamos que la ha guardado
            const string testLang = "_testlang_";
            const string testColor = "_testcolor_";
            // 2.1 Guardamos
            MonitorSistemaWPF.Config.GetInstance().lang = testLang;
            MonitorSistemaWPF.Config.GetInstance().color = testColor;
            MonitorSistemaWPF.Config.GetInstance().Save();
            // 2.2 Cargamos de nuevo los valores por defecto
            MonitorSistemaWPF.Config.GetInstance().InitDefaults();
            // 2.3 Cargamos los valores guardados en 2.1
            MonitorSistemaWPF.Config.GetInstance().Load();
            // 2.4 Comprobamos que lo que se ha cargado es lo habíamos guardado
            Assert.AreEqual(testLang, MonitorSistemaWPF.Config.GetInstance().lang);
            Assert.AreEqual(testLang, MonitorSistemaWPF.Config.GetInstance().color);

            // Guardamos la configuración anterior
            if (originalColor != null || originalLang != null)
            {
                MonitorSistemaWPF.Config.GetInstance().lang = originalLang;
                MonitorSistemaWPF.Config.GetInstance().color = originalColor;
            }
        }
    }
}
```

# Formulario 'WindowConfig'

- Los controles del formulario de configuración los inicializaremos con los datos del objeto singleton 'Config'

```
public partial class WindowConfig : Window
{
    public WindowConfig()
    {
        InitializeComponent();

        Config.GetInstance().Load();
        var lang = Config.GetInstance().lang;
        if (lang != null)
        {
            if (lang.Equals("ES"))
            {
                rbLangES.IsChecked = true;
            }
            else if (lang.Equals("ENG"))
            {
                rbLangENG.IsChecked = true;
            }
        }

        var color = Config.GetInstance().color;
        if (color != null)
        {
            // Color converter está en System.Windows.Media
            cpColor.SelectedColor = (Color?)ColorConverter.ConvertFromString(color);
        }
    }
}
```

# Formulario 'WindowConfig'

- Cuando cerramos los guardamos

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    Close();

    // aplicamos los colores seleccionados a la configuración
    if ((bool)rbLangES.IsChecked)
    {
        Config.GetInstance().lang = "ES";
    }.else if ((bool)rbLangENG.IsChecked)
    {
        Config.GetInstance().lang = "ENG";
    }
    Config.GetInstance().color = cpColor.SelectedColor.Value.ToString();
    Config.GetInstance().Save();
}
```

# Aplicamos los cambios de color

- Cuando cerramos los guardamos. En MainWindow.xaml.cs:

```
private void btnPersonalizar_Click(object sender, RoutedEventArgs e)
{
    var dlg = new WindowConfig();
    dlg.ShowDialog();
    aplicaConfiguracion();
}

<Style x:Key = "etiquetaTituloSeccion" TargetType = "Label">
    <Setter Property = "Background" Value = "#FFB90C0C" />
    <Setter Property = "Foreground" Value = "white" />
    <Setter Property = "FontSize" Value = "18" />
</Style>
```

Recuerda que el estilo lo tenemos en un recurso



Ordenador

Fabricante

ValorFabricante

```
<Label Content="Ordenador" Grid.Row="2" Grid.ColumnSpan="2"
      Style="{StaticResource etiquetaTituloSeccion}"/>
<Label Content="Fabricante" Grid.Row="3"/>
```

El problema que tenemos es que el tipo Setter de la clase System.Windows.Style está “Sealed” (es inmutable). Hasta que sepamos MVVM (próxima sesión) lo cambiaremos “manualmente”

# Aplicamos los cambios de color

- Cuando cerramos los guardamos. En MainWindow.xaml.cs:

```
private void btnPersonalizar_Click(object sender, RoutedEventArgs e)
{
    var dlg = new WindowConfig();
    dlg.ShowDialog();
    aplicaConfiguracion();
}

private void aplicaConfiguracion()
{
    var bgColor = new SolidColorBrush((Color)ColorConverter.ConvertFromString(Config.GetInstance().color));
    labelOrdenador.Background = bgColor;
    labelMemoria.Background = bgColor;
    labelDisco.Background = bgColor;
    labelRed.Background = bgColor;
    /*System.Windows.Style style = (Style)this.FindResource("etiquetaTituloSeccion");
    foreach (var setter in style.Setters)
    {
        // ... - no lo podemos cambiar porque la clase es inmutable
    }*/
}
```

# Aplicamos los cambios de color

- Cuando entramos en la aplicación cargamos los valores guardados

```
public MainWindow()
{
    InitializeComponent();
    Config.GetInstance().Load();
    aplicaConfiguracion();
    dispatcherTimer.Tick += new EventHandler(manejadorDispatcherTimer);
}
```

# Ampliación

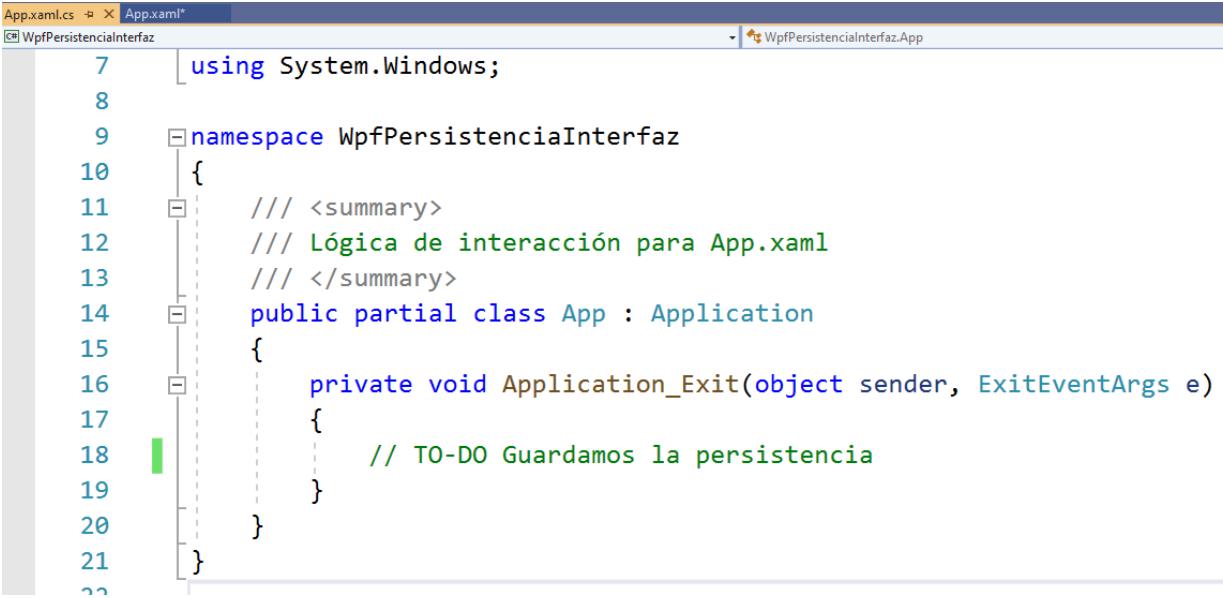
- Añadir botones Aceptar / Cancelar al diálogo de configuración
- Guardar / cargar otra propiedad como la posición de la ventana
- Podemos guardar opcionalmente (preguntando) la configuración capturando el evento de salida
- Para ello en App.xaml añadimos al elemento `<Application` el atributo  
`Exit="Application_Exit">`

(Siguiente diapositiva)

# Captura cierre aplicación

```
<Application x:Class="WpfPersistenciaInterfaz.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:WpfPersistenciaInterfaz"
    StartupUri="MainWindow.xaml"
    Exit="Application_Exit">
<Application.Resources>
```

- ▶ e implementar el manejador del evento



The screenshot shows a Windows-based IDE interface with two tabs at the top: "App.xaml.cs" and "App.xaml". The title bar displays the project name "WpfPersistenciaInterfaz". The code editor contains the following C# code:

```
7  using System.Windows;
8
9  namespace WpfPersistenciaInterfaz
10 {
11     /// <summary>
12     /// Lógica de interacción para App.xaml
13     /// </summary>
14     public partial class App : Application
15     {
16         private void Application_Exit(object sender, ExitEventArgs e)
17         {
18             // TODO Guardamos la persistencia
19         }
20     }
21 }
```

A green vertical bar is visible on the left side of the code editor, indicating a specific line or selection.

# Logging (WPF)

# Registro de acciones

- ▶ Para poder localizar errores en aplicaciones desplegadas (instaladas en ordenadores de clientes) necesitamos poder saber qué ha pasado
  - ▶ No podemos depurar
  - ▶ Es conveniente ir guardando información de rastreo (tracing)
  - ▶ No debemos bloquear el thread principal con la escritura de logs
  - ▶ Podemos usar una librería incorporada en .NET:  
[TraceSource](#)
  - ▶ U otras de terceros como [Log4Net](#)

# Logger: configuración

- ▶ Añadimos esta configuración a **app.config** (copiar / pegar)  
(esto escribirá TODOS los logs — nivel Verbose — en el fichero monitorWPF.log)

```
<configuration>
.... CONTENIDO ANTERIOR .....
<system.diagnostics>
    <trace autoflush="true"/>
    <sources>
        <source name="MonitorWPF" switchName="myswitch" >
            <listeners>
                <add name="textWriterListener" traceOutputOptions="DateTime"
                     type="System.Diagnostics.TextWriterTraceListener"
                     initializeData="monitorWPF.log">
                </add>
                <remove name="Default" />
            </listeners>
        </source>
    </sources>
    <switches>
        <add name="myswitch" value="Verbose" />
    </switches>
</system.diagnostics>
</configuration>
```

(Cuidado al copiar / pegar con las comillas)

# Logger: clase TraceSource

- ▶ Creamos el objeto logger en Config. Para ello debemos incluir el namespace System.Diagnostics

The screenshot shows a portion of a C# code editor. The code is as follows:

```
namespace MonitorSistemaWPF
{
    public class Config
    {
        private static TraceSource logger = new TraceSource("Config");

        public string ...
        public string ...
        private static ...
        public Cont ...
        private ...
        public stat ...
        {
            ...
        }
    }

    public void Load()
    {
        logger.TraceEvent(TraceEventType.Information, 1, "Config: cargando configuración");
        lang = Read("lang");
        color = Read("color");
        logger.TraceEvent(TraceEventType.Information, 1, "Config: color leído: '" + color + "'");
        logger.TraceEvent(TraceEventType.Information, 1, "Config: idioma leído: '" + lang + "'");
    }
}
```

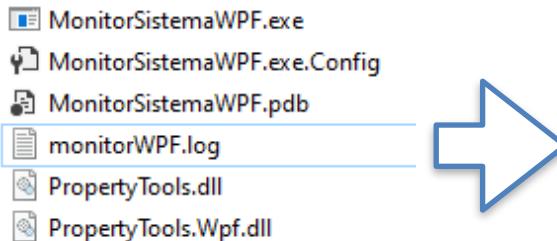
A tooltip is open over the line `using System.Diagnostics;`, showing options to generate a class or assembly reference. A red error bar is under the line `using System.Diagnostics;`, pointing to a CS0246 error message: "The type or namespace name 'TraceSource' could not be found (are you missing a using directive or an assembly reference?)". The code editor interface includes toolbars and a status bar at the bottom.

# Logger: clase TraceSource

- ▶ Llamamos a TraceEvent para cada evento que queramos registrar. Usaremos el tipo / nivel Information, Warning, ... según la severidad del mensaje

```
public void Load()
{
    logger.TraceEvent(TraceEventType.Information, 1, "Config: cargando configuración");
    lang = Read("lang");
    color = Read("color");
    logger.TraceEvent(TraceEventType.Information, 1, "Config: color leído: '" + color + "'");
    logger.TraceEvent(TraceEventType.Information, 1, "Config: idioma leído: '" + lang + "'");
}
```

Esto genera el fichero monitorWPF.log en el directorio del ejecutable (bin/debug)



```
MonitorWPF Information: 1 : Config: cargando configuración
DateTime=2021-10-12T19:43:26.4658808Z
MonitorWPF Information: 1 : Config: color leído: ''
DateTime=2021-10-12T19:43:26.4658808Z
MonitorWPF Information: 1 : Config: idioma leído: ''
DateTime=2021-10-12T19:43:26.4658808Z
```

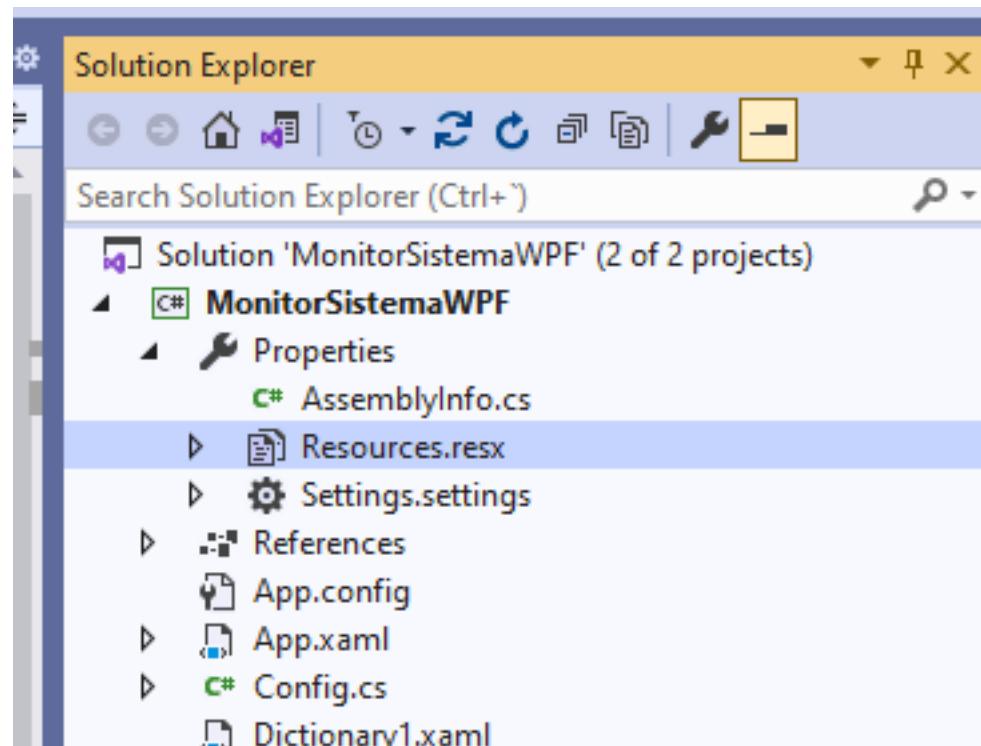
# Actividad

- ▶ Añadir logs a los eventos importantes de la aplicación, incluyendo logs registrando los posibles errores

# **Internacionalización (i18n) y localización (l10n)**

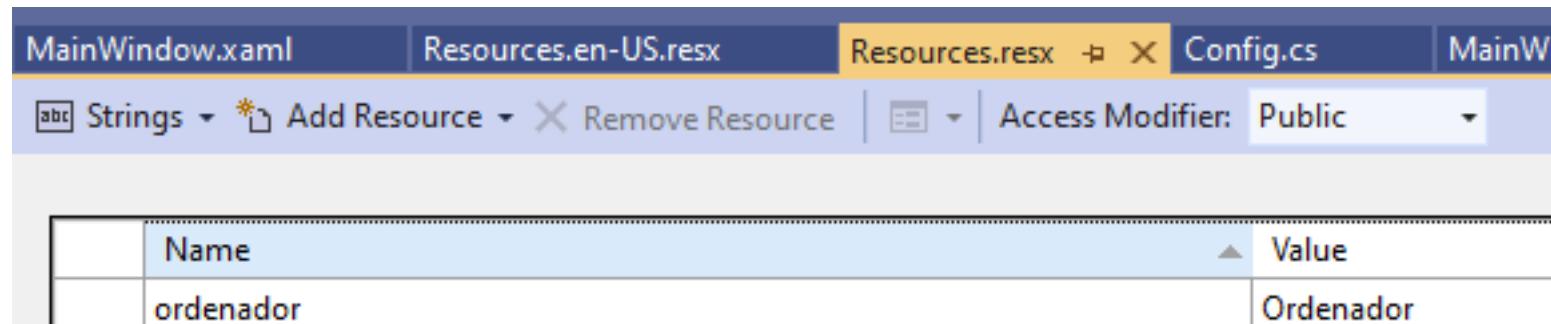
# i18n

- Internacionalizamos el programa añadiendo cadenas “traducibles”
- Lo hacemos usando el fichero Resources.resx



# i18n

El fichero de recursos debe ser **público**



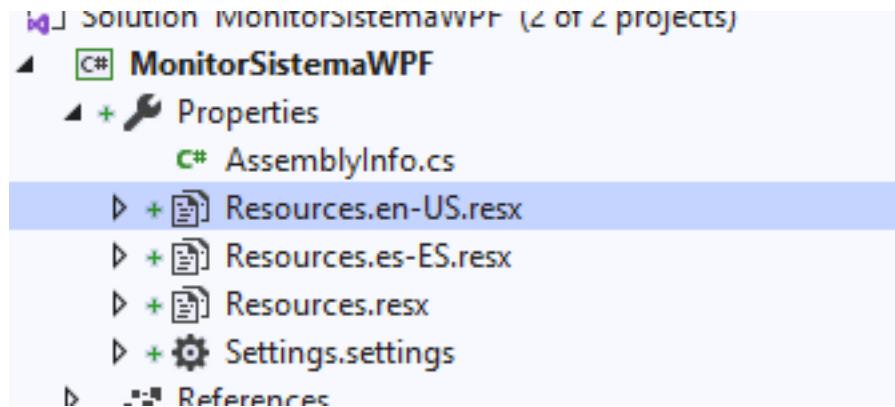
# i18n

- Creamos cadenas para cada elemento a traducir. La columna Name es la clave, conviene no usar acentos, espacios...

Name	Value
ordenador	Ordenador
discos	Discos
red	Red
memoria	Memoria
iniciar	Iniciar
personalizacion	Personalización
String1	

# i18n

- Creamos una copia por cada idioma a traducir, con extensión el código del idioma
- Códigos de idioma en: [https://docs.microsoft.com/en-us/openspecs/windows\\_protocols/ms-lcid/a9eac961-e77d-41a6-90a5-ce1a8b0cdb9c?redirectedfrom=MSDN](https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-lcid/a9eac961-e77d-41a6-90a5-ce1a8b0cdb9c?redirectedfrom=MSDN)



Name	Value
discos	Disks
iniciar	Start
memoria	Memory
ordenador	Computer
personalizacion	Customisation
red	Network

# i18n

- Configuramos WPF para que trabaje con el idioma
- Añadimos al fichero .XAML este atributo (véase que tiene el nombre del proyecto):

```
xmlns:p="clr-namespace:MonitorSistemaWPF.Properties"
```

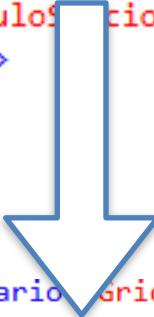
```
]<Window x:Class="MonitorSistemaWPF.MainWindow"
    ...
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expressionblend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:p = "clr-namespace:MonitorSistemaWPF.Properties"
    mc:Ignorable="d"
    Title="Monitor de sistema" Height="600" Width="400">
<Grid>
```

# i18n

- A todos los elementos que queramos traducir les cambiamos el literal en “Content” por “{x:Static p:Resources.CLAVE}”

donde CLAVE es la columna “Name” del .resx

```
<Label x:Name="labelUsuario" Content="ValorUsuario" Grid.Row="1"/>
<Label x:Name="labelOrdenador" Content="Ordenador" Grid.Row="2" Grid.ColumnSpan="2"
|   Style="{StaticResource etiquetaTituloSeccion}" />
<Label Content="Fabricante" Grid.Row="3"/>
```



```
<Label x:Name="labelUsuario" Content="ValorUsuario" Grid.Row="1"/>
<Label x:Name="labelOrdenador" Content="{x:Static p:Resources.ordenador}" Grid.Row="2" Grid.ColumnSpan="2"
|   Style="{StaticResource etiquetaTituloSeccion}" />
|   <Label Content="Fabricante" Grid.Row="3"/>
```

# i18n

- El ordenador donde estamos haciendo las pruebas está en inglés, por lo que coge automáticamente los contenidos del fichero de recursos inglés



# i18n

- Vamos a cambiar el idioma en la configuración
- Primero adaptamos los códigos que tenemos de idioma

```
private Config() { }

public static Config GetInstance()
{
    if (_instance == null)
    {
        _instance = new Config();
    }
    return _instance;
}

/* Estos métodos deberían ir en otra clase */

public void InitDefaults()
{
    lang = "es-ES";
}
```

```
public partial class WindowConfig : Window
{
    const string LANG_ENG = "en_US";
    const string LANG_ES = "es_ES";

    public WindowConfig()
    {
        InitializeComponent();

        Config.GetInstance().Load();
        var lang = Config.GetInstance().lang;
        if (lang != null)
        {
            if (lang.Equals(LANG_ES))
            {
                rbLangES.IsChecked = true;
            }
            else if (lang.Equals(LANG_ENG))
            {
                rbLangENG.IsChecked = true;
            }
        }

        var color = Config.GetInstance().color;
        if (color != null)
        {
            // Color converter está en System.Windows.
            cpColor.SelectedColor = (Color?)ColorConve
        }
    }

    private void Button_Click(object sender, RoutedEventArgs e)
    {
        Close();

        // aplicamos los colores seleccionados a la co
        if ((bool)rbLangES.IsChecked)
        {
            Config.GetInstance().lang = LANG_ES;
        }
        else if ((bool)rbLangENG.IsChecked)
        {
            Config.GetInstance().lang = LANG_ENG;
        }
    }
}
```

# i18n

- De forma sencilla, el idioma sólo se cambia antes de inicializarse los componentes que es cuando .net lee los resources.resx....
- Organizamos el código para que se cargue el idioma antes de inicializarse los componentes
- Cuando cambiemos la configuración deberemos reiniciar la aplicación para que se cambie el idioma

```
private void btnPersonalizar_Click(object sender, RoutedEventArgs e)
{
    var dlg = new WindowConfig();
    dlg.ShowDialog();
    cambiaColores();
    cambiaIdioma();
}

private void cambiaColores()
{
    var color = Config.GetInstance().color;
    if (color != null)
    {
        var bgColor = new SolidColorBrush((Color)ColorConverter.ConvertFromString(Config.GetInstance().color));
        labelOrdenador.Background = bgColor;
        labelMemoria.Background = bgColor;
        labelDisco.Background = bgColor;
        labelRed.Background = bgColor;
    }
}

private void cambiaIdioma()
{
    var lang = Config.GetInstance().lang;
    if (lang != null)
    {
        // Set the current user interface culture to the specific culture Russian
        System.Threading.Thread.CurrentThread.CurrentCulture =
            new System.Globalization.CultureInfo(lang);
    }
}
```

```
public MainWindow()
{
    Config.GetInstance().Load();
    cambiaIdioma();
    InitializeComponent();
    cambiaColores();
}
```

# i18n

- Actividad: que el idioma guardado en la configuración se cargue al iniciar la aplicación. Traducir toda la aplicación
- Actividad opcional: traducir las cadenas emitidas desde C#

# **Contenido obsoleto (WinForms)**

Lo mantenemos aquí como documentación opcional,  
**Úsese la versión WPF**

# Logging - obsoleto (WinForms)

# Registro de acciones

- ▶ Para poder localizar errores en aplicaciones desplegadas (instaladas en ordenadores de clientes) necesitamos poder saber qué ha pasado
  - ▶ No podemos depurar
  - ▶ Es conveniente ir guardando información de rastreo (tracing)
  - ▶ No debemos bloquear el thread principal con la escritura de logs
  - ▶ Podemos usar una librería incorporada en .NET:  
[TraceSource](#)
  - ▶ U otras de terceros como [Log4Net](#)

# Logger: configuración

- ▶ Para que se escriba en un fichero .log configuramos el logger en la aplicación (lo incrustamos en App.config — que ya tiene información sobre personalización)

```
<system.diagnostics>
  <trace autoflush="true" indentsize="4">
    <listeners>
      <add name="Calculadora" type="System.Diagnostics.TextWriterTraceListener"
initializeData="calculadora.log" />
      <remove name="Default" />
    </listeners>
  </trace>
</system.diagnostics>
```

# Logger: uso

- ▶ Vamos a registrar acciones en la calculadora
  - ▶ Cada vez que se realiza una operación la registramos
  - ▶ Por ejemplo, indicamos qué hemos leído de la configuración de personalización:
    - ▶ Usamos Trace.TraceInformation("....")
    - ▶ Si queremos que se escriba inmediatamente en el fichero usamos Flush.

```
static void Main()
{
    Trace.TraceInformation("Aplicación iniciada.");
    Trace.Flush();
    config = new Taller1.Config();

    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    recuperaConfig();
    Trace.TraceInformation("Configuración de personalización cargada: " + config.ToString());

    Application.Run(new Principal());
}
```

# Logger: uso

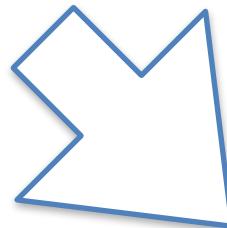
- ▶ Existen diversos niveles de log: error, warning, information

```
private void BtnCalcular_Click(object sender, EventArgs e)
{
    operando2 = Convert.ToInt32(labelDisplay.Text);
    Trace.TraceInformation("Operación " + operando1 + operacion.ToString() + operando2);
    switch (operacion) {
        case Operaciones.Suma:
            resultado = operando1 + operando2;
            break;
        case Operaciones.Resta:
            resultado = operando1 - operando2;
            break;
        case Operaciones.Division:
            if (operando2 == 0)
            {
                // el número 2 es un identificador del evento
                Trace.TraceWarning("Intento de división por cero");
            } else
    }
```

# Logger: resultado

Nombre

- ca
- en
- Calculadora.log
- personalizacion
- Taller1.exe
- Taller1.exe.config
- Taller1.pdb



\*Calculadora.log: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
$Taller1.exe Information: 0 : Aplicación inciada.  
Taller1.exe Information: 0 : Configuración de personalización cargada: Ta  
Taller1.exe Information: 0 : Operación 5Producto6  
Taller1.exe Information: 0 : Operación 0Division0  
Taller1.exe Warning: 0 : Intento de división por cero
```

# Timestamp

- ▶ Si queremos que se añada la fecha y hora de cada log añadimos esto a la configuración:
  - ▶ traceOutputOptions="DateTime, Timestamp"

```
<system.diagnostics>
  <trace autoflush="true" indentsize="4">
    <listeners>
      <add name="Calculadora"
          type="System.Diagnostics.TextWriterTraceListener"
          initializeData="Calculadora.log"
          traceOutputOptions="Timestamp, DateTime"/>
        <!-- Other listeners -->
    </listeners>
  </trace>
</system.diagnostics>
```

# Timestamp

- ▶ Si queremos que se añada la fecha y hora de cada log añadimos esto a la configuración:
  - ▶ traceOutputOptions="DateTime, Timestamp"

 Calculadora.log: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
Taller1.exe Information: 0 : Aplicación iniciada.
    DateTime=2020-10-08T15:19:55.3948514Z
    Timestamp=228390922193
Taller1.exe Information: 0 : Configuración de personalización cargada: Taller1.Conf
    DateTime=2020-10-08T15:19:55.4415395Z
    Timestamp=228391361559
Taller1.exe Information: 0 : Operación 0Suma26
    DateTime=2020-10-08T15:19:59.4414667Z
    Timestamp=228431280819
Taller1.exe Information: 0 : Operación 26Division0
    DateTime=2020-10-08T15:20:01.8481156Z
    Timestamp=228455340284
Taller1.exe Warning: 0 : Intento de división por cero
    DateTime=2020-10-08T15:20:01.8781951Z
    Timestamp=228455605753
```

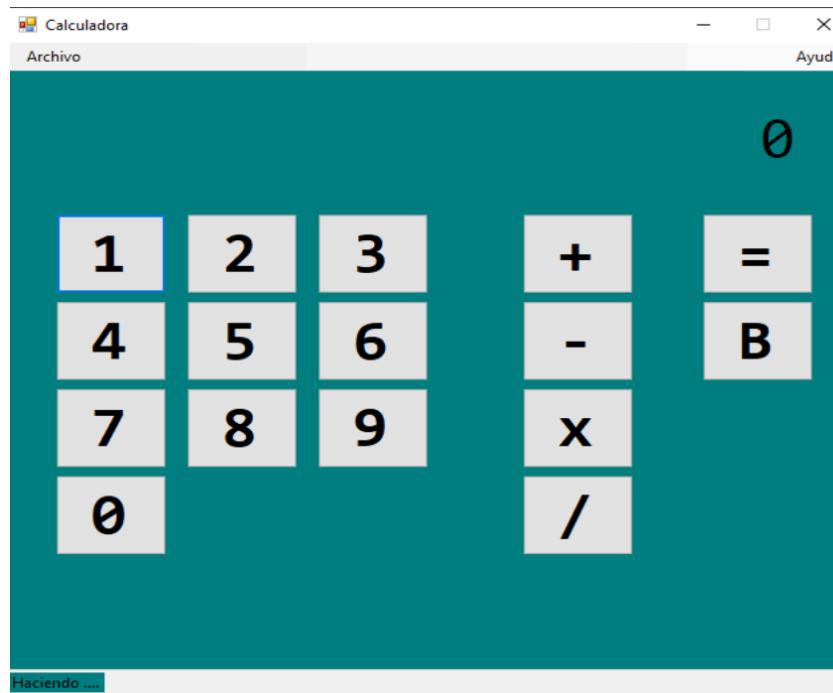
# Personalización (WinForms)

# Estrategia

- ▶ Vamos a crear un objeto estático llamado Config (podría tener cualquier otro nombre) donde guardaremos las propiedades personalizadas
  - ▶ Para personalizar un formulario modificará sus valores
  - ▶ Para hacerlas persistentes las guardaremos en un fichero
  - ▶ Para cargarlas de nuevo las leeremos de ese fichero

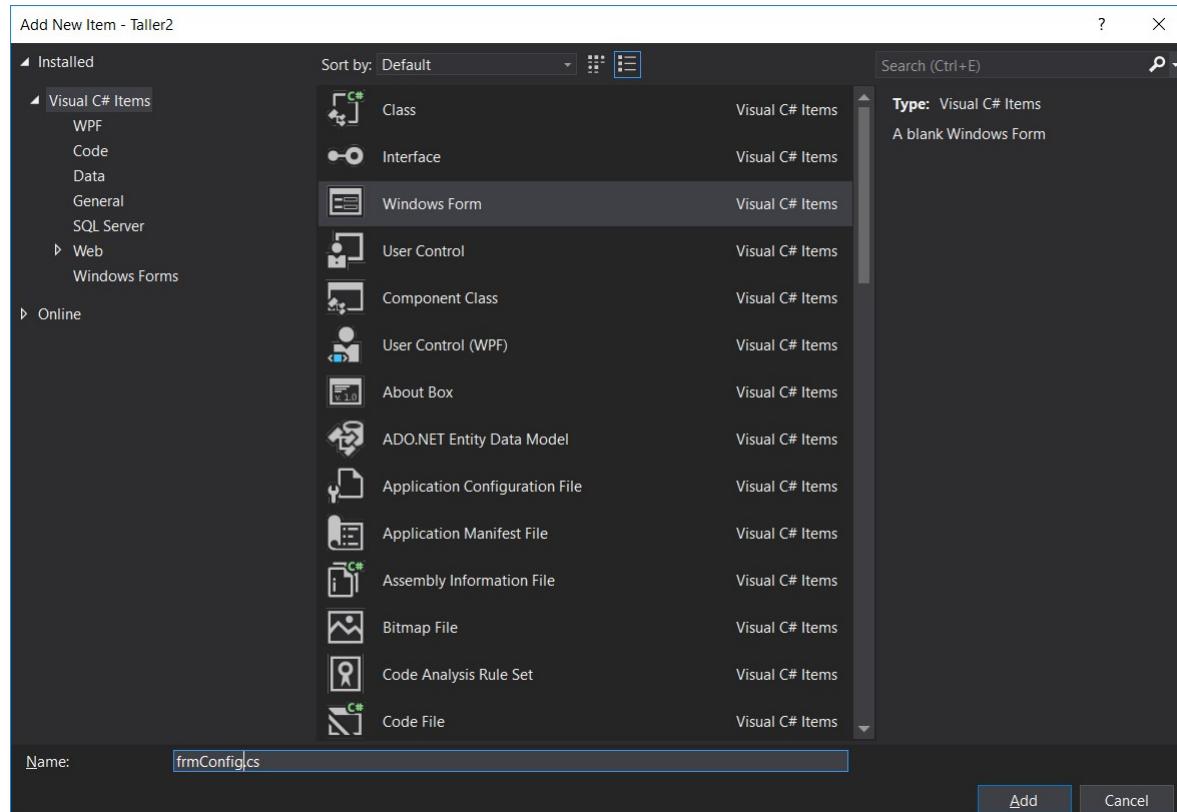
# Aplicación de ejemplo

- Partimos de la aplicación WinForms de la calculadora realizada en la primera sesión (disponible en Moodle)



# Persistencia del estado del interfaz

- Crearemos un nuevo formulario, más pequeño que el formulario principal.
  - Lo llamaremos FrmConfig. Cambiaremos 'text' a 'Configuración'
  - FormBorderStyle: FixedSingle. MaximizeBox y MinimizeBox a False.



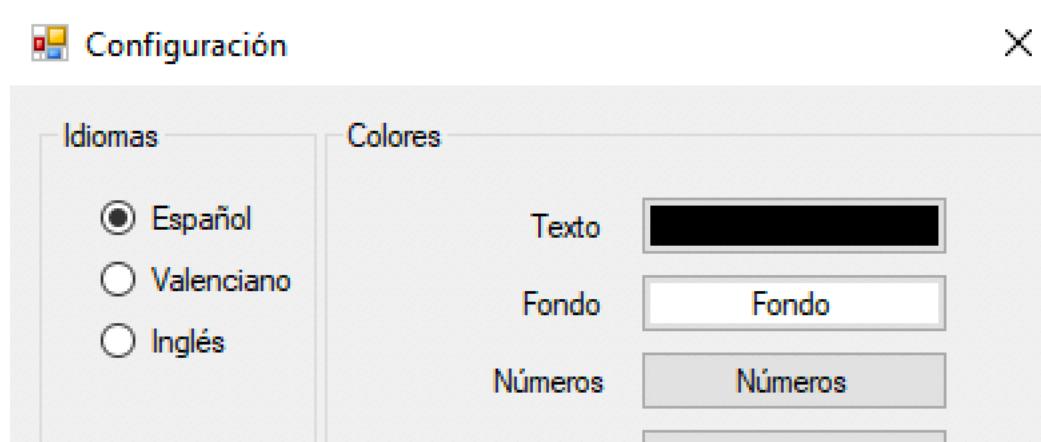
# Formulario 'frmConfig'

- Añadimos una nueva opción en el menu 'Archivo' llamada 'Configuración' (menuConfig).
- Añadiremos un separador entre esta nueva opción y la de 'Salir'.
- El evento de esta opción abrirá el formulario de configuración:
  - ShowDialog abre el formulario de forma 'modal'.

```
private void menuConfiguracion_Click(object sender,  
EventArgs e) {  
    FrmConfig config = new  
    FrmConfig();  
    config.ShowDialog(this);  
}
```

# Formulario 'frmConfig'

- El formulario contendrá:



# Formulario 'frmConfig'

- Le añadiremos dos botones abajo a la derecha: 'Guardar' y 'Cancelar'.
  - 'Guardar': nombre: btnGuardar, Etiqueta: 'Guardar'. Color: verde.
  - 'Cancelar': nombre: btnCancelar, Etiqueta: 'Cancelar'. Color: rojo.
    - Este botón tendrá el siguiente manejador al recibir el click:

```
private void btnCancelar_Click(object  
sender, EventArgs e) {  
  
    this.Hide();  
  
}
```

# **Formulario 'frmConfig'**

- Contenido del formulario de configuración:
  - Vamos a configurar algunos de los colores de la calculadora: botones numéricos, botones de operaciones, color de fondo y color del texto.
  - Además podremos configurar el idioma del aplicativo pudiendo optar por tres opciones: castellano, valenciano e inglés (u otros tres).

# **Formulario 'frmConfig'**

- Crearemos dos paneles o dos group boxes:
  - pnlIdiomas y pnlColores.
  - Dentro del panel 'pnlIdiomas' ubicaremos
    - un label 'lblIdiomas' con el texto 'Seleccione idioma del interfaz:' (si usamos un group box no será necesario)
    - Tres botones de radio llamados: radioES, radioCA y radioEN.

# **Formulario 'frmConfig'**

- Dentro del panel 'pnlColores' ubicaremos
  - un label 'lblColores' con el texto 'Seleccione la combinación de colores del interfaz:'
  - Cuatro etiquetas (opcional) y cuatro botones para seleccionar los colores parametrizables.
    - Color de texto de la calculadora: lblcolorTexto y btnColorTexto
    - Color del fondo de la calculadora: lblColorFondo y btnColorFondo
    - Color de los botones numéricos: lblColorBotonesNum y btnColorBotonesNum
    - Color de los botones de operaciones: lblColorBotonesOp y btnColorBotonesOp

# Clase Config

- Vamos a crear una clase 'Config.cs' para almacenar la configuración de la interfaz:

```
using System.Drawing;
```

*Color está en System.Drawing*

```
...
```

```
class Config
```

```
{
```

```
    public string lang { get; set; }
    public Color texto { get; set; }
    public Color fondo { get; set; }
    public Color botonesNUM { get; set; }
    public Color botonesOP { get; set; }
```

```
...
```

# Clase Config

- Continuación de la clase:

```
...
public Config() {
    lang="es";
    texto = SystemColors.ControlText; fondo
    = SystemColors.ControlLightLight;
    botonesNUM =
    SystemColors.ControlLight; botonesOP
    = SystemColors.ControlLight;
}
```

# Clase Config

- Ahora, incluiremos un objeto estático de clase 'Config' en nuestro programa, en la clase 'Program':

```
public static Taller3.Config config;  
static void Main() {  
    config = new Taller3.Config();  
    ...  
}
```

(Cambia Taller3 por el nombre del namespace donde tengas Config)

# Formulario 'frmConfig'

- Los controles del formulario de configuración los inicializaremos con los datos del objeto 'config' de la clase 'Program'.

```
public frmConfig() {  
    InitializeComponent();  
    btnColorTexto.BackColor = Program.config.texto;  
    btnColorFondo.BackColor = Program.config.fondo;  
    btnColorBotonesNum.BackColor = Program.config.botonesNUM;  
    btnColorBotonesOp.BackColor = Program.config.botonesOP;  
    switch (Program.config.lang) {  
        case "es":  
            radioES.Checked = true; break;  
        case "ca":  
            radioCA.Checked = true; break;  
        case "en":  
            radioEN.Checked = true; break;  
        default:  
            radioES.Checked = true; break;  
    }  
}
```

# Formulario 'frmConfig'

- Incluimos un ColorDialog en el formulario frmConfig y lo llamamos colorPicker.
- Los botones mostrarán un diálogo de selección de color al recibir un click. Usaremos una función común para ello.

```
private void btnColorTexto_Click(object sender, EventArgs e) {  
    Color c = this.selectColor(sender);  
    // y más cosas...  
}
```

# Formulario 'frmConfig'

- La función común:

```
private Color selectColor(object sender) {  
    Button yo = (Button)sender;  
    colorPicker.Color = Color.Black;  
    colorPicker.ShowDialog();  
    yo.BackColor = colorPicker.Color;  
    return colorPicker.Color;  
}
```

# **Formulario 'frmConfig'**

- Además, los manejadores de cada uno de los botones cambiarán el color de los controles afectados.
  - En cada cambio, actualizaremos el objeto 'Program.config'
- El botón para el color de texto:

```
private void btnColorTexto_Click(object sender, EventArgs e)
{
    Color c = this.selectColor(sender);
    this.Owner.ForeColor = c;
    Program.config.texto = c;
}
```

# **Formulario 'frmConfig'**

- El botón para el color de fondo:

```
private void btnColorFondo_Click(object sender, EventArgs e)
{
    Color col = this.selectColor(sender);
    foreach (Control c in this.Owner.Controls) {
        if (c.Name.StartsWith("pnl")) {
            // el panel de fondo es pnIDisplay
            c.BackColor = col;
        }
    }
    Program.config.fondo = col;
}
```

# Formulario 'frmConfig'

- El botón para el color de los números:

```
Color col = this.selectColor(sender);

Control [] panelNumeros = this.Owner.Controls.Find("pnlNumeros", true);
if (panelNumeros == null || panelNumeros.Length != 1)
{
    throw new Exception("No puedo encontrar un panel 'pNLNumeros'");
} else
{
    foreach (Control c2 in panelNumeros[0].Controls)
    {
        if (c2.Name == "btn1" || c2.Name == "btn2" ||
            c2.Name == "btn3" || c2.Name == "btn4" ||
            c2.Name == "btn5" || c2.Name == "btn6" ||
            c2.Name == "btn7" || c2.Name == "btn8" ||
            c2.Name == "btn9" || c2.Name == "btn0")
        {
            c2.BackColor = col;
        }
    }
}

Program.config.botonesNUM = col;
```

Comprobar que el panel se llama “pnlNumeros”  
¿Hace falta comprobar el nombre de los botones?

# Formulario 'frmConfig'

- El botón para el color de los números

*Para no tener que hacer tantas comparaciones podemos usar Tag y filtrar por ese valor ...*

Margin	3; 3; 3; 3
MaximumSize	0; 0
MinimumSize	0; 0
Modifiers	Private
Padding	0; 0; 0; 0
RightToLeft	No
Size	75; 74
TabStop	True
Tag	numeros
Text	

Bucle botones: btn

```
if (btn.Tag.Equals("numeros"))
```

# Formulario 'frmConfig'

- El botón para el color de las operaciones:

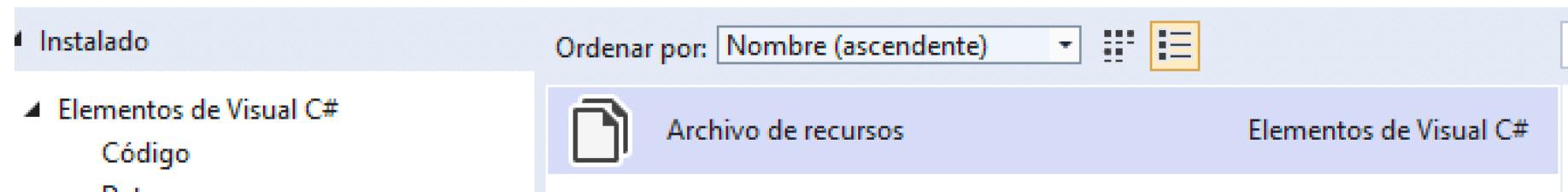
```
Color col = this.selectColor(sender);
Control[] panelOperadores = this.Owner.Controls.Find("pnlOperadores", true);
if (panelOperadores == null || panelOperadores.Length != 1)
{
    throw new Exception("No puedo encontrar un panel 'pnlOperadores'");
}
else
{
    foreach (Control c2 in panelOperadores[0].Controls)
    {
        c2.BackColor = col;
    }
}
Program.config.botonesOP = col;
```

# **Internacionalización (i18n) y localización (l10n) (WinForms) - contenido obsoleto**

# i18n

- Para el cambio de idioma, primero tenemos que 'internacionalizar' el programa. En una primera versión, sólo traduciremos las etiquetas de la barra de menú.
- Crearemos un nuevo **Archivo de recursos** 'Resource.resx' añadiremos variables y valores para los textos del programa:

Agregar nuevo elemento - Taller1



Es importante la R mayúscula  
(después veremos por qué)

# i18n

The screenshot shows a Windows application window titled "Cadenas" (Strings). The menu bar includes "Agregar recurso" (Add resource) and "Quitar recurso" (Remove resource). The toolbar has a "Modificador de acceso" (Access modifier) set to "Public". A table lists the following resources:

Nombre	Valor
mnArchivo	Archivo
mnAyuda	Ayuda
mnArchivoConfiguracion	Archivo de configuración
mnArchivoSalir	Salir
mnAyudaAcercade	Acerca de...

El fichero de recursos debe ser  
**público**

Crearemos dos ficheros más:  
**Resource.en.resx** y  
**Resource.ca.resx** para los  
otros dos idiomas del  
programa.

The screenshot shows a Windows application window with a table of resources. The table includes the following entries:

Nombre	Valor
mnArchivo	File
mnAyuda	Help
mnArchivoConfiguracion	Settings
mnArchivoSalir	Exit
mnArchivoAcercaDe	About
String1	

# i18n

menuArchivo, etc...  
son los objetos  
ToolStripMenuItem

- En el formulario principal, 'cargaremos' los textos del recurso usando una función:

```
public void i18n() {  
    menuArchivo.Text = Resource.mnArchivo;  
    menuAyuda.Text = Resource.mnAyuda;  
    menuConfig.Text = Resource.mnArchivoConfiguracion;  
    menuSalir.Text = Resource.mnArchivoSalir;  
    menuAcercaDe.Text = Resource.mnAyudaAcercade;  
  
    ...  
}
```

Véase cómo se carga **Resource**,  
no **resource** (importan las mayúsculas)

- ▶ C# Program.cs
- ▶  Resource.ca.resx
- ▶  Resource.en.resx
- ▶  Resource.resx

# i18n

- En el constructor del formulario principal, estableceremos el idioma del interfaz e invocaremos a la función anterior:

```
private string lang = Program.config.lang;  
...  
public Principal() {  
    ...  
    InitializeComponent();  
    // una vez cargados los componentes...  
    Thread.CurrentThread.CurrentCulture =  
    new CultureInfo(lang);  
    i18n();  
}
```

# i18n

- La función que 'localiza' nuestro interfaz, invocará a la función anterior 'i18n':

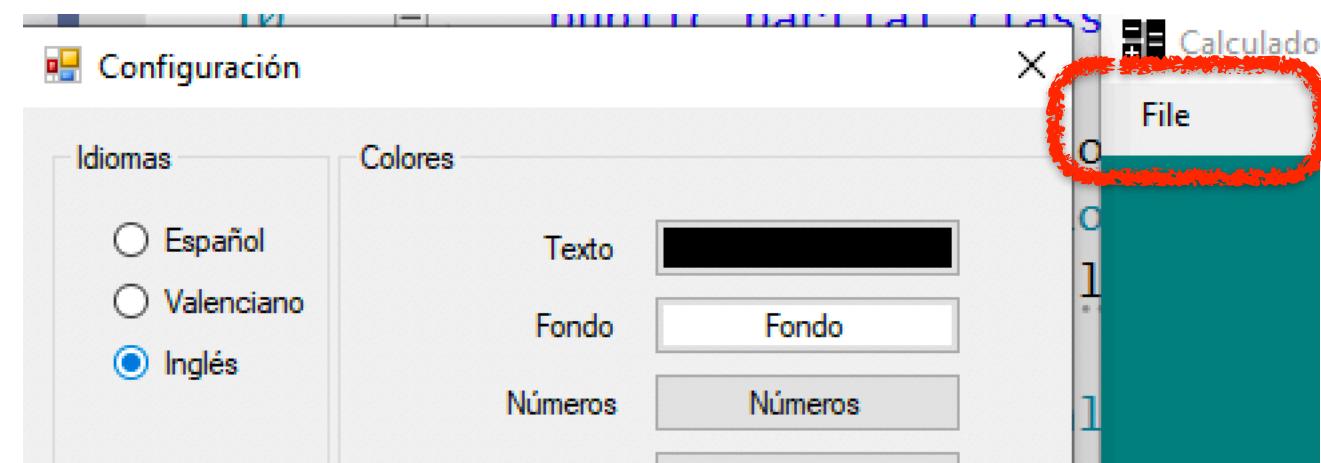
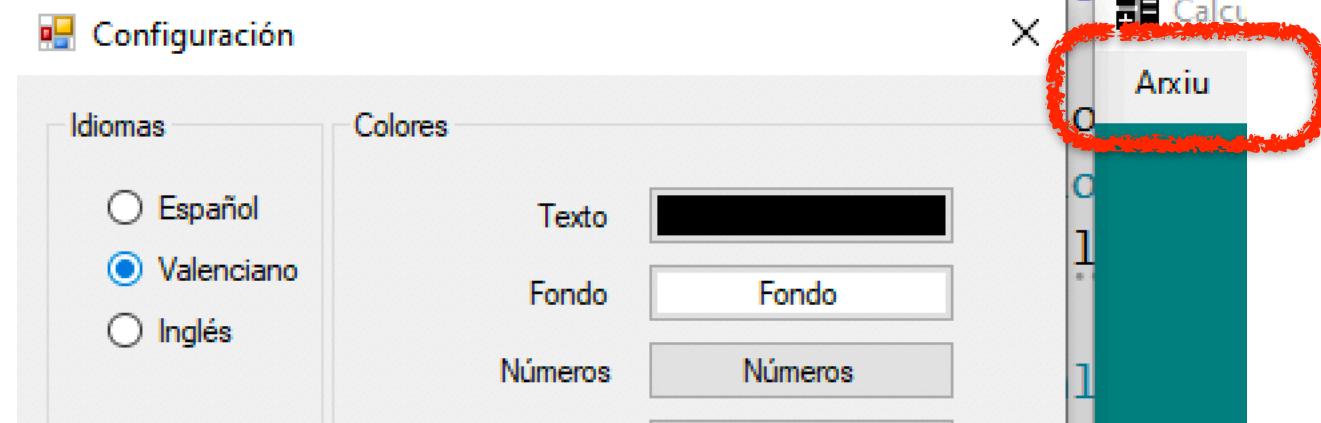
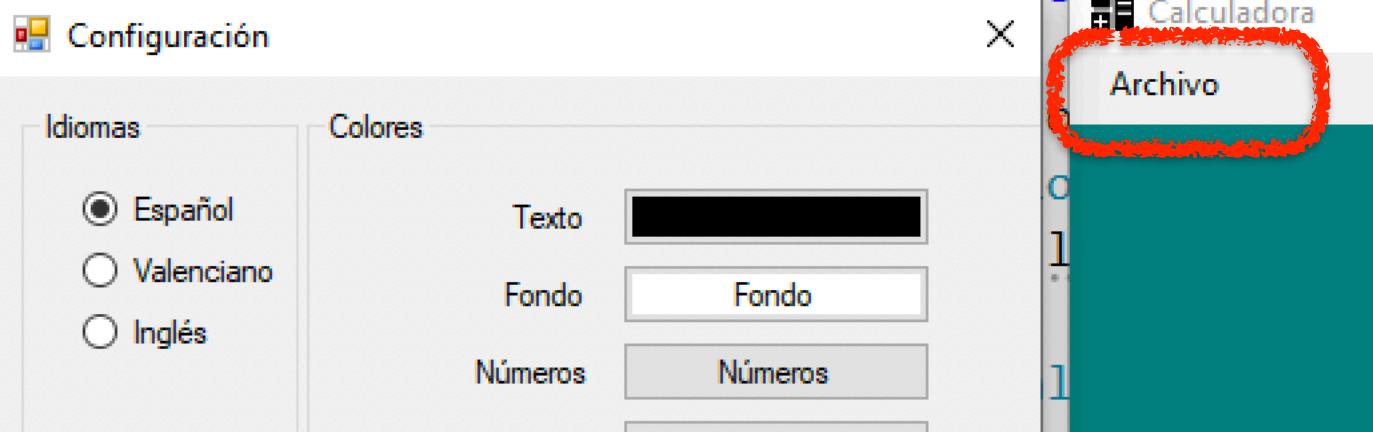
```
private void I10n(string lang) {  
    Program.config.lang = lang;  
  
    Thread.CurrentThread.CurrentCulture =  
    new CultureInfo(lang);  
  
    Principal p = this.Owner as Principal;  
  
    if (p != null)p.i18n();  
}
```

Esto lo hacemos en FormConfig.cs

# i18n

- Ahora vamos a implementar los manejadores que se invoquen al cambiar los botones de radio del selector de idioma. Los tres llamaran a una misma función:

```
private void radioES_CheckedChanged(object sender,  
EventArgs e) {  
    l10n("es");  
}  
...  
private void radioEN_CheckedChanged(object sender,  
EventArgs e) {  
    l10n("en");  
}
```



# **Persistencia del estado del interfaz (WinForms) - contenido obsoleto**

# **Guardar configuración**

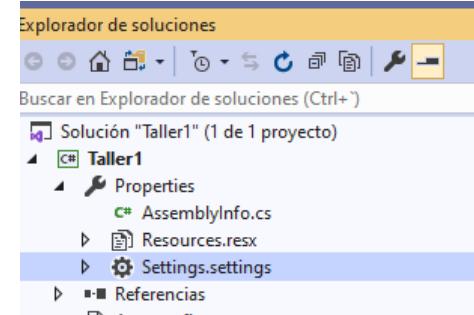
- Ahora vamos a implementar la operación de 'guardar' la configuración del interfaz.
  - Abriremos o crearemos el **Archivo de configuración de aplicaciones:** 'calculadora.config'
  - Volcaremos en él los datos 'serializados' del objeto Program.config.
  - Cerraremos el fichero.

# Guardar configuración

- El nombre del fichero lo vamos a guardar como un dato de configuración del programa, en el fichero 'Properties/Settings.settings'.

La configuración de la aplicación permite almacenar y recuperar de forma dinámica la configuración de p de color de un usuario y recuperarlas la próxima vez que se ejecute. Aprender más acerca de la configurac

	Nombre	Tipo	Ámbito	Valor
	fichero_config	string	Usuario	personalizacion
*	Valor	string	Usuario	



# Guardar configuración

- Manejador del evento click del botón de guardar:

```
string fichero =  
Properties.Settings.Default.fichero_config.ToString();  
StreamWriter fc = new StreamWriter(fichero, false);  
fc.WriteLine(Program.config.lang);  
fc.WriteLine(Program.config.texto.Name);  
fc.WriteLine(Program.config.fondo.Name);  
fc.WriteLine(Program.config.botonesNUM.Name);  
fc.WriteLine(Program.config.botonesOP.Name);  
fc.Close();  
MessageBox.Show("Configuración almacenada correctamente",  
"Aviso", MessageBoxButtons.OK, MessageBoxIcon.Information);  
this.Hide();
```

# Cargar configuración

- Lo siguiente: modificar el programa para que recupere la configuración del interfaz del fichero 'calculadora.config', si existe. En Program.cs:

```
private static void recuperaConfig() {  
    string fichero =  
        Properties.Settings.Default.fichero_config.ToString();  
    if (File.Exists(fichero)) {  
        StreamReader fc = new StreamReader(fichero);  
        Program.config.lang = fc.ReadLine();  
        Program.config.texto = Color.FromName(fc.ReadLine());  
        Program.config.fondo = Color.FromName(fc.ReadLine());  
        Program.config.botonesNUM = Color.FromName(fc.ReadLine());  
        Program.config.botonesOP = Color.FromName(fc.ReadLine());  
        fc.Close(); }  
}
```

# Cargar configuración

- En el main invocamos recuperarConfig antes de crear el formulario principal:

```
static void Main()
{
    config = new Taller1.Config();

    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    recuperarConfig();
    Application.Run(new Principal());
}
```

# Cargar configuración

- Finalmente, al cargar el **formulario principal**, lo configuraremos con los colores y datos del objeto 'Config':

```
public void colores() {  
    this.ForeColor = Program.config.texto;  
  
    pnlAcciones.BackColor = Program.config.fondo;  
  
    pnlNumeros.BackColor = Program.config.fondo;  
  
    pnlOperaciones.BackColor = Program.config.fondo;  
  
    btn0.BackColor = Program.config.botonesNUM;  
  
    ...  
}
```

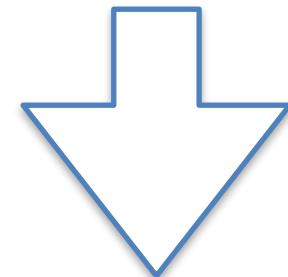
```
public Principal()  
{  
    InitializeComponent();  
    Thread.CurrentThread.CurrentCulture = new CultureInfo(lang);  
    i18n();  
    colores();  
}
```

# Ampliación: cargar configuración

```
public Principal()
{
    InitializeComponent();
    Thread.CurrentThread.CurrentCulture = new CultureInfo(lang);

    i18n(); // importante después de InitializeComponent
    personalizacion();
    operando1 = 0;
    operando2 = 0;
    resultado = 0;
}

private void personalizacion()
{
    // TODO Refactorizar código para que se cargue aquí toda la personalización
    this.pnlDisplay.BackColor = Program.config.fondo;
}
```



# Otros ejercicios de ampliación

- Sencillos:
  - Acabar de 'localizar' el resto de los literales del programa y aplicar la personalización al interfaz
  - Añadir un parámetro de configuración nuevo, por ejemplo, el color del display.
  - Añadir un parámetro nuevo que no sea un color: tipografía, o tamaño de fuente.
    - Añadirlo a la clase config.
    - Añadirlo al fichero 'calculadora.config'.
    - Recuperarlo al cargar.

# Ejercicios de ampliación

- Sencillos:
  - Implementar un mecanismo que permita volver a la configuración anterior en caso de hacer click en el botón de cancelar.
  - Permitir modificar el tamaño del formulario y hacerlo persistente

# Ejercicios de ampliación

- No tan sencillos:
  - Implementar un mecanismo que permita volver a la configuración anterior en caso de hacer click en el botón de cancelar.