GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

DESARROLLO DE INTERFACES
TEMA 4: ENLACE A DATOS

Departamento de Informática
Luis José Fortich Giner

En WPF (Windows Presentation Foundation), las Dependency Properties son un concepto clave que permite la creación de propiedades con funcionalidades avanzadas, como enlace de datos, animación y estilo. A diferencia de las propiedades regulares en las clases de .NET, las Dependency Properties tienen características específicas que las hacen adecuadas para la construcción de interfaces de usuario flexibles.

¿Qué son las Dependency Properties?

Las Dependency Properties son propiedades especiales en WPF que extienden las propiedades comunes de .NET Framework. La característica clave es que estas propiedades pueden establecerse, obtenerse y heredarse a través de elementos visuales en la jerarquía del árbol visual, lo que facilita la creación de interfaces de usuario dinámicas y flexibles.

Son propiedades que dependen del sistema de propiedades de WPF para su completo funcionamento. ¿Qué es el sistema de propiedades de WPF? Son un conjunto de servicios que se pueden utilizar para ampliar la funcionalidad de una propiedad.

Ejemplo: Control en WPF tiene propiedades que depende (necesita) del sistema de propiedades de WPF.

¿Por qué utilizar este sistema de dependencia? Para poder establecer las propiedades de un control en función de otros parámetros que pueden cambiar. Los parámetros que pueden cambiar: propiedades del sistema (temas y preferencias de usuario), data binding (Just in Time, toma valores distintos en tiempo de ejecución), animaciones, estilos, etc.

Podemos ver como un botón, si vamos a su definición en VSC, tiene ciertas DependencyProperties asignadas;

```
public class Button : ButtonBase
{
    public static readonly DependencyProperty IsDefaultProperty;
    public static readonly DependencyProperty IsCancelProperty;
    public static readonly DependencyProperty IsDefaultedProperty;

    public Button();

    public bool IsDefault { get; set; }
    public bool IsCancel { get; set; }
    public bool IsDefaulted { get; }

    protected override void OnClick();
    protected override AutomationPeer OnCreateAutomationPeer();
}
```

Todos los controles WPF tienen dependency properties. Como vemos en la imagen anterior, las propiedades IsDefault, IsCancel y IsDefaulted tiran de las DependencyProperty IsDefaultProperty, IsCancelProperty, IsDefaultProperty.

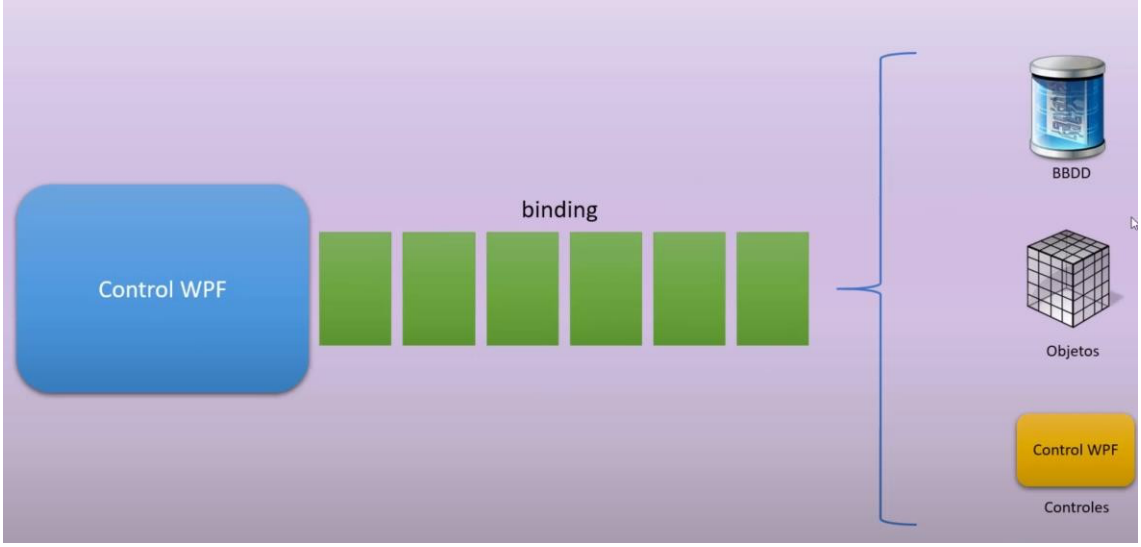
En el siguiente enlace tenéis un ejemplo de como crear una propiedad de dependencia:

https://learn.microsoft.com/es-es/dotnet/desktop/wpf/properties/how-to-implement-a-dependency-property?view=netdesktop-8.0

Data Binding

El Data Binding es una de las características clave de WPF (Windows Presentation Foundation) que permite establecer una conexión entre los datos y la interfaz de usuario de una manera eficiente y flexible.

En esencia, el Data Binding permite enlazar o asociar un elemento de la interfaz de usuario (como un control de texto, una etiqueta, etc.) a una propiedad o valor de un objeto de datos, como una clase en C#.



¿Cómo Funciona el Data Binding?

1. **Definición de Orígenes de Datos:**
   - Se define un origen de datos, que es una instancia de una clase o una fuente de datos como una colección (como una lista o un conjunto de datos).
2. **Establecimiento de la Relación de Vinculación:**
   - Se especifica el origen de datos y la propiedad que se desea enlazar en el elemento de la interfaz de usuario. Esto se hace generalmente en el archivo XAML.
3. **Actualización Automática:**
   - Cuando el valor del origen de datos cambia, la interfaz de usuario se actualiza automáticamente para reflejar ese cambio.

Ventajas del Data Binding:

1. **Desacoplamiento entre Vista y Modelo de Datos:**
   - Permite mantener separados los datos y la presentación. Esto facilita la modificación y mantenimiento del código.
2. **Actualizaciones Automáticas:**
   - Cuando los datos cambian, la interfaz de usuario se actualiza automáticamente, lo que reduce la necesidad de código adicional para mantener la coherencia.
3. **Facilita la Manipulación de Datos:**

- Simplifica la manipulación y gestión de datos en la interfaz de usuario sin la necesidad de código adicional.

En el siguiente enlace podéis encontrar una explicación detallada sobre el data binding:
https://learn.microsoft.com/es-es/dotnet/desktop/wpf/data/?view=netdesktop-8.0

INotifyPropertyChanged

INotifyPropertyChanged es una interfaz en .NET que permite a las clases notificar a los suscriptores cuando una propiedad ha cambiado. Esto es especialmente útil en aplicaciones de interfaz de usuario (UI), como aquellas desarrolladas con WPF, ya que permite una actualización automática de la interfaz cuando los datos subyacentes cambian.

¿Para qué sirve?

Cuando una clase implementa la interfaz INotifyPropertyChanged, está indicando que tiene propiedades cuyos cambios deben ser notificados. Esto es útil en escenarios donde los objetos de esa clase se utilizan enlazados a la interfaz de usuario (UI), como en el caso de WPF. Al implementar esta interfaz, la clase puede alertar automáticamente a la interfaz de usuario sobre cualquier cambio en sus propiedades.

Cómo utilizar INotifyPropertyChanged:

1. *Implementar la interfaz:*

La interfaz INotifyPropertyChanged tiene un único evento llamado PropertyChanged. La clase que desea notificar cambios en sus propiedades debe implementar esta interfaz.

```
public class MiClase : INotifyPropertyChanged
{
   // Resto de la clase
   public event PropertyChangedEventHandler PropertyChanged;
   public void OnPropertyChanged(string propertyName)
   {
      PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
   }
}
```

2. *Disparar el evento cuando cambia una propiedad:*

En cada propiedad cuyo cambio quieras notificar, debes llamar al método OnPropertyChanged después de haber actualizado el valor de la propiedad.

```
private string miPropiedad;

public string MiPropiedad

{

   get { return miPropiedad; }

   set

   {

      if (miPropiedad != value)

      {

         miPropiedad = value;

         OnPropertyChanged(nameof(MiPropiedad));

      }

   }

}
```

Aquí, nameof(MiPropiedad) es una forma segura de referenciar el nombre de la propiedad sin tener que escribirlo como una cadena, lo que ayuda a prevenir errores de escritura.

3. *Uso en WPF:*

Cuando implementas INotifyPropertyChanged en clases que representan datos en una aplicación WPF, puedes aprovechar el enlace de datos (data binding) en XAML. Esto significa que los controles en tu interfaz de usuario se actualizan automáticamente cuando las propiedades subyacentes cambian.

Por ejemplo, en XAML puedes tener algo como esto:

```
<TextBox Text="{Binding MiPropiedad}" />
```

Aquí, el contenido del TextBox se actualizará automáticamente cada vez que la propiedad MiPropiedad cambie.

INotifyPropertyChanged es esencial cuando trabajas con aplicaciones de interfaz de usuario en las que los datos pueden cambiar y deseas que esos cambios se reflejen automáticamente en la interfaz. Al implementar esta interfaz, te aseguras de que las actualizaciones de propiedades se comuniquen eficientemente a la interfaz de usuario sin tener que realizar actualizaciones manuales en la interfaz. Esto es especialmente útil en frameworks como WPF.

ObservableCollection

ObservableCollection es una clase proporcionada en el espacio de nombres System.Collections.ObjectModel de .NET que implementa la interfaz INotifyCollectionChanged.

Esta interfaz notifica a los suscriptores cuando se realizan cambios en la colección, como agregar, quitar o actualizar elementos. En el contexto de WPP, ObservableCollection es especialmente útil para enlazar datos a controles y mantener la interfaz de usuario sincronizada con los cambios en la colección de datos.

Declaración y Creación:

Primero, debes declarar e instanciar tu ObservableCollection.

```
using System.Collections.ObjectModel;
// …
public partial class MainWindow : Window
{
        public ObservableCollection<string> MiColeccion { get; set; }

        public MainWindow()
        {
            InitializeComponent();
            // Inicializar la ObservableCollection
            MiColeccion = new ObservableCollection<string>();
            // Agregar algunos elementos de ejemplo
            MiColeccion.Add("Elemento 1");
            MiColeccion.Add("Elemento  2");
    // Asignar la ObservableCollection como el DataContext (para el enlace de datos)
    DataContext = this;
  }
}
```

Enlace de Datos en XAML:

Puedes enlazar la ObservableCollection a controles en tu interfaz de usuario mediante el uso de la propiedad ItemsSource. Por ejemplo, si tienes un ListBox:

```
<ListBox ItemsSource="{Binding MiColeccion}" />
```

En este caso, cualquier cambio en MiColeccion se reflejará automáticamente en el ListBox.

Realizar Cambios:

Como ObservableCollection implementa INotifyCollectionChanged, cualquier cambio que realices en la colección (agregar, quitar, etc.) notificará automáticamente a los controles enlazados para que actualicen su presentación.

```
// Agregar un nuevo elemento
MiColeccion.Add("Nuevo Elemento");


// Quitar un elemento
MiColeccion.Remove("Elemento 1");
```

Los controles enlazados a MiColeccion se actualizarán automáticamente para reflejar estos cambios.

ObservableCollection es particularmente útil en escenarios de desarrollo de aplicaciones WPF donde deseas que la interfaz de usuario refleje automáticamente cualquier cambio en la colección de datos sin tener que gestionar manualmente la actualización de la interfaz de usuario.