# *Apuntes clases DI*

08/01/2025

## UD2

El lenguaje **XAML** es un estándar basado en XML. A diferencia de HTML, donde las etiquetas son fijas, en XML se pueden definir etiquetas personalizadas, lo que permite una gran flexibilidad en la creación de estructuras.

**WPF = Windows Presentation Foundation**

- La etiqueta principal es **MainWindow**, que actúa como el contenedor de la aplicación. Por defecto, esta ventana tiene un tamaño predeterminado de 450 (Height) x 800 (Width) píxeles, aunque se puede modificar para adaptarse al contenido o a diferentes dispositivos.

- Dentro de **MainWindow**, se utiliza una etiqueta **Grid**, que define la zona donde se ubicarán los controles de la interfaz. El **Grid** permite organizar los elementos de manera estructurada y flexible que facilita la disposición de los mismos en la ventana.

Los **canvas** asignan coordenadas específicas a cada uno de los controles. **¿Qué es el canva?** Pues lo que asigna coordenadas específicas a cada uno de los controles.

Hay **dos tipos de componentes**: los contenedores y los que no lo son. Button, un textbox, una etiqueta, un datepicker, un radio button, no son componentes contenedores.
El padding está dentro de algo que no es un contenedor.
El **margin** es lo que te dice la distancia del componente a los límites de su contenedor.

```
<Window x:Class="WpfApp1.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/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   xmlns:local="clr-namespace:WpfApp1"
   mc:Ignorable="d"
 Title="MainWindow" Height="450" Width="800">
 <Grid>
    <TextBlock>Eso es un bloque de texto!!</TextBlock> // texto
    <TextBlock><Run Language="es-es" Text="Penalty for Real Madrid"/></TextBlock>
    <TextBlock Margin="30">Eso es un bloque de texto!!</TextBlock>
 </Grid>
</Window>
```

22-01-2025

El uso de **Material Design** en la creación de interfaces, específicamente en WPF, da un aspecto más moderno y amigable a las aplicaciones.

Existen tutoriales que nos enseñan a mejorar la interfaz de nuestras aplicaciones.

Por ejemplo, hay un tutorial que explica cómo crear un formulario de inicio de sesión utilizando **Material Design de Google**, lo que le da un aspecto más moderno y profesional a la interfaz.

Además, también hay un video sobre la **creación de una aplicación de música en WPF.** Este tutorial es interesante porque implementa Material Design, lo que mejora la apariencia y la experiencia de usuario en nuestras aplicaciones.

24-01-2025

**Material Design** se aplica para diseñar interfaces en WPF, con énfasis en cómo manejar y solucionar problemas relacionados con enlaces de recursos que pueden cambiar.

El uso de la aplicación de Material Design en la creación de interfaces con WPF puede presentar problemas relacionados con enlaces de recursos, especialmente debido a cambios en las URLs de las bibliotecas. La gestión adecuada de recursos externos es importante. (Esto cuando no te sale)

## UD3

28-01-2025

En el contexto de **Windows Forms**, los archivos **DLL (Dynamic Link Library)** son claves porque son bibliotecas que contienen código que puede ser utilizado por múltiples aplicaciones.

La creación de controles implica definir componentes visuales con los que los usuarios puedan interactuar, como botones o selectores. Aquí, el código que se presenta es similar al que se usa cuando se manejan eventos como **"onclick"** en Windows Forms. Se trata de **envolver este código en una biblioteca para que otros puedan usarlo sin tener que escribirlo desde cero.**

**Las propiedades personalizadas en los componentes visuales** se manejan a través de **getters** y **setters**, muy similares a Java, pero con una sintaxis más compacta en C#. Esto permite **modificar atributos** como el color, la opacidad o el texto de los controles.

Los **eventos** son respuestas a cambios en la interfaz de usuario. En desarrollo de interfaces, es común **definir eventos como click**, **mouseover**, **hold**, o **press** para botones, u **onChange** para **controles** como **checkboxes** o **radio buttons**. Estos eventos permiten que la interfaz de usuario **reaccione dinámicamente a las interacciones del usuario.**

29-01-2025

La **diferencia entre clases estáticas y no estáticas** es fundamental. Las **clases estáticas** no se pueden instanciar, mientras que las **no estáticas** requieren **crear un objeto** para usarlas.

La clase **MathService** incluye **métodos para realizar operaciones matemáticas** como suma, resta y división. Esto ayuda a organizar el código y hacerlo reutilizable.

La **importancia de manejar excepciones**, como la división por cero, es clave. Lanzar excepciones permite que el programa que llama al método pueda gestionar esos errores.

**Compilar una biblioteca de clases (DLL) y agregarla a otros proyectos** permite **reutilizar** el **código** en diferentes aplicaciones.

**Importar y utilizar la DLL en un proyecto de consola y en una aplicación de Windows Forms** facilita la reutilización del código y la colaboración.

Crear una interfaz gráfica simple para una calculadora utilizando los métodos de la DLL permite realizar operaciones matemáticas de manera más accesible.

31-01-2025

**La importancia de desarrollar interfaces visualmente atractivas se destaca**. Un **ejemplo** es la creación de un **tablero de ajedrez utilizando XAML**, donde **se modela la interfaz** y se implementa la lógica del juego en C#.

**API (Interfaz de Programación de Aplicaciones)** consiste en un **conjunto de métodos y funciones** que se pueden **utilizar sin necesidad de conocer su implementación interna**. Al usar APIs, como la de Java, se **interactúa con funciones** sin saber cómo están programadas.

La **gestión de memoria** en lenguajes como **C** y **C++** resalta que el programador es responsable de reservar y liberar memoria, lo que puede llevar a problemas como **"memory leaks"** (fugas de memoria). En contraste, lenguajes como **Java** y **C#** manejan la memoria automáticamente a través de un recolector de basura.

El concepto de **encapsulación en programación orientada a objetos** se aborda, donde los **atributos de un objeto** son accesibles sólo a través de **métodos específicos (getters y setters)**. Esto mejora la **seguridad** y la **organización** del código.

Se comparan **C, C++**, **C#** y **Java**, destacando que **C#** es **similar a Java en muchos aspectos**, pero con **diferencias en la gestión de memoria**. **C#** es un **lenguaje propietario de Microsoft**, mientras que **C es de código abierto.**