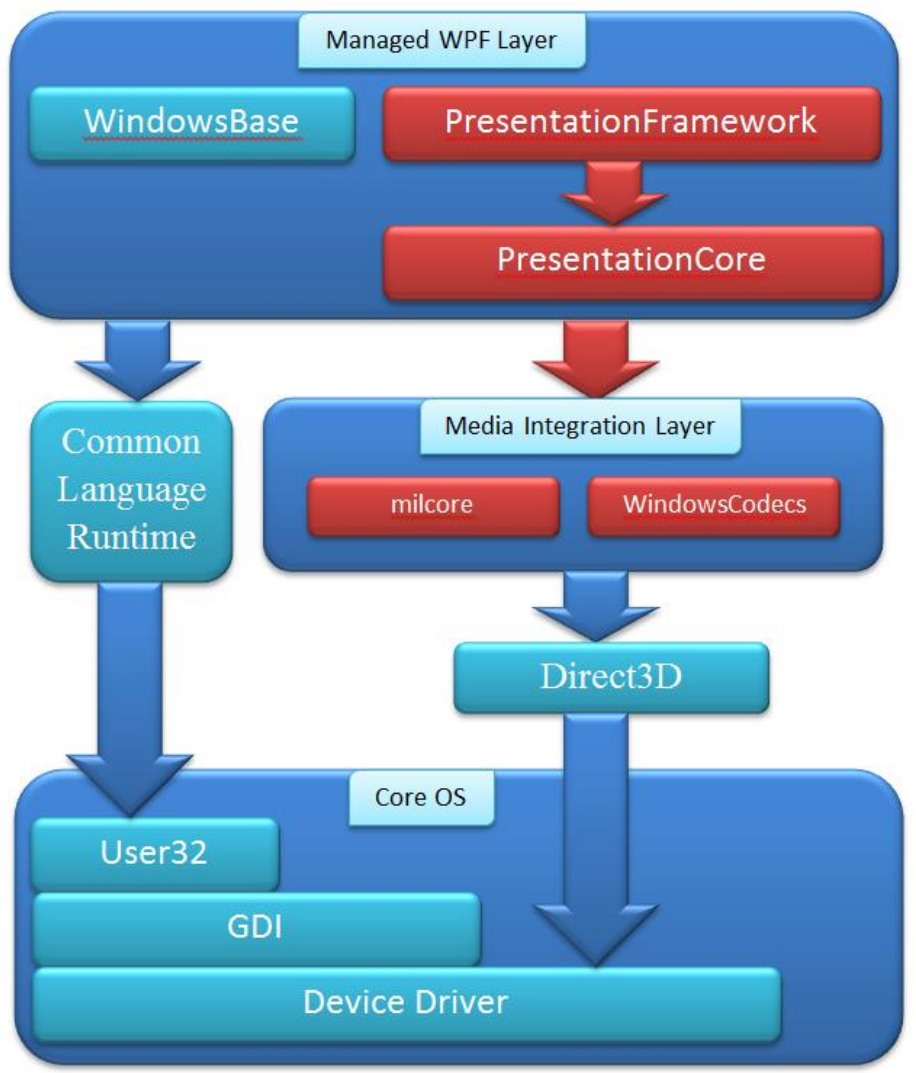
# ТЕХНОЛОГІЯ WINDOWS PRESENTATION FOUNDATION

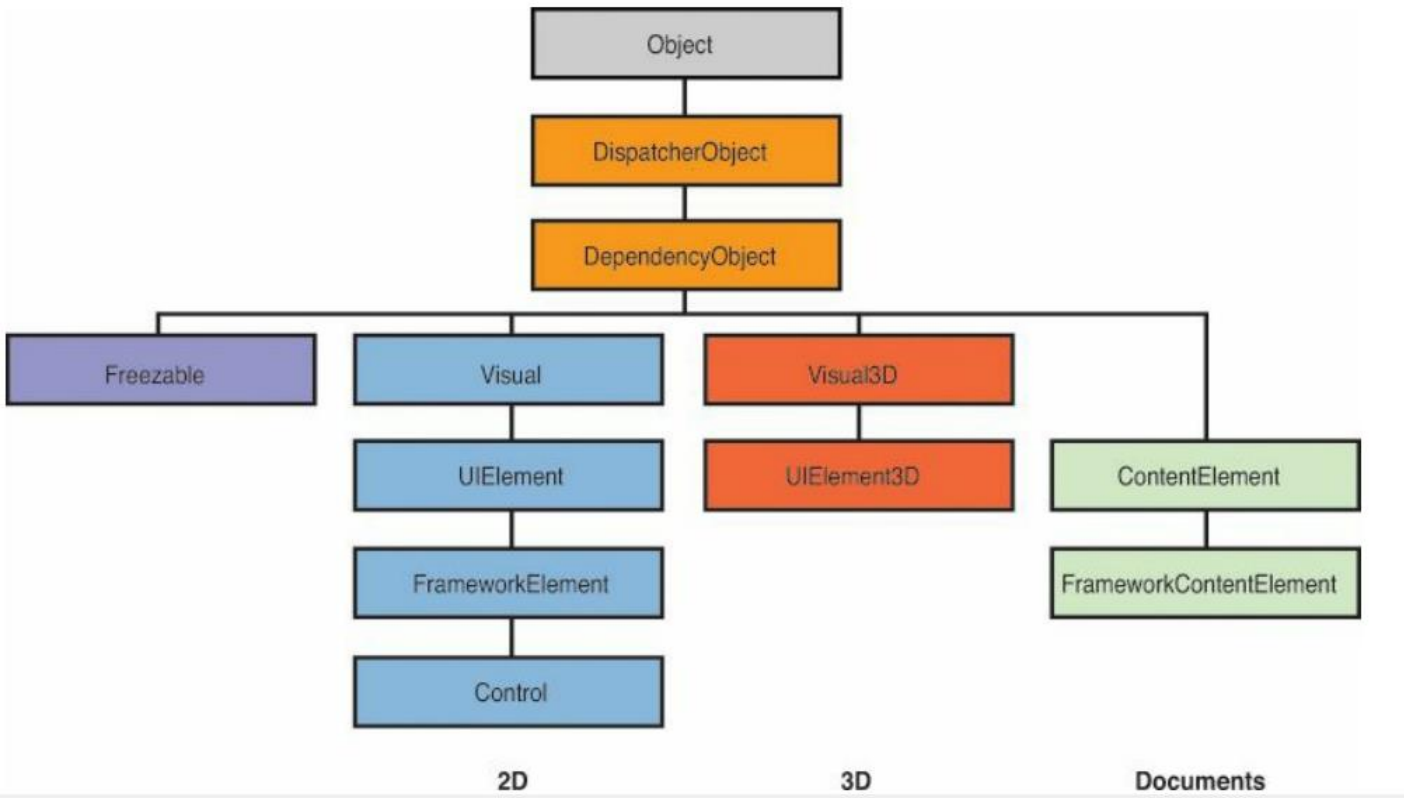
Питання 2.2.

# Архітектура WPF




- Presentation Framework – містить високорівневі типи даних WPF: Window, Controls, Styles, Layout Panels тощо.
  - Код та елементи управління з WPF-додатку в основному взаємодіють з цим прошарком.
- Presentation Core – включає базові типи, зокрема UIElement та Visual.
  - Майже всі елементи управління, з якими ви взаємодієте, породжені від цих типів. Presentation Framework використовує більшість типів, визначених на цьому рівні.
- MilCore – Media Integration Library – базова система рендерингу. MIL-код некерований. Даний прошарок перетворює WPF-елементи в Direct3D-сумісний формат.
- WindowsCodecs – забезпечує підтримку роботи із зображеннями: обробку, відображення, масштабування тощо.
- Direct3D – використовується для рендерингу графіки, створеної в WPF-додадтках.

# Базові класи WPF




- Фундаментальні класи технології WPF:
  - ***Object*** - базовий клас, від якого успадковуються решта .NET-класів.
  - ***DispatcherObject*** - базовий клас, призначений для об'єктів, до яких можна звертатись тільки з того потоку, де вони були створені.
    - Більшість класів WPF наслідують DispatcherObject и, следовательно, принципиально небезопасны относительно потоков.
    - Слово Dispatcher в назві класу відноситься до реалізованого в WPF варіанта циклу обробки повідомлень Win32.
  - ***DependencyObject*** - базовий клас, призначений для об'єктів, які підтримують властивості залежності.

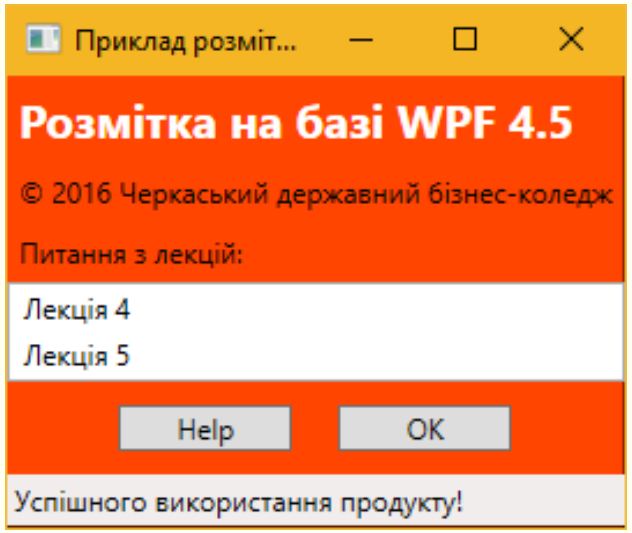
# Огляд ієрархії класів

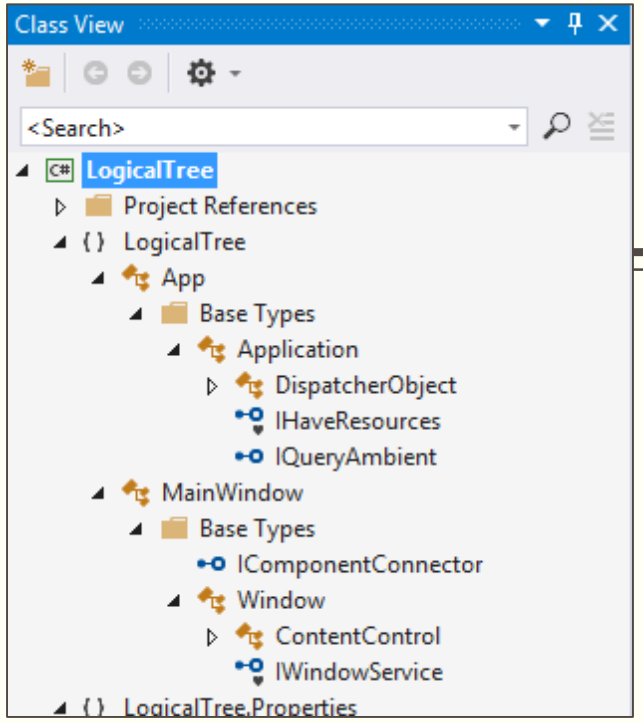
- Есть несколько фундаментальных для работы WPF классов:
  - ***Freezable*** - базовый класс для объектов, которые можно «заморозить в состоянии, разрешающем только чтение, - ради повышения производительности. К замороженным объектам можно безопасно обращаться из разных потоков, в отличие от объектов всех прочих классов, производных от DispatcberObject. Замороженный объект нельзя разморозить, однако можно клонировать, в результате чего получается незамороженная копия. По большей части объекты Freezable - это графические примитивы: кисти, перья, геометрические фигуры и т. д.
  - ***Visual*** - базовый класс для объектов, имеющих двумерное визуальное представление. Визуальные объекты подробно рассматриваются в главе 15 «Двумерная графика».
  - ***UlElement*** - базовый класс для двумерных визуальных объектов с поддержкой маршрутизации событий, привязки команд, компоновки и захвата фокуса. Эти механизмы обсуждаются в главе 5 «Компоновка с помощью панелей» и в главе 6 «События ввода: клавиатура, мышь, стилус и мультисенсорные устройства».
  - ***Visual3D*** — базовый класс для объектов, имеющих трехмерное визуальное представление. Рассматривается в главе 16 «Трехмерная графика».
  - ***UIElement3D*** - базовый класс для трехмерных визуальных объектов с поддержкой маршрутизации событий, привязки команд и захвата фокуса. Также рассматривается в главе 16.

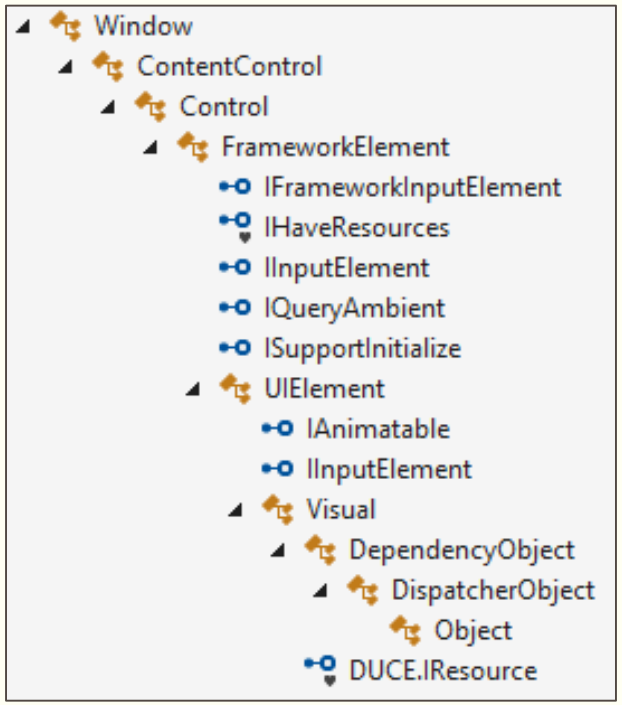
# Огляд ієрархії класів

- ***ContentElement*** - базовый класс, аналогичный UIElement, но предназначенный для тех частей содержимого, которые относятся к документам и потому не имеют собственного механизма визуализации.
  - Чтобы объект типа ContentElement появился на экране, им должен владеть объект класса, производного от Visual.
  - Часто для правильной визуализации объект ContentElement нуждается в нескольких объектах Visual (охватывающих несколько строк, столбцов и страниц).
- ***FrameworkElement*** - базовый класс, добавляющий поддержку стилей, привязки к данным, ресурсов и нескольких механизмов, общих для всех элементов управления в Windows, в частности всплывающих подсказок и контекстных меню.
- ***FrameworkContentElement*** - аналог FrameworkElement для содержимого.
- ***Control*** - базовый класс для таких хорошо знакомых элементов управления, как Button, ListBox и StatusBar. Класс Control добавляет к своему базовому классу FrameworkElement множество свойств, например Foreground, Background и FontSize, а также возможность тотального изменения стиля.

```
<Window x:Class="LogicalTree.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:LogicalTree"
        mc:Ignorable="d"
        Title="Приклад розмітки на базі WPF 4.5" SizeToContent="WidthAndHeight"
        Background="OrangeRed" d:DesignWidth="317.674" d:DesignHeight="252.63">
    <Grid>
        <StackPanel>
            <Label FontWeight="Bold" FontSize="20" Foreground="White">
                Розмітка на базі WPF 4.5
            </Label>
            <Label>© 2016 Черкаський державний бізнес-коледж</Label>
            <Label>Питання з лекцій:</Label>
            <ListBox>
                <ListBoxItem>Лекція 4</ListBoxItem>
                <ListBoxItem>Лекція 5</ListBoxItem>
            </ListBox>
            <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
                <Button MinWidth="75" Margin="10">Help</Button>
                <Button MinWidth="75" Margin="10">OK</Button>
            </StackPanel>
            <StatusBar>Успішного використання продукту!</StatusBar>
        </StackPanel>
    </Grid>
</Window>
```

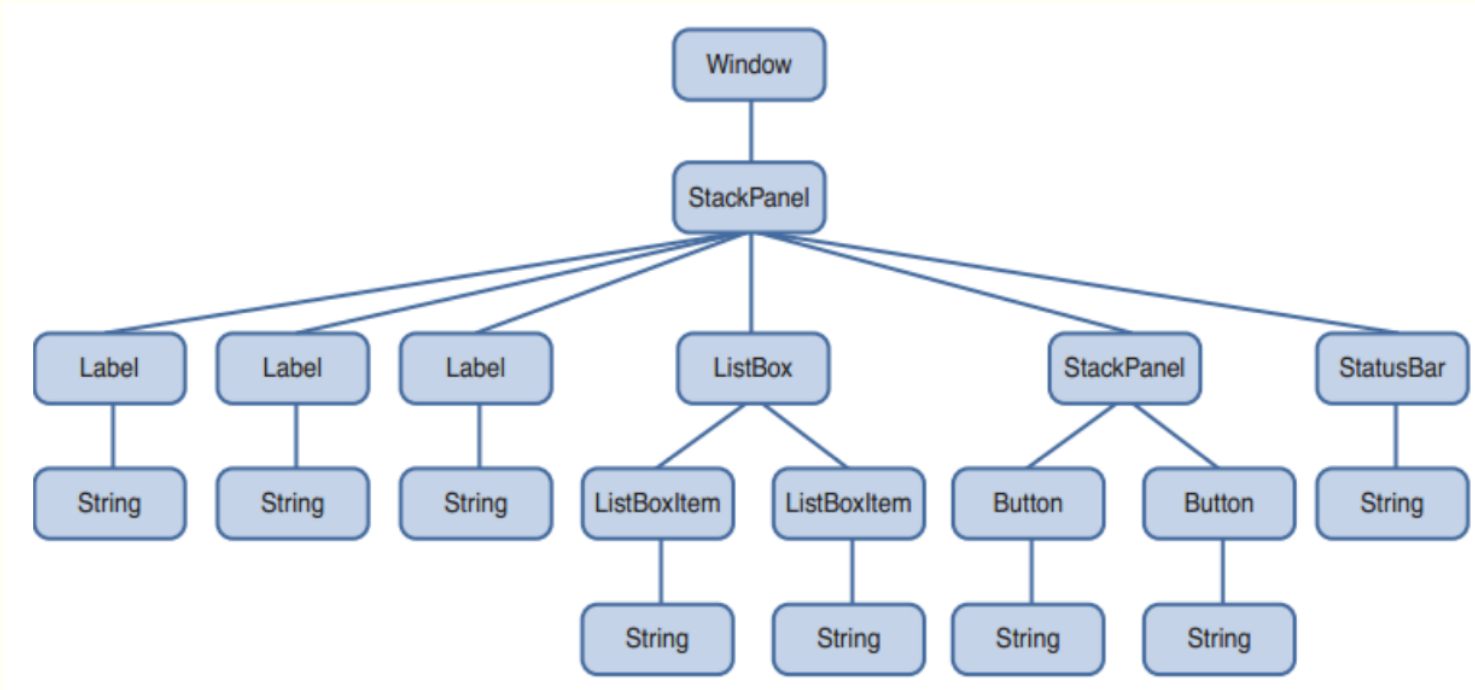





# Логічні та візуальні дерева

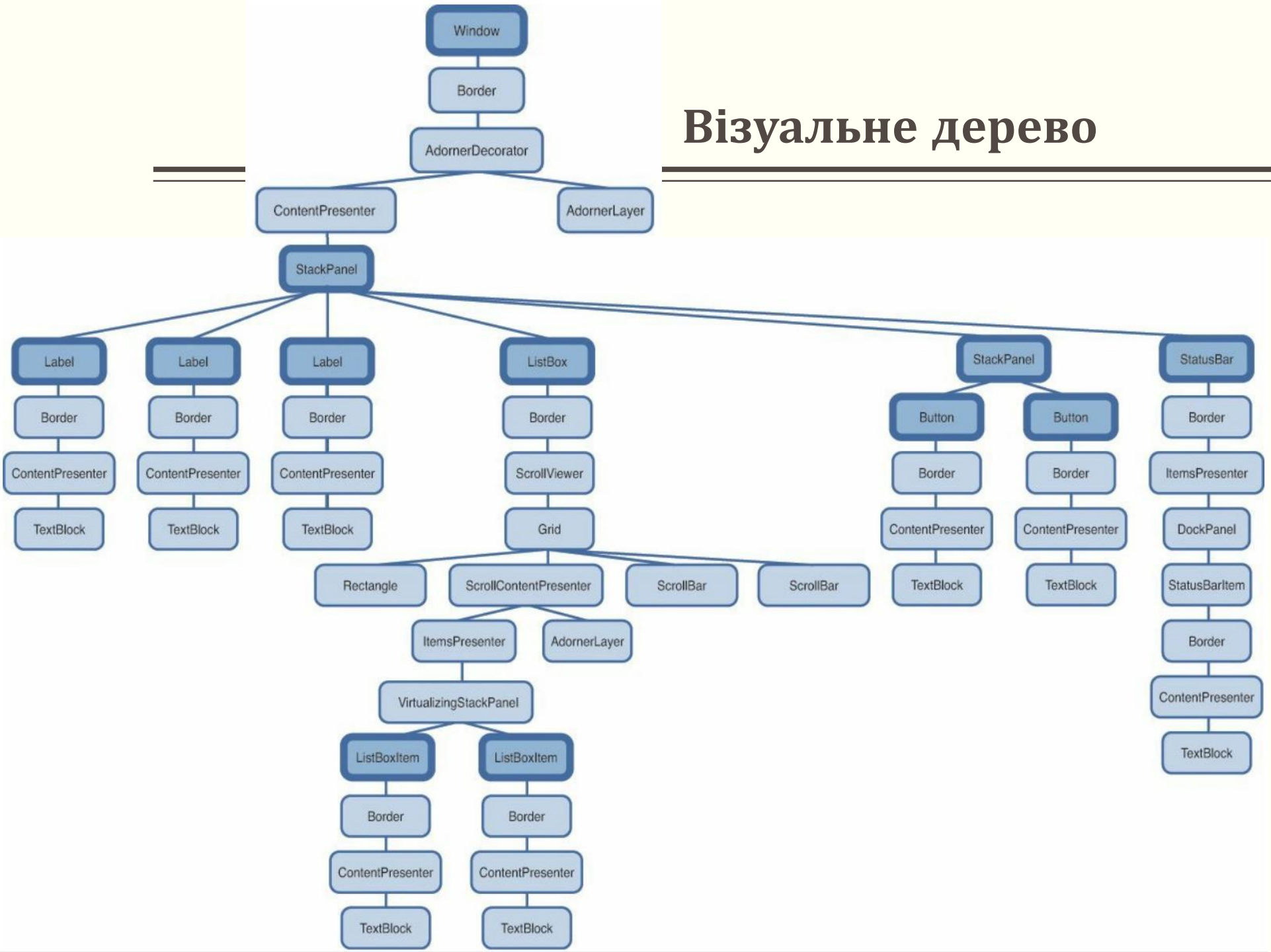
- У WPF інтерфейс користувача являється деревом об'єктів, яке називають ***логічним деревом.***
  - Тут коренем логічного дерева є об'єкт Window.
  - Дочірній елемент StackPanel містить кілька простих елементів управління та ще один StackPanel з двома кнопками Button.



# Візуальне дерево

- Повне дерево, яке містить всі візуалізовані елементи, називається ***візуальним деревом***.
  - Це розкрите логічне дерево, в якому кожен вузол є вершиною піддерева, що містить його візуальні компоненти.
  - Наприклад, ListBox – логічно єдиний елемент управління, проте за замовчуванням його візуальне представлення складається з більш простих WPF-елементів: рамки Border, двох стрічок прокрутки ScrollBar та ін.
- У візуальному дереві представлені не всі вузли логічного дерева, а лише елементи, породжені від класів System.Windows.Media.Visual або System.Windows.Media.Visual3D.
  - Решта не включаються, оскільки не володіють поведінкою візуалізації.
- Задумуватись про візуальне дерево має сенс лише тоді, коли потрібно радикально змінювати стилі елементів управління або виконувати низькорівневу рисовку

# Візуальне дерево



# Властивості залежності

- У WPF з’явились ***властивості залежності***; використовуються для призначення стилів, автоматичної прив’язки до даних, анімації та ін.
  - Залежить від ***постачальників***, які визначають її значення підчас виконання. Постачальником може бути анімація, батьківський елемент, що розповсюджує свої значення на потомків, тощо.
  - Жодна .NET-сумісна мова програмування (крім XAML) нічого не знає про властивості залежності.

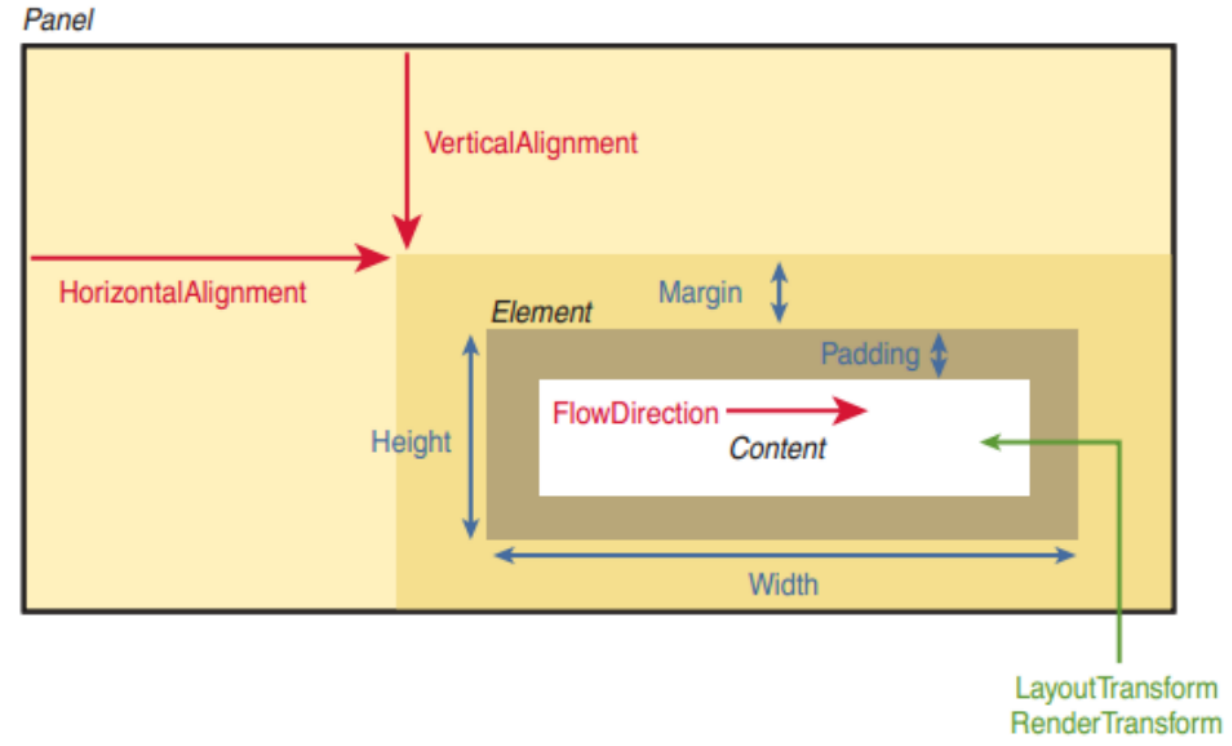
- Розширюють функціональність звичайних властивостей .NET у наступних напрямках:
  - Сповіщення про зміни
  - Успадкування значень властивостей
  - Підтримка кількох постачальників

# Реалізація властивості залежності

```
public class Button : ButtonBase
{
    // Свойство зависимости
    public static readonly DependencyProperty IsDefaultProperty;

    static Button()
    {
        // Зарегестрировать свойство
        Button.IsDefaultProperty = DependencyProperty.Register(“IsDefault”,
        typeof(bool), typeof(Button),
        new FrameworkPropertyMetadata(false,
        new PropertyChangedCallback(OnIsDefaultChanged)));
        ...
    }

    // Обертка в виде обычного свойства .NET (необязательно)
    public bool IsDefault
    {
        get { return (bool)GetValue(Button.IsDefaultProperty); }
        set { SetValue(Button.IsDefaultProperty, value); }
    }
//  Метод, вызываемый при изменении свойства (необязательно)
    private static void OnIsDefaultChanged(
        DependencyObject o, DependencyPropertyChangedEventArgs e) { … }
        ...
}
```

- За угодою всі поля типу DependencyProperty відкриті, статичні та мають назву, яка оканчивающееся словом Property.
  - Деякі компоненти інфраструктури вимагають обов'язкового дотримання цієї угоди, наприклад засоби локалізації, завантажувачі XAML та ін.

# Елементи управління та їх основні властивості




- Процедура встановлення розмірів і положень елементів управління (та інших елементів) називається ***компонуванням***, або ***верстkою макета***.
  - В її основі лежить механізм взаємодії між елементами-батьками та їхніми нащадками.
  - Спільно вони домовляються про остаточні розміри і положення.
  - Батьківські елементи, що підтримують компонування декількох дітей, називаються панелями, вони успадковуються від абстрактного класу System.Windows.Controls.Panel.
  - Всі елементи, які беруть участь в процесі компонування (як батьки, так і нащадки), успадковуються від класу System.Windows.UIElement.

# Керування розміром. Властивості Height і Width

- У всіх класах, породжених від FrameworkElement, є властивості Height і Width (типу double), а також MinHeight, MaxHeight, MinWidth і MaxWidth, якими можна користуватись для встановлення діапазону допустимих значень.
  - Всі ці властивості можна задавати в будь-якій комбінації як у процедурному коді, так і в XAML.
  - Зазвичай елемент намагається прийняти мінімально можливий розмір, тому якщо задано MinHeight або MinWidth, то при візуалізації обирається саме така висота або ширина за умови, що вміст не змушує збільшити розмір.
  - Проте збільшення можна обмежити за допомогою властивостей MaxHeight і MaxWidth (за умови, що ці значення більше відповідних мінімальних).
  - Якщо одночасно з мінімальними і максимальними значеннями задані властивості Height і Width, то останні мають пріоритет за умови, що потрапляють всередину діапазону між Min і Max.
  - Властивості Height і Width класу FrameworkElement за умовчанням приймають значення Double.NaN або явно задаються в XAML як “NaN” або “Auto” (нечутливі до регістру).
- Уникайте явного задавання розмірів!
  - Якщо явно задавати розміри елементів управління, особливо породжених від класу ContentControl, виникає ризик відсікання частини тексту при зміні системного шрифту чи перекладі тексту.
  - Наявність панелей дозволяє нечасто користуватись явним заданням розмірів.

# Керування розміром

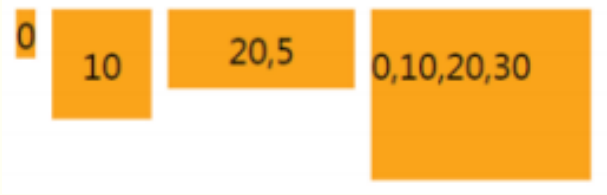
- У класі FrameworkElement є ще кілька властивостей, що відносяться до розмірів:
  - DesiredSize (успадковується від UIElement)
  - RenderSize (успадковується від UIElement)
  - ActualHeight і ActualWidth
- На відміну від попередніх 6 властивостей, вони є не вхідними даними для процедури компонування, а вихідними — представляють результат компонування, а тому доступні лише для зчитування.
  - Властивість ***DesiredSize*** обчислюється в ході компонування на основі значень інших властивостей (у тому числі Width, Height, MinXXX та MaxXXX) та місця, яке батьківський елемент готовий виділити.
  - Воно використовується панелями для внутрішніх цілей.
  - Властивість ***RenderSize*** представляє остаточний розмір елемента після завершення компонування, а ***ActualHeight*** і ***ActualWidth*** – те ж саме, що й RenderSize.Height і RenderSize.Width.
- Яким би способом не задавався розмір елемента, батьківський елемент вправі змінити остаточний розмір елемента на екрані.
  - Тому ці властивості корисні тоді, коли поведінка програми залежить від розміру елемента.
  - З іншого боку, значення решти властивостей щодо розміру цікаві для формулювання алгоритму.
  - Наприклад, якщо Height і Width не задані явно, вони отримають значення Double.NaN, незалежно від істинного розміру елемента.

# Керування розміром

- Будьте обережні з використанням у коді властивостей ActualHeight, ActualWidth або RenderSize!
  - При кожному компонуванні значення RenderSize (і ActualHeight, ActualWidth) оновлюється.
  - Проте компонування відбувається асинхронно, тому неможливо розраховувати на те, що в будь-який момент ці значення правильні.
  - Звертатись до них безпечно лише всередині обробника події LayoutUpdated, визначеного в класі UIElement.
  - Інший спосіб – метод UpdateLayout з класу UIElement, який синхронно виконує всі відкладені оновлення макета. Не рекомендується для застосування: часті звернення до UpdateLayout можуть негативно вплинути на продуктивність, а також немає гарантії коректної обробки реентерабельності в методах, що відносяться до компонування.

# Властивості Margin і Padding

- Властивість Margin визначено для всіх об'єктів, породжених від FrameworkElement.
- Властивість Padding – у всіх елементах управління, породжених від класу Control (а також у класі Border).
  - Margin задає зовнішнє поле навколо елемента, а Padding– внутрішній відступ між вмістом елементу та його межами.
  - Обидві властивості мають тип System.Windows.Thickness – клас, який може представляти одне, два або чотири значення типу double.



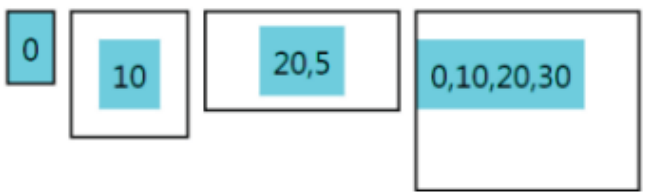

```
<!-- Отступы: -->

<!-- 1 значение: один и тот же отступ со всех четырех сторон: -->
<Label Padding="0" Background="Orange">0</Label>
<Label Padding="10" Background="Orange">10</Label>

<!-- 2 значения: первое значение относится к левому и правому
                    отступам, второе - к верхнему и нижнему:-->
<Label Padding="20,5" Background="Orange">20,5</Label>

<!-- 4 значения: левый, верхний, правый, нижний: -->
<Label Padding="0,10,20,30" Background="Orange">0,10,20,30</Label>
```

# Властивості Margin і Padding

```
<!-- Поля: -->

<Border BorderBrush="Black" BorderThickness="1">
    <!-- Поле отсутствует: -->
    <Label Background="Aqua">0</Label>
</Border>

<Border BorderBrush="Black" BorderThickness="1">
    <!-- 1 значение: одно и то же поле со всех четырех сторон: -->
    <Label Margin="10" Background="Aqua">10</Label>
</Border>

<Border BorderBrush="Black" BorderThickness="1">
!-- 2 значение: первое значение относится к левому и правому,
   полям, второе – к верхнему и нижнему : -->
<Label Margin="20,5" Background="Aqua">20,5</Label>
</Border>

<Border BorderBrush="Black" BorderThickness="1">
<!-- 4 значение: левое,верхнее,правое,нижнее: -->
<Label Margin="0,10,20,30" Background="Aqua">0,10,20,30</Label>
</Border>
```

- Синтаксис встановлення значень через кому, який підтримують властивості Margin і Padding, забезпечує конвертер типу System.Windows.ThicknessConverter, який конструює об'єкт типу Thickness із рядка.

```
myLabel.Margin = new Thickness(10);      // То же, что Margin="10" в XAML
myLabel.Margin = new Thickness(20, 5, 20, 5); // То же, что Margin="20,5" в XAML
myLabel.Margin = new Thickness(0,10,20,30); // То же, что Margin="0,10,20,30" в XAML
```



# Які одиниці вимірювання застосовуються в WPF?

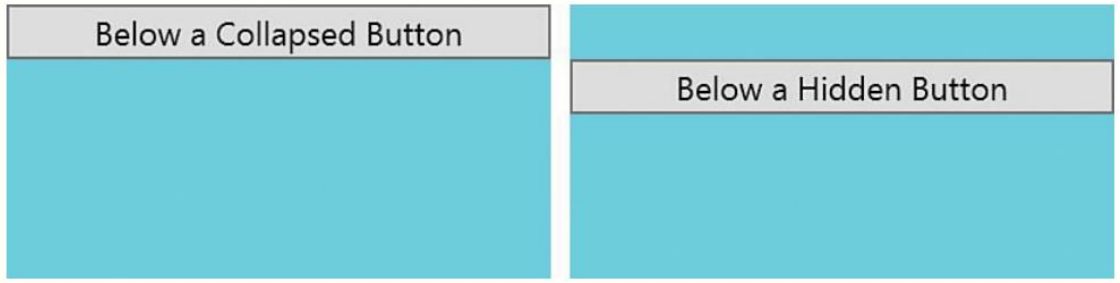
- Конвертер LengthConverter, що асоціюється з різними властивостями стосовно довжин, підтримує явне задання одиниць вимірювання cm, pt, in і px (за умовчанням).
  - За умовчанням всі абсолютні розміри виражаються в незалежних від пристрою пікселях.
  - Такі «логічні пікселі» представляють 1/96 дюйма незалежно від роздільної здатності екрану, вираженої в точках на дюйм (DPI).
  - Кількість незалежних від пристрою пікселів завжди задається значенням з подвійною точністю.
- Точна величина - 1/96 дюйма - не так суттєва, а обрана тому, що на типовому екрані з роздільною здатністю 96 DPI один незалежний від пристрою піксель у точності співпадає з одним фізичним пікселем.
  - Поняття «істинного» дюйма залежить від фізичного пристрою відображення.
  - Якщо додаток нарисує відрізок довжиною 1 дюйм на екрані ноутбука, то при виводі зображення на проектор довжина цього відрізка, очевидно, зміниться!

# Властивість Visibility

- Тип властивості елемента Visibility - не Boolean, а перелічення System.Windows.Visibility з трьома станами:
  - Visible – елемент видно, і він бере участь у компонуванні.
  - Collapsed – елемент не видно, і він не бере участь у компонуванні.
  - Hidden – елемент не видно, проте він бере участь у компонуванні.
- Згорнутий (Collapsed) елемент, по суті, має нульовий розмір, а прихований (Hidden) – зберігає свій початковий розмір.
  - Панель StackPanel зі згорнутою кнопкою:

```
<StackPanel Height="100" Background="Aqua">
<Button Visibility="Collapsed">Collapsed Button</Button>
<Button>Below a Collapsed Button</Button>
</StackPanel>
```

  - порівнюється з панеллю StackPanel з прихованою кнопкою:




```
<StackPanelHeight="100"Background="Aqua">
<Button Visibility="Hidden">Hidden Button</Button>
<Button>Below a Hidden Button</Button>
</StackPanel>
```

# Управління положенням

```
<StackPanel>
<Button HorizontalAlignment="Left" Background="Red">Left</Button>
<Button HorizontalAlignment="Center" Background="Orange">Center</Button>
<Button HorizontalAlignment="Right" Background="Yellow">Right</Button>
<Button HorizontalAlignment="Stretch" Background="Lime">Stretch</Button>
</StackPanel>
```


- За допомогою властивостей HorizontalAlignment і VerticalAlignment елемент може управляти розподілом надлишкового простору, виділеного йому батьківським елементом.
  - Значеннями властивостей є одноіменні перелічення з простору імен System.Windows:
    - HorizontalAlignment - Left, Center, Right, Stretch
    - VerticalAlignment - Top, Center, Bottom, Stretch
- Навіть якщо для елементу задано розтягування (Stretch) по горизонталі або вертикалі, пріоритет все-одно відаається явно заданій висоті чи ширині.
  - Властивості MaxHeight і MaxWidth теж більш пріоритетні, проте лише у випадку, коли їх значення менші за розмір, отриманий після розтягування.
  - Аналогічно для MinHeight і MinWidth.
  - Якщо властивість Stretch використовується в контексті, де на розмір елемента накладаються обмеження, то вона діє як тип вирівнювання Center (або Left, якщо елемент занадто великий і не може відцентруватись всередині свого батьківського елемента).

# Вирівнювання вмісту

- Крім HorizontalAlignment і VerticalAlignment, у класі Control є властивості HorizontalContentAlignment і VerticalContentAlignment.
  - Вони визначають порядок розміщення вмісту всередині елементу управління.

```
<StackPanel>
  <Button HorizontalContentAlignment="Left" Background="Red">Left</Button>
  <Button HorizontalContentAlignment="Center" Background="Orange">Center</Button>
  <Button HorizontalContentAlignment="Right" Background="Yellow">Right</Button>
  <Button HorizontalContentAlignment="Stretch" Background="Lime">Stretch</Button>
</StackPanel>
```

- Властивості вирівнювання вмісту належать до тих же типів перелічень, що й відповідні їм властивості вирівнювання.
  - Проте за умовчанням властивість HorizontalContentAlignment = Left, а VerticalContentAlignment = Top.

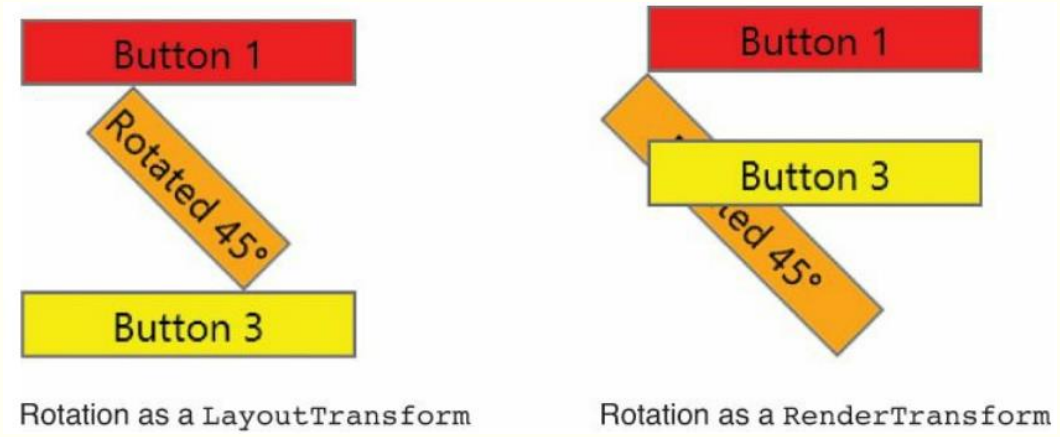
# Властивість FlowDirection

- Визначена в класі FrameworkElement (та ще деяких), дозволяє змінити напрям візуалізації внутрішнього вмісту елемента.
  - Застосовується до деяких панелей, де впливає на розміщення дочірніх елементів, а також модифікує спосіб вирівнювання вмісту всередині дочірніх елементів.
  - Тип властивості – перелічення System.Windows.FlowDirection, що набуває 2 значень: LeftToRight (за умовчанням у класі FrameworkElement) та RightToLeft.
  - Ідея: для мови, що записується справа наліво, потрібно задавати напрям RightToLeft.
  - Властивість FlowDirection не впливає на напрям запису букв у написі всередині кнопок: англійські букви завжди записуються зліва направо, арабські – справа наліво.
  - Прое поняття лівого і правого у решті частин інтерфейсу, які зазвичай повинні відповідати напрямку запису букв, міняються місцями.

```
<StackPanel>
  <Button FlowDirection="LeftToRight"
          HorizontalContentAlignment="Left" VerticalContentAlignment="Top"
          Height="40" Background="Red">LeftToRight</Button>
  <Button FlowDirection="RightToLeft"
          HorizontalContentAlignment="Left" VerticalContentAlignment="Top"
          Height="40" Background="Orange">RightToLeft</Button>
</StackPanel>
```

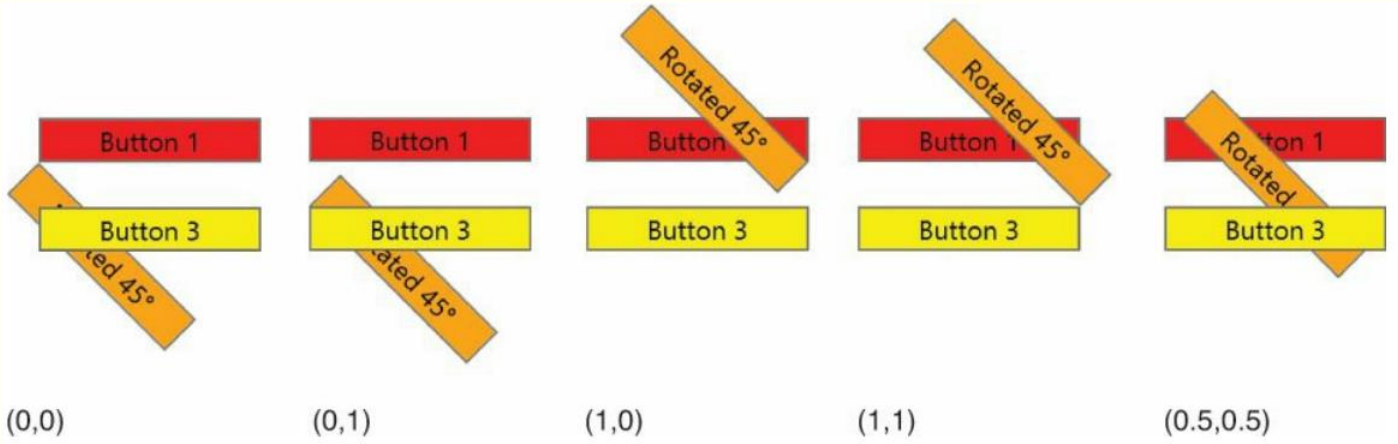
# Застосування перетворень

- У всіх підкласах FrameworkElement існує 2 властивості типу Transform, які дозволяють застосовувати перетворення:
  - LayoutTransform – застосовується до компонування елемента
  - RenderTransform (успадковано від UIElement) - застосовується після завершення компонування (безпосередньо перед візуалізацією елемента)
- В обох випадках перетворення застосовано до другої кнопки в прикладі.
  - Проте при LayoutTransform третя кнопка зсувається вниз, а при RenderTransform – залишається на місці.



# Властивість RenderTransformOrigin

- У класі UIElement наявна також властивість RenderTransformOrigin, яка представляє початкову (нерухому) точку перетворення.
  - Для перетворення RotateTransform початком є лівий верхній кут кнопки, навколо якого кнопка повертається.
  - Для перетворень, застосованих в режимі LayoutTransform, поняття початкової точки не визначено, тому що положення перетвореного елемента диктується правилами компонування, встановленими батьківським елементом.
- RenderTransformOrigin має тип System.Windows.Point, за умовчанням – (0,0).
  - Цій точці відповідає лівий верхній кут елемента.
  - Точка (0,1) представляє лівий нижній кут, (1,0) - правий верхній кут, а (1,1) - правий нижній кут.
  - Можна задавати й числа, більші за 1, - тоді початкова точка виявиться поза межами елемента.
  - Дробові значення теж допустимі. Зорема, точка (0.5, 0.5) представляє середину об'єкта.



- Завдяки конвертеру System.Windows.PointConverter значення RenderTransform.Origin можна задавати в XAML у вигляді 2 чисел, розділених комою (без дужок).
  - Для стандартних елементів управління зі стилями за умовчанням подібне перетворення виглядає безглуздо.
  - Часто перетворення мають смисл у додатках з сильно перевантаженими темами, проте навіть для стилізованих за умовчанням елементів перетворення разом з анімацією можуть створити цікаві ефекти.

```
<Button RenderTransformOrigin="0.5,0.5" Background="Orange">
<Button.RenderTransform>
<RotateTransform Angle="45"/>
</Button.RenderTransform>
    Rotated 45°
</Button/>
```

# 5 вбудованих 2D-перетворень, визначених у просторі імен System.Windows.Media

- **RotateTransform**
- **ScaleTransform**
- **SkewTransform**
- **TranslateTransform**
- **MatrixTransform**

- Перетворення RotateTransform повертає елемент ввідповідно до наступних властивостей типу double:
  - Angle - кут повороту в градусах (за умовчанням 0)
  - CenterX - абсциса центра повороту (за умовчанням 0)
  - CenterY - ордината центра повороту (за умовчанням 0)
- Точка (CenterX, CenterY) за умовчанням відповідає лівому верньому куту.
  - Властивості CenterX і CenterY беруться до уваги тільки тоді, коли перетворення застосовується в режимі RenderTransform, оскільки для режиму LayoutTransform положення центра повороту визначається батьківською панеллю.

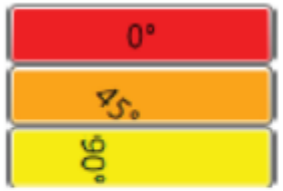
# Різниця між властивостями CenterX, CenterY для перетворень виду RotateTransform і властивістю RenderTransformOrigin елементу типу UIElement

- Здається, що при перетворенні елементу типу UIElement властивості CenterX, CenterY надмірні, оскільки вже є RenderTransformOrigin.
  - Обидва механізми визначають точку відліку для перетворення та обидва працюють тільки за умови перетворення в режимі RenderTransform.
- Проте CenterX і CenterY задають абсолютне положення початкової точки, а RenderTransformOrigin – відносне (в незалежних від пристрою пікселях).
  - Тому для правого верхнього кута елемента з шириною Width = 20 властивість CenterX = 20, CenterY = 0, а не (1, 0), як для RenderTransformOrigin.
  - Також при комбінації кількох перетворень RenderTransform задання CenterX і CenterY для окремих перетворень забезпечує більш точний контроль.
  - Відокремлені значення CenterX і CenterY типу double простіше використовувати для прив'язки даних, ніж одне значення RenderTransform типу Point.
- Проте властивість RenderTransform загалом корисніша за CenterX і CenterY.
  - Простіше записуються відносні координати: для CenterX і CenterY довелось би писати процедурний код, який обчислював би абсолютні зміщення.
  - Зауважте, що RenderTransform можна задавати паралельно з CenterX і CenterY.

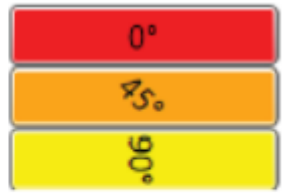
# Демонстрація перетворенння RotateTransform в режимі RenderTransform

- Застосовуємо до внутрішнього вмісту кнопок з 2 різними значеннями RenderTransformOrigin.
  - Може здатись, що елементи TextBlock всередині Button зліва повернуті не навколо лівого верхнього кута кнопки, проте це тому, що сам елемент TextBlock трохи більший за текст всередині нього.
  - RotateTransform має параметризовані конструктори, які приймають значення кута або кута і центру, для зручності виконання перетворення з процедурного коду.

```
<Button Background="Orange">
  <TextBlock RenderTransformOrigin="0.5,0.5">
  <TextBlock.RenderTransform>
    <RotateTransform Angle="45"/>
  </TextBlock.RenderTransform>
    45°
  </TextBlock>
</Button>
```



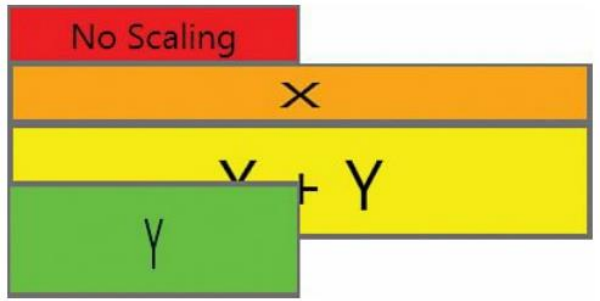

Поворот текста вокруг средней точки




Поворот текста вокруг средней точки

# Перетворення ScaleTransform

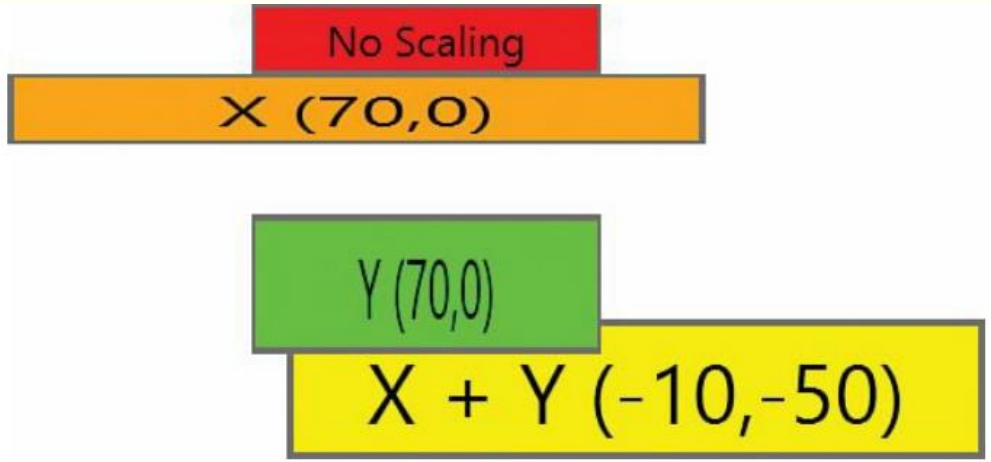
```
<StackPanel Width="100">
  <Button Background="Red">No Scaling</Button>
  <Button Background="Orange">
  <Button.RenderTransform>
    <ScaleTransform ScaleX="2"/>
  </Button.RenderTransform>
    X</Button>
  <Button Background="Yellow">
  <Button.RenderTransform>
    <ScaleTransform ScaleX="2" ScaleY="2"/>
  </Button.RenderTransform>
    X + Y</Button>
  <Button Background="Lime">
  <Button.RenderTransform>
    <ScaleTransform ScaleY="2"/>
  </Button.RenderTransform>
    Y</Button>
</StackPanel>
```




- Збільшує чи зменшує елемент по горизонталі, вертикалі чи в обох напрямках.
- Має 4 властивості типу double:
  - ScaleX - коефіцієнт зміни ширини елемента (за умовчанням 1)
  - ScaleY - коефіцієнт зміни висоти елемента (за умовчанням 1)
  - CenterX – початкова точка для масштабування по горизонталі (за умовчанням 0)
  - CenterY - початкова точка для масштабування по вертикалі (за умовчанням 0)
- Якщо ScaleX = 0.5, то ширина відрисованого елемента зменшиться вдвічі, а якщо 2, то вдвічі збільшиться.
  - Смисл властивостей CenterX і CenterY такий же, як для перетворення RotateTransform.

# Перетворення ScaleTransform

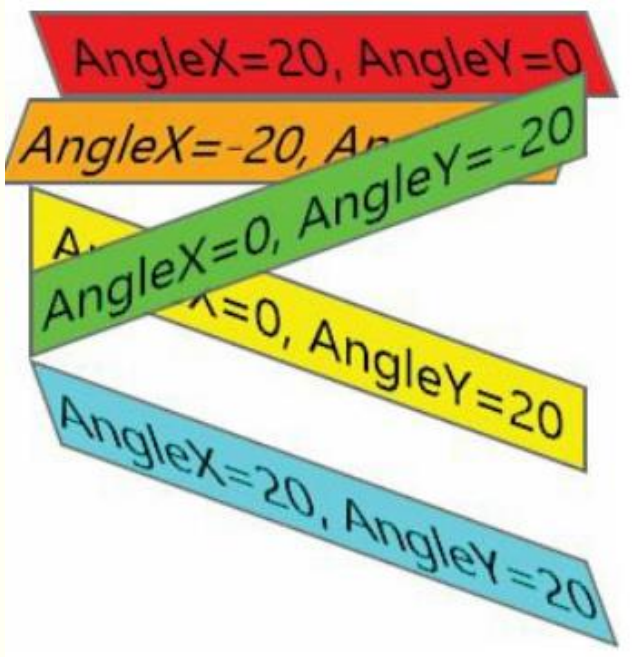
- Відобразимо кнопки з попереднього коду, для яких додатково задані властивості CenterX і CenterY.
  - У якості тексту кожної кнопки вказано координати початкової точки.
  - Зверніть увагу, що зелена кнопка не зсунута вліво, як оранжева, хоч CenterX в обох випадках = 70.
  - CenterX приймається до уваги тільки якщо значення ScaleX відрізняється від 1, а CenterY – якщо ScaleY не дорівнює 1.
  - Як і в решті класів перетворень, у ScaleTransform визначено кілька конструкторів для зручності створення об’єктів у процедурному коді.



# Вплив перетворень, подібних до ScaleTransform, на властивості ActualHeight і ActualWidth елемента типу FrameworkElement та на властивість RenderSize елемента типу UIElement

- Застосування перетворення до елементу типу FrameworkElement ніколи не змінює значень цих властивостей, незалежно від використання RenderTransform чи LayoutTransform.
  - Тому властивості ActualHeight і ActualWidth можуть містити «хибний» розмір елемента на екрані.
  - Проте таке рішення правильне, оскільки не зовсім зрозуміло, як слід виражати ці значення для деяких перетворень, а також через одноманітність перетворень довільних елементів при ілюзії їх нормальної візуалізації.
- Як перетворення ScaleTransform впливає на властивості Margin і Padding?
  - Властивість Padding масштабується разом з усім вмістом (відступ знаходиться всередині елемента).
  - Властивість Margin не масштабується, незважаючи на візуальний ефект.
- Якщо перетворення ScaleTransform застосовується в режимі LayoutTransform до елемента, який уже розтягнутий у напрямку масштабування, тоді воно береться до уваги лише в тому випадку, коли розмір після масштабування перевищує розмір у результаті розтягування.

# Перетворення SkewTransform




- Нахиляє елемент відповідно до значень 4 властивостей типу double:
  - AngleX– кут нахилу по горизонталі (за умовчанням 0)
  - AngleY– кут нахилу по вертикалі (за умовчанням 0)
  - CenterX– початкова точка для нахилу по горизонталі (за умовчанням 0)
  - CenterY– початкова точка для нахилу по вертикалі (за умовчанням 0)

# Перетворення TranslateTransform

- Паралельно переносить елемент відповідно до значень 2 властивостей типу double:
  - X - величина зміщення по горизонталі (за умовчанням 0)
  - Y - величина зміщення по вертикалі (за умовчанням 0)
- TranslateTransform не дає жодного ефекту, коли застосовується в режимі LayoutTransform, проте використання в режимі RenderTransform зручне, щоб «посунути» елементи.
  - Частіше за все це робиться динамічно у відповідь на дії користувача (чи в складі анімації).

# Перетворення MatrixTransform

- Низькорівневий механізм опису довільного двовимірного перетворення.
  - Доступна єдина властивість Matrix(типу System.Windows.Media.Matrix), що представляє матрицю перетворення розміром 3x3.


- MatrixTransform – єдине перетворення, конвертер якого дозволяє описати його в XAML простим рядком (клас TransformConverter).
  - Наприклад, для переміщення кнопки на 10 одиниць вправо і на 20 одиниць вниз:

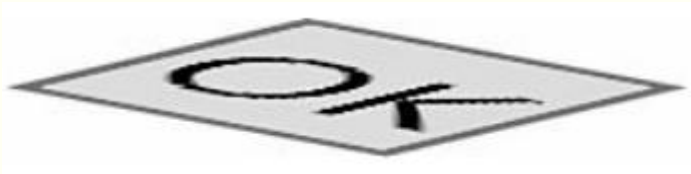
```
<Button RenderTransform="1,0,0,1,10,20" />
```

  - Порядок: M11, M12, M21, M22, OffsetX, OffsetY.

# Комбінування перетворень

- Клас TransformGroup також успадковується від класу Transform, а його задача – скомбінувати кілька дочірніх об'єктів типу Transform.
  - В процедурном коде объекты отдельных преобразований добавляются в коллекцию Children, в XAML это делается следующим образом:

```
<Button>
<Button.RenderTransform>
  <TransformGroup>
    <RotateTransform Angle="45"/>
    <ScaleTransform ScaleX="5" ScaleY="1"/>
    <SkewTransform AngleX="30"/>
  </TransformGroup>
</Button.RenderTransform>
  OK
</Button>
```




- Для підвищення продуктивності WPF спочатку обчислює комбіноване перетворення на основі потомків об'єкта TransformGroup, а потім застосовує одне результуюче перетворення.
  - У складі TransformGroup може кілька разів траплятись одне перетворення.
  - Например, два поворота MatrixTransform на 45°, эквивалентные одному повороту на 90°

# ДЯКУЮ ЗА УВАГУ!

**Наступне питання: Компонування інтерфейсу за допомогою контейнерних елементів**

# Не всі елементи типу FrameworkElement підтримують перетворення

Элементы, содержимое которых не является «родным» для WPF, не поддерживают преобразования, хотя и наследуют свойства LayoutTransform и RenderTransform. Например, к их числу относится элемент HwndHost, выступающий в роли владельца GDI-содержимого (обсуждается в главе 19 «Интероперабельность с другими технологиями»). Элемент управления Frame, который, в принципе, может содержать HTML-разметку (описывается в главе 9 «Однодетные элементы управления»), поддерживает преобразования в полном объеме, только если в нем нет HTML. В противном случае преобразование ScaleTransform можно применять для изменения размера, но содержимое при этом не масштабируется.

На рис. 4.15 показано, что происходит, когда на панели StackPanel размещено несколько кнопок и фрейм Frame, содержащий веб-страницу (с ограничением размера 100x100). Когда вся панель поворачивается и масштабируется, фрейм честно пытается выполнить масштабирование, но не поворачивается. В результате большая часть повернутых кнопок оказывается перекрыта.


Панель StackPanel до преобразования


StackPanel после масштабирования и поворота