Декорування та шаблонізація елементів управління

Питання 4.5.

Література

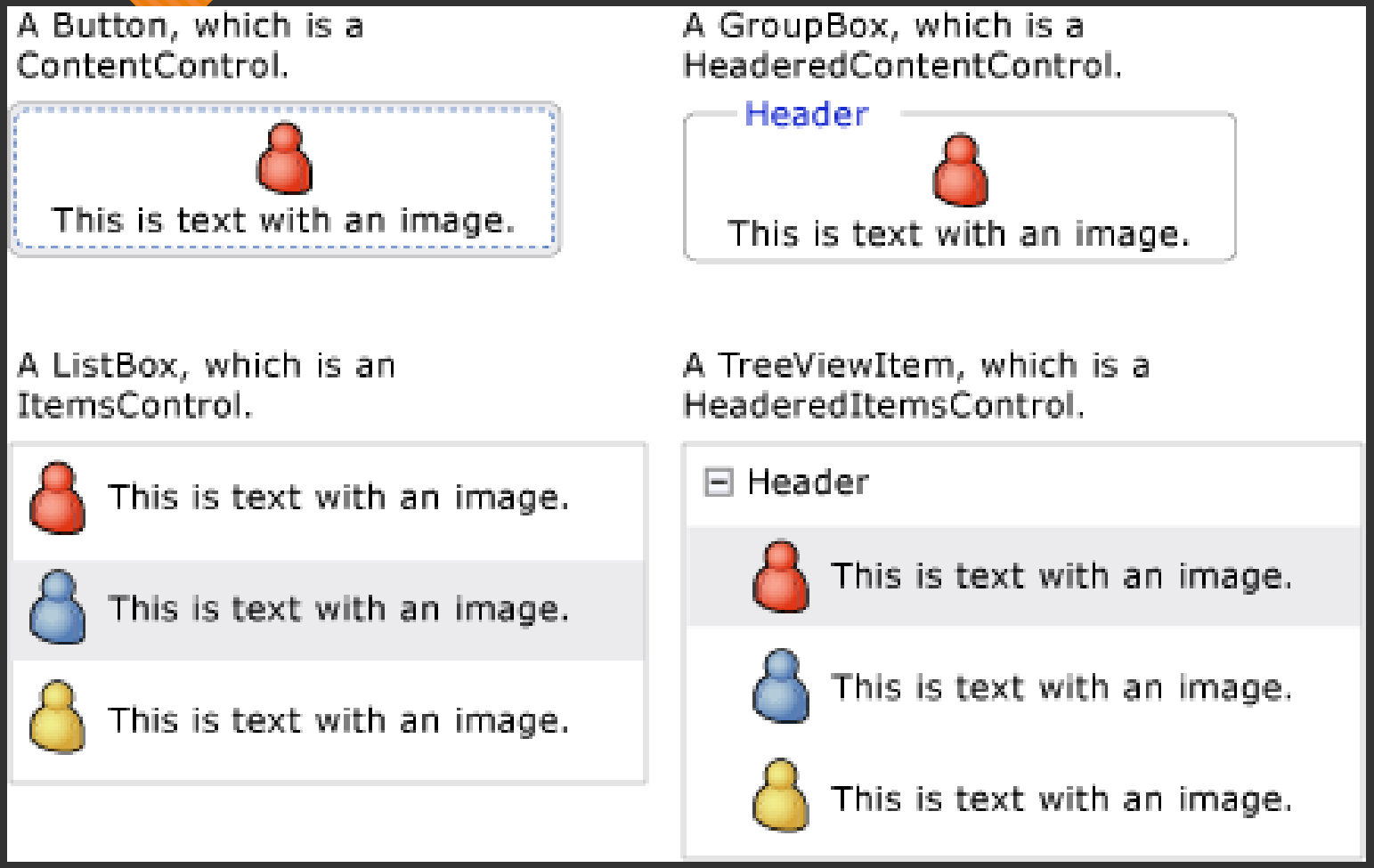

Sheridan Yuen
Mastering
Windows
Presentation
Foundation
Master the art of building modern desktop applications on Windows
Packt

Контентна модель WPF

- Деякі елементи управління можуть містити об'єкт довільного типу, зокрема string, DateTime або UIElement, що є контейнером для додаткових елементів.
  - Наприклад, Button може містити зображення та певний текст; CheckBox - значення DateTime.Now тощо.

Клас-контейнер для довільного вмісту	Вміст
ContentControl	Один довільний об'єкт.
HeaderedContentControl	Заголовок та один елемент, обидва – довільні об'єкти.
ItemsControl	Колекція довільних об'єктів.
HeaderedItemsControl	Заголовок та колекція довільних об'єктів.

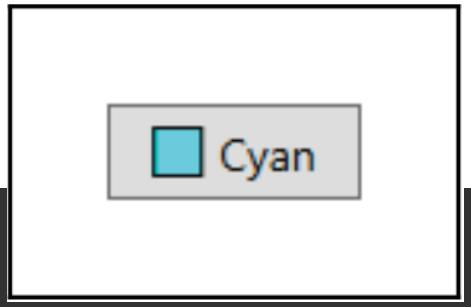
Контентна модель WPF



Елементи управління Content

- Використовуються нечасто, в основному, для рендерингу одного елемента даних у конкретному шаблоні.
  - Часто використовується ContentControl для відображення View Models та об'єкт DataTemplate, який рендерить пов'язаний з ним View.
  - Можна використовувати деяку форму ItemsControl, щоб показувати групу елементів та ContentControl – для обраного елемента.

```
<Button Width="80" Height="30" TextElement.FontSize="14">
  <StackPanel Orientation="Horizontal">
    <Rectangle Fill="Cyan" Stroke="Black" StrokeThickness="1" Width="16"
      Height="16" />
    <TextBlock Text="Cyan" Margin="5,0,0,0" />
  </StackPanel>
</Button>
```



- Можемо задати вміст за допомогою властивості Content.
  - Проте клас ContentControl задає властивість Content в атрибуті ContentPropertyAttribute, що дозволяє визначити вміст шляхом додавання дочірнього елементу до контрола.
- Якщо вміст має рядковий тип, можемо використовувати властивість ContentStringFormat з можливістю форматування.
  - В іншому випадку можна використати властивість ContentTemplate, щоб задати DataTemplate для використання під час рендерингу контенту.
  - Альтернатива: властивість ContentTemplateSelector типу DataTemplateSelector також дозволяє обирати DataTemplate, проте на основі деякої власно заданої умови.
  - Усі породжені класи мають доступ до цих властивостей, щоб сформувати виведення свого контенту.

Презентування контенту

- Клас ContentPresenter, по суті, презентує контент.
  - Єдине місце для оголошення – всередині ControlTemplate об'єкта ContentControl або одного з породжених класів.
  - Тоді оголошуємо їх на місці появи бажаного контенту.
- Визначення властивості TargetType для ControlTemplate при використанні ContentPresenter призведе до того, що її властивість Content неявно прив'яжеться до властивості Content відповідного елементу ContentControl.
  - Проте можна прив'язати дані явно до бажаного елементу.

```
<ControlTemplate x:Key="ButtonTemplate" TargetType="{x:Type Button}">
  <ContentPresenter Content="{TemplateBinding ToolTip}" />
</ControlTemplate>
```

- Якщо присвоїти властивості значення Header, WPF розгляне властивість Header об'єкта ContentControl, щоб неявно прив'язати дані до властивості Content презенттера.
  - Аналогічно шукатиме властивості HeaderTemplate та HeaderTemplateSelector, щоб неявно прив'язатись до властивостей ContentTemplate та ContentTemplateSelector.
  - Основне використання – в ControlTemplate для елементу HeaderedContentControl або породженого від нього класу.

```
<ControlTemplate x:Key="TabItemTemplate" TargetType="{x:Type TabItem}">
  <StackPanel>
    <ContentPresenter ContentSource="Header" />
    <ContentPresenter ContentSource="Content" />
  </StackPanel>
</ControlTemplate>
```

- Існують правила того, що відображатиме ContentPresenter.
  - Якщо властивості ContentTemplate або ContentTemplateSelector задано, визначений властивістю Content об'єкт даних матиме застосований до нього resulting data template.
  - Аналогічно при знаходженні data template потрібного типу в області видимості елементу ContentPresenter, it will be applied.
  - Якщо об'єкт-контент є UI-елементом або він повертається конвертером типів, елемент відображається напряму.
  - Якщо об'єкт є рядком або рядок повертається від конвертера типів, його буде присвоєно властивості Text елементу TextBlock з подальшим відображенням на екрані.
  - Решта об'єктів мають метод ToString(), що застосовується до них та результат виводиться в елементі TextBlock.

Items controls

- ItemsControl містить змінну кількість елементів ContentPresenter та дозволяє відображати колекцію елементів (items).
  - Це базовий клас для найпоширеніших collection controls, зокрема ListBox, ComboBox та TreeView.
  - Якщо потрібно просто вивести на екран набір елементів колекції, краще використовувати ItemsControl замість них (краща ефективність).

Застосування MVVM

- Зазвичай прив'язується колекція, яка реалізує інтерфейс IEnumerable, від рівня View Model до властивості ItemsControl.ItemsSource.
  - Також доступна властивість Items, яка відобразить елемента прив'язаної колекції.
  - Обидві властивості використовуються для виведення елементів колекції на екран, проте тільки одного виду в межах контексту (або тільки ItemsSource, або лише Items).
  - Інакше елементи колекції стануть read-only.
- Для виведення колекції, яка не реалізує IEnumerable потрібно додавати її елементи за допомогою властивості Items.
  - Властивість Items неявно застосовується при оголошуванні елементів колекції в якості вмісту елементу ItemsControl у XAML.
  - Проте за використання MVVM частіше береться властивість ItemsSource.

Відображення елементів в ItemsControl

- Кожен елемент колекції неявно огортається контейнерним елементом ContentPresenter.
  - Тип контейнерного елементу залежить від типу використаного collection control (ComboBox → в елементи ComboBoxItem).
- Властивості ItemContainerStyle та ItemContainerStyleSelector дозволяють стилізувати контейнерні елементи.
  - Необхідно, щоб представлені стилі були націлені на коректний тип контейнерного елементу управління.
  - Наприклад, для ListBox типом цільового стилю має бути ListBoxItem.

Властивості ContentTemplate та ContentTemplateSelector

- Аналогічні властивостям класу ContentControl і також дозволяють обирати DataTemplate на основі custom condition.
  - Як і властивість Content, вони неявно прив'язують до властивостей з тими ж назвами в templated parent, якщо задано властивість TargetType з ControlTemplate.
  - Для декількох елементів управління назви відповідних властивостей не співпадають.
  - Тоді можна використовувати властивість ContentSource as a shortcut для прив'язки властивостей Content, ContentTemplate та ContentTemplateSelector.

Властивість ItemsPanel класу ItemsControl

- Має тип ItemsPanelTemplate, який дозволяє змінювати тип панелі, яку використовує collection control для макету.
  - Для кастомізації шаблону ItemsControl маємо 2 варіанти залежно від способу рендерингу дочірніх елементів контрола.

```
<ControlTemplate x:Key="Template1" TargetType="{x:Type ItemsControl}">
  <StackPanel Orientation="Horizontal" IsItemsHost="True" />
</ControlTemplate>
```

  - Альтернатива: оголосимо елемент ItemsPresenter, який задає, куди буде відрендерено відповідні елементи.
  - Цей елемент буде замінятись в процесі роботи програми на actual items panel.

```
<ControlTemplate x:Key="Template2" TargetType="{x:Type ItemsControl}">
  <ItemsPresenter />
</ControlTemplate>
```

Властивість AlternationCount

```
<ListBox ItemsSource="{Binding Users}" AlternationCount="3">
  <ListBox.ItemContainerStyle>
    <Style TargetType="{x:Type ListBoxItem}">
      <Setter Property="FontSize" Value="14" />
      <Setter Property="Foreground" Value="White" />
      <Setter Property="Padding" Value="5" />
      <Style.Triggers>
        <Trigger Property="ListBox.AlternationIndex" Value="0">
          <Setter Property="Background" Value="Red" />
        </Trigger>
        <Trigger Property="ListBox.AlternationIndex" Value="1">
          <Setter Property="Background" Value="Green" />
        </Trigger>
        <Trigger Property="ListBox.AlternationIndex" Value="2">
          <Setter Property="Background" Value="Blue" />
        </Trigger>
      </Style.Triggers>
    </Style>
  </ListBox.ItemContainerStyle>
</ListBox>
```

Дозволяє по-різному стилізувати альтернативні контейнери.

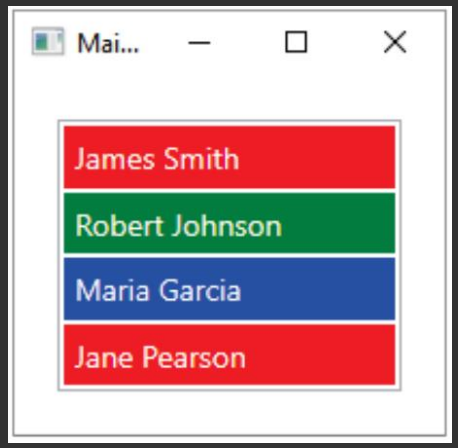
- Задаємо трійку для AlternationCount, щоб мати 3 різні стилі items, надалі цей шаблон буде повторюватись.
- Створюємо стиль для item containers за допомогою властивості ItemContainerStyle.

Продовження коду

```
<ListBox ItemsSource="{Binding Users}" AlternationCount="3">
  <ListBox.Resources>
    <AlternationConverter x:Key="BackgroundConverter">
      <SolidColorBrush>Red</SolidColorBrush>
      <SolidColorBrush>Green</SolidColorBrush>
      <SolidColorBrush>Blue</SolidColorBrush>
    </AlternationConverter>
  </ListBox.Resources>
  <ListBox.ItemContainerStyle>
    <Style TargetType="{x:Type ListBoxItem}">
      <Setter Property="FontSize" Value="14" />
      <Setter Property="Foreground" Value="White" />
      <Setter Property="Padding" Value="5" />
      <Setter Property="Background"
        Value="{Binding(ItemsControl.AlternationIndex),
        RelativeSource={RelativeSource Self},
        Converter={StaticResource BackgroundConverter}}" />
    </Style>
  </ListBox.ItemContainerStyle>
</ListBox>
```

Як альтернативу маємо оголошений екземпляр AlternationConverter для кожної властивості, що підлягає змінам.

Ці властивості прив'язуємо до властивості AlternationIndex та конвертера.



Властивість GroupStyle класу ItemsControl

- Дозволяє відображати дочірні items у групах. Для їх групування в UI потрібно виконати кілька кроків:
  - 1) Прив'язати CollectionViewSource з одним або кількома елементами PropertyGroupDescription до представленої раніше колекції Users.
  - 2) Встановимо її як значення ItemControl.ItemsSource, а потім задамо GroupStyle.
  - Приклад: об'єкт CollectionViewSource, оголошений в секції Resources:

```
<CollectionViewSource x:Key="GroupedUsers" Source="{Binding Users}">
  <CollectionViewSource.GroupDescriptions>
    <PropertyGroupDescription PropertyName="Name"
      Converter="{StaticResource StringToFirstLetterConverter}" />
  </CollectionViewSource.GroupDescriptions>
</CollectionViewSource>
```

Властивість GroupStyle класу ItemsControl

```
using System;
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace CompanyName.ApplicationName.Converters
{
  [ValueConversion(typeof(string), typeof(string))]
  public class StringToFirstLetterConverter : IValueConverter
  {
    public object Convert(object value, Type targetType, object parameter,
      CultureInfo culture)
    {
      if (value == null) return DependencyProperty.UnsetValue;
      return value.ToString()[0];
    }

    public object ConvertBack(object value, Type targetType,
      object parameter, CultureInfo culture)
    {
      return DependencyProperty.UnsetValue;
    }
  }
}
```

У даному випадку маємо тільки декілька об'єктів User, тому групування за назвою не призведе до утворення груп.

- Додамо конвертер, що повертатиме першу літеру кожної назви у відповідну групу за допомогою властивості Converter.
- У конвертері задаємо типи даних, які задіяні в реалізації конвертера в атрибуті ValueConversion.
- У методі Convert() перевіряємо валідність вхідного параметра input та повертаємо значення DependencyProperty.UnsetValue, якщо воно дорівнює null.

Продовження коду

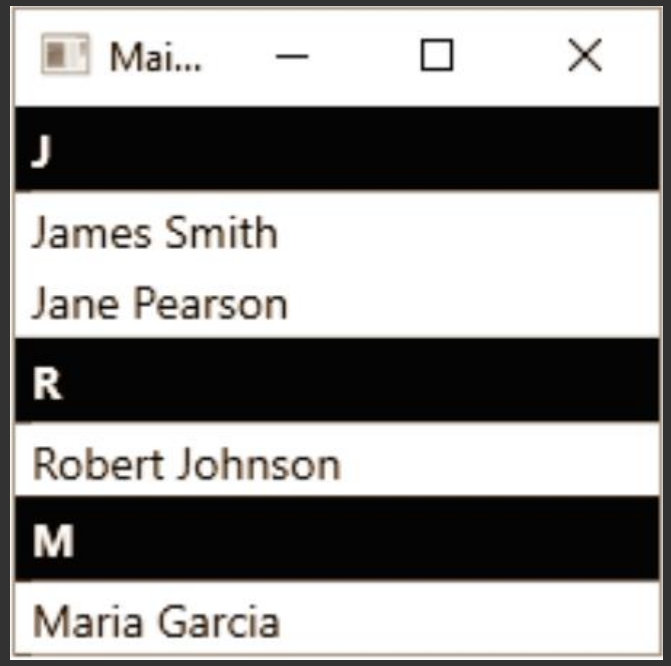
```
<ItemsControl ItemsSource="{Binding Source={StaticResource GroupedUsers}}"
  FontSize="14">
  <ItemsControl.GroupStyle>
    <GroupStyle>
      <GroupStyle.HeaderTemplate>
        <DataTemplate>
          <TextBlock Text="{Binding Name,
            Converter={StaticResource StringToFirstLetterConverter}}"
            Background="Black" Foreground="White" FontWeight="Bold"
            Padding="5,4" />
        </DataTemplate>
      </GroupStyle.HeaderTemplate>
    </GroupStyle>
  </ItemsControl.GroupStyle>
  <ItemsControl.ItemTemplate>
    <DataTemplate DataType="{x:Type DataModels:User}">
     <TextBlock Text="{Binding Name}" Background="White" Foreground="Black"
       Padding="0,2" />
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
```

As we do not need, or would not be able to convert anything back using this converter, the ConvertBack method simply returns the DependencyProperty.UnsetValue value.

By attaching this converter to the PropertyGroupDescription element, we are now able to group by the first letter of each name.

Пояснення до коду

- Для доступу до об'єкту GroupedUsers (з секції Resources) типу CollectionViewSource необхідно використовувати властивість Binding.Source.
  - Далі оголошуємо шаблон даних, що визначатиме, як виглядатиме заголовок кожної групи, у властивості HeaderTemplate.
  - Використовуємо екземпляр StringToFirstLetterConverter, який також оголошувався у відповідній колекції ресурсів, та задаємо кілька базових стильових властивостей.
  - Потім визначаємо другий шаблон даних, який встановлює, як буде виглядати кожен item у кожній групі.



Декоративні елементи (adorners) в WPF

- Декоративний елемент – це вид класів, які відрисовуються поверх усіх UI-елементів на спеціальному декоративному прошарку (adorner layer).
  - Рендеринг не залежить від налаштувань Panel.ZIndex.
  - Кожен декоративний елемент прив'язаний до елементу типу UIElement та рендериться незалежно та відносно позиції and independently rendered in a position that is relative to the adorned element.
- The purpose of the adorner is to provide certain visual cues to the application user.
  - For example, we could use an adorner to display a visual representation of UI elements that are being dragged in a drag and drop operation.
  - Alternatively, we could use an adorner to add "handles" to a UI control to enable users to resize the element.
- As the adorner is added to the adorner layer, it is the adorner layer that is the parent of the adorner, rather than the adorned element.
  - In order to create a custom adorner, we need to declare a class that extends the Adorner class.

- When creating a custom adorner, we need to be aware that we are responsible for writing the code to render its visuals.
  - However, there are a few different ways to construct our adorner graphics; we can use the OnRender or OnRenderSizeChanged methods and a drawing context to draw basic lines and shapes, or we can use the ArrangeOverride method to arrange .NET controls.
- Adorners receive events like other .NET controls, although if we don't need to handle them, we can arrange for them to be passed straight through to the adorned element.
  - In these cases, we can set the IsHitTestVisible property to false and this will enable pass-through hit-testing of the adorned element.

Приклад of a resizing adorner that lets us resize shapes on a canvas

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Controls.Primitives;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;

namespace CompanyName.ApplicationName.Views.Adorners
{
  public class ResizeAdorner : Adorner
  {
    private VisualCollection visualChildren;
    private Thumb top, left, bottom, right;

    public ResizeAdorner(UIElement adornedElement) : base(adornedElement)
    {
      visualChildren = new VisualCollection(this);
      top = InitializeThumb(Cursors.SizeNS, Top_DragDelta);
      left = InitializeThumb(Cursors.SizeWE, Left_DragDelta);
      bottom = InitializeThumb(Cursors.SizeNS, Bottom_DragDelta);
      right = InitializeThumb(Cursors.SizeWE, Right_DragDelta);
    }
```

```
private Thumb InitializeThumb(Cursor cursor,
  DragDeltaEventHandler eventHandler)
{
  Thumb thumb = new Thumb();
  thumb.BorderBrush = Brushes.Black;
  thumb.BorderThickness = new Thickness(1);
  thumb.Cursor = cursor;
  thumb.DragDelta += eventHandler;
  thumb.Height = thumb.Width = 6.0;
  visualChildren.Add(thumb);
  return thumb;
}

private void Top_DragDelta(object sender, DragDeltaEventArgs e)
{
  FrameworkElement adornedElement = (FrameworkElement)AdornedElement;
  adornedElement.Height =
    Math.Max(adornedElement.Height - e.VerticalChange, 6);
  Canvas.SetTop(adornedElement,
    Canvas.GetTop(adornedElement) + e.VerticalChange);
}

private void Left_DragDelta(object sender, DragDeltaEventArgs e)
{
  FrameworkElement adornedElement = (FrameworkElement)AdornedElement;
  adornedElement.Width =
    Math.Max(adornedElement.Width - e.HorizontalChange, 6);
  Canvas.SetLeft(adornedElement,
    Canvas.GetLeft(adornedElement) + e.HorizontalChange);
}
```

Продовження коду

```
private void Bottom_DragDelta(object sender, DragDeltaEventArgs e)
{
  FrameworkElement adornedElement = (FrameworkElement)AdornedElement;
  adornedElement.Height =
    Math.Max(adornedElement.Height + e.VerticalChange, 6);
}

private void Right_DragDelta(object sender, DragDeltaEventArgs e)
{
  FrameworkElement adornedElement = (FrameworkElement)AdornedElement;
  adornedElement.Width =
    Math.Max(adornedElement.Width + e.HorizontalChange, 6);
}

protected override void OnRender(DrawingContext drawingContext)
{
  SolidColorBrush brush = new SolidColorBrush(Colors.Transparent);
  Pen pen = new Pen(new SolidColorBrush(Colors.DeepSkyBlue), 1.0);
  drawingContext.DrawRectangle(brush, pen,
    new Rect(-2, -2, AdornedElement.DesiredSize.Width + 4,
    AdornedElement.DesiredSize.Height + 4));
}
```

```
    protected override Size ArrangeOverride(Size finalSize)
    {
      top.Arrange(
        new Rect(AdornedElement.DesiredSize.Width / 2 - 3, -8, 6, 6));
      left.Arrange(
         new Rect(-8, AdornedElement.DesiredSize.Height / 2 - 3, 6, 6));
       bottom.Arrange(new Rect(AdornedElement.DesiredSize.Width / 2 - 3,
         AdornedElement.DesiredSize.Height + 2, 6, 6));
       right.Arrange(new Rect(AdornedElement.DesiredSize.Width + 2,
         AdornedElement.DesiredSize.Height / 2 - 3, 6, 6));
       return finalSize;

     }

     protected override int VisualChildrenCount
     {
       get { return visualChildren.Count; }
     }

     protected override Visual GetVisualChild(int index)
     {
       return visualChildren[index];
     }
   }
}
```

Використання класу

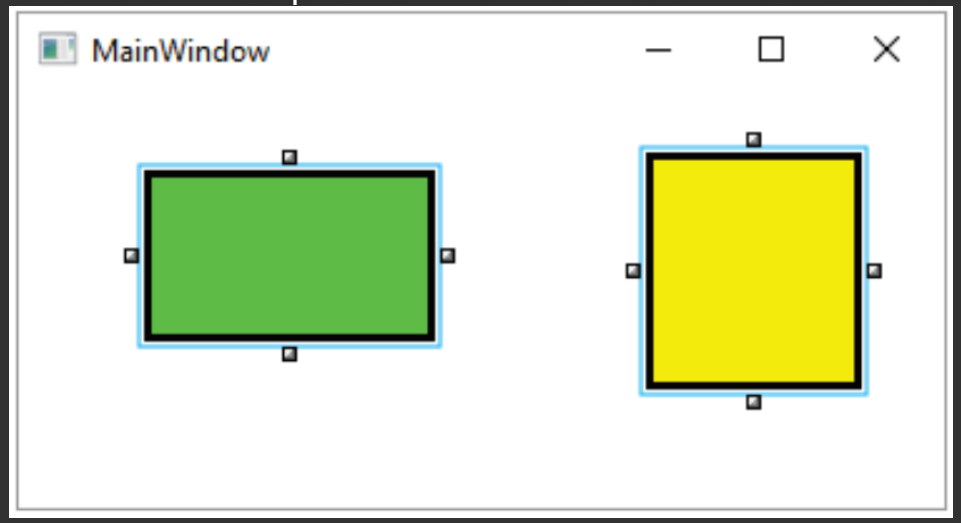
- Adorners need to be initialized in code and so, a good place to do this is in the UserControl.Loaded method, when we can be certain that the canvas and its items will have been initialized.
  - Note that as adorners are purely UI related, initializing them in the control's code behind does not present any conflict when using MVVM.

```
Loaded += View_Loaded;

...

private void View_Loaded(object sender, RoutedEventArgs e)
{
  AdornerLayer adornerLayer = AdornerLayer.GetAdornerLayer(Canvas);
  foreach (UIElement uiElement in Canvas.Children)
  {
    adornerLayer.Add(new ResizeAdorner(uiElement));
  }
}
```

Використання класу

```
<Canvas Name="Canvas">
  <Rectangle Canvas.Top="50" Canvas.Left="50" Fill="Lime"
    Stroke="Black" StrokeThickness="3" Width="150" Height="50" />
  <Rectangle Canvas.Top="25" Canvas.Left="250" Fill="Yellow"
    Stroke="Black" StrokeThickness="3" Width="50" Height="150" />
</Canvas>
```

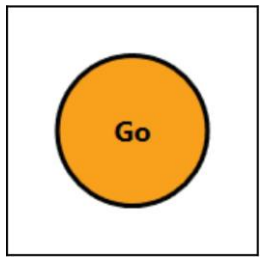
- we just need a Canvas panel named Canvas and some shapes to resize.



Шаблонізація елементів управління

- Всі UI-елементи, породжені від класу Control, постачають доступ до їх властивості Template.
  - Властивість типу ControlTemplate та дозволяє повністю замінити стандартний початковий шаблон.

```
<Button Content="Go" Width="100" HorizontalAlignment="Center">
  <Button.Template>
    <ControlTemplate TargetType="{x:Type Button}">
      <Grid>
        <Ellipse Fill="Orange" Stroke="Black" StrokeThickness="3"
          Height="{Binding ActualWidth,
          RelativeSource={RelativeSource Self}}" />
        <ContentPresenter HorizontalAlignment="Center"
          VerticalAlignment="Center" TextElement.FontSize="18"
          TextElement.FontWeight="Bold" />
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```



Шаблонізація елементів управління

- На відміну від стилів, при оголошенні ControlTemplate та встановленні властивості TargetType у колекції Resources без директиви x:Key, вона не буде неявно застосована до всіх кнопок у додатку.
  - Помилка компіляції: ***Each dictionary entry must have an associated key***.
  - Замість цього потрібно задати директиву x:Key явно для властивості Template елементу управління зі стилю.
  - If we want our template to be applied to every control of that type then we need to set it in the default style for that type.

```
<ControlTemplate x:Key="ButtonTemplate" TargetType="{x:Type Button}">
  ...
</ControlTemplate>
<Style TargetType="{x:Type Button}">
  <Setter Property="Template" Value="{StaticResource ButtonTemplate}" />
</Style>
```

Значання властивостей зазвичай не задаються жорстко

- Поки не потрібно можливості для користувачів задавати власні кольори on our templated controls.

```
<Button Content="Go" Width="100" HorizontalAlignment="Center"
  Background="Orange" HorizontalContentAlignment="Center"
  VerticalContentAlignment="Center" FontSize="18">
  <Button.Template>
    <ControlTemplate TargetType="{x:Type Button}">
      <Grid>
        <Ellipse Fill="{TemplateBinding Background}"
          Stroke="{TemplateBinding Foreground}" StrokeThickness="3"
          Height="{Binding ActualWidth,
          RelativeSource={RelativeSource Self}}" />
        <ContentPresenter HorizontalAlignment="{TemplateBinding
          HorizontalContentAlignment}"
          VerticalAlignment="{TemplateBinding
          VerticalContentAlignment}"
          TextElement.FontWeight="{TemplateBinding FontWeight}"
          TextElement.FontSize="{TemplateBinding FontSize}" />
      </Grid>
    </ControlTemplate>
  </Button.Template>
</Button>
```

Значання властивостей зазвичай не задаються жорстко

- Тепер жорстко задані значення застосовуються на власне button control, за виключенням властивості StrokeThickness.
  - Для класу Button немає властивості, яку можна використати для розкриття цієї внутрішньої властивості елемента управління.
  - Якби це було проблемою, можливо було б розкрити значення властивості у власній прикріпленій властивості та прив'язатись до нього на кнопці:

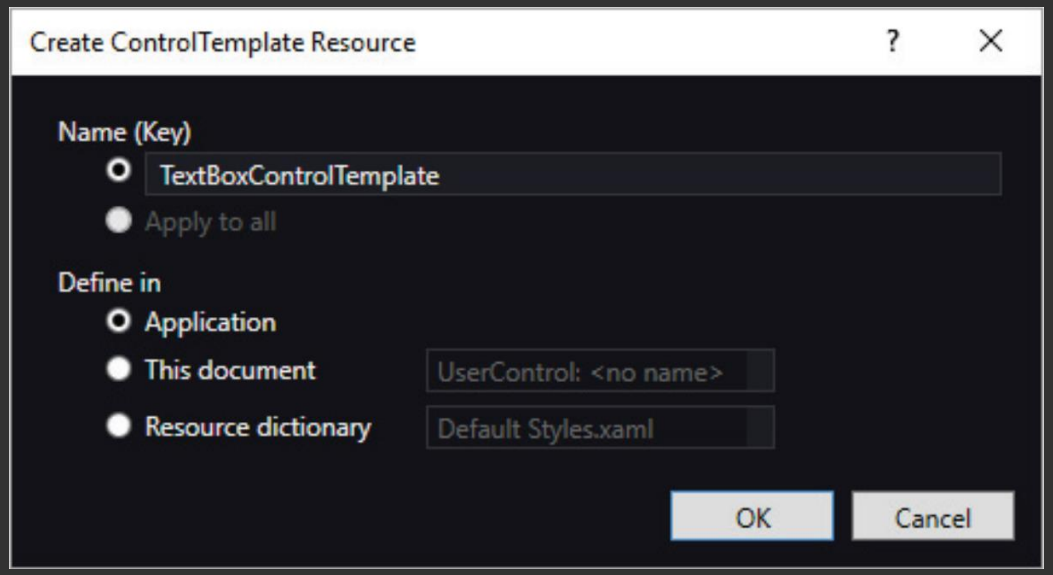
```
<Button Attached:ButtonProperties.StrokeThickness="3" ... />
```

- Можемо зробити так всередині control template:

```
<Ellipse
  Stroke="{Binding (Attached:ButtonProperties.StrokeThickness)}" ... />
```

Можливо, потрібно трохи скоригувати original template

- Оберіть елемент управління та натисніть F4, щоб відкрити вікно Properties.
  - У категорії Miscellaneous знайдіть властивість Template.
  - Натиснувши на маленький квадрат справа оберіть в меню Convert to New Resource…



Приклад шаблону для TextBox

```
<ControlTemplate TargetType="{x:Type TextBox}">
  <Border Name="border" BorderBrush="{TemplateBinding BorderBrush}"
    BorderThickness="{TemplateBinding BorderThickness}"
    Background="{TemplateBinding Background}"
    SnapsToDevicePixels="True">
    <ScrollViewer Name="PART_ContentHost" Focusable="False"
      HorizontalScrollBarVisibility="Hidden"
      VerticalScrollBarVisibility="Hidden" />
  </Border>
  <ControlTemplate.Triggers>
    <Trigger Property="IsEnabled" Value="False">
      <Setter Property="Opacity" TargetName="border" Value="0.56" />
    </Trigger>
    <Trigger Property="IsMouseOver" Value="True">
      <Setter Property="BorderBrush" TargetName="border"
        Value="#FF7EB4EA" />
    </Trigger>
    <Trigger Property="IsKeyboardFocused" Value="True">
      <Setter Property="BorderBrush" TargetName="border"
        Value="#FF569DE5" />
    </Trigger>
  </ControlTemplate.Triggers>
</ControlTemplate>
```

Приклад шаблону для TextBox

- Всередині елементу Border присутній ScrollViewer з назвою PART_ContentHost.
  - Префікс PART_ вказує, що цей елемент управління потрібен в шаблоні.
  - При ініціалізації текстбокса програмно додаються об’єкти TextBoxView та CaretElement в об’єкт ScrollViewer.
  - Такі спеціально іменовані елементи також потрібно реєструвати в оголошуючому класі.
  - Включення цих імнованих елементів управління у власні шаблони дозволяє підтримувати існуючу функціональність.

Приклад: важко, але працює

```
<TextBox Text="Hidden Text Box">
  <TextBox.Template>
    <ControlTemplate TargetType="{x:Type TextBox}">
      <ContentPresenter Content="{TemplateBinding Text}" />
    </ControlTemplate>
  </TextBox.Template>
</TextBox>
```

- Хоч текстбокс виведе задане текстове значення, у нього не буде внутрішнього бокса, як у звичайного.
  - Елемент ContentPresenter побачить рядок та за замовчуванням виведе його в елементі TextBlock.
  - Його властивість Text все ще прив'язана до властивості Text нашого текстбоксу, тому при фокусуванні він поводитиметься як і звичайний текстбокс.
  - Момент фокусування не видно, оскільки не додано відповідних тригерів.

Прикріплення властивостей

Створимо кнопку, яка дозволятиме задати друге tooltip message, що відображатиметься при control is disabled.

```
using System.Windows;
using System.Windows.Controls;

namespace CompanyName.ApplicationName.Views.Attached
{
  public class ButtonProperties : DependencyObject
  {
    private static readonly DependencyPropertyKey
      originalToolTipPropertyKey =
      DependencyProperty.RegisterAttachedReadOnly("OriginalToolTip",
      typeof(string), typeof(ButtonProperties),
      new FrameworkPropertyMetadata(default(string)));

    public static readonly DependencyProperty OriginalToolTipProperty =
      originalToolTipPropertyKey.DependencyProperty;

    public static string GetOriginalToolTip(
      DependencyObject dependencyObject)
    {
      return
        (string)dependencyObject.GetValue(OriginalToolTipProperty);
    }
```

Продовження коду

```
public static DependencyProperty DisabledToolTipProperty =
  DependencyProperty.RegisterAttached("DisabledToolTip",
  typeof(string), typeof(ButtonProperties),
  new UIPropertyMetadata(string.Empty, OnDisabledToolTipChanged));

public static string GetDisabledToolTip(
  DependencyObject dependencyObject)
{
  return (string)dependencyObject.GetValue(
    DisabledToolTipProperty);
}

public static void SetDisabledToolTip(
  DependencyObject dependencyObject, string value)
{
  dependencyObject.SetValue(DisabledToolTipProperty, value);
}
```

Продовження коду

```
    public static void OnDisabledToolTipChanged(DependencyObject
      dependencyObject, DependencyPropertyChangedEventArgs e)
    {
      Button button = dependencyObject as Button;
      ToolTipService.SetShowOnDisabled(button, true);
      if (e.OldValue == null && e.NewValue != null)
        button.IsEnabledChanged += Button_IsEnabledChanged;
      else if (e.OldValue != null && e.NewValue == null)
        button.IsEnabledChanged -= Button_IsEnabledChanged;
    }

    private static void Button_IsEnabledChanged(object sender,
      DependencyPropertyChangedEventArgs e)
    {
      Button button = sender as Button;
      if (GetOriginalToolTip(button) == null)
        button.SetValue(originalToolTipPropertyKey,
        button.ToolTip.ToString());
      button.ToolTip = (bool)e.NewValue ?
        GetOriginalToolTip(button) : GetDisabledToolTip(button);
    }
  }
}
```

Прикріплення властивостей

- Насамкінець, застосовуємо значення властивості e.NewValue, щоб визначити, задавати оригінальний чи disabled tooltip для властивості ToolTip елемента управління.
  - Якщо елемент управління задіяний, значення e.NewValue буде true, а оригінальний tooltip буде повернуто.
  - Якщо кнопка is disabled, відображатиметься disabled tooltip.
  - Можемо використати прикріплену властивість:

```
<Button Content="Save" Attached:ButtonProperties.DisabledToolTip="You must
  correct validation errors before saving" ToolTip="Saves the user" />
```

Комбінування елементів управління

- Для впорядкування кількох існуючих елементів управління деяким чином зазвичай використовують об'єкт UserControl.
  - Проте для побудови reusable control, зокрема address control, намагаємось відокремити these від наших Views, оголошуючи їх у папці Controls та просторі імен всередині проекту.
- При оголошенні таких елементів управління зазвичай визначають властивості залежності у code behind.
  - Поки немає business-related functionality в елементі управління, нормально використовувати code behind для обробки подій.
  - Якщо елемент управління business-related, можна використовувати View Model.

address control


```
<UserControl x:Class=
  "CompanyName.ApplicationName.Views.Controls.AddressControl"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
  xmlns:Controls=
    "clr-namespace:CompanyName.ApplicationName.Views.Controls"
  mc:Ignorable="d" d:DesignHeight="300" d:DesignWidth="300">
  <Grid>
    <Grid.ColumnDefinitions>
      <ColumnDefinition Width="Auto" SharedSizeGroup="Label" />
      <ColumnDefinition />
    </Grid.ColumnDefinitions>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
      <RowDefinition Height="Auto" />
    </Grid.RowDefinitions>
```

Продовження коду

```
    <TextBlock Text="House/Street" />
    <TextBox Grid.Column="1" Text="{Binding Address.HouseAndStreet,
      RelativeSource={RelativeSource
      AncestorType={x:Type Controls:AddressControl}}}" />
    <TextBlock Grid.Row="1" Text="Town" />
    <TextBox Grid.Row="1" Grid.Column="1"
      Text="{Binding Address.Town, RelativeSource={RelativeSource
      AncestorType={x:Type Controls:AddressControl}}}" />
    <TextBlock Grid.Row="2" Text="City" />
    <TextBox Grid.Row="2" Grid.Column="1"
      Text="{Binding Address.City, RelativeSource={RelativeSource
      AncestorType={x:Type Controls:AddressControl}}}" />
    <TextBlock Grid.Row="3" Text="Post Code" />
    <TextBox Grid.Row="3" Grid.Column="1"
      Text="{Binding Address.PostCode, RelativeSource={RelativeSource
      AncestorType={x:Type Controls:AddressControl}}}" />
    <TextBlock Grid.Row="4" Text="Country" />
    <TextBox Grid.Row="4" Grid.Column="1"
      Text="{Binding Address.Country, RelativeSource={RelativeSource
      AncestorType={x:Type Controls:AddressControl}}}" />
  </Grid>
</UserControl>
```

- Останні запитання:
  - Комбінування елементів управління (ст. 248-252)
  - Створення власних елементів управління (ст. 252-259)

Дякую за увагу!