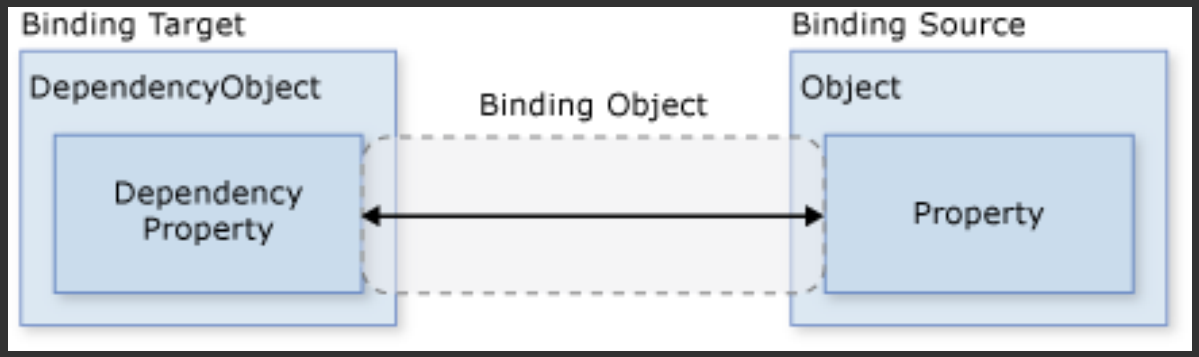
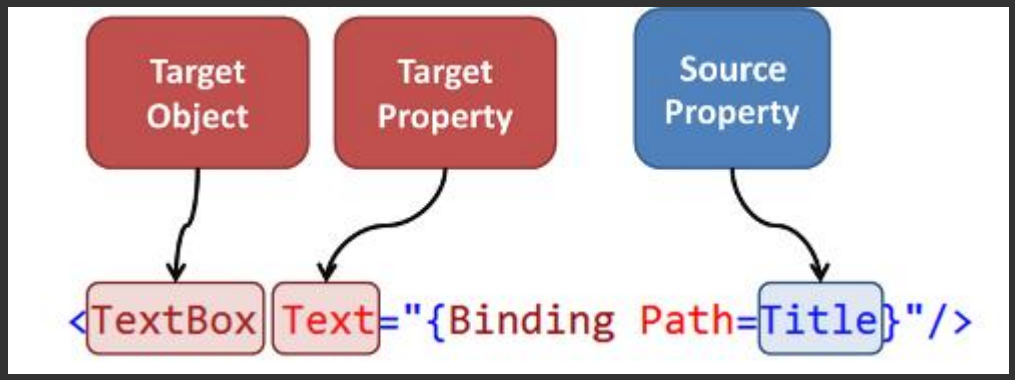
Технологія прив’язування даних

Питання 4.2.

Технологія прив'язування даних

- Технологія встановлення зв'язку між UI додатку та його бізнес-логікою з метою коректної синхронізації даних між ними.





Технологія прив'язування даних

- Обидві сторони прив'язки повинні забезпечити сповіщення про зміни для іншої сторони.
  - Властивістю джерела прив'язки може бути CLR-властивість або властивість залежності, проте цільова властивість (target property) повинна бути властивістю залежності
- Прив'язка даних зазвичай виконується в XAML за допомогою розширення розмітки {Binding}.
  - Цільова властивість зазвичай походить від елементу управління UI, оскільки ціль прив'язки повинна бути Dependency Property.

Синтаксис Binding path

- Прив'язку можна оголосити
  - in longhand, визначаючи елемент Binding у коді XAML.
  - in shorthand, використовуючи мову розмітки, яка транслюється в Binding-елемент за допомогою XAML.
- Властивість Binding.Path має тип PropertyPath.
  - Шлях прив'язки (binding path) є відносним для джерела прив'язування,
    - зазвичай задається за допомогою властивості DataContext або рядком.
  - Для прив'язки всього джерела прив'язування можна задати елементи:

```
{Binding Path=.}
```

```
{Binding .}
```

```
{Binding}
```

Binding path syntax

- Для прив'язки до більшості property paths використовуємо аналогічний синтаксис.
  - При прив'язці напряму до властивості об'єкта прив'язування можна використовувати назву властивості: {Binding PropertyName}
  - {Binding PropertyName.AnotherPropertyName}
    - Такий підхід відомий як непряме таргетування властивості (**indirect property targeting)**.
  - Коли прив'язуємось до елементу колекції або його властивості, використовується синтаксис оператора доступу за індексом.
    - {Binding CollectionPropertyName[0].PropertyName}

Особливості прив'язки до колекцій

- Існує спеціальний символ «/», який можна використовувати для доступу до обраного елементу будь-коли:
  - {Binding CollectionPropertyName/PropertyName}
  - Прив'язується властивість PropertyName поточного елементу колекції, визначеної властивістю CollectionPropertyName.
  - Якщо потрібно прив'язатись до властивості елементу колекції, коли колекція сама є елементом батьківської колекції, кількаразово використовується «/» в одному шляху прив'язки.
- Приклад

```
<StackPanel>
  <ListBox ItemsSource="{Binding Users}"
    IsSynchronizedWithCurrentItem="True" />
  <TextBlock Text="Selected User's Name:" />
  <TextBlock Text="{Binding Users/Name}" />
</StackPanel>
```

Прив'язка до прикріплених (attached) властивостей

- Необхідно огорнути дужками назви властивості та класу:
  - {Binding (ClassName.PropertyName)}
- Якщо прив'язана властивість кастомна, необхідно включити префікс простору імен XML:
  - {Binding (XmlNamespacePrefix:ClassName.PropertyName)}
- Зазвичай також потрібно задати ціль прив'язки.
  - Загалом це об'єкт, для якого задана прив'язка, або інший UI-елемент, тому часто застосовуються властивості RelativeSource або ElementName.

```
{Binding Path=(Attached:TextBoxProperties.Label),
  RelativeSource={RelativeSource AncestorType={x:Type TextBox}}}
```

Символи, яких слід уникати у шляху, прописаному в PropertyPath

- Символ «}».
  - Символ «\» слід екранувати: «\\».
  - Інші 2 символи: «=» та «,».
- Спеціальний символ використовується для екранування символу всередині indexer binding expression.
  - Замість символу «\» використовується the caret character (^).
- При явному оголошенні прив'язок у XAML необхідно екранувати «&» та «>», заміняючи їх на XML Entity forms - & та > відповідно.

Огляд класу Binding

- Використовує lower-level клас BindingException, який підтримує зв'язок між джерелом та ціллю прив'язки.
  - Використовуючи MVVM, розробники зазвичай не звертаються до внутрішнього класу, оскільки намагаються утримати функціональність на рівні ViewModel-ей.
  - Проте при створенні власних елементів управління може бути потрібно програмно оновлювати джерело прив'язки.
- У .NET 4.5 додано властивість ***Delay*** до класу Binding.
  - Дозволяє задати час (мс) для відкладання оновлення джерела прив'язки після змін у binding target property value.
  - Корисно при важких обчисленнях чи подібній обробці користувацького вводу в textbox.

```
<TextBox Text="{Binding Description,
  UpdateSourceTrigger=PropertyChanged, Delay=400}" />
```

Інші властивості класу Binding. Властивість FallbackValue

- Властивість FallbackValue сприяє продуктивності додатку.
- Для повернення значення від кожної прив'язки WPF Framework виконує 4 дії.
  - 1) валідація типу цільової властивості with the data bound value. За умови успіху спробує to resolve the binding path.
  - Інакше намагатиметься знайти конвертер типу, щоб повернути значення.
  - Якщо такого немає If it can't find one, or it returns the DependencyProperty.UnsetValue value, it will then look to see if the FallbackValue property has a value to provide it with.
  - If there is no fallback value, then a lookup is required to find the default value of the target Dependency Property.
- Встановлюючи властивість FallbackValue, можна:
  - 1) Заборонити WPF Framework виконувати пошук значення за замовчуванням для цільової Dependency Property.
  - 2) Усунути вивід трасувальної інформації.

Властивість Binding.TargetNullValue

- Подібна до властивості FallbackValue в тому, що забезпечує певний вивід навіть тоді, коли немає даних з джерела прив'язки.
  - Значення FallbackValue стає виводом, коли прив'язане значення неможливо отримати
  - Значення TargetNullValue використовується тоді, коли успішно прив'язане значення буде null.
- Можна виводити замість 'null' інше значення або повідомлення.
  - Для повідомлення потрібен окремий textbox.

```
<TextBox Text="{Binding Name, TargetNullValue='Please enter your name'}" />
```

Інші властивості класу Binding. Властивість IsAsync

- Для асинхронного доступу до властивостей або тривалого обчислення їх значень потрібно задати the IsAsync method to True on the binding.
  - Усуває блокування інтерфейсу під час очікування.
  - У цей час використовується fallback value (якщо задано, інакше - default value).

```
<Image Source="{Binding InternetSource, IsAsync=True,
  FallbackValue='pack://application:,,,/CompanyName.ApplicationName;
  component/Images/Default.png'}" />
```

Інші властивості класу Binding. Властивість StringFormat

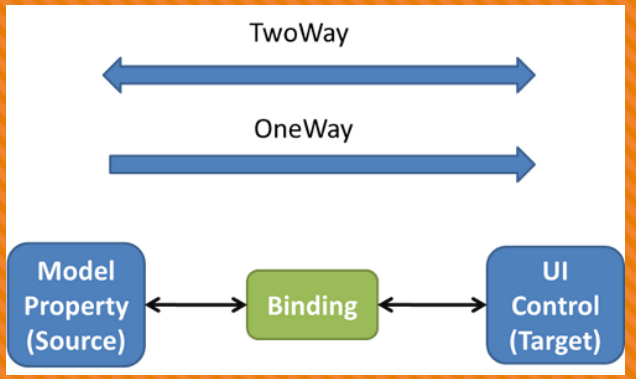
- Використовує метод String.Format, щоб форматувати data bound text output.
- Застереження щодо використання:
  - 1) we can obviously only use a single format item, that represents the single data bound value in a normal binding.
  - 2) необхідно ретельно оголошувати формат, оскільки {…} використовуються в XAML-розмітці.
    - Одне вирішення: одинарні лапки.

```
<TextBlock Text="{Binding Price, StringFormat='{0:C2}'}" />
```

    - Інше вирішення: екранування фігурними дужками.

```
<TextBlock Text="{Binding Price, StringFormat={}{0:C2}}" />
```

Directing data bound traffic



- Напрям обходу даних у кожній прив'язці визначається властивістю Binding.Mode.
- Це перелічення типу BindingMode з кількох значень:
  - ***OneWay*** – дані спрямовано від binding source (однієї View Model) до binding target (UI control).
    - В основному використовується для display only, or read-only purposes та ситуацій, коли значення прив'язаних даних неможливо змінити з UI.
  - ***TwoWay*** – дані вільно рухаються між View Models та UI controls.
    - Найбільш поширений режим прив'язування: зміни від елементів управління відображаються для View Models, і навпаки.
  - ***OneWayToSource*** – обернений до OneWay режим.
    - Режим корисний для захоплення введених даних, коли немає потреби змінювати прив'язані значення.
  - ***OneTime*** – аналог OneWay, проте спрацьовує лише раз.
    - Оновлюватиме відповідне значення при кожному встановленні значення DataContext.
  - ***Default*** - It directs the binding to use the binding mode that was declared from the specified target property.
    - Коли кожна Dependency Property оголошена, можна вказати режим прив'язки (One або **Two-Way) за замовчуванням**. Без вказування буде режим One-Way.

Прив'язка до різних джерел даних

- Загалом джерело прив'язки встановлюється за допомогою властивості FrameworkElement.DataContext.
  - This must be set for a binding to work, although it can be specified in the 1BUI property, or inherited from ancestor controls, so it does not have to be explicitly set.
- Приклад передбачає, що доречне джерело даних коректно задане для батьківського елементу управління:

```
<StackPanel>
  <TextBlock DataContext="{Binding User}" Text="{Binding Name}" />
  <TextBlock DataContext="{Binding User}" Text="{Binding Age}" />
</StackPanel>
```

Прив'язка до різних джерел даних

- Прив'язувати одне джерело до багатьох елементів управління можна простіше:

```
<StackPanel DataContext="{Binding User}">
  <TextBlock Text="{Binding Name}" />
  <TextBlock Text="{Binding Age}" />
</StackPanel>
```

  - Використовуємо можливість успадковування значення властивості DataContext від будь-яких батьківських елементів управління.
  - Фактично, при розробці кожного Window або UserControl зазвичай задають DataContext на таких високорівневих елементах управління, щоб будь-який вкладений контрол мав доступ до того ж джерела прив'язки.

Альтернативні способи прив'язки даних (без DataContext)

- 1) Властивість Source прив'язки дозволяє переозначити джерело прив'язування, успадковане від батьківського DataContext.

```
<TextBlock Text="{Binding Source={x:Static System:DateTime.Today},
  Mode=OneTime, StringFormat='{}© {0:yyyy} CompanyName'}" />
```

- 2) Властивість RelativeSource – можна вказати цільовий або батьківський елемент управління як джерело прив'язки.
- 3) Властивість ElementName – також використовується як shortcut для доступу до to accessing the parent control's DataContext.

Властивість RelativeSource

- Також дозволяє переозначити джерело прив'язування від DataContext, часто потрібна при спробах прив'язки до властивостей View Model від елементів DataTemplate.

```
<DataTemplate DataType="{x:Type DataModels:User}">
  <StackPanel>
    <TextBlock Text="{Binding Name}" />
    <TextBlock Text="{Binding DataContext.UserCount,
      RelativeSource={RelativeSource Mode=FindAncestor,
      AncestorType={x:Type Views:UserView}}}" />
  </StackPanel>
</DataTemplate>
```

  - Тут вказування властивості Mode, that specifies the relative position of the binding source compared to the binding target, не є обов'язковим.

```
<TextBlock Text="{Binding DataContext.UserCount,
  RelativeSource={RelativeSource
  AncestorType={x:Type Views:UserView}}}" />
```

Властивість RelativeSource

- Властивість Mode має тип перелічення RelativeSourceMode, у якого є 4 елементи: FindAncestor, Self, TemplatedParent, PreviousData.
  - 1) Приклад з FindAncestor можна розширити, використовуючи пов'язану з ним властивість RelativeSource.AncestorLevel, яка вказує рівень предка, на якому треба шукати джерело прив'язки.
  - Дана властивість дійсно корисна лише тоді, коли елемент управління має кілька предків одного типу

```
<StackPanel Tag="Outer">
  ...
  <StackPanel Orientation="Horizontal" Tag="Inner">
    <TextBlock Text="{Binding Tag, RelativeSource={RelativeSource
      Mode=FindAncestor, AncestorType={x:Type StackPanel},
      AncestorLevel=2}}" />
    ...
  </StackPanel>
</StackPanel>
```

Властивість RelativeSourceMode.Self

- binding source = binding target.
  - При застосуванні властивості Mode неявно присвоюється екземпляр Self.
  - Можна використати для прив'язки одної властивості UI-елементу до іншої:

```
<Rectangle Height="{Binding ActualWidth,
  RelativeSource={RelativeSource Self}}" Fill="Red" />
```

Властивість RelativeSourceMode.TemplatedParent

- Використовується тільки для доступу до властивостей елементів управління всередині ControlTemplate.
  - Шаблонізований предок зіставляється з об'єктом, який має застосований до себе ControlTemplate.
  - Властивість Mode неявно встановлюється як RelativeSourceMode.TemplatedParent.
  - Приклад: шаблонізований предок є екземпляром ProgressBar, який застосує цей шаблон до нього.
    - Використовуючи властивість TemplatedParent, можна отримати доступ до різних властивостей класу ProgressBar всередині ControlTemplate.

```
<ControlTemplate x:Key="ProgressBar" TargetType="{x:Type ProgressBar}">
  ...
  <TextBlock Text="{Binding Value,
    RelativeSource={RelativeSource TemplatedParent}}" />
  ...
</ControlTemplate>
```

Властивість RelativeSourceMode.PreviousData

- Корисна тільки тоді, коли визначається DataTemplate для елементів колекції.
  - Використовується для встановлення попереднього елемента колекції в якості джерела прив'язки.
  - Використовується не часто, при потребі порівняння сусідніх значень в колекції.

Властивість Binding.ElementName

- Дозволяє переозначати джерело прив'язки, встановлене DataContext.
  - Використовується для прив'язки властивості одного UI-елементу до властивості іншого елементу управління або іншої властивості того ж контрола.
  - Єдина потреба: іменувати елемент, до якого прив'язується поточний елемент управління.

```
<StackPanel Orientation="Horizontal" TextElement.FontSize="24" Margin="20">
  <CheckBox Name="Checkbox" Content="Service" Margin="0,0,10,0" />
  <TextBox Text="{Binding Service}"
    Visibility="{Binding IsChecked, ElementName=Checkbox,
    Converter={StaticResource BoolToVisibilityConverter}}" />
</StackPanel>
```

  - Тут властивість Visibility для TextBox прив'язується до властивості IsChecked для CheckBox

Властивість Binding.ElementName

- Також використовується як shortcut для доступу до DataContext батьківського елементу управління.
  - Якщо назвемо наш View, наприклад, This, можна використати властивість ElementName зсередини шаблону даних, щоб прив'язатись до властивості з батьківської View Model.

```
<DataTemplate DataType="{x:Type DataModels:User}">
  <StackPanel>
    <TextBlock Text="{Binding Name}" />
    <TextBlock Text="{Binding DataContext.UserCount, ElementName=This}" />
  </StackPanel>
</DataTemplate>
```

- Задаючи такі альтернативні джерела прив'язки, можна використовувати тільки один з трьох методів за раз.
  - Інакше - виключення

Прив'язка з пріоритетом

- Може виникнути потреба задати кілька source binding paths та відобразити їх на одну властивість (ціль прив'язки).
  - Можна використати клас MultiBinding.
  - Альтернатива: PriorityBinding.
- Клас PriorityBinding дозволяє задати кілька прив'язок та надати їм пріоритети (від вищого до нижчого).
  - Спеціальна функціональність: дозволяє показати значення від першої прив'язки, яка повертає валідне значення.
  - Якщо це не найбільш пріоритетна прив'язка, відображення оновиться значенням від прив'язки з найвищим пріоритетом, коли воно успішно визначиться (resolve).

PriorityBinding у дії

- Дозволяє задати прив'язку до нормальної властивості, яка негайно буде оброблена (resolve), поки потрібне значення завантажиться, обчислиться чи знайдеться іншим способом.
  - Постачається зображення за замовчуванням, поки потрібне зображення завантажується, або виводиться повідомлення, поки значення для відображення не обчислене.

```
<TextBlock>
  <TextBlock.Text>
    <PriorityBinding>
      <Binding Path="SlowString" IsAsync="True" />
      <Binding Path="FastString" Mode="OneWay" />
    </PriorityBinding>
  </TextBlock.Text>
</TextBlock>
```

```
public string FastString
{
  get { return "The value is being calculated..."; }
}
```

Прив'язка зсередини control templates

- TemplateBinding – тип прив'язки, що використовується всередині елементів ControlTemplate, щоб прив'язатись до властивостей шаблонізованого типу.
  - Подібно до властивості RelativeSource.TemplatedParent.
  - Спосіб зв'язування властивостей елементів всередині ControlTemplate з властивостями елементів за межами шаблону.

```
<ControlTemplate x:Key="ProgressBar" TargetType="{x:Type ProgressBar}">
  ...
  <TextBlock Text="{TemplateBinding Value}" />
  ...
</ControlTemplate>
```

  - TemplateBinding записується простіше за аналогічну властивість RelativeSource.TemplatedParent:

```
<TextBlock Text="{Binding Value,
  RelativeSource={RelativeSource TemplatedParent}}" />
```

TemplateBinding vs RelativeSource.TemplatedParent

- Рекомендується використовувати TemplateBinding замість RelativeSource.TemplatedParent.
- Хоч вони виконують однакову операцію прив'язки, існують деякі відмінності:
  - TemplateBinding перевіряється при компіляції, а RelativeSource.TemplatedParent – тільки під час виконання програми.
  - TemplateBinding – простіша форма прив'язки, для неї відсутні деякі властивості класу Binding, зокрема StringFormat і Delay.
  - TemplateBinding накладає додаткові обмеження на користувача: режим прив'язки перманентно встановлюється як OneWay, а джерело і ціль прив'язки повинні бути властивостями залежності.
  - Це спеціалізований тип прив'язки, спроектований ефективно виконувати саме поставлену задачу.

Зміни в джерелі даних для прив'язки

- Може бути потрібно задати значення за замовчуванням для нової форми, очистити значення зі старої форми або навіть задати form labels з представлених View Models.
  - Для цього View Models повинні реалізувати інтерфейс INotifyPropertyChanged.
- При прив'язці джерела даних до UI-елемента до події PropertyChanged об'єкта-джерела прикріплюється обробник події.
  - Коли сповіщення про зміну властивості від джерела прив'язки отримано, елемент управління оновлюється з новим значенням.
  - Подія PropertyChanged джерела прив'язки буде null, якщо не було прикріплено обробника і жодна з властивостей не прив'язана до UI-елементів.
  - Це причина перевірки на null перед raising цієї події.

Зміни в джерелі даних для прив'язки

- Усі режими прив'язки працюють у напрямку від binding source до binding target, крім OneWayToSource.
  - Проте тільки такий режим та режим TwoWay поширюють зміни в напрямку від binding target до binding source.
  - When the binding is working in either of these modes, it attaches a handler to the target control to listen for changes to the target property.
- When it receives notification of a change to the target property, its behavior is determined by the value of the binding's UpdateSourceTrigger property.
  - This property is of the enumeration type UpdateSourceTrigger, which has four members.

Перелічення UpdateSourceTrigger

- Найбільш поширеним є екземпляр PropertyChanged, який вказує, що властивість-джерело має оновлюватись, як тільки цільова властивість була змінена.
  - Це значення за замовчуванням для більшості елементів управління.
- Член LostFocus – наступне за популярністю значення – визначає, що прив'язка має оновлювати своє джерело, коли користувач переміщує фокус із прив'язаного елементу управління.
  - Наприклад, коли валідація спрацьовує, як тільки користувач завершив ввід даних у текстбокси.
- Екземпляр Explicit не оновлюватиме джерело прив'язки без явної інструкції для цього.
- Екземпляр Default подібний для екземпляру Default перелічення Binding.Mode в тому, що використовує значення, визначене кожною цільовою Dependency Property та є значенням за замовчуванням властивості UpdateSourceTrigger.

UpdateSourceTrigger.Explicit

- As we need to programmatically call the UpdateSource method of the internal BindingExpression object in order to propagate the changes to the binding source, this option is not generally used in our normal Views.
  - Instead, if used at all, we would find it in our CustomControl classes.
  - Note that calling the UpdateSource method will do nothing if the binding mode is not set to one of the OneWayToSource or TwoWay instances.
- If we had an instance of a textbox and we wanted to explicitly update the binding source that was data bound to its Text property, we can access the lower-level BindingExpression object from the BindingOperations.GetBindingExpression method and call its UpdateSource method.

```
BindingExpression bindingExpression =
  BindingOperations.GetBindingExpression(textBox, TextBox.TextProperty);
bindingExpression.UpdateSource();
```

Binding multiple sources to a single target property

- Більш поширений спосіб – використовувати поєднання об'єкту класу MultiBinding з класом, який реалізує інтерфейс IMultiValueConverter.
  - The MultiBinding class enables us to declare multiple binding sources and a single binding target.
  - If the Mode or UpdateSourceTrigger properties of the MultiBinding class are set, then their values are inherited by the contained **binding** elements, unless they have different values set explicitly.
- The values from the multiple binding sources can be combined in one of two ways:
  - Their string representations can be output using the StringFormat property,
  - we can use a class that implements the IMultiValueConverter interface to generate the output value.
    - This interface is very similar to the IValueConverter interface, but works with multiple data bound values instead.

- When implementing the IMultiValueConverter interface, we do not set the ValueConvertion attribute that we are accustomed to setting in the IValueConverter implementations that we have created.
  - In the Convert method that we need to implement, the **value** input parameter of type **object** from the IValueConverter interface is replaced by an object array named **values**, which contains our input values.
  - In the ConvertBack method, we have an array of type **Type** for the types of the binding targets and one of type **object** for the return types.
  - Apart from these slight differences, these two interfaces are the same.

Дякую за увагу!