Введение в привязку данных

Разработаем простую программу, демонстрирующую механизм привязки данных (Binding).
Больше информации:
https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/data/data-binding-overview?view=netdesktop-5.0
https://metanit.com/sharp/wpf/11.php

Создание окон

Основное окно («главное меню») будет иметь компоновку StackPanel.

XAML:

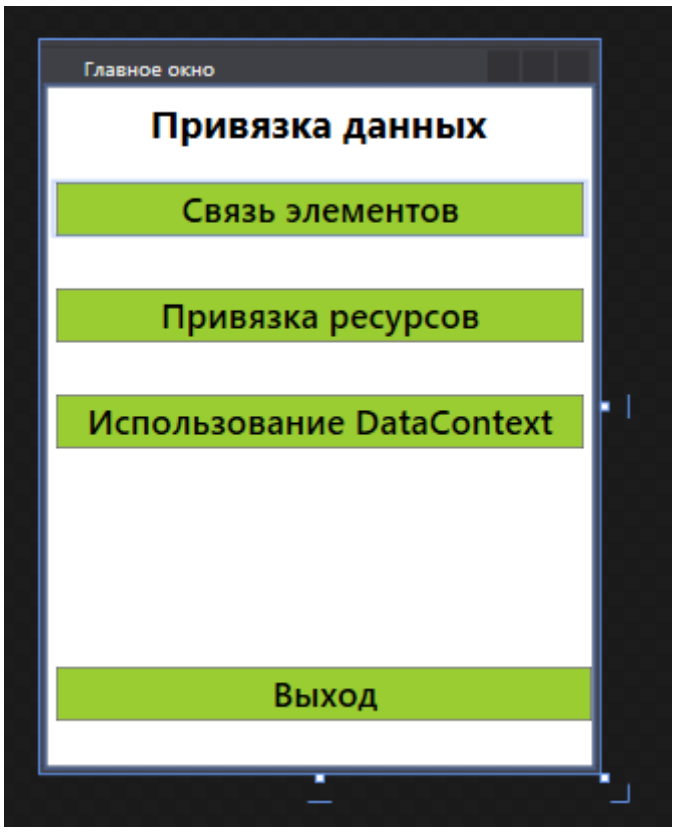
```
       <StackPanel>
         <Label Content="Привязка данных" FontSize="22" FontWeight="Bold"
HorizontalAlignment="Center"  />
         <Button Content="Связь элементов" />
         <Button Content="Привязка ресурсов"  />
         <Button Content="Использование DataContext"  />
         <Button Content="Выход" Margin="5,110,0,0" />
    </StackPanel>
```

Каждый из элементов может иметь свой набор ресурсов. Добавим (выше StackPanel) набор ресурсов для окна (Window):

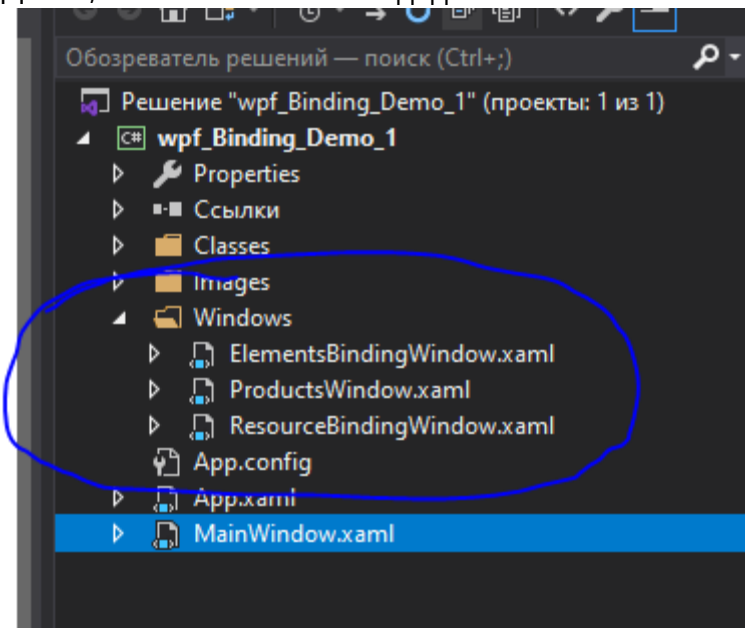
```
<Window x:Class="wpf_Binding_Demo_1.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:wpf_Binding_Demo_1"
        mc:Ignorable="d"
        Title="Главное окно" Height="420" Width="320">
    <Window.Resources>
        <Style TargetType="Button">
            <Setter Property="FontSize" Value="20" />
            <Setter Property="FontWeight" Value="SemiBold" />
            <Setter Property="Margin" Value="5, 15" />
            <Setter Property="Background" Value="YellowGreen" />
        </Style>
    </Window.Resources>
… далее StackPanel
```

Внутри, мы задаем стиль для всех элементов Button данного окна. Тип элемента указывается в теге Style (свойство TargetType), далее, элементами Setter настраиваются свойства (Property задает свойство, Value - значение).

Примерный результат:



Далее, самостоятельно создадите 3 окна:



Настройте переход по ним, используя метод ShowDialog. Напишите код для кнопки «Выход».

Окно привязки элементов (ElementsBindingWindow)

Зададим следующую разметку:

```
<StackPanel>

        <Label Content="Ввод" />
        <TextBox Name="InputTextBox" Text="Data binding" />
        <Label Content="Связанный Label (Text):" />
        <Label Content="qwerty" />
```

```
        <Label Content="Связанный Label (Text.Length)" />
        <Label Content="1234" />
        <Label Content="Связанный TextBlock" />
        <TextBlock FontSize="16">
            <Run Text="всего" />
            <Run TextDecorations="Underline" Text="1234" />
            <Run Text="знаков" />
        </TextBlock>
    </StackPanel>
```

Далее, начнем выполнять привязку. Чтобы привязать свойство одного элемента к свойству другого, используем {Binding ElementName=ИМЯ, Path=СВОЙСТВО}
Начнем с Label, который содержит qwerty, и изменим его Content:

```
<Label Content="{Binding ElementName=InputTextBox, Path=Text}" />
```

Если все сделано правильно, то в Label будет содержаться тот же текст, что и InputTextBox. Запустите приложение, попробуйте изменять ввод в TextBox. Проверьте, что значение Label также изменяется.

Path позволяет задавать и более хитрые конструкции, в частности, мы можем обращаться к свойствам свойств. Поскольку Text это обычный string, он имеет свойство Length (доступное только для чтения).
Зададим привязку для Label, где было «1234»:

```
<Label Content="{Binding ElementName=InputTextBox, Path=Text.Length}" />
```

Теперь, данный Label будет содержать длину текста в InputTextBox. Проверьте работоспособность.

Далее, также привяжем один из Run к Text.Length:

```
<TextBlock FontSize="16">
            <Run Text="всего" />
            <Run TextDecorations="Underline" Text="{Binding ElementName=InputTextBox,
Path=Text.Length}" />
            <Run Text="знаков" />
        </TextBlock>
```

Данный код приведет к ошибке, поскольку TextBlock предполагает изменение значения, в то время как свойство Length доступно только для чтения. Чтобы разрешить данную проблему зададим направление привязки в «одну сторону». Для этого используем Mode:

```
<TextBlock FontSize="16">
            <Run Text="всего" />
            <Run TextDecorations="Underline" Text="{Binding ElementName=InputTextBox,
Path=Text.Length, Mode=OneWay}" />
            <Run Text="знаков" />
        </TextBlock>
```

Если все работает, то добавим в StackPanel следующее:

```
<StackPanel.Background>
            <RadialGradientBrush>
                <GradientStop Color="DarkGray" Offset="1" />
                <GradientStop Color="LightGray" Offset="0.3" />
                <GradientStop Color="White" Offset="0" />
```

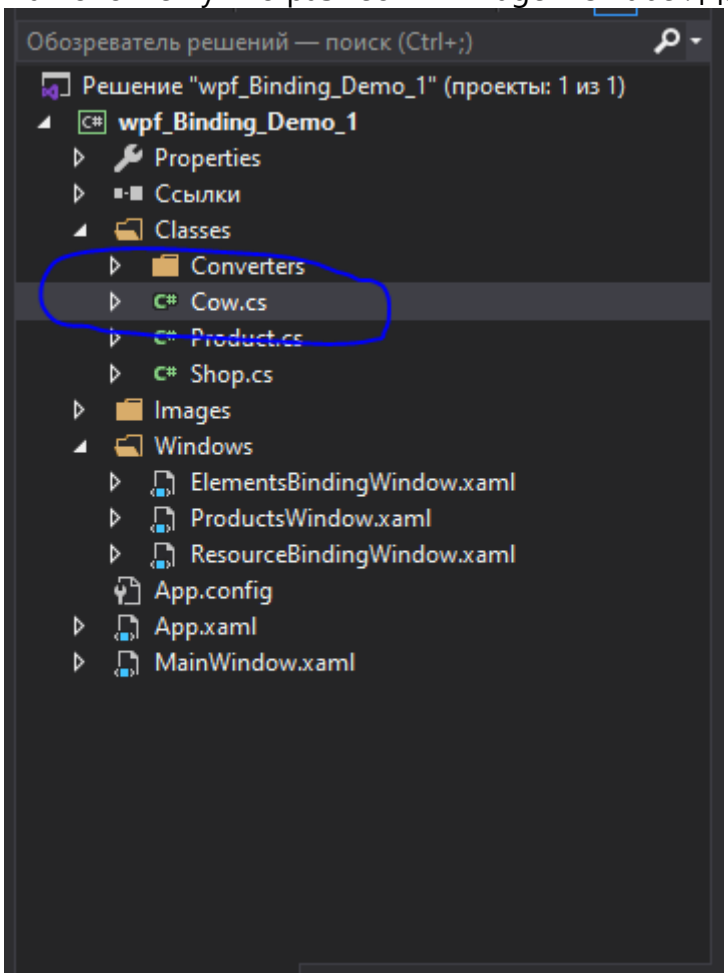
```
        </RadialGradientBrush>
    </StackPanel.Background>
    <StackPanel.Resources>
        <Style TargetType="Label">
            <Setter Property="FontSize" Value="14" />
        </Style>
    </StackPanel.Resources>
```

Первая часть позволяет задать фон (вместо Background=”NNN”, мы используем отдельный «контейнер» A.Background, в который помещаем градиентную кисть). Вторая часть задает стиль всех Label, указывая их размер.

Привязка ресурсов (ResourceBindingWindow)

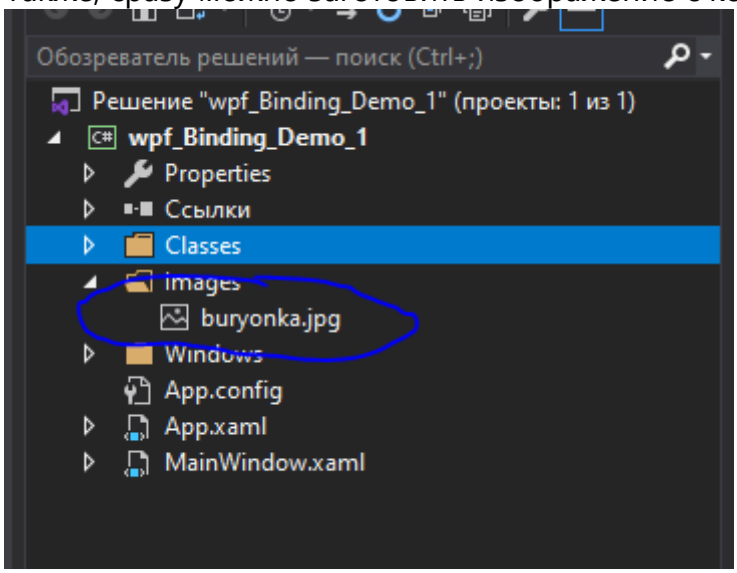
В данном окне используем компоновку Canvas.

На полотне нужно разместить Image и 3 Label. Далее, создадим класс Cow:



```
Ссылок: 0
public class Cow
{
    ссылка: 1
    public string Name { get; set; }
    Ссылок: 0
    public string Voice { get; set; }
    Ссылок: 0
    public string Image { get; set; }
    ссылка: 1
    public int Age { get; set; }
    Ссылок: 0
    public string FullAgeString
    {
        get
        {
            return $"{DateTime.Now.ToShortDateString()} {Name} {Age} лет";
        }
    }
}
```

Также, сразу можно заготовить изображение с коровой:



В ресурсах окна добавим экземпляр Cow. Для этого нужно подключить в XAML namespace и добавить строчку в ресурсах:

```
<Window x:Class="wpf_Binding_Demo_1.Windows.ResourceBindingWindow"
        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:wpf_Binding_Demo_1.Windows"
        xmlns:classes="clr-namespace:wpf_Binding_Demo_1.Classes"
        mc:Ignorable="d"
        Title="ResourceBindingWindow" Height="450" Width="600"
        ResizeMode="NoResize">
    <Window.Resources>
        <classes:Cow x:Key="Buryonka" Name="Буренка" Voice="Myyyy"
Image="./../Images/buryonka.jpg" Age="5" />
    </Window.Resources>
```

Убедитесь, что данное имя (в примере имя classes) совпадает при объявлении и использовании.

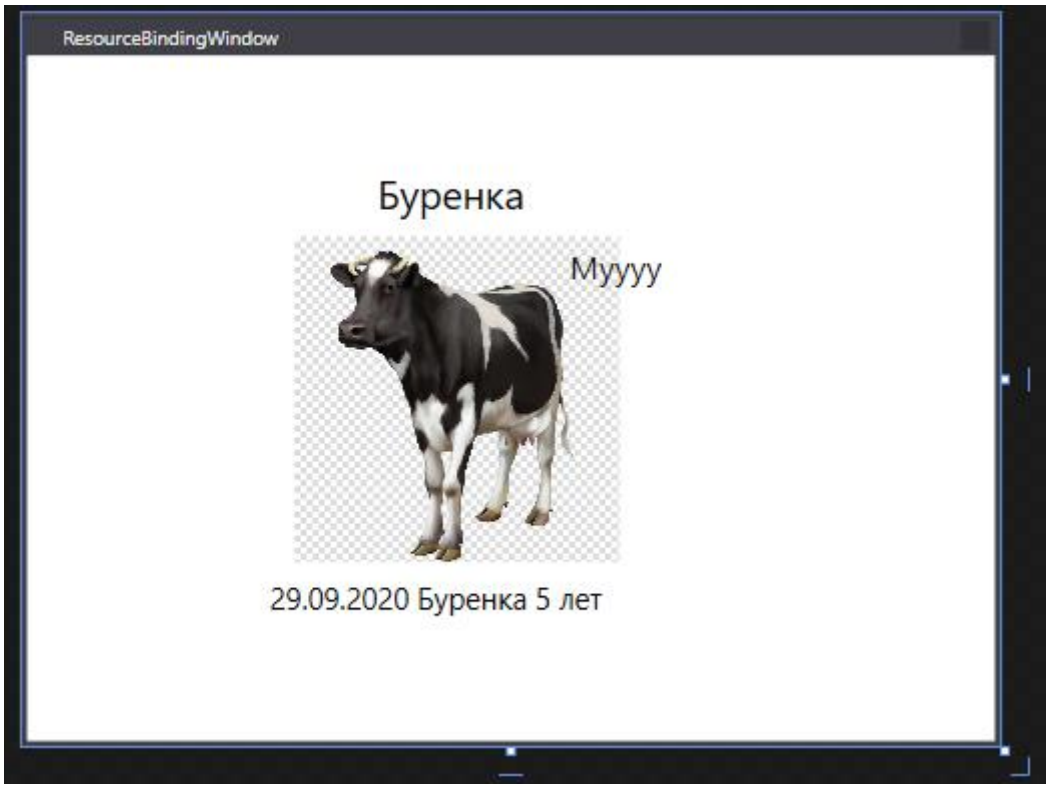
Теперь, выполним привязку к свойствам Buryonka. Также, можно задать координаты:

```
<Canvas>
        <Image Source="{Binding Source={StaticResource Buryonka},Path=Image}" Width="200"
Canvas.Left="164" Canvas.Top="111" />
        <Label Content="{Binding Source={StaticResource Buryonka},Path=Name}"
FontSize="24" Canvas.Left="210" Canvas.Top="64" />
        <Label Content="{Binding Source={StaticResource Buryonka},Path=Voice}"
FontSize="20" Canvas.Left="328" Canvas.Top="111" />
        <Label Content="{Binding Source={StaticResource Buryonka},Path=FullAgeString}"
FontSize="18" Canvas.Left="144" Canvas.Top="316" />
    </Canvas>
```

Здесь работает следующий шаблон:
{Binding Source=ИСТОЧНИК, Path=СВОЙСТВО}

Поскольку источником является ресурс, мы используем ссылку на него
{StaticResource Buryonka}

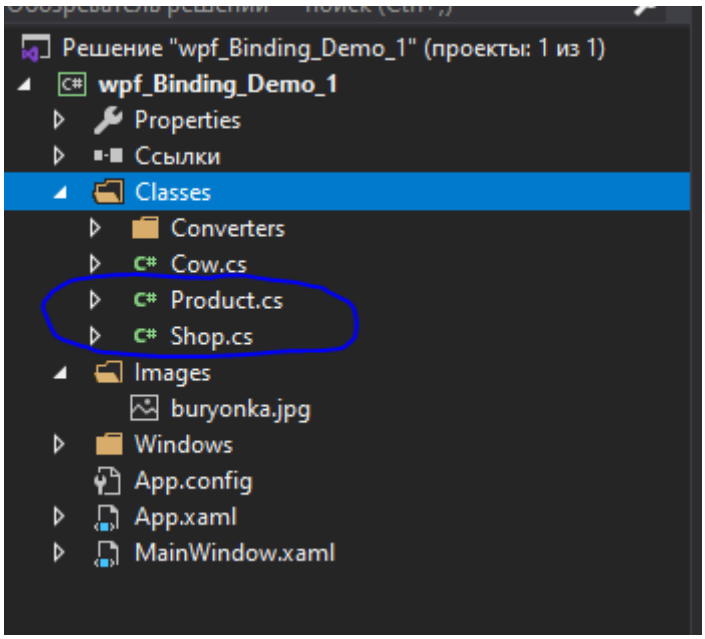
Обратите внимание на использование FullAgeString, который «вычисляется» за счет написания геттера. Данный трюк может оказаться полезным.



Использование DataContext

В WPF можно использовать объект в качестве контекста данных. Это позволяет привязывать члены данного объекта (в основном публичные свойства) к элементам управления.

Создадим два класса:




```
namespace wpf_Binding_Demo_1.Classes
{
    Ссылок: 5
    public class Product
    {
        Ссылок: 2
        public string ProductName { get; set; } // название
        Ссылок: 0
        public string ImageFile { get; set; } // изображение
        Ссылок: 2
        public int Count { get; set; } // количество
        Ссылок: 3
        public decimal Price { get; set; } // цена
        Ссылок: 4
        public double Discount { get; set; } // скидка (в долях, от 0 до 1)

        // true, если скидка имеется
        Ссылок: 0
        public bool HasDiscount
        {
            get
            {
                return Discount > 0;
            }
        }

        // цена со скидкой
        Ссылок: 0
        public decimal DiscountPrice
        {
            get
            {
                return Price * (decimal)(1 - Discount);
            }
        }

    }
}
```

Проблемы не найдены.

```
namespace wpf_Binding_Demo_1.Classes
{
    Ссылок: 3
    class Shop
    {
        ссылка: 1
        public List<Product> Products { get; set; } = new List<Product>();
        ссылка: 1
        public string Name { get; private set; }
        ссылка: 1
        public Shop(string name)
        {
            Name = name;
        }


    }
}
```

Зададим разметку

```
<StackPanel>
        <Label Content="{Binding Name}" />
        <ListBox HorizontalContentAlignment="Stretch">
            <ListBox.ItemTemplate>
                <DataTemplate>

                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </StackPanel>
```

В коде напишем:

```
namespace wpf_Binding_Demo_1.Windows
{
    using wpf_Binding_Demo_1.Classes;
    /// <summary>
    /// Логика взаимодействия для ProductsWindow.xaml
    /// </summary>
    Ссылок: 4
    public partial class ProductsWindow : Window
    {
        private Shop shop;
        ссылка: 1
        public ProductsWindow()
        {
            InitializeComponent();
            DataContext = shop = new Shop("Мечи & топоры");
            shop.Products.AddRange(new Product[]
            {
                new Product { ProductName = "Топорик", Count = 2, Discount = 0, Price = 6000 },
                new Product { ProductName = "Томагавк", Count = 3, Discount = 0.5, Price = 4000 }
            });

        }
    }
}
```

Привяжем Content у Label к названию:

```
<Label Content="{Binding Name}" />
```

Поскольку мы имеем контекст (shop в качестве DataContext) WPF будет известно, что такое Name – это публичное свойство класса Shop.

Аналогичным образом привяжем:

```
<ListBox ItemsSource="{Binding Products}" HorizontalContentAlignment="Stretch">
```

Источником элементов будет свойство Products, которое является коллекцией Product. Далее, внутри DataTemplate, мы должны использовать свойства элемента коллекции, то есть этого самого Product. Напишем:

```
<DataTemplate>
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="1*" />
            <ColumnDefinition Width="5*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <Image  Width="100" />
        <StackPanel Grid.Column="1">
            <Label Content="{Binding ProductName}" />
            <TextBlock>
                <Run Text="Количество: " />
                <Run Text="{Binding Count}" />
            </TextBlock>
            <TextBlock>
                <Run Text="Цена" />
                <Run Text="{Binding Price}"  />
                <Run Text="рублей" />
            </TextBlock>
        </StackPanel>
        <Button Grid.Column="2" Content="Купить" />
    </Grid>
</DataTemplate>
```

Результат будет примерно следующий:

Конверторы (IConverter)

Создадим внутри описания товара следующий элемент:

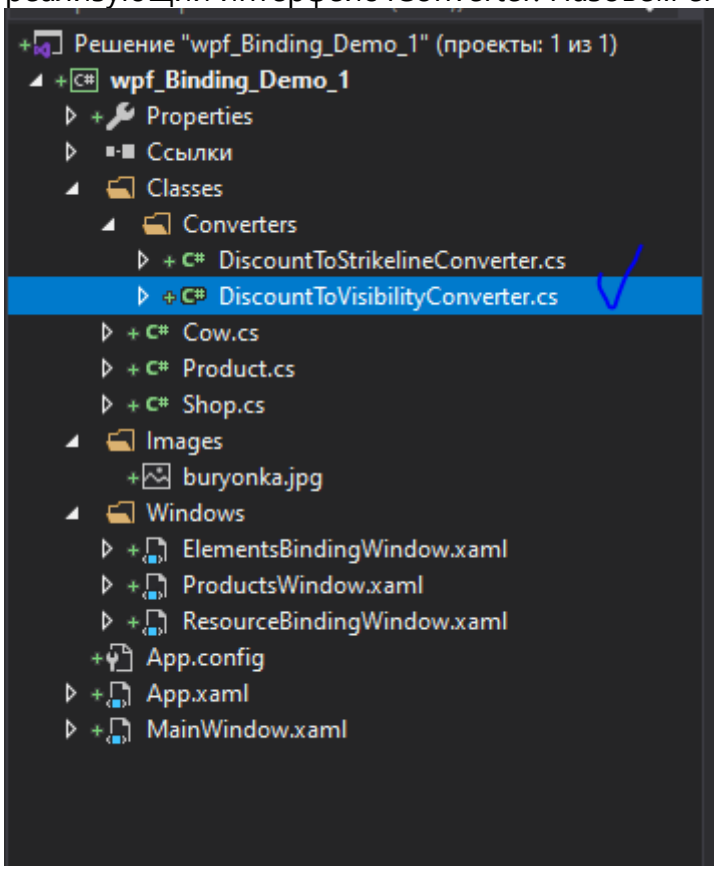
```
<TextBlock>
    <Run Text="Скидка!" />
</TextBlock>
```

Хотелось бы показывать данный элемент в том случае, если есть скидка. Теоретически, это могло бы выглядеть так:

```
<TextBlock Visibility="{Binding HasDiscount}">
    <Run Text="Скидка!" />
</TextBlock>
```

Однако, это не сработает, поскольку свойство Visibility имеет тип, отличный от bool (в частности тип так и называется Visibility, и это обычный enum).

Чтобы решить данную проблему, мы можем создать конвертер – класс, реализующий интерфейс IConverter. Назовем его DiscountToVisibilityConverter:



```
namespace wpf_Binding_Demo_1.Classes.Converters
{
    using System.Globalization;
    using System.Windows;
    using System.Windows.Data;

    Ссылок: 0
    class DiscountToVisibilityConverter : IValueConverter
    {
        ссылка: 1
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            var hasDiscount = (bool)value;
            if (hasDiscount)
                return Visibility.Visible;
            else
                return Visibility.Collapsed;
        }

        ссылка: 1
        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            return (Visibility)value == Visibility.Visible;
        }
    }
}
```

Здесь, value – это значение для конвертации. Методы Convert и ConvertBack задают логику преобразования в обе стороны («прямая» и «обратная»).

Convert берет значение скидки (bool), и если скидка существует возвращает Visibility.Visible, иначе – значение, соответствующее скрытому элементу, без резервирования пространства в окне.

ConvertBack берет значение Visibility, и если оно «видим», то результатом будет true.

Создадим экземпляр DiscountToVisibilityConverter с помощью ресурсов:

```
<Window.Resources>
  <converters:DiscountToVisibilityConverter x:Key="DiscountConverter" />
</Window.Resources>
```

Namespace converters добавьте по аналогии с classes в окне с коровой.

Теперь, используем наш конвертер при привязке:

```
<TextBlock Visibility="{Binding HasDiscount,
       Converter={StaticResource DiscountConverter}}">
     <Run Text="Скидка!" />
</TextBlock>
```

Если все успешно, то вы увидите скидку только там, где она была задана.
Изменим немного XAML код:

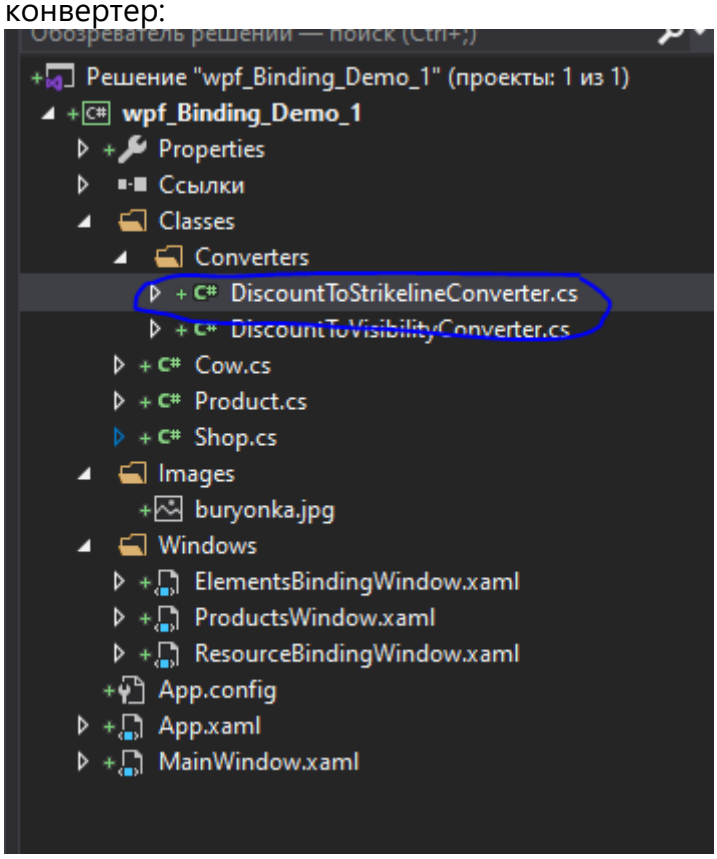
```
<TextBlock
Visibility="{Binding HasDiscount,Converter={StaticResource DiscountConverter}}"
       Foreground="Red"
       FontSize="12" FontWeight="SemiBold">
              <Run Text="СКИДКА! НОВАЯ ЦЕНА:" />
              <Run Text="{Binding DiscountPrice, Mode=OneWay}" />
              <Run Text="рублей." />
</TextBlock>
```

Проверьте результат:

Отлично! Теперь сделаем еще одну штуку.

Зачеркнем старую цену, если имеется скидка. Чтобы задать подчеркивание, зачеркивание или иной подобный эффект, существует свойство TextDecoration. Для зачеркивания: TextDecoration = “Strikethrough”

TextDecoration также имеет тип, отличный от bool, поэтому мы создадим конвертер:



```
namespace wpf_Binding_Demo_1.Classes.Converters
{
    using System.Windows.Data;
    using System.Windows;

    Ссылок: 0
    class DiscountToStrikelineConverter : IValueConverter
    {
        ссылка: 1
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            bool HasDiscount = (bool)value;
            if (HasDiscount)
            {
                return TextDecorations.Strikethrough;
            }
            else
            {
                return new TextDecoration();
            }
        }

        ссылка: 1
        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            var decoration = value as TextDecoration;
            if (decoration != null)
            {
                return decoration.Location == TextDecorationLocation.Strikethrough;
            }
            else
            {
                throw new ArgumentException();
            }
        }
    }
}
```

TextDecorations представляет собой набор заготовленных «украшательств». Если мы имеем скидку (значение - true), то вернем Strikethrough, иначе вернем новый декоратор, который будет «пустым».

Обратное преобразование возвращает true, если текст зачеркнут (то есть HasDiscount == true, при зачеркнутом тексте).

Используем данный декоратор. Создадим экземпляр в ресурсах:

```
<Window.Resources>
    <converters:DiscountToStrikelineConverter x:Key="DecorationConverter" />
    <converters:DiscountToVisibilityConverter x:Key="DiscountConverter" />
</Window.Resources>
```

Напишем для TextBlock:

```
<TextBlock>
        <Run Text="Цена" />
        <Run Text="{Binding Price}"
             TextDecorations="{Binding HasDiscount,
             Converter={StaticResource DecorationConverter}}" />
        <Run Text="рублей" />
</TextBlock>
```

Результат:

Дополнительно

Самостоятельно добавьте изображения (нужно добавить их в отдельный каталог проекта, указать имена файлов в коде при создании коллекции и задав привязку свойства Source у Image).

Задайте оформление информации о товаре (изменяя внешний вид Label и TextBlock). Например, название товара можно сделать более крупным и использовать полужирный шрифт.

Добавьте ниже ListBox строку, в которой было бы написано: «Всего наименований: N», где N – количество наименований товаров. Подсказка: List имеет свойство Count.