В лабораторной работе №4 использовались привязки, которые устанавливали связь между двумя элементами. WPF так же позволяет создавать выражения привязки, которые извлекают данные из невизуальных объектов (объектов любых классов). Основное требование: привязку можно выполнять только к общедоступным (public) свойствам (нельзя делать привязку к полям и методам).

В популярных шаблонах (паттернах) проектирования приложений MVC, MVP, MVVM принято разделять бизнес-логику (данные предметной области) и пользовательский интерфейс. Классы, представляющие данные предметной области называются *моделями*. Модель содержит бизнес-данные, может содержать операции, преобразования и правила для манипулирования этими данными. Однако, модель не содержит никакой информации, как эти данные нужно визуализировать.

Пример простой модели данных, представляющей товар в магазине:

```
public class MarketItem
{
    public int Id { get; set; }

    public string Name { get; set; }

    public string Description { get; set; }

    public decimal Price { get; set; }
}
```

Модель является упрощённой и содержит следующие характеристики товара: артикул (Id), название, описание и цену.

Теперь создадим простую XAML-разметку для отображения такой модели:

```
<UniformGrid Margin="8" Columns="2" x:Name="itemGrid">
    <TextBlock Text="Наименование: "/>
    <TextBlock Text="{Binding Path=Name}" TextWrapping="Wrap"/>

    <TextBlock Text="Стоимость: "/>
    <TextBlock Text="{Binding Price}"/>

    <TextBlock Text="Артикул: "/>
    <TextBlock Text="{Binding Id}"/>

    <TextBlock Text="Описание: "/>
    <TextBlock Text="{Binding Description}" TextWrapping="Wrap"/>
</UniformGrid>
```

Для отображения нужного свойства указывается выражение привязки:

```
{Binding Path=Name}
```

Допускается упрощённая форма:

```
{Binding Name}
```

Осталось привязать нужный объект данных к интерфейсу. Для примера создадим объект в коде:

```
var item = new MarketItem()
{
    Id = 10001,
    Name = "Transcend microSDHC 16GB карта памяти",

    Description = "Карта памяти Transcend microSDHC Class 10 обладает отличными " +
       " рабочими характеристиками при размере всего лишь 1/10 от размера SD карты.",
    Price = 485
};
```

Привязка выполняется с помощью свойства элемента DataContext:

```
itemGrid.DataContext = item;
```

Обратите внимание, в данном примере задаётся свойство DataContext для диспетчера компоновки (UniformGrid), однако привязка действует и на все вложенные элементы TextBlock. Можно было задать свойство DataContext не для UniformGrid, а для каждого элемента TextBlock – результат был бы таким же. Иными словами, WPF сначала проверяет свойство DataContext элемента, который содержит выражение привязки. Если свойство не задано, WPF начинает идти вверх по дереву элементов, пока не будет обнаружен контекст данных.

Вложенные объекты

В выражениях привязки допускается использовать вложенные объекты. Создадим класс Price для описания цены, содержащий значение цены (Value) и валюту (Currency)

```
public class Price
{
    public string Currency { get; set; }

    public decimal Value { get; set; }
}
```

В классе MarketItem изменим свойство Price:

```
public Price Price { get; set; }
```

Теперь свойство Price – это объект класса Price (одинаковые имена не запрещаются), в свою очередь объект цены содержит свойства Currency и Value. Выражения привязки для них будут следующие:

```
{Binding Path=Price.Currency}

{Binding Path=Price.Value}
```

Двунаправленная привязка

Привязка к объектам, так же как и привязка к элементам, поддерживает режим TwoWay. Заменим в XAML строку, отображающую свойство Description:

```
<TextBox Text="{Binding Path=Description, Mode=TwoWay,
    UpdateSourceTrigger=PropertyChanged}" TextWrapping="Wrap"/>
```

Теперь, при изменении текста элемента TextBox, будет меняться значение свойства Description привязанного объекта. Значение параметра UpdateSourceTrigger определяет режим применения изменений к источнику данных. Возможны следующие режимы:

- **PropertyChanged**. Источник обновляется немедленно, когда изменяется целевое свойство.
- **LostFocus**. Источник обновляется, когда привязанный элемент теряет фокус.
- **Explicit**. Источник не обновляется, пока не будет вызван метод BindingExpression.UpdateSource().

Mode=TwoWay означает режим двунаправленной привязки (для элемента TextBox данный режим используется по умолчанию, поэтому параметр Mode можно не указывать).

DataTrigger

DataTrigger работает схожим образом с обычным триггером (Trigger). Отличие в том, что DataTrigger реагирует на изменение свойств не самого элемента, а свойств привязанного объекта.

Добавим в класс MarketItem ещё одно свойство Availability (есть ли товар в наличии).

```
    public bool Availability { get; set; }
```

Если товара в наличии нет, будем отображать весь текст более бледным цветом. Для этого добавим EventTrigger, реагирующий на свойство контекста данных Availability, и меняющий свойство элемента Opacity.

```
    <UniformGrid Margin="8" Columns="2" x:Name="itemGrid">
        <UniformGrid.Resources>
            <Style x:Key="ValueStyle" TargetType="FrameworkElement">
                <Style.Triggers>
                    <DataTrigger Binding="{Binding Availability}" Value="False">
                        <Setter Property="Opacity" Value="0.5"/>
                    </DataTrigger>
                </Style.Triggers>
            </Style>
        </UniformGrid.Resources>
        <TextBlock Text="Наименование: "/>
        <TextBlock Text="{Binding Path=Name}" TextWrapping="Wrap"
             Style="{StaticResource ValueStyle}"/>

        <TextBlock Text="Стоимость: "/>
        <TextBlock Text="{Binding Path=Price.Value}"
             Style="{StaticResource ValueStyle}"/>
        ...
```

Загрузка модели данных из файла

В примере выше объект контекста данных, описывающий товар, создавался прямо в коде. Далее будет рассмотрено чтение данных из файла формата JSON.

Пример содержимого JSON-файла, описывающего объект MarketItem:

```
{
  "Id": 73766,
  "Name": "Asus ZenPad 8 Z380KNL",
  "Description": "Внешний вид Asus ZenPad 8 Z380KNL - яркий, стильный, современный - не
оставит равнодушным ни одного пользователя!",
  "Price": {
    "Value": 13990,
    "Currency": "RUR"
  },
  "Availability": true
}
```

Сперва нужно подключить к проекту библиотеку Json.NET, для этого требуется выполнить команду в консоли диспетчера пакетов:

```
PM> Install-Package Newtonsoft.Json
```

Следующий код выполняет загрузку данных из JSON-файла и устанавливает свойство DataContext диспетчера компоновки:

```
using System.IO;
using Newtonsoft.Json;
...
    // Загрузка содержимого файла в строку
    string json = File.ReadAllText("item.json");

    // Десериализация JSON-данных в объект MarketItem
    var item = JsonConvert.DeserializeObject<MarketItem>(json);

    // Привязка данных к графическому интерфейсу
    itemGrid.DataContext = item;
```

Сохранить изменения объекта данных можно следующим образом:

```
    string json = JsonConvert.SerializeObject(item);
    File.WriteAllText("item.json", json1);
```

Задание

К заданию прилагается JSON-файл (выдаётся индивидуально)

1. Написать класс – модель данных, свойства должны соответствовать данным JSON;
2. В окне расположить элементы для редактирования данных, сделать привязку к свойствам класса-модели. Для текстовых и числовых данных использовать элементы TextBox, для булевских типов – элементы CheckBox.
3. Сделать загрузку и сохранение в JSON-файл.