Валидация данных

Валидация данных

При работе с данными важную роль играет валидация данных. Прежде чем использовать полученные от пользователя данные, нам надо убедиться, что они введены правильно и представляют корректные значения.

ExceptionValidationRule

Один из встроенных способов проверки введенных данных в WPF представлен классом **ExceptionValidationRule**. Этот класс обозначает введенные данные как некорректные, если в процессе ввода возникает какое-либо исключение, например, исключение преобразования типов.

ExceptionValidationRule

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        loadData();
    }

    private void loadData()
    {
        Person p = new Person();

        this.DataContext = p;
    }
}
```

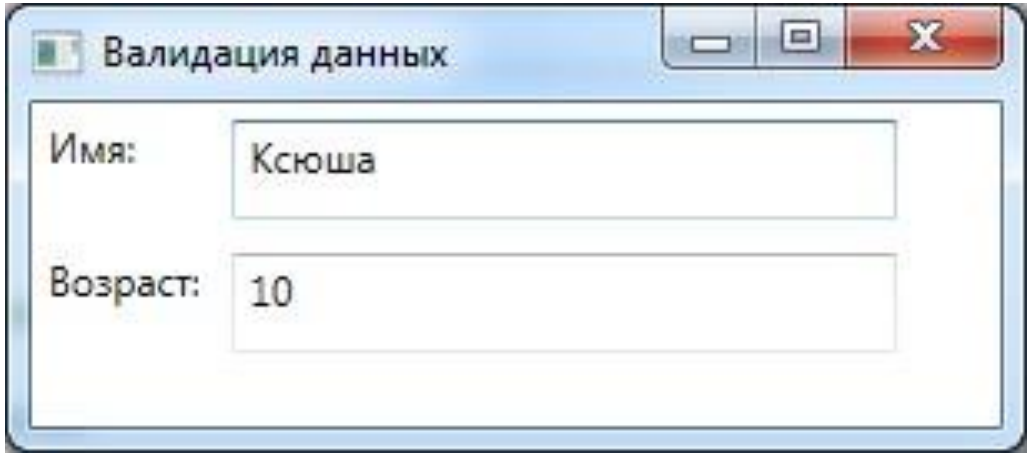
```
<TextBox Name="tbAge" Grid.Row="1" Grid.Column="1" Width="200" Margin="5">
    <TextBox.Text>
        <Binding Path="Age" UpdateSourceTrigger="PropertyChanged">
            <Binding.ValidationRules>
                <ExceptionValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

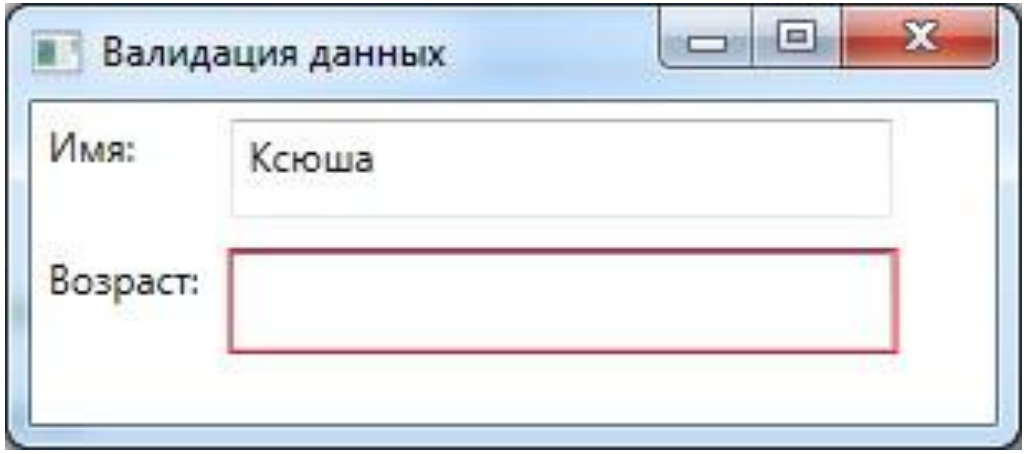
ExceptionValidationRule

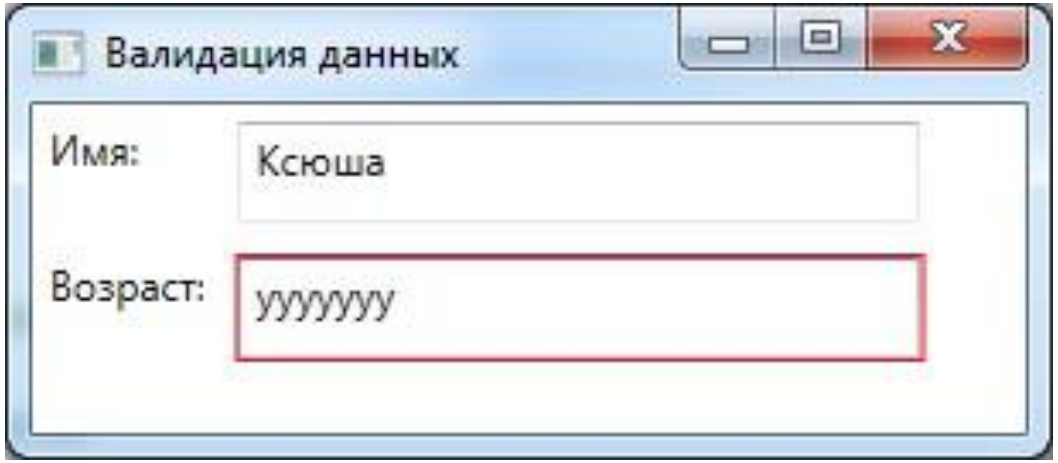
```
<TextBox Name="tbAge" Grid.Row="1" Grid.Column="1" Width="200" Margin="5">
    <TextBox.Text>
        <Binding Path="Age" UpdateSourceTrigger="PropertyChanged">
            <Binding.ValidationRules>
                <ExceptionValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

В данном случае мы задаем объект Binding для свойства Text. Данный объект имеет коллекцию парвил валидации вводимых данных - **ValidationRules**. Эта коллекция принимает только одно правило валидации, представленное классом ExceptionValidationRule.

ExceptionValidationRule







Валидация по исключениям

Валидация на основе исключений достаточно простой и в то же время достаточно эффективный метод, в том случае, когда от валидации не нужно сложных взаимных проверок данных, поэтому он используется достаточно часто, так как не требует много времени на реализацию.

Валидация по исключения

```
public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
```

```
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        LoadData();
    }

    public void LoadData()
    {
        Person Vasya = new Person();
        Vasya.Name = "Вася";
        Vasya.Age = 20;

        tbName.DataContext = Vasya;
        tbAge.DataContext = Vasya;
    }
}
```

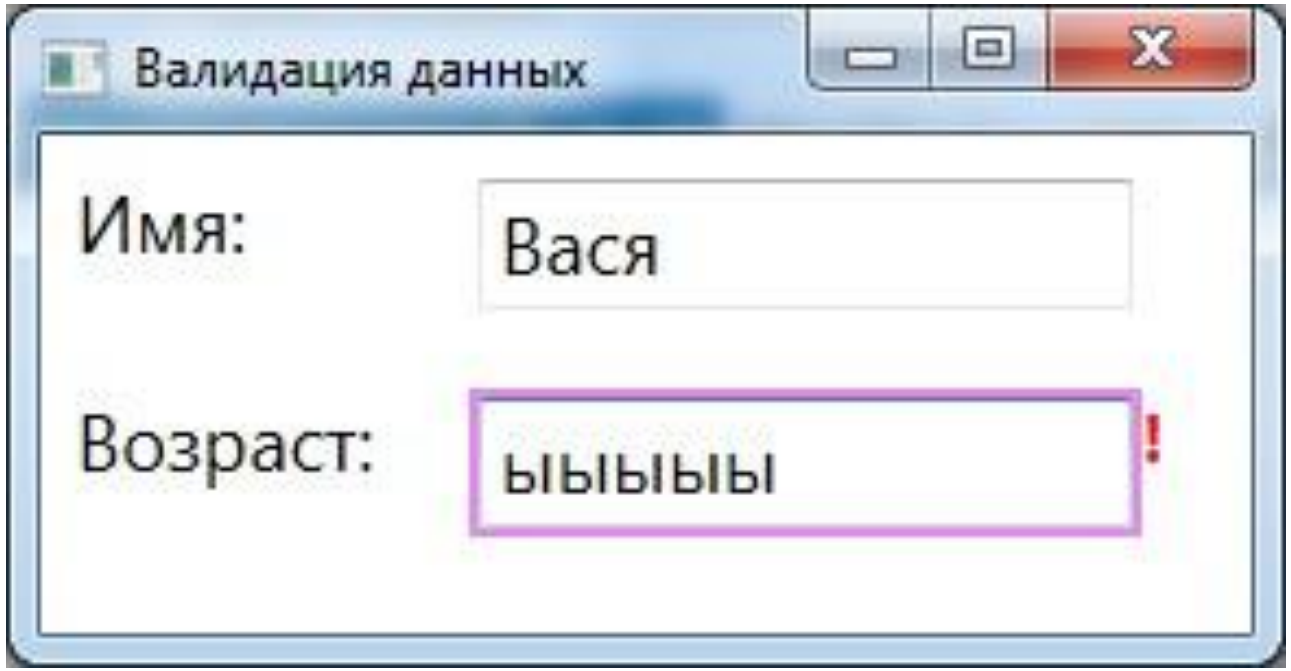
```
<TextBox x:Name="tbAge"  Grid.Column="1" Grid.Row="1" Height="30"
         Validation.ErrorTemplate="{StaticResource validationFailed}"
         Text="{Binding Path=Age, ValidatesOnExceptions=true, NotifyOnValidationError=True,
         UpdateSourceTrigger=PropertyChanged}">
</TextBox>
```

Валидация по исключениям

```
<Window.Resources>
    <ControlTemplate x:Key="validationFailed">
        <StackPanel Orientation="Horizontal">
            <Border BorderBrush="Violet" BorderThickness="2">
                <AdornedElementPlaceholder />
            </Border>
            <TextBlock Foreground="Red" FontSize="16" FontWeight="Bold">!</TextBlock>
        </StackPanel>
    </ControlTemplate>
</Window.Resources>
```

```
<TextBox x:Name="tbAge"  Grid.Column="1" Grid.Row="1" Height="30"
         Validation.ErrorTemplate="{StaticResource validationFailed}"
         Text="{Binding Path=Age, ValidatesOnExceptions=true, NotifyOnValidationError=True,
         UpdateSourceTrigger=PropertyChanged}">
</TextBox>
```

Валидация по исключениям



Валидация с использованием интерфейса IDataErrorInfo

Интерфейс IDataErrorInfo предоставляет один из лучших методов проверки данных. Для реализации данного метода нужно в классе нашего объекта реализовать данный интерфейс, который состоит из одного индексатора и свойства. Свойство используется для указания общей ошибки, а индексатор указывает на конкретное свойство, в котором произошла ошибка. Выглядит данный интерфейс следующим образом:

```
public interface IDataErrorInfo
{
string Error { get; }
string this[string propertyName] { get; }
}
```

Валидация с использованием интерфейса IDataErrorInfo

Индексатор и свойство возвращают null, если ошибки не было или строку с описанием ошибки, которую также можно предоставить пользователю, например с помощью подсказки или сообщения.

Валидация с использованием интерфейса IDataErrorInfo

В качестве правила валидации используется класс DataErrorValidationRule:

```
<TextBox Name="tbSex" Grid.Row="2" Grid.Column="1" Width="200" Margin="5">
    <TextBox.Text>
        <Binding Path="Sex" UpdateSourceTrigger="PropertyChanged">
            <Binding.ValidationRules>
                <DataErrorValidationRule />
            </Binding.ValidationRules>
        </Binding>
    </TextBox.Text>
</TextBox>
```

Валидация с использованием интерфейса IDataErrorInfo

Указать в разметке параметр ValidatesOnDataErrors=True

```
<TextBox x:Name="tbName" Grid.Row="0" Grid.Column="1" Width="200" Margin="5"
         Text="{Binding Path=Name, ValidatesOnDataErrors=True, UpdateSourceTrigger=PropertyChanged}"></TextBox>
<TextBox x:Name="tbAge" Grid.Row="1" Grid.Column="1" Width="200" Margin="5"
         Text="{Binding Path=Age, ValidatesOnDataErrors=True, UpdateSourceTrigger=PropertyChanged}"></TextBox>
```

Для программной установки контрола в состояние ошибки валидации используют два метода: Validation.MarkInvalid и Validation.ClearInvalid.

Для обозначения объекта как такого, в котором данные неверны:

```
Validation.MarkInvalid(name.GetBindingExpression(TextBox.TextProperty), new ValidationError(new
ExceptionValidationRule(), name.GetBindingExpression(TextBox.TextProperty)));
```

Для очистки состояния ошибки валидации:

```
Validation.ClearInvalid(name.GetBindingExpression(TextBox.TextProperty));
```