Каждый элемент управления WPF имеет средство, определяющее способ его визуализации. Это средство называется *шаблоном элемента управления* (controlTemplate). Внешний вид любого элемента может быть полностью переопределён. Например, можно изменить внешний вид кнопки, допустим, сделать её круглой. При этом, сохраняется логика работы элемента (т.е. кнопка остаётся кнопкой, которую можно нажимать для активизации действия).

Чтобы изменить внешний вид элемента, нужно создать для него шаблон в ресурсах:

```
<Window.Resources>
    <ControlTemplate x:Key="ButtonTemplate" TargetType="Button">
        <Border BorderBrush="#909090" BorderThickness="2" CornerRadius="2"
                TextBlock.Foreground="White" Background="#C0C0C0" Name="border1">
            <ContentPresenter HorizontalAlignment="Center" VerticalAlignment="Center"/>
        </Border>
    </ControlTemplate>
</Window.Resources>
```

Элемент ContentPresenter указывает местоположение содержимого кнопки. Аналогично стилям, для шаблонов задаётся свойство TargetType – тип элемента, для которого определяется данный шаблон (в данном примере – это кнопка, Button).

Чтобы применить шаблон к элементу, нужно задать для элемента свойство Template:

```
<Button Width="120" Height="24" Template="{StaticResource ButtonTemplate}">
    Кнопка с шаблоном
</Button>
```

Свойство Template, как и любое другое свойство элемента, можно задавать не только в самом элементе, но и в стиле:

```
<Style TargetType="Button">
    <Setter Property="Template" Value="{StaticResource ButtonTemplate}"/>
</Style>
```

Триггеры шаблонов

Кнопка в данном примере не меняет внешний вид при наведении курсора мыши. Это можно решить, добавив к шаблону триггеры (по аналогии с триггерами для стилей).

```
<ControlTemplate.Triggers>
    <Trigger Property="IsMouseOver" Value="True">
        <Setter TargetName="border1" Property="Background" Value="#D0D0D0" />
    </Trigger>
</ControlTemplate.Triggers>
```

В данном примере шаблон состоит из двух элементов: Border, ContentPresenter (конечно, их может быть и больше). При изменении свойства IsMouseOver нам нужно изменить свойство Background именно элемента Border. Для этого элементу назначено имя (border1), и данное имя используется для установки свойства TargetName объекта Setter.

Привязка шаблона (TemplateBinding)

Зададим для кнопки из примера свойство Foreground:

```
<Button Foreground="Yellow" Width="120" Height="24"
          Template="{StaticResource ButtonTemplate}">
    Кнопка с шаблоном
</Button>
```

Для стандартной кнопки (без переопределения шаблона) текст должен стать жёлтым. Однако в данном примере с использованием шаблона изменение свойства Foreground не оказывает никакого эффекта. Тем не менее, в WPF имеется инструмент, позволяющий шаблону извлечь значение из элемента управления, к которому применен шаблон. Механизм называется TemplateBinding (привязка шаблона).

Добавим привязку шаблона:

```
...
<Border BorderBrush="#909090" BorderThickness="2" CornerRadius="2"
  TextBlock.Foreground="{TemplateBinding Foreground}" Background="#C0C0C0" Name="border">
...
```

Это обеспечивает достижение требуемого эффекта: теперь при изменении свойства Foreground кнопки цвет текста будет меняться.

В некоторых случаях, например в триггерах шаблона, не разрешается использовать TemplateBinding. Следующий XAML-код вызовет ошибку в программе:

```
    <Trigger Property="IsMouseOver" Value="True">
        <Setter TargetName="border" Property="Background"
               Value="{TemplateBinding Background}" />
    </Trigger>
```

Однако, выражение с TemplateBinding можно заменить на более сложное выражение привязки, которое будет работать:

```
    <Trigger Property="IsMouseOver" Value="True">
        <Setter TargetName="border" Property="Background"
  Value="{Binding Path=Background, RelativeSource={RelativeSource TemplatedParent}}" />
    </Trigger>
```

Задание

1.

- Создать шаблон для круглой/овальной кнопки;
- Цвет кнопки задать градиентом из двух цветов;
- При наведении мышью цвета должны изменяться на более яркие.
- Расположить в окне приложения несколько круглых кнопок с текстом разных цветов.

2.

- Создать второй шаблон для круглой/овальной кнопки, одноцветной (без градиента);
- Создать стиль для кнопки, задающий первый шаблон по умолчанию, и второй шаблон при нажатии на кнопку (событие IsPressed);
- Применить стиль ко всем кнопкам.

3.

- Задать кнопкам различные значения Background;
- Во втором шаблоне цвет фона кнопки заменить на цвет, задаваемый в свойстве Background кнопки;
- В первом шаблоне аналогично заменить один из цветов градиента (для привязки понадобится вложенное свойство Background.Color).