Aula 3 - Estilos, modelos e Data Binding

Resumo:

- Estilos
- Controls and Data Templates
- *Data Binding*

Este guião pressupõe a resolução do guião da aula 2 até a alínea 2.5 (está disponível uma possível solução no ficheiro Aula_02_VS2013.zip).

3.1. Definição de um estilo

O WPF disponibiliza estilos que permitem modificar a aparência ou comportamento de um controlo. Os estilos permitem reutilizar elementos facilitando a criação, coerência e manutenção de interfaces gráficas. No exemplo da última aula as etiquetas (`labels`) cursos (DETI-Home) e disciplinas/ECTS (DETI-Cursos) usam a mesma formatação, vamos definir um estilo para aplicar nesses controlos.

Defina um estilo nas `Application ressources` (para que o mesmo esteja disponível para toda a aplicação) usando o, por exemplo, o código seguinte:

```
<!-- List header text style -->
<Style x:Key="labelStyle" TargetType="{x:Type Label}">
     <Setter Property="Foreground" Value="White" />
     <Setter Property="VerticalAlignment" Value="Center" />
     <Setter Property="HorizontalAlignment" Value="Left" />
</Style>
```

Para aplicar um estilo, coloque no elemento em questão o código seguinte retirando o código de formatação local que deixa de ser necessário.

```
Style="{StaticResource labelStyle}"
```

Aplique o estilo nas três `labels`. Da mesma forma defina um estilo para o `border` usados nesses `labels` e aplique-o nos locais adequados. Use a mesma formatação que foi usada na última aula.

Imagine que quere modificar o título “cursos” na página DETI-Home para usar negrito. É possível herdar propriedades entre estilos usando o código seguinte ao criar o estilo:

```
BasedOn="{StaticResource baseStyle}"
```

Crie um novo estilo que herda as propriedades originais do estilo `LabelStyle` mas com negrito. Aplique o estilo no local adequado.

Finalmente faça as modificações necessárias para utilizar como cor de fundo nesses `labels` a cor tijolo (#9a3324).

3.2. Utilização de estilos

Formate os vários elementos da página usando o código xaml que segue, repare que nem todos têm um estilo associado (`targetStyle`) veja a diferença ao nível do código.

```
 <!-- Header text style -->
<Style x:Key="headerTextStyle">
  <Setter Property="Label.VerticalAlignment" Value="Center"></Setter>
  <Setter Property="Label.FontFamily" Value="Trebuchet MS"></Setter>
  <Setter Property="Label.FontWeight" Value="Bold"></Setter>
  <Setter Property="Label.FontSize" Value="18"></Setter>
  <Setter Property="Label.Foreground" Value="#0066cc"></Setter>
</Style>
<!-- Button style -->
<Style x:Key="buttonStyle" TargetType="{x:Type Button}">
       <Setter Property="Width" Value="125" />
     <Setter Property="Height" Value="25" />
     <Setter Property="Margin" Value="0,10,0,0" />
     <Setter Property="HorizontalAlignment" Value="Right" />
</Style>
```

Utilize estes estilos para formatar alguns elementos da página DETI-Home, nomeadamente:

- O título da página “cursos do DETI” com o estilo `headerTextStyle`
- O botão de seleção “ver” com o estilo `buttonStyle`

Modifique também a página DETI-Cursos para utilizar os estilos no título da página “disciplinas” com o estilo `headerTextStyle`.

Depois destas modificações, a interface gráfica deve ser semelhante, mas garantindo coerência e facilitando modificações. Altere os estilos associados ao texto das tabelas (com border) para usar e a fonte `Courier New` com tamanho 14.

3.3. Modelos para controlos (control template)

No exemplo anterior o estilo é usado para definir propriedades. No caso de controlos podem ser usados os `control templates` para especificar como é desenhado o próprio controlo. Adicione no estilo do botão o `template` seguinte e veja o resultado.

```
       <Setter Property="Template">
       <Setter.Value>
           <ControlTemplate TargetType="{x:Type Button}">
         <Grid>
                     <Ellipse Fill="#9a3324" Stroke="black"/>
                   <ContentPresenter HorizontalAlignment="Center"
       VerticalAlignment="Center"/>
              </Grid>
           </ControlTemplate>
        </Setter.Value>
        </Setter>
```

Aplique este `template` ao botão ver. Se estiver associado ao botão uma tecla de atalho (com o _) note que para habilitar essa funcionalidade deve ativar e opção `RecognizesAccessKey="True"` no `ContentPresenter`.

Modifique o `control template` para que ao passar por cima do botão a cor do mesmo passe a outra cor a sua escolha. Para tal, depois da definição da grelha com a elipse adicione o código seguinte (repare que tem de dar um nome a elipse para a conseguir identificar).

```
<ControlTemplate.Triggers>
            <Trigger Property="IsMouseOver" Value="True">
               <Setter TargetName="MyEllipse" Property="Fill" Value="Green"/>
        </Trigger>
</ControlTemplate.Triggers>
```

Modifique algumas propriedades do `control template` (forma, cor, trigger) definindo um botão ao seu gosto. Pode usar outras formas (rectangle, polygon,…) cores ou triggers.

3.4. Exemplo de Data Binding

O data binding permite uma associação entre controlo e dados facilitando o preenchimento de controlos com informação. Nesta secção vamos criar a lista de cursos de uma forma mais dinâmica usando para tal data binding e algum código associado.

Comece por criar uma classe para permitir definir a lista de objetos para preencher a listbox dos cursos. Adicione a classe seguinte no código associada ao DETI-Home.

```
public class Curso
{
    private string _nome;
    private int _CodCurso;

    public int CodCurso
    {
        get { return _CodCurso; }
        set { _CodCurso = value; }
    }

    public string Nome
    {
        get { return _nome; }
        set { _nome= value; }
    }
}
```

Adicione ainda a classe seguinte que contêm uma função que retorna a lista de cursos (para compilar deve adicionar `using System.Collections.ObjectModel;`)

```
public class ListaCursos : ObservableCollection <Curso>
    {
        public ListaCursos()
        {
            Add(new Curso{Nome ="Computadores e Telemática", CodCurso=8240});
            Add(new Curso{Nome ="Electrónica e Telecomunicações", CodCurso=8204});
            Add(new Curso{Nome ="Engenharia Informática", CodCurso=8295, });
        }
       }
```

Modifique agora o código para fazer a ligação (binding) entre a listbox dos cursos o código agora adicionado.

Ao realizar um binding é necessário especificar:

- Um source: que indica qual o objeto ou classe ao qual o binding se refere. Muitas vezes o source é especificado no data context estando disponível para vários binding.

- Um path: se o binding é feito a um objeto, o path (ou xPath no caso do XML) indica qual o valor a indicar no binding. Se este não for especificado, o binding é feito a todo o objeto.

Para definir o source, deve especificar o DataContext no xaml através do namespace adicionando o código seguinte (deve substituir DETI pelo seu namespace).

```
xmlns:local="clr-namespace:DETI"
```

Deve agora tornar o source disponível usando uma chave associada ao mesmo. Pode fazer isso no grid.ressource usando o código:

```
<Grid.Resources>
   <local:ListaCursos x:Key="DETICursos"/>
</Grid.Resources>
```

Este código permite ter acesso ao objeto dentro do contexto da classe DETI-Home. Sendo possível fazer binding da informação a listbox utilizando o código seguinte:

```
<ListBox Name="cursosListBox" Grid.Column="2" Grid.Row="2"
ItemsSource="{Binding Source={StaticResource DETICursos}}">
```

Execute o código e veja o que ocorre.

Note que tem de retirar ou modificar o código da função Navigate pois a ListBox retorna agora um objeto do tipo Cursos e não um ListBoxItem.

3.5. Modelo para os dados (Data template)

A semelhança dos modelos para os controlos (control Templates), existem modelos para os dados (Data Templates) que permitem modificar o aspeto visual de itens de dados por exemplo nas listbox e nas combobox.

No exemplo anterior como não existe nenhum formato associado aos dados o Data Binding simplesmente converte os dados/objetos para String e apresenta o resultado.

Adicione o código seguinte para formatar os dados provenientes. Neste caso é indicado qual o campo a usar para preencher a lista.

```
<ListBox.ItemTemplate>
  <DataTemplate>
            <Label Content="{Binding Path=Nome}"></Label>
      </DataTemplate>
</ListBox.ItemTemplate>
```

Utilize um StackPanel para apresentar na listbox o nome e o código de curso associado.

Mais informação sobre estilos, templates, e data binding pode ser encontrada em:
https://msdn.microsoft.com/en-us/library/ms745683(v=vs.110).aspx
https://msdn.microsoft.com/en-us/library/ms752347(v=vs.110).aspx

3.6. Funções para a conversão (opcional)

Vamos agora associar uma cor a cada curso na classe criada ao efeito. Associe uma string com um valor hexadecimal a cada curso (por exemplo aquamarine #7FFFD4, darksalmon #E9967A e lemonchiffon #FFFACD).

Mudar a cor de fundo do `ListBoxItem` associado a cada curso usando data binding para associar a cor ao background dos labels.

Imagine que queria indicar ao lado do código o nome da cor utilizada, seria necessário fazer uma conversão entre a string hexadecimal e o nome da cor. Isso pode ser realizado com um value converter.

Adapte o código seguinte para fazer essa conversão

```
    public class ColorConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
        {
            // Do the conversion from hexadecimal to string
            return "lemonchiffon";
        }

        public object ConvertBack(object value, Type targetType, object
parameter, CultureInfo culture)
        {
            // Do the conversion from string to hexadecimal
            return null;

        }
       }
```

Para compilar é necessário adicionar a biblioteca seguinte:

```
using System.Globalization;
```

A utilização da função de conversão no xaml é indicada dentro do binding como segue:

```
Converter={StaticResource myConverter}
```

Sendo necessário criar uma instância da classe de conversão nas resources (por exemplo grid.resources) com o código seguinte:

```
<Grid.Resources>
    <local:ColorConverter x:Key="myConverter"/>
</Grid.Resources>
```