Bài 7

Kiểu hiển thị (Style) và Khuôn mẫu (Template)

WPF giới thiệu hai khái niệm là *Kiểu hiển thị* (Style) và *Khuôn mẫu* (Template) cho phép xây dựng các mẫu thuộc tính hiển thị áp dụng chung cho nhiều đối tượng UI trên giao diện người dùng. Bài giảng này tập trung giới thiệu hai khái niệm này và cách sử dụng chúng thông qua các ví dụ cụ thể.

1. Giới thiệu về Kiểu hiển thị (Style)

Thông thường, khi xây dựng một giao diện đồ hoạ, ta thường thiết lập cùng giá trị các thuộc tính hiển thị trên nhiều đối tượng UI khác nhau. Ví dụ, bạn muốn đặt tất cả các tiêu đề (Label) trong ứng dụng với phông chữ "Times New Roman", cỡ 14px, in đậm. Điều này có thể thực hiện dễ dàng với CSS trong một ứng dụng Web, nhưng không đơn giản đối với WinForm. WPF nhận ra sự cần thiết này và giải quyết bằng việc đưa ra thành phần 'Style'.

Thành phần 'Style' cho phép người lập trình lưu trữ một danh sách các giá trị thuộc tính vào một nơi thuận tiện. Nó tương tự như cách làm việc của CSS trong các ứng dụng Web. Thông thường, các Style được lưu trữ trong phần Resource hoặc một thư mục Resource riêng của project. Các thuộc tính quan trọng nhất của thành phần Style bao gồm BasedOn, TargetType, Setters và Triggers.

Được xem như một loại tài nguyên, Style có thể được định nghĩa ở bất kỳ phân cấp nào trong cây trực quan, ví dụ cho một StackPanel, Window hoặc thậm chí ở mức Application. Việc đặt khai báo Style lẫn với các mã chức năng XAML thường dễ gây nhầm lẫn khi mở rộng ứng dụng. Lời khuyên ở đây là không đặt khai báo Style trong App.xaml hay các file chức năng xaml, mà lưu chúng trong một file xaml tài nguyên riêng. Lưu ý rằng các tài nguyên có thể được chia nhỏ thành các file độc lập sao cho các file ảnh như jpeg có thể được lưu trữ riêng rẽ.

Một khi đã chia thành các file tài nguyên riêng thì vấn đề tiếp theo sẽ là việc làm sao để tìm tham chiếu tới tài nguyên bạn cần. Ở đây, ta dùng một giá trị khoá duy nhất: Khi định nghĩa một tài nguyên trong XAML, bạn định nghĩa một giá trị khoá duy nhất cho tài nguyên đó thông qua thuộc tính *x:Key*. Kể từ sau đó, bạn có thể tham chiếu tới tài nguyên này bằng việc sử dụng giá trị này.

Sau đây, các thuộc tính quan trọng trong Style sẽ được lần lượt giới thiệu.

1.1. Các thành phần thuộc tính trong Style

1.1.1 BasedOn

Thuộc tính này giống như tính chất kế thừa, trong đó, một Style kế thừa thuộc tính chung của một Style khác. Mỗi kiểu hiện thị chỉ hỗ trợ một giá trị BaseOn. Sau đây là một ví dụ nhỏ:

```
<!--Khai báo Style được kế thừa-->
<Style x:Key="Style1">
 ...
</Style>
<!--Khai báo Style kế thừa-->
<Style x:Key="Style2" BasedOn="{StaticResource Style1}">
 ...

</Style>
```

1.1.2 TargetType

Thuộc tính TargetType được sử dụng để giới hạn loại điều khiển nào được sử dụng Style đó. Ví dụ nếu ta có một Style với thuộc tính TargetType thiết lập cho nút bấm (Button), thì Style này sẽ không thể áp dụng cho kiểu điều khiển TextBox. Cách thiết lập thuộc tính này minh họa trong ví dụ sau:

```
<Style TargetType="{x:Type Button}">
....

</Style>
```

1.1.3 Setters

Setters cho phép thiết lập một sự kiện hay một thuộc tính với một giá trị nào đó. Trong trường hợp thiết lập một sự kiện, chúng liên kết với một sự kiện và kích hoạt hàm xử lý tương ứng. Trong trường hợp thiết lập một thuộc tính, chúng đặt giá trị cho thuộc tính đó.

Sau đây là một ví dụ về việc sử dụng EventSetters để liên kết sự kiện, trong đó, sự kiện nhắm chuột vào nút bấm (Click) được liên kết:

```
<Style TargetType="{x:Type Button}">
    <EventSetter Event="Click" Handler="b1SetColor"/>

</Style>
```

Tuy nhiên, Setter thường được dùng để thiết lập giá trị thuộc tính hơn cả. Ví dụ:

```
<!--Đặt thuộc tính màu vàng cho nền nút bấm-->
<Style TargetType="{x:Type Button}">
    <Setter Property="BackGround" Value="Yellow"/>

</Style>
```

1.1.4 Triggers

Mô hình thiết lập kiểu hiển thị và khuôn mẫu của WPF cho phép bạn định ra các Trigger bên trong Style của bạn. Trigger là đối tượng cho phép bạn áp dụng những thay đổi về thuộc tính giao diện khi những điều kiện nhất định (ví dụ khi một giá trị Property nào đó bằng true, hoặc một sự kiện nào đó xảy ra) được thoả mãn.

Ví dụ sau đây minh hoạ một Style có định danh được áp dụng cho điều khiển Button. Style này định nghĩa một thành phần Trigger, có tác dụng thay đổi thuộc tính màu chữ của nút bấm khi thuộc tính IsPressed (nút đang bị bấm xuống) là true.

```
<Style x:Key="Triggers" TargetType="Button">
    <Style.Triggers>
        <Trigger Property="IsPressed" Value="true">
        <Setter Property = "Foreground" Value="Green"/>
        </Trigger>
    </Style.Triggers>
</Style>
```

Một số dạng khác của Trigger sử dụng trong Style:

DataTrigger

DataTrigger Đại diện cho một Trigger áp dụng cho giá trị thuộc tính hoặc thực hiện hành động khi dữ liệu liên kết thoả mãn một điều kiện định trước. Trong ví dụ sau, DataTrigger được xác định sao cho nếu

như giá trị *Tỉnh* trong mục dữ liệu *Nơi làm việc* bằng "*HN*" thì màu chữ của mục dữ liệu tương ứng trong ListBox được tô đỏ:

```
<Style TargetType="ListBoxItem">
    <Style.Triggers>
       <DataTrigger Binding="{Binding Path=Tinh}" Value="HN">
          <Setter Property="Foreground" Value="Red" />
       </DataTrigger>
    </Style.Triggers>
</Style>
```

Có một loại Trigger đặc biệt sử dụng nhiều hơn một giá trị để kích hoạt hoạt động, có tên gọi là Multitrigger. Với loại Trigger này ta có thể thiết lập nhiều điều kiện trong một Trigger. Ví dụ:

```
<Style TargetType="ListBoxItem">
   <Style.Triggers>
      <MultiDataTrigger>
       <MultiDataTrigger.Conditions>
          <Condition Binding="{Binding Path=TenCongViec}" Value="CNTT" />
          <Condition Binding="{Binding Path=Tinh}" Value="HN" />
       </MultiDataTrigger.Conditions>
       <Setter Property="Background" Value="Cyan" />
      </MultiDataTrigger>
   </Style.Triggers>
</Style>
```

Trong ví dụ này, đối tượng dữ liệu buộc với điều khiển phải có TenCongViec="CNTT" *và* Tinh="HN", thì màu chữ của mục dữ liệu tương ứng trên ListBox được tô đỏ.

EventTrigger

EventTrigger là loại Trigger đặc biệt áp dụng cho một tập các hành động tương ứng với một sự kiện. Các EventTrigger đặc biệt ở chỗ chúng chỉ cho phép các hành động hoạt họa được kích hoạt. Chúng không cho phép các thuộc tính bình thường được thiết lập làm cơ sở như đối với các Trigger khác. Sau đây là một ví dụ của EventTrigger:

```
<EventTrigger RoutedEvent="Mouse.MouseEnter">
  <EventTrigger.Actions>
```

```
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation
              Duration="0:0:0.2"
              Storyboard.TargetProperty="MaxHeight"
              To="90"  />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger.Actions>
    </EventTrigger>

    <EventTrigger RoutedEvent="Mouse.MouseLeave">
      <EventTrigger.Actions>
        <BeginStoryboard>
          <Storyboard>
            <DoubleAnimation
              Duration="0:0:1"
              Storyboard.TargetProperty="MaxHeight"  />
          </Storyboard>
        </BeginStoryboard>
      </EventTrigger.Actions>
    </EventTrigger>
```

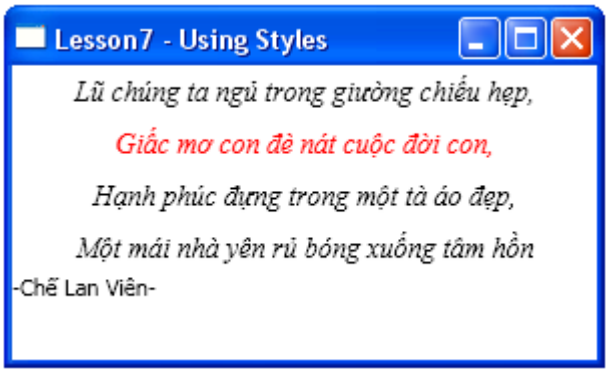
1.2 Một ví dụ đầy đủ về sử dụng Style

Sau đây là một ví dụ đầy đủ về việc sử dụng Style. Trong ví dụ minh hoạ này, hai Style được định nghĩa cho panel chính. Style thứ nhất quy định các thuộc tính tĩnh về phông chữ, áp dụng đối với đối tượng UI là Control. Style thứ hai kế thừa các thuộc tính này từ Style thứ nhất và chỉ áp dụng cho Label. Style thứ hai quy định thêm phản ứng của các đối tượng là Label trong StackPanel khi con trỏ chuột lướt qua, cụ thể, màu chữ sẽ chuyển đỏ. Sau đây là mã XAML tương ứng:

```
    <Window x:Class="Lesson7.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Lesson7 - Using Styles" Height="300" Width="300"
        >
      <!--Sử dụng Stack Panel làm Panel chính-->
```

```
        <StackPanel>
          <!--Khai báo tài nguyên trong StackPanel-->
          <StackPanel.Resources>
            <!--Trong trường hợp này, tài nguyên là hai Style:-->
            <!--(1) Style quy định về kiểu phông chữ, áp dụng với Control-->
            <Style x:Key="baseStyle" TargetType="{x:Type Control}">
              <Setter Property="FontFamily" Value="Times New Roman" />
              <Setter Property="FontSize" Value="12" />
              <Setter Property="FontStyle" Value="Italic" />
              <Setter Property="HorizontalAlignment" Value="Center" />
            </Style>

            <!--(2) Style kế thừa từ Style trước, quy định phản ứng với sự kiện
-->
            <Style BasedOn="{StaticResource baseStyle}" TargetType="{x:Type
Label}">
              <!--Khai báo trigger-->
              <Style.Triggers>
                <!--Sự kiện khi con trỏ chuột lướt qua-->
                <Trigger Property="IsMouseOver" Value="True">
                  <Setter Property="Foreground" Value="Red" />
                </Trigger>
              </Style.Triggers>
            </Style>
          </StackPanel.Resources>
          <!--Kết thúc khai báo tài nguyên-->

          <!--Khai báo phần tử trên giao diện-->
          <Label>Lũ chúng ta ngủ trong giường chiếu hẹp, </Label>
          <Label>Giấc mơ con đè nát cuộc đời con, </Label>
          <Label>Hạnh phúc đựng trong một tà áo đẹp, </Label>
          <Label>Một mái nhà yên rủ bóng xuống tâm hồn </Label>
          <TextBlock>-Chế Lan Viên-</TextBlock>
        </StackPanel>
      </Window>
```

Kết quả là:



Hình 7.1 - Sử dụng Style

Chú ý khi áp dụng một Style được thiết lập giá trị x:Key cho một đối tượng UI cụ thể, ta phải thiết lập thuộc tính Style trong khai báo đối tượng đó. Ví dụ:

```
      <Style x:Key="TitleText" TargetType="{x:Type TextBlock}">
            <Setter Property="FontFamily" Value="Times New Roman" />
            <Setter Property="FontSize" Value="12" />
            <Setter Property="FontStyle" Value="Italic" />
            <Setter Property="HorizontalAlignment" Value="Center" />
      </Style>

      <TextBlock Style="{StaticResource TitleText}">Đoạn text có áp dụng
Style</TextBlock>
      <TextBlock>Đoạn text không áp dụng Style</TextBlock>
```

Trong ví dụ trên, chỉ TextBlock có thiết lập thuộc tính Style tham chiếu đến Style có giá trị khoá TitleText (`Style="{StaticResource TitleText}"`) mới chịu tác dụng của Style này. Style trong TextBlock còn lại là ngầm định.

2. Giới thiệu về Khuôn mẫu (Template)

Bằng việc sử dụng Style, ta có thể tạo ra một diện mạo nhất quán và dễ sửa đổi cho giao diện ứng dụng. Tuy nhiên, đôi khi bạn muốn đi xa hơn. Chẳng hạn, bạn muốn các nút bấm không phải là hình chữ nhật như thường lệ mà là hình ellipse. Hay bạn muốn hiển thị một tập dữ liệu nhân viên trong một công ty, trong đó, mỗi bản ghi nhân viên lại được trình bày theo một định dạng xác

định. Bạn không thể đạt được điều này bằng những Setter căn bản trong Style. Trong trường hợp đó, bạn phải dùng đến khái niệm gọi là *Khuôn mẫu* (Template).

Trong WPF, có hai dạng khuôn mẫu được sử dụng: ControlTemplate dùng để định lại cấu trúc hiển thị cho điều khiển UI; và DataTemplate dùng để định ra cách thức hiển thị dữ liệu. Phần sau đây sẽ trình bày lần lượt hai dạng khuôn mẫu này.

2.1 ControlTemplate

2.1.1 ControlTemplate là gì?

Phần lớn các điều khiển đều bao gồm *diện mạo* và *hành vi*. Xét một nút bấm: diện mạo của nó là vùng nổi lên mà ta có thể bấm vào, trong khi hành vi là sự kiện Click được phát động để phản ứng với hành động nhắp chuột vào nút bấm đó.

Đôi khi có những điều khiển cung cấp các hành vi mà ta cần nhưng lại không có diện mạo mà ta mong muốn. Tới giờ, chúng ta có thể dùng các Setter của thành phần Style để thiết lập các giá trị thuộc tính có ảnh hưởng tới diện mạo của điều khiển. Tuy nhiên, để thay đổi cấu trúc của một điều khiển hoặc thiết lập giá trị thuộc tính cho các component có chứa một điều khiển, ta cần dùng đến ControlTemplate.

Trong WPF, ControlTemplate của một điều khiển định nghĩa diện mạo cho điều khiển đó. Bạn có thay đổi cấu trúc hay diện mạo của một điều khiển bằng cách định nghĩa một ControlTemplate mới cho dạng điều khiển đó. Trong trường hợp bạn không định nghĩa riêng một ControlTemplate cho điều khiển của bạn, thì một template ngầm định phù hợp với giao diện chung của hệ thống sẽ được sử dụng, giống như những gì ta nhìn thấy đối với một nút bấm truyền thống.

Một điều cần nhớ là khi bạn tạo một ControlTemplate cho điều khiển, bạn đang thay thế toàn bộ ControlTemplate của điều khiển đó. Ví dụ, bạn có thể định nghĩa ControlTemplate cho điều khiển Button như sau:

```
      <Style TargetType="Button">
          <!--Đặt giá trị true để không sử dụng bất kỳ giá trị thuộc tính nào
của theme hệ thống-->
```

```
        <Setter Property="OverridesDefaultStyle" Value="True"/>
        <!--Thiết lập khuôn dạng mẫu cho điều khiển Button-->
        <Setter Property="Template">
            <Setter.Value>
              <ControlTemplate TargetType="Button">
                <Grid>
                   <Ellipse Fill="Navy"/>
                  <!--Đánh dấu nơi bắt đầu đặt nội dung của Button: chính
giữa-->
                   <ContentPresenter HorizontalAlignment="Center"
                                     VerticalAlignment="Center"/>
                </Grid>
              </ControlTemplate>
            </Setter.Value>
         </Setter>

    </Style>
```

Khi áp dụng, nút bấm sẽ có dạng như một hình Ellipse:

Hình 7.2 – Tạo một điều khiển Button có dạng hình Ellipse sử dụng ControlTemplate

Chú ý rằng diện mạo của Button khi nó nhận được focus hoặc được bấm hiện thời đều thuộc vào phần diện mạo ngầm định của nút bấm mà ta đã thay thế. Vì ta đã thiết lập thuộc tính `OverridesDefaultStyle` bằng true, mọi thuộc tính ngầm định đều bị bỏ qua. Do đó, nếu cần thay đổi diện mạo của Button khi nhận được focus hay bị bấm, ta phải định nghĩa lại diện mạo trong những trường hợp này.

2.1.2 Một ví dụ về sử dụng ControlTemplate

Trong phần này, chúng ta cùng xây dựng một ControlTemplate định nghĩa một ListBox mà trong đó, các chỉ mục được sắp xếp theo chiều ngang (thay vì chiều dọc như thông thường) và có các góc được uốn cong. Sau đây là đoạn mã XAML minh hoạ:

```
<Window x:Class="Lesson7.Window3"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Lesson7 - ControlTemplate Example 02" Height="300" Width="300"
  >

<StackPanel>
  <!--Khai báo tài nguyên của panel chính-->
  <StackPanel.Resources>
    <!--Sử dụng một Style để chứa khai báo ControlTemplate-->
    <Style TargetType="ListBox">
      <!--Khai báo một Setter của thuộc tính Template-->
      <Setter Property="Template">
        <Setter.Value>
          <!--Khai báo định nghĩa ControlTemplate-->
          <ControlTemplate TargetType="ListBox">
            <!--Khai báo bán kính uốn góc và màu nền-->
            <Border CornerRadius="5" Background="Orange">
              <ScrollViewer HorizontalScrollBarVisibility="Auto">
                <!--Sử dụng một StackPanel sắp xếp theo chiều ngang-->
                <StackPanel Orientation="Horizontal"
                            VerticalAlignment="Center"
                            HorizontalAlignment="Center"
                            IsItemsHost="True"/>
              </ScrollViewer>
            </Border>
          </ControlTemplate>
        </Setter.Value>
      </Setter>
    </Style>
  </StackPanel.Resources>
  <!--Kết thúc khai báo tài nguyên-->
```

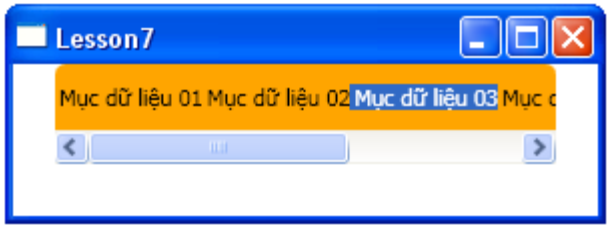
```
        <!--Khai báo ListBox-->
        <ListBox Width="250" Height="50">
          <ListBoxItem>Mục dữ liệu 01</ListBoxItem>
          <ListBoxItem>Mục dữ liệu 02</ListBoxItem>
          <ListBoxItem>Mục dữ liệu 03</ListBoxItem>
          <ListBoxItem>Mục dữ liệu 04</ListBoxItem>
          <ListBoxItem>Mục dữ liệu 05</ListBoxItem>
        </ListBox>
      </StackPanel>

    </Window>
```

Theo cách trên, bạn xây dựng một ControlTemplate thông qua sử dụng một Style, cụ thể là khai báo trong một Setter cho thuộc tính Template. Một cách khác nữa là bạn có thể gán trực tiếp thuộc tính Template của một điều khiển cho một ControlTemplate. Với cách này, ControlTemplate cần dùng phải được xây dựng trước, trong phần Resourse chẳng hạn, và được gán khoá định danh thông qua x:Key, và sau đó được sử dụng như một tài nguyên tĩnh (khai báo StaticResource).

Như bạn có thể thấy trong ví dụ trên, lớp ControlTemplate cũng có thuộc tính TargetType như đối với lớp Style. Tuy nhiên, cần lưu ý rằng, nếu ta xây dựng một ControlTemplate độc lập, với thuộc tính TargetType được thiết lập cho một kiểu điều khiển nào đó, thì ControlTemplate đó không được tự động áp dụng cho kiểu điều khiển này. Cũng lưu ý rằng thuộc tính TargetType là bắt buộc trong một khai báo ControlTemplate nếu như template đó có chứa thành phần ContentPresenter.

Trong ví dụ trên, một thuộc tính quan trọng cần có là IsItemsHost. Thuộc tính IsItemsHost được sử dụng để xác định đây là template của một điều khiển chứa các mục con, và các mục con sẽ được sắp xếp trong đó. Thiết lập thuộc tính này bằng true trong StackPanel có nghĩa là bất kỳ một mục nào được thêm vào ListBox sẽ được xếp vào StackPanel. Chú ý thuộc tính này chỉ có trong kiểu Panel.

Kết quả của đoạn mã trên như minh họa trong hình 7.3.



Hình 7.3 – Một ví dụ về xây dựng và sử dụng một ControlTemplate cho ListBox

2.2 DataTemplate

2.2.1 DataTemplate là gì?

DataTemplate được sử dụng để định ra cách thức hiển thị các đối tượng dữ liệu. Đối tượng DataTemplate đặc biệt hữu dụng khi bạn móc nối một điều khiển chứa mục con (ItemsControl) kiểu như ListBox với một danh mục dữ liệu. Không có sự định hướng cụ thể, một ListBox sẽ ngầm định hiển thị các đối tượng trong danh sách dưới dạng chuỗi ký tự. Với việc sử dụng DataTemplate, chúng ta có thể định khuôn dạng hiển thị của mỗi mục con trong ListBox với nhiều đặc tính trực quan như màu sắc, hình ảnh, phông chữ…

2.2.2 Một ví dụ sử dụng DataTemplate

Trong ví dụ này, thông tin về các nhân viên trong một văn phòng được hiển thị sử dụng DataTemplate. Trước hết, ta phải định nghĩa nguồn dữ liệu, cụ thể ở đây là danh sách nhân viên.

Để làm điều này, đầu tiên, ta xây dựng lớp nhân viên (Person), đơn giản bao gồm họ tên (Name) và ảnh chân dung (ImageRef). Sau đây là mã C#:

```
namespace Lesson7{

   /**
     * Định nghĩa lớp thành phần dữ liệu Person
     */
   public class Person
    {
        public Person(string name, string imageRef)
```

```
            {
                this.Name = name;
                this.ImageRef = imageRef;
            }

            private string _name;
            public string Name
            {
                get { return _name; }
                set { _name = value; }
            }

            private string _imageRef;
            public string ImageRef
            {
                get { return _imageRef; }
                set { _imageRef = value; }
            }
        }
    }
```

Tiếp theo, ta xây dựng lớp chứa danh sách nhân viên, giả sử có tên Staffs. Mã lệnh C# như sau:

```
    namespace Lesson7{
    public class Staffs
    {
            private List<Person> staffs;

            public IEnumerable<Person> StaffList
            {
                get { return staffs; }
            }

            public Staffs()
            {
                staffs = new List<Person>();
                staffs.Add(new Person("Mary", "mary.jpg"));
```

```
                staffs.Add(new Person("Johny", "johny.jpg"));
                staffs.Add(new Person("Olaf", "olaf.jpg"));
                staffs.Add(new Person("Scooby Doo", "scooby_doo.jpg"));
            }

    }

    }
```

Như đã thấy, lớp Staffs thực chất chứa đựng một danh sách có kiểu Person (biến staffs). Đối tượng của lớp này khi được tạo lập sẽ khởi tạo một danh sách định trước gồm 4 nhân viên có tên và đường dẫn ảnh tương ứng (Mary, Johny, Olaf và Scooby Doo). Danh sách nhân viên có thể truy nhập thông qua thuộc tính StaffList của lớp Staffs.

Tiếp theo, ta xây dựng giao diện chính bằng mã XAML:

```
    <Window x:Class="Lesson7.Window5"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:Lesson7"
        Title="Lesson 7 - WPF DataTemplate Example" Height="360" Width="530"
        WindowStartupLocation="CenterScreen">

      <Window.Resources>
        <!--Định nghĩa nguồn dữ liệu-->
        <local:Staffs x:Key="MyStaffList"/>
      </Window.Resources>

      <StackPanel>
        <StackPanel.Resources>
          <!--Định nghĩa cách hiển thị mục dữ liệu thông qua một file xaml
riêng rẽ-->
          <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
              <ResourceDictionary Source="/StaffDataTemplate.xaml" />
            </ResourceDictionary.MergedDictionaries>
          </ResourceDictionary>
        </StackPanel.Resources>
```

```
        <!-- The Person-items -->
        <ItemsControl x:Name="personItems"
            HorizontalAlignment="Stretch"
            Margin="10"
            VerticalAlignment="Center"
             Background="Orange"
            ItemsSource="{Binding Source={StaticResource MyStaffList},
Path=StaffList}"
            />
      </StackPanel>

    </Window>
```

Như đã thấy, phần tử UI chính được dùng trên giao diện là một ItemsControl, loại điều khiển cho phép hiển thị nhiều mục hiển thị con trong nó. Nội dung các mục hiển thị được gắn kết với một nguồn dữ liệu là thuộc tính StaffList của một đối tượng thuộc lớp Staffs có tên MyStaffList (`ItemsSource="{Binding Source={StaticResource MyStaffList}, Path=StaffList}"`). Đối tượng dữ liệu này được khai báo trong phần Resources của Window (`<local:Staffs x:Key="MyStaffList"/>`).

Trong khi đó, việc sử dụng DataTemplate để quy định cách thức hiển thị của mỗi mục dữ liệu trong StaffList lại được đặt trong một file .xaml riêng rẽ có tên StaffDataTemplate.xaml (`<ResourceDictionary Source="/StaffDataTemplate.xaml" />`). Trong trường hợp này, mặc dù hoàn toàn có thể định nghĩa DataTemplate trực tiếp trong mỗi điều khiển hay trong Resources của Window trong cùng một file .xaml, việc định nghĩa DataTemplate trong một file riêng như thế này cho phép ta dễ dàng sửa đổi và tái sử dụng DataTemplate ở nhiều form khác nhau mà không phải sao chép lại mã lệnh. Sau đây là mã XAML trong file StaffDataTemplate.xaml:

```
    <ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:local="clr-namespace:Lesson7" >

      <!-- Định nghĩa cách thức hiển thị cho mỗi mục dữ liệu nhân viên -->
      <DataTemplate DataType="{x:Type local:Person}">
```

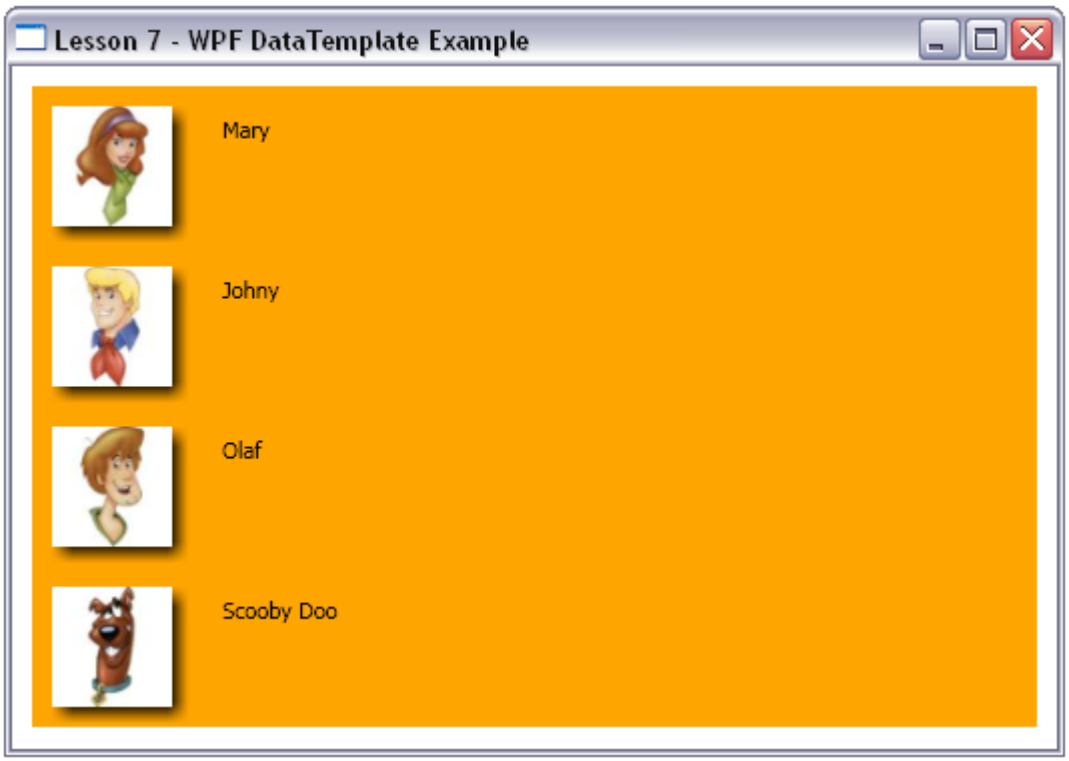
```
        <DataTemplate.Resources>
          <!--Khai báo một đối tượng chuyển đổi đường dẫn ảnh thành ảnh-->
          <local:PersonImageConverter x:Key="imageConverter" />
        </DataTemplate.Resources>


        <StackPanel Background="Orange" Orientation="Horizontal">
              <!-- Hiển thị Hình ảnh -->
              <Image Margin="10" Width="60" Height="60"
                     Source="{Binding Path=ImageRef,
                            Converter={StaticResource imageConverter}}">
                <Image.BitmapEffect>
                  <DropShadowBitmapEffect />
                </Image.BitmapEffect>

              </Image>

              <!-- Hiển thị Tên -->
              <TextBlock x:Name="personName"
                         Text="{Binding Name}"
                         Padding="15,15"
                         Foreground="Black" />

        </StackPanel>

      </DataTemplate>

    </ResourceDictionary>
```

DataTemplate được định nghĩa ở đây chỉ được dùng với kiểu dữ liệu thuộc lớp Person (`DataType="{x:Type local:Person}"`). Cách thức hiển thị tên nhân viên được khai báo trong một phần tử TextBlock mà thuộc tính Text được liên kết với thuộc tính Name của mỗi đối tượng Person (`Text="{Binding Name}"`). Trong khi đó, cách thức hiển thị ảnh của nhân viên được khai báo trong một phần tử Image, trong đó, phần nguồn ảnh được liên kết với thuộc tính ImageRef (`Source="{Binding Path=ImageRef,Converter={StaticResource imageConverter}}"`). Ở đây, để hiển thị hình ảnh của nhân viên thay vì đường dẫn tới ảnh, ta xây dựng một lớp chuyển đổi có tên PersonImageConverter. Mã lệnh C# cho lớp này như sau:

```
namespace Lesson7{

public class PersonImageConverter : IValueConverter
    {
        #region IValueConverter Members

        /// <summary>
        /// Hàm chuyển đổi từ đường dẫn ảnh sang đối tượng Bitmap
        /// </summary>
        public object Convert(object value, Type targetType,
            object parameter, System.Globalization.CultureInfo culture)
        {
            string imageName = value.ToString();
            Uri uri = new Uri(imageName, UriKind.RelativeOrAbsolute);
            BitmapFrame source = BitmapFrame.Create(uri);

            return source;
        }

        public object ConvertBack(object value, Type targetType,
            object parameter, System.Globalization.CultureInfo culture)
        {
            throw new NotImplementedException();
        }

        #endregion

    }

}
```

Lưu ý các ảnh được đặt trong thư mục chứa file chạy của chương trình.

Kết quả là:



Hình 7.4 – Hiển thị danh sách nhân viên sử dụng DataTemplate

Câu hỏi Ôn tập

1. Thành phần Style quy định các thuộc tính hiển thị định sẵn có thể được áp dụng cho:

a. Chỉ một đối tượng điều khiển UI duy nhất

b. Tất cả các đối tượng thuộc một lớp UI cụ thể (ListBox, TextBox, vv) quy định bởi thuộc tính TargetType trong Style đã định (trong trường hợp thành phần Style không được đặt khóa định danh x:Key)

c. Các đối tượng UI thuộc các lớp khác nhau nhưng cùng kế thừa từ một lớp UI (ví dụ, Control) quy định bởi thuộc tính TargetType trong thành phần Style đã định; và có thuộc tính Style tham chiếu đến khóa của Style đã định (trường hợp Style có đặt khóa định danh x:Key)

d. b hoặc c tương ứng với từng trường hợp Style đã định có khóa định danh x:Key hay không

Trả lời: d

2. Thuộc tính Triggers trong thành phần Style quy định:

a. Hàm xử lý sự kiện tương ứng với một sự kiện xảy ra trên đối tượng UI có áp dụng Style

b. Những thay đổi về thuộc tính hiển thị khi những điều kiện nhất định trên đối tượng UI có áp dụng Style được thoả mãn

Trả lời: b

3. Sử dụng thuộc tính BaseOn, một Style có thể kế thừa từ:

a. Duy nhất một Style cơ sở

b. Nhiều Style cơ sở tùy thuộc vào giá trị thiết lập cho BaseOn

Trả lời: a

4. Loại Trigger nào sau đây cho phép thiết lập nhiều điều kiện cho một dạng thay đổi thuộc tính hiển thị:

a. EventTrigger

b. MultiTrigger

c. DataTrigger

Trả lời: b

5. Sự khác biệt giữa Style và ControlTemplate áp dụng cho một lớp đối tượng UI là gì?

Trả lời: Style cho phép thiết lập giá trị các thuộc tính hiển thị *sẵn có* (màu nền, phông chữ, vân vân) của một lớp đối tượng điều khiển UI, trong khi ControlTemplate cho phép quy định lại cấu trúc hiển thị của lớp đối tượng UI (hình dạng, cách sắp xếp các phần tử con, vân vân).

6. Trong một ứng dụng WPF, khai báo Style và Template có thể được đặt ở đâu?

a. Đặt trong phần Resources của bất kỳ mức phân cấp nào trong cây trực quan trong cùng file .xaml khai báo giao diện

b. Trong một file .xaml riêng và được tham chiếu trong file giao diện có sử dụng Style hoặc Template thông qua khai báo đường dẫn trong phần tử `<ResourceDictionary>`

c. Một trong hai phương án trên

Trả lời: c

Tài liệu Tham khảo

1. WPF Styles and Templates, http://homepage.ntlworld.com/herring1/styletemp.html
2. WPF: A Beginner's Guide - Part 6 of n, http://www.codeproject.com/KB/WPF/BeginWPF6.aspx
3. WPF Styles and Control Templates, http://en.csharp-online.net/WPF_Styles_and_Control_Templates
4. .NET 3.0 Crash Course - Part 6: WPF Styles and Control Templates, http://devlicio.us/blogs/rob_eisenberg/archive/2006/12/03/net-3-0-crash-course-part-6-wpf-styles-and-control-templates.aspx
5. WPF DataBinding, Styles and DataTemplates, http://www.codegod.de/WebAppCodeGod/wpf-databinding-styles-and-datatemplates-AID406.aspx
6. Data Templating Overview, http://msdn.microsoft.com/en-us/library/ms742521.aspx