Bài 10. Phát triển ứng dụng giao diện đồ họa (GUI)

- **Mục đích:** Bài học này giới thiệu phương pháp xây dựng, sử dụng Menu, Toolbar, Style, Template bằng ngôn ngữ XAML trong ứng dụng WPF.

- **Yêu cầu:** Sau khi học xong bài này sinh viên có thể tạo được một số ứng dụng tổng hợp Menu và Toobar, cho phép xây dựng các mẫu thuộc tính hiển thị áp dụng chung cho nhiều đối tượng UI trên giao diện người dùng.

- **Hình thức tổ chức dạy học:** Thực hành, tự học

- **Thời gian:** Thực hành(online: 4, offline: 3) Tự học, tự nghiên cứu: 04

- **Nội dung:**

+ NỘI DUNG BÀI HỌC ONLINE

1. Xây dựng thực đơn và sử dụng thực đơn 2
   1.1. Xây dựng thực đơn với các Menu Item đơn giản 3
   1.2. Menu Item với trạng thái Checked và Unchecked 6
   1.3. Menu Item với biểu tượng hình ảnh 8
2. Xây dựng và sử dụng thanh công cụ (Toolbar) 9
   2.1. Nút công cụ thông thường 9
   2.2. Nút công cụ có trạng thái 11
3. Ví dụ tổng hợp về xây dựng Menu và Toolbar 13
   3.1. Tạo ứng dụng WPF 14
   3.2. Mã XAML của giao diện 14
   3.3. Mã lệnh C# xử lý các sự kiện 16

+ NỘI DUNG BÀI HỌC OFFLINE

4. Kiểu hiển thị (Style) và Khuôn mẫu (Template) 17
   4.1. Giới thiệu về Kiểu hiển thị (Style) 17
   4.2. Giới thiệu về Khuôn mẫu (Template) 22
Tài liệu Tham khảo 31

Thực đơn (Menu) và thanh công cụ (Toolbar) là một trong những thành phần quan trọng của cửa sổ, chúng chứa đựng các chức năng chính của chương trình mà người dùng có thể thực hiện. Thanh thực đơn chứa hầu hết tất cả chức năng chính của chương trình, tổ chức theo dạng phân cấp, trong khi thanh công cụ thường chứa một số chức năng thiết yếu mà người dùng hay quan tâm dưới dạng các biểu tượng hình ảnh để người dùng có thể thao tác một cách nhanh chóng.

Hình 10.1 minh họa cửa sổ chương trình với thanh thực đơn và thanh công cụ.

Bài này giới thiệu phương pháp xây dựng và sử dụng Thực đơn và Thanh công cụ bằng ngôn ngữ XAML trong ứng dụng WPF từ những ví dụ đơn giản là các mục menu (Menu Item) thông thường (không có biểu tượng và trạng thái) đến những ví Menu Item với biểu tượng hình ảnh (Icon) và các trạng thái (Checked và UnChecked).

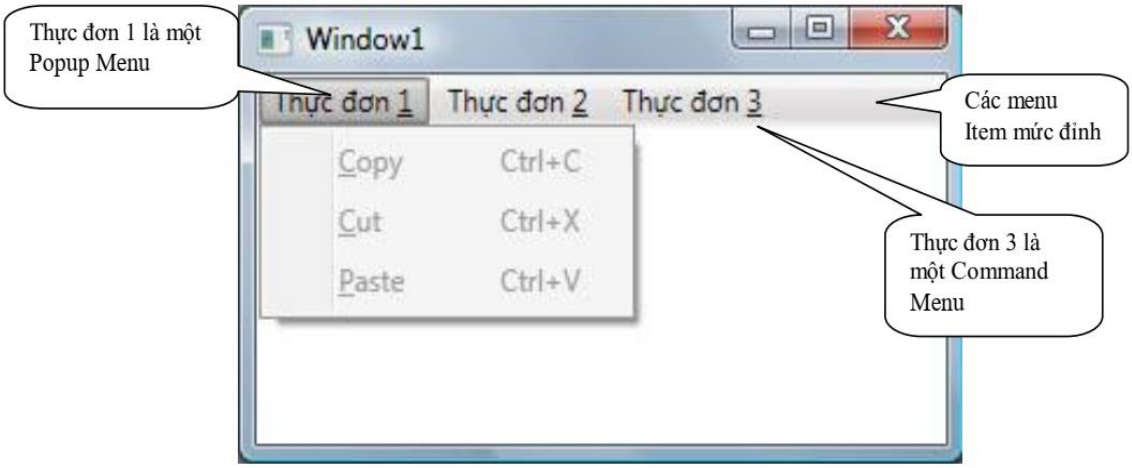
1. Xây dựng thực đơn và sử dụng thực đơn

Thực đơn (*Menu*) là điều khiển gồm nhiều phần tử được tổ chức dưới dạng phân cấp. Thanh thực đơn thường nằm trên đỉnh cửa số (dưới thanh tiêu đề). Các phần tử thực đơn (*Menu Item*) xuất hiện trên thanh thực đơn còn được gọi là Menu Item mức đỉnh. Mỗi Menu Item mức đỉnh có thể chứa nhiều Menu Item cấp dưới (Sub Menu) hoặc được gắn trực tiếp với các bộ quản lý sự kiện (Event handler) như sự kiện *Click* hay các lệnh của hệ thống được xây dựng sẵn (như Copy, Cut, Paste,..). Tương tự như vậy, mỗi Menu Item cấp dưới lại có thể chứa nhiều Menu Item cấp dưới của chính nó. Khi một Menu Item chứa các Menu Item cấp dưới thì thường được gọi là Popup Menu, các Menu Item cấp dưới sẽ xuất hiện khi người dùng nhấn chuột lên Popup Menu. Nếu Menu Item được gắn trực tiếp với với bộ quản lý sự kiện hay một lệnh của hệ thống thì được gọi là Command Menu, nó sẽ thực thi một câu lệnh mong muốn khi người dùng nhấn chuột hoặc nhấn phím tắt (ký tự trên bàn phím gắn với Menu Item) để chọn nó.

Ta sẽ tìm hiểu từng bước xây dựng và sử dụng menu bắt đầu từ Menu với các Menu Item đơn giản, tiếp đến là các Menu Item có trạng thái (Checked, UnChecked) và Menu Item có biểu tượng hình ảnh.

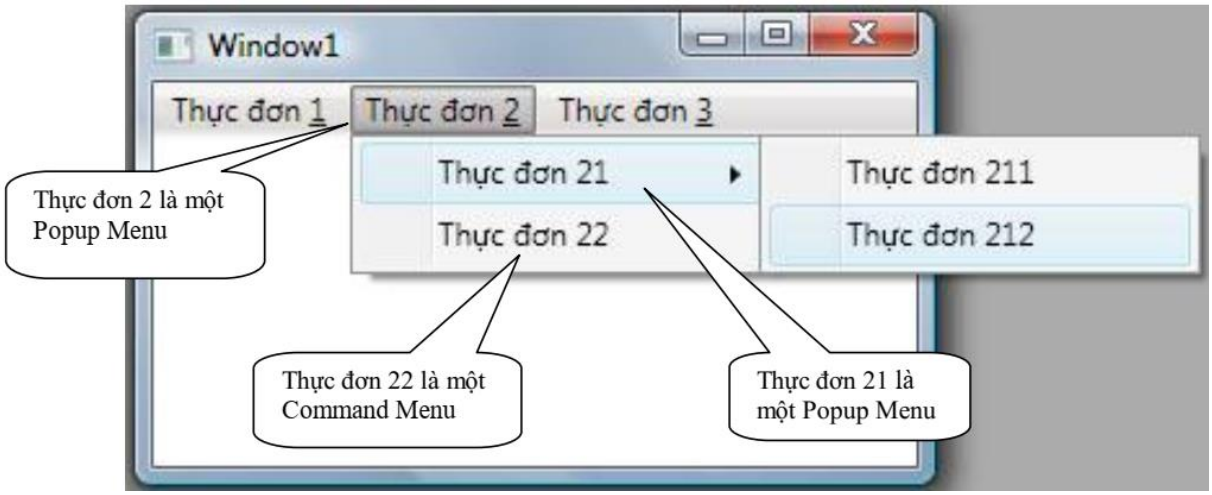
1.1.Xây dựng thực đơn với các Menu Item đơn giản

Trong phần này ta tìm hiểu từng bước xây dựng một thanh menu đơn giản gồm 3 Menu Item mức đỉnh là Thực đơn 1, Thực đơn 2 và Thực đơn 3. Trong đó, Thực đơn 1 và Thực đơn 2 là các Menu Popup (có chứa các menu con), Thực đơn 3 là loại Command Menu Item, nó không chứa Menu con mà sẽ thực thi một câu lệnh khi ta nhấn chuột vào nó như minh họa ở Hình 10.2.



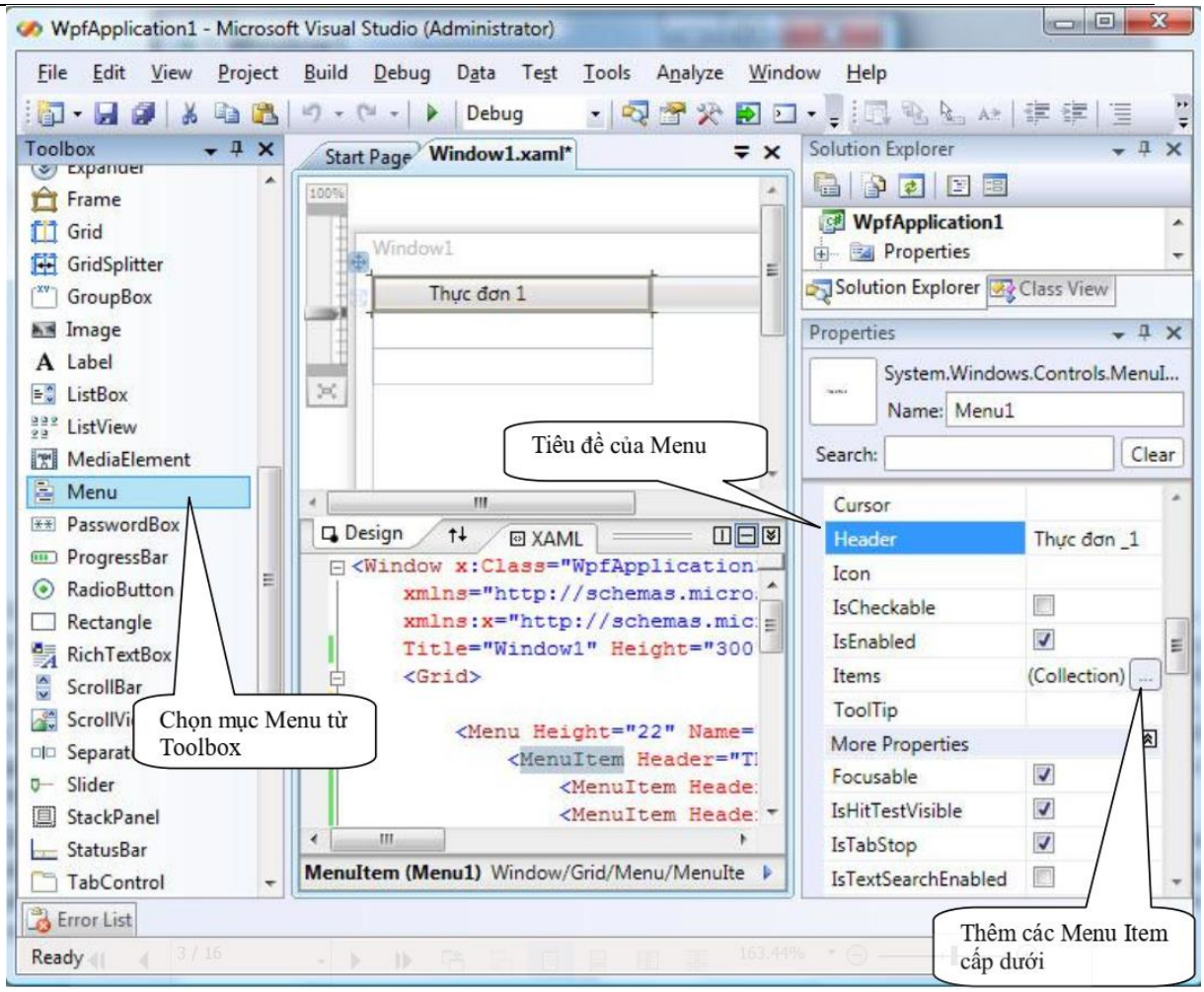
Hình 10.2. Một ví dụ về cửa số với thanh thực đơn

Hình 10.3 minh họa Thực đơn 2 có hai Menu Item cấp dưới là Thực đơn 21 và Thực đơn 22. Trong đó, Thực đơn 21 là Popup Menu chứa hai thực đơn cấp dưới của nó là Thực đơn 211 và Thực đơn 212, Thực đơn 22 là Command Menu Item.



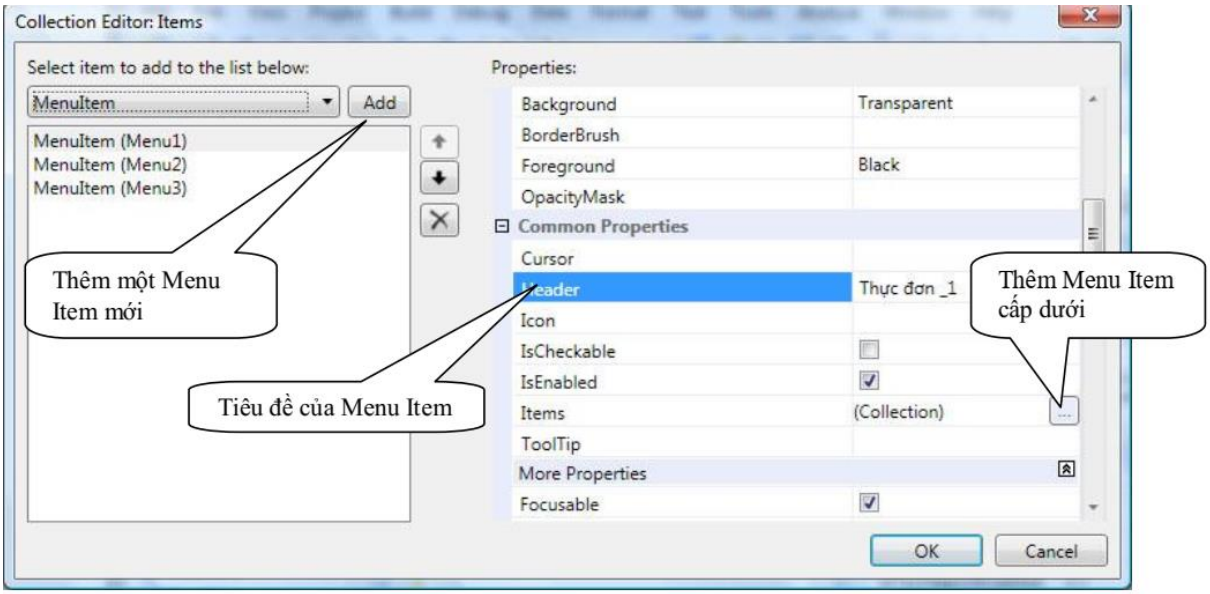
Hình 10.3. Ví dụ về thực đơn cấp dưới là một Popup Menu

Ta có thể tạo menu bằng công cụ trực quan hoặc gõ trực tiếp mã lệnh XAML. *Tạo Menu bằng công cụ trực quan.* Xem hình 10.4 minh họa công cụ tạo Menu cho ứng dụng WPF. Chọn menu từ ToolBox, sau đó Nhấn chuột lên cửa sổ để tạo Menu mới. Tiếp đến nhấn nút trên thanh Properties củaMenu để mở hộp thoại quản lý các Menu Item như hình 1.5.



Hình 10.4 Công cụ soạn thảo Menu cho ứng dụng WPF

Hình 10.5 là hộp thoại quản lý các Menu Item, nhấn nút để thêm Menu Item và nhập tiêu đề cho Menu Item ở mục Header. Nếu muốn thêm các Menu Item cấp dưới của Menu Item hiện tại, nhấn nút .



Hình 10.5. Hộp thoại Quản lý các Menu Item

Sử dụng mã XAML để tạo thực đơn

Đoạn mã trình Menu trên bằng XAML:

```
<Grid>
<Menu Height="22" Name="menu1" VerticalAlignment="Top" >
<MenuItem Header="Thực đơn _1" Name="Menu1">
<MenuItem Header="_Copy" Command="ApplicationCommands.Copy"
ToolTip="Copy văn bản"/>
<MenuItem Header="_Cut" Command="ApplicationCommands.Cut" ToolTip="Cắt
văn bản"/>
<MenuItem Header="_Paste" Command="ApplicationCommands.Paste" ToolTip="Dán văn
bản"/>
</MenuItem>
<MenuItem Header="Thực đơn _2" Name="Menu2">
<MenuItem Header="Thực đơn 21">
<MenuItem Header="Thực đơn 211" Click="MenuItem211_Click" />
<MenuItem Header="Thực đơn 212" Click="MenuItem212_Click" />
</MenuItem>
<MenuItem Header="Thực đơn 22" Click="MenuItem22_Click" />
</MenuItem>
<MenuItem Header="Thực đơn _3" Click="MenuItem3_Click" Name="Menu3" />
</Menu>
</Grid>
```

Thanh Menu được bắt đầu bằng thẻ <Menu> và kết thúc bằng thẻ đóng </Menu> . Có nhiều thuộc tính của thẻ này, trong ví dụ trên thì Height="22" : Chiều cao menu là 22 pixel. Name="menu1" : Tên của menu là menu1. Tên menu được mã trình C# sử dụng để quản lý nó. VerticalAlignment="Top" : Menu được căn để nằm bên trên của Grid chứa nó.

Các *Popup Menu* được tạo bởi thẻ <MenuItem> và kết thúc bằng thẻ đóng </MenuItem> . Giữa cặp thẻ này là các thẻ <MenuItem> khác để tạo nên các Menu Item cấp dưới của nó.

Các *Command Menu* thì được tạo bởi thẻ <MenuItem/>, không có thẻ đóng. Một số thuộc tính cơ bản của Menu Item bao gồm Header="Thực đơn _1": Tiêu đề hay nhãn của Menu Item. Dấu gạch dưới đặt trước ký tự sẽ được sử dụng làm *phím tắt* khi *kết hợp với phím Alt* để gọi Menu Item bằng bàn phím. Trong ví dụ này thì ký tự 1 được dùng làm phím tắt cho Menu Item "Thực đơn 1", ký tự được dùng làm phím tắt sẽ được hiển thị với dấu gạch chân khi người dùng nhấn phím Alt để mở Menu. Name="Menu1": Tên của Menu Item, cần thiết cho mã trình C# có thể can thiệp vào Menu Item.

ToolTip="Cắt văn bản": Lời chú thích cho Menu Item khi di chuột qua. Đối với các Command Menu, có hai cơ chế thực thi lệnh khi chọn Menu. Nếu muốn gắn Command Menu với các lệnh có sẵn của hệ thống như: Copy, Cut, Paste,.. thì ta sử dụng thuộc tính Command của Menu Item. Ví dụ, lệnh <MenuItem Header="_Copy" Command="ApplicationCommands.Copy"/> làm cho Menu Item Copy này sẽ thực hiện công việc copy dòng văn bản đang được chọn trong cửa sổ vào bộ nhớ đệm. Chú ý, các lệnh của hệ thống bắt đầu bằng ApplicationCommands. Nếu muốn gắn Command Menu với các hàm xử lý sự kiện tự định nghĩa thì sử dụng

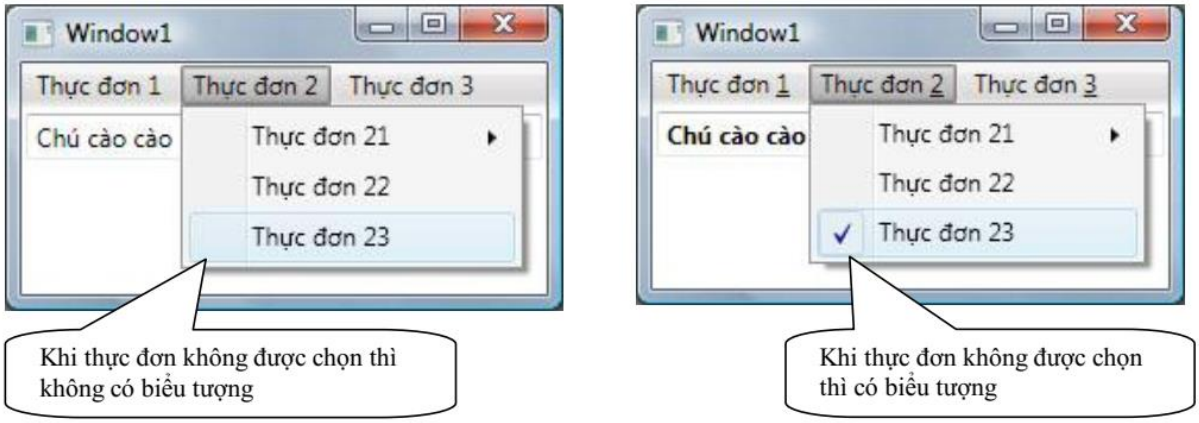
thuộc tính Click của MenuIte m. Ví dụ, <MenuItem Header="Thực đơn 211" Click="MenuItem211_Click" /> để yêu cầu khi chọn "Thực đơn 211" thì sẽ gọi hàm MenuItem211_Click.

Dưới đây là đoạn mã trình C# khai báo các hàm xử lý sự kiện khi nhấn vào các Menu khai báo bằng XAML trên. Khi các menu được chọn thì các hàm tương ứng được khai báo trong thuộc tính Click sẽ được gọi và làm nhiệm vụ đơn giản làm hiển thị hộp thông báo.

```
namespace WpfApplication1
{
/// <summary>
/// Interaction logic for Window1.xaml
/// </summary>
public partial class Window1 : Window
{
public Window1()
{
InitializeComponent();
}
//Hàm xử lý sự kiện nhấn Menu "Thực đơn 211"
private void MenuItem211_Click(object sender, RoutedEventArgs e)
{
MessageBox.Show("Bạn chọn Menu 211");
}
//Hàm xử lý sự kiện nhấn Menu "Thực đơn 212"
private void MenuItem212_Click(object sender, RoutedEventArgs e)
{
MessageBox.Show("Bạn chọn Menu 212");
}
//Hàm xử lý sự kiện nhấn Menu "Thực đơn 22"
private void MenuItem22_Click(object sender, RoutedEventArgs e)
{
MessageBox.Show("Bạn chọn Menu 22");
}
//Hàm xử lý sự kiện nhấn Menu "Thực đơn 3"
private void MenuItem3_Click(object sender, RoutedEventArgs e)
{
MessageBox.Show(" Bạn chọn Menu 3");
}
}
}
```

1.2. Menu Item với trạng thái Checked và Unchecked

Khi làm việc với Menu, đôi khi ta có những chức năng với đặc thù có hai trạng thái On/Off. Ví dụ như chương cần có một Menu Item để làm cho một Textbox hiển thị ở dạng chữ đậm và chữ thường, người dùng mong muốn Menu Item thể hiện được trạng thái On/Off tương ứng với kiểu chữ (chữ đậm/chữ thường) trên Textbox. Điều khiển Menu của WPF cung cấp cho chúng ta loại Menu Item với hai trạng thái Checked và UnChecked.



Hình 10.6. Minh họa Menu Item với trạng thái Checked và UnChecked

Để tạo ra Menu Item có trạng thái, ta sử dụng thuộc tính IsCheckable="True" của Menu Item.

Mã lệnh tạo "Thực đơn 23" như sau:

```
<MenuItem Header="Thực đơn _2" Name="Menu2">
<MenuItem Header="Thực đơn 21">
<MenuItem Header="Thực đơn 211" Click="MenuItem211_Click" />
<MenuItem Header="Thực đơn 212" Click="MenuItem212_Click" />
</MenuItem>
<MenuItem Header="Thực đơn 22" Click="MenuItem22_Click" />
<!--Thực đơn có trạng thái Checked và UnChecked-->
<MenuItem Header="Thực đơn 23" IsCheckable="True" Checked="Menu23_Checked"
Unchecked="Menu23_Unchecked"/>
</MenuItem>
```

Đối với Menu Item có trạng thái, mỗi khi Menu được chọn sẽ phát sinh một trong hai sự kiện Checked và Uncheked tương ứng. Ta sử dụng các thuộc tính Checked và Unchecked để gắn các hàm xử lý sự kiện cần được thực thi. Ở ví dụ trên, hàm Menu23_Checked sẽ được gọi khi Menu "Thực đơn 23" được đánh dấu, và hàm Menu23_Unchecked được gọi khi Menu "Thực đơn 23" được bỏ dấu chọn. Mã nguồn minh họa của hai hàm trên như sau:

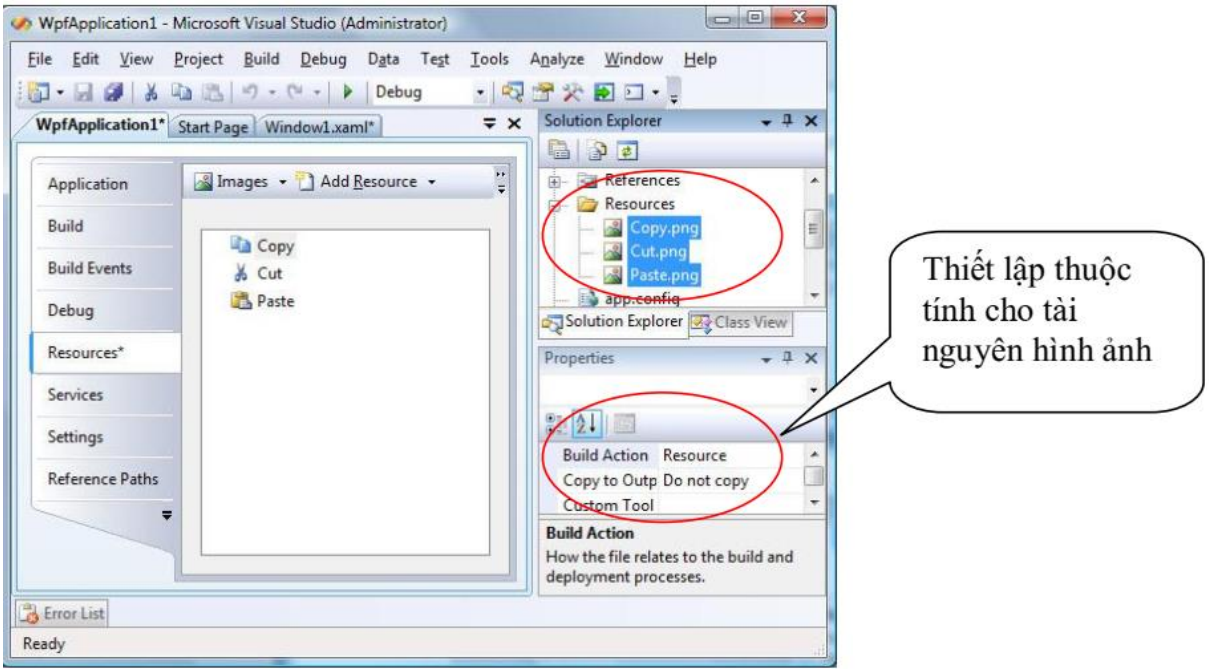
```
public partial class Window1 : Window
{
public Window1()
{
InitializeComponent();
}
//.....................................................
//Hàm xử lý sự kiện khi Menu "Thực đơn 23" được đánh dấu chọn
private void Menu23_Checked(object sender, RoutedEventArgs e)
{
//Thiết lập thuộc tính chữ đậm cho textBox1
textBox1.FontWeight = FontWeights.Bold;
}
//Hàm xử lý sự kiện khi Menu "Thực đơn 23" được bỏ dấu chọn
private void Menu23_Unchecked(object sender, RoutedEventArgs e)
{
```

```
//Thiết lập thuộc tính chữ thường cho textBox1
textBox1.FontWeight = FontWeights.Normal;
}
}
```

1.3. Menu Item với biểu tượng hình ảnh

Với các ứng dụng xây dựng trên nền .Net, công việc xây dựng Thực đơn và Thanh công cụ với biểu tượng hình là khá đơn giản, có thể sử dụng công cụ thiết kế giao diện trực quan. Ví dụ, muốn tạo menu có biểu tượng hình ảnh (icon), chúng ta thêm Menu Trip vào form, sau đó thêm các mục cho thực đơn (Menu Item). Nhấn chuột phải lên từng mục và chọn "*Set Image*" là thành công. Tuy nhiên, khi xây dựng ứng dụng WPF, không có mục chọn "*Set Image*" khi nhấn chuột phải vào một mục trong thực đơn. Chúng ta phải nạp các tệp hình ảnh, biểu tượng và tài nguyên (Resource) của ứng dụng và viết một số lệnh XAML để gắn biểu tượng cho Menu Item.
Nạp các tệp hình ảnh, biểu tượng vào tài nguyên của ứng dụng.

i. Trên thanh thực đơn Visual Studio, chọn Project → Properties sẽ hiện ra bảng cài đặt các thông số cài đặt cho ứng dụng.
ii. Chọn mục resources.
iii. Trong mục Add Resource chọn Add Existing File nếu đã có sẵn File biểu tượng hình ảnh trên máy hoặc chọn New Images hay Add New Icon tùy ý.
iv. Chú ý, sau khi thêm được các File hình ảnh biểu tượng vào tài nguyên, để các điều khiển trên cửa sổ như Menu, Toolbar sử dụng được chúng, ta phải thiết lập thuộc tính *'Build Action :Resource'* và *'Copy to Output Directory : Do not copy'*.



Hình 10.7. Thêm tài nguyên và thiết lập thuộc tính cho tài nguyên ảnh

Viết mã lệnh XAML gắn biểu tượng cho các Menu Item.

```
<StackPanel>
<!--Khai bao Tài nguyên cần sử dụng-->
```

```
<StackPanel.Resources>
<Image x:Key="copy" Source="Resources/Copy.png" Width="16" Height="16"></Image>
<Image x:Key="cut" Source="Resources/Cut.png" Width="16" Height="16"></Image>
<Image x:Key="paste" Source="Resources/Paste.png" Width="16" Height="16"></Image>
</StackPanel.Resources>
<Menu Height="22" Name="menu1" VerticalAlignment="Top" >
<MenuItem Header="Thực đơn _1" Name="Menu1">
<!—Ba Menu Copy, Cut, Paste được gắn biểu tượng-->
<MenuItem Header="_Copy" Command="ApplicationCommands.Copy" ToolTip="Copy văn bản"
Icon="{StaticResource copy}" />
<MenuItem Header="_Cut" Command="ApplicationCommands.Cut" ToolTip="Cắt văn bản"
Icon="{StaticResource cut}" />
<MenuItem Header="_Paste" Command="ApplicationCommands.Paste" ToolTip="Dán văn bản"
Icon="{StaticResource paste}" />
</MenuItem>
<MenuItem Header="Thực đơn _2" Name="Menu2">
<MenuItem Header="Thực đơn 21">
<MenuItem Header="Thực đơn 211" Click="MenuItem211_Click" />
<MenuItem Header="Thực đơn 212" Click="MenuItem212_Click" />
</MenuItem>
<MenuItem Header="Thực đơn 22" Click="MenuItem22_Click" />
<!--Thực đơn có trạng thái Checked và UnChecked-->
<MenuItem Header="Thực đơn 23" IsCheckable="True" Checked="Menu23_Checked"
Unchecked="Menu23_Unchecked"/>
</MenuItem>
<MenuItem Header="Thực đơn _3" Click="MenuItem3_Click" Name="Menu3" />
</Menu>
<TextBox Name="textBox1" TextWrapping="Wrap" Margin="2">
Chú cào cào nhỏ, ngồi trong đám cỏ.
</TextBox>
</StackPanel>
```

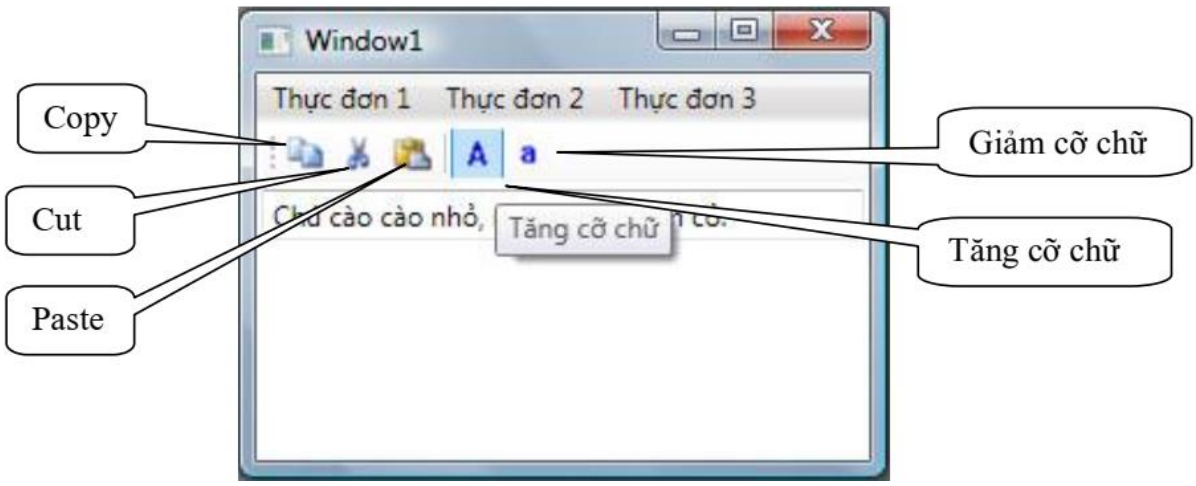
Như vậy, chúng ta đã xây dựng thành công thanh menu với các biểu tượng đẹp mắt hay menu với trạng thái Checked/UnChecked cũng như biết cách gắn các hàm xử lý sự kiện cho các menu. Phần tiếp theo ta sẽ tìm hiểu về thanh công cụ.

2. Xây dựng và sử dụng thanh công cụ (Toolbar)

Thanh công cụ (Toolbar) là thanh chứa các chức năng dưới dạng các dãy hình ảnh biểu tượng, mỗi biểu tượng gắn với một mục chức năng cụ thể. Thông thường các Toolbar chứa những chức năng thiết yếu mà người dùng hay quan tâm nhất, bởi vì thanhToolbar có ưu điểm là dễ dàng thao tác. Một cửa số có thể có một hoặc nhiều thanh Toolbar. Trong phần này ta tìm hiểu phương pháp xây dựng thanh Toolbar với các nút chức năng thông thường và các nút chức năng có trạng thái (Checked/UnChecked).

2.1. Nút công cụ thông thường

Chúng ta bắt đầu tìm hiểu các bước xây dựng thanh Toolbar với các nút bấm đơn giản như ví dụ minh họa ở hình 10.8. Thanh công cụ bao gồm năm nút: Copy , Cut, Paste thực hiện các chức năng có sẵn của hệ thống, Nút *A* và *a* gắn với hàm xử lý sự kiện tự xây dựng, làm nhiệm vụ tăng/giảm cỡ chữ của Textbox bên dưới.



Hình 10.8. Ví dụ minh họa Toolbar

Thanh công cụ được xây dựng bằng đoạn mã XAML sau:

```
<StackPanel>
<ToolBar Height="26" Name="toolBar1" Width="280" HorizontalAlignment="Left" >
<Button Height="23" Name="button1" Width="23"
Command="ApplicationCommands.Copy" ToolTip="Copy văn bản">
<Image Source="Resources/Copy.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button2" Width="23" Command="ApplicationCommands.Cut"
ToolTip="Cắt văn bản">
<Image Source="Resources/Cut.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button3" Width="23"
Command="ApplicationCommands.Paste" ToolTip="Dán văn bản">
<Image Source="Resources/Paste.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Separator/>
<Button Height="23" Name="button4" ToolTip="Tăng cỡ chữ" Width="23"
Click="IncreaseFont_Click">
<Image Source="Resources/inc.ico" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button5" ToolTip="Giảm cỡ chữ" Width="23"
Click="DecreaseFont_Click">
<Image Source="Resources/Dec.ico" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
</ToolBar>
</StackPanel>
```

Mã XAML tạo thanh công cụ được được bắt đầu bằng thẻ <ToolBar> và kết thúc bằng thẻ đóng </ToolBar>.

Các nút lệnh (Button) của thanh công cụ được tạo bởi thẻ <Button> và kết thúc bằng thẻ đóng </Button> .

Name=" button1": Tên của Button, cần thiết cho mã trình C# có thể can thiệp vào Button. ToolTip="Copy văn bản ": Lời chú thích cho Button khi di chuột qua.

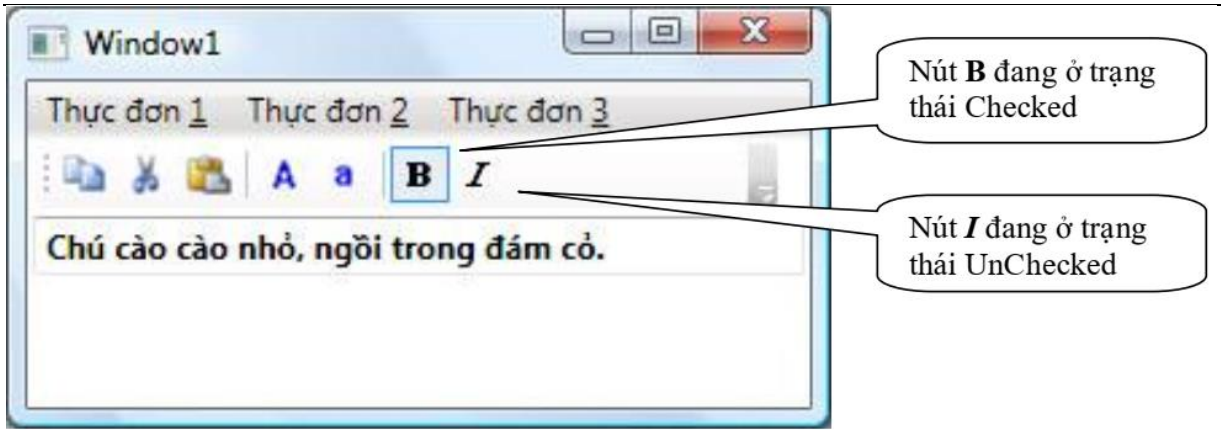
Có hai cơ chế thực thi lệnh khi chọn nút lệnh trong Toolbar. Nếu muốn gắn nút lệnh với các lệnh có sẵn của hệ thống như: Copy, Cut, Paste,.. thì ta sử dụng thuộc tính Command của Button. Ví dụ, lệnh <Button Height="23" Name="button1" Width="23" Command="ApplicationCommands.Copy" oolTip="Copy văn bản"> làm cho nút lệnh Copy này sẽthự c hiện công việc copy dòng văn bản đang được chọn trong cửa sổ vào bộ nhớ đệm. Chú ý, các lệnh của hệ thống bắt đầu bằng ApplicationCommands.

Nếu muốn gắn nút lệnh với các hàm xử lý sự kiện tự định nghĩa thì sử dụng thuộc tính Click của Button. Ví dụ, <Button Height="23" Name="button4" ToolTip="Tăng cỡ chữ" Width="23" Click="IncreaseFont_Click"> để yêu cầu khi chọn "button4" thì sẽ gọi hàm IncreaseFont_Click. Giữa cặp thẻ <Button> và </Button> là thẻ <Image Source="Resources/Copy.png" Width="16" Height="16" HorizontalAlignment="Left" /> để định nghĩa hình ảnh biểu tượng của nút bấm. Thẻ <Separator/> dùng để tạo ra vạch phân cách giữa cách nút bấm. Mã nguồn minh họa của hai hàm xử lý sự kiện nhấn nút button4và button5như sau:

```
public partial class Window1 : Window
{
public Window1()
{
InitializeComponent();
}
//.....................................................
//Hàm xử lý sự kiện khi button4 được nhấn
private void IncreaseFont_Click(object sender, RoutedEventArgs e)
{
if (textBox1.FontSize < 18)
{
textBox1.FontSize += 2;//Tăng cỡ font chữ của textBox1 lên 2 đơn vị
}
}
//Hàm xử lý sự kiện khi button5 được nhấn
private void DecreaseFont_Click(object sender, RoutedEventArgs e)
{
if (textBox1.FontSize > 10)
{
textBox1.FontSize -= 2;//Giảm cỡ font chữ của textBox1 lên 2 đơn vị
}
}
}
```

2.2. Nút công cụ có trạng thái

Ngoài các nút bấm thông thường, thanh công cụ còn cho phép tạo ra các nút bấm có trạng thái, khi ở trang thái được chọn (Checked) thì sẽ có màu nền khác và có đường viền để người dùng có thể nhận biết được trạng thái của nút đó (xem minh họa hình 10.9).



Hình 10.9. Ví dụ minh họa Toolbar với trạng thái Checked và UnCheked

Thanh công cụ với các nút có trạng thái được xây dựng bằng đoạn mã XAML sau:

```
<StackPanel>
<ToolBar Height="26" Name="toolBar1" Width="280" HorizontalAlignment="Left" >
<Button Height="23" Name="button1" Width="23"
Command="ApplicationCommands.Copy" ToolTip="Copy văn bản">
<Image Source="Resources/Copy.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button2" Width="23" Command="ApplicationCommands.Cut"
ToolTip="Cắt văn bản">
<Image Source="Resources/Cut.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button3" Width="23"
Command="ApplicationCommands.Paste" ToolTip="Dán văn bản">
<Image Source="Resources/Paste.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Separator/>
<Button Height="23" Name="button4" ToolTip="Tăng cỡ chữ" Width="23"
Click="IncreaseFont_Click">
<Image Source="Resources/inc.ico" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button5" ToolTip="Giảm cỡ chữ" Width="23"
Click="DecreaseFont_Click">
<Image Source="Resources/Dec.ico" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Separator/>
<!--Nút bấm với trạng thái Checked và UnChecked-->
<CheckBox Name="check1" ToolTip="Chữ đậm" Checked="Bold_Checked"
Unchecked="Bold_Unchecked">
<Image Source="Resources/bold.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</CheckBox>
<CheckBox Name="check2" ToolTip="Chữ nghiêng" Checked="Italic_Checked"
Unchecked="Italic_Unchecked">
<Image Source="Resources/Italic.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</CheckBox>
</ToolBar>
</StackPanel>
```

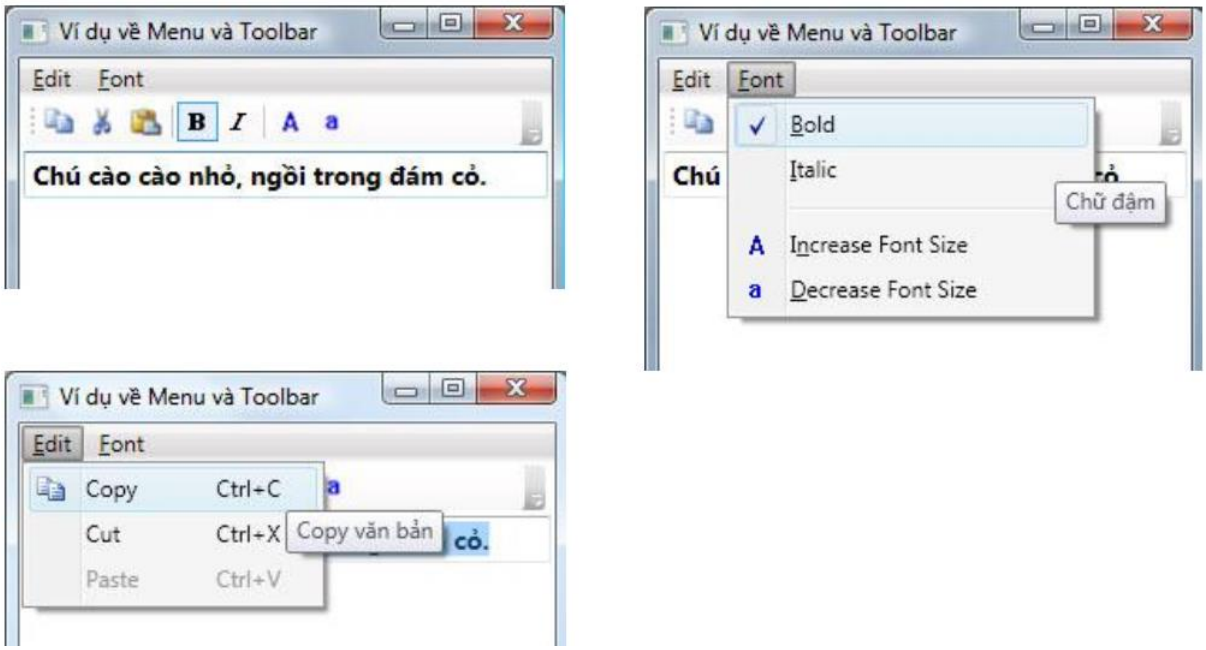
Khác với các nút lệnh thông thường được tạo bởi thẻ <Button> và kết thúc bằng thẻ đóng </Button>, các nút lệnh có trạng thái được tạo nên bởi thẻ <CheckBox Name="check1" ToolTip="Chữ đậm" Checked="Bold_Checked" Unchecked="Bold_Unchecked"> và kết thúc bằng thẻ đóng </CheckBox>. Nút lệnh có trạng thái phát sinh hai sự kiện Checked và Unchecked,tương ứng với trạng thái của nút là được chọn hay bỏ chọn khi người dùng nhấn nút. Trong ví dụ trên, khi nút check1 được chọn thì hàm Bold_Checked được gọi và nút check1bỏ chọn thì hàm Bold_Unchecked được gọi. Hai hàm này do ta tự xây dựng với mã lệnh như sau. Mã nguồn minh họa của hai hàm xử lý sự kiện nhấn nút check1:

```
public partial class Window1 : Window
{
public Window1()
{
InitializeComponent();
}
//.....................................................
private void Bold_Checked(object sender, RoutedEventArgs e)
{
//Thiết lập thuộc tính chữ đậm cho textBox1
textBox1.FontWeight = FontWeights.Bold;
}
private void Bold_Unchecked(object sender, RoutedEventArgs e)
{
//Thiết lập thuộc tính chữ thường cho textBox1
textBox1.FontWeight = FontWeights.Normal;
}
private void Italic_Checked(object sender, RoutedEventArgs e)
{
//Thiết lập thuộc tính chữ nghiêng cho textBox1
textBox1.FontStyle = FontStyles.Italic;
}
private void Italic_Unchecked(object sender, RoutedEventArgs e)
{
//Thiết lập thuộc tính chữ thẳng đứng cho textBox1
textBox1.FontStyle = FontStyles.Normal;
}
}
```

Ngoài các nút bấm có trạng thái kiểu này, thanh công cụ còn cho phép tạo ra các nút bấm có trạng thái loại trừ nhau bằng cách sử dụng thẻ <RadioButton> để tạo ra nút bấm. Như vậy, chúng ta đã hoàn tất bài học xây dựng và quản lý thanh Thực đơn và thanh công cụ. Tiếp theo là ví dụ tổng hợp lại các kiến thức trên.

3. Ví dụ tổng hợp về xây dựng Menu và Toolbar

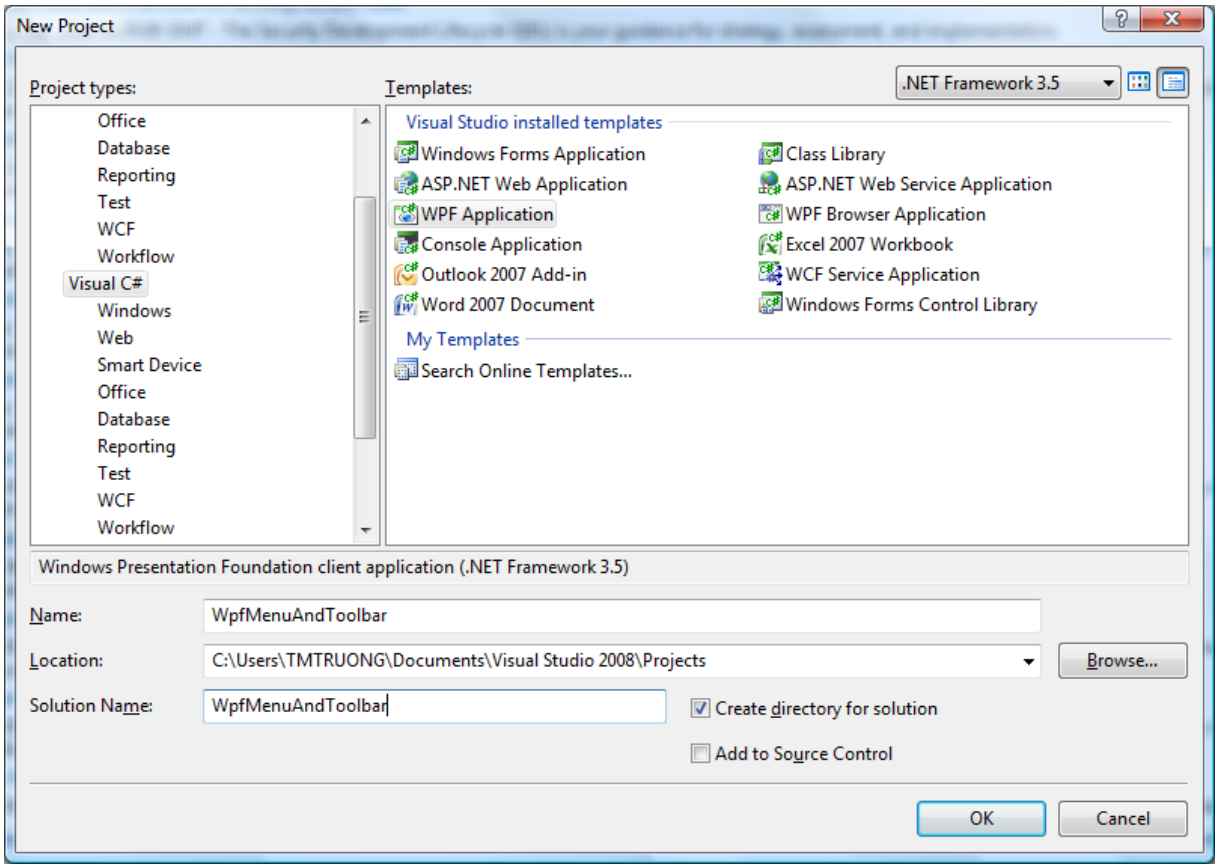
Xây dựng cửa sổ bao gồm một thanh thực đơn, một thanh công cụ và một hộp soạn thảo như minh họa ở hình 10.10



Hình 1.10. Ví dụ minh họa tổng hợp về Menu và Toolbar

Các bước thực hiện như sau:

3.1. Tạo ứng dụng WPF



Ở đây, ta chú ý chọn .Net Framework 3.5.

3.2. Mã XAML của giao diện

Mở file Window1.xaml tương ứng với file code Window1.xaml.cs.

```
<Window x:Class="WpfMenuAndToolbar.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Ví dụ về Menu và Toolbar" Height="300" Width="300"> <StackPanel >
<!--Khai báo Tài nguyên cần sử dụng-->
<StackPanel.Resources>
<Image x:Key="inc" Source="Resources/inc.ico" Width="16"
Height="16"></Image>
<Image x:Key="dec" Source="Resources/dec.ico" Width="16"
Height="16"></Image>
<Image x:Key="copy" Source="Resources/Copy.png" Width="16"
Height="16"></Image>
</StackPanel.Resources>
<!--Khai báo thanh thực đơn-->
<Menu Name="Menu_main">
<MenuItem Header="_Edit" >
<MenuItem Command="ApplicationCommands.Copy" ToolTip="Copy văn bản"
Icon="{StaticResource copy}"/>
<MenuItem Command="ApplicationCommands.Cut" ToolTip="Cắt văn bản"/>
<MenuItem Command="ApplicationCommands.Paste" ToolTip="Dán văn
bản"/>
</MenuItem>
<MenuItem Header="_Font" >
<MenuItem Header="_Bold" Name="Menu21" ToolTip="Chữ đậm"
IsCheckable="True" Checked="Bold_Checked" Unchecked="Bold_Unchecked"/>
<MenuItem Header="_Italic" Name="Menu22" ToolTip="Chữ nghiêng"
IsCheckable="True" Checked="Italic_Checked" Unchecked="Italic_Unchecked"/>
<Separator/>
<MenuItem Header="I_ncrease Font Size" ToolTip="Tăng cỡ chữ" Icon="{StaticResource
inc}" Click="IncreaseFont_Click"/>
<MenuItem Header="_Decrease Font Size" ToolTip="Giảm cỡ chữ"
Icon="{StaticResource dec}" Click="DecreaseFont_Click"/>
</MenuItem>
</Menu>
<!--Khai báo thanh công cụ-->
<ToolBar Height="26" Name="toolBar1" Width="280" HorizontalAlignment="Left"
>
<Button Height="23" Name="button1" Width="23"
Command="ApplicationCommands.Copy" ToolTip="Copy văn bản">
<Image Source="Resources/Copy.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button2" Width="23"
Command="ApplicationCommands.Cut" ToolTip="Cắt văn bản">
<Image Source="Resources/Cut.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button3" Width="23"
Command="ApplicationCommands.Paste" ToolTip="Dán văn bản">
<Image Source="Resources/Paste.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Separator/>
<CheckBox Name="check1" ToolTip="Chữ đậm" Checked="Bold_Checked"
Unchecked="Bold_Unchecked">
<Image Source="Resources/bold.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</CheckBox>
<CheckBox Name="check2" ToolTip="Chữ nghiêng" Checked="Italic_Checked"
```

```
Unchecked="Italic_Unchecked">
<Image Source="Resources/Italic.png" Width="16" Height="16"
HorizontalAlignment="Left" />
</CheckBox>
<Separator/>
<Button Height="23" Name="button6" ToolTip="Tăng cỡ chữ" Width="23"
Click="IncreaseFont_Click">
<Image Source="Resources/inc.ico" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
<Button Height="23" Name="button7" ToolTip="Giảm cỡ chữ" Width="23"
Click="DecreaseFont_Click">
<Image Source="Resources/Dec.ico" Width="16" Height="16"
HorizontalAlignment="Left" />
</Button>
</ToolBar>
<!--Khai báo hộp soạn thảo-->
<TextBox Name="textBox1" TextWrapping="Wrap" Margin="2">
Chú cào cào nhỏ, ngồi trong đám cỏ.
</TextBox>
</StackPanel>
</Window>
```

3.3. Mã lệnh C# xử lý các sự kiện

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
namespace WpfMenuAndToolbar
{
/// <summary>
/// Interaction logic for Window1.xaml
/// </summary>

public partial class Window1 : System.Windows.Window
{

public Window1()
{
     InitializeComponent();
}
private void Bold_Checked(object sender, RoutedEventArgs e)
{
textBox1.FontWeight = FontWeights.Bold;
check1.IsChecked = true;
Menu21.IsChecked = true;
}
private void Bold_Unchecked(object sender, RoutedEventArgs e)
{
textBox1.FontWeight = FontWeights.Normal;
check1.IsChecked = false;
Menu21.IsChecked = false;
}
```

```
private void Italic_Checked(object sender, RoutedEventArgs e)
{
textBox1.FontStyle = FontStyles.Italic;
check2.IsChecked = true;
Menu22.IsChecked = true;
}
private void Italic_Unchecked(object sender, RoutedEventArgs e)
{
textBox1.FontStyle = FontStyles.Normal;
check2.IsChecked = false;
Menu22.IsChecked = false;
}
private void IncreaseFont_Click(object sender, RoutedEventArgs e)
{
if (textBox1.FontSize < 18)
{
textBox1.FontSize += 2;
}
}
private void DecreaseFont_Click(object sender, RoutedEventArgs e)
{
if (textBox1.FontSize > 10)
{
textBox1.FontSize -= 2;
}
}
}
}
```

F5 để chạy ứng dụng. Ta sẽ thu được kết quả tương tự như yêu cầu.

4. 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.

4.1. 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 đơngiản 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 nguyênbạ n 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.

4.1.1. Các thành phần thuộc tính trong Style

4.1.1.1. 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>
```

4.1.1.2. TargetType2

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>
```

4.1.1.3 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>
```

4.1.1.4. 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:

DataTrigger3

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 enCongViec=”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>
```

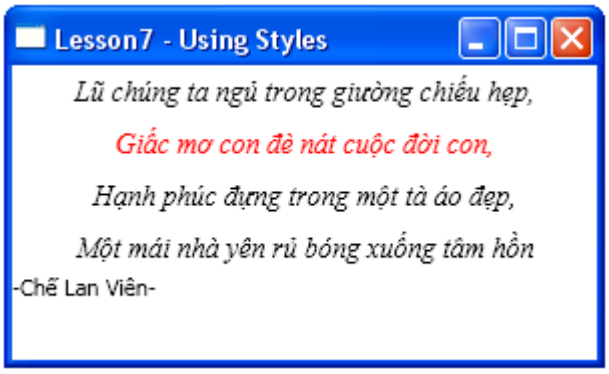
4.1.2. Một ví dụ đầy đủ về sử dụng Style

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: 5



Hình 10.11 - 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.

4.2. 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.

4.2.1 ControlTemplate

4.2.1.1 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 10.12 – 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.

4.2.1.2 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"/>
```

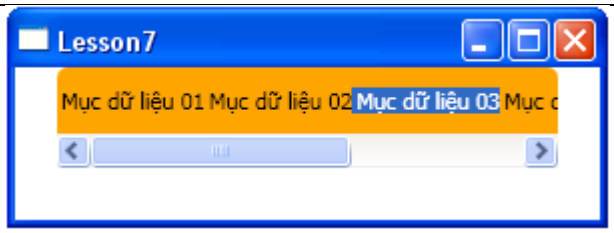
```
</ScrollViewer>
</Border>
</ControlTemplate>
</Setter.Value>
</Setter>
</Style>
</StackPanel.Resources>
<!--Kết thúc khai báo tài nguyên-->
<!--Khai báo ListBox-->
<ListBox Width="250" Height="50">
<ListBoxItem>Mục dữ liệu 01</ListBoxItem>
<ListBoxItem>Mục dữ liệu 02</ListBoxItem>
<ListBoxItem>Mục 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 10.13.



Hình 10.13 – Một ví dụ về xây dựng và sử dụng một ControlTemplate cho ListBox

4.2.2. DataTemplate

4.2.2.1. 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…

4.2.2.2. 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 Name9
{
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 báotrong 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>
```

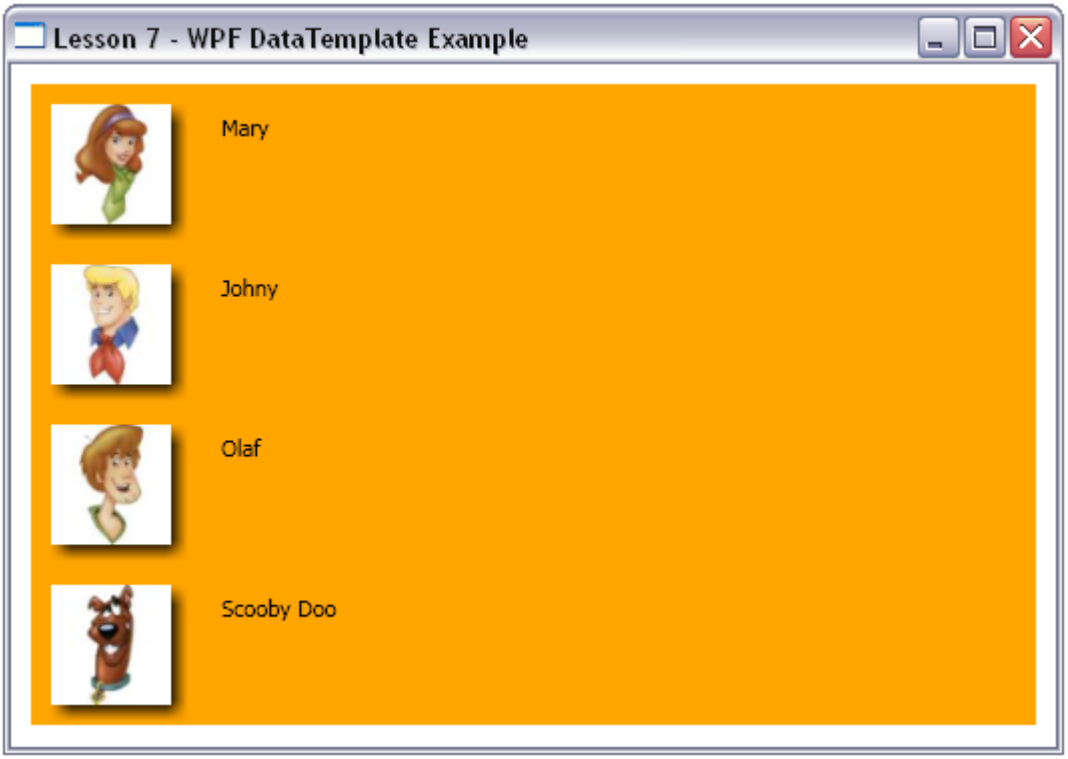
DataTemplate được định nghĩa ở đây chỉ được dùng với kiểu dữ liệu thuộc lớp Person (DataType="{x:Type local:Person}"). Cách thức hiển thị tên nhân viên được khai báo trong một phần tử TextBlock mà thuộc tính Text được liên kết với thuộc tính Name của mỗi đối tượng Person (Text="{Binding Name}"). Trong 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:12



Hình 10.14 – Hiển thị danh sách nhân viên sử dụng DataTemplate

Tài liệu Tham khảo

[1].Windows Presentation Foundation, URL: http://msdn.microsoft.com/en-us/library/ms754130.aspx.
[2]. WPF - XAML Introduction, URL: http://homepage.ntlworld.com/herring1/xamlintro.html.
[3]. Menu Icons in WPF, URL: http://anuraj.wordpress.com/2008/06/23/menu-icons-in-wpf/.
[4]. WPF Sample Series, URL: http://karlshifflett.wordpress.com/2008/01/23/wpf-sample-series-stretchtoolbar-width-of-window/.